gestione della memoria fisica

strategia complessiva di gestione della memoria fisica – 1

- comportamento (iniziale) di Linux nella gestione della memoria fisica
  - una certa quantità di memoria viene allocata inizialmente al *Sistema Operativo* e non viene mai deallocata
  - le eventuali richieste di *memoria dinamica* da parte *del SO* stesso vengono soddisfatte con la massima priorità
  - quando un *processo* richiede memoria, questa gli viene allocata con liberalità, cioè senza particolari limitazioni
  - tutti i dati letti dal disco vengono conservati indefinitamente, in aree di memoria chiamate *disk cache (buffer)* e di cui la *page cache* fa parte, per essere eventualmente riutilizzati
- questo comportamento può anche durare a lungo, ma a un certo punto la memoria RAM disponibile può ridursi e risulta necessario richiedere interventi di riduzione delle pagine occupate (**page frame reclaiming**)

strategia complessiva di gestione della memoria – 2

- una pagina viene **scaricata** se vengono svolte le operazioni seguenti
  - se la pagina è stata letta da disco e non è stata mai modificata, la pagina viene *resa disponibile* per un uso diverso
  - se la pagina è stata **modificata**, cioè se il suo **Dirty bit** è posto a 1, prima di rendere la pagina *disponibile* per altri usi la pagina va *scritta su disco*
- la deallocazione applica i seguenti tipi di azioni nell'ordine indicato (vedi anche *sperimentazione comportamento complessivo* e comando **`free`**)
  1. le pagine di page cache non utilizzate dai processi vengono scaricate; se questo non è sufficiente
  2. alcune pagine utilizzate dai processi vengono scaricate; se anche questo non è sufficiente
  3. un processo viene eliminato completamente (killed)
- il sistema deve intervenire prima che la riduzione della RAM sotto una soglia minima renda impossibile qualsiasi intervento, mandando il sistema in blocco

allocazione della memoria fisica

- l'unità di base per l'allocazione della memoria è la pagina, ma
  - Linux cerca di allocare blocchi di pagine contigue
  - e di mantenere la memoria il meno frammentata possibile
  - *approfondimento*: ordine dei blocchi e *Buddy System*

- la paginazione permette di operare su uno spazio virtuale continuo anche se le corrispondenti pagine fisiche non lo sono (e quindi l'allocazione per blocchi può sembrare non utile), ma per esempio
  - la memoria è acceduta anche dai canali DMA in base a indirizzi fisici, non virtuali; se un buffer del DMA supera la dimensione della pagina deve essere costituito da pagine contigue
  - la rappresentazione della RAM libera risulta più compatta

deallocazione della memoria fisica

- la memoria richiesta dai processi e dalle disk cache tende a crescere continuamente
  - quella destinata ai *processi* viene rilasciata quando un processo termina
    - ✓ cresce a causa dell'aumento del numero di processi o della crescita delle pagine allocate a ogni processo
    - ✓ ma può decrescere spontaneamente (exit e rilascio esplicito di memoria)
  - quella destinata alle *disk cache* cresce sempre

- la quantità di memoria libera può quindi scendere a un livello critico (low memory)
  - interviene **PFRA** (**Page Frame Reclaiming Algorithm**)
  - dato che PFRA ha bisogno di allocare memoria per il proprio funzionamento, per evitare che il sistema raggiunga una saturazione della memoria e vada in crash l'attivazione di PFRA deve avvenire prima di tale momento
  - PFRA cerca di riportare il numero di pagine libere a un livello che chiameremo **maxFree**

scomposizione del problema di Page Frame Reclaiming

il problema complessivo può essere scomposto in

1. **scelta delle pagine da liberare**, che a sua volta può essere suddiviso in
    1. determinazione di ***quando*** e ***quante*** pagine è necessario liberare
    2. determinazione di ***quali*** pagine liberare; il meccanismo utilizzato da PFRA per scegliere quali pagine liberare si basa sui principi di LRU (scelta delle pagine non utilizzate da più tempo) e può essere a sua volta suddiviso in due sottoproblemi
        1. ***mantenimento dell'informazione*** relativa all'***accesso*** alle pagine
        2. ***scelta delle pagine meno utilizzate*** in base a tale informazione
2. il meccanismo di ***swapping***, cioè l'effettivo scaricamento da memoria (***swap_out***) delle pagine con liberazione della memoria stessa, e l'eventuale ricaricamento in memoria (***swap_in***) delle pagine scaricate

1.1 – Determinazione di *quando* e *quante* pagine liberare

parametri

- ***freePages***: è il numero di pagine di memoria fisica libere in un certo istante
- ***requiredPages***: è il numero di pagine che vengono richieste per una certa attività da parte di un processo (o del SO)
- ***minFree***: è il numero minimo di pagine libere sotto il quale non si vorrebbe scendere
- ***maxFree***: è il numero di pagine libere al quale PFRA tenta di riportare freePages

PFRA è invocato nei casi seguenti

- **invocazione diretta** da parte di un processo che richiede ***requiredPages*** pagine di memoria se $(freePages - requiredPages) < minFree$, cioè se l'allocazione delle pagine richieste porterebbe la memoria fisica sotto il livello di guardia
- **attivazione periodica** tramite **kswapd** (kernel swap daemon), una funzione che viene attivata *periodicamente* e invoca PFRA se $freePages < maxFree$

PFRA determina il numero di pagine da liberare (**toFree**) tenendo conto delle pagine richieste con l'obiettivo di riportare sempre il sistema ad avere **maxFree** pagine libere

$$toFree = maxFree - freePages + requiredPages$$

1.2 – Determinazione di *quali* pagine liberare

dal punto di vista di PFRA i tipi di pagine sono

- pagine non scaricabili
  - pagine statiche del SO dichiarate non scaricabili
  - pagine allocate dinamicamente dal SO
  - pagine appartenenti alla pila S dei processi
- pagine mappate sul file eseguibile dei processi che possono essere scaricate senza mai riscriverle (codice e costanti)
- pagine che richiedono l'esistenza di una *Swap Area* su disco per essere scaricate
  - pagine dati
  - pagine della Pila U
  - pagine dello Heap
- pagine che sono mappate su un file: pagine appartenenti alla *Disk Cache*

1.2.1 – Mantenimento dell'informazione relativa all'accesso alle pagine

- esistono due liste globali *ordinate*, dette **LRU list**, che collegano tutte le pagine allocate appartenenti ai processi
  - ***active list***: contiene tutte le pagine che sono state accedute recentemente e non possono essere scaricate; in questo modo PFRA non dovrà scorrerle per scegliere un pagina da scaricare
  - ***inactive list***: contiene le pagine inattive da molto tempo che sono quindi candidate per essere scaricate
- periodicamente le due liste vengono scandite per spostare le pagine: l'obiettivo dello spostamento delle pagine all'interno e tra le due liste è di ***accumulare le pagine meno utilizzate in coda alla inactive****, mantenendo nella active le pagine più utilizzate di recente di tutti i processi*
- è un'approssimazione di LRU teorico a causa di vincoli HW (x64 non tiene traccia del numero di accessi alla memoria
  - Linux realizza l'approssimazione basata sul **bit di accesso A presente nel TLB**
    - **A** viene posto a 1 da x64 ogni volta che la pagina viene acceduta
    - **A** viene azzerato esplicitamente dal sistema operativo

spostamento delle pagine tra le due liste

- l'algoritmo descritto è semplificato rispetto a quello reale ma ne riflette la logica generale
- a ogni pagina viene associato un flag, **ref** (referenced), oltre al bit di accesso **A** definito dallo hardware
- il flag **ref** «raddoppia» il numero di accessi a una pagina necessari per spostarla da una lista all'altra
- definiamo una funzione periodica ***Controlla_liste*****,** attivata da kswapd, che rappresenta una sintesi semplificata rispetto alle funzioni reali del sistema e che esegue una scansione di ambedue le liste
  1. **scansione della active dalla coda** spostando eventualmente alcune pagine alla inactive
  2. **scansione della inactive dalla testa** (escluse le pagine appena inserite provenienti dalla active) spostando eventualmente alcune pagine alla active

  ogni pagina viene quindi processata una sola volta

- per completare la descrizione, alla funzione *Controlla_liste* dobbiamo aggiungere
  - funzioni che pongono in testa alla active con ref = 1 le *nuove pagine caricate da un processo*
  - funzioni che eliminano dalle liste pagine liberate definitivamente per la terminazione di un processo

funzione periodica *Controlla_liste*

A è ricavato dal TLB e ref dallo stato di partenza delle liste, mentre P è la pagina

