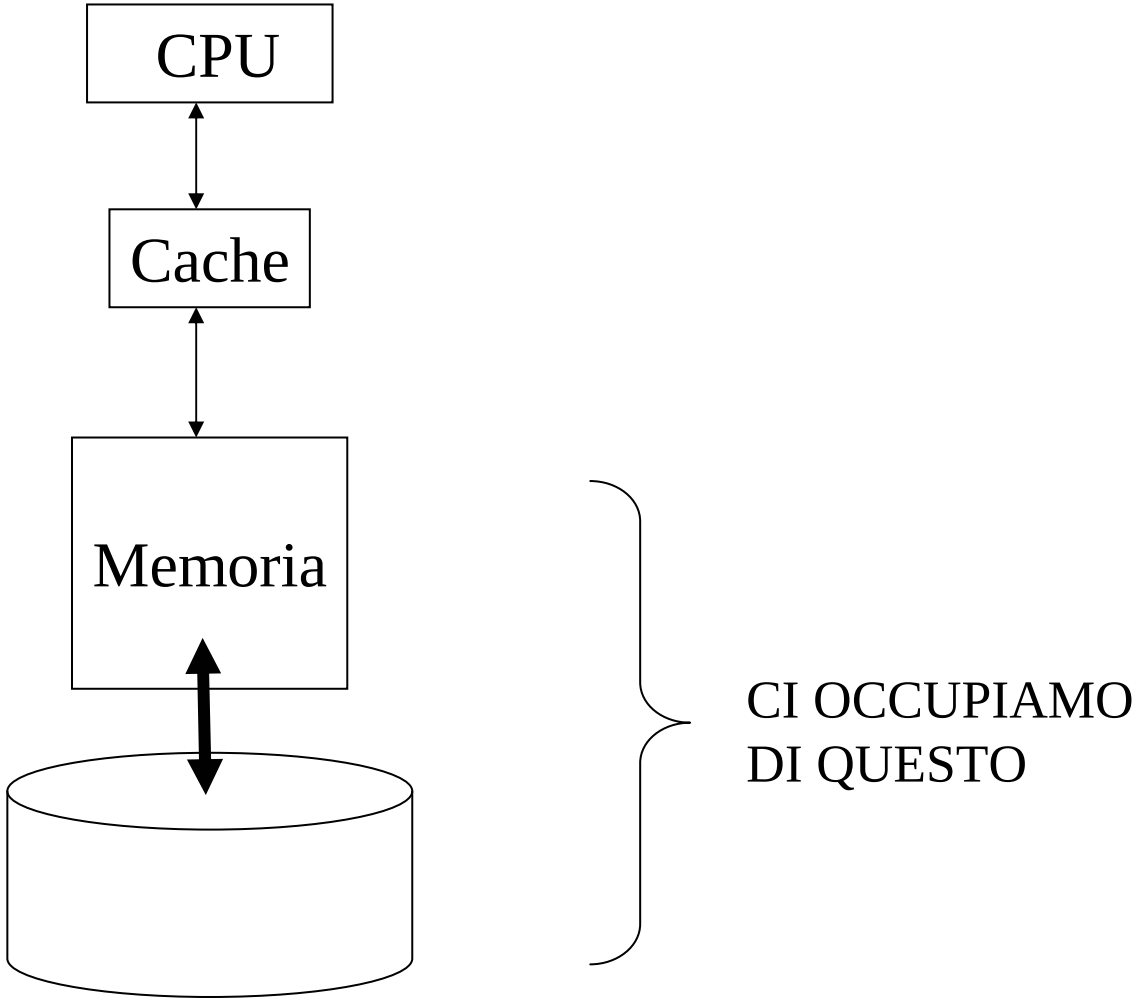
Calcolatori Elettronici B
a.a. 2006/2007

MEMORIA VIRTUALE

Massimiliano Giacomin

Gerarchia di memorie



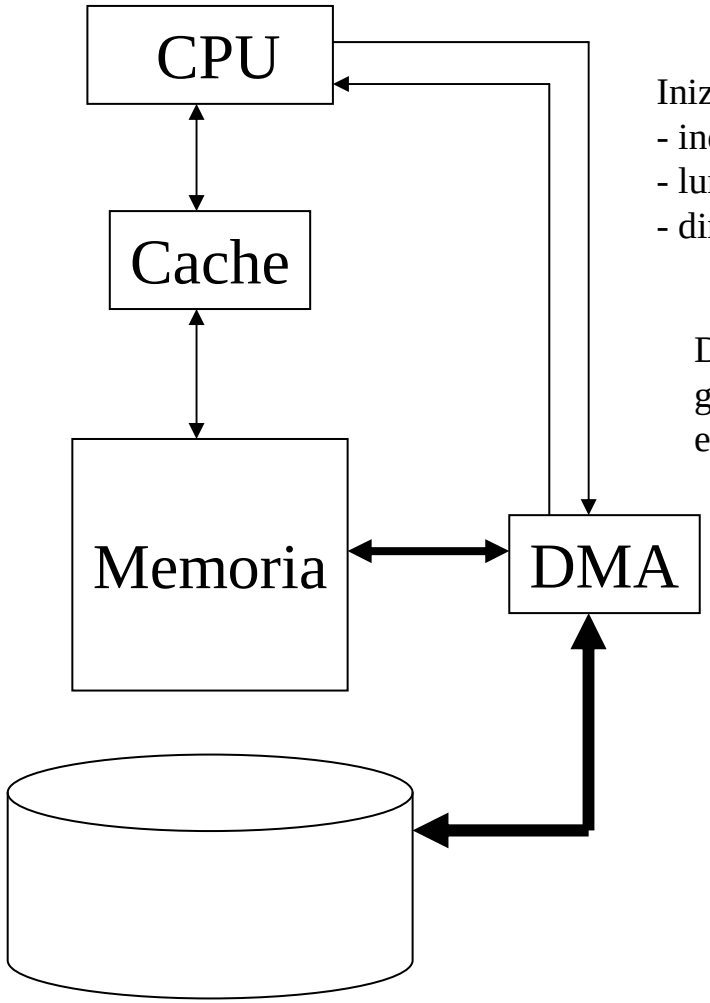
Richiami sulla gestione dell'I/O

Modalità di gestione e sincronizzazione con i dispositivi

- **A controllo di programma (o "programmato")**
  interfacce di I/O controllate direttamente dal programma mediante cicli di attesa
- **Interrupt**
  ogni volta che un dispositivo è "pronto" (per leggere o scrivere il dato) può interrompere la normale esecuzione del processore (interrupt)
- **Accesso Diretto alla Memoria** (Direct Memory Access, DMA)
  un dispositivo dedicato è in grado di gestire autonomamente il trasferimento di un blocco di dati, sollevando da questo compito il processore (che deve solo provvedere all'inizializzazione del dispositivo DMA e che viene interrotto solo alla fine del trasferimento)

RIVEDIAMO IN DETTAGLIO DMA (caso trasferimento disco – memoria):
ha un ruolo chiave nella gestione della memoria virtuale

DMA (Direct Memory Access)



Inizio CPU invia
- indirizzo di partenza
- lunghezza del blocco da trassferire
- direzione (I/O)

DMA gestisce trasferimento inviando gli opportuni comandi a dispositivo e memoria [modo “programmato”]

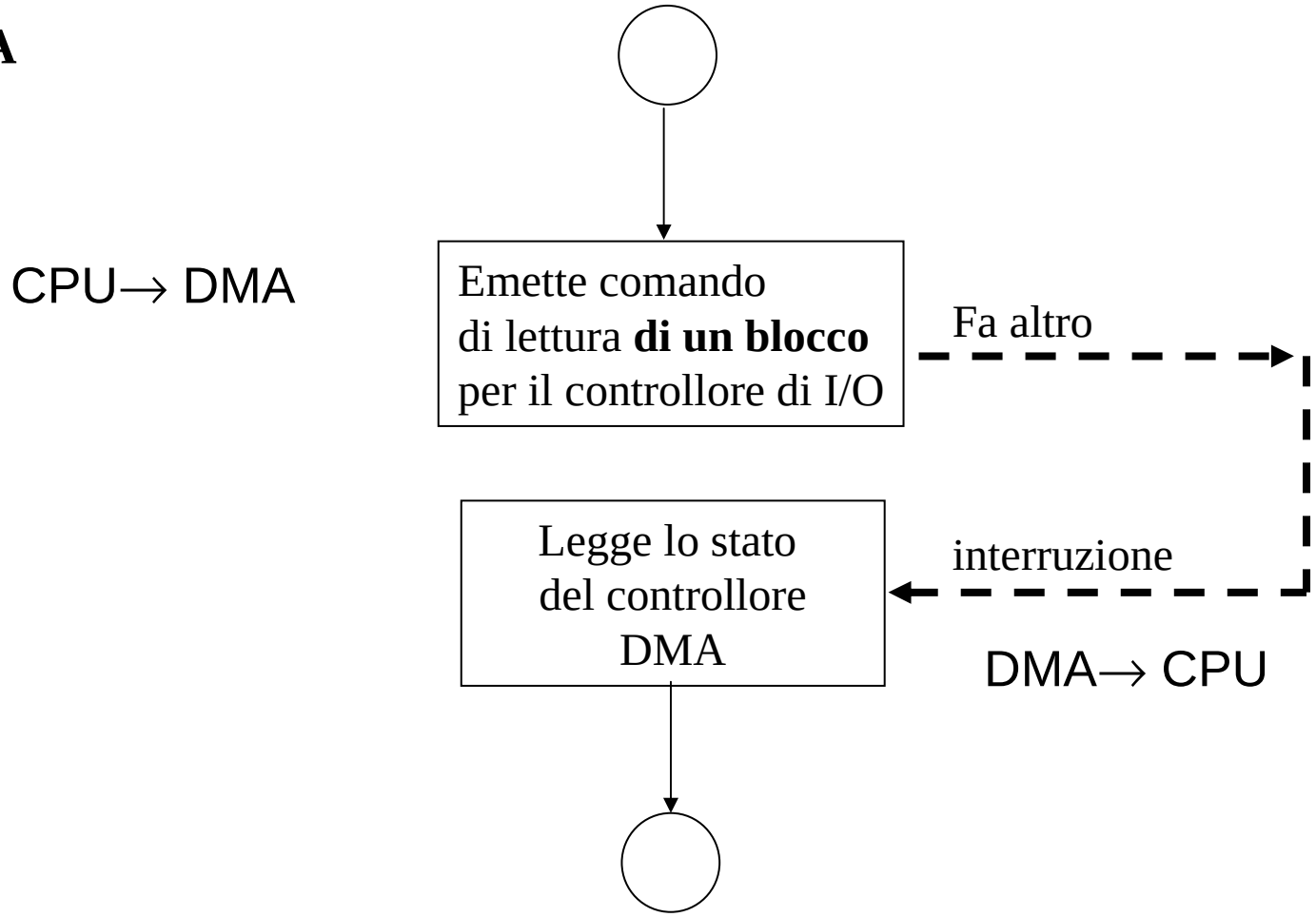
Fine:
DMA invia interrupt per informare fine operazione

Il processore non è impegnato nell’acquisizione del singolo dato

[P.es. salvataggio in memoria, incremento indice, salvataggio e ripristino stato,…]

ma solo all’inizio e alla fine del trasferimento di un blocco!

