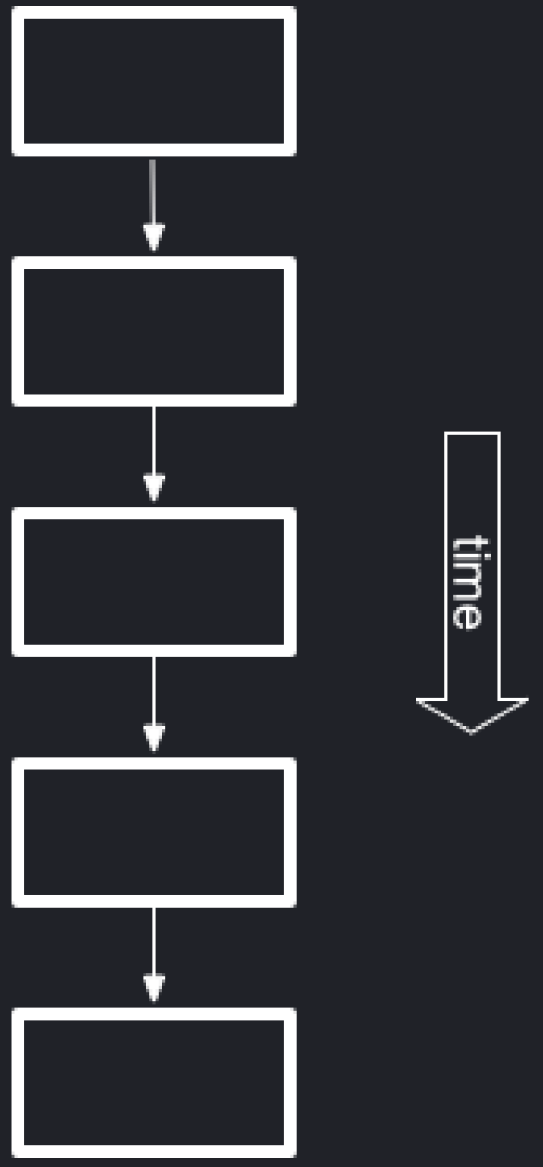
Quick Introduction to Parallelism

Mark Sherman, Ph.D.
Emmanuel College
July 14, 2020

Serial Execution

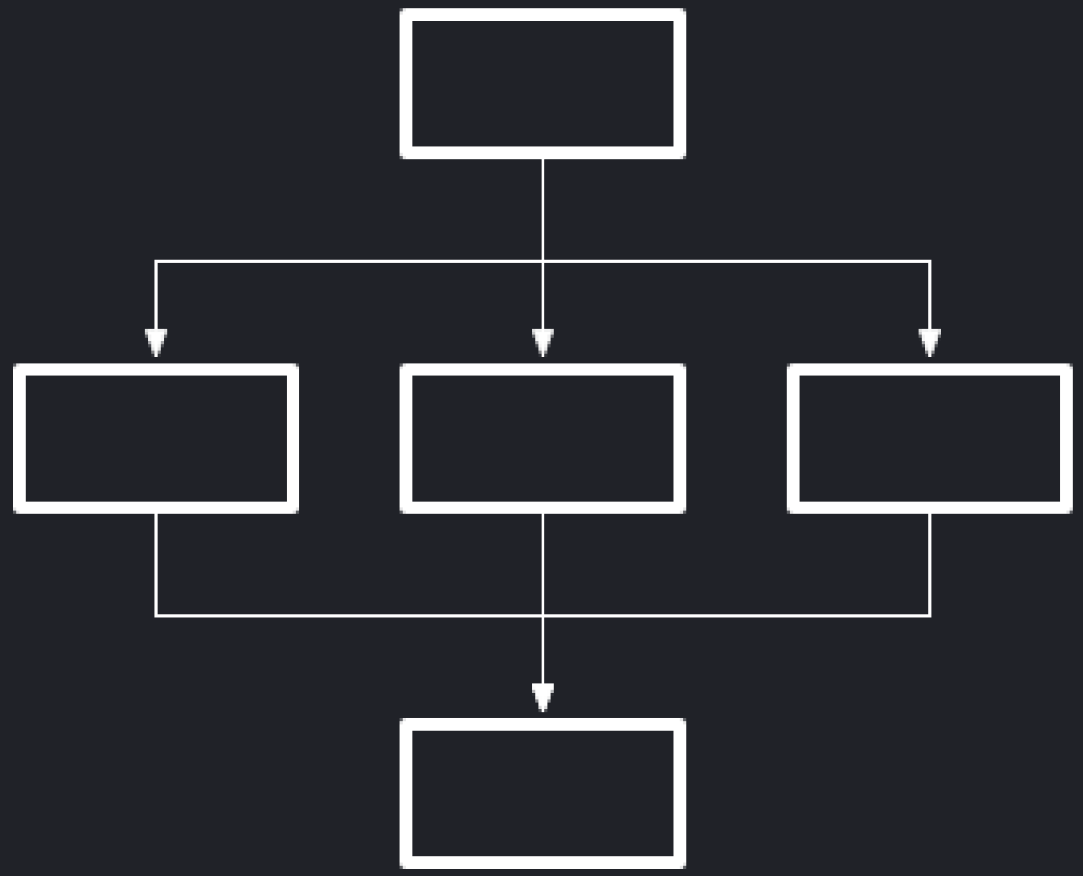
- Everything you know so far is *serial*
- also called *sequential* or *single-threaded*

