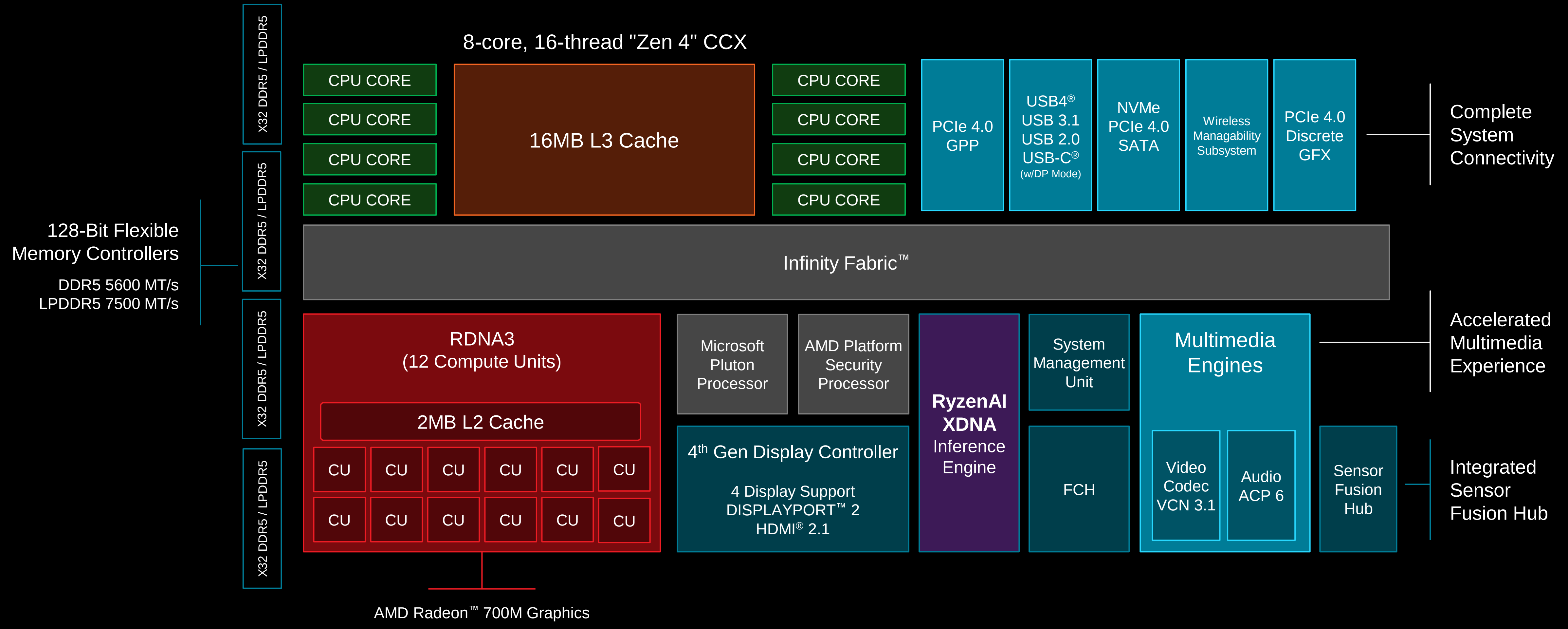
Mapping ML to the AMD RyzenAI Architecture

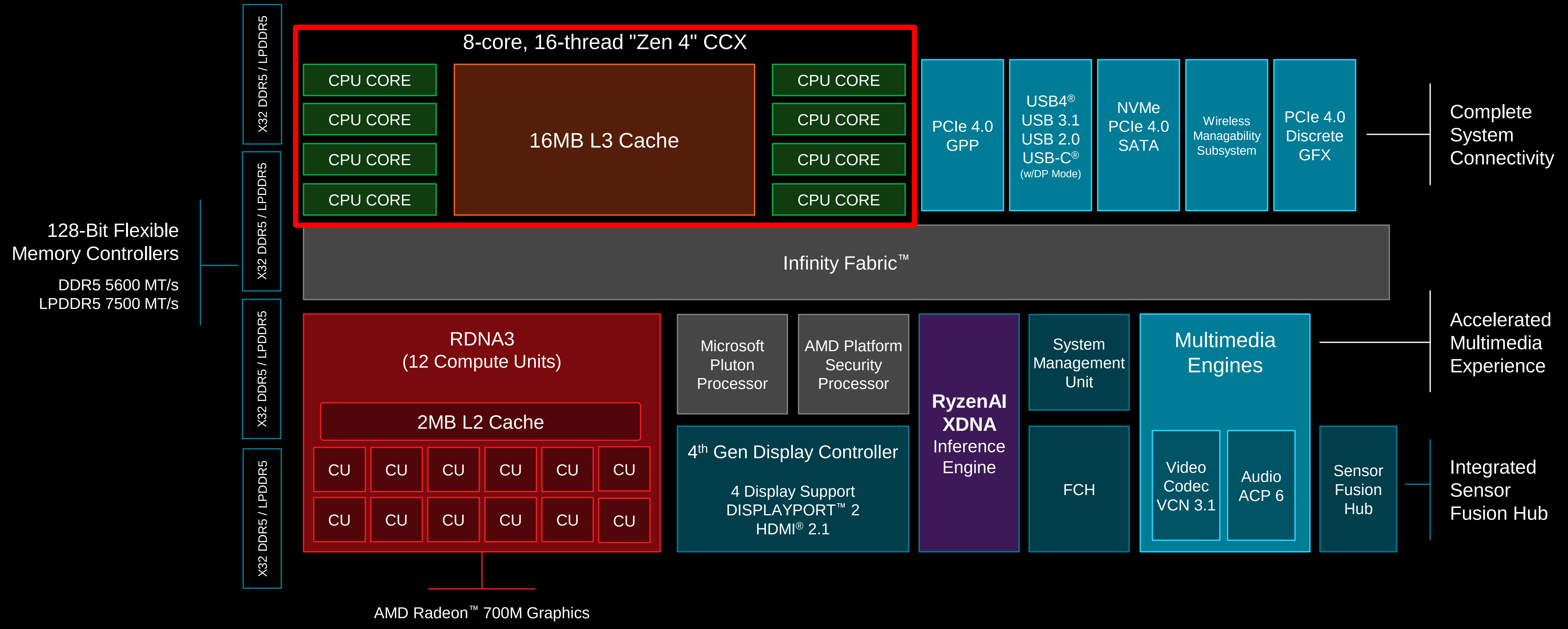
Elliott Delaye
Senior Fellow
FastML for Science, Nov 2, 2023

