Gestione della memoria

Argomenti:

- Memoria virtuale
- Paginazione della memoria
- TLB
- Segmentazione della memoria

Materiale di studio:

- Capitoli 8.3 e 8.5

Memoria virtuale

- La memoria vista da un processo (programma) è ideale:
  - consiste di uno spazio unico di 2^w locazioni, dove w sono i bit usati per l'indirizzo.
  - Il processo ha accesso a tutte le locazioni.
  - Non deve condividere la memoria con altri programmi.

- Questa visione però non è reale:
  - La memoria fisica è una risorsa condiva con altri processi e il sistema operativo.
  - La memoria fisicamente disponibile potrebbe essere minore di quella vista dal processo.

- E' però utile mantenere una distinzione tra quello che vede un processo e la memoria fisica perché semplifica lo sviluppo di programmi.
  - Questa tecnica si chiama **memoria virtuale** per sottolineare che un processo vede una memoria che non esiste fisicamente (virtuale).
  - Hardware e sistema operativo provvedono a mappare la memoria virtuale in quella fisica.

Gestione della memoria

La gestione della memoria effettuata dal sistema operativo usando appositi meccanismi forniti dall'hardware consiste nel:

- **Suddividere la memoria** per consentire in essa la presenza di più processi (presupposto indispensabile per usare in modo efficiente il processore in un sistema multitasking),

- Effettuare la suddivisione della memoria in **modo efficiente**, per farci stare il numero maggiore possibile di processi: quanto maggiore è questo numero, tanto minore è la probabilità che la ready list sia vuota e che il processore rimanga inattivo (idle).

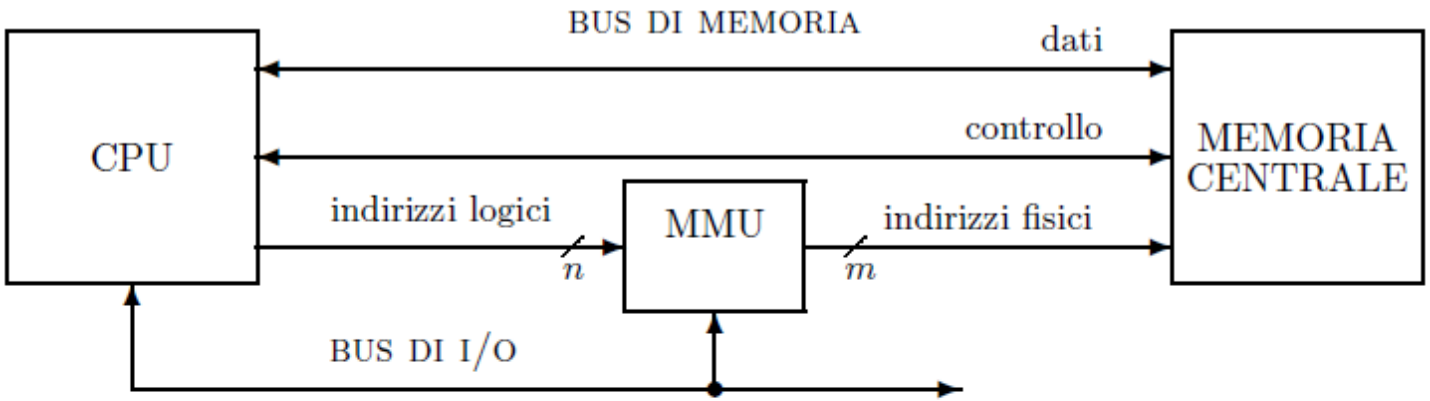
- Mantenere questa **gestione trasparente ai processi** e gestire la mappatura tra memoria virtuale e memoria fisica di ogni processo.

Indirizzi logici e indirizzi fisici

- Gli indirizzi richiesti da un processo non individuano direttamente indirizzi fisici della memoria.
  - **Indirizzi logici** → indirizzi nella memoria virtuale del processo.
  - **Indirizzi fisici** → indirizzi nella memoria fisica del processore
- La MMU trasforma indirizzi logici in indirizzi fisici.

Memory management unit

- Queste operazioni vengono effettuata tramite la **Memory Management Unit** (MMU)



Tecniche di gestione della memoria

Le tecniche di gestione della memoria più comuni sono:

- Paginazione
- Segmentazione

Paginazione

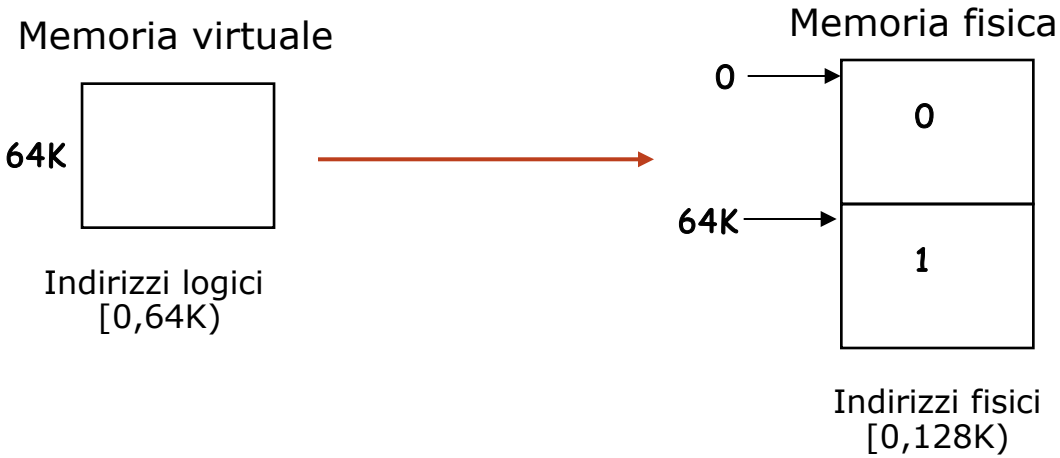
- La memoria fisica viene divisa in blocchi, detti **pagine fisiche** o **frame**, di dimensioni uguali e spesso piccole (es: 4KB);
- Anche la memoria virtuale è diviso in blocchi delle medesime dimensioni, detti **pagine logiche**.
- La tecnica di paginazione consente di "mappare" ciascuna pagina logica in un qualsiasi frame fisico.
- La MMU realizza questo meccanismo.

Versione semplificata di MMU

Storicamente, i primi sistemi di MMU avevano lo scopo di consentire a processori con pochi bit di indirizzo di accedere ad una memoria fisica più estesa.

Esempio: mappare indirizzi a 16 bit del processo (memoria max 64 Kbyte) in indirizzi a 17 bit.

Schema di semplice MMU



Primo esempio

- Supponiamo la memoria fisica sia divisa in due pagine da 64KB

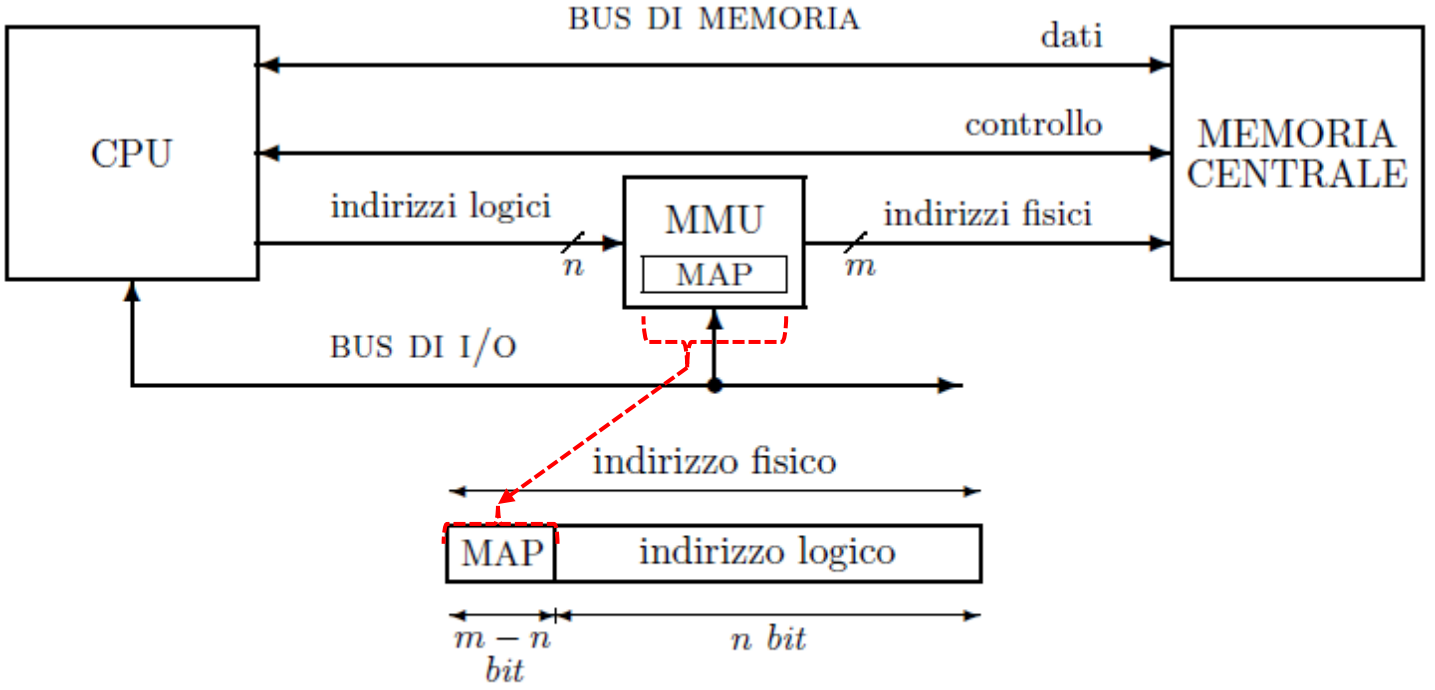
- Problema: trasformare indirizzi logici generati dal processo (nel range 0÷64K) in indirizzi fisici nel range 64K÷128K, per accedere alla parte alta (seconda pagina) di quella memoria.
- La trasformazione effettuata dal MMU è semplice: l'indirizzo fisico si ottiene aggiungendo un bit (il più significativo) a quello logico; questo bit seleziona la pagina cui si accede:
  - 0 individua la prima pagina della memoria fisica (range 0÷64K);
  - 1 individua la seconda pagina (range 64K÷128K).

Generalizzazione

- Se si aggiungessero q bit (più significativi) all'indirizzo logico si potrebbe accedere ad una memoria fisica costituita da 2^q pagine da 64 KByte.

- I primi sistemi di MMU operavano questo tipo di semplice trasformazione (indirizzo logico $\rightarrow$ indirizzo fisico), ottenuta preponendo all'indirizzo logico i q bit contenuti in un **registro di mappa (MAP)**.

