

# I processi

- Concetto di processo
- Scheduling dei processi
- Operazioni sui processi
- Processi cooperanti
- Comunicazione fra processi

# Il modello a processi

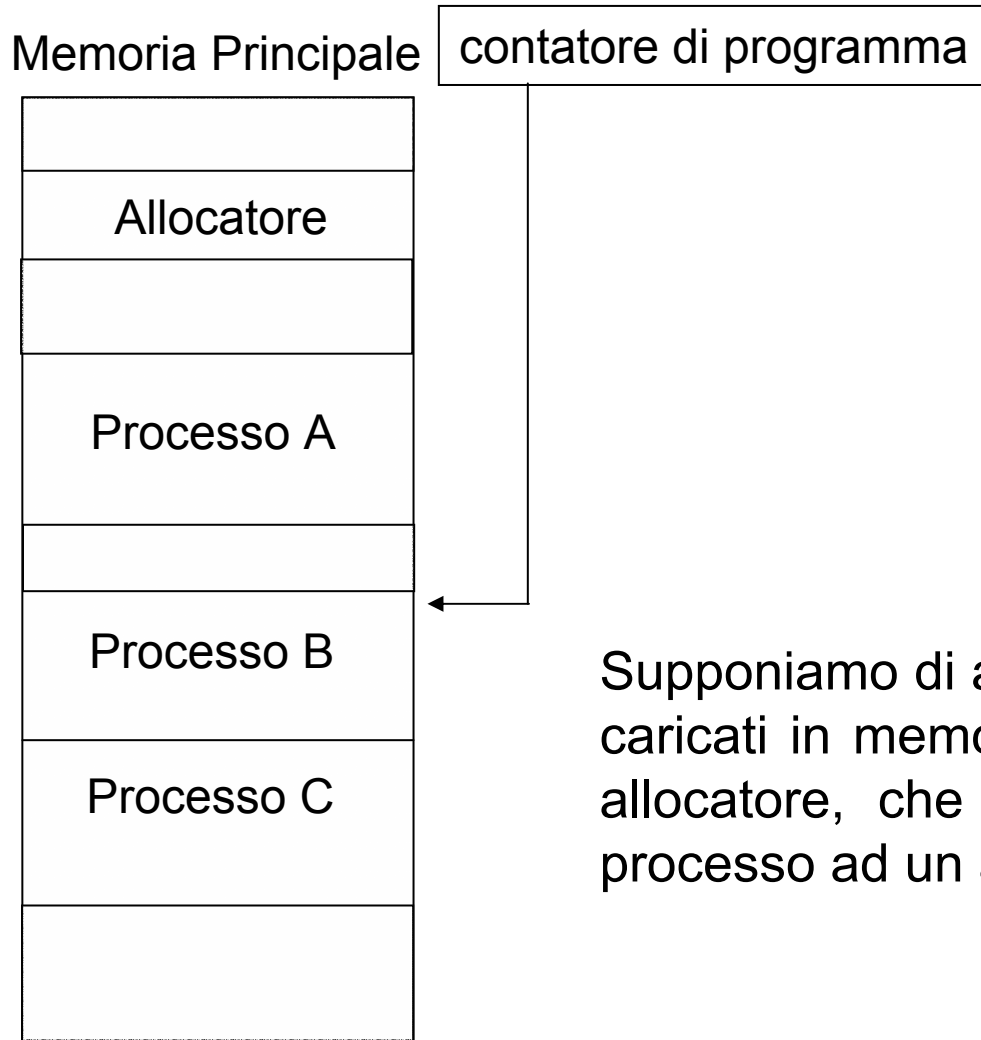
- Consideriamo un informatico appassionato di cucina che sta preparando la torta di compleanno per sua figlia.
- L'informatico ha la ricetta per la torta, e una cucina ben fornita con gli ingredienti necessari: farina, uova, zucchero, vaniglia e così via.
- In questa analogia,
  - ◆ la ricetta è il programma (cioè l'algoritmo espresso con una notazione adeguata),
  - ◆ l'informatico è il processore (CPU),
  - ◆ gli ingredienti della torta sono i dati in ingresso.
- Il processo è l'attività che il cuoco svolge leggendo la ricetta, raccogliendo gli ingredienti, impastando e cuocendo la torta.

*Tanenbaum*

# Concetto di processo

- Un sistema operativo esegue programmi di varia natura:
  - ◆ Sistemi batch: *job*
  - ◆ Sistemi time-sharing: *programmi utente* o *task*
- I libri di testo impiegano indifferentemente il termine *job* o *processo*.
- **Processo** — un programma in esecuzione; l'esecuzione di un processo deve avvenire in modo sequenziale.
- Un processo include:
  - ◆ il *program counter*
  - ◆ lo *stack*
  - ◆ una sezione dati

# Stati dei processi



Supponiamo di avere tre processi completamente caricati in memoria e di avere a disposizione un allocatore, che trasferisce il processore da un processo ad un altro.

# Tracce dei processi

$\alpha+0$	$\beta+0$	$\chi+0$
$\alpha+1$	$\beta+1$	$\chi+1$
$\alpha+2$	$\beta+2$	$\chi+2$
$\alpha+3$	$\beta+3$	$\chi+3$
$\alpha+4$		$\chi+4$
$\alpha+5$		$\chi+5$
$\alpha+6$		$\chi+6$
$\alpha+7$		$\chi+7$
$\alpha+8$		$\chi+8$
$\alpha+9$		$\chi+9$
$\alpha+10$		$\chi+10$
$\alpha+11$		$\chi+11$
Traccia di A	Traccia di B	Traccia di C

Tracce dei processi nella prima parte dell'esecuzione, con 12 istruzioni eseguite nei processi A e C. B esegue 4 istruzioni dopo di che chiama un'operazione di I/O

1	$\alpha+0$		26	$\chi+4$	
2	$\alpha+1$		27	$\chi+5$	
3	$\alpha+2$		28	-----	Tempo di esecuzione scaduto
4	$\alpha+3$		39	$\delta+0$	
5	$\alpha+4$		30	$\delta+1$	
6	$\alpha+5$		31	$\delta+2$	
Tempo di esecuzione scaduto			32	$\delta+3$	
			33	$\delta+4$	
			33	$\delta+5$	
			34	$\alpha+6$	
			35	$\alpha+7$	
			36	$\alpha+8$	
7	$\delta+0$		37	$\alpha+9$	
8	$\delta+1$		38	$\alpha+10$	
9	$\delta+2$		49	$\alpha+11$	
10	$\delta+3$		40	-----	Tempo di esecuzione scaduto
11	$\delta+4$		41	$\delta+0$	
12	$\delta+5$		42	$\delta+1$	
13	$\beta+0$		43	$\delta+2$	
14	$\beta+1$		44	$\delta+3$	
15	$\beta+2$		45	$\delta+4$	
16	$\beta+3$		46	$\delta+5$	
Richiesta di I/O			47	$\chi+6$	
			48	$\chi+7$	
			49	$\chi+8$	
			50	$\chi+9$	
			51	$\chi+10$	
			52	$\chi+11$	
17	$\delta+0$				Tempo di esecuzione scaduto
18	$\delta+1$				
19	$\delta+2$				
20	$\delta+3$				
21	$\delta+4$				
22	$\delta+5$				
23	$\chi+0$				
24	$\chi+1$				
25	$\chi+2$				
26	$\chi+3$				

# Stato del processo

- Mentre viene eseguito un processo cambia **stato**:
  - ◆ **New** (nuovo): Il processo viene creato.
  - ◆ **Running** (in esecuzione): Le istruzioni vengono eseguite.
  - ◆ **Waiting** (in attesa): Il processo è in attesa di un evento.
  - ◆ **Ready** (pronto): Il processo è in attesa di essere assegnato ad un processore.
  - ◆ **Terminated** (terminato): Il processo ha terminato la propria esecuzione.

