

GPUS e CUDA

Thiago Vivan Bastos

Fernando William Cruz

Bruno Martins Valério Bomfim

CPU x GPU

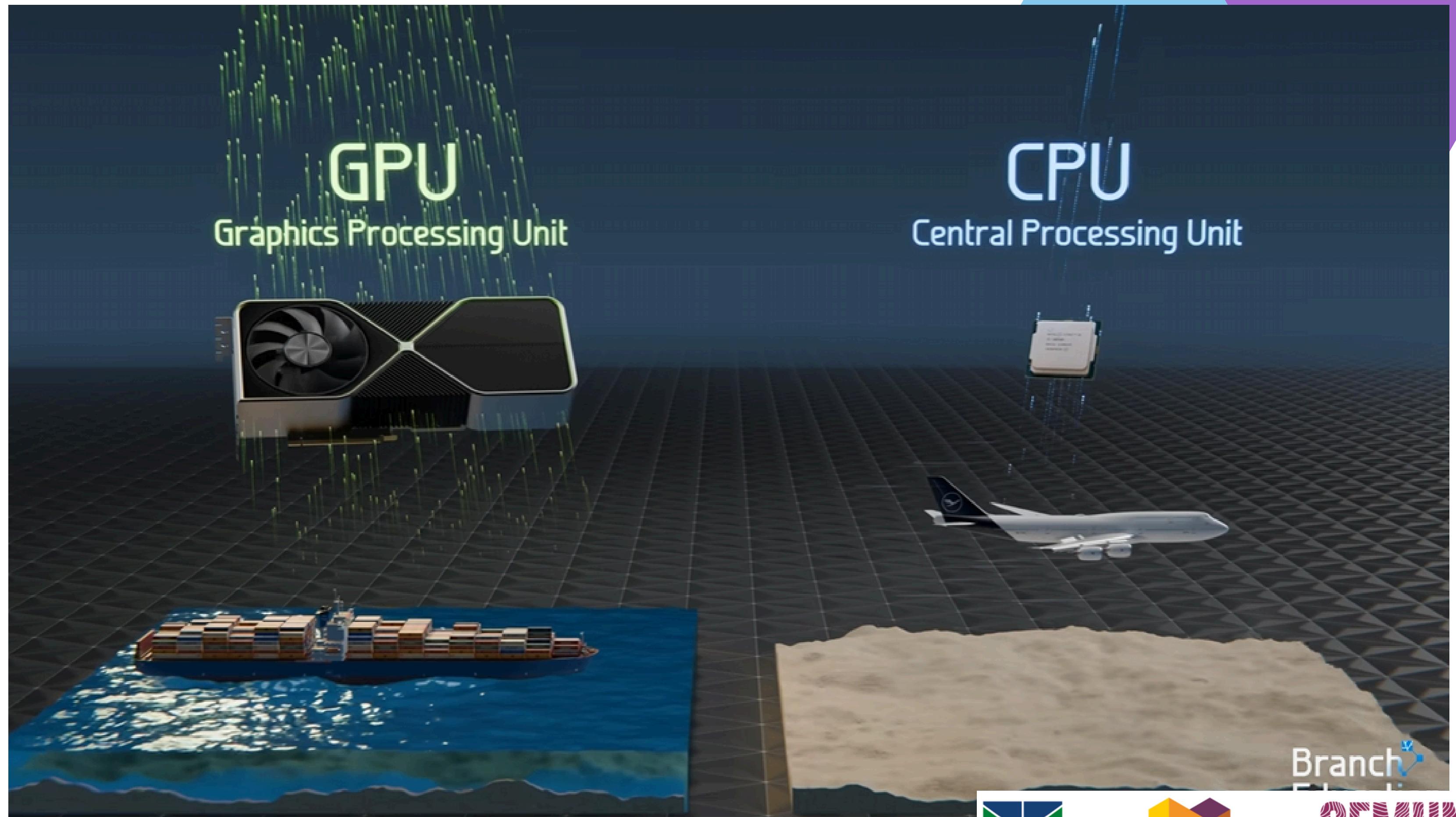
CPU

- Flexível
- Tarefas Sequenciais;
- Unidade de controle central
- Operações complexas
- Suporte a Sistemas operacionais, I/O, NET

GPU

- Processamento simultâneo;
- Renderização;
- Cálculos simples;
- $A \times B + C = D$ (FMA - Fused Multiply and add)
- Sem suporte a OS's, I/O, NET





Aspecto	CPU	GPU
Propósito	Processamento geral e serial	Processamento paralelo massivo
Núcleos	Poucos núcleos otimizados para desempenho	Milhares de núcleos pequenos e paralelos
Desempenho	Alta eficiência em tarefas sequenciais	Superior em tarefas paralelas
Aplicações	Sistemas operacionais, aplicativos gerais	Renderização gráfica, IA, modelagem 3D
Eficiência energética	Variável, foco em otimização para cargas mistas	Altamente eficiente em paralelismo
Memória	Cache maior e mais rápido	Memória de alta largura de banda
Custo	Geralmente menor por núcleo	Pode ser mais caro devido à especialização

