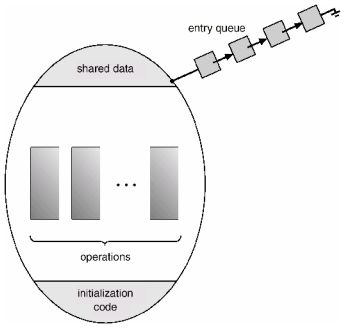
2. Monitors : definizione

Costrutti di alto livello per la sincronizzazione che permettono la sicura condivisione di tipi di dati astratti tra processi concorrenti

type *monitor-name* = **monitor**
dichiarazioni di variabili
procedure entry *P1* (…);
begin … **end**;
procedure entry *P2* (…);
begin … **end**;
…
procedure entry *Pn* (…);
begin…**end**;
begin
codice di inizializzazione
end

2. Monitors : rappresentazione



C'e' una ***coda di ingresso*** al monitor comune a tutti i processi che vogliono ***eseguire una delle entry procedure*** (operations)

2. Monitors: variabili *condition* …

Per permettere ad un processo di ***fermarsi in attesa all'interno*** di un monitor deve essere dichiarata una variabile di ***tipo condition***

var *x: condition*

… e relative operazioni

Il processo che invoca l'operazione

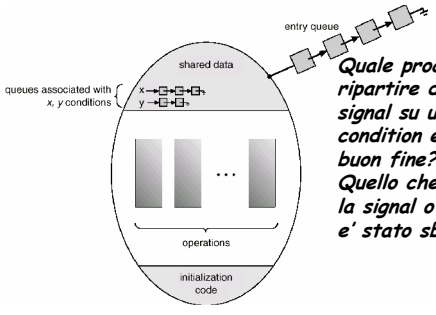
x.wait

viene sospeso finche' un altro processo invoca

x.signal

L'operazione *x.signal* fa riprendere al piu' un unico processo. Se nessun processo era sospeso, allora l'operazione non ha effetto.

Schematica rappresentazione di un monitor con variabili condition



Quale processo deve ripartire dopo che una signal su una variabile condition e' andata a buon fine?!
Quello che ha eseguito la signal o quello che e' stato sbloccato?!

Un esempio : PROD-CONS…

```
monitor ProducerConsumer
    condition full, empty;
    integer count;
    procedure insert(item: integer);
    begin
        if count = N then wait(full);
        insert_item(item);
        count := count + 1;
        if count = 1 then signal(empty)
    end;
    function remove: integer;
    begin
        if count = 0 then wait(empty);
        remove = remove_item;
        count := count - 1;
        if count = N - 1 then signal(full)
    end;
    count := 0;
end monitor;
```

… e Dining Philosophers

```
type dining-philosophers = monitor
    var state : array [0..4] of :(thinking, hungry, eating);
    var self : array [0..4] of condition ;
    procedure entry pickup (i : 0..4);
        begin
            state[i] := hungry ;
            test (i);
            if state[i] ≠ eating then self[i].wait ;
        end;

    procedure entry putdown (i : 0..4);
        begin
            state[i] := thinking;
            test (i+4 mod 5);
            test (i+1 mod 5);
        end;
```



```
procedure test(k: 0..4);
        begin
            if state[k+4 mod 5] ≠ eating
                and state[k] = hungry
                and state[k+1 mod 5] ≠ eating
                then begin
                    state[k] := eating;
                    self[k].signal;
                end;

            end;

        begin
            for i := 0 to 4
                do state[i] := thinking;
        end
```