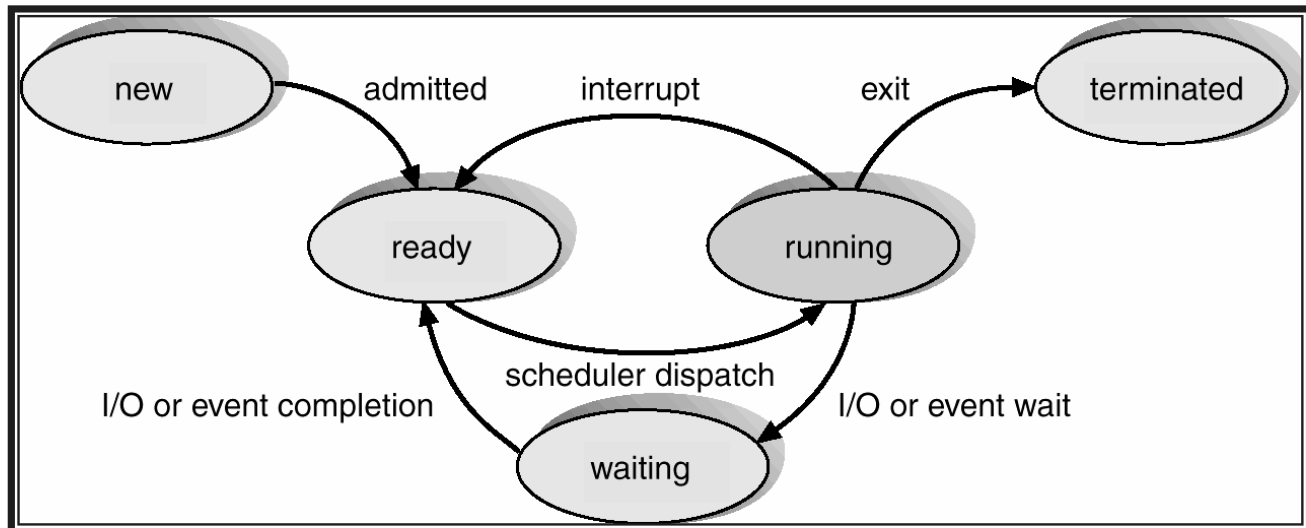
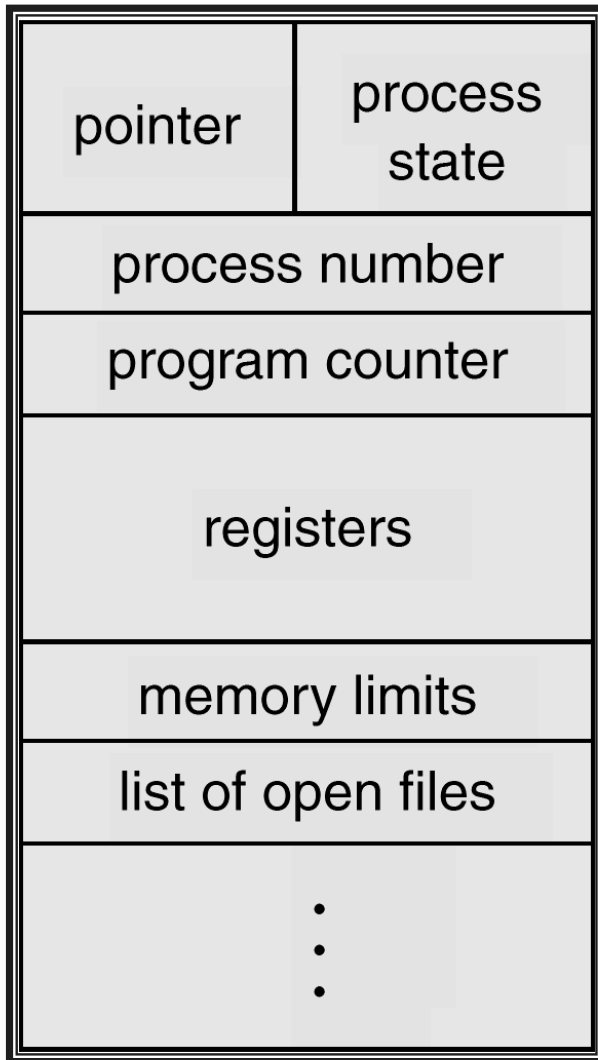


Diagramma degli stati di un processo

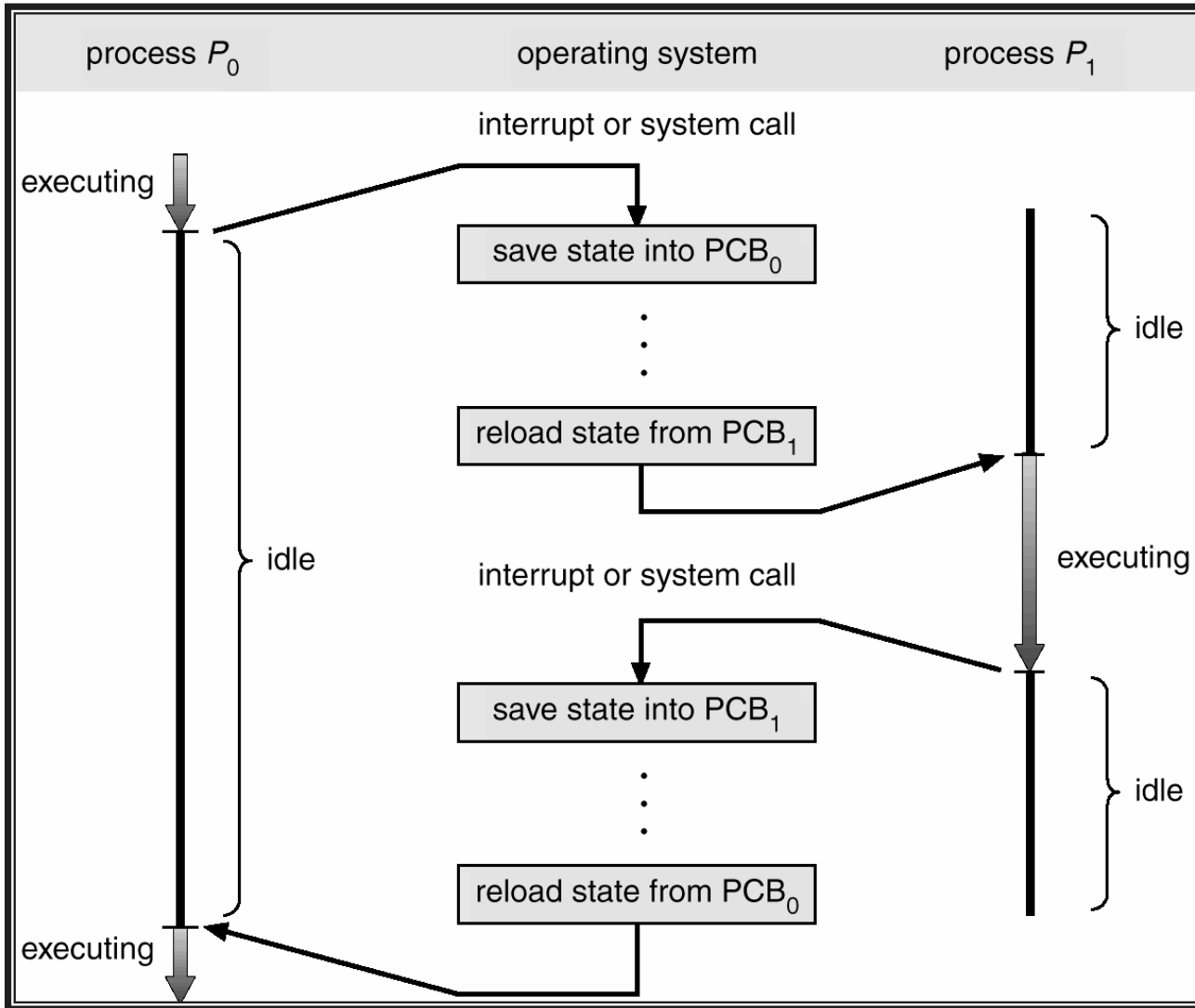
# Process Control Block (PCB)



