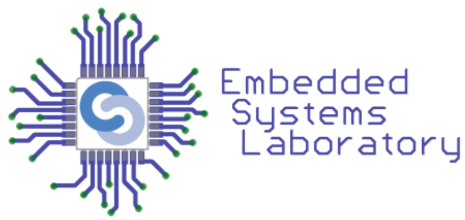


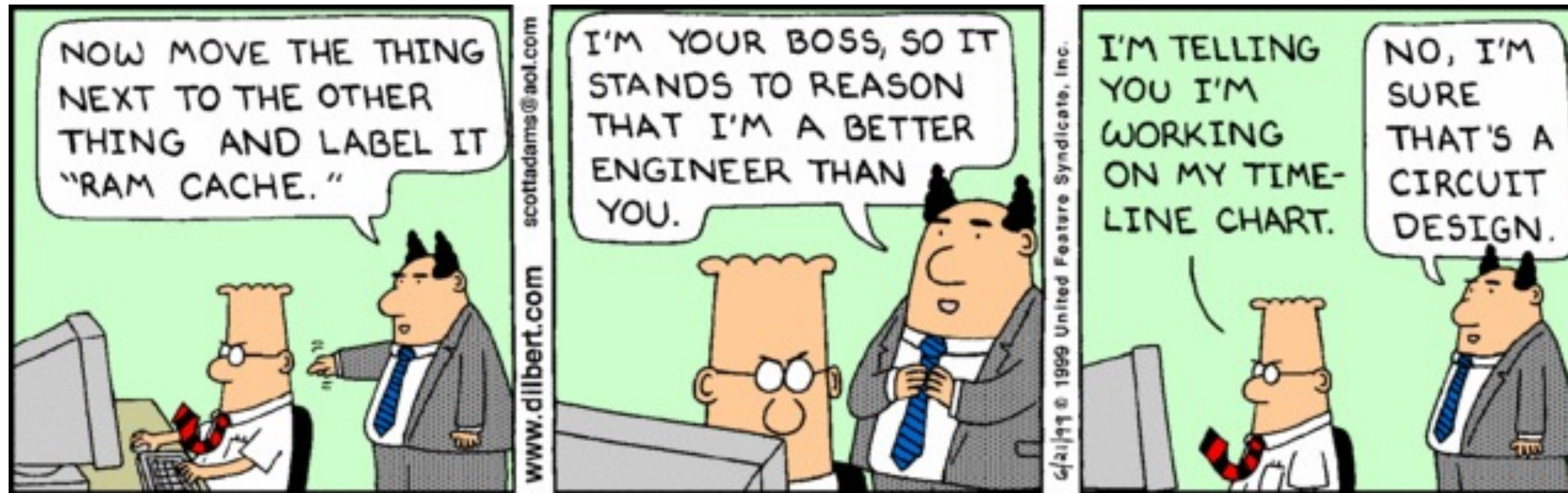
Calculatoare Numerice (2)

- Cursul 2 -

Memoria Cache

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

Comic of the Day



http://dilbert.com/strips/comic/1999-06-21/

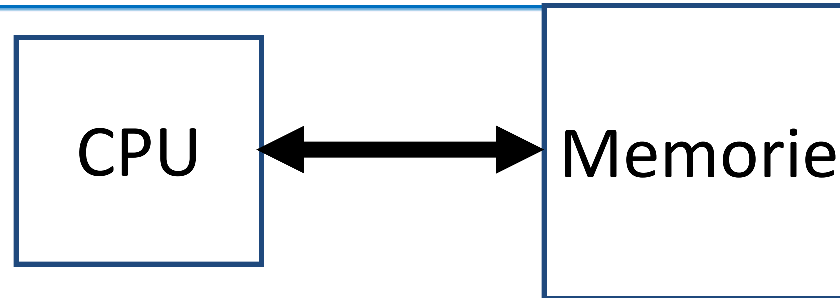
Organizarea memoriei cache

Viteza procesoarelor crește mai rapid decât cea a memoriilor

- Diferența de viteză dintre procesor și memorie crește

Cuprins
Necesitatea unei memorii cache
Cum funcționează un cache?
Cache mapat direct
Cache mapat set-asociativ
Memoria cache și memoria principală
Îmbunătățirea performanței memoriei cache

CPU-Memory Bottleneck



Performanța calculatoarelor este de obicei limitată de lățimea de bandă a memoriei și de latență

- Latență (timpul necesar pentru un singur acces)
  - Memory access time >> Processor cycle time
- Lățime de bandă (numărul de accese pe unitatea de timp)

  Daca un procent m% instrucțiuni dintr-un program accesează memoria:

  -> 100% + m referiri la memorie/instrucțiune

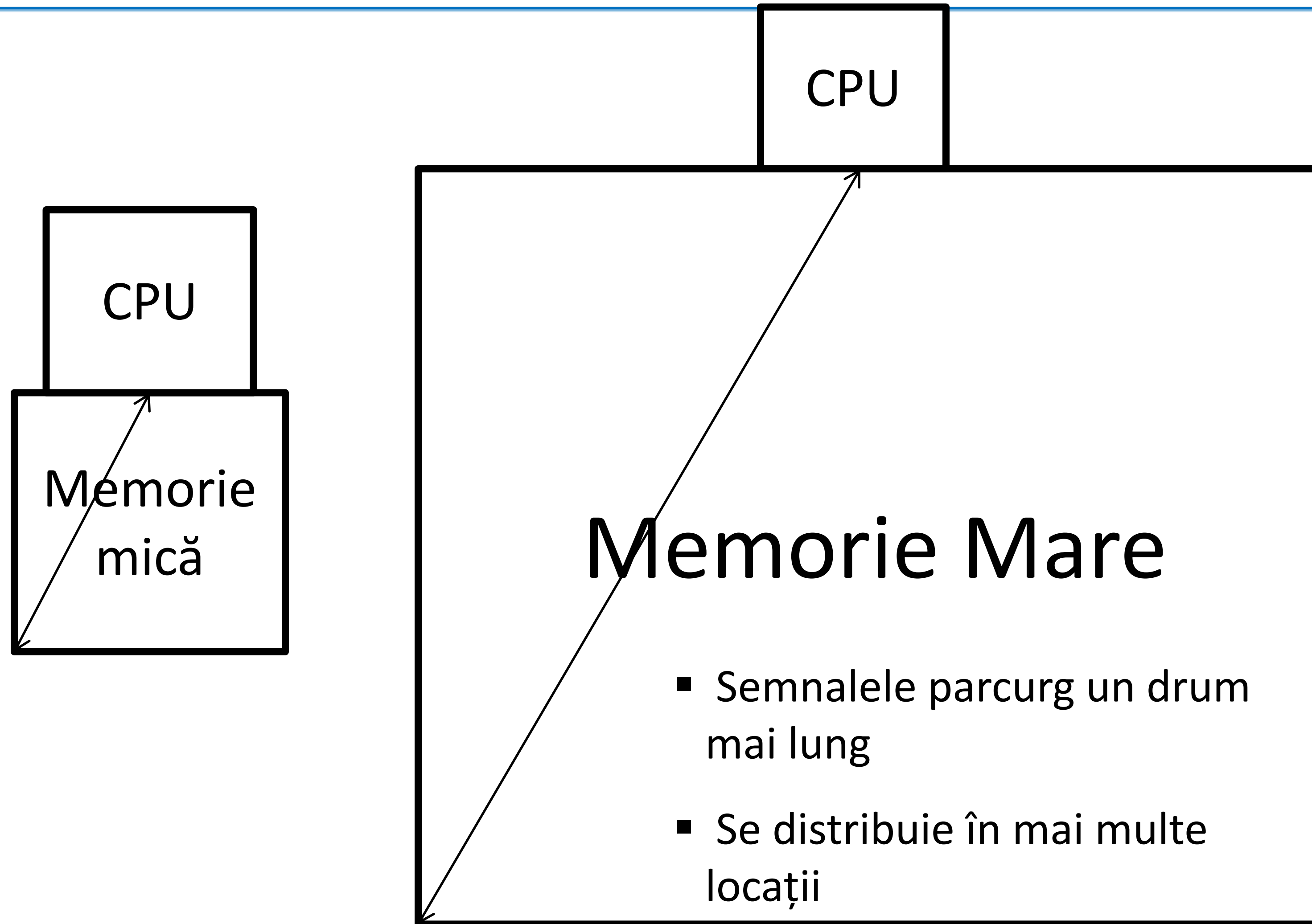
  -> CPI = 1 necesită 100% + m referințe la memorie/ciclu (p.p. o arhitectură RISC)

Diferența dintre procesor și DRAM (latență)

µProc 60%/year
1000
100
10
1
Performanță
CPU
Processor-Memory
Performance Gap:
(growing 50%/yr)
DRAM
7%/year
DRAM
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
Timp

Procesor la 3GHz superscalar accesează în 100ns memoria DRAM sau poate să execute 1,200 instrucțiuni în timpul unui singur acces la memorie!

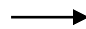
Dimensiunea fizică afectează latența

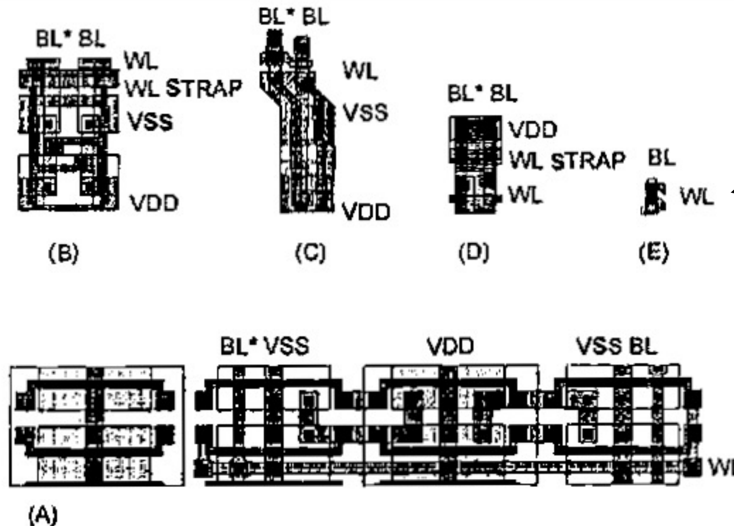


- Semnalele parcurg un drum mai lung
- Se distribuie în mai multe locații

Dimensiunile relative ale celulelor de memorie

On-Chip SRAM in logic chip





1 Memory cell in 0.5µm processes
a) Gate Array SRAM
b) Embedded SRAM
c) Standard SRAM (6T cell with local interconnect)
d) ASIC DRAM
e) Standard DRAM (stacked cell)

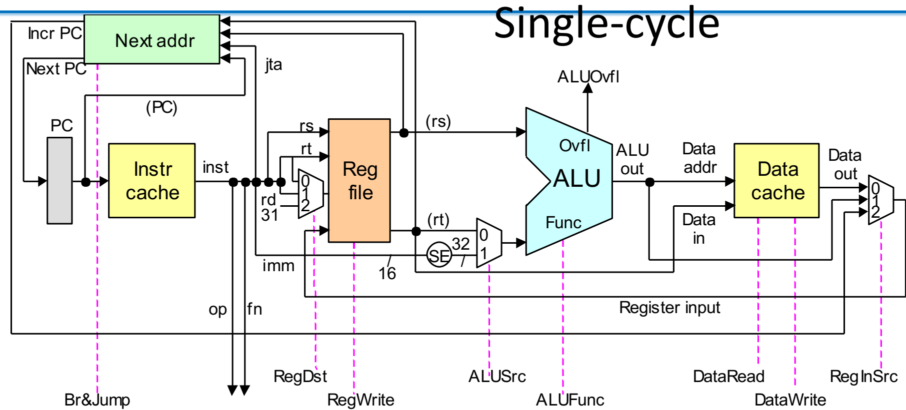
DRAM on memory chip

[Foss, "Implementing Application-Specific Memory", ISSCC 1996]

Memory	Process	Cell size (μm^2)	Cell efficiency	Bits in 100mm²(10^3)	Gate size (μm^2)	Gate utilization	Gates in 100mm² (10^3)
Gate array SRAM	3-metal ASIC	370	80%	216	185	70%	378
Embedded SRAM	3-metal ASIC	67	70%	1045	185	70%	378
Standard SRAM	2-metal 6T local int.	43	65%	1512	245	40%	163
Embedded ASIC-DRAM	3-metal ASIC	23	60%	2609	185	70%	378
Standard DRAM	2-metal stacked cell	3.2	50%	15625	411	40%	97

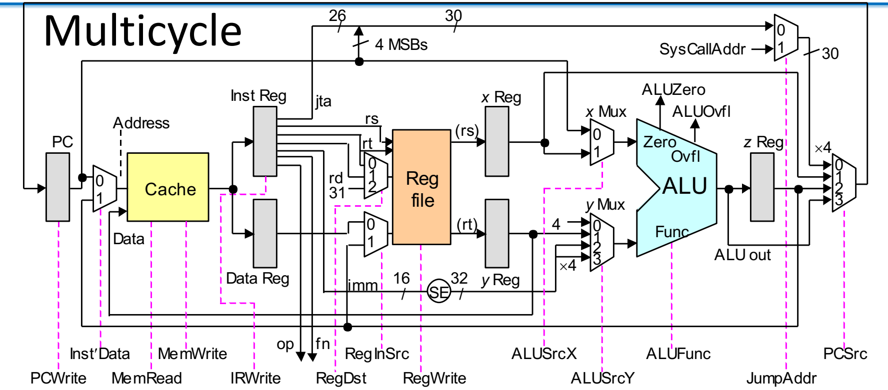
Table 1: Memory and logic density for a variety of 0.5µm implementations.

