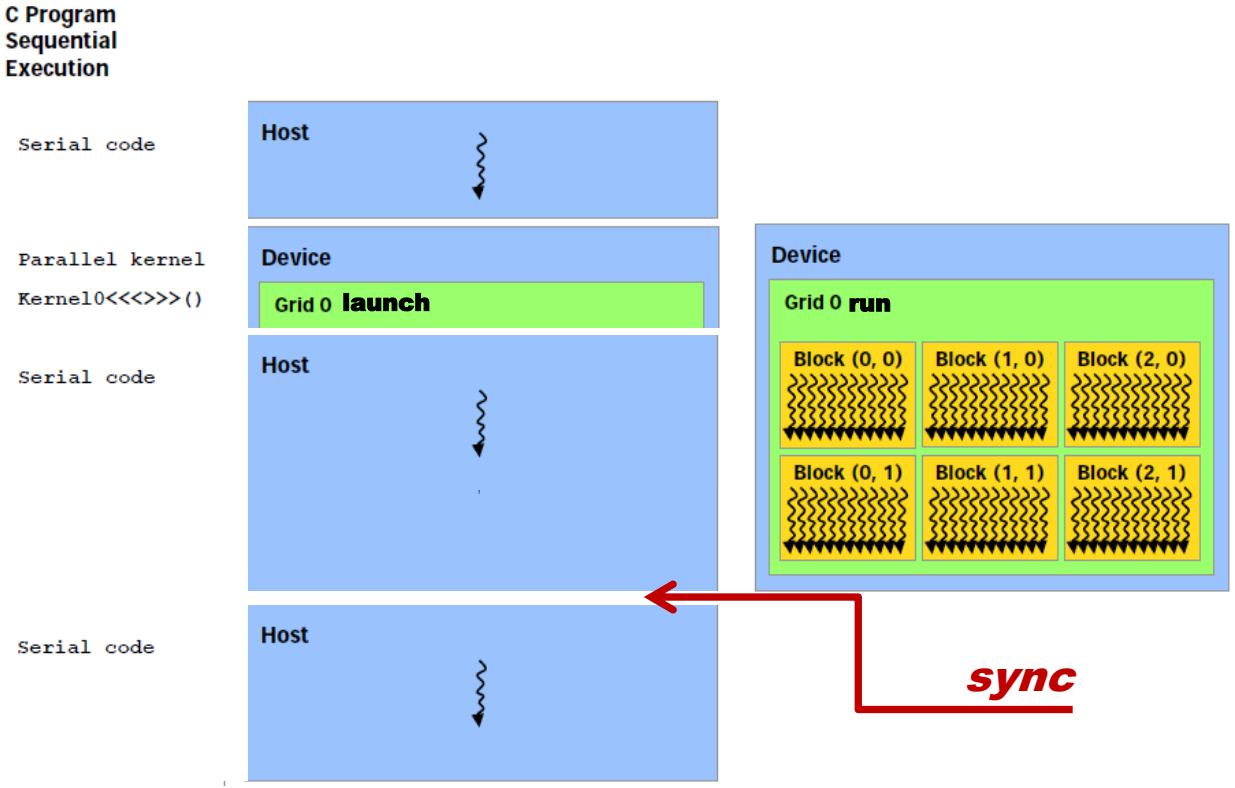
Hybrid Architecture



Hello World!


```
int main()
{
    ...
    // Kernel invocation
    VecAdd<<<1, N>>>(A, B, C);
}
```

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

Syntax

- `__host__ & __device__`
  - Compilation target but can be both
- `__global__`
  - Kernel function
- `<<< magic happens here >>>`
  - Kernel launch

Kernel is “normal” C (almost)

- No return value
- No recursion
- Inline functions
  - No function pointer
- No exceptions when using C++

Built-in Variables

- Which thread in a block?

```
uint3 threadIdx
```

- Which block in a grid?

```
uint3 blockIdx
```

- Which data to work on?

```
dim3 blockDim
dim3 gridDim
```

New Types for Convenience

- Some new types

  ```
  int2, int3, int4, uint2, uint3, ...
  float2, float3, float4, double2, ...
  ```

