High Performance Computing in Julia

from the ground up.

CUDA Kernel Programming

10/10

CUDA Programming Model

- CUDA provides a model for partitioning a workload into small units of work called **threads**
- This is a SIMT approach of “Single Instruction/Program Multiple Threads”. This program is called a **kernel**
- As the programmers, we have to **manually partition** our workload into threads which can perform operations in parallel
- Each thread will perform an operation dependent on its **index** – i.e. the id of the current thread

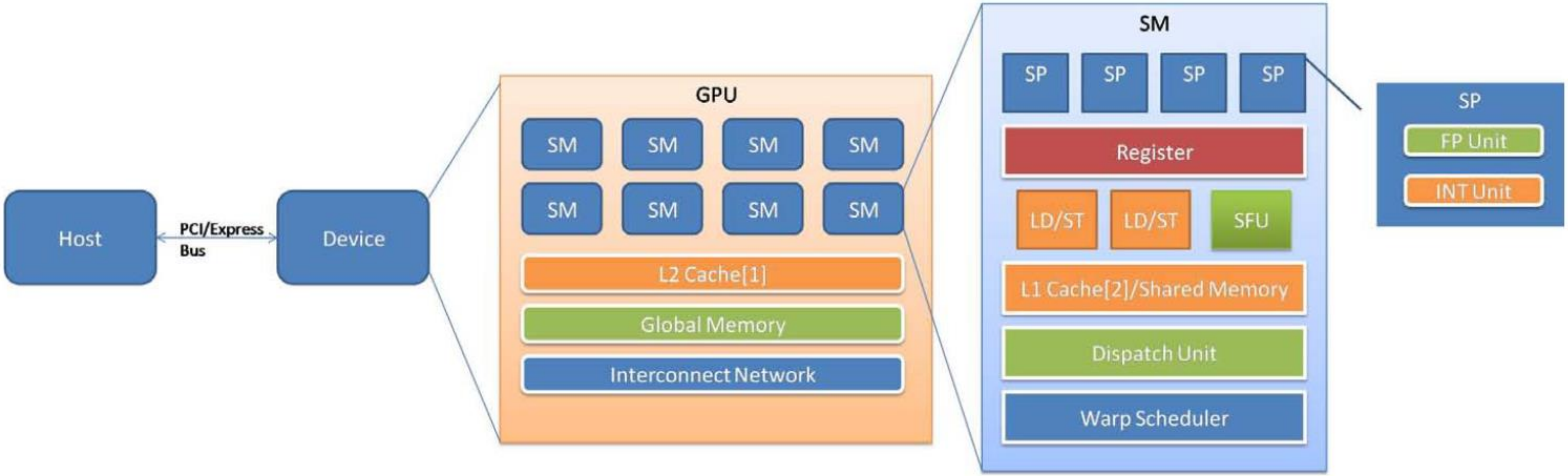
What is a kernel?

- A **kernel** is a **program/function** compiled for a device like a GPU
- Previously, these were called **compute shaders**, as they originally came about from hijacking graphics shaders for performing compute

First CUDA Kernel

Live Demonstration

Fig 1: Sivalingam, Karthee "GPU Acceleration of a Theoretical Particle Physics Application"



Why do we need blocks of threads?

- A block of threads has access to the **same shared memory**
- Threads within a block can **synchronise** with one another
- Threads from **different blocks** both have access to global memory, but **does not** have access to the **same shared memory**

CUDA Indexing

- Each CUDA kernel has access to a set of labels to identify the current thread
- A kernel is mapped onto a **grid** which is a collection of **blocks**, where each **block** contains a group of **threads**
- Each **block** has a 3D index, specifying the position in the grid
- Each **thread** within a **block** has a 3D index, specifying the position in the **block**

CUDA Indexing (1D)

https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf

CUDA Indexing (1D)

Index:	1	2	3	4	5	6	7	8

https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf

CUDA Indexing (1D)

Index:	1	2	3	4	5	6	7	8
Thread Index:	1	2	3	4	1	2	3	4

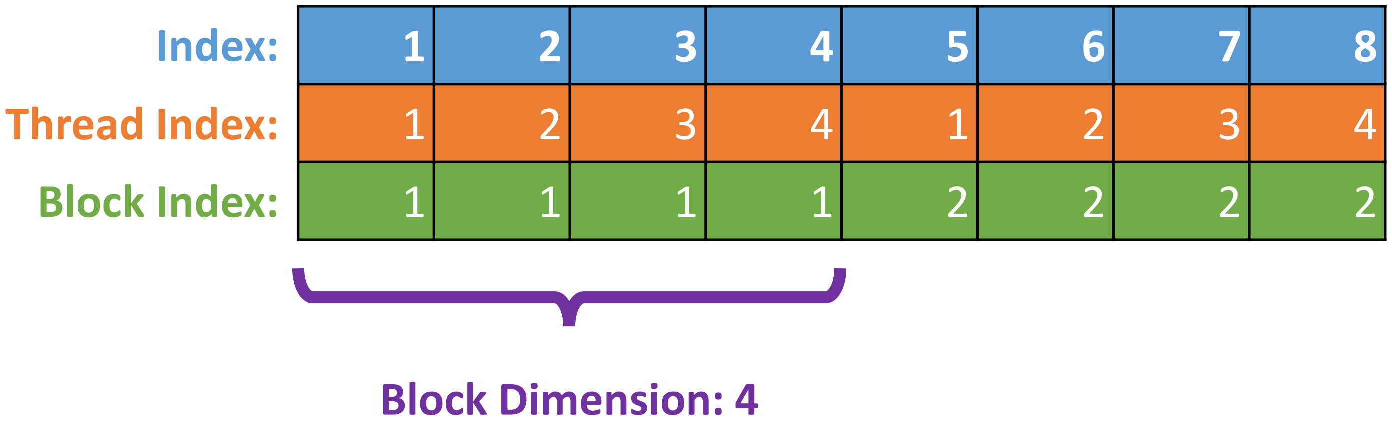
https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf

CUDA Indexing (1D)

Index:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Thread Index:	1	2	3	4	1	2	3	4
Block Index:	1	1	1	1	2	2	2	2

https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf

CUDA Indexing (1D)

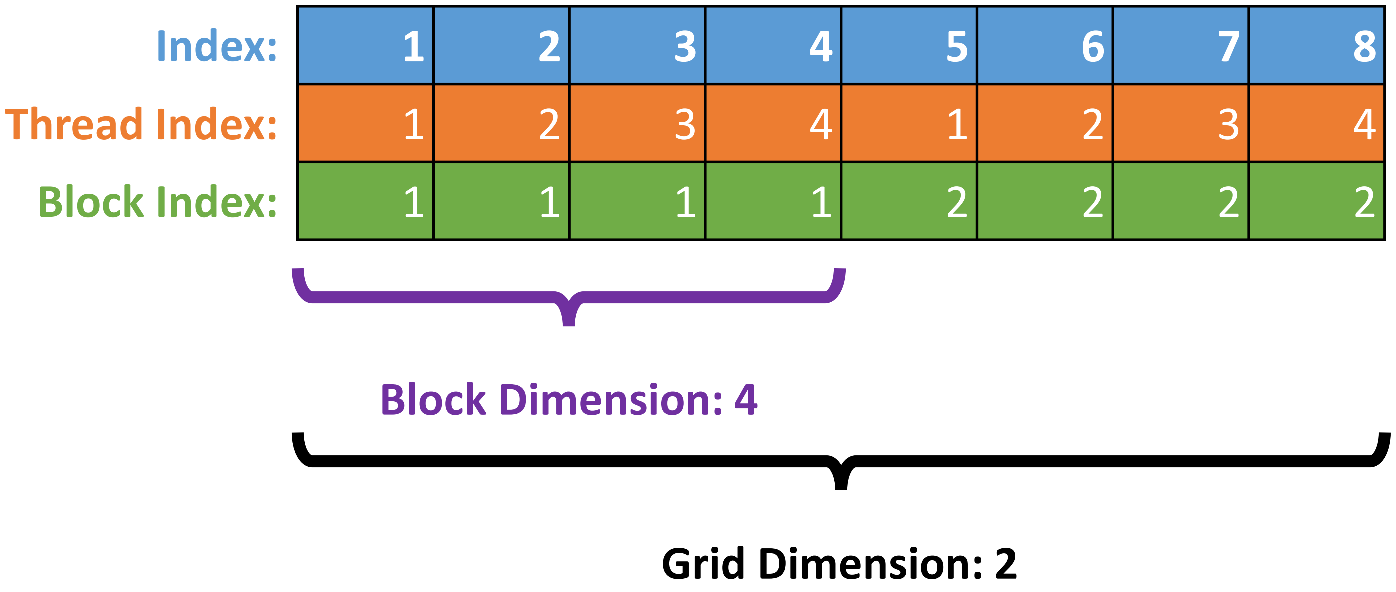


Index:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Thread Index:	1	2	3	4	1	2	3	4
Block Index:	1	1	1	1	2	2	2	2

Block Dimension: 4

https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf

CUDA Indexing (1D)



Index:	1	2	3	4	5	6	7	8
Thread Index:	1	2	3	4	1	2	3	4
Block Index:	1	1	1	1	2	2	2	2

Block Dimension: 4

Grid Dimension: 2

https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf

CUDA Indexing (1D)

Thread Index

Block Index

Array Index

$$i = t + (b - 1)d_b$$

Block Dimension

Index:	1	2	3	4	5	6	7	8
Thread Index:	1	2	3	4	1	2	3	4
Block Index:	1	1	1	1	2	2	2	2

