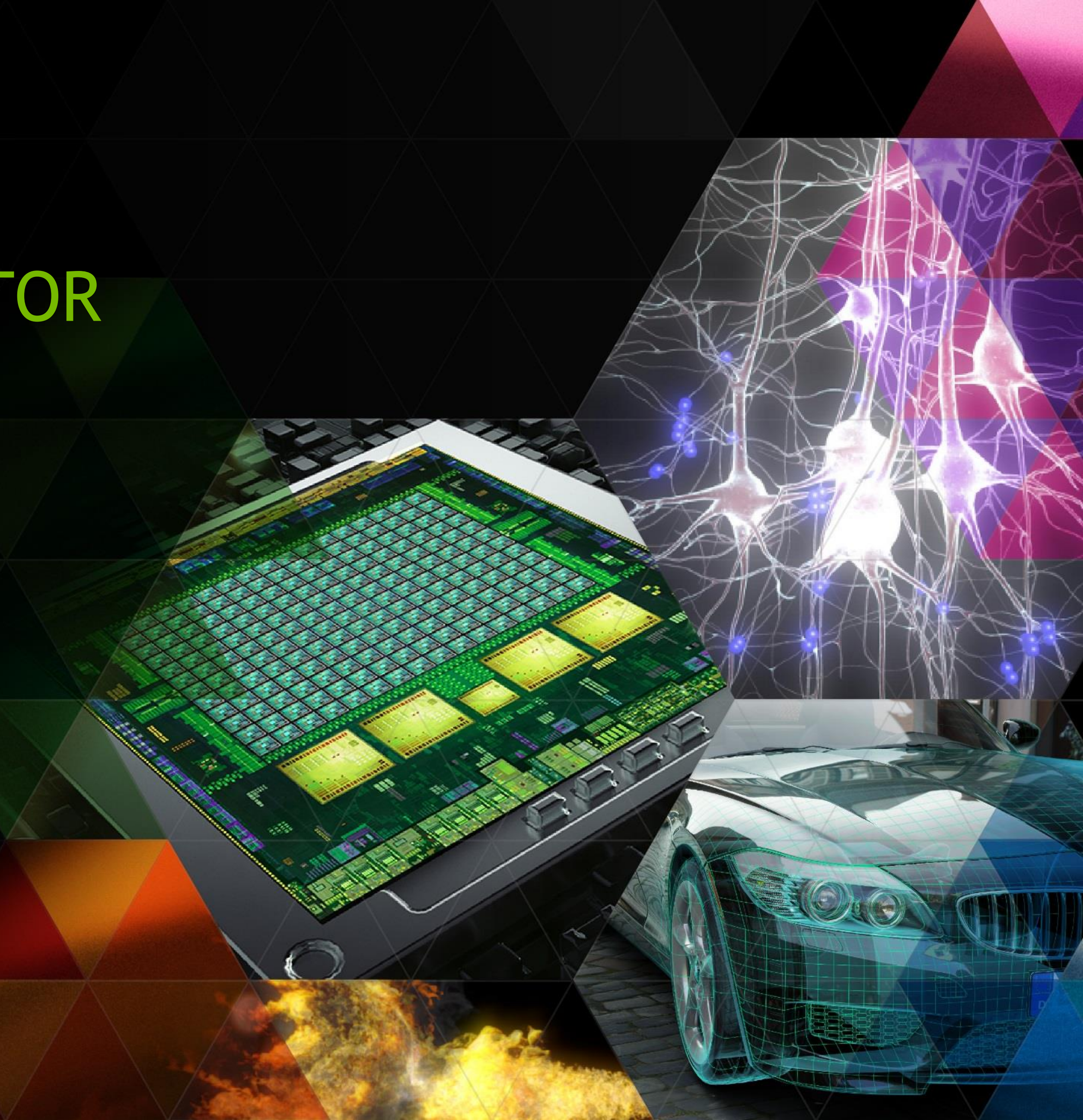
GPGPU AND ACCELERATOR ARCHITECTURE TRENDS

