HPC&IP

GPGPU Computing

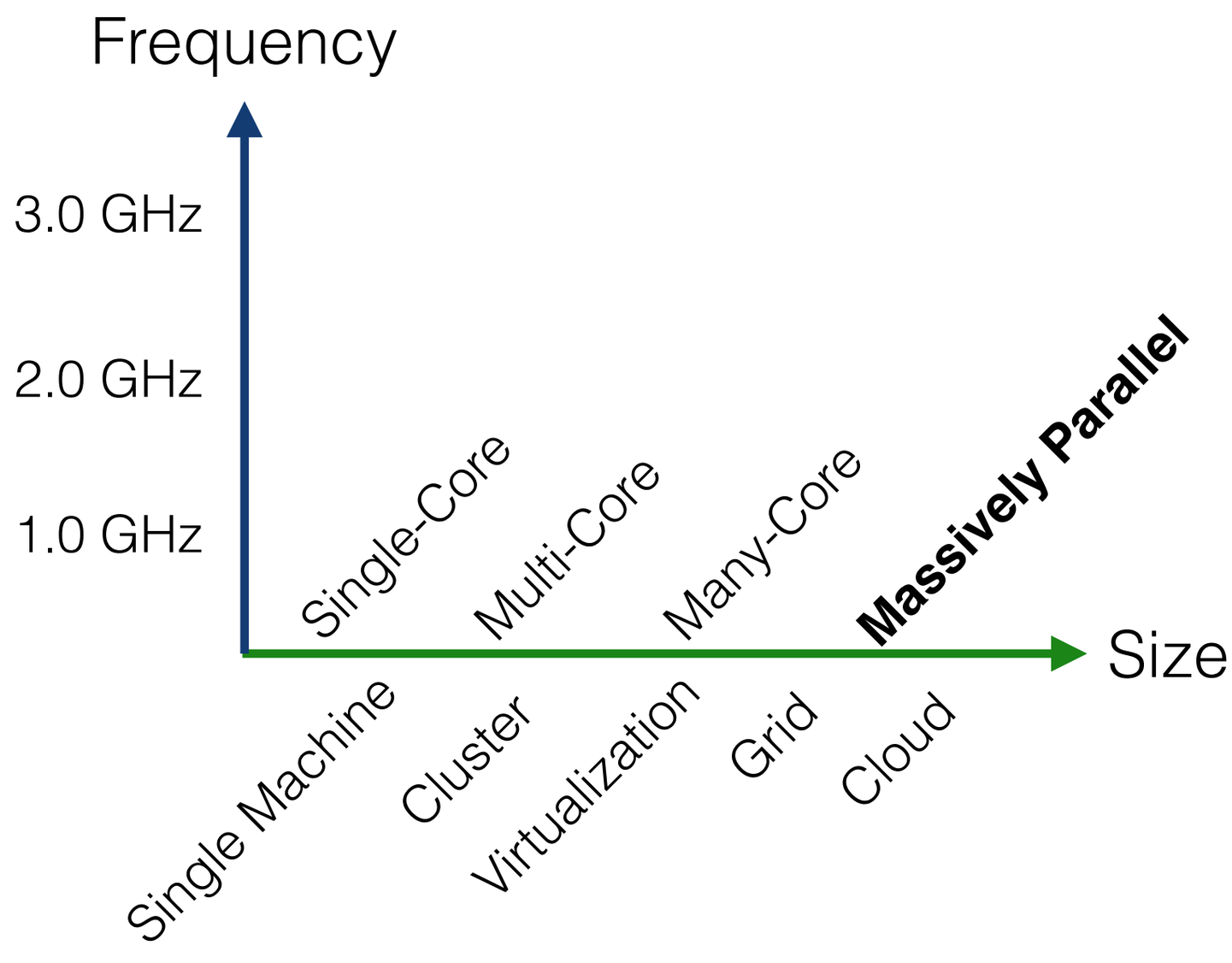
introduction lecture

Jeffrey J. Yu
HPC&IP Laboratory

Agenda

- Basic Concepts
- How to Use
- GPU Generations
- Architectures
- Layers in Studying
- Future Guess

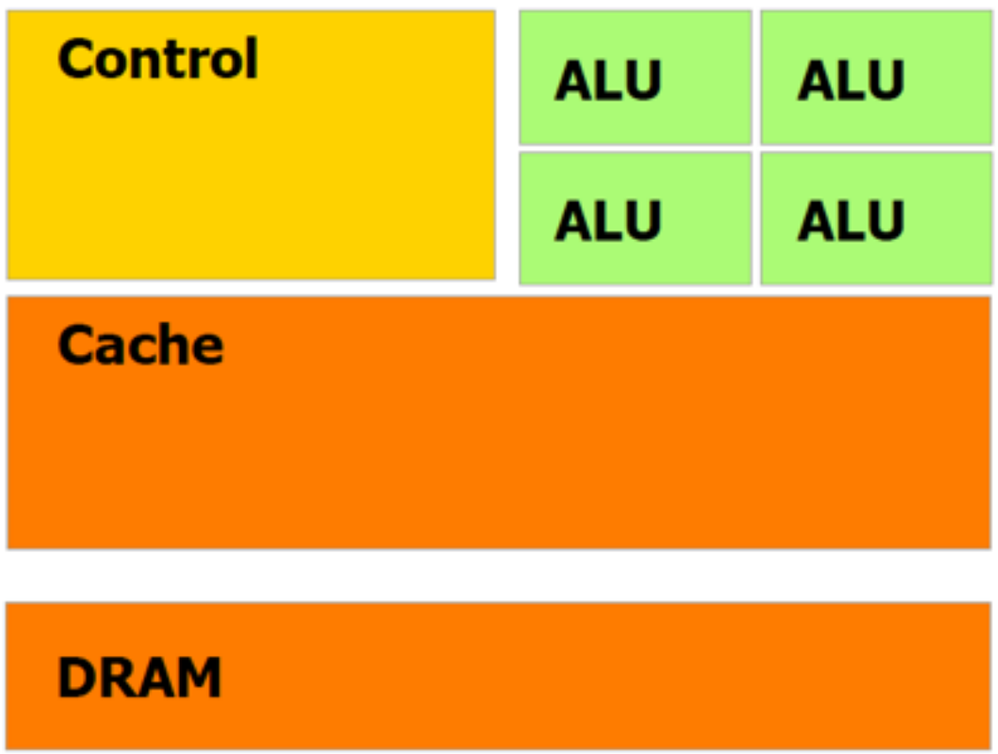
Introduction



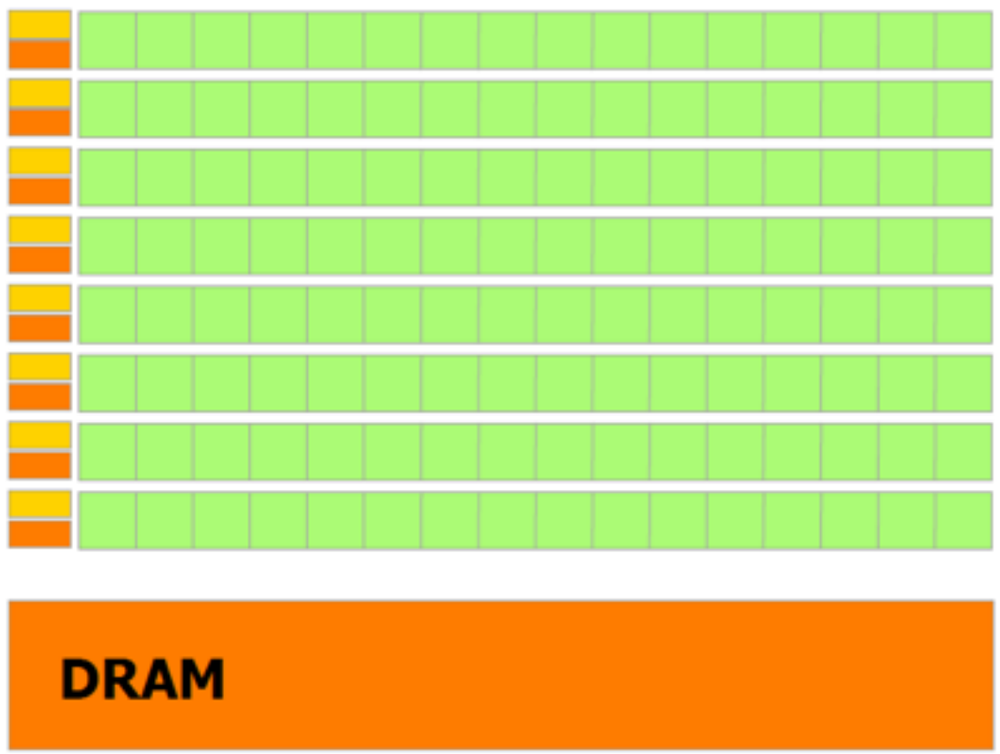
Basic Concepts

- What is GPGPU?
  - General Purposed Graphic Processing Unit
- Why it?
  - different architecture makes GPGPU performing extremely better than CPU with much less cost.
  - a heterogeneous computing example