Block Dimension: 4

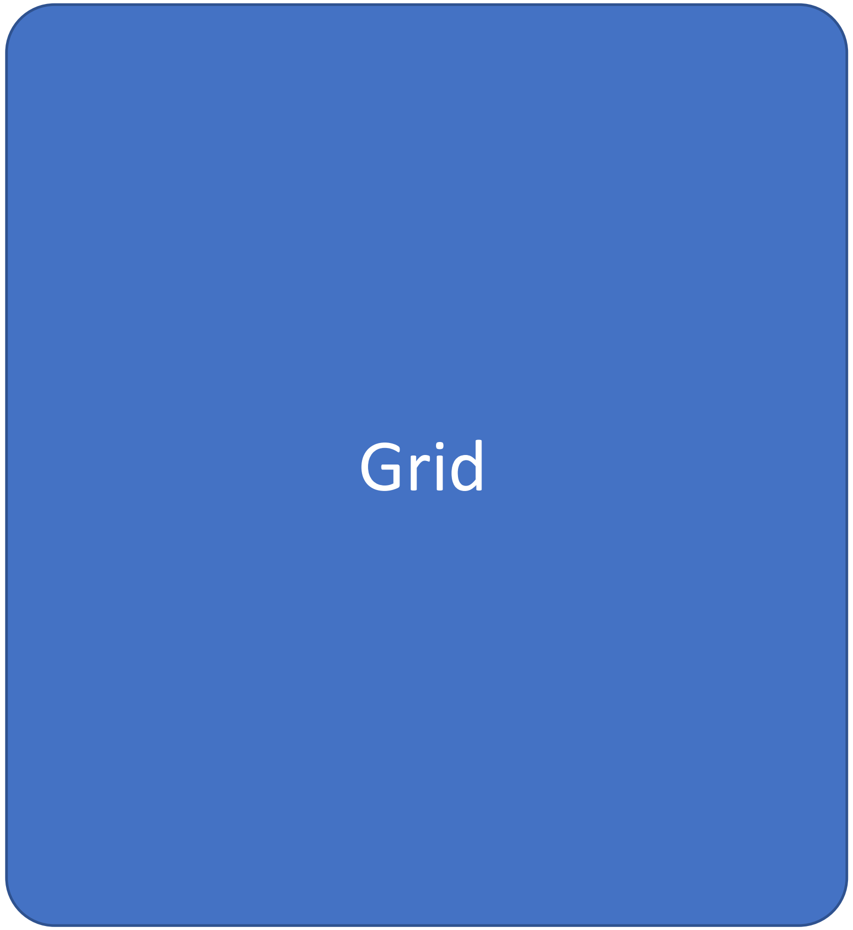
Grid Dimension: 2

https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf

First CUDA Kernel (Continued)

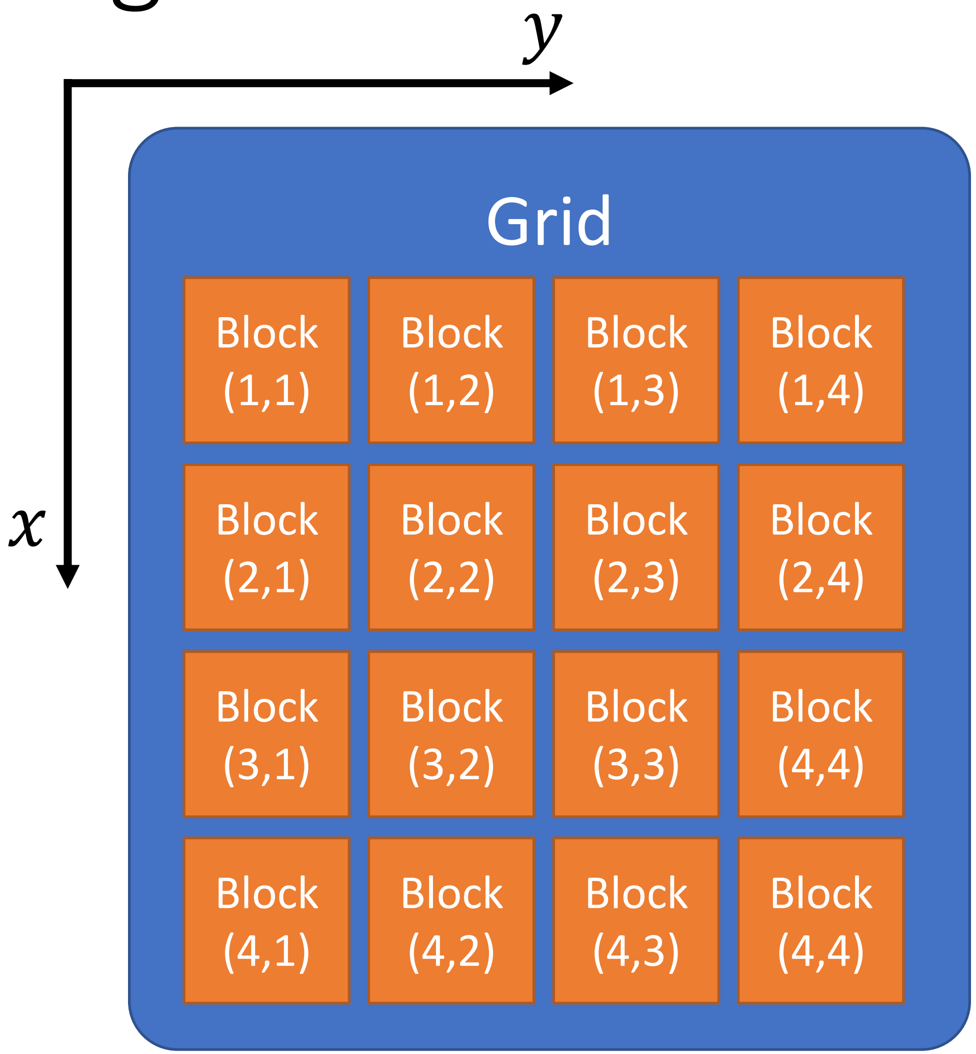
Live Demonstration

CUDA Indexing



Grid

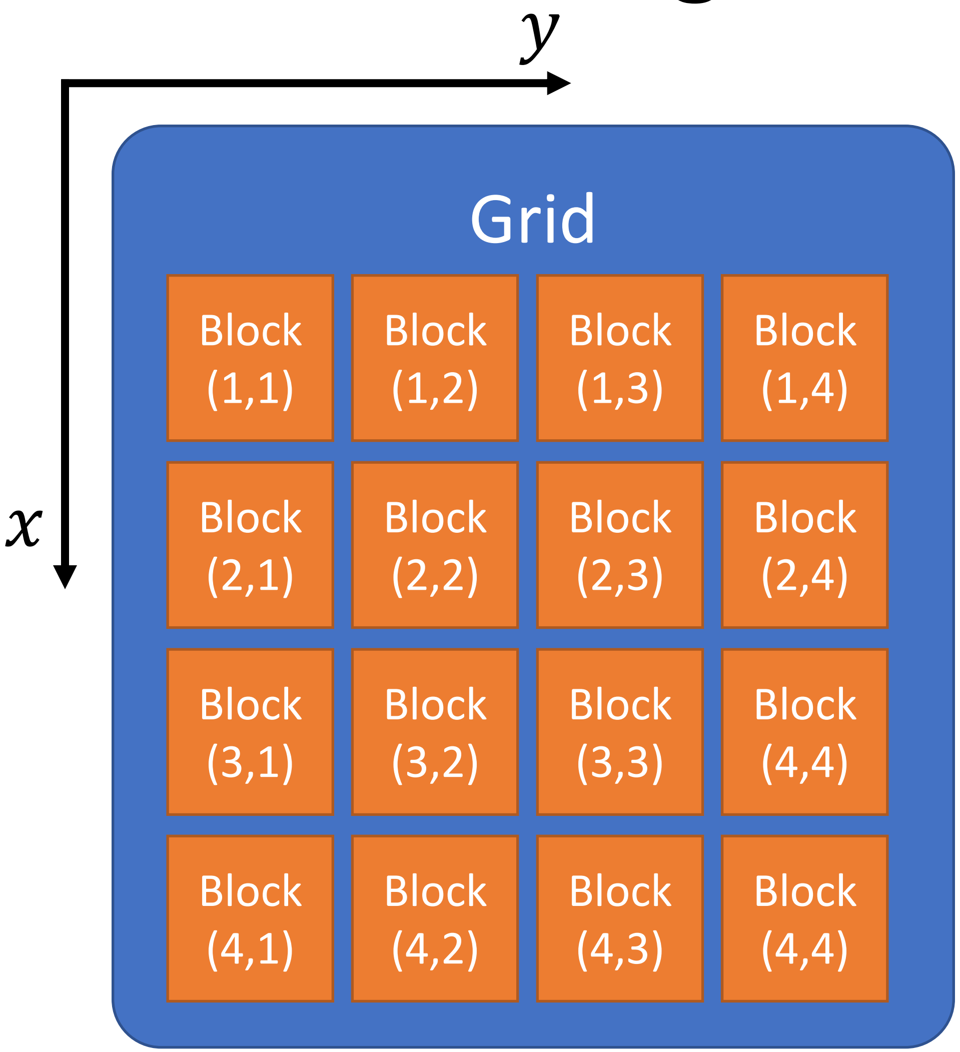
CUDA Indexing



Grid

Block (1,1)	Block (1,2)	Block (1,3)	Block (1,4)
Block (2,1)	Block (2,2)	Block (2,3)	Block (2,4)
Block (3,1)	Block (3,2)	Block (3,3)	Block (4,4)
Block (4,1)	Block (4,2)	Block (4,3)	Block (4,4)