Schema di semplice MMU

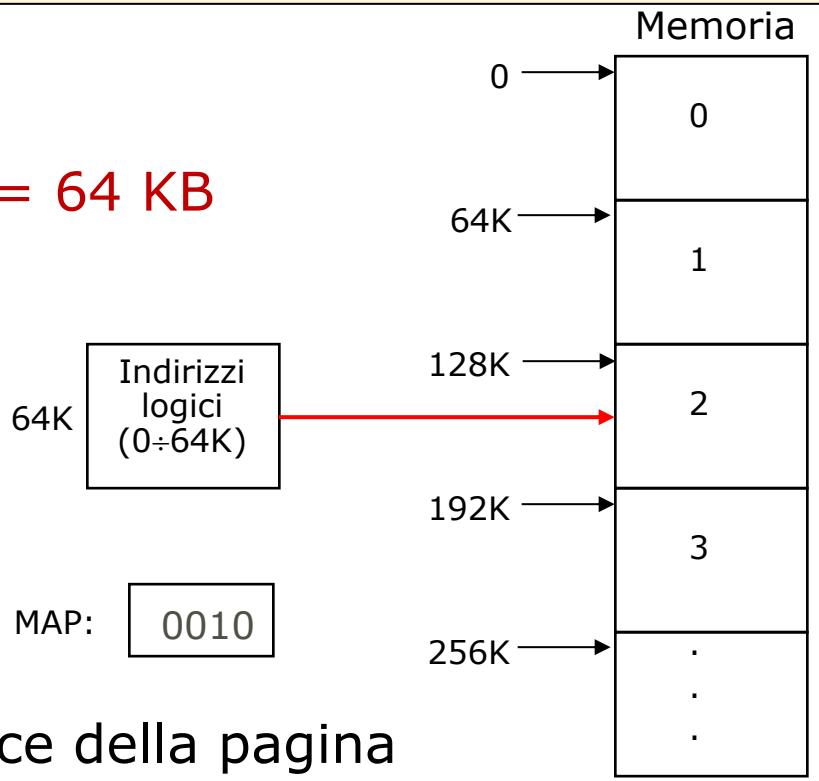


Accesso alla memoria tramite un modulo MMU.

Il processore deve essere dotato di una istruzione (privilegiata) con cui modificare il registro MAP per accedere alla pagina fisica desiderata. Nella figura ciò è ottenuto con un'istruzione di I/O.

Schema di semplice MMU

- Indirizzi logici da n = 16 bit
- Memoria logica indirizzabile: 2^{16} = 64 KB
- Indirizzi fisici da m = 20 bit
- Memoria fisica: 2^{20} = 1 MB
- La memoria fisica è ripartita in
  16 pagine da 64 KB
- Bit da aggiungere: q = 4



Il registro di mappa fornisce l'indice della pagina fisica su cui è mappato l'indirizzo logico

indirizzo logico:		0111 1111 0011 1100
indirizzo fisico:	0010	0111 1111 0011 1100

Schema di semplice MMU

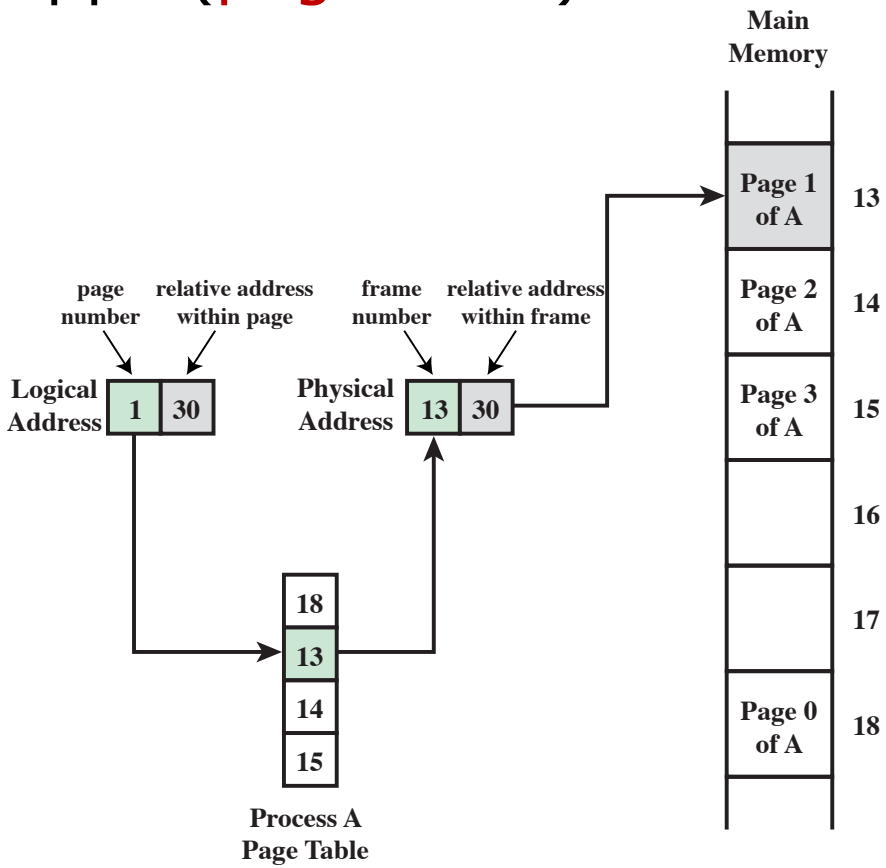
- L'esempio precedente funziona solo quando la memoria logica consiste di una sola pagina.
  - Ad esempio, quando lo spazio di indirizzamento logico è molto piccolo.

- In genere conviene dividere in pagine anche lo spazio di indirizzi logico
  - Fornisce maggior flessibilità e permette un uso più efficiente della memoria.

- Le pagine logiche hanno la stessa dimensione delle pagine fisiche.

Page Table

- Sia $p = 2^q$ il numero di pagine logiche
- Il modulo MMU contiene un insieme di $p = 2^q$ registri di mappa (page table)

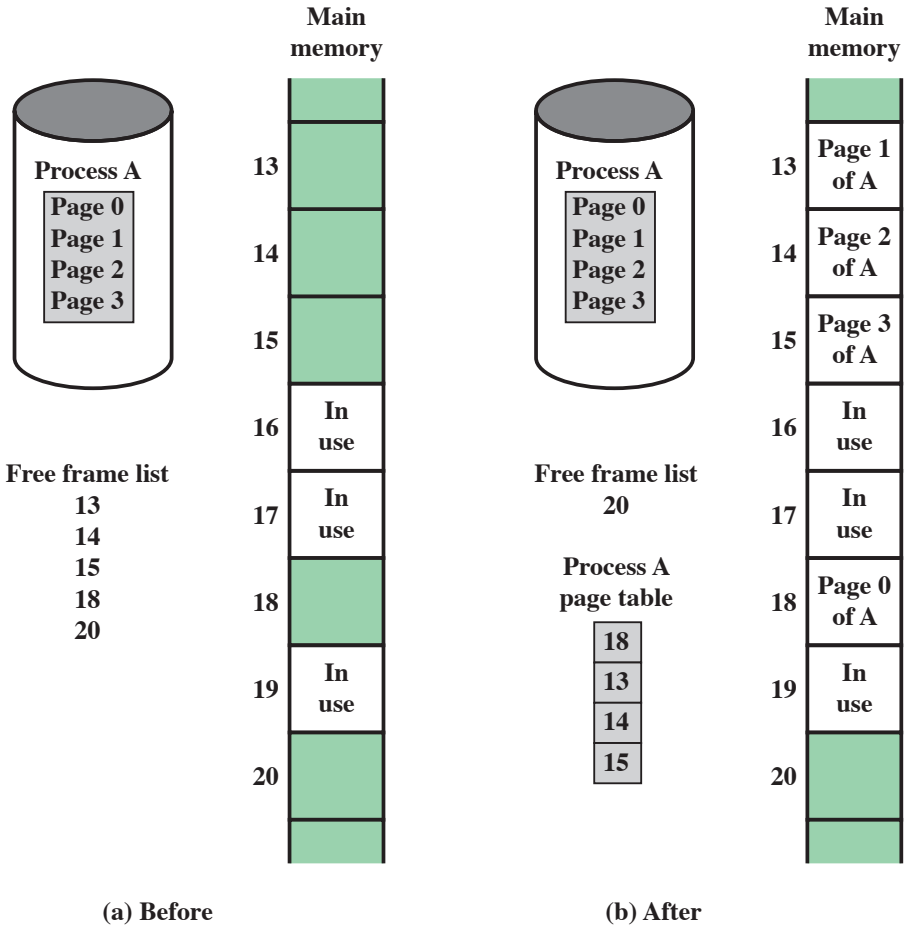


Page Table

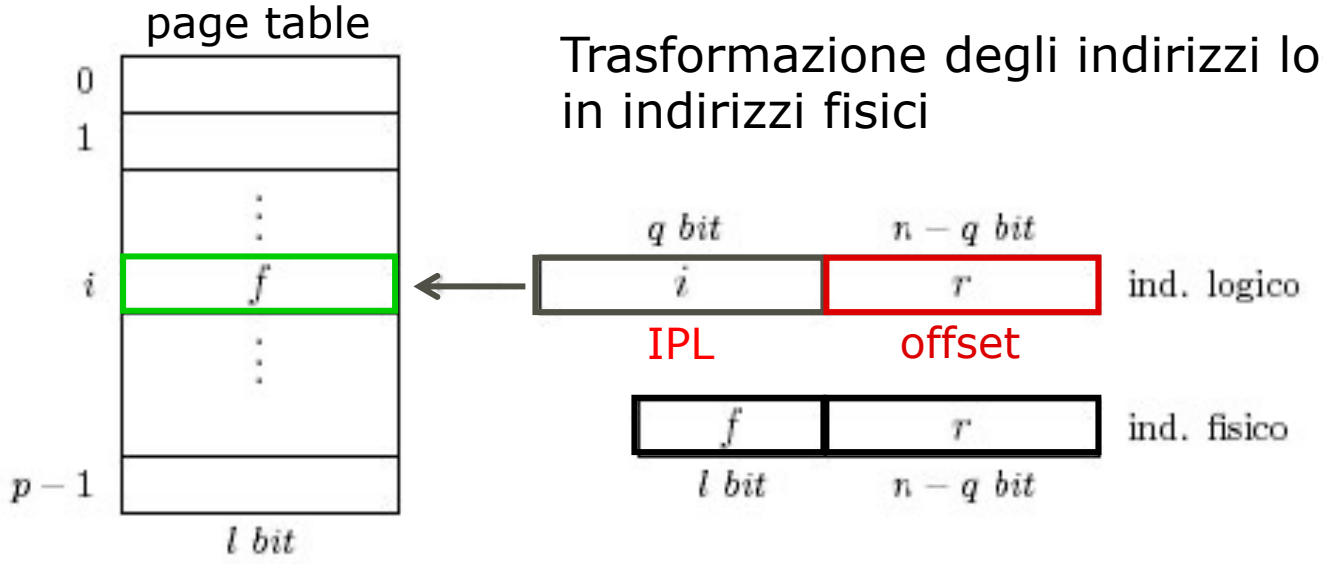
- Gli indirizzi logici generati dal processore durante l'esecuzione di un programma iniziano dall'indirizzo logico 0 e occupano una o più pagine logiche

- I registri della page table associata al programma specificano in quali pagine fisiche sono mappate le sue pagine logiche.

