Άσκηση 1

typedef struct {

- unsigned int id;
- int val;
- pthread_mutex_t lock;

} mysem_t;

Η πληροφορία για κάθε σηματοφόρο που έχει δημιουργηθεί αποθηκεύεται σε ένα δυναμικά δεσμευμένο πίνακα δεικτών σε mysem_t.

Χρήση mutex γίνεται για να προστατευτεί η μεταβλητή val κάθε σηματοφόρου υπό ταυτόχρονη εκτέλεση.

Mysem_down: στην περίπτωση που θα μπλοκάρει το νήμα που την κάλεσε τότε η semop() καλείτε αφού ξεκλειδωθεί το mutex. Επειδή αλλιώς δεν μπορούσε κάποια up να μπει στο ΚΤ της ώστε να ξυπνήσει. Θα είχαμε αδιέξοδο για κάθε νήμα που καλεί up/down σε αυτό το σηματοφόρο.

mysem_down()

```
set system call flags
lock mutex
decrement semaphore value
If semaphore value is 0 {
        Sytem V down;
        Unlock mutex;
        RETURN 1;
}
unlock mutex;
system V down();
RETURN 1;
```

mysem_up()

```
set system call flags;
lock mutex;
If semaphore value is 1 {
        Unlock mutex;
        RETURN 0;
}
increment semaphore value;
unlock mutex;
system V up;
RETURN 0;
```

Δεν αποτελεί race condition το γεγονός ότι down καλείται μετά το unlocking του mutex.

Άσκηση 2

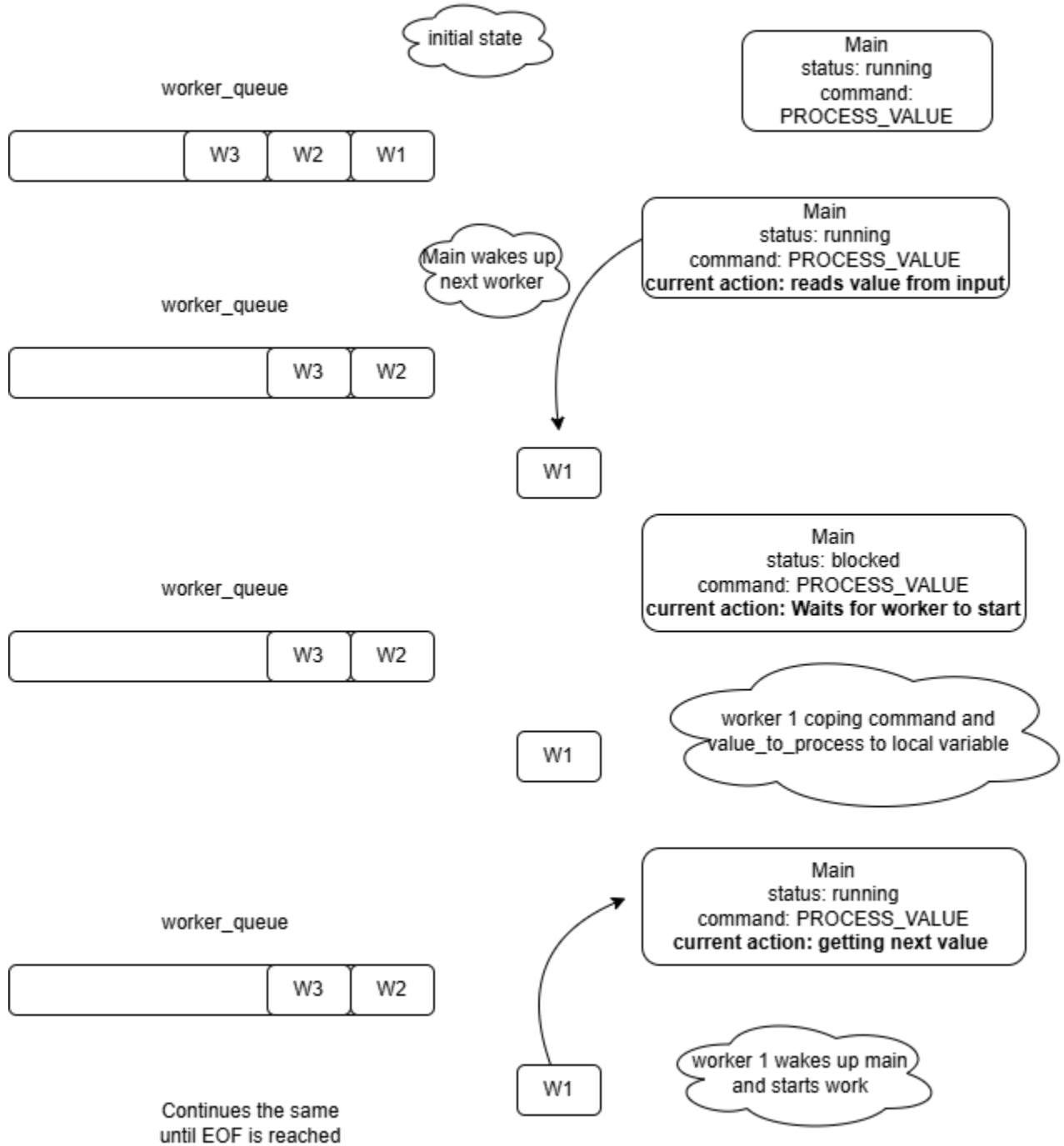
Main thread

```
Create N workers
Initialize main_sleep semaphore to 0
Initialize worker_queue semaphore to 0
Set command to PROCESS_VALUE
While (end of input not reached) {
        read next value
        store value to process
        /* tell next worker to start */
        up(worker_queue);
        /*Wait worker to start working*/
        down(main_sleep);
    }
    Set command to TERMINATE
    For every worker i {
        /* tell next worker to start */
        Up(worker_queue);
        /*wait for worker to indicate termination*/
        down(main_sleep);
    }
    RETURN 0;
```

