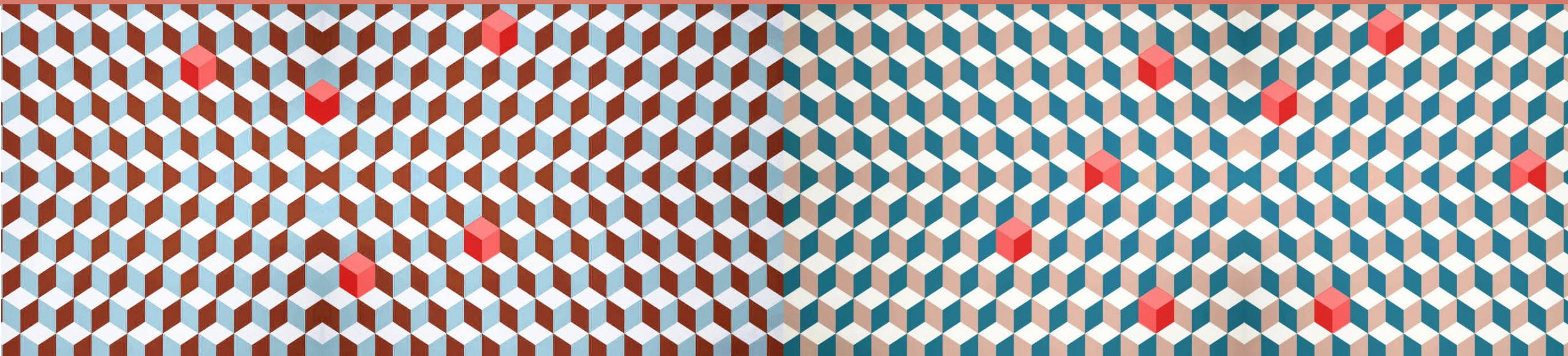
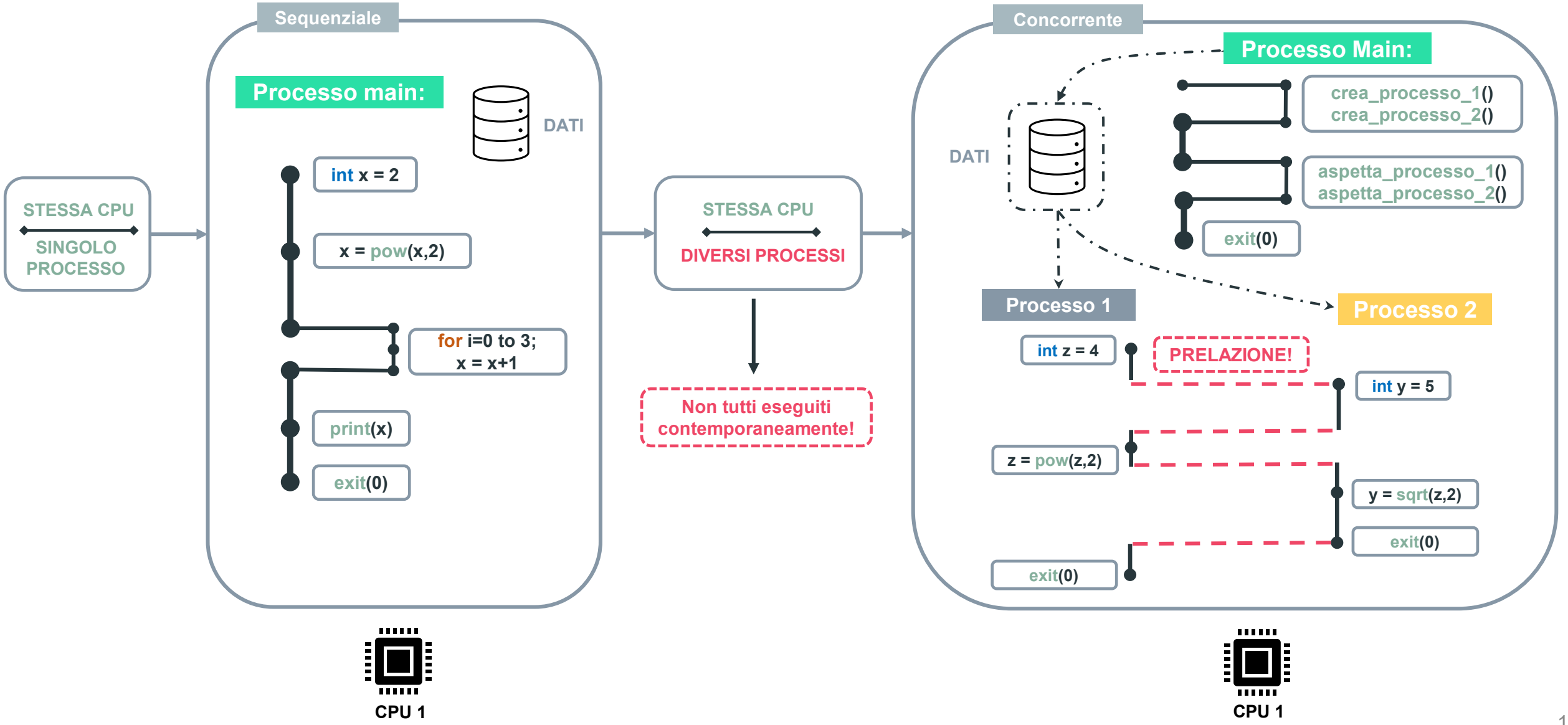
Sincronizzazione:

Esercitazione e concetti

Tutor: Giovanni Hauber



Sequenziale e Concorrente

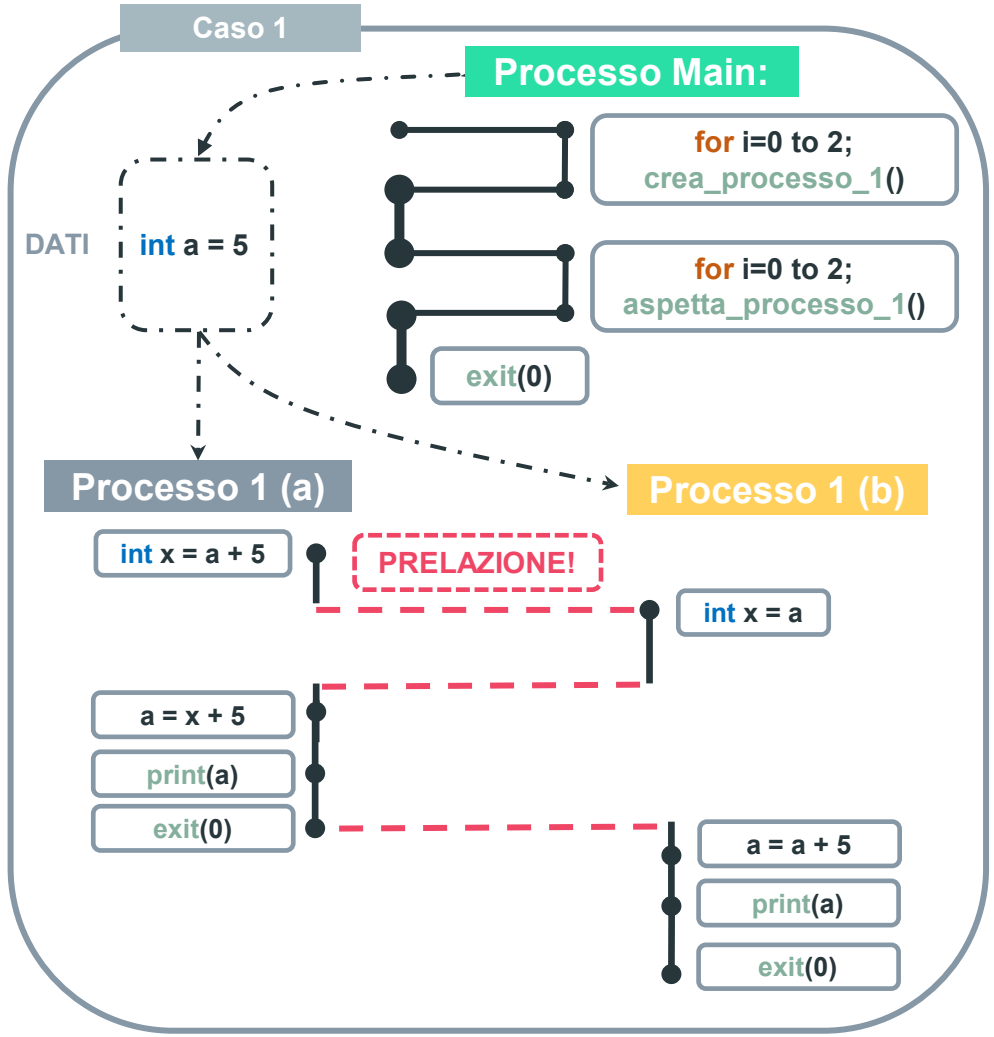


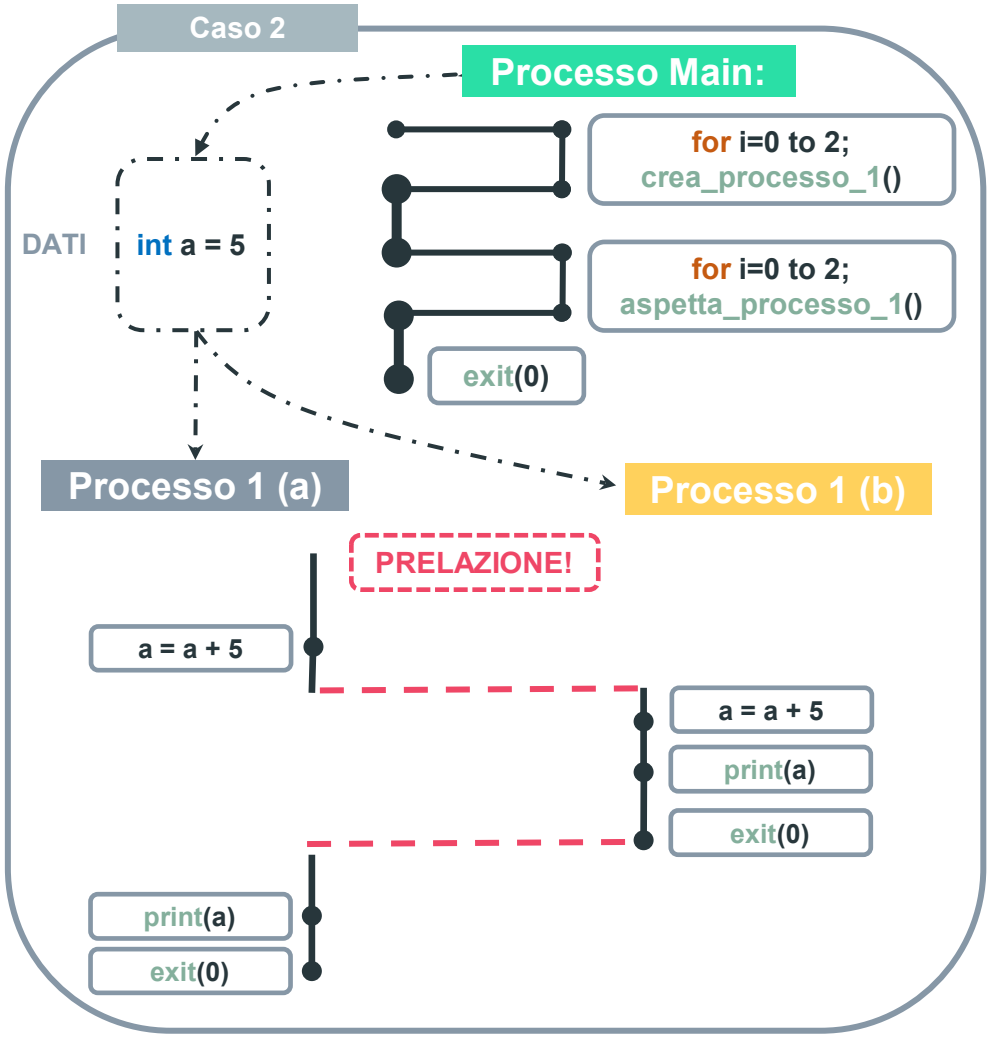
Race Condition

STESSA CPU

