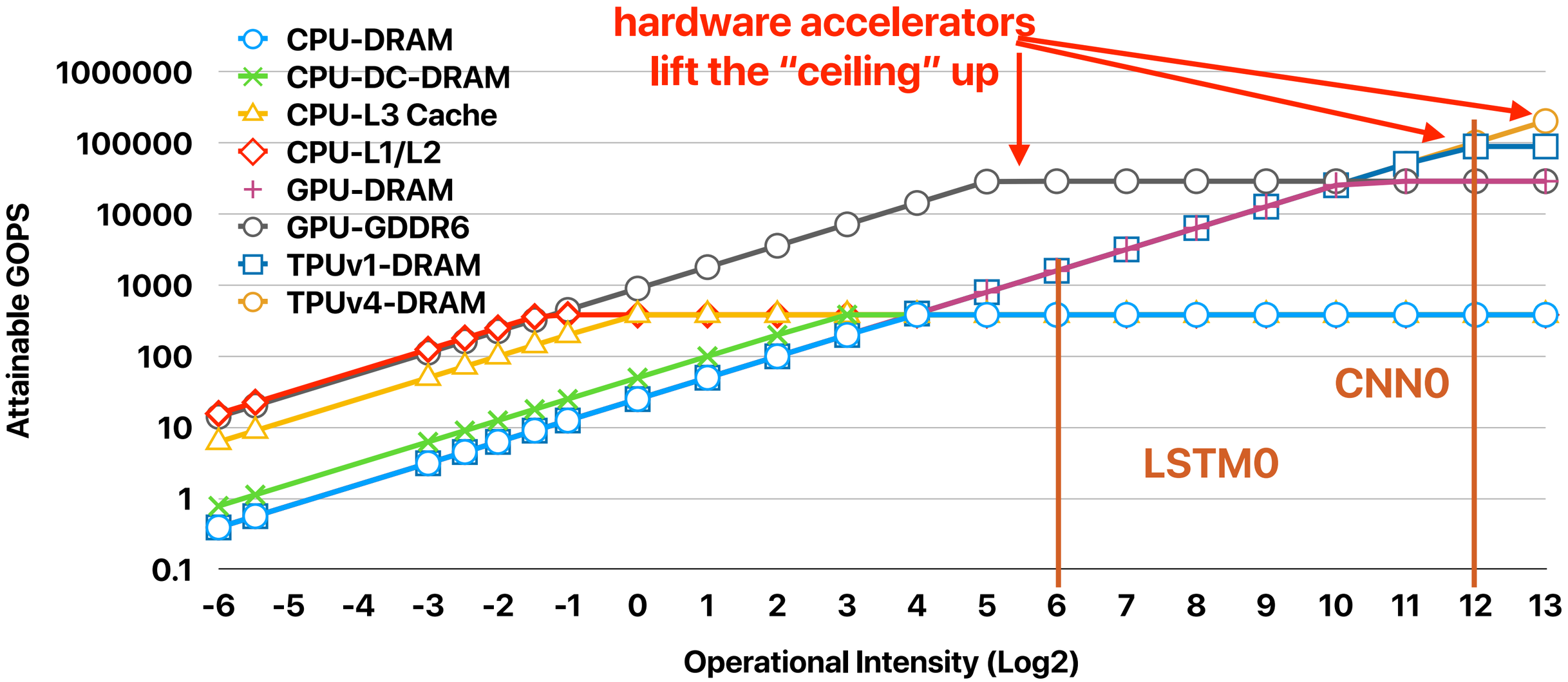
Raise the roofline — hardware accelerators (2)

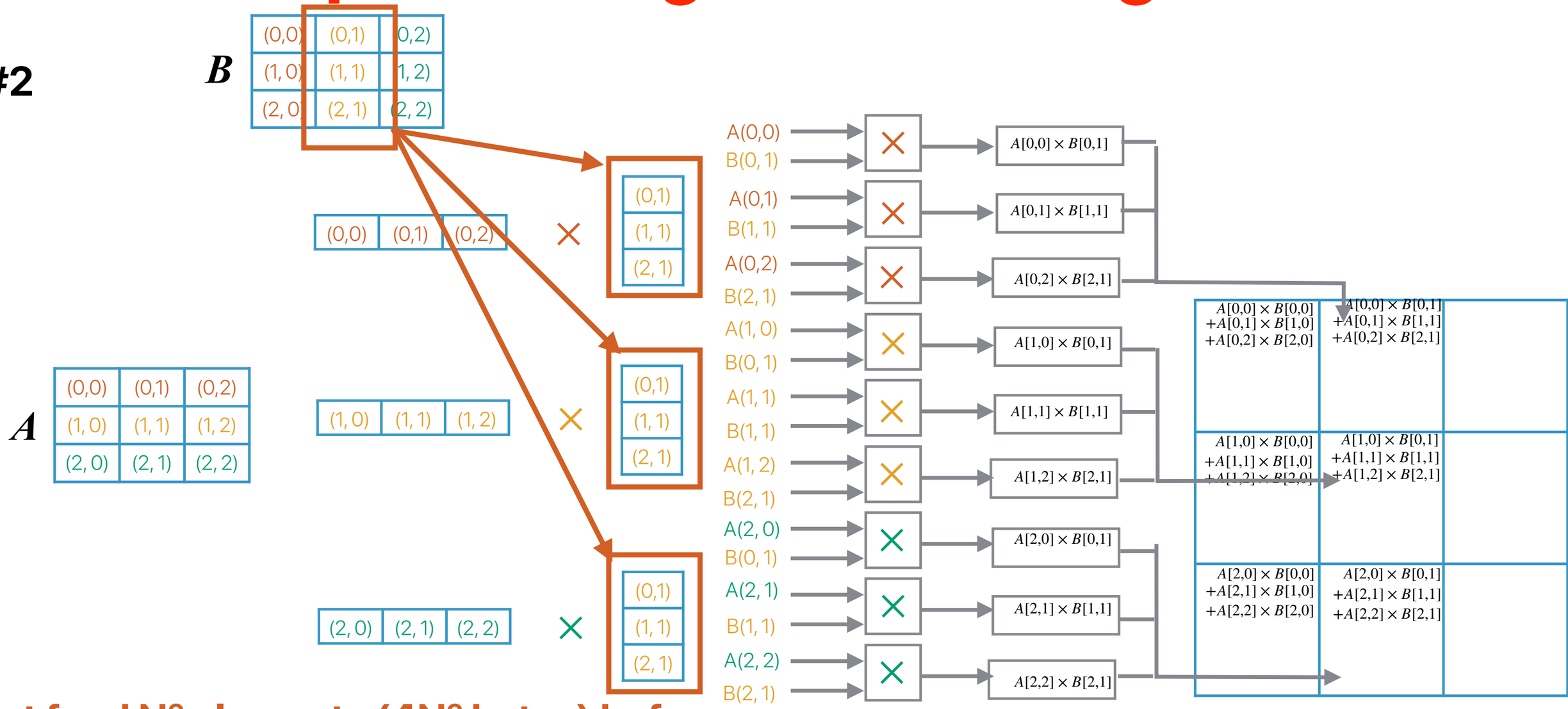
Hung-Wei Tseng

Recap: the rooflines of modern systems



Vector processing for MM using N^2 PEs

#2



Must feed N^2 elements ($4N^2$ bytes) before each vector MAC operation

Systolic Arrays used by AI/ML Accelerators

#3

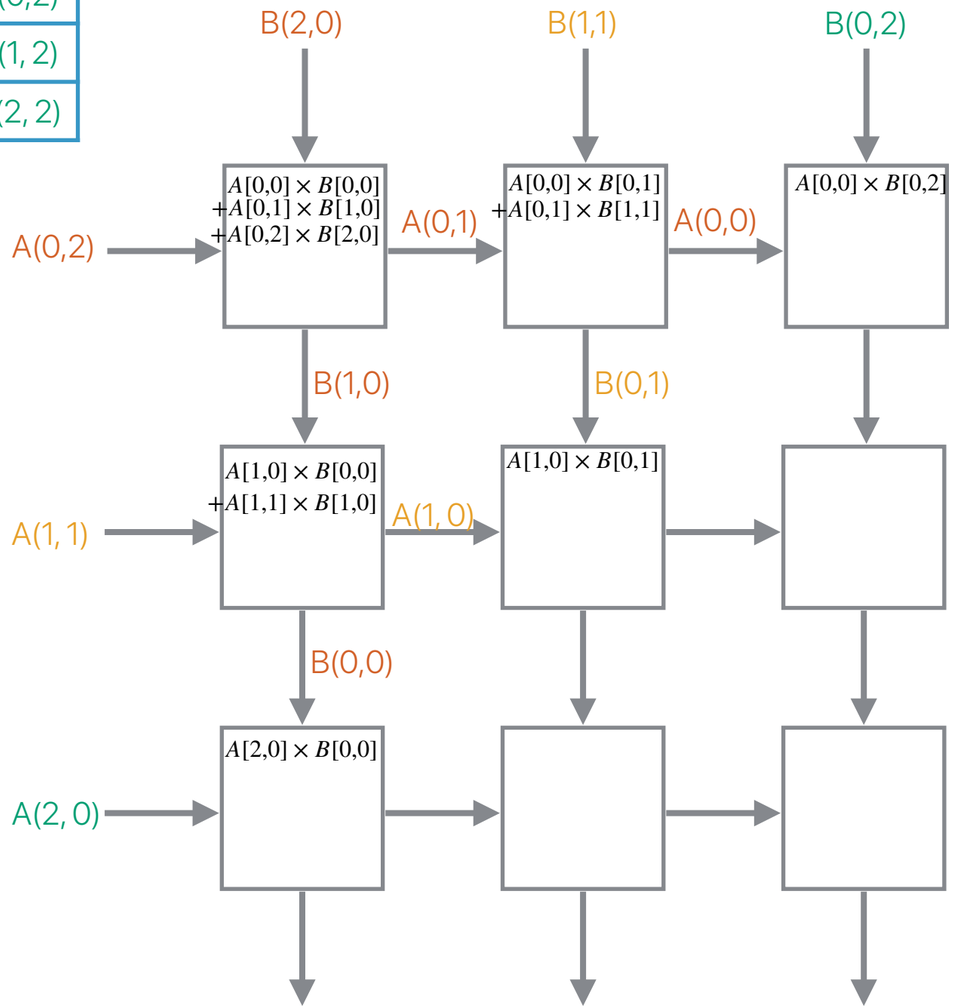
B

(0,0)	(0,1)	(0,2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)

