PARALLEL PROGRAMMING

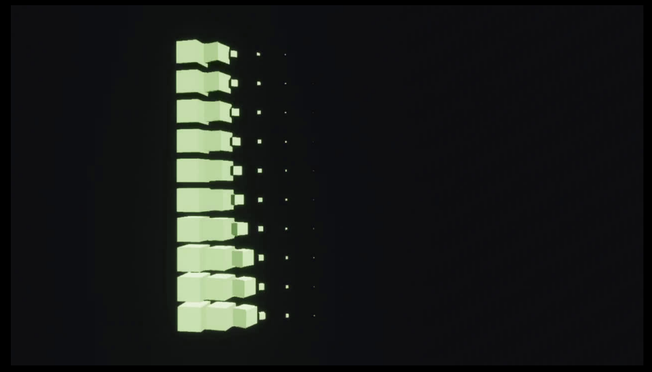
An Introduction by Nixon

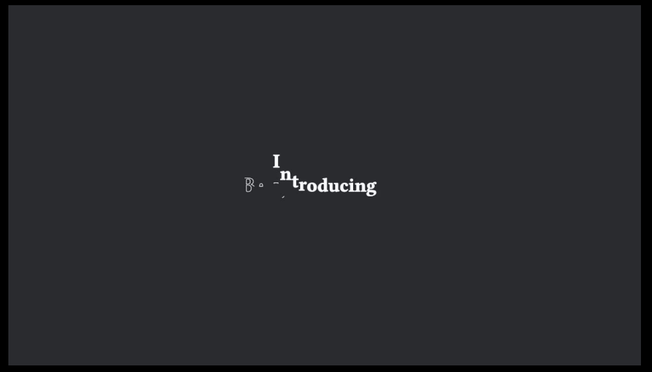
VOXELL

RUST MY

About Me

- Computer Graphics Engineer
- AI Enthusiast





VOXELL

Let’s start off with a question…

If you were plowing a field, which would you rather use?

2 strong oxen

or

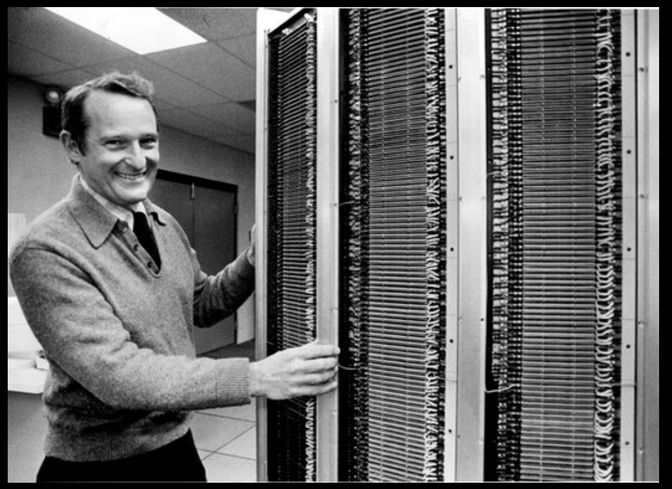
1024 chickens

If you were plowing a field, which would you rather use?

