

# Jumătatea a 2-a a semestrului

- curs
  - structura hardware a unui sistem de calcul
  - finalizare - test scris (săptămâna a 8-a)
    - condiție susținere test - cel mult 2 absențe la laborator
- laborator
  - limbaj de asamblare
  - finalizare - test practic (săptămâna a 7-a)

# Cuprins

- I. Structura unui sistem de calcul
- II. Memoria
- III. Unitatea centrală de procesare (CPU)
- IV. Îmbunătățirea performanței CPU
- V. Pipeline
- VI. Sistemul de întreruperi
- VII. Sistemul de operare

# I. Structura unui sistem de calcul

# Componentele principale

- Unitatea centrală de procesare (CPU)
- Memoria
- Dispozitive periferice (I/O = *input/output*)
- Magistrale
  - de date
  - de adrese
  - de control

# Unitatea centrală de procesare

- numită și procesor
- execută instrucțiunile indicate de programator
- realizează prelucrarea datelor
- coordonează funcționarea celorlalte componente

# Memoria

- stocarea informațiilor
  - date
  - instrucțiuni
- furnizarea informațiilor la cerere
- rol pasiv
  - "răspunde" la cererile venite din exterior
  - nu inițiază niciodată un transfer

# Dispozitivele periferice

- comunicarea cu exteriorul
- funcții - foarte variate
  - preluare date
  - afișare
  - imprimare
  - stocare (persistentă)
  - etc.

# Magistralele

- căi de legătură între CPU, memorie și periferice
- după informația care le parcurge
  - de date - date și instrucțiuni
  - de adrese - adrese pentru memorie și dispozitive periferice
  - de control - semnale prin care CPU comunică cu celelalte circuite și le controlează



# II. Memoria

# Tipuri de memorie

- ROM (*Read-Only Memory*)
  - conținutul său poate fi citit, dar nu și modificat
  - nevolatilă (nu își pierde conținutul la întreruperea alimentării)
- RAM (*Random Access Memory*)
  - conținutul său poate fi citit și modificat
  - volatilă (își pierde conținutul la întreruperea alimentării)

# Memoria ROM - tehnologii

- PROM (*Programmable ROM*) - conținutul său poate fi programat de utilizator
- EPROM (*Erasable PROM*) - poate fi șters și reprogramat de mai multe ori
  - UVEEPROM (*Ultra-Violet EPROM*) - ștergere prin expunere la radiație UV
  - EEPROM (*Electrical EPROM*) - ștergere prin impulsuri electrice