Clusters em CPU's e GPU's

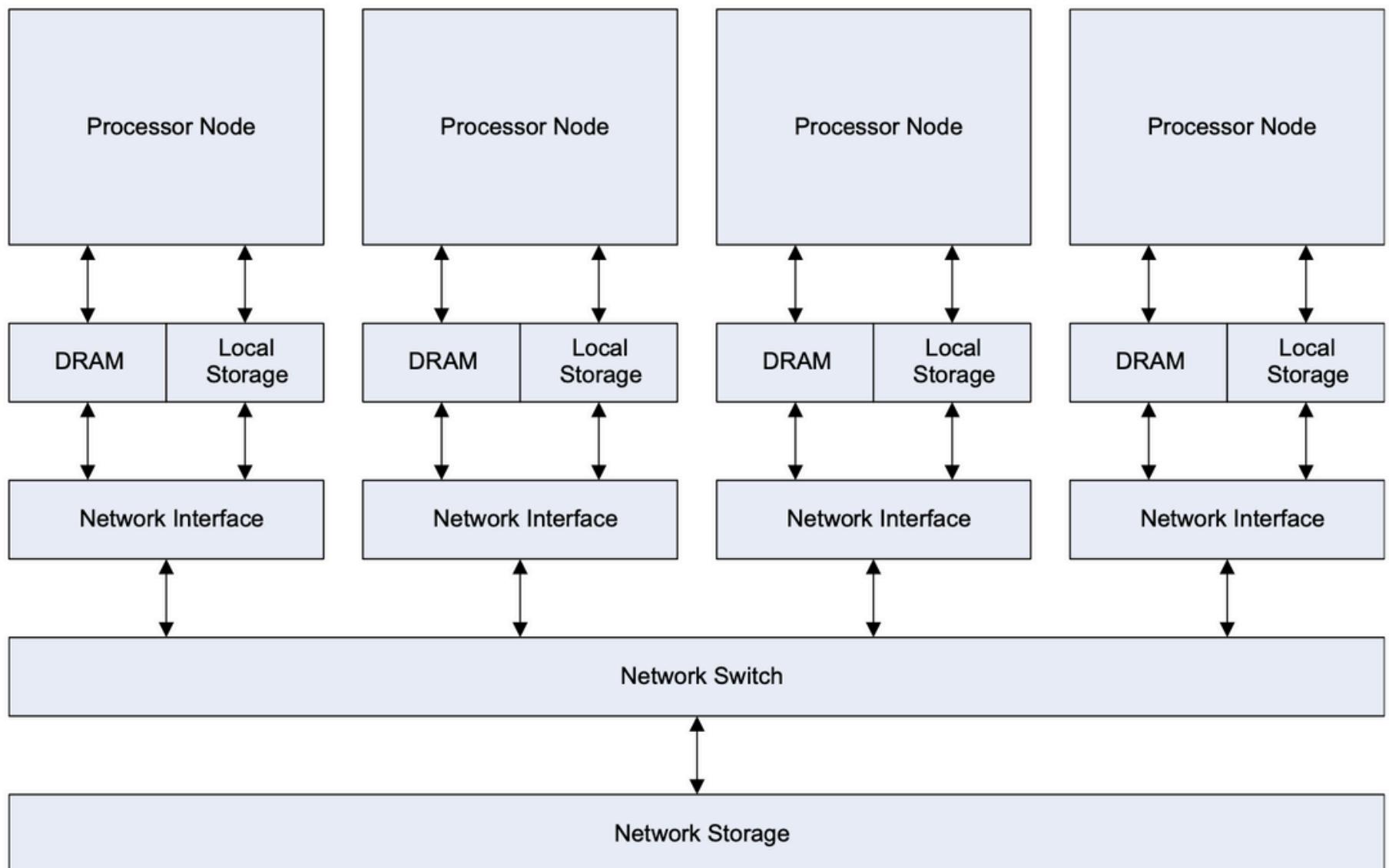


FIGURE 1.6

Typical cluster layout.

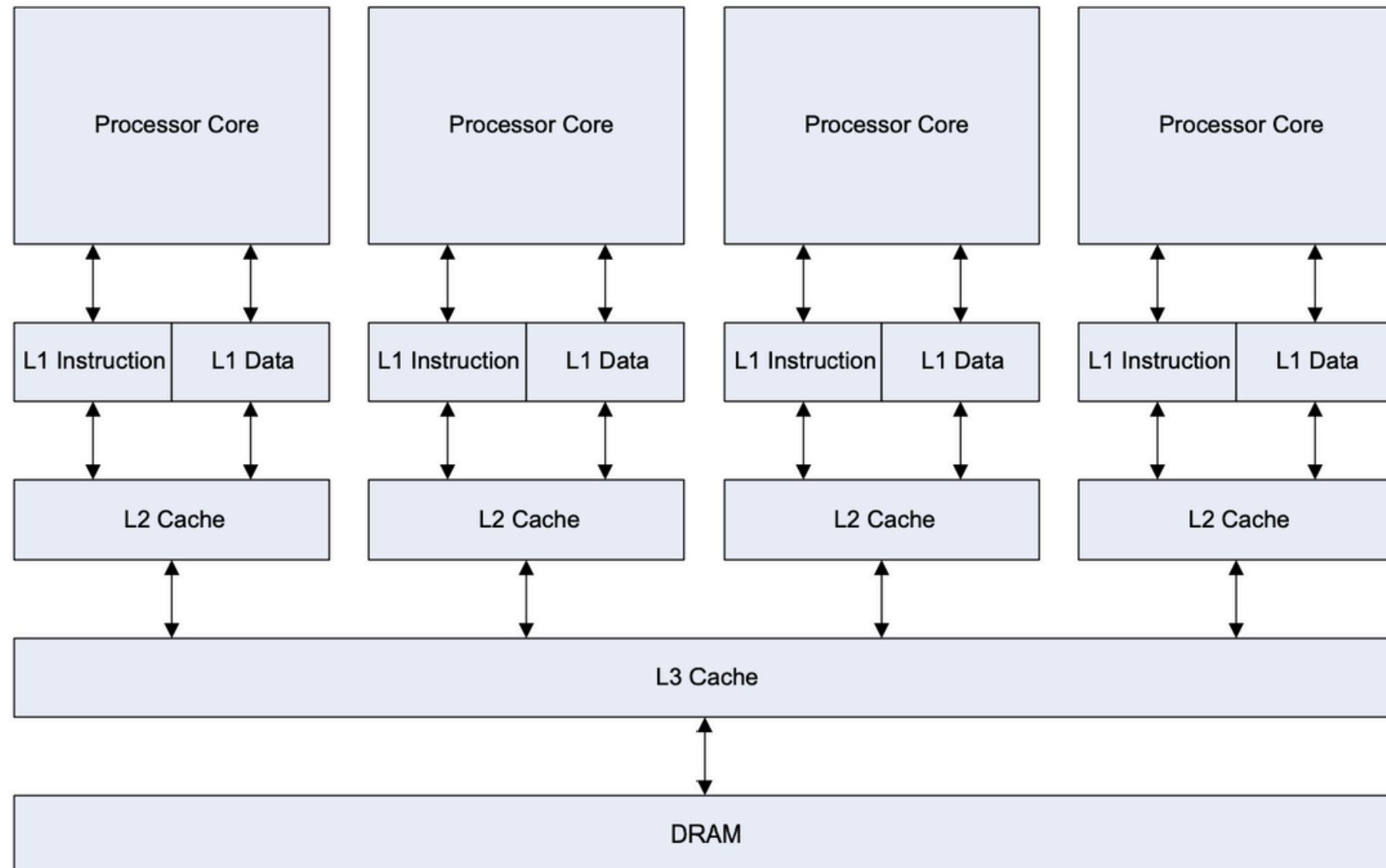


FIGURE 1.1

Typical modern CPU cache organization.

