Computadors

El Computador i els Seus Elements (1/2)

Grau en Ciència i Enginyeria de Dades

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

Creative Commons License

Aquest document utilitza Creative Commons Attribution 3.0 Unported License

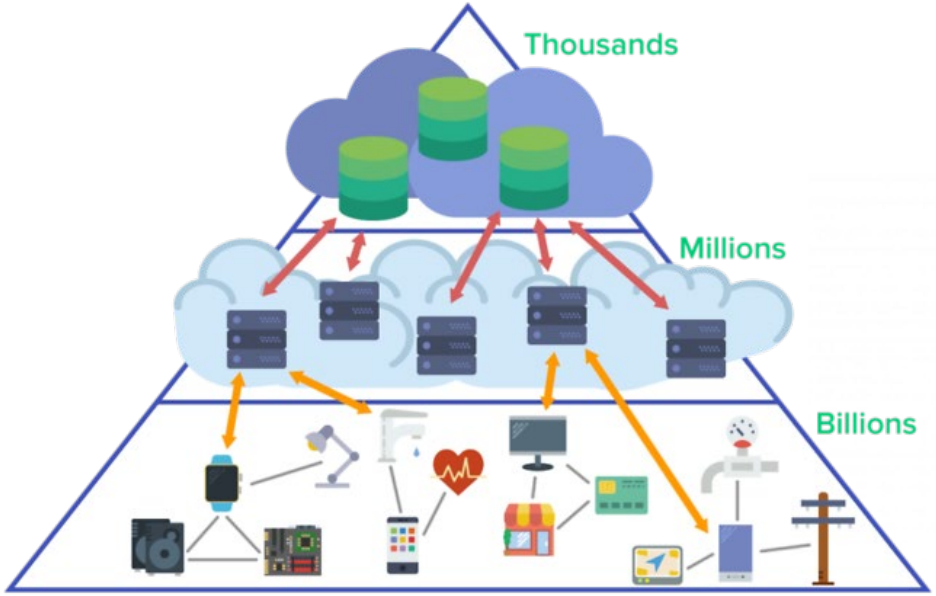
Els detalls d'aquesta licencia es poden trovar a
https://creativecommons.org/licenses/by-nc-nd/4.0

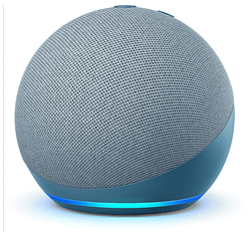
Index

- Aquest tema té tres parts
  1) **Components Bàsics i Jerarquia de Memòria**
  2) Fluxe d'execució i colls d'ampolla

Des de Sistemes Informàtics fins al Processador

- Un sistema informàtic consisteix en hardware i software de sistema…
  - …per executar programes
- Els components poden ser diferents, però el concepte és el mateix

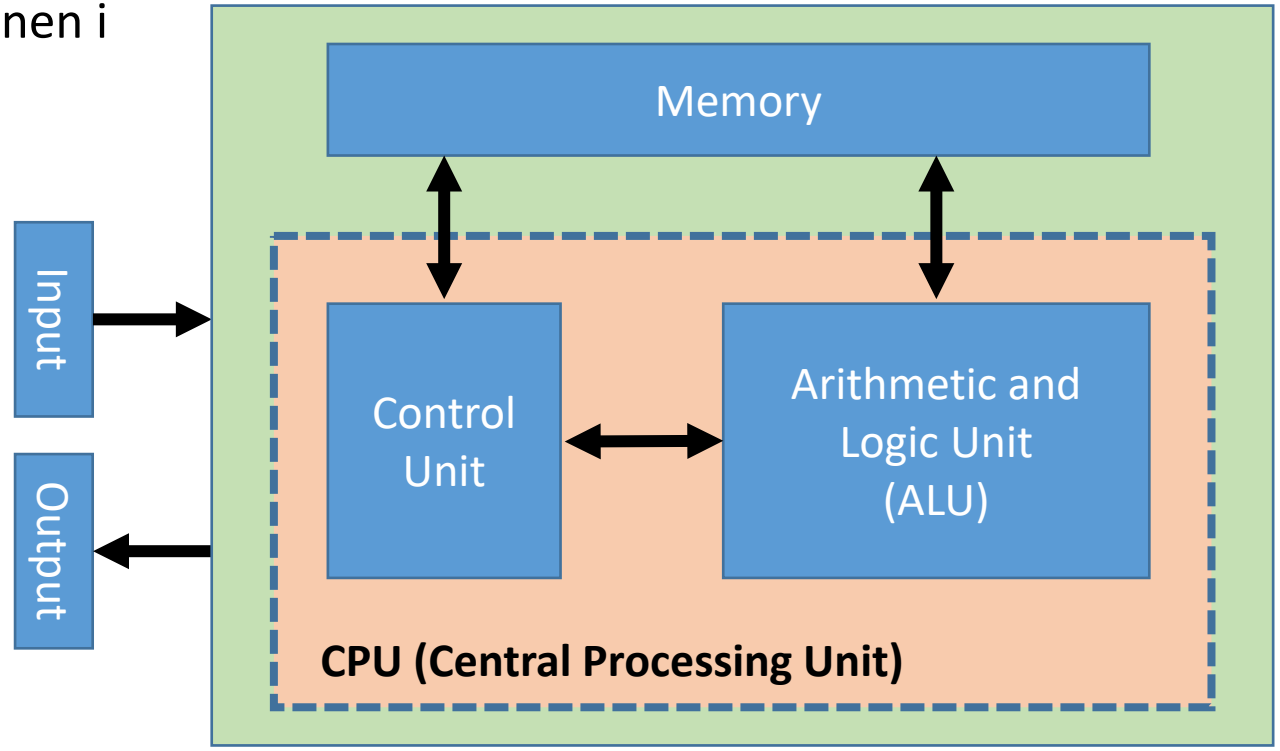




Arquitectura de Computadors

- **Arquitectura de Computadors**
  - Conjunt de regles que defineixen la manera en que **els components hardware i software** es combinen i interactúen entre ells per treballar com un dispositiu funcional
- **Arquitectura Von-Neumann* (la més usada)**
  - El disseny defineix la manera en que s'obtenen i es processen les instrucción i les dades:

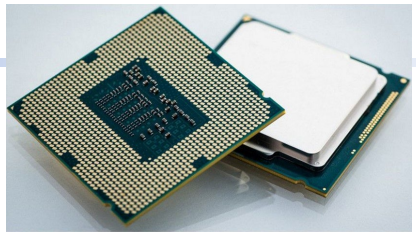
  1) La CPU rep instruccions i dades
     - Des d'un dispositiu d'entrada o memòria
  2) La CPU les processa
  3) Els resultats s'envíen fora
     - A un dispositiu de sortida o memòria
- **Arquitectura Harvard**
  - Enfocament alternatiu (e.g. DSPs)
    - Memòria separada per instruccions i per dades

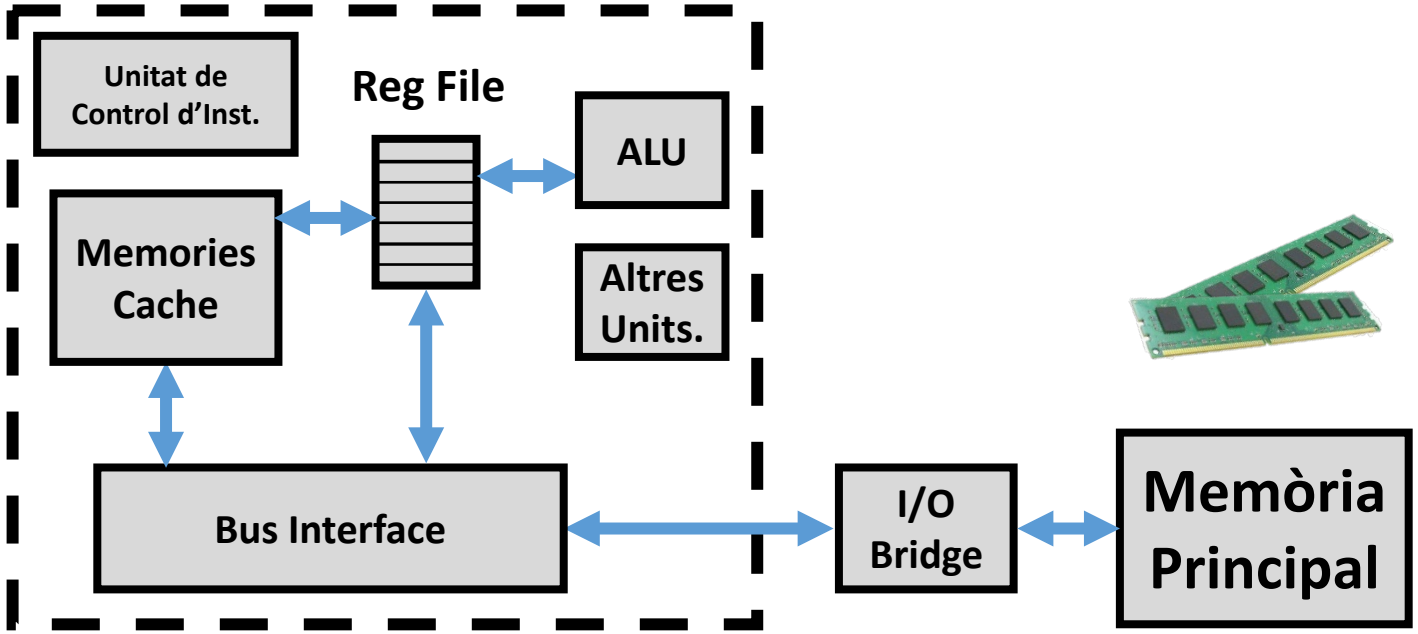


* https://www.britannica.com/biography/John-von-Neumann

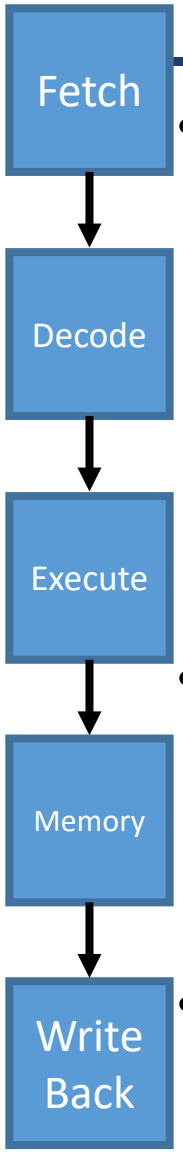
Arquitectura del Processador

- **Unitat de Control d'Instruccions**
  - Gestiona l'execució de les instruccions
- **Bank de Registres (Reg File)**
  - L'emmagatzematge (de mida "word") de més proper accés
- **Memòries Cache**
  - Jerarquía de memòria per reduir l'espai CPU-mem
- **Interfície del Bus**
  - Circuitaria per transaccions de memòria
- **ALU (Arithmetic/Logic Unit)**
  - Operacions senzilles
- **Altres Unitats**
  - Unitats complexes i/o acceleradors

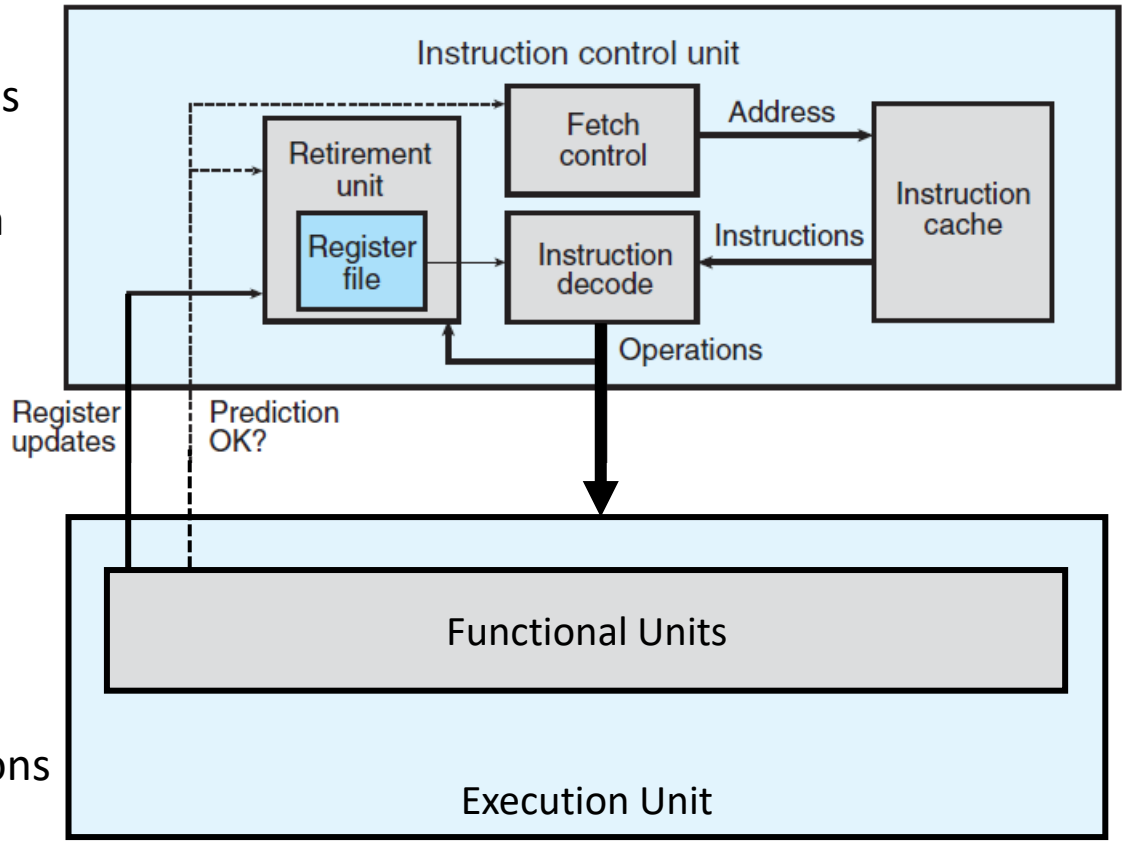




Execució d'Instruccions



- **Instruction Control Unit…**
  - **Fetch Control:** porta la següent seqüència d'instruccions
    - Què passa si no està clar quina és la següent instrucció?
  - **Instruction Decode:** prepara l'instrucció per executar-la
    - Tradueix d'instruccions a operacions primitives o microoperacions
      - Depèn de l'ISA (Instruction Set Architecture)
    - És carreguen els operands des del **Register File**
      - És la unitat més petita i ràpida de la jerarquia de memòria
- **Execution Unit…**
  - Envia les operacions primitives a les unitats funcionals corresponents
  - Les **unitats funcionals** són les que realitzen les operacions
    - Load, Store, ALU, FP, …
- **Tornada a la Instruction Control Unit…**
  - **Retirement Unit:** fa el seguiment de l'execució per guardar els resultats en el ordre correcte

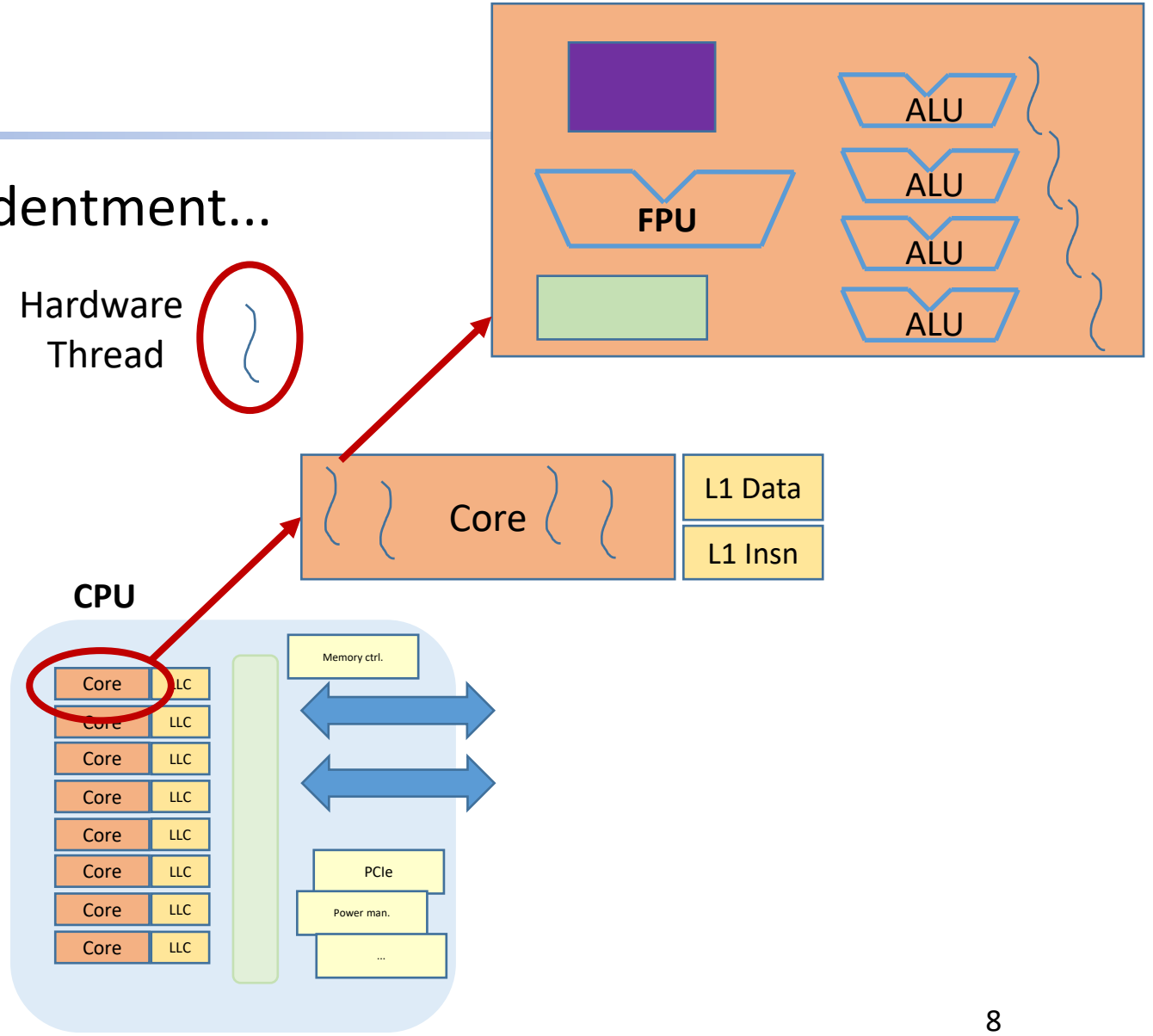


Hardware Thread (Fil)