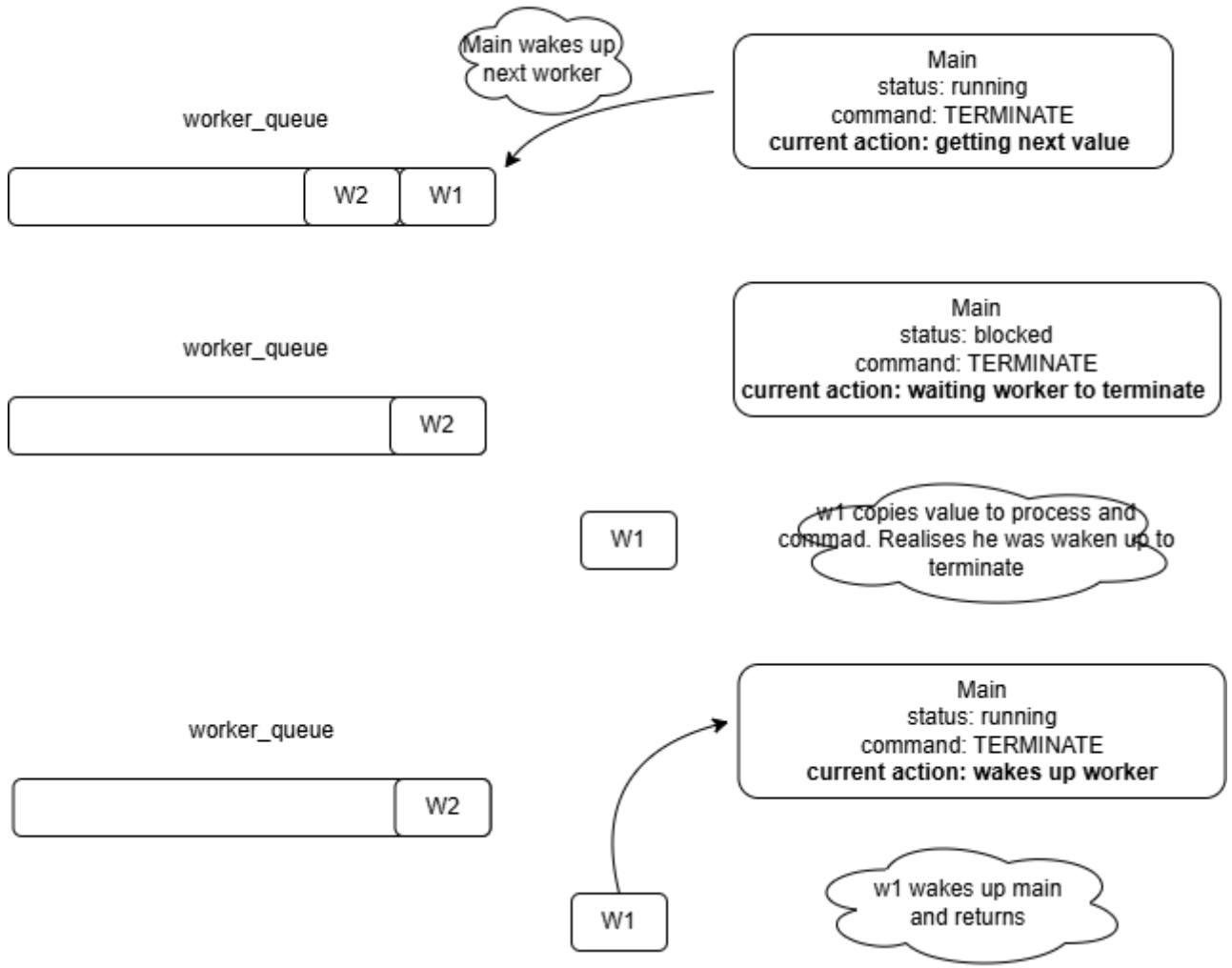
Worker thread

```
WHILE (1) {
        /* wait main */
        down(worker_queue);
        copy value to process in a local variable;
        copy command to a local variable;
        IF main's command is TERMINATE {
            /*tell main that you are terminating*/
            Up(main_sleep);
            BREAK;
        }
        /* tell main you are starting work */
        up(main_sleep);
        is_prime(value);
}
RETURN NULL;
```

