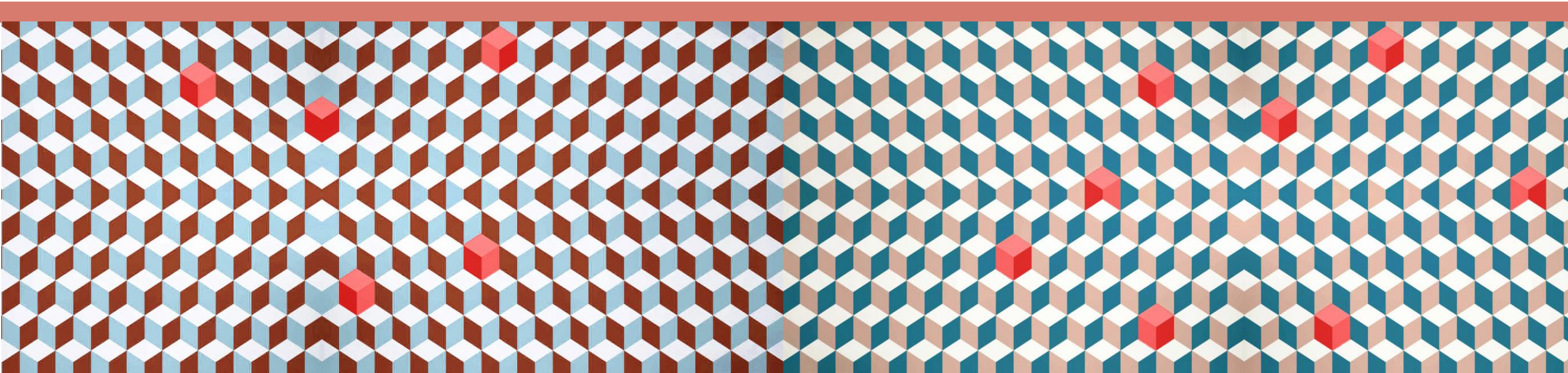