- Cada hardware thread fa independentment...
  - Fetch
  - Decode
  - Execute
  - Memory access
  - Store results (Write Back)

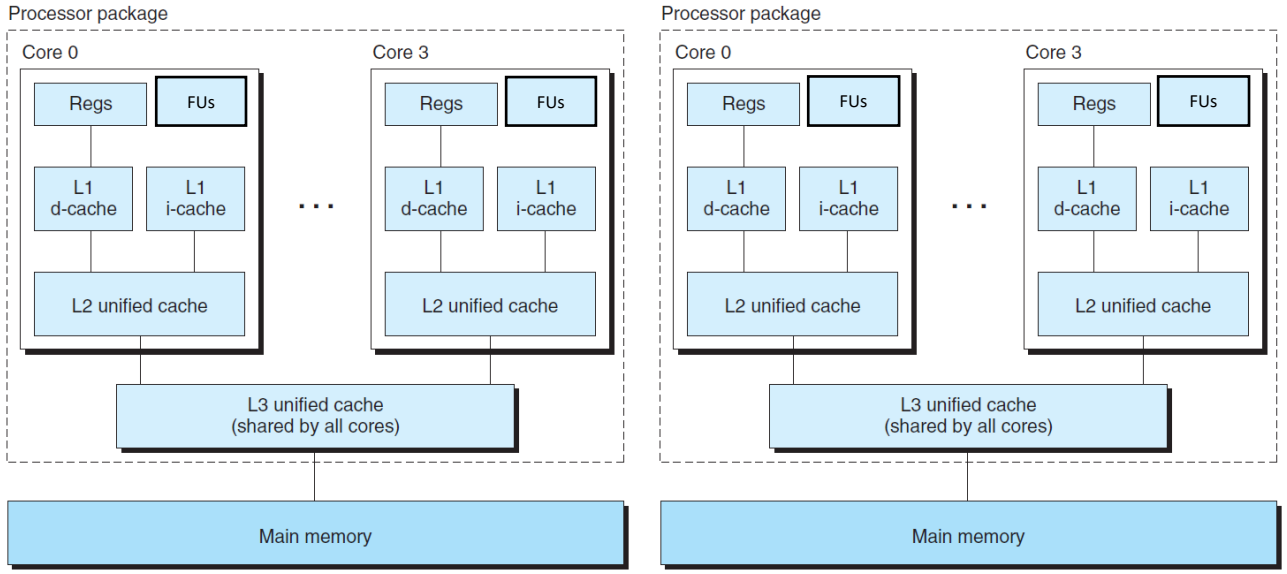
Quan s'executa un únic thread per core,
té tots els recursos disponibles!
- Amplada de banda de memòria
- Unitats Funcionals

- Multithreading
  - Executar varis threads en paral·lel



Concurrència a nivel de Thread

- Fil (Thread): fluxe de control
  - Software thread: fil d'execució que representa el fluxe de control d'un programa
  - Hardware thread: conjunt de recursos Hw necessaris per executar un Sw thread
- El Sistema Operatiu gestiona la concurrència (varies activitats simultànies) i aprofita els diferents nivells d'abstracció de paral·lelisme
  - Multi-processador
    - Varis processadors gestionats pel mateix Sistema Operatiu
  - Multi-core
    - Varis CPUs integrats en el mateix chip
  - Multi-thread
    - Una CPU executa múltiples fluxes de control
    - Alguns recursos Hw són replicats
      - E.g. program counter, registre files, simple ALU
    - Alguns recursos Hw són compartits
      - E.g. complex ALU, FPU



Instruction Set Architecture (ISA)

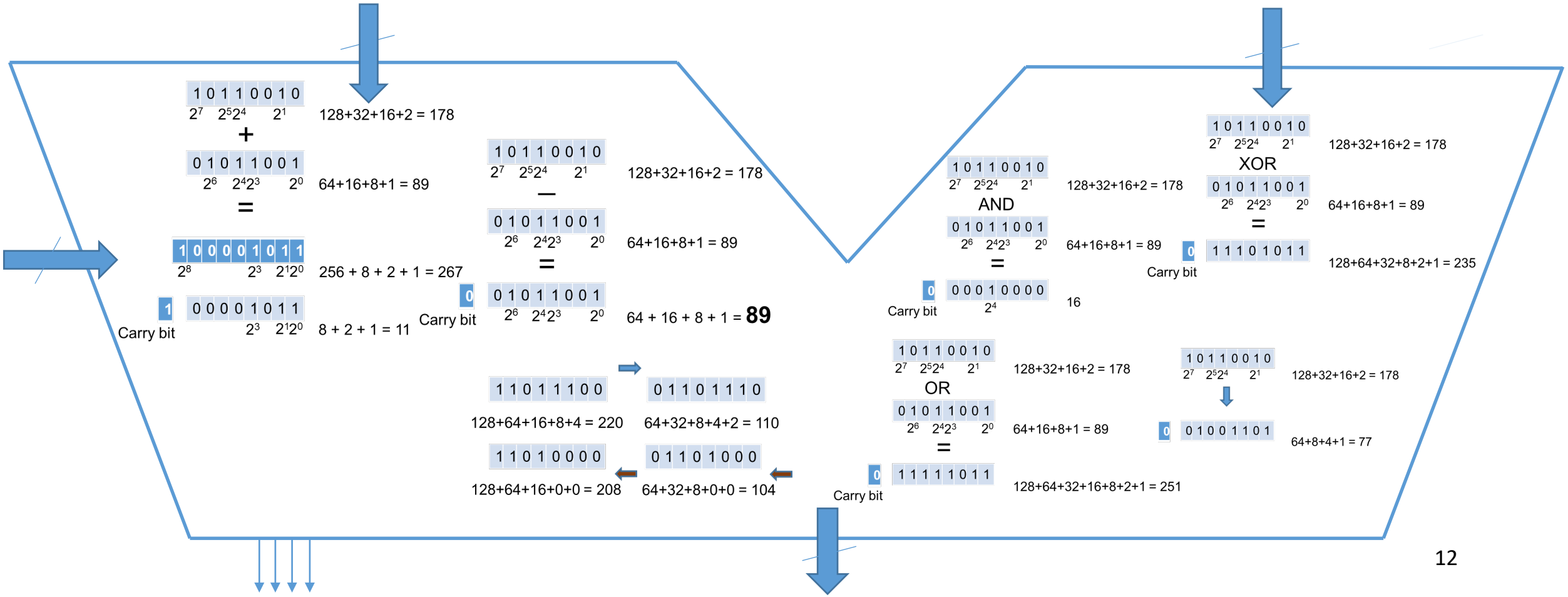
- El Conjunt d'Instruccions, **Instruction Set Architecture (ISA)**, defineix…
  - El format i tipus d'instruccions
  - Codificació de les operacions
  - La manera d'accedir a posicions de memòria
  - Registres
  - Mecanismes de Paral·lelisme (e.g. SIMD)
  - …
- Compromisos en la complexitat de l'ISA: "forat" semàntic
  - Més proper a llenguatges d'alt nivell vs més proper a senyals de control del hardware
    - Reduced Instruction Set Computing **(RISC)**, com ara MIPS, ARM, RISC-V
    - Complex Instruction Set Computing **(CISC)**, com ara Motorola 68K, els primers processadors Intel x86
    - Híbrid CISC-RISC, com ara els processasdors Intel x86 posteriors
    - Altres…
  - Impacte en la complexitat del compilador i de la microarquitectura
- **Llenguatge ensamblador** és la codificació en format text del codi máquina
  - Instruccions de baix nivel i registres

El Banc de Registres (Register File)

- L'unitat de memòria més petita i ràpida de la jerarquia de memòria
  - Està en la CPU
- Registres de diferents mides (depèn de l'ISA)
  - 64-bit, 32-bit, però també sub-registres per accedir a 16 i 8 bits
- Diferentes categories en funció del seu propòsit
  - General-Purpose Registers (GPRs)
    - E.g. x86-64 té 16 GPRs complets i 52 sub-registres
      - RAX, EAX, AX, AH, AL
  - FPU Registers
  - Altres conjunts de registres, entre altres…
    - Special-purpose (e.g. program counter, status register)
    - Vector register

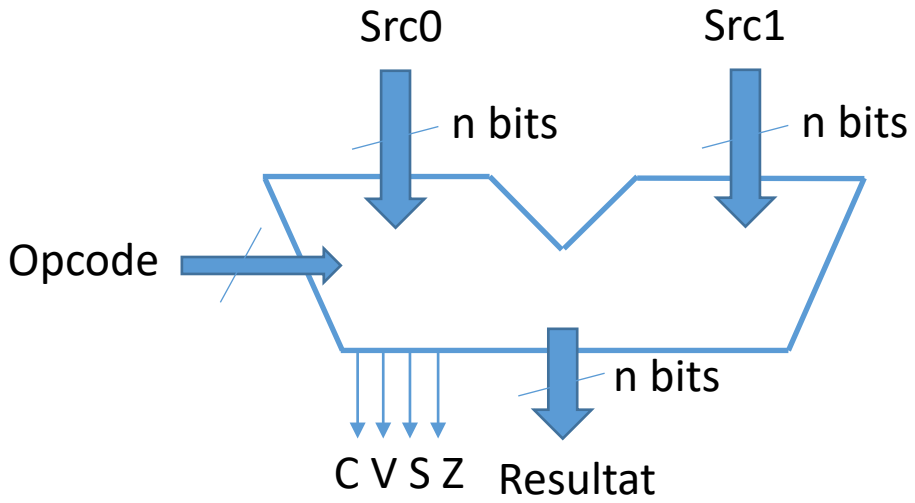
ALU (Arithmetic and Logic Unit)

- Operacions senzilles aritmètico-lògiques



ALU (Arithmetic and Logic Unit)

- Exemple d'ALU amb opcodes de 4-bits



- C: Flag de Carry
- V: Flag d'Overflow
- S: Flag de Signe
- Z: Flag de Zero

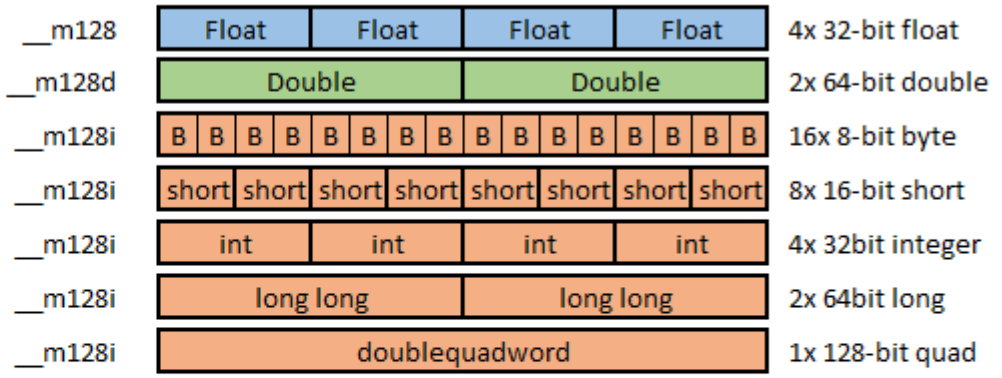
Opcode				Operació
0	0	0	0	Src0 + Src1
0	0	0	1	Src0 – Src1
0	0	1	0	Src0 * Src1
0	0	1	1	Src0 / Src1
0	1	0	0	Shift left (Src0) by Src1
0	1	0	1	Shift right (Src0) by Src1
0	1	1	0	Rotate left (Src0) by Src1
0	1	1	1	Rotate right (Src0) by Src1
1	0	0	0	Src0 AND Src1
1	0	0	1	Src0 OR Src1
1	0	1	0	Src0 XOR Src1
1	0	1	1	NOT(Src0)
1	1	0	0	NOT(Src1)
1	1	x	x	Reserved for future use

Paral·lelisme SIMD

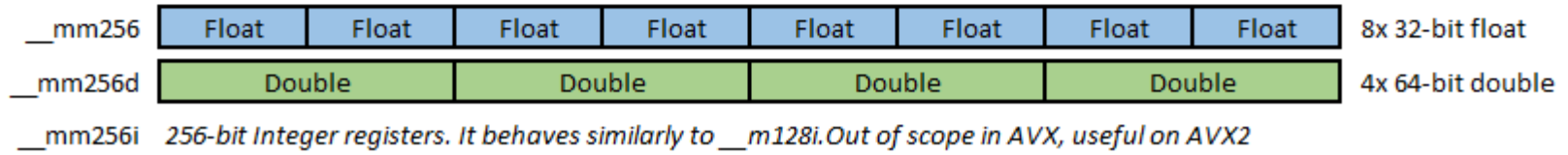
- **Paral·lelisme SIMD**: Single Instruction Multiple Data
  - Exemple basat en Intel x86-64 (64 bits)
    - SSE (Streaming SIMD Extensions, 128-bit); AVX2 (Advanced Vector eXtensions, 256-bit); AVX512 (512-bit)
    - Registres + Instruccions

- SISD, **SIMD**, MISD, MIMD
  - Single vs Multiple
  - Instruction; Data

SSE Data Types (16 XMM Registers)

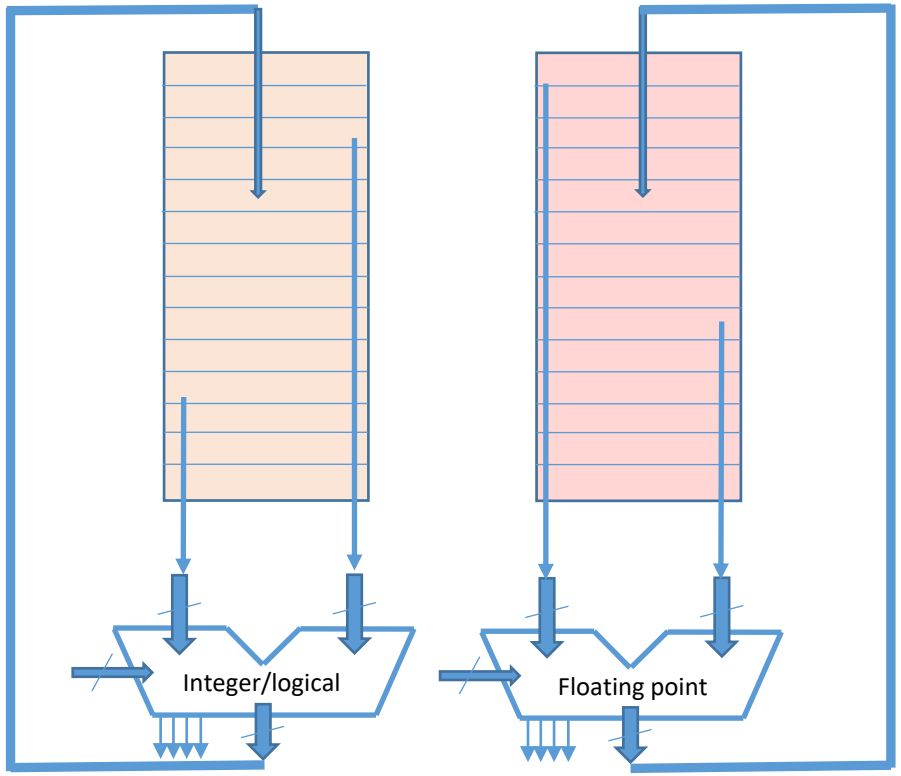


AVX Data Types (16 YMM Registers)

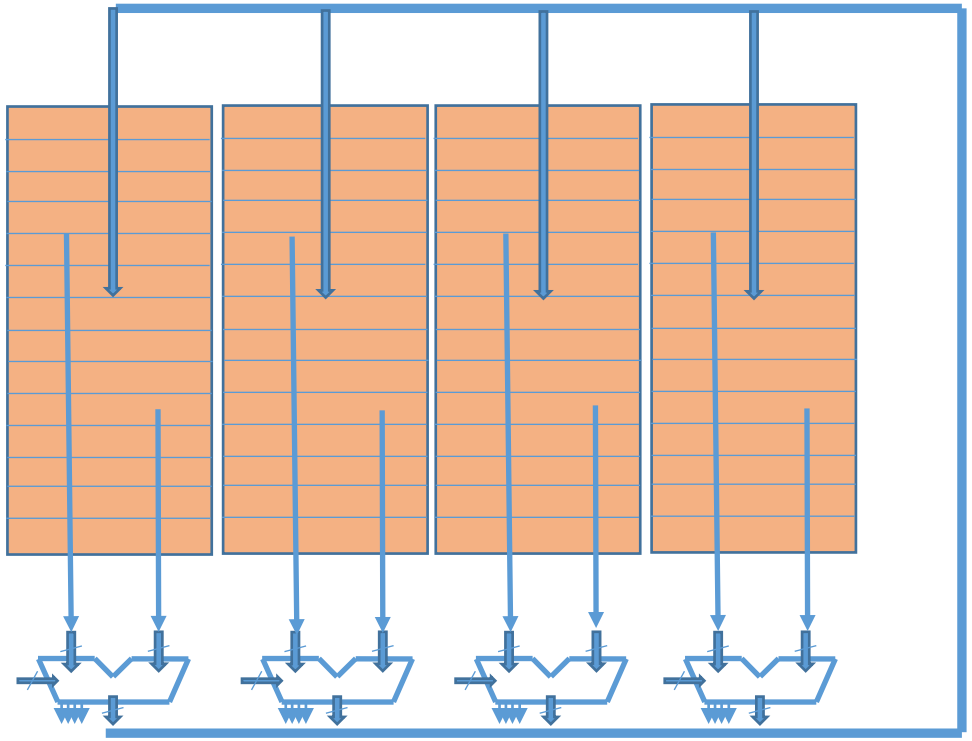


Registres + ALU/FPU

- Operacions que tenen diferents latències segons complexitats i optimitzacions de les operacions



Integer Floating point



Vector (integer and floating point)