Necesitatea memoriei cache



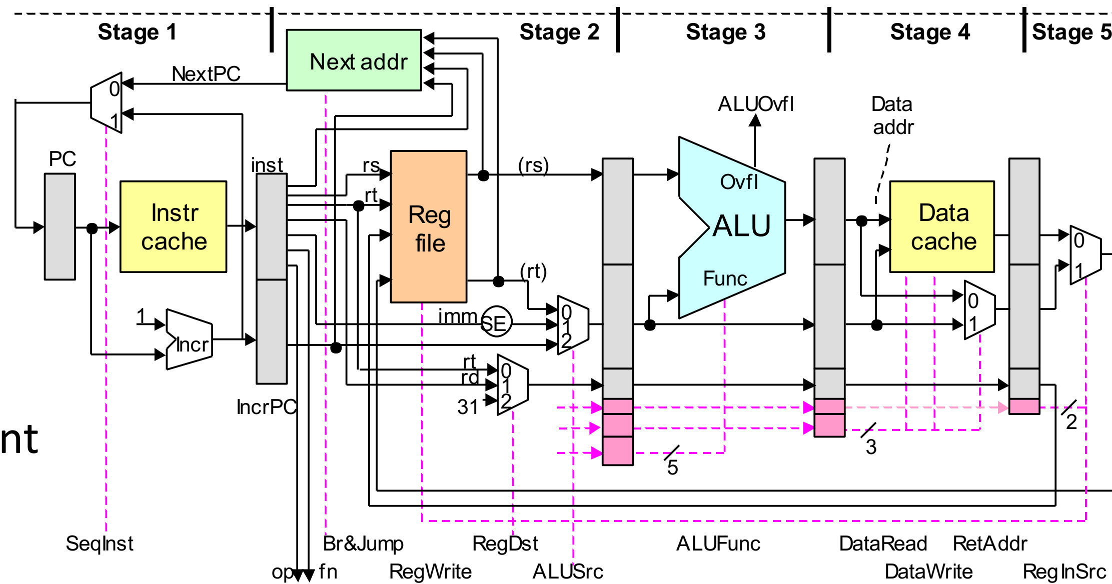
125 MHz

CPI = 1



500 MHz

CPI ≅ 4



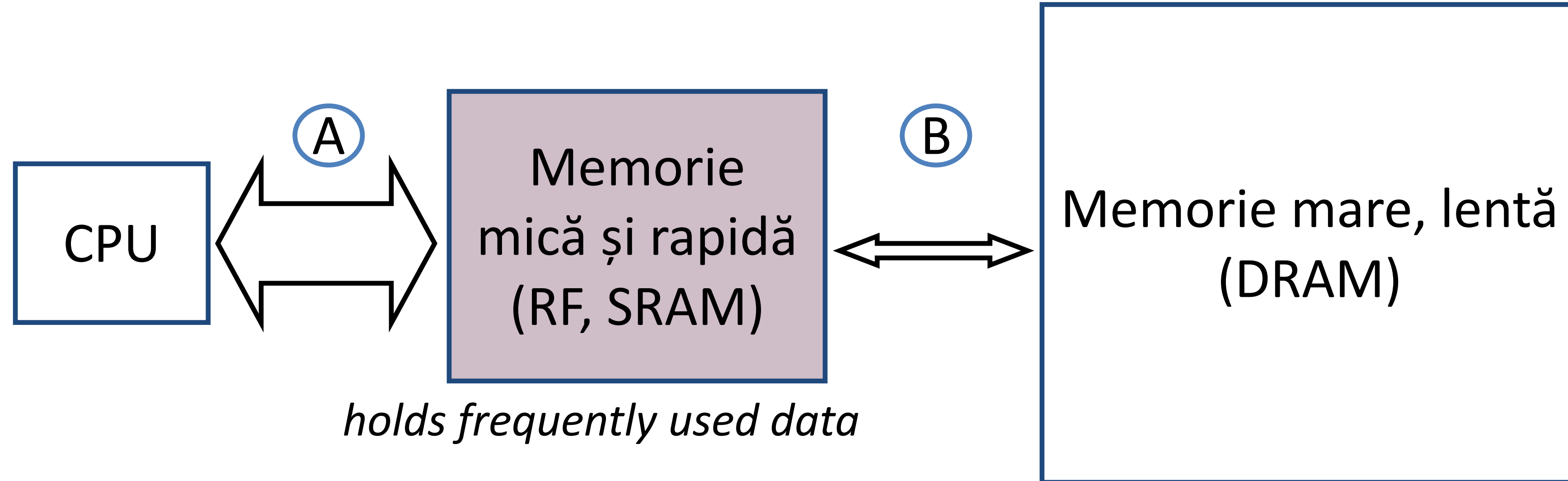
Pipelined

500 MHz

CPI ≅ 1.1

Toate cele 3 implementări presupun timpi de acces de 2-ns pentru date și instrucțiuni; memoriile RAM tipice sunt de 10-50x mai lente

Organizarea ierarhică a memoriilor



- *capacitate*: Registre << SRAM << DRAM
- *latență*: Registre << SRAM << DRAM
- *lățime de bandă:* on-chip >> off-chip

Pentru un acces de date:

if data $\in$ fast memory $\Rightarrow$ low latency access *(SRAM)*

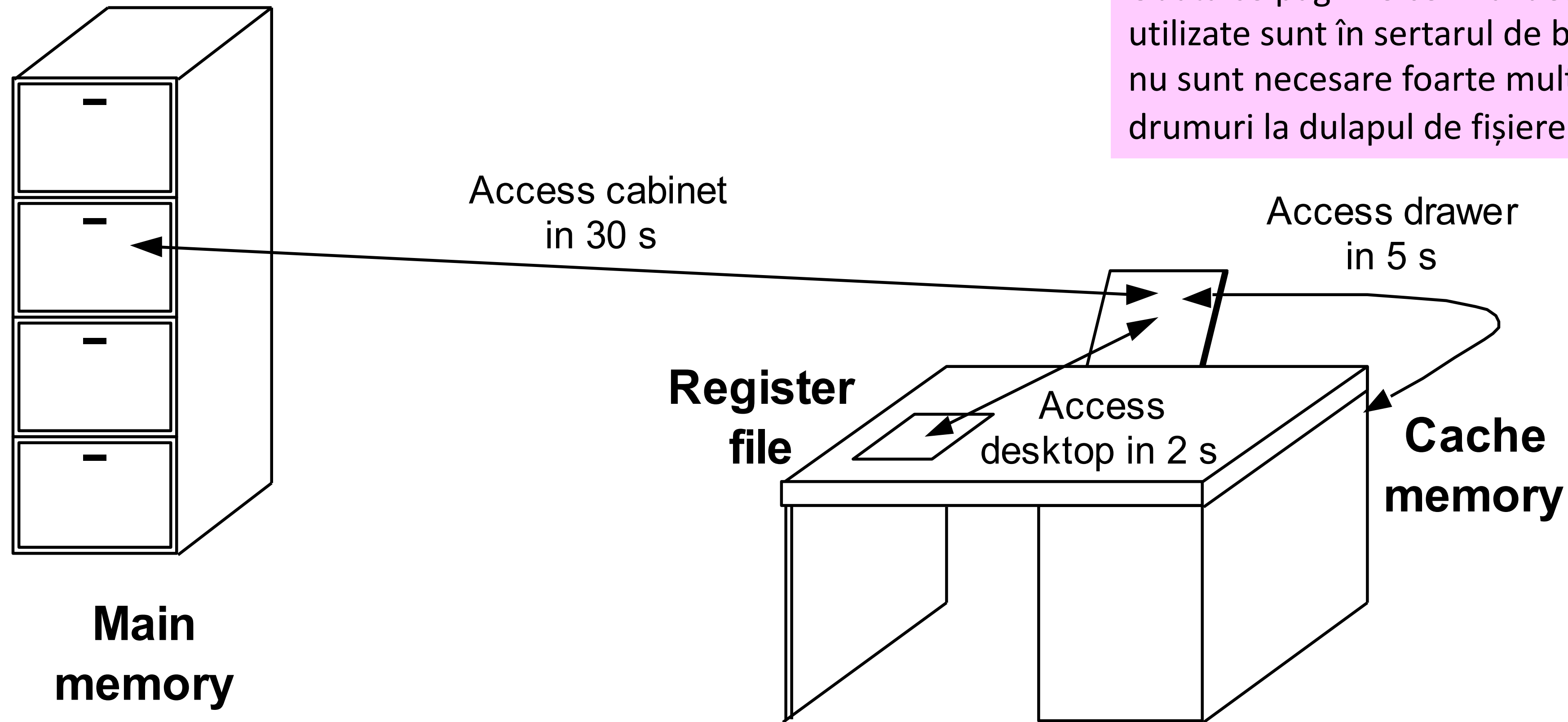
if data $\notin$ fast memory $\Rightarrow$ high latency access *(DRAM)*

Managementul ierarhiei de memorii

- Stocare rapidă/mică, de ex. registre
  - Adresa este specificată de obicei în corpul instrucțiunii
  - Implementată direct ca o tabelă de registre
    - *Dar hardware-ul poate să facă operații transparente software-ului, de ex. managementul stivei, redenumirea registrelor etc.*
- Stocare de mari dimensiuni/lentă, de ex. memoria principală
  - Adresa calculată de obicei din valorile stocate în registre
  - Implementată de obicei ca o ierarhie de memorii (cache-mem. princ.) în care hardware-ul decide ce date sunt aduse și ținute în memoria rapidă
    - *Software-ul poate să dea anumite indicii, de ex. nu face prefetch sau nu stoca în cache anumite date*

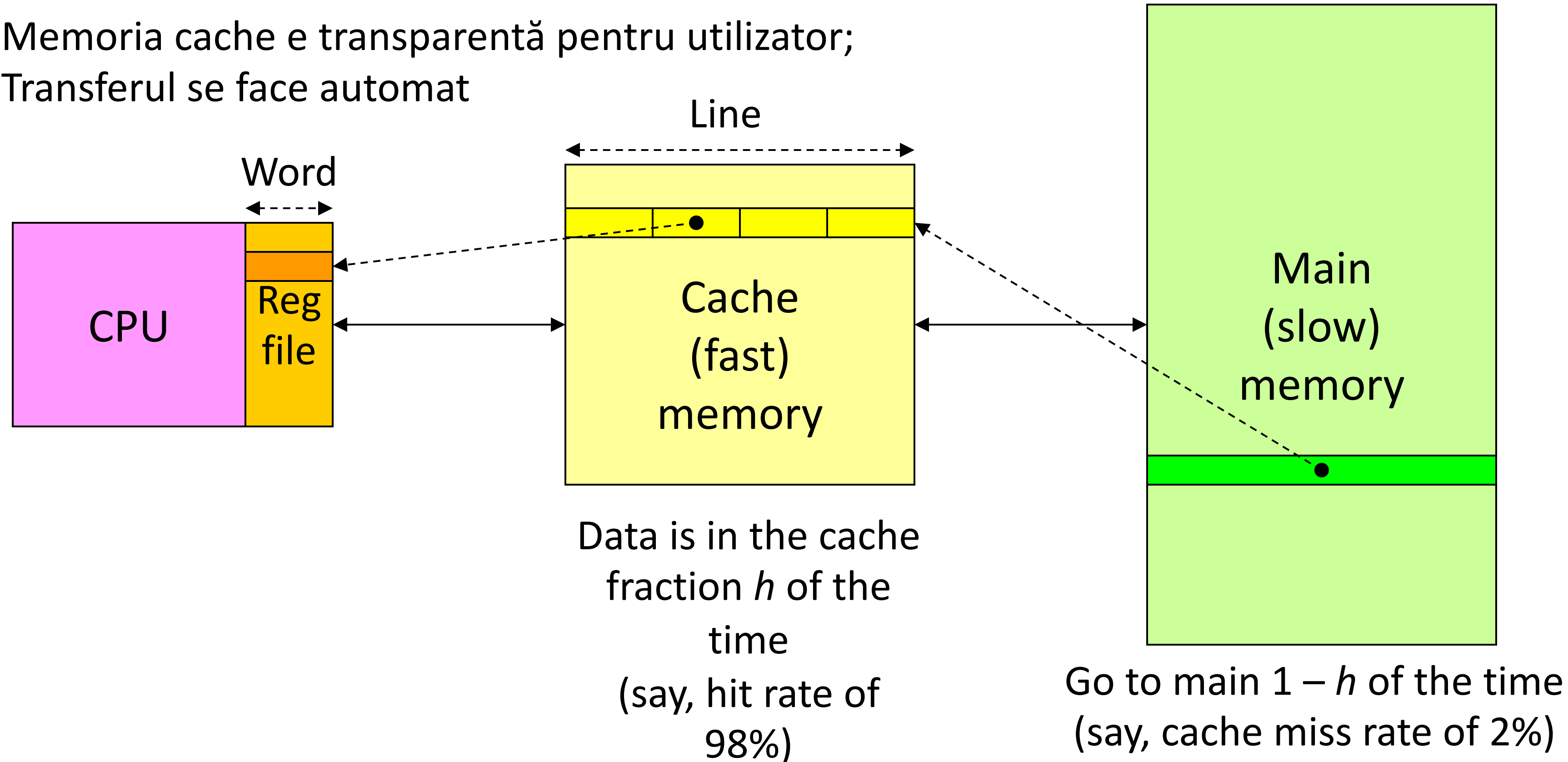
Analogia dulap de fișiere – sertar de birou

Odată ce paginile cel mai des utilizate sunt în sertarul de birou, nu sunt necesare foarte multe drumuri la dulapul de fișiere.