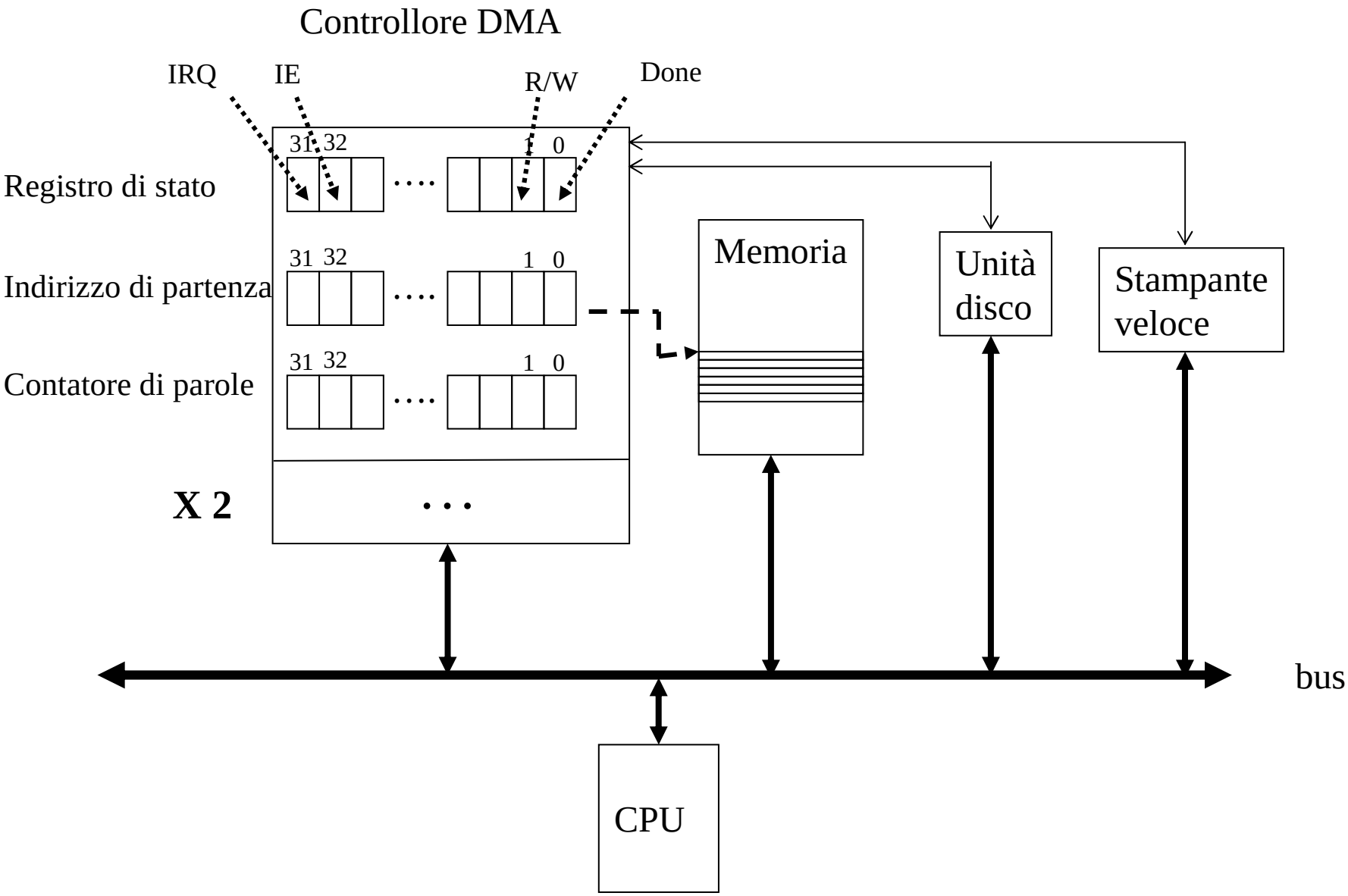
I/O con DMA



NB: processore dialoga con DMA come con un normale dispositivo I/O;
controllore DMA ha vari registri:

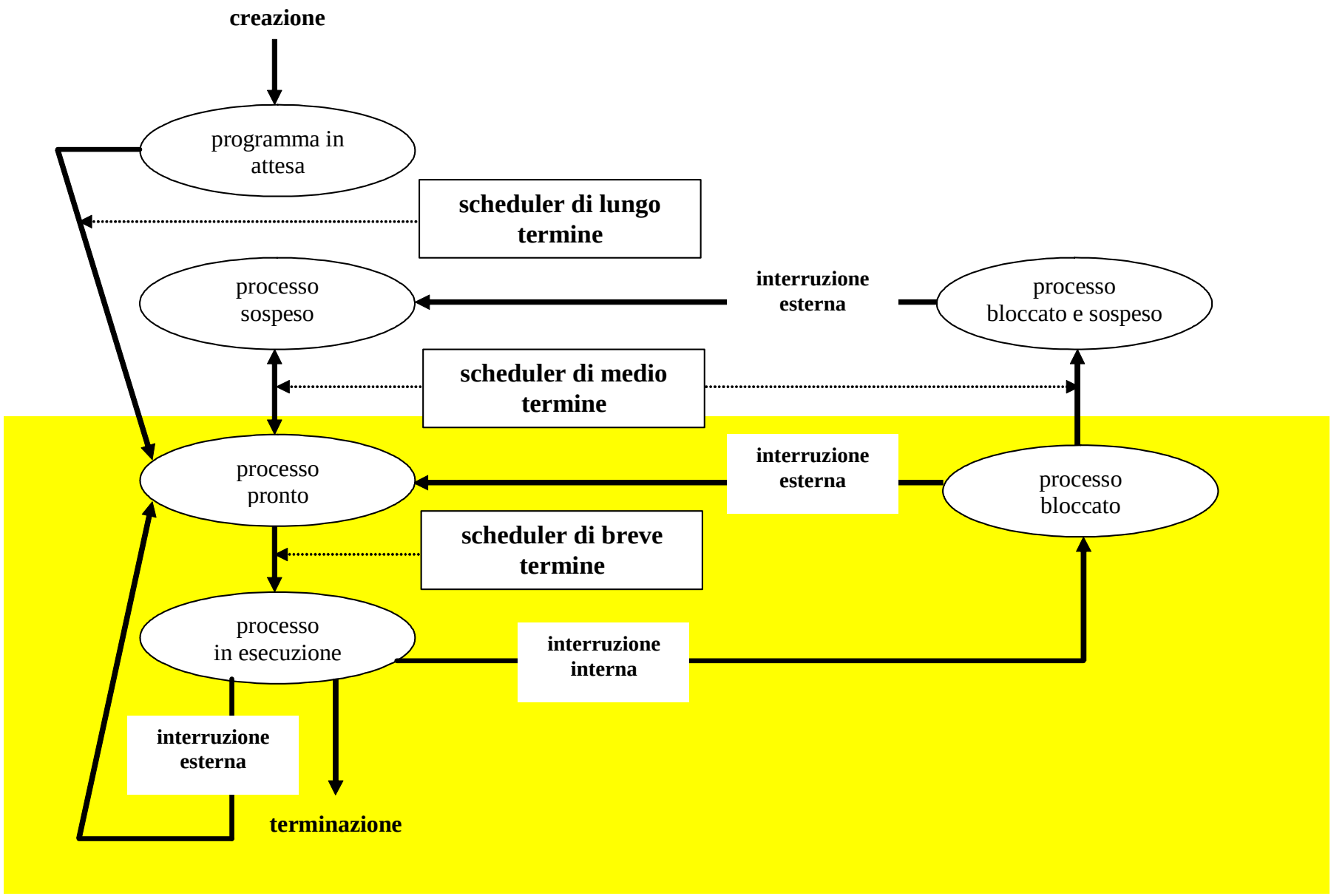
- stato e controllo [bit IRQ, IE, R/W, Done]
- indirizzo di partenza
- contatore di parole

ES: un controllore DMA a due canali (può controllare due dispositivi)



NB: si vede che può esserci conflitto sul bus [con processore o con vari DMA]

Richiami (da informatica A): gestione dei processi da parte del sistema operativo



- Per ogni processo, il sistema operativo gestisce un “descrittore”, struttura dati che ne memorizza lo stato (necessario per poter riprenderne l’esecuzione)
- Per lo stato di “pronto” e “bloccato”, code di processi (descrittori)
- Nella transizione di un processo da “in esecuzione” a “pronto” o “bloccato”, il sistema operativo effettua un “cambio di contesto”:
  - salva il contesto del processo corrente
    (copia il contenuto dei registri nel descrittore)
  - inserisce il descrittore nella coda dei processi pronti o bloccati, rispettivamente
  - seleziona il processo pronto da porre in esecuzione (scheduling di breve termine)
  - ripristina il contesto del processo da porre in esecuzione
    (copiando nei registri i valori prelevati dal descrittore)
  - restituisce il controllo al processo (tramite una istruzione di “return from interrupt)
- Meccanismo di protezione: il processore può funzionare in due stati distinti
  - stato utente
  - stato supervisore (disponibilità di istruzioni privilegiate)

  Programmi utente girano in modo utente – l’unico modo per passare nello stato supervisore è tramite eccezione (controllo ceduto al sistema operativo)
- Il descrittore di un processo contiene anche informazioni sulle risorse assegnate

NB: schema di principio - tutto ciò verrà affrontato nel corso di Sistemi Operativi

Definizione di memoria virtuale

Indica il *sistema* di circuiti e programmi che gestisce automaticamente la memoria RAM (cache+memoria principale) + memoria secondaria come un unico *sistema* di memoria.

