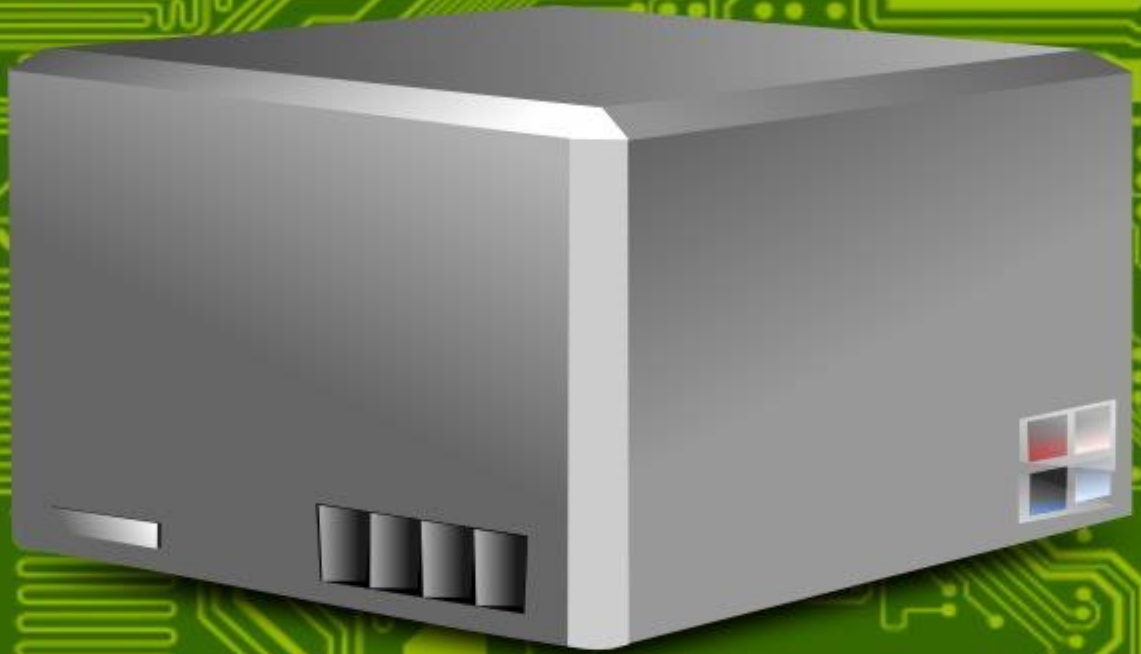
GPU Computing

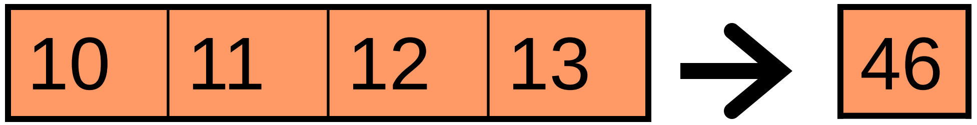


规约算法

Hui Liu
Email: hui.sc.liu@gmail.com

Reduction Operations

- Multiple values are reduced into a single value
  - ADD, MUL, AND, OR, ….