1. **scansione della active list dalla coda**
   - se A = 1
     - azzera A
     - se (ref = 1) sposta P in testa alla active
     - se (ref = 0) pone ref = 1
   - se A = 0
     - se (ref = 1) pone ref = 0
     - se (ref = 0) sposta P in testa alla inactive con ref = 1

2. **scansione della inactive list dalla testa** (escluse le pagine appena inserite provenienti dalla active)
   - se A = 1
     - azzera A
     - se (ref = 1) sposta P in coda alla active con ref = 0
     - se (ref = 0) pone ref = 1
   - se A = 0
     - se (ref = 1) pone ref = 0
     - se (ref = 0) sposta P in coda alla inactive

esercizio 1

- per indicare sinteticamente le pagine con ref = **1** le scriviamo in lettere **maiuscole**, quelle con ref = **0** in lettere **minuscole**
- stato iniziale delle liste active: PP4 PP3 PP2 PC0 inactive: pp1 pp0
- si mostri l'evoluzione delle liste per tre attivazioni di kswapd sapendo che le pagine accedute (cioè con bit A = 1) a ogni attivazione sono sempre solo:
  Pc0, Pp4 e Pp1

soluzione

attivazioni	active				inactive		
stato iniz	PP4	PP3	PP2	PC0	pp1	pp0	
1	PP4	PC0	pp3	pp2	PP1	pp0	
2	PP4	PC0	pp1		PP3	PP2	pp0
3	PP4	PC0	PP1		pp3	pp2	pp0

esercizi con *Controlla_liste* e accessi alla memoria

- definiamo il seguente tipo di evento composto

  Read (lista pagine lette) – Write (lista pagine scritte) – N kswapd

- questo tipo di evento viene interpretato nel modo seguente
  - ripeti per N volte le operazioni di lettura / scrittura / kswapd
  - esegui un'ultima volta le sole operazioni di lettura / scrittura

- nota bene
  - se N = 0 vengono eseguite una volta sola le operazioni di lettura / scrittura, con le regole già viste in precedenza
  - se la lista di operazioni è vuota, viene eseguito N volte kswapd

esercizio 2

stato iniziale

```
      LRU ACTIVE:   PP5  PP4  PP3  PP2  PP1  PC2  PP0  PC0
      LRU INACTIVE: -

   STATO del TLB
         Pc0 : 0 -  0: 1:   ||   Pp0 : 1 -  1: 1:   ||
         Pc2 : 2 -  0: 1:   ||   Pp1 : 3 -  1: 1:   ||
         Pp2 : 4 -  1: 1:   ||   Pp3 : 5 -  1: 1:   ||
         Pp4 : 6 -  1: 1:   ||   Pp5 : 7 -  1: 1:   ||
               VOID         ||         VOID         ||
```

Evento 1: Read (Pc2) – Write (Pp1, Pp2, Pp3, Pp4, Pp5) – 1 kswapd

Evento 2: Read (Pc2) – Write (Pp1, Pp2, Pp3) – 1 kswapd

Evento 3: Read (Pc1) – Write (Pp1) – 1 kswapd

Evento 4: Read (Pc1) – Write (Pp1) – 1 kswapd

esercizio 2 – soluzione – 1

nel TLB iniziale tutte le pagine hanno bit A a 1

=== Evento 1: Read (Pc2) – Write (Pp1, Pp2, Pp3, Pp4, Pp5) – 1 kswapd ====

```
LRU ACTIVE:   PP5  PP4  PP3  PP2  PP1  PC2  PP0  PC0
LRU INACTIVE: -
STATO del TLB
      Pc0 : 0 -  0: 0:   ||   Pp0 : 1 -  1: 0:   ||
      Pc2 : 2 -  0: 1:   ||   Pp1 : 3 -  1: 1:   ||
      Pp2 : 4 -  1: 1:   ||   Pp3 : 5 -  1: 1:   ||
      Pp4 : 6 -  1: 1:   ||   Pp5 : 7 -  1: 1:   ||
            VOID         ||         VOID         ||
```

esercizio 2 – soluzione – 2

=== Evento 2: Read (Pc2) – Write (Pp1, Pp2, Pp3) – 1 kswapd ====

```
LRU ACTIVE:   PP5  PP4  PP3  PP2  PP1  PC2  pp0  pc0
LRU INACTIVE: -
STATO del TLB
      Pc0 : 0 -  0: 0:   ||   Pp0 : 1 -  1: 0:   ||
      Pc2 : 2 -  0: 1:   ||   Pp1 : 3 -  1: 1:   ||
      Pp2 : 4 -  1: 1:   ||   Pp3 : 5 -  1: 1:   ||
      Pp4 : 6 -  1: 0:   ||   Pp5 : 7 -  1: 0:   ||
            VOID         ||         VOID         ||
```

esercizio 2 – soluzione – 3

=== Evento 3: Read (Pc1) – Write (Pp1) – 1 kswapd ====

```
LRU ACTIVE:   PC1  PP3  PP2  PP1  PC2  pp5  pp4
LRU INACTIVE: PP0  PC0
STATO del TLB
      Pc0 : 0 -  0: 0:   ||   Pp0 : 1 -  1: 0:   ||
      Pc2 : 2 -  0: 0:   ||   Pp1 : 3 -  1: 1:   ||
      Pp2 : 4 -  1: 0:   ||   Pp3 : 5 -  1: 0:   ||
      Pp4 : 6 -  1: 0:   ||   Pp5 : 7 -  1: 0:   ||
      Pc1 : 8 -  0: 1:   ||         VOID          ||
```

esercizio 2 – soluzione – 4

=== Evento 4: Read (Pc1) – Write (Pp1) – 1 kswapd ====

```
LRU ACTIVE:   PC1  PP1  pp3  pp2  pc2
LRU INACTIVE: PP5  PP4  pp0  pc0
STATO del TLB
      Pc0 : 0 -  0: 0:   ||   Pp0 : 1 -  1: 0:   ||
      Pc2 : 2 -  0: 0:   ||   Pp1 : 3 -  1: 1:   ||
      Pp2 : 4 -  1: 0:   ||   Pp3 : 5 -  1: 0:   ||
      Pp4 : 6 -  1: 0:   ||   Pp5 : 7 -  1: 0:   ||
      Pc1 : 8 -  0: 1:   ||         VOID          ||
```

1.2.2 – scaricamento delle pagine della lista *inactive*

- prima di tutto vengono scaricate le *pagine appartenenti alla Page Cache non più utilizzate da nessun processo*, in ordine di NPF
  - sono quelle che nella tabella (dei descrittori) della memoria fisica hanno come unico elemento i riferimenti file

