AI Accelerated Devices

9th May 2023 -Day 2, Session 2A

sambath.narayanan@gmail.com

THIAGARAJAR COLLEGE OF ENGINEERING, MADURAI
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Online Faculty Development Program
"Artificial Intelligence for Edge Computing", 8th-12th May, 2023.

Date	10.00am-10.30 am	10.30 am – 11.30 am*		11.45am -1.15 pm *		2.15 pm-4.45 pm *
08.05.2023 Monday	Inaugural Function	**Session 1A** AI for UAVs - Opportunities and Challenges *Ramesh Naidu. L CDAC*	BREAK	**Session 1B** AI for UAVs - Opportunities and Challenges *Ramesh Naidu. L CDAC*	BREAK	**Session 1C** Internet of Things Edge Analytics *Industry Expert*
09.05.2023 Tuesday	**Session 2A** AI Accelerated Devices *Sambath Narayanan P Dataever Consulting*			**Session 2B** Computer Vision *Sambath Narayanan P Dataever Consulting*		**Session 2C** ML models for Edge Computing: Architecture and Use-cases *Dr.S.Karthik NIT Andhrapradesh*
10.05.2023 Wednesday	**Session 3A** Edge Computing-Thing Speak *Akhil Gopinath Mathworks*			**Session 3B** Edge Computing-MatLab *Akhil Gopinath Mathworks*		**Session 3C** Digital Twin *Akhil Gopinath Mathworks*
11.05.2023 Thursday	**Session 4A** Industrial Internet of Things (IIoT) *Supriya Jena & Sunil G Kumar IBM*			**Session 4B** Analytics in IIoT *Rajesh Sharma IBM*		**Session 4C** IIoT Use-cases-Demonstration *Supriya Jena & Sunil G Kumar IBM*
12.05.2023 Friday	**Session 5A** Edge Intelligence *Sundaravelu (Sundar) Shanmugam, Kyndryl*			**Session 5B** Data Analytics at Edge *Sundaravelu (Sundar) Shanmugam Kyndryl*		**Session 5C** Research Opportunities in Edge Computing & Valedictory Function

(* Subject to change based on the Session engagement)

Acknowledgement

Acknowledgment

- In this presentation we have used many tools, libraries and images from NVIDIA. We acknowledge the same.
- In other places, if we use some information, we provide the URL for the same
- If we missed out something, we will correct the same

Overall Session Agenda

AI Accelerated Devices

- Why AI Acceleration?
- What is AI Acceleration?
- Acceleration Devices of importance
- AI Acceleration Strategy

AI Accelerartion Demos

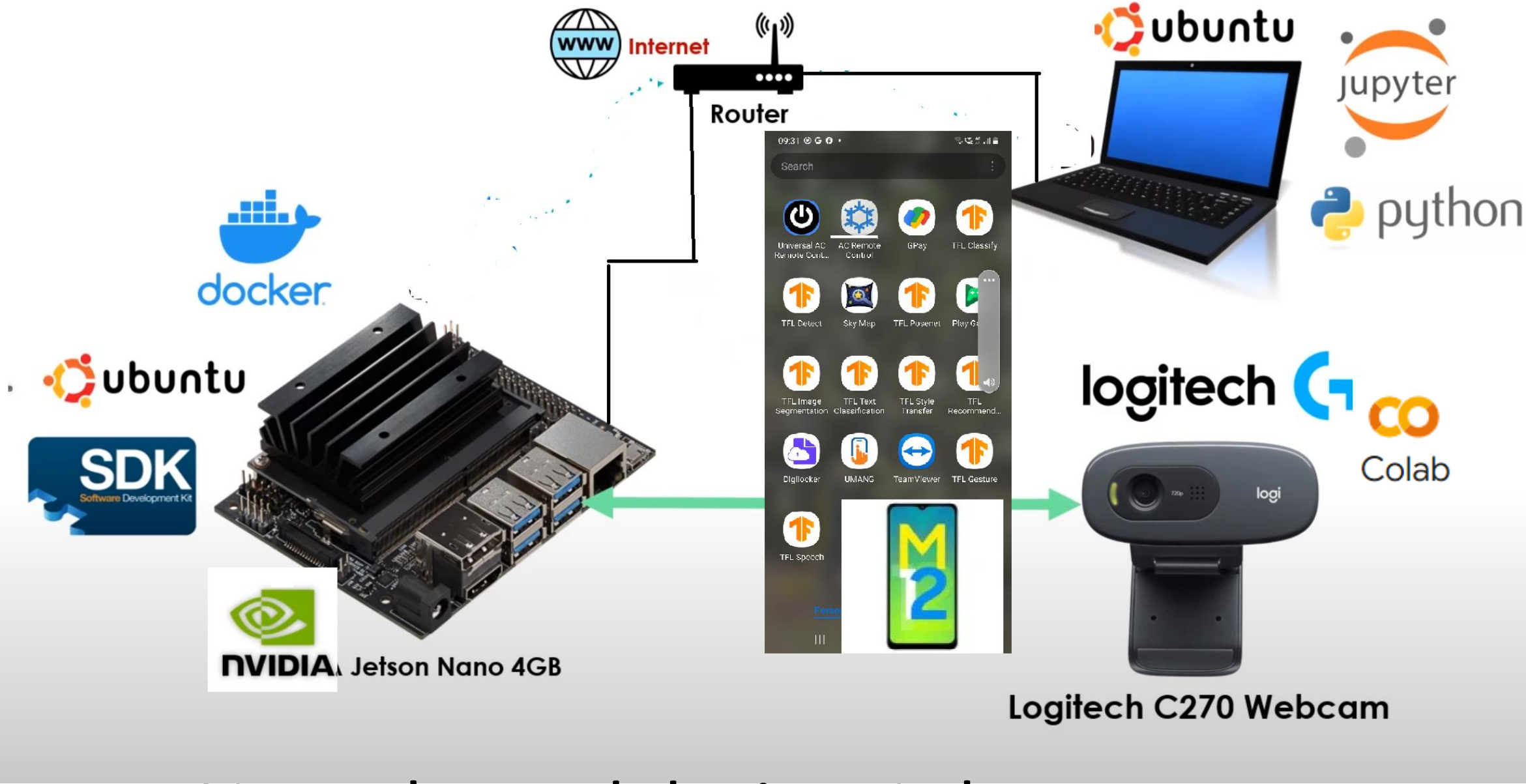
- Basic Demos
- Industrial – Real Time CFD
- Kaggle CPU – GPU Benchmark
- Others

Training Acceleration

- GPU /FPGA Acceleration: Using specialized hardware like distributed GPUs or TPUs can speed up training by leveraging their parallel processing capabilities.
- Model Pruning: reduce the model's size and complexity, without compromising model accuracy
- Optimized Libraries
- Optimized Data Types
- SIMD using Tensor cores
- Pre-Trained models: utilizing pretrained models and performing transfer learning can save significant training time

AI Inference Acceleration

- GPU /FPGA Acceleration: Using specialized hardware like distributed GPUs or TPUs can speed up training by leveraging their parallel processing capabilities.
- Model Compression: Methods like model distillation or network pruning can reduce the model's size, leading to faster inference by reducing the number of operations.
- Optimized Runtime
- Optimized Data Types
- Pre-Trained models: utilizing pretrained models and performing transfer learning can save significant training time
- GPU Containerized Environment:



AI accelerated devices Lab setup

Challenges

Why AI acceleration Required – Large size

- Increasing Number of Parameters:
  - GPT-3 has 175 billion parameters
- Increasing Data Size:
  - The IMDB-Wiki Computer Vision dataset, which contains more than 500,000 images of human faces
- Large GPU systems:
  - 9 exaflops of computing power was unveiled by Google Cloud in May 2022 This cluster is powered by Cloud TPU v4 Pods

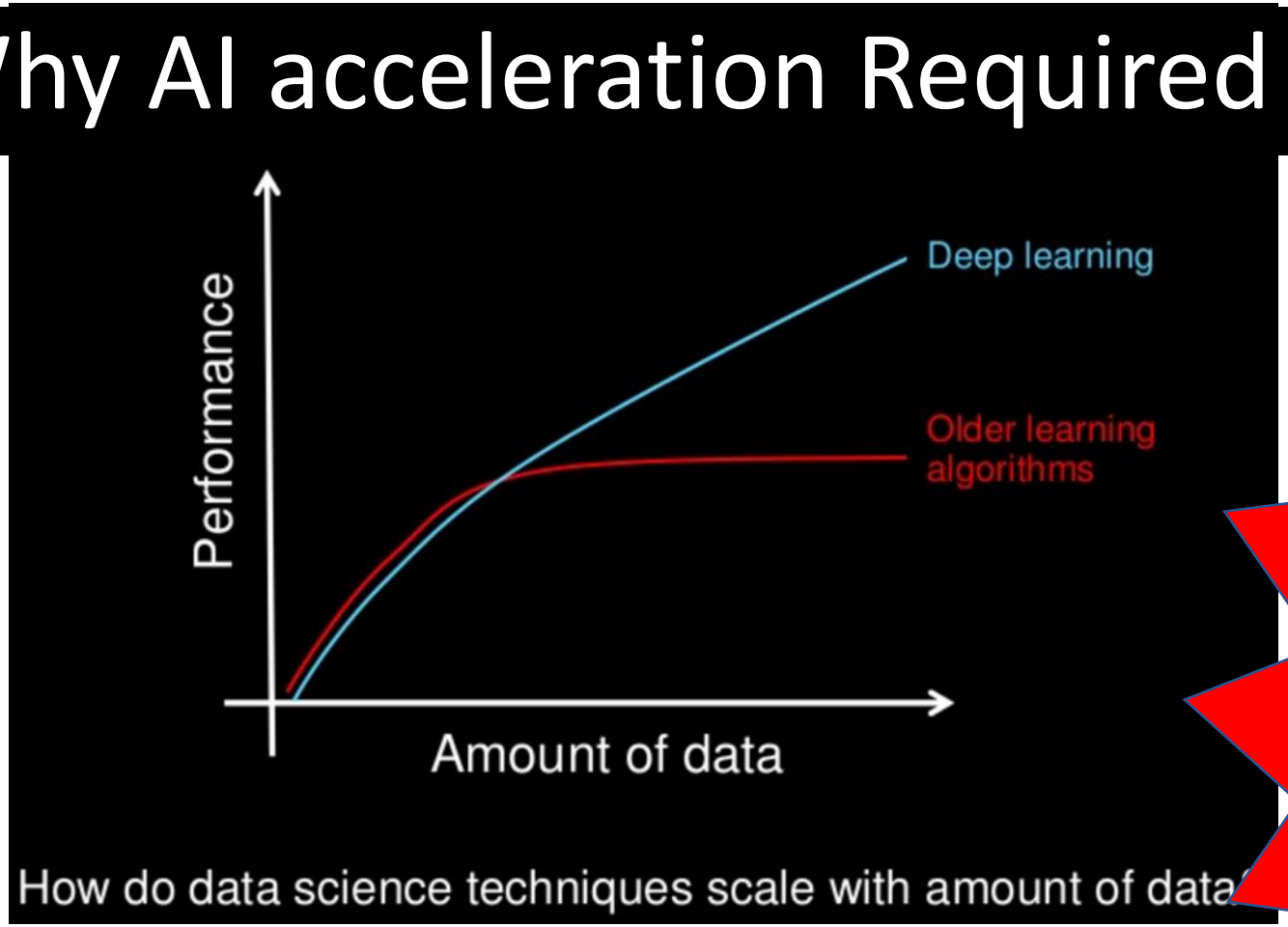
Why AI acceleration Required ?

- Aggressive Response time:
  - AI problems in autonomous vehicle applications is around 50 milliseconds
  - This response time is crucial for autonomous cars to make smooth and safe decisions in real-time, including interacting with the environment around the car, such as pedestrians and traffic lights. Additionally, autonomous vehicles equipped with radar or lidar sensors and a camera system have a reaction time of 0.5 seconds
  - However, human reaction times can be up to 0.5 seconds, which is slower than the response time required for autonomous vehicles

Why AI acceleration Required ?

Data sources

- IoT configurations, where sensors or devices collect raw data
- Open source Data repositories
- ImageNet
- CIFAR
- MNIST
- COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features



Source: Prof. Andrew Ng

But, AI modeling involving large datasets requires large compute power.

Can AI address Healthcare industry Challenges ?

- **Radiologist** is a medical practitioner trained in analyzing MRI or CT scan, X Ray and using the analysis to diagnose medical conditions

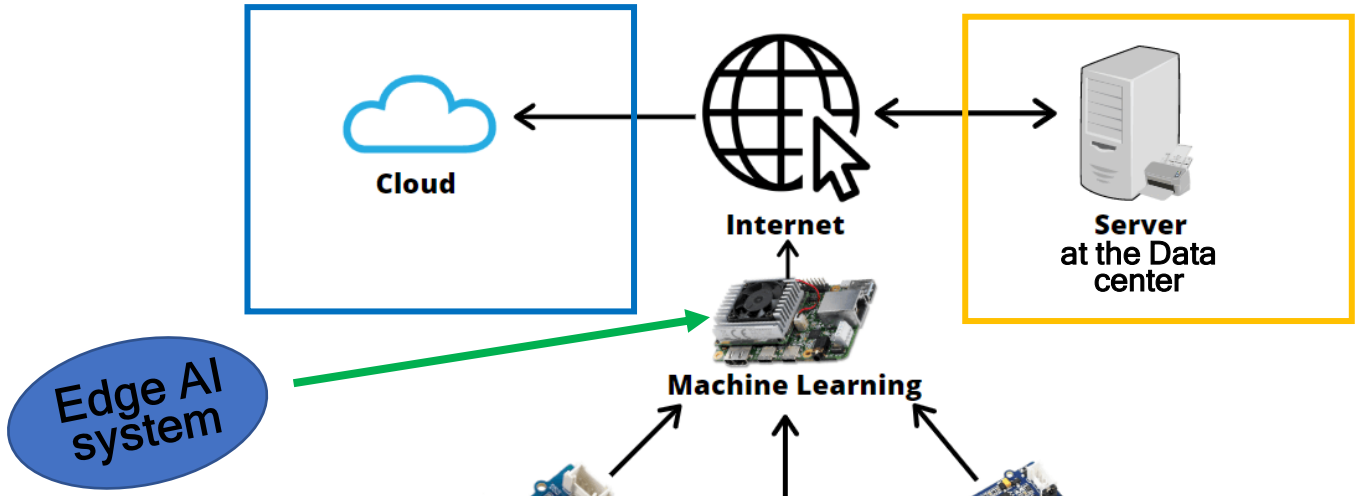
Number of scans professionals have to analyze = 1,000 a day

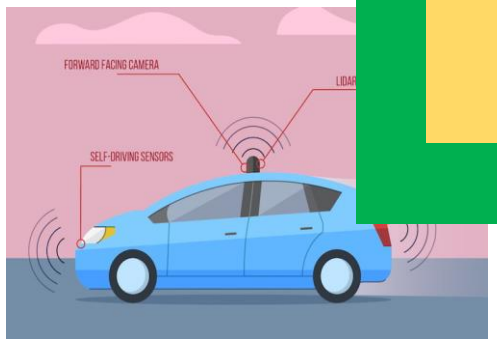
This can lead to lengthy delays between scan and treatment – **even when someone needs urgent care**.

Trends

Edge AI

- **Edge AI** where machine learning algorithms are being **run locally(Edge)** on an high performance hardware device or embedded systems (SoC, with GPU) as compared to on servers

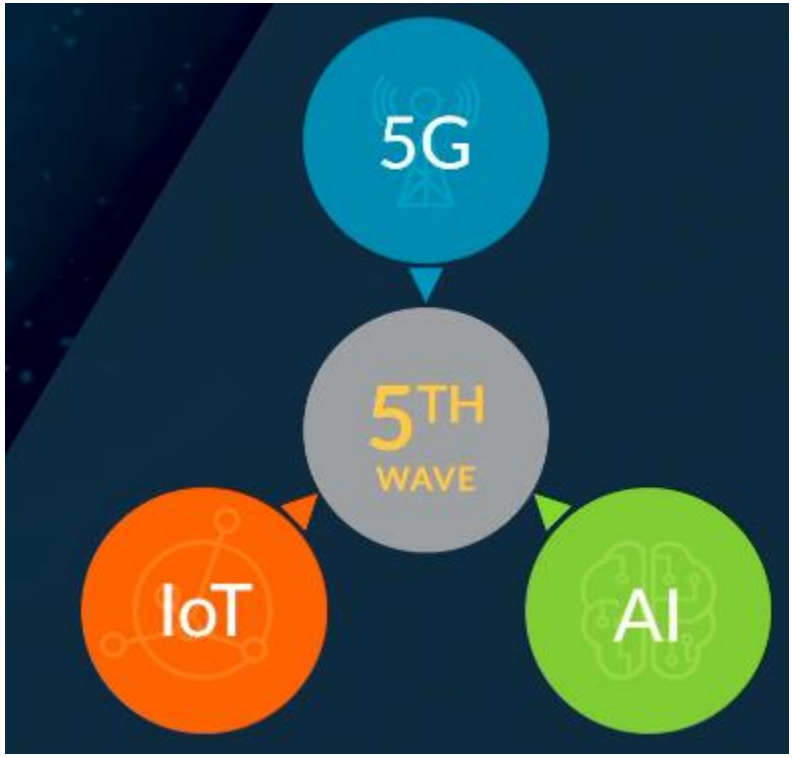




When AI / ML /DL processing is integrated with Edge computing, then it is known as **Edge AI**

Top technology trend

- By the end of 2024, 75% of companies will shift from piloting to operationalizing AI, driving a five times increase in streaming data and analytics infrastructures.