Basic Concepts

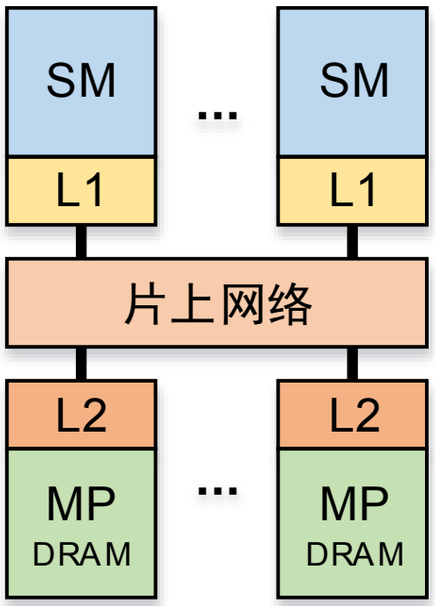


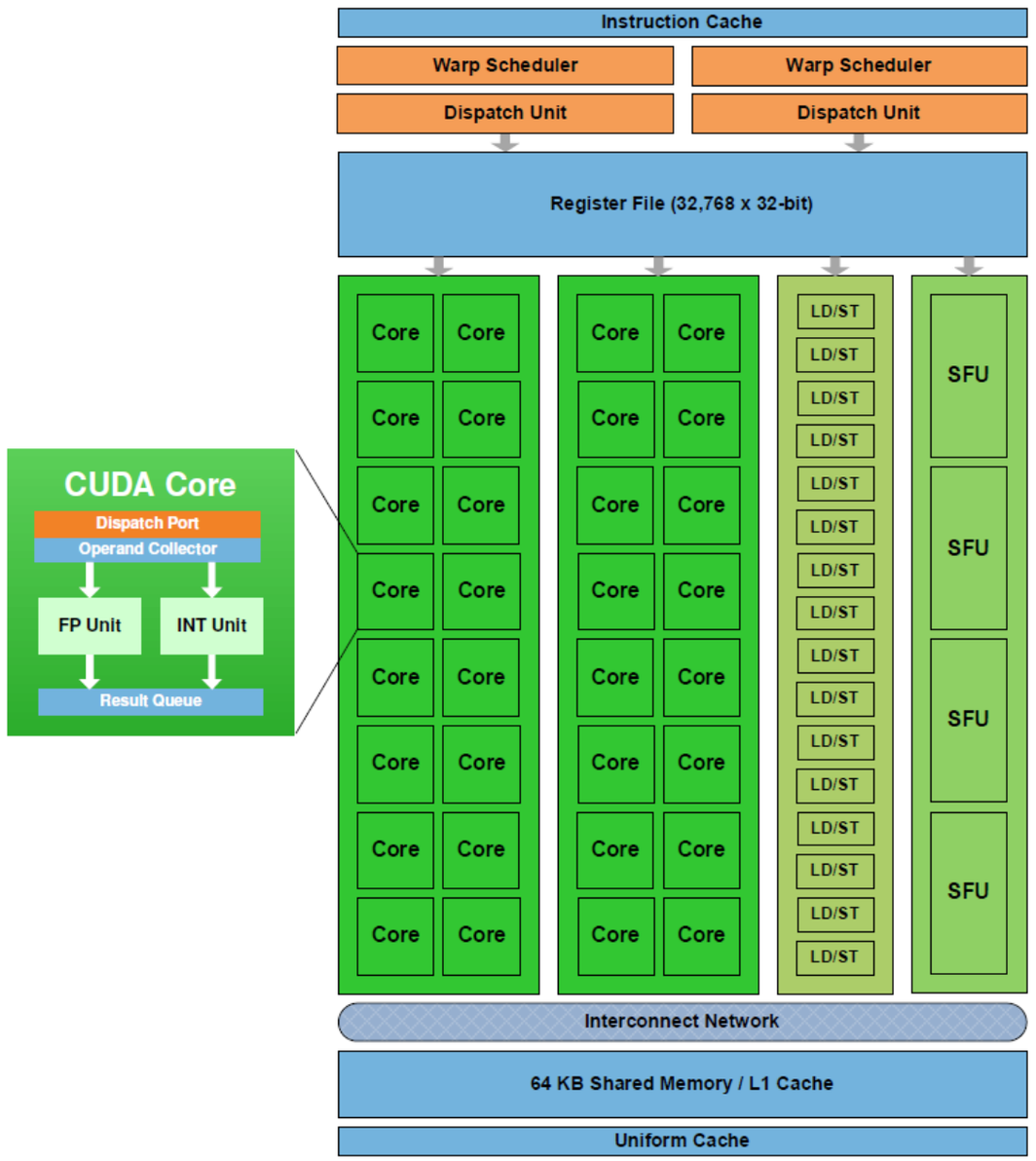
CPU



GPU

Basic Concepts

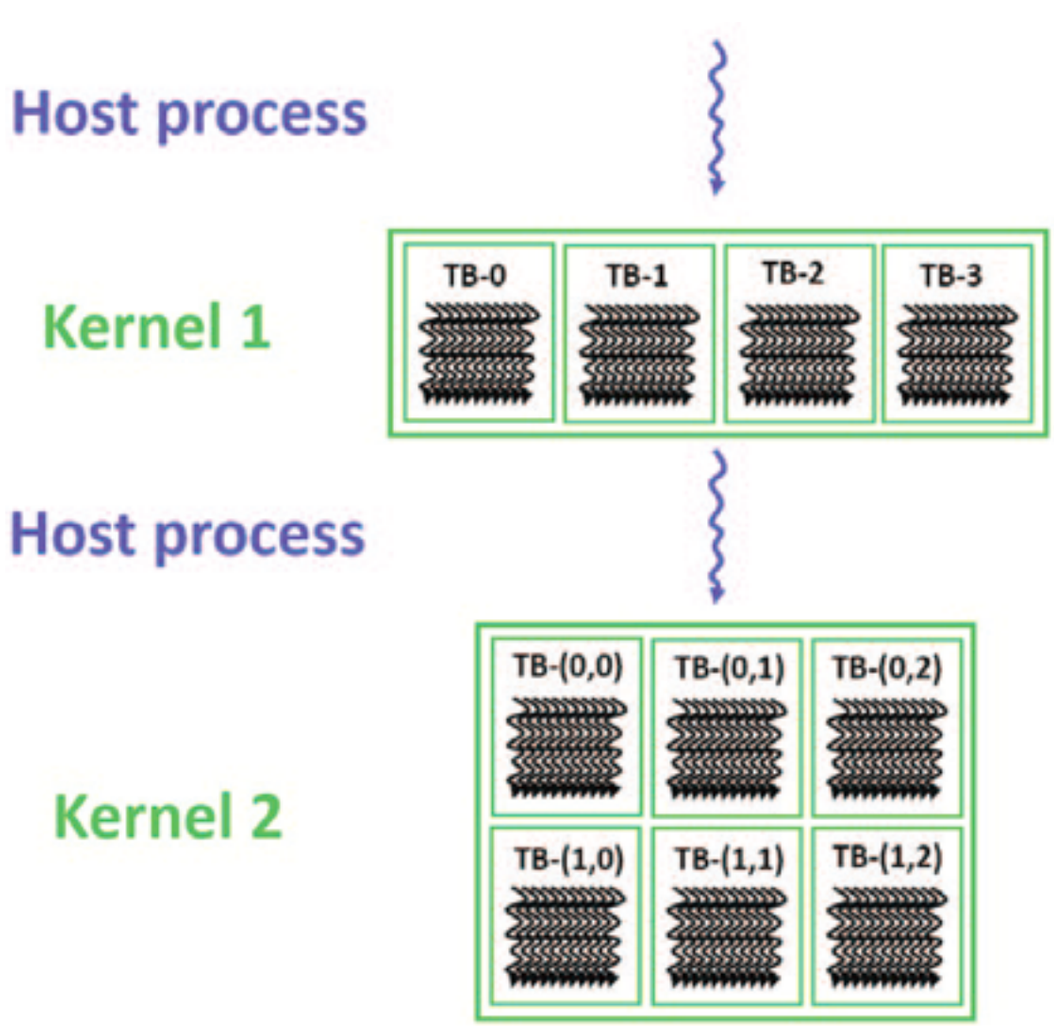


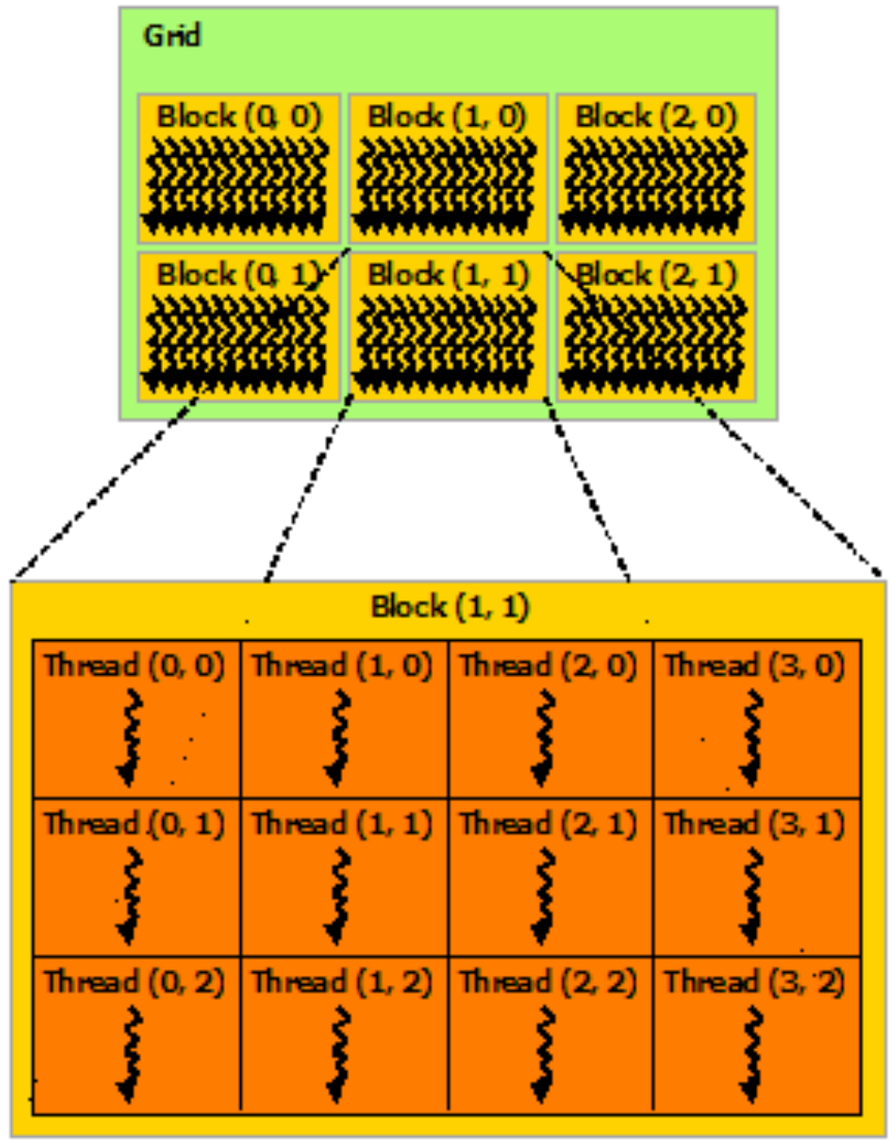


Fermi Streaming Multiprocessor (SM)