- poi vengono scelte le *pagine virtuali della inactive*, partendo dalla coda, ma tenendo conto della condivisione nel modo seguente
  - una pagina virtuale viene considerata scaricabile *solo se tutte le pagine condivise sono più invecchiate di essa*
  - ovvero se una pagina fisica è condivisa tra molte pagine virtuali, essa viene considerata scaricabile solo quando nella scansione della *inactive* si sono già trovate tutte le pagine che la condividono

esercizio 3

stato iniziale: maxFree = 3 minFree = 2

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                    ||  01 : Pc1 / <XX,1>          ||
      02 : Pp1                     ||  03 : Pp0 D                 ||
      04 : <G,0>                   ||  05 : <G,1>                 ||
      06 : <G,2>                   ||  07 : <G,3>                 ||
      08 : Pp2                     ||  09 : ----                  ||
      10 : ----                    ||  11 : ----                  ||

  LRU ACTIVE: PP2  PP1  PC1          LRU INACTIVE: pp0
```

si mostri lo stato della memoria e delle LRU list dopo i quattro eventi consecutivi seguenti

1. il processo P alloca le pagine di pila Pp3, Pp4 e Pp5
2. il processo P esegue una fork (R)
3. il processo P accede alle pagine Pc1, Pp1, Pp2 e Pp4, mentre kswapd interviene quattro volte
4. il processo P esegue sbrk (5) e poi scrive le pagine Pd0, Pd1, Pd2 e Pd3 (si indichino gli NPV delle pagine della inactive che vengono scaricate)

esercizio 3 – soluzione – *Evento 1: Write (Pp3, Pp4, Pp5)*

le richieste di pagine e le liberazioni avvengono nel seguente ordine

- la pagina Pp3 viene allocata in pagina 9 (free = 2)
- richiesta una pagine per Pp4, interviene PFRA, required = 1, da liberare = 3 – 2 + 1 = 2 pagine, **vengono liberate due pagine di Page Cache**, **cioè le pagine 4 e 5**, Pp4 viene allocata in pagina 4
- la pagina Pp5 viene allocata in pagina 5

```
____MEMORIA FISICA____(pagine libere: 2)_____________________________
      00 : <ZP>                  ||  01 : Pc1 / <XX,1>          ||
      02 : Pp1                   ||  03 : Pp0 D                 ||
      04 : Pp4                   ||  05 : Pp5                   ||
      06 : <G,2>                 ||  07 : <G,3>                 ||
      08 : Pp2                   ||  09 : Pp3                   ||
      10 : ----                  ||  11 : ----                  ||
LRU ACTIVE:  PP5  PP4  PP3  PP2  PP1  PC1       LRU INACTIVE: pp0
```

esercizio 3 – soluzione – *Evento 2: il processo P esegue una fork (R)*

- fork condivide inizialmente tutte le pagine tra i processi P (padre) ed R (figlio), poi richiede una pagina per la pagina in cima alla pila del processo padre P (pagina Pp5)
- dato che freePages = 2, viene invocato PFRA con requiredPages = 1 e si ha toFree = 3 – 2 + 1 = 2
- **vengono quindi liberate le due pagine 6 e 7 della Page Cache**
- la pagina **Pp5** viene allocata in pagina 6
- **tutte le nuove pagine sono inserite progressivamente in testa alla active o inactive (ossia nella stessa lista e nello stesso ordine di quelle del padre e con lo stesso ref)**

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                  ||  01 : Pc1 / Rc1 / <XX,1>   ||
      02 : Pp1 / Rp1             ||  03 : Pp0 / Rp0 D          ||
      04 : Pp4 / Rp4             ||  05 : Rp5                  ||
      06 : Pp5                   ||  07 : ----                 ||
      08 : Pp2 / Rp2             ||  09 : Pp3 / Rp3            ||
      10 : ----                  ||  11 : ----                 ||
LRU ACTIVE:    RP5  RP4  RP3  RP2  RP1  RC1  PP5  PP4  PP3  PP2  PP1  PC1
LRU INACTIVE:  rp0  pp0
```

esercizio 3 – soluzione – *Evento 3: Read (Pc1, Pp1, Pp2, Pp4) – 4 kswapd*

- quattro esecuzioni di kswapd portano in inactive tutte le pagine tranne quelle accedute; le pagine di R vengono spostate prima di quelle di P, perché le pagine di P sono inizialmente in stato di accedute nel TLB, e dunque si accodano in inactive a quelle di P (tranne che per la pagina **pp0** poiché era già in inactive)

LRU ACTIVE: PP4 PP2 PP1 PC1

LRU INACTIVE: pp5 pp3 rp5 rp4 rp3 rp2 rp1 rc1 rp0 pp0

- nell'evento successivo sarà necessario scaricare due pagine da inactive
  - **pp0** non viene scaricata, perché è in pagina condivisa con pagina meno invecchiata
  - **rp0** viene scaricata insieme a **pp0** → liberata pagina 3 (freePages = 3)
  - **rc1**, **rp1**, **rp2**, **rp3** e **rp4** non vengono scaricate perché condivise
  - **rp5** viene scaricata, liberando pagina 5 (freePages = 4)

LRU INACTIVE diventerà: pp5 pp3 rp4 rp3 rp2 rp1 rc1

esercizio 3 – soluzione – *Evento 4: sbrk (5), Write (Pd0, Pd1, Pd2, Pd3)*

si indichno gli NPV delle pagine della inactive che vengono scaricate

- la pagina **Pd0** viene allocata in pagina 7 (freePages = 2)
- poi interviene PFRA per liberarne 2: scaricate **rp0** e **rp5** (vedi pagina precedente)

LRU INACTIVE: pp5 pp3 rp4 rp3 rp2 rp1 rc1

- **Pd1** viene allocata in pagina 3 (freePages = 3) e **Pd2** viene allocata in pagina 5 (freePages = 2)
- interviene PFRA per liberarne 2: scaricate **pp3** / **rp3** e **pp5** (pagine 9 e 6)
- **Pd3** viene allocata in pagina 6 (freePages = 3)

____MEMORIA FISICA____(pagine libere: 3)_____

00 : <ZP>	01 : Pc1 / Rc1 / <XX,1>
02 : Pp1 / Rp1	03 : Pd1
04 : Pp4 / Rp4	05 : Pd2
06 : Pd3	07 : Pd0
08 : Pp2 / Rp2	09 : ----
10 : ----	11 : ----

LRU ACTIVE: PD3 PD2 PD1 PD0 PP4 PP2 PP1 PC1

LRU INACTIVE: rp4 rp2 rp1 rc1

complessivamente scaricate le 4 pagine: Pp0 / Rp0 Rp5 Pp3 / Rp3 e Pp5

2. – scaricamento delle pagine – swapping

- il meccanismo di swapping richiede che sia definita almeno una ***Swap Area*** su disco costituita da un file o da una partizione (vedi capitolo sul Filesystem)
- per semplicità negli esempi ipotizzeremo che esista una sola Swap Area di tipo file, che chiameremo anche ***Swap File***
- una Swap Area consiste di una sequenza di ***Page Slots***
  - ognuno di dimensione uguale a una pagina
  - **SWPN** (Swap Page Number) è l'identificatore di un Page Slot
  - esiste un contatore per ogni Page Slot detto ***swap_map_counter***
  - lo ***swap_map_counter*** è usato per tenere traccia del numero di PTE che riferiscono la pagina fisica scaricata su Swap Area (cioè il numero di pagine virtuali condivise in tale pagina fisica)
- la Swap Area è usata dalle pagine da scaricare che non sono mappate su file, ossia serve per le pagine anonime
  - pagine di tipo PRIVATE di area D, T e P che sono state scritte e dunque modificate
  - e pagine sdoppiate per COW