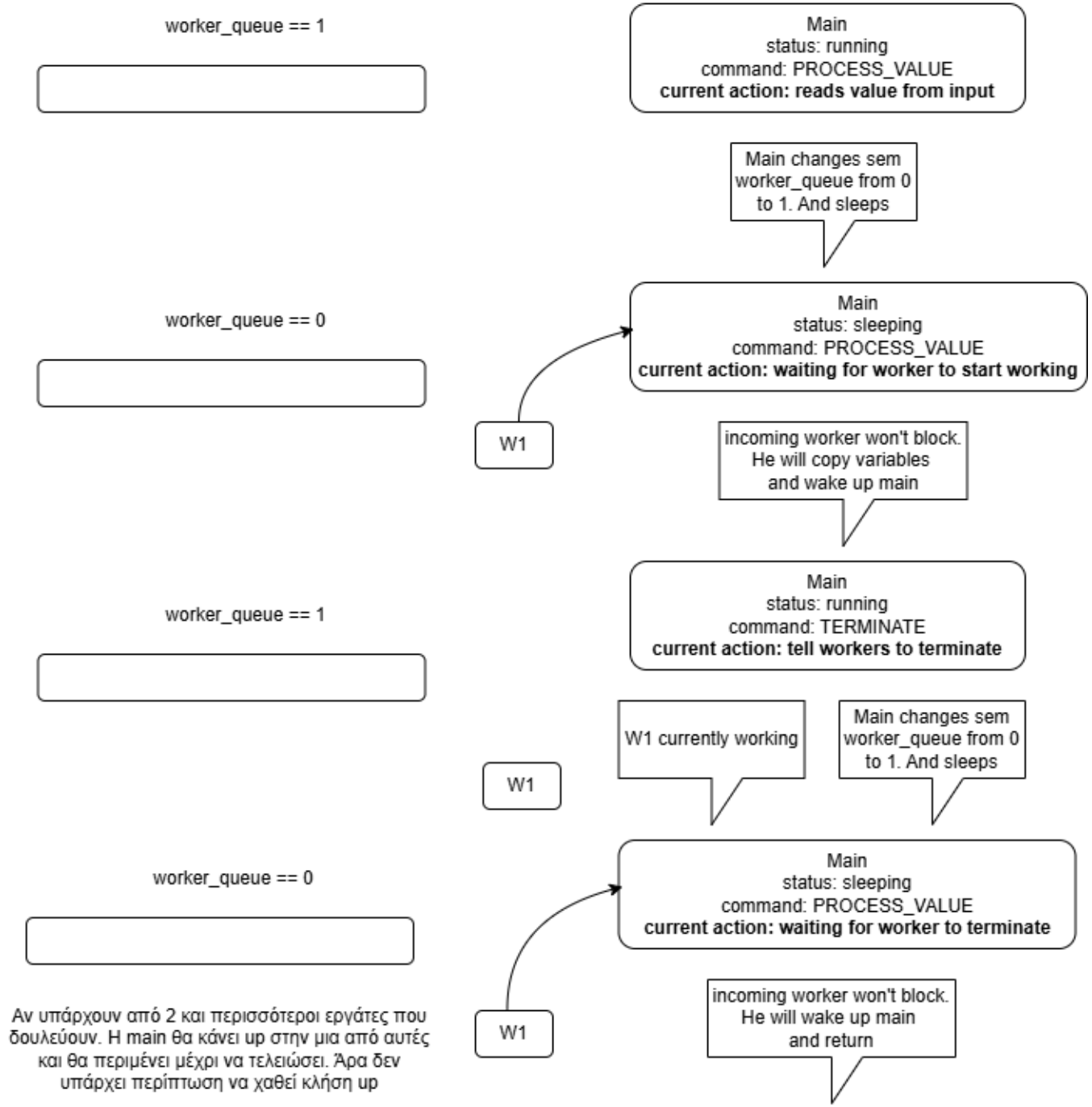
Set work to worker concept



Termination concept



Exceptions



Αν υπάρχουν από 2 και περισσότεροι εργάτες που δουλεύουν. Η main θα κάνει up στην μια από αυτές και θα περιμένει μέχρι να τελειώσει. Άρα δεν υπάρχει περίπτωση να χαθεί κλήση up