2 strong oxen

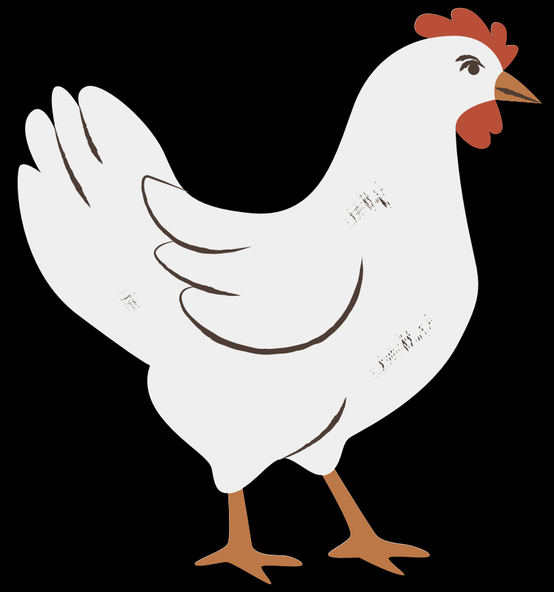
or

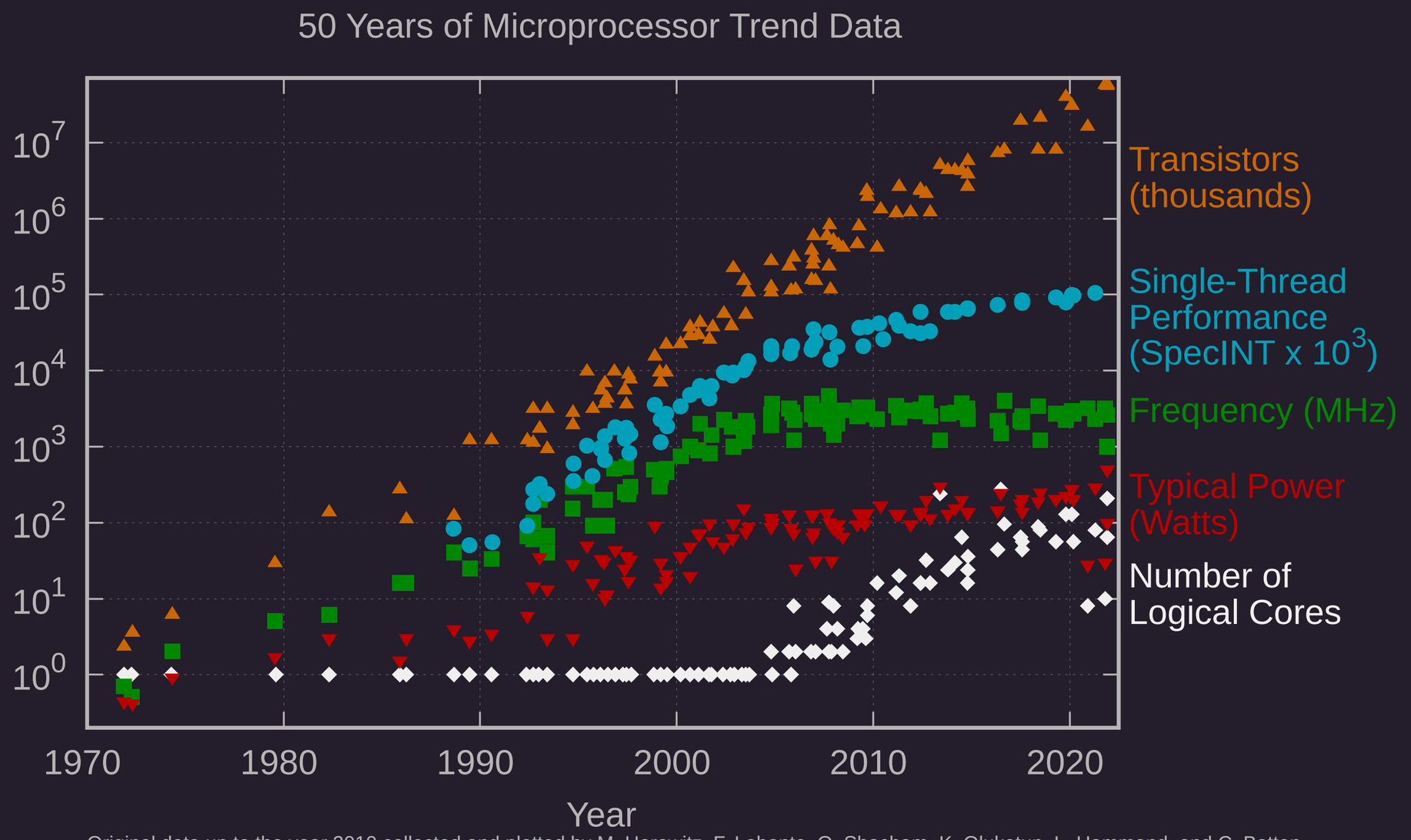
1024 chickens



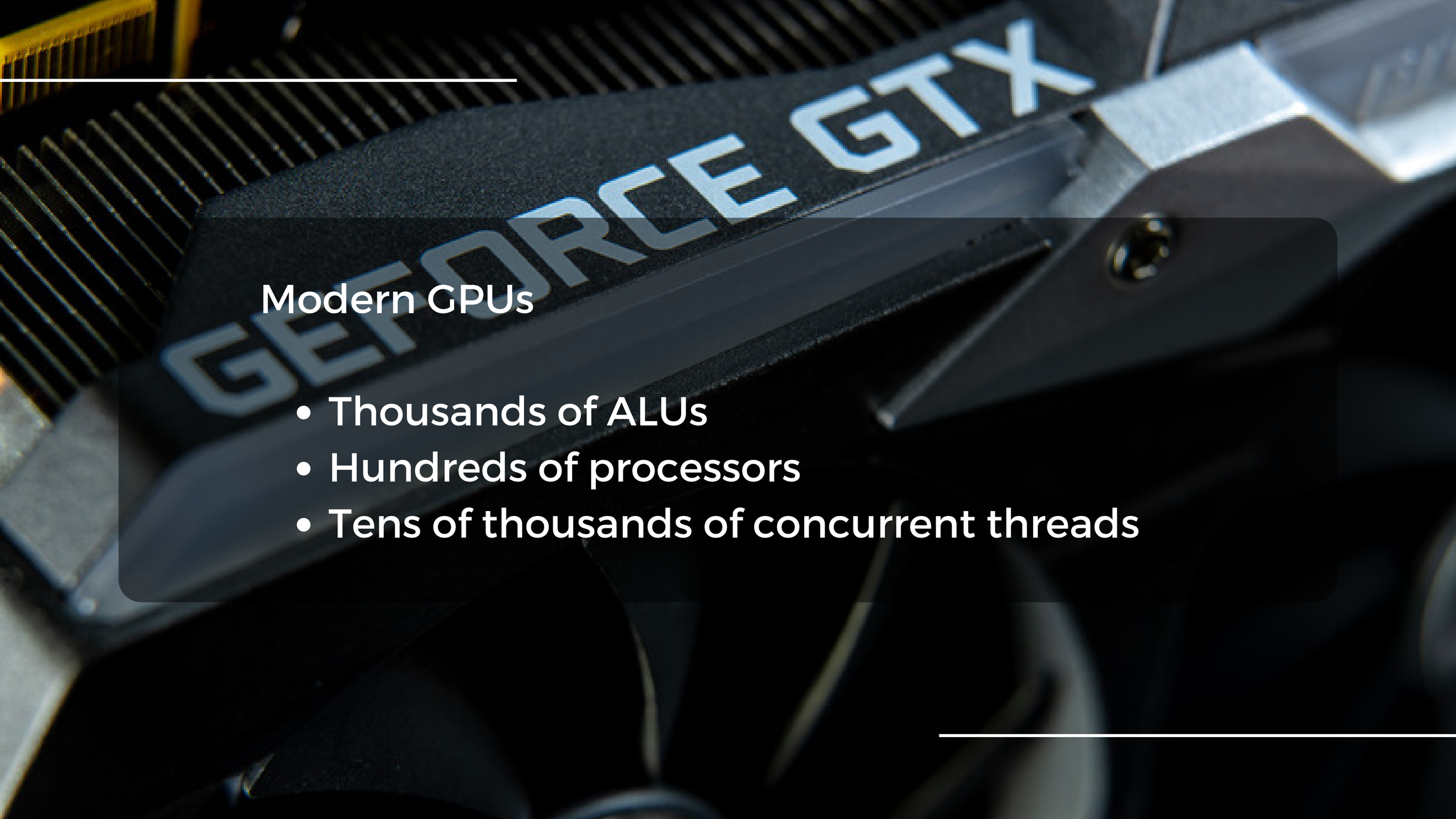
Seymour Cray

I





Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

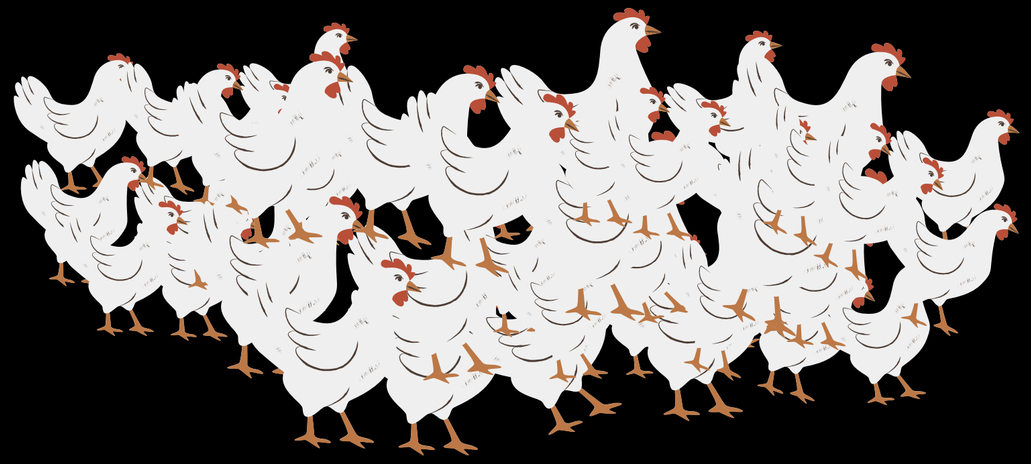


Modern GPUs

- Thousands of ALUs
- Hundreds of processors
- Tens of thousands of concurrent threads

We are doom

Why?



Managing chickens is hard!

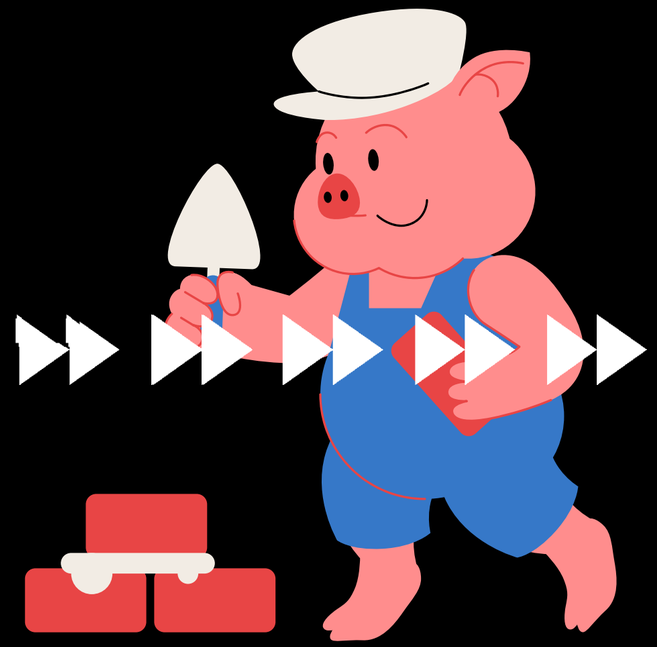
CHAPTER 1
LAYING BRICKS

Just start laying bricks!

- Lay bricks 1 by 1
- By yourself

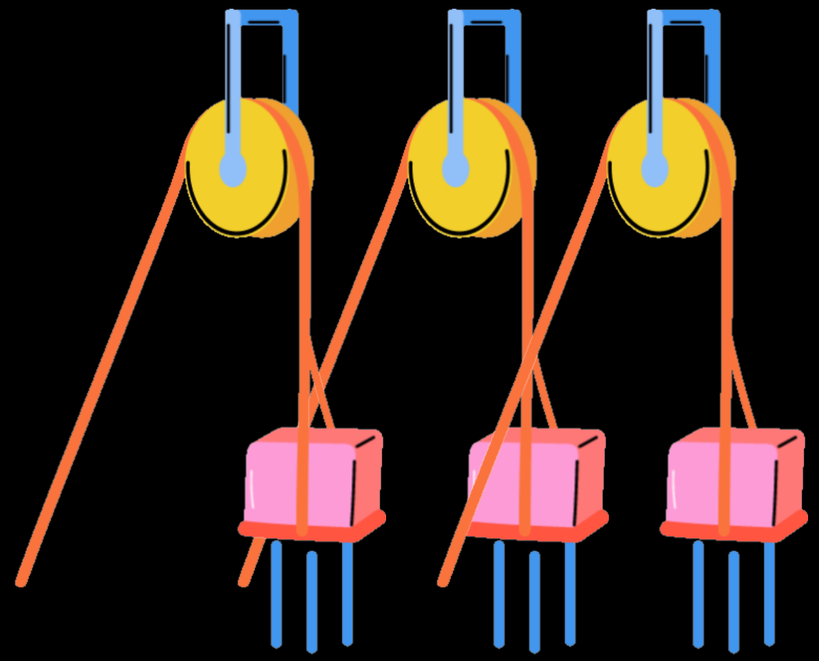
Lay faster!

- Instead of laying 5 brick per minute, lay 10 bricks per minute!
- There is a limit as to how fast a person lays brick.



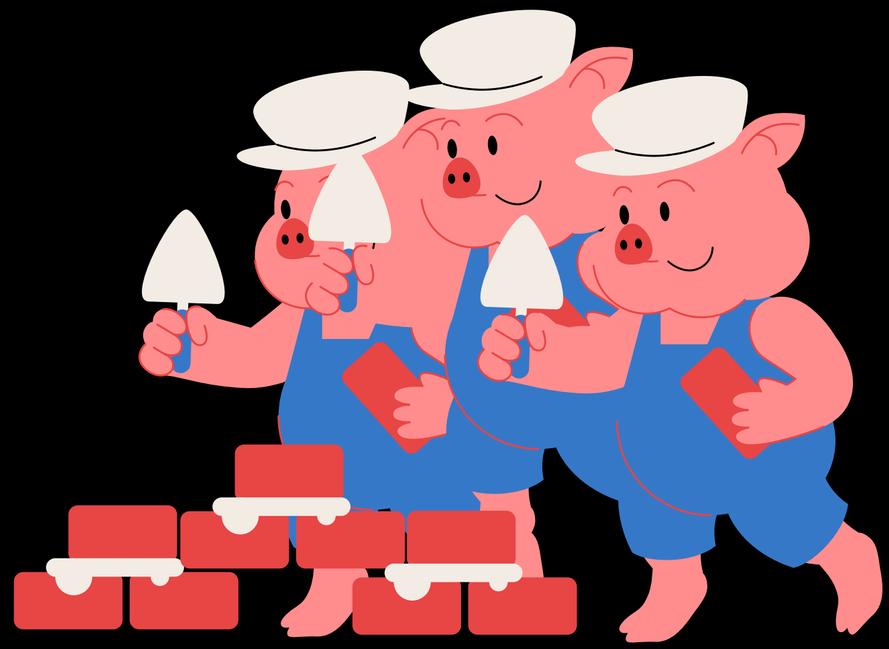
Use a machine 😮

- Buy a brick laying machine that can lay multiple bricks at once.



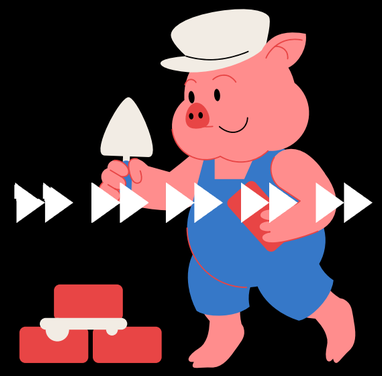
Hire more people

- Hire more people to lay bricks together!