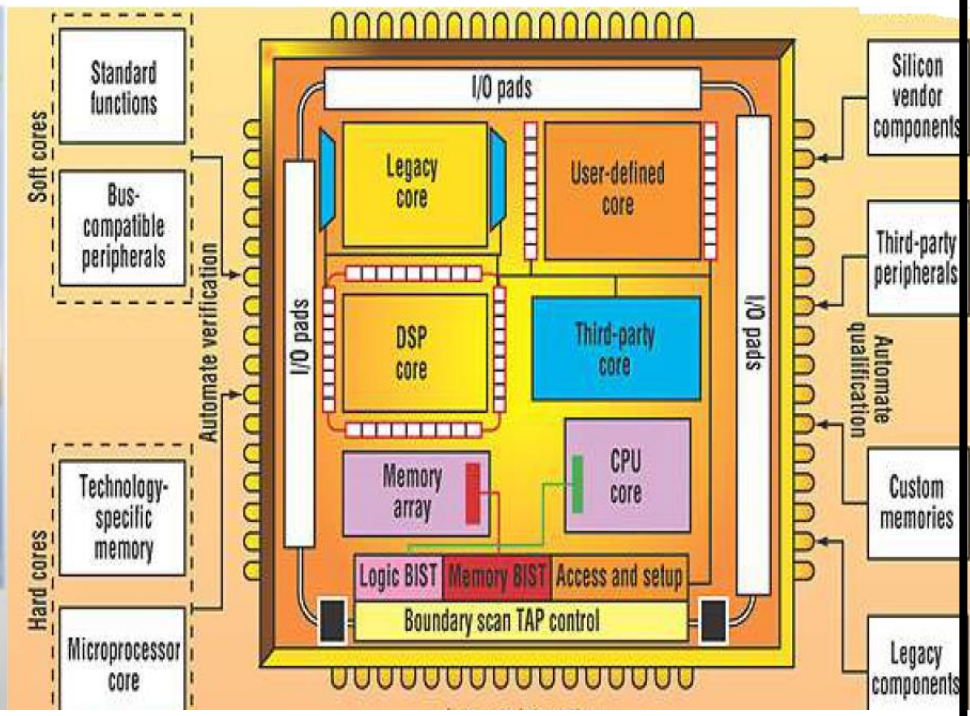


Technologies of relevance today's topic

IOT

1 trillion IoT devices by 2035

SoC



AI

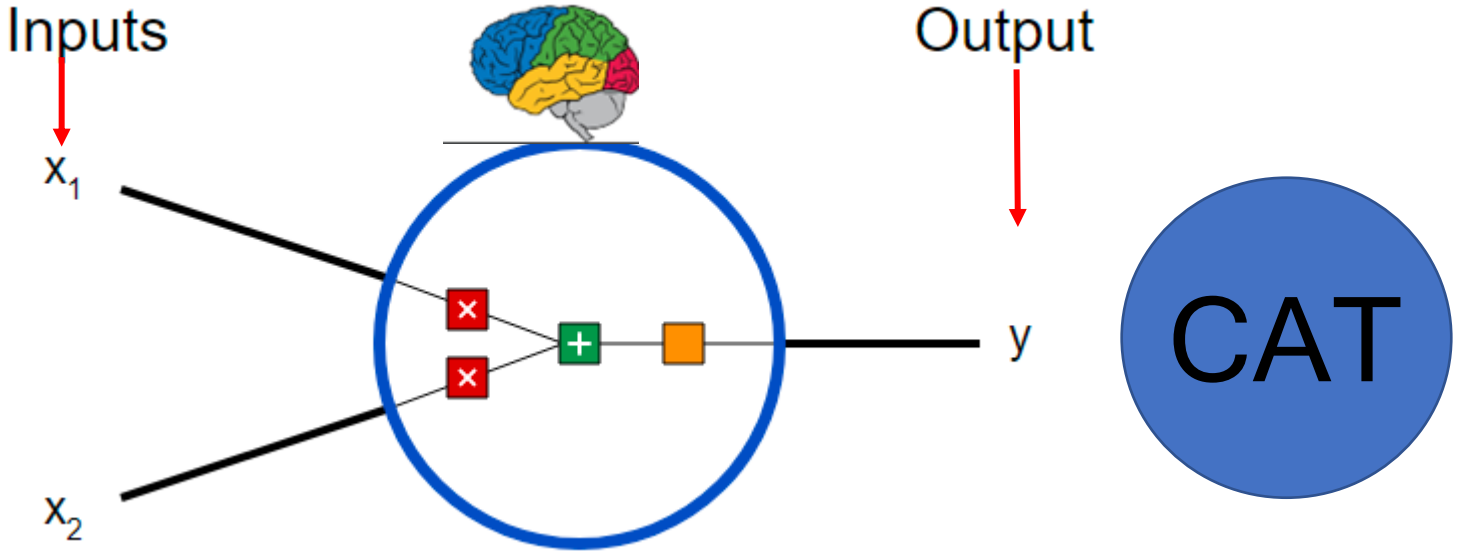
$17 trillion total economic impact by 2030

IOT – Internet of Things, SoC – System on a Chip, AI – Artificial Intelligence

AI Workload Characteristics

Building Blocks Neurons - combine Neurons into Neural Network(NN)

NN is Deep Learning



Input, processing and output together is also known as Perceptron

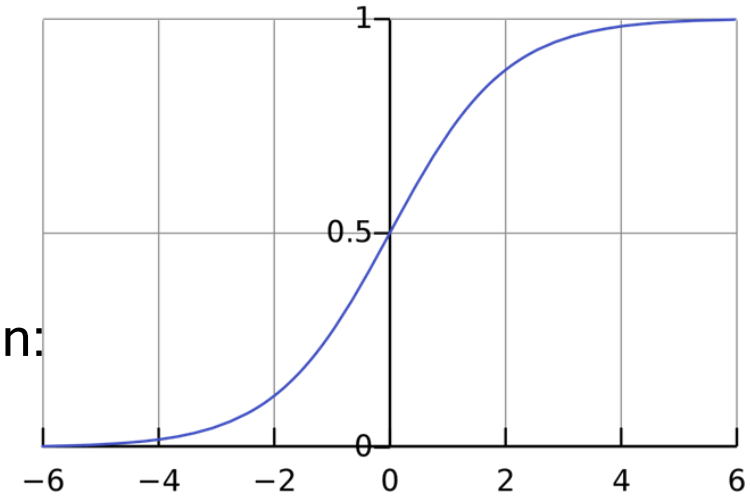
Example: Does the passengers Biometrics identity matches? which is a binary classification (Yes it matches 1 or No it doesn't match 0)

Typical AI workload pattern

Three things are happening here

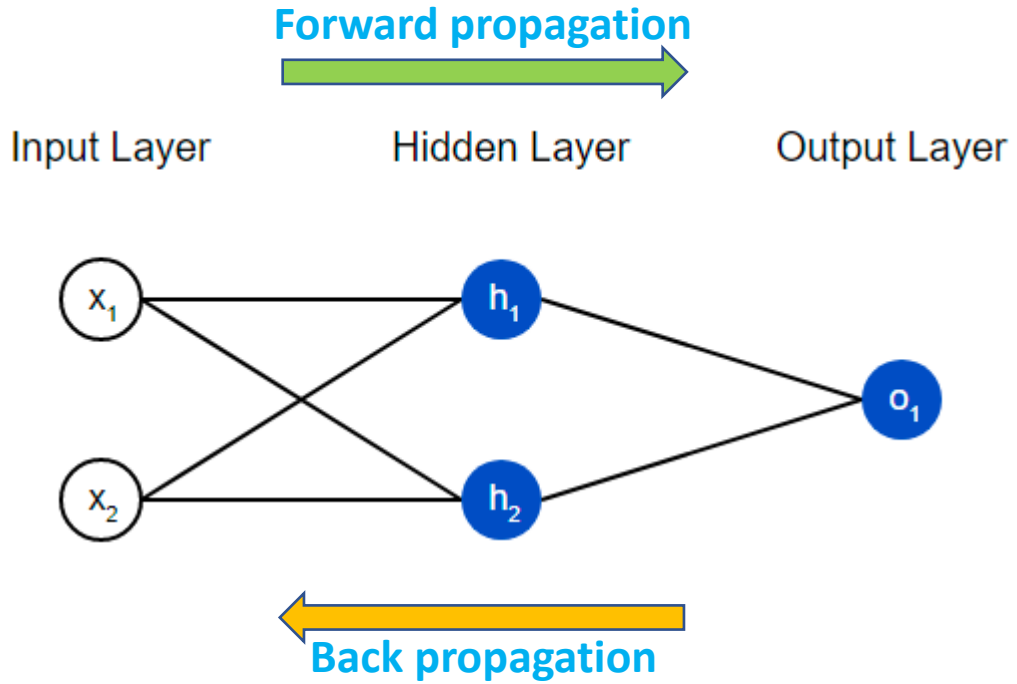
1. First, each input is multiplied by a weight:
   - x1 → x1 * **w1**
   - x2 → x2 * **w2**
2. all weighted inputs are added together with a bias **b**:
   - (x1 * **w1**) + (x2 * **w2**) + **b**
3. Finally, the sum is passed through an activation function:
   - y = **S** (x1 * w1 + x2 * w2 + b)

$$f(x) = \frac{1}{1 + e^{-x}}$$



- The activation function is used to turn an unbounded input into an output that has a nice, predictable form. A commonly used activation function is the sigmoid function:

Combining Neurons into Neural Network(NN)



This network has 2 inputs, a hidden layer with 2 neurons (*h*1 and *h*2), and an output layer with 1 neuron (*o*1). Notice that the inputs for *o*1 are the outputs from *h*1 and *h*2 - that's what makes this a network.

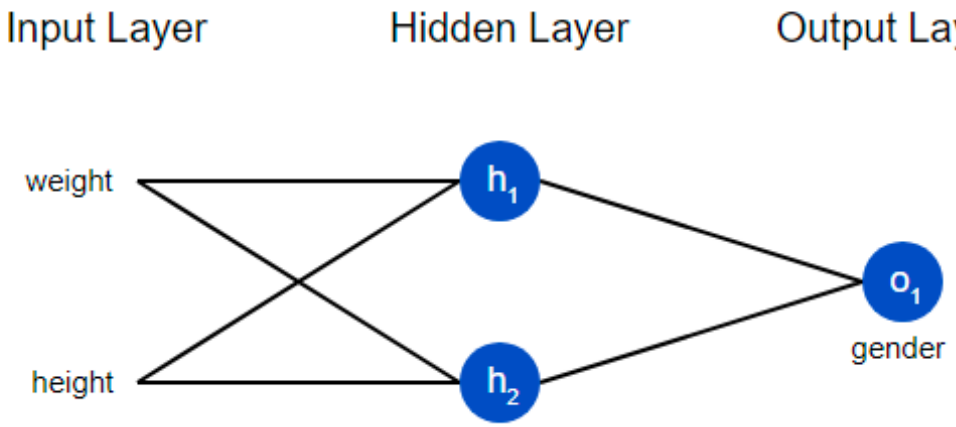
Training a Neural Network(NN)

Name	Weight (lb)	Height (in)	Gender
Alice	133	65	F
Bob	160	72	M
Charlie	152	70	M
Diana	120	60	F

Male = 0

Female = 1

Let's train our network to predict someone's gender given their weight and height:



Minimize Loss - Mean Squared Error (MSE)

- First quantify how “good” it’s doing so that it can try to do “better”.
- We’ll use the mean squared error (MSE) Loss:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2$$

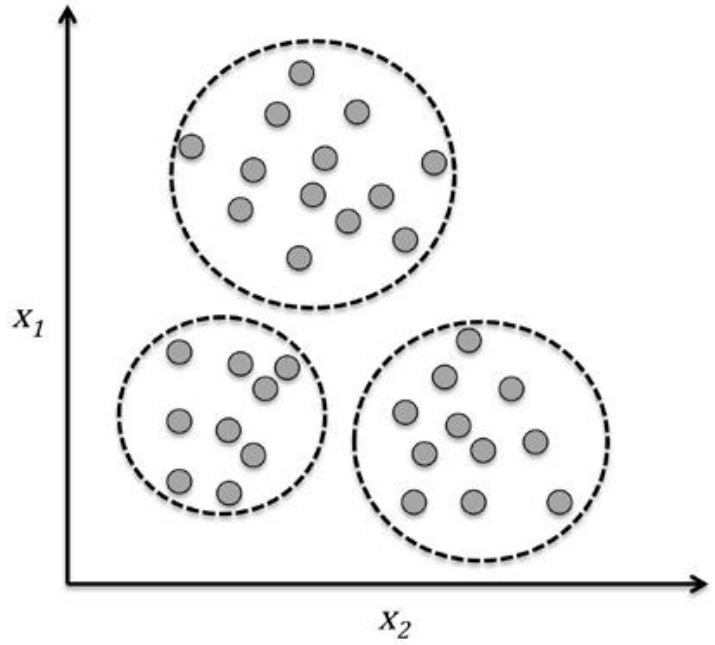
Where,

- n is the number of samples, which is 44 (Alice, Bob, Charlie, Diana).
- y represents the variable being predicted, which is Gender.
- y_{true} is the true value of the variable (the “correct answer”).
- For example, y_{true} for Alice would be 11 (Female).
- y_{pred} is the predicted value of the variable. It’s whatever our network outputs.
- $(y_{true} - y_{pred})^{**} 2$ is known as the squared error.
- Our loss function is simply taking the average over all squared errors .

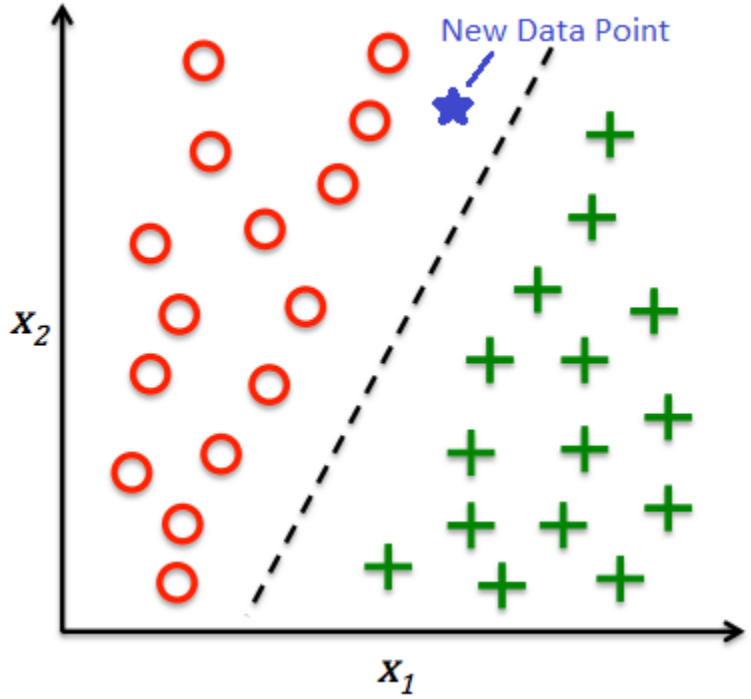
Better predictions = Lower loss.

Training a network = trying to minimize its loss.

Clustering



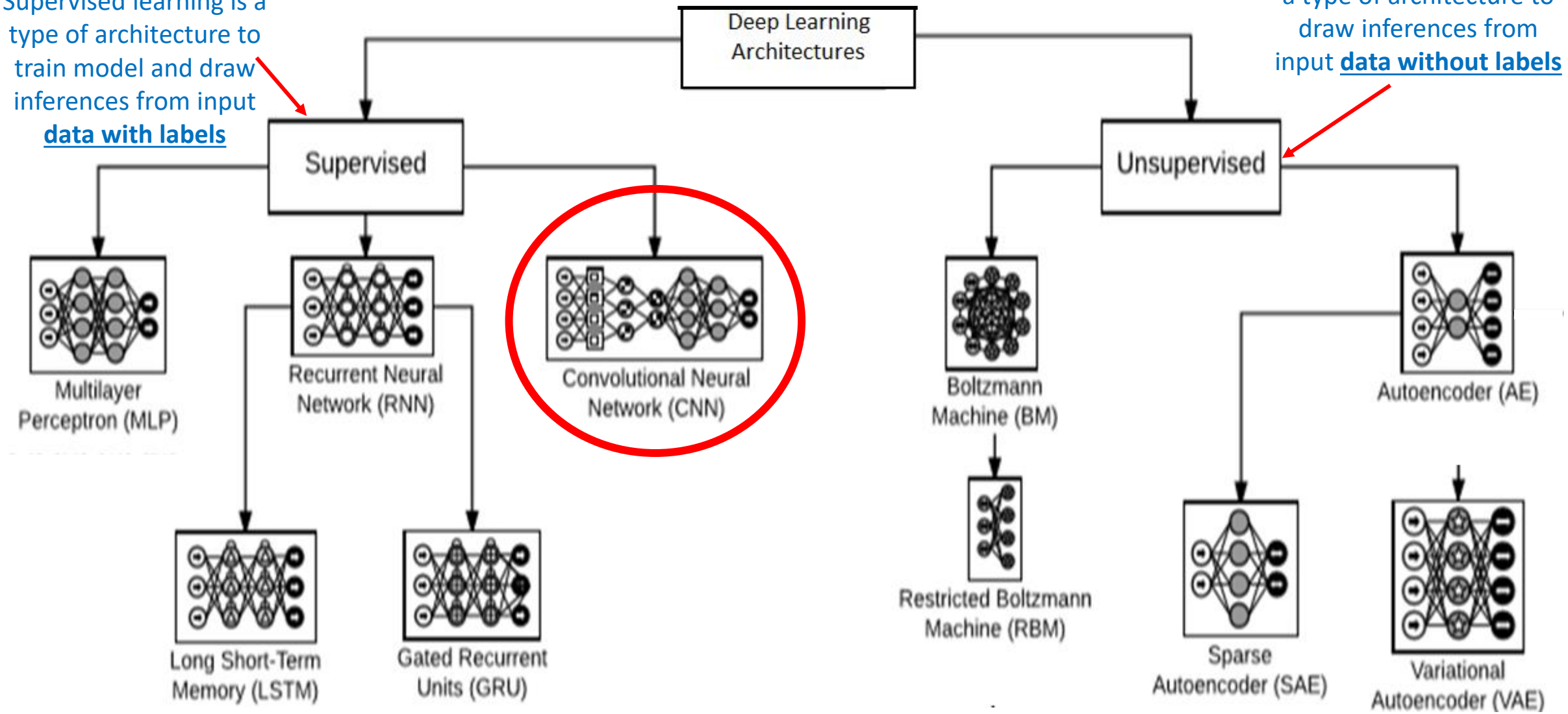
Classification



Deep Learning Architectures

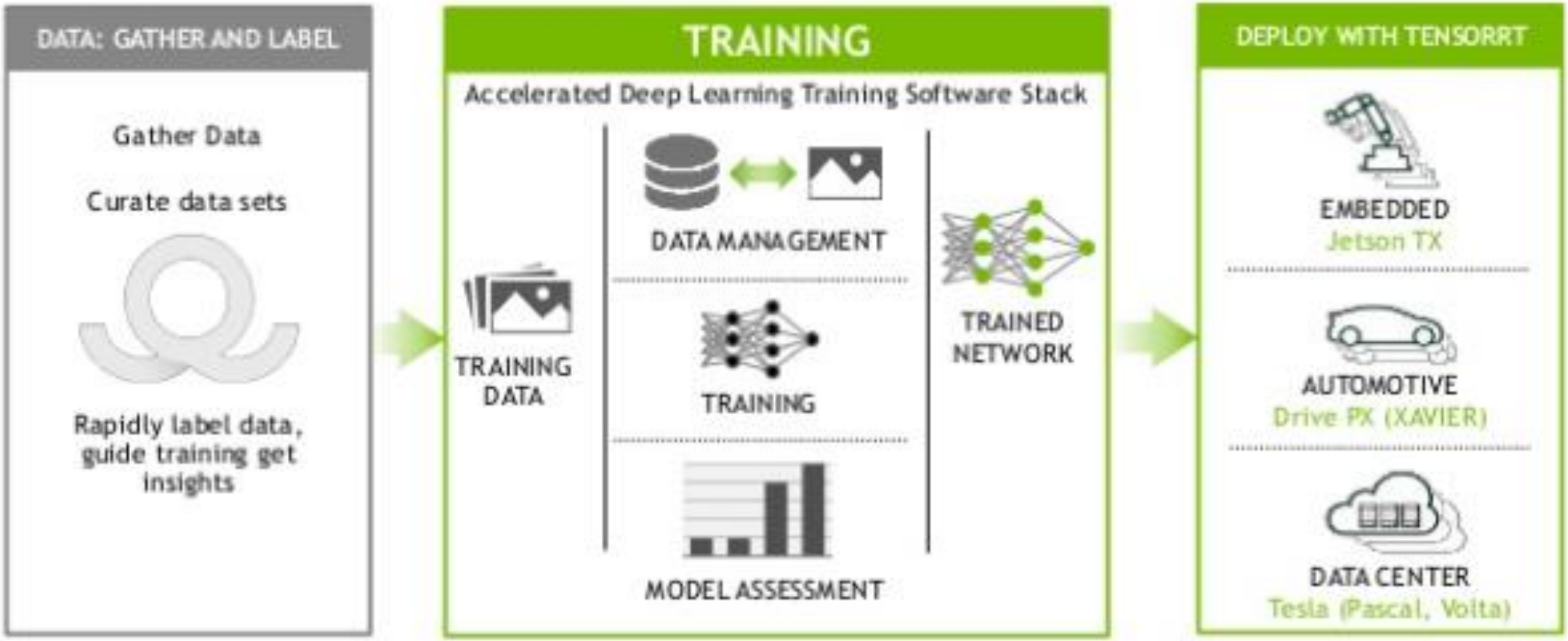
Supervised learning is a type of architecture to train model and draw inferences from input **data with labels**