DIVERSI PROCESSI

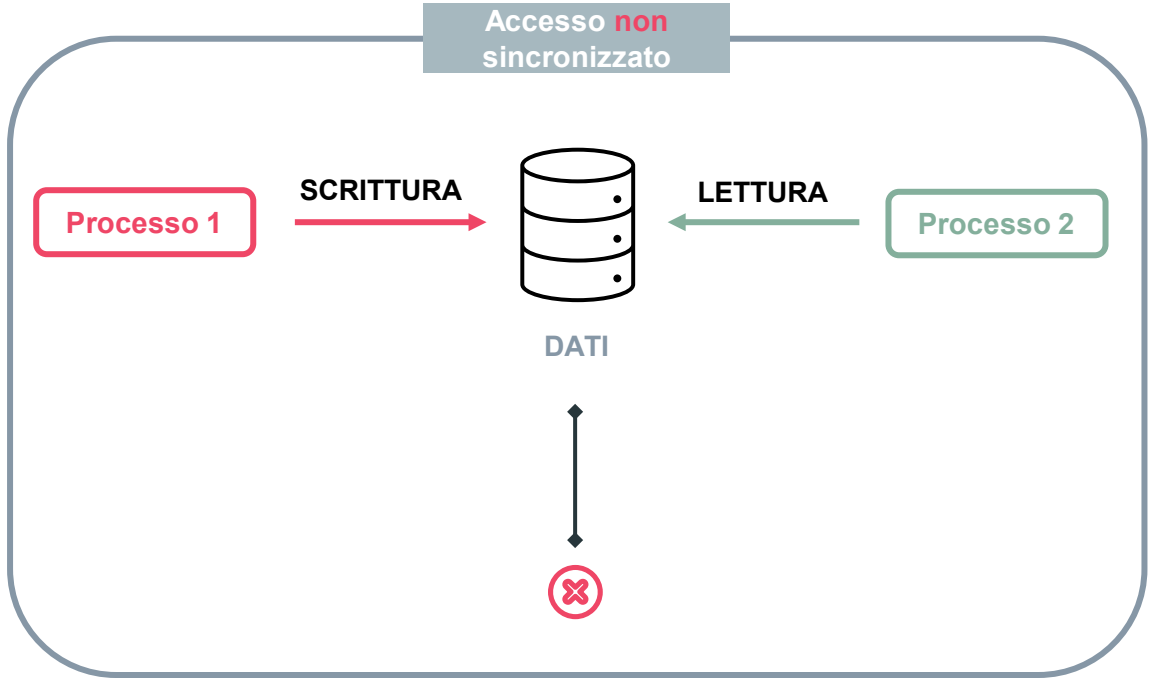
Non tutti eseguiti contemporaneamente!

