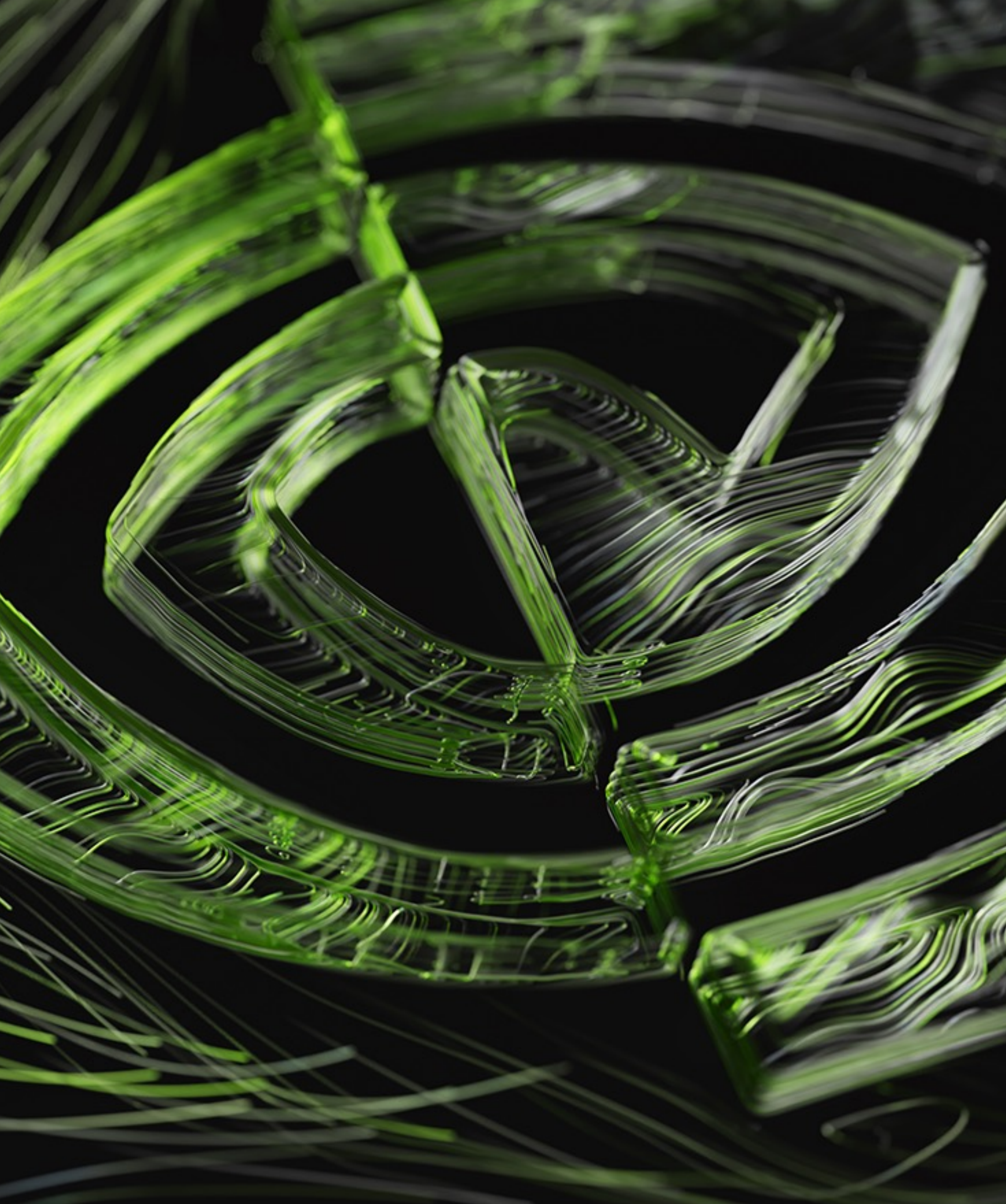


GPU コンピューティング入門 & ハンズオン



# Agenda

- Introduction to GPU Computing

---
- GPU Computing

---
- CUDA

---
- ハンズオン

---
- Lab 1

---
- Lab 2

---

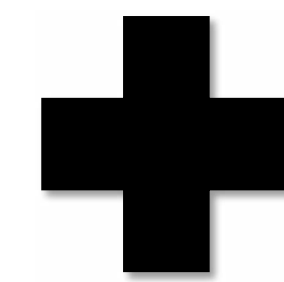
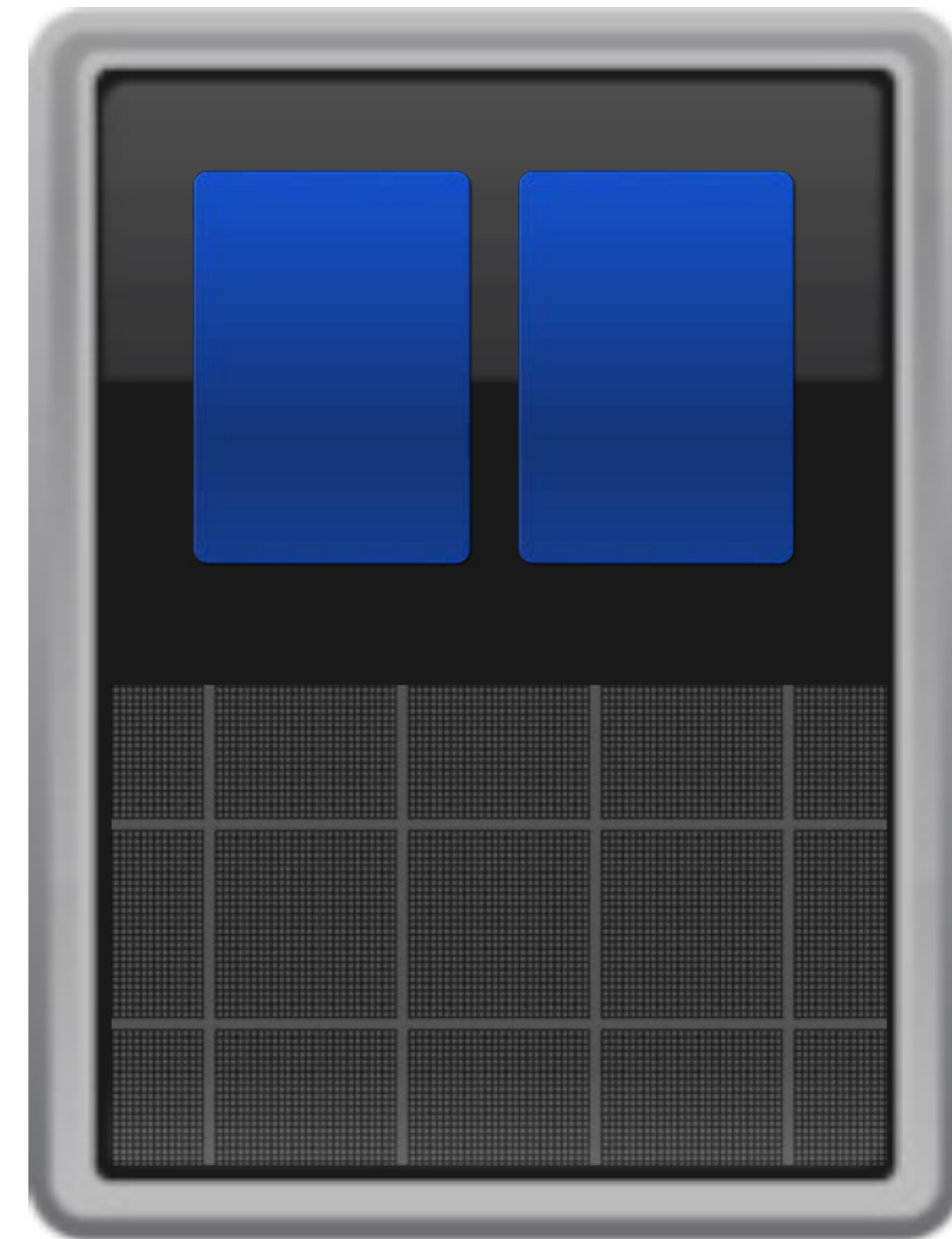
The background of the slide is a dark, abstract visualization of data or computation. It features numerous thin, glowing green lines that curve and flow across the frame, creating a sense of dynamic movement and complexity. The lines are more densely packed in some areas, particularly towards the right side, where they form a more intricate, almost crystalline structure. The overall effect is reminiscent of a high-speed data stream or a complex network of connections.

