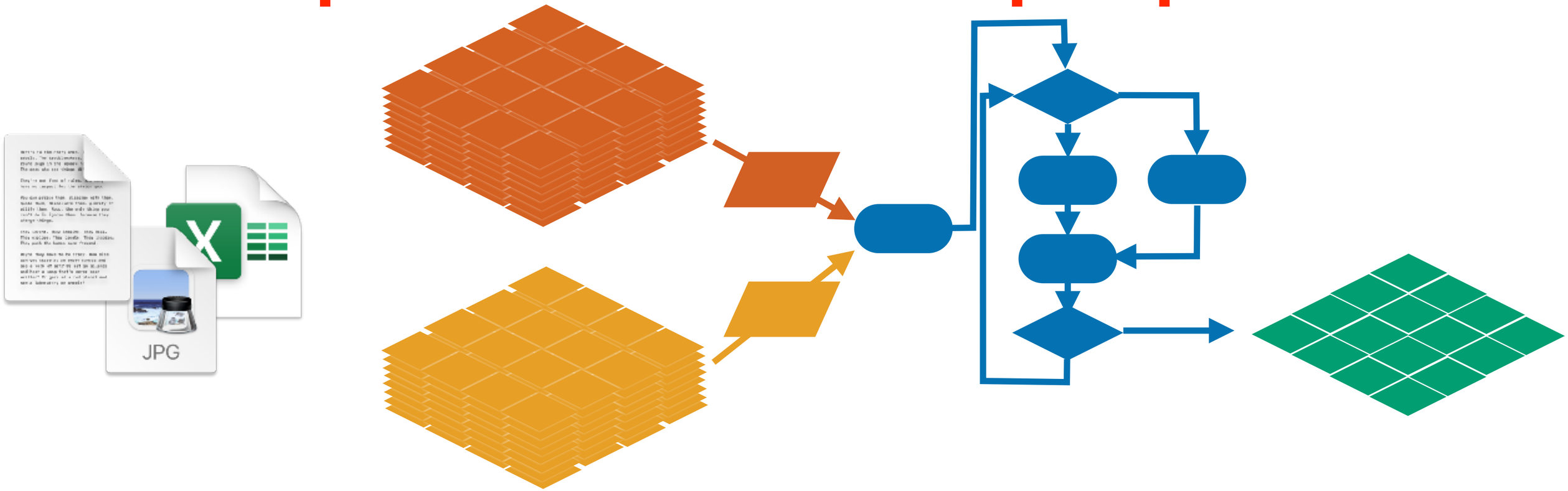
Hardware accelerators (2)

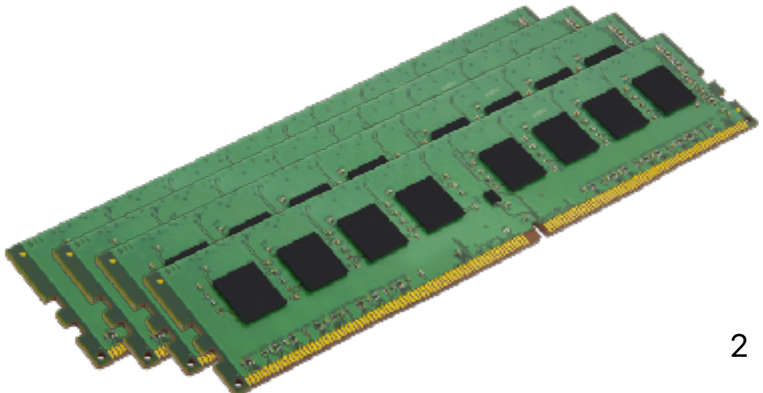
Hung-Wei Tseng

Recap: From the software perspective

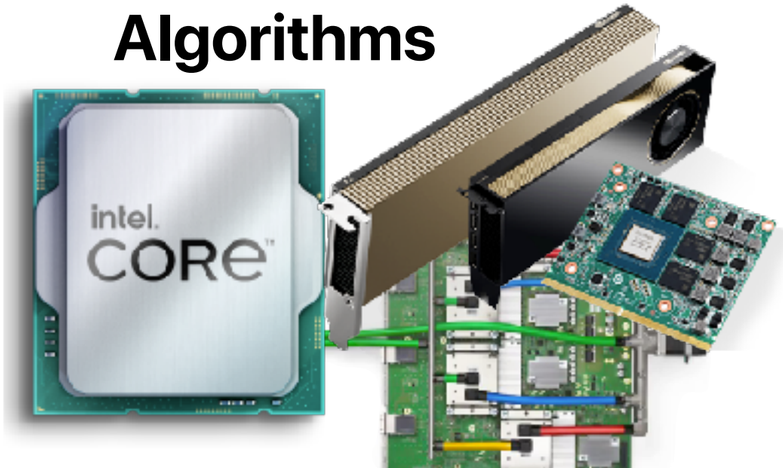


Source “data"