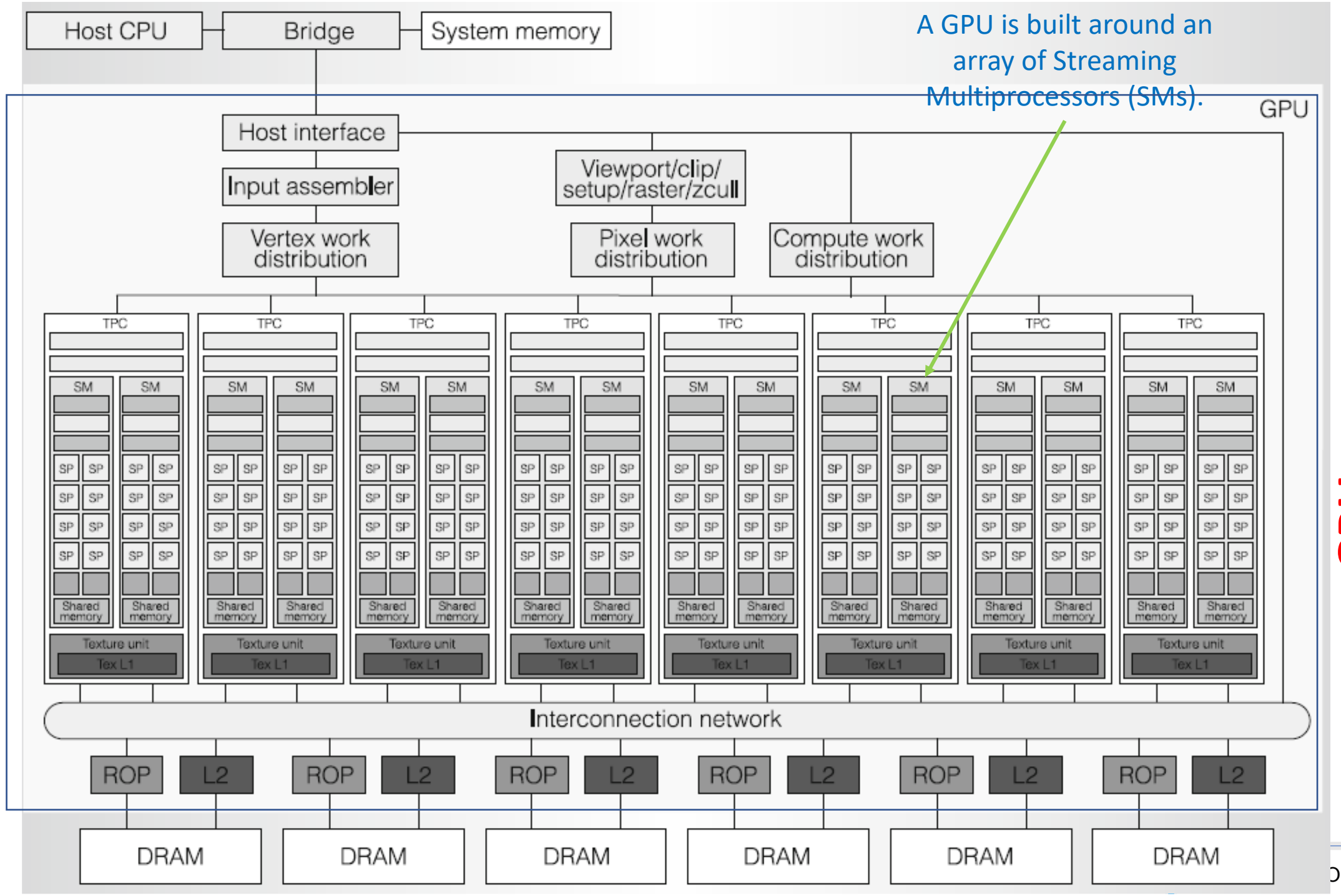
Unsupervised learning is a type of architecture to draw inferences from input **data without labels**

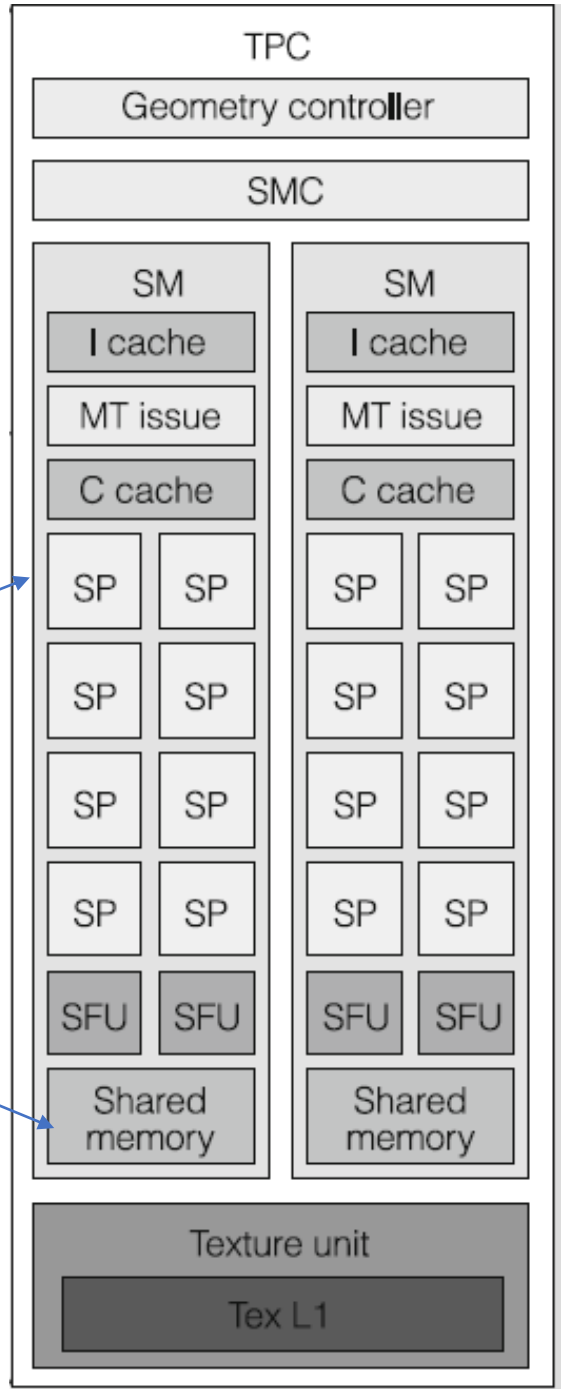


Input Cell | Output Cell | Hidden Cell | Recurrent Cell | Memory Cell | Gated Memory Cell

DEEP LEARNING WORKFLOW




A GPU is built around an array of Streaming Multiprocessors (SMs).
Host CPU
Bridge
System memory
GPU
Host interface
Input assembler
Vertex work distribution
Viewport/clip/ setup/raster/zcull
Pixel work distribution
Compute work distribution
TPC
SM
SP
Shared memory
Texture unit
Tex L1
Interconnection network
ROP
L2
DRAM
GPU
GPU

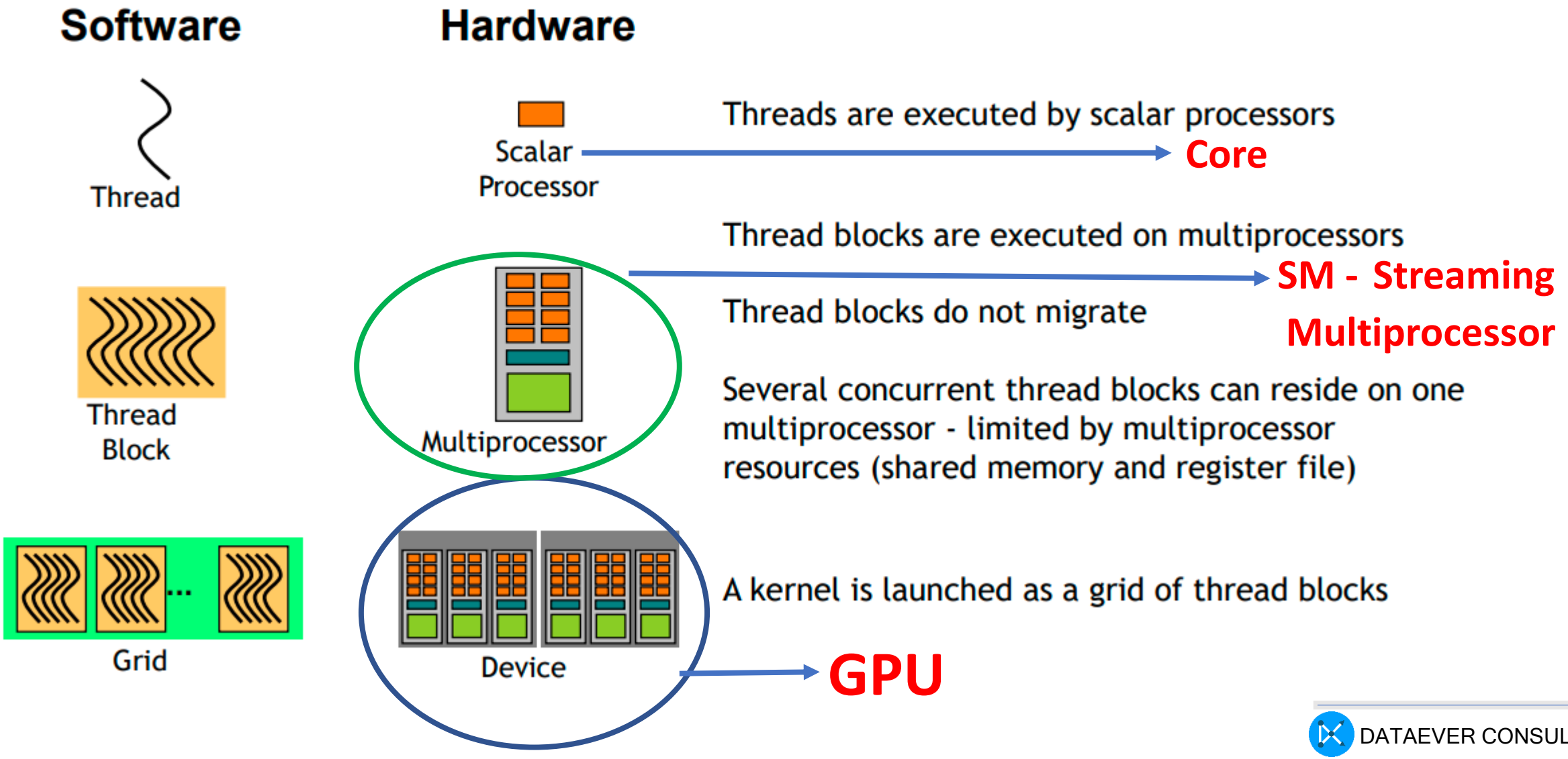


Threads from the same block have access to a shared memory(SM) and their execution can be synchronized

CUDA-Compute Unified Device Architecture
SIMT – Single Instruction Multiple Threads
TPC- Texture Processing Cluster
GPC – GPU Processing Cluster
SM - Streaming Multi Processor
SMC – SM cluster
SFU – SP Function Unit
SP – Core / sequential processor

How the execution happens in GPU?

`run ./deviceQuery and show`



A warp is a collection of threads, 32

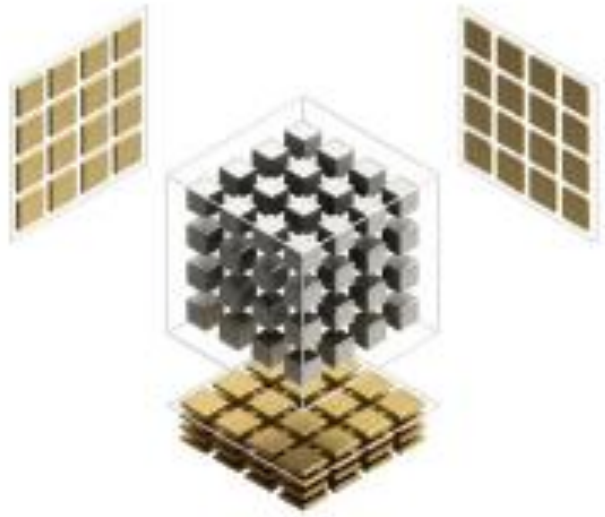
Threads from the same block have access to a shared memory(SM) and their execution can be synchronized

SM - Streaming Multi Processor
SFU – SP Function Unit
SP – Core / sequential processor

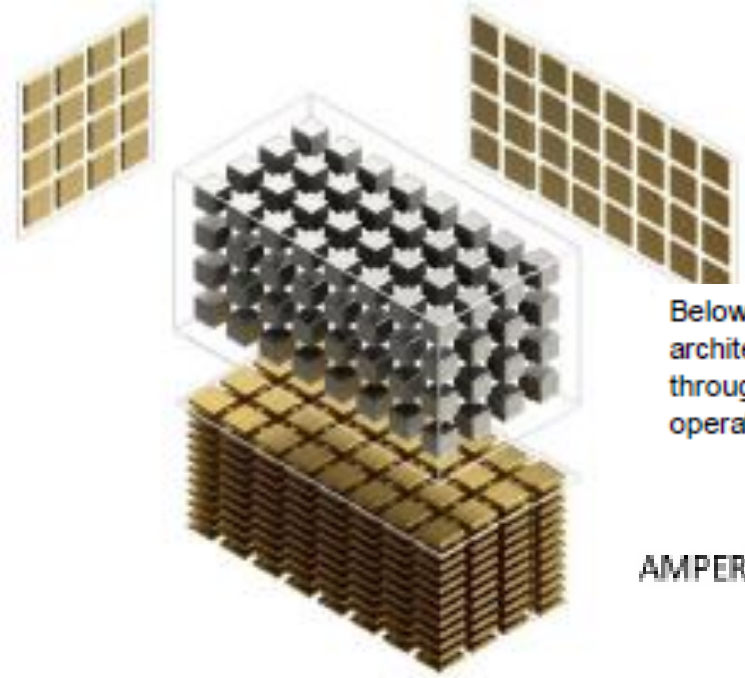
Warps execution

- Warps are the basic unit of execution in an Streaming Multiprocessors (SM)
- Once a thread block is scheduled to an SM, threads in the thread block are further partitioned into warps
- A warp consists of 32 consecutive threads and all threads in a warp are executed in Single Instruction Multiple Thread (SIMT) fashion
  - all threads execute the same instruction, and each thread carries out that operation on its own private data

	TU102 SM (RTX 2080 Super)	GA100 SM (A100)	GA10x SM (RTX 3080)
GPU Architecture	NVIDIA Turing	NVIDIA Ampere	NVIDIA Ampere
Tensor Cores per SM	8	4	4
FP16 FMA operations per Tensor Core	64	Dense: 256 Sparse: 512	Dense: 128 Sparse: 256
Total FP16 FMA operations per SM	512	Dense: 1024 Sparse: 2048	Dense: 512 Sparse: 1024



TURING ARCHITECTURE TENSOR CORE
(GeForce RTX 2080 Super)

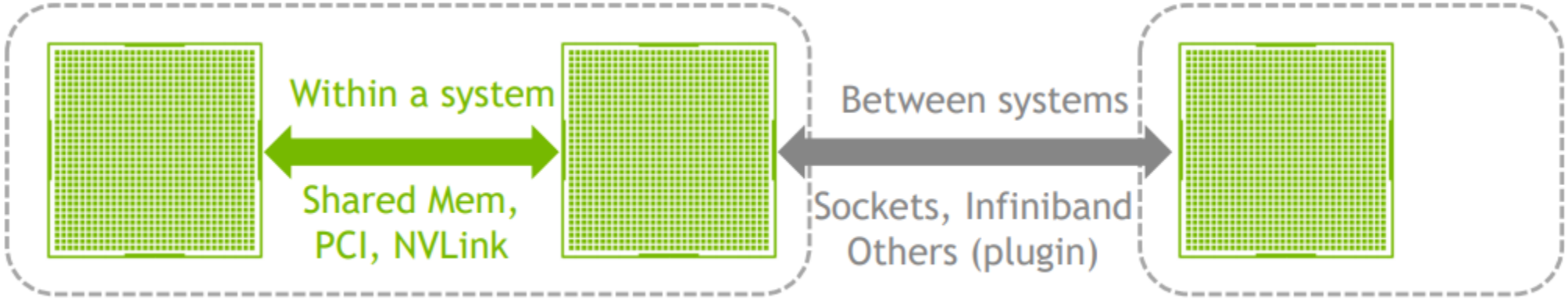


AMPERE ARCHITECTURE TENSOR CORE with Sparsity
(GeForce RTX 3080)

Below is a visual depiction of a single Ampere architecture Tensor Core
architecture Tensor Core performing matrix math calculations and show
throughputs of RTX 3080 vs RTX 2080 Super as represented by the st
operations performed over the same amount of time.