Esempio di memoria suddivisa in Pagine



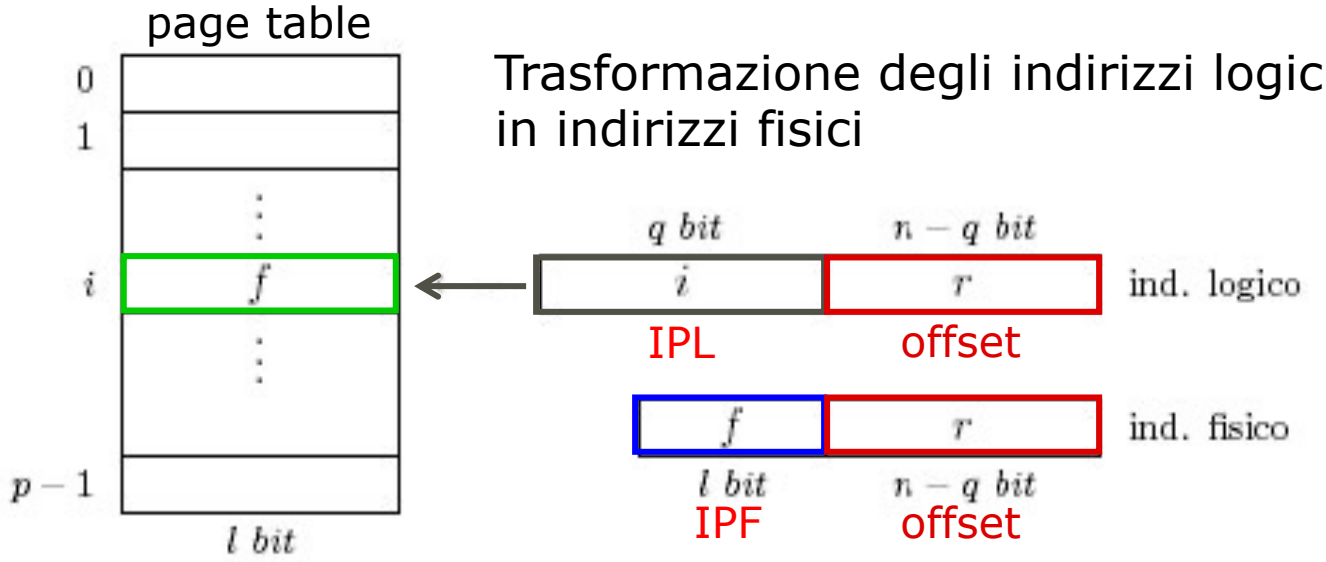
Campi dell'indirizzo logico



L'indirizzo logico:

- i *q* bit più significativi dell'indirizzo logico forniscono l'indice *i* della pagina logica (IPL) cui l'indirizzo appartiene;
- i rimanenti (*n* - *q*) bit contengono l'offset *r* di quell'indirizzo all'interno della pagina logica.

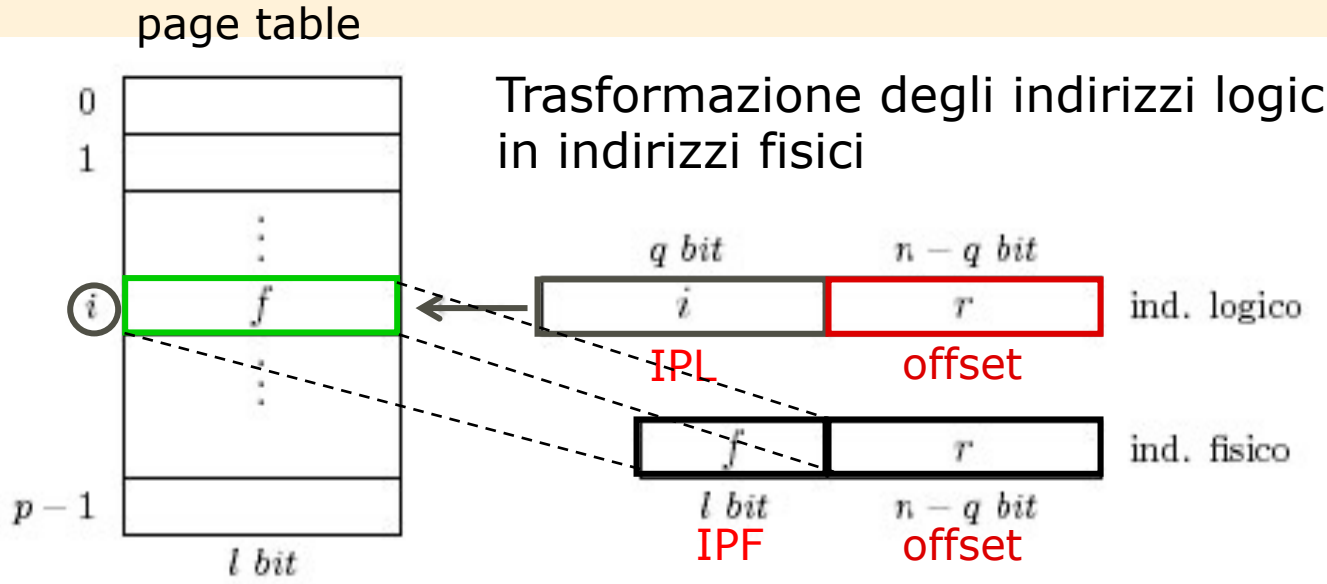
Campi dell'indirizzo fisico



L'indirizzo fisico è costituito da:

- *i* bit (più significativi) che forniscono l'indice *f* della pagina fisica (IPF o frame #), che sostituiscono i *q* bit dell'IPL;
- (*n* - *q*) bit che contengono l'offset *r* dell'indirizzo fisico: questo offset è il medesimo specificato nell'indirizzo logico (le pagine logiche e quelle fisiche hanno lunghezza uguale)

Trasformazione indirizzo logico → fisico

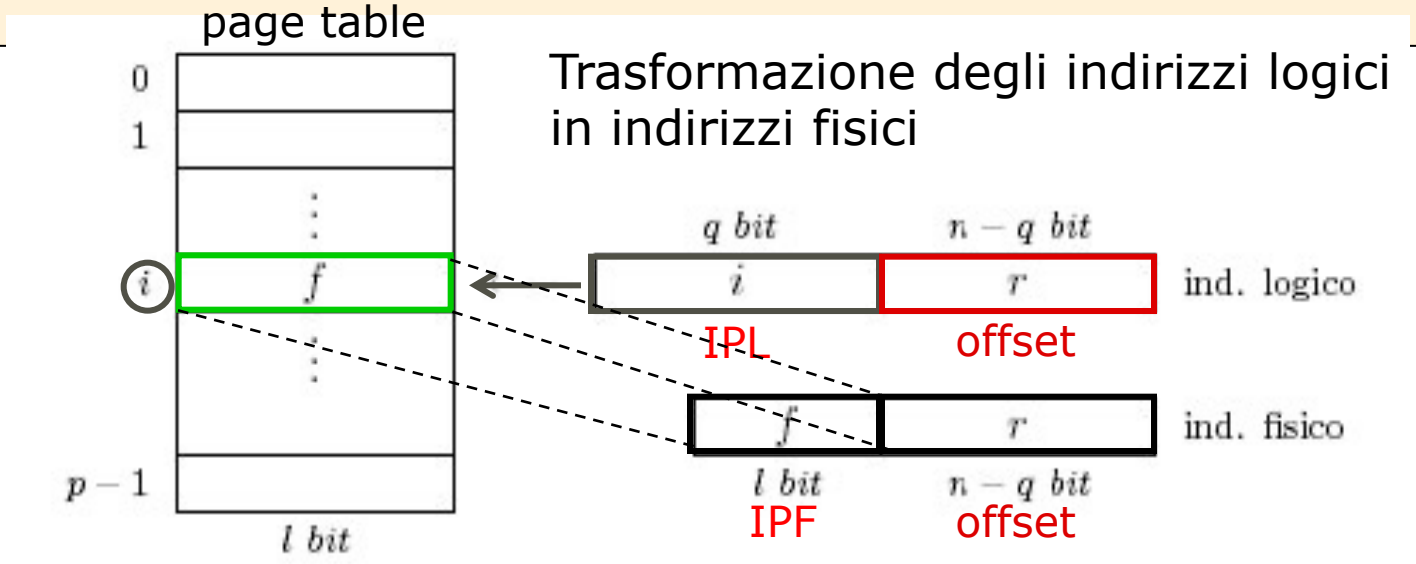


L'indirizzo fisico si ottiene sostituendo l'IPL (q bit) con l'IPF (l bit)

- l'indice di pagina logica *i* individua l'*i*-esimo registro di mappa, che contiene l'indice *f* della pagina fisica (IPF); questo IPF individua, tra le 2^l pagine fisiche presenti, quella in cui è mappata l'*i*-esima pagina logica;

- la trasformazione indirizzo logico → indirizzo fisico si ottiene sostituendo i *q* bit dell'IPL con gli *l* bit dell'IPF, estratti dall' l'*i*-esimo registro di mappa.

MMU con page table



Se è $q < l$ (indirizzo logico più corto di quello fisico):

- allora la memoria fisica è più estesa di quella logica (situazione comune in passato, quando i processori avevano capacità di indirizzamento limitate);

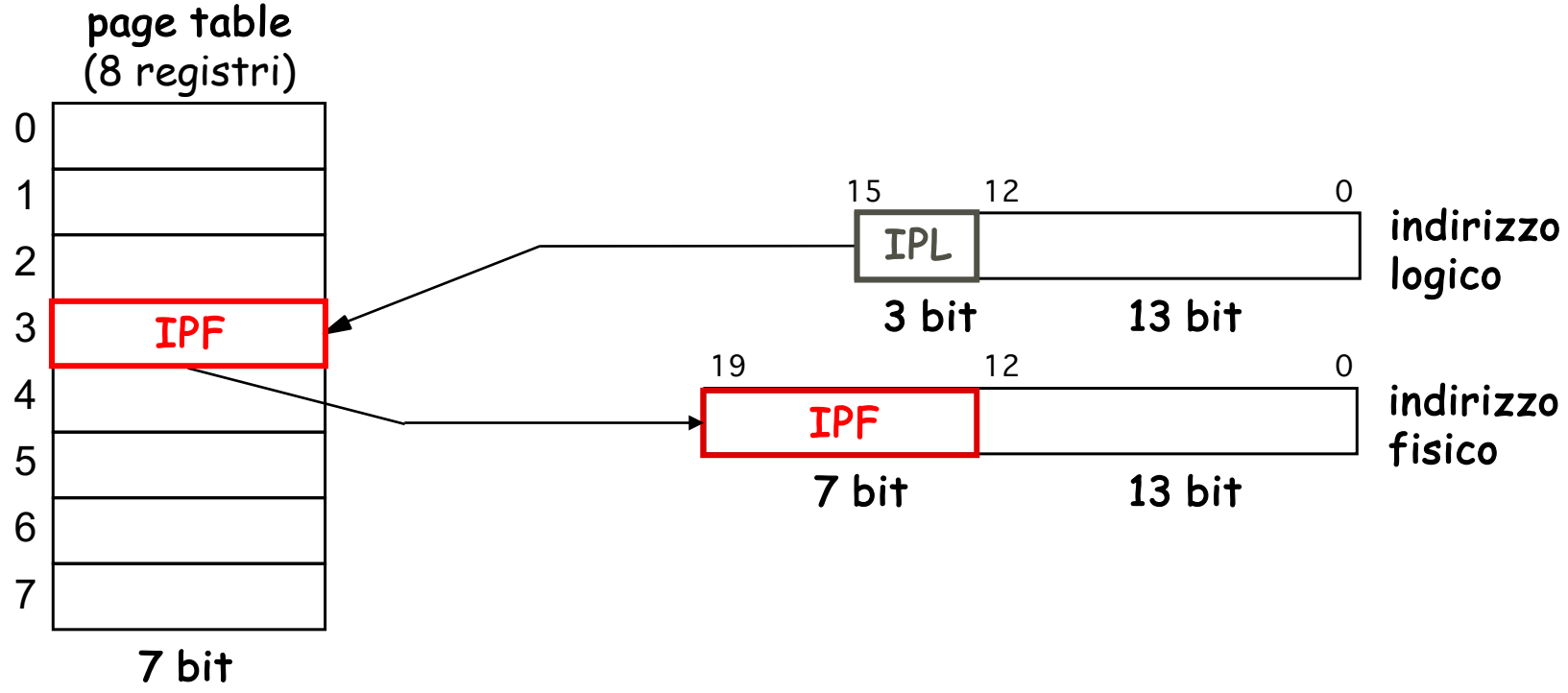
Se è $q > l$ (indirizzo logico più lungo di quello fisico):

- allora la memoria logica è più estesa di quella fisica (situazione di figura, comune negli elaboratori attuali).