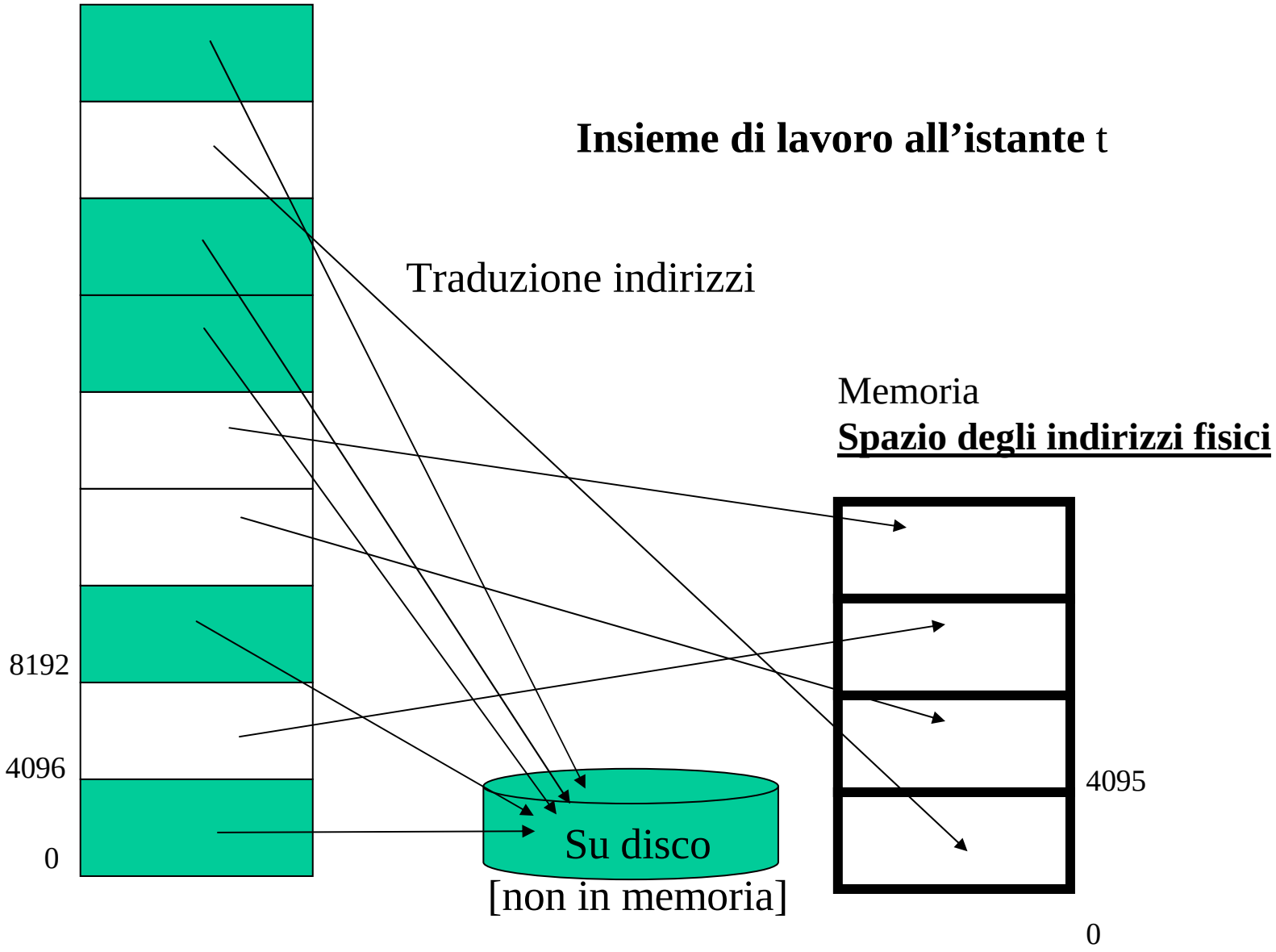
Motivazioni

- Tecnica di gestione della memoria usando la memoria principale come 'cache' del disco, sopperendo alle differenze di:
  - velocità [tempo di accesso RAM molto minore]: CPU accede a RAM
  - capacità [capacità RAM molto minore rispetto a disco e a capacità di indirizzamento della CPU]: necessità di "spostare le pagine"
- Permettere la condivisione della memoria da parte di più "processi", assicurando la non interferenza tra gli indirizzi "da essi generati"
  $\Rightarrow$ protezione tra diversi processi e tra processi e sistema operativo]

NB: entrambi gli obiettivi sono conseguiti in modo trasparente rispetto alla CPU
[P.es. per il secondo obiettivo il programmatore non deve gestire gli overlay]

IDEA DI BASE: SPAZIO VIRTUALE (associato ad ogni processo) e SPAZIO FISICO

Spazio degli indirizzi virtuali di un processo



• Programmi e dati composti da “**pagine**” [locazioni consecutive in memoria]:
pagina= unità minima trasferita fra memoria e disco

• Ogni processo ha uno “spazio virtuale” costituito da più pagine;
ciascuna pagina virtuale viene “mappata” in una pagina fisica di memoria
$\Rightarrow$ traduzione indirizzi fisici in indirizzi virtuali

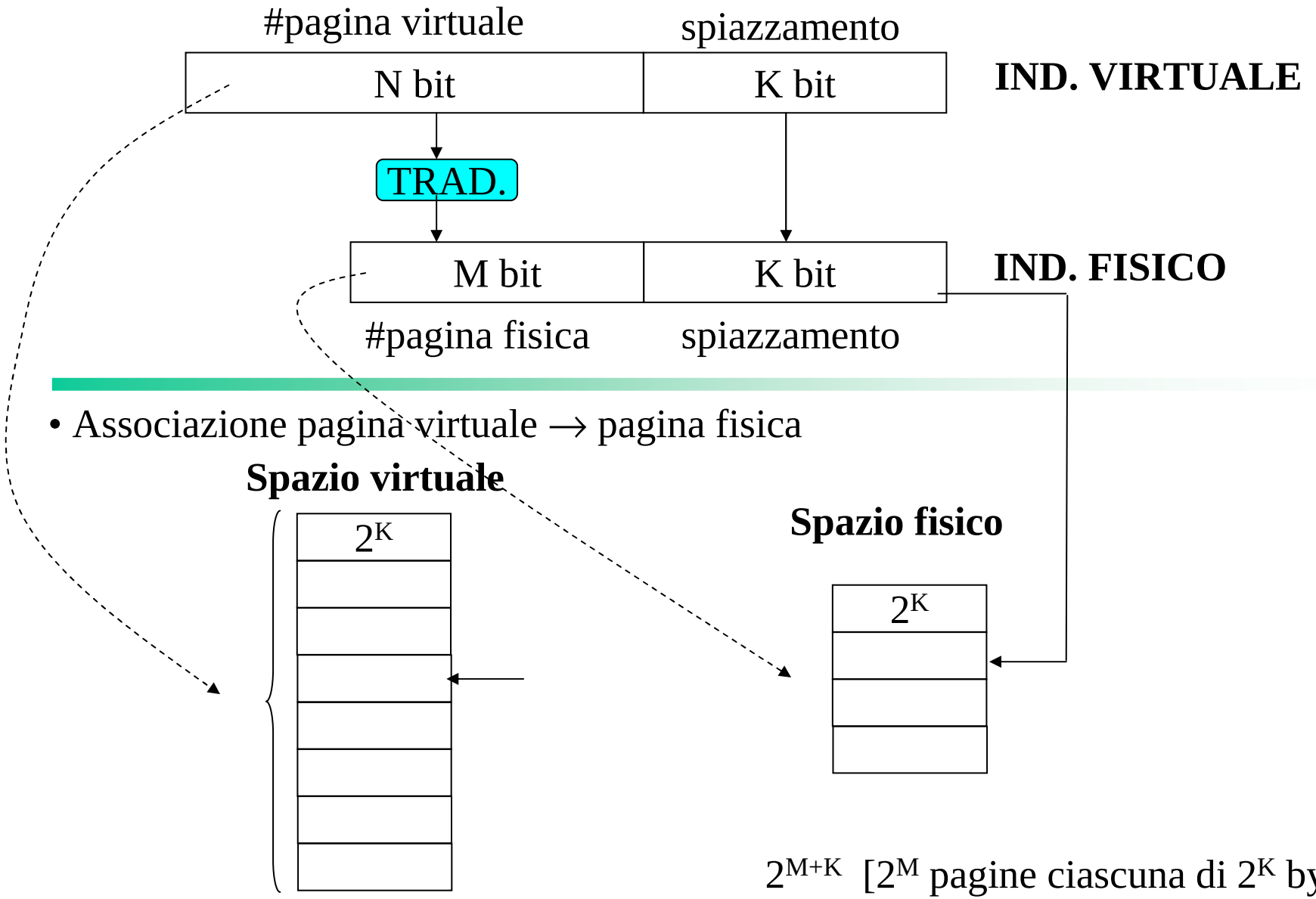
• Un indirizzo virtuale può essere mappato in memoria (se presente) o su disco (se il blocco non è caricato in memoria): in questo caso il sistema operativo provvede a caricare in memoria la pagina fisica

- Lo spazio virtuale può essere più grande dello spazio fisico disponibile

- Ogni processo può disporre di un proprio spazio virtuale (che comincia da 0) mappato nello spazio fisico; il sistema operativo ha il proprio spazio virtuale, detto *spazio di sistema*

PROTEZIONE: spazi virtuali di diversi processi mappati in pagine fisiche distinte

• **Traduzione da indirizzo virtuale a indirizzo fisico**:



NB

- La dimensione delle pagine virtuali è uguale a quella delle pagine fisiche ed è pari a 2^K dove K è il numero di bit destinati allo spiazzamento (offset)

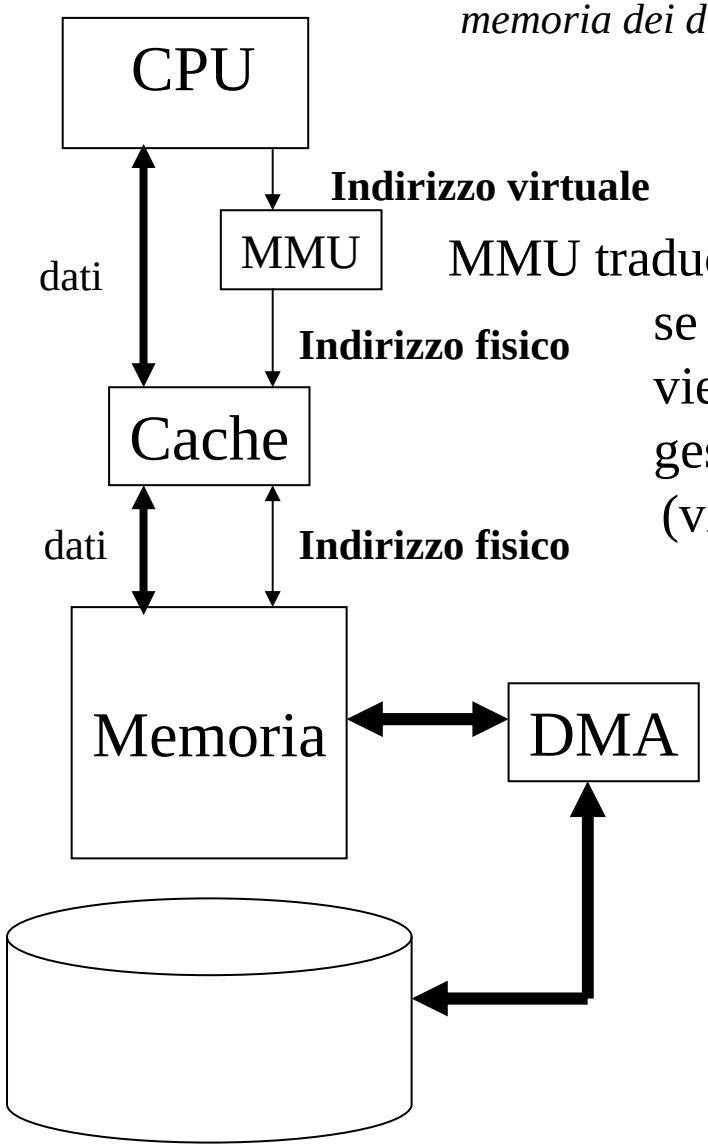
- Se N > M, lo spazio virtuale è maggiore dello spazio fisico: si crea l'illusione di uno spazio di memoria maggiore rispetto a quello che la memoria fisica rende disponibile

- Le pagine fisiche non devono essere necessariamente contigue: pagine contigue dello spazio virtuale possono essere mappate in pagine fisiche non contigue: si evita il problema della frammentazione esterna (spazio in memoria disponibile per un programma ma frammentato in porzioni ciascuna insufficiente)

ORA VEDIAMO COME REALIZZARE CONCRETAMENTE LA TRADUZIONE:
Vogliamo che ogni processo abbia uno spazio virtuale ad esso dedicato!