# GPU Computing

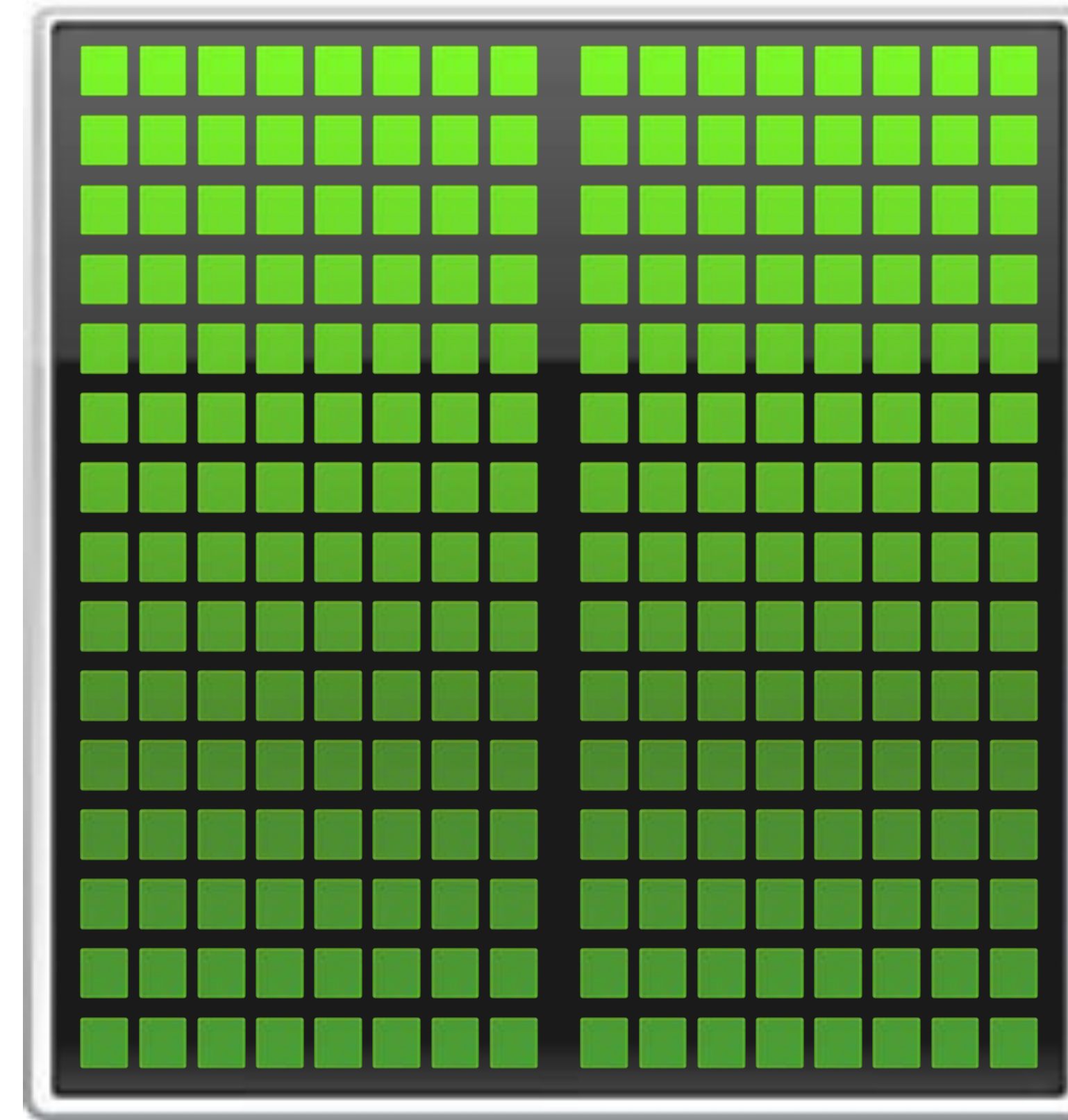
# GPU コンピューティング

Low latency + High throughput

CPU

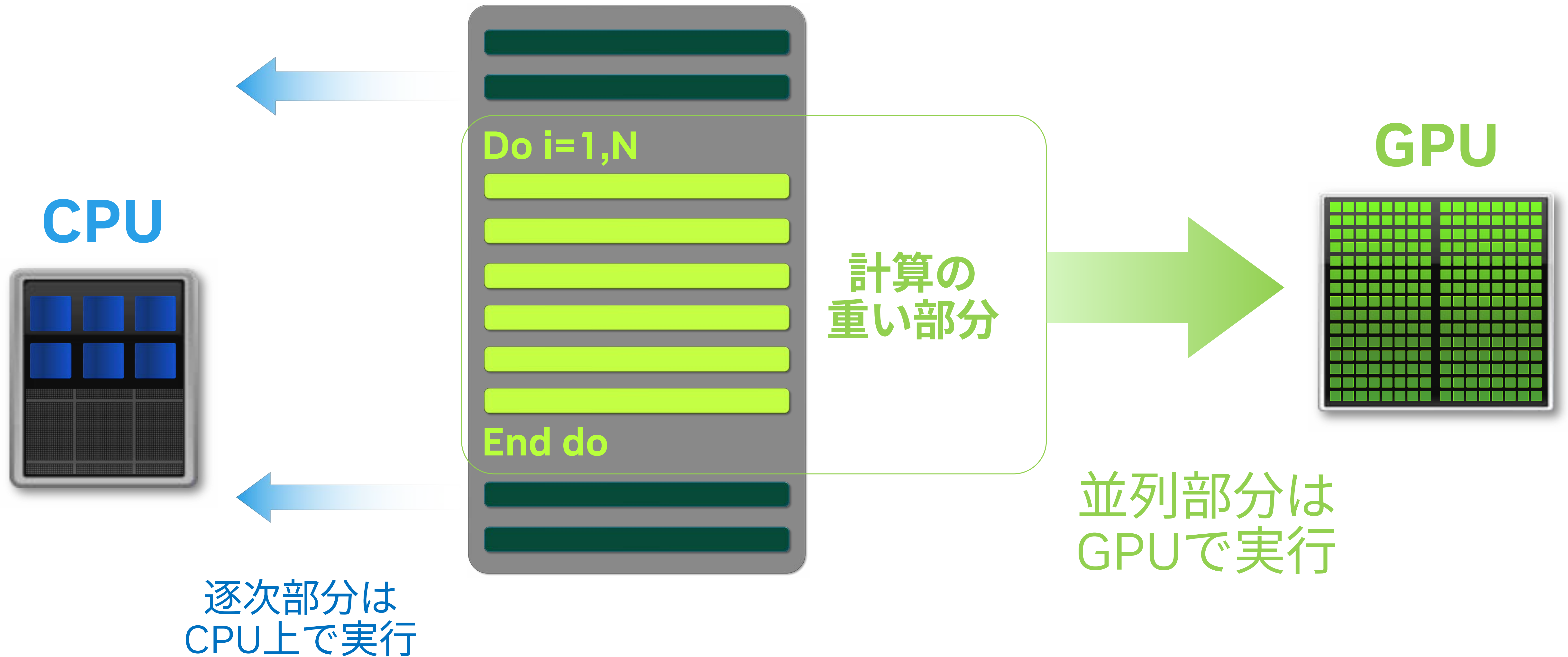


GPU



# アプリケーション実行

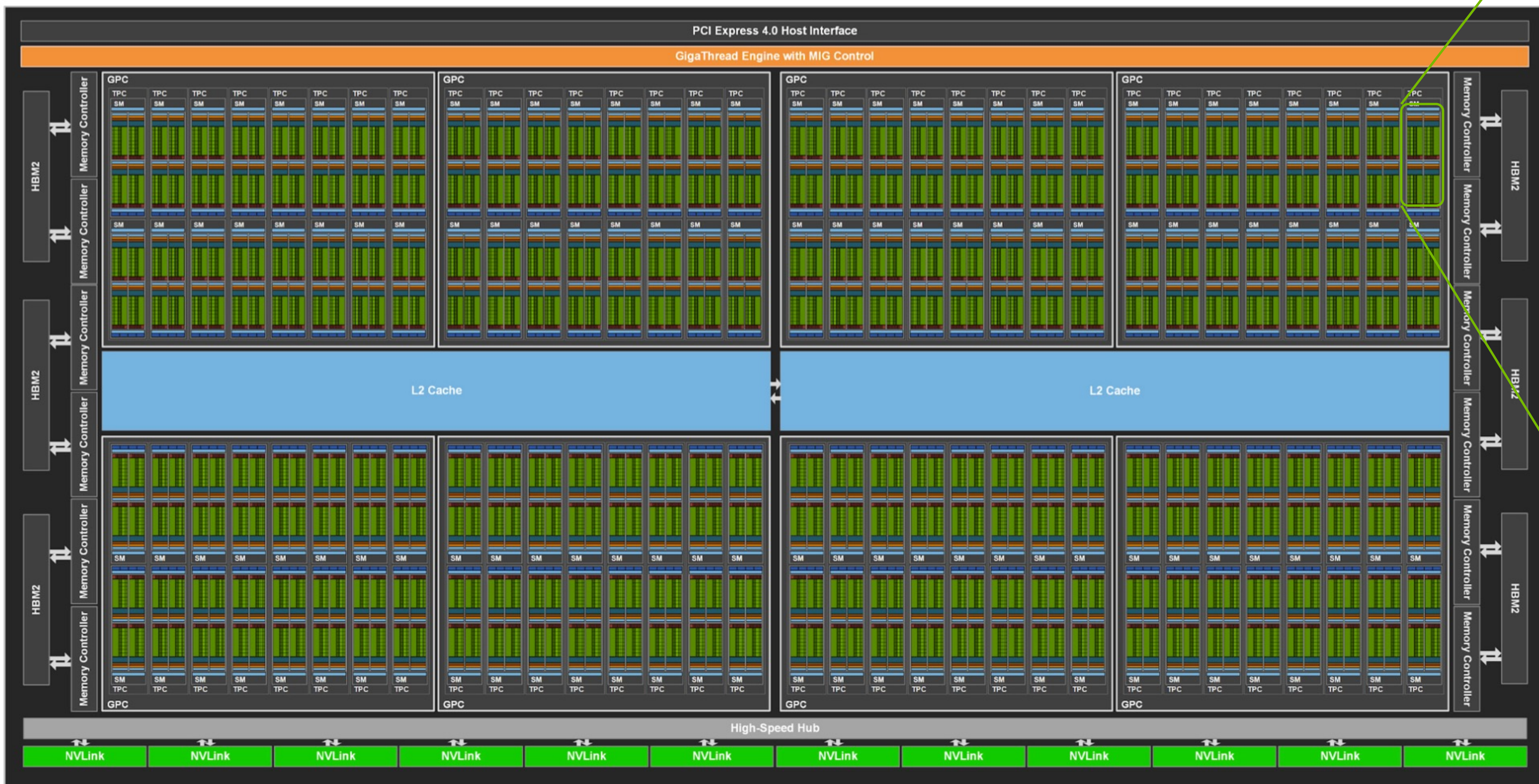
アプリケーション・コード



# GPU の構造

## NVIDIA A100

大量の CUDA コア  
並列性の抽出が鍵



64 CUDA core / SM

108 Streaming Multiprocessor (SM) / chip

# GPU Applications

<https://www.nvidia.com/en-us/gpu-accelerated-applications/>



Products Solutions Industries For You

Shop Drivers Support



## Accelerated Apps Catalog

Use the search bar below to find out if your favorite app is one of the hundreds being accelerated by NVIDIA GPUs. Can't find a specific app? Come back and search again as more are added every month.

Filter Categories

Filter Industries

Sort A-Z

AI Accelerated

Share

Search apps



Prev

Show

15

per page

Page

1

Next

Shenzhen Rayvision Technology Co Ltd

### 3D CAT.live

Real-time rendering cloud service for 3D applications. The massive GPU computing power in the cloud is used to process heavy image rendering calculations and stream output to the terminal device synchronously, thereby realizing light weight

3D Slicer

### 3D Slicer

3D Slicer is an open-source software platform for medical image informatics, image processing, and three-dimensional visualization. Slicer brings free, powerful cross-platform processing tools to physicians, researchers, and the general public.

QT Imaging Inc

### 3D Ultrasound Tomography/Volography

3D ultrasound breast, pediatric and whole body tomography.  
➤ Quantitative high resolution 3D US images of tissue properties of breast with GPU



# GPU Applications

<https://www.nvidia.com/en-us/gpu-accelerated-applications/>

The screenshot shows the NVIDIA GPU Applications website. At the top, there is a navigation bar with the NVIDIA logo and links for Products, Solutions, Industries, and For You. On the right, there are links for Shop, Drivers, and Support, along with search and user icons. Below the navigation bar, there is a filter section with a search box labeled "Search apps". The filter categories are listed in a grid, with "All Categories" selected and highlighted in green, showing a count of [1090]. A green banner with white text is overlaid on the filter list, stating "1000 以上のアプリケーションが GPU に対応" (Over 1000 applications are GPU compatible). At the bottom of the filter section, there are two buttons: "Clear Filters" and "Apply Filters".

Category	Count
<input checked="" type="checkbox"/> All Categories	[1090]
<input type="checkbox"/> 3D Rendering	[114]
<input type="checkbox"/> Animation and Modeling	[47]
<input type="checkbox"/> Astronomy & Astrophysics	
<input type="checkbox"/> Big Data & Data Mining	[47]
<input type="checkbox"/> Bioinformatics & Genomics	[59]
<input type="checkbox"/> Broadcast Graphics & Infrastructure	[41]
<input type="checkbox"/> Business Intelligence & Analytics	
<input type="checkbox"/> Business Planning and Optimization	
<input type="checkbox"/> Computational Fluid Dynamics (CFD)	
<input type="checkbox"/> Computational Photography	
<input type="checkbox"/> Computer Vision & Machine Vision	
<input type="checkbox"/> Conversational AI	[30]
<input type="checkbox"/> Customer Engagement	[11]
<input type="checkbox"/> Data Technology and Analytics	[46]
<input type="checkbox"/> Databases	
<input type="checkbox"/> Design & Visualization	[111]
<input type="checkbox"/> Development Tools & Libraries	[42]
<input type="checkbox"/> Diagnostic Imaging	[30]
<input type="checkbox"/> Drug Discovery	
<input type="checkbox"/> Education & Training	[14]
<input type="checkbox"/> Electronic Design Automation (EDA)	[34]
<input type="checkbox"/> Energy Exploration & Generation	[11]
<input type="checkbox"/> Finance & Economics	
<input type="checkbox"/> Game	[11]
<input type="checkbox"/> Game Development	[7]
<input type="checkbox"/> Genomic Primary Analysis	[1]
<input type="checkbox"/> Genomic Secondary Analysis	
<input type="checkbox"/> Geoscience	[22]
<input type="checkbox"/> Goods Movements and Logistics	[3]
<input type="checkbox"/> Graphic Design	[6]
<input type="checkbox"/> Healthcare IT	
<input type="checkbox"/> Industrial Inspection	[13]
<input type="checkbox"/> Internet of Things (IoT)	[11]
<input type="checkbox"/> M&E: Color management	[24]
<input type="checkbox"/> M&E: Compositing and Finishing	
<input type="checkbox"/> M&E: On-set, review and stereo	[14]
<input type="checkbox"/> Machine Learning & AI	[126]
<input type="checkbox"/> Material Science	[30]
<input type="checkbox"/> Medical Analytics	
<input type="checkbox"/> Medical Imaging	[26]
<input type="checkbox"/> Microscopy	[33]
<input type="checkbox"/> Military simulation	[6]
<input type="checkbox"/> Molecular Dynamics	
<input type="checkbox"/> Molecular visualization and Docking	[15]
<input type="checkbox"/> NLP	[25]
<input type="checkbox"/> Neuroscience	[4]
<input type="checkbox"/> Numerical Analytics	
<input type="checkbox"/> Pharmacometrics	[2]
<input type="checkbox"/> Photography/image processing	[43]
<input type="checkbox"/> Physics	[36]
<input type="checkbox"/> Predictive Maintenance	
<input type="checkbox"/> Programming Languages & Compilers	[7]
<input type="checkbox"/> Quantum Chemistry	[34]
<input type="checkbox"/> Robotics & autonomous machines	[15]
<input type="checkbox"/> Scientific Visualization	

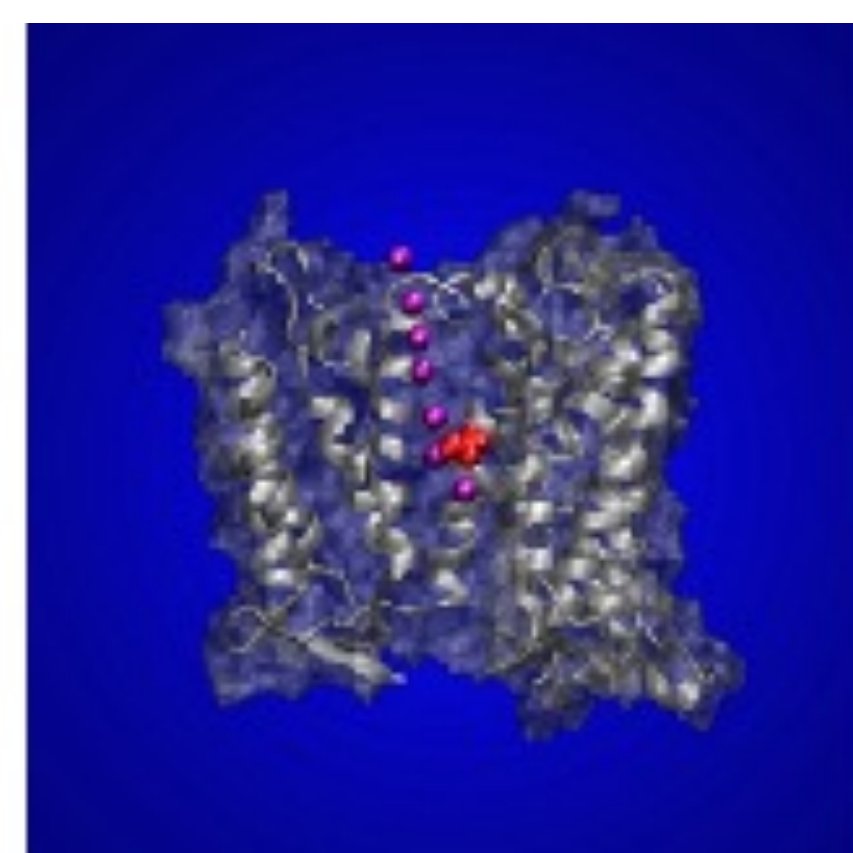
> Automation of Manual Process of Financial Spreading & Analysis of Financial Documents



