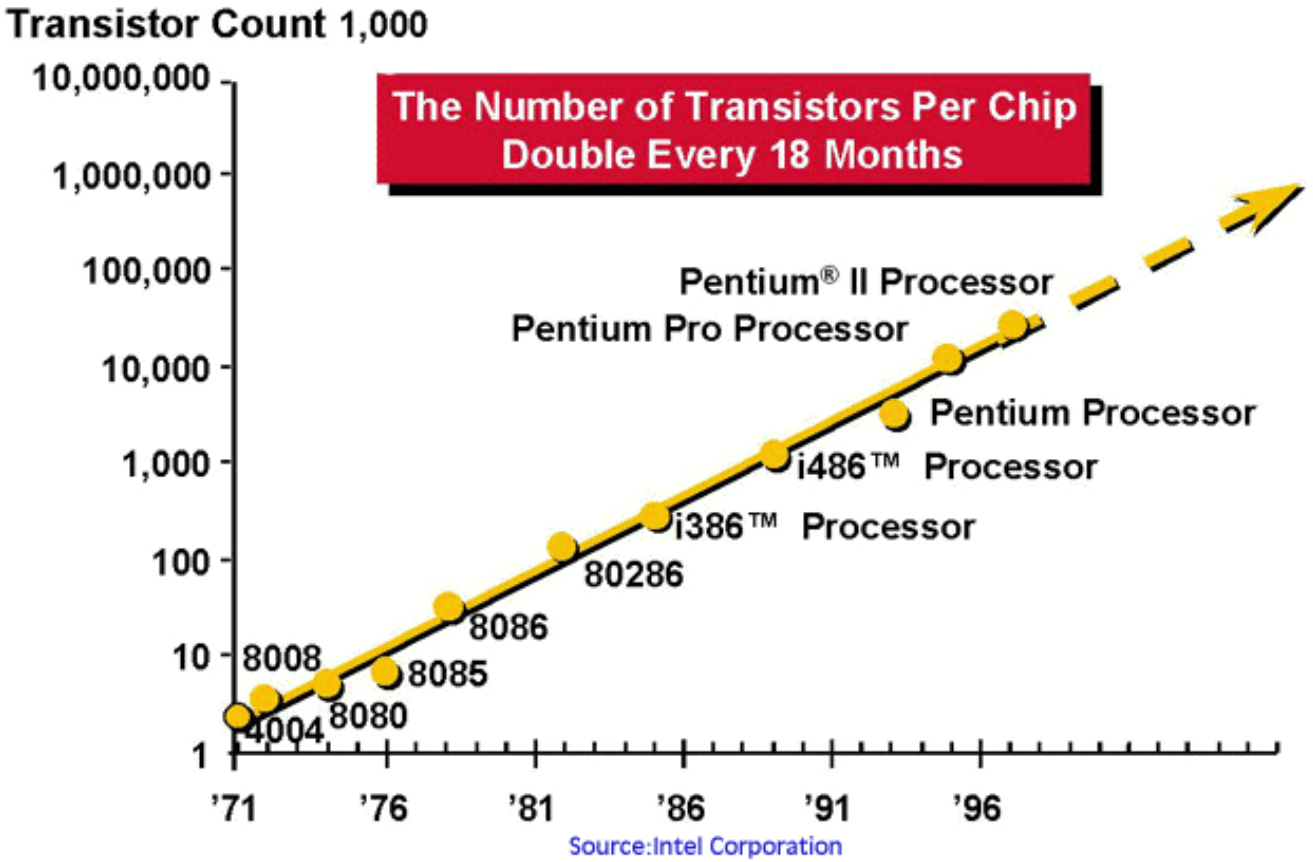
Architetture dei sistemi di elaborazione

Prestazioni dell'Hardware

Fino ai primi anni 2000 l'evoluzione dei sistemi di calcolo è stata «governata» dalla **Legge di Moore** (Gordon E. Moore, 1965, Electronics Magazine).

- Progresso tecnologico → complessità dei processori sempre maggiore, grazie alla crescente densità di transistor all'interno dei chip
- Per effetto dell'aumento della densità dei transistori, a partire dagli anni 70 le performance dei processori crescono costantemente:
  - fino al 2000**: raddoppio della densità ogni 18 mesi** -> aumento della **capacità di elaborazione del chip** -> aumento della **velocità di calcolo**

Legge di Moore



Limiti fisici alle prestazioni di singoli chip di elaborazione

A partire dai primi anni 2000 ci si è trovati **sempre più prossimi ai limiti** fisici alla densità dei transistor sui chip.

A causa di **effetti parassiti indotti dai transistori sui chip** (effetto joule) le previsioni della legge di Moore sono state sempre di più disattese.

Non è stato più possibile aumentare la frequenza di clock → necessità di aumentare la capacità di calcolo a parità di frequenza.

