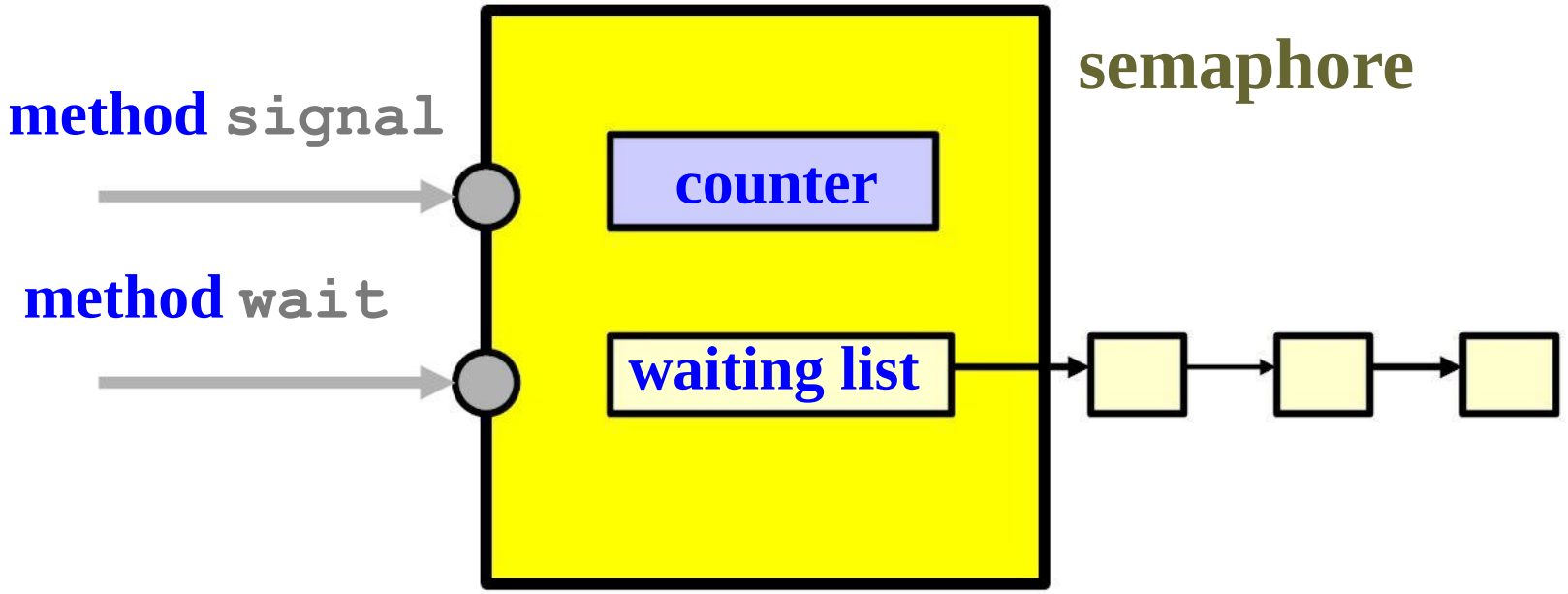
OPERATING SYSTEM

TERM PAPER

NAME :- RATAN KUMAR
ROLL NO. :- 33200118021
DEPT. :- CSE (5TH SEM)
TOPIC:- SEMAPHORES

Semaphores

- A *semaphore* is an object that consists of a counter, a waiting list of processes and two methods (*e.g.*, functions): `signal`and `wait`.



Semaphore Method: wait

```
void wait(sem S)
{
    S.count--;
    if (S.count < 0) {
        add the caller to the waiting list;
        block();
    }
}
```

- After decreasing the counter by 1, if the counter value becomes negative, then
  - add the caller to the waiting list, and then
  - block itself.

Semaphore Method: signal

```
void signal(sem S)
{
    S.count++;
    if (S.count <= 0) {
        remove a process P from the waiting list;
        resume(P);
    }
}
```

- After increasing the counter by 1, if the new counter value is not positive, then
  - remove a process P from the waiting list,
  - resume the execution of process P, and return

Important Note: 1/4

```
S.count--;                          S.count++;
if (S.count<0) {                    if (S.count<=0) {
    add to list;                        remove P;
    block();                            resume(P);
}                                   }
```

- If `S.count < 0`, `abs(S.count)` is the number of waiting processes.
- This is because processes are added to (*resp.*, removed from) the waiting list only if the counter value is `< 0` (*resp.*, `<= 0`).

