La Memoria Virtuale

Introduzione alla gestione della memoria
Memoria virtuale
Paginazione
Segmentazione

Memoria Virtuale

- **La memoria può essere *virtualizzata* allo scopo di:**
  - *Far condividere lo stesso spazio di memoria fisica a più programmi*
  - *Far vedere a ciascun programma uno spazio di indirizzamento più ampio di quello effettivamente disponibile*
- **Spazio di indirizzamento virtuale**
  - Dipende esclusivamente dalle modalità di indirizzamento:
  - **ES**. indirizzi a 16 bit . 64k
- **Spazio di indirizzamento fisico**
  - Dipende dalle dimensioni effettive delle memoria

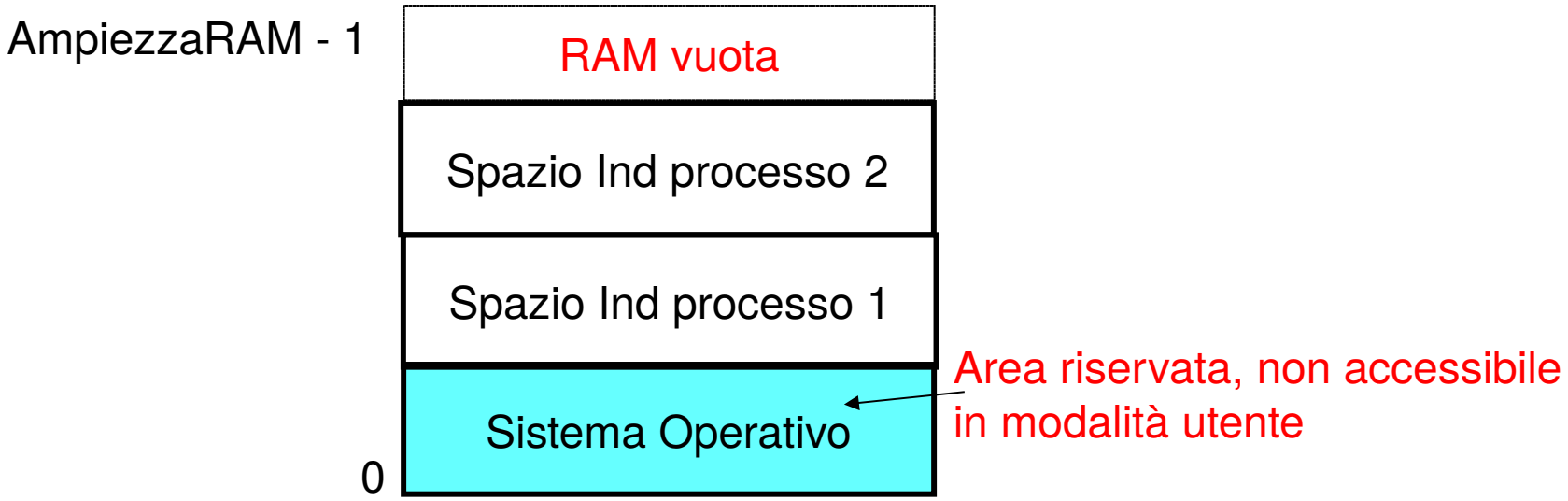
Principi di funzionamento

- I programmi indirizzano lo *spazio virtuale*
- L'intero spazio virtuale è allocato in memoria secondaria
- Si mantiene in memoria centrale solo un sottoinsieme dello *spazio virtuale*
- Si trasferiscono man mano solo le parti dello spazio virtuale alle quali il programma fa riferimento
- Gestione completamente *trasparente* da parte del Sistema Operativo
- **Problemi**
  - Traduzione degli indirizzi
  - Efficienza del trasferimento tra disco e memoria centrale

Gestione della memoria

- Tutti i programmi che compongono il SO ed i programmi applicativi attivi usano contemporaneamente la RAM
- Il *gestore della memoria* si preoccupa di fare condividere la RAM ai vari processi in esecuzione in modo che :
  - ogni processo abbia il suo spazio *privato* distinto dagli altri (e inaccessibile agli altri)
  - ogni processo abbia abbastanza memoria per eseguire il proprio algoritmo e raccogliere i suoi dati

Gestione della memoria (2)