# Memoria RAM - tehnologii

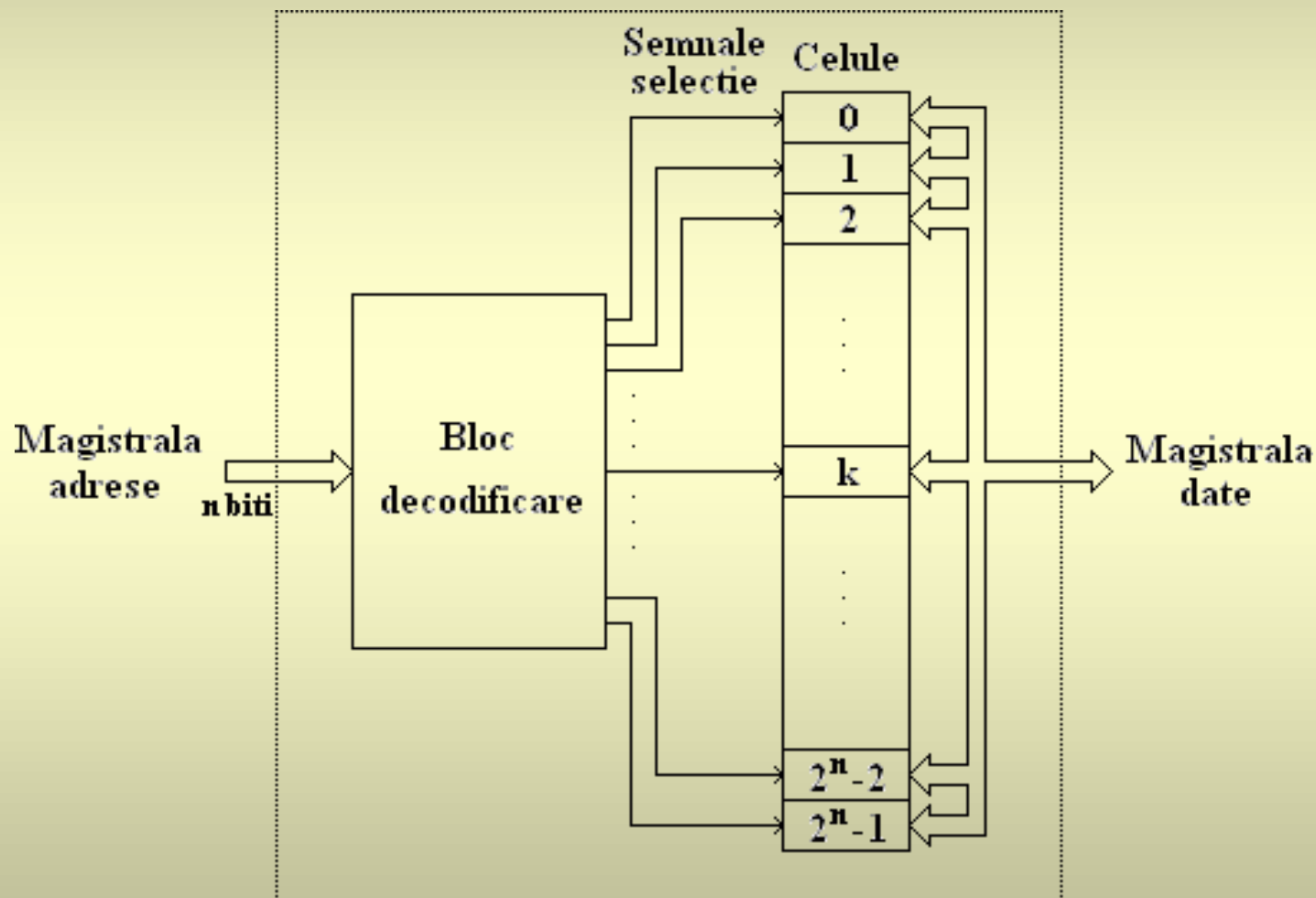
- SRAM (*Static RAM*)
  - viteză mare
  - preț ridicat
- DRAM (*Dynamic RAM*)
  - mai lentă
  - densitate de integrare mare → spațiu ocupat mic
  - preț mai redus

## Structura memoriei (1)

- șir unidimensional de celule (locații)
- fiecare celulă are asociat un număr unic - *adresa*
- bloc decodificare - selectează locația cu adresa indicată
- dimensiunea circuitului de memorie - dată de numărul de biți de adresă

$$\text{dimensiune} = 2^{nr\_biti\_adresa}$$

## Structura memoriei (2)



# Caracteristici tehnice

- timpul de acces
  - durata dintre momentul primirii comenzii și momentul furnizării rezultatului
  - același pentru toate celulele dintr-un circuit
  - ordin de mărime: nanosecunde
- capacitatea
  - până la 256 GB/circuit
  - în creștere continuă

# II.1. Memoria cache



# Problema

- procesorul este mai rapid decât memoria
  - este obligat să aștepte până când primește datele și instrucțiunile din memorie
- performanța procesorului nu este exploatată
- cauze
  - dezvoltarea tehnologică
  - factori economici

# Soluția

- legile localizării - determinate empiric
- valabile pentru marea majoritate a aplicațiilor
- tipuri de localizare
  - spațială
  - temporală

# Localizare temporală

- dacă o locație de memorie este accesată la un moment dat, este foarte probabil să fie accesată din nou în viitorul apropiat
- exemple
  - variabilele sunt folosite în mod repetat
  - bucle de program - instrucțiunile se repetă

