Indice

- Modelli di comunicazione tra processi (IPC)
- Strumenti di programmazione per IPC

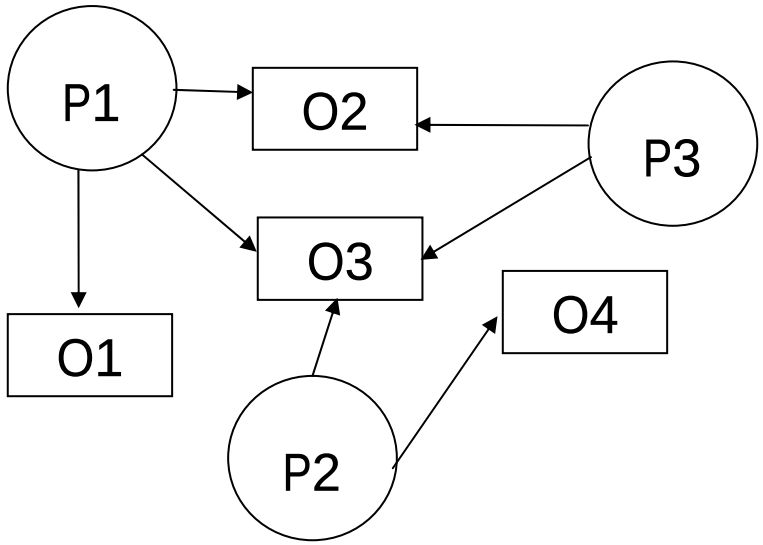
- Interazione tra processi
  - Interferenza e Mutua Esclusione
  - Problemi di IPC: Produttore-Consumatore

Modelli di IPC

- Processo: programma in esecuzione
- Due processi possono o devono interagire:
  - Per trasferire informazioni
  - Per garantire corretta sequenza delle operazioni
  - Senza disturbare altri processi durante attività critiche
- Interazione tra processi:
  cooperazione, competizione, interferenza
- Modelli:
  - Ambiente Globale (WorkStation, PC)
  - Scambio di Messaggi (Sistemi distribuiti, Cluster)

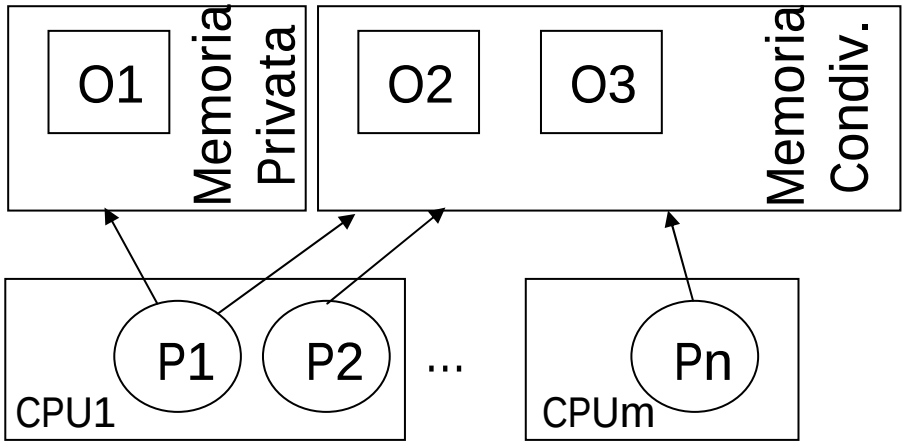
Modello ad ambiente globale 1

- Il sistema è visto come un insieme di:
  - Processi (Pi) con diritti d'accesso alle risorse (→)
  - Oggetti o Risorse (Oj)
    - Private
      Es: O1, O4
    - Condivise
      Es: O2, O3



Modello ad ambiente globale 2

- Astrazione per *Sistemi Multiprogrammati*
- Costituiti da uno o più processori che hanno accesso ad una memoria comune
- Interazioni in memoria condivisa

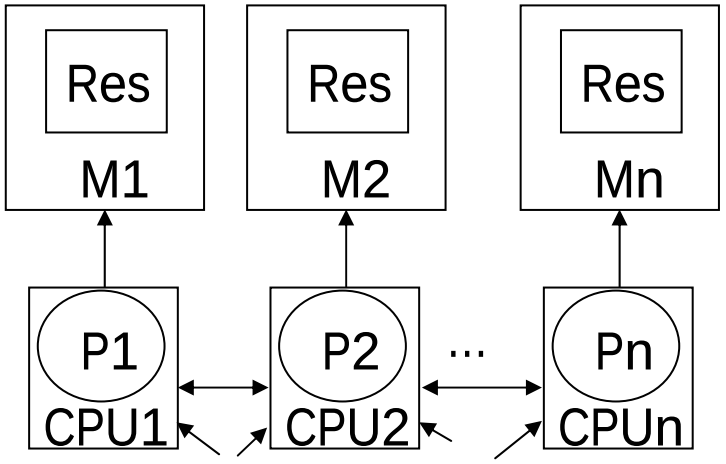


Modello a scambio di messaggi 1

- Sistema: insieme di processi ciascuno operante in ambiente locale (protetto)
  - Le interazioni (comunicazione, sincronizzazione) avvengono tramite scambio di messaggi
- Una risorsa appartenente a Pi non è accessibile direttamente da un altro processo Pj