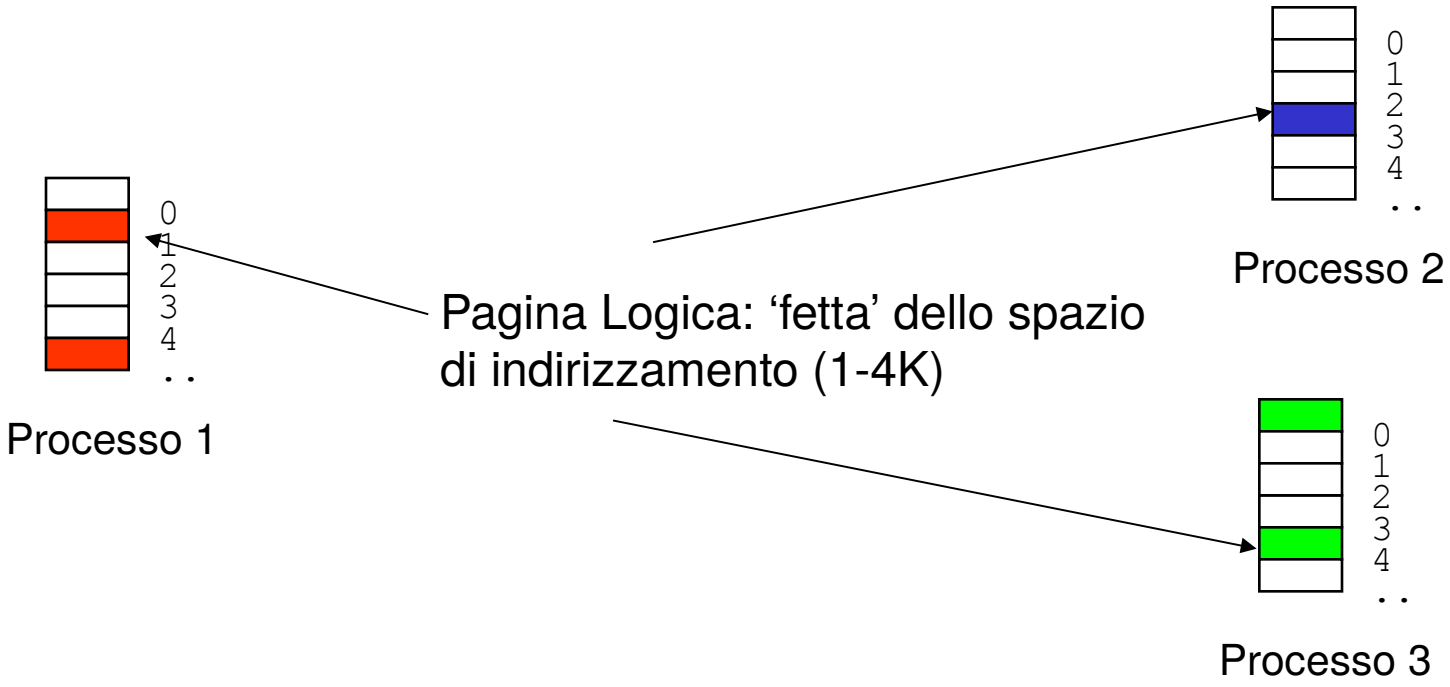


Gestione della memoria (3)

- Problemi:
  - limite all'ampiezza dello SI
    - Attualmente ogni processo usa almeno 4GB di spazio di indirizzamento, con RAM assai più piccole…
  - memoria sottoutilizzata
    - aree dello spazio di indirizzamento che non sono utilizzate occupano RAM (es. gap fra heap e stack)
- Soluzione: memoria virtuale
  - ad ogni istante carico in RAM solo le parti di SI che servono per una certa fase dell'esecuzione
  - diverse soluzioni: vedremo la *paginazione*

Paginazione: idea base (1)

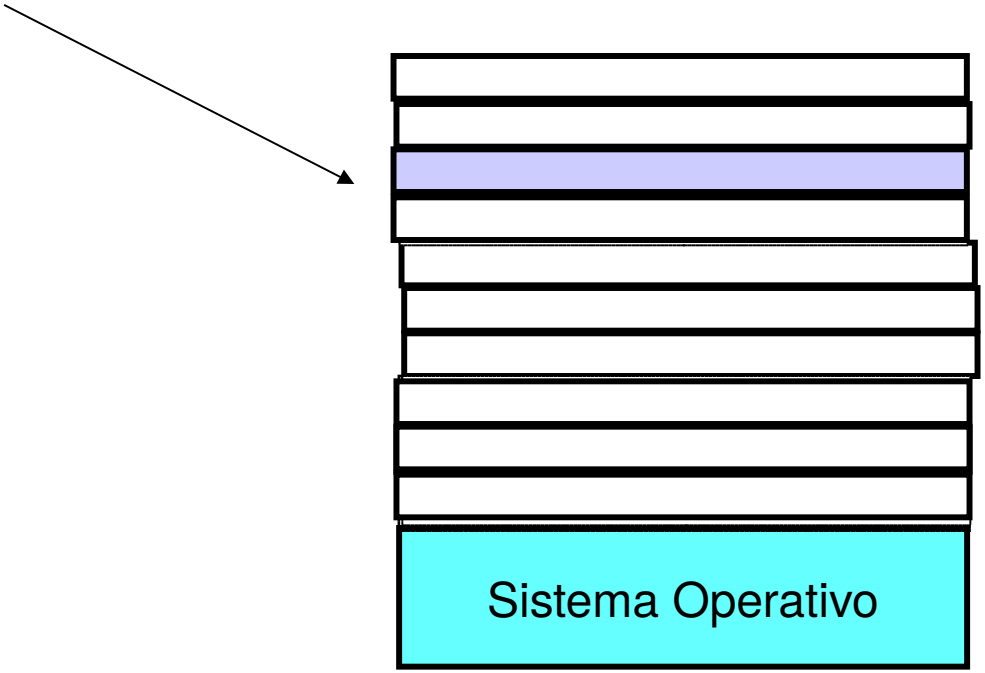
- Lo spazio di indirizzamento di ogni processo è diviso in ‘fette’ (*pagine logiche*) tutte della stessa ampiezza



Paginazione: idea base (2)

