IT UNIVERSITY OF CPH

GPUs, FPGAs, Accelerators

Ehsan Yousefzadeh-Asl-Miandoab

(ehyo@itu.dk)

Some slides are for later reading.

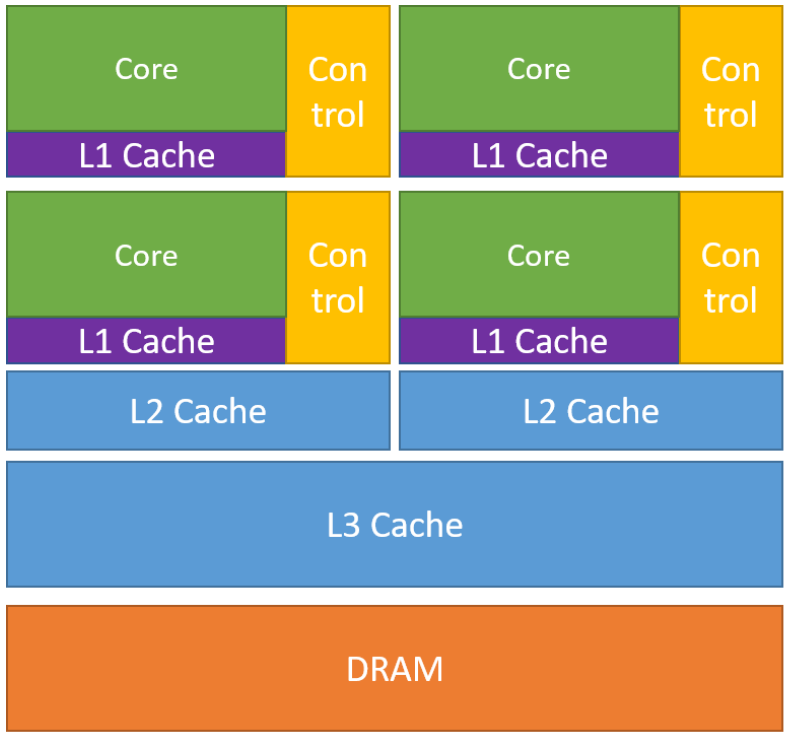
Search and read more about unfamiliar keywords.

Content

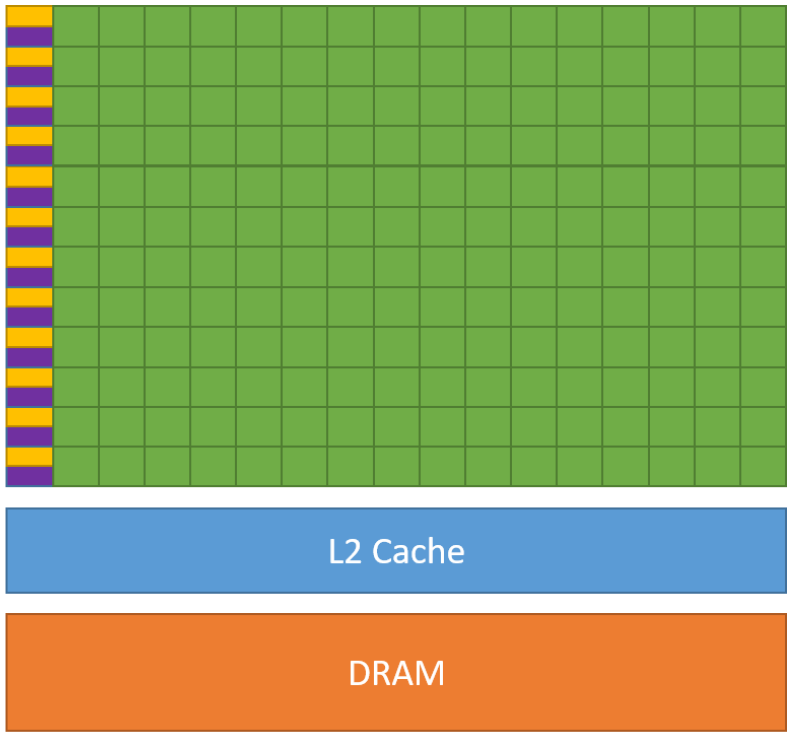
- **GPUs**
  - Story of NVIDIA GPUs
- FPGA
- Accelerator
- Tradeoff of processors

GPUs

- Parallel Processor
- Throughput-oriented
- Low Working Frequency



CPU



GPU

GPUs

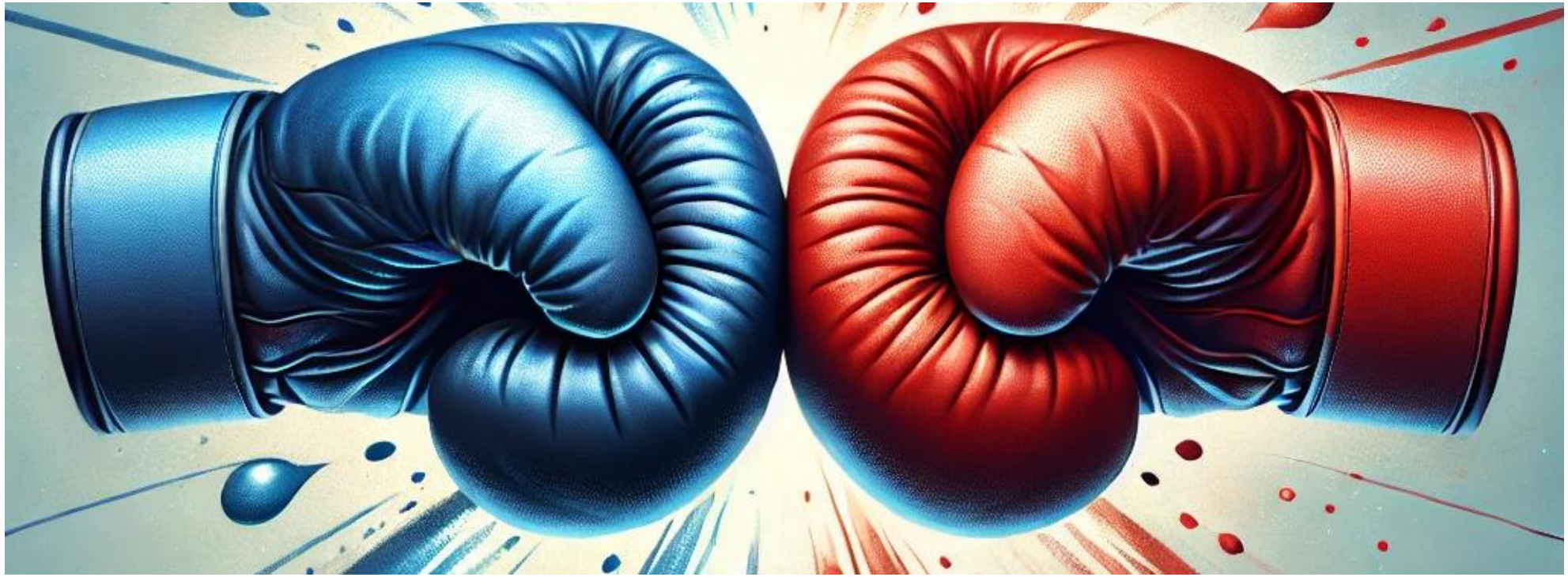
- Parallel Processor
- Throughput-oriented
- Low Working Frequency

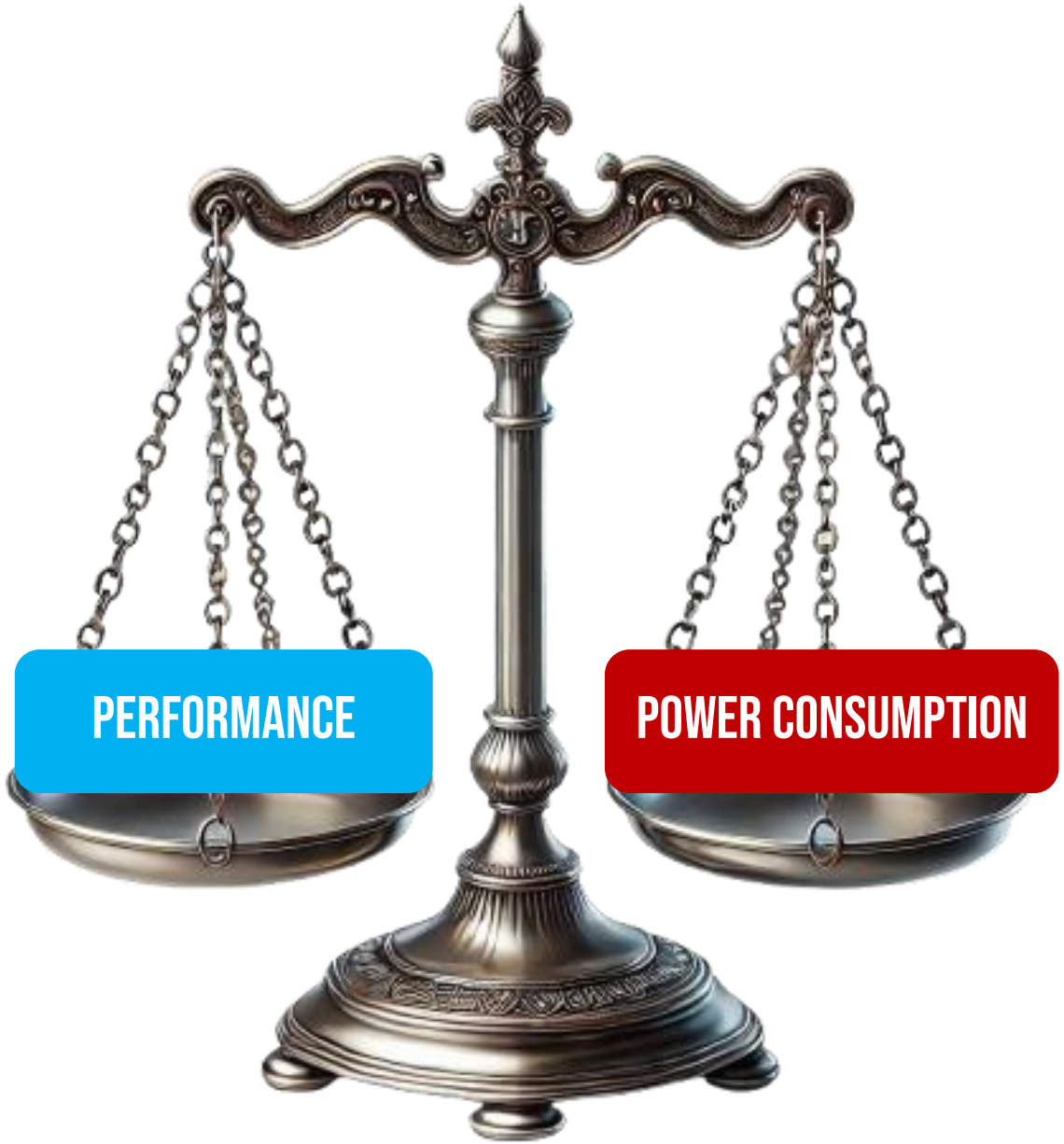
$$P = \alpha \cdot C \cdot V^2 \cdot f$$

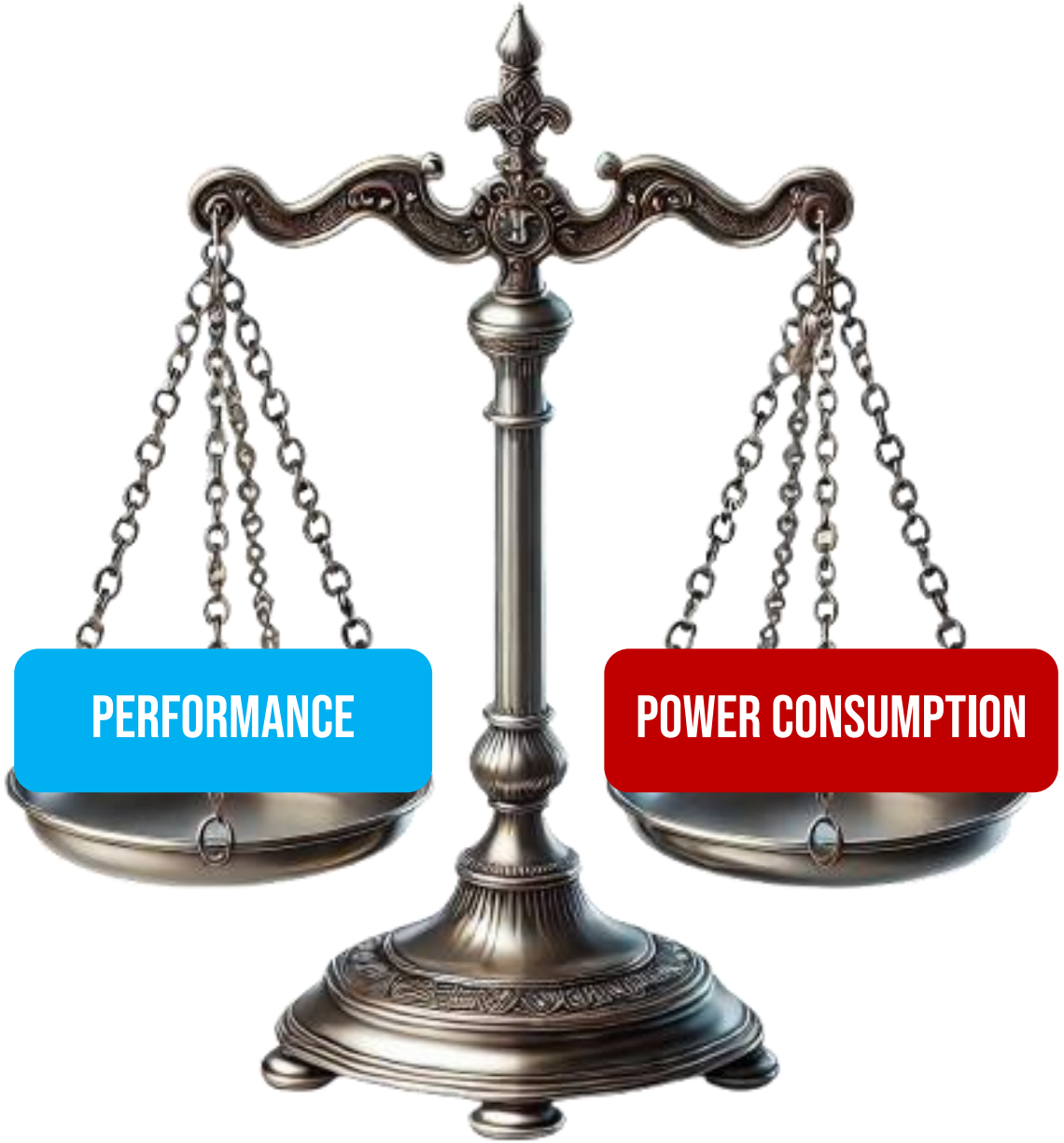
where:

- P: Total power consumption of the processor.
- α: Activity factor, representing the fraction of transistors actively switching in each clock cycle. It ranges from 0 to 1.
- C: Capacitance of the processor's circuits, a measure of how much charge the circuits can hold.
- V: Supply voltage, or operating voltage, applied to the processor.
- f: Operating frequency (clock speed) of the processor, which directly influences the speed of operations.

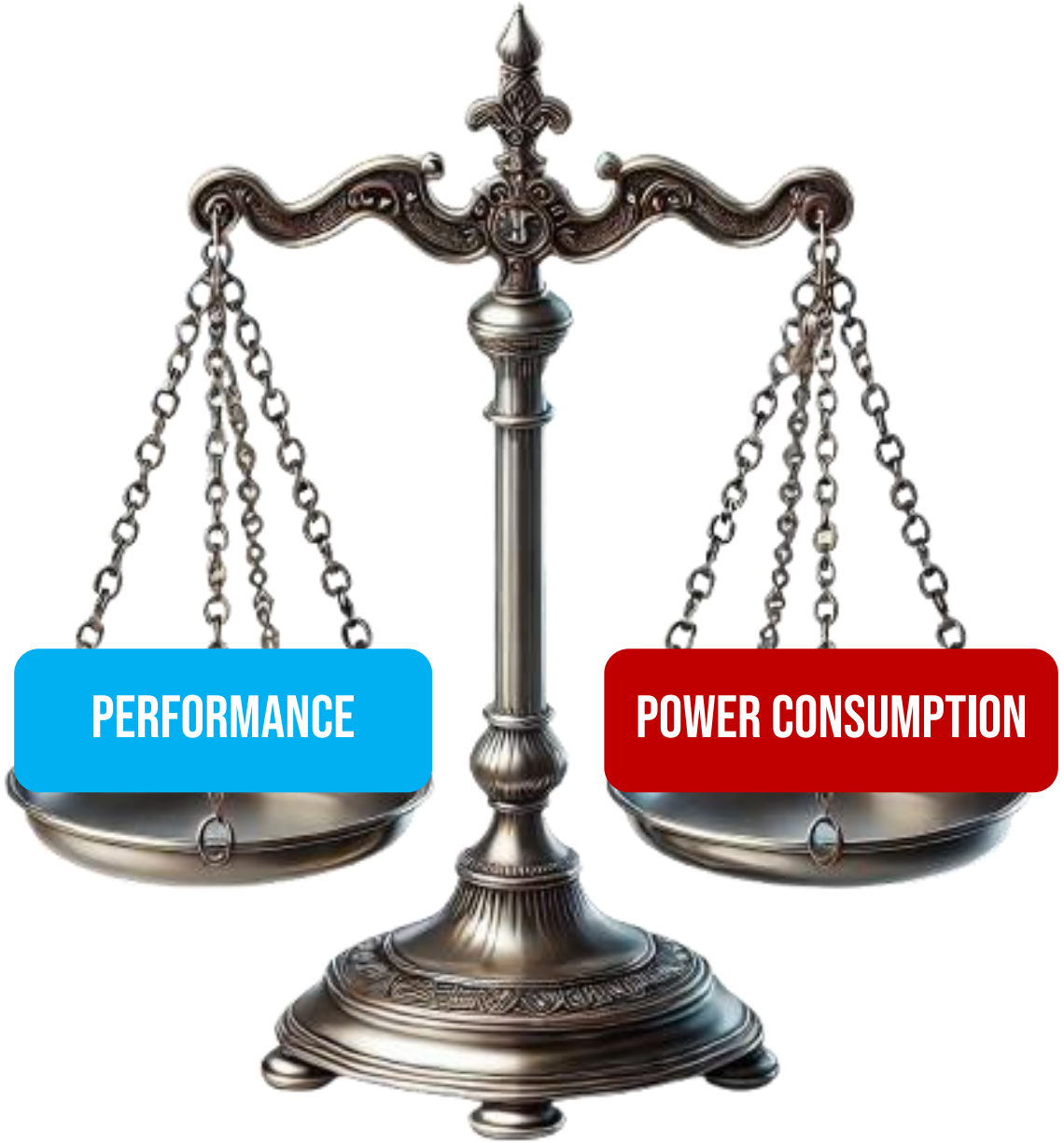
COMPUTER ARCHITECTURE IS THE ART AND SCIENCE OF TRADEOFFS!




