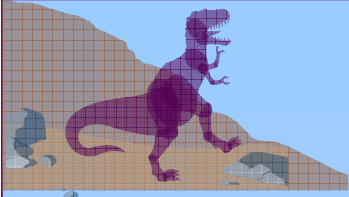


Capitolo 4: Processi

- Concetto di processo
- Scheduling dei processi
- Operazioni sui processi
- Processi cooperanti
- Comunicazione tra processi
- Comunicazione nei sistemi client-server

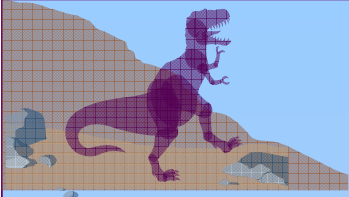




Concetto di processo

- Un sistema operativo esegue differenti programmi:
 - ☞ Un **sistema a lotti** (*batch system*) esegue **lavori** (*job*)
 - ☞ Un **sistema a partizione del tempo** (*time-shared system*) esegue **programmi utenti** o **task**
- Il vostro libro di testo utilizza i termini *lavoro* e *processo* in modo quasi intercambiabile.
- **Processo**: un programma in esecuzione; l'esecuzione del processo deve avvenire in modo sequenziale.
- Un processo comprende:
 - ☞ contatore di programma (*program counter*)
 - ☞ pila (*stack*)
 - ☞ sezione di dati (*data section*)



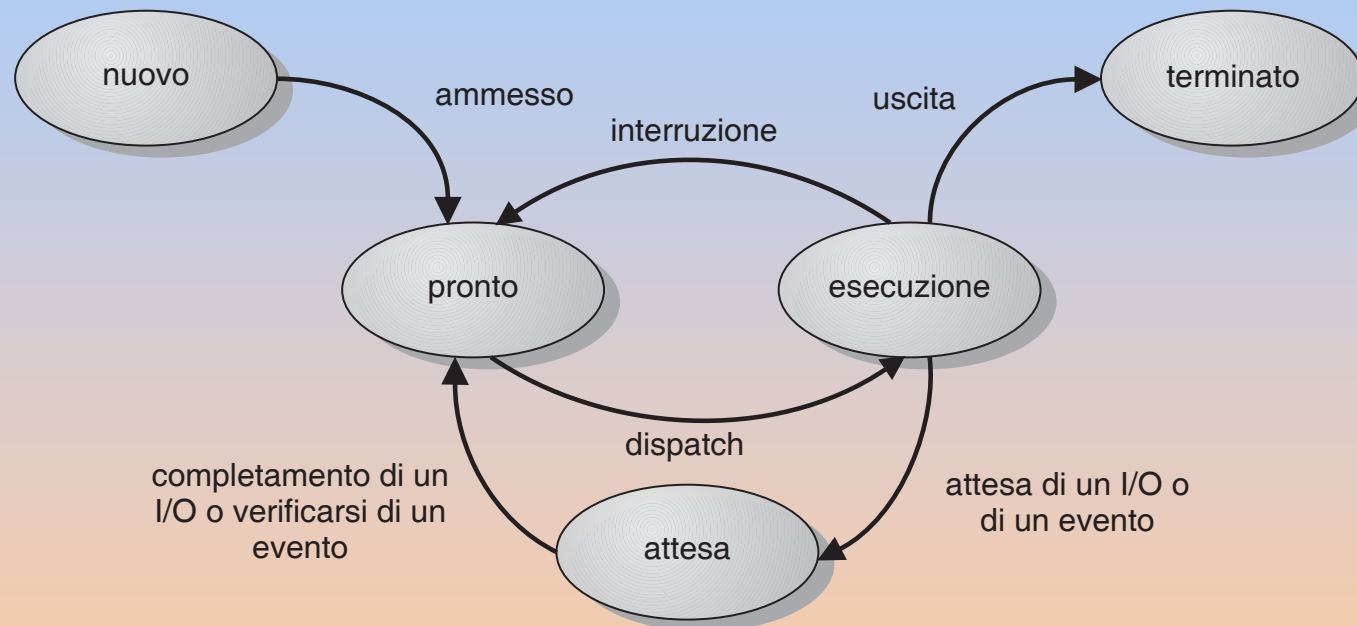


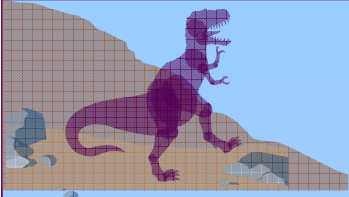
Stato del processo

- Mentre un processo è in esecuzione, è soggetto a cambiamenti di *stato*:
 - ☞ **nuovo**: si crea il processo.
 - ☞ **esecuzione**: le istruzioni vengono eseguite.
 - ☞ **attesa**: il processo attende che si verifichi qualche evento.
 - ☞ **pronto**: il processo attende di essere assegnato a un'unità d'elaborazione.
 - ☞ **terminato**: il processo ha terminato l'esecuzione.



Diagramma di transizione degli stati di un processo



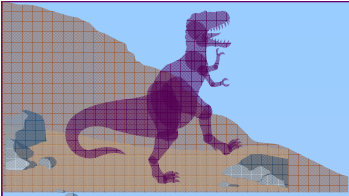


Blocco di controllo di un processo (PCB)

Informazioni connesse a un processo specifico.

- Stato del processo
- Contatore di programma
- Registri di CPU
- Informazioni sullo scheduling di CPU
- Informazioni sulla gestione della memoria
- Informazioni di contabilizzazione delle risorse
- Informazioni sullo stato dell'I/O