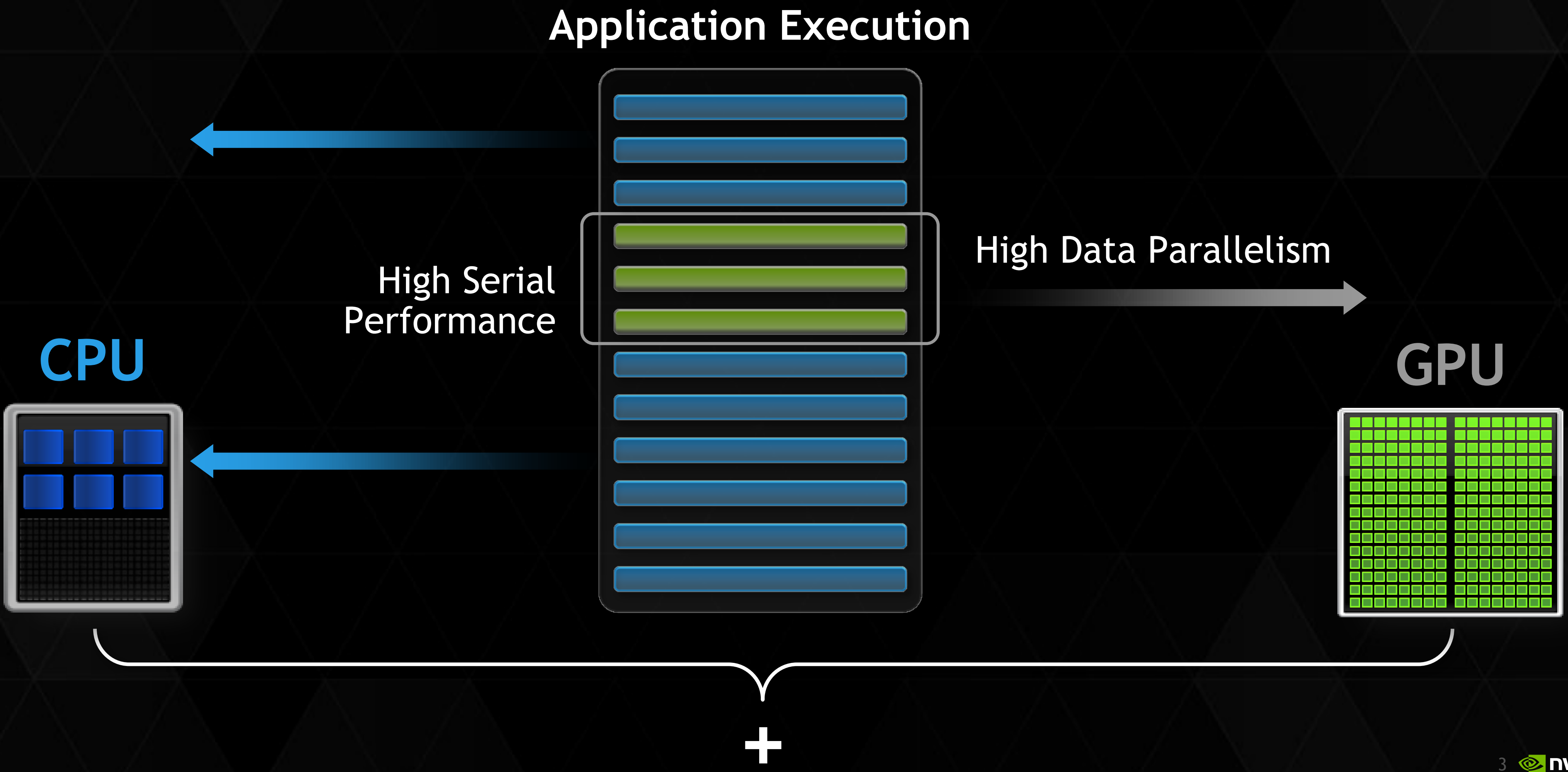
Nikolay Sakharnykh,

Developer Technology Engineer



GPU Architecture & Best Practices

WHAT IS HETEROGENEOUS COMPUTING?