PERFORMANCE
POWER CONSUMPTION



$$P = \alpha \cdot C \cdot V^2 \cdot f$$



$$P = \alpha \cdot C \cdot V^2 \cdot f$$

Higher Performance ~ Higher Speed

Higher Speed ~ Increase working frequency (f)

GPU & Parallelism

GPUs, or Graphics Processing Units, are parallel processors designed to handle multiple tasks simultaneously. Unlike CPUs, which focus on sequential processing with a few powerful cores, GPUs have thousands of smaller cores optimized for parallel execution, allowing them to process large blocks of data at once. For a program to benefit from GPU power, it must have a parallel nature or be parallelizable, meaning tasks can be divided into smaller, independent operations that can run concurrently across GPU cores.

Where GPUs Excel: Example 1

Image and Video Processing

GPUs are optimized for manipulating large matrices, making them ideal for image and video processing. When rendering high-resolution images or processing video frames, GPUs can handle the parallel nature of pixel and frame-based operations, such as color transformation, shading, and rendering effects. By executing operations for thousands of pixels simultaneously, GPUs achieve faster processing times, which is critical in real-time rendering for video games, 3D modeling, and visual effects.

Where GPUs Excel: Example 2

Deep Learning and AI

Neural networks, especially in deep learning, involve repetitive matrix multiplications across multiple layers of neurons. GPUs excel at these tasks by parallelizing matrix operations—specifically, tensor computations, which involve linear algebra at a massive scale. Libraries like CUDA and cuDNN (from NVIDIA) are optimized to leverage GPU cores for deep learning frameworks such as TensorFlow and PyTorch, enabling accelerated training times and efficient model inference. For example, training a convolutional neural network (CNN) on image data can be significantly faster on a GPU than on a CPU due to the ability to process multiple convolution operations in parallel.

Where CPUs Win: Example 1

General-Purpose Computing

CPUs are designed to handle a wide range of tasks with complex branching logic, which is essential for running operating systems, applications, and user-driven tasks. CPUs feature fewer but more powerful cores optimized for executing instructions in sequence, making them better suited for varied operations that require frequent decision-making and task-switching, such as file management, internet browsing, and office applications.

Where CPUs Win: Example 2

Low-Latency Requirements

CPUs are optimized for low-latency tasks where the system must respond quickly to input. For example, database servers and real-time analytics require low-latency responses to user queries or data updates. With their large cache memory and high single-threaded performance, CPUs provide immediate access to data and can handle I/O operations more effectively than GPUs, which are optimized for large, parallelizable tasks rather than quick task switching.

Where CPUs Win: Example 3

Single-Threaded Performance

Some applications, like code compilation and certain scientific algorithms, are inherently sequential and cannot be parallelized. CPUs have higher clock speeds and superior single-thread performance compared to GPUs, making them ideal for these single-threaded tasks. For instance, certain steps in data analysis, such as indexing or data sorting, are faster on a CPU as they benefit from its focus on sequential processing rather than parallel execution.

GPUs as Throughput-Oriented Processors

GPUs are often described as "throughput-oriented processors," meaning they are designed to maximize the volume of data processed over time, rather than focusing on minimizing the time taken for each individual operation. This high throughput is achieved by utilizing a large number of simpler cores that operate in parallel, enabling GPUs to handle substantial amounts of data simultaneously. Unlike CPUs, which are optimized for low latency and excel in sequential processing tasks, GPUs are optimized for throughput, making them particularly effective for applications that require processing large datasets in parallel. This design makes GPUs highly suitable for tasks such as image processing, scientific simulations, and deep learning, where handling many operations at once is essential for performance.

Why GPUs Operate at Lower Frequencies

GPUs are considered "low working frequency processors," meaning they operate at lower clock speeds compared to CPUs. While CPUs are designed with high clock frequencies to execute individual tasks quickly, GPUs focus on parallelism rather than speed per core. By using lower clock frequencies, GPUs consume less power per core, allowing them to support thousands of cores running simultaneously. This design makes them highly efficient for tasks requiring massive parallel computation, such as rendering graphics, running machine learning models, and processing large data arrays, where the collective processing power of many cores is more beneficial than the high-speed performance of a few cores.

Content

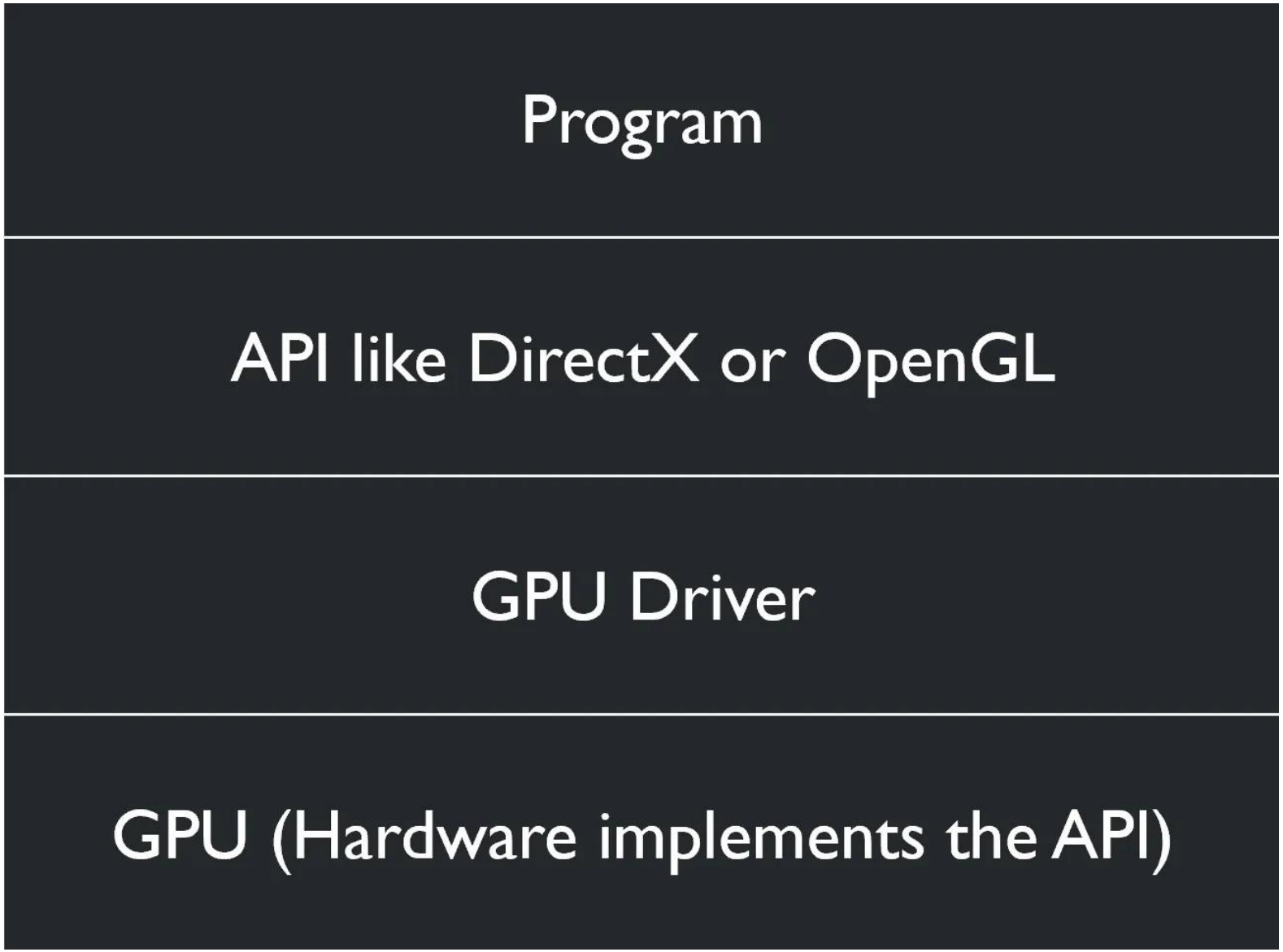
- GPUs
  - **Story of NVIDIA GPUs**
- FPGA
- Accelerator
- Tradeoff of processors

Story of NVIDIA GPUs

- Early GPUs: specific-function co-processors (ASICs) for graphics
- Programmability: Graphics APIs, e.g., DirectX and OpenGL
- Programmability required:
  - Expertise in Computer Graphics
  - Data Transformation Techniques

ASICs: Application Specific Integrated Circuits

An ASIC (Application-Specific Integrated Circuit) is a type of hardware designed to perform a specific task or set of tasks very efficiently. Unlike general-purpose processors, like CPUs, which are versatile and can handle a wide variety of instructions, ASICs are custom-built to perform only one particular function or application. Because of this specialization, ASICs can operate at much higher speeds and lower power consumption than general-purpose processors when handling their designated tasks. ASICs are commonly used in devices that need to execute repetitive operations very quickly and with high efficiency, such as image processing, network devices, audio processing, and specialized parts of smartphones. For example, in a smartphone, an ASIC might be dedicated to handling image signal processing (ISP) for the camera, allowing it to quickly enhance photos by adjusting colors, brightness, and noise with minimal power consumption. However, the downside of ASICs is their lack of flexibility—once manufactured, they cannot be repurposed for other tasks, and designing them is costly and time-consuming. This trade-off makes ASICs ideal for high-volume applications where their speed and efficiency justify the cost of custom design.


Program
API like DirectX or OpenGL
GPU Driver
GPU (Hardware implements the API)

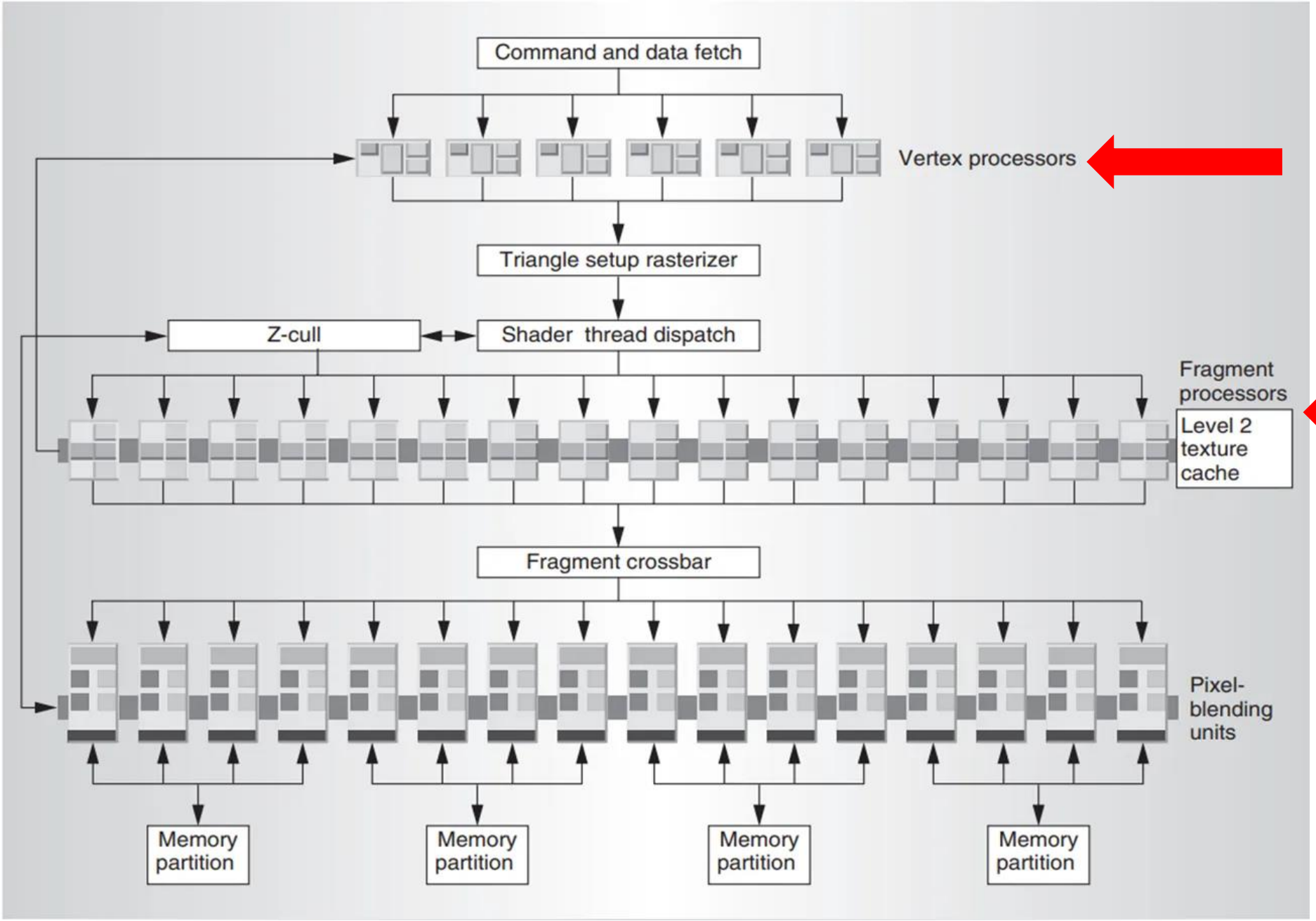
Pre-requisites Terminology

> ***Shader**: a computer program performing graphics-related tasks*
>
> ***vertex**: a data structure describing a certain attribute, e.g., the position of a point in 2D or 3D space, or multiple points on a surface.*
>
> ***Vertex shader**: a program that transforms each vertex's 3D position in virtual space to the 2D coordinate at which it appears on the screen.*
>
> ***Pixel or fragment shader**: a program that computes the color, brightness, contrast, and other attributes of a single pixel or fragment.*

Early GPUs

- Early GPUs (before 2007) architecture:
  - Separate stages of vertex and pixel processors

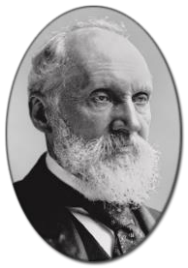
EARLY GPUS: THE ERA BEFORE CUDA UNLEASHED GENERAL-PURPOSE POWER



Montrym, J. & Moreton, Henry, "The GeForce 6800," Micro (2005), IEEE. 25. 41–51. 10.1109/MM.2005.37

Early GPUs





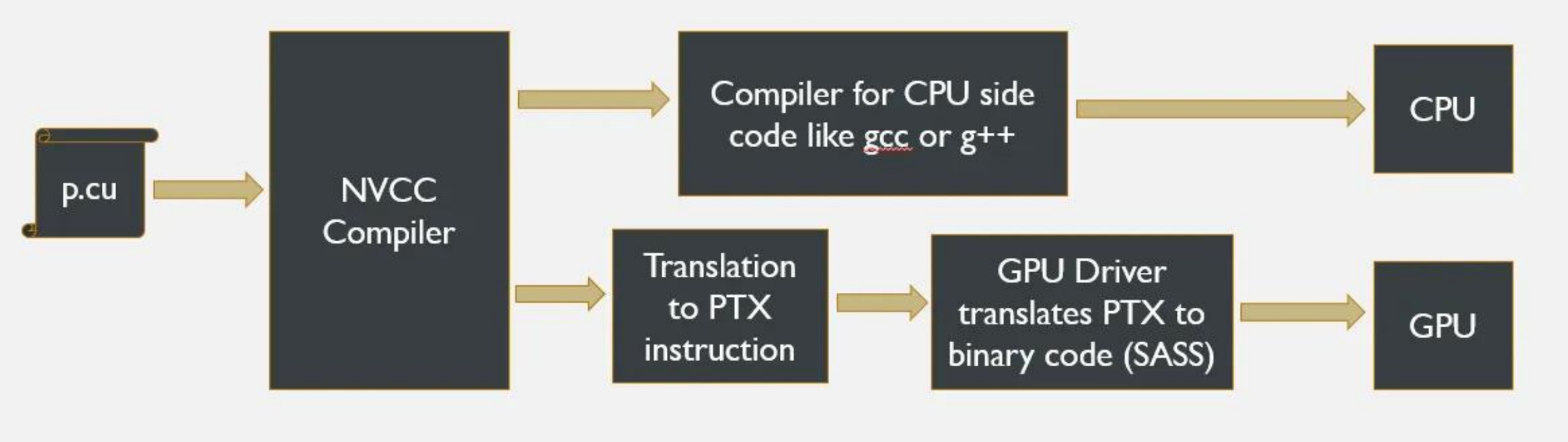
uArch.	Year	CUDA?	#Trans	Fab. Tech.	OpenGL	DirectX	GPU Cards
Fahrenheit	1998	No!	7-15M	350-240nm	1.2	5.0, 6.0	Vanta, Riva Models
Celsius	1999	No!	20-29M	180-150nm	1.5	7.0	**GeForce 256** GeForce 2 series
Kelvin	2001	No!	36-57M	150nm	1.5	8.0	**GeForce 3** and 4 series
Rankine	2003	No!	~125M	150-130nm	2.1	9.0a	GeForce 5 (or FX) series
Curie	2004	No!	~220M	130-80nm	2.1	9.0c	GeForce 6 and 7 series

MOORE'S LAW EFFECT IS EVIDENT IN THE TABLE ON HOW THE NUMBER OF TRANSISTORS INCREASES OVER THE YEARS.

