Il nucleo di un sistema a processi

Nucleo di un Sistema a Processi

- Il **modello a processi** prevede l'esistenza di tante unità di elaborazione (***macchine virtuali***) quanti sono i processi; ogni macchina possiede come set di istruzioni elementari quelle corrispondenti all'unità centrale reale più le istruzioni relative alla ***creazione*** *ed* ***eliminazione*** *dei processi, al meccanismo di* ***comunicazione*** *e* ***sincronizzazione*** (compresa la comunicazione con i dispositivi di I/O visti come processori esterni).
- Questo modello consente di mettere in evidenza le **proprietà logiche** di comunicazione e sincronizzazione tra processi senza doversi occupare degli aspetti implementativi legati alle particolari caratteristiche del processore fisico (es. gestione delle interruzioni).

Def: Si chiama **nucleo** (*kernel*) il modulo (o insieme di funzioni) realizzato in *software, hardware o firmware* che supporta il concetto di processo e realizza gli strumenti necessari per la gestione dei processi.

- Il nucleo costituisce il *livello più interno* di un qualunque sistema basato su processi. Ad esempio:
  - il livello più elementare di un *sistema operativo multiprogrammato;*
  - il supporto a tempo di esecuzione di un *linguaggio per la programmazione concorrente*.
- Il nucleo è il solo modulo che è *conscio* dell'esistenza delle ***interruzioni***. I processi che colloquiano con i dispositivi utilizzano opportune primitive del nucleo che provvedono a **sospenderli** in attesa del completamento dell'azione richiesta.

- Quando l'azione è completata, il relativo segnale di interruzione inviato dal dispositivo alla CPU viene ***catturato*** dal nucleo che provvede a ***risvegliare*** il processo sospeso.

- La ***gestione delle interruzioni*** è quindi ***invisibile ai processi*** ed ha come unico effetto rilevabile di rallentare la loro esecuzione sulle rispettive macchine virtuali.

- Nel seguito verranno descritte le ***strutture dati*** e le ***procedure*** di un nucleo per sistemi monoprocessore e multiprocessore.

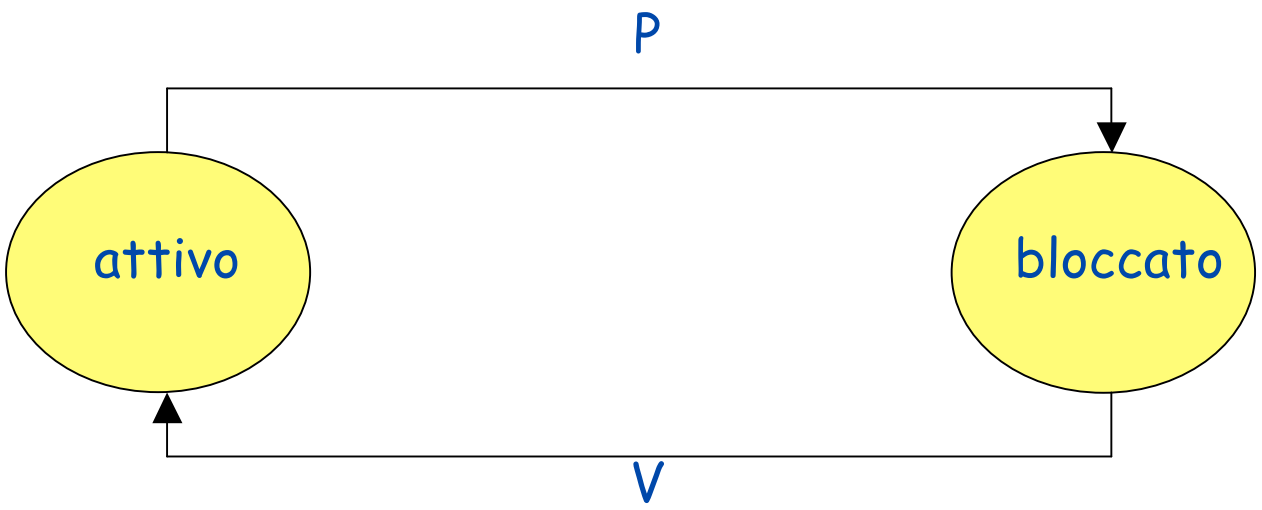
Obiettivo: **realizzazione dei processi** e loro **sincronizzazione**.

Non verranno trattati problemi quali la allocazione dinamica della memoria e le politiche di scheduling