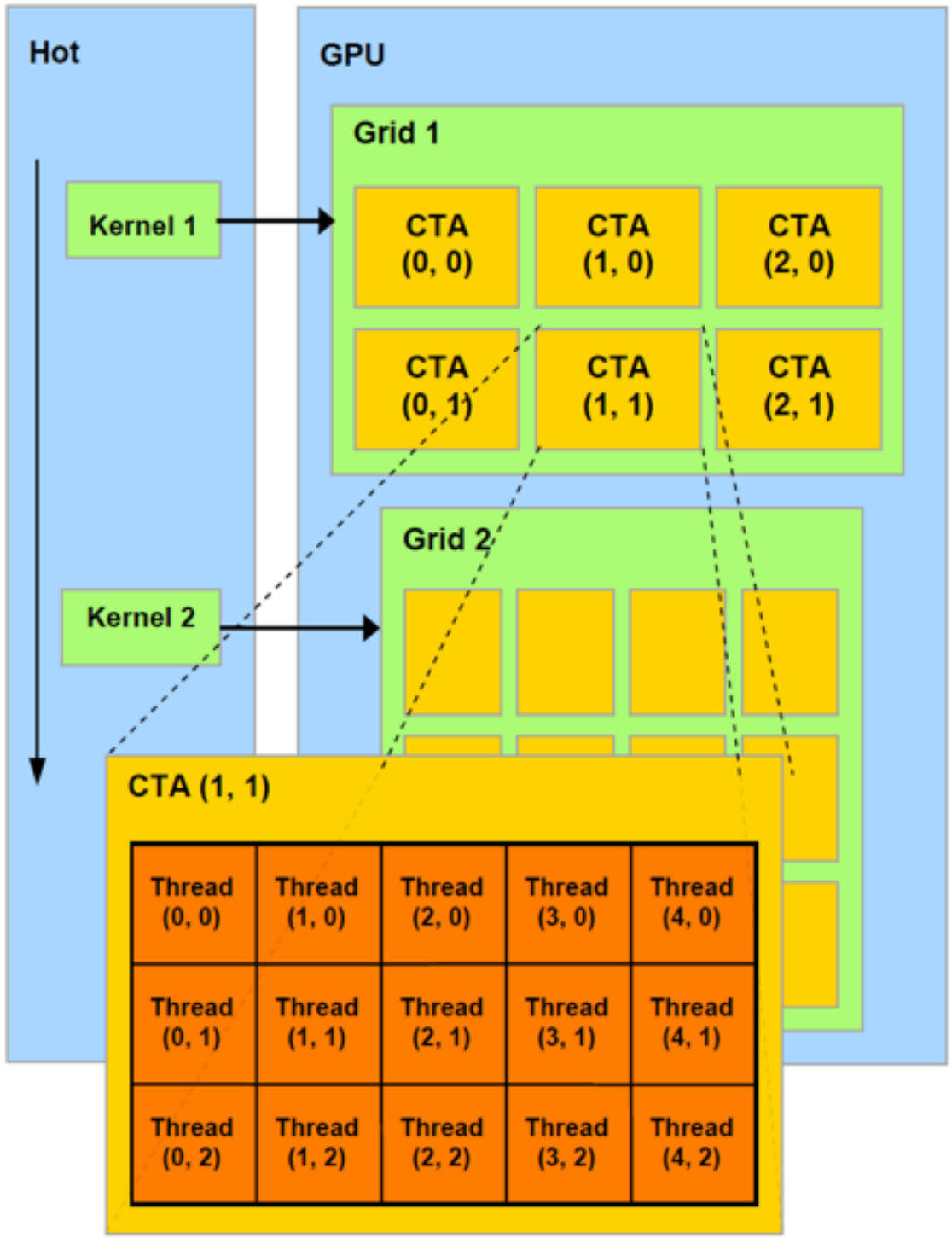
How to Use

- OpenCL vs CUDA





How to Use


Hot
GPU
Grid 1
Kernel 1
CTA (0, 0)
CTA (1, 0)
CTA (2, 0)
CTA (0, 1)
CTA (1, 1)
CTA (2, 1)
Grid 2
Kernel 2
CTA (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (4, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (4, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)
Thread (4, 2)

How to Use

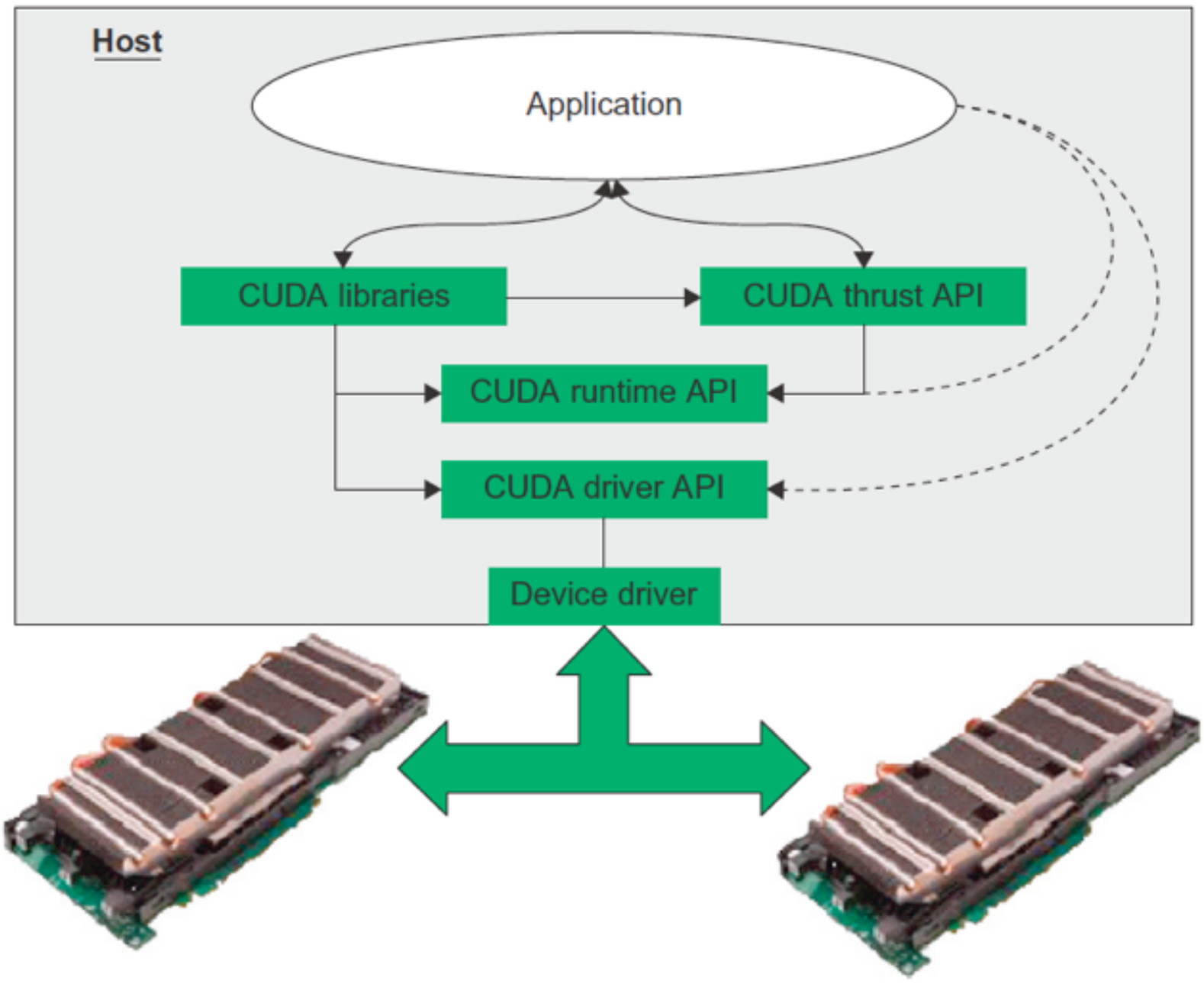
- CUDA Example

```
// Kernel definition
__global__ void MatAdd(
    float A[N][N], float B[N][N],
    float C[N][N])
{
  int i = threadIdx.x;
  int j = threadIdx.y;
  C[i][j] = A[i][j] + B[i][j];
}

int main()
{
  ...
  // Kernel invocation with one block of N * N * 1 threads
  int numBlocks = 1;
  dim3 threadsPerBlock(N, N);
  MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
  ...
}
```

How to Use

- Installation



How to Use

- Software Interface
  - Driver API

```
cuModuleLoad(&cuModule, “MatAdd.ptx”);
cuModuleGetFunction(&matAdd, cuModule, “matAdd”);
cuLauchKernel(matAdd, blockSize, gridSize, param);
```

  - Runtime API

```
MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
```

  - Thrust API

```
fill(thrust::raw_pointer_cast(&a[0]), N);
sumA = thrust::reduce(a.begin(), a.end(), 0);
```

How to Use

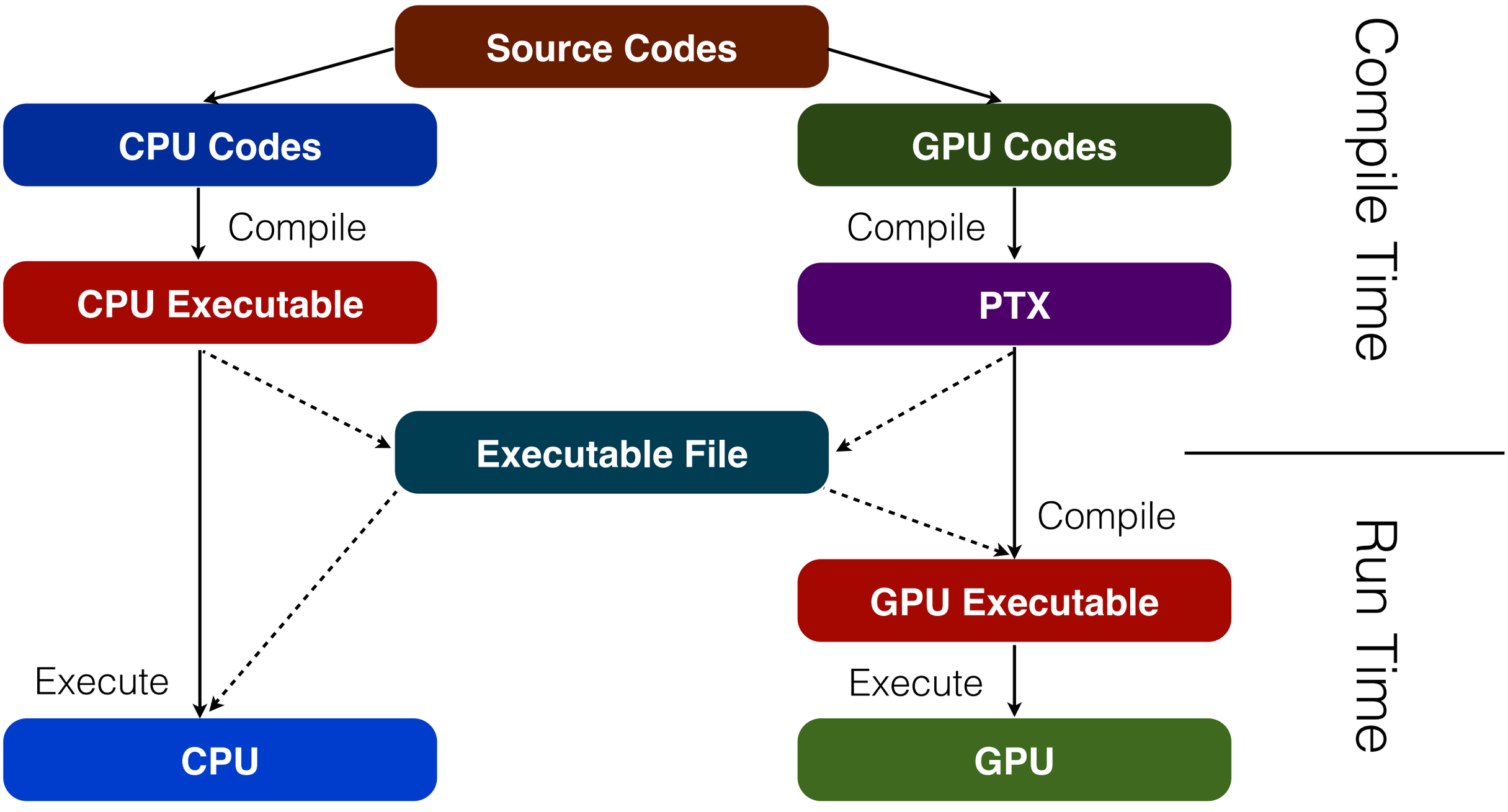
- Software Interface
  - **Kernel**: __global__
    Host Launches, Device Executes
    Immediate return after launching
  - **Host Function**: __host__
    Normal function, Host launches and executes
  - **Device Subprogram**: __device__
    Device launches and executes, run as an inline function.