AMD Ryzen™ 7040 Series for Mobile – The ‘Phoenix’ SoC



* Certain capabilities and features dependent upon OEM enablement

AMD Ryzen™ 7040 Series for Mobile – The ‘Phoenix’ SoC



X32 DDR5 / LPDDR5

X32 DDR5 / LPDDR5

X32 DDR5 / LPDDR5

X32 DDR5 / LPDDR5

128-Bit Flexible
Memory Controllers

DDR5 5600 MT/s
LPDDR5 7500 MT/s

8-core, 16-thread "Zen 4" CCX

CPU CORE

CPU CORE

CPU CORE

CPU CORE

16MB L3 Cache

CPU CORE

CPU CORE

CPU CORE

CPU CORE

PCIe 4.0
GPP

USB4®
USB 3.1
USB 2.0
USB-C®
(w/DP Mode)

NVMe
PCIe 4.0
SATA

Wireless
Managability
Subsystem

PCIe 4.0
Discrete
GFX

Complete
System
Connectivity

Infinity Fabric™

RDNA3
(12 Compute Units)

2MB L2 Cache

CU CU CU CU CU CU

CU CU CU CU CU CU

AMD Radeon™ 700M Graphics

Microsoft
Pluton
Processor

AMD Platform
Security
Processor

4th Gen Display Controller

4 Display Support
DISPLAYPORT™ 2
HDMI® 2.1

**RyzenAI
XDNA**
Inference
Engine

System
Management
Unit

FCH

Multimedia
Engines

Video
Codec
VCN 3.1

Audio
ACP 6

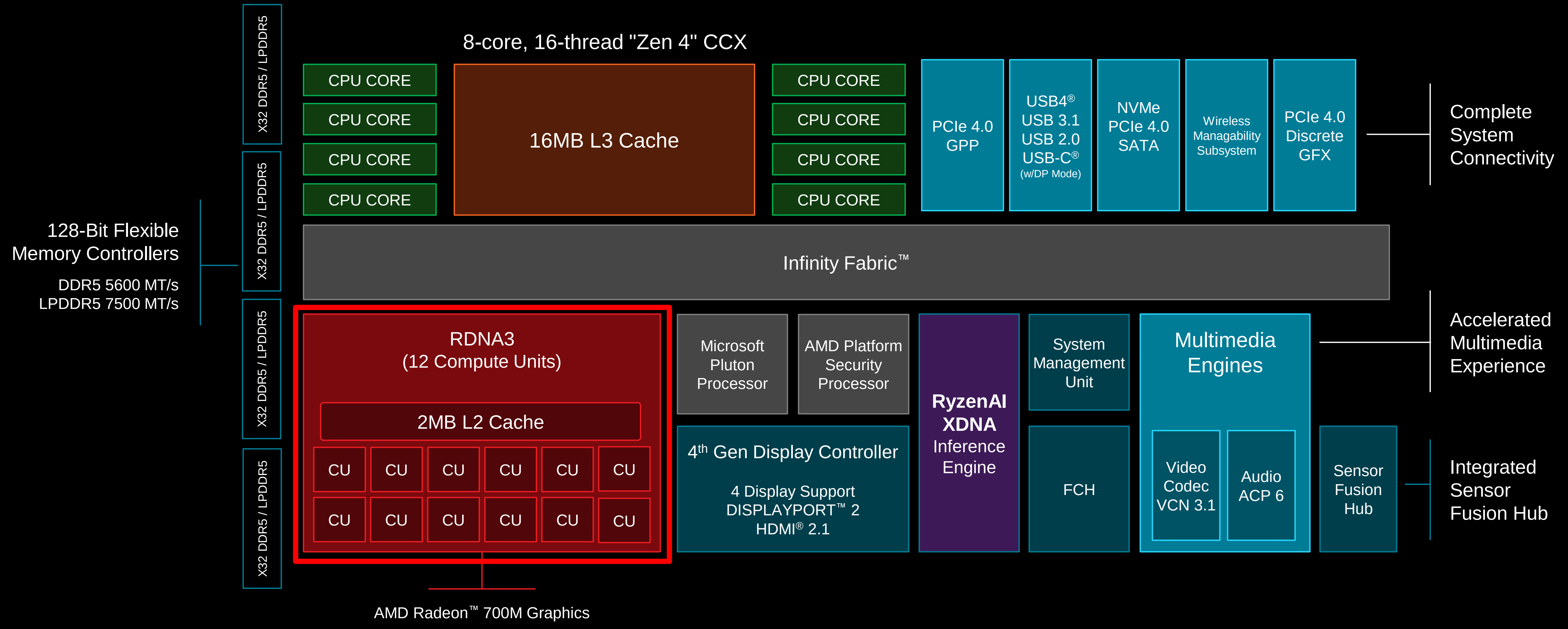
Sensor
Fusion
Hub

Accelerated
Multimedia
Experience

Integrated
Sensor
Fusion Hub

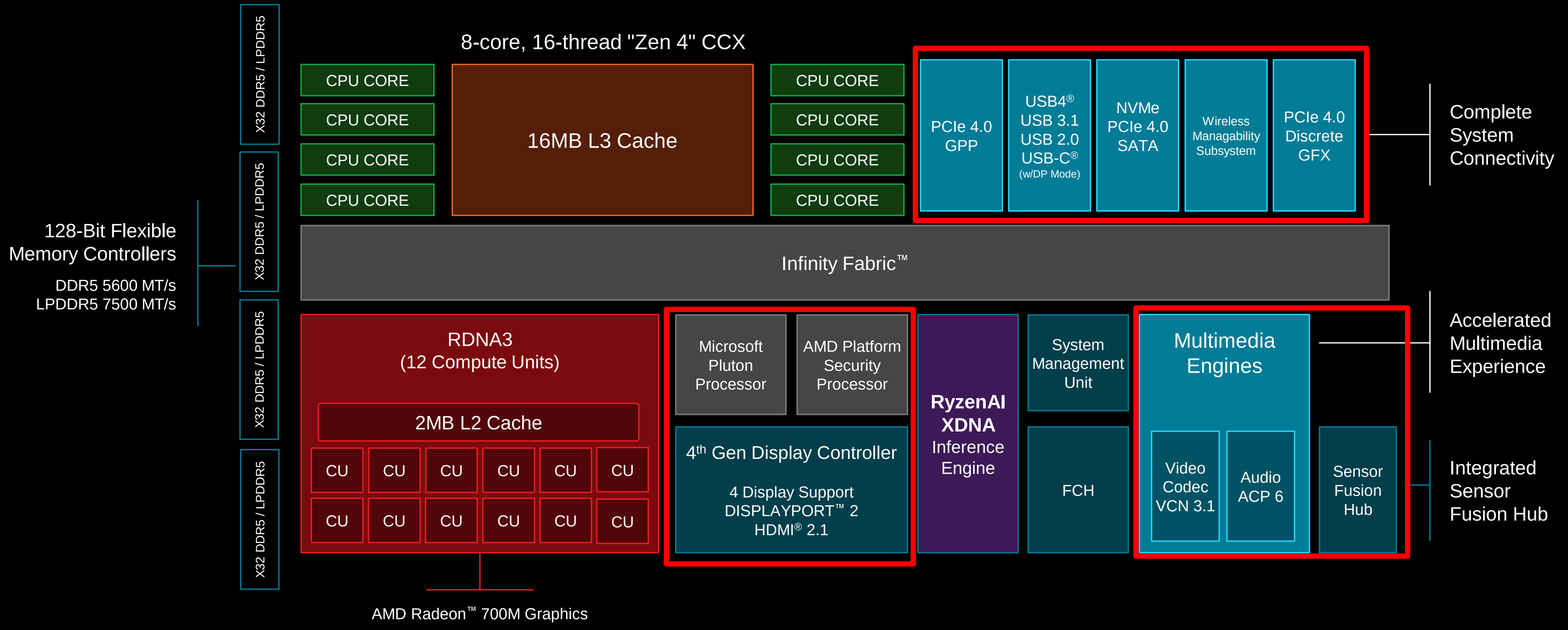
* Certain capabilities and features dependent upon OEM enablement

AMD Ryzen™ 7040 Series for Mobile – The 'Phoenix' SoC



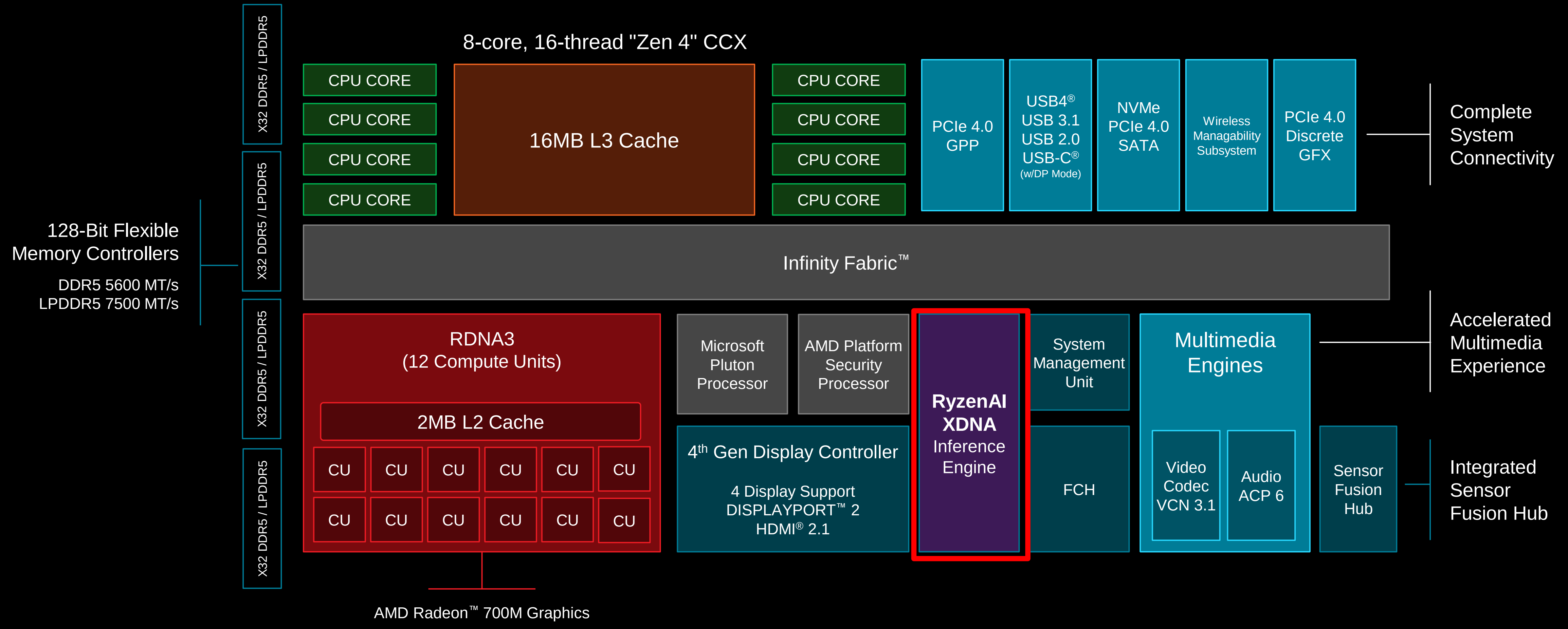
* Certain capabilities and features dependent upon OEM enablement