Page Table: caso $q < l$

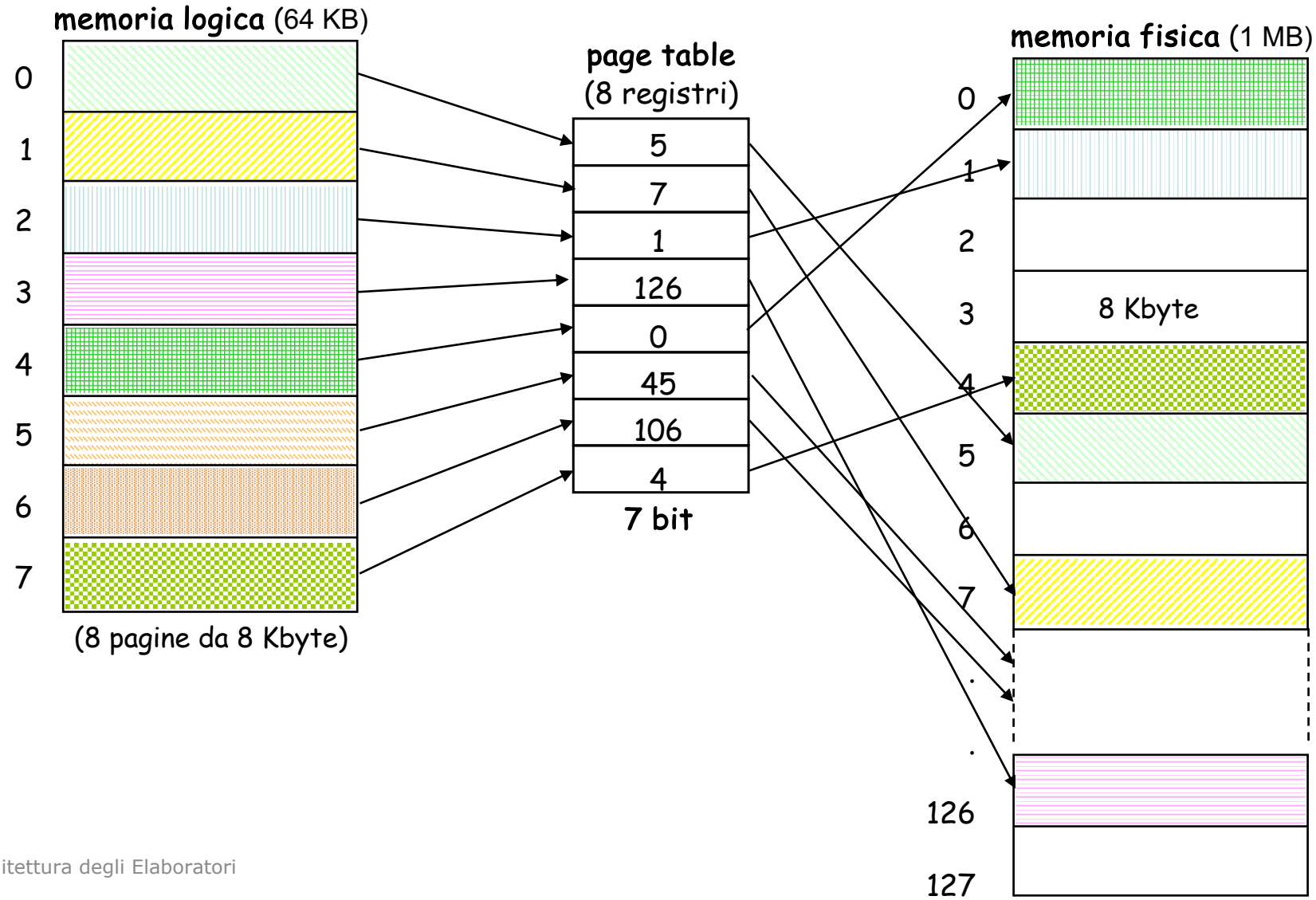
- Si consideri il caso di un processore che genera indirizzi logici da n = 16 bit (può indirizzare 2^{16} = 64 Kbyte);
- la memoria fisica abbia indirizzi fisici da m=20 bit (può indirizzare 2^{20} = 1 Mbyte):
  - si può suddividere lo spazio degli indirizzi logici in 8 pagine;
  - l'indice di pagina logica è di 3 bit (q =3)
  - la dimensione di ciascuna pagina è di 64K/8 = 8 Kbyte;
  - 8 Kbyte corrispondono ad un offset di 13 bit,
  - la memoria fisica risulta comprendere 1M/8K = 128 pagine,
  - l'indice di pagina fisica è di 7 bit (l =7),
  - la MMU ha una page table con 8 registri di mappa (uno per ciascuna delle 8 pagine logiche) da 7 bit.

Caso $q < l$: indirizzo logico → fisico



Impostando opportunamente il contenuto dei registri di MMU nella page table, è possibile "mappare" le 8 pagine logiche su 8 qualsiasi delle 128 pagine fisiche.

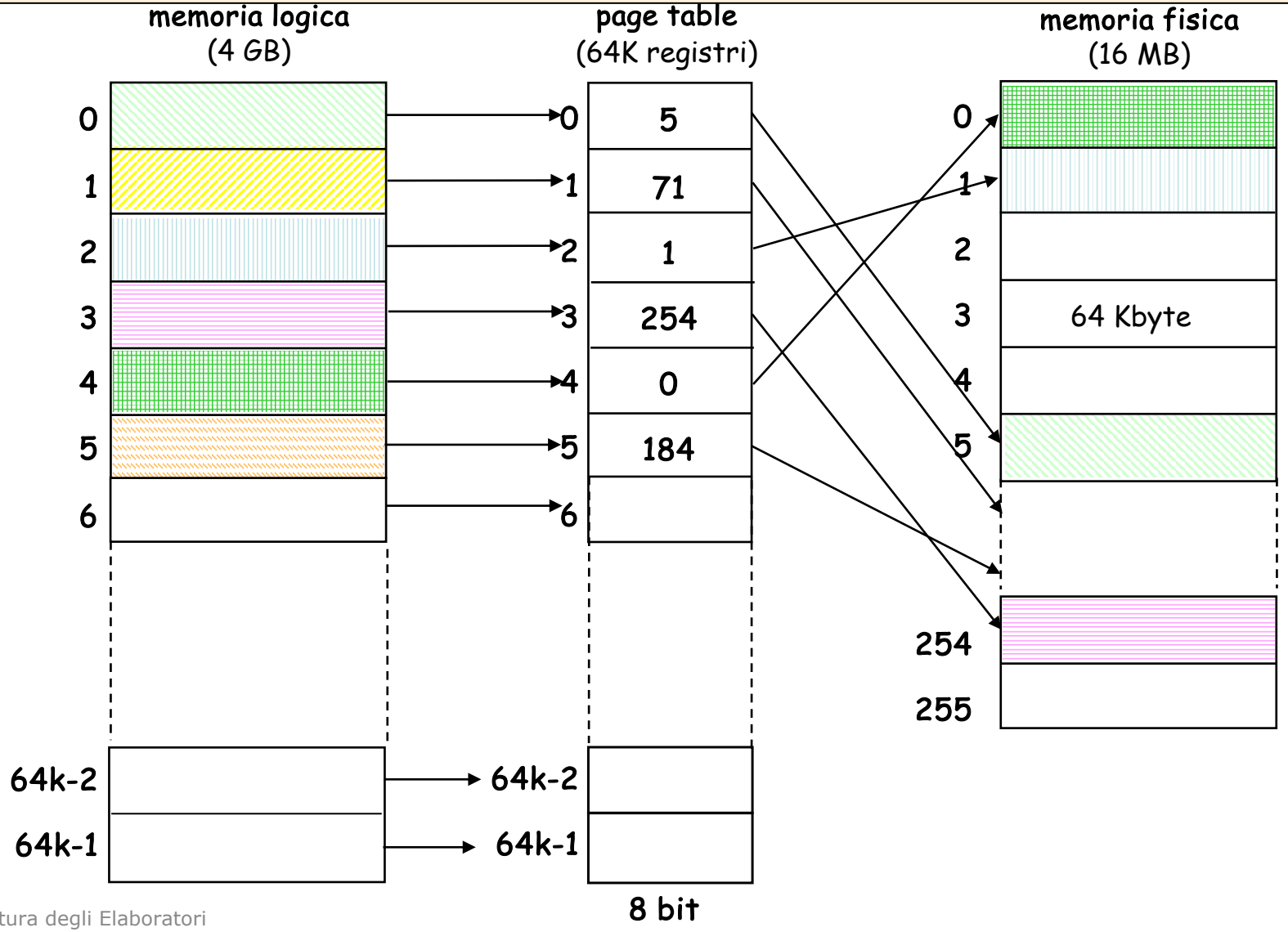
Caso q < l : pagine logiche → fisiche



Page Table: caso $q > l$

- Si consideri il caso di un processore che genera indirizzi (logici) da n = 32 bit (può indirizzare 2^{32} = 4 Gbyte);
- memoria fisica da 16 Mbyte (indirizzi fisici da m = 24 bit):
  - si può suddividere lo spazio degli indirizzi logici in pagine da 64 KByte,
  - il numero di pagine logiche è 4G/64K = 64K,
  - l'indice di pagina logica è di 16 bit (q =16),
  - 64 Kbyte corrispondono ad un offset di 16 bit,
  - la memoria fisica risulta comprendere 16M/64K = 256 pagine,
  - l'indice di pagina fisica è di 8 bit: 2^8 = 256 (l =8),
  - la MMU ha una page table con 64K registri di mappa da 8 bit.

Caso q > l : pagine logiche → fisiche



Osservazioni sulla page table (1/2)

- E’ normale che un processo usi solo le prime k pagine logiche disponibili: quando è in esecuzione quel processo sono significativi solo i primi k elementi della page table, che individuano dove si trovano le k pagine fisiche su cui vengono mappate le sue k pagine logiche.
- Altri processi possono occupare altre pagine fisiche della stessa memoria.
- A ciascun processo è associato il set di IPF che vengono inseriti nella page table (ad opera del sistema operativo) quando quel processo viene mandato in esecuzione.
- Quando sorge la necessità di sospendere l’esecuzione di un processo per mandarne in esecuzione un altro, sarà necessario inserire nella page table gli indici di pagina fisica (IPF) del nuovo processo.

Osservazioni sulla page table (2/2)

- Nel caso in cui il numero di **pagine logiche sia elevato** (64K nell'esempio precedente), non è praticabile l'uso di un modulo MMU con una page table delle dimensioni necessarie a contenere tanti registri di mappa (interni al *chip*) quante sono le pagine logiche ad essi associate.
- In tal caso si può adottare una soluzione che preveda di collocare la **page table in un'area protetta della memoria fisica**.
- Il modulo MMU contiene solo un puntatore alla page table associata al processo attivo (**page table pointer**).
  - La page table è una risorsa critica, accessibile solo con istruzioni riservate al sistema operativo.
  - Ogni accesso alla memoria richiede ora due accessi alla memoria: uno alla page table per ottenere l'IPF, uno alla pagina fisica cercata
  - Questo numero si riduce tramite cache e TLB (vedi slide più avanti)

Resident set

- Gli indirizzi logici di memoria vengono tradotti in indirizzi fisici durante l'esecuzione, quindi:
  - Le pagine di un processo possono essere collocati in aree non contigue della memoria.
  - Le pagine possono subire spostamenti in memoria fisica e occupare aree di memoria diverse, in momenti diversi.