INTER-GPU COMMUNICATION

Intra-node and Inter-node



GPU compute specs:Turing,Ampere

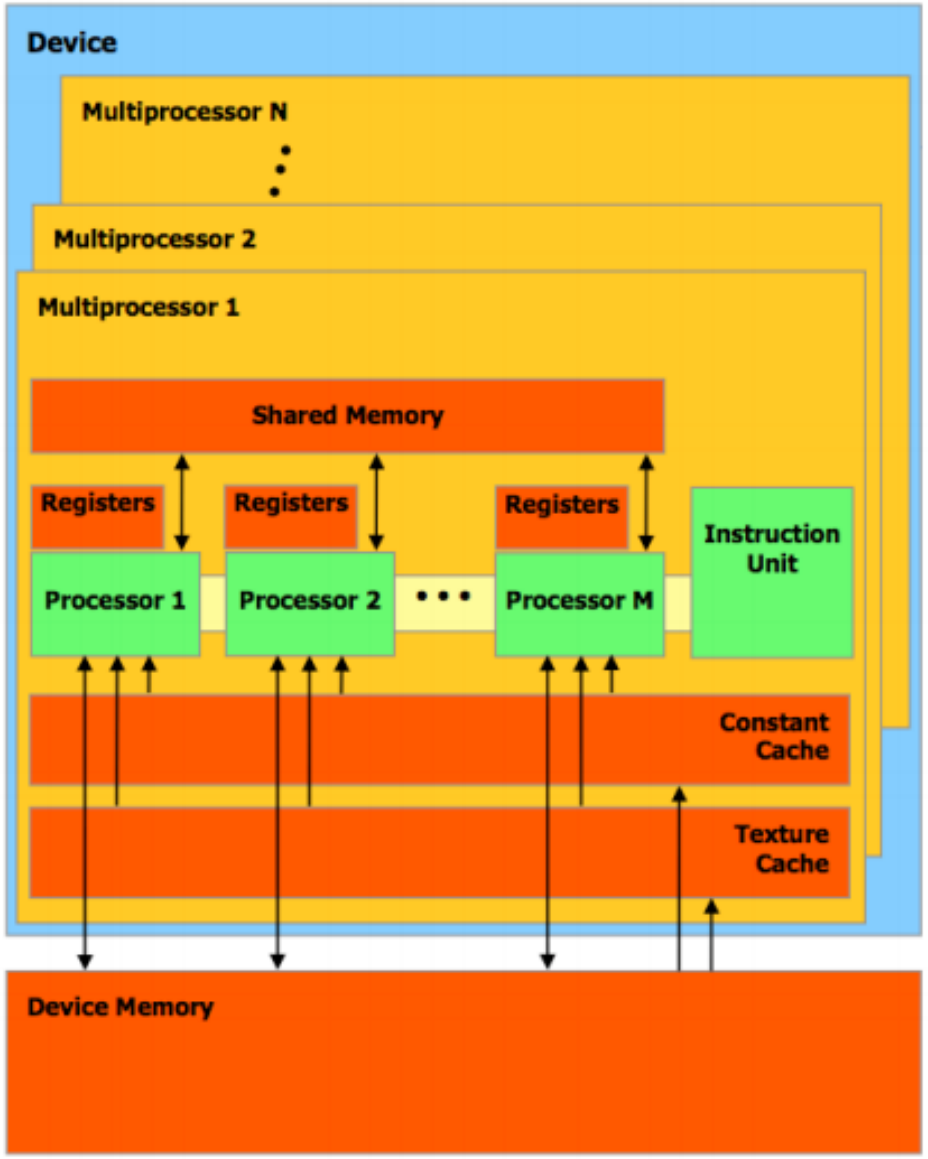
Graphics Card	GeForce RTX 2080 Founders Edition	GeForce RTX 2080 Super Founders Edition	GeForce RTX 3080 10 GB Founders Edition
GPU Codename	TU104	TU104	GA102
GPU Architecture	NVIDIA Turing	NVIDIA Turing	NVIDIA Ampere
GPCs	6	6	6
TPCs	23	24	34
SMs	46	48	68
CUDA Cores / SM	64	64	128
CUDA Cores / GPU	2944	3072	8704
Tensor Cores / SM	8 (2nd Gen)	8 (2nd Gen)	4 (3rd Gen)
Tensor Cores / GPU	368	384 (2nd Gen)	272 (3rd Gen)
RT Cores	46 (1st Gen)	48 (1st Gen)	68 (2nd Gen)
GPU Boost Clock (MHz)	1800	1815	1710
Peak FP32 TFLOPS (non-Tensor)[1]	10.6	11.2	29.8
Peak FP16 TFLOPS (non-Tensor)[1]	21.2	22.3	29.8
Peak BF16 TFLOPS (non-Tensor)[1]	NA	NA	29.8
Peak INT32 TOPS (non-Tensor)[1,3]	10.6	11.2	14.9
Peak FP16 Tensor TFLOPS with FP16 Accumulate[1]	84.8	89.2	119/238[2]
Peak FP16 Tensor TFLOPS with FP32 Accumulate[1]	42.4	44.6	59.5/119[2]
Peak BF16 Tensor TFLOPS with FP32 Accumulate[1]	NA	NA	59.5/119[2]
Peak TF32 Tensor TFLOPS[1]	NA	NA	29.8/59.5[2]
Peak INT8 Tensor TOPS[1]	169.6	178.4	238/476[2]

https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf

GPU memory specs:Turing,Ampere

Graphics Card	GeForce RTX 2080 Founders Edition	GeForce RTX 2080 Super Founders Edition	GeForce RTX 3080 10 GB Founders Edition
Peak INT4 Tensor TOPS[1]	339.1	356.8	476/952[2]
Frame Buffer Memory Size and Type	8192 MB GDDR6	8192 MB GDDR6	10240 MB GDDR6X
Memory Interface	256-bit	256-bit	320-bit
Memory Clock (Data Rate)	14 Gbps	15.5 Gbps	19 Gbps
Memory Bandwidth	448 GB/sec	496 GB/sec	760 GB/sec
ROPs	64	64	96
Pixel Fill-rate (Gigapixels/sec)	115.2	116.2	164.2
Texture Units	184	192	272
Texel Fill-rate (Gigatexels/sec)	331.2	348.5	465
L1 Data Cache/Shared Memory	4416 KB	4608 KB	8704 KB
L2 Cache Size	4096 KB	4096 KB	5120 KB
Register File Size	11776 KB	12288 KB	17408 KB
TGP (Total Graphics Power)	225 W	250 W	320W
Transistor Count	13.6 Billion	13.6 Billion	28.3 Billion
Die Size	545 mm^2	545 mm^2	628.4 mm^2
Manufacturing Process	TSMC 12 nm FFN (FinFET NVIDIA)	TSMC 12 nm FFN (FinFET NVIDIA)	Samsung 8 nm 8N NVIDIA Custom Process

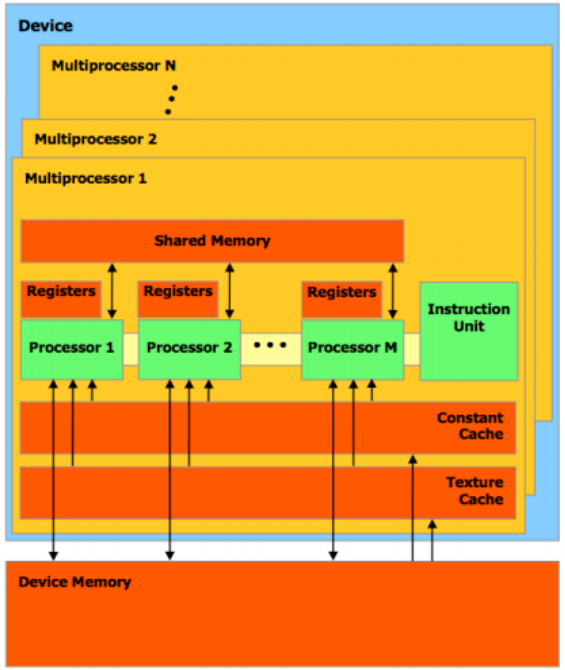
CUDA memory types



- Shared memory partitioned amongst Thread Blocks resident on the Streaming Multiprocessors
- Registers are partitioned amongst Threads

Types of Memory

- Data stored in **register memory** is visible only to the thread that wrote it and lasts only for the lifetime of that thread
- **Local memory** has the same scope rules as register memory, but performs slower
- Data stored in **shared memory** is visible to all threads within that block and lasts for the duration of the block. This is invaluable because this type of memory allows for threads to communicate and share data between one another
- Data stored in **global memory** is visible to all threads within the application (including the host), and lasts for the duration of the host allocation compared to global memory



Types of Memory

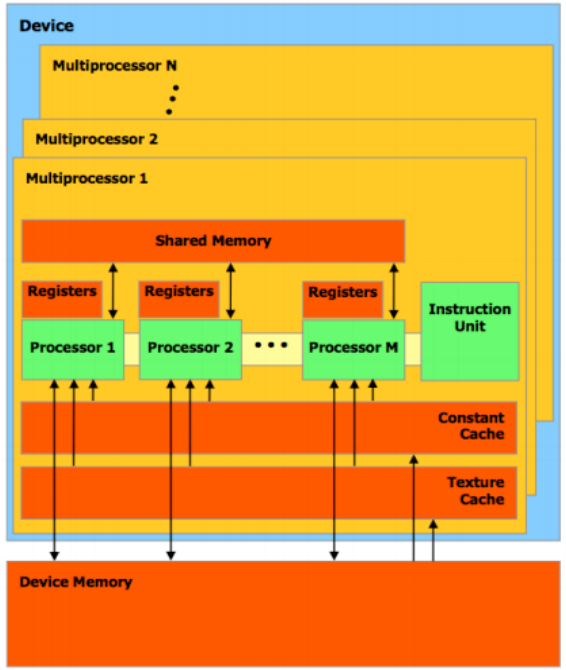
Constant and texture memory for special applications

Constant memory is used for data that will not change over the course of a kernel execution and is read only

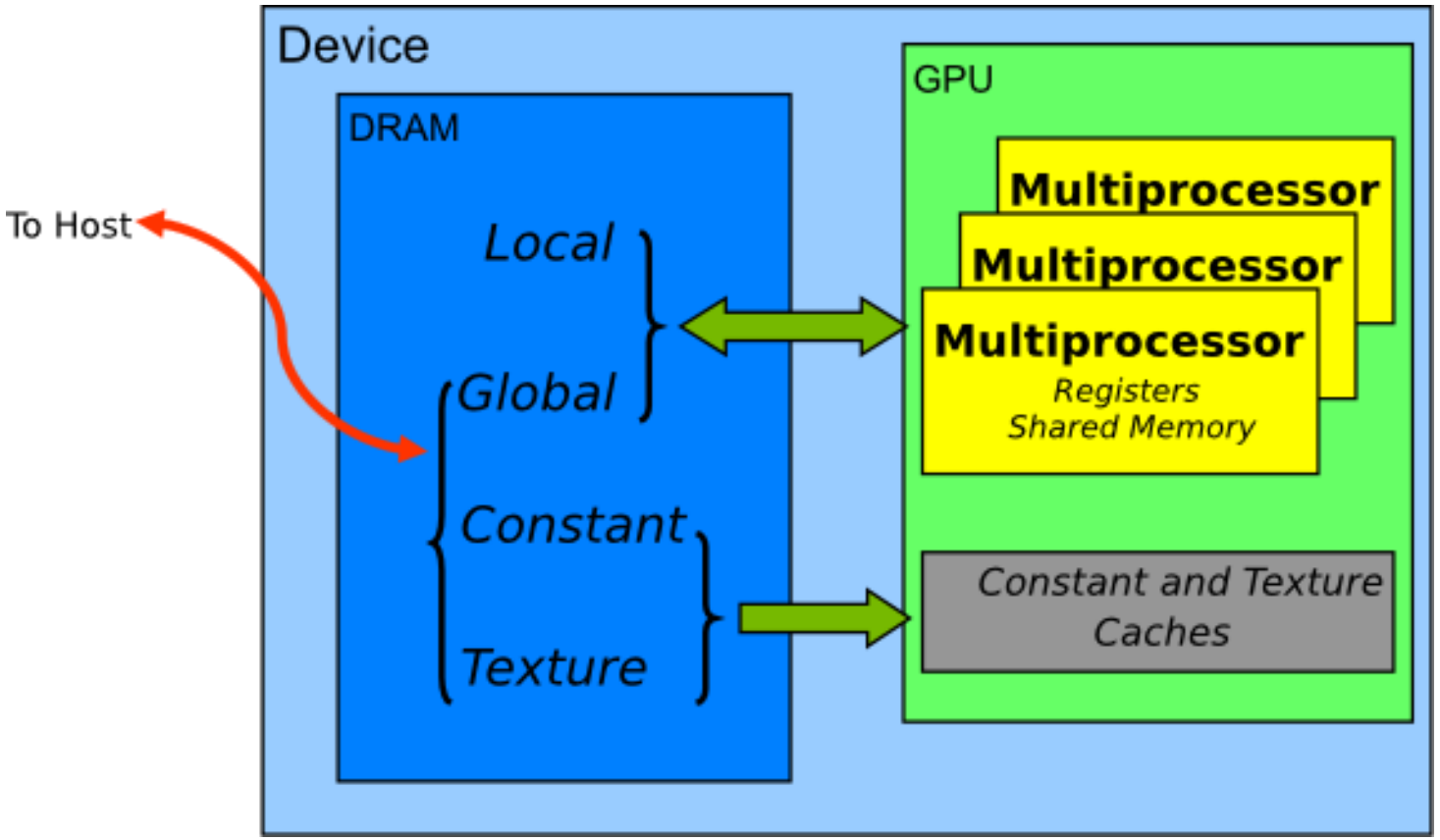
- Using constant rather than global memory can reduce the required memory bandwidth, however, this performance gain can only be realized when a warp of threads read the same location.

Texture memory is another variety of read-only memory on the device

- When all reads in a warp are physically adjacent, using texture memory can reduce memory traffic and increase performance compared to global memory.



Memory spaces

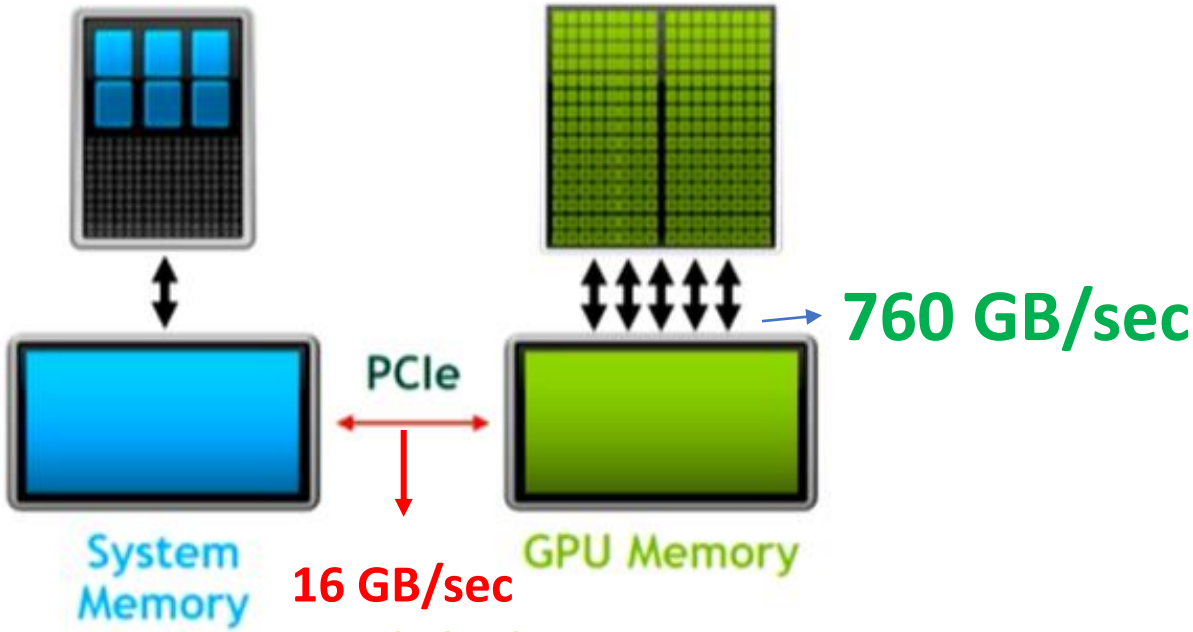


Memory spaces, which have different characteristics that reflect their distinct usages in CUDA applications