Foile de pe birou (registre) sau dintr-un sertar (cache) sunt mult mai ușor de accesat decât cele dintr-un dulap de fișiere (memoria principală).

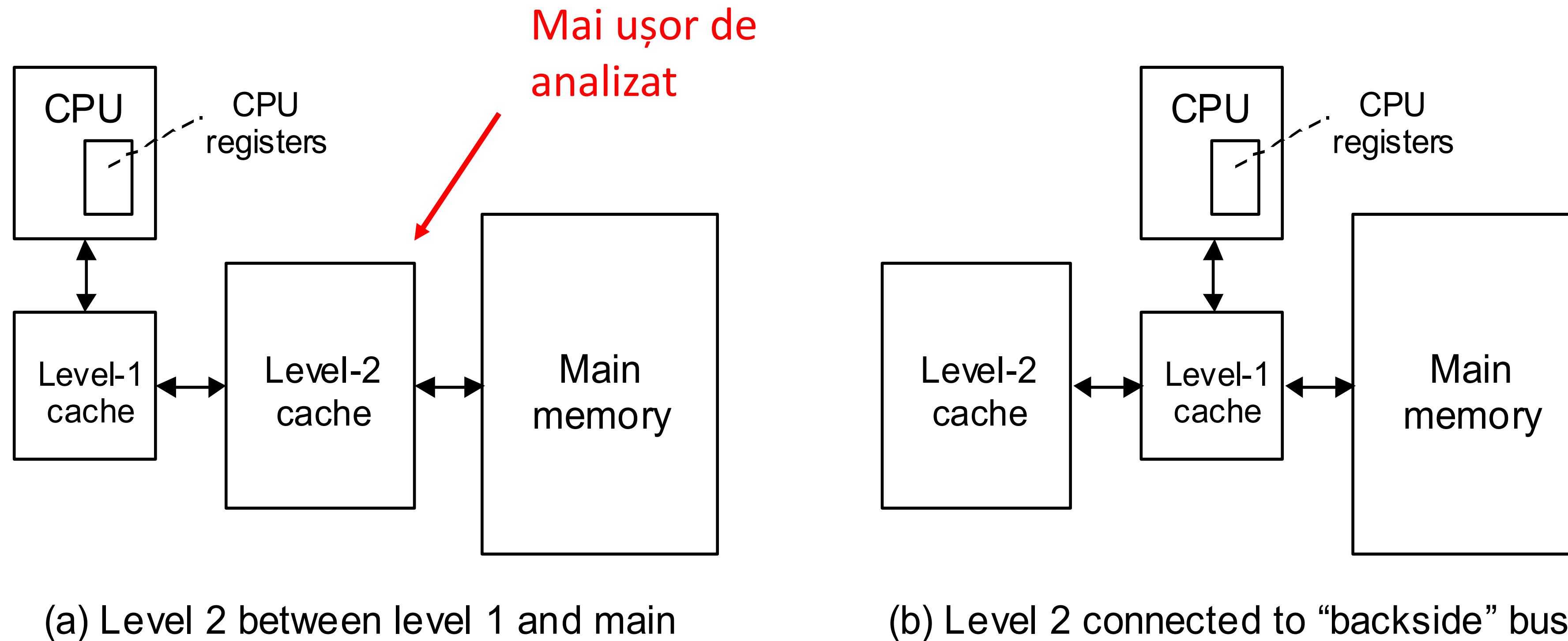
Cache, rata Hit/Miss și timpul efectiv de acces



Un nivel cache cu rata de hit h

$$C_{eff} = hC_{fast} + (1 - h)(C_{slow} + C_{fast}) = C_{fast} + (1 - h)C_{slow}$$

Niveluri multiple de cache



Memoriile cache funcționează ca un buffer între procesorul ultra-rapid și memoria principală care este mult mai lentă.

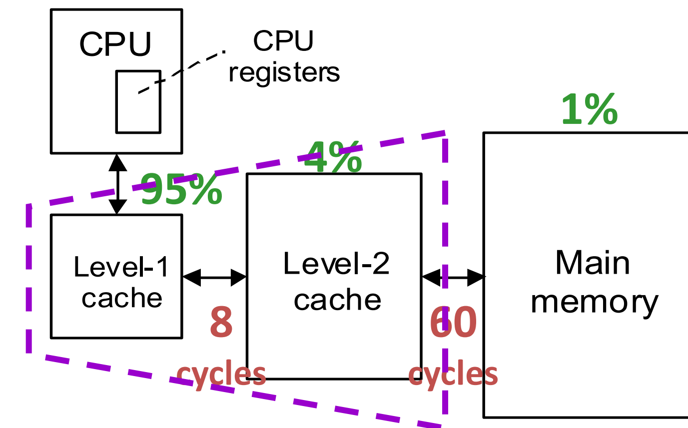
Performanțele unui sistem cu cache pe două niveluri

Un sistem cu cache L1 și L2 are un CPI de 1.2 fără cache miss. Pentru fiecare instrucțiune sunt, în medie p=1.1 accese la memorie.

Care este CPI efectiv dacă luăm în calcul și rata cache miss?

Care este rata efectivă de hit și penalizarea pentru miss dacă cache-ul L1 și L2 sunt modelate ca un singur cache?

Level	Local hit rate	Miss penalty
L1	95 %	8 cycles
L2	80 %	60 cycles



Soluție

$C_{eff} = C_{fast} + p(1 - h_1)[C_{medium} + (1 - h_2)C_{slow}]$

Deoarece C_{fast} e inclus în CPI de 1.2, trebuie să calculăm pentru restul

$CPI = 1.2 + 1.1(1 - 0.95)[8 + (1 - 0.8)60] = 1.2 + 1.1 \times 0.05 \times 20 = 2.3$

Global: hit rate 99% (95% + 80% of 5%), miss penalty 60 cicli

Parametrii memoriei cache

Cache size (în octeți sau cuvinte). Un cache mai mare poate să țină mai multe date, dar este mai mare și mai lent.

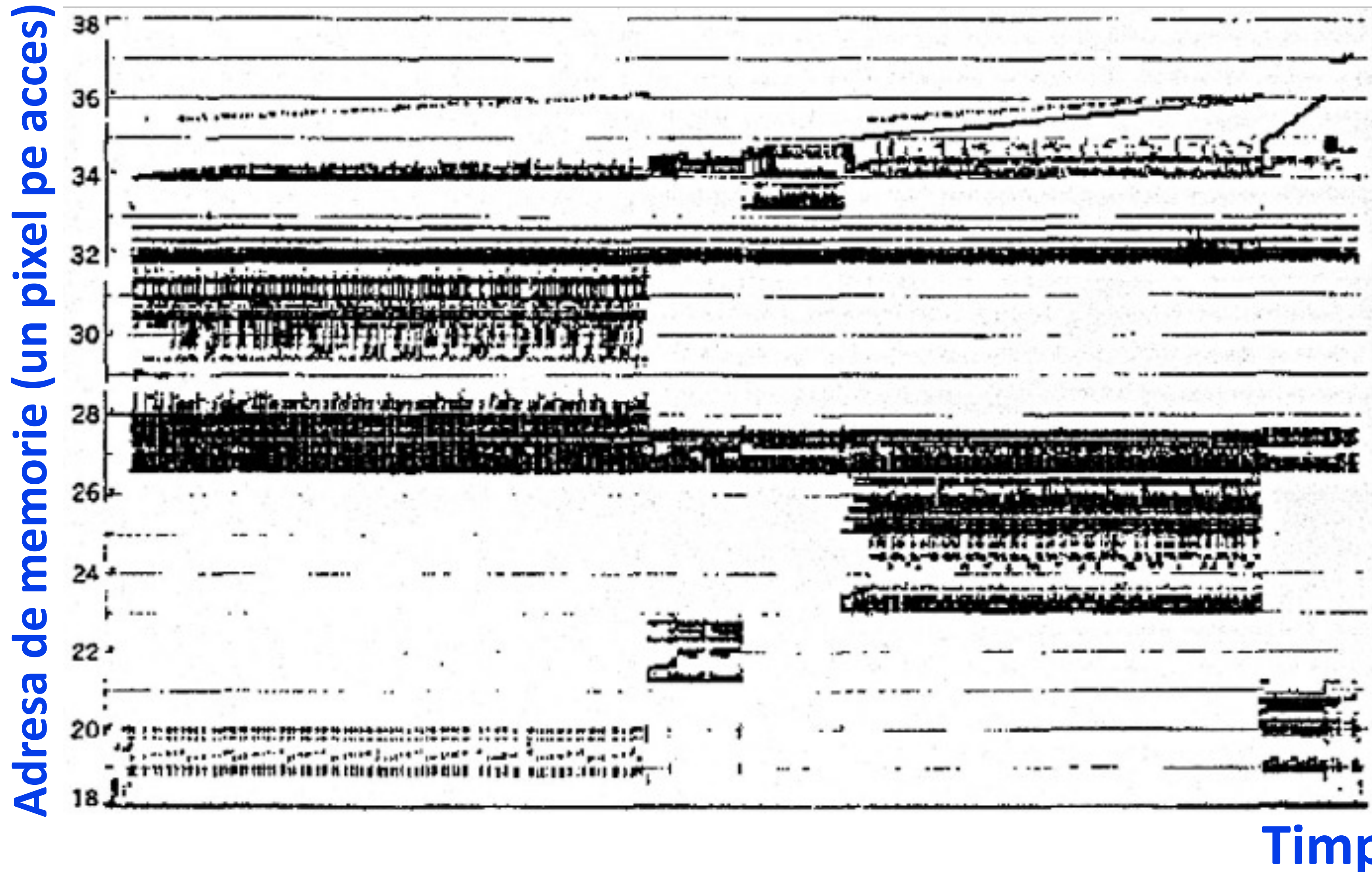
Block or cache-line size (unitatea de transfer de date dintre cache și mem. Principală). O linie mai mare pentru cache inseamna mai multe date aduse în cache la fiecare miss.Poate să îmbunătățească rata de hit dar poate să aducă și date mai puțin utile în cache.

Placement policy. Determinarea locului unde o linie cache este plasată. Politicile mai flexibile implică un cost hardware mai mare și pot sau nu să aducă performanțe mărite (datorate complexității mărite a localizării).

Replacement policy. Determinarea blocului din cache ales pentru suprascriere (în care plasăm o linie nouă din cache). Abordări tipice: alegerea unui bloc aleator sau Least Recently-Used (LRU).

Write policy. Determină dacă actualizările în cache sunt transmise imediat memoriei principale (*write-through*) sau blocurile modificate sunt copiate înapoi în memoria principală atunci când trebuie copiate (*write-back* sau *copy-back*).

