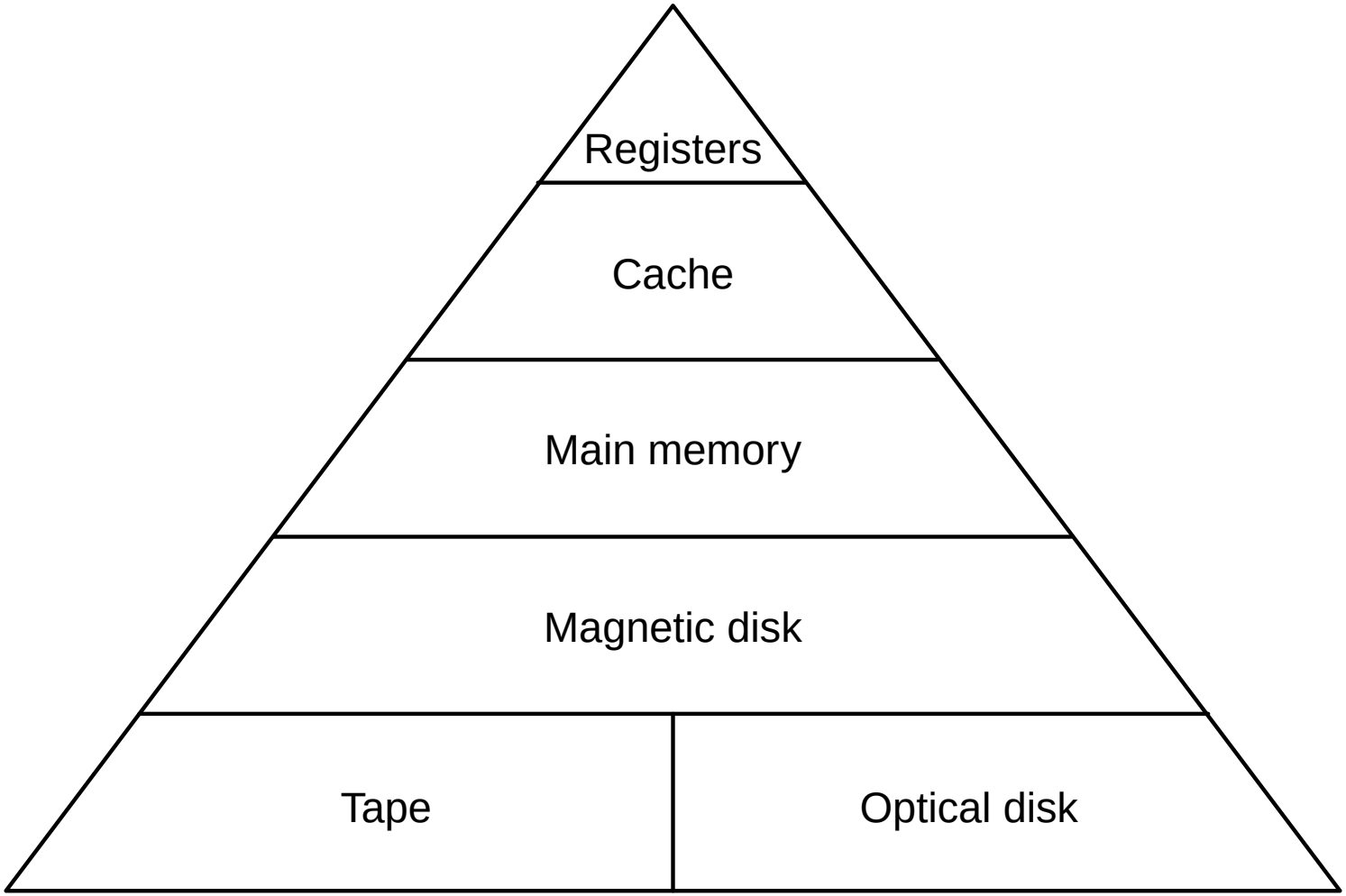
Organizzazione della memoria fisica

Piramide della memoria fisica nel computer



Organizzazione accesso a memoria

La ricerca di un compromesso tra velocità e capacità ha portato a livelli di memoria intermedi tra registri, memoria principale e memoria di massa.

Organizzazione accesso a memoria

La ricerca di un compromesso tra velocità e capacità ha portato a livelli di memoria intermedi tra registri, memoria principale e memoria di massa.

Memoria cache:

- intermedia a registri e memoria principale
- realizzata con hardware veloce e costoso.

Organizzazione accesso a memoria

La ricerca di un compromesso tra velocità e capacità ha portato a livelli di memoria intermedi tra registri, memoria principale e memoria di massa.

Memoria cache:

- intermedia a registri e memoria principale
- realizzata con hardware veloce e costoso.

Memoria virtuale paginata:

- intermedia a memoria principale e di massa
- espone una memoria principale più estesa appoggiandosi fisicamente a quella di massa.

Organizzazione accesso a memoria

La ricerca di un compromesso tra velocità e capacità ha portato a livelli di memoria intermedi tra registri, memoria principale e memoria di massa.

Memoria cache:

- intermedia a registri e memoria principale
- realizzata con hardware veloce e costoso.

Memoria virtuale paginata:

- intermedia a memoria principale e di massa
- espone una memoria principale più estesa appoggiandosi fisicamente a quella di massa.

Memoria virtuale segmentata:

- intermedia a memoria principale e di massa
- riserva segmenti di memoria virtuale al software.

Memoria cache

La memoria cache mantiene le istruzioni e i dati che saranno più probabilmente usati.

La politica di occupazione della cache sfrutta due regole statistiche

- località temporale: istruzioni e dati usati di recente hanno maggior probabilità di essere richiamati
- località spaziale: istruzioni e dati contigui a quelli usati recentemente hanno maggior probabilità di essere richiamati.

L'occupazione della cache è interamente gestita a livello hardware.

Accesso alla memoria principale

La CPU normalmente accede alla memoria principale solo attraverso la cache.

L'accesso con successo all'istruzione o al dato presente nella cache prende il nome di cache hit.

Se l'accesso non ha successo (cache miss),

- una parte dalla memoria cache è sovrascritta con la parte di memoria principale contenente il dato
- si accede al dato nella cache.

Poichè un cache miss costa alla CPU più di un accesso alla memoria principale, la cache è giustificata solo se i cache miss sono poco frequenti.

Convenienza della memoria cache

Dati i vincoli
h: probabilità di cache hit
t_c: tempo di accesso alla cache
t_p: tempo di accesso alla memoria principale,

Convenienza della memoria cache

Dati i vincoli
h: probabilità di cache hit
t_c: tempo di accesso alla cache
t_p: tempo di accesso alla memoria principale,

allora il tempo medio t_M di accesso alla memoria in presenza della cache è vantaggioso se

$$t_M = t_c + (1 - h) \cdot t_p \quad < \quad t_p$$

Convenienza della memoria cache

Dati i vincoli
h: probabilità di cache hit
t_c: tempo di accesso alla cache
t_p: tempo di accesso alla memoria principale,

allora il tempo medio t_M di accesso alla memoria in presenza della cache è vantaggioso se

$$t_M = t_c + (1 - h) \cdot t_p \quad < \quad t_p$$

La cache dunque è conveniente se

$$h > t_c / t_p \quad .$$

In più, la cache ha un costo iniziale rilevante.

