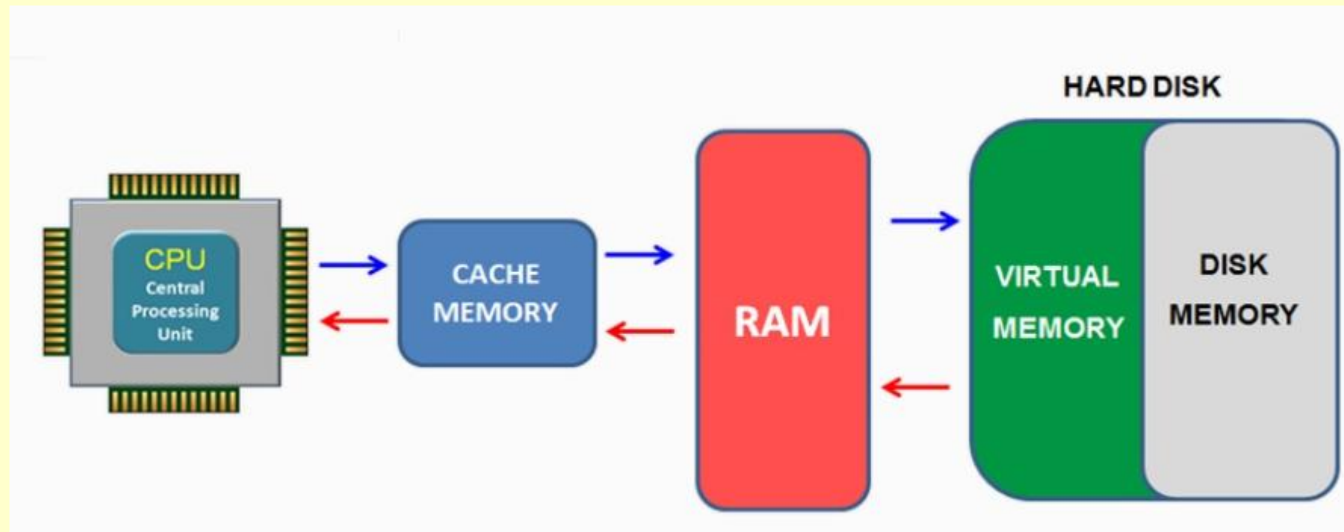
„Legile ţării nu interzic nimănui să fie imbecil”   G. Moisil

C3. Memoria Cache si Memoria virtuala



1. Rolul MC
2. Elementele MC
3. Arhitectura MC
4. Organizarea MC
5. MC la Pentium
6. Identificarea caracteristicilor MC
7. Memoria virtuala

References:

https://mecha-mind.medium.com/demystifying-cpu-caches-with-examples-810534628d71

1. ROLUL MC

Registers

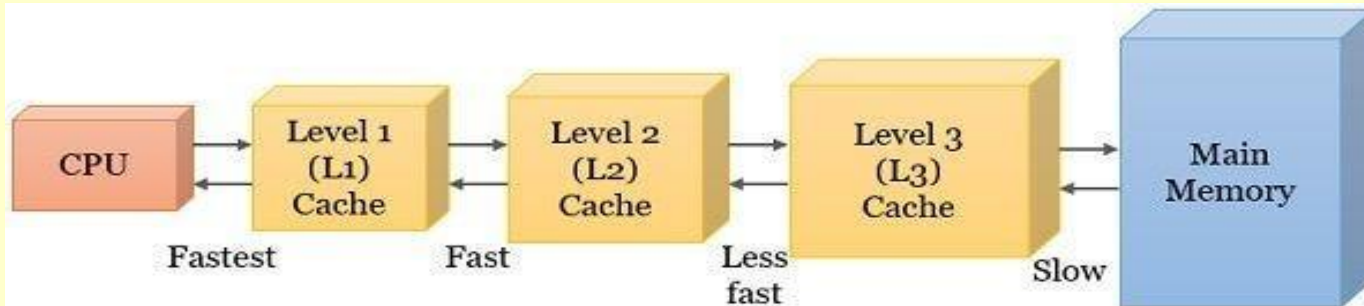
L1 Cache

L2 Cache

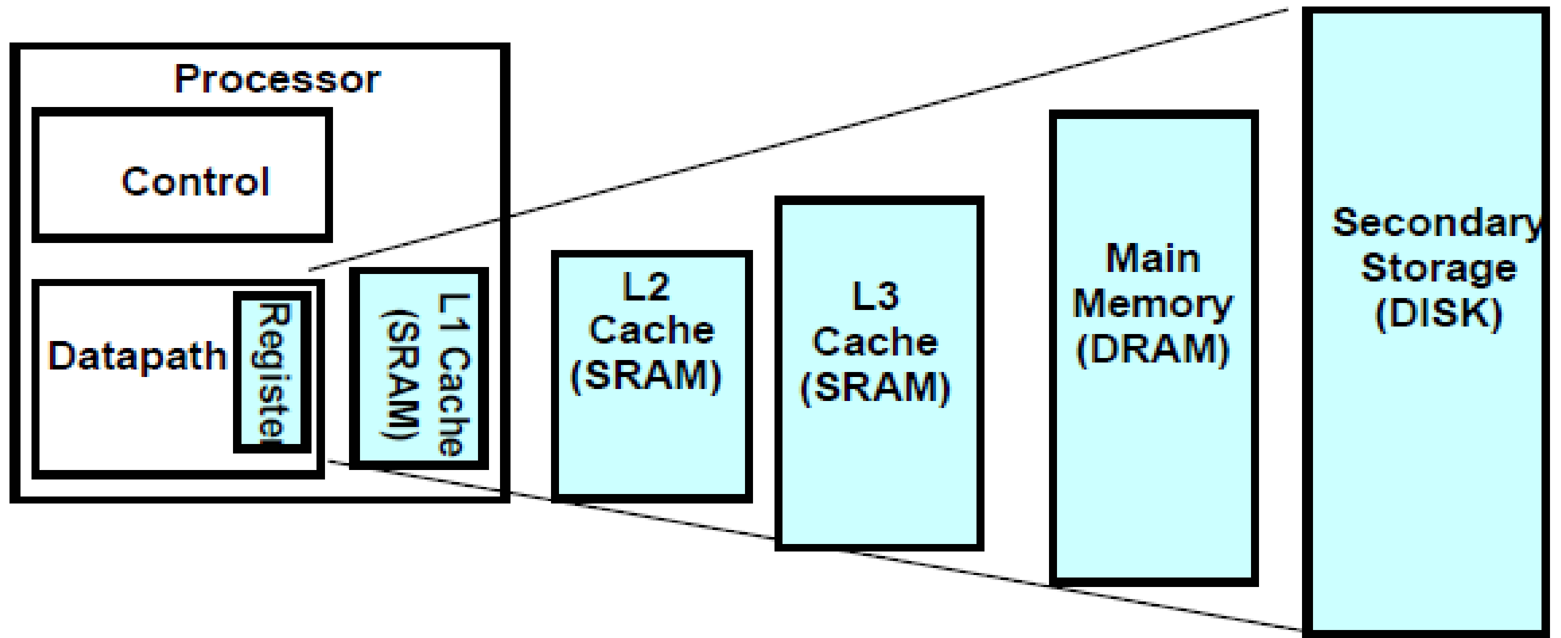
CPU

RAM

Main Memory



Cache Organization in Computer



Level	Access Time	Typical Size	Technology	Managed By
Registers	1-3 ns	1 KB ?	Custom CMOS	Compiler
Level 1 Cache (on-chip)	2-8 ns	8 KB-128 KB	SRAM	Hardware
Level 2 Cache (off-chip)	5-12 ns	0.5 MB - 8 MB	SRAM	Hardware
Main Memory	10-60 ns	64 MB - 1 GB	DRAM	Operating System
Hard Disk	3,000,000 - 10,000,000 ns	20 - 100 GB	Magnetic	Operating System/User

- Ce este MC?

  - O memorie (SRAM) de mica capacitate care contine cele mai recent accesate locatii din memoria de baza (DRAM) –

  - conceptul de MC apare la sistemul IBM S360-85 (1968)

- De ce este utila prezenta MC?

- la procesoarele actuale timpul de citire instr.>>> timpul de executie

Tacc ~ 60ns DRAM ; *Tex* =1CLK ~ 10ns - Pentium 100

==> gâtuire la intrarea procesor ; *Tacc ~5-10ns MC*

- Cum poate o memorie de mica capacitate sa imbunatateasca performantele sistemului?

- principiul este *"locality of reference" (localizarea referintei)*

https://igoro.com/archive/gallery-of-processor-cache-effects/

Ierarhizarea memoriei functioneaza datorita localizarii referintei

Principiul Localizarii

– Programele acceseaza o zona relativ mica a spatiului de adrese la un moment dat

• regula 90/10: 90% din accese se fac in 10% din locatiile de memorie

Tipuri de Localizare

– *Spatiala:* daca o secventa a fost accesata recent, secventele invecinate urmeaza sa fie accesate in curand

– *Temporala:* daca o secventa a fost accesata recent, aceasta tinde sa fie reaccesata in curand

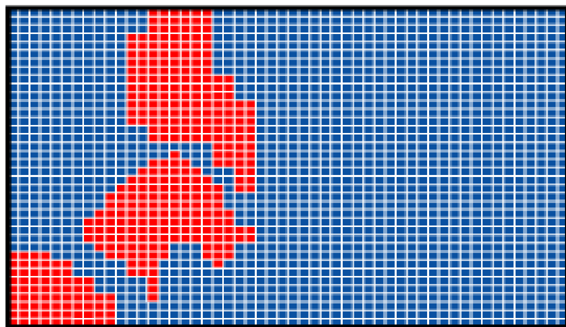
• Ideea de baza a ierarhizarii memoriei cache :

– Se plaseaza o copie a celor mai recent accesate date/cod la nivelele superioare ale ierarhiei memoriei

– Procesorul cauta cea mai apropiata copie a datelor/codului

Localizarea spațială

- Localizarea spațială este cea mai ușoară localizare de înțeles, pentru că cei mai multi au aplicatii multimedia si au folosit mp3 playere, DVD playere și alte tipuri de aplicații, care au seturi de date mari si constau din fișiere ordonate.
- Considerand un fișier MP3, care are o multime de blocuri de date, care sunt trimise către procesor de la secvența de început a fișierului la capăt. În cazul în care procesorul rulează Winamp și doar a solicitat secventa de la 2':23'' din 5' , a unui fișier MP3 putem fi siguri că urmeaza seventele 2':24'', 2':25'', și așa mai departe.
- Acest lucru este la fel cu un fișier pe DVD, precum și cu multe alte tipuri de fișiere media cum ar fi imagini, desene AutoCAD, etc. Toate aceste aplicații utilizează ***seturi mari de date secvențiale, ordonate,*** care se transmit secvential către CPU



In imagine, celulele rosii sunt zone compacte de date din zona de memorie. Aceasta imagine prezinta un program cu *localizare spațială destul de buna*, deoarece celulele roșii sunt grupate împreună. Într-o aplicație cu localizare spațială slabă , celulele rosii vor fi distribuite aleatoriu între celulele albastre.

Localizare temporală

- Consideram un simplu filtru implementat pe Photoshop care inversează o imagine, pentru a produce o imagine negativa ; Există o mică bucată de cod care realizează aceasta inversiune pe fiecare pixel, începand de la un colț și continua în secvență pana la colțul opus
- Acest cod este doar o buclă mică care este executata în mod repetat ptr. fiecare pixel, asa ca este un exemplu de cod care este reutilizat iar și iar.

- *Aplicațiile media, jocurile și simulările ,* deoarece folosesc multe bucle mici, care itereaza pe seturi de date foarte mari, au *o localizare temporală excelenta pentru cod*. Cu toate acestea, este important de reținut că aceste tipuri de aplicații *au localizari temporale extrem de slabe pentru date.*

```
sum =0;
for (i=0; i<n; i++)
     sum +=a(i);
return sum;
```

Observatii

- Revenind la exemplul cu fișierul MP3, acesta este de obicei redat printr-o singura trecere a succesiunii de date și nici una dintre partile sale nu sunt repetate. In acest caz, este de fapt un fel de risipa a stoca un astfel de fișier în memoria cache, deoarece este stocat acolo doar temporar, înainte de a trece prin CPU. Atunci când o aplicație umple cache-ul cu date, care nu au cu adevărat nevoie să fie stocate în cache, deoarece acestea nu vor fi folosite din nou, ca urmare se spune că aplicatia *"polueaza memoria cache".*

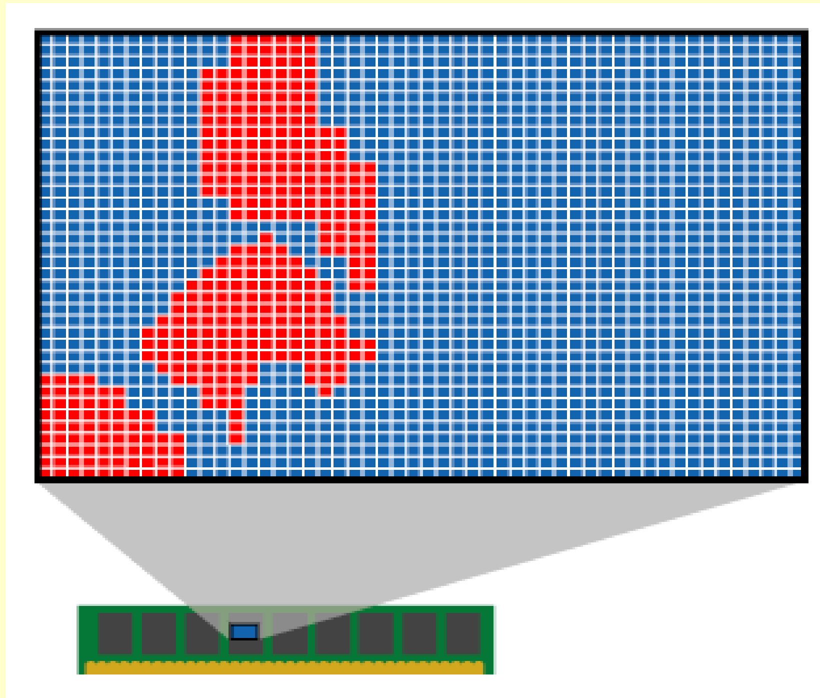
- *Aplicațiile media, jocurile și alte asemenea aplicatii sunt „mari poluatori" ai memoriei cache*, motiv pentru care nu au fost prea afectate de lipsa cache-ului la procesoarele Celeron originale. Pentru că au transmis sirul de date prin procesor la o rată foarte mare, ele de fapt nu sunt interesate că datele lor nu erau în cache. Deoarece aceste date nu au fost necesare din nou prea curând , faptul că nu a fost într-un cache ușor accesibil nu conteaza.

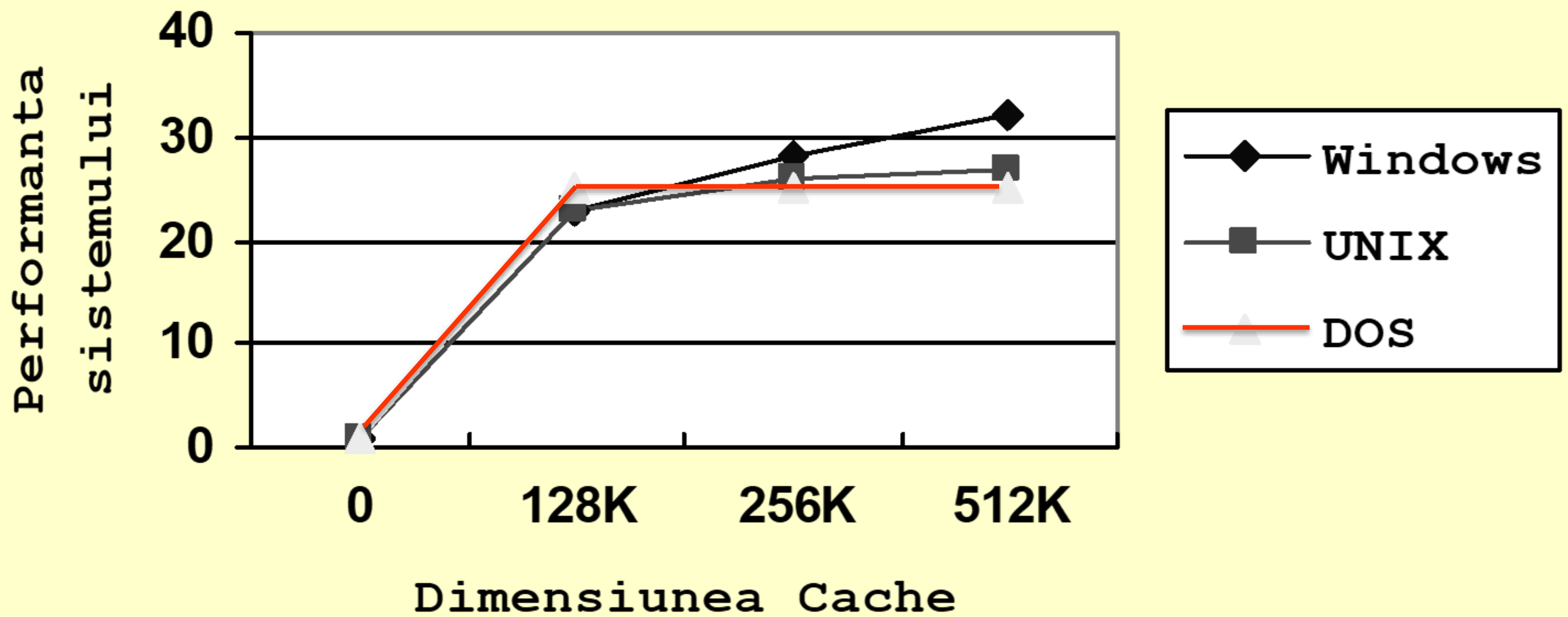
- Cat de eficient este acest mecanism?