“Data” structures

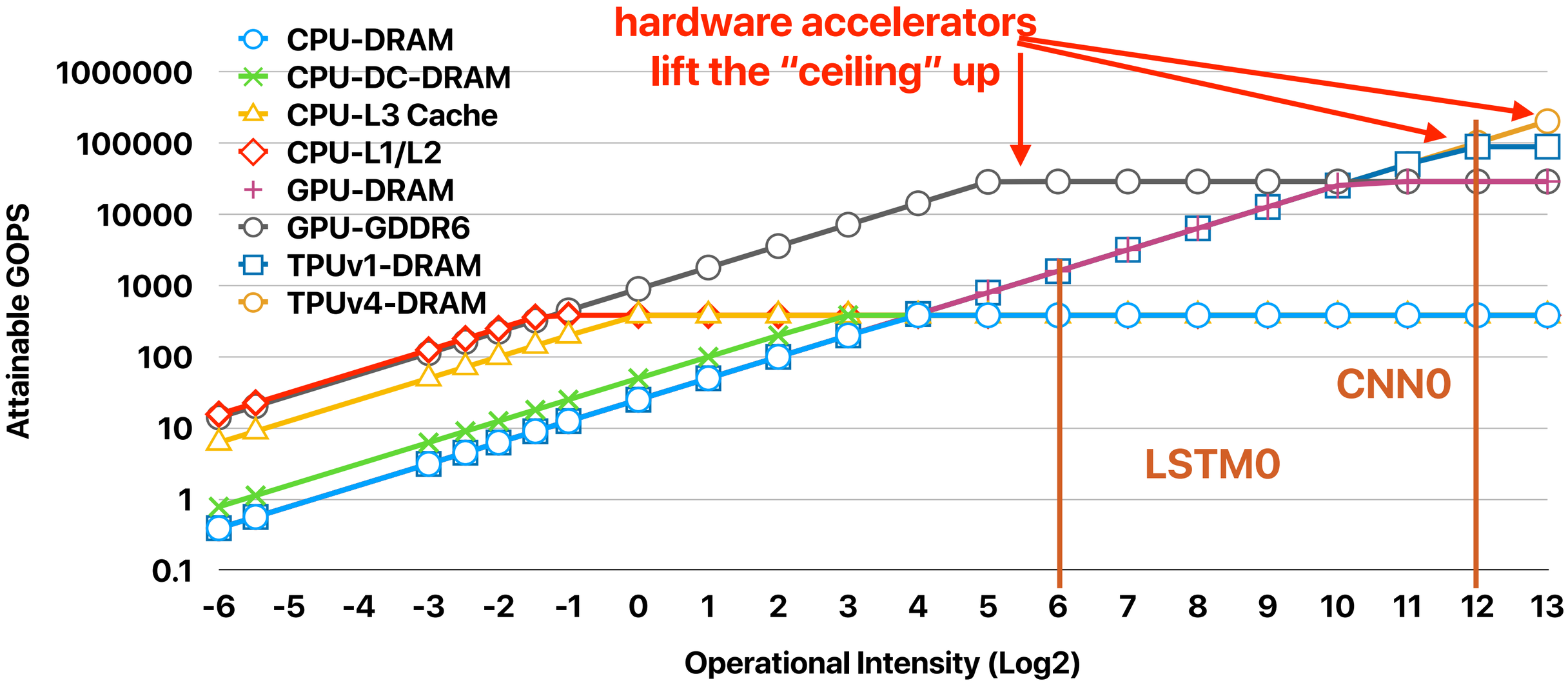


Algorithms

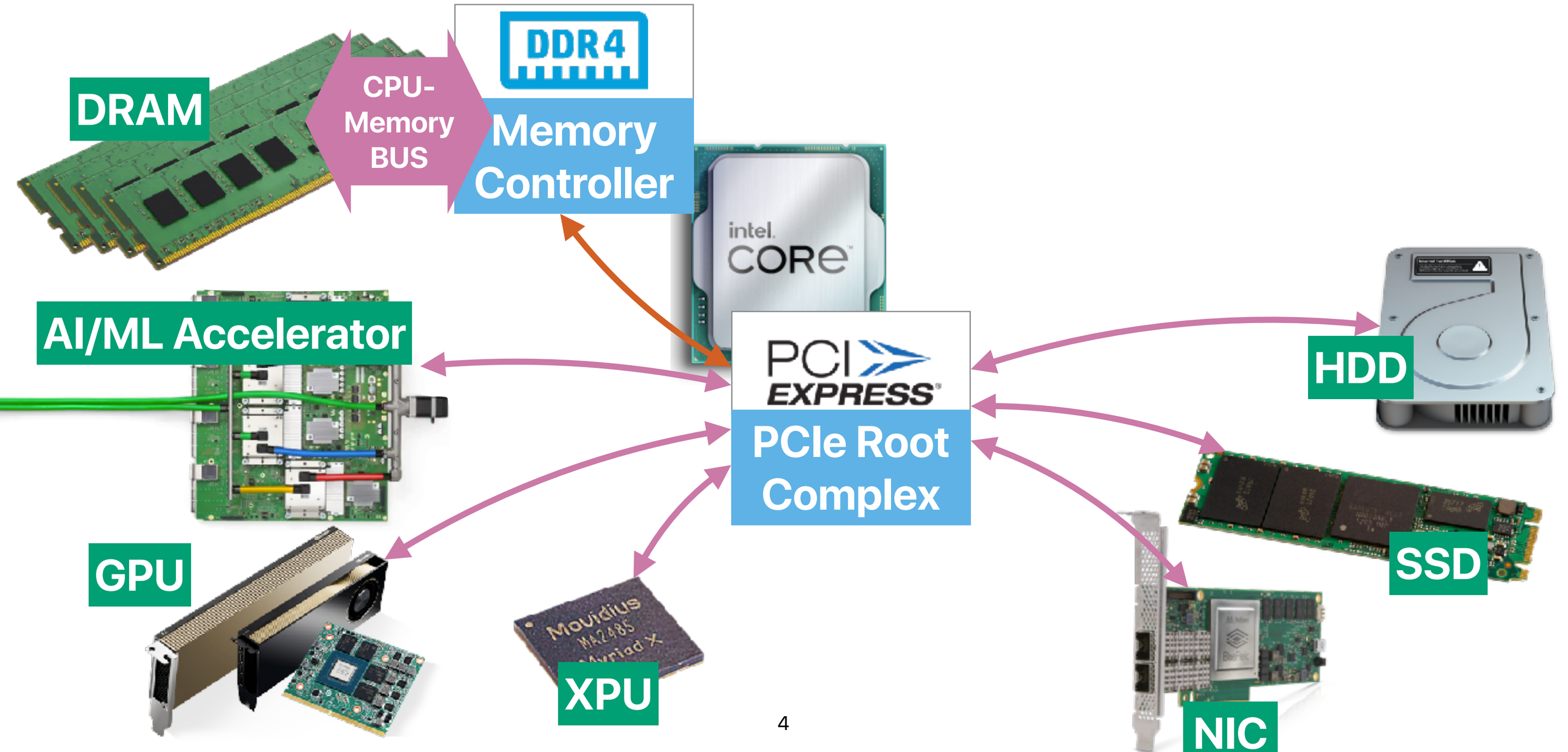


Result

Recap: the rooflines of modern systems



Recap: the landscape of heterogeneous computing



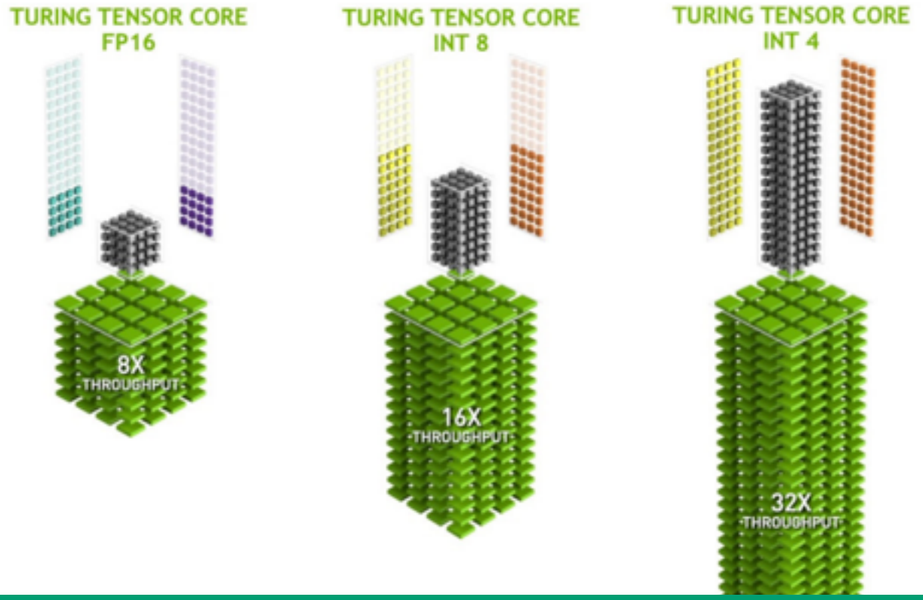
Recap: Turing architecture — specialized for key workloads

- 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.

Recap: 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 =

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

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

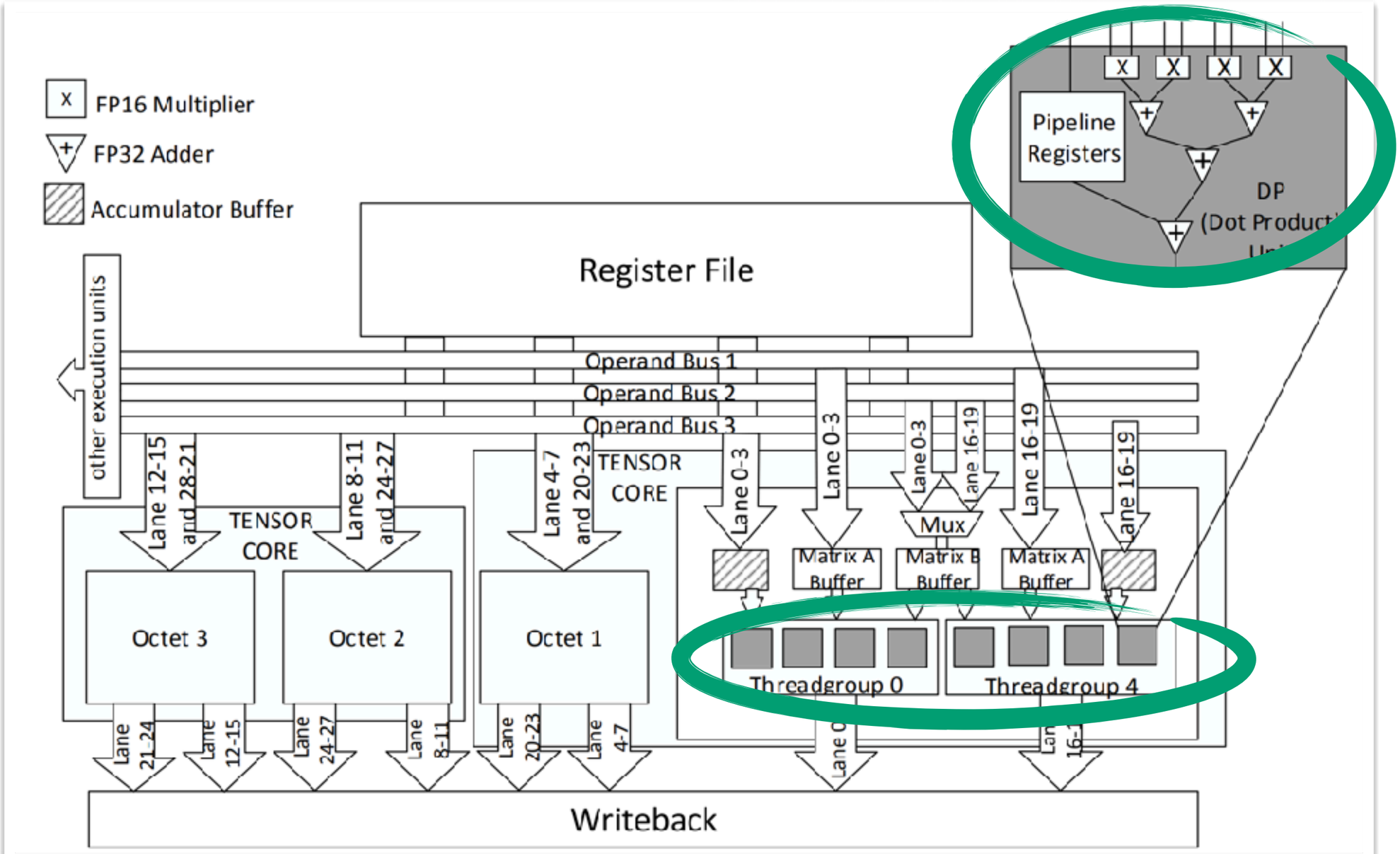
$$\frac{8192 \times 8192 \times 8192 \times 2}{320 \times 128} = 26843546 \ Tensor \ core \ cycles$$

Tensor cores make matrix processing 16x faster

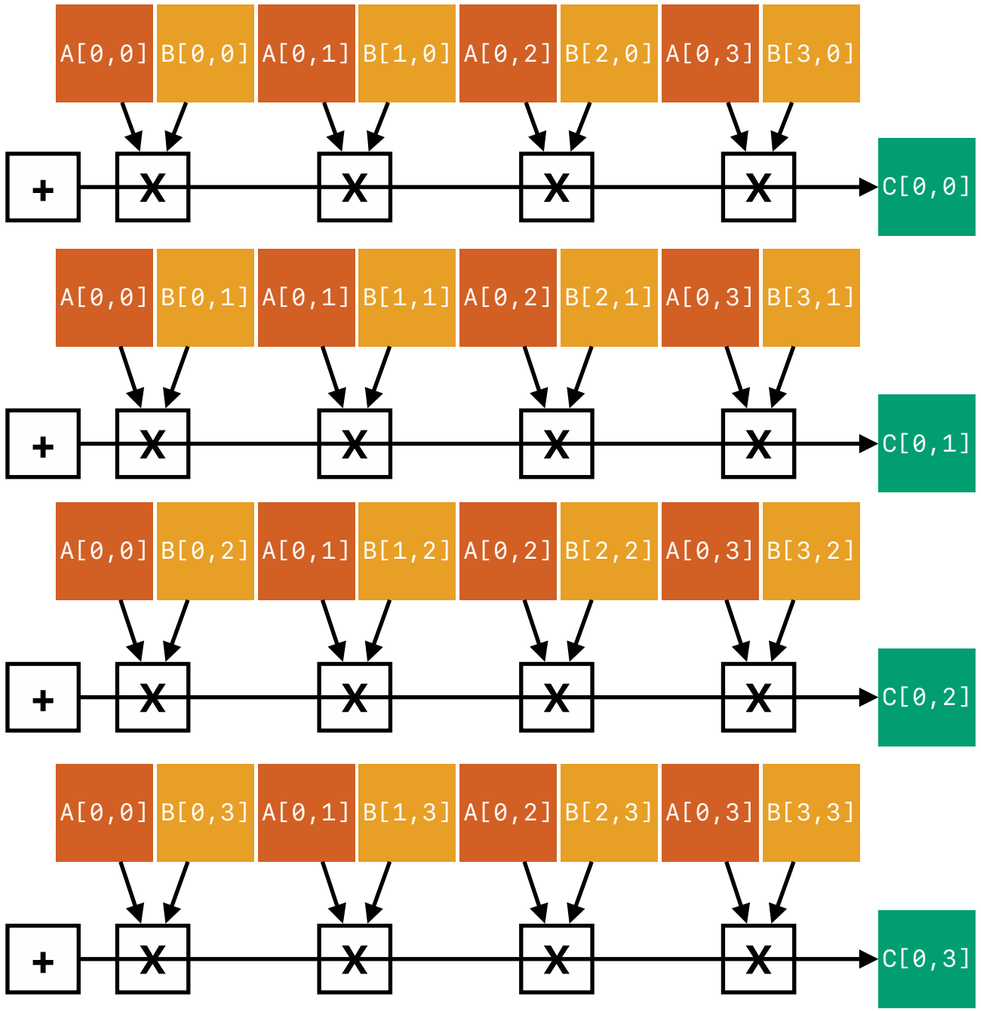
Each tensor core operation performs 4x4x4 MMA in one cycle
~ 128 FP operations in conventional scalar models