Exemple de codi

20 10 60 20 61 37 72 84 00 00 00 00 00 00 00 20 12 20 01 70 68 00 00 00 00 00 00 00

rrmovq %rcx, %rax
- 0 as cc: always
- 1 as reg: %rcx
- 0 as reg: %rax

addq %rdx, %rax

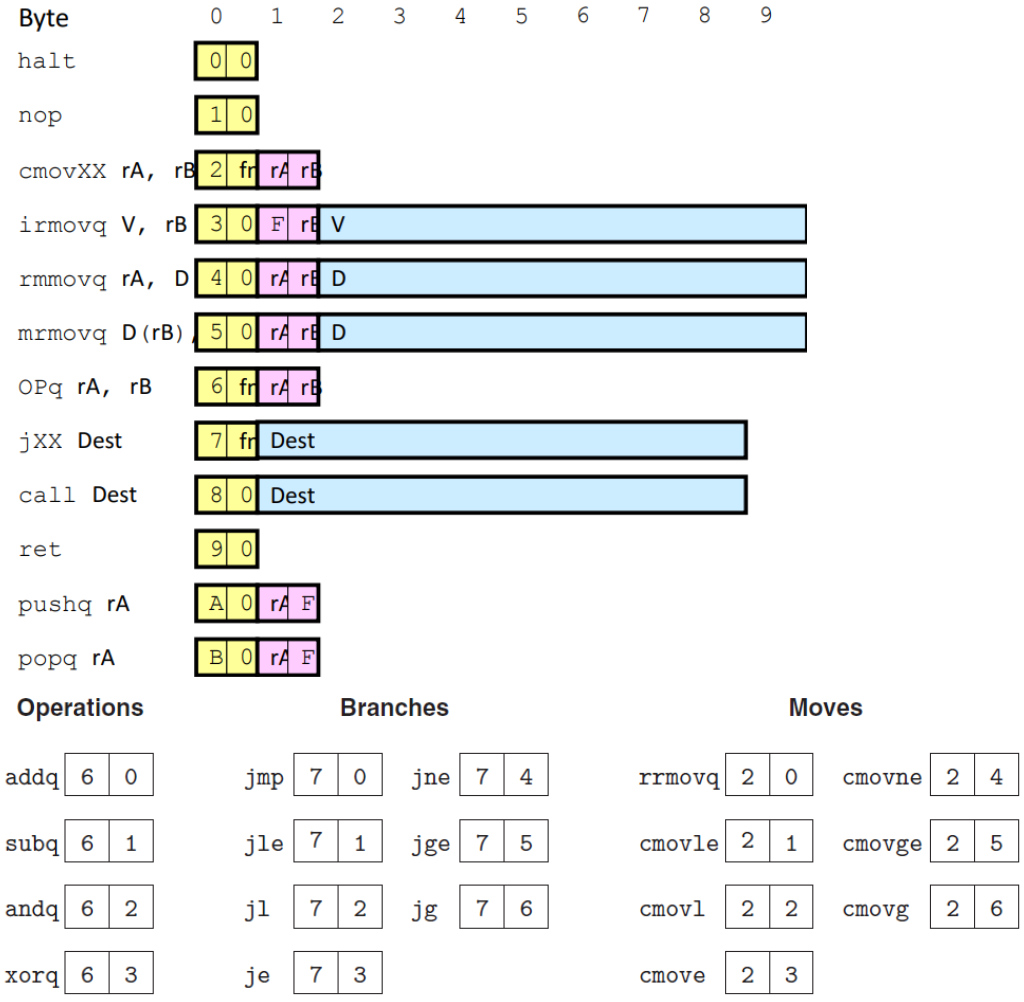
subq %rbx, %rdi
- 0 as fn: add
- 1 as fn: sub

jl 0x84
- 2 as cc: l (less than)
- hex 84 00… as little endian Dest: 0x84

rrmovq %rcx, %rdx

rrmovq %rax, %rcx

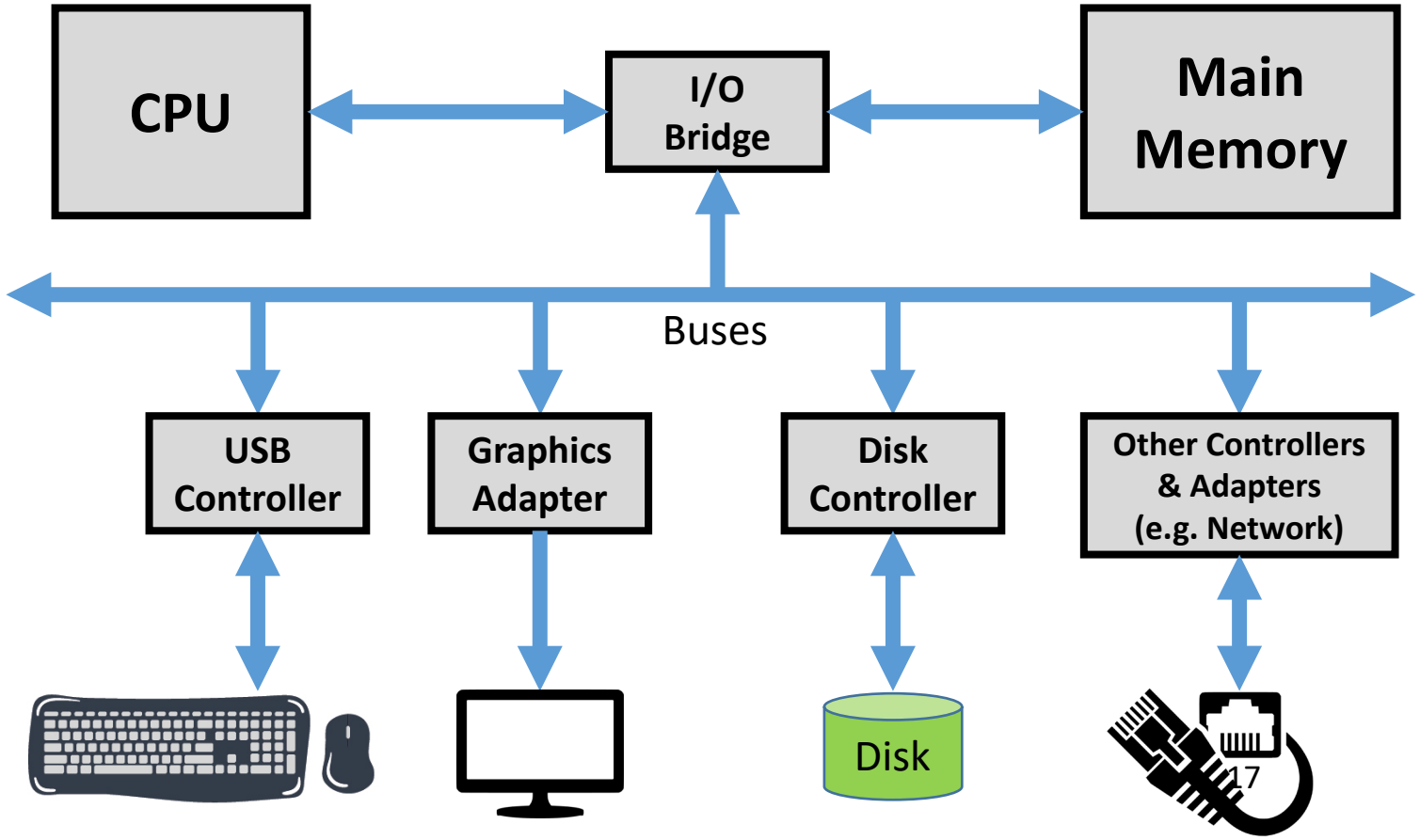
jmp 0x68



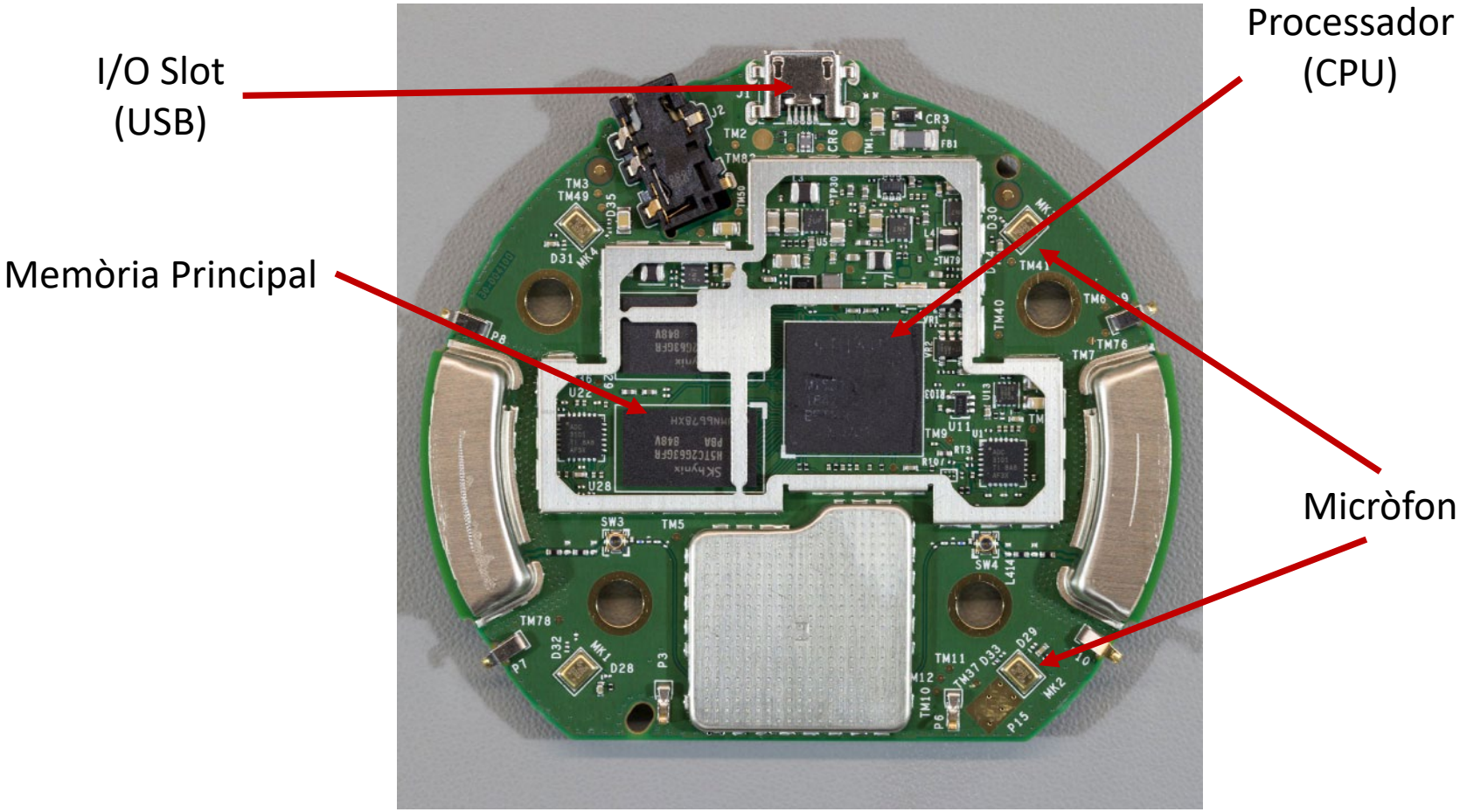
Exemple obtingut de les transparències de University of Virginia (link)

Organització del Hardware

- **Processador or Central Processing Unit (CPU)**
  - Executa instruccions portades des de memòria
- **Memòria Principal**
  - Memòria Volàtil (RAM)
- **Buses**
  - Connexions cablejades
  - Conjunts de Bytes
- **I/O Bridge**
  - Hub d'interconnexió
  - Northbridge/Southbridge
- **Dispositius I/O**
  - Device, Controller, Adapter
  - Storage No-volàtil(e.g. Disc)



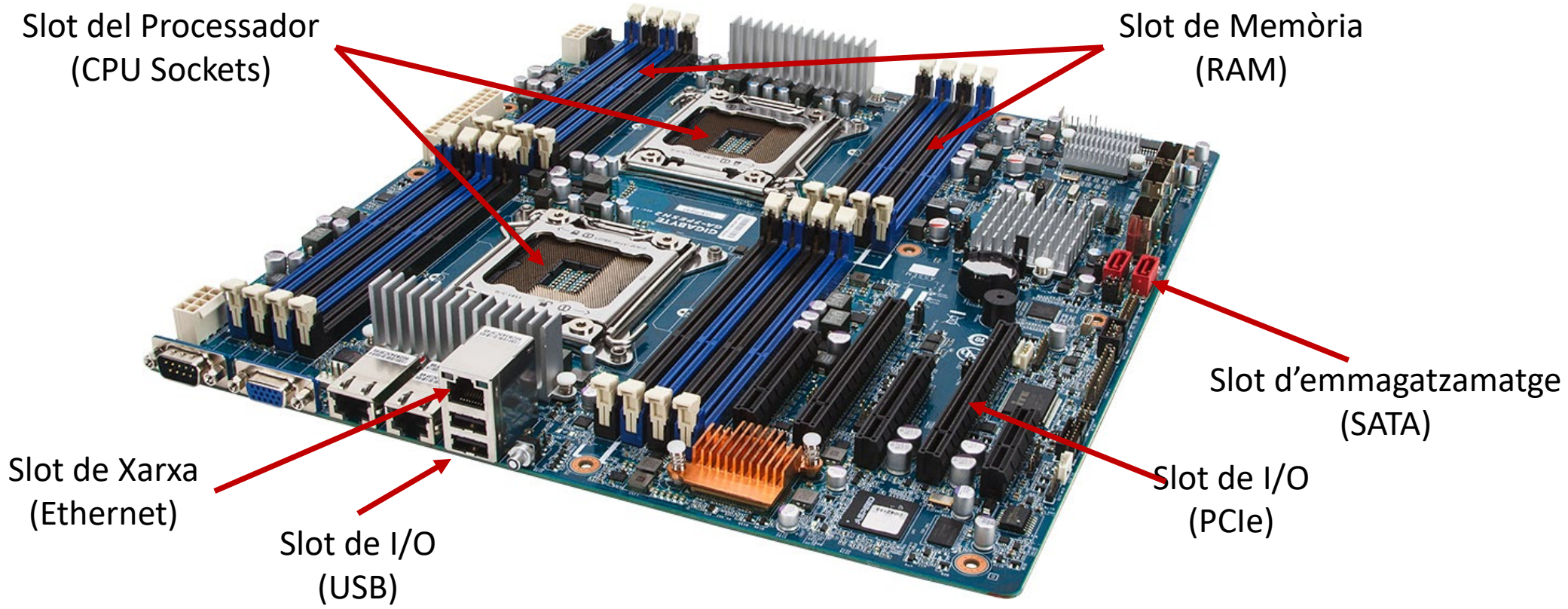
Exemple del interior d'un Virtual Assistant



https://www.briandorey.com/post/amazon-echo-input-teardown

Exemple de Placa Base amb Multiprocessador

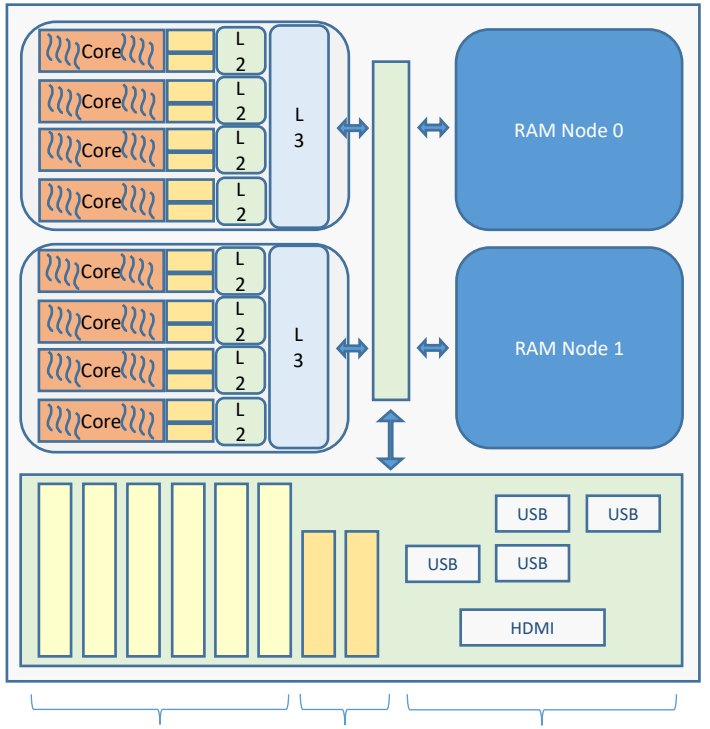
- **Placa Base:** El circuit principal en el que els components es connecten



https://community.mellanox.com/s/article/understanding-numa-node-for-performance-benchmarks

Components d'E/S

- El Bus d'E/S permet l'accés a
  - Acceleradors (GPUs, FPGAs)
  - Discos
  - Xarxa
  - Perifèrics d'Interfície Humà-Màquina



