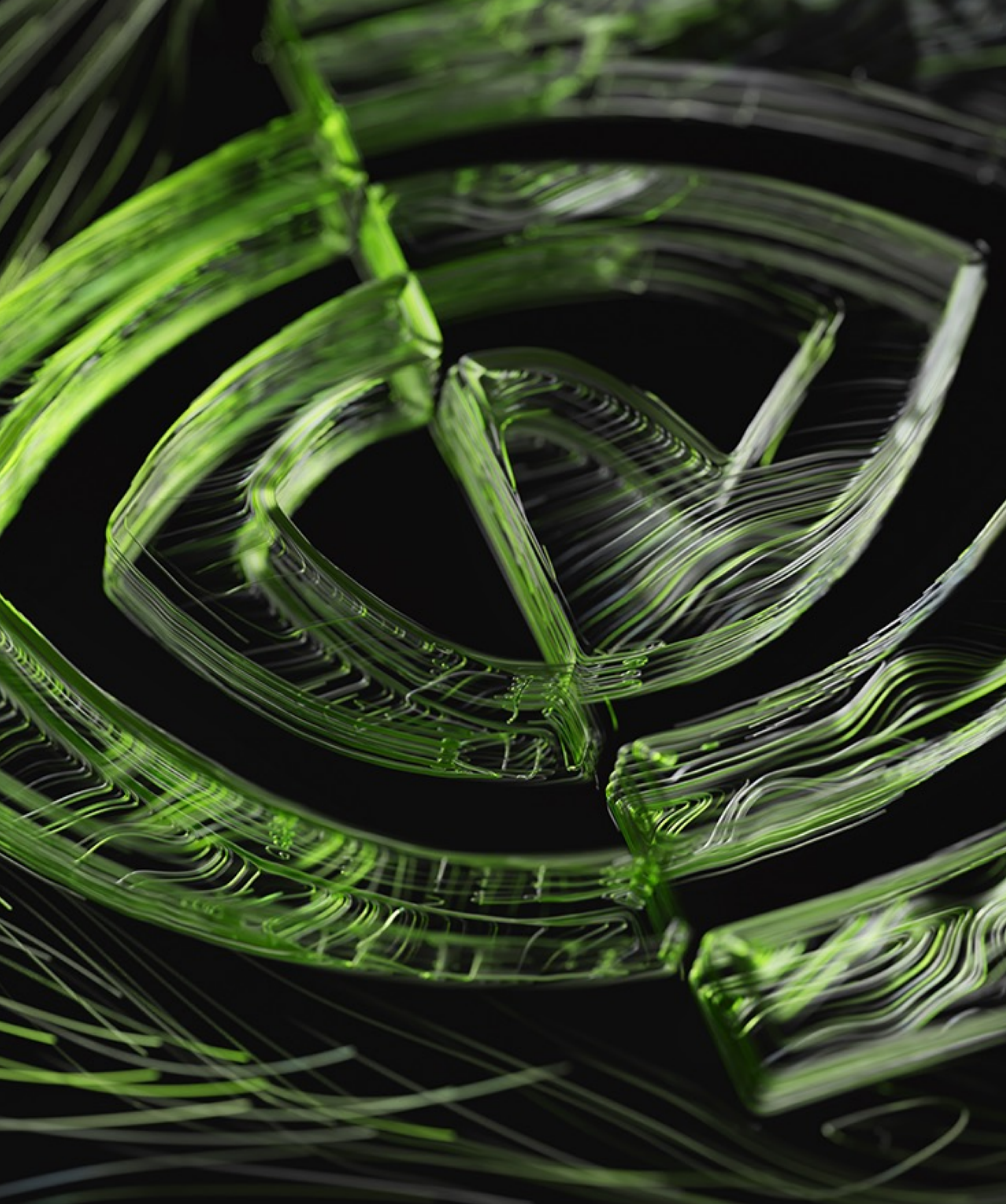
GPU コンピューティング入門 & ハンズオン



Agenda

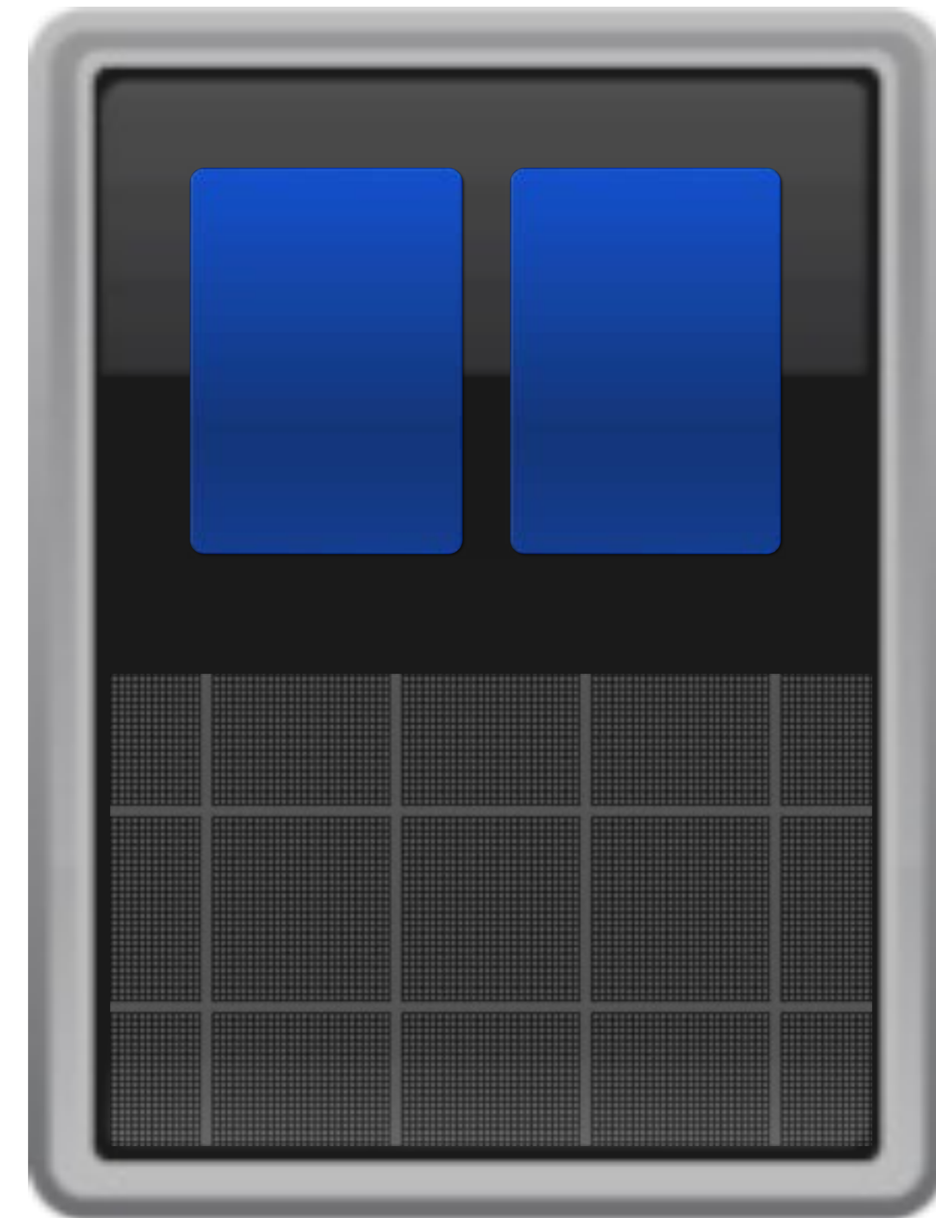
- Introduction to GPU Computing
  - GPU Computing
  - CUDA
- ハンズオン
  - Lab 1
  - Lab 2