  E.g., struct of { x, y, z, w}

- For dimensions: `dim3`
  - Neglected dimensions are 1 automatically

Nvidia Compiler

- New compiler for special syntax

  nvcc

- Generates CPU/host and GPU/device codes
  - PTX instructions for GPU
  - Uses normal C compiler underneath

Memory access

Reduction

- How to aggregate?
  - E.g., count, sum, min, max
- Think of vector dot product
  - Map: $c_n=a_nb_n$ products
  - Reduce: Σc_n

Strategies

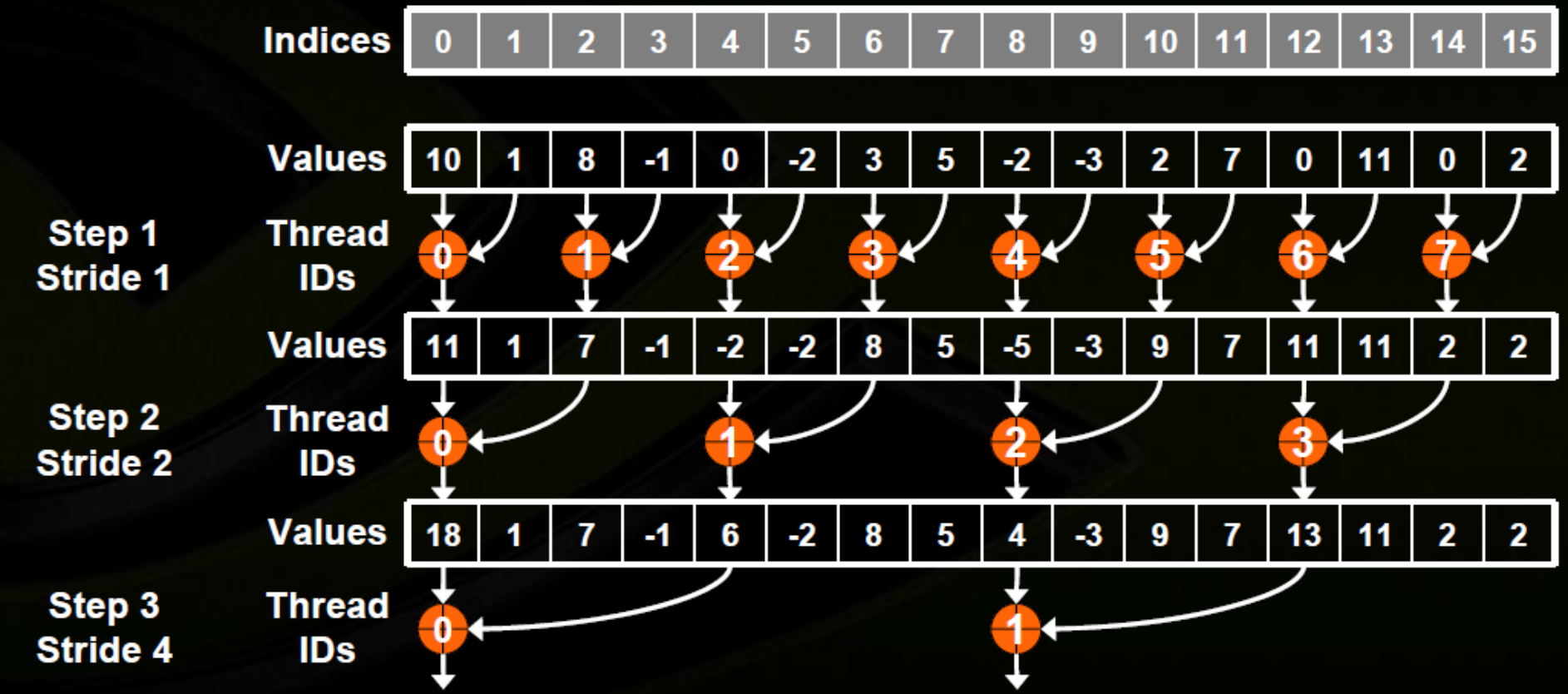
- Pyramid
  - E.g., each thread adds two elements
  - Keep repeating
- Naïve implementation
  - Launch new kernels at each level
  - Synchronize device