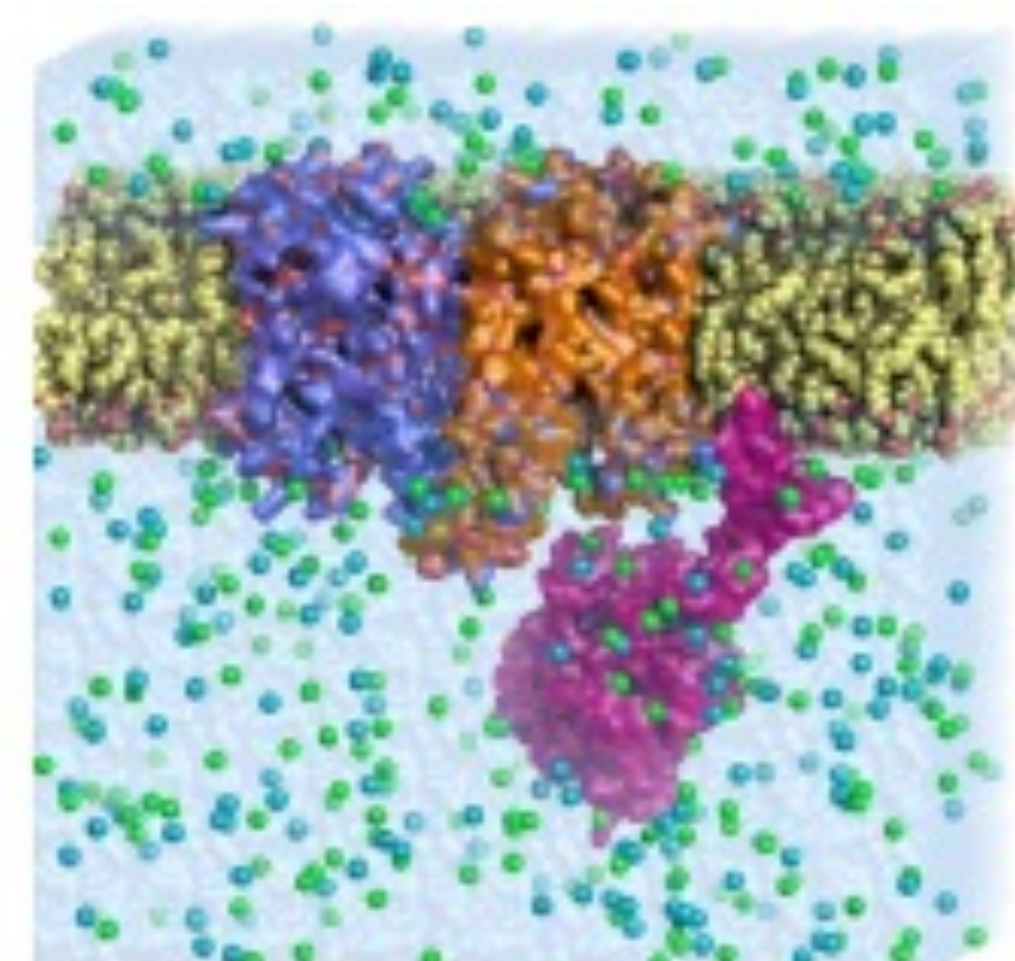
# Leading Applications



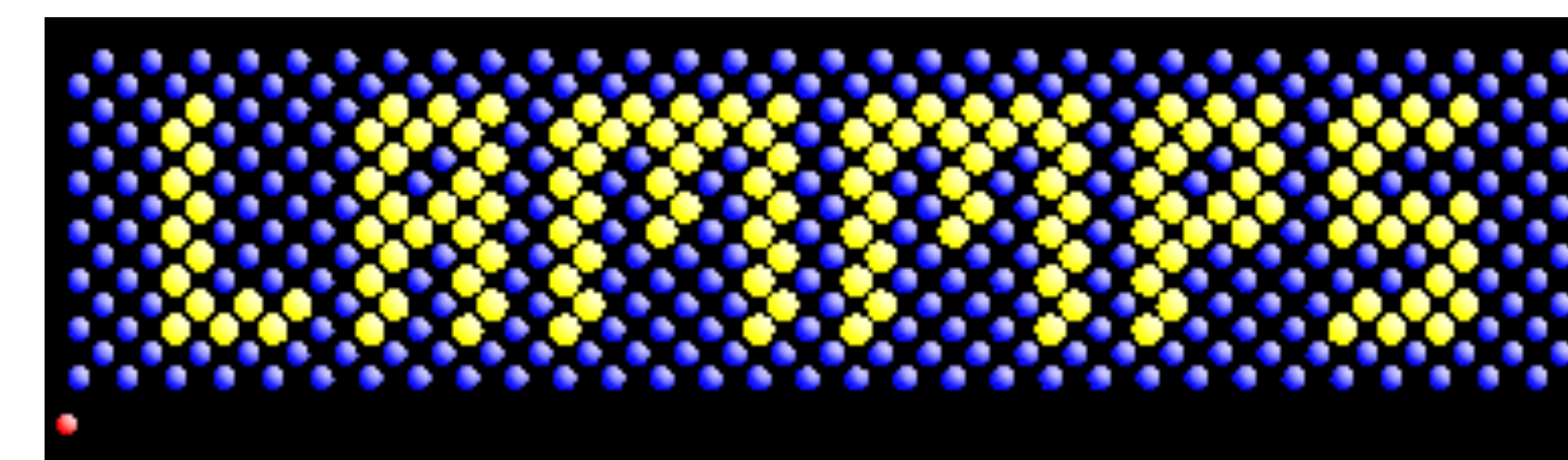
*Gaussian*



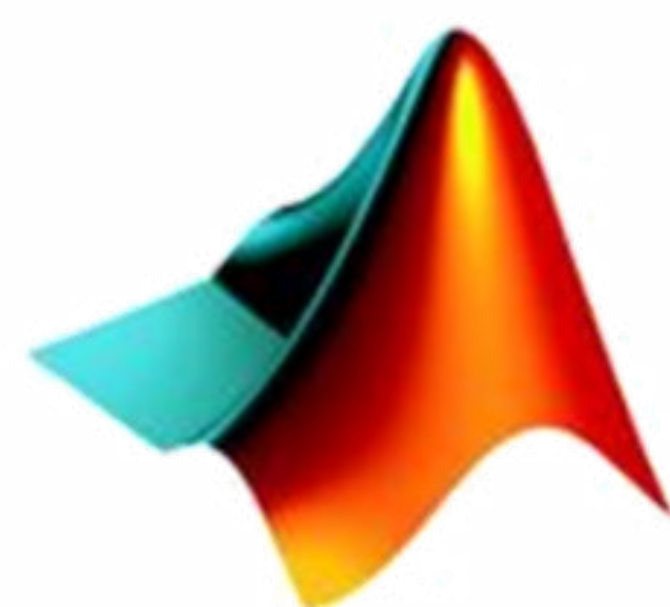
Desmond



Amber



**HOOMD-blue**



MATLAB®

# アプリを GPU 対応する方法

Application

Library

GPU 対応ライブラリに  
チェンジ  
簡単に開始

OpenACC

既存コードに  
ディレクティブを挿入  
簡単に加速

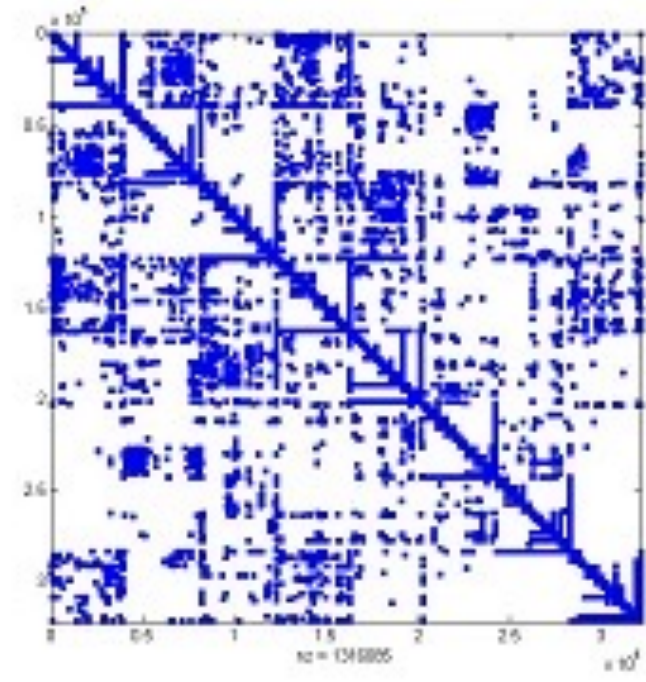
CUDA

主要処理を CUDA で記述  
高い自由度

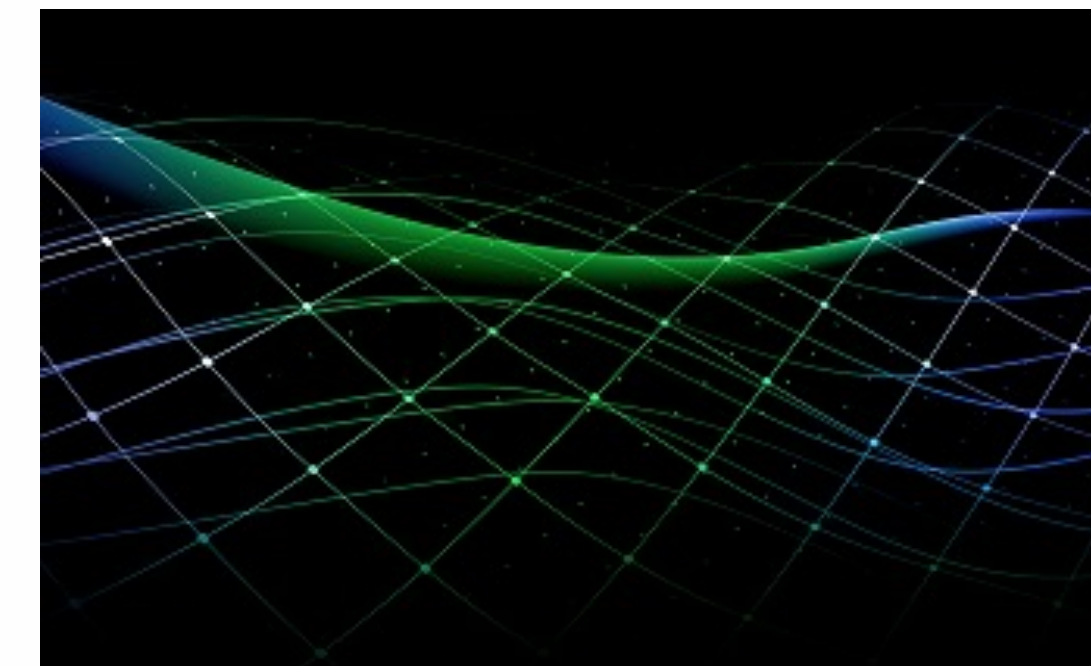
# NVIDIA Libraries



cuBLAS



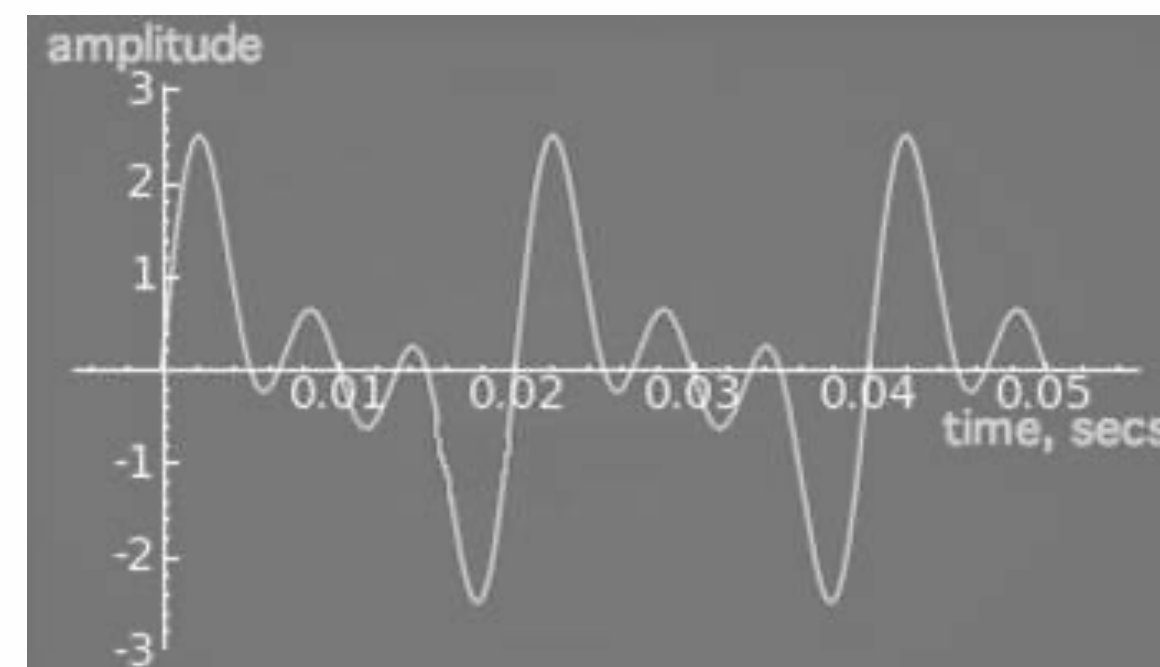
cuSPARSE



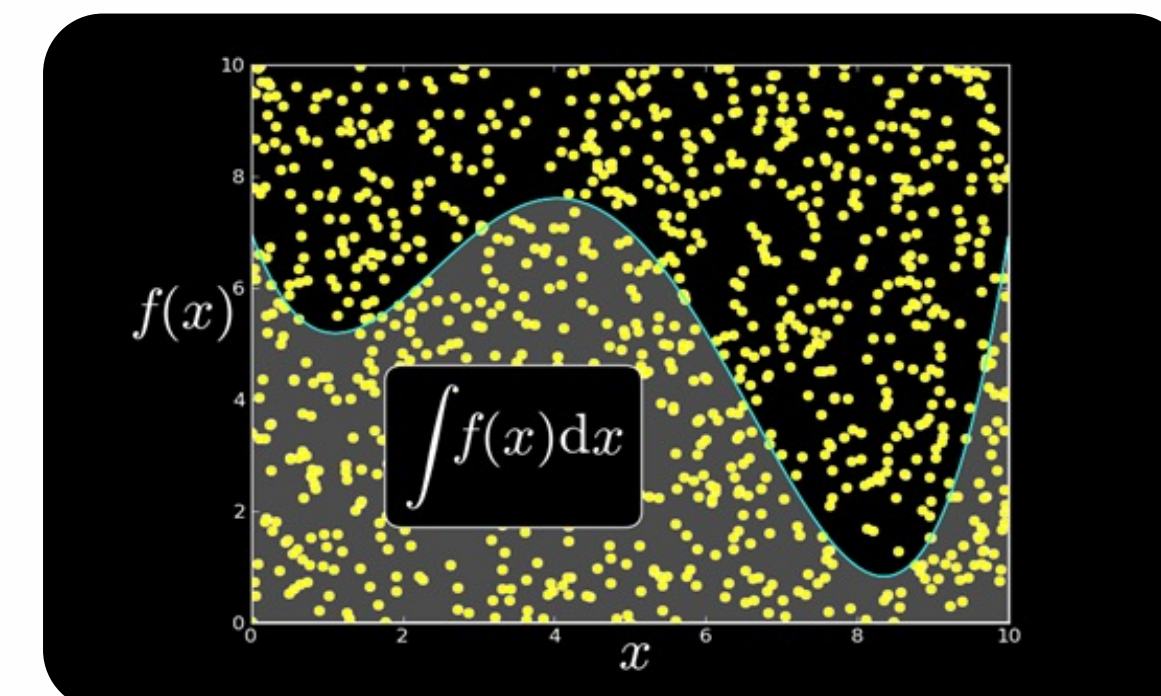
cuSOLVER



AmgX



cuFFT



cuRAND



nvGRAPH



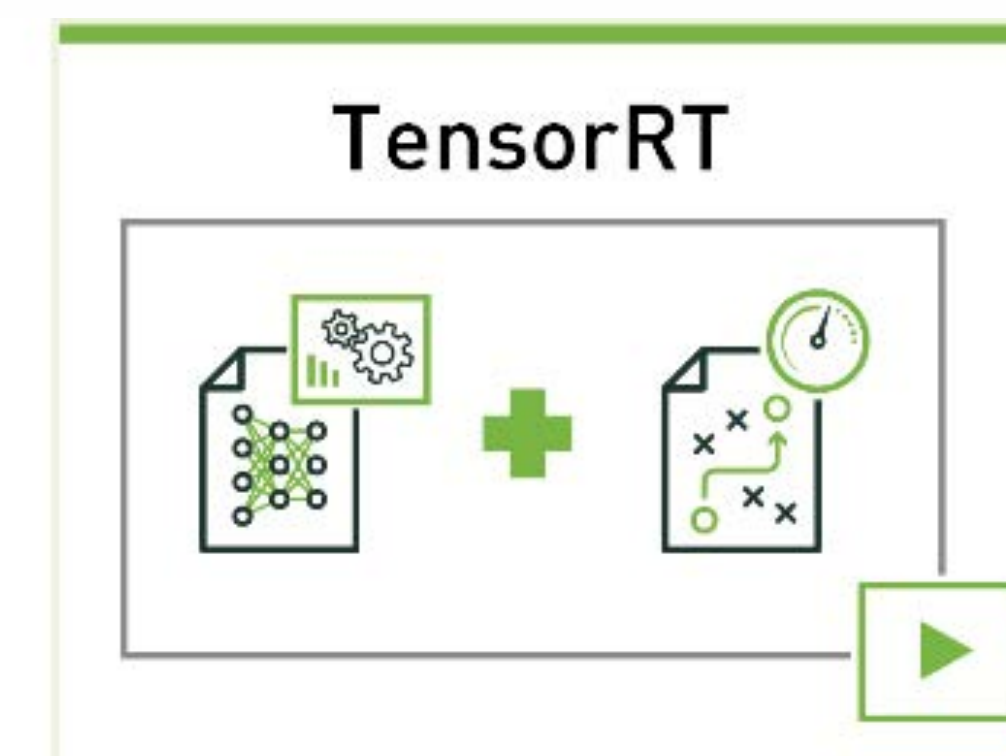
Thrust



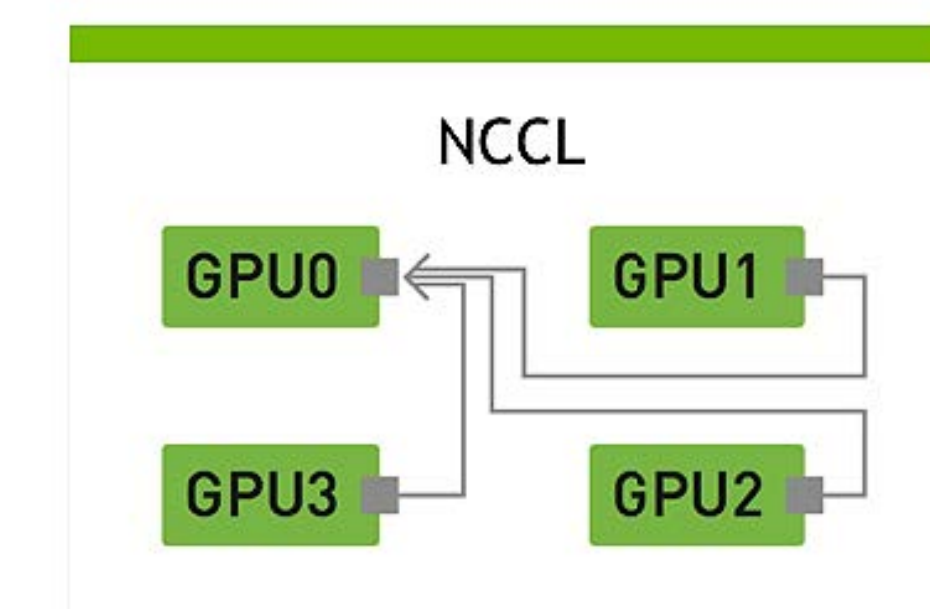
Performance Primitives



cuDNN



TensorRT



NCCL

# Partner Libraries



Computer Vision



Audio and Video



Matrix, Signal and Image



Linear Algebra



Math, Signal and Image



Graph



Sparse direct solvers



Linear Algebra



Linear Algebra



Computational  
Geometry



Sparse Iterative  
Methods



Real-time visual  
simulation

# アプリを GPU 対応する方法

Application

Library

GPU 対応ライブラリに  
チェンジ  
簡単に開始

OpenACC

既存コードに  
ディレクティブを挿入  
簡単に加速

CUDA

主要処理を CUDA で記述  
高い自由度

# SAXPY ( $Y = A * X + Y$ )

## OpenMP

```
void saxpy(int n,  
          float a,  
          float *x,  
          float *restrict y)  
{  
#pragma omp parallel for  
  for (int i = 0; i < n; ++i)  
    y[i] += a*x[i];  
}  
  
...  
saxpy(N, 3.0, x, y);  
...
```

## OpenACC

```
void saxpy(int n,  
          float a,  
          float *x,  
          float *restrict y)  
{  
#pragma acc parallel copy(y[:n]) copyin(x[:n])  
  for (int i = 0; i < n; ++i)  
    y[i] += a*x[i];  
}  
  
...  
saxpy(N, 3.0, x, y);  
...
```

# アプリを GPU 対応する方法

Application