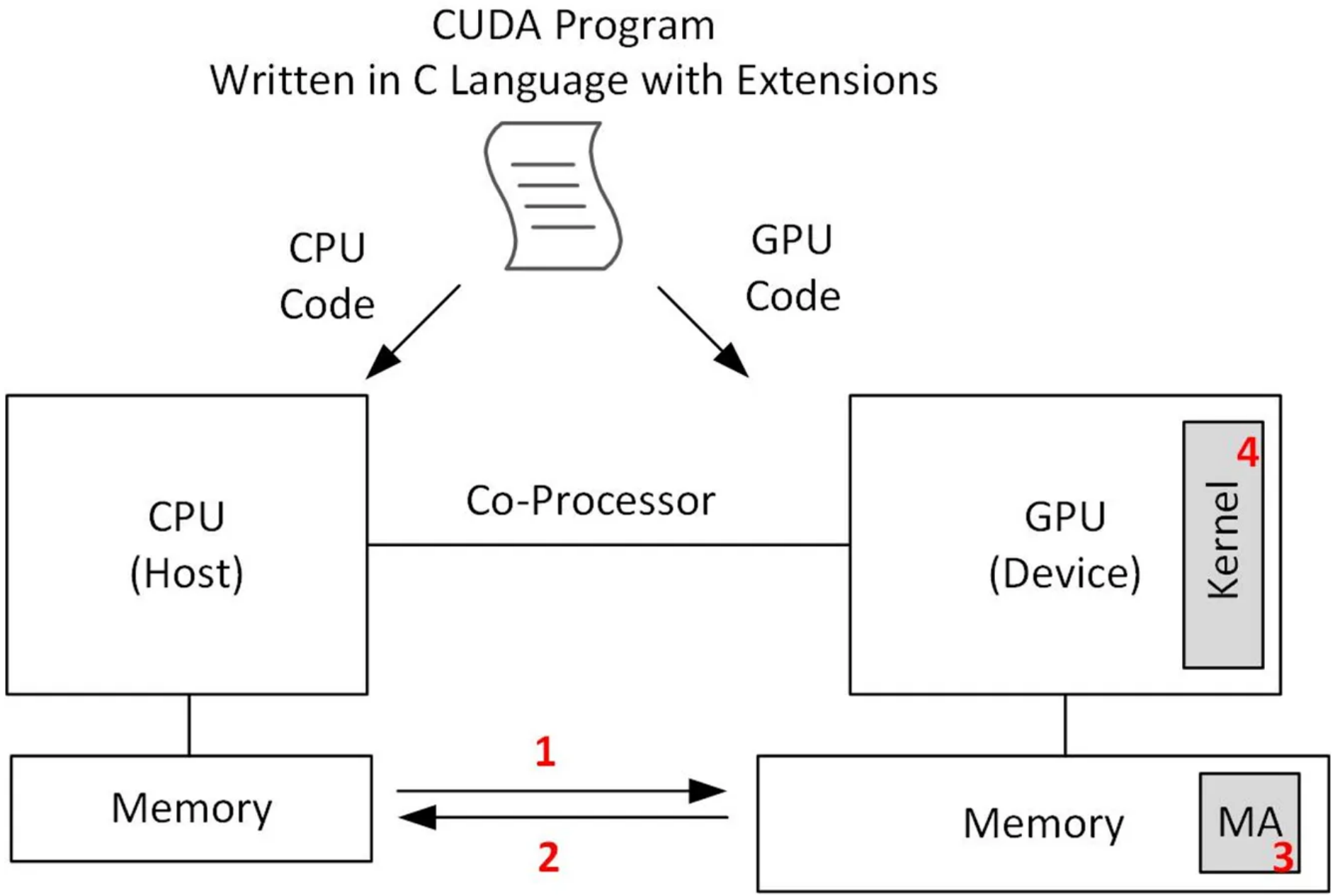
The Big Shift: CUDA Revolutionizes GPU Computing

- **CUDA**: Compute Unified Device Architecture
- A parallel computing **framework platform** and **API** allowing software to use certain types of NVIDIA GPUs for general-purpose computing
- CUDA programming is possible for C/C++ programmers can use ‘CUDA C/C++’ for programming NVIDIA GPUs

From Code to Execution: The CUDA Process

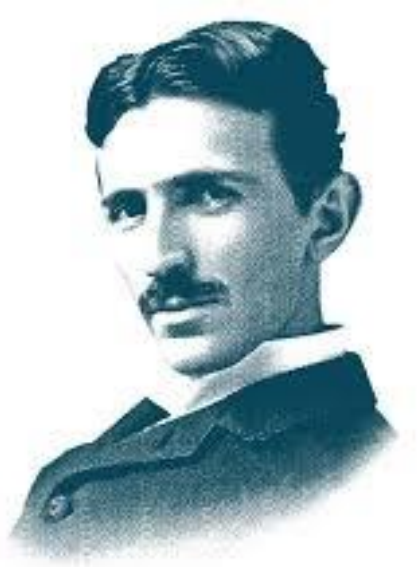



```
$ nvcc my_program.cu -o my_program
```


CUDA Program
Written in C Language with Extensions
CPU
Code
GPU
Code
CPU
(Host)
Co-Processor
GPU
(Device)
Kernel
4
Memory
1
2
Memory
MA
3

Microarchitecture Innovations Enabling the CUDA Revolution

- At 2006, NVIDIA introduced **Tesla** Microarchitecture
- Implementing a UNIFIED SHADER MODEL
  - (remember Separate stages of vertex and pixel processors)



- Benefits of the Unifications
  - Better of Management of Hardware Resources
  - Load Balancing between Vertex and Pixel (fragment) stage
  - Simpler GPU Design

- Cores became: (1) Sequential (2) Scalar – meaning being able to work on only one computation task at a time

Microarchitecture Innovations Enabling the CUDA Revolution

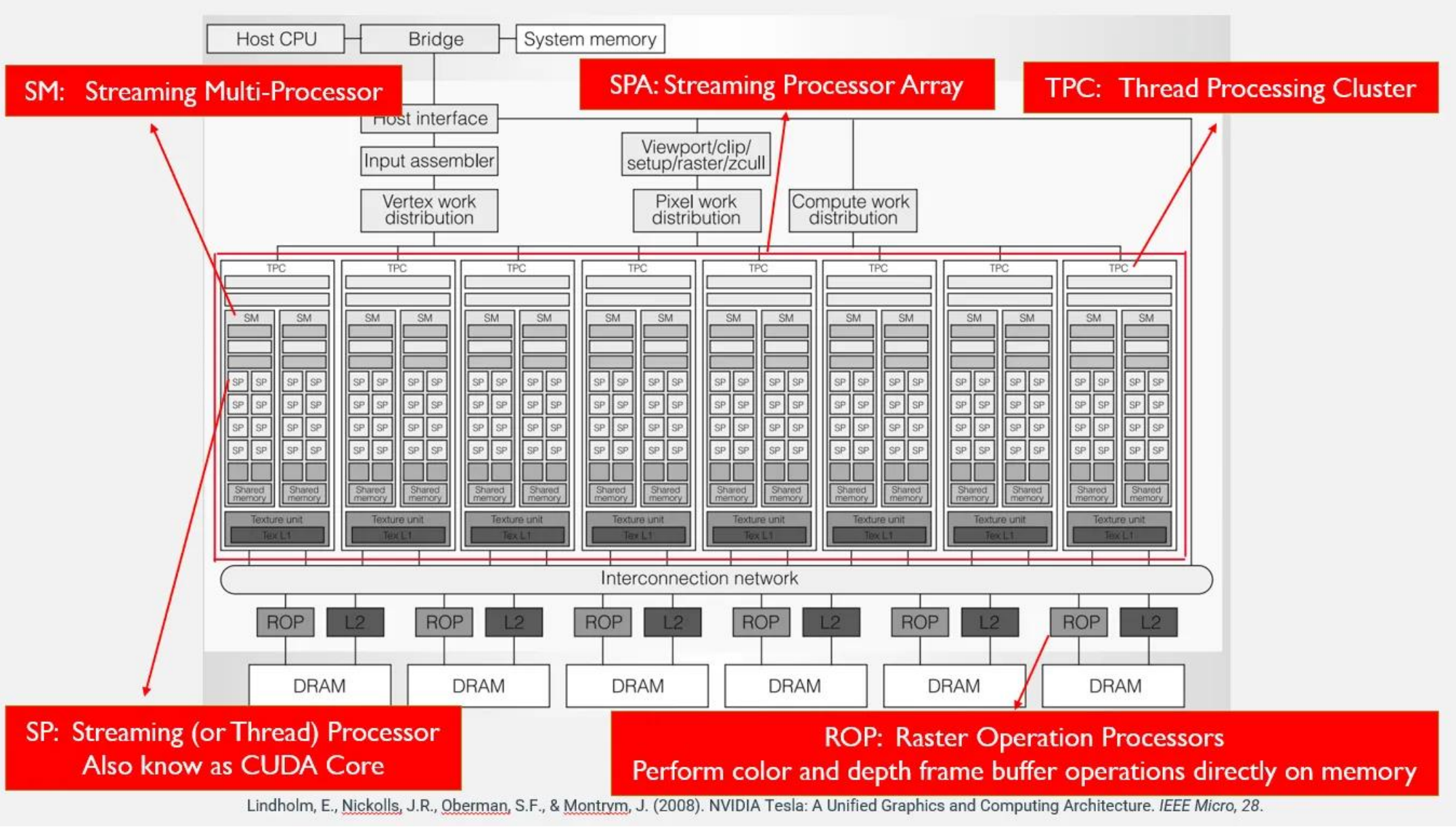
- Changed their names to: CUDA cores
- They grouped together in Streaming Multiprocessors (SM), which replaced split stages of vertex and fragment units
- Each SM receives Thread Blocks that are composed of groups of threads in the number of 32, which are called **warp**.
- All threads in a warp execute the same instruction at the same time but on different data (**SIMT: Single Instruction Multiple Thread**).

Microarchitecture Innovations Enabling the CUDA Revolution

- Jonah Alben: the senior vice president of GPU engineering at NVIDIA

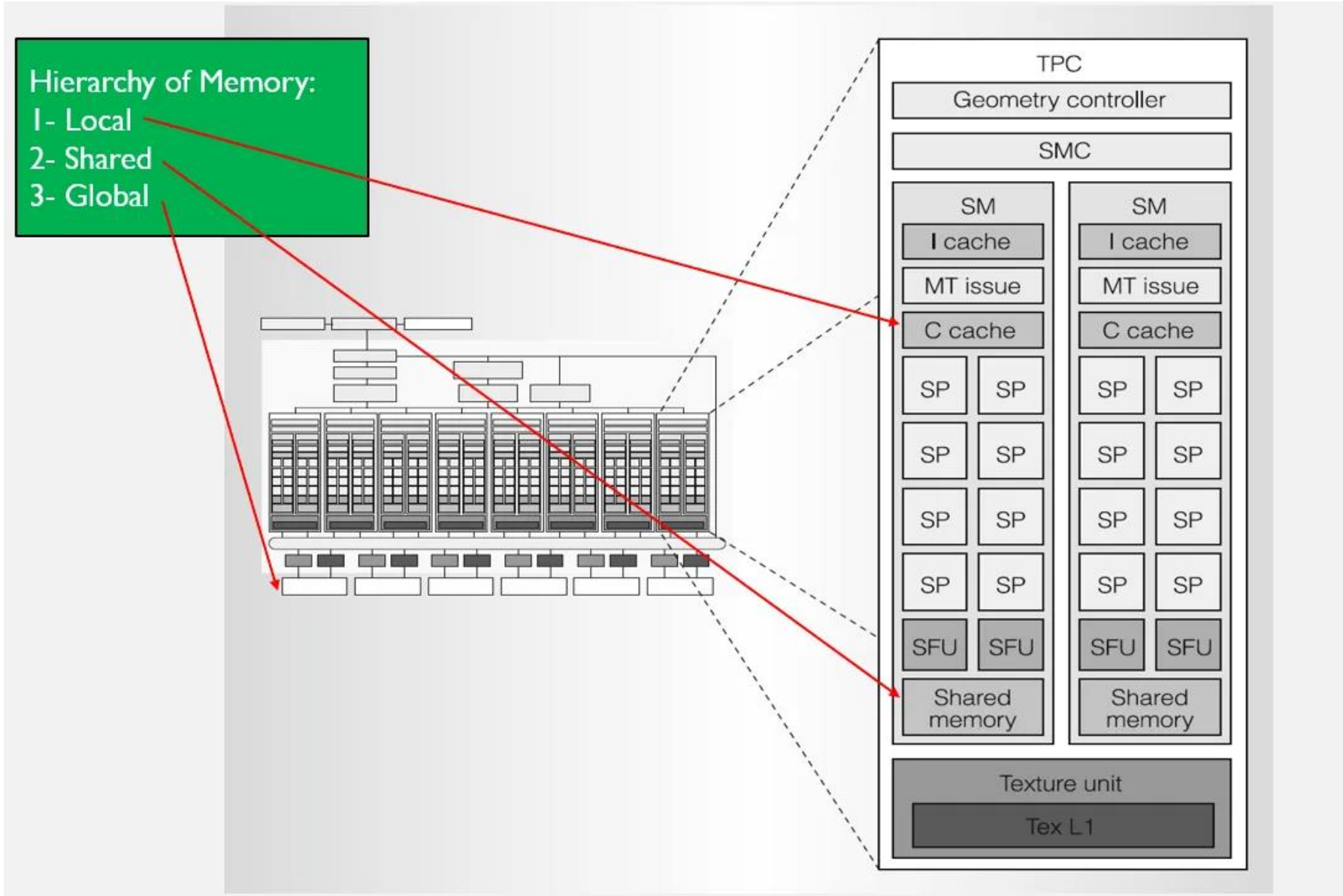
> *We pretty much threw out the entire shader architecture from NV30/NV40 and made a new one from scratch with a new general processor architecture (SIMT), that also introduced new processor design methodologies.*

NVIDIA Tesla 2006 Architecture



Lindholm, E., Nickolls, J.R., Oberman, S.F., & Montrym, J. (2008). NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro, 28*.

Inside a TPC



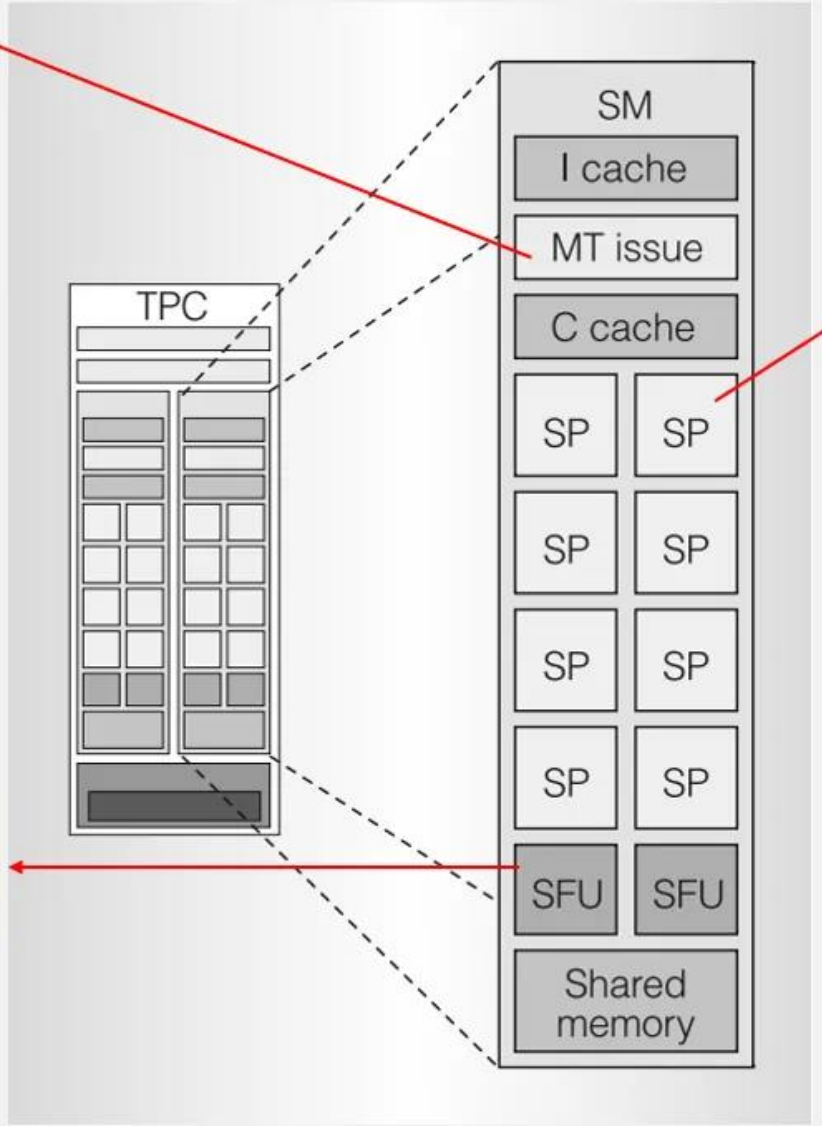
Lindholm, E., Nickolls, J.R., Oberman, S.F., & Montrym, J. (2008). NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro, 28*.