Informazione associata a ciascun processo:

- Stato del processo
- Program counter
- Registri della CPU  
(accumulatori, indice, stack pointer)
- Informazioni sullo scheduling della CPU  
(priorità, puntatori alle code di scheduling)
- Informazioni sulla gestione della memoria  
(registri base e limite, tabella pagine/segmenti)
- Informazioni di contabilizzazione delle risorse  
(numero job/account, tempo di CPU)
- Informazioni sullo stato di I/O  
(lista dispositivi/file aperti)

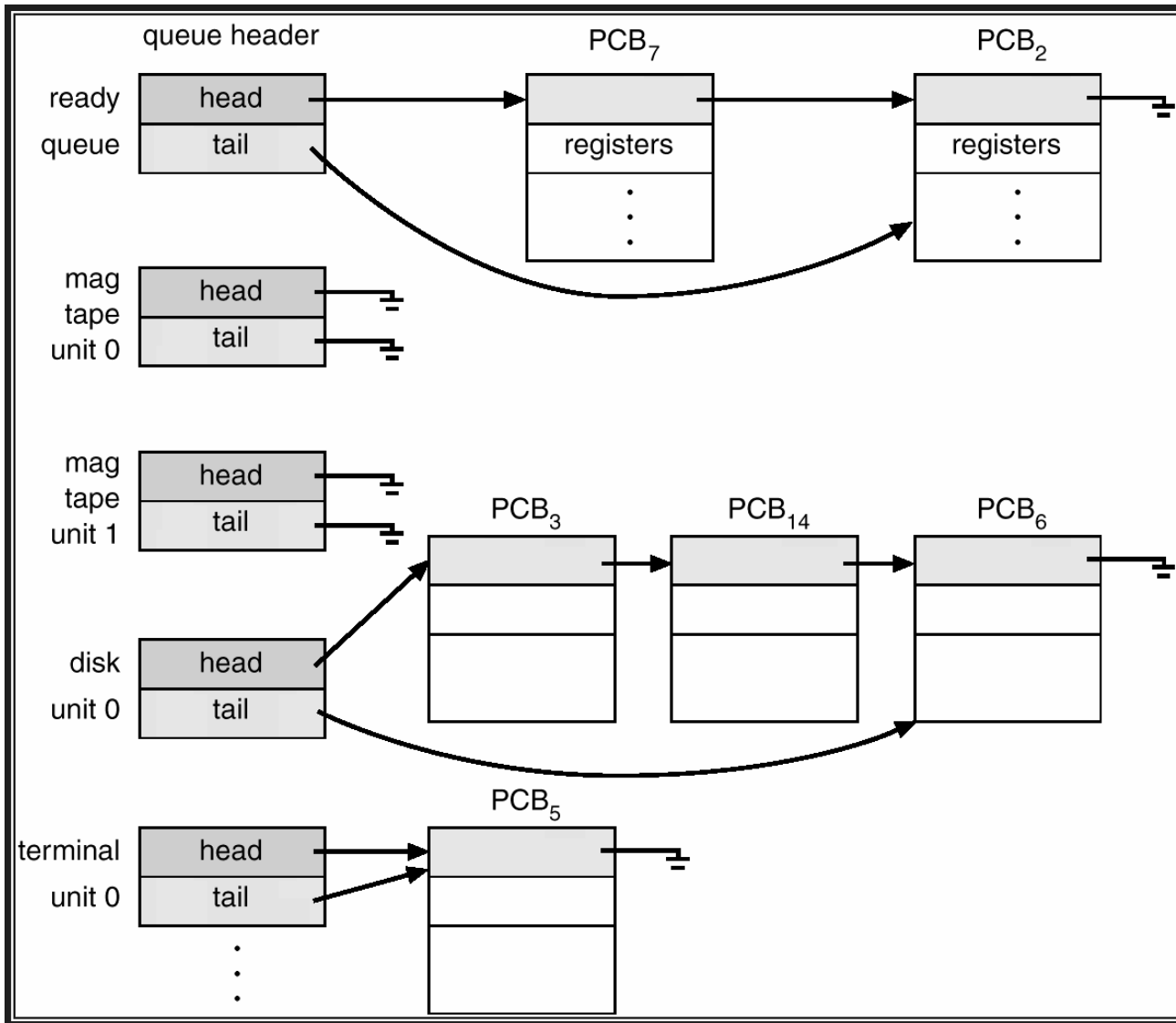
# Commutazione della CPU fra processi



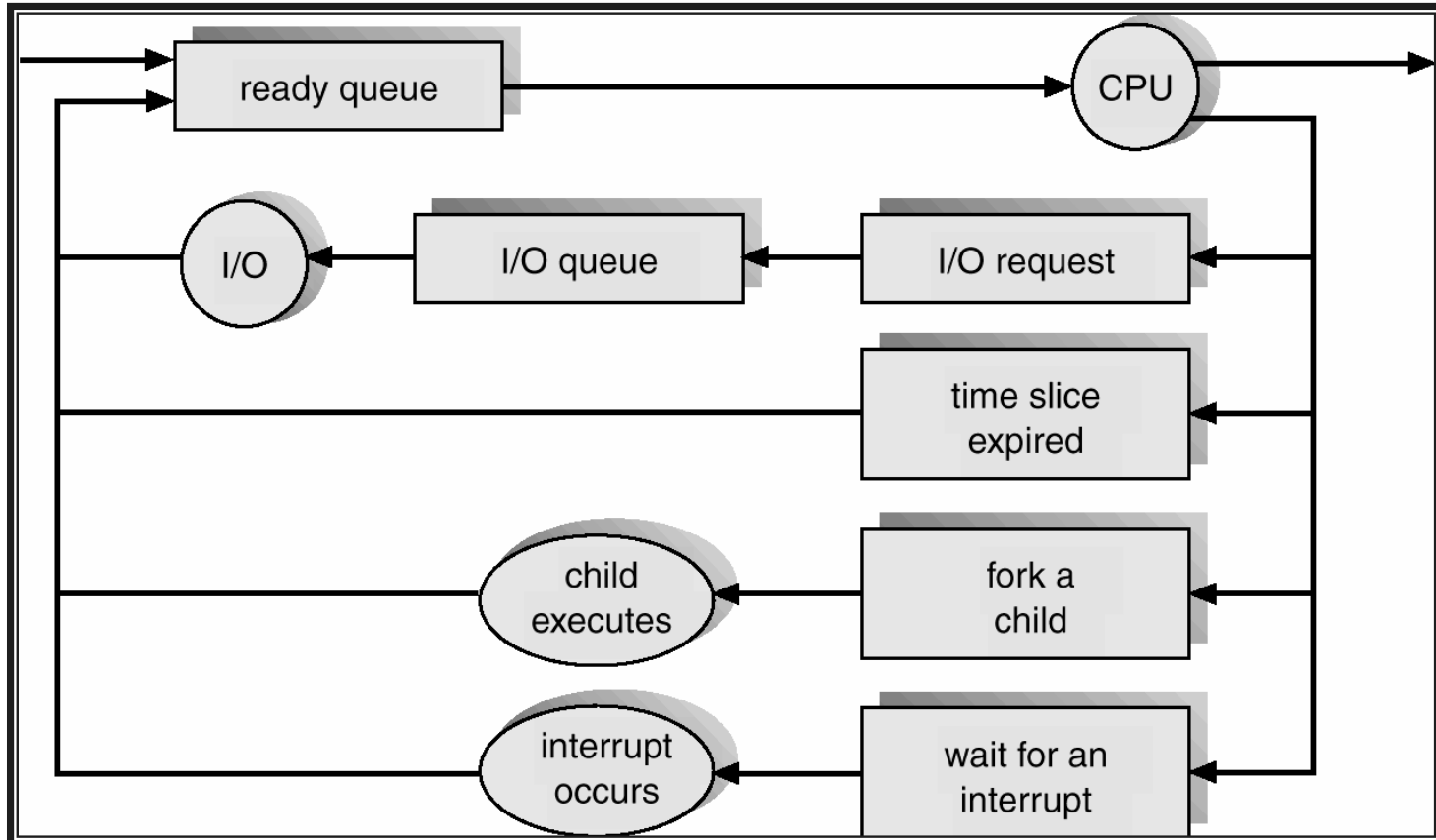
# Code per lo scheduling di processi

- **Coda dei job** — Insieme di tutti i processi presenti nel sistema.
- **Ready queue** (Coda dei processi pronti) — Insieme di tutti i processi pronti ed in attesa di esecuzione, che risiedono in memoria centrale.
- **Code dei dispositivi** — Insieme di processi in attesa per un dispositivo di I/O.
- I processi si “spostano” fra le varie code.

# Ready queue e code ai dispositivi di I/O



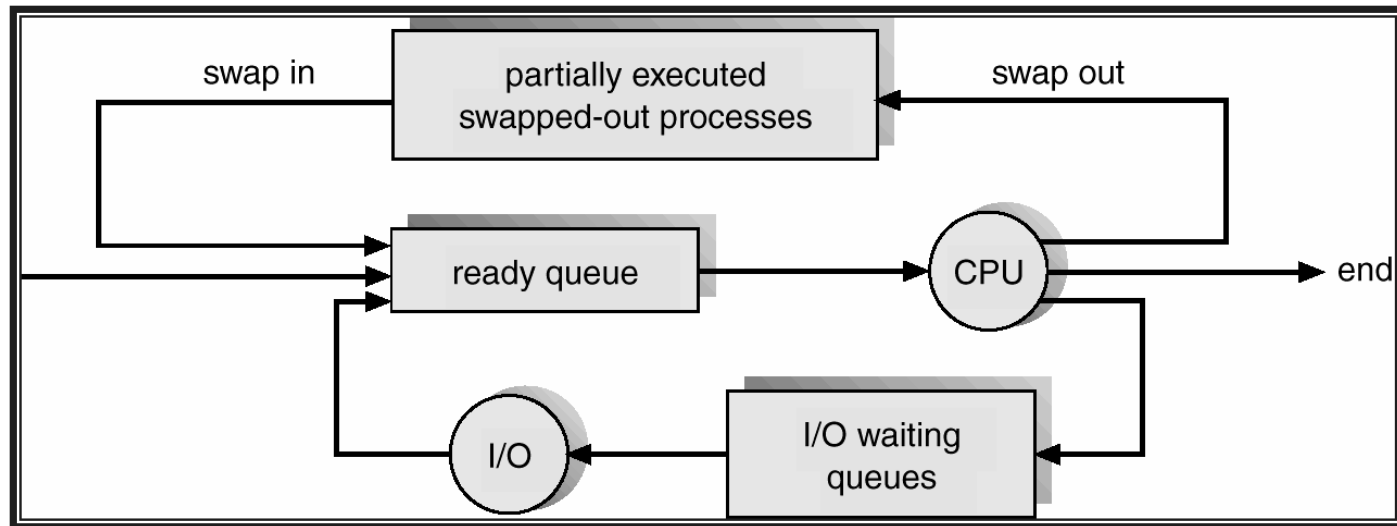
# Diagramma di accodamento per lo scheduling dei processi



Ogni riquadro rappresenta una coda. Le ellissi racchiudono le risorse che servono le code, mentre le frecce indicano il flusso dei processi nel sistema.

# Tipi di scheduler

- Scheduler a **lungo termine** (o scheduler dei job): seleziona quali processi devono essere portati dalla memoria di massa alla ready queue (in memoria centrale).
- Scheduler a **breve termine** (o scheduler della CPU): seleziona quale processo debba essere eseguito successivamente, ed alloca la CPU.