(P100) – 16KB MC contine ~ 90% din adresele cerute, deci 90% de accese rapide (SRAM)

- De ce nu inlocuim DRAM cu SRAM?

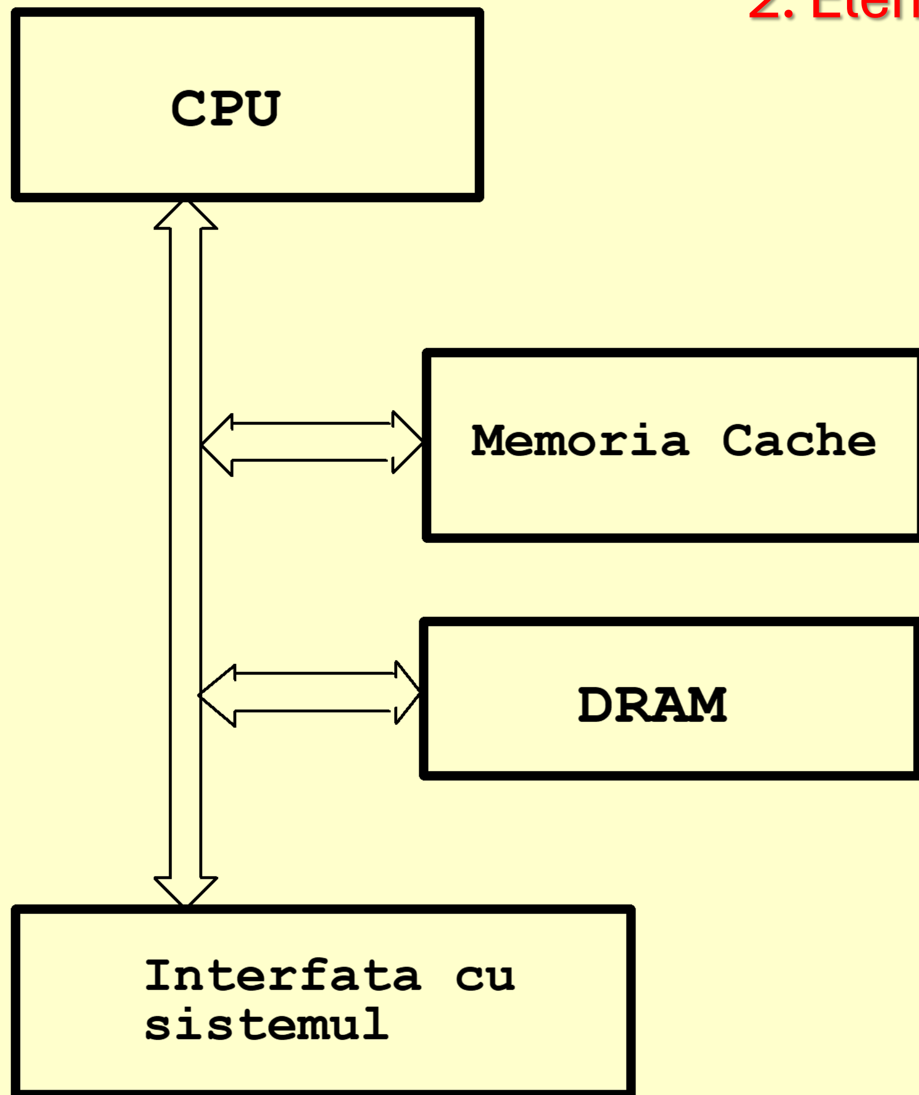
  - cost + consum





Performanţele relative ale SO funcţie de mărimea memoriei cache (www. Intel.com)

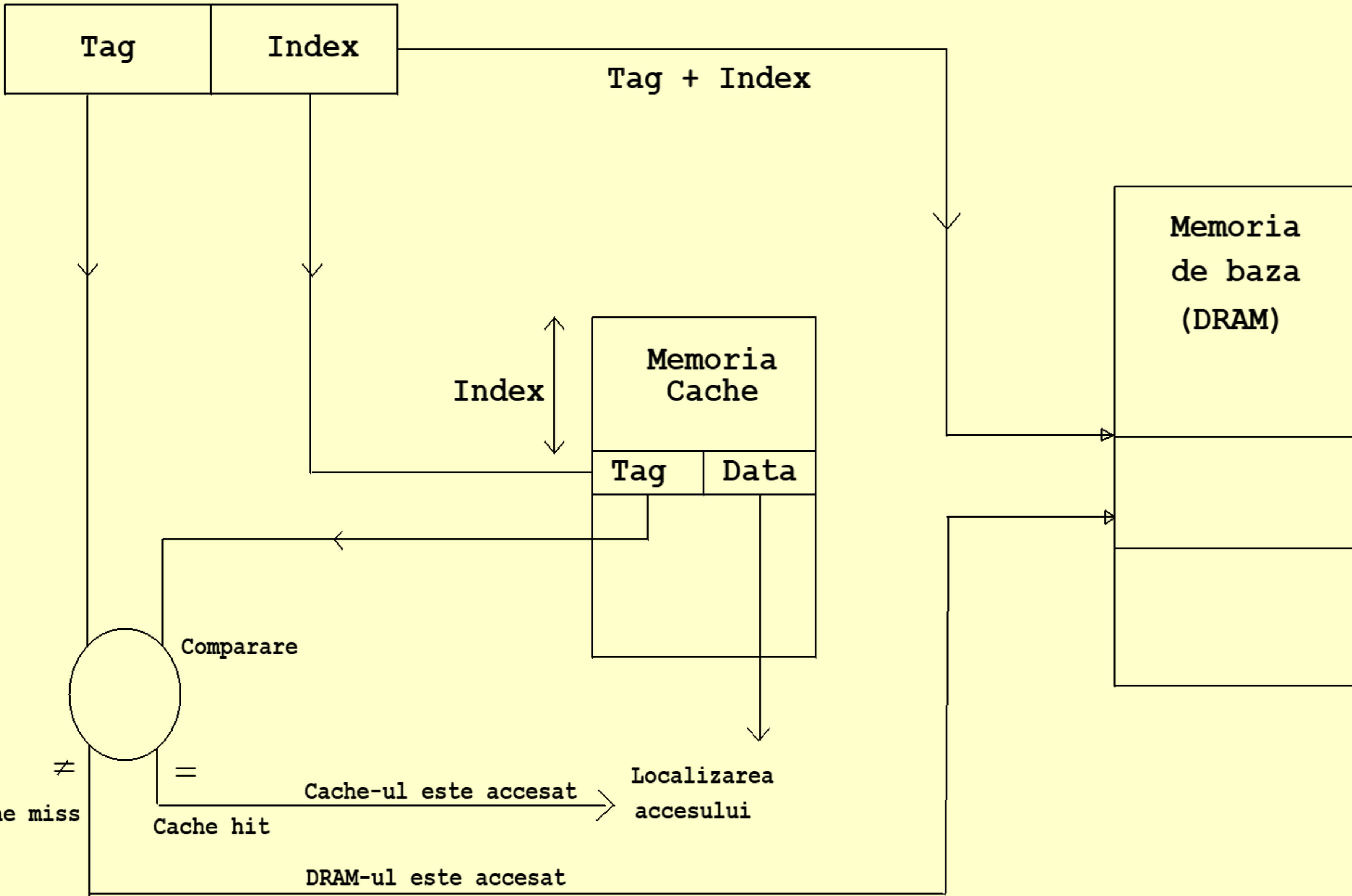
2. Elementele memoriei cache



Modelul de bază al memoriei cache

- **SRAM** este blocul de memorie SRAM care păstrează datele/codul aplicatiei
- **Tag RAM-ul** (TRAM) este o porţiune de SRAM care stochează adresele datelor stocate în SRAM.
- **Controller-ul cache** gestionează accesul la memoria cache, iar sarcinile lui :

-supravegherea fluxului de date cerut de procesor
- împrospătarea SRAM şi TagRAM
-implementarea metodei de scriere
-Stabileste dacă cererea este un *„cache miss”* sau *„cache hit”*



Adresa memoriei
de la procesor
Tag
Index
Tag + Index
Memoria
de baza
(DRAM)
Index
Memoria
Cache
Tag
Data
Comparare
≠
=
Cache miss
Cache-ul este accesat
Cache hit
Localizarea
accesului
DRAM-ul este accesat

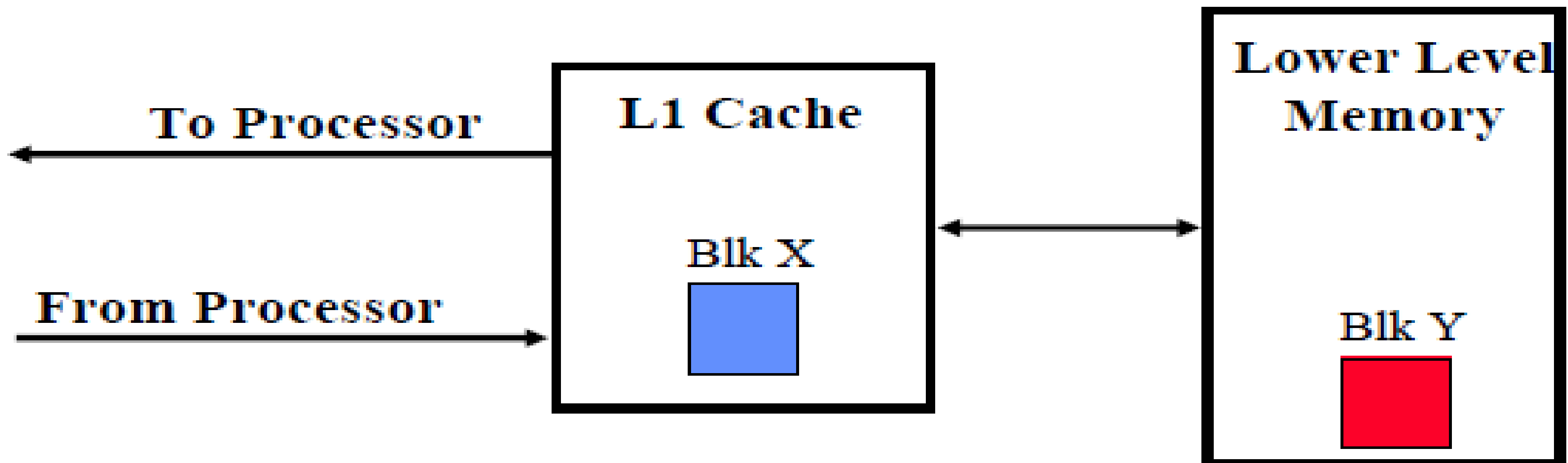
Terminologie

•„**Cache hit**" desemnează o tranzacţie în care memoria cache conţine informaţiile cerute de procesor (ex. Block X)

•„**Cache Miss**" indică o tranzacţie în care memoria cache nu conţine informaţiile cerute de procesor (Block Y)

• **Consistenţa cache-ului. MC** este o copie a unei zone mici din memoria de bază (DRAM); este important ca acesta să reflecte întotdeauna ce este în memoria de bază, adica :

(SRAM) = (DRAM)



• **spionare (snoop)** -Supravegherea liniilor de adrese de către controller-ul cache, pentru un transfer – acesta permite cache-ului să vadă dacă datele ce fac obiectul transferului sunt conţinute de el

• operaţia de actualizare -cache-ul preia informaţiile de pe liniile de date- se numeşte **„snarf”**

• Procesele de **„Snoop-Snarf”** permit MC să îşi menţină consistenţa

•**Hit**: data se afla in anumite blocuri ale cache-ului (exemplu: Block X)

–Hit Rate: fractiunea de accese reusite in cache

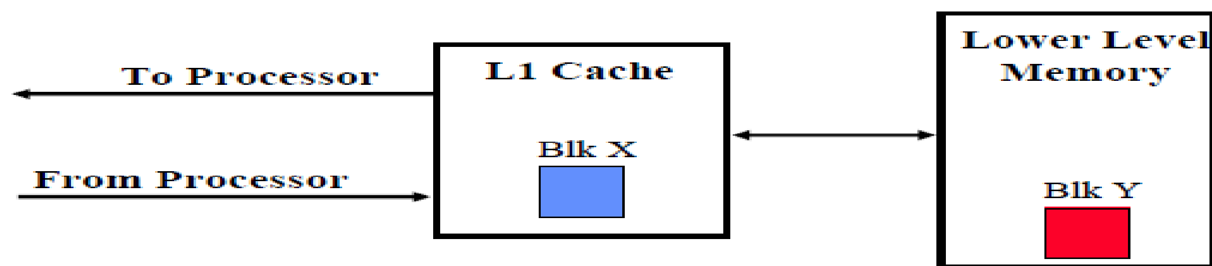
–Hit Time: timp de acces SRAM + Timp de determinare hit/miss

•**Miss**: data trebuie preluata din nivelele inferioare ale memoriei (BlockY)

–Miss Rate = 1 -(Hit Rate)

–Miss Penalty time: Timpul de extragere a blocului din nivelele inferioare ale memoriei

Average memory-access time = Hit time + Miss rate x Miss penalty



- Măsurătorile de *“hit rate"* sunt de obicei efectuate pe aplicații de referință *(benchmarks)*. Rata efectivă a succesului (hit rate) variază foarte mult de la o aplicație la alta.
- În special, aplicațiile de streaming video și audio au adesea ***rata de succes aproape de zero,*** deoarece fiecare byte de date din flux este citit o singura dată, folosit și apoi nu este citit sau scris din nou. Chiar mai rău, mulți algoritmi de cache permit ca aceste date de streaming să umple cache-ul, stergând informațiile din memoria cache care vor fi utilizate din nou în curând ***(poluarea cache-ului)***

Pentru a descrie inconsistenţa datelor din MC:

•**„Dirty Data”** adică date eronate („murdare”), atunci când datele sunt modificate în cache, dar nu şi în memoria de bază ;

•**„Stale Date”** („date depăşite”) când datele sunt modificate în memoria de bază, dar nu şi în memoria cache.

Calcularea timpului mediu de acces la memorie

Să presupunem că un calculator are o organizare a memoriei cu o ierarhie de numai două nivele: o ***memorie cache și memoria principală.*** Care este timpul mediu de acces la memorie având în vedere timpii de acces și ratele de ratare/miss din tabel?

Timpii de acces și ratele de ratare

Memory Level	Access Time (Cycles)	Miss Rate
Cache	1	10%
Main Memory	100	0%

Soluție

Timpul mediu de acces la memorie este de: $t_{acc} = 1 + 0,1*(100) = 11$ cicluri.

Calcularea performanței cache-ului

Să presupunem că un program are 2.000 de instrucțiuni de accesare a datelor (încărcări sau stocări) și că 1 250 dintre aceste valori de date solicitate se găsesc în memoria cache. Celelalte 750 de valori de date sunt furnizate procesorului de către memoria principală sau de către memoria de pe disc. Care sunt ratele de miss și de hit pentru memoria cache?

Soluție

Rata de miss este 750/2000 = 0,375 = 37,5%.

Rata de hit este de 1250/2000 = 0,625 = 1 -0,375 = 62,5%.

