Redes de Petri

Aplicación a problemas clásicos de concurrencia

CAROLINA PISTILLO

ELIANA GAMARRA

TOMÁS LÓPEZ HIDALGO

Filósofos

Fumadores

Lectores y escritores

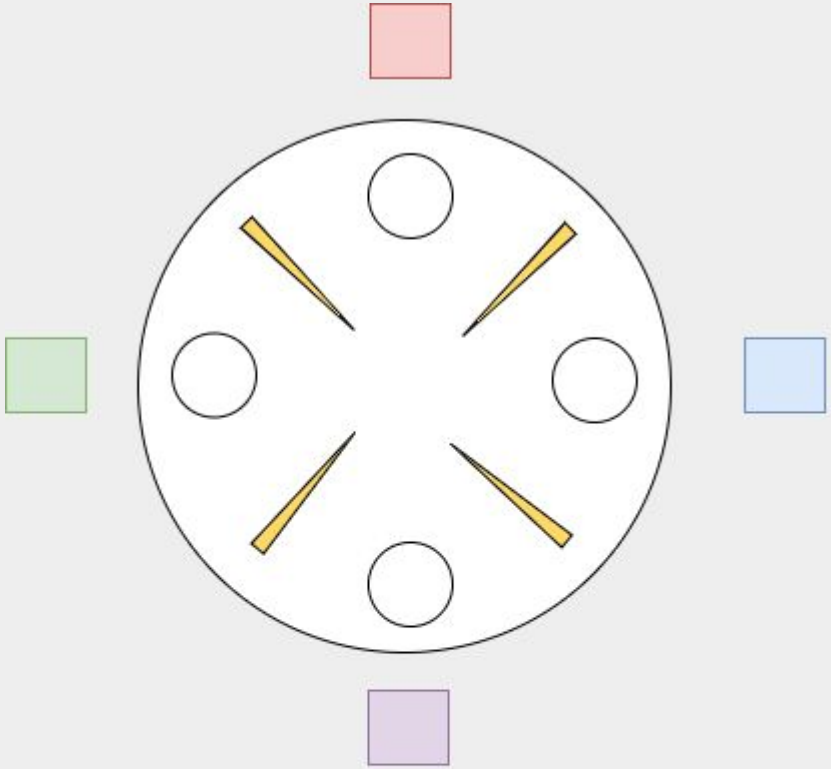
Problema de los filósofos

Cinco filósofos se sientan alrededor de una mesa y pasan su vida cenando y pensando.

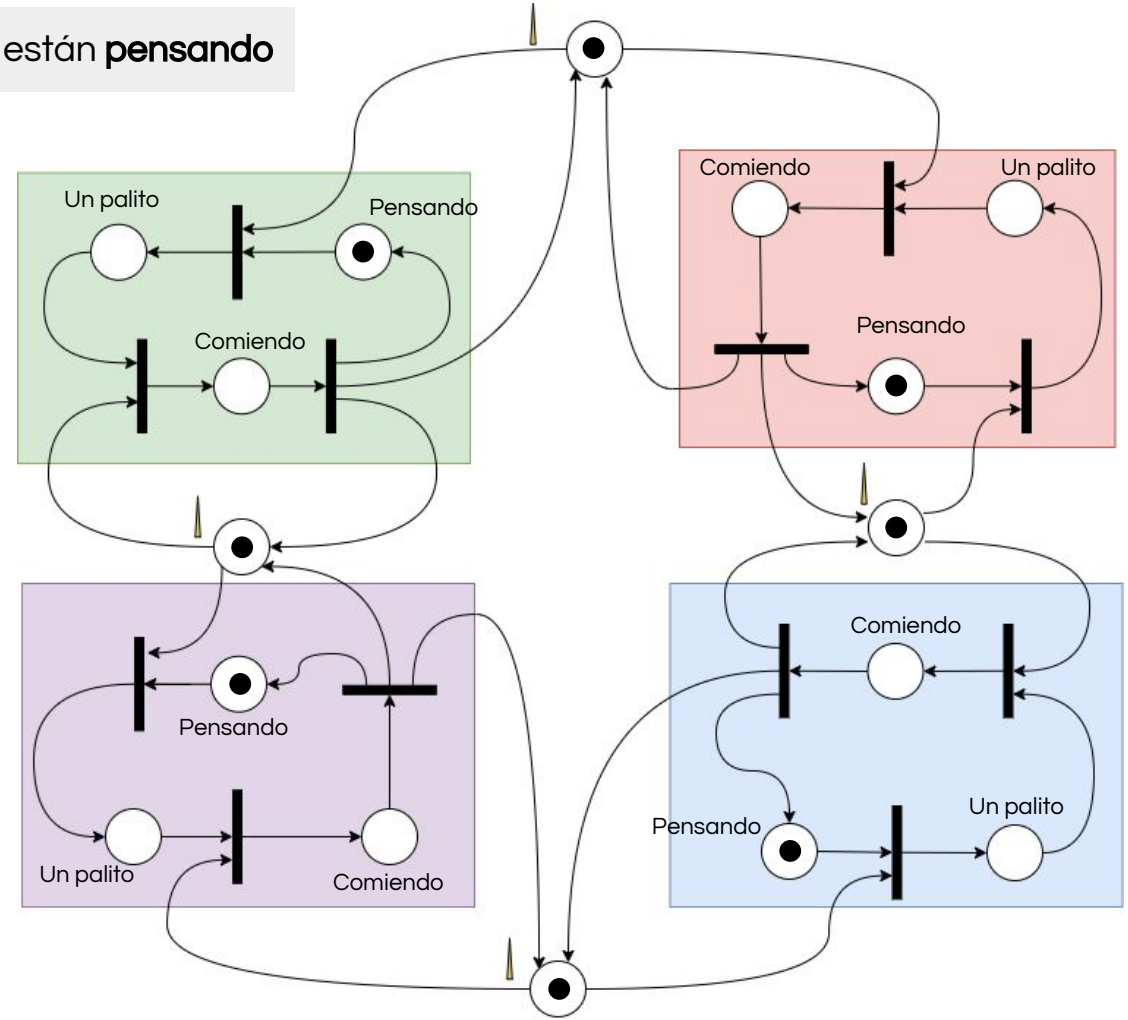
Cada filósofo tiene un plato de fideos y un palito chino a la izquierda de su plato.

Para comer los fideos son necesarios dos palitos y cada filósofo sólo puede tomar los que están a su izquierda y derecha (en ese orden). Si cualquier filósofo toma un palito y el otro está ocupado, se quedará esperando, con el palito en la mano, hasta que pueda tomar el otro palito, para luego empezar a comer.

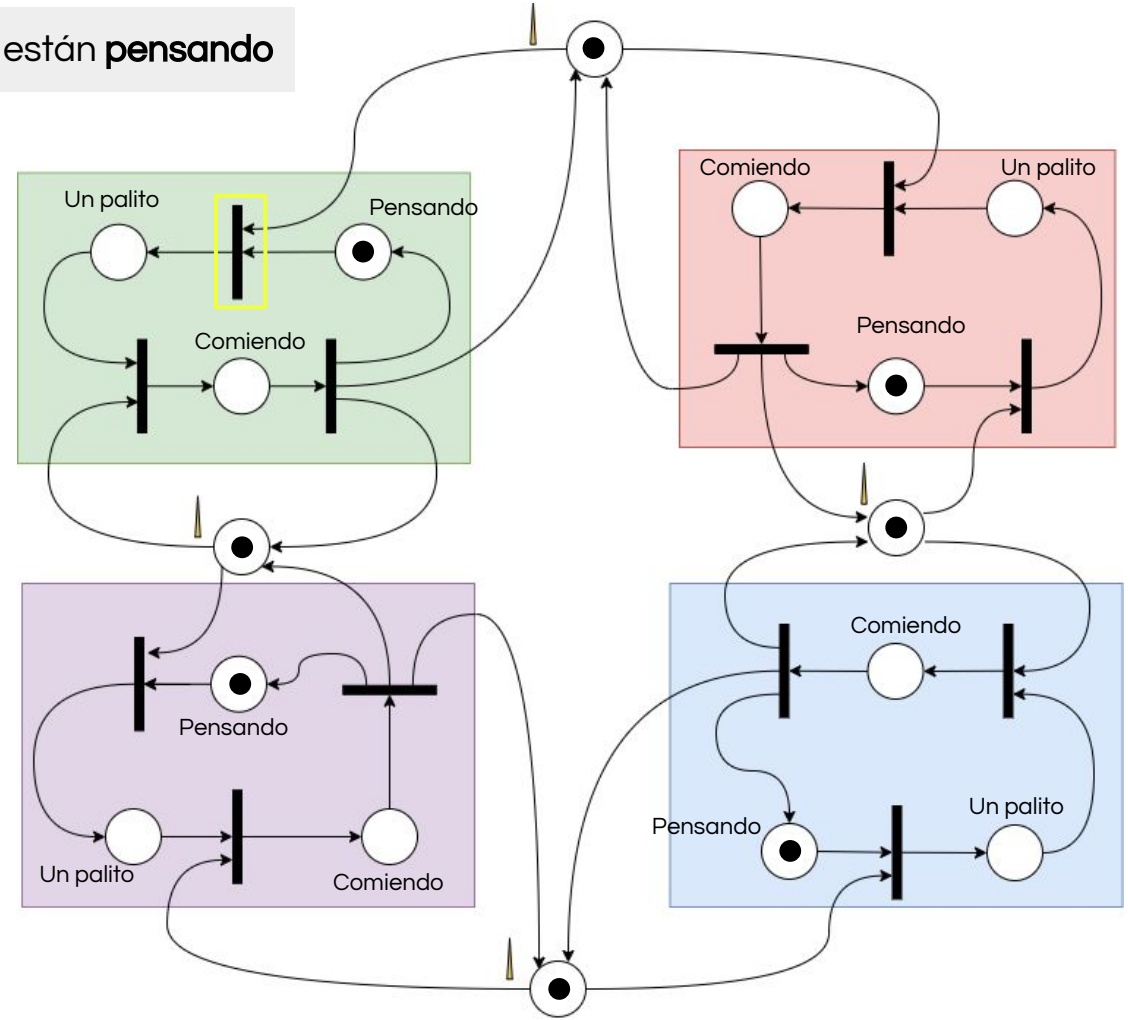
4 filósofos



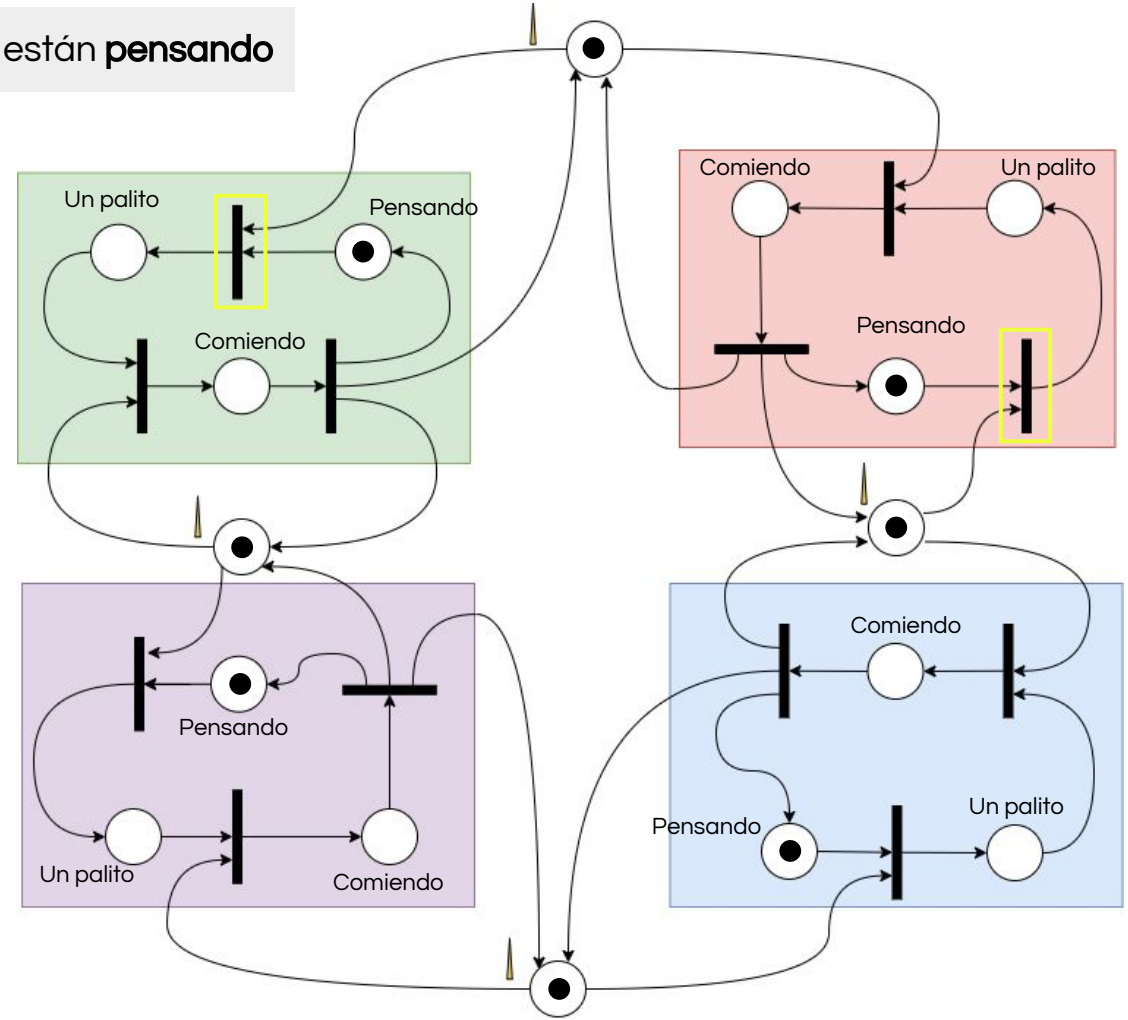
Todos los filósofos están **pensando**



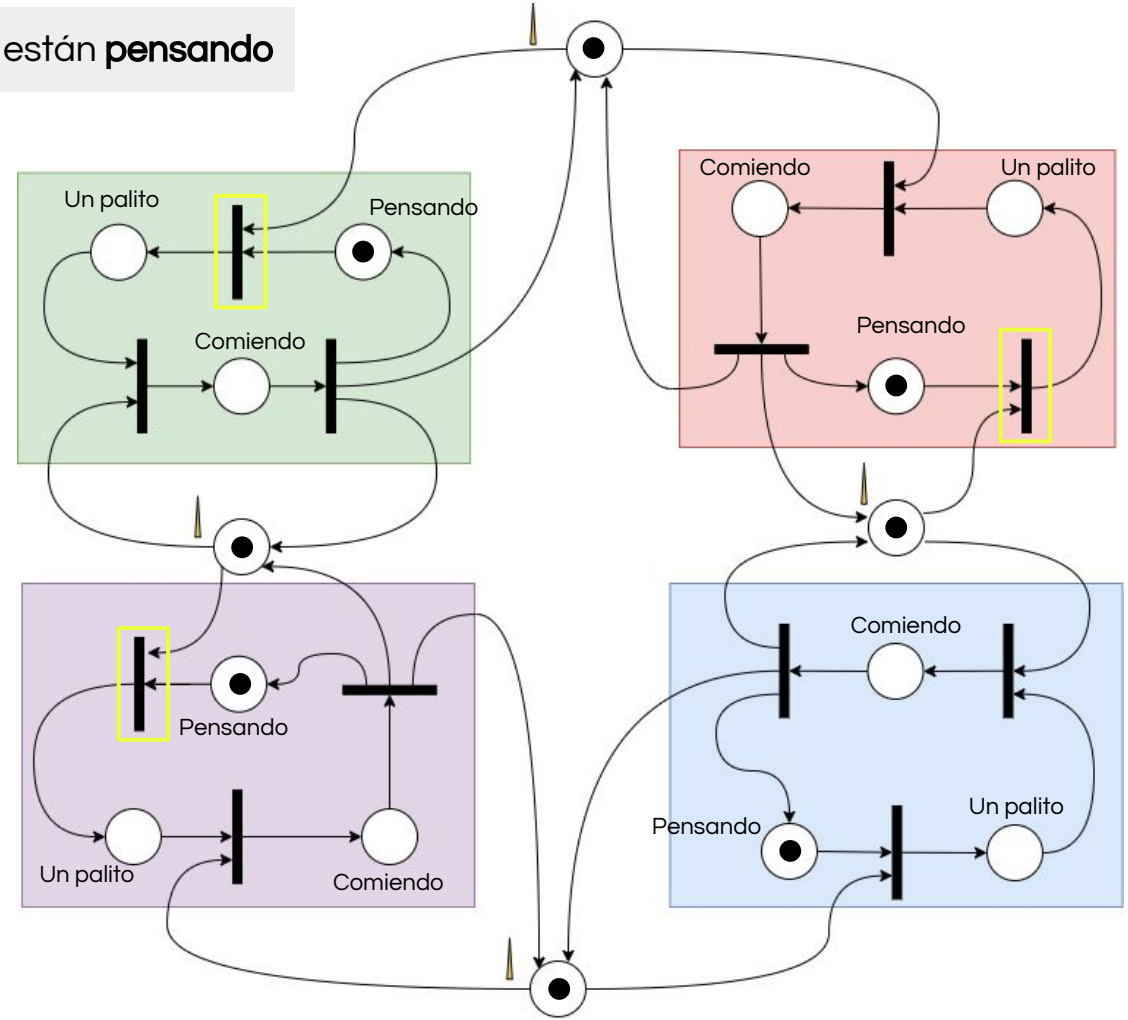
Todos los filósofos están **pensando**



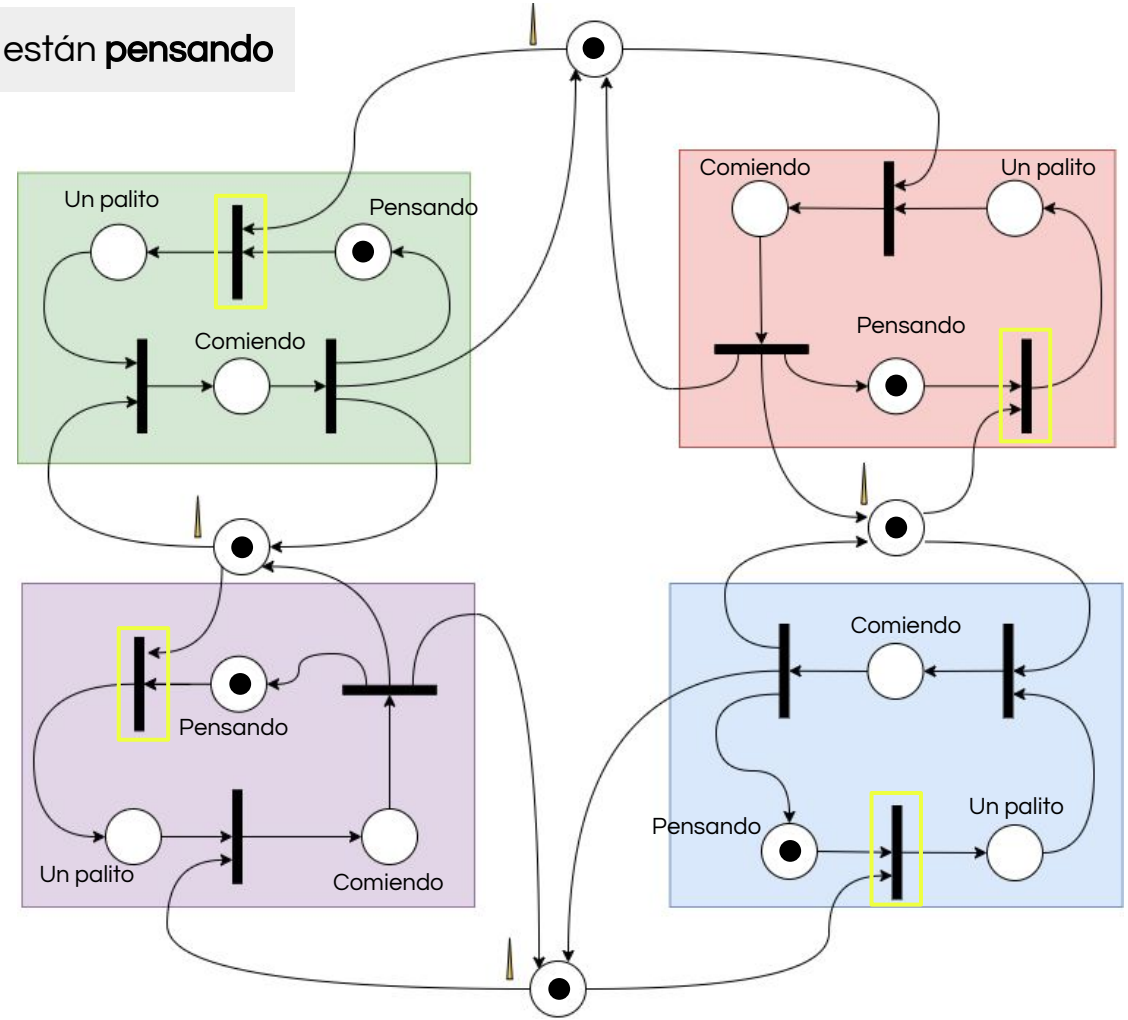
Todos los filósofos están **pensando**



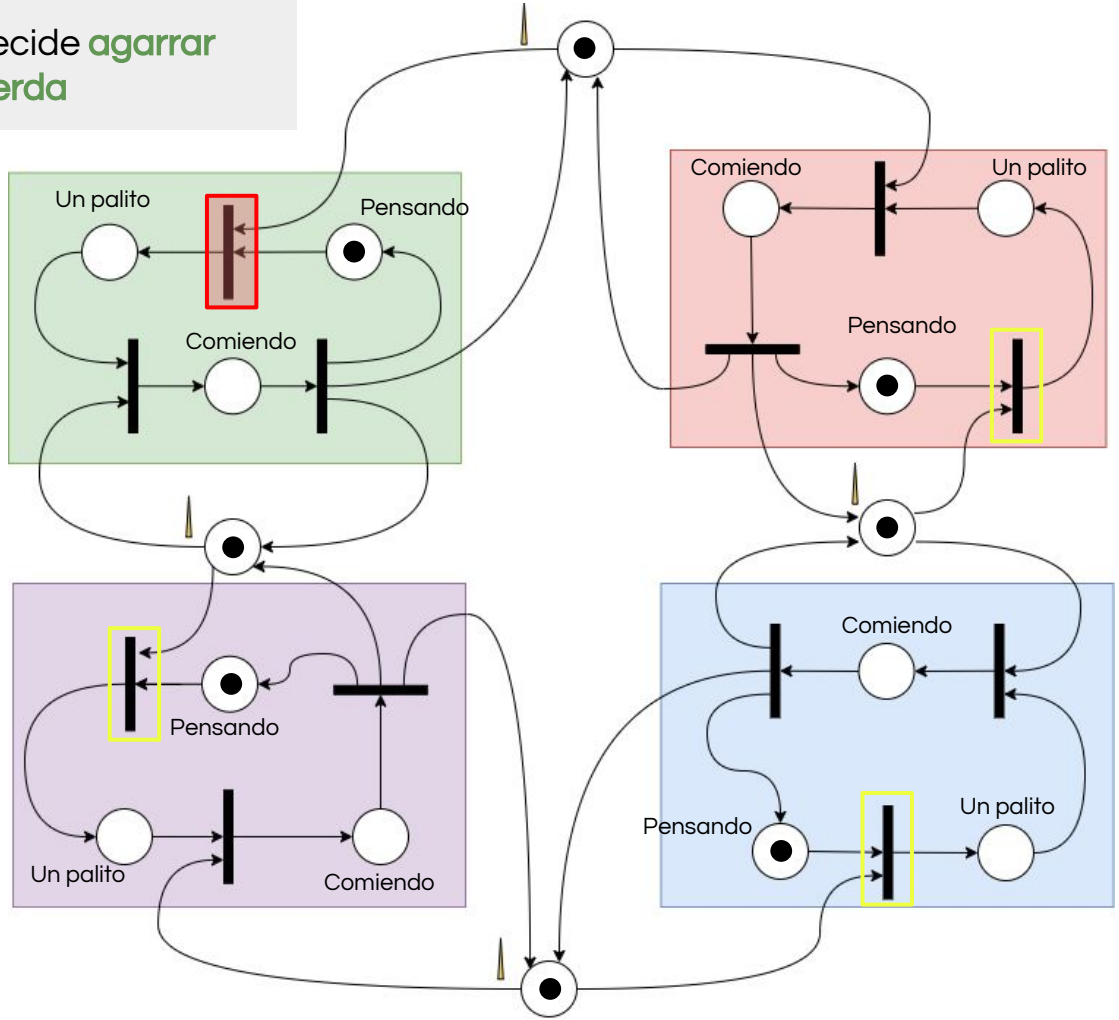
Todos los filósofos están **pensando**



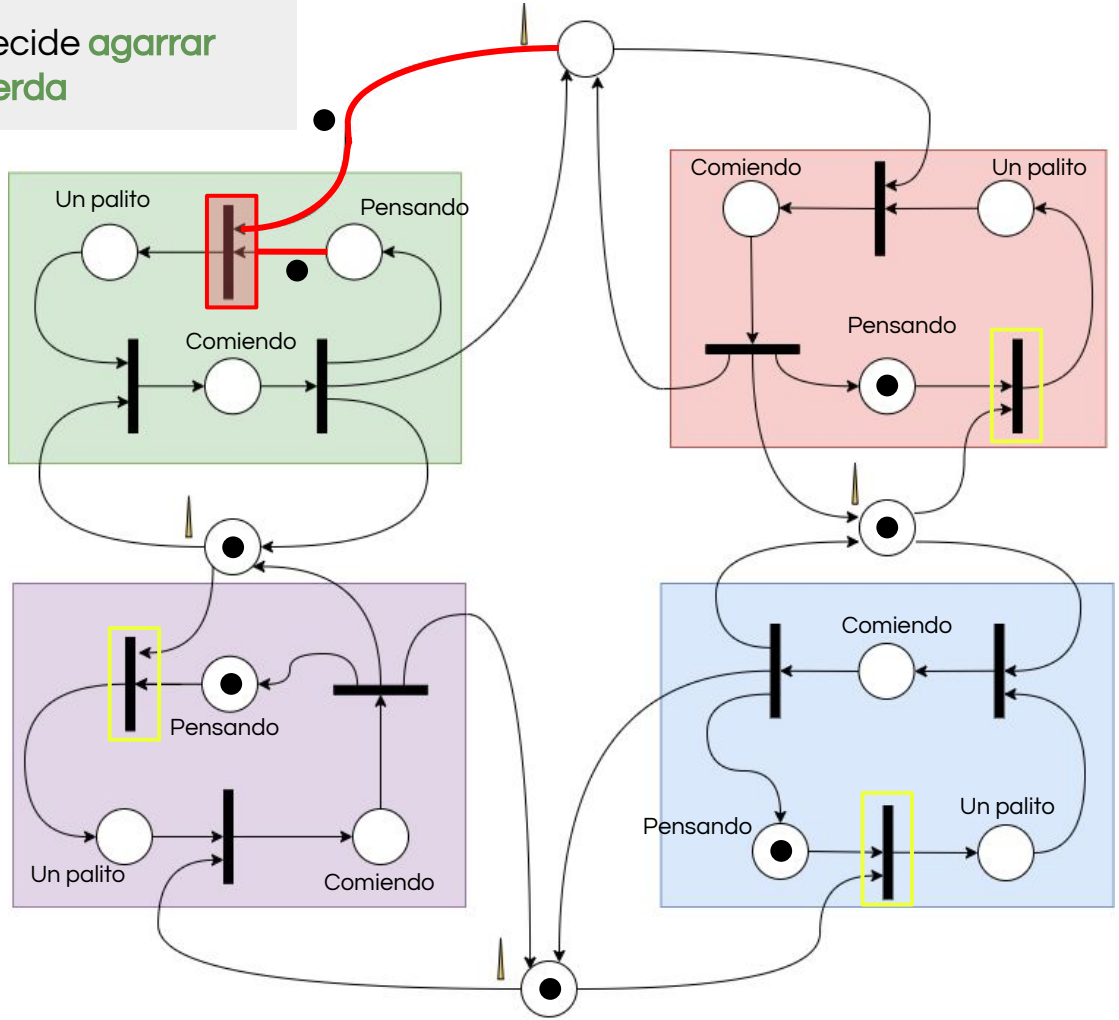
Todos los filósofos están **pensando**



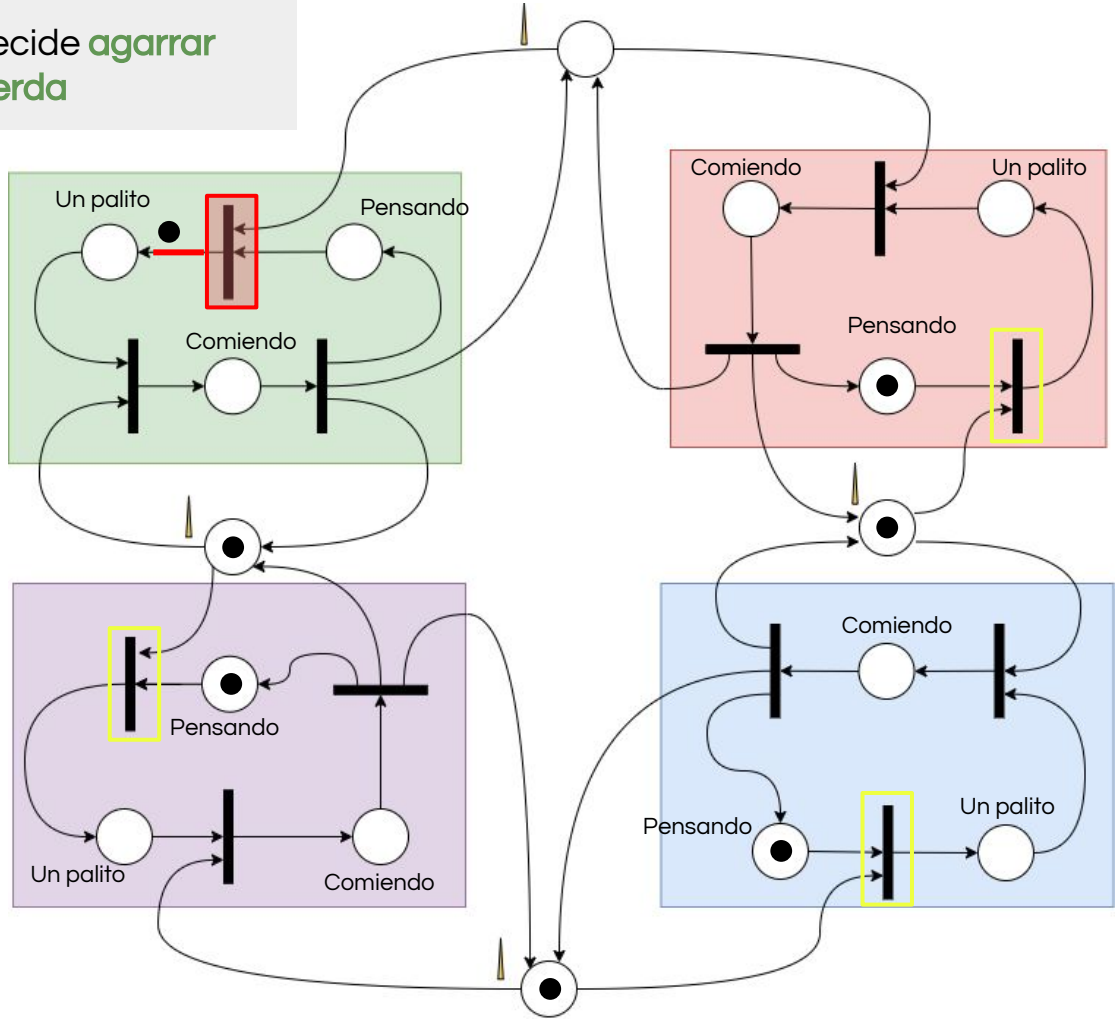
El filósofo verde decide agarrar el palito a su izquierda



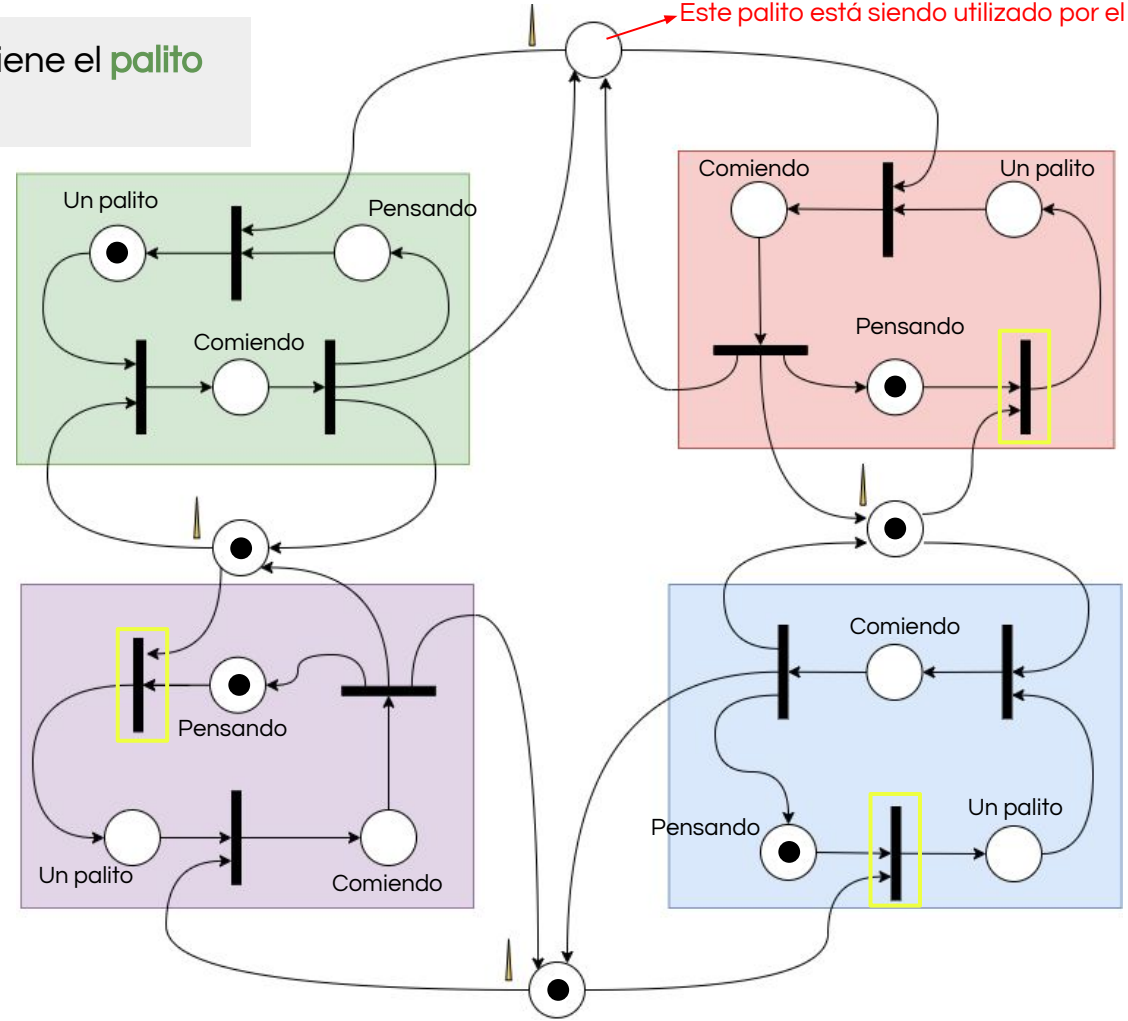
El filósofo verde decide agarrar el palito a su izquierda



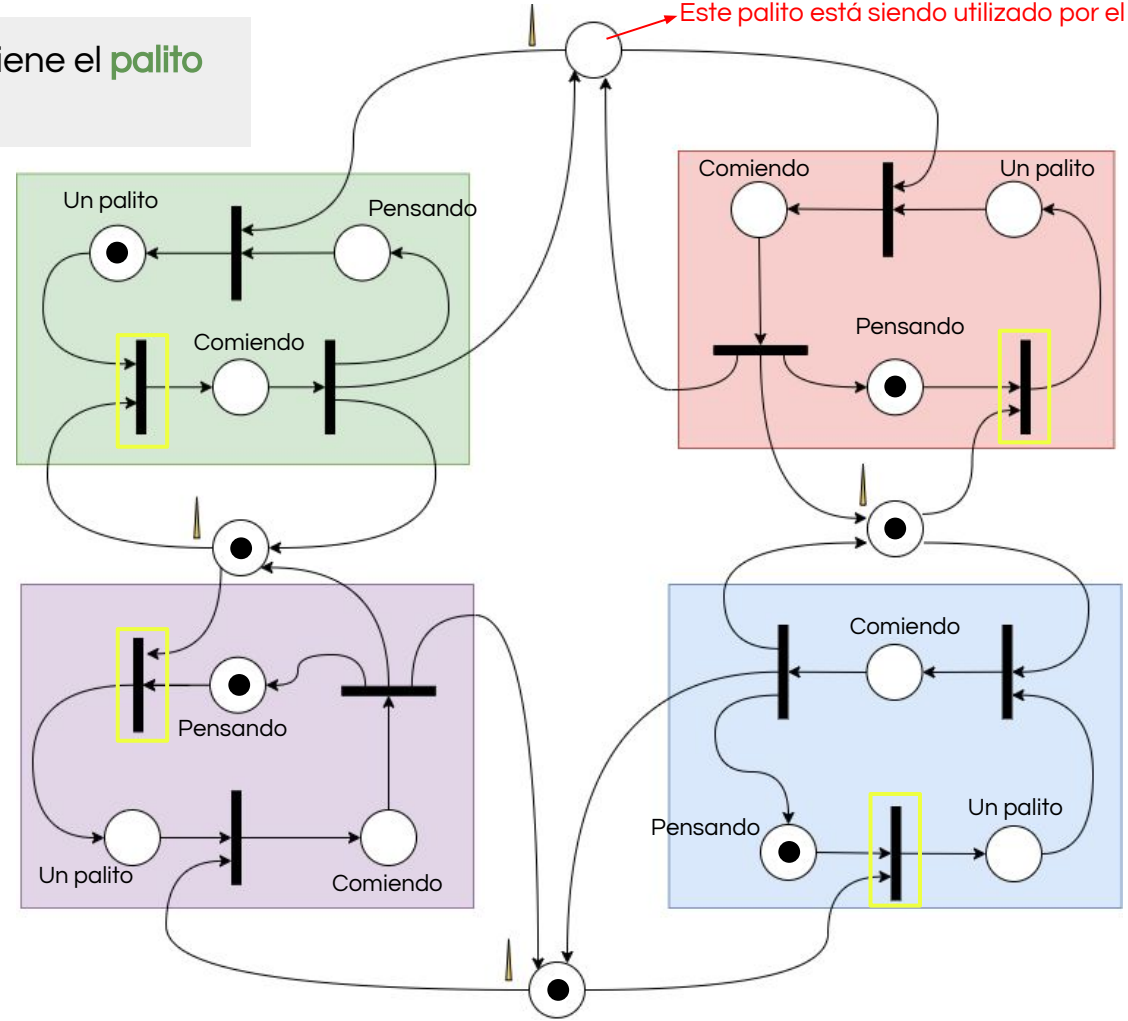
El filósofo verde decide agarrar el palito a su izquierda



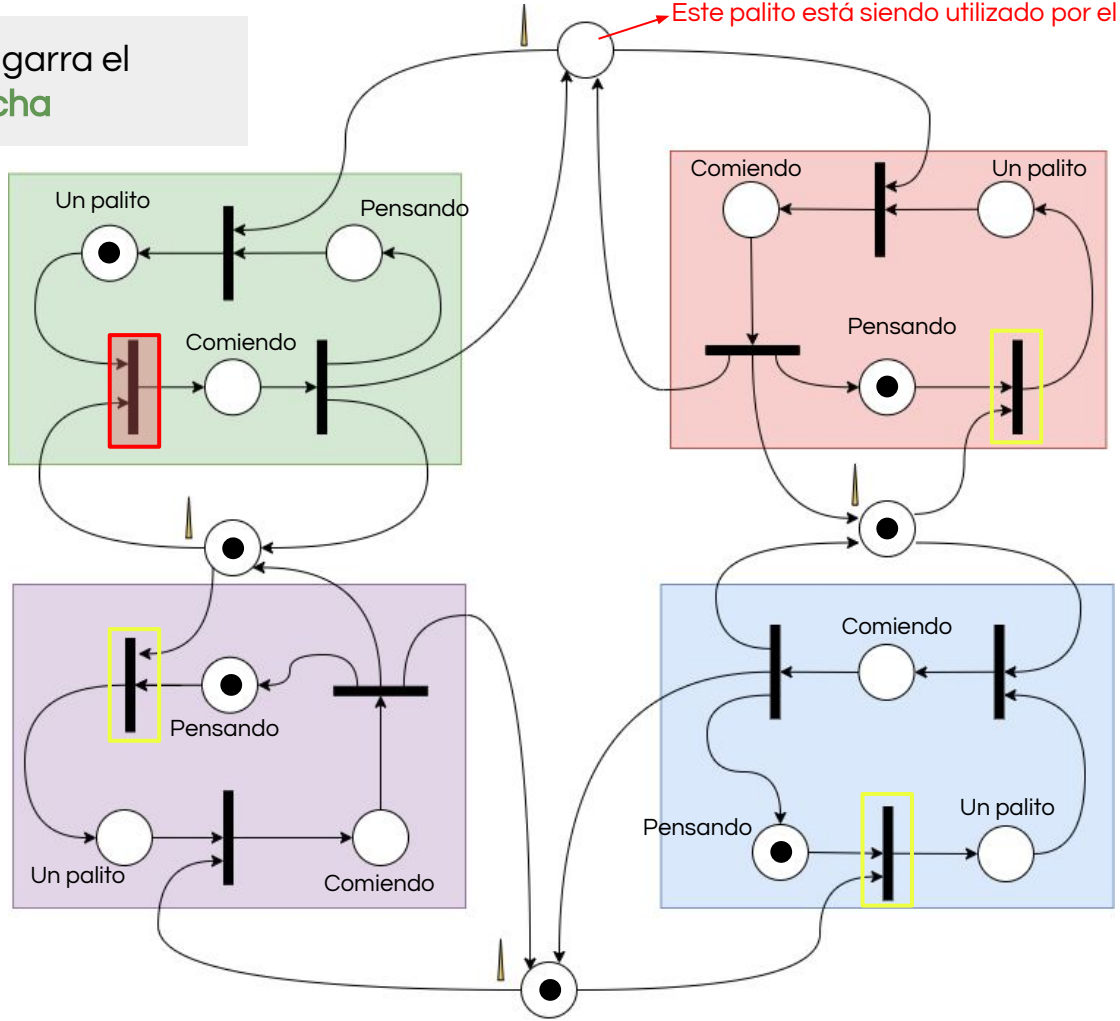
El filósofo verde tiene el palito de su izquierda



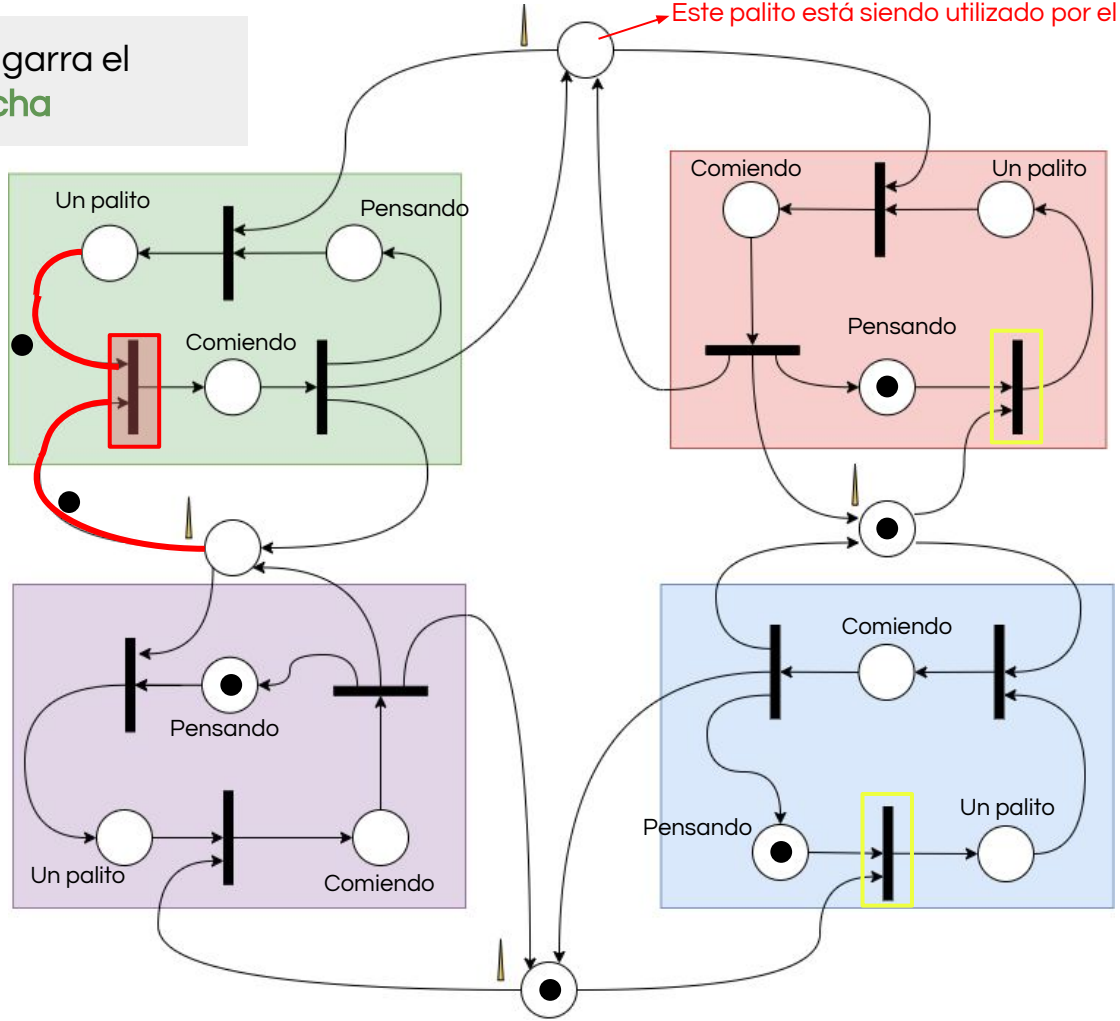
El **filósofo verde** tiene el **palito de su izquierda**



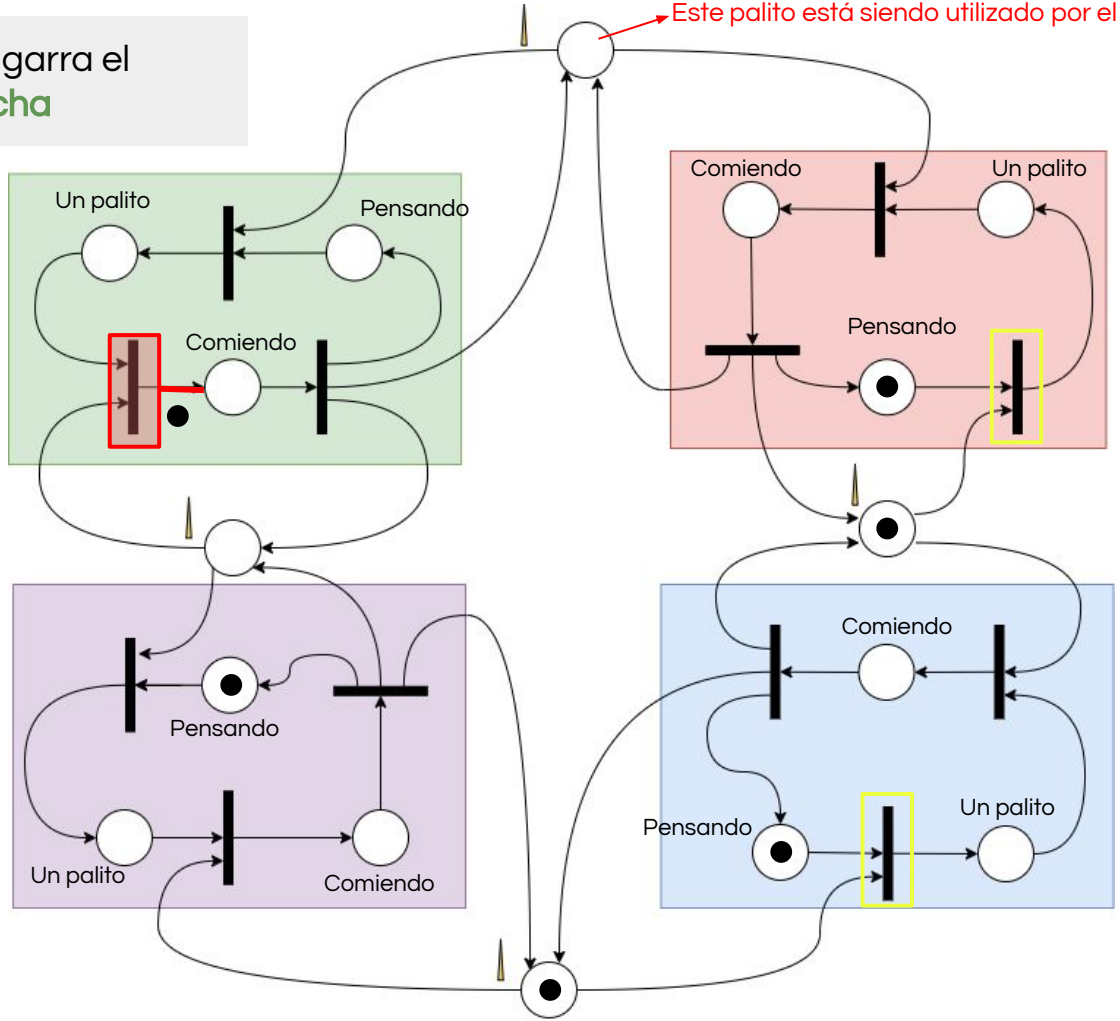
El filósofo verde agarra el palito de su derecha



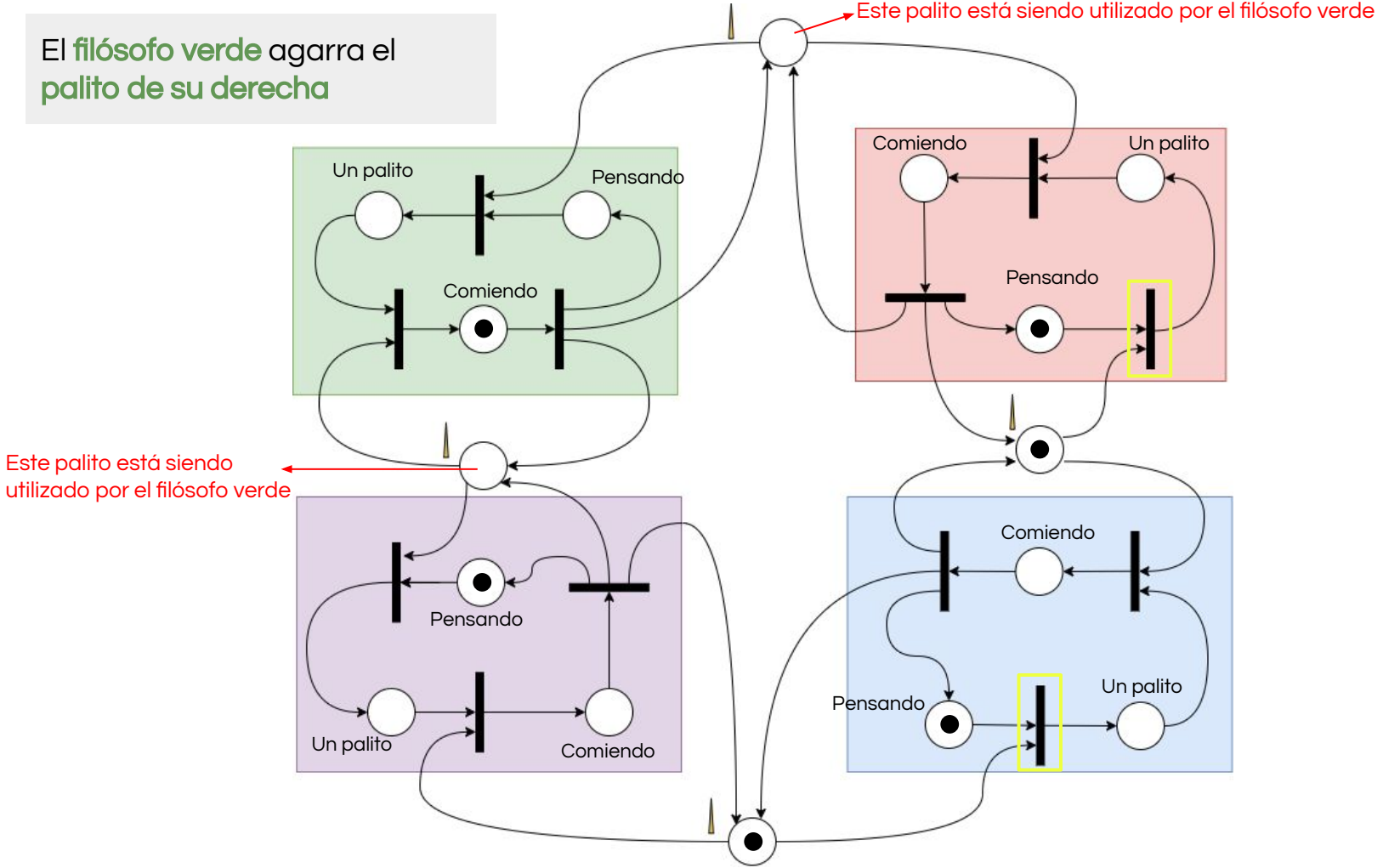
El filósofo verde agarra el palito de su derecha



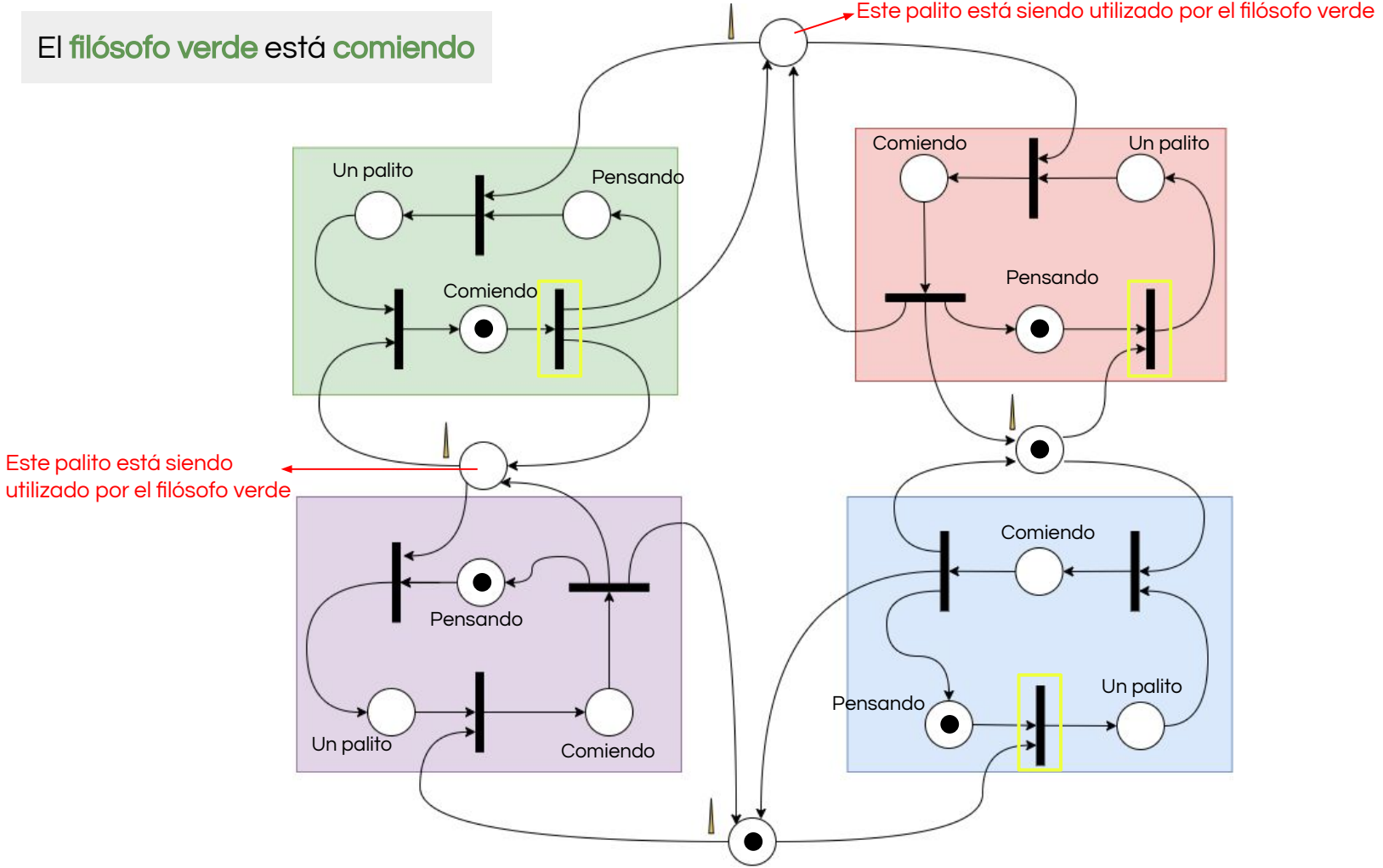
El filósofo verde agarra el palito de su derecha



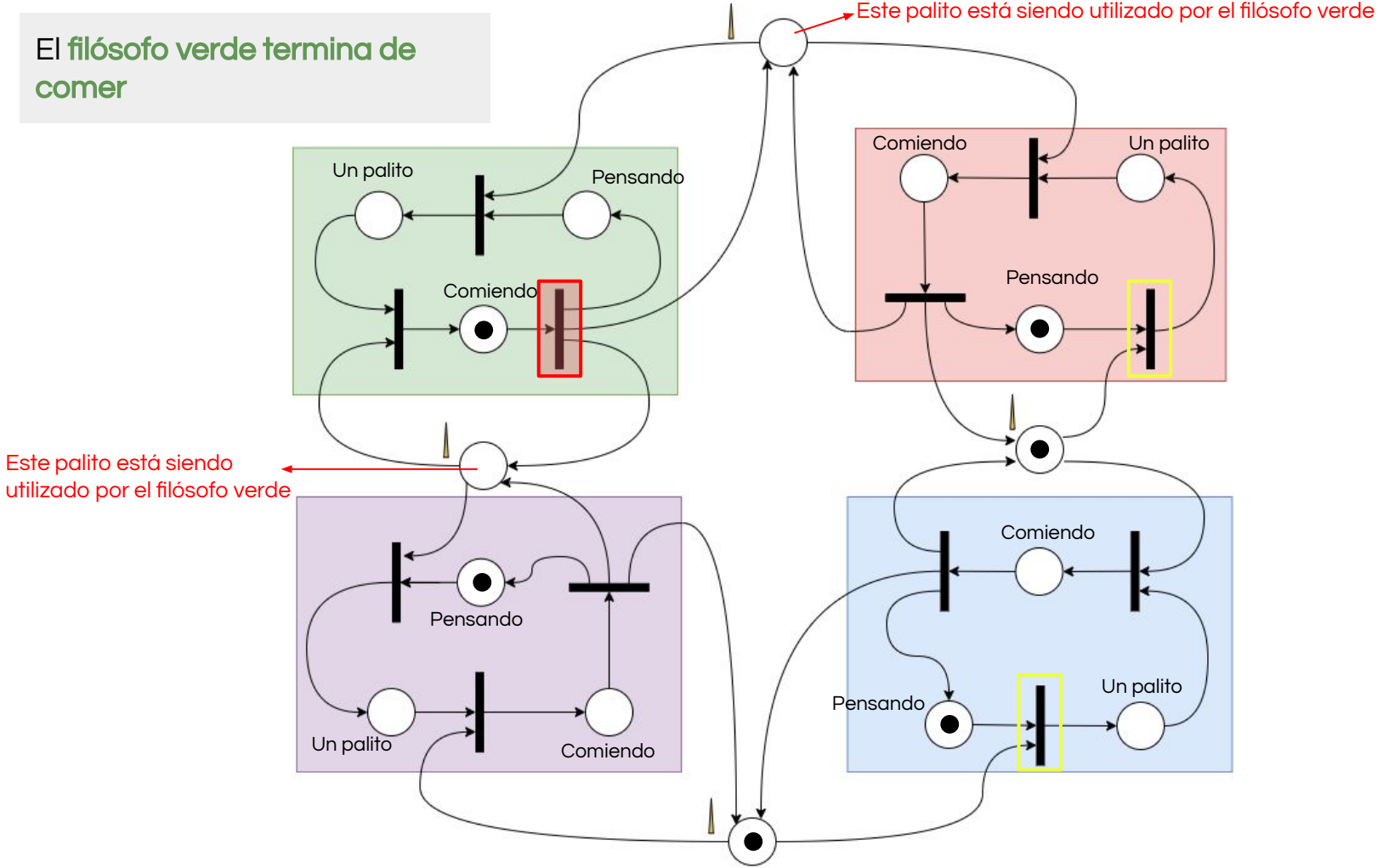
El **filósofo verde** agarra el **palito de su derecha**