Graficul referințelor la memorie



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Șabloane tipice de referință la memorie

Adresă

n loop iterations

Instruction fetches

subroutine call

subroutine return

Acces la stivă

argument access

vector access

scalar accesses

Acces de date

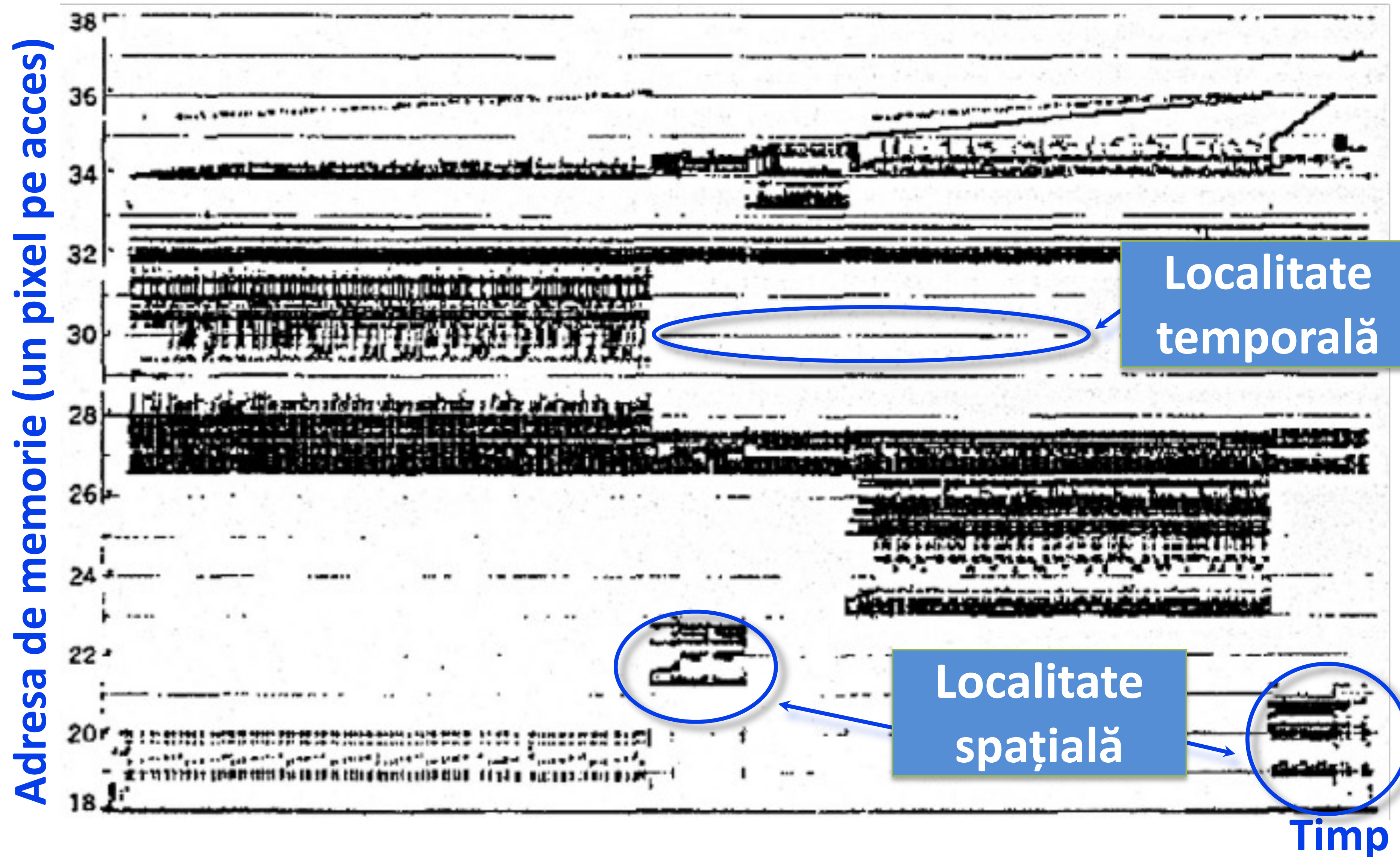
Timp

Două proprietăți previzibile ale referințelor la memorie

- **Localitate temporală**: Dacă o locație de memorie este accesată, este foarte probabil ca aceeași locație de memorie să fie accesată și în viitorul apropiat.

- **Localitate spațială**: Dacă o locație de memorie este accesată, este foarte probabil ca programul să acceseze și locațiile din imediata vecinătate în viitorul apropiat.

Modele și corelații pentru accesul la memorie

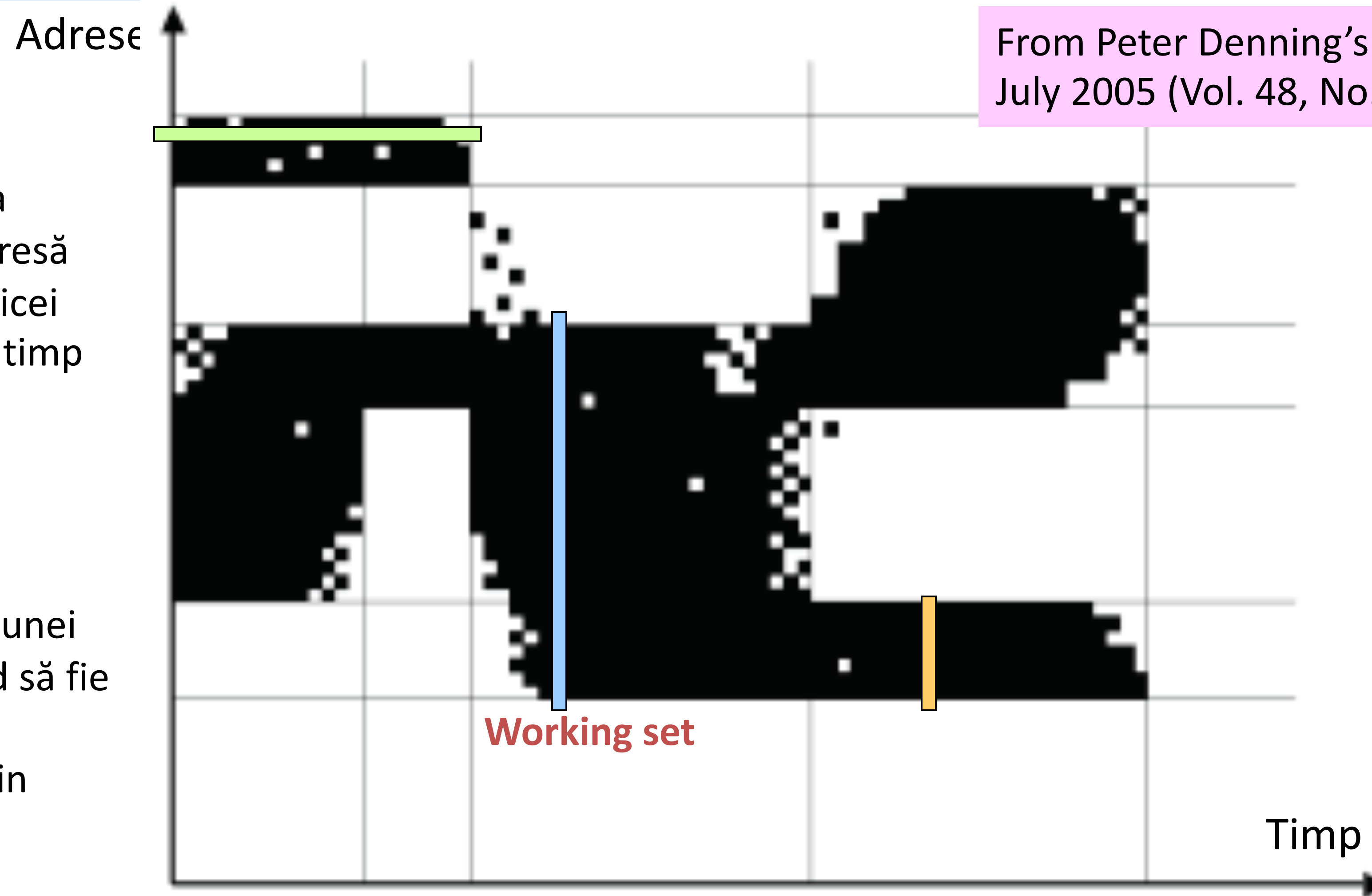


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Localitatea temporală și spațială

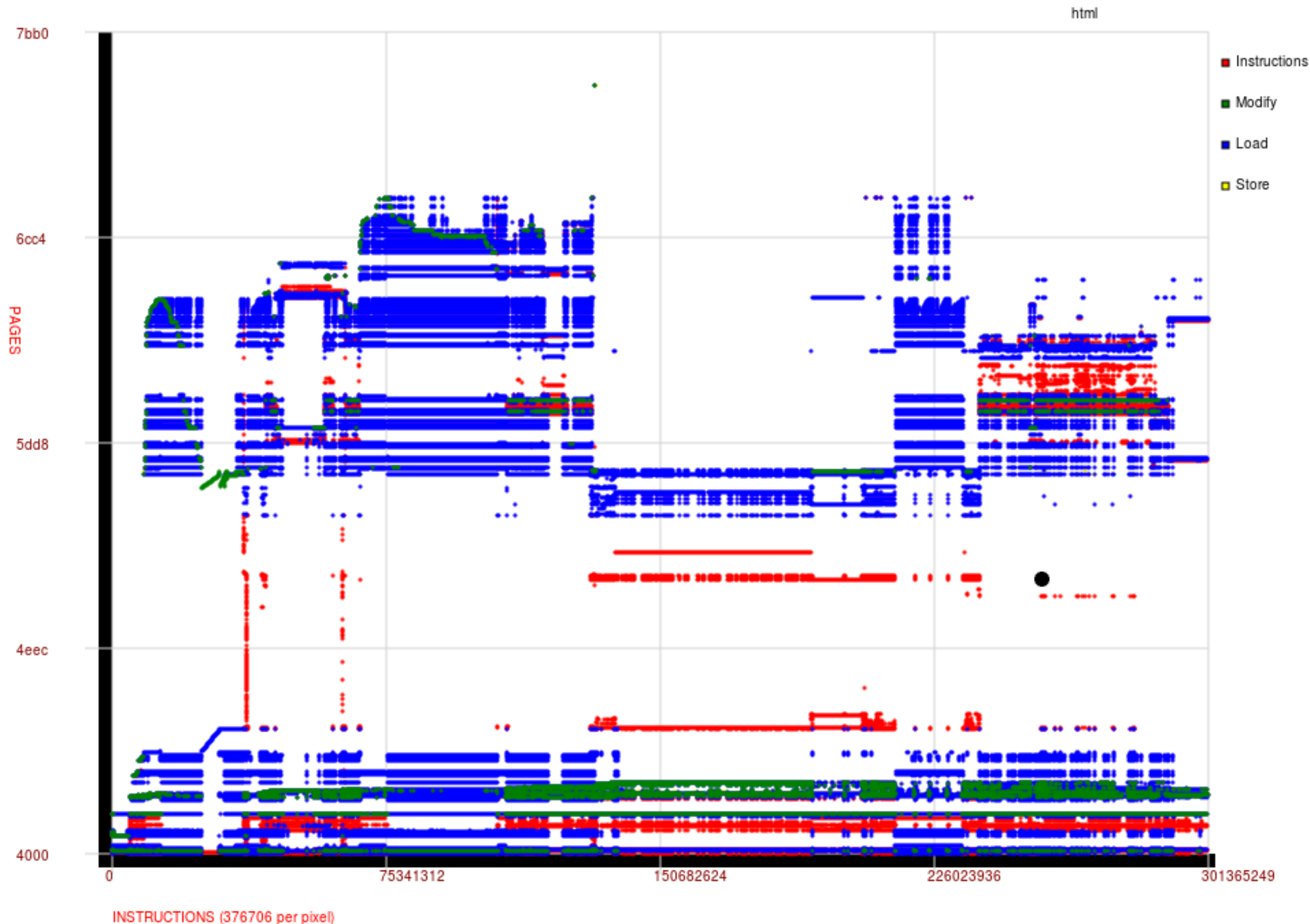
Temporal:
Accesele la aceeași adresă sunt de obicei grupate în timp

Spațial:
La accesul unei locații, tind să fie accesate și adresele din imediata vecinătate



From Peter Denning's *CACM* paper, July 2005 (Vol. 48, No. 7, pp. 19-24)

Exemplu: Pagini memorie accesate la deschiderea și închiderea Firefox



https://cartesianproduct.wordpress.com/2011/11/05/some-thoughts-on-the-linux-page-cache/

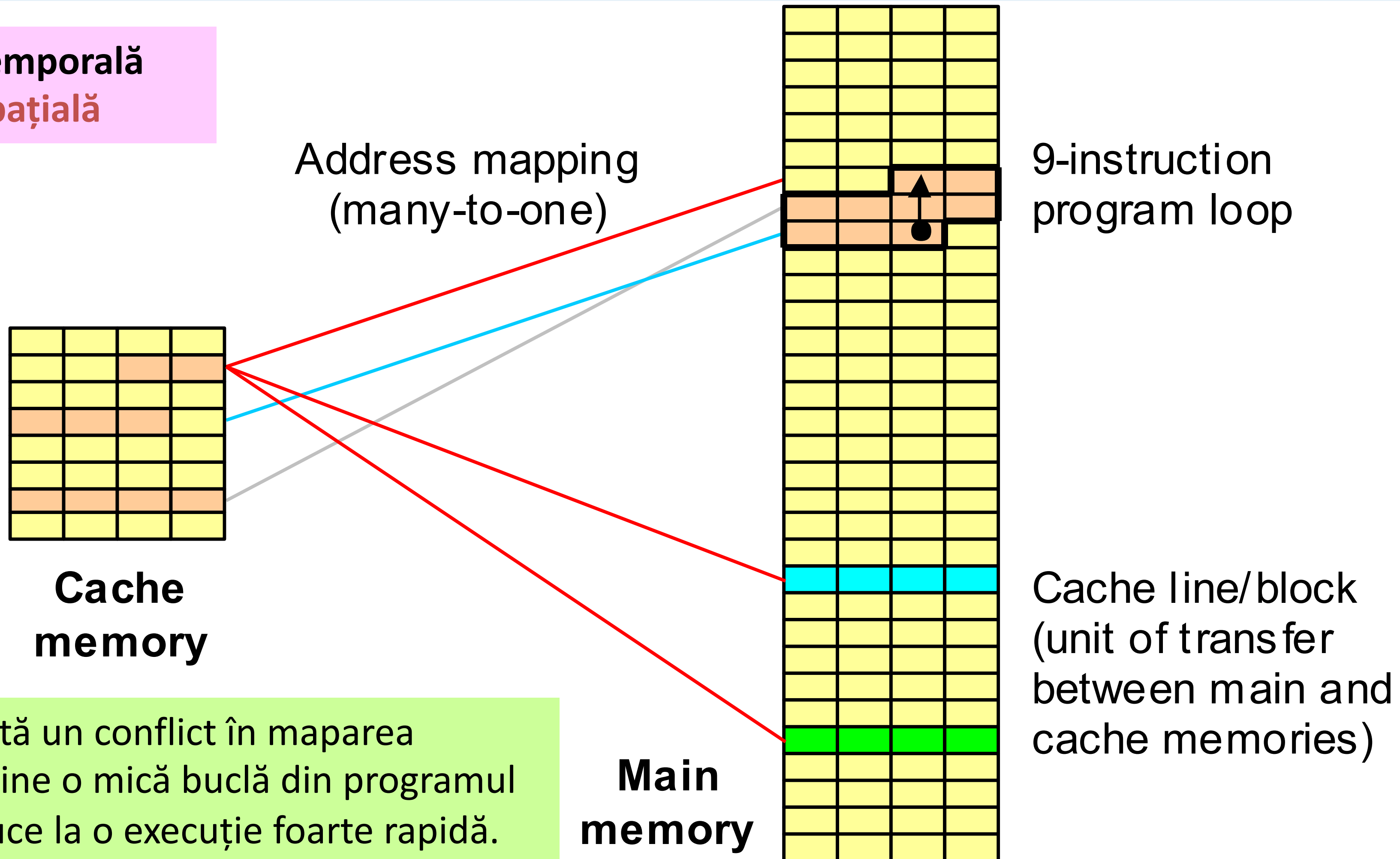
Memoriile cache exploatează ambele tipuri de predicții

- Exploatează localitatea temporală prin memorarea conținutului adreselor de memorie accesate recent.