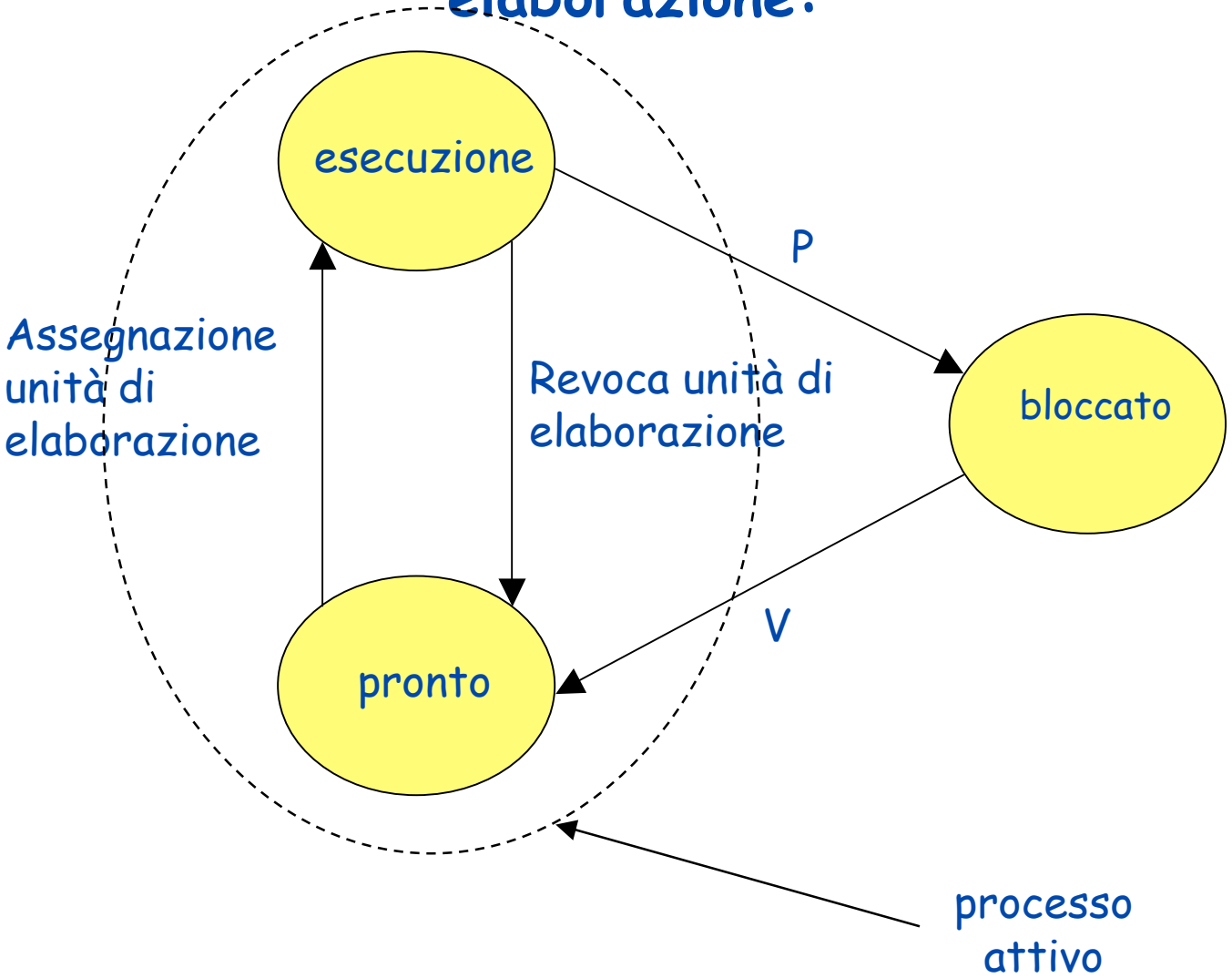
Caratteristiche fondamentali del nucleo

- **Efficienza**: condiziona l'intera struttura a processi. Per questo motivo esistono sistemi in cui alcune o tutte le operazioni del nucleo sono realizzate in hardware o mediante microprogrammi.
- **Dimensioni**: la semplicità delle funzioni richieste al nucleo fa si che la sua dimensione risulti estremamente limitata
- **Separazione tra meccanismi e politiche**: il nucleo deve, per quanto possibile contenere solo meccanismi consentendo così, a livello di processi, di utilizzare tali meccanismi per la realizzazione di diverse politiche di gestione a seconda del tipo di applicazione

Stati di un processo:



Stati di un processo in sistemi in cui il numero di processi supera il numero delle unità di elaborazione:



- Quando un processo perde il controllo del processore, il contenuto dei registri del processore viene salvato in un'area di memoria associata al processo, chiamata ***descrittore***.

- Ciò consente una ***maggiore flessibilità*** nella politica di assegnazione del processore ai processi, rispetto alla soluzione di salvare le informazioni nello ***stack*** : infatti, in un istante generico, uno qualunque dei processi pronti può ottenere l'uso del processore e ricaricare i registri con i valori salvati nel descrittore di processo.

Contesto di un processo: è l'insieme delle informazioni contenute nei *registri del processore* e relative al processo

- La funzione fondamentale del nucleo di un sistema a processi è la **gestione delle transizioni di stato** dei processi. Più precisamente il nucleo deve:

a) Gestire il *salvataggio* ed il *ripristino* dei contesti dei processi:

  - Quando un processo abbandona il controllo dell'unità di elaborazione fisica (passaggio dallo stato di esecuzione allo stato di bloccato o di pronto), tutte le informazioni contenute nei registri di tale unità **devono essere trasferite nel descrittore**.
  - Analogamente quando un processo riprende l'esecuzione (passaggio dallo stato di pronto allo stato di esecuzione) tutte le informazioni contenute nel suo descrittore **devono essere** trasferite **nei registri di macchina**.

b) Scegliere a quale tra i processi pronti assegnare l'unità di elaborazione (*scheduling* della CPU):

Quando un processo abbandona il controllo dell'unità di elaborazione, il nucleo deve scegliere tra tutti i processi pronti quello da mettere in esecuzione. La scelta può essere o di tipo FIFO, oppure può utilizzare la priorità dei processi.

c) Gestire le interruzioni dei dispositivi esterni

traducendole eventualmente in attivazione di processi da bloccato a pronto.

d) Realizzare le primitive di sincronizzazione dei processi

gestendo il passaggio dei processi dallo stato di esecuzione allo stato di bloccato e da bloccato a pronto (primitive wait e signal), la preparazione di un descrittore per un processo ed il suo inserimento nella coda dei processi pronti o le operazioni inverse (creazione e distruzione di un processo)

STRUTTURE DATI del Nucleo

Descrittore del processo. Contiene le seguenti informazioni:

- **Identificazione del processo:**

  Ad ogni processo è associato un nome che lo identifica univocamente durante il suo tempo di vita.

- **Modalità di servizio dei processi:**
  - *FIFO*
  - *Priorità* (fissa o variabile)
  - *Deadline.*Il descrittore contiene un valore che sommato all'istante di richiesta di servizio da parte del processo determina il tempo massimo entro il quale la richiesta può essere soddisfatta.
  - *Quanto di tempo.* Sistemi time sharing

- **Contesto del processo:**

  contatore di programma, registro di stato, registri generali, indirizzo dell'area di memoria privata del processo.

- **Identificazione del processo successivo:**

  a seconda del loro stato i processi vengono inseriti, come si vedrà, in apposite *code*. Ogni descrittore contiene pertanto l'identificatore del processo successivo nella stessa *coda*.

Descrittore del Processo

Esempio di realizzazione:

```
typedef struct
{   int indice_priorità;
    int delta_t;
}modalità_di_servizio;

typedef struct
{ int nome;
   modalità_di_servizio servizio;
    tipo_contesto contesto;
   int   successivo;
}descrittore_processo;

/* insieme di tutti i descrittori:*/
descrittore_processo descrittori[num_max_proc];
```

Coda dei processi pronti

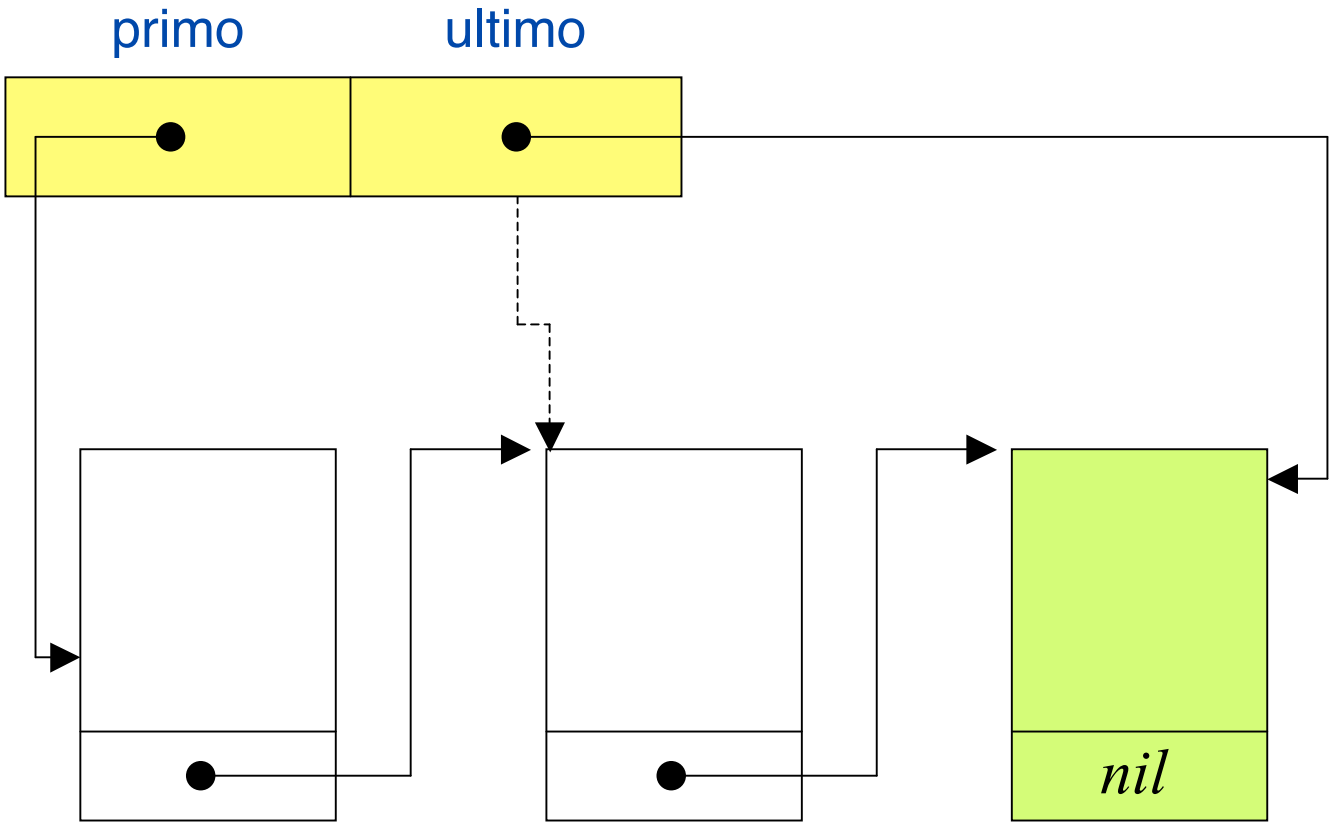
- Esistono *una o più code* di processi pronti (caso di processi con priorità).Quando un processo e` riattivato per effetto di una signal, viene inserito al fondo della coda corrispondente alla sua priorità.
- La coda dei processi pronti contiene un *processo fittizio* (**dummy process**) che va in esecuzione quando tutti i processi sono temporaneamente bloccati. Il processo *dummy* ha la *priorità più bassa* ed è sempre nello *stato di pronto*.
- Il processo dummy rimane in esecuzione fino a quando qualche altro processo diventa pronto, eseguendo un ciclo senza fine (oppure una funzione di servizio, o una di bassa priorità).