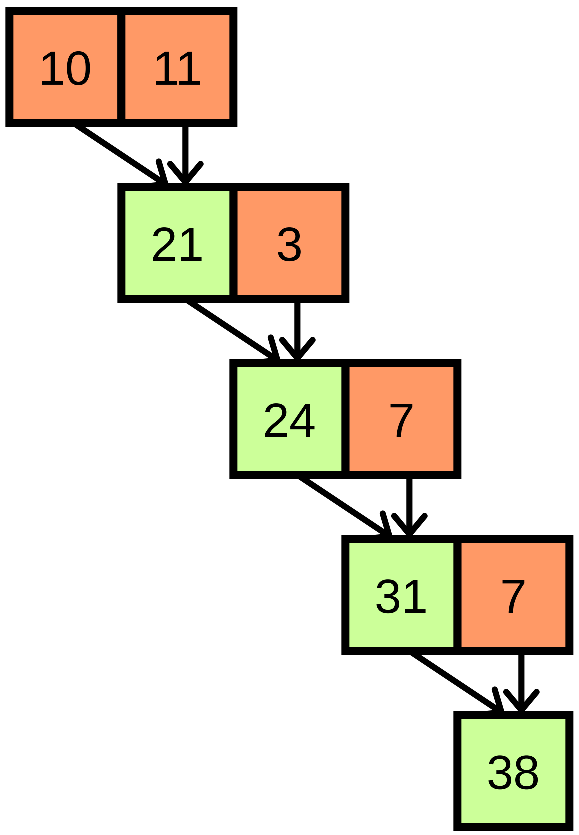


10 | 11 | 12 | 13 → 46

- Useful primitive

- Easy enough to allow us to focus on optimization techniques

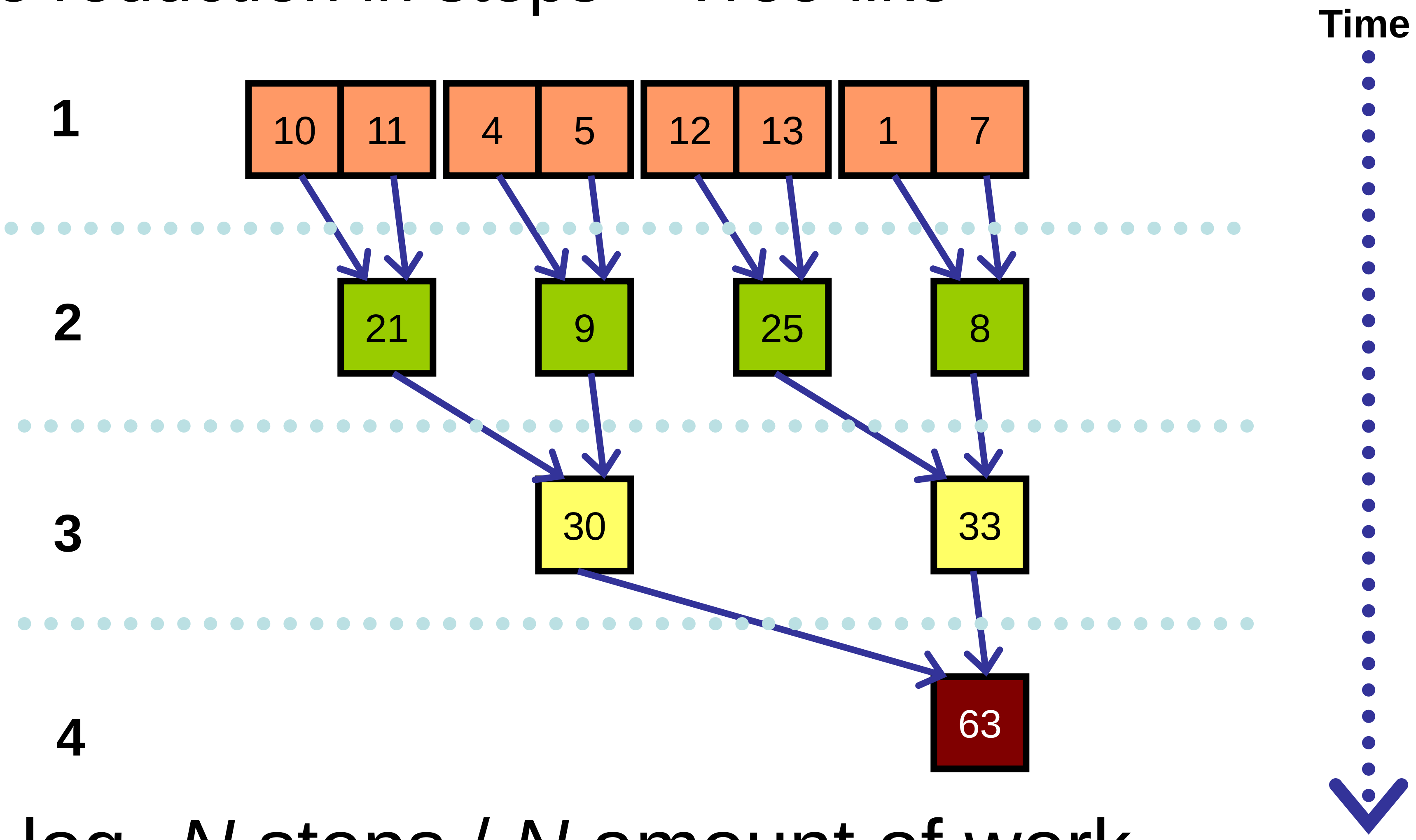
Sequential Reduction



- Start with the first two elements --> partial result
- Process the next element
- O(*N*)