How to Use

- Software Interface
  - **Host Interface**
    - Memory Operation and Data Transfer
    - Device Setup and Query
    - Profiling and Synchronization
  - **Device Interface**
    - Mathematics
    - Synchronization & Atomic
    - Data R/W in different memory

How to Use

- Compile

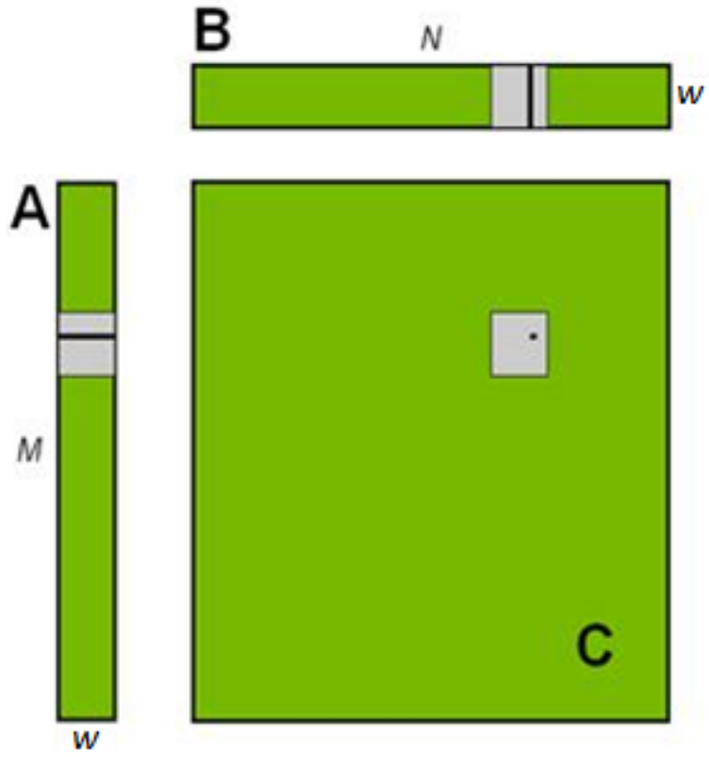


Programming Structure

- Create handlers and allocate memory resources
- Data transfer from host to device
- Launch the kernels
- Data transfer from device to host
- Release or depose the used resources

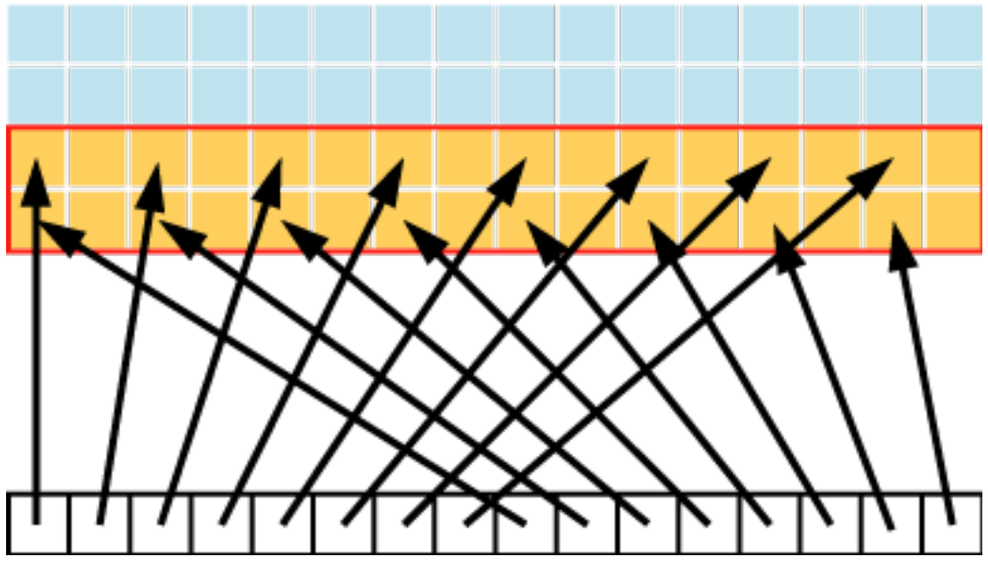
GPGPU Memory

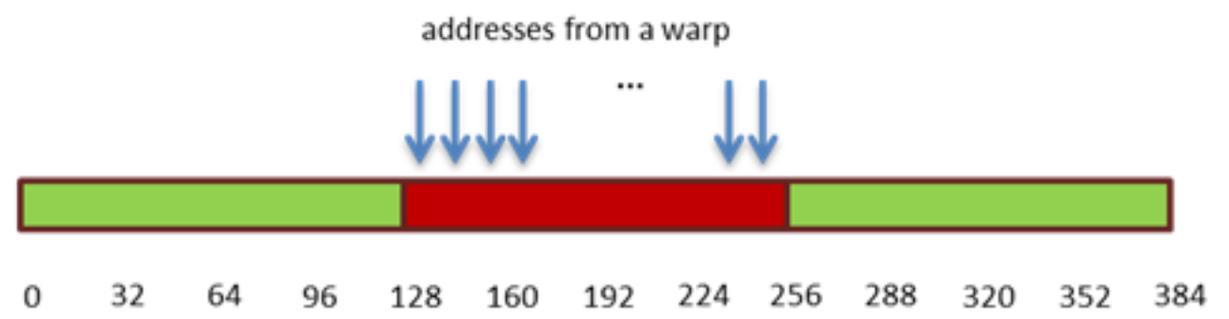
- General Memory
  - **Global Memory**
  - **L2/L1 Cache**
  - **Shared Memory**
  - **Register / Local Memory**

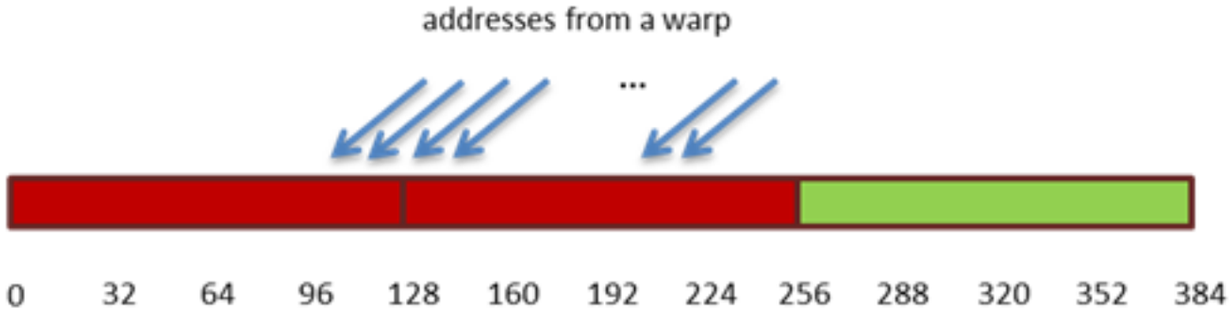


GPGPU Memory

- Coalesced Memory Access
  - **Continuous memory requests are combined and processed.**