Application Execution

High Serial Performance

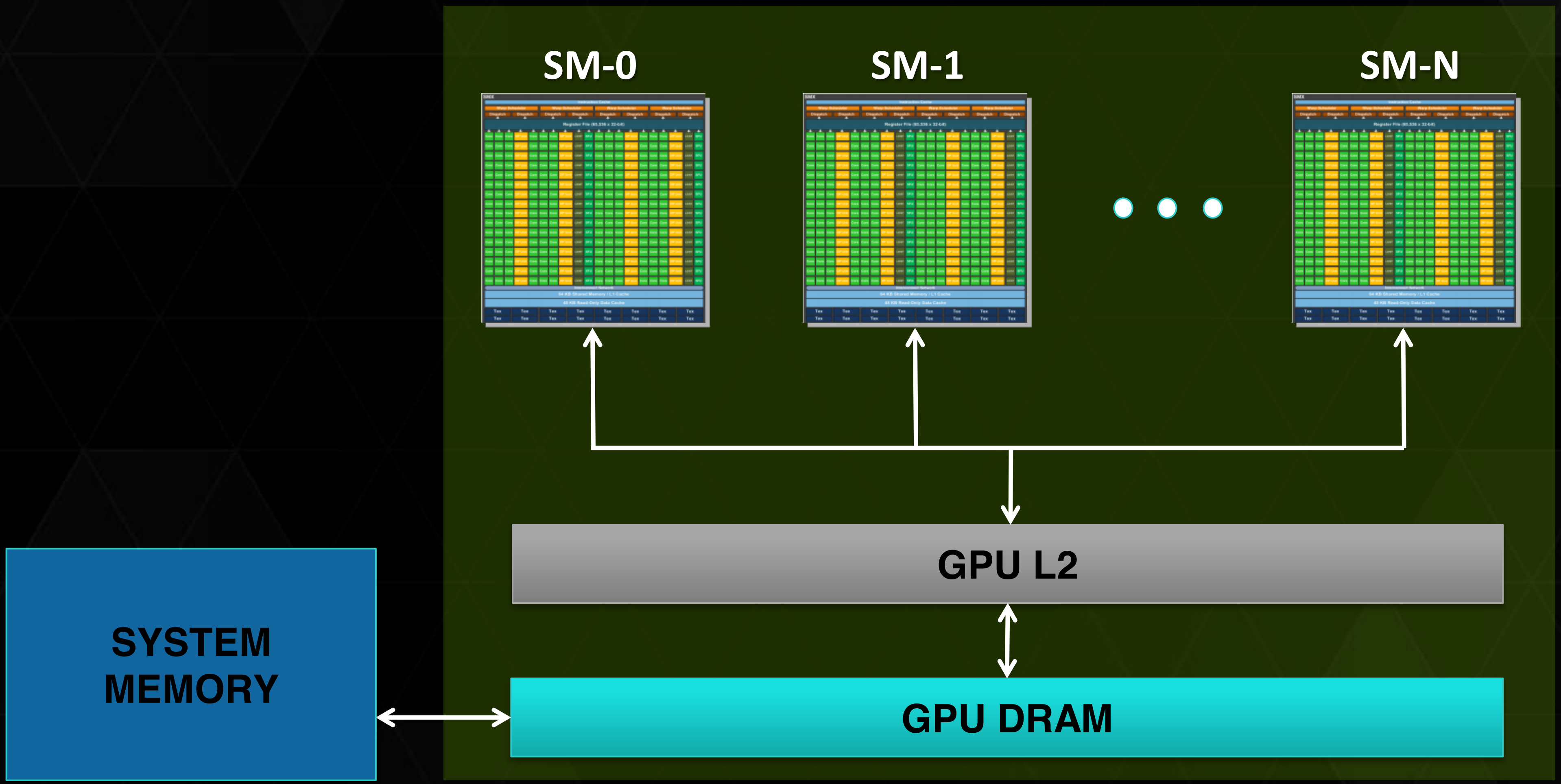
High Data Parallelism

CPU

GPU

+

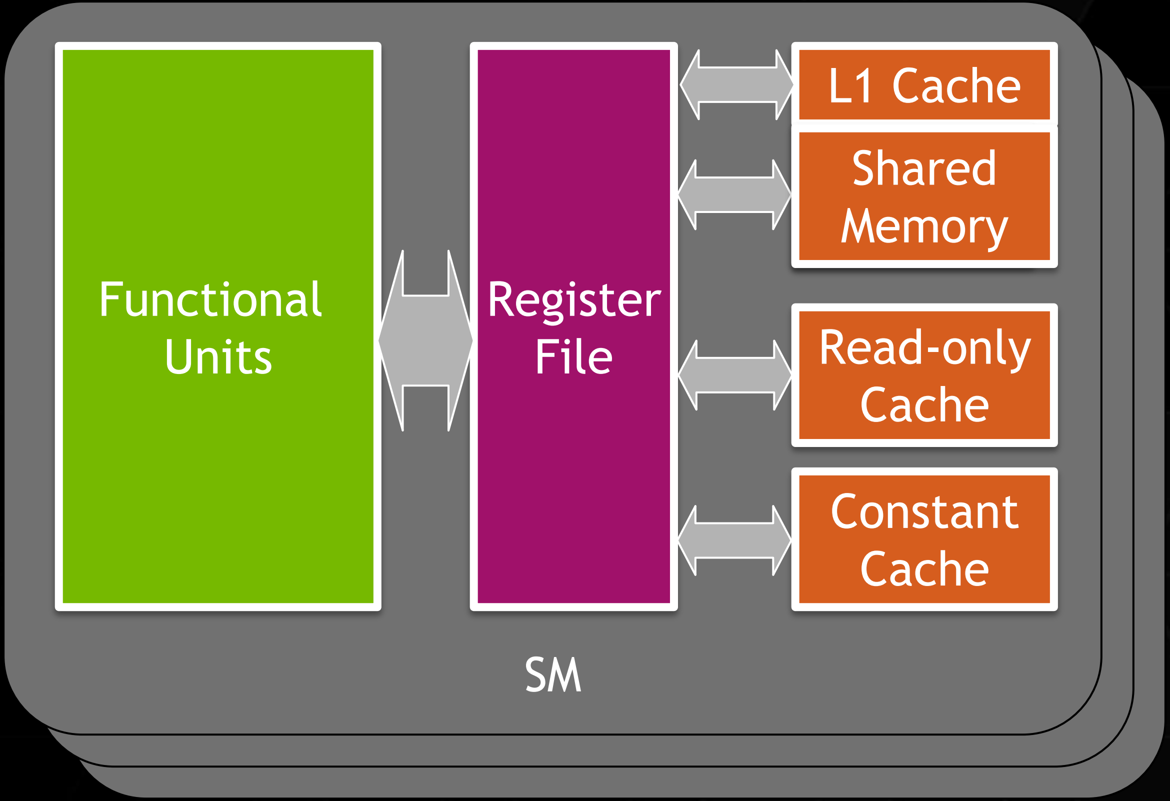
GPU ARCHITECTURE



GPU SM ARCHITECTURE

Kepler SM

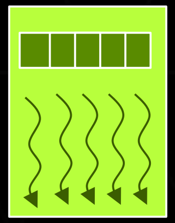
- Functional Units = CUDA cores
  - 192 SP FP operations/clock
  - 64 DP FP operations/clock
- Register file (256KB)
- Shared memory (16-48KB)
- L1 cache (16-48KB)
- Read-only cache (48KB)
- Constant cache (8KB)



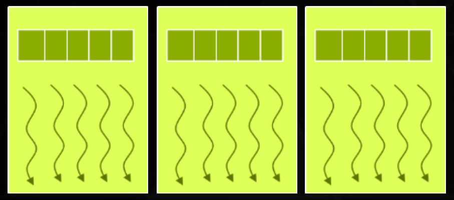
SIMT EXECUTION MODEL

- Thread: sequential execution unit
  - All threads execute same sequential program
  - Threads execute in parallel
- Thread Block: a group of threads
  - Threads within a block can cooperate
    - Light-weight synchronization
    - Data exchange
- Grid: a collection of thread blocks
  - Thread blocks do not synchronize with each other
  - Communication between blocks is expensive

Thread



Thread Block

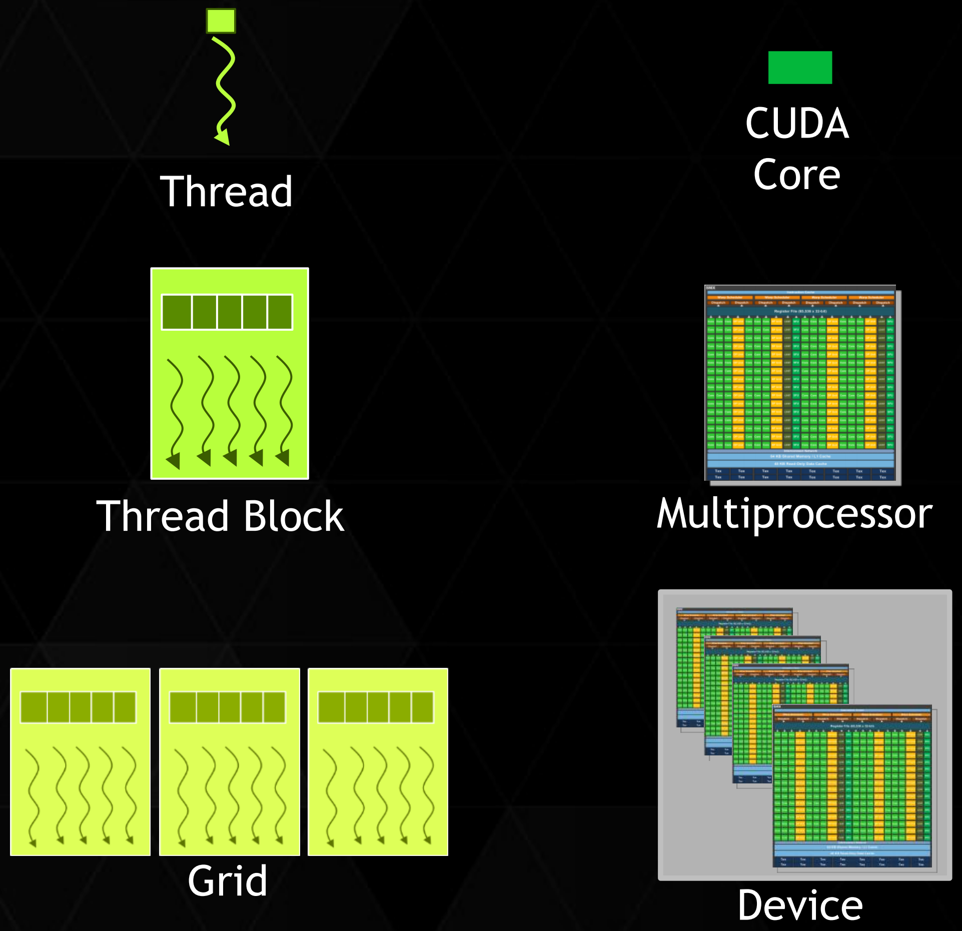


Grid

SIMT EXECUTION MODEL

Software

Hardware



Thread

CUDA Core

Thread Block

Multiprocessor

Grid

Device

Threads are executed by CUDA Cores

Thread blocks are executed on multiprocessors

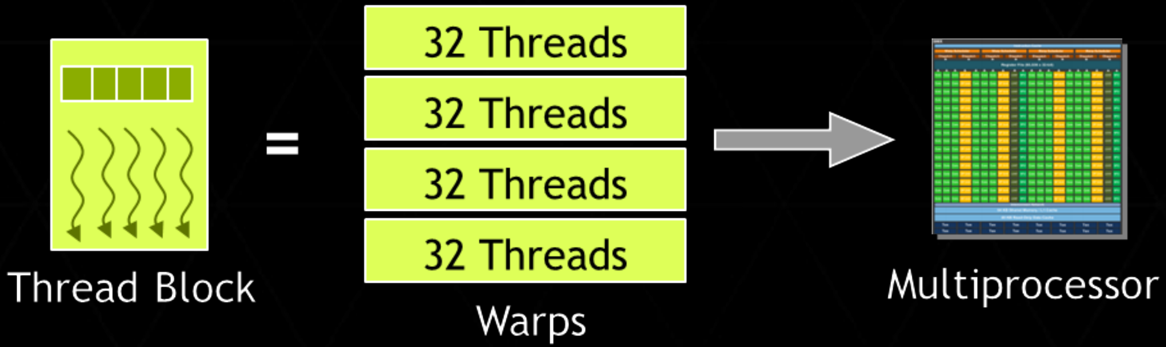
Thread blocks do not migrate

Several concurrent thread blocks can reside on one multiprocessor - limited by multiprocessor resources (shared memory and register file)