Miglioramento delle prestazioni

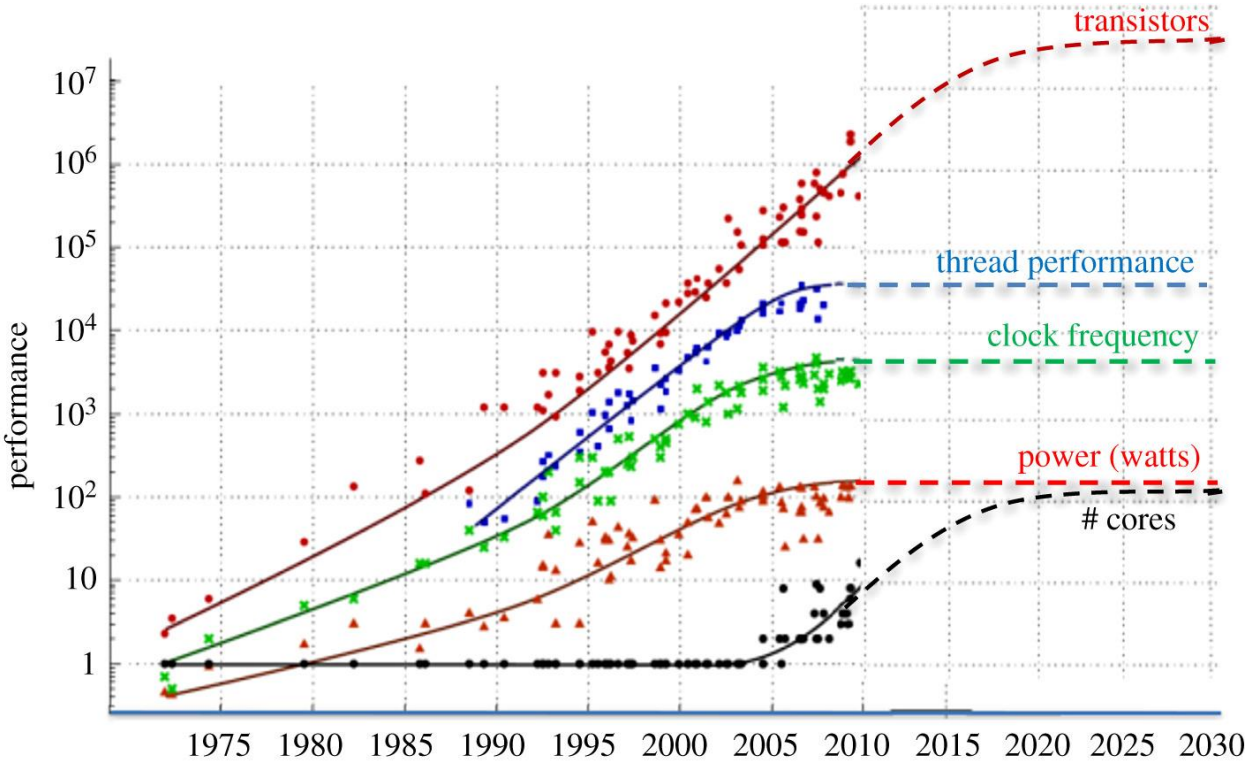
L'esigenza di prestazioni computazionali crescenti con frequenze di clock invariate ha trovato soluzione nell'introduzione di varie forme di parallelismo al livello hw.

Se l'hw è in grado di svolgere più operazioni per ciclo, la velocità di elaborazione dell'intero sistema aumenta.

Parallelismo come forma di accelerazione dell'hardware:

- più unità di elaborazione su singolo chip (multicore hardware, gpu, fpga)
- più processori su più chip (multiprocessori, cluster, ecc.)

Il futuro


transistors
thread performance
clock frequency
power (watts)
cores
performance
10^7
10^6
10^5
10^4
10^3
10^2
10
1
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030

Evoluzione delle Archietture

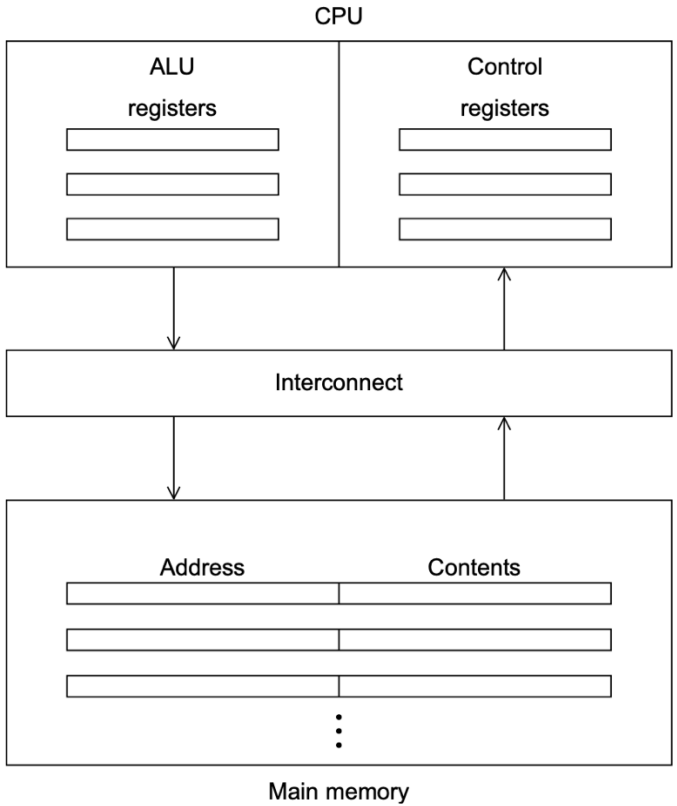
Il modello di Von Neumann

Il modello di **Von Neumann** descrive lo schema funzionale di un tradizionale sistema sequenziale.