Only need to fetch 2N elements (8N bytes) before we start

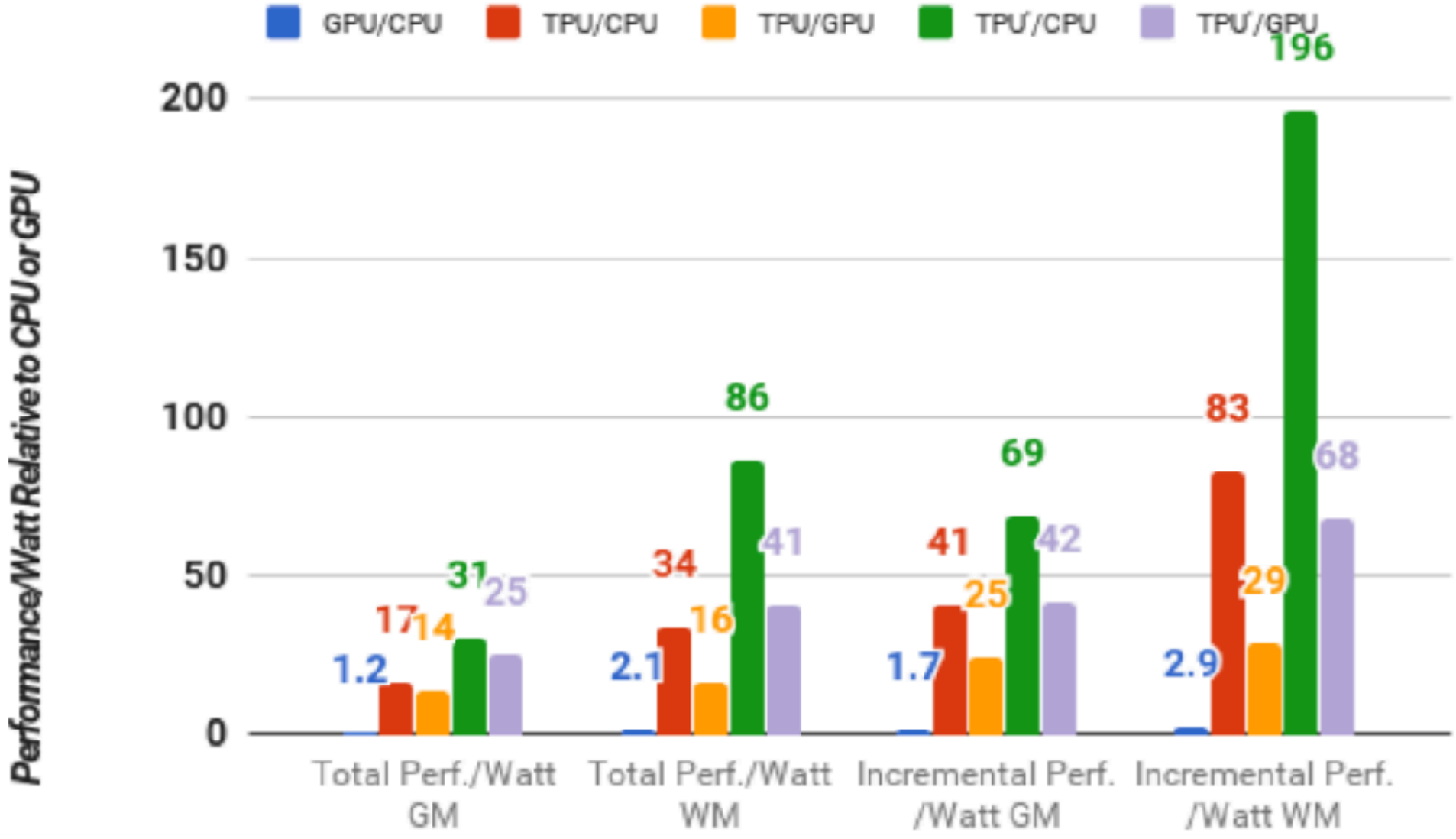
A

(0,0)	(0,1)	(0,2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)



H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Energy efficiency is better! — reduced cost!



3:00

Do you think TPU is a good enough design for AI/ML applications?

Is TPU a good enough design?

Outline

- Recent development of TPUs
- Ray Tracing Accelerators
- General-purpose approximate computing on NPUs

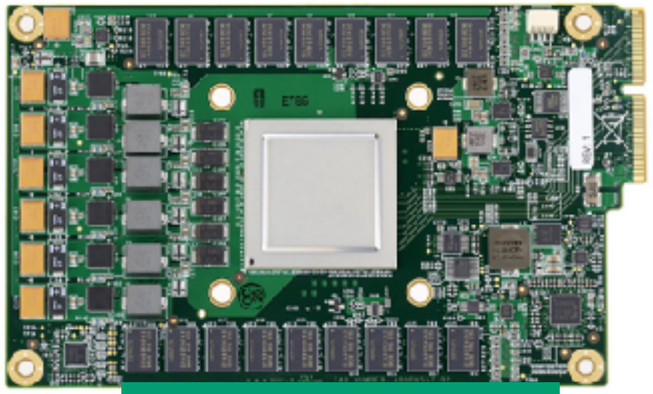
10 lessons learned from 3 generations of Google TPUs

- Logic, wires, SRAM & DRAM improve unequally
- Leverage prior compiler optimizations
- Design for performance per TCO vs CapEx
- Backwards ML compatibility
- Inference DSAs need air cooling for global scale
- Some inference apps need floating point arithmetic
- Production inference normally needs multi-tenancy
- DNNs grow ~1.5x/year in memory and compute
- DNN workloads evolve with DNN breakthroughs
- Inference SLO limit is P99 latency, not batch size

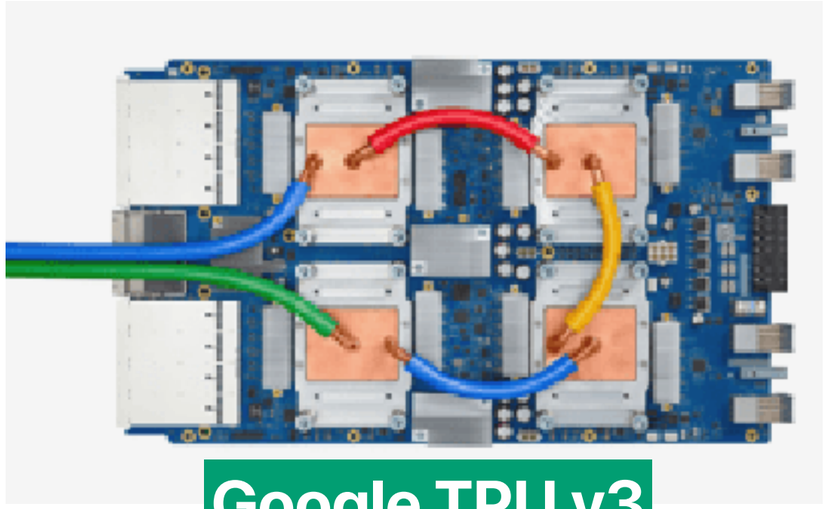
N. P. Jouppi *et al.*, "Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product," *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Valencia, Spain, 2021, pp. 1-14, doi: 10.1109/ISCA52012.2021.00010.

