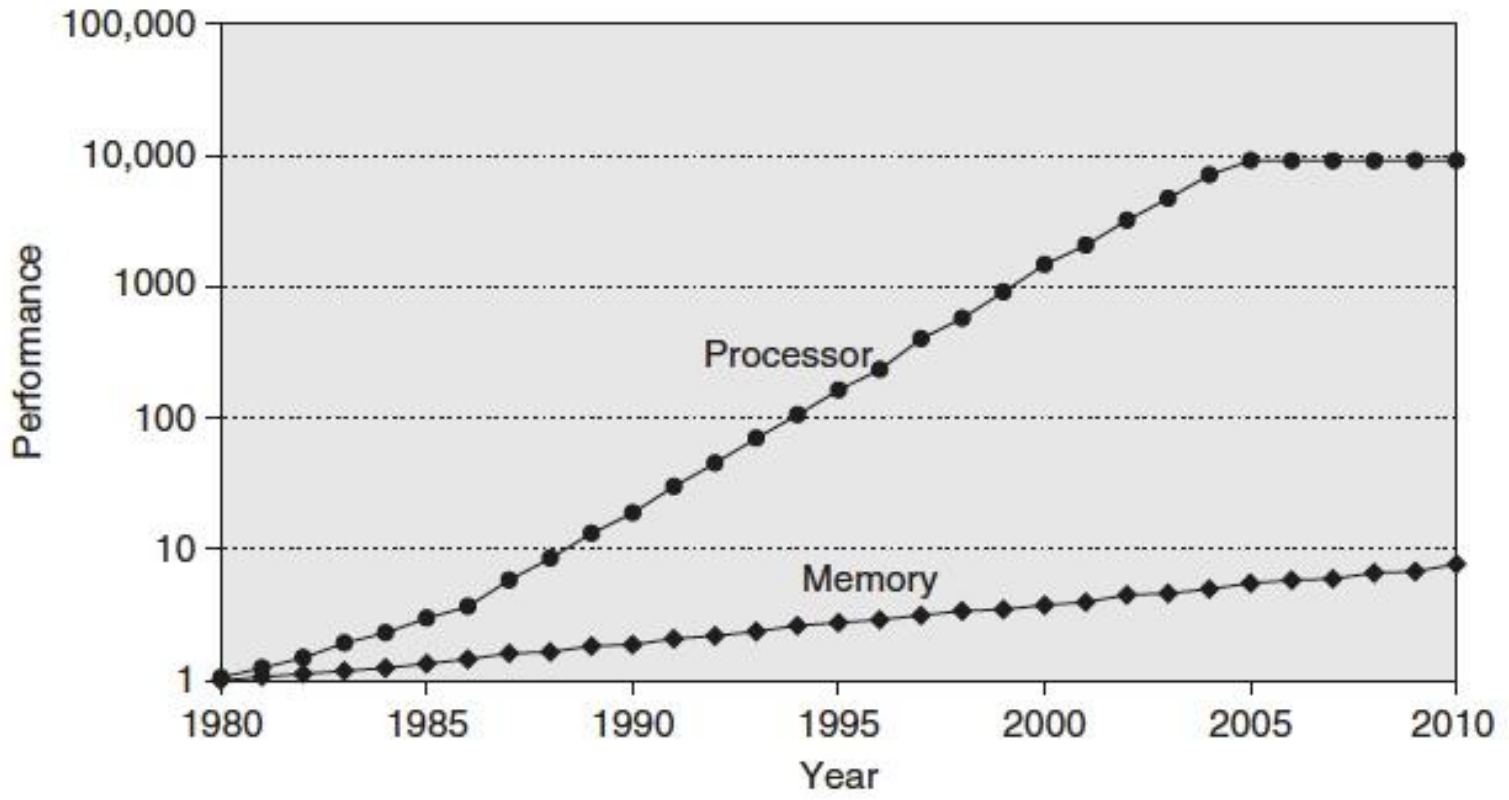
Arhitectura Calculatoarelor

Curs 12: Memorii

E-mail: florin.oniga@cs.utcluj.ro

Web: http://users.utcluj.ro/~onigaf, secțiunea Teaching/AC

Performanța procesorului vs. a memoriei



[1]

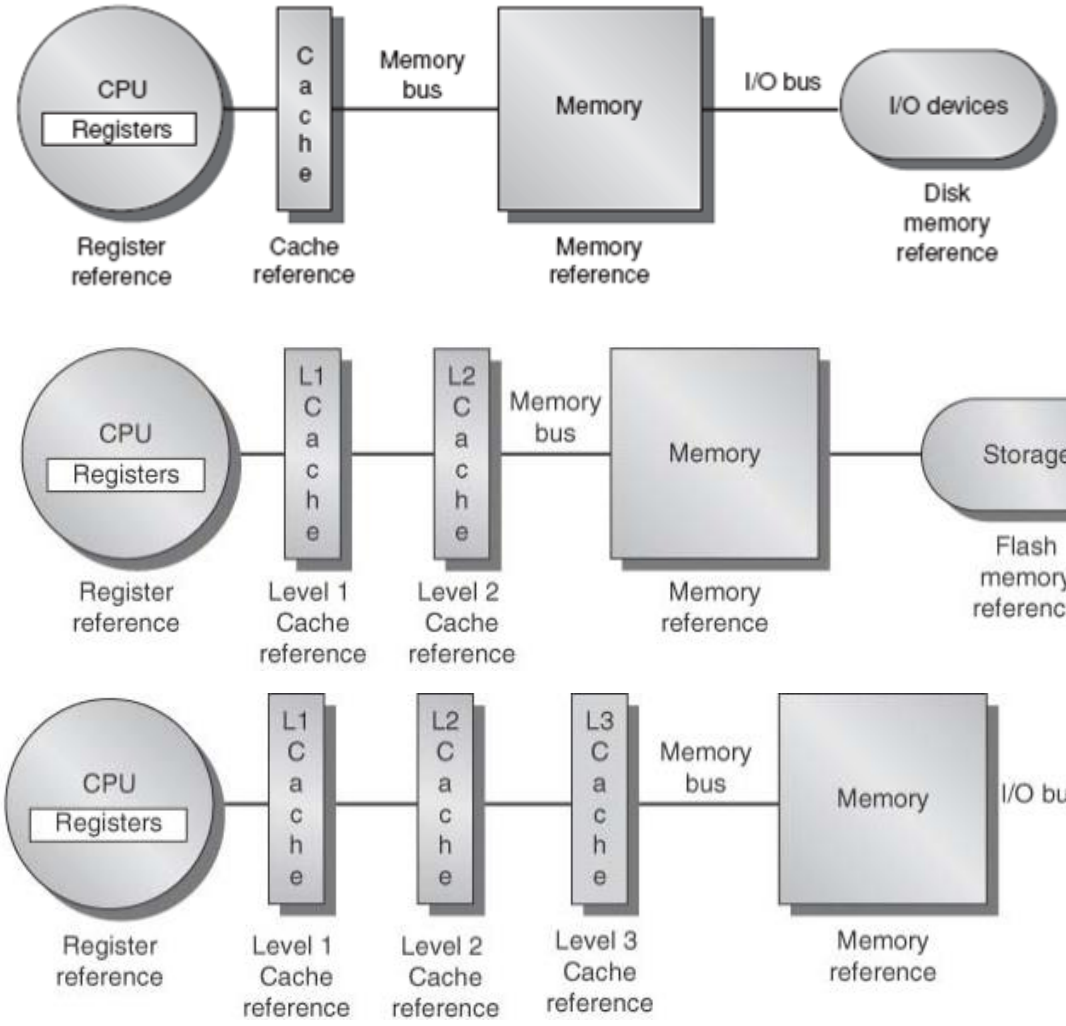
Discrepanță crescută: performanța procesorului (nucleu unic) vs memorie (DRAM)

Tehnologie	Timp de acces tipic	Cost / GB în 2012
SRAM semiconductor memory	0.5 – 2.5 ns	$500 – $1000
DRAM semiconductor memory	50 – 70 ns	$10 – $20
Flash semiconductor memory	5,000 – 50,000 ns	$0.75 – $1.00
Magnetic disk	5,000,000 – 20,000,000 ns	$0.05 – $0.10

Tipuri de memorii: timp de acces, pret (2012)