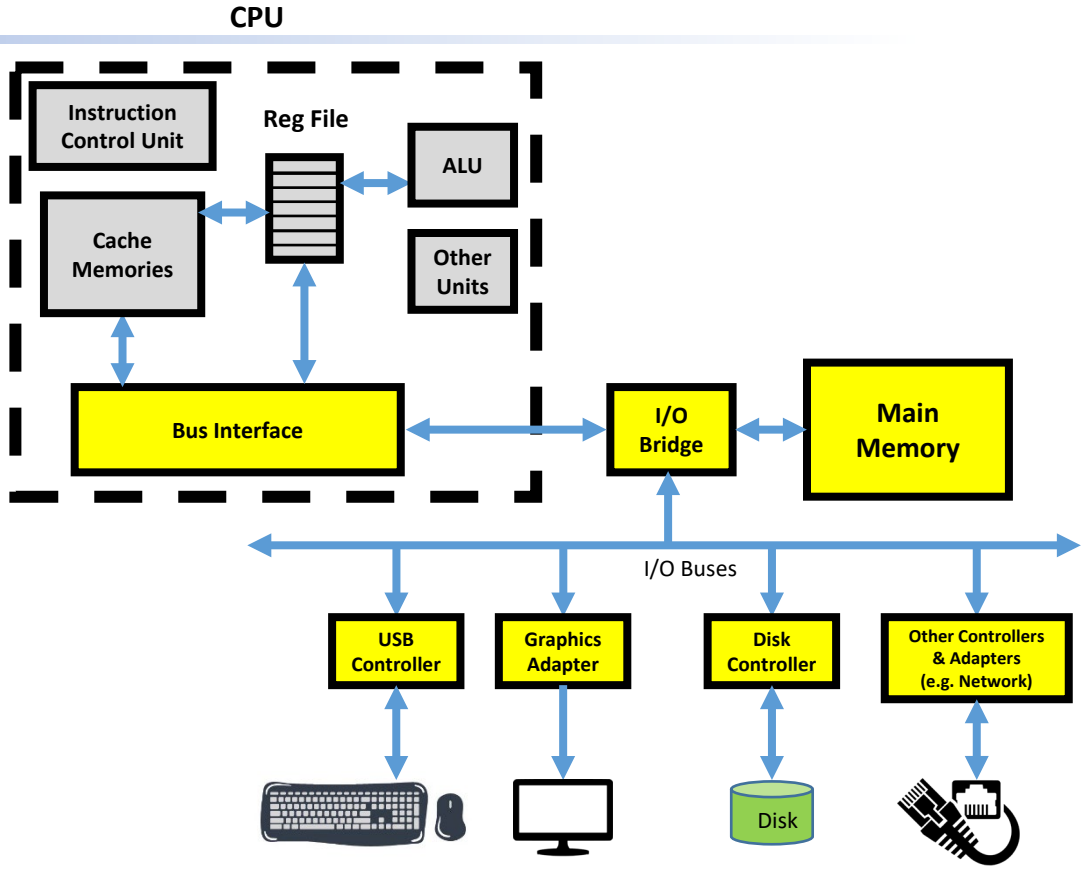
PCIe cards
GPUs / FPGAs
Networking

Sata
Disks

HMI Peripherals
Keyboard/mouse
Video
Audio

Accés al Bus i E/S

- Un bus és una agrupació de cables
  - Porta adreces, dades i senyals de control



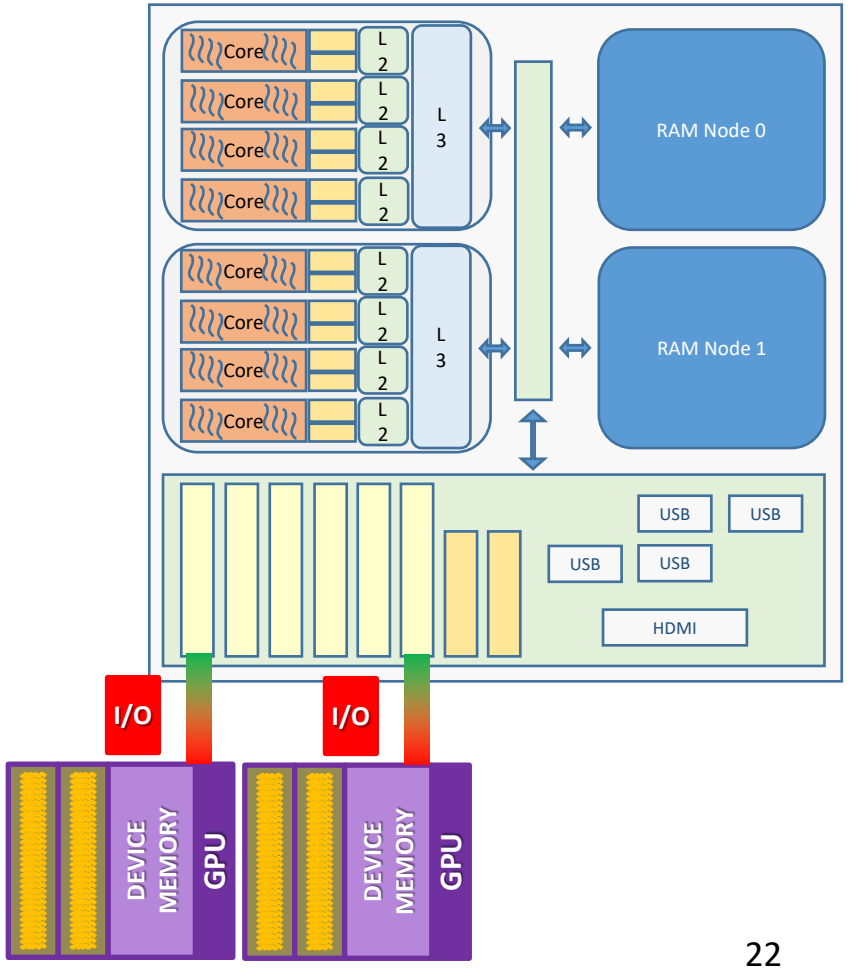
- La CPU pot comunicar-se amb dispositius d'E/S
  - Tècnica d'E/S: mapeig de memòria
    - Un dispositiu està "mapejat" a una adreça concreta de memoria (a.k.a. port d'E/S)

- DMA: Direct Memory Access
  - Els perifèrics poden llegir/escriure directament a/des d'adreces de memòria

Acceleradors

- Components addicionals per computació
  - Necessitat de transferir
    - codi i dades

    a/des d'un dispositiu
  - Sincronització
- Espai de memòria mapejada
  - Utilitza adreces específiques per accedir al dispositiu
  - Accés només permès des del SO



Unitats de processament especial per IA i Dades

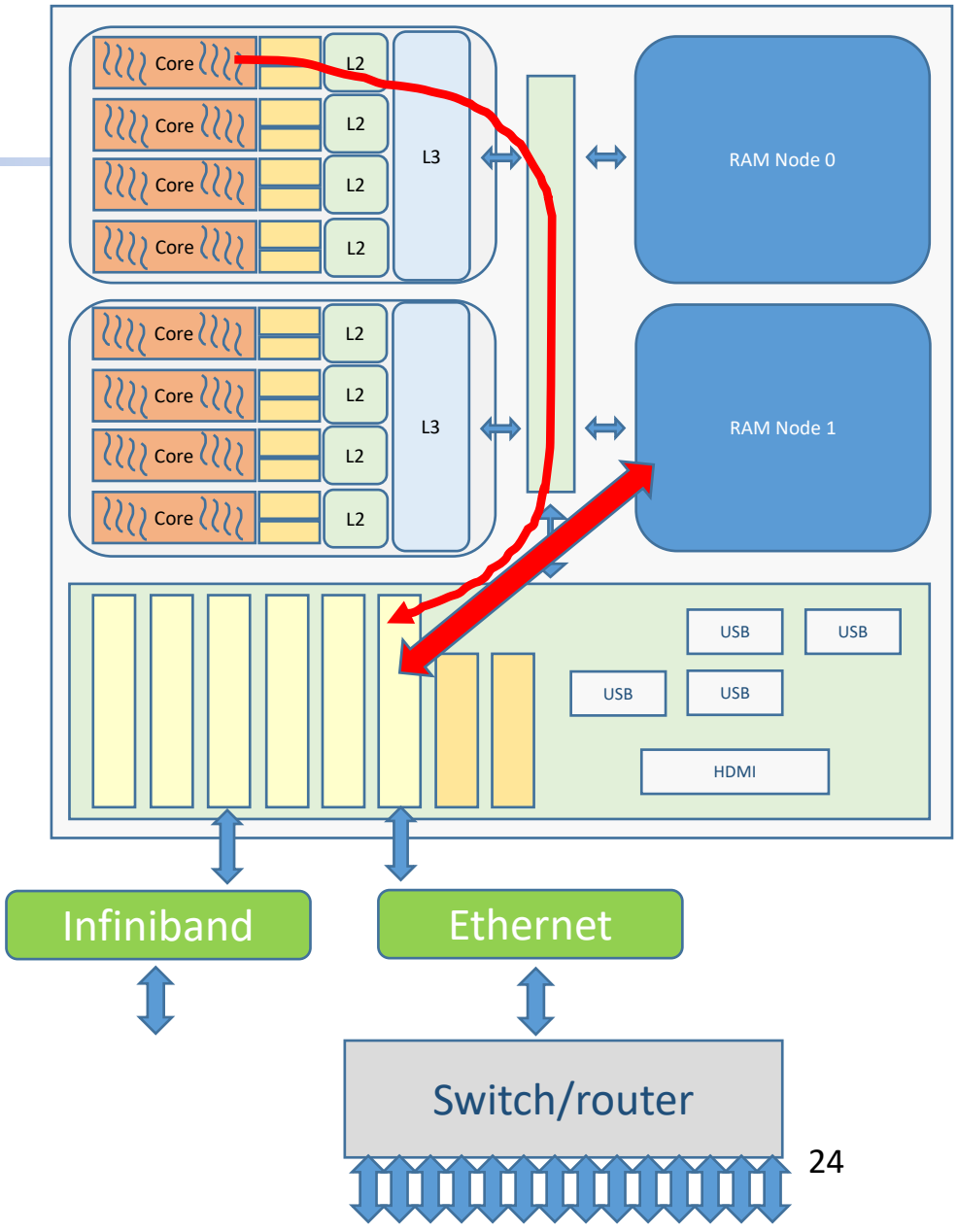
- GPU ve de Graphical Processing Unit
  - Algunes inclouen “Tensor Cores”
    - Hardware especial per accelerar la computació de Tensors

- GPUs pot afrontar necessitats de l’IA i processament de Dades
  - Necessitats de molta amplada de banda de memòria
    - Aprofita l’optimització de l’amplada de banda de **PCIe**
      - Varis GB/s
  - Altes capacitats de computació
    - Paral·lelisme a nivel de thread molt alt
      - Milers de nuclis (cores)
  - Altres cursos de GIA expliquen com utilitzar-ho en codi

- Hi han altres acceleradors per IA i processament de dades
  - E.g. FPGA, ASIC, TPU

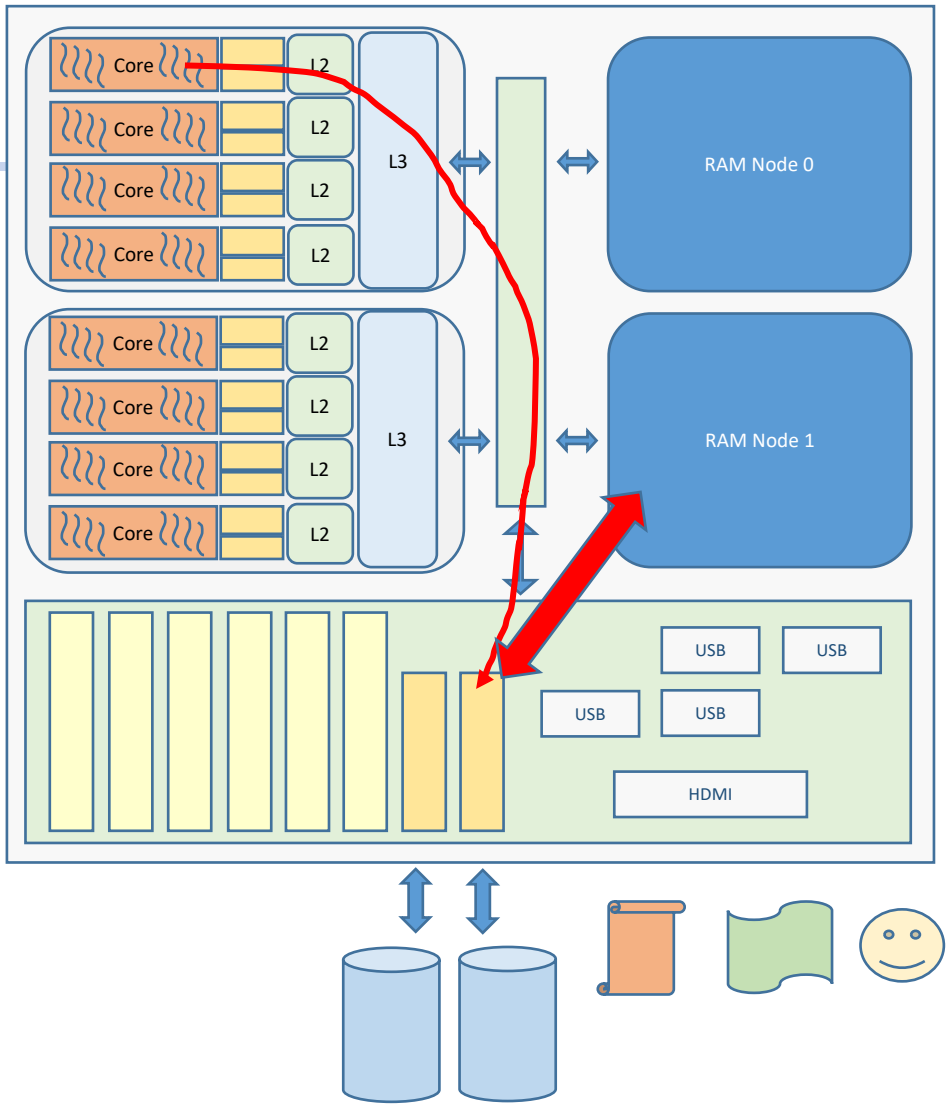
Xarxa

- Enviar/rebre informació a
  - Servidors
  - Network-attached disks
- Protocols
  - Low-level – ethernet packet
  - High-level – TCP/IP
- Transferència de dades per DMA



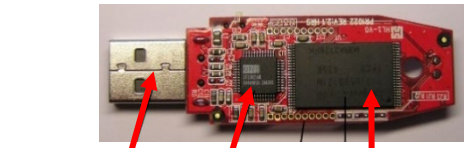
Sata i perifèrics HMI

- Discos
- Video
  - Amb memòria especial de video
- USB
  - Teclat, ratolí...

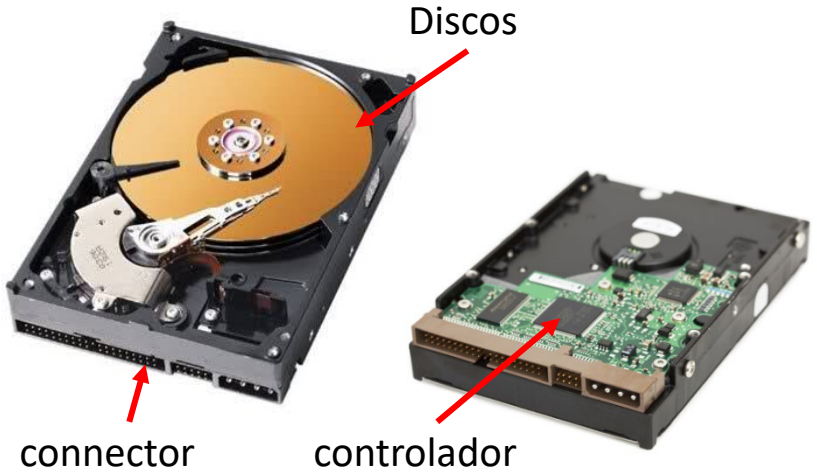


Discos

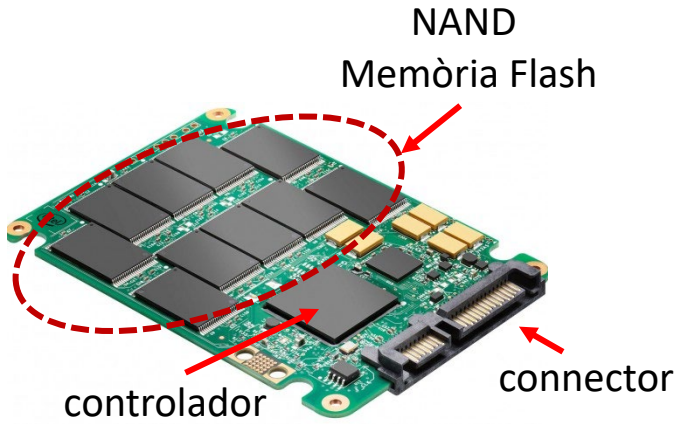
- Memòria no volàtil
  - Emmagatzema valors (dades/instrs) inclús quan no hi ha energia
    - Hi han altres memòries no volàtils a més a més del disc (e.g. ROM)
- Components similars vs diferents
  - Impacte en el rendiment i la capacitat



USB Drive



Hard Disk Drive



Solid State Drive (SSD)

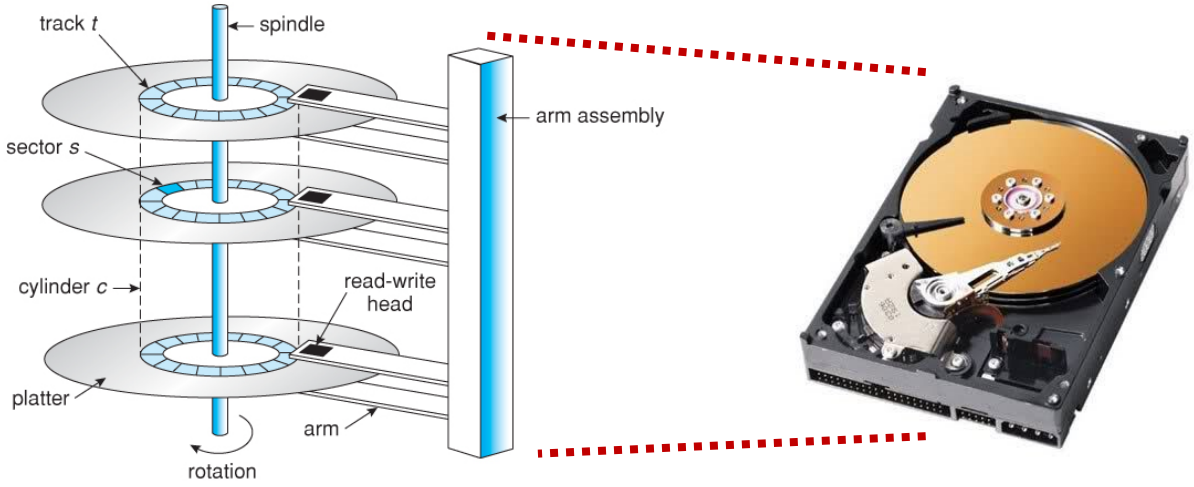
Com es guarden físicament els valors en els discos?

