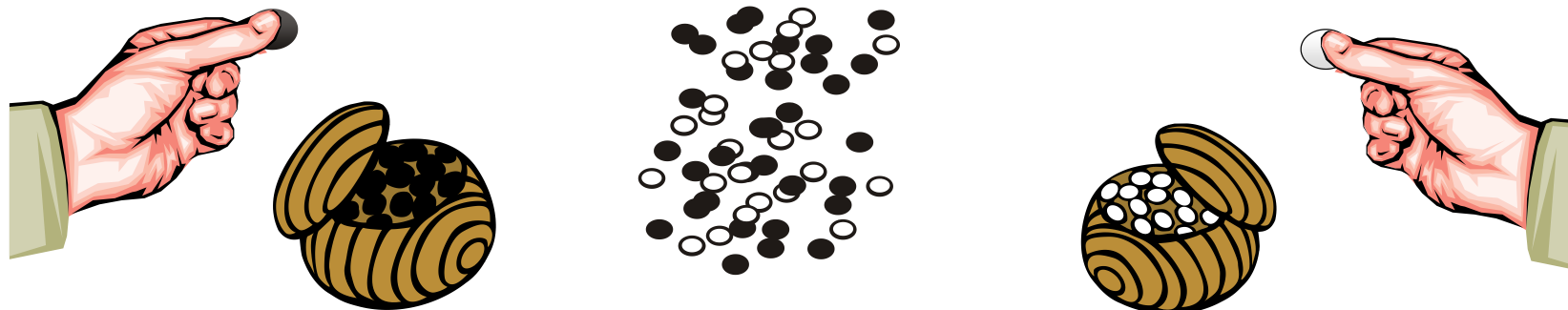




Some Examples

Example 1

- The worker who produces Go accidentally mixes an equal number of black and white pieces in a box, and now separates the black and white with an automatic sorting system consisting of two concurrent processes, which function as follows:
 - (1) Process A: picks up one black piece, Process B: picks up one white piece;
 - (2) Each process picks one and only one piece at a time. At any time, at most one process is permitted to pick up one piece.



Example 1

- Analysis

```
Process_A() {  
    While (True)  
    {  
        pick a ●;  
    }  
}
```

```
Process_B() {  
    While (True)  
    {  
        pick a ○;  
    }  
}
```

Example 1

- **Step 1:** Determine the relationship between processes.
By (2), it is known that there is a mutually exclusive relationship between processes.
- **Step 2:** Determine the semaphore and its value.
Because process A and process B enter the box mutually exclusive to pick pieces, and the box is the public resource, so set a semaphore **s**, the value depends on the number of public resources, because there is only one box, the initial value of **s** is set to 1.

Example 1

- Implementation: semaphore **s**=1;

```
Process_A() {  
    While (True)  
    {  
        Wait(s);  
        pick a ●;  
        Signal(s);  
    }  
}
```

```
Process_B() {  
    While (True)  
    {  
        Wait(s);  
        pick a ●;  
        Signal(s);  
    }  
}
```

Example

- The key to determining whether processes are mutually exclusive is to see whether a **public resource** is shared between processes, and a public resource corresponds to a **semaphore**. Determining the value of the semaphore is a key point, and the value represents the number of available resource entities.

Example

- Use **PV primitives** to implement process synchronization
 - Unlike process mutual exclusion, the semaphore in process synchronization is only related to the process and the restricted process, not to the whole group of concurrent processes. Therefore, the semaphore is called **private semaphore**.
 - Steps to realize process synchronization by using PV primitive :
 - 1) determine the relationship between processes is synchronous, set **private semaphore** for each concurrent process;
 - 2) assign initial value to private semaphore;
 - 3) use PV primitive and private semaphore to specify the execution sequence of each process.
- Let's add a condition to example 1 that the processes are synchronized.