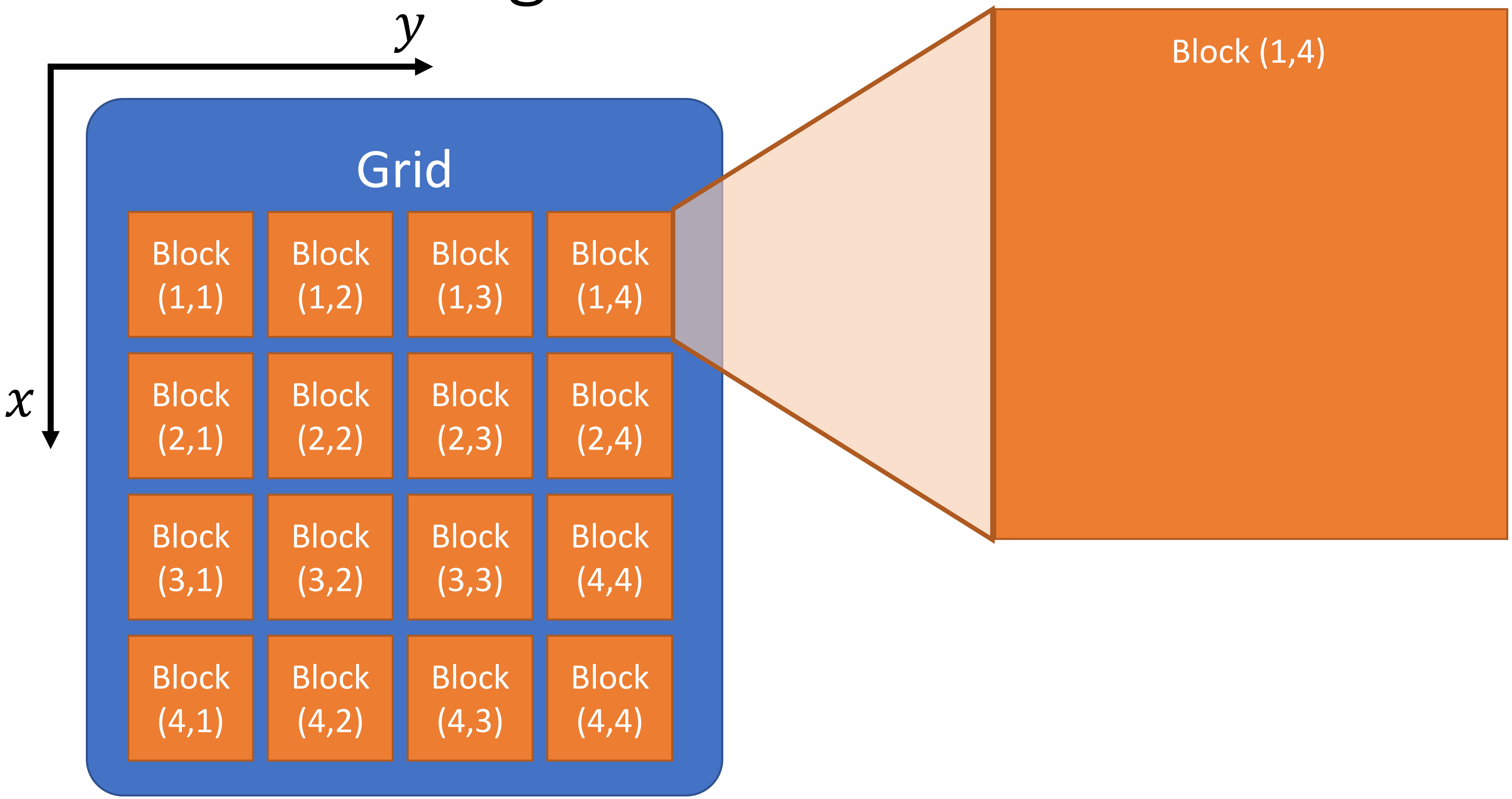
CUDA Indexing



Grid

Block (1,1)	Block (1,2)	Block (1,3)	Block (1,4)
Block (2,1)	Block (2,2)	Block (2,3)	Block (2,4)
Block (3,1)	Block (3,2)	Block (3,3)	Block (4,4)
Block (4,1)	Block (4,2)	Block (4,3)	Block (4,4)

CUDA Indexing

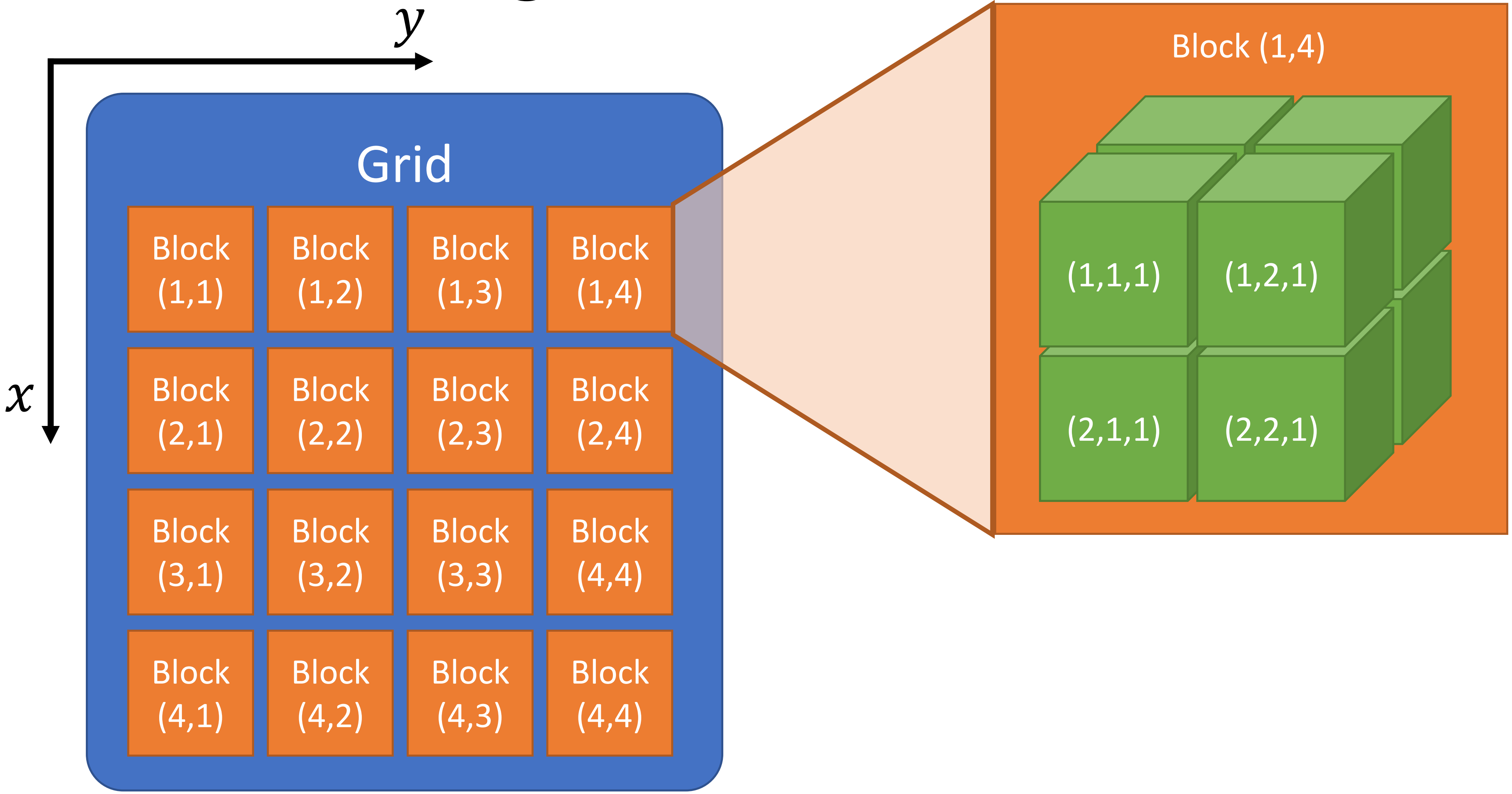


Grid

Block (1,1)	Block (1,2)	Block (1,3)	Block (1,4)
Block (2,1)	Block (2,2)	Block (2,3)	Block (2,4)
Block (3,1)	Block (3,2)	Block (3,3)	Block (4,4)
Block (4,1)	Block (4,2)	Block (4,3)	Block (4,4)

Block (1,4)