Ierarhii de memorie

Exemple de ierarhii de memorie, multi-nivel



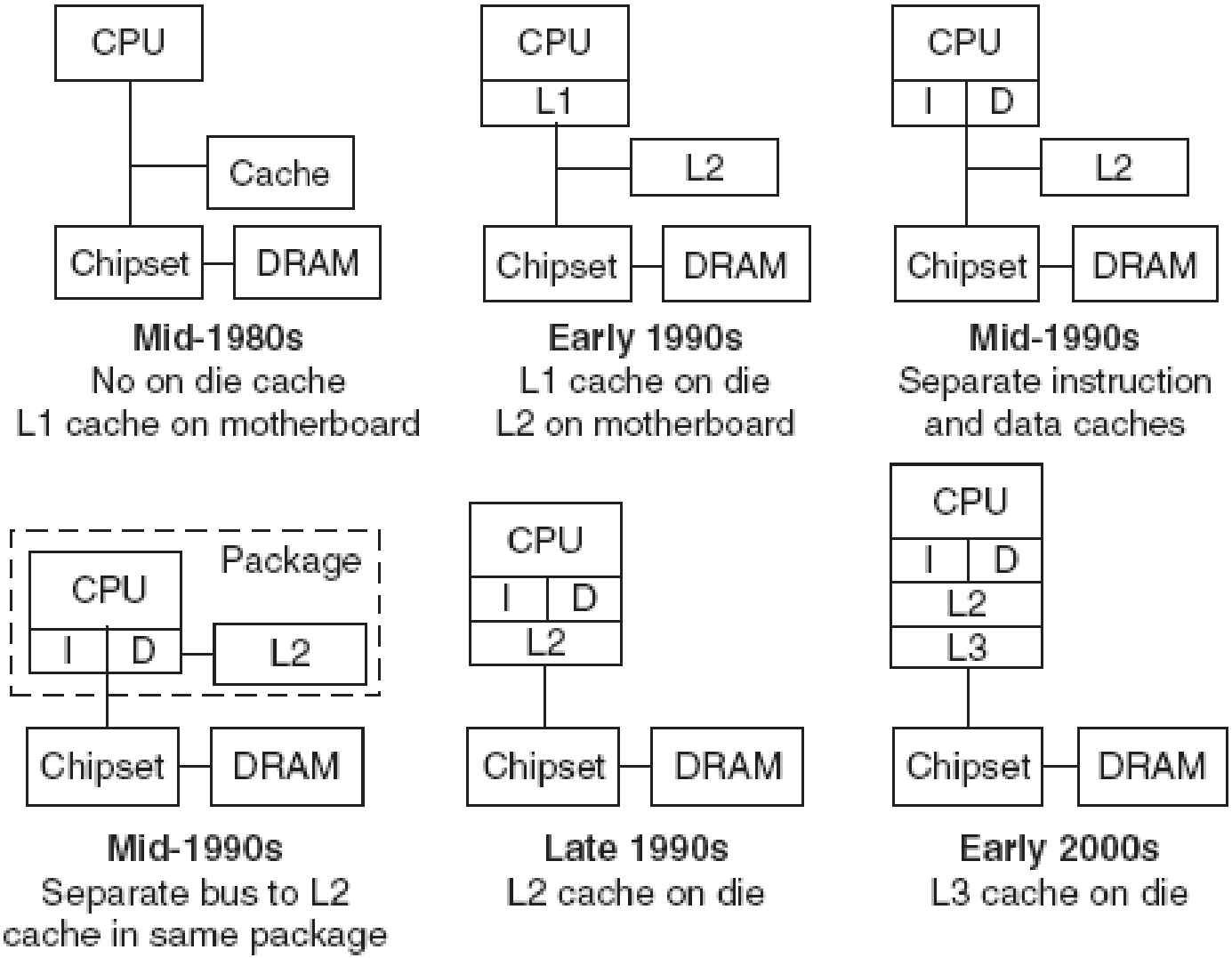
Soluţie: Memorie ierarhică

Scop: iluzie de memorie mare, rapidă și ieftină

[1]

Cu creșterea distanței de la procesor => scade viteza / crește capacitatea

Ierarhii de memorie - Evoluție



Evoluția memoriei ierarhice, Memorii cache separate pentru instrucțiuni și date

De ce funcționează ierarhizarea (oferă performanța dorită la un cost redus) ?

Memoria cache – Ideea de bază

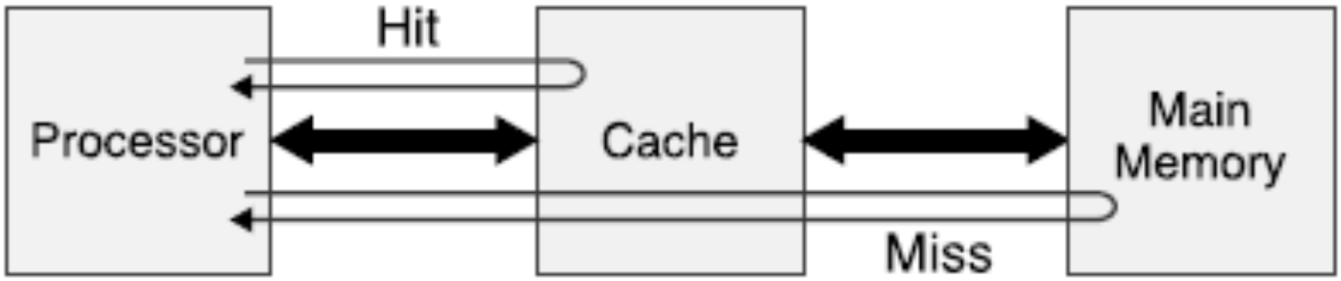
Principiul Localizării (Locality in eng.)

1. **Localizare Temporală**
   - O locație a fost accesată - probabil va fi accesată din nou, în curând
   - Se păstrează aproape de procesor datele recent accesate

2. **Localizare Spațială**
   - O locație a fost accesată - probabil ca adresele vecine vor fi accesate în curând
   - Transfer de blocuri de cuvinte succesive la nivele superioare

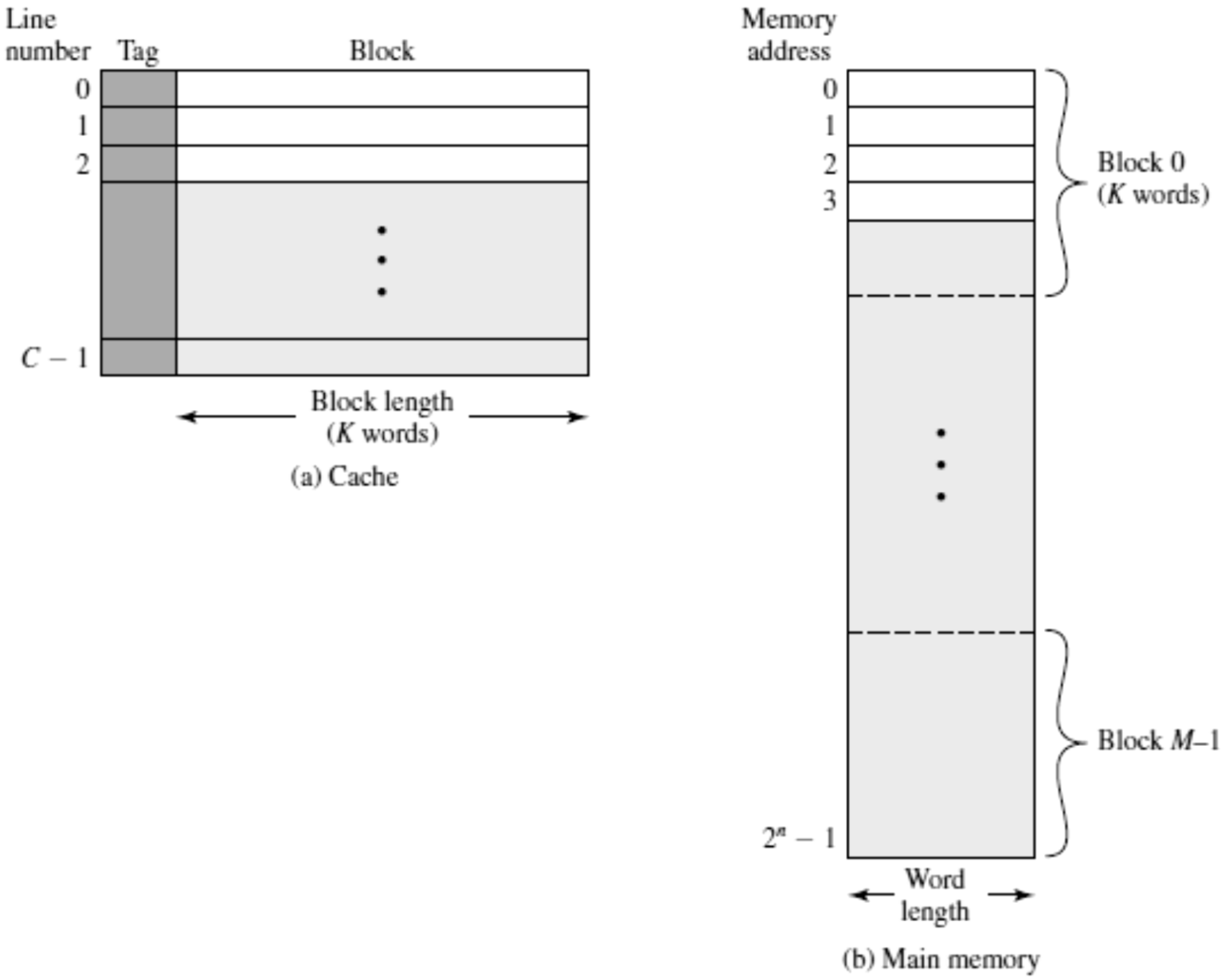
Memoria cache - Terminologie

- **hit**: data cerută se găsește în nivelul superior de cache.
- **miss**: data cerută nu se găsește în nivelul superior.
- **hit rate**: fracțiunea de accese la memorie care sunt *hit*
- **miss rate**: fracțiunea de accese la memorie care nu sunt hit: miss rate = 1 – hit rate
- **hit time**: timpul necesar pentru a determina dacă un acces la memorie este hit + timpul necesar pentru a livra datele din nivelul superior de cache la CPU
- **miss penalty:** timpul necesar pentru a determina dacă un acces la memorie este miss + timpul necesar pentru a înlocui blocul din nivelul superior cu cel din nivelul inferior + timpul pentru a livra datele din nivelul superior de cache la CPU
- **Average Memory Access Time (AMAT)** = **Hit time** + **Miss rate** x **Miss penalty**.



- Îmbunătățirea performanței → reducere AMAT.
  - Reducere miss rate, miss penalty, sau hit time

Memoria cache



Noțiunea de bloc în cache, respectiv bloc în memorie

Memoria cache

Patru probleme de rezolvat pentru proiectarea unei memorii cache

Q1: Unde poate fi plasat un bloc în cache? *(Block placement)*

- Asociativitate: Asociativă total (Fully Associative), Asociativă pe set (Set Associative), Mapată direct (Direct Mapped)

Q2: Cum se găsește un bloc în cache? *(Block identification)*

- Tag / Index / Block

Q3: La un cache miss, cum se selectează blocul de înlocuit? *(Block replacement)*