L'unica CPU è collegata alla memoria centrale (che contiene dati e istruzioni) attraverso un mezzo di interconnessione (es: bus).

La separazione tra memoria e CPU costituisce una limitazione nella velocità di accesso a dati e istruzioni → **«Von Neumann bottleneck»** solita storia sulla memoria

Questa limitazione influisce sulla velocità di elaborazione del sistema.



Limiti del modello di Von Neumann

Von Neumann Bottleneck:

Poiché CPU e memoria sono collegate tramite il bus, la velocità di fetching di istruzioni e dati dipende dalla velocità di trasmissione del Bus → anche se la potenza della CPU è considerevole, la **velocità di esecuzione** è **limitata** dalla banda di trasmissione del **bus**. ok

Estensione del modello di Von Neumann

Per mitigare gli effetti del bottle-neck, il modello di Von Neumann è stato esteso con l'introduzione di:

- **Memorie Cache**
- **Parallelismo di basso livello**:
  - **Instruction Level Parallelism (ILP)**
  - **Hardware multithreading**

Memoria Cache

E’ una memoria **associativa**:

- **ad accesso veloce**: risiede sul chip del processore e si colloca a un livello intermedio, tra i registri e la memoria centrale
- **di capacità limitata**: la cache non può contenere tutte le istruzioni e i dati necessari al programma in esecuzione

Viene gestita con criteri basati sul principio di **località:**

- Località **temporale**: se un dato è stato usato recentemente, è probabile che venga riusato nel prossimo futuro.

- Località **spaziale**: se un dato è stato acceduto, probabilmente serviranno anche i dati vicini in memoria.

👉 Quindi la cache memorizza blocchi di dati/istruzioni presi dalla RAM che la CPU ha appena usato o che è probabile userà a breve.

Cache Hit & Miss

Quando la CPU necessita di un'informazione in memoria, sono possibili 2 casi:

1. **cache hit:** l'informazione richiesta è presente in cache → accesso **veloce**
2. **cache miss:** l'informazione richiesta non è presente in cache → necessità di load dalla memoria centrale --> accesso **lento**

La gestione della cache ha come obiettivo il **contenimento del numero dei cache miss**:

se è tale da mantenere **hit-rate** (% hit su totale degli accessi) sufficientemente elevato, gli effetti del Von Neumann bottleneck possono essere limitati.

Parallelismo Low Level: ILP

E' una tecnica che permette a un processore di sovrapporre l'esecuzione di più istruzioni dividendole in fasi sequenziali.

Si basa sul fatto che l'esecuzione di ogni istruzione viene attuata attraverso una **sequenza di fasi**;

Ad esempio: somma di due float **C=A+B**

1. ***fetch*** operandi A e B
2. ***confronto esponenti*** ed eventuale **shift**
3. ***somma***
4. ***normalizzazione*** del risultato C
5. ***memorizzazione*** di C

Ognuna delle fasi può essere affidata a un'**unità funzionale** HW indipendente che opera in parallelo alle altre:

- **pipelining**: tutte le unità funzionali sono collegate tra loro in una pipeline; fasi diverse di istruzioni diverse possono essere eseguite in parallelo;
- **multiple issue:** più istanze di ogni unità funzionale

Es: ILP pipeline

Table 2.3 Pipelined Addition. Numbers in the Table Are Subscripts of Operands/Results

Time	Fetch	Compare	Shift	Add	Normalize	Round	Store
0	0						
1	1	0					
2	2	1	0				
3	3	2	1	0			
4	4	3	2	1	0		
5	5	4	3	2	1	0	
6	6	5	4	3	2	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
999	999	998	997	996	995	994	993
1000		999	998	997	996	995	994
1001			999	998	997	996	995
1002				999	998	997	996
1003					999	998	997
1004						999	998
1005							999

unità funzionali collegate in pipeline

A regime (es. T=6), ogni unità funzionale processa un'istruzione diversa.

Limitazioni ILP

L'efficacia di tecniche ILP può ridursi nel caso di **dipendenze tra istruzioni successive**: se in un programma è presente una serie di istruzioni tra loro dipendenti , il parallelismo a livello di istruzione è limitato.

Esempio: si consideri la sequenza di istruzioni

```
a = b + c
d = e + f
g = a + d
```

La terza istruzione dipende dai risultati delle due precedenti→ per poter essere processata è necessario il completamento delle precedenti**.**

Infatti: la terza operazione aritmetica (a+d) deve attendere che la pipeline completi l'ultima fase (memorizzazione) della prima e della seconda istruzione.