Important Note: 2/4

```
S.count--;                          S.count++;
if (S.count<0) {                    if (S.count<=0) {
    add to list;                        remove P;
    block();                            resume(P);
}                                   }
```

- The waiting list can be implemented with a queue if FIFO order is desired.
- However, the correctness of a program should not depend on a particular implementation of the waiting list.
- Your program should not make any assumption about the ordering of the waiting list.

Important Note: 3/4

```
S.count--;                          S.count++;
if (S.count<0) {                    if (S.count<=0) {
    add to list;                        remove P;
    block();                            resume(P);
}                                   }
```

- The caller may be blocked in the call to `wait()`.
- The caller never blocks in the call to `signal()`. If `S.count > 0`, `signal()` returns and the caller continues. Otherwise, a waiting process is released and the caller continues. In this case, *two* processes continue.

The Most Important Note: 4/4

```
S.count--;                          S.count++;
if (S.count<0) {                    if (S.count<=0) {
    add to list;                        remove P;
    block();                            resume(P);
}                                   }
```

- `wait()` and `signal()` must be executed *atomically* (*i.e.*, as one uninterruptible unit).
- Otherwise, *race conditions* may occur.
- **Homework:** use execution sequences to show race conditions if `wait()` and/or `signal()` is not executed atomically.

Three Typical Uses of Semaphores

- **There are three typical uses of semaphores:**
  - **mutual exclusion:**

    Mutex (*i.e.*, *Mut*ual *Ex*clusion) locks
  - **count-down lock:**

    Keep in mind that semaphores have a counter.
  - **notification:**

    Indicate an event has occurred.

Use 1: Mutual Exclusion (Lock)

initialization is important

```
semaphore S = 1;
int       count = 0;
```

Process 1

```
while (1) {
    // do something
    S.wait();
        count++;
    S.signal();
    // do something
}
```

Process 2

```
while (1) {
    // do something
    S.wait();
        count--;
    S.signal();
    // do something
}
```