- One thing happens at a time



Parallel Execution

- Does multiple things at the same time
- Combines into one result



Not Everything Works Well

- Some code can be broken up into parallel tasks easily
- Some code is fundamentally sequential

Which of these could be done easily in parallel?

```
let list = [1, 1];

for(let i = 2; i < 6; i++) {
    list[i] = list[i - 1] + list[i - 2]
}
```

```
let list = [ 1, 1, 2, 3, 5, 8 ];
let newList = [];

for(let i = 0; i < list.length; ++i) {
    newList[i] = list[i] * 2;
}
```

The second one.

```
let list = [ 1, 1, 2, 3, 5, 8 ];
let newList = [];

for(let i = 0; i < list.length; ++i) {
    newList[i] = list[i] * 2;
}
```

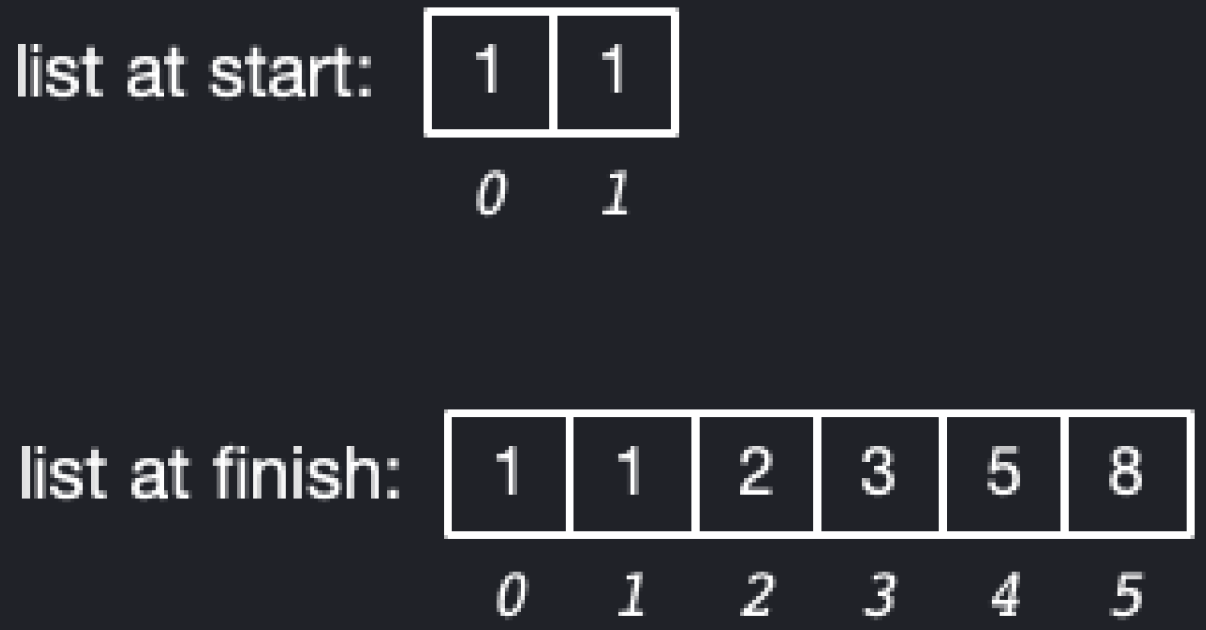
- Each trip through the loop has no dependence on other trips
- The operation can be done on each cell in isolation

Let's look at the first example - the difficult one

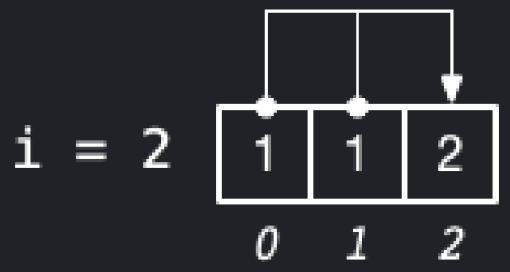
```
let list = [1, 1];

for(let i = 2; i < 6; i++) {
    list[i] = list[i - 1] + list[i - 2]
}
```

- The operation on `list[i]` depends on prior data
- The operations must be done in order or it breaks