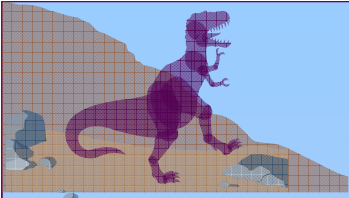




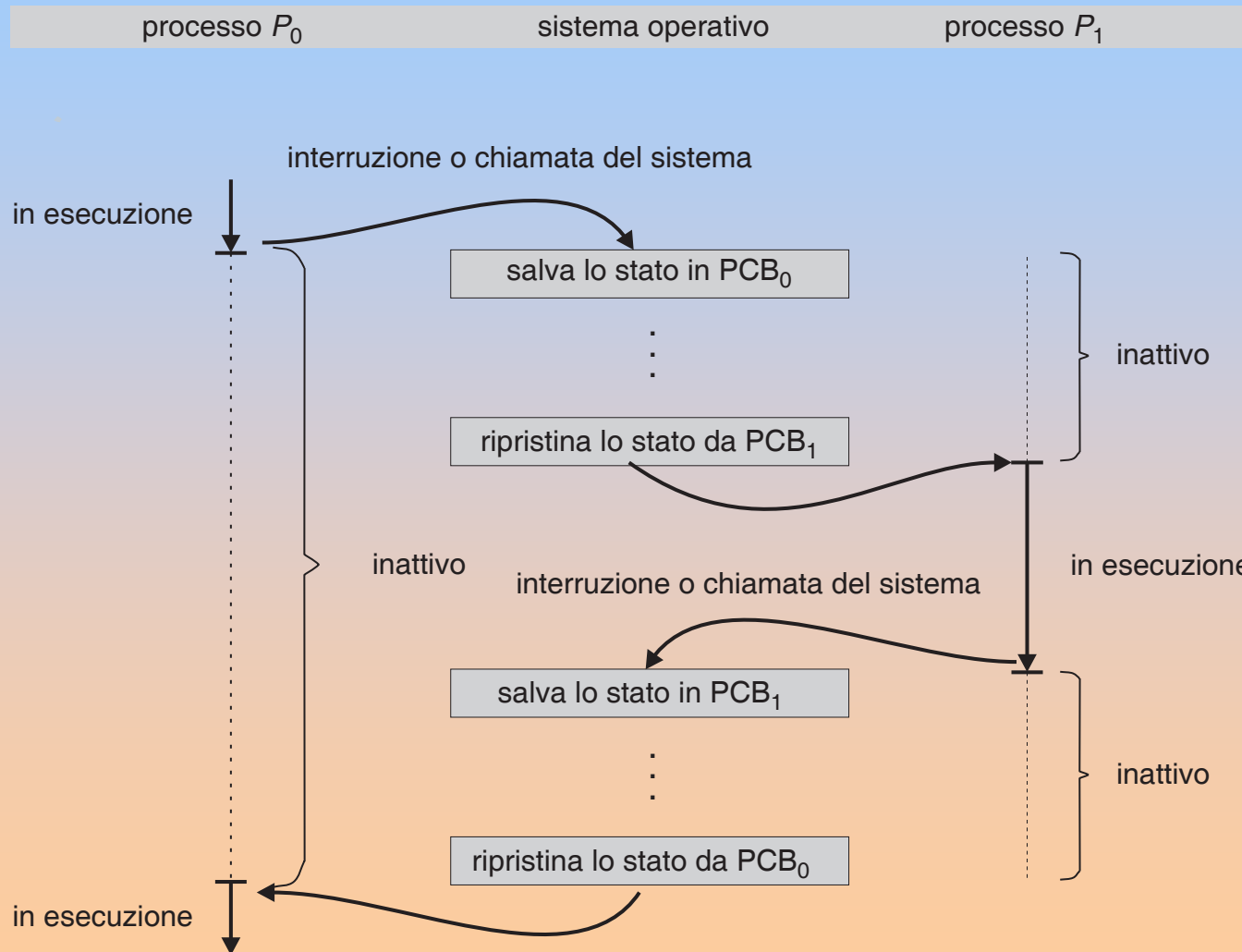
Process Control Block (PCB)

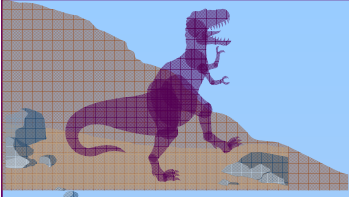
puntatore	stato del processo
numero del processo	
contatore di programma	
registri	
limiti di memoria	
elenco dei file aperti	
■ ■ ■	





La CPU può essere commutata tra i processi





Code di scheduling dei processi

- **Coda di processi** (*job queue*): insieme di tutti i processi del sistema.
- **Coda dei processi pronti** (*ready queue*): processi presenti nella memoria centrale, pronti e in attesa di essere eseguiti.
- **Coda del dispositivo** (*device queue*): elenco dei processi che attendono la disponibilità di un particolare dispositivo di I/O.