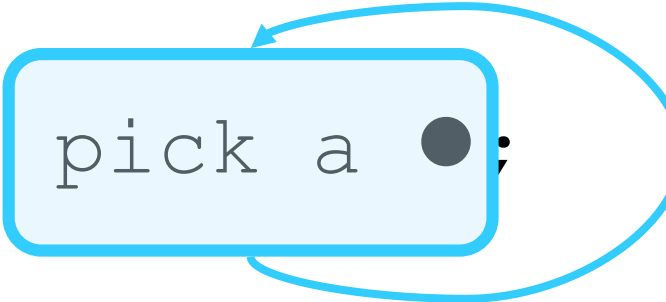
Example 1+

- Additional constraints :
 - (3) When one process picks a piece (black or white), it must let another process pick a piece (white or black). So, the two process must be executed by turns.


Example 1+

- Analysis

```
Process_A() {  
  While (True)  
  {  
    pick a ●;  
  }  
}
```

A light blue rounded rectangle highlights the loop body of Process_A, containing the text "pick a ●;". A blue arrow starts from the right side of this box and points back to the "While (True)" line, indicating a loop iteration.

```
Process_B() {  
  While (True)  
  {  
    pick a ○;  
  }  
}
```

A dark gray rounded rectangle highlights the loop body of Process_B, containing the text "pick a ○;". A blue arrow starts from the right side of this box and points back to the "While (True)" line, indicating a loop iteration.

Example 1+

- Analysis

- Step 1: determine the relationship between processes. According to functions (1), (2) and (3), the relationship between processes is **synchronous**.
- Step 2: determine the semaphore and its value. Processes A and B share the public resource of the box, but the two processes must take turns to pick different colored pieces, so they should exchange messages each other.
 - For process A, it sets a private semaphore B to determine whether process A can pick a black piece, with an initial value of 1.
 - For process B, a private semaphore W is used to determine whether process B can pick a white piece, and the initial value is 0.
 - Of course, you can also set the initial value of B to be 0 and W to be 1.

Example 1+

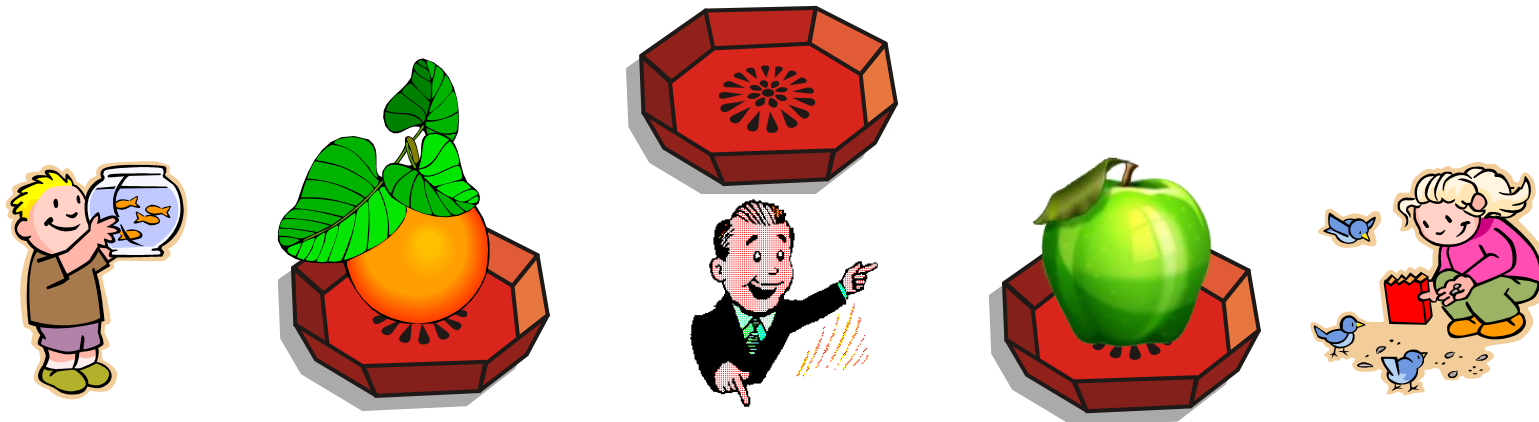
- Implementation: semaphore $B=1$, $W=0$; (or $B=0$, $W=1$)

```
Process_A() {  
    While (True)  
    {  
        Wait(B) ;  
        pick a ● ;  
        Signal(W) ;  
    }  
}
```

```
Process_B() {  
    While (True)  
    {  
        Wait(W) ;  
        pick a ● ;  
        Signal(B) ;  
    }  
}
```