Linee di cache

La contiguità spaziale viene implememententata direttamente a livello fisico in forma di linea di cache.

Linee di cache

La contiguità spaziale viene implememententata direttamente a livello fisico in forma di linea di cache.

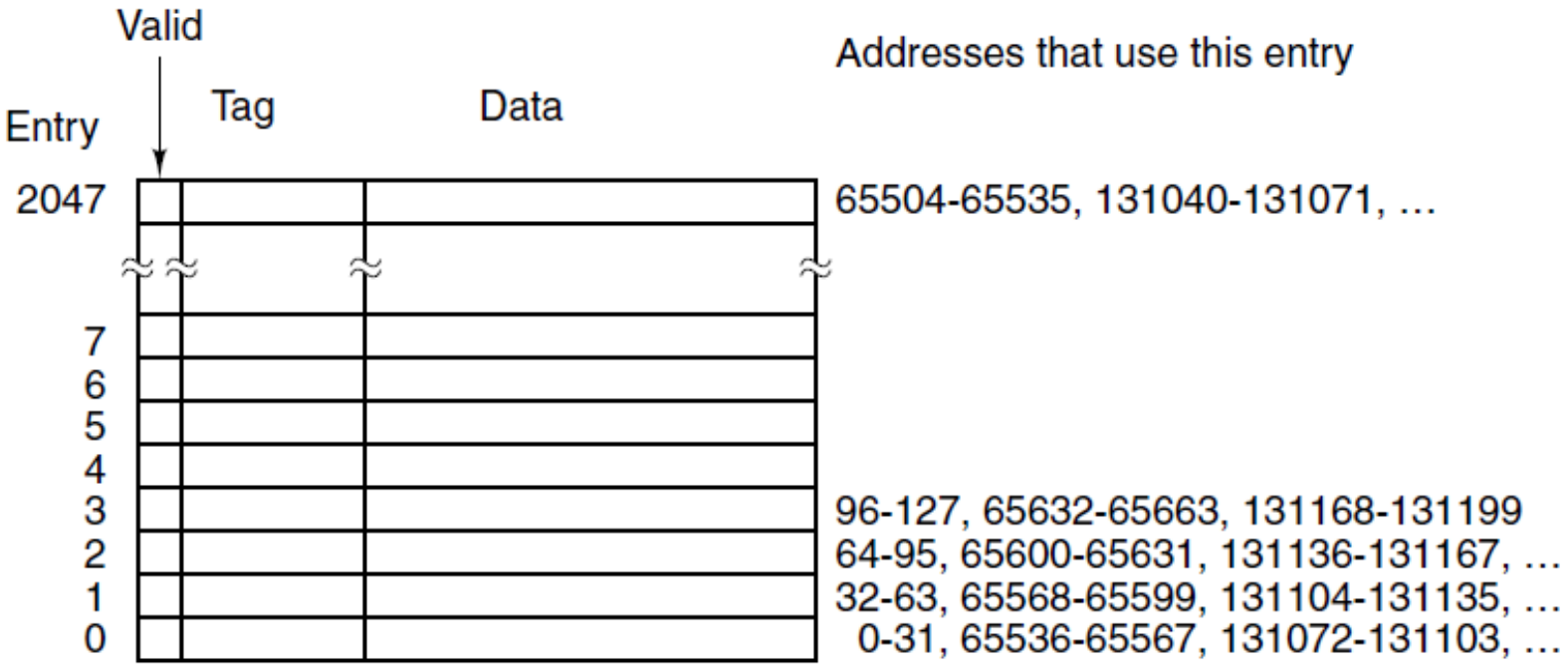
La memoria principale viene suddivisa in regioni uguali (es.: 32 o 64 byte) di locazioni contigue, ciascuna delle quali è mappata staticamente in una e una sola linea di cache.

A fronte di un cache miss, l'intera linea di cache viene aggiornata con una nuova linea contenente il dato.

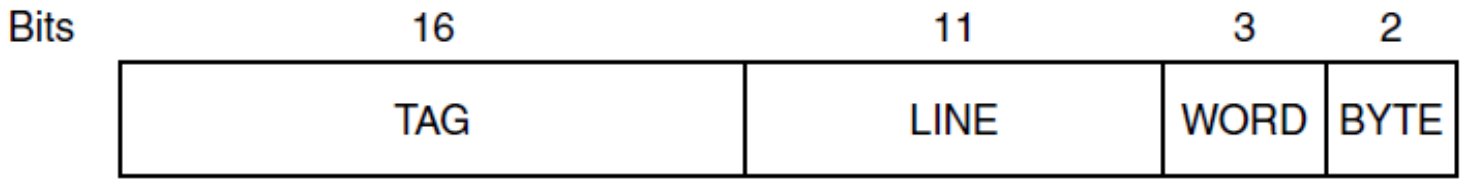
Infine, linee di cache contigue contengono regioni contigue della memoria principale.

Es.: cache con 2048 linee di 32 byte

Indirizzi e word di 32 bit; locazioni di un byte



(a)



(b)

Indirizzamento del dato in cache

L'indirizzo viene interpretato dall'hardware direttamente:

- TAG: individua la regione della memoria mappata sulla linea
- LINE: individua la linea di cache
- WORD: word all'interno della linea di cache
- BYTE: byte all'interno del word.

Se il campo TAG della linea corrisponde ai 16 bit più significativi dell'indirizzo allora c'è cache hit, e la ricerca del dato nella linea prosegue.

Ogni linea contiene anche un bit di validità dei dati.

Es.: la cache occupa $2^{11} \cdot (1 + 16 + 2^3 \cdot 2^5)/2^3$ byte.

Accesso diretto: vantaggi e svantaggi

La cache appena vista è detta ad accesso diretto.

Vantaggi:

- hardware semplice
- accesso molto veloce.

