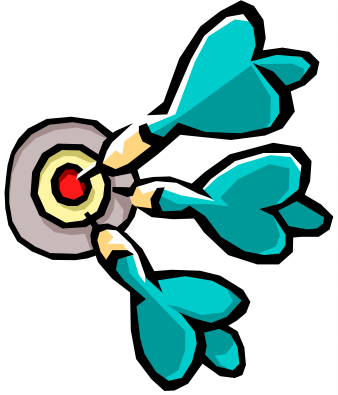
Interprocess communication

Programmazione di Sistema

A.A. 2017-18

Argomenti



- Concorrenza e processi
- Comunicazione tra processi

Concorrenza e processi

- L'uso dei thread permette di sfruttare le risorse computazionali presenti in un elaboratore
  - La presenza di uno spazio di indirizzamento condiviso facilita la coordinazione e la comunicazione
- Ci sono situazioni in cui la presenza di un singolo spazio di indirizzamento non è possibile o desiderabile
  - Riuso di programmi esistenti
  - Scalabilità su più computer
  - Sicurezza

Concorrenza e processi

- È possibile decomporre un sistema complesso in un insieme di processi collegati
  - Creandoli a partire da un processo genitore
  - Permettendo la cooperazione indipendentemente dalla loro genesi
- Ad ogni processo è associato almeno un thread (primary thread)
  - Un sistema multiprocesso è intrinsecamente concorrente
  - Solleva gli stessi problemi di interferenza e necessità di coordinamento

Processi in Windows

- Costituiscono entità separate, senza relazioni di dipendenza esplicita tra loro
- La funzione CreateProcess(…)
  - Crea un nuovo spazio di indirizzamento
  - Lo inizializza con l'immagine di un eseguibile
  - Attiva il thread primario al suo interno
- Il processo figlio può condividere variabili d'ambiente ed handle a file, semafori, pipe …
  - … ma non può condividere handle a thread, processi, librerie dinamiche e regioni di memoria

Processi in Linux

- Si crea un processo figlio con l'operazione fork()
  - Crea un nuovo spazio di indirizzamento «identico» a quello del processo genitore
  - I due processi condividono i riferimenti alle stesse pagine di memoria fisica
- Dopo l'esecuzione di fork(), tutte le pagine sono marcate con il flag «CopyOnWrite»
  - Eventuali scritture comportano la duplicazione della pagina e la separazione tra i due spazi di indirizzamento

Creazione di processi

- La funzione exec*() sostituisce l'attuale immagine di memoria dello spazio di indirizzamento
  - Ri-inizializzandola a quella descritta dall'eseguibile indicato come parametro

Esempio

```
int main ( const int argc, const char* const
argv[] ) {
    pid_t  ret = fork();
    switch (ret) {
       case -1:
          puts( "parent: error: fork
failed!" );break;
       case  0:
          puts( "child: here (before execl)!" );
          if (execl( "./ch.exe", "./ch.exe",
0 )==-1)
              perror( "child: execl failed:" );
          puts( "child: here (after execl)!" );
          //non si dovrebbe arrivare qui
          break;
        default:
              printf( "par: child pid=%d \n",
ret );
              break;
    }
  return 0;
```