Library

GPU 対応ライブラリに  
チェンジ  
簡単に開始

OpenACC

既存コードに  
ディレクティブを挿入  
簡単に加速

CUDA

主要処理を CUDA で記述  
高い自由度

# SAXPY ( $Y = A * X + Y$ )

## CPU

```
void saxpy(int n, float a,
          float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] += a*x[i];
}
```

```
...
saxpy(N, 3.0, x, y);
...
```

## CUDA

```
__global__ void saxpy(int n, float a,
                    float *x, float *y)
{
    int i = threadIdx.x + blockDim.x * blockIdx;
    if (i < n)
        y[i] += a*x[i];
}

...

cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, size, cudaMemcpyHostToDevice);
saxpy<<< N/128, 128 >>>(N, 3.0, d_x, d_y);
cudaDeviceSynchronize();
cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);

...
```



# アプリを GPU 対応する方法

CuPy

Library

GPU 対応ライブラリに  
チェンジ  
簡単に開始

OpenACC

既存コードに  
ディレクティブを挿入  
簡単に加速

CUDA

主要処理を CUDA で記述  
高い自由度

# SAXPY ( $Y = A * X + Y$ )

## CPU

```
import numpy as np

def saxpy(a, x, y):
    return a * x + y

...

x = np.arange(N, dtype=...)
y = np.arange(N, dtype=...)

y = saxpy(a, x, y)

...
```

## CuPy

```
import numpy as np
import cupy as cp

def saxpy(a, x, y):
    return a * x + y

...

x = np.arange(N, dtype=...)
y = np.arange(N, dtype=...)

d_x = cp.asarray(x)
d_y = cp.asarray(y)
d_y = saxpy(a, d_x, d_y)
y = cp.asnumpy(d_y)

...
```

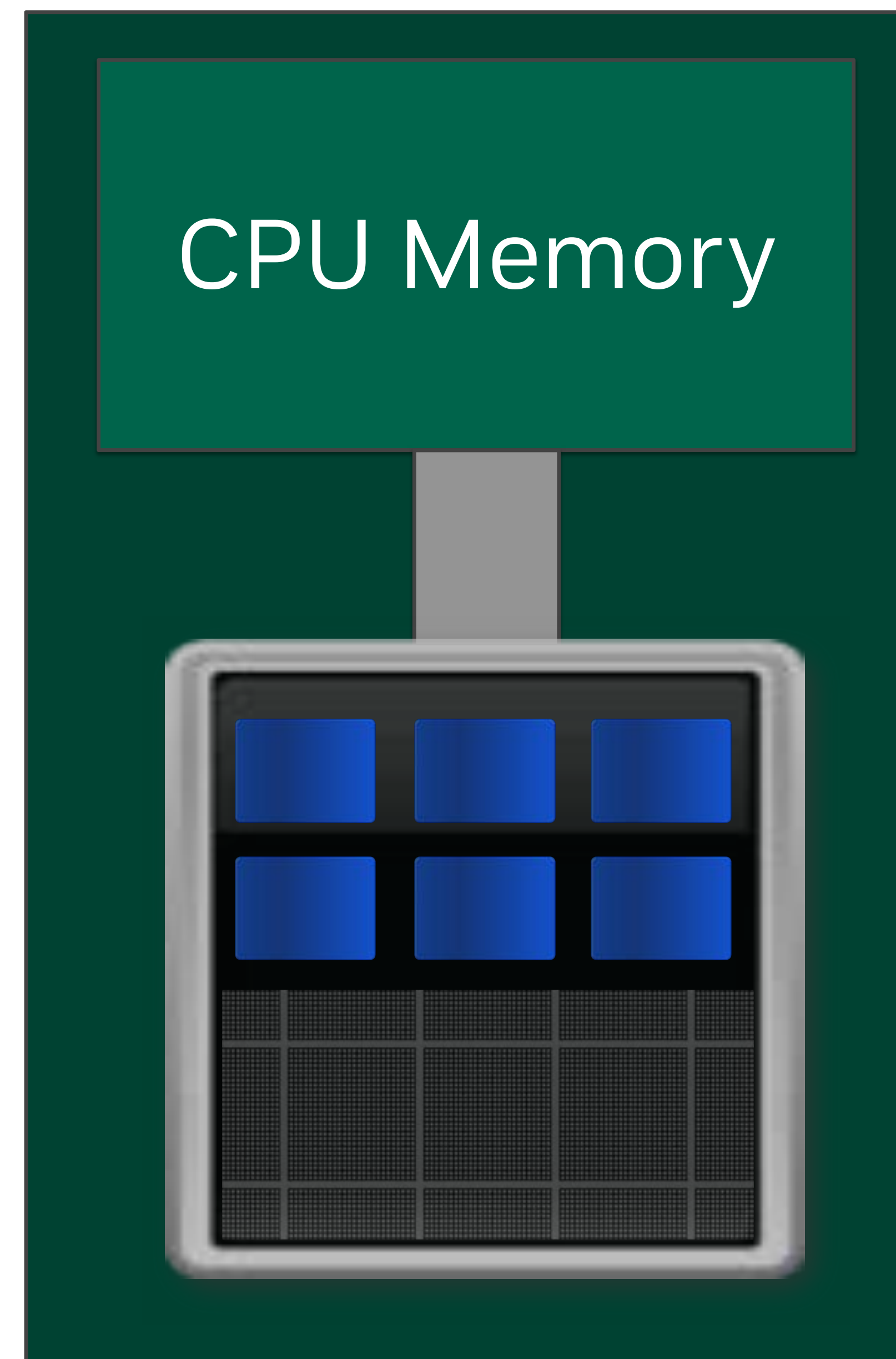
The background features a dark, almost black, space filled with numerous thin, glowing green lines that create a sense of motion and depth. On the right side, there is a prominent, glowing green grid or mesh structure that appears to be composed of many overlapping, slightly offset planes, giving it a three-dimensional, crystalline appearance. The overall aesthetic is futuristic and technical.