Data transfer between Host and Device

The peak theoretical bandwidth between the

GPU memory and the GPU > host system memory and GPU memory



No. Minimize as much as possible

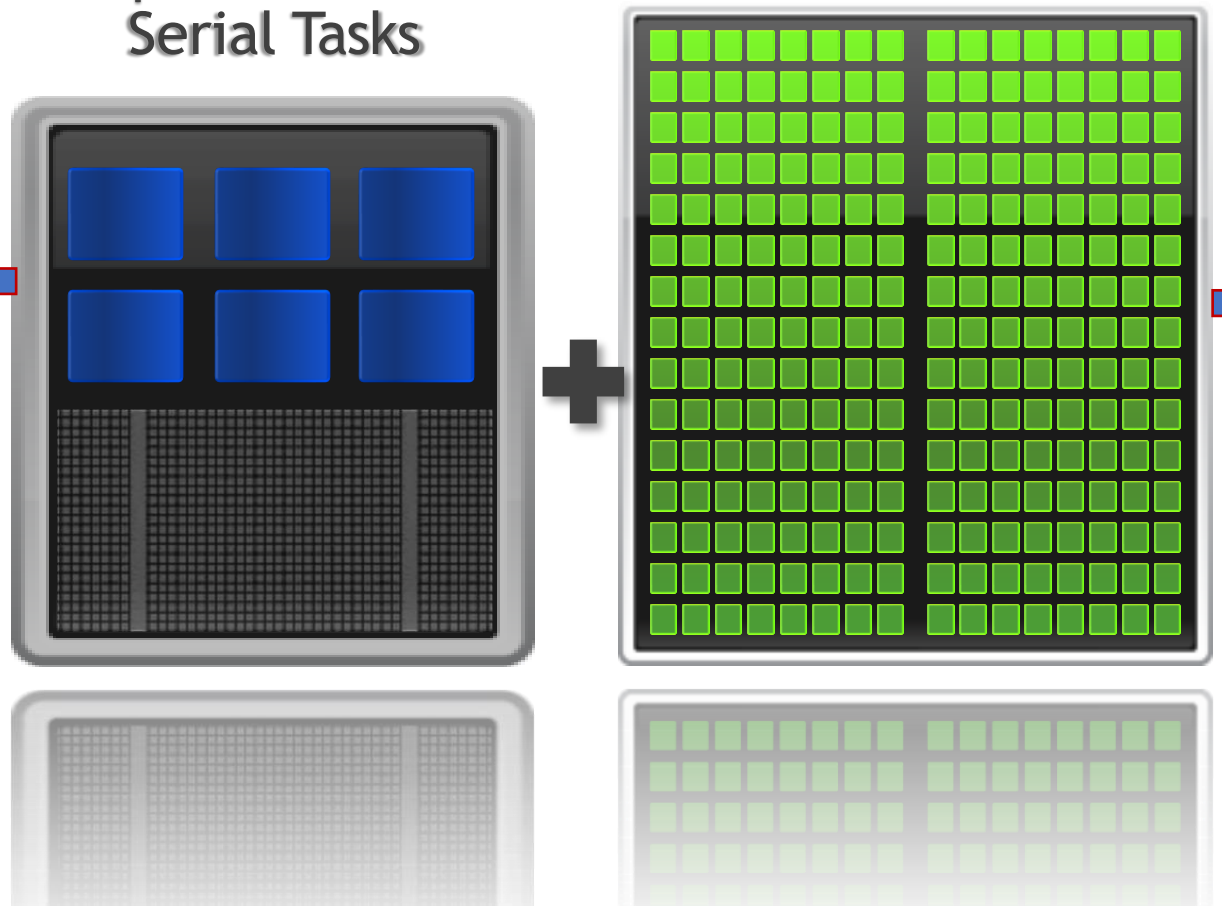
ACCELERATED COMPUTING

10X PERFORMANCE & 5X ENERGY EFFICIENCY FOR HPC

CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks

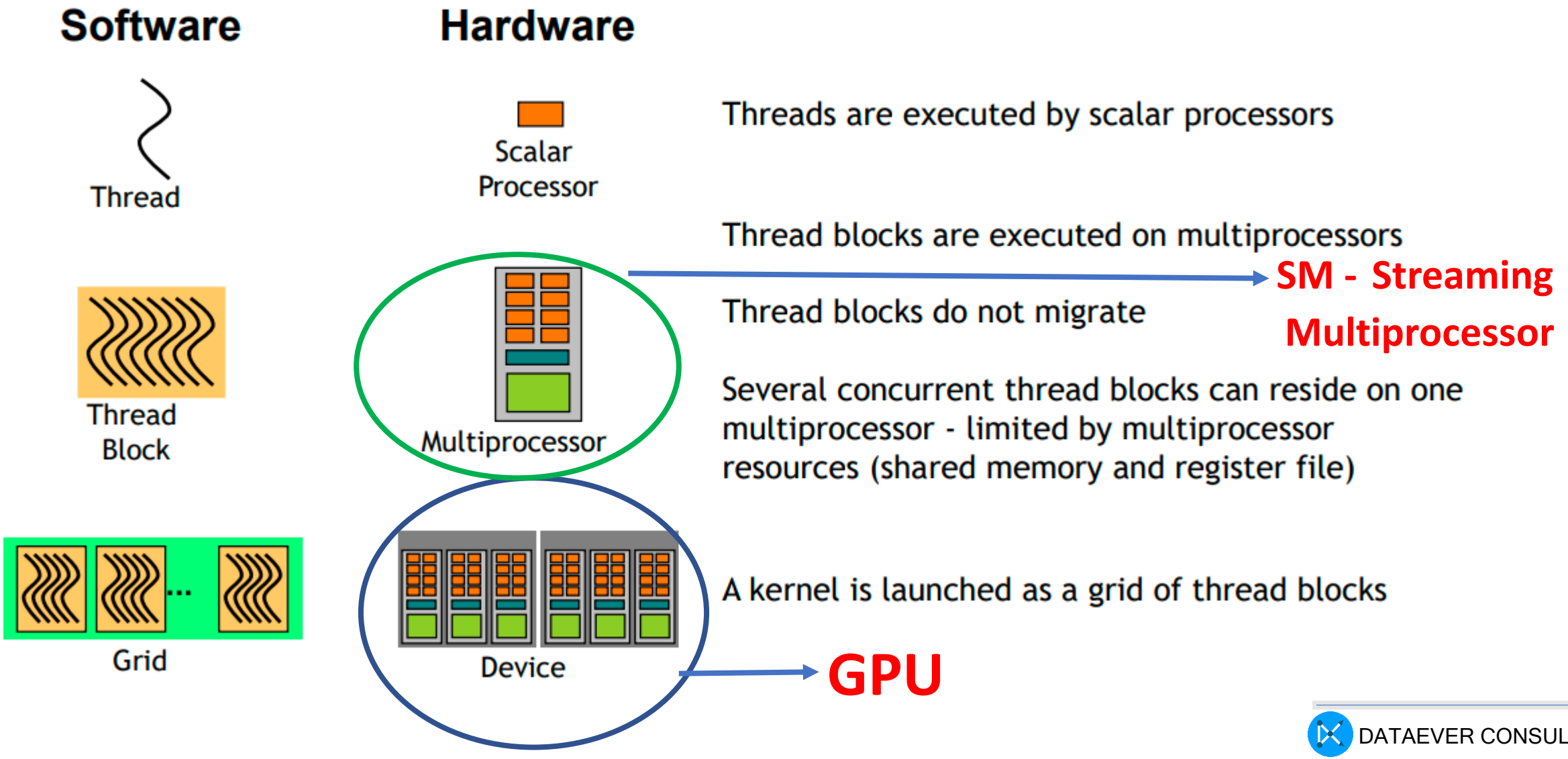
HOST



DEVICE

How the execution happens in GPU?

`run ./deviceQuery and show`



AI Acceleration in Neural Network

Three things are happening in NN computing

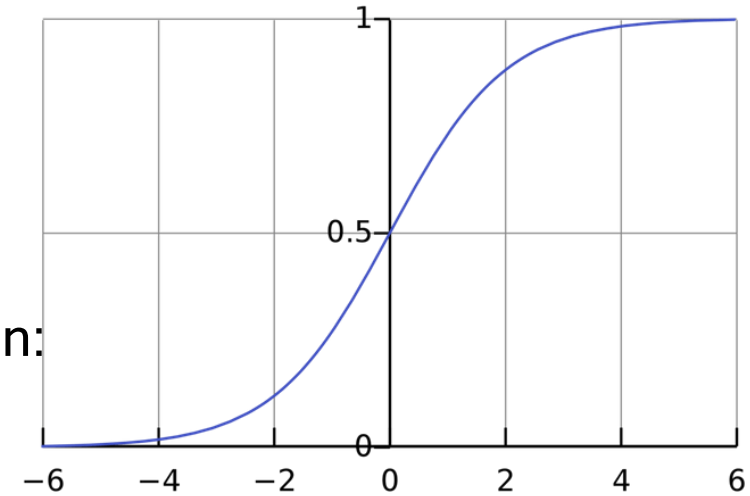
$$S(x) = \frac{1}{1 + e^{-x}}$$

1. First, each input is multiplied by a weight:
   - x1 → x1 * **w1**
   - x2 → x2 * **w2**

   SIMD
2. all weighted inputs are added together with a bias **b**:
   - (x1 * **w1**) + (x2 * **w2**) + **b**

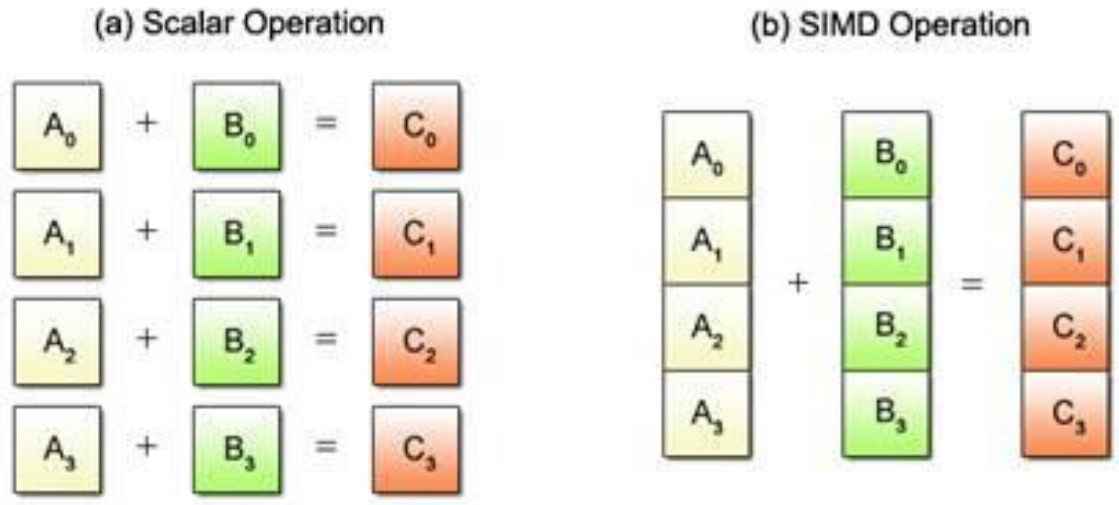
   SIMD
3. Finally, the sum is passed through an activation function:
   - y = **f** (x1 * w1 + x2 * w2 + b)

   SIMD



- The activation function is used to turn an unbounded input into an output that has a nice, predictable form. A commonly used activation function is the sigmoid function:

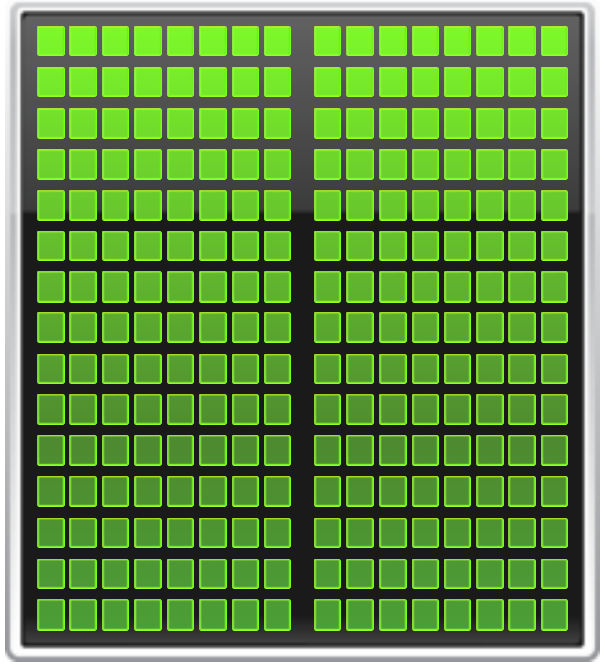
GPU driven AI acceleration



GPU Accelerator

Optimized for

SIMD



CUDA Software Environment

- The challenge is to develop application software that scales up to leverage the many processor cores, and 3D graphics applications scales leverage many GPUs

- The CUDA parallel programming model is designed to overcome this challenge while maintaining a low learning curve for programmers familiar with standard programming languages such as C

GPU Computing Applications

Libraries and Middleware

cuDNN TensorRT	cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX iRay	MATLAB Mathematica

Programming Languages

C	C++	Fortran	Java Python Wrappers	DirectCompute	Directives (e.g. OpenACC)

CUDA-Enabled NVIDIA GPUs

NVIDIA Ampere Architecture (compute capabilities 8.x)				Tesla A Series
NVIDIA Turing Architecture (compute capabilities 7.x)		GeForce 2000 Series	Quadro RTX Series	Tesla T Series
NVIDIA Volta Architecture (compute capabilities 7.x)	DRIVE/JETSON AGX Xavier		Quadro GV Series	Tesla V Series
NVIDIA Pascal Architecture (compute capabilities 6.x)	Tegra X2	GeForce 1000 Series	Quadro P Series	Tesla P Series
				

Tensor RT (Tensor Flow RunTime)

- Introduction:
  - Nvidia TensorRT is an SDK for high-performance deep learning inference.
  - It optimizes trained neural networks based on TensorFlow for deployment on AI Accelerated Devices like NVIDIA GPU
- Features:
  - Precision calibration for reduced compute and memory usage.
  - Dynamic tensor memory management for increased inference throughput.
  - Integration with Nvidia DeepStream

Tensor RT

Model: ResNet-50
Batch size: 64
Precision: FP32
Device: NVIDIA Tesla V100

	Native Inference time	TensorRt inference Time	Speedup
1	6.7 ms	1.2 ms	5.5

Tensor RT performance comparison

Titan V GPU with Tensor RT

- Object Detection, COCO dataset, ResNet50-based model
  - 33 x speedup
- Image classification
  - 18x speedup

Intel Xeon Gold 6140 CPU

- Object Detection, COCO dataset, ResNet50-based model
  - 1x Speedup
- Image classification
  - 1x speedup

Inference benchmark performance comparison

Jetson Nano embedded GPU with Tensor RT

- 48x speedup
- Image classification, ImageNet dataset, ResNet50-based model

Intel Core i7-6700K CPU without Tensor RT

- 1x speedup
- Image classification, ImageNet dataset, ResNet50-based model

Xylinx FPGA architecture

Xilinx® UltraScale™ architecture comprises high-performance FPGA, MPSoC, and RFSoC families that address a vast spectrum of system requirements with a focus on lowering total power consumption through numerous innovative technological advancements.

Artix® UltraScale+ FPGAs: High…ptimized device for critical networking applications, vision a…

Kintex® UltraScale FPGAs: High…ing both monolithic and next-generation stacked silicon i…gic ratios and next-generation transceivers, combined with low-…cost.

Kintex UltraScale+™ FPGAs: Inc…ce BOM cost. The ideal mix of high-performance peripherals an…FPGAs have numerous power options that deliver the optimal balance between the required system performance and the smallest power envelope.

Virtex® UltraScale FPGAs: High-capacity, high-performance FPGAs enabled using both monolithic and next-generation SSI technology. Virtex UltraScale devices achieve the highest system capacity, bandwidth, and performance to address key market and application requirements through integration of various system-level functions.

Virtex UltraScale+ FPGAs: The highest transceiver bandwidth, highest DSP count, and highest on-chip and in-package memory available in the UltraScale architecture. Virtex UltraScale+ FPGAs also provide numerous power options that deliver the optimal balance between the required system performance and the smallest power envelope.

Zynq® UltraScale+ MPSoCs: Combine the Arm® v8-based Cortex®-A53 high-performance energy-efficient 64-bit application processor with the Arm Cortex-R5F real-time processor and the UltraScale architecture to create the industry's first programmable MPSoCs. Provide unprecedented power savings, heterogeneous processing, and programmable acceleration.

An FPGA, a programmable hardware doesn't do anything itself but it can be configured to be just about any digital circuit you want. Nothing physically changes. You simply load a configuration into the FPGA and it starts behaving like the circuit you wanted.

Figure 2: Compared to alternative devices such as GPUs and ASICs, FPGAs are the only viable choice for the event trigger processing because they provide extremely low latency. While the large numerical processing capability of GPUs is attractive, these technologies are optimized for high throughput, not low latency.

Optimize Trigger Filter Algorithm Development CERN where a global community conducts **Research in fundamental physics to better understand the universe** and how it works.

The LHC at CERN is the place where subatomic particles are accelerated and smashed to produce new particles, which are then revealed by an array of detectors and sensor systems. High energy particle physics experiments at CERN, like the recent observation of the Higgs boson, are the key to advancing the frontiers of human knowledge about the universe.

Compared to GPUs and ASICs, FPGAs are viable choice for the event trigger processing because they provide extremely low latency. While the large numerical processing capability of GPUs is attractive, these technologies are optimized for high throughput, not low latency. **The very high data rates (150 Terabytes/second) in the CMS detector requires event processing in real-time. The trigger filter algorithm was also modified.**

Host and Device ?

Source code in HIP has two flavors: Host code and Device code

- The Host is the CPU
- Host code runs here
- Usual C++ syntax and features
- Entry point is the 'main' function
- HIP API can be used to create device buffers, move between host and device, and launch device code.

- The Device is the GPU
- Device code runs here
- C-like syntax
- Device codes are launched via "kernels"
- Instructions from the Host are enqueued into "streams"

X86-64 micro architectures

Nehalem

- released November 17, 2008, built on a 45 nm process and used in the Core i7, Core i5, Core i3 microprocessors.
  - **Westmere**: 32 nm shrink of the Nehalem microarchitecture with several new features.

Sandy Bridge

- 32 nm microarchitecture, released January 9, 2011. Formerly called Gesher but renamed in 2007.[1] First x86 to introduce 256 bit AVX instruction set and implementation of YMM register.
- **Ivy Bridge**: successor to Sandy Bridge, using 22 nm process, released in April 2012.