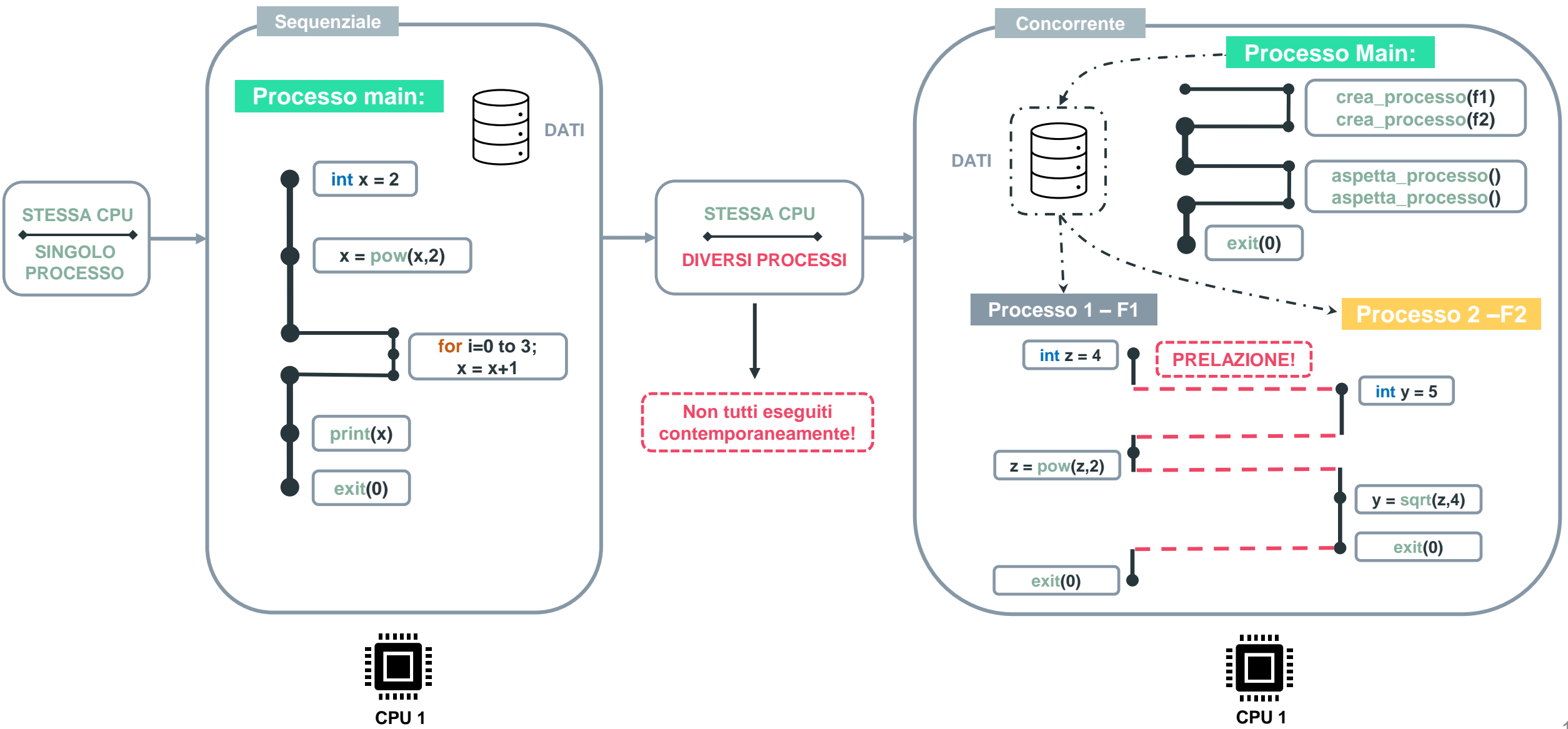
# Sincronizzazione:

Esercitazione e  
concetti

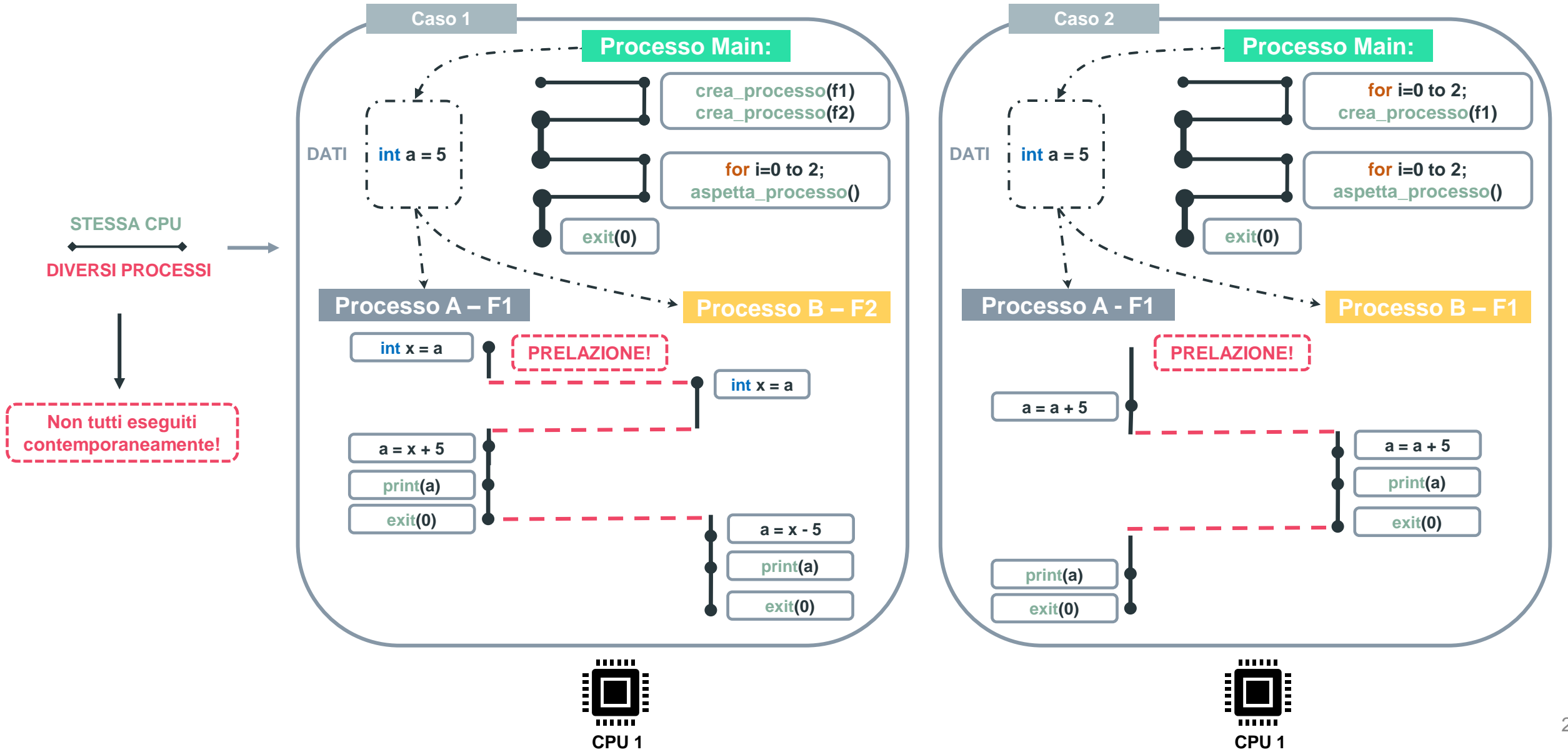
Tutor: Giovanni Hauber



# Sequenziale e Concorrente

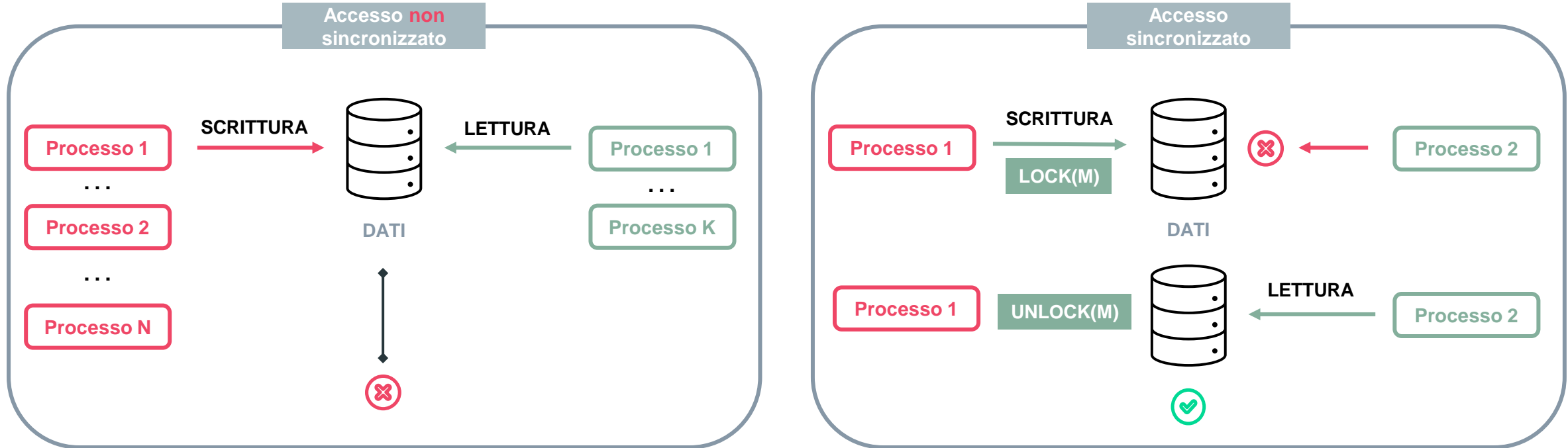


# Race Condition

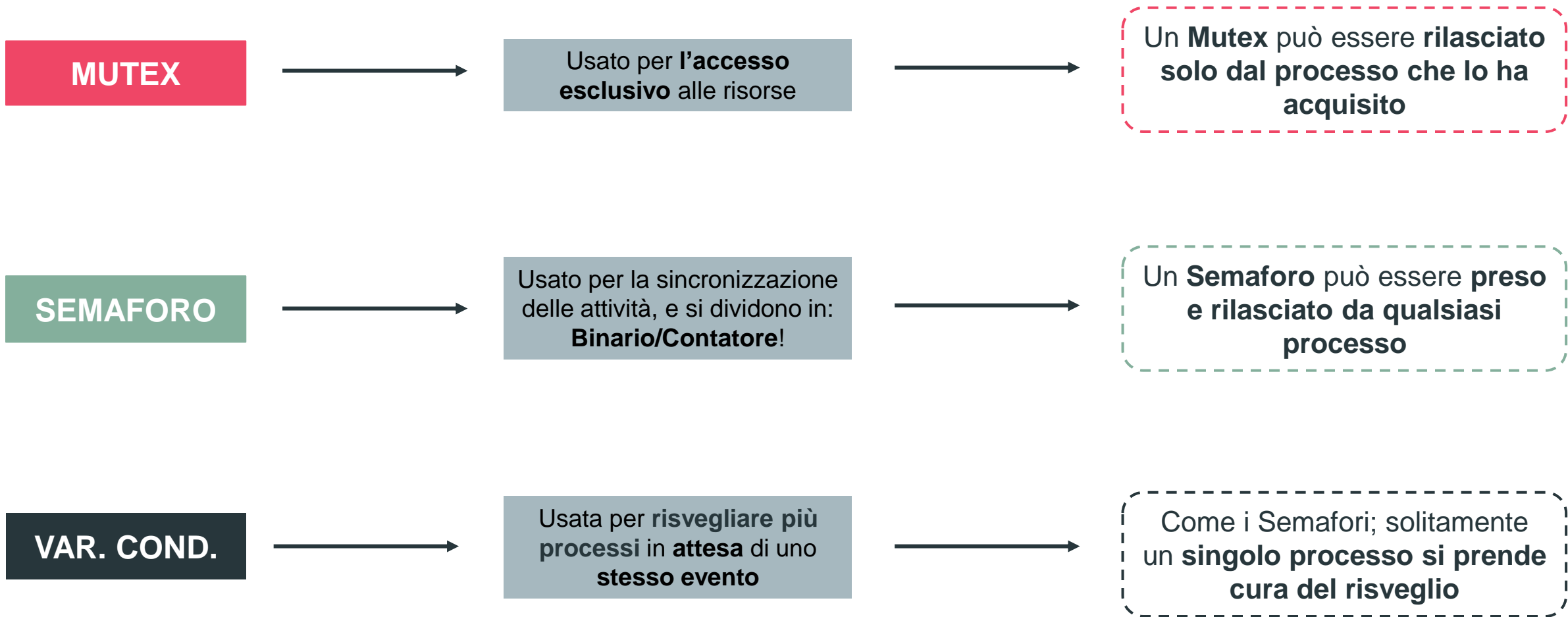


# Ma cos'è la sincronizzazione?

« La **sincronizzazione** dei processi è il task che ha come scopo sia quello di **coordinare l'accesso ai dati condivisi** in modo tale che non ci siano due processi che, contemporaneamente effettuino l'accesso alle risorse potendo generare eventuali inconsistenze, sia quello di far comunicare due o più processi. L'obiettivo è quindi quello di garantire l'accesso **in mutua esclusione e/o di coordinare diverse azioni**. »