- Non è necessario che tutti le pagine di un processo siano presenti in memoria durante l'esecuzione:
  - Il sistema operativo può caricare in memoria solo alcune pagine (**resident set**: insieme delle pagine logiche presenti in memoria fisica)
  - Le altre pagine possono rimanere nella memoria secondaria (disco), liberando memoria fisica.

Memory fault

- Quando il processo P richiede un indirizzo che non si trova nel resident set (**memory fault**) interviene il sistema operativo che carica nella memoria fisica la pagina contenuta nel disco (**swap in**).
- Quando la memoria fisica è totalmente occupata, la lettura di un nuovo blocco in seguito a un memory fault, richiede di:
  - Scegliere con una opportuna **politica di rimpiazzo** quale pagina in memoria fisica deve essere rimpiazzata (**swap out**).
  - Inserisce la nuova pagina.

Principio di località

I sistemi di memoria virtuale funzionano grazie al **principio di località**

- In un intervallo di tempo limitato, gli indirizzi di memoria richiesti da un processo tendono ad essere localizzati nelle medesime aree (località temporale);

- Una volta portato in memoria fisica un blocco, il processo continuerà ad accedere a quel blocco per un po' di tempo (località spaziale)

Costo dei page fault

- Siano:
  - $\mathbf{t_A}$ il tempo di accesso alla memoria centrale,
  - $\mathbf{p_F}$ la probabilità di page fault,
  - $\mathbf{t_F}$ il tempo necessario a gestire un page fault,

- Assumiamo che mediamente un accesso su N provochi un **memory fault:**
  - probabilità di page fault: $p_F = 1/N$ $(0 \leq p_F \leq 1)$,
  - il sistema è tanto più efficiente quanto maggiore è N (quanto minore è p_F).

Costo dei page fault

- il **tempo medio di accesso alla memoria**, in presenza di page fault è:

$$t_M = (1 - p_F) \times t_A + p_F \times t_F = t_A + p_F \times (t_F - t_A)$$

- Dati t_A e t_F (determinati dall'HW), t_M cresce linearmente con p_F.
- Nel caso sia $t_A = 60$ ns, $t_F = 20$ ms, $p_F = 10^{-3}$ (un page fault ogni 1000 accessi: N=1000):
  - il tempo medio è: t_M (ns) $= 60 + 10^{-3} \times (20 \times 10^6 - 60) \cong 20060$ ns e il computer rallenta di più di 300 volte.
  - Per contenere il rallentamento sotto il 10%, occorre un N > 3300000 (meno di 1 page fault / 3300000 accessi).

Trashing

- Può accadere che il sistema operativo decida di rimpiazzare (swap out) un blocco proprio poco prima che il processo cerchi di accedervi.
- In tal caso il blocco va riportato subito dopo in memoria (swap in), con il conseguente swap out di un altro.

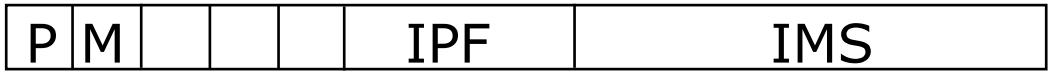
- Se questo fenomeno succede troppo frequentemente, si parla di **trashing**:
  - il processore passa gran parte del suo tempo ad eseguire swap, invece che ad eseguire le istruzioni dei processi.

Paginazione – versione completa

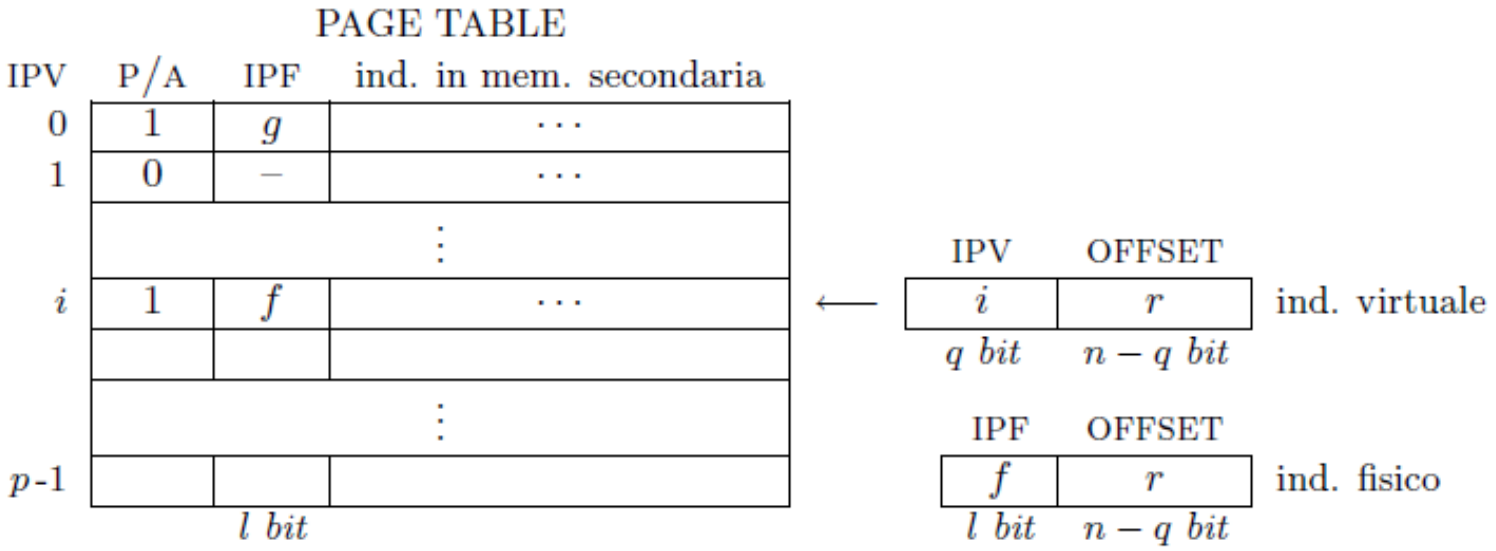
- Ad ogni processo è associata una page table,
- L'elemento di indice i nella page table corrisponde alla i-esima pagina della memoria virtuale del processo e contiene:
  - l'indice della pagina fisica (IPF) nella memoria fisica in cui è collocata la i-esima pagina del processo,
  - il suo indirizzo in memoria secondaria (IMS),
  - alcuni bit di controllo

Paginazione – versione completa

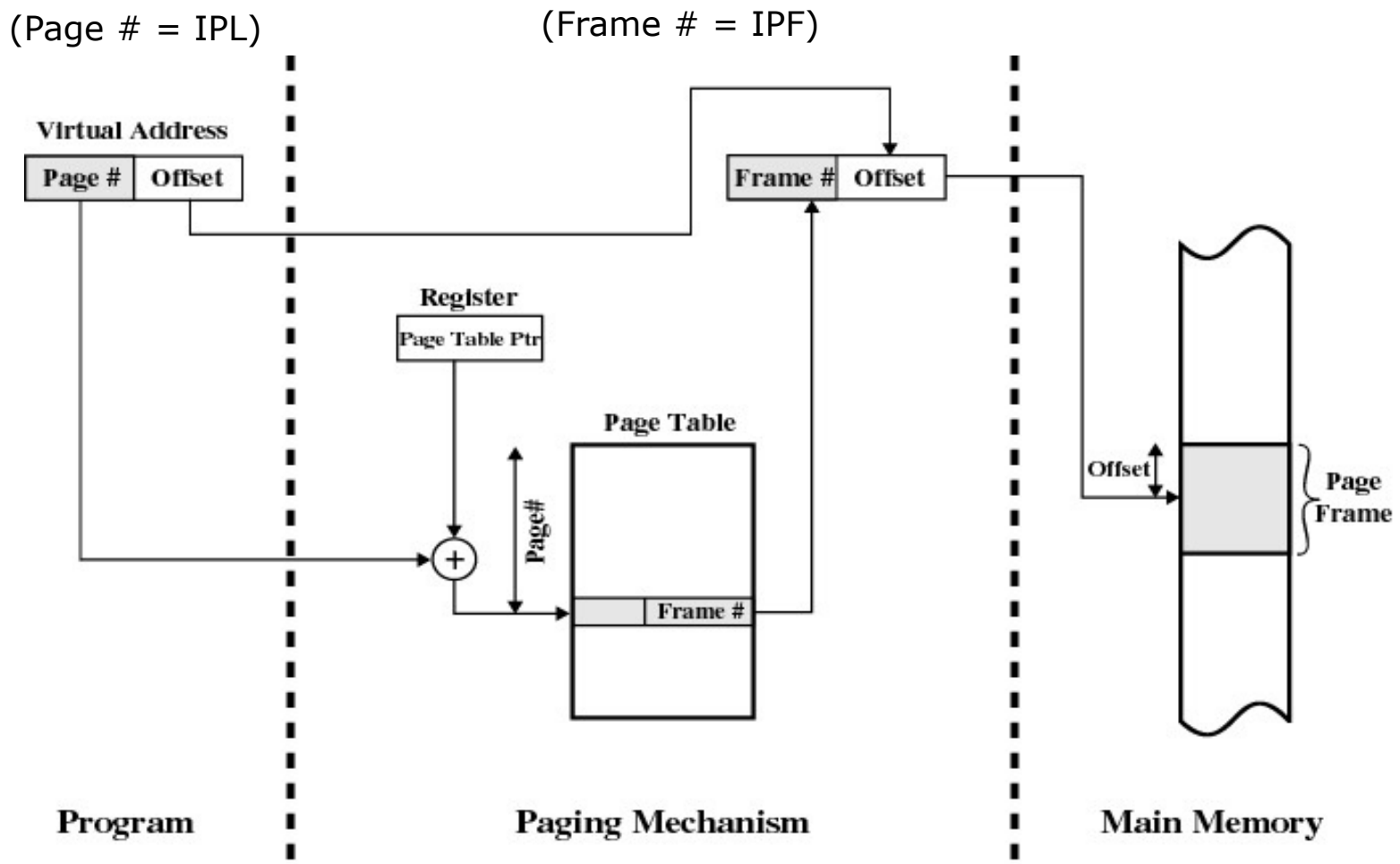
- Esempio di bit di controllo:
  - P = pagina presente nella memoria fisica,
  - M = pagina modificata durante la permanenza in memoria fisica (se non è stata modificata non occorre ricopiarla su disco in caso di rimpiazzo),
  - altri bit di protezione (sola lettura, sola esecuzione, accesso riservato al S.O., …)



Paginazione – versione completa



Sistema con paginazione

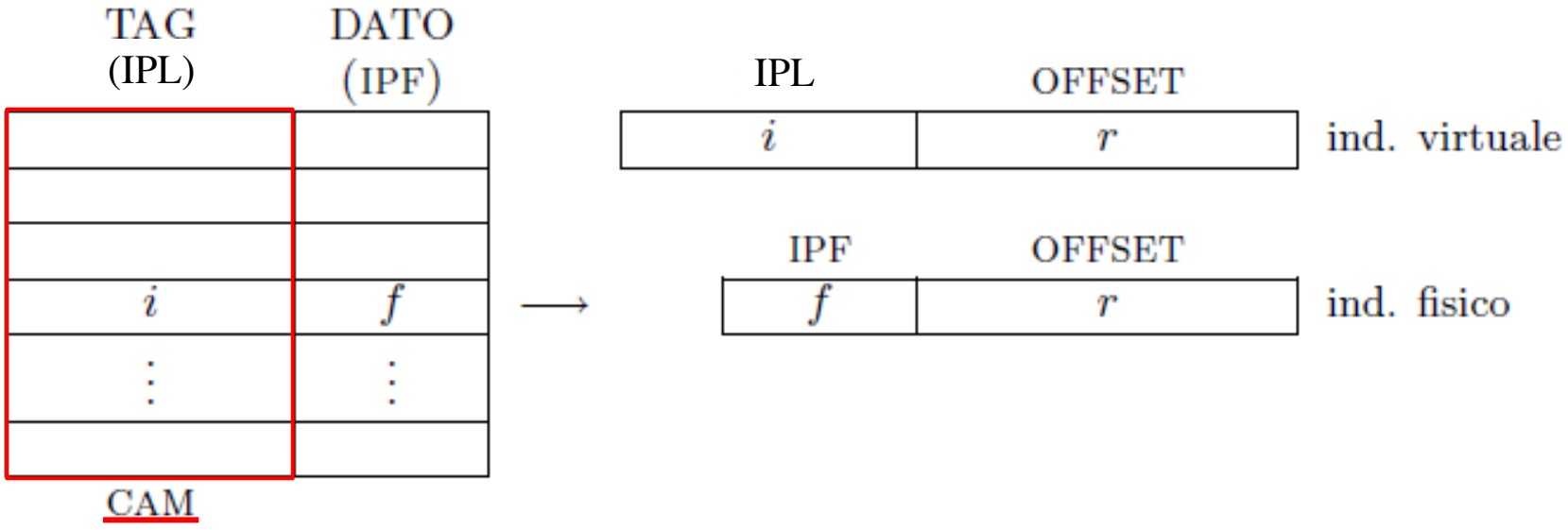


Translation lookaside buffer (TLB)

