Introduzione alla Programmazione Concorrente

Corso di Laurea in Ingegneria Informatica

Università degli Studi di Napoli Federico II

Anno Accademico 2024/2025, Canale San Giovanni

Introduzione alla Sincronizzazione dei Processi

- Sommario:
  - Concorrenza e Race condition
  - Tipi di interazione tra i processi
  - Modelli di interazione
- Riferimenti:
  - P. Ancilotti, M.Boari “Programmazione concorrente e distribuita”, Mc-Graw-Hill (Capp.2 e 3)

Programmazione concorrente

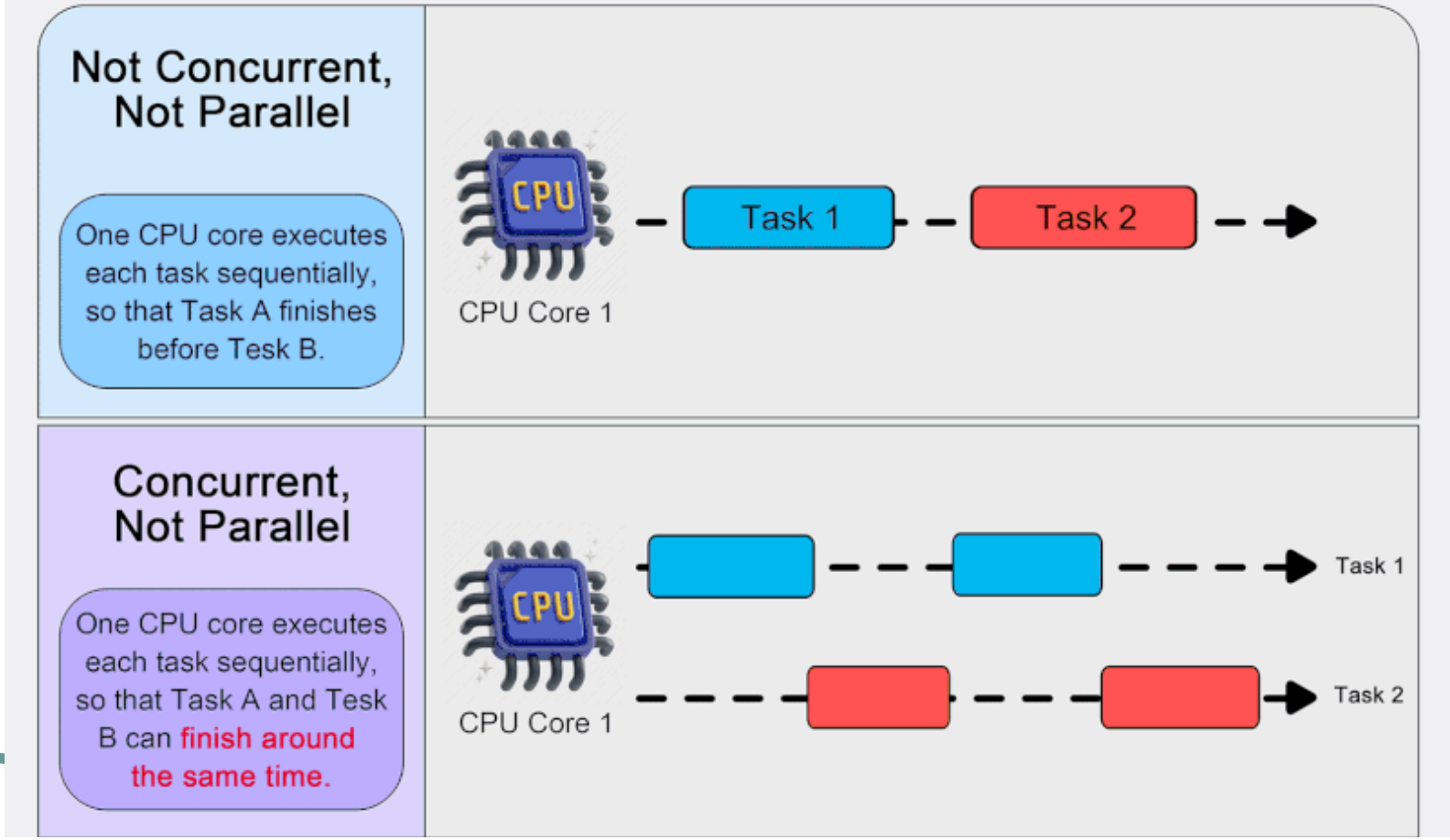
- L'insieme delle tecniche, delle metodologie e degli strumenti...
- ...per lo sviluppo di software come un **insieme di attività svolte simultaneamente**

Multiprogrammazione

- Multiprogrammazione: la forma più elementare di programmazione concorrente
- Esecuzione intercalata di più processi (o threads) sulla stessa CPU

Multiprogrammazione

- Il sistema diventa una macchina astratta che dispone di più processori virtuali, uno per ogni processo



Multiprogrammazione

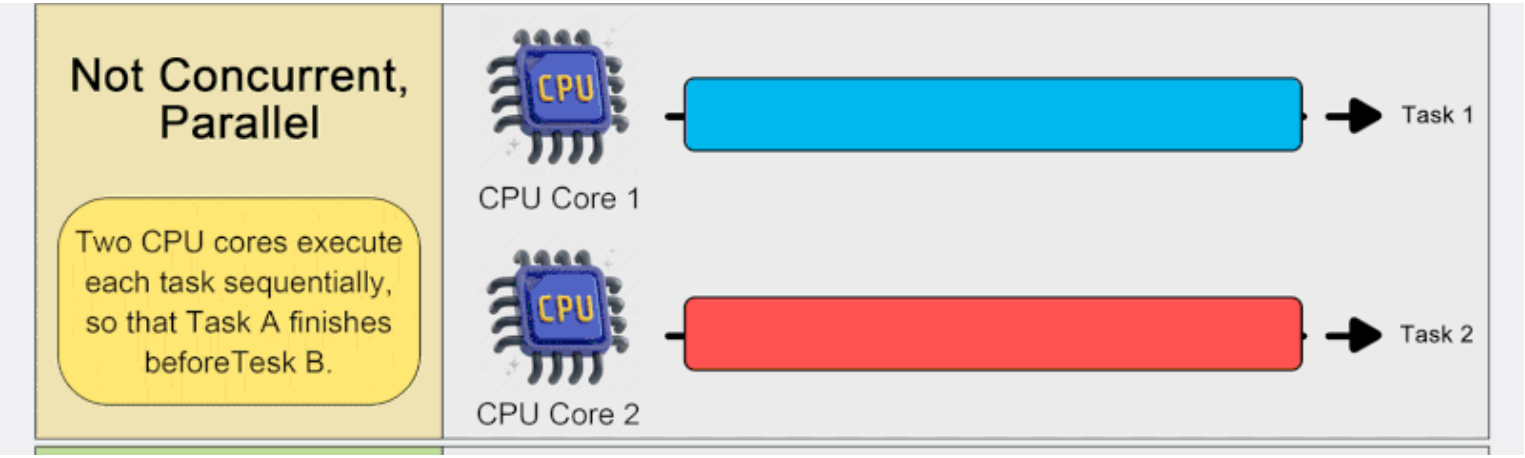
- Il sistema diventa una macchina astratta che dispone di più processori virtuali, uno per ogni processo

La multiprogrammazione:

- Migliora l'efficienza complessiva del sistema...
- ...ma non migliora la velocità di esecuzione del singolo programma

Parallelismo

- Il parallelismo utilizza più CPU fisiche su cui eseguire più processi contemporaneamente
- Concetto differente da concorrenza (no "alternanza")