Haswell

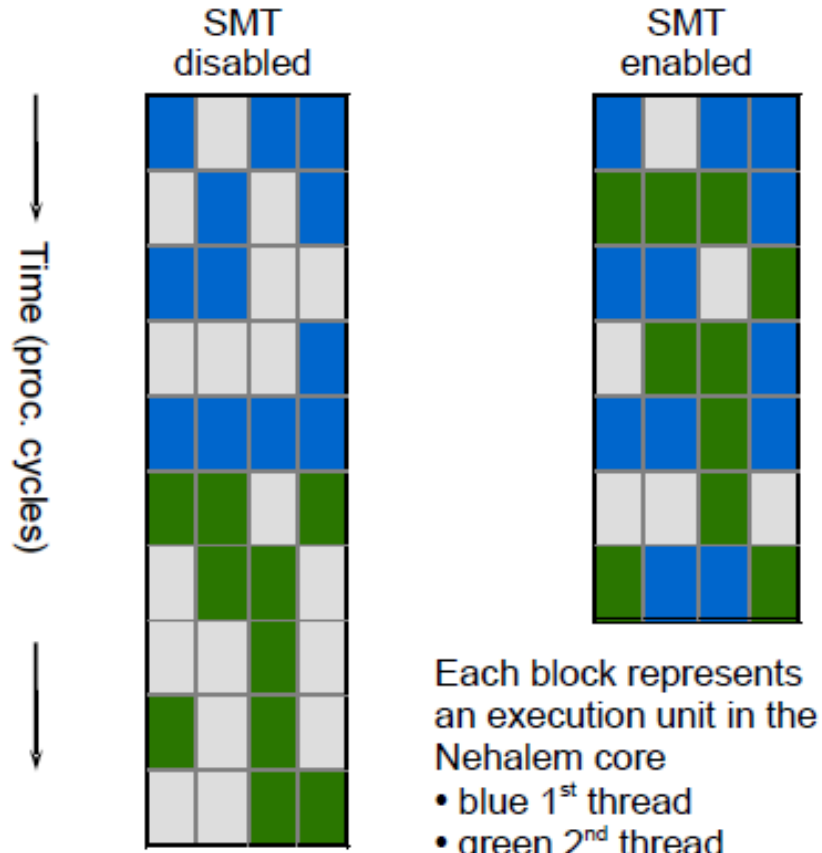
- 22 nm microarchitecture, released June 3, 2013. Added a number of new instructions, including FMA.
- Broadwell: 14 nm shrink of the Haswell microarchitecture, released in September 2014. Formerly called Rockwell.

Skylake

- 14 nm microarchitecture, released August 5, 2015.
  - Koby Lake: successor to Skylake, released in August 2016, broke
  - Intel's Tick-Tock schedule due to delays with the 10 nm process.
  - **Coffee Lake:** successor to Kaby Lake, using 14+ nm process, released in October 2017
  - Cascade Lake: server and high-end desktop successor to Kaby Lake-X, using 14 nm process, released in April 2019
  - Comet Lake: successor to Coffee Lake, using 14++ nm process, released in August 2019
  - Cooper Lake: server-only, optimized for AI oriented workloads using bfloat16, with limited availability only to Intel priority partners, using 14++ nm process, released in 2020

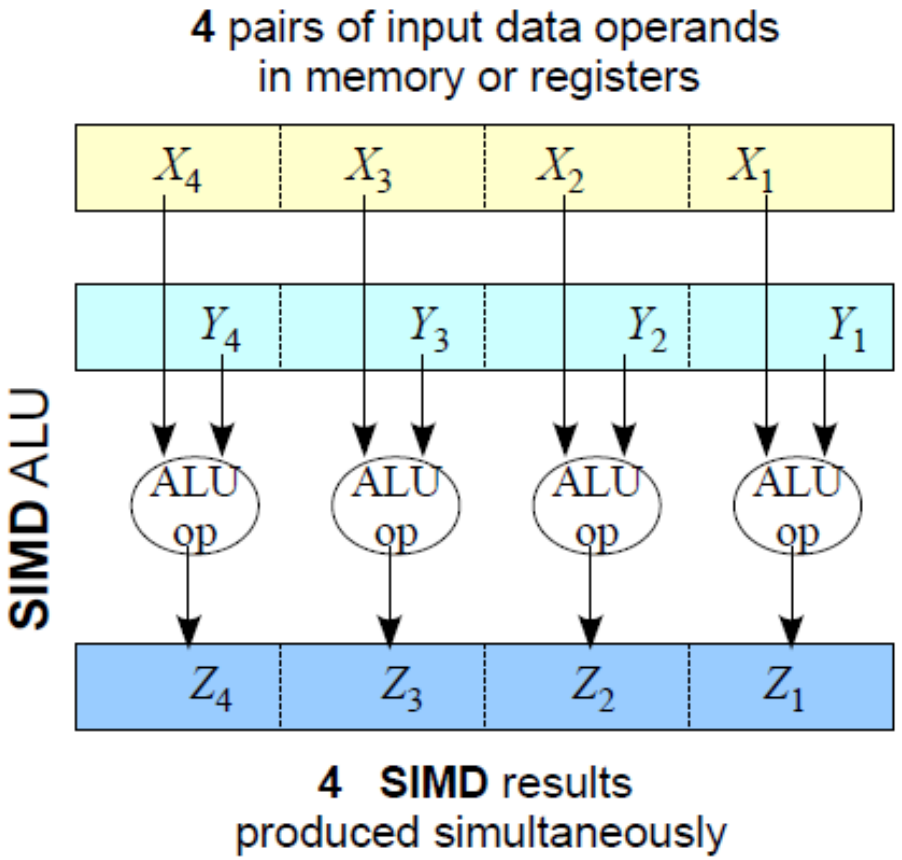
Simultaneous Multi-Threading (SMT) Support in Nehelam

- A Nehalem core supports SMT, or "Hyper-Threading". SMT is a pipeline design and implementation permits more than one hardware threads to execute simultaneously within each core. For Nehalem, two threads can be simultaneously executing within each core.

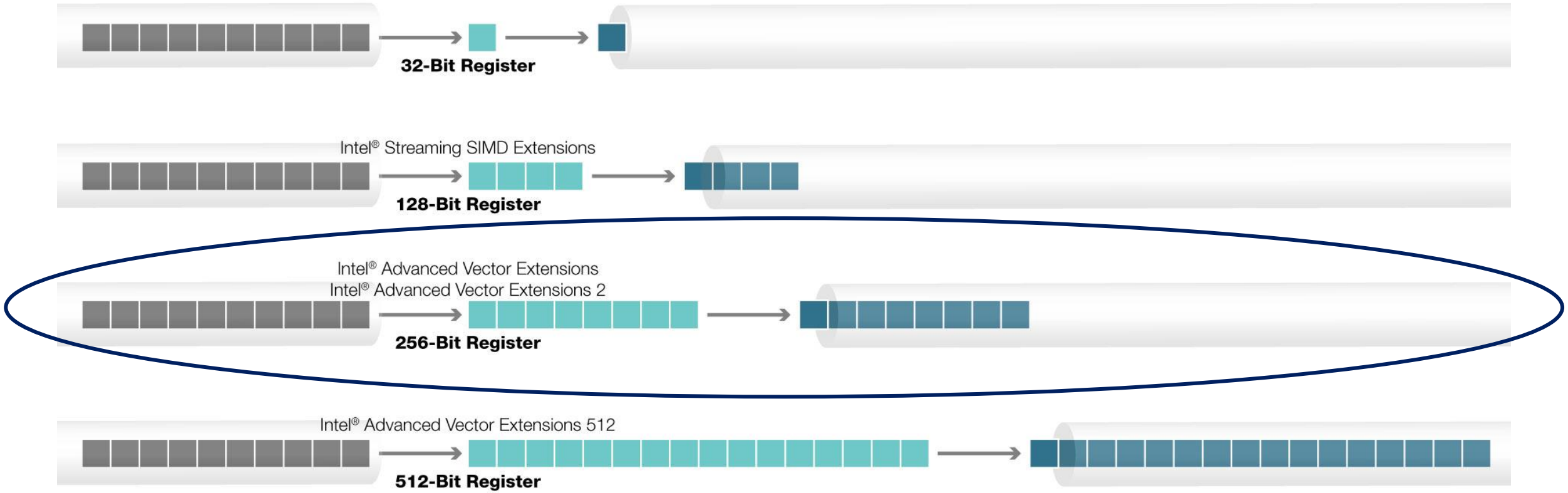


SIMD

- SIMD instructions apply the same FP or integer operation to collections of input data pairs simultaneously



Example: AVX


32-Bit Register
Intel® Streaming SIMD Extensions
128-Bit Register
Intel® Advanced Vector Extensions
Intel® Advanced Vector Extensions 2
256-Bit Register
Intel® Advanced Vector Extensions 512
512-Bit Register

Streaming SIMD Extensions (SSE)

- Supplemental Streaming SIMD Extensions 3 (SSSE3) SSSE3 introduces 32 new instructions to accelerate eight types of computations on packed integers.

SSE4.1

- SSE4.1 introduces 47 new instructions to accelerate video, imaging and 3D applications. SSE4.1 also improves compiler vectorization and significantly increase support for packed dword computation.

SSE4.2

- During 2008 Intel introduced a new set of instructions collectively called as SSE4.2.

SSE4

- SSE4 has been defined for Intel's 45nm products including Nehalem. A set of 7 new instructions for SSE4.2 were introduced in Nehalem architecture in 2008. The first version of SSE4.1 was present in the Penryn processor.
- SSE4.2 instructions are further divided into 2 distinct sub-groups, called "STTNI" and "ATA".

STring and Text New Instructions (STTNI)

- operate on strings of bytes or words of 16bit size. There are four new STTNI instructions which accelerate string and text processing. For example, code can parse XML strings faster and can carry out faster search and pattern matching. Implementation supports parallel data matching and comparison operations.

AI Acceleration Demo

CPU vs GPU Benchmark Classification CIFAR data running on Kaggle

Login to aijuser@dhruv

Visit the following website , login and edit

https://www.kaggle.com/code/snsmsssss/image-classification-ann-cpu-gpu-benchmarking/edit

Explain the code, model and data set

Train the model, it will take time

Later visit the run and show the performance difference

How to check an installed NVIDIA GPU ?

```
Login to sambath@firefly
lspci | grep –i nvidia
nvidia-smi
cat /proc/driver/nvidia/version
```

	GeForce 910M	384 @ 0.64 GHz	64 Bit @ 2000 MHz
Codename	N16S-GTR-B/S		
Architecture	Maxwell		
Pipelines	384 - unified		
Core Speed	1122 - 1242 (Boost) MHz		
Memory Speed	4000 MHz		
Memory Bus Width	64 Bit		
Memory Type	GDDR5, DDR3		
Max. Amount of Memory	4096 MB		
Shared Memory	no		
DirectX	DirectX 12 (FL 11_0), Shader 5.0		
Transistor Count	1870 Million		
technology	28 nm		
Features	GPU Boost 2.0, Optimus, PhysX, CUDA, GeForce Experience,		

https://www.notebookcheck.net/NVIDIA-GeForce-940MX.156033.0.html

./deviceQuery

```
S go2NVI_Samples
cd 1_Utilities/deviceQuery
./deviceQuery
```

/deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking

Detected 1 CUDA Capable device(s)

Device 0: "GeForce 940MX"

CUDA Driver Version / Runtime Version 11.0 / 11.0

CUDA Capability Major/Minor version number: 5.0

Total amount of global memory: 2004 MBytes (2101870592 bytes)

(3) Multiprocessors, (128) CUDA Cores/MP: 384 CUDA Cores

GPU Max Clock rate: 1242 MHz (1.24 GHz)

Memory Clock rate: 1001 Mhz

Memory Bus Width: 64-bit

L2 Cache Size: 1048576 bytes

Maximum Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)

Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers

Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers

Total amount of constant memory: 65536 bytes

Max dimension size of a thread block (x,y,z): (1024, 1024, 64)

Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)

Maximum memory pitch: 2147483647 bytes

Texture alignment: 512 bytes

Concurrent copy and kernel execution: Yes with 1 copy engine(s)

Run time limit on kernels: Yes

Integrated GPU sharing Host Memory: No

Support host page-locked memory mapping: Yes

Alignment requirement for Surfaces: Yes

Device has ECC support: Disabled

Device supports Unified Addressing (UVA): Yes

Device supports Managed Memory: Yes

Device supports Compute Preemption: No

Supports Cooperative Kernel Launch: No

Supports MultiDevice Co-op Kernel Launch: No

Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0

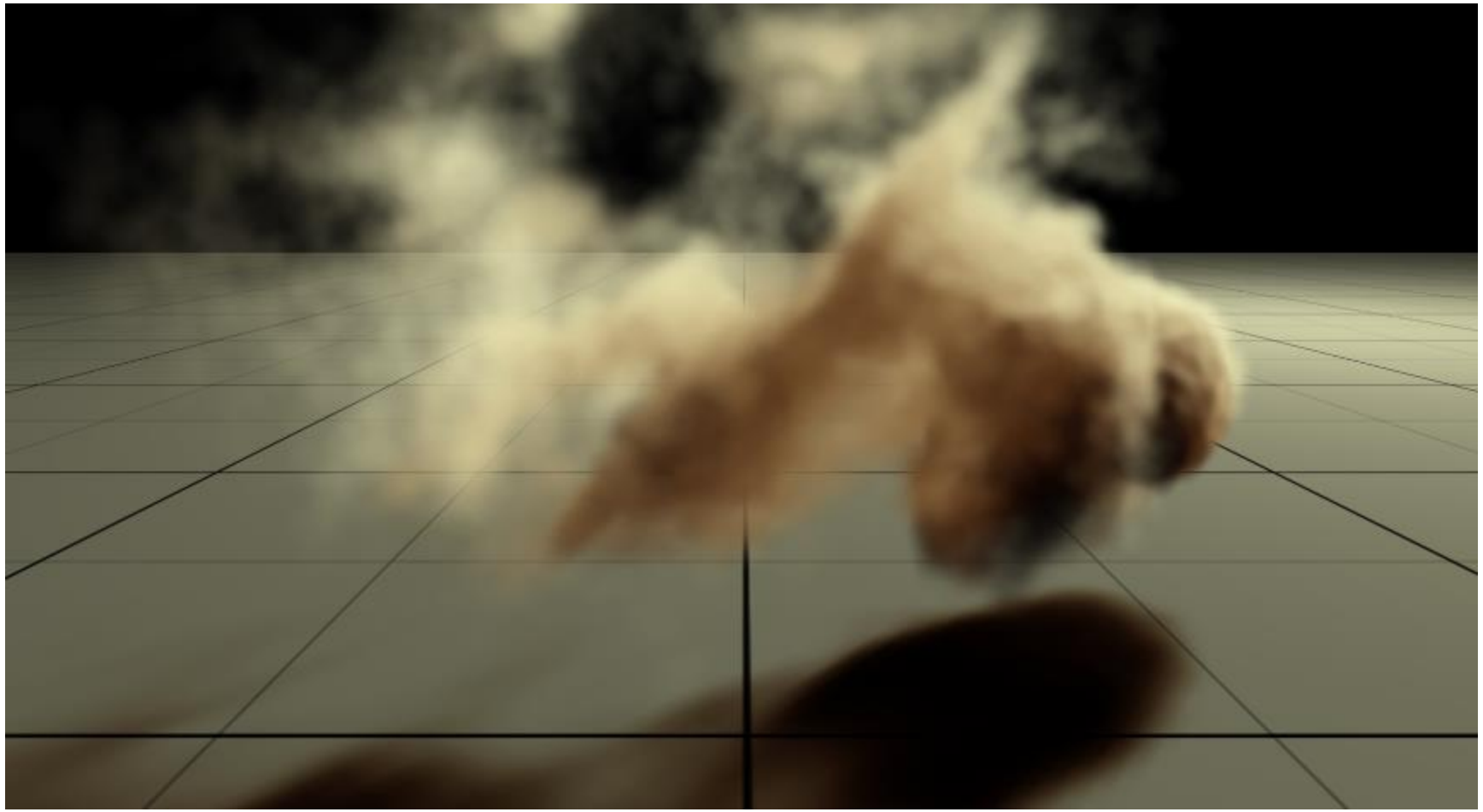
Compute Mode:

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.0, CUDA Runtime Version = 11.0, NumDevs = 1

Result = PASS

The Power of Hardware acceleration for Real Time ray Tracing



Visual Computing

RTX

- RTX – Ray Tracing Texel eXtreme
- Shadows, Reflections using RayTracing
- Hardware and software support
- Raytracing + Rasterization = Hybrid Rendering approach
- Scientific Visualization
- Nvidia – Microsoft - Direct X RayTracingAPI
- Applications: Energy exploration, gaming, film and video production

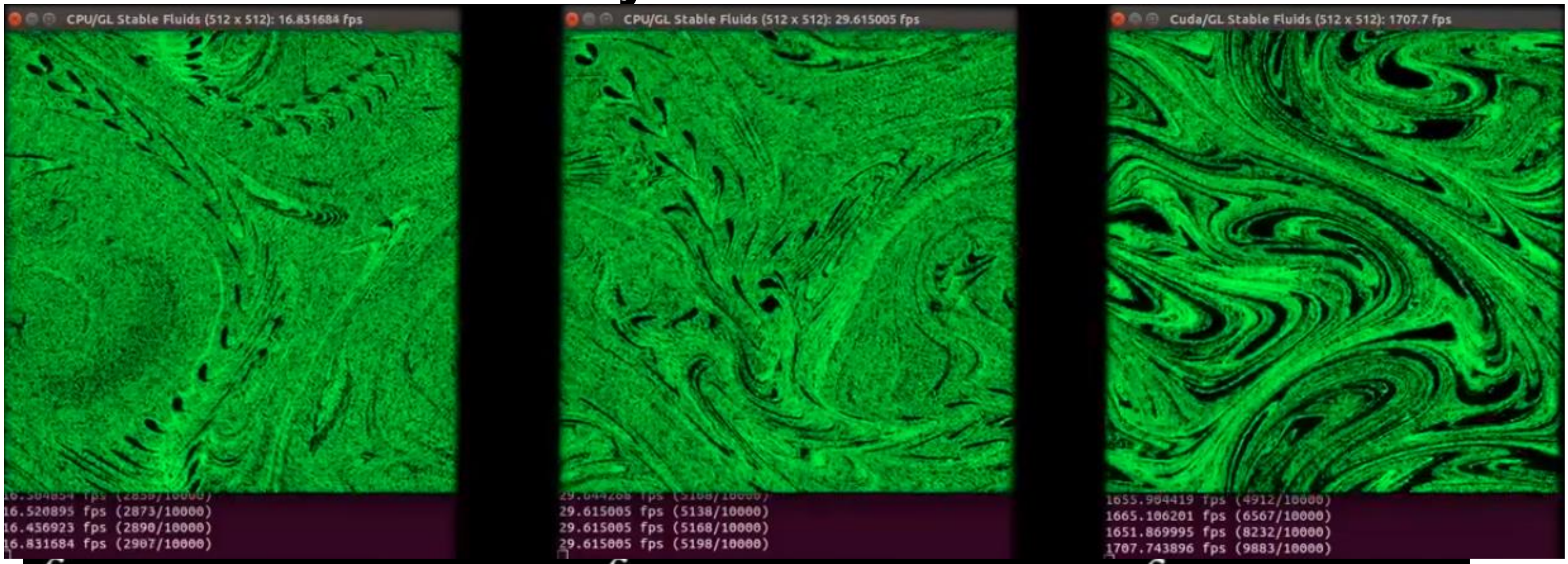
Smoke Particles

- Smoke simulation with volumetric shadows using half-angle slicing technique.
- Physical simulation
- CUDA for simulations
- Graphics interoperability
- SIMD
- GPU device 0: “Maxwell with Compute Capability 5.5
- 77 frames / second