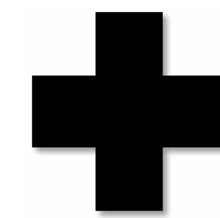
GPU Computing

GPU コンピューティング

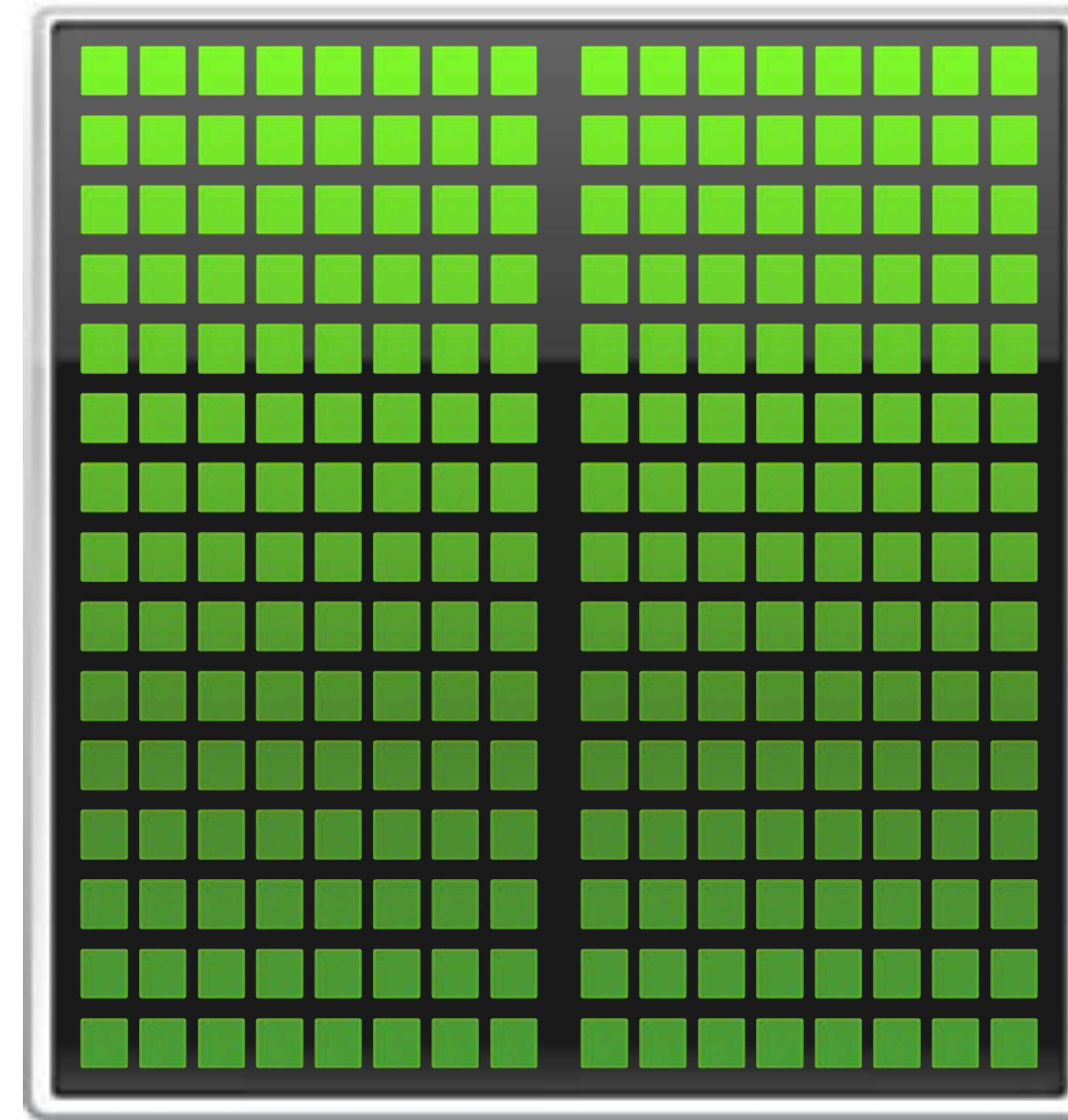
Low latency + High throughput

CPU





GPU

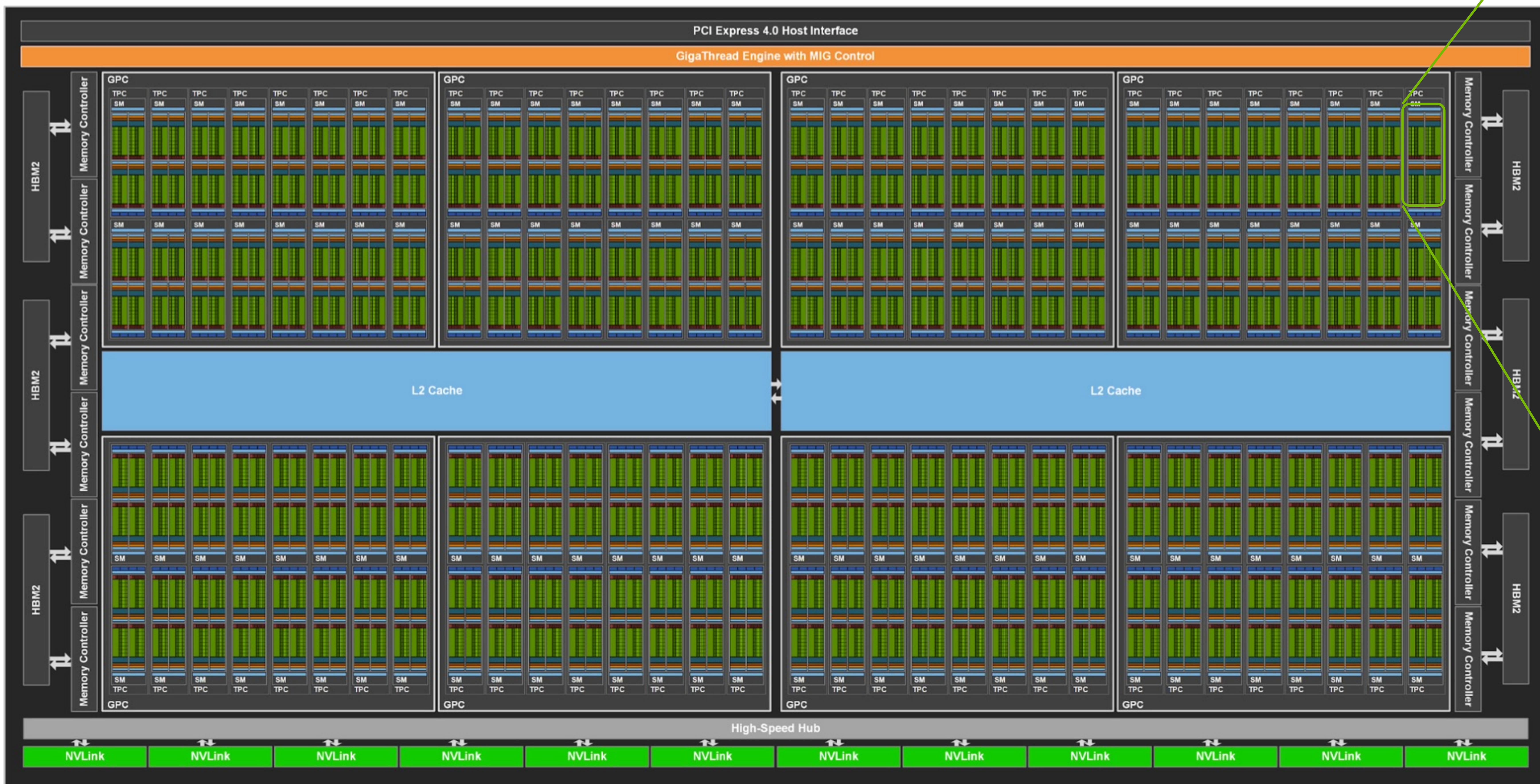


アプリケーション実行

GPU の構造

NVIDIA A100

大量の CUDA コア
並列性の抽出が鍵



108 Streaming Multiprocessor (SM) / chip

64 CUDA core / SM

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 AZ | 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/

NVIDIA. Products Solutions Industries For You Shop Drivers Support

Filter Categories Filter Industries Sort AZ AI Accelerated Share Search apps

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

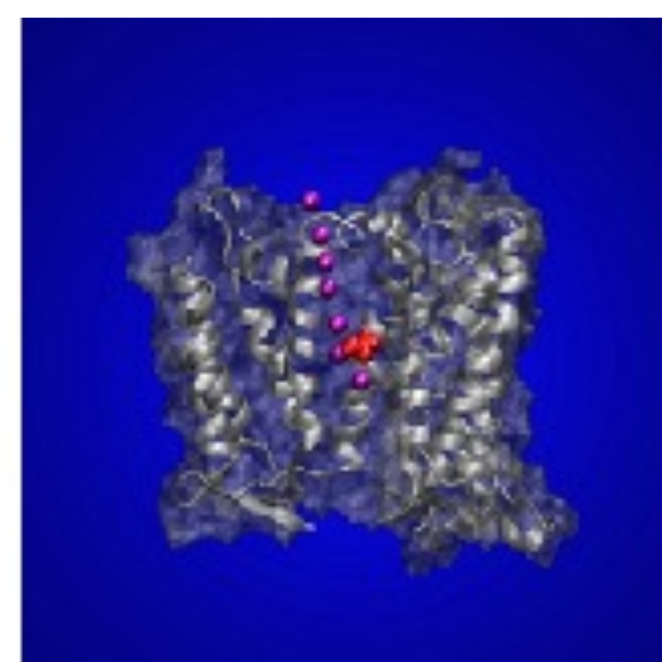
1000 以上のアプリケーションが GPU に対応

Clear Filters Apply Filters

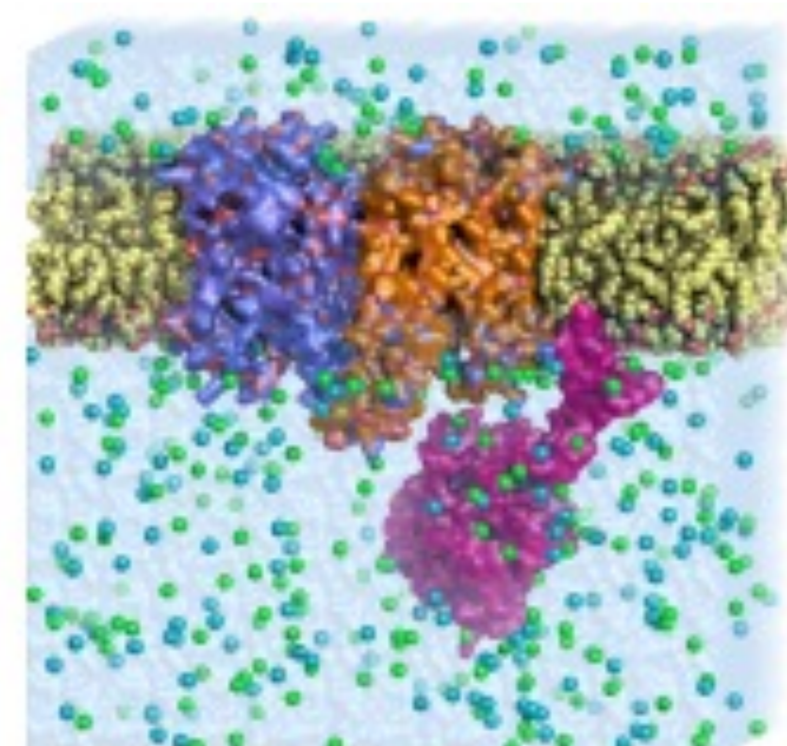
Automation of Manual Process of Financial Spreading & Analysis of Financial Documents