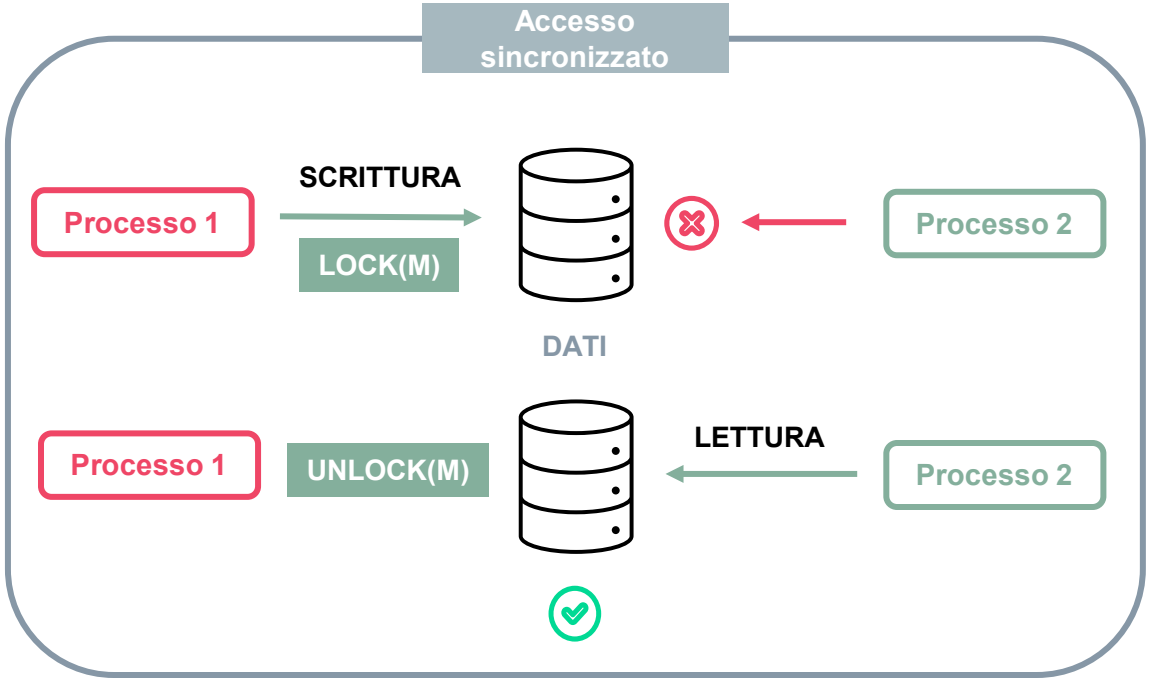




Ma cos'è la sincronizzazione?

« La sincronizzazione dei processi è il task che ha come scopo 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. L'obiettivo è quindi quello di garantire l'accesso in mutua esclusione. »





Semafori, Mutex e variabili di condizione

MUTEX → Usato per **l'accesso esclusivo** alle risorse → Un **Mutex** può essere **rilasciato solo dal processo che lo ha acquisito**

SEMAFORO → Usato per la sincronizzazione delle attività, e si dividono in: **Binario/Contatore**! → Un **Semaforo** può essere **preso e rilasciato da qualsiasi processo**

VAR. COND. → Usata per **risvegliare più processi** in **attesa** di uno **stesso evento** → Come i Semafori; solitamente un **singolo processo si prende cura del risveglio**

Fila ad un negozio

PROBLEMA: 2 ENTITA'

GUARDIA

Si prende cura di far entrare un numero predefinito di persone per volta (i.e.: 1,2,N) all'interno del negozio, e all'uscita di queste ripete il suo ciclo

PERSONE

Aspettano il loro turno per l'entrata nel negozio, fanno compere all'internodi esso e poi escono, segnalando l'uscita

Possibile Soluzione

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

processo_main() {

  crea_processo(guardia)

  for i=0 to K:
    crea_processo(persona)

  // aspettare che i processi finiscano

}
```

```
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

}
```

```
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

PROBLEMA: 3 ENTITA'

PERSONE

… E una volta entrati, **aspettano che un commesso li serva**; appena serviti, escono dal negozio e segnalano l'uscita.

GUARDIA

IDENTICO!

ADDETTI

Appena liberi, **servono i clienti** ed **accedono ad una mensola di scarpe** per trovare il paio di scarpe che richiedono.