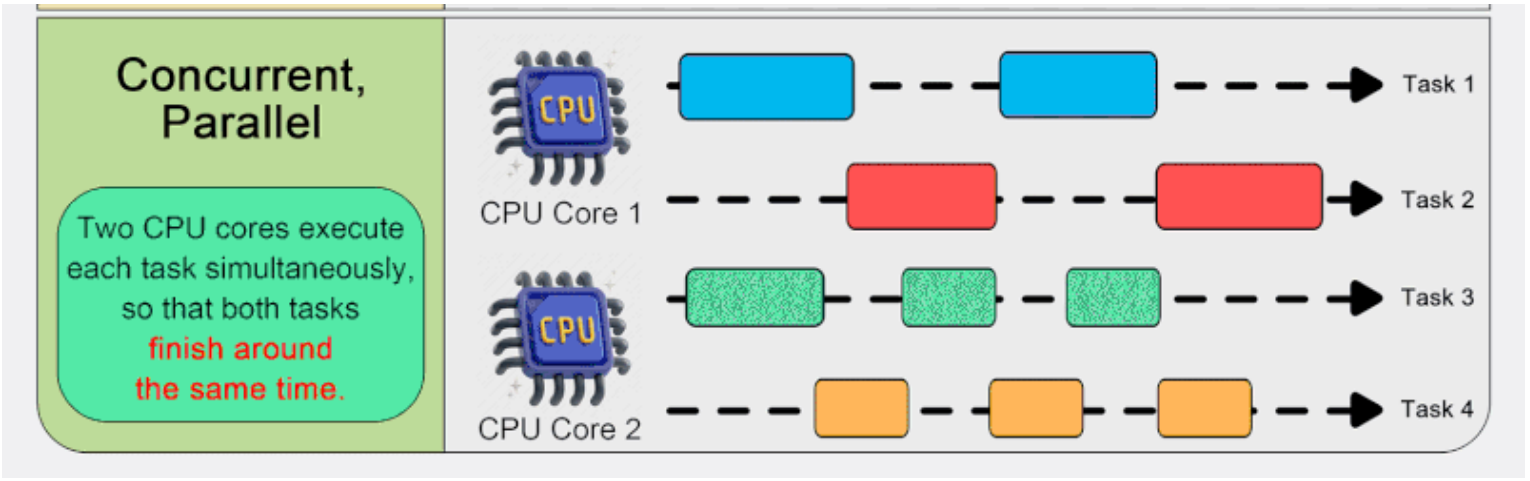
Parallelismo

- Il parallelismo utilizza più CPU fisiche su cui eseguire più processi contemporaneamente
- Concetto differente da concorrenza (no "alternanza")

Anche il parallelismo, se usato da solo, **non** migliora la **velocità di esecuzione del singolo programma**

Concorrenza vs. parallelismo

- Concorrenza e parallelismo possono essere applicati insieme
- Per accelerare il singolo programma, è necessario **suddividerlo in più elaborazioni concorrenti**, ed eseguirle su **più processori**



Programmazione concorrente

- Nella programmazione concorrente, il programmatore **definisce delle elaborazioni concorrenti**
- Ogni elaborazione è eseguita in un processo/thread

Migliora la **velocità di esecuzione di un programma** rispetto al caso non-concorrente

Elaborazioni concorrenti

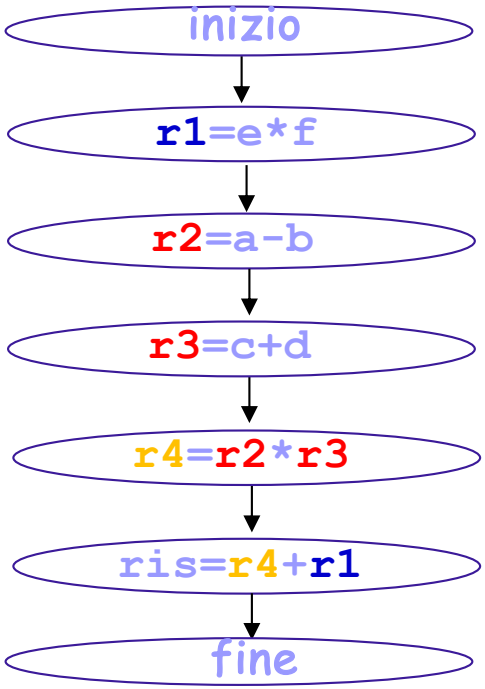
- **Esempio:** Supponiamo di dover scrivere un programma per il calcolo della seguente espressione aritmetica

$$(a-b) * (c+d) + (e*f)$$

Grafo di precedenza

Possiamo rappresentare le elaborazioni con un grafo di precedenza

$$(a-b)*(c+d)+(e*f)$$



Nodi: singole elaborazioni del programma

Archi: precedenze temporali tra le elaborazioni

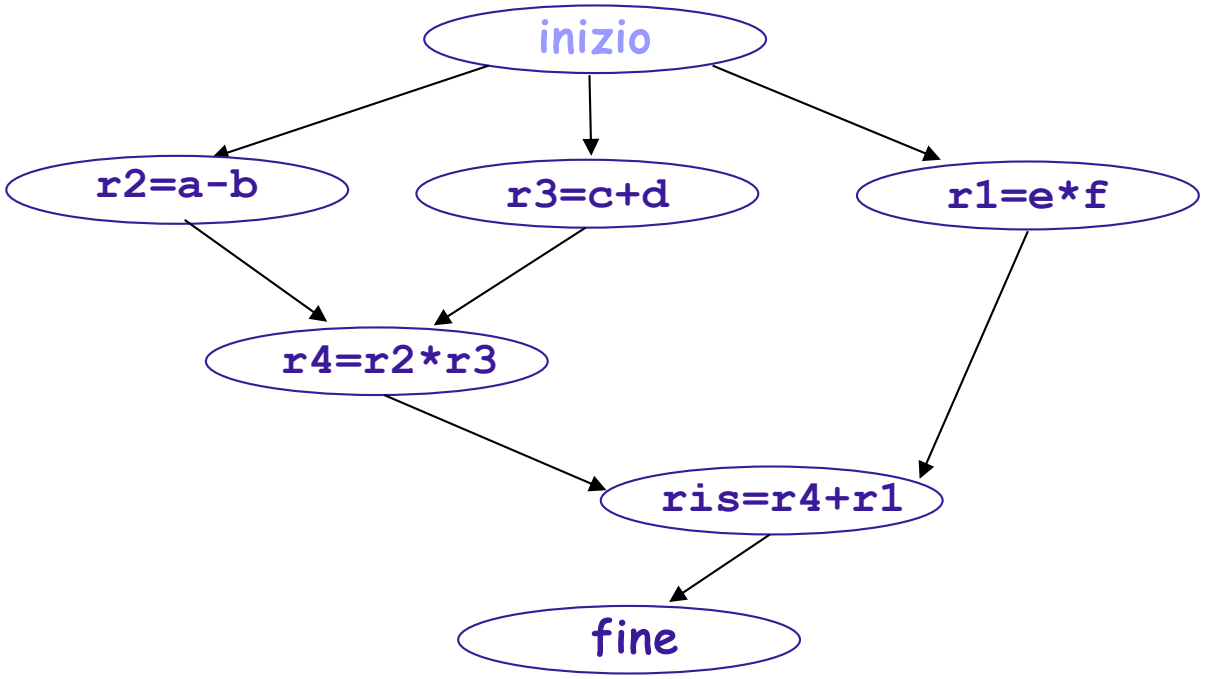
Nella **programmazione tradizionale** (non concorrente), le elaborazioni sono **sequenziali**.

Non vi è concorrenza.

