

# GPU Computing

Laurea Magistrale in Informatica - AA 2024/25

Docente **G. Grossi**

Lezione 1 – Introduzione al corso

# Innovazione many-core

HPC

# Why HPC?

EXA = 1,152,921,504,606,846,976 =  $2^{60}$  =  $1024^{6}$  ≈  $10^{18}$

PETA = 1,125,899,906,842,624 =  $2^{50}$  =  $1024^{5}$  ≈  $10^{15}$

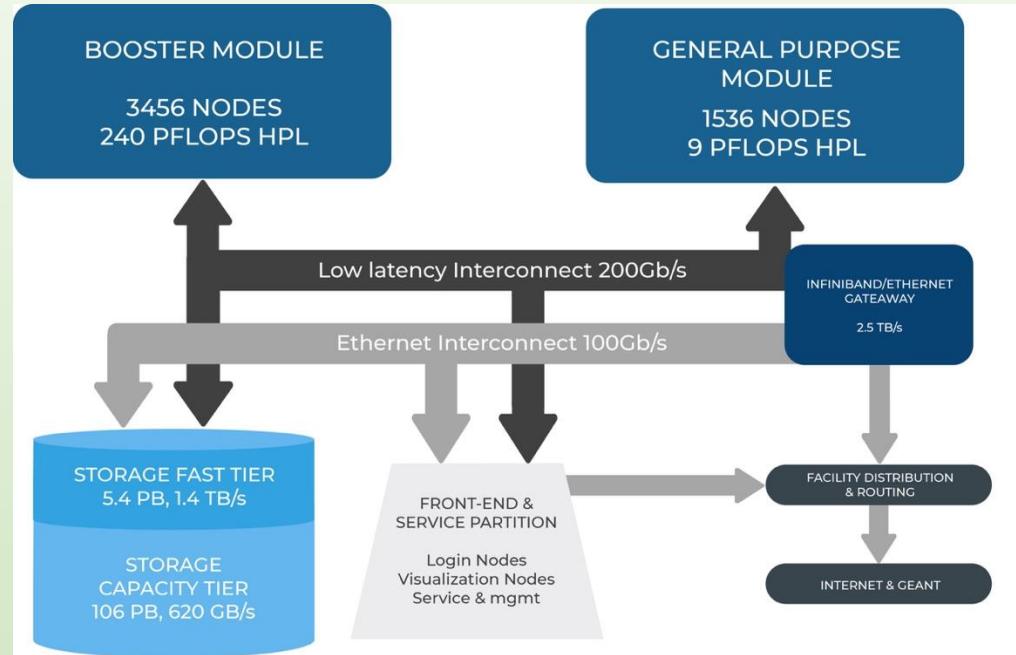
TERA = 1,099,511,627,776 =  $2^{40}$  =  $1024^{4}$  ≈  $10^{12}$

GIGA = 1,073,741,824 =  $2^{30}$  =  $1024^{3}$  ≈  $10^{9}$

MEGA = 1,048,576 =  $2^{20}$  =  $1024^{2}$  ≈  $10^{6}$

KILO = 1,024 =  $2^{10}$  =  $1024^{1}$  ≈  $10^{3}$

# Leonardo (EuroHPC JU)



- ✓ Features a custom **BullSequana X2135** “Da Vinci” blade
  - 1 x CPU Intel **Xeon 8358** 32 cores, 2,6 GHz booster
  - **512 (8 x 64) GB** RAM DDR4 3200 MHz
  - 4 X **Nvidia** custom **Ampere** GPU 64GB HBM2
  - 2 x **NVidia HDR 2x100 Gb/s** cards
- ✓ Performance per node: **89.4 TFLOPs peak**



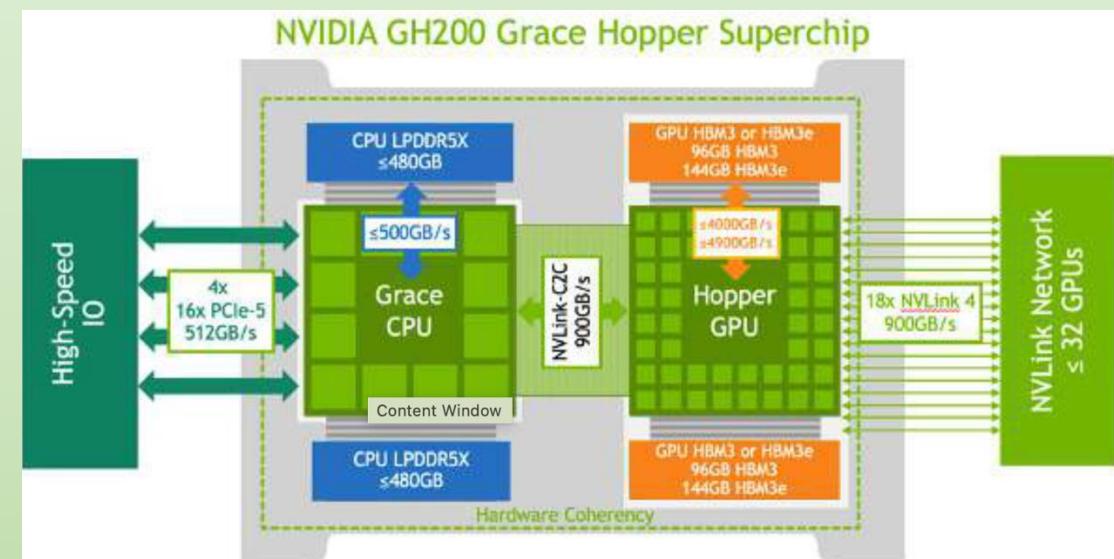
# NVIDIA GH200 Grace Hopper Superchip

## JUPITER The Arrival of Exascale in Europe

Forschungszentrum Jülich will be home to Europe's first exascale computer with a booster module comprising close to **24,000 NVIDIA GH200 Superchips**

Being the **world's most powerful AI system**, JUPITER can deliver over **90 exaflops** of performance for **AI training** and **1 exaflop** for high performance computing (**HPC**) applications while consuming only 18.2 megawatts of power.

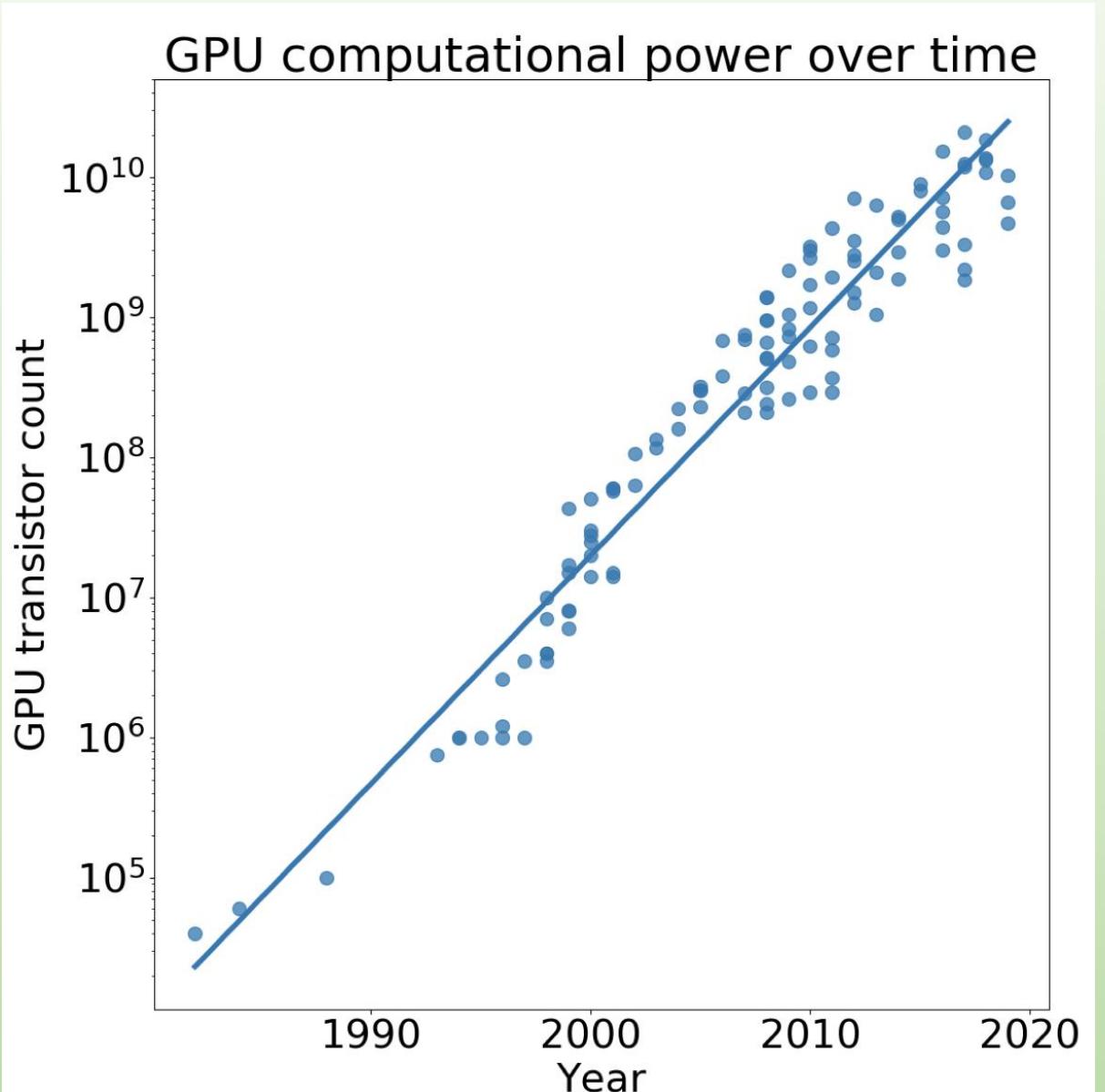
	Grace Hopper (GH200)
CPU Cores	72
CPU Architecture	Arm Neoverse V2
CPU Memory Capacity	<=480GB LPDDR5X (ECC)
CPU Memory Bandwidth	<=512GB/sec
GPU SMs	132
GPU Tensor Cores	528
GPU Architecture	Hopper
GPU Memory Capacity	<=96GB
GPU Memory Bandwidth	<=4TB/sec
GPU-to-CPU Interface	900GB/sec NVLink 4
TDP	450W - 1000W



# Moore's Law

*"The processor speeds, or overall processing power for computers will double about every 18 month's "*

A logarithmic plot of GPU transistor count (proportional to computational power) versus time shows Moore's Law holds true for GPUs



# Dalla tradizione (la CPU)...

## Central Processing Unit

- Fonte primaria per il **calcolo**
- Composta da due parti principali: l'unità di **controllo** e l'unità di **elaborazione**
- Potente per applicazioni **general purpose**
- Tecnologia **consolidata**
- Di solito con  $\sim k \cdot 10$  ( $k \leq 3$ ) **potenti core**
- Limiti per calcoli paralleli su **larga scala**



# ... all'efficienza (la GPU)

## Graphics Processing Unit

- ✓ Nasce per la grafica!
- ✓ Fino a **10k** core (vs  $\sim 10$  della CPU)
- ✓ Utile solo per **problemi** altamente **parallelizzabili** (velocità 10x, 100x ...)
- ✓ Introduce il paradigma **GP-GPU (General Purpose – GPU)**
- ✓ Tecnologia in continua evoluzione



# Motivazioni x calcolo parallelo

Parallel patterns

# GPUs - Motivazioni

**Grandi porzioni di calcoli sono lo “stesso calcolo”!!**

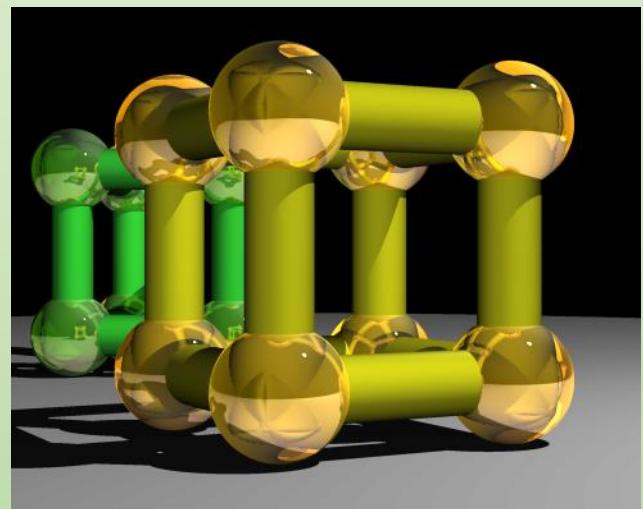
Raytracing:

```
for all pixels (i,j) in image:  
    From camera eye point,  
        calculate ray point and direction in 3d space  
    if ray intersects object:  
        calculate lighting at closest object point  
        store color of (i,j)  
Assemble into image file
```

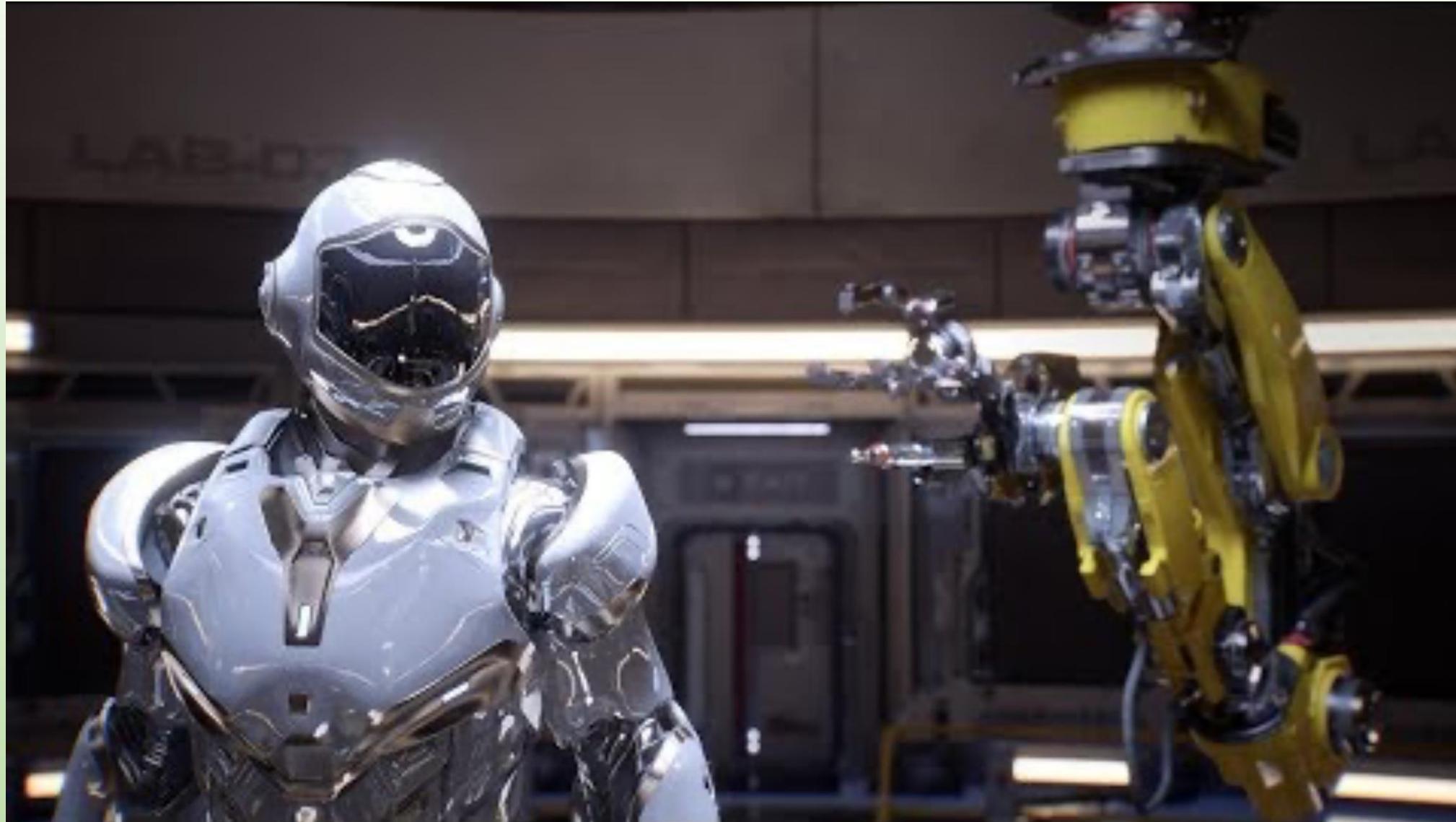
Ogni pixel potrebbe essere calcolato simultaneamente, con sufficiente parallelismo!