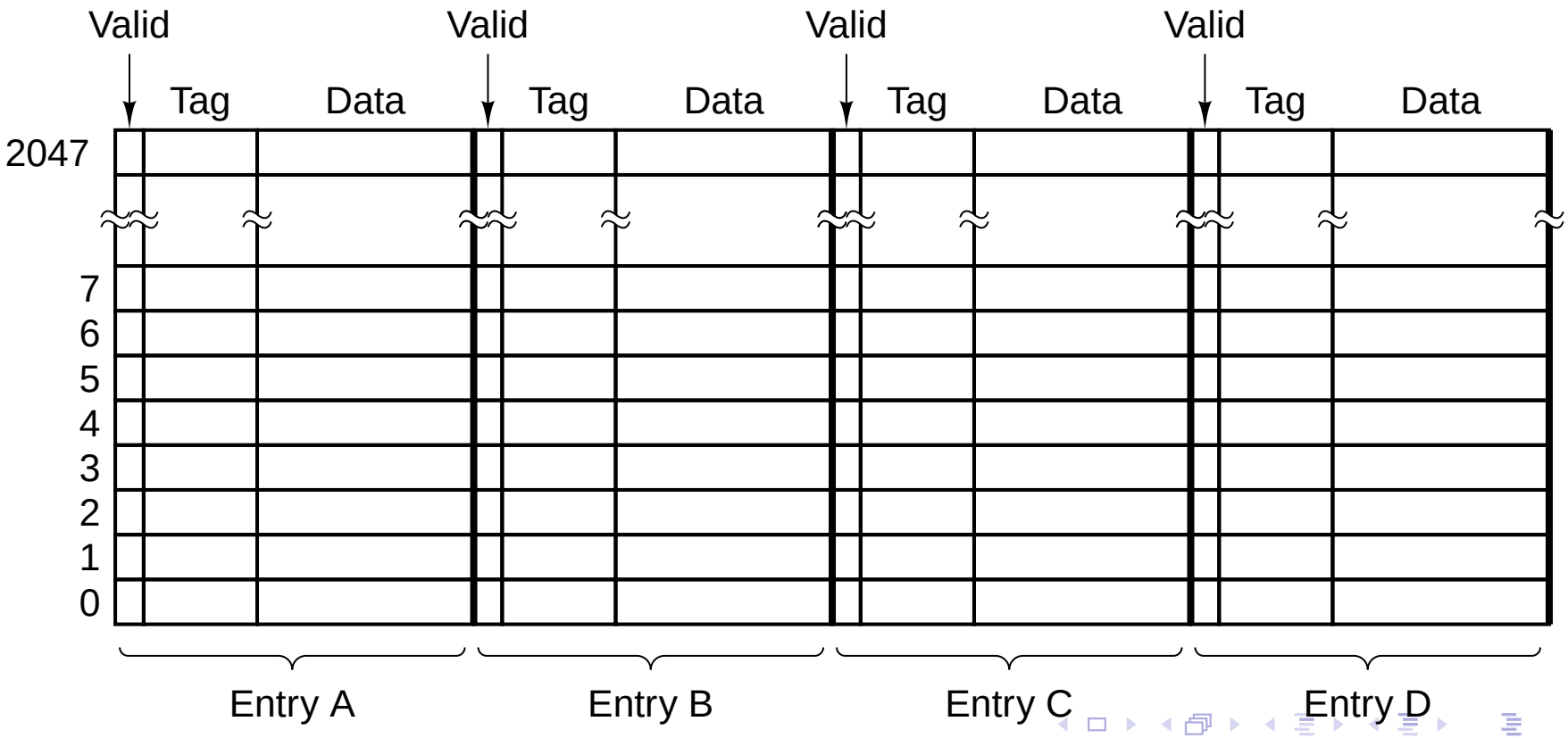
Svantaggio:

- diventa problematica per programmi grandi, che possono generare un elevato numero di cache miss.

Es. critico (diverso campo TAG, identico campo LINE): un programma accede ripetutamente a due locazioni contenute in regioni di memoria distinte che però mappano sulla stessa linea di cache.

Cache associativa a *n* vie

La cache associativa a *n* vie sfrutta anche la località temporale: unione di *n* tabelle (cache ad accesso diretto) su cui depositare dati da regioni di memoria che mappano sulla stessa linea.



Aggiornamento cache associative

La cache associativa è più complessa da realizzare:

- evita i casi critici di accesso a regioni che mappano sulla stessa linea di cache
- in caso di cache miss, la località temporale suggerisce di sovrascrivere la linea usata meno recentemente: least recently used (LRU)
- per ogni set di linee, una lista deve descrivere l'ordine di accesso alle tabelle.

Architetture attuali:

- Core i7: 4/8/16-way associative (a seconda del livello)
- ARM Cortex A15: 2/16-way associative.

Cache: scrittura in linea di cache

La scrittura in memoria in linea di principio non è un'operazione bloccante per il processore, che potrebbe non attenderne il completamento.

Alternative per la scrittura del dato nella cache

- write through: si scrive nella cache e nella memoria principale. Mantiene la coerenza dei dati ma genera più accessi alla memoria
- write deferred, write back: si scrive solo nella cache e si segna il bit di validità (dirty bit); la memoria principale è aggiornata quando l'accesso si rende necessario per sovrascrivere la linea di cache. Generalmente più efficiente.

Cache: scrittura fuori linea di cache

La scrittura in cache ha senso solo se il dato è in linea.

Se la locazione contenente il dato non è in linea,

- si scrive solo in memoria principale (no-write allocate),
- la regione di memoria contenente il dato prima viene portata in linea; poi, il dato è scritto e il bit di validità è marcato (write allocate).

Poichè write allocate non richiede la coerenza della memoria principale, usualmente

- no-write allocate si accompagna a write through
- write allocate si accompagna a write back.

Memoria virtuale

Il problema della carenza e della protezione della memoria principale si manifesta da subito (anni ’50), quando più utenze per motivi di costi eseguivano programmi diversi su un unico mainframe computer.

Il problema viene risolto espandendo virtualmente la memoria principale, utilizzando parte della memoria di massa per mantenere temporaneamente regioni di memoria principale statisticamente meno in uso.

A fronte di una disponibilità virtualmente illimitata di memoria principale, la memoria virtuale è inoltre

- suddivisa in pagine di memoria
- suddivisa in segmenti ad accesso limitato.

Memoria virtuale: indirizzamento

Ideati negli anni ’60, gli overlay erano regioni di memoria esplicitamente caricate e scaricate dalla memoria di massa durante l’esecuzione.

La memoria virtuale (anni ’70) delega la gestione degli overlay al sistema operativo in forma di pagine di memoria. Il sistema operativo a questo punto

- fornisce uno spazio d’indirizzamento sostanzialmente illimitato a ogni programma
- mappa gli spazi d’indirizzamento nell’insieme (limitato e disgiunto) degli indirizzi fisici
- organizza gli indirizzi in pagine collocate in memoria principale oppure di massa a seconda degli accessi a cui sono soggette.

Caratteristiche della pagina

La pagina di memoria

- è caratterizzata da un'estensione comune a tutte le pagine
- ha una dimensione che è una potenza di 2, al fine di ottimizzarne l'indirizzabilità in analogia alla dimensione della memoria principale
- può essere presente in memoria principale e in memoria di massa (analogia col dato in cache)
- può essere presente solo in memoria di massa
- può non essere presente (pagine fisicamente inesistenti. Es.: heap virtuale a disposizione delle strutture dati dinamiche).