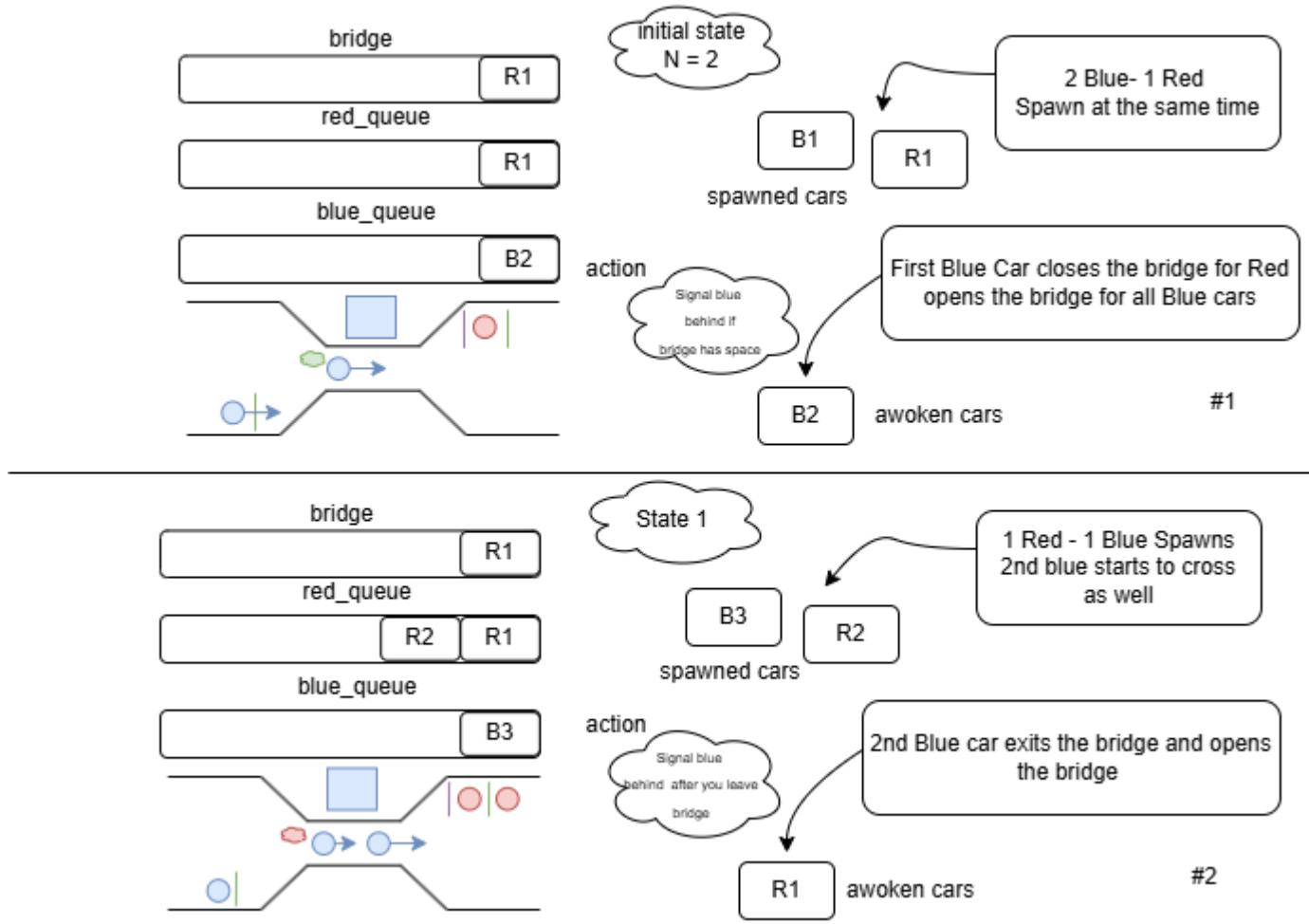
Άσκηση 3

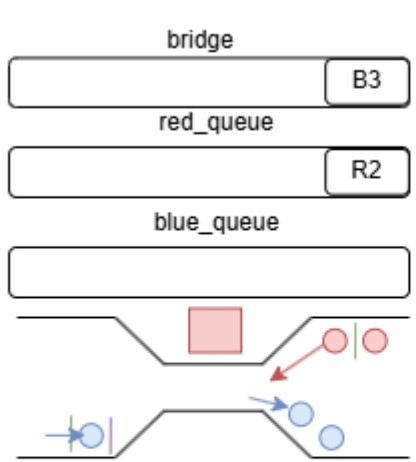
```
down(team_q);
down(team_lock);
if (first) { down(bridge); }
bridge_cars++;
total_cars++;

if (room for more) { up(team_q); }

up(team_lock);

cross();
down(team_lock);
bridge_cars--;
if (last) {
        if (total_cars >= N) { up(team_q); }
        total_cars = 0;
        up(bridge);
}
up(team_lock);
```


bridge
R1
red_queue
R1
blue_queue
B2
initial state
N = 2
2 Blue- 1 Red
Spawn at the same time
B1
R1
spawned cars
action
Signal blue
behind if
bridge has space
First Blue Car closes the bridge for Red
opens the bridge for all Blue cars
B2
awoken cars
#1
bridge
R1
red_queue
R2
R1
blue_queue
B3
State 1
1 Red - 1 Blue Spawns
2nd blue starts to cross
as well
B3
R2
spawned cars
action
Signal blue
behind after you leave
bridge
2nd Blue car exits the bridge and opens
the bridge
R1
awoken cars
#2



State 2

No new cars

Signal red behind if bridge has space

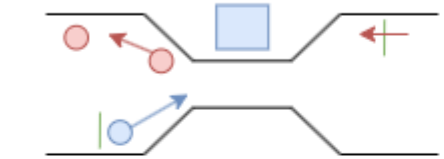
First Red Car closes the bridge for Blue opens the bridge for all Red cars

R2 awoken cars

#3

State 3

Repeat just like with two blues ...