Coda dei processi pronti

- Esempio di realizzazione:

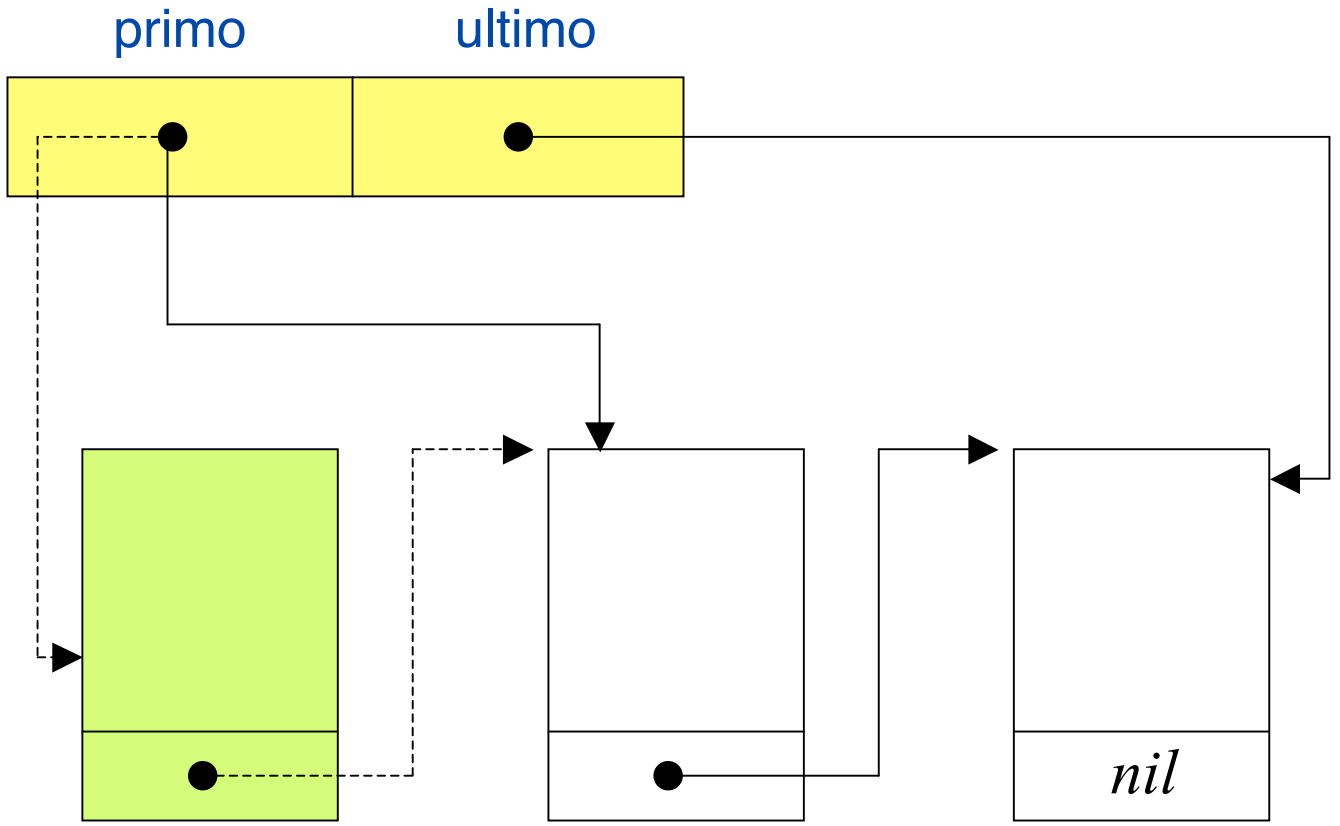
```
typedef struct
{int primo,ultimo;}descrittore_coda;

typedef descrittore_coda coda_a_ livelli[Npriorità];

coda_a_livelli  coda_processi_pronti;
```

Inserimento di un descrittore in coda:




```
void Inserimento(int P, descrittore_coda C){..}
/* inserisce il processo di indice P nella coda C */
```

Prelievo di un descrittore da una coda:




```
int Prelievo(descrittore_coda C){..}

/* estrae il primo processo dalla coda C e restituisce
il suo indice */
```

Descrittori liberi:

- Coda nella quale sono concatenati i descrittori *disponibili* per la creazione di nuovi processi e nella quale *sono ritornati* i descrittori dei processi eliminati

```
descrittore_coda descrittori_liberi;
```

Processo in esecuzione:

- Il nucleo necessita di conoscere quale processo è in esecuzione. Questa informazione, rappresentata dall'indice del descrittore del processo, viene contenuta in un particolare registro del processore :

```
int  processo_in_esecuzione; /* indice del processo
                                che sta usando la CPU*/
```

- Quando il nucleo è inizializzato (il che avviene durante l'operazione di *bootstrap* dell'elaboratore), viene creato un processo e l'indice del processo viene inserito nel registro *processo in esecuzione*.

Funzioni del Nucleo

- Le funzioni del nucleo realizzano le operazioni di ***transizione di stato*** per i singoli processi. Ogni transizione prevede il prelievo, da una coda, del descrittore del processo coinvolto ed il suo inserimento in un'altra coda.
- Si utilizzano a questo scopo due procedure: ***Inserimento*** e ***Prelievo*** di un descrittore da una coda. Se la coda è vuota si adotta l'ipotesi che il campo *primo* assuma il valore -1.
- Analogamente verrà assegnato il valore -1 al campo *successivo* dell'ultimo descrittore nella coda.