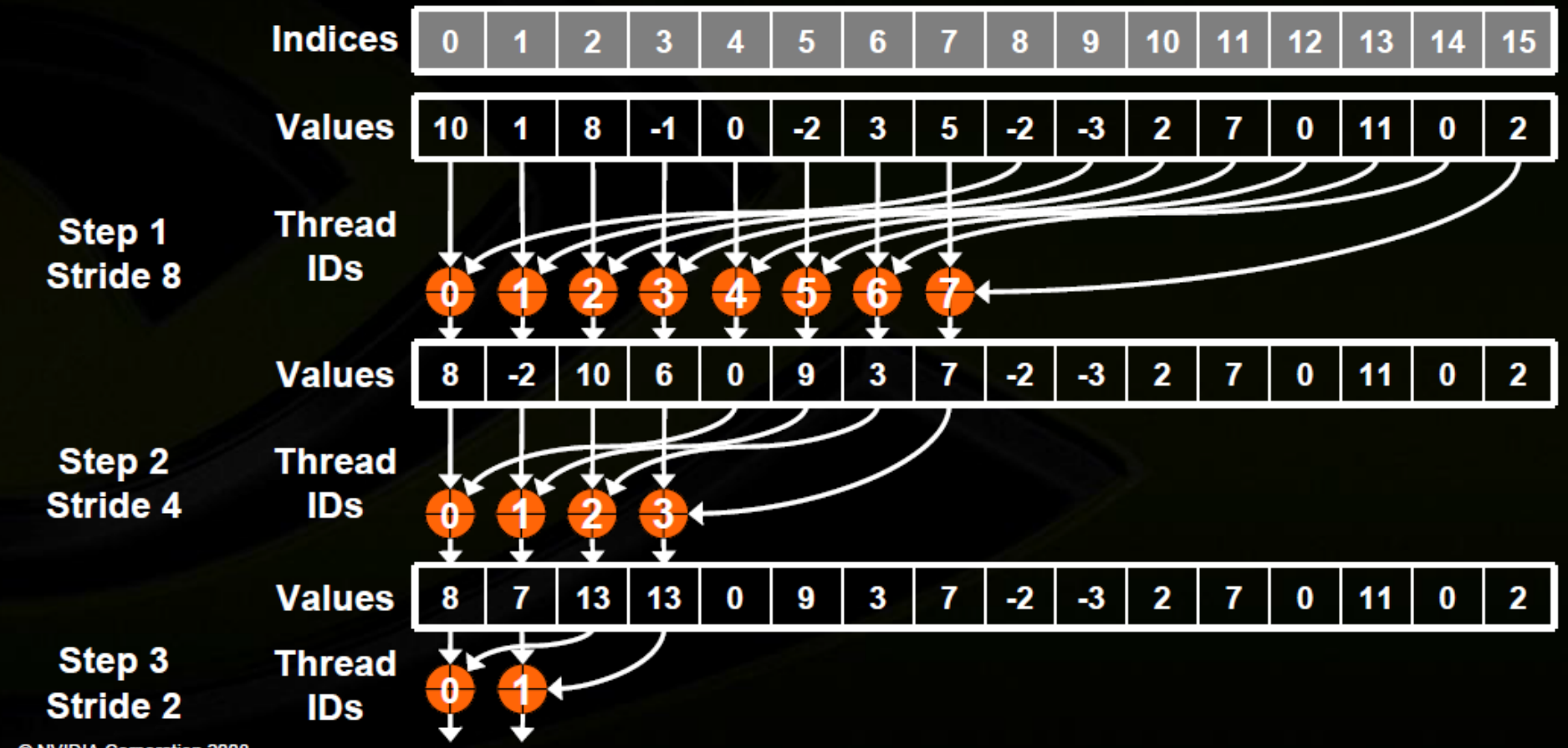
Memory Access

- Hardware detour: banks of memory
  - Fast parallel access, if indexed right

Coalesced memory access!