Un raggio colpisce una superficie, genera fino a tre nuovi tipi di raggi: **riflessione**, **rifrazione** ed **ombra**



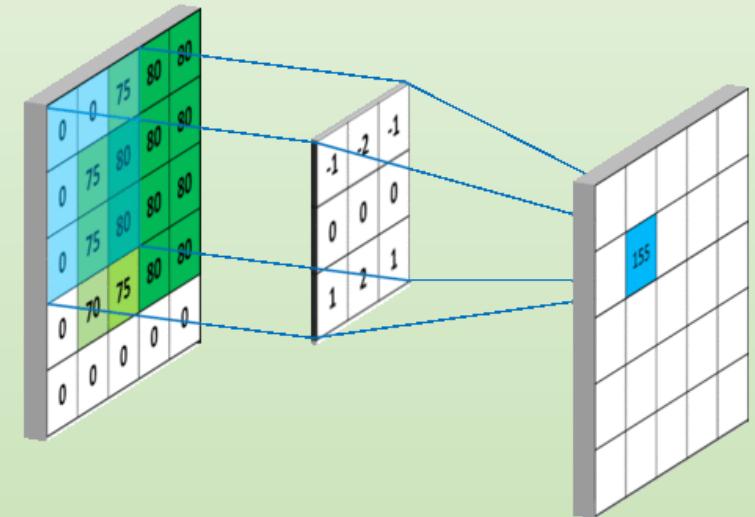
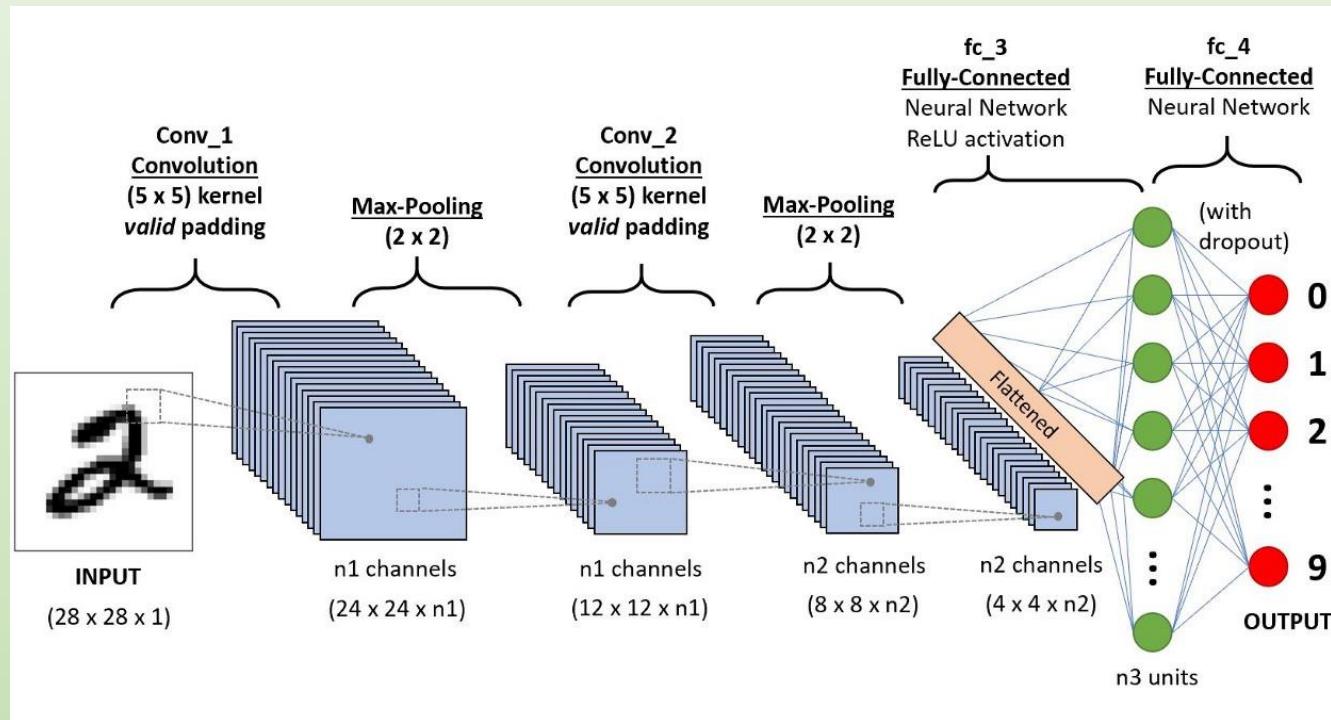
# Real-time ray tracing



<https://www.youtube.com/watch?v=u8tDgvvGWSE>

# Convolutional Neural Networks (CNN)

**convoluzione:** prodotto ripetuto molte volte!



# Transformer

## Self-Attention

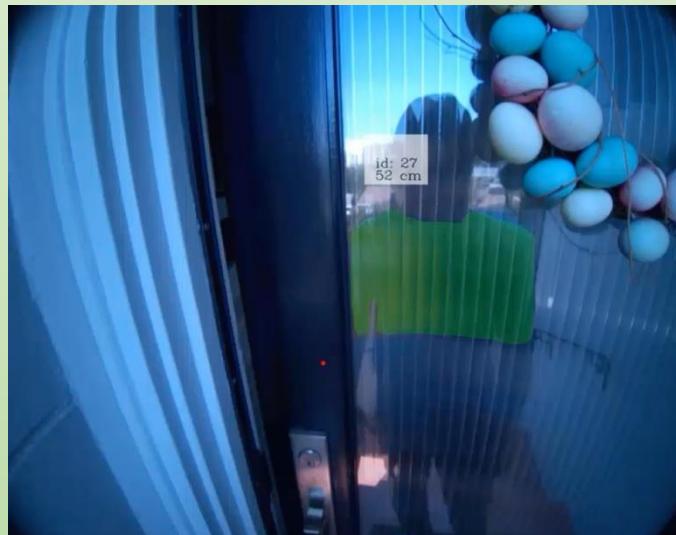
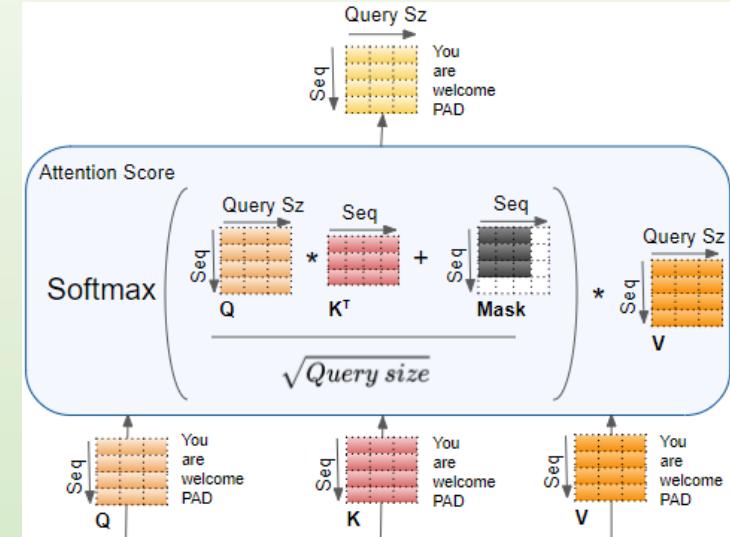
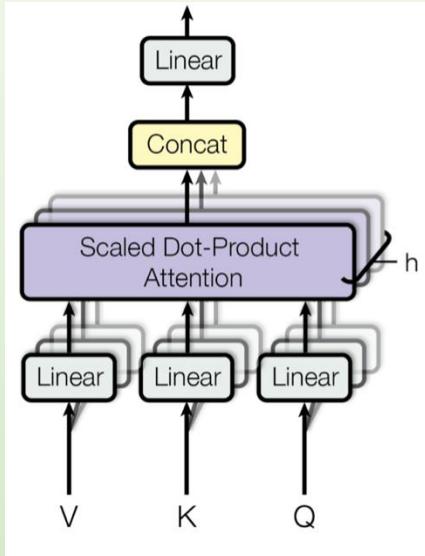
**What is self-attention?** Self-attention calculates a weighted average of feature representations with the weight proportional to a similarity score between pairs of representations. Formally, an input sequence of  $n$  tokens of dimensions  $d$ ,  $X \in \mathbf{R}^{n \times d}$ , is projected using three matrices  $W_Q \in \mathbf{R}^{d \times d_q}$ ,  $W_K \in \mathbf{R}^{d \times d_k}$ , and  $W_V \in \mathbf{R}^{d \times d_v}$  to extract feature representations  $Q$ ,  $K$ , and  $V$ , referred to as query, key, and value respectively with  $d_k = d_q$ . The outputs  $Q$ ,  $K$ ,  $V$  are computed as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \quad (1)$$

So, self-attention can be written as,

$$S = D(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \quad (2)$$

where softmax denotes a *row-wise* softmax normalization function. Thus, each element in  $S$  depends on all other elements in the same row.



# GPUs - Motivazioni

Grandi porzioni di calcoli sono lo “stesso calcolo”!!

Prodotto riga e colonna:

for all entry  $C(i,j)$ :

    for all  $k = 1:\text{num\_col}(A)$

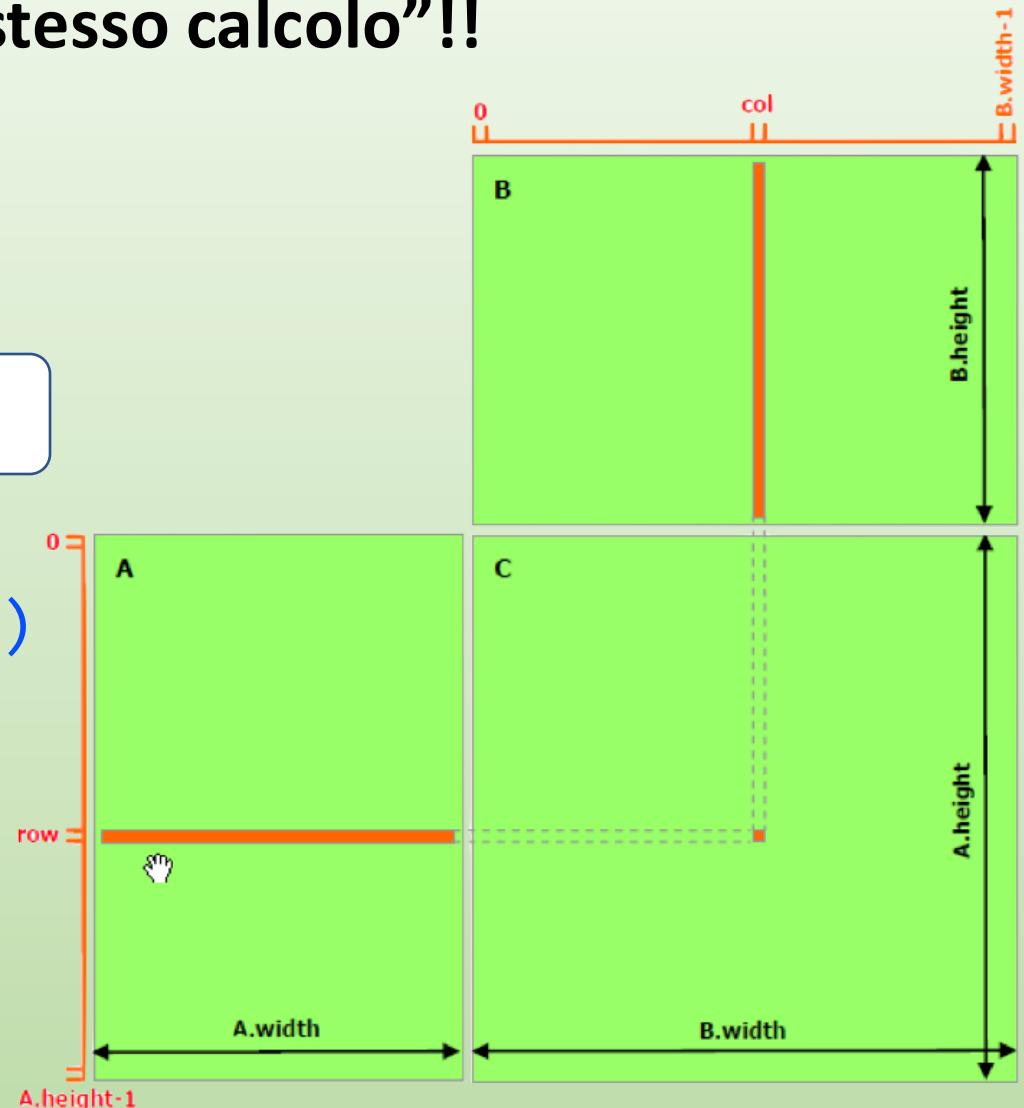
        calculate  $s = s + A(i,k)*B(k,j)$

    store  $s$  in  $C(i,j)$

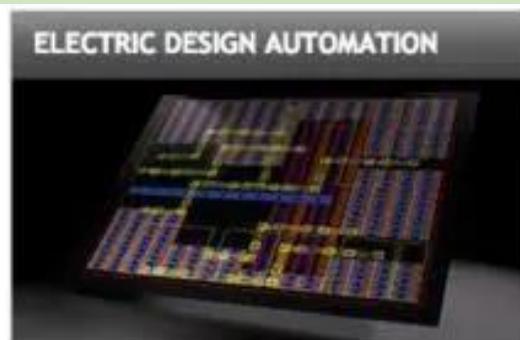
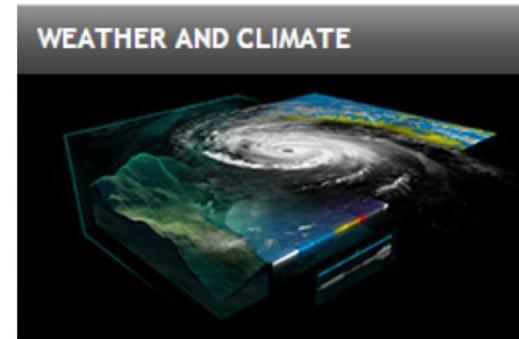
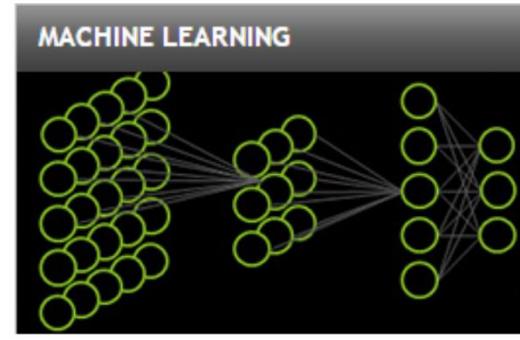
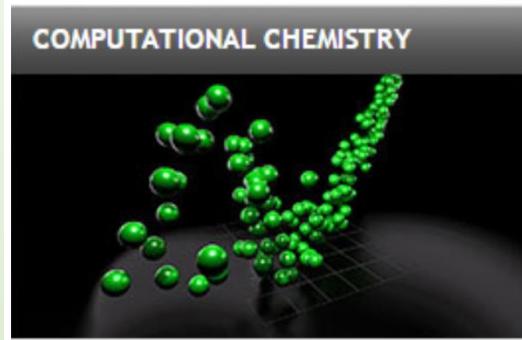
Complessità del prodotto matriciale:  $\mathcal{O}(n^3)$