Recent advancement of TPUs

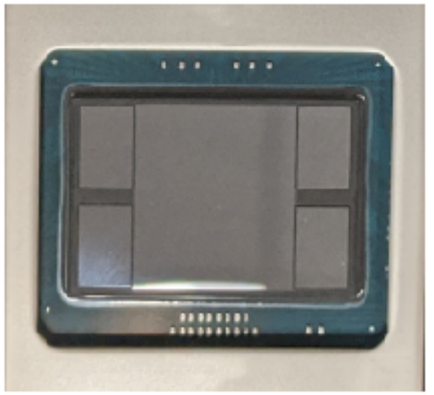
Cloud TPUs

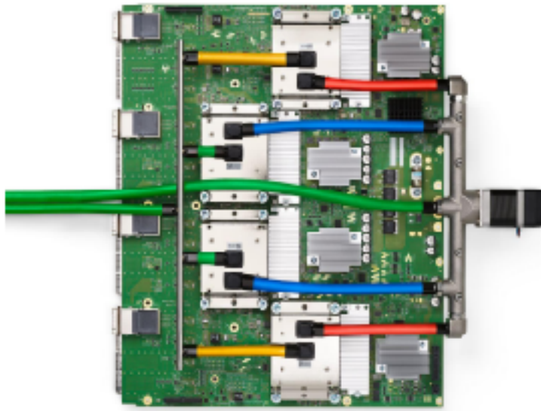


Google TPU v1

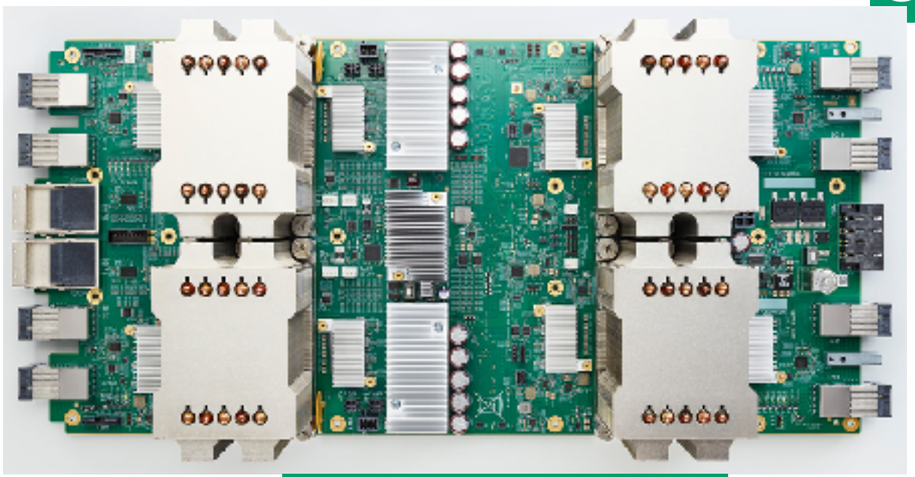


Google TPU v3

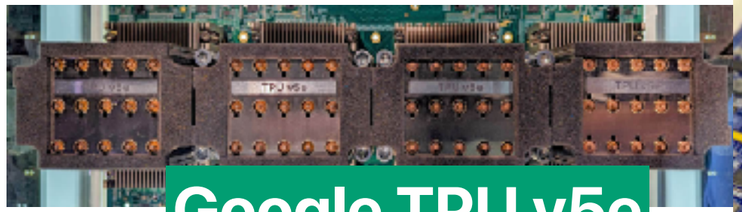




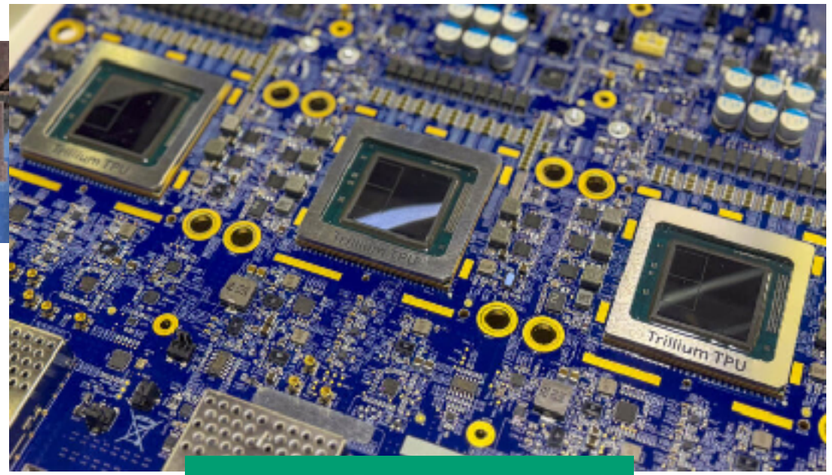
Google TPU v4



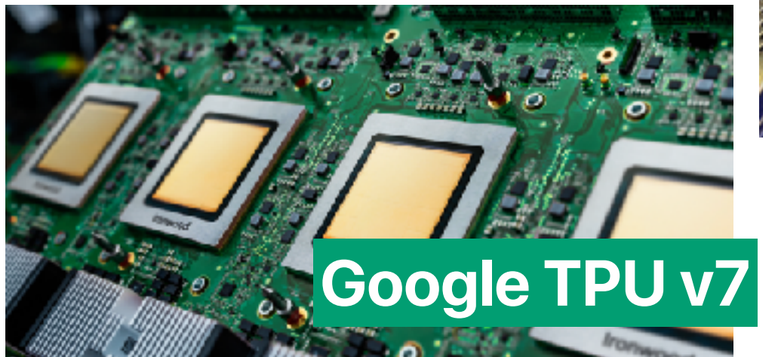
Google TPU v2



Google TPU v5e



Google TPU v6



Google TPU v7

TPUs on smaller devices

Edge TPU

Google Tensor G1/G2/G3

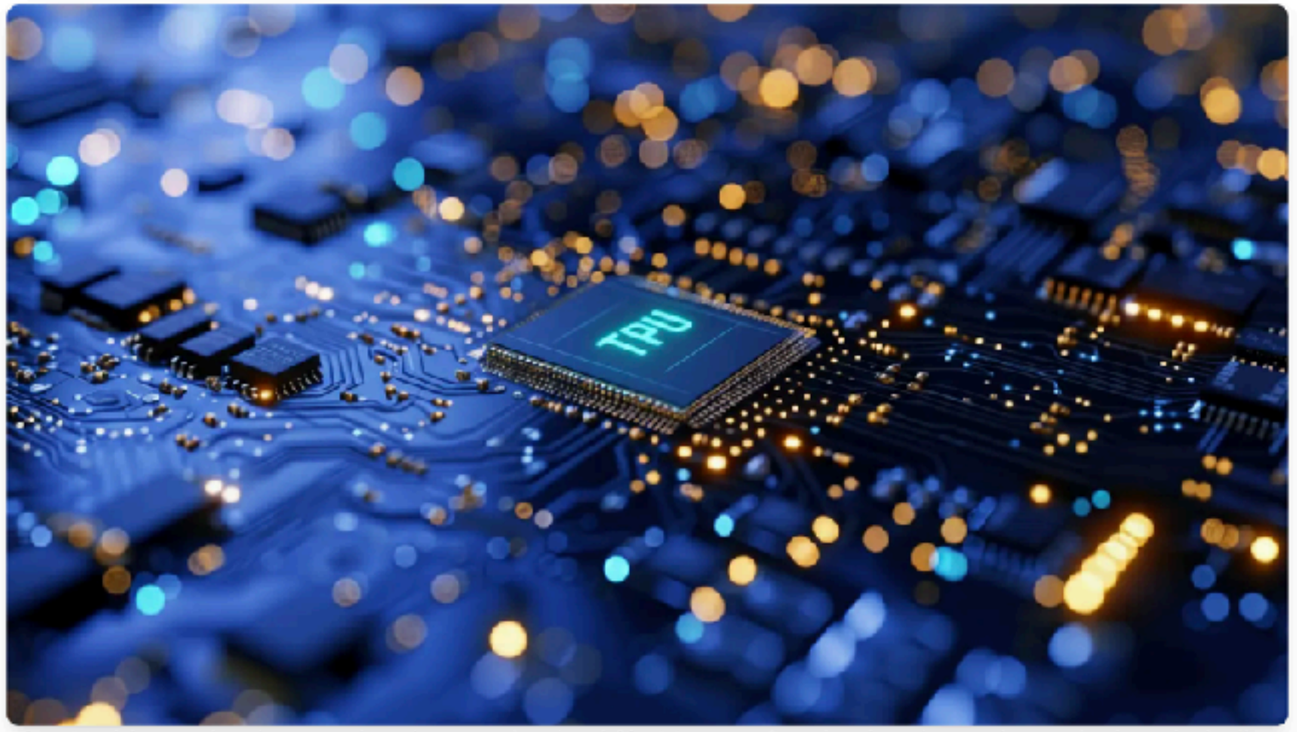
7 generations of TPUs

	TPU v1	TPU v2	TPU v3	TPU v4	TPU v5 e/p	TPU v6e	TPU v7
Announced	2015	2017	2018	2020	2023	2024	2025
Target	Inference	Training + Inference	Training	Training + Inference	Inference(v5e);Training(v5p)	Training + Inference	Inferencing (LLM)
Peak Performance Per Chip	92 TOPS	45 TFLOPS	420 TFLOPS	1000 TFLOPS	393 TOPS (Int8); 459 TFLOPS (v5p)	~2000 TOPS	4614 TFLOPS
On-chip Memory	28 MB	8 GB	32 GB/chip; 128 GB/board	≥128 GB per board	95 GB	2× v5 HBM capacity	192 GB per chip
Memory bandwidth (GB/s)	167	600	900	1200	2765	1638	7380
Interconnect	PCIe	4 chips per board; 64 per pod	1024+ per pod	4096 per pod with Optical Circuit Switches (OCS)	256-chip pod	Jupiter fabric: 100K chips/pod	~9,216 chips/ pod
Power (W)	40	200-280	123-262	125-250	200-300	300	600
Features	256×256 CPU-style systolic array	Introduced b float16 training, liquid cooling	Further liquid cooling, higher memory.	Architecture built around 5-D torus + re-configurable optical fabric; Adds SparseCores: ~5% of die foembedding	v5e (inference-optimized, eight "shapes" supporting up to 2T parameter models; v5p (training- oriented)	Trained Google Gemini 2.0; Major leap: 4.7× peak perf per chip over v5, 2× memory and interface bandwidth, 67% energy efficiency gain	First TPU "inference-first" design

← Back to all posts

Google's TPU Revolution: The $13 Billion Challenge to Nvidia's AI Chip Dominance

December 2, 2025 10 min read



The AI chip market is experiencing a seismic shift. Morgan Stanley has just released a bombshell forecast: **Google's Tensor Processing Unit (TPU) production could reach 7 million units by 2028,** potentially injecting $13 billion in new revenue and adding $0.40 to earnings per share. But this isn't just a story about Google's growth—it's about a fundamental challenge to Nvidia's

3:00

Compare GPU/TPU designs, who is more competitive in the future (or say, do you think NVIDIA have a bleak future)?

RTX on—The NVIDIA Turing GPU

3:00

Why RTX architecture?

Turing architecture — specialized for key workloads

- CUDA cores for vector processing
- Tensor cores for AI
- RT cores for VR/AR

> While Turing is packed with features and horsepower,[1] we made fundamental advancements in several key areas—streaming multiprocessor (SM) efficiency, a Tensor Core for AI inferencing, and an RTCore for ray-tracing acceleration.

Tensor Cores

Tensor cores for deep learning

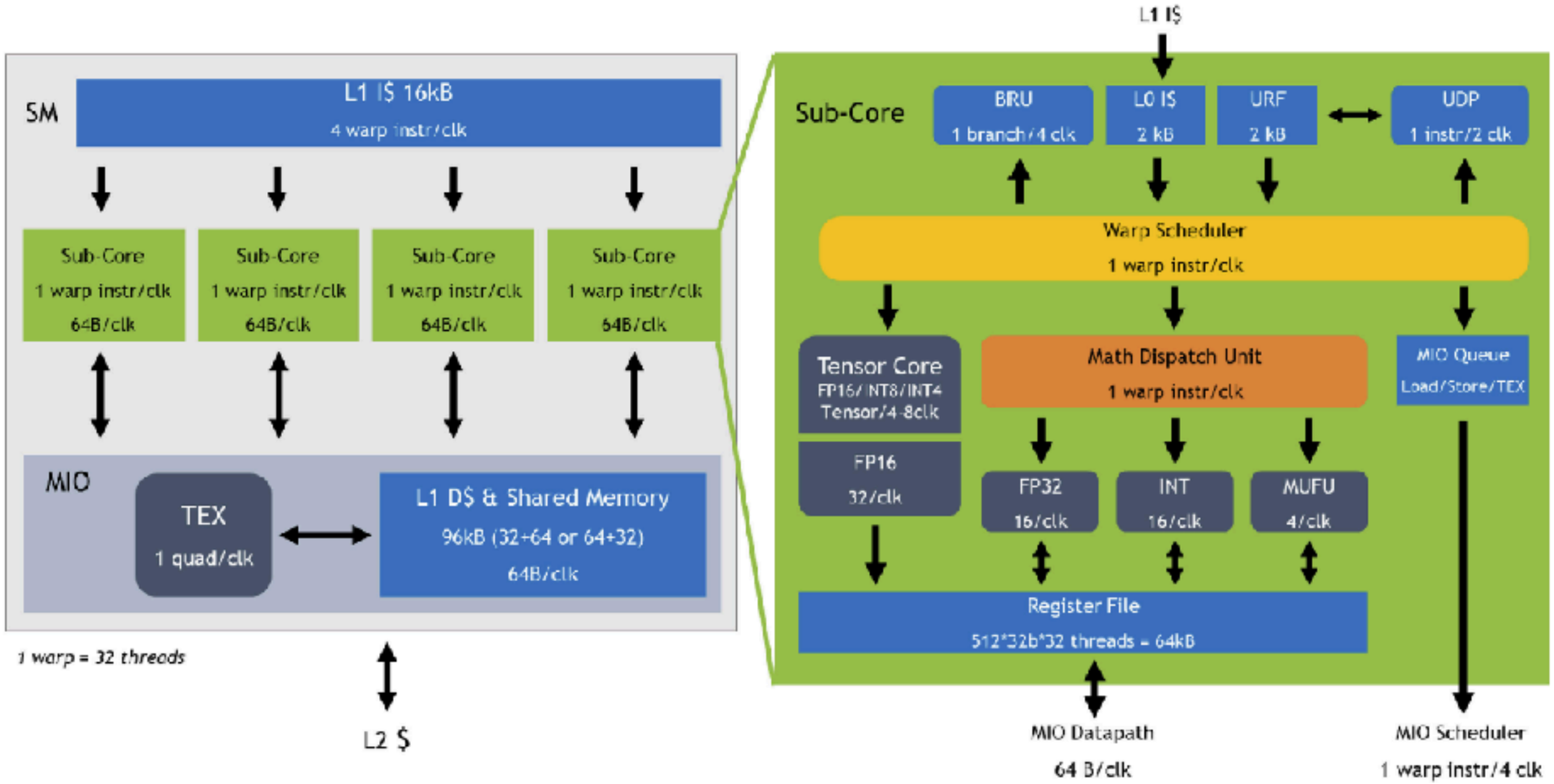


Figure 1. Turing GPU SM, comprising four subcores and a memory interface (MIO). Math throughput, memory bandwidth, L1 data cache topology, register file and cache capacity, and a new uniform datapath were all designed or modified to increase processor efficiency over the previous generation.