Modello a scambio di messaggi 2

- Astrazione per *Sistemi Distribuiti* (privi di memoria comune)
- Ciascun processore ha associata una memoria privata
- Il trasferimento dei messaggi può avvenire:
  - via rete
  - tramite memoria condivisa



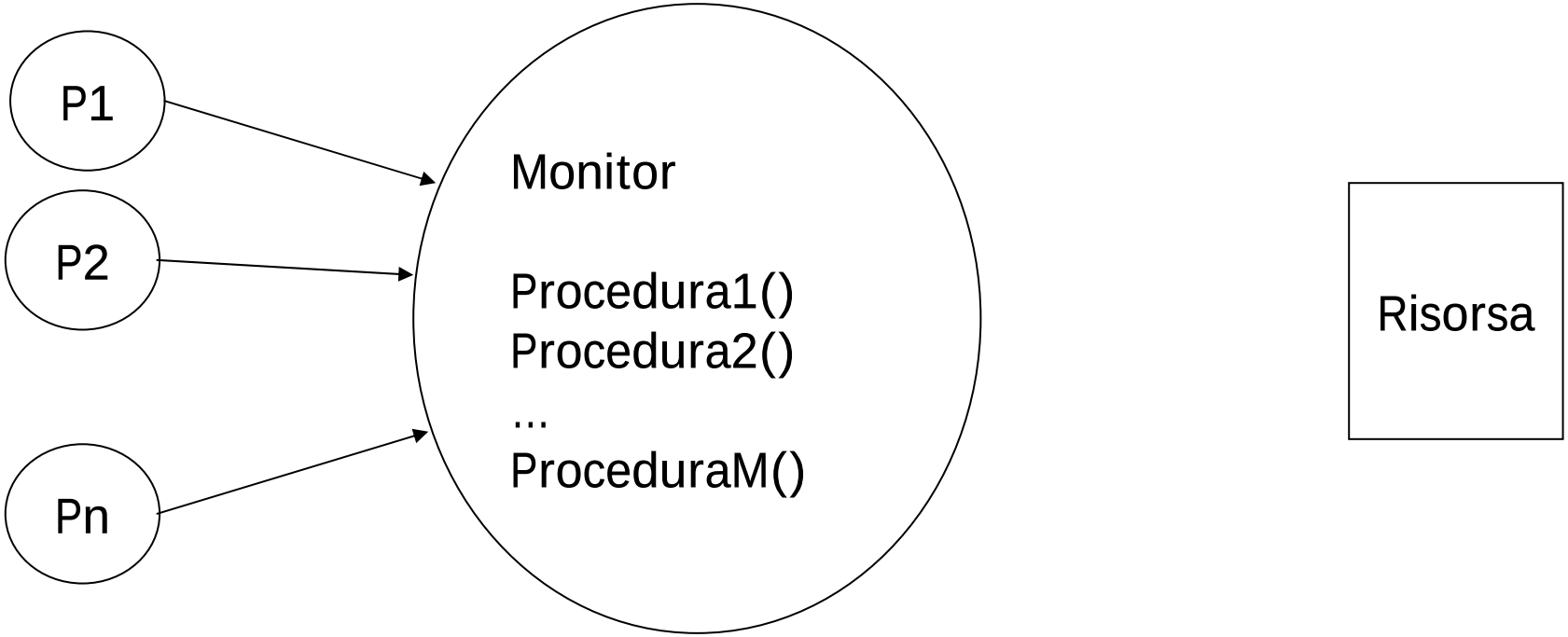
Strumenti di Programmazione

- Indipendenti dal linguaggio
- Specifici del linguaggio

- Ambiente Globale
  - Monitor (Brinch Hansen, Hoare 1973)
  - Semafori e Primitive di sincronizzazione (Dijkstra, 1965)
  - Altri strumenti
- Scambio di Messaggi
  - Send(m), Receive(m)