$\mathcal{O}(n^2)$

$\mathcal{O}(n)$



# Domains with CUDA-Accelerated Applications



# Parallelismo

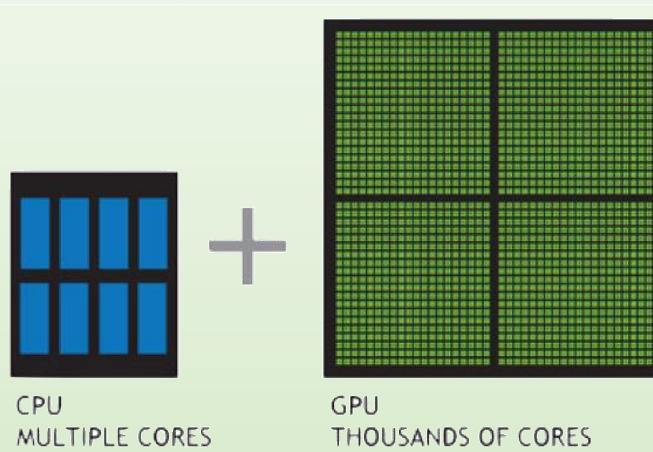
- ✓ **Definizione:** capacità di eseguire parti di un calcolo in modo concorrente
- ✓ **Obiettivo:** risolvere velocemente problemi di grandi dimensioni
  - Con più parallelismo si può:
    - risolvere problemi più grandi nello stesso tempo
    - risolvere un problema di dimensioni fisse in un tempo più breve
- ✓ **Grado di parallelismo:** quanto sono grandi le unità?
  - bit, istruzioni, blocchi, iterazioni di cicli, procedure, ...
- ✓ **Focus del corso:** parallelismo esplicito a livello di thread
  - **Thread** = unità di esecuzione costituita da una sequenza di istruzioni e gestita dal sistema operativo o da un sistema di runtime

# Il paradigma GP-GPU



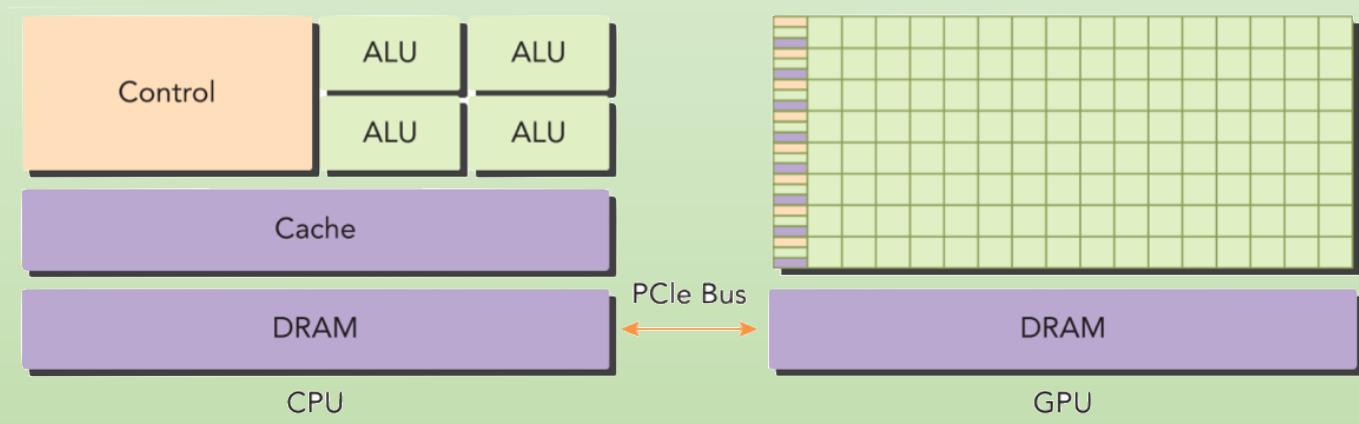
- ✓ **General-purpose GPU (GP-GPU)** si riferisce all'uso della GPU (graphics processing unit) per eseguire computazioni scientifiche, multimediali, ingegneristiche... a carattere generale
- ✓ **Modello eterogeneo**: implica l'uso di CPU e GPU insieme dando vita a un modello calcolo di co-processing
- ✓ **Separazione**: la parte sequenziale dell'applicazione esegue su CPU mentre la parte parallela (quella a maggiore intensità computazionale) viene accelerata dalla GPU
- ✓ **Trasparente**: dal punto di vista utente, l'applicazione esegue nel complesso più velocemente avvalendosi delle elevate prestazioni della GPU
- ✓ **Mapping**: una function sulla GPU implica riscrivere la function per esporla al parallelismo della GPU aggiungendo le annotazioni "C" che spostano dati ed eseguono una function (kernel) sulla GPU
- ✓ **Problema**: massimizzare la potenza di calcolo minimizzando l'energia consumata

# Architetture eterogenee

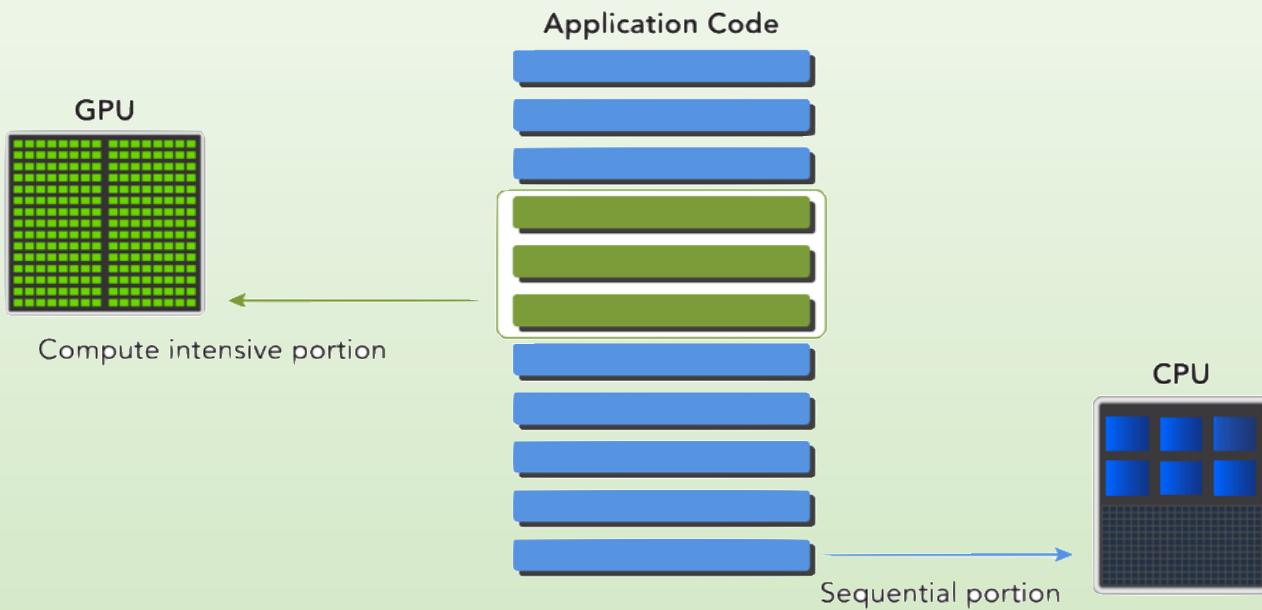


- ✓ La GPU è un **coprocessore** e non una piattaforma standalone, vengono anche chiamati **acceleratori**
- ✓ La CPU è chiamata **host** e la GPU **device**
- ✓ Opera congiuntamente con la CPU attraverso il **bus PCI-Express**
- ✓ La GPU necessita di **trasferimenti diretti di memoria** da parte della CPU
- ✓ CPU "orchestra" la **sincronizzazione**

- ✓ Differenze di esecuzione dei task
- ✓ **CPU** = pochi core ottimizzati per l'elaborazione sequenziale
- ✓ **GPU** = architettura massicciamente parallela che consiste di migliaia di core che cooperano in modo efficiente per trattare molteplici task concorrentemente



# Applicazioni ibride



- ✓ Le applicazioni eterogenee sono caratterizzate da:
  - Codice host
  - Codice device
- ✓ Il codice host esegue su CPU e il codice device esegue su GPU
- ✓ Le GPU sono usate per accelerare le esecuzioni di codice parallelo su dati parallelî

- ✓ CPU: **gestione dell'ambiente**, dell'**IO** e della gestione dei **dati per il device** stesso, caricare task intensivi sul device
- ✓ GPU: usata per accelerare l'esecuzione di questa porzione di codice basandosi sul **parallelismo dei dati**
- ✓ CPU: ottimizzata per sequenze di operazioni in cui il controllo del **flusso** è **impredicibile**
- ✓ GPU: ideali per carichi dominati da **semplice flusso** di controllo

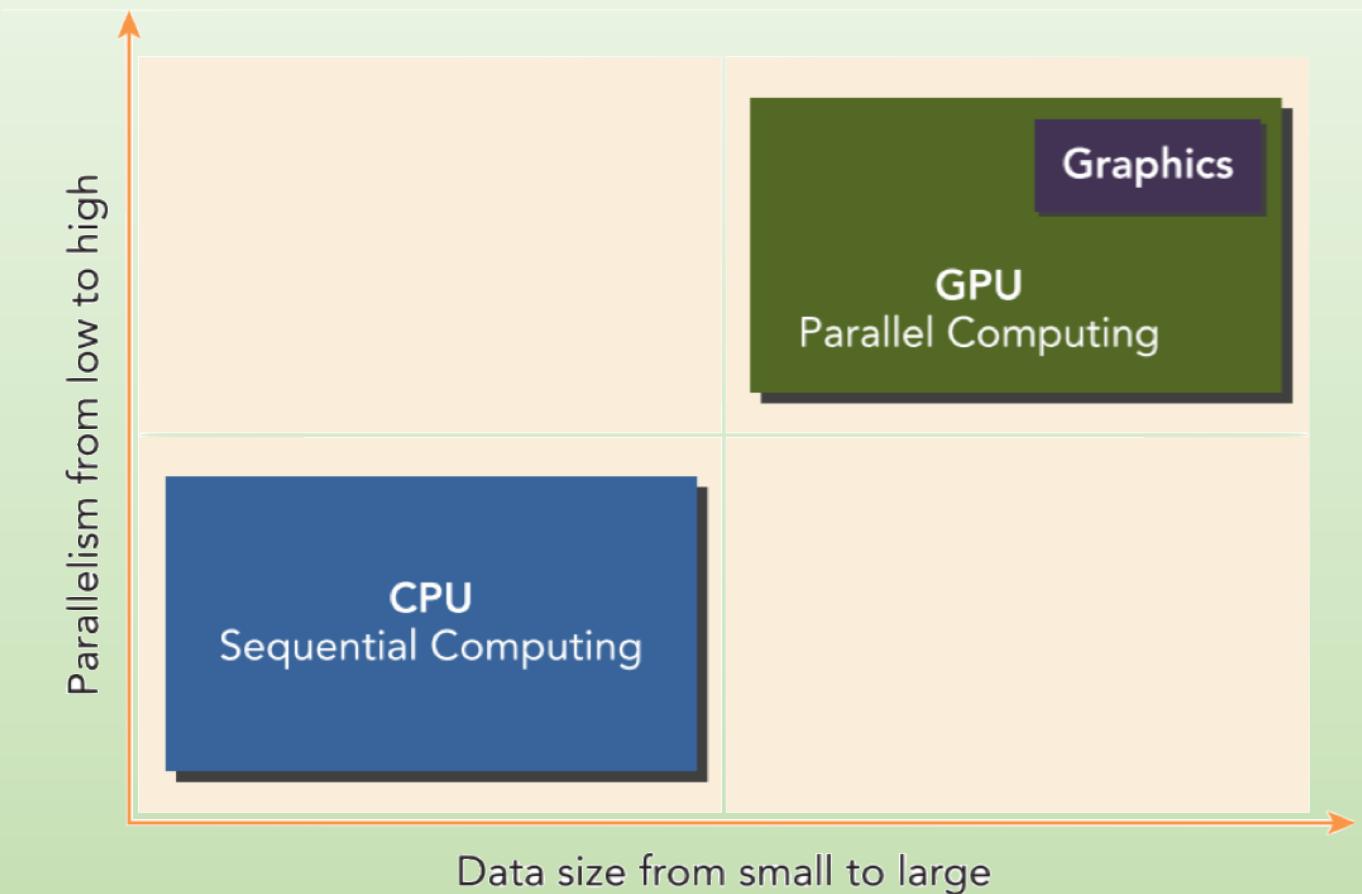
# Realizzazione del calcolo parallelo

Il **calcolo parallelo** può essere realizzato in vari modi, tra cui:

1. **Parallelismo di dati:** suddivisione dei dati in parti uguali e l'elaborazione simultanea di ogni parte su più processori
2. **Parallelismo di task:** suddivisione del lavoro in attività indipendenti e l'assegnazione simultanea di queste attività a diversi processori
3. **Parallelismo di istruzioni:** suddivisione delle istruzioni di un programma in parti indipendenti e l'esecuzione simultanea di queste istruzioni su più processori

# Parallelismo dei dati vs big data

- ✓ Ci sono due dimensioni che differenziano lo scope di applicabilità di CPU e GPU:
  - **livello di parallelismo**
  - **dimensione dei dati**

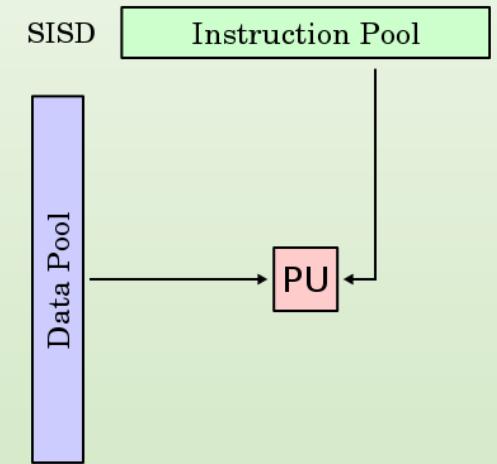


# Tassonomia di Flynn

**Parallel computer** = collezione di unità di elaborazione che comunicano e cooperano per risolvere velocemente un problema di grandi dimensioni

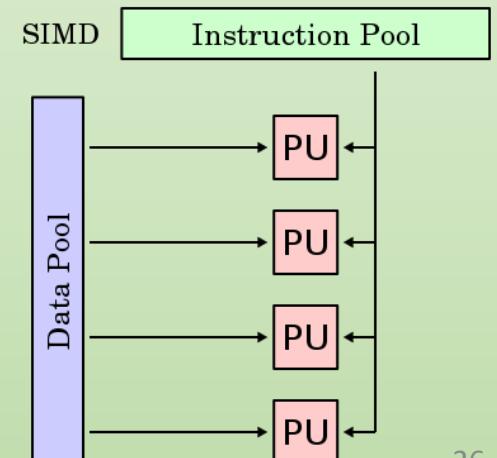
## ✓ SISD (Single Instruction Single Data)

- **Una unità** di computazione che accede a programma e dati
- **Nessun parallelismo**: le operazioni vengono eseguite sequenzialmente, su un dato alla volta (la classica architettura di von Neumann)



## ✓ SIMD (Single Instruction Multiple Data)

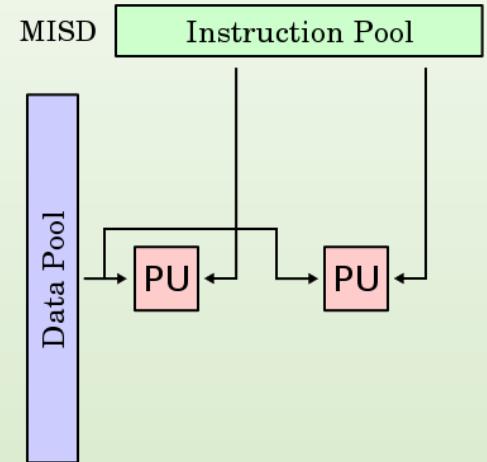
- **Molte unità** di computazione con accesso a memoria privata (condivisa o distribuita) per i dati , memoria globale unica per le istruzioni
- **Stessa istruzione**: ad ogni passo un processore centrale invia un'istruzione alle unità che leggono dati privati
- **Applicazioni** con alto grado di parallelismo ne traggono grande beneficio (multimedia, computer graphics, simulazioni, etc.)