Realizzazione: schema generale

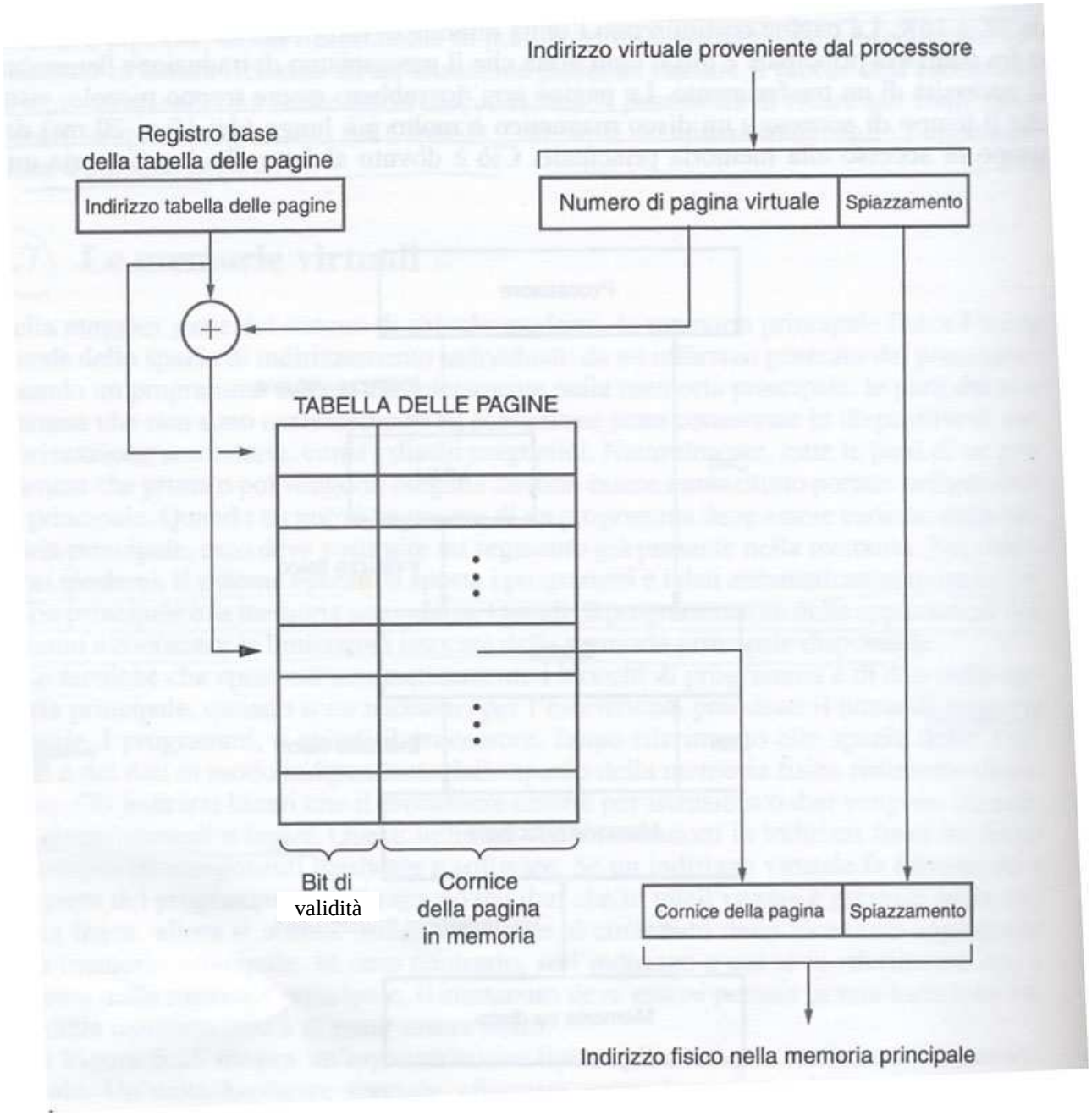
(*hardware e SO collaborano per gestire uno spazio degli indirizzi più grande di quello di memoria una velocità di trasferimento vicina a quella della cache e per garantire la non invasione di memoria dei differenti processi)*



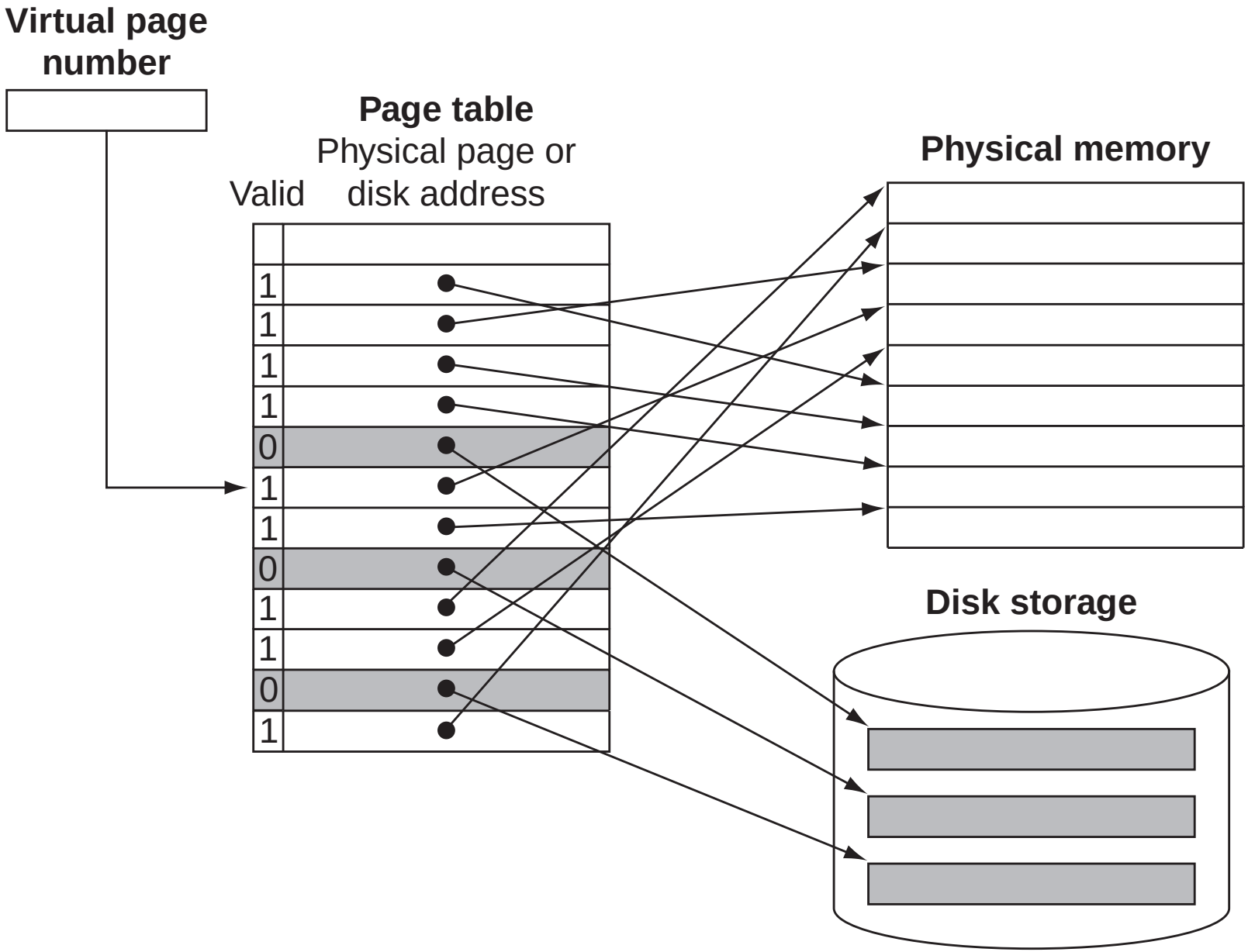
MMU traduce indirizzi virtuali in reali:
se i dati sono in memoria, ok; se sono su disco, viene invocato il Sistema Operativo per gestire il trasferimento dal disco in Memoria (via DMA)

DMA gestisce il trasferimento di pagine: riceve comando e alla fine segnala fine transazione.