Inside a SM

Enabling/ disabling threads in warp when their PC converge or diverge

Round Robin scheduling for warps
OR
Greedy-Fetch Policy: Fetching from one warp till missing on cache (needing data)

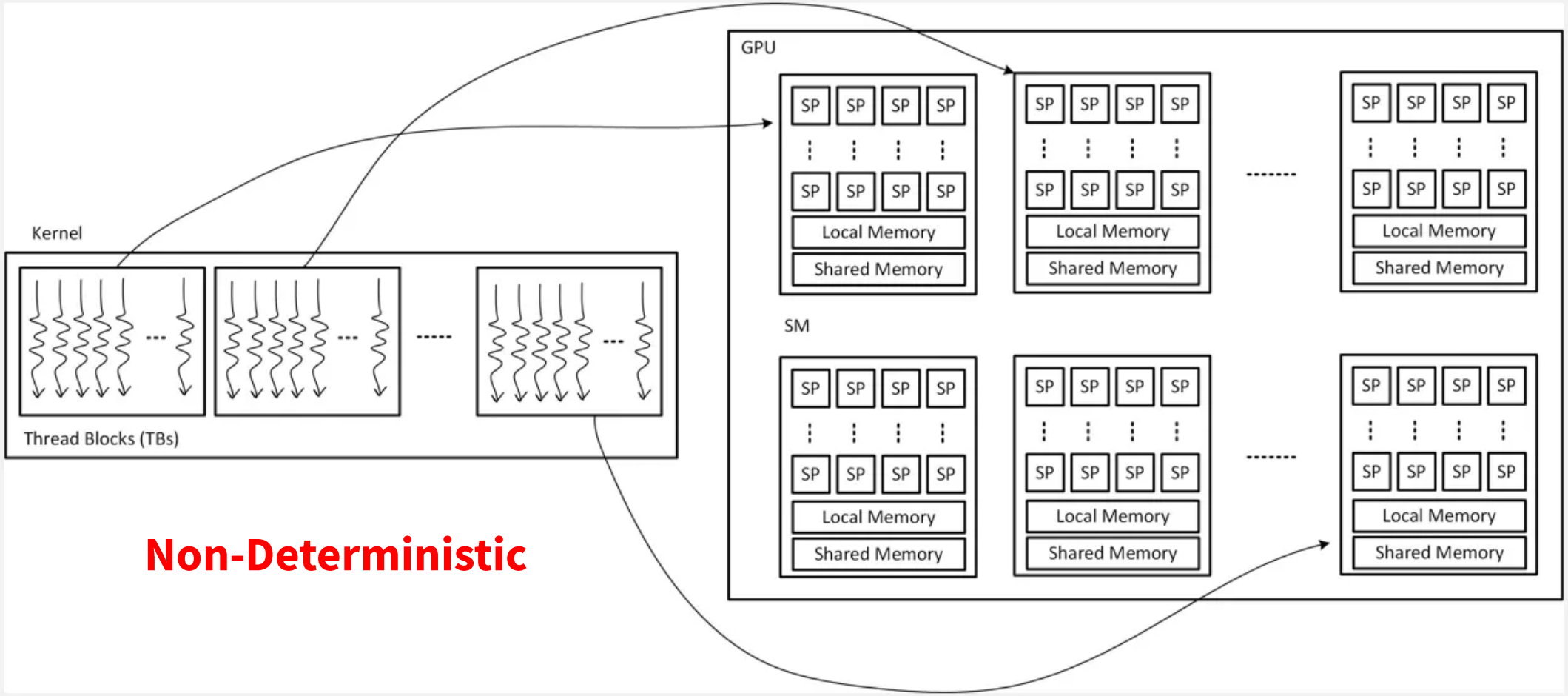
Special Function Unit (SFU)
For complex math calculations



Able to do integer (24 bit) and float32 operations per clock (float64 operations must be done in software)

Lindholm, E., Nickolls, J.R., Oberman, S.F., & Montrym, J. (2008). NVIDIA Tesla: A Unified Graphics and Computing Architecture. *IEEE Micro, 28*.

THE EXECUTION MODEL!

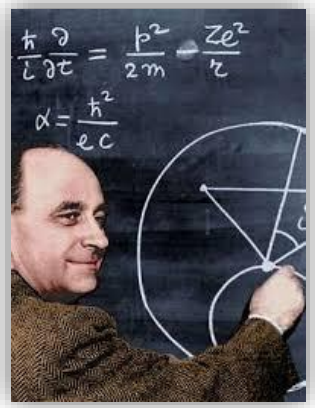


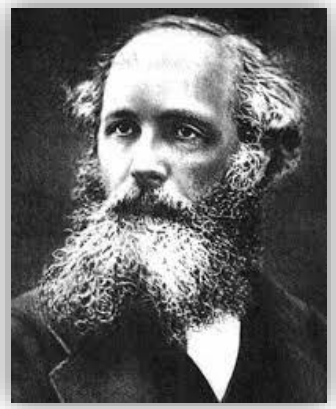
Non-Deterministic

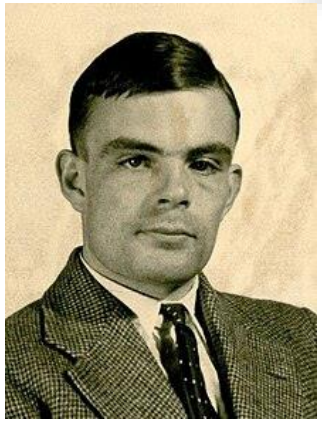
https://github.com/ehsanyousefzadehasl/PCwGPGPUs

Over time

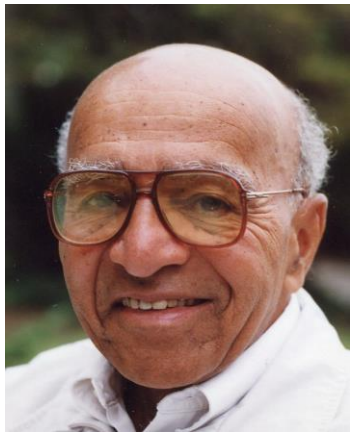
- Fermi (2010)
- Kepler (2012)
- Maxwell (2014)
- Pascal (2016)
- Volta (2017)
- Turing (2018)
- Ampere (2020)
- Hopper (2022)
- Ada Lovelace (2022)
- Blackwell (2024)









More?

- **Next Lecture we will see Advanced Capabilities of NVIDIA GPUs, and learn how to program with CUDA.**

- **NVIDIA GPUs story**

  NVIDIA GPUs story. | by Ehsan Yousefzadeh-Asl-Miandoab | Medium

- **Parallel Computing with GPUs**

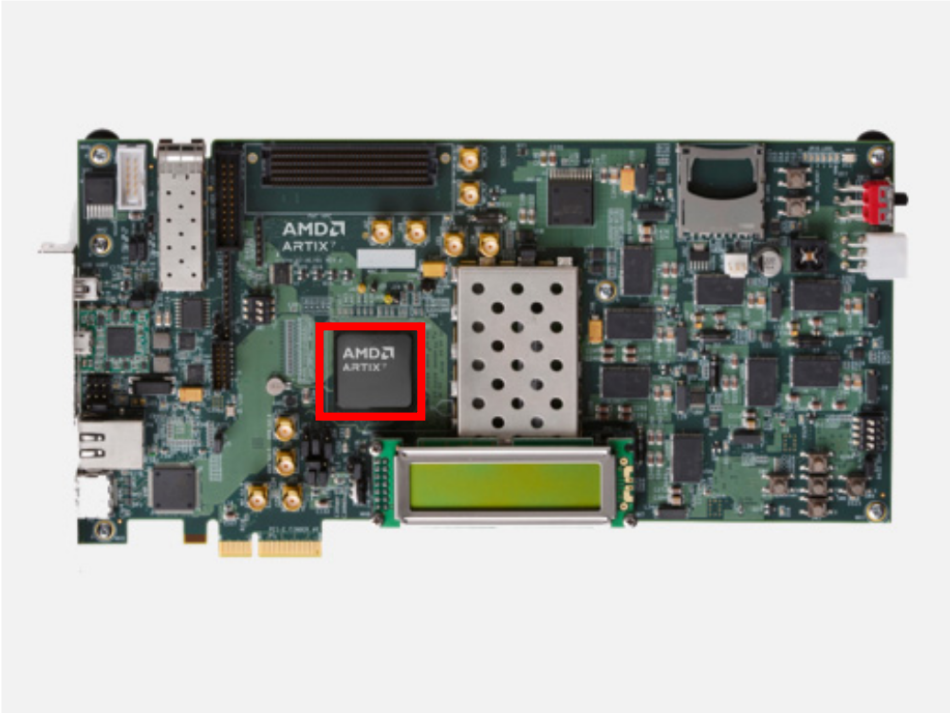
  ehsanyousefzadehasl/PCwGPGPUs: Parallel Computing with GPGPUs

Content

- GPUs
  - Story of NVIDIA GPUs
- **FPGA**
- Accelerator
- Tradeoff of processors

FPGAs: Field Programmable Gate Array

- A hardware that can be configured after being manufactured
- **Initial Purpose**:
  - **Prototyping** digital circuits and testing Logic Designs (before being manufactured as an ASIC)
  - Implementing small, custom digital circuits (using the FPGA as that circuit)
- **Flexibility** was considered more important, not performance or power
- Early Challenges:
  - A **deep understanding of hardware design**
  - **Low-level hardware description languages (HDLs)** like **Verilog** and **VHDL**
- Early FPGA work demanded a solid grasp of
  - **Digital logic** concepts,
  - **Timing analysis**, and **circuit behavior**
  - Hardware Experts (Digital Electronic Engineers)



Click to Enlarge

AMD Artix 7 FPGA AC701 Evaluation Kit

by: AMD

AMD

The Artix™ 7 FPGA AC701 Evaluation Kit features the leading system performanc quickly prototyping for your cost sensitive applications.

Price: $1,678.00
Part Number: EK-A7-AC701-G
Lead Time: 6 Weeks
Device Support: Artix-7

Buy

https://www.xilinx.com/products/boards-and-kits/ek-a7-ac701-g.html

FPGAs Big Players

- **Xilinix** and **Altera**
  - Developed many of the innovations in FPGA technology
    - Advanced Architecture
    - High-level Programming Tools

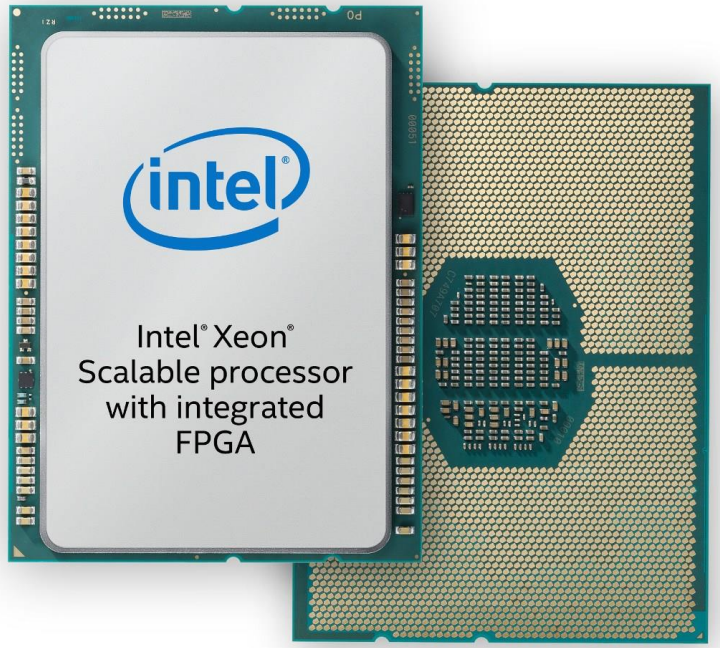
- **Intel** acquired **Altera**
  - integrating Altera's FPGA technology into Intel's product lineup

- **AMD** acquired **Xilinx** in 2022
  - To enhance its data center and high-performance computing offerings with Xilinx's FPGA technology

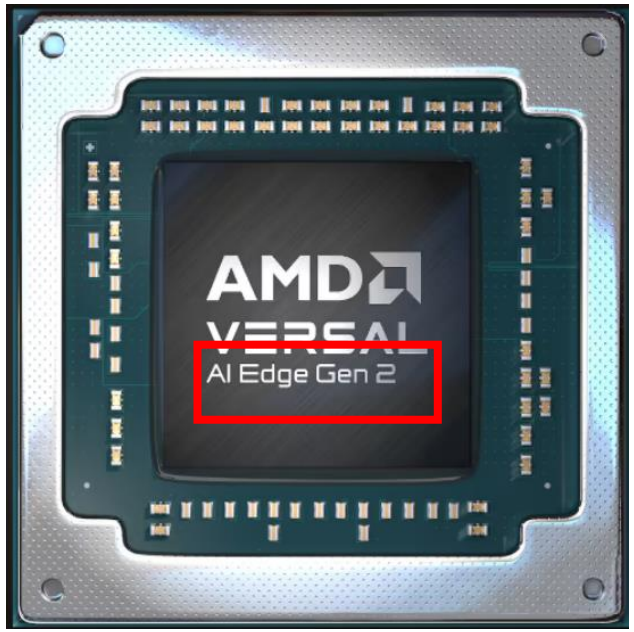
Integration with CPUs

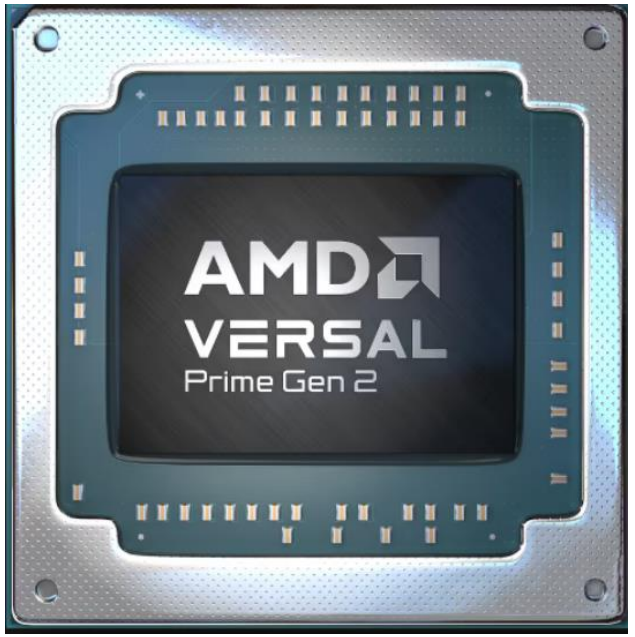
- Both Intel and AMD
  - **integrate FPGAs with CPUs on a single chip** or in multi-chip modules
    - combines the **flexibility** and **parallelism** of FPGAs with the general-purpose processing power of CPUs
    - **Heterogeneous Computing**

- **Benefits**
  - Sharing Same Memory Space
  - Reducing Latency between CPU/FPGA
  - FPGA acceleration for tasks, like:
    - Encryption
    - Compression
    - Machine Learning Inference
    - Data Filtering



Integration with CPUs





AI Edge Series Gen 2

Next-generation AI Engines, high-performance integrated CPUs, and programmable logic enabling preprocessing, AI inference, and postprocessing for AI-driven embedded systems—all in a single device.

Prime Series Gen 2

High-performance integrated CPUs, programmable logic, and 8K video processing for next-level classic embedded systems across a wide range of markets.

https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal.html

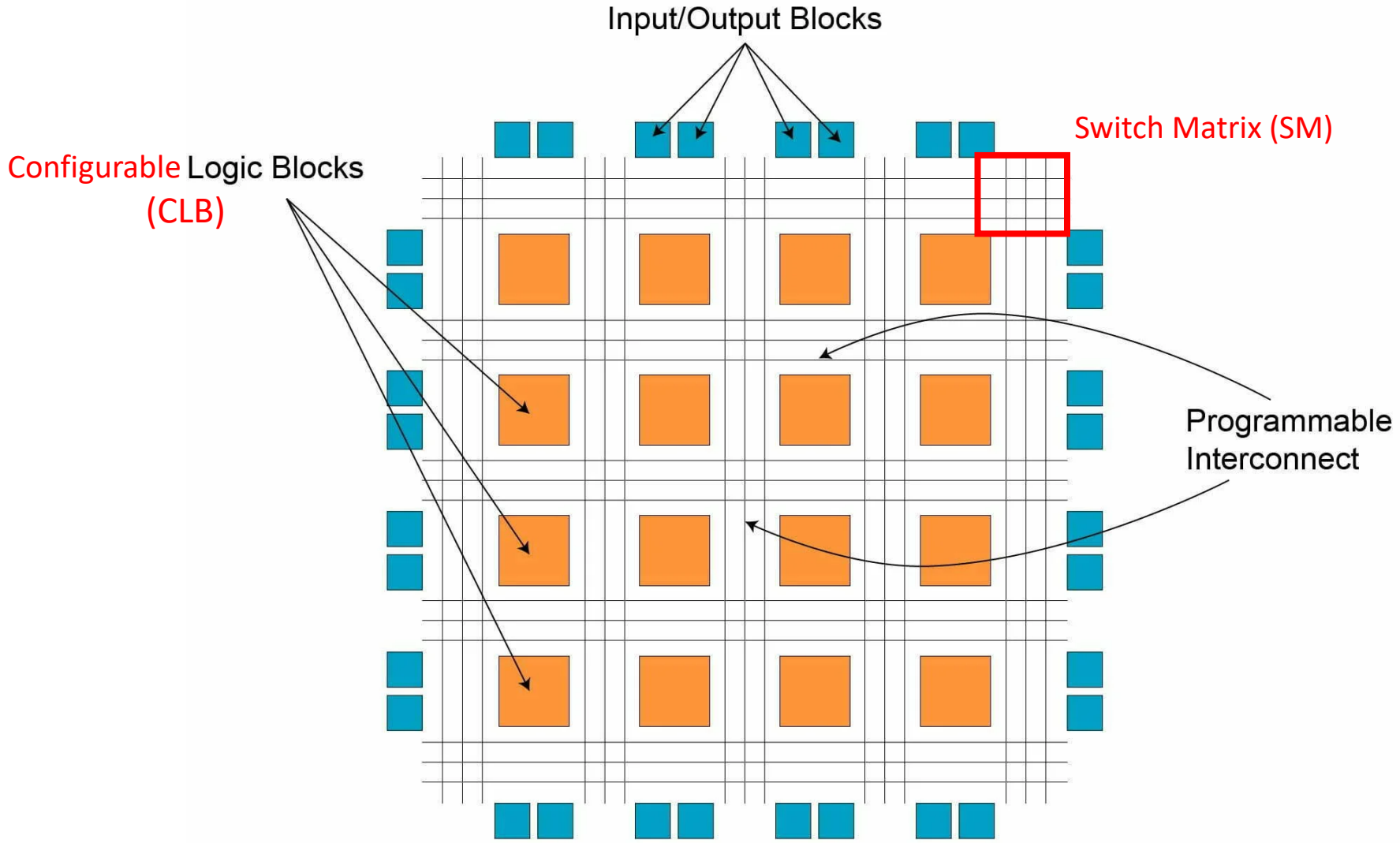
Applications of FPGAs

- **Initial Applications**: Early on, FPGAs were mainly used in **telecommunications**, **signal processing**, and **embedded systems** where custom digital logic was required but dedicated ASICs *weren't economical*.
- **Modern Applications**:
  - **Data Centers**: For accelerating tasks like **AI inference**, **data processing**, and **high-speed networking**.
  - **5G and Telecommunications**: For **signal processing**, **packet processing**, and handling data transfer in base stations.
  - **Automotive**: Used in advanced driver-assistance systems (ADAS) and autonomous vehicle applications, where reconfigurability allows for updates as new features are developed.
  - **Financial Services**: In high-frequency trading, where FPGAs can execute complex algorithms with extremely low latency.
  - **Aerospace and Defense**: For **radar processing**, **encryption**, and **secure communications**, where *real-time processing* and reliability are critical.