3. ARHITECTURA MC

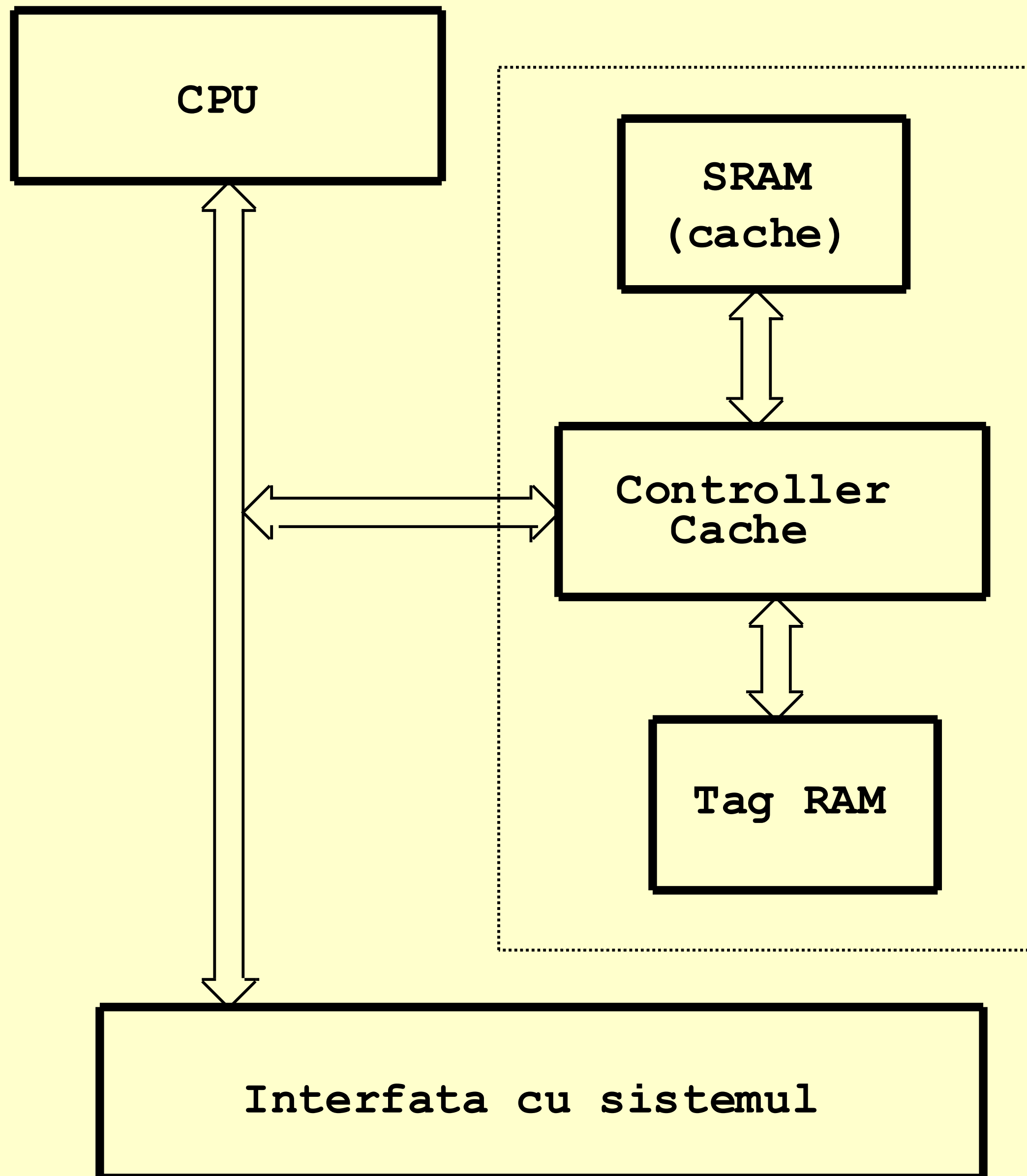
- Cache-ul este caracterizat de: - arhitectură de citire (RD)
  - metodă de scriere (WR)

arhitectura de citire : - "look aside"
- "look through“

metoda de scriere : - "write-back"
- "write through”
- ”write-bypass ”

https://en.wikibooks.org/wiki/Microprocessor_Design/Cache

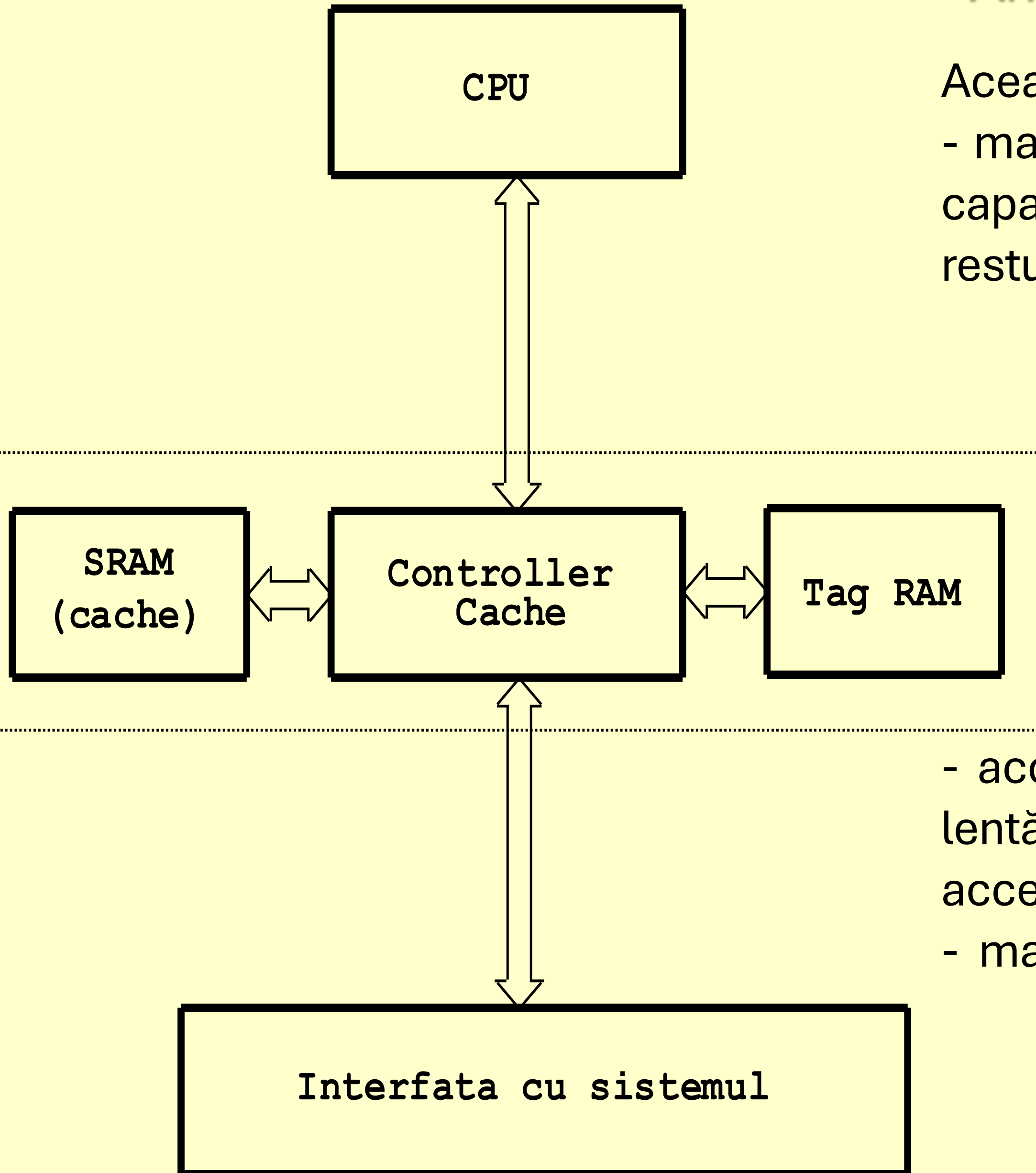
Arhitectura de citire „Look-aside”



Cache-ul *„look-aside”* este:

- simplu
- ieftin
- timp de răspuns bun în caz de *“cache miss”*

Arhitectura „Look-through”



Această arhitectură este:

- mai complexă, deoarece trebuie să fie capabilă să controleze accesul CPU la restul sistemului

- accesul la memoria sistem este mai lentă deoarece memoria de bază este accesată doar după accesul cache-ului.
- mai scumpa

Metode de scriere

- La metoda ***„write-back”*** memoria cache lucrează ca un buffer, MC recepţionează datele şi finalizează ciclul, iar apoi când bus-ul sistem este disponibil MC scrie datele în memoria de bază.

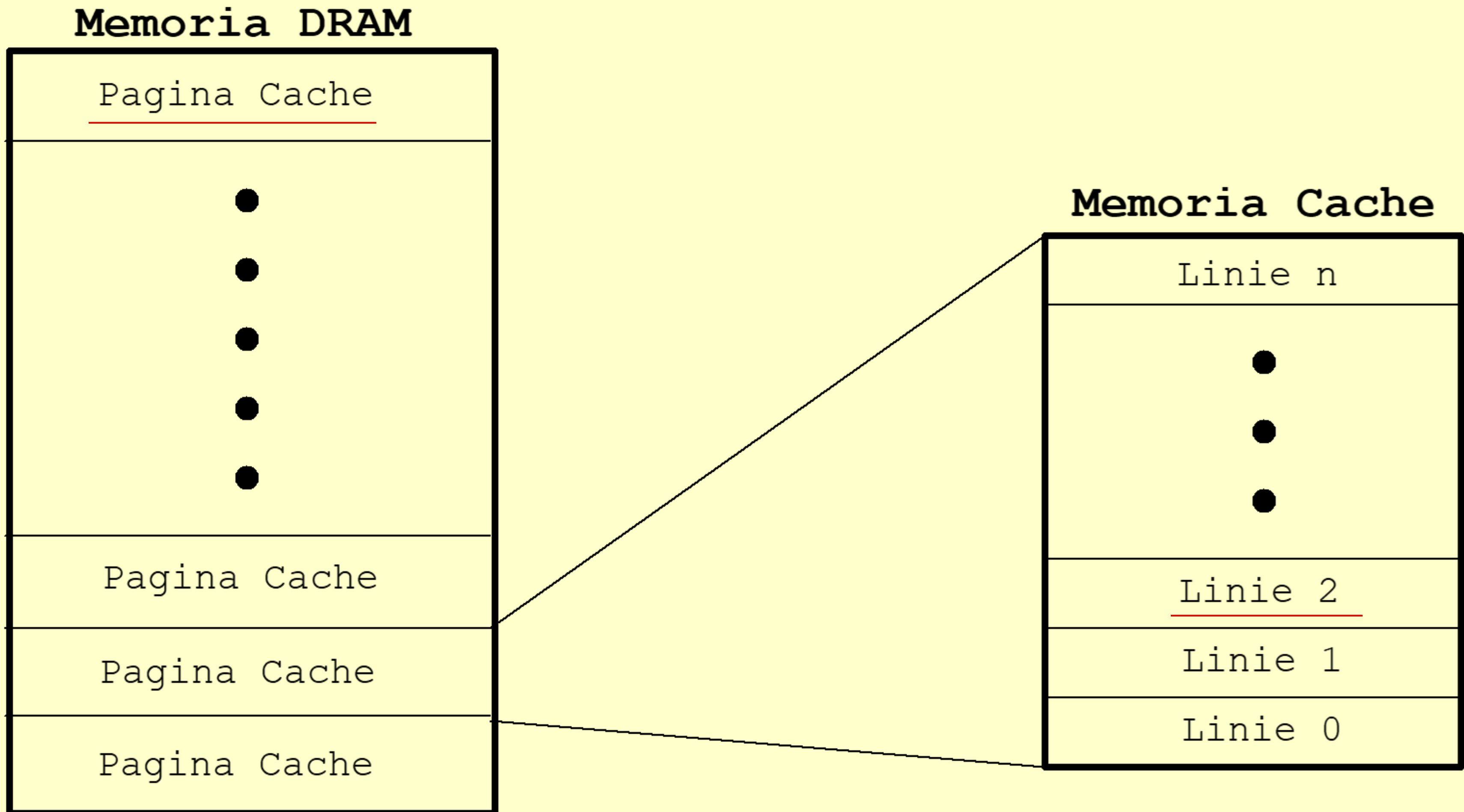
- asigură performanţa maximă, permiţând procesorului să-şi continue lucrul în timp ce memoria principală este actualizată mai târziu.
- Controlul scrierii în memoria de bază măreşte complexitatea memoriei cache şi costul.

- La metoda ***“write-through”*** procesorul scrie în MC şi în memoria de bază.

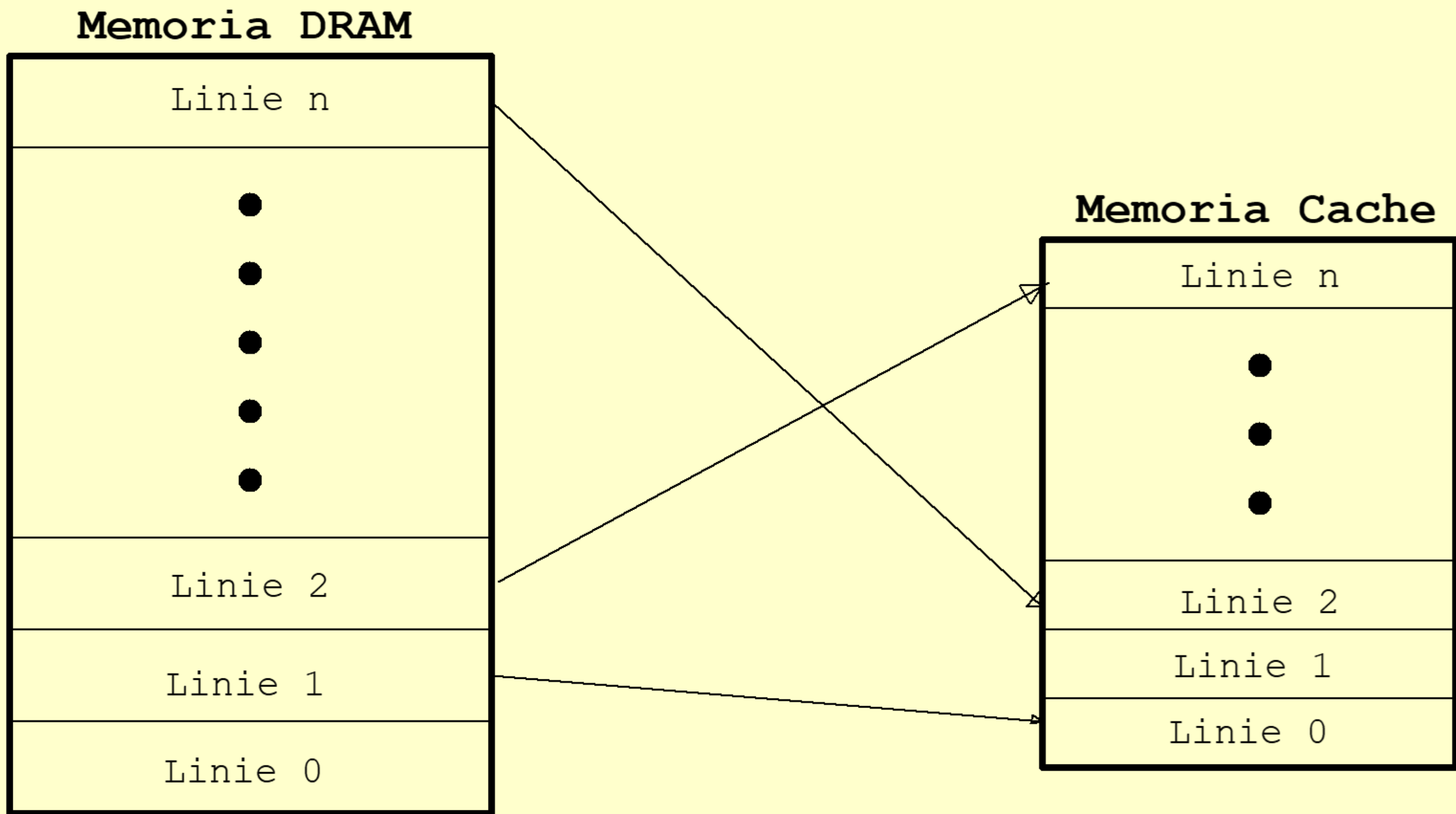
- Cache-ul îşi actualizează conţinutul însă ciclul de scriere continuă până când datele sunt stocate şi în memoria de bază;
- Metoda este mai puţin complexă şi de aceea mai ieftină;
- Performanţa mai modestă deoarece procesorul trebuie să aştepte actualizarea memoriei de bază

- La metoda ***“write bypass”*** unele procesoare scriu direct în memoria principală, ocolind memoria cache. Dacă această locație nu este deja stocată în memoria cache, atunci nu trebuie făcut nimic altceva. Dacă locația este deja stocată în memoria cache, atunci datele vechi din memoria cache trebuie să fie marcate ca fiind “invalide" ("stale"), astfel încât dacă procesorul citește vreodată acea locație, CPU va citi cea mai recentă valoare din memoria principală mai degrabă decât unele valori anterioare din cache.