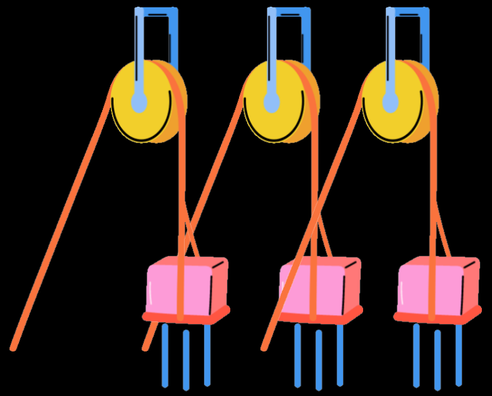
Analogy

Faster clock cycles



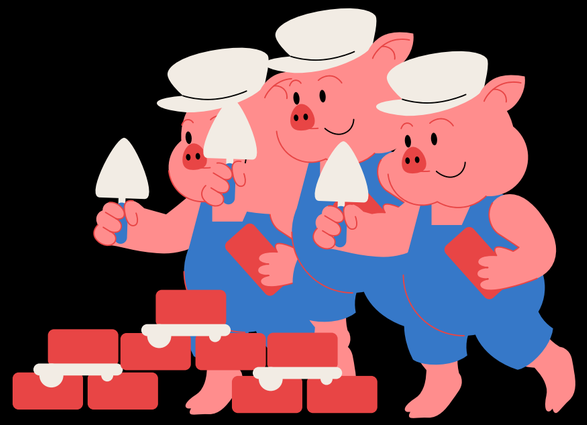
Lay faster

Instruction level parallelism



Use a machine

More processors



Hire more people

CHAPTER 2

SOFTWARE PARALLELISM

Parallelism is a **subset** of **concurrency**

Parallelism is a **subset**
of **concurrency**

Concurrency is dealing with a lot of things at once. Parallelism is doing a lot of things at once.

- Rob Pike

What is software parallelism?

- Divide the work into units of tasks.
- Run independently.
- No dependencies from other tasks.
- Can be executed simultaneously.

How it ends up being executed on the hardware is **completely irrelevant** to us.

CHAPTER 3

DATA-PARALLEL PRIMITIVES

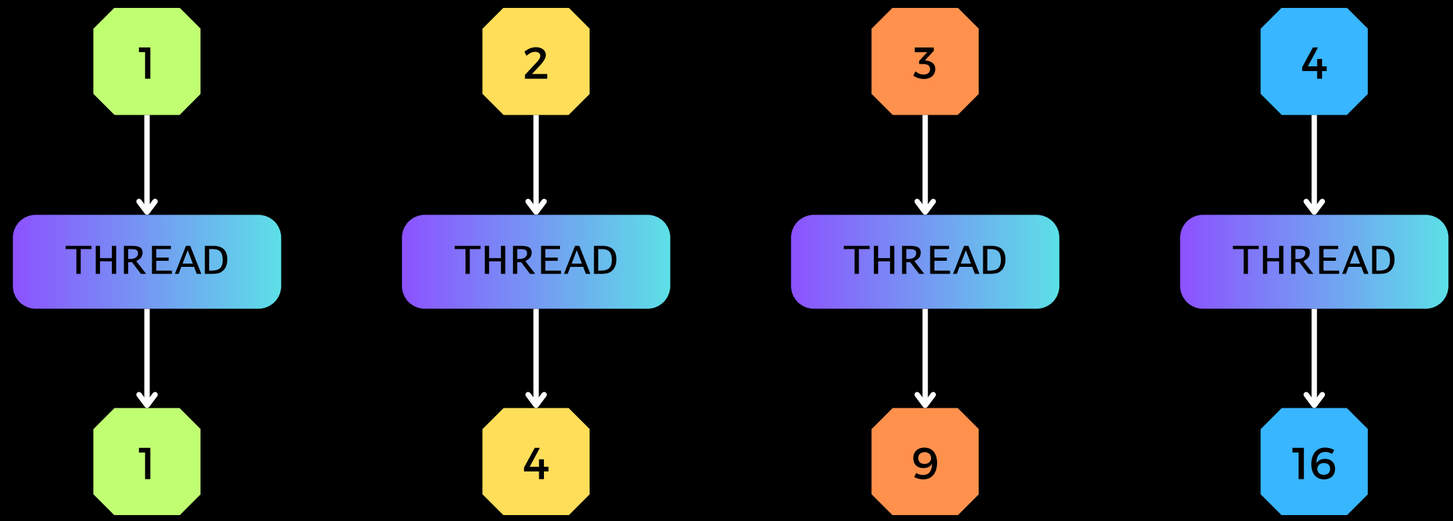
What are data-parallel primitives? (DPP)

- Building blocks.
- Commonly used patterns.
- Combined to create complex parallel algorithms.
- Reuse existing high performance implementations.