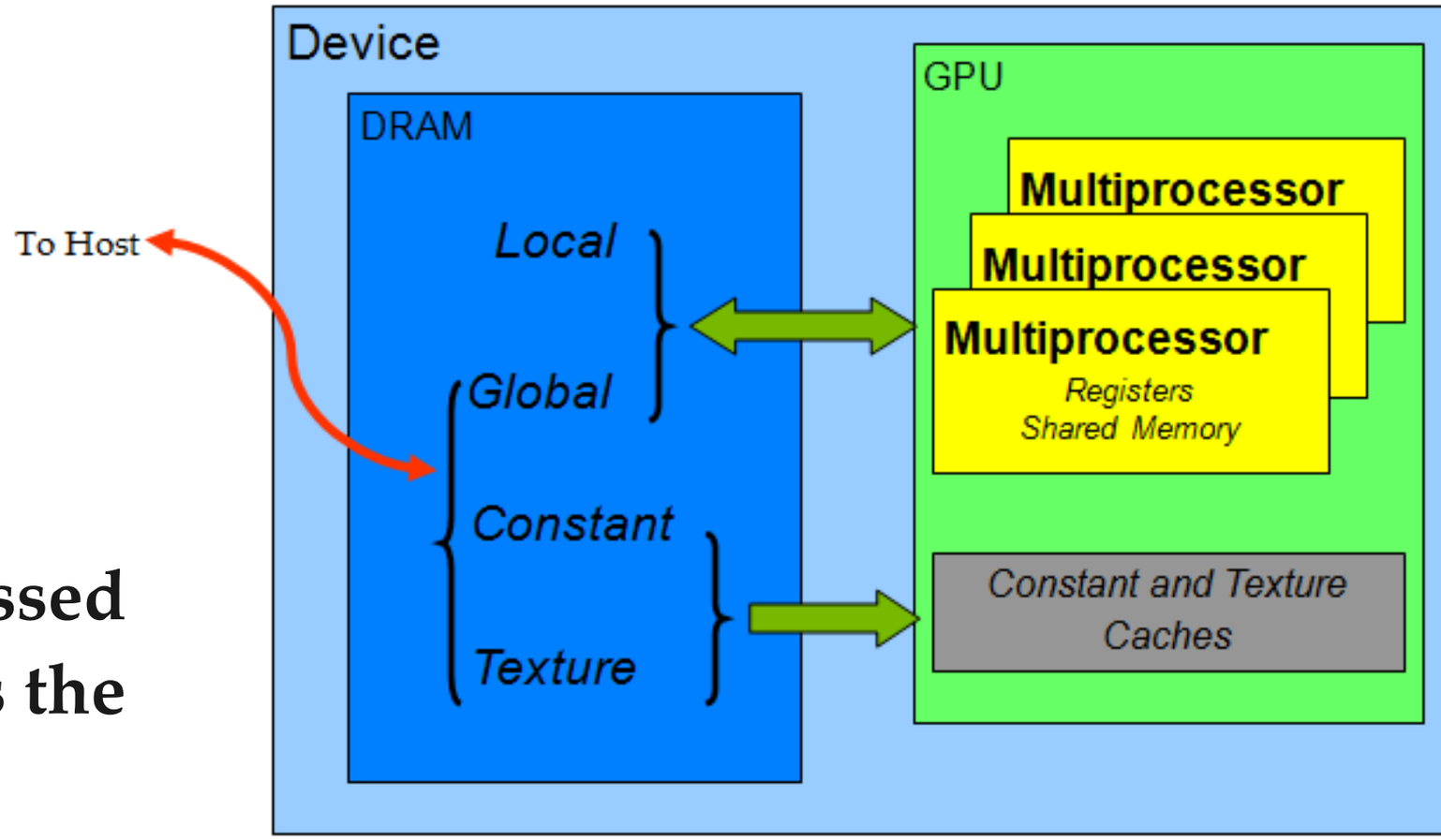






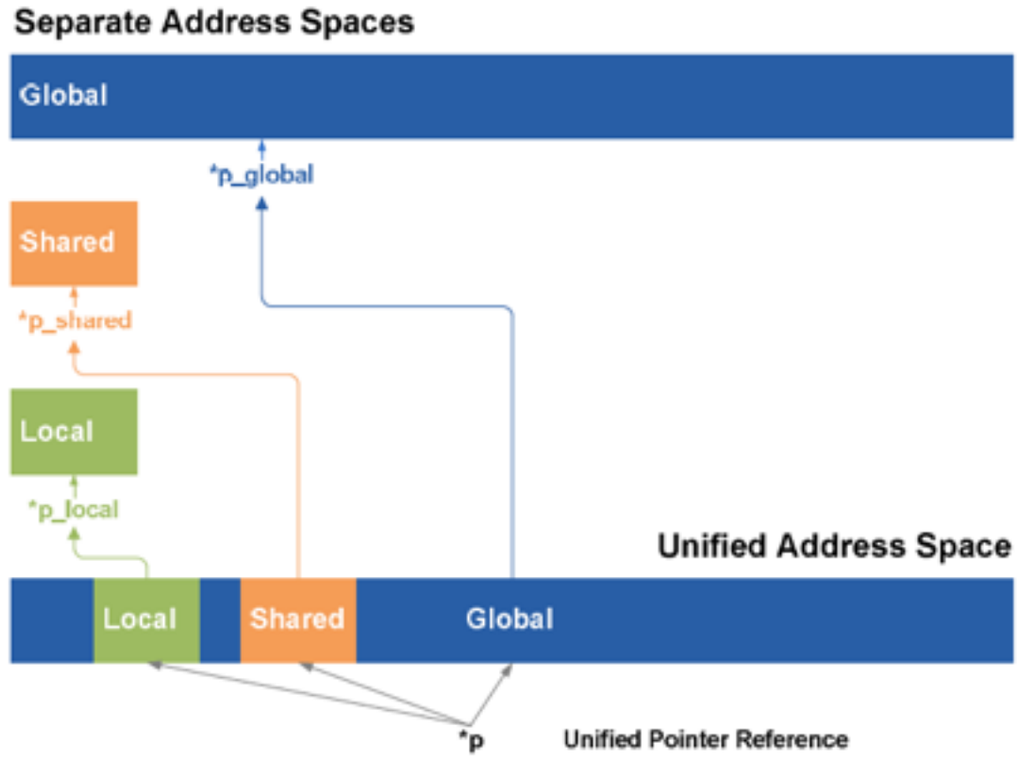
GPGPU Memory

- Specific Memory
  - **Texture Memory**
  - **Constant Memory**
  - **Surface Memory**
- **These memory are accessed used a data path bypass the general cache.**



GPGPU Memory

- Host Memory
  - **Dynamic Memory from malloc**
  - **Page-locked Memory**
  - **Write-combined Memory**
  - **Mapped Memory**
  - **Unified Virtual Address Space**



Streams & Events

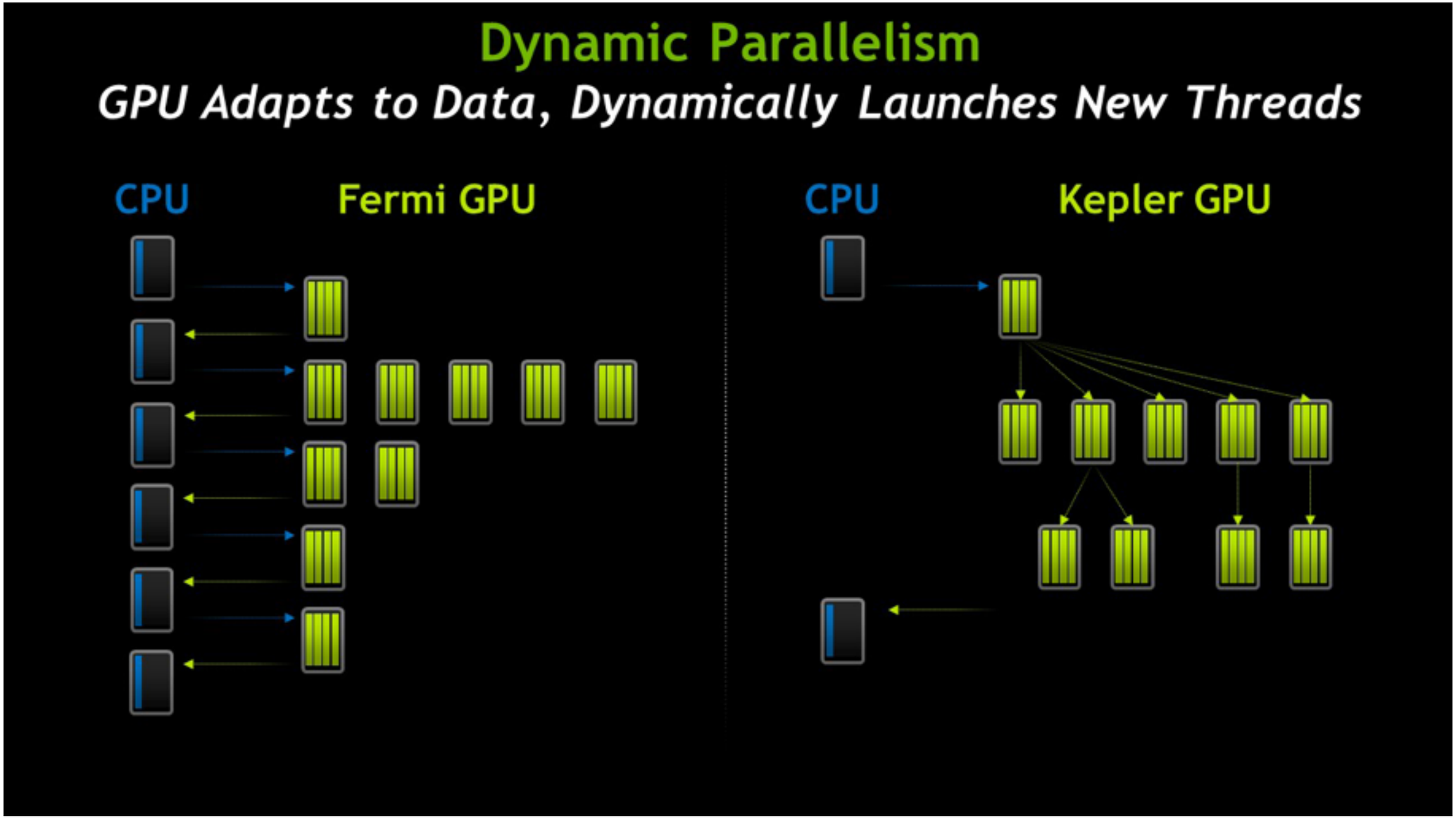
- Streams: Concurrent computing

- Events: Time stamps
  - **Calculate the elapsed time**
  - **Give an anchor to synchronize**

Synchronizations

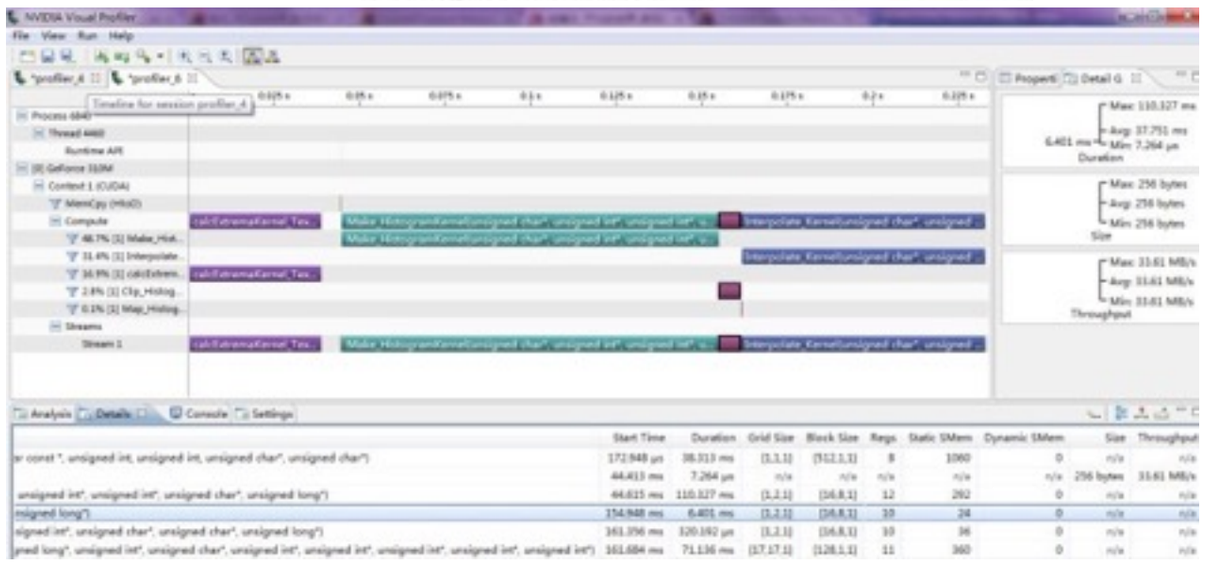
- Device codes
  - **__syncthreads**
- Host codes
  - **cudaDeviceSynchronize**
  - **cudaStreamSynchronize**
  - **cudaEventSynchronize**