4. Organizarea memoriei cache

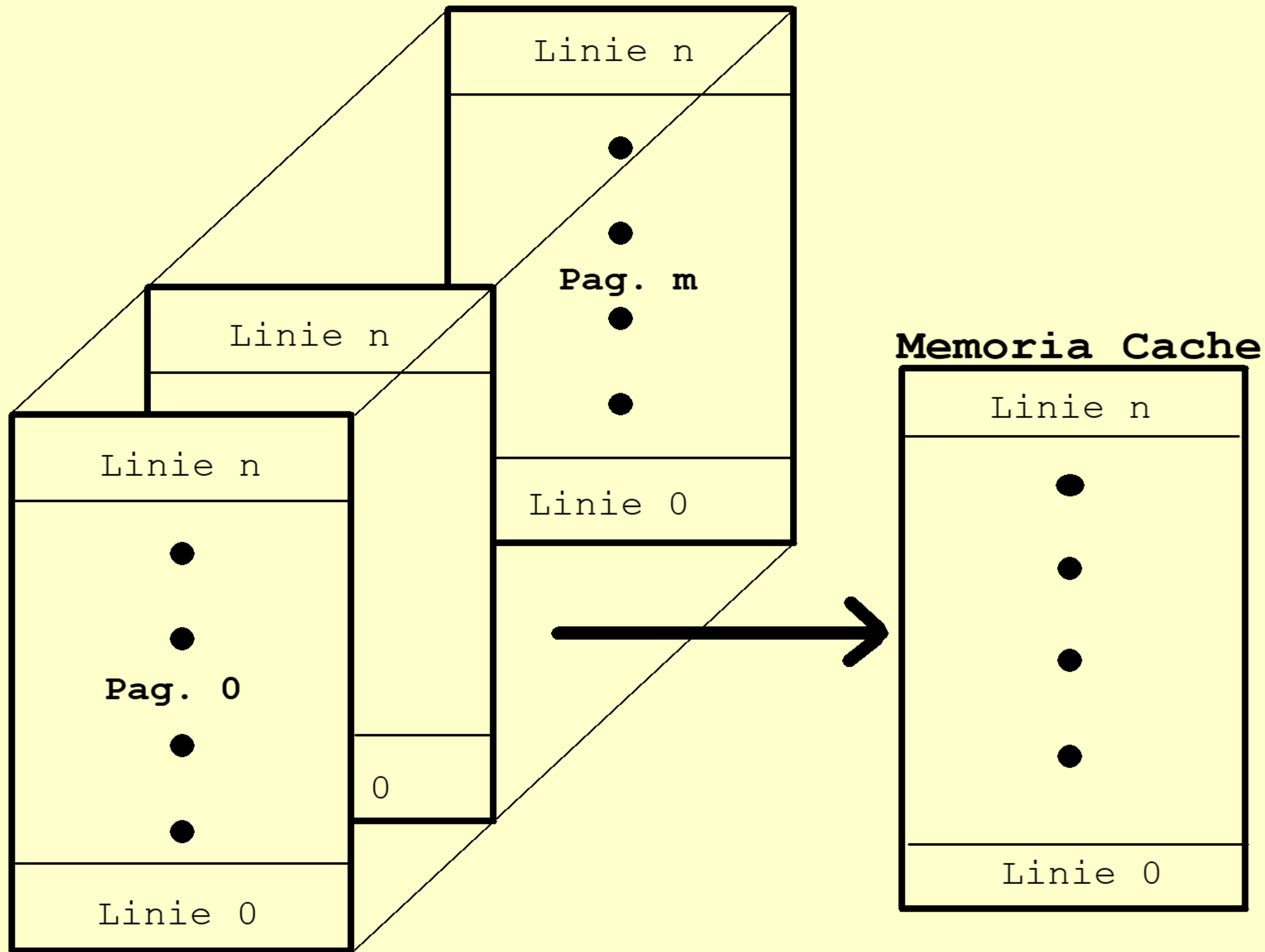


- *O linie cache* este un bloc de date din cache care reprezintă cea mai mică unitate de transfer de date între memoria principală DRAM și M. cache
- *Un slot cache* se referă la o locație specifică din cache unde poate fi stocată o linie cache.



Memoria cache total asociativă (full associative)

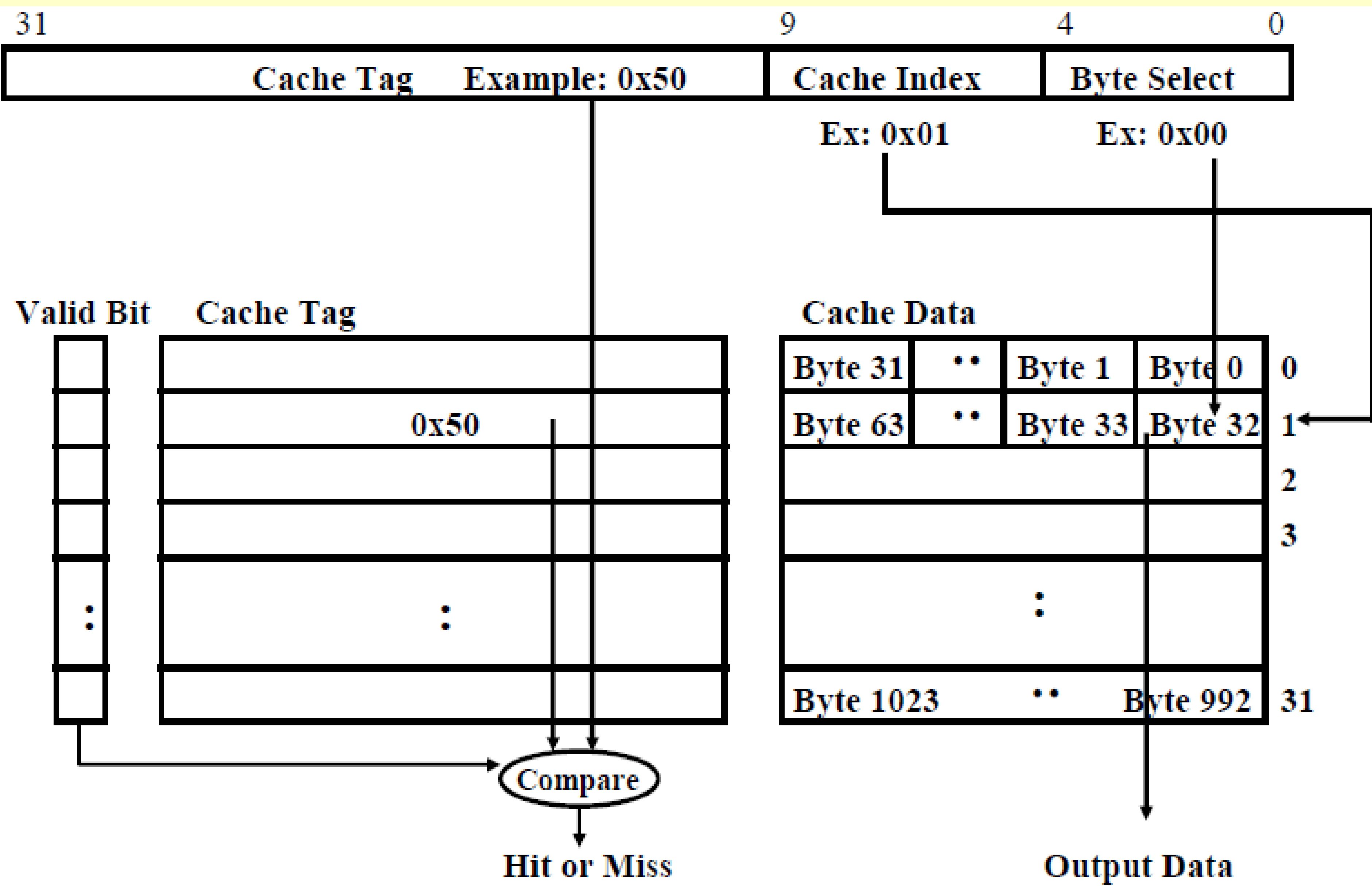
Obs:
- cele mai bune performante
- Complexitate
- MC <4KB

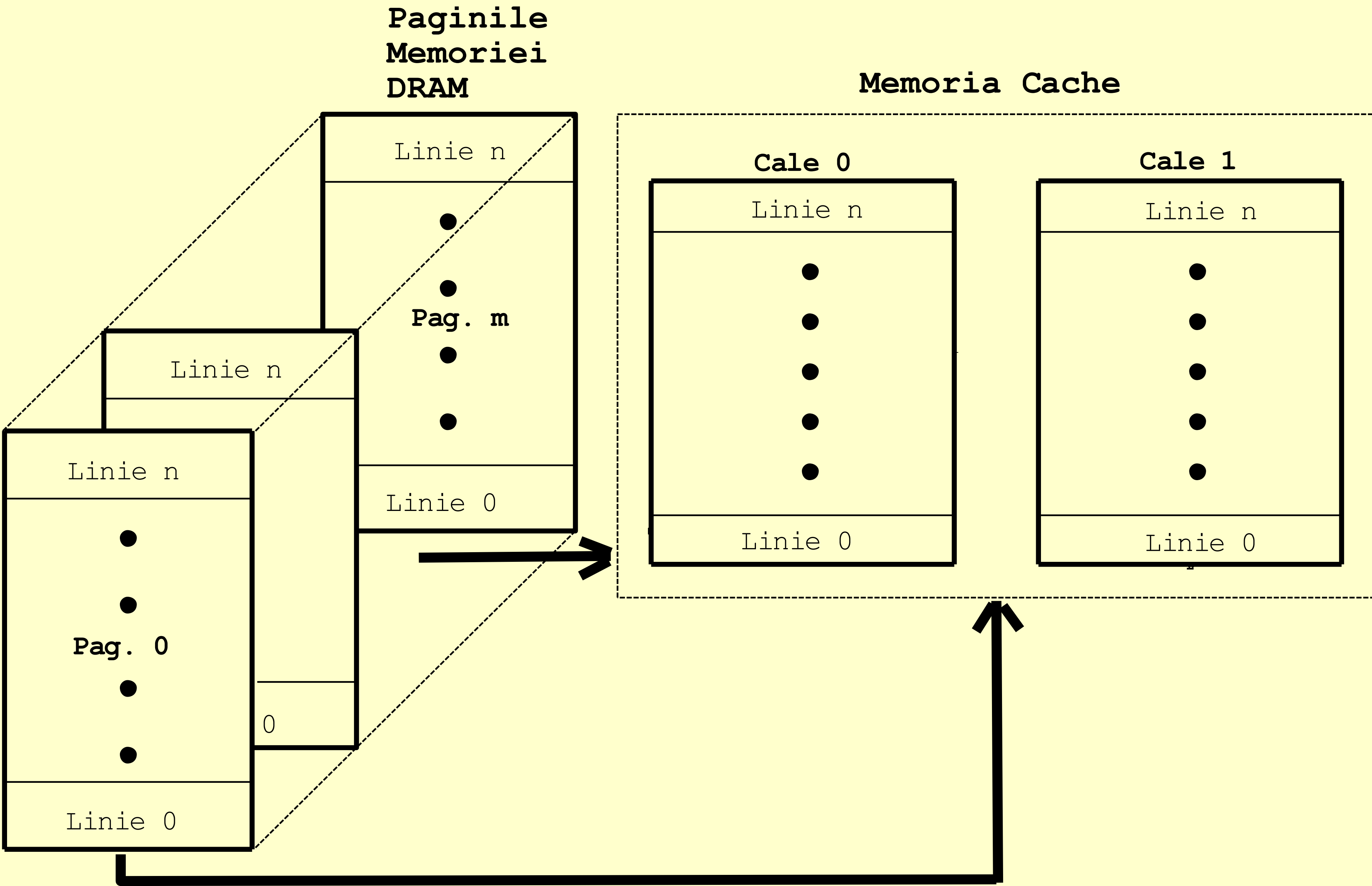


Memoria Cache cu mapare directa (direct mapped)

Obs. Simpla, ieftina, performante modeste

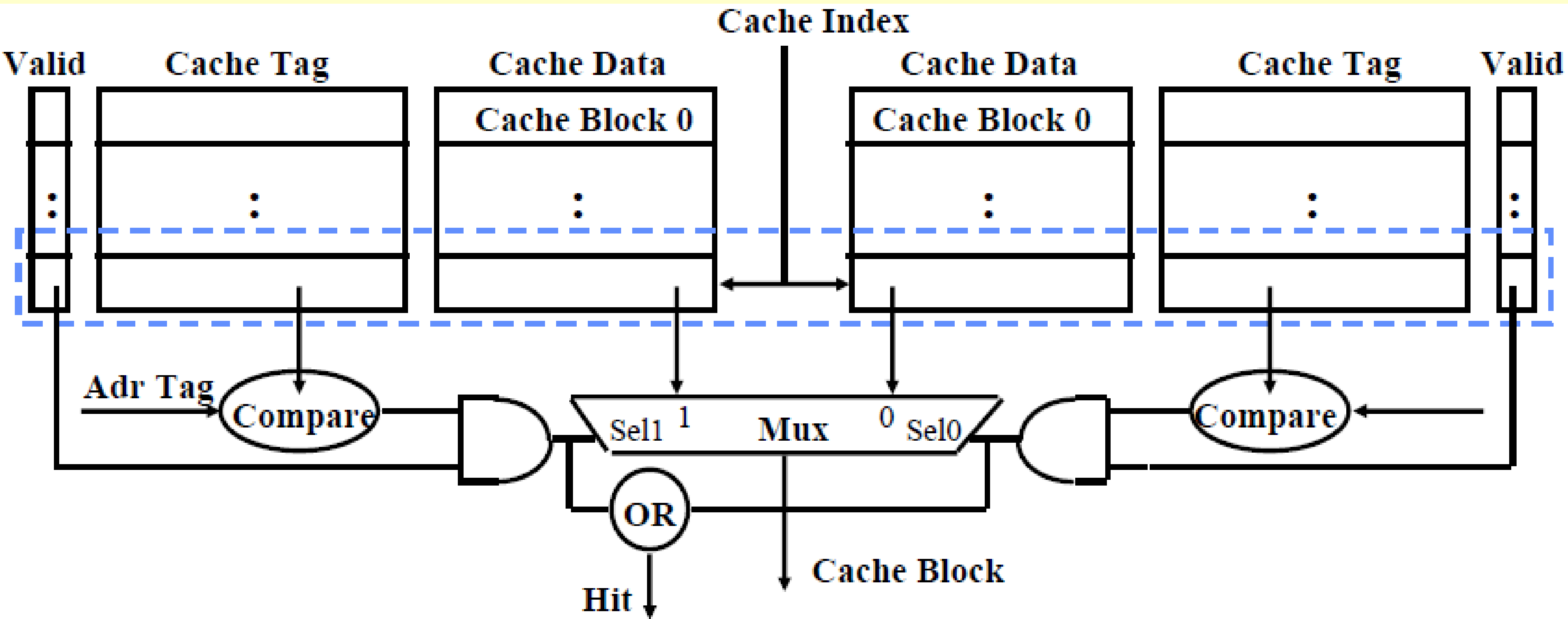
1 KB Direct Mapped Cache, 32B blocks





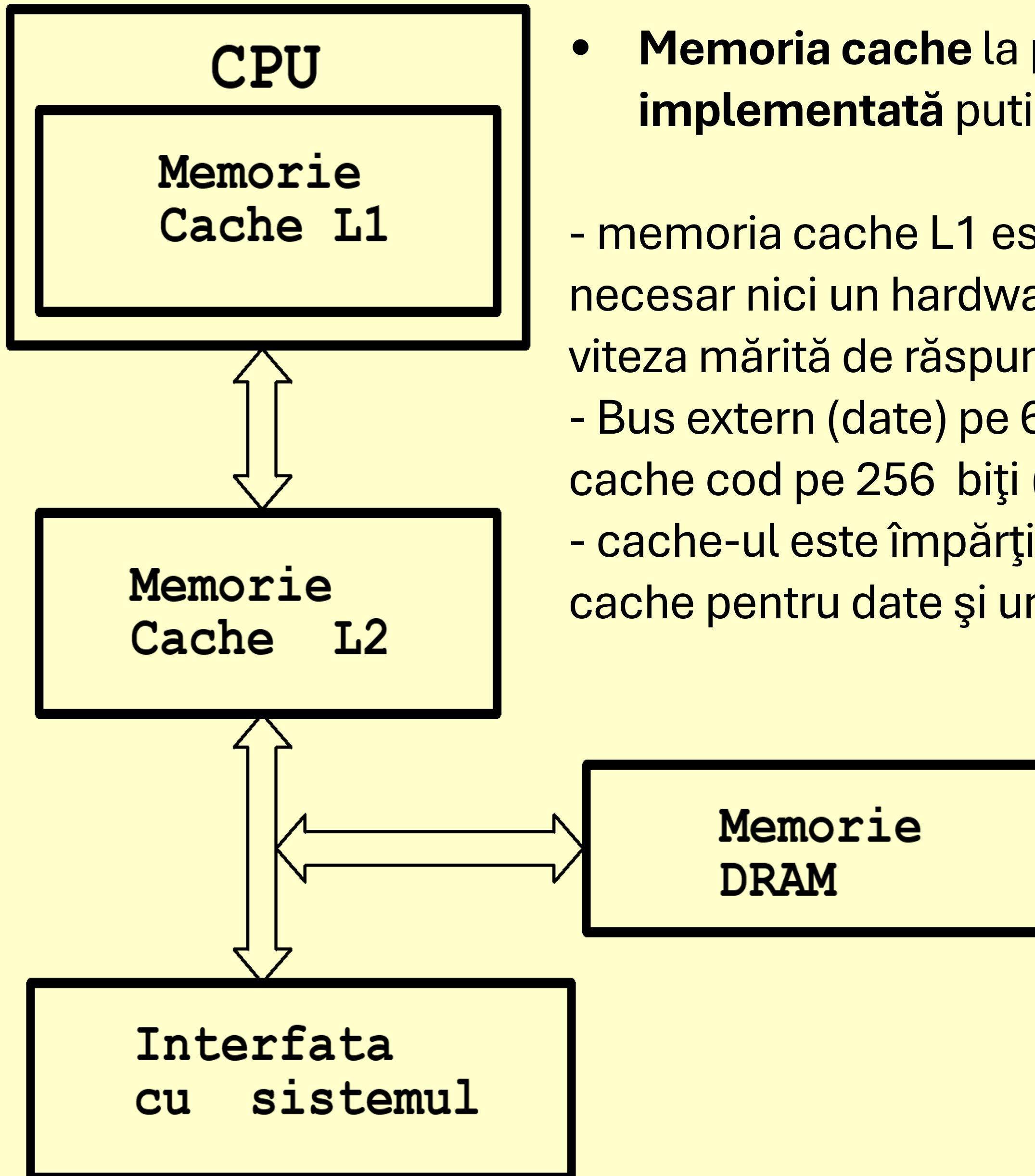
Memorie cache asociativa cu două căi (2 way set associative)