Turing Architecture

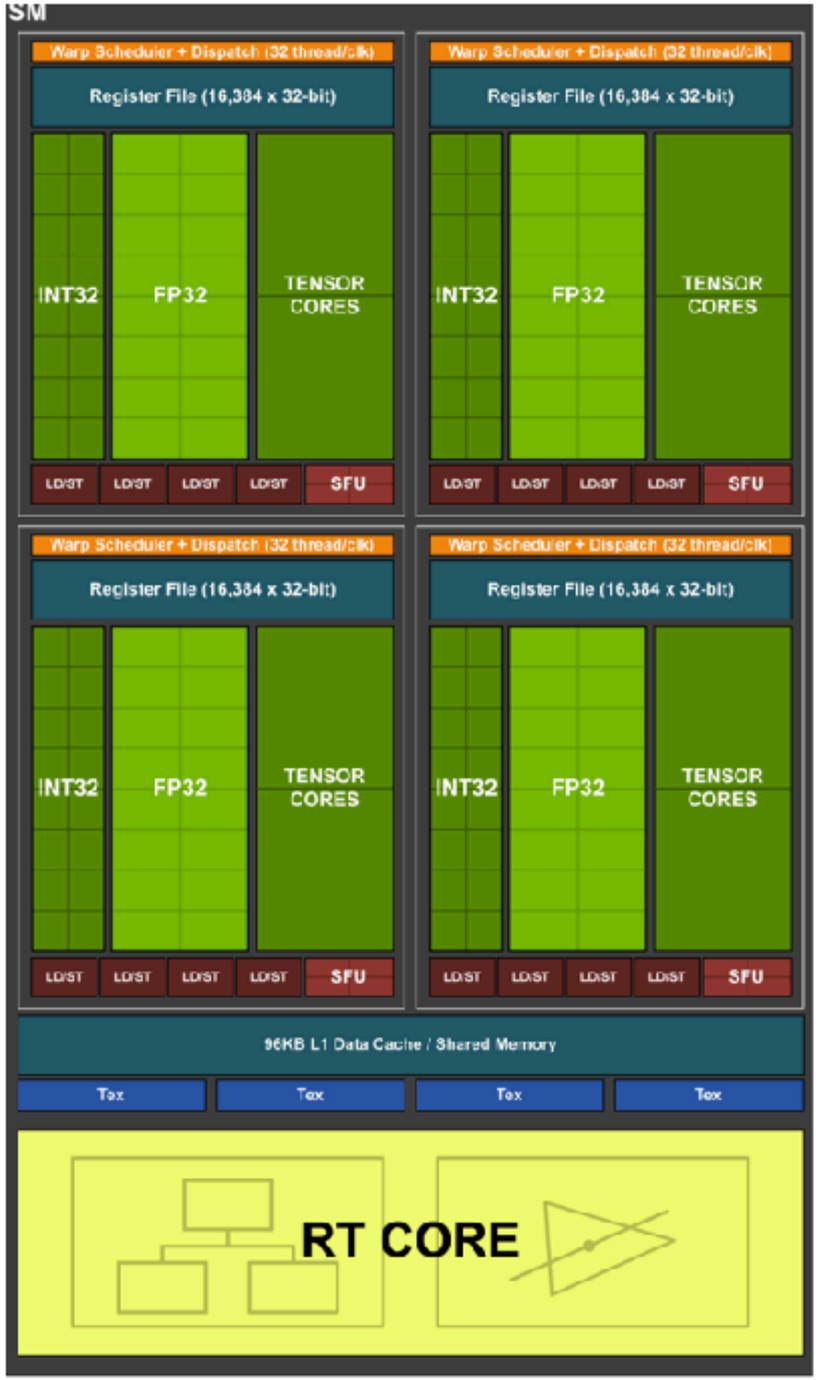


Figure 4. Turing TU102/TU104/TU106 Streaming Multiprocessor (SM)

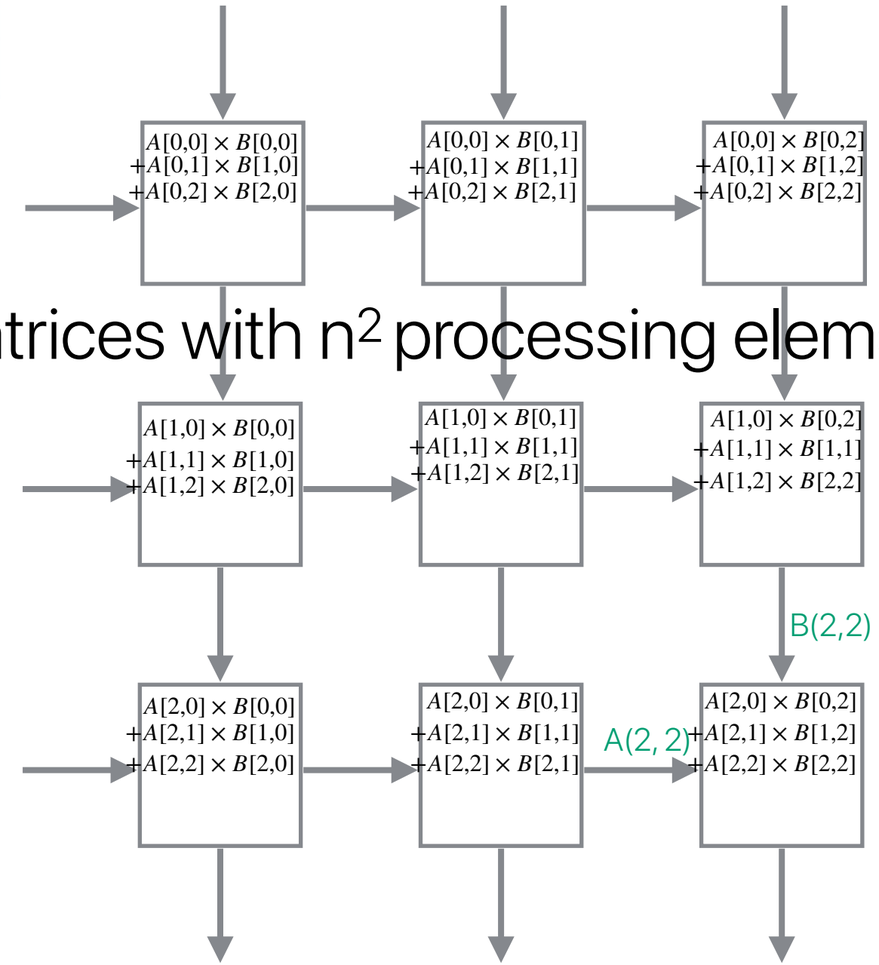
NVIDIA, "NVIDIA Turing GPU architecture: Graphics reinvented," 2018. [Online]. Available: https://www. nvidia.com/content/dam/en-zz/Solutions/designvisualization/ technologies/ turing-architecture/NVIDIATuring- Architecture-Whitepaper.pdf

Systolic Arrays used by AI/ML Accelerators

#7

B

(0,0)	(0,1)	(0,2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)



For n^2 matrices with n^2 processing elements: $3 \times n - 2$

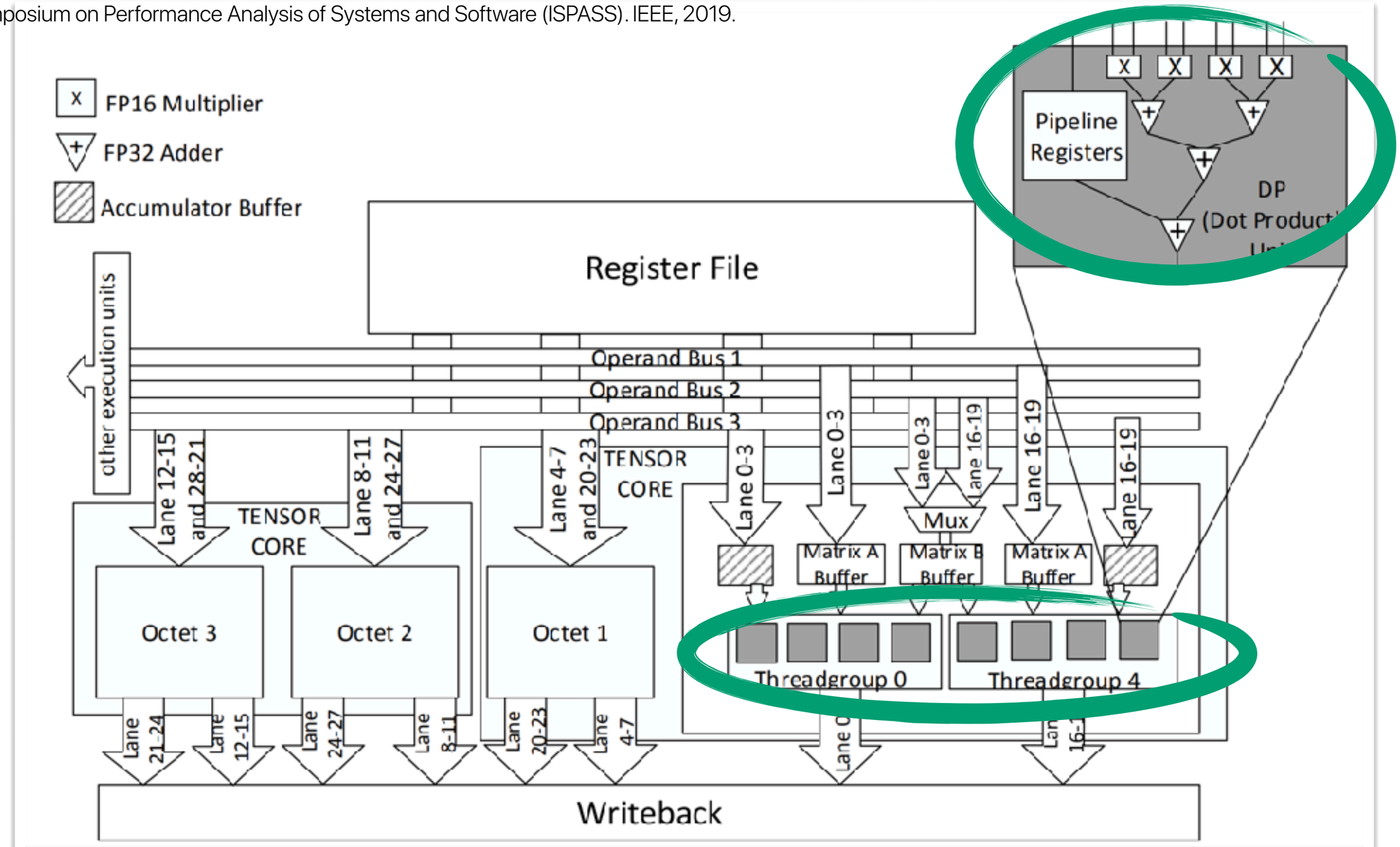
A

(0,0)	(0,1)	(0,2)
(1, 0)	(1, 1)	(1, 2)
(2, 0)	(2, 1)	(2, 2)

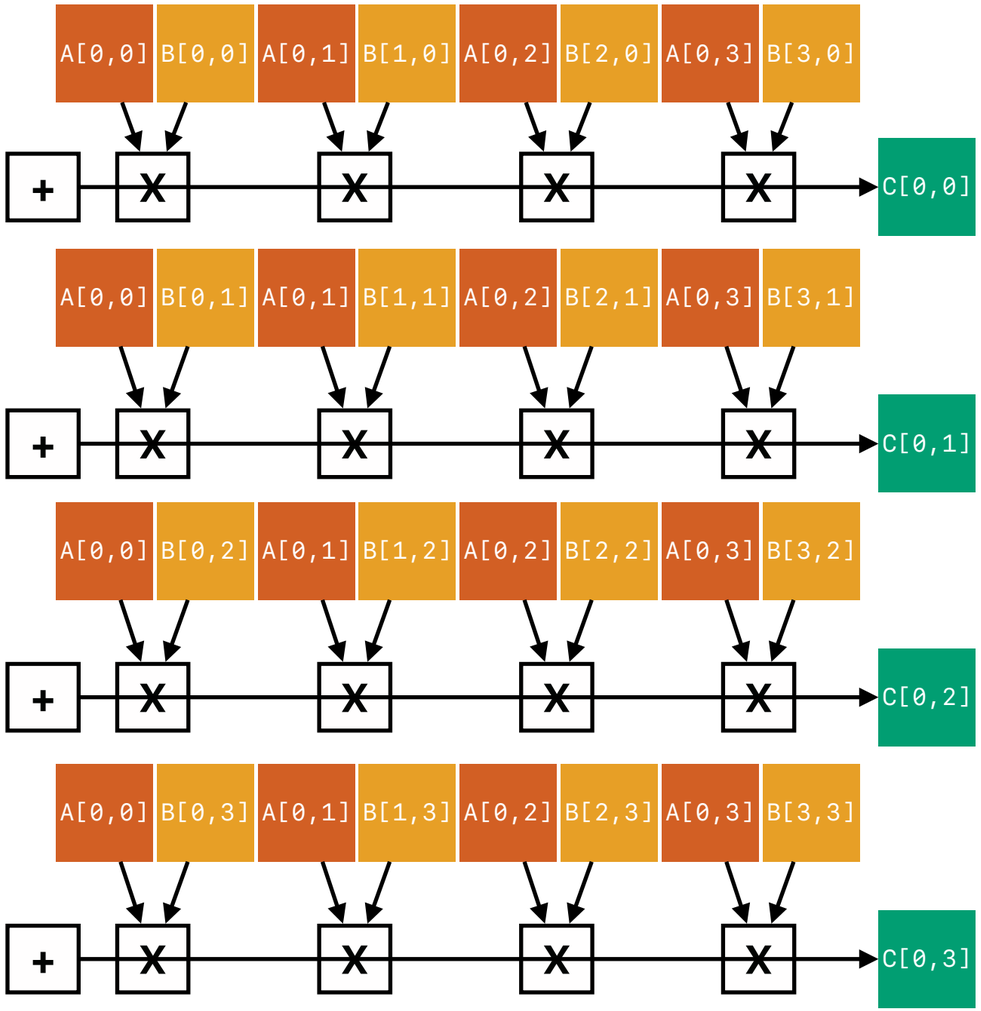
H.T. Kung, "Why systolic architectures?," in IEEE Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

Inside tensor cores

Raihan, Md Aamir, Negar Goli, and Tor M. Aamodt. "Modeling deep learning accelerator enabled gpus." 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2019.