Coda dei processi pronti e diverse code di dispositivi I/O

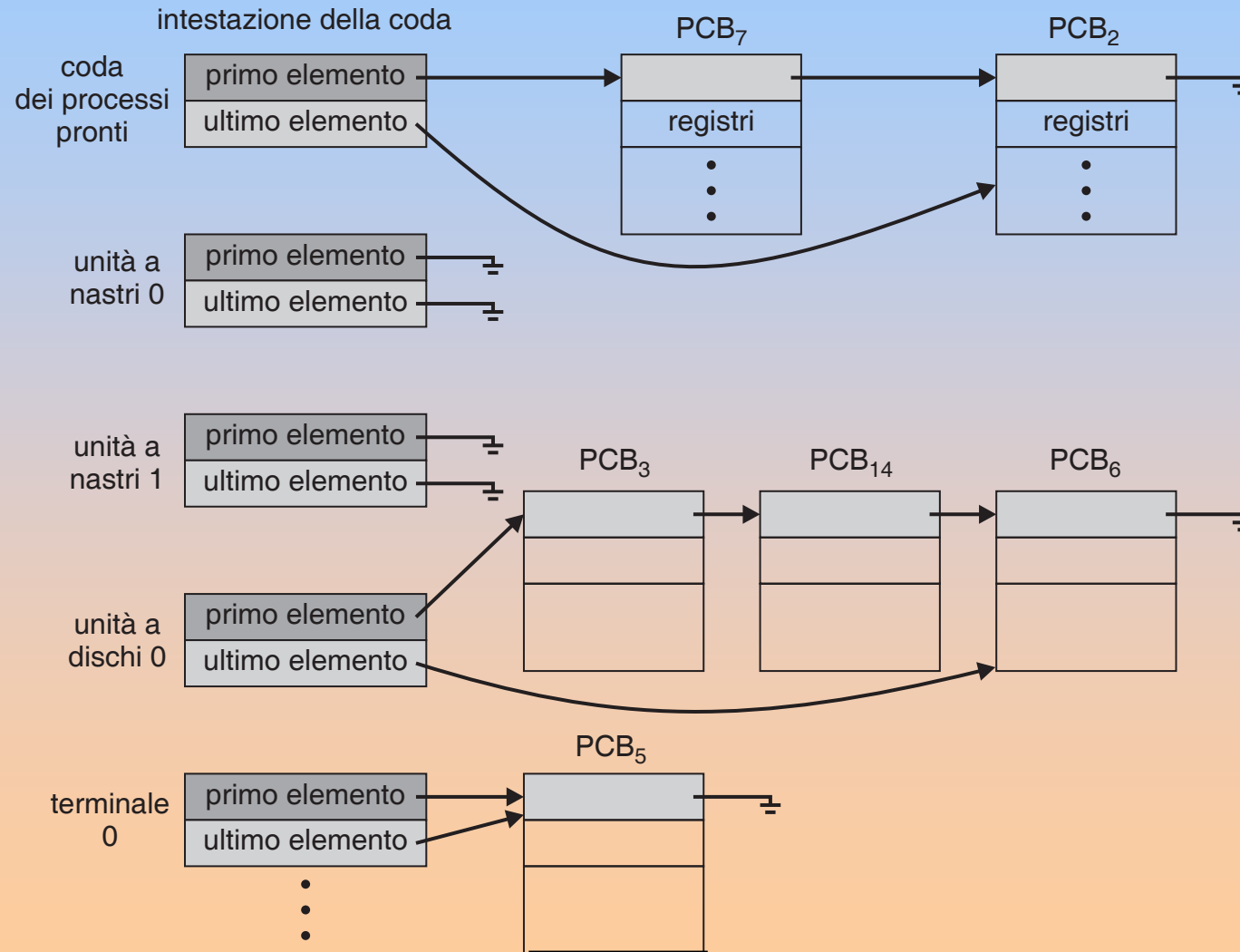
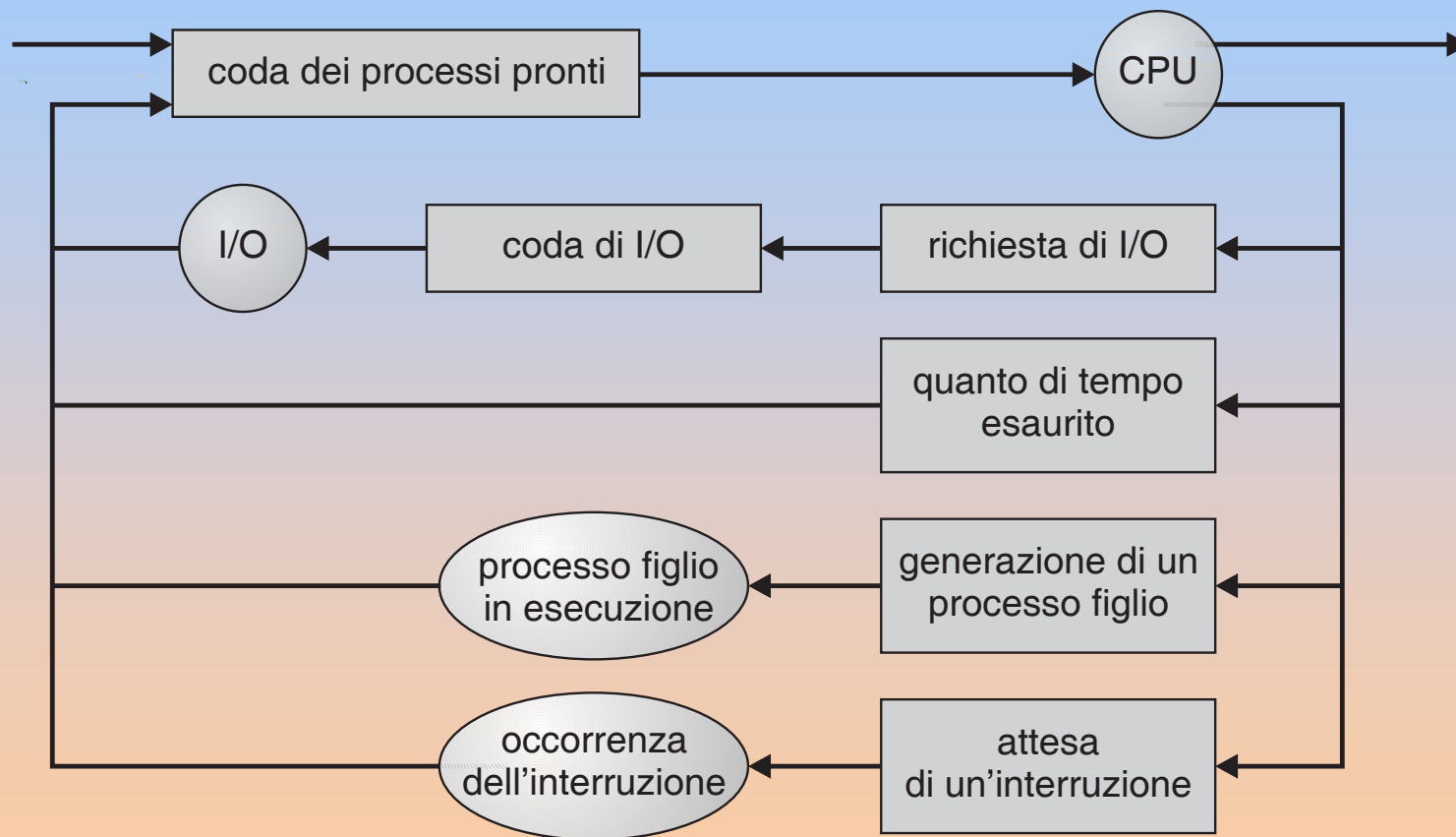
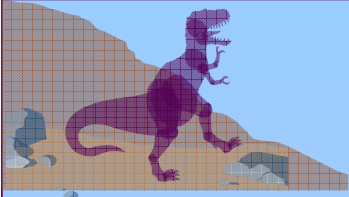