CUDA Indexing



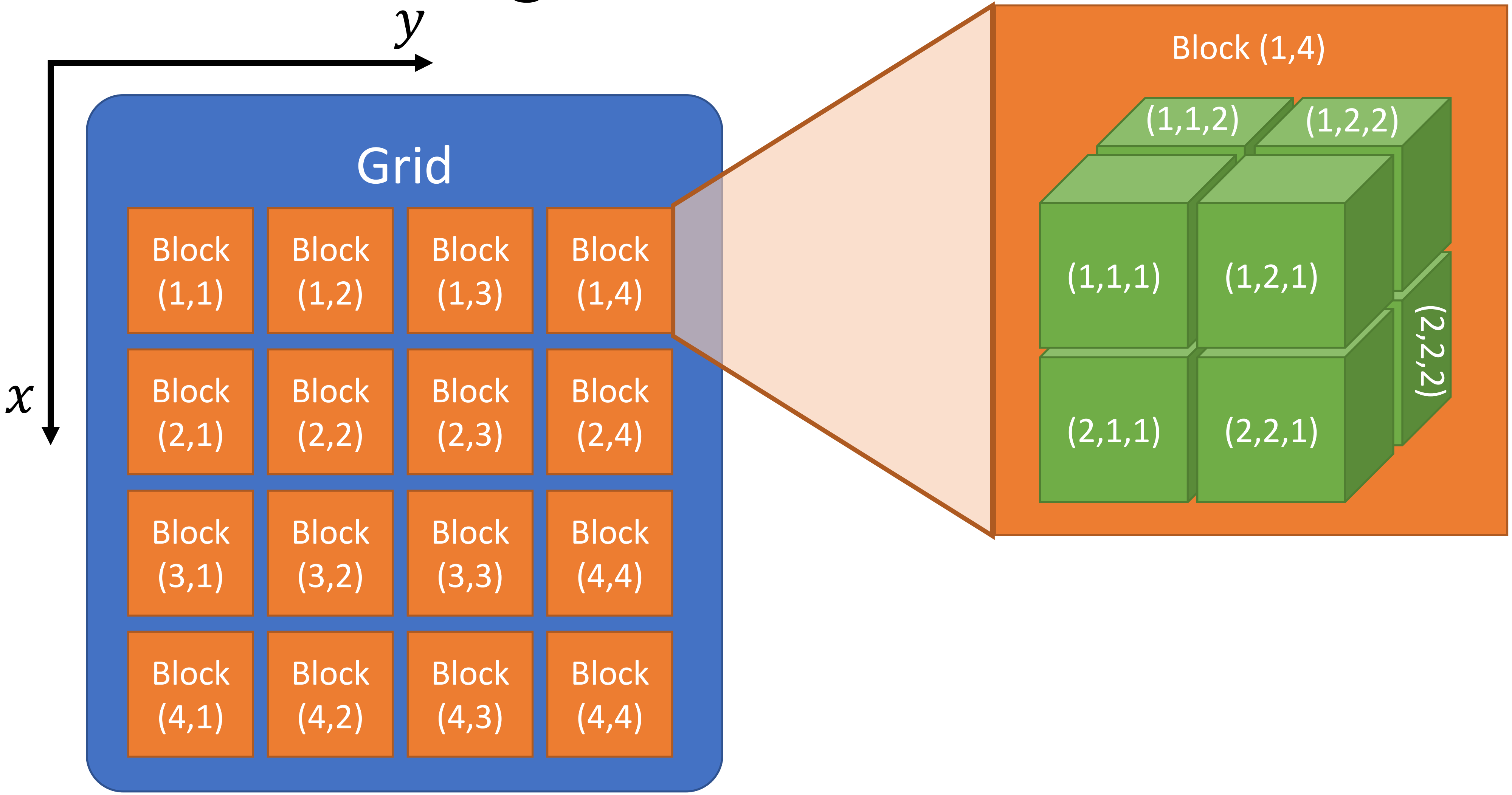
Grid

Block (1,1)	Block (1,2)	Block (1,3)	Block (1,4)
Block (2,1)	Block (2,2)	Block (2,3)	Block (2,4)
Block (3,1)	Block (3,2)	Block (3,3)	Block (4,4)
Block (4,1)	Block (4,2)	Block (4,3)	Block (4,4)

Block (1,4)

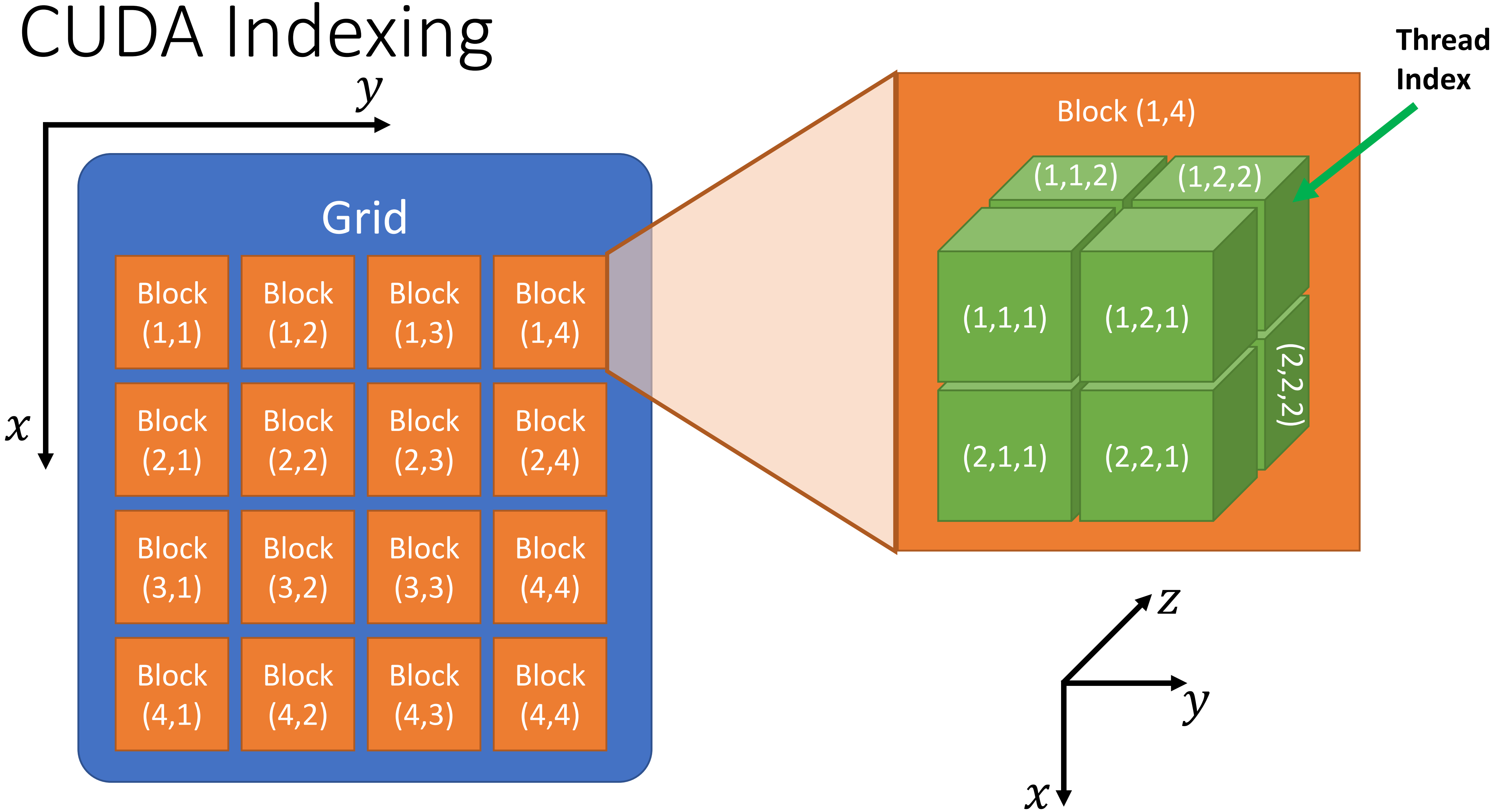
(1,1,1)	(1,2,1)
(2,1,1)	(2,2,1)