- Usando la page table (tenuta in memoria) per trasformare un indirizzo virtuale in un indirizzo fisico, c'è il problema che ogni accesso alla memoria ne richiede almeno due:
  - uno per accedere alla page table;
  - uno per accedere al dato.
- così i tempi di accesso alla memoria raddoppiano!
- per limitare questo effetto, si può usare una cache veloce in cui tenere gli elementi della page table usati più di recente; questa cache si chiama Translation Lookaside Buffer (TLB) e ciascun suo elemento contiene:
  - Un numero di pagina logica
  - La corrispondente pagina fisica

Translation lookaside buffer (TLB)

- Per ridurre i tempi di accesso alla page table, è possibile collocare una sua porzione (corrispondente alle pagine fisiche più usate) in una memoria cache chiamata Translation Lookaside Buffer (TLB):



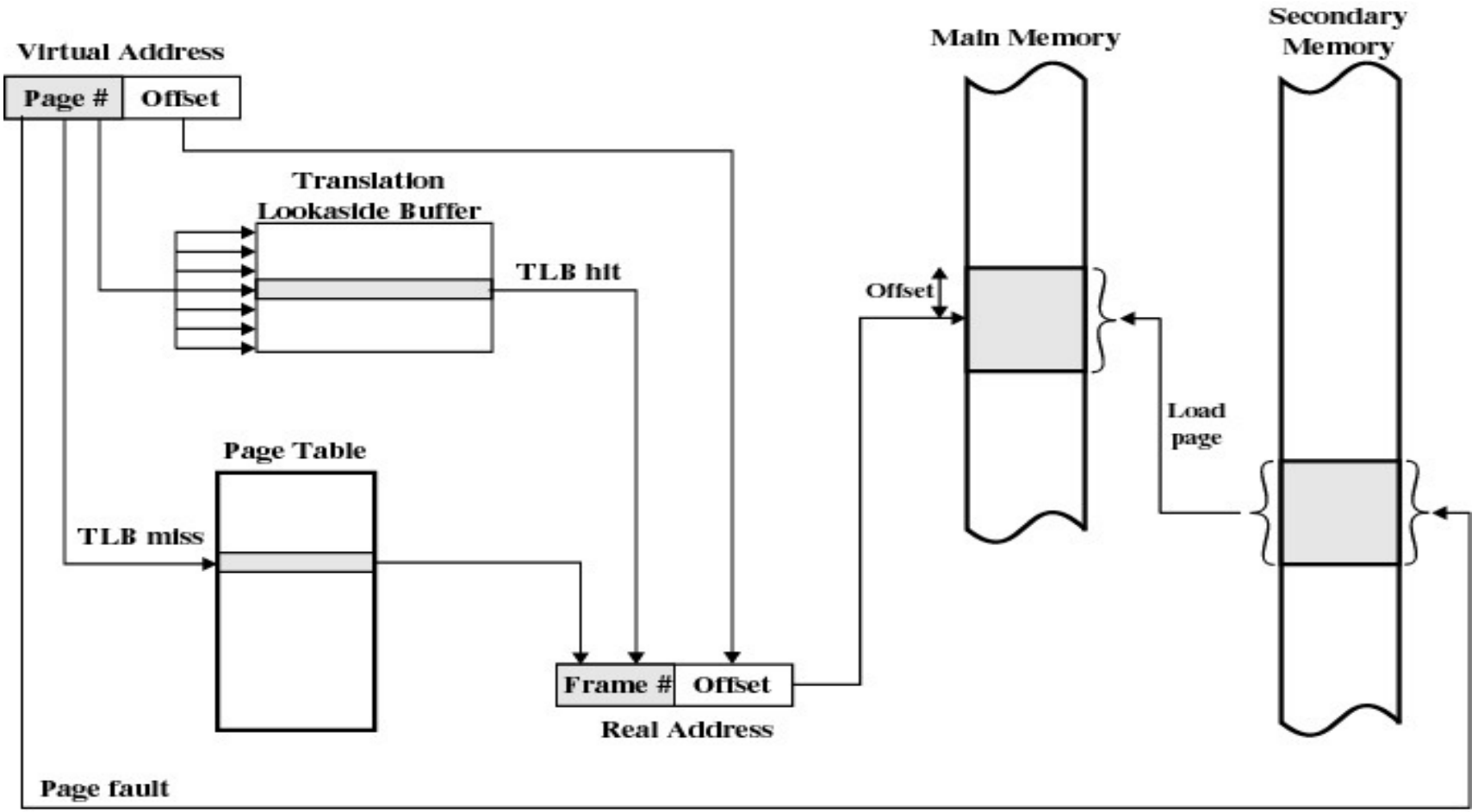
TLB (*Translation Lookaside Buffer*).

Uso del TLB

Dato un indirizzo virtuale, il IPL viene cercato nella memoria associativa (CAM) del TLB:

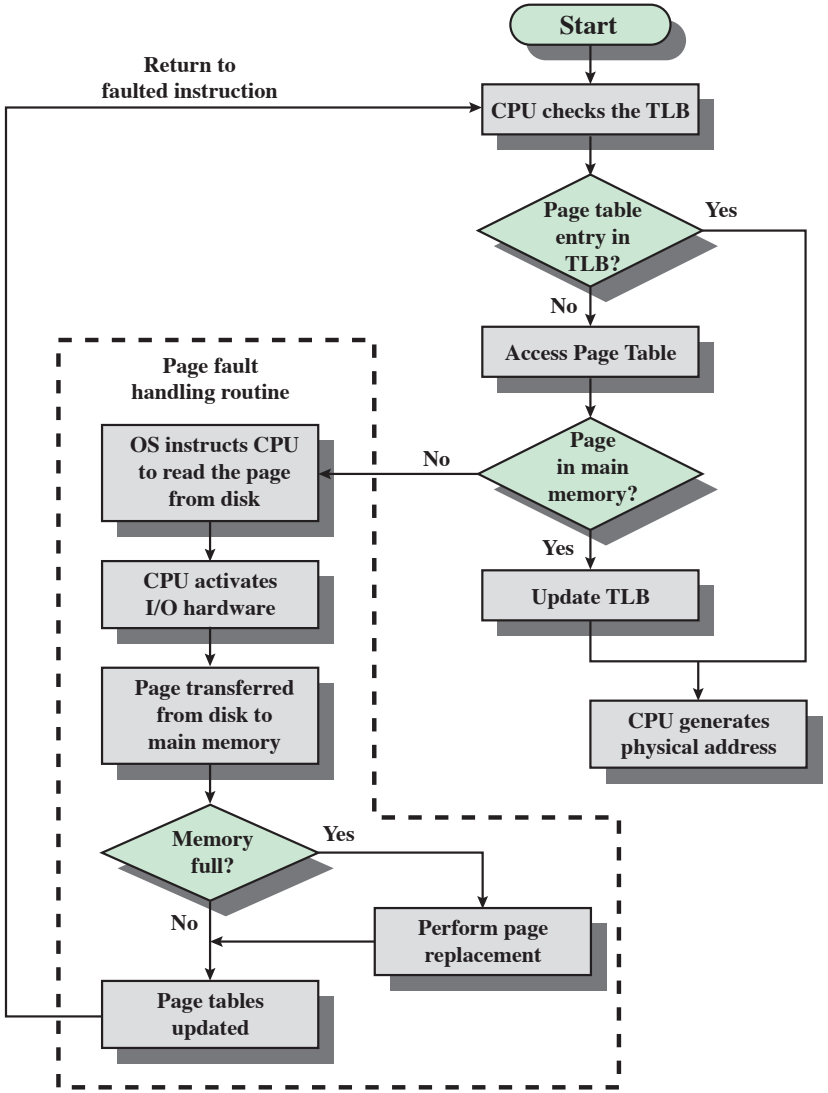
- se trovato (hit), si estrae il corrispondente IPF e si costruisce l'indirizzo fisico;
- se non c'è (miss), si accede alla page table (indice=IPL):
  - se l'elemento della page table indica che la pagina cercata è presente in un frame nella memoria fisica:
    - si costruisce l'indirizzo fisico
    - si aggiorna il TLB inserendo l'elemento trovato;
  - altrimenti si ha un page fault:
    - si porta in un frame la pagina cercata (swap in),
    - si aggiorna la page table (nuova pagina presente),
    - si costruisce l'indirizzo fisico
    - si aggiorna il TLB inserendo il nuovo elemento.