- Qualsevol dispositiu d'emmagatzematge ha d'organitzar la memòria
  - E.g.: DVD, hard-disk, pen-drive, etc.
- **Capacitat**: Nombre màxim de Bytes que pot guardar en un disc

$$\text{Capacity} = \frac{\#\text{ bytes}}{\text{sector}} \times \frac{\text{average }\#\text{ sectors}}{\text{track}} \times \frac{\#\text{ tracks}}{\text{surface}} \times \frac{\#\text{ surfaces}}{\text{platter}} \times \frac{\#\text{ platters}}{\text{disk}}$$

- **Sector**: L'unitat més petita d'emmagatzematge que pot ser llegida/escrita
  - **Definit per hardware**
  - Mida fixa (normalment 512 Bytes)

Alguns paràmetres impacten en el rendiment

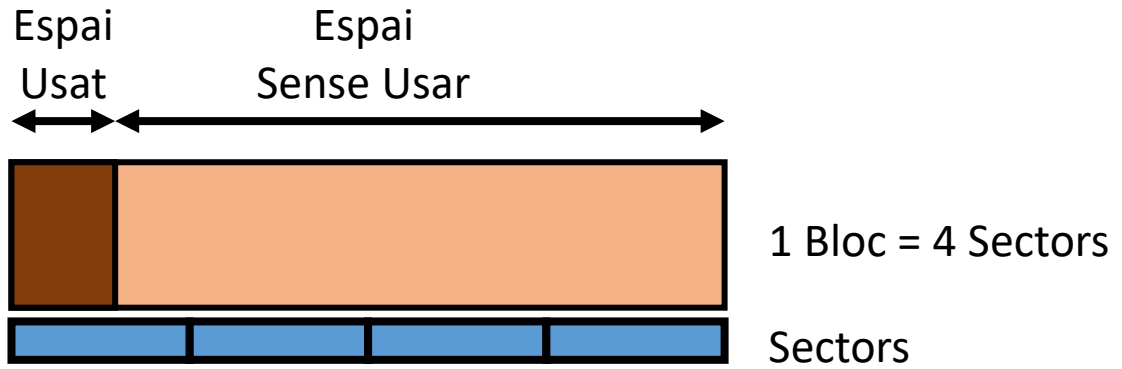


Velocitat: Revolucions Per Min.
e.g. 5400 RPM-15000 RPM
Max Amplada de Banda: Bytes Per Sec.
e.g. pocs MB/s – cents MB/s
(Mbps **no és el mateix que** MBps)

Com s'organitza un dispositiu de storage?

- **Bloc**: Un grup de sectors (l'unitat més petita per assignar memòria)
  - **Definit pel SO (quan es dona format al dispositiu)**

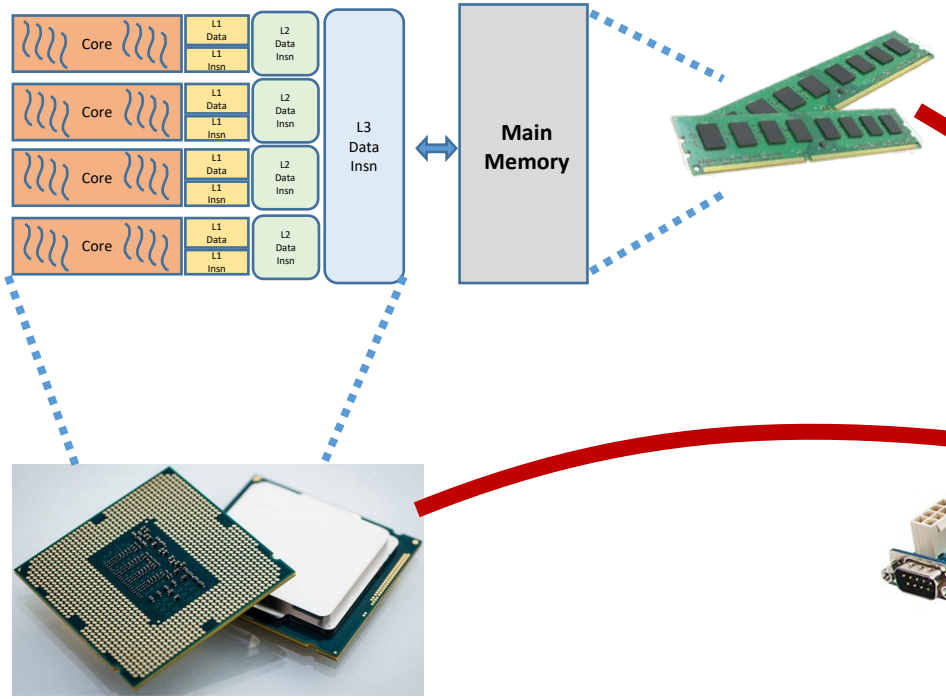
- Però, quin és la millor mida de bloc????
  - Si és probable usar fitxers grans…
    - Blocs grans
  - Si és probable usar fitxers petits…
    - Blocs petits

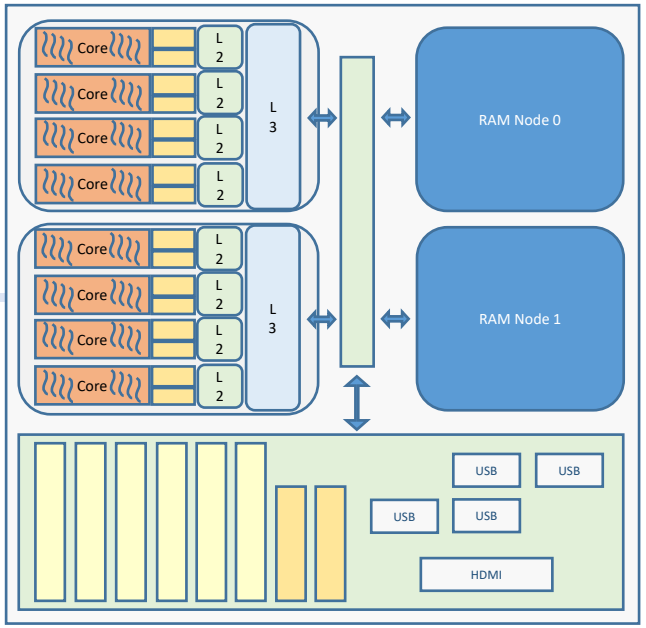


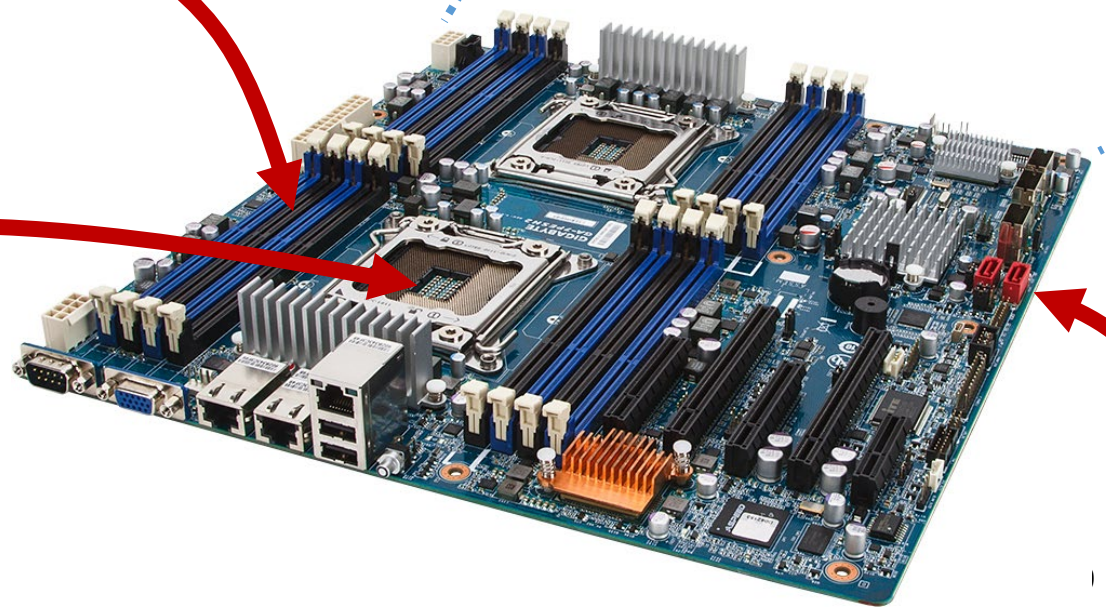
- Quin és l'impacte d'una mida incorrecte de bloc???
  - Massa gran: **fragmentació** (i.e. malgastar espai)
  - Massa petit: pèrdua de rendiment per fer masses accesos al dispositiu

- Blocs grans tenen l'advantatge d'una de les formes de **Localitat** (localitat espacial)
  - Parlarem sobre localitat en transparències posteriors…

Visió Global de l'exemple

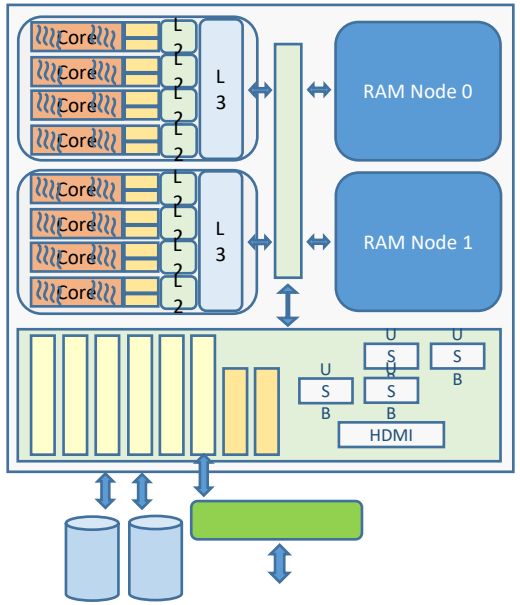




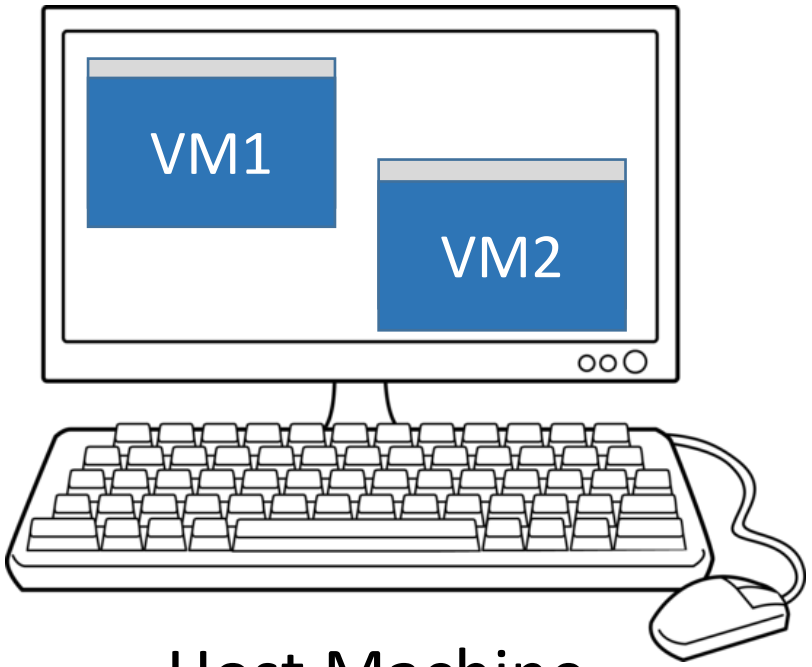


Virtual Machine (VM)

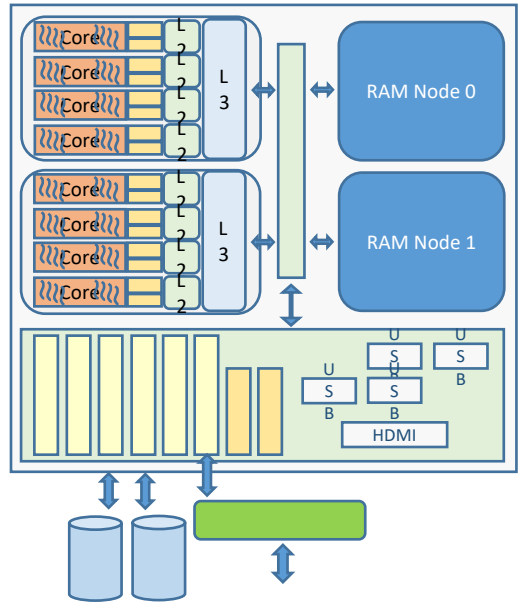
- Software que virtualitza tot l'entorn (recursos físics i SO) d'un ordinador
  - Recursos virtualitzats poden ser emulats (simula el comportament real) o enllaçats a recursos reals de la máquina amfitrió (host)
  - Es poden executar varis VMs alhora en el mateix host. Cadascún pot usar un SO diferent



VM1



Host Machine



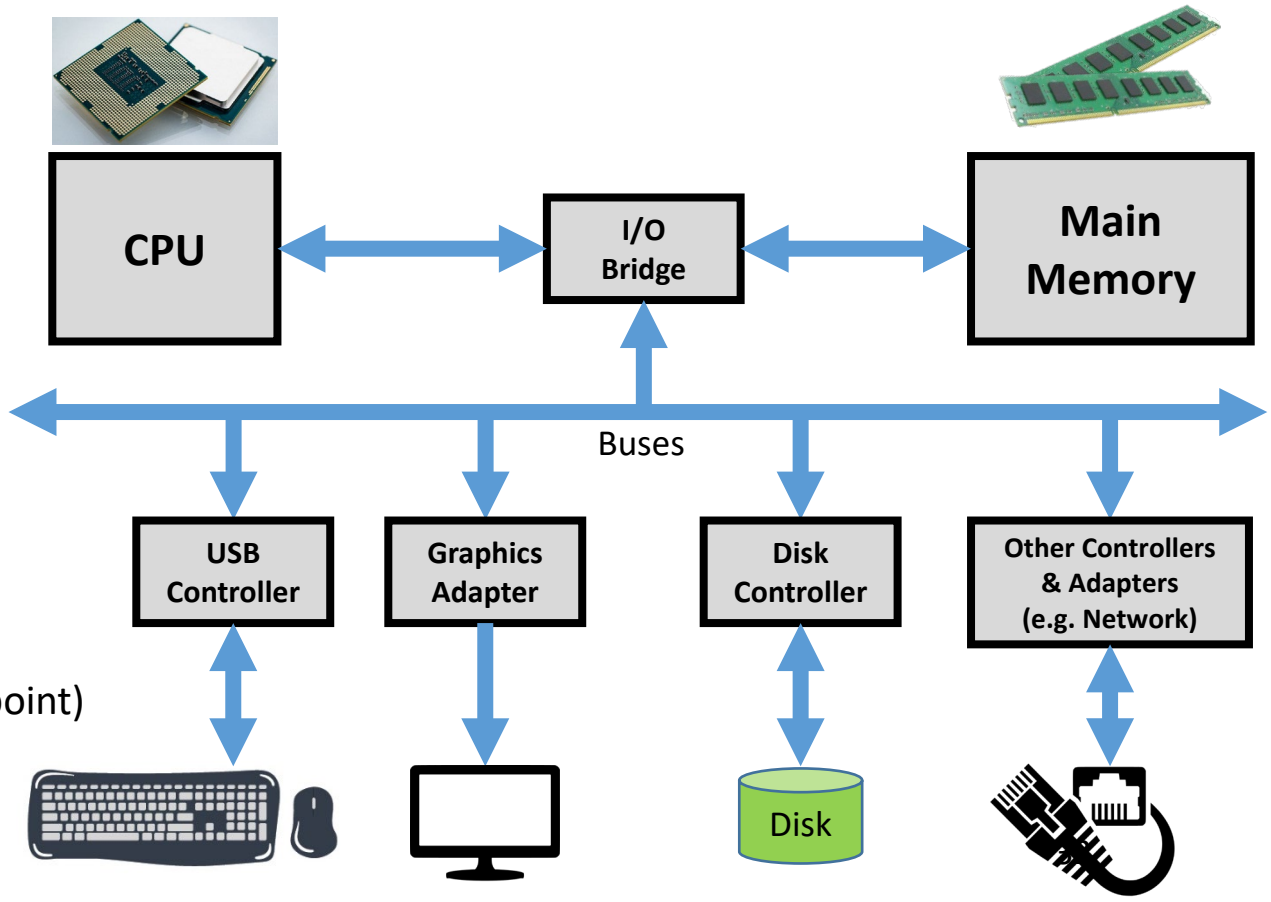
VM2

Index

- Aquest tema té tres parts
  1) **Components Bàsics i Jerarquia de Memòria**
  2) Fluxe d'execució i colls d'ampolla

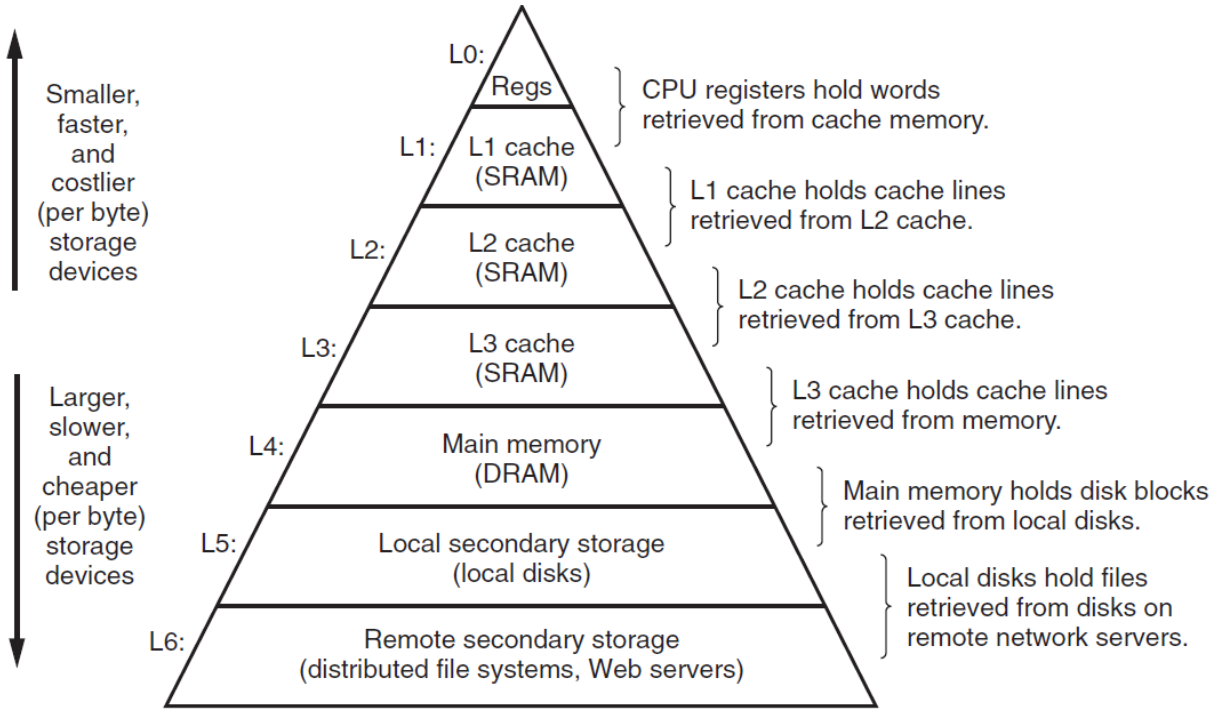
Bases...