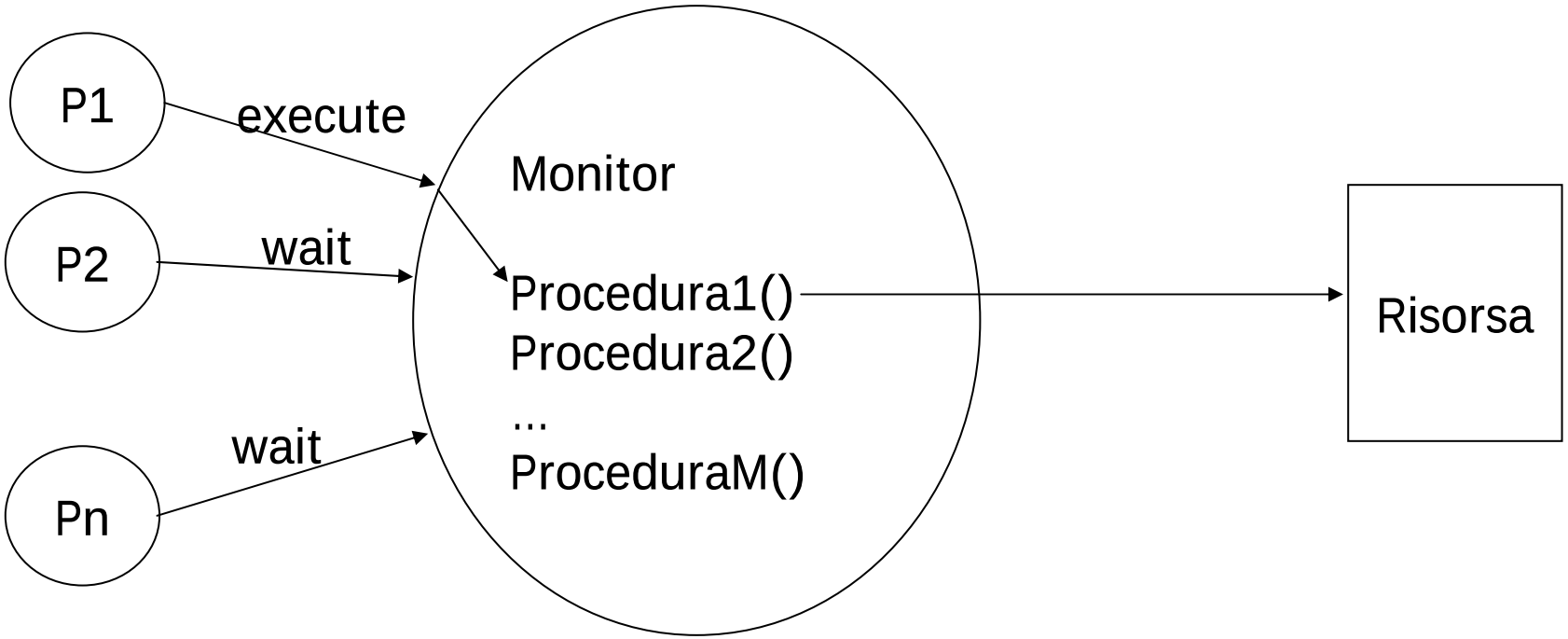
Monitor

- Virtualizzazione della risorsa
- Accesso tramite un processo gestore



Monitor

- Un processo alla volta accede alla risorsa



Interazione tra processi: cooperazione

- Interazione prevedibile e desiderata (insita nella logica del programma)
- Prevede scambio di informazioni
  - Segnale temporale (Sync)
  - Messaggi (Com+Sync)
- Sincronizzazione diretta o esplicita (gestita dal programmatore)

Interazione tra processi: competizione

- Interazione prevedibile e NON desiderata
- Necessaria: macchina concorrente dispone di risorse limitate
  - Competizione per l'uso di risorse comuni che non possono essere usate contemporaneamente
- Sincronizzazione indiretta o implicita (gestita dal SO)

Interazione tra processi: interferenza

- Interazione NON prevedibile e NON desiderata
  - Inserimento nel programma di interazioni non richieste dalla natura del problema
    - Es: P1 chiama P2 senza necessità e lo rallenta
  - Erronea soluzione a problemi di interazione (cooperazione, competizione) necessari per il corretto funzionamento del programma
    - Es: P1 chiama P2 ma al momento sbagliato
- Dipende dalla velocita' relativa dei processi: risultati "dipendenti dal tempo"
- Debug molto complesso!

Processi Interferenti

- Problema: eliminare le interferenze
  - L'eliminazione di alcune interferenze risulta semplificata se la macchina concorrente fornisce meccanismi di protezione degli accessi
  - Per evitare interferenze dovute a interazioni previste ma programmate in modo errato è opportuno adottare tecniche di *multiprogrammazione strutturata*
    (Es: strutture formali opportune come i semafori)

Interferenza - Esempio 1

Pi	Pj
...	...
Count=Count+1	Count=Count+1
...	...

Count e' condivisa tra Pi e Pj

Interferenza - Esempio 1

Count = Count+1

```
LOAD    A, Count
INCR    A
STORE   Count, A
```

Interferenza - Esempio 1

LOAD	A, Count	(Pi)	
INCR	A	(Pi)	
STORE	Count, A	(Pi)	
LOAD	A, Count		(Pj)
INCR	A		(Pj)
STORE	Count, A		(Pj)

Funzionamento corretto

Interferenza - Esempio 1

LOAD	A, Count	(Pi)	
LOAD	A, Count		(Pj)
INCR	A		(Pj)
STORE	Count, A		(Pj)
INCR	A	(Pi)	
STORE	Count, A	(Pi)	

Contesto singolo per ogni processo !