# CUDA

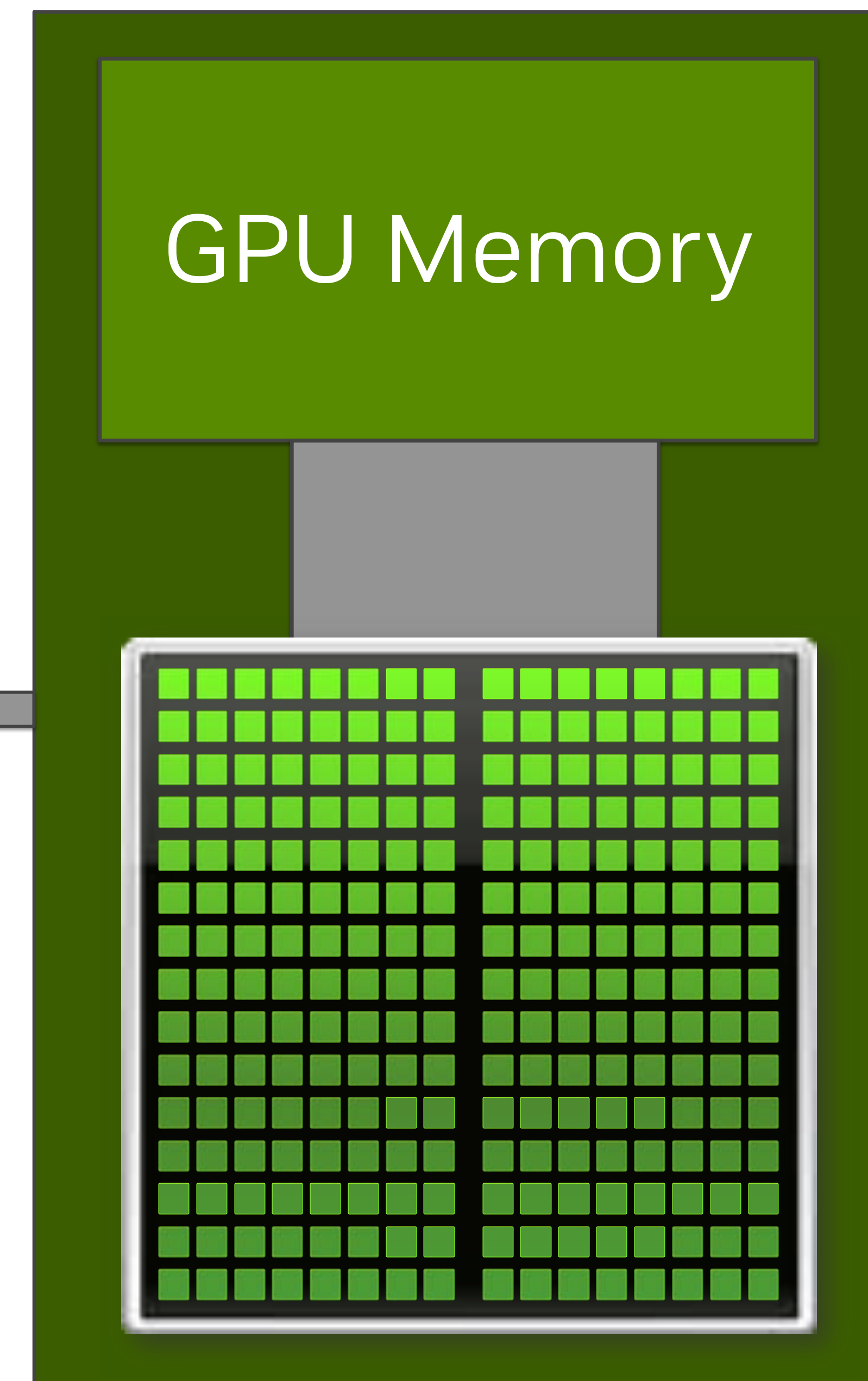
- プログラミングモデル
- アーキテクチャ

# GPU コンピューティング

CPU



PCI



GPU

- 高スループット指向のプロセッサ
- 分離されたメモリ空間

# GPU プログラム

## CPU

```
void saxpy(int n, float a,
           float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] += a*x[i];
}
```

```
...
saxpy(N, 3.0, x, y);
...
```

## CUDA

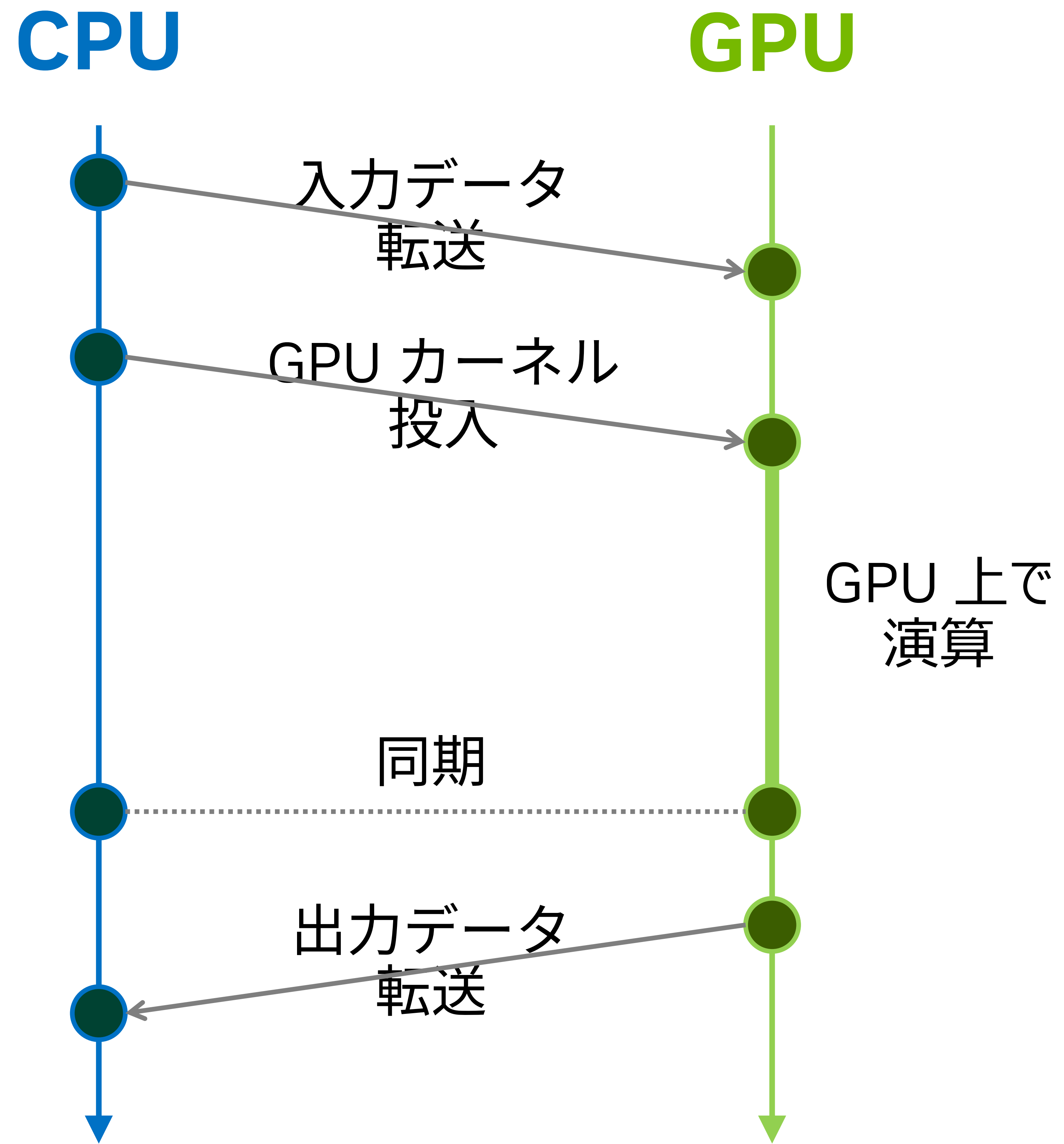
```
__global__ void saxpy(int n, float a,
                      float *x, float *y)
{
    int i = threadIdx.x + blockDim.x * blockIdx;
    if (i < n)
        y[i] += a*x[i];
}

...

cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, size, cudaMemcpyHostToDevice);
saxpy<<< N/128, 128 >>>(N, 3.0, d_x, d_y);
cudaDeviceSynchronize();
cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);

...
```

# GPU 実行の基本的な流れ



- GPU は、CPU からの制御で動作
- 入力データ: CPU から GPU に転送 (H2D)
- GPU カーネル: CPU から投入
- 出力データ: GPU から CPU に転送 (D2H)

\*GPU カーネル : GPU 上で実行される関数

# GPU プログラム

## CPU

```
void saxpy(int n, float a,
           float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] += a*x[i];
}
```

## CUDA

```
__global__ void saxpy(int n, float a,
                      float *x, float *y)
{
    int i = threadIdx.x + blockDim.x * blockIdx;
    if (i < n)
        y[i] += a*x[i];
}
```

入力データ転送

カーネル起動

同期

出力データ転送

```
...
saxpy(N, 3.0, x,
...

```

```
...
cudaMemcpy(d_x, x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, size, cudaMemcpyHostToDevice);
saxpy<<< N/128, 128 >>>(N, 3.0, d_x, d_y);
cudaDeviceSynchronize();
cudaMemcpy(y, d_y, size, cudaMemcpyDeviceToHost);
...

```

# GPU カーネル

## CPU

```
void saxpy(int n, float a,
          float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] += a*x[i];
}

...
saxpy(N, 3.0, x, y);
...
```

## CUDA

```
__global__ void saxpy(int n, float a,
                    float *x, float *y)
{
    int i = threadIdx.x + blockDim.x * blockIdx;
    if (i < n)
        y[i] += a*x[i];
}

...
saxpy<<< N/128, 128 >>>(N, 3.0, d_x, d_y);
...
```

Global スレッド ID

- GPU カーネル: 1 つの GPU スレッドの処理内容を記述
- 基本: 1 つの GPU スレッドが、1 つの配列要素を担当



# Execution Configuration

ブロック数とブロックサイズ

スレッド ID

```
__global__ void saxpy(int n, float a,  
                    float *x, float *y)  
{  
    int i = threadIdx.x + blockDim.x * blockIdx;  
    if (i < n)  
        y[i] += a*x[i];  
}  
  
...  
saxpy<<< N/128, 128 >>>(N, 3.0, d_x, d_y);  
...
```

ブロック ID

ブロックサイズ

ブロック数

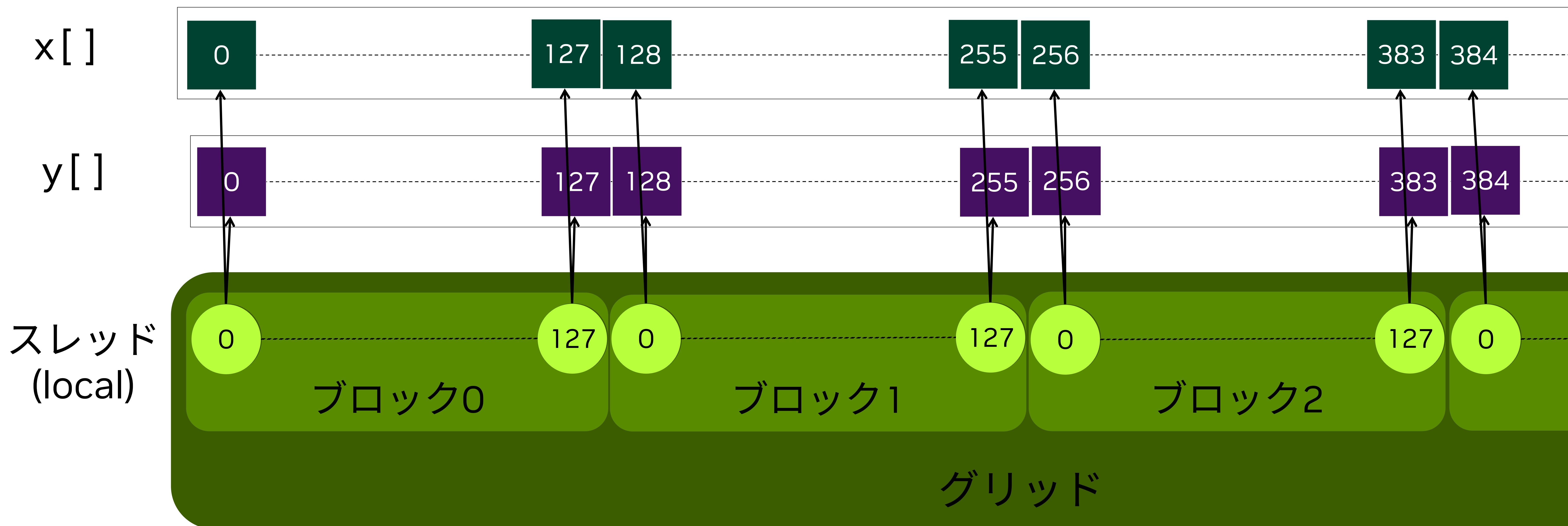
ブロックサイズ

ブロック数 x ブロックサイズ  $\geq$  配列要素数

# スレッド階層

スレッド、ブロック、グリッド

$$y[i] = a * x[i] + y[i]$$



- ブロックサイズ (スレッド数/ブロック) は、カーネル毎に設定可能
- 推奨: 128 or 256 スレッド

# Execution Configuration

ブロック数とブロックサイズ

```
__global__ void saxpy(int n, float a,  
                    float *x, float *y)  
{  
    int i = threadIdx.x + blockDim.x * blockIdx;  
    if (i < n)  
        y[i] += a*x[i];  
}  
  
...  
saxpy<<< N/256, 256 >>>(N, 3.0, d_x, d_y);  
...
```

ブロック数

ブロックサイズ

ブロック数 x ブロックサイズ = 配列要素数

## 2D 配列の GPU カーネル例

```
__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    int j = threadIdx.y + blockDim.y * blockIdx.y;
    if ( i < N && j < N )
        C[i][j] = A[i][j] + B[i][j];
}

...
dim3 sizeBlock( 64, 4 );
dim3 numBlocks( N/sizeBlock.x, N/sizeBlock.y );
MatAdd<<< numBlocks, sizeBlock >>>(A, B, C);
...
```

Global スレッド ID (x)