2-way Set Associative Caches



- Avantaj: 2 posibile locatii/bloc => rata hit mai buna
- Dezavantaj: mai lent datorita multiplexorului final

5. Memoria Cache la procesoarele Pentium



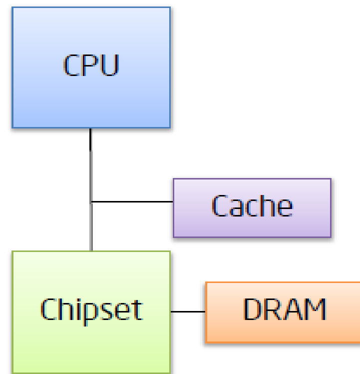
- **Memoria cache** la procesoarele **Pentium** este **implementată** putin **diferit** de principiile anterioare

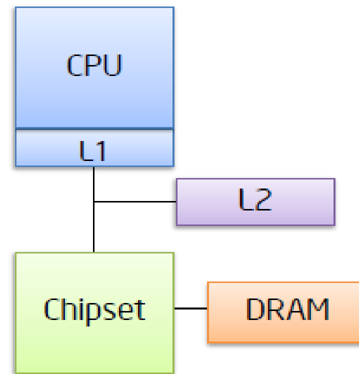
- memoria cache L1 este integrată on-chip (nu mai este necesar nici un hardware extern, reduce costul sistemului, viteza mărită de răspuns)
- Bus extern (date) pe 64 de biţii / bus-ul intern şi bufferul cache cod pe 256 biţi (64x4).
- cache-ul este împărţit în două componente separate un cache pentru date şi unul pentru cod

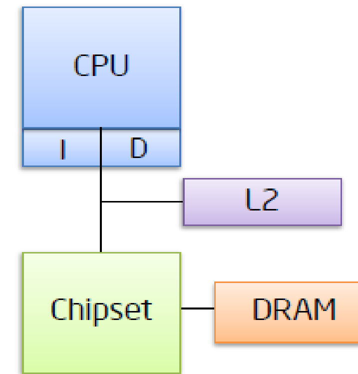
Procesor	Dimensiunea Cache
80486DX	8 Ko L1
Pentium	16 Ko L1
Pentium Pro	16 Ko L1; 256/512 Ko L2
Pentium II	32 Ko L1; 256/512 Ko L2
Pentium III	32 Ko L1; 256/512 Ko L2
Pentium 4 Pentium 4 EE	20 Ko L1; 512 Ko L2 + 2MB L3

Dimensiunea memoriei cache la diferite procesoare Intel

- Ambele memorii cache au structura unei memorii cache asociative, cu două căi.
- Mărimea liniei cache este 32 octeţi/256 biţi.
- O linie cache este încărcată printr-o serie de patru citiri pe bus-ul de date de 64 biţi
- Fiecare cale cache conţine 128 linii cache
- Mărimea paginii cache este de 4 Kocteţi sau 128 linii
- metoda de scriere în MC la procesorul Pentium poate fi controlată prin software prin CD – Cache Disable, NW (Not Write-Through), din registrul de control al procesorului, CR0







386

No on-die cache.
Level 1 cache
on motherboard

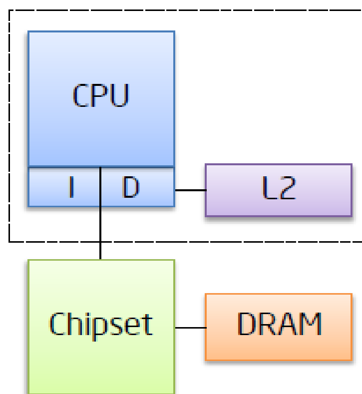
486

Level 1 cache on-die.
Level 2 cache
on motherboard

Pentium

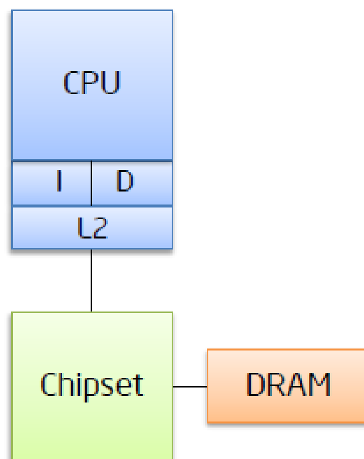
Separate Instruction
and Data Caches

Problema	Solutie	Processorul la care caracteristica apare ptr. prima data
Memorie externă mai lentă decât magistrala de sistem	Se adauga cache-ul extern utilizând o tehnologie de memorie mai rapidă.	386
Creșterea vitezei procesorului face ca bus-ul extern sa devina un obstacol pentru accesul în cache.	Se muta cache-ul extern pe chip, functionând la aceeasi viteză ca si procesorul.	486
Cache-ul intern este destul de mic, datorită spațiului limitat pe chip	Se adauga cache-ul extern L2 utilizând o tehnologie mai rapidă decât memoria principală	486
Atunci când atât prefetcher-ul de instructiuni, cât si unitatea de executie necesită simultan accesul la memoria cache apare conflict. În acest caz, prefetcher-ul este blocat în timp ce accesul la unitatea de executie are loc.	Se creaza memorii cache de instrucțiuni si date separate	Pentium



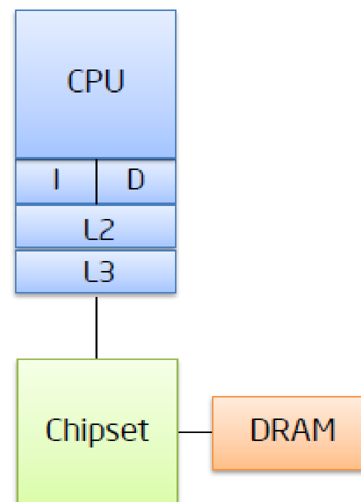
Pentium II

Separate bus to L2 cache in same package



Pentium III

L2 cache on-die



Core i7

L3 cache on-die

Problema	**Solutie**	**Processorul la care caracteristica apare ptr. prima data**
Cresterea vitezei procesorului are drept consecintă faptul că bus-ul extern devine un obstacol în calea accesului la cache-ul L2.	Se creaza un back-side bus separat care rulează la o viteză mai mare decât bus-ul principal extern(din fată) BSB este dedicat memoriei cache L2.	Pentium II
	Se muta L2 cache pe chip-ul procesorului	Pentium III
Unele aplicații se ocupă cu baze de date masive și trebuie să aibă acces rapid la cantități mari de date. Cache-urile pe chip sunt prea mici.	Se adauga L3 cache extern	Pentium III
	Se muta L3 cache on-chip.	Pentium 4

Numarul memoriilor Cache

- Inițial se folosește cache-ul unic extern și cu cresterea densitatii de integrare pe chipuri, a devenit disponibil spațiu pentru cache-ul on-chip.
- Sistemul beneficiază prin acestea de reducerea activității magistralei externe (accelerează timpul de execuție).
- Deoarece magistrala cache-ului on-chip este mai scurtă decât magistrala sistem, bus-ul on chip este mai rapid (este mai puțina întârziere).
- În această perioadă de utilizare a cache-ului on-chip, magistrala sistem va fi disponibilă pentru alte activități.

Este de dorit cache-ul extern?

- Da, această organizare se numește cache-ul de nivel doi.
- Cache-ul intern este L1, iar cache-ul extern este L2.
- Cu cache-ul L2 (SRAM), în caz de miss, procesorul trebuie să acceseze DRAM-ul sau ROM-ul direct prin magistrala sistem (lent și duce la scăderea performanței).
- Multe sisteme utilizează bus separat între MC L2 și procesor pentru a reduce sarcina pe magistrala sistem.
- Prin reducerea continuă a componentelor procesorului, multe sisteme localizeaza L2 pe chip procesor (îmbunătățirea performanței)

Cache unificat vs cache divizat

- La început, cache-ul L1 era utilizat atât pentru date, cât și pentru instrucțiuni.
- Apoi a devenit uzuală împărțirea memoriei cache într-una pentru date și una pentru instrucțiuni.
- Motivele cache-ului unificat sunt:

- Pentru o anumita mărime, va avea o rată de hit mai mare (ca rezultat al echilibrării automate intre date și instrucțiuni).

- Doar o singura memorie cache trebuie să fie implementată și proiectată.

- Cache-urile separate, elimină conflictul, în special pentru procesoarele superscalare (PowerPC și Pentium…), care accentuează instructiunile paralele și prefetch-ul instrucțiunilor viitoare. Acest lucru este foarte important pentru orice design care depinde de pipeline.

Cache unificat vs cache divizat

- Procesoarele de înaltă performanță au în mod invariabil 2 cache-uri L1 separate: cache-ul de instrucțiuni și cache-ul de date (I-cache și D-cache). Această "memorie cache" are mai multe avantaje față de un cache unificat:

- *Simplitatea cablării:* decodorul și planificatorul sunt legate numai de cache-ul de cod; registrele, ALU și FPU sunt legate doar de D-cache.

- *Viteza:* CPU-ul poate citi datele din cache-ul D, în timp ce încarcă instrucțiunile următoare din cache-ul cod. Sistemele multi-CPU au în mod obișnuit o memorie cache L1 I + L1 D pentru fiecare CPU, fiecare direct-mapped, pentru viteză.

- Pe de altă parte, într-un procesor de înaltă performanță*, alte nivele de memorie cache,* dacă există - L2, L3 etc. - precum și memoria principală - sunt de obicei unificate. Avantajele unui *cache unificat* (și o memorie principală unificată) sunt:

- Unele programe ruleaza cea mai mare parte a timpului într-o mică parte a programului care procesează o mulțime de date. Alte programe rulează o mulțime de subrutine diferite asupra unei cantități mici de date.
- Un *cache unificat echilibrează* automat proporția cache-ului folosit pentru instrucțiuni și cea pentru date - pentru a obține aceeași performanță într-o memorie cache divizată ar necesita o memorie cache mai mare.

- Când SO încarca un fișier executabil din spațiul de stocare - un cache divizat cere ca CPU să stearga și să reîncarce MC, in timp ce un cache unificat nu necesită acest lucru.

⚡ The Core Problem: Latency vs. Benefit

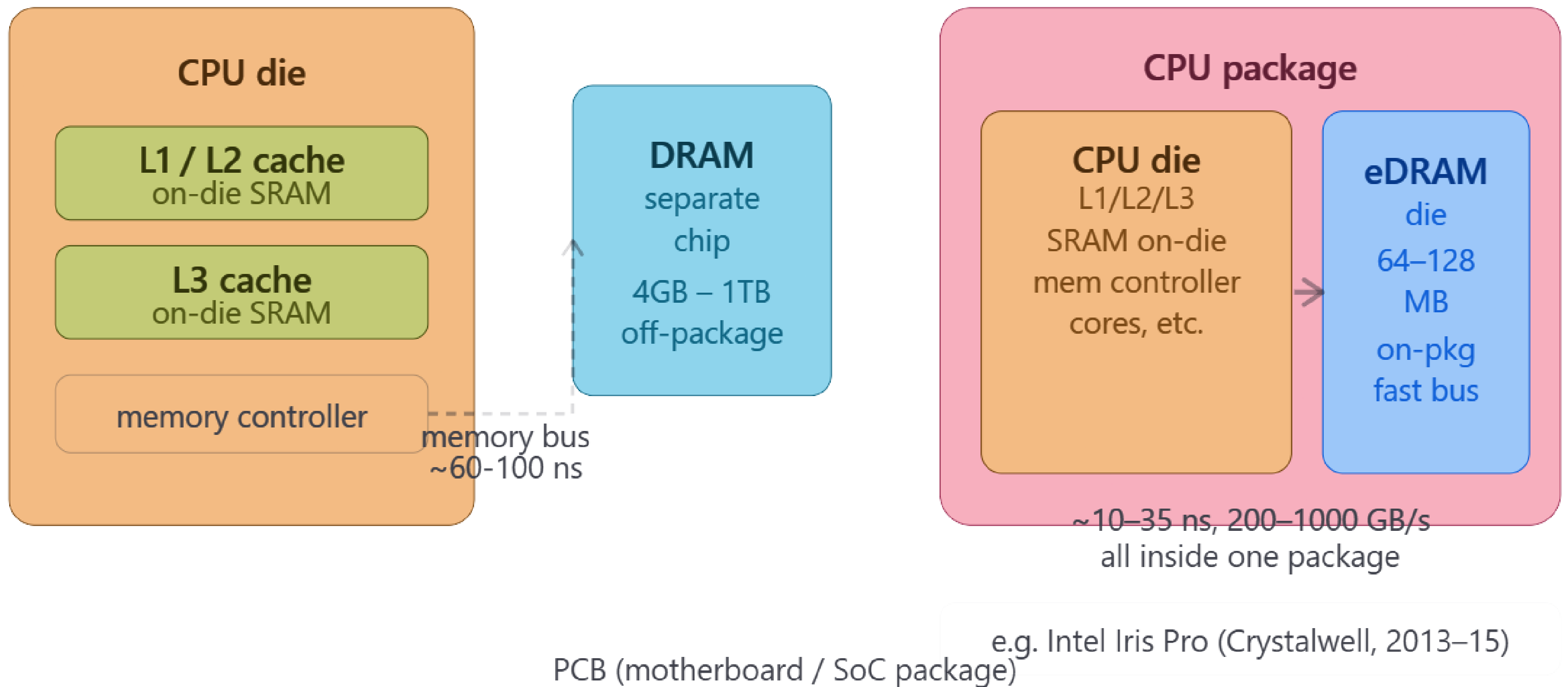
Cache Level	Location	Latency	Size
L1	Inside CPU core	~1-4 cycles	32-128 KB
L2	Inside CPU	~10-15 cycles	256 KB - 2 MB
L3	Shared on CPU die	~30-50 cycles	8-64 MB
L4	**Off-die (external)**	**~50-100+ cycles**	**64-128 MB**

Cache-ul L4 nu este utilizat pe scară largă din următoarele motive:

- Este prea lent pentru a oferi o îmbunătățire semnificativă a latenței față de memoria DRAM
- Cache-urile L3 au ajuns la o dimensiune suficient de mare pentru a acoperi majoritatea sarcinilor de lucru
- Costul ridicat pentru un câștig minim în ceea ce privește performanța procesorului
- Soluțiile moderne, precum cache-ul stivuit 3D, reprezintă o soluție mai bună pentru aceeași problemă

DRAM vs eDRAM — side-by-side

	DRAM	eDRAM
Location	**Separate chip on PCB** Off-package, standalone module	**On the CPU package** Integrated die or stacked die
Latency	**~60–100 ns** High — long signal path to CPU	**~10–35 ns** Low — very short on-package path
Bandwidth	**25–100 GB/s (DDR5)** Limited by narrow bus width	**200–1000+ GB/s** Wide internal bus, dense traces
Capacity	**4 GB – 1+ TB** Easily expandable via slots	**64 MB – a few GB** Fixed at manufacture, not upgradable
Cost per GB	**~$3–10 / GB** Commodity, mass produced	**~$50–200+ / GB** Premium — complex integration