# Localizare spațială

- dacă o locație de memorie este accesată la un moment dat, este foarte probabil ca și locațiile vecine să fie accesate în viitorul apropiat
- exemple
  - parcurgerea tablourilor
  - bucle de program

# Legile localizării - consecințe

- la un moment dat, un program folosește numai o (mică) parte din locațiile sale de memorie
- un circuit de capacitate mai mică poate reține aceste locații
- acest circuit poate fi mai rapid
- sporul de performanță poate fi major

# Memoria cache

- circuit rapid și de capacitate mică
  - interpus între procesor și memoria principală
- reține locațiile din memoria principală accesate cel mai recent de procesor
- procesorul solicită conținutul unei locații
  - se caută mai întâi în cache
  - dacă locația nu este găsită în cache - căutare în memoria principală

# Caracteristici

## Viteza

- foarte mare - la același nivel cu procesorul
  - tehnologie performantă (SRAM)
  - dimensiune redusă → bloc de decodificare mai simplu, deci mai rapid

## Prețul

- rezonabil
  - datorită dimensiunii reduse

# Parametri de performanță (1)

- H - rata de succes (*hit ratio*)
  - procentajul cazurilor în care locația căutată a fost găsită în cache
- M - rata de insucces (*miss ratio*)
  - procentajul cazurilor în care locația căutată nu a fost găsită în cache

$$0 \leq H, M \leq 1$$

$$M = 1 - H$$



## Parametri de performanță (2)

- $T_c$  - timpul de acces la cache
- $T_m$  - timpul de acces la memorie în cazul unei ratări în cache
- $T$  - timpul mediu de acces la memorie (cu cache)
- $T_p$  - timpul de acces la memoria principală (în absența cache-ului)

# Performanța memoriei cache (1)

$$T = T_c \cdot H + T_m \cdot M$$

- dacă  $T < T_p \rightarrow$  spor de viteză
- $T$  - mărime statistică
- cazuri extreme
  - $H=100\%$  ( $M=0$ ):  $T = T_c \rightarrow$  ideal
  - $H=0$  ( $M=100\%$ ):  $T = T_m \rightarrow$  pierdere de viteză  
( $T_m > T_p$ )

## Performanța memoriei cache (2)

Situația reală - exemplu

- $T_c = 2 \text{ ns}$
- $T_p = 10 \text{ ns}$
- $T_m = 11 \text{ ns}$
- $H = 95\%$
- $T = 2.45 \text{ ns} = 0.245 T_p \rightarrow$  viteza de acces crește de peste 4 ori

# Adresare

- adresa din cache nu corespunde cu adresa din memoria principală
- căutarea se face după adresa din memoria principală
- deci cache-ul trebuie să rețină și adresele locațiilor în memoria principală

## Linii de cache

- cache-ul se folosește de localizarea temporală
- cum se poate exploata și localizarea spațială?
- când se aduce o locație în cache, se aduc și locațiile vecine
  - acestea formează o *linie de cache*

## Politica de înlocuire

- cache mic - se umple repede
- noi linii aduse în cache - trebuie eliminate altele mai vechi
  - eliminare - scriere în memoria principală
- care linii trebuie eliminate?
  - scopul - creșterea vitezei
- cele care nu vor fi accesate în viitorul apropiat !!!

# Îmbunătățirea performanței

- depinde de două mărimi
  - timpul de acces la cache ( $T_c$ )
  - rata de succes ( $H$ )
- nu pot fi optimizate simultan
- influențate de
  - tehnologie
  - politica de înlocuire