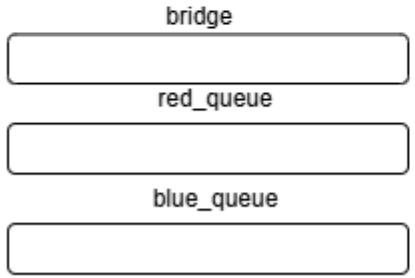


State 4

No new cars

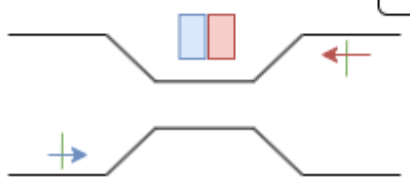
Signal next blue behind if bridge has space

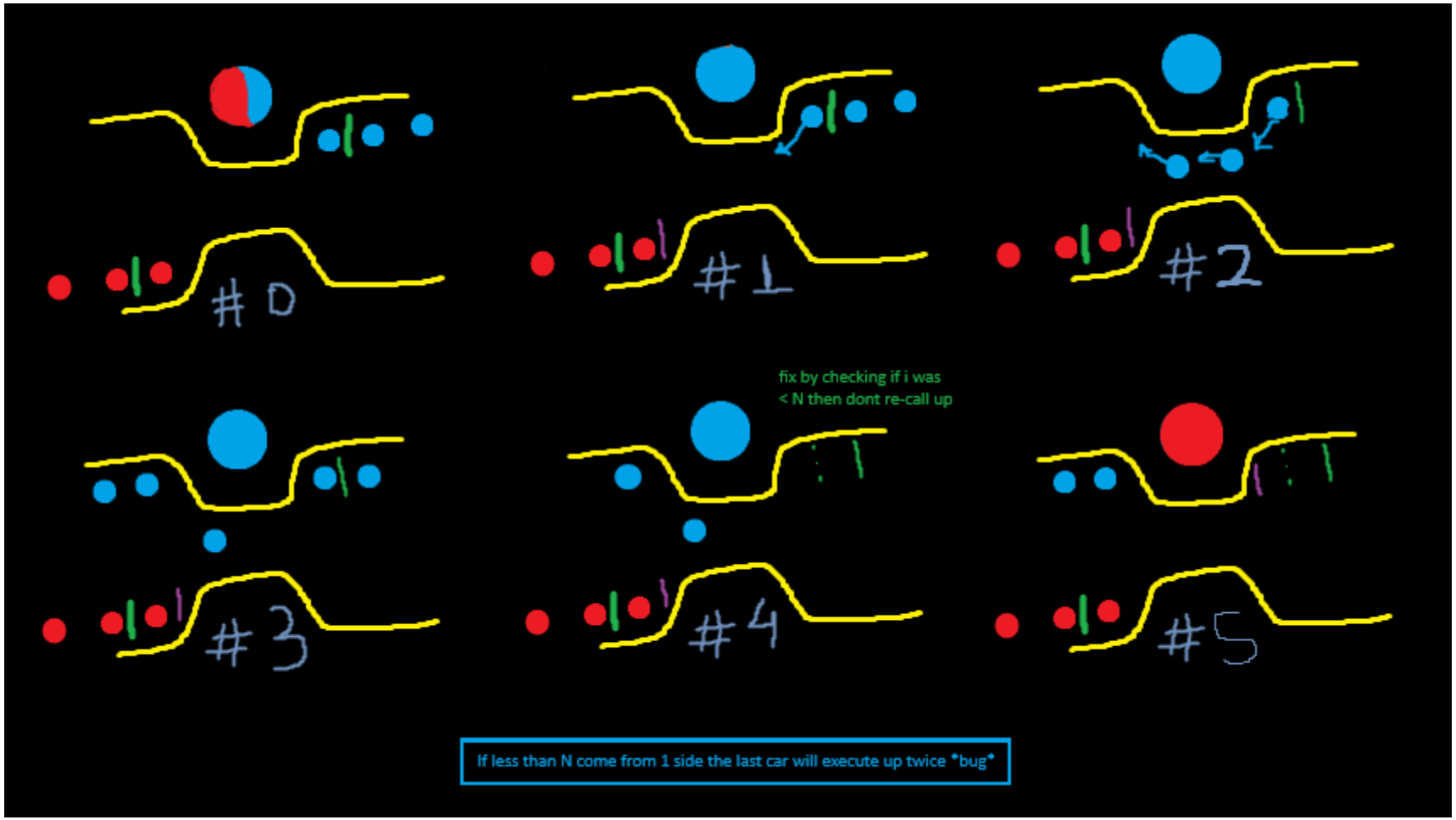
First Blue Car closes the bridge for Red opens the bridge for all Blue cars