Esempio 1

Pi	Pj
...	...
Count=Count+1	Count=Count+1
...	...

Count viene incrementata una sola volta!

Esempio 2

- v e' una variabile condivisa tra P e Q
  - Processo P: incrementa v di 1
  - Processo Q: stampa v e la azzera

```
P                    Q
...                  ...
v:=v+1               print v
...                  v := 0
                     ...
```

Esempio 2

- Le istruzioni di P e Q possono mescolarsi arbitrariamente e dar luogo a diverse possibili sequenze di esecuzione

v:=v+1	(P)	print v	(Q)	print v	(Q)
print v	(Q)	V:=0	(Q)	v:=v+1	(P)
V:=0	(Q)	v:=v+1	(P)	V:=0	(Q)
V+1, V=0		V, V=1		V, V=0	

Il risultato dipende dal tempo!

Mutua Esclusione

- Corsa Critica: situazione in cui due o più processi interferiscono
- Sezione Critica (SC): parte di un programma che accede a dati condivisi
  - Almeno un processo deve scrivere
  - La lettura e' sicura solo se atomica
    - Non usare strutture dati complesse!
  - Due SC appartengono alla stessa classe se operano sugli stessi dati
  - Sezioni Critiche appartenenti alla stessa classe devono escludersi mutuamente nel tempo

Mutua Esclusione

- I processi che accedono a SC della stessa classe devono essere eseguiti in Mutua Esclusione
- L'obiettivo è evitare che più di un processo alla volta acceda a dati condivisi

ME - Esempio 1

- P1 e P2 utilizzano la stessa stampante (un solo processo alla volta può averne il controllo)

```
RICHIESTA                    RILASCIO
                             
...                          ...
repeat until libera;         libera := TRUE;
libera := false;             ...
...
```

```
STATO INIZIALE
libera := TRUE;

Pi:
...
RICHIESTA
... Utilizzo ...
RILASCIO
...
```

ME - Esempio 1

- P1 e P2 utilizzano la stessa stampante (un solo processo alla volta può averne il controllo)

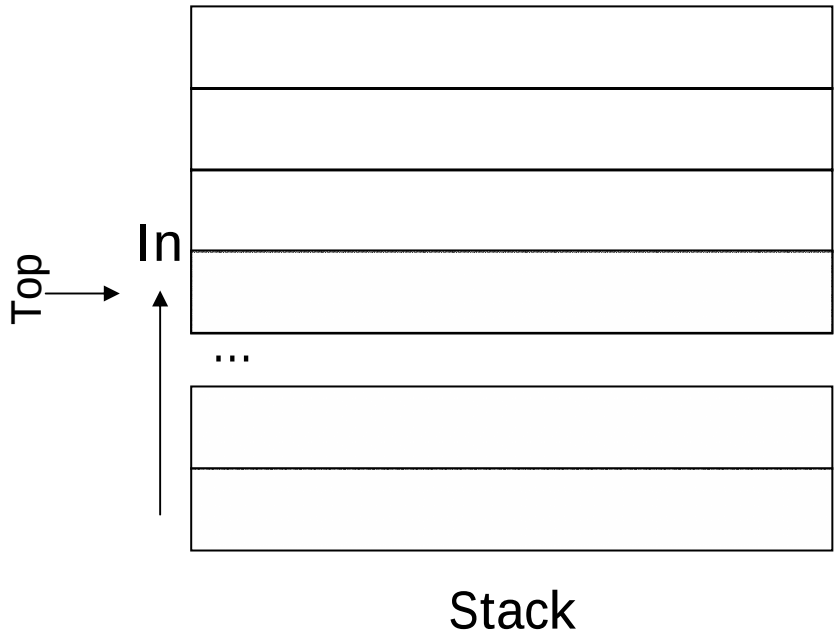
RICHIESTA	RILASCIO	STATO INIZIALE
...	...	libera := TRUE;
repeat until libera;	libera := TRUE;	Pi:
libera := false;		...
...		RICHIESTA
		... Utilizzo ...
		RILASCIO
		...

T0: repeat until libera;	(P1)	
T1: repeat until libera;		(P2)
T2: libera:=false;	(P1)	
T3: libera:=false;		(P2)

Stampante assegnata ad entrambi!

ME - Esempio 2

Due processi hanno accesso ad uno stack per depositare e prelevare messaggi



Inserimento(y)	Prelievo(x)
...	...
Top:=Top+1;	x:=Stack[Top];
Stack[Top]:=y;	Top:=Top-1;
...	...