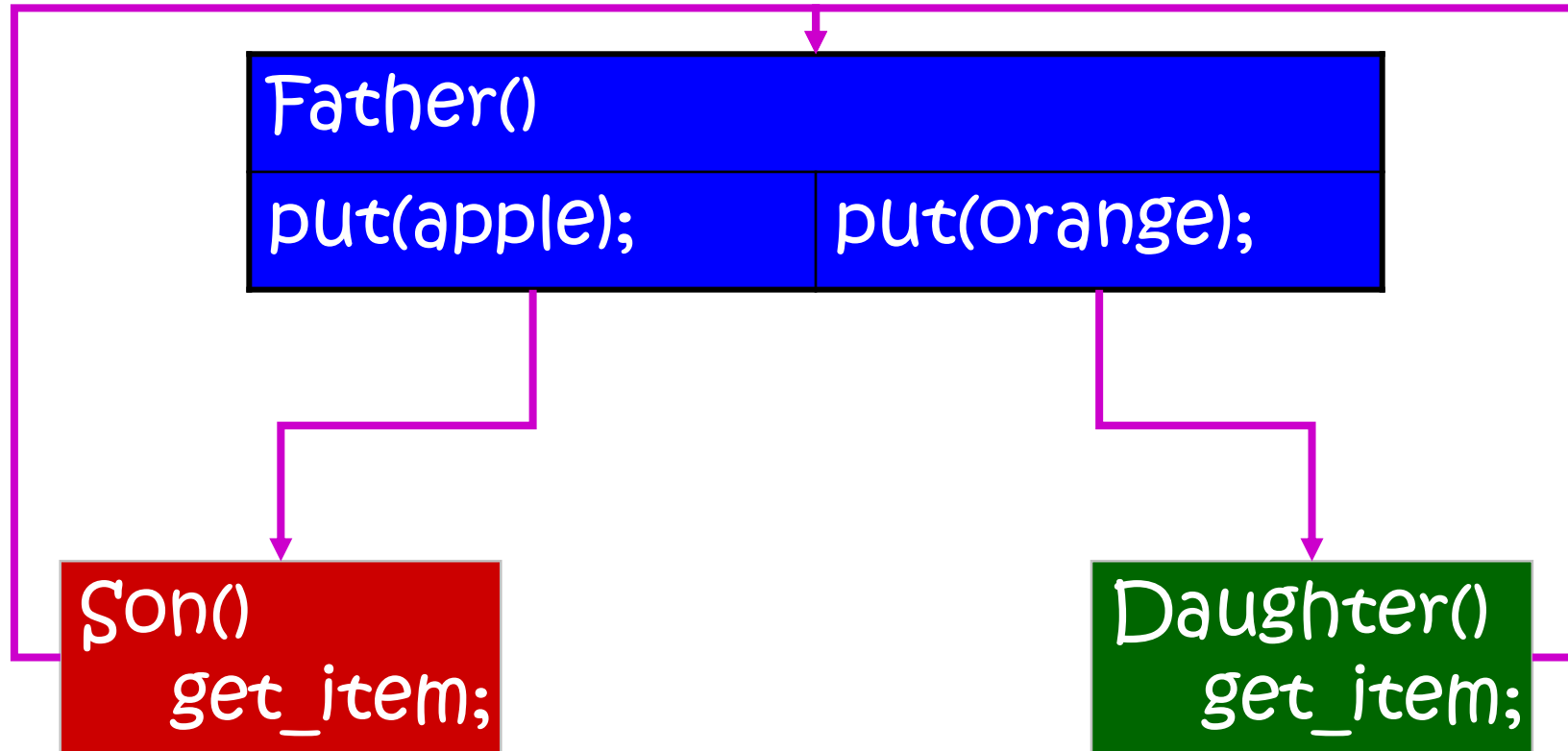
Example 2

- There is an empty plate on the table, which allows **only one** fruit. The father can put fruit in the plate, and only oranges and apples are put. The son only eats the oranges in the plate, while the daughter waits for the apples in the plate. Only one fruit can be placed at a time when the plate is empty.
- Please use P,V primitives to implement the three concurrent processes: **father**, **son** and **daughter**.