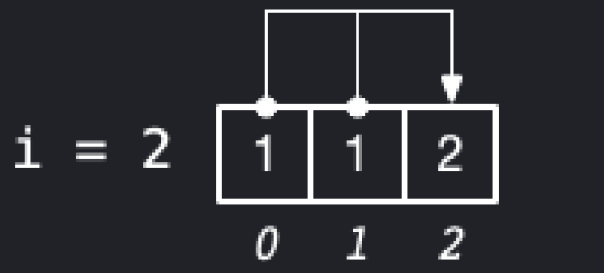
First Example: Why it's difficult

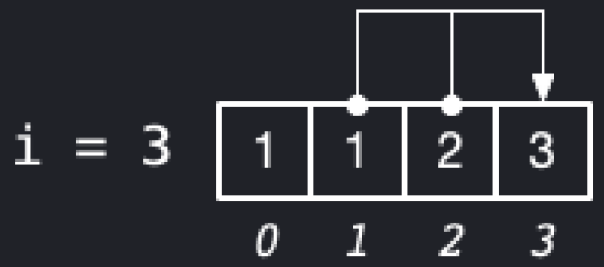


First Example: Why it's difficult

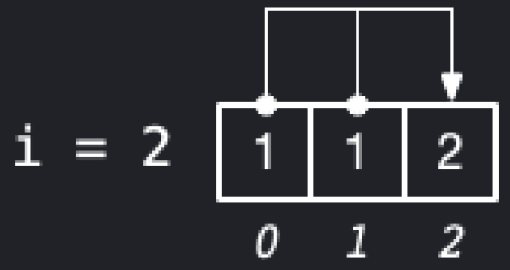


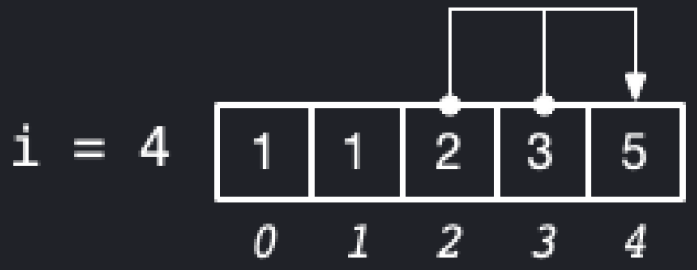
First Example: Why it's difficult

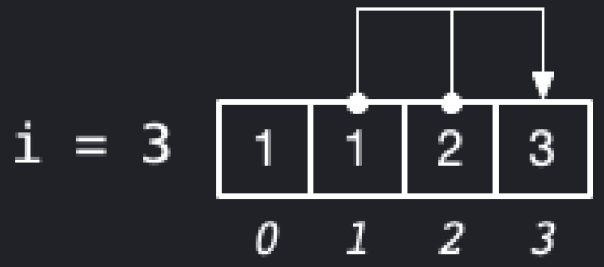




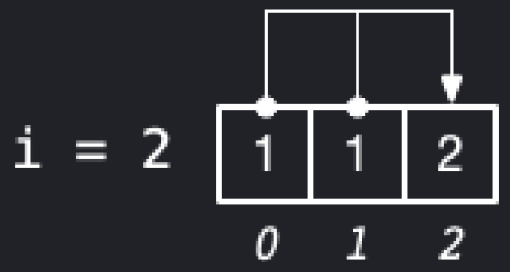
First Example: Why it's difficult