ME - Esempio 2

- L'esecuzione contemporanea delle due procedure può portare ad un uso scorretto della risorsa
- Es: P1 inserisce mentre P2 preleva:

```
T0: Top := top + 1      P1
T1: x=Stack[Top]        P2
T2: Top := Top - 1      P2
T3: stack[Top] := y     P1
```

ME - Esempio 2

- L'esecuzione contemporanea delle due procedure può portare ad un uso scorretto della risorsa
- Es: P1 inserisce mentre P2 preleva:

T0: Top := top + 1	P1
T1: x=Stack[Top]	P2 (x indefinito)
T2: Top := Top - 1	P2
T3: stack[Top] := y	P1 (sovrascritto ultimo elemento)

Soluzioni Storiche

- Definizione tempi lasciata al programmatore
  - Concorrenza per modo di dire ...
- Disabilitare le interruzioni
  - L'utente disabilita le IRQ ...
- Variabili di Lock
  - Si sposta il problema della ME ...
- Strumenti di Sincronizzazione

Mutua Esclusione

- Problema:
  I processi rimangono per lo più impegnati a svolgere i propri compiti e di tanto in tanto incontrano una sezione critica

Mutua Esclusione

- Soluzione:
  la ME deve avere queste caratteristiche:
  - Due processi non devono mai essere insieme in sezioni critiche della stessa classe
  - Nessuna ipotesi sulla velocità relativa dei processi
  - Nessun processo fuori dalla SC deve bloccarne altri
  - Nessun processo deve attendere indefinitamente prima di entrare in SC

Mutua Esclusione tramite HW: TSL – Test and Set Lock

- TSL: Legge e pone in un registro il contenuto di una parola di memoria quindi scrive un valore non nullo nella stessa parola
  - Lettura e scrittura della parola sono indivisibili
  - Nessun altro processore puo' accedere alla locazione di memoria finche' TLS non e' terminata
- TSL blocca l'accesso al bus di memoria alle altre CPU finche' non e' terminata

Mutua Esclusione tramite HW: TSL – Test and Set Lock

Lock()

enter_region:

```
TSL register, lock ; copia il lock nel registro e pone lock=1
cmp register, #0   ; il lock era a 0?
jne enter_region   ; se era diverso da 0 c'era un lock: cicla
ret                ; ritorna al chiamante: entra in SC
```

UnLock()

```
move lock, #0      ; metti 0 in lock (atomica)
ret                ; ritorna al chiamante
```

Mutua Esclusione tramite Primitive Lock() e UnLock()

```
Lock(x);
    <Sezione Critica>
UnLock(x);
```

- Problemi:
  - Attesa attiva (Busy Wait): mentre un processo è in SC tutti gli altri sono in Busy Wait
  - Inversione di priorità

Inversione di Priorità

- Due Processi H (Alta priorità) e L (Bassa priorità)
  - L e' in sezione critica
  - H diventa "Pronto" (es: fine operazione di I/O)
  - H viene schedulato. Entra nella Lock(): Busy Wait
  - L non viene schedulato mentre H e' in esecuzione
  - L non esce da SC, H non esce dalla Lock()

Starvation: H ha priorità alta ma non viene eseguito

Semafori

- Semaforo:
  - variabile intera non negativa ($s \geq 0$) con valore iniziale $s=s_0$
  - Al semaforo è associata una coda di attesa Qs
  - Qs contiene info sui processi in attesa sul semaforo

Semafori

- Sono ammesse solo 2 primitive Wait(s) e Signal(s)
  - Realizzate tramite chiamate al S.O.
  - Eseguite in modo Monitor (l'accesso ad s è in ME)

Wait()

```
Begin
   if s = 0 then
      <Processo sospeso>
      <Descrittore del Processo corrente inserito in Qs>
   else
      s := s - 1
End
```

- Può essere passante (s > 0)
- Oppure bloccante (s = 0). In questo caso:
  - Si verifica un Context Switch
  - Il processo corrente viene sospeso

Signal()

```
Begin
   if <Esiste un processo in Qs> then
      <Descrittore del Processo Rimosso da Qs>
      <Stato del Processo Rimosso modificato in Pronto>
   else
      s := s + 1
End
```

- Sempre passante
- Non modifica lo stato del processo che la esegue
- La riattivazione del processo sospeso avviene tramite politica FIFO (Fairness, No Starvation)

ME Tramite Semaforo Wait e Signal

- Ad ogni classe di SC si associa un semaforo
- Prologo = wait(), Epilogo = signal()
- ES: A, B SC $\in$ Stessa Classe, S Semaforo ($S_0 = 1$)

Processo P1	Processo P2
...	...
Wait(s)	Wait(s)
<Sezione Critica A>	<Sezione Critica B>
Signal(s)	Signal(s)
...	...

