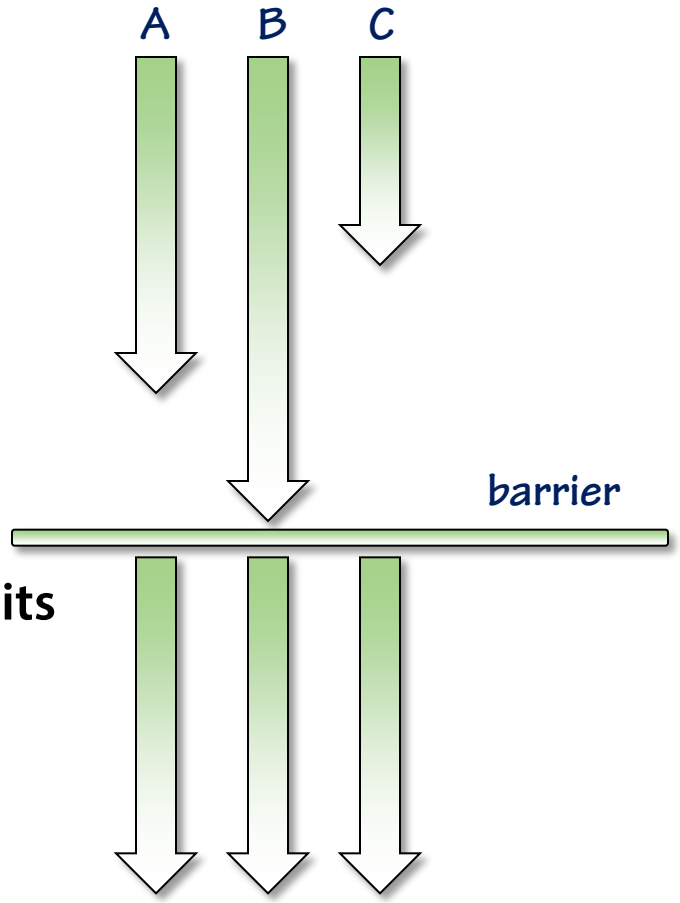
Thread Cooperation and Synchronization

Dmitri Nesteruk
@dnesteruk

pluralsight
hardcore developer training

Thread Synchronization

- Threads can take different amounts of time to complete a part of a computation
- Sometimes, you want all threads to reach a particular point before continuing their work
- CUDA provides a thread barrier function `__syncthreads()`
- A thread that calls `__syncthreads()` waits for other threads to reach this line
- Then, all threads can continue executing

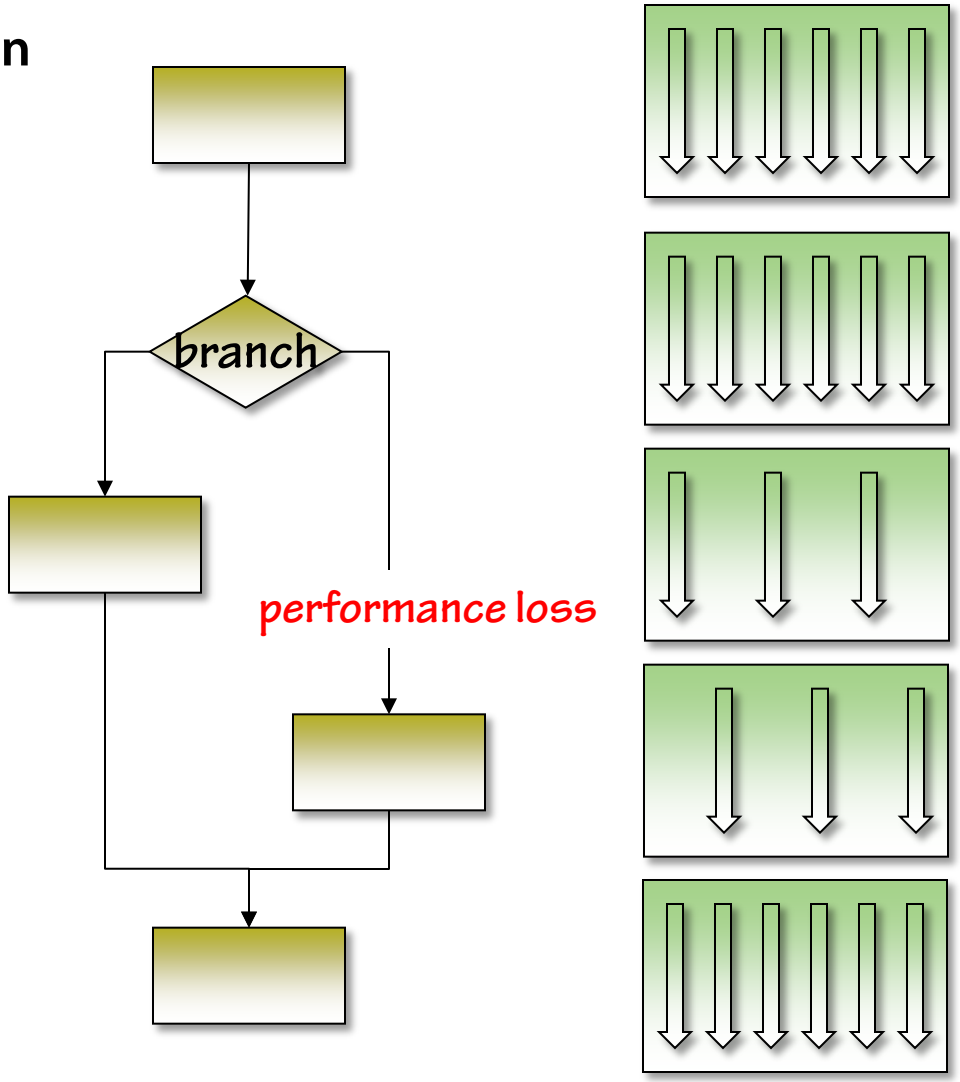


Restrictions

- `__syncthreads()` only synchronizes threads within a block
- A thread that calls `__syncthreads()` waits for other threads to reach this location
- All threads *must* reach `__syncthreads()`
  - ```
    if (x > 0.5)
    {
      __syncthreads(); // bad idea
    }
    ```
- Each call to __syncthreads() is unique
  - ```
    if (x > 0.5)
    {
      __syncthreads();
    } else {
      __syncthreads(); // also a bad idea
    }
    ```

Branch Divergence

- Once a block is assigned to an SM, it is split into several warps
- All threads within a warp must execute the same instruction at the same time
- I.e., `if` and `else` branches *cannot* be executed concurrently
- Avoid branching if possible
- Ensure `if` statements cut on warp boundaries



Summary

- **Threads are synchronized with __syncthreads()**
  - Block scope
  - All threads must reach the barrier
  - Each __syncthreads() creates a unique barrier
- **A block is split into warps**
  - All threads in a warp execute same instruction
  - Branching leads to warp divergence (performance loss)
  - Avoid branching or ensure branch cases fall in different warps