Accesso alla memoria paginata

L'accesso alla memoria paginata ha analogie con l'accesso alla memoria cache. Data una richiesta di accesso alla memoria, il sistema operativo

- determina il numero della pagina cui appartiene l'indirizzo virtuale. Nel caso più semplice:
  pagina = indirizzo virtuale / dimensione pagina

Accesso alla memoria paginata

L'accesso alla memoria paginata ha analogie con l'accesso alla memoria cache. Data una richiesta di accesso alla memoria, il sistema operativo

- determina il numero della pagina cui appartiene l'indirizzo virtuale. Nel caso più semplice: pagina = indirizzo virtuale / dimensione pagina
- controlla se la pagina è presente nella memoria principale

Accesso alla memoria paginata

L'accesso alla memoria paginata ha analogie con l'accesso alla memoria cache. Data una richiesta di accesso alla memoria, il sistema operativo

- determina il numero della pagina cui appartiene l'indirizzo virtuale. Nel caso più semplice: pagina = indirizzo virtuale / dimensione pagina
- controlla se la pagina è presente nella memoria principale
- se non è presente (page fault) carica la pagina dalla memoria di massa, sovrascrivendo una pagina presente in memoria principale
- calcola l'indirizzo fisico in memoria principale della locazione richiesta e accede al dato.

Mappatura degli indirizzi virtuali

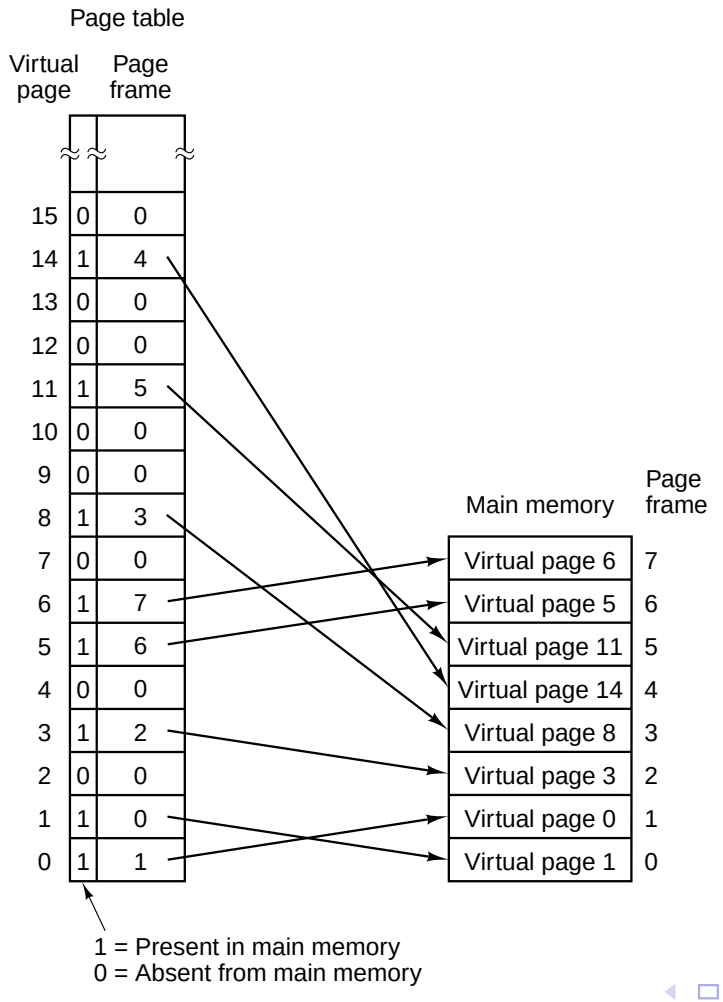
A differenza della memoria cache, una pagina può risiedere ovunque in memoria principale: si rinuncia alla velocità massima a vantaggio di una maggior flessibilità.

Viene utilizzata una tabella delle pagine (page table), chiamata anche mappa della memoria (PMT: page-map table), che a ogni pagina associa

- un bit che ne segnala la presenza o meno in memoria principale
- l'indirizzo fisico di inizio pagina in memoria principale (indirizzo base), valido se il bit di presenza è asserito.

Tabella delle pagine

Es. (giocattolo): memoria virtuale di 16 pagine, 8 pagine fisiche



Accesso a una locazione in memoria

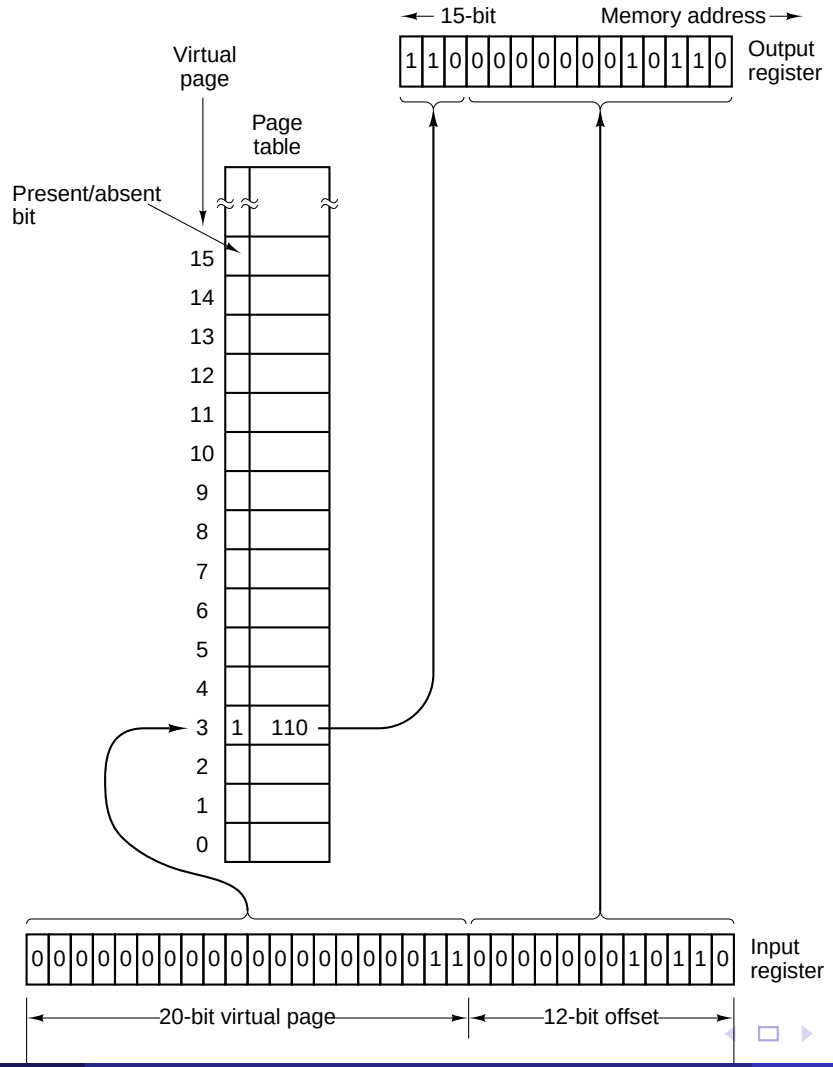
Se la pagina è presente in memoria principale,

- il resto della divisione che dà il numero di pagina è la posizione del dato nella pagina (offset)
- recuperato l'indirizzo base dalla tabella delle pagine, l'indirizzo fisico della locazione contenente il dato si ottiene giustapponendo indirizzo base e offset:
  [indirizzo fisico] = [indirizzo base offset].