Current State and Modern Applications

- **State of FPGAs**: FPGAs are now key components in **data center** and **edge computing** infrastructures, used for both **prototyping** and **production** systems. With continued investment from Intel and AMD, FPGAs are becoming more tightly integrated with CPUs and GPUs, enhancing performance for a wider range of applications (**Heterogenous Computing**).
- **Modern Applications**: FPGAs are focusing on accelerating complex, compute-heavy tasks in areas such as:
  - **Machine Learning and AI**: FPGA-based AI inference accelerators are growing in popularity due to their **low latency** and ability to be **reconfigured** for different models or algorithms.
  - **Real-Time Data Processing**: FPGAs are ideal for tasks that require real-time, high-throughput processing, such as **video streaming, IoT data analysis**, and **sensor data processing**.
  - **Cybersecurity**: FPGAs can be used for encryption, decryption, and secure processing, particularly in industries that require high security and flexibility.
  - **Cloud Computing**: FPGAs are now available as part of cloud platforms like AWS (with F1 instances) and Microsoft Azure, providing hardware acceleration for users without needing physical access to an FPGA device.

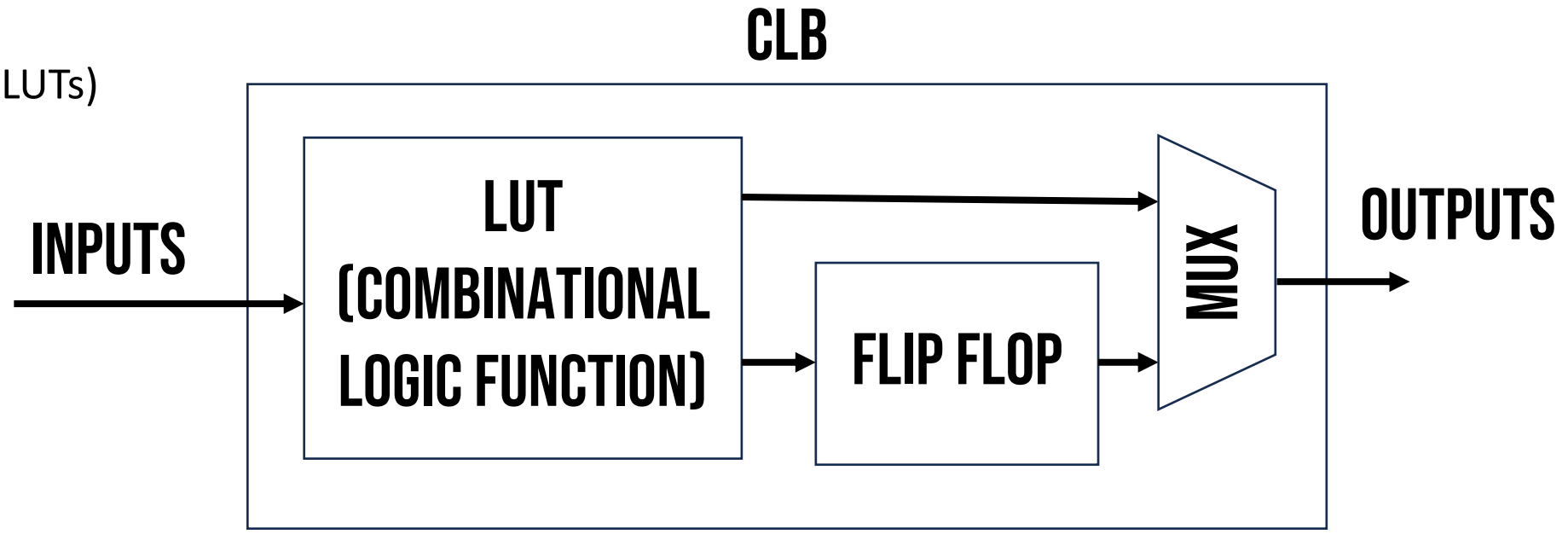
Inside an FPGA



https://www.logic-fruit.com/blog/fpga/fpga-design-architecture-and-applications/

Inside an FPGA

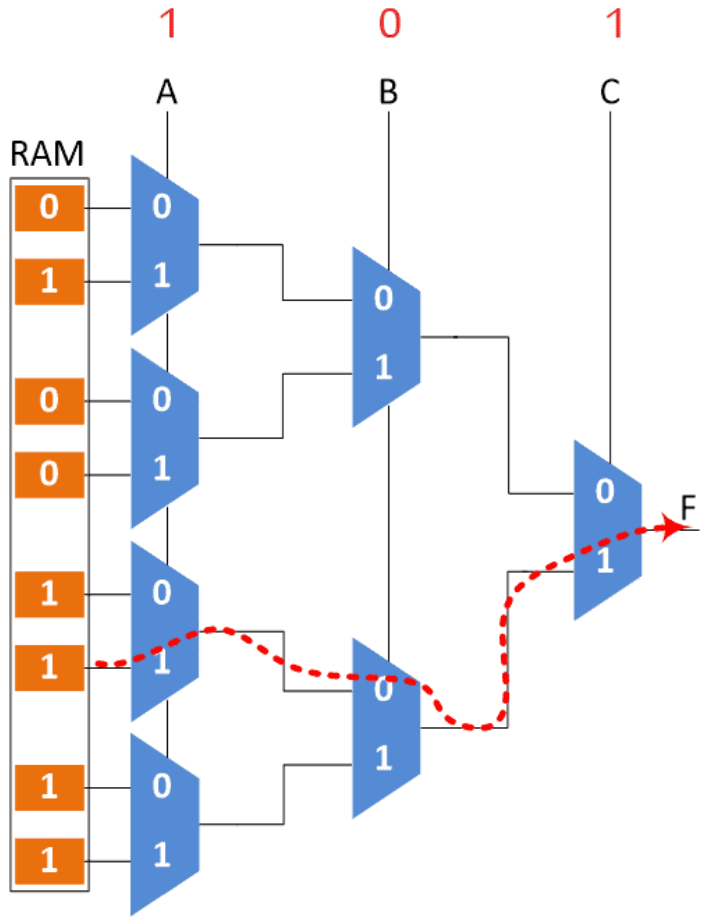
- Configurable Logic Block (**CLBs**): **Processing Units of an FPGA**
  - Can be configured to perform
    - Basic logic functions like AND, OR, NOT, XOR
    - More complex combinational and sequential logic
  - Inside CLBs:
    - Look-Up Tables (LUTs)
    - Flip-Flops (FFs)



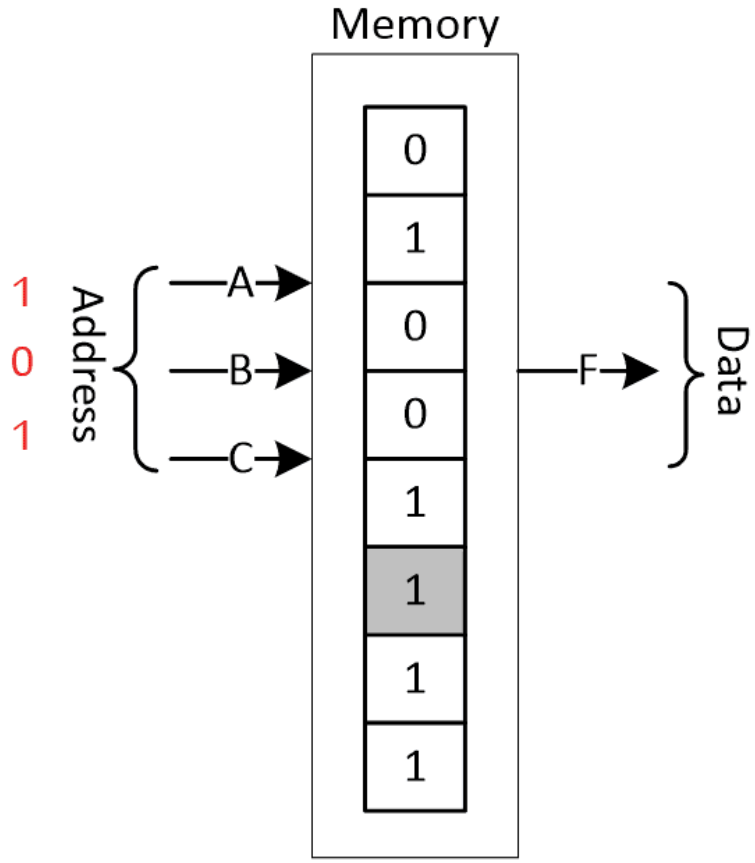
Inside an FPGA

- Interconnects
  - Connecting CLBs, allowing signals to flow between different parts of the FPGA
    - Switch Matrix
- I/O Blocks
  - To interface with external components
- Clocking resources
  - ensures consistent timing throughout the chip, providing synchronized operation for flip-flops and other sequential logic components

Inside an LUT

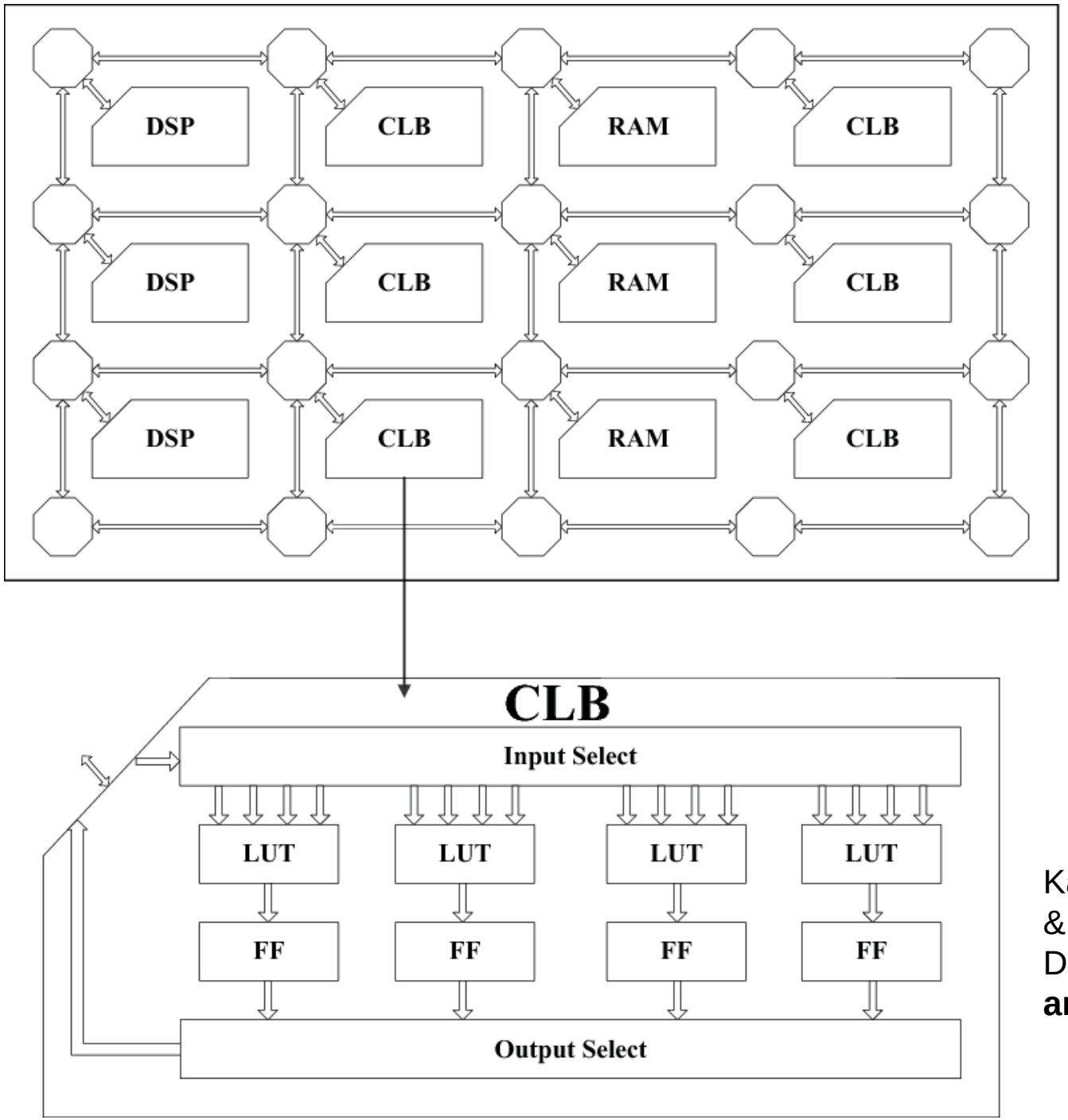


LUT in FPGA



Look-up table (LUT)

https://fpgatek.com/what-is-an-fpga/



Kastner, Ryan & Jung, Chair & Cho, Uk & Rao, Bhaskar & Sherwood, Timothy & Swanson, Steven & Tullsen, Dean. **GUSTO: General architecture design Utility and Synthesis Tool for Optimization**.

Programming an FPGA

- Fundamentally different from programming a CPU or GPU
- Programming is writing instructions for a processor to execute
- Using a Hardware Description Language (HDL) means:
  - Describing the hardware circuitry
- HDL example (Low Level):
  - Verilog (**VERI**fication and **LOG**ic)
  - VHDL (**V**HSIC **H**ardware **D**escription **L**anguage)
    - VHSC: Very High-Speed Integrated Circuit)
- High Level Synthesis (HLS) tools:
  - Vivado HLS

Process of programming an FPGA

1. **Design Description**
   - The developer describes the intended logic

2. **Synthesis**
   - The HDL is synthesized into a **netlist**, which is a description of the logic gates and interconnections needed to implement the design

3. **Place and Route**
   - The FPGA design tool assigns specific CLBs, LUTs, and interconnects on the FPGA to the gates and connections in the netlist. This process defines which blocks perform each function and how they are wired.

Low Level programming with Verilog

BEHAVIORAL LEVEL

REGISTER TRANSFER LEVEL (RTL)

GATE LEVEL

SWITCH LEVEL

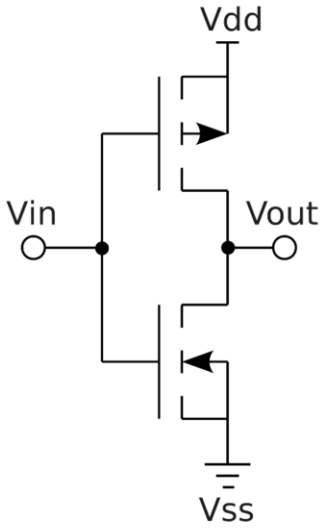
```
not n1(not_a, a);
not n2(not_b, b);

and a0(oa0, D, not_a, not_b);
```

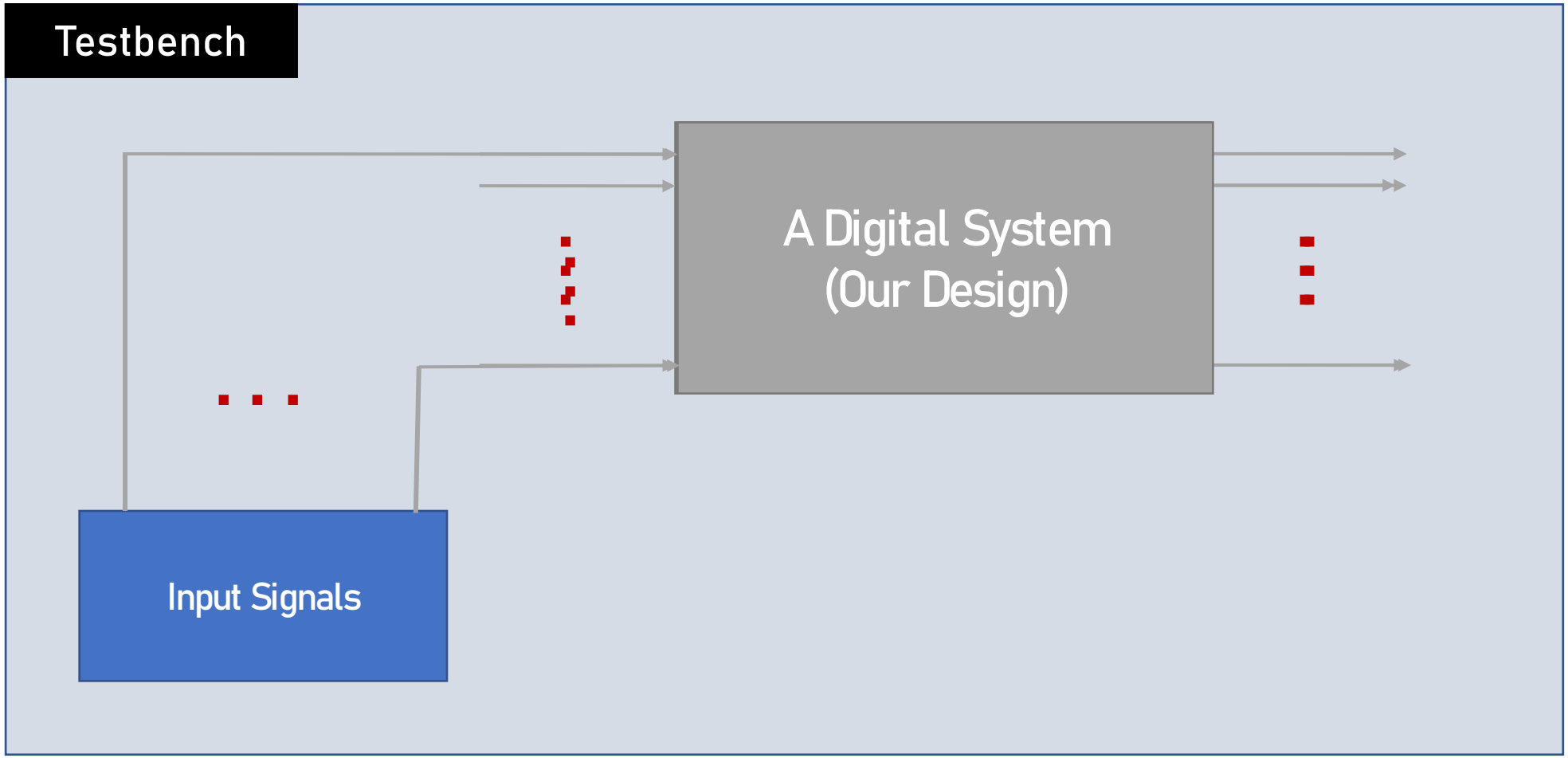
cmos_inverter.v

```
`timescale 1ns/100ps
module cmos_inverter(out,in);
    output out;
    input  in;
    supply0 GND;
    supply1 PWD;

    // out, in, control
    pmos p0(out, PWD, in);
    nmos n0(out, GND, in);
endmodule
```