Diagramma di accodamento per lo scheduling dei processi

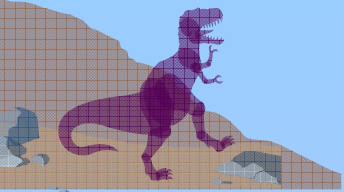




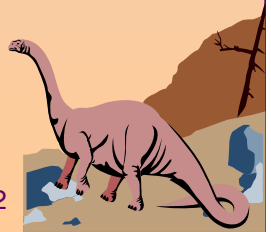
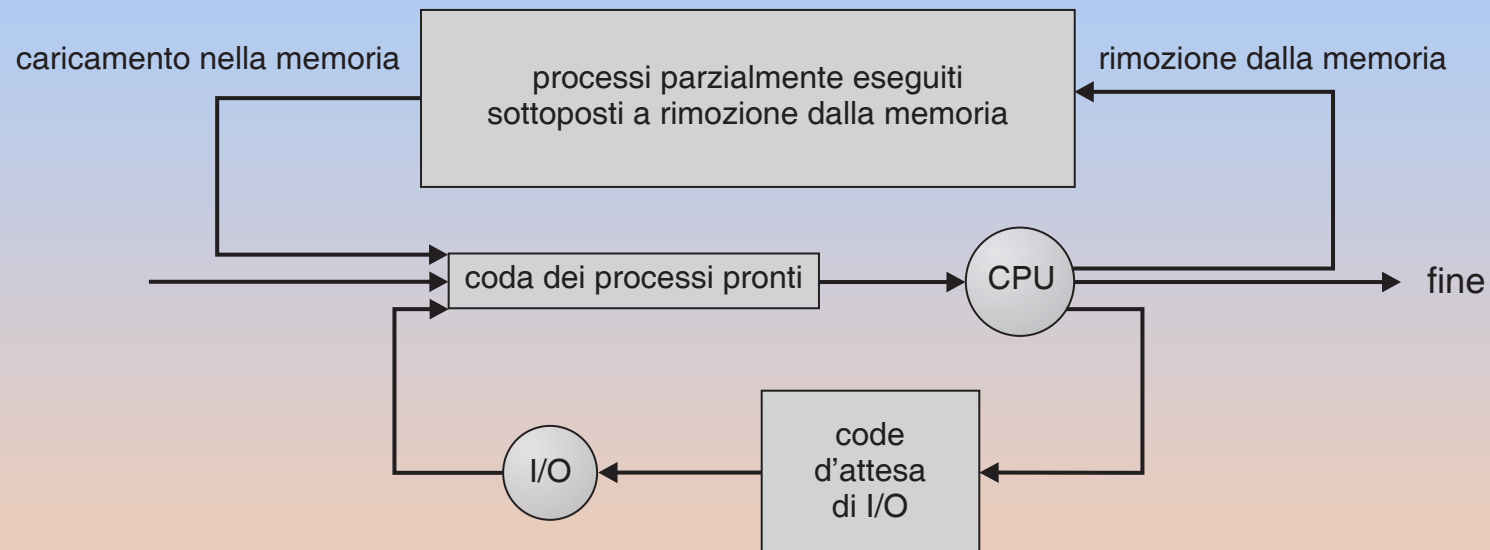
Scheduler

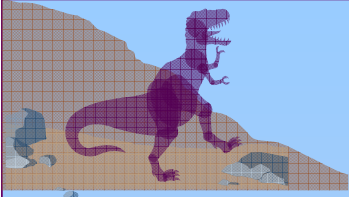
- Lo **scheduler a lungo termine** (o *job scheduler*) seleziona i processi che devono essere caricati nella coda dei processi pronti.
- Lo **scheduler di CPU** (*short-term scheduler* o *CPU scheduler*) fa la selezione tra i lavori pronti per essere eseguiti e assegna la CPU a uno di essi.





Aggiunta di scheduling a medio termine al diagramma di accodamento

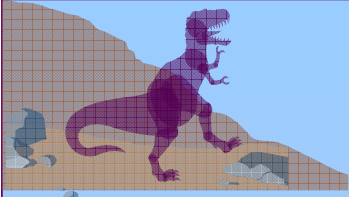




Scheduler (Cont.)

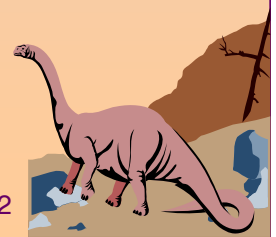
- Lo scheduler di CPU viene invocato frequentemente (millisecondi) \Rightarrow (deve essere veloce).
- Lo scheduler a lungo termine viene invocato meno frequentemente (secondi, minuti) \Rightarrow (può essere lento).
- Lo scheduler lungo termine controlla il **grado di multiprogrammazione**.
- I processi possono essere caratterizzati come:
 - ☞ **Processo con prevalenza di I/O** (*I/O-bound process*): impiega la maggior parte del proprio tempo nell'esecuzione di operazioni di I/O.
 - ☞ **Processo con prevalenza d'elaborazione** (*CPU-bound process*): richiede poche operazioni di I/O e impiega la maggior parte del proprio tempo nelle elaborazioni.

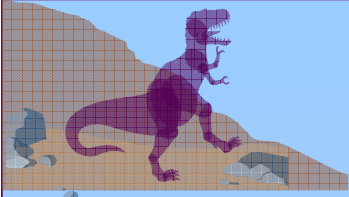




Cambio di contesto

- Passaggio della CPU a un nuovo processo, con conseguente registrazione dello stato del processo vecchio e il caricamento dello stato precedentemente registrato del nuovo processo.
- Il tempo necessario al cambio di contesto è puro sovraccarico: il sistema non compie alcun lavoro utile durante la commutazione.
- La durata del cambio di contesto dipende molto dall'architettura.

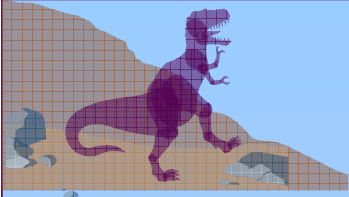




Creazione di un processo

- Un processo genitore può creare numerosi processi figli che, a loro volta, possono creare altri processi creando un *albero* di processi.
- Condivisione delle risorse
 - ➡ Genitore e figlio condividono tutte le risorse.
 - ➡ Il processo figlio condivide un sottoinsieme delle risorse del genitore.
 - ➡ Genitore e figlio non condividono alcuna risorsa.
- Esecuzione
 - ➡ Genitore e figlio possono essere eseguiti in modo concorrente.
 - ➡ Il processo genitore attende che alcuni o tutti i suoi processi figli terminino.





Creazione di un processo (Cont.)

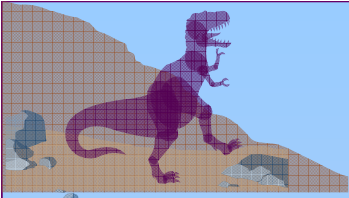
■ Spazio d'indirizzi

- ☞ Il processo figlio è un duplicato del processo genitore.
- ☞ Nel processo figlio si carica un programma.