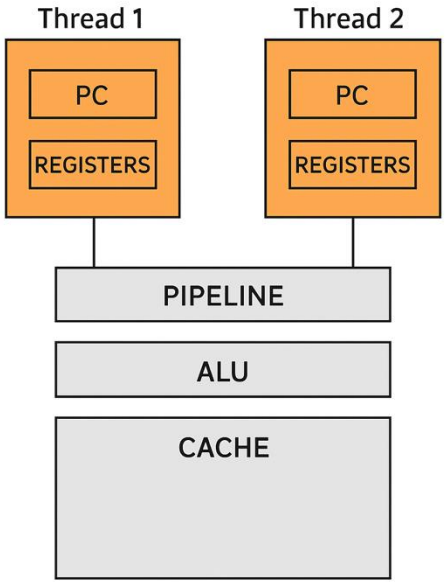
Parallelismo Low Level: Hardware multi-threading

In alternativa/aggiunta a ILP, i processori moderni offrono **parallelismo a livello di thread** (thread level parallelism) mediante **HW multithreading:**

è una tecnica che permette a più thread (ad esempio, 2 thread) di **condividere** la stessa CPU (core).

Ciò è reso possibile:

- dalla **duplicazione** dei registri che mantengono lo stato di ogni thread (PC, altri registri..)
- da un **meccanismo HW** che implementa il context switch tra un thread ed un altro in modo molto efficiente: per ogni cambio di contesto si commuta la CPU verso il set di registri del thread schedulato (nessun salvataggio né ripristino verso/da memoria!).



Parallelismo Low Level: Hardware multi-threading

La gestione dei thread è governata da uno scheduler hardware, che decide ad ogni ciclo a quale thread associare l'unità di elaborazione.

L'obiettivo è contrastare gli effetti del Von Neumann bottleneck → limitare le attese dovute a **cache miss.**

Due approcci:

- **Multithreading a grana fine (*fine-grained*)**: viene eseguito un context switch dopo ogni istruzione.
  - **Vantaggio**: Attese dovute a cache miss vengono «nascoste» allocando la CPU ad altri thread.
  - **Svantaggio**: velocità di esecuzione dei thread ridotta.

- **Multithreading a grana grossa (*coarse-grained*)**: il context switch avviene quando il thread corrente è in una situazione di attesa (ad esempio: in caso di «**cache miss**», deve attendere il caricamento del'informazione dalla memoria centrale).
  - **Vantaggio**: meno context switch, quindi velocità media di esecuzione dei thread più alta.
  - **Svantaggio**: il context switch è più costoso (necessità di vuotare la pipeline ILP). Throughput più basso.

Architetture ad elevate prestazioni per High performance Computing

ILP e **Hardware multithreading** hanno permesso un miglioramento delle prestazioni dei processori, tuttavia tali meccanismi sono trasparenti per gli sviluppatori dei programmi. → modello **Von Neumann Esteso**

Nei sistemi HPC, invece, il parallelismo disponibile per l'esecuzione dei programmi è visibile al programmatore, che deve progettare il software in modo da sfruttare al meglio tutte le risorse computazionali a disposizione. → architetture «**non Von Neumann»**