Parallel Reduction

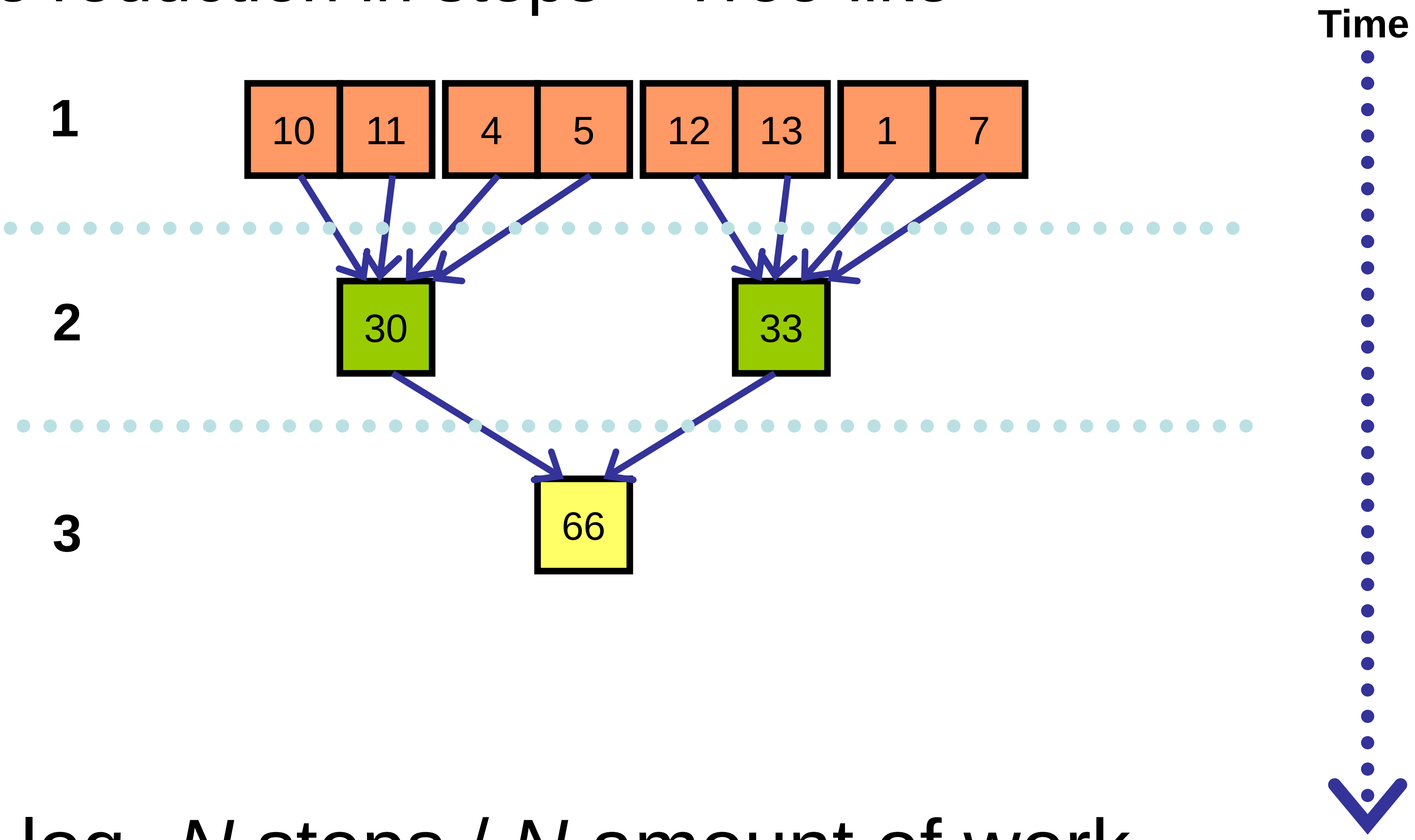
- Pair-wise reduction in steps – Tree-like



- $\log_2 N$ steps / N amount of work

Parallel Reduction – Different Degree-Trees possible

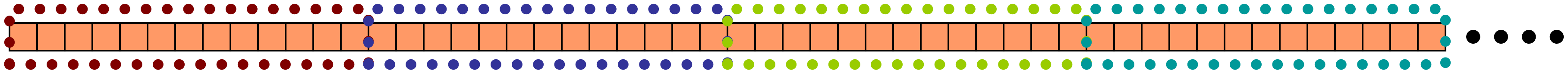
- Pair-wise reduction in steps – Tree-like