Tensor Cores



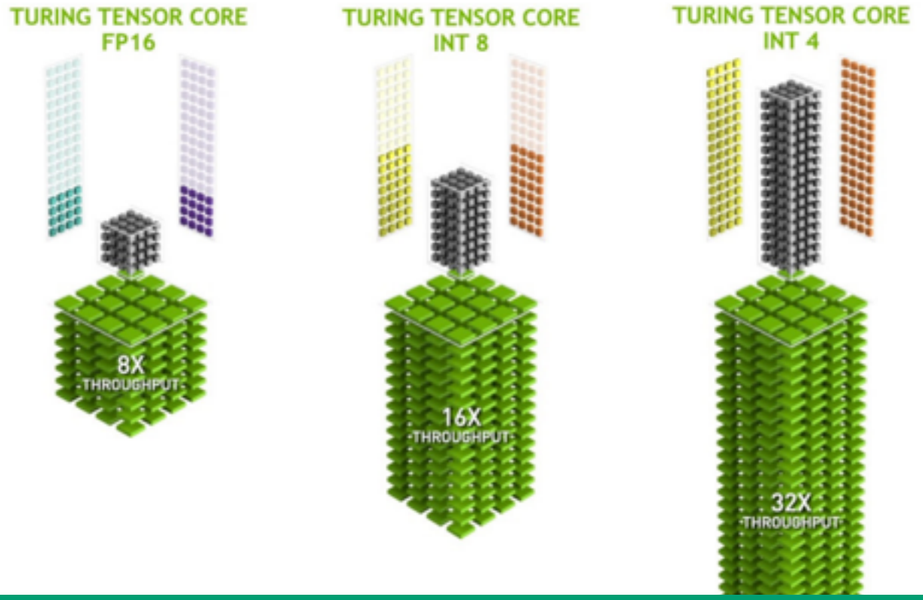
A[0,0]	A[0,1]	A[0,2]	A[0,3]
A[1,0]	A[1,1]	A[1,2]	A[1,3]
A[2,0]	A[2,1]	A[2,2]	A[2,3]
A[3,0]	A[3,1]	A[3,2]	A[3,3]

B[0,0]	B[0,1]	B[0,2]	B[0,3]
B[1,0]	B[1,1]	B[1,2]	B[1,3]
B[2,0]	B[2,1]	B[2,2]	B[2,3]
B[3,0]	B[3,1]	B[3,2]	B[3,3]

C[0,0]	C[0,1]	C[0,2]	C[0,3]
C[1,0]	C[1,1]	C[1,2]	C[1,3]
C[2,0]	C[2,1]	C[2,2]	C[2,3]
C[3,0]	C[3,1]	C[3,2]	C[3,3]

Efficiency of Tensor Cores

The NVIDIA Tesla T4 GPU includes 2,560 CUDA Cores and 320 Tensor Cores, delivering up to 130 TOPs (Tera Operations per second) of INT8 and up to 260 TOPS of INT4 inferencing performance (see Appendix A, *Turing TU104 GPU* for more Tesla T4 specifications). Compared to CPU-based inferencing, the Tesla T4, powered by the new Turing Tensor Cores, delivers up to 40X higher inference performance[6] (see Figure 9).



FLOPS of 8K by 8K matrix multiplications =

$$8192 \times 8192 \times 8192 \times 2$$

$$\frac{8192 \times 8192 \times 8192 \times 2}{2560} = 429496730 \ CUDA \ core \ cycles$$

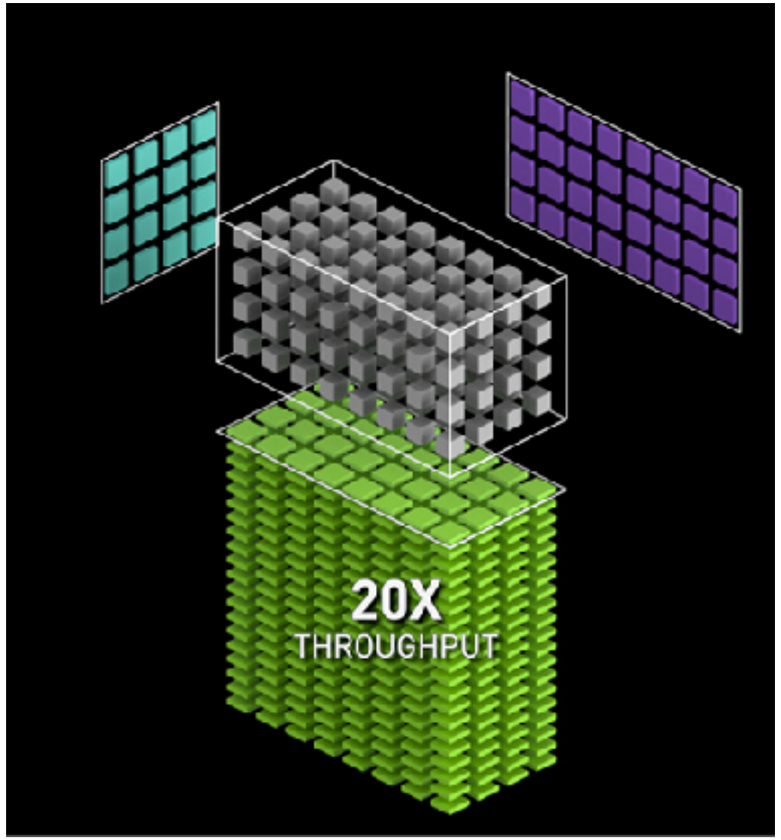
$$\frac{8192 \times 8192 \times 8192 \times 2}{320 \times 128} = 26843546 \ Tensor \ core \ cycles$$

Tensor cores make matrix processing 16x faster

Each tensor core operation performs 4x4x4 MMA in one cycle
~ 128 FP operations in conventional scalar models

NVIDIA, "NVIDIA Turing GPU architecture: Graphics reinvented," 2018. [Online]. Available: https://www. nvidia.com/content/dam/en-zz/Solutions/designvisualization/technologies/turing-architecture/NVIDIATuring- Architecture-Whitepaper.pdf

Efficiency of Tensor Cores (RTX 4090)



	RTX 4090
CUDA Cores	16384
Tensor Cores	512

FLOPS of 8K by 8K matrix multiplications =

$$8192 \times 8192 \times 8192 \times 2$$

$$\frac{8192 \times 8192 \times 8192 \times 2}{16384} = 67{,}108{,}864 \ \textit{CUDA core cycles}$$

$$\frac{8192 \times 8192 \times 8192 \times 2}{512 \times 256} = 8{,}388{,}608 \ \textit{Tensor core cycles}$$

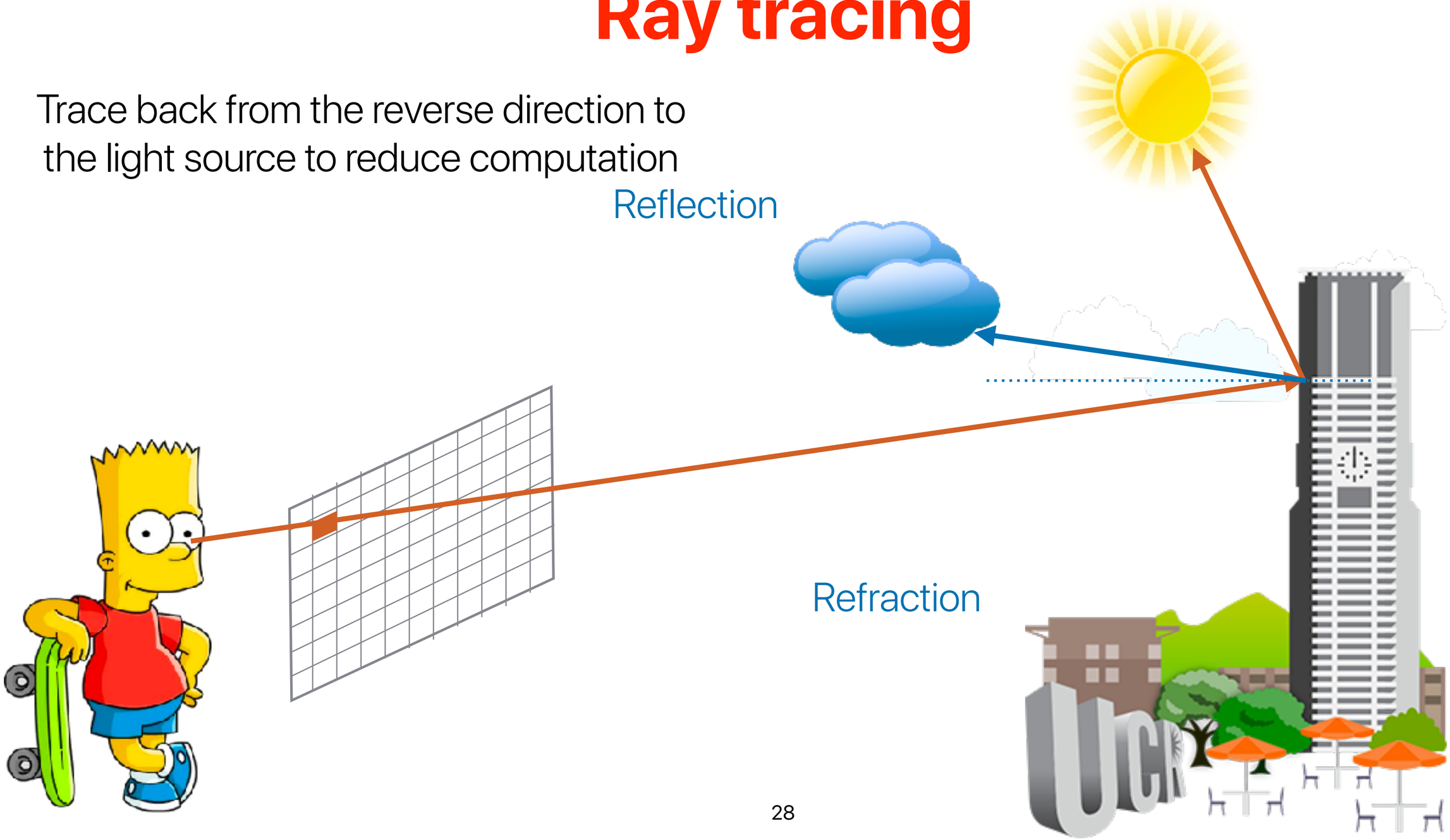
Each tensor core operation performs 8x4x4 MMA in one cycle
~ 256 FP operations in conventional scalar models