From Node0, home dir ~

```
s gotosmoke
./smokeParticles
```

Real time Fluid Dynamics – CPU vs GPU



fps average:	fps average:	fps average:
17.008575	29.928940	1469.928101
CPU	CPU (Optimized)	GPU

CONSULTING

Real time Fluid Dynamics - CPU vs GPU

- Real-Time Fluid Dynamics: CPU vs GPU
  - https://youtu.be/fE0P6H8eK4I
  - Programming Models
  - OpenMP, Open ACC, MPI
- Implementation Documentation
  - CUDA/OpenGL Fluid simulation by Nolan Goodnight

```
cd

s gotoCPUFluids

make clean

make

Show the display
```

```
cd

s gotoGPUFluids

./fluidsGL

Show the display
```

Sample matrix multiply CUDA C code

```
s go2mm

 Show the CUDA code
 vi mm.cu

  source go-compile
  source go-run
```

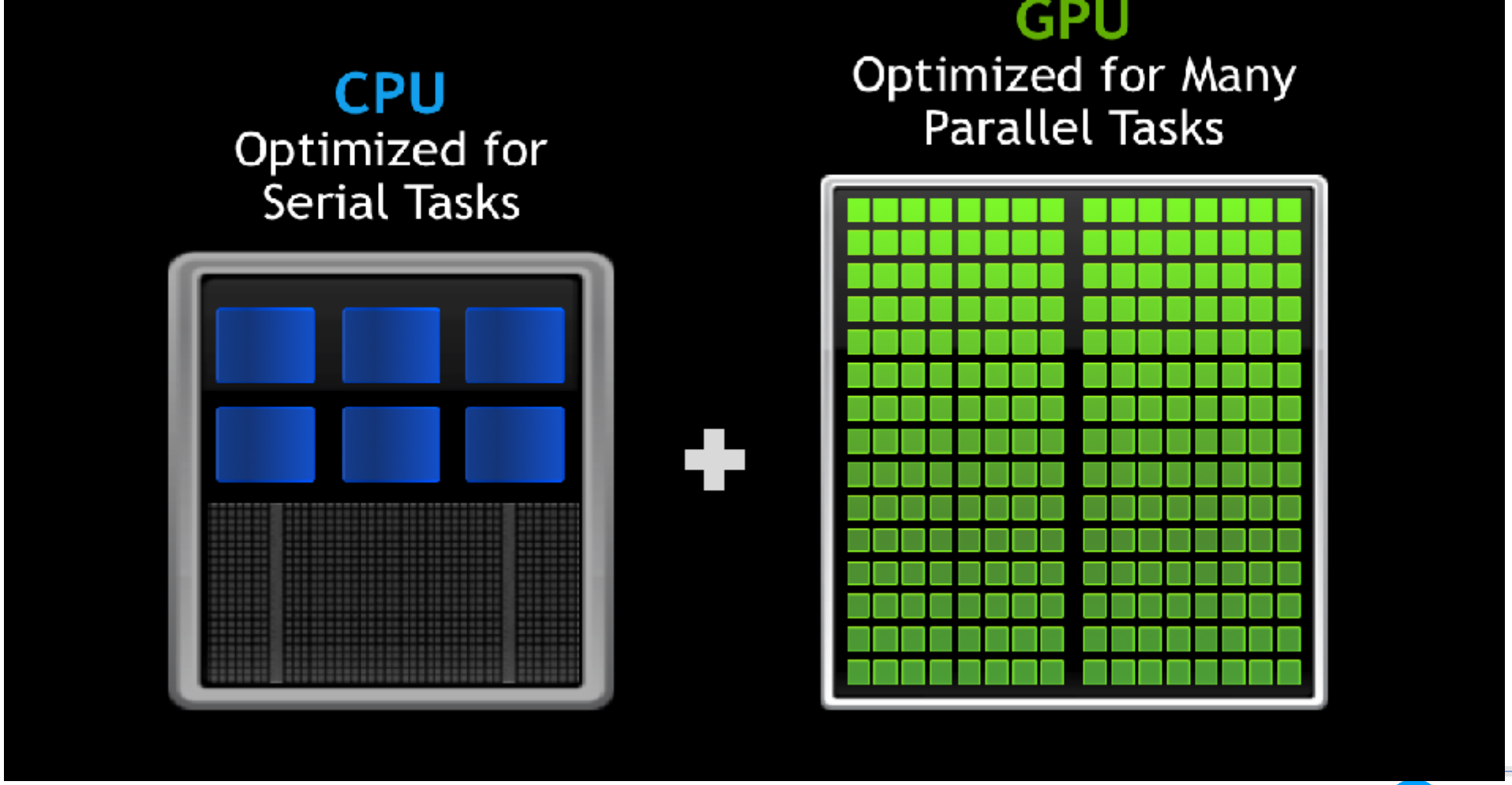

https://www.notebookcheck.net/NVIDIA-GeForce-940MX.156033.0.html

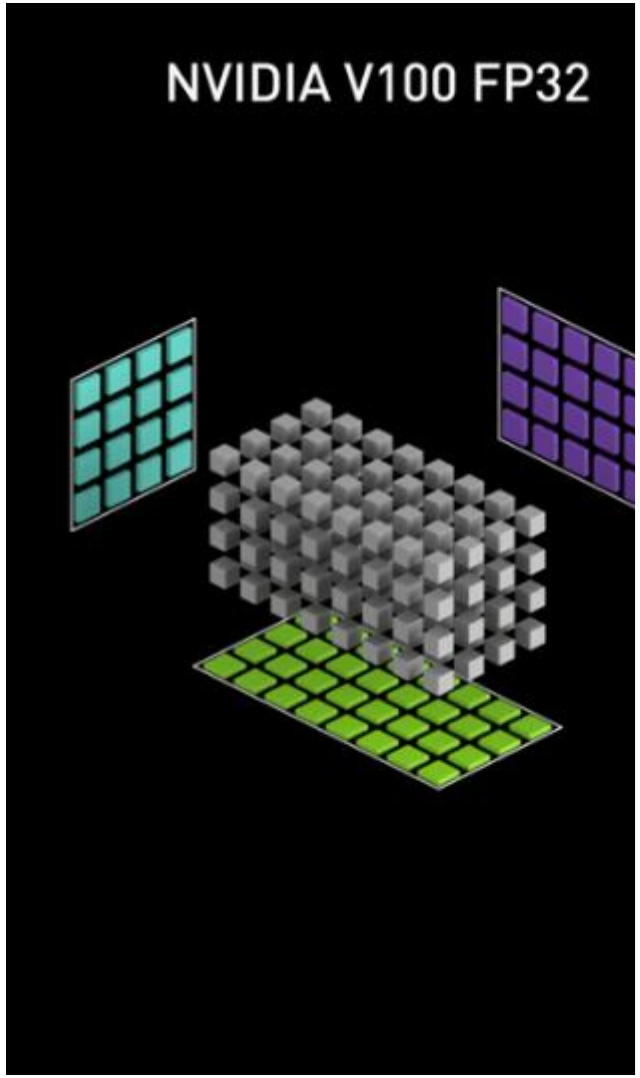
Graphics Acceleration through GPU - demo

```
s go2NVISamples

cd 2_Graphics

./volumeRender

./marchingCubes

./simpleTexture
```

AI accelerated Devices / AI Hardware

High Performance Accelerated Computing





A100 GPU introduces fine-grained structured sparsity

With the A100 GPU, NVIDIA introduces fine-grained structured sparsity, a novel approach that doubles compute throughput for deep neural networks.

Sparsity is possible in deep learning because the importance of individual weights evolves during the learning process, and by the end of network training, only a subset of weights have acquired a meaningful purpose in determining the learned output. The remaining weights are no longer needed.

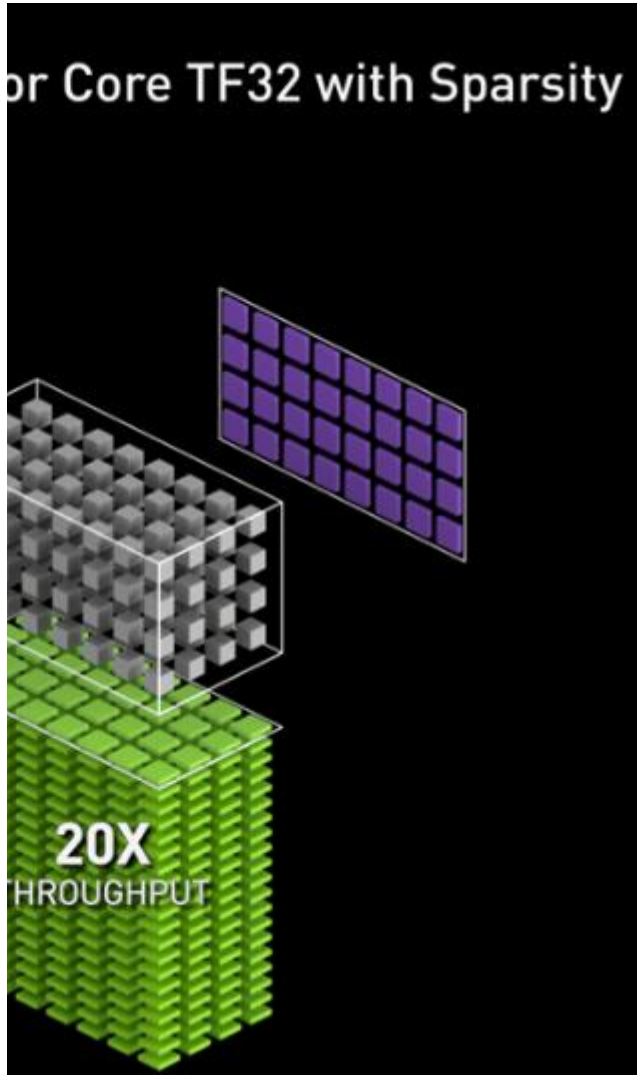
Fine grained structured sparsity imposes a constraint on the allowed sparsity pattern, making it more efficient for hardware to do the necessary alignment of input operands. Because deep learning networks are able to adapt weights during the training process based on training feedback, NVIDIA engineers have found in general that the structure constraint does not impact the accuracy of the trained network for inferencing. This enables inferencing acceleration with sparsity.

For training acceleration, sparsity needs to be introduced early in the process to offer a performance benefit, and methodologies for training acceleration without accuracy loss are an active research area.

Sparse matrix definition

Structure is enforced through a new 2:4 sparse matrix definition that allows two non-zero values in every four-entry vector. A100 supports 2:4 structured sparsity on rows, as shown in Figure 9.

Due to the well-defined structure of the matrix, it can be compressed efficiently and reduce memory storage and bandwidth by almost 2x.



A100 GPU introduces fine-grained structured sparsity

With the A100 GPU, NVIDIA introduces fine-grained structured sparsity, a novel approach that doubles compute throughput for deep neural networks.

Sparsity is possible in deep learning because the importance of individual weights evolves during the learning process, and by the end of network training, only a subset of weights have acquired a meaningful purpose in determining the learned output. The remaining weights are no longer needed.

Fine grained structured sparsity imposes a constraint on the allowed sparsity pattern, making it more efficient for hardware to do the necessary alignment of input operands. Because deep learning networks are able to adapt weights during the training process based on training feedback, NVIDIA engineers have found in general that the structure constraint does not impact the accuracy of the trained network for inferencing. This enables inferencing acceleration with sparsity.

For training acceleration, sparsity needs to be introduced early in the process to offer a performance benefit, and methodologies for training acceleration without accuracy loss are an active research area.

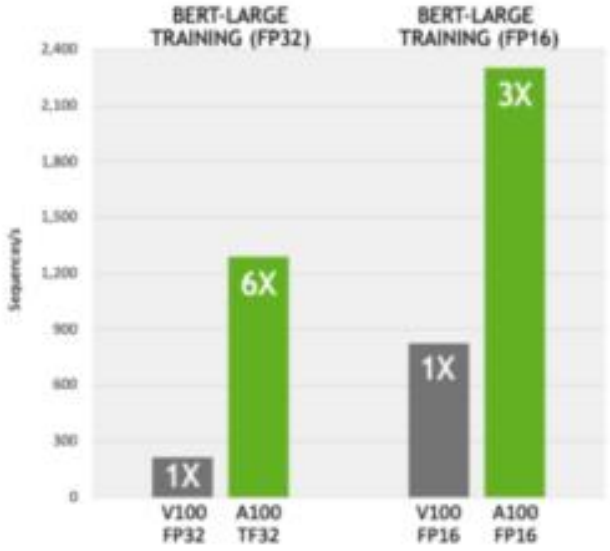
Sparse matrix definition

Structure is enforced through a new 2:4 sparse matrix definition that allows two non-zero values in every four-entry vector. A100 supports 2:4 structured sparsity on rows, as shown in Figure 9.

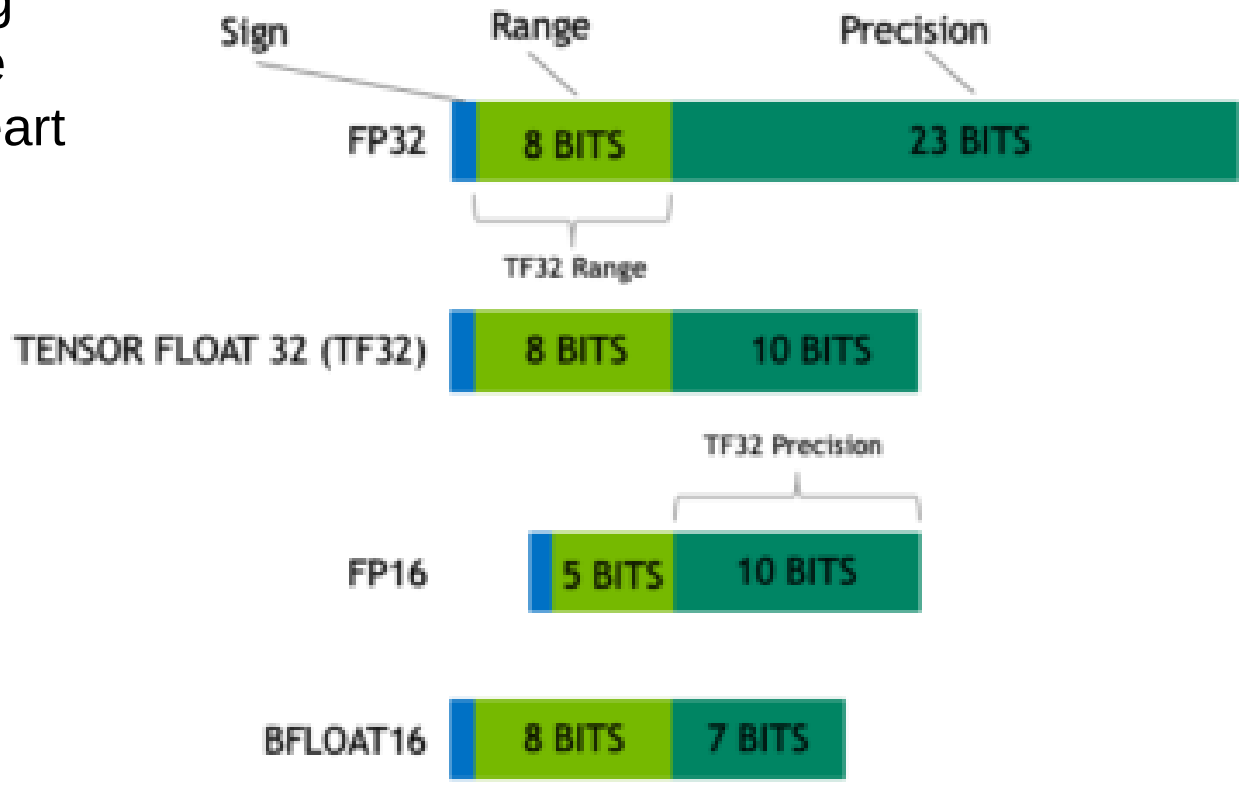
Due to the well-defined structure of the matrix, it can be compressed efficiently and reduce memory storage and bandwidth by almost 2x.