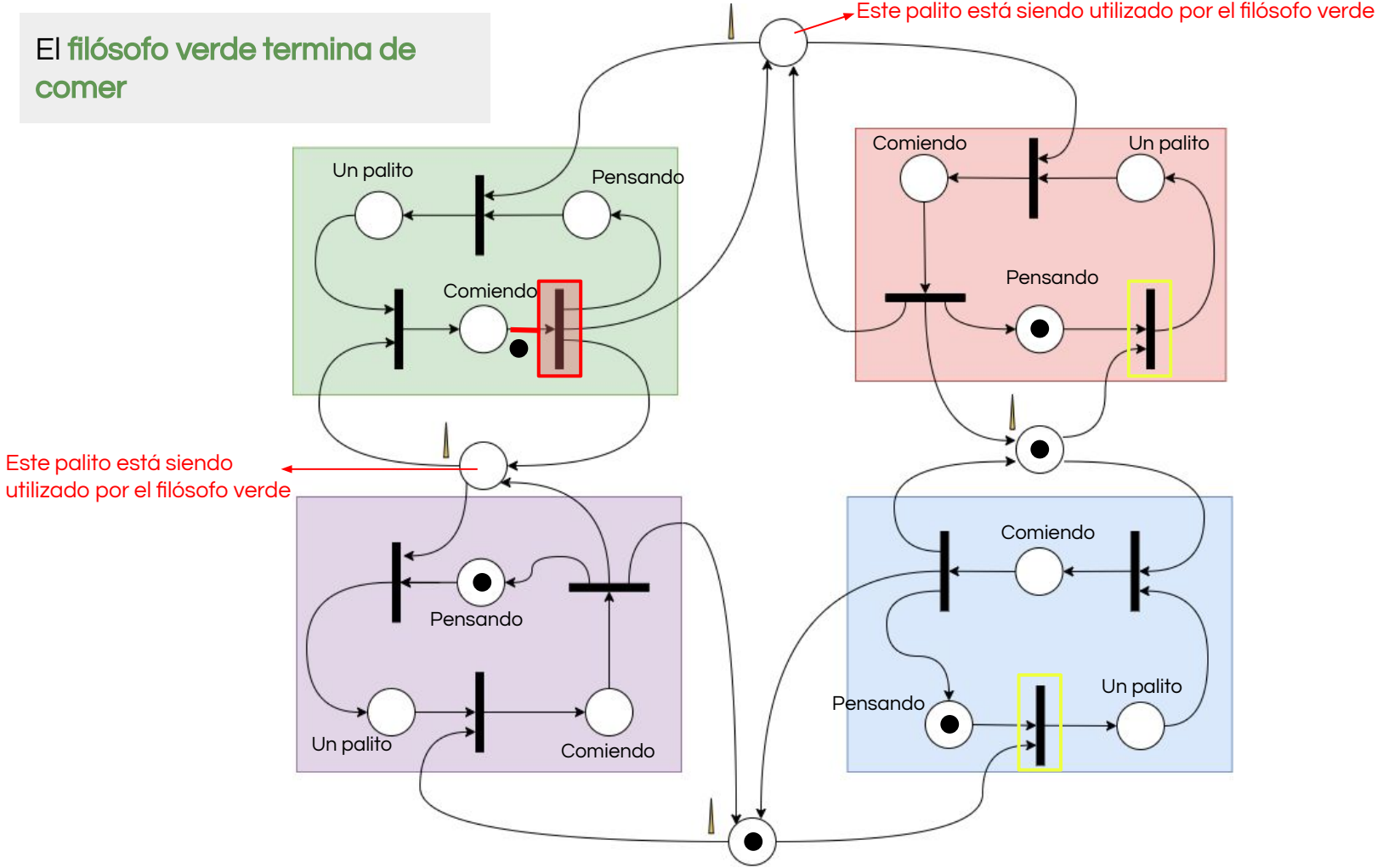
El filósofo verde está comiendo



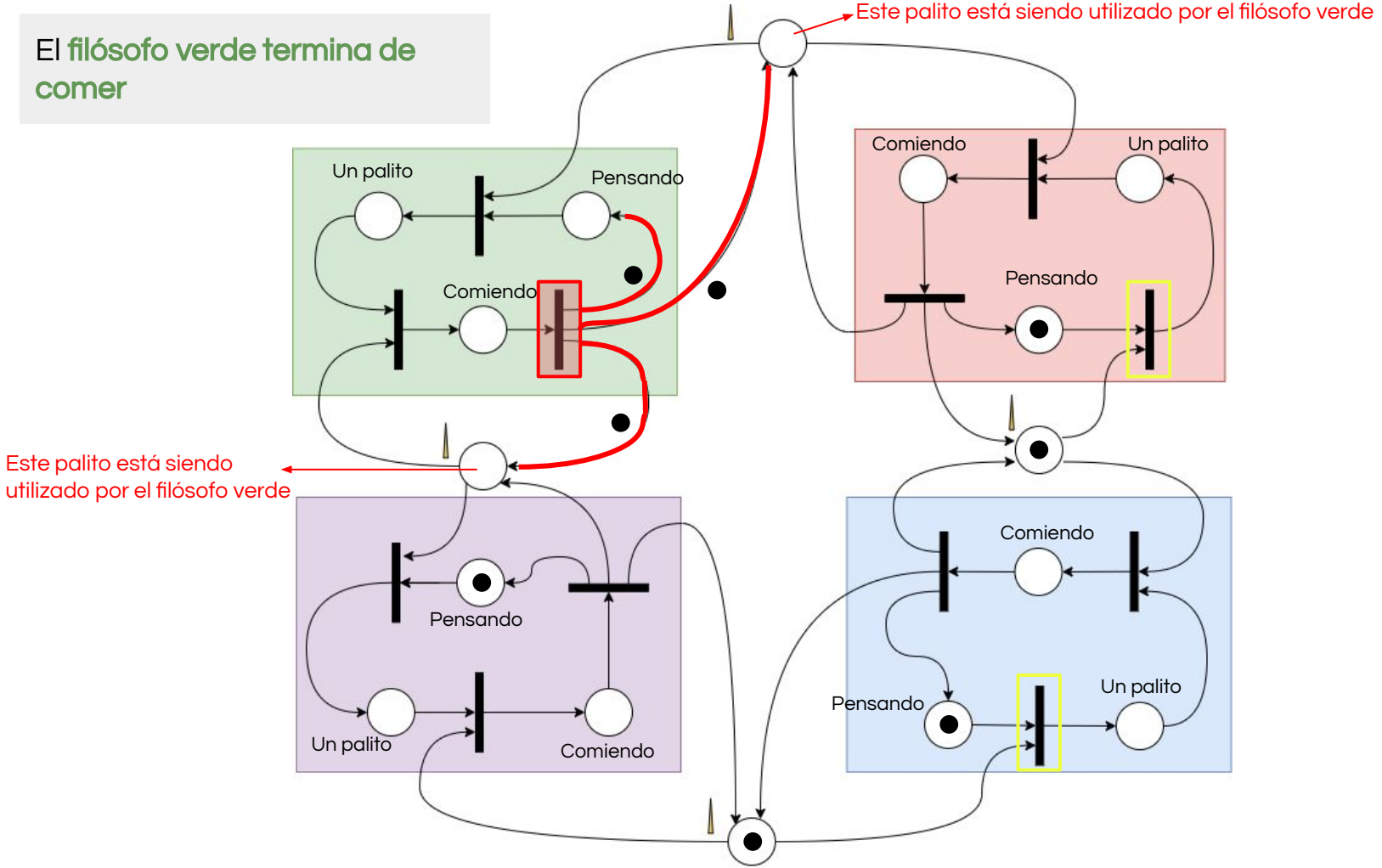
El filósofo verde termina de comer



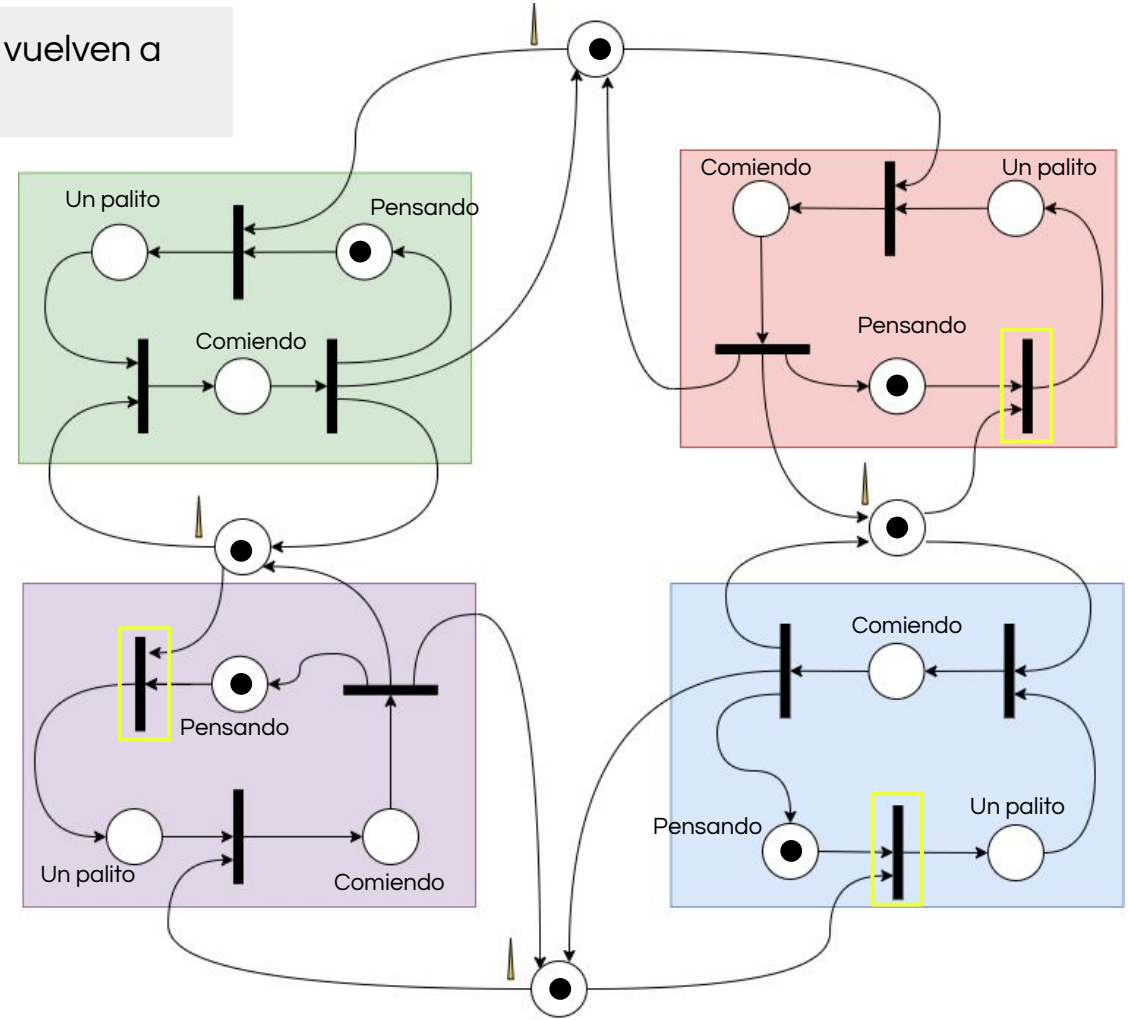
El filósofo verde termina de comer



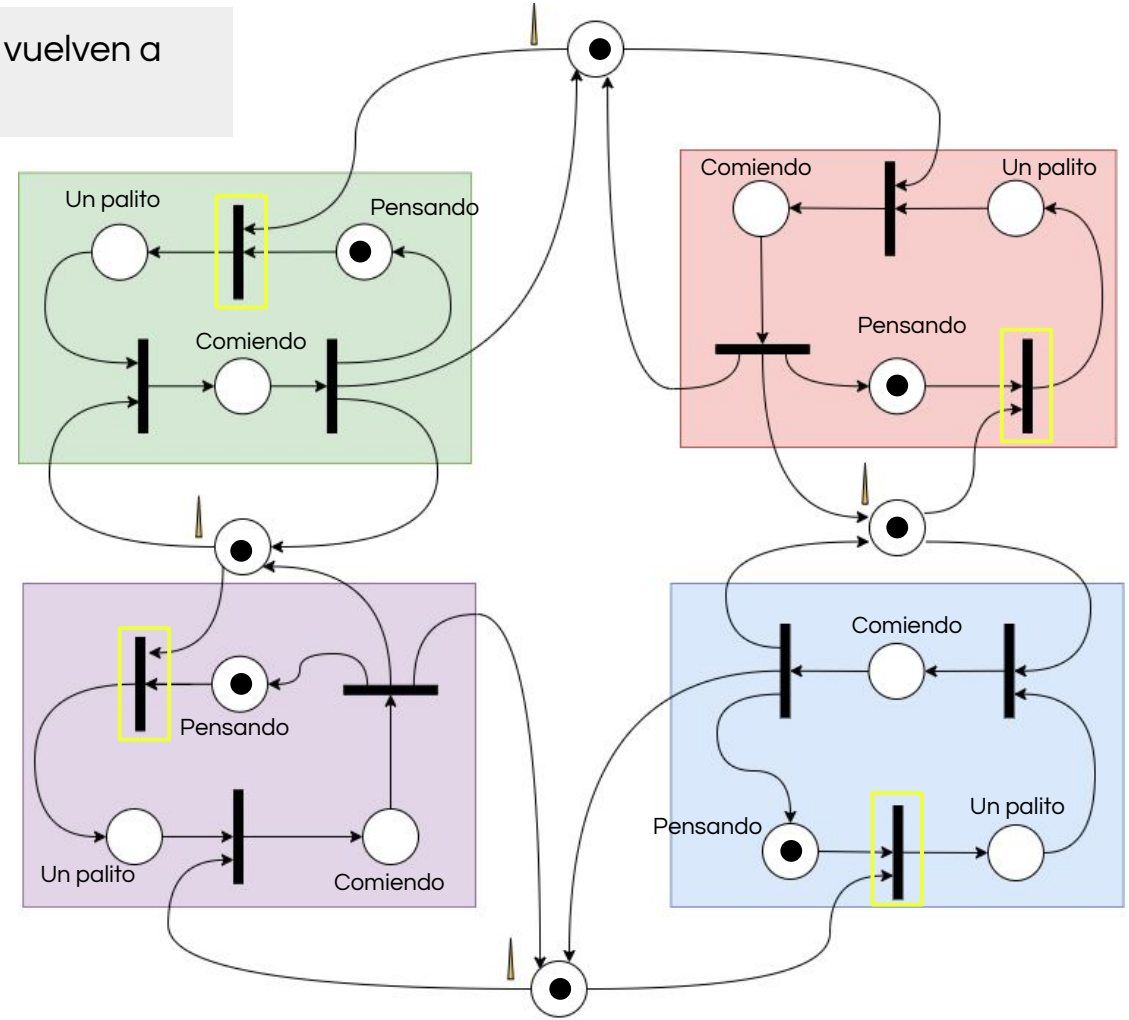
El filósofo verde termina de comer



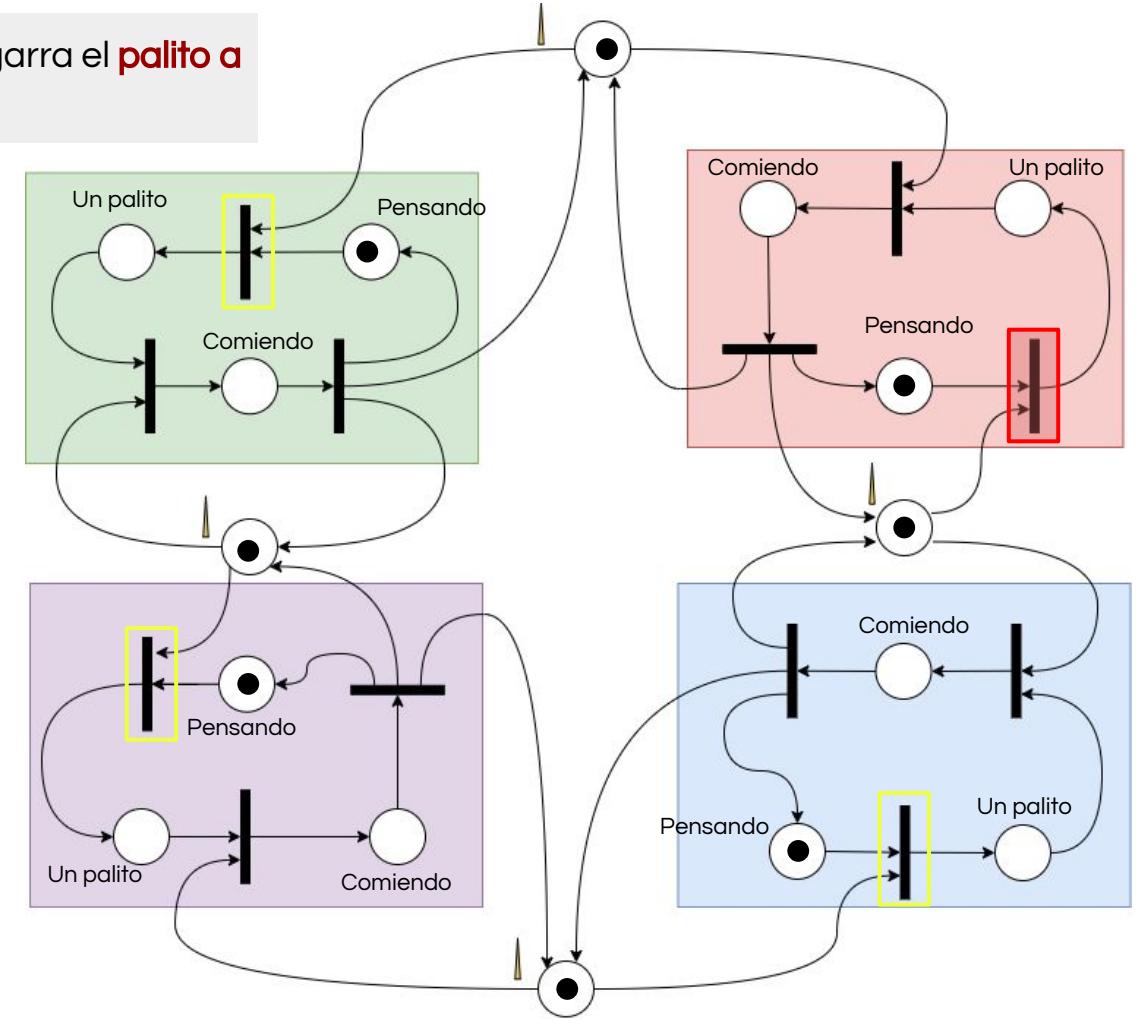
Todos los filósofos vuelven a estar **pensando**



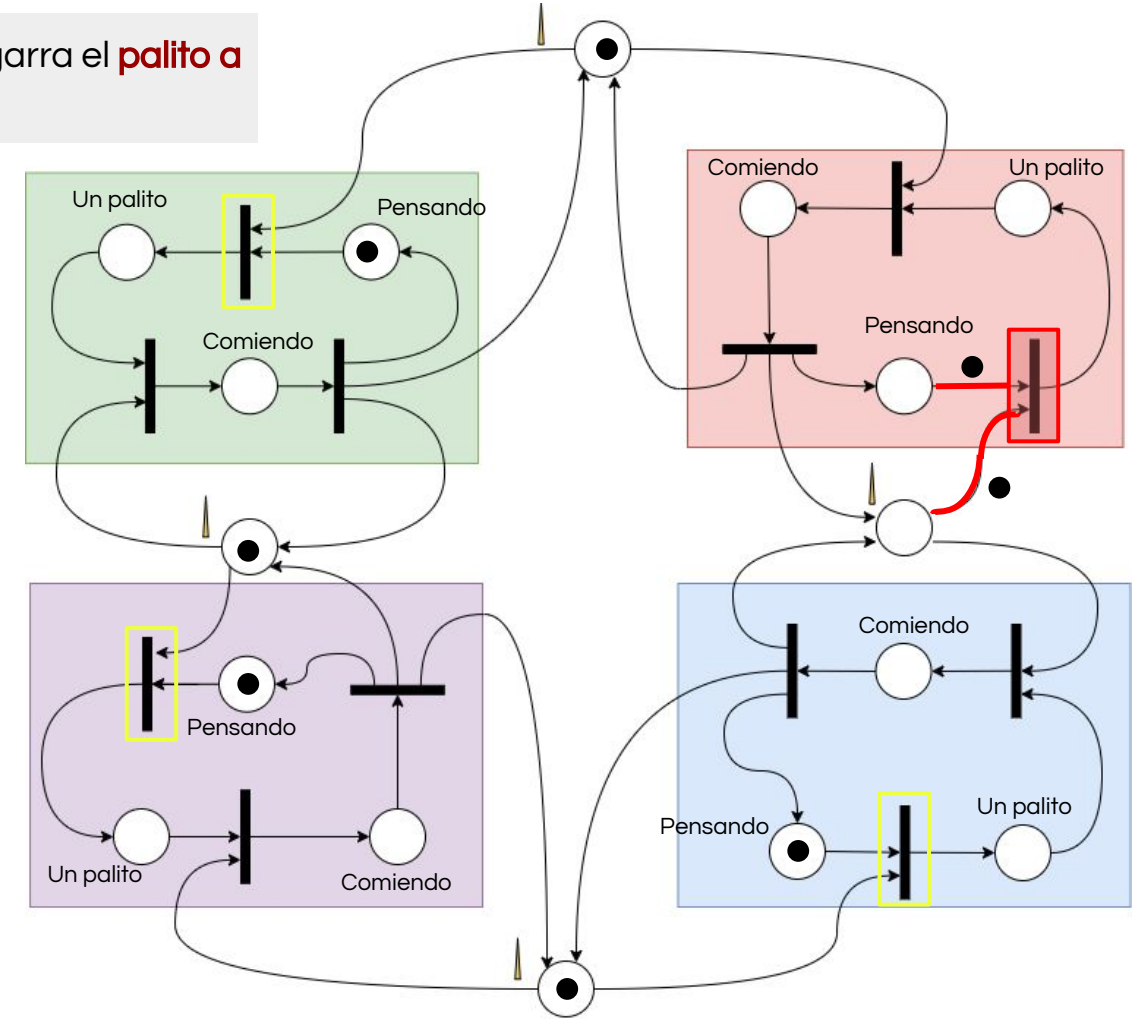
Todos los filósofos vuelven a estar **pensando**



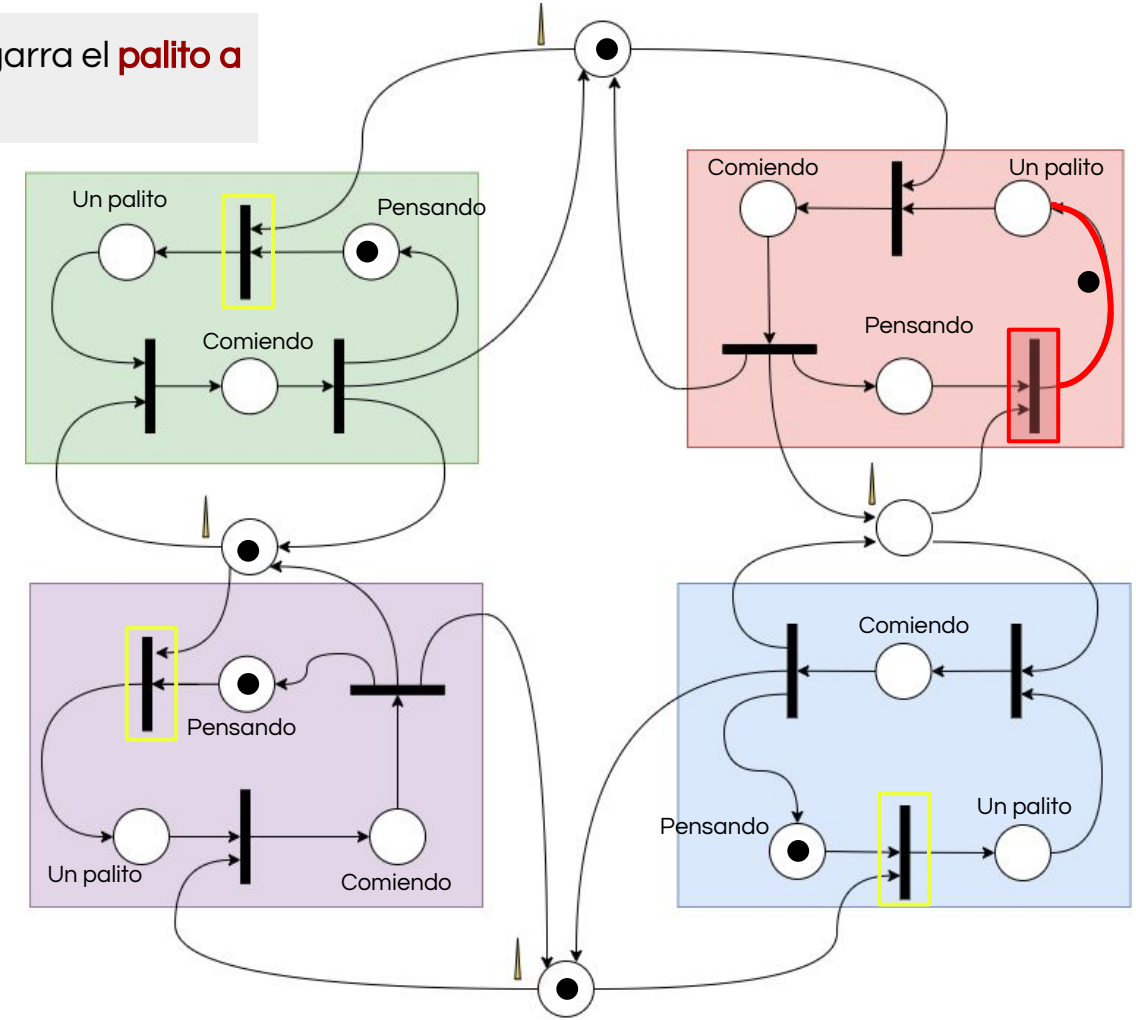
El **filósofo rojo** agarra el **palito a su izquierda**



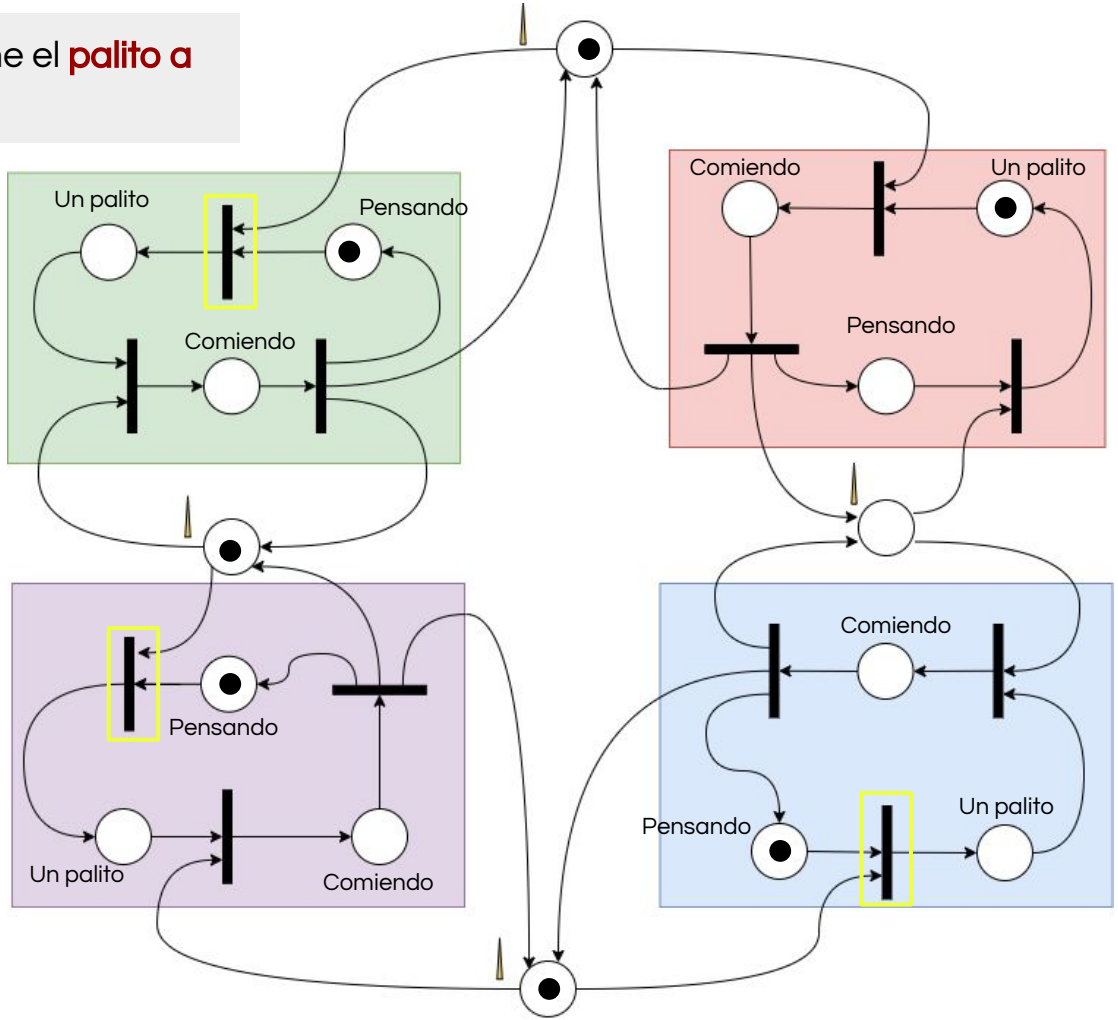
El **filósofo rojo** agarra el **palito a su izquierda**



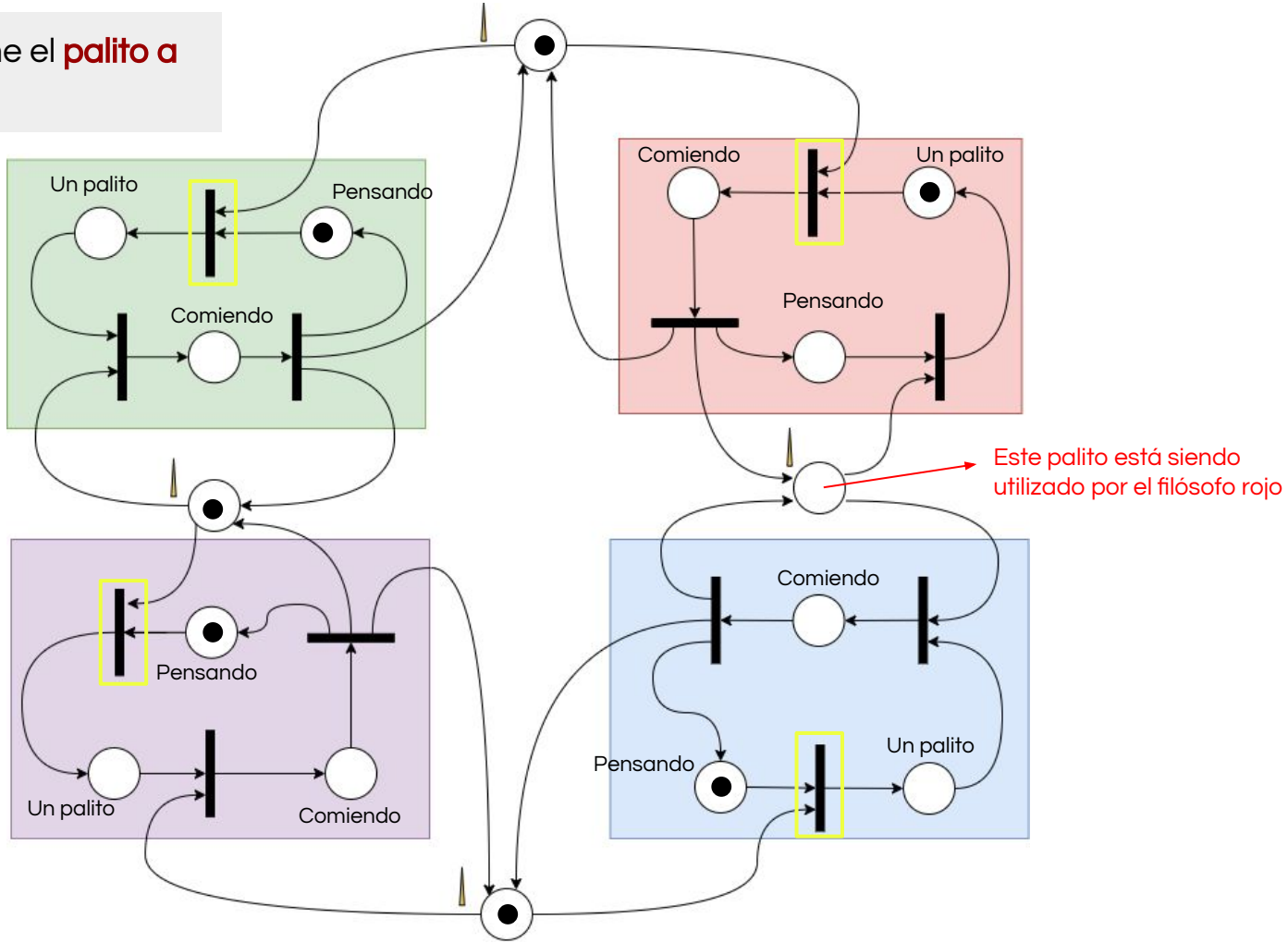
El **filósofo rojo** agarra el **palito a su izquierda**



El **filósofo rojo** tiene el **palito a su izquierda**

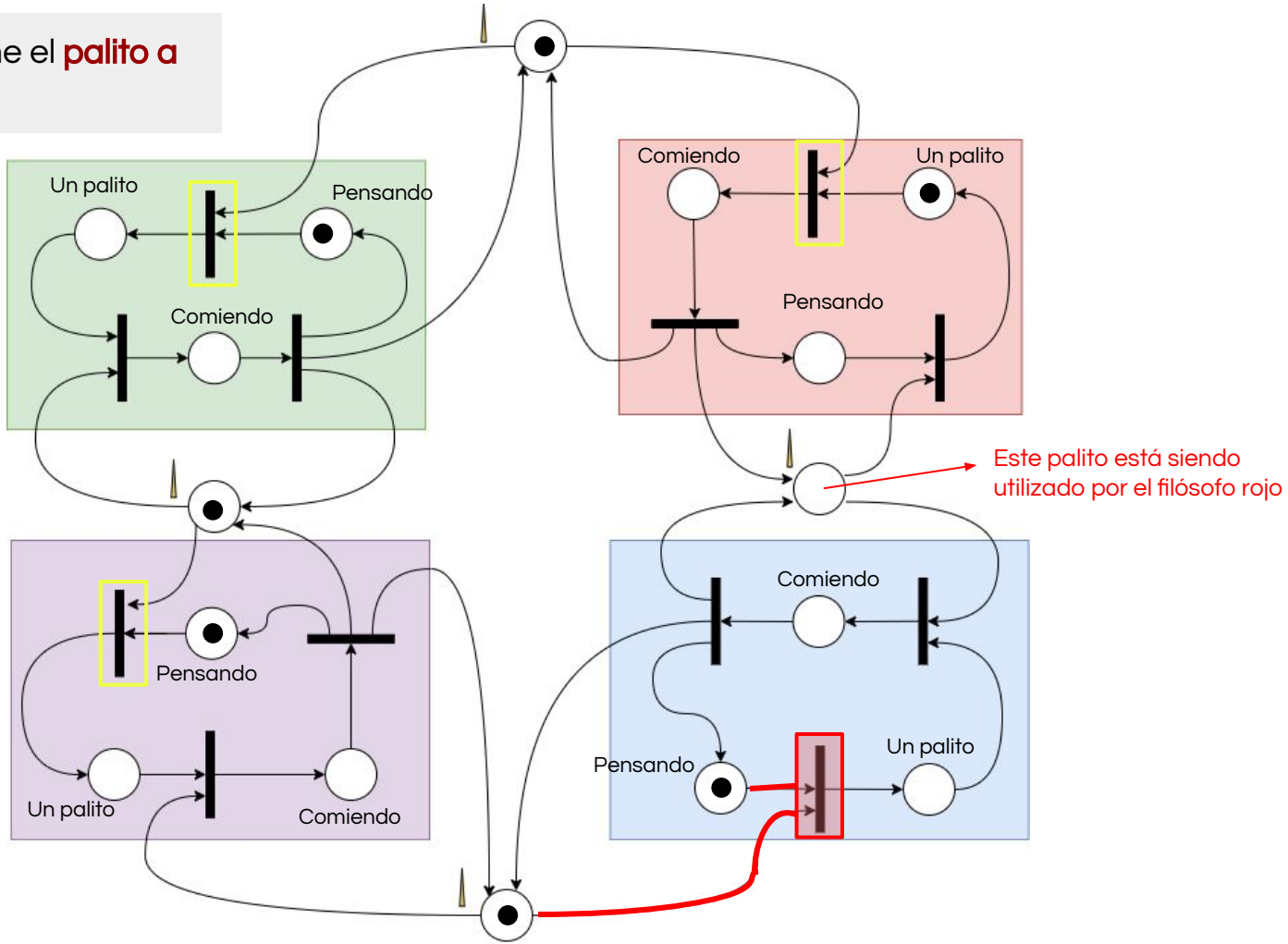


El **filósofo rojo** tiene el **palito a su izquierda**



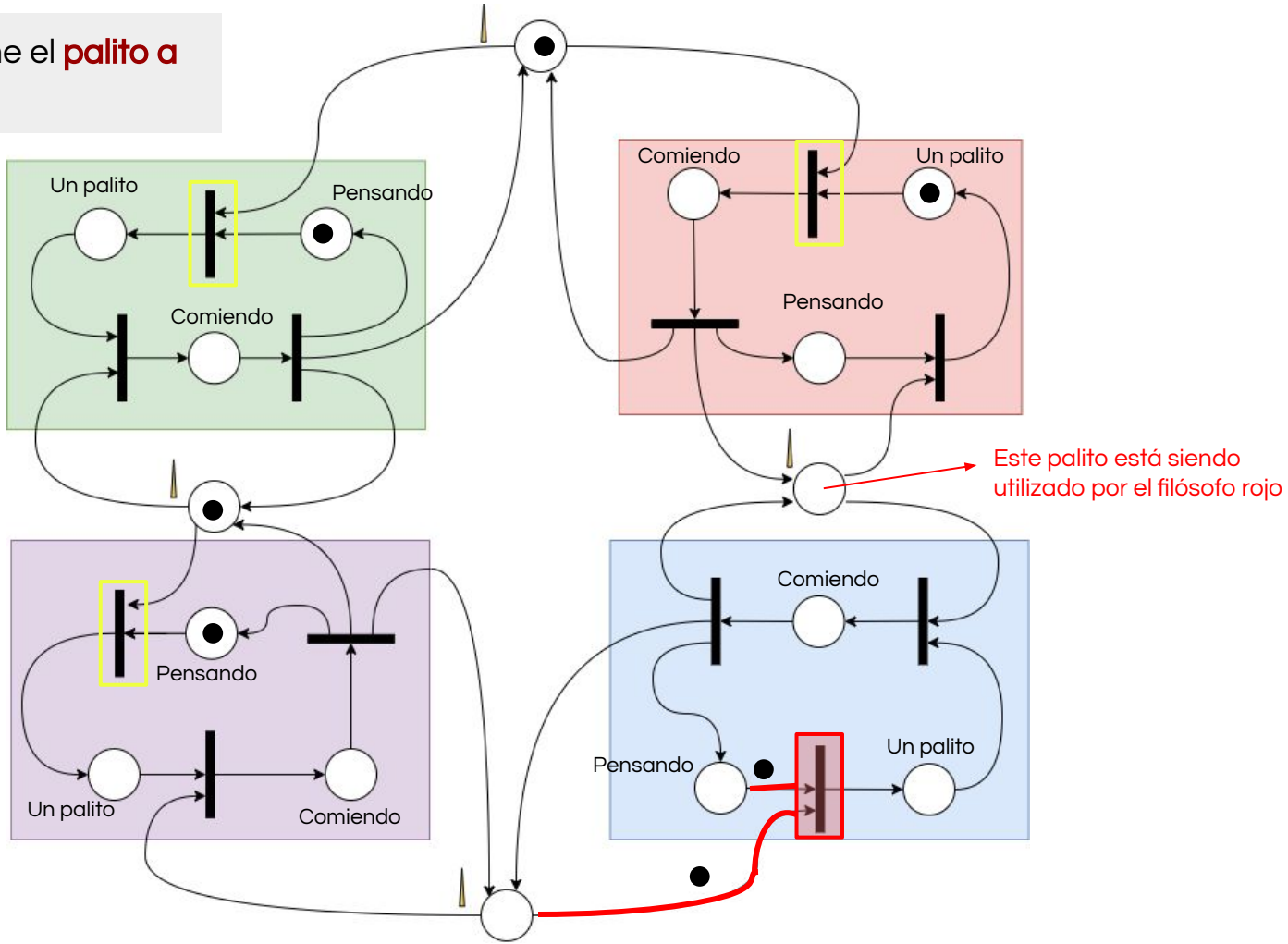
El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** agarra el **palito a su izquierda**



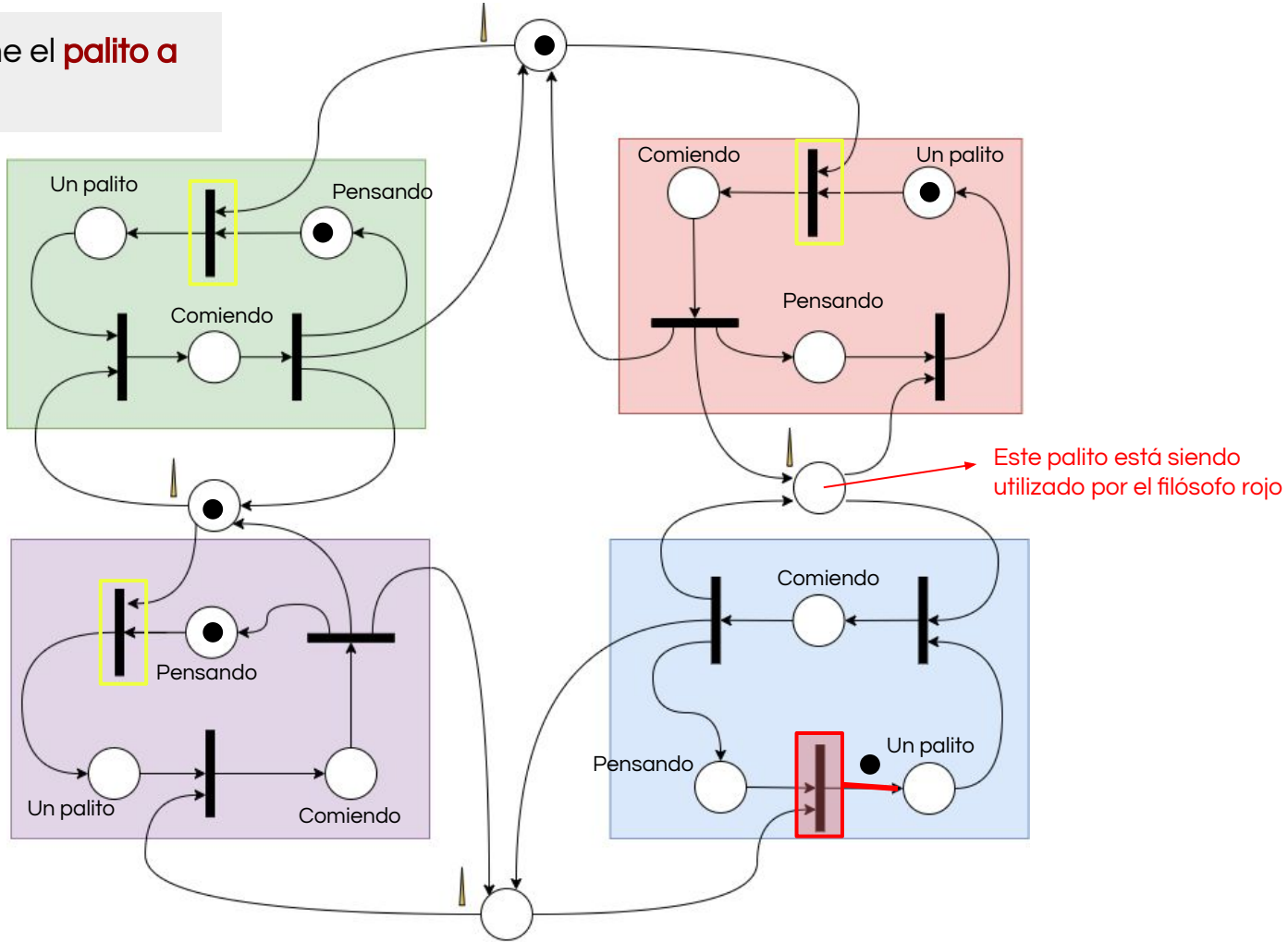
El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** agarra el **palito a su izquierda**



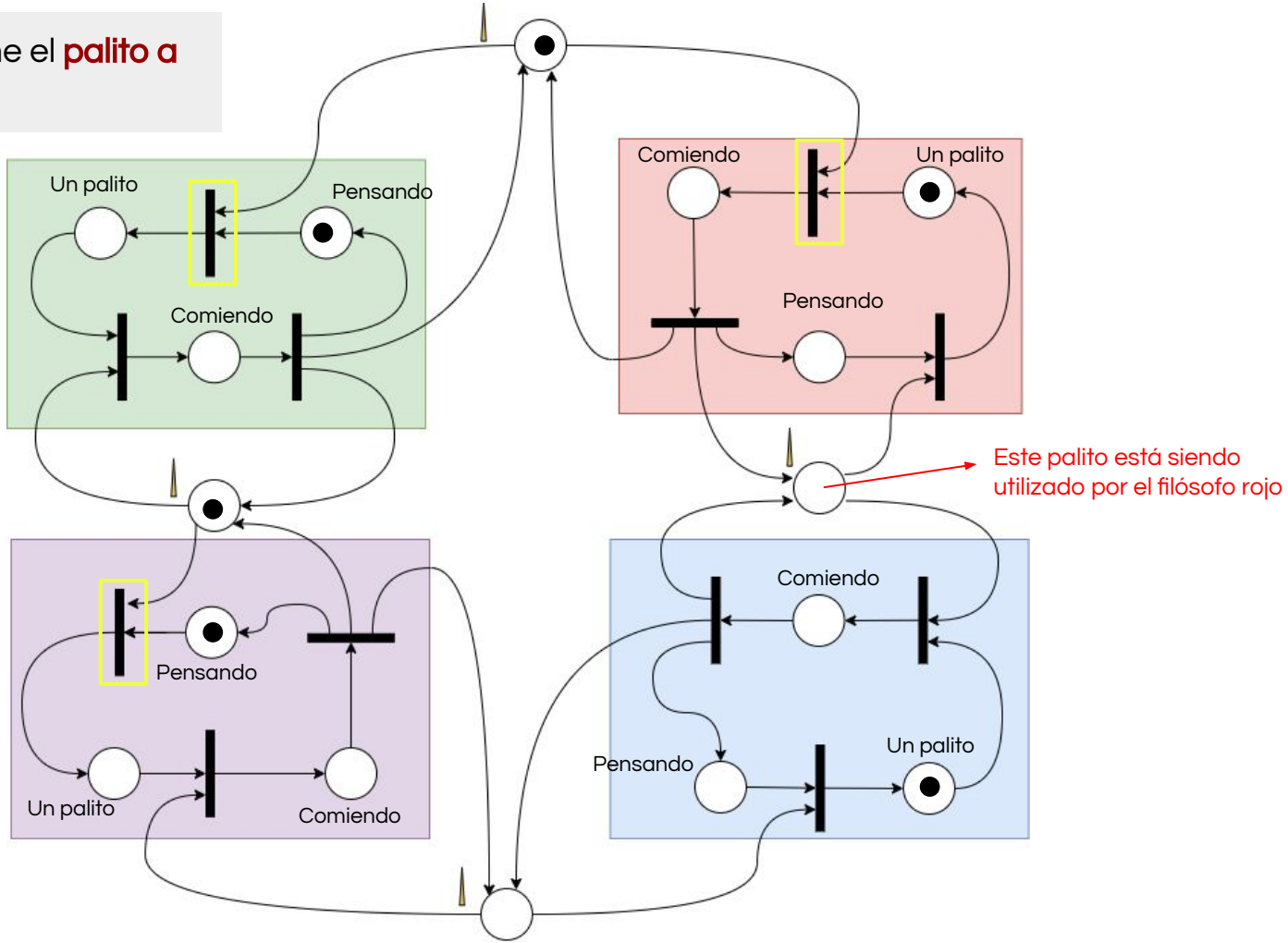
El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** agarra el **palito a su izquierda**



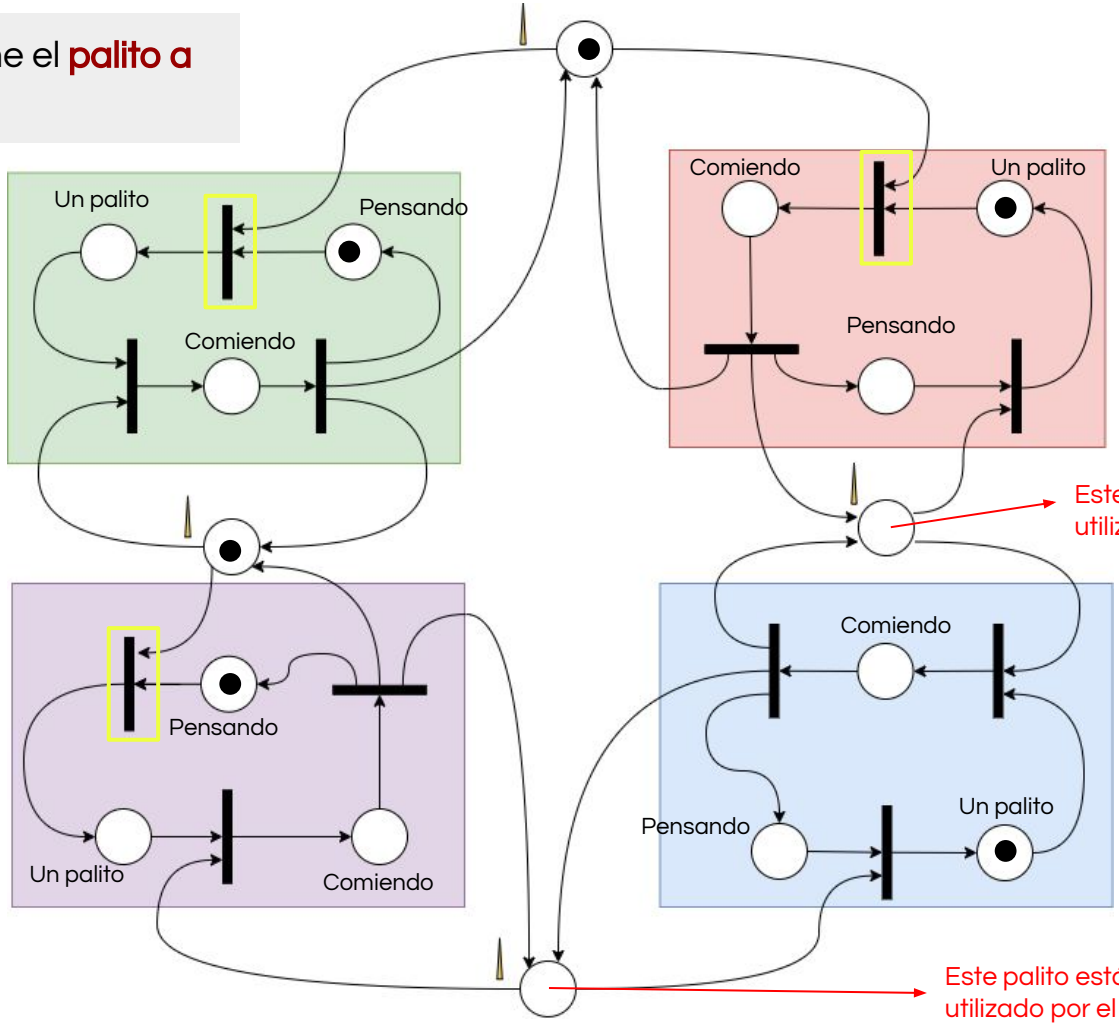
El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**



El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

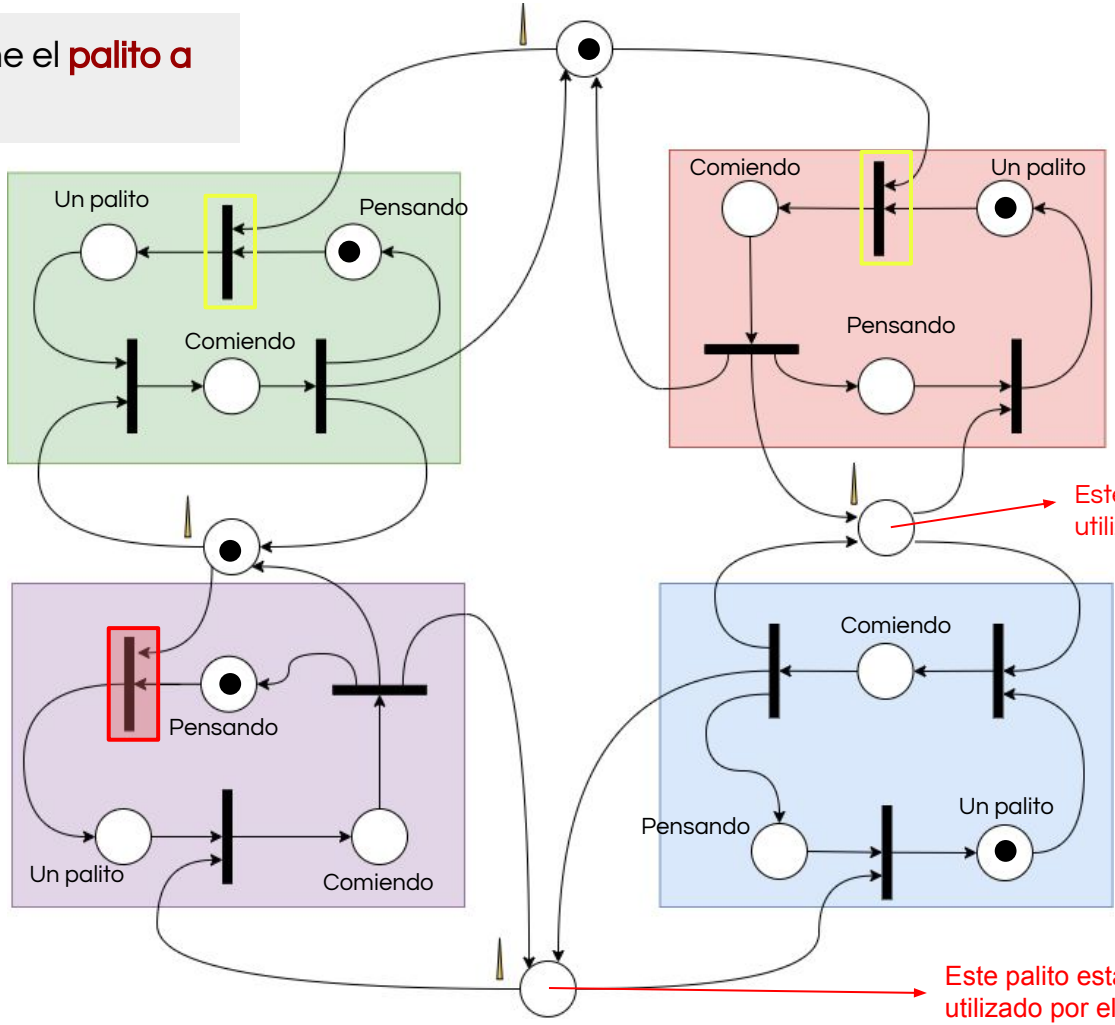


El **filosofo azul** no puede hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** agarra el **palito a su izquierda**

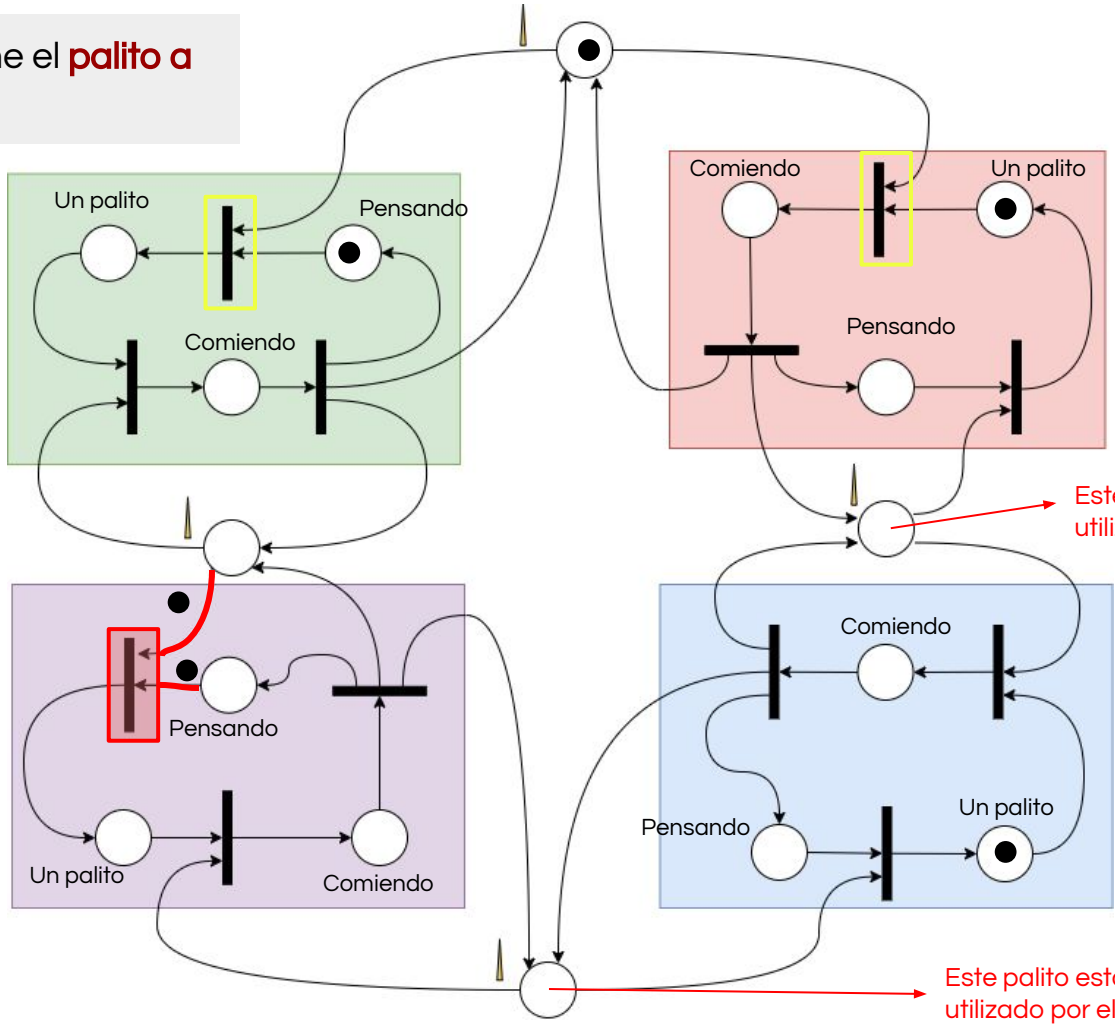


El **filosofo azul** no puede hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** agarra el **palito a su izquierda**

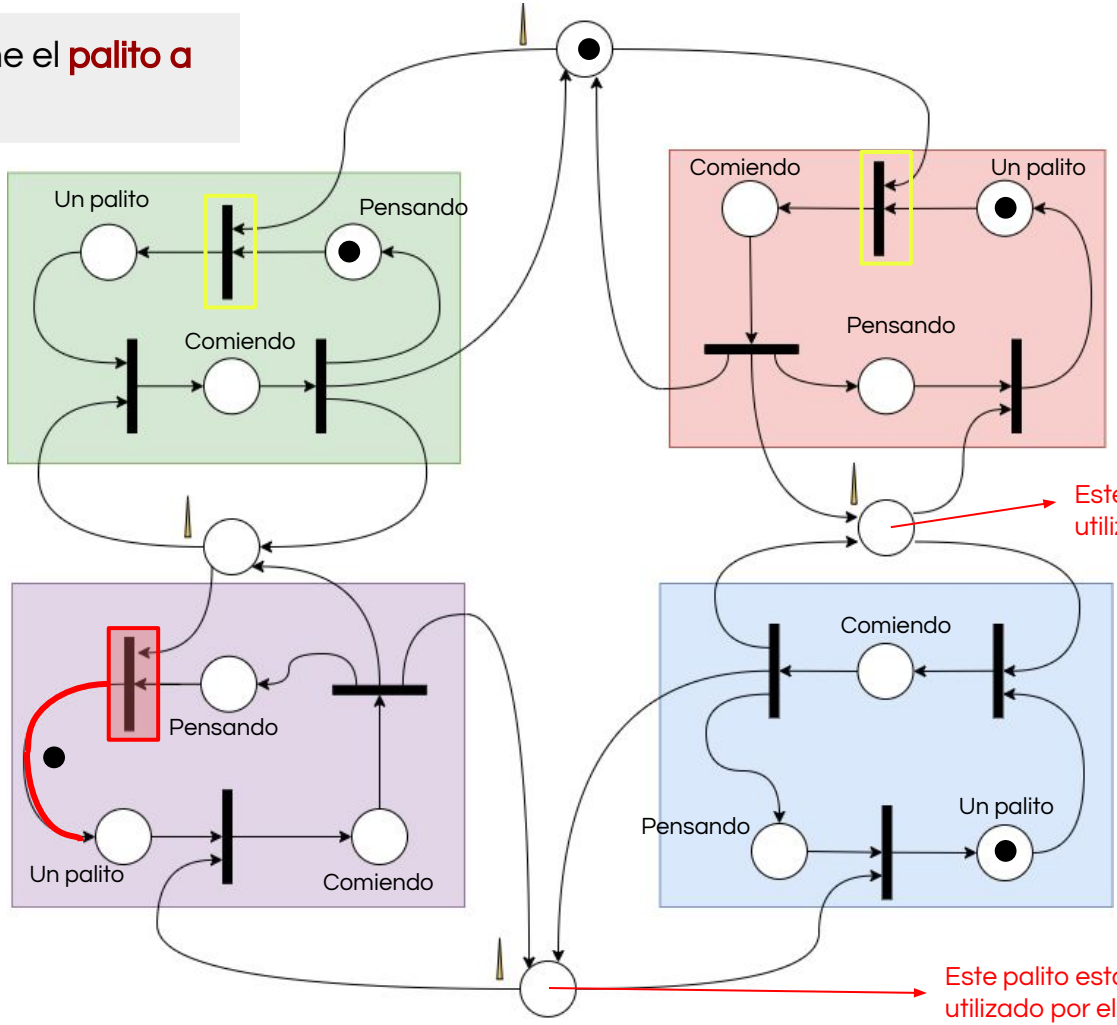


El **filosofo azul** no puede hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** agarra el **palito a su izquierda**