Fork() e thread

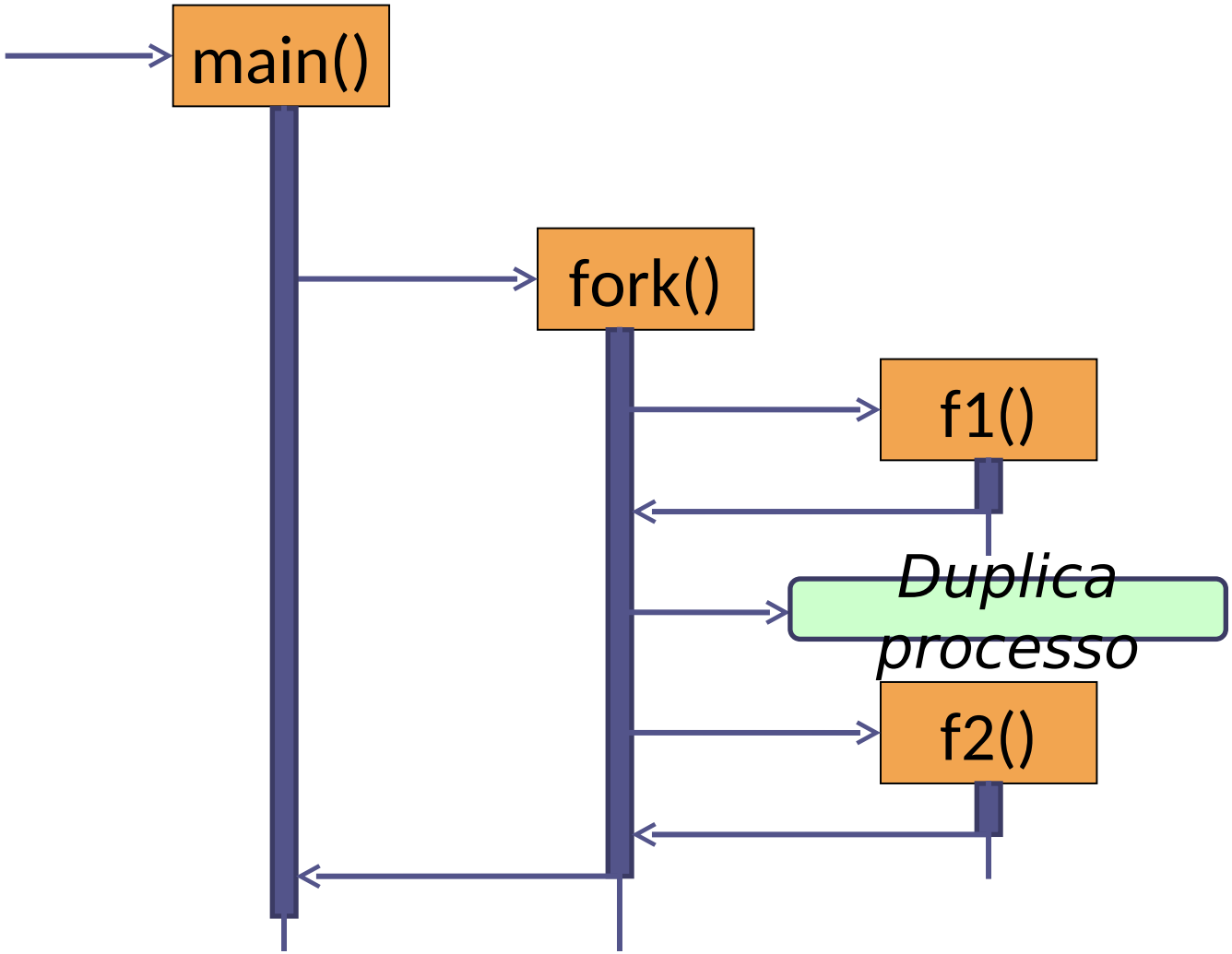
- Nel caso di programmi concorrenti, l'esecuzione di fork() crea un problema
  - Il processo figlio conterrà un solo thread
  - Gli oggetti di sincronizzazione presenti nel padre possono trovarsi in stati incongruenti
- int pthread_atfork(
      void (*prepare)(void),
      void (*parent)(void),
      void (*child)(void));
  - Registra un gruppo di funzioni che saranno chiamate in corrispondenza delle invocazioni a fork()