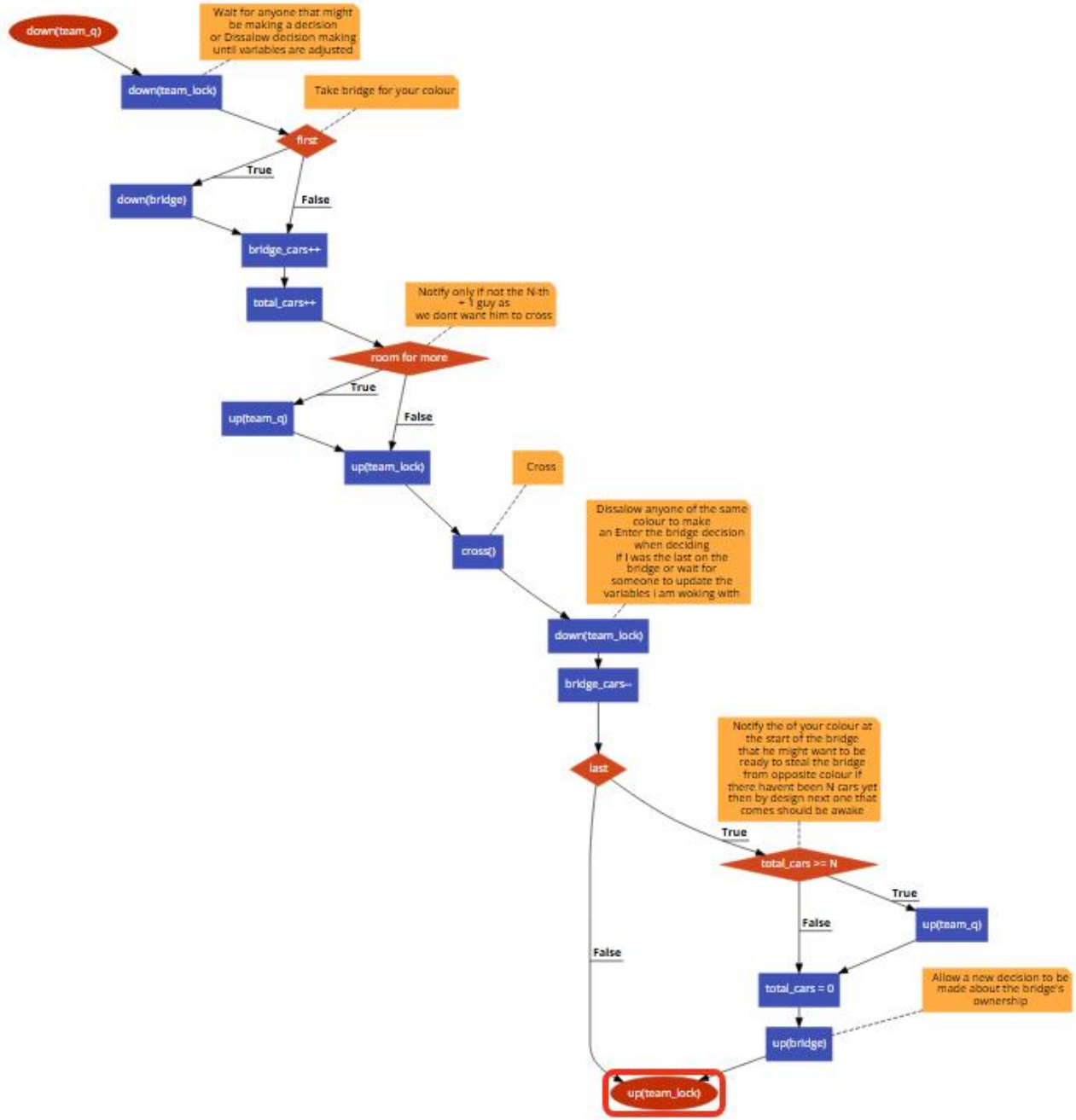


R3... awoken cars

#4




#0
#1
#2
fix by checking if i was
< N then dont re-call up
#3
#4
#5
If less than N come from 1 side the last car will execute up twice *bug*


down(team_q)
Wait for anyone that might be making a decision or Dissalow decision making until variables are adjusted
down(team_lock)
Take bridge for your colour
first
True
False
down(bridge)
bridge_cars++
total_cars++
Notify only if not the N-th - 1 guy as we dont want him to cross
room for more
True
False
up(team_q)
up(team_lock)
Cross
cross()
Dissalow anyone of the same colour to make an Enter the bridge decision when deciding if I was the last on the bridge or wait for someone to update the variables I am woking with
down(team_lock)
bridge_cars--
last
Notify the of your colour at the start of the bridge that he might want to be ready to steal the bridge from opposite colour if there havent been N cars yet then by design next one that comes should be awake
True
False
total_cars >= N
True
False
up(team_q)
total_cars = 0
Allow a new decision to be made about the bridge's ownership
up(bridge)
up(team_lock)

Άσκηση 4

```
typedef struct {
    pthread_t id;
    int max_passengers;
    bool *park_closed;
    int *passengers_waiting;
    mysem_t *mtx;
    mysem_t *train_wait;
    mysem_t *wait_passenger_action;
    mysem_t *passenger_queue;
    mysem_t *ride_queue;
} thread_info_t;
```