Se la pagina non è presente è eseguita una chiamata di sistema: il sistema operativo carica la pagina dalla memoria di massa e aggiorna la tabella delle pagine.

Calcolo dell'indirizzo fisico

Es.: memoria virtuale di 2^{20} pagine, con 8 pagine fisiche di 2^{12} byte



Es.: calcolo di un indirizzo fisico

Calcoliamo l'indirizzo fisico della locazione di indirizzo virtuale a 32 bit

`I = 00000000000000001100000010110`

appartenente a una memoria virtuale di 2^{20} pagine, con 8 pagine fisiche di 2^{12} byte (N.B.: 20+12=32):

- `I / 2`^{12} = 11_2, resto 10110_2
- la page table alla riga 11_2 mostra il bit di validità asserito e indica l'indirizzo base 110_2
- la locazione è dunque presente all'indirizzo fisico 11010110_2.

N.B.: il riempimento ottimale della memoria principale si ottiene se gli indirizzi base sono multipli della dimensione della pagina.

Memorizzazione della page table

La page table non può logicamente essere paginata. In pratica non lo è nemmeno in parte per motivi di efficienza: non avrebbe senso accedere alla memoria di massa durante la mappatura di un indirizzo virtuale.

Come vedremo, ogni programma in esecuzione ha una page table dedicata (es.: 10 MB). La cache quindi non può contenere tutte le page table.

Per non perdere efficienza, parte delle page table è contenuta in una memoria cache dedicata (Translation Lookside Buffer, TLB) contenuta all'interno dalla Memory Management Unit (MMU), un hardware specializzato posto nel chip della CPU.

Page frame

Il page frame è una regione della memoria principale contenente una pagina fisica.

Di norma esiste un'area di memoria principale dedicata a contenere i page frame.

Page	Virtual addresses
15	61440 – 65535
14	57344 – 61439
13	53248 – 57343
12	49152 – 53247
11	45056 – 49151
10	40960 – 45055
9	36864 – 40959
8	32768 – 36863
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(a)

Page frame	Bottom 32K of main memory Physical addresses
7	28672 – 32767
6	24576 – 28671
5	20480 – 24575
4	16384 – 20479
3	12288 – 16383
2	8192 – 12287
1	4096 – 8191
0	0 – 4095

(b)

Alcuni indirizzi di memoria non vengono virtualizzati.
Es.: memory mapped I/0.

Dimensioni della pagina

Conviene avere pagine grandi per

- ridurre il numero di accessi alla memoria di massa
- sfruttare meglio la località spaziale dei dati
- avere page table compatte.

Dimensioni della pagina

Conviene avere pagine grandi per

- ridurre il numero di accessi alla memoria di massa
- sfruttare meglio la località spaziale dei dati
- avere page table compatte.

Conviene avere pagine piccole per

- sfruttare meglio l'area di memoria principale dedicata ai page frame
- ridurre il tempo di un accesso alla memoria di massa.

Anni '70: pagine di 0.5 – 1 KB.
Attualmente: pagine di 4 KB $\sim$ 4 MB.

Page fault

Ogni page fault genera una trap. La relativa procedura di sistema

- cerca un page frame vuoto
- se non esiste, sovrascrive un page frame secondo due possibili politiche:
  - quello usato meno recentemente (least recently used, LRU); sfrutta la località temporale
  - quello caricato meno recentemente (first-in first-out, FIFO); più semplice da realizzare
- se la pagina da sovrascrivere è stata modificata (meccanismo del dirty bit) preliminarmente aggiorna la copia in memoria di massa
- carica la pagina cercata nella memoria principale.

Il working set delle pagine

Un processo è un'istanza in esecuzione di un programma. I page frame devono dunque essere ripartiti tra tutti i processi.

Il working set è l'insieme delle pagine di uso corrente. Es.: quelle usate dai processi nell'ultimo secondo.

Il working set dovrebbe essere sempre contenuto in memoria principale, pena il rallentamento del sistema a causa di continui page fault (thrashing).

Nei casi di memoria principale insufficiente, il thrashing può essere mitigato adattando l'algoritmo eseguito dalla procedura di page fault.

Distribuzione di pagine tra i processi

Assegnare a ogni processo una page table distinta permette di assegnare identici indirizzi di memoria virtuale a processi identici.

Quindi, è utile suddividere la memoria virtuale assegnando ciascuna pagina a un solo processo.

Al lancio di un processo sono possibili due opzioni:

- un sottoinsieme di pagine viene automaticamente caricato in memoria principale
- nessuna pagina caricata in memoria principale: demanding paging, paginazione su richiesta.

Normalmente le pagine assegnate ai processi costituiscono un sovrainsieme del working set.

Limiti della paginazione

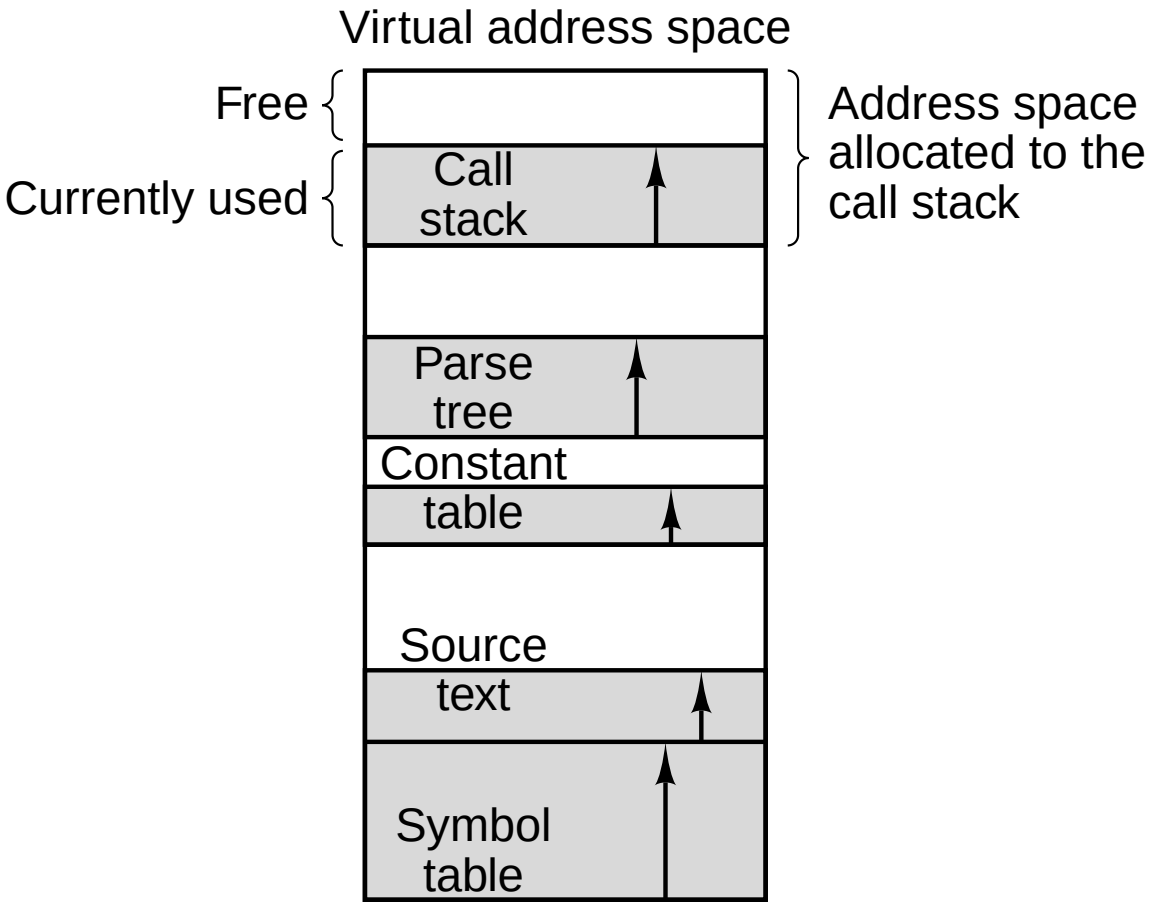
La paginazione non fornisce ai programmi utente e al sistema operativo regioni logicamente distinte della memoria virtuale. Es.:

- aree per sistema operativo e per chiamate di sistema
- aree per i programmi utente
- aree per le procedure eseguite dai programmi utente
- sottoaree per il singolo programma utente.

Per questo motivo, non fornisce ai programmi utente e al sistema operativo un meccanismo di protezione della memoria dall'accesso da parte di altri processi.

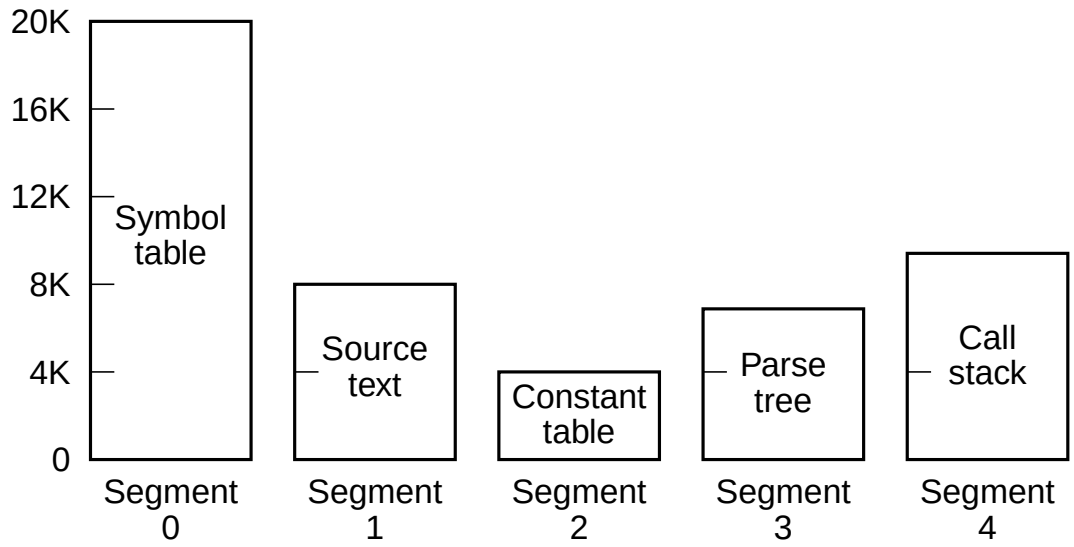
La memoria di un processo

Ogni processo necessita di memoria per: letterali, istruzioni, costanti, dati variabili, procedure.



Suddivisione logica della memoria

La memoria di un processo quindi consta di segmenti logicamente indipendenti che occupano regioni distinte.



In una memoria virtuale segmentata dunque ogni processo accede a segmenti logici distinti della memoria virtuale anche in assenza di paginazione.

Descrittore del segmento

Un processo accede ai dati contenuti nei propri segmenti attraverso i descrittori e l'offset del dato.

Il descrittore del segmento deve contenere

- il numero del segmento, che ne individua la posizione in memoria virtuale
- il livello di protezione, che individua l'accessibilità del segmento.

Una tabella dei segmenti mappa ogni numero nel corrispondente indirizzo fisico. In assenza di indirizzo fisico il sistema operativo carica il segmento in memoria principale attraverso un meccanismo simile a quello della paginazione.

Segmentazione non paginata

In un sistema senza virtualizzazione paginata, il caricamento di nuovi segmenti prima o poi costringe a scaricare dalla memoria principale i segmenti dei processi meno adoperati (swapping).

La rimozione dei segmenti è complessa poichè deve tener conto della loro lunghezza e del loro utilizzo più recente.

Poichè i segmenti hanno dimensione variabile, la ricerca di regioni libere in memoria principale durante lo swapping è problematica.

Lo swapping genera spazi liberi residui di piccole dimensioni in memoria principale (checkerboarding).