- Exploatează localitatea spațială prin aducerea de blocuri de date din proximitatea locațiilor recent accesate.

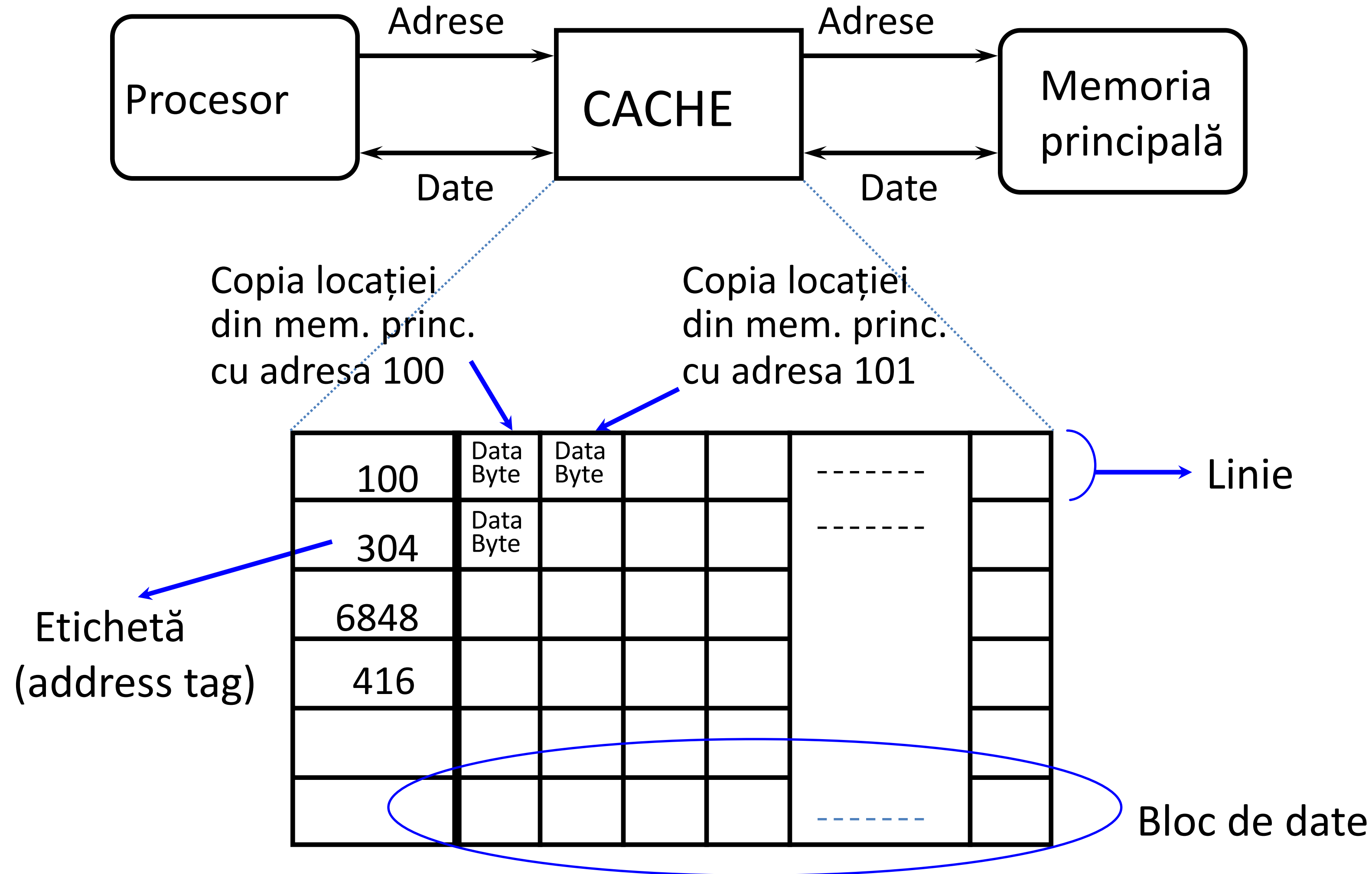
Cum funcționează un cache?

Localitate temporală
Localitate spațială



Presupunem că nu există un conflict în maparea adreselor, cache-ul va ține o mică buclă din programul principal, ceea ce va duce la o execuție foarte rapidă.

În interiorul unei memorii cache



Beneficiile caching-ului legate de legea lui Amdahl

Formulați problema cu sertarele din slide-ul anterior folosind legea lui Amdahl. Presupuneți că aveți un hit rate h.

Soluție

Fără sertarul din birou, orice document e accesat în 30s. De exemplu, ca să accesăm 1000 documente durează 30 000s. Sertarul face ca un procent h din cazuri să fie tratate de 6 ori mai repede, timpul de acces rămânând acelasi pentru restul de 1 - h procente. Prin urmare, avem un speedup de $1/(1 – h + h/6) = 6 / (6 – 5h)$. Dacă îmbunătățim timpul de acces la sertar (viteza memoriei cache) putem să creștem speedup-ul, dar, cât timp rata de miss rămâne $1 – h$, nici un speed-up nu poate depăși $1 / (1 – h)$. Pentru un $h = 0.9$, de exemplu, avem un speed-up de 4, iar limita superioară este 10, pentru un timp de acces extrem de scurt.

Notă: Unii oameni ar pune toate dosarele pe birou, în ideea că așa se poate atinge un speed-up mai mare. Nu vă recomand așa ceva!

Compulsory, Capacity, & Conflict Misses

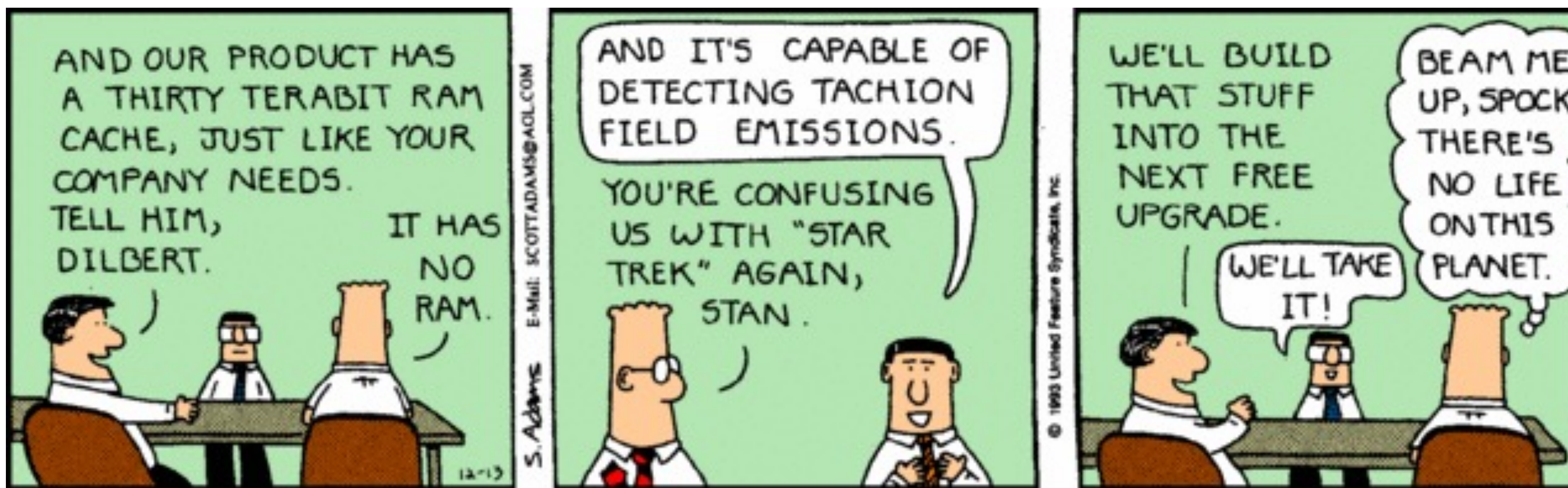
Compulsory misses: Dacă avem o politică de fetching *on-demand,* primul acces la orice resursă va fi întotdeauna un miss. O parte din aceste miss-uri ”obligatorii” pot fi evitate prin prefetching.

Capacity misses: Trebuie să ne debarasăm de o parte din date pentru a face loc altora. Acest lucru duce la miss-uri, datorate capacității limitate a memoriei cache.

Conflict misses: Ocazional, există spațiu ocupat de date care sunt inutile, dar strategia de alocare/mapare ne forțează să invalidăm intrări utile pentru a aduce date noi. Acest lucru poate duce de asemenea la miss-uri.

Dacă avem un cache de capacitate fixă, primele două tipuri de miss sunt mai mult sau mai puțin fixate în jurul unor valori date. Al treilea tip, însă, este influețat de strategia de mapare, care este sub controlul utilizatorilor.

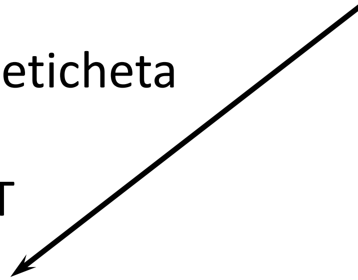
Discutăm în continuare despre două tipuri de mapare: directă și set-asociativă.

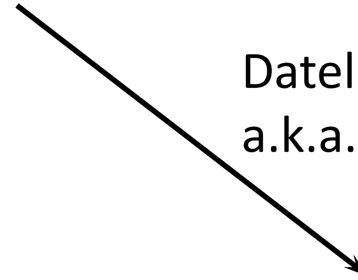


http://dilbert.com/strips/comic/1993-12-13/

Algoritmul de funcționare (Read)

Parcurge adresele date de procesor și caută etichetele care corespund. Atunci, ori:





Am găsit eticheta
în cache
a.k.a. HIT

Datele nu sunt în cache
a.k.a. MISS

Întoarce procesorului
Copia datelor din cache

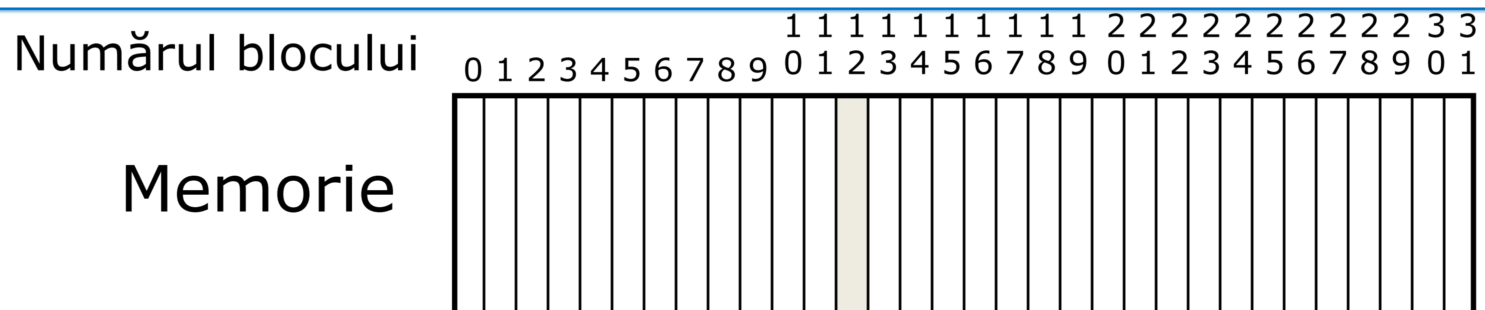
Citește blocul de date din mem. princ.