- Scheduler a **medio termine** (*swapper*): rimuove processi dalla memoria (e dalla contesa per la CPU) e riduce il grado di multiprogrammazione.



Scheduling a medio termine

# Tipi di scheduler

- Lo scheduler a breve termine viene chiamato molto spesso ( $\approx 100$  millisecondi)  $\Rightarrow$  deve essere veloce.
- Lo scheduler a lungo termine viene chiamato raramente (secondi, minuti)  $\Rightarrow$  può essere lento (ma efficiente).
- Lo scheduler a lungo termine controlla il *grado di multiprogrammazione*.
- I processi possono essere classificati in:
  - ◆ *Processi I/O-bound*: impiegano più tempo effettuando I/O rispetto al tempo impiegato per elaborazioni (in generale, si hanno molti *burst* di CPU di breve durata).
  - ◆ *Processi CPU-bound*: impiegano più tempo effettuando elaborazioni (in generale, si hanno pochi burst di CPU di lunga durata).

# Cambio di contesto

- Il figlio dell'informatico irrompe nella cucina piangendo e dicendo di essere stato punto da un'ape.
  - ◆ L'informatico segna il punto della ricetta a cui è arrivato (lo stato del processo corrente viene salvato),
  - ◆ tira fuori un libro di pronto soccorso e comincia a seguire le sue indicazioni.
- Questo è un esempio di come il processore cambi contesto da un processo (cucinare) ad un altro processo a priorità più alta (somministrare cure mediche), ciascuno con un programma diverso (la ricetta e il libro di pronto soccorso).
- Quando la puntura dell'ape è stata medicata, l'informatico ritorna alla sua torta, continuando dal punto dove era stato interrotto.