regole generali di swapping

- quando PFRA chiede di scaricare una pagina fisica (**swap_out**)
  - viene allocato un *Page Slot* nella Swap Area
  - la pagina fisica viene copiata nel Page Slot e liberata (**operazione su disco**)
  - allo *swap_map_counter* viene assegnato il numero di pagine virtuali che condividono la pagina fisica
  - in ogni PTE che condivideva la pagina fisica viene registrato il SWPN del Page Slot al posto del NPF e il bit di presenza viene azzerato
- quando un processo accede a una pagina virtuale scaricata su Swap Area
  - si verifica un Page Fault
  - il gestore del Page Fault attiva la procedura di caricamento (**swap_in**)
  - viene allocata una pagina fisica in memoria
  - il Page Slot indicato dalla PTE viene copiato nella pagina fisica (**operazione su disco**)
  - la PTE viene aggiornata inserendo il NPF della pagina fisica al posto del SWPN del Page Slot

swap_out

- per ogni pagina che deve essere liberata si tratta di determinare
  - se la pagina è *Dirty* (D), e in tale caso il suo contenuto va salvato su disco
  - se la pagina è *mappata su file in maniera SHARED* oppure è *anonima*
    - ✓ se è mappata su file va scritta su tale file
    - ✓ se è anonima va salvata nella *Swap Area*
  - se mappata su file ma non Dirty, libero la PF ma non scarico la pagina su file
- determinare se una pagina è Dirty è banale per le pagine fisiche non condivise, ma può essere complesso per le pagine condivise
- una pagina fisica PFx è Dirty se
  - il suo descrittore di pagina fisica è marcato D a seguito di un TLB flush
  - una delle pagine virtuali condivise in PFx è contenuta nel TLB ed è marcata D
- negli esercizi lo SWPN delle pagine scaricate in Swap Area è indicato nella TP preceduto da una lettera S, per distinguerlo da un normale NPF
- per esempio, se un processo avesse scaricata la pagina **d0** in Swap Area nel Page Slot **1**, nella TP ci sarebbe la PTE ⟨**d0: s1**⟩

esercizio 4

dato il seguente stato di memoria e TLB (fine evento 2 esercizio 3)

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                    ||  01 : Pc1 / Rc1 / <XX,1>   ||
      02 : Pp1 / Rp1               ||  03 : Pp0 / Rp0 D          ||
      04 : Pp4 / Rp4               ||  05 : Rp5 D                 ||
      06 : Pp5                     ||  07 : ----                 ||
      08 : Pp2 / Rp2               ||  09 : Pp3 / Rp3            ||
      10 : ----                    ||  11 : ----                 ||
____STATO del TLB_____________________________________________________
      Pc1 : 01 -  0: 1:    ||   Pp0 : 03 -  1: 0:    ||
      Pp1 : 02 -  1: 1:    ||   Pp2 : 08 -  1: 1:    ||
      Pp3 : 09 -  1: 0:    ||   Pp4 : 04 -  1: 1:    ||
      Pp5 : 06 -  1: 0:    ||         -----          ||
```

si vogliono liberare le pagine Pp0 / Rp0 Rp5 Pp3 / Rp3 Pp5

soluzione: vanno scaricate in Swap Area le pagine Pp0 / Rp0 Rp5 Pp3 / Rp3 Pp5

esercizio 4 – soluzione

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                  ||  01 : Pc1 / Rc1 / <XX,1>  ||
      02 : Pp1 / Rp1             ||  03 : Pd1                 ||
      04 : Pp4 / Rp4             ||  05 : Pd2                 ||
      06 : Pd3                   ||  07 : Pd0                 ||
      08 : Pp2 / Rp2             ||  09 : ----                ||
      10 : ----                  ||  11 : ----                ||
SWAP FILE:  Pp0 / Rp0   Rp5   Pp3 / Rp3   Pp5    ----    ----    ----

PROCESSO: P  ***************************************************
    PT: <c0 :-->     <c1 :01  R>  <k0 :-->    <s0 :-->    <d0 :07  W>  <d1 :03  W>
        <d2 :05 W>   <d3 :06  W>  <d4 :-->    <p0 :s0 R>  <p1 :02  R>  <p2 :08  R>
        <p3 :s2 R>   <p4 :04  R>  <p5 :s3 W>  <p6 :-->
PROCESSO: R  ***************************************************
    PT: <c0 :-->     <c1 :01 R>   <k0 :-->     <s0 :-->     <p0 :s0 R>  <p1 :02   R>
        <p2 :08  R>  <p3 :s2 R>   <p4 :04  R>  <p5 :s1  W>  <p6 :-->
```

swap-in

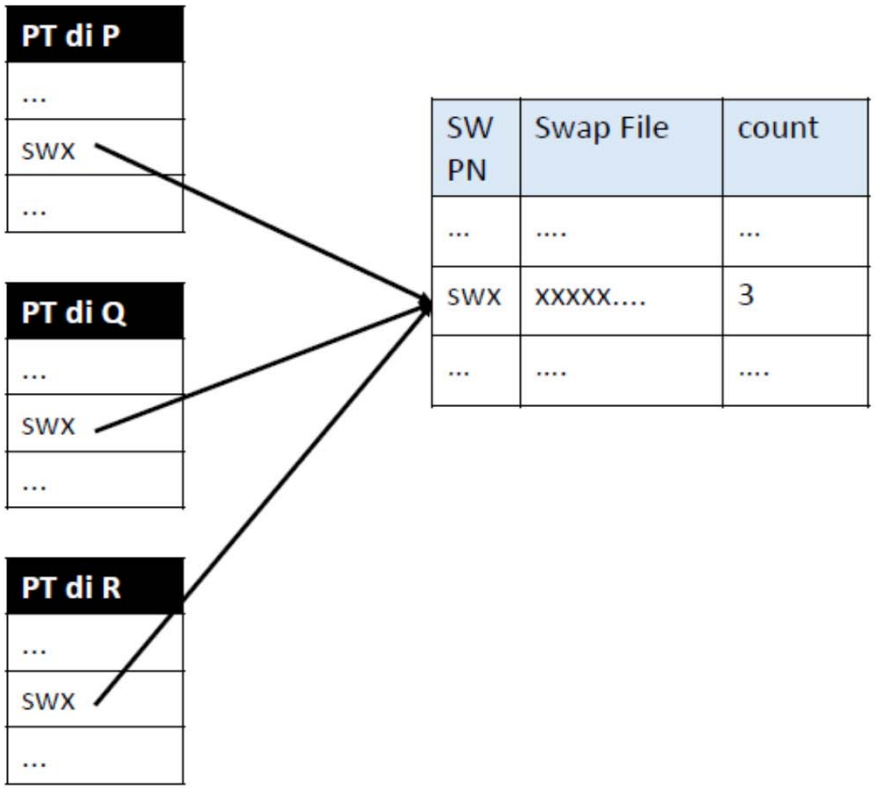
- spesso una pagina sulla cui viene eseguito uno swap_in in seguito va di nuovo scaricata
- LINUX tenta di limitare il più possibile i trasferimenti tra Swap Area e memoria nel caso di pagine che vengono scaricate in Swap Area più volte senza essere modificate
  - non cancella la pagina dalla Swap Area al momento dello swap_in
  - a un successivo swap_out, se la pagina non è stata mai modificata dal momento dello swap_in, non è necessario riscriverla su disco
- *Swap Cache*
  - *l'insieme delle pagine che sono state rilette da Swap Area e non sono state modificate*
  - *e alcune strutture ausiliarie* (**Swap_Cache_Index**) che permettono di gestirla come se tali pagine fossero mappate sulla Swap Area
- una pagina appartiene alla Swap Cache se è presente sia in memoria sia su Swap Area e se è registrata nello Swap_Cache_Index