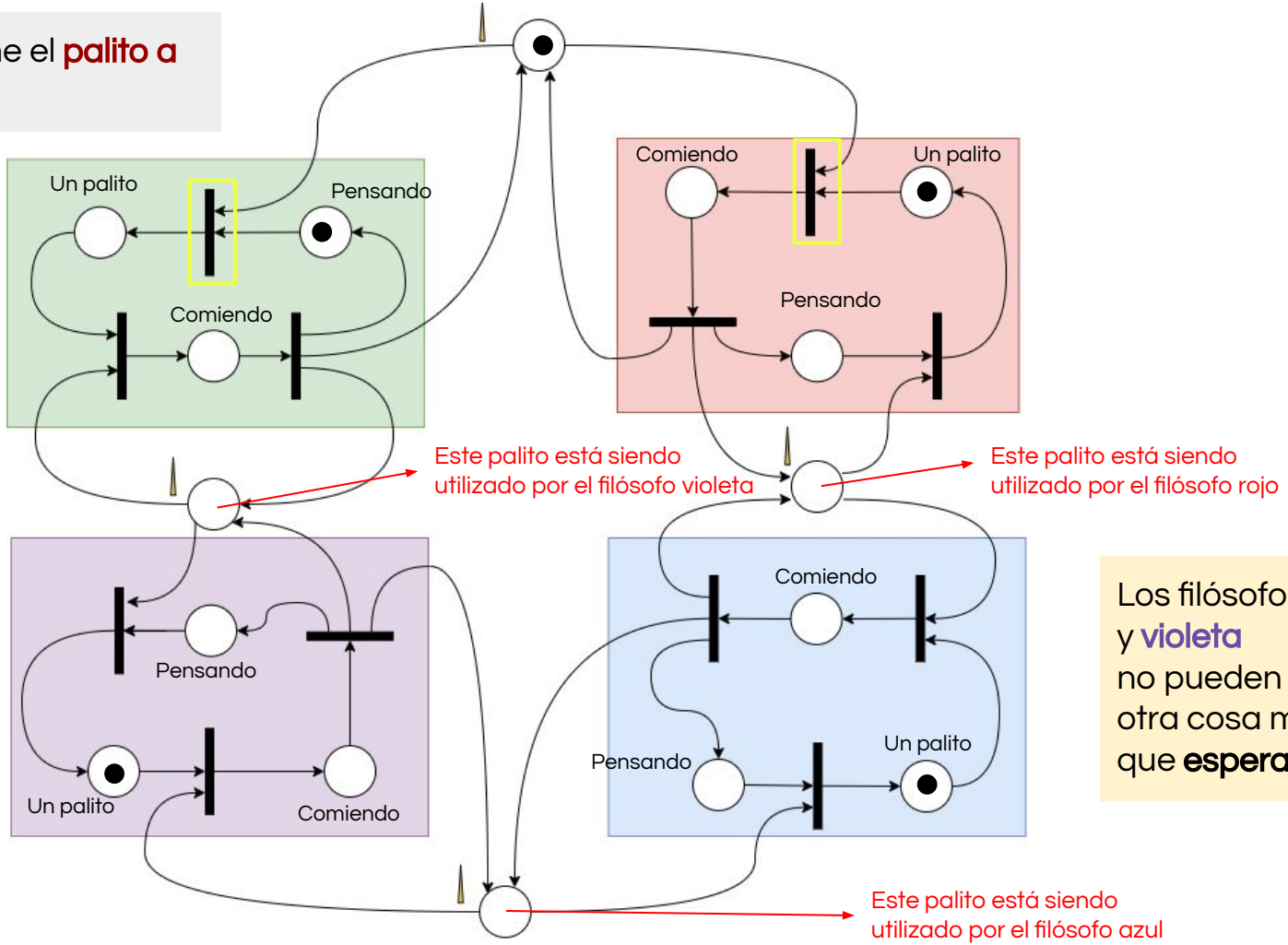


El **filosofo azul** no puede hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** tiene el **palito a su izquierda**



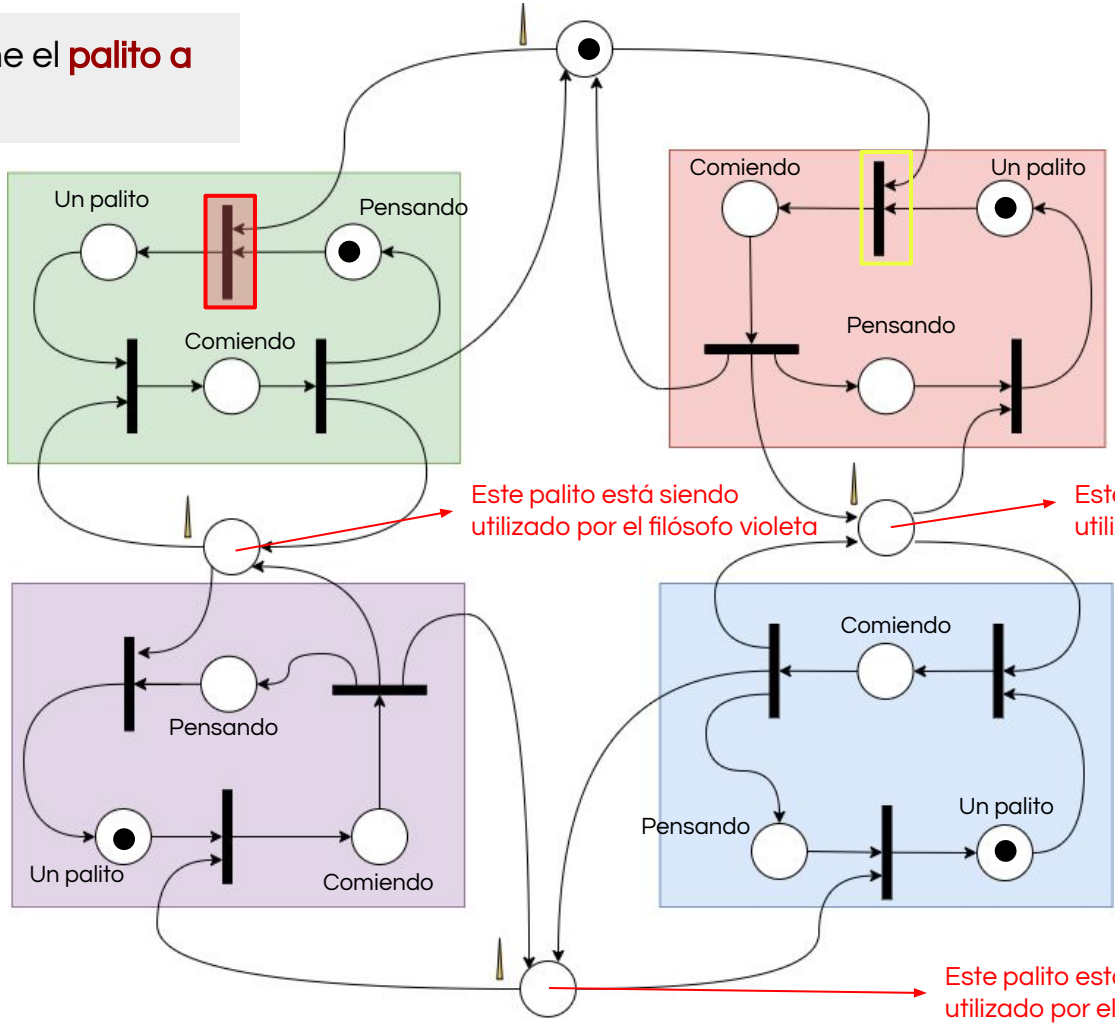
Los filósofos **azul** y **violeta** no pueden hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** tiene el **palito a su izquierda**

El **filósofo verde** agarra el **palito a su izquierda**



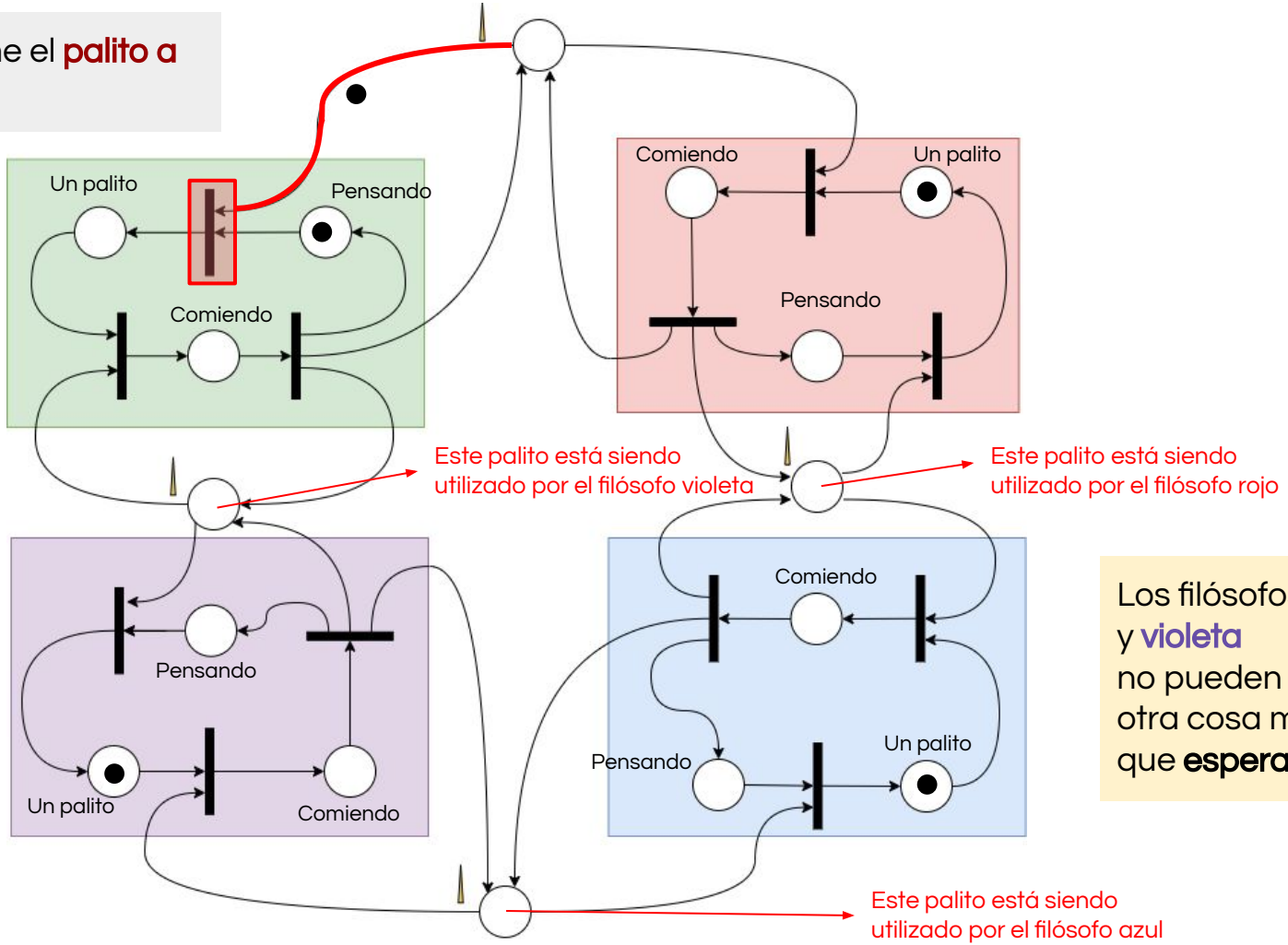
Los filósofos **azul** y **violeta** no pueden hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** tiene el **palito a su izquierda**

El **filósofo verde** agarra el **palito a su izquierda**



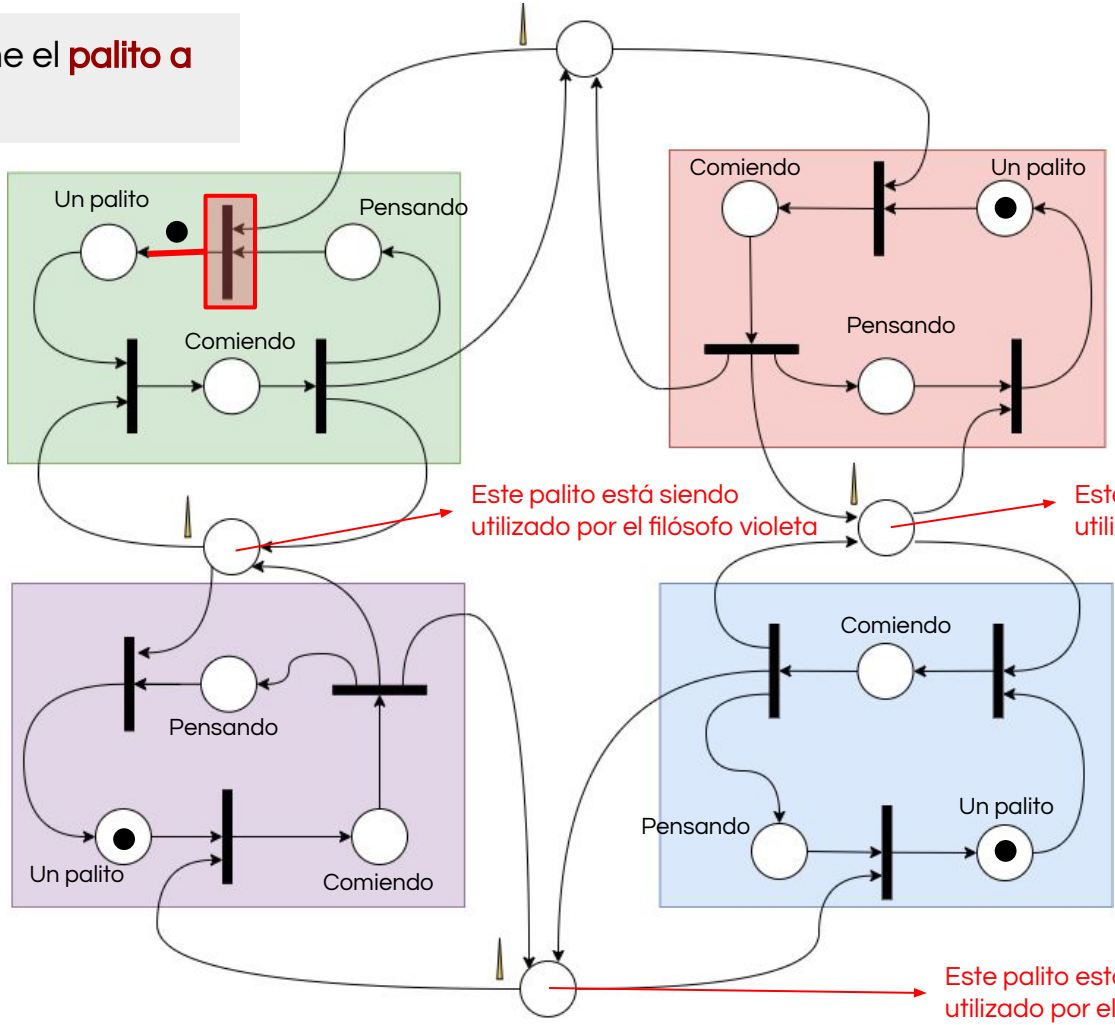
Los filósofos **azul** y **violeta** no pueden hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** tiene el **palito a su izquierda**

El **filósofo verde** agarra el **palito a su izquierda**



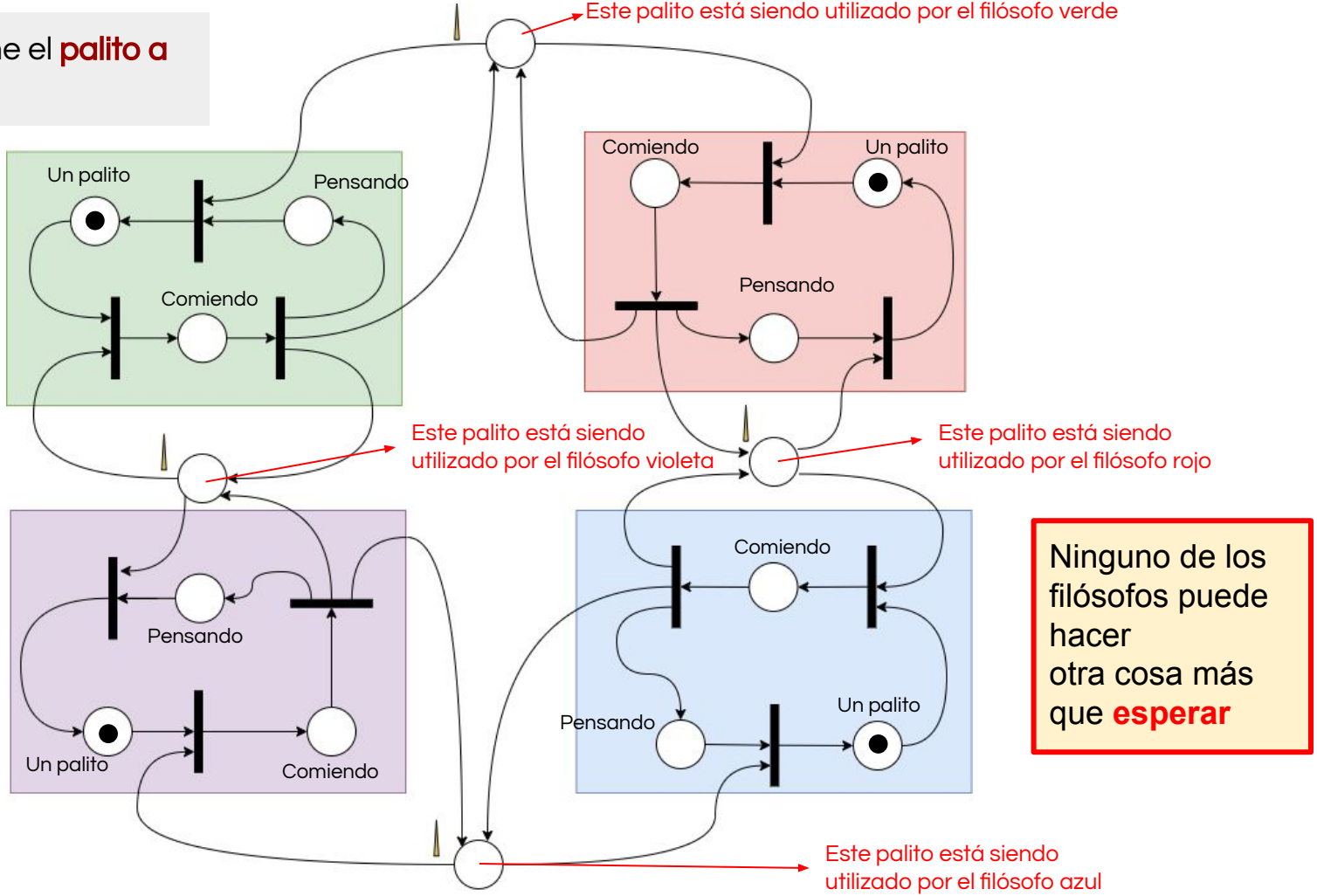
Los filósofos **azul** y **violeta** no pueden hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** tiene el **palito a su izquierda**

El **filósofo verde** tiene el **palito a su izquierda**



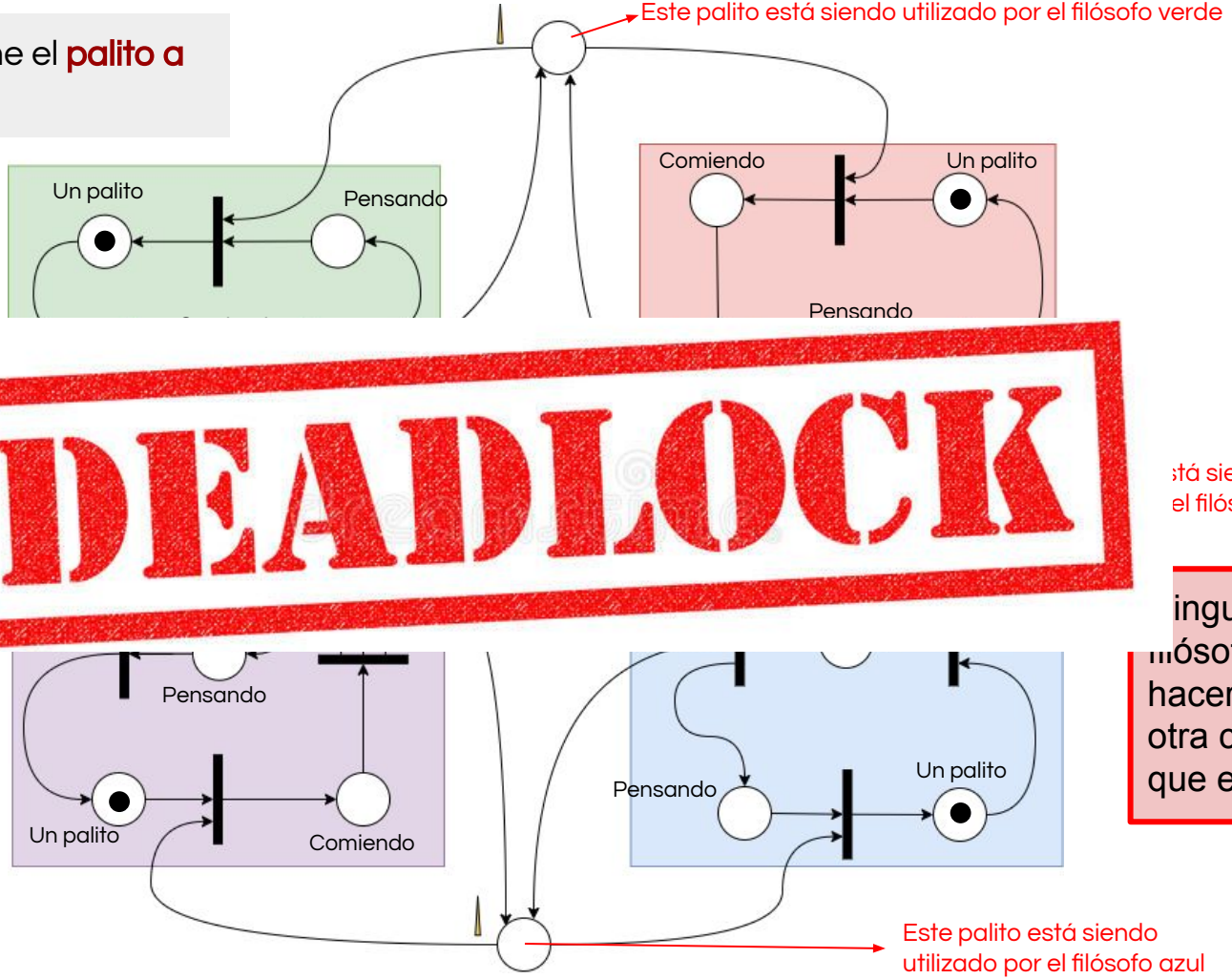
Ninguno de los filósofos puede hacer otra cosa más que **esperar**

El **filósofo rojo** tiene el **palito a su izquierda**

El **filósofo azul** tiene el **palito a su izquierda**

El **filósofo violeta** tiene el **palito a su izquierda**

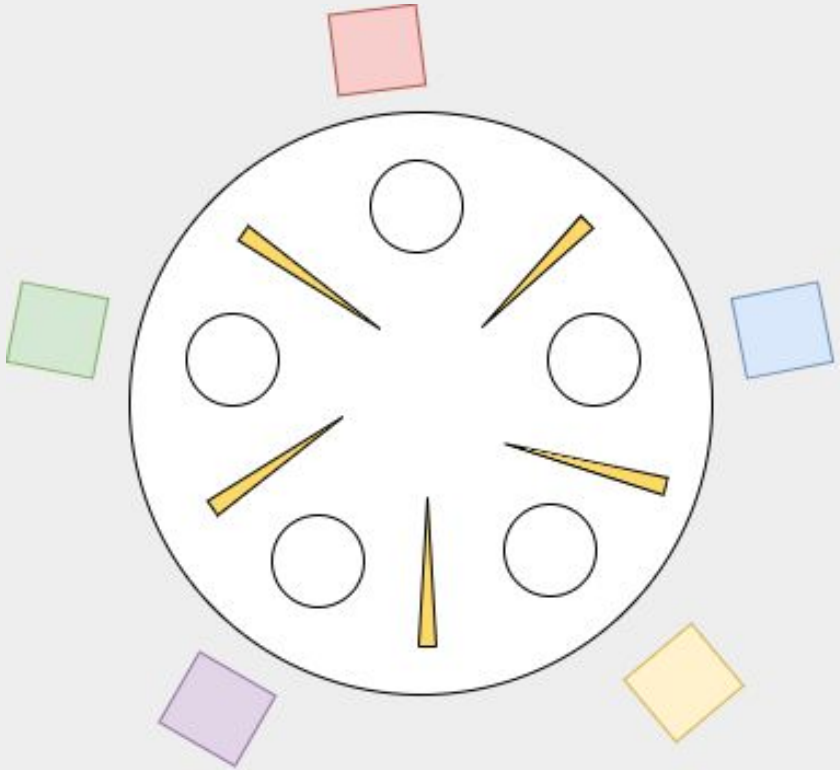
El **filósofo verde** tiene el **palito a su izquierda**

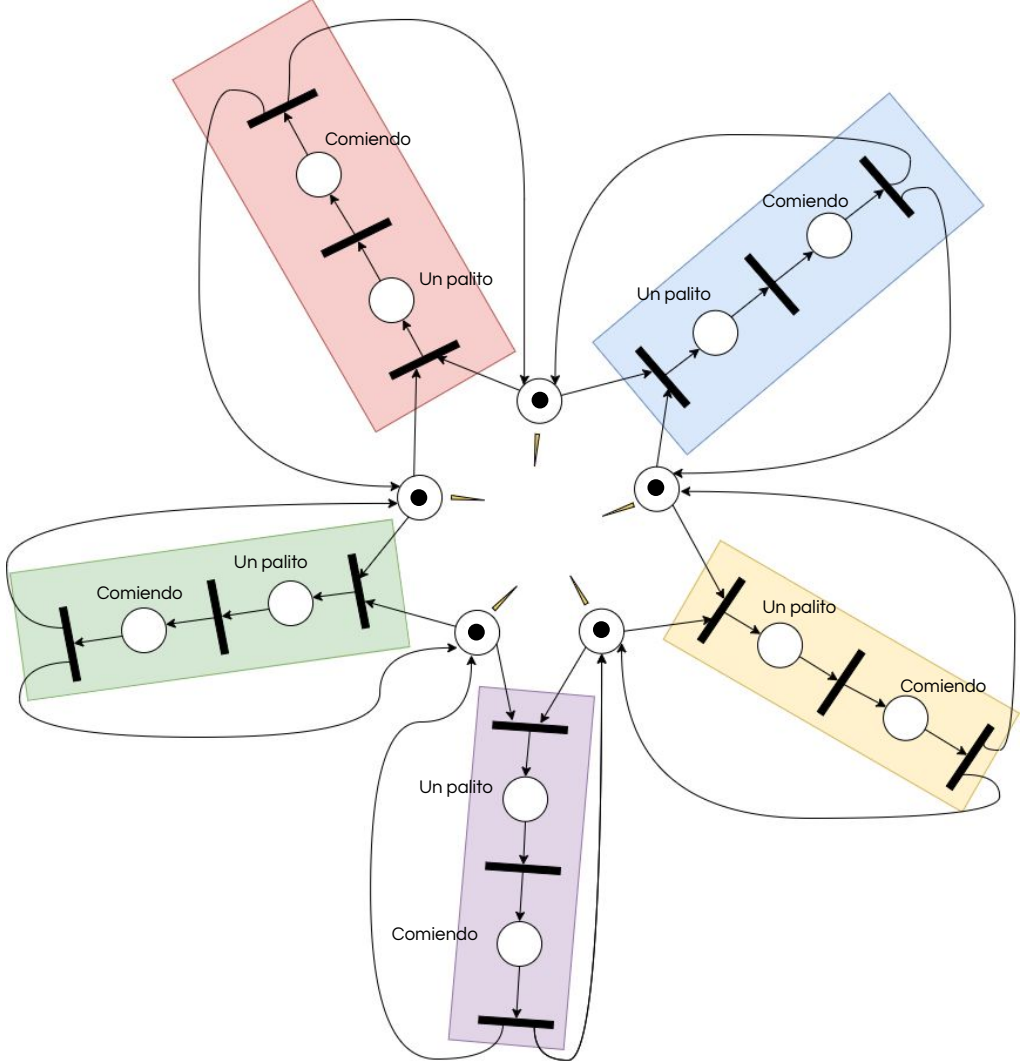


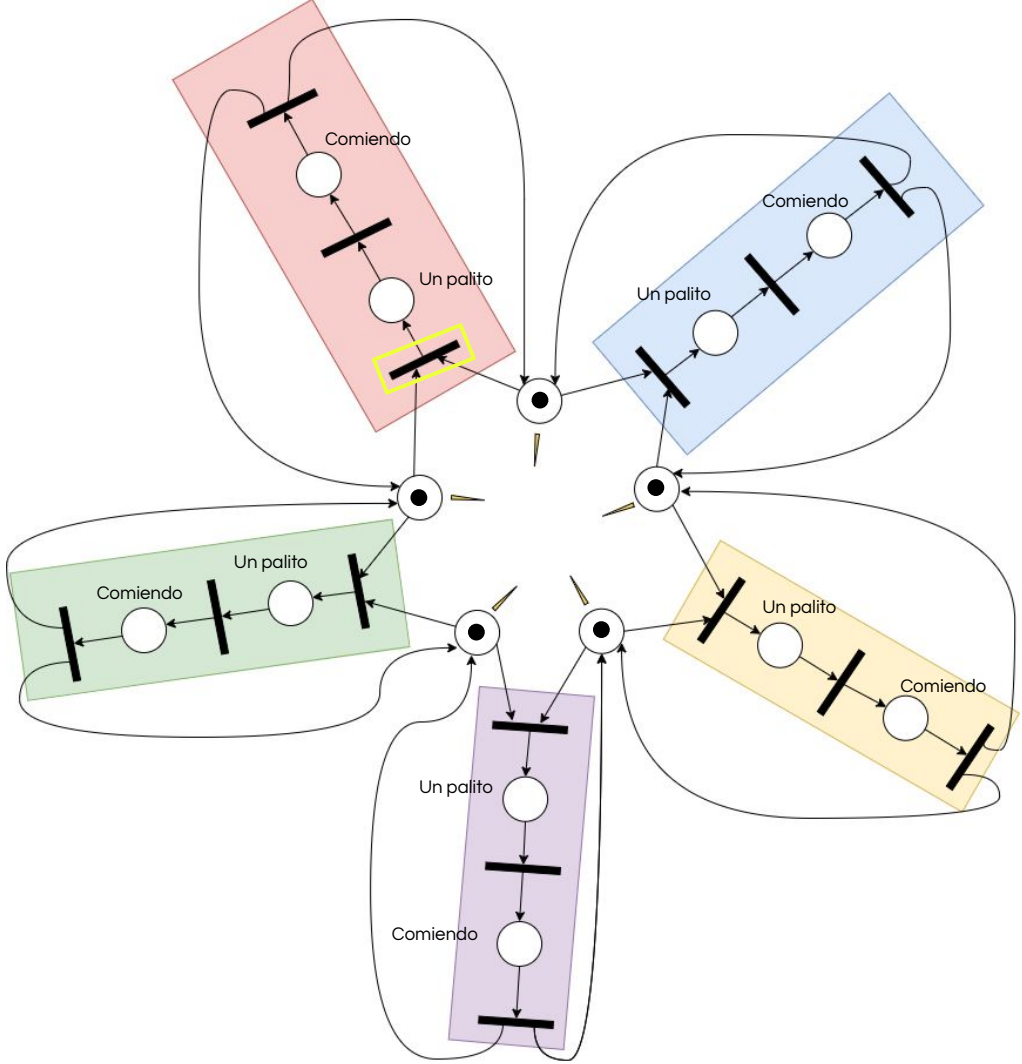
tá siendo
el filósofo rojo

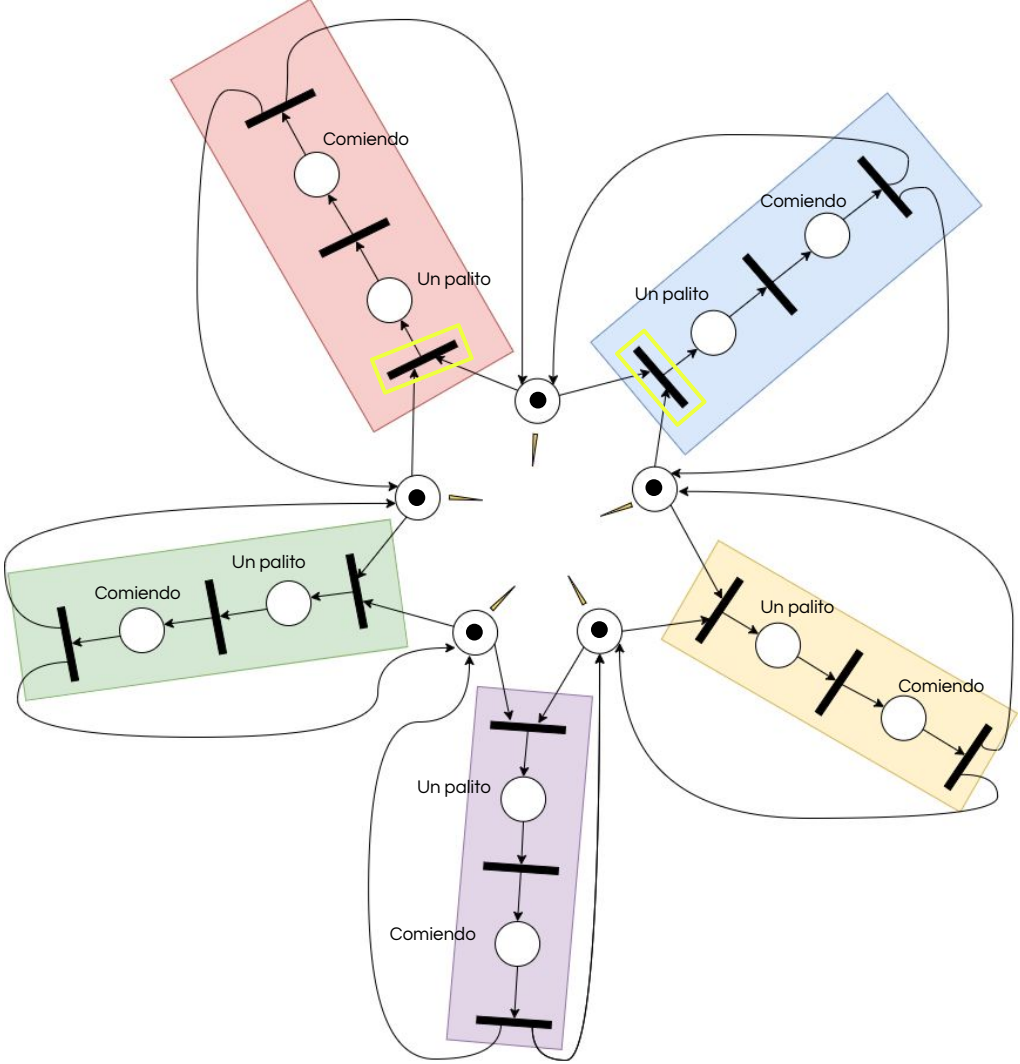
inguno de los
lósofos puede
hacer
otra cosa más
que esperar

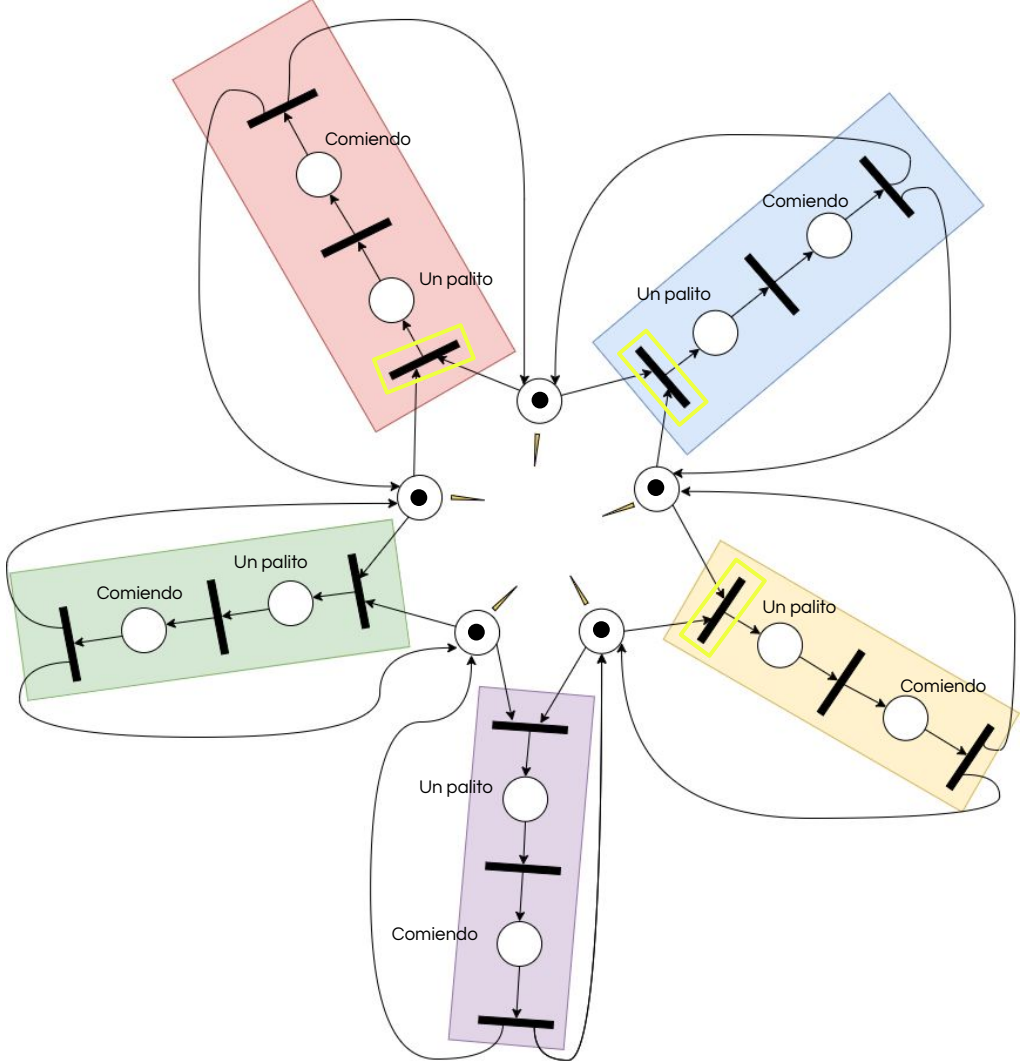
5 filósofos - sin deadlock

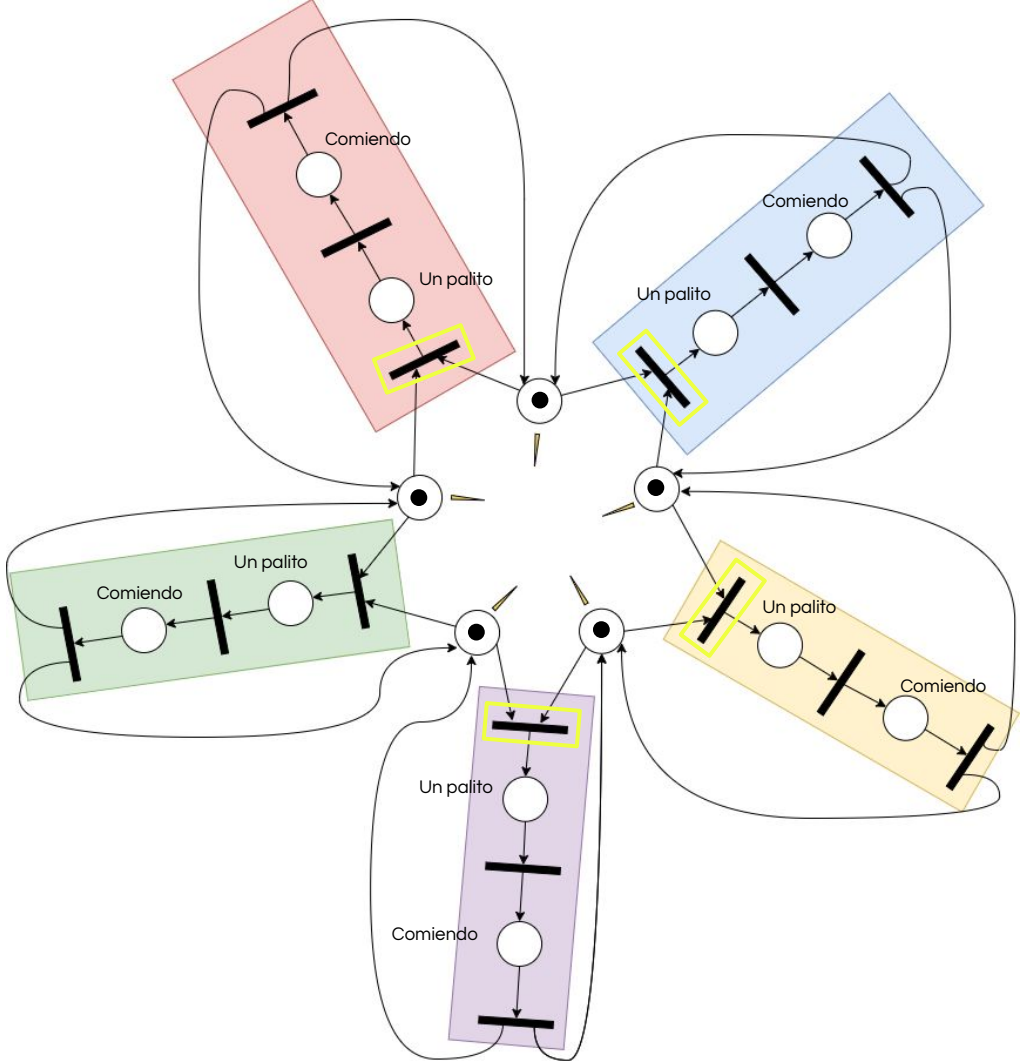


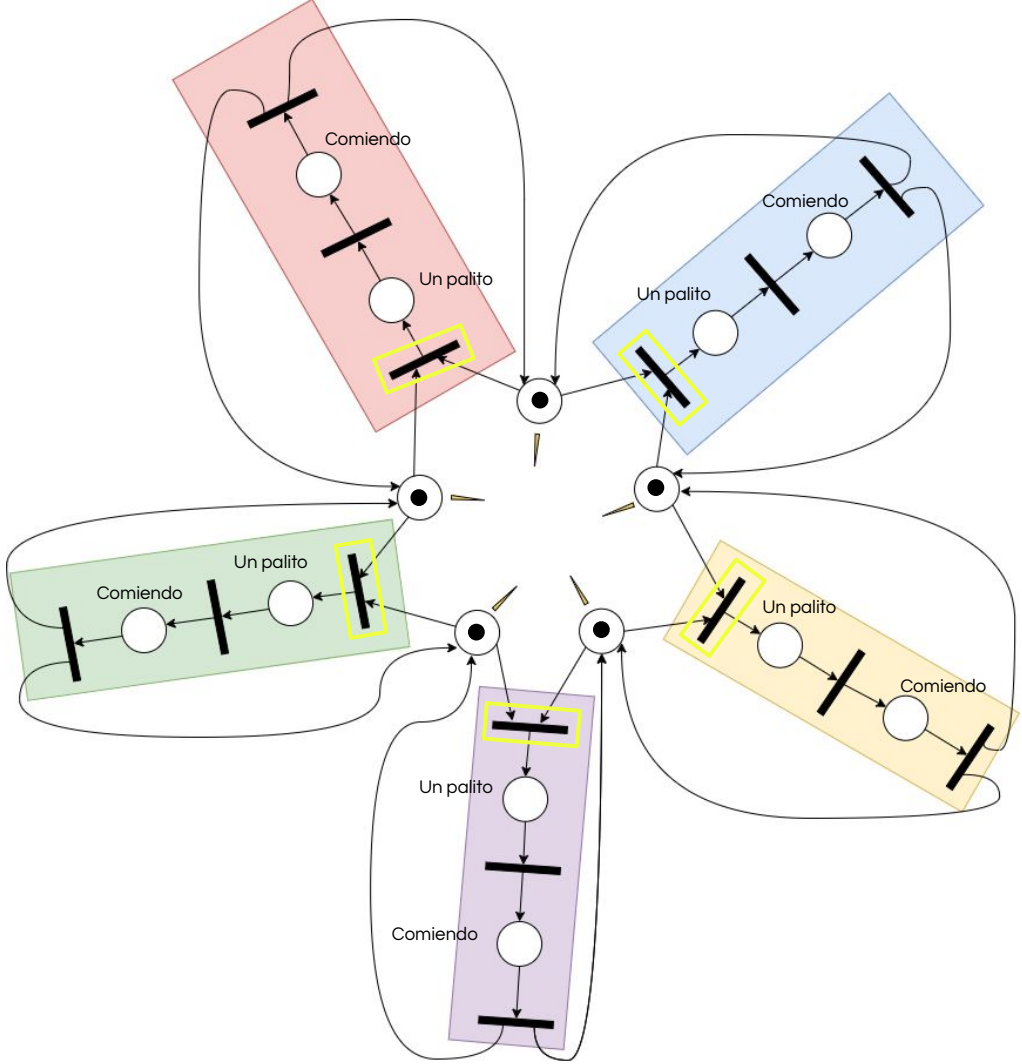

Comiendo
Un palito
Comiendo
Un palito
Un palito
Comiendo
Un palito
Comiendo
Un palito
Comiendo


Comiendo
Un palito
Comiendo
Un palito
Un palito
Comiendo
Un palito
Comiendo
Un palito
Comiendo

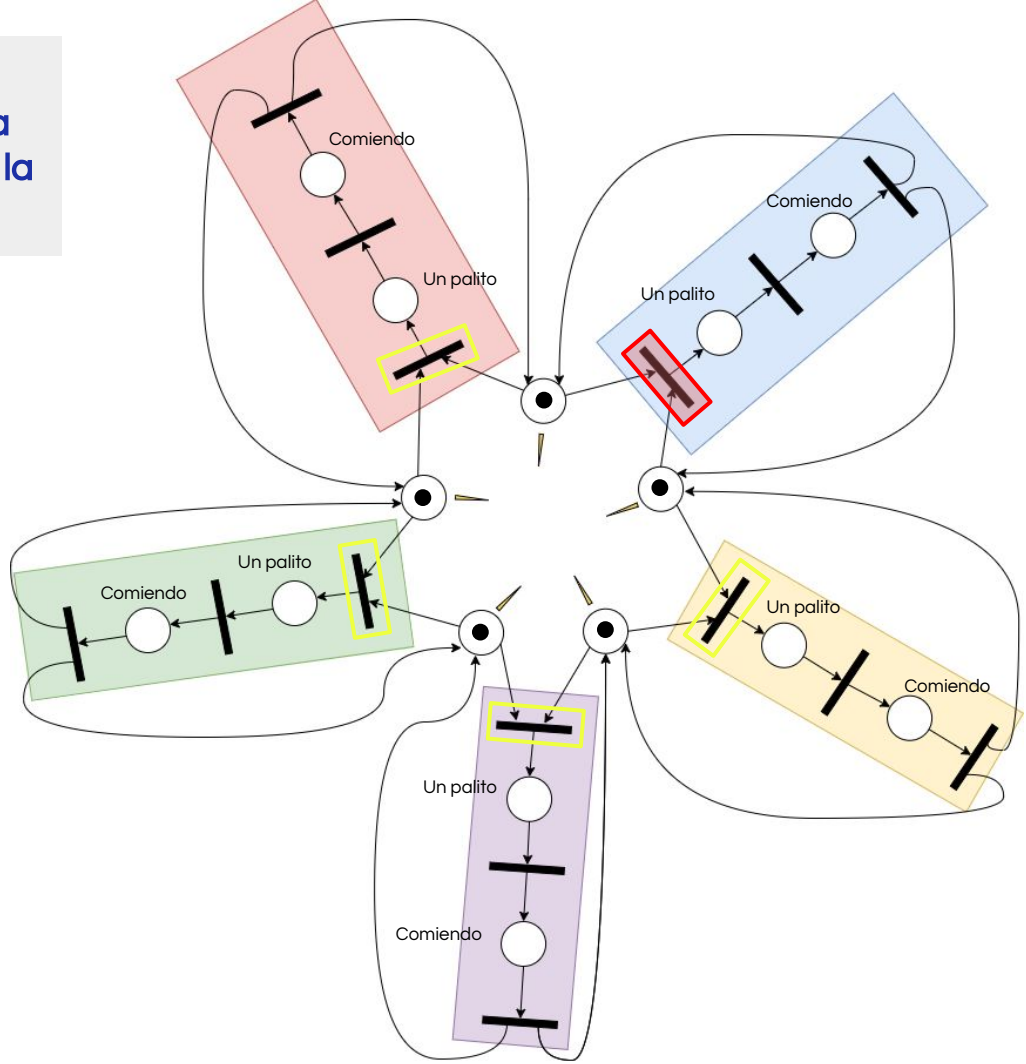

Comiendo
Un palito
Comiendo
Un palito
Un palito
Comiendo
Comiendo
Un palito
Un palito
Comiendo


Comiendo
Un palito
Comiendo
Un palito
Un palito
Comiendo
Un palito
Comiendo
Un palito
Comiendo

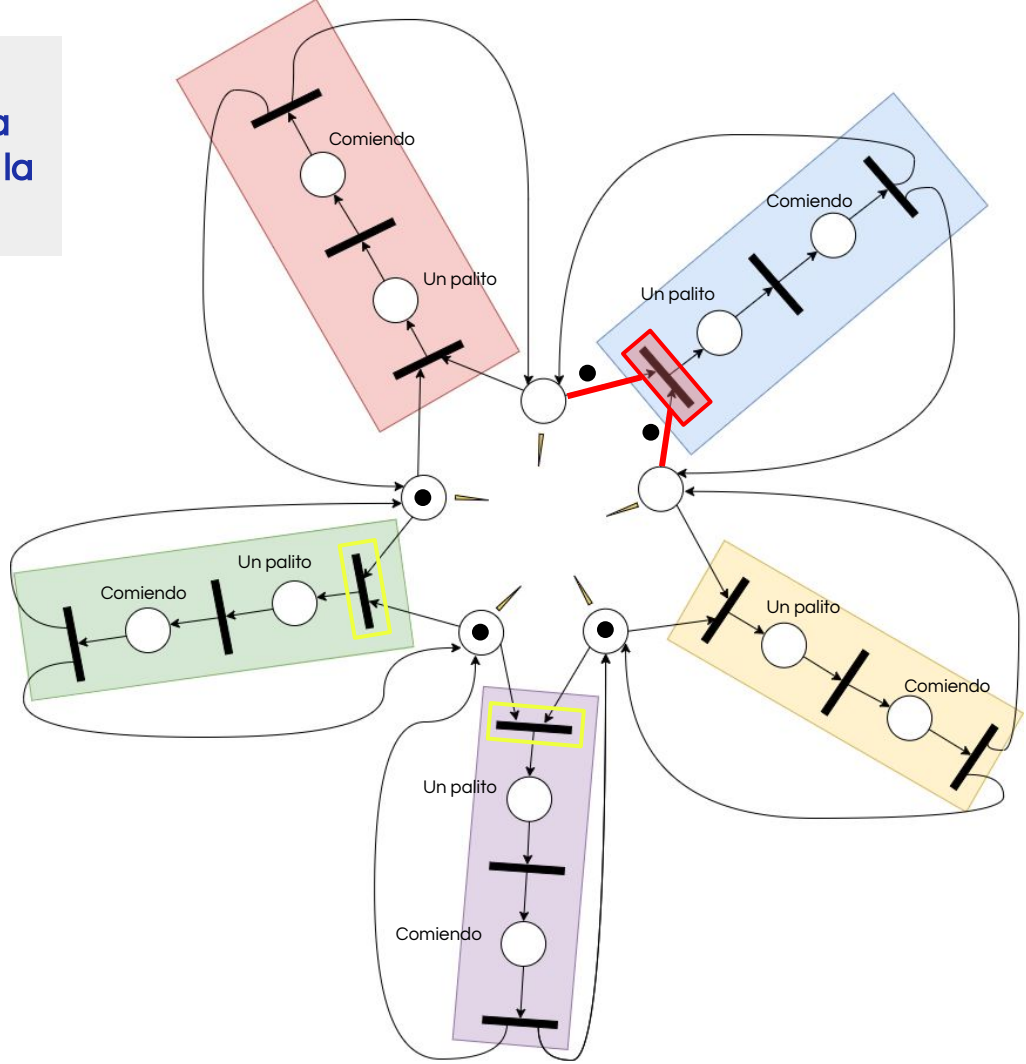

Comiendo
Un palito
Comiendo
Un palito
Un palito
Comiendo
Comiendo
Un palito
Un palito
Comiendo


Comiendo
Un palito
Comiendo
Un palito
Un palito
Comiendo
Comiendo
Un palito
Un palito
Comiendo

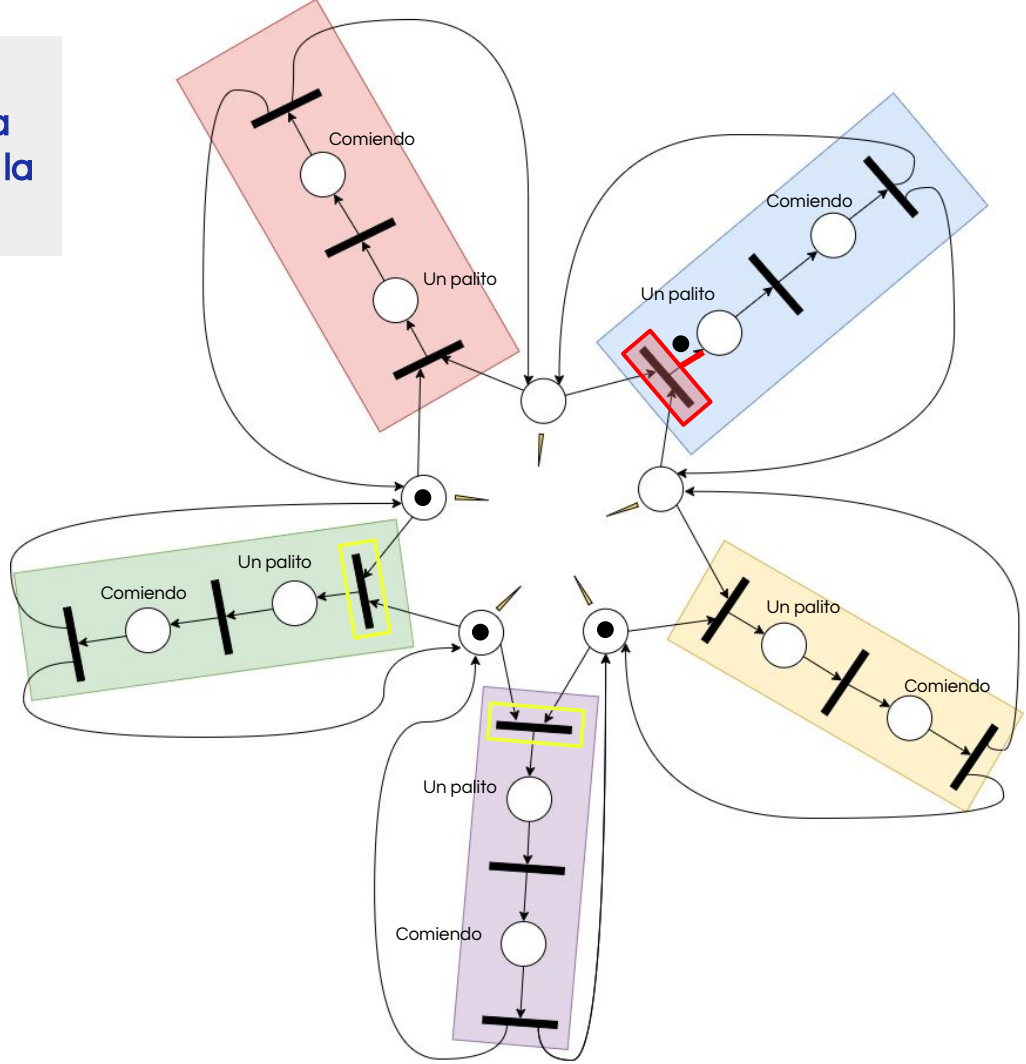
El **filósofo azul** agarra el **palito a su izquierda y seguidamente el de la derecha**



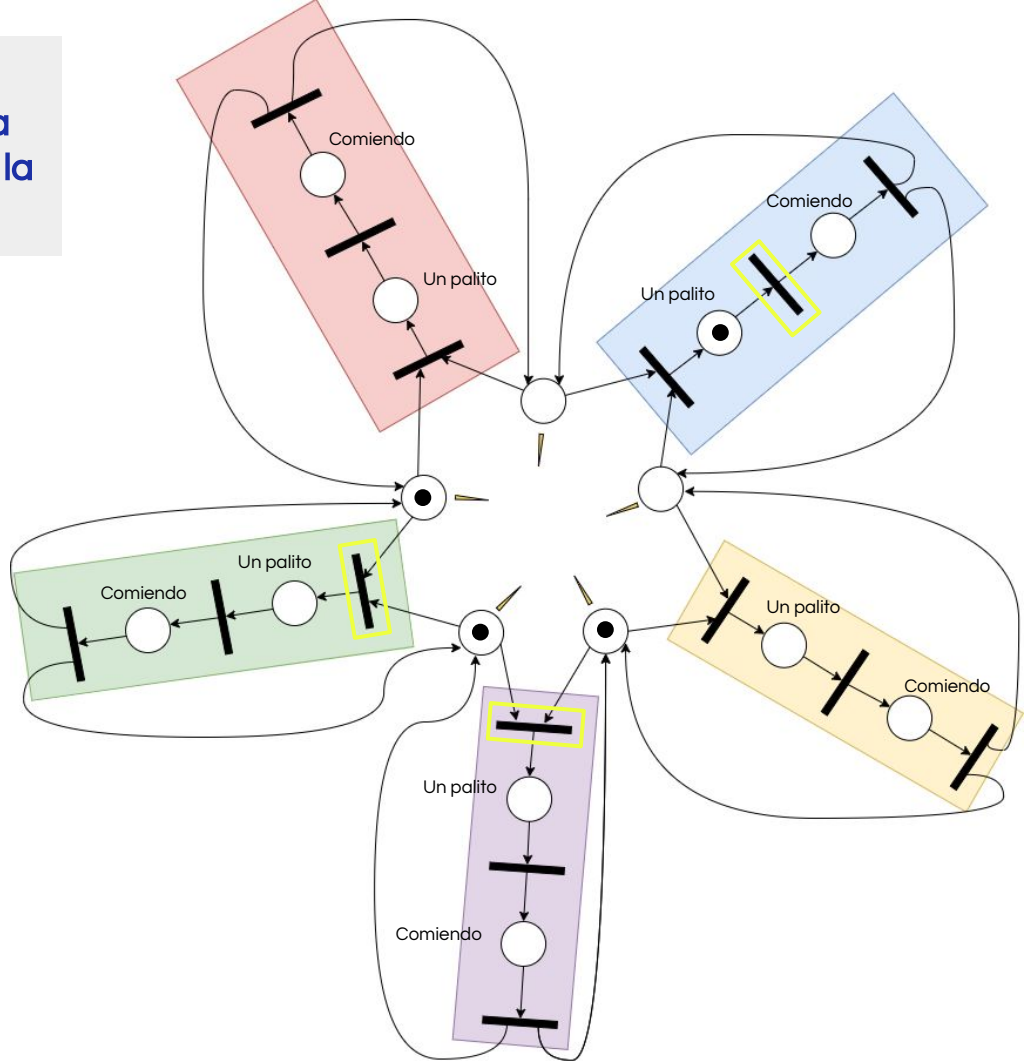
El **filósofo azul** agarra el **palito a su izquierda y seguidamente el de la derecha**



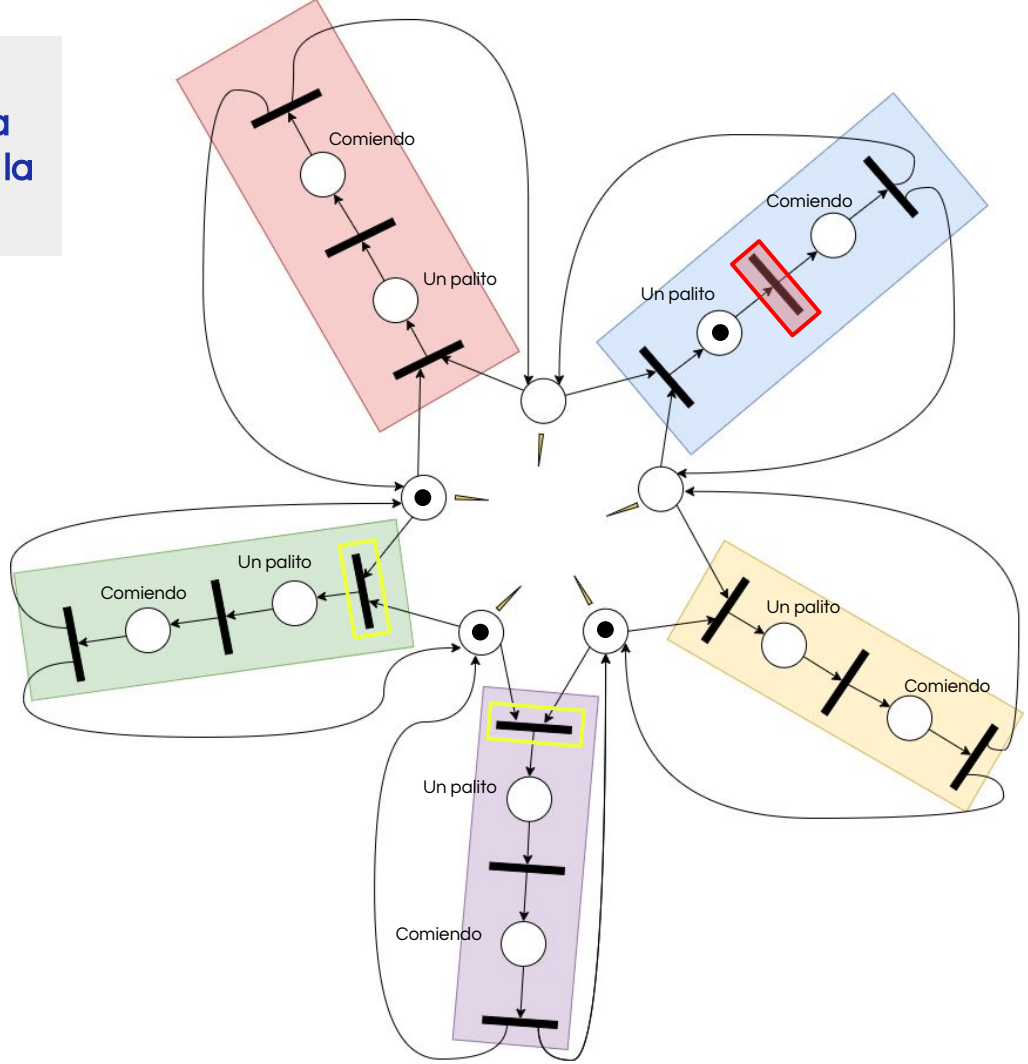
El **filósofo azul** agarra el **palito a su izquierda y seguidamente el de la derecha**