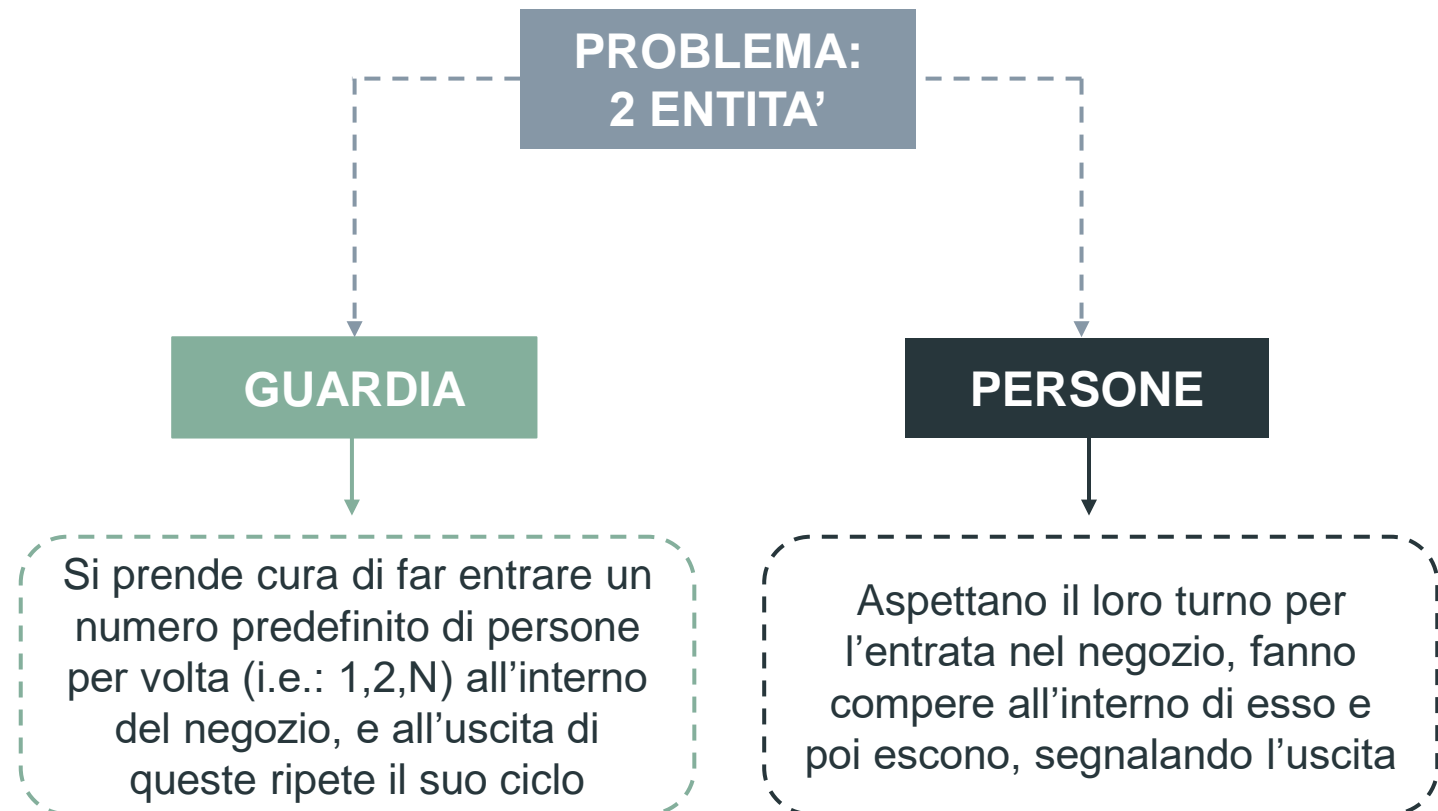


# Semafori, Mutex e variabili di condizione





# Fila ad un negozio



# L Possibile Soluzione

```
guardia() {
```

```
  repeat
```

```
    // Aspetta che arrivino persone
```

```
    wait(persone_in_fila)
```

```
    // Se ci sono posti liberi nel negozio
```

```
    wait(persone_ammesse)
```

```
    // Fai entrare una persona
```

```
    signal(turno)
```

```
  forever
```

```
}
```

```
semaforo contatore: persone_ammesse=N
semaforo contatore: persone_in_fila=0
semaforo binario: turno=0
```

```
processo_main() {
```

```
  crea_processo(guardia)
```

```
  for i=0 to K:
```

```
    crea_processo(persona)
```

```
    // aspettare che i processi finiscano
```

```
}
```

```
persona() {
```

```
  // Aggiungiti alla coda delle persone
```

```
  signal(persone_in_fila)
```

```
  // Aspetta il tuo turno
```

```
  wait(turno)
```

```
  // Fai compere nel negozio
```

```
  fai_compere();
```

```
  // Esci dal negozio
```

```
  signal(persone_ammesse)
```

```
}
```