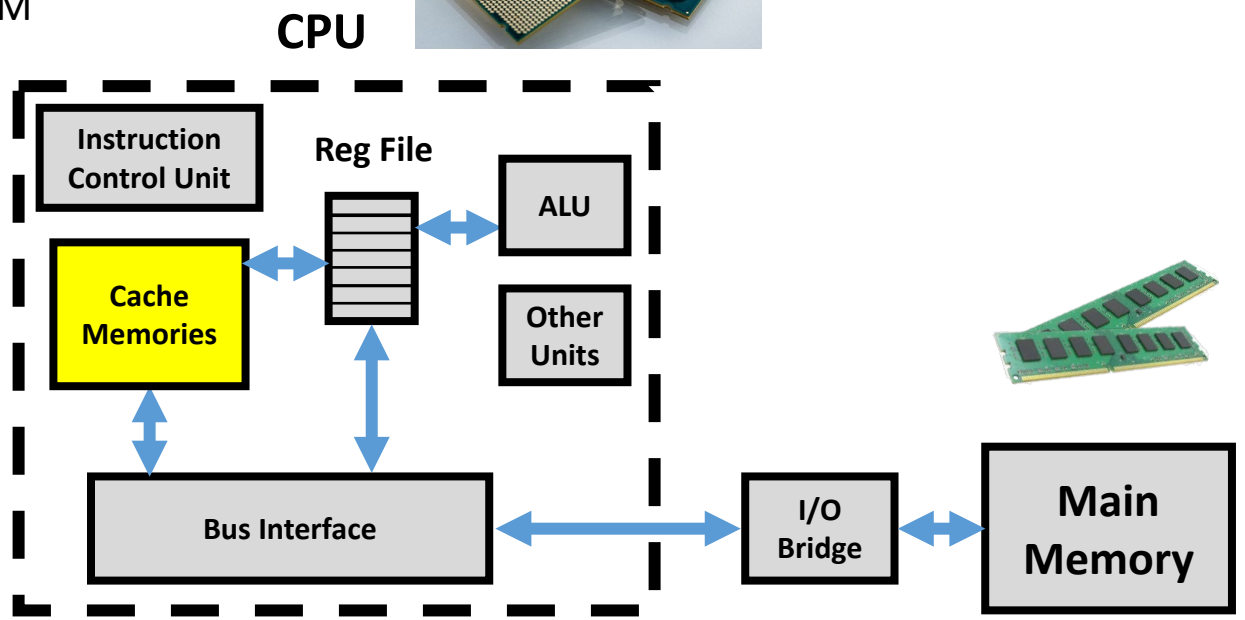
- La CPU només pot accedir a
  - Registres
  - Memòria
- No pot accedir directament a altres
  - Instrs i dades s'han de carregar en memòria per poder ser
    - Arquitectura Von Neumann
- Quan un programa es llença …
  - Es reserva memòria
  - Carrega l'executable des del disc a memòria
  - La CPU començ a executar la primera instr (i.e. entry point)



Jerarquia de Memòria

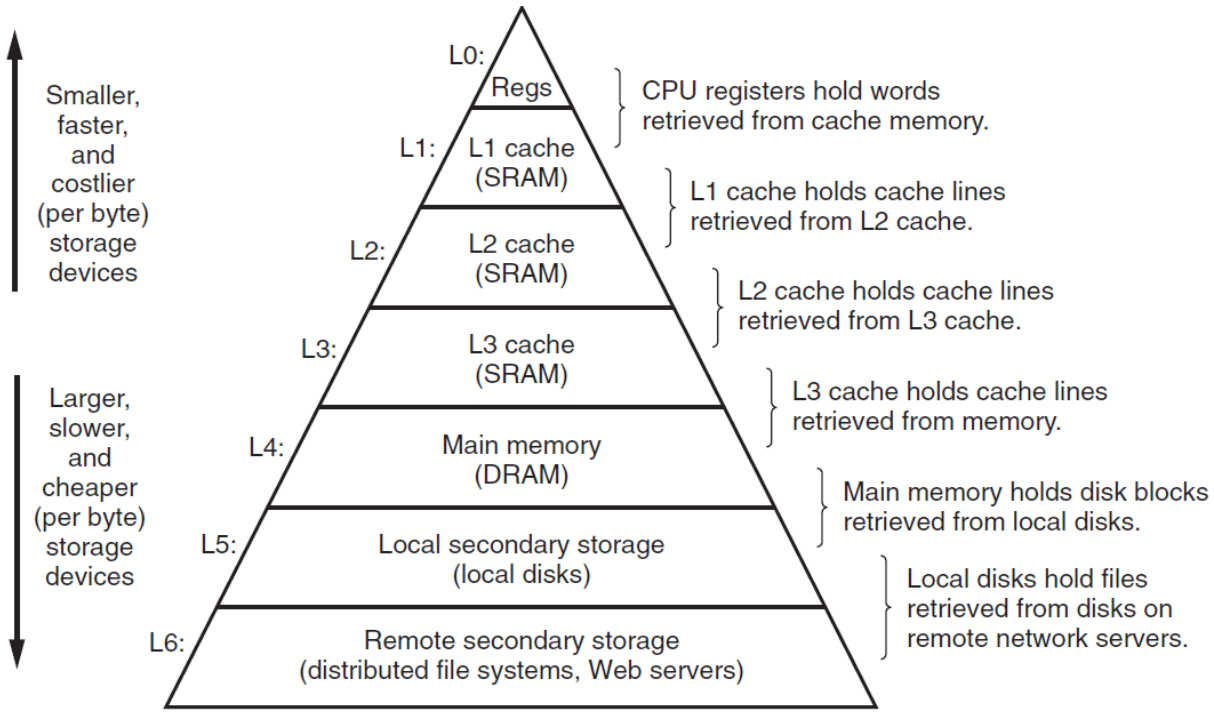
- **RAM**: L'unitat básica de storage és una cel·la (1 bite per cel·la); l'unitat bàsica adreçable és una supercel·la (1 Byte)
  - Static RAM (SRAM) per la **jerarquia de cache**
  - Variants de Dynamic (DRAM) per la **memòria principal**
    - E.g. SDRAM, DDR(2, 3, 4) SDRAM
  - SRAM és més ràpida (~10x) i més cara (~100x) que DRAM



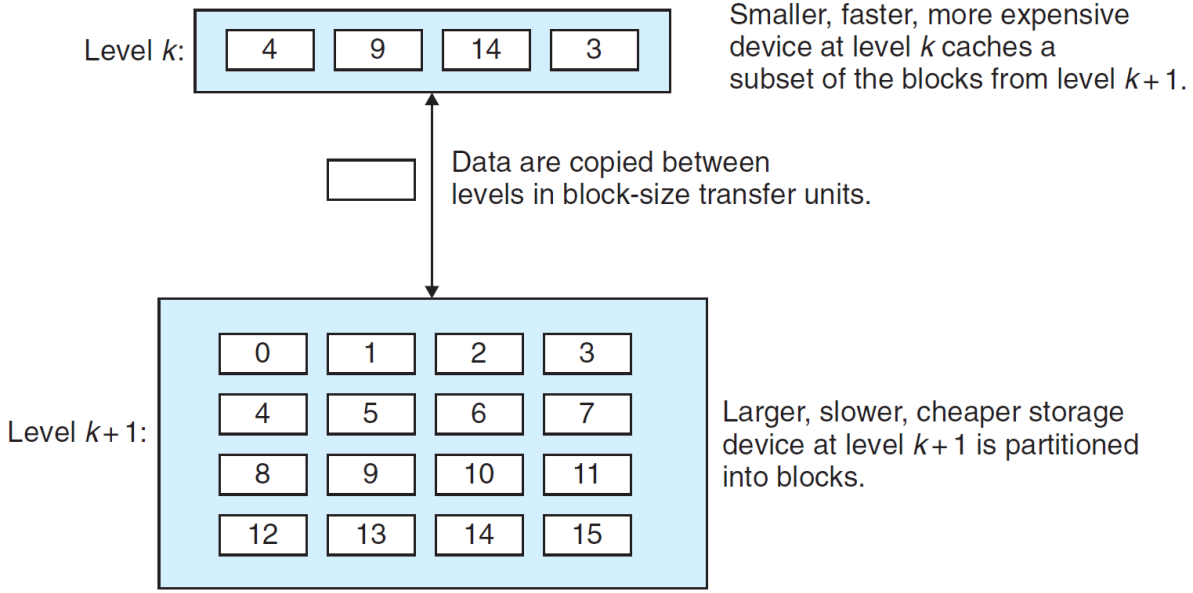


Memory Hierarchy

- **RAM**: L'unitat básica adreçable de storage és una supercel·la (1 Byte) composta per 8 cel·les (1 bit cada cel·la)
  - Static RAM (SRAM) per la **jerarquia de cache**
  - Variants de Dynamic (DRAM) per la **memòria principal**
    - E.g. SDRAM, DDR(2, 3, 4) SDRAM
  - SRAM és més ràpida (~10x) i més cara (~100x) que DRAM



- **Caching:** utilitza un dispositiu d'emmagatzematge més petit, però més ràpid per portar elements guardats en memòries més grans i lentes
  - Normalment els processadors van des de L0 (registres) fins a L3
  - Explota la **Localitat…**



Localitat

- El principi de localitat és la tendencia a accedir elements que estan propers a altres en l'espai o temps
  - **Localitat Espacial**: des d'una adreça de memòria en concret, posteriorment s'accedeix a adreces en posicions properes
  - **Localitat Temporal**: des d'una adreça de memòria en concret, posteriorment s'accedeix a la mateixa adreça en un curt periode de temps

- Si hi ha una alta localitat espacial…portar no només els bytes sol·licitats, sinó també els de les posicions adjacents

- Si hi ha una alta localitat temporal…mantener els bytes accedits en nivells alts de la jerarquia de memoria (cache)

- Els diferents tipus de paral·lelisme també explota la localitat espacial
  - E.g. SIMD

Exemple de Localitat

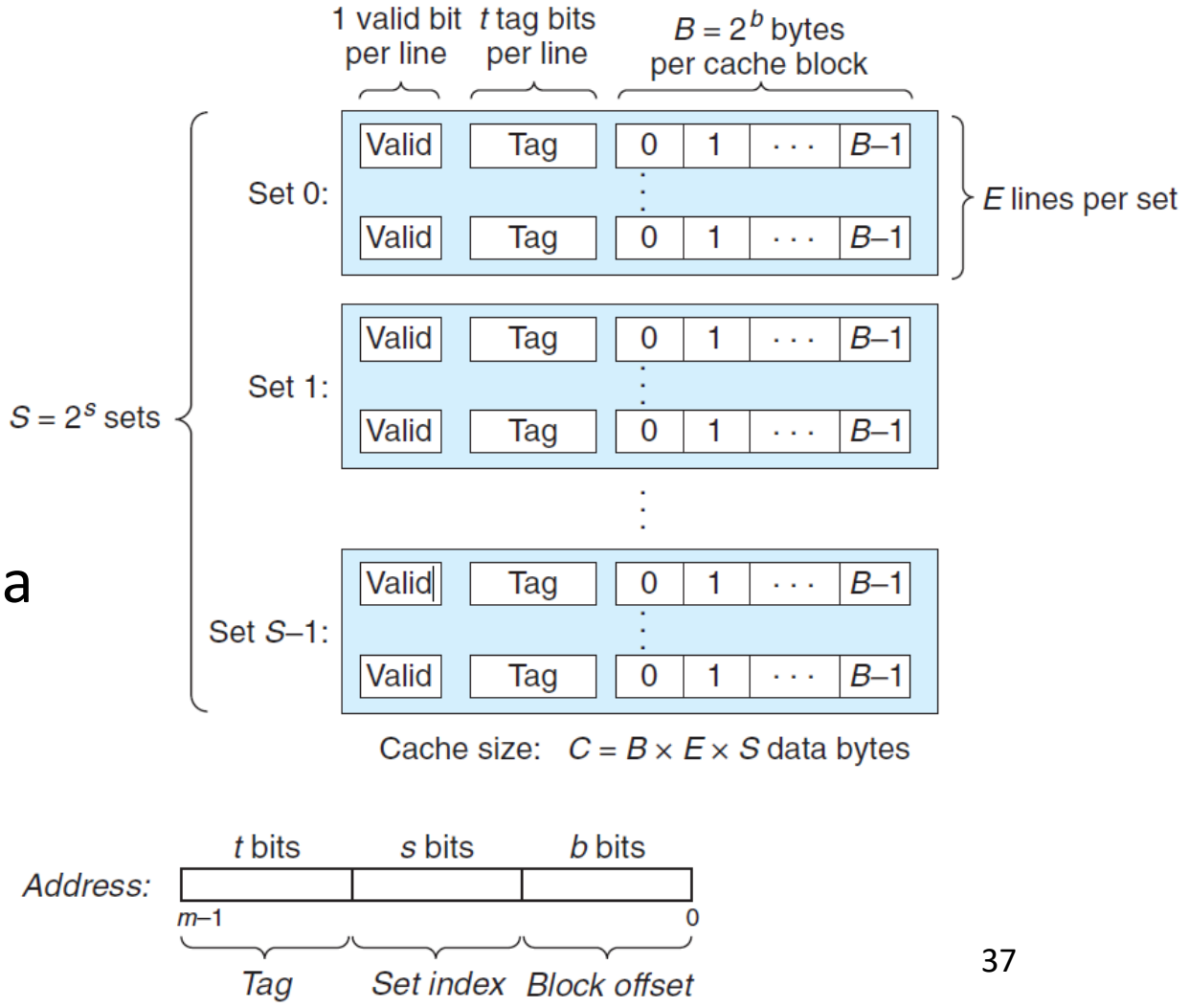
- Quins tipus de localitat identifiques en el següent fragment de codi...
  - ...en referències a instruccions?
  - …en referències a dades?

```
func (vec, n)
{
        tmp = 0;
        for (i=0; i<n; i++)
                tmp += vec[i];
        return tmp;
}
```

Organització de la cache

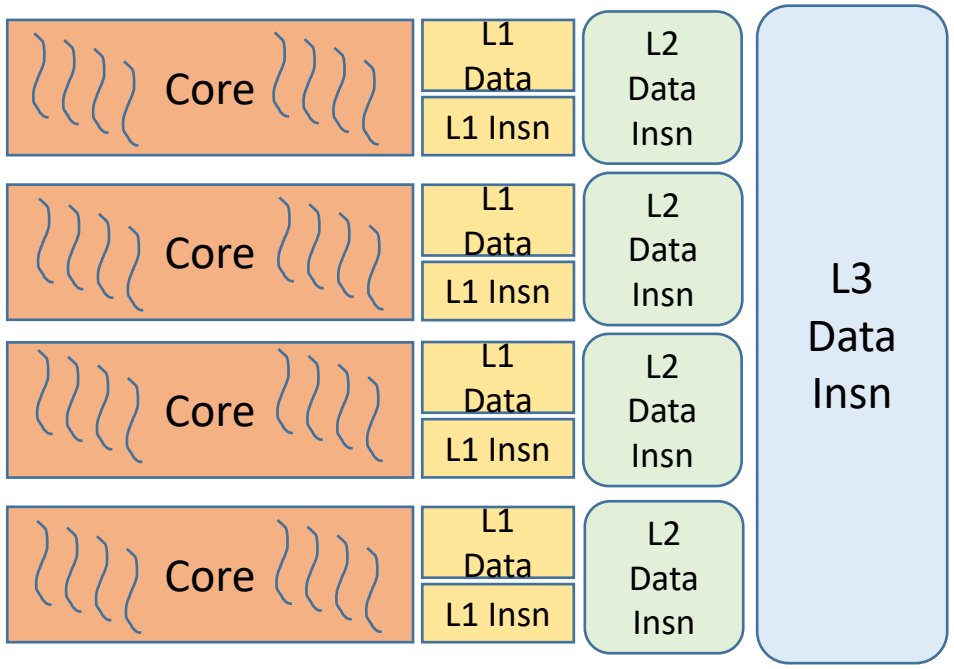
- Una cache consisteix en…
  - Un array de conjunts
  - Un conjunt (set) té una o més línies
  - Una línia té…
    - Un bit de validessa
    - Uns pocs bits per l'etiqueta (tag)
    - Varios bytes de dades

- Una adreça de memòria es decomposa
  - Una adreça de *m* bits
    - t bits pel tag
    - s bits pel set
    - b bits per seleccionar el Bloc (offset)