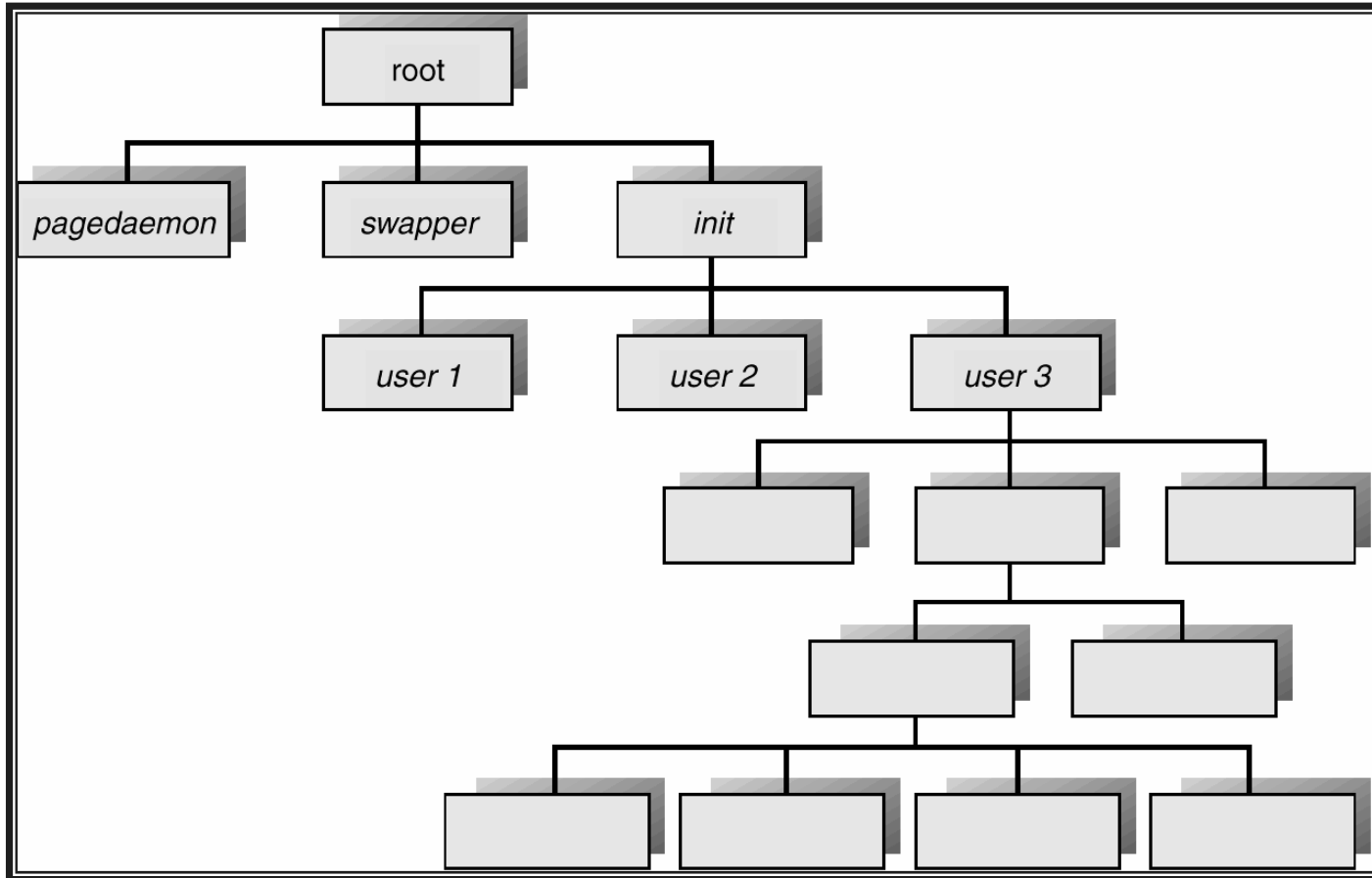
# Cambio di contesto (*context switch*)

- Quando la CPU passa da un processo all'altro, il sistema deve salvare lo stato del vecchio processo e caricare lo stato precedentemente salvato per il nuovo processo.
- Il tempo di *context-switch* è un sovraccarico (*overhead*); il sistema non lavora utilmente mentre cambia contesto.
- Il tempo di *context-switch* dipende dal supporto hardware (velocità di accesso alla memoria, numero di registri da copiare, istruzioni speciali, gruppi di registri multipli).

# Creazione di processi

- Il processo padre crea processi figli che, a loro volta, creano altri processi, formando un albero di processi.
- Condivisione di risorse
  - ◆ Il padre e il figlio condividono tutte le risorse.
  - ◆ I figli condividono un sottoinsieme delle risorse del padre.
  - ◆ Il padre e il figlio non condividono risorse.
- Esecuzione
  - ◆ Il padre e i figli vengono eseguiti concorrentemente.
  - ◆ Il padre attende la terminazione dei processi figli.
- Spazio degli indirizzi
  - ◆ Il processo figlio è un duplicato del processo padre.
  - ◆ Nel processo figlio è stato caricato un diverso programma.
- In **UNIX**: la system call **fork** crea un nuovo processo, la **execve** viene impiegata dopo una **fork** per sostituire lo spazio di memoria del processo originale con un nuovo programma.

# Albero dei processi in un tipico sistema UNIX



# Terminazione di processi

- Il processo esegue l'ultima istruzione e chiede al sistema operativo di essere cancellato per mezzo di una specifica chiamata di sistema (**exit** in UNIX) che compie le seguenti operazioni:
  - ◆ Può restituire dati (output) al processo padre (**wait**).
  - ◆ Le risorse del processo vengono deallocate dal SO.
- Il padre può terminare l'esecuzione dei processi figli (**abort**) se...
  - ◆ Il figlio ha ecceduto nell'uso di alcune risorse.
  - ◆ Il compito assegnato al figlio non è più richiesto.
  - ◆ Il padre termina.
    - ✓ Il sistema operativo non consente ad un processo figlio di continuare l'esecuzione se il padre è terminato. Questo fenomeno è detto *terminazione a cascata* e viene avviato dal SO.

# Processi cooperanti

- Un processo è **indipendente** se non può influire su altri processi nel sistema o subirne l'influsso.
- Processi **cooperanti** possono influire su altri processi o esserne influenzati.
- La presenza o meno di dati condivisi determina univocamente la natura del processo.
- Vantaggi della cooperazione fra processi
  - ◆ Condivisione di informazioni
  - ◆ Accelerazione del calcolo (in sistemi con più CPU)
  - ◆ Modularità
  - ◆ Convenienza

# Problema del produttore–consumatore

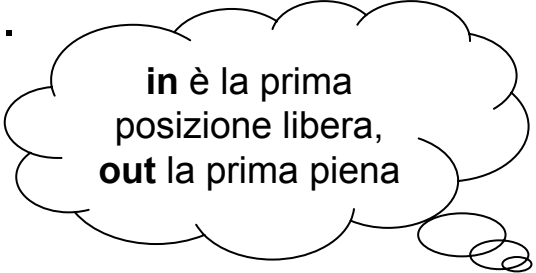
- È un paradigma classico per processi cooperanti. Il processo **produttore** produce informazioni che vengono consumate da un processo **consumatore**.
  - ◆ *Buffer illimitato*: non vengono posti limiti pratici alla dimensione del buffer.
  - ◆ *Buffer limitato*: si assume che la dimensione del buffer sia fissata.
- Esempio: Un programma di stampa produce caratteri che verranno consumati dal driver della stampante.