AMD Ryzen™ 7040 Series for Mobile – The ‘Phoenix’ SoC



* Certain capabilities and features dependent upon OEM enablement

AMD Ryzen™ 7040 Series for Mobile – The ‘Phoenix’ SoC



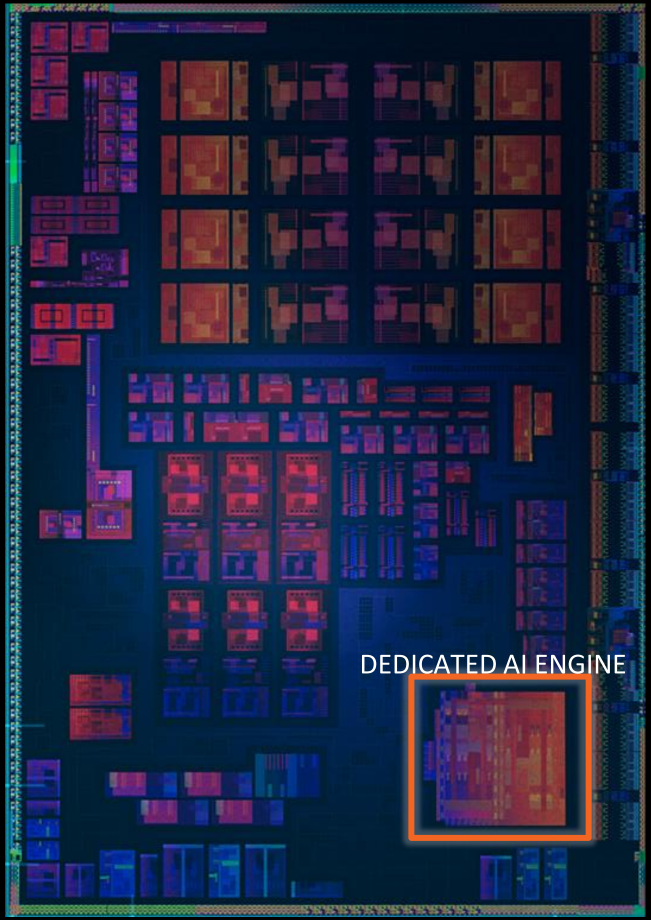
* Certain capabilities and features dependent upon OEM enablement

INTRODUCING RYZEN™ AI

RYZEN™ PRO 7040 PROCESSOR DIE = CPU + GPU + Ryzen™ AI

WORLD’S FIRST INTEGRATED AI ENGINE IN AN X86 PROCESSOR WITH **RYZEN PRO 7040 SERIES**

- **Optimized AI workloads** for best system efficiency
- Up to **4 concurrent AI streams** for real-time multi-tasking
- Processes up to **10 Trillion AI Operations Per Second**
- Experience **on-device AI** everyday on business laptops



AMD Ryzen™ 7040 Series for Mobile - Introduction

"ZEN 4" Core

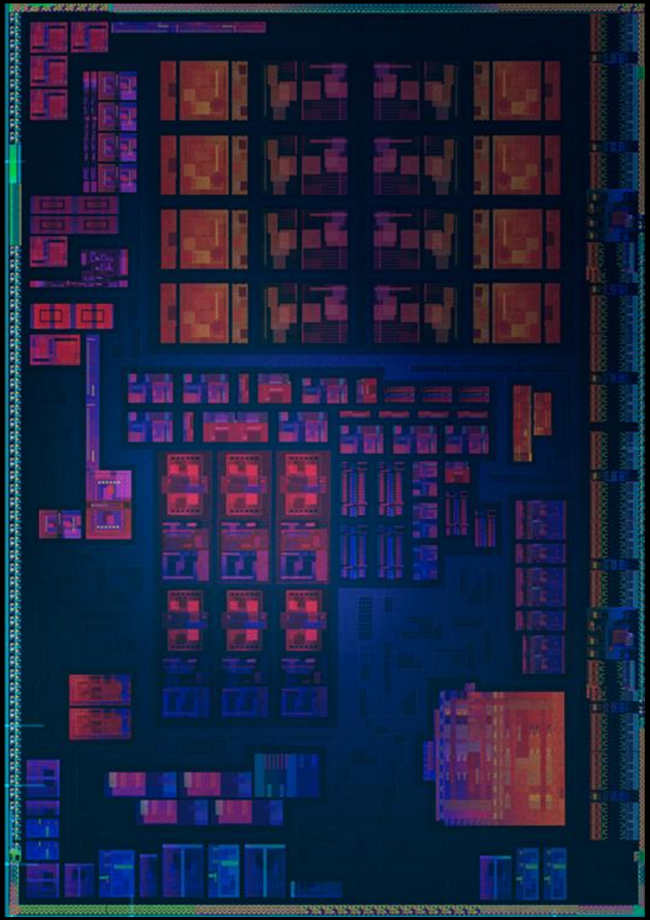
- High performance and efficient x86 cores
- Up to 13%* higher IPC

RDNA™ 3 Graphics

- Improved perf/W per compute unit

XDNA AI Engine

- IPU - Inference Processing Unit
- First integrated AI engine on an x86 processor, powering AMD Ryzen™ AI



All can run ML models… but how to choose?

* See Endnotes: RPL-005, RPL-006, PHX-3, PHX-25, PHX-29, GD-220

AMD Ryzen™ 7040 Series for Mobile - Introduction

"ZEN 4" Core

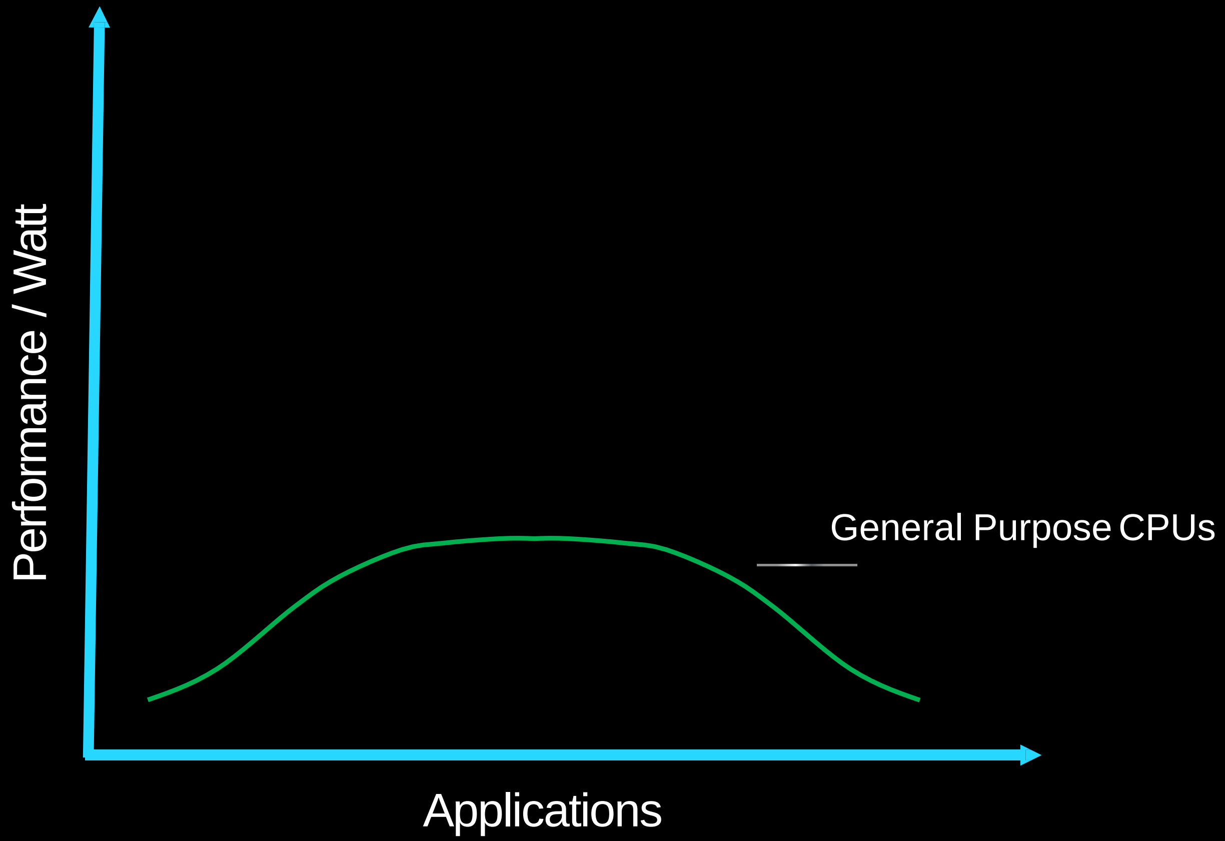
- High performance and efficient x86 cores
- Up to 13%* higher IPC