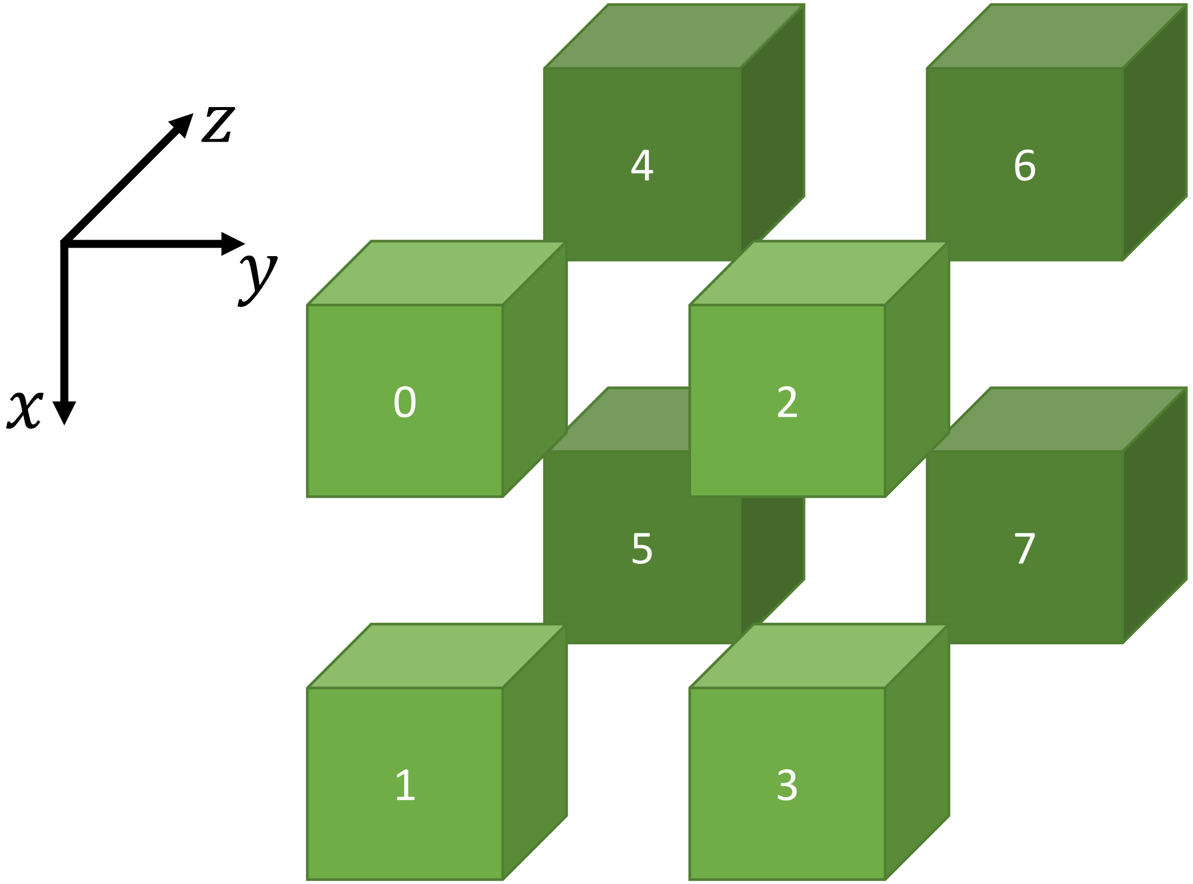
CUDA Indexing



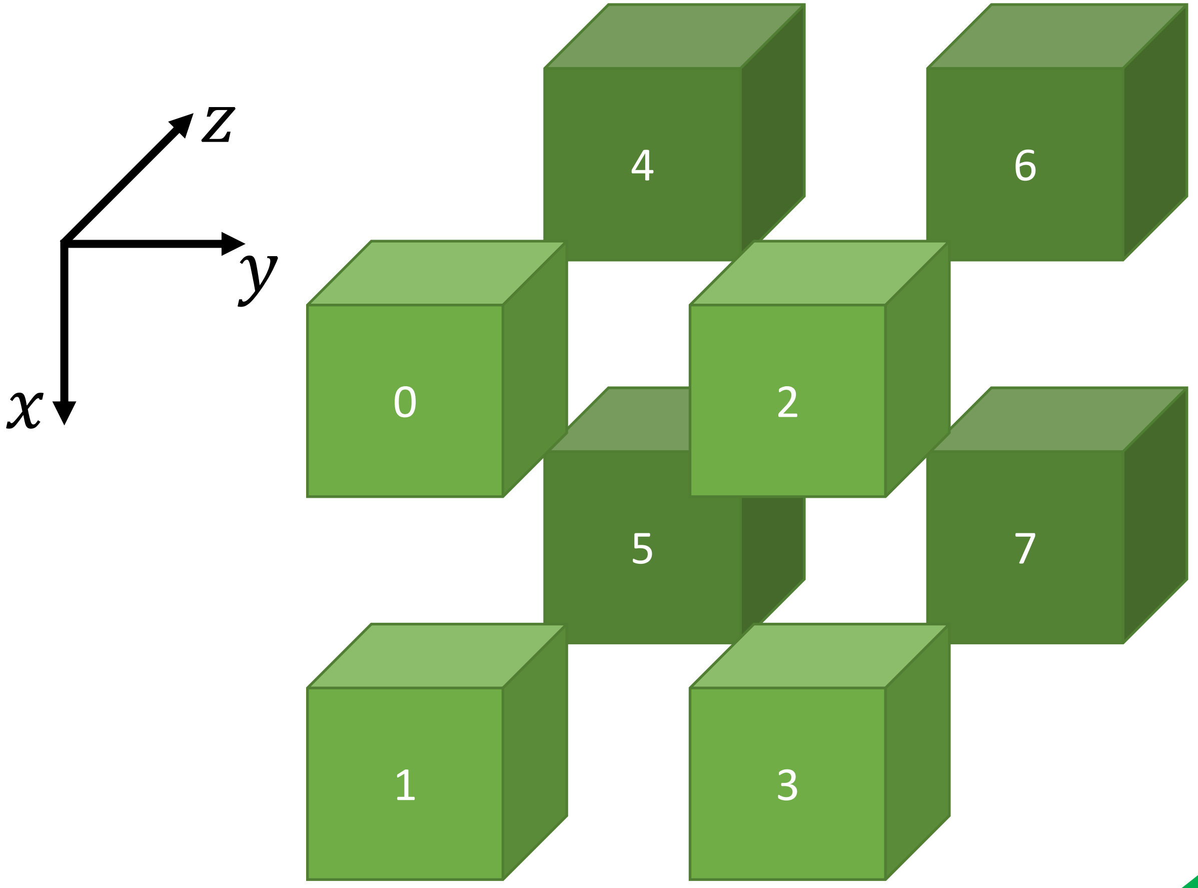
Block (1,1)	Block (1,2)	Block (1,3)	Block (1,4)
Block (2,1)	Block (2,2)	Block (2,3)	Block (2,4)
Block (3,1)	Block (3,2)	Block (3,3)	Block (4,4)
Block (4,1)	Block (4,2)	Block (4,3)	Block (4,4)