# Tipuri constructive de cache

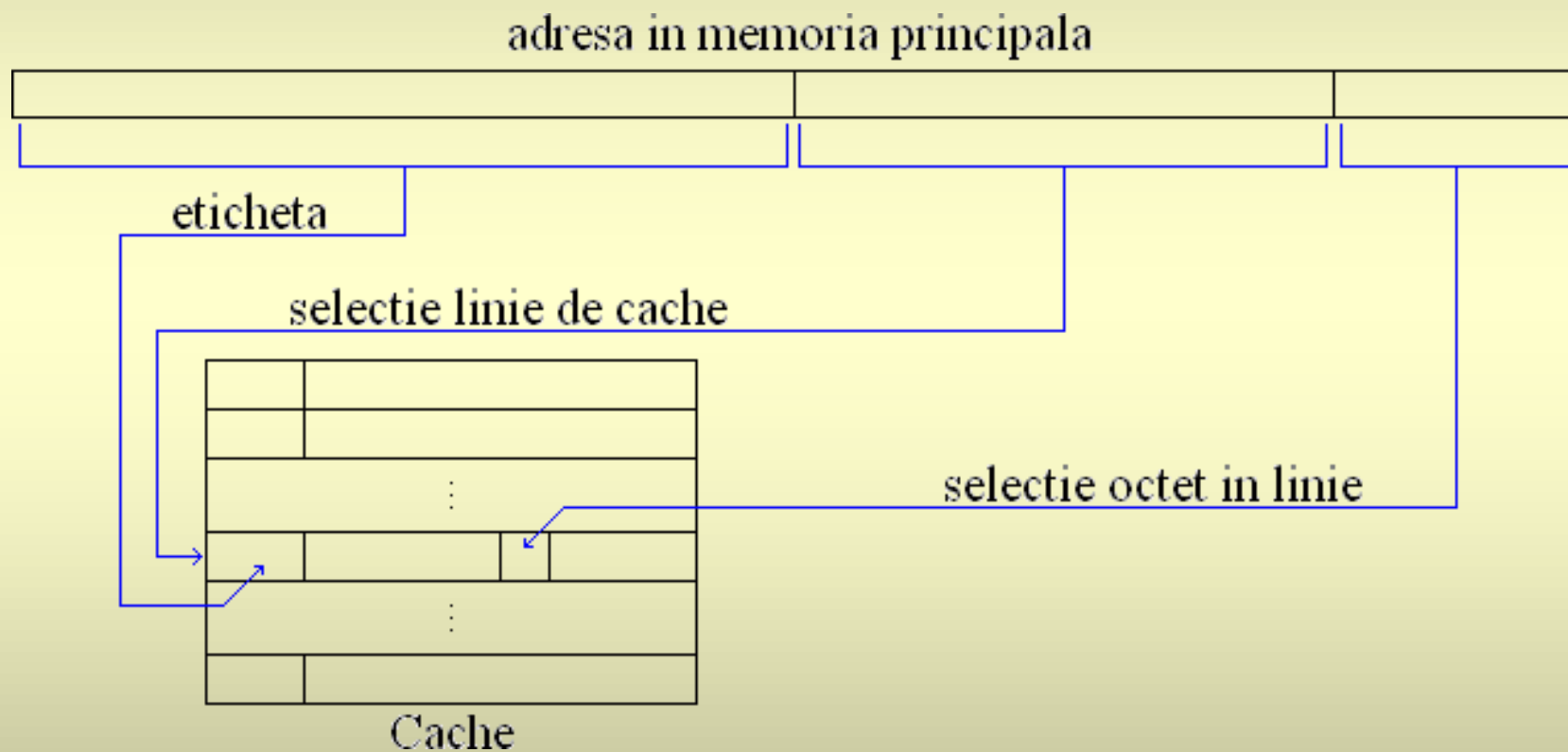
- cu adresare directă (*direct mapped cache*)
- total asociativ (*fully associative cache*)
- parțial asociativ (*set associative cache*)



# Cache cu adresare directă (1)

- plasarea unei locații în cache
  - linia de cache este întotdeauna aceeași
  - depinde de adresa din memoria principală
- adresa din memoria principală - 3 părți
  - eticheta - se memorează în cache
  - selectorul liniei de cache
  - selectorul octetului în cadrul liniei

## Cache cu adresare directă (2)



## Cache cu adresare directă (3)

### Exemplu

- adresa în memoria principală - 32 biți
- dimensiune cache:  $2^{11}$  linii  $\times$   $2^5$  octeți/linie
- adresa în memoria principală se împarte în
  - selectorul liniei de cache - 11 biți
  - selectorul octetului în cadrul liniei - 5 biți
  - eticheta - 16 biți (  $= 32 - 11 - 5$  )