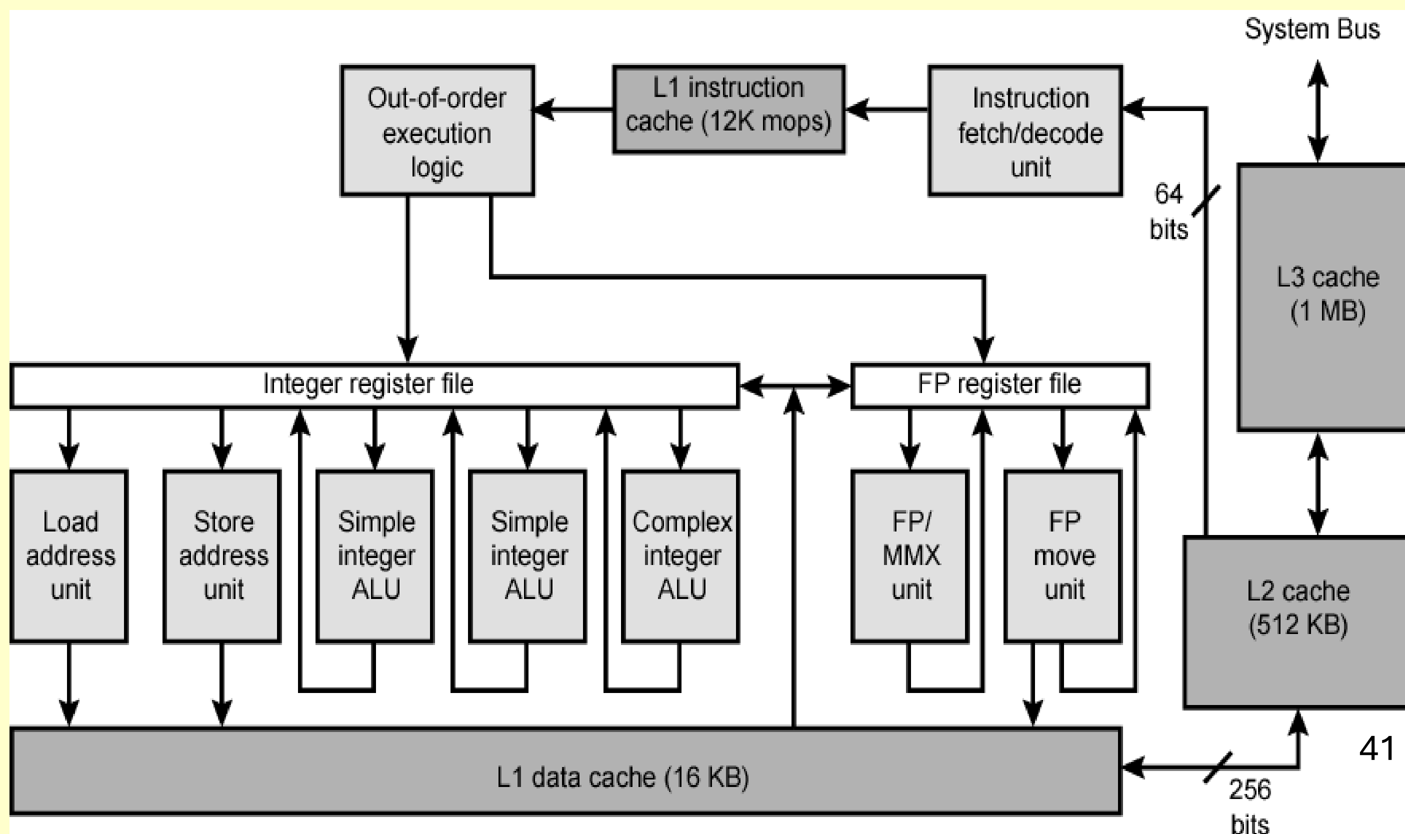
- RAM (DDR4, DDR5, LPDDR5) este memoria principală standard în aproape orice computer. Este ieftină, scalabilă și poate fi actualizată — însă semnalul trebuie să traverseze placa de circuit imprimat (PCB) pentru a ajunge la procesor (CPU), ceea ce crește latența și limitează lățimea de bandă.eDRAM (DRAM încorporată) integrează memoria în același pachet cu procesorul, reducând considerabil traseul semnalului.
- Rezultatul este o latență de 3–10 ori mai mică și o lățime de bandă de 5–20 ori mai mare decât în cazul DRAM-ului obișnuit. Dezavantajul este că este scumpă și fixă — nu se poate adăuga mai multă.De ce eDRAM-ul este important pentru anumite sarcini de lucru:

- GPU-urile și plăcile grafice integrate sunt cele mai mari beneficiare — acestea au nevoie de lățime de bandă mare, iar eDRAM-ul le alimentează mult mai rapid
- Sarcinile generale ale CPU înregistrează câștiguri modeste (~5–15%), deoarece cache-ul L3 absoarbe deja majoritatea cererilor de memorie
- Cipurile din seria M de la Apple utilizează un concept similar — un pool mare de memorie unificată cu lățime de bandă mare (LPDDR5X) strâns integrat pe pachet — motiv pentru care cifrele lor privind lățimea de bandă a memoriei (~200–800 GB/s) depășesc cu mult memoria tradițională a PC-urilor
- eDRAM este, în esență, un cache foarte mare și foarte rapid, situat între L3 și DRAM-ul principal. Este instrumentul potrivit pentru sarcini care necesită lățime de bandă mare (grafică, inferență ML), dar este prea costisitor și prea mic pentru a înlocui DRAM-ul în totalitate.

Pentium 4 Processor

- **Fetch/Decode Unit**
  - Citeste instructiunile din L2 cache
  - Decodeaza in micro-ops
  - Stocheaza micro-ops in L1 cache
- **Out of order execution logic**
  - Planifica micro-ops
  - Bazat pe dependenta datelor si a resurselor
  - Se pot executa speculativ

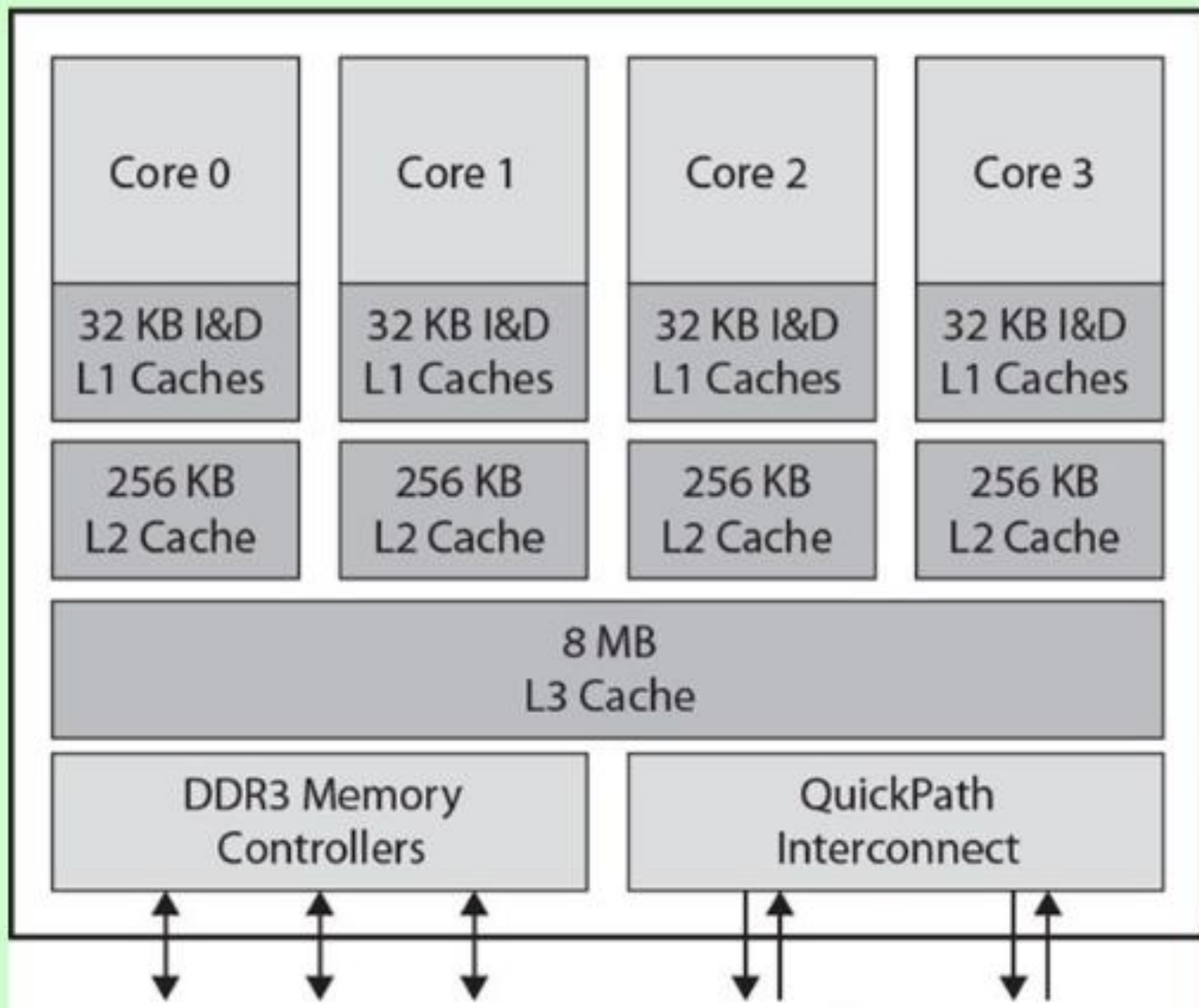
- **Execution units**
  - Executa micro-ops
  - Datele din L1 cache
  - Rezultatele in registre
- **Memory subsystem**
  - L2 cache si bus sistem



Pentium 4 Design Reasoning

- Decodeaza instructiunile in RISC like micro-ops inainte de L1 cache
- Micro-ops au lungime fixa
  — *Pipeline Superscalar si planificare*
- Performanta imbunatatita separand decodarea de planificare & pipelining
- Data cache este write-back
  — Poate fi configurata write-through
- L1 cache controlat de 2 biti in registrul CR0
  — CD = cache disable
  — NW = not write through
  — 2 instructiuni de invalidare (flush) cache si write-back apoi invalidare
- L2 si L3 cache sunt 8-way set-associative
  — Line size 128 bytes

Intel core i7 block diagram



6. Identificarea caracteristicilor memoriei cache

- Instrucţiunea CPUID poate returna şi informaţii despre mărimea şi caracteristicile MC interne EAX= 2, instrucţiunea CPUID încarcă regiştrii EAX, EBX, ECX şi EDX cu descriptori care indică caracteristicile cache-ului procesorului şi caracteristicile TLB.

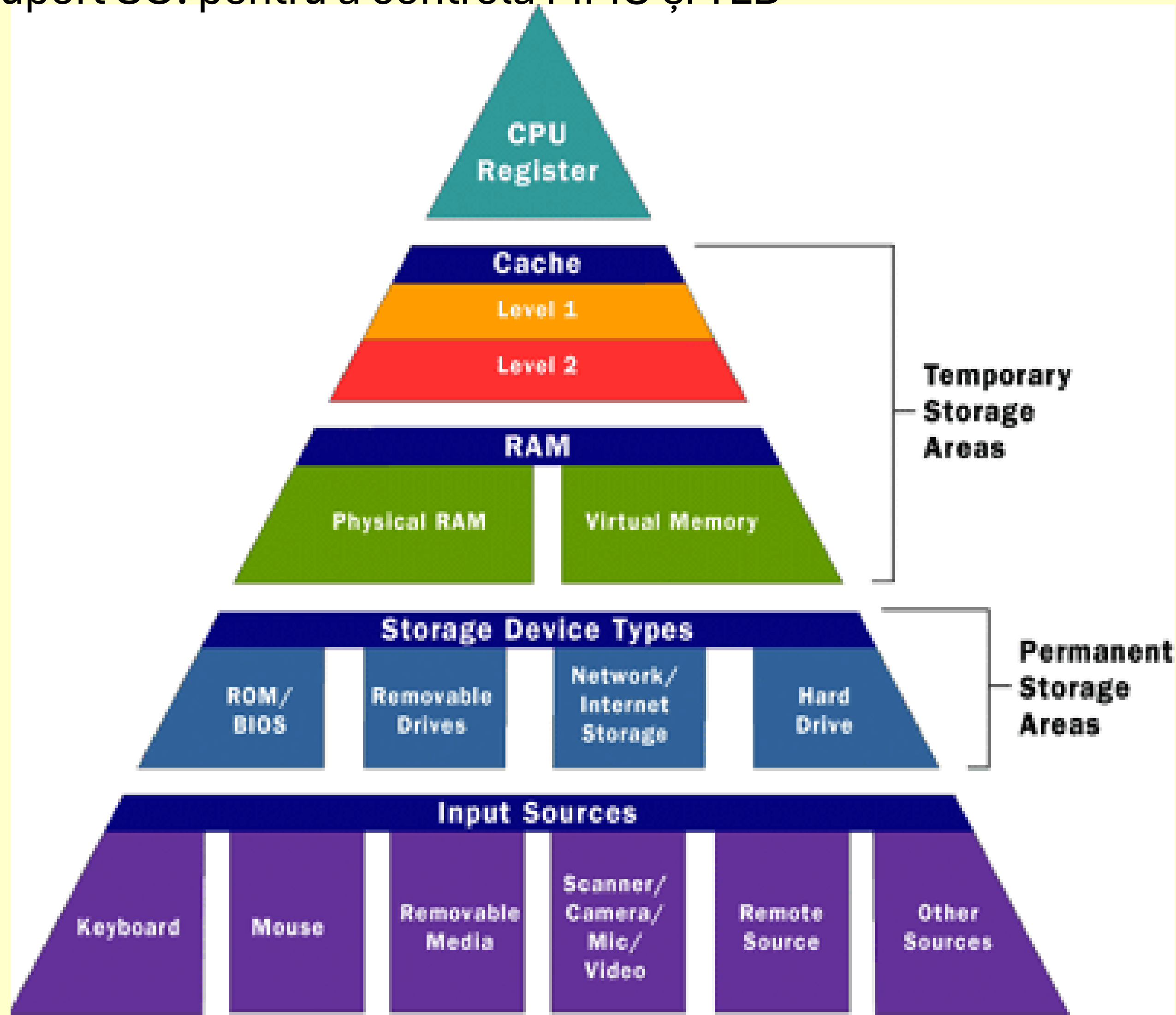
Reg. Biti	31-24	23-16	15-8	7-0
EAX	66h	5Bh	50h	01h
EBX	00h	00h	00h	00h
ECX	00h	00h	00h	00h
EDX	00h	7Ah	70h	40h

Descriptorii returnati de instructiunea CPUID la un procesor P4 (pentru EAX=2)

- (66h) Cache date, 8Ko, 4 căi asociativă, linie cache de 64 octeţi
- (5Bh) TLB date care mapează pagini de 4Ko/ 4Mo, total asociativă, cu 64 de intrări
- (50h) TLB instrucţiuni care mapează pagini de 4Ko/2Mo/4Mo, total asociativă, cu 64 intrări
- (7Ah) Cache unificat de 256Ko, asociativă cu 8 căi, linie cache de 64 octeţi
- (70h) Trace cache de instrucţini care poate stoca până la 12K-µOps, asociativă cu 8 căi
- (40h) Fără cache L3

7. Memoria virtuala - MV

- Abstractizare de bază oferita de SO pentru gestionarea memoriei
- MV necesită atât sprijin hardware cat și a SO
  - suport hardware : unitate de management al memoriei (MMU) și tampon (TLB)
  - suport SO: pentru a controla MMU și TLB



Motivații pentru memoria virtuala

1. DRAM-ul este utilizat ca un cache pentru hard disc (HDD)

- Spatiul de adrese al unui proces poate depăși mărimea fizică DRAM
- Suma dimensiunilor proceselor poate depăși mărimea DRAM

2. Simplifica managementul memoriei

- Procese multiple pot fi rezidente în memoria principală
    - Fiecare proces cu propriul spațiu de adrese
- Numai codul și datele " **active**" sunt de fapt în memorie
    - Se aloca mai multă memorie la procese după cum este necesar