Tensor cores make matrix processing 8x faster

RT Cores

Ray tracing

Trace back from the reverse direction to the light source to reduce computation



Simplest ray tracing algorithm

```
for (int j = 0; j < imageHeight; ++j) {
    for (int i = 0; i < imageWidth; ++i) {
        // compute primary ray direction
        Ray primRay;
        computePrimRay(i, j, &primRay);
        // shoot prim ray in the scene and search for the intersection
        Point pHit;
        Normal nHit;
        float minDist = INFINITY;
        Object object = NULL;
        for (int k = 0; k < objects.size(); ++k) {
            if (Intersect(objects[k], primRay, &pHit, &nHit)) {
                float distance = Distance(eyePosition, pHit);
                if (distance < minDistance) {
                    object = objects[k];
                    minDistance = distance;  //update min distance
                }
            }
        }
        if (object != NULL) {
            // compute illumination
            Ray shadowRay;
            shadowRay.direction = lightPosition - pHit;
            bool isShadow = false;
            for (int k = 0; k < objects.size(); ++k) {
                if (Intersect(objects[k], shadowRay)) {
                    isInShadow = true;
                    break;
                }
            }
        }
        if (!isInShadow)
            pixels[i][j] = object->color * light.brightness;
        else
            pixels[i][j] = 0;
    }
}
```

https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/raytracing-algorithm-in-a-nutshell.html

```
for (int j = 0; j < imageHeight; ++j) {
    for (int i = 0; i < imageWidth; ++i) {
        // compute primary ray direction
        Ray primRay;
        computePrimRay(i, j, &primRay);
        // shoot prim ray in the scene and search for the intersection
        Point pHit;
        Normal nHit;
        float minDist = INFINITY;
        Object object = NULL;
        for (int k = 0; k < objects.size(); ++k) {
            if (Intersect(objects[k], primRay, &pHit, &nHit)) {
                float distance = Distance(eyePosition, pHit);
                if (distance < minDistance) {
                    object = objects[k];
                    minDistance = distance;  //update min distance
                }
            }
        }
        if (object != NULL) {
            // compute illumination
            Ray shadowRay;
            shadowRay.direction = lightPosition - pHit;
            bool isShadow = false;
            for (int k = 0; k < objects.size(); ++k) {
                if (Intersect(objects[k], shadowRay)) {
                    isInShadow = true;
                    break;
                }
            }
        }
        if (!isInShadow)
            pixels[i][j] = object->color * light.brightness;
        else
            pixels[i][j] = 0;
    }
}
```

3:00

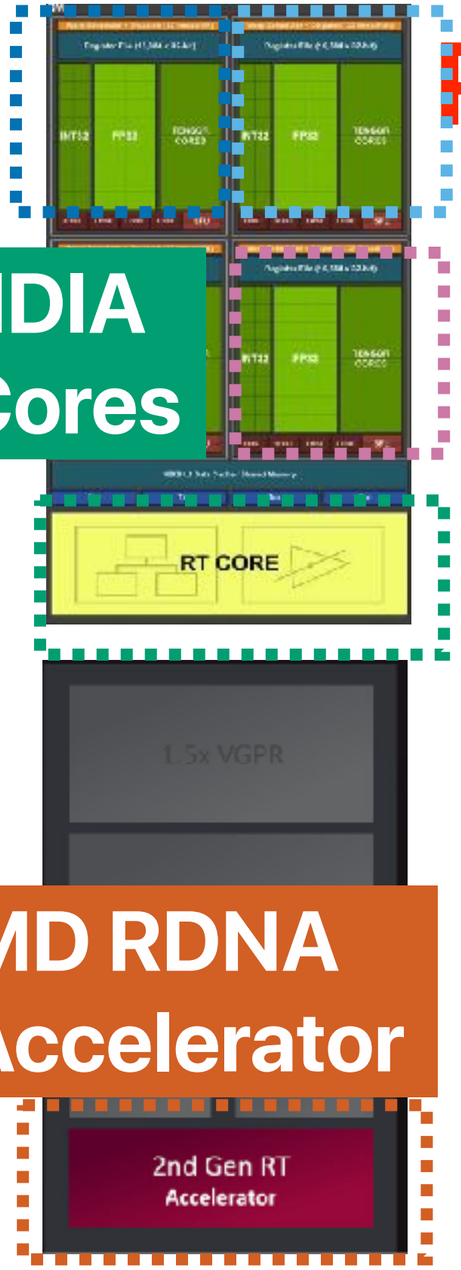
Why we need an accelerator instead of just using GPUs?

Simplest ray tracing algorithm

```
for (int j = 0; j < imageHeight; ++j) {
    for (int i = 0; i < imageWidth; ++i) {
        // compute primary ray direction
        Ray primRay;
        computePrimRay(i, j, &primRay);
        // shoot prim ray in the scene and search for the intersection
        Point pHit;
        Normal nHit;
        float minDist = INFINITY;
        Object object = NULL;
        for (int k = 0; k < objects.size(); ++k) {
            if (Intersect(objects[k], primRay, &pHit, &nHit)) {
                float distance = Distance(eyePosition, pHit);
                if (distance < minDistance) {
                    object = objects[k];
                    minDistance = distance;  //update min distance
                }
            }
        }
        if (object != NULL) {
            // compute illumination
            Ray shadowRay;
            shadowRay.direction = lightPosition - pHit;
            bool isShadow = false;
            for (int k = 0; k < objects.size(); ++k) {
                if (Intersect(objects[k], shadowRay)) {
                    isInShadow = true;
                    break;
                }
            }
        }
        if (!isInShadow)
            pixels[i][j] = object->color * light.brightness;
        else
            pixels[i][j] = 0;
    }
}
```

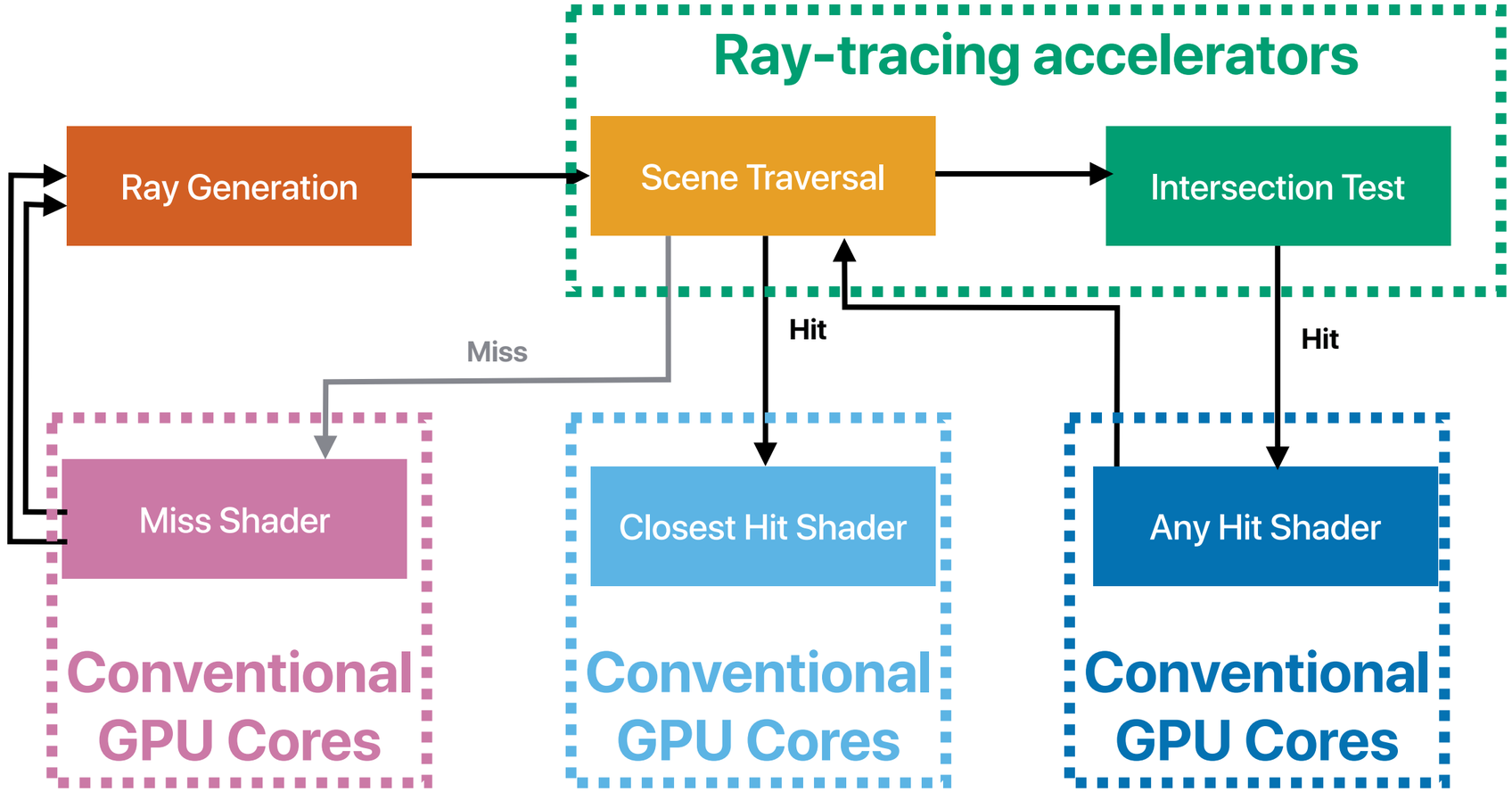
https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/raytracing-algorithm-in-a-nutshell.html