RDNA™ 3 Graphics

- Improved perf/W per compute unit

XDNA AI Engine

- IPU - Inference Processing Unit
- First integrated AI engine on an x86 processor, powering AMD Ryzen™ AI



Tailored compute for every client use-case

* See Endnotes: RPL-005, RPL-006, PHX-3, PHX-25, PHX-29, GD-220

AMD Ryzen™ 7040 Series for Mobile - Introduction

"ZEN 4" Core

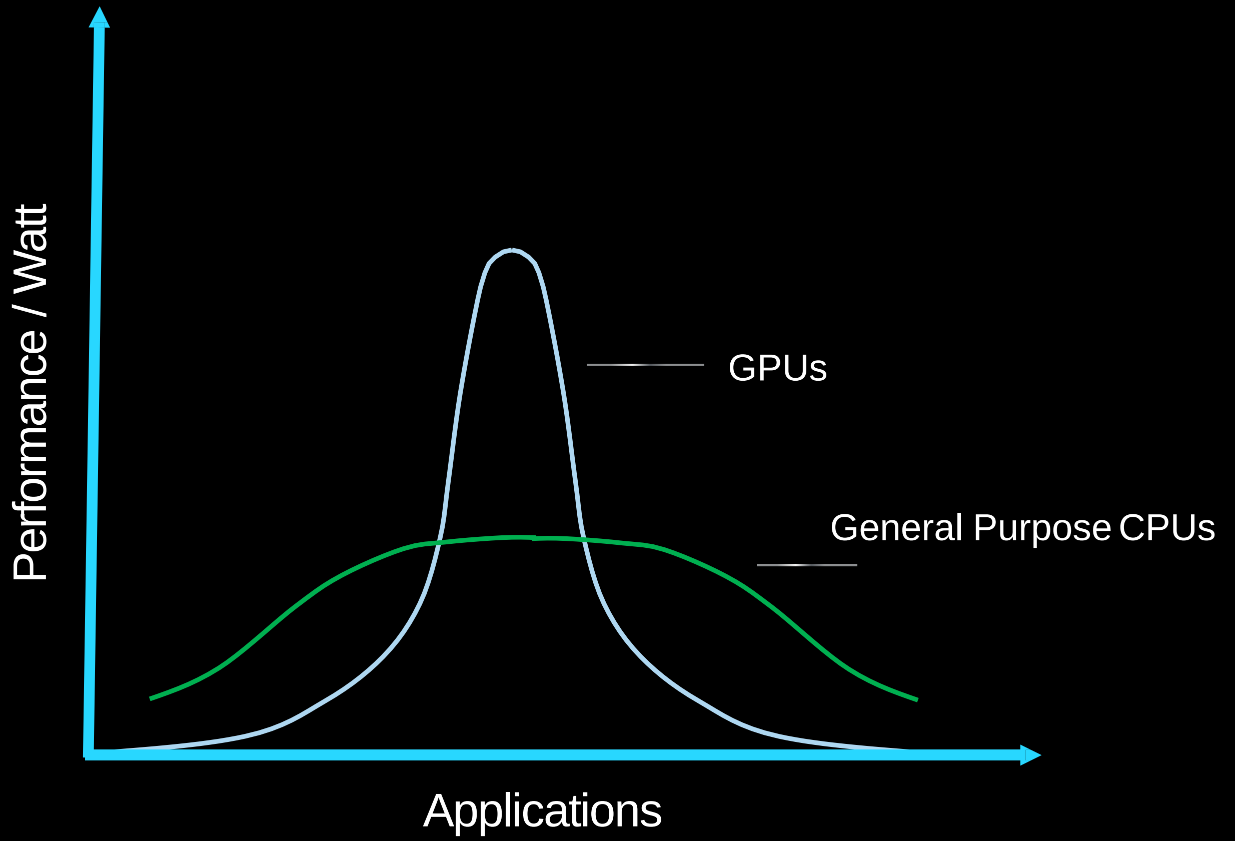
- High performance and efficient x86 cores
- Up to 13%* higher IPC

RDNA™ 3 Graphics

- Improved perf/W per compute unit

XDNA AI Engine

- IPU - Inference Processing Unit
- First integrated AI engine on an x86 processor, powering AMD Ryzen™ AI