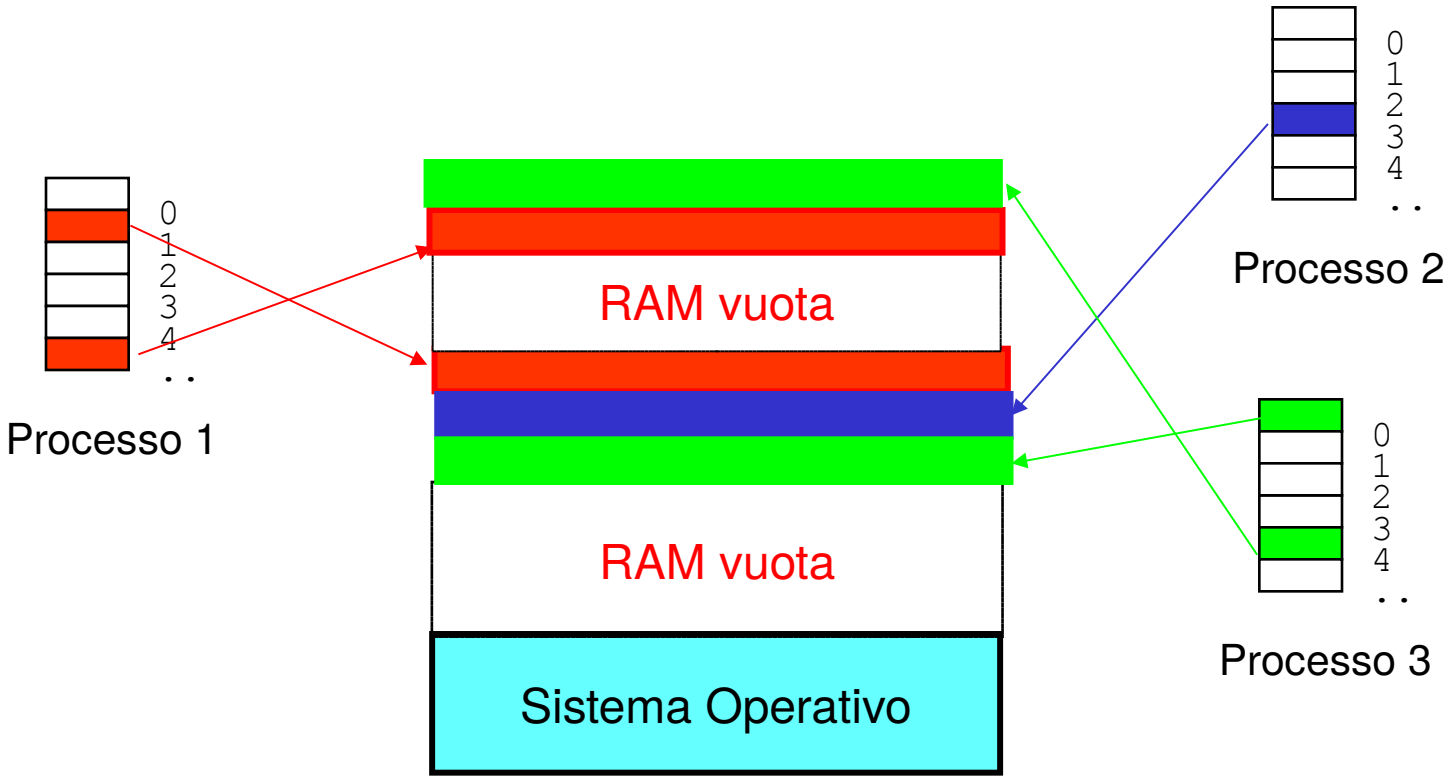
- Anche la RAM disponibile per i processi utente è divisa in pagine della stessa ampiezza (*pagine fisiche*)

Pagina Fisica: 'fetta' dello RAM
stessa ampiezza della pagina logica



Paginazione: idea base (3)

- Ad ogni istante solo le pagine necessarie per i processi in esecuzione sono caricate in RAM (si usa la località!)