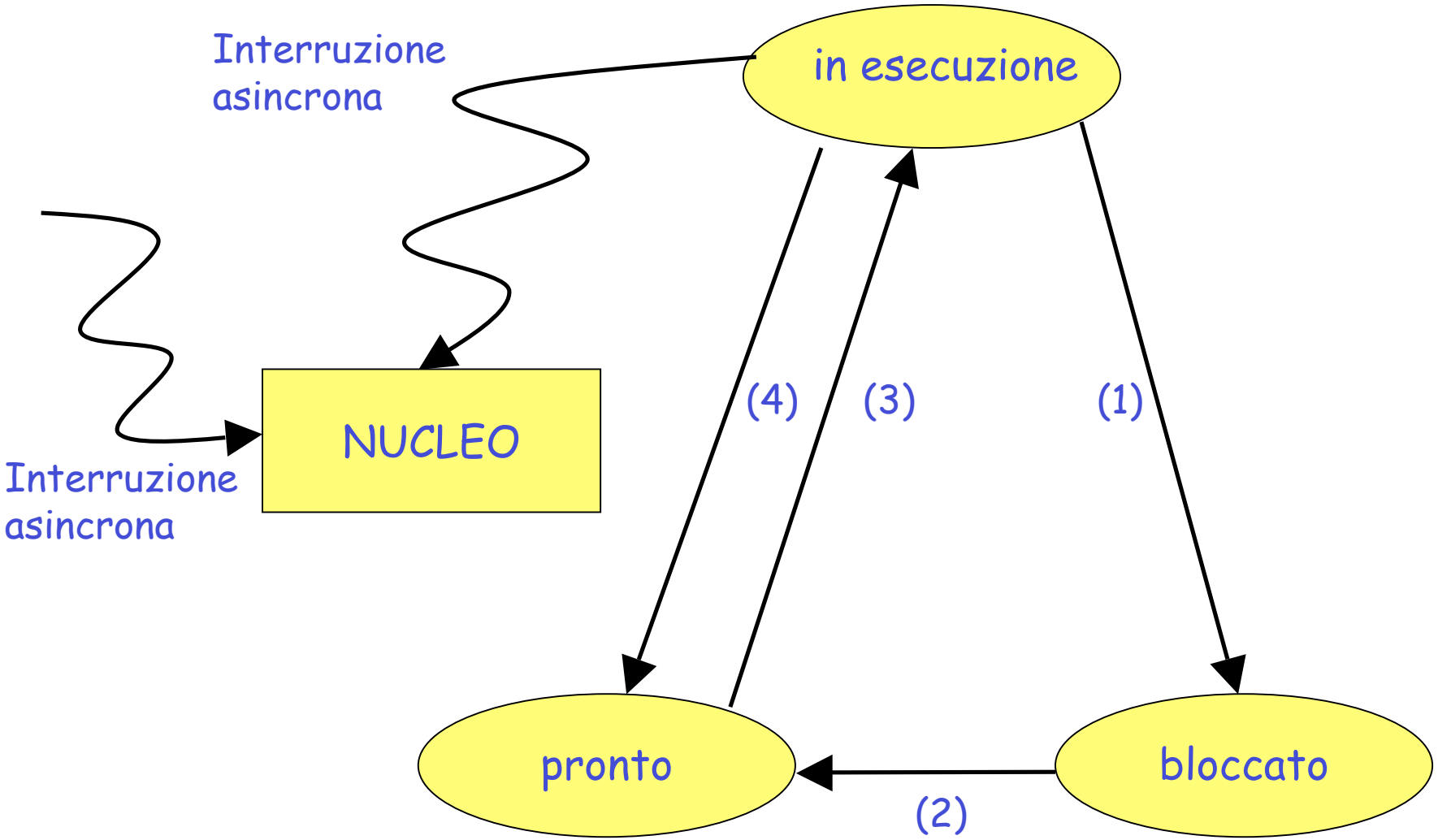
Funzioni del nucleo

- Le funzioni del nucleo possono essere suddivise *in due livelli:*
  - **Livello superiore:** contiene tutte le funzioni direttamente utilizzabili dai processi sia esterni (dispositivi di I/O) sia interni, quali *risposta ai segnali di interruzione* e *primitive per la creazione, eliminazione e sincronizzazione dei processi.*
  - **Livello inferiore:** realizza le funzionalità di ***cambio di contesto***: ***salvataggio*** del contesto del processo che si sospende nel suo descrittore, ***scelta*** di *un nuovo processo* da mettere in esecuzione tra quelli pronti e ***ripristino*** del suo contesto.

- *L'ambiente di esecuzione* delle funzioni del nucleo ha caratteristiche *distinte* da quello dei processi.
- Le funzioni del nucleo, per motivi di protezione, sono le *sole* che possono operare sulle *strutture dati* che rappresentano lo stato del sistema(descrittori, code di descrittori, semafori…) e le *sole* che possono utilizzare *istruzioni privilegiate* (abilitazione e disabililtazione delle interruzioni ,invio di comandi ai dispositivi).
- Le funzioni del nucleo devono essere eseguite in modo *mutuamente esclusivo*.
- I due ambienti corrispondono a stati *diversi* di operazione dell'elaboratore (*stati di supervisione e di utente*). Il meccanismo di passaggio da uno all'altro è basato sul *meccanismo delle interruzioni*

Più precisamente:

- Nel caso di funzioni chiamate da processi esterni, il passaggio all'ambiente del nucleo è ottenuto tramite il *meccanismo di risposta al segnale di interruzione* (interruzioni *asincrone o esterne*)
- Nel caso di funzioni chiamate da processi interni, il passaggio è ottenuto tramite l'esecuzione di *istruzioni del tipo chiamata al supervisore*, SVC (interruzioni *sincrone o interne*).
- In entrambi i casi, al completamento della funzione richiesta, il trasferimento all'ambiente di utente avviene tramite il *meccanismo di ritorno da interruzione.*


Interruzione
asincrona
in esecuzione
NUCLEO
Interruzione
asincrona
(4)
(3)
(1)
pronto
bloccato
(2)

Funzioni dei livello inferiore: CAMBIO DI CONTESTO

- *Salvataggio* del contesto del processo in esecuzione nel suo descrittore.(**Salvataggio_stato**)
- *Inserimento del descrittore* nella coda dei processi bloccati o dei processi pronti.
- *Rimozione del processo a maggior priorità* dalla coda dei pronti e caricamento dell'identificatore di tale processo nel registro *processo in esecuzione* (**Assegnazione_CPU**)
- *Caricamento* del contesto del nuovo processo nei registri di macchina.(**Ripristino_stato)**

Realizzazione

```
void Salvataggio_stato( )
{       int j;
        j = processo_ in_esecuzione;
        descrittori[j].contesto= <valori dei registri CPU>;
}

void Ripristino_stato( )
{  int j;
   j = processo_ in_esecuzione;
   <registro-temp>=descrittori[j].servizio.delta_t;
  <registri-CPU>= descrittori[j].contesto ;
}

void Assegnazione_CPU( )
{  int  k=0,j;
        while (coda_processi_pronti[k].primo)==-1)
                k++;
   j=Prelievo (coda_processi_pronti [k]);
   processo_in_esecuzione=j;
}
```