A kernel is launched as a grid of thread blocks

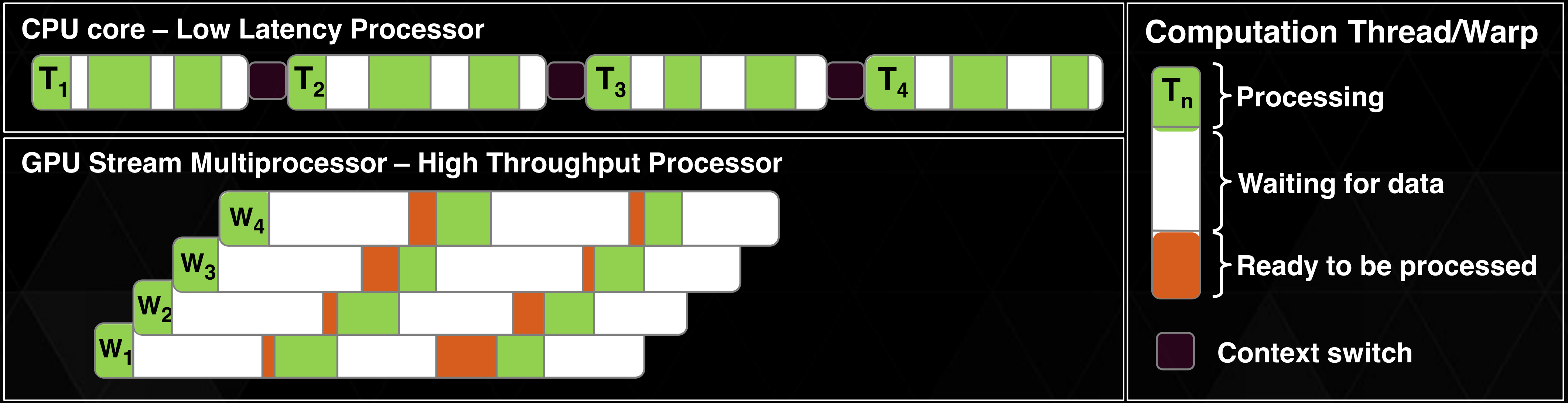
SIMT EXECUTION MODEL

- Threads are organized into groups of 32 threads called “warps”
- All threads within a warp execute the same instruction simultaneously



LOW LATENCY OR HIGH THROUGHPUT?

- CPU architecture must minimize latency within each thread
- GPU architecture hides latency with computation from other threads



ACCELERATOR FUNDAMENTALS

- We must expose enough parallelism to saturate the device
  - Accelerator threads are slower than CPU threads
  - Accelerators have orders of magnitude more threads

Fine-grained parallelism is good

t0	t1	t2	t3
t4	t5	t6	t7
t8	t9	t10	t11
t12	t13	t14	t15

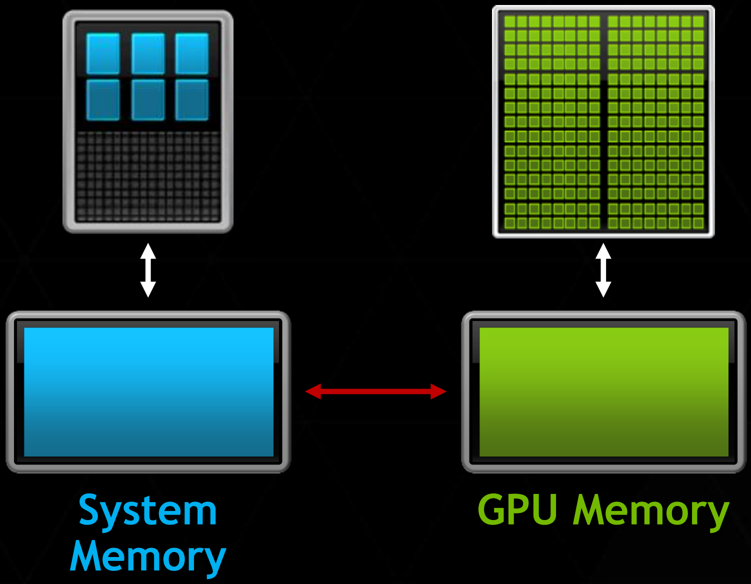
Coarse-grained parallelism is bad

t0	t0	t0	t0
t1	t1	t1	t1
t2	t2	t2	t2
t3	t3	t3	t3

BEST PRACTICES

Optimize Data Locality: GPU

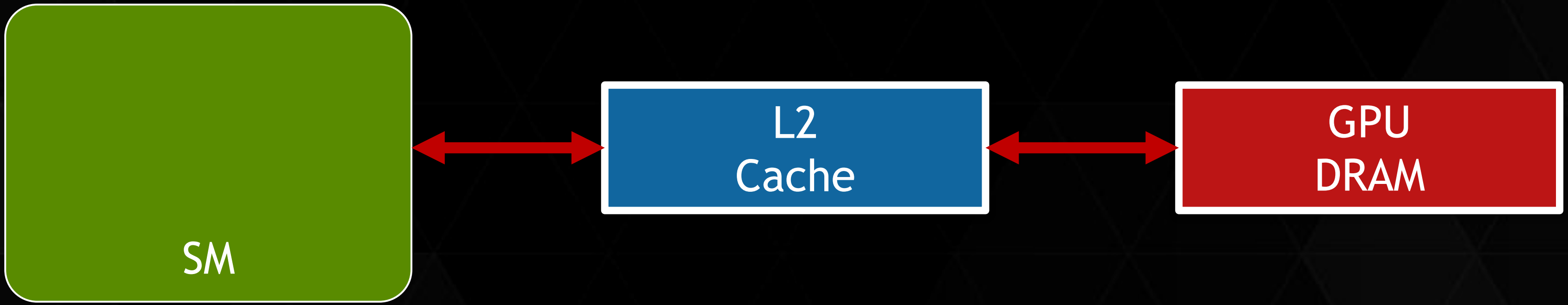
- Minimize data transfers between CPU and GPU