Ray Tracing Accelerators



NVIDIA RT Cores

AMD RDNA RT Accelerator



Recent development of NVIDIA GPUs

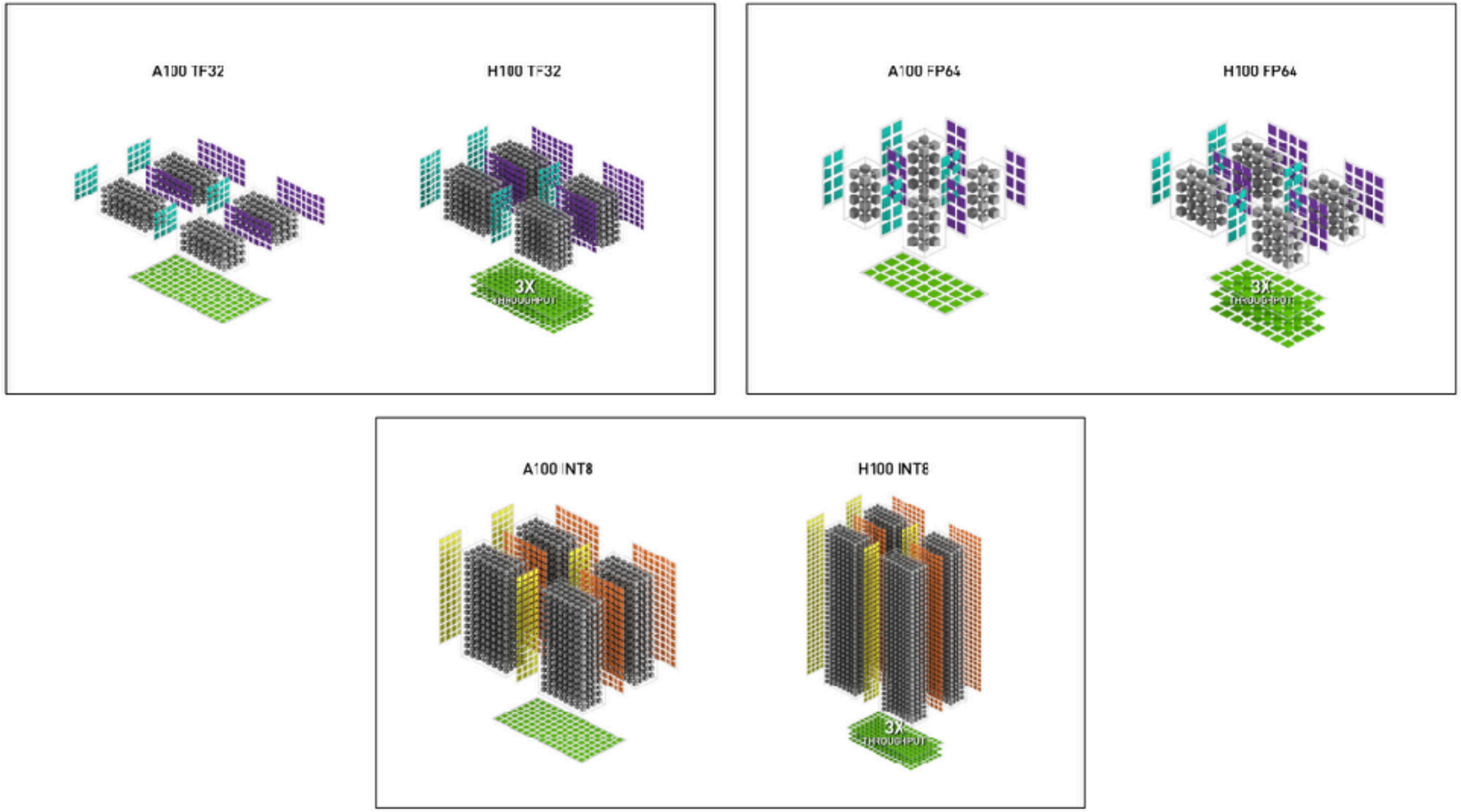


Table 2. H100 speedup over A100 (H100 Performance, TC=Tensor Core)

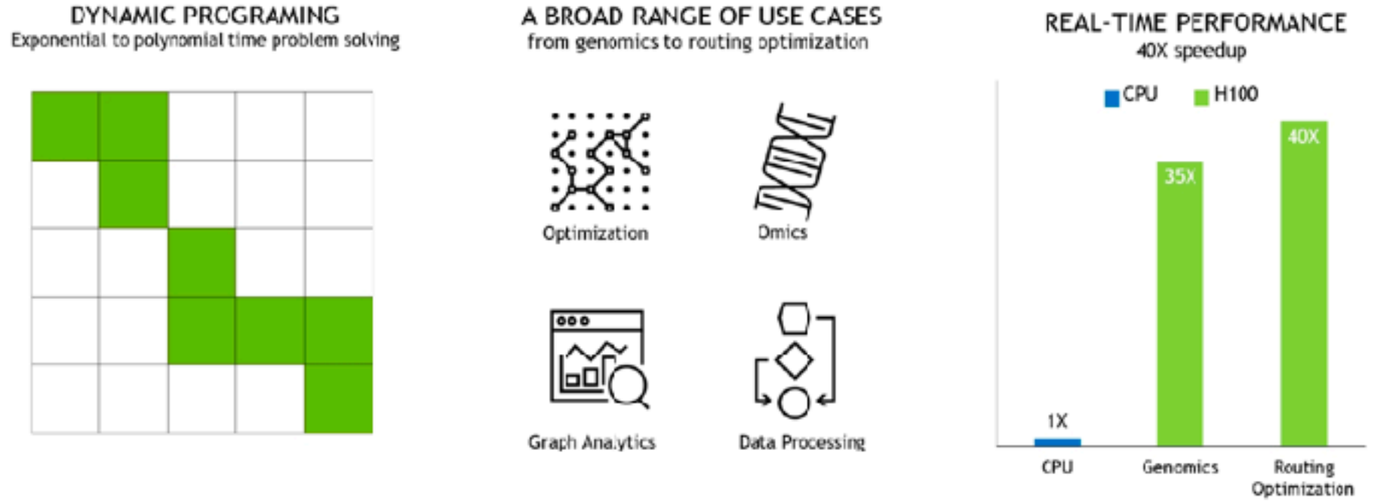
	A100	A100 Sparse	H100 SXM5	H100 SXM5 Sparse	H100 SXM5 Speedup vs A100
FP8 Tensor Core	NA	NA	1978.9 TFLOPS	3957.8 TFLOPS	6.3x vs A100 FP16 TC
FP16	78 TFLOPS	NA	133.8 TFLOPS	NA	1.7x
FP16 Tensor Core	312 TFLOPS	624 TFLOPS	989.4 TFLOPS	1978.9 TFLOPS	3.2x
BF16 Tensor Core	312 TFLOPS	624 TFLOPS	989.4 TFLOPS	1978.9 TFLOPS	3.2x
FP32	19.5 TFLOPS	NA	66.9 TFLOPS	NA	3.4x
TF32 Tensor Core	156 TFLOPS	312 TFLOPS	494.7 TFLOPS	989.4 TFLOPS	3.2x
FP64	9.7 TFLOPS	NA	33.5 TFLOPS	NA	3.5x
FP64 Tensor Core	19.5 TFLOPS	NA	66.9 TFLOPS	NA	3.4x
INT8 Tensor Core	624 TOPS	1248 TOPS	1978.9 TFLOPS	3957.8 TFLOPS	3.2x

New DPX Instructions for Accelerated Dynamic Programming

Many "brute force" optimization algorithms have the property that a sub-problem solution is reused many times when solving the larger problem. Dynamic Programming is an algorithmic technique for solving a complex recursive problem by breaking it down into simpler sub-problems. By storing the results of sub-problems, without the need to recompute them when needed later, Dynamic Programming algorithms reduce the computational complexity of exponential problem sets to a linear scale.

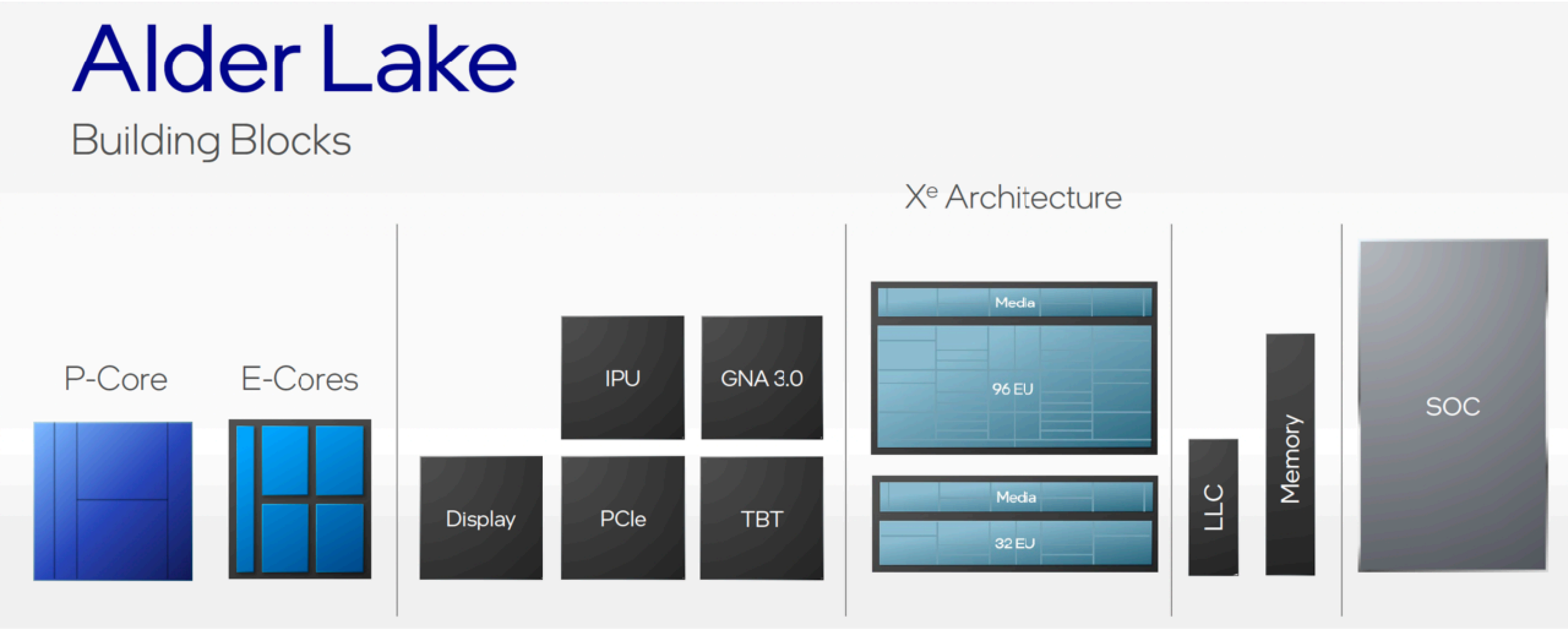
Dynamic programming is commonly used in a broad range of optimization, data processing, and genomics algorithms. In the rapidly growing field of genome sequencing, the Smith-Waterman dynamic programming algorithm is one of the most important methods in use. In the robotics space, Floyd-Warshall is a key algorithm used to find optimal routes for a fleet of robots through a dynamic warehouse environment in real-time.

H100 introduces DPX instructions to accelerate the performance of Dynamic Programming algorithms by up to 7x compared to Ampere GPUs. These new instructions provide support for advanced fused operands for the inner loop of many DP algorithms. This will lead to dramatically faster times-to-solutions in disease diagnosis, logistics routing optimizations, and even graph analytics.

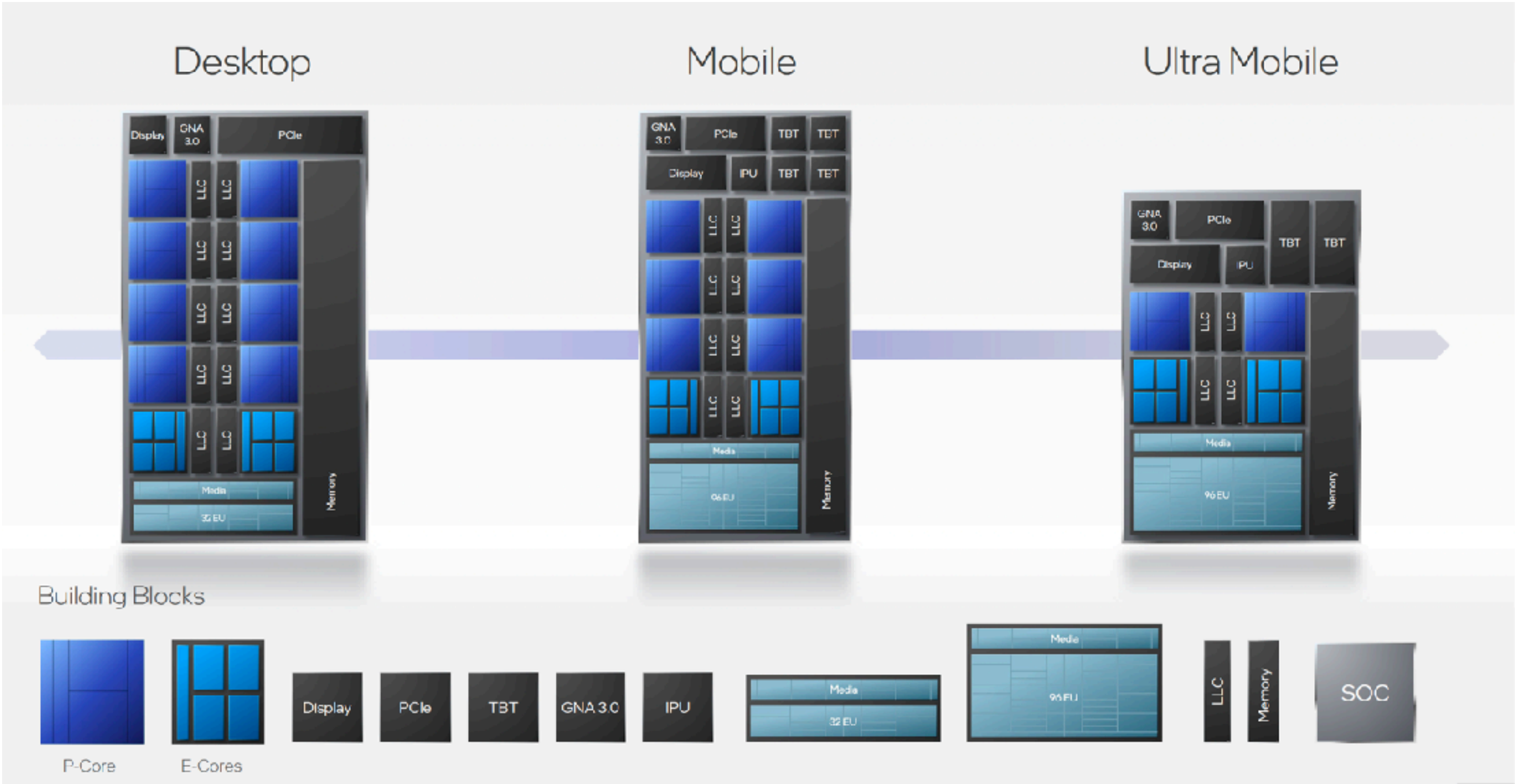


Processors are also heterogeneous now!