- $\log_4 N$ steps / N amount of work

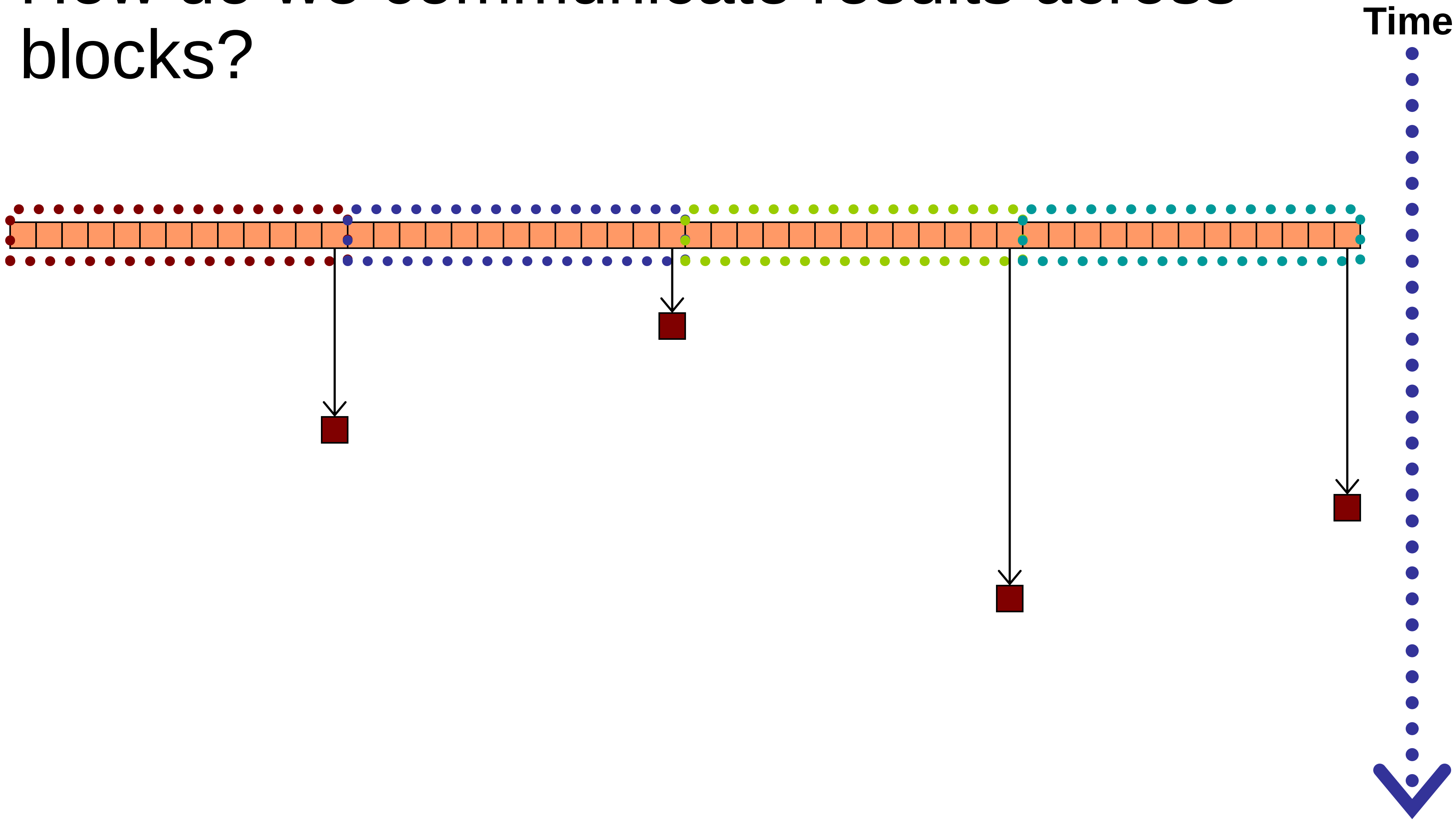
CUDA Strategy

- Single Block:
  - – Use Tree-Like approach
- Multiple Blocks?
  - – Not a necessity
    - one thread can always process many elements
    - But, will suffer from low utilization
  - – Utilize GPU resources
  - – Useful for large arrays
- Each block processes a portion of the input