Leading Applications

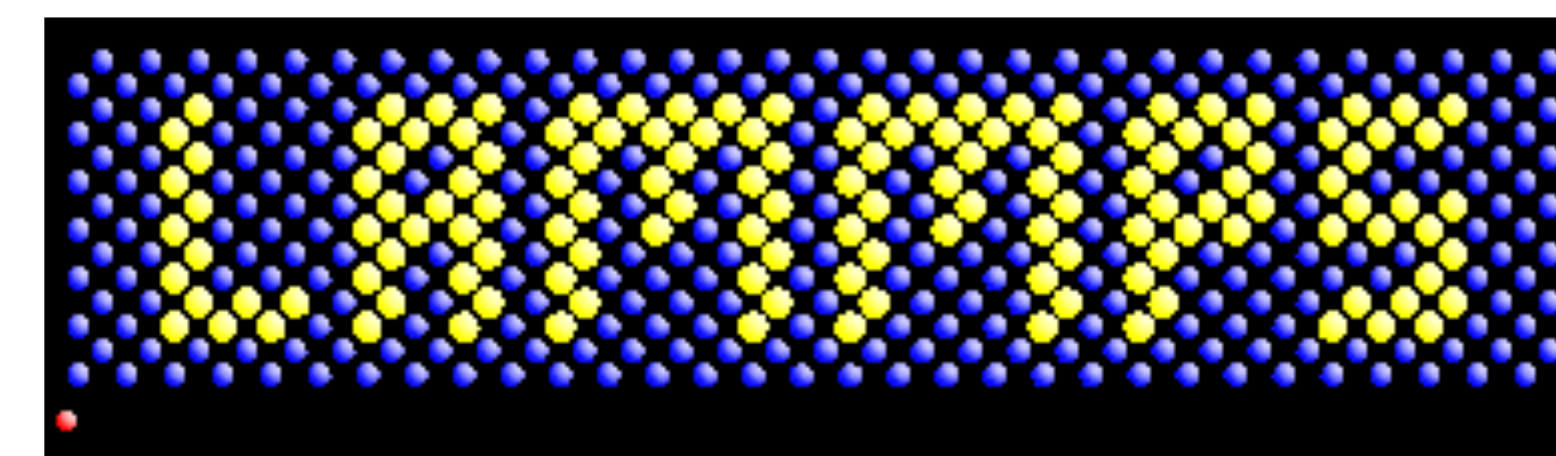
Gaussian

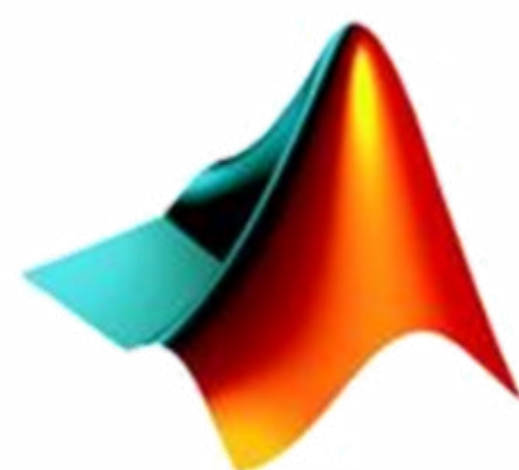


Desmond



Amber





MATLAB®

HOOMD-blue

アプリを GPU 対応する方法

Application

Library

OpenACC

CUDA

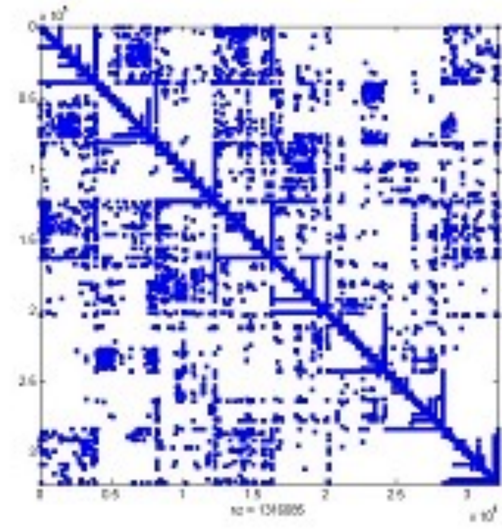
GPU 対応ライブラリに
チェンジ
簡単に開始

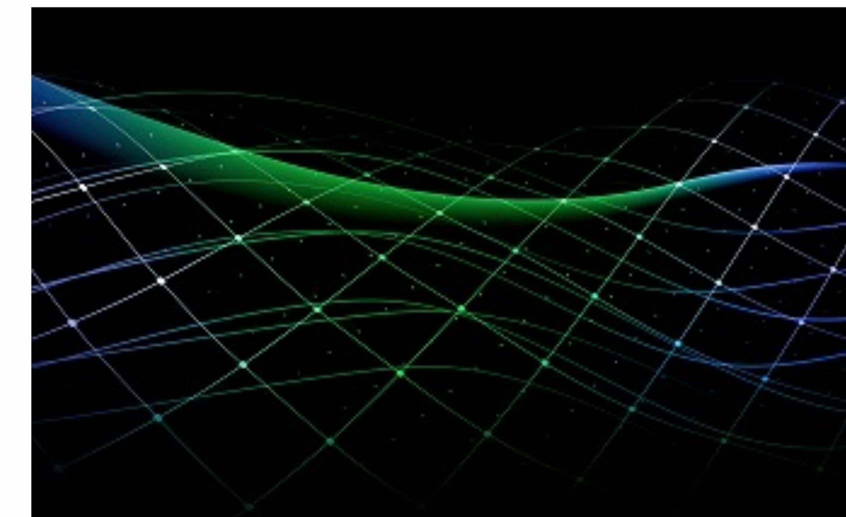
既存コードに
ディレクティブを挿入
簡単に加速

主要処理を CUDA で記述
高い自由度

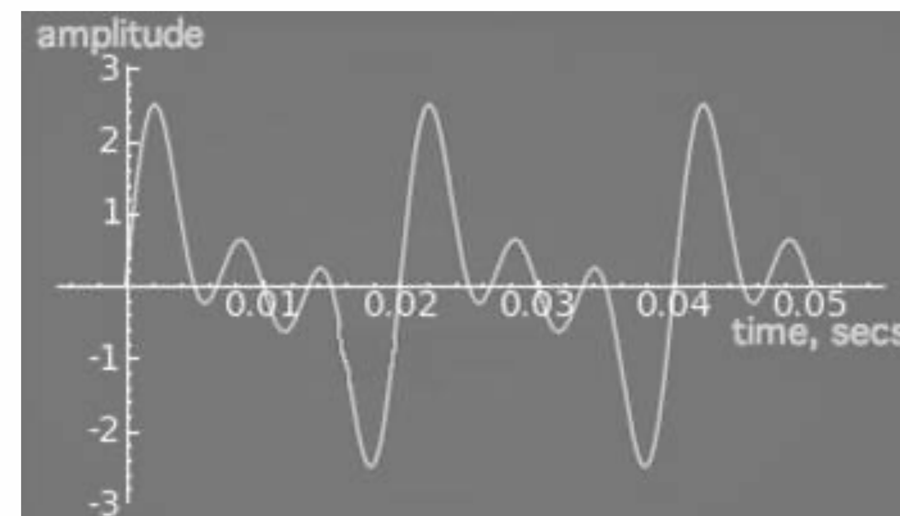
NVIDIA Libraries

cuBLAS

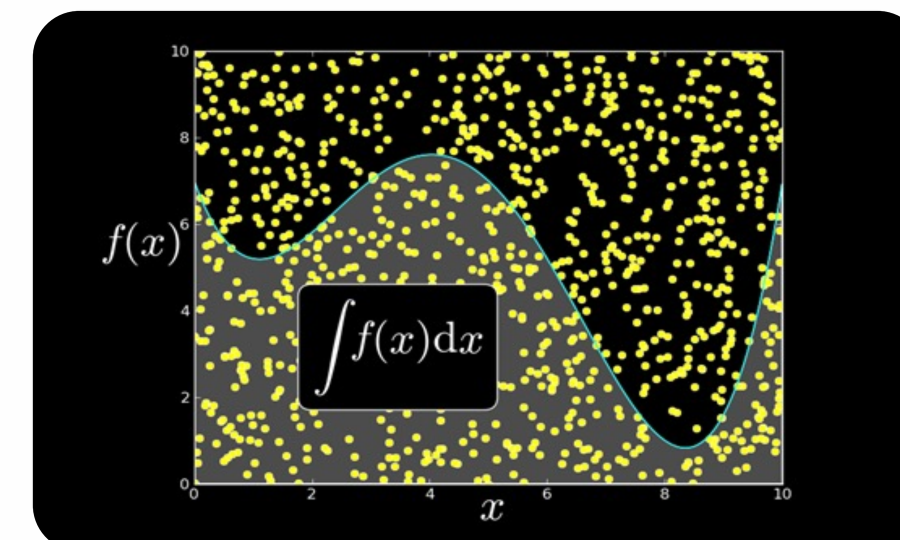

cuSPARSE


cuSOLVER

AmgX



cuFFT



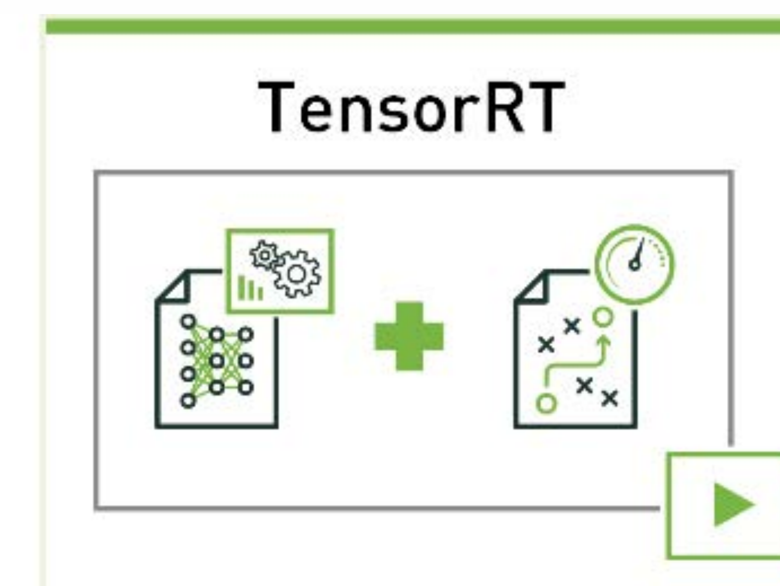
cuRAND

nvGRAPH

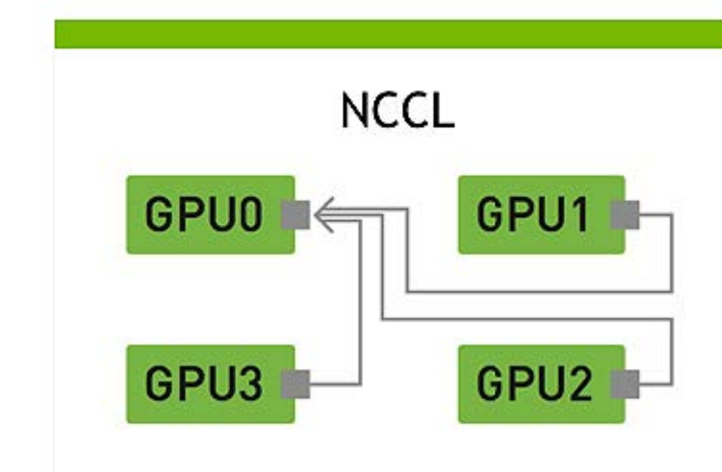
Thrust

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	OpenACC	CUDA
GPU 対応ライブラリにチェンジ 簡単に開始	既存コードにディレクティブを挿入 簡単に加速	主要処理を CUDA で記述 高い自由度

NVIDIA.

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	OpenACC	CUDA
GPU 対応ライブラリにチェンジ 簡単に開始	既存コードにディレクティブを挿入 簡単に加速	主要処理を CUDA で記述 高い自由度

NVIDIA.