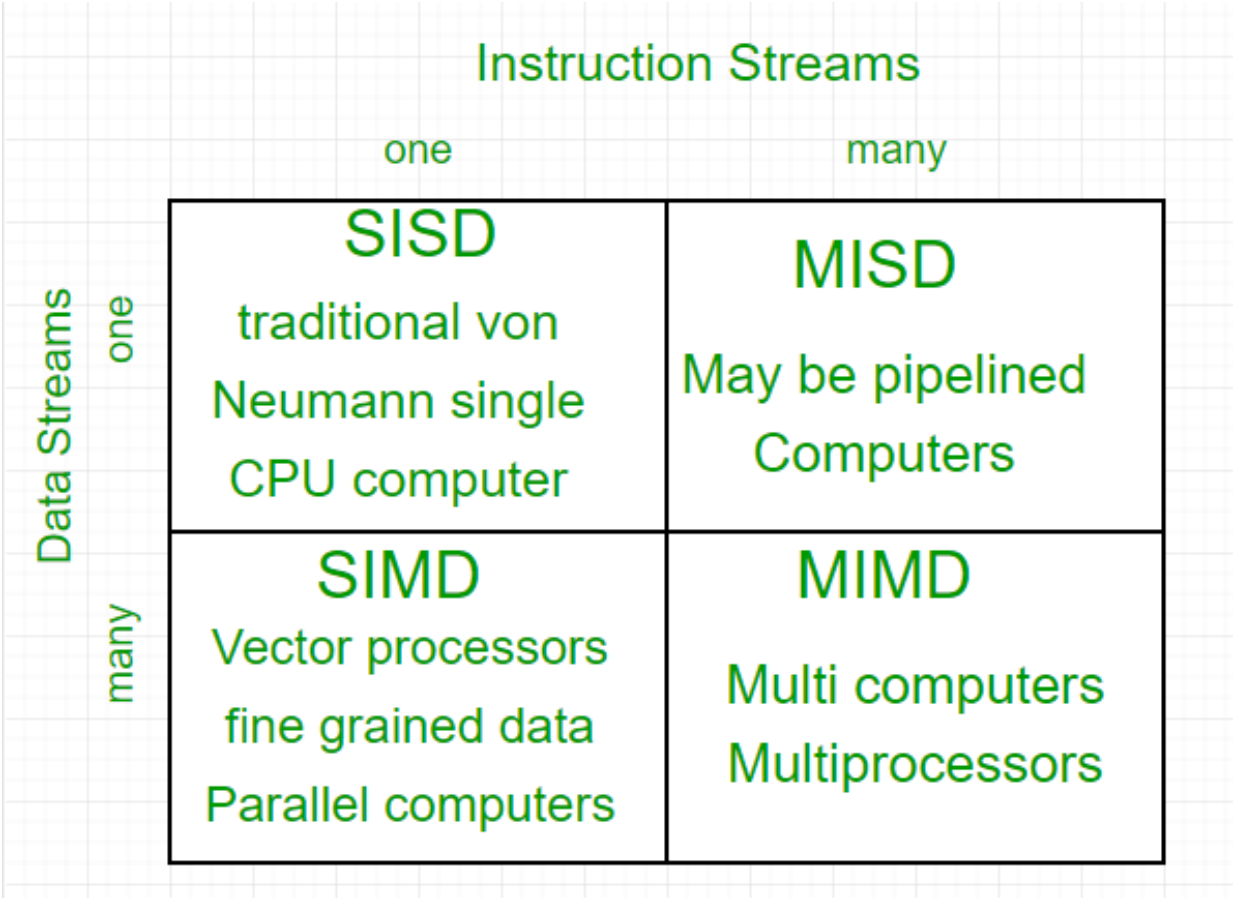
Classificazione delle Architetture

Le architetture dei sistemi di elaborazione possono avere caratteristiche profondamente diverse, in base alle risorse che li compongono e a come vengono gestite.

Una delle classificazioni più note dei sistemi di calcolo è quella di Michael J. **Flynn** (1966), basata su due criteri:

- Flusso di istruzioni (**Instruction Stream**): quanti flussi di istruzioni il sistema è in grado di eseguire contemporaneamente
- Flusso di dati (**Data Stream**): quanti flussi diversi di dati possono essere elaborati contemporaneamente.

Flynn Taxonomy (1966)

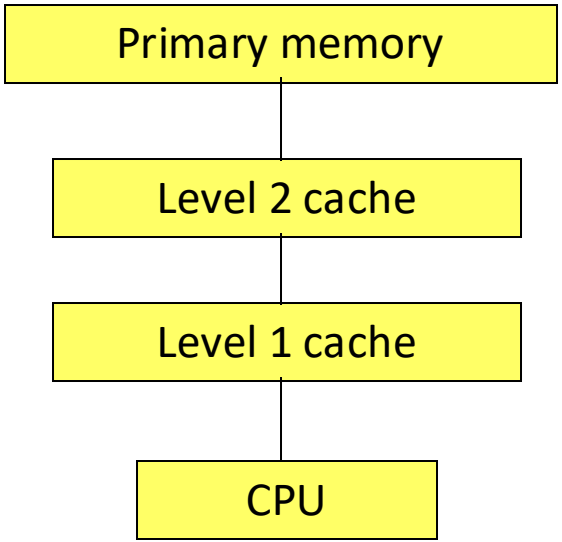


Categorie di Flynn

- **SISD**: single instruction, single data. Questa categoria individua il modello classico di Von Neumann, in cui non c'è parallelismo né di istruzioni, né di dati: un solo processore esegue un unico flusso di istruzioni su un singolo flusso di dati.
- **SIMD**: architetture composte da molte unità di elaborazione che eseguono contemporaneamente la stessa istruzione su dati diversi. Ad esempio: GPU, processori vettoriali.
- **MISD**: più istruzioni in parallelo sullo stesso flusso di dati. Molto raro nella pratica: modello utilizzato solo in sistemi di tolleranza ai guasti (fault tolerance→elaborazioni ridondanti).
- **MIMD**: più flussi diversi di istruzioni vengono eseguiti in parallelo su flussi di dati diversi. In questa categoria rientrano i **multiprocessori**, i **multicomputers**, i **cluster** e molti sistemi **HPC**

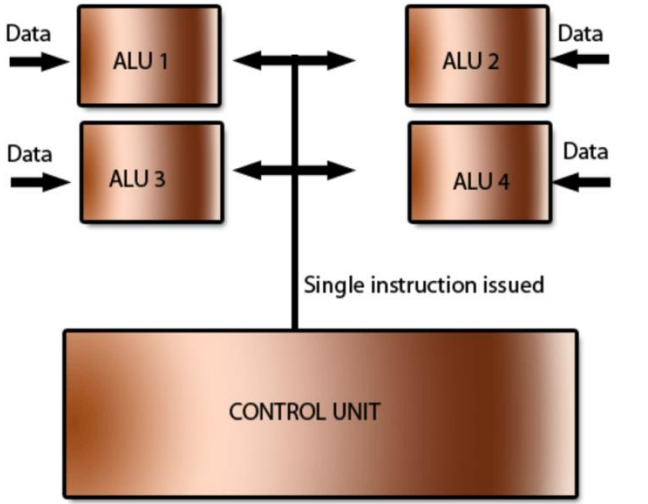
SISD

- **Classico sistema monoprocessore**



SIMD

Array processors: architetture composte da più unità di elaborazione gestite da un'unica unità di controllo, in modo che eseguano in modo sincrono la stessa istruzione su data stream separati:



An array processor - a Single Instruction Multiple Data computer

Vengono sfruttati per il calcolo ad alte prestazioni e sono particolarmente adatti per applicazioni di calcolo scientifico (dati in forma vettoriale/matriciale)

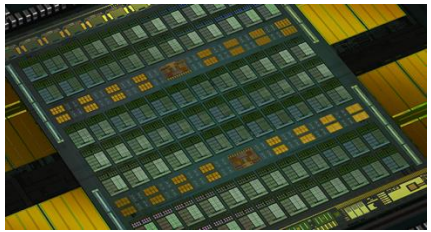
SIMD

SINCRONICITA' nel FUNZIONAMENTO: Ad ogni istante le unità di calcolo presenti a livello HW eseguono la stessa istruzione su dati diversi.