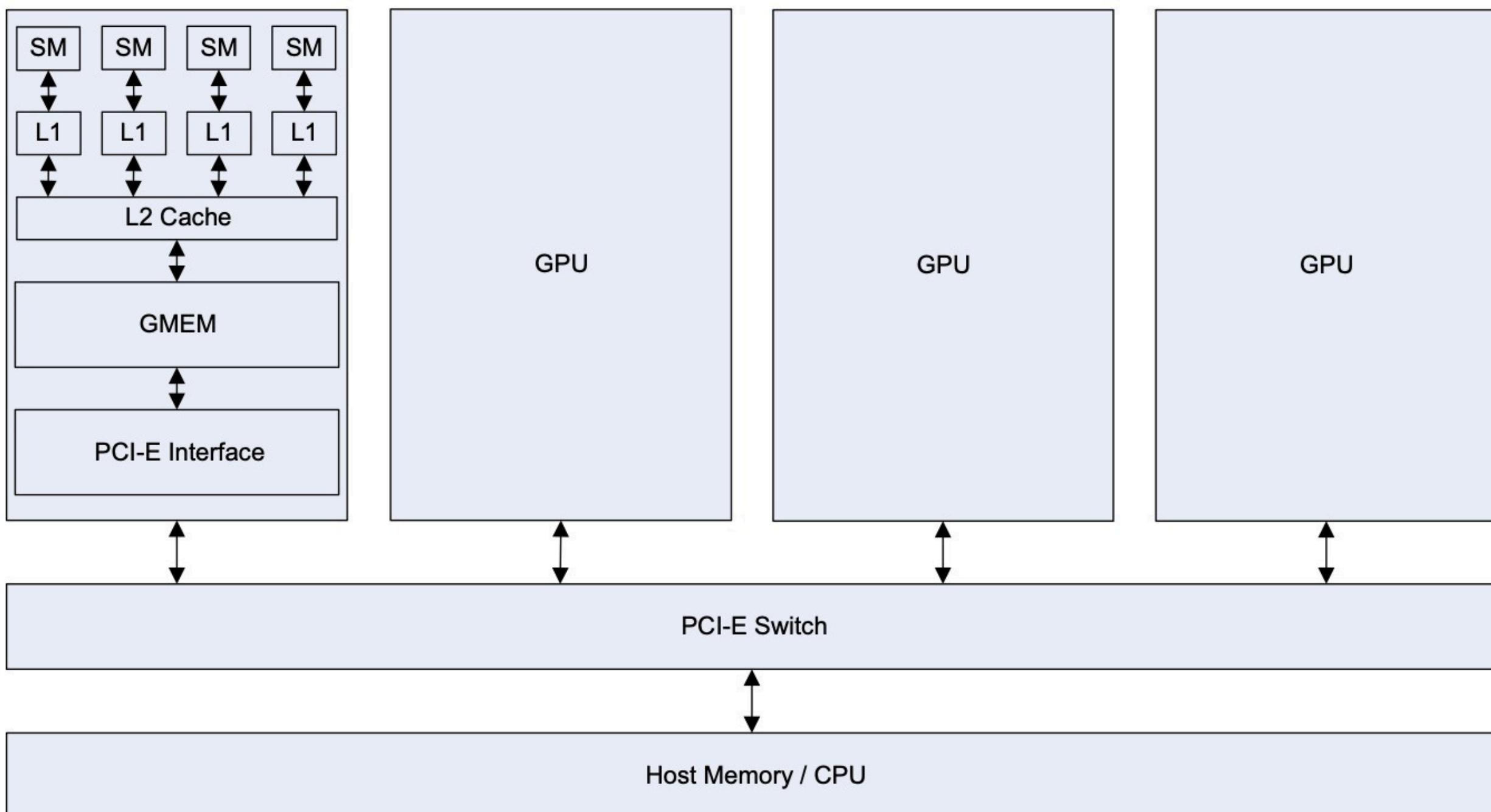


FIGURE 1.7

GPUs compared to a cluster.

Arquiteturas GPU

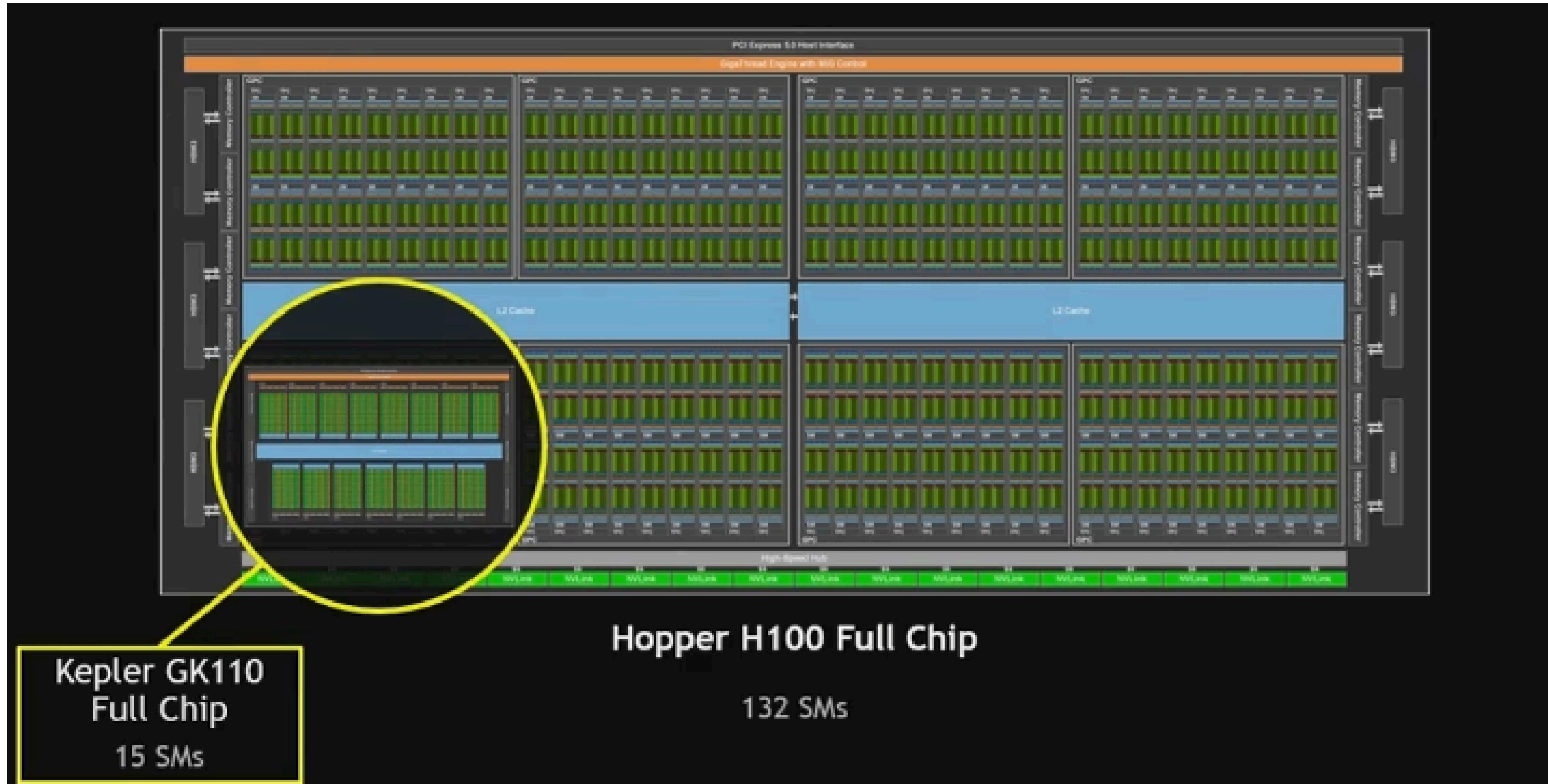
CUDA (Compute Unified Device Architecture)