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

Recap: Tensor Core Architecture



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]

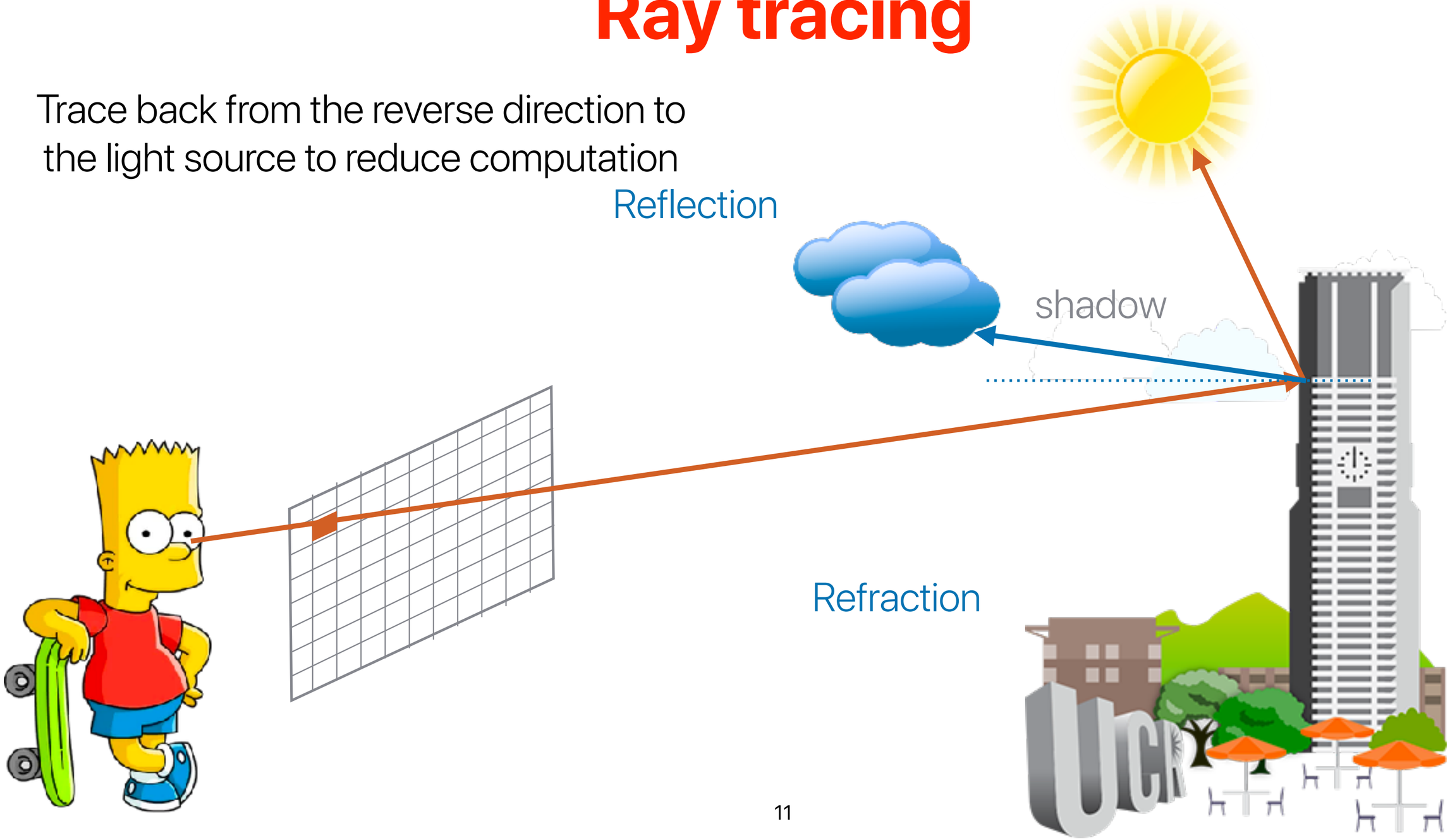
Outline

- RT Cores
- FPGAs
- General-purpose approximate computing on NPUs

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

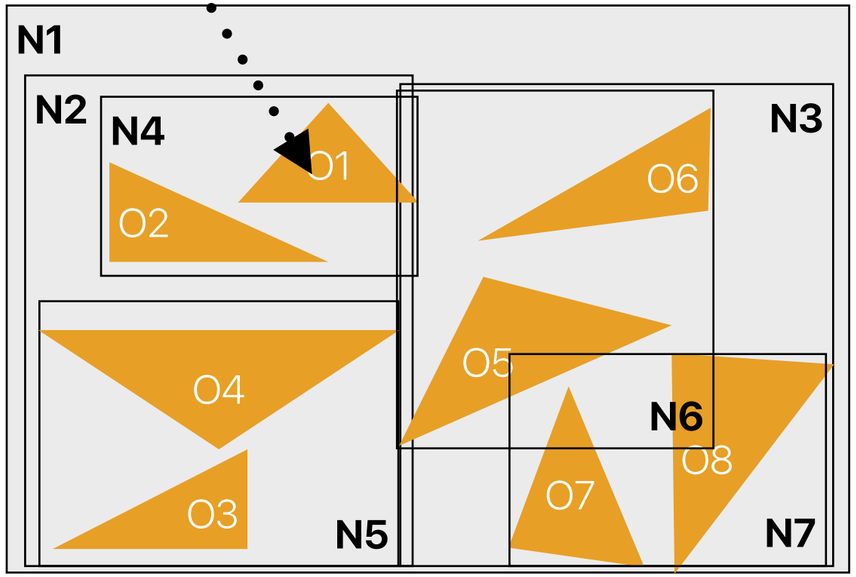
Why we need an accelerator instead of just using GPUs?