## Cache cu adresare directă (4)

### Exemplu (continuare)

- adresa în memoria principală:  $45097373_{(10)}$

$00000010101100000010000110011101_{(2)}$

- eticheta:  $0000001010110000_{(2)} = 688_{(10)}$

- linia de cache:  $00100001100_{(2)} = 268_{(10)}$

- octetul în cadrul liniei:  $11101_{(2)} = 29_{(10)}$

## Cache cu adresare directă (5)

Exemplu (continuare)

- Ce adrese din memoria principală sunt aduse în linia de cache?

000000101011000000100001100xxxxx<sub>(2)</sub>

00000010101100000010000110000000 ÷

000001010110000001000011001111<sub>(2)</sub>

45097344 ÷ 45097375<sub>(10)</sub>

## Cache cu adresare directă (6)

Conținutul unei linii de cache

- un bit care indică dacă linia conține date valide
  - inițial, toate liniile sunt goale, deci invalide
- câmpul etichetă
- datele propriu-zise, aduse din memoria principală

# Cache cu adresare directă (7)

## Avantaje

- implementare simplă
- timp de acces ( $T_c$ ) redus

## Dezavantaje

- lipsă de flexibilitate
- politică de înlocuire neperformantă - rată de succes ( $H$ ) scăzută

## Cache cu adresare directă (8)

### Exemplu

```
for (i=0; i<1000; i++) a=a+i;
```

- adrese:  $i \rightarrow 3806240$ ,  $a \rightarrow 1756566572$
- ambele sunt memorate în cache în linia 161
- accese alternative  $\rightarrow$  înlocuiri dese în cache  
 $\rightarrow$  număr mare de ratări



## Cache total asociativ (1)

- realizat cu memorii asociative
  - memoria obișnuită - acces la o locație pe baza adresei sale
  - memoria asociativă - permite și regăsirea locației pe baza conținutului său
  - implementare - valoarea căutată este comparată în paralel cu toate locațiile
    - de ce în paralel?

## Cache total asociativ (2)

### Avantaje

- plasarea datelor din memoria principală - în orice linie de cache
- se pot alege convenabil adresele aduse în linia de cache
- se pot implementa politici de înlocuire eficiente - rată de succes (H) ridicată