Verilog Example

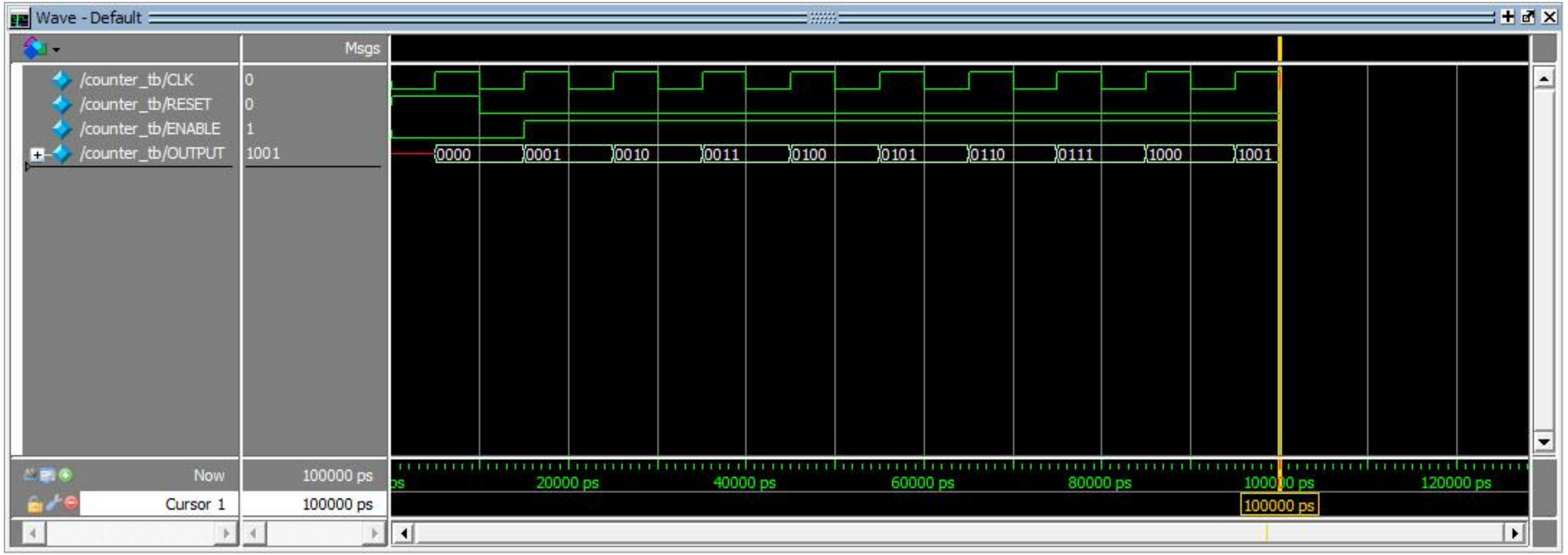


Design: a 4-bit counter

```
`timescale 1ns/100ps
module counter(count, clk, reset, enable);
    input clk, reset, enable;
    output [3:0] count;
    reg [3:0] count;
    always @(posedge clk) begin
        if(reset == 1'b1) begin
            count <= 0;
        end else if(enable == 1'b1) begin
            count <= count + 1;
        end
    end
endmodule
```

Testbench

```
`timescale 1ns/100ps
module counter_tb();
    reg    CLK, RESET, ENABLE;
    wire [3:0] OUTPUT;
    // instantiating from counter module
    counter C0(.count(OUTPUT), .clk(CLK), .reset(RESET), .enable(ENABLE));
    initial begin
        CLK = 0; RESET = 1; ENABLE = 0;
        #10 RESET = 0; // 10 nanoseconds delay
        #5 ENABLE = 1;
    end
    always
        #5 CLK = !CLK;
endmodule
```


Wave - Default
Msgs
/counter_tb/CLK
/counter_tb/RESET
/counter_tb/ENABLE
/counter_tb/OUTPUT
0
0
1
1001
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
Now
100000 ps
Cursor 1
100000 ps
20000 ps
40000 ps
60000 ps
80000 ps
120000 ps
100000 ps