- Più processori ed 1 sola control unit.

Esempio: GPU

Nelle prime GPU i processori grafici sono costituiti da molti core vincolati all'esecuzione della stessa istruzione; in quelle più recenti si ha un'architettura ibrida, in cui gruppi di core diversi possono eseguire flussi di istruzioni diverse.



Nvidia Tesla V100 Volta architecture (5,120 CUDA cores)

Sistemi MIMD

Asincronicità delle attività nei diversi nodi: più processori, ognuno con la sua propria control unit.

Ogni CPU esegue una sequenza di istruzioni diversa dagli altri.

Necessità di interazione tra nodi

2 categorie di sistemi:

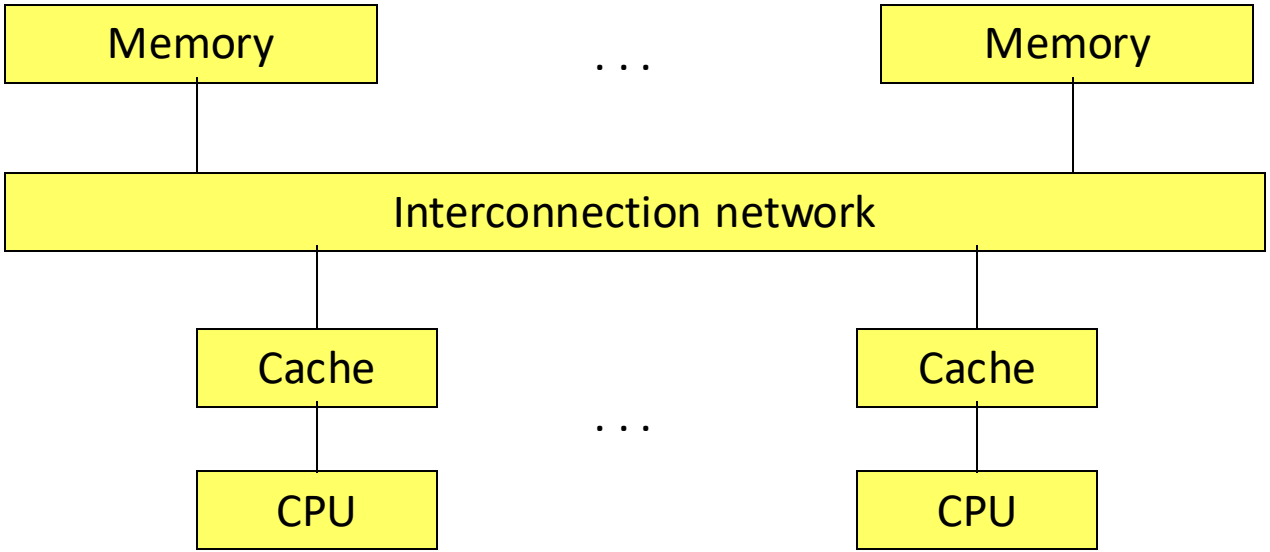
shared memory (UMA, NUMA multiprocessor/multicore)

distributed memory (cluster, HPC supercomputers)

Necessario **supporto a livello architetturale** per consentire l'**interazione**:

- reti di interconnessione; possibilmente: bassa latenza e banda elevata
- Memoria condivisa & cache -> problema cache coherence

Shared- Memory Multiprocessors

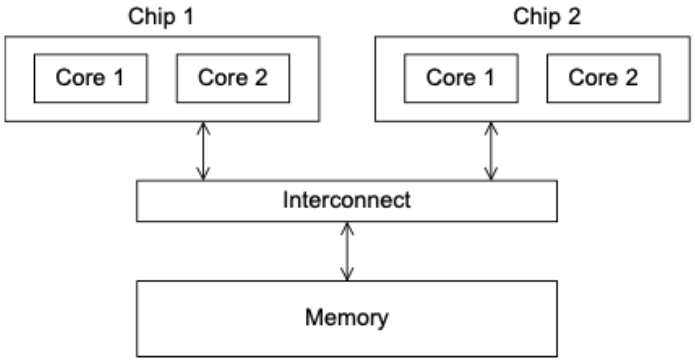


Shared memory systems

Fanno parte di questa categoria i sistemi **multicore** e **multiprocessore**.