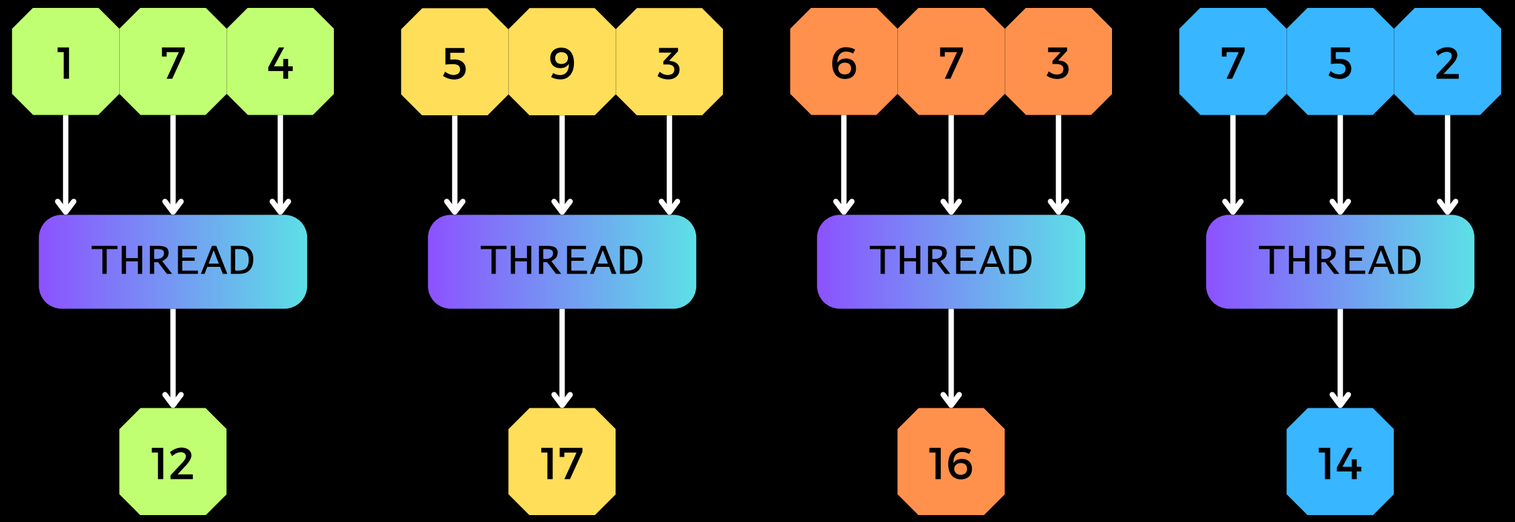
Common Types of DPP

- Map
- Stencil
- Reduce
- Scan
- Gather/scatter
- Compaction
- Partition

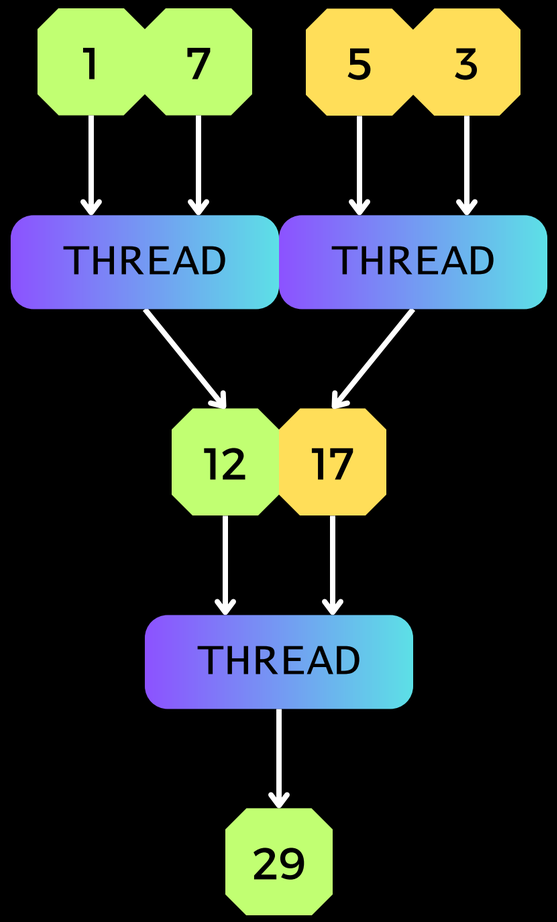
Map



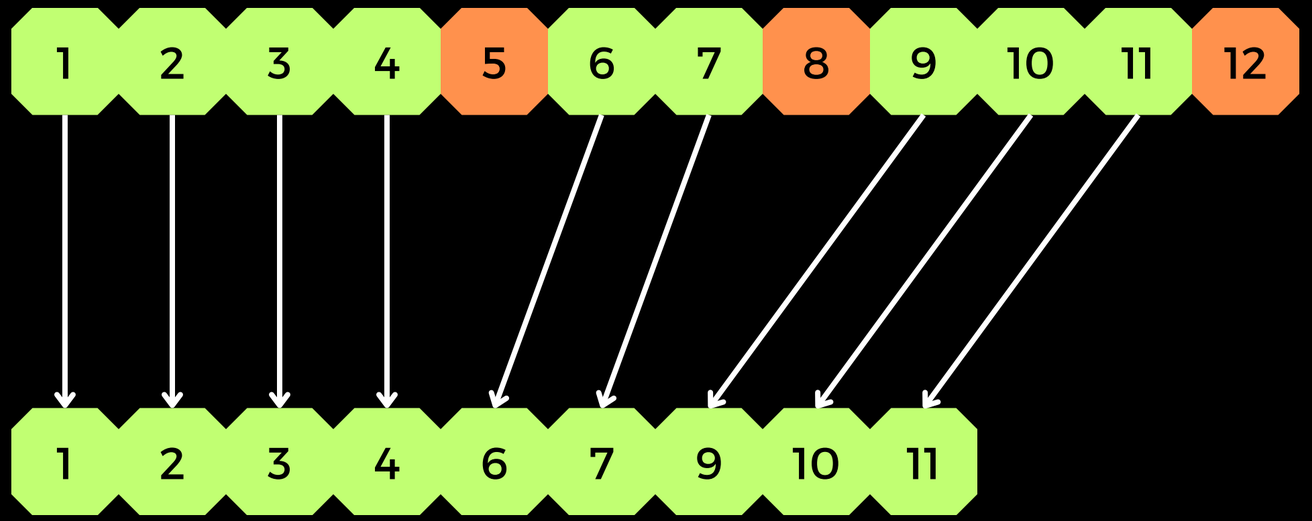
Stencil



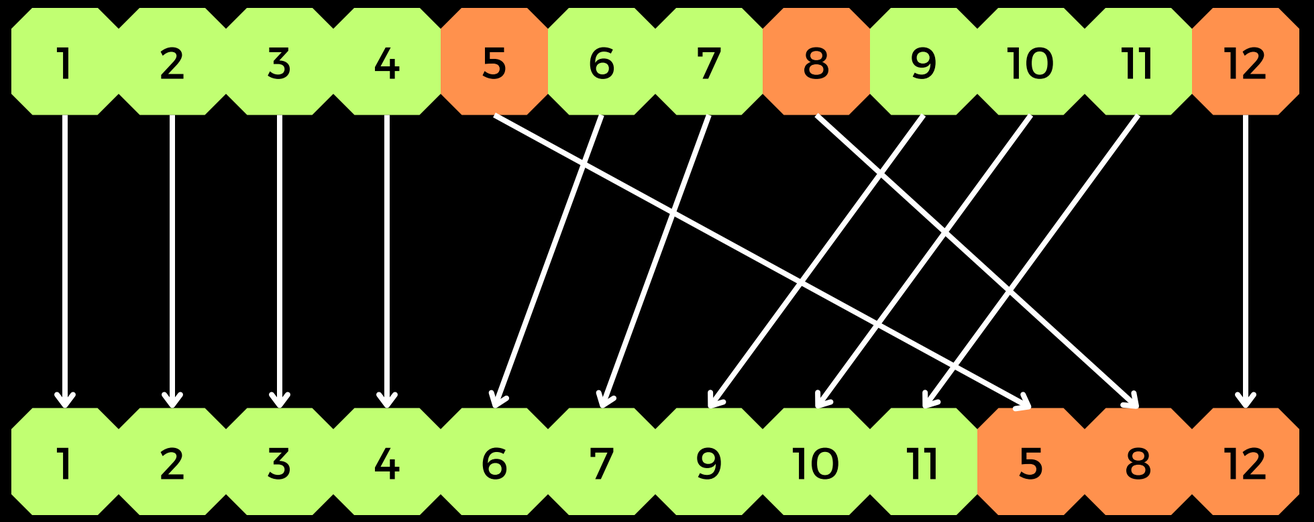
Reduce



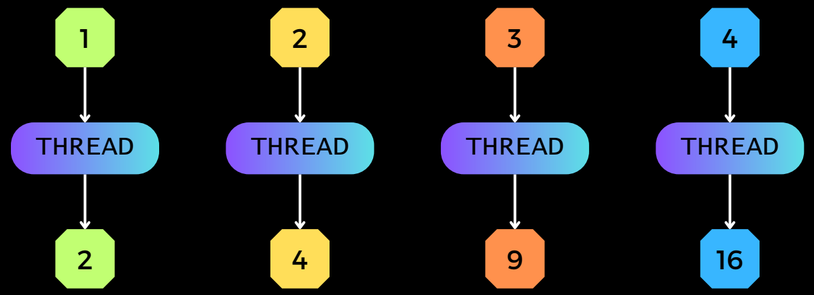
Compaction



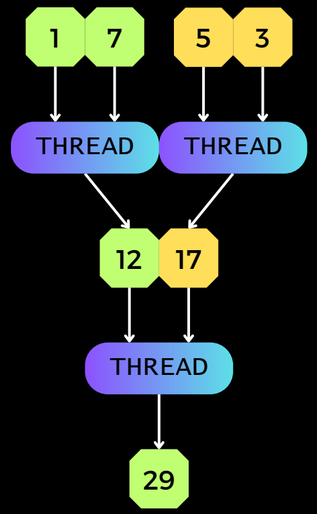
Partition



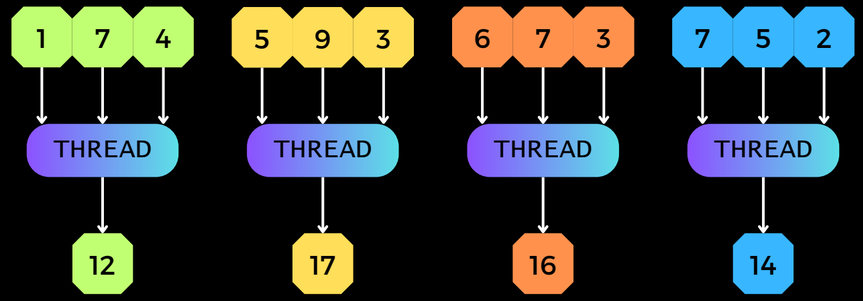
Map



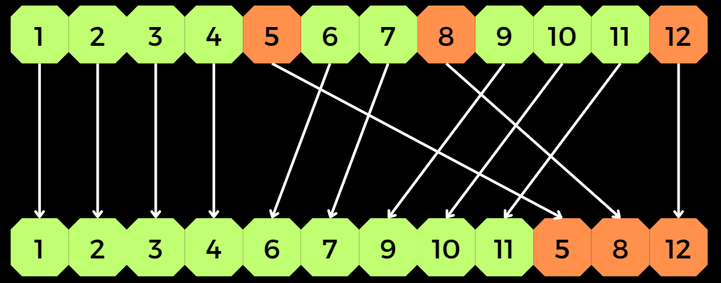
Reduce



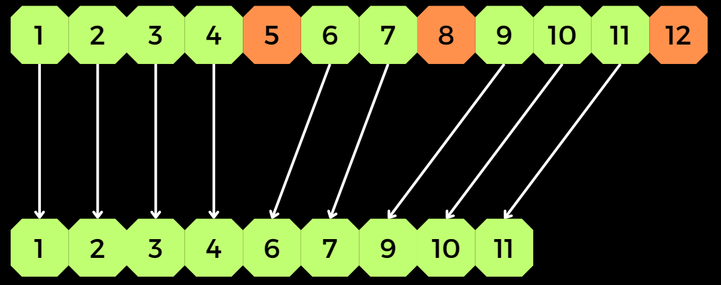
Stencil



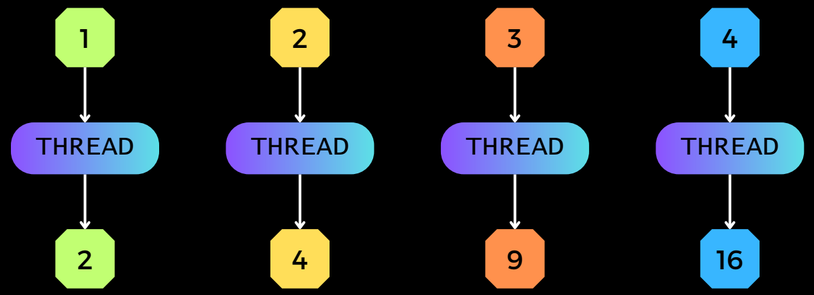
Partition



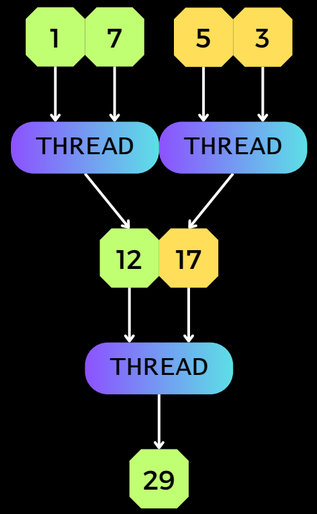