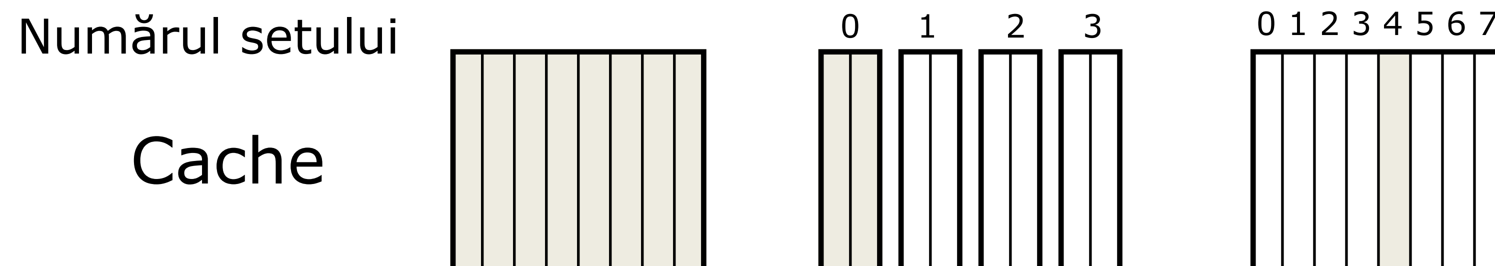
Așteaptă …

Întoarce datele procesorului și actualizează cache-ul

Q: Ce linie din cache înlocuim?

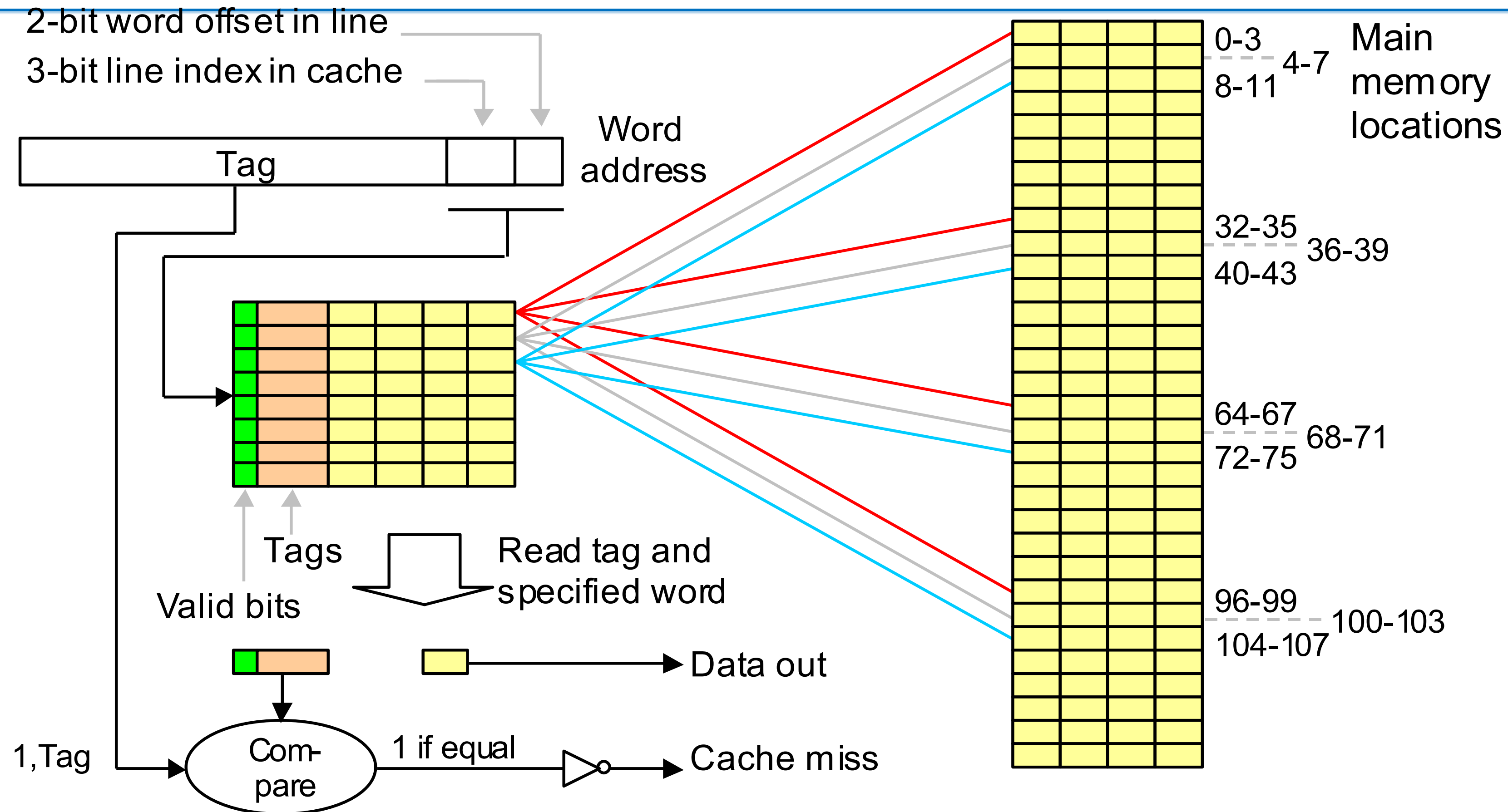
Politica de plasare a liniilor





	Complet Asociativ	(2-way) Set Asociativ	Mapat direct
blocul 12 poate fi plasat	oriunde	oriunde în setul 0 *(12 mod 4)*	numai în blocul 4 *(12 mod 8)*

Cache mapat direct



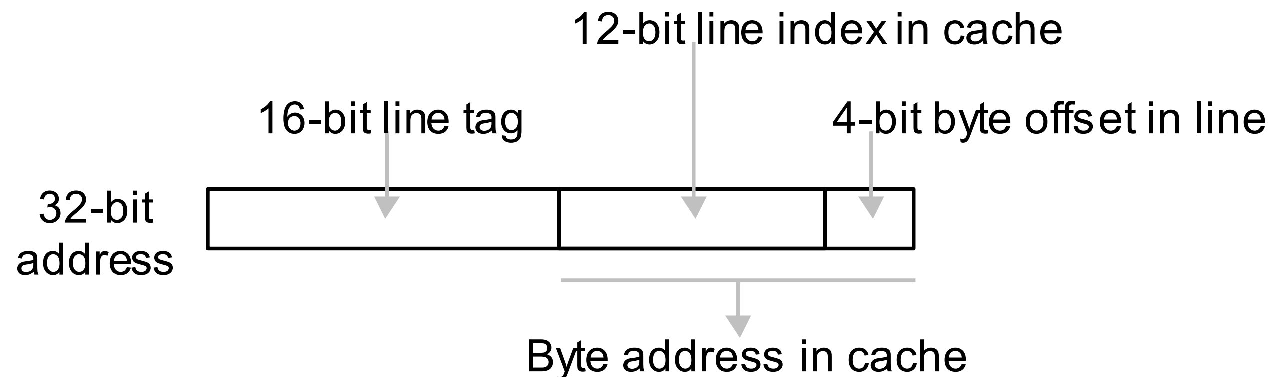
Memorie cache mapată direct ce conține 32 de cuvinte cu opt linii a câte 4 cuvinte.
Fiecare linie are o etichetă asociată și un bit de validitate.

Accesul la o memorie cache mapată direct

Descrieți adresarea unei memorii cache cu adrese pe 32 de biți, cu datele accesibile la nivel de octet. Linia de cache are o lățime *W* = 16 B. Dimensiunea cache *L* = 4096 linii (64 KB).

Soluție

Poziția unui octet în linie este codificată pe $\log_2 16 = 4$ b. Adresa de index a unei linii de cache este $\log_2 4096 = 12$ b.
Rămân 32 – 12 – 4 = 16 b pentru etichetă.

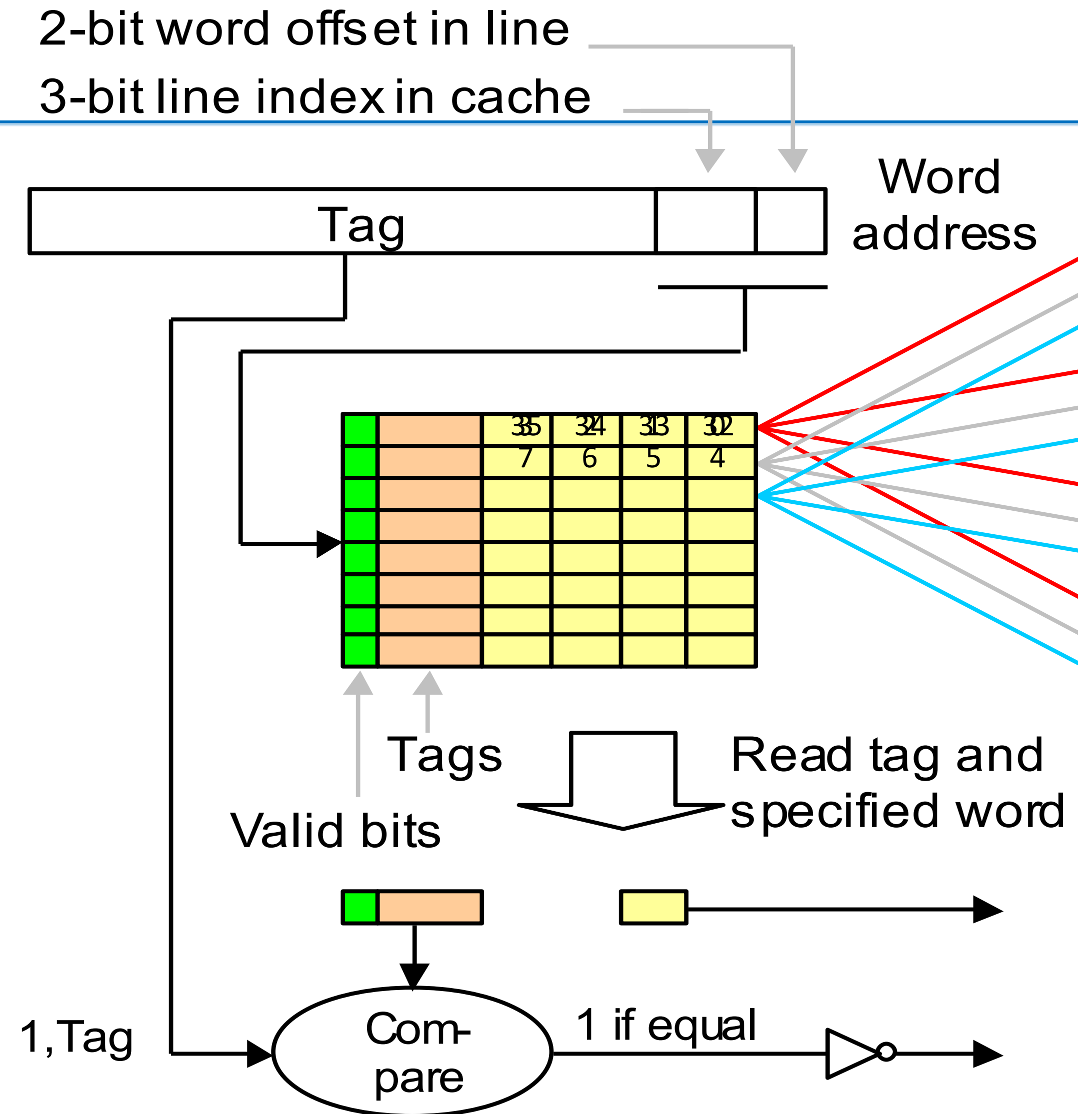


Componentele unei adrese de 32 de biți pentru un cache mapat direct cu adresare la nivel de octet.