CUDA Indexing



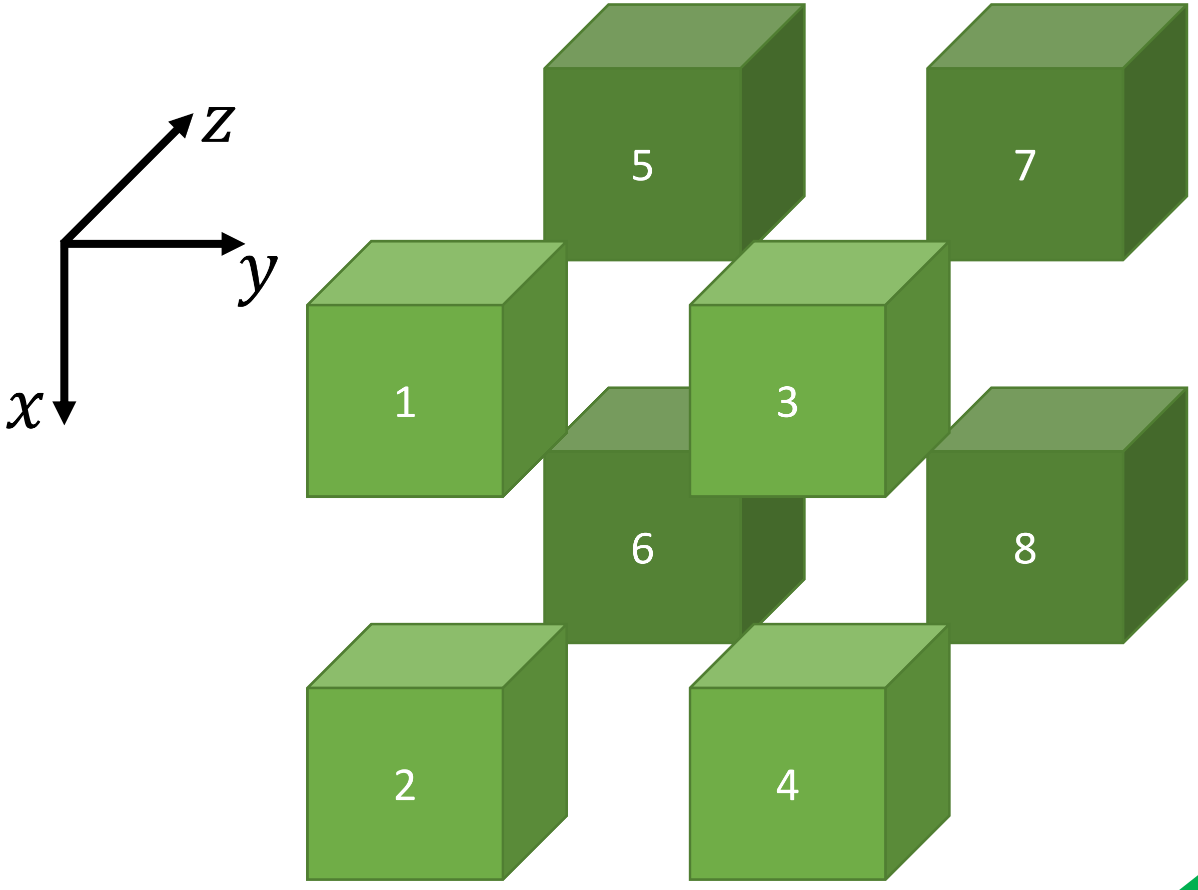


$$i = ?$$



Zero-Based Indexing

$$i = x + yd_x + zd_xd_y$$



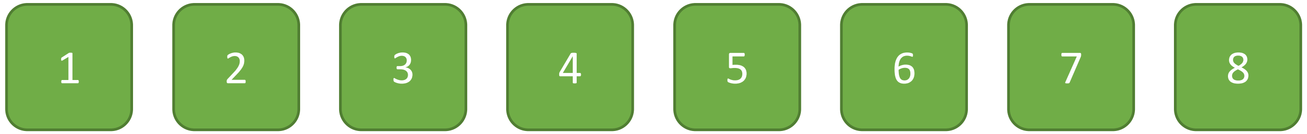
One-Based Indexing

$$i = x + (y - 1)d_x + (z - 1)d_x d_y$$

Shared Memory & Synchronisation

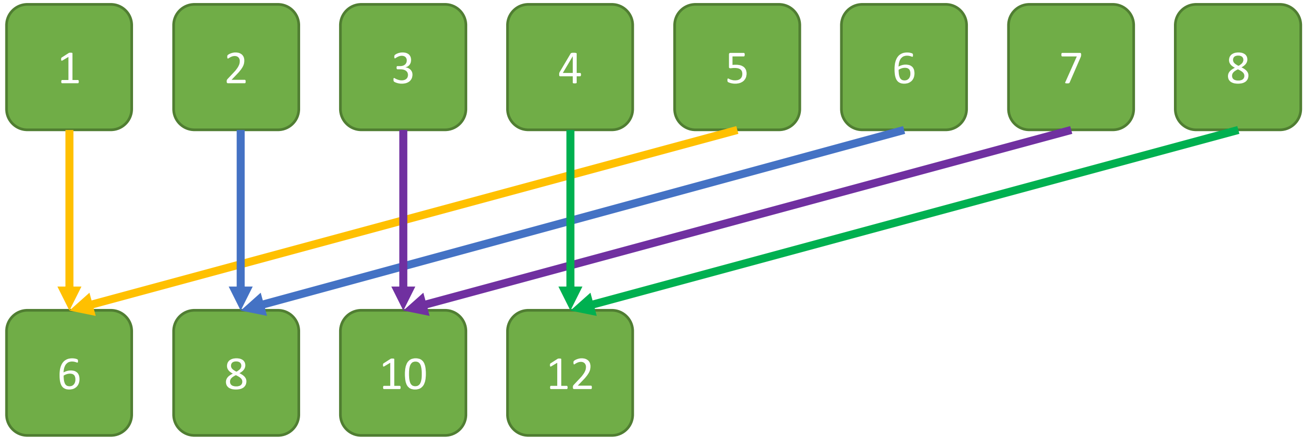
- Sometimes it is useful to have multiple threads have access to **shared memory**
- When multiple threads have access to shared memory – we introduce the threat of **race conditions**
- We need some **synchronisation** mechanisms to ensure correctness

Example: Reduction (Addition)

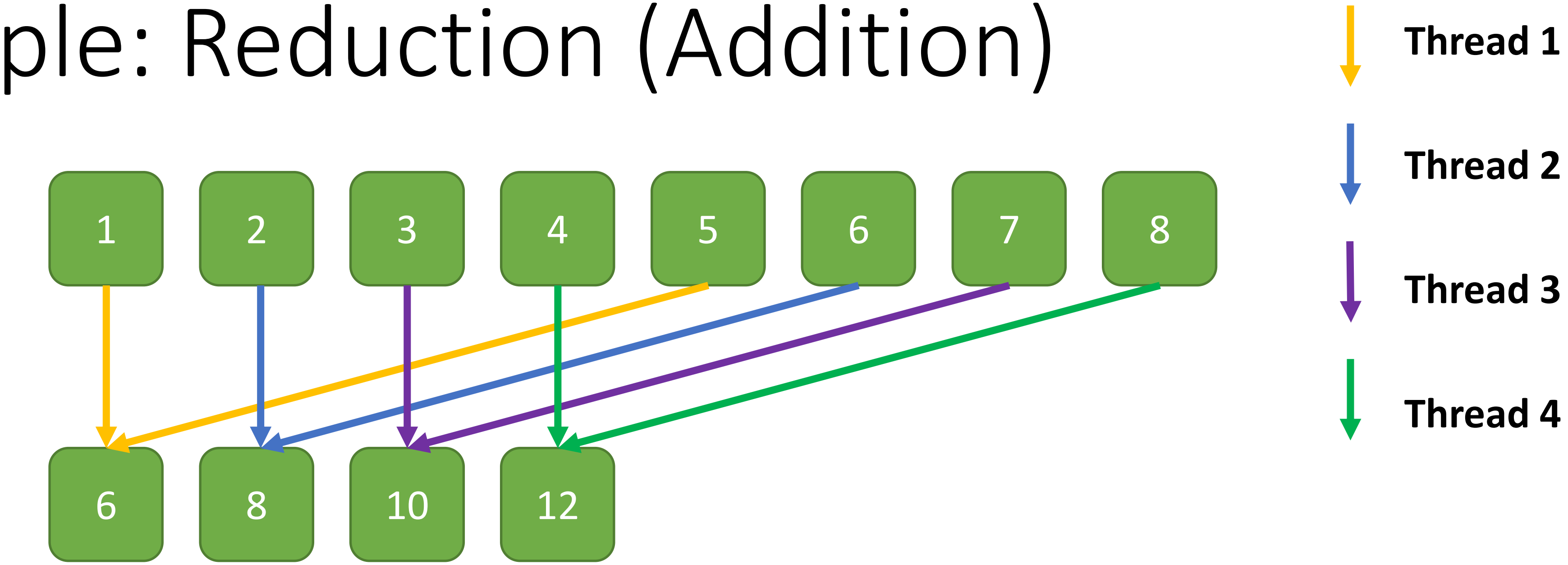


1 2 3 4 5 6 7 8

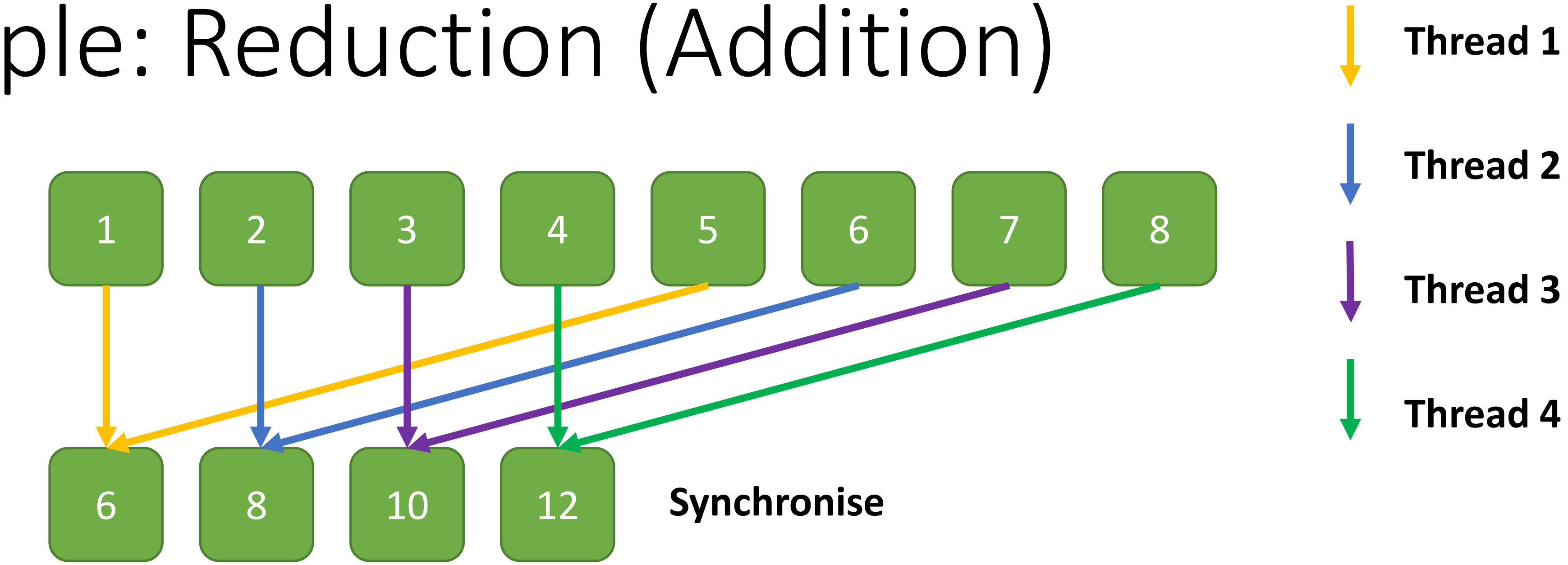
Example: Reduction (Addition)



Example: Reduction (Addition)