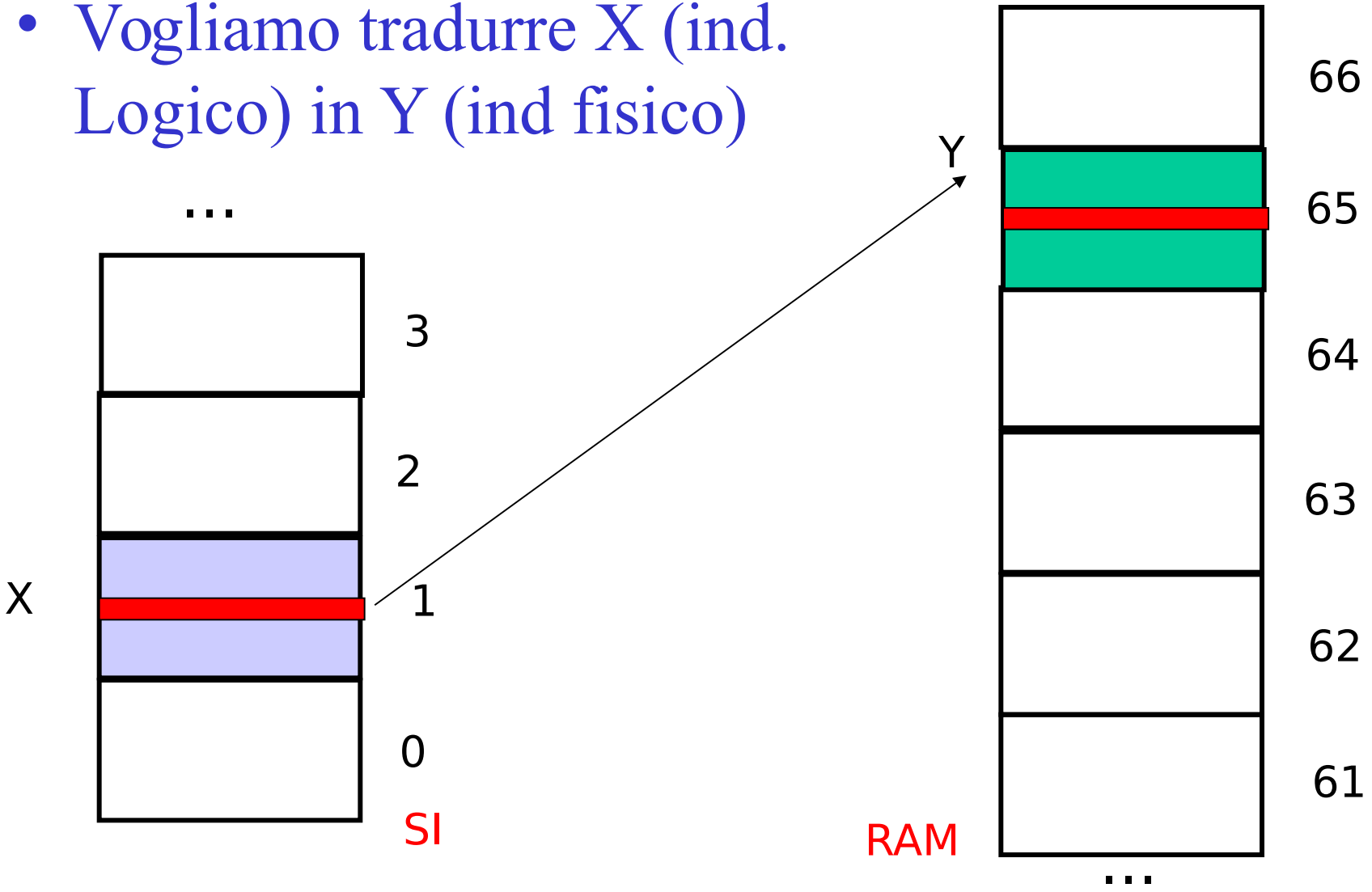


Paginazione: problema

- Esecuzione (corretta) dei programmi utente parzialmente caricati:
  - tradurre correttamente l'indirizzo logico X (relativo allo spazio di indirizzamento) nell'indirizzo fisico Y (parola di RAM) in cui è caricato
  - bloccare automaticamente accessi ad aree non caricate in RAM (*page fault*)
  - aggiornare automaticamente l'insieme delle pagine in memoria
    - scaricando/caricando da memoria secondaria quelle necessarie in una certa fase

Traduzione indirizzi ...

- Vogliamo tradurre X (ind. Logico) in Y (ind fisico)



Traduzione indirizzi (2)

- Osservate che

  X = #pagLogica * s + offset

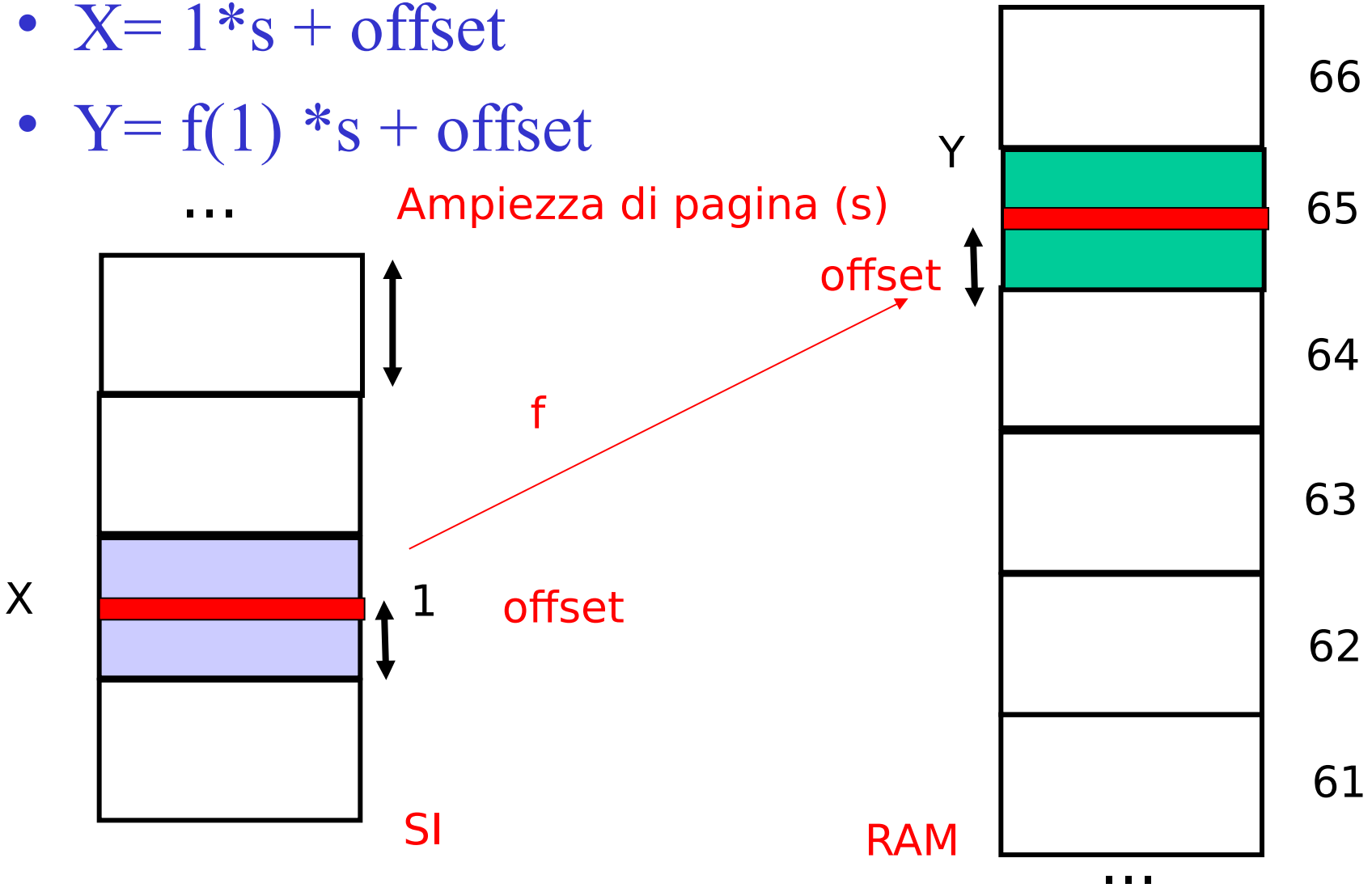
  - s ampiezza della pagina (logica e fisica)
  - offset indirizzo dentro la pagina (fra 0 ed s-1)
- quindi