High Level Synthesis (HSL), MM example

```
#include <iostream>

// Define matrix size
#define SIZE 4

// Top-level function for HLS synthesis
void matrix_multiply(float A[SIZE][SIZE], float B[SIZE][SIZE], float C[SIZE][SIZE]) {
    // Perform matrix multiplication
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            C[i][j] = 0;
            for (int k = 0; k < SIZE; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

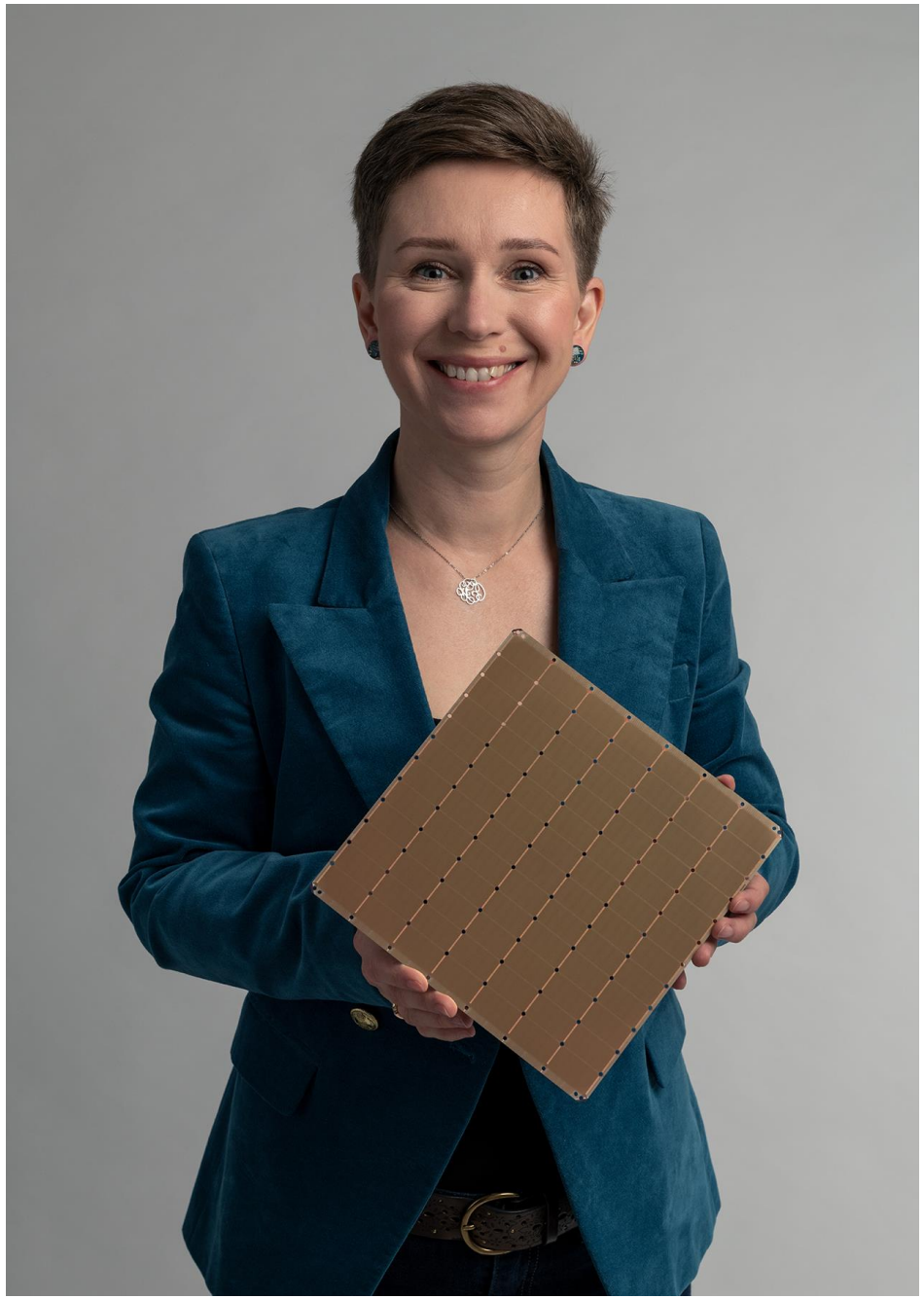
Content

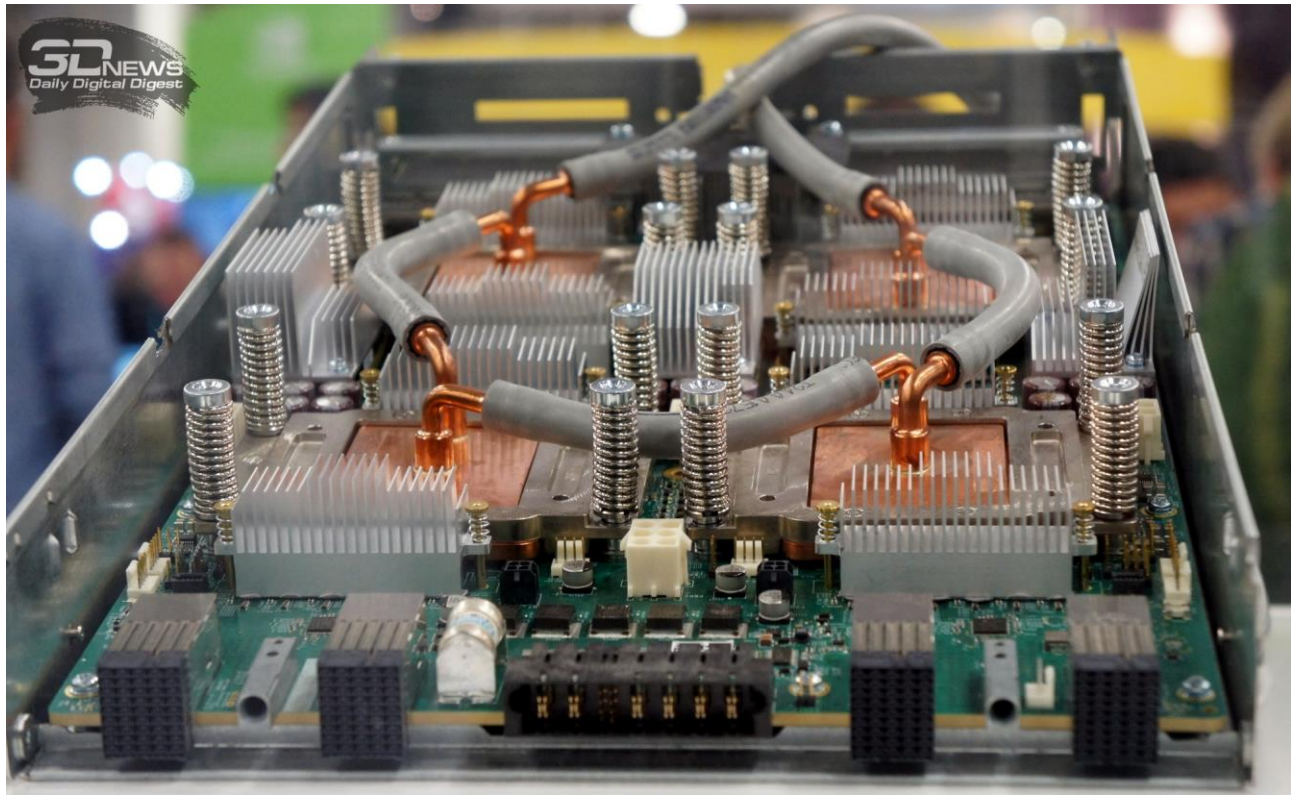
- GPUs
  - Story of NVIDIA GPUs
- FPGA
- **Accelerator**
- Tradeoff of processors

Accelerator

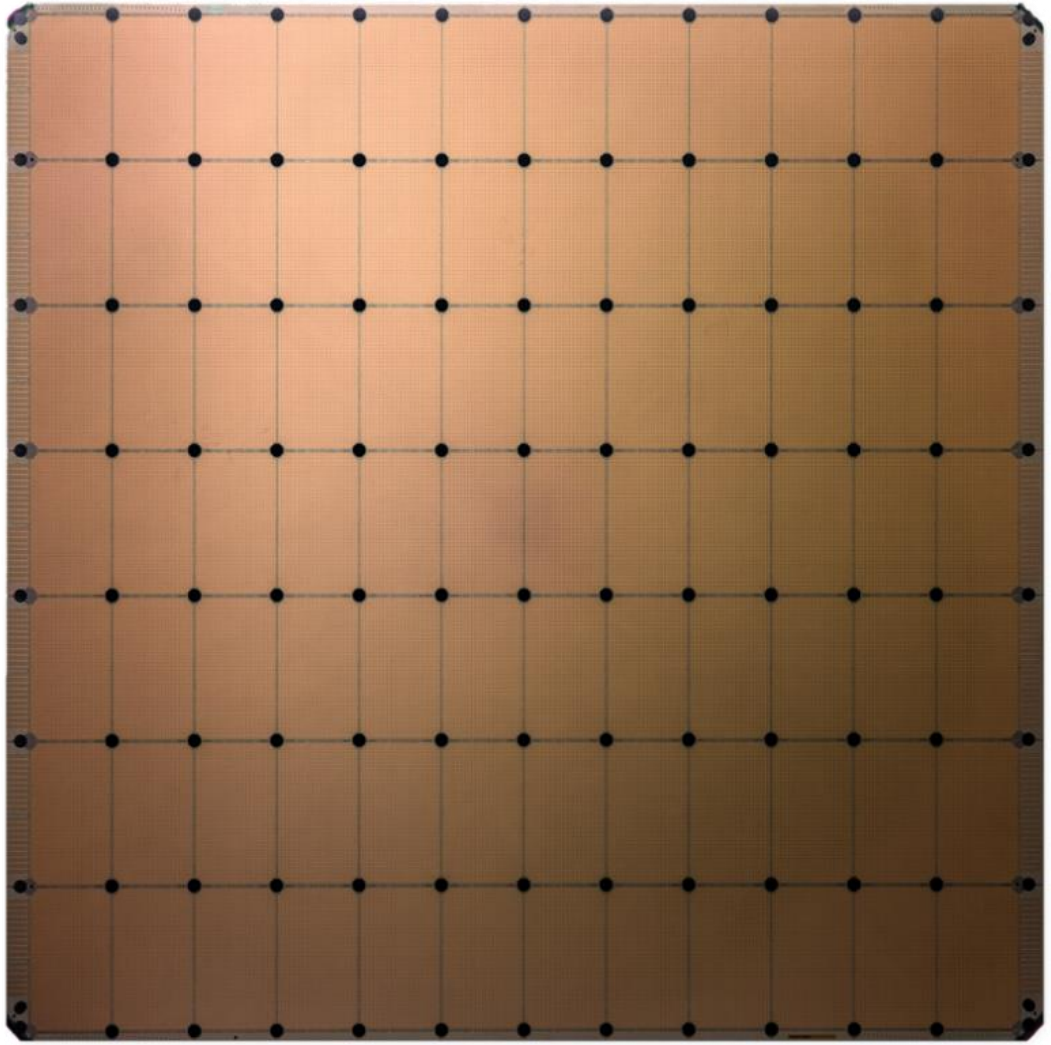
- Accelerator accelerates (compared to CPU)!

- Accelerator is a flexible term
  - AI Accelerator: chips like TPUs and Cerebras
  - General Purpose Accelerators like GPUs
  - Custom Accelerator like ASICs or FPGAs configured for a specific application




3DNEWS
Daily Digital Digest


groq



CEREBRAS WSE-3

46,225mm² Silicon

4 Trillion transistors

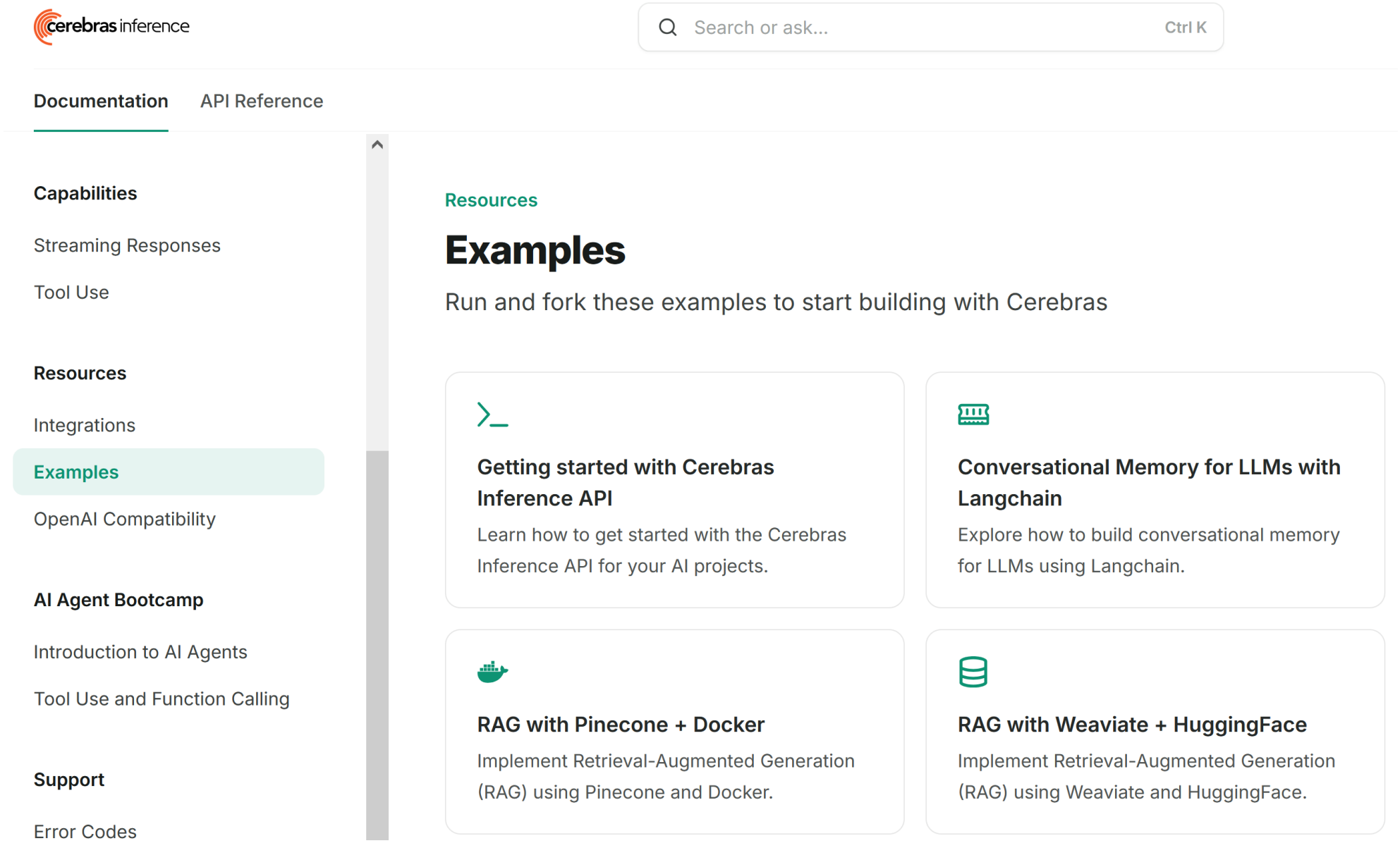
LARGEST GPU

826mm² Silicon

80 Billion transistors

https://cerebras.ai/blog/cerebras-architecture-deep-dive-first-look-inside-the-hw/sw-co-design-for-deep-learning

Reading and Learning Loop!



Cerebras Wafer-Scale Engine (WSE)

- **Cerebras WSE is an entire silicon wafer used as a single massive chip**
- The large surface area enables an extensive number of processing cores, memory, and interconnections in a single chip, reducing the need for multiple, smaller chips communicating across a board.
- The WSE includes **over 40 GB of on-chip SRAM memory** spread across the wafer. This proximity to the processing cores provides **high bandwidth** and **low latency** memory access. This eliminates the need for external memory modules, reducing delays and energy consumption associated with off-chip data transfer, which is a common bottleneck in traditional AI accelerators.
- Cerebras uses a **high-speed, low-latency interconnect fabric** that connects all cores on the chip, enabling efficient data flow across the entire wafer.
- **Optimization for AI Workloads**:
  - The Cerebras WSE is specifically optimized for **deep learning and AI workloads**, handling large matrix multiplications and tensor operations at a scale that traditional chips struggle with.
  - The architecture allows for **model parallelism**, where different parts of a neural network can run concurrently, making it highly efficient for training and inference tasks on large models.

Google Tensor Processing Unit (TPU)

- A custom-designed accelerator developed by Google specifically for AI and machine learning tasks.

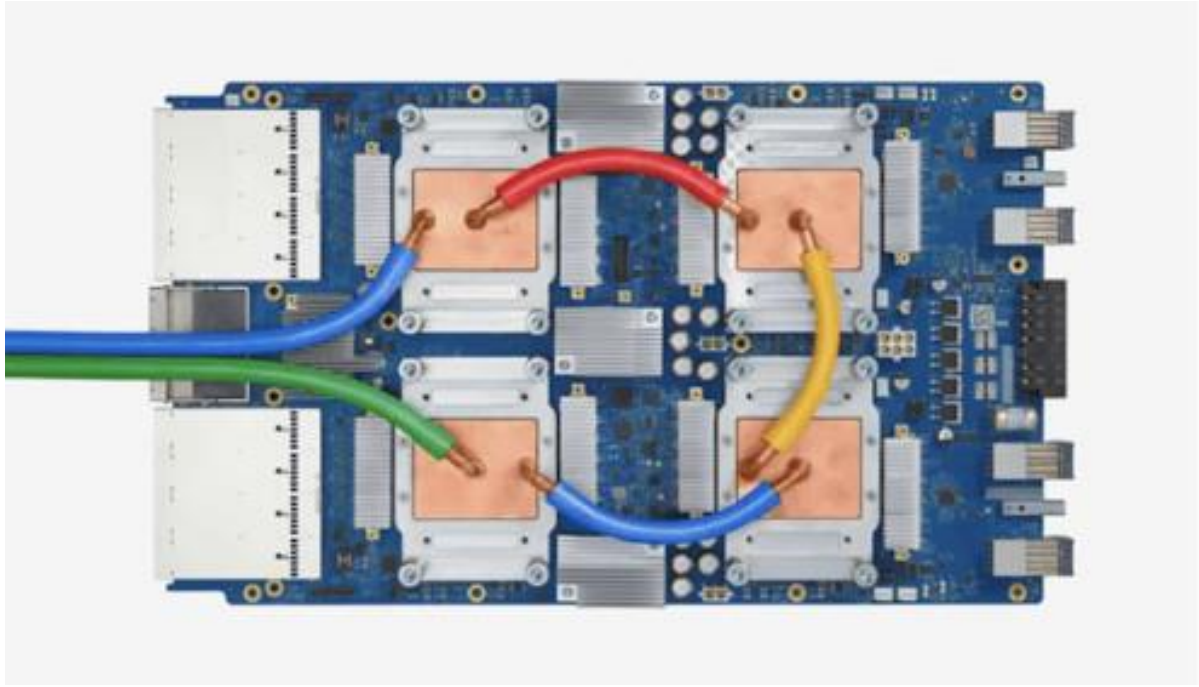
- The core of the TPU architecture is: Matrix Multiply Unit (**MXU**)
  - Inside MXUs, there are **128x128 systolic arrays**
    - thousands of multiplications and additions happen in parallel, significantly speeding up tensor operations.

- Unlike CPUs, and GPUs, TPUs designed specially for AI tasks
  - They avoid unnecessary caches and complex control units

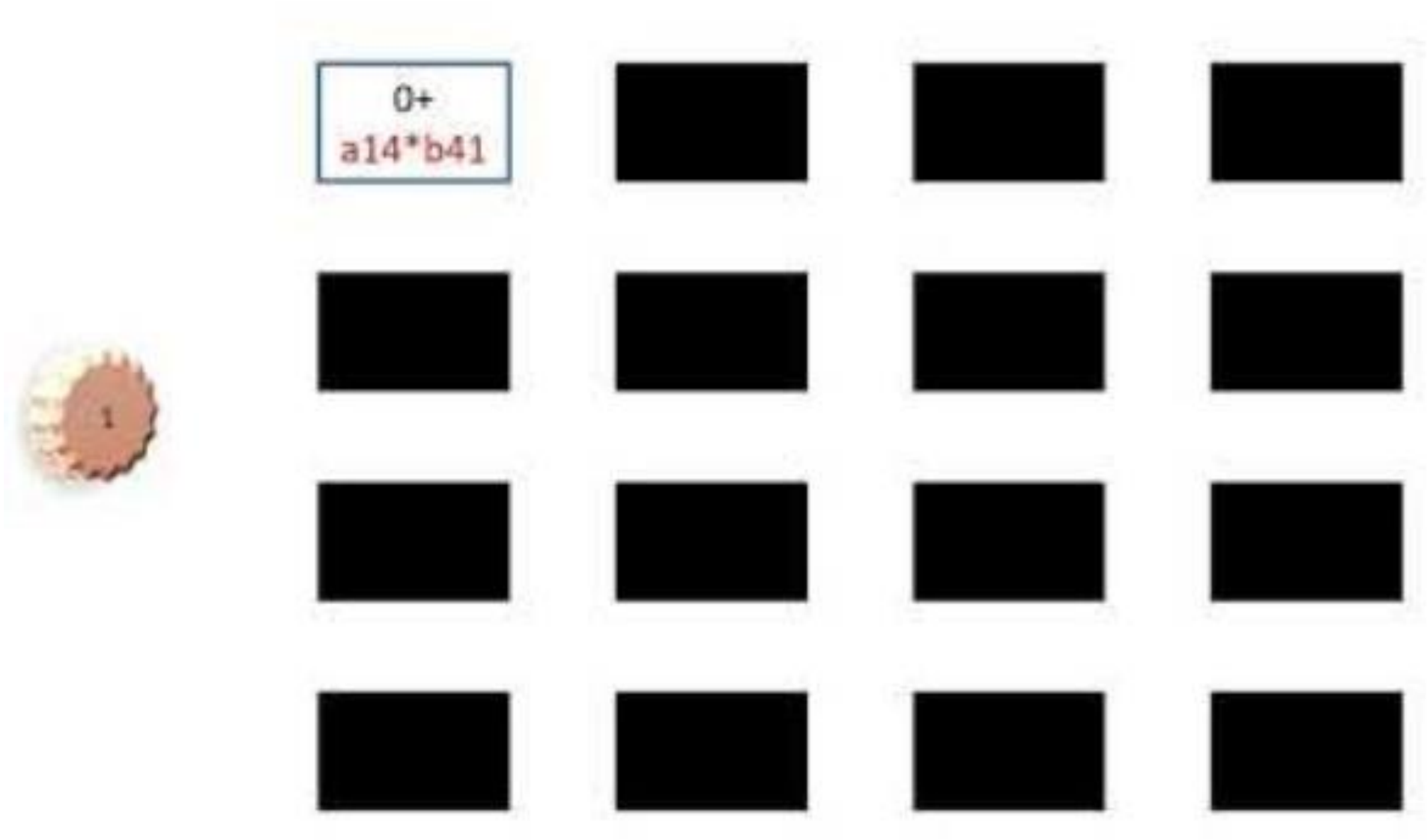
- High Bandwidth Memory (HBM)

- Programming with TensorFlow and JAX library (Google's **XLA compiler** is used to further optimize TensorFlow code for TPU hardware, by fusing operations and minimizing memory usage, making TPU execution more efficient.)

Scalability with TPU pods!



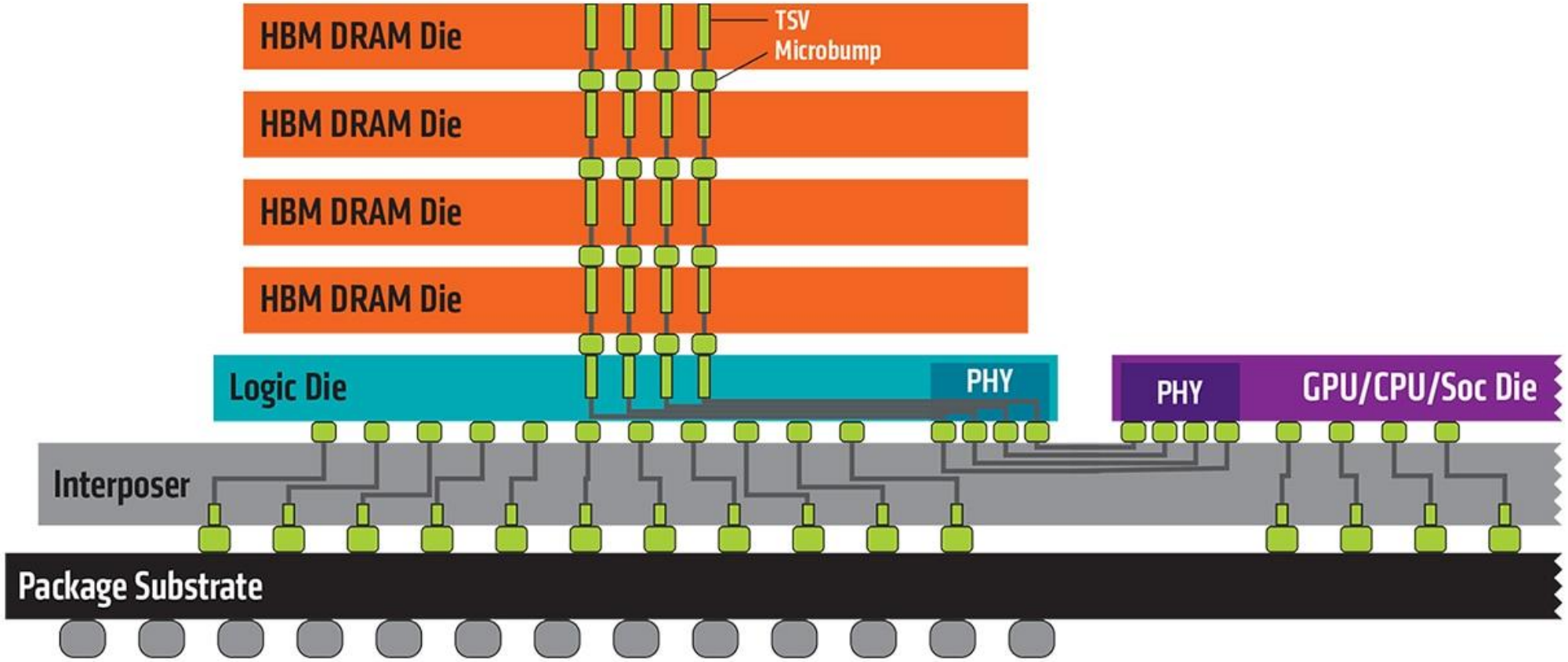
How Systolic Arrays work!



High Bandwidth Memory (HBM)

- **High Bandwidth Memory (HBM)** is an advanced type of memory technology designed to provide faster data transfer rates and higher memory bandwidth than traditional memory types, such as DDR (Double Data Rate) memory. HBM achieves this efficiency through a unique design where memory chips are **stacked vertically** and connected using a technology called **through-silicon vias (TSVs)**. This stacking allows multiple layers of memory to be connected closely, enabling data to travel shorter distances compared to traditional memory layouts.