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 + blodkDim.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

Application

Library

OpenACC

CUDA

GPU 対応ライブラリに
チェンジ
簡単に開始

既存コードに
ディレクティブを挿入
簡単に加速

主要処理を CUDA で記述
高い自由度

SAXPY (Y = A*X + Y)

CPU

```python
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)


...
```

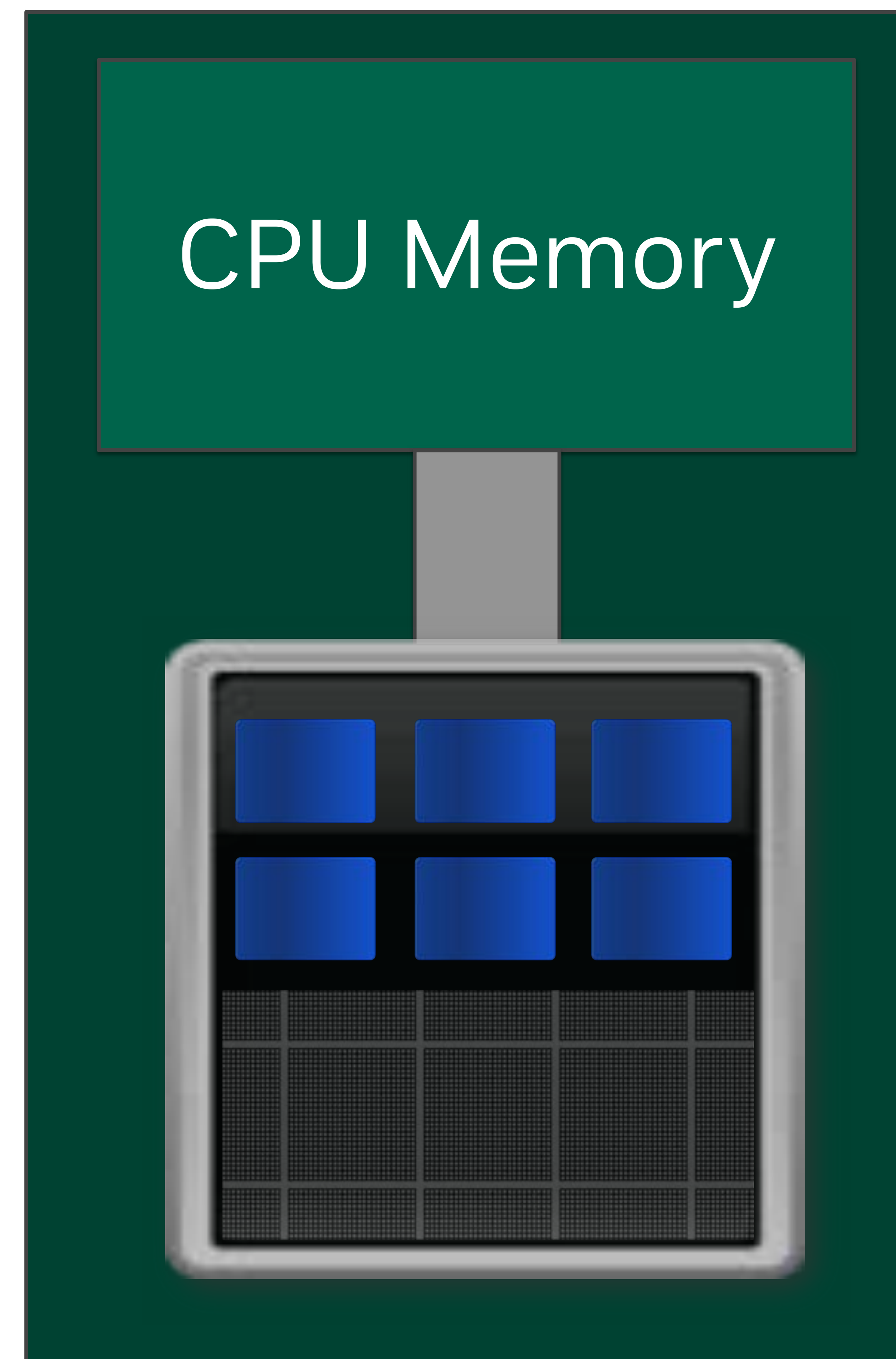
CuPy

```python
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)


...
```

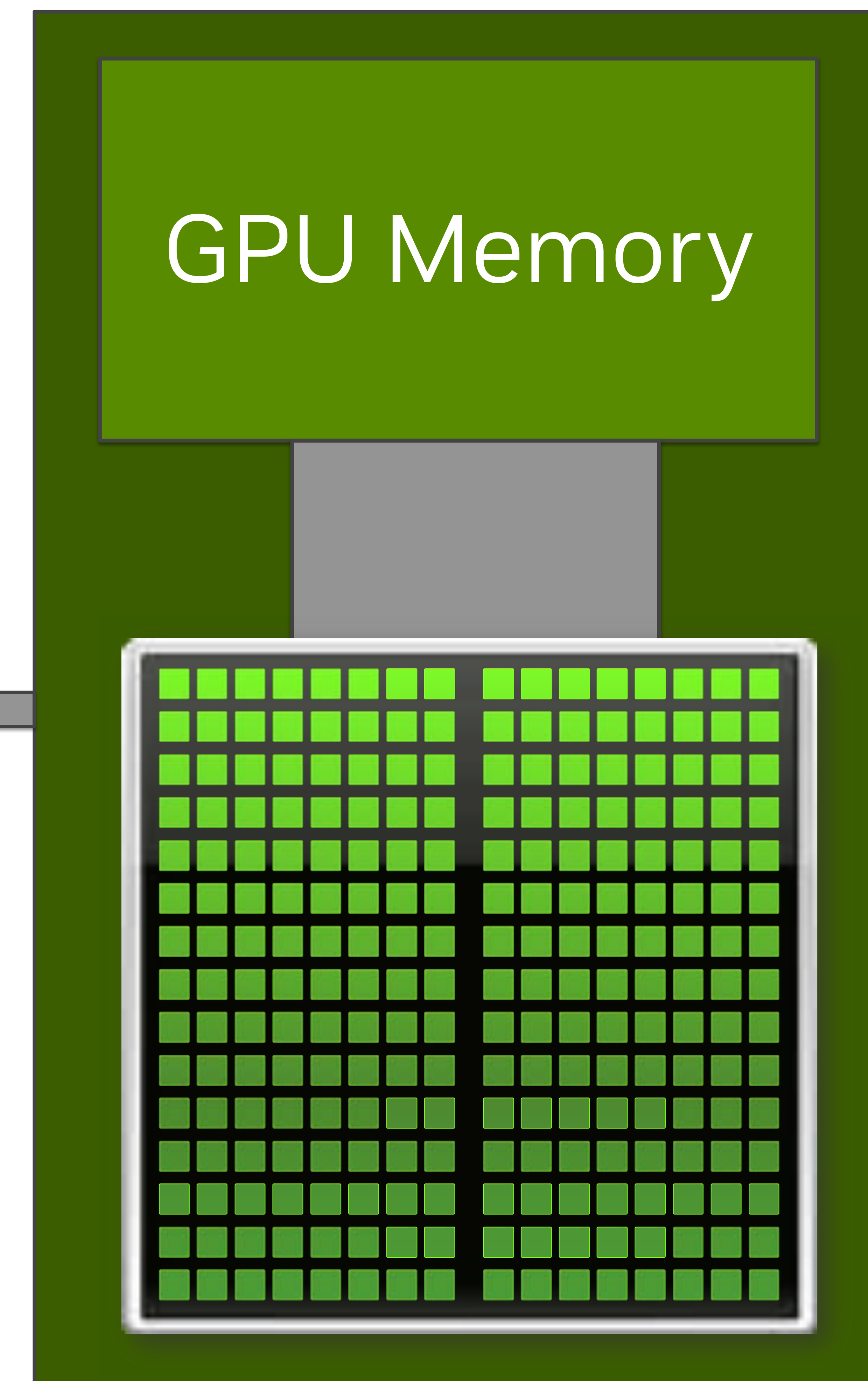
- プログラミングモデル
- アーキテクチャ

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

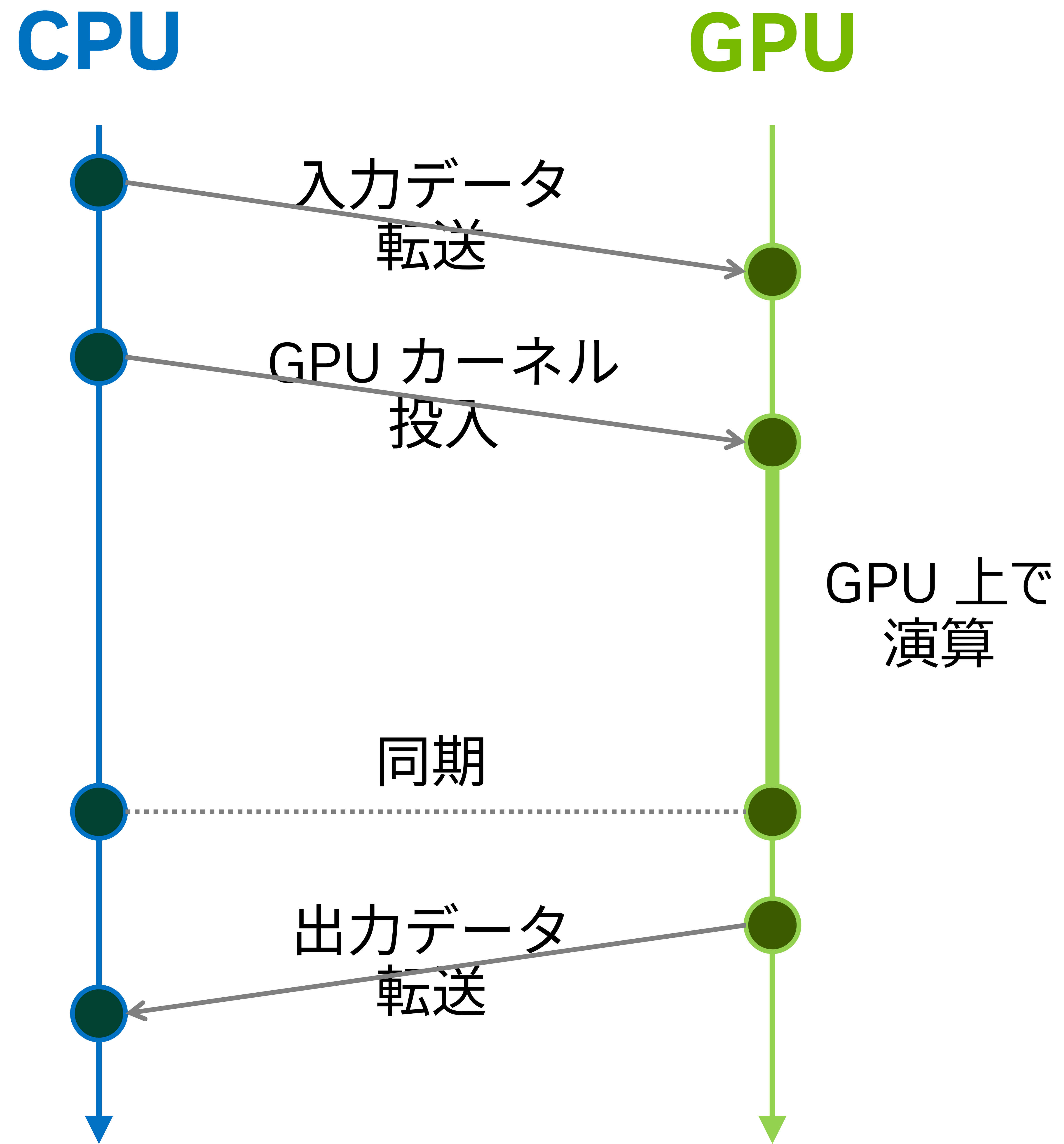
```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] += a*x[i];
}




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

CUDA