  Y= f(#pagLogica) *s + offset

  - f() funzione che associa ad ogni pagina logica il numero di pagina fisica in cui è caricata
  - NB f() è definita solo per le pagine caricate

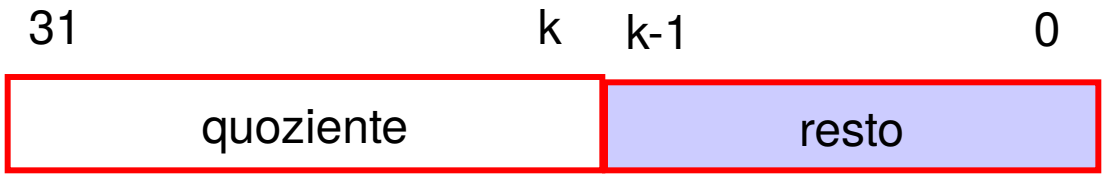
Traduzione indirizzi (3)

- X= 1*s + offset
- Y= f(1) *s + offset



Traduzione indirizzi (4)

- La traduzione degli indirizzi deve essere veloce!
  - Va fatta ad ogni accesso in memoria
- Come si calcolano `#pagLogica` e `offset` ?
  - Sono quoziente e resto della divisione per `s`
  - in generale la divisione è molto costosa!
  - È semplice se $s=2^k$ perché stiamo lavorando con indirizzi binari



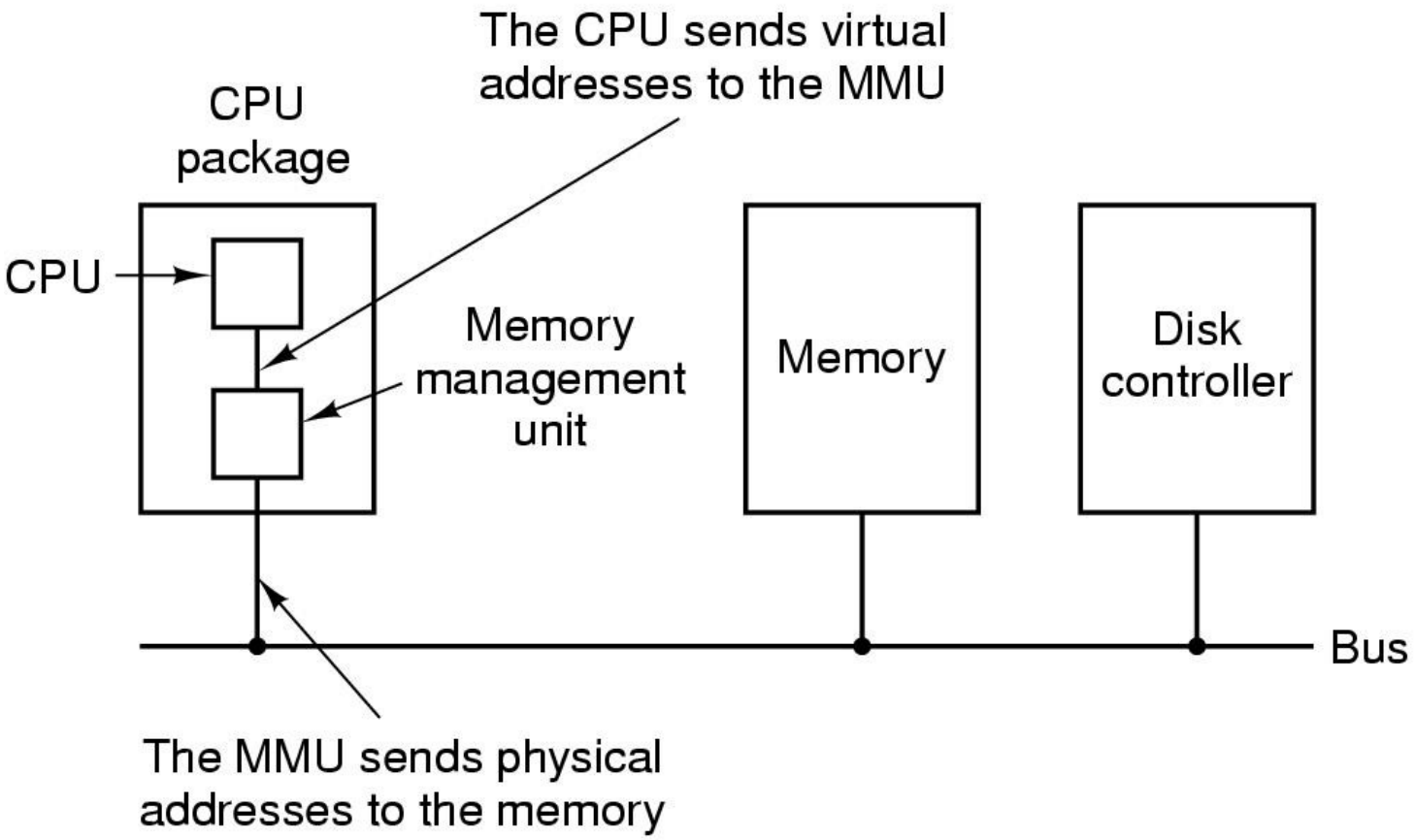
Traduzione indirizzi (5)

- Quindi il calcolo è veloce
  - es. pagine di 4KB= 2^{12}B basta selezionare (hw) i primi 12 bit per offset ed il resto per #pagLogica
- Come si calcola la funzione di corrispondenza f() ?
  - Serve una tabella (la *tabella delle pagine, TP*)
  - TP[#pagLogica] è il *descrittore di pagina* e contiene
    - il numero di pagina fisica corrispondente
    - bit Presente-Assente (se la pagina è in memoria o no)
    - altro