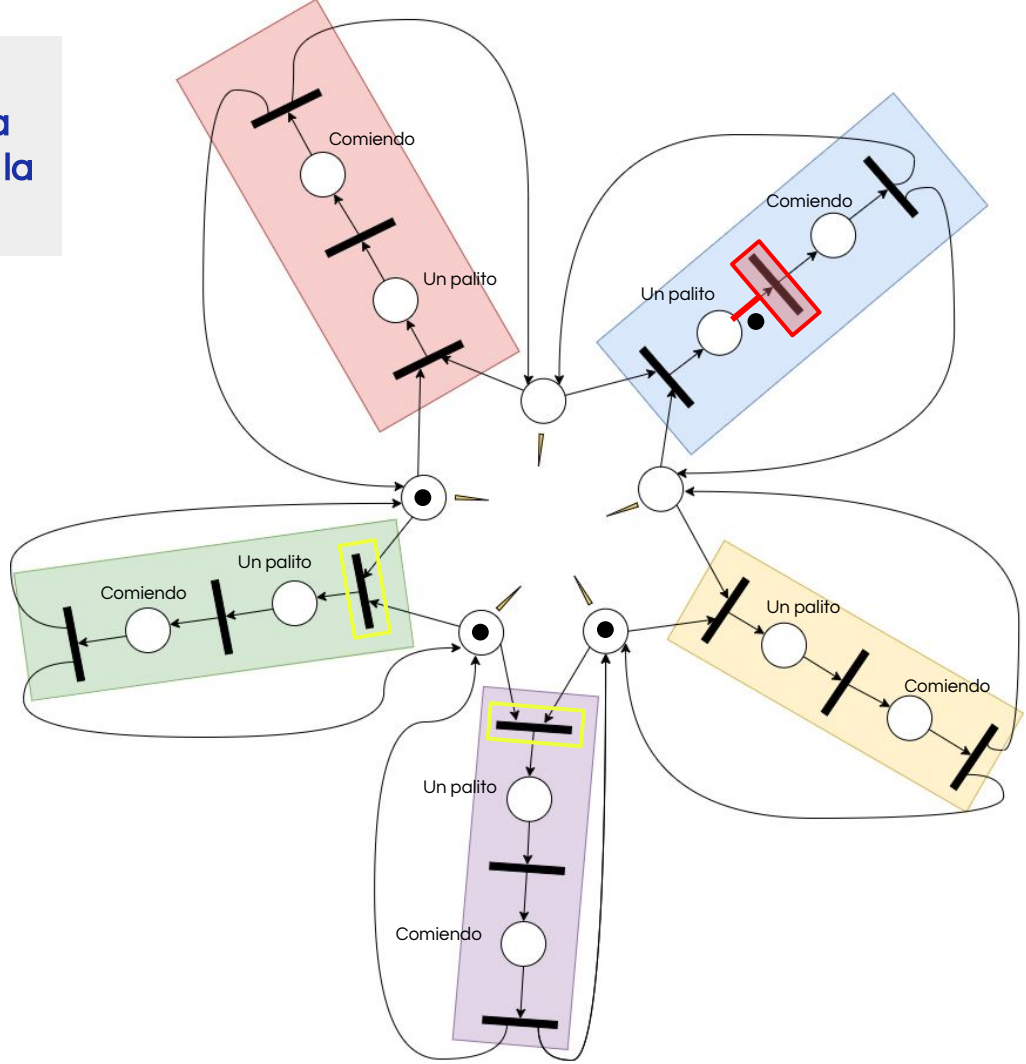
El **filósofo azul** agarra el **palito a su izquierda y seguidamente el de la derecha**



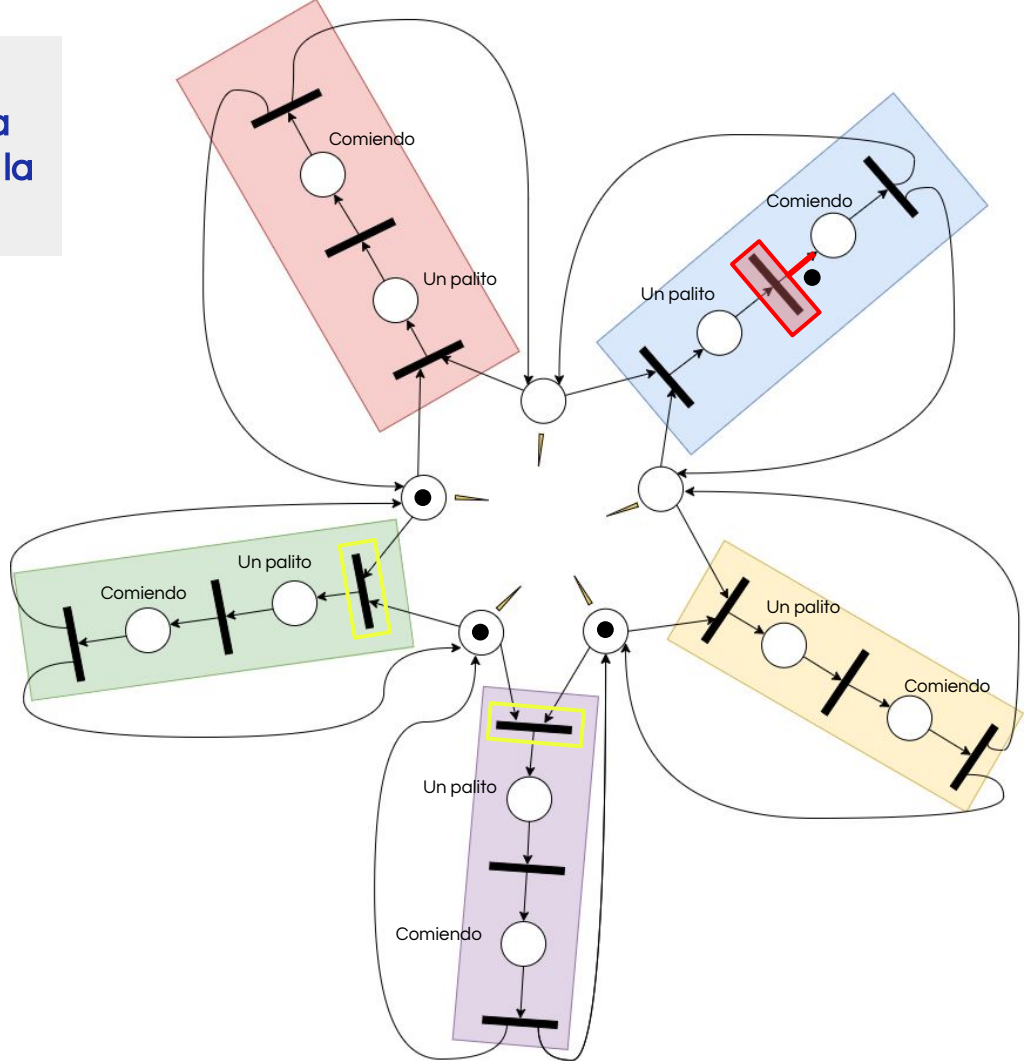
El **filósofo azul** agarra el **palito a su izquierda y seguidamente el de la derecha**



El **filósofo azul** agarra el **palito a su izquierda y seguidamente el de la derecha**

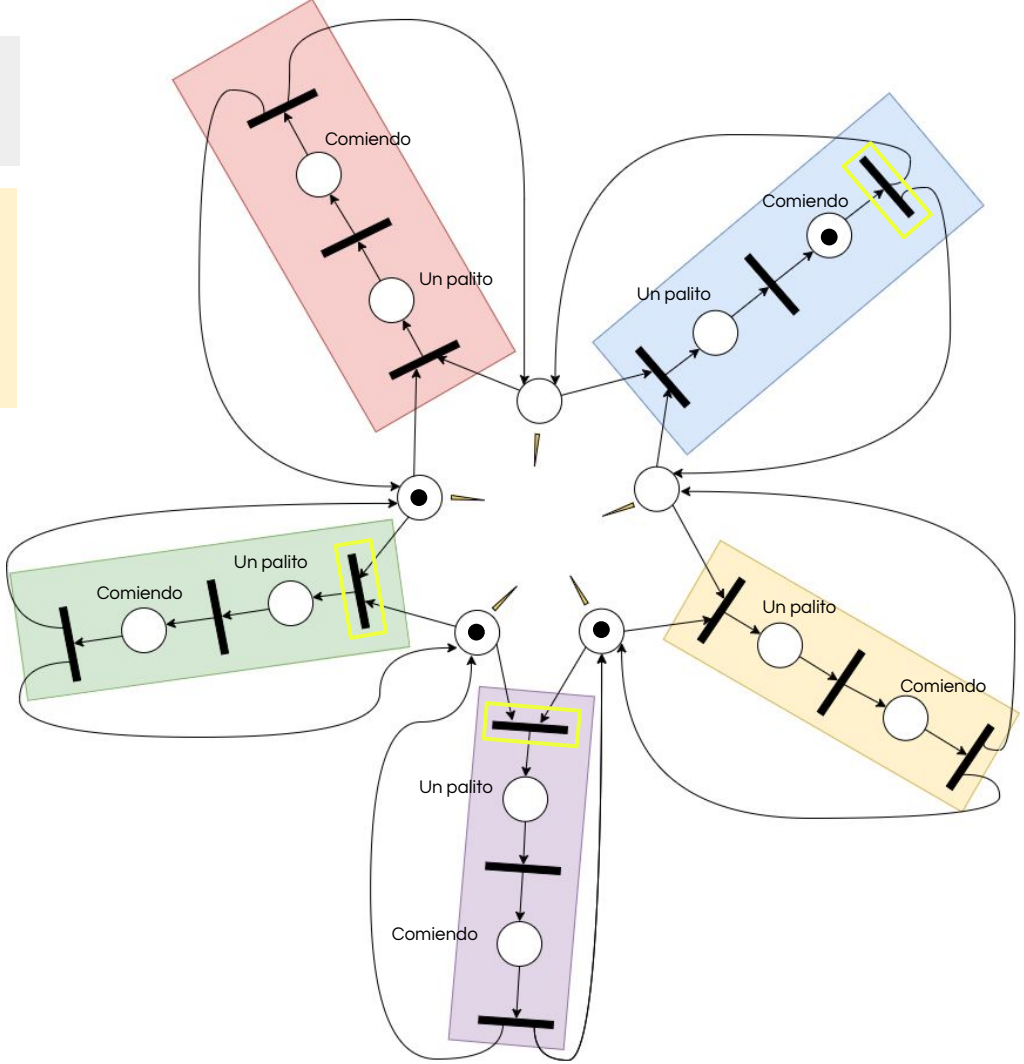


El **filósofo azul** agarra el **palito a su izquierda y seguidamente el de la derecha**



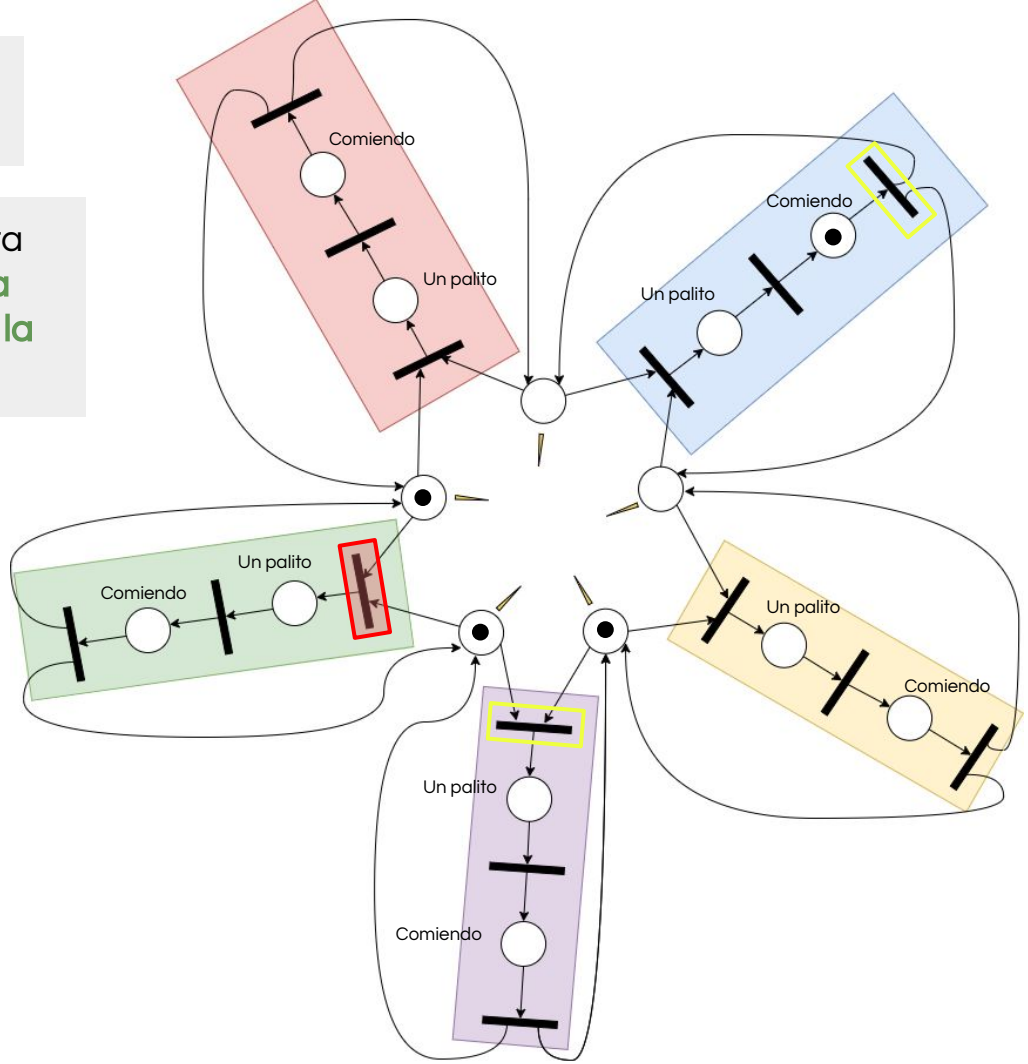
El **filósofo azul** está **comiendo**

Sólo el filósofo **verde** o el **violeta** **pueden comer** mientras el azul está comiendo



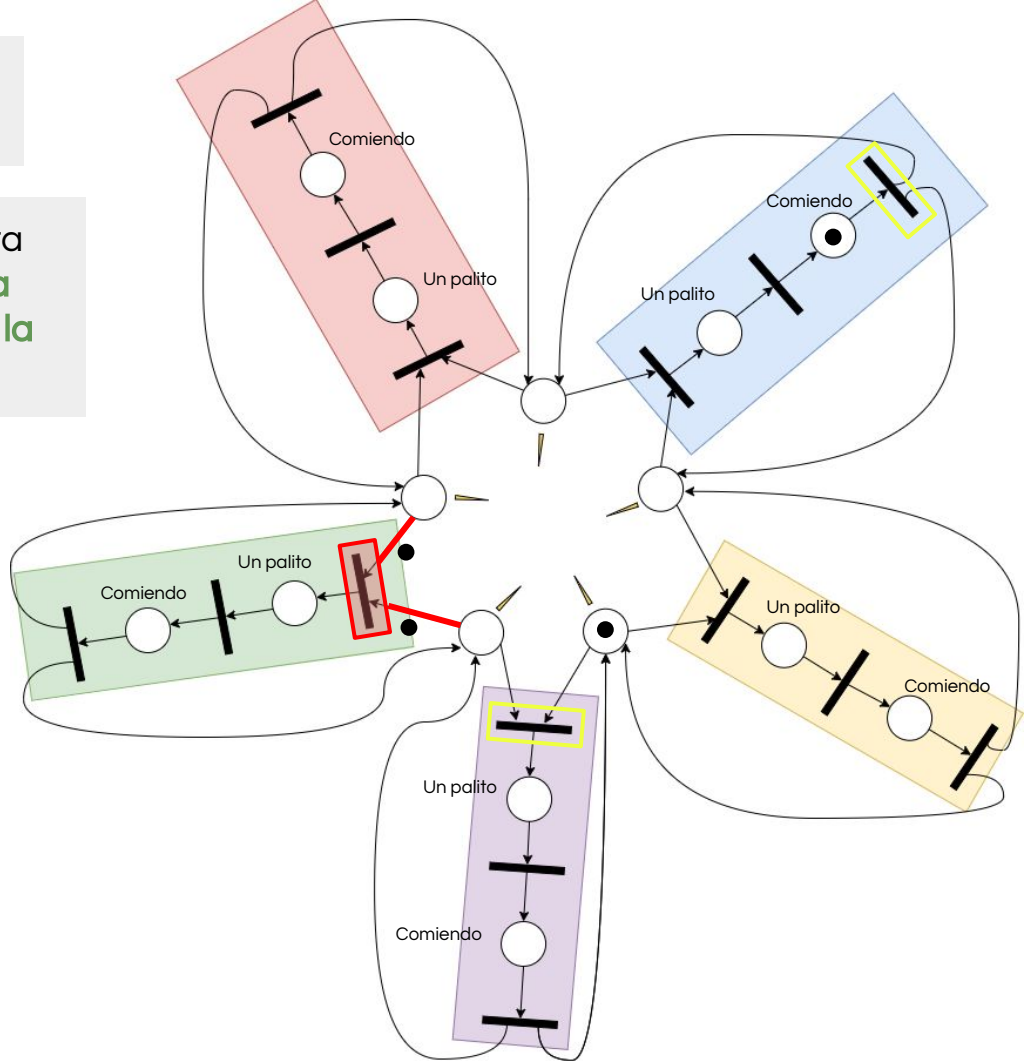
El **filósofo azul** está **comiendo**

El **filósofo verde** agarra el **palito a su izquierda** y **seguidamente** el **de la derecha**



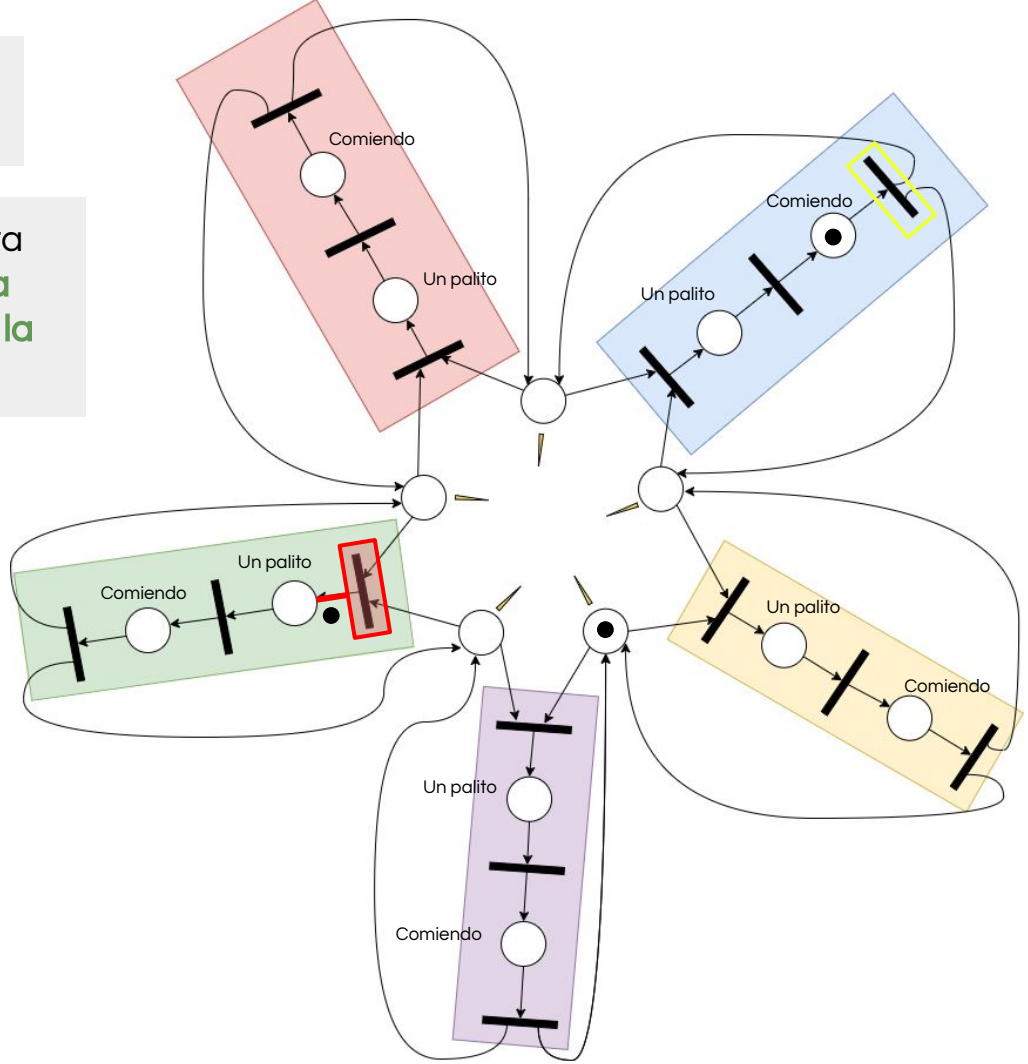
El **filósofo azul** está **comiendo**

El **filósofo verde** agarra el **palito a su izquierda** y **seguidamente** el **de la derecha**



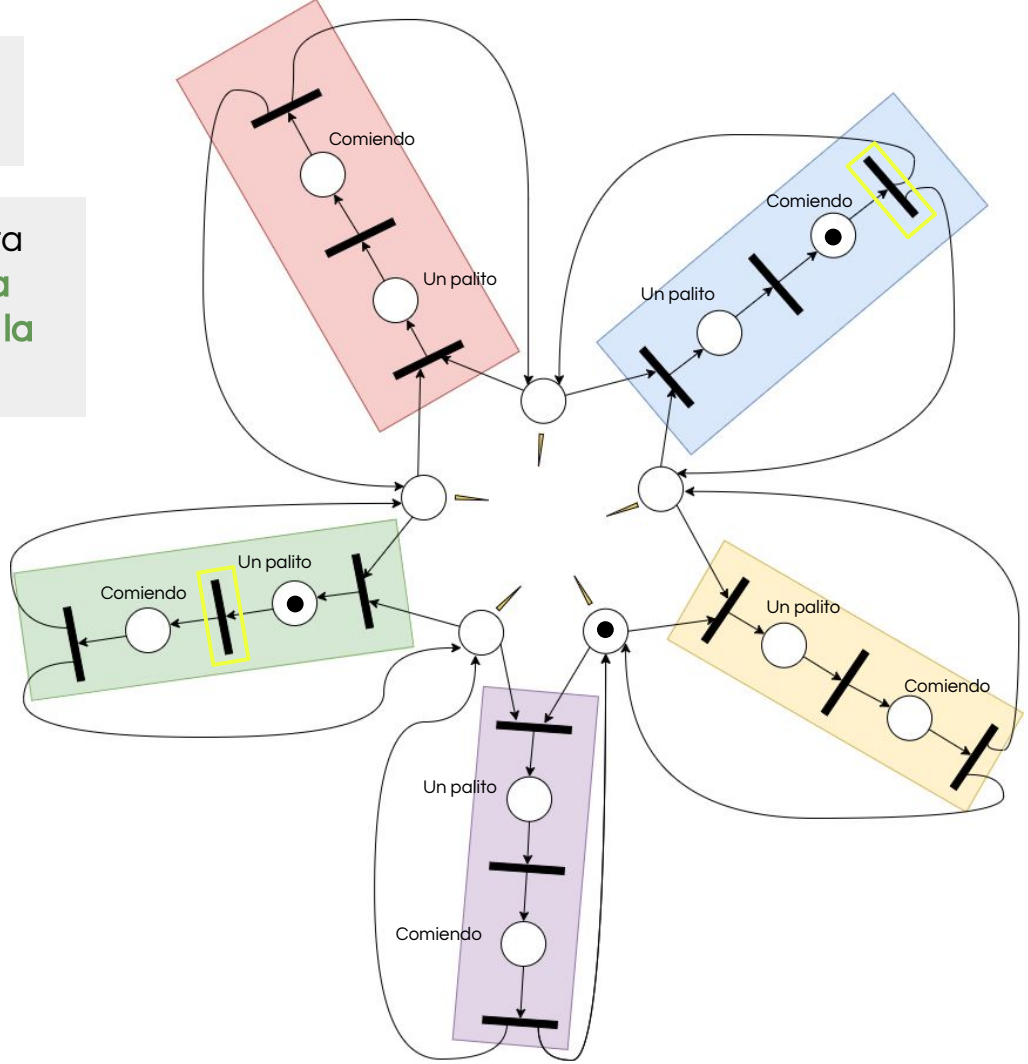
El **filósofo azul** está **comiendo**

El **filósofo verde** agarra el **palito a su izquierda** y **seguidamente** el **de la derecha**



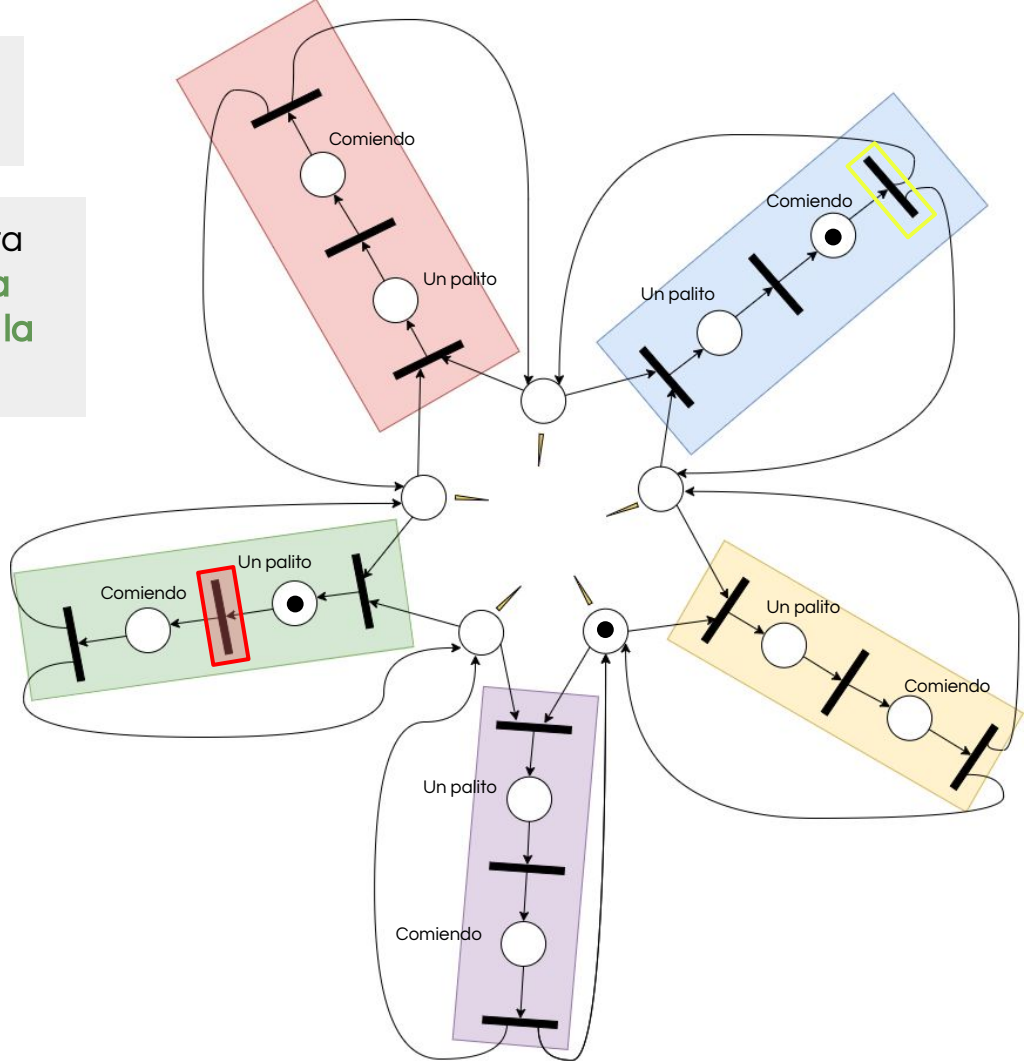
El **filósofo azul** está **comiendo**

El **filósofo verde** agarra el **palito a su izquierda** y **seguidamente** el **de la derecha**



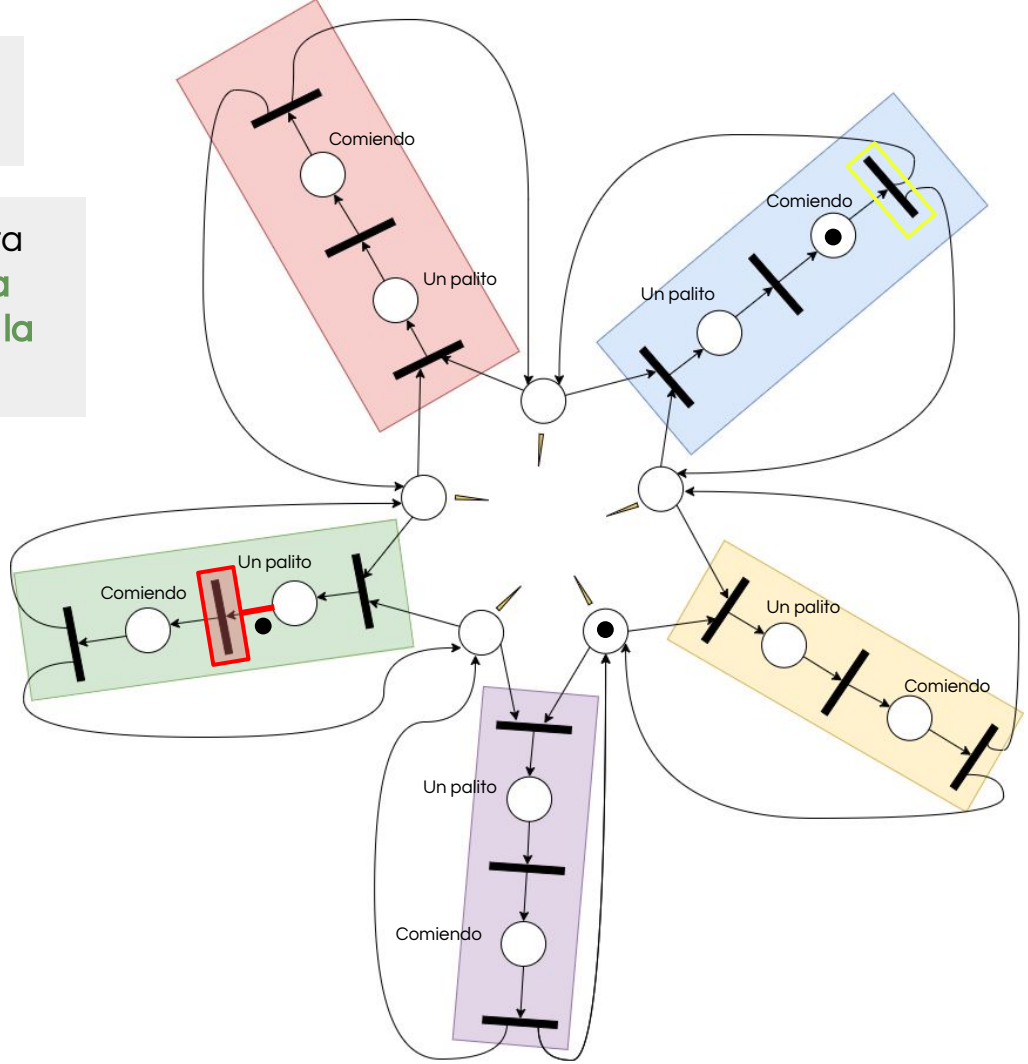
El **filósofo azul** está **comiendo**

El **filósofo verde** agarra el **palito a su izquierda** y **seguidamente** el **de la derecha**



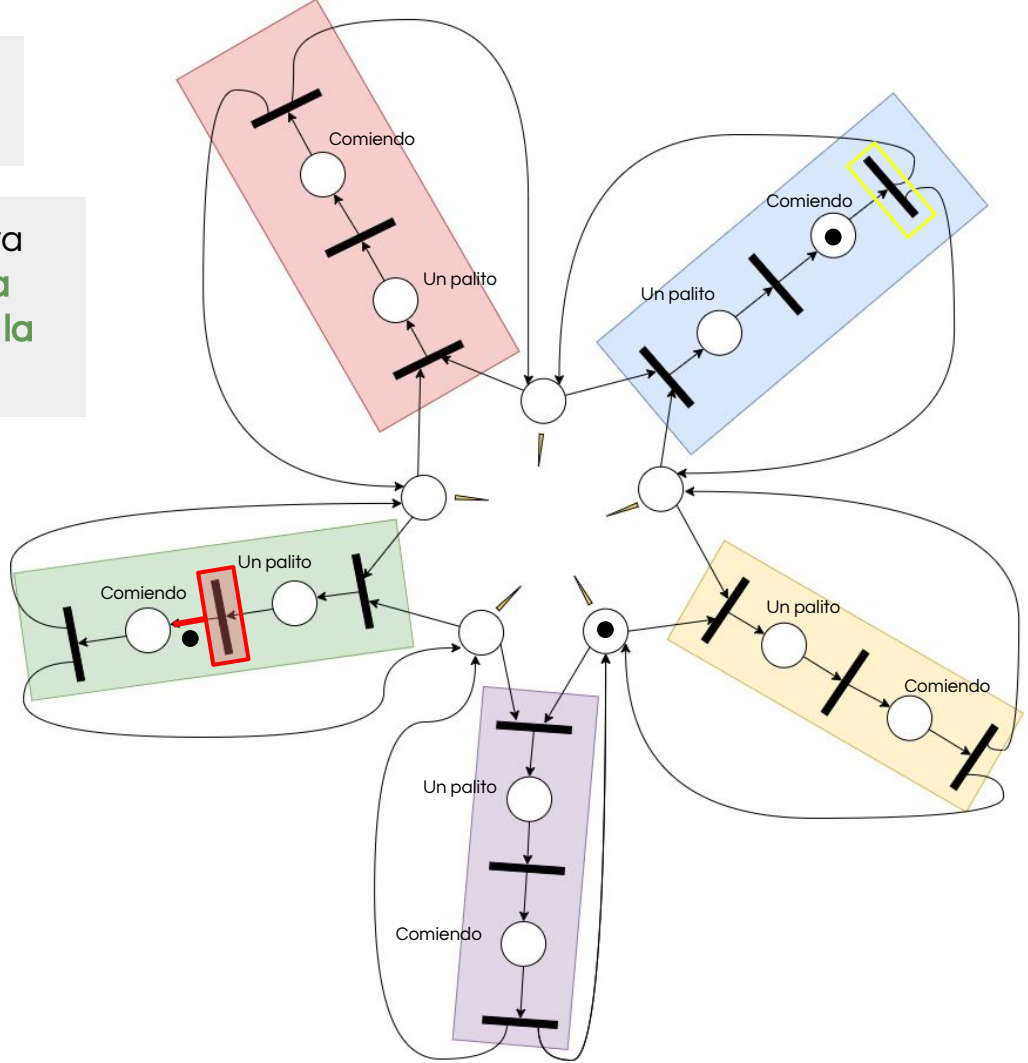
El **filósofo azul** está **comiendo**

El **filósofo verde** agarra el **palito a su izquierda** y **seguidamente** el de la derecha



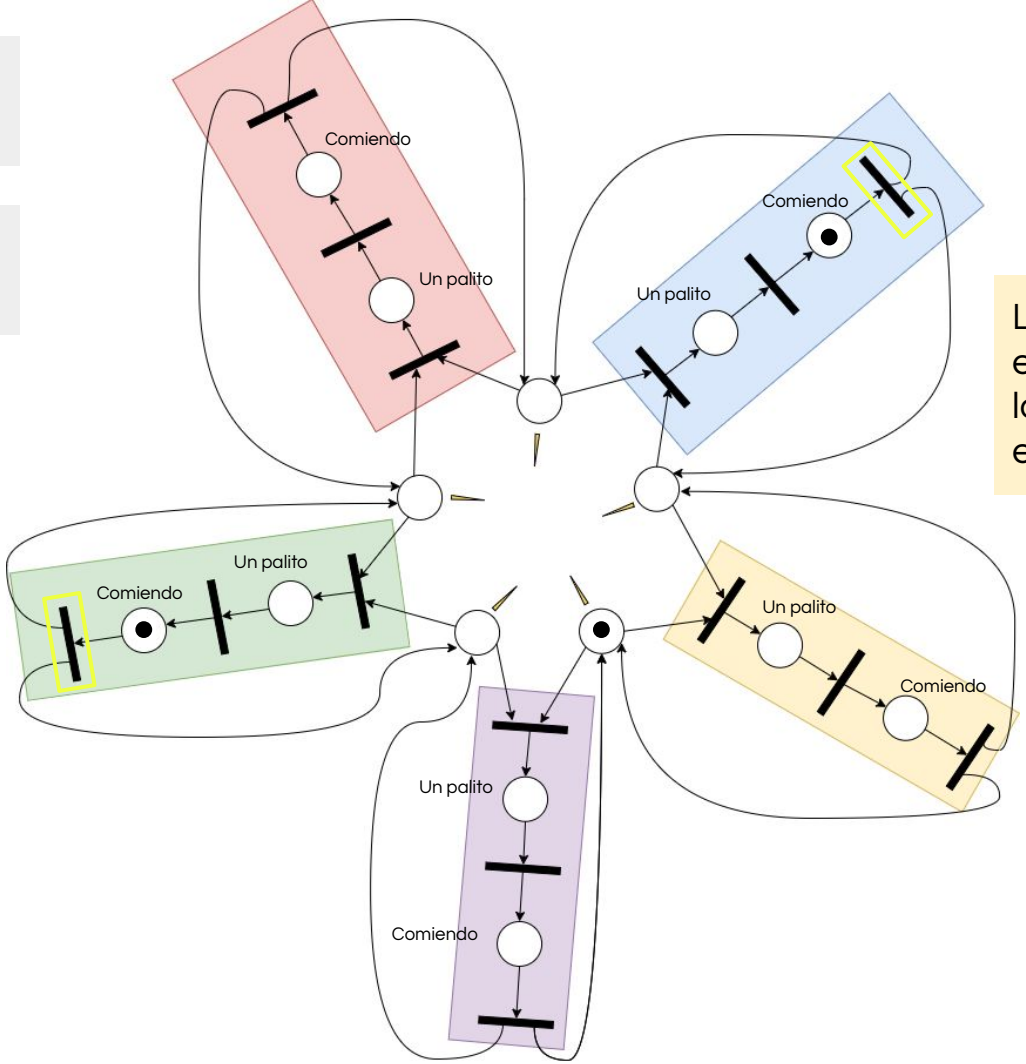
El **filósofo azul** está **comiendo**

El **filósofo verde** agarra el **palito a su izquierda** y **seguidamente** el **de la derecha**



El **filósofo azul** está **comiendo**

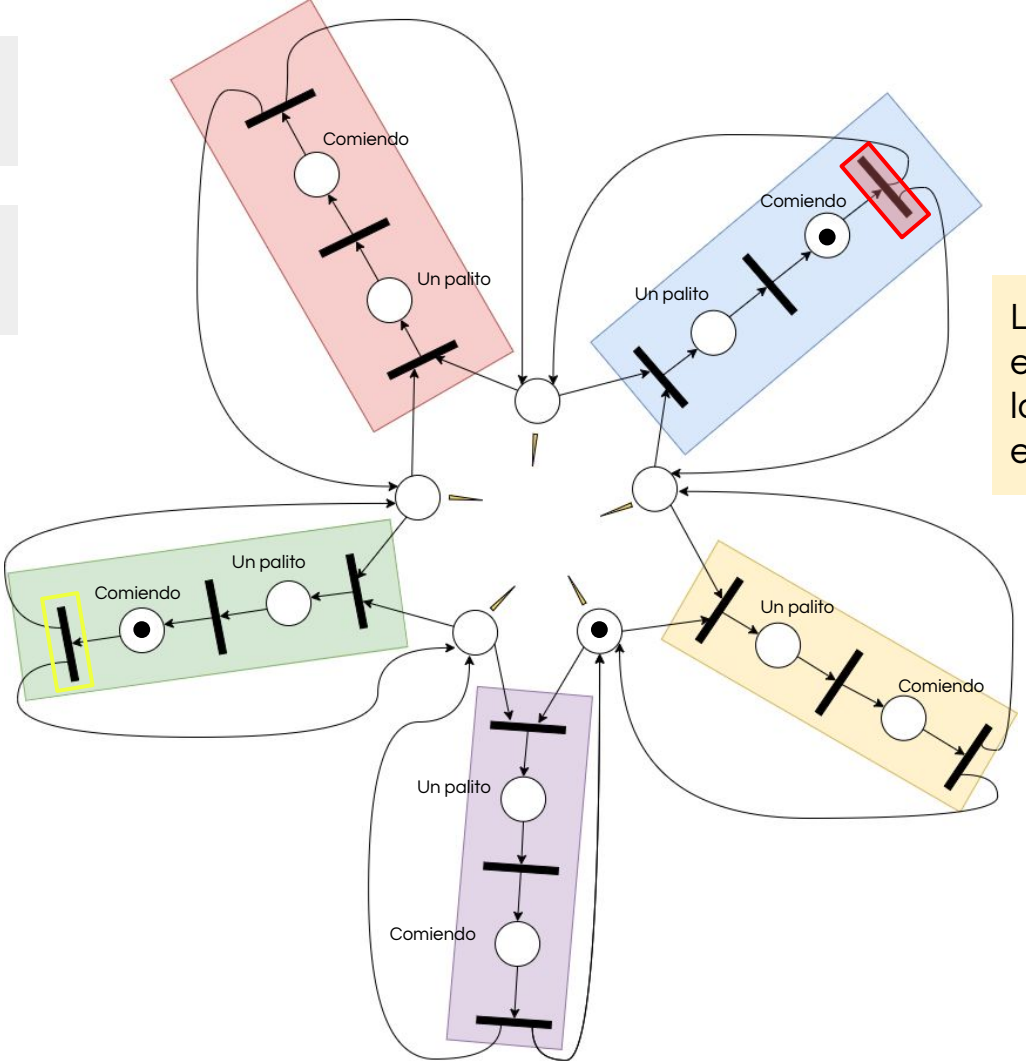
El **filósofo verde** está **comiendo**



Los **demás filósofos** están **esperando** a que los que están al lado de ellos no esten comiendo

El **filósofo azul deja de comer**

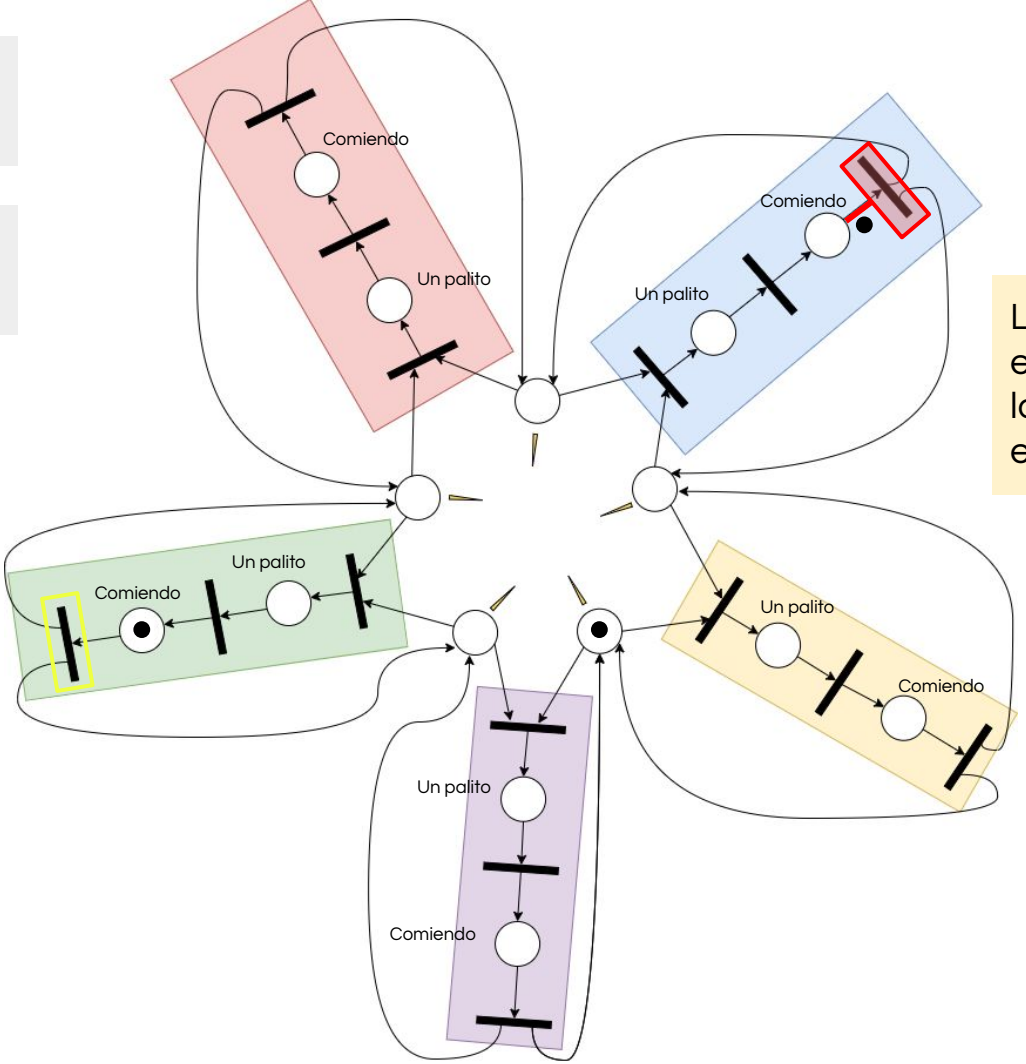
El **filósofo verde** está **comiendo**



Los **demás filósofos** están **esperando** a que los que están al lado de ellos no esten comiendo

El **filósofo azul deja de comer**

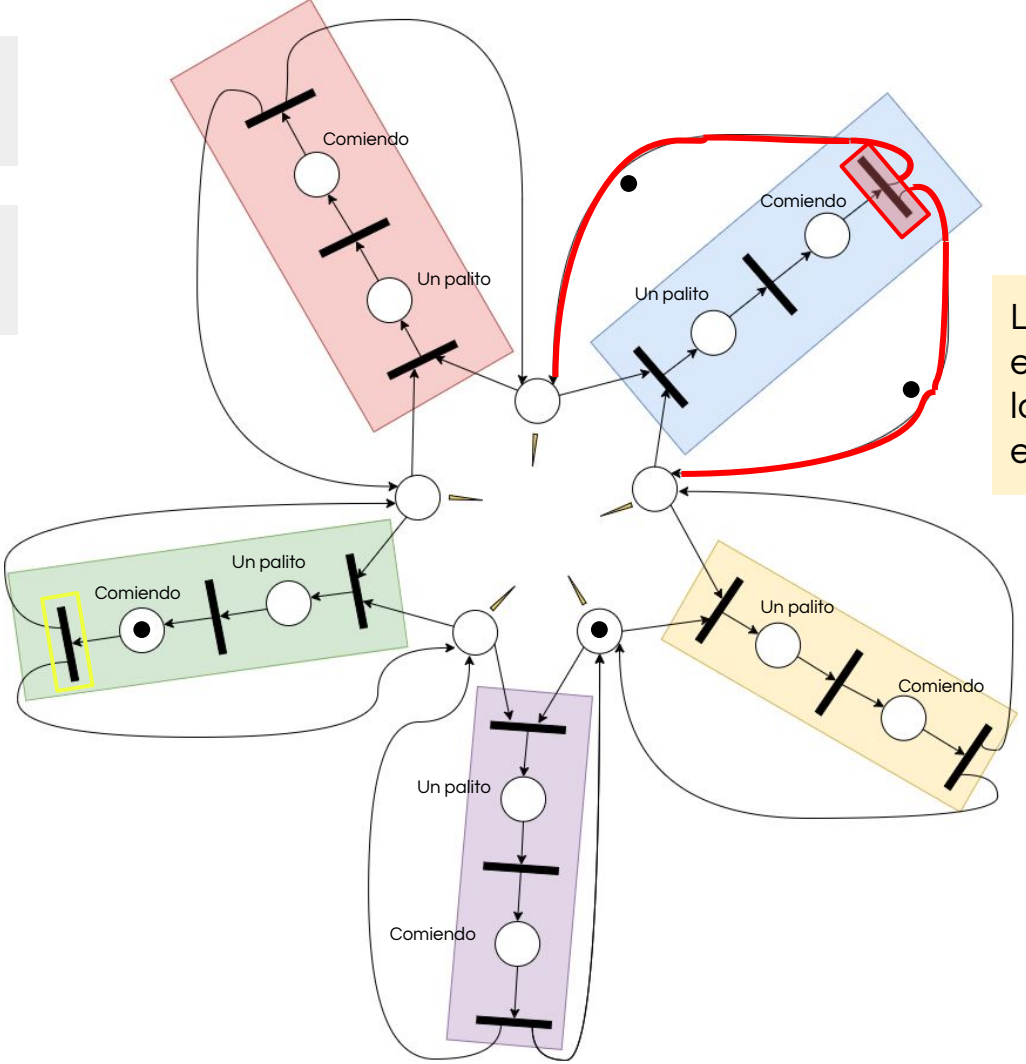
El **filósofo verde** está **comiendo**



Los **demás filósofos** están **esperando** a que los que están al lado de ellos no esten comiendo

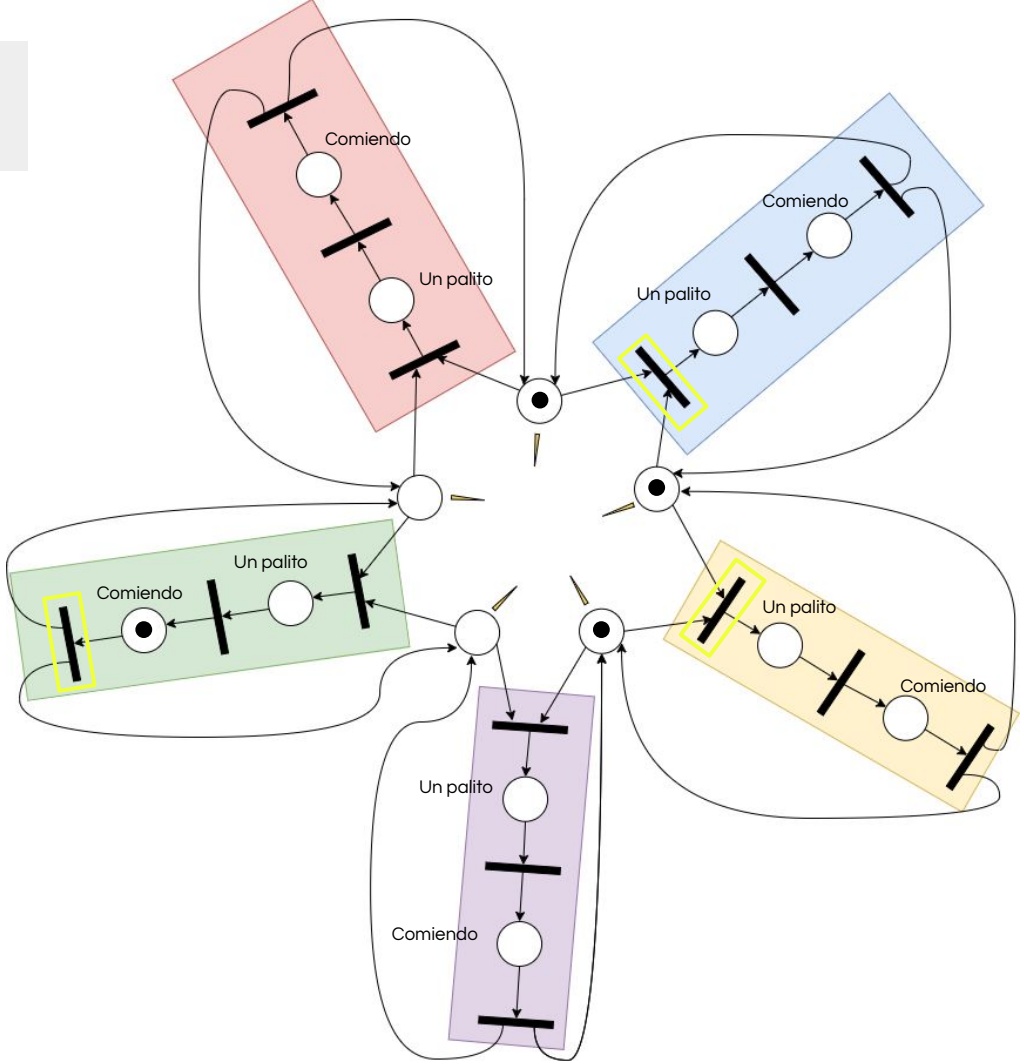
El **filósofo azul deja de comer**

El **filósofo verde** está **comiendo**



Los **demás filósofos** están **esperando** a que los que están al lado de ellos no esten comiendo

El **filósofo verde** está **comiendo**

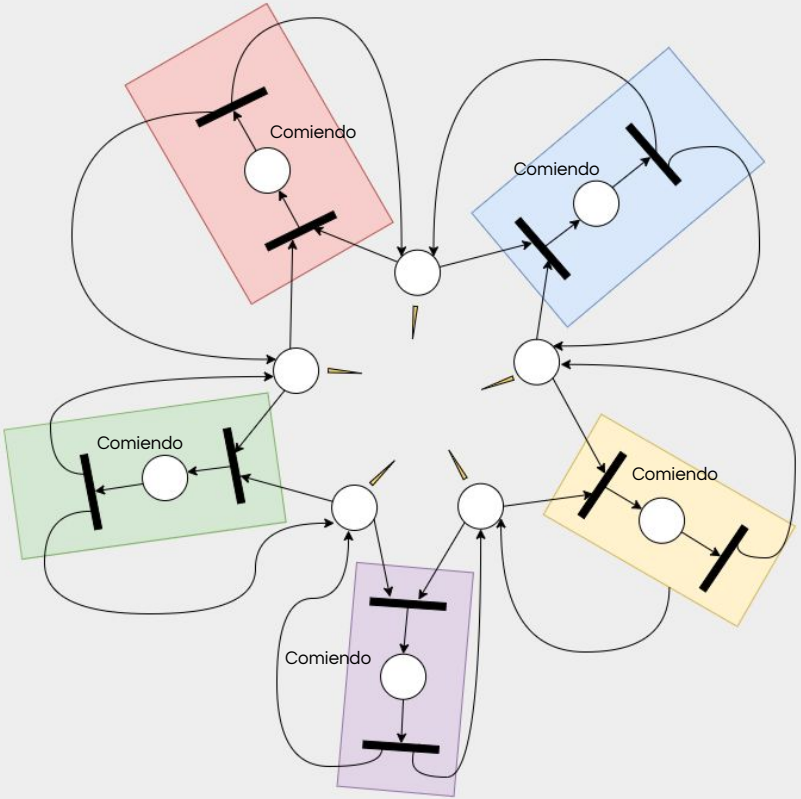


El **filosofo amarillo** ahora **puede comer**

Los filósofos **rojo** y **violeta** tienen que **esperar a que termine el verde** para poder intentar comer

Implementación en RUST

https://github.com/crpistillo/philosophers_starvation




```
let eating_states = Arc::new(RwLock::new(
                            vec!(false, false, false, false, false)));
```

```
fn eat(&self, fork:Arc<Vec<Semaphore>>, eating_states:Arc<RwLock<Vec<bool>>>) {
    loop {
        if let Ok(mut states_mut) = eating_states.write()
        {
            if (states_mut[self.id] == false
                && states_mut[self.left_philosopher()] == false
                && states_mut[self.right_philosopher()] == false)
            {
                states_mut[self.id] = true;
                fork.get(self.left_fork()).unwrap().acquire();
                fork.get(self.right_fork()).unwrap().acquire();

                println!("Filosofo {} comiendo!", self.id);
            }
        }

        thread::sleep(Duration::from_millis(thread_rng().gen_range(1000, 2000)));

        if let Ok(mut states_mut) = eating_states.write()
        {
            if (states_mut[self.id] == true)
            {
                fork.get(self.left_fork()).unwrap().release();
                fork.get(self.right_fork()).unwrap().release();
                states_mut[self.id] = false;

                println!("Filosofo {} pensando!", self.id);
            }
        }
    }
```


¡STARVATION!
Alterna entre 0 y 3
Alterna entre 0 y 2
Alterna entre 1 y 3
Alterna entre 1 y 4
Alterna entre 2 y 4

Solución a starvation: “fairness”

https://github.com/crpistillo/philosophers_fairness

```
let times_philosopher_ate = Arc::new(RwLock::new(vec!(0,0,0,0,0)));
```

```
fn eat(&self, fork:Arc<Vec<Semaphore>>, eating_states:Arc<RwLock<Vec<bool>>>,
           times_philosopher_ate:Arc<RwLock<Vec<usize>>>) {
        loop {
            if let Ok(mut states_mut) = eating_states.write()
            {
                if let Ok(mut times_ate_mut) = times_philosopher_ate.write()
                {
                    if (states_mut[self.id] == false
                        && states_mut[self.left_philosopher()] == false
                        && states_mut[self.right_philosopher()] == false
                        && times_ate_mut[self.id] < MAX_CONSECUTIVE_TIMES)
                    {
                        times_ate_mut[self.id] = times_ate_mut[self.id] + 1;
                        states_mut[self.id] = true;
                        fork.get(self.left_fork()).unwrap().acquire();
                        fork.get(self.right_fork()).unwrap().acquire();

                        println!("Filosofo {} comiendo!", self.id);
                    }

                    if(!times_ate_mut.contains(&0))
                    {
                        for item in &mut *times_ate_mut { *item = 0; }
                    }
                }
            }

            thread::sleep ...
```

Si ya comió MAX_CONSECUTIVE_TIMES - 1, le da prioridad a los otros filósofos que tienen hambre

Si todos comieron al menos una vez, se vuelve al estado inicial

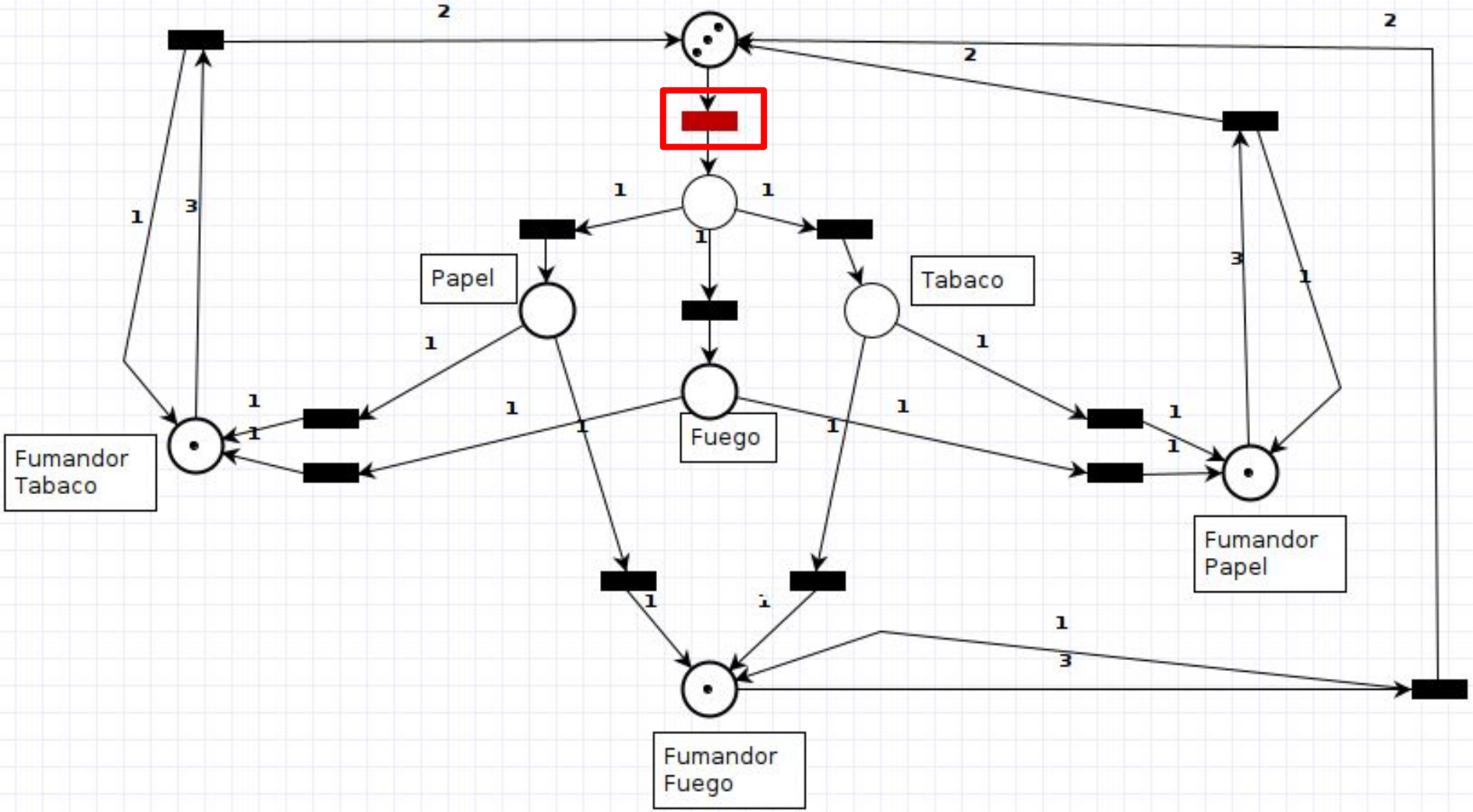
Problema de los fumadores

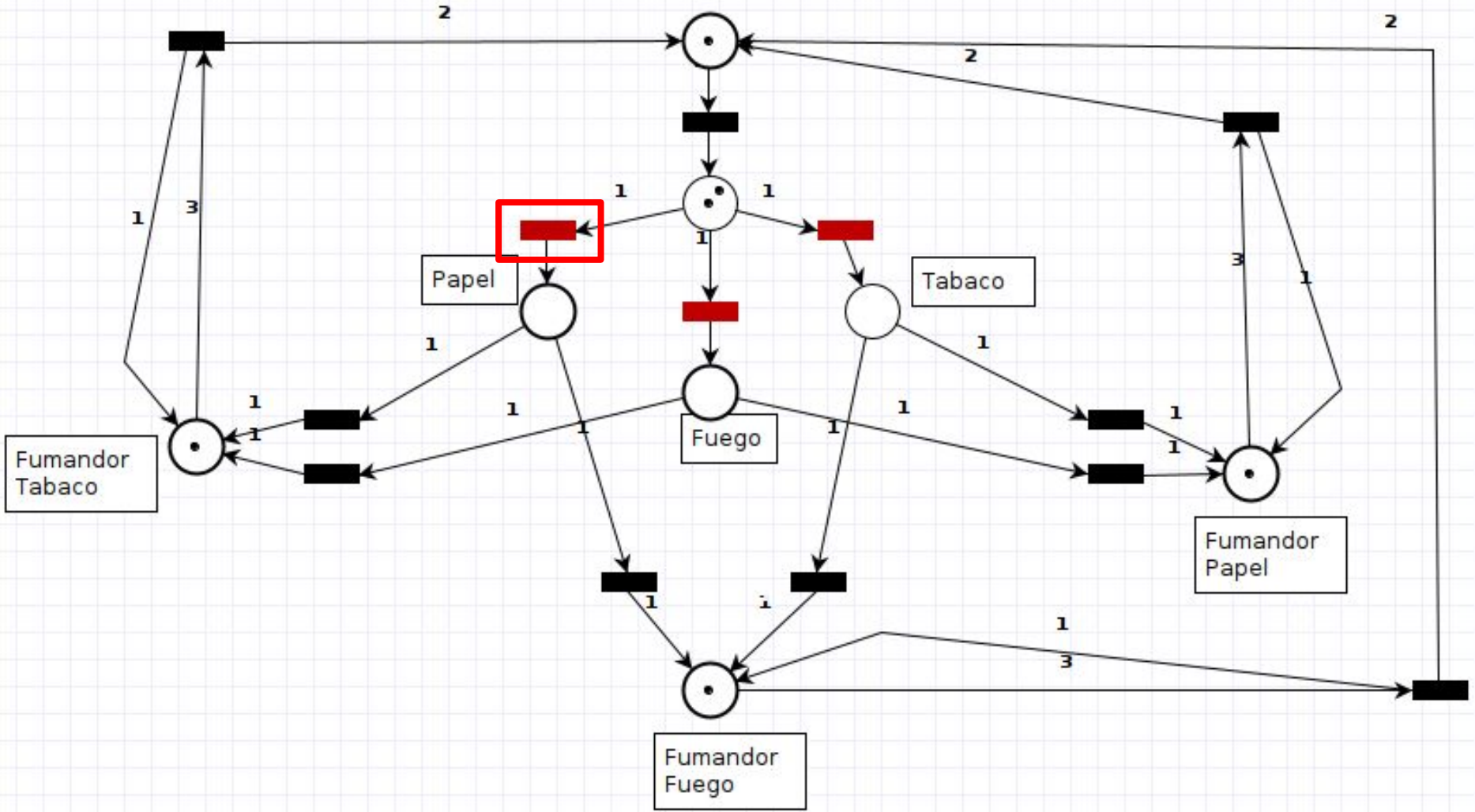
Consideremos que para fumar un cigarrillo se necesitan 3 ingredientes: tabaco, papel y fuego.