How about multiple blocks

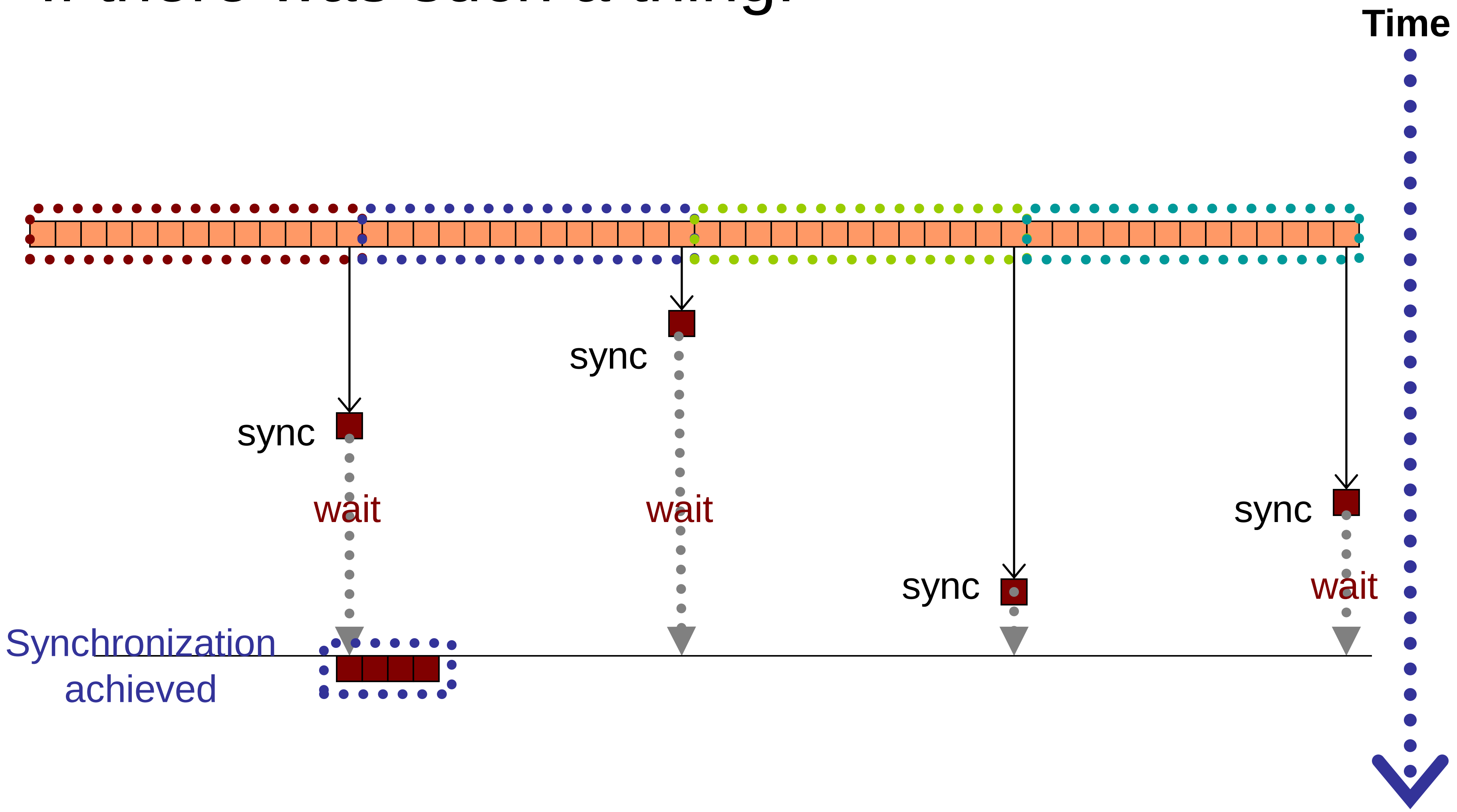
- How do we communicate results across blocks?



- The key problem is synchronization:
  - How do we know that each block has finished?