# Soluzione con buffer limitato e memoria condivisa

## ■ Dati condivisi

```
#define BUFFER_SIZE 10
typedef struct {
    ...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

- La soluzione ottenuta è corretta, ma consente l'utilizzo di soli  $BUFFER\_SIZE-1$  elementi.



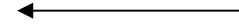
**in** è la prima  
posizione libera,  
**out** la prima piena

# Soluzione con buffer limitato e memoria condivisa

```
item nextProduced;

while (1) {
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```

Processo  
Produttore



```
item nextConsumed;

while (1) {
    while (in == out)
        ; /* do nothing */
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
}
```

Processo  
Consumatore



# Comunicazione tra processi (IPC)

- **IPC, *Inter-Process Communication***: Meccanismo per la comunicazione e la sincronizzazione fra processi.
- Sistema di messaggi — i processi comunicano fra loro senza far uso di variabili condivise.
- La funzionalità IPC consente due operazioni:
  - ◆ **send(messaggio)** — la dimensione del messaggio può essere fissa o variabile;
  - ◆ **receive(messaggio)**.
- Se i processi  $P$  e  $Q$  vogliono comunicare, devono:
  - ◆ stabilire fra loro un canale di comunicazione;
  - ◆ scambiare messaggi per mezzo di send/receive.
- Implementazione del canale di comunicazione:
  - ◆ fisica (es. memoria condivisa, bus hardware);
  - ◆ logica (proprietà logiche).

# Problemi di implementazione

- Come vengono stabiliti i canali (connessioni)?
- È possibile assegnare un canale a più di due processi?
- Quanti canali possono essere stabiliti fra ciascuna coppia di processi comunicanti?
- Qual è la capacità di un canale?
- Il formato del messaggio che un canale può gestire è fisso o variabile?
- Stabilire canali monodirezionali o bidirezionali?

# Comunicazione diretta

- I processi devono “nominare” esplicitamente i loro interlocutori (modalità simmetrica):
  - ◆ **send** ( $P, \text{messaggio}$ ) — invia un messaggio al processo  $P$
  - ◆ **receive**( $Q, \text{messaggio}$ ) — riceve un messaggio dal processo  $Q$
- Proprietà del canale di comunicazione:
  - ◆ I canali vengono stabiliti automaticamente.
  - ◆ Ciascun canale è associato esattamente ad una coppia di processi.
  - ◆ Tra ogni coppia di processi comunicanti esiste esattamente un canale.
  - ◆ Il canale può essere unidirezionale (ogni processo collegato al canale può soltanto trasmettere/ricevere), ma è normalmente bidirezionale.

# Comunicazione indiretta

- I messaggi vengono inviati/ricevuti da **mailbox** (*porte*).
  - ◆ Ciascuna mailbox è identificata con un **id** unico.
  - ◆ I processi possono comunicare solamente se condividono una mailbox.
- Proprietà dei canali di comunicazione:
  - ◆ Un canale viene stabilito solo se i processi hanno una mailbox in comune.
  - ◆ Un canale può essere associato a più processi.
  - ◆ Ogni coppia di processi può condividere più canali di comunicazione.
  - ◆ I canali possono essere unidirezionali o bidirezionali.
- Operazioni:
  - ◆ creare una nuova mailbox
  - ◆ Inviare/ricevere messaggi attraverso mailbox
  - ◆ distruggere una mailbox

# Comunicazione indiretta

## ■ Primitive di comunicazione:

- ✦ **send**( $A$ , *messaggio*) — invia un messaggio alla mailbox  $A$
- ✦ **receive**( $A$ , *messaggio*) — riceve un messaggio dalla mailbox  $A$

## ■ Condivisione di mailbox

- ✦  $P_1$ ,  $P_2$ , e  $P_3$  condividono la mailbox  $A$ .
- ✦  $P_1$ , invia;  $P_2$  e  $P_3$  ricevono.
- ✦ Chi si assicura il messaggio?

## ■ Soluzioni:

- ✦ Permettere ad un canale di essere associato ad al più due processi.
- ✦ Permettere ad un solo processo alla volta di eseguire un'operazione di ricezione.
- ✦ Permettere al SO di selezionare arbitrariamente il ricevente. Il sistema comunica l'identità del ricevente al trasmittente.

# Sincronizzazione

- Lo scambio di messaggi può essere sia *bloccante* che *non-bloccante*.
- In caso di scambio di messaggi bloccante la comunicazione è **sincrona**.
- In caso di scambio di messaggi non-bloccante la comunicazione è **asincrona**.
- Le primitive **send** e **receive** possono essere sia bloccanti che non-bloccanti.

# Buffering

- La coda dei messaggi legata ad un canale può essere implementata in tre modi.
  1. **Capacità zero** — Il canale non può avere messaggi in attesa al suo interno. Il trasmittente deve attendere che il ricevente abbia ricevuto il messaggio (*rendezvous*).
  2. **Capacità limitata** — Lunghezza finita pari a  $n$  messaggi. Se il canale è pieno, il trasmittente deve attendere.
  3. **Capacità illimitata** — Lunghezza infinita. Il trasmittente non attende mai.

# Condizioni di eccezione

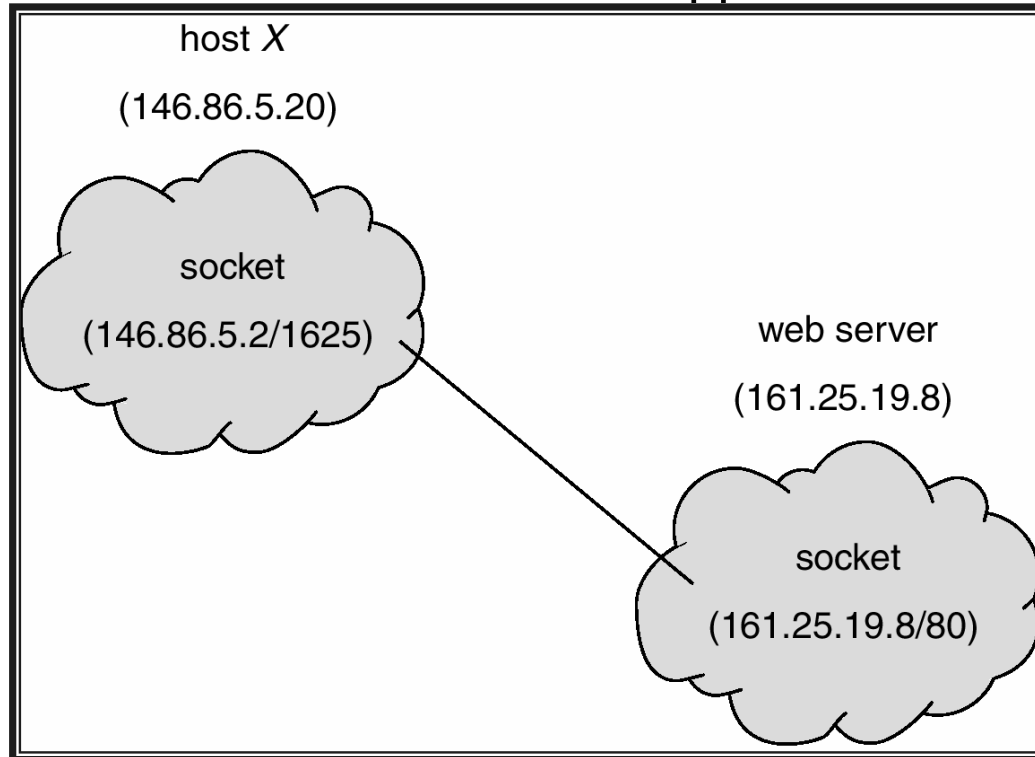
- **Terminazione del processo:** un processo trasmittente o ricevente può terminare prima che un messaggio sia stato elaborato  $\Rightarrow$  messaggi mai ricevuti, processi bloccati in attesa.
- **Messaggi perduti:** a causa di guasti sui canali di comunicazione; il SO o il processo trasmittente sono responsabili del rilevamento della condizione di eccezione e della ripetizione del messaggio.
- **Messaggi alterati:** a causa di disturbi sul canale di comunicazione; il messaggio deve essere ritrasmesso.