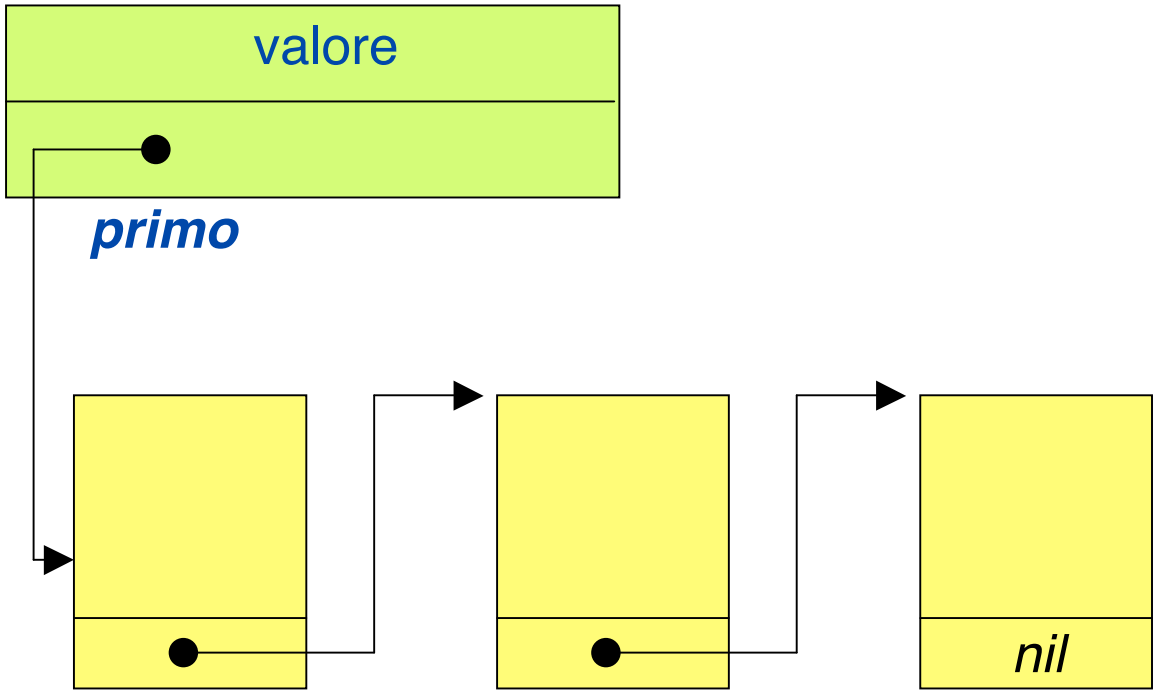
Gestione del temporizzatore

Per consentire la modalità di servizio a *divisione di tempo* è necessario che il nucleo gestisca un *dispositivo temporizzatore* tramite un'apposita procedura *che ad intervalli di tempo fissati*, provveda a sospendere il processo in esecuzione ed assegnare l'unità di elaborazione ad un altro processo.

```
void Cambio_di_Contesto()
{       int j,  k ;
        Salvataggio_stato();
        j = processo_in esecuzione;
        k=descrittori[j].servizio.priorità;
        Inserimento (j,coda_processi_pronti[k]);
        Assegnazione_CPU ( );
        Ripristino_stato ( );
}
```

Semafori

- Un semaforo viene implementato tramite:
  - *Una variabile intera* che rappresenta il suo valore(>=0)
  - *Un puntatore ad una lista* di processi in attesa sul semaforo (*bloccati*)

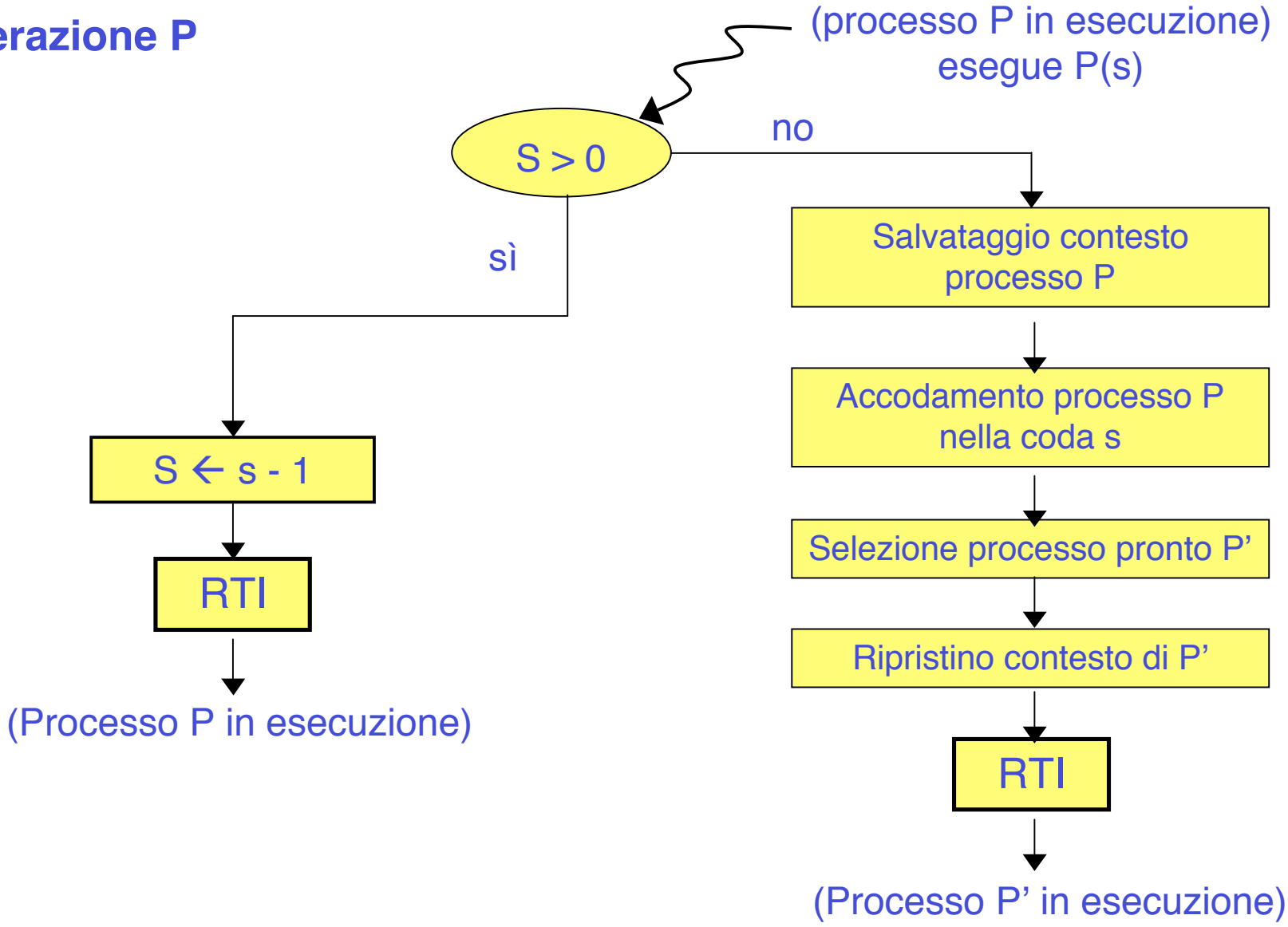


- Se non ci sono processi in coda il puntatore contiene la costante ***nil*** (ad esempio, -1).
- La lista realizza una politica **FIFO** tra i processi sospesi: i processi risultano ordinati secondo il loro tempo di arrivo nella coda associata al semaforo.
- Il descrittore di un processo viene inserito nella coda del semaforo come conseguenza di una primitiva **wait** non passante; viene prelevato per effetto di una **signal**.

```
typedef struct
{   int contatore;
    coda_a_livelli coda;
} descr_semaforo;
```