Train thread

```
WHILE (1) {
    /* wait until max passengers arrive */
    down(train_wait);
    IF is closed
      BREAK;

    /* wake up passengers and wait for each to board */
    FOR every passenger from i to max_passengers {
      up(passenger_queue);
      down(wait_passenger_action);
    }

    // start ride
    sleep(10);

    down(mtx);
    passengers_waiting -= max_passengers;
    IF passengers waiting are enough for a ride {
      // go for a ride
      up(train_wait);
    }
    up(mtx);
  }
```

Passenger thread

```
down(mtx)
  passengers_waiting++;

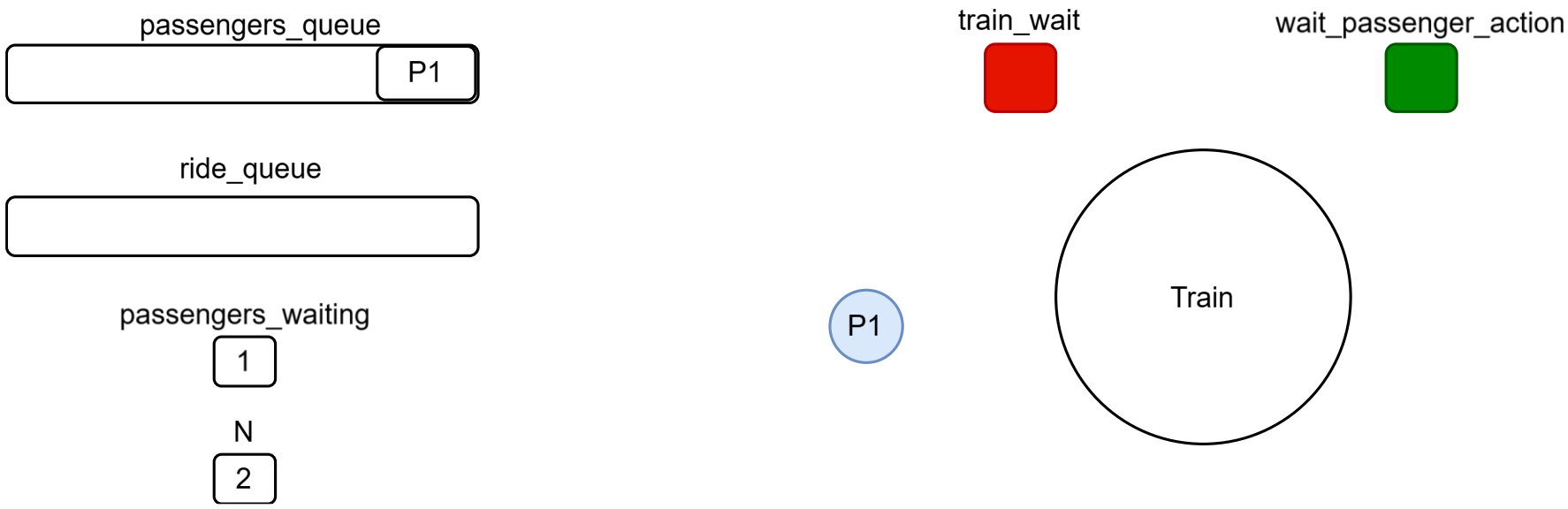
  IF you are the N-th passenger {
    up(train_wait)
  }
  up(mtx);

  /* wait for train to wake you up */
  down(passenger_queue);
  up(wait_passenger_action);

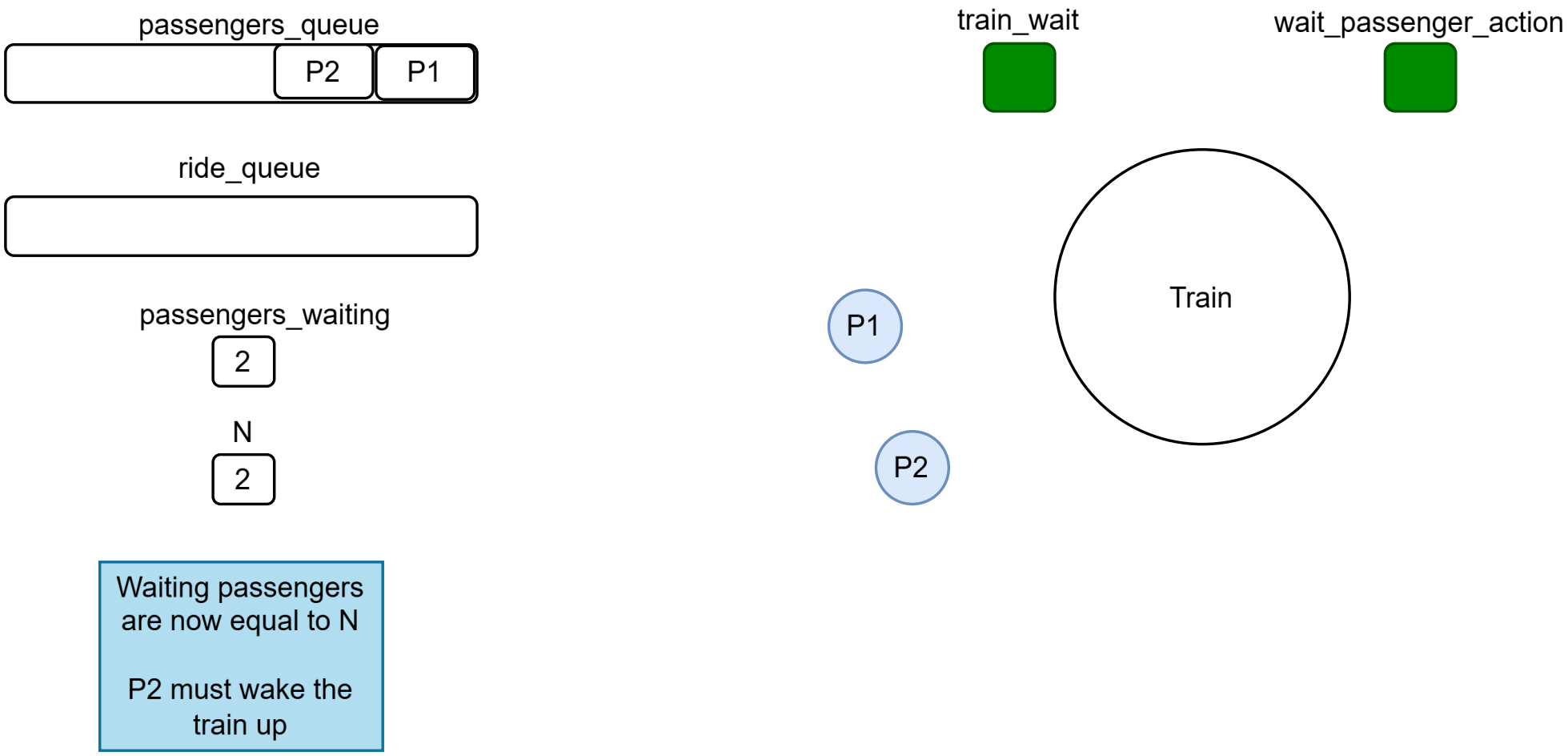
  /* go for a ride */
  down(ride_queue);

  up(wait_passenger_action);
```