```
__global__ void saxpy(int n, float a,
                      float *x, float *y)
{
  int i = threadIdx.x + blodkDim.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);
...
```

NVIDIA.

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];
}

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

CUDA

```
__global__ void saxpy(int n, float a,
                      float *x, float *y)
{
  int i = threadIdx.x + blodkDim.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);
...
```

入力データ転送

カーネル起動

同期

出力データ転送

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 + blodkDim.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

NVIDIA

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 + blodkDim.x * blockIdx;
  if (i < n)
    y[i] += a*x[i];
}

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

スレッド ID

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 ≧ 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

NVIDIA.

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 + blodkDim.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], f
{
   int i = threadIdx.x + blodkDim.x * blockIdx.x;
   int j = threadIdx.y + blodkDim.y * blockIdy.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)

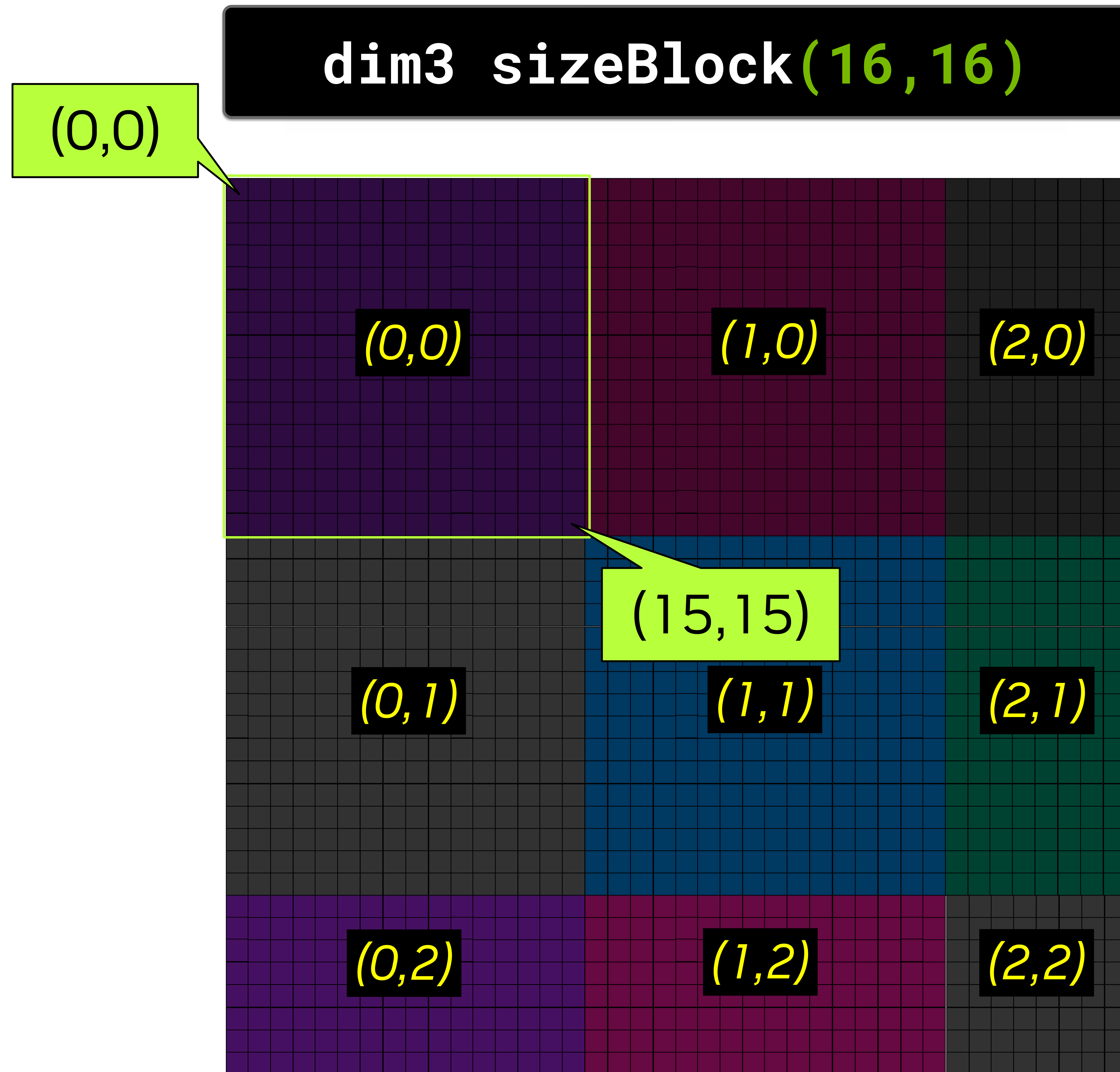
Global スレッド ID (y)