ME Tramite Semaforo Wait e Signal

- La natura primitiva di Wait e Signal assicura la proprietà di mutua esclusione:
  - per qualunque numero di processi
  - qualunque sia la velocità relativa dei processi
  - nessun processo rimane indefinitamente in attesa (un processo non si può riappropriare della SC che ha appena liberato se altre richieste sono pendenti)
- Sono risolti i problemi di Busy Wait

Indivisibilità di Wait e Signal

- Occorre garantire che l'azione di analisi e modifica del semaforo non sia separata dall'azione di sospensione del processo

Esempio:

Sia inizialmente s := 0

T0: if s = 0	P1	Wait()
T1: s := s + 1	P2	Signal()
T2: <Sospensione>	P1	Wait(): Processo sospeso su s = 1

Indivisibilità di Wait e Signal

- Wait() e Signal() sono considerate SC brevi
- Indivisibilità ottenuta mediante Lock() e UnLock() sia per Mono- che Multiprocessore
  - MonoProcessore:
    inibizione interruzioni = blocco scheduling
    Aumento Prestazioni:
    non vengono eseguite le attese attive
    degli altri processi

Wait atomica

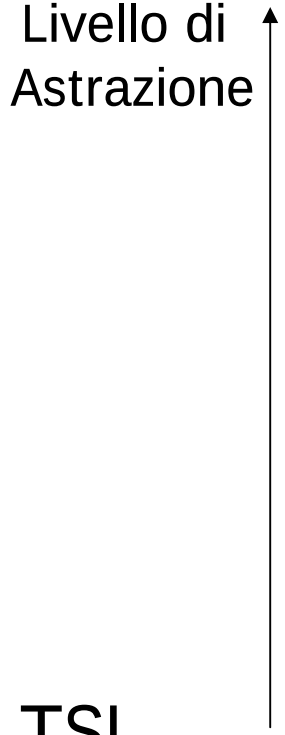
```
Wait(Semaphore):
        begin;
                <Disabilitazione Interruzioni>;
                lock(x);
                <Codice della Wait>;
                unlock(x);
                <Abilitazione delle Interruzioni>;
        end;
```

Signal atomica

```
Signal(Semaphore):
        begin;
                <Disabilitazione Interruzioni>;
                lock(x);
                <Codice della Signal>;
                unlock(x);
                <Abilitazione delle Interruzioni>;
        end;
```

Livelli di Sezioni Critiche

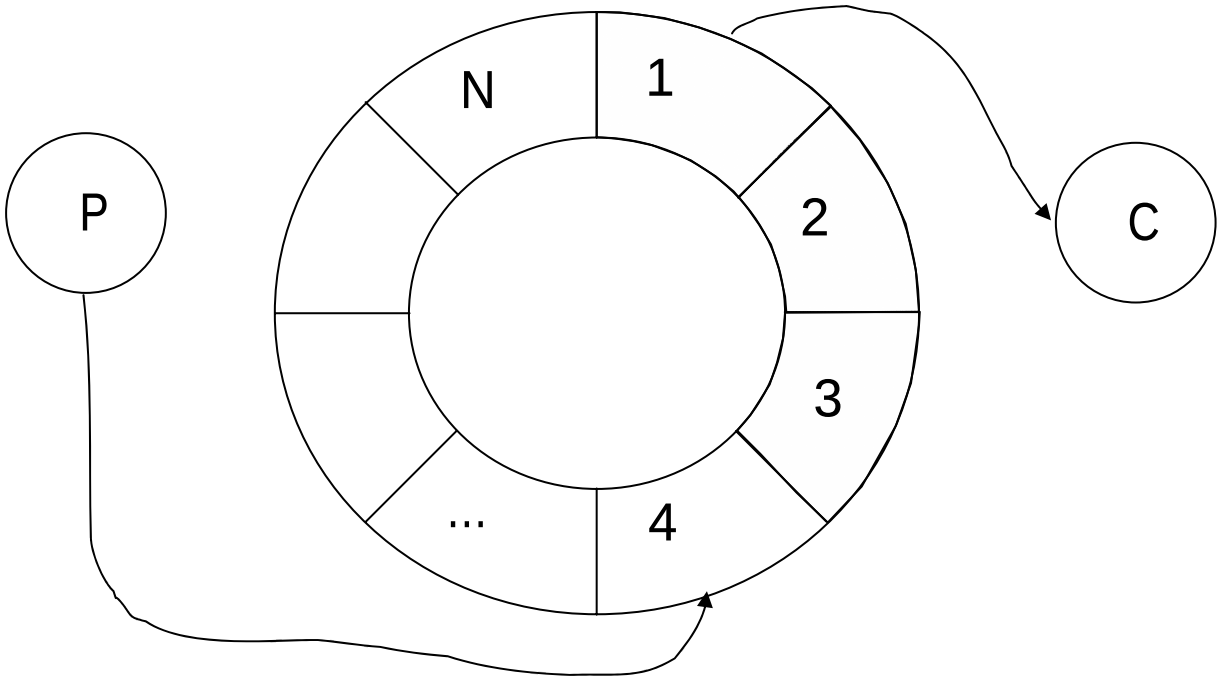
- Livello Utente:
  - SC: Programma
  - ME: Wait(), Signal()
- Livello SO:
  - SC: Wait(), Signal()
  - ME: Lock(), Unlock()
- Livello HW:
  - Lock(), Unlock() realizzate tramite TSL