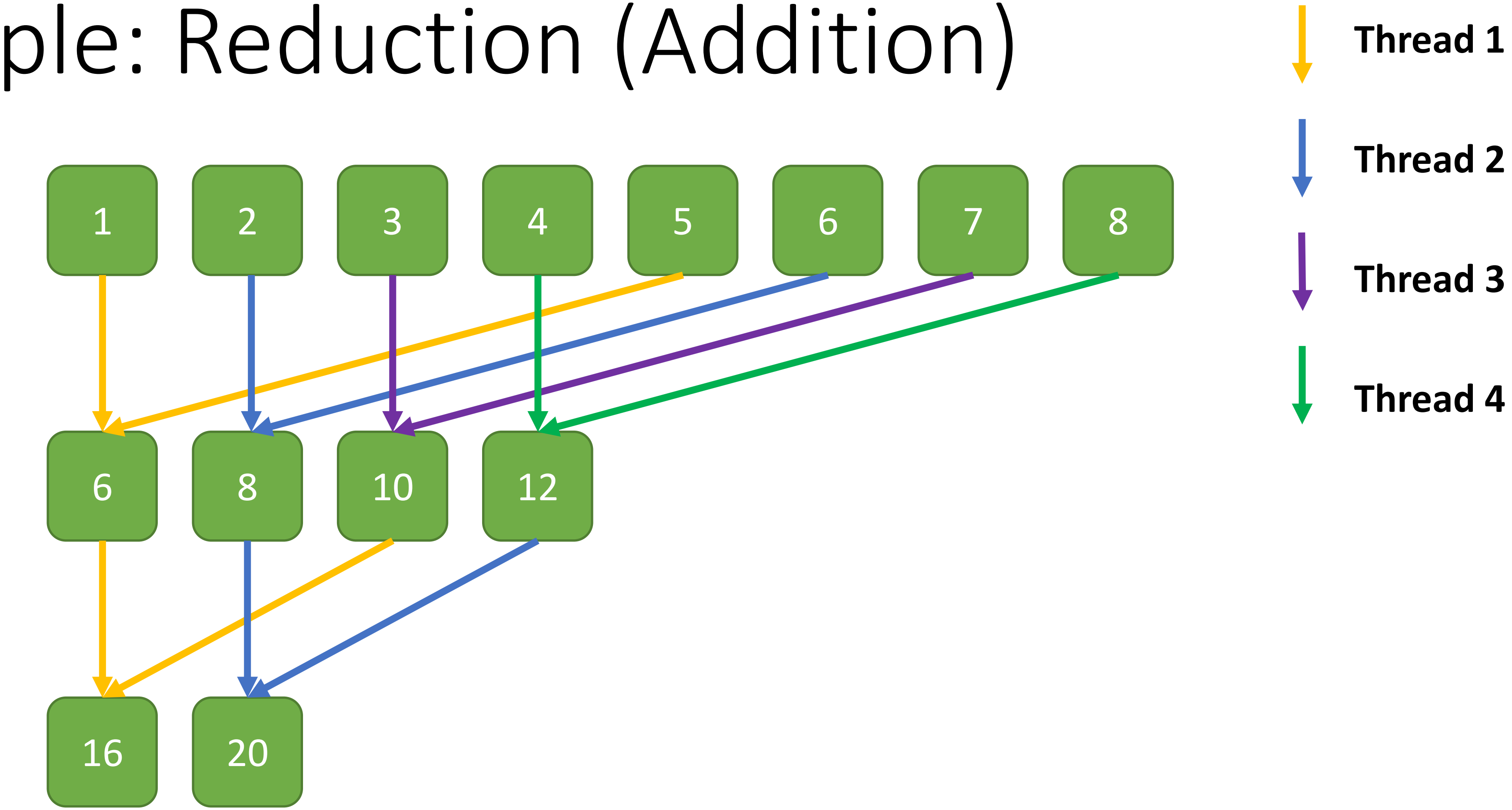


Example: Reduction (Addition)

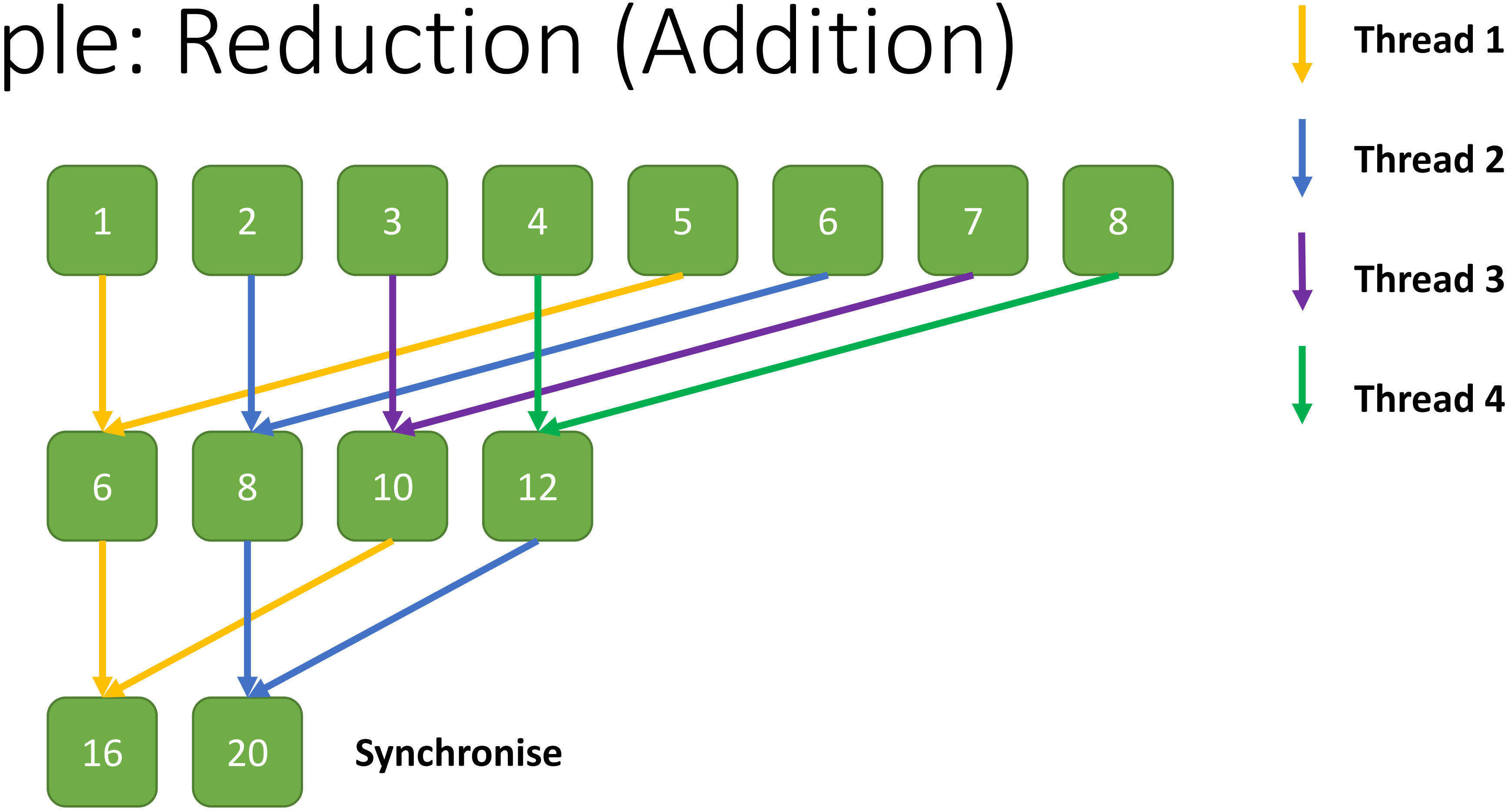


Synchronise

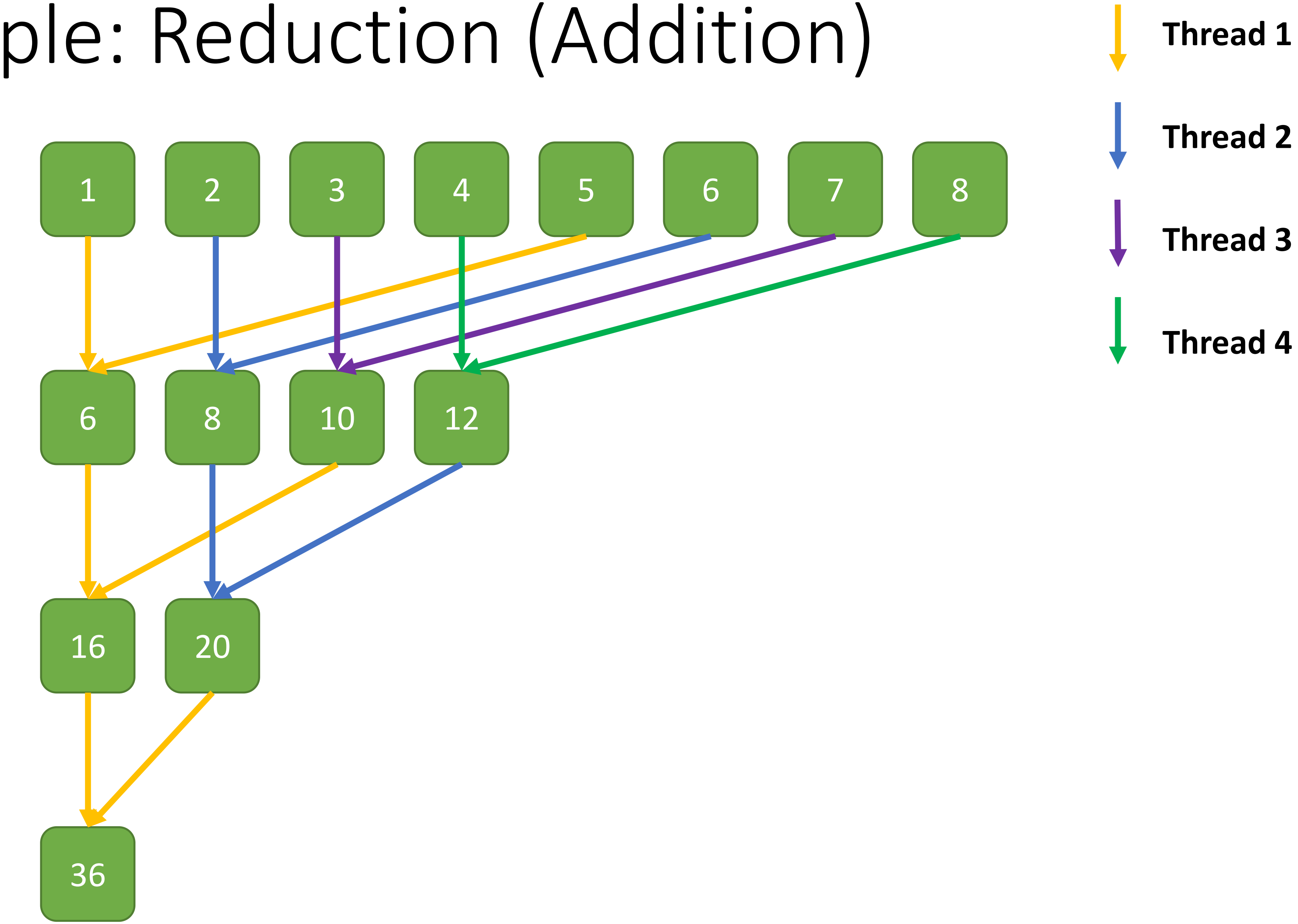
Example: Reduction (Addition)