BEST PRACTICES

Optimize Data Locality: SM

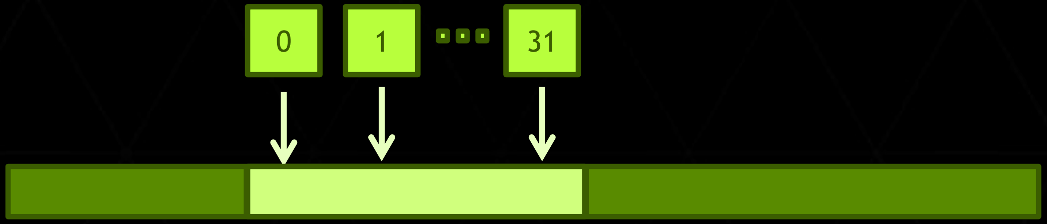
- Minimize redundant accesses to L2 and DRAM
  - Store intermediate results in registers instead of global memory
  - Use shared memory for data frequently used within a thread block
  - Use `const __restrict__` to take advantage of read-only cache



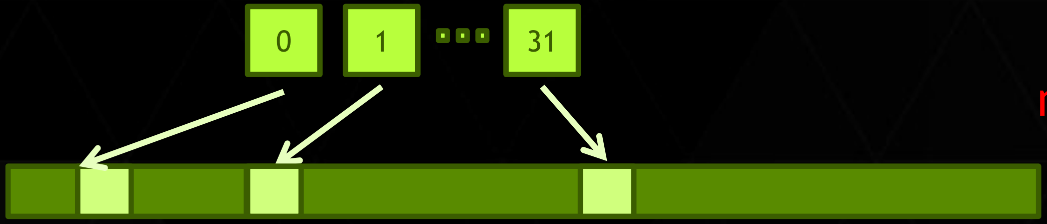
BEST PRACTICES

Coalesce Memory Requests

- If multiple addresses from a warp lie within the same cache line, that line is moved only once
- Best case: all addresses lie in a single L1 cache line (128B)



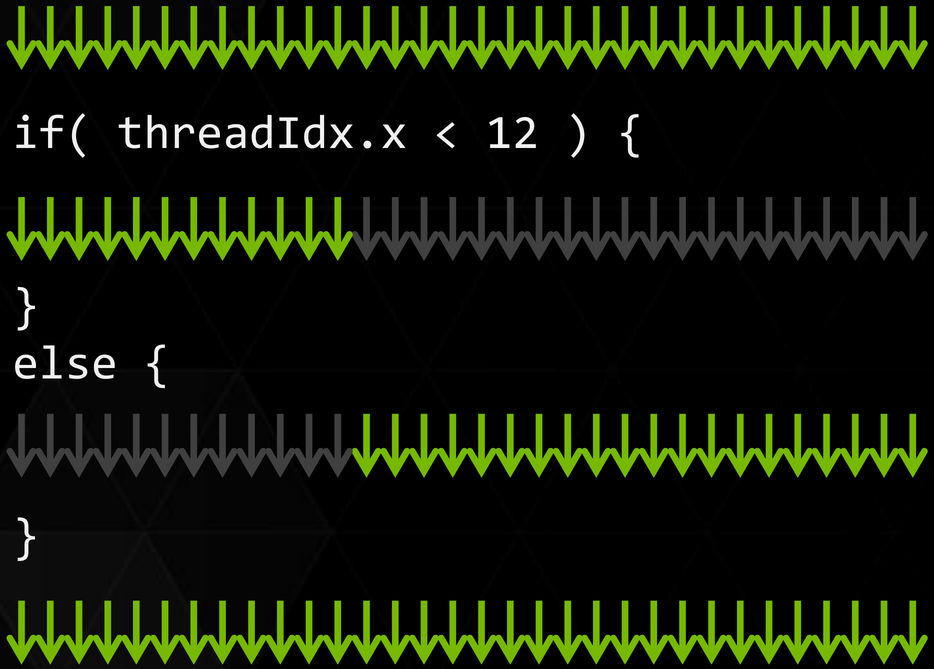
- Worst case: 32 separate L1 transactions (31 replays)



more memory traffic
more issued instructions

BEST PRACTICES

Avoid Warp Divergence



```
if( threadIdx.x < 12 ) {

}
else {

}
```

Instructions are issued per warp

Different execution paths within a warp are serialized

Different warps can execute different code with no impact on performance

Avoid branching on thread index

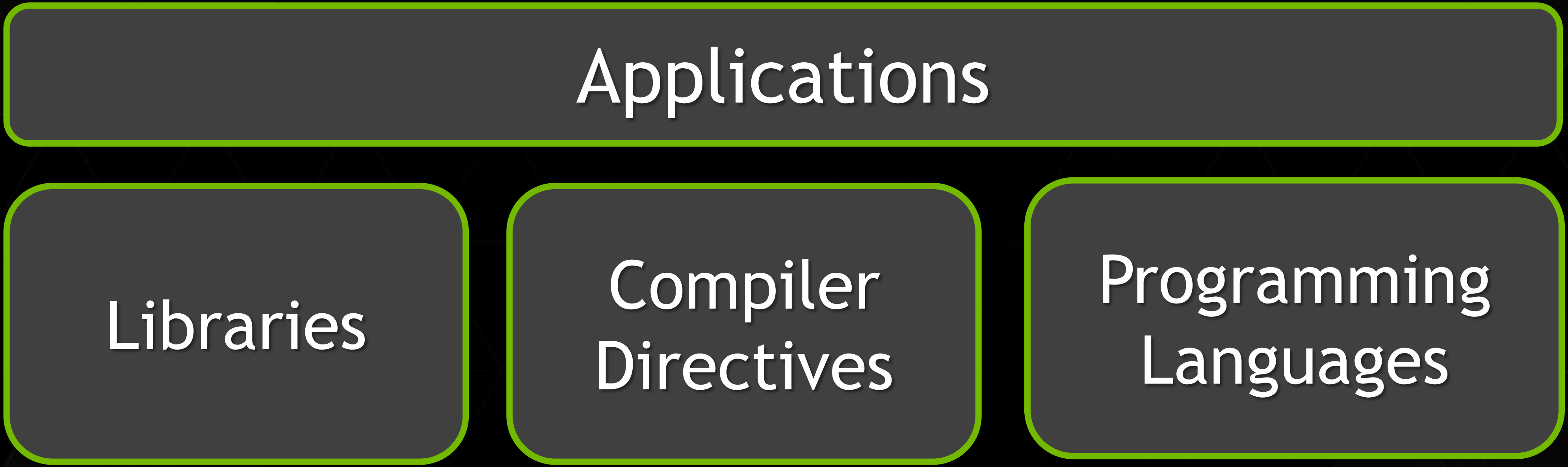
BEST PRACTICES

Quick Summary

- Expose enough parallelism
- Optimize data locality
    - Minimize transfers between CPU and GPU
    - Minimize redundant accesses to GPU DRAM
- Avoid memory divergence
    - Ensure global accesses are coalesced
- Avoid warp divergence
    - Ensure threads in a warp execute the same path