Jerarquia Cache

- **Instruction** cache, **data** cache i **unified** cache



Exemples

AMD

Private L1
Shared L2
Shared L3

Intel Core i7

Private L1
Private L2
Shared L3

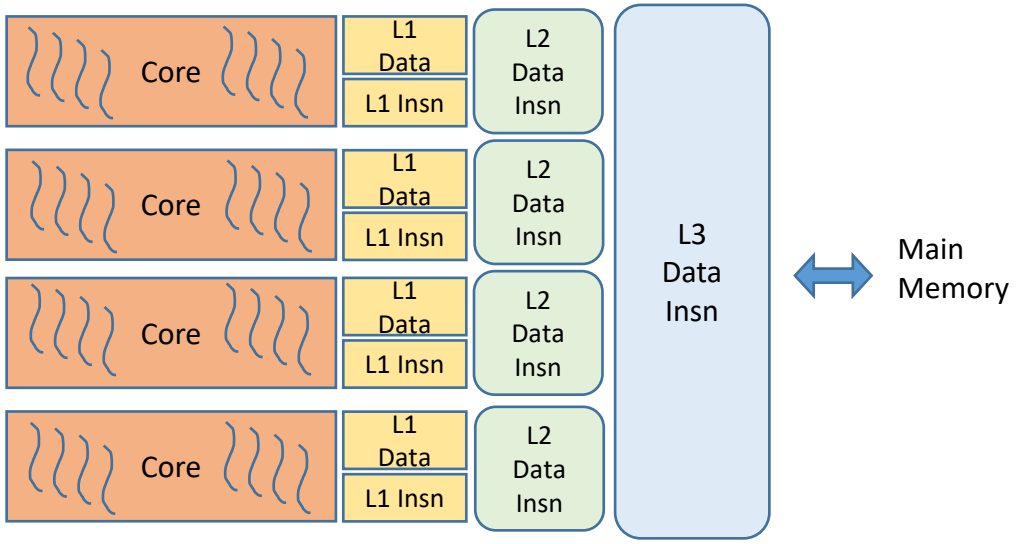
Cache type	Access time (cycles)	Cache size (C)	Assoc. (E)	Block size (B)	Sets (S)
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	10	256 KB	8	64 B	512
L3 unified cache	40–75	8 MB	16	64 B	8,192

Estratègies d'escriptura

- Què passa si hi ha un encert?
  - Write-through
    - Tant aviat com hi ha una escriptura en el nivell i-èssim de cache, s'envia la modificació al nivell i+1
    - Senzill d'implementar, però incrementa el tràfic en el bus
    - Es pot utilizar un bufer (write-buffer) per millorar el rendiment d'actualització de memòria
    - Les errades de lectura són menys costosos perquè no impliquen una escriptura
  - Write-back
    - Quan es fa una escriptura en el nivell i-èsim de cache, NO s'envia al nivell i+1. S'endarrereix tant com espugui
    - Menys transferències → per tant hi haurà més amplada de banda disponible per dispositius d'E/S
- Què passa si hi ha una errada?
  - Write-no-allocate (a.k.a. write around)
    - Les dades no es carreguen en la cache, sinó que s'escriuen directament al següent nivell de cache
    - Les dades només es carreguen quan hi han errades en lectures
  - Write-allocate (a.k.a. fetch on write)
    - Les dades es carreguen en la cache, seguides d'una operación d'escriptura que encert
    - Intenta explotar la localitat espacial d'escriptures, però incrementa el tràfic
- Normalment…
  - …nivells més baixos (e.g. L1) de cache són write-through; nivells més alts (e.g. L2) són write-back
  - …quan s'utilitza write-through també s'utilitza write-no-allocate; write-back també va amb write-allocate

Requisits: Consistència & Coherència

- Els accessos fan referència a posicions de memòria, no a posicions de la cache
  - Les memòries cache són gestionades de manera transparent pel hardware
  - **Coherència en Memòria**: qualsevol lectura des de qualsevol CPU a una posició en concret de memòria, retorna l'últim valor escrit en aquella posició
  - **Consistència en Memòria**: assegura que les escriptures a diferentes posicions de memòria es veuran en l'ordre correcte des de totes les CPUs
- Hi ha protocols (e.g. snoopy) per assegurar aquests requisits




```
void add_vectors( float * c, float * a, float * b )
{
  int i;
  for (i=0; i < N; i++) c[i] = a[i] + b[i];
}
```

Errada de Cache (Cache Miss)

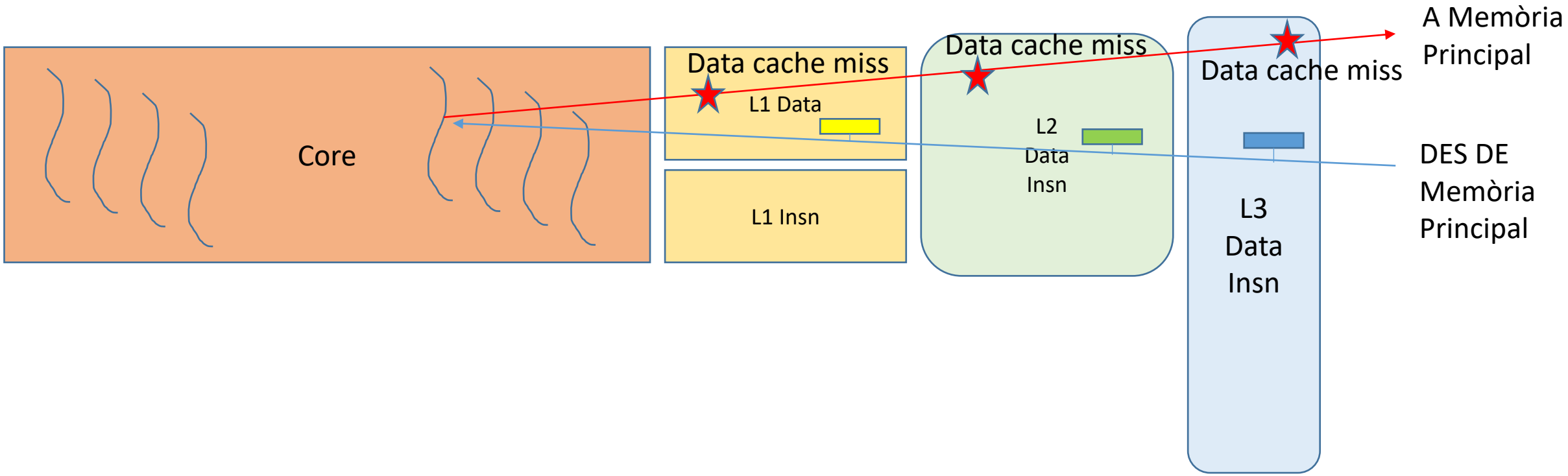
- Penalització per errada: temps addicional degut a una errada en el nivell-k de cache

- Tipus d'errades de Cache
  - **Cold or compulsory (obligatoria/inicial) miss**…
    - Passa quan la cache està buida
  - **Conflict (conflicte) miss**…
    - Succeix quan varis elements coincideixen en el mateix bloc
      - E.g. (i mod 4)…0, 4, 8, 12, 16 … tots aquests elements coincideixen en el bloc 0
  - **Capacity (capacitat) miss**…
    - Succeix quan el **conjunt de treball** (i.e. blocs actius de cache) és més gran que el nombre de blocs de cache

- Impacte directe de...
  - **Política d'emplaçament**: determina on va el bloc que es carrega
  - **Política de reemplaçament**: determina quin bloc es fa fora per carregar un de nou

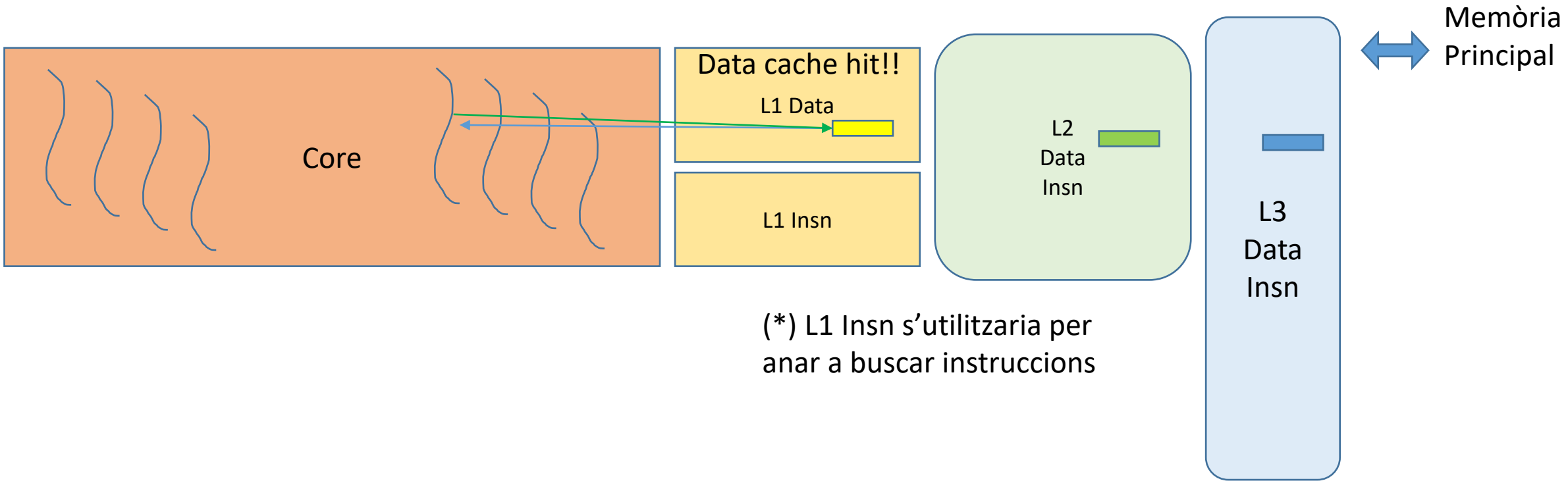
Accés Detallat a Memòria (MISS: errada)

- Instrucció LOAD: carregar dades que NO estan en les caches
  - Similar a anar a buscar instruccions que NO estan en les caches



Accés Detallat a Memòria (HIT: encert)

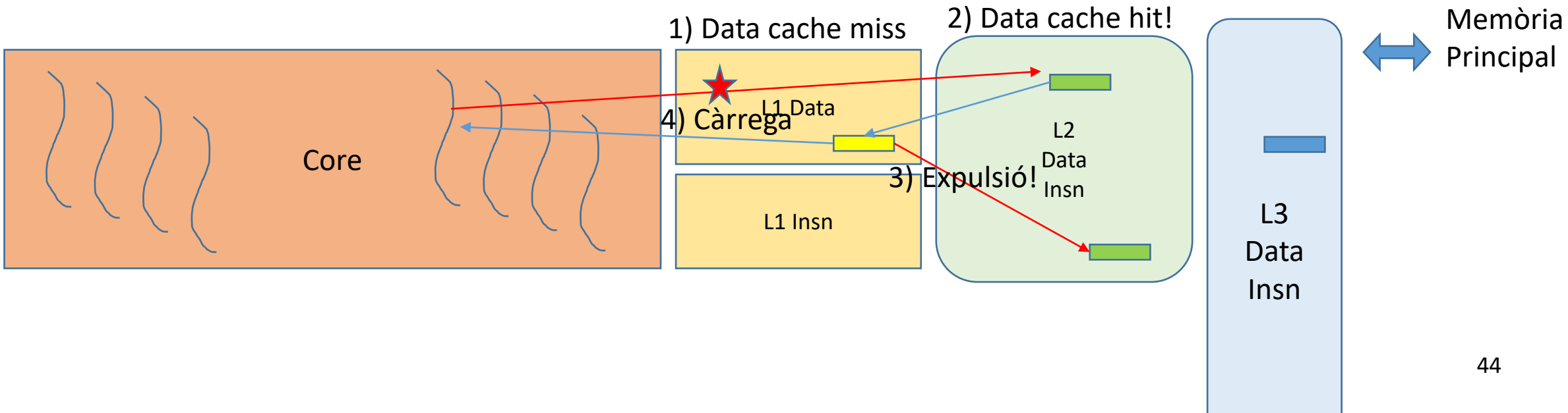
- Instrucció LOAD: carregar dades que estan en la L1 Dades
  - Similar a anar a buscar instruccions que estan en la L1 Insn (*)



(*) L1 Insn s'utilitzaria per anar a buscar instruccions

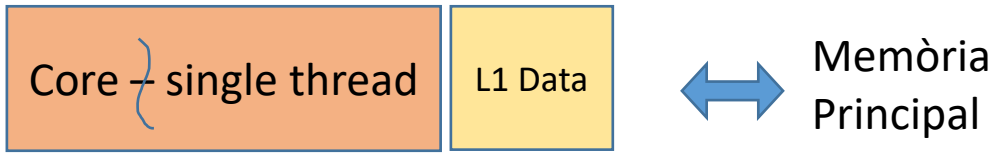
Accés Detallat a Memòria (Eviction: expulsió)

- La gestió de la cache és una característica Hw complexa
  - Què passa quan la cache està plena...
    i s'han d portar més dades/instrs?
    - Cache eviction... Els valors que fa més temps han sigut accedits poden ser expulsats al següent nivell de cache

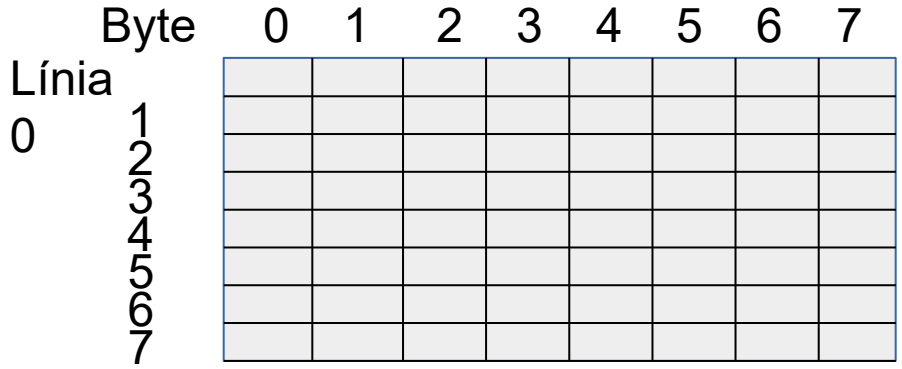


Alguns exemples

- Tenim un ordinador senzill amb la següent estructura interna:



  - Level 1 cache
    - 8 línies de cache
    - 8 bytes per línia de cache
    - 64 bytes en total



Exemple 1

- Executem el següent programa:

```
double V[16];       // using addresses to 0x100 to 0x17f
int i;              // using a CPU register

for (i=0; i < 16; i++)
    V[i] = 0.3;
```

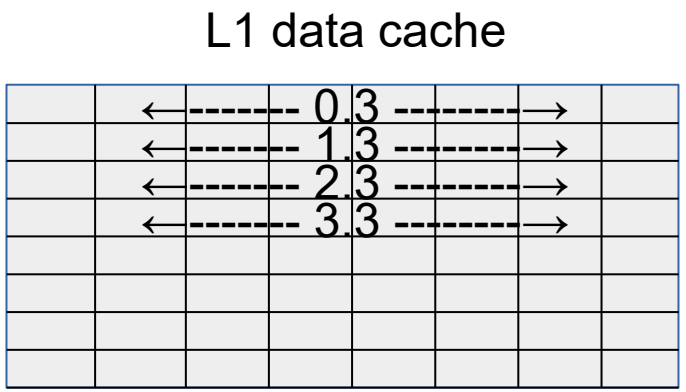
- Determina els ***hits*** i ***misses*** i ***errades de capacitat*** que succeeixen en la cache de dades L1
- Assumim que no hi ha interacció amb una possible cache d'instruccions L1

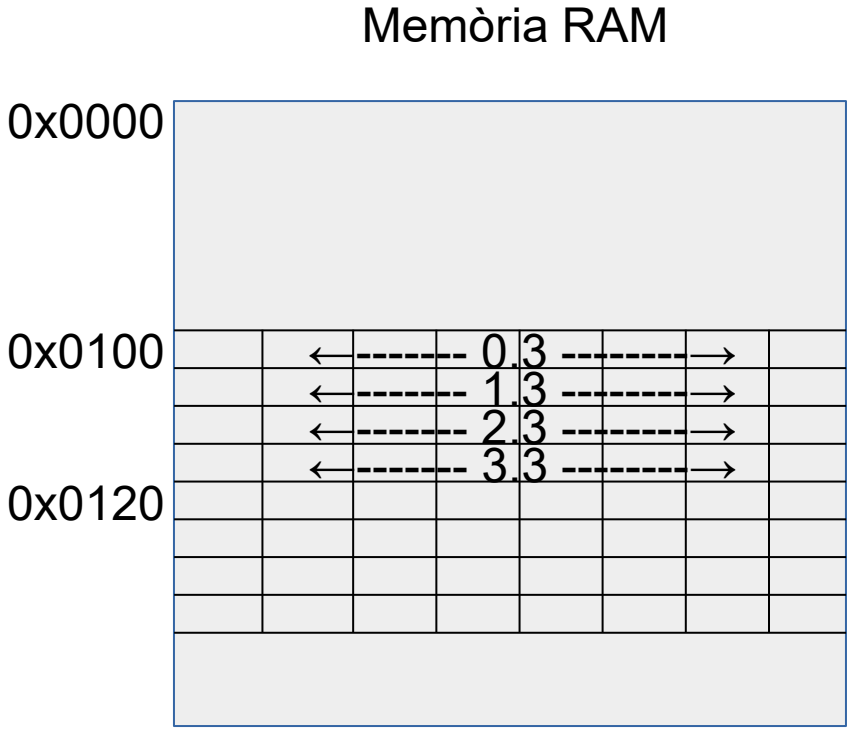
Exemple 1

- Executem el següent programa:

```
double V[16];       // using addresses 0x100 to 0x17f
int i;              // using a CPU register

for (i=0; i < 16; i++)
    V[i] = 0.3 + (double) i;
```

i = 0	–	–
V[0] = 0.3	0x0100	miss
i++	–	–
V[1] = 1.3	0x0108	miss
i++	–	–
V[2] = 2.3	0x0110	miss
i++	–	–
V[3] = 3.3	0x0118	miss
i++	–	–