# Tassonomia di Flynn

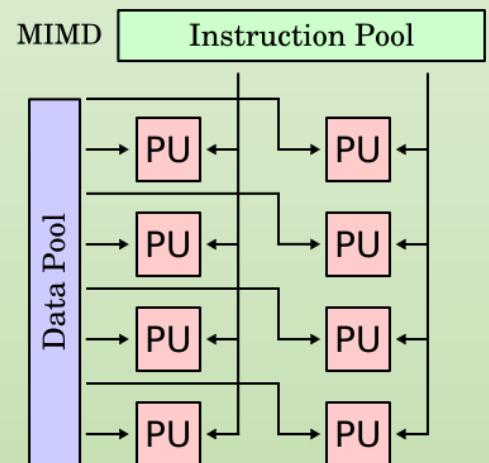
## ✓ MISD (Multiple Instruction Single Data)

- **Molte unità** di computazione con accesso a memoria privata di programma, accesso comune a memoria globale unica per i dati
- **Stesso dato:** ad ogni passo ogni unità ottiene lo stesso dato e carica un'istruzione da eseguire dalla memoria privata
- **Parallelismo** a livello istruzione su medesimo dato (nessun sviluppo commerciale noto)

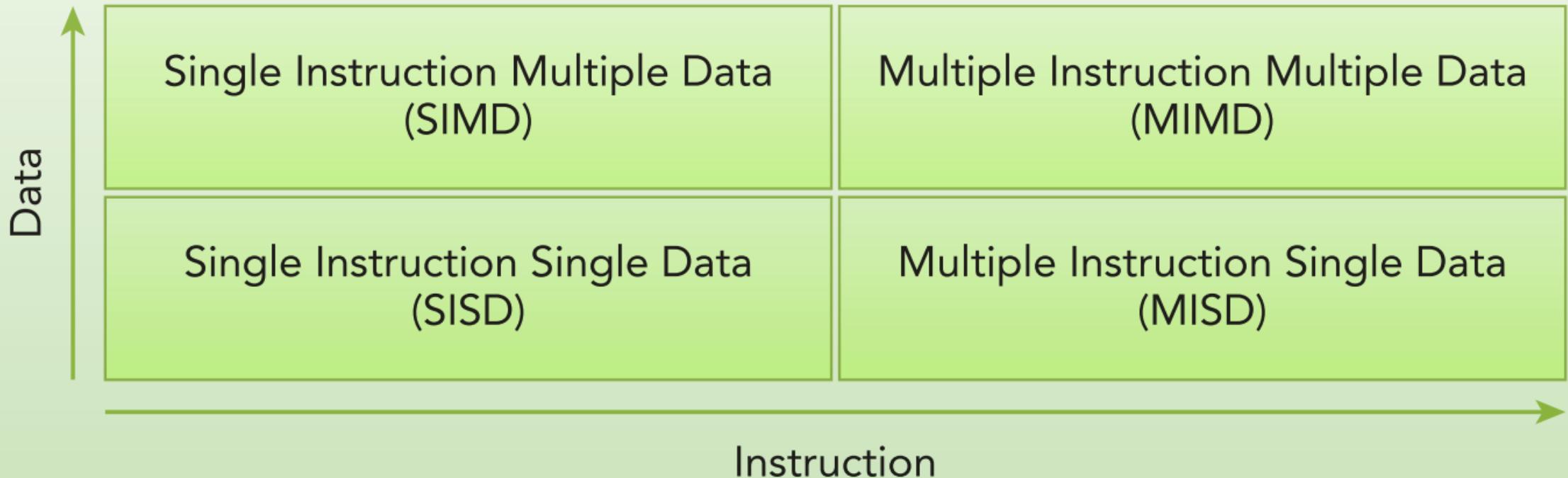


## ✓ MIMD (Multiple Instruction Multiple Data)

- **Molte unità** di computazione con accesso separato a istruzioni e dati su memoria condivisa o distribuita
- **Passo:** in uno step ogni unità carica la propria istruzione e la esegue su un dato separatamente e asincronicamente da altri
- **Esempi:** processori multicore, sistemi distribuiti, datacenter



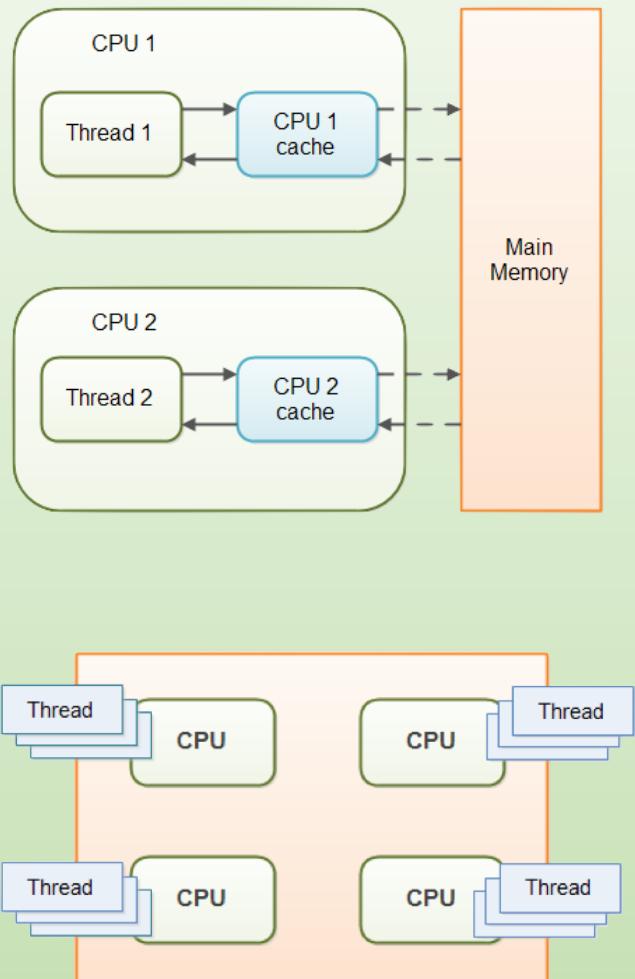
# Tassonomia di Flynn (sintesi)



# SIMT model (oltre Flynn)

## ✓ SIMT (Single Instruction Multiple Threads)

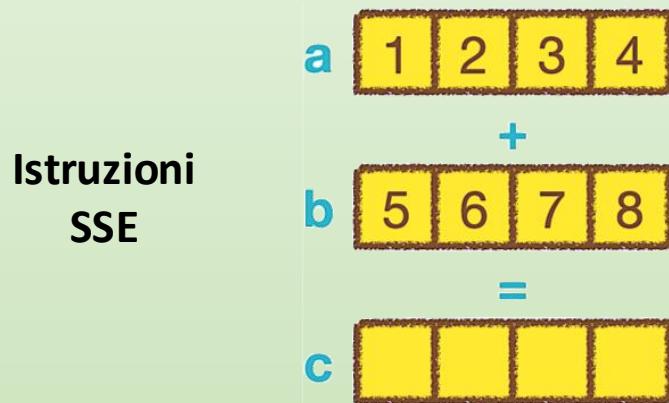
- **Multithreading**: in sistemi operativi multitasking rappresenta un diffuso modello di programmazione ed esecuzione che consente a thread multipli di coesistere all'interno del contesto di un processo in esecuzione
- **Modello SIMT** è usato in parallel computing dove si combina il modello SIMD con il multithreading
- **Condivisione** di risorse ma esecuzioni indipendenti che forniscono al programmatore una utile astrazione del concetto di esecuzione concorrente
- **Estensione** al modello di elaborazione che prevede che un processo abiliti esecuzioni parallele su sistemi multiprocessore o multicore
- **Implementato** su diverse GPU e fondamentale per GPGPU (introdotto d NVIDIA), per es. alcuni supercomputer combinano CPU con GPU



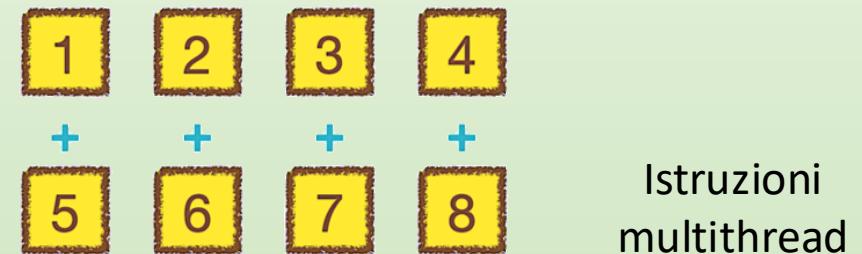
# SIMT vs SIMD

Il modello **SIMT** prevede tre caratteristiche chiave che **SIMD** non contempla:

1. Ogni thread ha il proprio **instruction address counter**
2. Ogni thread ha il proprio **register state** e in generale un **register set**
3. Ogni thread può avere un **execution path** indipendente



```
__m128 a = _mm_set_ps (4, 3, 2, 1);
__m128 b = _mm_set_ps (8, 7, 6, 5);
__m128 c = _mm_add_ps (a, b);
```



```
float a[4] = {1, 2, 3, 4};
float b[4] = {5, 6, 7, 8}, c[4];
{
    int id = ... ; // my thread ID
    c[id] = a[id] + b[id];
}
```

# NVIDIA GPUs

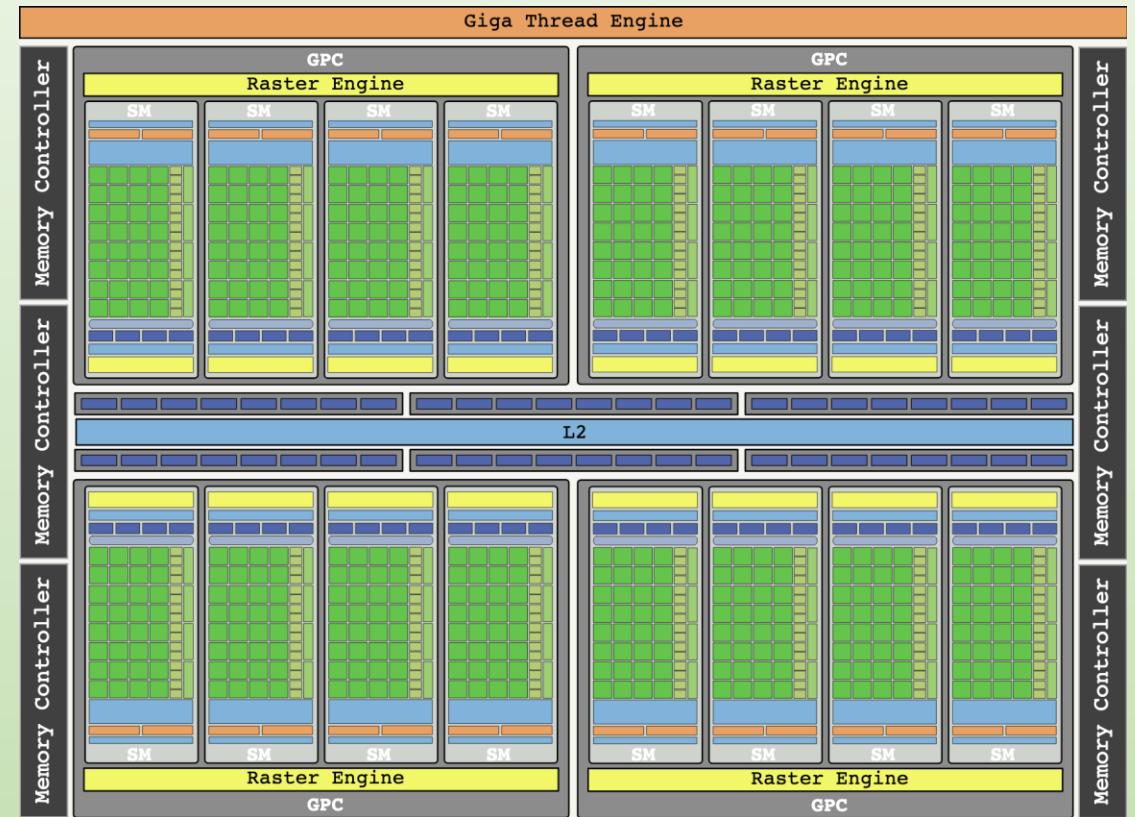
Architetture e modelli delle GPU

# Architetture per GPU NVIDIA

Compute Capability	Architettura	Esempi di GPU	Caratteristiche principali
1.x	Tesla	GeForce 8xxx, 9xxx, GTX 2xx	CUDA iniziale, no cache L1, doppia precisione limitata
2.x	Fermi	GTX 400, 500	Cache configurabile, doppia precisione migliorata
3.x	Kepler	GTX 600, 700, Tesla K20/K40	Dynamic Parallelism, migliori texture, maggiore efficienza energetica
5.x	Maxwell	GTX 900, Titan X (Maxwell)	NVLink introdotto, miglior efficienza energetica
6.x	Pascal	GTX 10xx, Tesla P100	Memoria unificata migliorata, aumento nei registri
7.x	Volta	Tesla V100	Tensor Cores per AI, memoria HBM2
8.x	Ampere	RTX 30xx, A100	Miglioramenti nei Tensor Cores e RT Cores, supporto FP16/TF32
9.x	Hopper	H100	Transformer Engine, avanzamenti nei Tensor Cores
10.x	Blackwell	RTX 50xx, B100	Tensor Core di nuova generazione, efficienza migliorata