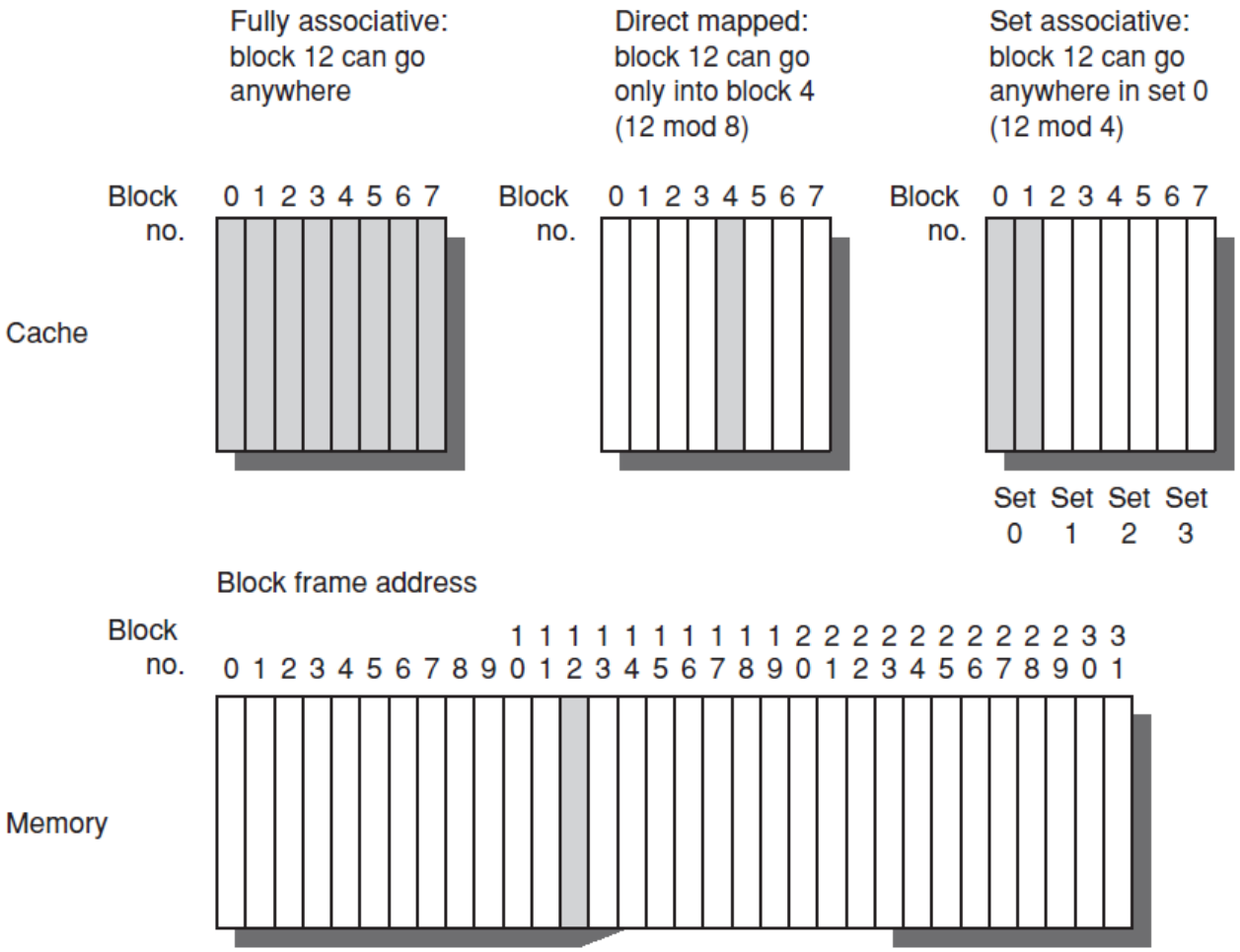
- Random, LRU, FIFO, etc.

Q4: Cum se realizează o scriere in cache?

- Write Back or Write Through, Write Buffer

Q1: Unde poate fi plasat un bloc în cache?

Organizarea memoriei cache



– Direct mapped: fiecare bloc are o singură locație corespunzătoare în cache;
Maparea adresei:
(Block address) MOD (Number of blocks in cache)

– Fully associative: un bloc poate fi plasat oriunde în cache.

– Set associative: un bloc poate fi plasat într-un set restrâns de locații în cache, n seturi de locații, „n-way set associative cache”
Maparea adresei:
(Block address) MOD (Number of sets in cache)

Exemplu: memorie cache cu 8 blocuri [1]

“The miss rate of a direct-mapped cache of size X is about the same as a 2- to 4-way set associative cache of size X/2.”

Q1: Unde poate fi plasat un bloc în cache?

Exemplu: Configurațiile unui cache cu 8 blocuri, diferite asociativități

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Direct Mapped Cache ⇔
1-way Set Associative Cache!

Set = un grup de blocuri, index în memoria cache

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

2-way Set Associative Cache

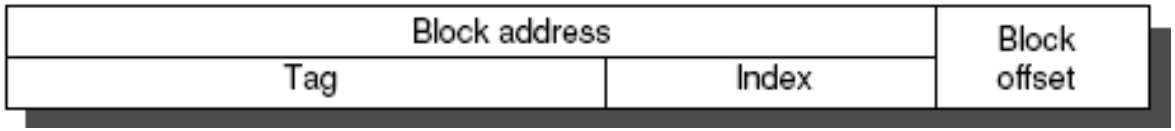
Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

4-way Set Associative Cache

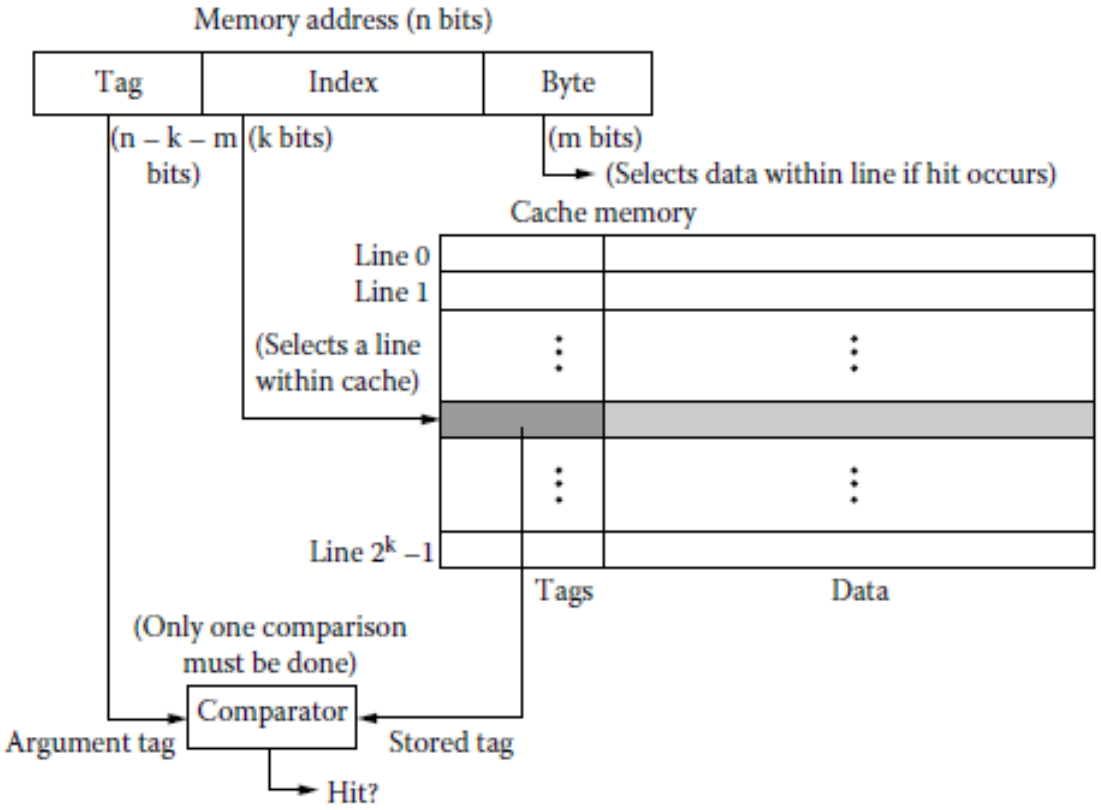
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

8-way Set Associative (Fully Associative)

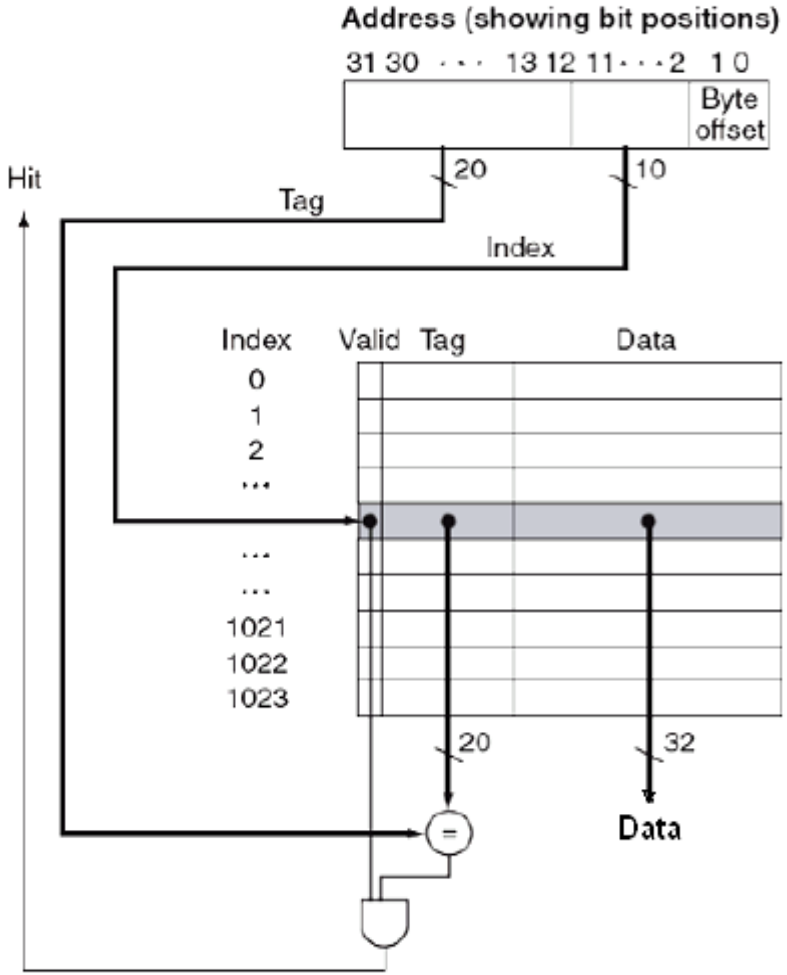
Q2: Cum se găsește un bloc în cache?