Esempio: Sistema Multicore

Normalmente: più processori, ognuno con più core:



Esempio:
Intel Xeon Phi 7290
72 cores
x4 hyperthreading

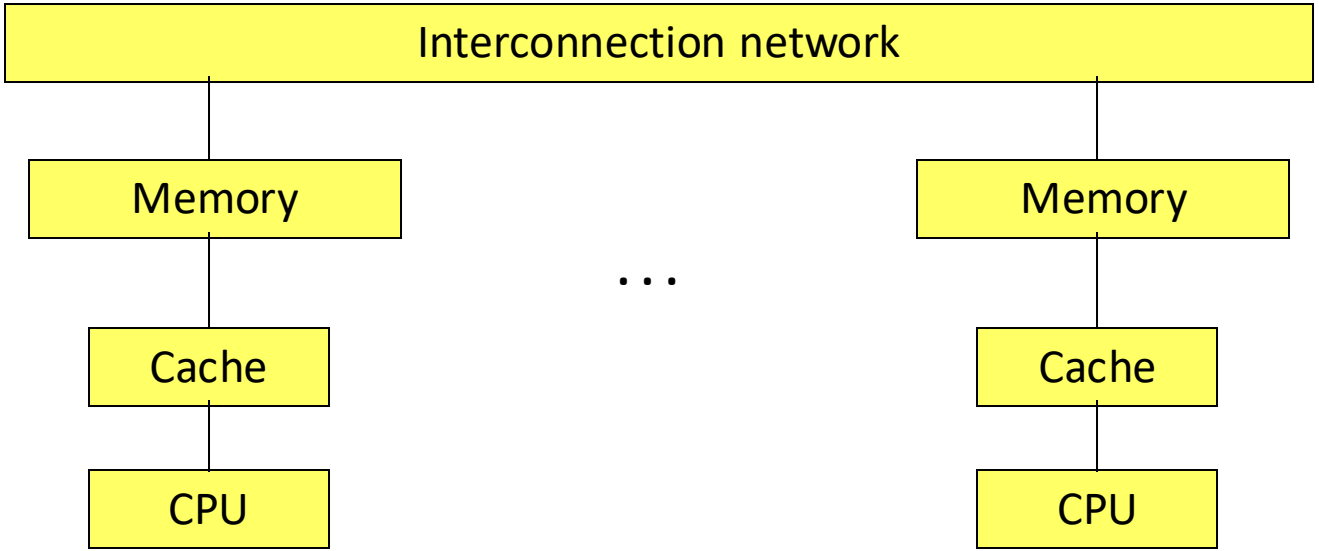
Sistemi Multiprocessore

2 modelli architetturali:

- **UMA** (Uniform Memory Access): sistemi a multiprocessore con un **numero ridotto di processori** (da 2 a 30 circa):
  - la rete di interconnessione è realizzata da un *memory bus* o da *crossbar switch*.
  - ***UMA*** → il tempo di accesso uniforme da ogni processore ad ogni locazione di memoria.
  - Si chiamano anche symmetric multiprocessors (SMP).

- **NUMA** (Non Uniform Memory Access): sistemi con un **numero elevato di processori** (decine o centinaia):
  - la memoria è organizzata ***gerarchicamente*** (per evitare la congestione del bus).
  - La rete di interconnessione è un insieme di *switches* e *memorie* strutturato ad albero. Ogni processore ha memorie che sono più vicine (accesso più veloce) ed altre più lontane (accesso più lento).
  - ***NUMA*** : il tempo di accesso dipende dalla distanza tra processore e memoria.

Distributed-memory: Multicomputers e Network systems



Distributed-memory: classificazione

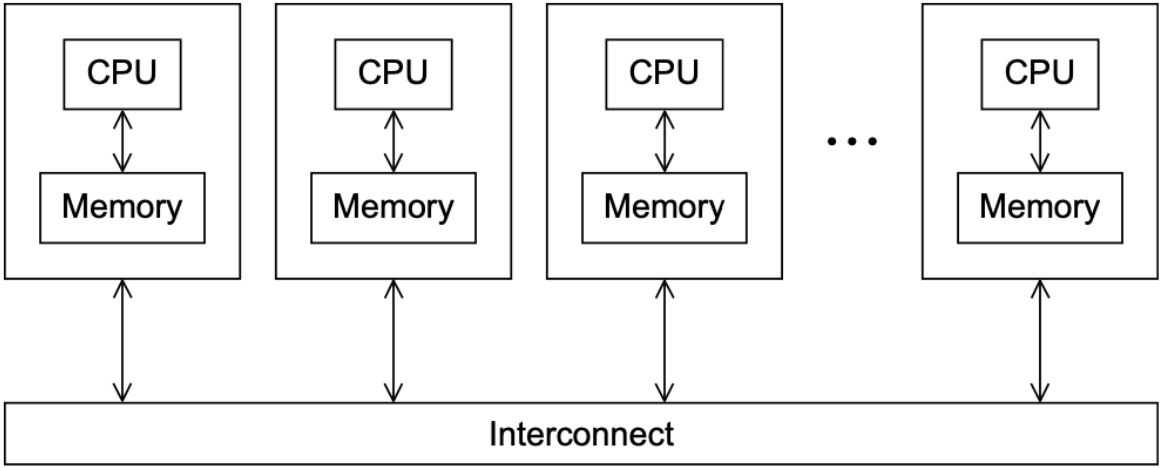
Due modelli:

- ***Multicomputer*** : I processori e la rete sono fisicamente vicini (nella stessa struttura): tightly coupled machine.
  - La rete di interconnessione offre un cammino di comunicazione tra i processori ad alta velocità e larghezza di banda (es. Cluster of Computers COW, HPC systems).