Compaction



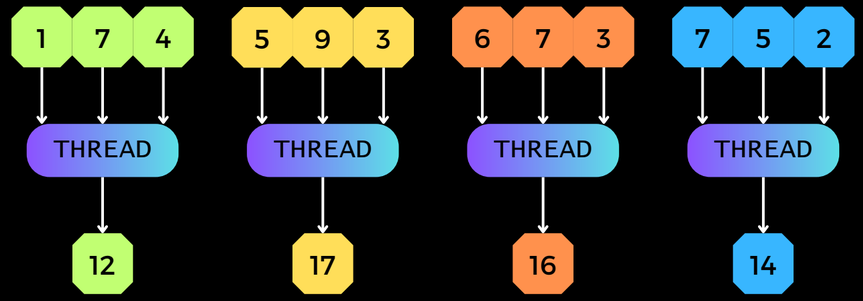
Map



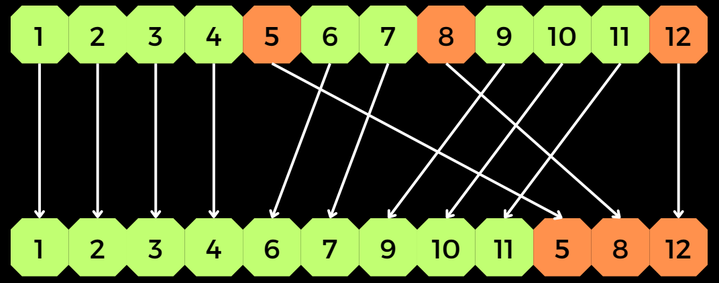
Reduce



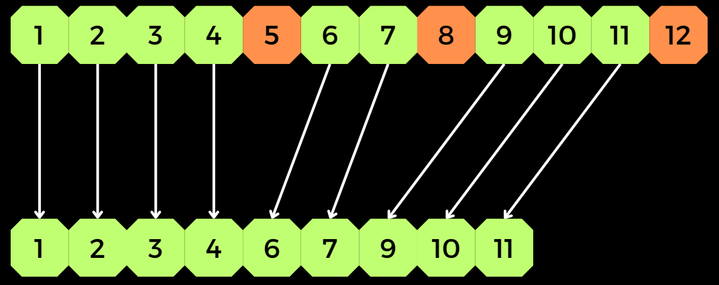
Stencil



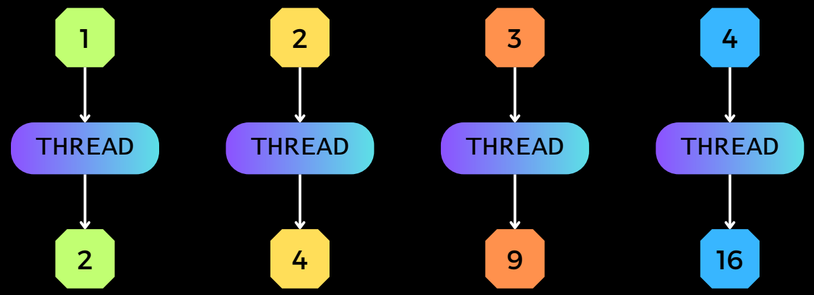
Partition



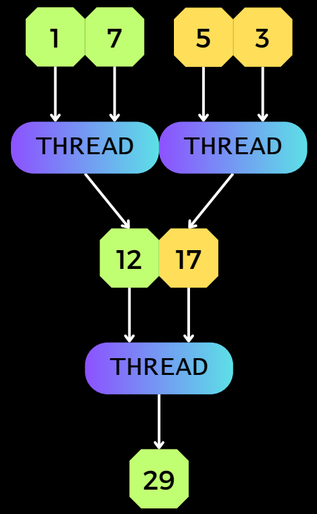
Compaction



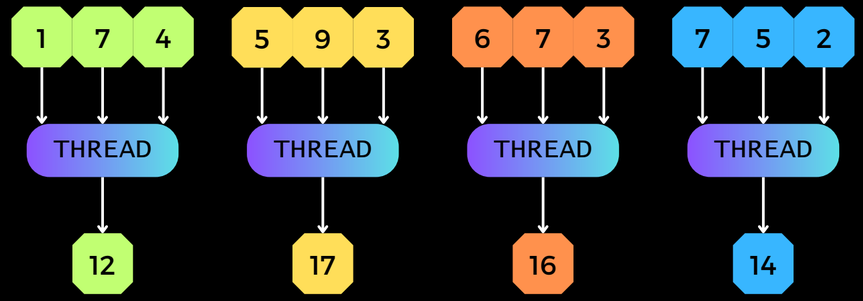
Map



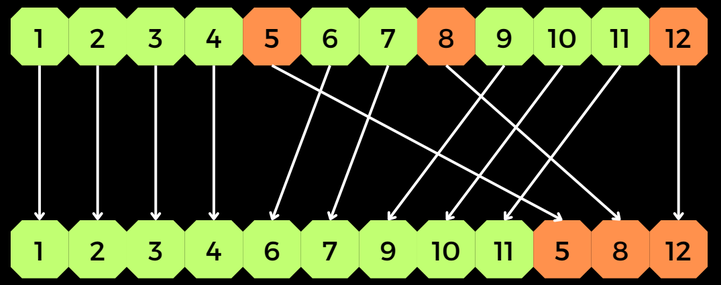
Reduce



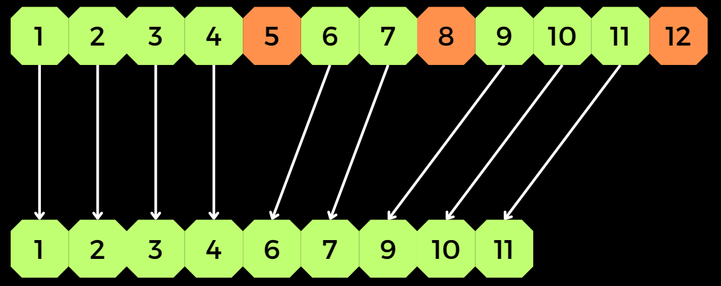
Stencil



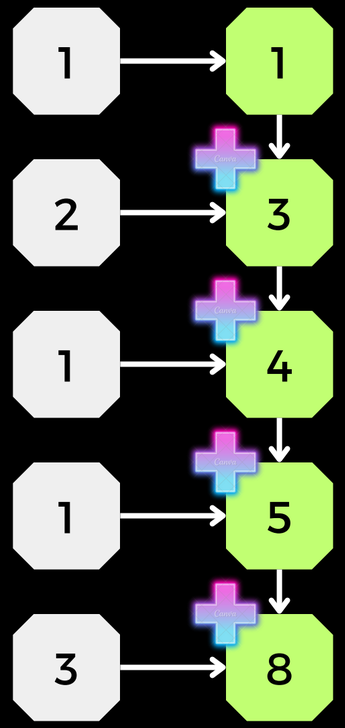
Partition



Compaction



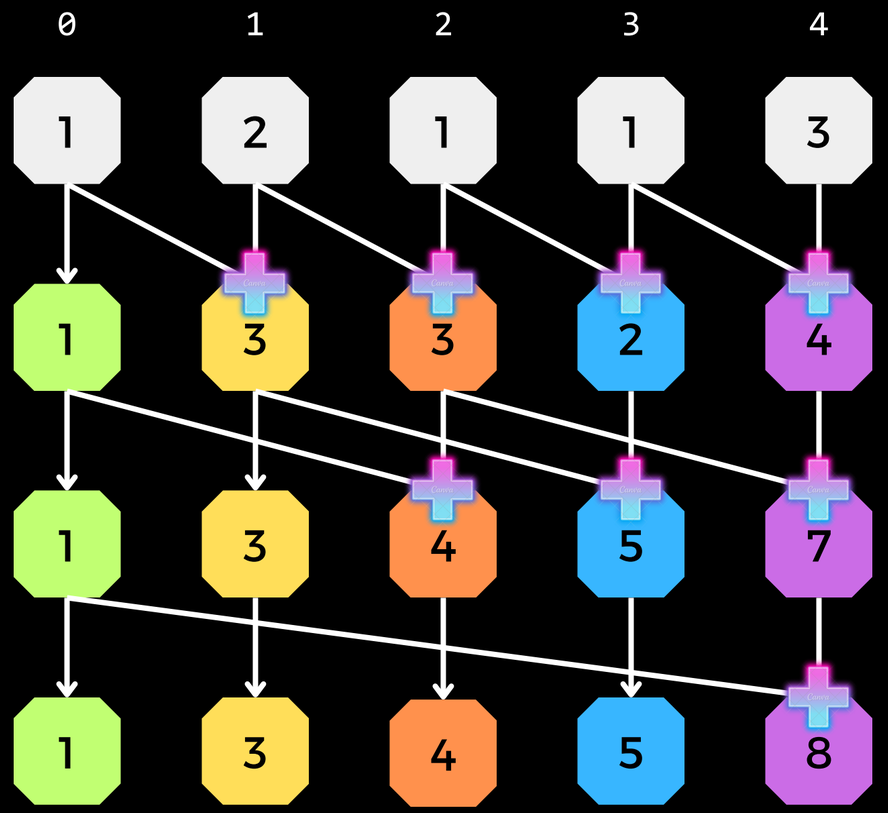
Sum Scan (in serial)

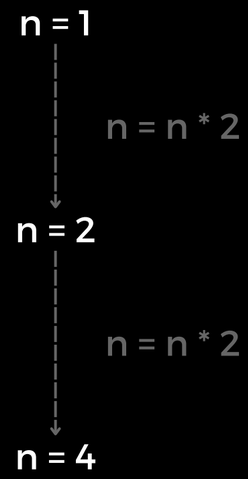


O(n)