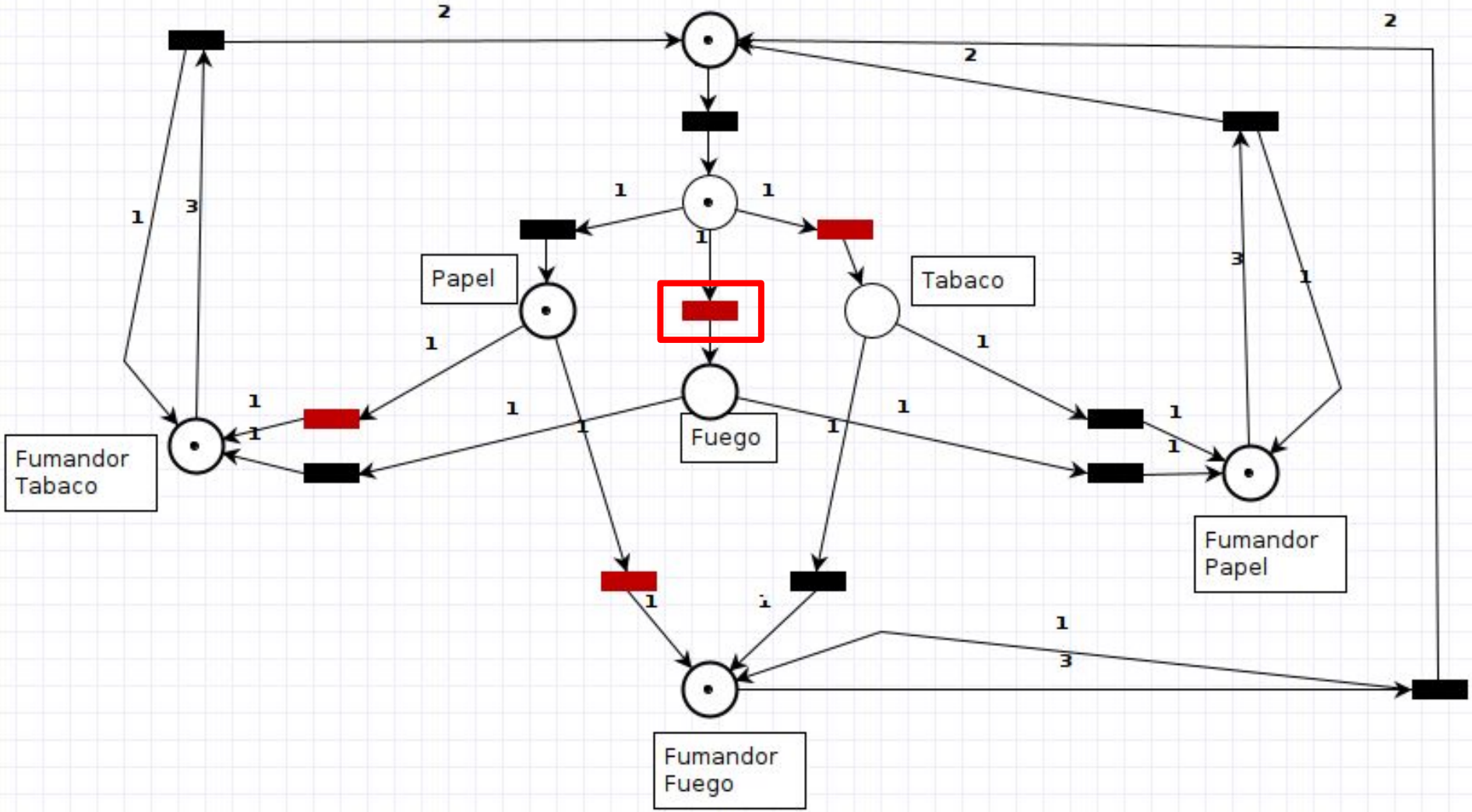
Hay 3 fumadores alrededor de una mesa. Cada uno tiene uno solo de los ingredientes y necesita los otros dos para fumar. Cada fumador posee un ingrediente distinto y le faltan los otros dos.

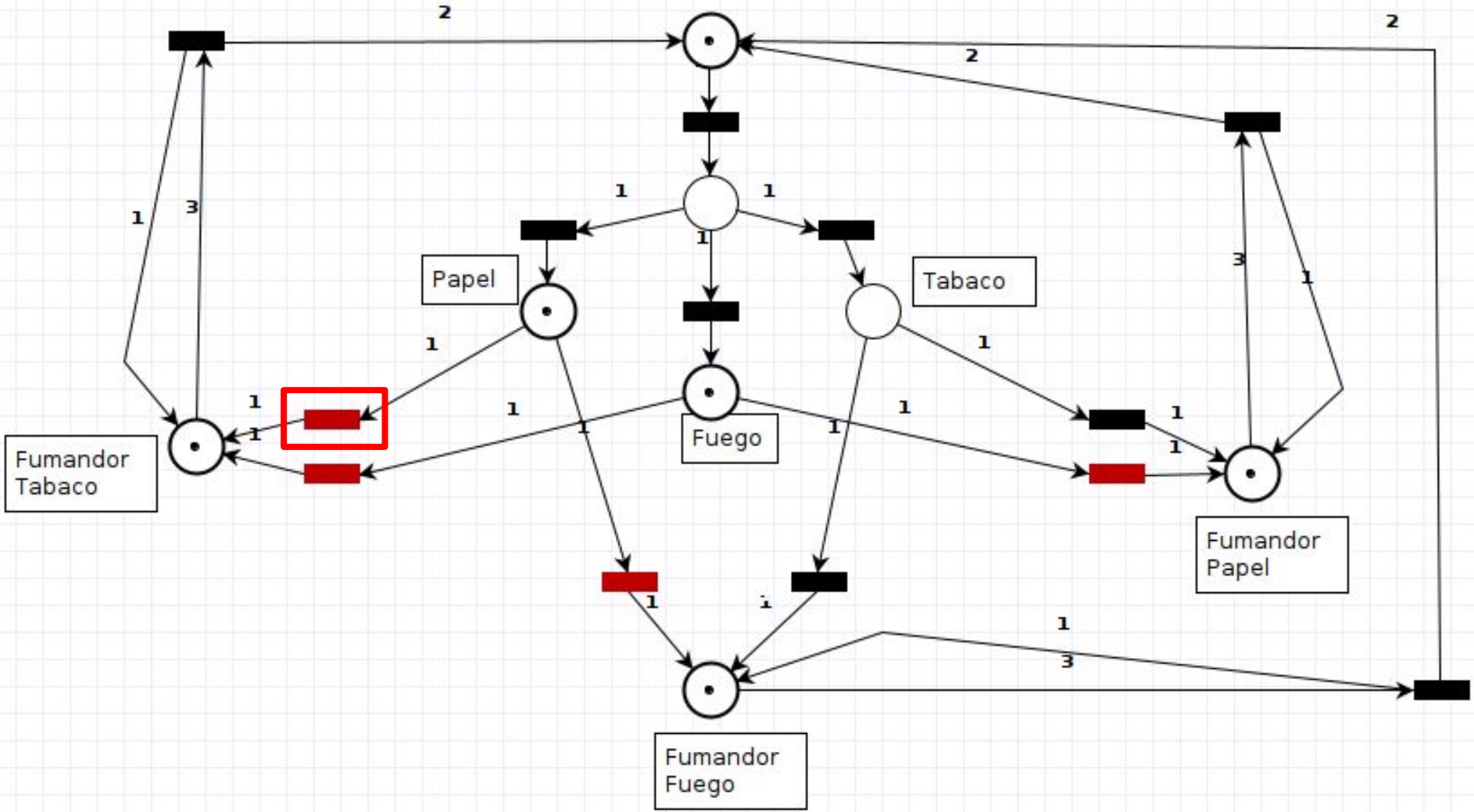
Existe además un agente que pone aleatoriamente dos ingredientes en la mesa. El fumador que los necesita los tomará para hacer su cigarrillo y fumará un rato. Cuando el fumador termina, el agente pone otros dos ingredientes

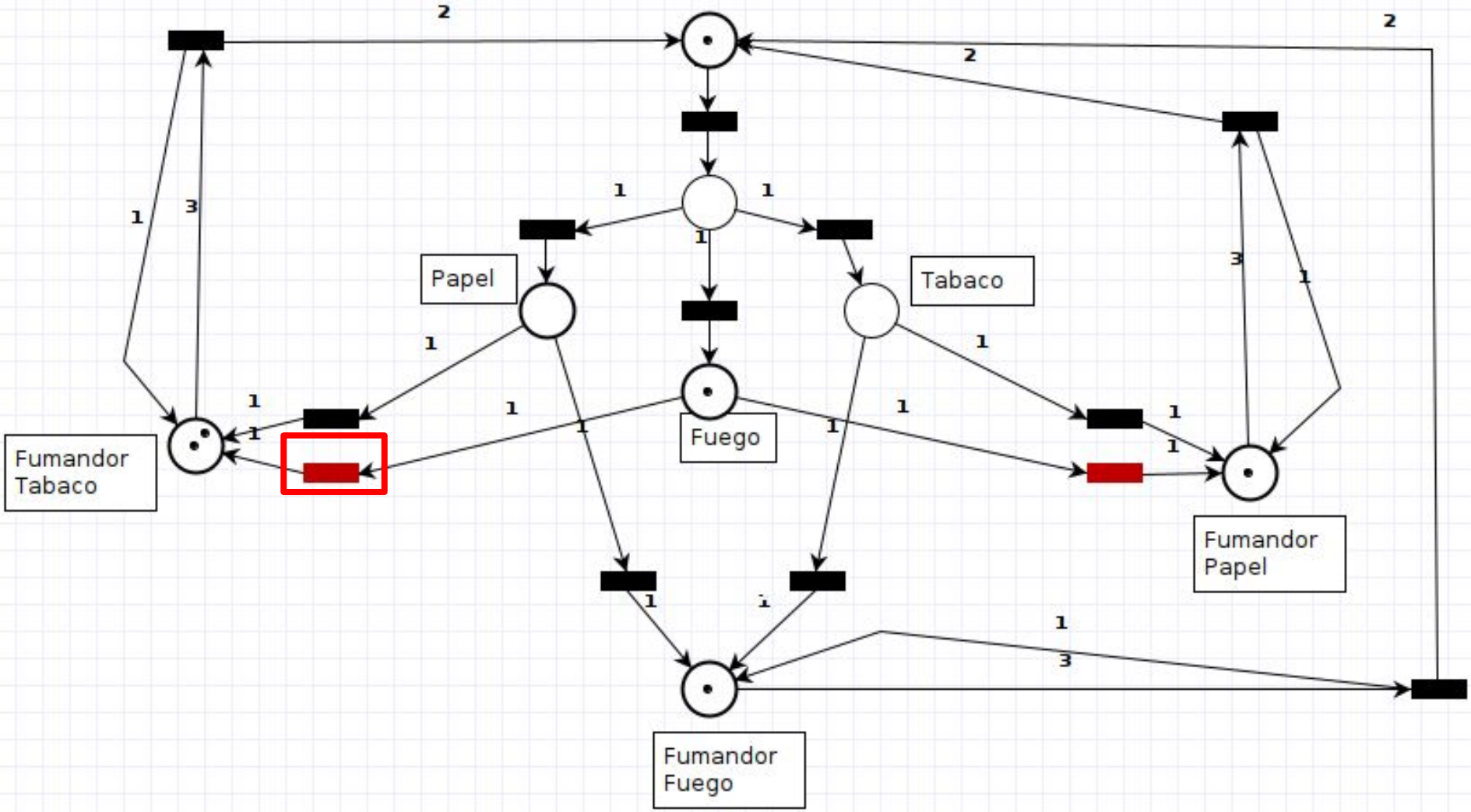
Ejemplo con Deadlock

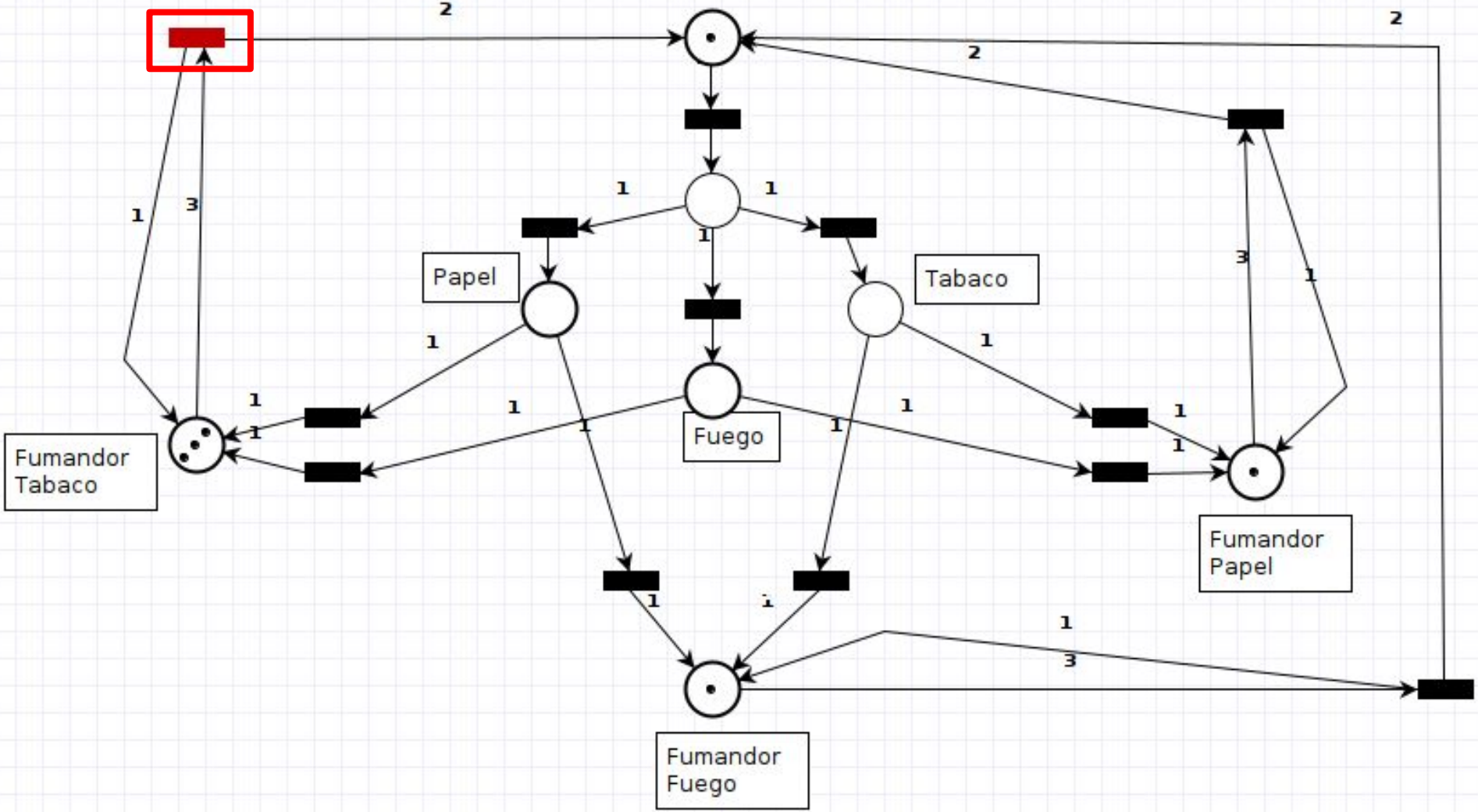

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

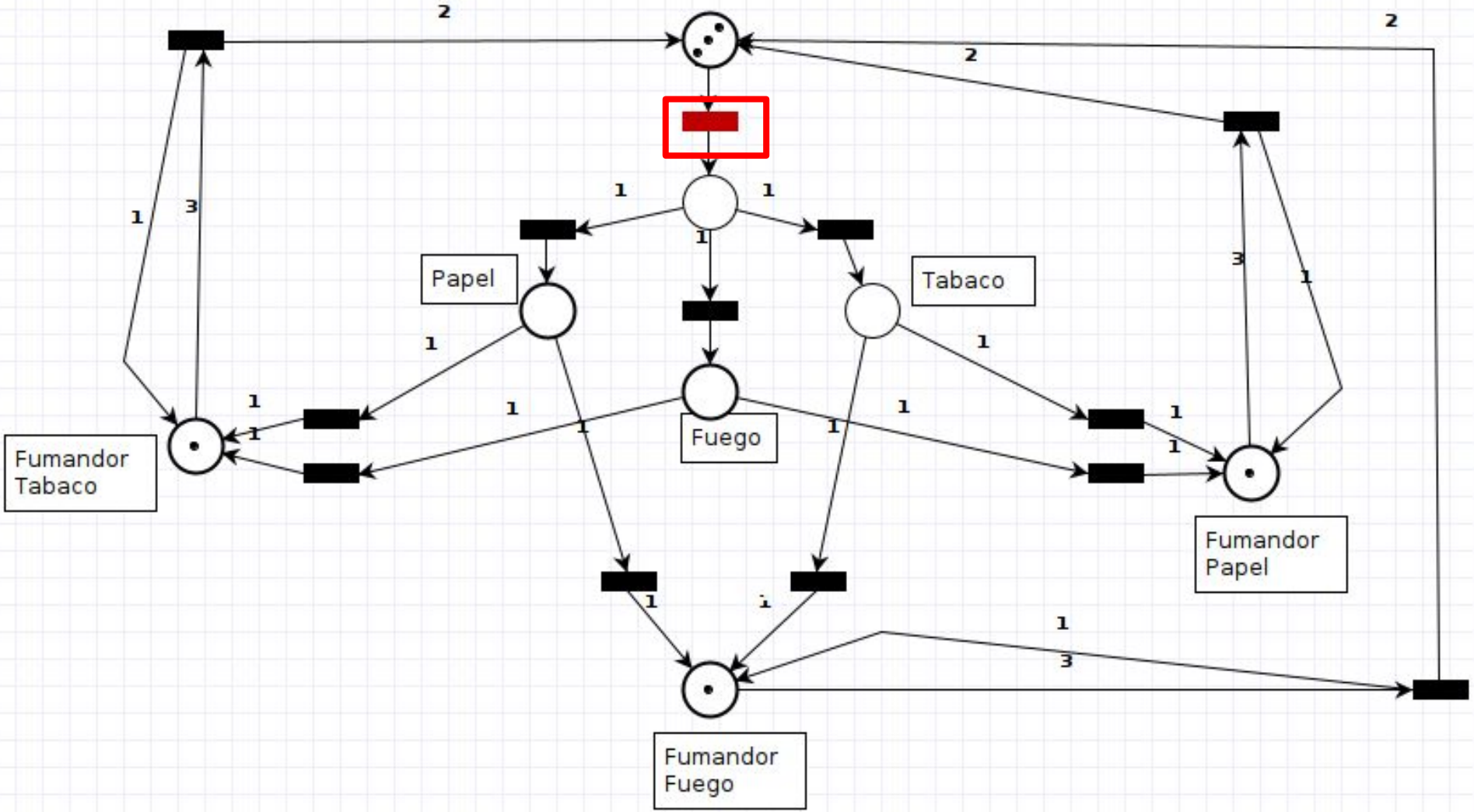

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

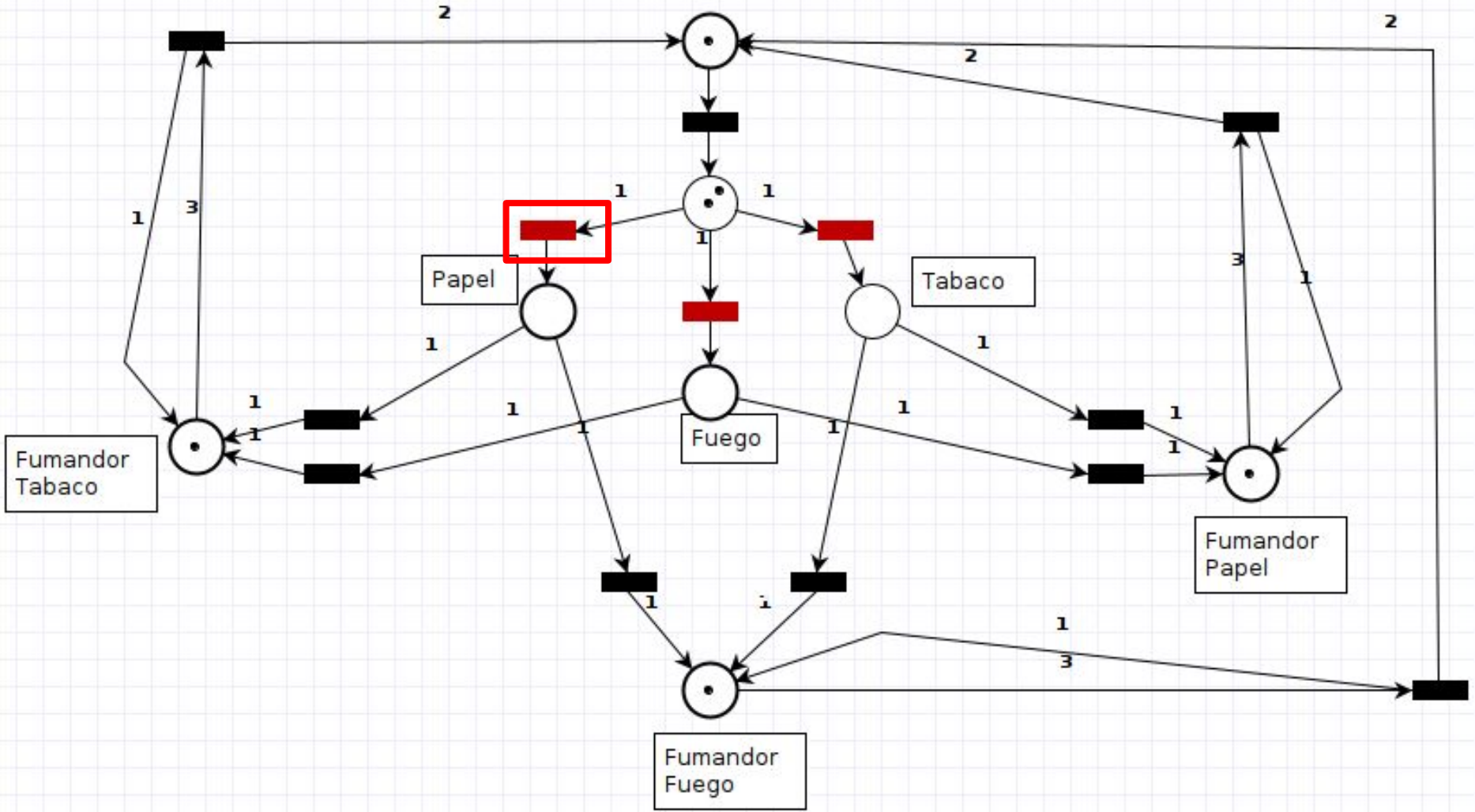

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

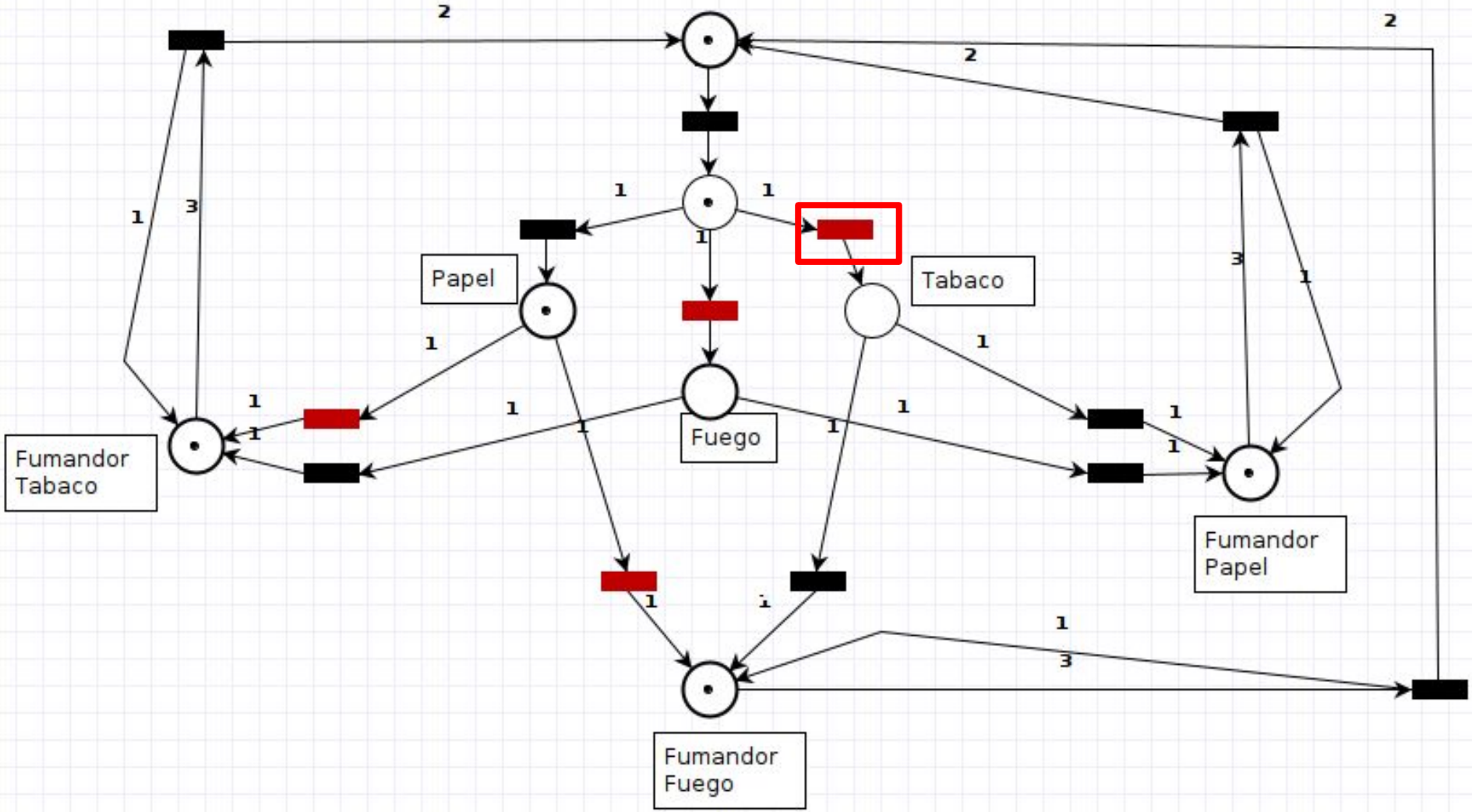

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

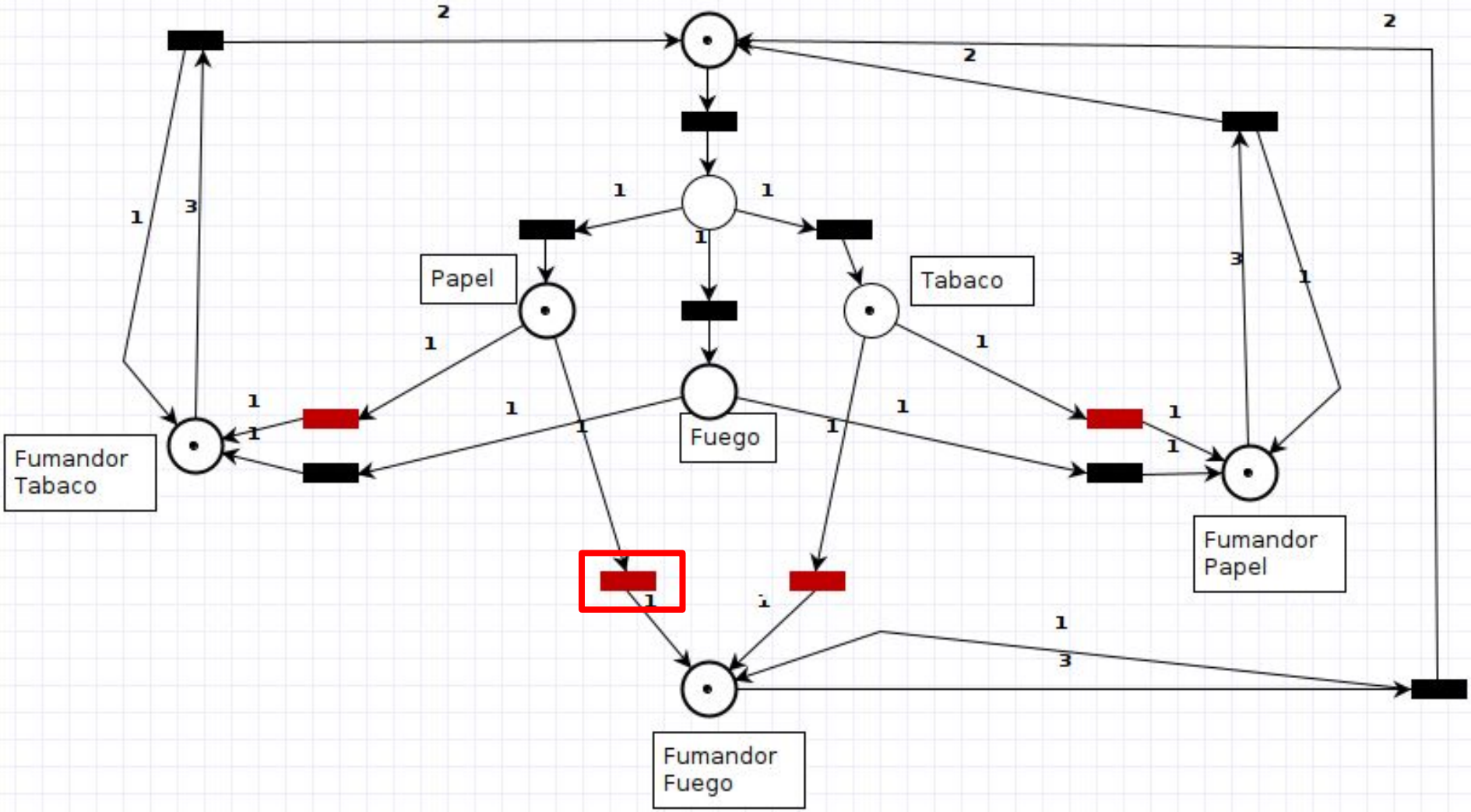

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

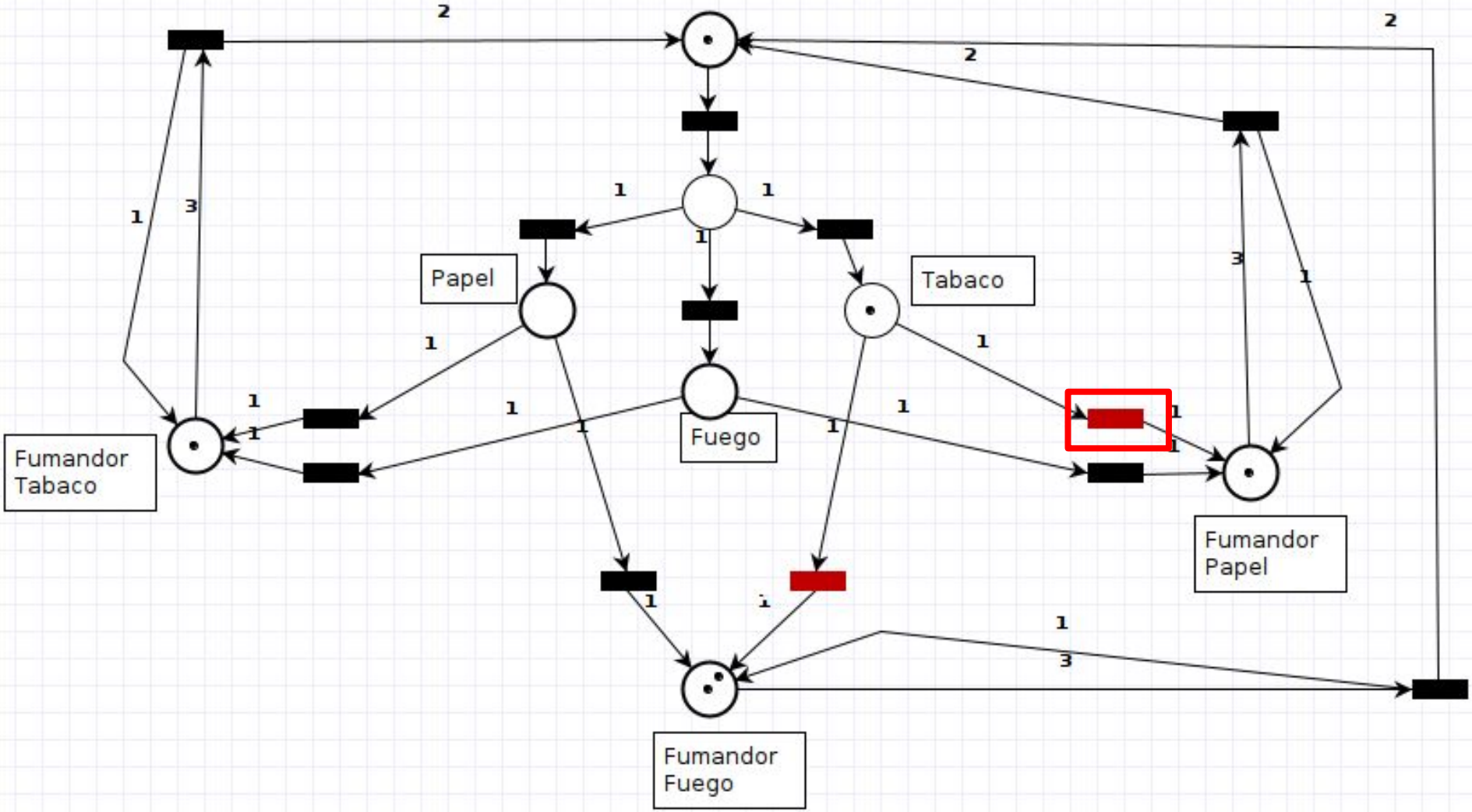

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

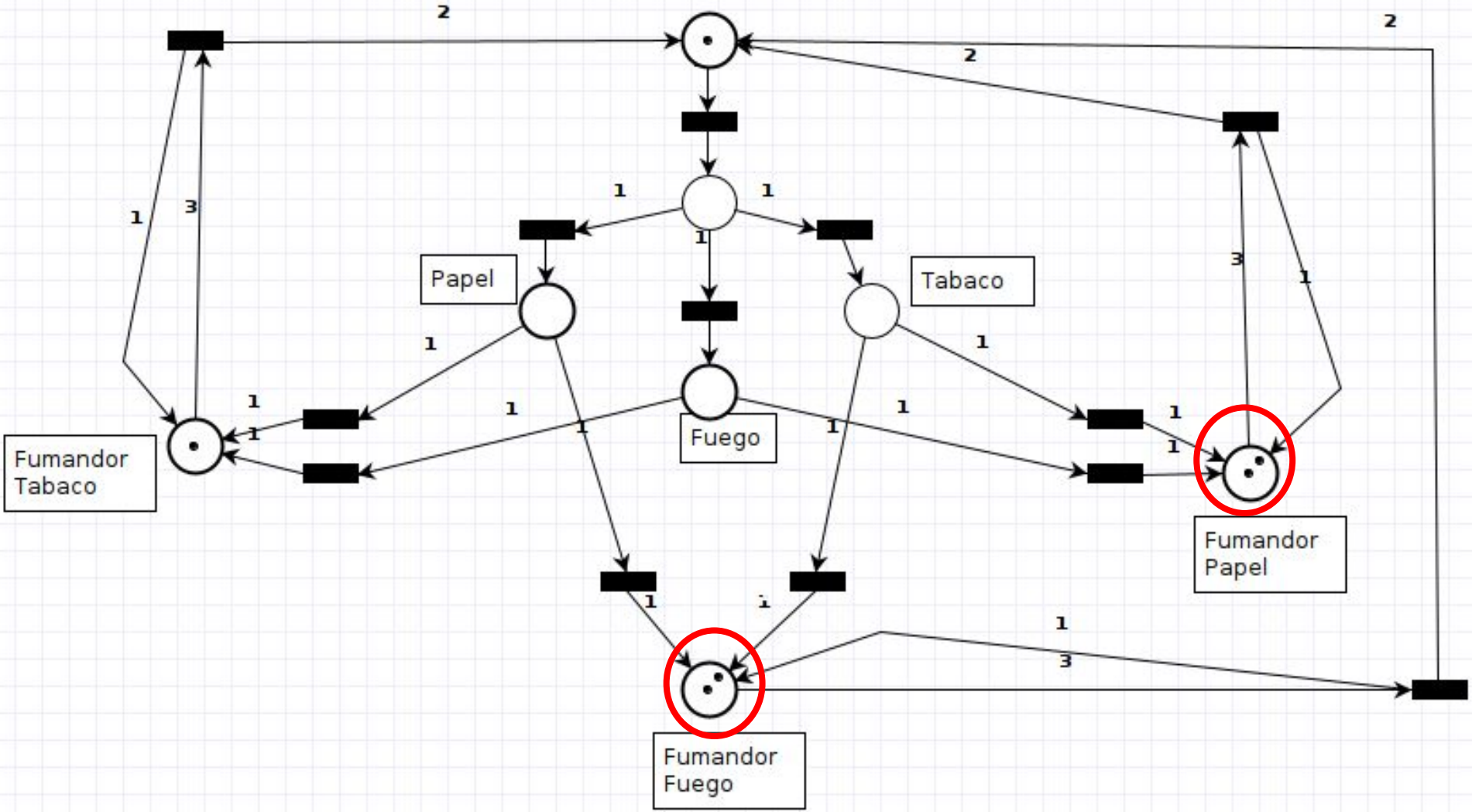

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

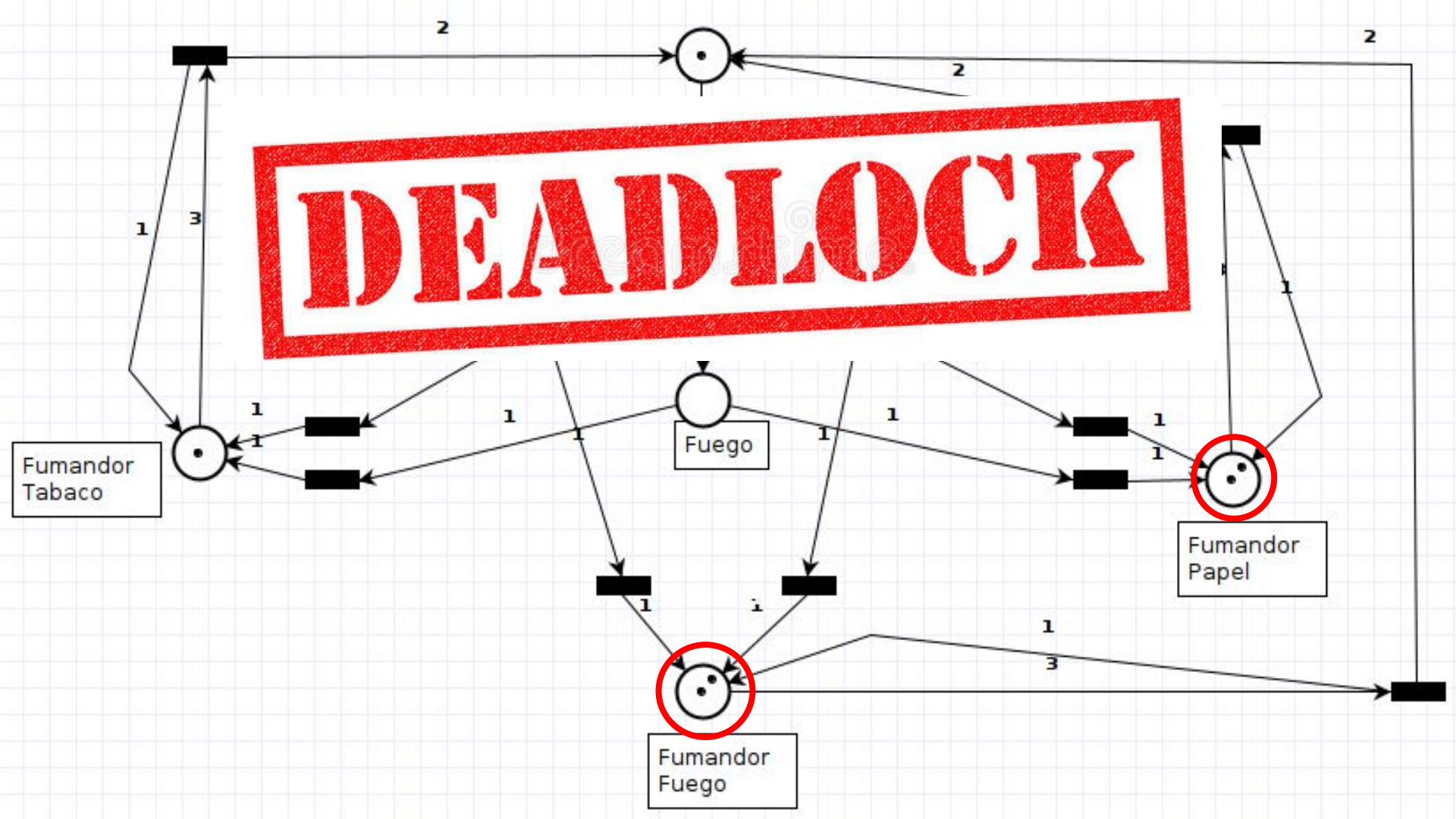

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego


Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

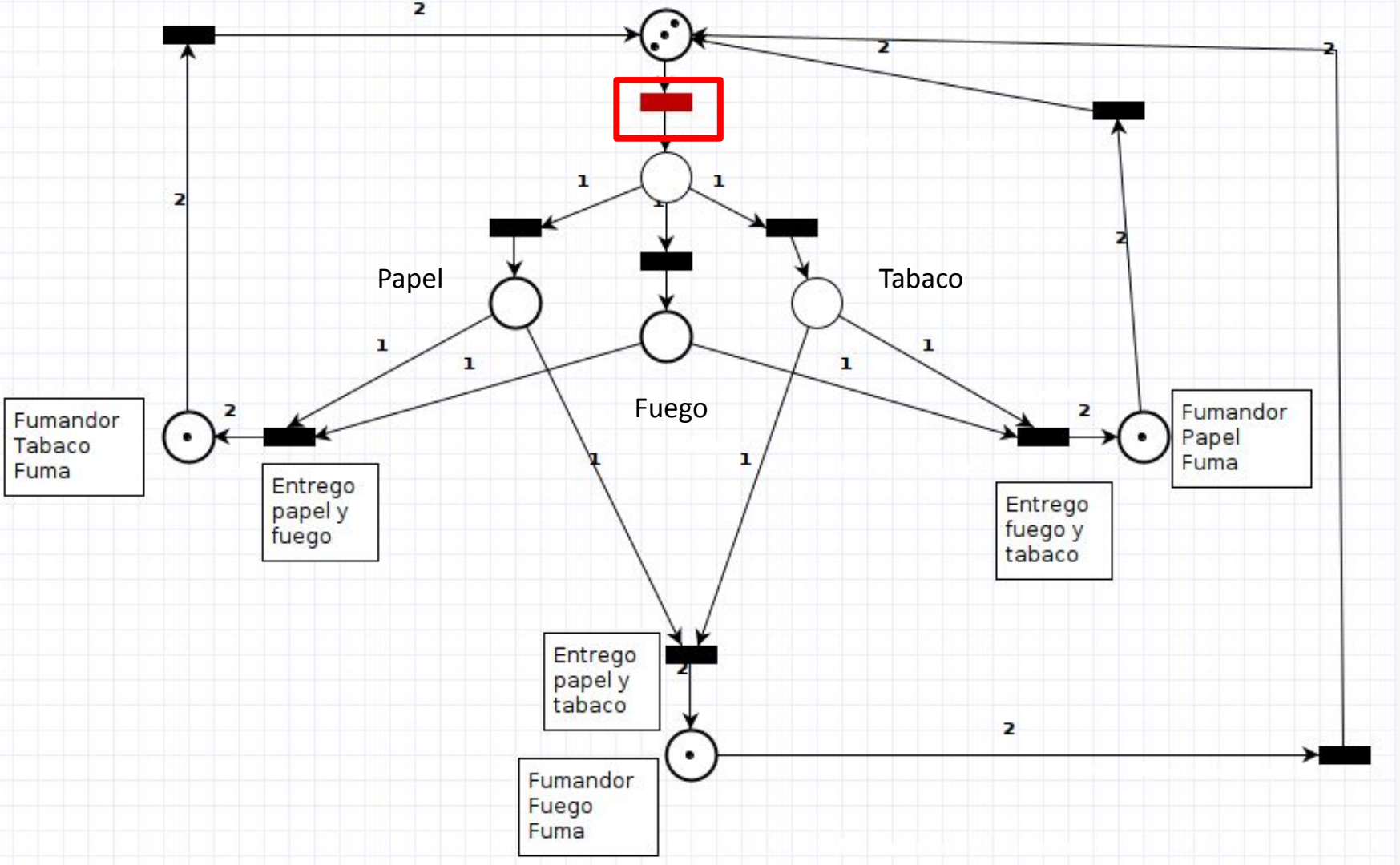

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

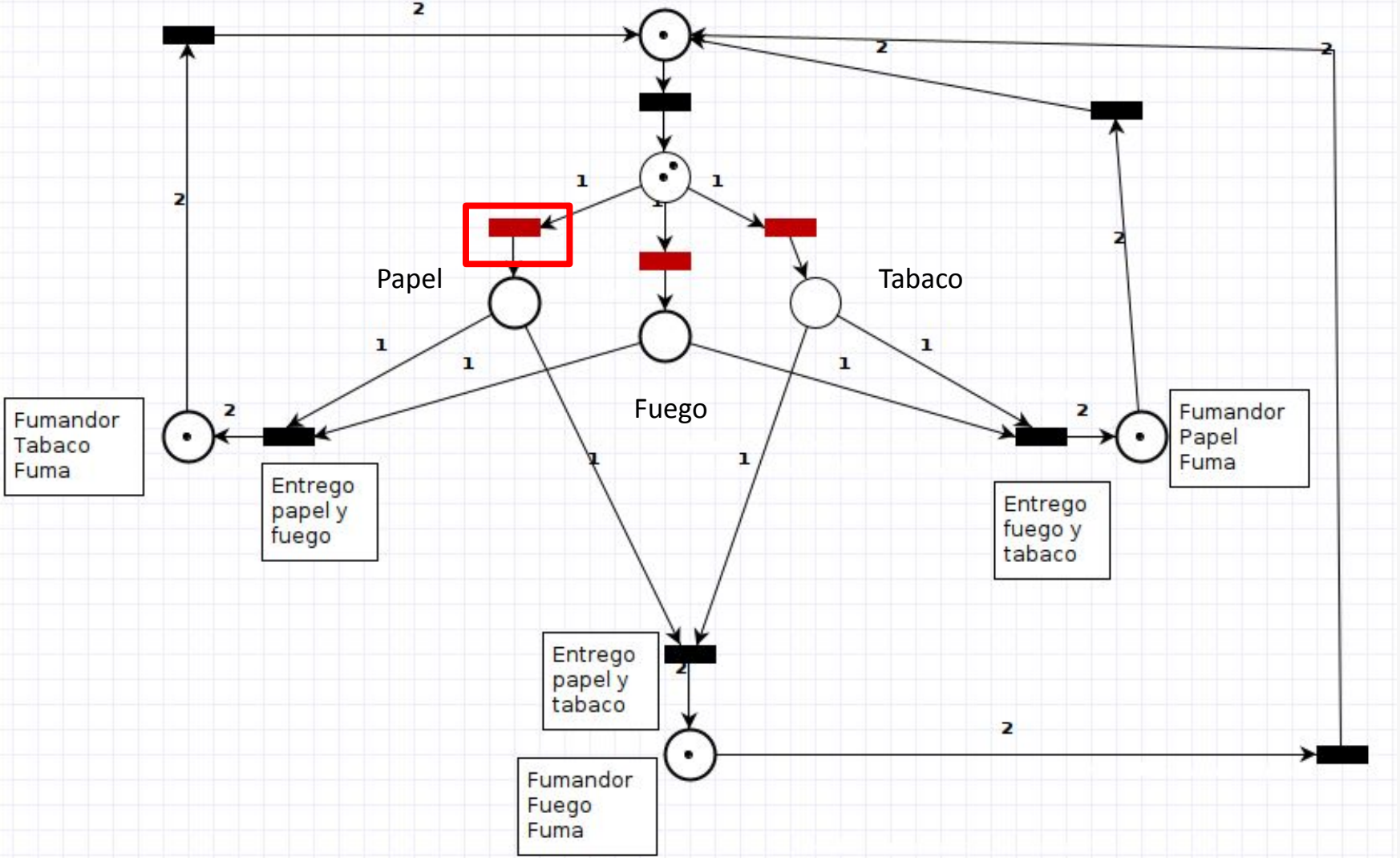

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

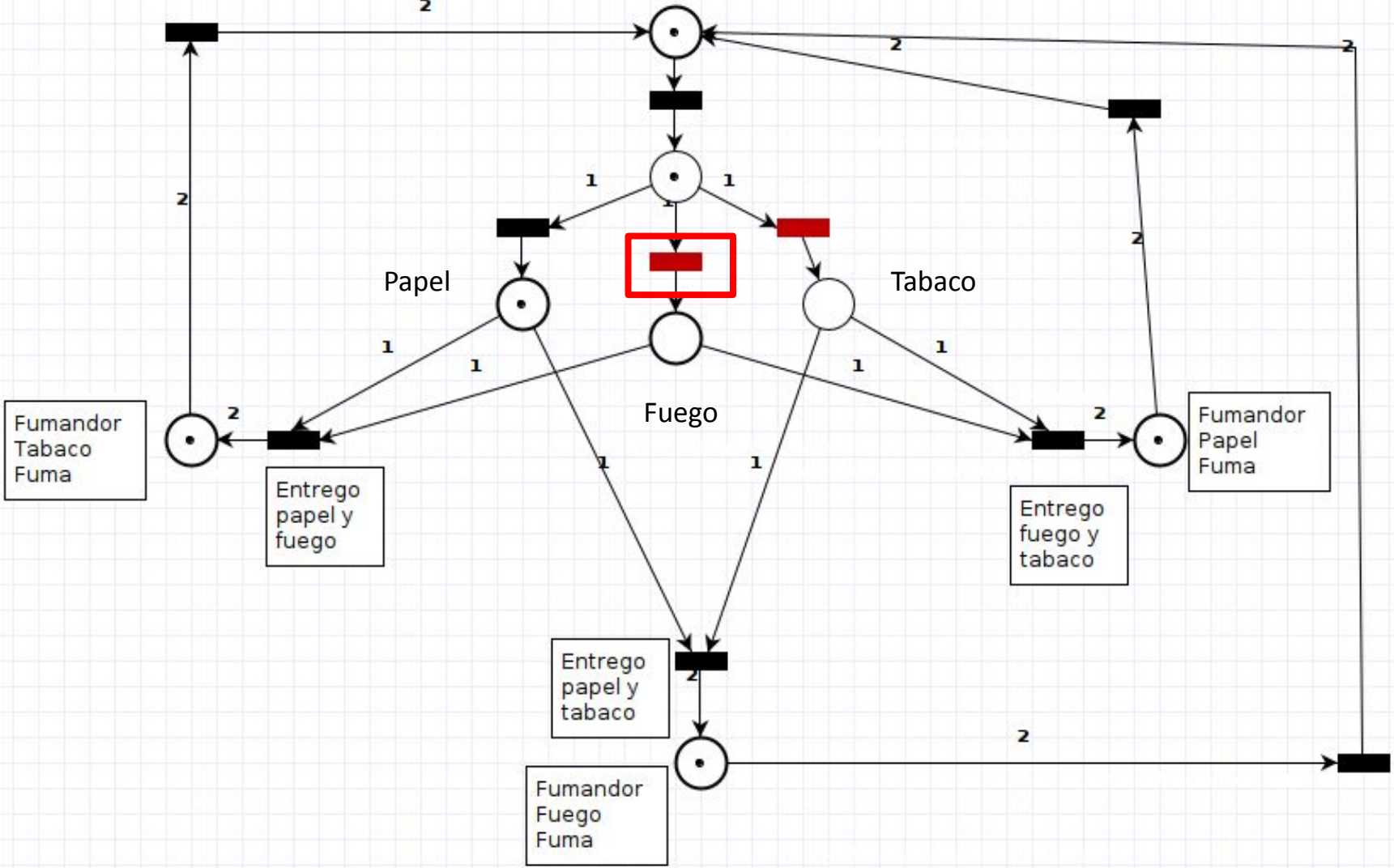

Papel
Tabaco
Fuego
Fumandor Tabaco
Fumandor Papel
Fumandor Fuego

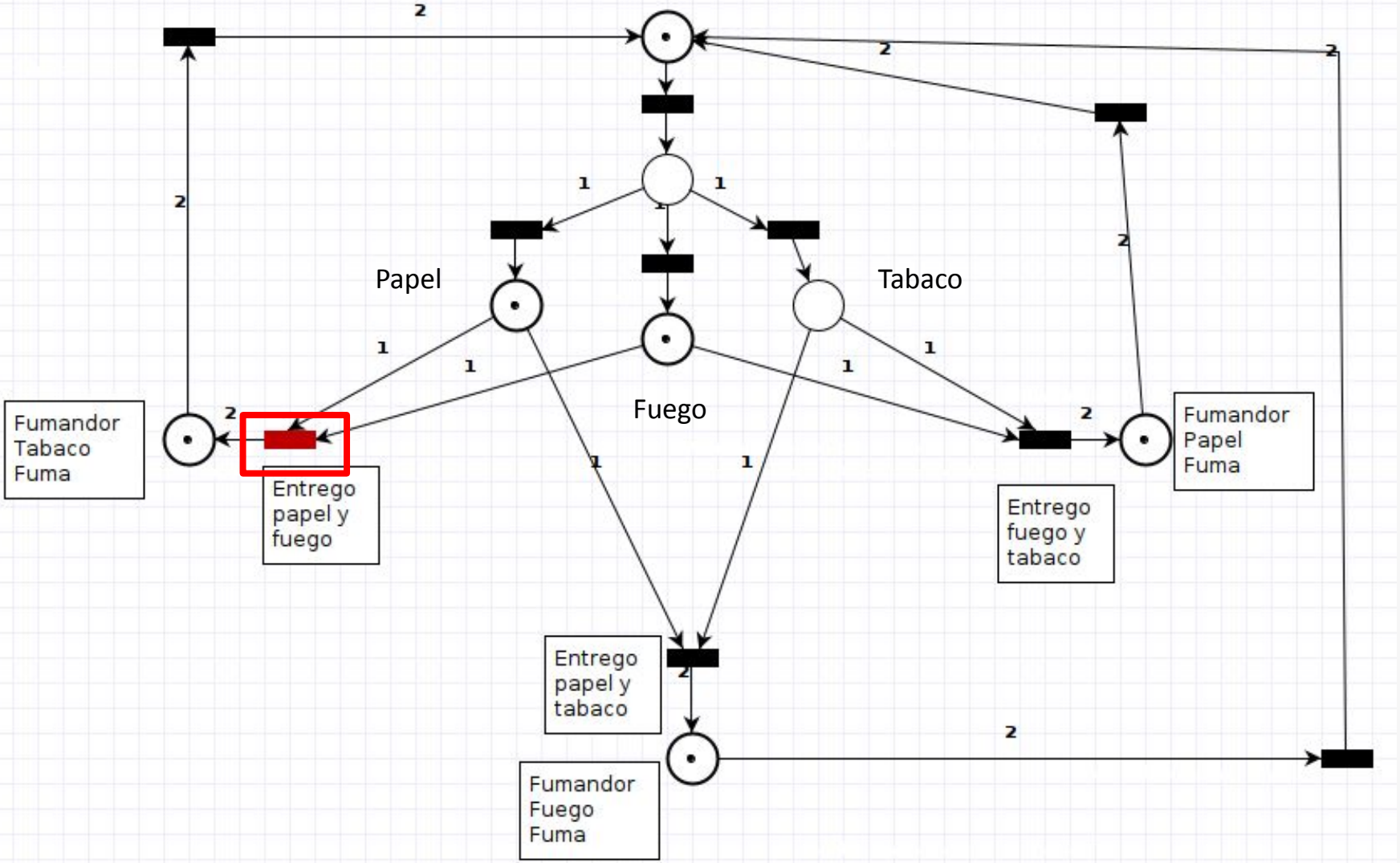

DEADLOCK
Fumandor Tabaco
Fuego
Fumandor Papel
Fumandor Fuego

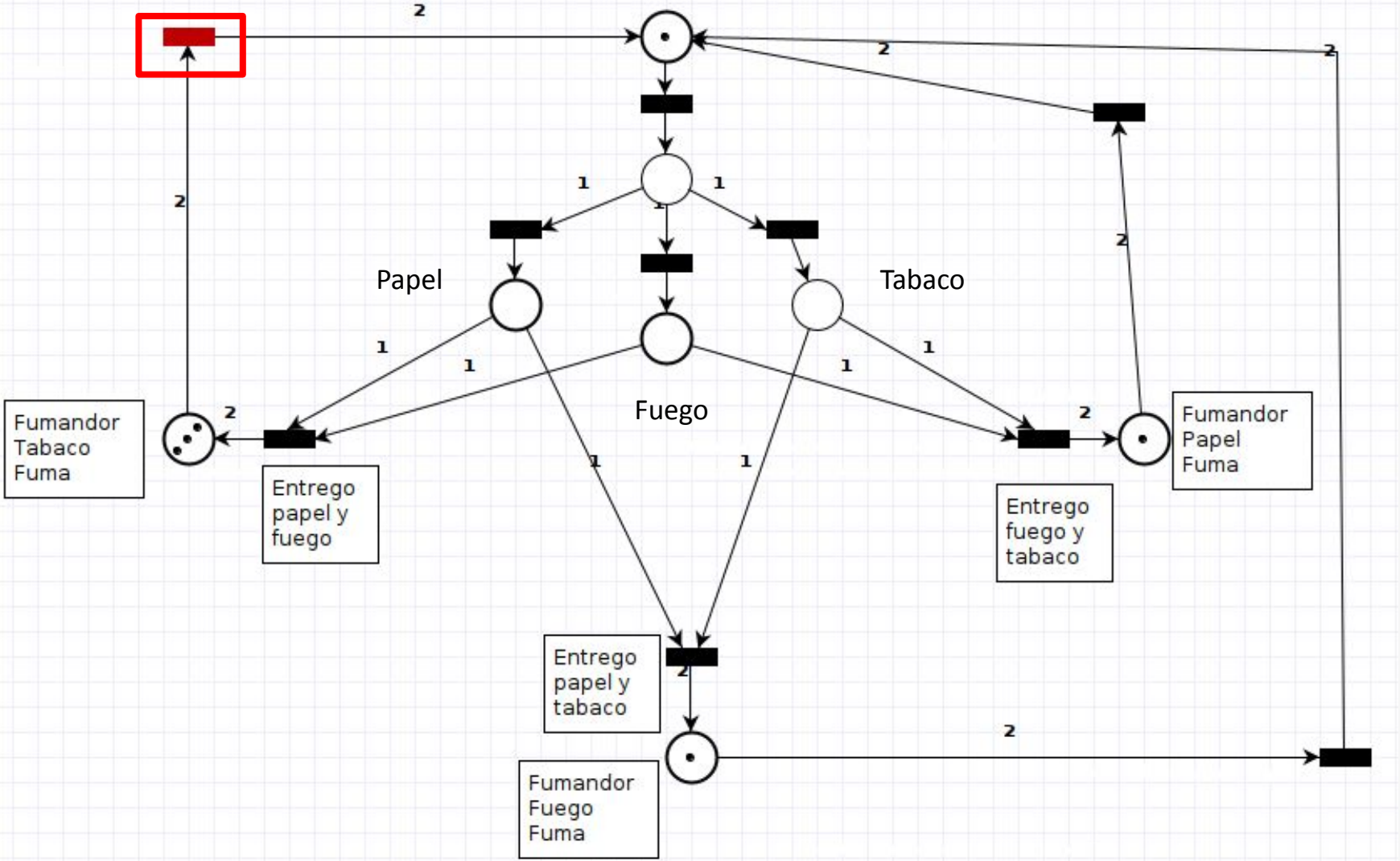
Ejemplo sin Deadlock

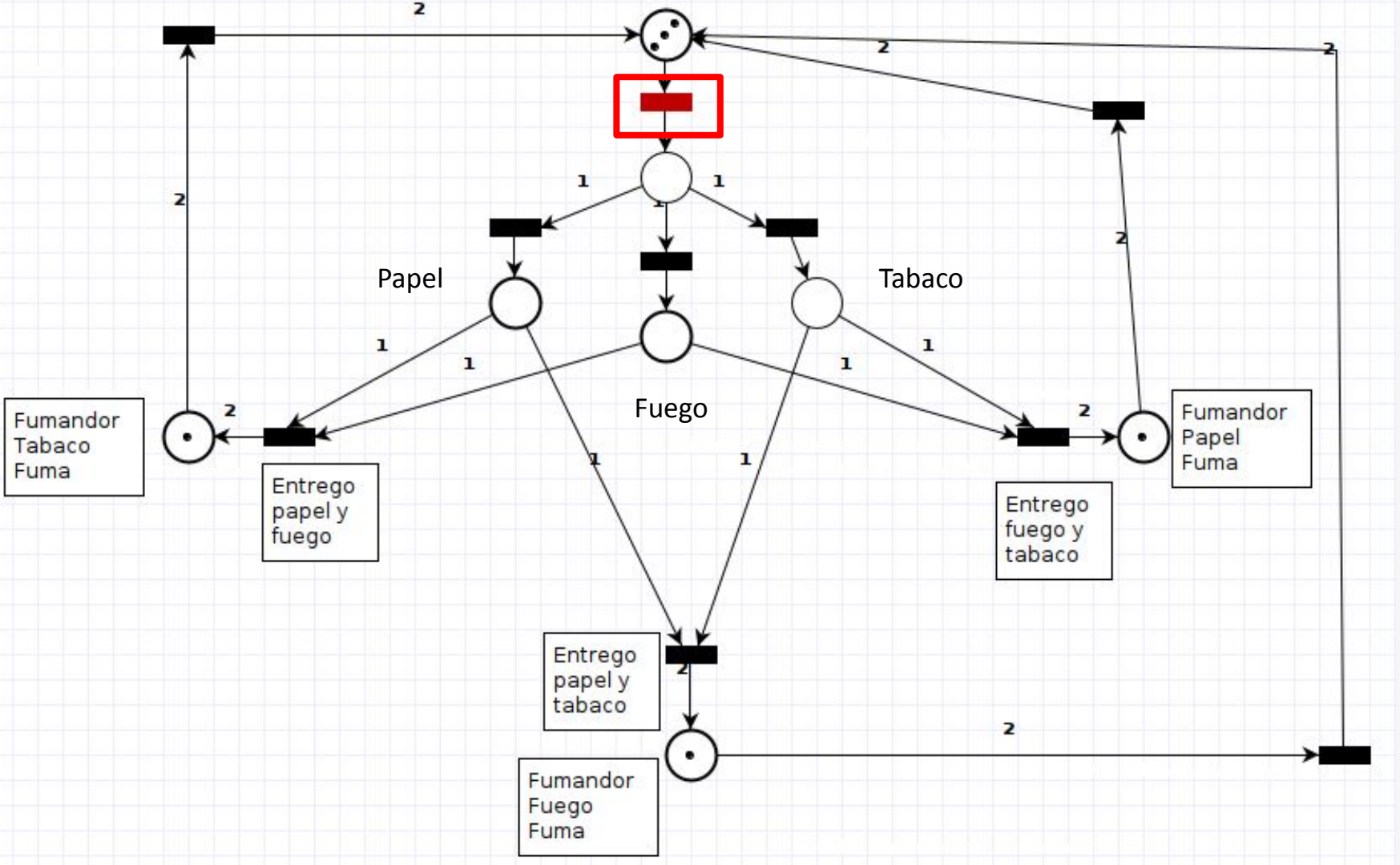

Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Fumandor Papel Fuma
Entrego fuego y tabaco
Entrego papel y tabaco
Fumandor Fuego Fuma