Dynamic Parallelism



Tools for Developments

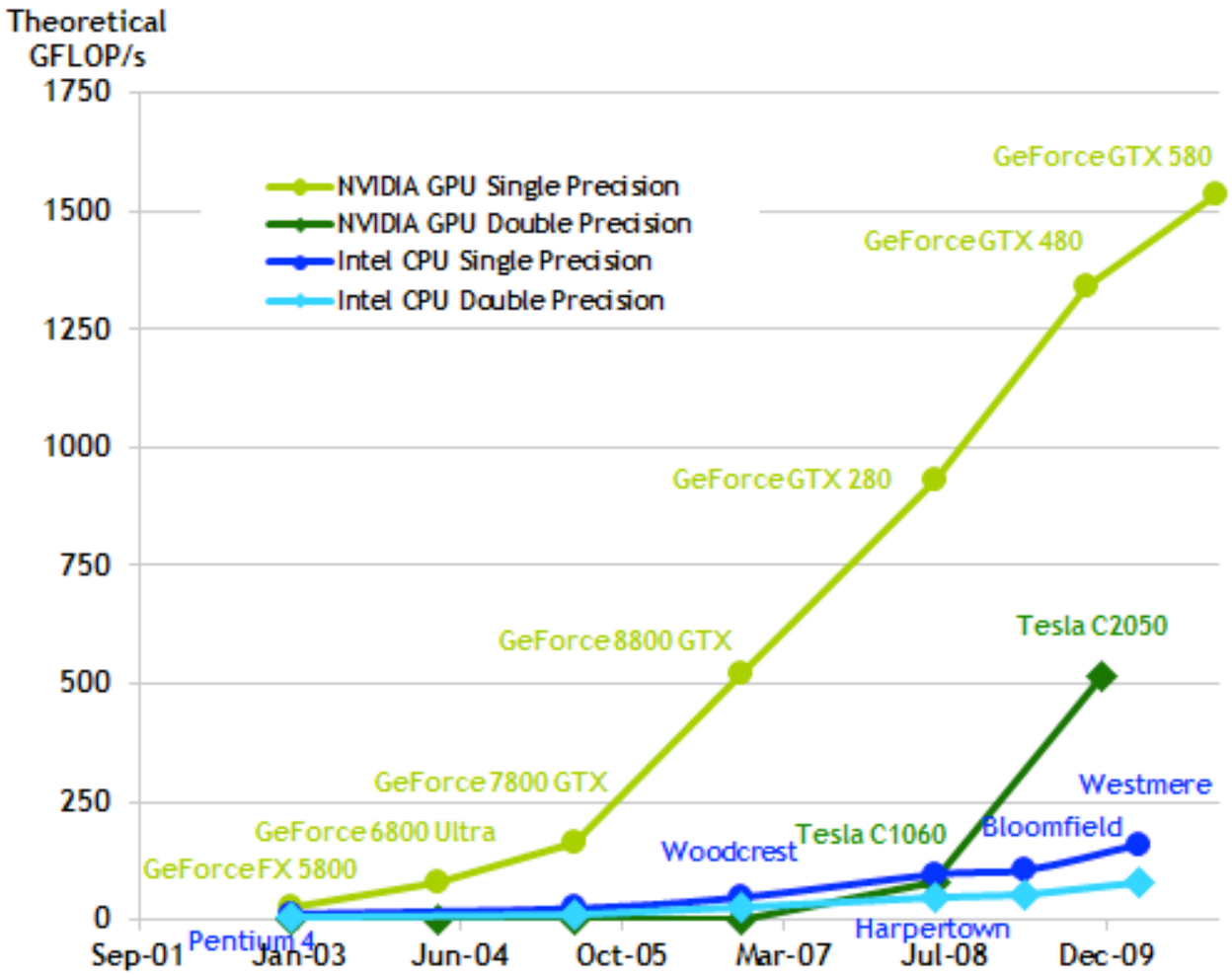
- Debugger
  - **cuda-gdb**
- Profiler
  - **NVIDIA Visual Profiler**
- Others



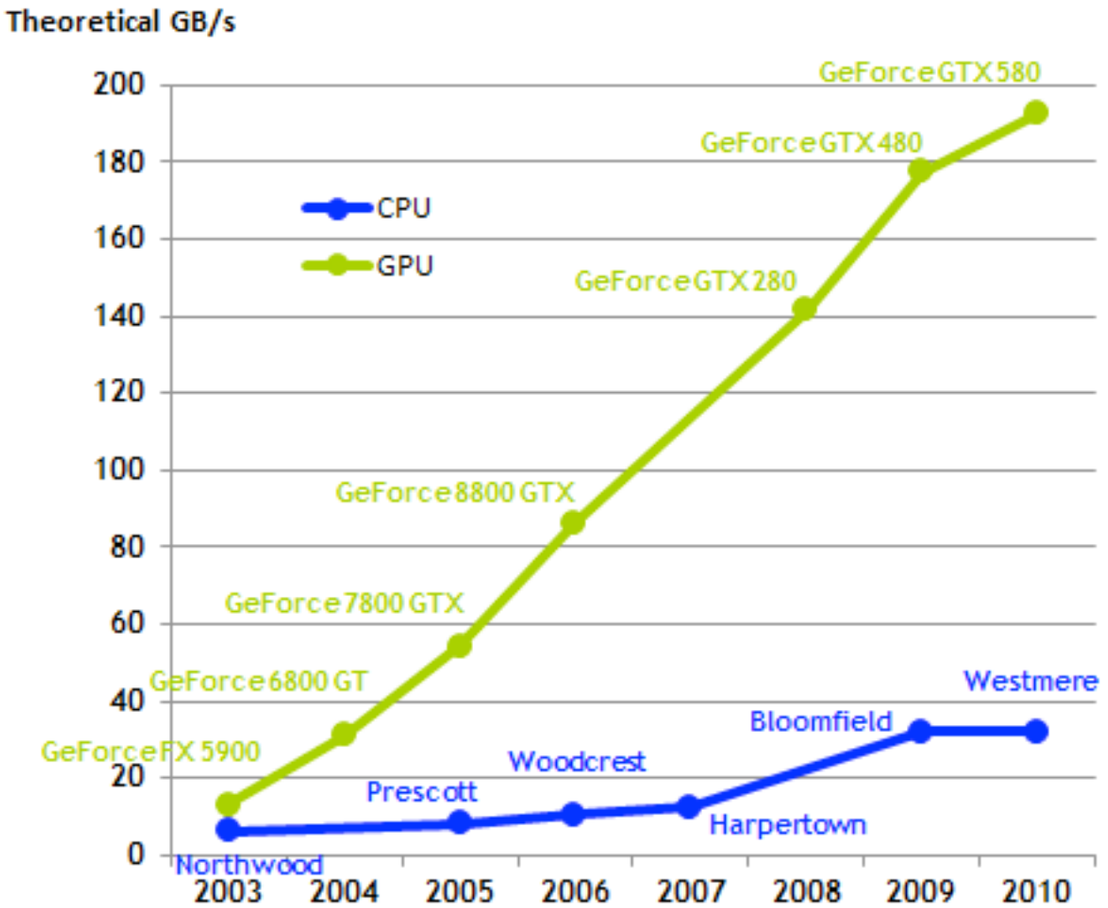
GPU Generations

- Before 1991, CPU performs computing
- 1991-2001, GPU with fix capability
- 2001-2006, Programmable GPU
- 2006-Today, GPGPU, OpenGL
- Future, Heterogeneous co-processor
- Flexibility, Programmability, Generalization

GPU Generations



Floating-Point Operation



Memory Bandwidth

GPU Generations

Tesla (cc *1.x*)

Texture Memory
Constant Memory

No Cache

Mapped Memory

Fixed Shared Memory

3D Block
2D Grid

Fermi (cc 2.*x*)

Surface Memory

2 Level Cache

UVA

Configurable Shared Memory

3D Block & Grid
Recursive Execution

Print in Kernel
Debug
Multi-Kernel Execution

Kepler (cc 3.5)

Larger Memory Resource Size

More Cores in SM

Dynamic Parallelism
Shuffle Instruction

Hyper-Q
(Multiple work queue)

culib Library

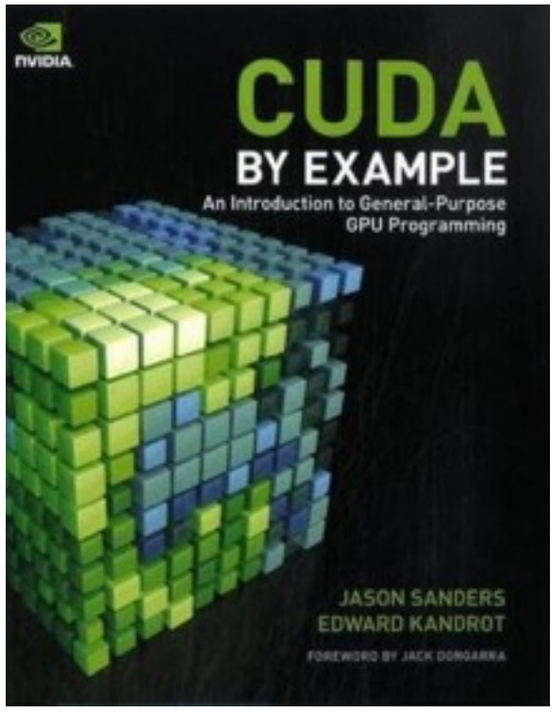
CUDA vs OpenCL

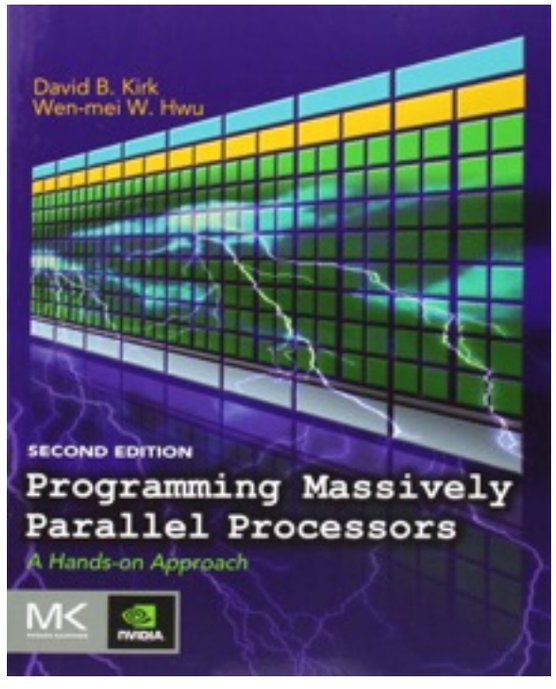
- Almost the same in programming model
- CUDA is only supported by NVIDIA. OpenCL is an open standard.
  - **Embedded systems**
- CUDA provides more features due to the target hardware.