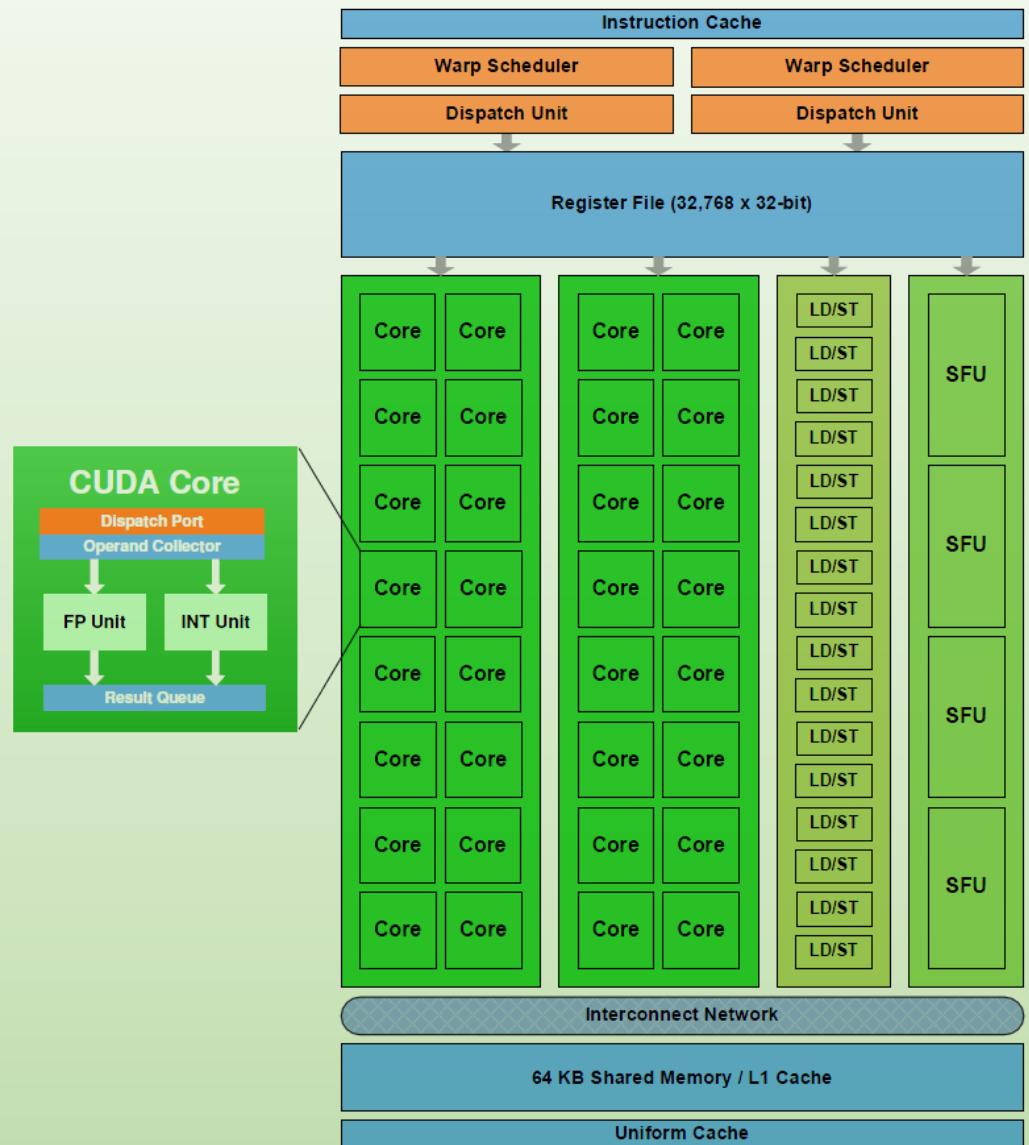
# Fermi microarchitecture (2010-2014)

- ✓ Parallel Processing: Each **Streaming Multiprocessor (SM)** had 32 CUDA cores, dual warp schedulers
- ✓ Enhanced Memory Hierarchy: Introduced L1 cache per SM, a unified L2 cache
- ✓ Improved **double-precision (FP64)** performance for scientific computing
- ✓ Allowed **multiple kernels** to run **simultaneously** and enabled faster **thread switching**
- ✓ Improved **atomic operations, thread synchronization, and integer processing**



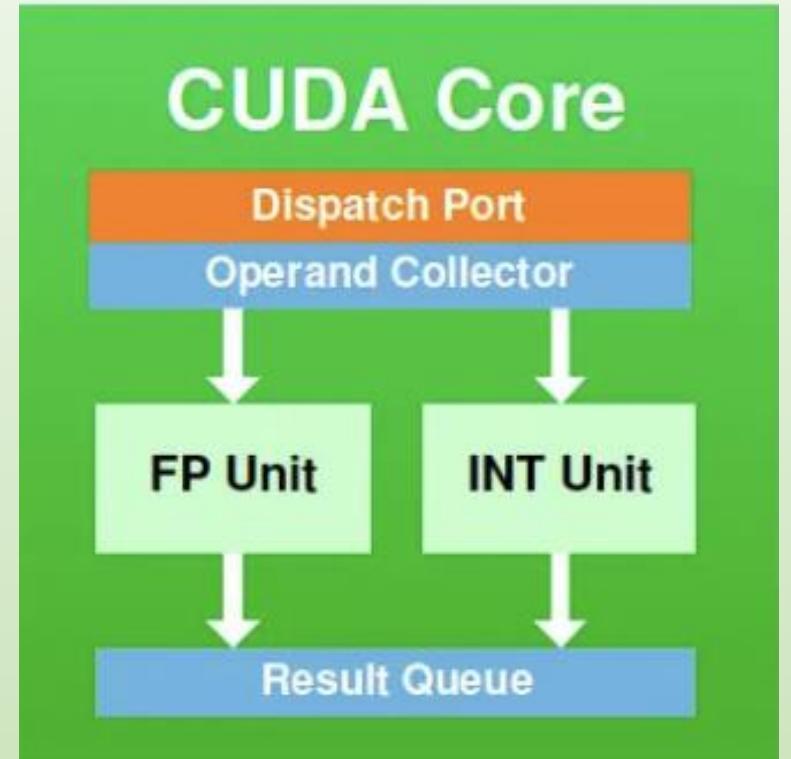
# SM (Streaming Multiprocessor)

- ✓ L'architettura GPU è costruita attorno a un array scalabile di **Streaming Multiprocessors (SM)**
- ✓ Ogni **SM** in una GPU è progettato per supportare l'esecuzione **concorrente** di centinaia di **thread**
- ✓ Molteplici **SM** per GPU NVIDIA GPU eseguono thread in **gruppi of 32** chiamati **warp**
- ✓ Tutti i **thread** in un **warp** eseguono la stessa istruzione allo stesso tempo



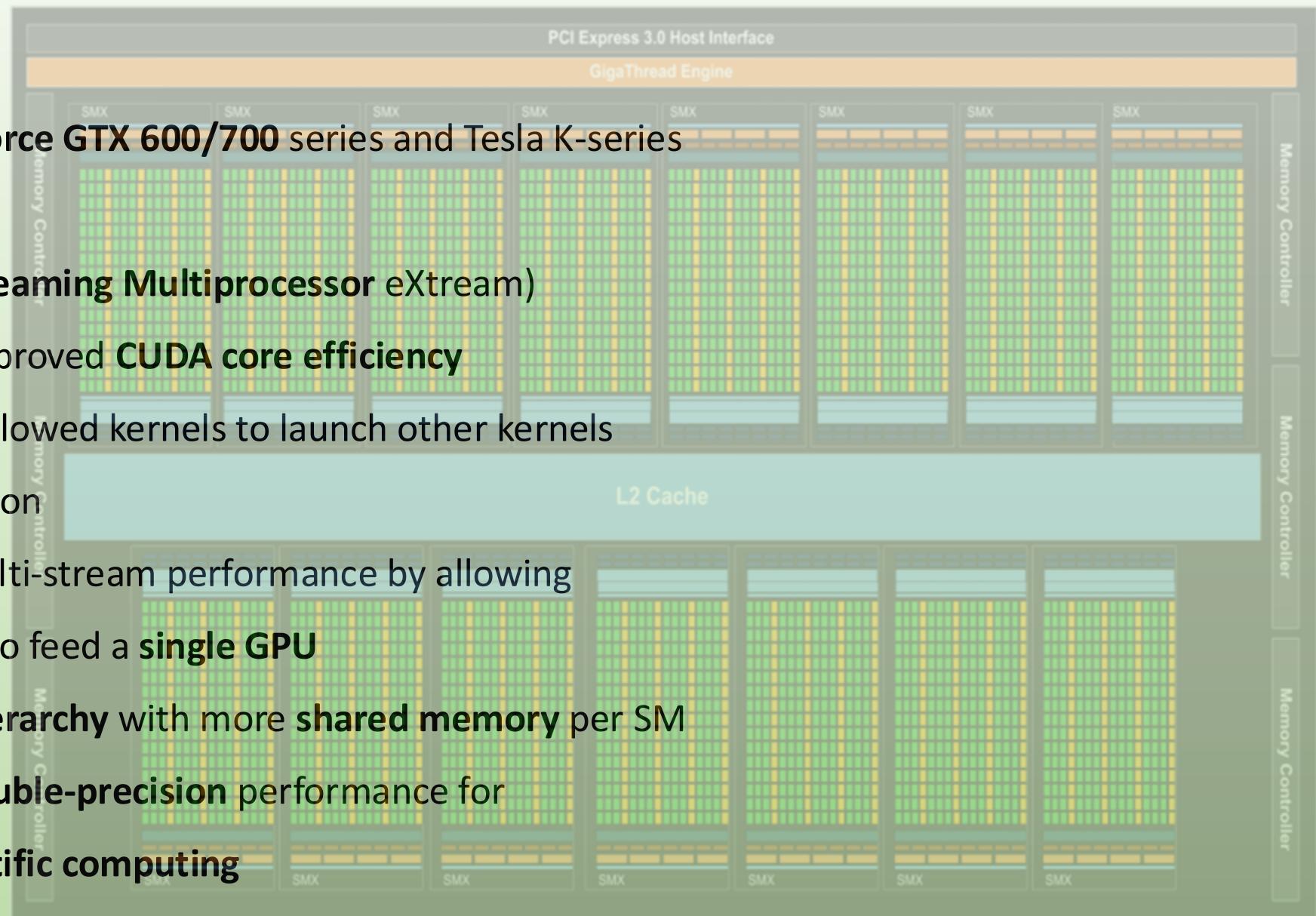
# Cuda core (Fermi)

- ✓ It's a vector processing component
- ✓ Works on a single operation
- ✓ It's the building block of SM
- ✓ Contains an FP and INT unit
- ✓ As the process reduces them (e.g. 28nm) they increase in number per SM



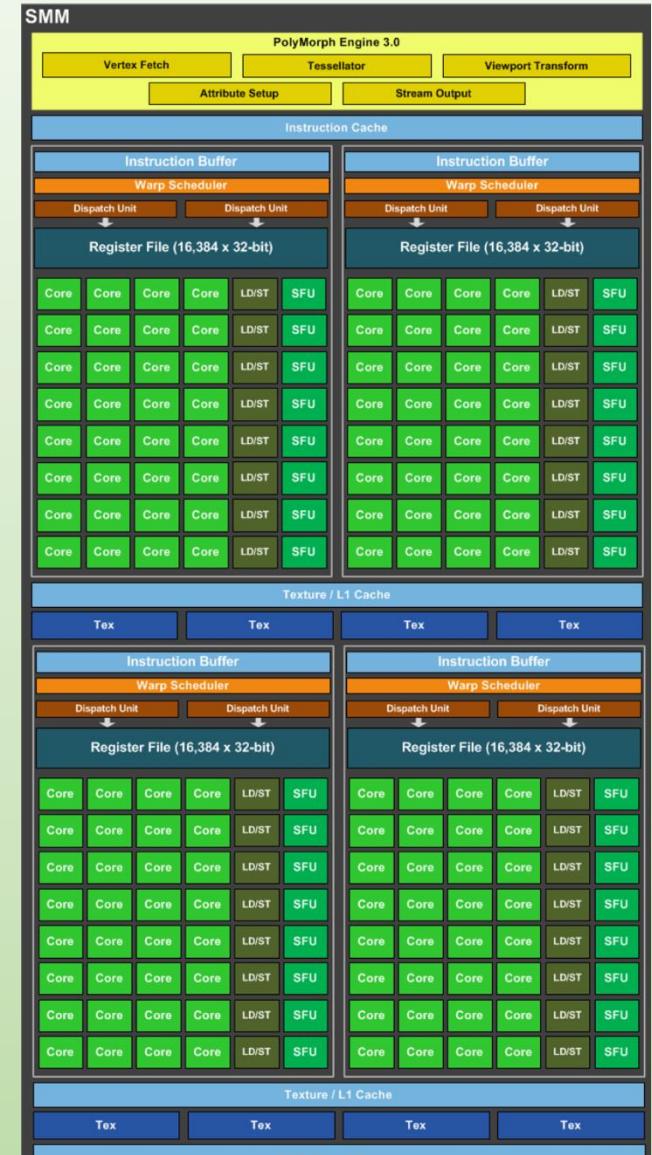
# Kepler Microarchitecture (2012-2014)

- ✓ Introduced in the **GeForce GTX 600/700** series and Tesla K-series GPUs
- ✓ Featured the **SMX (Streaming Multiprocessor eXtream)** architecture, which improved **CUDA core efficiency**
- ✓ **Dynamic Parallelism** allowed kernels to launch other kernels without CPU intervention
- ✓ **Hyper-Q** improved multi-stream performance by allowing multiple **CPU threads** to feed a **single GPU**
- ✓ Optimized **memory hierarchy** with more **shared memory** per SM
- ✓ Designed with high **double-precision** performance for professional and **scientific computing**



# Maxwell Microarchitecture (2014-2016)

- ✓ Replaced **SMX** with **SMM** (Streaming Multiprocessor Maxwell),  
improving **power efficiency** and **performance** per CUDA **core**
- ✓ Improved **instruction scheduling**, **cache hierarchy**, larger **L2 cache**
- ✓ Enhanced **power efficiency**, making it more suitable for **mobile** and  
**embedded** applications
- ✓ More **efficient CUDA cores**, delivering better performance at lower  
clock speeds compared to **Kepler**
- ✓ NVIDIA VXGI (Voxel Global Illumination) introduced for **real-time** global  
**illumination** in games



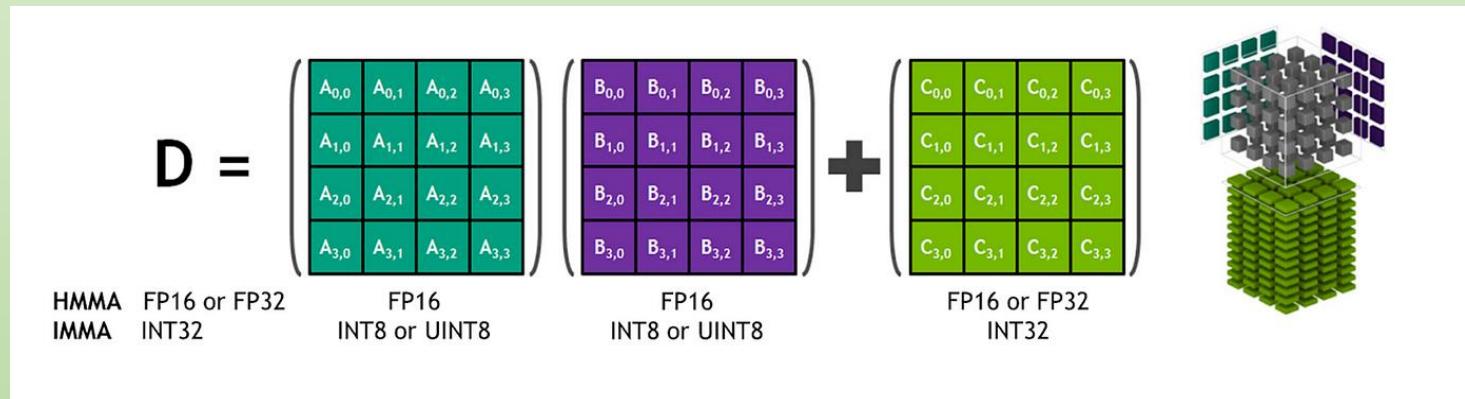
# Pascal Microarchitecture (2016-2018)