Grafo di precedenza

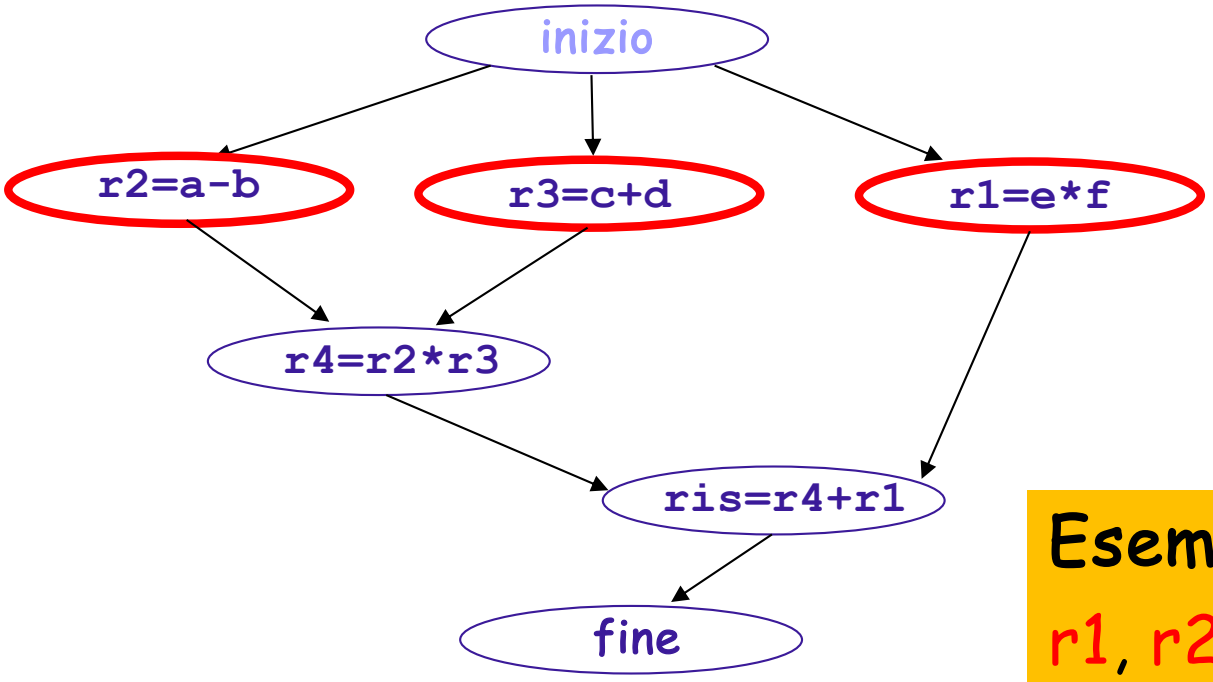
- L'ordinamento non deve essere necessariamente sequenziale
- Molte applicazioni possono essere rappresentate come processi non sequenziali

Grafo di precedenza ad ordinamento parziale



In un processo non-sequenziale, alcune elaborazioni **non hanno una relazione di precedenza temporale** (ordinamento parziale)

Grafo di precedenza ad ordinamento parziale



Esempio:

r1, r2, r3 possono essere eseguite in concorrenza.

Il risultato finale ("ris") è indifferente all'ordine con cui sono eseguite.

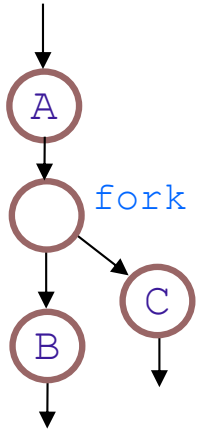
Programmazione concorrente

- Primitive per **definire attività indipendenti** (processi, threads)
- Primitive per la **comunicazione e sincronizzazione** tra attività eseguite in modo concorrente

Fork/Join

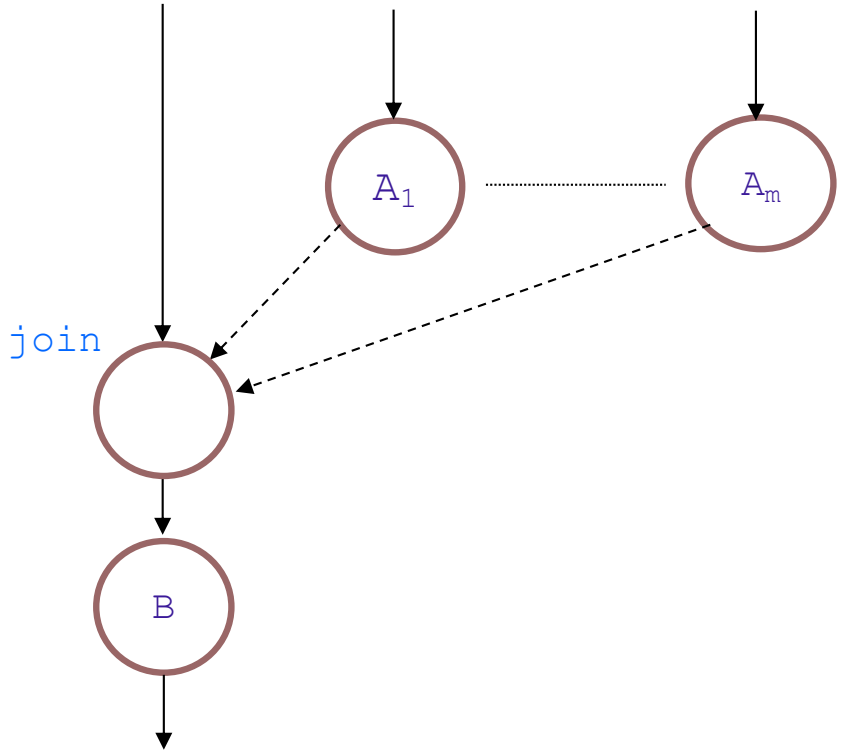
- L'esecuzione di una operazione di fork crea ed attiva un nuovo processo "figlio"
- Esso esegue in concorrenza con il processo "padre"

```
    process  p;
    =======
A:  ........;
    p = fork func;
B:  ........;
    =======
    =======

void  func () {
    C: ........;
    =======
}
```



Fork/Join

- L'operazione di join consente al processo di sincronizzarsi con la terminazione di un altro processo, creato tramite la fork



….le problematiche

- Comunicazione tra processi
- Condivisione delle risorse
- Sincronizzazione tra processi
- Assegnazione del tempo del processore

Esempio di (mancata) sincronizzazione tra processi

```
int counter = 0;

processo1() {
     for(1 ... 1,000,000)
          counter++;
}

processo2() {
     for(1 ... 1,000,000)
          counter++;
}
```

Al termine,
counter = 2,000,000
... in teoria!

Esempio di (mancata) sincronizzazione tra processi

```
6   volatile int counter = 0;
7   int loops;
8
9   void *worker(void *arg) {
10      int i;
11      for (i = 0; i < loops; i++) {
12          counter++;
13      }
14      return NULL;
15  }
16
17  int main(int argc, char *argv[]) {
18      if (argc != 2) {
19          fprintf(stderr, "usage: threads <loops>\n");
20          exit(1);
21      }
22      loops = atoi(argv[1]);
23      pthread_t p1, p2;
24      printf("Initial value : %d\n", counter);
25      Pthread_create(&p1, NULL, worker, NULL);
26      Pthread_create(&p2, NULL, worker, NULL);
27      Pthread_join(p1, NULL);
28      Pthread_join(p2, NULL);
29      printf("Final value   : %d\n", counter);
30      return 0;
31  }
```