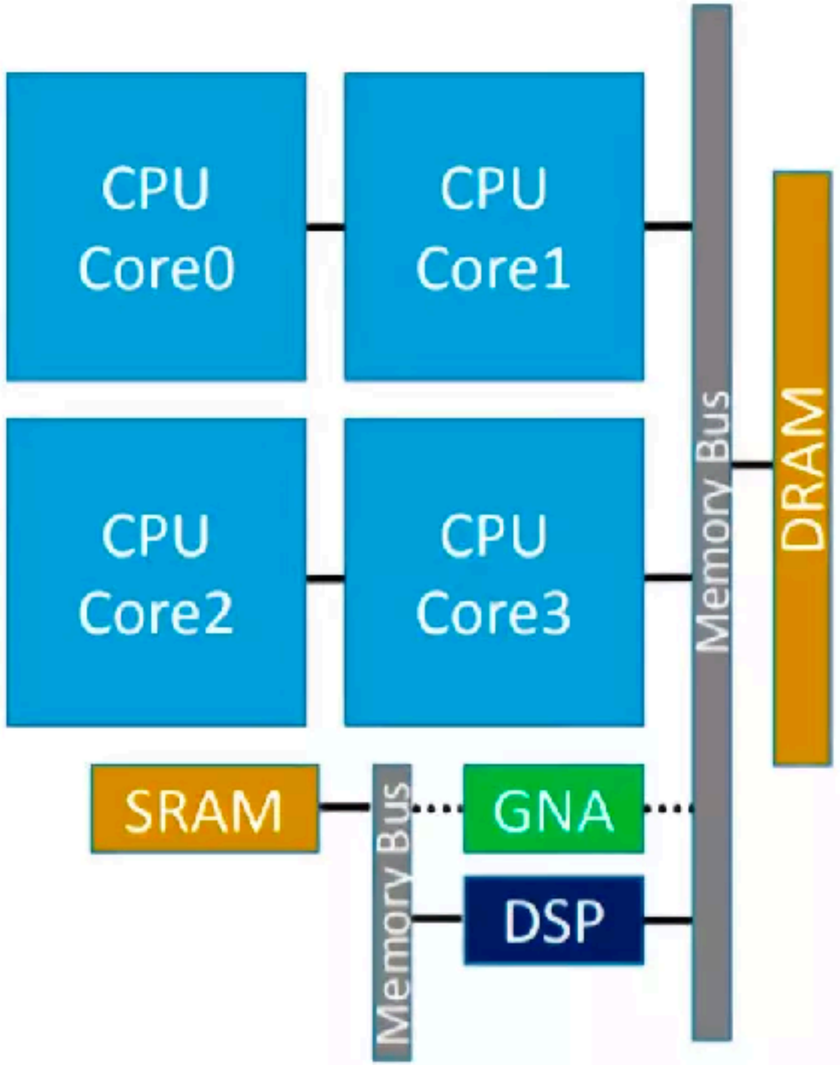
The intel Alder Lake



The intel Alder Lake



intel processor architecture



Each chip is now also a collection of accelerators

intel

Introducing Intel® Core™ Ultra Processors for the Edge

https://cdrdv2-public.intel.com/817736/817736_Heterogeneous%20AI%20Powerhouse-Intel%20Core%20Ultra-WP-Rev1.0.pdf

3.0 Introducing Intel® Core™ Ultra Processors for the

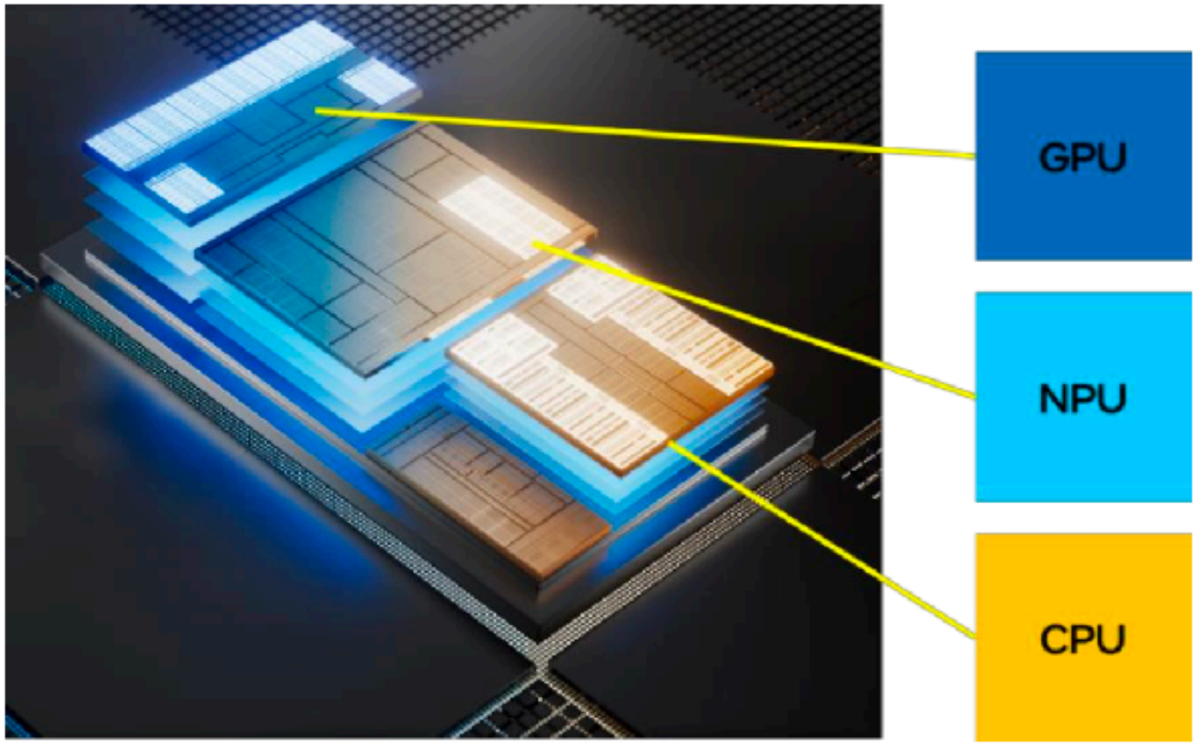
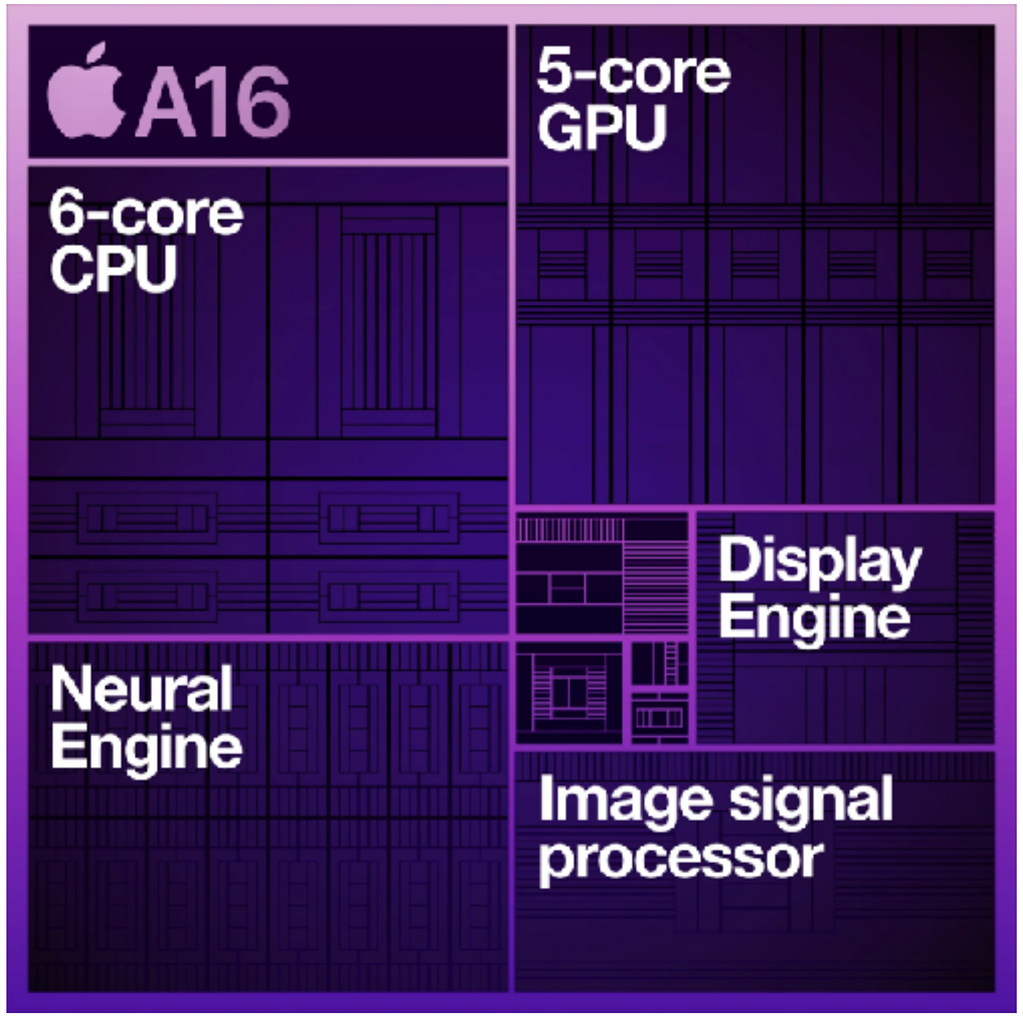


Figure 1: High-Level Architecture Diagram

Each chip is now also a collection of accelerators



Roogle Project Meetings

- Make an appointment through the Google Calendar
- Available resources
  - SmartSSD, Edge TPUs, CUDA GPUs, Tensor Cores, or something else
- Project ideas
  - Accelerating applications through AI/ML accelerators
  - Accelerating applications through intelligent storage devices
  - Accelerating applications through innovative parallel programming models that hardware accelerators enable
  - Anything related to what we discussed in this class!

Electrical
Computer Science
Engineering

277

つづく