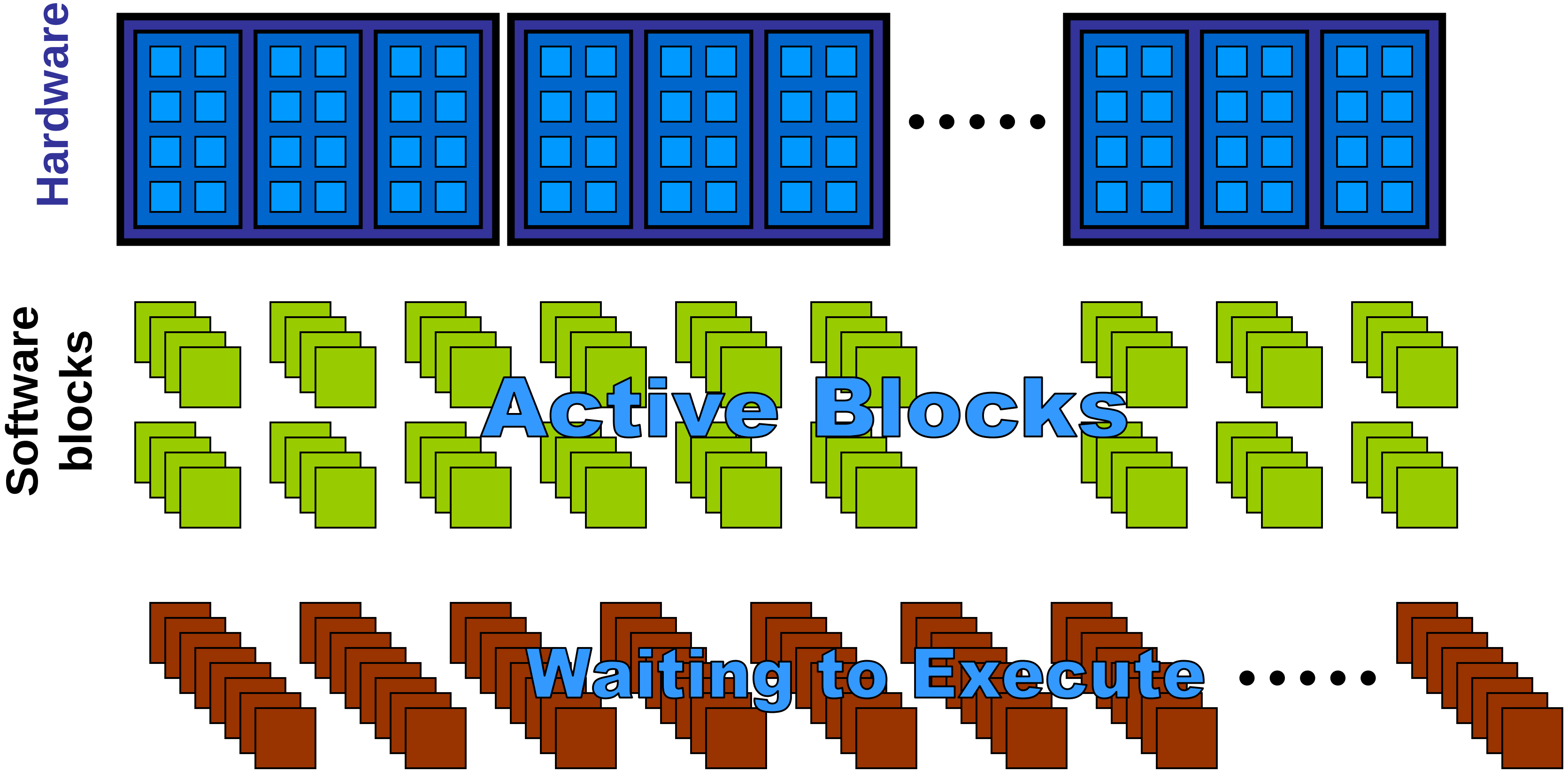
Global Synchronization

- If there was such a thing:

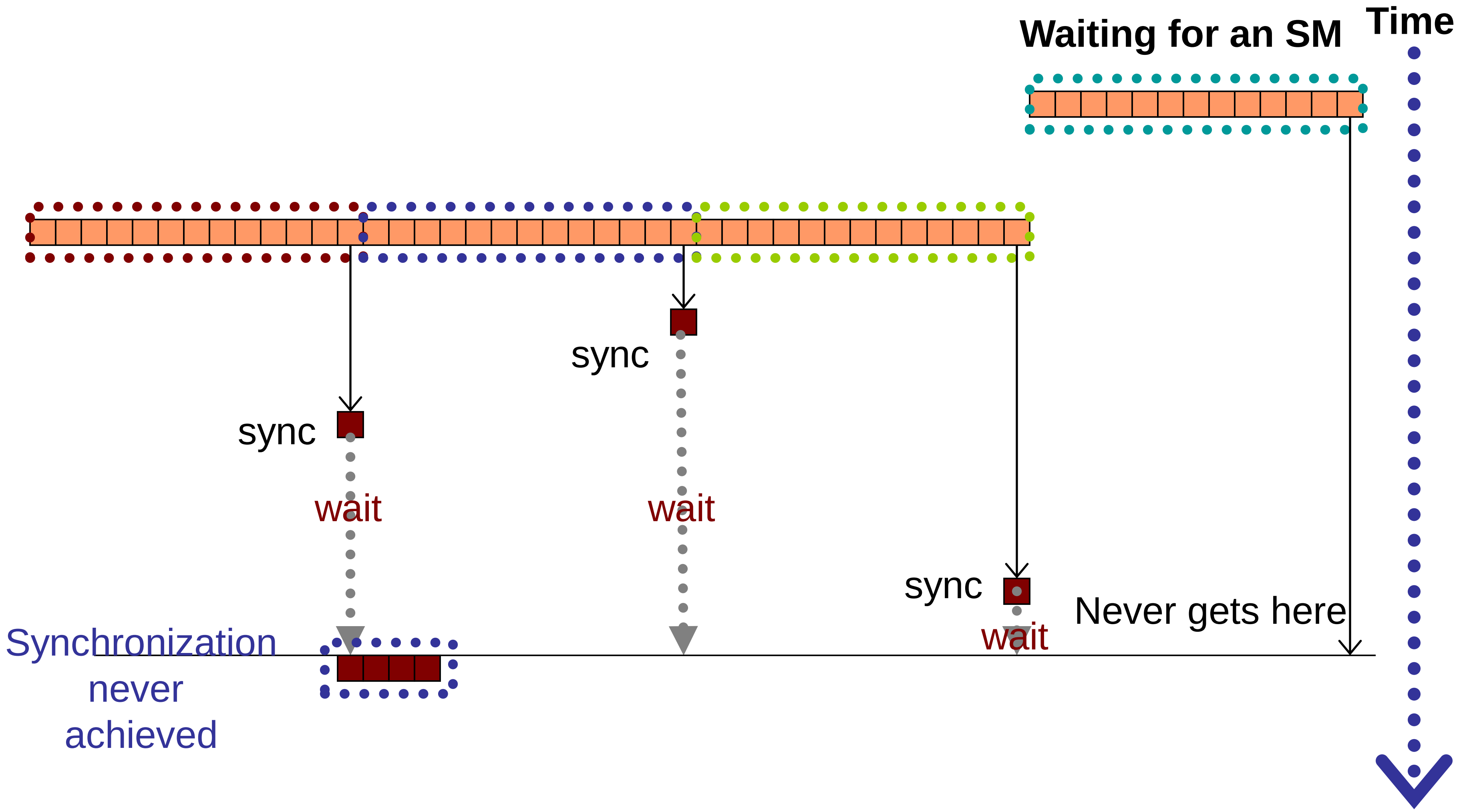


The Problem with Global Synchronization

- CUDA does not support it
  - One reason:
    - it's expensive to implement
  - Another reason:
    - "choose" between **limited blocks** or **deadlock**