Sum Scan

```
if i >= n {
    x[i] = x[i] + x[i - n]
}
```

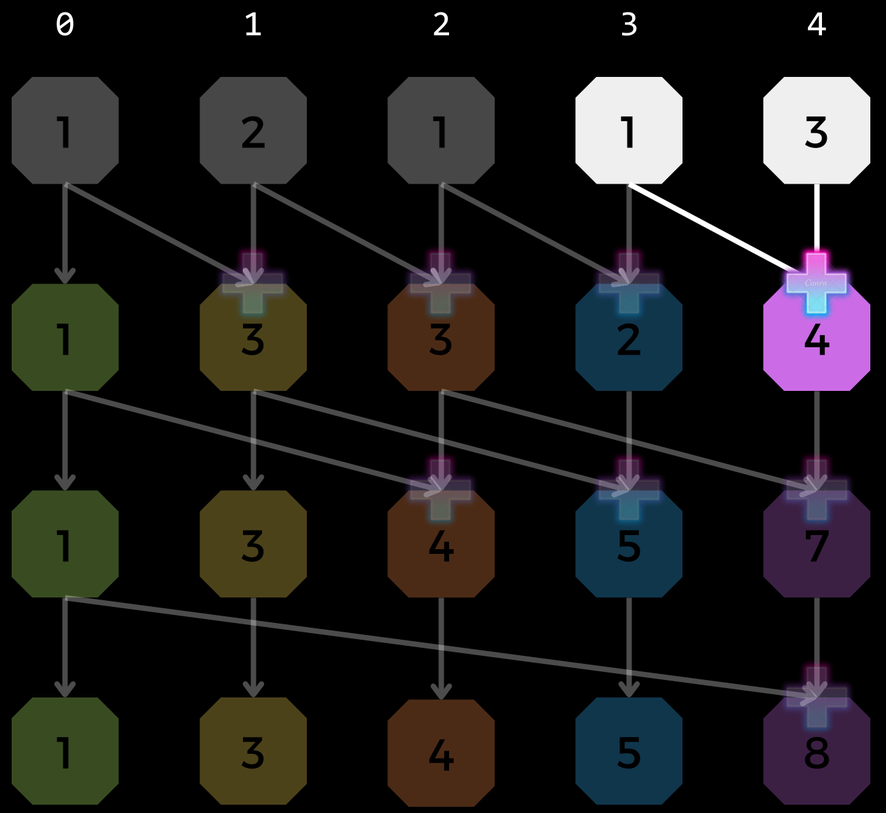


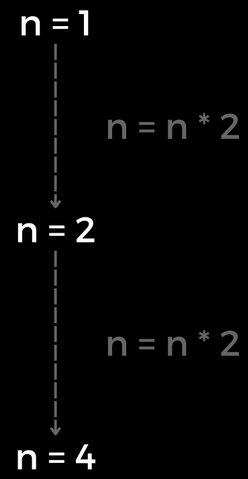


$O(Log_2 n)$

Sum Scan

```
if i >= n {
    x[i] = x[i] + x[i - n]
}
```

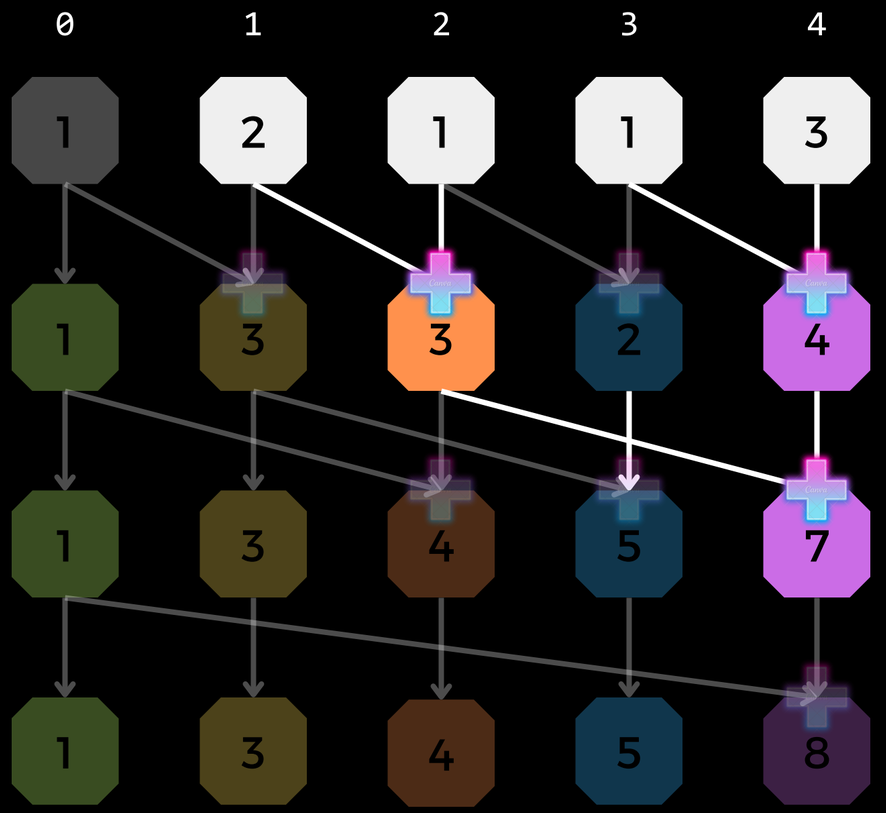


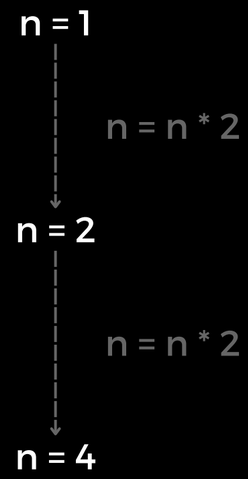


$O(Log_2 n)$

Sum Scan

```
if i >= n {
    x[i] = x[i] + x[i - n]
}
```

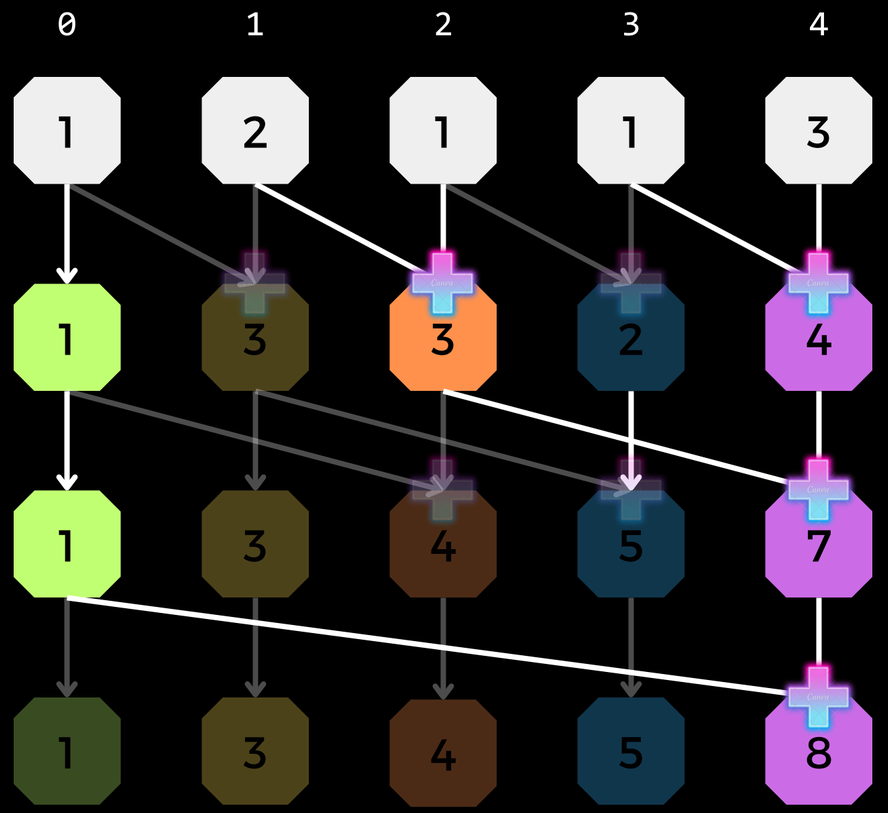


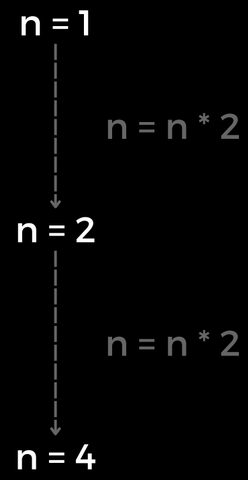


$O(Log_2 n)$

Sum Scan

```
if i >= n {
    x[i] = x[i] + x[i - n]
}
```



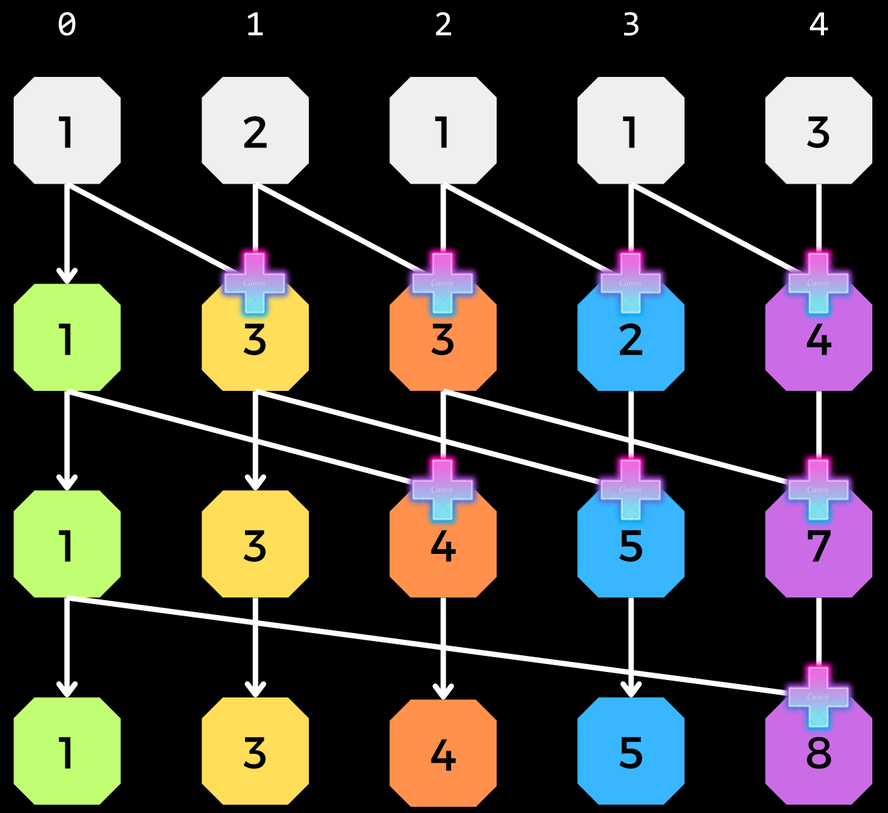


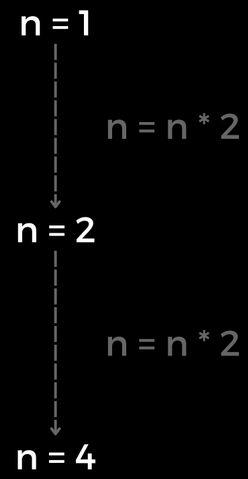