meccanismo della Swap Cache

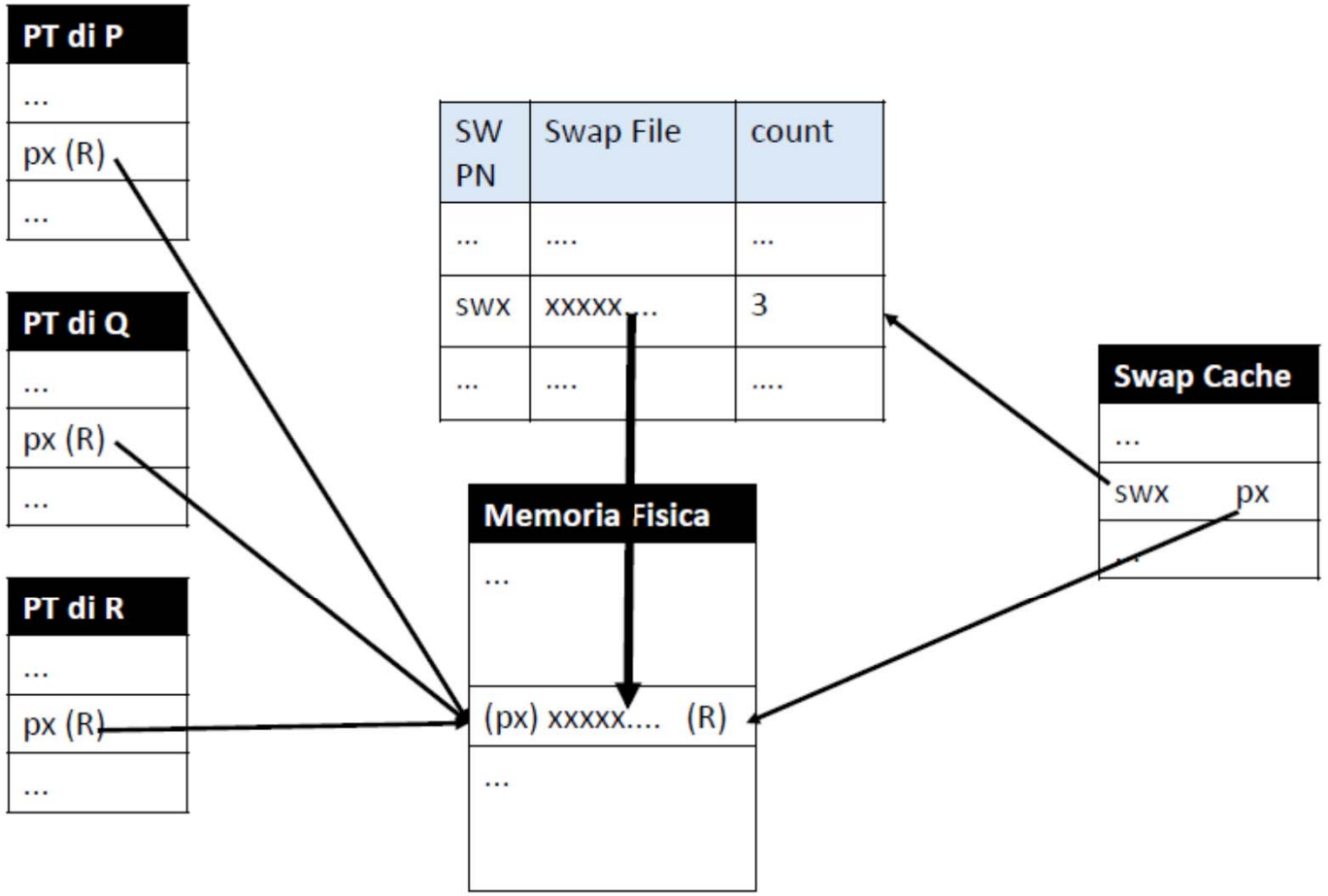
- le pagine che richiedono swapping sono solo le ***pagine anonime*** (PRIVATE) e la loro gestione risulta essere così
  - quando si esegue uno *swap_in* la pagina viene caricata in memoria fisica, ma la copia nella Swap Area non viene eliminata; *la pagina caricata in memoria viene marcata in sola lettura* (per poi potere gestire COW)
  - nello Swap_Cache_Index viene inserito un descrittore che contiene i riferimenti sia alla pagina fisica sia al Page Slot
  - finché la pagina caricata viene solamente letta non ci sono azioni ulteriori, e se la pagina viene nuovamente scaricata allora non è necessario riscriverla su disco (è come se fosse etichettata non Dirty)
- se la pagina viene scritta, si verifica un Page Fault (per violazione protezione – COW) che causa le seguenti operazioni
  - viene allocata una nuova pagina che diventa privata, cioè non appartiene più alla Swap Area
  - la sua protezione viene posta a W
  - il contatore swap_map_counter viene decrementato
- se il contatore è diventato 0, il Page Slot nella Swap Area viene liberato
- ecco la relazione relazione tra lo swap_in e le liste LRU
  - la NPV che ha causato lo Swap_in, viene inserita in testa alla active con ref = 1
  - eventuali altre NPV condivise con quella vengono poste in coda alla inactive con ref = 0