Traduzione indirizzi (6)

- Cosa succede se la pagina non è in memoria?
  - Presente-Assente = 0, si genera un *page fault*
  - l'esecuzione del processo viene bloccata
  - va in esecuzione il *gestore della memoria*
  - la pagina logica viene localizzata su disco e caricata in RAM
  - se la RAM è piena si individua una *pagina vittima* da scaricare dalla RAM
  - *algoritmi di rimpiazzamento* : servono a selezionare la pagina vittima

La Memory Management Unit



Organizzazione tipica dell'hw: posizione e funzione della MMU

La MMU (2)

Operazioni di una MMU con 16 pagine di 4KB

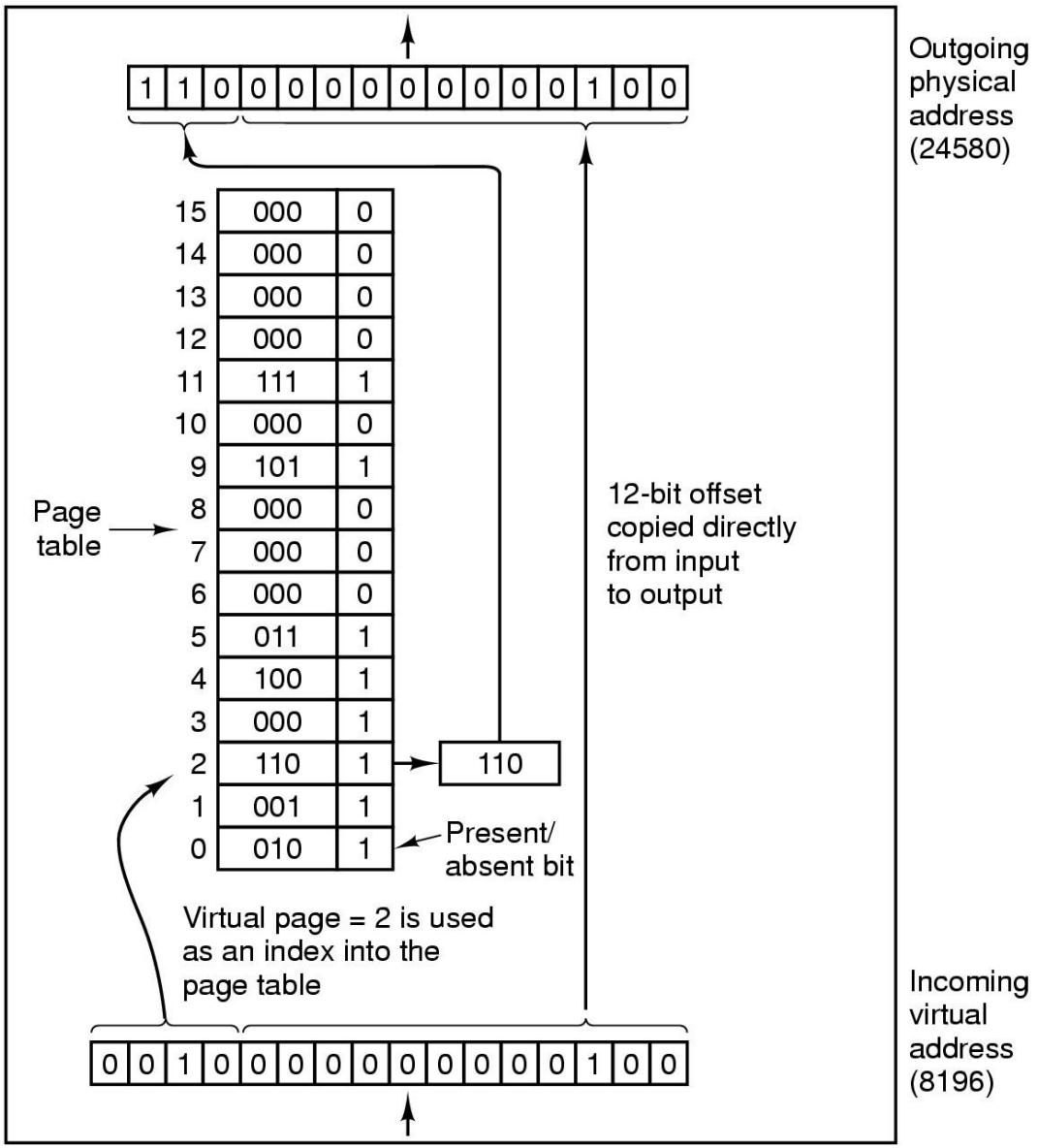
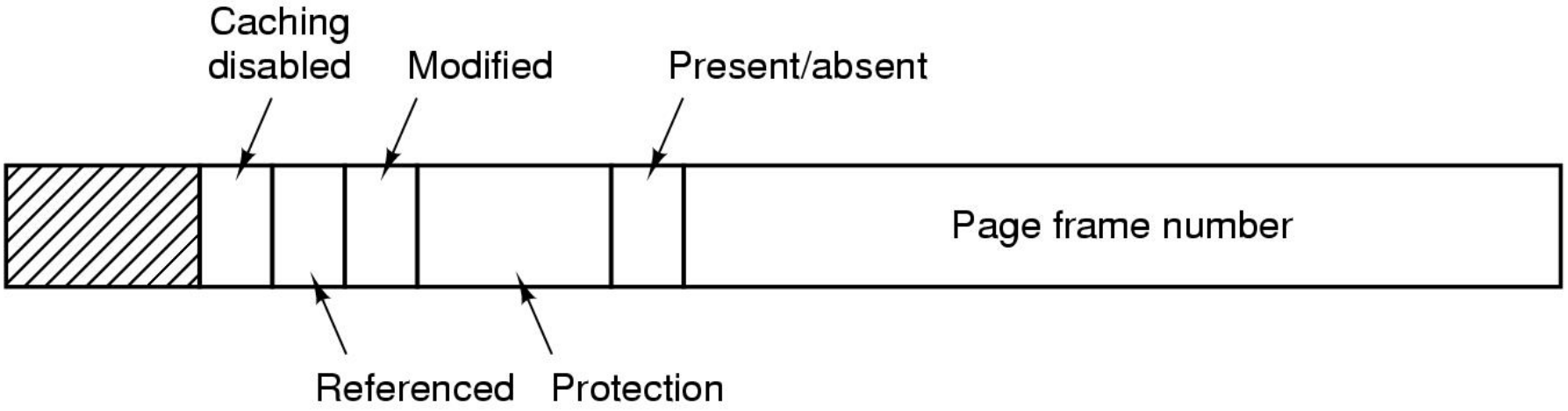