Adresa în cache pentru set-associative sau direct-mapped
Fully associative – fără câmpul de index

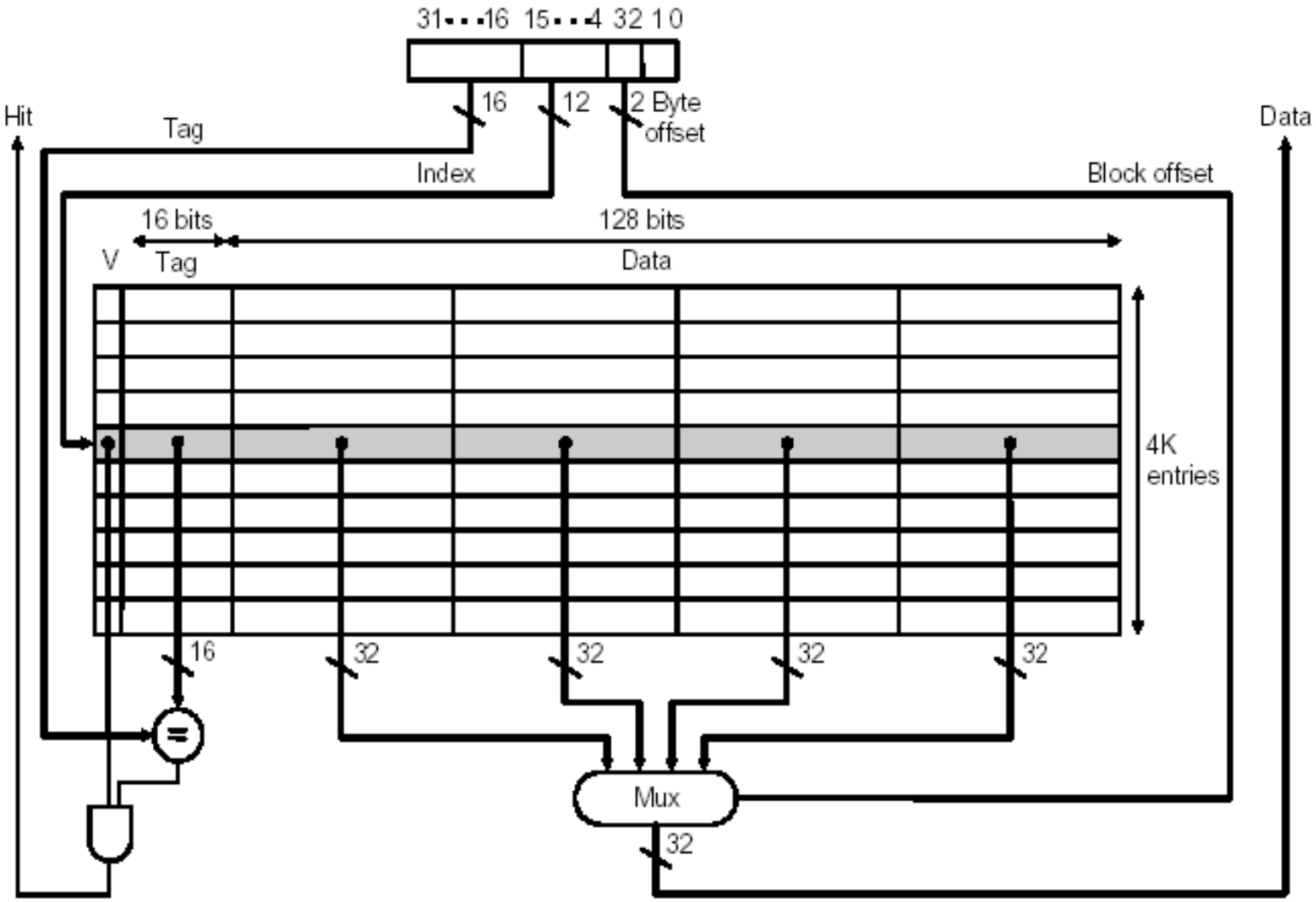


Direct mapped cache
(general, partea de adresare)



Direct Mapped Cache,1024 blocuri,
1 cuvânt (32 biți) / bloc

Q2: Cum se găsește un bloc în cache?



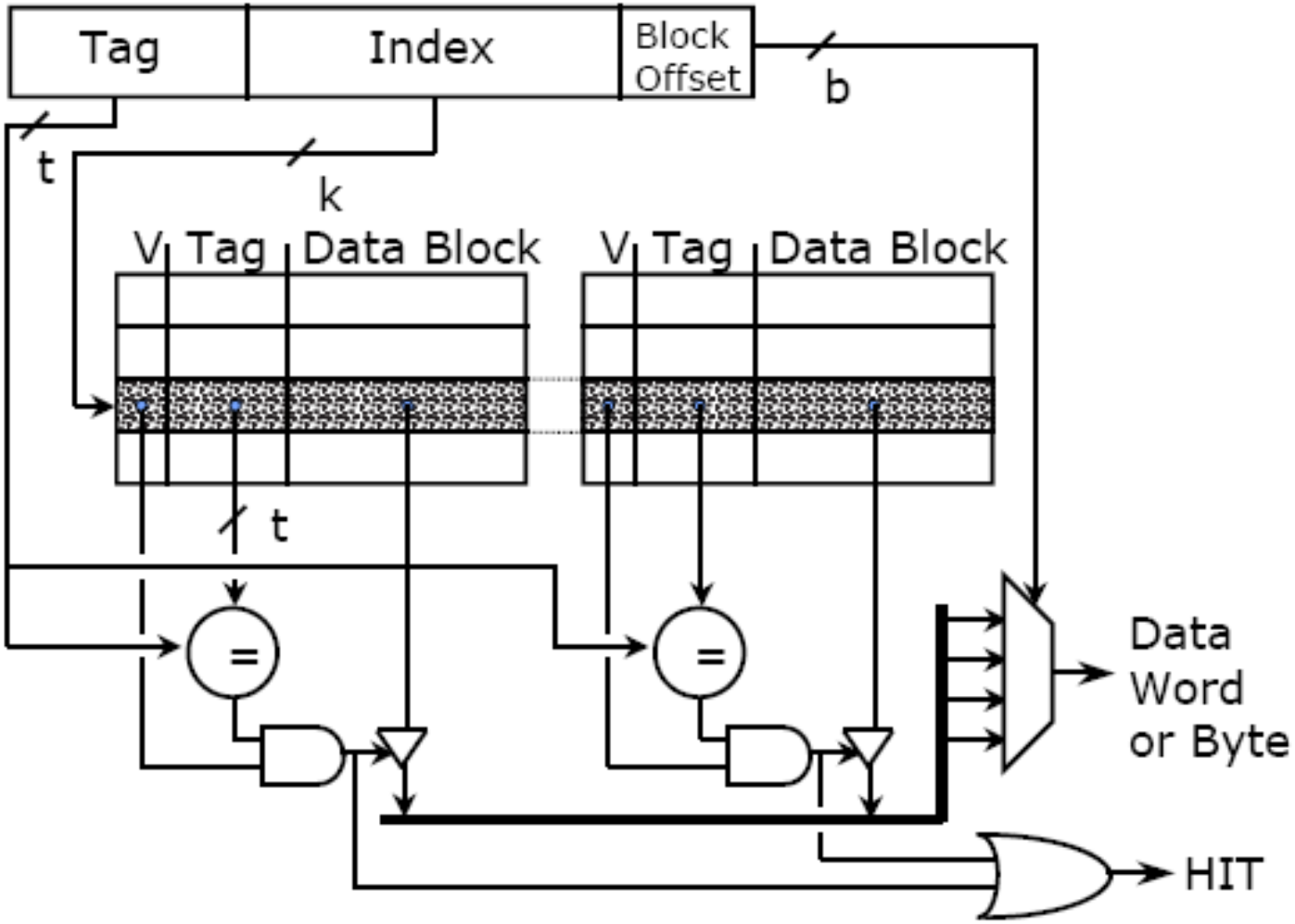
Direct mapped cache cu date (cuvinte) multiple / bloc. Exploatare – localitate spațială

- 64KB cache, 4k blocks, 4 cuvinte / bloc;
- Rolul biților din adresa: *byte offset* ignorat (se returnează cuvinte întregi), următorii 2 biți *block offset* (care cuvânt dintre cele 4 din bloc), următorii 12 biți reprezintă indexul blocului în cache

Q2: Cum se găsește un bloc în cache?

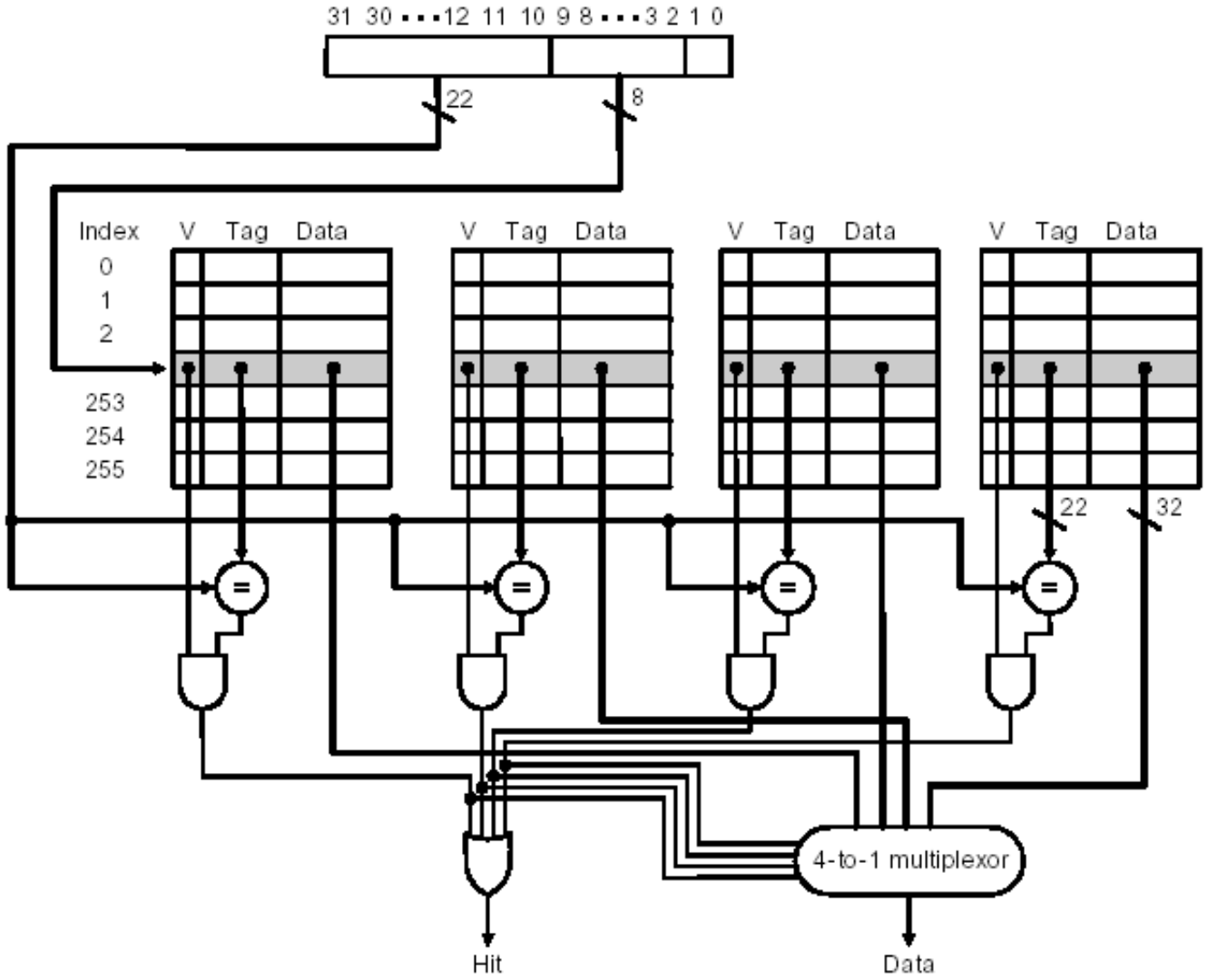
N-way set associative: N posibile locații pentru fiecare index

- (!) Implementarea constă din N cache-uri cu mapare directă operează în paralel



Exemplu: 2-Way Set-Associative Cache (adițional cu selecție între cuvânt / octet) / Asociativă pe set cu 2 căi

Q2: Cum se găsește un bloc în cache?