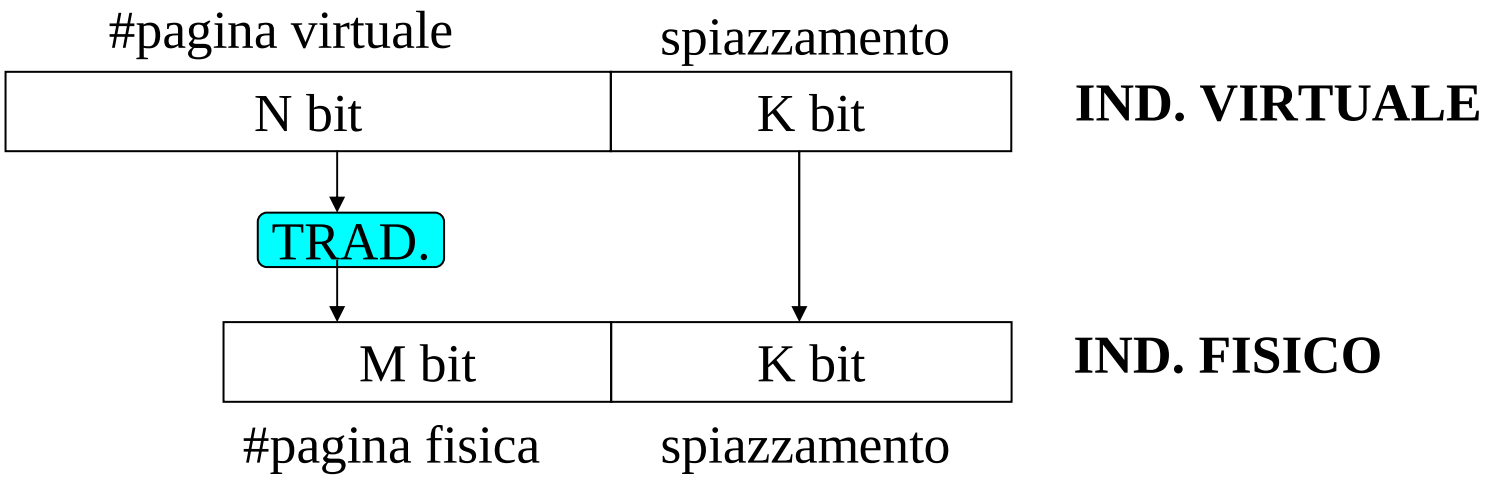
Realizzazione: la traduzione degli indirizzi via tabella delle pagine



Significato del bit di validità:



• La tabella delle pagine crea l'associazione pagina virtuale → pagina fisica



Spazio virtuale: 2^{N+K} [2^N pagine ciascuna di 2^K bytes]

Spazio fisico: 2^{M+K} [2^M pagine ciascuna di 2^K bytes]

• Ogni volta che c'è un accesso in memoria, consultazione tabella delle pagine:

- l'indirizzo virtuale è presente (bit di validità a 1): traduzione nell'indirizzo fisico
- l'indirizzo virtuale è assente (bit validità a 0): la pagina no in memoria, ma su disco

⇒ MMU solleva eccezione invocando il sistema operativo, che:

- trasferisce una pagina in memoria, eventualmente eliminandone una
- aggiorna di conseguenza la tabella delle pagine
- restituisce il controllo al processo interrotto

Più in dettaglio (gestione multiprocessi)

- Ogni processo ha un proprio spazio virtuale, corrispondente alla propria tabella delle pagine.
- Il contesto di ogni processo include:
  - valori correnti dei registri utilizzati (incluso PC)
  - indirizzo tabella delle pagine da esso utilizzata
- Alla creazione di un processo, il sistema operativo:
  - riserva spazio su disco per tutte le pagine usate dal processo (swap space)
  - crea tabella delle pagine
  - crea struttura dati per associazione pagina virtuale → pagina su disco

    [può essere contenuta nello stesso campo della tabella delle pagine, ma è necessario mantenere informazione anche per le pagine caricate in memoria (dobbiamo sapere dove caricarle su disco quando vengono rimpiazzate!)]

• Quando un processo è in esecuzione, il registro base tabella delle pagine indica la tabella delle pagine ad esso associata

[Associazione spazio virtuale processo – memoria fisica]

• Il sistema operativo mantiene inoltre una struttura che per ogni pagina fisica in memoria denota il processo e la pagina virtuale associata

[Associazione memoria fisica – pagine virtuali di diversi processi]

• ACCESSO AD UN INDIRIZZO VIRTUALE:

hit: la tabella delle pagine restituisce direttamente l'indirizzo fisico corrispondente

miss: eccezione per miss di pagina: controllo passato al sistema operativo

- viene avviato trasferimento del blocco mancante via DMA che richiederà molti cicli del processore: il processo corrente passa nello stato di bloccato [cambio di contesto]
- se tutte le pagine fisiche della memoria sono utilizzate, il sistema operativo ne seleziona una e la rimpiazza con quella richiesta (utilizzando tipicamente la politica LRU)

NB:

- Lo spazio virtuale del sistema operativo – non quello dei processi- comprende tabelle delle pagine dei processi: solo il S.O può modificarle.
- Ogni volta che il S.O. rilascia il controllo ad un processo, gli assegna la tabella delle pagine cambiando opportunamente il registro base della tabella delle pagine. L'indirizzo base [che identifica la tabella] fa parte dello "stato" del processo salvato dal S.O. e ripristinato quando si cede il controllo al processo
- Ogni processo utente ha a disposizione uno spazio virtuale illimitato [da 0 in poi] e non deve preoccuparsi della collocazione fisica degli altri processi
- Protezione:
  - il Sistema Operativo assegna pagine fisiche distinte ai processi
  - pagine possono essere condivise tra processi ma con bit di protezione (permessi di lettura o scrittura)
  - i processi non "vedono" le pagine fisiche destinate al sistema operativo
  - in "stato di utente" i programmi non possono eseguire istruzioni privilegiate; in particolare non possono modificare il registro base della tabella pagine
  - passaggio a "stato supervisore" solo via eccezione per invocare S.Operativo

NOTA

RAM è cache rispetto a memoria di massa. Notare però che:

- l'indirizzamento diretto non è praticabile: page fault si pagano troppo cari; quindi la RAM è gestita come cache completamente associativa
- gestione page fault via SW: overhead SW trascurabile rispetto a miss + flessibilità
- è necessaria la tabella per mantenere i riferimenti: sarebbe troppo oneroso confrontare i tag in memoria centrale…
- per minimizzare il numero di page-fault, la sostituzione avviene con politica LRU, di solito approssimata [si hanno molti blocchi in memoria]:
  es. un bit per ogni blocco (detto *use bit* o *reference bit*) viene posto a 1 ad ogni accesso, periodicamente il S.O. li pone tutti a 0: pagina da eliminare scelta tra quelle che hanno il reference bit a 0
- write-through non è praticabile: tempo di accesso al disco è molto alto
  $\Rightarrow$ un bit di modifica per ogni blocco e strategia "write back"

In particolare, ogni pagina ha nella tabella i seguenti bit (oltre a reference bit):

- bit di validità [se è 0 la pagina non è in memoria; S.O. può invalidare una pagina senza eliminarla fisicamente dalla memoria]
- bit di modifica
- gruppo di bit che indicano il permesso [in lettura/scrittura]

UN PROBLEMA: la dimensione della tabella delle pagine può essere considerevole

Es:

- 32 bit di indirizzo virtuale (generato dal processore)
- dimensione di una pagina 4 KB (ovvero 2^{12} bytes)
- 4 bytes per ogni item della tabella delle pagine

→ Risulta ovviamente un campo per il numero di pagina virtuale di 20 bit (32 – 12). Quindi 2^{20} pagine.

→ Dimensione della tabella delle pagine: $2^{20} * 4 = 4MB$

Si consideri che ogni processo ha associata una tabella delle pagine…

Alcune strategie per ridurre la dimensione della tabella delle pagine:

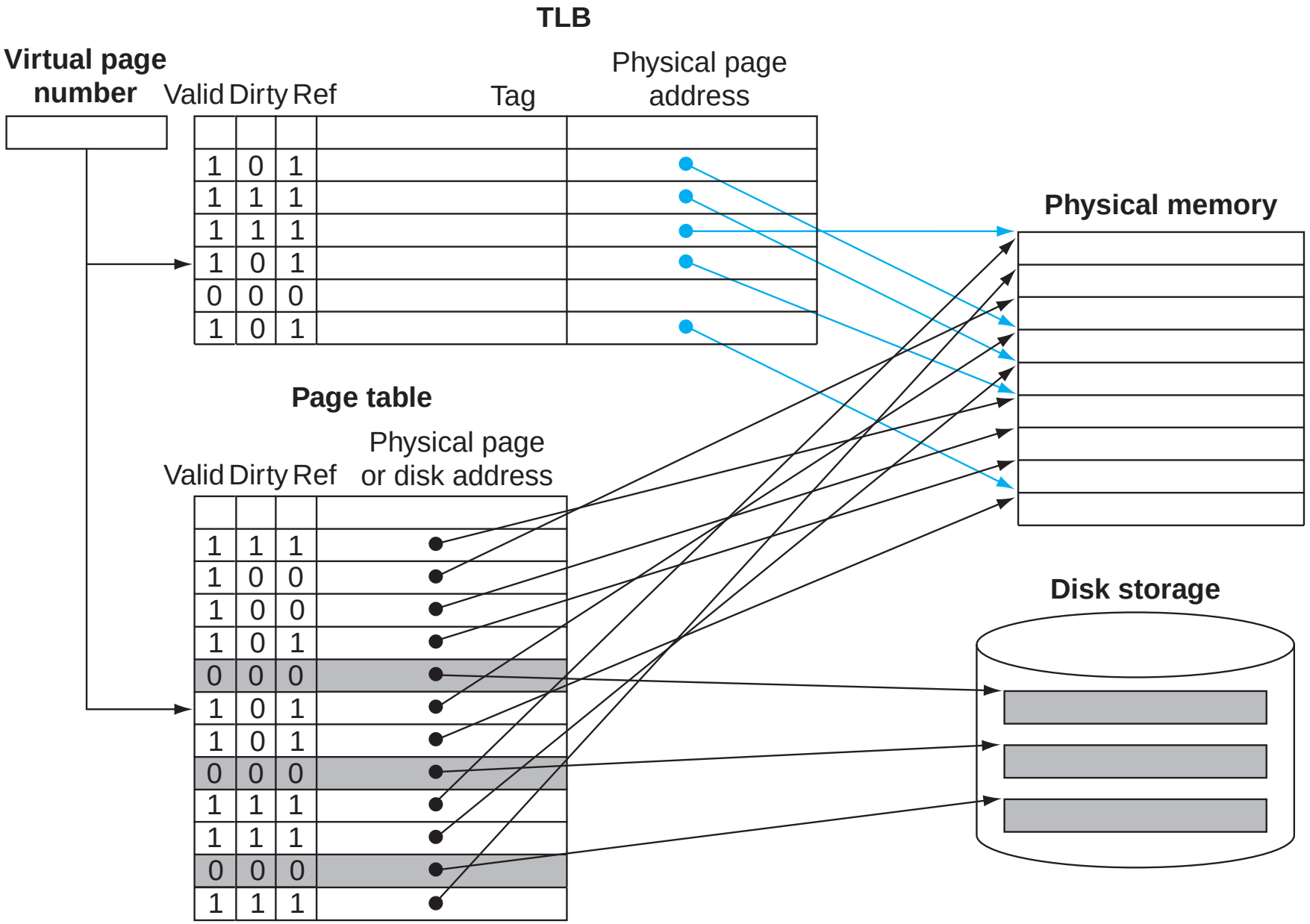
- permettere che la tabella delle pagine possa espandersi dinamicamente (utilizzo di un registro limite)
- gestione della tabella con funzione di hash
- utilizzo di livelli multipli di tabelle
  (livello 1: blocchi di più pagine indirizzati dai bit più significativi: ciascun blocco indica – se usato – la relativa tabella “di livello 2”)
- paginazione delle tabelle (poste nello spazio virtuale del sistema operativo)