Implementazione **multi-thread** dell'esempio del counter

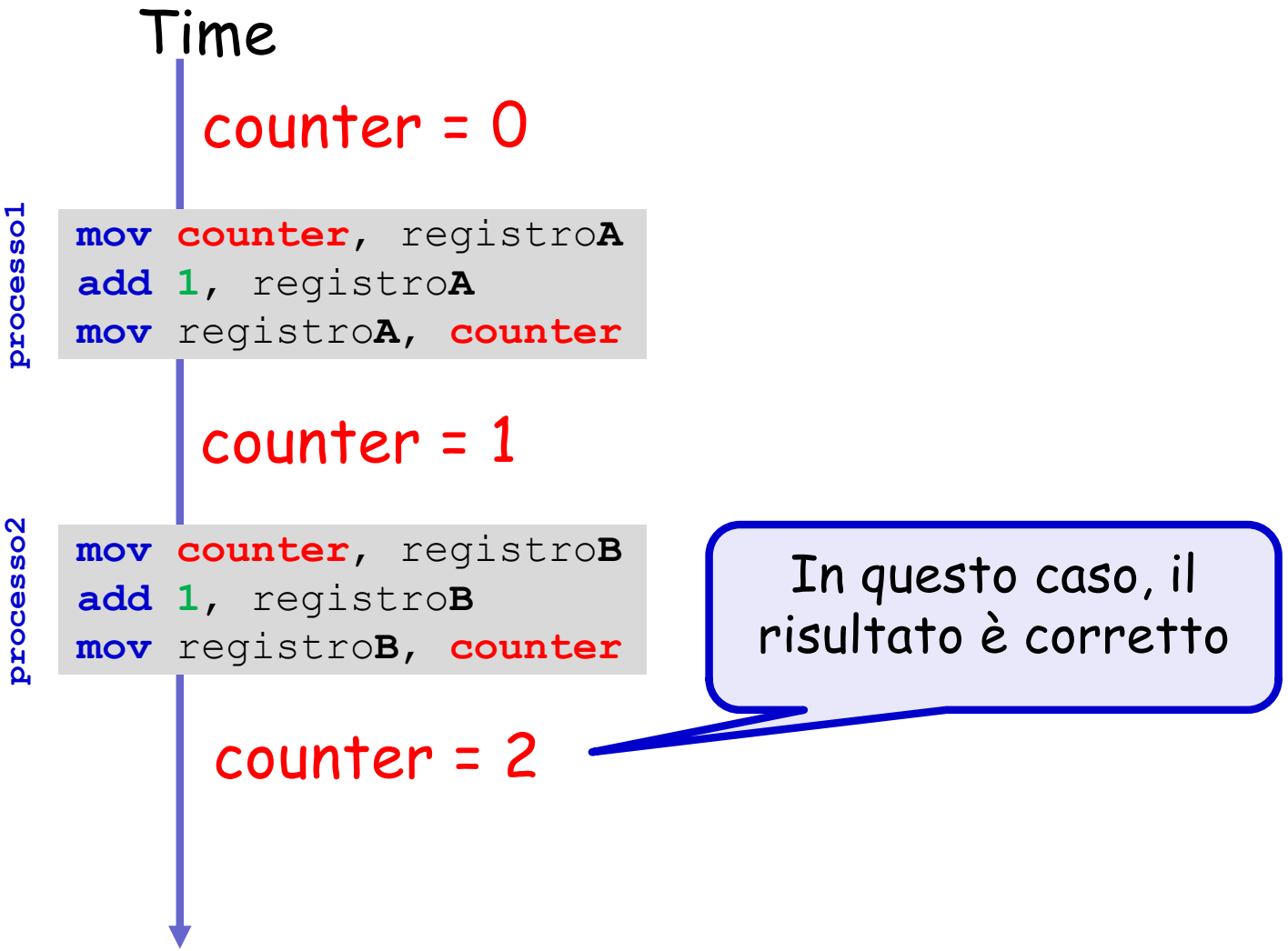
https://github.com/remzi-arpacidusseau/ostep-code/ (cartella "intro")

Esempio di (mancata) sincronizzazione tra processi

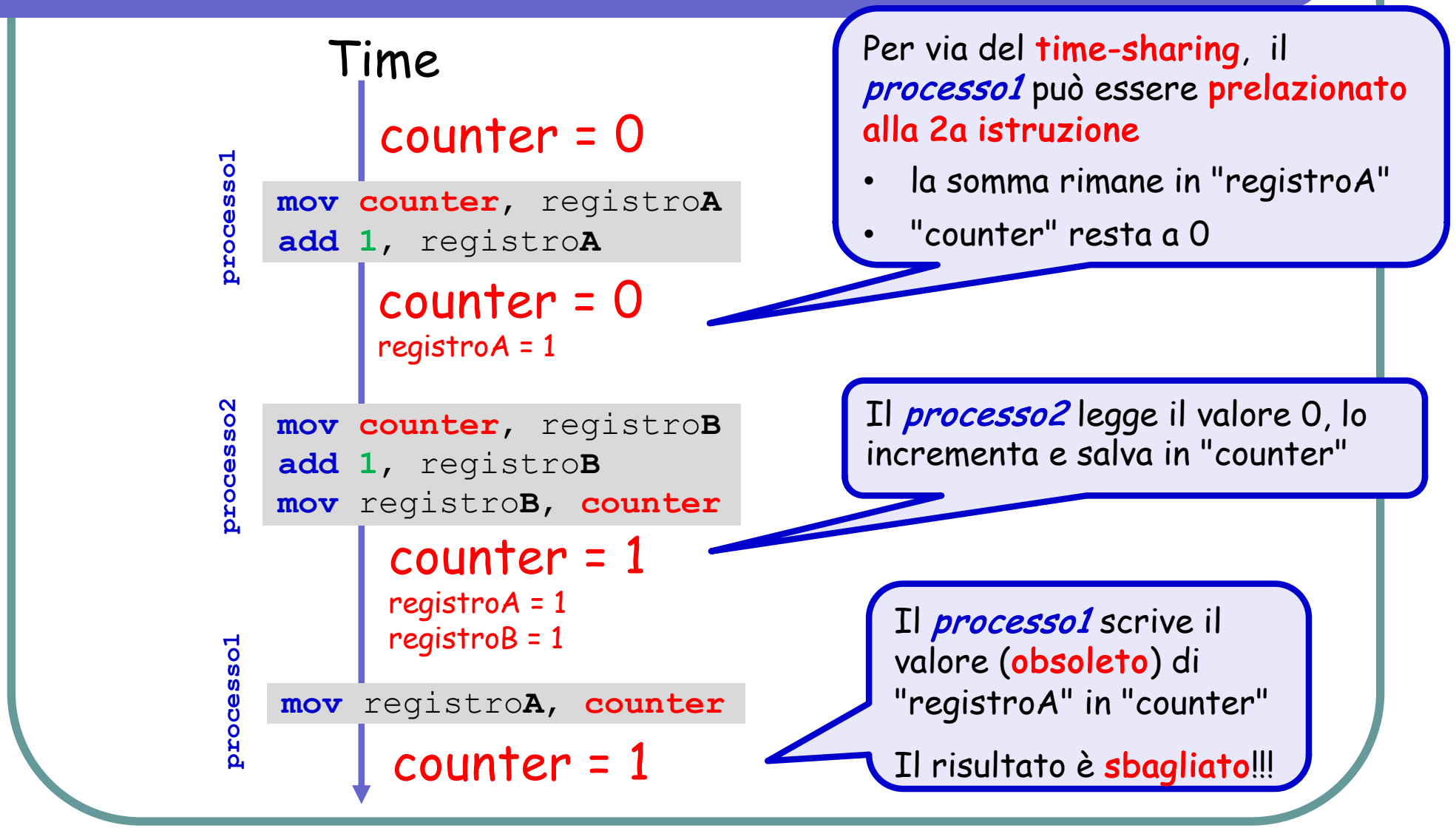
```
int counter = 0;

processo1() {
     for(1 ... 1,000,000)
          mov counter, registroA
          add 1, registroA
          mov registroA, counter
}

processo2() {
     for(1 ... 1,000,000)
          mov counter, registroB
          add 1, registroB
          mov registroB, counter
}
```