Semafori: Funzioni di sincronizzazione

Operazione P

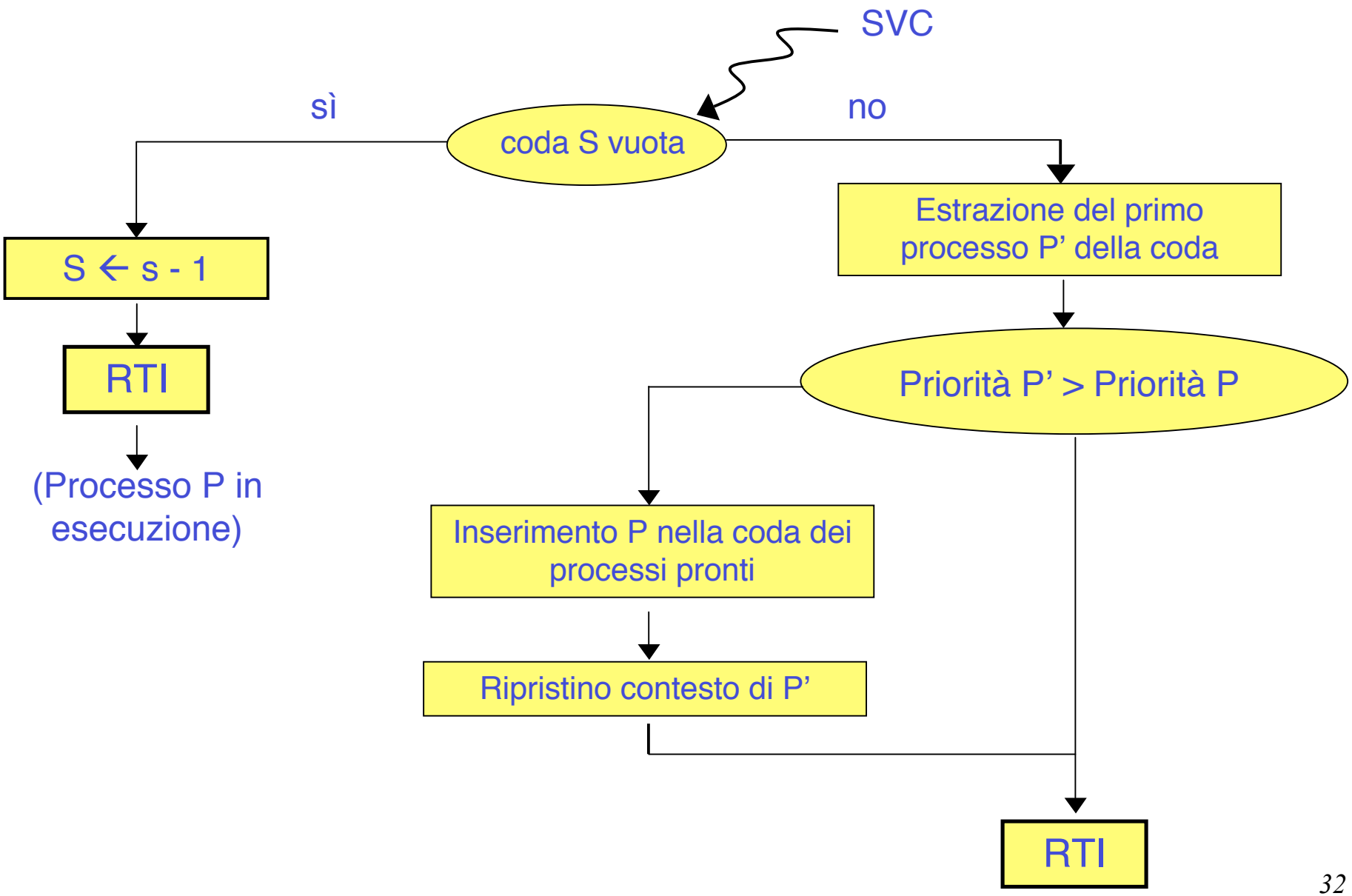


Realizzazione di P

```
/* insieme di tutti i semafori: */
descr_semaforo semafori[num_max_sem];

/*ogni semaforo e` rappresentato dall'indice che lo
   individua nel vettore semafori*/
typedef int semaforo;

void P (semaforo s)
{       int j;int k;
        if (semafori[s].contatore ==0)
        {       Salvataggio_stato();
                j=processo_in_esecuzione;
                k=descrittori[j].servizio.priorità;
                Inserimento(j,semafori[s].coda[k]);
                Assegnazione CPU();
                Ripristino_stato();
        }
        else contatore--;
}
```

Operazione V

chiamata a V(s)



Realizzazione V

```
void V (semaforo s)
{  int j,k,p,q; /*j,k: processi; p,q indici priorità*/
   q=1;
   while (semafori[s].coda[q].primo==-1 && q < min_priorità )
        q++;
   if(semafori[s].coda[q].primo!=-1)
   {    k=Prelievo (semafori[s].coda[q]);
        j= processo_in_ esecuzione;
        p= descrittori[j].servizio.priorità;
        if( p>=q)
                Inserimento (k,coda_processi_pronti[q]);
        else {
                Salvataggio_stato();
                Inserimento(j,coda_processi_pronti[p]);
                processo_in_esecuzione=k;
                Ripristino_stato();
        }
   }
   else semafori[s].contatore ++;
   }
}
```

Meccanismo di passaggio dall'ambiente di nucleo all'ambiente dei processi e viceversa.

- E' costituito dal *meccanismo di interruzioni* (esterne o asincrone, interne o sincrone).In entrambi i casi , al completamento della funzione richiesta, il trasferimento all'ambiente di utente avviene utilizzando il *meccanismo di ritorno da interruzione*.

- Architettura ipotetica: ad ogni processo è associata una pila (*stack*) gestita tramite il registro *stack point*. La pila rappresenta l'area di lavoro del processo e contiene variabili temporanee ed i record di attivazione delle procedure chiamate.

- *Registri* :PC e PS (registro di stato), R1,R2,..Rn, R1',R2'..Rn', SP1, SP1' (registri generali e stack pointer) associati rispettivamente agli ambienti di nucleo e dei processi.