ALTRO PROBLEMA

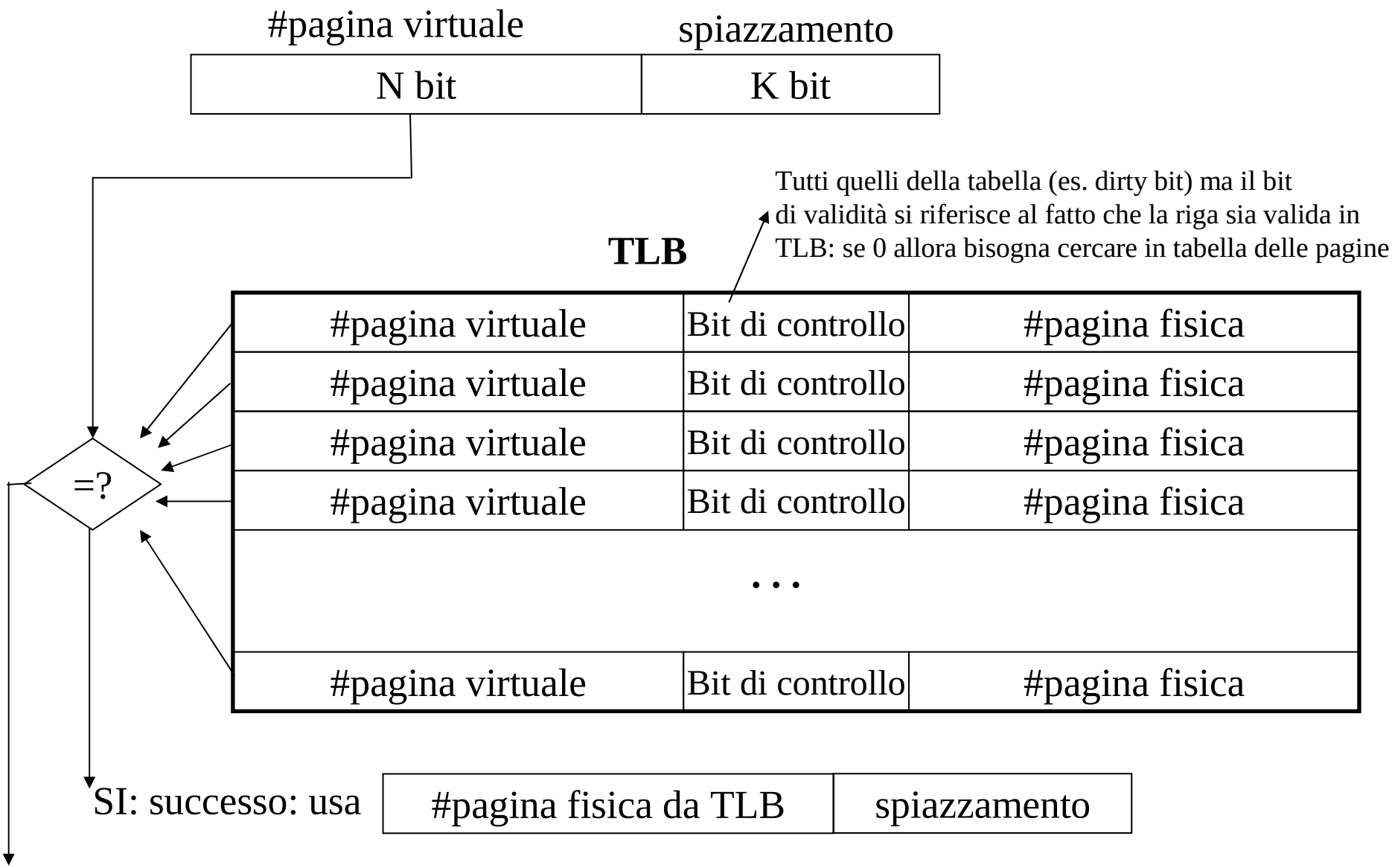
La tabella delle pagine deve essere acceduta ogni volta che si usa un indirizzo virtuale [fetch istruzioni + accesso a memoria dati!]

⇒ Per gli stessi motivi che impongono l'uso di cache, essa deve stare in MMU all'interno del processore…
… ma le dimensioni notevoli non lo permettono… essa è in RAM…

⇒ Uso una cache della tabella delle pagine detta **Translation Lookaside Buffer [TLB]** =
Buffer per la traduzione degli indirizzi



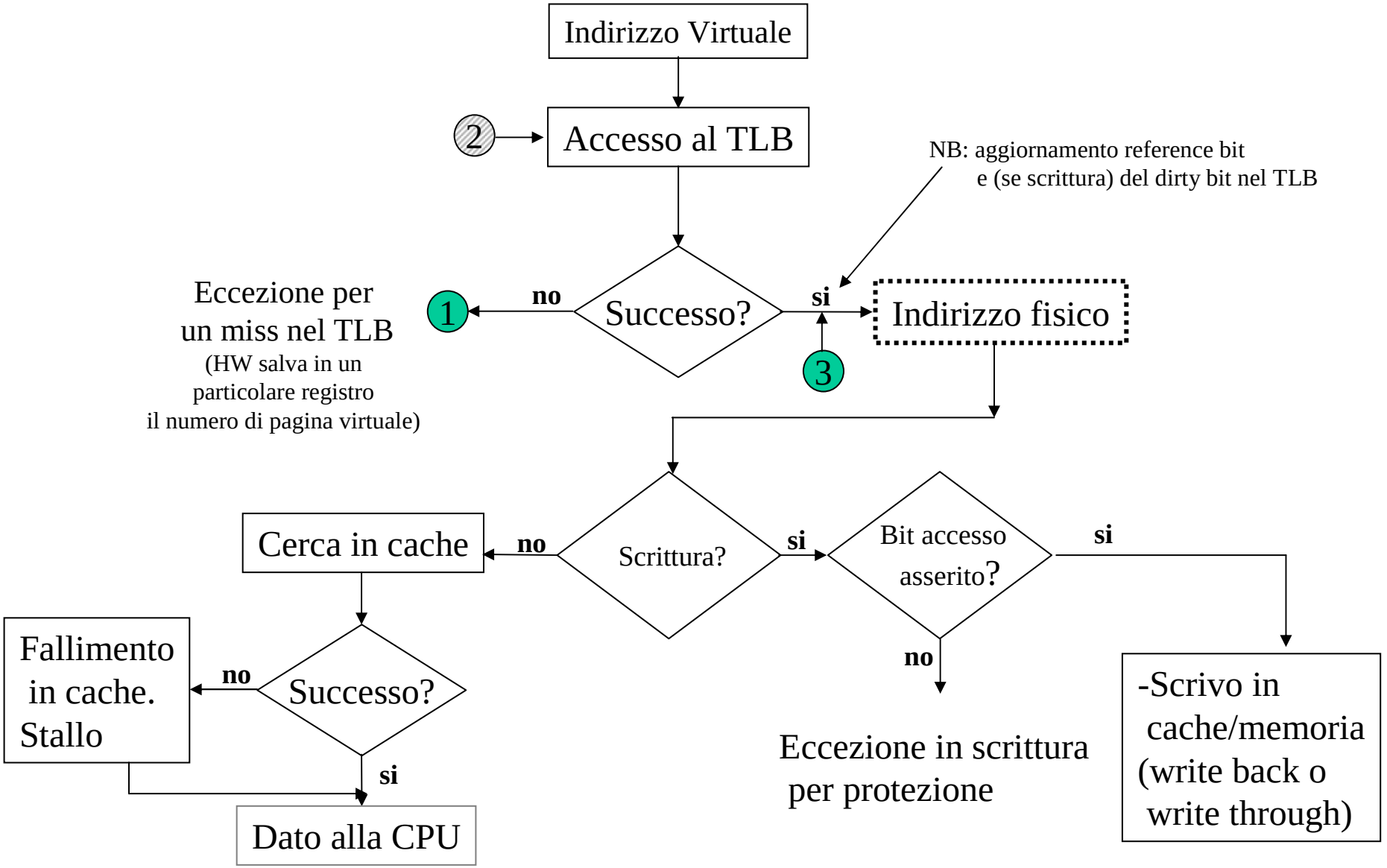
NB: [Valido in TLB] =1 indica valido e “in memoria fisica”, 0: miss TLB



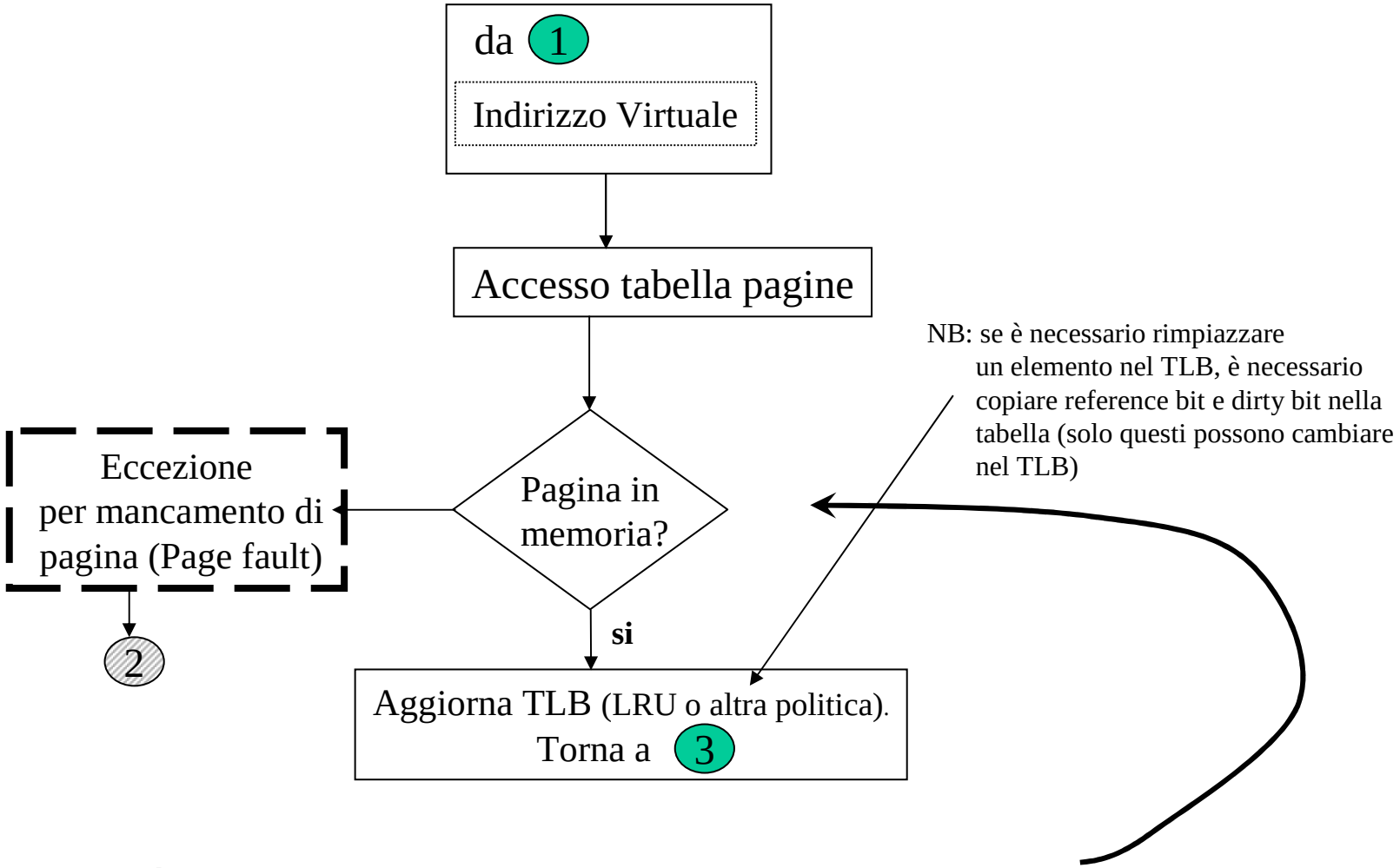
NO: usa tabella delle pagine per accedere a indirizzo fisico