Exemplu: 4-way set associative cache (4-căi)

- 4-way set-associative cache cu 4 comparatoare și un multiplexor 4:1
- Mărime cache: 4 KB, format din 1K blocuri = 256 seturi x 4 blocuri / set, 1 cuvânt (4 octeți) / bloc

Q2: Cum se găsește un bloc în cache?

Cache set-associative, Avantaje

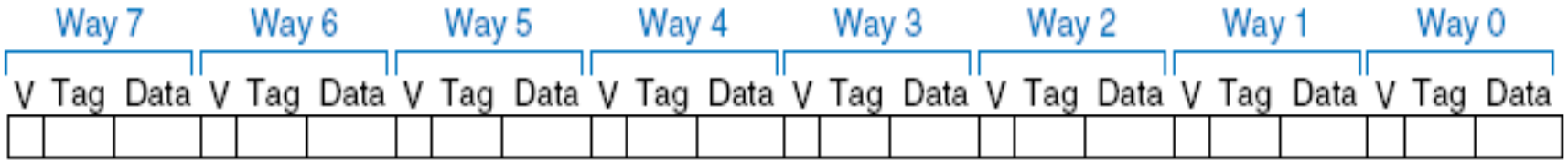
- Hit rate crește pentru aceeași dimensiune de cache
- Mai puține „Conflict Misses” (urmează definirea) - lipsuri datorită strategiei de înlocuire

Cache set-associative, Dezavantaje

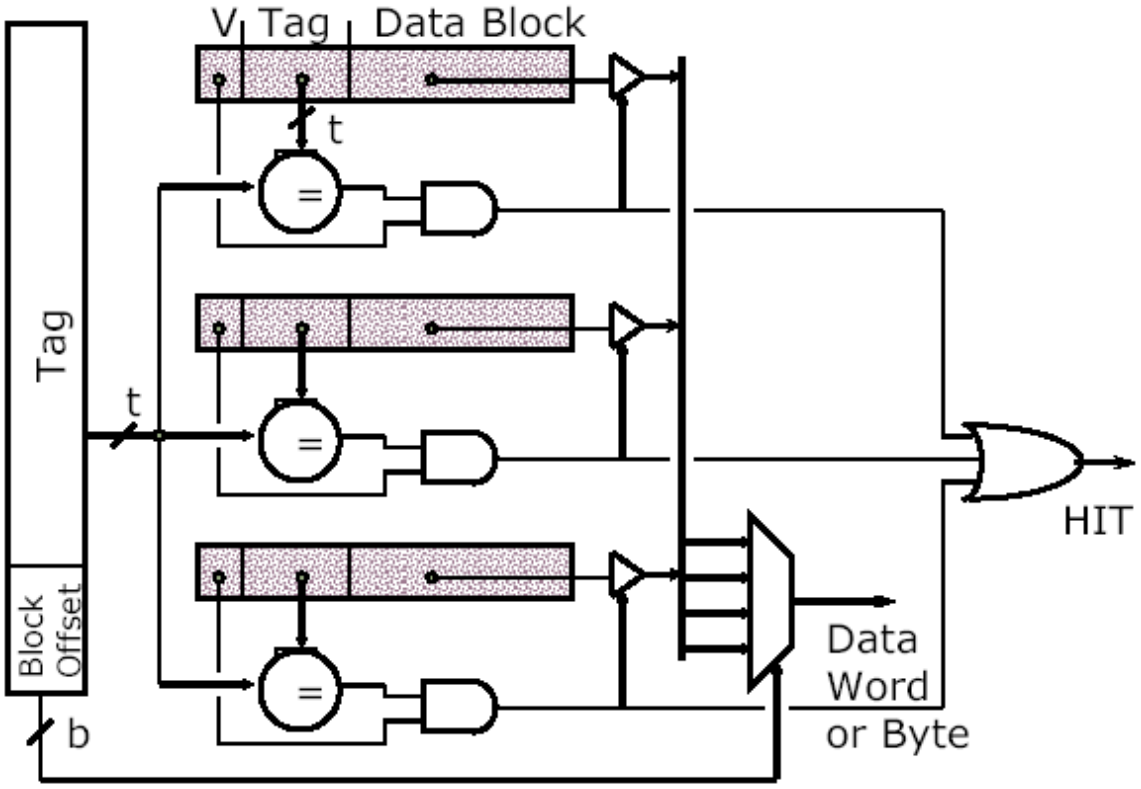
- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparatoare vs. 1
  - Extra MUX pentru selecția blocului din set - întârzie data
  - Datele apar după decizia Hit/Miss și selecția setului
  - În direct mapped cache, dpdv temporal, blocul din Cache este disponibil înainte de Hit/Miss (unde este logică suplimentară)

Q2: Cum se găsește un bloc în cache?

Cache total asociativ



Cache total asociativ, 8 blocuri



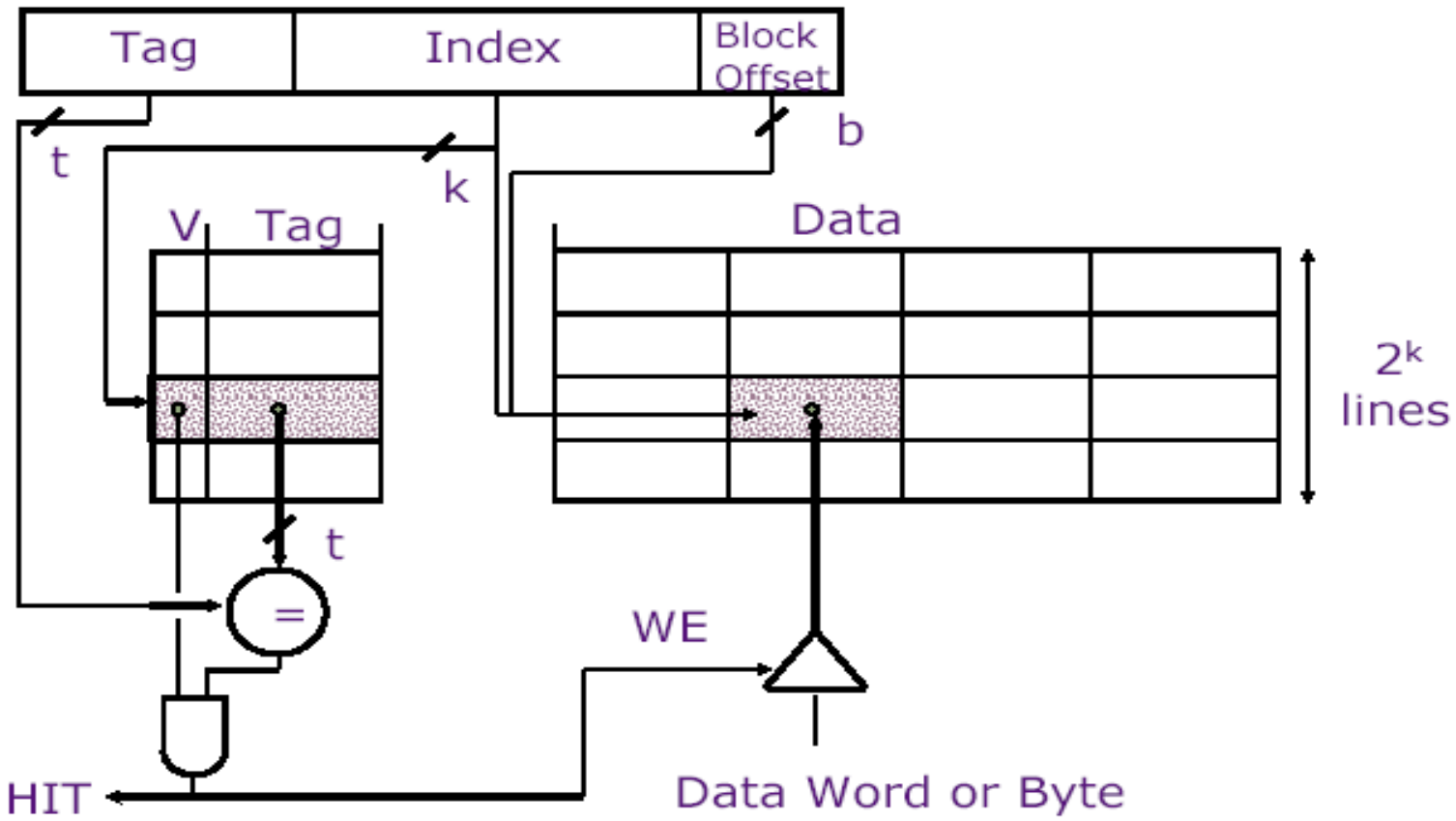
Fully associative cache (complet asociativă)

- Nu exista index - Cache Tag se compară în paralel cu toate celulele cache-ului.
- Necesită multe comparatoare.
- Se implementează cu o memorie asociativă - content addressable memory (CAM).
- Conflict Miss = 0 pentru cache total asociativ.

Q3: La un cache miss, care bloc e înlocuit?