Uso del TLB: schema grafico

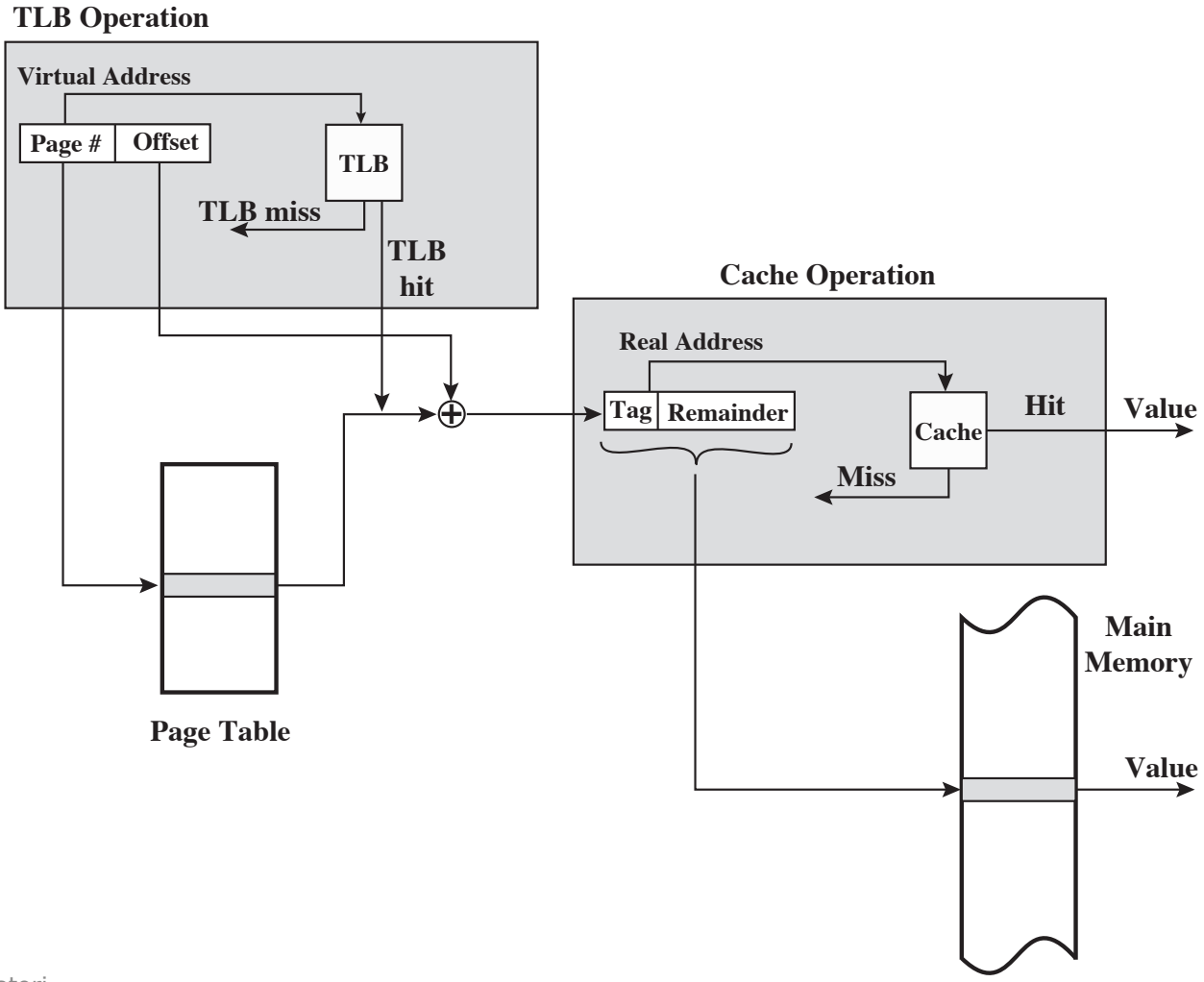


Uso del Translation Lookaside Buffer

Uso del TLB: Ricerca di una Pagina



Uso del TLB e della Cache



MMU: vantaggi della paginazione

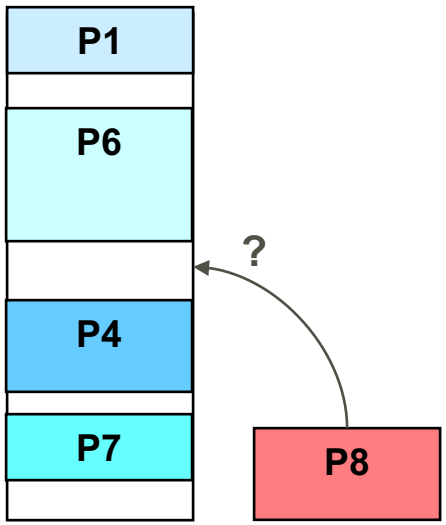
- Protezione (read only),
- Rilocamento,
- Frammentazione dello spazio libero in memoria
- Protezione da interferenze tra programmi,
- Comunicazione tra processi.

Protezione

- Associando alcuni bit di controllo a ciascun elemento della page table, è possibile realizzare nell'hardware opportuni meccanismi di controllo degli accessi alle pagine fisiche:
  - una pagina può essere impostata come read only,
  - ad una pagina può essere consentito l'accesso solo se il processore opera in modo supervisore

Frammentazione dello spazio libero

Via via che alcuni processi terminano la loro esecuzione e liberano aree di memoria e altri vengono caricati in memoria (nelle aree libere) per essere eseguiti, se ogni processi necessita di un'area di memoria contigua, dopo un po' la situazione in memoria sarà quella indicata in figura:



- le aree libere (bianche) sono frammentate,
- pur essendoci complessivamente spazio libero sufficiente a contenerlo, il programma P8 non può essere caricato in memoria,
- un MMU consentirebbe di caricare P8 nelle pagine fisiche libere, ovunque queste si trovino.

Rilocamento

- Il programmatore non sa in quale posizione di memoria il programma sarà caricato quando verrà eseguito.

- La posizione effettiva dipende da:
  - quali e quanti **altri moduli** compongono il programma,
  - quali e quanti **altri programmi** saranno già presenti in memoria nel momento in cui esso verrà caricato.

- Inoltre, se l'area di memoria occupata dal programma durante l'esecuzione, dovesse servire per altri scopi,
  - il programma può dover essere temporaneamente ricopiato su disco (swap),
  - successivamente, quando viene riportato in memoria può venir **collocato in una posizione diversa** (rilocato).

- Il rilocamento è reso facile dalla presenza di un modulo MMU: lo spostamento di una pagina si può ottenere semplicemente modificando la page table.

Protezione da interferenze

- È necessario impedire che un processo acceda a locazioni di memoria di un altro processo (o del sistema operativo);
- Quando il sistema operativo decide di sospendere l'esecuzione di un programma e di passare ad eseguirne un altro, la commutazione di contesto con cui ciò può essere fatto, oltre alla sostituzione di PC, SR, registri, ecc. (previo salvataggio, per consentirne il successivo ripristino), deve prevedere anche la sostituzione (previo salvataggio) del contenuto della page table;
- Così ogni programma può accedere solo alle proprie pagine fisiche: qualsiasi indirizzo (logico) il processore generi, viene mappato su una delle pagine fisiche associate al programma stesso;
- L'hardware impedisce così ad un programma di "vedere" le pagine fisiche che contengono dati e istruzioni degli altri programmi e rende impossibile accedervi.

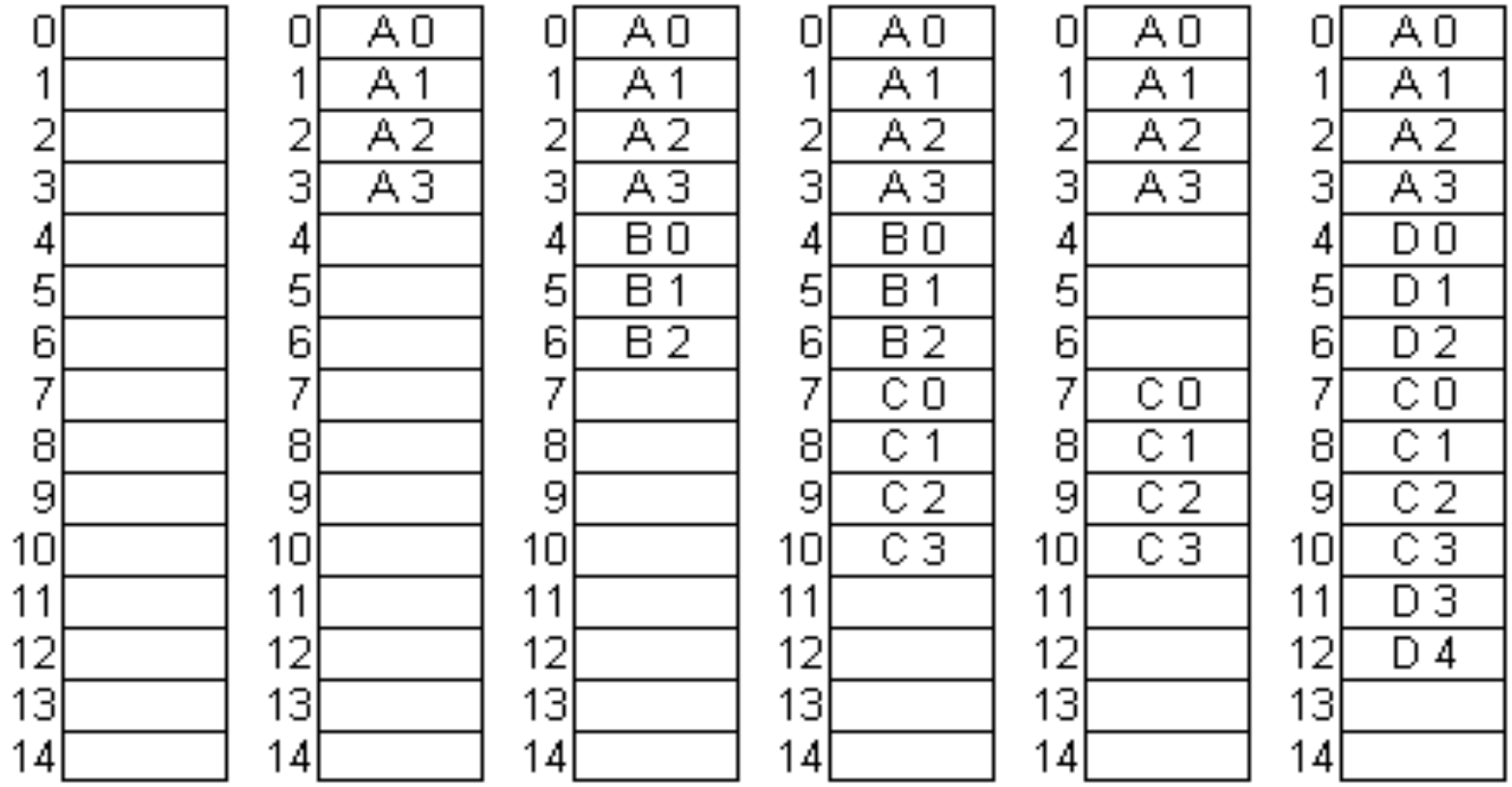
Comunicazione tra processi (condivisione)

- È a volte necessario consentire che processi diversi accedano ad aree di memoria comuni (ad es. dati comuni di processi cooperanti che fanno parte del medesimo programma);

- Nel caso in cui un insieme di subroutine sia utilizzato da più processi, conviene che di esse vi sia in memoria un'unica copia (rientrante), accessibile a tutti, piuttosto che averne tante copie replicate per ciascun processo;

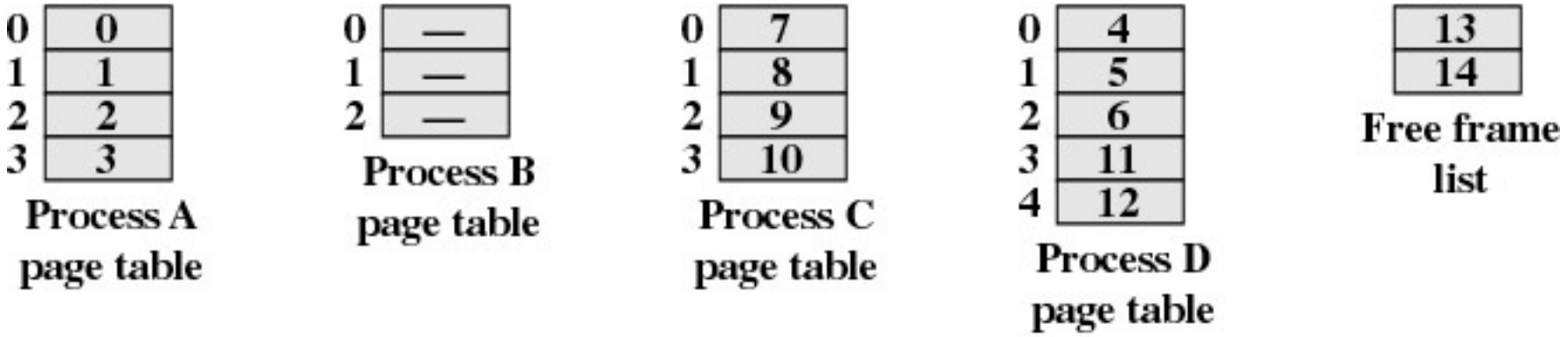
- Per consentire a più processi di condividere una pagina fisica è sufficiente che il suo IPF sia presente nella page table di ciascuno di quei processi.

Esempio di paginazione con più processi



- Il processo A ha 4 pagine; B ne ha 3, C ne ha 4, D ne ha 5;
- 3 pagine di D usano i frame liberati dallo swap-out di B.