Per la CPU, l'istruzione
counter++
equivale a 3 istruzioni macchina

Esempio di (mancata) sincronizzazione tra processi



Esempio di (mancata) sincronizzazione tra processi



Race Condition

…si dice che per l'esempio precedente si verifica una "**Race Condition**", ovvero…

Una condizione in cui più processi leggono o scrivono dati **condivisi**, e il risultato finale di tali operazioni **dipende dall'ordine (e dalla "velocità")** di esecuzione delle istruzioni dei processi

Processi interagenti

- **Processi interagenti:**
  - l'esecuzione di **P1** è influenzata da **P2**, e viceversa
  - Il comportamento del programma può essere **imprevedibile** ("non riproducibile")...
  - ..., ossia, dipendere dall'ordine in cui sono eseguiti nel sistema

Tipi di interazione

Competizione: uso di risorse **comuni** che **non devono** essere utilizzate contemporaneamente (**mutua esclusione**)

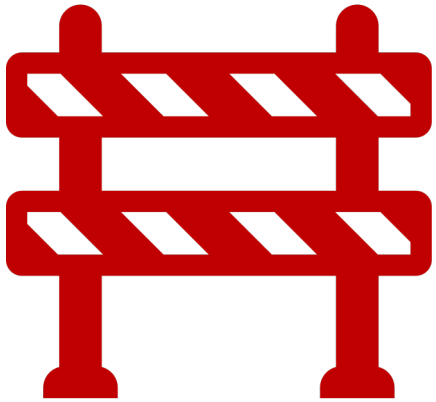
Cooperazione: scambio di informazioni per portare a termine una elaborazione comune (**comunicazione**)

Interferenza: **erronea soluzione** di problemi di competizione e di cooperazione, oppure uso **non autorizzato** di risorse comuni

Sincronizzazione

Per un corretto funzionamento, è necessario inserire nel programma dei **vincoli nella esecuzione** alle elaborazioni concorrenti

- locks
- semafori
- barriere
- guardie
- monitor
- ...

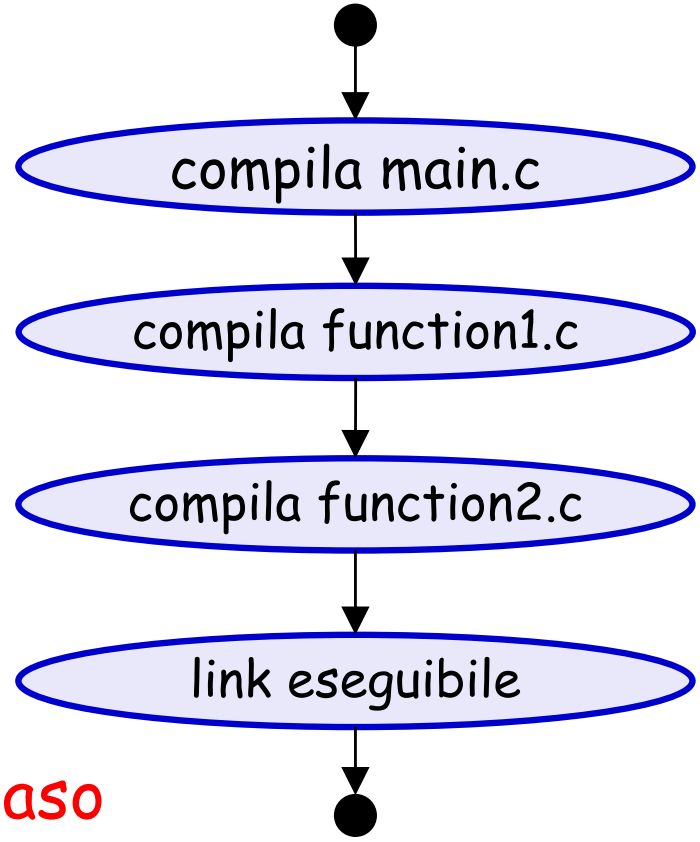


I vincoli di sincronizzazione **riducono la velocità di esecuzione dei programmi concorrenti**, per garantire la **correttezza dei risultati**

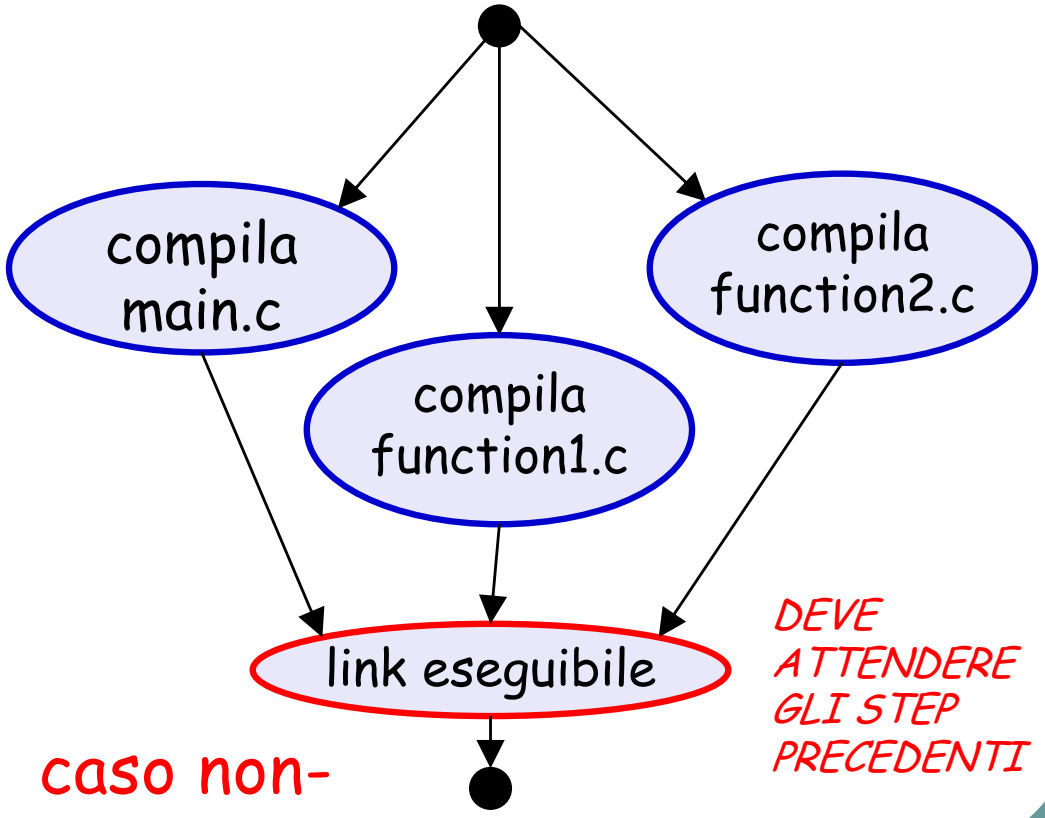
Esempio di sincronizzazione

- Consideriamo ad esempio il seguente comando:

```
gcc  main.c  function1.c  function2.c  -o eseguibile
```