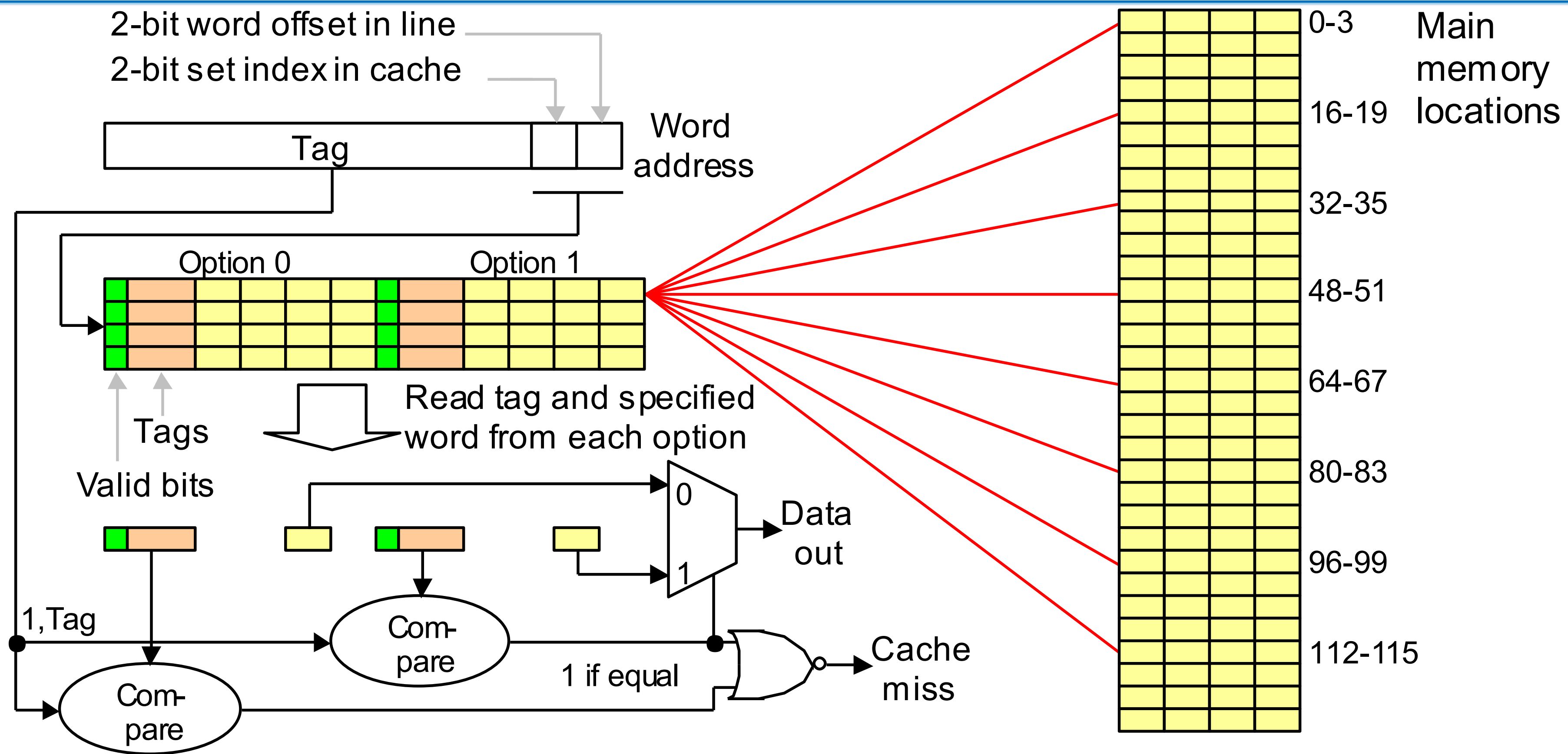
Comportamentul unui cache mapat direct

Trace pentru adrese:
1, 7, 6, 5, 32, 33, 1, 2, . . .

1: miss, fetch la liniile 3, 2, 1, 0
7: miss, fetch la liniile 7, 6, 5, 4
6: hit
5: hit
32: miss, fetch la 35, 34, 33, 32
(înlocuiește 3, 2, 1, 0)
33: hit
1: miss, fetch la 3, 2, 1, 0
(înlocuiește 35, 34, 33, 32)
2: hit
... ș.a.m.d.



Cache set-asociativ



Cache set-asociativ ce conține 32 de cuvinte de date aranjate în două seturi ce conțin linii de 4 cuvinte.

Accesul la un cache set-asociativ

Descrieți schema de adresare pentru o memorie adresabilă la nivel de octet cu adrese pe 32 de biți.

Lățimea liniei de cache $2^W = 16$ B.

Mărimea setului $2^S = 2$ linii.

Mărimea cache $2^L = 4096$ linii (64 KB).

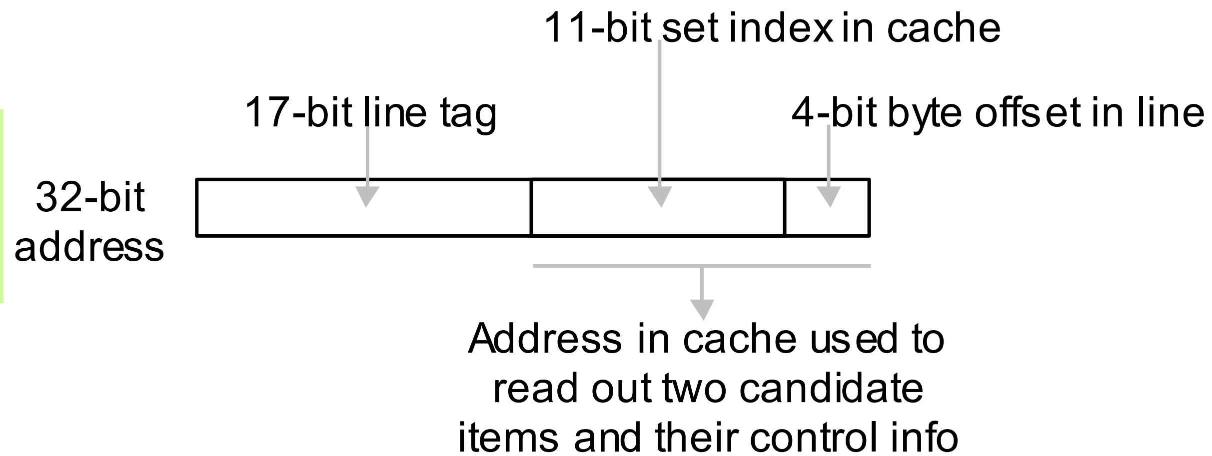
Soluție

Adresa unui octet din linie $\log_2 16 = 4$ b.

Adresa de index într-un set cache $(\log_2 4096/2) = 11$ b.

Rămân 32 – 11 – 4 = 17 b pentru etichetă.

Componentele unui cache set-asociativ cu adresare pe 32 de biți și 2 seturi.



Maparea adreselor

Un cache set-asociativ de 64 KB eight-way este adresabil la nivel de octet și conține linii de 32 de octeți. Adresele de memorie au 32 de biți.

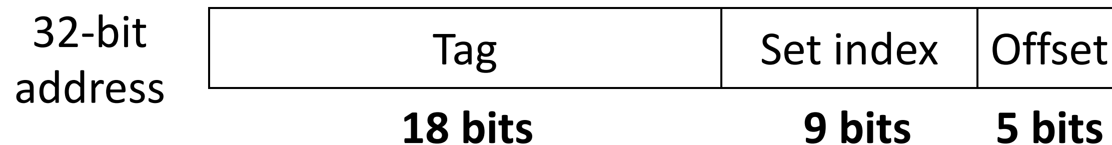
a. Care este dimensiunea etichetelor acestui cache?

b. Ce adrese din memoria principală sunt mapate în setul numărul 5?

Soluție

a. O adresă (32 b) = 5 b byte offset + 9 b set index + 18 b tag

b. Adresele care au set-index (lung de 9 biți) egal cu 5 au o formă generală $2^{14}a + 2^{5}\times 5 + b$, de ex. 160-191, 16 554-16 575, . . .



Tag width = $32 - 5 - 9 = 18$

Set size = 4×32 B = 128 B
Number of sets = $2^{16}/2^{7} = 2^{9}$

Line width = 32 B = 2^{5} B

Cache set-asociativ cu 2 seturi

Tag | Index | Block Offset

b

t

k

V | Tag | Data Block

V | Tag | Data Block

t

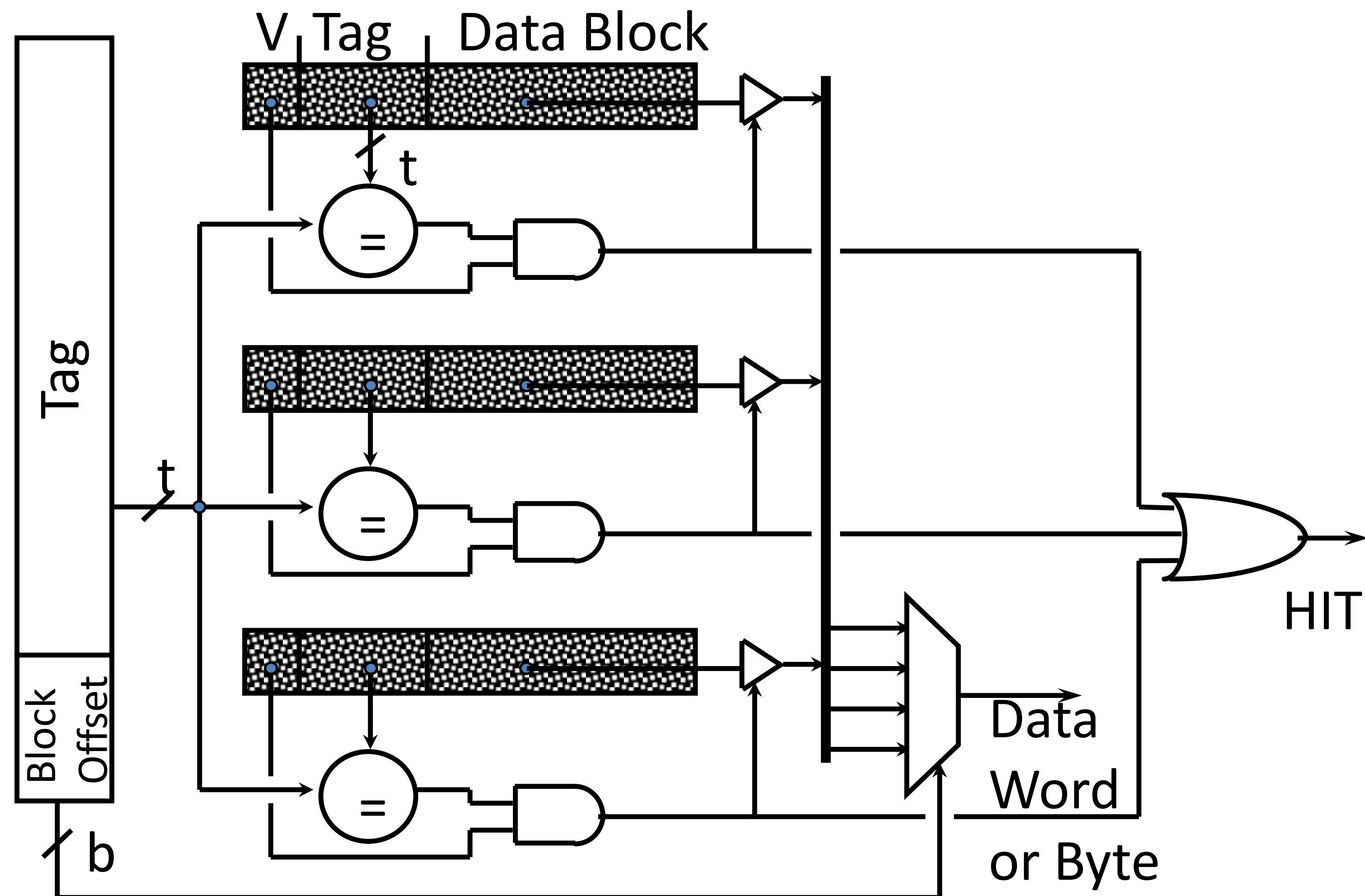
=

=

Data Word or Byte

HIT

Cache complet asociativ



Politica de înlocuire

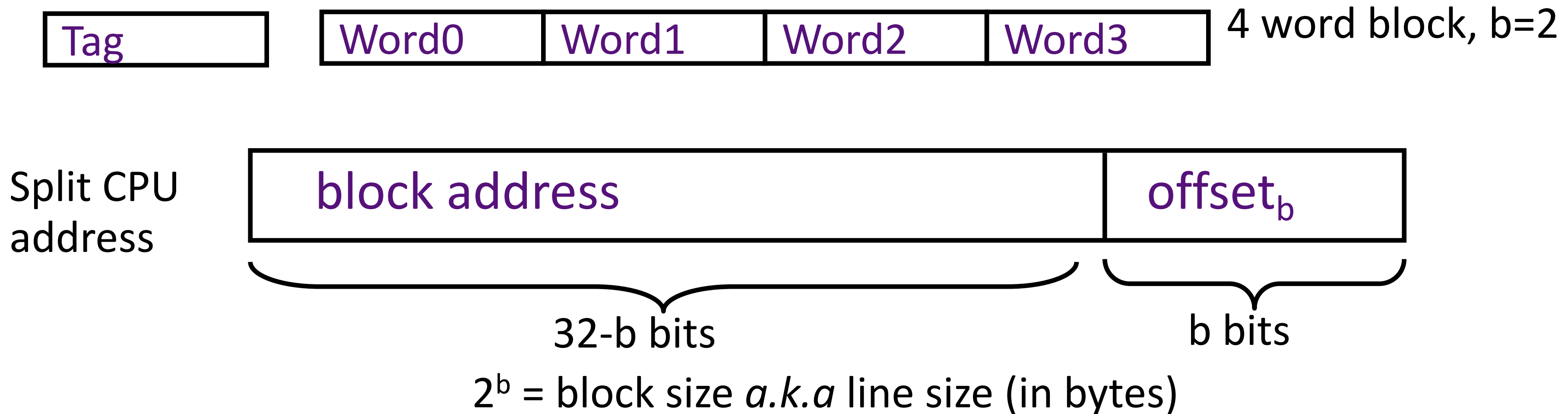
Într-un cache asociativ, care bloc dintr-un set trebuie invalidat atunci când setul se umple?