Simplest ray tracing algorithm

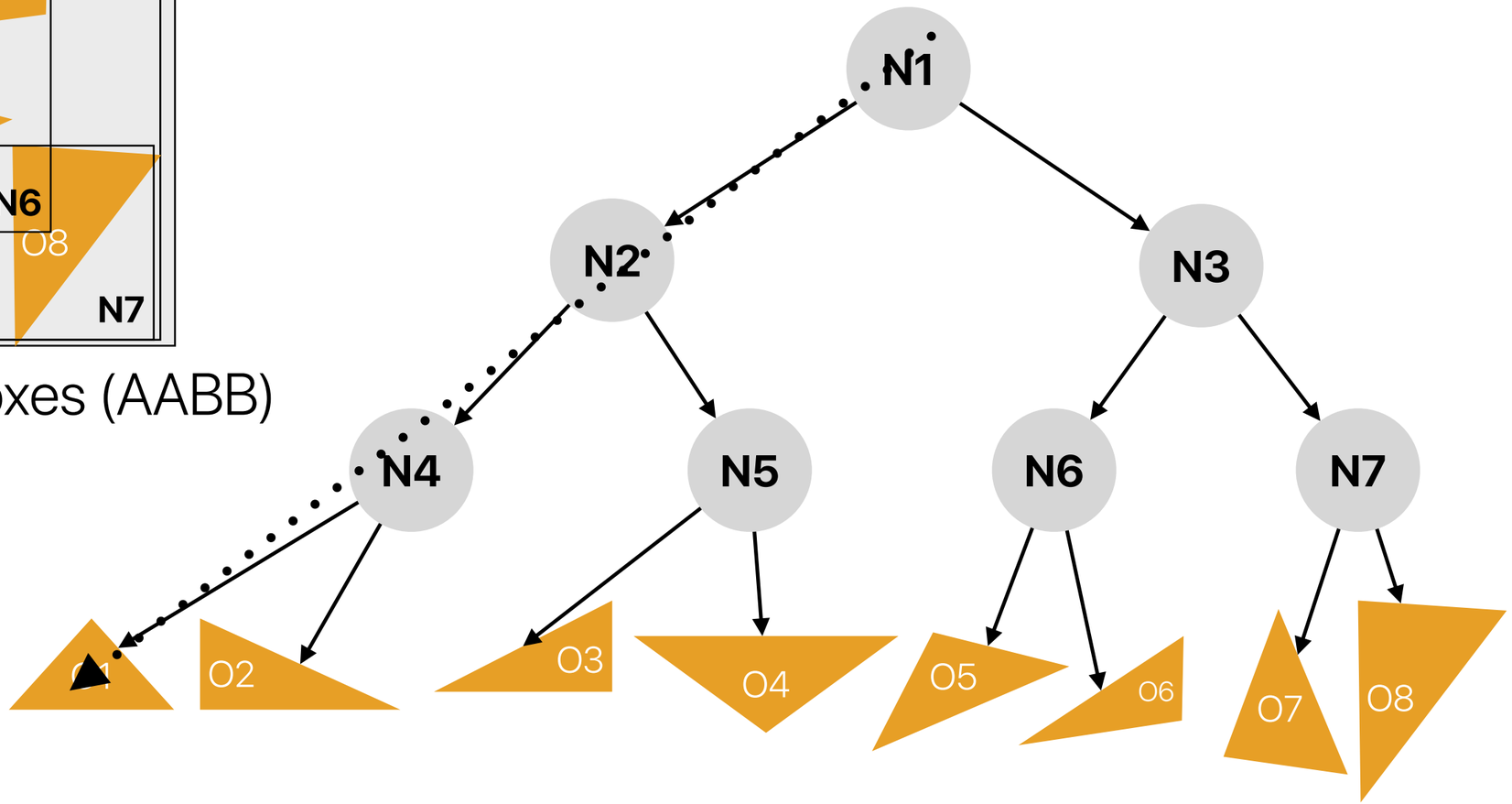
```
for (int j = 0; j < imageHeight; ++j) {
    for (int i = 0; i < imageWidth; ++i) {
        // compute primary ray direction
        Ray primRay;
        computePrimRay(i, j, &primRay);
        // shoot prim ray in the scene and search for the intersection
        Point pHit;
        Normal nHit;
        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

Bounding volume hierarchy



axis-aligned bounding boxes (AABB)



Recent development of NVIDIA GPUs

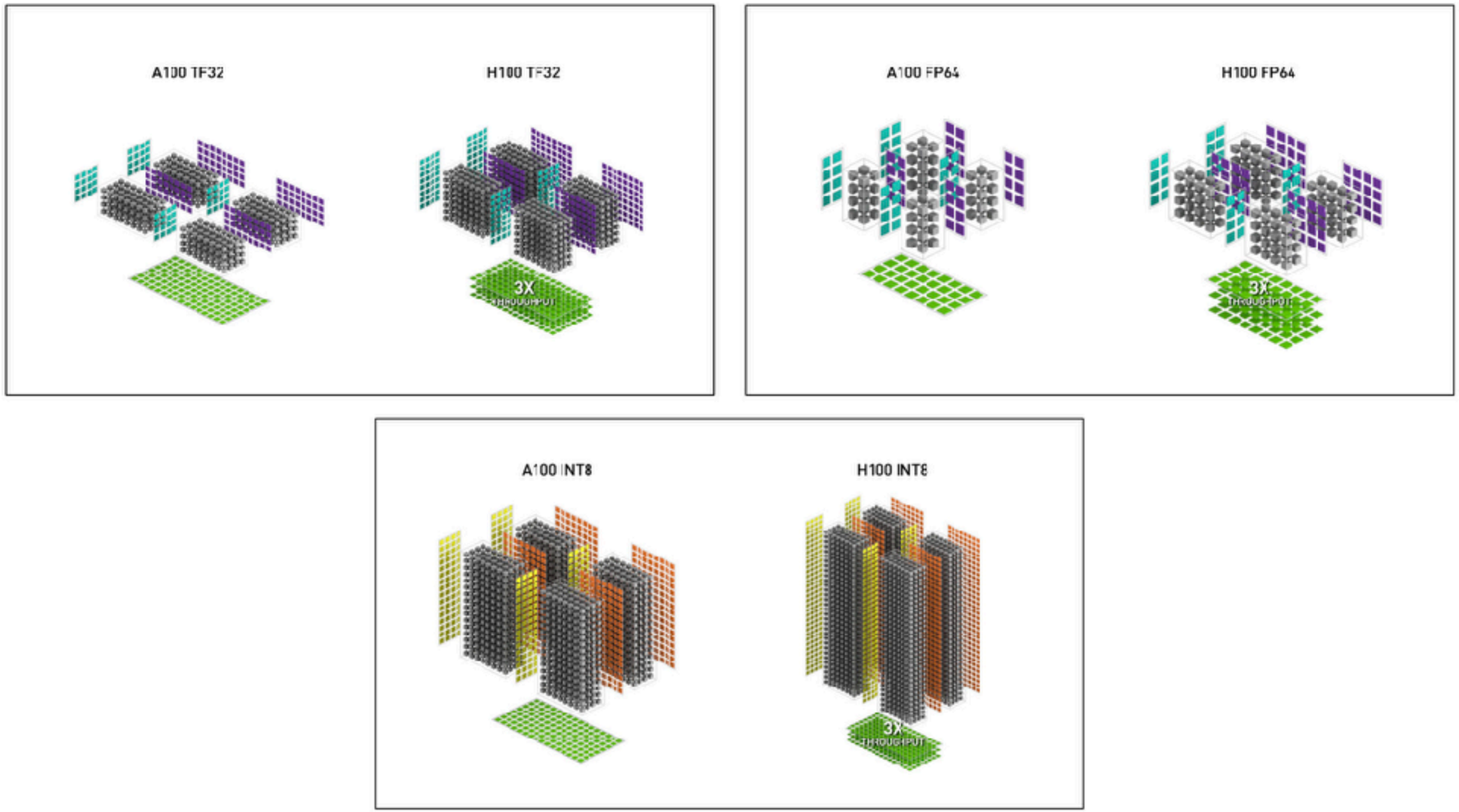


Table 2. H100 speedup over A100 (H100 Performance, TC=Tensor Core)

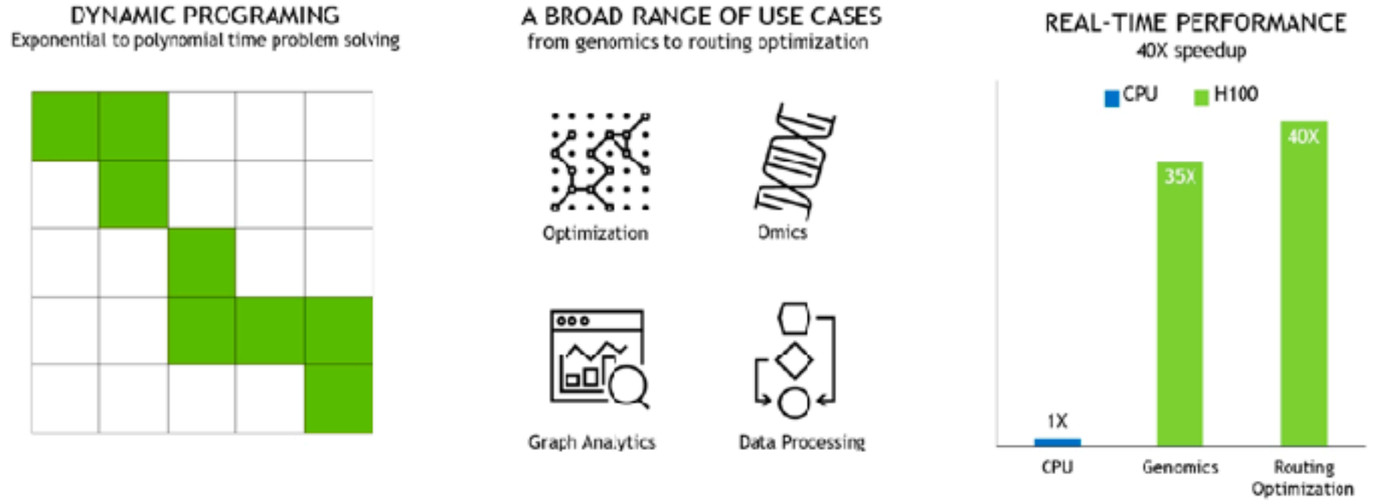
	A100	A100 Sparse	H100 SXM5	H100 SXM5 Sparse	H100 SXM5 Speedup vs A100
FP8 Tensor Core	NA	NA	1978.9 TFLOPS	3957.8 TFLOPS	6.3x vs A100 FP16 TC
FP16	78 TFLOPS	NA	133.8 TFLOPS	NA	1.7x
FP16 Tensor Core	312 TFLOPS	624 TFLOPS	989.4 TFLOPS	1978.9 TFLOPS	3.2x
BF16 Tensor Core	312 TFLOPS	624 TFLOPS	989.4 TFLOPS	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.

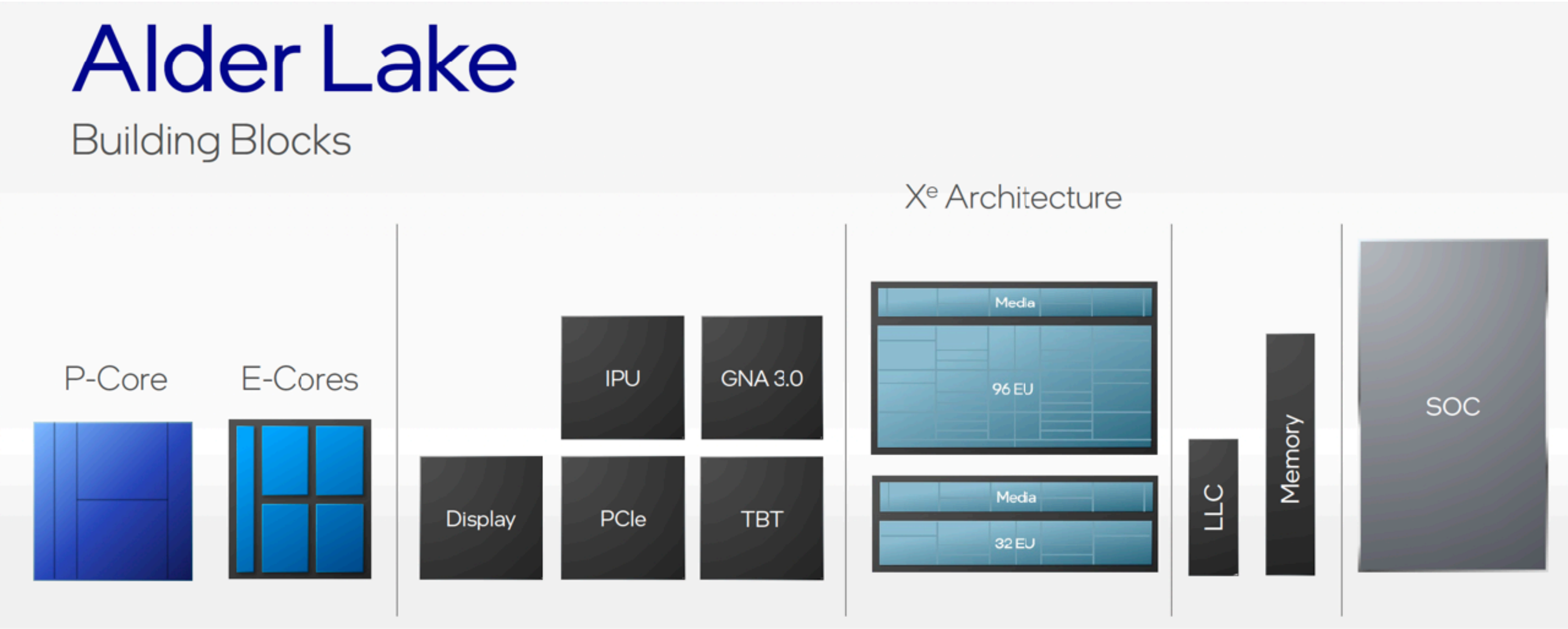
Dynamic programming is commonly used in a broad range of optimization, data processing, and genomics algorithms. In the rapidly growing field of genome sequencing, the Smith-Waterman dynamic programming algorithm is one of the most important methods in use. In the robotics space, Floyd-Warshall is a 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.

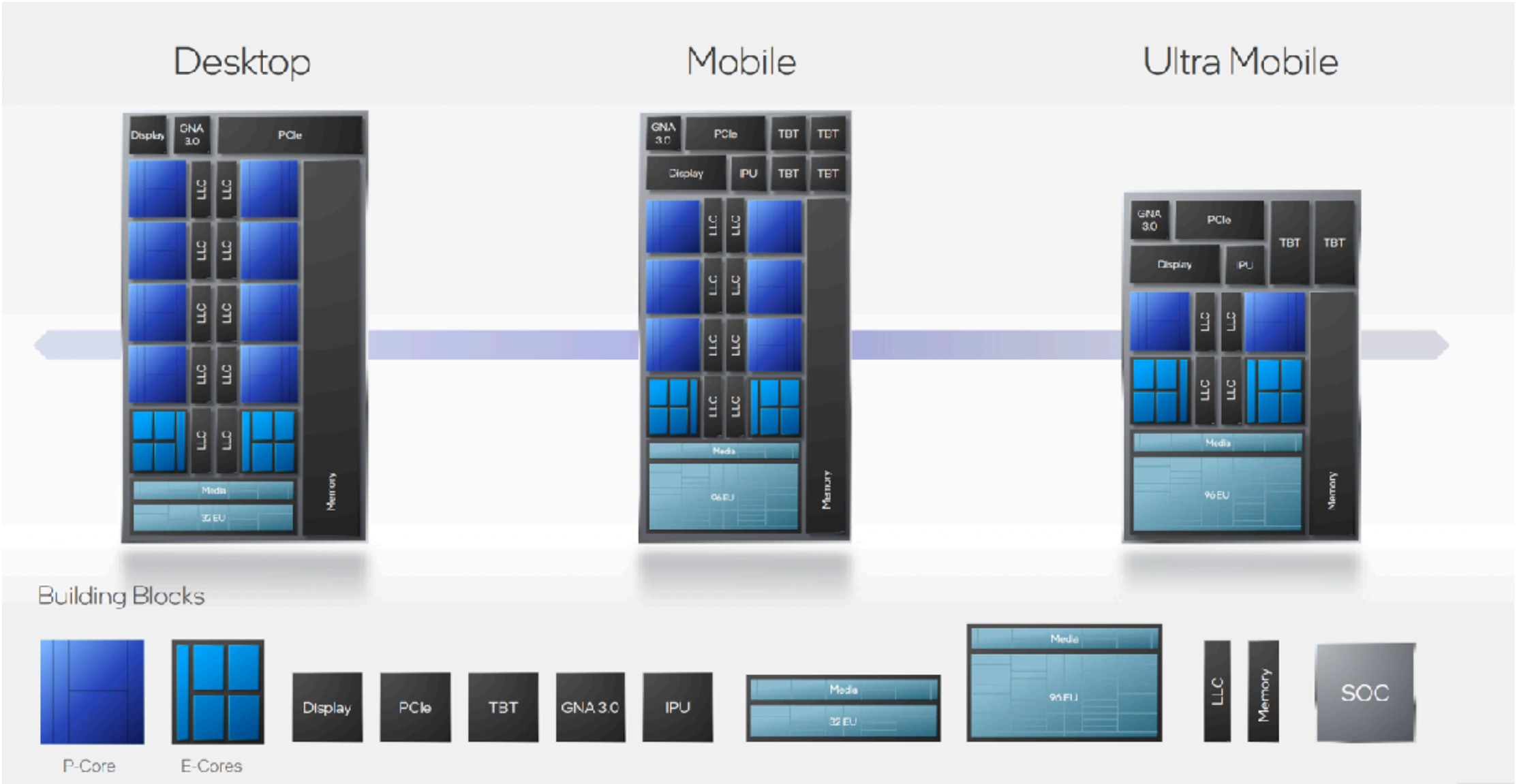


Processors are also heterogeneous now!

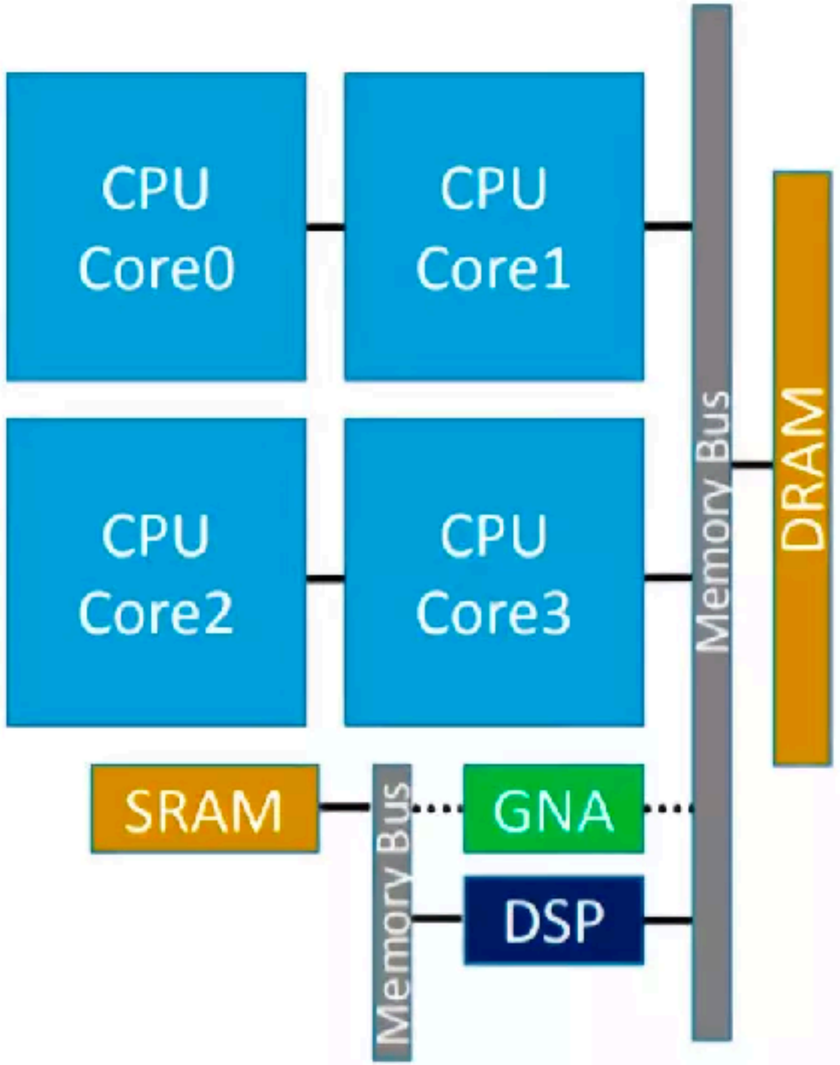
The intel Alder Lake



The intel Alder Lake



intel processor architecture



3:00

Is SoC (i.e., a processor with GPUs/ accelerators in the same package) a good idea compared with standalone accelerators?

Single-chip or standalone ones?

Is single-chip, heterogeneous processor a good idea?

- Potentially reduce the system interconnect bandwidth demand (still need to use intra-chip interconnect)
- Reduce the total system size and cost
- The design of the memory controller is complicated. For example, CPUs are more latency sensitive and want pairs of scalar values. GPUs/accelerators are more bandwidth demanding and want pairs of vectors
  - J. Power, M. D. Hill and D. A. Wood, "Supporting x86-64 address translation for 100s of GPU lanes," 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014, pp. 568-578
- The power consumption per-chip is still limiting the overall performance
  - You can have only moderate processor cores with moderate GPU cores
  - You can have performance processor cores with wimpy graphic cores
  - You have to dynamically switch each part on/off
- Or you have to increase power consumption — NVIDIA's H100 goes up to 700W!!!

Is system-wide heterogeneous processing a better idea?

- Easier for heat dissipation
  - Each processor can be more powerful
    - gamers/datacenters still prefers discrete GPUs
    - Cloud TPUs are standalone ones
- System size is larger, cost of ownership can be higher
- Data movement going through system interconnects

Programming interface — NV's WMMA