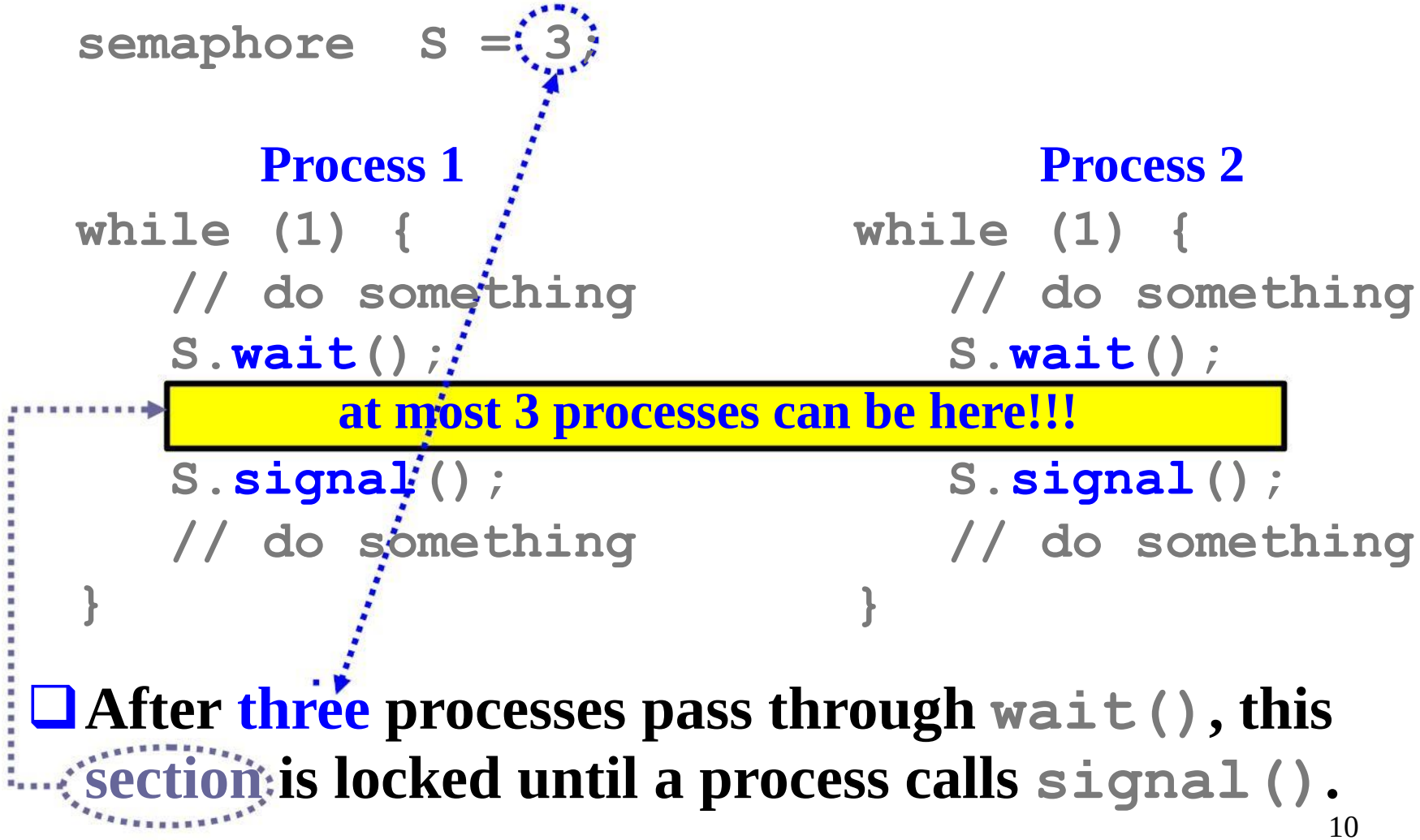
entry

critical sections

exit

- What if the initial value of `S` is zero?
- `S` is a *binary semaphore* (similar to a *lock*).

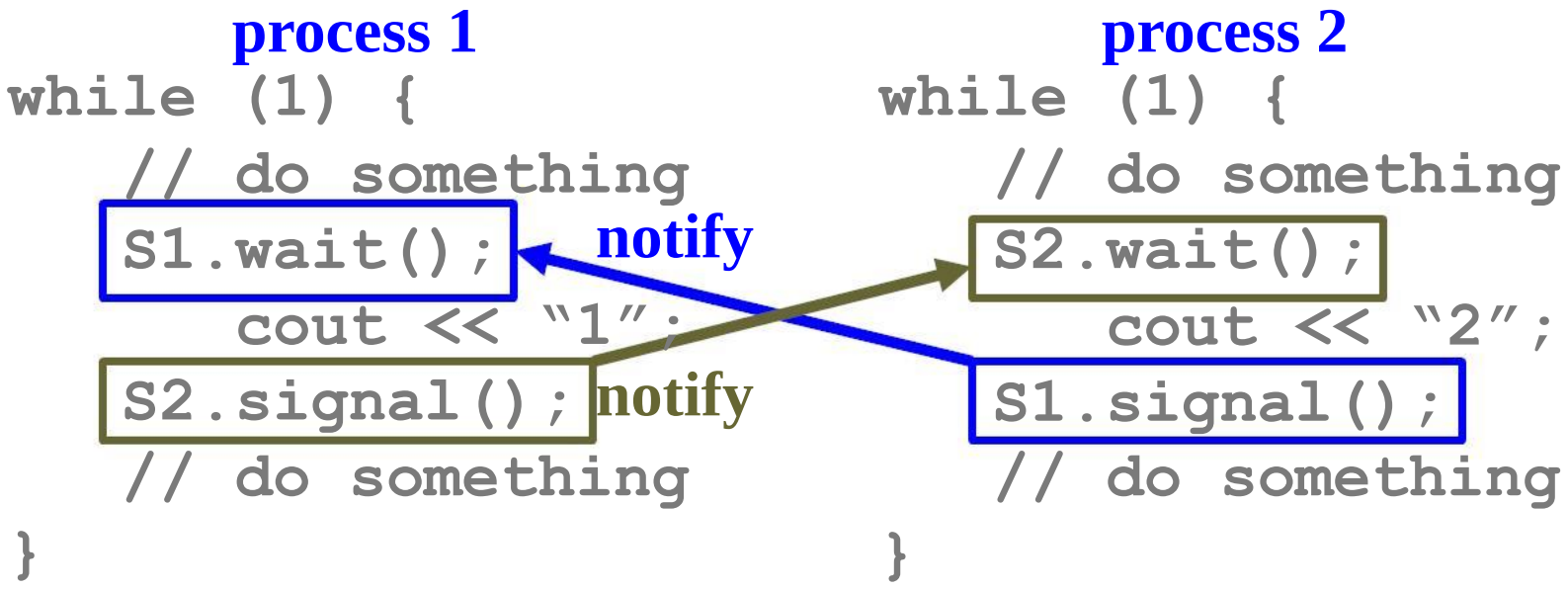
Use 2: Count-Down Counter



- After three processes pass through `wait()`, this section is locked until a process calls `signal()`.

Use 3: Notification

```
semaphore S1 = 1, S2 = 0;
```



- Process 1 uses `S2.signal()` to notify process 2, indicating "I am done. Please go ahead."
- The output is `1 2 1 2 1 2 ……`
- What if both `S1` and `S2` are both 0's or both 1's?
- What if `S1` = 0 and `S2` = 1?