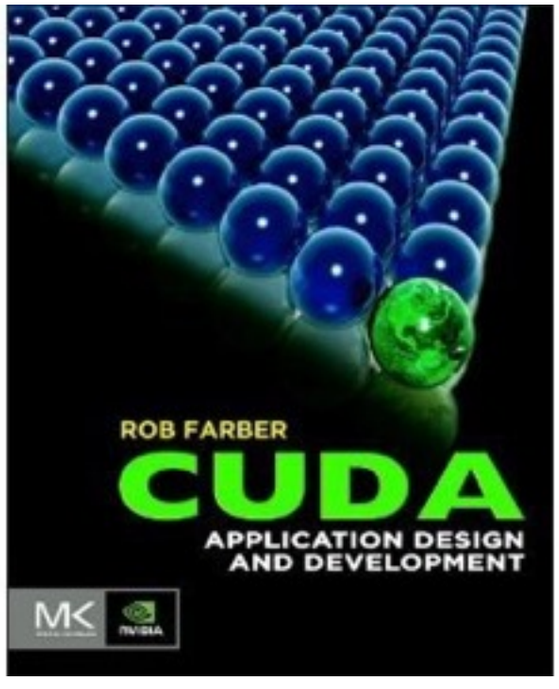
Useful Resources

- CUDAZone:
  https://developer.nvidia.com/cuda-zone
- GPGPU
  http://gpgpu.org/

Useful Resources







Architectures

- Levels of shared resources with host
  - GPUs in cluster (network)
    - independent graphic card / chip (host)
    - integral graphic chip (memory)
    - fusion CPU-GPU chip (cache)

Layers in Studying

- Application Layer
- Programming Layer
- Platform Layer
- Architecture Layer

- PS: Layer boundaries are not so clear

Application Layer

- Algorithm implementations on GPGPU
  - any algorithms
  - many strategies for specific algorithms
    - need to be summarized rather than case by case
- Examples:
  - Earthquake simulation [ES]
  - GPU Gems: a book with CUDA practices

[ES] Unat, D., Zhou, J., Gui, Y., Baden, S. B., & Cai, X. (2012). Accelerating a 3D finite-difference earthquake simulation with a C-to-CUDA translator. Computing in Science & Engineering, 14(3), 48–59. doi:10.1109/MCSE.2012.44

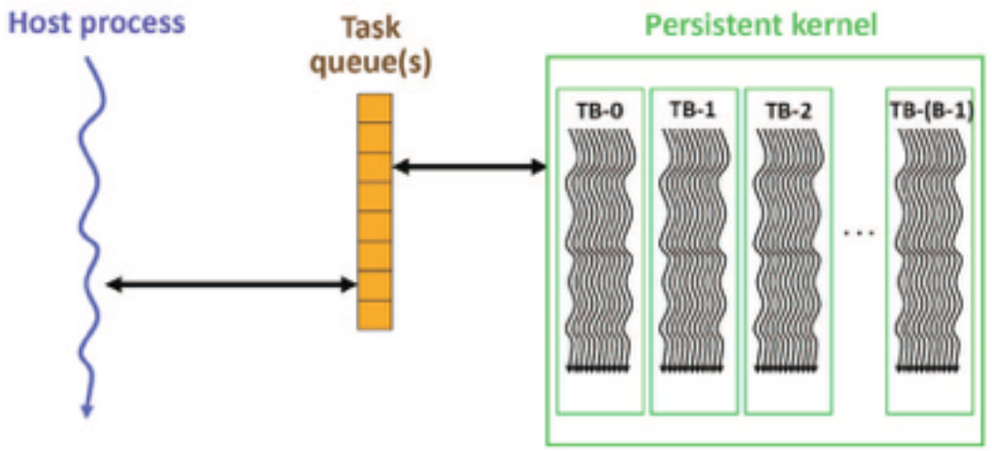
Programming Layer

- Programming Model
- Compiling System
- Instruction Set

Programming Layer

- Programming Model
  - how to express the parallel algorithm
  - model enhancements

- Examples:
  - Task queue paradigm[TQP]
  - Texture Pool[TPL]



[TQP] Chen L, Villa O, Krishnamoorthy S, et al. Dynamic Load Balancing on Single- and Multi-GPU Systems[C]. IEEE International Symposium on Parallel & Distributed Processing, 2010, 1-12.
[GCV] Allusse Y, Horain P, Agarwal A et al. GpuCV: an opensource GPU-accelerated framework forimage processing and computer vision[C]. Proc of the 16th ACM international conference on Multimedia, 2008: 1089-1092.
[TPL] Yu Y, Wang Y, Guo Z, Tang K, & Guo H. Memory Resource Pool Devisal in CUDA to Avoid Accessing Collision[J]. Journal of Chinese Computer Systems. Accepted.

Programming Layer

- Compiling System
  - automatic code generating
  - code optimization
  - language compatibility
- Examples:
  - C-Cuda Trans[CCT], hiCUDA[HCU]
  - PyCUDA[PCU], CUDA-Lite[CUL]

[CCT] Baskaran M, Ramanujam J, Gupta R, et al. Automatic C-to-CUDA Code Generation for Affine Programs[C]. LNCS Compiler Construction. Berlin, 2010: 244-263
[HCU] Han TD, Abdelrahman TS. hiCUDA: A High-level Directive-based Language for GPU Programming[C]. Proc of Workshop on General Purpose Processing on Graphics Processing Units, 2009: 1-10
[PCU] Klöckner A, Pinto N, Lee Y, et al. PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation[J]. Parallel Computing, 2011, 38(3): 157-174.
[CUL] Ueng S, Lathara M, Baghsorkhi S, et al. CUDA-Lite: Reducing GPU Programming Complexity[C]. LNCS Languages and Compilers for Parallel Computing, 2008: 1-15

Programming Layer

- Instruction Set
  - PTX ISA enhancements
    - general purposed computing with graphic languages (*old fashioned*)
- Examples:
  - Analysis PTX Kernels [PTX]

Andrew Kerr, Gregory Diamos, Sudhakar Yalamanchili, "A characterization and analysis of PTX kernels," iiswc, pp.3-12, 2009 IEEE International Symposium on Workload Characterization (IISWC), 2009

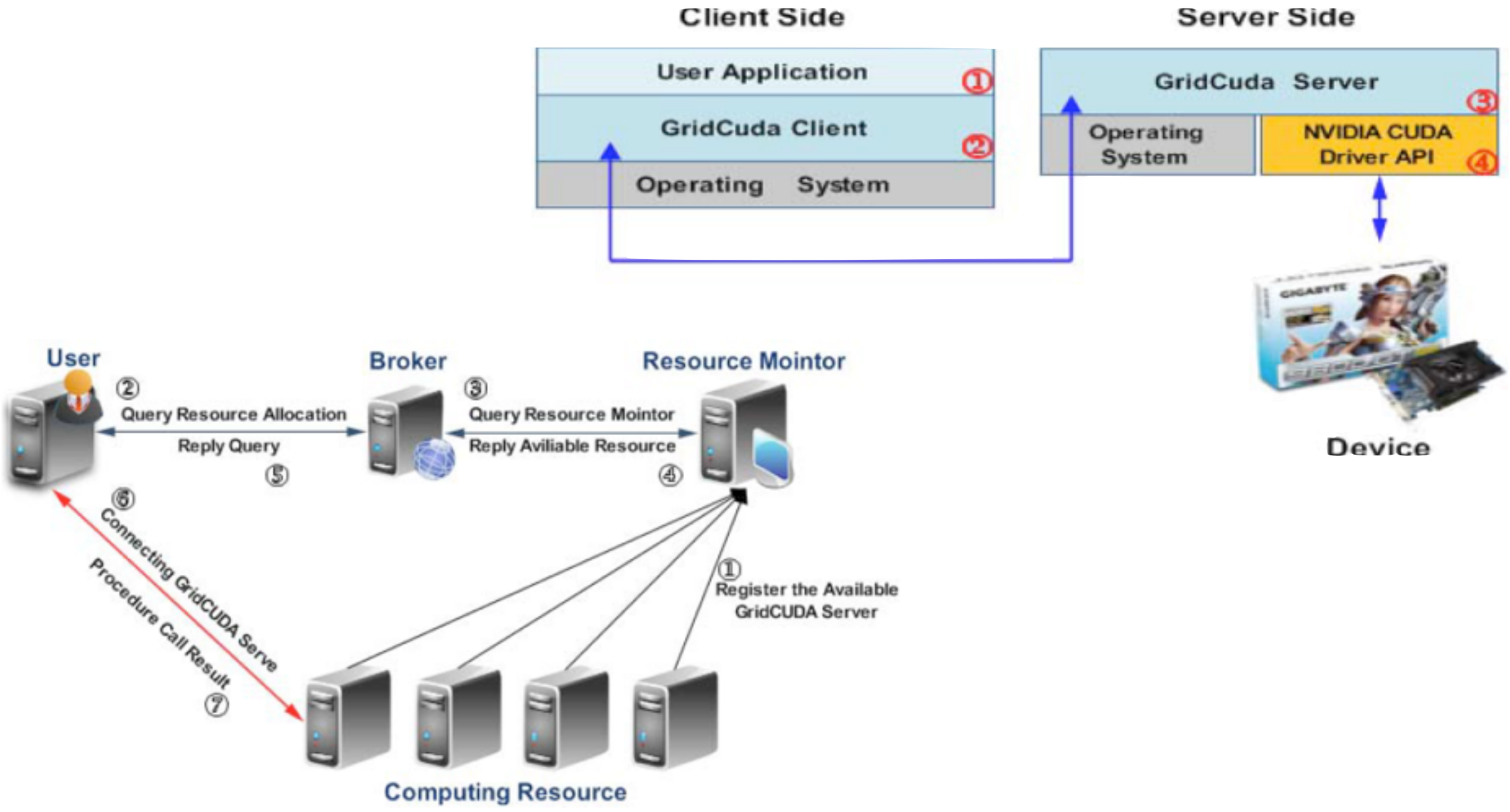
Platform Layer

- Frameworks and issues under different platforms
    - Framework expand current lib
    - Issues with introducing frameworks
- Examples:
    - rCUDA, GridCuda

[rCUDA] Duato J, Igual FD, Mayo R, et al. An efficient implementation of GPU virtualization in high performance clusters[C], Euro-Par Workshops, LNCS, 2010: 385-394.
[GridCuda] Liang TY, Chang YW. GridCuda: A Grid-enabled CUDA Programming Toolkit [C]. Workshops of International Conference on Advanced Information Networking and Applications, 2011: 141-146.