caso sequenziale



caso non-sequenziale

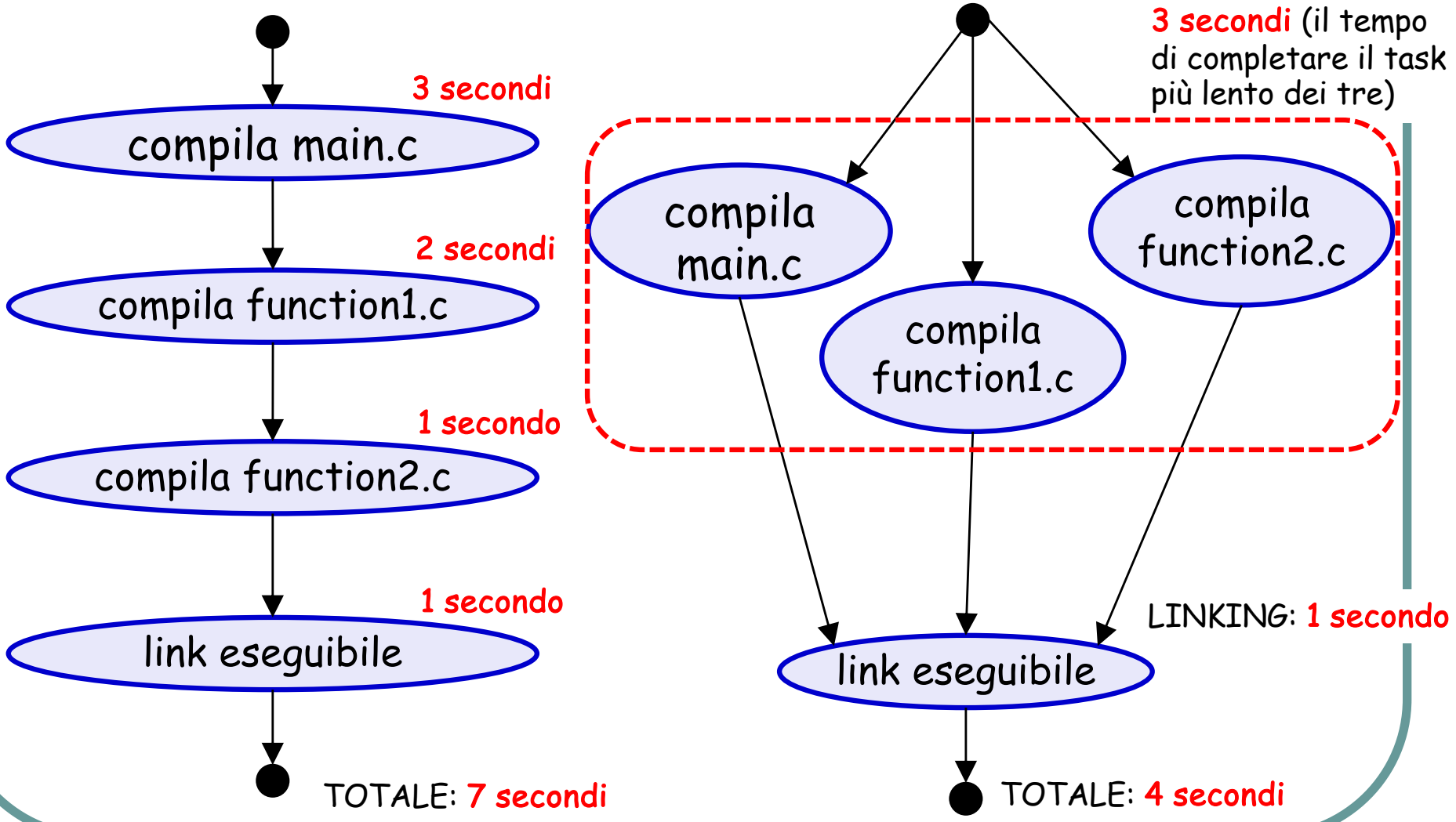
Esempio di sincronizzazione

- Supponiamo che su una macchina monoprocessore ci vogliano:
  - 3 secondi per compilare main.c
  - 2 secondi per compilare function1.c
  - 1 secondo per compilare function2.c
  - 1 secondo per il linking dei file oggetto

Tempo di esecuzione sequenziale:
→ 7 secondi

Se si hanno 3 CPU a disposizione, in quanto tempo è possibile completare le stesse operazioni?

Esempio di sincronizzazione



Speed-up

Lo **speed-up** è una metrica che indica il (mancato) "guadagno" nel passare da un programma sequenziale a un programma parallelo a esso equivalente

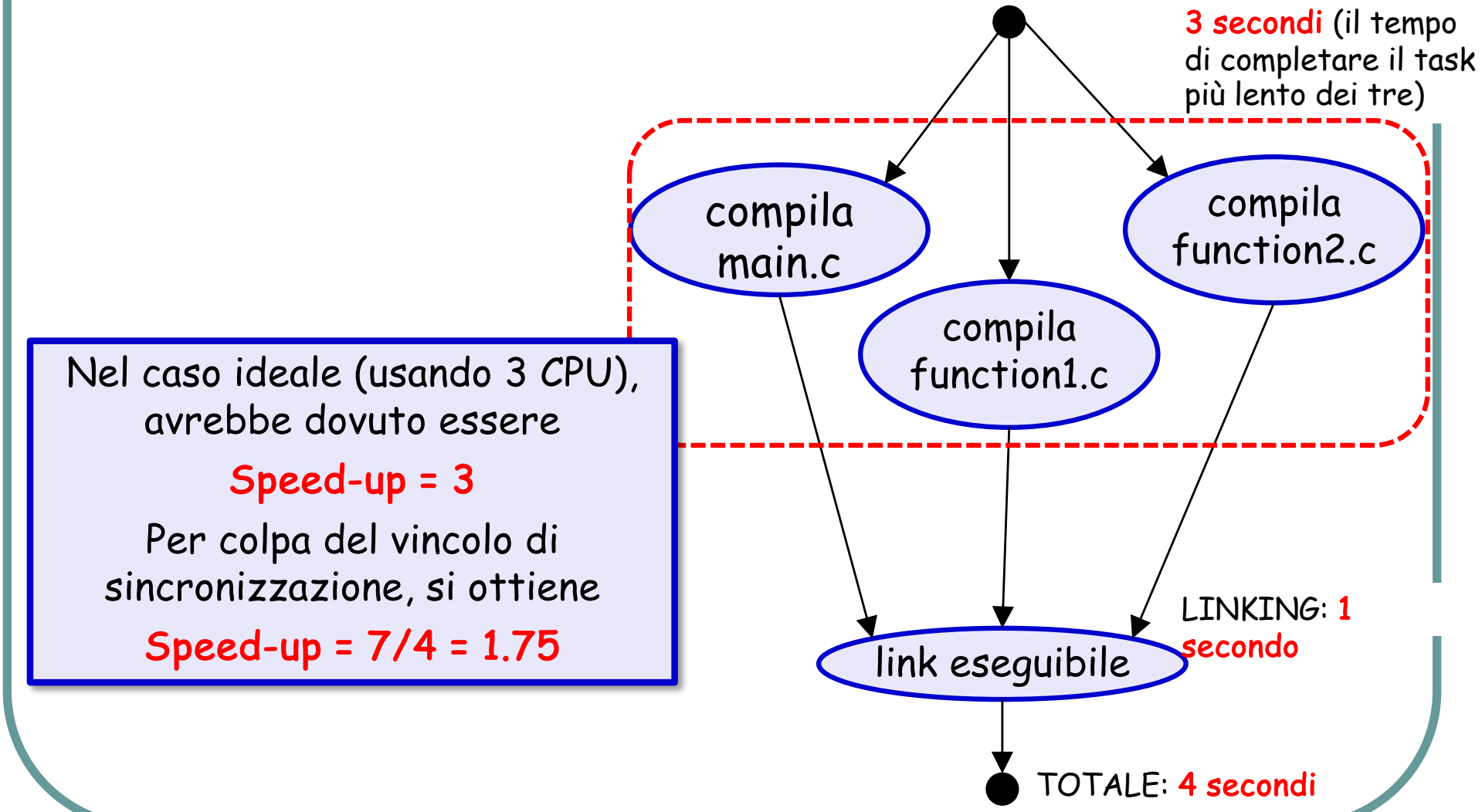
Speed-up:

$$S = \frac{T_s}{T_p}$$

T_S: tempo di **esecuzione sequenziale** del processo
T_P: tempo di **esecuzione parallela**

Idealmente, **S = N = numero di processi/CPU** nel sistema
...ma non sempre si verifica!