Problema del Produttore e Consumatore

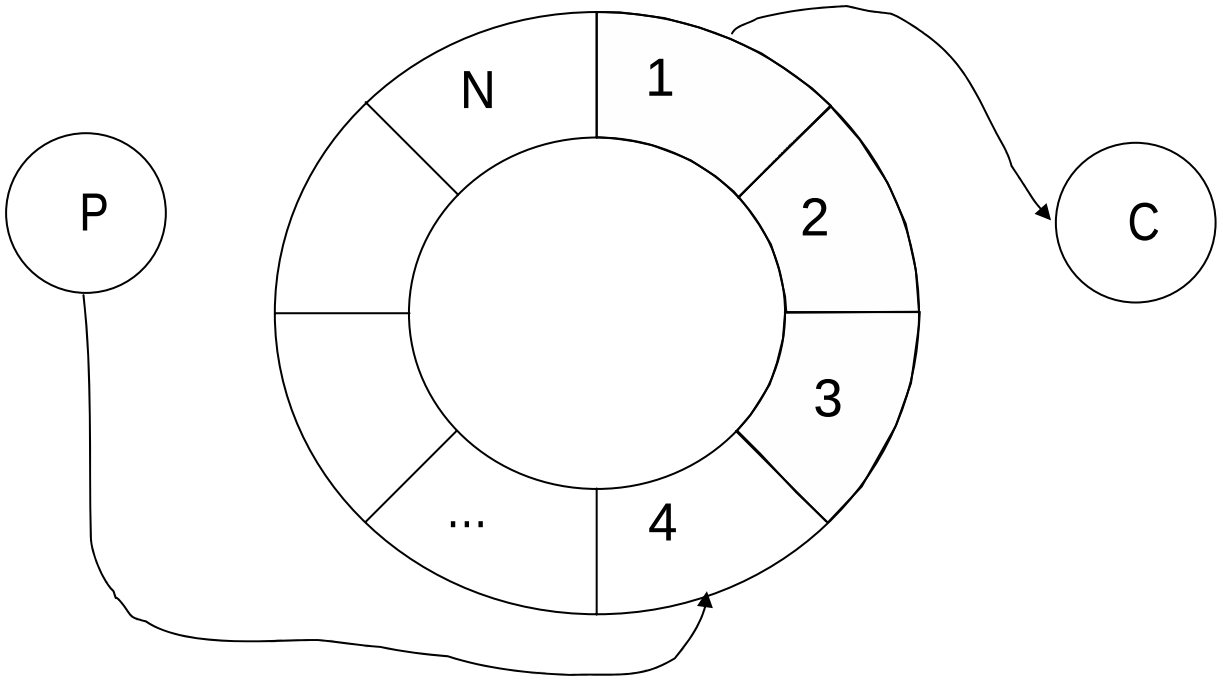
Buffer condiviso e limitato

Possono accedervi: P in scrittura e C in lettura



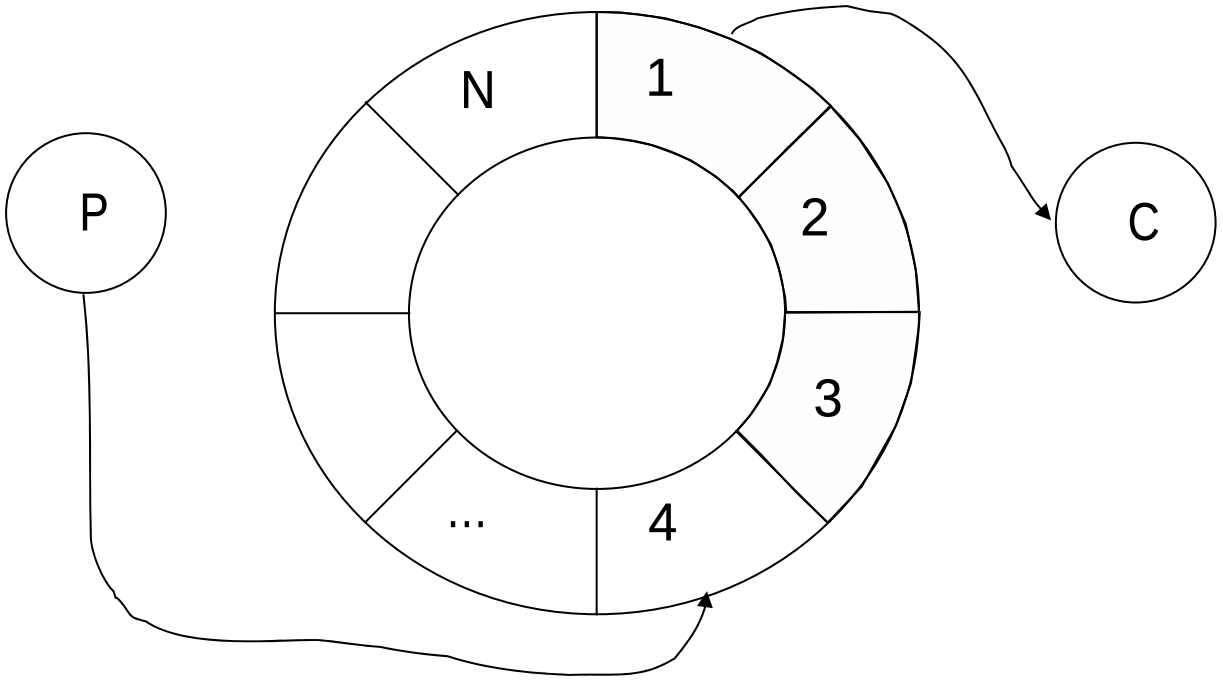
Problema del Produttore e Consumatore

P NON può inserire un messaggio se il Buffer è pieno



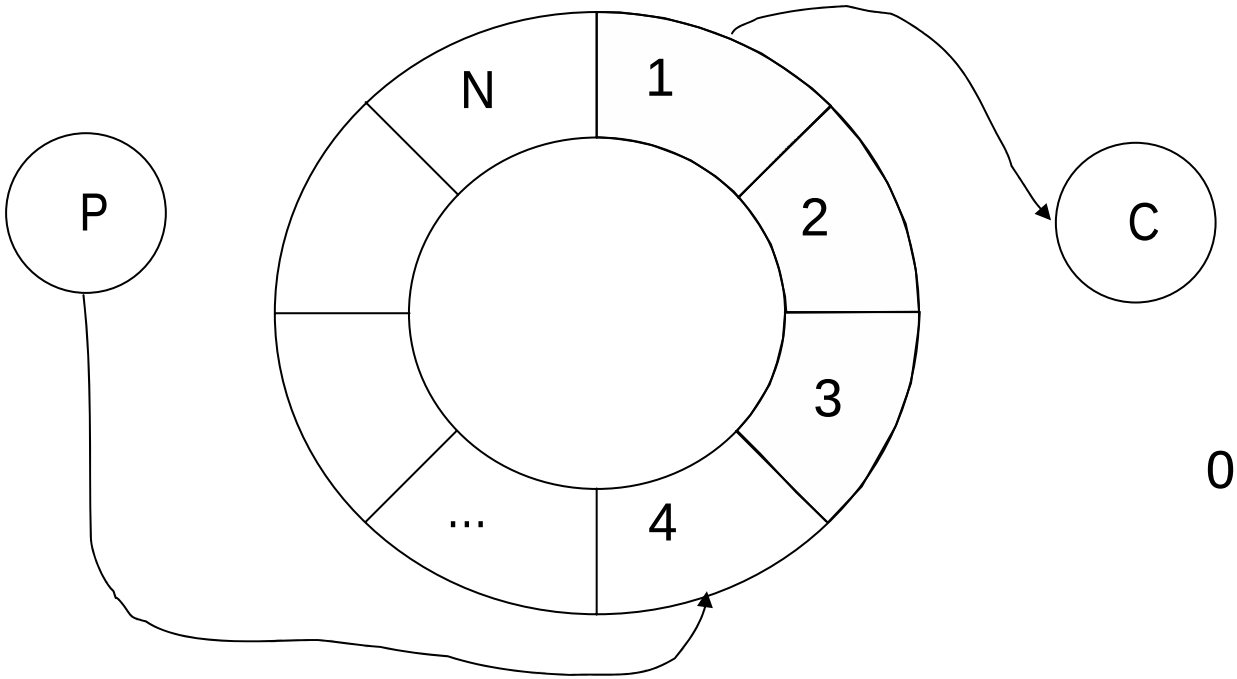
Problema del Produttore e Consumatore

C non può prelevare se Buffer è vuoto



Problema del Produttore e Consumatore

d = numero dei messaggi depositati
e = numero dei messaggi estratti
N = dimensione del buffer



0<e<d<N

Problema del Produttore e Consumatore

La Soluzione richiede due Semafori:

- "Messaggio disponibile" MsgDisp (S_0=0)
- "Spazio Disponibile" SpzDisp (S_0=N)

```
Produttore (P)
Begin
  repeat
     <Produzione Msg>
     Wait(SpzDisp)
     <Deposito Msg>
     Signal(MsgDisp)
  forever
end
```

```
Consumatore (C)
Begin
  repeat
     Wait(MsgDisp)
     <Prelievo Msg>
     Signal(SpzDisp)
     <Consumazione Msg>
  forever
end
```

Problema del Produttore e Consumatore

- È una soluzione simmetrica: non privilegia né P né C
- P e C possono operare in parallelo su messaggi diversi
- P e C NON possono operare contemporaneamente sullo stesso messaggio
  - Buffer Pieno (P bloccato dalla wait)
  - Buffer Vuoto (C bloccato dalla wait)