esempio

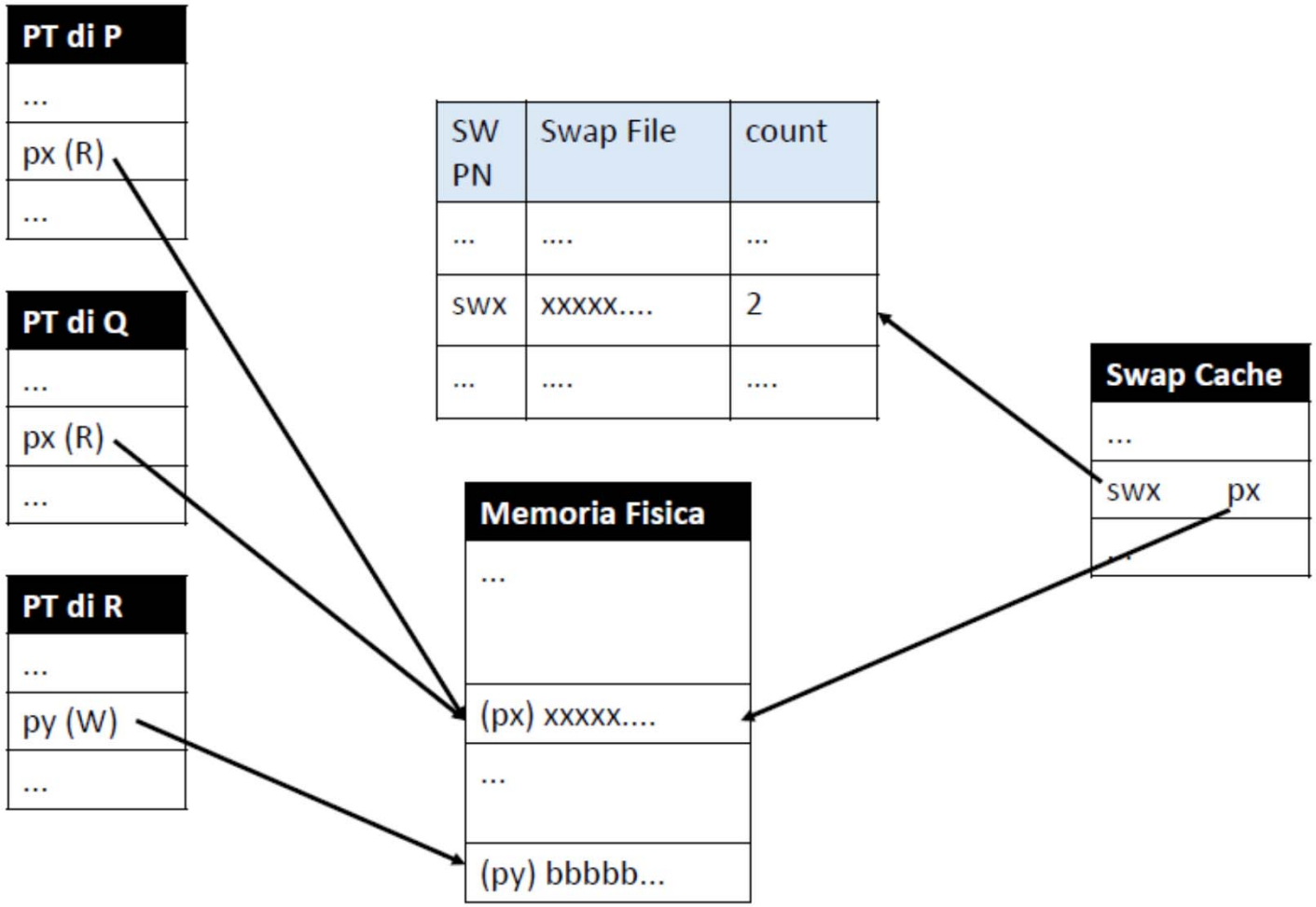
Stato iniziale



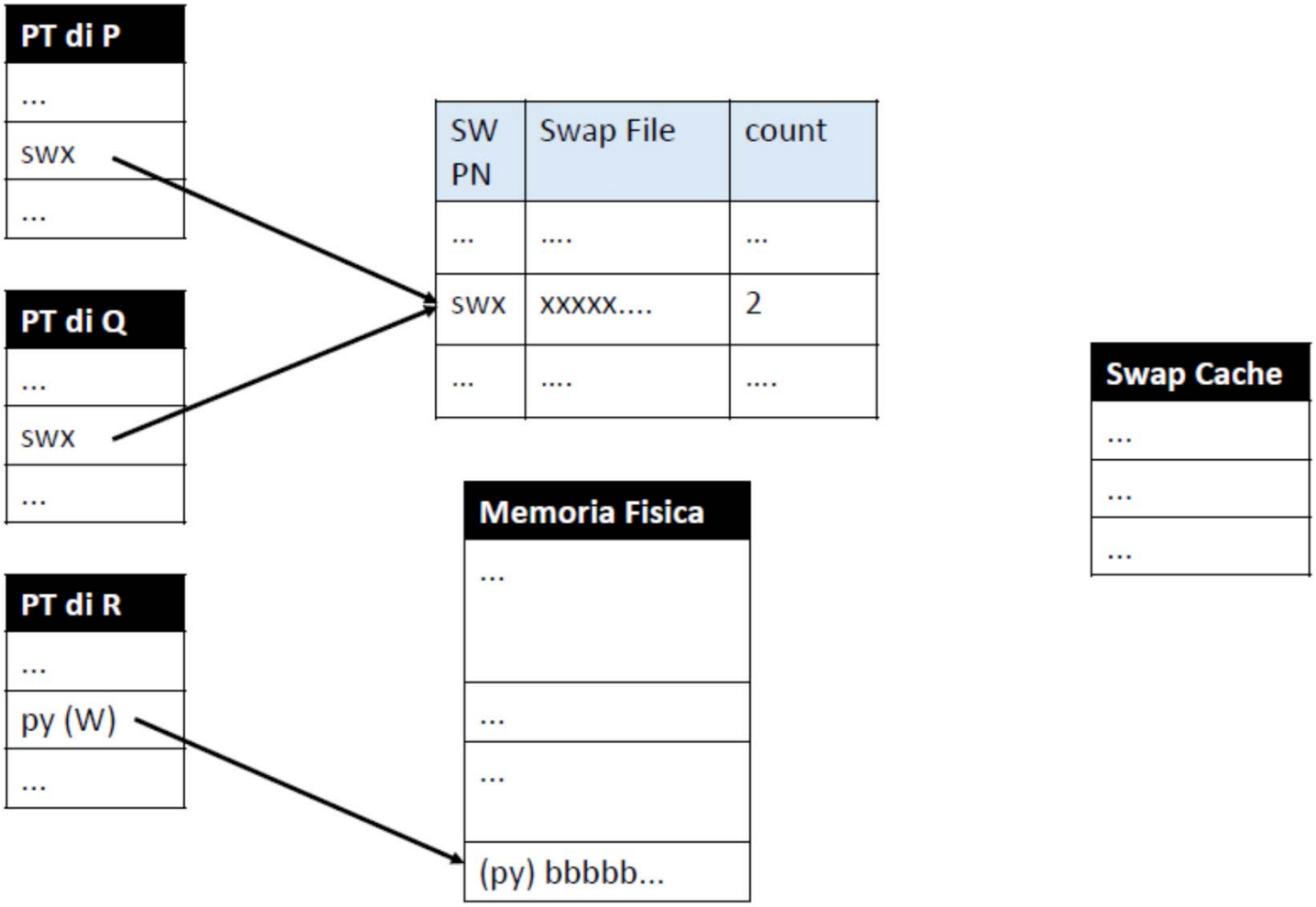
Un processo esegue una lettura sulla pagina → swap_in