Speed-up - esempio



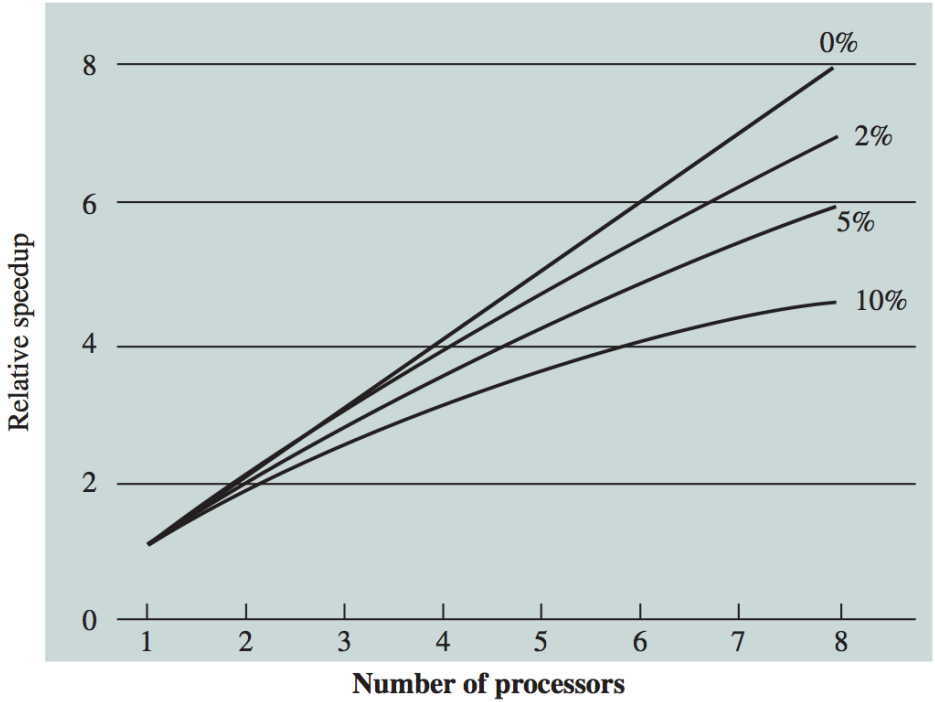
Nel caso ideale (usando 3 CPU), avrebbe dovuto essere

Speed-up = 3

Per colpa del vincolo di sincronizzazione, si ottiene

Speed-up = 7/4 = 1.75

Speed-up



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions

Lo speed-up è limitato dalla **quantità di operazioni sequenziali (non-parallelizzabili)** nel programma

Dipende dalle caratteristiche del problema da risolvere!

Legge di Amdahl

$$S = \frac{N}{1 + (N - 1) \cdot f}$$

- Maggiore la **percentuale "f" di codice sequenziale** (non-parallelizzabile), minore lo speed-up
- Nel caso **f=0** (tutto parallelizzabile), **S = N**
- Nel caso di **f=1** (tutto sequenziale), **S = 1** (lo speed-up si riduce a 1!)

Vincoli di sincronizzazione

Problemi di **competizione**:
si impone che un solo processo alla volta acceda alla risorsa comune

Problemi di **cooperazione**:
si impone un ordinamento (parziale) tra le operazioni dei processi

Modelli di programmazione concorrente

- Modello ad ambiente globale
- Modello ad ambiente locale

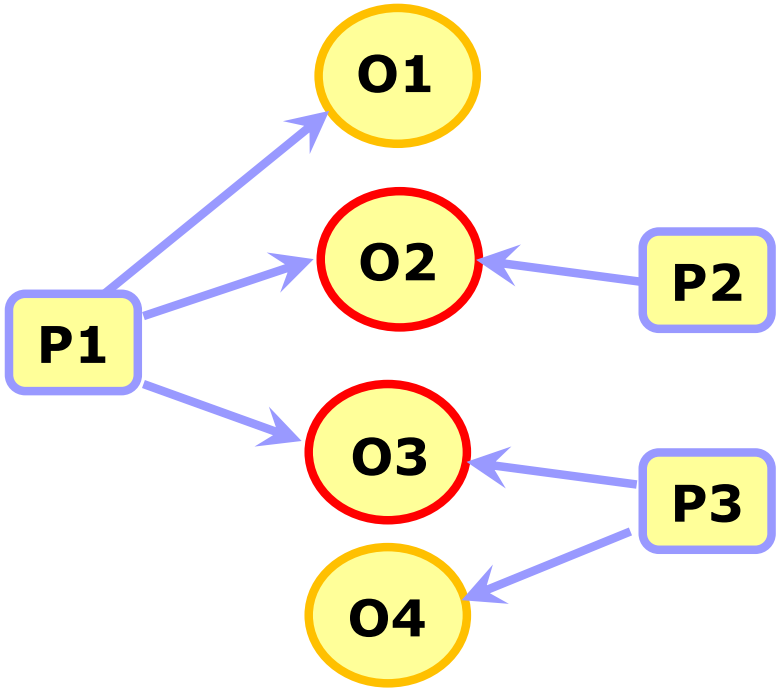
Le risorse

- Dicesi **risorsa** un qualunque **oggetto**, **fisico** o **logico**, di cui un processo necessita per le sue elaborazioni

- Le risorse possono essere
  - **private (o locali)**: visibili da un solo processo
  - **comuni (o globali)**: condivise fra più processi