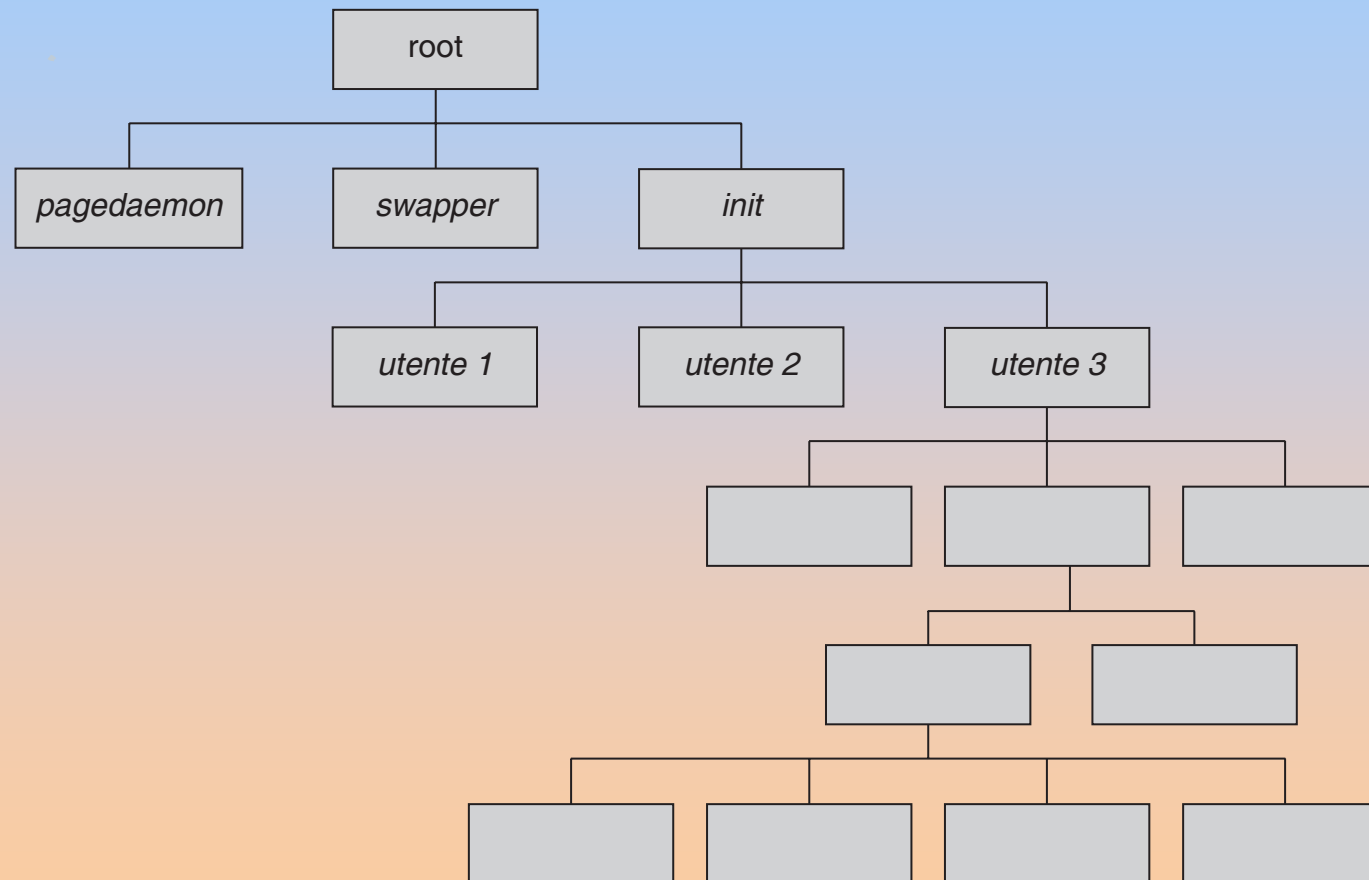
■ Esempi UNIX

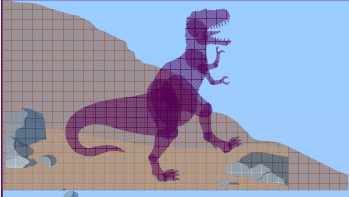
- ☞ la chiamata di sistema **fork** crea un nuovo processo
- ☞ la chiamata di sistema **exec** dopo una **fork** sostituisce lo spazio di memoria del processo con un nuovo programma.





Albero di processi in un tipico sistema UNIX

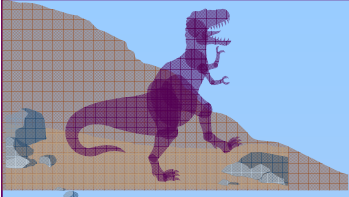




Terminazione di un processo

- Un processo termina quando finisce l'esecuzione della sua ultima istruzione e chiede al sistema operativo di essere cancellato usando la chiamata del sistema **exit**.
 - ☞ Il processo figlio può riportare alcuni dati al processo genitore attraverso la chiamata del sistema **wait**.
 - ☞ Tutte le risorse del processo sono liberate dal sistema operativo.
- Un processo genitore può porre termine all'esecuzione di uno dei suoi processi figli (**abort**) perché:
 - ☞ Il processo figlio ha ecceduto nell'uso di alcune risorse.
 - ☞ Il compito assegnato al processo figlio non è più richiesto.
 - ☞ Il processo genitore termina.
 - 📄 Il sistema operativo non consente a un processo figlio di continuare l'esecuzione in tale circostanza.
 - 📄 **Terminazione a cascata** (*cascading termination*).