- **Direct-mapped cache** – numai un loc de plasare, fără variante posibile
- **Asociative (set or full) cache** – este necesar un algoritm hardware de înlocuire:
  - Least recently used (LRU) – se înlocuiește blocul cel mai vechi (de la ultima folosire)
    - Ex. pentru un cache 2-way set associative, fiecare bloc are un bit USE.
    - Când se adresează un bloc, bitul lui de USE se setează pe 1, iar bitul celuilalt bloc se setează pe 0.
    - La o înlocuire se elimină blocul din set care are USE = 0.
    - Pentru asociativitate de ordin superior se folosesc abordări mai complexe
  - First-in-first-out (FIFO) - se înlocuiește blocul cel mai vechi (de la introducerea în cache)
    - FIFO se implementează cu un algoritm de tip round robin / bufer circular.
  - Least Frequently Used (LFU) – numărător pe fiecare bloc, incrementat la folosire
    - Se elimină blocul cu cele mai puține adresări
  - Random – blocurile de eliminat se aleg aleator
    - În simulare această metodă oferă performanțe doar puțin mai slabe ca cele de mai sus

Q4: Cum se realizează o scriere în cache?



Scriere în cache

- Modificarea blocului nu se începe până la confirmarea accesului ca hit.
- Mărimea scrierii: numai un număr de octeţi (1-8) specificat trebuie schimbat.

Q4: Cum se realizează o scriere în cache?

Metode de scriere în cache

- **Write-Through – WT**
  - Se înlocuiesc datele în cache și în memorie
  - Este lent – cere acces la memorie pentru scriere, în ambele cazuri.
  - Crește performanța WT cu write buffer – blocurile sunt memorate, așteaptă scrierea în memorie – procesorul poate lucra până la umplerea write buffer-ului
  - Avantaje: Read misses nu pot genera scrieri (vezi mai jos la WB), asigură coerența datelor
- **Write-Back – WB**
  - Blocul de date se scrie numai în cache. Memoria principală se actualizează (WB) numai când blocul respectiv se înlocuieşte în cache (Read miss)
  - Mai eficient decât WT, mai complex de implementat.
  - Folosind un Dirty bit per bloc se poate reduce traficul cu memoria (scrieri inutile).
- **Write Once** - Prima scriere ca write through; următoarele ca write back

Q4: Cum se realizează o scriere în cache?

Write miss: la un miss se aloca un bloc în cache?

- Write allocate: blocul este adus in cache, apoi urmează acţiunea de write hit.

- No write allocate: se scrie numai în memoria de nivel inferior.

- Combinatii uzuale:
  - Write Through și no write allocate; chiar și în cazul scrierilor succesive la un bloc, se scrie în memoria de nivel mai inferior
    - WT cu write buffers, astfel nu se introduc așteptări.
  - Write Back cu write allocate; se presupune că scrierile succesive la un bloc vor fi capturate de cache.

Q4: Cum se realizează o scriere în cache?

Write Buffer

- Conține: Dirty lines evacuate din WB cache, sau toate scrierile pentru WT cache
- Reduce penalizarea pentru Read Miss
  - Procesorul nu aşteaptă pentru scrieri si read miss-uri pot fi servite înainte de scrieri

- Implementat ca o memorie FIFO, conţine datele de scris în memorie
- Controller-ul de Memorie scrie din buffer în memorie si eliberează locația scrisa.
  - CPU va aștepta pentru o scriere numai dacă Write Buffer este plin

Memoria Cache - Performanță

Timpul mediu de acces la memorie:

Average memory access time AMAT = Hit time + Miss rate × Miss penalty

Influența asupra timpului de execuție pentru un program:

CPU time = (CPU execution clock cycles + Memory stall clock cycles) × Clock cycle time

CPU execution clock cycles – include ciclurile de ceas de execuție (ALU etc.) și accesarea memoriei în caz de hit la cache

Memory stall clock cycles – include penalizările adiționale la lucrul cu memoria (miss rate x miss penalty x număr mediu de accese la memorie per instrucțiune)

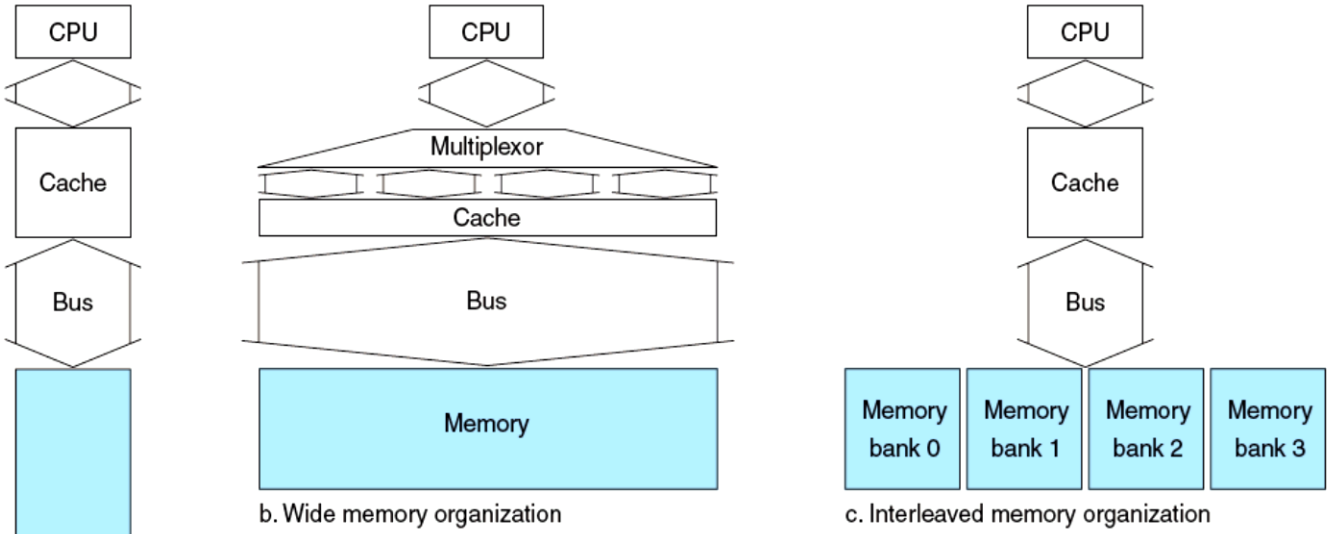
Cauze pentru „Cache Misses”

- **Cunoscut ca modelul celor 3 “C”**
  - Compulsory – apare implicit Miss la prima accesare a unui bloc, chiar dacă memoria cache este infinit de mare (numit și cold start / „pornire la rece”)
  - Capacity – memoria cache este prea mică pentru datele folosite de un program, apare Miss chiar dacă politica de înlocuire este perfectă
  - Conflict – Miss care apare datorită strategiei de plasare a blocurilor, nu apare la cache fully associative
- **Al patrulea “C”**: Coherence - Miss cauzat de lipsa de coerență a memoriei cache (Multiprocessors)

Design change	Effect on miss rate	Possible negative performance effect
Increase cache size	Decreases capacity misses	May increase access time
Increase associativity	Decreases miss rate due to conflict misses	May increase access time
Increase block size	Decreases miss rate due to spatial locality	Increases miss penalty. Very large block size can increase miss rate

Conexiuni Cache - Memorie

Tehnici de organizare a memoriei pentru creșterea lățimii benzii de transmisie a datelor



Exemplu

Presupunem o ierarhie de memorie caracterizată prin:

- Un bloc de cache = 4 cuvinte
- 1 ciclu pentru trimiterea adresei (1 ciclu de bus)
- 15 cicluri pentru acces memorie (pentru un bloc de memorie)
- 1 ciclu pentru transmiterea datelor (1 ciclu de bus)

Variante de conectare CPU – Cache – Memorie

Care variantă e mai performată dpdv al penalizării în caz de Miss ? Dar din punct de vedere al costurilor ?

Conexiuni Cache - Memorie

Penalizarea în caz de Miss:

- Cazul a) – memorie lățime 1 cuvânt
  - 1 + 4*15 + 4*1 = 65 cicluri
- Cazul b) – Memoria și magistrală de 4 cuvinte – cost crescut hardware
  - 1 + 1*15 +1*1 = 17 cicluri
- Cazul c) – Bank-uri de memorie intercalate – varianta folosită în sistemele actuale
  - 1 +1*15 + 4*1 = 20 cicluri

Evoluția memoriilor cache

Processor	Year	Frequency (MHz)	Level 1 Data Cache	Level 1 Instruction Cache	Level 2 Cache
80386	1985	12 – 40	none	none	None
80486	1989	16 – 150	8 KB unified		None on chip
Pentium	1993	60 – 100	8 KB	8 KB	None on chip
Pentium Pro	1995	150 – 200	8 KB	8 KB	256 KB – 1 MB
Pentium II	1997	233 – 450	16 KB	16 KB	256 KB – 512 KB
Pentium III	1999	450 – 1400	16 KB	16 KB	256 KB – 512 KB
Pentium 4	2001	1400 – 3730	8-16 KB	12 KB	256 KB – 2 MB
Pentium M	2003	900 – 2130	32 KB	32 KB	1 – 2 MB on chip
Core Duo	2005	1500 – 2160	32 KB / core	32 KB / core	2 MB shared on chip
Skylake (ex. Core I7)	2015	500-4000 (+)	32 KB / core	32 KB / core	256 KB / Core

Evoluția memoriei cache pentru familia de micro-procesoare Intel x86 (mai puțin Level 3 cache care a apărut mai recent)