Esempio

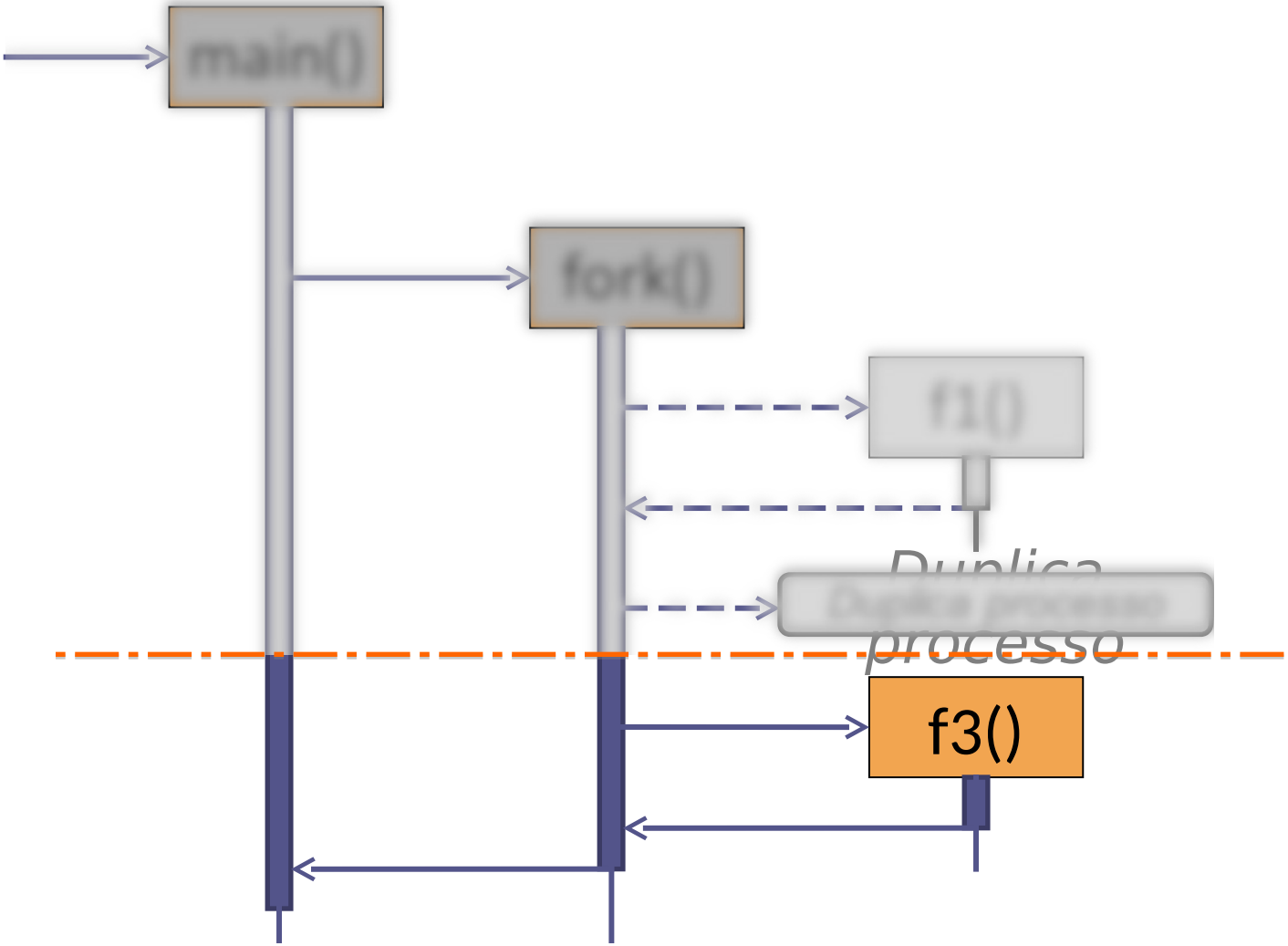
```
void f1() { … }
void f2() { … }
void f3() { … }

int main() {
   pthread_atfork(f1,f2,f3);
   //…
   int res= fork();
   if (res== -1 ) { /* errore */  }
   else if (res == 0) {      /* child */ }
   else {     /* parent */ }
}
```