L'esecuzione di una primitiva da parte di P corrisponde all'esecuzione di una istruzione *di tipo SVC:*

1. **Interruzione** di tipo sincrono
2. **Salvataggio di PC e PS** relativi a **P** in cima alla pila del nucleo.
3. **Caricamento in PC e PS** dell'indirizzo della procedura di risposta all'interruzione e di PS del nucleo.
4. **Esecuzione della procedura di risposta** all'interruzione con chiamata alla primitiva di nucleo richiesta (es. P)
5. ***Wait passante***: esecuzione di **ritorno dall'interruzione** che ripristina in PC e PS i valori del processo contenuti nella pila del nucleo.
6. **Wait non passante: salvataggio stato...**

ESTENSIONE AL CASO MULTIPROCESSORE

Primo modello

- Ogni processo può operare su *ogni unità di elaborazione*. Esiste *un'unica copia del nucleo* nella memoria comune.

- *Accesso esclusivo* alle sue *strutture dati* può essere ottenuto con le primitive *lock* e *unlock*.

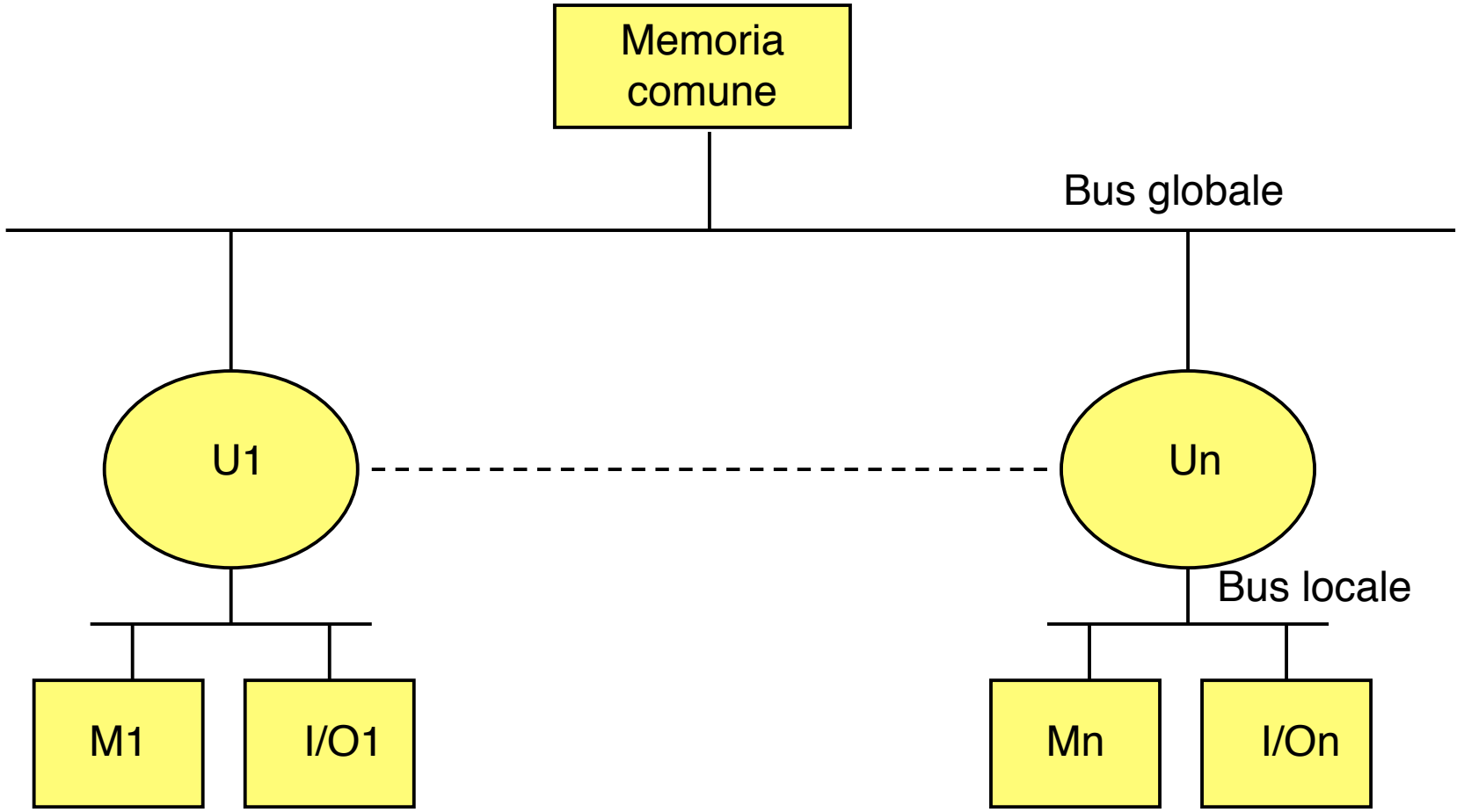
- ***Limitazione del grado di parallelismo***, escludendo a priori ogni possibilità di esecuzione contemporanea di più funzioni del nucleo, ad esempio *due P su due semafori diversi*.

- Un *maggior grado di concorrenza* può essere ottenuto proteggendo

  - **i singoli semafori**
  - **la coda dei processi pronti**

  tramite variabile logiche *x* ad essi associate ed usate dalle *lock* e *unlock* (soluzione hardware al problema della mutua esclusione).

- In questo caso due operazioni P su semafori diversi
  - possono operare in *modo contemporaneo* se non risultano sospensive.
  - In caso contrario, *vengono sequenzializzati solo gli accessi alla coda dei processi pronti.*


Memoria
comune
Bus globale
U1
Un
Bus locale
M1
I/O1
Mn
I/On

- L'esecuzione della *V* può portare all'attivazione di un processo con *priorità superiore* ad almeno uno dei processi in esecuzione.

- Il nucleo deve provvedere a *revocare l'unità di elaborazione* al processo con *priorità più bassa* ed assegnarla al processo riattivato.

- Occorre che il nucleo mantenga l'informazione del processo in esecuzione a *più bassa priorità* e *del nodo fisico* su cui opera.

- Inoltre è necessario un *meccanismo di segnalazione* tra le unità di elaborazione.

Siano:

Pi il processo che esegue la *V* operando su *Ui*

Pj il processo riattivato

Pk il processo a più bassa priorità che opera su *Uk*

Il nucleo attualmente eseguito da *Ui* invia un segnale di interruzione a *Uk*.

Uk, utilizzando le funzioni del nucleo, inserisce Pk nella coda dei processi pronti e *mette in esecuzione il processo Pj*.

Osservazioni:

- Anche se un processo può essere eseguito da ogni processore, può risultare *più efficiente* assegnarlo ad un *determinato processore*.

- I processori possono accedere più rapidamente alla loro memoria privata piuttosto che a quella remota (*sistemi con accesso alla memoria non uniforme*).

- Il processo dovrebbe essere eseguito sul processore la cui memoria privata *contiene il suo codice*.

  I processori hanno *memorie cache o TLB* (memoria virtuale). Il processo dovrebbe essere assegnato al processore sul quale era stato precedentemente eseguito.

  Lista dei processi pronti *per ogni processore*. Problema del ***load-balancing***

ESTENSIONE AL CASO MULTIPROCESSORE

Secondo modello:

- Si basa sull'ipotesi che l'insieme di processi sia partizionabile in tanti sottoinsiemi *(**nodi virtuali**)* *lascamente connessi*, cioè con un ridotto numero di interazioni reciproche.

- Ciascun ***nodo virtuale*** è **assegnato ad un *nodo fisico***. Tutte le informazioni relative *al nodo virtuale* come descrittori dei processi, semafori locali, code dei processi pronti, vengono allocate sulla ***memoria privata*** del nodo fisico.