Tailored compute for every client use-case

* See Endnotes: RPL-005, RPL-006, PHX-3, PHX-25, PHX-29, GD-220

AMD Ryzen™ 7040 Series for Mobile - Introduction

"ZEN 4" Core

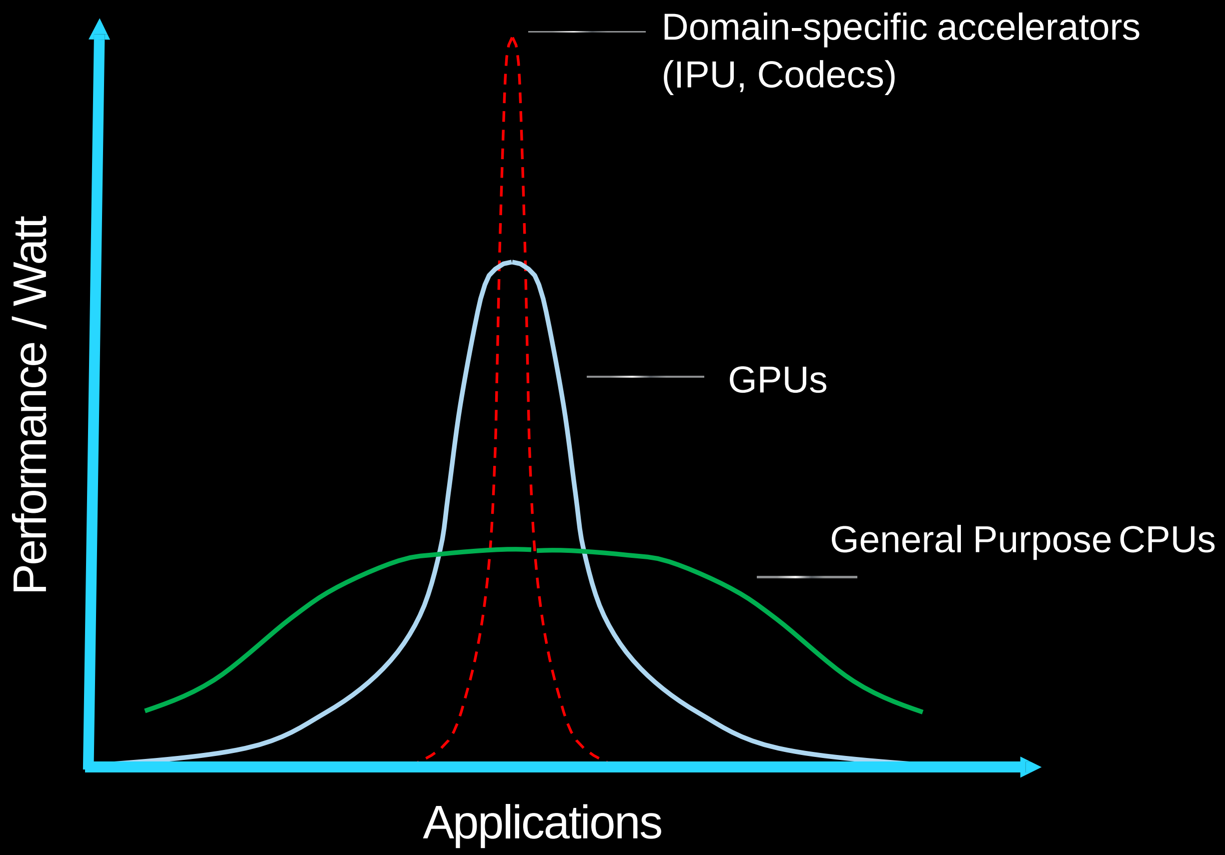
- High performance and efficient x86 cores
- Up to 13%* higher IPC

RDNA™ 3 Graphics

- Improved perf/W per compute unit

XDNA AI Engine

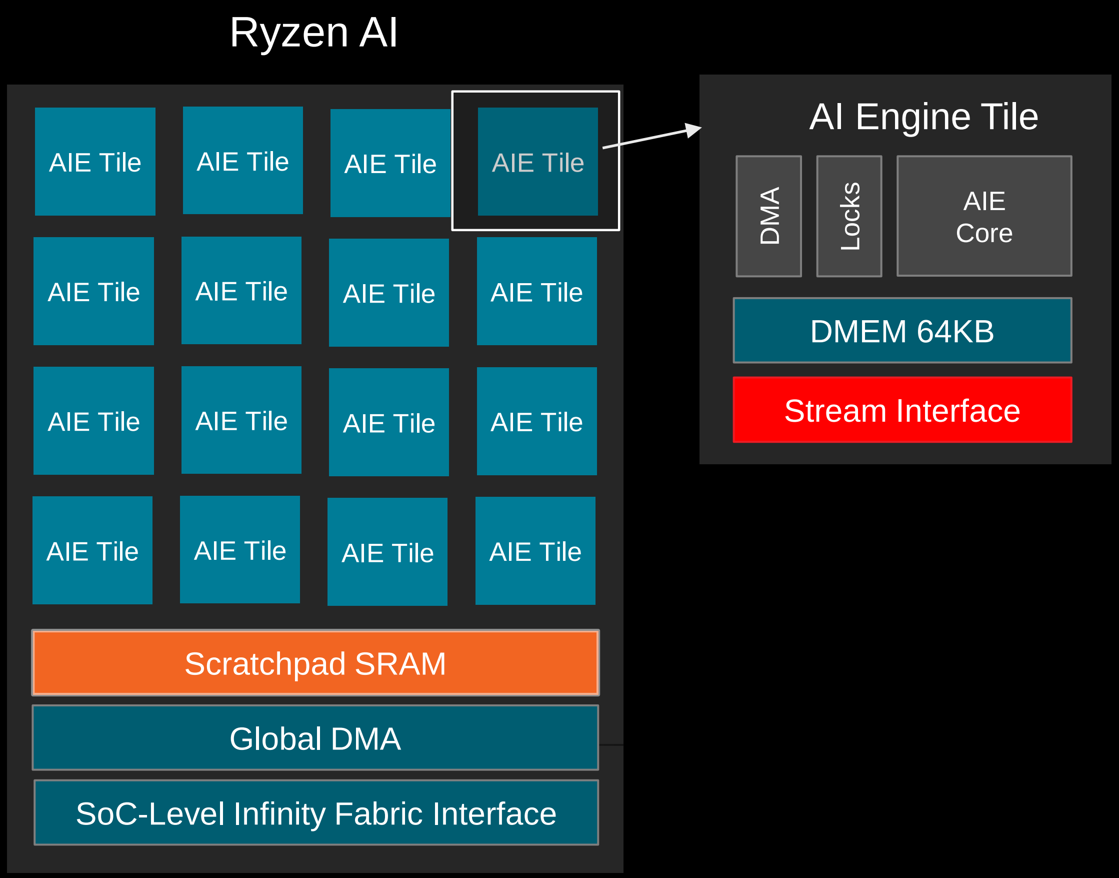
- IPU - Inference Processing Unit
- First integrated AI engine on an x86 processor, powering AMD Ryzen™ AI