```
__global__ void WMMAF16TensorCore(half *A, half *B, float *C, float *D) {

    int ix = (blockIdx.x * blockDim.x + threadIdx.x)/WARP_SIZE;
    int iy = (blockIdx.y * blockDim.y + threadIdx.y);

    wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::row_major> a_frag;
    wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b_frag;
    wmma::fragment<wmma::accumulator, M, N, K, float> ab_frag;
    wmma::fragment<wmma::accumulator, M, N, K, float> c_frag;

    wmma::fill_fragment(ab_frag, 0.0f);

    // AB = A*B

    int a_col, a_row, b_col, b_row, c_col, c_row;

    a_row = ix * M;
    b_row = iy * N;

    for (int k=0; k<K_TOTAL; k+=K) {
        a_col = b_col = k;

        if (a_row < M_TOTAL && a_col < K_TOTAL && b_row < K_TOTAL && b_col <
N_TOTAL) {
            // Load the inputs
            wmma::load_matrix_sync(a_frag, A + a_col + a_row * M_TOTAL,
M_TOTAL);
            wmma::load_matrix_sync(b_frag, B + b_col + b_col * K_TOTAL,
K_TOTAL);

            // Perform the matrix multiplication
            wmma::mma_sync(ab_frag, a_frag, b_frag, ab_frag);
        }
    }

    // D = AB + C
    c_col = b_row;
    c_row = a_row;

    if (c_row < M_TOTAL && c_col < N_TOTAL) {
        wmma::load_matrix_sync(c_frag, C + c_col + c_row * N_TOTAL, N_TOTAL,
wmma::mem_row_major);

        for (int i = 0; i < c_frag.num_elements; i++) {
            c_frag.x[i] = ab_frag.x[i] + c_frag.x[i];
        }
        // Store the output
        wmma::store_matrix_sync(D + c_col + c_row * N_TOTAL, c_frag,
N_TOTAL, wmma::mem_row_major);
    }
}
```

intel GNA

```
// Open selected  device
status = Gna2DeviceOpen(deviceIndex);
CleanupOrError(status, deviceIndex, nullptr, GNA2_DISABLED, GNA2_DISABLED,
               "main", "Gna2DeviceOpen()");