Platform Layer

- rCUDA & Grid CUDA



Platform Layer

- Ecosystem
  - debugger
  - profiling, checking, & tracing methods
- Examples:
  - Benchmarking GPUs for DLA [BG]
  - Memory Performance Evaluation [MPE]

[BG] Volkov, V., & Demmel, J. W. (2008). Benchmarking GPUs to tune dense linear algebra. ACM/IEEE Conference on Supercomputing (pp. 1–11). Austin, TX: ACM/IEEE. doi:10.1109/SC.2008.5214359
[MPE] Baghsorkhi, S. S., Gelado, I., Delahaye, M., & Hwu, W. W. (2012). Efficient performance evaluation of memory hierarchy for highly multithreaded graphics processors. ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (pp. 23–33). New York, New York, USA: ACM Press. doi:10.1145/2145816.2145820

Architecture Layer

- Schemes in hardware designing
  - Computing resource scheduling, Cache layout, Memory accessing ...
- Examples:
  - CPU-assisted GPGPU[CG], TAP[TAP]
  - Spatial Multitasking [SM], TCB[TCB]

[CG] Yang, Y., Xiang, P., Mantor, M., & Zhou, H. (2012). CPU-assisted GPGPU on fused CPU-GPU architectures. International Symposium on High Performance Computer Architecture (pp. 1–12). New Orleans, LA: IEEE. doi:10.1109/HPCA.2012.6168948

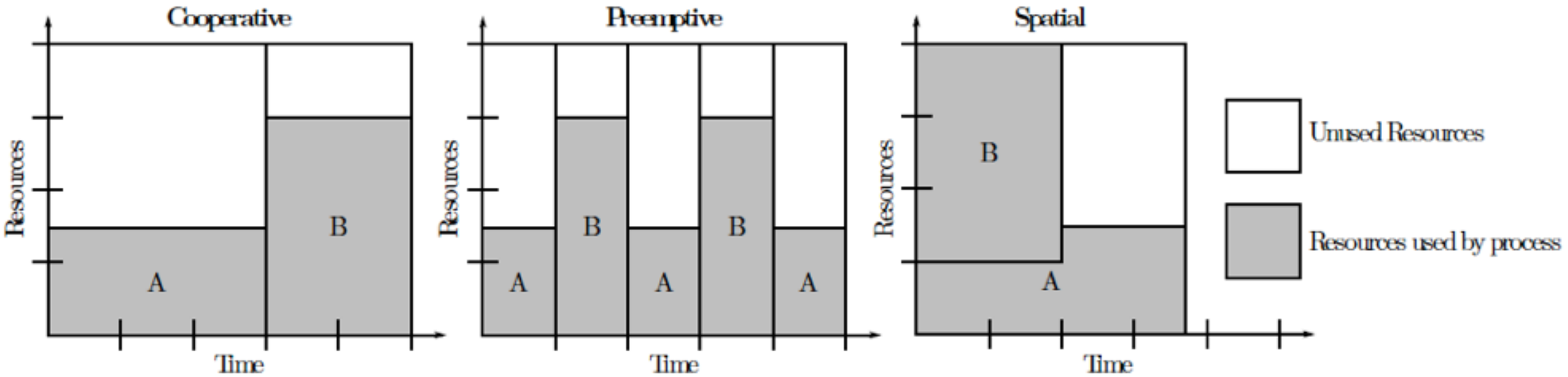
[TAP] Lee, J., & Kim, H. (2012). TAP: a TLP-aware cache management policy for a CPU-GPU heterogeneous architecture. International Symposium on High Performance Computer Architecture (pp. 1–12). New Orleans, LA: IEEE. doi:10.1109/HPCA.2012.6168947

[SM] Adriaens, J. T., Compton, K., Kim, N. S., & Schulte, M. J. (2012). The Case for GPGPU Spatial Multitasking. International Symposium on High Performance Computer Architecture (pp. 1–12). New Orleans, LA: IEEE. doi:10.1109/HPCA.2012.6168946

[TCB] Fung, W. W. L., & Aamodt, T. M. (2011). Thread block compaction for efficient SIMT control flow. *International Symposium on High Performance Computer Architecture* (pp. 25–36). San Antonio, TX: IEEE. doi:10.1109/HPCA.2011.5749714

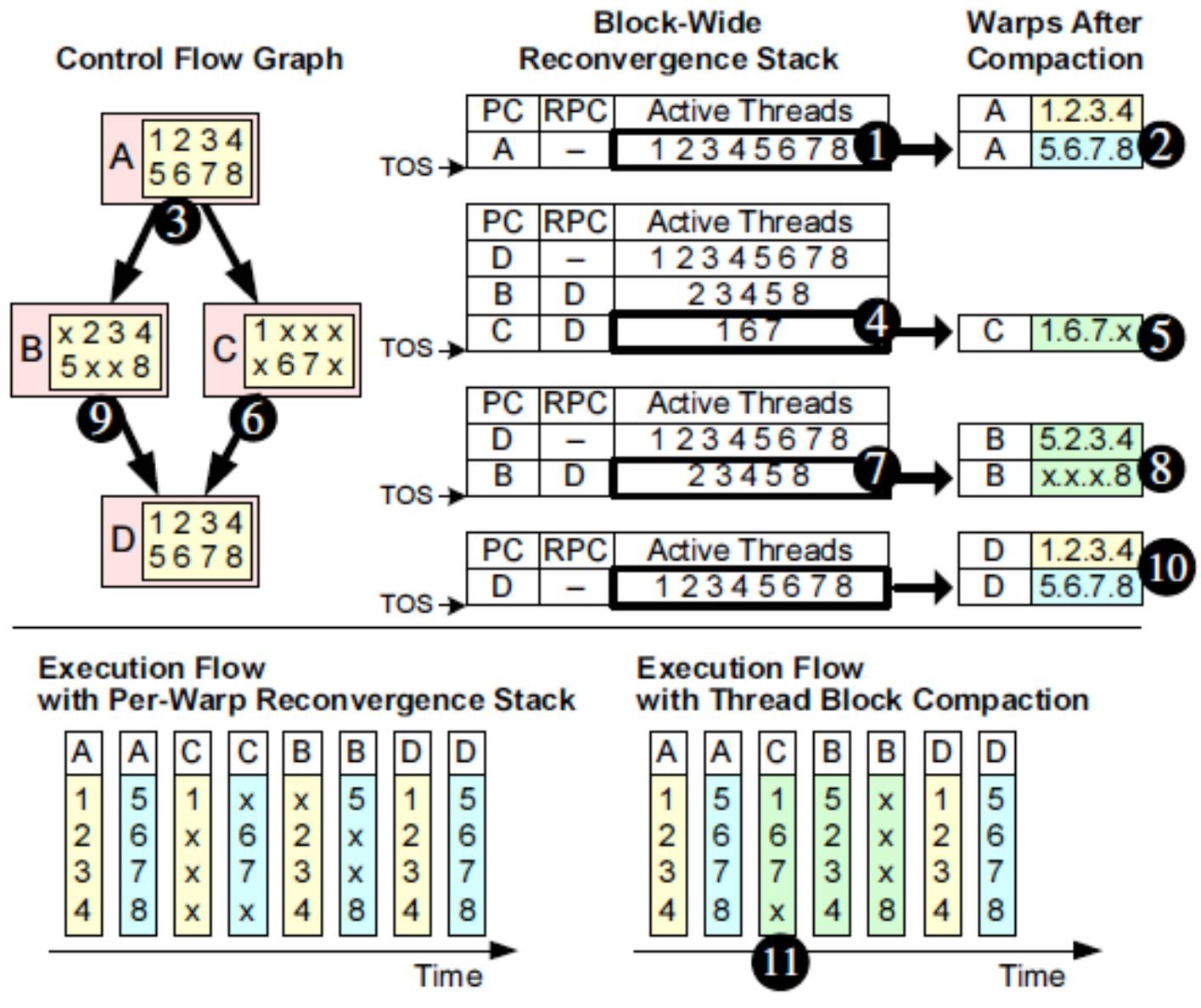
Architecture Layer

- Spatial Multitasking



Architecture Layer

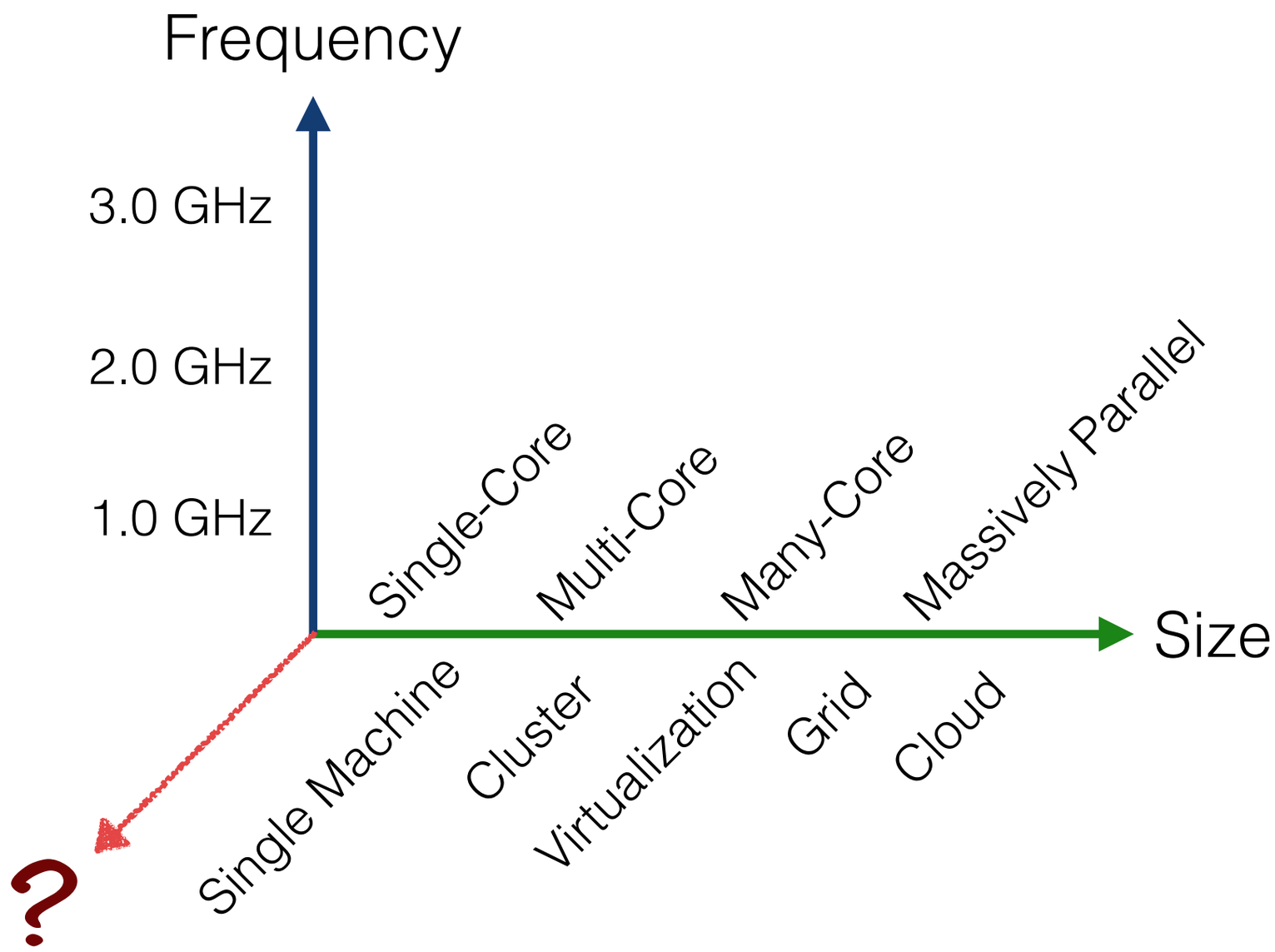
- TBC：Thread Block Compaction



Future Guess

- NOT official wording :-)
- Heterogeneous Computing
  - Beyond GPU, what about other computing resources?
- Virtualization & Cloud Computing
  - GPU and other co-processor integration
  - Issues & Schemes in new architecture

Future Guess



HPC&IP

Thanks for your attention