- ✓ New SM design, more SMs per GPU,  
higher **clock speed**
- ✓ **GDDR5X & HBM2** memory support  
Increased memory **bandwidth**
- ✓ Deep Learning & AI Acceleration: 16-bit  
**(FP16)** Half-Precision compute support
- ✓ Introduced **NVLink**, replacing **PCIe** for  
**GPU-to-GPU** communication, significantly  
improving **multi-GPU**
- ✓ **Unified memory**: programs can access  
both CPU and GPU RAM



# Volta/Turing Microarchitecture (2017-18)

- ✓ **Tensor Cores** provided dedicated hardware for matrix operations, massively accelerating deep learning workloads (e.g., **training** and **inference** in AI models)
- ✓ Redesigned **SMs** delivered higher **performance** per CUDA core
- ✓ **HBM2** (High Bandwidth Memory 2) providing **900 GB/s** memory **bandwidth**
- ✓ Enhanced CUDA **core efficiency** improving overall **FP32** and **FP64** performance
- ✓ **NVLink 2.0** faster GPU-to-GPU communication allowing **multiple** Volta **GPUs** to work together

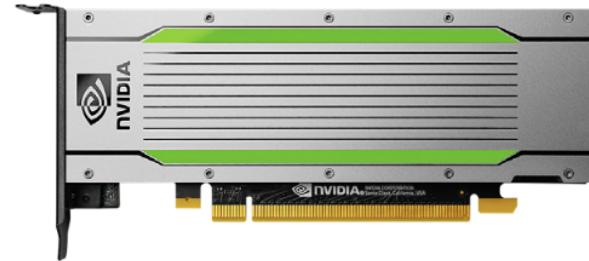


# Volta SM



# NVIDIA Tesla T4

Datacenter e HPC



## SPECIFICATIONS

GPU Architecture	<b>NVIDIA Turing</b>
NVIDIA Turing Tensor Cores	<b>320</b>
NVIDIA CUDA® Cores	<b>2,560</b>
Single-Precision	<b>8.1 TFLOPS</b>
Mixed-Precision (FP16/FP32)	<b>65 TFLOPS</b>
INT8	<b>130 TOPS</b>
INT4	<b>260 TOPS</b>
GPU Memory	<b>16 GB GDDR6 300 GB/sec</b>
ECC	<b>Yes</b>
Interconnect Bandwidth	<b>32 GB/sec</b>
System Interface	<b>x16 PCIe Gen3</b>
Form Factor	<b>Low-Profile PCIe</b>
Thermal Solution	<b>Passive</b>
Compute APIs	<b>CUDA, NVIDIA TensorRT™, ONNX</b>

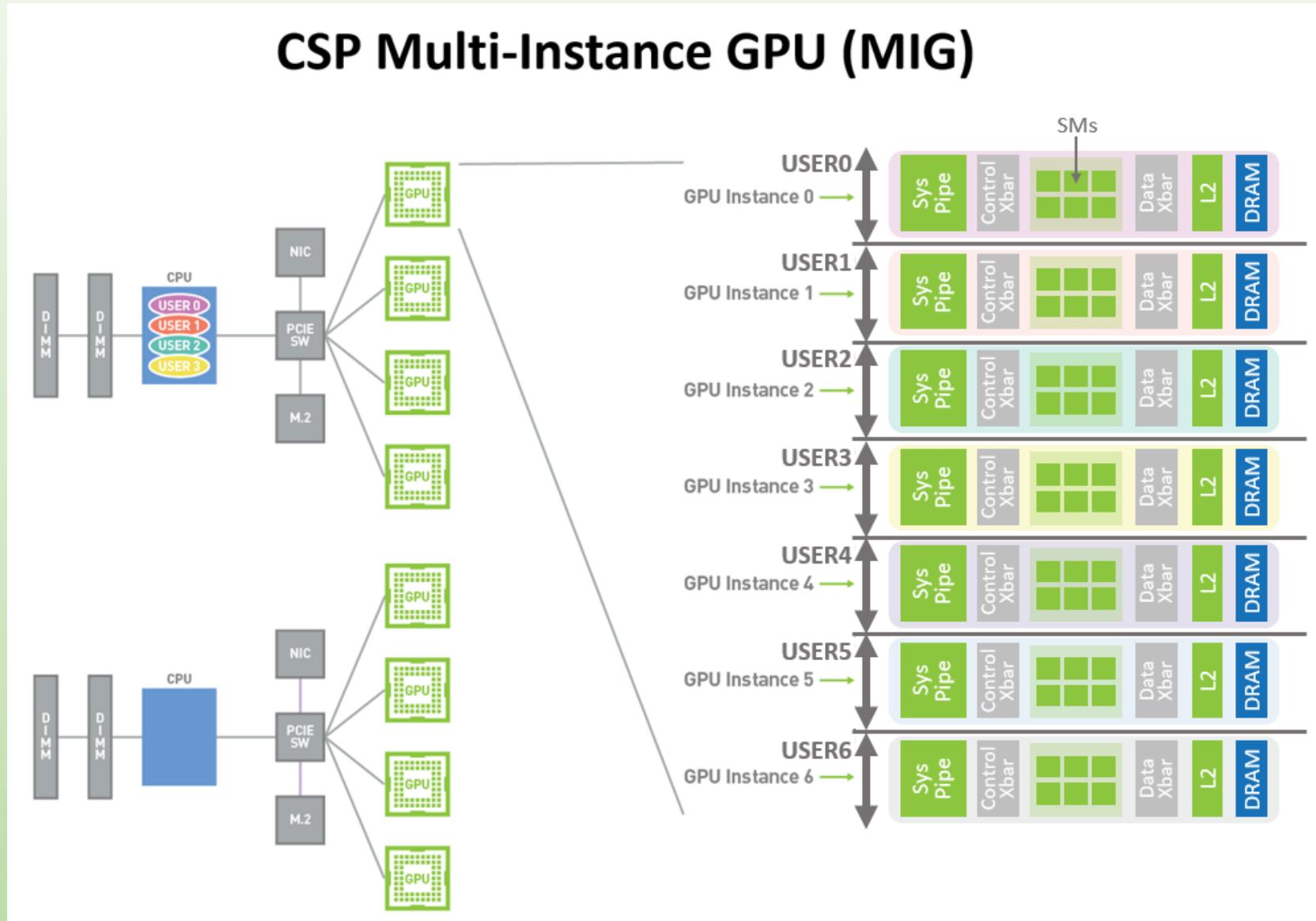
# Ampere Microarchitecture (2020)

- ✓ Enhanced CUDA Cores: mix of **FP32** and **INT32** units, doubling **FP32** throughput comp. to Turing
- ✓ Third-Generation **Tensor Cores**: Improved AI performance with sparse matrix acceleration
- ✓ Second-Generation **RT Cores**: Better **real-time ray tracing** performance with hardware-accelerated triangle intersection and motion **blur support**
- ✓ Memory and **Bandwidth Enhancements**: Uses **GDDR6X** memory (for gaming) and **HBM2e** (for data center)
- ✓ **Multi-Instance GPU (MIG)** for A100: Allows data center GPUs to be partitioned into **smaller independent instances** for better resource allocation
- ✓ **PCIe 4.0** and **NVLink 3.0**: Provides higher bandwidth for faster **CPU-GPU** and **GPU-GPU communication**

# A100 GPU hardware architecture

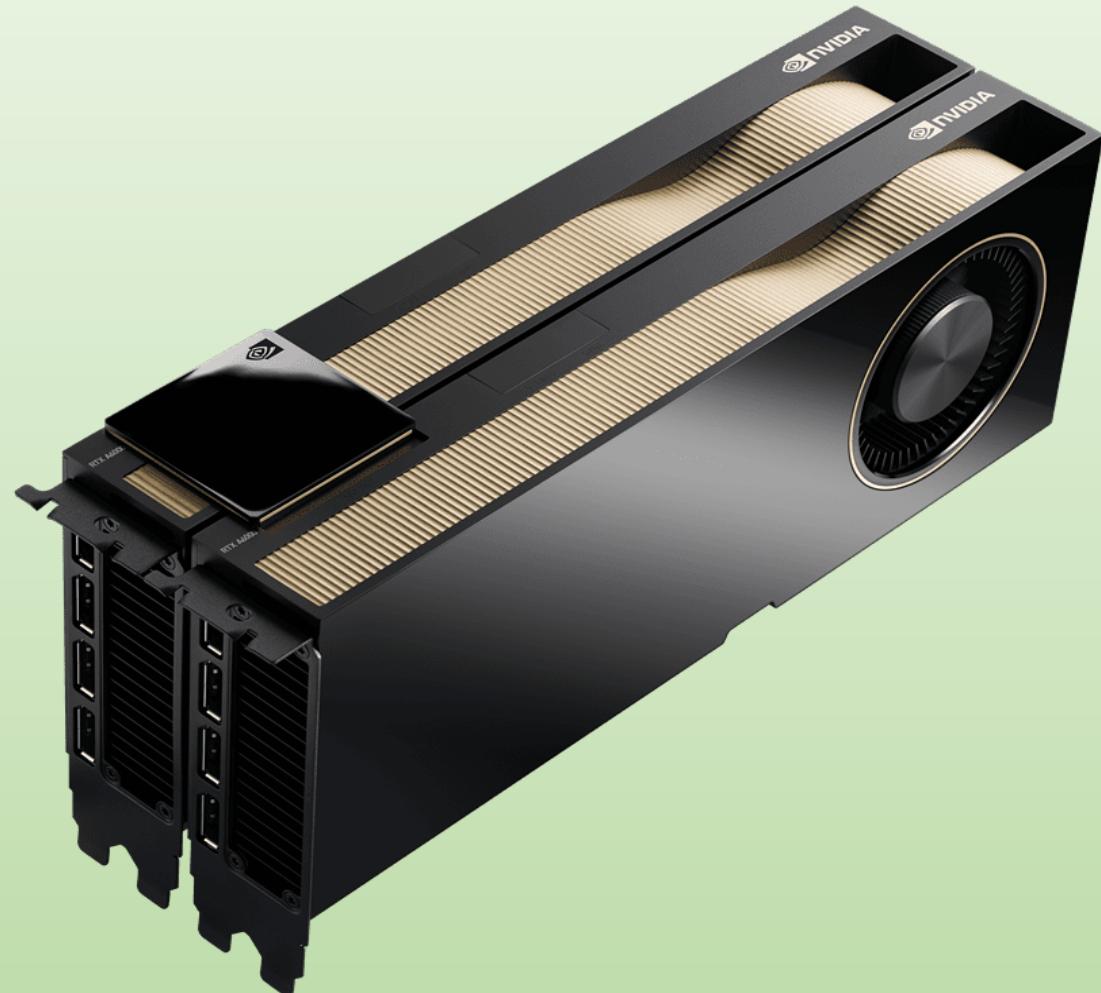


# MIG architecture (Ampere)



# NVIDIA Quadro A6000

Grafica professionale (CAD, CAE) e visualizzazione

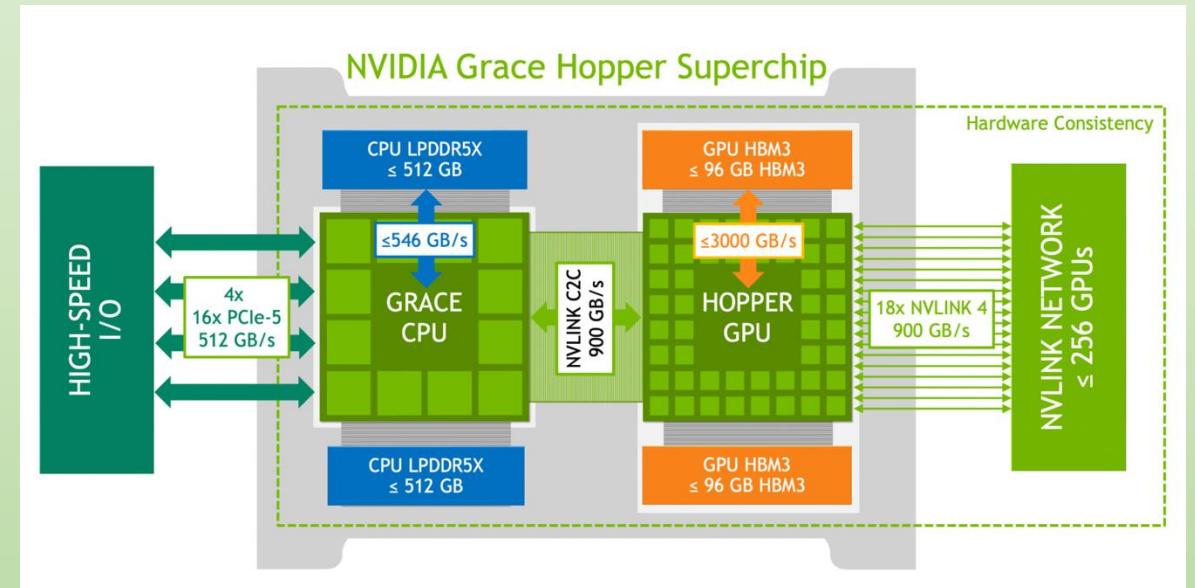


# Hopper Microarchitecture (2022)

- ✓ **Transformer Engine:** Optimized for deep learning, particularly **Transformer models** used in **GPT, BERT LLMs**
- ✓ Fourth-Generation **Tensor Cores:** Enhanced matrix multiplication and **AI acceleration** with FP8 and FP16 support, Delivers up to **9x faster AI** training compared to Ampere
- ✓ **3NVLink 4.0** supports **PCIe Gen 5.0** boosting data transfer speeds
- ✓ **Confidential Computing & Multi-Instance GPU (MIG):** protects sensitive workloads with **encrypted memory**
- ✓ Improved Memory System: **80GB HBM3** memory, with up to **3TB/s bandwidth**
- ✓ **Dynamic Programming & Workload Efficiency:** Introduces **Thread Block Clusters**, enhancing parallel processing efficiency

# Grace-Hopper superchip (2022)

- ✓ ARM Neoverse-Based Design: Uses **ARM Neoverse V2 cores**, optimized for **HPC** and **AI** workloads, supports up to **72 cores**, delivering high throughput and **parallelism**
- ✓ **LPDDR5X** Memory offering up to **1TB/s** memory **bandwidth**, less power consumption compared to DDR5
- ✓ **Optimized for AI and HPC applications** that require fast data access
- ✓ **NVLink-C2C** for High-Speed CPU-GPU Communication (**7x faster than PCIe Gen 5**)
- ✓ Optimized for scalability in multi-node HPC clusters
- ✓ The **Grace CPU** is designed for **data centers, AI training, cloud computing, and supercomputing**, providing a **power-efficient alternative to x86 CPUs**



# Blackwell microarchitecture (2024)

- ✓ **Advanced AI Capabilities:** Blackwell introduces AI multi-frame generation, enhancing DLSS 4's frame rates by up to 2x compared to its predecessors, while maintaining or exceeding native image quality with low system latency.
- ✓ **Neural Shaders:** The architecture incorporates RTX Neural Shaders, integrating small neural networks into programmable shaders, paving the way for a new era of graphics innovation.
- ✓ **AI Management Processor (AMP):** Blackwell features an AI Management Processor that enables multiple AI models—such as speech, translation, vision, animation, and behavior—to share the GPU simultaneously with graphics workloads.
- ✓ **Enhanced Performance and Efficiency:** The architecture delivers up to 30 times more performance and 25 times greater energy efficiency compared to its predecessor, the Hopper GPU generation, making it highly suitable for data center-scale generative AI workflows.