Esempio di page table con più processi



Page table all'istante (f)

Segmentazione

- I programmi sono costituiti da **moduli**, che vengono scritti e compilati separatamente.
- Per il programmatore il modo più naturale di vedere la memoria non è uno spazio di indirizzi lineare (da 0 fino all'indirizzo massimo), ma è costituito da tanti spazi di indirizzi separati (uno per ciascun modulo).
- Se il sistema operativo e l'hardware consentono di gestire in forma di moduli gli indirizzi dei programmi e dei dati, allora diventa facile:
  - associare ai diversi moduli diversi livelli di protezione (read-only, execute-only, …),
  - organizzare la condivisione di moduli.
- La tecnica più appropriata per la gestione a moduli degli indirizzi è costituita dalla **segmentazione**.

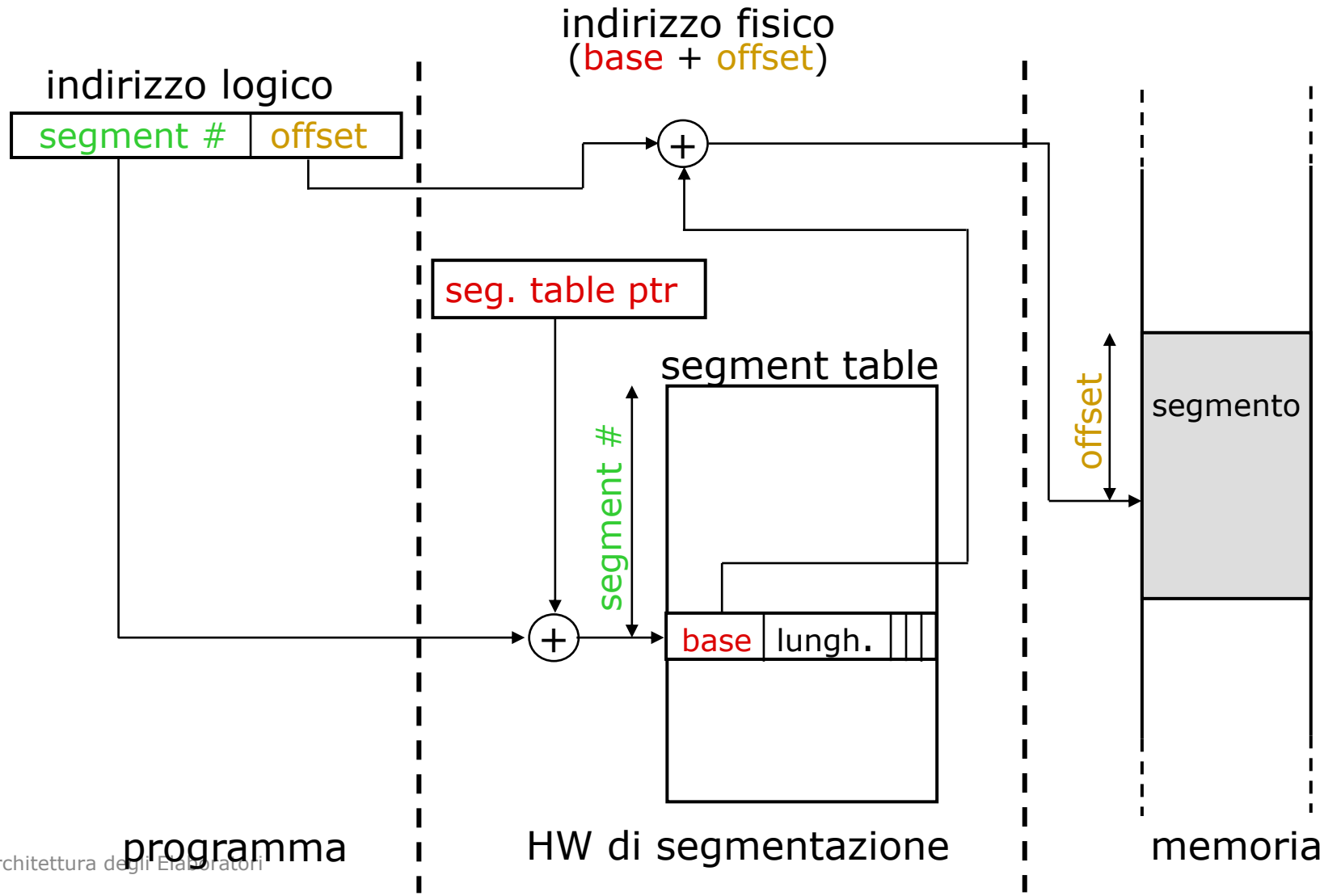
Segmentazione

- I moduli (segmenti) di un processo sono in genere di dimensioni diverse.
- C'è una lunghezza massima per i segmenti.
- Un indirizzo è costituito da due parti: un **indice di segmento** e un **offset.**
- A ciascun processo è associato un insieme di segmenti (uno per ciascuno dei suoi moduli) e una **segment table** contenente, per ciascuno dei suoi segmenti, le seguenti informazioni:
  - l'indirizzo iniziale del segmento (inserito dal SO quando il segmento viene caricato in memoria),
  - la dimensione del segmento,
  - bit di controllo (presente, modificato, …),
  - bit di protezione (read only, permessi, …).

Da indirizzo logico a indirizzo fisico:

- L'indirizzo logico ha due componenti:
  - l'indice di segmento IS individua un elemento della segment table da cui si estrae:
    - l'indirizzo iniziale del segmento (IIS),
    - la dimensione del segmento (DS),
  - l'offset OS viene confrontato con la dimensione DS:
    - se OS $>$ DS l'indirizzo non è valido ($\rightarrow$ eccezione),
    - se OS $\leq$ DS l'indirizzo è valido;
- L'indirizzo fisico è dato da IIS + OS
  - indirizzo iniziale del segmento (estratto dalla segment table) + l'offset (estratto dall'indirizzo logico).

Uso della segment table



Esempio di segmentazione

segm. A0 (codice di A)	segmento A1 (dati di A)	segm. B0 (cod. B)	segmento B1 (dati di B)	
0x250 Byte	0x754 Byte	0x236 Byte	0x348 Byte	

segment table del processo A:

indice segmento	indirizzo iniziale	lunghezza segmento	bit di contr.	bit di prot.
0	0x0	0x250	p	rx
1	0x250	0x754	pm	rw

segment table del processo B:

indice segmento	indirizzo iniziale	lunghezza segmento	bit di contr.	bit di prot.
0	0x9A4	0x236	p	rx
1	0xBDA	0x348	pm	rw

Esempio di uso della segment table

- un indirizzo logico (da 20 bit) sia costituito da:
  - 4 bit (indice di segmento) + 16 bit (offset);
- Ogni processo può avere:
  - al massimo 16 segmenti,
  - la dimensione massima di un segmento è di 64KB;

segment table del processo A:

indice segmento	indirizzo iniziale	lunghezza segmento	bit di contr.	bit di prot.
0	0x0	0x250	p	rx
1	0x250	0x754	pm	rw

- indirizzo logico del processo A:

0xI 0x064E

- per ottenere il corrispondente indirizzo fisico:
  - indice di segmento = 1; offset = 0x064E (< 0x754: OK)
  - indirizzo fisico = 0x250 + 0x64E = 0x89E

Segmentazione e paginazione

- Come per la paginazione, è possibile spostare segmenti non usati nella memoria secondaria.

- La segmentazione è visibile al programmatore, mentre la paginazione no.
  - Inoltre la segmentazione può causare la frammentazione della memoria se i segmenti sono molto diversi tra loro.

- La struttura regolare della paginazione rende l'approccio più interessante dal punto di vista hardware

- E' possibile avere un sistema ibrido paginazione-segmentazione.

Segmentazione + Paginazione

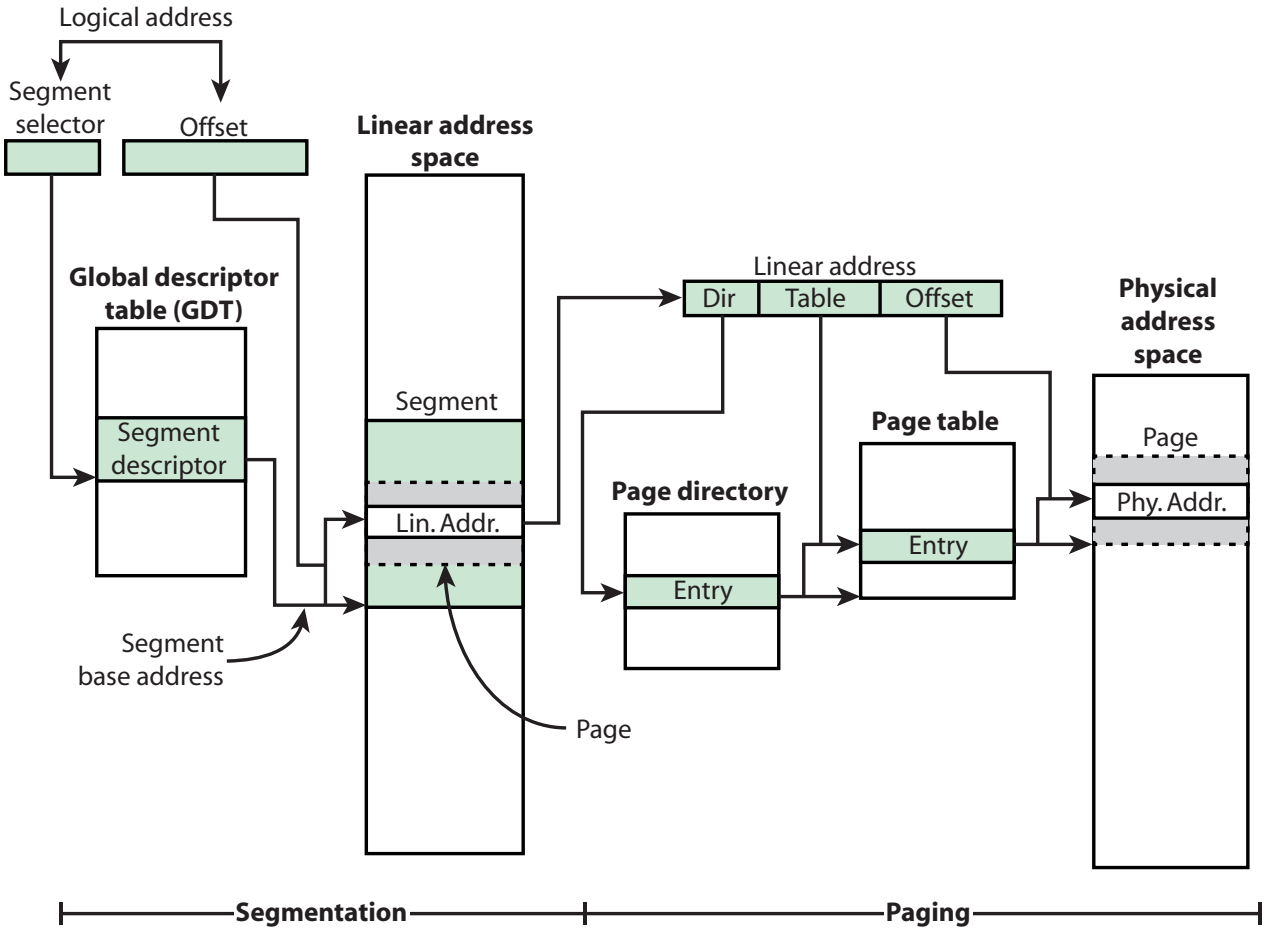


Figure 8.21 Intel x86 Memory Address Translation Mechanisms