- Embarrassingly Parallel Operations
- SIMD (Single instruction Multiple Data)
- Integração com Processadores;
- Amplamente adotado em IA e ML;

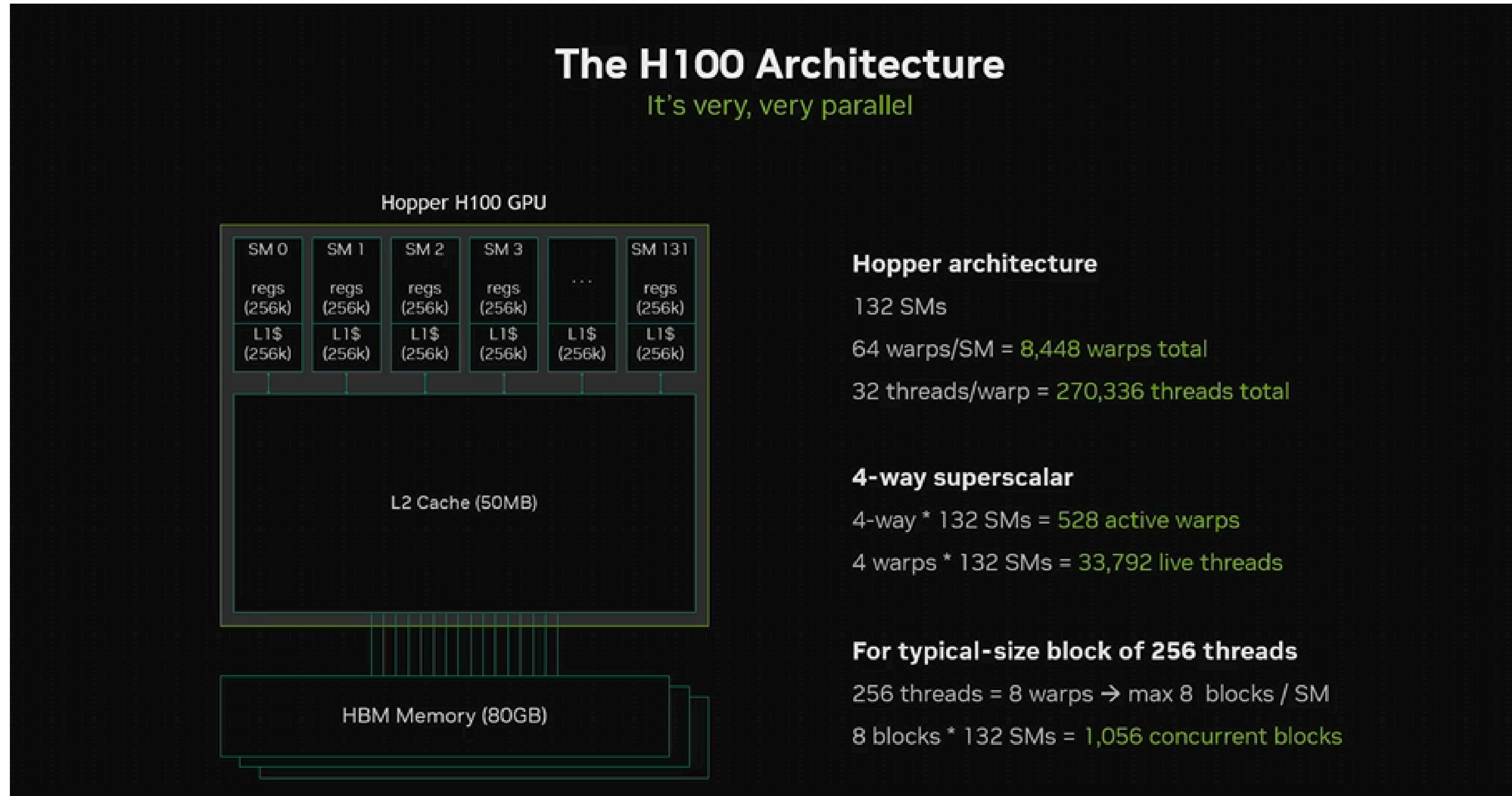
Exemplos:

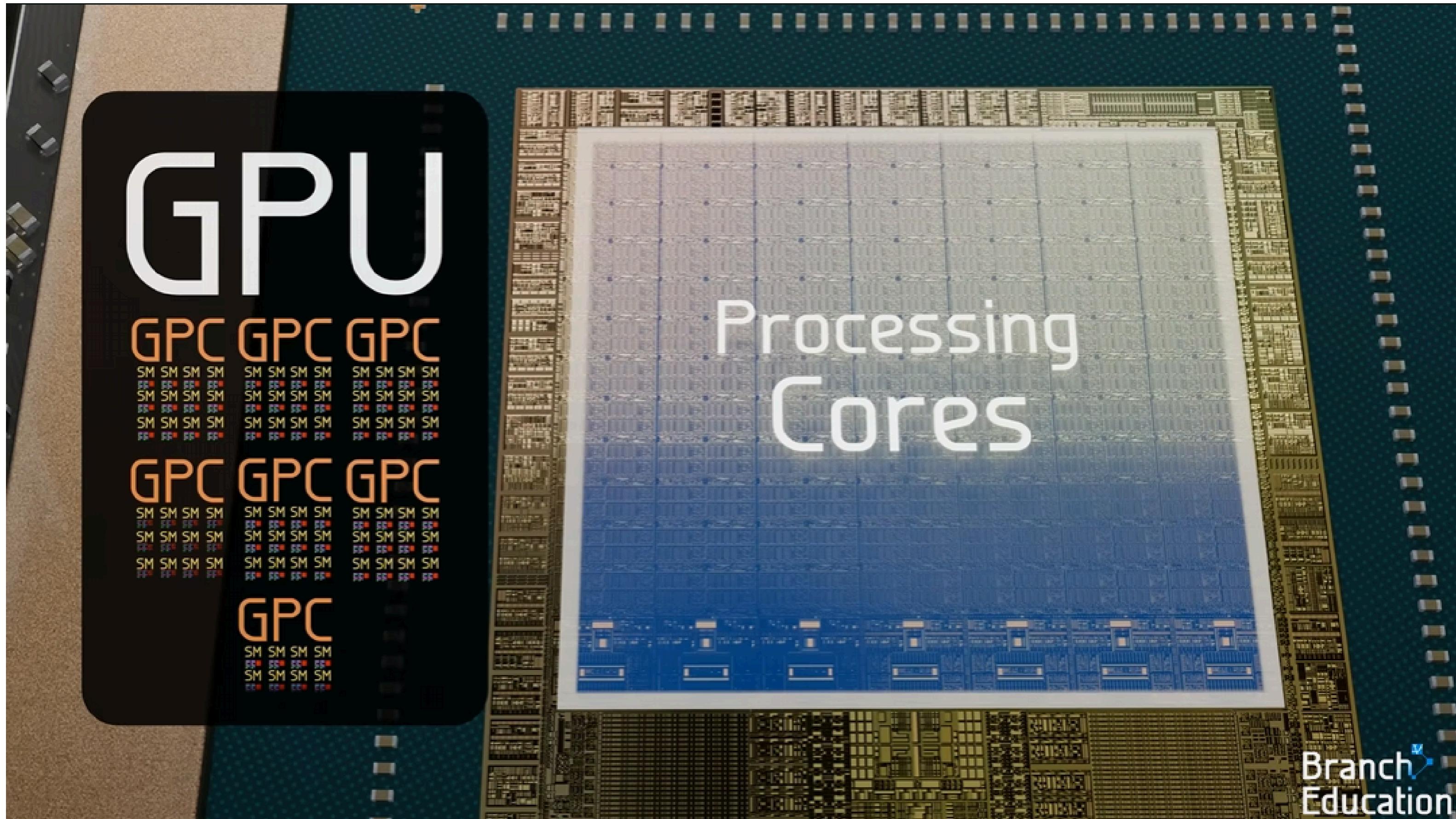
NVIDIA GeForce;