Tabella delle Pagine



- Informazioni contenute in un descrittore di pagina
  - il formato dipende dall'hw
  - NON ci sono informazioni su dove trovare la pagina su disco (dipende dal SO)

Traduzione indirizzi (6)

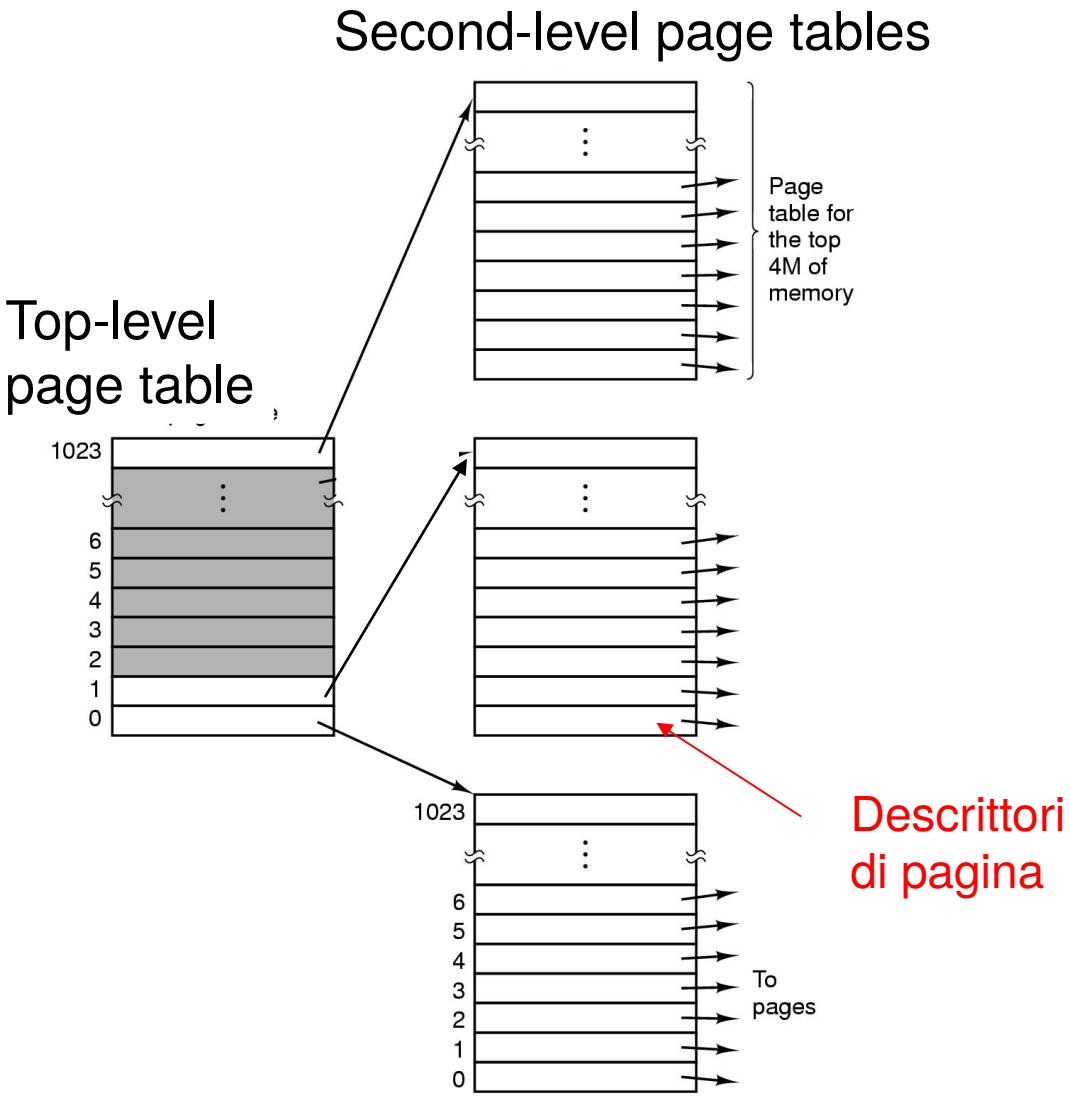
- L'accesso alla tabella delle pagine deve essere veloce
  - non può stare solo in RAM
    - se no duplica il tempo di accesso
  - non può stare tutta in MMU
    - prenderebbe troppo spazio
    - indirizzi a 32 bit e pagine di 4K, la size(TP)=2^{20} record
    - non tutti i descrittori servono contemporaneamente
  - si usa una piccola cache dei descrittori in MMU
    - TLB (Translation Lookaside Buffer) o memoria associativa
    - tutta la tabella è in RAM