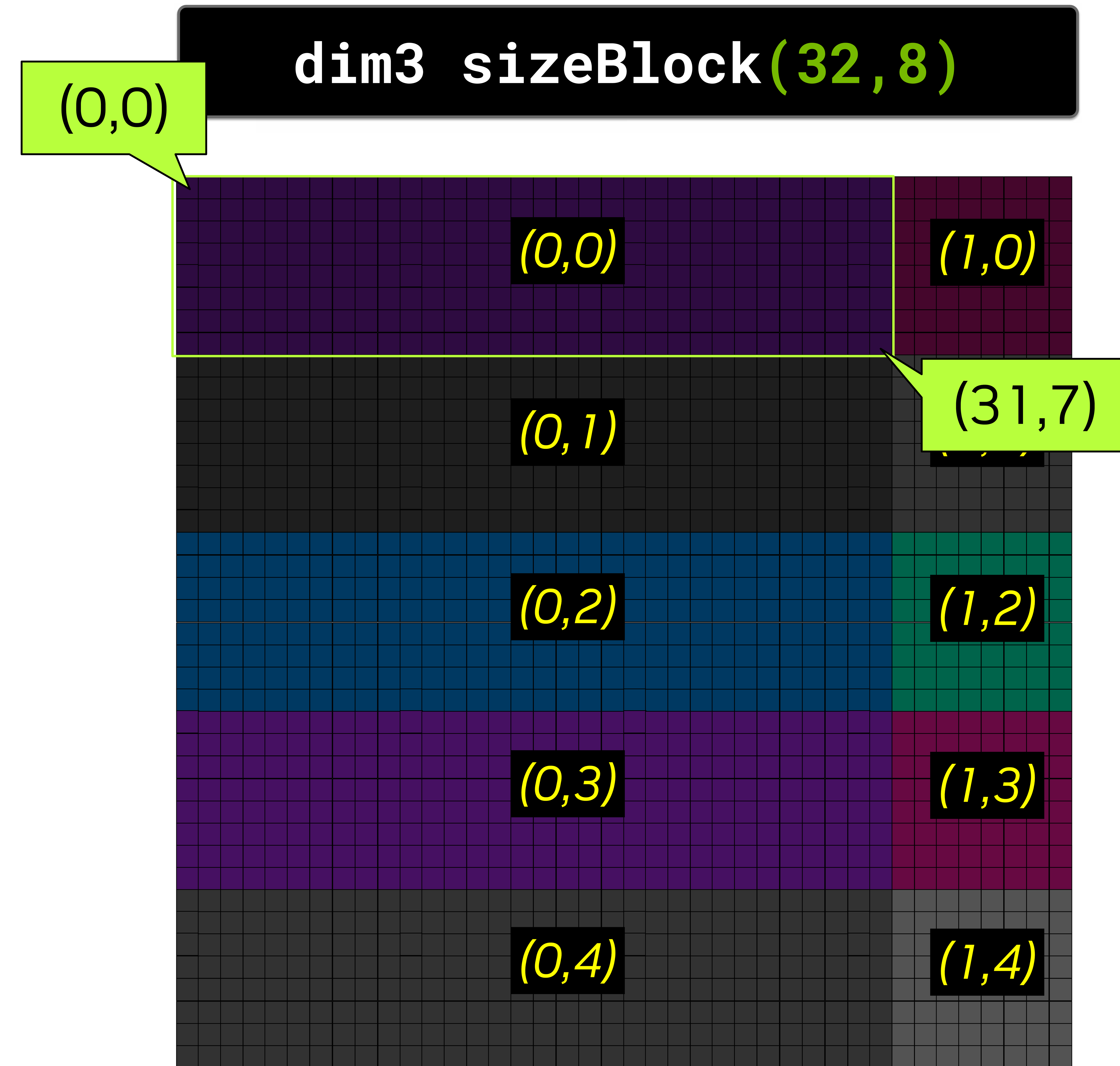
ブロックサイズ (x,y)

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



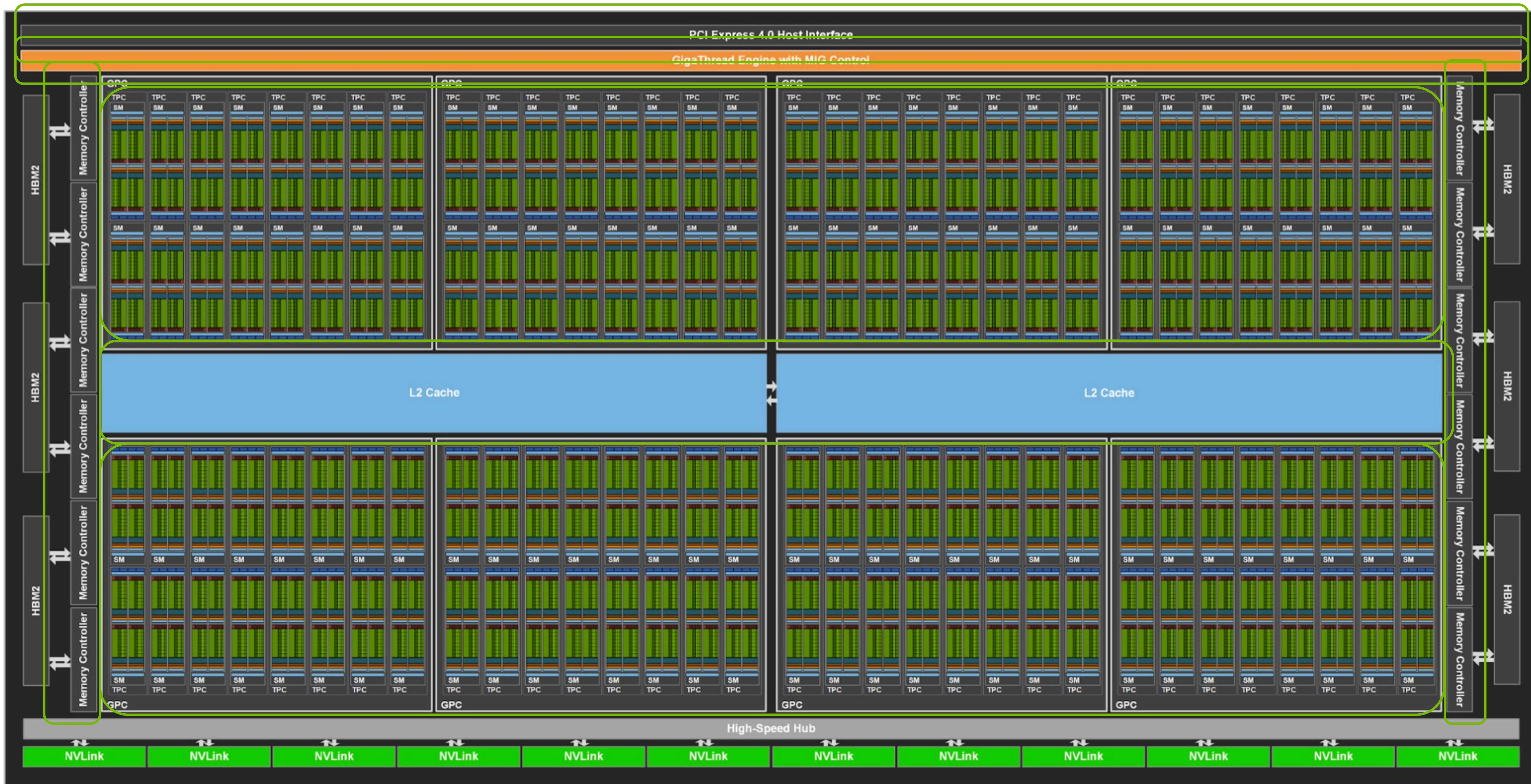


ブロック ID (blockIdx) スレッド ID (threadIdx)

NVIDIA.

- 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

NVIDIA.

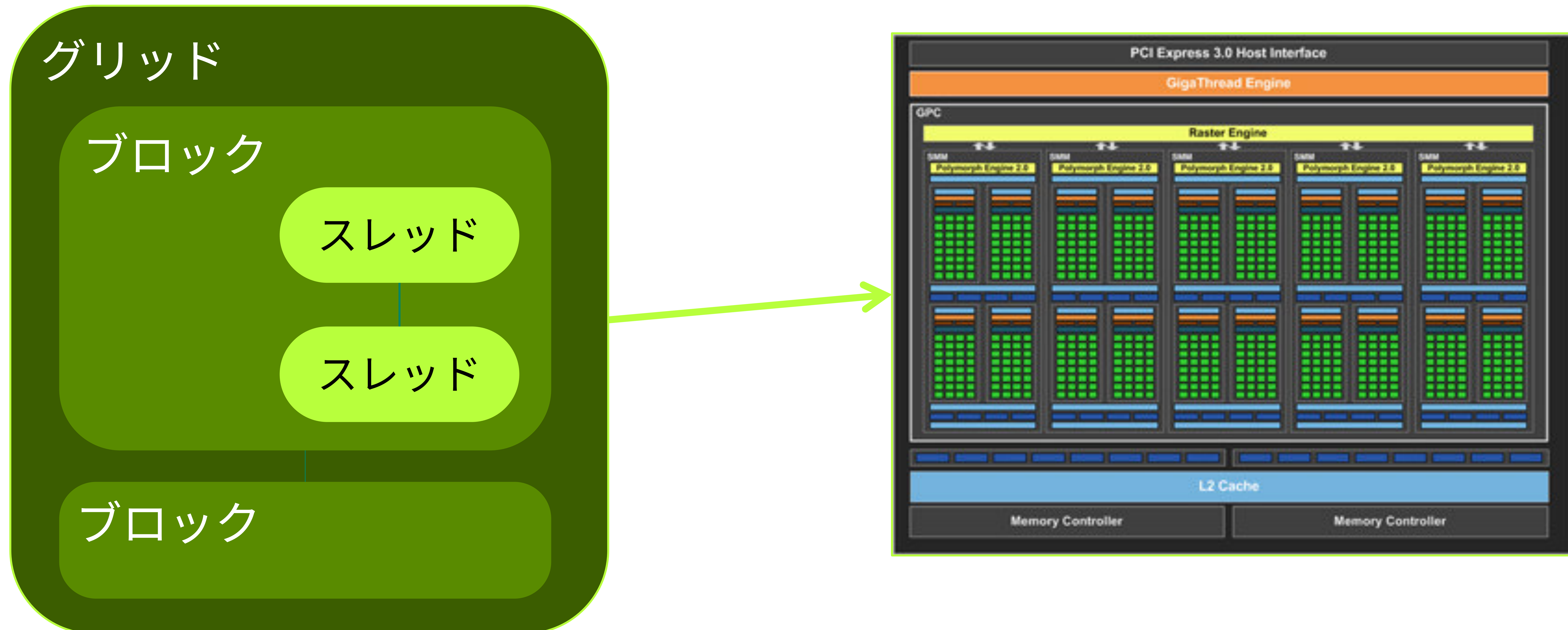
SM (Streaming Multi-processor)

- 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

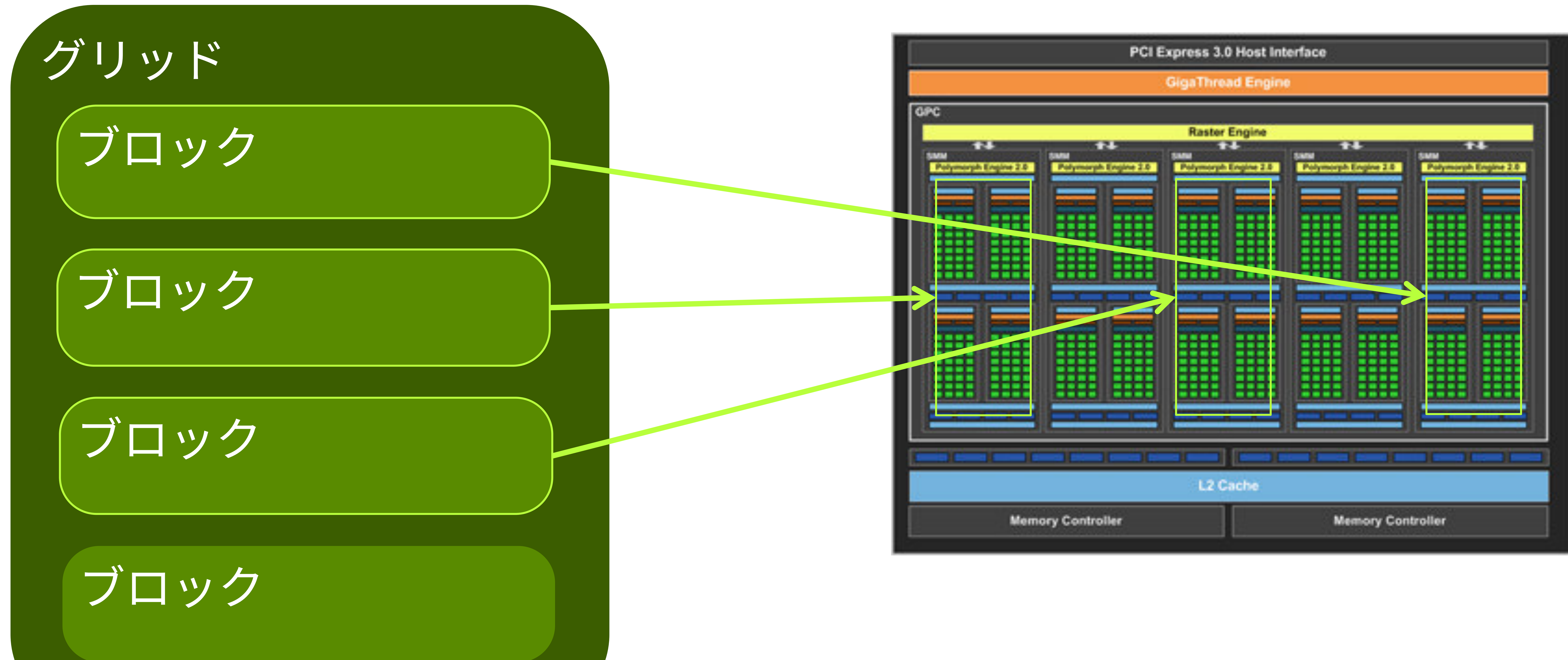
GPU カーネル実行の流れ

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



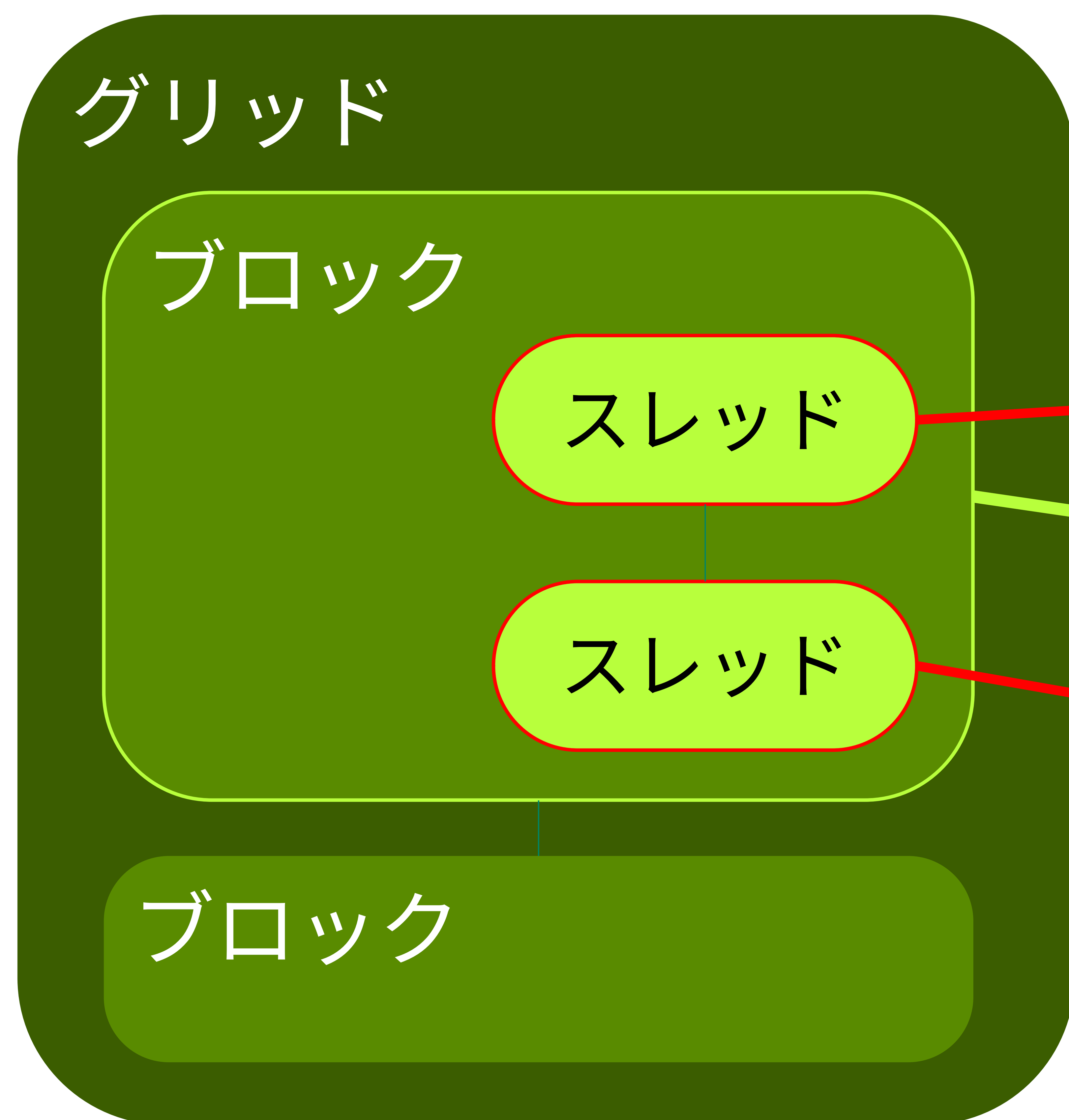
ブロックを SM に割り当て

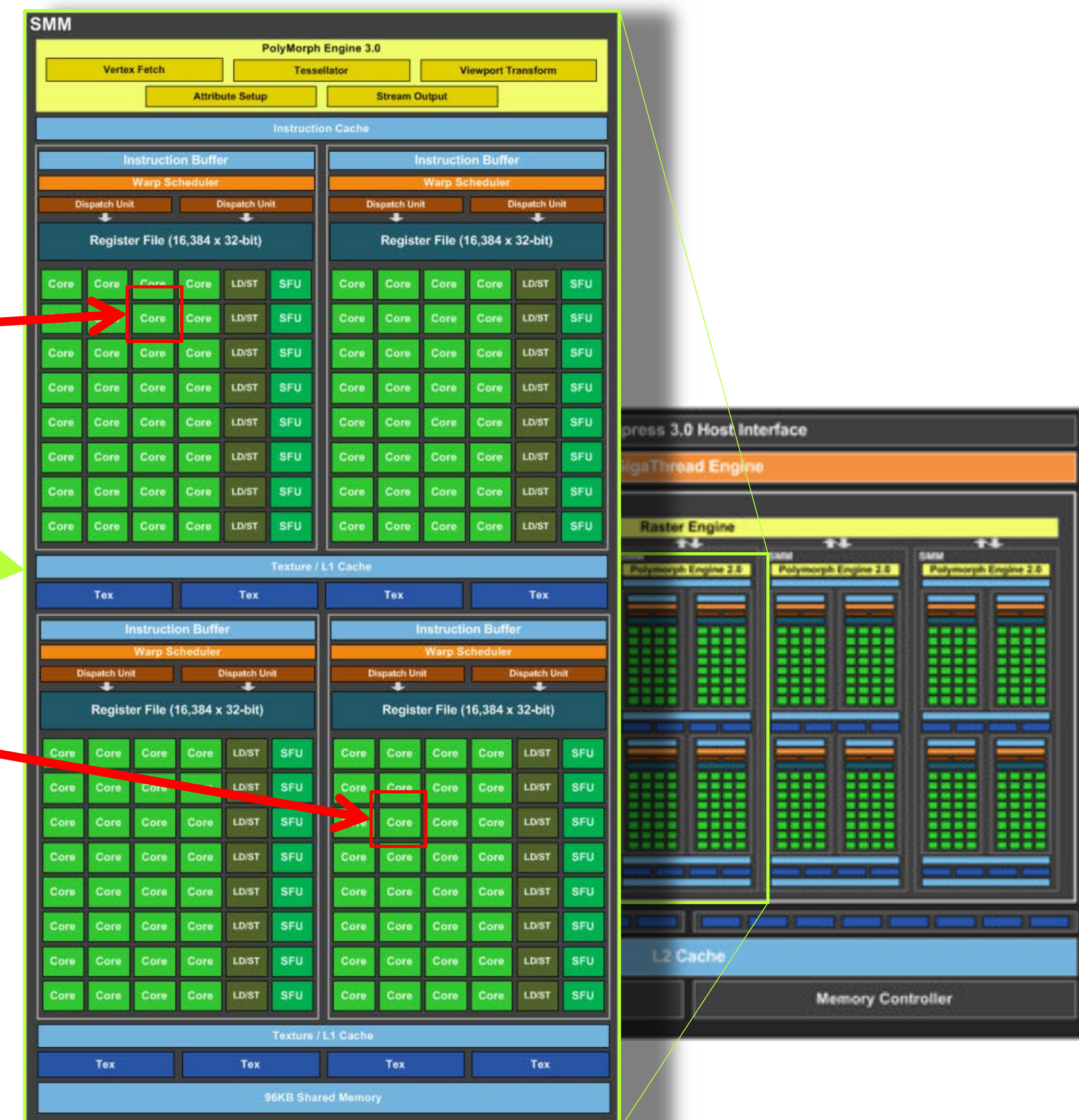
- 各ブロックは、互いに独立に実行
  - ブロック間では同期しない、実行順序の保証なし