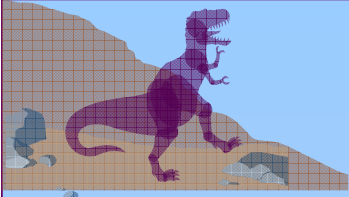




Processi cooperanti

- Un processo è **indipendente** se non può influire su altri processi nel sistema o subirne l'influsso.
- Un processo è **cooperante** se influenza o può essere influenzato da altri processi in esecuzione nel sistema.
- Vantaggi dei processi cooperanti
 - ➡ Condivisione d'informazioni
 - ➡ Accelerazione del calcolo
 - ➡ Modularità
 - ➡ Convenienza

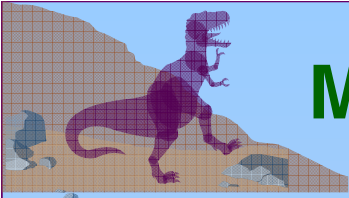




Il problema del produttore e del consumatore

- Usuale paradigma per i processi cooperanti: un processo **produttore** produce informazioni che sono consumate da un processo **consumatore**.
 - ☞ **memoria illimitata** (*unbounded-buffer*): non pone limiti alla dimensione del vettore.
 - ☞ **memoria limitata** (*bounded-buffer*): presuppone l'esistenza di una dimensione fissata del vettore.



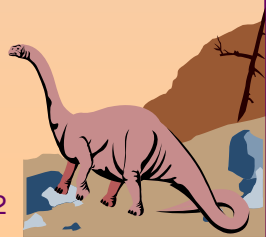


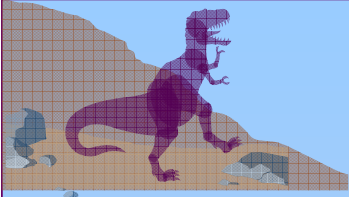
Memoria limitata – Soluzione con memoria condivisa

- Dati condivisi

```
#define DIM_VETTORE 10
typedef struct {
    ...
} elemento;
elemento vettore[DIM_VETTORE];
int inserisci = 0;
int preleva = 0;
```

- La soluzione è corretta ma permette di avere al massimo **DIM_VETTORE - 1** elementi contemporaneamente nel vettore.



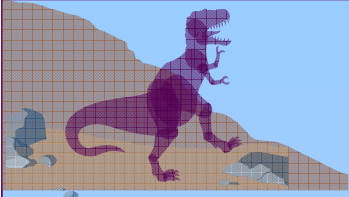


Memoria limitata – Processo produttore

```
item appena_Prodotto;

while (1) {
    while (((inserisci + 1) % DIM_VETTORE) == preleva)
        ; /* non fa niente */
    vettore[inserisci] = appena_Prodotto;
    inserisci = (inserisci + 1) % DIM_VETTORE;
}
```

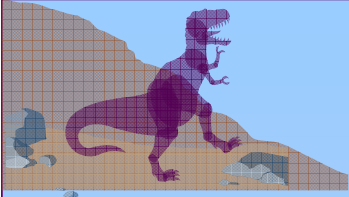




Memoria limitata – Processo consumatore

```
item da_Consumare;  
  
while (1) {  
    while (inserisci == preleva)  
        ; /* non fa niente */  
    da_Consumare = vettore[preleva];  
    preleva = (preleva + 1) % DIM_VETTORE;  
}
```

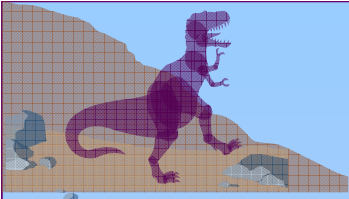




Comunicazione tra processi (IPC)

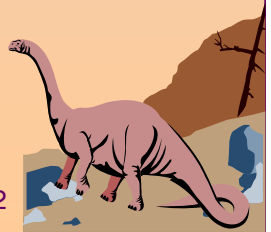
- Attraverso le funzioni di IPC i processi possono comunicare tra loro e sincronizzare le proprie azioni.
- Un sistema di scambio di messaggi permette ai processi di comunicare tra loro senza ricorrere a dati condivisi.
- Un sistema di IPC fornisce le due operazioni:
 - ☞ **send**(*messaggio*) : dimensione fissa o variabile
 - ☞ **receive**(*messaggio*)
- Se i processi *P* e *Q* vogliono comunicare, devono:
 - ☞ stabilire un **canale di comunicazione** tra loro
 - ☞ Scambiare messaggi mediante **send/receive**
- Realizzazione di un canale di comunicazione
 - ☞ fisica (es. memoria condivisa, bus, reti)
 - ☞ logica (es. proprietà logiche)

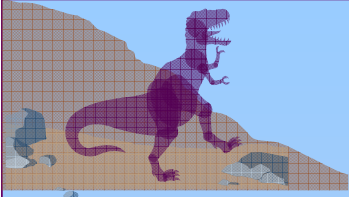




Questioni implementative

- Come sono stati creati i canali di comunicazione?
- Può un canale essere associato a più di due processi?
- Quanti canali possono esserci tra ogni coppia di processi comunicanti?
- Qual è la capacità di un canale di comunicazione?
- La dimensione di un messaggio deve essere fissa o variabile?
- Il canale di comunicazione è unidirezionale o bidirezionale?





Comunicazione diretta

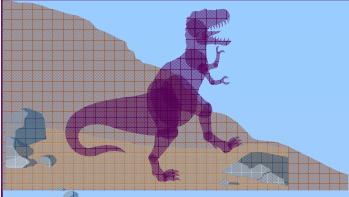
- I processi devono nominarsi reciprocamente in modo esplicito:

- ✎ **send**(P , *messaggio*) : invia messaggio al processo P
- ✎ **receive**(Q , *messaggio*) : riceve, in messaggio, un messaggio dal processo Q

- Caratteristiche di un canale di comunicazione:

- ✎ I canali vengono stabiliti automaticamente.
- ✎ Un canale è associato esattamente a una coppia di processi.
- ✎ Esiste esattamente un canale tra ciascuna coppia di processi.
- ✎ Il canale può essere unidirezionale, ma è in genere bidirezionale.

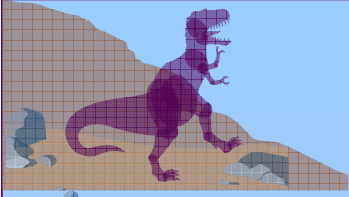




Comunicazione indiretta

- I messaggi vengono inviati a delle **porte** (dette anche *mailbox*) che li ricevono
 - ☞ Ciascuna porta è identificata in modo univoco.
 - ☞ Due processi possono comunicare solo se condividono una porta.
- Caratteristiche di un canale di comunicazione:
 - ☞ Tra una coppia di processi si stabilisce un canale solo se entrambi i processi condividono una stessa porta
 - ☞ Un canale può essere associato a più di due processi.
 - ☞ Ciascuna coppia di processi può condividere più canali di comunicazione.
 - ☞ Il canale può essere unidirezionale o bidirezionale.