## Cache total asociativ (3)

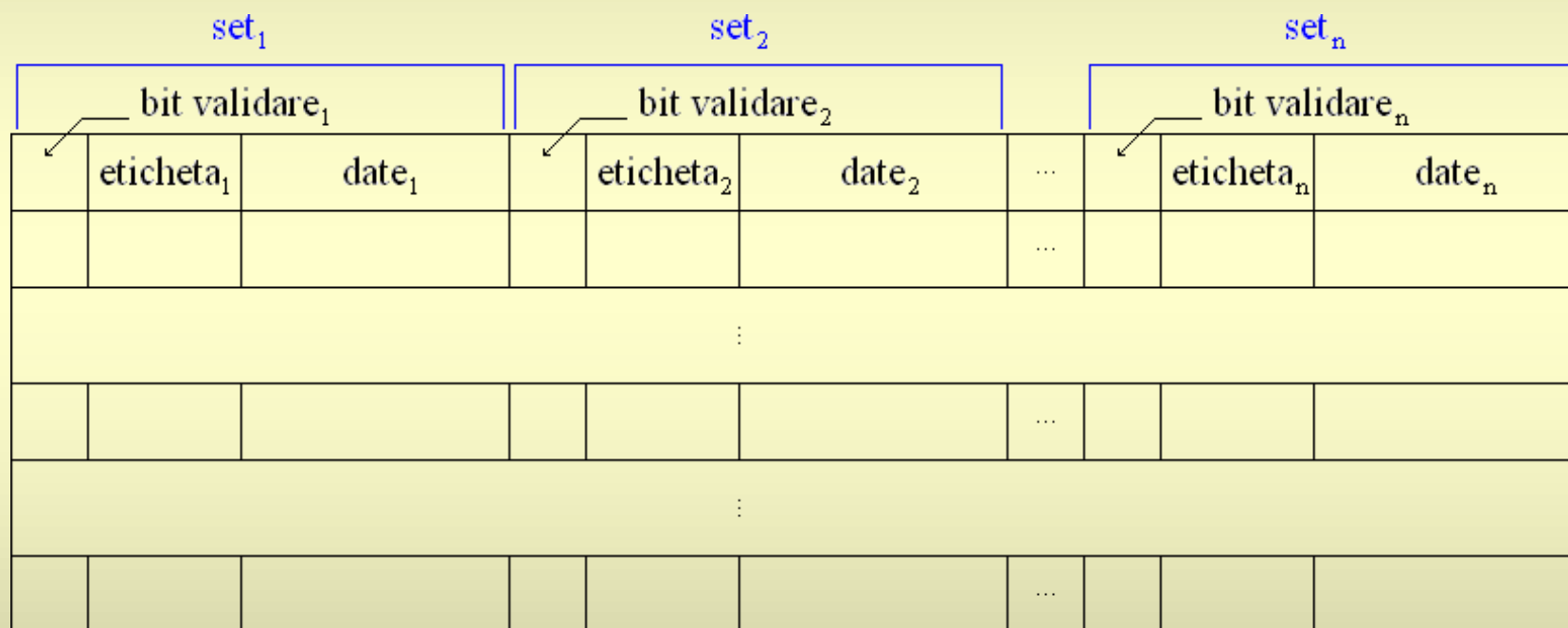
### Dezavantaje

- timp de acces ( $T_c$ ) mare
  - memoriile asociative - lente
  - algoritmi complecși de înlocuire - timp suplimentar consumat
- hardware complicat pentru memoriile asociative și algoritmi de înlocuire

## Cache parțial asociativ (1)

- numit și cache asociativ pe seturi
- derivat din cache-ul cu adresare directă
- fiecare linie de cache conține mai multe seturi de date (4, 8, 16, ...)
- structura unui set
  - bit de validare
  - etichetă
  - date din memoria principală

## Cache parțial asociativ (2)



## Cache parțial asociativ (3)

Timpul de acces ( $T_c$ )

- puțin mai mare decât la cache-ul cu adresare directă
  - trebuie verificate toate cele  $n$  seturi

Rata de succes ( $H$ )

- ridicată
  - elimină problema suprapunerilor

## Scrierea în cache (1)

- scriere într-o locație care nu se află în cache
- unde se face scrierea?
- variante
  - doar în memoria principală - nu se poate
    - de ce?
  - doar în cache (*write-back*)
  - atât în cache, cât și în memoria principală (*write-through*)

## Scrierea în cache (2)

Cache de tip *write-back*

- scrierea se face doar în cache
- datele ajung în memoria principală doar la evacuarea din cache
- viteză mare
- probleme în sistemele multiprocesor



## Scrierea în cache (3)

### Cache de tip *write-through*

- scrierea se face atât în cache, cât și în memoria principală
- mai lent
  - datorită accesului la memoria principală
- ambele tipuri de cache sunt larg folosite

# Conceptul de cache - extindere

- aplicabil nu doar la procesoare
- tipul de problemă: comunicarea cu o entitate lentă, de mare capacitate
- soluția: interpunerea unei entități cu capacitate mai mică și viteză mai mare
  - reține ultimele date vehiculate

# Unde mai putem folosi ideea?

## Aplicabilitate

- oriunde funcționează legile localizării
- hardware
- software

# Example (1)

## Cache-uri de disc

- 2 direcții de aplicare
  - hardware - circuit de memorie integrat în controller
  - software - o zonă din memoria sistem
- entitatea mai mare și lentă - discul
- entitatea mai mică și rapidă - memoria

## Example (2)

### Browserul web

- ultimele pagini accesate sunt reținute pe disc
  - numai localizare temporală - de ce?
- entitatea mai mare și lentă - rețeaua (Internet)
- entitatea mai mică și rapidă - discul