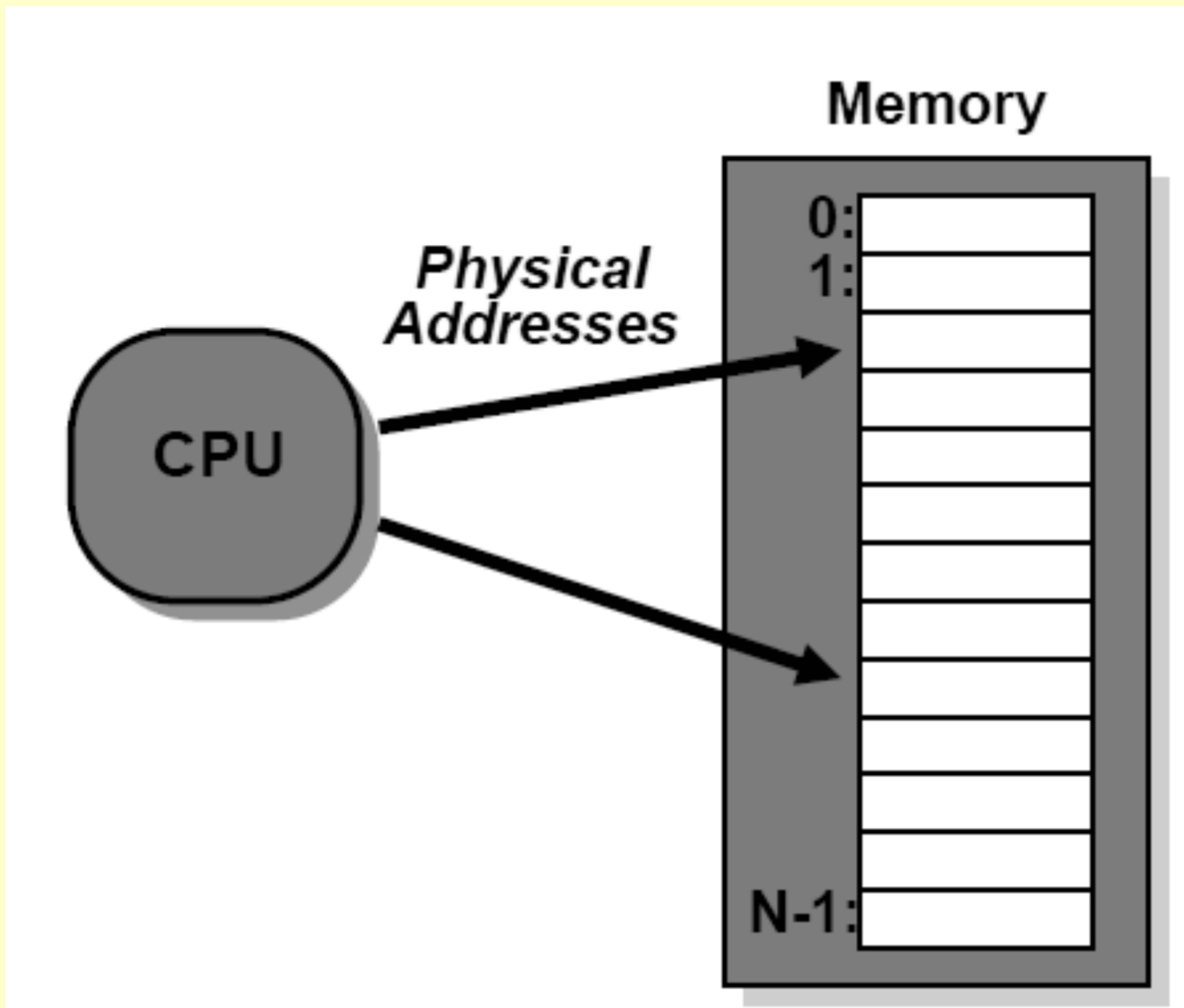
3. Ofera protecție a proceselor

- Un proces nu poate interfera cu altul, deoarece acestea operează în spații diferite de adrese
- User-ul unui proces nu poate avea acces la informații privilegiate
    - diferite secțiuni ale spațiilor de adrese au diferite permisiuni

https://www.slideshare.net/emery/virtual-memory-and-paging

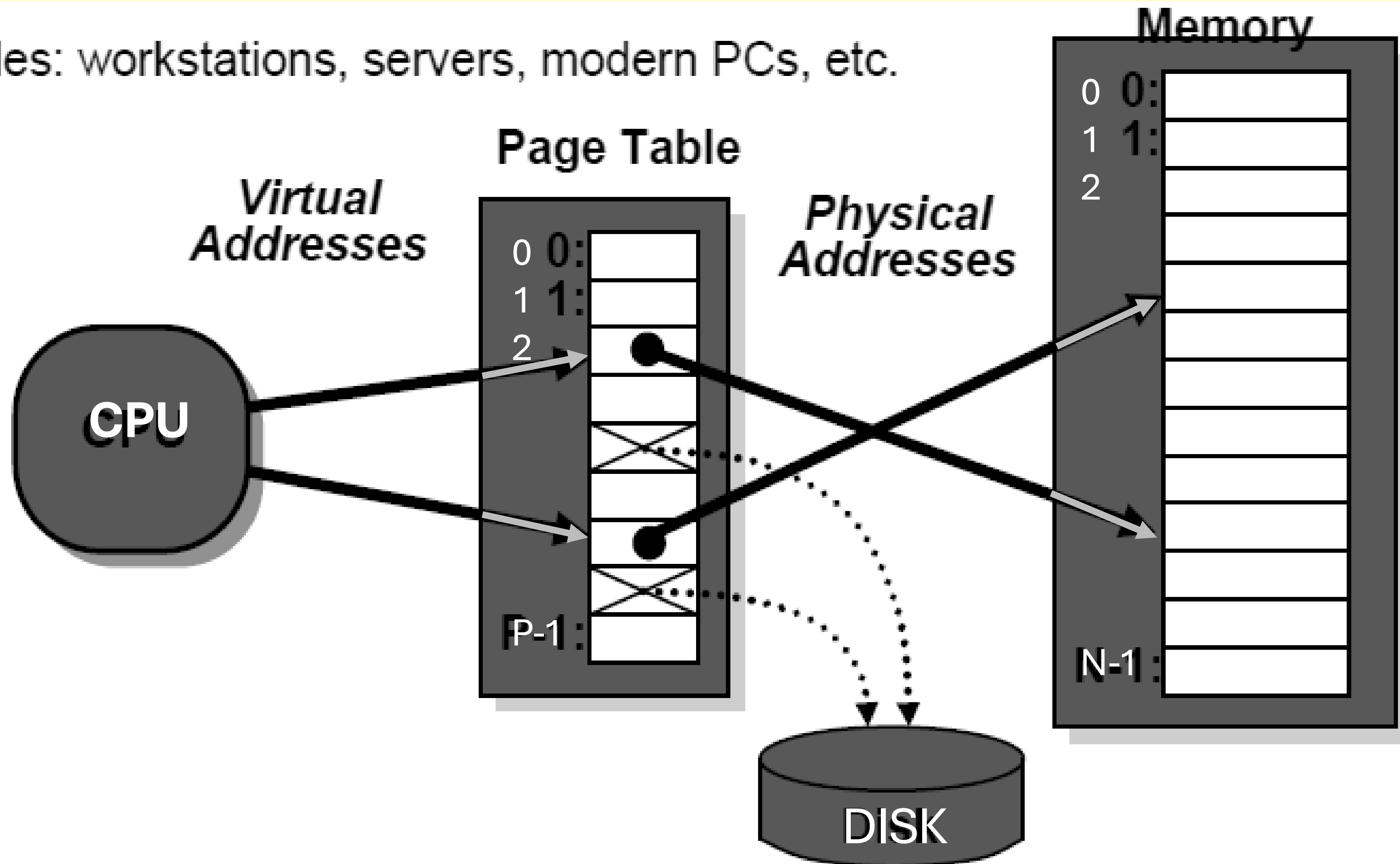
Sistem numai cu memorie fizica

Ex: PC/sisteme vechi, aproape toate sistemele embedded



- Adresele generate de CPU acceseaza bytes direct in memoria fizica

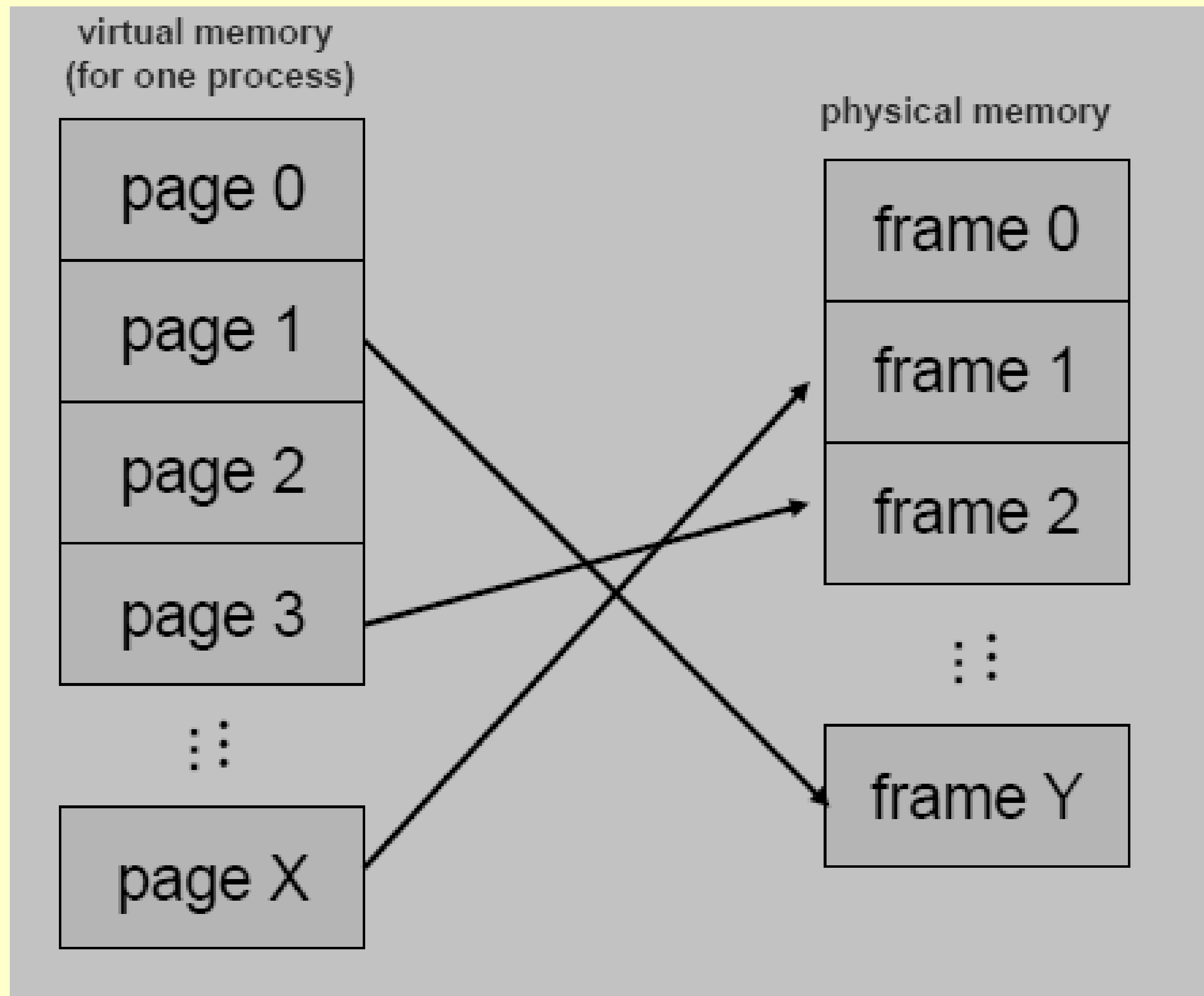
Sisteme cu Memorie Virtuala



Translatarea Adreselor: Hardware-ul converteste adresa *virtuala in adresa fizica* printr-un lookup table (*page table*) controlat de SO

Paginarea

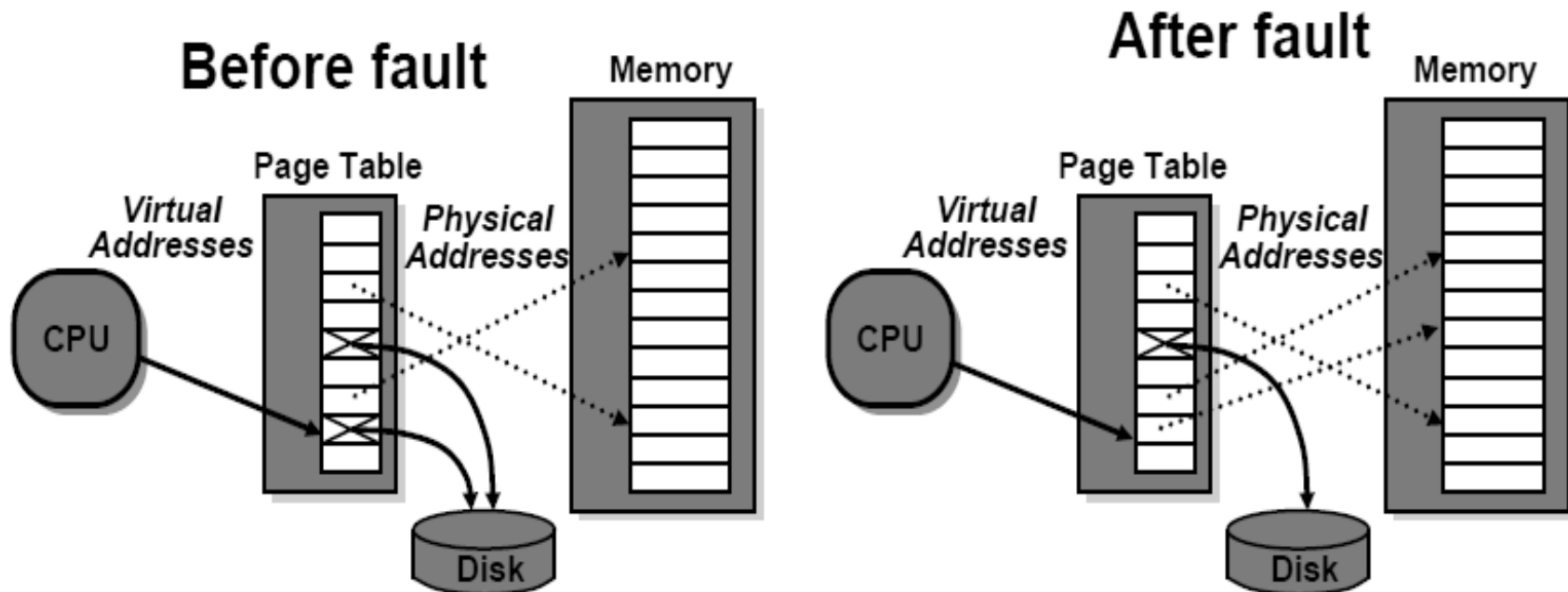
- Unitatea de memorie virtuala este numita pagina (*page)*
- Unitatea de memorie fizica este numita cadru(*frame)* sau (*page frame*)



https://slideplayer.com/slide/2503436/

Page Fault (similar cu “Cache Miss”)

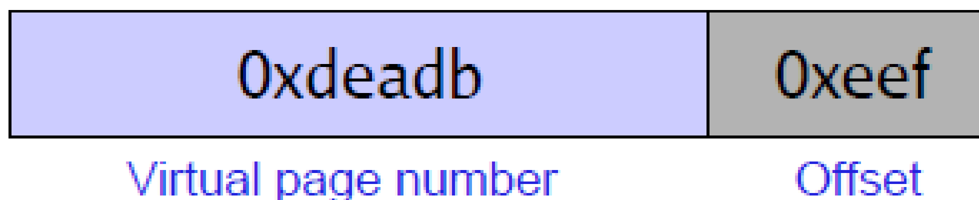
- Ce se intampla daca o pagina este mai degraba pe disk decat in memorie?
– Intrarea in ‘’Page table’’ indica ca adresa virtuala nu este in memorie
– handler-ul de exceptii al SO cere mutarea datelor de pe disk in memorie
- procesul curent se suspenda, celelalte pot fi reluate
- SO are control complet asupra plasarii datelor etc.