Comunicazione indiretta

■ Operazioni:

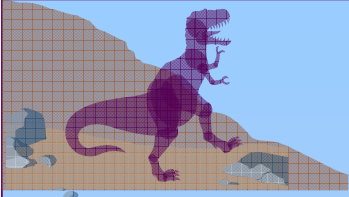
- ✎ creare una nuova porta
- ✎ inviare e ricevere messaggi tramite la porta
- ✎ rimuovere una porta

■ Primitive:

send(A , *messaggio*) : invia messaggio alla porta A

receive(A , *messaggio*) : riceve, in messaggio, un messaggio dalla porta A





Comunicazione indiretta

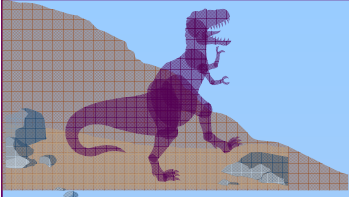
■ Condivisione della porta:

- ✎ P_1 , P_2 , e P_3 condividono la porta A.
- ✎ P_1 invia un messaggio ad A; P_2 e P_3 eseguono una `receive` da A.
- ✎ Quale processo riceverà il messaggio?

■ Soluzioni:

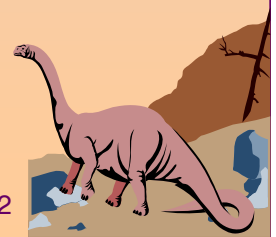
- ✎ consentire che un canale sia associato al massimo a due processi
- ✎ consentire a un solo processo alla volta di eseguire un'operazione `receive`
- ✎ Consentire al sistema di decidere arbitrariamente quale processo riceverà il messaggio. Il sistema può comunicare l'identità del ricevente al trasmittente.

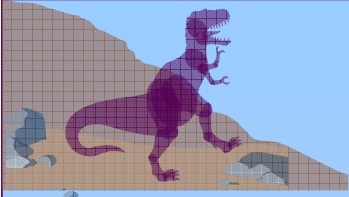




Sincronizzazione

- Lo scambio di messaggi può essere **sincrono** (**bloccante**) oppure **asincrono** (**non bloccante**).
- Anche le primitive **send** and **receive** possono essere sincrone oppure asincrone.

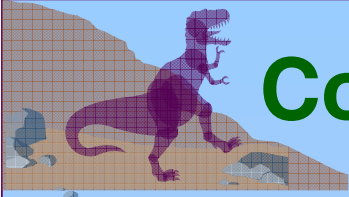




Code di messaggi (buffering)

- Se la comunicazione è diretta o indiretta, i messaggi scambiati tra processi comunicanti risiedono in code temporanee. Fondamentalmente esistono tre modi per realizzare queste code:
 1. **Capacità zero:** 0 messaggi
Il trasmittente deve fermarsi finché il ricevente prende in consegna il messaggio (*rendezvous*).
 2. **Capacità limitata:** la coda ha lunghezza finita n
Il trasmittente deve attendere se il canale è pieno.
 3. **Capacità imitata:** la coda ha lunghezza potenzialmente infinita
Il trasmittente non si ferma mai.

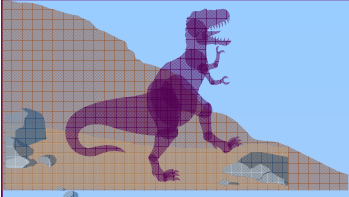




Comunicazione nei sistemi client-server

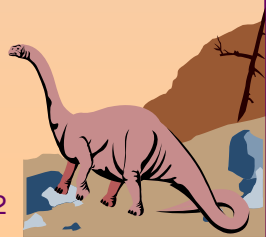
- Socket
- Chiamate di procedure remote, RPC
- Invocazione di metodi remoti, RMI (Java)



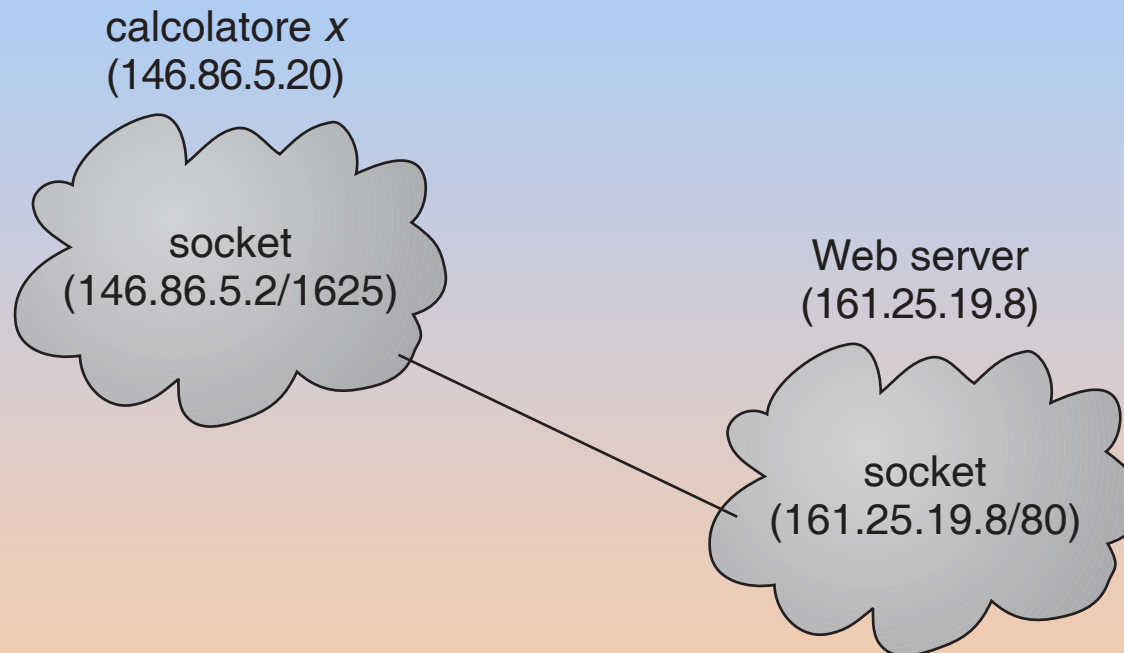


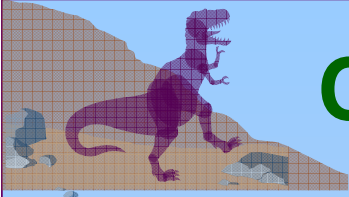
Sockets

- Una socket è definita come *l'estremità di un canale di comunicazione*.
- Ogni socket è identificata da un indirizzo IP concatenato a un numero di porta.
- La socket **161.25.19.8:1625** si riferisce alla porta **1625** sul calcolatore **161.25.19.8**
- La comunicazione avviene attraverso una coppia di socket.