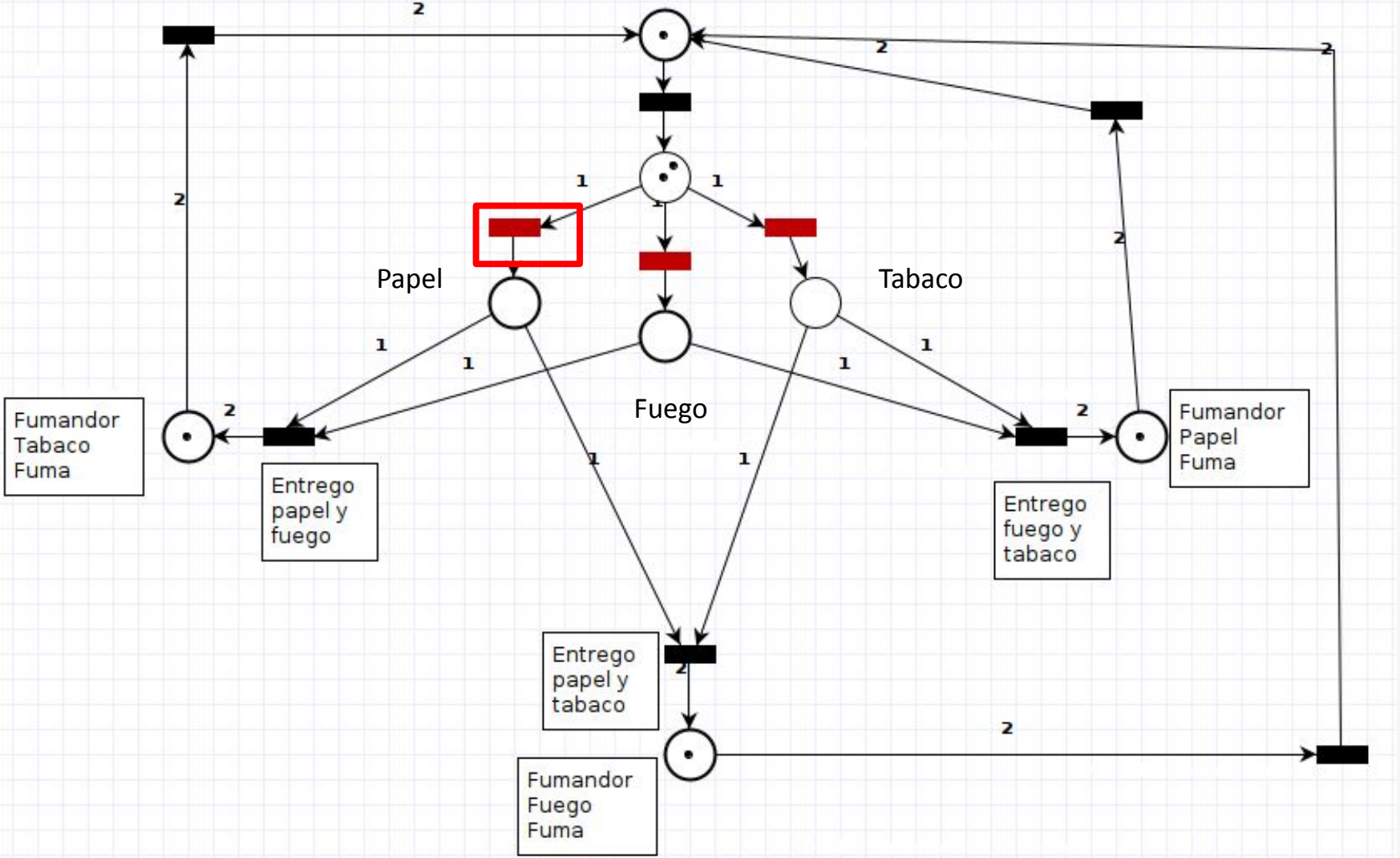

Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Entrego fuego y tabaco
Fumandor Papel Fuma
Entrego papel y tabaco
Fumandor Fuego Fuma

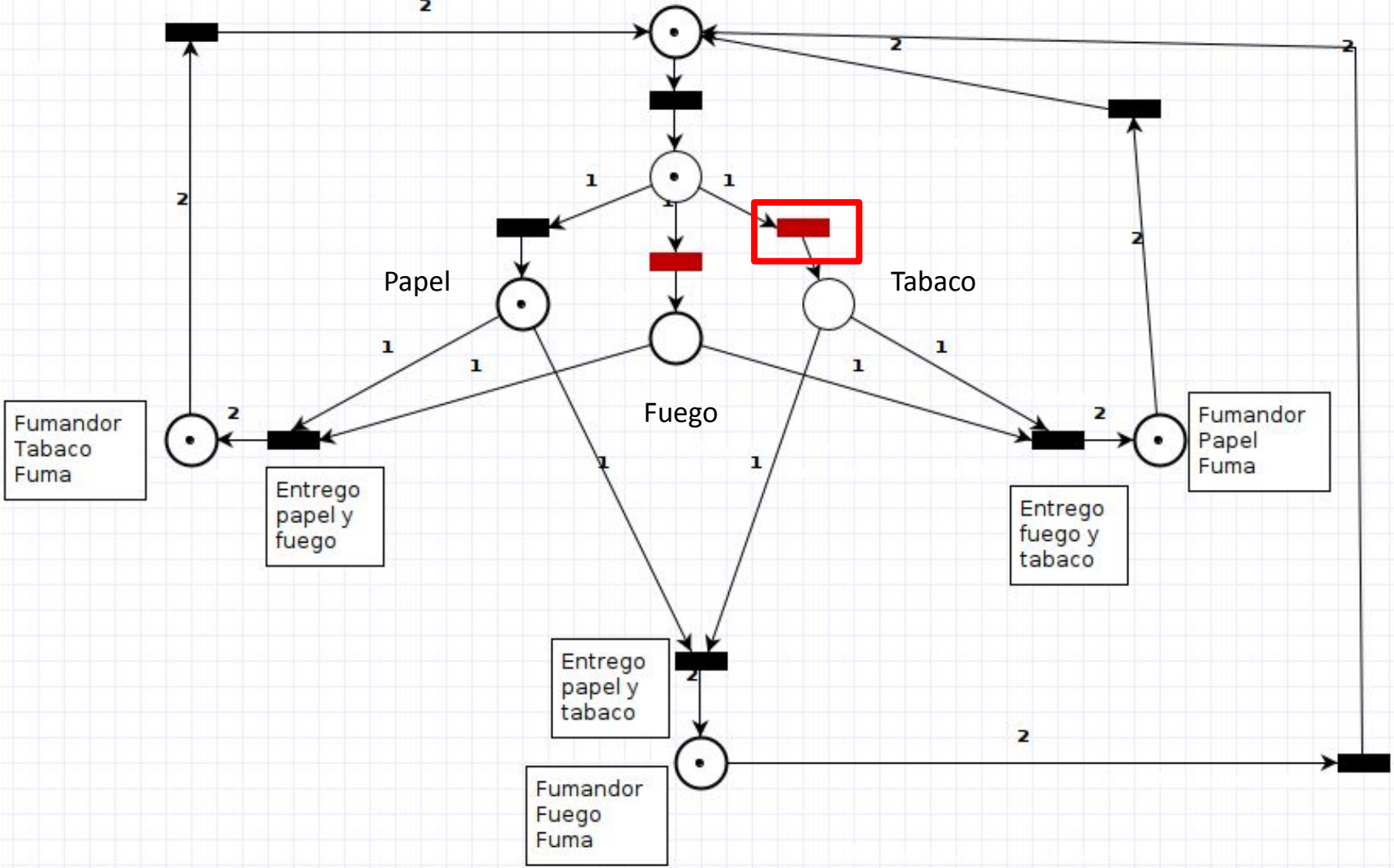

Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Fumandor Papel Fuma
Entrego fuego y tabaco
Entrego papel y tabaco
Fumandor Fuego Fuma

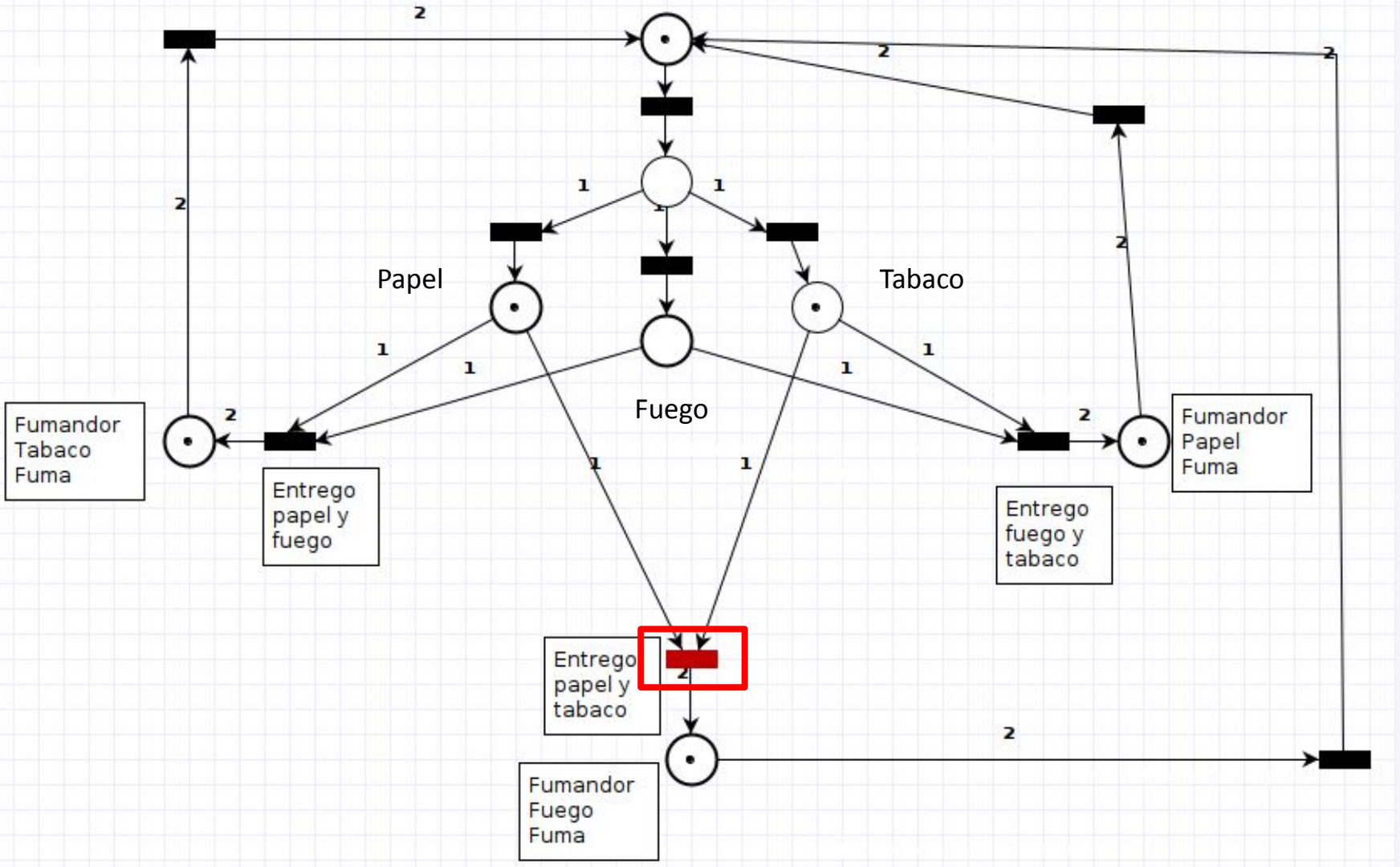

Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Entrego fuego y tabaco
Fumandor Papel Fuma
Entrego papel y tabaco
Fumandor Fuego Fuma

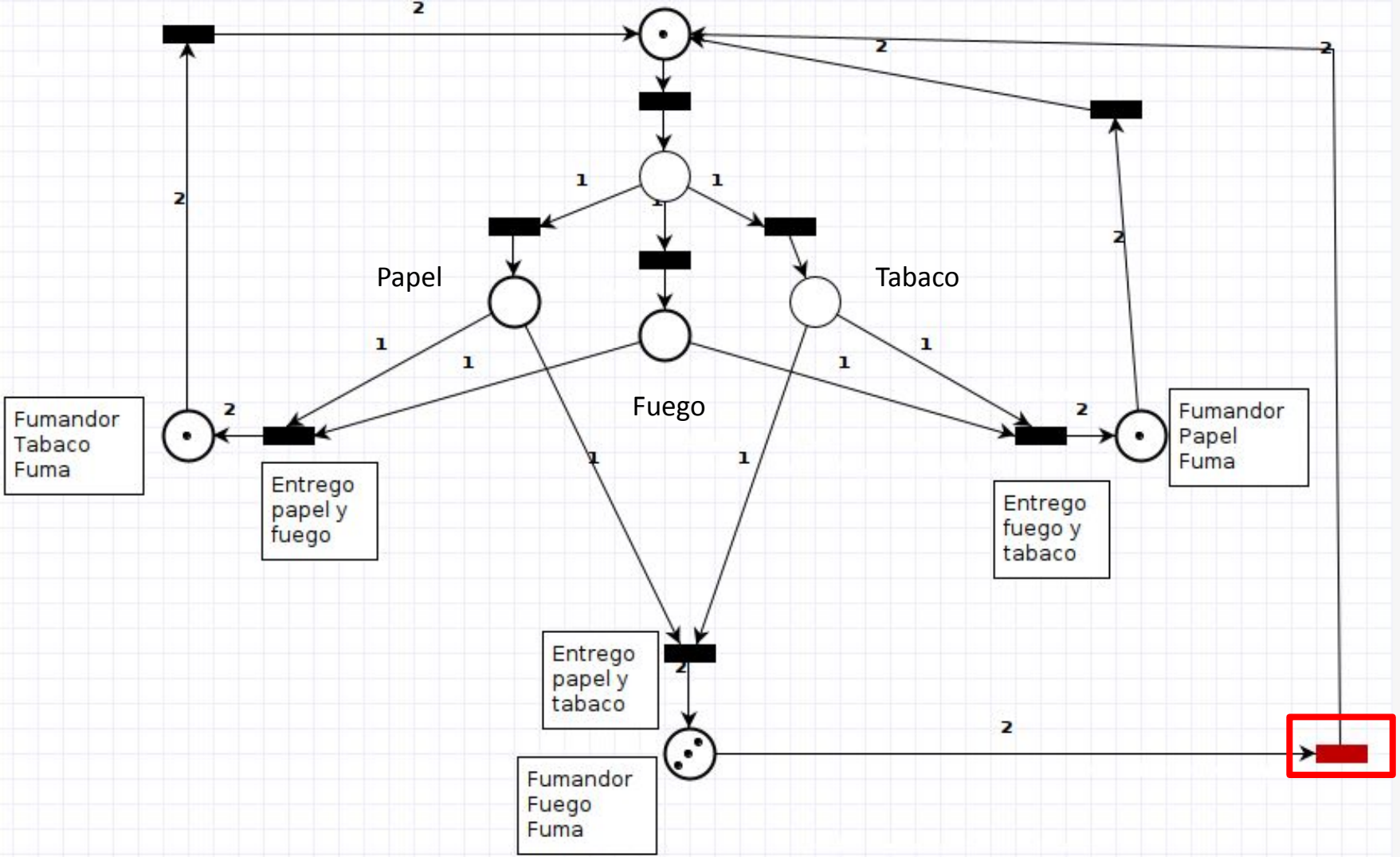

Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Entrego fuego y tabaco
Fumandor Papel Fuma
Entrego papel y tabaco
Fumandor Fuego Fuma

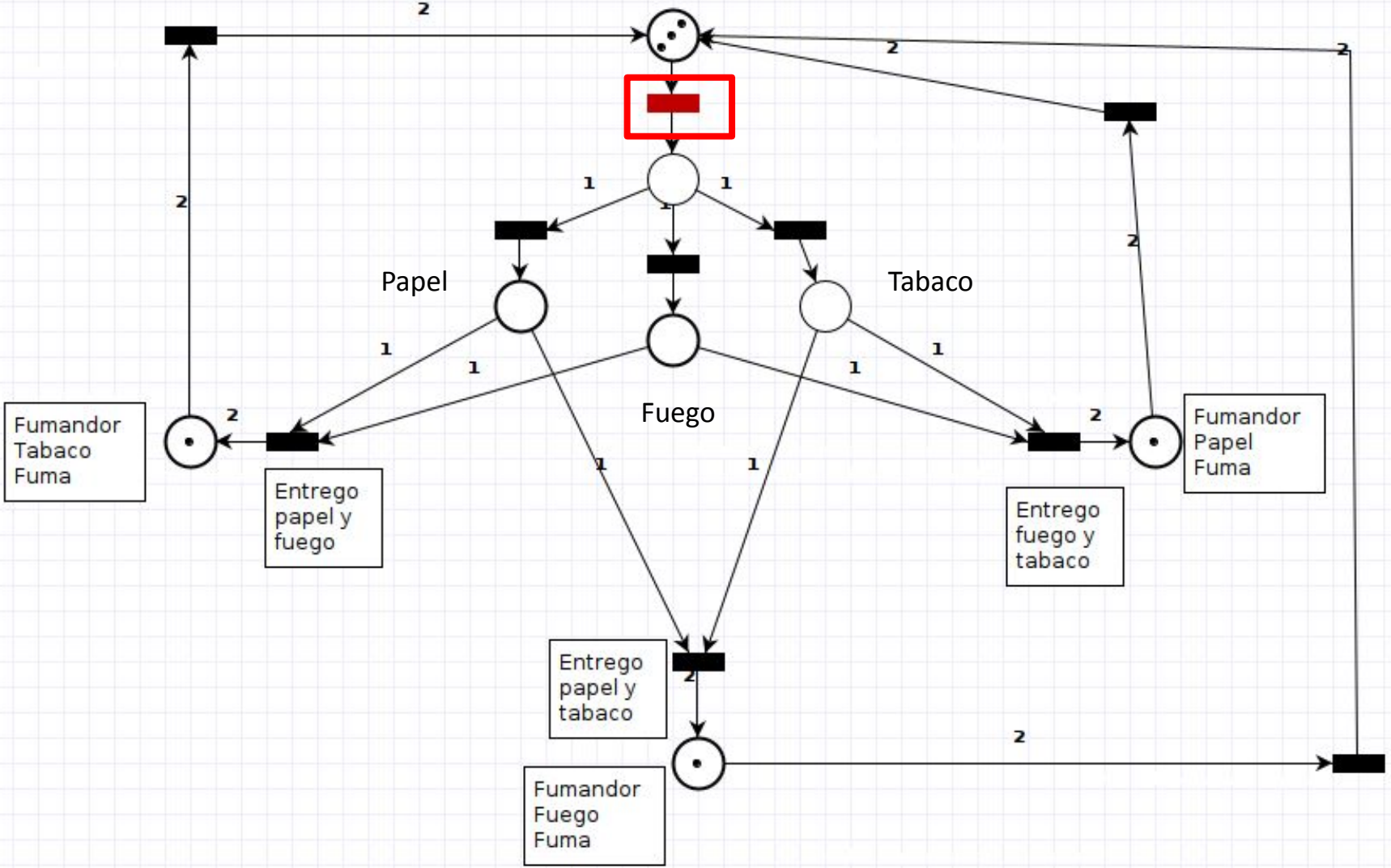

Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Fumandor Papel Fuma
Entrego fuego y tabaco
Entrego papel y tabaco
Fumandor Fuego Fuma


Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Entrego fuego y tabaco
Fumandor Papel Fuma
Entrego papel y tabaco
Fumandor Fuego Fuma


Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Fumandor Papel Fuma
Entrego fuego y tabaco
Entrego papel y tabaco
Fumandor Fuego Fuma


Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Entrego fuego y tabaco
Fumandor Papel Fuma
Entrego papel y tabaco
Fumandor Fuego Fuma


Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Entrego papel y tabaco
Entrego fuego y tabaco
Fumandor Papel Fuma
Fumandor Fuego Fuma


Papel
Tabaco
Fuego
Fumandor Tabaco Fuma
Entrego papel y fuego
Fumandor Papel Fuma
Entrego fuego y tabaco
Entrego papel y tabaco
Fumandor Fuego Fuma

Código en que nos basamos


```rust
let smokers:Vec<JoinHandle<()>> =  (0..N): Range<usize>
    .map(|i: usize|  {
        let agent_sem_smoker: Arc<Semaphore> = agent_sem.clone();
        let ingredient_sems_smoker: Arc<Vec<Semaphore>> = ingredient_sems.clone();
        thread::spawn(move || loop {
            let me = Ingredients::from_usize(i).unwrap();
            for ing_id: usize in 0..N {
                if ing_id != i {
                    let ing = Ingredients::from_usize(ing_id).unwrap();
                    println!("[Fumador {:?}] Esperando {:?}", me, ing);
                    ingredient_sems_smoker[ing_id].acquire();
                    println!("[Fumador {:?}] Obtuve {:?}", me, ing);
                }
            }
            println!("[Fumador {:?}] Fumando", me);
            thread::sleep(dur: Duration::from_secs(2));
            agent_sem_smoker.release();
            println!("[Fumador {:?}] Terminé", me);
        })
    }): impl Iterator<Item = JoinHandle<…>>
    .collect();
```

Código en que nos basamos

```
let pushers:Vec<JoinHandle<()>> =  (0..N): Range<usize>
    .map(|i: usize|  {
        let smoker_sems_pusher: Arc<Vec<Semaphore>> = smoker_sems.clone();
        let ingredient_sems_pusher: Arc<Vec<Semaphore>> = ingredient_sems.clone();
        let ingredients_found_pusher: Arc<RwLock<Vec<bool>>> = ingredients_found.clone();
        thread::spawn(move || loop {
            let me = Ingredients::from_usize(i).unwrap();
            println!("[Pusher {:?}] Esperando", me);
            ingredient_sems_pusher[i].acquire();
            println!("[Pusher {:?}] Mi ingrediente esta en la mesa", me);
            if let Ok(mut ings: RwLockWriteGuard<Vec<bool>>) = ingredients_found_pusher.write() {
                let mut pushed: bool = false;
                for smoker_id: usize in 0..N {
                    let mut all_req: bool= true;
                    for ing_id: usize in 0..N {
                        if ing_id != smoker_id && ing_id != i {
                            all_req = all_req && ings[ing_id]
                        }
                    }
                    if all_req {
                        println!("[Pusher {:?}] Despertando a {:?}", me, Ingredients::from_usize(smoker_id).unwrap());
                        smoker_sems_pusher[smoker_id].release();
                        for clean: usize in 0..N {
                            ings[clean] = false;
                        }
                        pushed = true;
                        break;
                    }
                }
                if !pushed {
                    println!("[Pusher {:?}] Lo pongo en el tablero", me);
                    ings[i] = true;
                }
            }
        })
    }): impl Iterator<Item = JoinHandle<_>>
    .collect();
```

Problema de los lectores y escritores

Queremos resolver el acceso concurrente a recursos de la mejor manera posible. Esto es, permitir múltiples accesos read-only de forma (casi) no bloqueante. Mientras que para el caso de los accesos de escritura queremos hacerlo de forma protegida (atómica)

Process 1	Process 2	Allowed / Not Allowed
Writing	Writing	Not Allowed
Reading	Writing	Not Allowed
Writing	Reading	Not Allowed
Reading	Reading	Allowed

Código en que nos basamos

```
thread::spawn(move || loop {
    let (lock, cvar) = &*pair_reader;

    {
        let mut _guard = cvar.wait_while(lock.lock().unwrap(), |state| {
            state.writing
        }).unwrap();
        _guard.readers += 1;
    }

    data_read();

    lock.lock().unwrap().readers -= 1;
    cvar.notify_all();
})
```

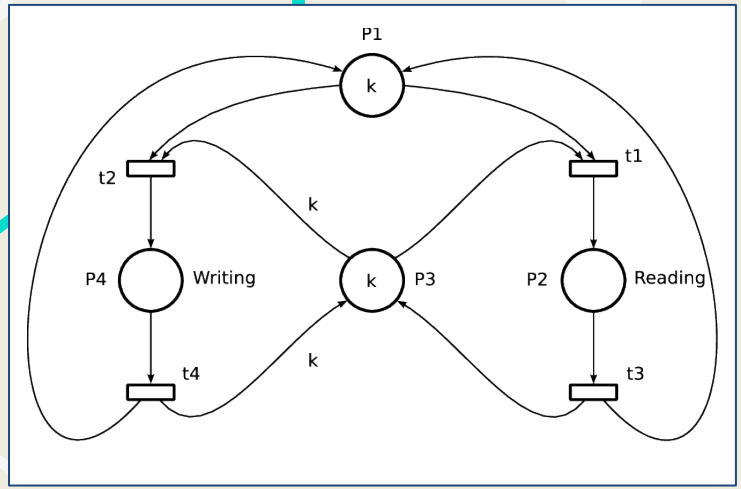
Reader thread

```
thread::spawn(move || loop {
    return;
    let (lock, cvar) = &*pair_writer;

    {
        let mut _guard = cvar.wait_while(lock.lock().unwrap(), |state| {
            state.writing || state.readers > 0
        }).unwrap();
        _guard.writing = true;
    }

    data_write();

    lock.lock().unwrap().writing = false;
    cvar.notify_all();
})
```

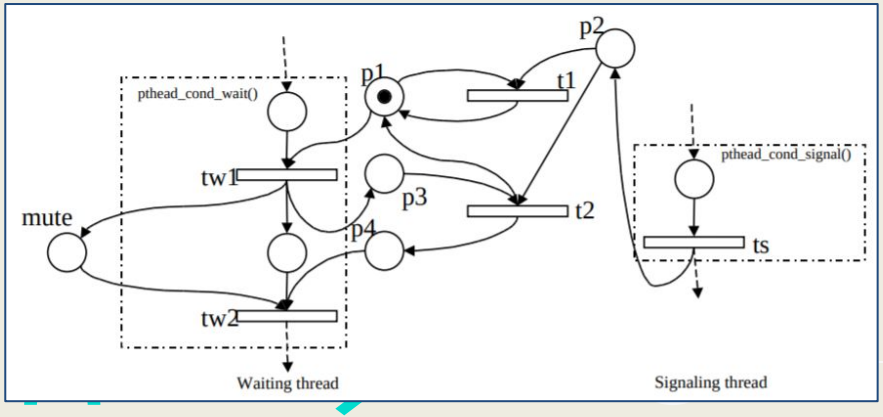
Writer thread

Condition

Bloques de construcción



Condición entre readers y writers



Modelo de condvar por thread

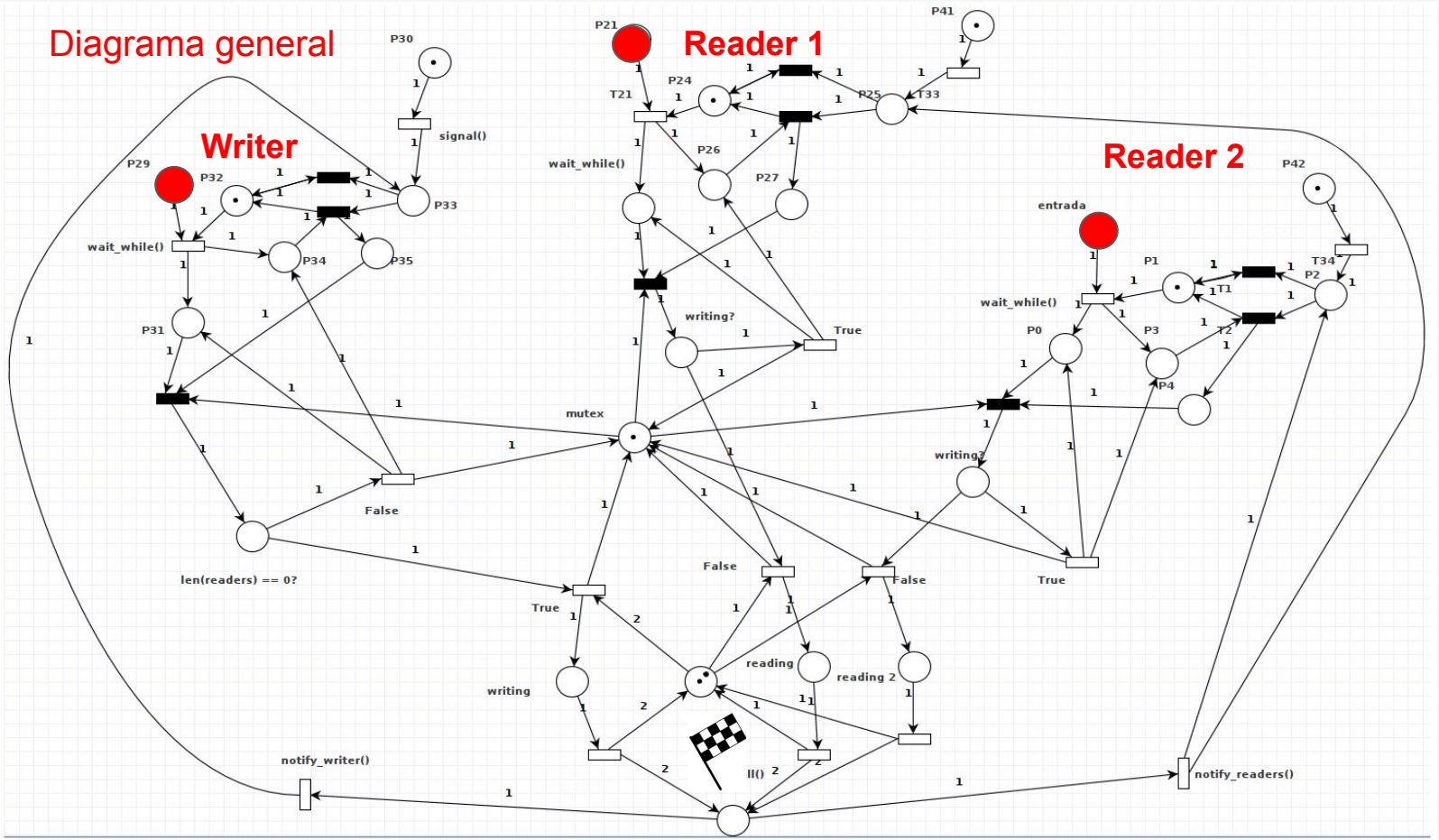
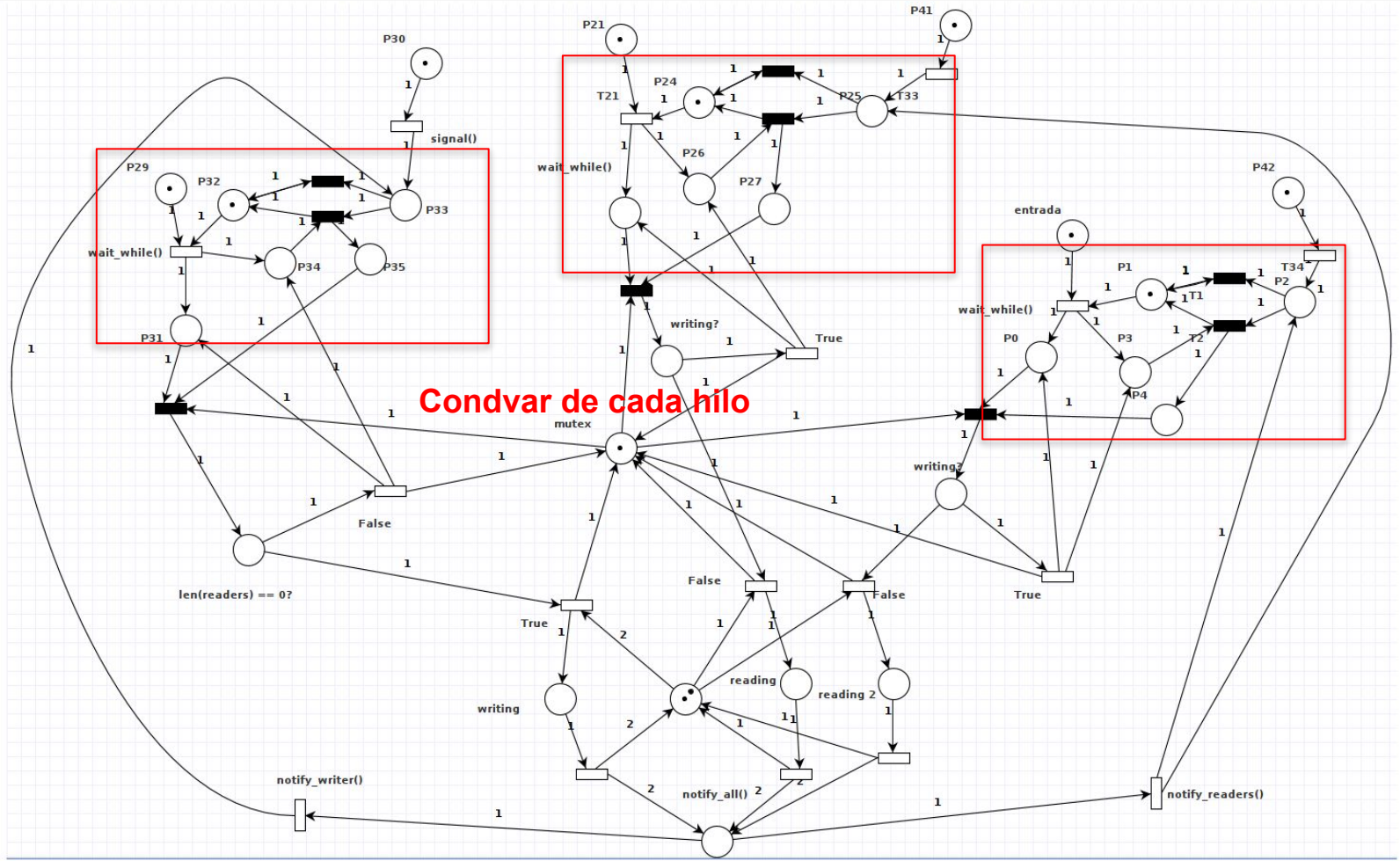
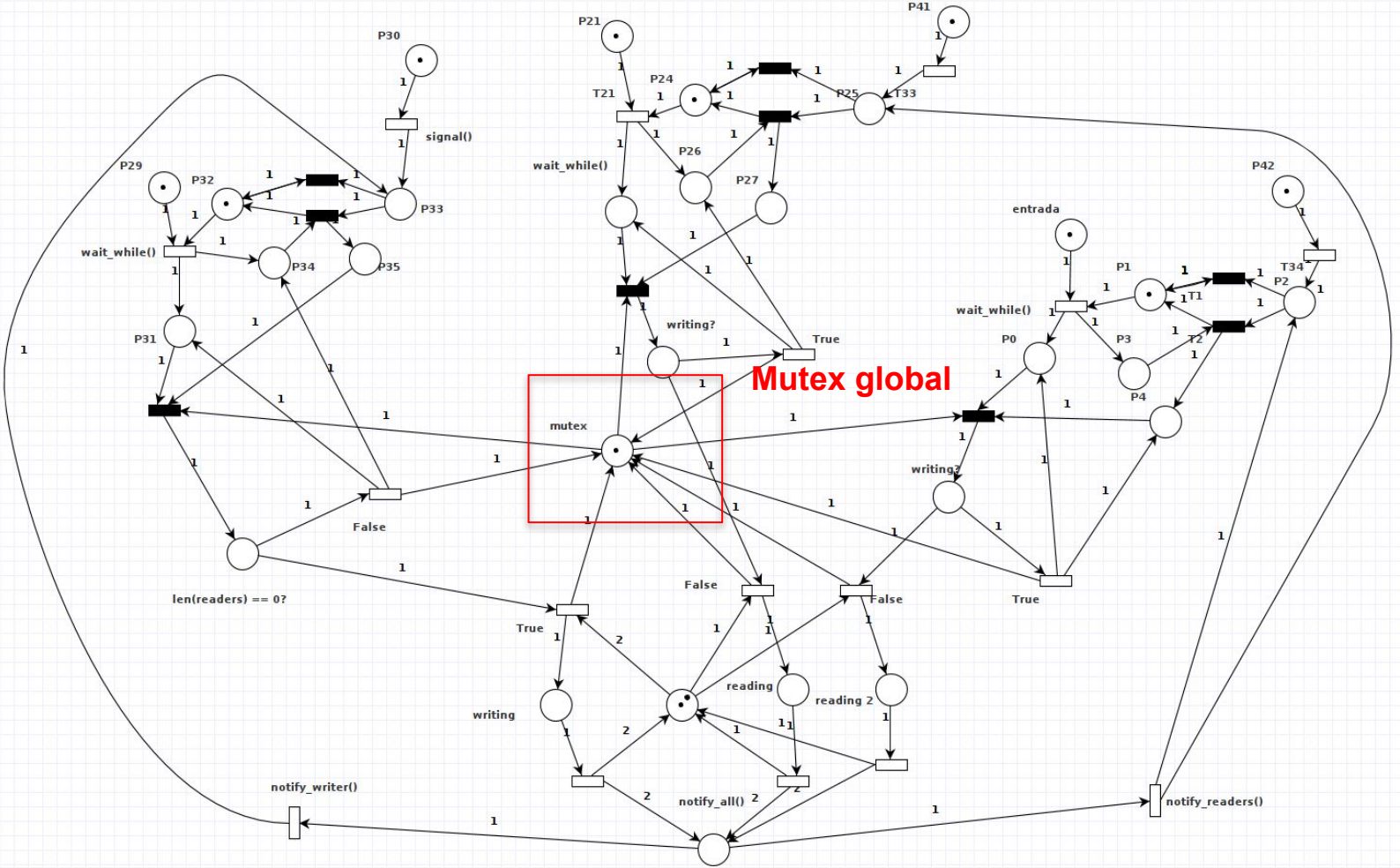
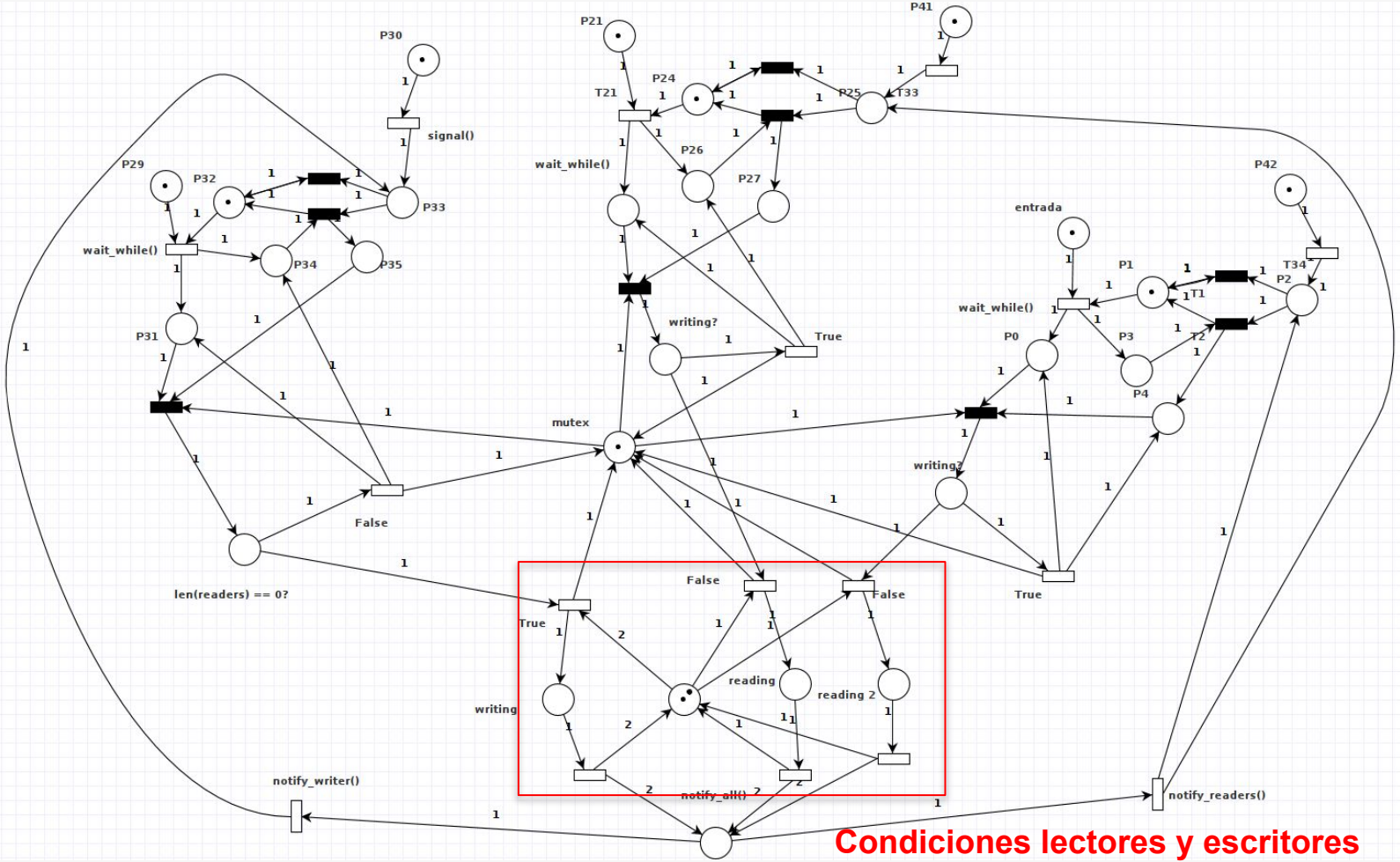


Diagrama general
Writer
Reader 1
Reader 2
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
ll()
notify_readers()
entrada
P1
T1
P2
T34
P42
wait_while()
P0
P3
T2
P4
writing?
True
False

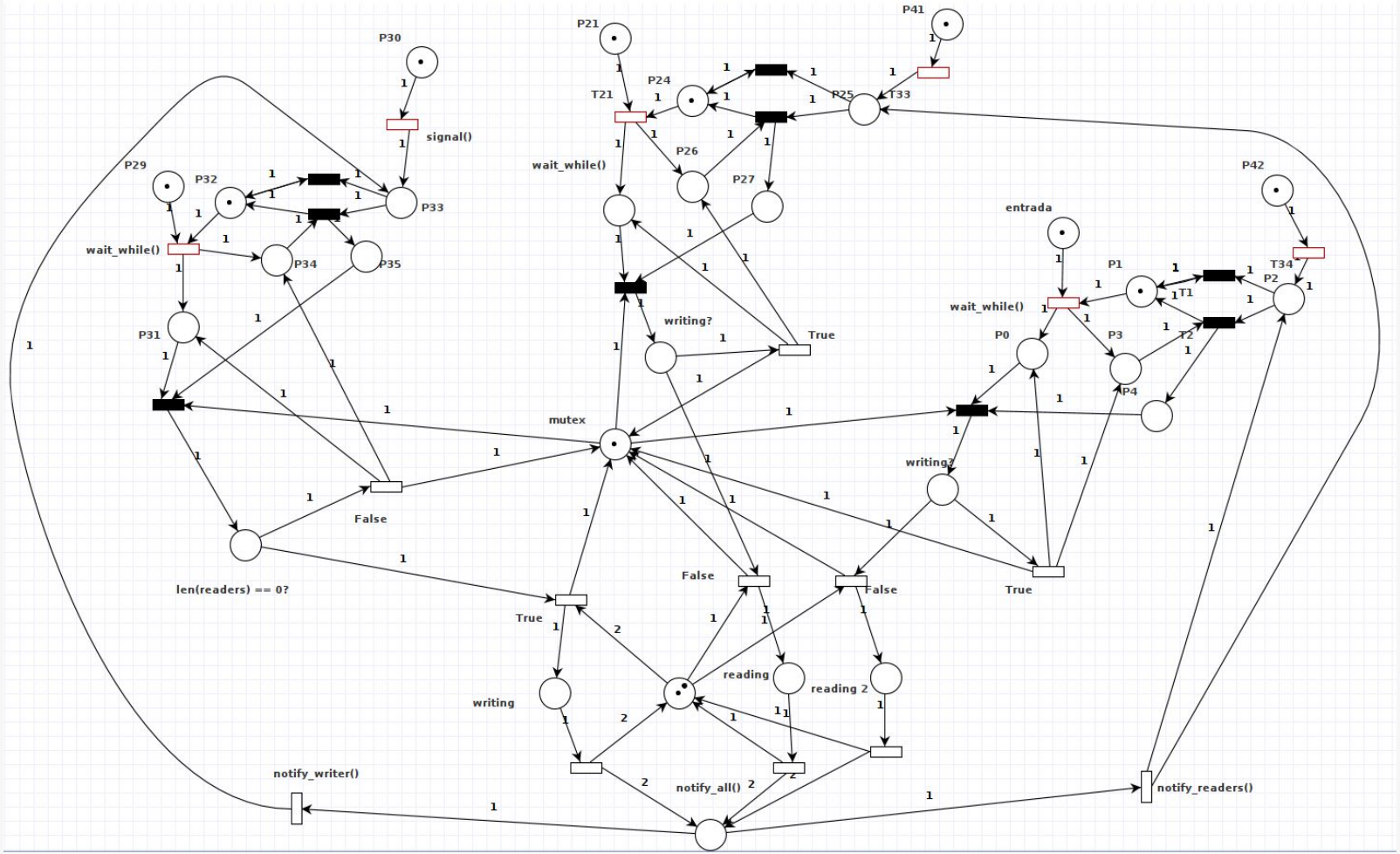

Condvar de cada hilo
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_all()
notify_writer()
notify_readers()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?

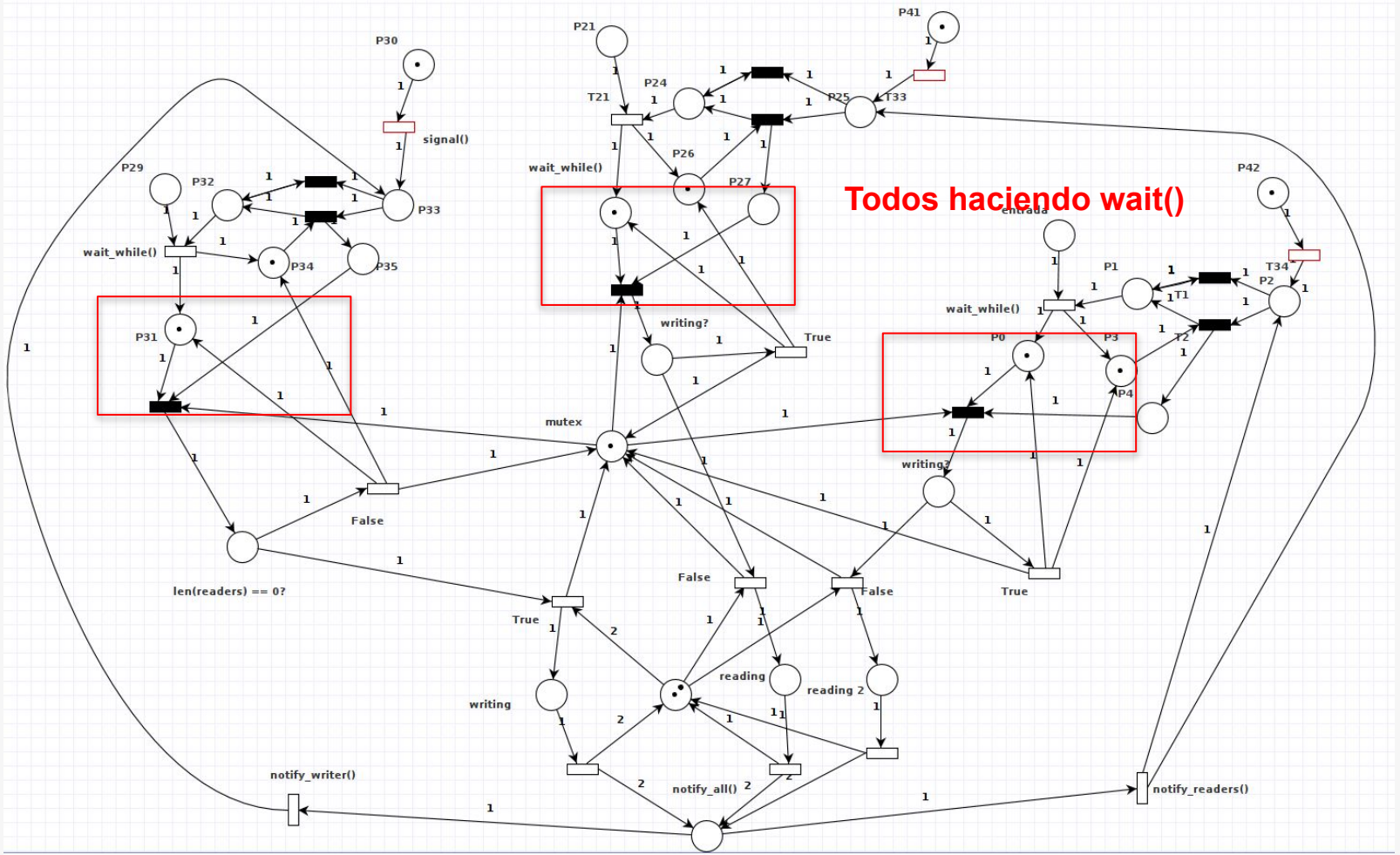

Mutex global
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?

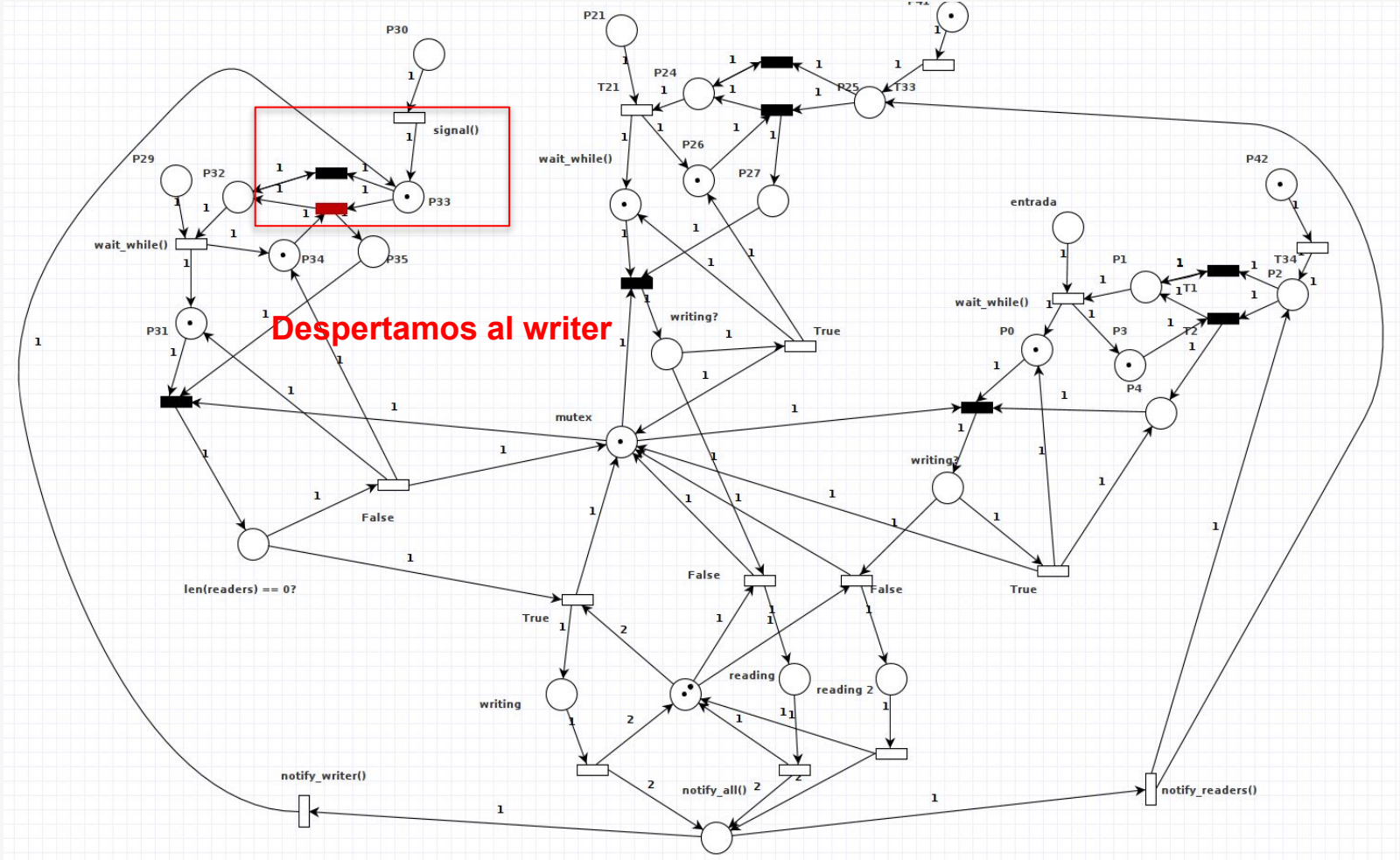

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
False
True
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?
True
Condiciones lectores y escritores

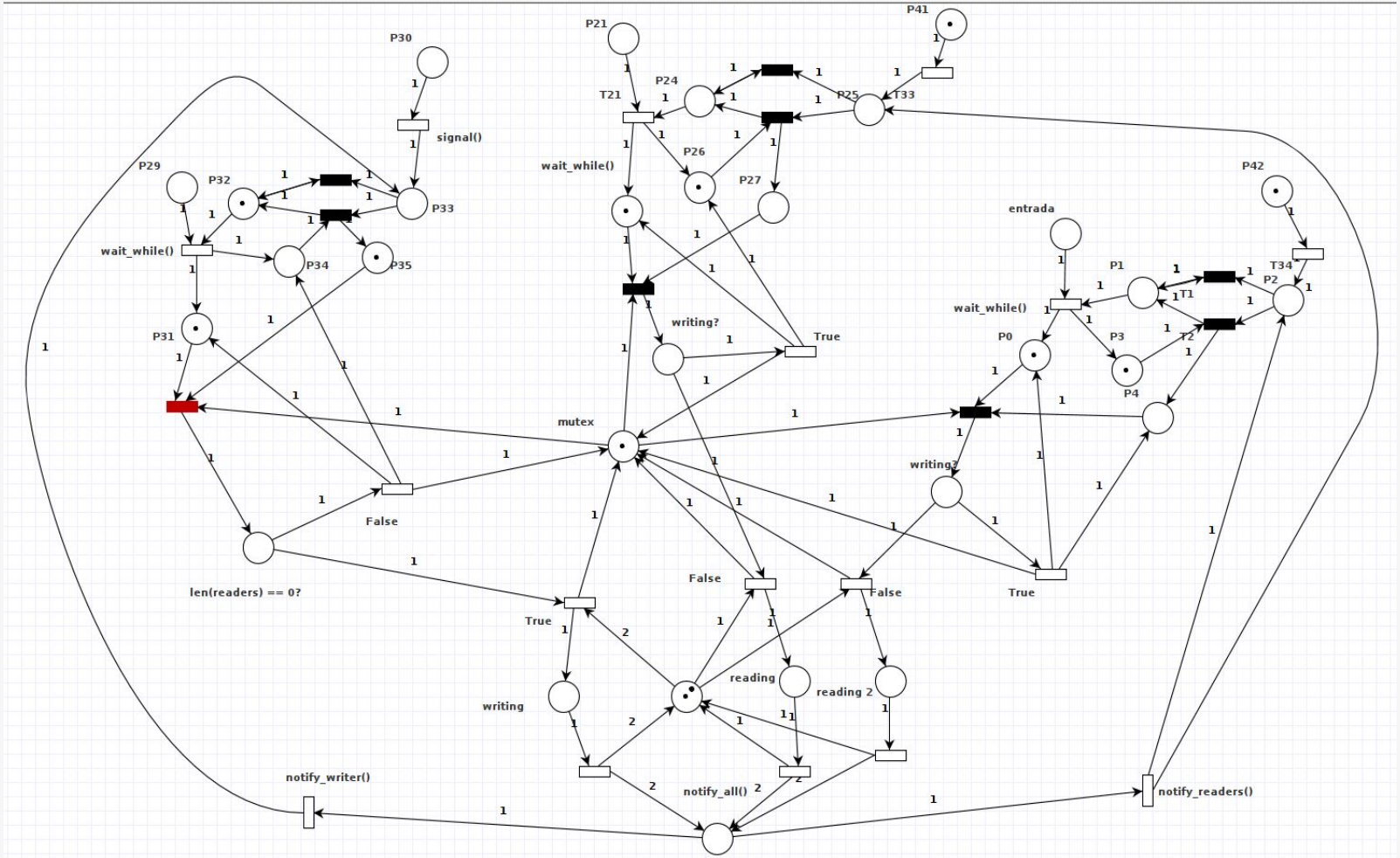
Veamos algunos casos

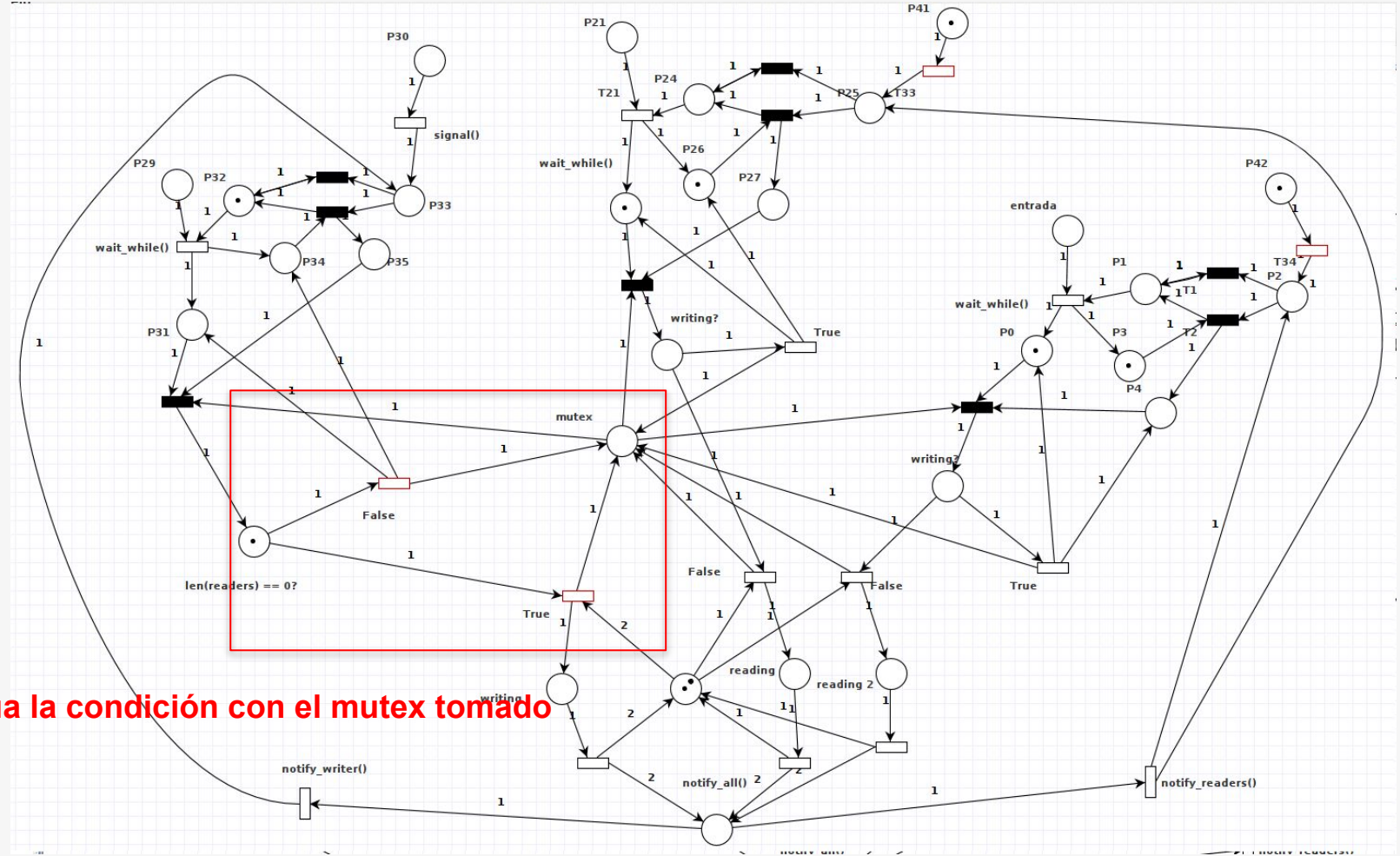
Writer → Reader 1 → Reader 2


P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
True
writing
reading
reading 2
notify_writer()
notify_all()
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
True
False
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing?
notify_readers()