Deadlock

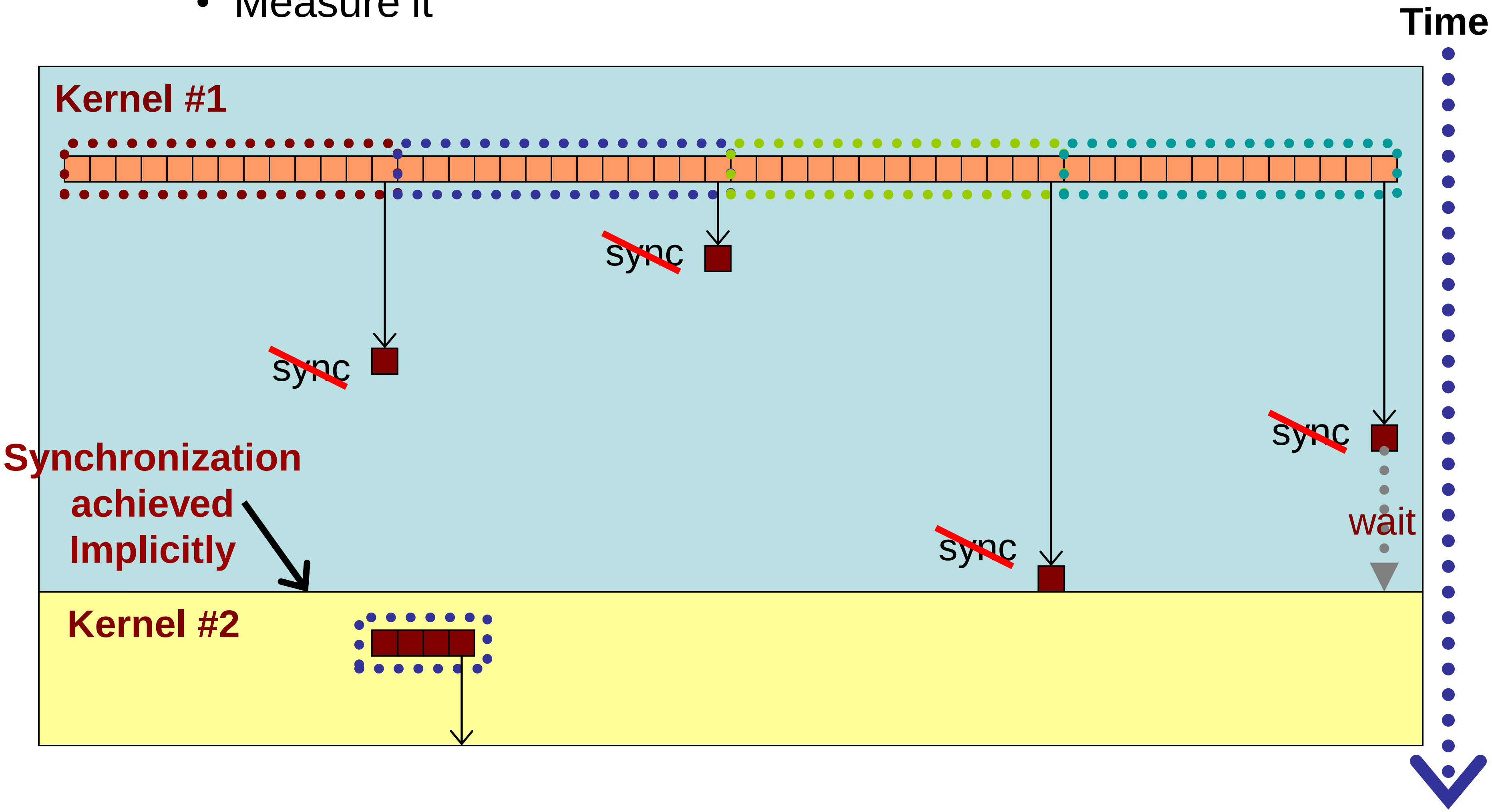

Waiting for an SM
Time
sync
sync
sync
wait
wait
wait
Never gets here
Synchronization
never
achieved

The Problem with Global Synchronization / Summary

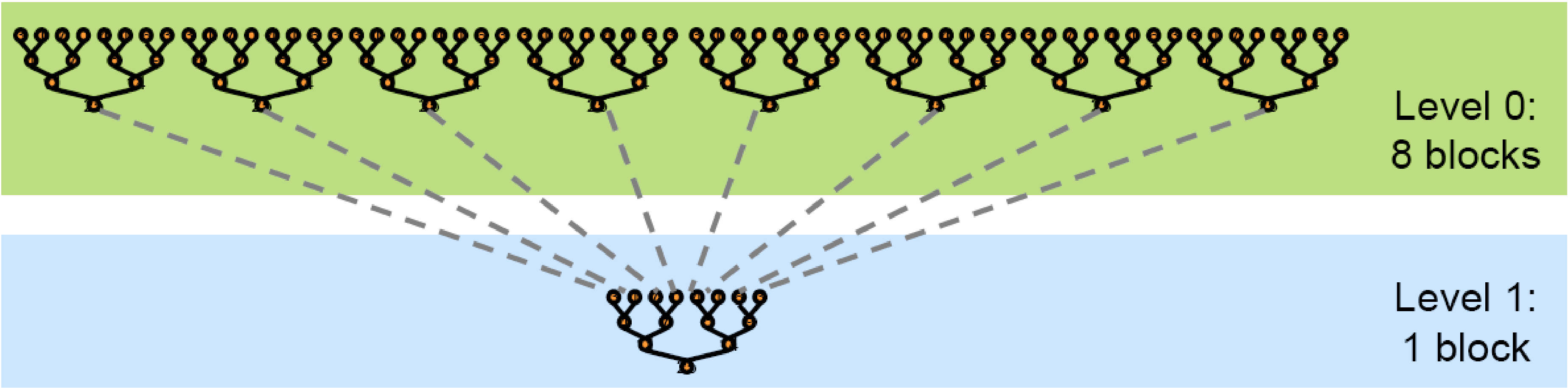
- If there was global sync
  - Global sync after each block
  - Once all blocks are done, continue recursively
- CUDA does not have global sync:
  - Expensive to support
  - Would limit the number of blocks
  - Otherwise deadlock will occur
    - Once a block gets assigned to an SM it stays there
    - Each SM can take only 8 blocks
    - So, at most #SMs x 8 blocks could be active at any given point of time
- Solution: Decompose into multiple kernels

Decomposing into Multiple Kernels

- Implicit Synchronization between kernel invocations
  - Overhead of launching a new kernel non-negligible
    - Don't know how much
    - Measure it



Reduction: Big Picture



- **The code for all levels is the same**
- **The same kernel code can be called multiple times**

THANK YOU