Comunicazione tramite socket



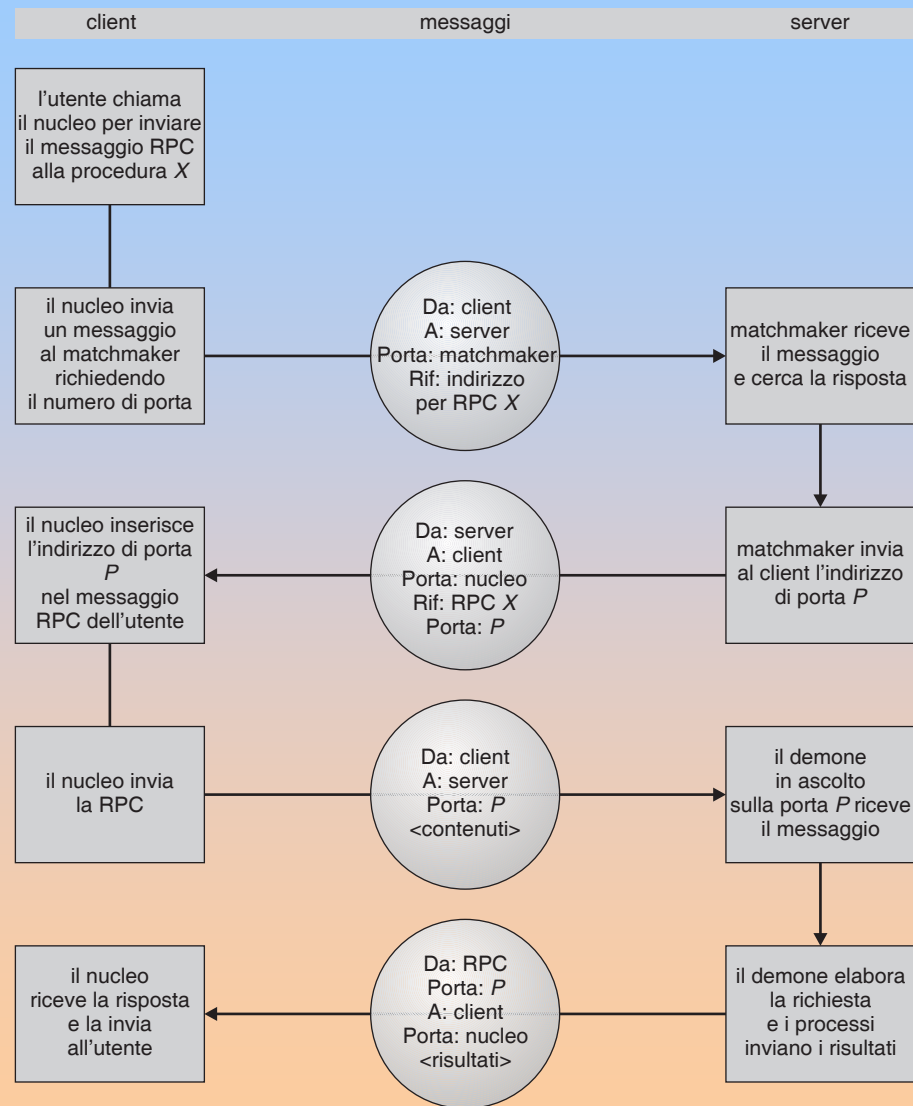


Chiamate di procedure remote (RPC)

- La RPC è stata progettata per astrarre il meccanismo della chiamata di procedura affinché si possa usare tra sistemi collegati tramite una rete.
- **Stub**: segmento di codice: proxy lato client per la procedura in corso sul server.
- Il segmento di codice di riferimento individua la porta del server e struttura i parametri (*marshalling*).
- Un analogo segmento di codice di riferimento nel server riceve questo messaggio e invoca la procedura nel server.



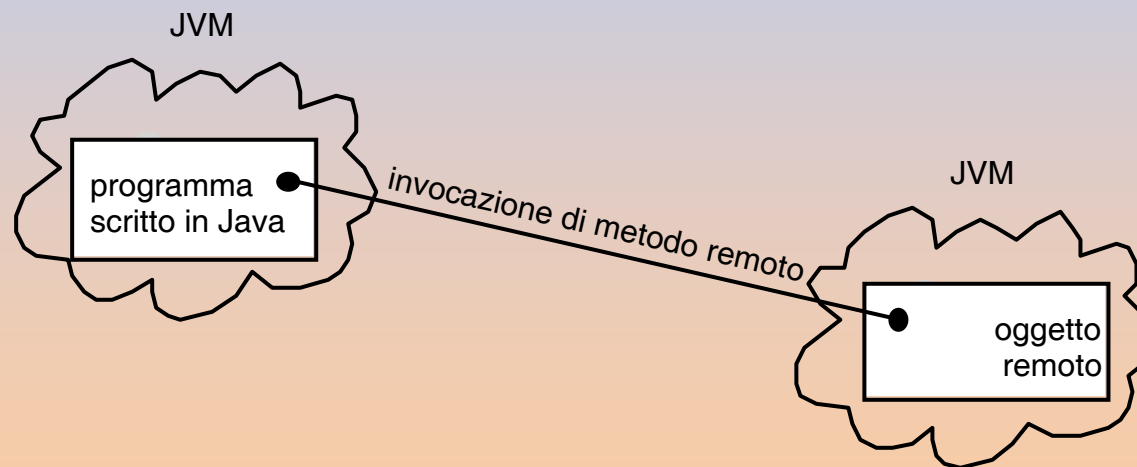
Esecuzione di una RPC





Invocazione di metodi remoti

- L'invocazione di metodi remoti (RMI, remote method invocation) è una funzione del linguaggio Java simile alla RPC.
- Una RMI consente a un thread di invocare un metodo di un oggetto remoto.



Strutturazione dei parametri