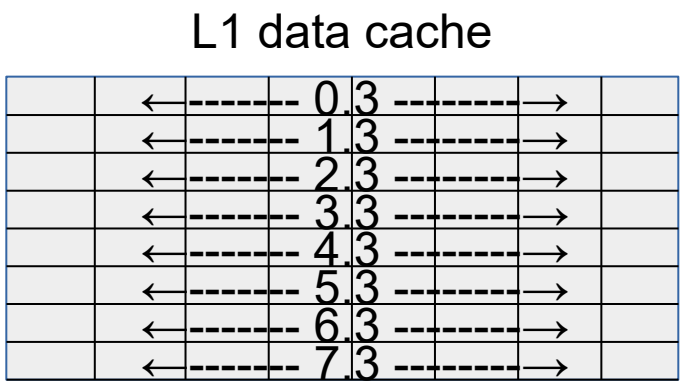


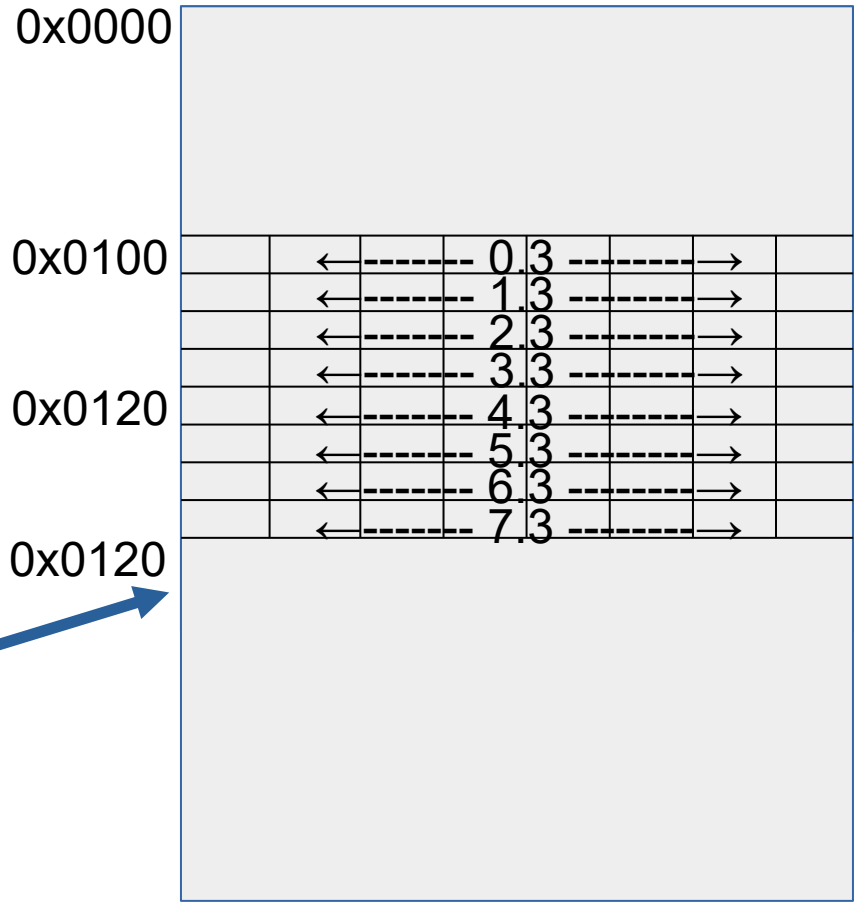
Exemple 1

- Executem el següent programa:

```
double V[16];       // using addresses 0x100 to 0x17f
int i;              // using a CPU register

for (i=0; i < 16; i++)
    V[i] = 0.3 + (double) i;
```

V[4] = 4.3	0x0120	miss
i++	–	–
V[5] = 5.3	0x0128	miss
i++	–	–
V[6] = 6.3	0x0130	miss
i++	–	–
V[7] = 7.3	0x0138	miss
i++	–	–
V[8] = 8.3	0x0140	???



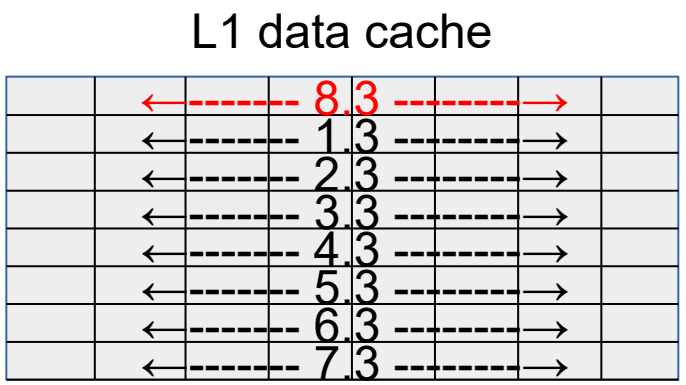


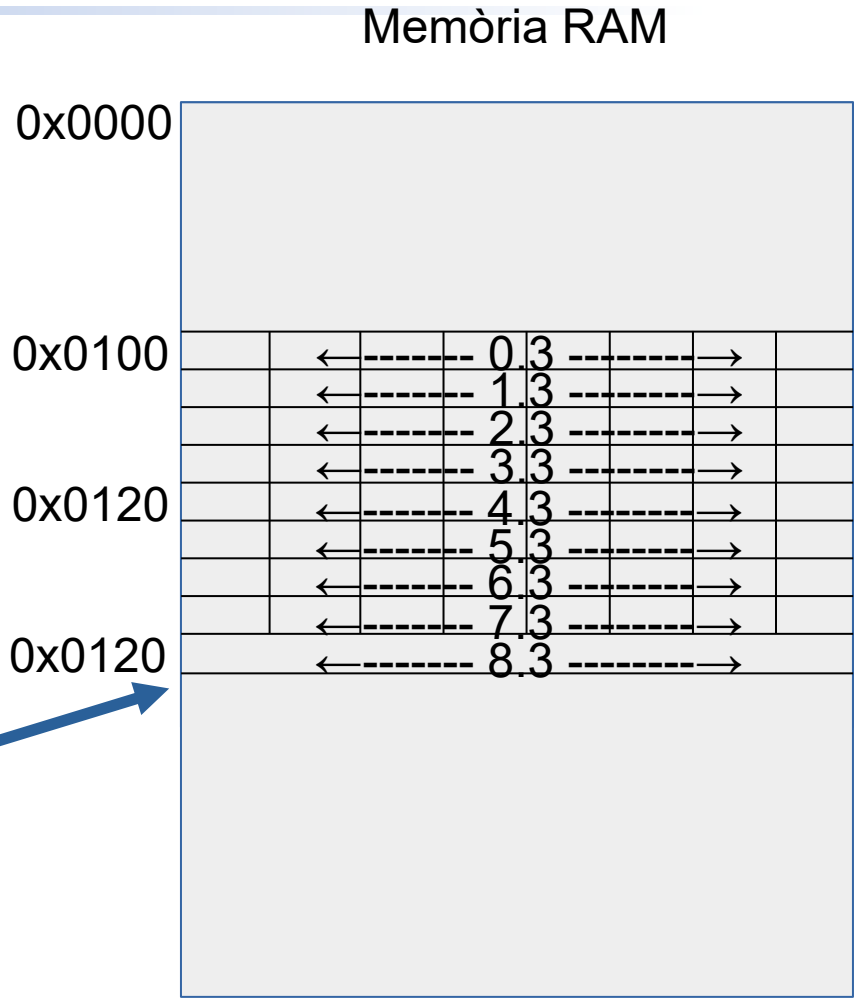
Exemple 1

- Executem el següent programa:

```
double V[16];       // using addresses 0x100 to 0x17f
int i;              // using a CPU register

for (i=0; i < 16; i++)
    V[i] = 0.3 + (double) i;
```

V[4] = 4.3	0x0120	miss
i++	–	–
V[5] = 5.3	0x0128	miss
i++	–	–
V[6] = 6.3	0x0130	miss
i++	–	–
V[7] = 7.3	0x0138	miss
i++	–	–
V[8] = 8.3	0x0140	miss & errada de capacitat



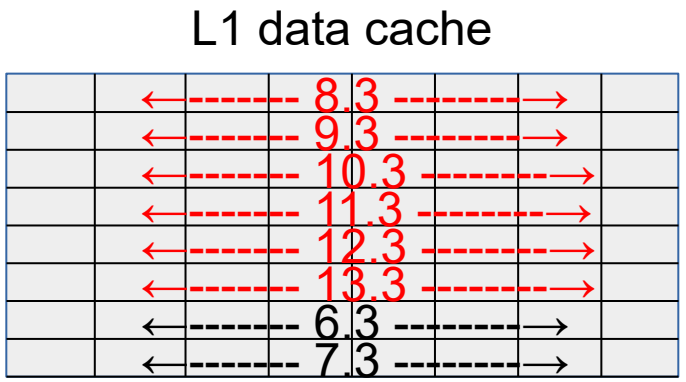


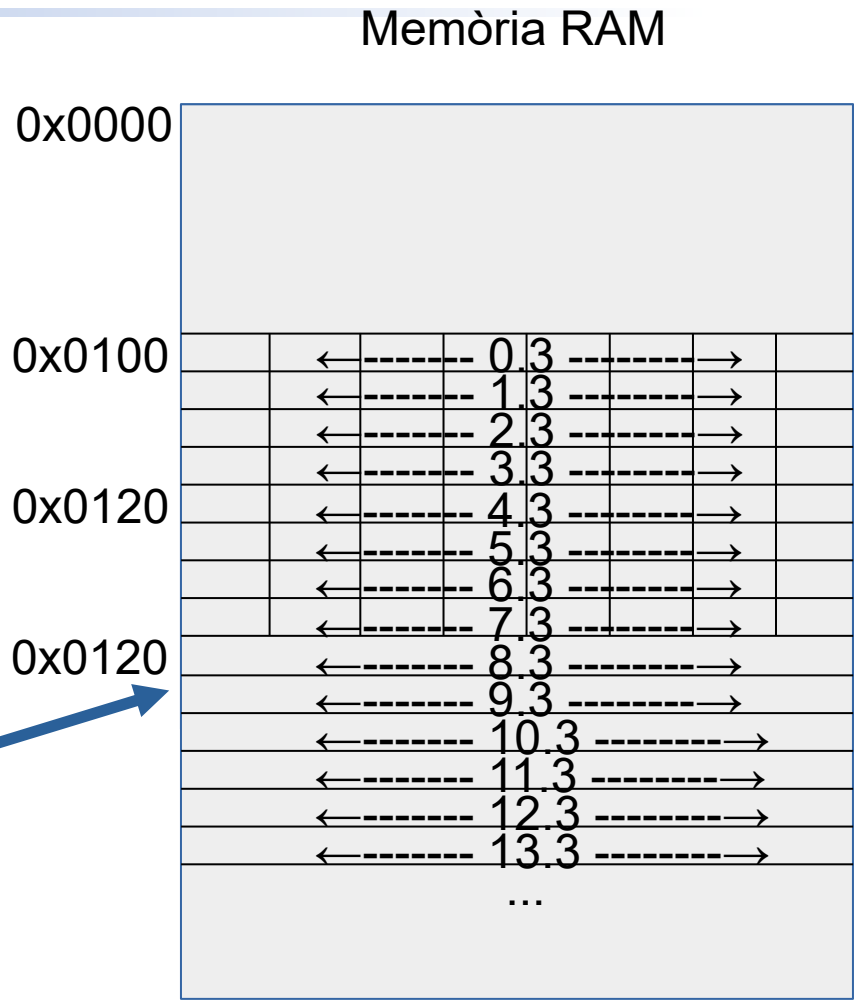
Exemple 1

- Executem el següent programa:

```
double V[16];       // using addresses 0x100 to 0x17f
int i;              // using a CPU register

for (i=0; i < 16; i++)
    V[i] = 0.3 + (double) i;
```

V[9] = 9.3	0x0148	miss & e.c.
i++	–	–
V[10] = 10.3	0x0150	miss & e.c.
i++	–	–
V[11] = 11.3	0x0158	miss & e.c.
i++	–	–
V[12] = 12.3	0x0160	miss & e.c.
i++	–	–
V[13] = 13.3	0x0168	miss & errada de capacitat
…		



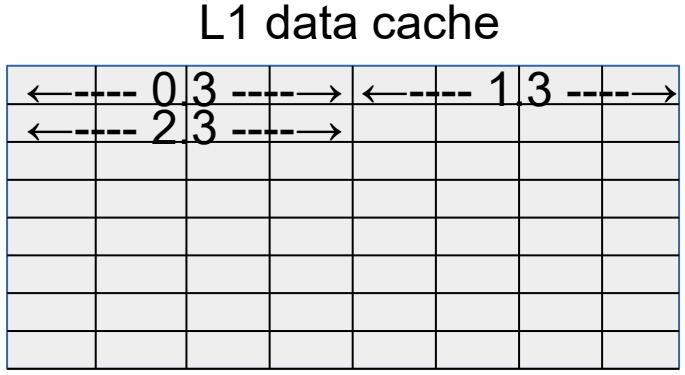


Exemple 2

- Ara canviem el programa per utilitzar «float» de precisió senzilla:

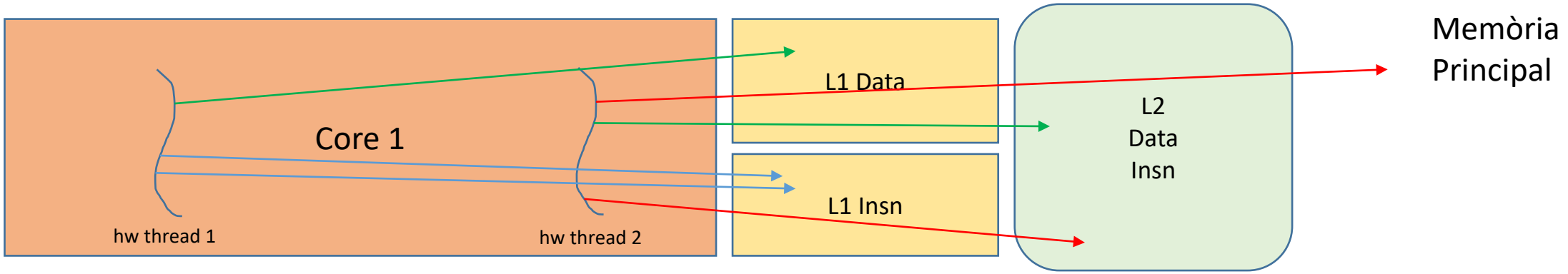
```
float V[32];          // using addresses 0x100 to 0x17f
int i;                // using a CPU register

for (i=0; i < 32; i++)
    V[i] = 0.3 + (float) i;
```

- Quin és el comportament del programa respecte a l'ús de la cache (hits, misses, errades de capacitat)?
- ...i respecte la memòria RAM?



Exemple 3

Assumim que l'arquitectura d'aquest processador executa un únic procés amb varis software threads



SW1 corrent en HW1
SW2 corrent en HW2

Exemple 3

- Shared L1, line cache size 128 (0x080) bytes
- Shared L2, line cache size 256 (0x100) bytes

Ordre d'execució	Core	HW Thread	Instr/dada	@ Memòria	L1 Insn	L1 Data	L2 Insn/Data
1	1	1	Instrucció	0x001000			
2	1	1	Dada	0x002000			
3	1	1	Instrucció	0x001008			
4	1	1	Dada	0x002080			
5	1	1	Instrucció	0x001010			
6	1	1	Dada	0x002100			
7	1	1	Instrucció	0x001018			
8	1	1	Instrucció	0x001020			
9	1	1	Data	0x002180			

Exemple 3

- Shared L1, line cache size 128 (0x080) bytes
- Shared L2, line cache size 256 (0x100) bytes

Ordre d'execució	Core	HW Thread	Instr/dada	@ Memòria	L1 Insn	L1 Data	L2 Insn/Data
10	1	2	Instrucció	0x001000			
11	1	2	Dada	0x002000			
12	1	2	Instrucció	0x001008			
13	1	2	Dada	0x002080			
14	1	2	Instrucció	0x001010			
15	1	2	Dada	0x002100			
16	1	2	Instrucció	0x001018			
17	1	2	Instrucció	0x001210			
18	1	2	Data	0x002280			

Exemple 3

- Shared L1, line cache size 128 (0x080) bytes
- Shared L2, line cache size 256 (0x100) bytes

Ordre d'execució	Core	HW Thread	Instr/dada	@ Memòria	L1 Insn	L1 Data	L2 Insn/Data
1	1	1	Instrucció	0x001000	Miss 0x1000	--	Miss 0x1000
2	1	1	Dada	0x002000	--	Miss 0x2000	Miss 0x2000
3	1	1	Instrucció	0x001008	Hit 0x1000	--	--
4	1	1	Dada	0x002080	--	Miss 0x2080	Hit 0x2000
5	1	1	Instrucció	0x001010	Hit 0x1000	--	--
6	1	1	Dada	0x002100	--	Miss 0x2100	Miss 0x2100
7	1	1	Instrucció	0x001018	Hit 0x1000	--	--
8	1	1	Instrucció	0x001020	Hit 0x1000	--	--
9	1	1	Data	0x002180	--	Miss 0x2180	Hit 0x2100

Exemple 3

- Shared L1, line cache size 128 (0x080) bytes
- Shared L2, line cache size 256 (0x100) bytes

Ordre d'execució	Core	HW Thread	Instr/dada	@ Memòria	L1 Insn	L1 Data	L2 Insn/Data
10	1	2	Instrucció	0x001000	Hit 0x1000	--	--
11	1	2	Dada	0x002000	--	Hit 0x2000	--
12	1	2	Instrucció	0x001008	Hit 0x1000	--	--
13	1	2	Dada	0x002080	--	Hit 0x2000	--
14	1	2	Instrucció	0x001010	Hit 0x1000	--	--
15	1	2	Dada	0x002100	--	Hit 0x2100	--
16	1	2	Instrucció	0x001018	Hit 0x1000	--	--
17	1	2	Instrucció	0x001210	Miss 0x1200	--	Miss 0x1200
18	1	2	Data	0x002280	--	Miss 0x2280	Miss 0x2200

Bibliografia

- Computer Systems – A Programmer's perspective (3rd Edition)
  - Randal E. Bryant, David R. O'Hallaron, Person Education Limited, 2016
  - https://discovery.upc.edu/permalink/34CSUC_UPC/11q3oqt/alma991004062589706711
    - Several Chapters

- Computer Organization and Design (5th Edition)
  - D. Patterson, J. Hennessy, and P. Alexander
  - http://cataleg.upc.edu/record=b1431482~S1*cat
    - Several Chapters