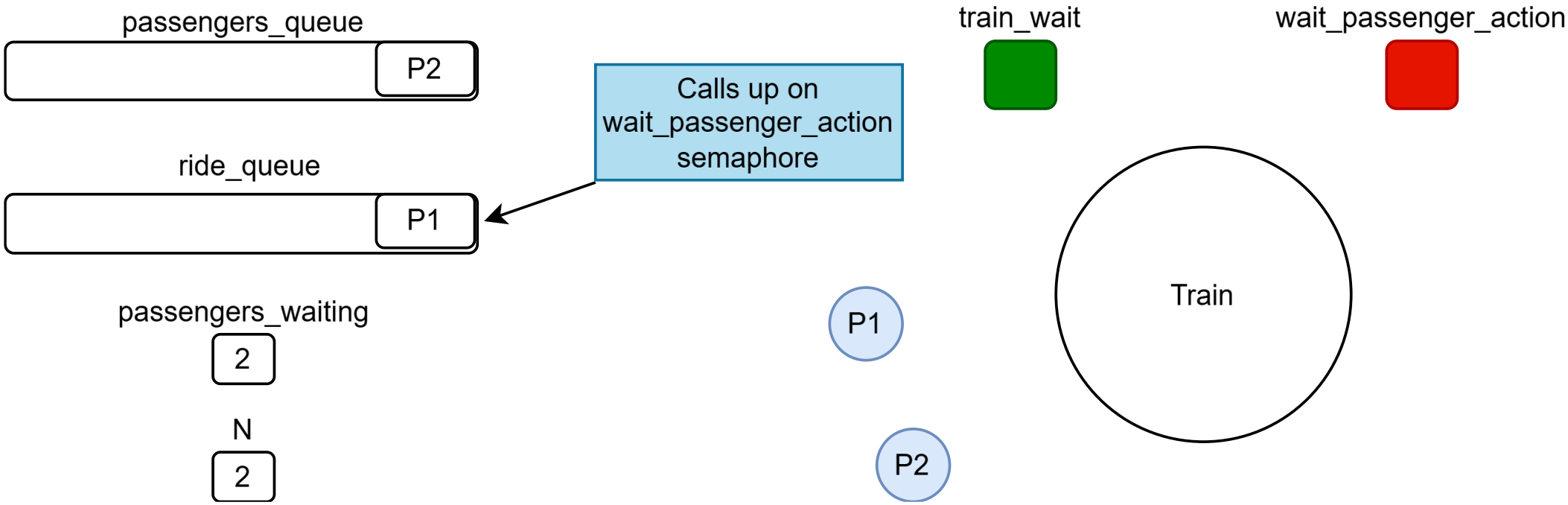
New passenger



New passenger wakes train up



Train wakes up passenger and waits for acknowledgement



Train starts the ride



Train ends the ride