Processo genitore



Processo figlio



IPC - InterProcess Communication

- Il S.O. impedisce il trasferimento diretto di dati tra processi
  - Ogni processo dispone di uno spazio di indirizzamento separato
  - Non è possibile sapere cosa sta capitando in un altro processo
- Ogni S.O. offre alcuni meccanismi per superare tale barriera in modo controllato
  - Permettendo lo scambio di dati…
  - …e la sincronizzazione delle attività

Rappresentazione delle informazioni scambiate

- Indipendentemente dal tipo di meccanismo adottato, occorre adattare le informazioni scambiate
  - Così da renderle comprensibili al destinatario

Rappresentazione interna

- Internamente, un processo può usare una varietà di rappresentazioni
  - Tipi elementari (numerici, logici, caratteri, …)
  - Tipi strutturati (record, array, classi, …)
  - Puntatori per strutture dati complesse (alberi, grafi, …)
- La rappresentazione interna non è adatta ad essere esportata
  - I puntatori non hanno senso al di fuori del proprio spazio di indirizzamento
  - Alcune informazioni (handle) non sono esportabili

Rappresentazione esterna

- Formato intermedio che permette la rappresentazione di strutture dati arbitrarie
  - Sostituendo i puntatori con riferimenti indipendenti dalla memoria
- Formati basati su testo
  - XML, JSON, CSV, …
- Formati binari
  - XDR, HDF, …

Serializzazione

- Le rappresentazioni esterne possono essere trattate come blocchi compatti di byte
  - Possono essere duplicati e trasferiti senza comprometterne il significato
- I dati vengono scambiati nel formato esterno
  - La sorgente esporta le proprie informazioni (marshalling)
  - Il destinatario ricostruisce una rappresentazione su cui può operare direttamente (unmarshalling)
- Le operazioni di marshalling e unmarshalling possono essere codificate esplicitamente
  - O essere eseguite da codice generato automaticamente dall'ambiente di sviluppo