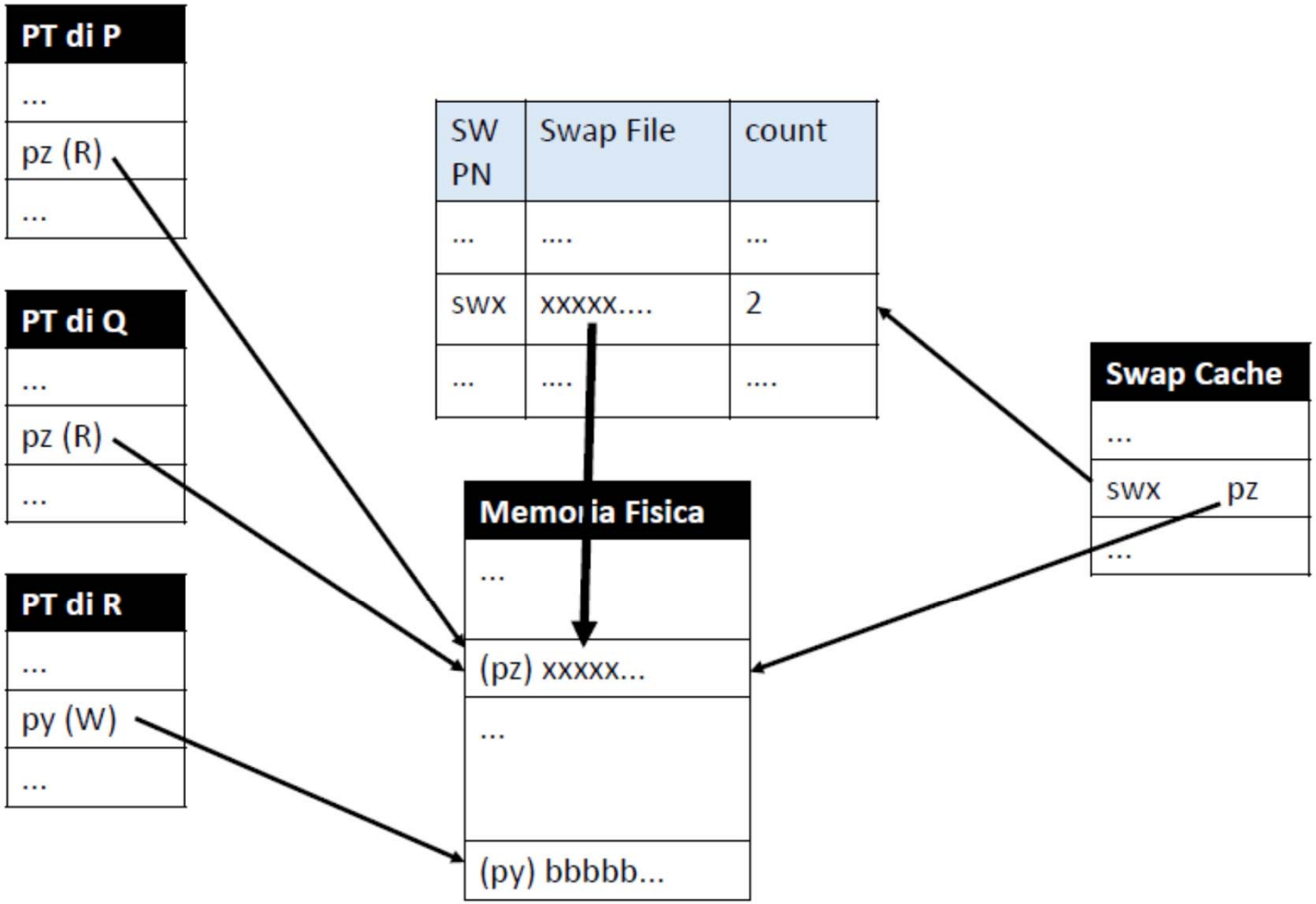
R esegue una scrittura sulla pagina



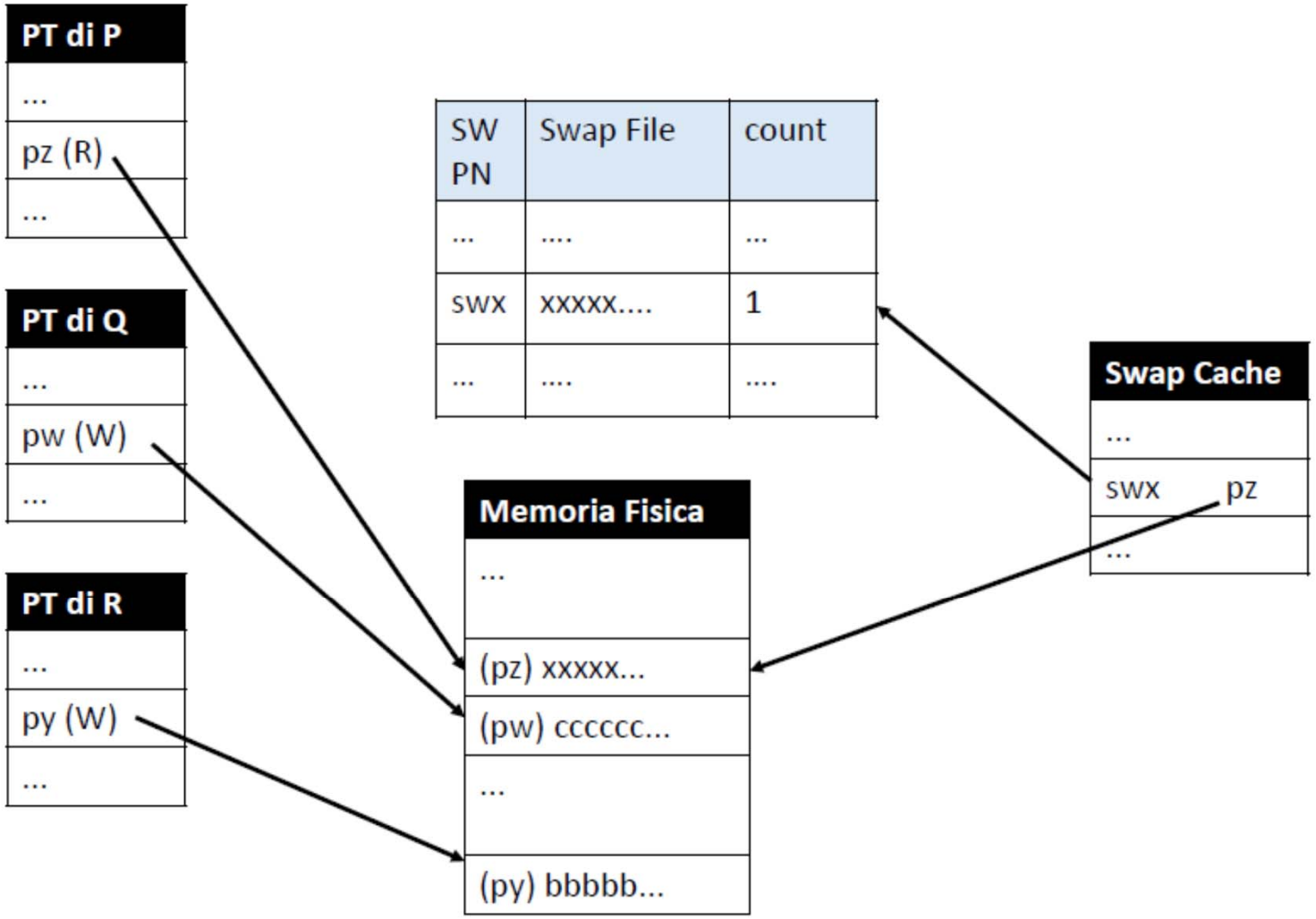
px viene nuovamente scaricata (swap_out)