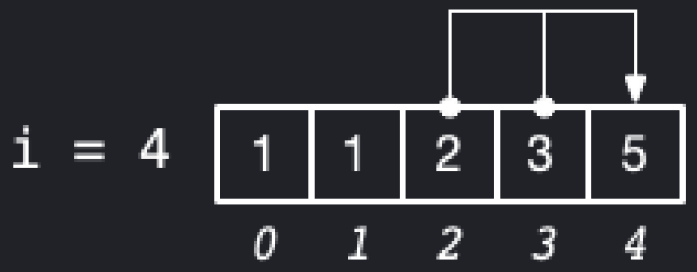


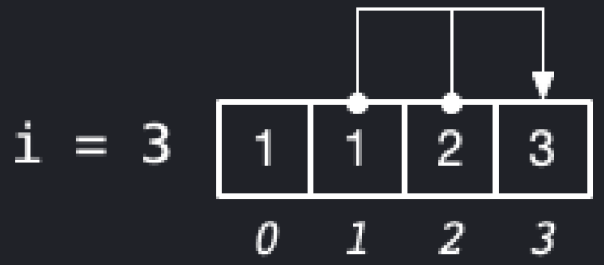


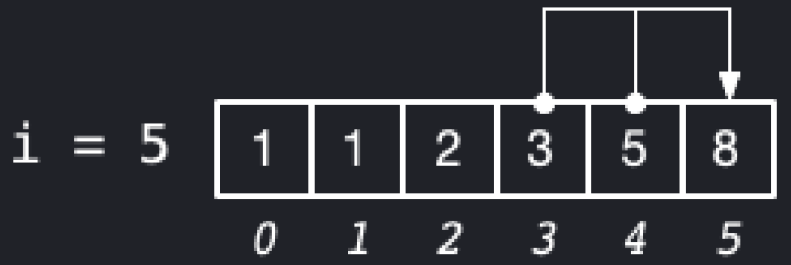


First Example: Why it's difficult





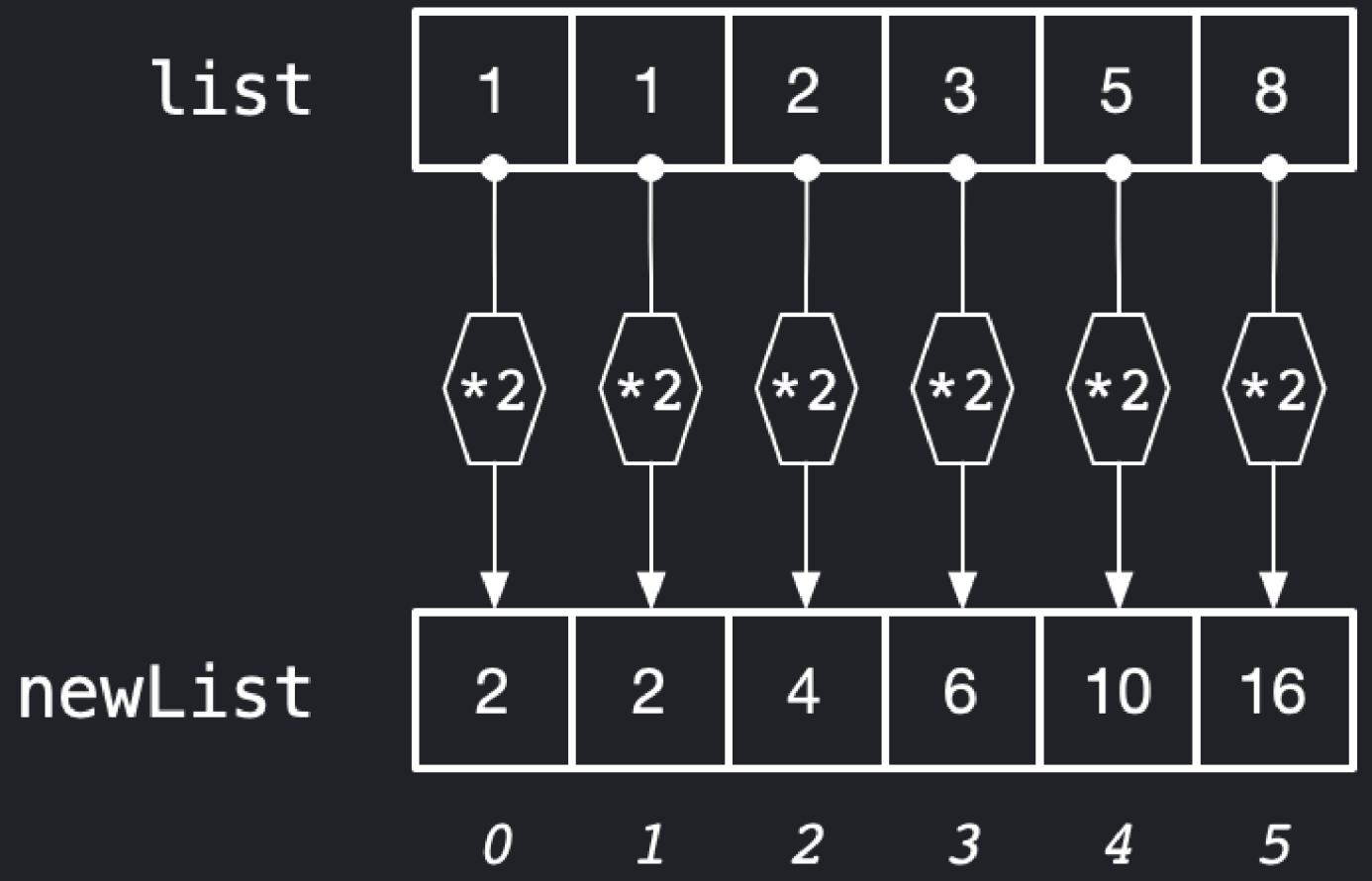




Back to the second example - the easy one

```
let list = [ 1, 1, 2, 3, 5, 8 ];
let newList = [];

for(let i = 0; i < list.length; ++i) {
    newList[i] = list[i] * 2;
}
```