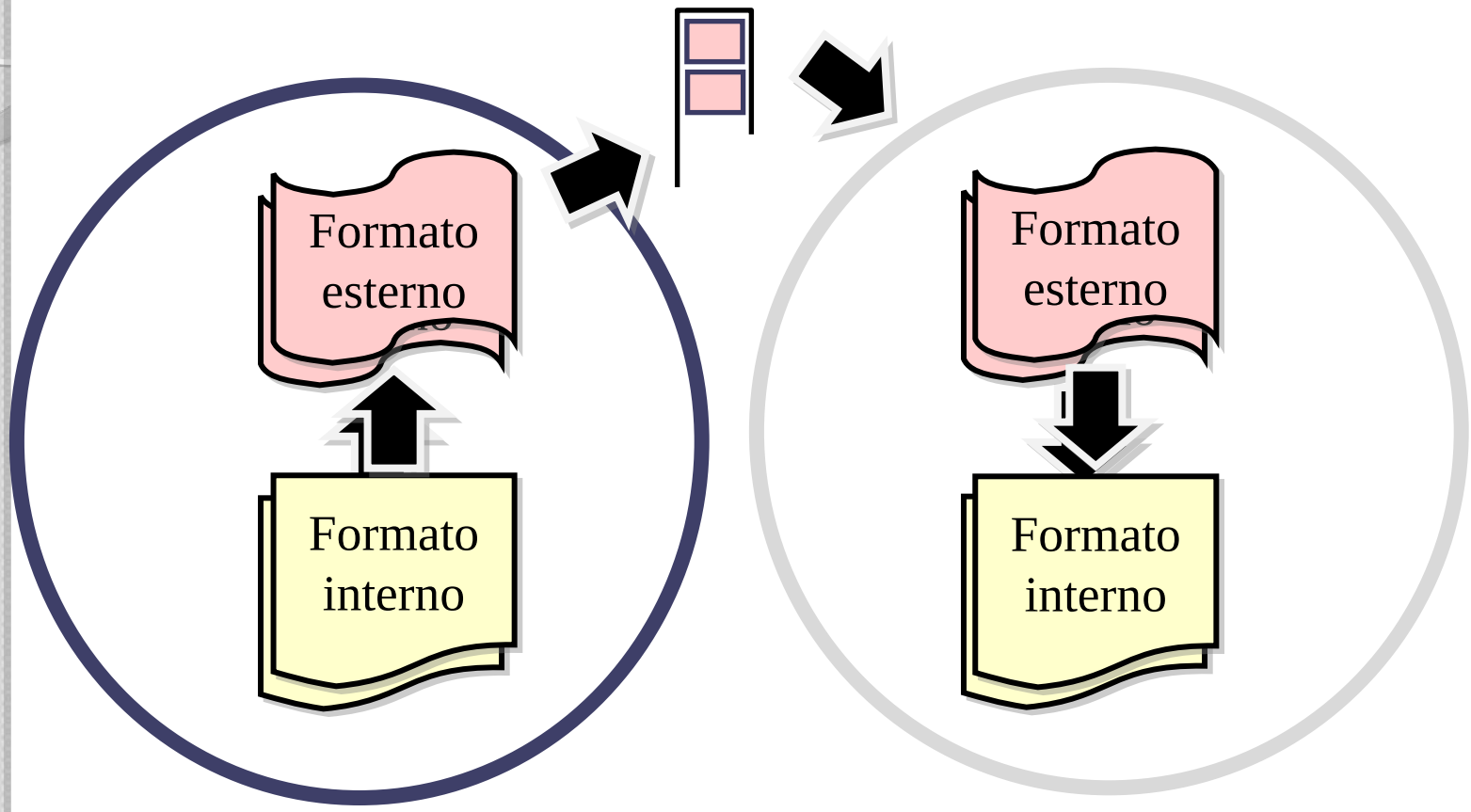
Tipi di IPC

- Code di messaggi
- Pipe
- Memoria condivisa
- Altro
  - File, socket, segnali, …

Code di messaggi

- Permettono il trasferimento atomico di blocchi di byte
  - Comportamento FIFO
  - Sono possibili più mittenti
  - L'inserimento di ciascun blocco sincronizza mittente e destinatario