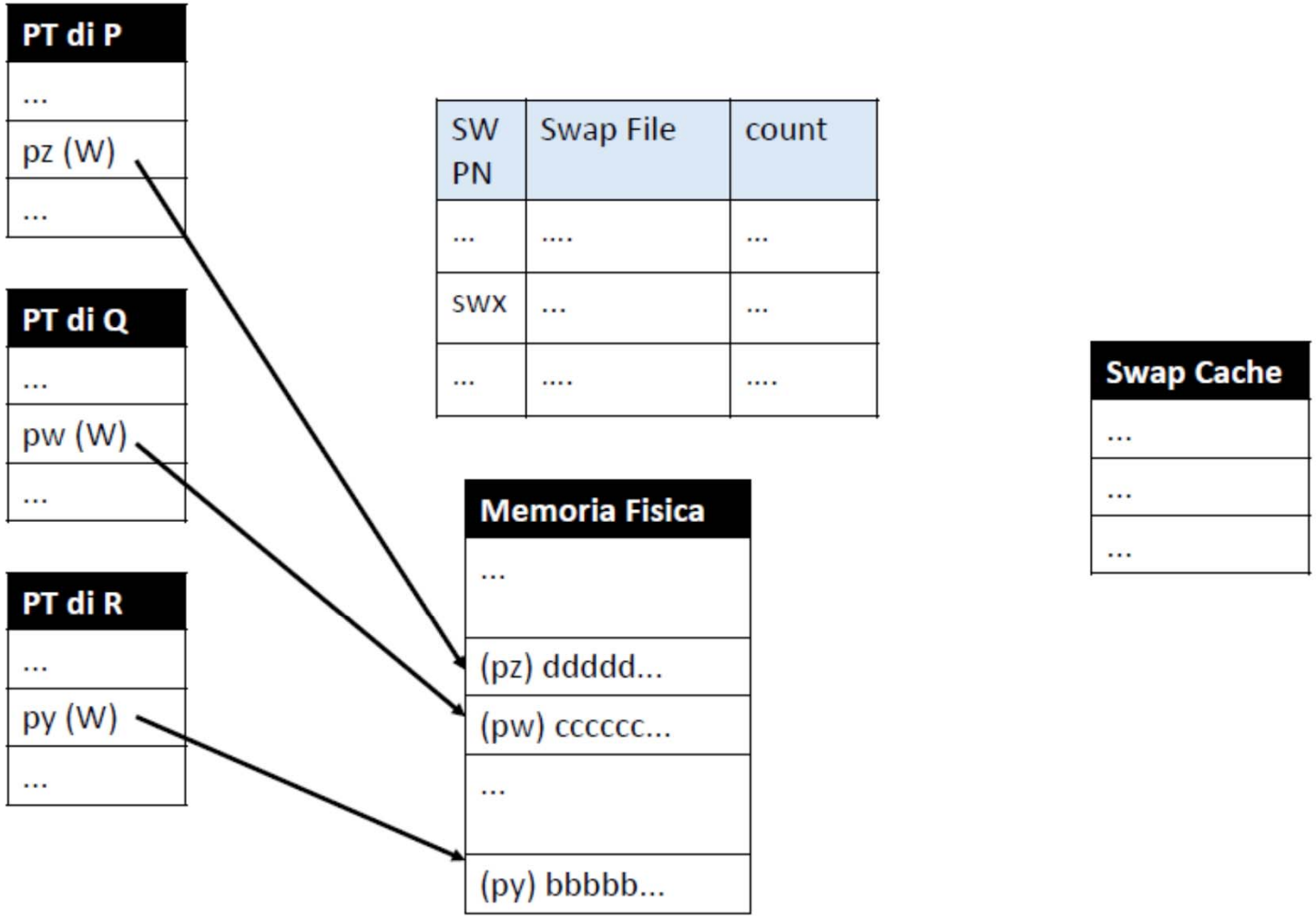
P o Q leggono px → swap_in



Q esegue una scrittura



P esegue una scrittura



esercizio 5

stato iniziale (da stato finale esercizio 4 modificando liste LRU)

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                    ||  01 : Pc1 / Rc1 / <XX,1>  ||
      02 : Pp1 / Rp1               ||  03 : Pd1                 ||
      04 : Pp4 / Rp4               ||  05 : Pd2                 ||
      06 : Pd3                     ||  07 : Pd0                 ||
      08 : Pp2 / Rp2               ||  09 : ----                ||
      10 : ----                    ||  11 : ----                ||
SWAP FILE:  Pp0 / Rp0   Rp5   Pp3 / Rp3   Pp5   ----   ----   ----
LRU ACTIVE:   PD1  PD0  PP4  PC1
LRU INACTIVE: pd3  pd2  pp2  pp1  rp4  rp2  rp1  rc1
```

eventi

1) lettura di **Pp0** **e** **Pp5**, e scrittura di **Pp3**

2) scrittura di **Pp5**

esercizio 5 – soluzione – *evento 1: Read (Pp0, Pp5), Write (Pp3)*

- swap_in di **Pp0 / Rp0** in pagina 9 (freePages = 2)
- lettura di **Pp5**, chiamata PFRA e swap_out di **Pp1** e **Pp2** da pagine **2** e **8** (freePages = 4)
- swap_in di **Pp5** in pagina 2 (freePages = 3)
- swap_in di **Pp3 / Rp3** in pagina 8 (freePages = 2)
- scrittura di **Pp3** con COW, chiamata PFRA e swap_out di **Pd2** e **Pd3** da pagine **5** e **6** (freePages = 4)
- scrittura di **Pp3** in pagina 5 (freePages = 3)

```
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                  ||  01 : Pc1 / Rc1 / <XX,1>  ||
      02 : Pp5                   ||  03 : Pd1                 ||
      04 : Pp4 / Rp4             ||  05 : Pp3                 ||
      06 : ----                  ||  07 : Pd0                 ||
      08 : Rp3                   ||  09 : Pp0 / Rp0           ||
      10 : ----                  ||  11 : ----                ||
SWAP FILE:  Pp0 / Rp0   Rp5   Rp3   Pp5   Pp1 / Rp1   Pp2 / Rp2   Pd2   Pd3   ----   ----   ----
LRU ACTIVE:    PP3  PP5  PP0  PD1  PD0  PP4  PC1
LRU INACTIVE:  rp4  rc1  rp0  rp3
```

NB: le pagine Pp0 / Rp0, Rp3 e Pp5 sono in Swap Cache (ricaricate ma non modificate)

esercizio 5 – soluzione – *evento 2: Write (Pp5)*

la scrittura di Pp5 elimina Pp5 dalla Swap Area (e quindi dalla Swap Cache) rendendo Pp5 una pagina privata del processo P, abilitata normalmente in scrittura

```
PROCESSO: P  ************************************************
    PT: <c0 :-->    <c1 :01 R>  <k0 :-->    <s0 :-->    <d0 :07 W>  <d1 :03 W>
        <d2 :s6 W>  <d3 :s7 W>  <d4 :-->    <p0 :09 R>  <p1 :s4 R>  <p2 :s5 R>
        <p3 :05 W>  <p4 :04 R>  <p5 :02 W>  <p6 :-->
____MEMORIA FISICA____(pagine libere: 3)_____________________________
      00 : <ZP>                 ||  01 : Pc1 / Rc1 / <XX,1>  ||
      02 : Pp5                  ||  03 : Pd1                 ||
      04 : Pp4 / Rp4            ||  05 : Pp3                 ||
      06 : ----                 ||  07 : Pd0                 ||
      08 : Rp3                  ||  09 : Pp0 / Rp0           ||
      10 : ----                 ||  11 : ----                ||
SWAP FILE: Pp0/Rp0  Rp5  Rp3  ----  Pp1/Rp1  Pp2/Rp2  Pd2  Pd3
LRU ACTIVE:   PP3  PP5  PP0  PD1  PD0  PP4  PC1
LRU INACTIVE: rp4  rc1  rp0  rp3
```

interferenza tra gestione della memoria e scheduling

- l'allocazione e la deallocazione della memoria interferiscono con i meccanismi di scheduling
- supponiamo che
  - un processo Q a bassa priorità sia un forte consumatore di memoria
  - contemporaneamente sia in funzione un processo P molto interattivo
- può accadere che, mentre il processo P è in attesa, il processo Q carichi progressivamente tutte le sue pagine forzando fuori memoria le pagine del processo P (e magari anche della Shell da dove Q era stato invocato)
- quando il processo P viene risvegliato ed entra rapidamente in esecuzione grazie ai suoi elevati diritti di esecuzione (VRT basso), si verifica un ritardo dovuto al caricamento delle pagine che erano state scaricate

Out Of Memory Killer (OOMK)

- in sistemi molto carichi l'algoritmo PFRA può non riuscire a risolvere la situazione
- in tale caso come estrema ratio deve invocare l'algoritmo OOMK, che seleziona un processo e lo elimina (kill)
- l'algoritmo OOMK viene invocato quando la memoria libera è molto poca e PFRA non è riuscito a liberare sufficienti pagine fisiche
- significativamente, la funzione più delicata di OOMK si chiama `select_bad_process` e ha il compito di fare una scelta intelligente del processo da eliminare, in base ai seguenti criteri
  - il processo abbia molte pagine occupate, cosicché la sua eliminazione dia un contributo significativo di pagine libere
  - abbia svolto poco lavoro
  - abbia priorità bassa (tendenzialmente indica processi poco importanti)
  - non abbia privilegi di superuser (questi processi svolgono funzioni importanti)
  - non gestisca direttamente componenti hardware, per non lasciarli in uno stato inconsistente

trashing (congestione)

- i meccanismi adottati da Linux sono complessi e il loro comportamento è difficile da predire in diverse condizioni di carico
- calibrare i parametri nei diversi contesti può essere uno dei compiti più difficili per l'amministratore del sistema
- pertanto non è escluso che in certe situazioni si verifichi un fenomeno detto ***trashing*** (o ***congestione***)
  - il sistema continua a deallocare pagine che vengono nuovamente richieste dai processi
  - le pagine vengono continuamente scritte e rilette da disco, e nessun processo riesce a progredire fino a terminare e liberare risorse