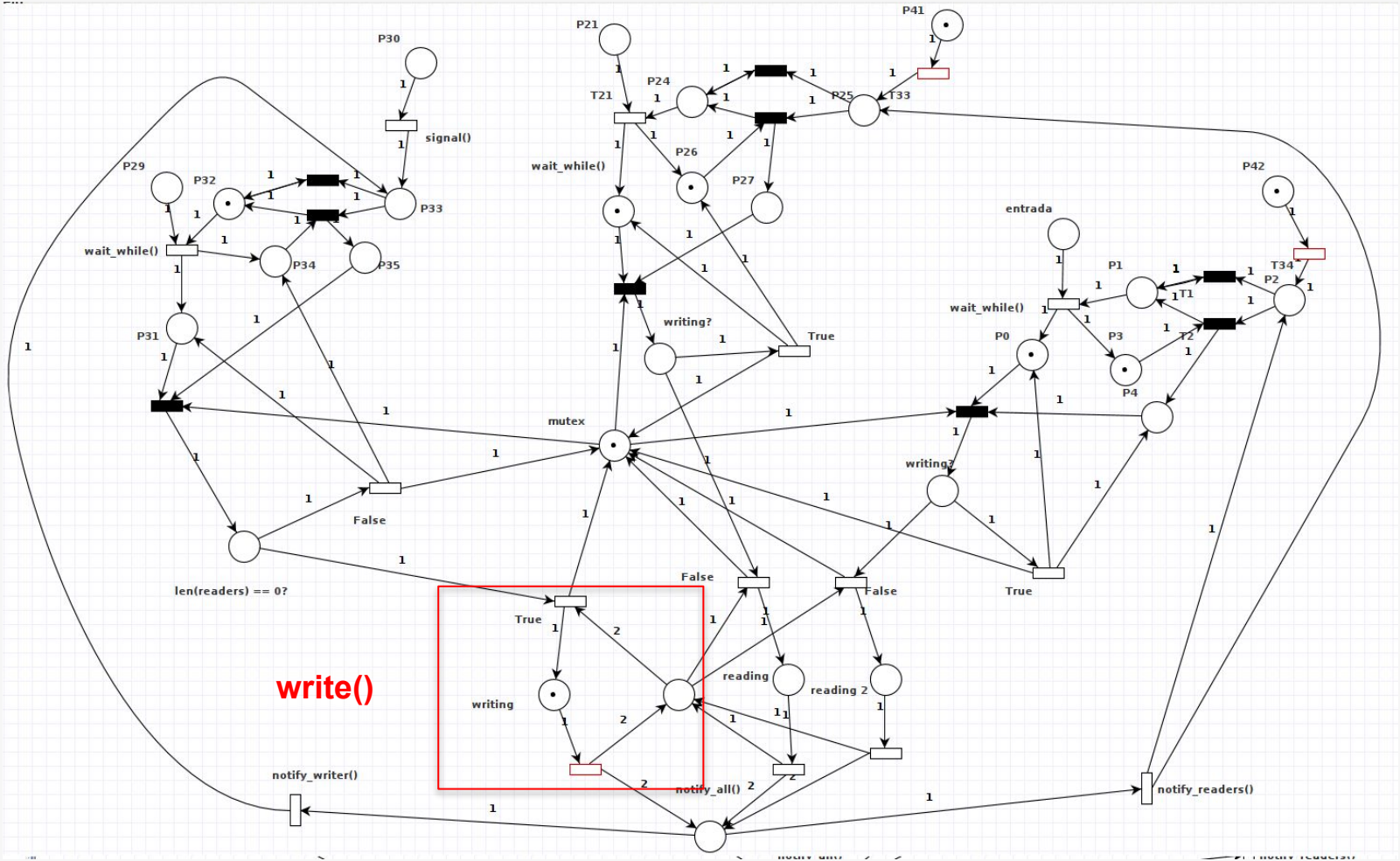

Todos haciendo wait()
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P24
T21
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing2

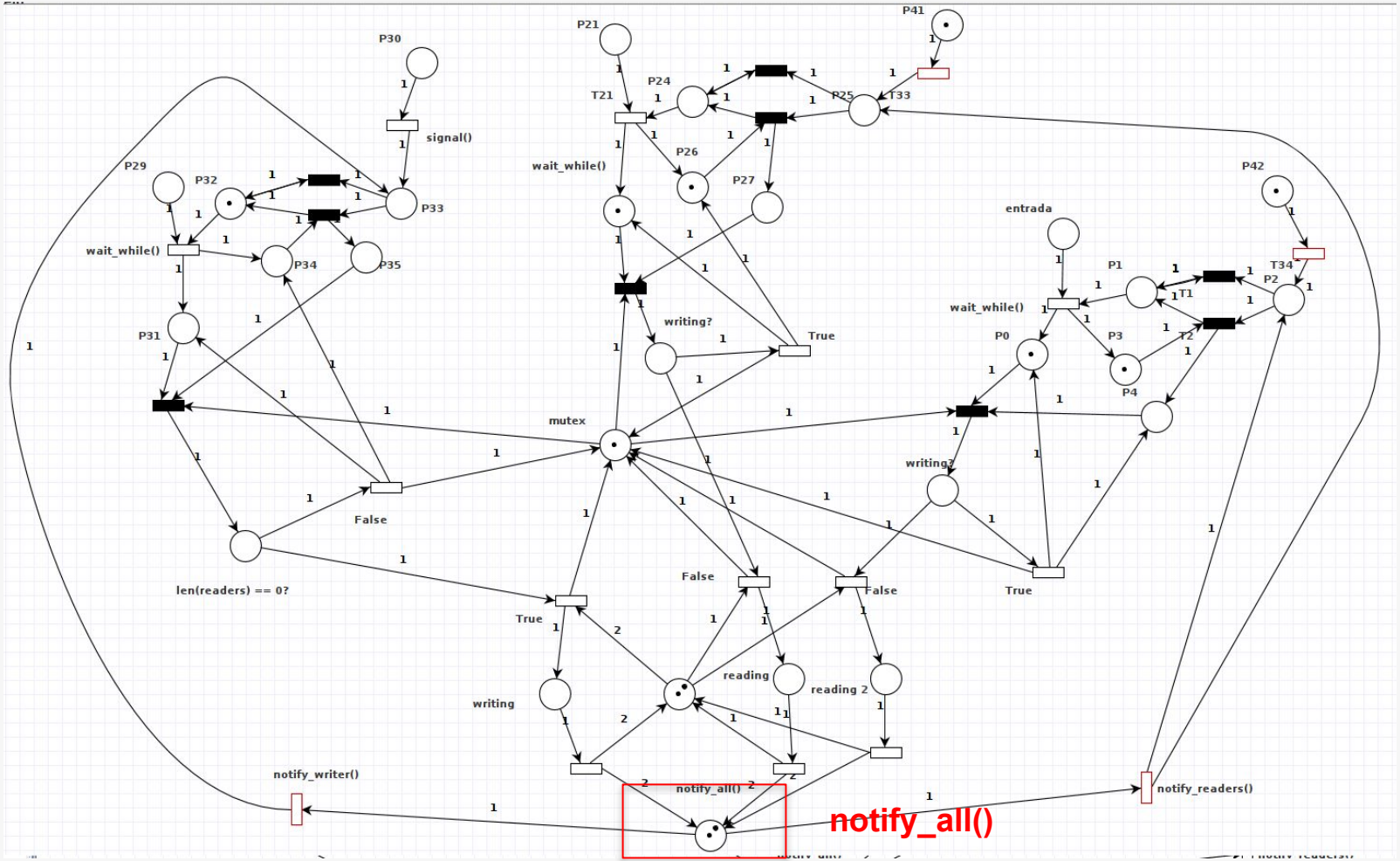

Despertamos al writer
P30
signal()
P33
P29
P32
wait_while()
P34
P35
P31
P21
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing?

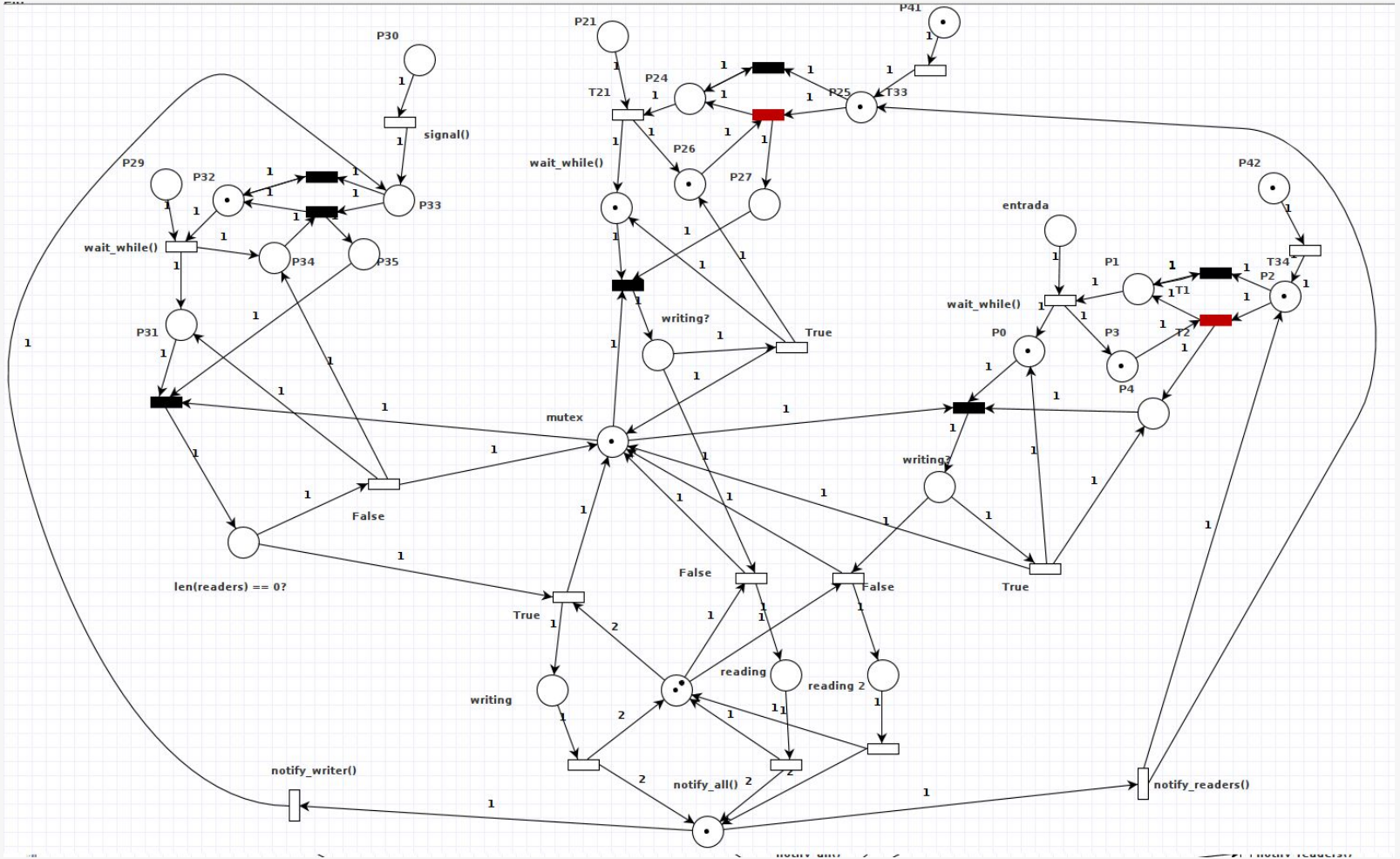

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?

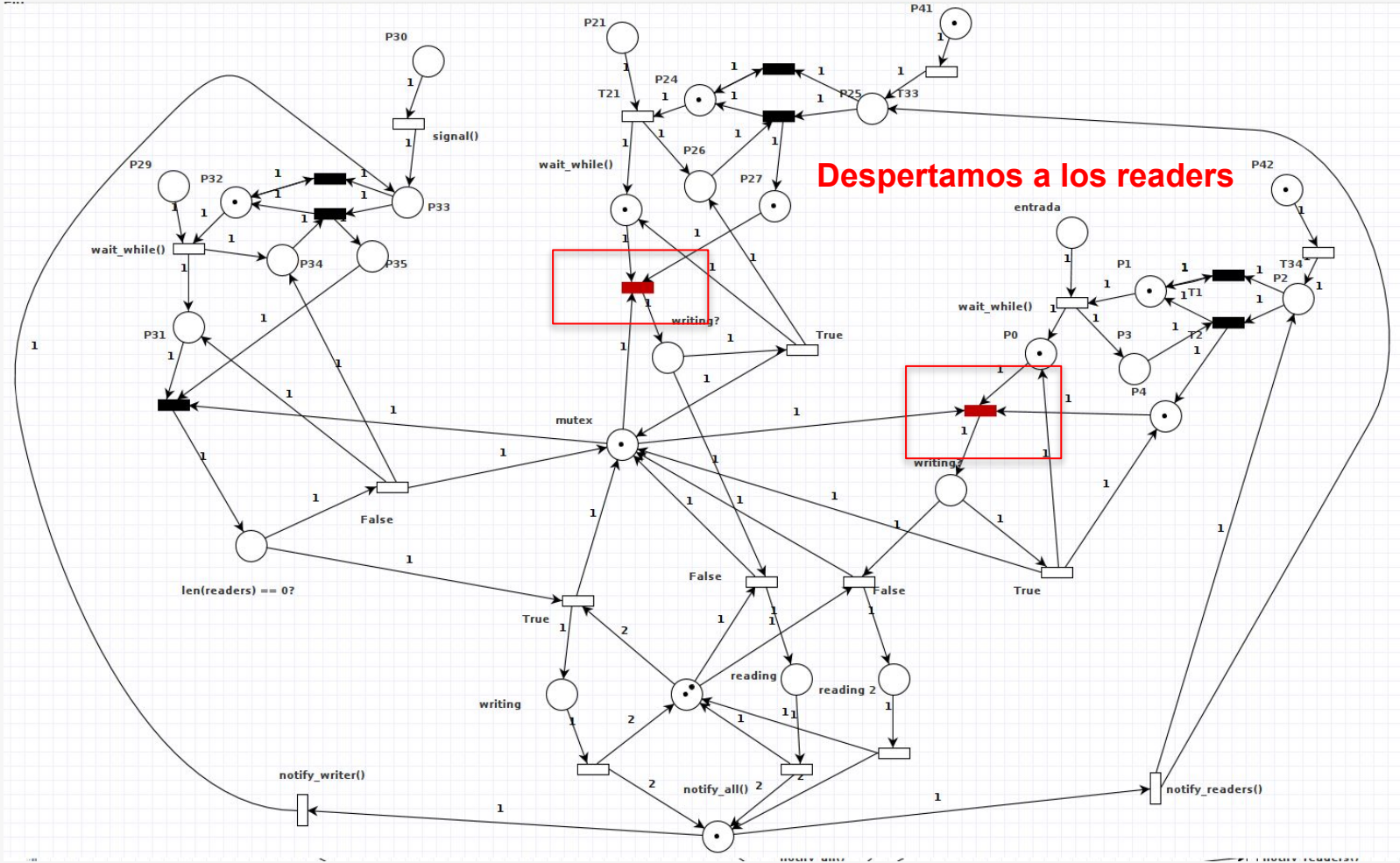


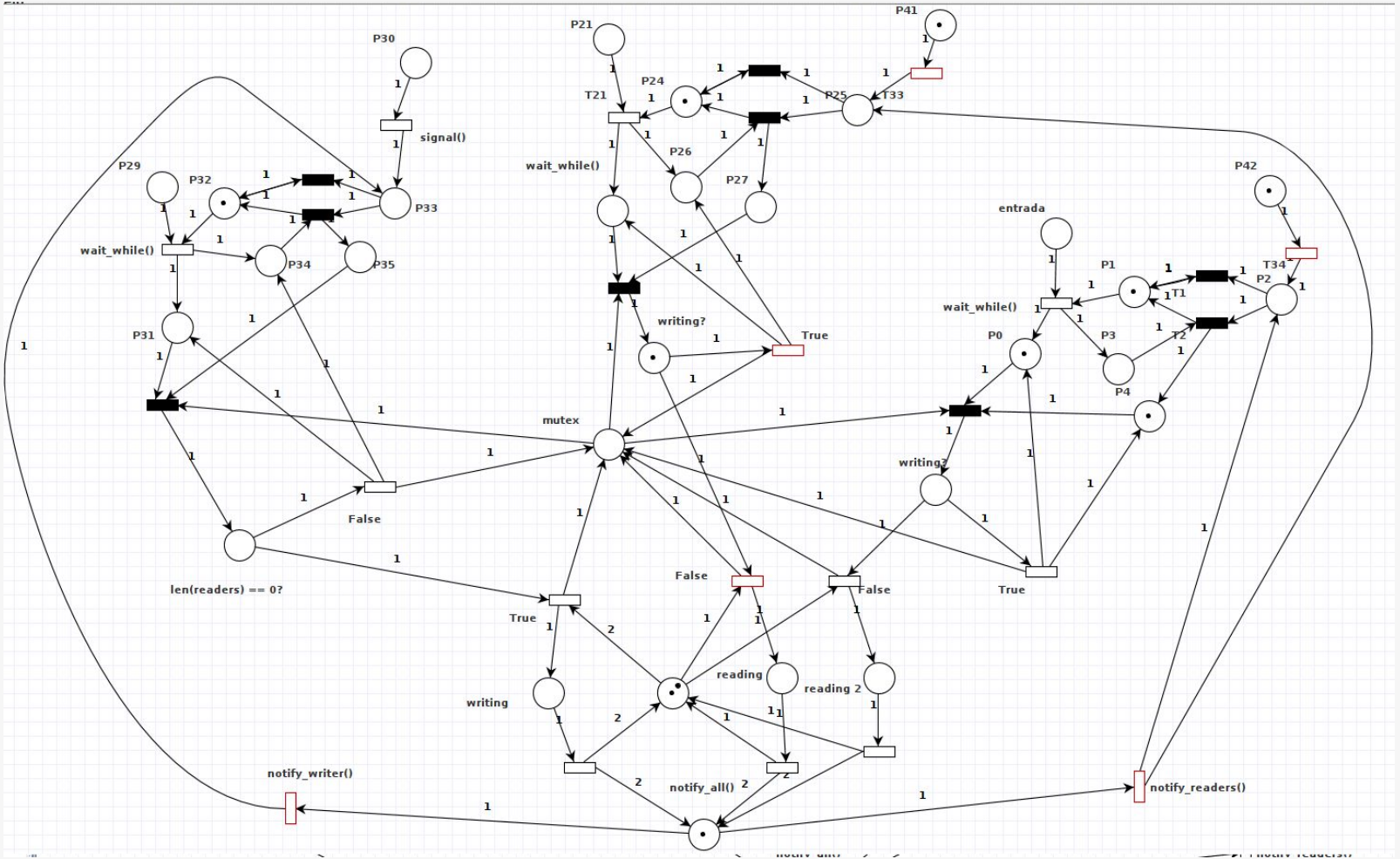
Evalua la condición con el mutex tomado

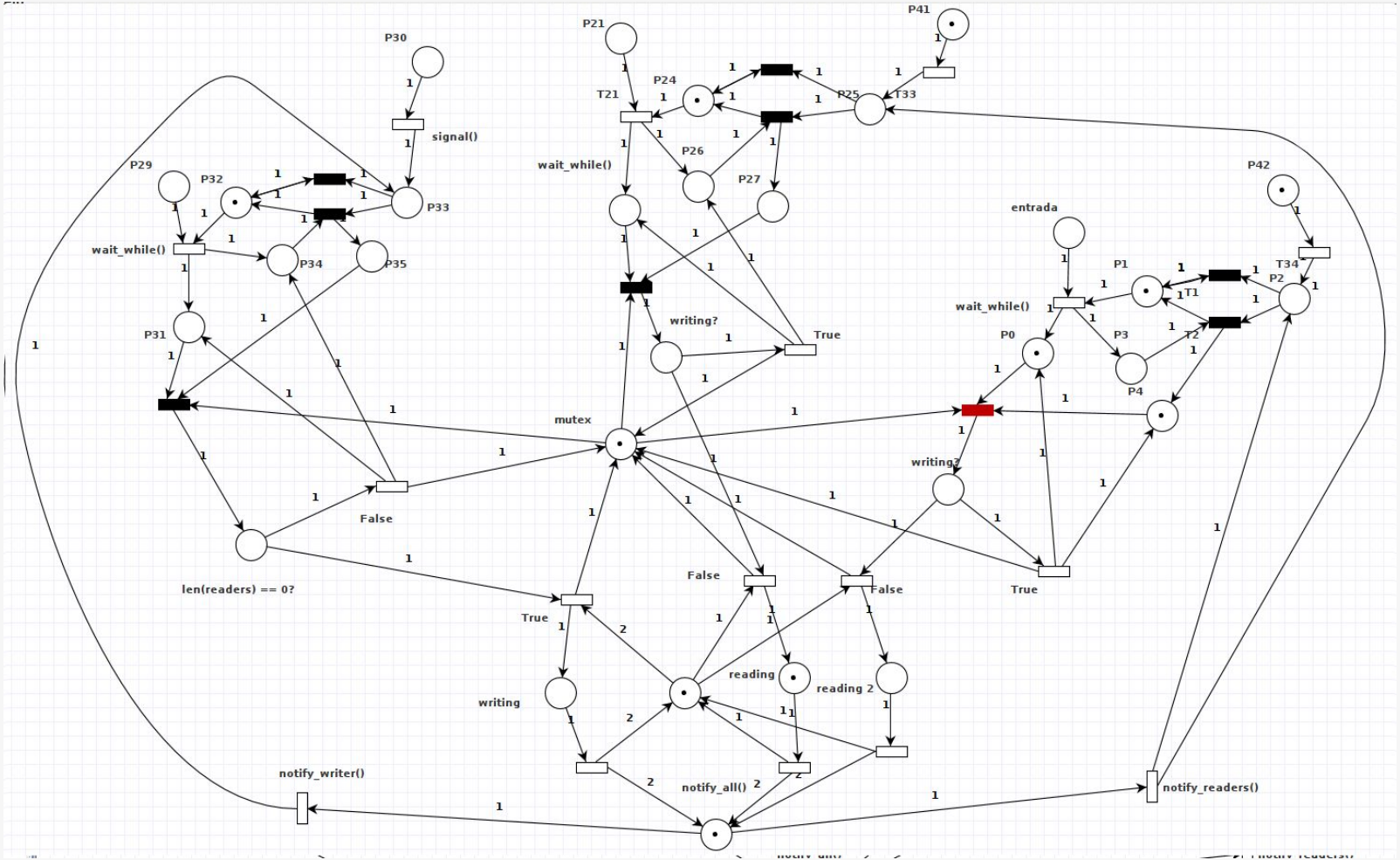

write()
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
True
False
writing
reading
reading 2
notify_writer()
notify_all()
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing?
False
True
notify_readers()

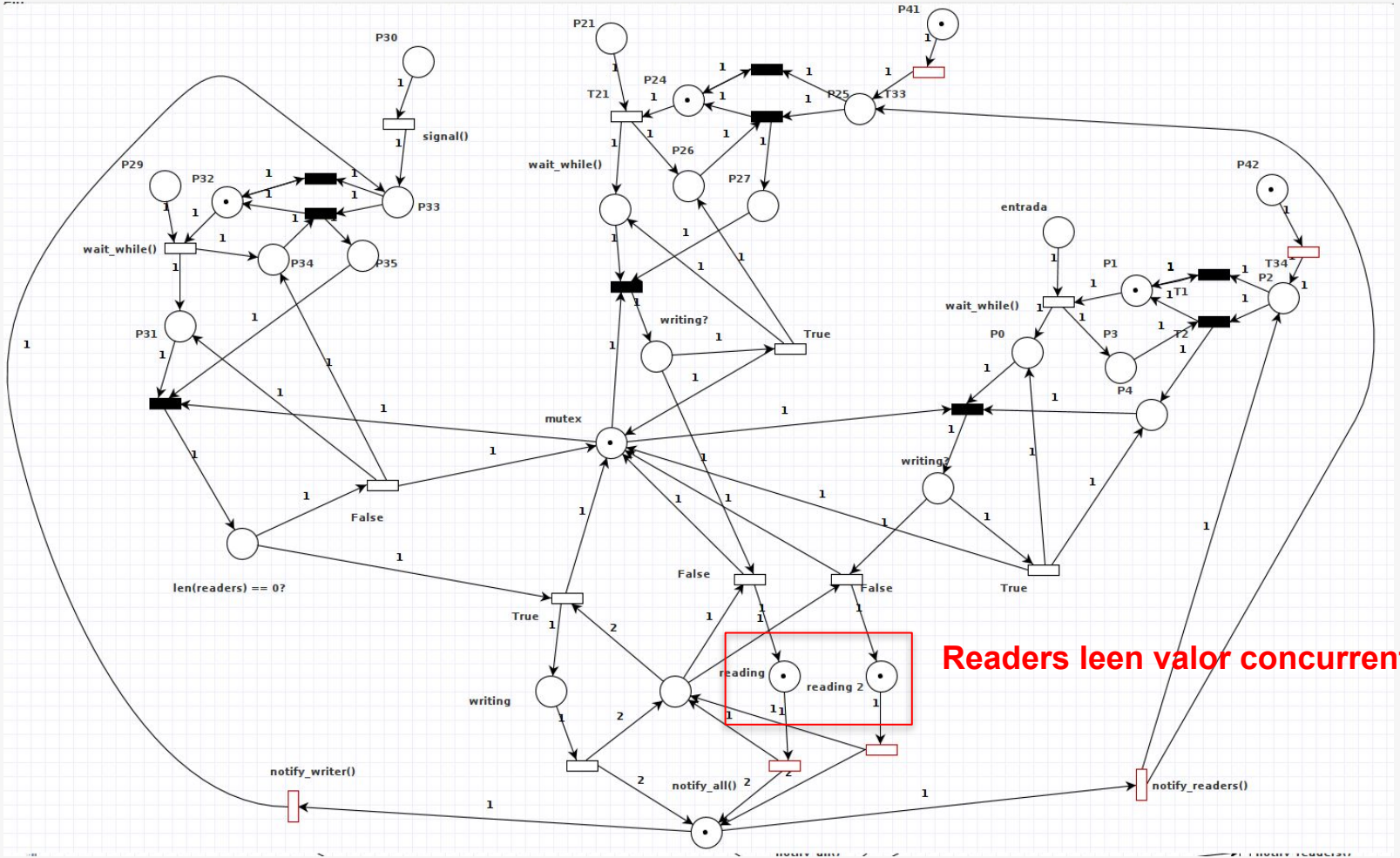

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
T34
P1
P2
T1
wait_while()
P0
P3
T2
P4
writing?
notify_all()

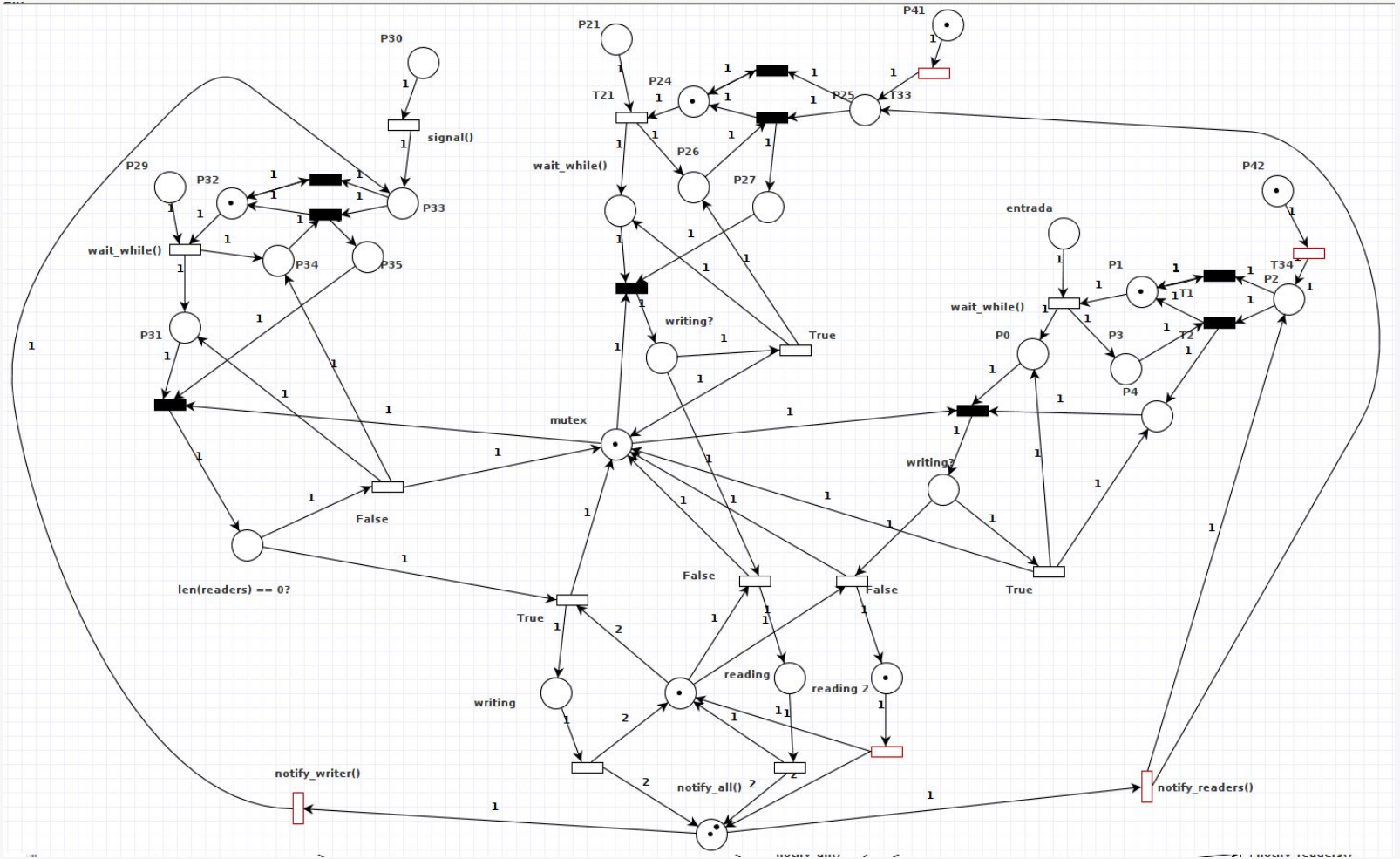

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
True
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
False
True
writing
reading
reading 2
notify_all()
notify_writer()
notify_readers()
entrada
P1
T1
P2
T34
P42
wait_while()
P0
P3
T2
P4
writing?
True
False


Despertamos a los readers
P30
signal()
P21
P41
T21
P24
P25
T33
P26
P27
wait_while()
P29
P32
P33
wait_while()
P34
P35
P31
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing2

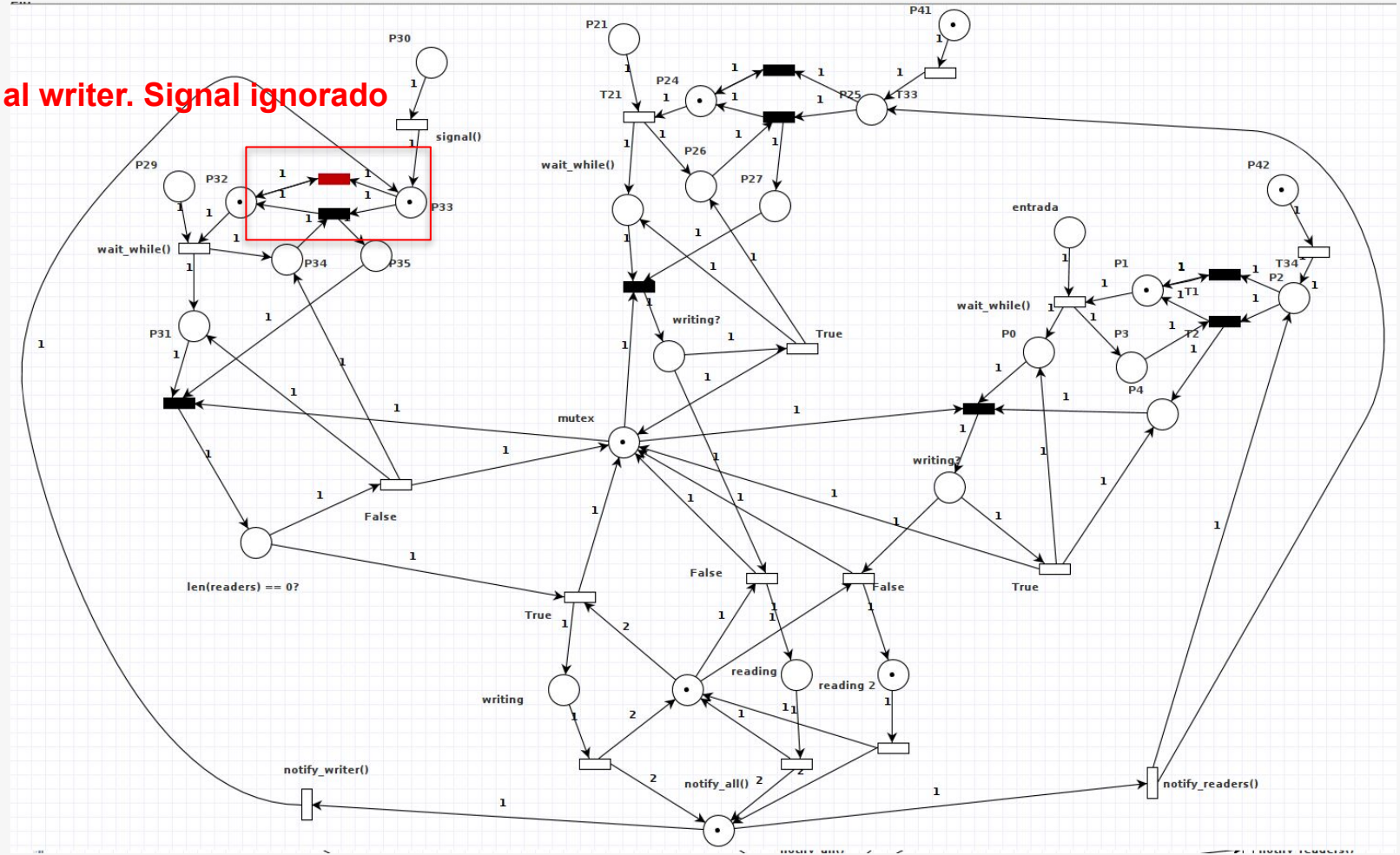

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
T34
P1
P2
T1
wait_while()
P0
P3
T2
P4
writing2

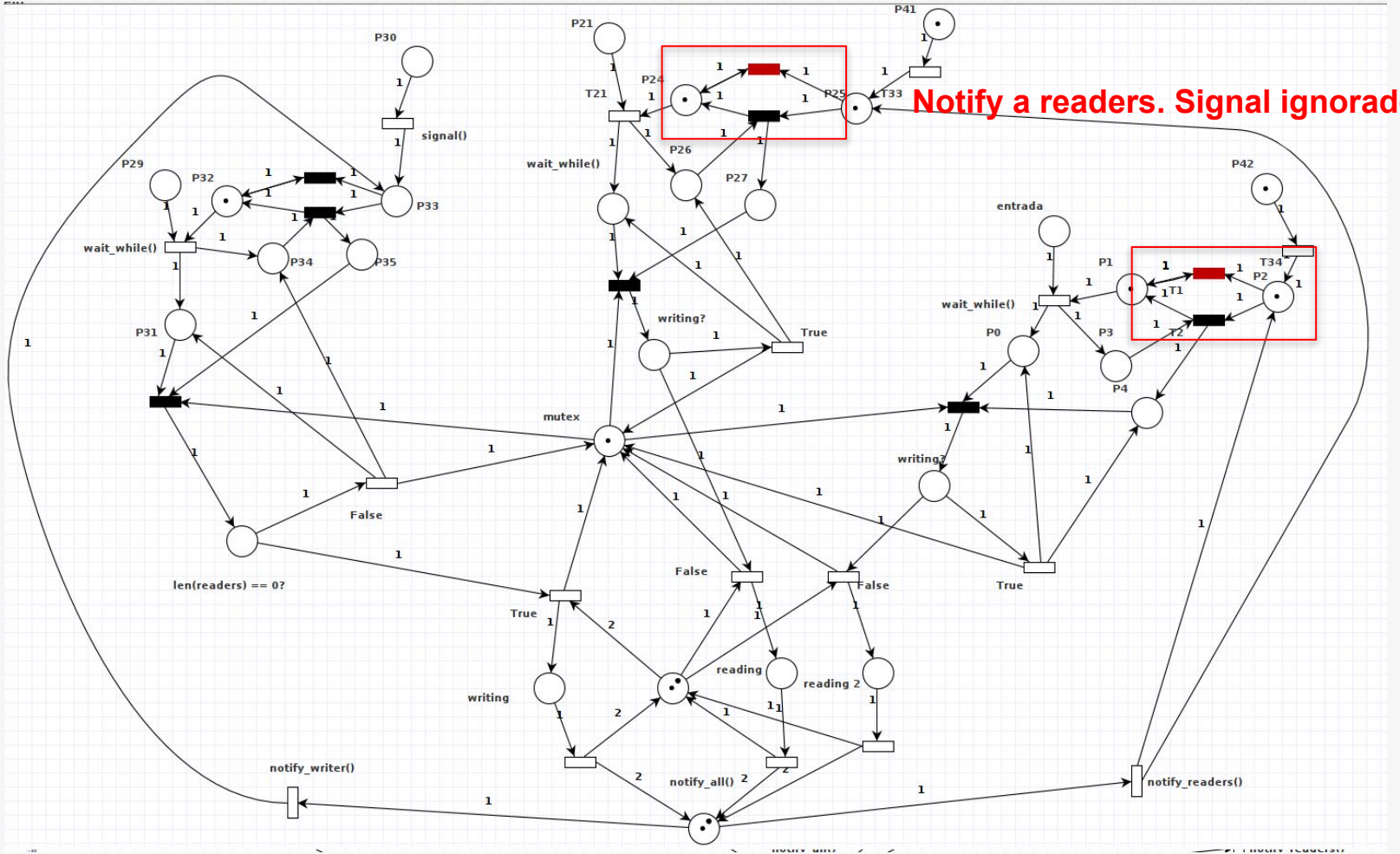

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
P24
T21
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
T34
P1
P2
T1
wait_while()
P0
P3
T2
P4
writing?

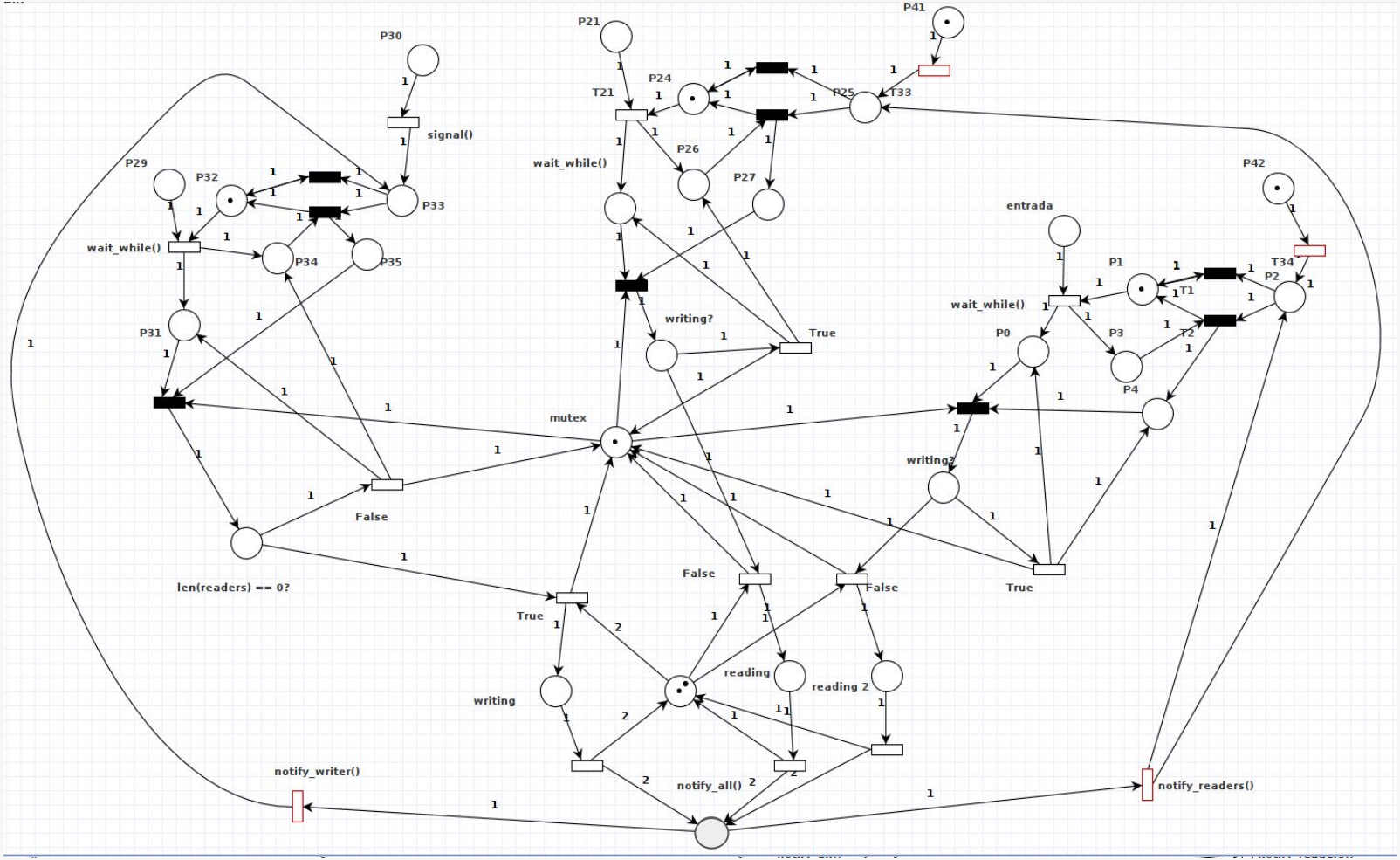

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T1
T34
P2
wait_while()
P0
P3
T2
P4
writing?
Readers leen valor concurrentemente


P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
True
writing
notify_writer()
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
reading
reading 2
notify_all()
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing?
False
True
notify_readers()

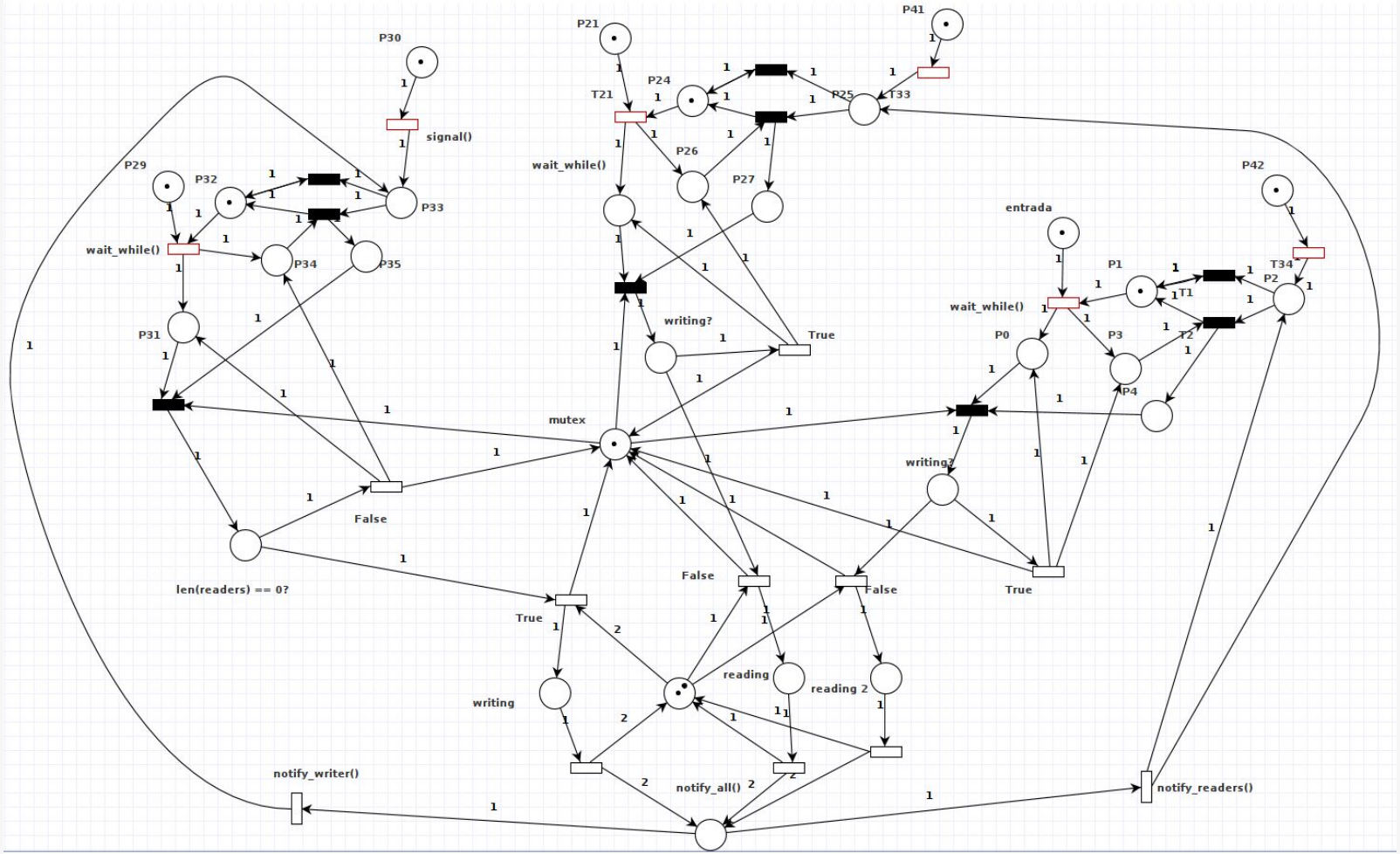
Notify al writer. Signal ignorado

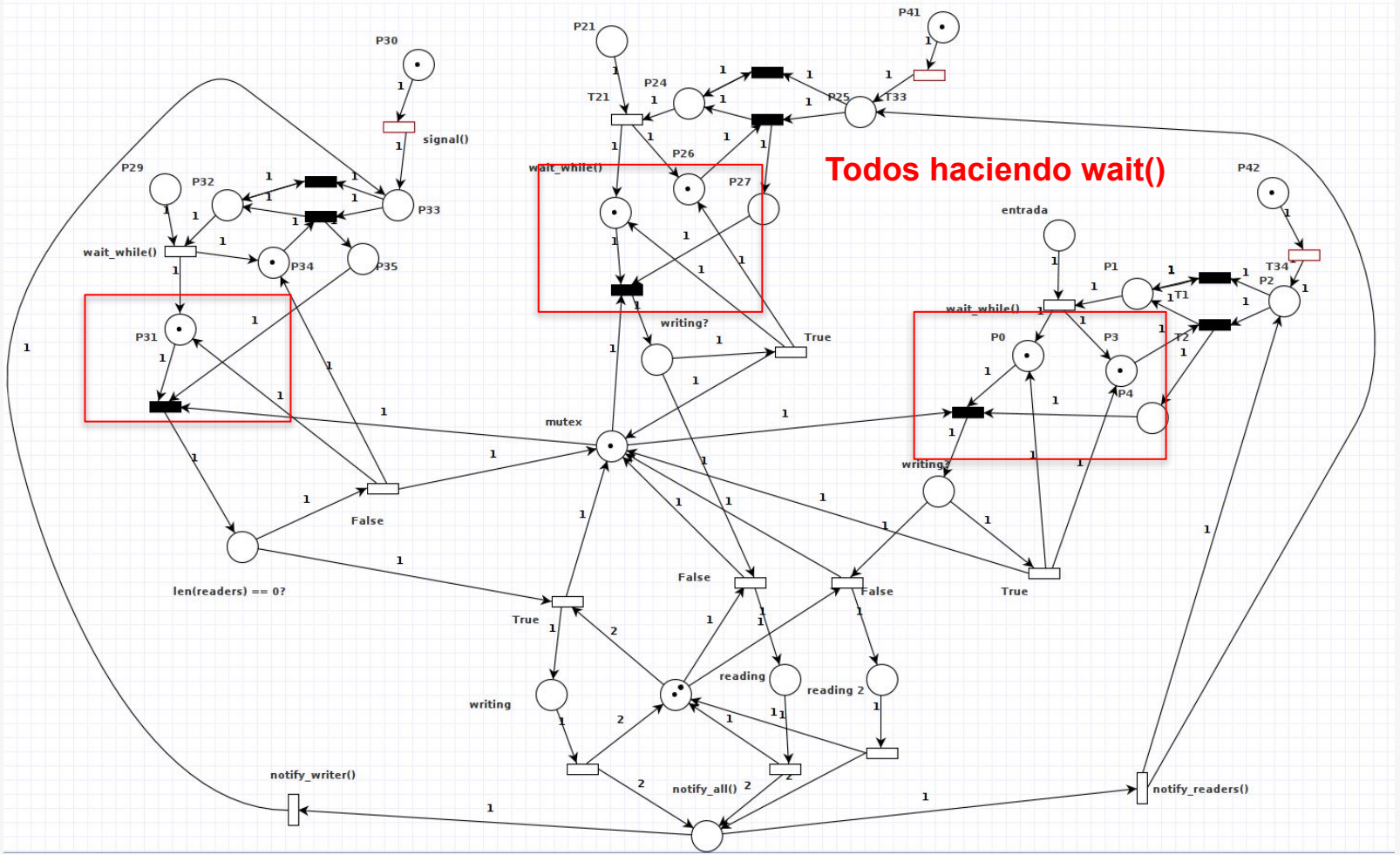


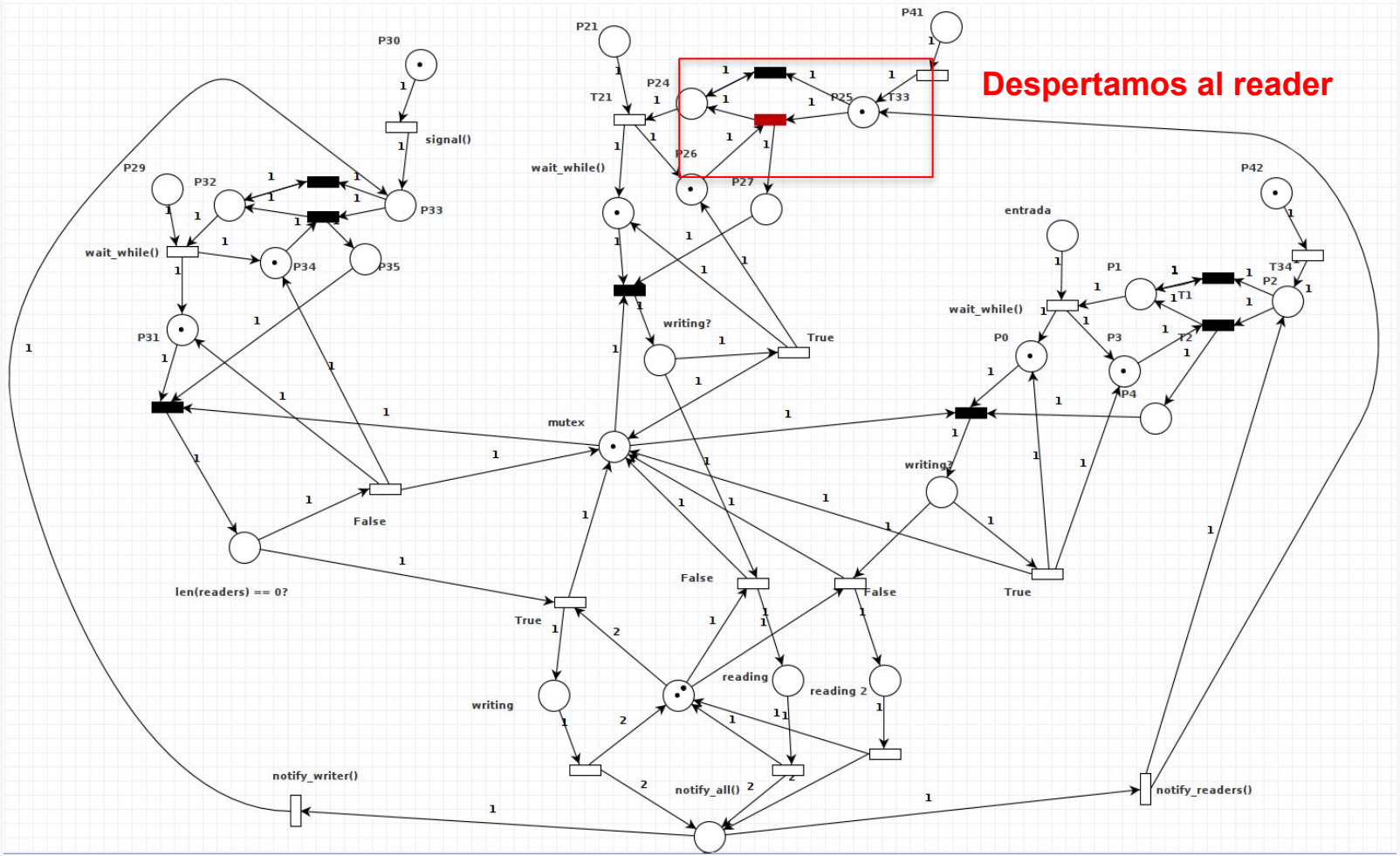

Notify a readers. Signal ignorado
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P24
T21
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
reading
reading 2
writing
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing?
True

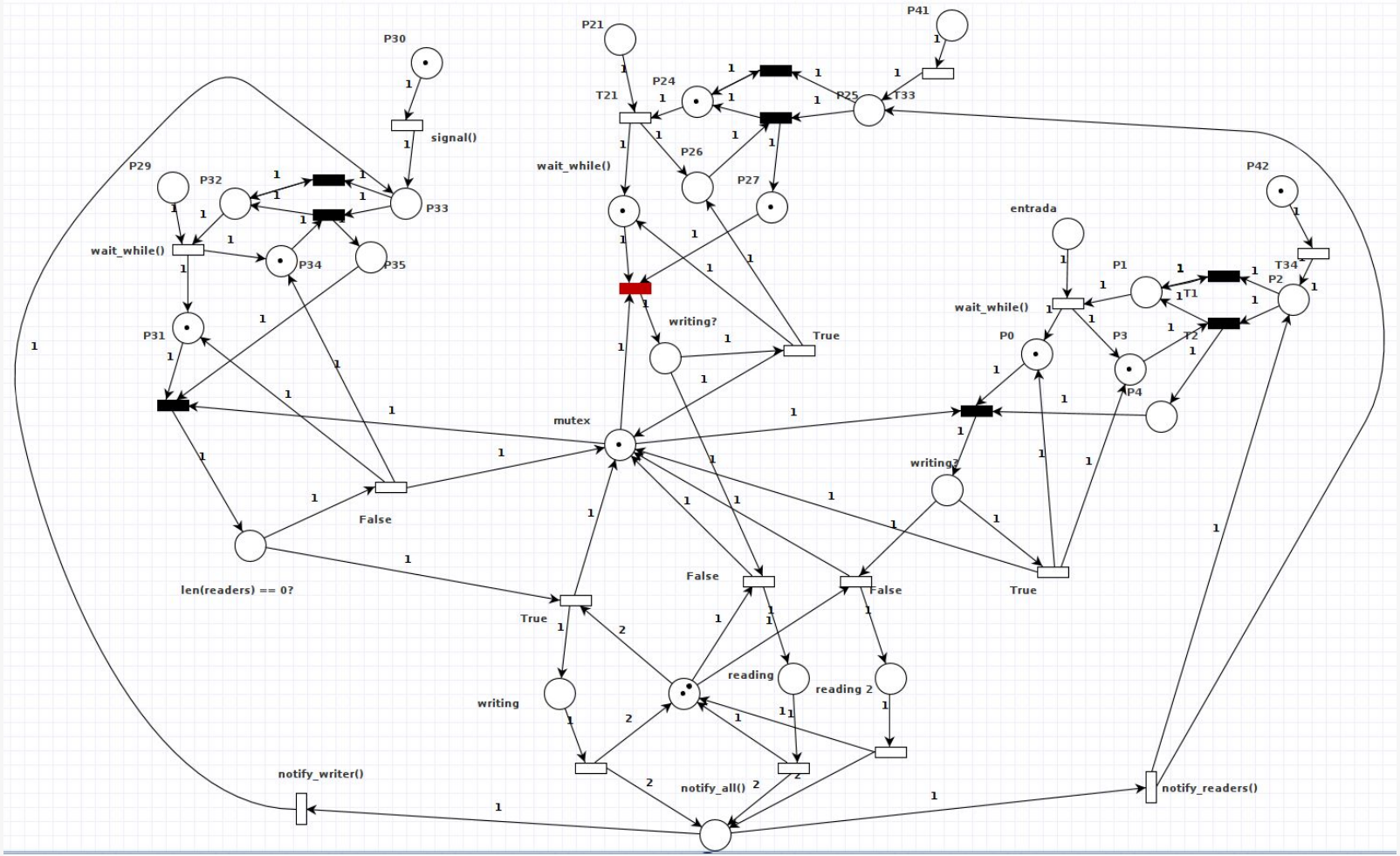

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_all()
notify_writer()
notify_readers()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing2