/* Calculate model memory parameters for GnaAlloc. */
int buf_size_weights = Gna2RoundUpTo64(sizeof(weights));
// note that buffer alignment to 64-bytes is required by GNA HW
int buf_size_inputs = Gna2RoundUpTo64(sizeof(inputs));
int buf_size_biases = Gna2RoundUpTo64(sizeof(biases));
int buf_size_outputs = Gna2RoundUpTo64(H * B * 4); // (4 out vectors, H elems
in each one, 4-byte elems)

auto rw_buffer_size = Gna2RoundUp(buf_size_inputs + buf_size_outputs, 0x1000);
auto bytes_requested = rw_buffer_size + buf_size_weights + buf_size_biases;

// Allocate GNA memory (obtains pinned memory shared with the device)
uint32_t bytes_granted;
void* memory = nullptr;
status = Gna2MemoryAlloc(bytes_requested, &bytes_granted, &memory);

/* Prepare model memory layout. */
auto model_memory = reinterpret_cast<uint8_t*>(memory);

auto rw_buffers = model_memory;

auto pinned_inputs = reinterpret_cast<int16_t*>(rw_buffers);
memcpy_s(pinned_inputs, buf_size_inputs, inputs, sizeof(inputs)); // puts the
inputs into the pinned memory
rw_buffers += buf_size_inputs; // fast-forwards current pinned memory pointer
to the next free block

auto pinned_outputs = reinterpret_cast<int32_t*>(rw_buffers);
rw_buffers += buf_size_outputs; // fast-forwards the current pinned memory
pointer by the space needed for outputs

model_memory += rw_buffer_size;
auto weights_buffer = reinterpret_cast<int16_t*>(model_memory);
memcpy_s(weights_buffer, buf_size_weights, weights, sizeof(weights)); // puts
the weights into the pinned memory
model_memory += buf_size_weights; // fast-forwards current pinned memory
pointer to the next free block

auto biases_buffer = reinterpret_cast<int32_t*>(model_memory);
memcpy_s(biases_buffer, buf_size_biases, biases, sizeof(biases)); // puts the
biases into the pinned memory
model_memory += buf_size_biases; // fast-forwards current pinned memory pointer
to the next free block

/* Prepare neural network topology,
 * Single FullyConnectedAffine layer in this example. */

/* Prepare and initialize FullyConnectedAffine operation operands with GNA API
helpers */
auto inputTensor = Gna2TensorInit2D(W, B, Gna2DataTypeInt16, pinned_inputs);
auto outputTensor = Gna2TensorInit2D(H, B, Gna2DataTypeInt32, pinned_outputs);
auto weightTensor = Gna2TensorInit2D(H, W, Gna2DataTypeInt16, weights_buffer);
auto biasTensor = Gna2TensorInit1D(H, Gna2DataTypeInt32, biases_buffer);
auto activationTensor = Gna2TensorInitDisabled();

/* Create single FullyConnectedAffine operation (layer) */
auto operation = Gna2Operation{};
status = Gna2OperationInitFullyConnectedAffine(&operation, customAlloc,
                                               &inputTensor, &outputTensor,
&weightTensor, &biasTensor,
                                               &activationTensor);

/* Create data-flow model with single operation (layer) */
Gna2Model model = {1, &operation};
uint32_t modelId = GNA2_DISABLED;
status = Gna2ModelCreate(deviceIndex, &model, &modelId);

// Create request configuration used for queueing inference requests
uint32_t configId = GNA2_DISABLED;
status = Gna2RequestConfigCreate(modelId, &configId);

// Set model input data buffer, operation 0, operand 0, for this sample
status = Gna2RequestConfigSetOperandBuffer(configId, 0, 0, pinned_inputs);

// Set model output data buffer, operation 0, operand 1, for this sample
status = Gna2RequestConfigSetOperandBuffer(configId, 0, 1, pinned_outputs);

// [optional] Set acceleration mode automatic (software emulation used if no
hardware detected)
status = Gna2RequestConfigSetAccelerationMode(configId,
Gna2AccelerationModeAuto);