# RTX 50 Series Blackwell GPUs



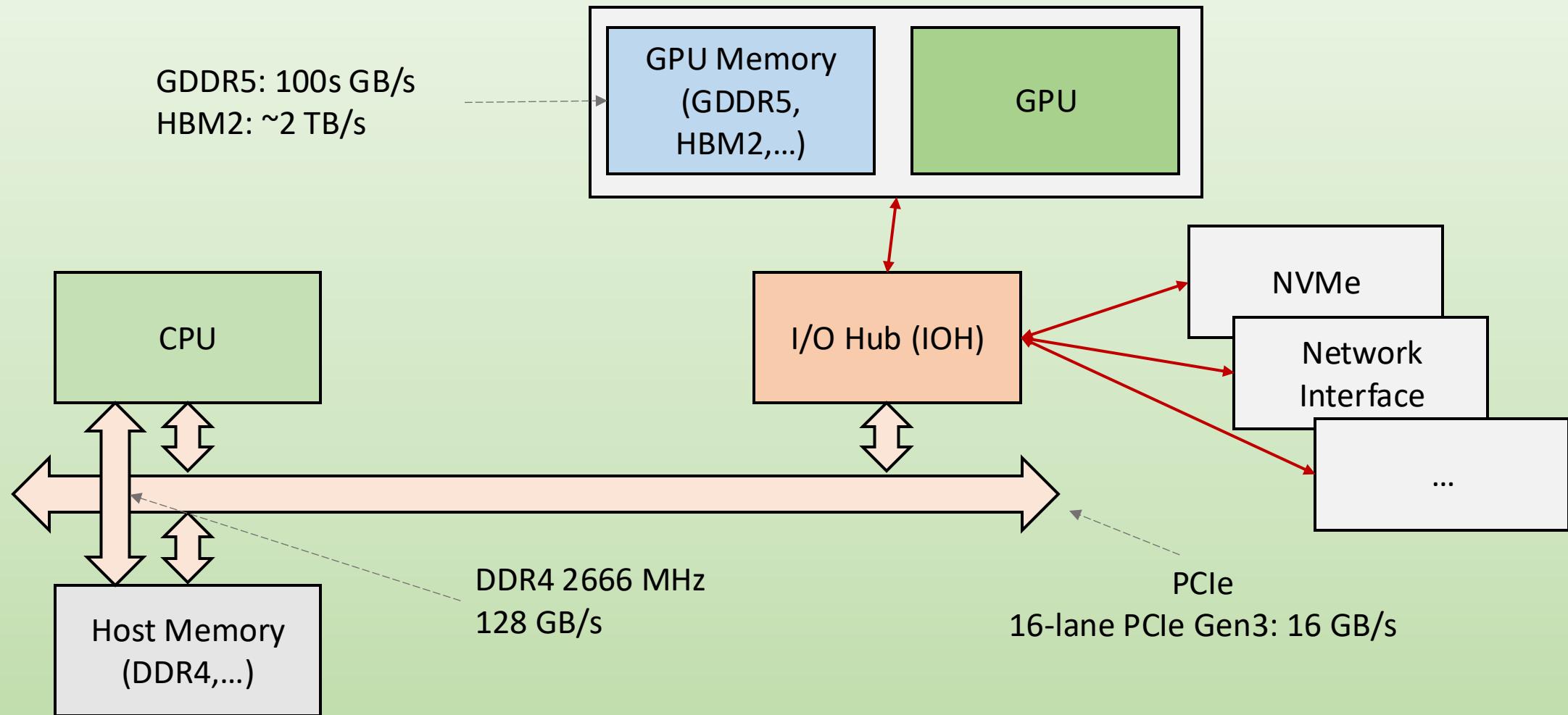
# Compute capability

- ✓ La **Compute Capability (CC)** di NVIDIA è la versione dell'architettura CUDA supportata da una **GPU NVIDIA**
- ✓ Definisce le funzionalità hardware disponibili, come il numero di core CUDA, il supporto per le istruzioni avanzate, l'uso della memoria, risorse, etc.

## Data Center Products

GPU	Compute Capability
NVIDIA Blackwell GPU (GB200)	10.0
NVIDIA Blackwell GPU (B200)	10.0
NVIDIA H200	9.0
NVIDIA H100	9.0
NVIDIA L4	8.9
NVIDIA L40S	8.9
NVIDIA L40	8.9
NVIDIA A100	8.0
NVIDIA A40	8.6
NVIDIA A30	8.0
NVIDIA A10	8.6
NVIDIA A16	8.6
NVIDIA A2	8.6
NVIDIA T4	7.5
NVIDIA V100	7.0
Tesla P100	6.0
Tesla P40	6.1
Tesla P4	6.1
Tesla M60	5.2
Tesla M40	5.2
Tesla K80	3.7
Tesla K40	3.5
Tesla K20	3.5
Tesla K10	3.0

# System Architecture Snapshot With a GPU



# CUDA zone

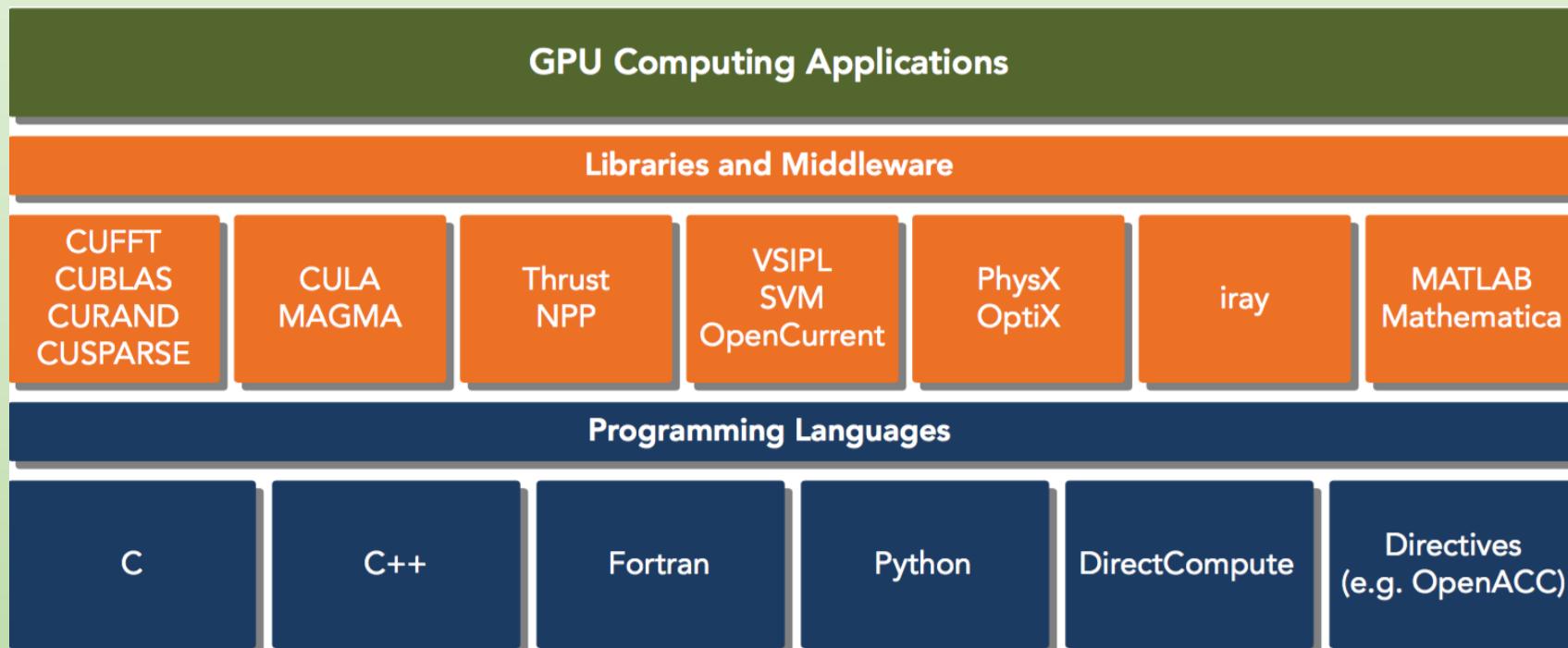
CUDA toolkit

# CUDA: Compute Unified Device Architecture

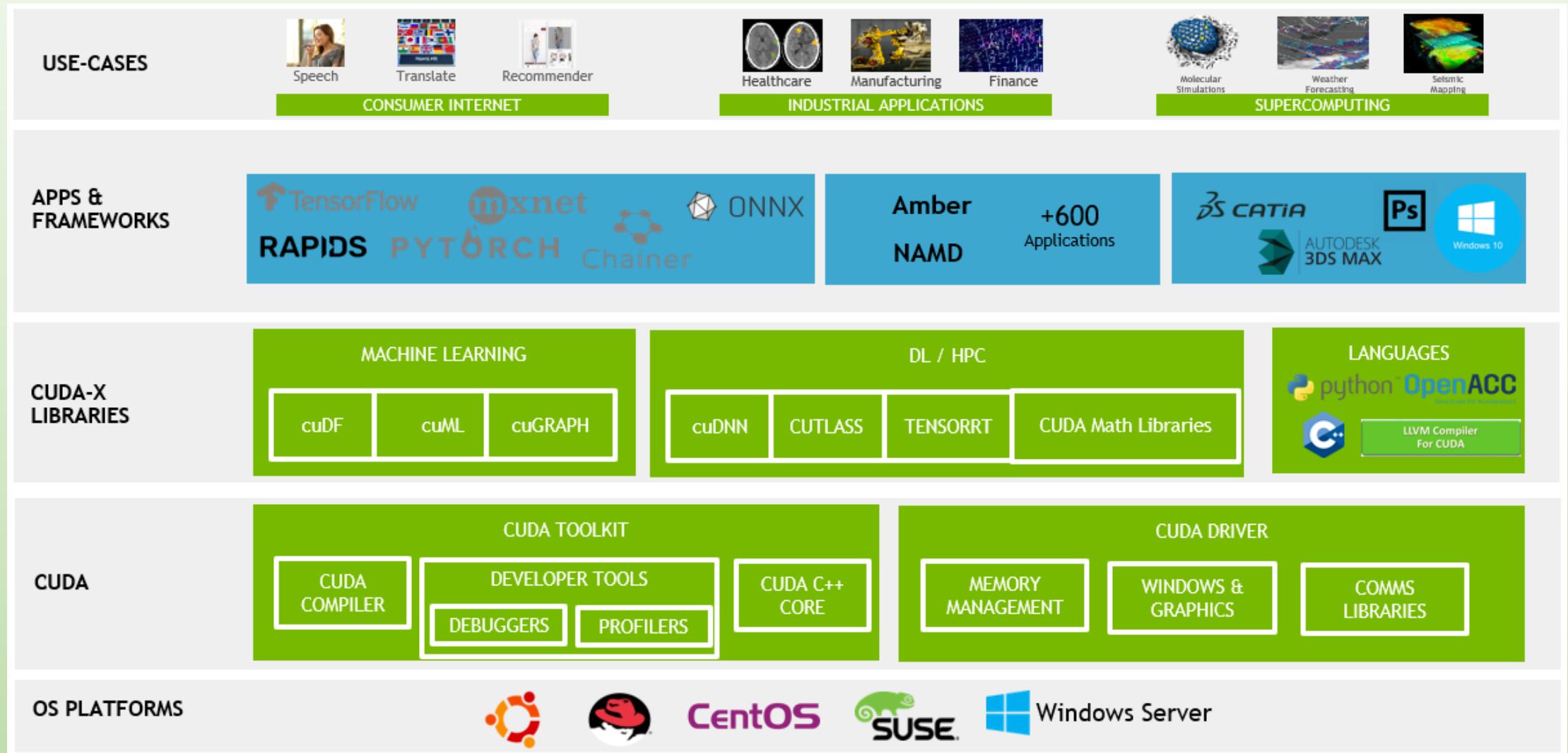
- ✓ It enables a **general-purpose programming model** on NVIDIA GPUs
- ✓ Current CUDA toolkit 12.4
- ✓ Enables explicit GPU **memory management**
- ✓ The **GPU** is viewed as a **compute device** that:
  - Is a **co-processor** to the CPU (or host)
  - Has its own **DRAM** (global memory in CUDA parlance)
  - Runs many **threads** in parallel

# CUDA platform

- ✓ The **CUDA platform** is accessible through CUDA-accelerated libraries, compiler directives, application programming interfaces (API), and extensions to industry-standard programming languages, including C, C++, Fortran, and Python
- ✓ **CUDA C** is an extension of standard ANSI C with a handful of language extensions to enable heterogeneous programming, and straightforward APIs to manage devices, memory, and other tasks.