$O(Log_2 n)$

Hillis Steele

Sum Scan

```
if i >= n {
    x[i] = x[i] + x[i - n]
}
```



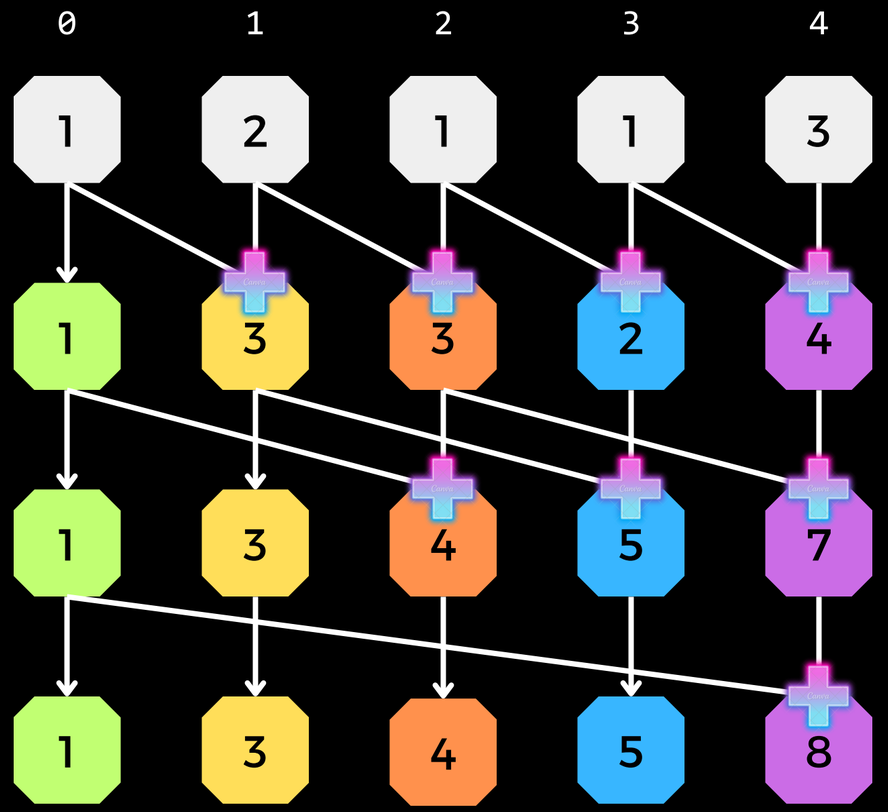


$O(Log_2 n)$

Hillis Steele

Sum Scan

```
if i >= n {
    x[i] = x[i] + x[i - n]
}
```



DENARY

n = 1

n = n * 2

n = 2

n = n * 2

n = 4

BINARY

n = 001

n = n << 1

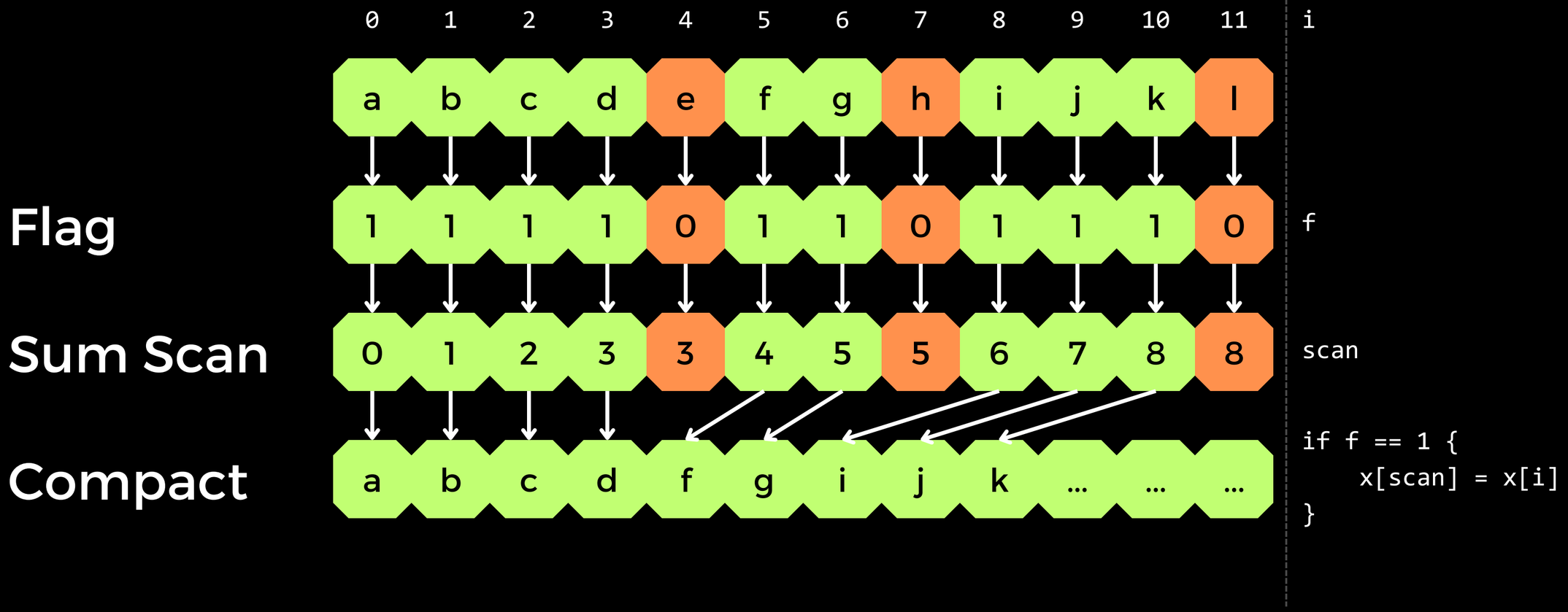
n = 010

n = n << 1

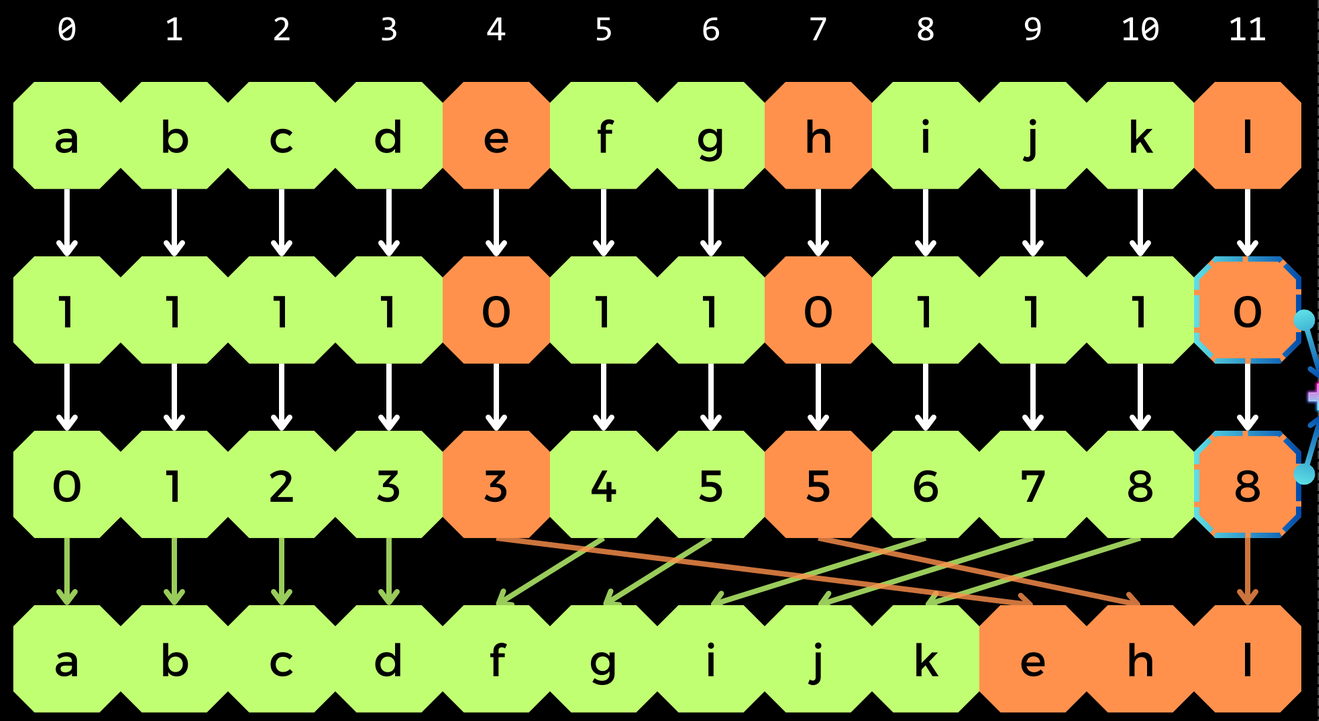
n = 100

$O(Log_2 n)$

Compaction


0 1 2 3 4 5 6 7 8 9 10 11
i
a b c d e f g h i j k l
Flag
1 1 1 1 0 1 1 0 1 1 1 0
f
Sum Scan
0 1 2 3 3 4 5 5 6 7 8 8
scan
Compact
a b c d f g i j k … … …
if f == 1 {
x[scan] = x[i]
}