Back to the second example - the easy one

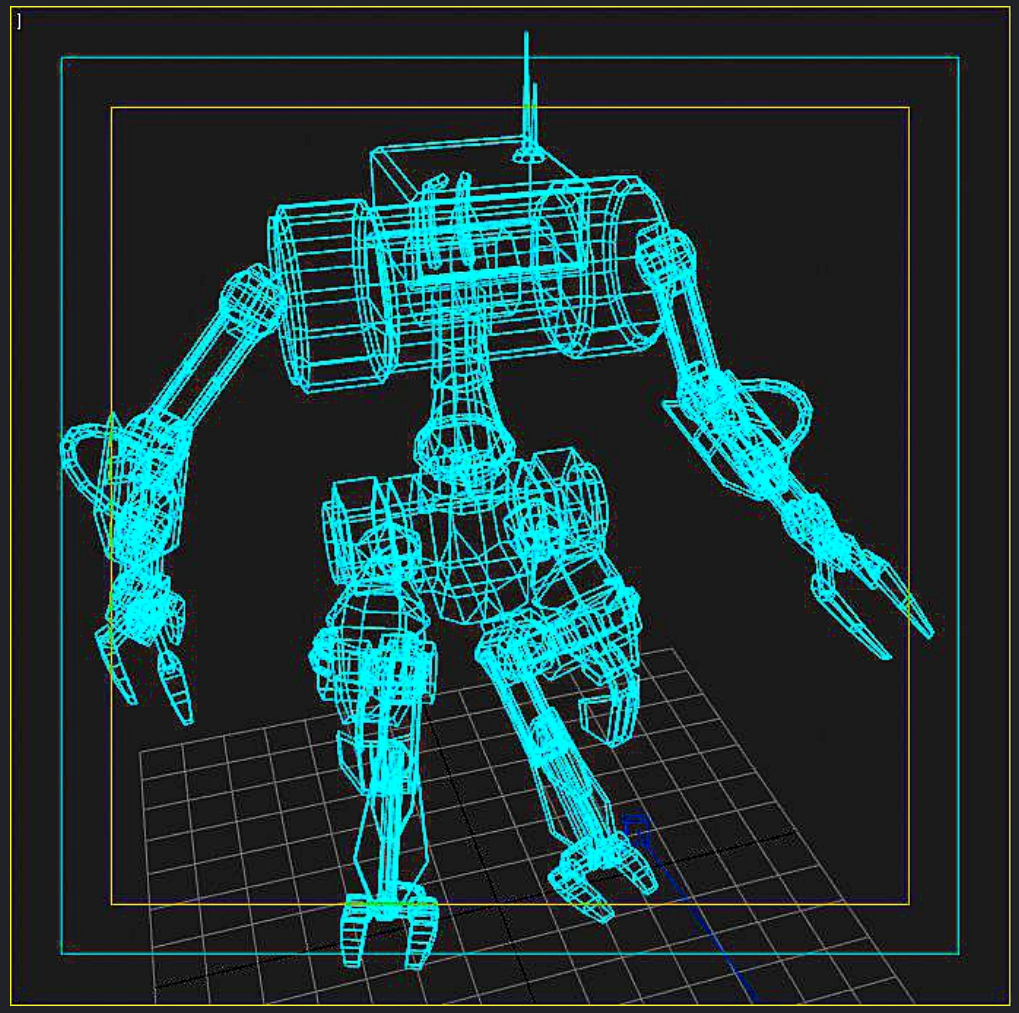


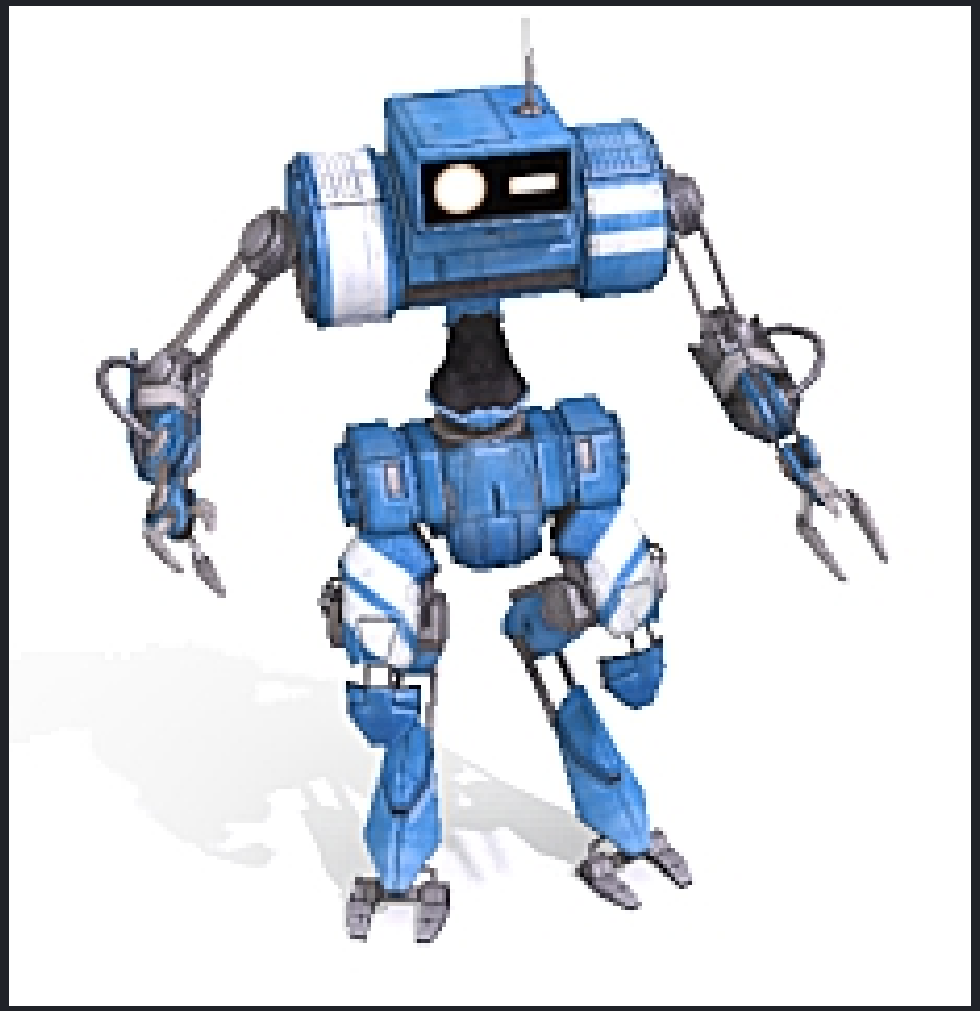
This is called Data-Parallel

the same operation done over different parts of the data

- it's what we use in **Machine Learning**
- and in **Data Science**, and graphics, and most applications

3D Graphics





Video Cards (GPUs)