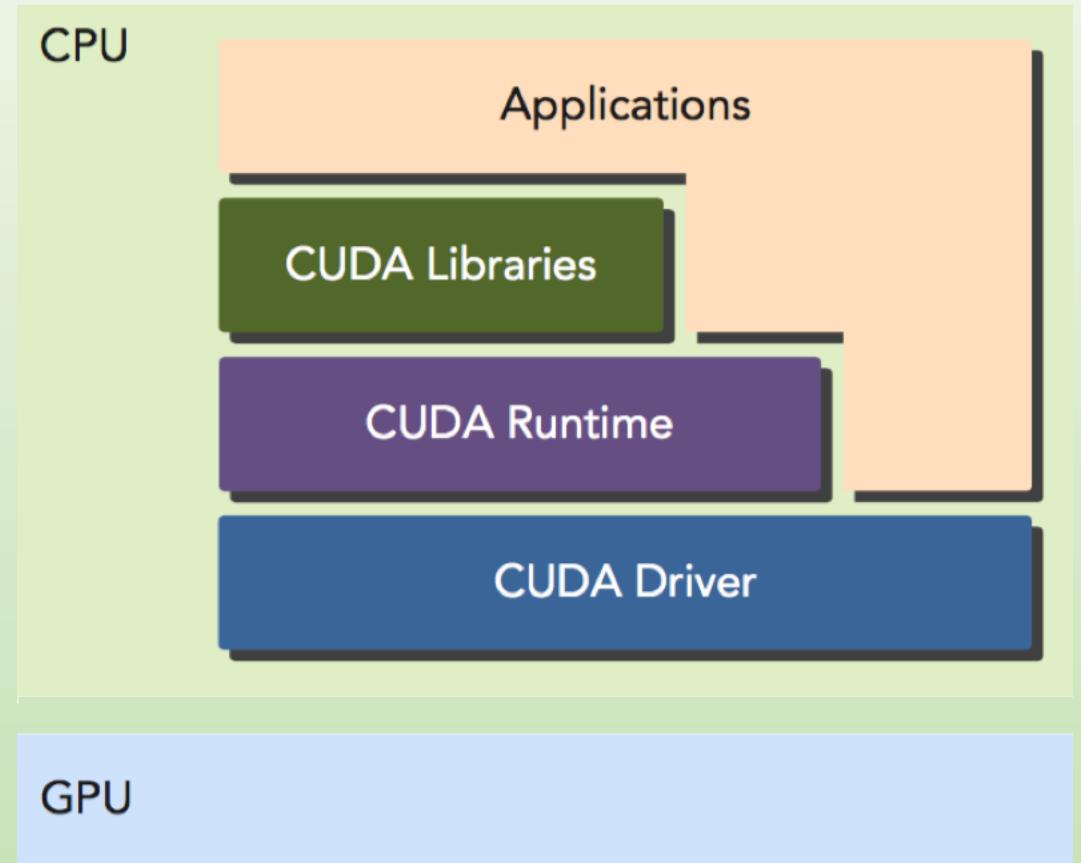


# CUDA platform



# CUDA APIs

- ✓ CUDA fornisce due livelli di API per la gestione della GPU e l'organizzazione dei thread
  - ✓ **CUDA Runtime API**
  - ✓ **CUDA Driver API**
- ✓ Le **driver API** sono API a **basso livello** e piuttosto difficili da programmare ma danno un maggior controllo della GPU
- ✓ Le **runtime API** sono API ad **alto livello** implementate sulle driver API
- ✓ Ogni **funzione** delle runtime API è **suddivisa** in diverse **operazioni basilari** fornite dalle driver API



# Differenze tra API

Caratteristica	Driver API (cu*)	Runtime API (cuda*)
<b>Livello di astrazione</b>	Basso (più complesso)	Alto (più facile da usare)
<b>Gestione memoria</b>	Manuale (cuMemAlloc())	Automatica (cudaMalloc())
<b>Gestione moduli PTX</b>	Sì (cuModuleLoad())	No
<b>Compilazione</b>	Può essere usata senza nvcc	Richiede nvcc
<b>Dipendenza dalla versione</b>	Indipendente	Legata alla versione del driver
<b>Usabilità</b>	Complessa	Più semplice

## ◆ Runtime API Caratteristiche:

- Più user-friendly, con funzioni come cudaMalloc(), cudaMemcpy(), cudaFree()
- Richiede che il codice sia compilato con nvcc, che lega la versione del runtime alla versione del driver
- Più adatta alla maggior parte delle applicazioni CUDA standard

## ◆ Driver API Caratteristiche:

- Non dipende dal CUDA runtime (quindi può essere usata senza legame con versioni specifiche di CUDA)
- Permette la gestione manuale di **moduli PTX/CUBIN**, che possono essere compilati ed eseguiti a runtime
- È più complessa da usare, con funzioni come culInit(), cuMemAlloc(), cuModuleLoad()
- Più stabile tra versioni diverse di CUDA

# Difference between the driver and runtime APIs

The driver and runtime APIs are very similar and can for the most part be used interchangeably. However, there are some key differences worth noting between the two

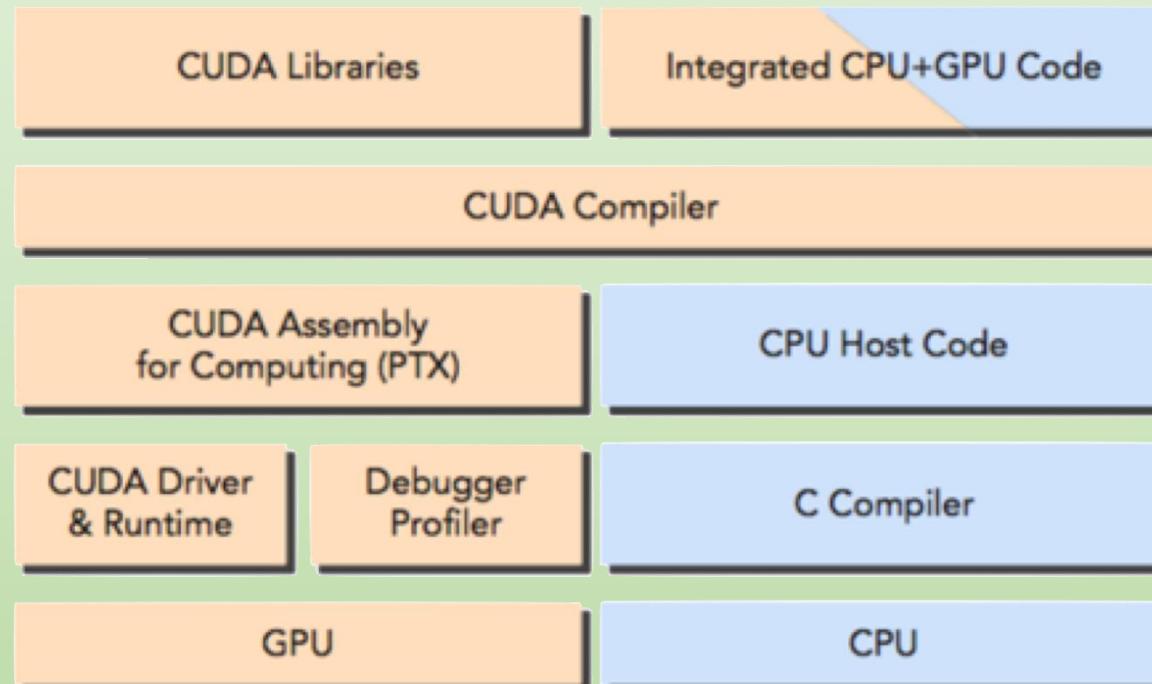
## Complexity vs. control

The **runtime API** eases device code **management** by providing **implicit initialization, context management, and module management**. This leads to simpler code, but it also **lacks the level of control** that the **driver API** has.

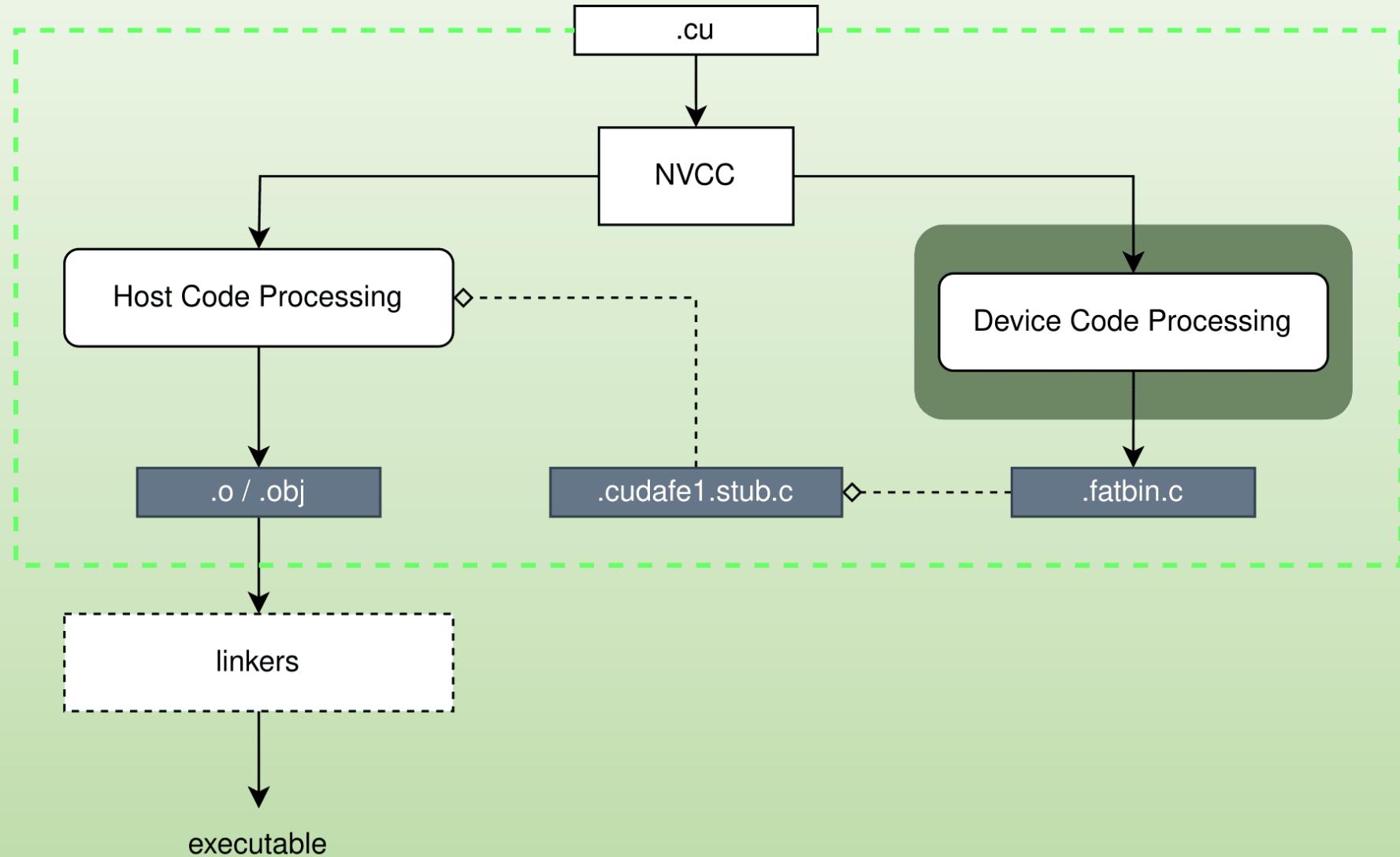
In comparison, the **driver API** offers more **fine-grained control**, especially over contexts and module loading. **Kernel launches** are much **more complex** to implement, as the execution **configuration** and kernel **parameters** must be specified with **explicit** function calls. However, unlike the runtime, where all the kernels are automatically loaded during initialization and stay loaded for as long as the program runs, with the driver API it is possible to only keep the modules that are currently needed loaded, or even dynamically reload modules. The **driver API** is also **language-independent** as it only deals with **cubin** objects.

# Un programma CUDA

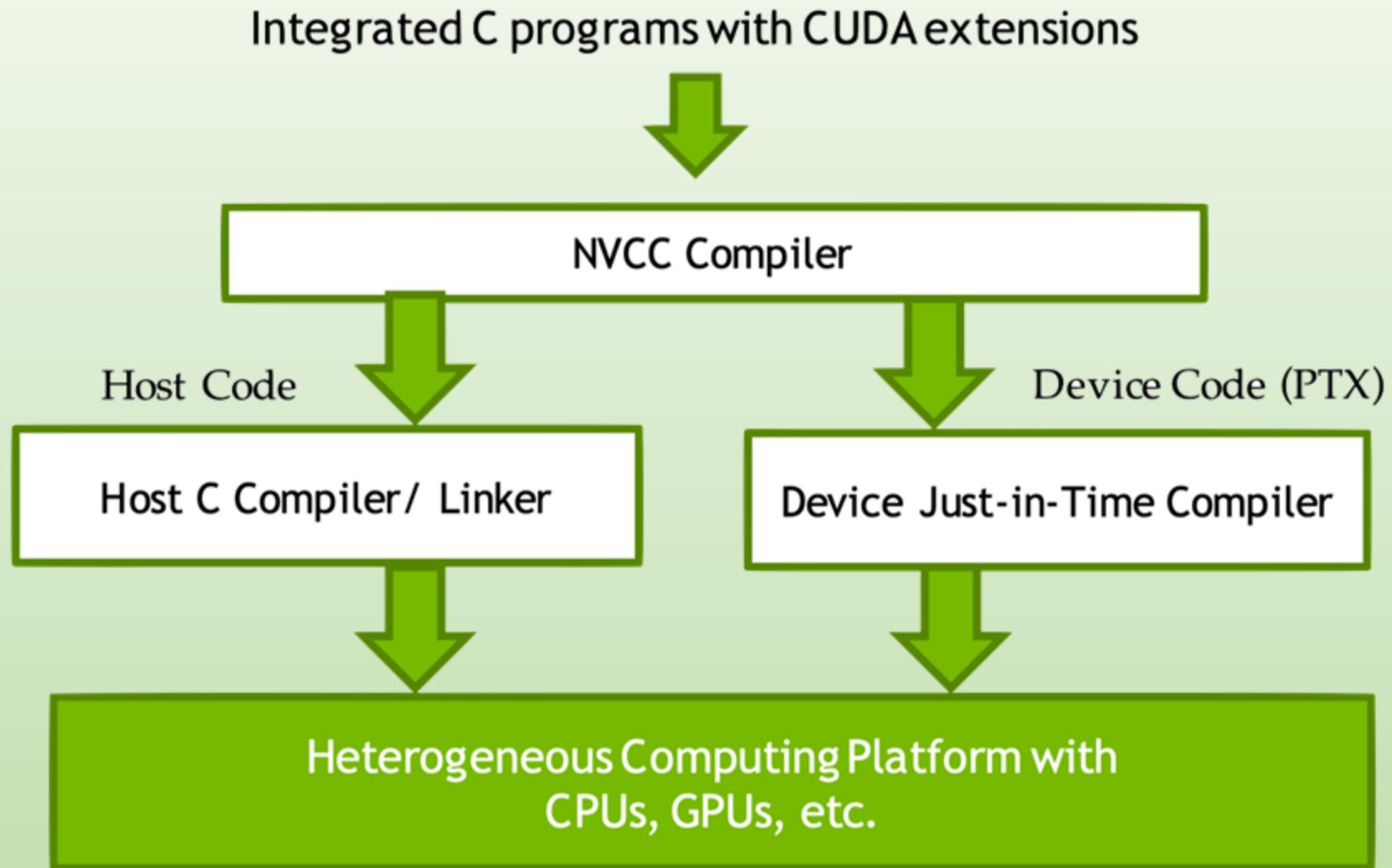
- ✓ Un programma CUDA consiste di una mistura di due parti:
  1. il **codice host** eseguito su CPU
  2. il **codice device** eseguito su GPU
- ✓ Il compilatore **nvcc** CUDA separa il codice host da quello device durante la **compilazione**



# Compilazione di un programma CUDA



# Compilazione di un programma CUDA



# Questo corso

- ✓ Introduzione ai sistemi di calcolo eterogenei basati su CPU e GPU
- ✓ Programmazione parallela e il concetto di GPGPU (General Purpose GPU)
- ✓ Il modello di programmazione CUDA: API e linguaggio CUDA C
- ✓ Il modello di esecuzione CUDA: kernel, blocchi, thread e sincronizzazione
- ✓ Il modello di memoria di CUDA: globale, condivisa e costante
- ✓ Stream, concorrenza e ottimizzazione delle prestazioni
- ✓ Librerie di CUDA SDK accelerate da GPU
- ✓ Pattern di parallelismo negli algoritmi
- ✓ Elementi di deep learning in CUDA/Python
- ✓ Sviluppo e implementazione di applicazioni su GPU NVIDIA

# Riferimenti bibliografici

## Testi generali:

1. J. Cheng, M. Grossman, T. Mckercher, [Professional Cuda C Programming](#), Wrox Pr Inc. (1. edizione), 2014 (cap 1)
2. Jason Sanders, Edward Kandrot, CUDA by examples: an introduction to general-purpose GPU programming, Pearson Education, 2011
3. Dispensa (online) di parallel computing:  
<http://www.cse.unt.edu/~tarau/teaching/parpro/papers/Parallel%20computing.pdf>

## NVIDIA docs:

1. CUDA Programming: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/>
2. CUDA C Best Practices Guide: <http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/>