- Aleatoriu
- Least-Recently Used (LRU)
    - Starea pentru cache LRU trebuie actualizată la fiecare acces
    - Implementare funcțională și fezabilă posibiă doar pentru un număr mic de seturi (2-way)
    - pseudo-LRU – arbore binar folosit pentru cache 4-8 way
- First-In, First-Out (FIFO) a.k.a. Round-Robin
    - folosit în cache-urile complet-asociative
- Not-Most-Recently Used (NMRU)
    - FIFO, cu excepția blocului/blocurilor cel mai recent folosite

Este un efect de ordin secundar. De ce?

Înlocuirea se petrece NUMAI la un cache miss

Mărimea blocurilor și localitatea spațială



Un bloc de dimensiuni mari are avantaje distincte d.p.d.v. hardware

- mai puțin overhead pentru etichete
- exploatează capabilitatea memoriei DRAM de a transfera date în rafală
- exploatează transferurile în rafală prin magistralele de date de lățime mare (32-64 biți)

Care sunt dezavantajele creșterii dimensiunii blocurilor?

Mai puține blocuri => mai multe conflicte. Poate să irosească lățime de bandă.

Cache și memoria principală

Cache separat: memorii cache diferite pentru date și instrucțiuni (L1)
Cache unificat: conține instrucțiuni și date (L1, L2, L3)

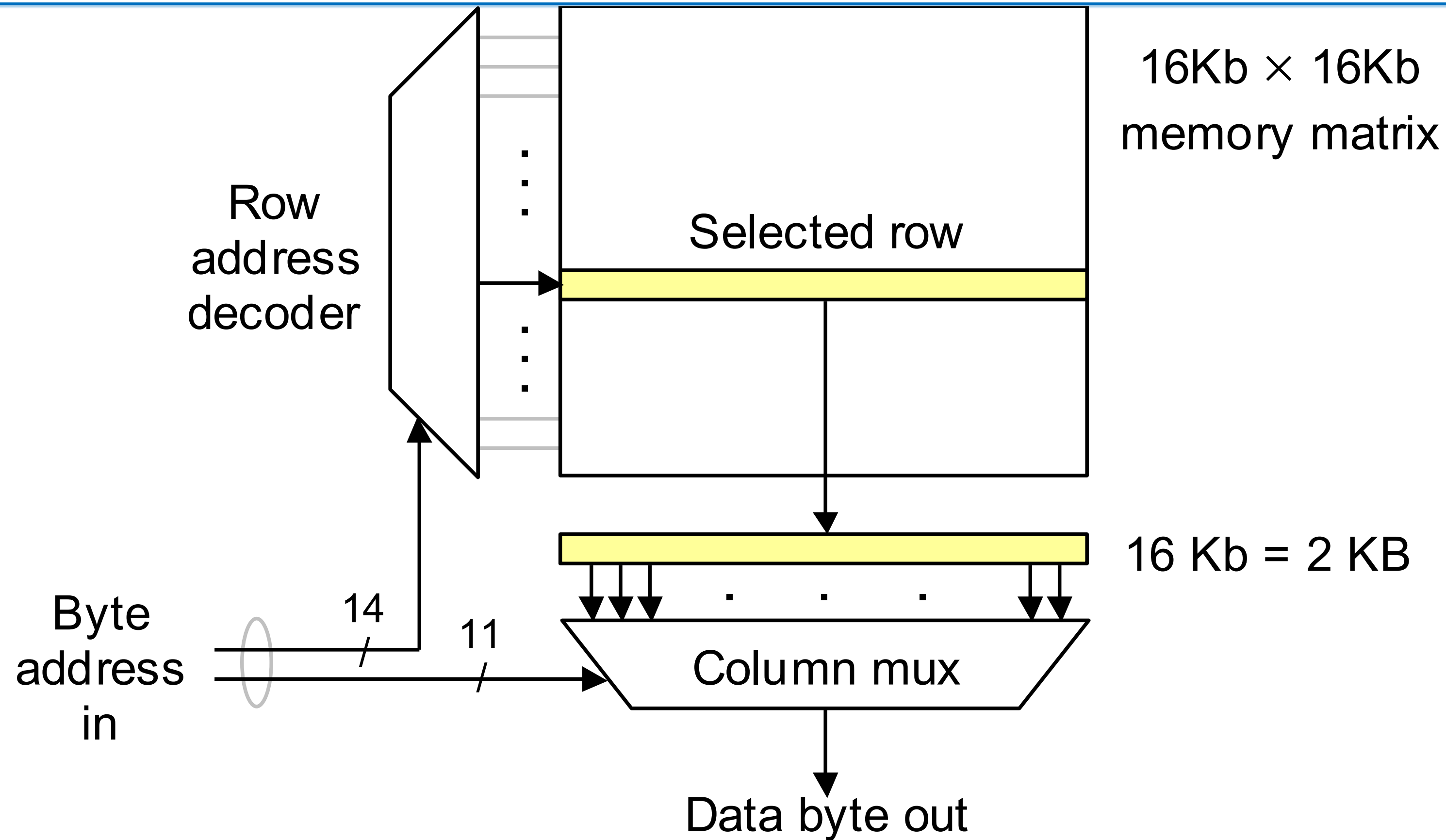
Arhitectură Harvard: memorii de date și instrucțiuni separate
Arhitectură von Neumann: o singură memorie pentru date și instrucțiuni

Probleme la scriere:

Write-through încetinește memoria cache pentru a permite memoriei principale să scrie datele

Write-back sau copy-back este mai puțin problematic, dar tot dăunează performanțelor din cauza acceselor multiple la memorie.
Soluție: Dotează memoria cache cu buffering la scriere a.î. nu trebuie să aștepte memoria principală.

Transferuri de date rapide între cache și memoria principală



Un chip DRAM de 256 Mb este organizat ca o memorie 32M × 8: patru astfel de chipuri constituie o memorie de 128MB.

Îmbunătățirea performanțelor memoriei cache

Pentru o dimensiune cache dată, există următoarele probleme de design:

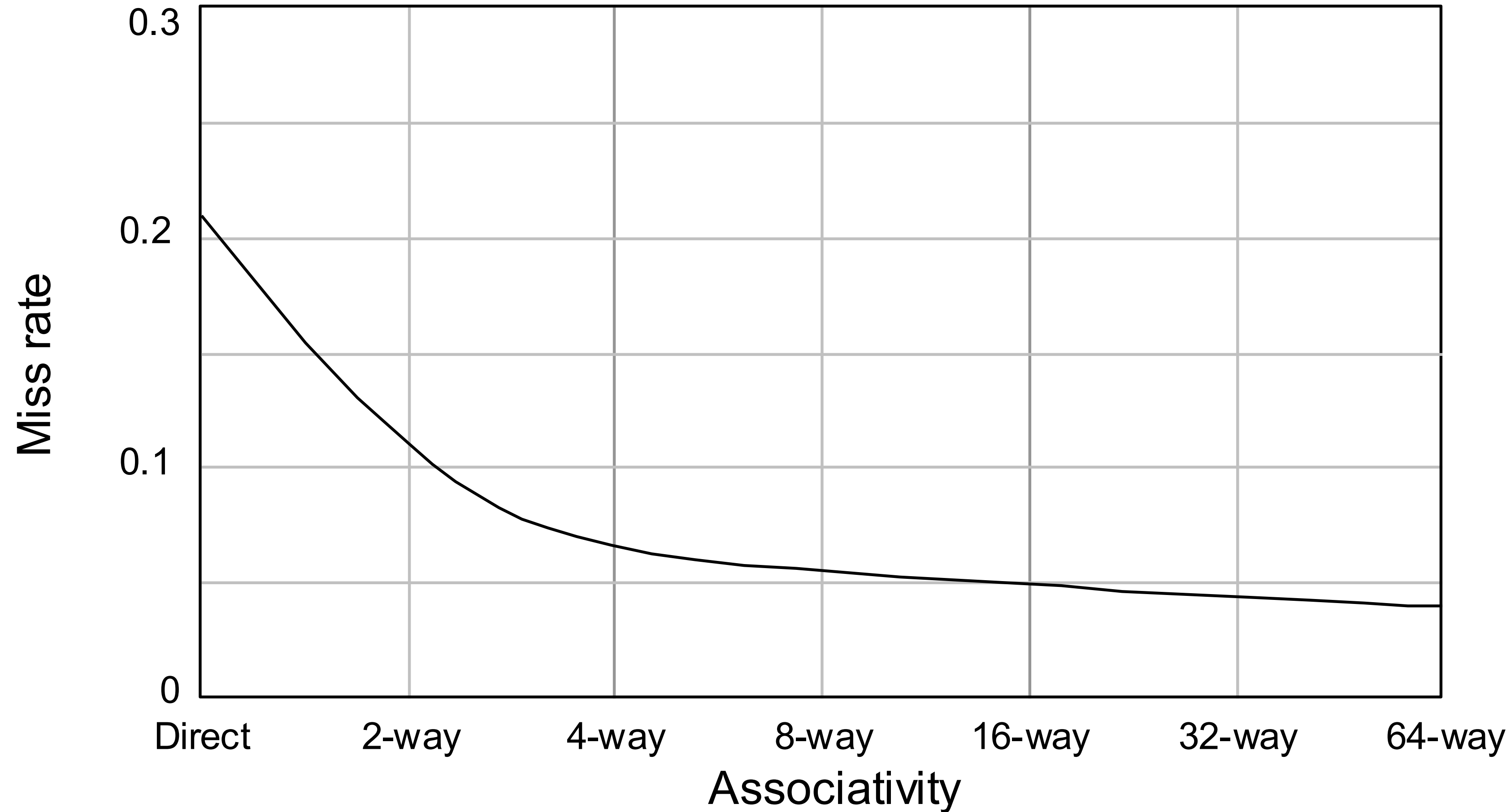
Lățimea liniei (2^W). O valoare prea mică a *W* cauzează mai multe accese la memoria principală; o valoare prea mare crește penalizarea pentru un miss și poate să încarce memoria chache cu date de utilitate scăzută, care pot fi înlocuite înainte de a fi utilizate.

Mărimea setului sau asociativitatea (2^S). Mapare directă ($S = 0$) este simplă și rapidă; o mai mare asociativitate duce la o complexitate mărită și la timpi de acces mai mari dar tinde să reducă miss-urile conflictuale.

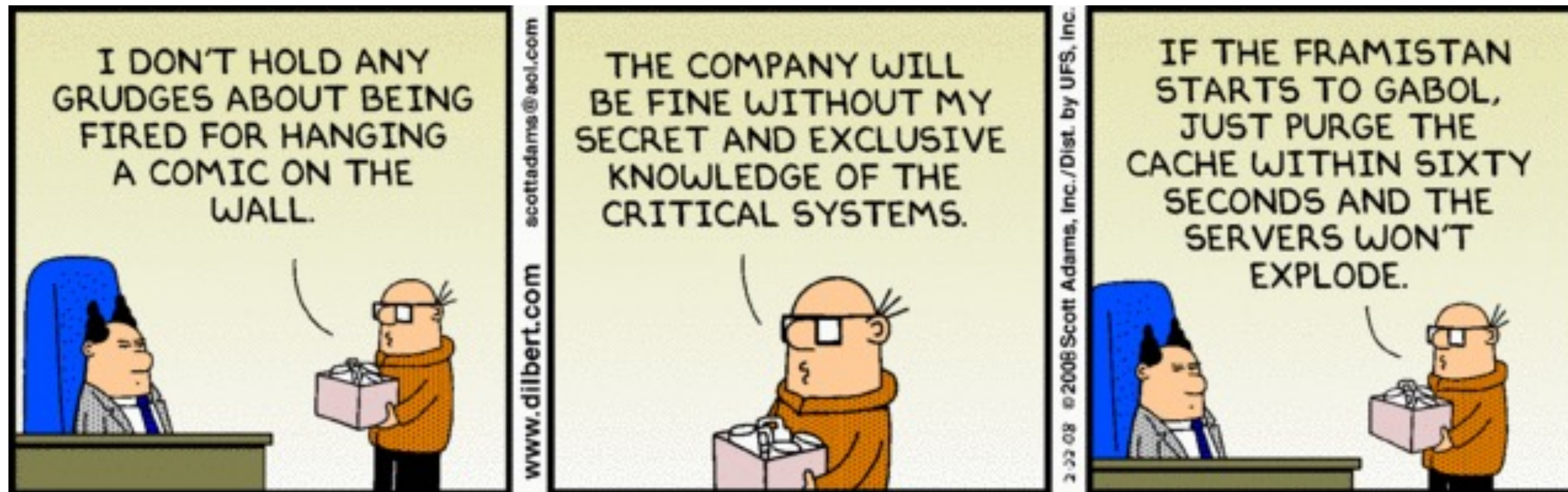
Line replacement policy. De obicei este algoritmul LRU (least recently used); nu este o problemă pentru cache-ul mapat direct. Funcționează bine și un algoritm de selecție aleatoare a liniilor pe care să le invalidăm.

Write policy. Memoriile cache moderne sunt foarte rapide, a.î. *Write-through* nu este aproape niciodată o politică bună. De obicei alegem *write-back* sau *copy-back*, folosind buffering la scriere pentru a minimiza impactul adus de latența memoriei principale.

Efectele asociativității asupra performanței



Îmbunătățirea performanțelor memoriei cache în funcție de asociativitate.



http://dilbert.com/strips/comic/2008-02-22/