// Enqueue inference request (non-blocking call)
```

Lessons learned from accelerators?

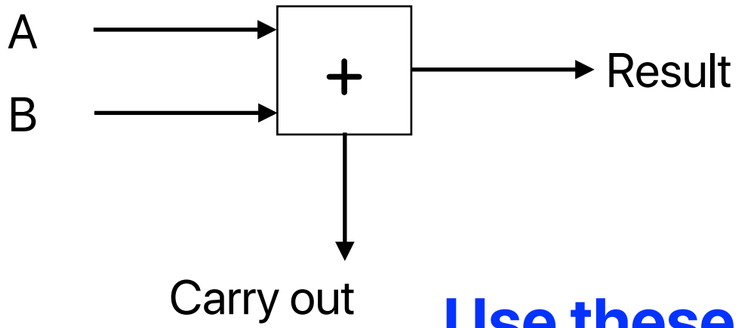
- Accelerators “lift up” the roofline
  - Applications/compute kernels with higher arithmetic densities may be feasible
    - NN is feasible after GPGPU
    - Trade “complexity” with parallelism
  - Applications are more likely to be memory-bound
  - Your software should try to avoid frequent memory access
    - Try to use memory closer to the processing elements
  - The hardware design must not ignore the importance of memory bandwidth
- The most “efficient” system design must land on the “turning point” of your roofline model
  - TPU’s 167GB/sec memory bandwidth is an example

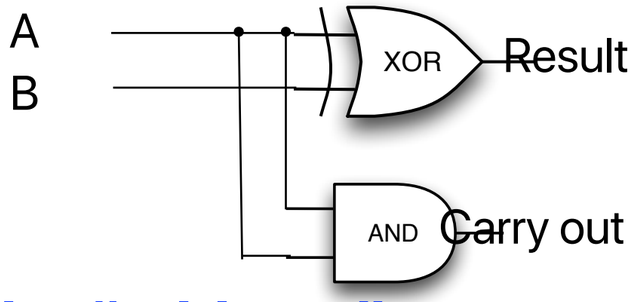
A Cloud-Scale Acceleration Architecture

Adrian Caulfield, Eric Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, Doug Burger
Microsoft

How to implement a half adder?

- What gates do you need to implement a half adder?





Use these as the "address"

This is what we really want for a certain feature!

A	B	Result	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

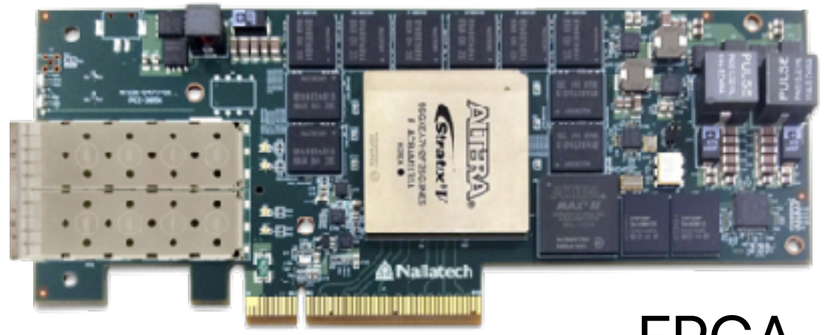
Use SRAM cells to keep these

Lookup table

A	B	Result	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

SRAM Cell Value
00
10
10
01

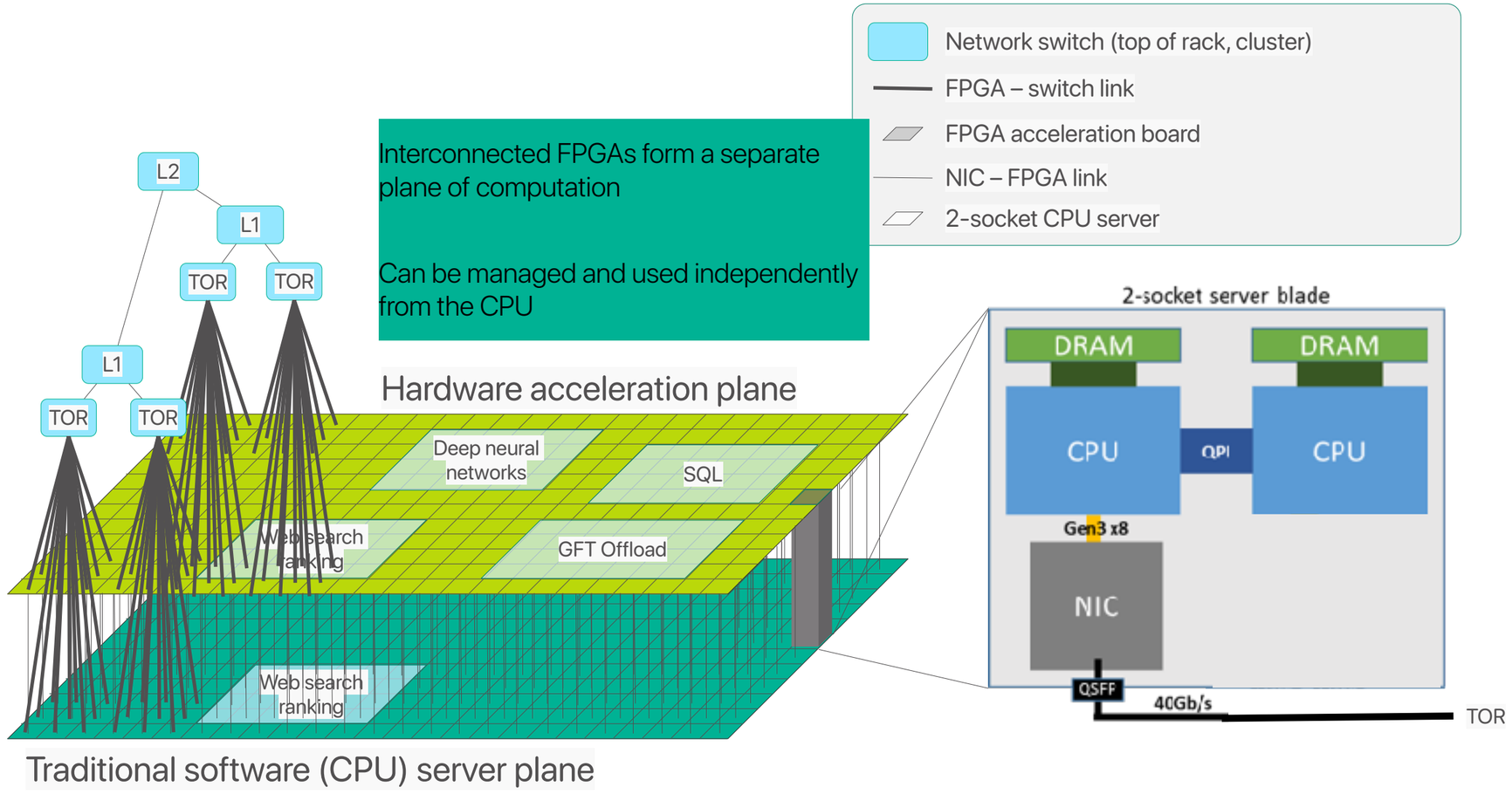
FPGA



FPGA

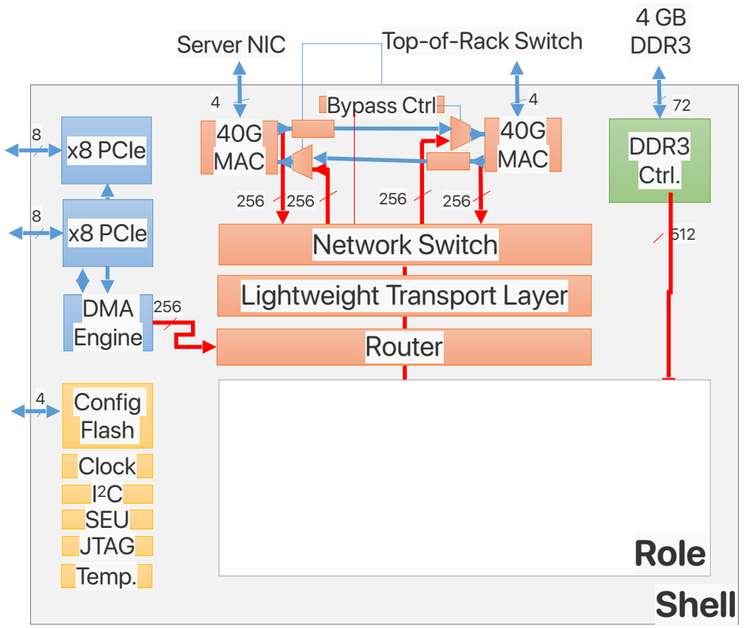
- Field Programmable Gate Array
  - An array of “Lookup tables (LUTs)”
  - Reconfigurable wires or say interconnects of LUTs
  - Registers
- An LUT
  - Accepts a few inputs
  - Has SRAM memory cells that store all possible outputs
  - Generates outputs according to the given inputs
- As a result, you may use FPGAs to emulate any kind of gates or logic combinations, and create an ASIC-like processor

Configurable cloud



Gen2 shell

- Foundation for all accelerators
  - Includes PCIe, Networking and DDR IP
  - Common, well tested platform for development
- Lightweight Transport Layer
  - Reliable FPGA-to-FPGA Networking
  - Ack/Nack protocol, retransmit buffers
  - Optimized for lossless network
  - Minimized resource usage



3:00

Why does M$ integrate FPGAs in their data centers?

Why FPGA?

This model offers significant flexibility. From the local perspective, the FPGA is used as a compute or a network accelerator. From the global perspective, the FPGAs can be managed as a large-scale pool of resources, with acceleration

hyperscale infrastructure. The acceleration system we describe is sufficiently flexible to cover three scenarios: local compute acceleration (through PCIe), network acceleration, and global application acceleration, through configuration as pools of remotely accessible FPGAs. Local acceleration handles high-

These programmable architectures allow for hardware homogeneity while allowing fungibility via softwa[illegible]ifferent services. They must be highly *flexible* at the [illegible]stem [illegible]

Flexible

This paper described Configurable Clouds, a datacenter-scale acceleration architecture, based on FPGAs, that is both scalable and flexible. By putting in FPGA cards both in I/O

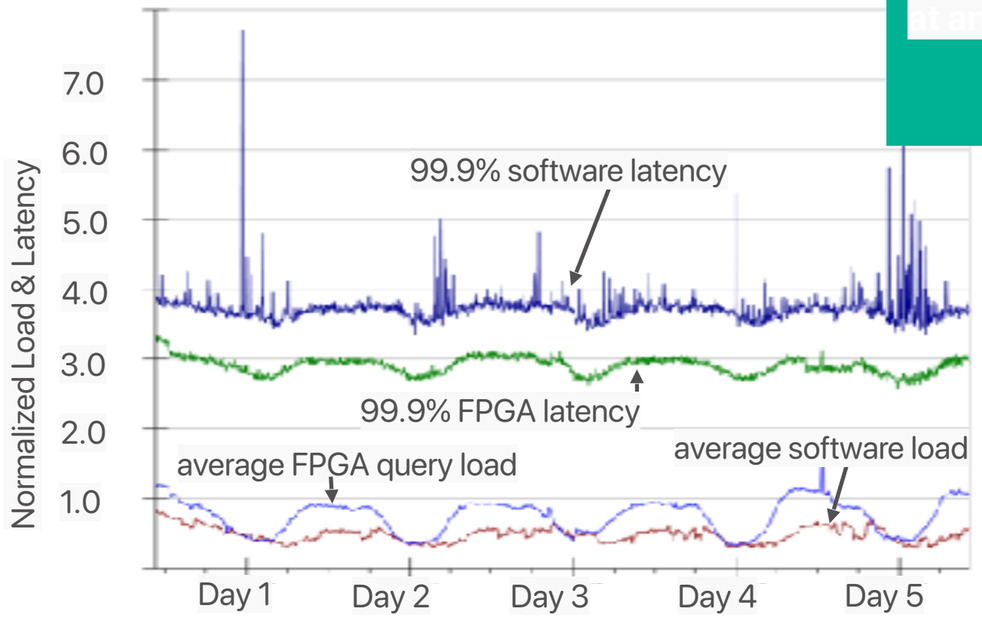
In addition to architectural requirements that provide sufficient flexibility to justify scale production deployment, there are also physical restrictions in current infrastructures that

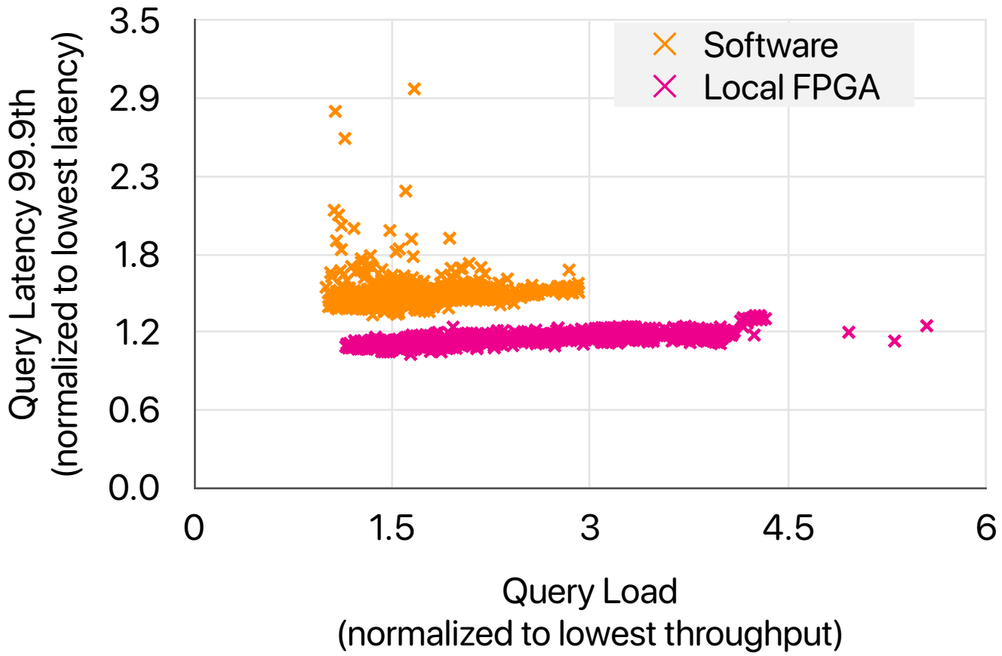
Use cases

- Local: Great service acceleration
- Infrastructure: Fastest cloud network
- Remote: Reconfigurable app fabric (DNNs)

5 day bed-level latency

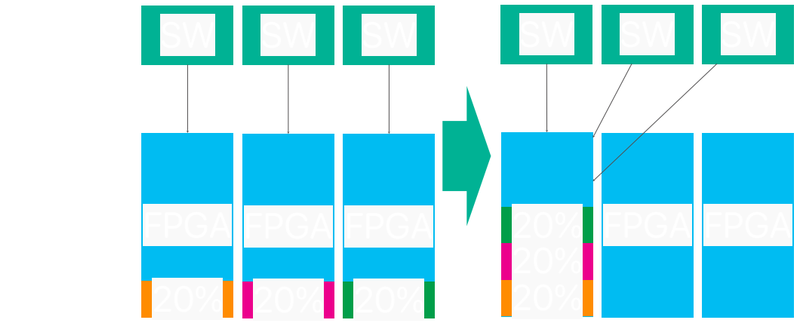
- Lower & more consistent 99.9th tail latency
- In production for years

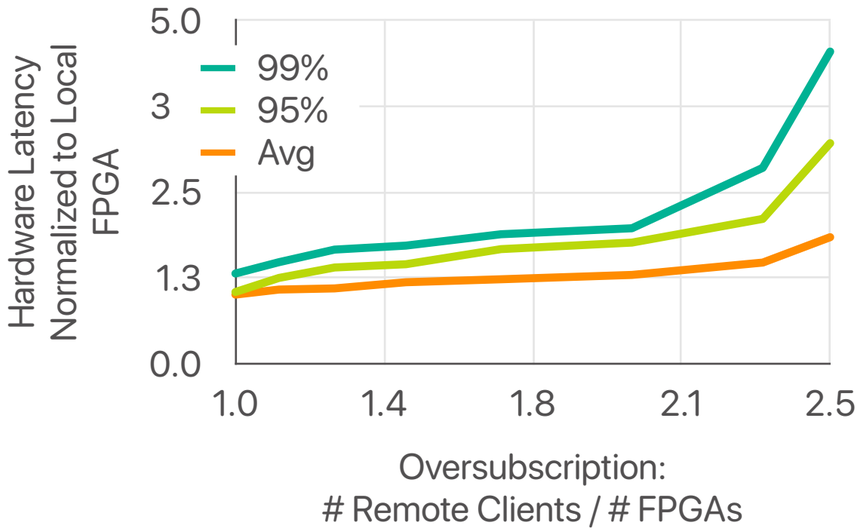




Shared DNN

- Economics: consolidation
  - Most accelerators have more throughput than a single host requires
  - Share excess capacity, use fewer instances
  - Frees up FPGAs for other use services
- DNN accelerator
  - Sustains 2.5x busy clients in microbenchmark, before queuing delay drives latency up



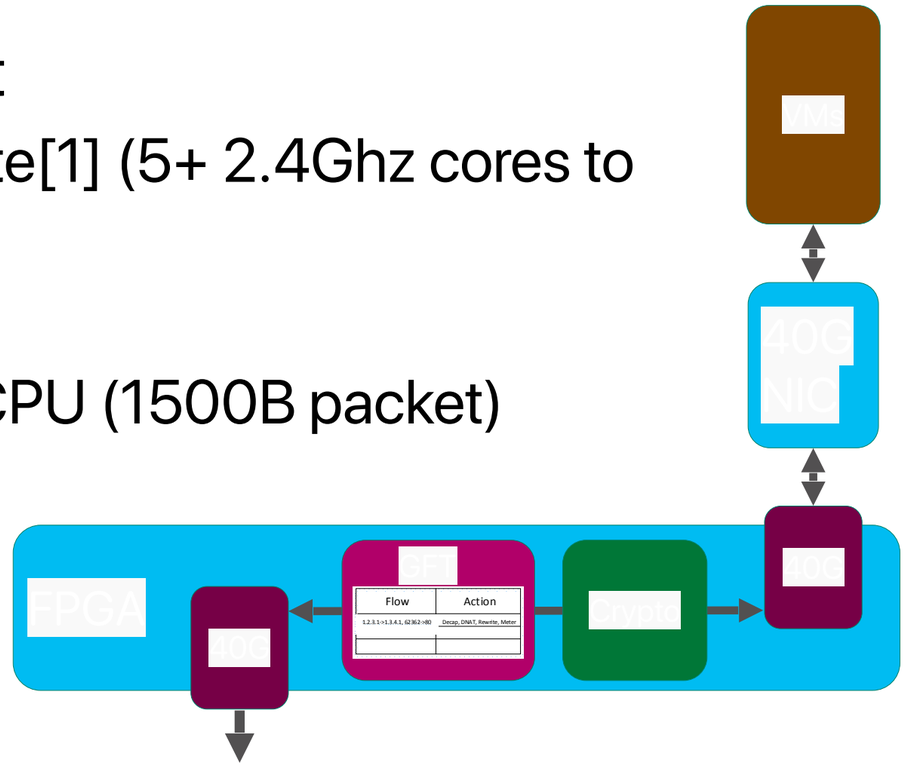


Accelerated networking

- Software defined networking
  - Generic Flow Table (GFT) rule based pack
  - 10x latency reduction vs software, CPU lo
  - 25Gb/s throughput at 25µs latency – the fastest cloud network

> Our FPGA implementation supports full 40 Gb/s encryption and decryption. The worst case half-duplex FPGA crypto latency for AES-CBC-128-SHA1 is 11 μs for a 1500B packet, from first flit to first flit. In software, based on the Intel numbers, it is approximately 4 μs. AES-CBC-SHA1 is, however,

- Capable of 40 Gb line rate encrypt and decrypt
  - On Haswell, AES GCM-128 costs 1.26 cycles/byte[1] (5+ 2.4Ghz cores to sustain 40Gb/s)
  - CBC and other algorithms are more expensive
  - AES CBC-128-SHA1 is 11µs in FPGA vs 4µs on CPU (1500B packet)
  - **Higher latency, but significant CPU savings**



How can we understand a paper within an hour?

Why, what, and how

- Why — the most important thing of a research paper
  - The big problem/question this paper is trying to address/answer
  - Why should you, as a reader, care about this paper
- What — novel ideas or interesting things to the why
  - Novel solutions to the why
  - Novel high-level design of the system
  - Novel use of an existing mechanism to solve a new problem