Code di messaggi



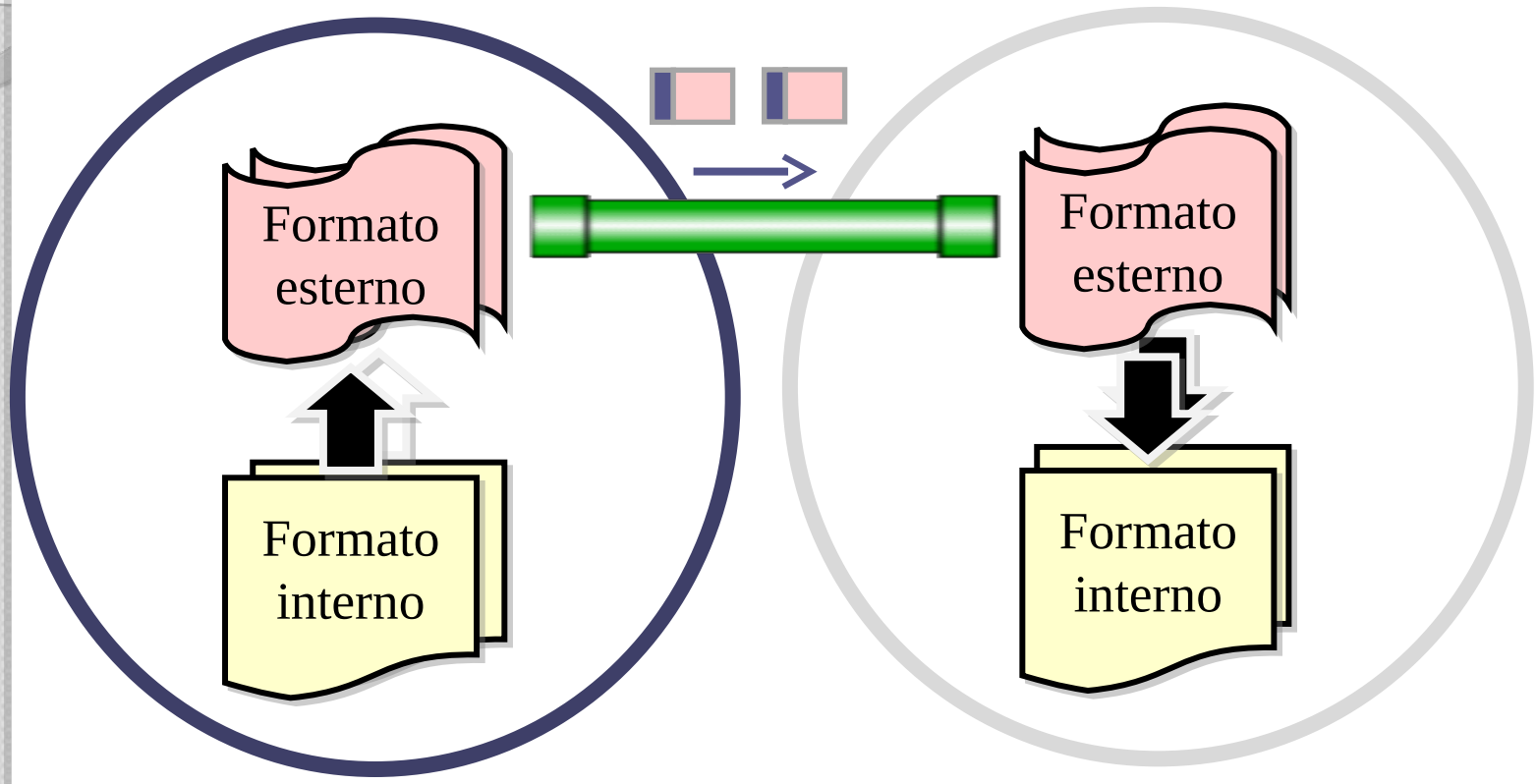
Spazio di indirizzamento 1

Spazio di indirizzamento 2

Pipe

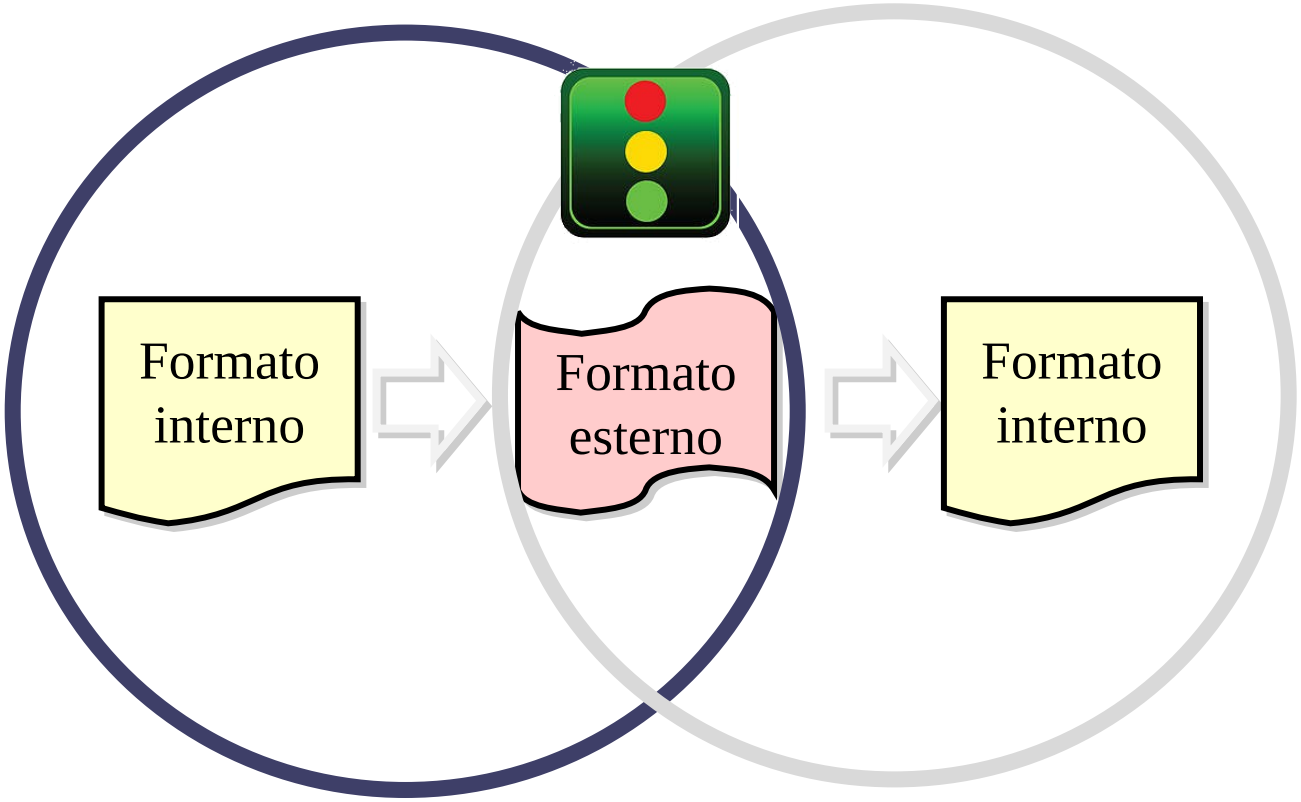
- «Tubi» che permettono il trasferimento di sequenze di byte di dimensioni arbitrarie
  - Occorre inserire marcatori che consentano di delimitare i singoli messaggi
  - Comunicazione sincrona 1-1

Pipe



Memoria condivisa

- Insieme di pagine fisiche mappate in due o più spazi di indirizzamento
  - Richiedono ulteriori meccanismi di sincronizzazione per evitare interferenze tra i thread dei processi coinvolti



Semafori

- Costrutti di sincronizzazione a basso livello
  - Basati sulla manipolazione atomica di un valore intero
  - Il valore è gestito dal S.O. e non può mai diventare negativo
- Due operazioni di base:
  - Up()
    - incrementa il valore
  - Down()
    - Se il valore è = O, si blocca in attesa di un incremento
    - Decrementa il valore

Semafori e sincronizzazione

- Un semaforo inizializzato ad 1 può essere visto come un mutex
  - Si proteggono le sezioni critiche di codice racchiudendole tra le operazioni down() e up()

Identità dei canali

- I canali di comunicazione sono creati dal S.O. su richiesta dei singoli processi
  - Ad ogni canale viene associato un identificativo univoco a livello di sistema
- I processi che cooperano devono accordarsi sul tipo e sull'identità del canale usato per la comunicazione /sincronizzazione

Fattori che influenzano la scelta del meccanismo

- Relazione tra i processi
  - Dipendenza esplicita
  - Nessuna dipendenza
- Tipo di comunicazione richiesto
  - Mono-/bidirezionale
- Numero di processi coinvolti

Spunti di riflessione

- Si realizzi un programma Windows che legga una stringa da linea di comando e la utilizzi come nome di un eseguibile da lanciare in un processo separato