Partition



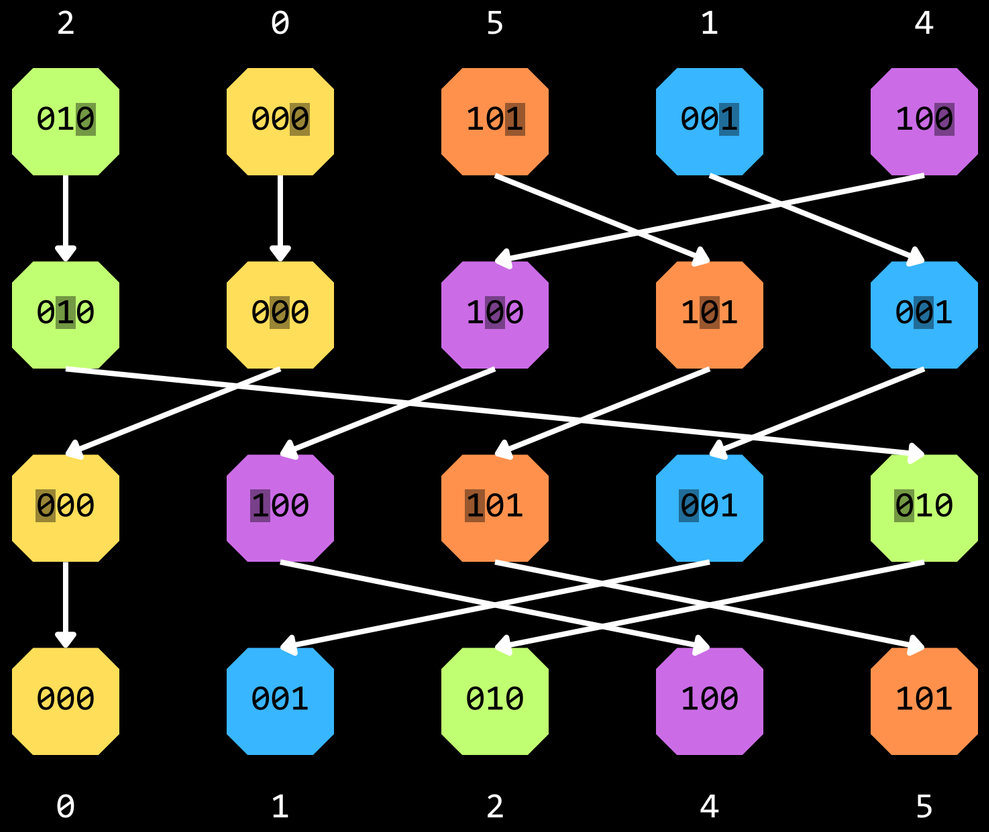
Flag

Sum Scan

Compact

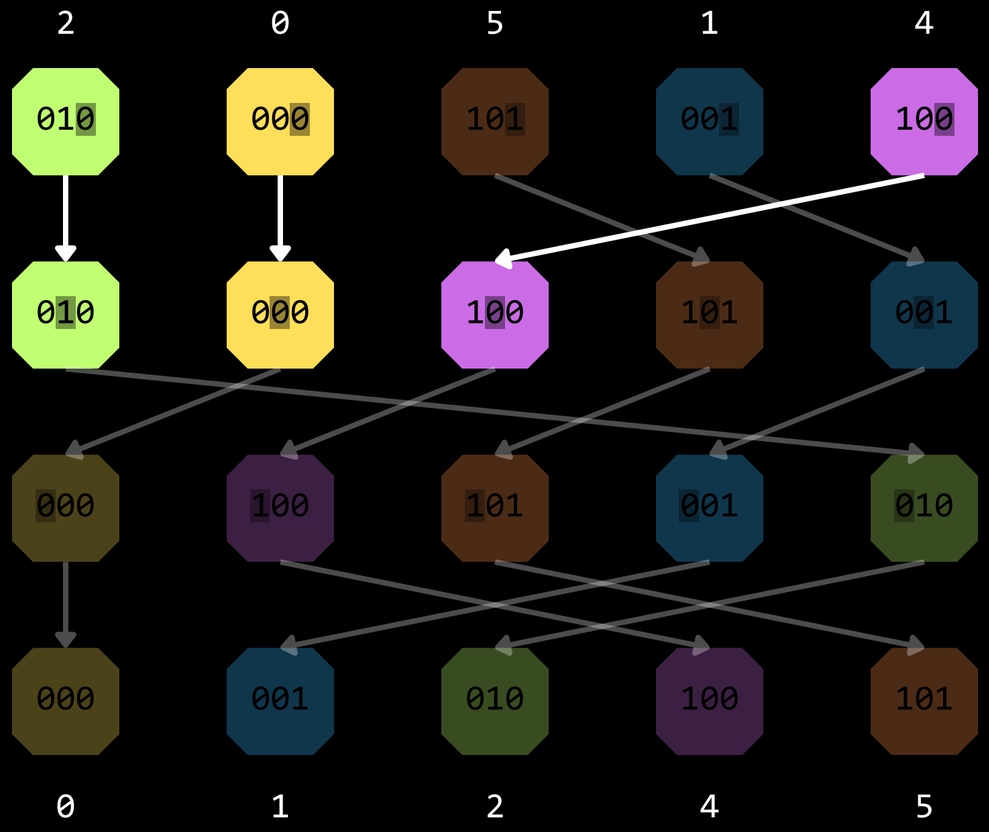
```
if f == 1 {
    x[scan] = x[i]
} else {
    x[ones + i - scan] = x[i]
}
```

Radix Sort?? 



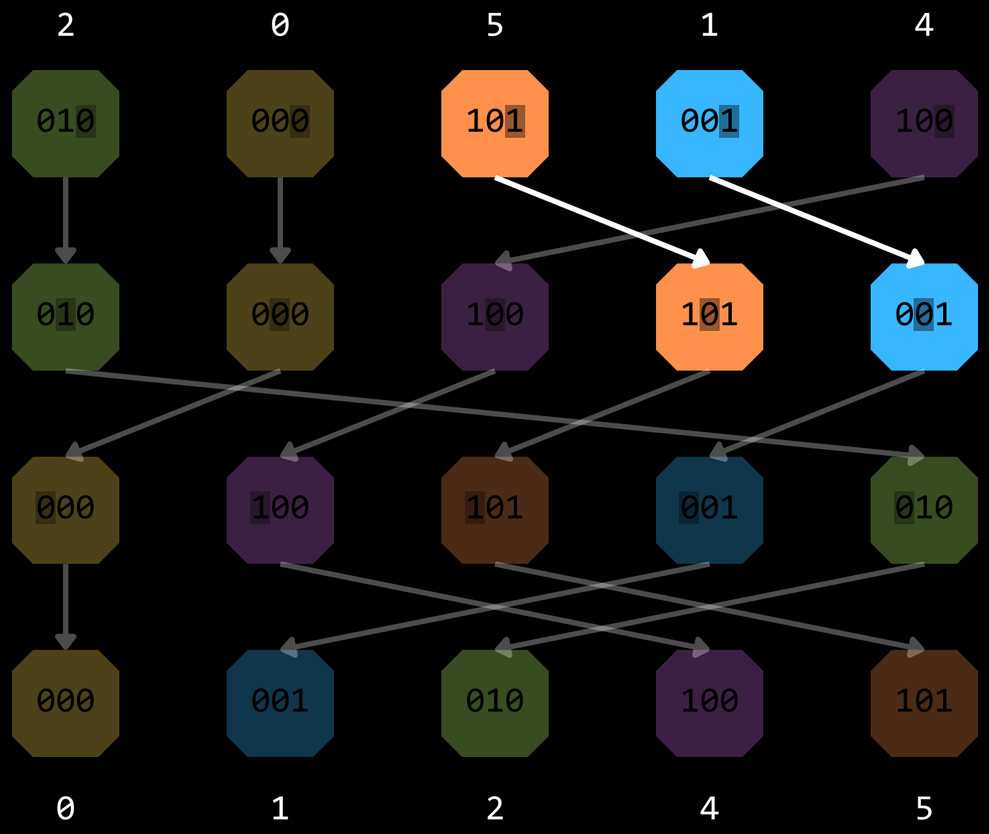
Solve this using DPPs!

Radix Sort?? 



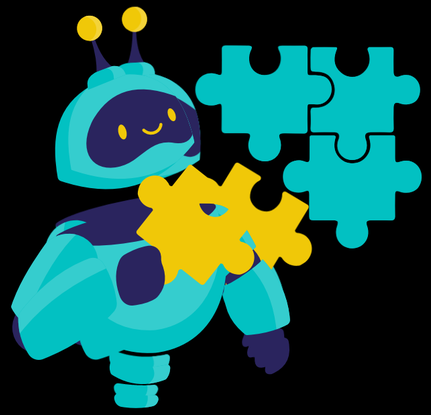
Solve this using DPPs!

Radix Sort?? 🙀



Solve this using DPPs!

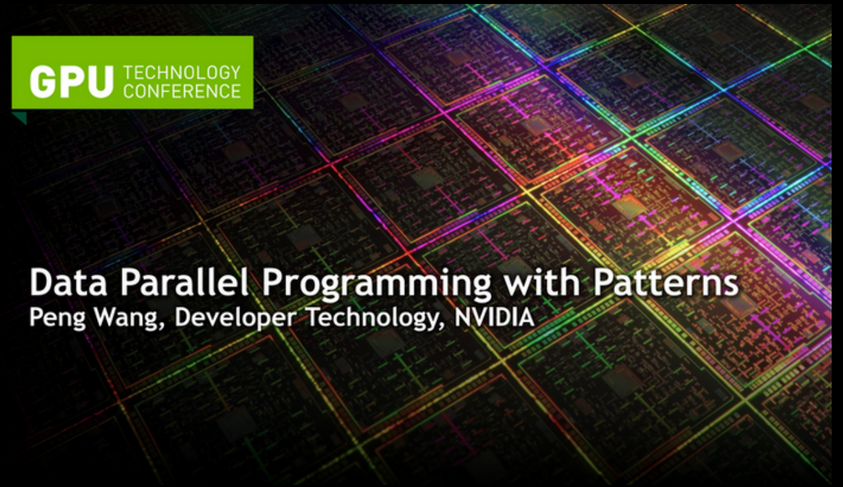
CHAPTER 4
CONCLUSION

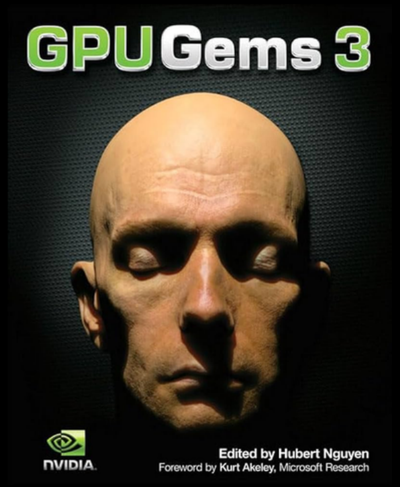


Look out! 👀

- Thread count (hardware is still important)
- Write/mutate data (utilize atomics if necessary)
- Graph coloring
- Memory is slow
- Beware of branching
- Data structures
- Use DPP to express your algorithm

Additional Resources





Q&A?

VOXELL

RUST MY

THANK YOU

HAVE FUN PARALLELIZING!

VOXELL

RUST MY