- How — implementation of the paper (not a contribution, but make the paper more convincing)
  - How does the implementation reflects the what?

Neural Acceleration for General-Purpose Approximate Programs

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, Doug Burger*
University of Washington and Microsoft*
In 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2012)

3:00

What's the why of "Neural Acceleration for General-Purpose Approximate Programs"

Why "Neural Acceleration for General-Purpose Approximate Programs"

- ASICs/accelerators are energy-efficient, but hard-to-program
- FPGAs are slow if memory accesses are frequent
- GPUs do not work well if the workload is not regular
- Many applications tolerate inexact computation

1. Introduction

Energy efficiency is a primary concern in computer systems. The cessation of Dennard scaling has limited recent improvements in transistor speed and energy efficiency, resulting in slowed general-purpose processor improvements. Consequently, architectural innovation has become crucial to achieve performance and efficiency gains [10].

However, there is a well-known tension between efficiency and programmability. Recent work has quantified three orders of magnitude of difference in efficiency between general-purpose processors and ASICs [21, 36]. Since designing ASICs for the massive base of quickly changing, general-purpose applications is currently infeasible, practitioners are increasingly turning to programmable accelerators such as GPUs and FPGAs. Programmable accelerators provide an intermediate point between the efficiency of ASICs and the generality of conventional processors, gaining significant efficiency for restricted domains of applications.

Programmable accelerators exploit some characteristic of an application domain to achieve efficiency gains at the cost of generality. For instance, FPGAs exploit copious, fine-grained, and irregular parallelism but perform poorly when complex and frequent accesses to memory are required. GPUs exploit many threads and SIMD-style parallelism but lose efficiency when threads diverge. Emerging accelerators, such as BERET [19], Conservation Cores and Qs-

Tolerance to approximation is one such program characteristic that is growing increasingly important. Many modern applications—such as image rendering, signal processing, augmented reality, data mining, robotics, and speech recognition—can tolerate inexact computation in substantial portions of their execution [7, 14, 28, 41]. This tolerance can be leveraged for substantial performance and energy gains.

What this paper proposed?

- Replacing the original algorithm with a machine learning model
- Learning algorithm, langage/compilation framework, architectural interface

This paper introduces a new class of programmable accelerators that exploit approximation for better performance and energy efficiency. The key idea is to *learn* how an original region of approximable code behaves and replace the original code with an efficient computation of the learned model. This approach contrasts with previous work on approximate computation that extends conventional microarchitectures to support selective approximate execution, incurring instruction bookkeeping overheads [1, 8, 11, 29], or requires vastly different programming paradigms [4, 24, 26, 32]. Like emerging flexible accelerators [18, 19, 47, 48], our technique automatically offloads code segments from programs written in mainstream languages; but unlike prior work, it leverages changes in the semantics of the offloaded code.

We have identified three challenges that must be solved to realize effective trainable accelerators:

1. A **learning algorithm** is required that can accurately and efficiently mimic imperative code. We find that neural networks can approximate various regions of imperative code and propose the Parrot transformation, which exploits this finding (Section 2).
2. A **language and compilation framework** should be developed to transform regions of imperative code to neural network evaluations. To this end, we define a programming model and implement a compilation workflow to realize the Parrot transformation (Sections 3 and 4). The Parrot transformation starts from regions of approximable imperative code identified by the programmer, collects training data, explores the topology space of neural networks, trains them to mimic the regions, and finally replaces the original regions of code with trained neural networks.
3. An **architectural interface** is necessary to call a neural processing unit (NPU) in place of the original code regions. The NPU we designed is tightly integrated with a speculative out-of-order core. The low-overhead interface enables acceleration even when fine-grained regions of code are transformed. The core communicates both the neural configurations and run-time invocations to the NPU through extensions to the ISA (Sections 5 and 6).

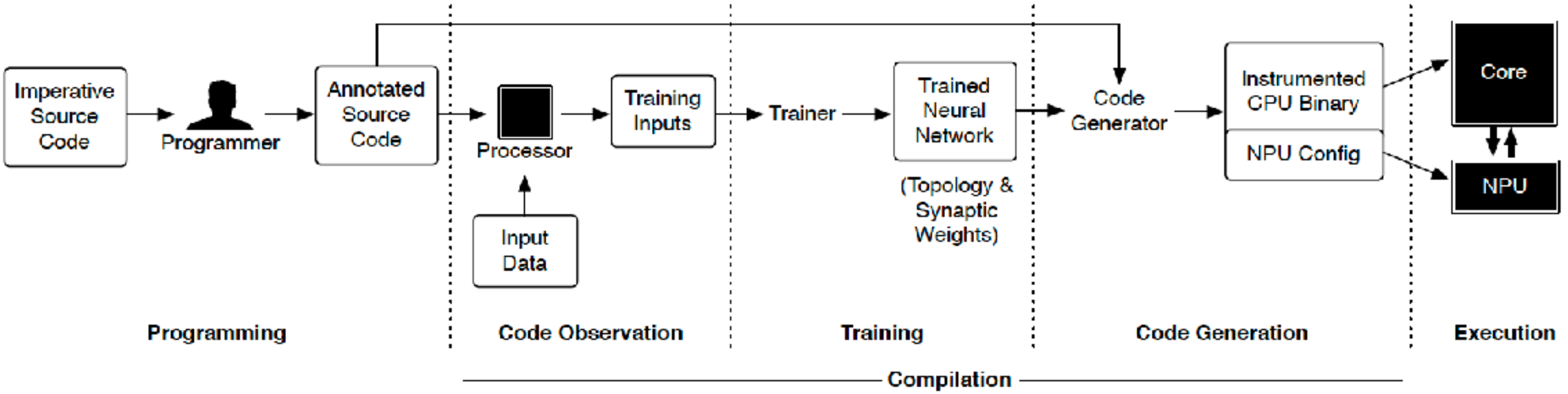


Figure 1: The Parrot transformation at a glance: from annotated code to accelerated execution on an NPU-augmented core.

Roogle Project Meetings

- Make an appointment through the Google Calendar
- We're trying to make a company project theme
- 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