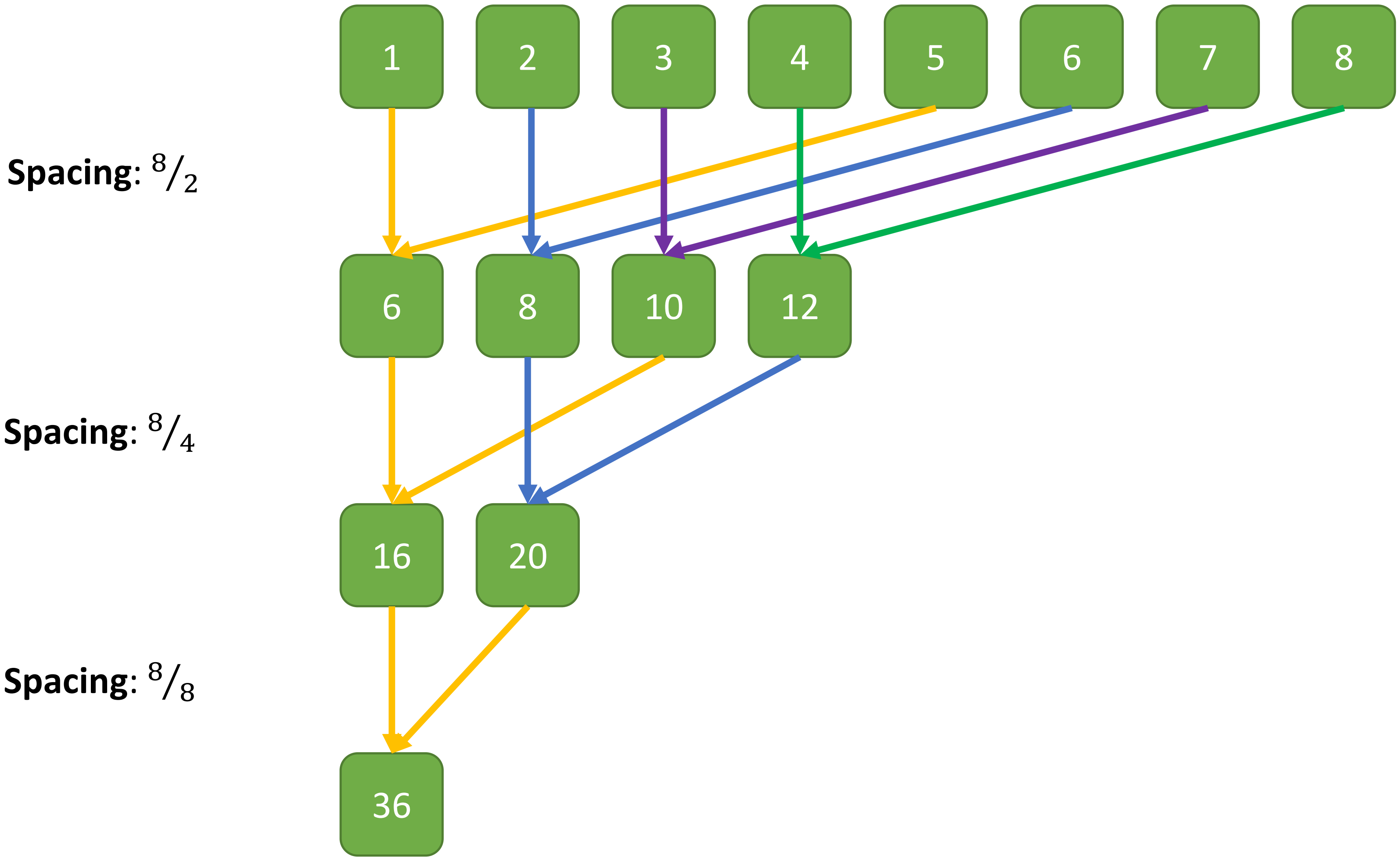
Example: Reduction (Addition)



Example: Reduction (Addition)



Example: Reduction (Addition)



Spacing: $^8/_2$

Spacing: $^8/_4$

Spacing: $^8/_8$

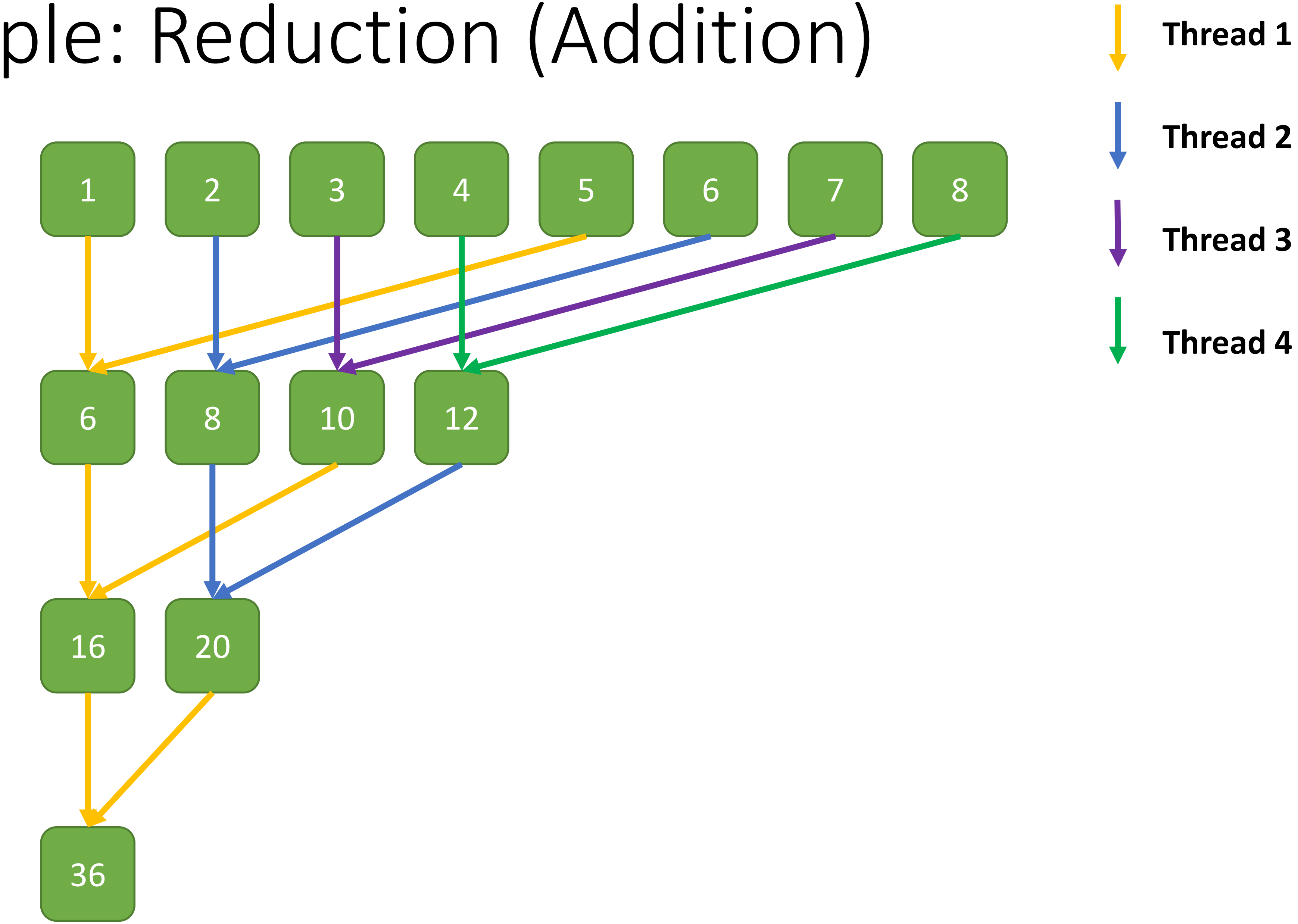
↓ **Thread 1**

↓ **Thread 2**

↓ **Thread 3**

↓ **Thread 4**

Example: Reduction (Addition)



Monte-Carlo π Estimation in CUDA

Live Demonstration

Further Resources

- “**CUDA by Example**” - https://developer.nvidia.com/cuda-example
  A book written by NVIDIA engineers. It is written for C, but the API for Julia is very similar, making the book more accessible
- “**GPU Programming in Julia**” - Workshop JuliaCon 2021 - https://www.youtube.com/watch?v=Hz9IMJuW5hU
- **Julia Discourse** - https://discourse.julialang.org/c/domain/gpu
- **Julia Slack** - https://julialang.org/slack/

Final Session

Assignment

https://classroom.github.com/a/q9ycWkI6

Task:

- Calculate the visualisation for the Julia set fractal using the GPU

Julia Set