Massively Parallel and hyper-optimized for 3D Graphics

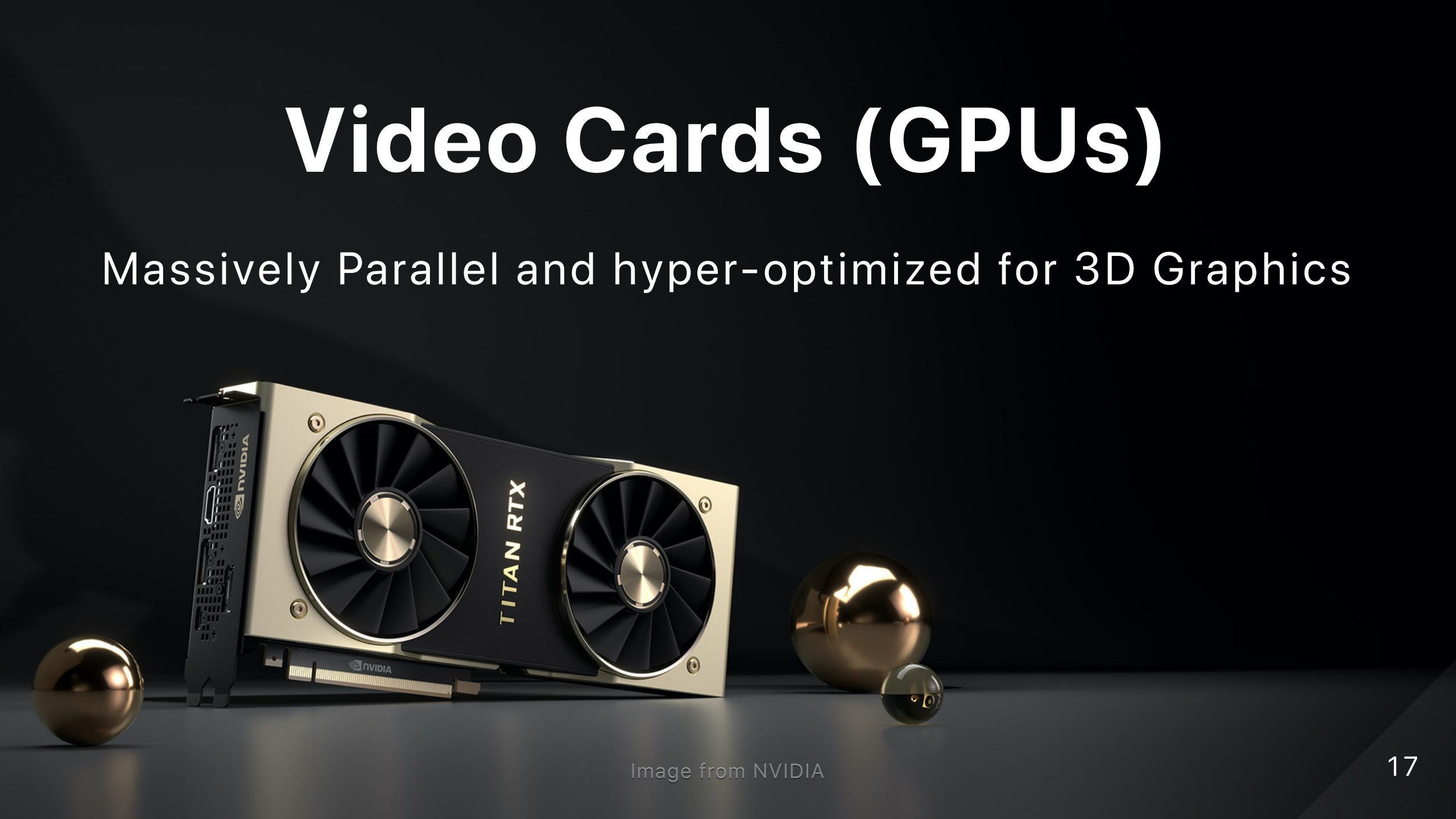


Image from NVIDIA

- This is NVIDIA's TU102 chip
- Just looking at it, you can see it has many repeated structures, supporting it's parallel nature.