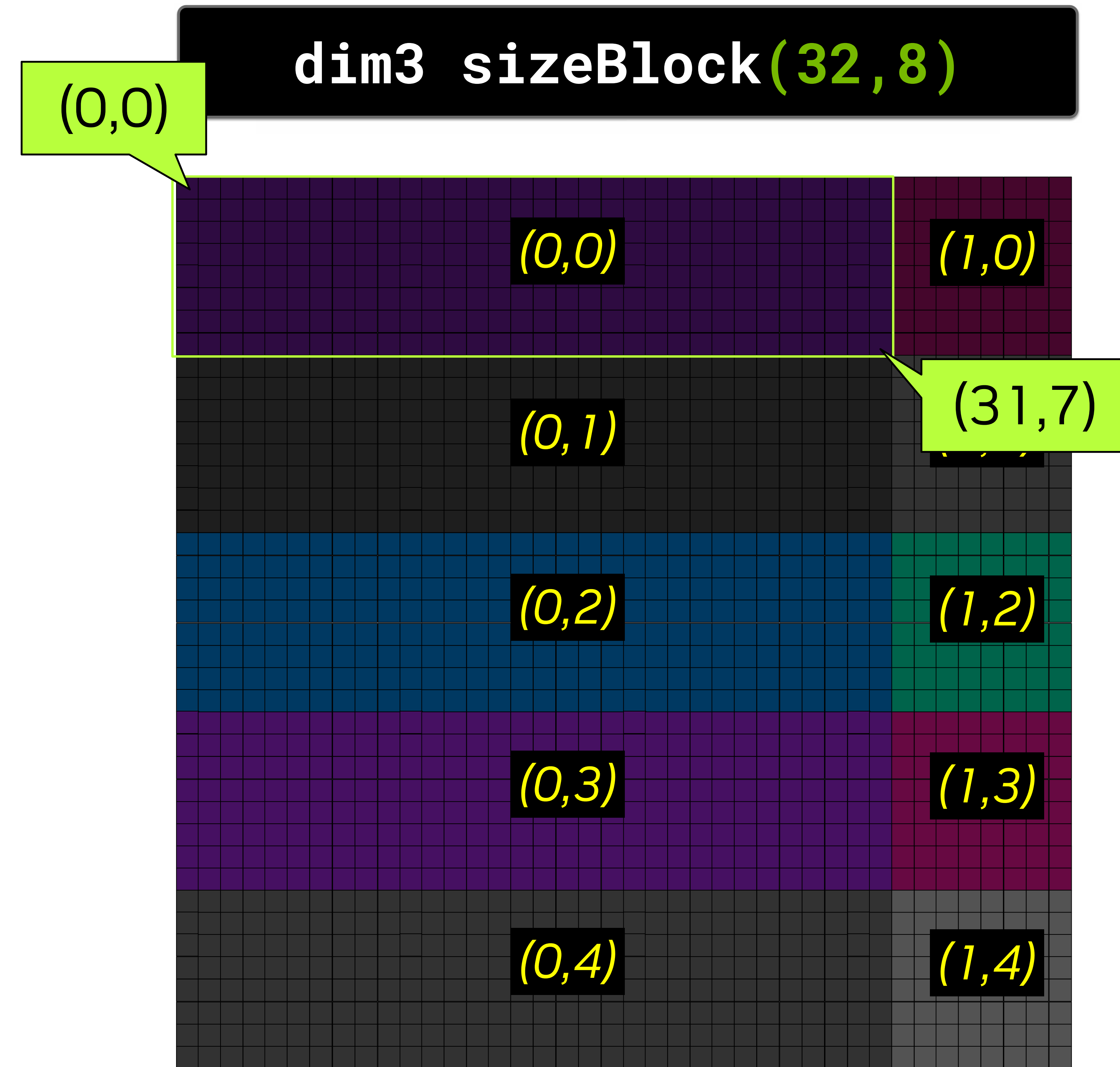
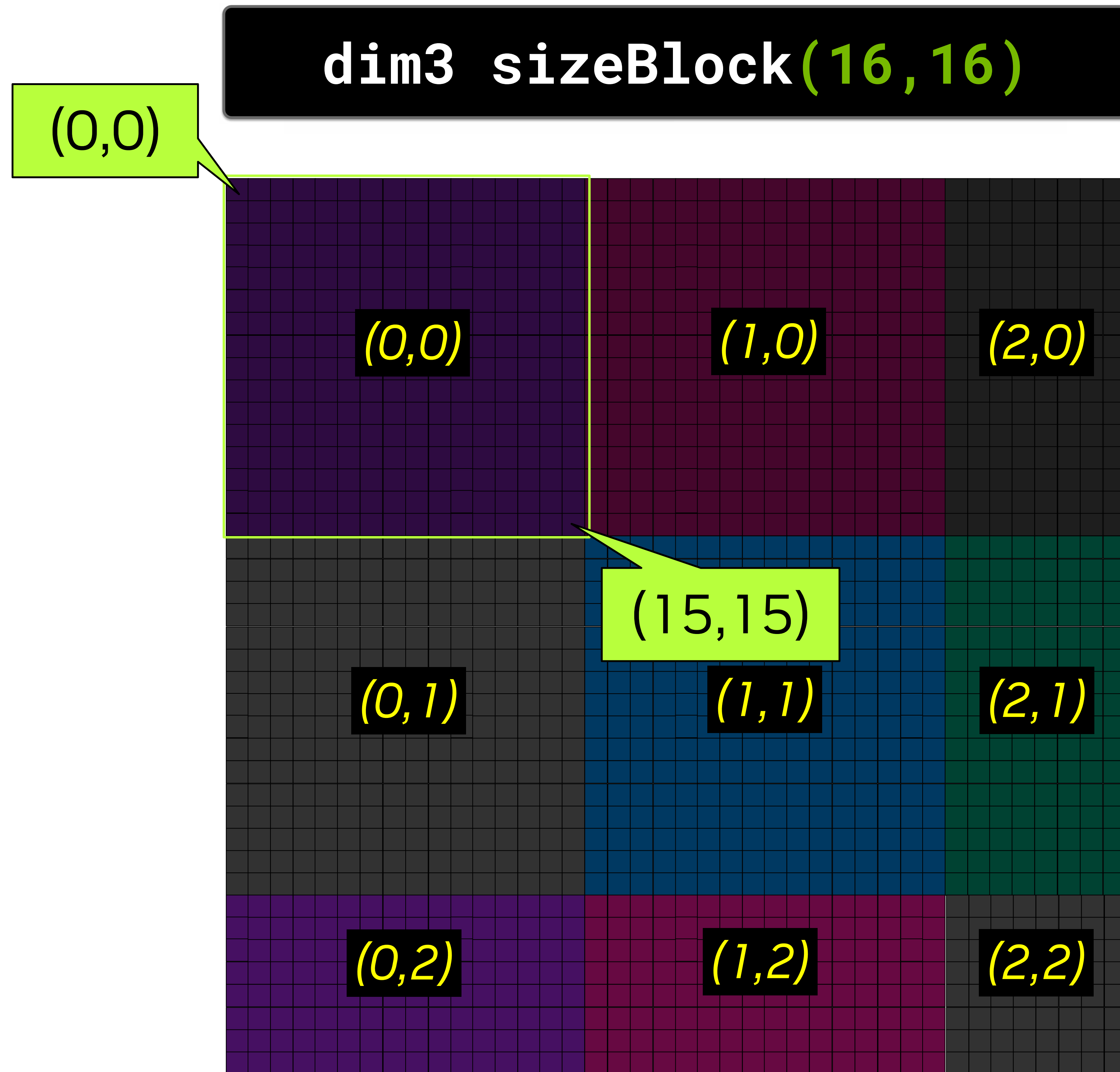
Global スレッド ID (y)

ブロックサイズ (x,y)

ブロック数 (x,y)

- ブロックサイズ (ブロック形状) は、1D~3D で表現可能

# ブロック マッピング、スレッド マッピング



ブロック ID (blockIdx)

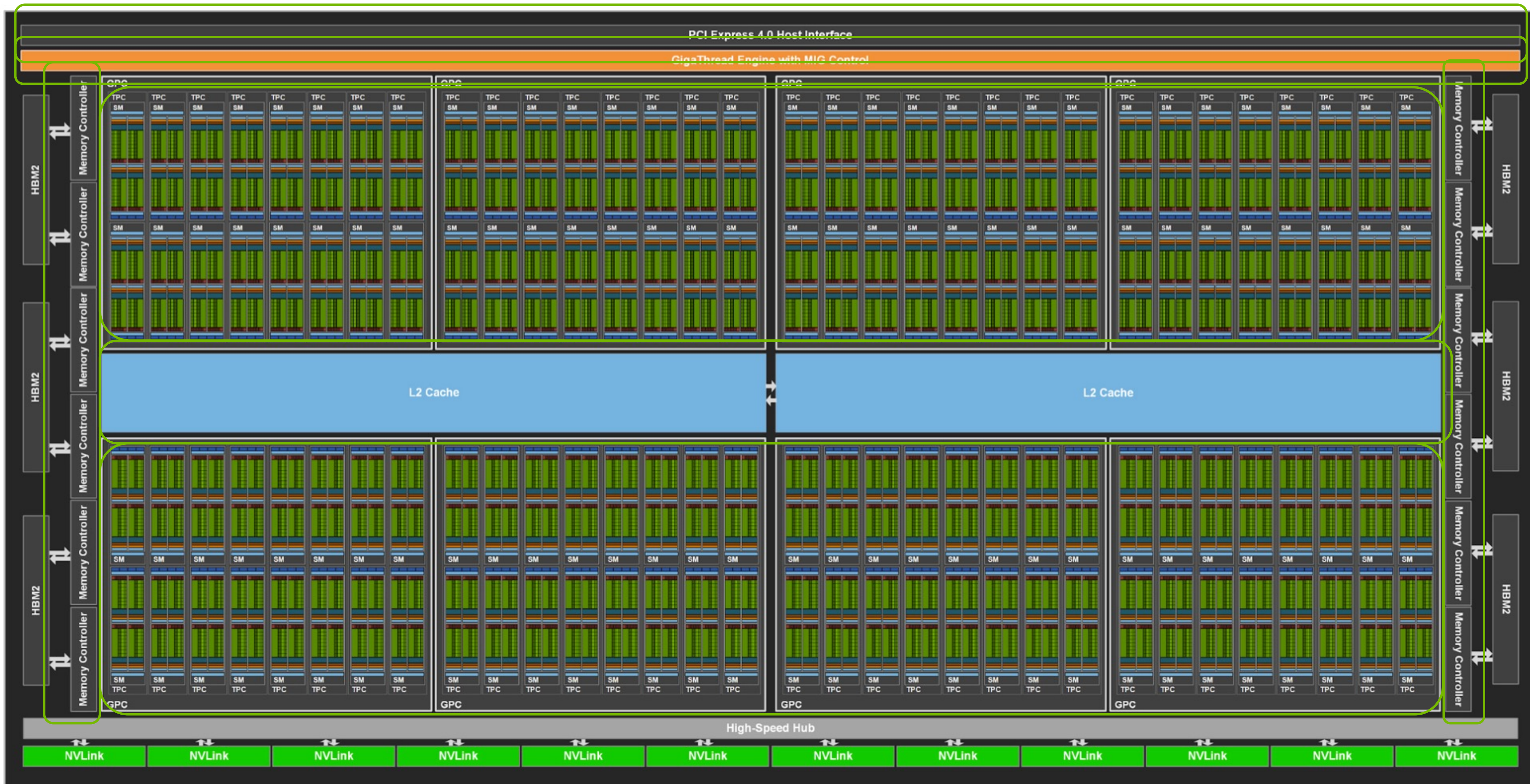
スレッド ID (threadIdx)

The background features a complex pattern of thin, overlapping lines in shades of green and white against a black background. The lines are arranged in a way that suggests depth and movement, with some lines appearing to curve and others to intersect, creating a sense of a three-dimensional structure or a dynamic flow. The overall effect is reminiscent of a stylized, abstract representation of a network or a data stream.

# CUDA

- プログラミングモデル
- **アーキテクチャ**

# GPU アーキテクチャ概要



- PCI I/F
  - ホスト接続インタフェース
- Giga Thread Engine
  - SM に処理を割り振るスケジューラ
- DRAM I/F (HBM2e)
  - 全 SM、PCI I/F からアクセス可能なメモリ (デバイスメモリ、フレームバッファ)
- L2 cache (40 MB)
  - 全 SM からアクセス可能な R/W キャッシュ
- SM (Streaming Multiprocessor)
  - 「並列」プロセッサ、A100 : 108

# SM (Streaming Multi-processor)

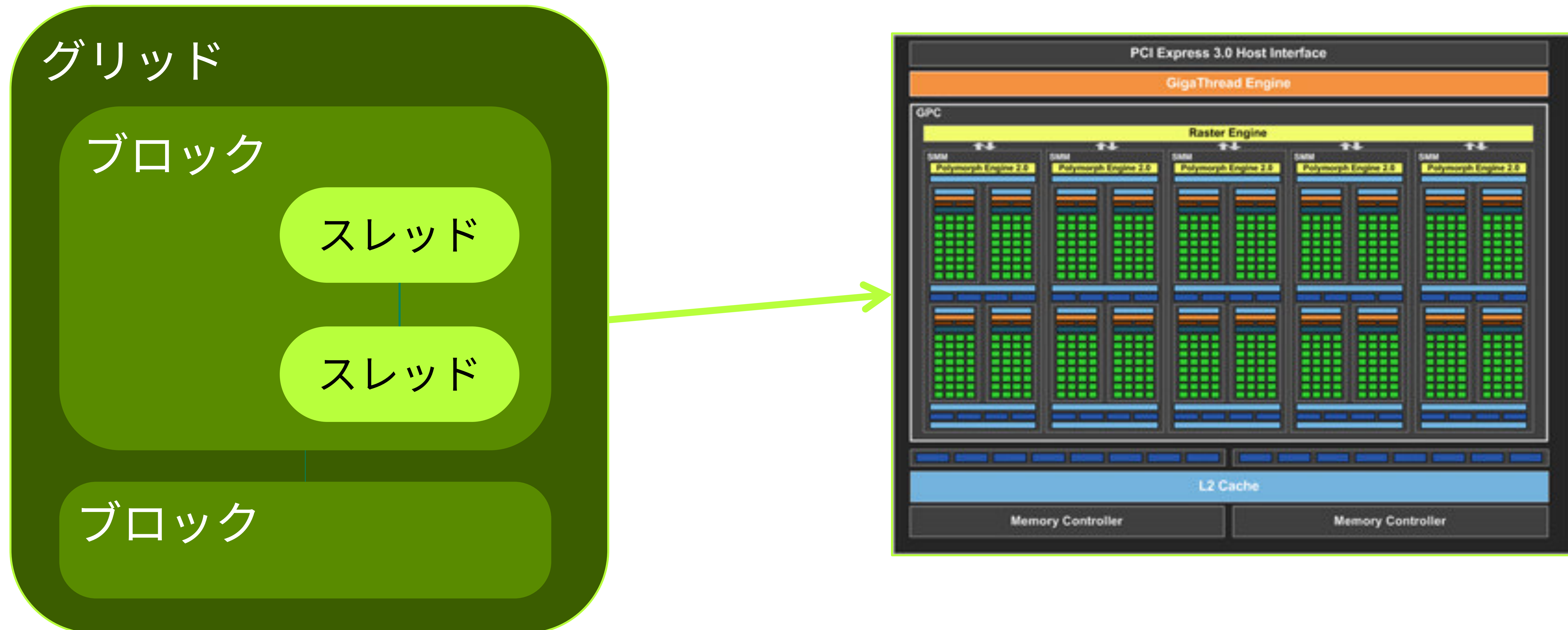


- 演算ユニット
  - INT32: 64 個
  - FP32: 64 個
  - FP64: 32 個
  - TensorCore: 4 個
- Other units
  - LD/ST, SFU, etc
- レジスタ (32 bit): 64K 個
- 共有メモリ/L1 キャッシュ: 192 KB