Possibile Soluzione

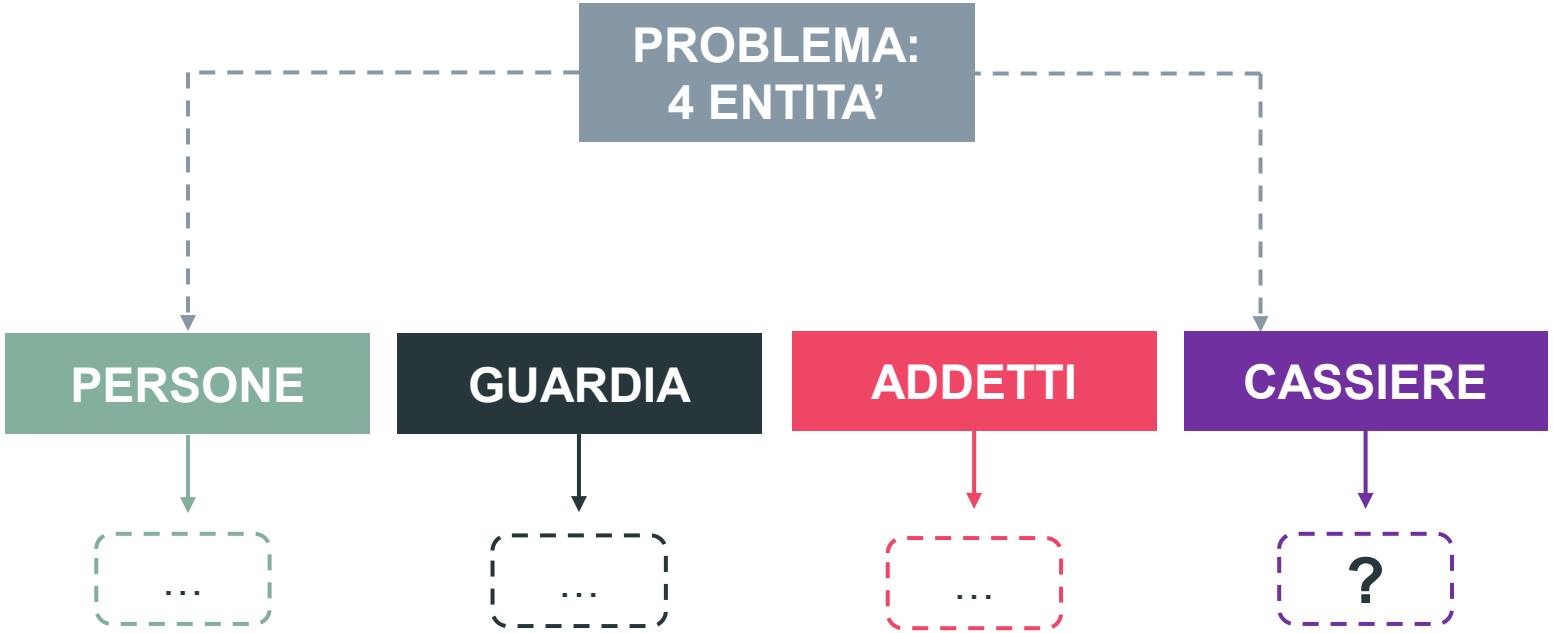
```
guardia() {
. . . . .
}
```

```
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=J

// 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

}
```

Fila ad un negozio: cassiere



Esercizio

- 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.

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_disponible[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 VASSOIO=1
```

Possibile soluzione: Cliente

```
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
```

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

    lock(accesso_risorse)
    contenitore_loc = primo_contenitore_libero ++
    unlock(accesso_risorse)

    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
    unlock(accesso_risorse)

    signal(contenitore_disponibile)
  }
}
```

Possibile soluzione: Addetto

```
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
```

```
addetto() {
Repeat

  sleep(1)
  min = inf
  ind = -1
  for(i=0;i<M;i++){
    if (ripiani_contenitori[i] <= min && ripiani_contenitori[i] != 0) {
    ind = i
    min = ripiani_contenitori[i]
   }
  }


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

Forever
}
```