Programming GPUs

3 WAYS TO ACCELERATE APPLICATIONS



Applications

Libraries

Compiler Directives

Programming Languages

Easy to use
Most Performance

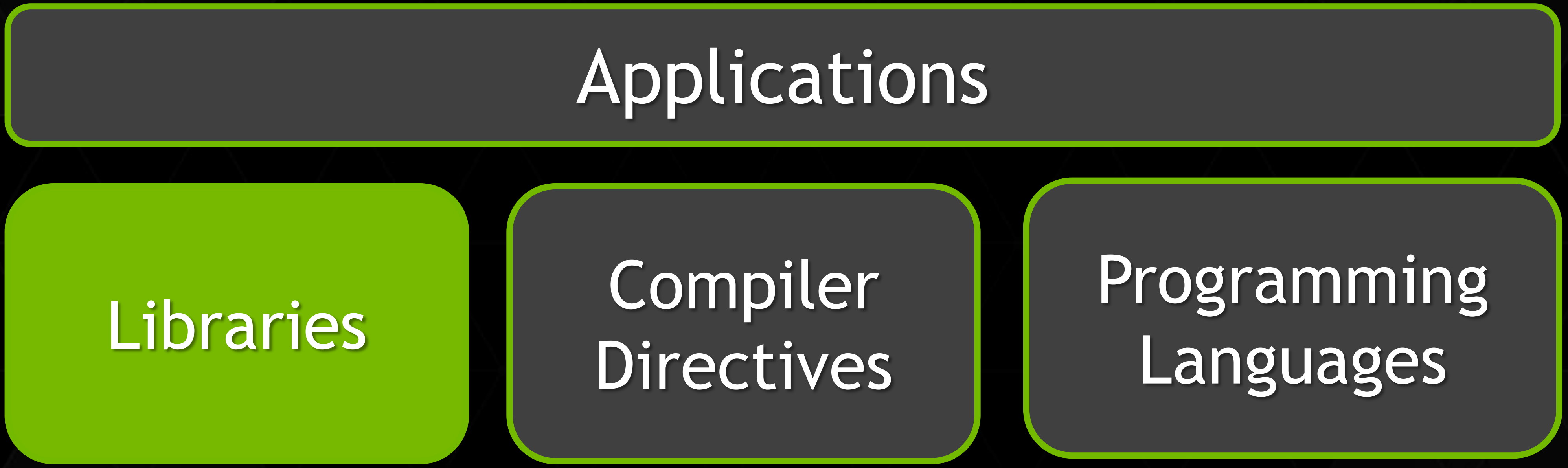
Easy to use
Portable code

Most Performance
Most Flexibility

SIMPLICITY & PERFORMANCE

- Accelerated Libraries
  - Little or no code change for standard libraries, high performance
  - Limited by what libraries are available
- Compiler Directives
  - Based on existing programming languages, so they are simple and familiar
  - Performance may not be optimal, directives often do not expose low level architectural details
- Parallel Programming languages
  - Expose low-level details for maximum performance
  - Often more difficult to learn and more time consuming to implement

3 WAYS TO ACCELERATE APPLICATIONS



Applications

Libraries

Compiler Directives

Programming Languages

Easy to use
Most Performance

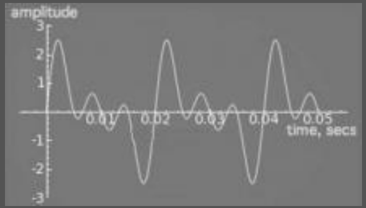
Easy to use
Portable code

Most Performance
Most Flexibility

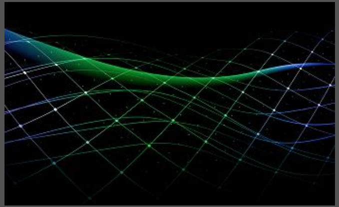
NVIDIA DEVELOPER LIBRARIES

cuBLAS
cuBLAS-XT
NVBLAS

cuFFT
cuFFT-XT

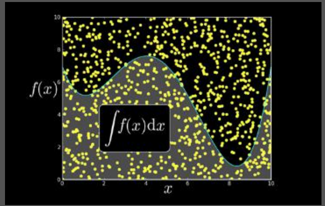


cuSPARSE
cuSOLVER
AMGX



cuDNN

cuRAND

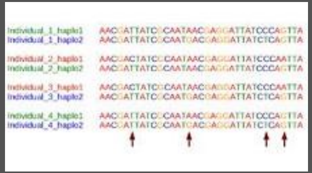


THRUST

NPP

NVENC

NVBIO



https://developer.nvidia.com/gpu-accelerated-libraries

BLAS LIBRARIES

CUBLAS

- Step 1: Substitute library calls with equivalent CUDA library calls

```
saxpy ( … )                cublasSaxpy ( … )
```

- Step 2*: Manage data locality if necessary

  - with CUDA: `cudaMalloc(), cudaMemcpy(), etc.`
  - with CUBLAS: `cublasAlloc(), cublasSetVector(), etc.`

- Step 3: Rebuild and link the CUDA-accelerated library

```
nvcc myobj.o -l cublas
```

https://developer.nvidia.com/cuBLAS

BLAS LIBRARIES

NVBLAS

- Drop-in replacement for level 3 BLAS routines (i.e. GEMM)
  - Automatically uses all available GPUs in the system
  - No need to modify or even recompile your application
  - LD_PRELOAD=<path to libnvblas.so> <application>

http://docs.nvidia.com/cuda/nvblas

SPARSE MATRIX LIBRARIES

- CUSPARSE
  - Collection of sparse matrix building blocks
  - Supports multiple matrix formats
  - https://developer.nvidia.com/cuSPARSE

- CUSOLVER
  - Collection of sparse and dense solvers
  - Similar to LAPACK
  - https://developer.nvidia.com/cusolver

- AmgX
  - Algebraic Multi-Grid Solver
  - Flexible configuration
  - Krylov methods
  - Parallel smoothers
  - MPI support
  - https://developer.nvidia.com/amgx

THRUST

- C++ template library for CUDA
  - Mimics Standard Template Library (STL)

Data Structures

- `thrust::device_vector`
- `thrust::host_vector`
- `thrust::device_ptr`
- Etc.