NB: necessariamente, TLB mantiene solo riferimenti che si trovano in memoria fisica, non su disco! Quindi il bit di validità "della tabella" non è necessario (è 1)

Accesso ad una locazione di memoria (ispirato FastMATH TLB)



Gestione miss TLB (ispirato MIPSR2000)

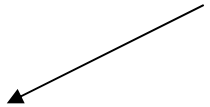


NB: un'alternativa è quella di copiare elemento della tabella delle pagine in TLB, senza controllare page fault; se non è valido [pagina è su disco] page fault avverrà dopo, accedendo a TLB. Privilegia miss TLB - hit di pagina

NB:

• miss TLB può essere gestito via SW tramite una routine di gestione dell'eccezione
(ovvero, vi sono istruzioni macchina – privilegiate, usate da sistema operativo – per caricare item da tabella delle pagine a TLB) o direttamente da HW

• page fault invece è gestito sempre da SW
[routine del Sistema Operativo chiamata tramite una eccezione]



- Sullo stadio di fetch
(miss di pagina durante il caricamento di una istruzione)
- Sullo stadio "Mem"
(miss di pagina durante l'accesso al dato – es. lw e sw)

NB2: mentre la memoria come visto è sempre gestita come cache completamente associativa, per TLB si usano vari gradi di associatività
[nei lucidi precedenti si era supposto TLB completamente associativo]
- in particolare, TLB più grandi rinunciano alla completa associatività

NB 3: ogni volta che il sistema operativo rimpiazza una pagina fisica in memoria RAM (corrispondente ad una pagina virtuale di un determinato processo) con un'altra pagina prelevata dal disco, deve provvedere a:

- modificare opportunamente le tabelle delle pagine (come già visto)
- aggiornamento TLB:

  vengono invalidate le entry che si riferiscono alle pagine virtuali (le cui pagine fisiche sono) eliminate da RAM e spostate su disco

  [in questo modo siamo sicuri che un accesso successivo alla pagina virtuale spostata generi un page fault]
- flush delle cache (bit di validità a 0 per i blocchi che si riferiscono alle parti della RAM che sono state modificate!)

  [per assicurare la coerenza cache – memoria RAM]

Un problema

- Ogni volta che c'è il cambio di contesto, S.O. cambia la tabella delle pagine (via registro base) ritornando alla tabella del processo cui si cede controllo;
- TLB deve essere "ripulito" [invalidando bit di validità] per evitare che il processo usi ancora le corrispondenze "vecchie" [interferendo con lo spazio fisico del processo precedente] – inizialmente, ciò comporta miss nel TLB per le corrispondenze utilizzate, che vengono caricate nel TLB
- Se il nuovo processo esegue poche istruzioni e magari il controllo ritorna al precedente, TLB è di nuovo "ripulito" ed il processo precedente deve di nuovo ripassare attraverso la tabella delle pagine (miss TLB)
- Soluzione più efficiente:
  - tag del TLB contiene un campo addizionale: *identificatore di processo*
  - un registro (aggiornato da sistema operativo) indica il processo corrente
  - TLB non ripulito, ma hit si ha solo se coincidono processo corrente e identificatore di processo

Scelta della Dimensione delle Pagine in memoria centrale

A favore di pagina grande:

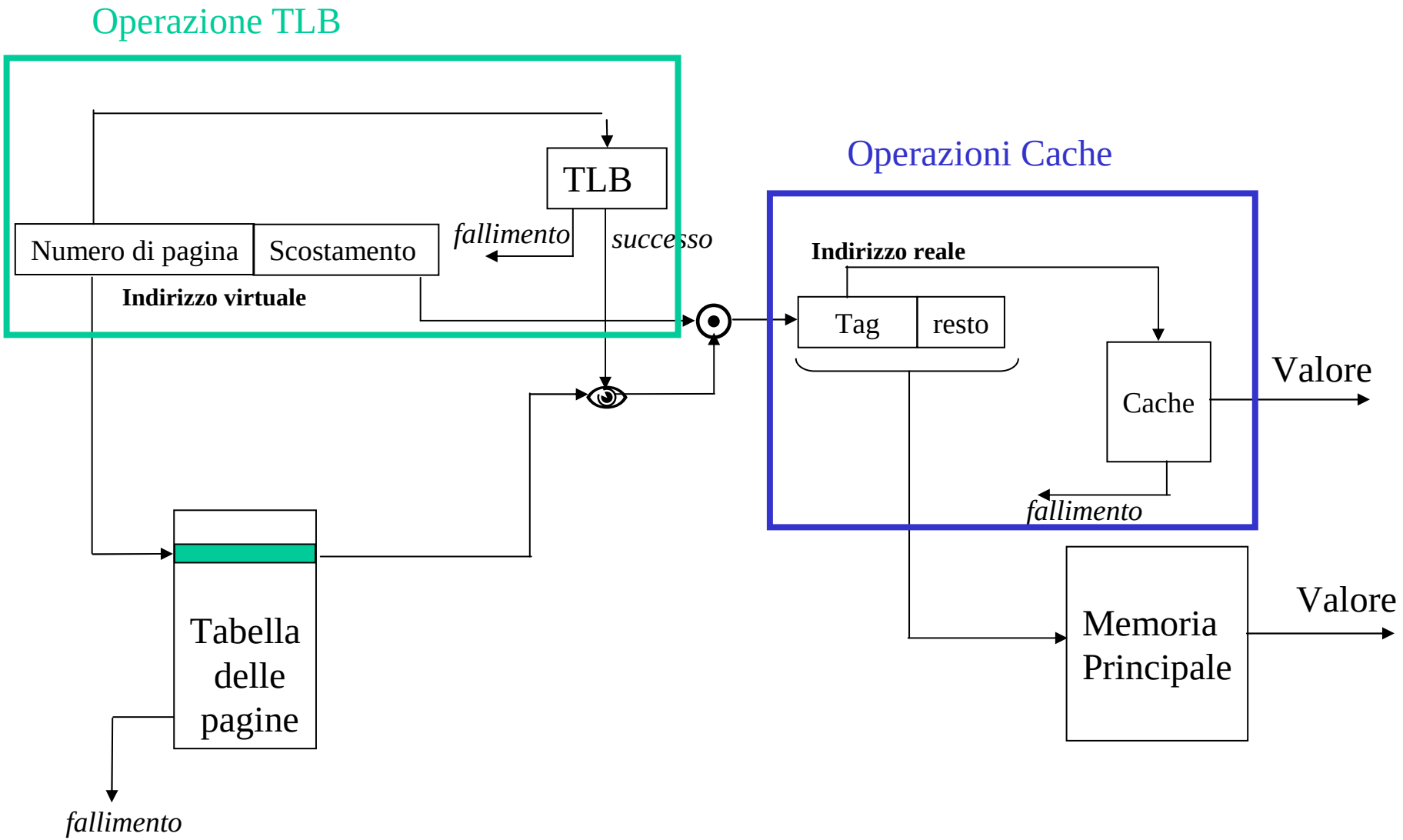
- minor numero pagine, quindi tabella delle pagine piccola
- trasferimento più economico vs. miss per tempo di accesso del disco: privilegiare la riduzione dei page fault
- TLB più piccolo (numeri di pagina virtuale e fisica più piccoli)
- riduzione miss TLB (a parità di dimensioni, più elementi!)

A favore di pagina piccola:

- meno memoria sprecata [evita parti di pagina mai utilizzate]
- per processi piccoli: accelera start-up

Nella pratica, le pagine devono essere abbastanza grandi [32-64 KB] per l'elevato costo di miss dovuto all'alto tempo di accesso al disco: meglio ridurre i page fault e sopportare un tempo di trasferimento più lungo [comunque molto minore del tempo di accesso]

Visione di insieme: Buffer di traduzione e operazioni in cache



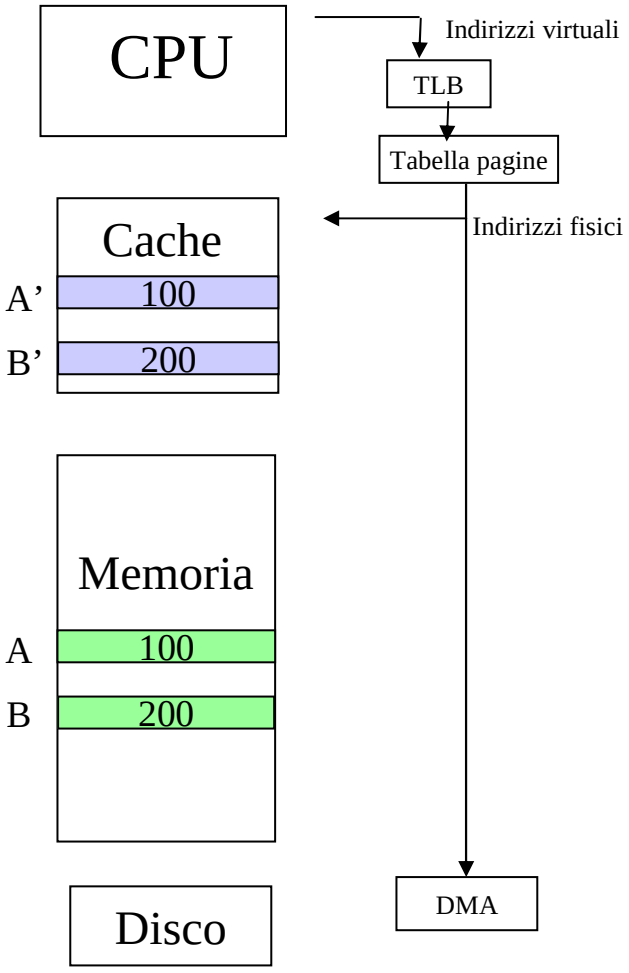
Eccezione: chiamata al sistema operativo