Modello ad ambiente globale

- L'interazione avviene mediante **risorse comuni**

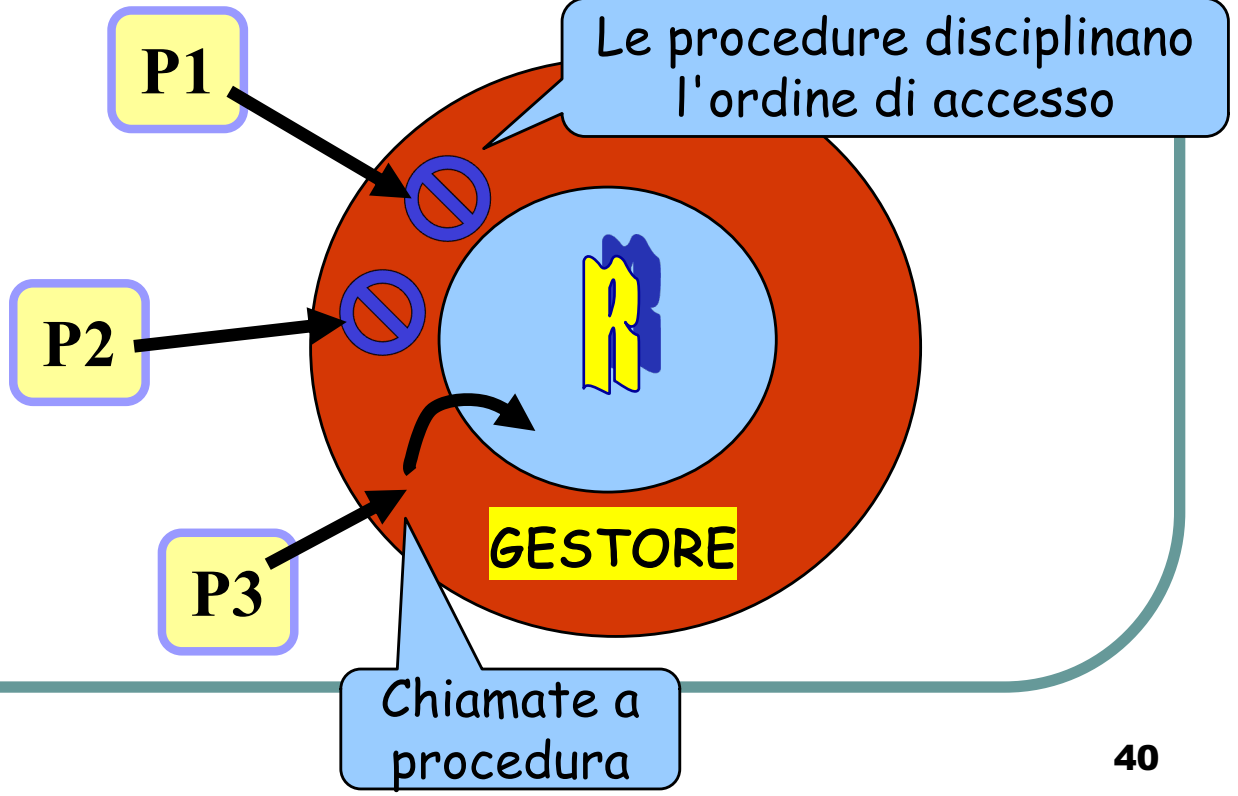


O1, O4 risorse private

O2, O3 risorse comuni

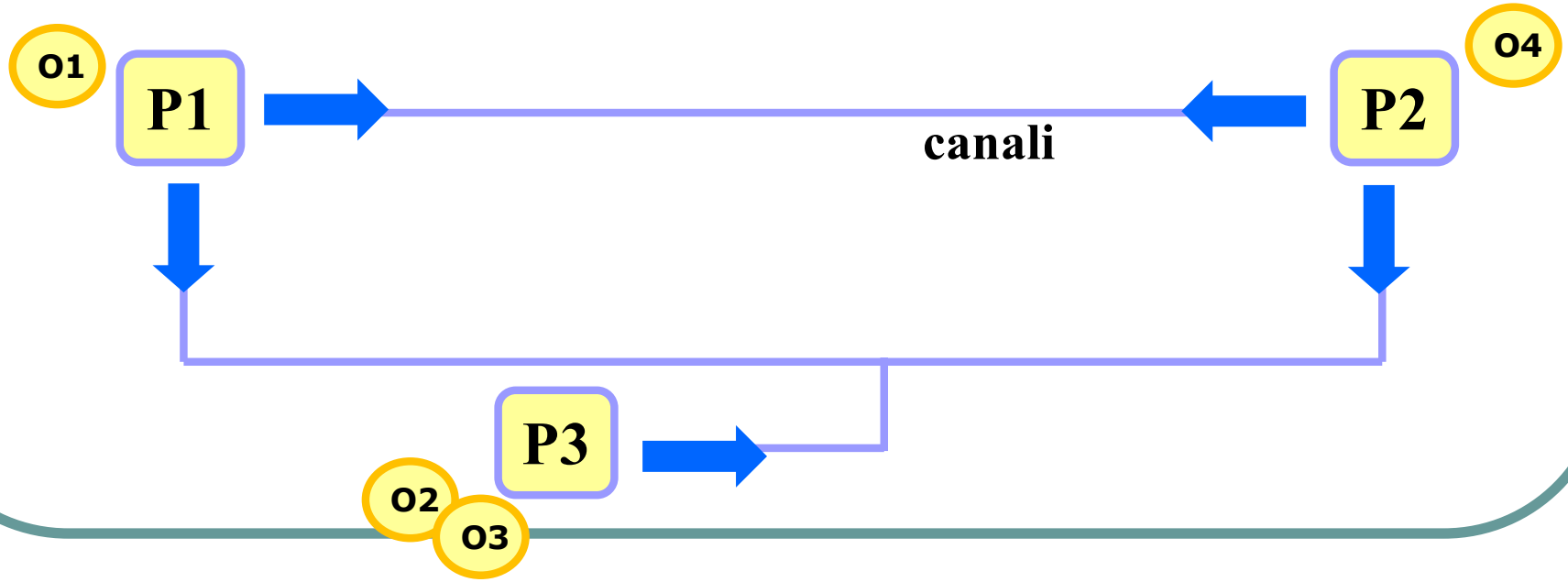
Risorsa Gestore (modello ad ambiente globale)

- Per garantire i vincoli di sincronizzazione, ci si avvale della intermediazione di un **gestore**:
  - insieme di **procedure**, e relative **strutture dati**, che operano sulla risorsa
  - il gestore è esso stesso condiviso tra tutti i processi



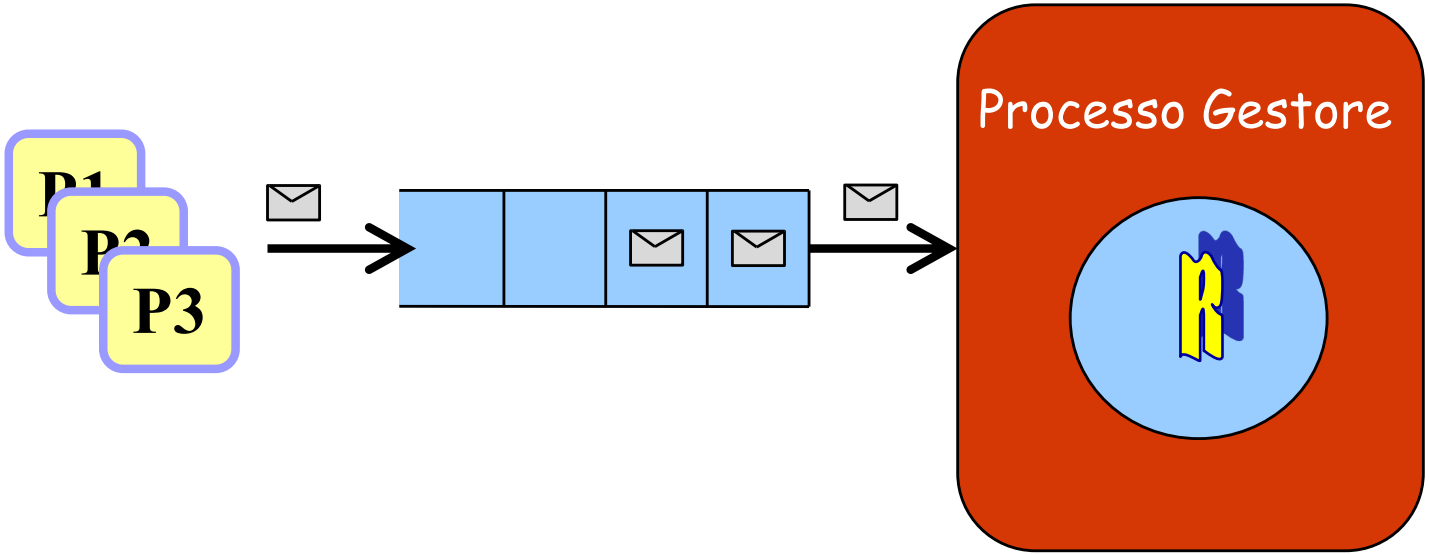
Modello ad ambiente locale

- Le risorse sono private ai processi
- Gli altri processi accedono alla risorsa interagendo con il processo proprietario
- Le interazioni avvengono tramite scambio di messaggi



Processo gestore (modello ad ambiente locale)

- In questo caso, la risorsa è acceduta dai processi per via di un **processo gestore**
- Un processo che intende usare la risorsa, deve **inviare un messaggio di richiesta** al processo gestore
- Il processo gestore esegue l'operazione richiesta



Strumenti di sincronizzazione

- Modello ad ambiente globale:
  - **semafori** e primitive **wait** e **signal**
- Modello ad ambiente locale:
  - **messaggi** e primitive **send** e **receive**

Quiz

1. Se un programma concorrente ha una "**race condition**", il risultato finale è **indifferente** all'ordine e dalla velocità di esecuzione delle singole operazioni

   - ○ Vero
   - ○ Falso

https://forms.office.com/r/DJ8UZZBFx6