Algorithms

- `thrust::sort`
- `thrust::reduce`
- `thrust::exclusive_scan`
- Etc.

https://developer.nvidia.com/thrust

THRUST EXAMPLE: SAXPY

STL C++ Code

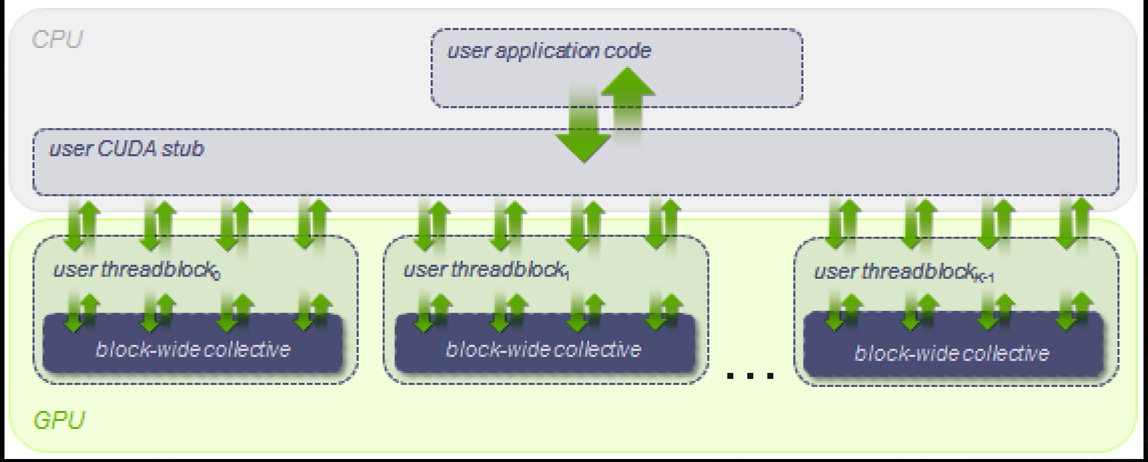
```
int N = 1<<20;
std::vector<float> x(N), y(N);

...




// Perform SAXPY on 1M elements
std::transform(x.begin(), x.end(),
               y.begin(), y.end(),
               2.0f * _1 + _2);
```

Parallel C++ Code

```
int N = 1<<20;
thrust::host_vector<float> x(N), y(N);

...

thrust::device_vector<float> d_x = x;
thrust::device_vector<float> d_y = y;

// Perform SAXPY on 1M elements
thrust::transform(d_x.begin(), d_x.end(),
                  d_y.begin(), d_y.begin(),
                  2.0f * _1 + _2);
```

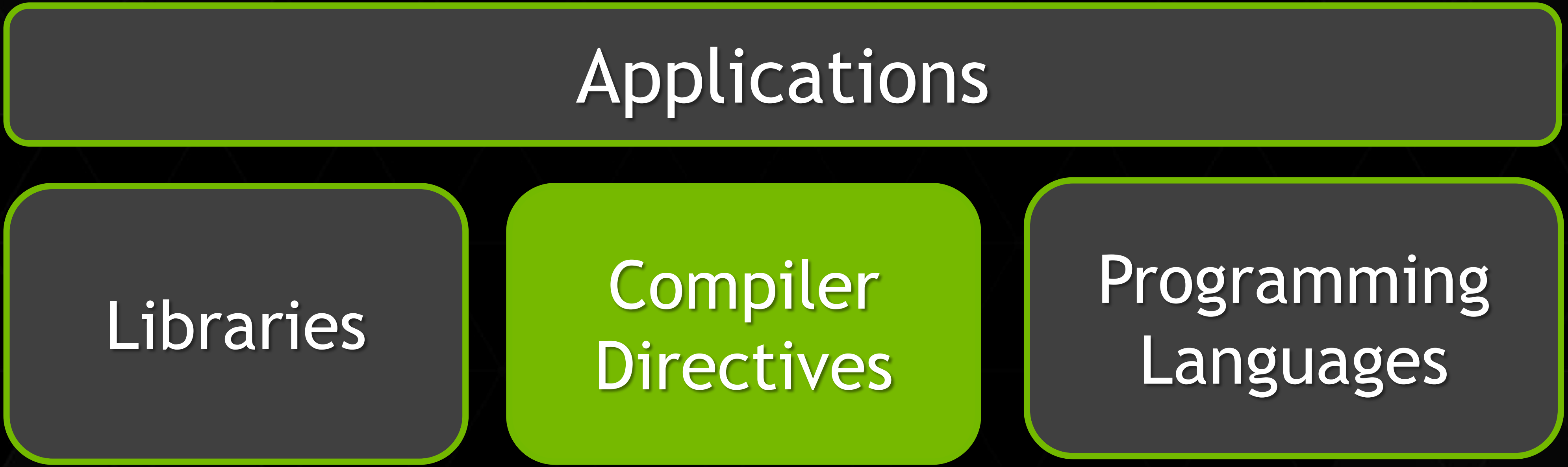
CUB

- Library of SIMT collective primitives for block-wide and warp-wide kernel programming
    - Cooperative sort, prefix sum, reduction, histogram, etc.



http://nvlabs.github.io/cub

3 WAYS TO ACCELERATE APPLICATIONS



Applications

Libraries

Compiler Directives

Programming Languages

Easy to use
Most Performance

Easy to use
Portable code

Most Performance
Most Flexibility

OPENACC DIRECTIVES SYNTAX

- C/C++

```
#pragma acc directive [clause [,] clause] …]
...structured code block
```

- Fortran

```
!$acc directive [clause [,] clause] …]
...structured code block
!$acc end directive
```

OPENACC EXAMPLE: SAXPY

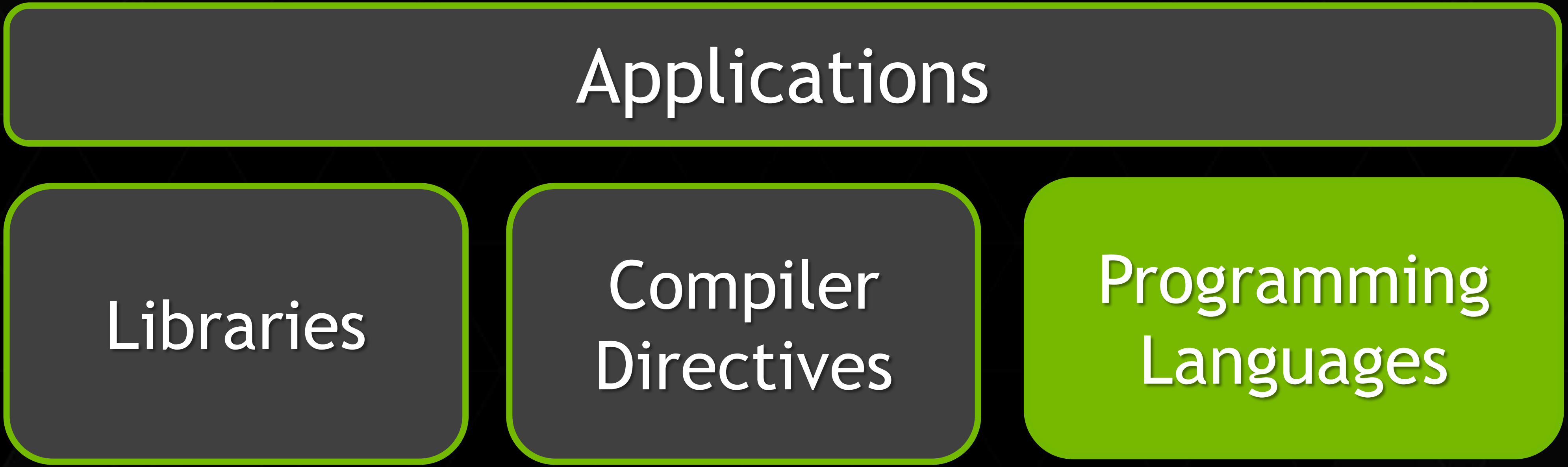
SAXPY in C

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

...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

SAXPY in Fortran

```fortran
subroutine saxpy(n, a, x, y)
  real :: x(n), y(n), a
  integer :: n, i

  !$acc parallel loop
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
  !$acc end parallel loop
end subroutine saxpy

...
! Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x, y)
...
```