- Se nelle memorie private vengono allocate anche le *funzioni del nucleo*, tutte le interazioni locali al *nodo virtuale* possono avvenire **indipendentemente** e **concorrentemente** a quelle di altri nodi virtuali.

- La memoria comune dovrà contenere tutte le informazioni relative ai ***semafori*** utilizzati da processi appartenenti a *nodi virtuali differenti*.

- Ogni semaforo sarà protetto da un *indicatore x* usato da *lock(x) e unlock(x)*.

- ***Rappresentante del processo.*** Insieme minimo di informazioni sufficienti per identificare sia il *nodo fisico* su cui il processo opera, sia il *descrittore* contenuto nella memoria privata del processore.

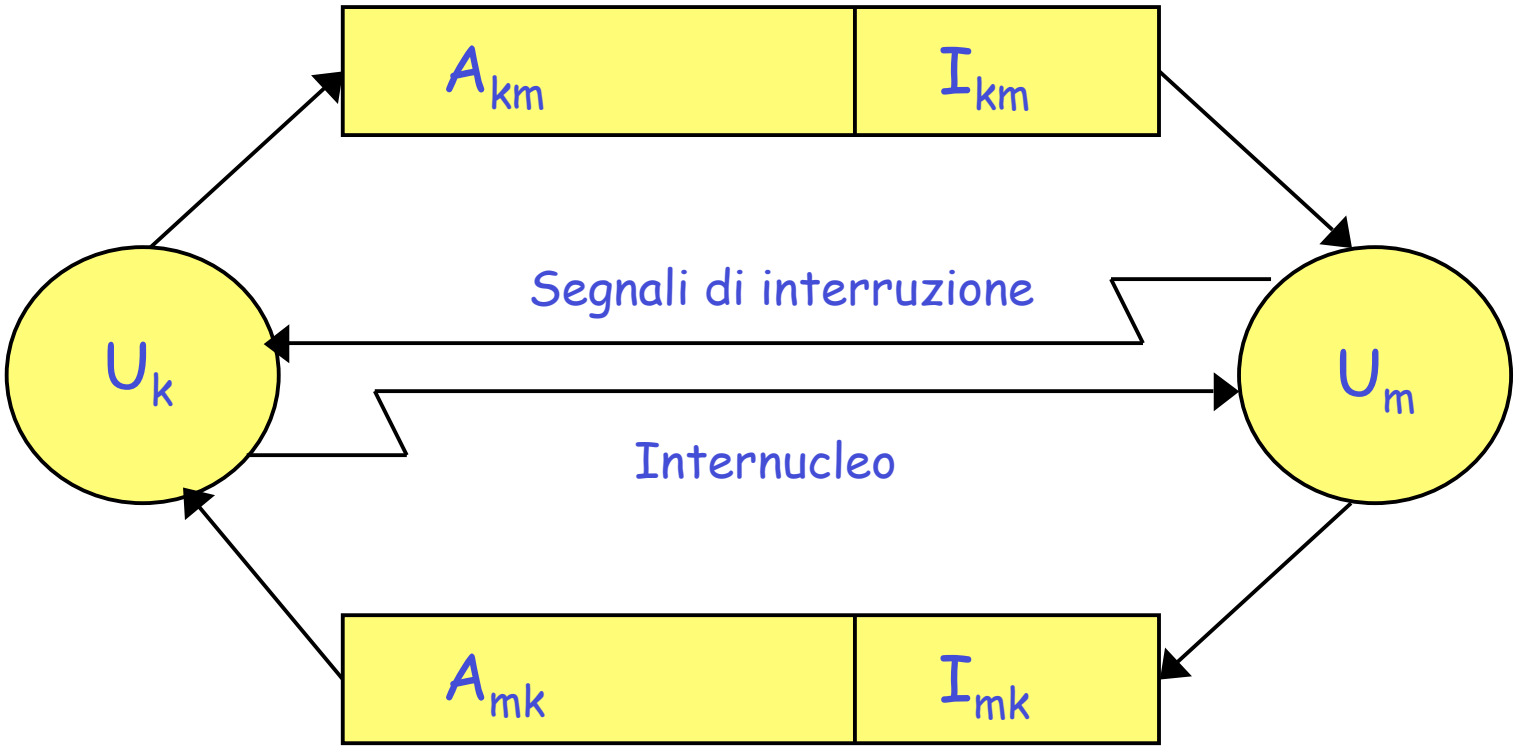
Esecuzione di una *P sospensiva (su un semaforo condiviso tra nodi virtuali)*: il nucleo Nk del nodo fisico k-esimo sul quale opera il processo Pi, provvede a:

- sospendere sul semaforo il *rappresentante di Pi*.
- inserire il descrittore di *Pi* in una apposita coda nella memoria privata del nodo fisico k-esimo.

Esecuzione di una *V* su un semaforo (con rappresentanti in coda) da parte di un processo *Pi* che opera sul *nodo Uk*:

- il rappresentante del processo Pj che appartiene al *nodo Um* viene eliminato dalla coda associata al semaforo (es. FIFO);
- il nucleo *Nk* provvede a ***comunicare*** l'identità del processo Pj risvegliato al nucleo *Nm;*
- *Nm* provvede a mettere in esecuzione il processo *Pj* o ad inserirlo nella coda dei processi pronti nella sua memoria locale.

Comunicazione tra nuclei



- *Akm* , *Amk* sono aree di comunicazione tra i nodi *Uk* e *Um*

- *Nk* inserisce in *Akm* l'identità del processo *Pj* risvegliato e lancia un'*interruzione* a *Nm.*

- Rispondendo a questa interruzione *Nm* provvede a prelevare le informazioni contenute in *Akm* ed a eseguire le operazioni già descritte.
  Analogamente per le comunicazioni tra *Nm* e *Nk.*

- *Ikm* e *Imk* sono indicatori che servono per sincronizzare le operazioni dei due nuclei.

- Se *Ikm* ha valore 1 significa che l'informazione contenuta in *Akm* non è ancora stata prelevata da *Nm*; se ha valore 0 significa che *Akm* è disponibile per un ulteriore messaggio da *Nk* a *Nm*

Realizzazione P (caso multiprocessore, modello 2)

```
void P(semaforo sem):
{     if (<sem appartiene alla memoria privata>)
            <come nel caso monoprocessore>
      else
      {     lock(x);
            < esecuzione della P con eventuale
            sospensione del rappresentante del
            processo  nella coda di sem>;
            unlock(x);
      }
}
```

Realizzazione V (caso multiprocessore, modello 2)

```
void V (semaforo sem)
{ if (<sem appartiene alla memoria privata>)
        <caso monoprocessore>
  else {
        lock(x)
        if  (<la coda non è vuota>)
           if (<il processo appartiene al nodo del segnalante>)
              <caso monoprocessore>;
           else {
             <si elimina il processo dalla coda>;
             <si determina l'area di comunicazione con il nodo
             cui il processo appartiene>;
             if (<area occupata>) <si aspetta>;
             else <si inserisce nell'area l'identificatore del
             processo riattivato e si pone l'indicatore a 1>;
             <si invia interrupt al nodo cui appartiene il
             processo>;
        else <si incrementa il valore del semaforo>;
        unlock(x)
  }
}
```