Ottimizzazione dello spazio libero

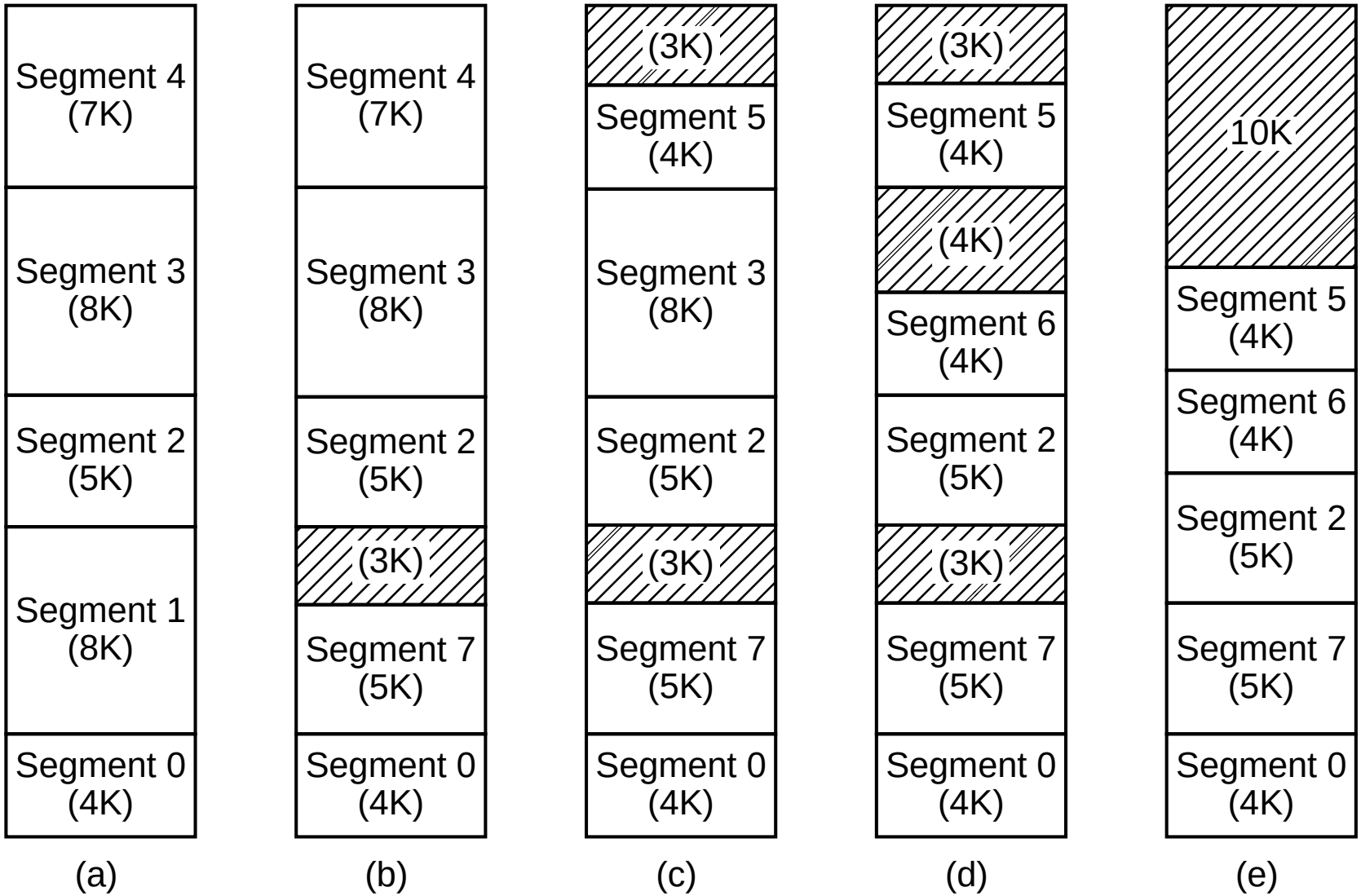
Le strategie di inserimento di un nuovo segmento in memoria principale includono la ricerca

- dello spazio libero più piccolo: best fit, complicato da realizzare
- del primo spazio libero trovato: first fit, semplice e con buone prestazioni.

Un fallimento della ricerca costringe a unire gli spazi liberi residui attraverso lo spostamento e la riunione di segmenti in memoria (compattazione).

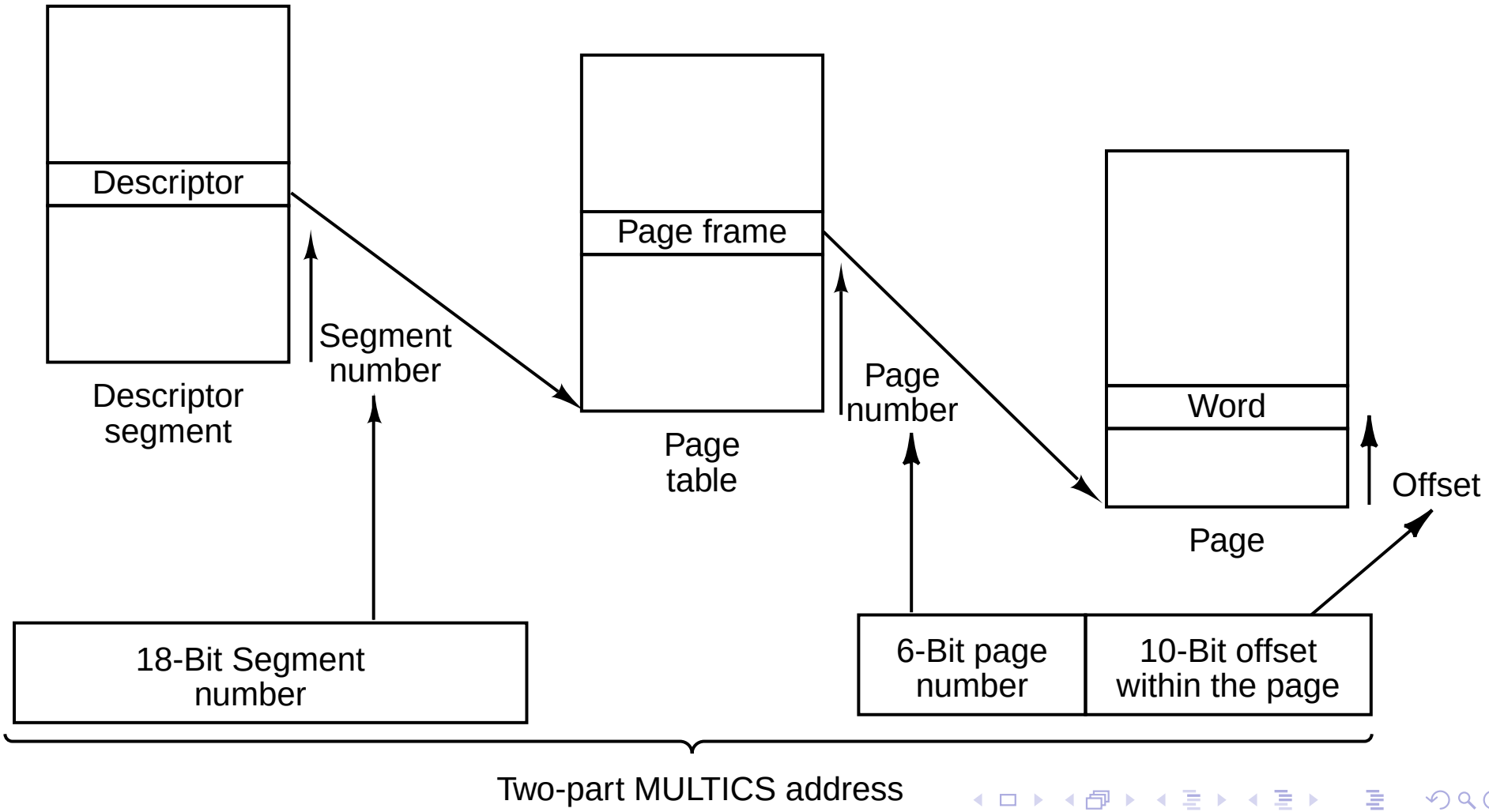
La compattazione richiede tempo e non può essere utilizzata frequentemente. La segmentazione non paginata è quindi problematica.

Es.: swapping e compattazione



Segmentazione paginata: MULTICS

Ogni segmento è diviso in pagine. Il descrittore del segmento indirizza una page table del segmento.