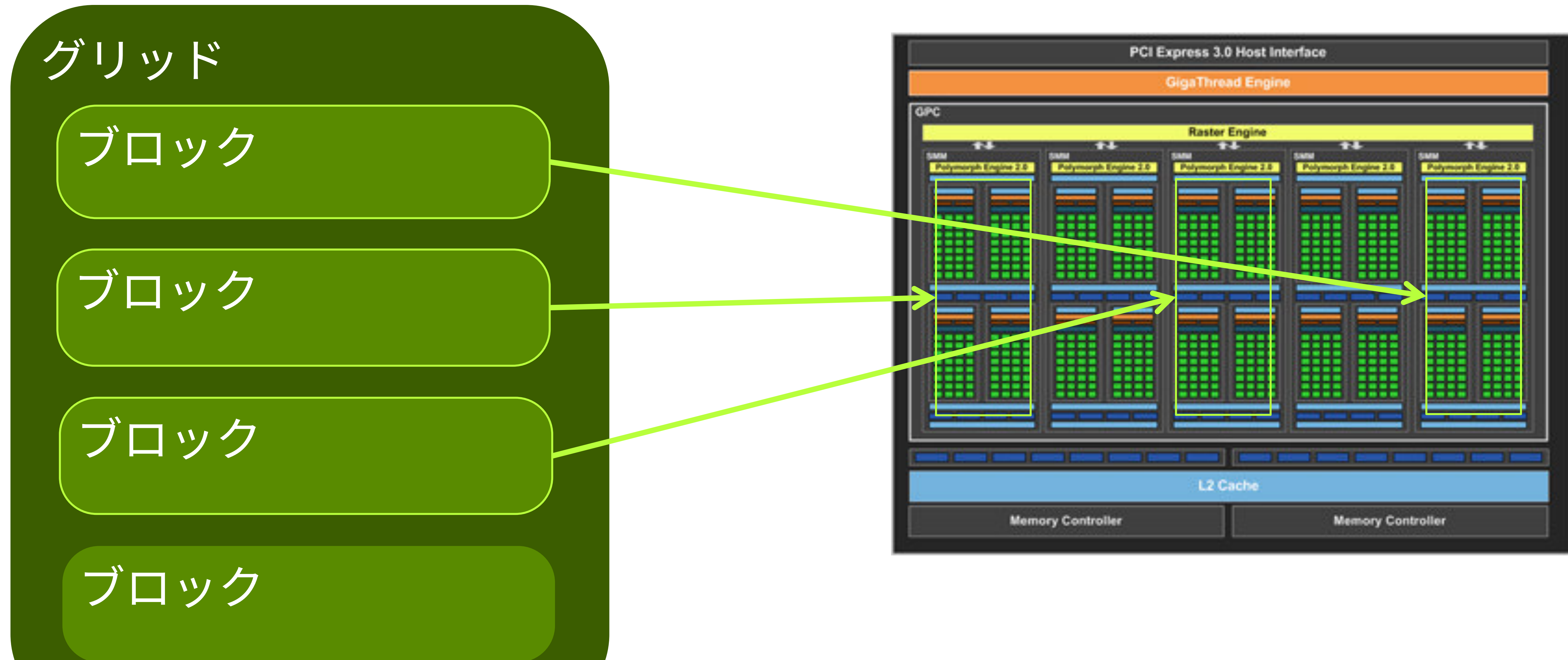
# GPU カーネル実行の流れ

- CPU が、GPU にグリッドを投入
- 具体的な投入先は、Giga Thread Engine



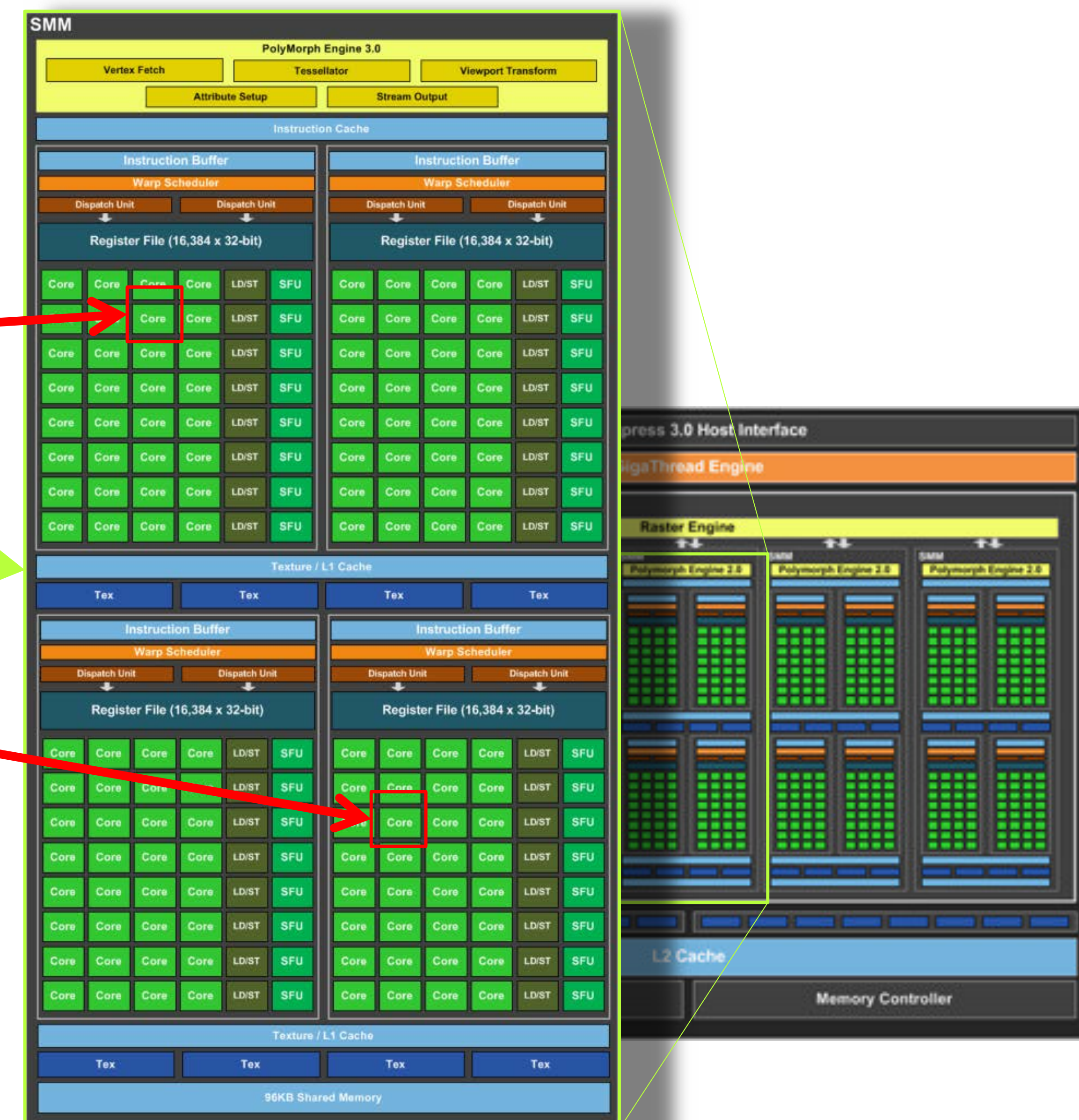
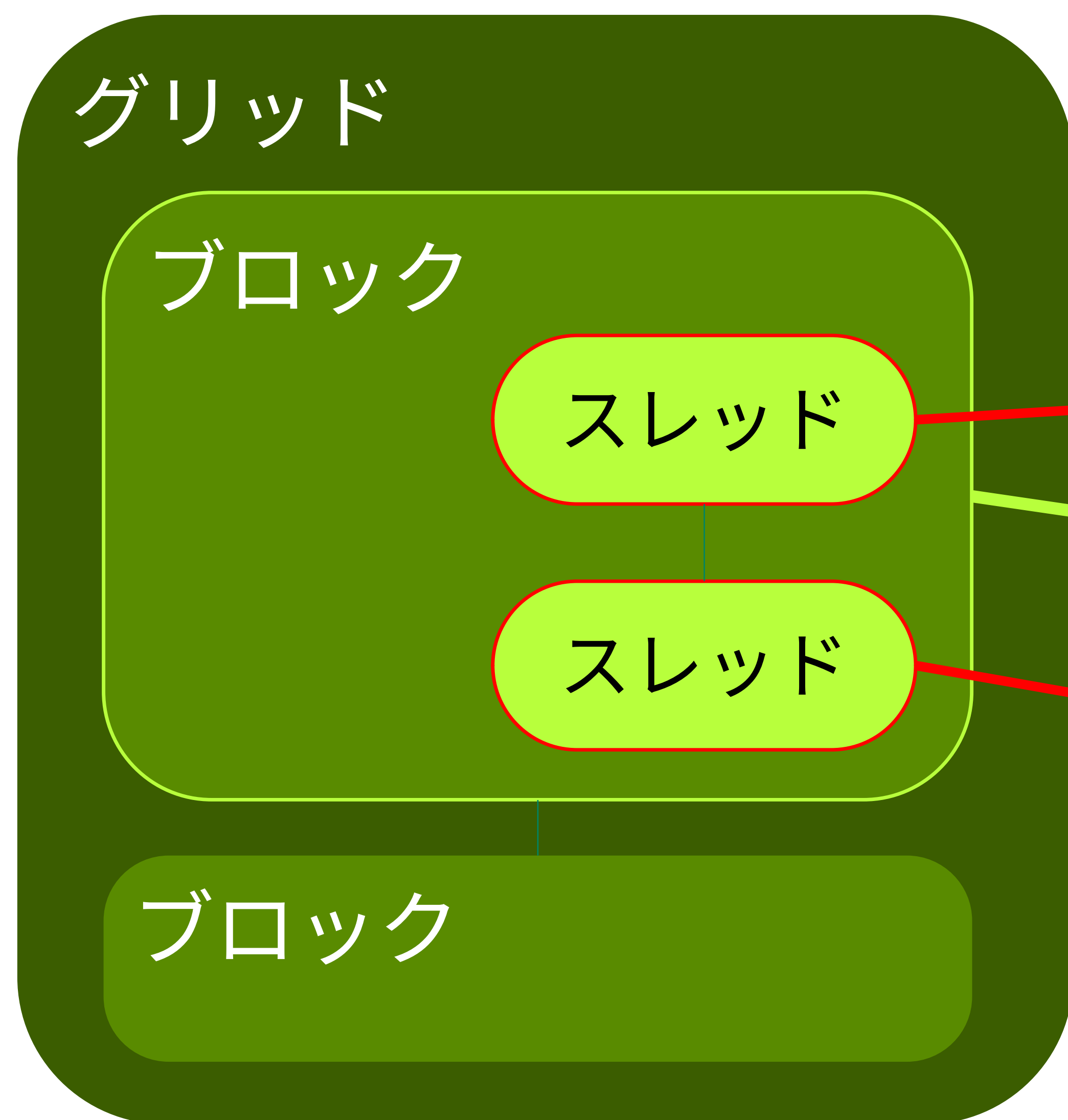
# ブロックを SM に割り当て

- 各ブロックは、互いに独立に実行
  - ブロック間では同期しない、実行順序の保証なし
- 1つのブロックは複数 SM にまたがらない
  - 1つの SM に複数ブロックが割り当てされることはある