Tailored compute for every client use-case

* See Endnotes: RPL-005, RPL-006, PHX-3, PHX-25, PHX-29, GD-220

XDNA Architecture



XDNA Architecture Capabilities

Broad AI Model Support
Transformers, CNNs, others

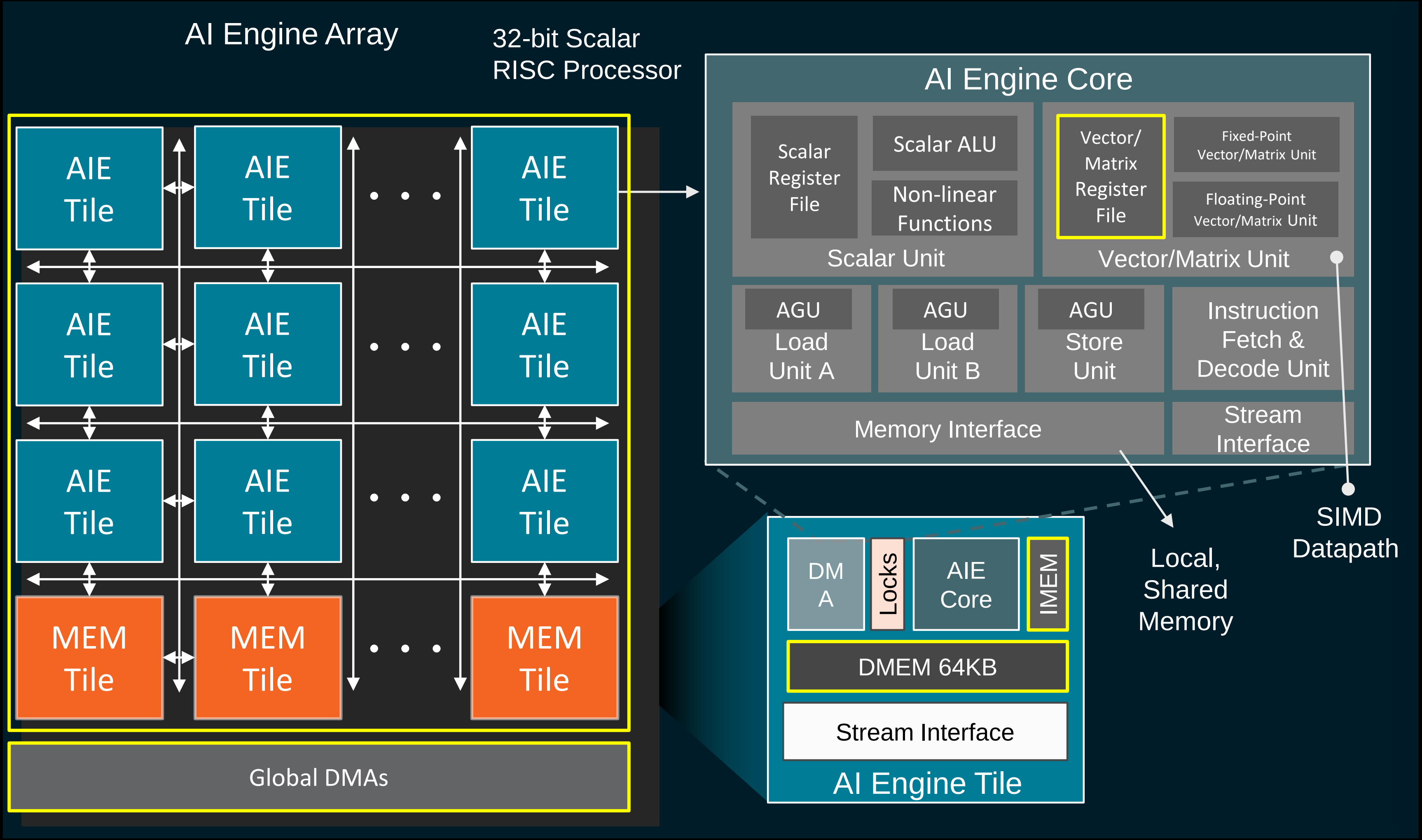
First Generation XDNA on Ryzen 7040 as 5x4 Array
Up to 12.5 INT8 TOPs, 25 INT4 TOPs, 6.25 BF16 TFLOPs

Real-time Performance
Up to 4 concurrent spatial streams (DNN DPU)

Advanced Features
50% weight sparsity

Power Efficiency Features
Fine-grained clock gating

XDNA AIE-ML Core Tile



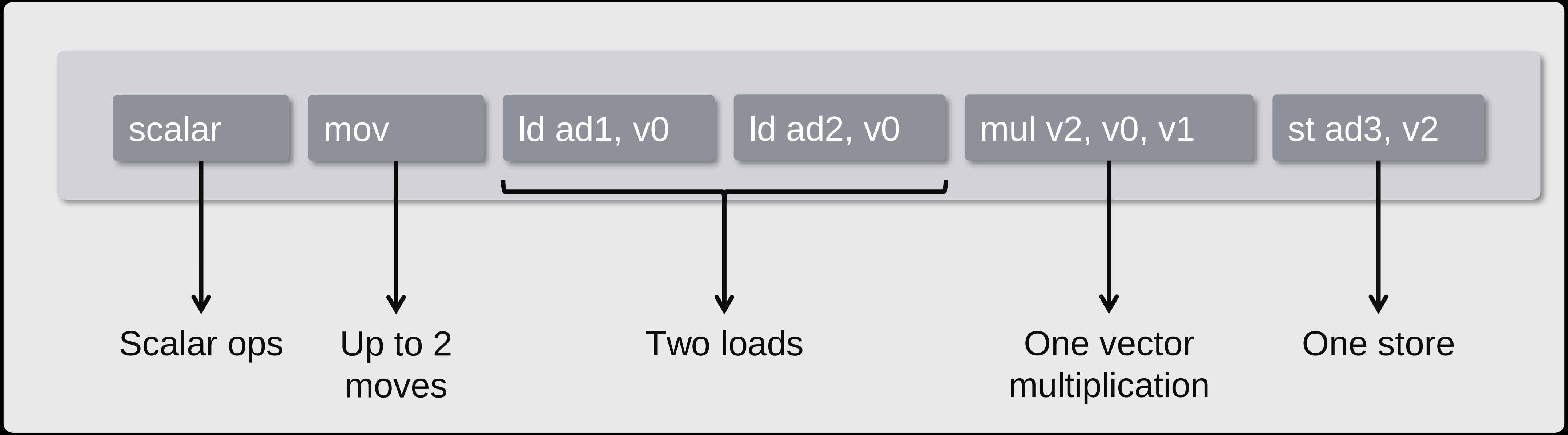
Based on same AIE-ML in AMD Versal Devices