Arquiteturas GPU

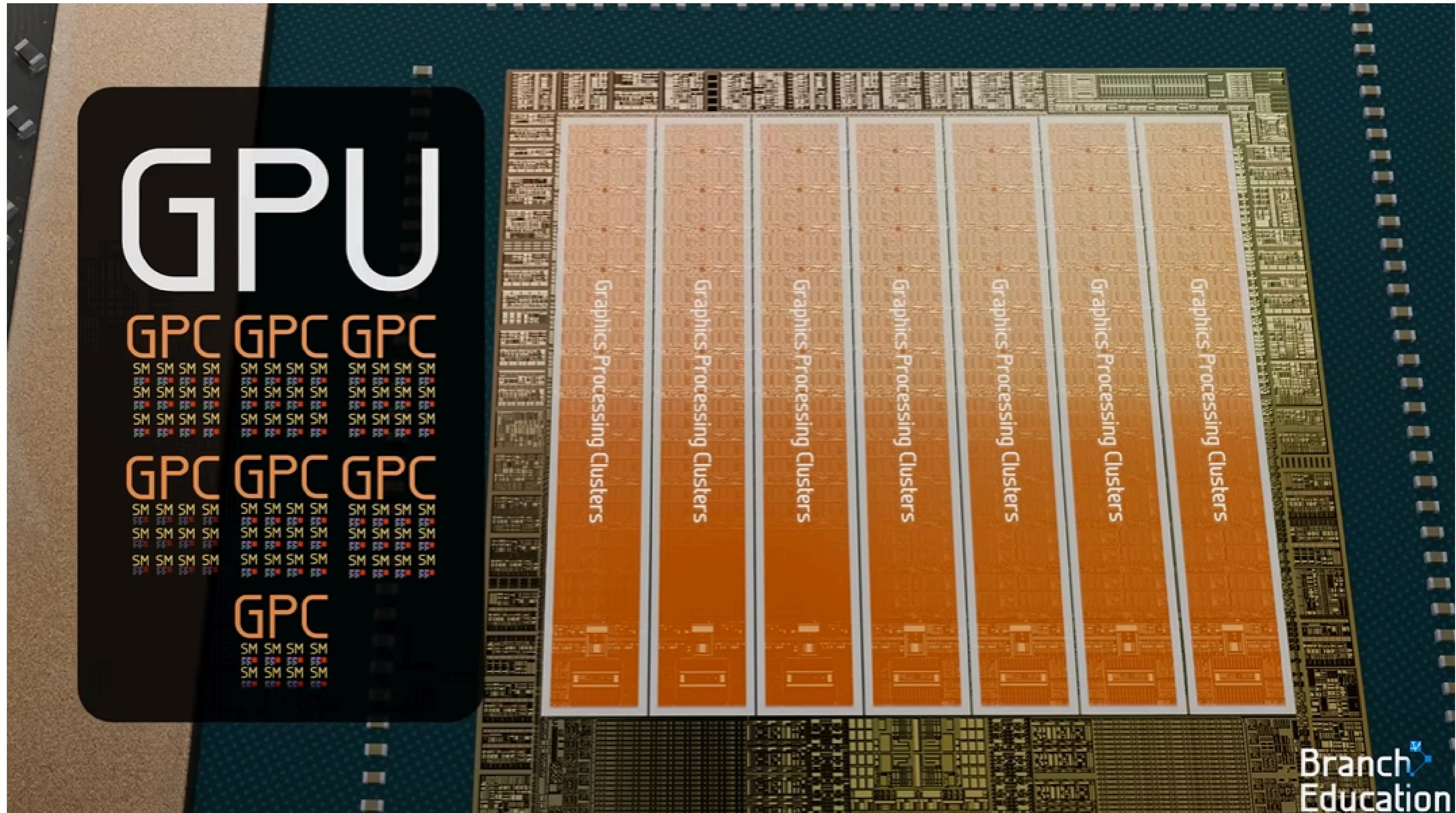


Arquiteturas GPU

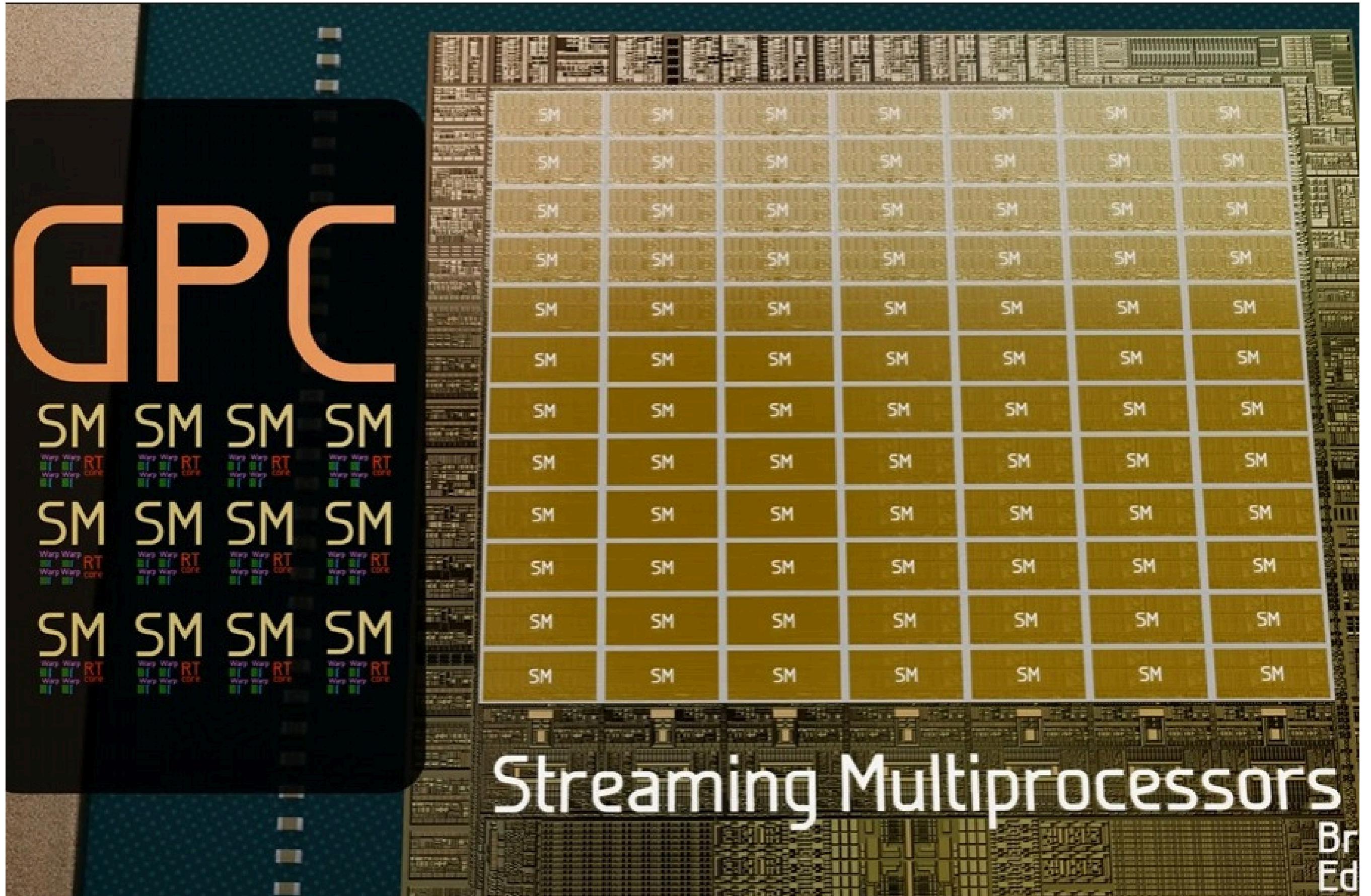




Branch
Education

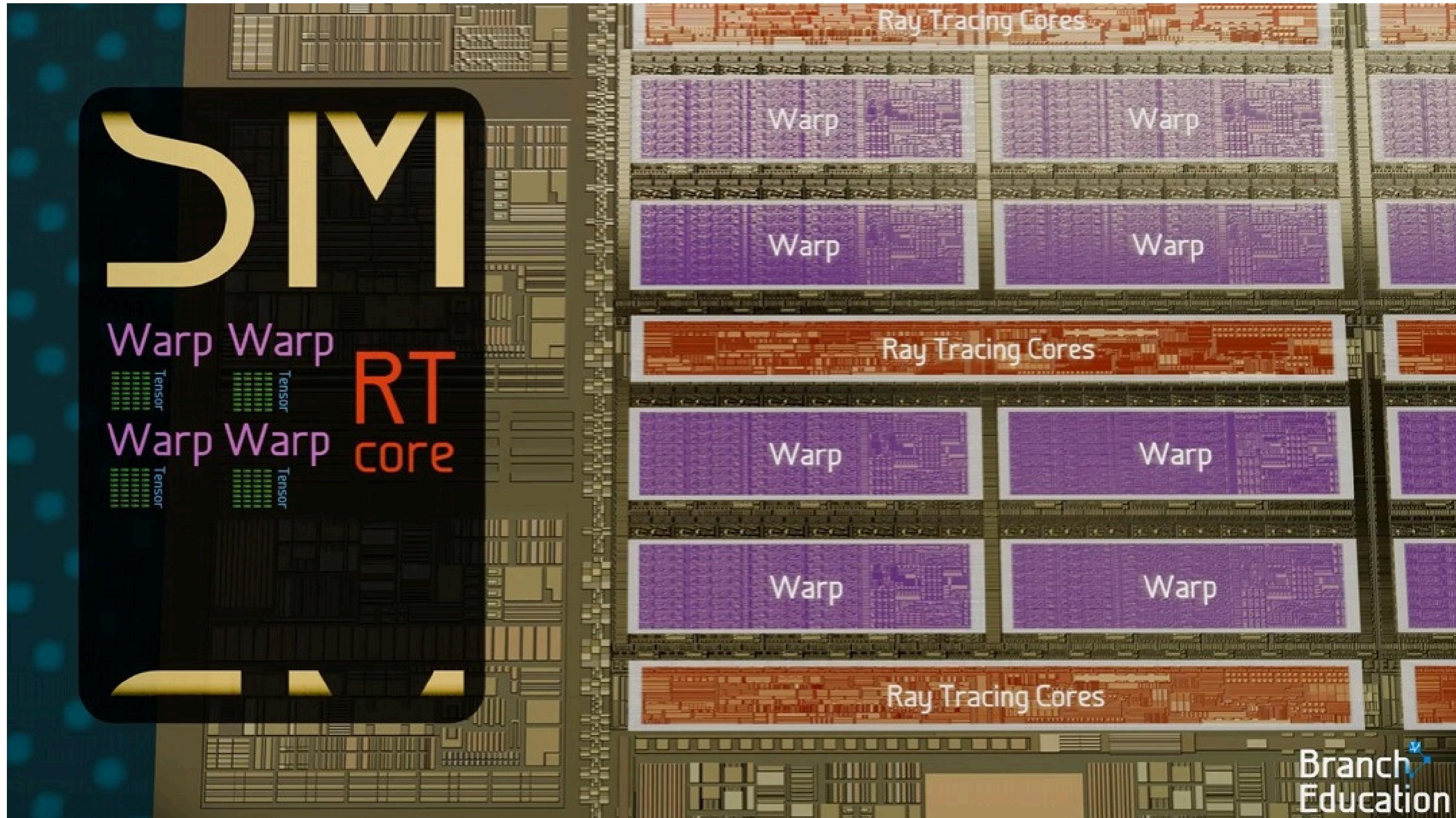


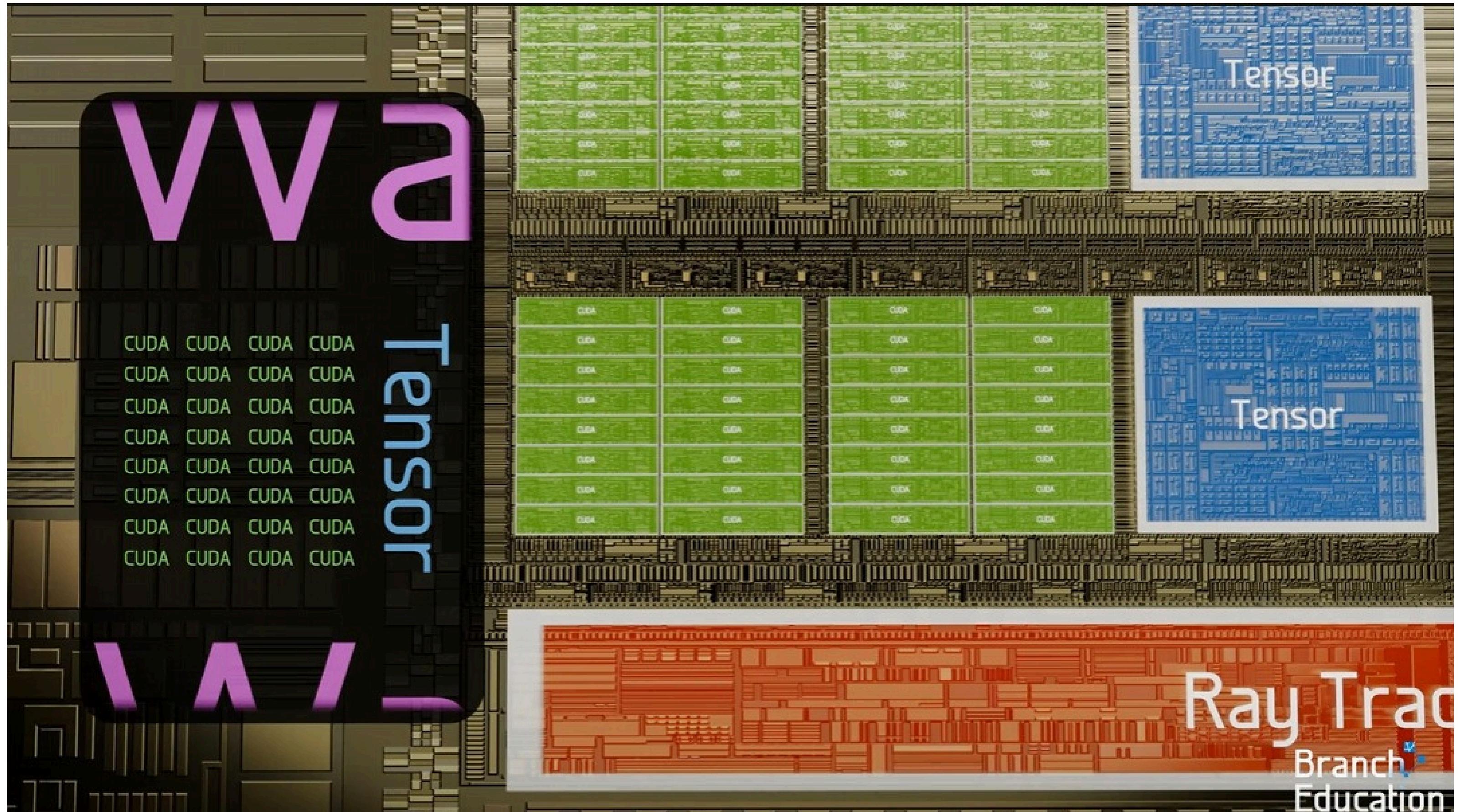
Branch
Education

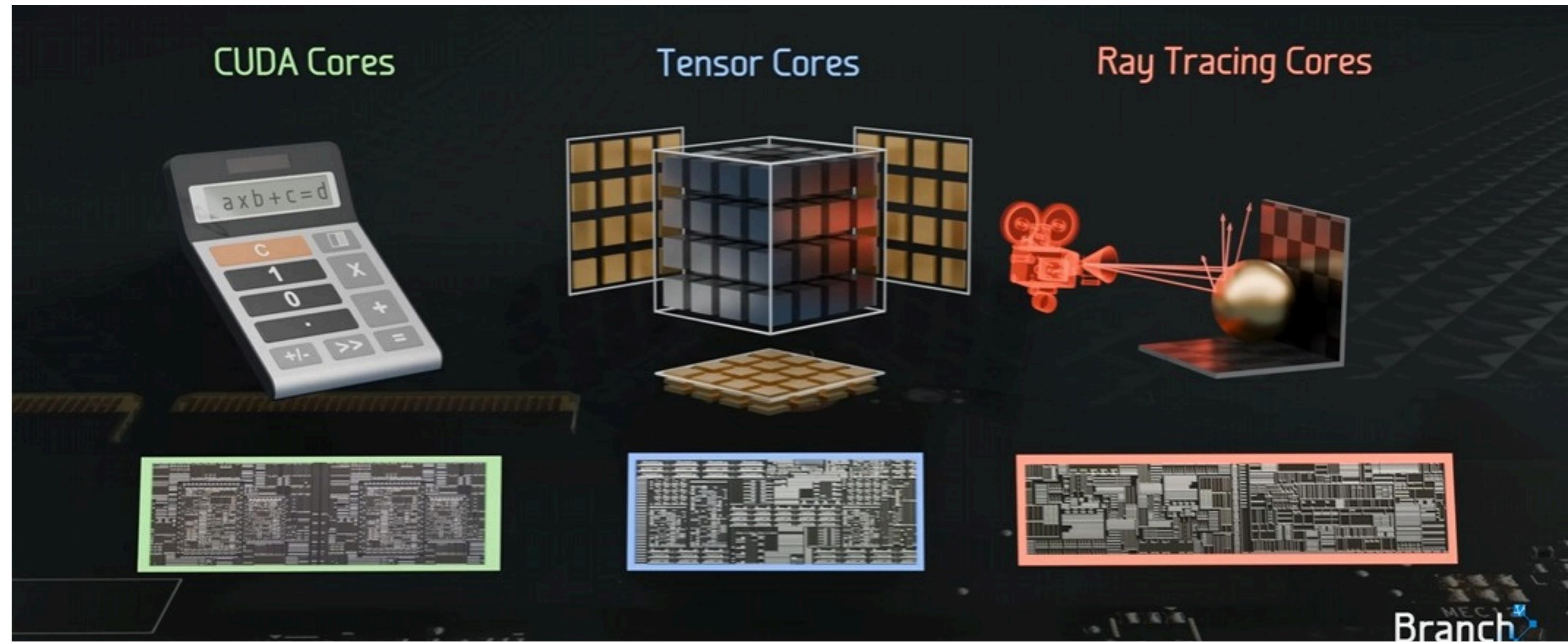


Streaming Multiprocessors

Br
Ed







Cuda cores

10752

$$A \times B + C = D$$

Tensor cores

336

$$[A] \times [B] + [C] = [D]$$

*transformações geométricas
e neural networks

Ray tracing cores

84

algoritmos de
Ray-Tracing

Evolução das GPUs

- **Primeira geração:** Fixas, projetadas apenas para renderização gráfica básica.
- **Segunda geração:** Introdução de pipelines programáveis, permitindo maior flexibilidade no desenvolvimento de shaders.
- **Arquiteturas modernas:**
 - Suporte a computação geral (GPGPU)
 - Ray Tracing
 - Tensor Cores
 - Avanços em resfriamento e eficiência energética, como tecnologias de gerenciamento de consumo dinâmico.

Programação para GPUs

OpenMP

CUDA

- Permite que programadores utilizem GPUs para tarefas de computação geral (GPGPU)
- Memória compartilhada entre threads do mesmo bloco, otimizando desempenho.
- Suporte a estruturas multidimensionais: threadIdx (x, y, z) e blockIdx.

- Usada para programação paralela em CPUs e GPUs
- Ela inclui diretrizes que permitem parallelismo em nível de thread