# Comunicazione client-server

- **Socket**
- **Chiamate a procedura remota** (*RPC- Remote Procedure Calls*)
- **Invocazione di metodi remoti** (*RMI- Remote Method Invocation*)

# Socket

- Una socket è definita come *un'estremità di un canale di comunicazione*.
- Concatenazione di indirizzi IP e porte.
- La socket **161.25.19.8:1625** si riferisce alla porta **1625** sull'host **161.25.19.8**
- La comunicazione avviene tra coppie di socket.



# Socket

- In genere le socket impiegano un'architettura client-server: il server è in ascolto delle richieste dei client su una determinata porta.
- I server che svolgono servizi specifici (FTP, HTTP, SMTP,...) sono in ascolto su porte note (tutte le porte <1024 sono considerate note e si usano per servizi standard).
- Quando un processo client richiede una connessione, il server assegna una porta, su cui stabilisce la connessione.

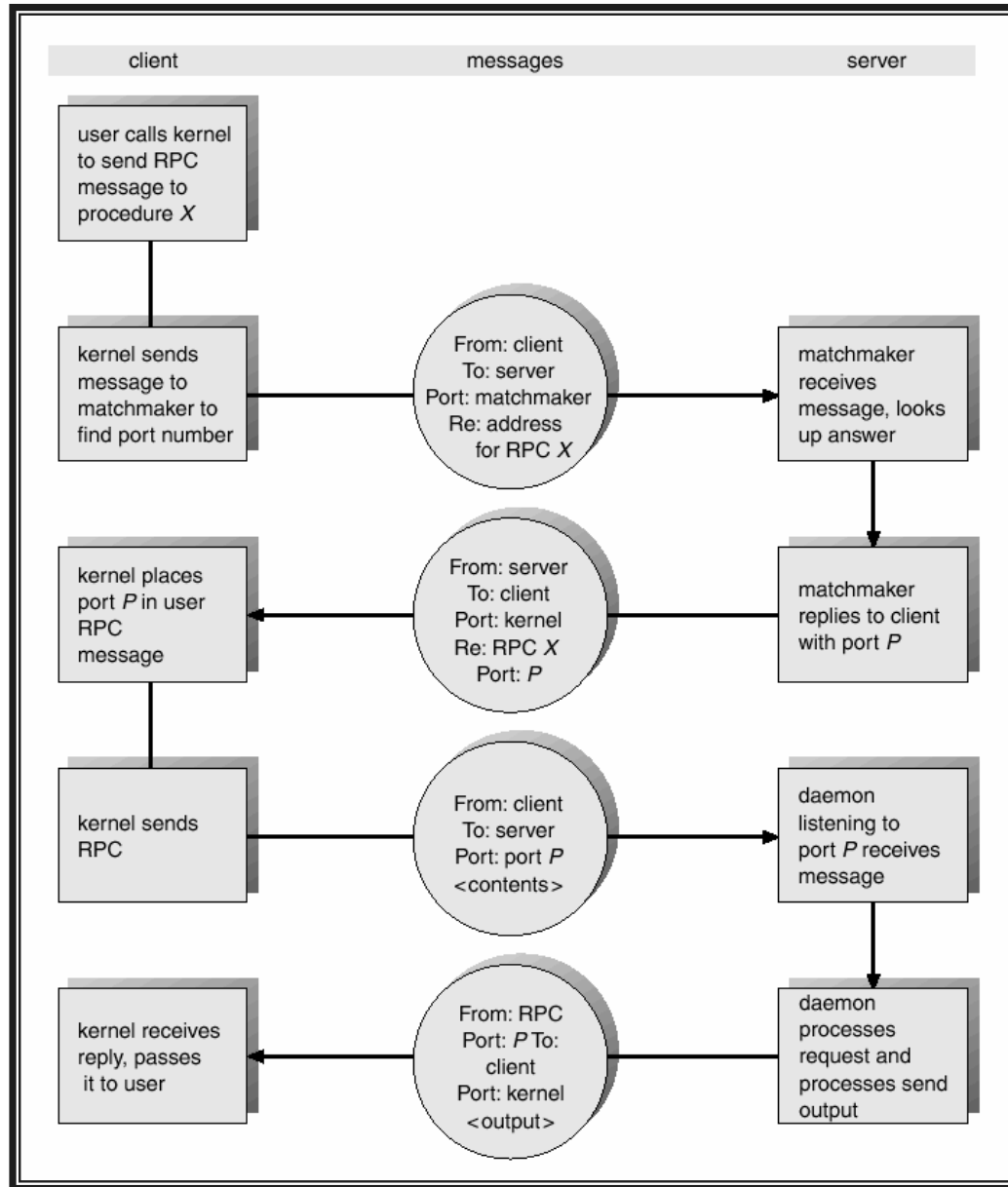
# Chiamate a procedure remote

- La **chiamata a procedura remota** (RPC- *Remote Procedure Call*) astrae il concetto di chiamata a procedura tra processi in sistemi di rete.
- **Stub**– segmento di codice dal lato client per la procedura sul server.
- Lo stub individua la porta sul server e struttura i parametri (*marshalling*).
- Un analogo segmento di codice sul server riceve il messaggio, attiva la procedura sul server e, se necessario, restituisce i dati al client.

# Problemi nella RPC

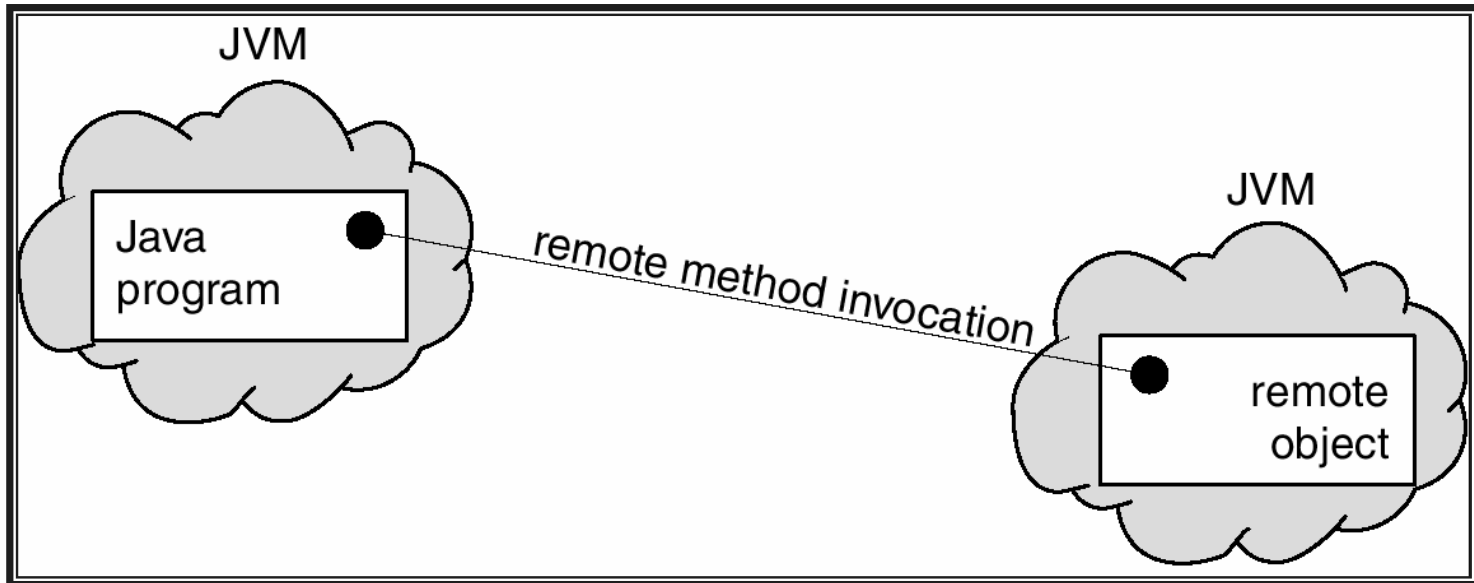
- Tutti i problemi legati alla rappresentazione dei dati sono trasparenti all'utente.
- Le chiamate a procedura remota possono non funzionare correttamente o essere duplicate ed essere eseguite più volte a causa di errori sulla rete di comunicazione.
- RPC richiede la conoscenza dei numeri delle porte del server.
  - ◆ Predeterminati
  - ◆ Matchmaker