TF32 - Tensor Float arithmatic

TF32 a great alternative to FP32 for crunching through single-precision math, specifically the massive multiply-accumulate functions at the heart of deep learning



BERT Large Training (FP32 & FP16) measures Pre-Training phase, uses PyTorch including (2/3) Phase1 with Seq Len 128 and (1/3) Phase 2 with Seq Len 512, V100 is DGX1 Server with 8xV100, A100 is DGX A100 Server with 8xA100, A100 uses TF32 Tensor Core for FP32 training



https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/

cuDNN 8.9

- NVIDIA CUDA Deep Neural Network (cuDNN) is a GPU-accelerated library of primitives for deep neural networks.
- It provides highly tuned implementations of routines arising frequently in DNN applications.
- Added support for FP8 fused-multi-head attention training and inference support targeting BERT on NVIDIA Hopper GPUs.
- Added support for transformer models training and inference using Flash Attention in cuDNN runtime fusion engine

CPU / No cuDNN

```
Epoch 1/5 1875/1875
[==============================] - 84s
43ms/step - loss: 0.1403 - accuracy: 0.9561
- val_loss: 0.0421 - val_accuracy: 0.9865

Epoch 2/5 1875/1875
[==============================] - 75s
40ms/step - loss: 0.0442 - accuracy: 0.9864
- val_loss: 0.0398 - val_accuracy: 0.9876
```

GPU / cuDNN

```
Epoch 1/5 1875/1875
[==============================] - 24s
6ms/step - loss: 0.1409 - accuracy: 0.9565
- val_loss: 0.0480 - val_accuracy: 0.9845

Epoch 2/5 1875/1875
[==============================] - 11s
6ms/step - loss: 0.0469 - accuracy: 0.9862
- val_loss: 0.0605 - val_accuracy: 0.9798
```

Sample code with cuDNN 8.9

In Colab, Open an existing file that is
~/EDGE-AI/CU-DNN/cuDNN_example.ipynb

Run with cuDNN & NocuDNN & show the performance

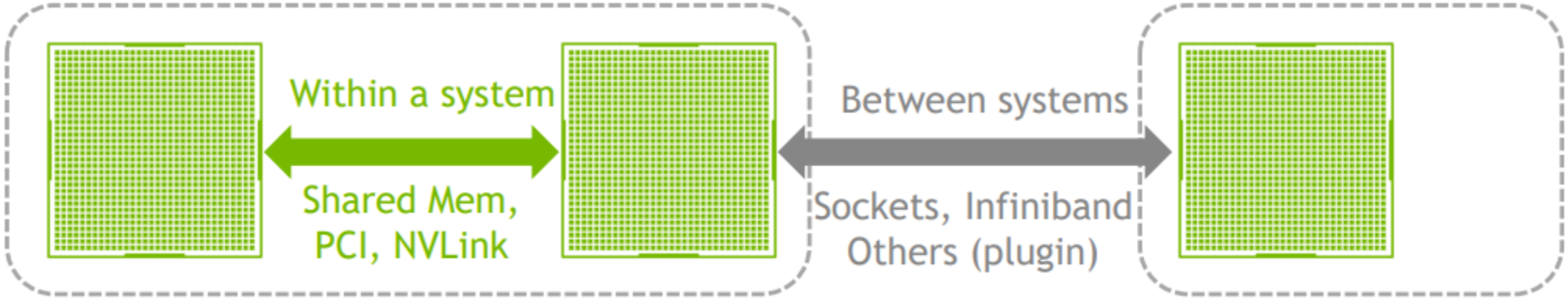
```
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.models import Sequential

# Define a simple CNN model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10))

# Compile the model with cuDNN support
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'],
              experimental_run_tf_function=False) # Enable cuDNN

# Load the dataset and train the model
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
model.fit(train_images, train_labels, epochs=5, validation_data=(test_images, test_labels))
```

True when
No cuDNN

False when
cuDNN
enabled

INTER-GPU COMMUNICATION

Intra-node and Inter-node



GPU specifications

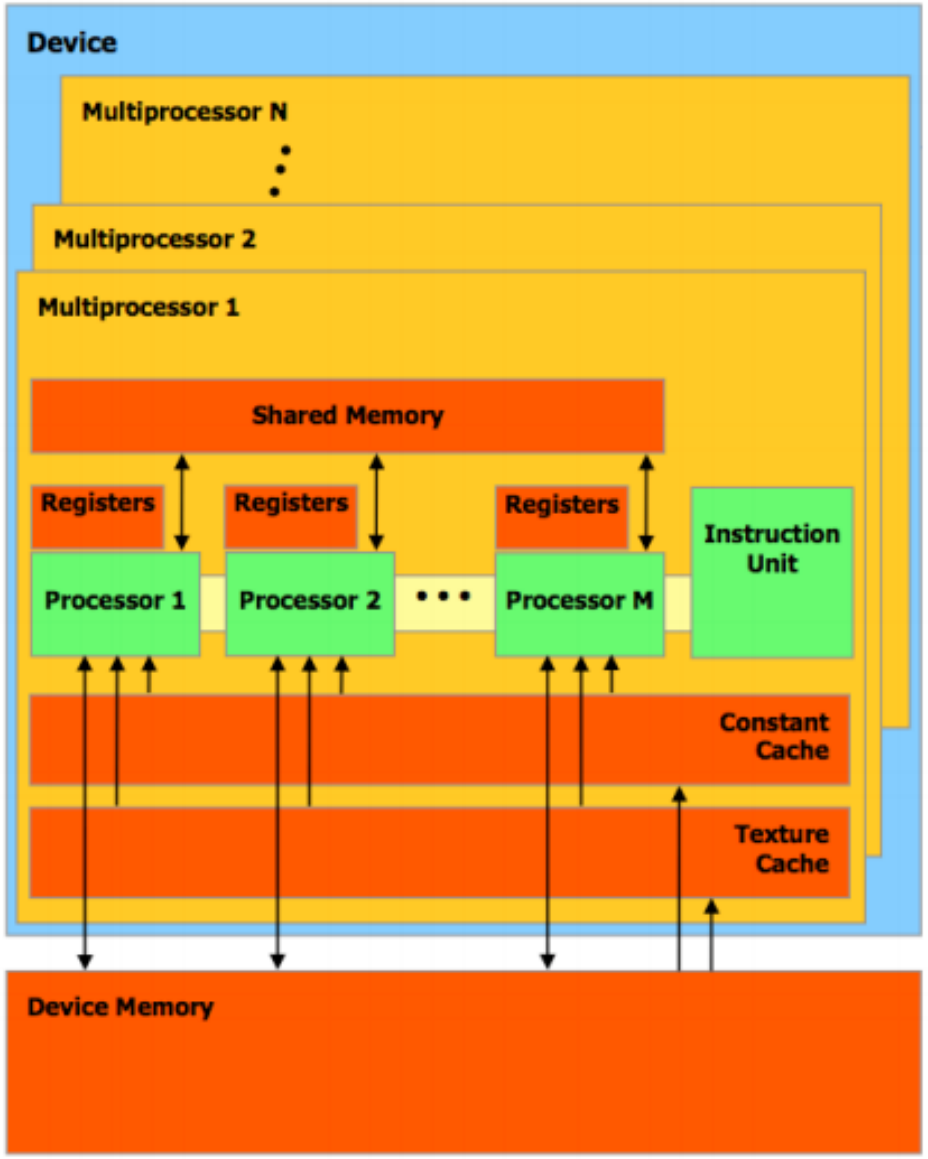
Graphics Card	GeForce RTX 2080 Founders Edition	GeForce RTX 2080 Super Founders Edition	GeForce RTX 3080 10 GB Founders Edition
GPU Codename	TU104	TU104	GA102
GPU Architecture	NVIDIA Turing	NVIDIA Turing	NVIDIA Ampere
GPCs	6	6	6
TPCs	23	24	34
SMs	46	48	68
CUDA Cores / SM	64	64	128
CUDA Cores / GPU	2944	3072	8704
Tensor Cores / SM	8 (2nd Gen)	8 (2nd Gen)	4 (3rd Gen)
Tensor Cores / GPU	368	384 (2nd Gen)	272 (3rd Gen)
RT Cores	46 (1st Gen)	48 (1st Gen)	68 (2nd Gen)
GPU Boost Clock (MHz)	1800	1815	1710
Peak FP32 TFLOPS (non-Tensor)[1]	10.6	11.2	29.8
Peak FP16 TFLOPS (non-Tensor)[1]	21.2	22.3	29.8
Peak BF16 TFLOPS (non-Tensor)[1]	NA	NA	29.8
Peak INT32 TOPS (non-Tensor)[1,3]	10.6	11.2	14.9
Peak FP16 Tensor TFLOPS with FP16 Accumulate[1]	84.8	89.2	119/238[2]
Peak FP16 Tensor TFLOPS with FP32 Accumulate[1]	42.4	44.6	59.5/119[2]
Peak BF16 Tensor TFLOPS with FP32 Accumulate[1]	NA	NA	59.5/119[2]
Peak TF32 Tensor TFLOPS[1]	NA	NA	29.8/59.5[2]
Peak INT8 Tensor TOPS[1]	169.6	178.4	238/476[2]

https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf

GPU specifications

Graphics Card	GeForce RTX 2080 Founders Edition	GeForce RTX 2080 Super Founders Edition	GeForce RTX 3080 10 GB Founders Edition
Peak INT4 Tensor TOPS[1]	339.1	356.8	476/952[2]
Frame Buffer Memory Size and Type	8192 MB GDDR6	8192 MB GDDR6	10240 MB GDDR6X
Memory Interface	256-bit	256-bit	320-bit
Memory Clock (Data Rate)	14 Gbps	15.5 Gbps	19 Gbps
Memory Bandwidth	448 GB/sec	496 GB/sec	760 GB/sec
ROPs	64	64	96
Pixel Fill-rate (Gigapixels/sec)	115.2	116.2	164.2
Texture Units	184	192	272
Texel Fill-rate (Gigatexels/sec)	331.2	348.5	465
L1 Data Cache/Shared Memory	4416 KB	4608 KB	8704 KB
L2 Cache Size	4096 KB	4096 KB	5120 KB
Register File Size	11776 KB	12288 KB	17408 KB
TGP (Total Graphics Power)	225 W	250 W	320W
Transistor Count	13.6 Billion	13.6 Billion	28.3 Billion
Die Size	545 mm^2	545 mm^2	628.4 mm^2
Manufacturing Process	TSMC 12 nm FFN (FinFET NVIDIA)	TSMC 12 nm FFN (FinFET NVIDIA)	Samsung 8 nm 8N NVIDIA Custom Process

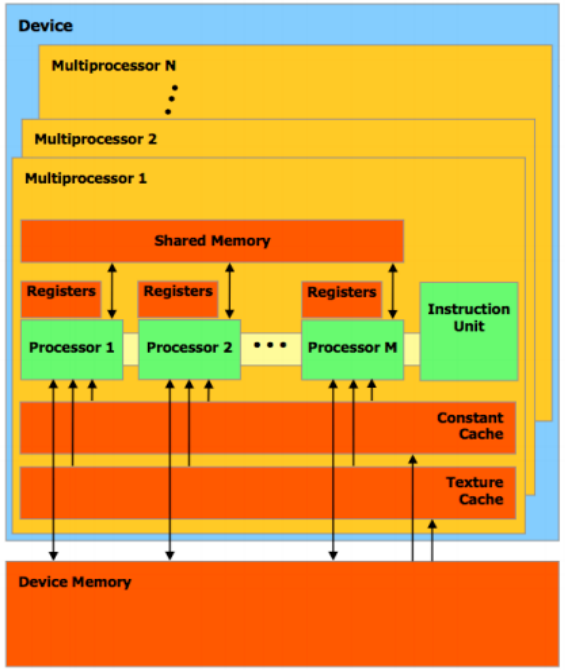
CUDA memory types



- Shared memory partitioned amongst Thread Blocks resident on the Streaming Multiprocessors
- Registers are partitioned amongst Threads

Types of Memory

- Data stored in register memory is visible only to the thread that wrote it and lasts only for the lifetime of that thread
- Local memory has the same scope rules as register memory, but performs slower
- Data stored in shared memory is visible to all threads within that block and lasts for the duration of the block. This is invaluable because this type of memory allows for threads to communicate and share data between one another
- Data stored in global memory is visible to all threads within the application (including the host), and lasts for the duration of the host allocation compared to global memory



Types of Memory

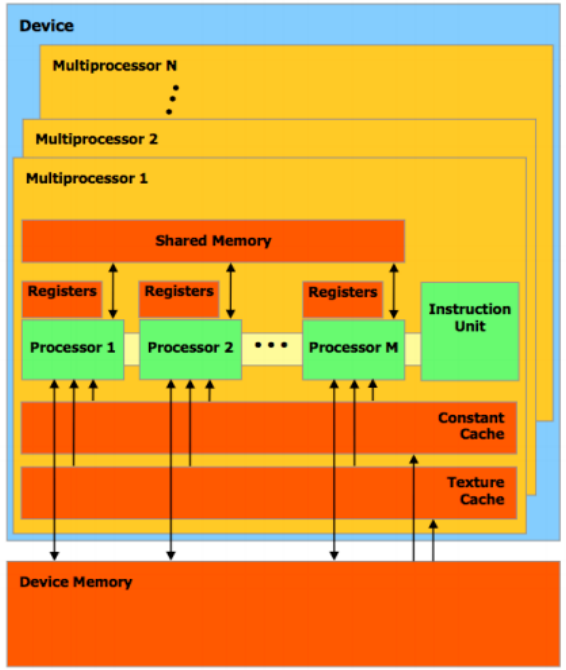
Constant and texture memory for special applications

Constant memory is used for data that will not change over the course of a kernel execution and is read only

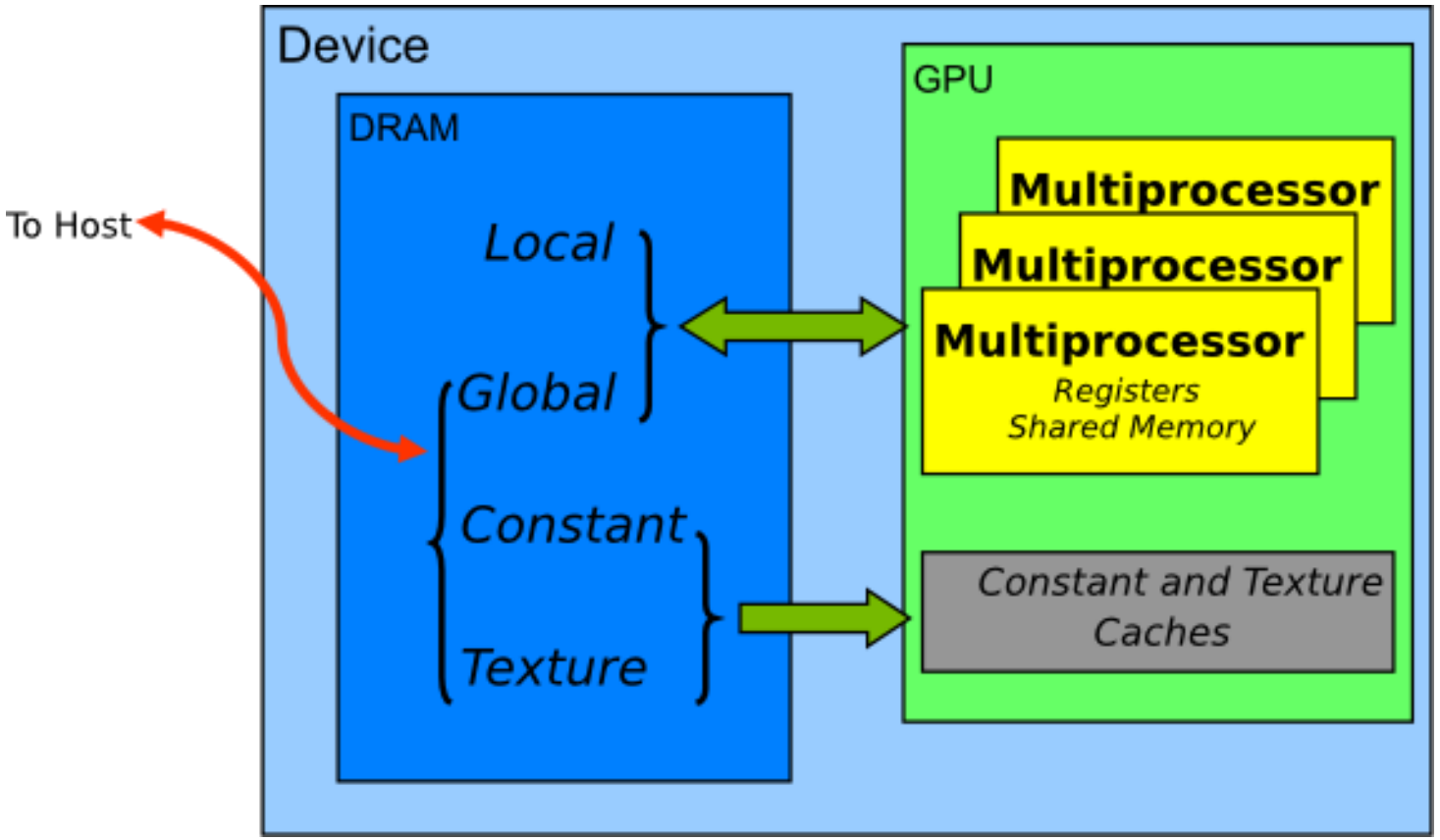
- Using constant rather than global memory can reduce the required memory bandwidth, however, this performance gain can only be realized when a warp of threads read the same location.

Texture memory is another variety of read-only memory on the device

- When all reads in a warp are physically adjacent, using texture memory can reduce memory traffic and increase performance compared to global memory.



Memory spaces



Memory spaces, which have different characteristics that reflect their distinct usages in CUDA applications

Software/Library/Programming

FBLearner · UBER Michelangelo · Amazon SageMaker · Determined AI · Neptune · H2O · FLOYD · DOMINO DATA LAB

"All-in-one"

Data
- Versioning: Quilt, DVC, Pachyderm
- Labeling: figure eight, scale, prodigy
- Workflow: Luigi, TRIFACTA
- Database
- Storage: NFS, S3
- or

Development
- Distributed Training: Distributed TensorFlow
- Frameworks: PyTorch, Keras, fast.ai
- Software Engineering: jupyter, git, Vim

Training/Evaluation
- Hyperparam Optimization: SIGOPT
- Experiment Management: comet, Weights & Biases, TensorBoard, Losswise
- Resource Management: RiseML, slurm
- or

Deployment
- Monitoring
- Hardware / Mobile: NVIDIA TensorRT, TensorFlow Lite
- Web: ALGORITHMIA
- Interchange: ONNX
- CI / Testing: Buildkite
- or or

ING

Transfer Learning

- **CenterNet** (2019) is an object detection architecture based on a deep convolution neural network trained to detect each object as a triplet (rather than a pair) of keypoints, so as to improve both precision and recall
- **EfficientDet** (2019) is an object detection architecture built to scale up model efficiency in computer vision. This architecture achieves much better efficiency than prior architectures across a wide spectrum of resource constraints
- **MobileNet** is an object detector released in 2017 as an efficient CNN architecture designed for mobile and embedded vision application. This architecture uses proven depth-wise separable convolutions to build lightweight deep neural networks for mobile and embedded vision applications.
    - MobileNets are based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks.

Huang et al.[9]

Transfer Learning

- **RetinaNet** is an architecture developed by the Facebook research team in 2018. RetinaNet uses a Feature Pyramid Network (FPN) backbone on top of a feed-forward ResNet architecture to generate a rich, multi-scale convolutional feature pyramid. It is a one-staged detector (that is, a single network, unlike R-CNN, which is 2-staged).
- **R-CNN** (2014) is a 2-stage object detection architecture. It is a region-based CNN that uses a Region Proposal Network to generate regions of interests in the first stage, and then sends the region proposal down the pipeline for object classification and bounding box regression.
- **ExtremeNet** (2019) is a bottom-up object detection framework that detects four extreme points (top-most, left-most, bottom-most, right-most) of an object to find extreme points, by predicting four multi-peak heatmaps for each object category.

Huang et al.[9]

Books

- "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, 2021, MIT Press

Thank you !

AI Accelerated Devices

Day 2, Session 2A

sambath.narayanan@gmail.com