- Suporte ao processamento SIMD (Single Instruction Multiple Data).

CUDA

- **Thread:** Unidade básica de execução.
- **Bloco:** Um grupo de threads que executa um kernel.
- **Grade (Grid):** Coleção de blocos organizados em uma ou mais dimensões.

VARIÁVEIS CUDA PARA IDENTIFICAÇÃO DE THREADS

- **threadIdx.x, threadIdx.y, threadIdx.z:** Identificadores da thread dentro de um bloco.
- **blockIdx.x, blockIdx.y, blockIdx.z:** Identificadores do bloco dentro de uma grade.
- **blockDim.x, blockDim.y, blockDim.z:** Dimensões do bloco (número de threads por dimensão).
- **gridDim.x, gridDim.y, gridDim.z:** Dimensões da grade (número de blocos por dimensão).

3 Dimensões no CUDA

```
__global__ void volumeProcess(float *input, float *output, int width, int height, int depth) {  
    int x = threadIdx.x + blockIdx.x * blockDim.x;  
    int y = threadIdx.y + blockIdx.y * blockDim.y;  
    int z = threadIdx.z + blockIdx.z * blockDim.z;  
  
    if (x < width && y < height && z < depth) {  
        int idx = z * width * height + y * width + x;  
        output[idx] = input[idx] * 2.0f;  
    }  
}
```

Esta função realiza um **processamento volumétrico** em 3D, onde cada thread é responsável por calcular um ponto no espaço tridimensional (**x, y, z**), utilizando identificadores de **threads e blocos para acessar a posição correspondente** no volume de entrada e aplicar uma operação ao volume de saída.