Modello di memoria nei processori x86 (cenni)

Modello di memoria retrocompatibile: memoria semplice oppure segmentata e/o paginata.

- Esistenza di segmenti codice e segmenti dati
- Segmenti locali al processo e globali nel sistema
- Local Descriptor Table (LDT)
- Global Descriptor Table (GDT)

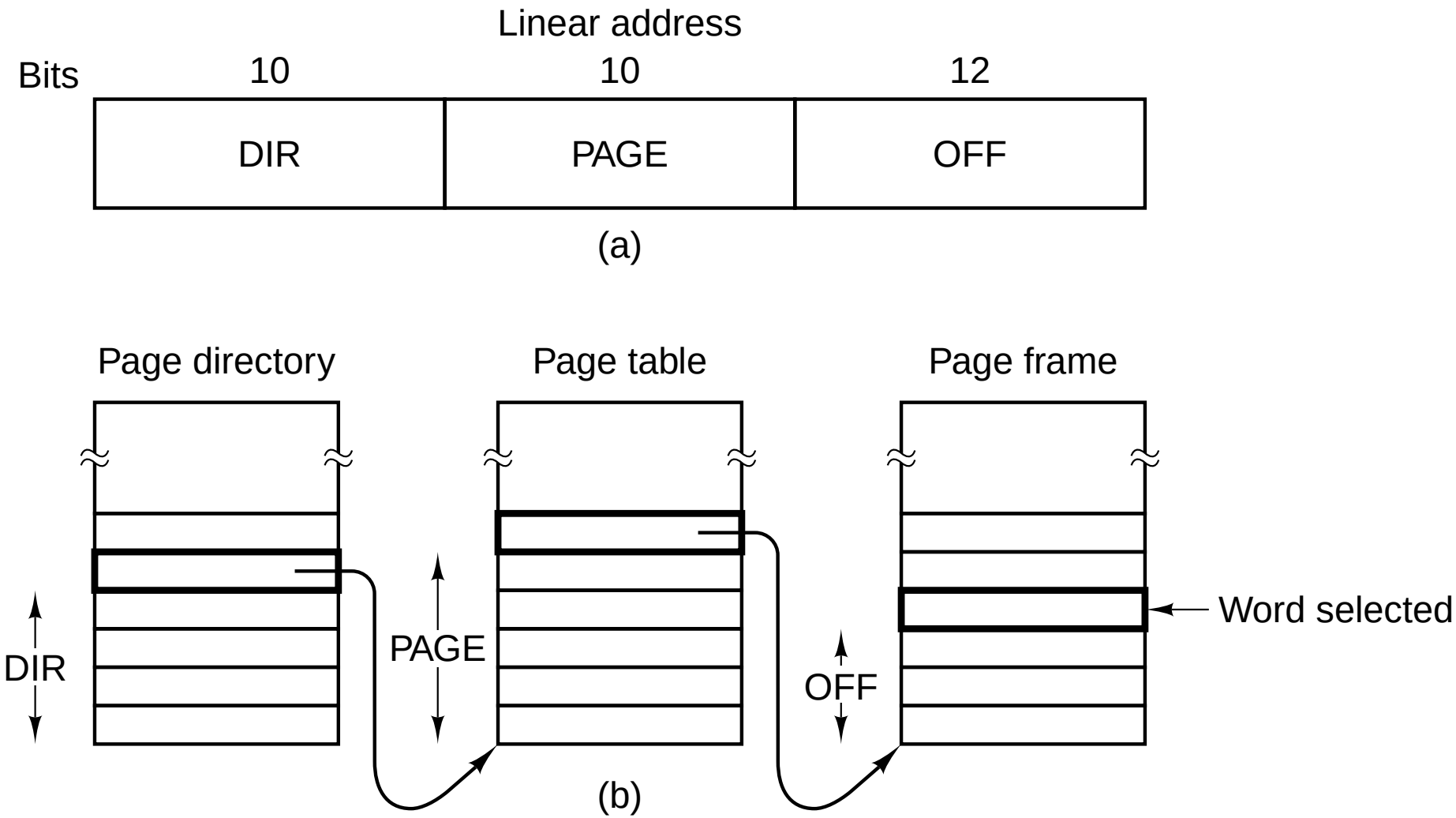
Modello di memoria nei processori x86 (cenni)

Modello di memoria retrocompatibile: memoria semplice oppure segmentata e/o paginata.

- Esistenza di segmenti codice e segmenti dati
- Segmenti locali al processo e globali nel sistema
- Local Descriptor Table (LDT)
- Global Descriptor Table (GDT)
- Paginazione non attivata: accesso diretto alla memoria (segmentazione pura)
- Paginazione attivata: accesso tramite paginazione (segmentazione paginata).

x86: Calcolo dell'indirizzo lineare

Esistenza di una page directory (tabella delle tabelle)

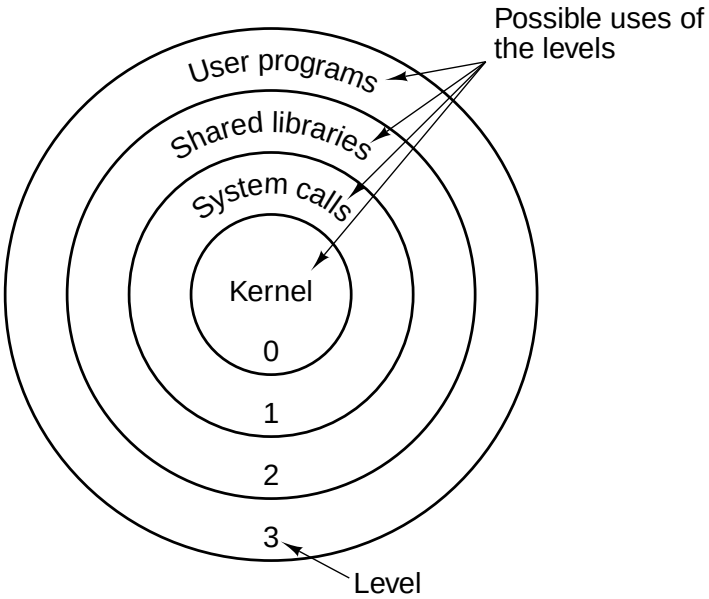


x86: Organizzazione page tables

- Ogni segmento accede a una page table.
- Page table divisa in più sottotabelle.
- Problemi di dimensione: parte della page table contenuta in memoria secondaria.
- Tempi di accesso: nella MMU una memoria cache per i collegamenti segmento–pagina usati più recentemente.
- Utilizzo di un unico segmento $\rightarrow$ paginazione pura.

x86: Protezione

Un processo che ha livello di privilegio N (come da PSW, program status word) non può accedere a segmenti con livello di protezione minore di N.



Un processo accede a livelli inferiori di protezione solo attraverso opportune call gate: procedure di sistema richiamabili da punti di ingresso ufficiali.