Example 2

- Analysis



Example 2

- Semaphore empty=1;
- Semaphore orange=0,
- Semaphore apple=0;

```
Father() {  
    while (TRUE) {  
        wait(empty);  
        put_item;  
        if (item==apple)  
            signal(apple);  
        else  
            signal(orange);  
    }  
}
```

```
Son() {  
    while (TRUE) {  
        wait(orange);  
        get_item;  
        signal(empty);  
    }  
}
```

```
Daughter() {  
    while (TRUE) {  
        wait(apple);  
        get_item;  
        signal(empty);  
    }  
}
```

Example 3

- There are three concurrent processes R, M, and P that share the same buffer, assuming that the buffer can hold only one record.
 - Process R is responsible for reading information from the input device and putting it into the buffer after each record is read in.
 - Process M processes the read records in the buffer.
 - Process P prints out the processed record.
 - After the input record is processed and output, the buffer can store the next record.
- Write concurrent programs that they can execute correctly.

Example 3

- PV primitive: three processes share a buffer, they must work **synchronously**, three semaphores can be defined:
 - S1: indicates whether the read record can be put into the buffer, with an initial value of 1.
 - S2: indicates whether records in the buffer can be processed, with an initial value of 0.
 - S3: indicates whether the record is processed and can be output, with an initial value of 0.

Example 3

- Implementation: semaphore $S1=1$, $S2=0$, $S3=0$;

```
process R {  
    while (TRUE) {  
        // read record;  
        P(S1);  
        // put record in to buffer;  
        V(S2);  
    }  
}
```

```
process M {  
    while (TRUE) {  
        P(S2);  
        // process record;  
        V(S3);  
    }  
}
```

```
process P {  
    while (TRUE) {  
        P(S3);  
        // print processed record;  
        V(S1);  
    }  
}
```

Example 4

- On a bus, the activities of the driver and the conductor are:
 - Driver: start the vehicle, drive normally, stop at the station.
 - Conductor: get passengers on the bus, close the door, sell ticket, open the door, let passengers get off.
- Use PV operations to control them.


Example 4

- Analysis


- Step 1: determine the relationship between processes.
 - The conductor can only work when the driver stops at the station.
 - Similarly, the conductor closes the door before the driver can work.
 - So, there is a **synchronous** relationship between the driver and the conductor.
- Step 2: determine the semaphore and its value.
 - Since the driver and the conductor need to exchange messages, the driver process sets a private semaphore **run**, which is used to determine whether the driver can do the work. The initial value is 0.
 - The conductor process sets a private semaphore, **stop**, to determine whether to stop and whether the conductor can open the door, with an initial value of 0.

Example 4

- Semaphore $run=0, stop=0$;



```
driver() {  
    while(TRUE) {  
        P(run)  
        //start the vehicle;  
        //drive normally;  
        //stop at the station;  
        V(stop);  
    }  
}
```



```
conductor() {  
    while(TRUE) {  
        //Get passengers on;  
        //close the door;  
        V(run);  
        //sell tickets;  
        P(stop);  
        //open the door;  
        //let passengers off;  
    }  
}
```

Summary: Use semaphore to solve problem

1. Determine concurrent and sequential operations;
2. Determine **the rules** of mutual exclusion and synchronization;
3. Determine the **procedure** of mutual exclusion and synchronization;
4. Determine the **number and meaning of semaphores**;
5. Determine the **initial value** of the semaphore;
6. Determine the **position** of P and V operations.

Note: whenever a process needs to be blocked, it must be thought of **waking it up** at some point.