Memoria Virtuală

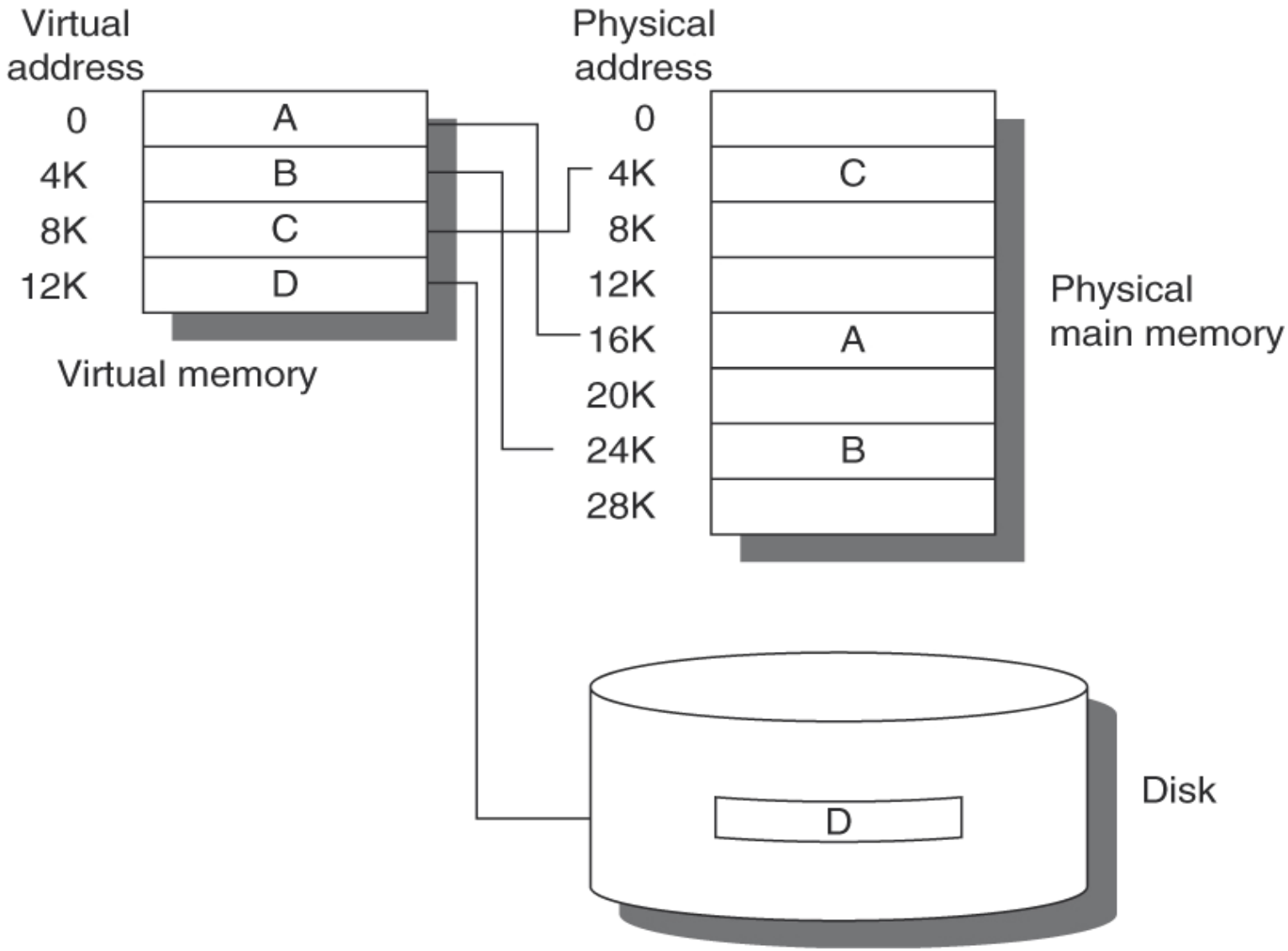
- Spațiul de adrese virtuale = spațiul adresabil de programe este determinat de ISA
- Memoria principală $\leq$ capacitatea disk-ului $\leq$ spațiul de adrese virtuale
- Memoria virtuală se organizează în blocuri de mărime fixă, numite pagini
- Memoria fizică de asemenea se consideră o colecție de pagini de aceleași dimensiuni
- Unitatea de date care se transferă între disk și memoria principală este o pagină
- Memoria Virtuală – avantaje:
  - Iluzie de memorie fizică mare
  - Relocarea programelor
  - Protecție

Memoria Virtuală

- **Memoria principală** – rol de cache pentru memorie secundară, disc magnetic
- **Pagini**: blocuri pentru memoria virtuală

- **Page faults**: data nu este în memoria principală → obligatoriu acces la disk pentru aducere
  - Miss penalty este f. mare / pagini relativ mari (ex. 4KB)
  - Reducerea page fault este importantă (merită politica LRU)
  - Page faults pot fi tratate în software (sistemul de operare) în loc de hardware
    - Timp suplimentar în software mic față de timpul acces disk
  - Write-through prea scump, folosim write-back

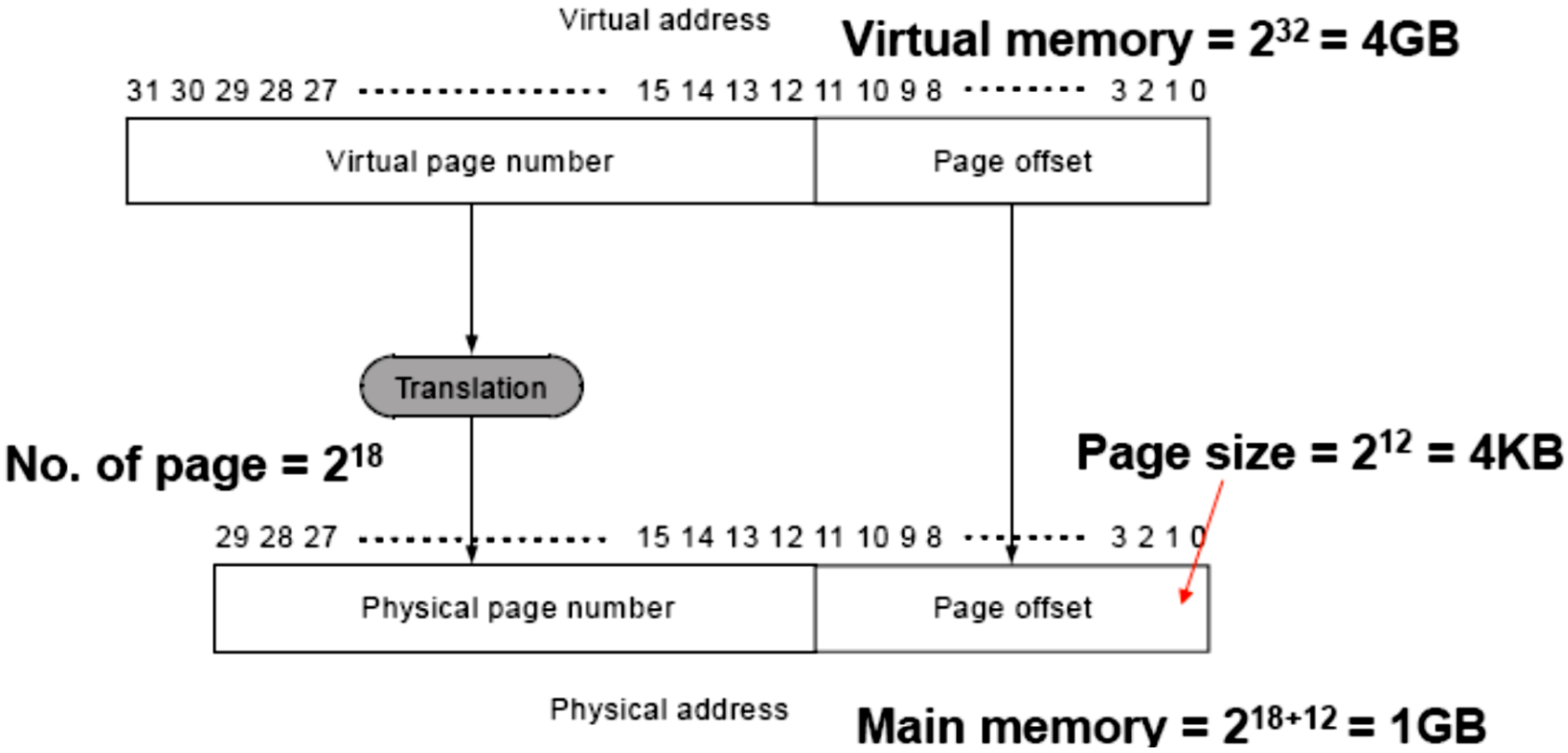
Memoria Virtuală



Program logic – spațiu continuu de adrese virtuale – 4 pagini, A, B, C, si D.

Memoria Virtuală

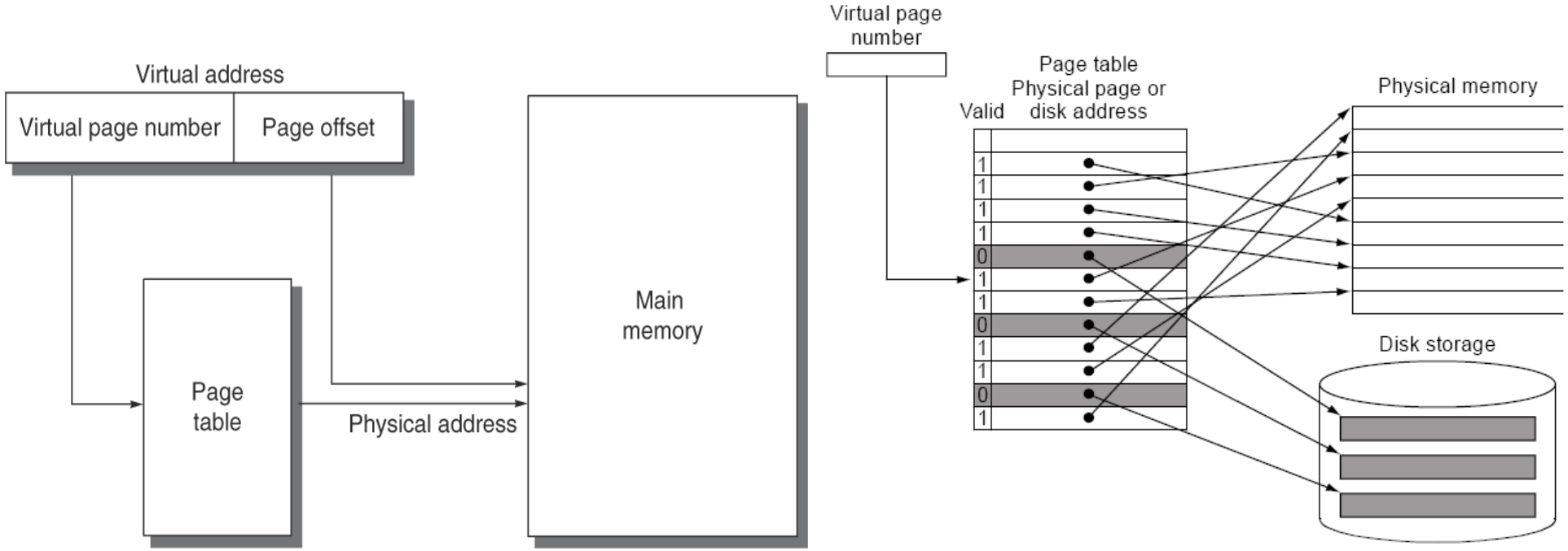
Transformare adrese virtuale → adrese fizice, exemplu:



- Numărul de biţi în offsetul de pagină determină mărimea paginii
- De regulă, numărul de pagini virtuale > numărul paginilor fizice
- Necesită un tabel de translatare, numit tabel de pagini (page table)

Memoria Virtuală

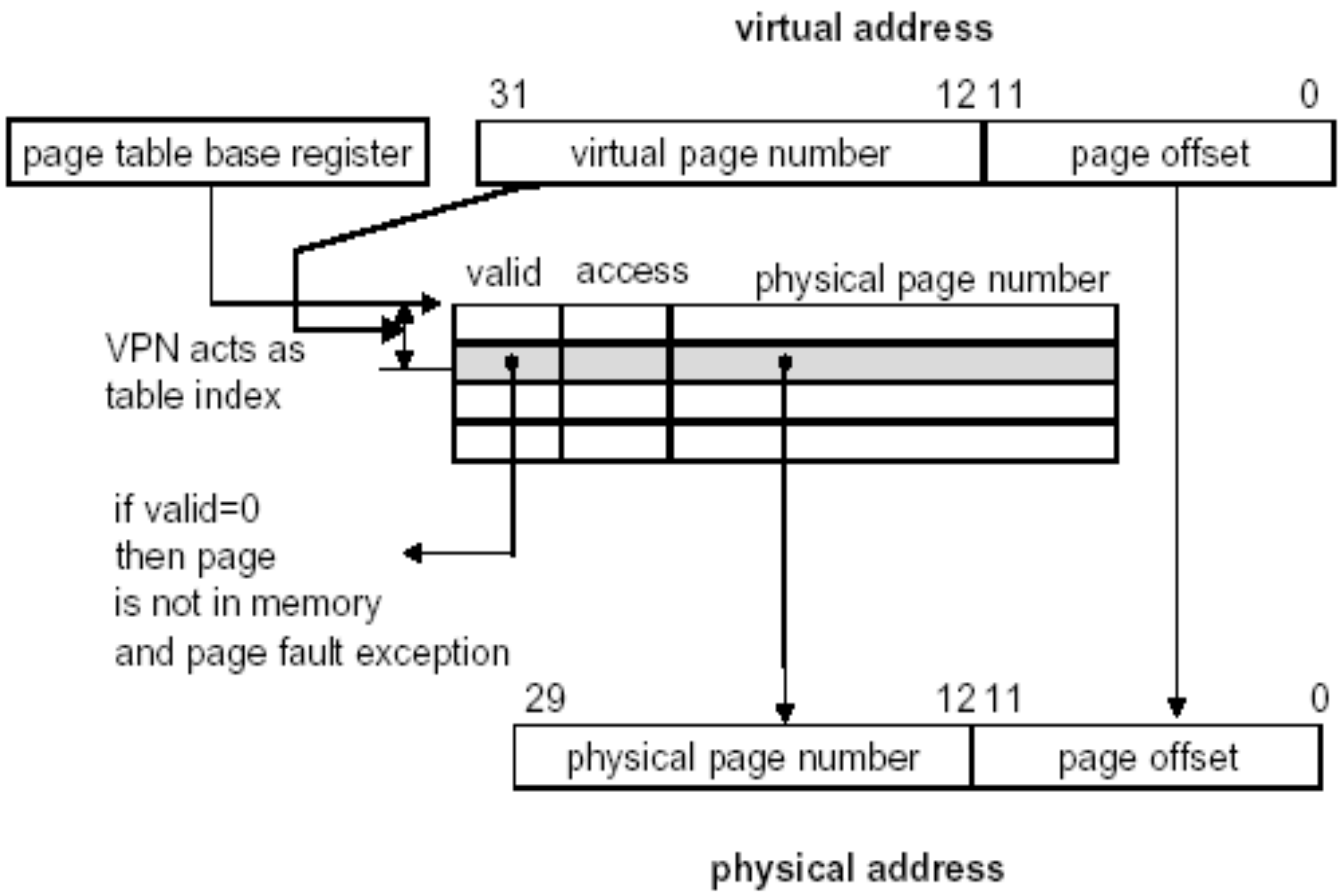
Cum se plasează și cum se găsește o pagină?



Tabel de pagini - mapează pagina virtuală la pagina fizică / pagina de disk

- Paginile se localizează prin tabele indexate: tabele de pagini
- Fiecare program are tabel propriu de pagini
- Un registru (page table register) indică începutul tabelului de pagini
- Tabelul de pagini implementează translatarea: adresa virtuală → adresa fizică

Memoria Virtuală



Exemplu:

- adresa virtuală 32 biti - 4 GB memoria virtuala
- pagină de 4 KB,
- Adresa fizica 30 biţi -1 GB memoria fizica;
- Virtual Page Number=20 biţi,
- Physical Page Number= 18 biţi

Translatarea adresei cu tabel de pagini

Pentru a evita tabele mari de pagini:

- Fiecare program are tabel de pagini propriu
- Registrul de pagină indică începutul tabelei de pagină a programului
- Alte tehnici, ex. tabele de pagini pe mai multe nivele, *hashing virtual address, etc.*

Memoria Virtuală

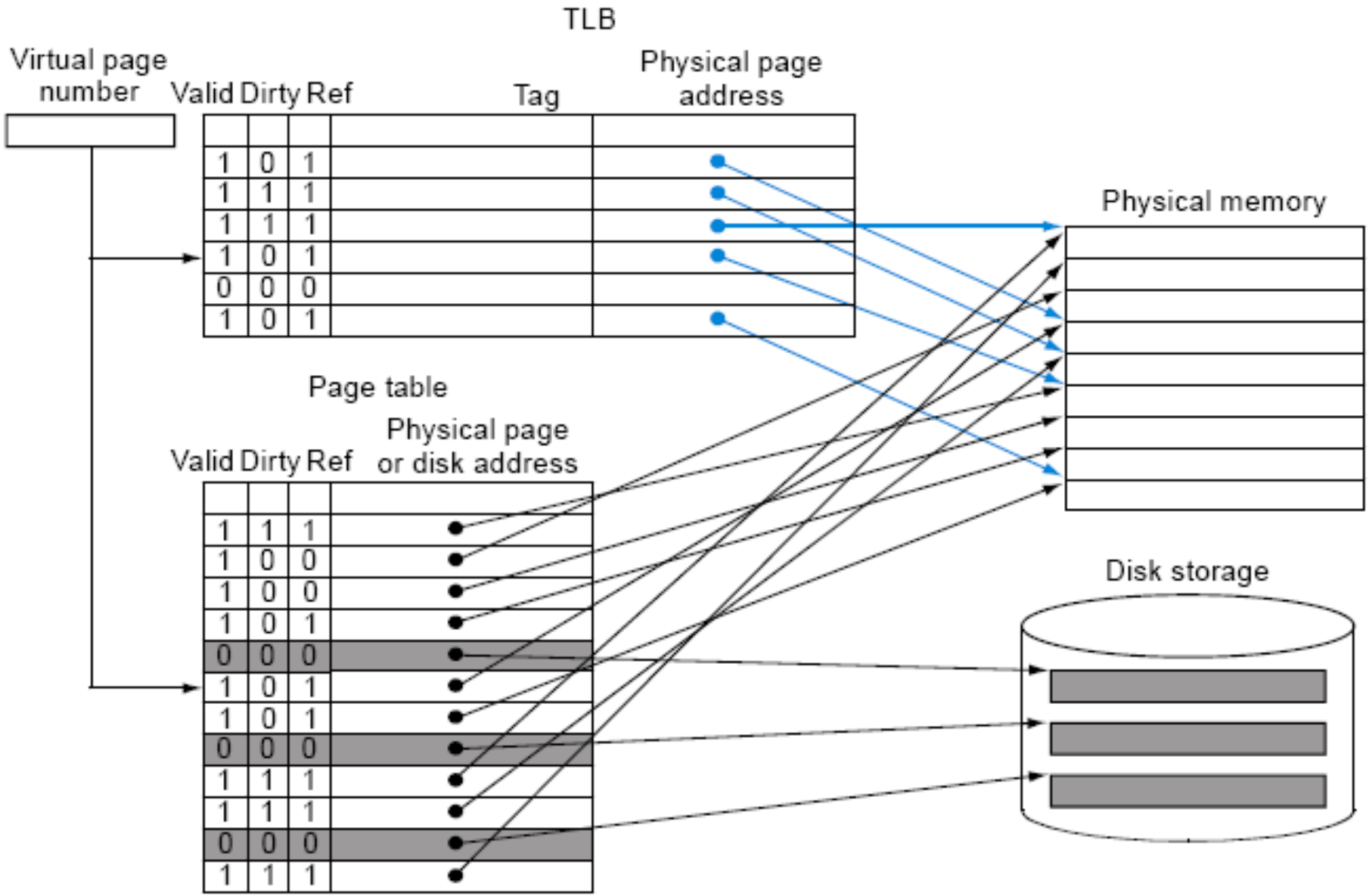
Unde în memoria principală se plasează paginile aduse din memoria secundară?

- În eventuale pagini ne-ocupate
- Dacă toate paginile sunt ocupate, se alege o pagină pentru înlocuire
- Politica LRU de înlocuire (least recently used)
- Paginile înlocuite sunt scrise înapoi pe disc (în spațiul dedicat – swap space).

Se poate optimiza partea de translatare a adreselor: TLB (Translation Lookaside Buffer)

- Mecanismul de translatare a adreselor este încet
  - Necesită cicluri suplimentare pentru a accesa tabelul de pagini din memoria principală și pentru a obține adresa fizică
- TLB – memorie cache pentru intrări în tabelul de pagini care au fost folosite recent

Memoria Virtuală



Translatare cu TLB – un cache total asociativ

Probleme

Un procesor generează adrese pe 32 de biți pentru o memorie adresabilă pe octet. Proiectați o memorie cache de 8 KB (strict pentru biții de date, nu include biții de etichetă/tag). Dimensiunea unui bloc din cache este de 32 de octeți. Se va face schema memoriei cache, și se va explica mecanismul de decodificare a adresei pentru:

- Cache de tip direct mapped
- Cache de tip 4-way associative.

Referințe

[1] Computer Architecture - A Quantitative Approach, 4th edition J.L. Hennessy, D.A. Patterson Elsevier, 2007