Translatarea adreselor virtuale

- Translatarea adreselor virtuale la adrese fizice este efectuata de MMU
- *Adresa Virtuala* este divizata intr-un *virtual page number* si un *offset*
- Maparea paginilor virtuale la pagini fizice este facuta prin “*page table*”

0xdeadbeef =



virtual address

virtual page # | offset

0xeef

0xdeadb

page table

page frame #

physical address

page frame # | offset

Page table entry

physical memory

page frame 0

page frame 1

page frame 2

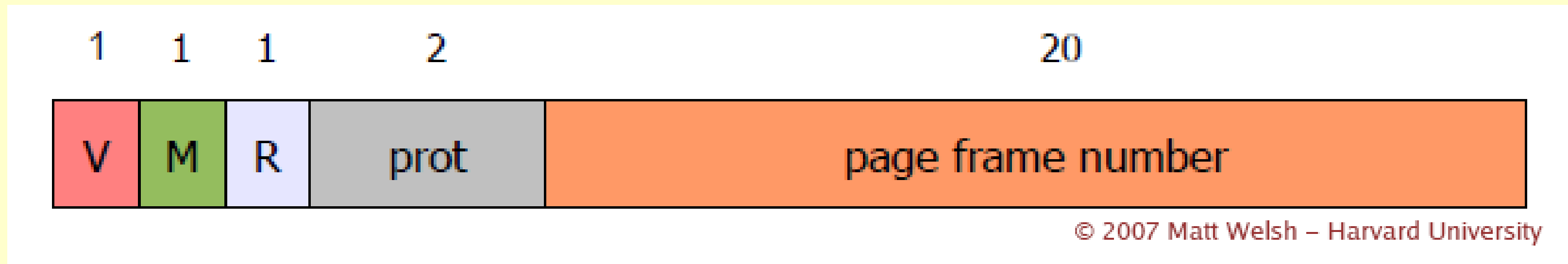
page frame 3

page frame Y

© 2007 Matt Welsh – Harvard University

Intrari in Tabela paginilor/ Page Table Entries (PTEs)

- *Formatul tipic al PTE depinde de arhitectura CPU !*



- *Diversi biti sunt accesati de MMU la fiecare acces de pagina:*

- Valid bit (V): dacă pagina corespunzătoare este în memorie
- Modify bit (M): indică dacă o pagină este "murdara" (modificata)
- Reference bit (R): Indică dacă o pagină a fost accesata (read /write)
- Protection bits: Specifica daca o pag. poate fi citita, scrisa sau executata
- Page frame number: locatia fizica a paginii in DRAM

Tabelele de pagini/page tables

- stocheaza maparea adreselor virtuale-la-adrese fizice

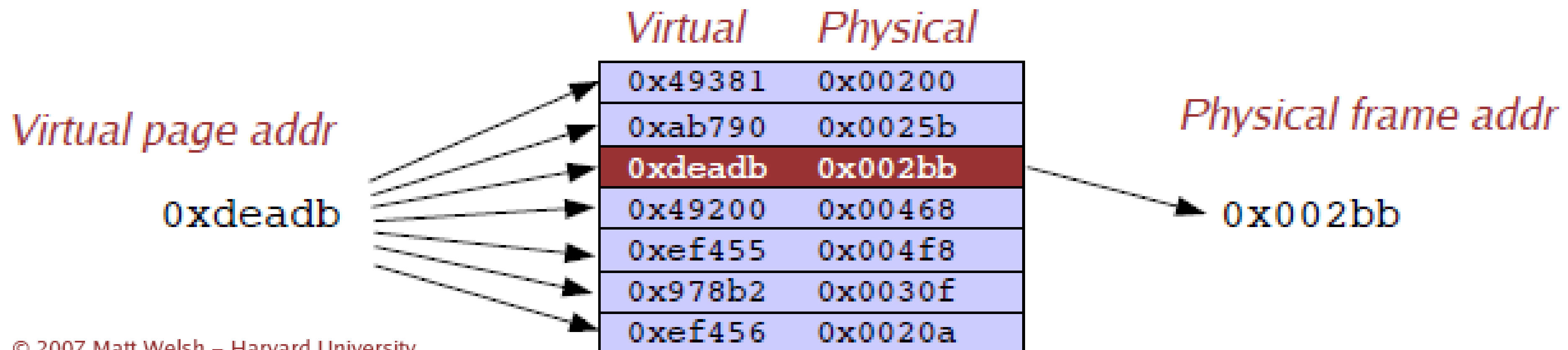
  - ✓ Fiecare proces are o tabela de paginare
  - ✓ Fiecare proces are un spatiu de adresare separat ptr. protectie
  - ✓ Dar, mai multe pagini pot accesa aceiasi adresa fizica!
  - ✓ *Tabelele de pag. sunt stocate in memorie !*
  - ✓ Unitatea MMU are un registru special numit ***page table base pointer***
  - ✓ Acesta pointeaza adresa de memorie fizica din varful tabelei de pagini/page table pentru procesul curent in executie.

● La fiecare acces la memorie, trebuie sa avem un acces separat ptr. consultarea tabelei de pagini !

Solutie: *Translation Lookaside Buffer (TLB)*

Translation Lookaside Buffer (TLB)

- Este un cache rapid, dar mic, direct pe CPU
- *Sistemele Pentium 6 au TLB separate ptr. date si instructiuni, cu 64 intrari fiecare*
- *Pastreaza* cele mai recente translatari de adr. virtuale la adr. fizice
- TLB sunt implementate ca fully associative cache
- Orice adresă poate fi stocata în oricare intrare din cache
- Toate înregistrările sunt căutate " în paralel " la fiecare translatie de adrese
- Un miss in TLB, cere de fapt ca MMU sa încerce să facă translarea de adr.

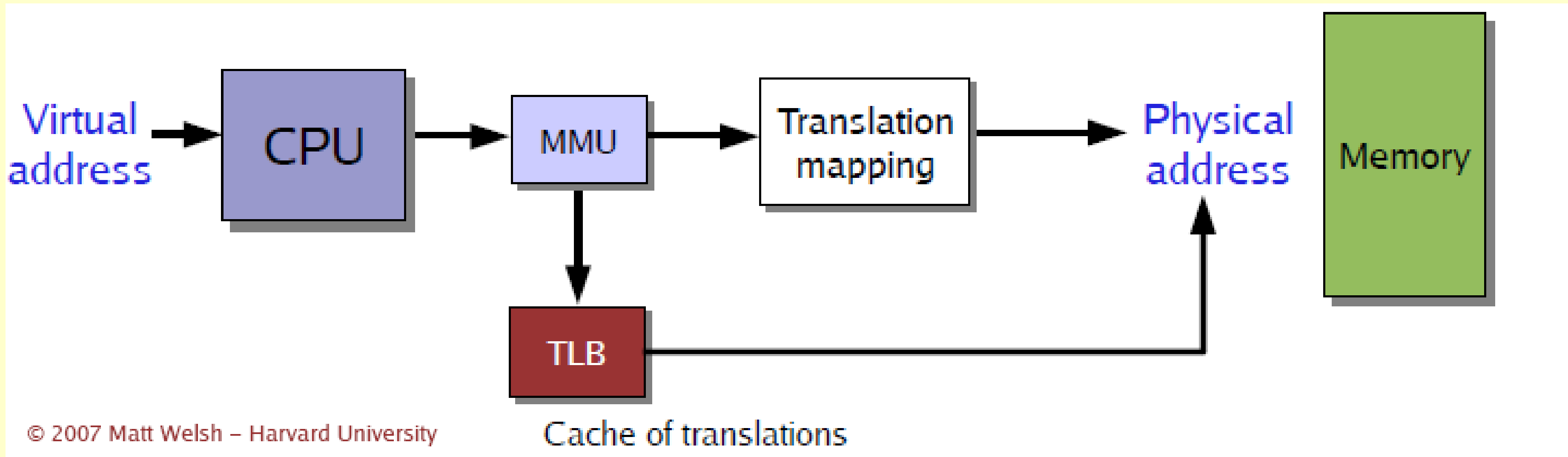


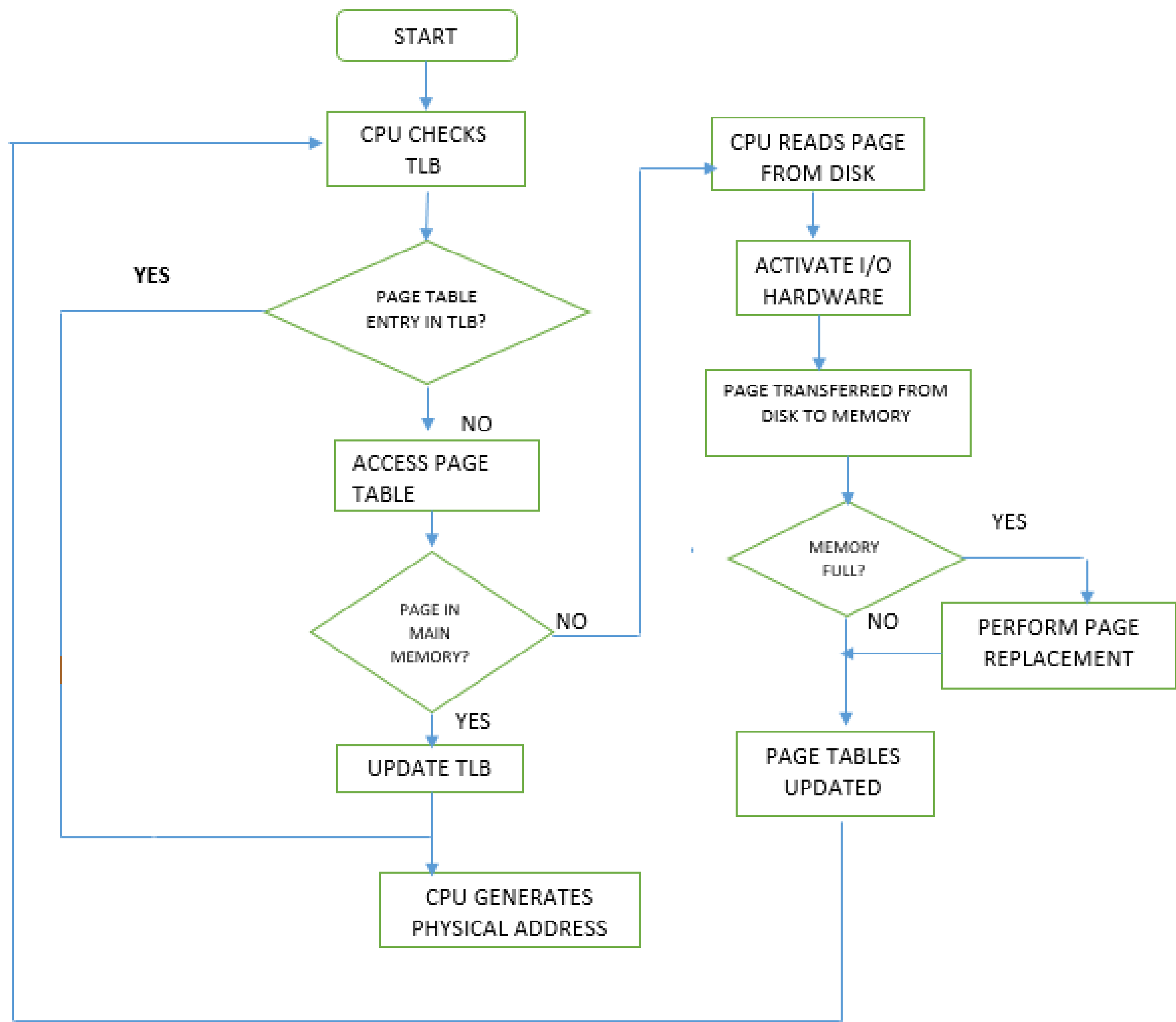
- **Memory Management Unit (MMU)**

- Hardware-ul care translateaza o adr. virtuala la o adr. fizică
- Fiecare referire la memorie este trecuta prin MMU

- **Translation Lookaside Buffer (TLB)**

- TLB=Cache-ul MMU ptr. translatarea adreselor virtuale-la- adrese fizice mai rapid
- Este o optimizare importanta!





Flowchart shows the working of a TLB. For simplicity, the page-fault routine is not mentioned.

Stallings, William (2014). Operating Systems: Internals and Design Principles. United States of America: Pearson. ISBN 978-0133805918.

Avantajele și dezavantajele sistemelor de memorie virtuală

- ***Avantajul*** *principal al sistemelor de memorie virtuală este posibilitatea de a încărca și executa un proces care necesita o cantitate mai mare de memorie fizica decât cea disponibila, prin încărcarea procesul pe părți și apoi se executa*
- *Avantajul constă în capacitatea sistemului de a elimina fragmentarea externă*
- Poate reloca programul in timp ce ruleaza

- ***Dezavantajul*** este că sistemele de memorie virtuală tind să fie lente și necesită sprijin suplimentar de la hardware-ul complex al sistemului de translatare al adreselor.
- Viteza de executie a unui proces într-un sistem de memorie virtuală poate egala, dar nu depășesc niciodată, viteza de executie a aceluiaș proces cu Virtual Memory oprit.
- Accesele frecvente la hard-disk scurteaza durata de viata a dispozitivului

Difference Between SRAM and DRAM

Parameters	SRAM	DRAM
Power Supply to Retain Data	Requires a consistent power supply	Needs electrical refresh periodically
Construction	A flip-flop circuit consisting of 6 transistors	A single capacitor and a transistor
Structure and Design Complexity	Complex	Simple
Power Consumption	Low	High
Storage Capacity	Lower storage capacity (1 MB to 16 MB)	High storage capacity (4 GB to 16 GB)
Speed/Access Time	Faster	Slower
Number of Transistors	6	1
Charge Leakage	No	Yes, if not refreshed regularly
Cost	Expensive	Less expensive
Application	Used in the L2 and L3 cache memory.	Used in a computer's primary memory

Modern Cache Hierarchy: Intel Core & AMD Ryzen

►**Intel 13th Gen (Raptor Lake) — per core:** L1-I: 32 KB, L1-D: 48 KB, L2: 2 MB; shared L3: up to 36 MB

►**AMD Ryzen 7000 (Zen 4) — per core:** L1-I: 32 KB, L1-D: 32 KB, L2: 1 MB; shared L3: up to 96 MB (with 3D V-Cache stacking)

►**Standard cache line size:** 64 bytes on all modern x86 CPUs (since Pentium 4). Pentium I/II used 32-byte lines.

►**L3 cache architecture:** Shared across all cores; 8 to 16-way set-associative; can be inclusive or exclusive (AMD)

►**Typical access latencies (CPU cycles):** L1: 4-5 cycles, L2: 12-14 cycles, L3: 30-50 cycles, DRAM: 200-300 cycles

►**L4 / eDRAM (some Intel designs):** Intel Iris Pro (Haswell/Broadwell) added 128 MB embedded DRAM as L4 for GPU/CPU bandwidth

Cache Coherence: The MESI Protocol

►**Why coherence matters:** In multi-core CPUs, each core has a private L1/L2 cache. If Core 0 writes to address X, Core 1 must not read a stale copy - cache coherence protocols enforce consistency.

►**MESI — 4 states per cache line:** Modified (dirty, exclusive owner), Exclusive (clean, only copy), Shared (clean, multiple copies), Invalid (line not valid)

►**Modified (M):** this core wrote Line; other caches must not have this line. Must write back to RAM before another core can read it.

►**Exclusive (E):** Only copy in all caches; identical to main memory. Can silently transition to M on a write.

►**Shared (S):** Multiple caches hold this line (all identical to RAM). Cannot write without first broadcasting an Invalidate.

►**Invalid (I):** This cache line is stale or absent. Next access will cause a cache miss and a coherence fetch.

►**MESIF / MOESI extensions:** Intel uses MESIF (adds Forward state for peer-to-peer sharing). AMD uses MOESI (adds Owned state to avoid write-back).

Hardware Pre-fetching: hiding memory latency

►**The latency problem:** A cache miss stalls the pipeline for 100s of cycles (DRAM latency). Pre-fetching loads data before it is needed, hiding this latency.

►**Stream pre-fetcher:** Detects sequential or strided access patterns (e.g., walking an array) and pre-fetches the next cache line(s) ahead of time. Present in all modern Intel and AMD CPUs.

►**Stride pre-fetcher:** Detects accesses with a fixed stride (e.g., accessing every 4th element). Pre-fetches future elements at the detected stride offset.

►**Spatial pre-fetcher (Intel):** Completes a 128-byte aligned pair of cache lines whenever one of the two is fetched — exploits spatial locality at the hardware level.

►**Software pre-fetch (PREFETCHNT / PREFETCHT0 instructions):** Programmer or compiler inserts pre-fetch hints ahead of known data loads. Allows fine-grained control but risks polluting the cache if used incorrectly.

►**Non-temporal stores (MOVNTPS/MOVNTDQ):** Write data directly to memory, bypassing the cache hierarchy (write-combining buffer). Ideal for streaming writes that will never be re-read.

►**Pre-fetch limitations:** Irregular access patterns (pointer chasing, graph traversal) defeat hardware pre-fetchers. Software restructuring (blocking, tiling) is then required.

Cache-Friendly Programming: Optimization Techniques

►**The core principle:** Data accessed together should reside together in memory. Maximize spatial and temporal locality to increase hit rate and minimize miss penalties.

►**Loop tiling / blocking:** For matrix operations, process sub-blocks that fit in L1/L2 cache. Classic example: matrix multiply tiling reduces L2 misses from O(n^3) to O(n^3 / B) where B is the block size.

►**Structure of Arrays (SoA) vs Array of Structures (AoS):** AoS: {x,y,z,w, x,y,z,w,...} — loads 4 fields per cache line even when only one is needed. SoA: {xxxx..., yyyy...} — all x values packed together, better for SIMD and partial access.

►**False sharing in multi-threaded code:** Two threads writing to different variables that share the same 64-byte cache line cause constant MESI invalidations (ping-pong effect). Solution: pad structures to 64-byte boundaries.

►**Cache-oblivious algorithms:** Algorithms like van Emde Boas trees and recursive matrix multiply achieve optimal cache performance at all levels of the hierarchy without knowing cache sizes.

►**Profiling tools:** Linux perf, Intel VTune, AMD uProf, Valgrind/Cachegrind — measure L1/L2/L3 miss rates, identify hotspots, and guide cache optimization efforts.

►**Rule of thumb — 90/10 cache rule:** 90% of execution time is spent in 10% of the code. Identify that 10%, analyze its access patterns, and optimize memory layout for cache efficiency.