```
dim3 blockSize(8, 8, 8)  
dim3 gridSize((width + 7) / 8, (height + 7) / 8, (depth + 7) / 8)  
volumeProcess<<<gridSize, blockSize>>>(input, output, width, height, depth)
```

OpenMP

Diretivas SIMD no OpenMP

OpenMP oferece a diretiva **#pragma omp simd** para paralelizar loops, utilizando instruções vetoriais otimizadas. Essas diretivas podem ser usadas para explorar paralelismo em GPUs.

Uso com Vetores: #include <omp.h>

```
void vectorAdd(float *a, float *b, float *c, int N) {  
    #pragma omp simd  
    for (int i = 0; i < N; i++) {  
        c[i] = a[i] + b[i];  
    }  
}
```

A diretiva **#pragma omp simd** informa ao compilador que as iterações do loop podem ser executadas em paralelo.

REFERÊNCIAS

- Artigo: Introdução à arquitetura de GPUs. Disponível em:
whtwtpws.i.c.unicamp.br/~ducatte/mo401/1s2009/T2/045116-t2.pdf
- Slides: Introdução à programação paralela em GPU. Disponível em:
lhiettfp.ifs.ufrgs.br/pub/Cursos/Cuda/aula01.pdf
- Livro: CUDA PROGRAMMING A DEVELOPER'S GUIDE TO PARALLEL COMPUTING WITH GPUs - Shane Cook
- Video: How do Graphics Cards Work? Exploring GPU Architecture
<https://www.youtube.com/watch?v=h9Z4oGN89MU>
- Artigo: CPU vs GPU: Why GPUs are More Suited for Deep Learning? Disponível em: <https://www.analyticsvidhya.com/blog/2023/03/cpu-vs-gpu/>