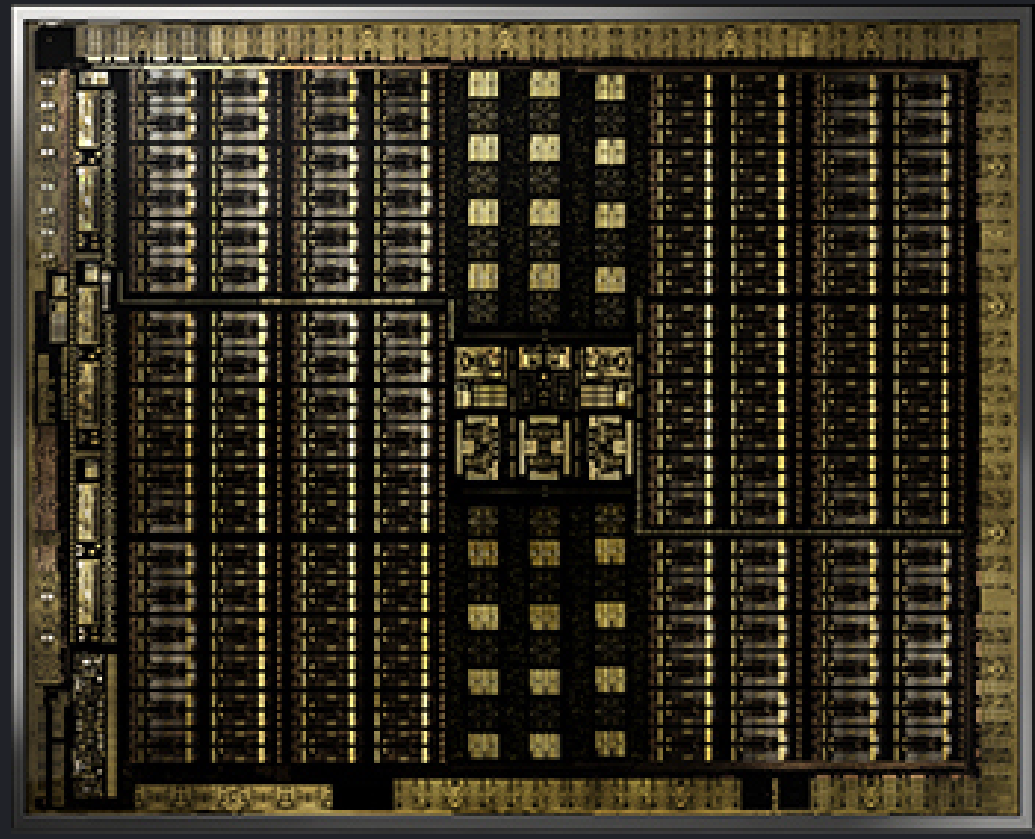


Image from NVIDIA

Here's the block diagram

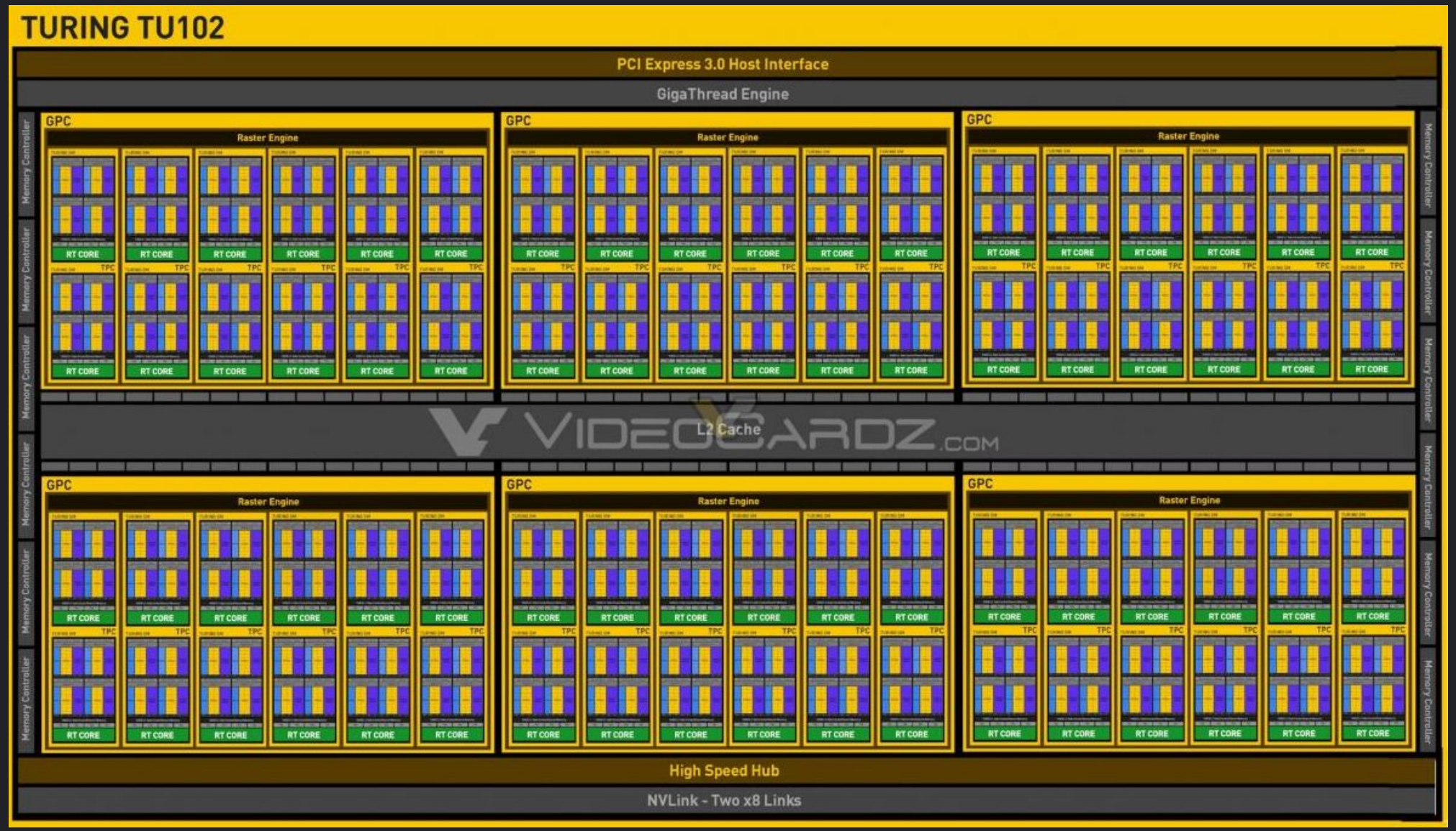


Diagram by videocardz.net

Quick history of GPGPU

- GPU: Graphical Processing Unit
- GPGPU: General Purpose GPU

1. In the beginning, there were CPUs, general-purpose
2. Video games started happening
3. GPUs got super good at 3D video games
4. Realized GPUs were good for other work
5. Figured out how to make "general-purpose" work fit into the shape GPUs need, so you can run your operations massively parallel, and *FAST*

You can do parallel work on a CPU

- CPUs are more versatile and more flexible
- Compute units in a CPU are faster and more capable
- CPUs have far fewer compute units than GPUs

Mark's Everyday Computer 2020

	Compute Units	Clock Speed
CPU	12	2,200-4,100 MHz
GPU	768	907 MHz

Super-High-End Workstation

	Compute Units	Clock Speed
CPU	128	2,700-4,200 MHz
GPU	8,192	1,410 MHz

You can do parallel work on multiple computers!

This is called Distributed Computing

- Many individual computers networked together
  - called a *cluster*
- Computers will have CPUs and possibly GPUs, too
- Shared Storage, management, and other resources
  - Most supercomuters are giant clusters

Summary

- Sequential vs Parallel execution
- We'll be doing Data-Parallel computing
- Same operations in parallel over different parts of the data
- Modern 3D graphics cards are data-parallel beasts
- GPGPU is the field of doing general-purpose computing using GPUs
- CPUs are different than GPUs and each has a time and a place
- Parallel execution can also be distributed over a cluster of machines

Parallel Code Activity

- Here is a naïve, sequential bit of code to find the largest value in a list of numbers.
- Can this be run in parallel? Why or why not?

```
// assume the variable 'list' contains the list
let max = Number.MIN_VALUE;
for(let i = 0; i < list.length; i++) {
    if (list[i] > max) {
        max = list[i];
    }
}
```

- How could this be rewritten to find the max in a parallel way?