# Fila ad un negozio: compere di scarpe





# L Possibile Soluzione

```
guardia() {
    .....
}
```

```
persona() {
```

```
// Aggiungiti alla coda delle persone
signal(persone_in_fila)
```

```
// Aspetta il tuo turno
wait(turno)
```

```
// Entra nel negozio e segnala la presenza
signal(cliente)
```

```
// Aspetta l'aiuto dell'addetto
wait(addetto)
```

```
// Paga le scarpe prese
paga_scarpe();
```

```
// Esci dal negozio
signal(persone_ammesse)
```

```
}
```

```
semaforo contatore: persone_ammesse=N
semaforo binario: turno=0
semaforo contatore: persone_in_fila=0
```

```
// Due nuovi semafori contatore
semaforo contatore: cliente=0
semaforo contatore: addetto=0
```

```
// Mutex per mutua esclusione
mutex: accesso_scaffale
```

```
// Risorsa: array di P elementi
// con valore 100 ognuno
array: scaffale[P] = 100
```

```
processo_main() {
```

```
-- crea_processo(guardia)
```

```
for i=0 to J:
    crea_processo(addetto)
```

```
for i=0 to K:
    crea_processo(persona)
```

```
// aspettare che i processi finiscano
```

```
}
```

```
addetto() {
```

```
repeat
```

```
// Aspetta che un cliente arrivi in negozio
wait(cliente)
```

```
// Genera posto casuale dello scaffale
posto_scarpe = rand(1,P)
```

```
// Accedi allo scaffale in mutua esclusione:
// Decrementa di 1 la disponibilità dello scaffale
// nel posto P, quindi rilascia la mutua esclusione
wait(Mutex)
scaffale[P] -= 1
signal(Mutex)
```

```
// Segnala al cliente che può andare
signal(addetto)
```

```
forever
```

```
}
```

In un negozio, c'è un barbiere che taglia i capelli, una poltrona da barbiere e  $n$  sedie per i clienti i quali si fanno tagliare i capelli e/o rimangono in attesa. Se non ci sono clienti, il barbiere si mette sulla sedia e si addormenta, quando arriva un cliente esso sveglia il barbiere e si fa tagliare i capelli sulla sedia e se arriva un cliente mentre il barbiere sta tagliando i capelli a un altro cliente, quest'ultimo si mette in attesa su una delle sedie. Se tutte le sedie sono occupate, il cliente se ne va scocciato!

In un negozio, c'è un **barbiere che taglia i capelli**, una poltrona da barbiere e n sedie per i **clienti i quali si fanno tagliare i capelli e/o rimangono in attesa**.

Devono essere rispettate le seguenti condizioni:

- Se **non ci sono clienti**, il **barbiere** si mette sulla sedia e si **addormenta**
- Quando **arriva un cliente**, **sveglia il barbiere** e si fa **tagliare i capelli** sulla sedia
- Se **arriva un cliente mentre il barbiere sta tagliando i capelli** a un altro cliente, quest'ultimo **si mette in attesa** su una delle sedie
- Se **tutte le sedie sono occupate**, il **cliente se ne va scocciato!**

# L Possibile soluzione:

```
process Barbiere {  
    while (true) {  
        - attendi cliente  
        - // taglia capelli  
        - libera poltrona  
    }  
}
```

```
process Cliente {  
    - verifica se c'è posto, altrimenti esci  
    - attendi turno  
    - // fatti tagliare i capelli  
}
```

# Possibile soluzione:

```
main() {

    crea_processo(Barbiere)

    for i=0 to K: (K >>> N)
        crea_processo(Cliente)

    // attendi uscita se necessario

}

process Barbiere {
    while (true) {
        wait(cliente_in_attesa)
        // taglia capelli
        signal(poltrona)
    }
}
```

```
mutex:          accesso_risorse          =          1
semaforo contatore: cliente_in_attesa = 0
Semaforo binario: poltrona = 1
int sedieLibere = N

process Cliente {
    lock(accesso_risorse)
    if (sedieLibere==0) {
        unlock(accesso_risorse)
        return;
    }

    sedieLibere--;
    unlock(accesso_risorse)
    signal(cliente_in_attesa)
    wait(poltrona)
    // Capelli tagliati
    lock(accesso_risorse)
    sedieLibere++;
    unlock(accesso_risorse)

}
```



- In un ristorante self-service, i clienti, dopo aver mangiato, dispongono i vassoi in  $M$  contenitori, ognuno di  $K$  ripiani.
- **Periodicamente**, un addetto sceglie un contenitore **tra quelli in cui ci sono più ripiani liberi**, lo svuota, lava i piatti e riporta il contenitore in sala.