- 1 つのブロックは複数 SM にまたがらない
  - 1 つの SM に複数ブロックが割り当てされることはある



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~~

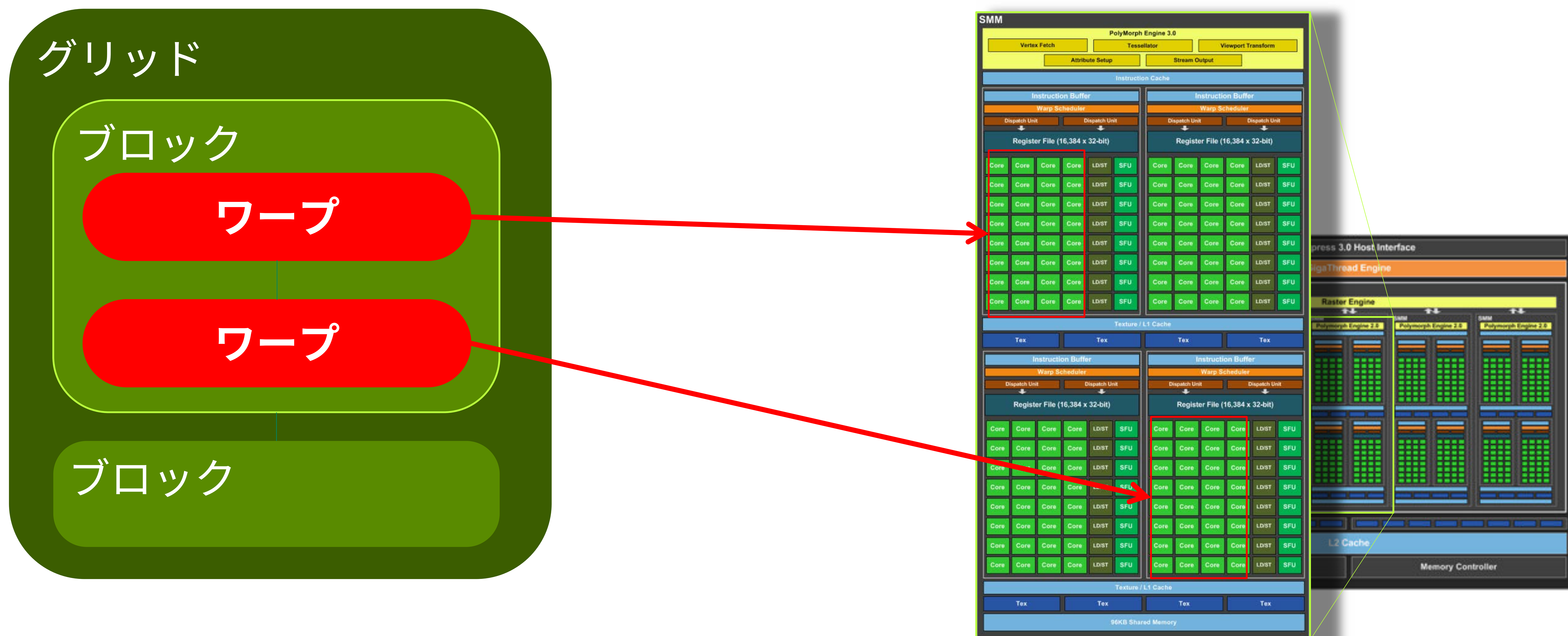




NVIDIA

GPU カーネル実行の流れ

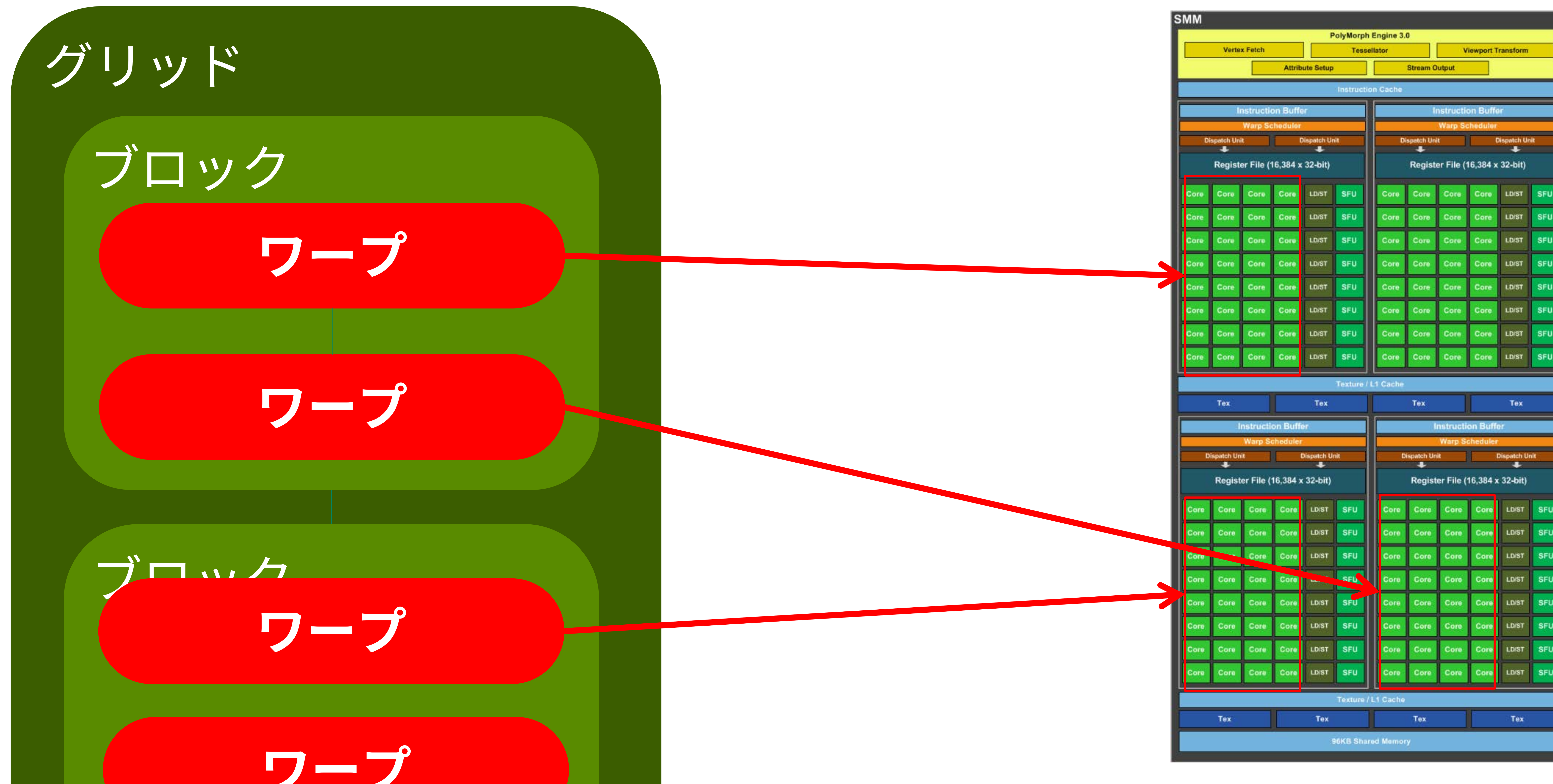
- SM 内のスケジューラが、**ワープ**を CUDA コアに投入
  - ワープ: 32 スレッドの塊
  - ブロックをワープに分割、実行可能なワープを、空 CUDA コアに割り当てる



ワープの CUDA コアへの割り当て

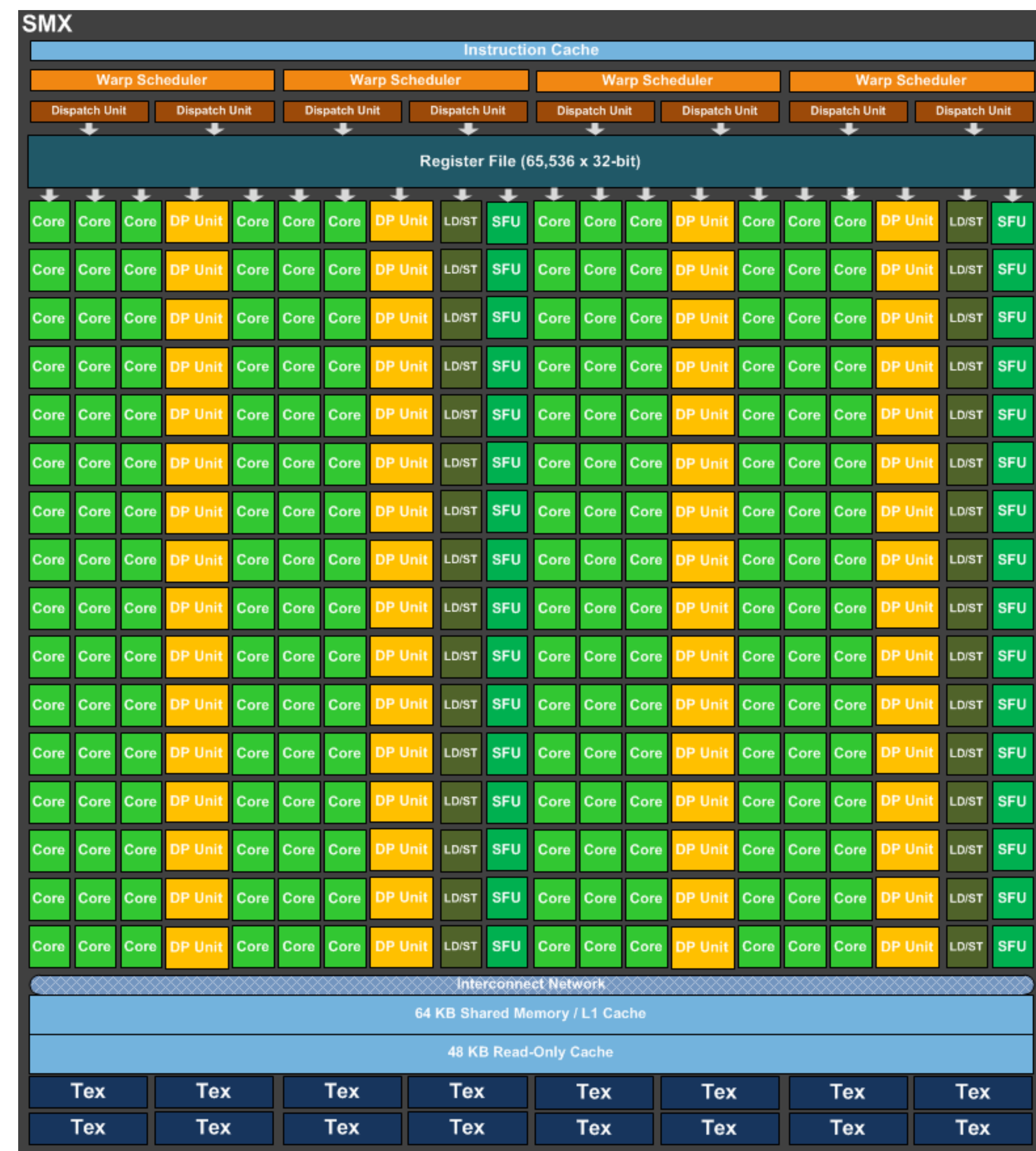
- ワープ内の 32 スレッドは、同じ命令を同期して実行
- 各ワープは、互いに独立して実行
  - 同じブロック内のワープは、明示的に同期可能 (__syncthreads())

SIMT
(Single Instruction Multiple Threads)

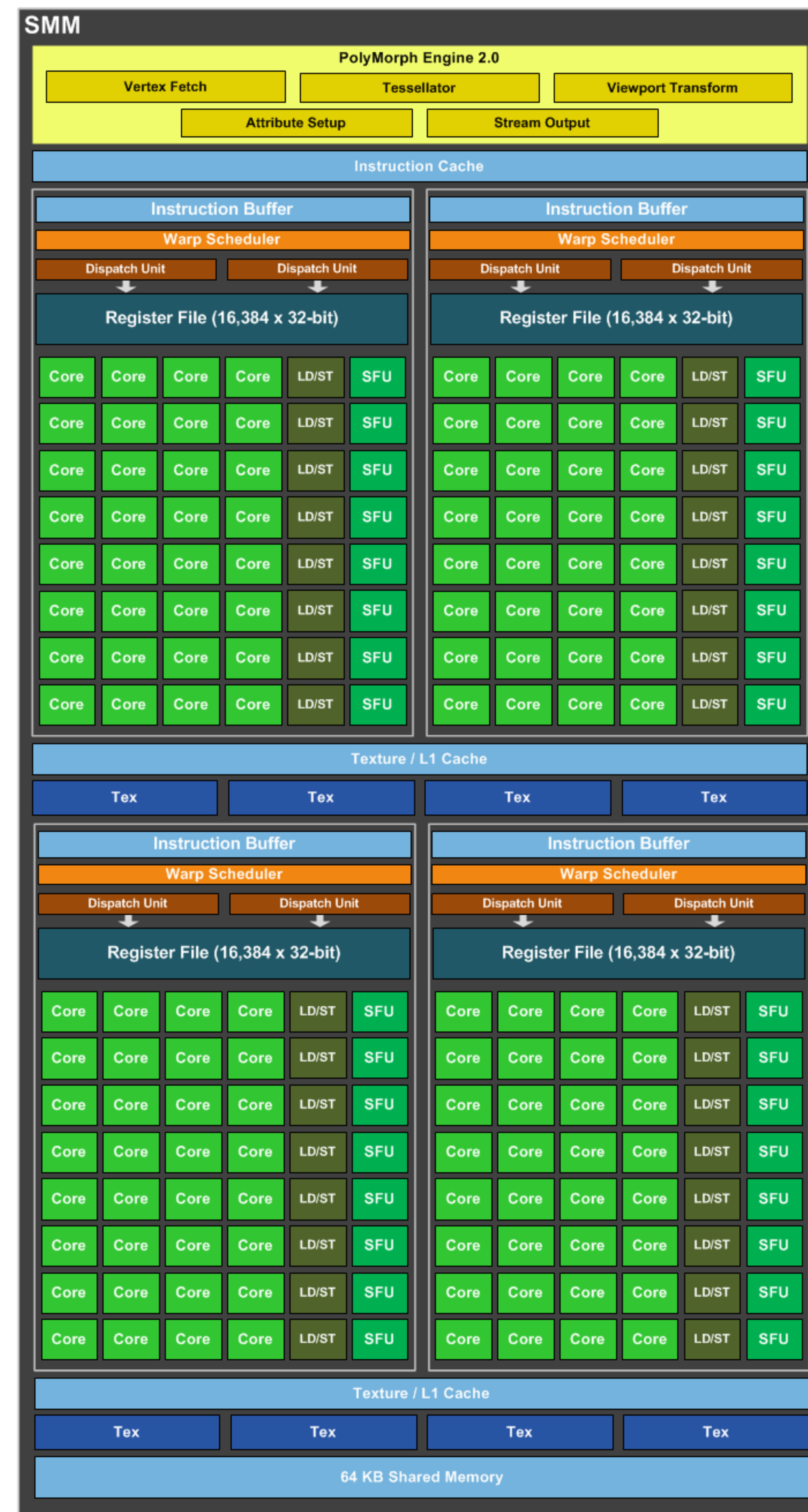


NVIDIA

GPU アーキの変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

NVIDIA