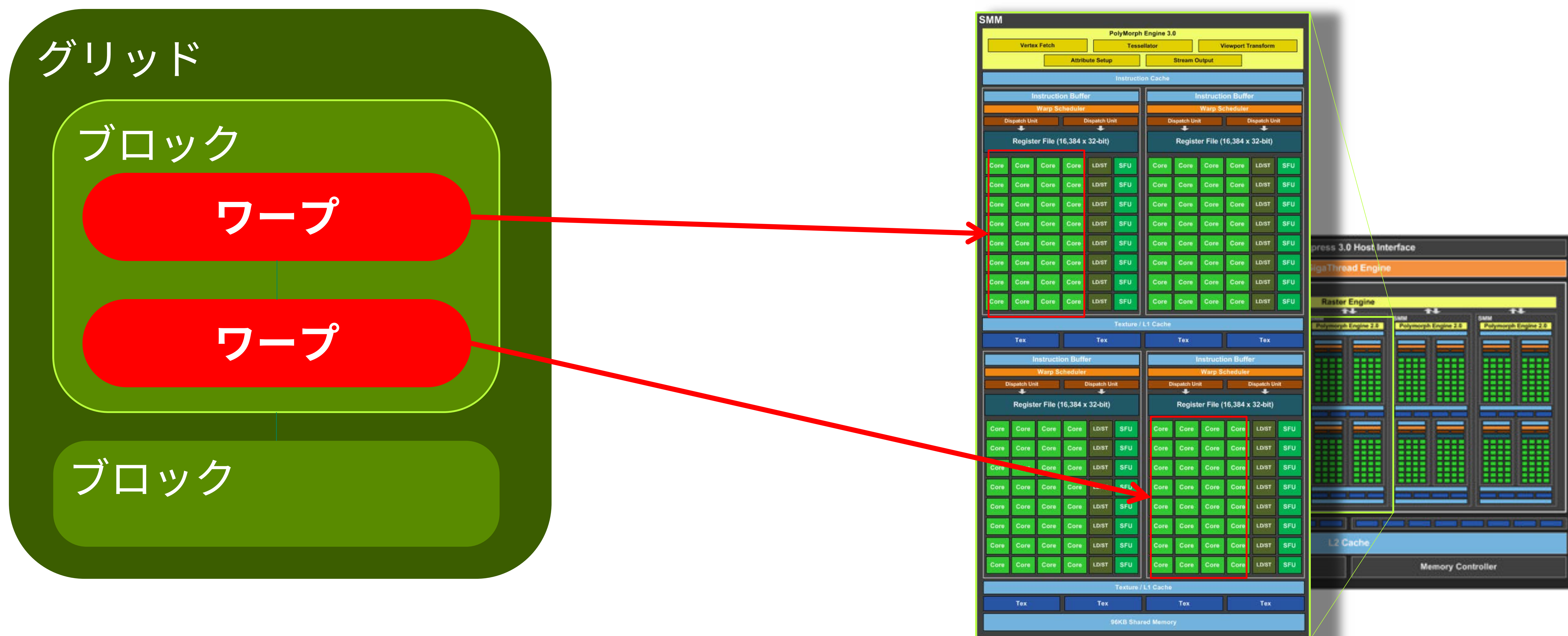
# GPU カーネル実行の流れ

- SM 内のスケジューラが、スレッドを CUDA コアに投入



# GPU カーネル実行の流れ

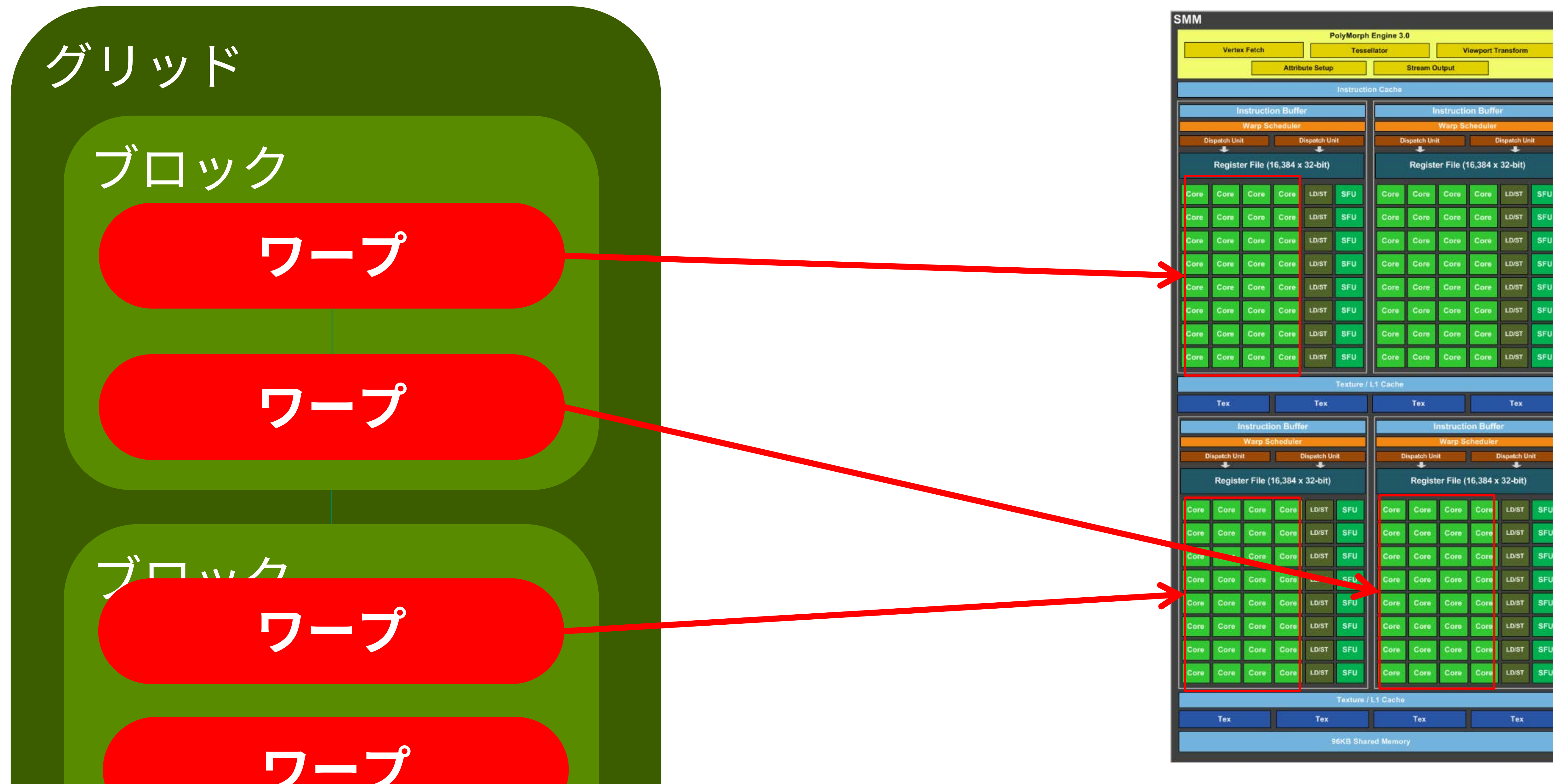
- SM 内のスケジューラが、**ワープ**を CUDA コアに投入
  - ワープ: 32 スレッドの塊
  - ブロックをワープに分割、実行可能なワープを、空 CUDA コアに割り当てる



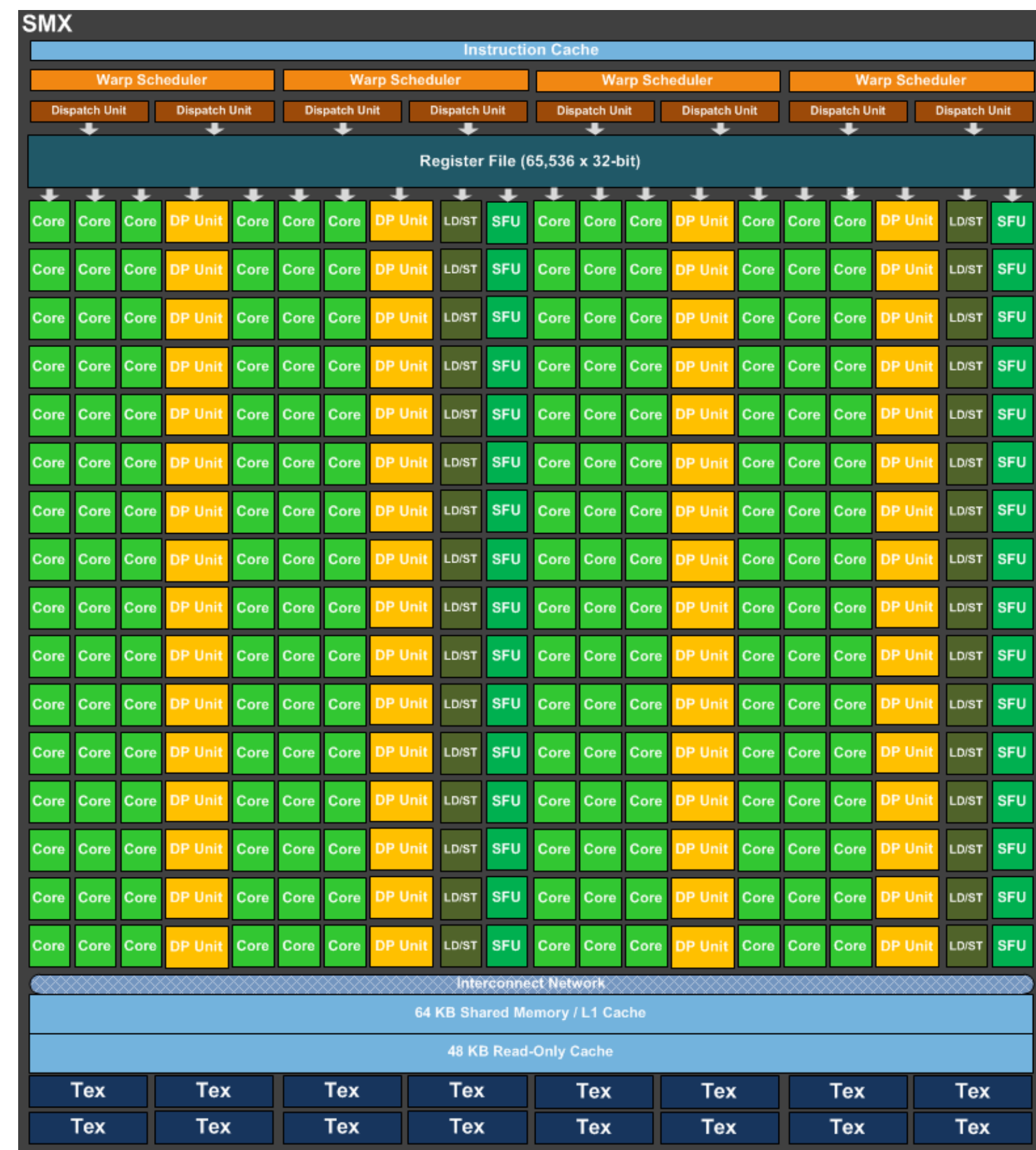
# ワーブの CUDA コアへの割り当て

- ワーブ内の 32 スレッドは、同じ命令を同期して実行
- 各ワーブは、互いに独立して実行
  - 同じブロック内のワーブは、明示的に同期可能 (`__syncthreads()`)

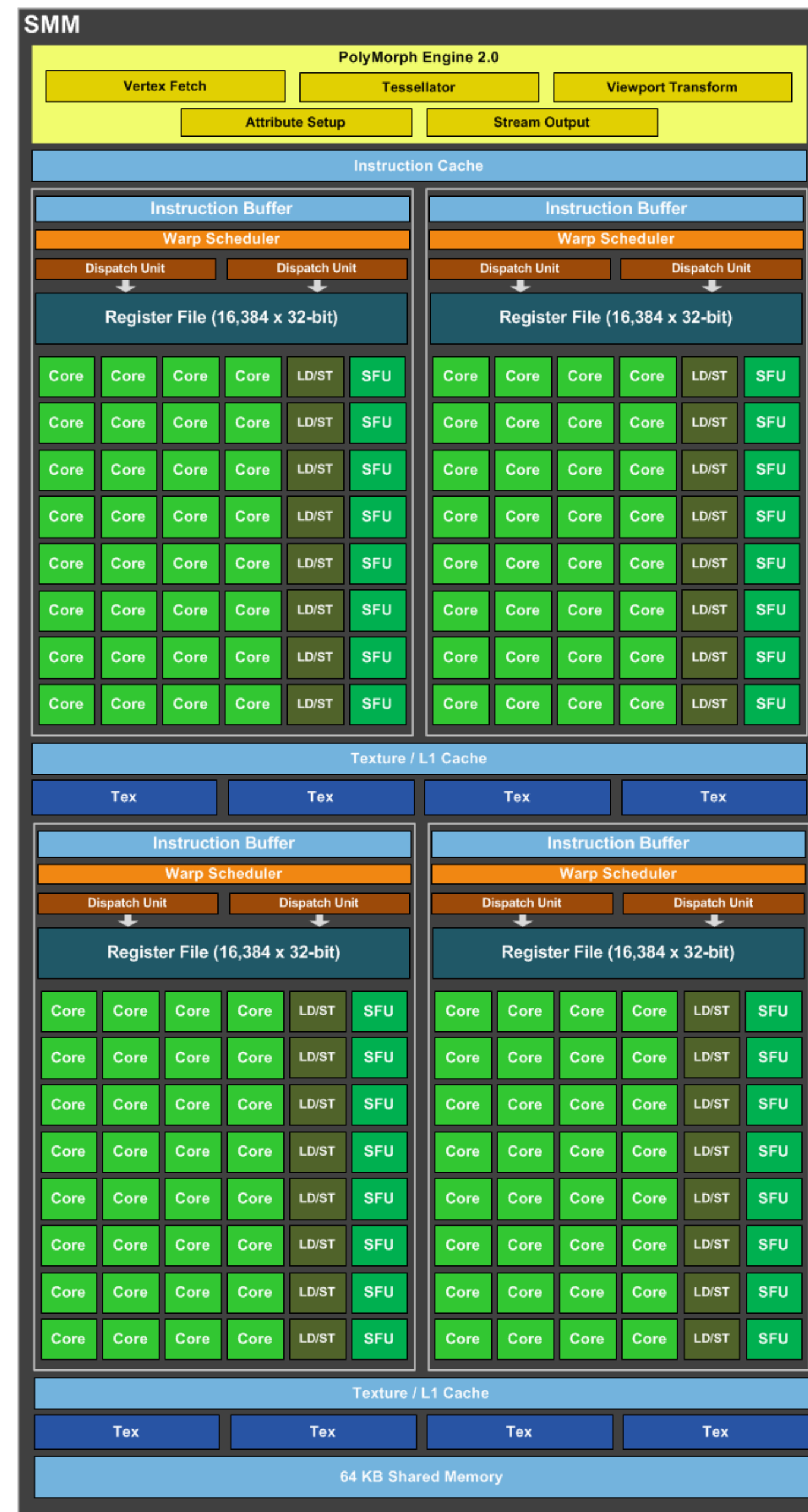
SIMT  
(Single Instruction Multiple Threads)



# GPU アーキの变化を問題としないプログラミングモデル



Kepler, CC3.5  
192 cores /SM



Maxwell, CC5.0  
128 cores /SM



Pascal, CC6.0  
64 cores /SM



Ampere, CC8.0  
64 cores /SM