TLB - *Translation Lookaside Buffer* o Memoria Associativa

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

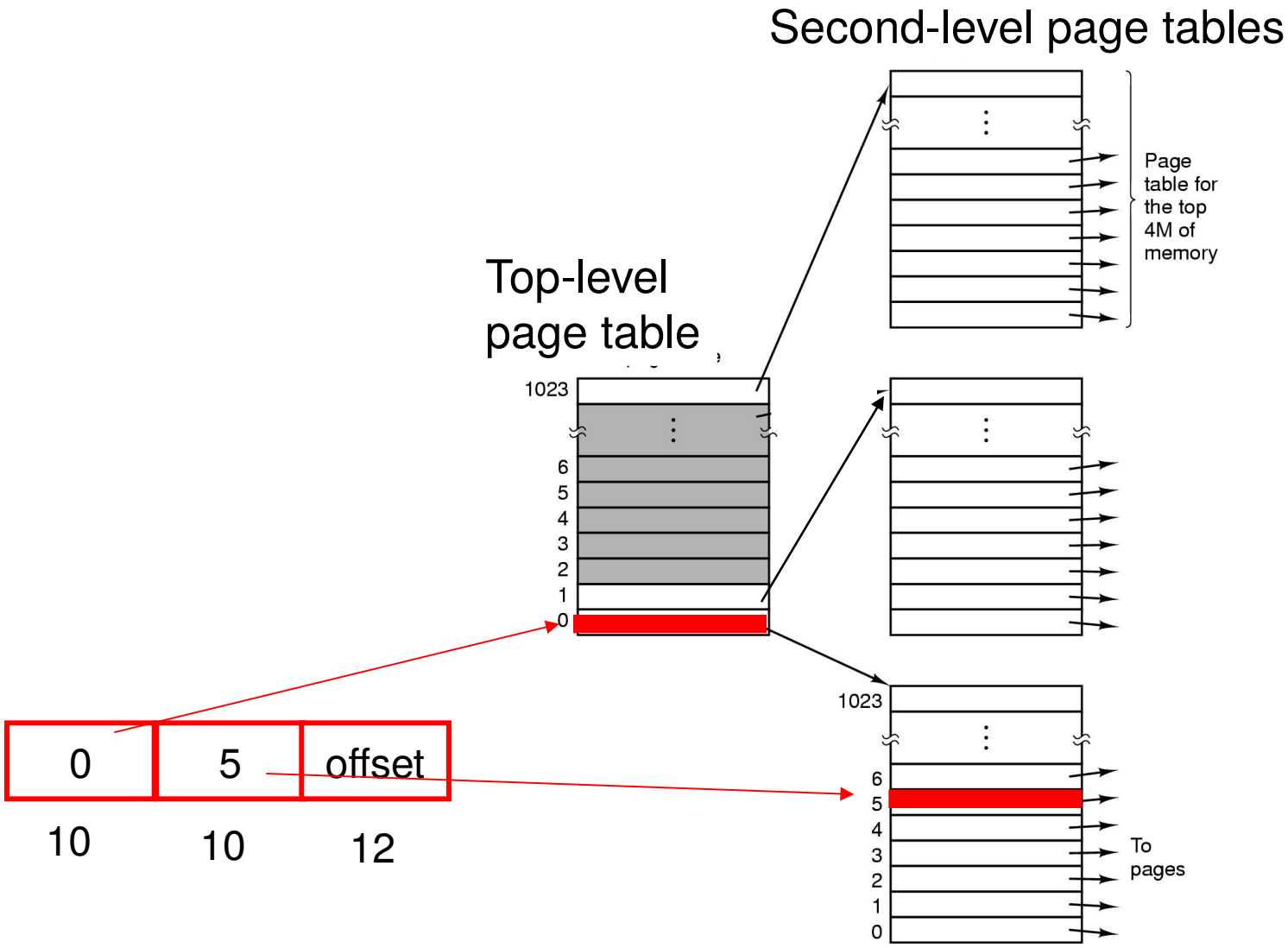
Esempio di TLB

Tabella delle Pagine a due livelli



PT1	PT2	offset
10	10	12

Tabella delle Pagine a due livelli (2)



Implementazione della Paginazione

Il Sistema operativo invoca i meccanismi di paginazione in quattro circostanze:

1. Creazione di un Processo
   - Determina la dimensione dello spazio di indirizzamento
   - Crea la tabella delle pagine
2. Esecuzione di un Processo (*context switch*)
   - Reset della MMU per il nuovo processo
   - Aggiornamento del TLB (*flush*)
3. Page fault
   - Determina l'indirizzo logico che ha causato il page fault
   - Sposta una pagina su disco (se necessario) e carica la pagina richiesta
4. Terminazione di un Processo
   - Dealloca la tabella delle pagine e le pagine del processo

Gestione del Page Fault (1)

1. Una eccezione provoca l'invocazione del nucleo, salvando almeno il PC sullo stack (hw)
2. Salvataggio registri generali e altri reg. interni (assembler)
3. Il sistema determina la pagina logica richiesta
   - registro speciale o software
4. Il sistema verifica la validità dell'indirizzo, e ricerca una pagina libera o, in alternativa, una pagina vittima
5. Se la pagina vittima selezionata è stata modificata (dirty), viene scritta su disco

Gestione del Page Fault (2)

6. Il sistema richiede la lettura della pagina logica dal disco
   - (schedulatore) va in esecuzione un altro processo pronto
7. Quando la lettura è completata (interruzione), si aggiorna la tabella delle pagine
8. Viene ripristinata l'istruzione che ha causato il page fault
9. Il processo che ha causato il page fault viene schedulato
10. Ripristino dei registri, ritorno in modo utente (assembler)
    - Il processo riprende l'elaborazione come se il page fault non fosse avvenuto