Consiglio: I contenitori in cui riporre i vassoi sono  $M$  ed ognuno di essi ha  $K$  ripiani (i.e: 5 contenitori, ognuno dei quali ha 3 ripiani liberi). Questo può essere risolto usando

- un'array 2D `contenitori[M][K]` e l'accesso in mutua esclusione ad esso per il controllo
- un'array di semafori contatori con una struttura ausiliaria per il controllo (a cui si accede sempre in mutua esclusione).

# L Possibile soluzione: Main

```
main() {
```

```
    crea_processo(addetto)
```

```
    // Nota:  $J > M$ !
```

```
    for i=0 to J:
```

```
        crea_processo(Cliente)
```

```
}
```

```
// Semafori contatore: il primo rappresenta i contenitori  
// disponibili, il secondo invece i ripiani per lo specifico  
// contenitore (array di semafori contatore)
```

```
semaforo contatore: contenitore_disponibile=M;
```

```
semaforo contatore: ripiano_disponibile[M]=K;
```

```
// Mutex per l'accesso alla sezione critica
```

```
mutex: accesso_risorse=1;
```

```
// Indice che tiene conto del contenitore a cui stiamo facendo  
// accesso
```

```
int primo_contenitore_libero=0;
```

```
// Struttura ausiliaria che tiene conto dell'M-simo  
// contenitore quanto risulti pieno, per facilitare il controllo  
// da parte dell'addetto
```

```
int ripiani_contenitori[M]=0
```

```
// Costante
```

```
int VASSOI0=1
```

# L Possibile soluzione: Cliente

```

cliente(){
    while(1) {
        wait(contenitore_disponibile);

        // Prendo la mutua esclusione, faccio una copia locale ed incremento
        lock(accesso_risorse)
        contenitore_loc = primo_contenitore_libero ++
        unlock(accesso_risorse)

        // La wait simula la presa di quello specifico ripiano, ed accediamo
        // all'array con un indice unico (quindi non ce necessita di mutex)!
        wait(ripiano_disponibile[contenitore_loc])
        ripiani_contenitori[contenitore_loc] += VASSOIO;

        // Non c'è la signal sul ripiano poiche è l'addetto che lo svuota!

        lock(accesso_risorse)
        primo_contenitore_libero -= 1 // decremento l'indice del contenitore incrementato prima
        unlock(accesso_risorse)

        signal(contenitore_disponibile)
    }
}

```

```

semaforo contatore: contenitore_disponibile=M;
semaforo contatore: ripiano_disponibile[M]=K;
mutex: accesso_risorse=1;
int primo_contenitore_libero=0;
int ripiani_contenitori[M]=0
int VASSOIO=1

```

# L Possibile soluzione: Addetto

```

addetto() {
Repeat
  // «periodicamente svuoto»
  sleep(rand(0.00001, 0.1))
  min = inf
  ind = -1
  // Per ogni ripiano (M IN TOTALE)
  for(i=0;i<M;i++){
    lock(mutex) // nota: qui il mutex potrebbe essere opzionale!
    if (ripiani_contenitori[i] <= min && ripiani_contenitori[i] != 0) {
      unlock(mutex)
      ind = i
      min = ripiani_contenitori[i]
    } else unlock(mutex)
  }

  if (min != inf) {
    for(i=0;i<min;i++) signal(ripiano_pieno[i])
  }

Forever
}

```

```

semaforo contatore: contenitore_disponibile=M;
semaforo contatore: ripiano_disponibile[M]=K;
mutex: accesso_risorse=1;
int primo_contenitore_libero =0;
int ripiani_contenitori[M]=0
int VASSOIO=1

```