- ***Network systems*****:** i nodi sono collegati da una rete locale (es.Ethernet) o da una rete geografica (Internet): loosely coupled systems.

I nodi di un distributed memory system possono essere o singoli processori o shared memory multiprocessor (→sistemi ibridi o eterogenei).

HPC Distributed Memory systems



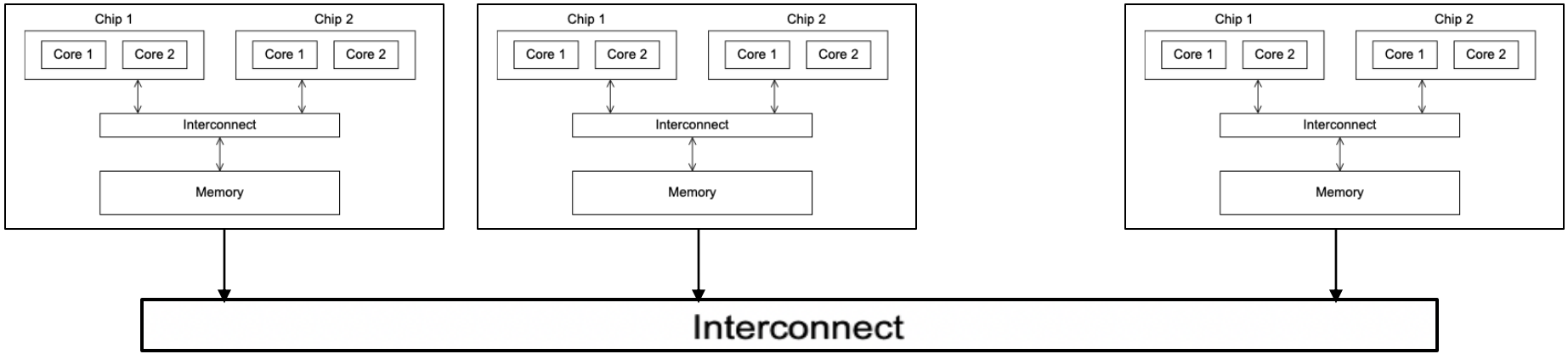
- E' fondamentale che l'interconnessione tra nodi venga realizzata in modo tale da garantire:
  - latenza bassa
  - elevata bandwith

👉 No Bus, ma **reti ad alte prestazioni** (topologie mesh, toroidali, ipercubi, ecc).

HPC systems: modelli ibridi

La maggior parte dei sistemi HPC oggi combina il modello a memoria distribuita con il modello shared memory. Ad esempio:

- **cluster** di nodi a memoria distribuita collegati da reti ad alte prestazioni
- ogni nodo è un **multiprocessore**: insieme di processori **multicore**.

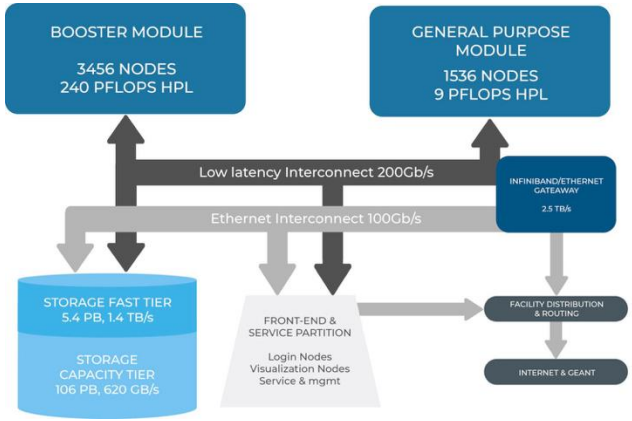


Struttura tipica di un cluster HPC

Esempio: **Leonardo, Cineca**

- 4992 computing nodes subdivided in:
  - 3456 booster nodes Intel Xeon 8358 each with :
    - 32 cores, 2.6 GHz → Cores: 110592 (32 cores/node),
    - 4XNvidia custom Ampere GPU 64GB HBM2,
    - RAM: (8x64) GB DDR4 3200 MHz
  - 1536 data-centric nodes Intel Saphire Rapids, 4.8 GHz, each with;
    - 2x56 cores → Cores: 172032 (112 cores/node)
    - RAM: (48x32) GB DDR5 4800 MHz
- 16 visualization nodes 2 x Icelake ICP06 32cores 2.4GHz, 3 NVIDIA Tesla V100, RAM: (16 x 32) GB DDR5 4800 MHz
- 137,6 PB (raw) Large capacity storage, 620 GB/s
- High Performance Storage 5.7 PB, 1.4 TB/s Based on 31 x DDN Exascaler ES400NVX2
- Login and Service nodes: 16 Login nodes are available. 16 service nodes for I/O and cluster management.

E' il 10^ sistema HPC più potente al mondo (Giugno 2025)



→ 282.624 cores