Possiamo avere ad ogni accesso:

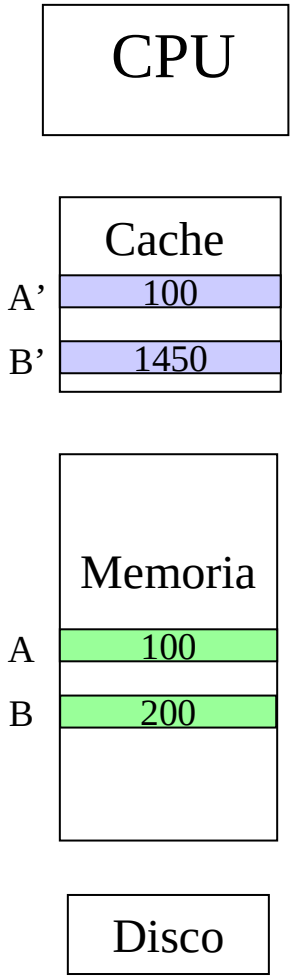
- TLB: hit o miss
- tabella delle pagine: hit o miss
- cache: hit o miss

TLB	Tabella pagine	Cache	Possibile? In quale caso?
Hit	Hit	Hit	Possibile, ed è il caso più fortunato!
Hit	Hit	Miss	Possibile (NB: hit TLB implica presenza in RAM) – indirizzo non in cache
Miss	Hit	Hit	Possibile: pagina non in TLB, ma presente in RAM (e indirizzo in cache)
Miss	Hit	Miss	Possibile: miss TLB, ma pagina presente in RAM – indirizzo non in cache
Miss	Miss	Miss	Possibile: pagina non in RAM (necessariamente indirizzo non in cache!)
Hit	Miss	Miss	IMPOSSIBILE: se presente in TLB, la pagina è in RAM
Hit	Miss	Hit	IMPOSSIBILE: se presente in TLB, la pagina è in RAM
Miss	Miss	Hit	IMPOSSIBILE: se la pagina non è in RAM, non può essere nella cache

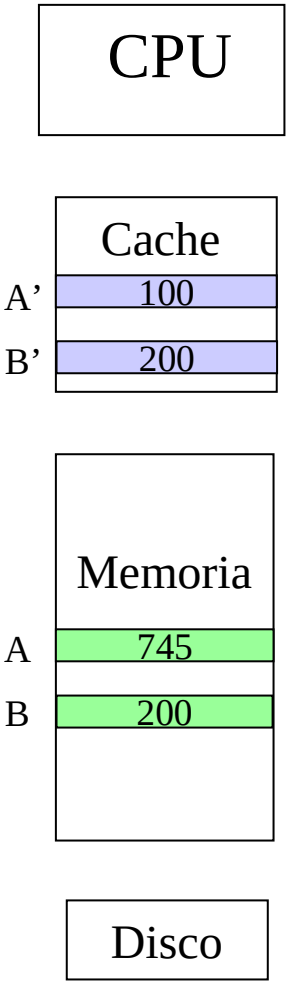
Il problema della coerenza dei dati



Cache e Memoria
Coerenti
[A]=[A’]& [B]=[B’]



Cache e Memoria
Incoerenti per operazione CPU
[B]≠[B’], [B] vecchio



Cache e Memoria
Incoerenti per operazione
DMA (caricamento pagina)
[A] ≠[A’]. [A’] vecchio

Coerenza cache - memoria principale

1) Operazione CPU (lettura o scrittura)

Successo (hit) la parola cercata è in cache

in lettura: OK

in scrittura: problema della coerenza con i dati in memoria; due alternative:

tecniche di **write through**

[aggiornamento contemporaneo memoria]

tecniche di **write back** (dirty bit)

[aggiornamento memoria in seguito, quando il blocco deve essere rimosso dalla cache]

Insuccesso (miss):

in lettura: spostamento di un blocco in memoria cache

(con tecniche di load-through si può inviare subito la parola alla cache e poi spostare il blocco)

in scrittura: due alternative

write through:

scrittura in memoria RAM senza trasferire blocco in cache

write back:

trasferimento blocco in cache e scrittura nuove informazioni

NB: serve anche un **bit di validità** per ogni blocco della cache, che indica se il blocco è valido: HIT solo se bit di validità a 1, altrimenti il blocco in cache non è valido

- all'inizio tutti i bit di validità a 0 [cache non caricata]
- bit di validità di un blocco posto a 1 quando un blocco da memoria principale è trasferito nella cache [blocco caricato e aggiornato]

2) Operazione DMA

(trasferimento pagina fisica RAM-Disco, ma anche operazione di I/O con un dispositivo generico al posto del disco)

- ogni volta che DMA trasferisce dati da dispositivo a memoria RAM [blocco]:
  se il blocco è presente in cache il sistema operativo pone il bit di validità cache a 0 [dati incoerenti con RAM]
- ogni volta che DMA trasferisce dati da memoria RAM [blocco] a dispositivo:
  se la cache utilizza write-back [write in blocchi cache senza aggiornare memoria] la memoria può non riflettere la cache: una possibilità è lo svuotamento (flush) della cache: i blocchi con dirty bit a 1 sono trasferiti in memoria RAM

Coerenza RAM - disco

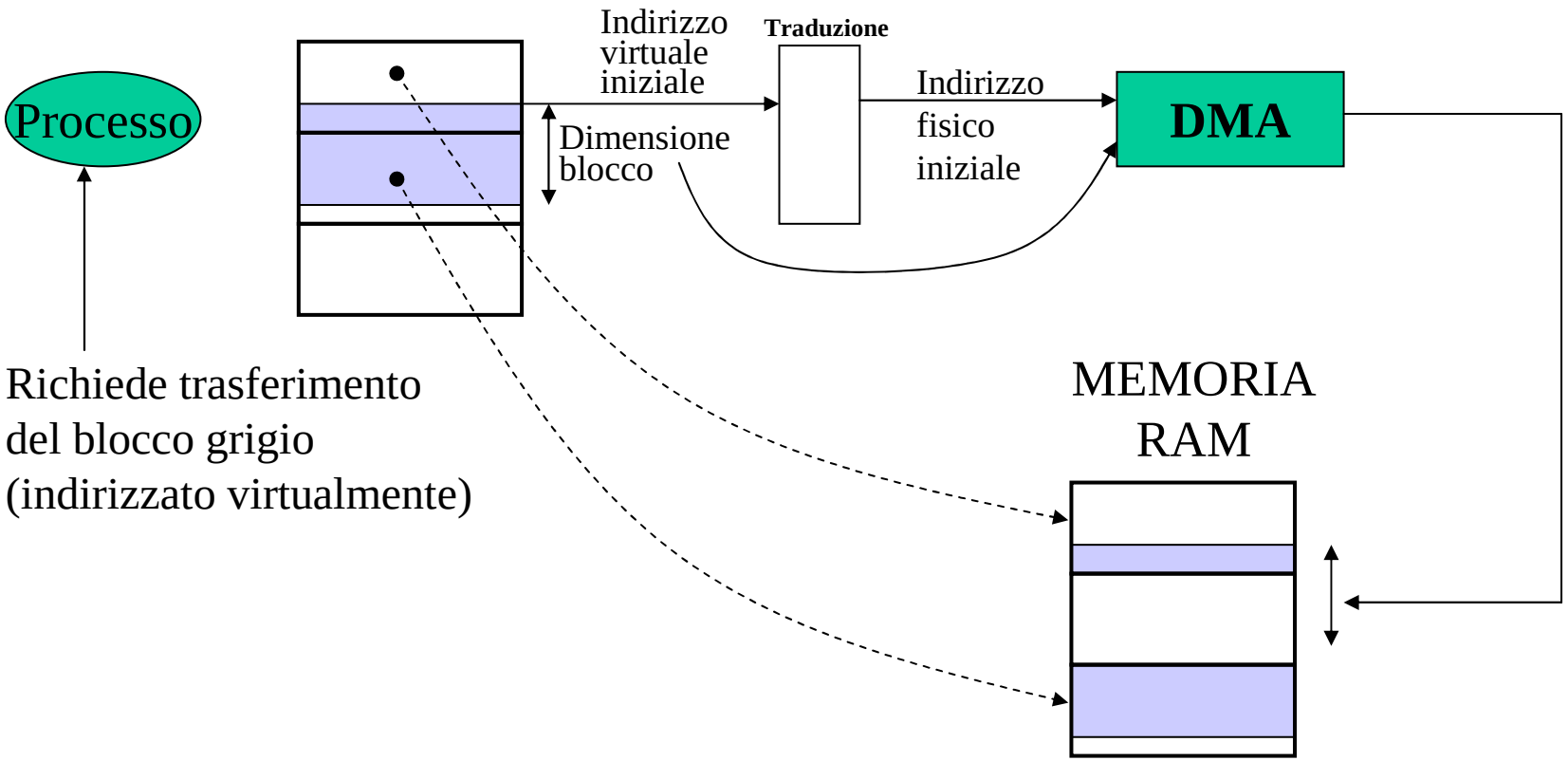
Il sistema operativo assicura la coerenza tramite il meccanismo visto di memoria virtuale – la RAM è gestita come cache del disco utilizzando write-back
(ogni volta che un blocco viene rimpiazzato, se modificato viene copiato nel disco)

Coerenza TLB – tabella delle pagine

Si è visto che il sistema operativo, ogni volta che rimpiazza una pagina fisica in memoria, aggiorna di conseguenza la tabella delle pagine e invalida tutti gli eventuali campi del TLB che riferivano la pagina eliminata

Un altro problema di coerenza

Operazione di I/O richiesta da un processo realizzata via DMA



Un blocco contiguo nello spazio virtuale può non essere contiguo nello spazio fisico: senza accorgimenti, il DMA non trasferisce il blocco nel posto giusto

Soluzioni

1) DMA lavora con indirizzi virtuali:

   il DMA ha una tabella di traduzione da indirizzo virtuale a fisico, che utilizza durante il trasferimento; il sistema operativo provvede a ricopiare parte della tabella delle pagine nel DMA, in modo che tutti gli indirizzi virtuali siano tradotti correttamente

2) DMA lavora con indirizzi fisici:

   il sistema operativo spezza il blocco da trasferire in blocchi più piccoli, ciascuno dei quali è contenuto in una sola pagina fisica (contigui)

   - può richiedere al DMA i trasferimenti “uno a uno”
   - se il DMA lo consente, può comandare al DMA di eseguire l’intera sequenza di trasferimenti