# Esecuzione di RPC



# Invocazione di metodi remoti

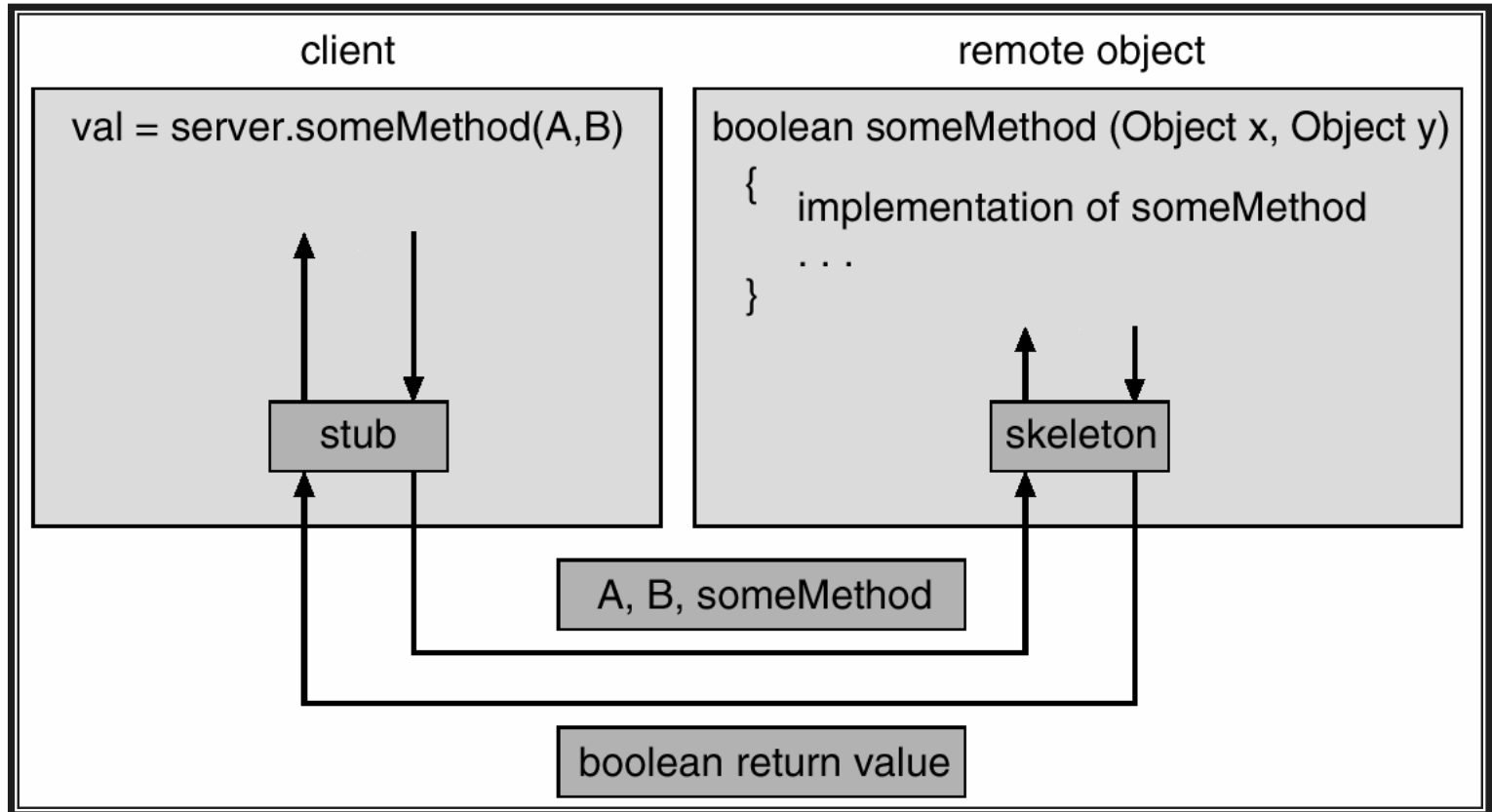
- L'**Invocazione a metodo remoto** (*RMI - Remote Method Invocation*) è un meccanismo Java simile all' RPC.
- L'RMI permette ad un programma Java in esecuzione su un sistema di invocare un metodo di un oggetto remoto.



# Invocazione di metodi remoti

- Per rendere i metodi remoti trasparenti al client ed al server, nel client è presente un segmento di codice (*stub*) nel client ed uno (*skeleton*) nel server.
- Quando viene invocato un metodo remoto, nel client viene chiamato il codice di riferimento, che struttura i parametri del metodo, li impacchetta, e li invia al server.
- Nel server i parametri del metodo vengono destrutturati e viene invocato nel server il metodo richiesto.
  - ◆ Se i parametri sono oggetti locali, sono passati per copiatura (*serializzazione*), altrimenti per riferimento.
  - ◆ Per poter essere serializzato, il stato di un oggetto si deve poter scrivere come un flusso di byte.

# Strutturazione dei parametri (*marshalling*)



# Sommario

- Un processo è un programma in esecuzione. Nel corso della sua attività cambia di stato, e tale stato è definito dall'attività corrente.
- Un processo, se non è in esecuzione, è in una coda. Le code principali sono quella di richiesta di I/O e quella dei processi pronti ad essere eseguiti dalla CPU.
- Ogni processo è rappresentato da un PCB, e i PCB si possono collegare per formare le code.
- La scelta dei processi che si contenderanno la CPU è detta scheduling a lungo termine (*job scheduling*).
- Lo scheduling a breve termine consiste nella scelta di un processo dalla coda dei processi pronti.

# Sommario

- I processi possono essere eseguiti in maniera concorrente per aumentare le prestazioni, la condivisione delle informazioni e la modularità.
- I processi cooperanti hanno bisogno di comunicare tra loro. Gli schemi principali sono la memoria condivisa e lo scambio dei messaggi.
- Una socket è un punto terminale di comunicazione. Con una coppia di socket è possibile mettere in comunicazione una coppia di applicazioni.
- Nei sistemi distribuiti è possibile eseguire procedure in applicazioni remote.
- La versione object-oriented dell'RPC è l'RMI.