Next examples by Cyril Zeller (NVIDIA)





Memory Hierarchy

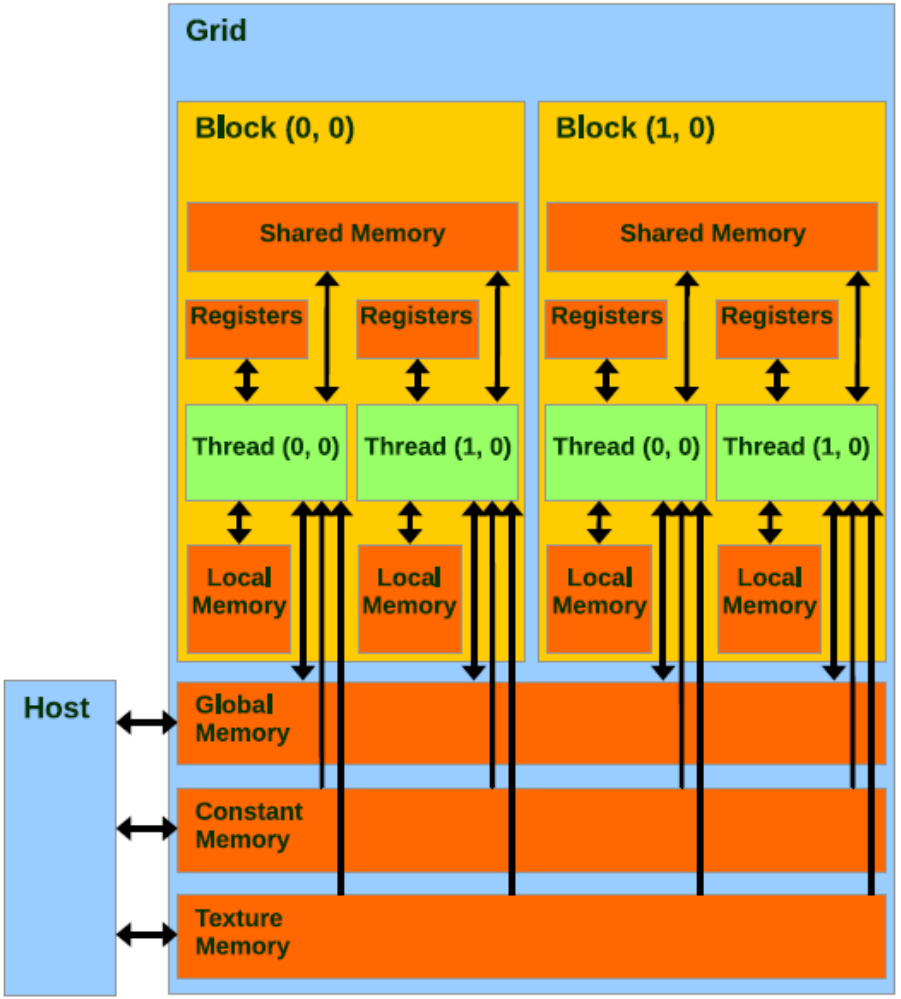
- Global memory
  - Grid
- Shared memory
  - Block
- Registers
  - Thread

Memory Hierarchy

- L1 cache
  - 48KB or 16KB per block – automatic speedup
- Constant memory
  - 64KB readonly total memory for lookup tables
- Textures
  - Fast 1D, 2D, 3D access with interpolation…

Memory Access

- Global memory
  - Grid – coalesced access
- Shared memory
  - Block – can do coalesced
- Registers
  - Thread



Shared Memory

- Use the `__shared__` keyword
  - Read & write access 100x faster
  - 16KB or 48KB per block
  - Variable declaration in kernel
  - Static arrays, now also dynamic
  - See <<< more magic happens here >>>
- Need to synchronize threads!!

Synchronization in a Block

- Special function in kernel: `__syncthreads()`
  - Threads wait at this barrier for the last one(s)
  - Variants at higher levels of compute capability
    - Count, Bitwise OR/AND
  - Do not call it in conditional branches…

Atomic Operations

- Many variants: Inc, Add, And, …
  - Availability depends on compute capability level
  - See the Appendix in the Programming Guide
- Work on shared and global memory!
  - Grid: global access – slow if many collisions
  - Block: shared is faster access and less threads

Examples

http://pastebin.com/u/CUDA

GPU Access at JHU


```
$ ssh gpuclassN@gpulogin.hhpc.jhu.edu
$ passwd
$ cp –r ~budavari/Class .
$ nvcc –m 64 -arch sm_20 XYZ.cu
$ qsub … –q debug –I
```