- HBM's vertical stacking also places it much closer to the processor, typically on the same package or die. This proximity reduces **latency**—the delay before data transfer begins—and significantly **increases bandwidth**, or the rate at which data can move between memory and the processor. HBM can handle **massive amounts of data** simultaneously, making it especially beneficial for **high-performance computing tasks** like AI, deep learning, and graphics rendering, where large datasets need to be processed quickly.
- The efficiency of HBM comes not only from its high bandwidth but also from its **power efficiency**. By keeping memory closer to the processor and enabling faster data access, HBM reduces the energy needed to transfer data, resulting in lower overall power consumption. This combination of high bandwidth, low latency, and energy efficiency makes HBM an ideal choice for modern accelerators, such as GPUs and TPUs, where fast, efficient memory access is crucial for performance.

HBM


HBM DRAM Die
HBM DRAM Die
HBM DRAM Die
HBM DRAM Die
TSV
Microbump
Logic Die
PHY
PHY
GPU/CPU/Soc Die
Interposer
Package Substrate

Content

- GPUs
  - Story of NVIDIA GPUs
- FPGA
- Accelerator
- **Tradeoff of processors**

Tradeoff!

Processor	Programmability	Goal	Flexibility (Different Programs)	Promised Performance
CPU	Easy (use any programming language you know)	Latency Oriented	Super High	Fair
GPU	Medium (learning CUDA!)	Throughput Oriented	High	Medium
FPGA	Hard (Learn Verilog + Digital electronics basics)	Both	Low	High
Accelerator	Very Hard (read docs and learn concepts of the filed like AI)	Both	Super Low	Super High

Explore and Discover!

CUDA C++ Programming Guide (nvidia.com)

CUDA Toolkit - Free Tools and Training | NVIDIA Developer

NVIDIA Blog

Deep Learning Institute and Training Solutions | NVIDIA

DGX Platform | NVIDIA

Intel Field Programmable Gate Arrays (FPGA) Technical Training | Intel

Product - Chip - Cerebras

Products | Coral

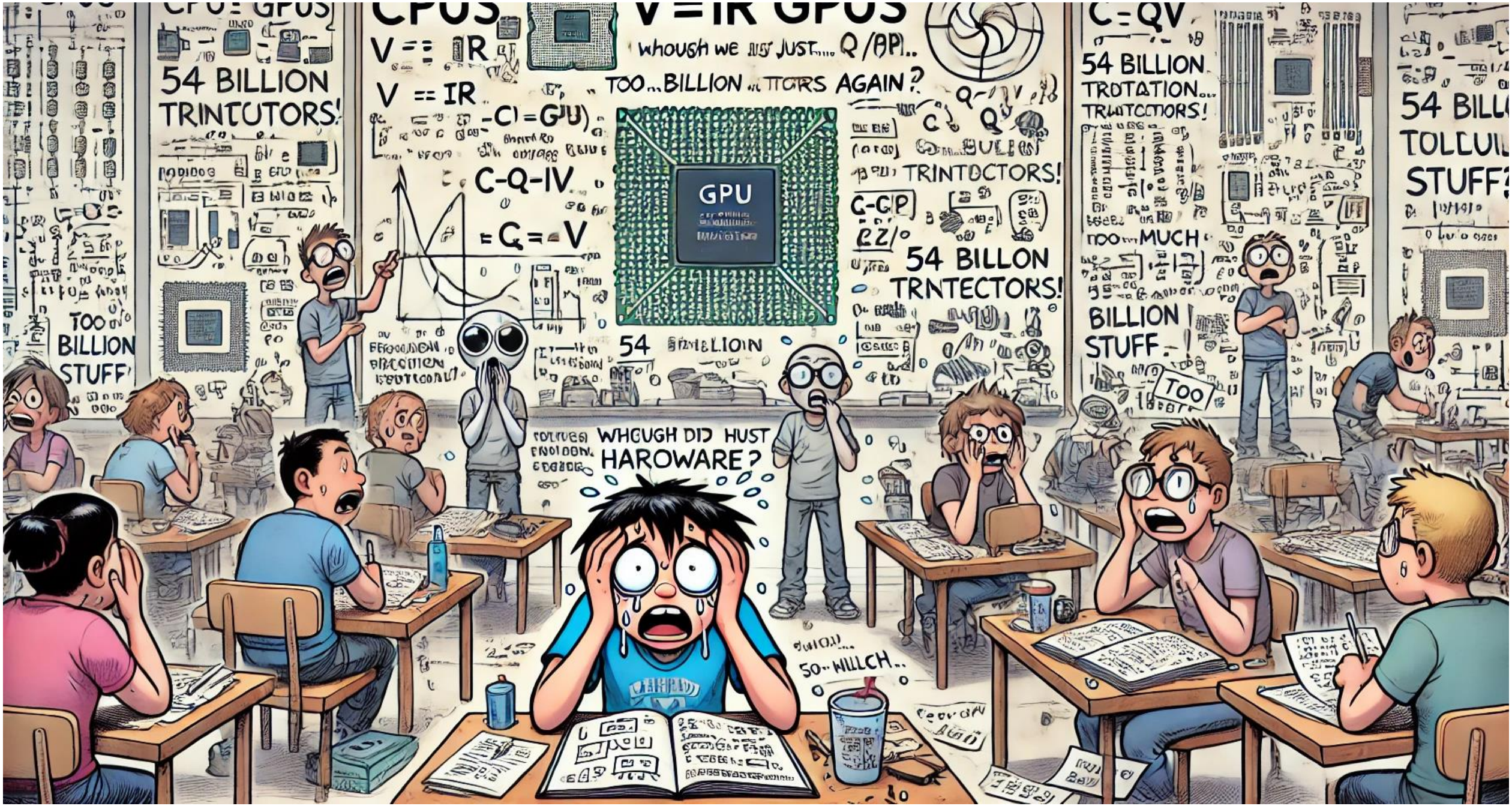
Altera® FPGAs and Programmable Devices (intel.com)

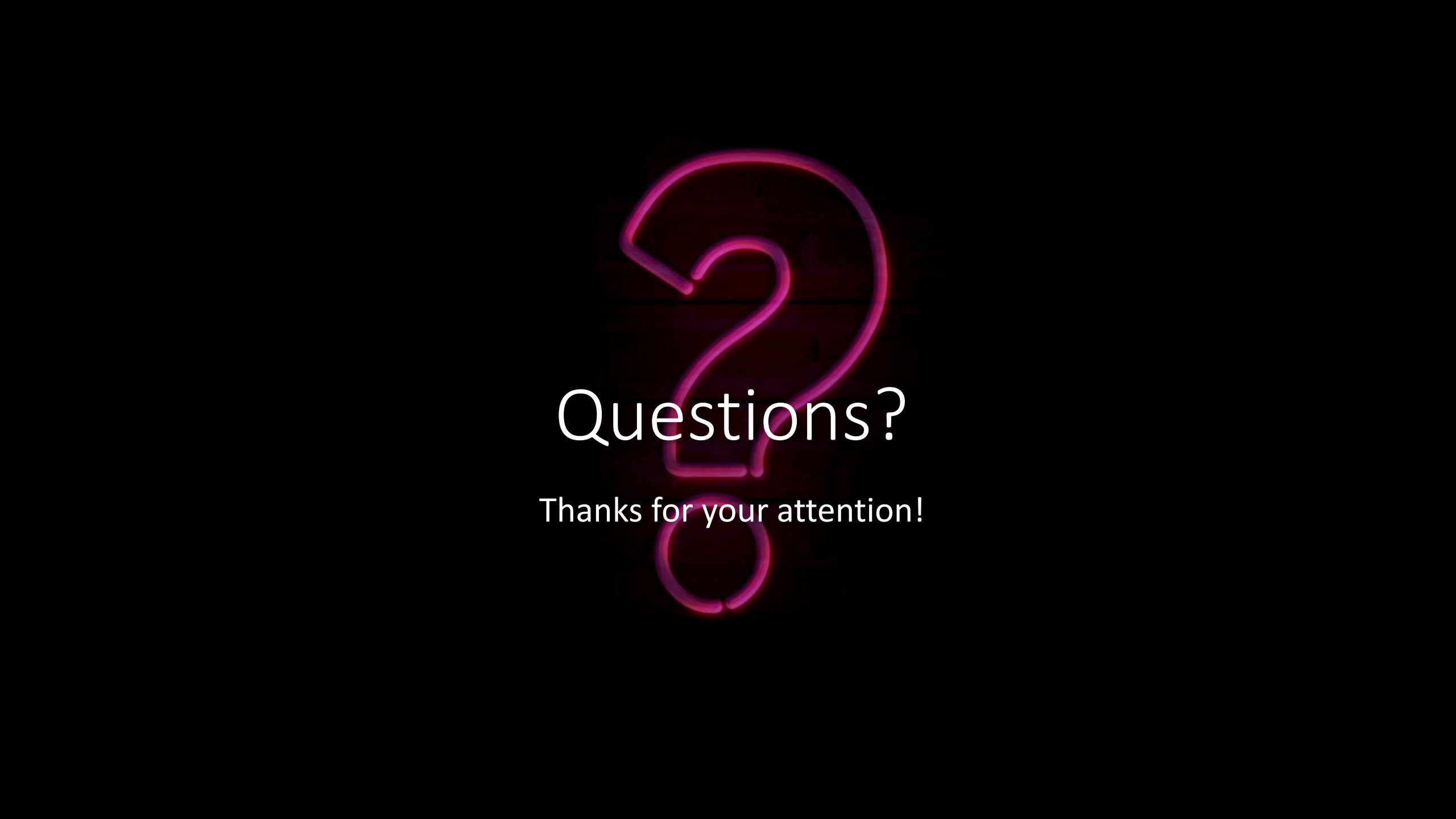
Reimagining the Data Center (amd.com)

FPGAs & 3D ICs (xilinx.com)

Intel® Core™ Processors - View Latest Generation Core Processors

AMD Processors | AMD


54 BILLION
TRANSISTORS!
V = IR
C=QV
GPU
TOO BILLION STUFF
54 BILLION
TRANSISTORS!
TOO MUCH
BILLION STUFF
WHOUGH DID HUST HAROWARE?
SO... MUCH...



Questions?

Thanks for your attention!

Backup in case anyone is curious!

Modern CPUs

- They fetch and execute more than one instruction (a windows of instruction)
  - Higher throughput
- Advanced Hardware Execution Mechanisms to execute faster
- Employ Cache Hierarchy to fill the Memory-Processor performance gap
  - Temporal/ Spatial Locality
- They have several cores (parallel computing)

Hennessy and Patterson

Tomasulo

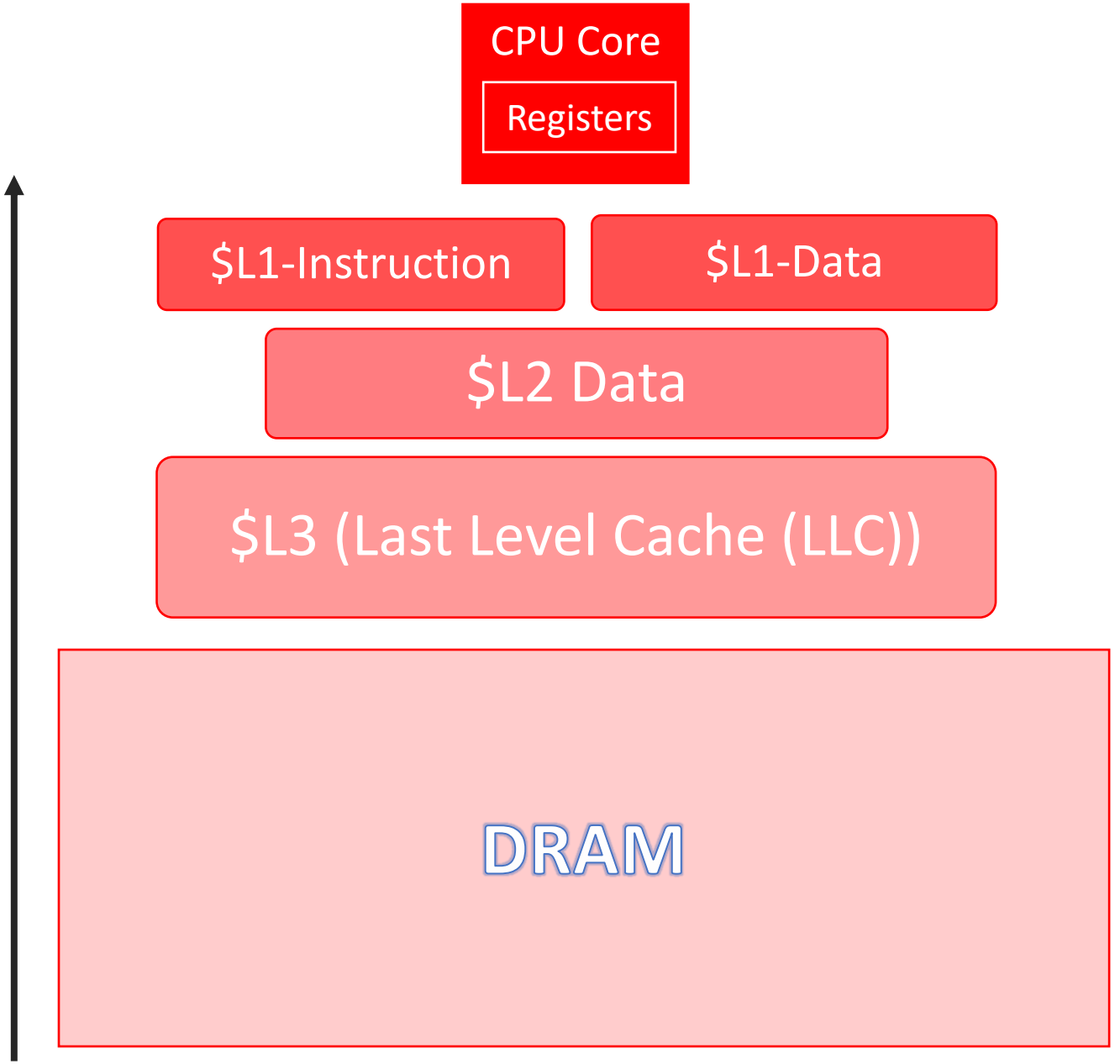
Yale Patt

Cache Hierarchy

Less Access latency
More Data Locality

Less Storage Capacity

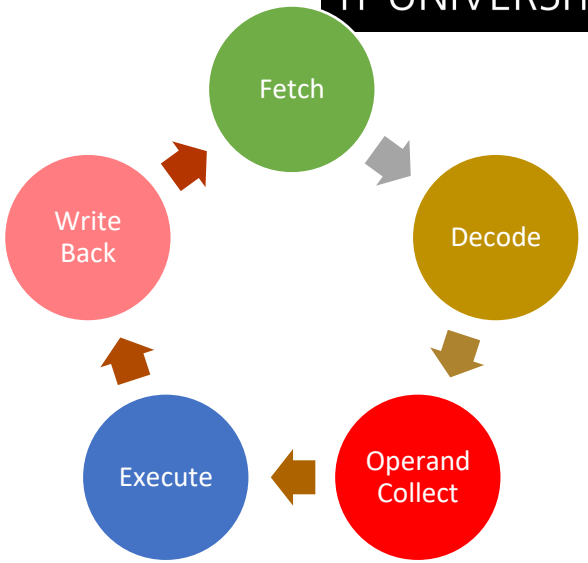
More Expensive per bit

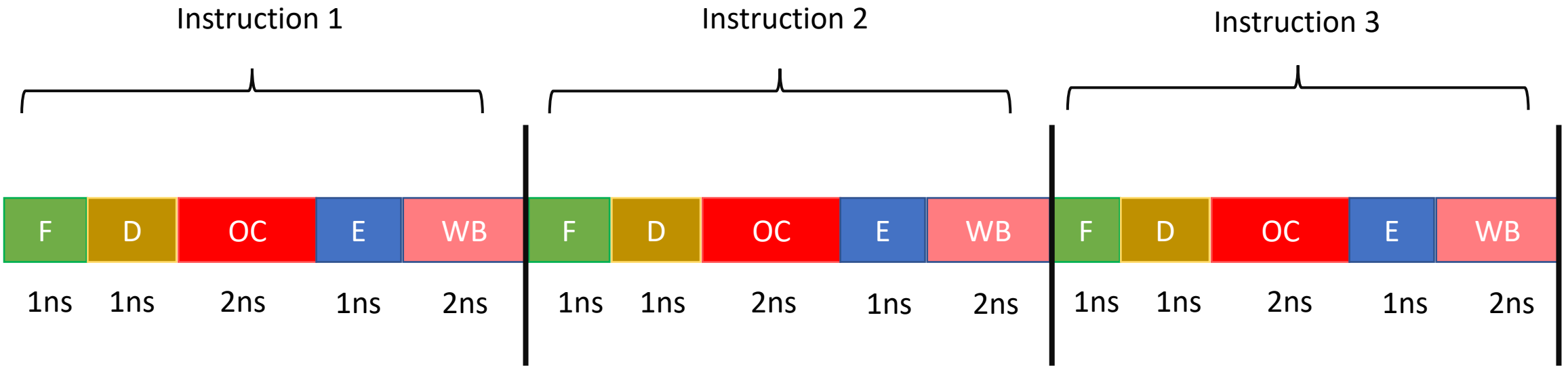


Pipelining

- Basic Processor

Inst1	O11, O12
Inst2	O21, O22
Inst3	O31, O32

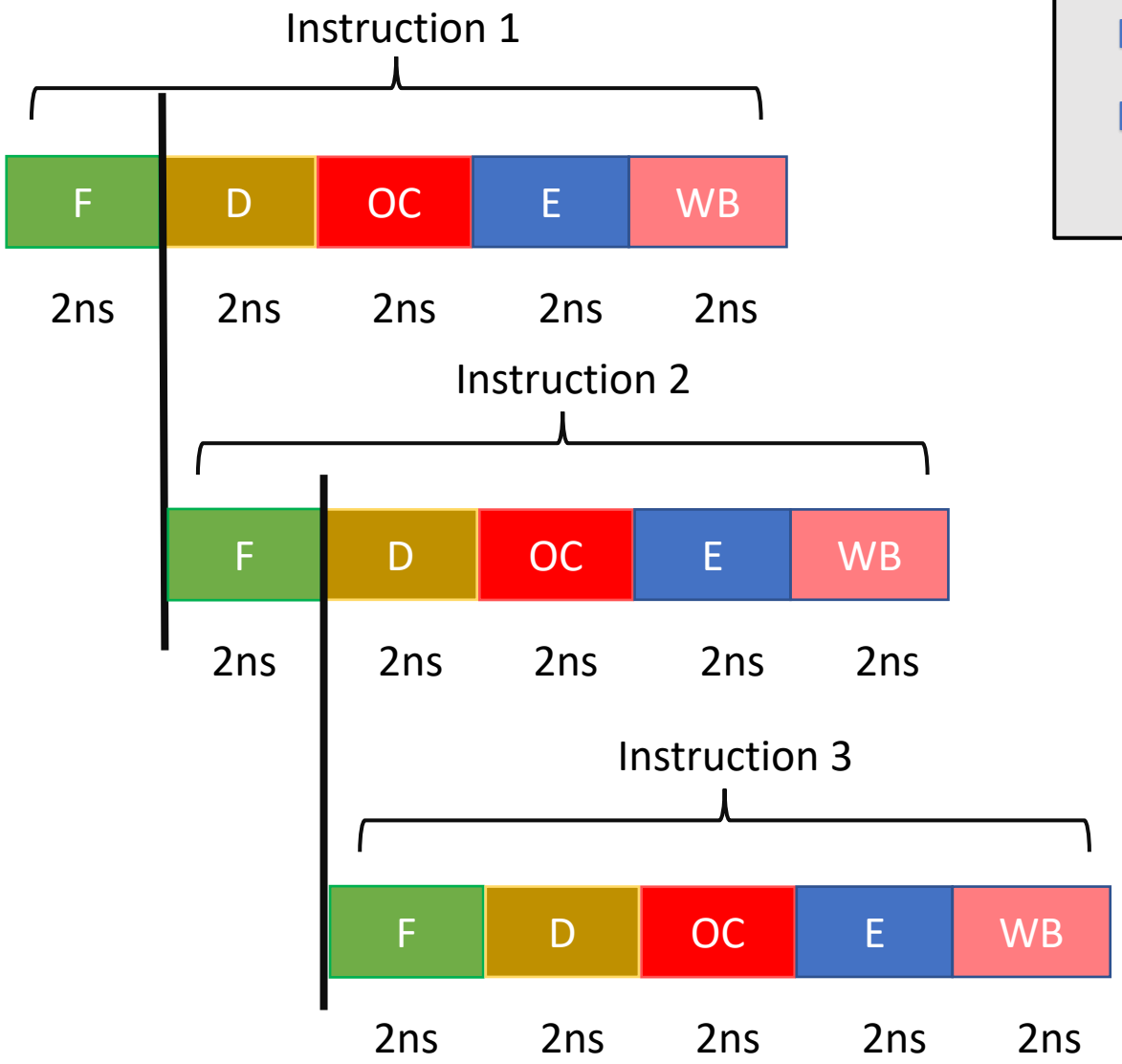




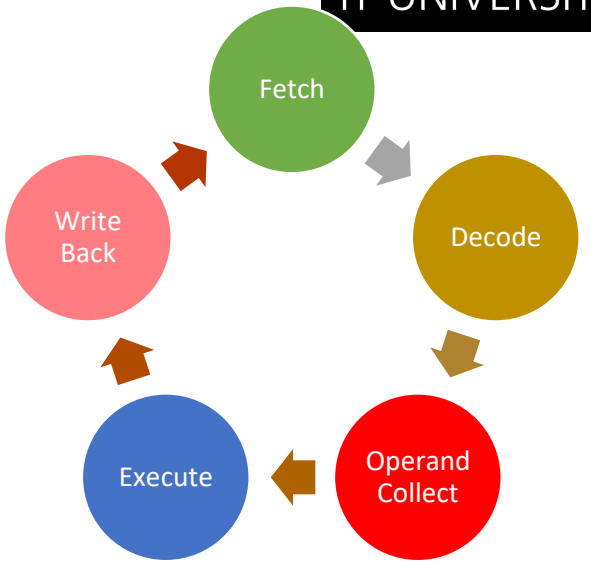
Overall Time of Executing three Instruction: 3 * (1ns + 1ns + 2ns + 1ns + 2ns) = 3 * 7ns = 21ns

Pipelining

- Pipelined Basic Processor



Inst1	O11, O12
Inst2	O21, O22
Inst3	O31, O32



Overall Time of Executing three Instruction:

= (2ns + 2ns + 2ns + 2ns + 2ns) + 2ns + 2ns
= 10ns + 2ns + 2ns = 14ns

Implicit Parallelism
Instruction-Level Parallelism (ILP)