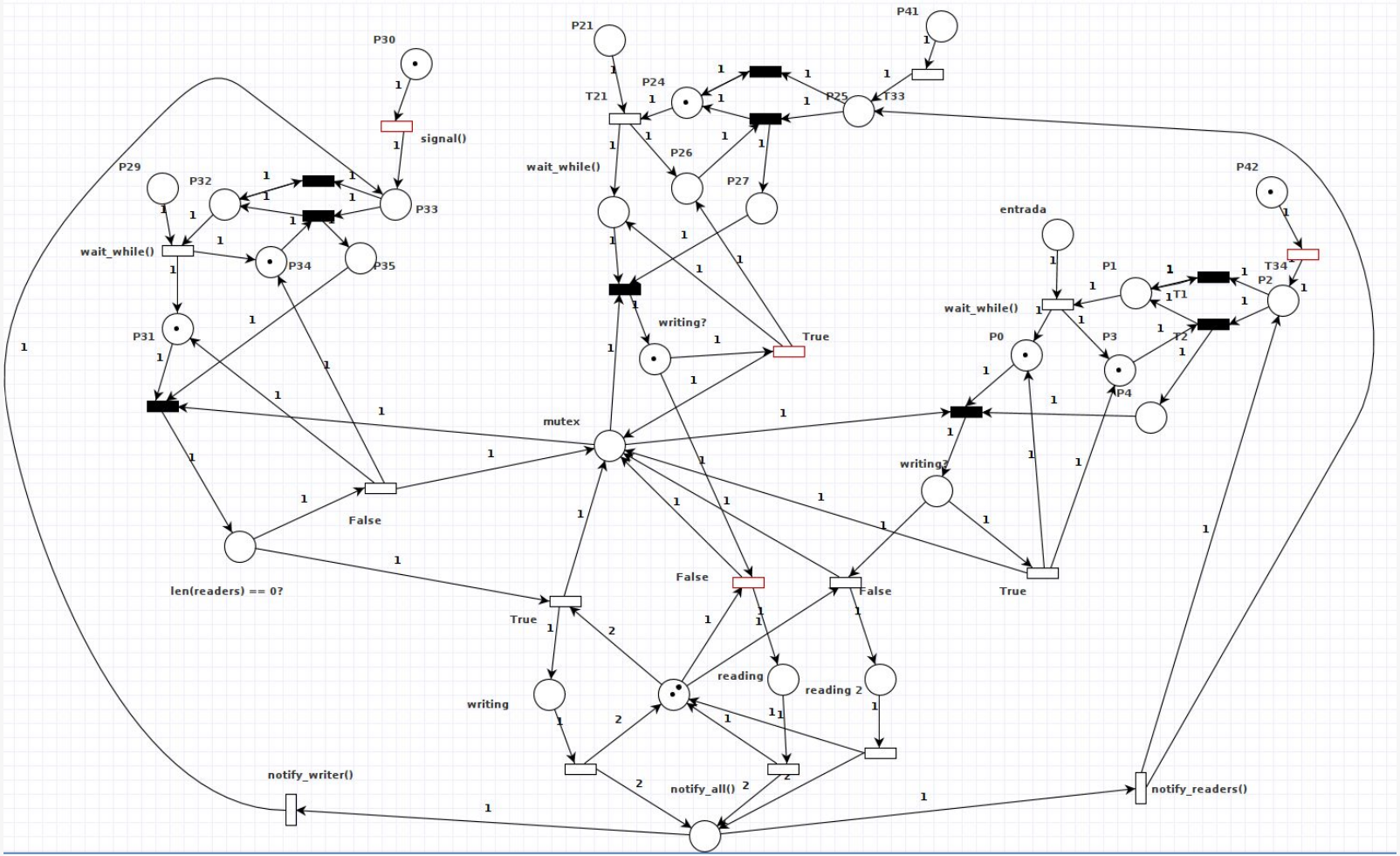
Reader 1 → Writer → Reader 2

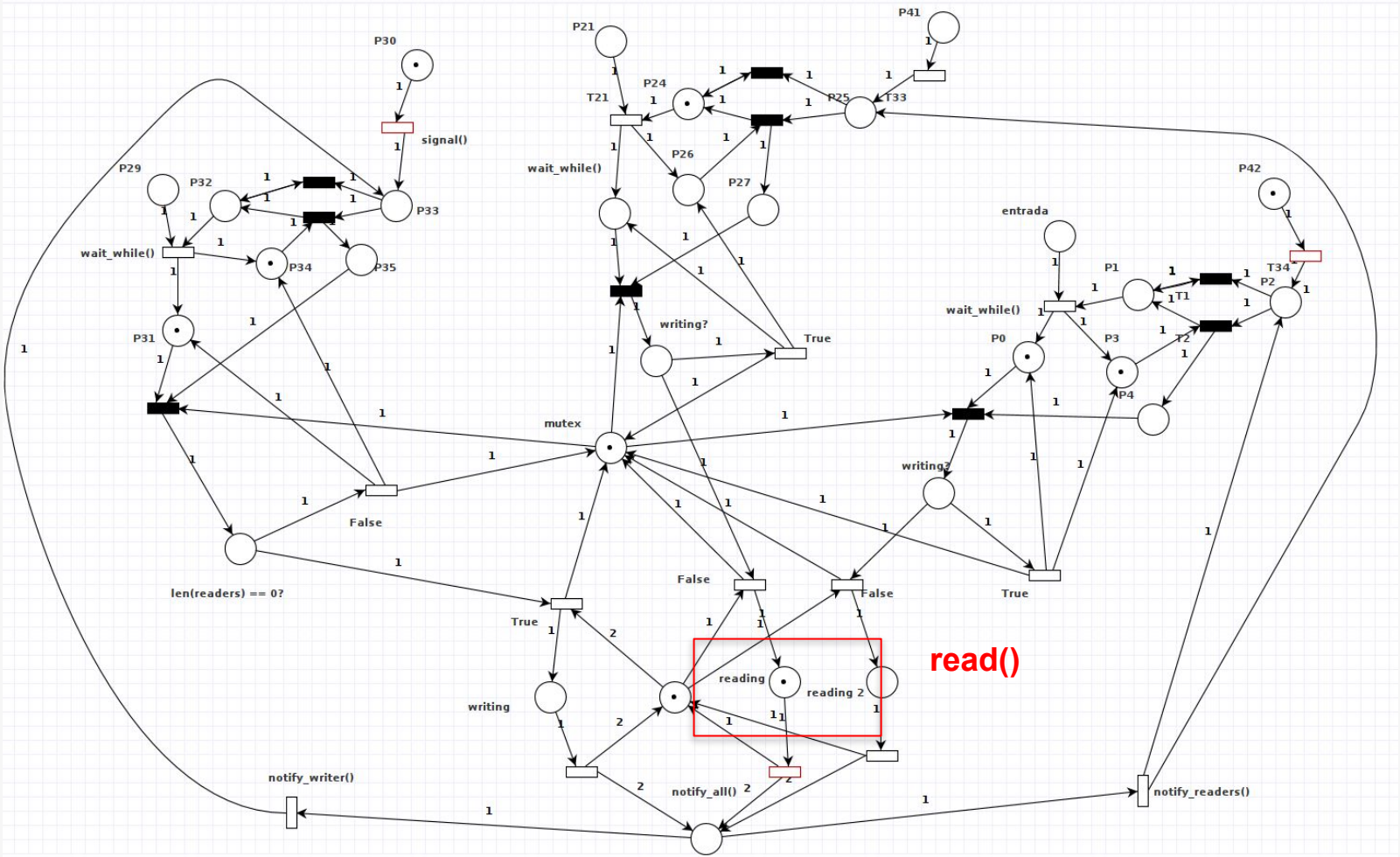

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
True
writing
reading
reading 2
notify_writer()
notify_all()
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
True
False
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing?
notify_readers()

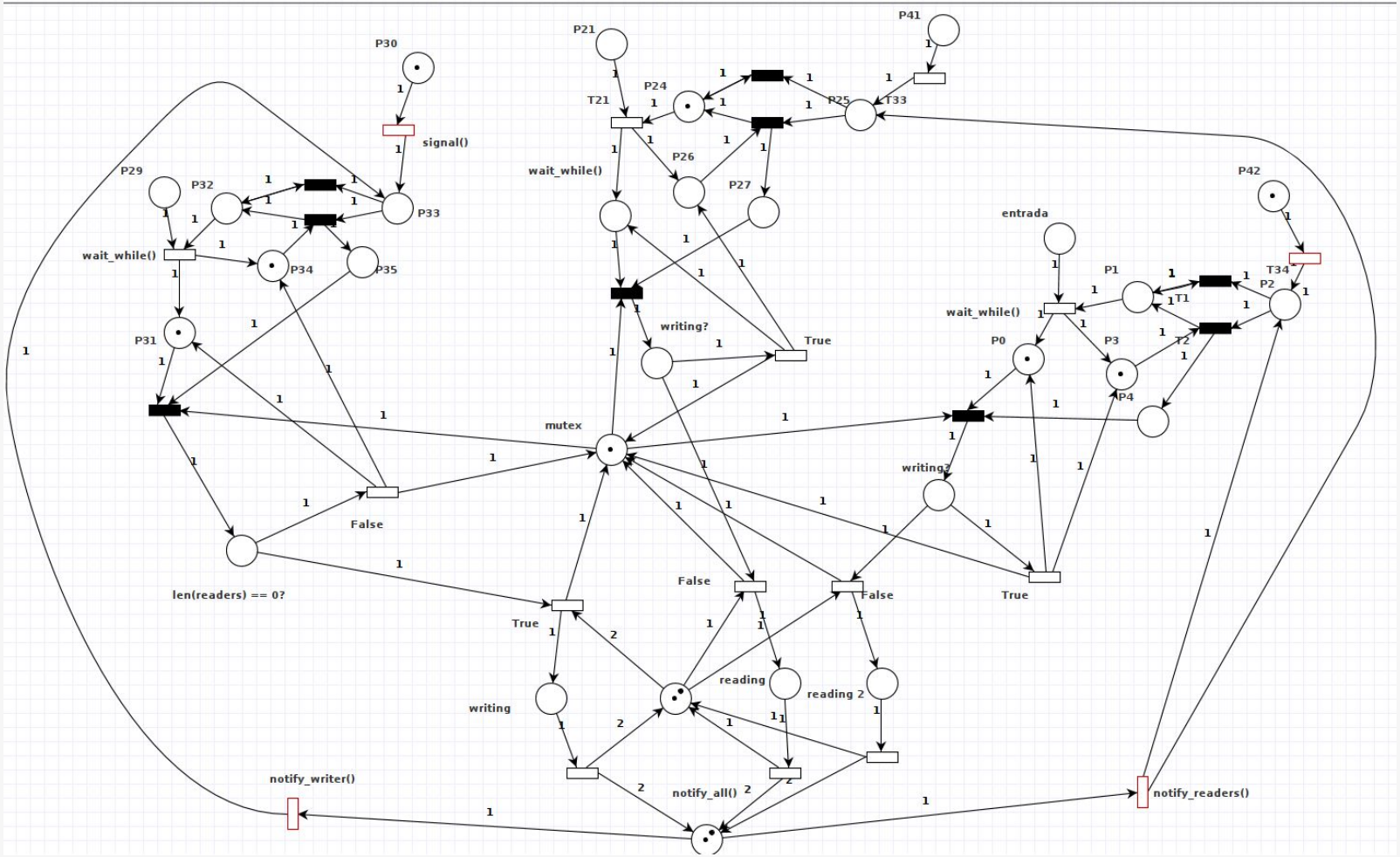

Todos haciendo wait()
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
T21
P24
P25
P41
T33
P26
wait_while()
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing2


Despertamos al reader
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing2

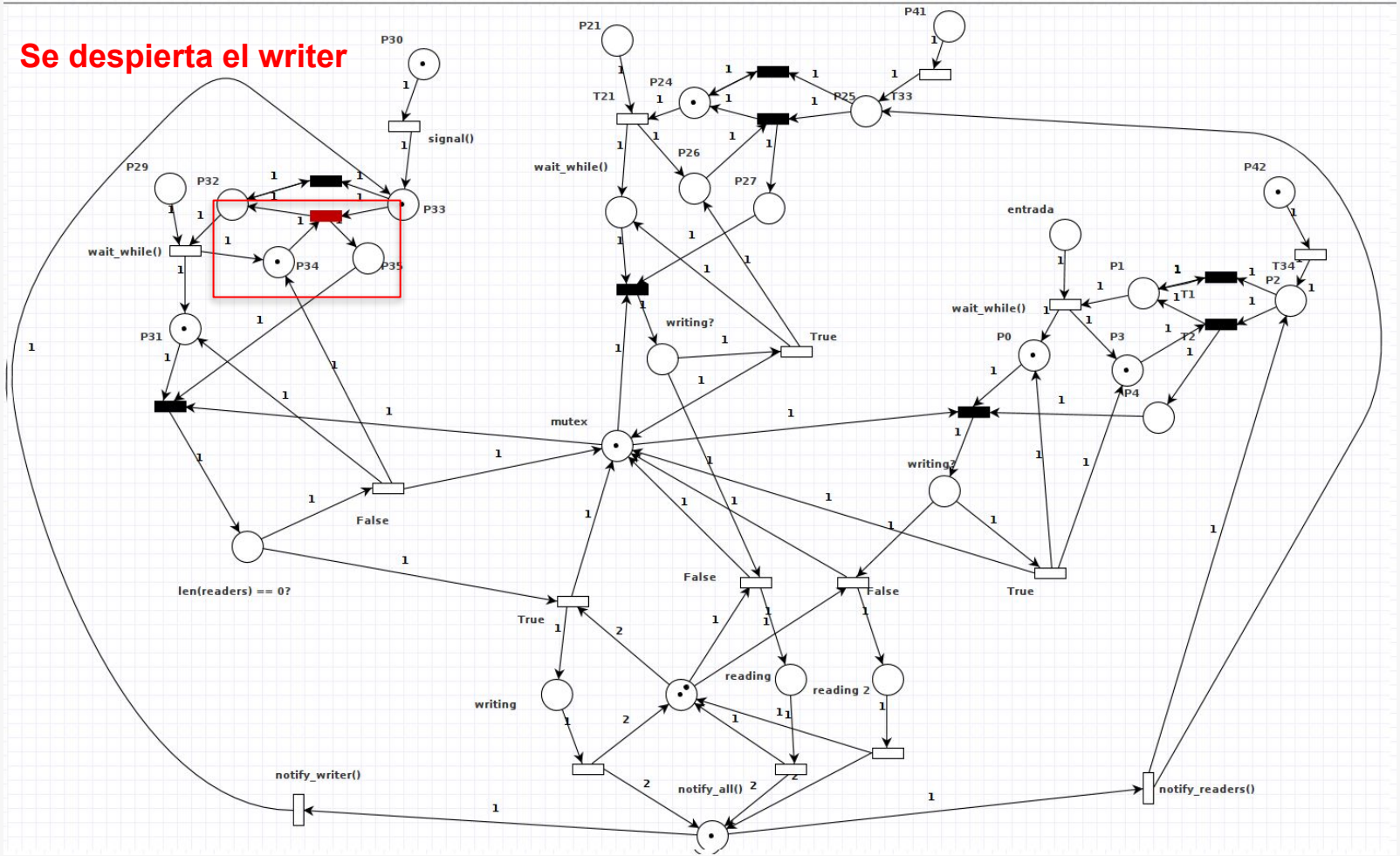

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
True
False
writing
reading
reading 2
notify_writer()
notify_all()
P42
entrada
P1
T1
P2
T34
wait_while()
P0
P3
T2
P4
writing2
False
True
notify_readers()

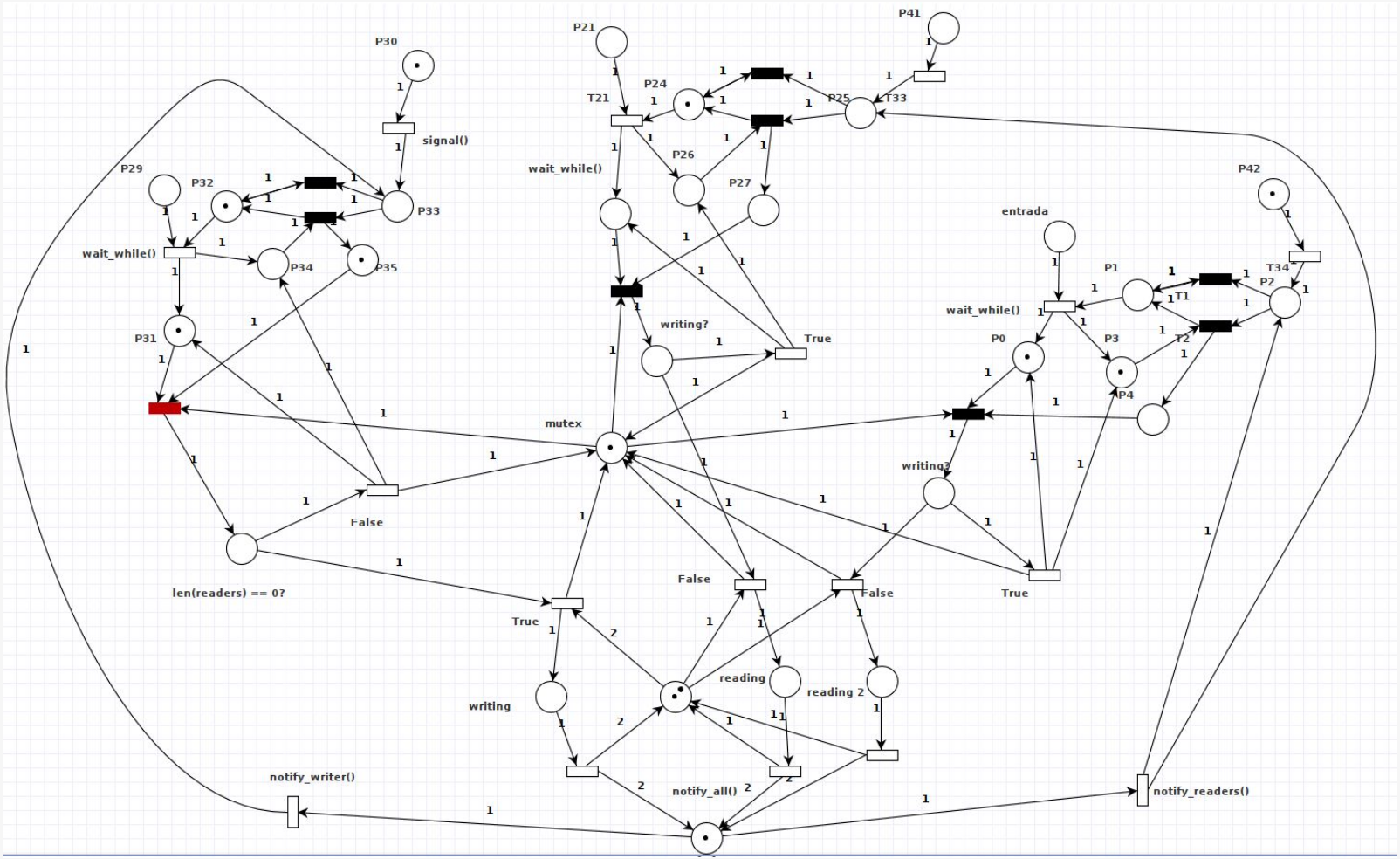

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?
notify_readers()

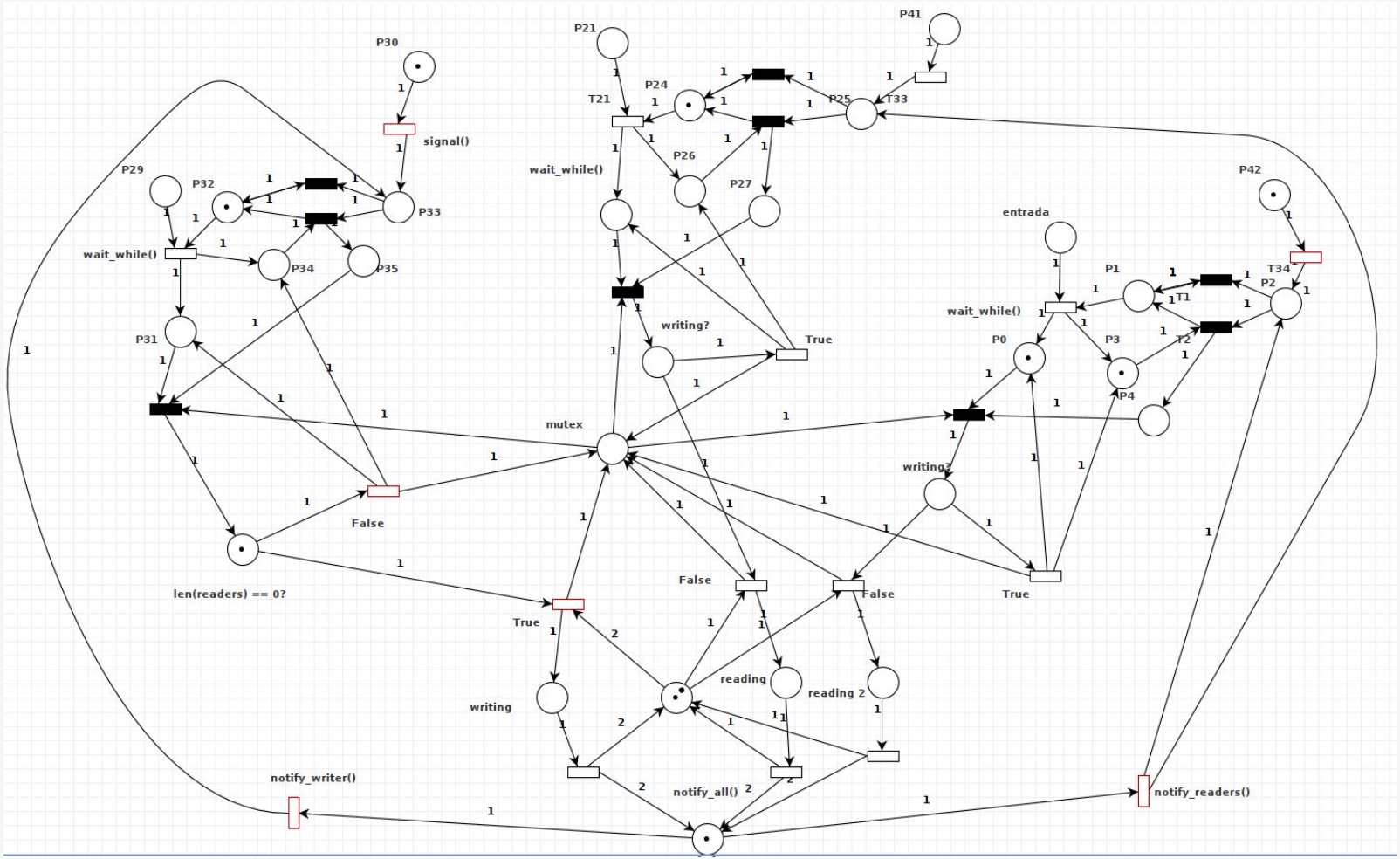

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing2
read()

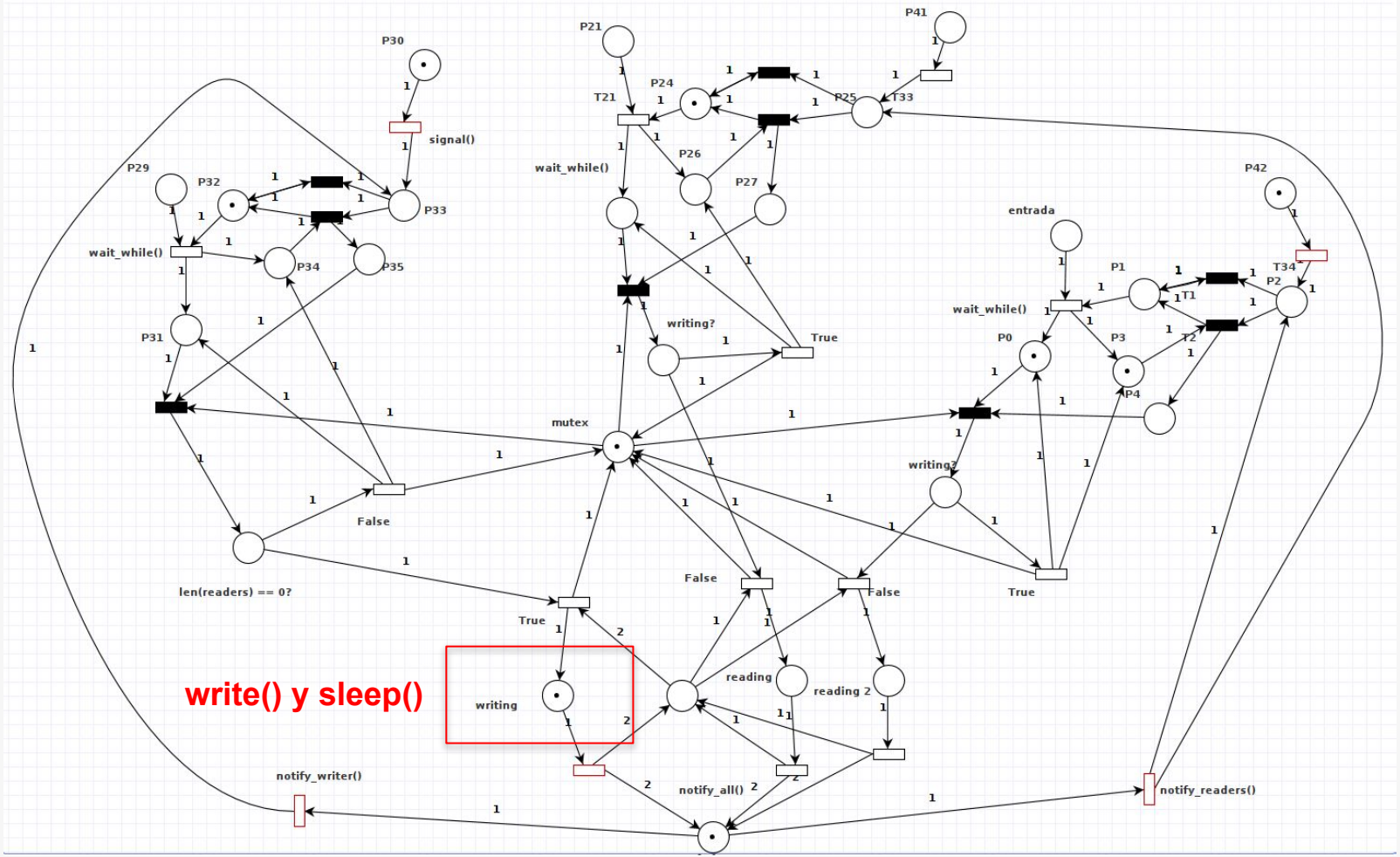

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
P24
T21
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing3

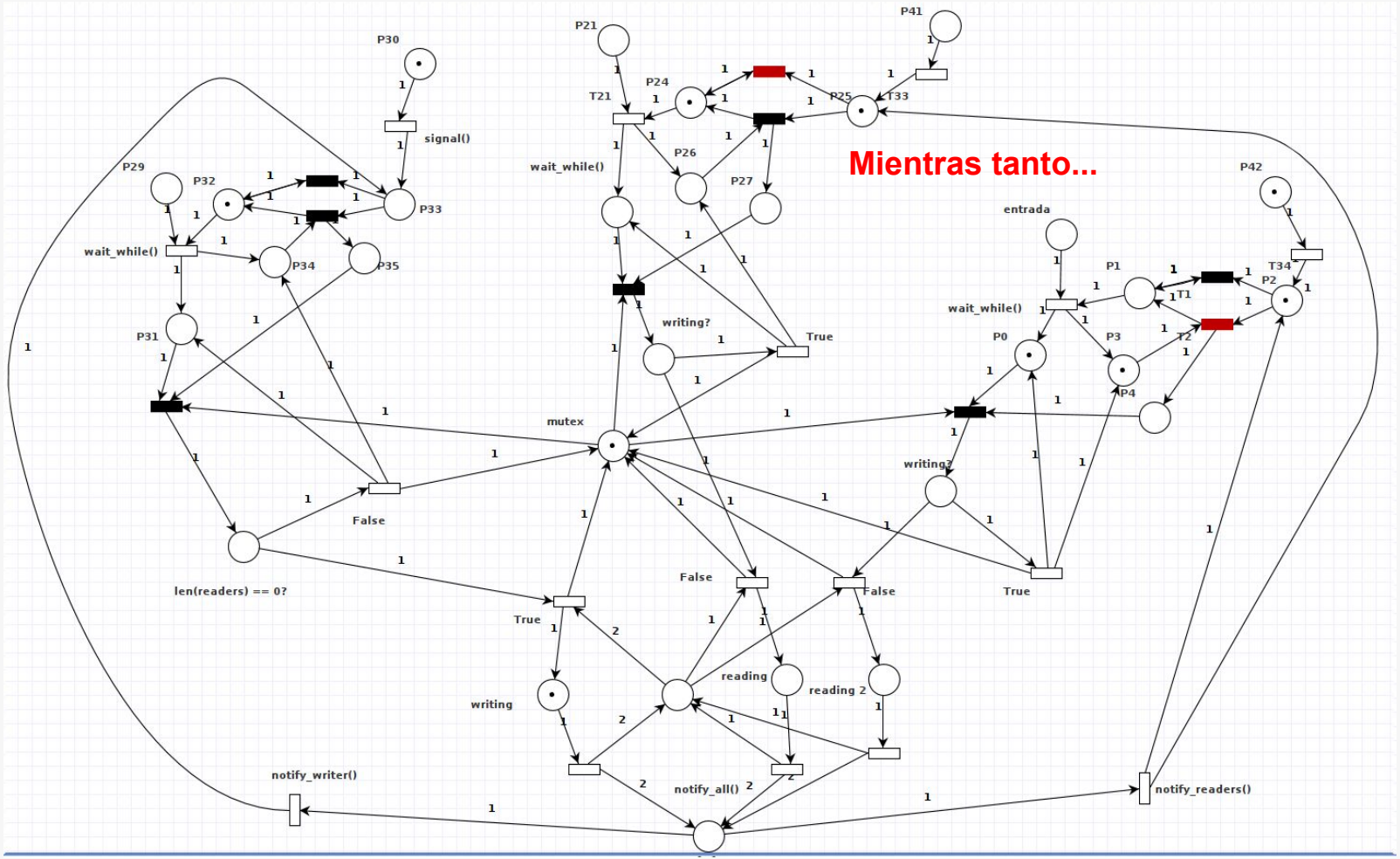
Se despierta el writer

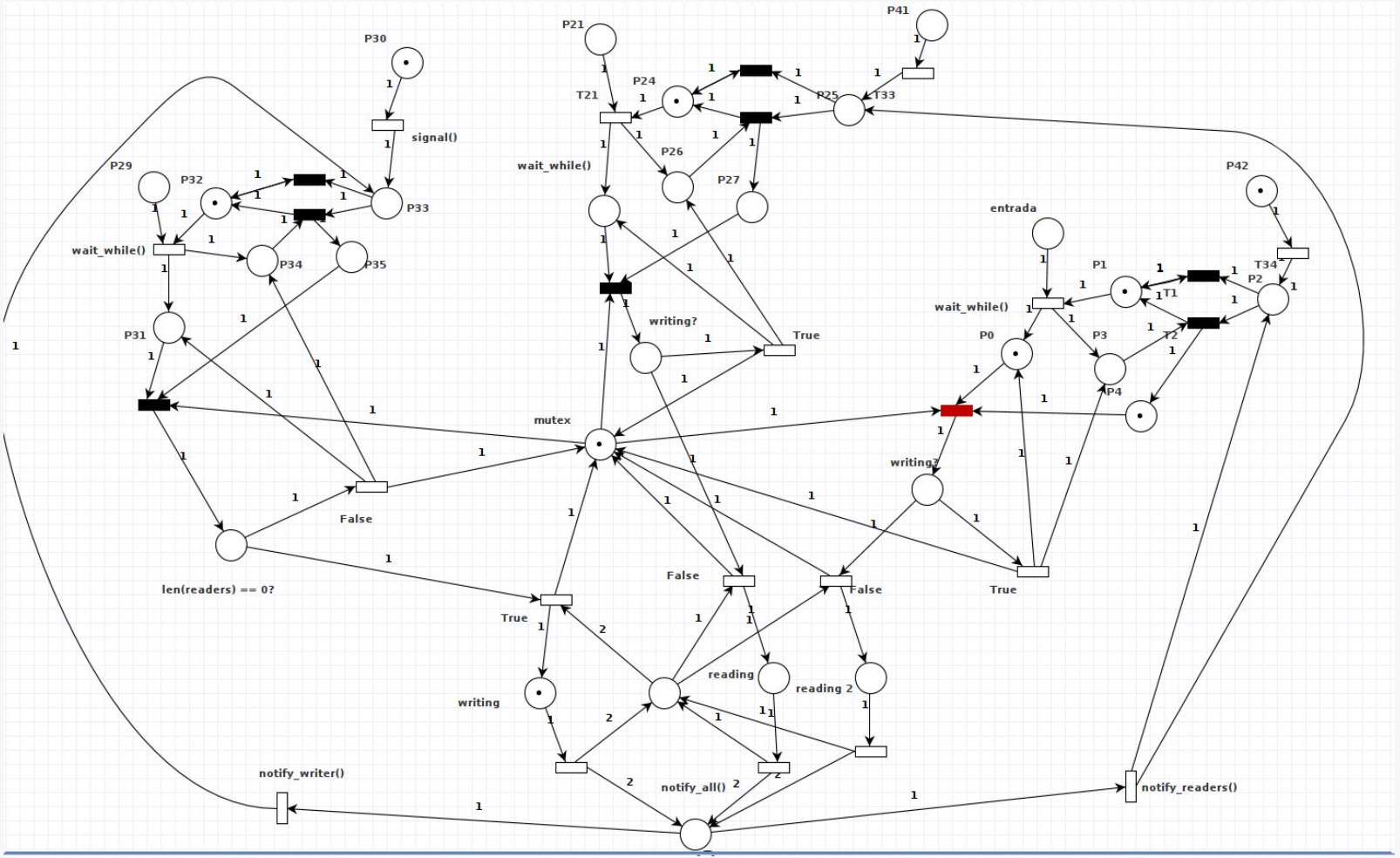


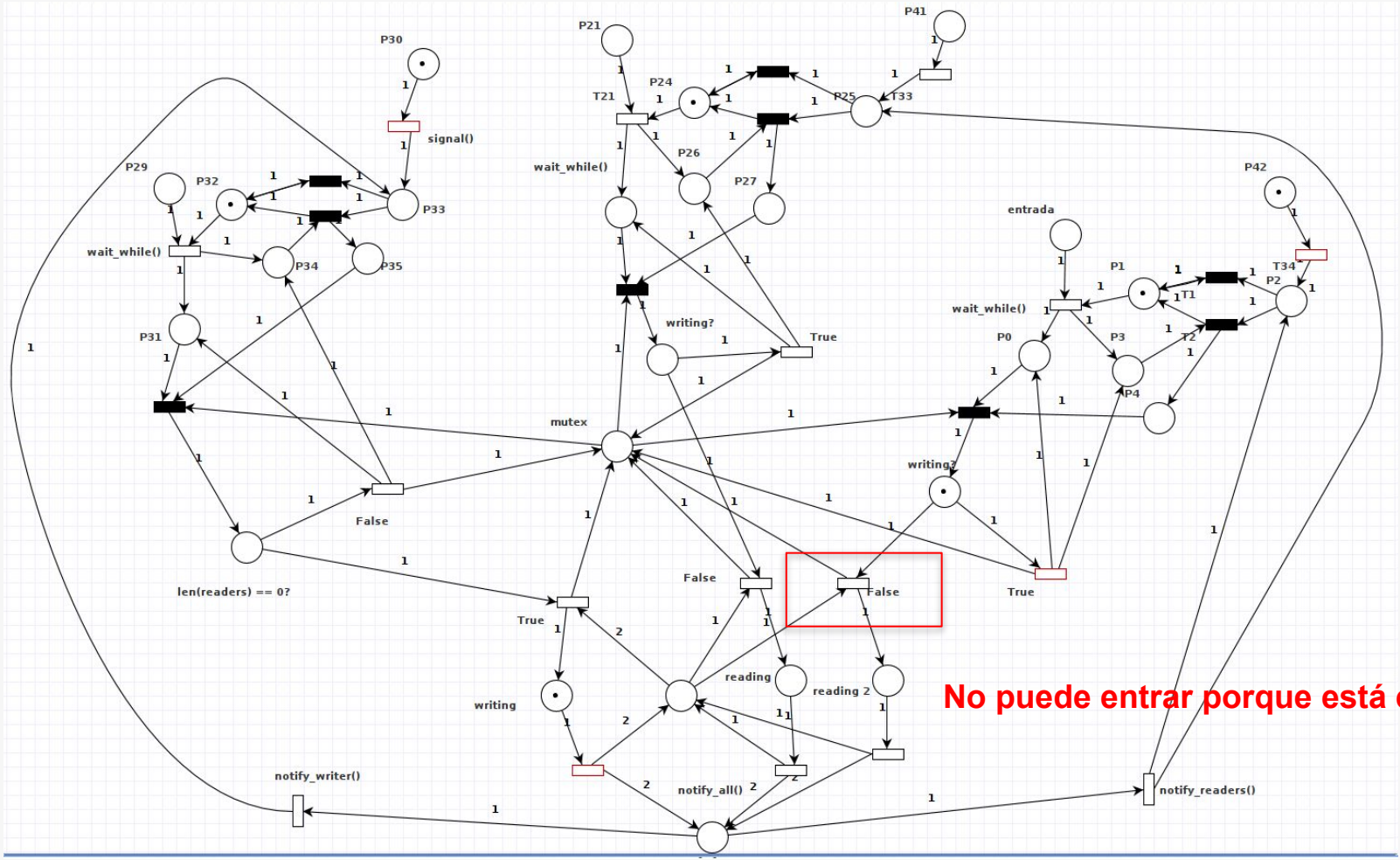

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
mutex
False
len(readers) == 0?
True
P21
P41
P24
T21
P25
T33
wait_while()
P26
P27
writing?
True
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?
False
False
True
reading
reading 2
writing
notify_writer()
notify_all()
notify_readers()

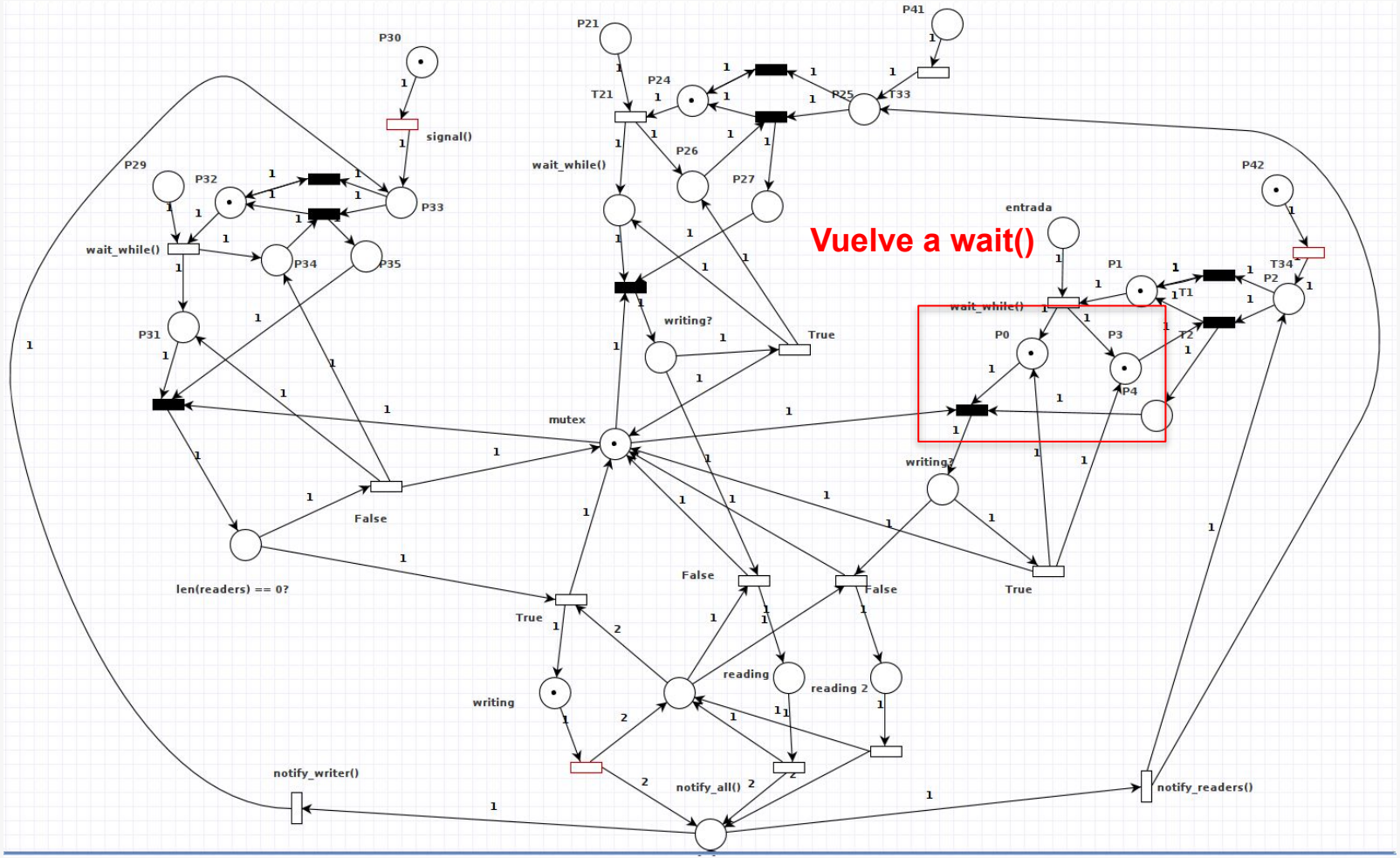

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
P21
T21
P24
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
True
False
writing
reading
reading 2
notify_all()
notify_writer()
notify_readers()
entrada
P1
T1
P2
T34
P42
wait_while()
P0
P3
T2
P4
writing2
False
True

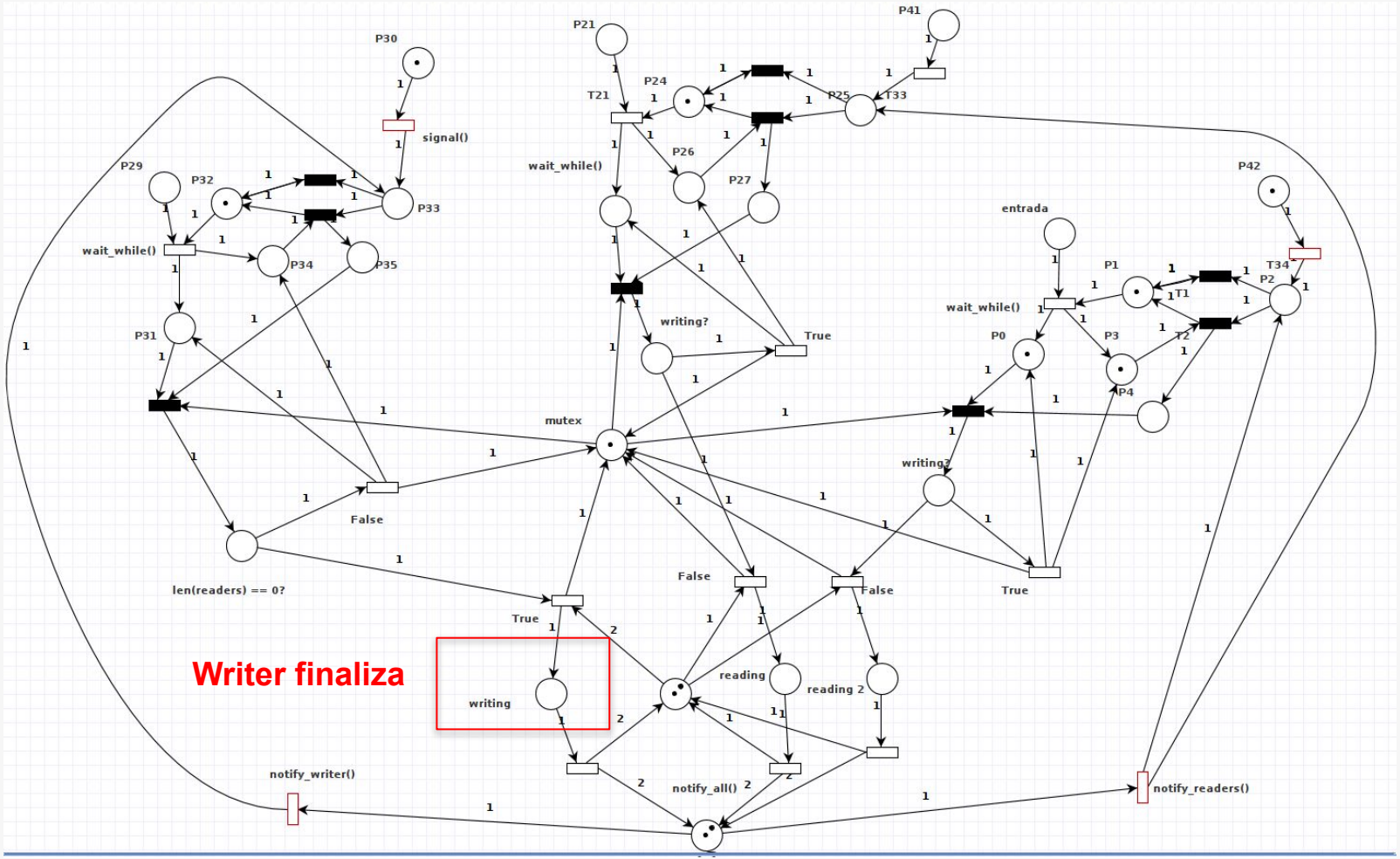

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
True
writing
P21
T21
P24
P25
P41
T33
wait_while()
P26
P27
writing?
True
mutex
False
False
reading
reading 2
notify_writer()
notify_all()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing2
False
True
notify_readers()
write() y sleep()

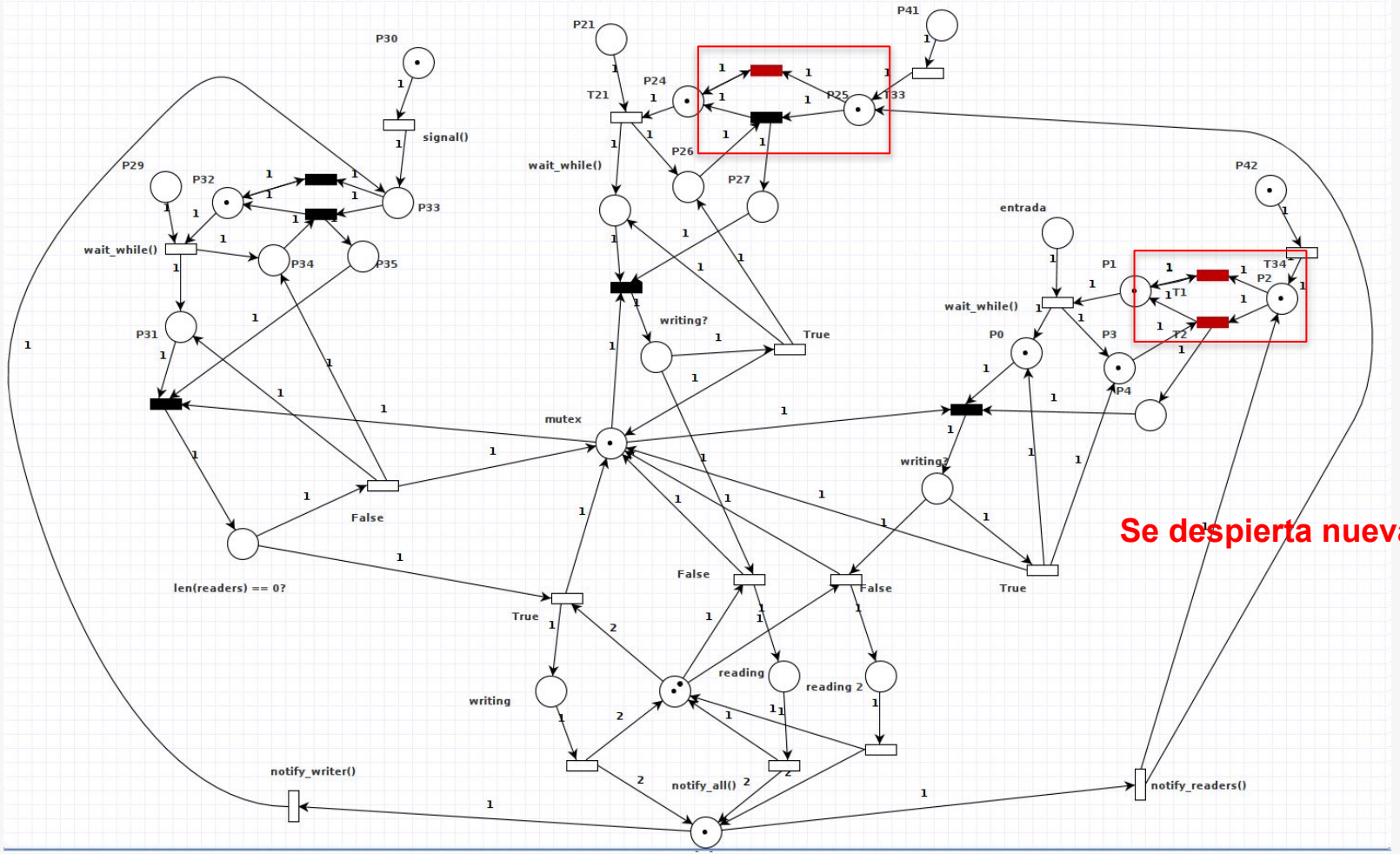

Mientras tanto...
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P24
T21
P25
T33
P41
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P1
T1
P2
T34
P42
wait_while()
P0
P3
T2
P4
writing2


P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
False
len(readers) == 0?
True
P21
P41
P24
T21
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing2

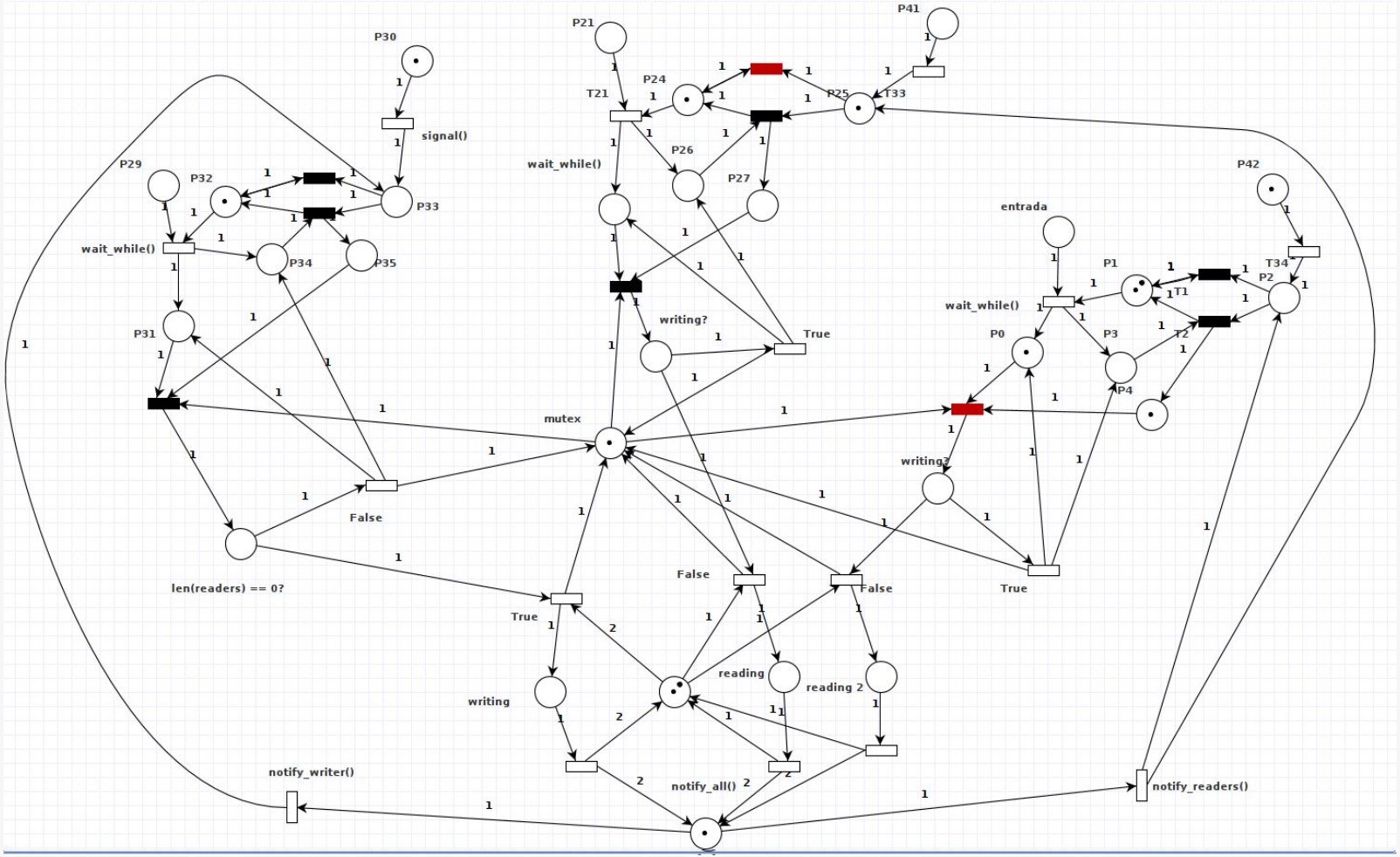

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
P42
entrada
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?
False
True
No puede entrar porque está el writer

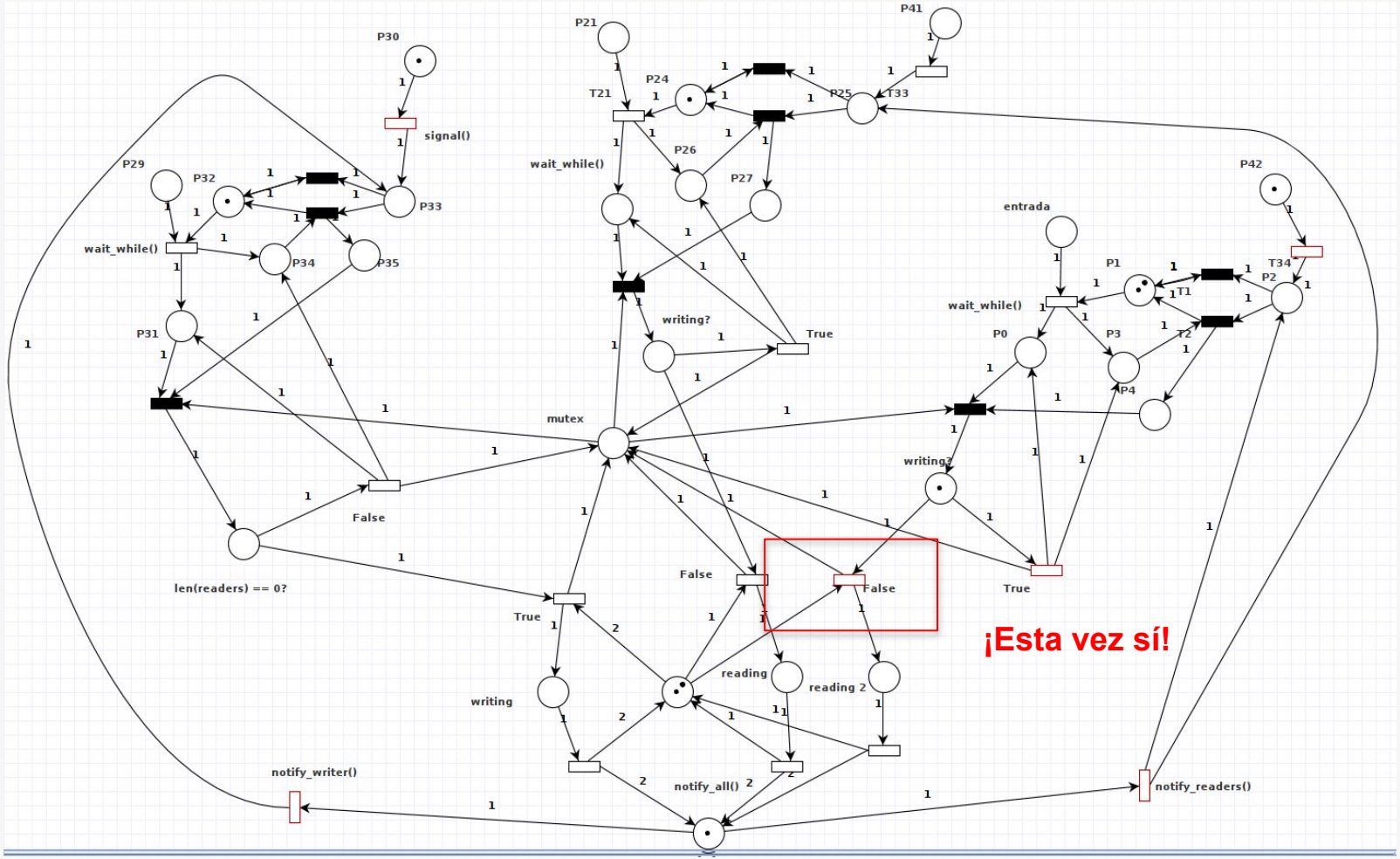

Vuelve a wait()
P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?

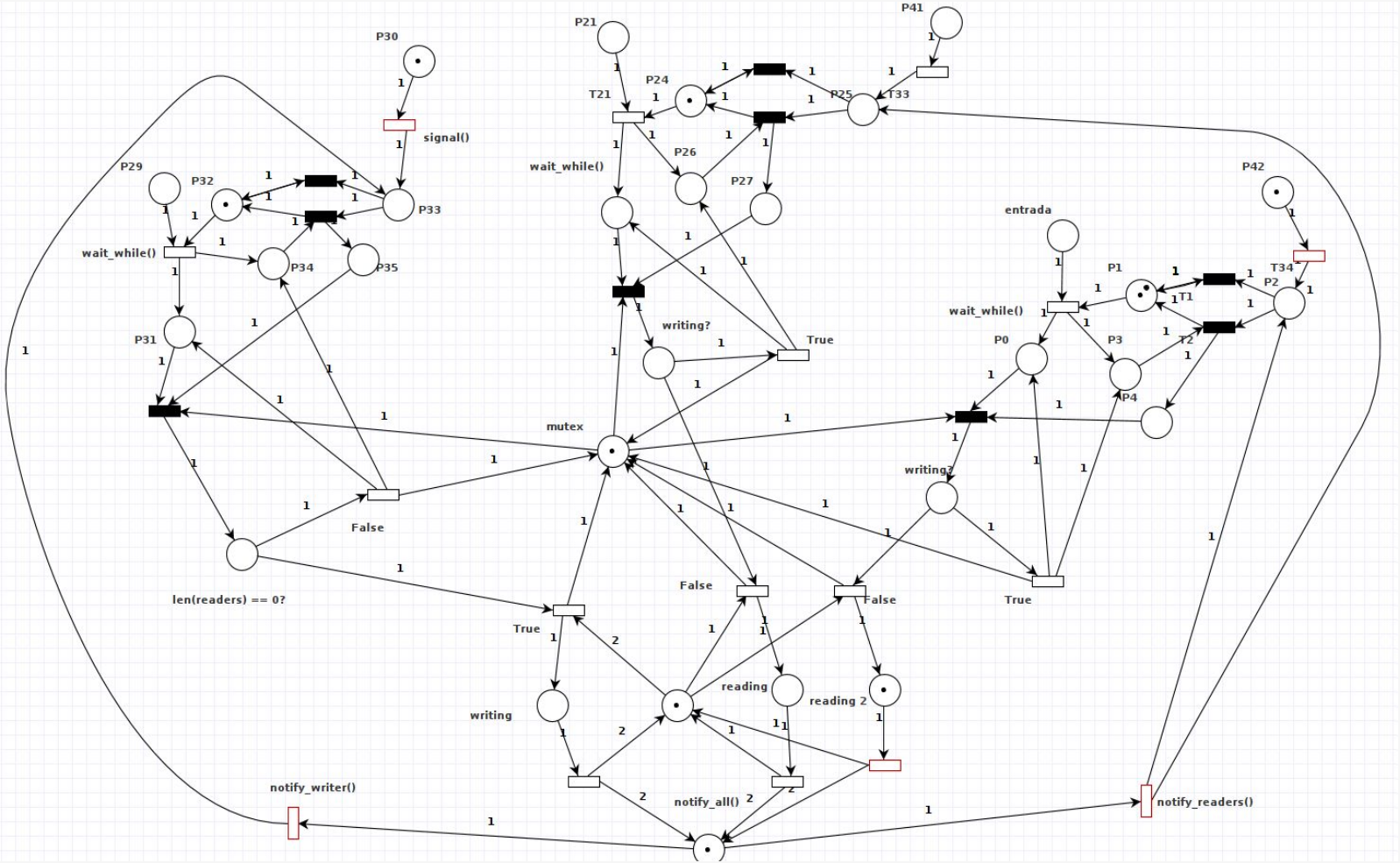

P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
T34
P1
P2
T1
wait_while()
P0
P3
T2
P4
writing2
Writer finaliza

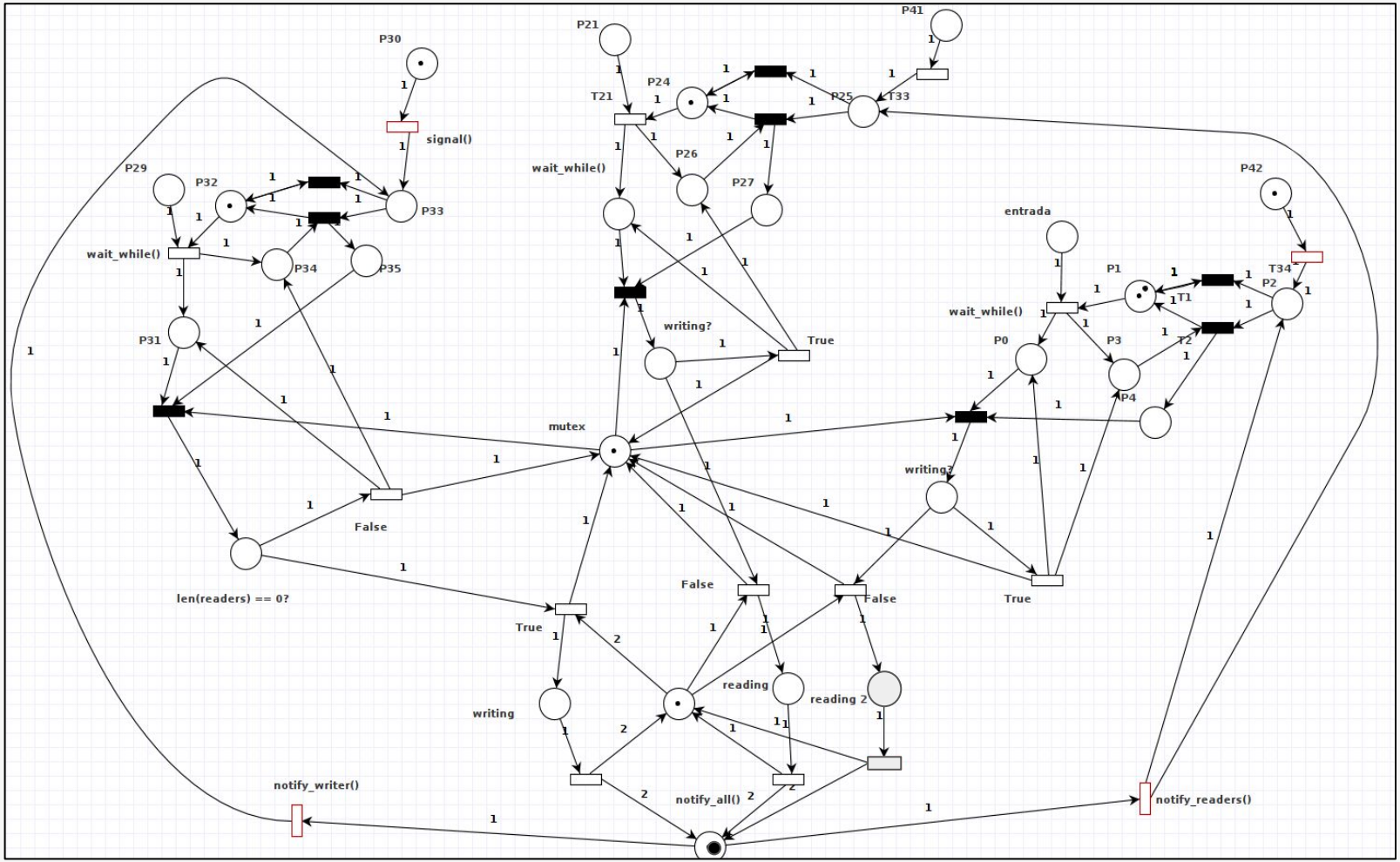


Se despierta nuevamente


P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
T21
P24
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
T34
P1
P2
T1
wait_while()
P0
P3
T2
P4
writing2


P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
P24
T21
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_all()
notify_writer()
notify_readers()
entrada
P42
P1
T34
P2
T1
wait_while()
P0
P3
T2
P4
writing?
¡Esta vez sí!


P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
P24
T21
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
T34
P1
P2
T1
wait_while()
P0
P3
T2
P4
writing?


P30
signal()
P29
P32
P33
wait_while()
P34
P35
P31
P21
P41
P24
T21
P25
T33
wait_while()
P26
P27
writing?
True
mutex
False
len(readers) == 0?
True
False
False
True
writing
reading
reading 2
notify_writer()
notify_all()
notify_readers()
entrada
P42
T34
P1
P2
T1
T2
wait_while()
P0
P3
P4
writing?

¡Gracias!