3 WAYS TO ACCELERATE APPLICATIONS



Applications

Libraries

Compiler Directives

Programming Languages

Easy to use
Most Performance

Easy to use
Portable code

Most Performance
Most Flexibility

GPU PROGRAMMING LANGUAGES

Numerical analytics	MATLAB, Mathematica, LabVIEW
Fortran	CUDA Fortran
C	CUDA C
C++	CUDA C++
Python	PyCUDA, Copperhead
F#	Alea.cuBase

CUDA EXAMPLE: SAXPY

Serial C Code

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

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

Parallel CUDA Code

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

// Perform SAXPY on 1M elements
saxpy_parallel<<<4096,256>>>(n,2.0,x,y);
```

CUDA MEMORY MANAGEMENT

Without Unified Memory

```
void sortfile(FILE *fp, int N) {
  char *data, *d_data;
  data = (char*) malloc(N);
  cudaMalloc (&d_data, N);

  fread(data, 1, N, fp);

  cudaMemcpy(d_data,data,N,H2D);
  qsort<<<...>>>(d_data,N,1,compare);
  cudaMemcpy(data,d_data,N,D2H);

  use_data(data);

  cudaFree(d_data);
  free(data);
}
```

Unified Memory

```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);


  fread(data, 1, N, fp);


  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);

}
```

http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/

CUDA MEMORY MANAGEMENT

- cudaMalloc & cudaMemcpy
  - Explicitly track host and device memory
  - Explicitly relocate data (sync or async)
  - Expresses data locality (most performance)

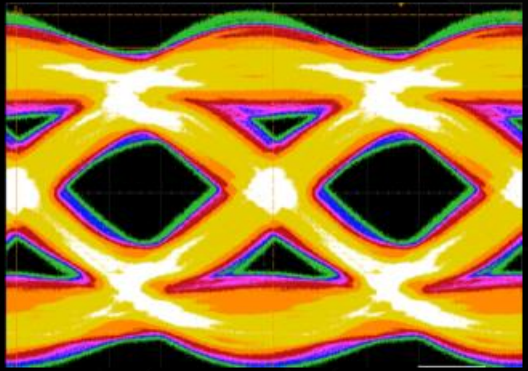
- cudaMallocManaged
  - Single pointer for host & device memory
  - Automatic relocation at launch and sync
  - Easier porting of application (simplicity)

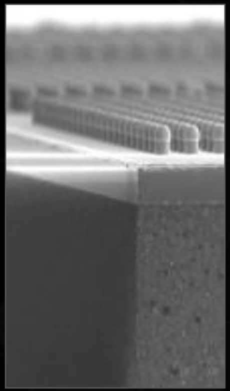
ADVICE: develop with cudaMallocManaged, optimize with cudaMalloc if necessary

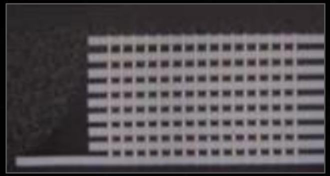
The Future of GPUs

NEW TECHNOLOGIES

- NVLINK
  - GPU high speed interconnect
  - 5x-12x PCI-E Gen3 bandwidth
  - Planned support for POWER® CPUs
- HBM (Stacked Memory)
  - 4x higher bandwidth (~1 TB/s)
  - 3x larger capacity
  - 4x more energy efficient per bit







IMPROVING UNIFIED MEMORY

```
void sortfile(FILE *fp, int N) {
  char *data, *d_data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(d_data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char*) malloc(N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  free(data);
}
```

SUMMIT

2017 OLCF Leadership System

- Vendor: IBM (Prime) / NVIDIA™ / Mellanox Technologies®
- At least 5X Titan’s Application Performance
- Approximately 3400 nodes, each with:
    - IBM POWER9 CPUs + NVIDIA Volta GPUs
    - CPUs and GPUs connected with high speed NVLink
    - Large coherent memory: over 512 GB (HBM + DDR4)
    - Over 40 TF peak performance
- Dual-rail Mellanox® EDR-IB full, non-blocking fat-tree interconnect

SUMMIT

How does Summit compare to Titan

Feature	Summit	Titan
Application Performance	5-10x Titan	Baseline
Number of Nodes	~3,400	18,688
Node performance	> 40 TF	1.4 TF
Memory per Node	>512 GB (HBM + DDR4)	38GB (GDDR5+DDR3)
NVRAM per Node	800 GB	0
Node Interconnect	NVLink (5-12x PCIe 3)	PCIe 2
System Interconnect (node injection bandwidth)	Dual Rail EDR-IB (23 GB/s)	Gemini (6.4 GB/s)
Interconnect Topology	Non-blocking Fat Tree	3D Torus
Processors	IBM POWER9 NVIDIA Volta™	AMD Opteron™ NVIDIA Kepler™
File System	120 PB, 1 TB/s, GPFS™	32 PB, 1 TB/s, Lustre®
Peak power consumption	10 MW	9 MW

SUMMIT

Titan & Summit Application Differences

- Fewer but much more powerful nodes
  - 1/6th the number of nodes, but ~25x more powerful
- Must exploit more node-level parallelism
  - Multiple CPUs and GPU to keep busy
  - Likely requires OpenMP or OpenACC programming model
- Very large memory
  - Summit has ~15x more memory per node than Titan
- Interconnect is only ~3x the bandwidth of Titan
  - Need to exploit data locality within nodes to minimize message passing traffic

RESOURCES

Learn more about GPUs

- CUDA resource center:
  - http://docs.nvidia.com/cuda
- GTC on-demand and webinars:
  - http://on-demand-gtc.gputechconf.com
  - http://www.gputechconf.com/gtc-webinars
- Parallel Forall Blog:
  - http://devblogs.nvidia.com/parallelforall
- Self-paced labs:
  - http://nvlabs.qwiklab.com