Key Features

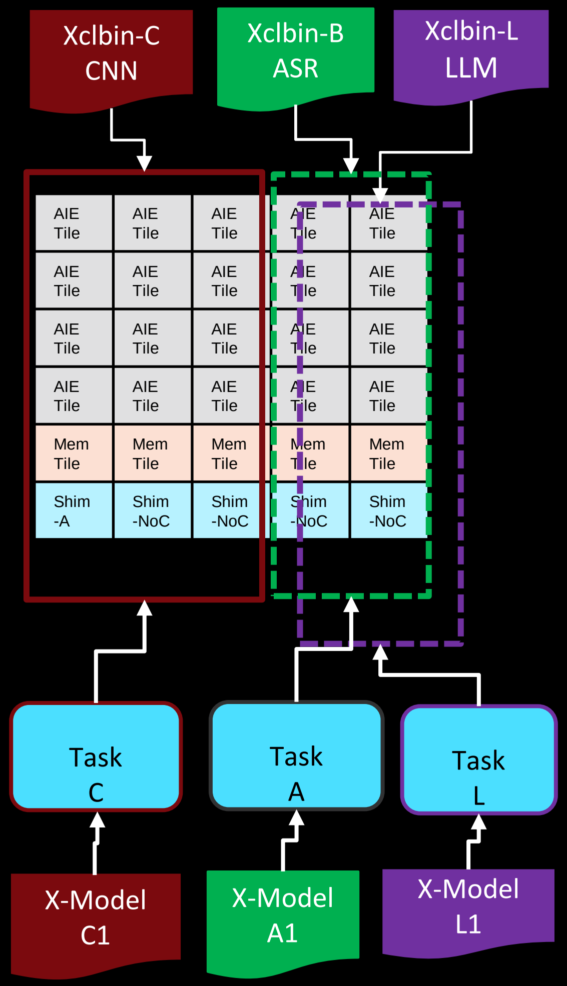
- Array of AIE-ML Compute tiles with SIMD/Scalar datapaths, local memory
- Full NSEW connectivity to neighbors
- Streaming channels running full vertical/horizontal length of the array
- Dedicated SRAM within the array for shared tile access

Multiple levels of parallelism

- Instruction-level parallelism: multiple operations in one cycle
- Data-level parallelism: vector data path (SIMD)
- Processor-level parallelism: Array of AI Engines



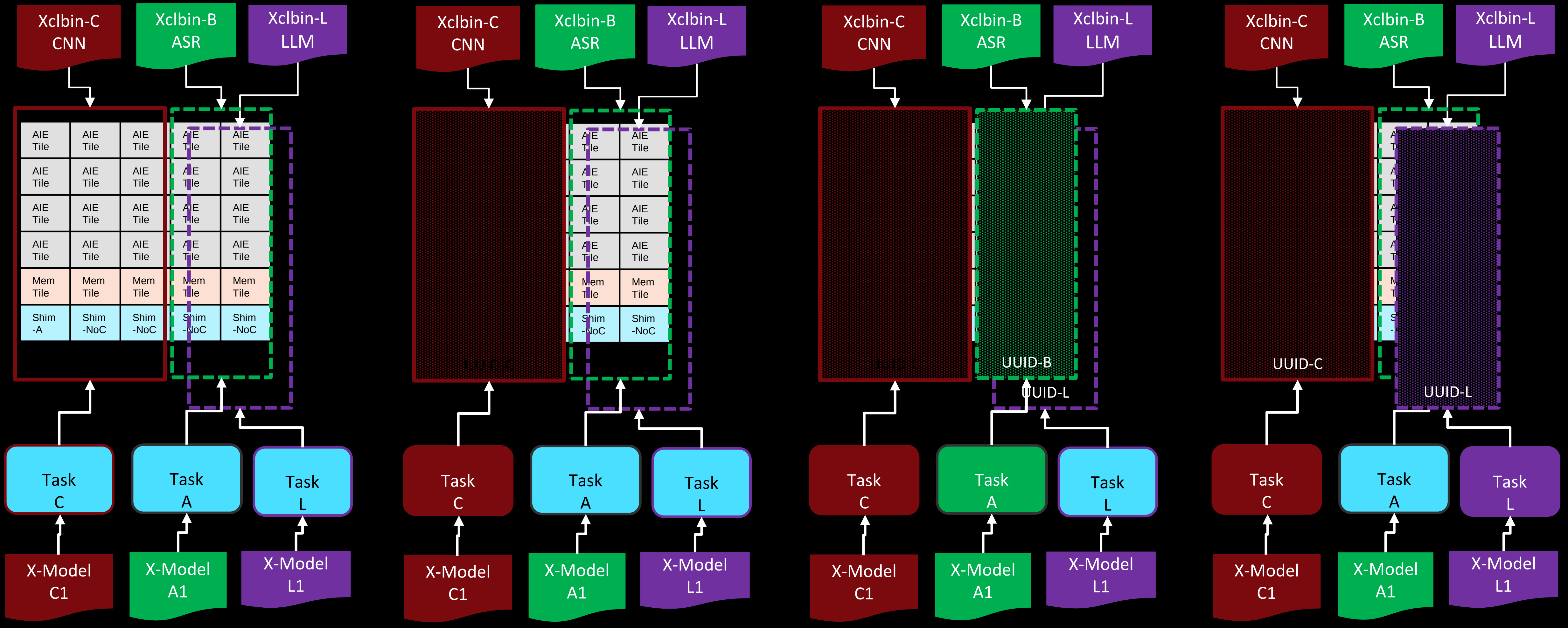
Spatial and Temporal Execution using XDNA



ASR & LLM provisioned for temporal sharing; CNN provisioned a dedicated partition

- Architecture designed to allow multiple spatial partitions of variable sizes
- Temporal sharing of resources scheduled through runtime
- Real time and deterministic processing within the array

Spatial and Temporal Execution using XDNA



ASR & LLM provisioned for temporal sharing; CNN provisioned a dedicated partition

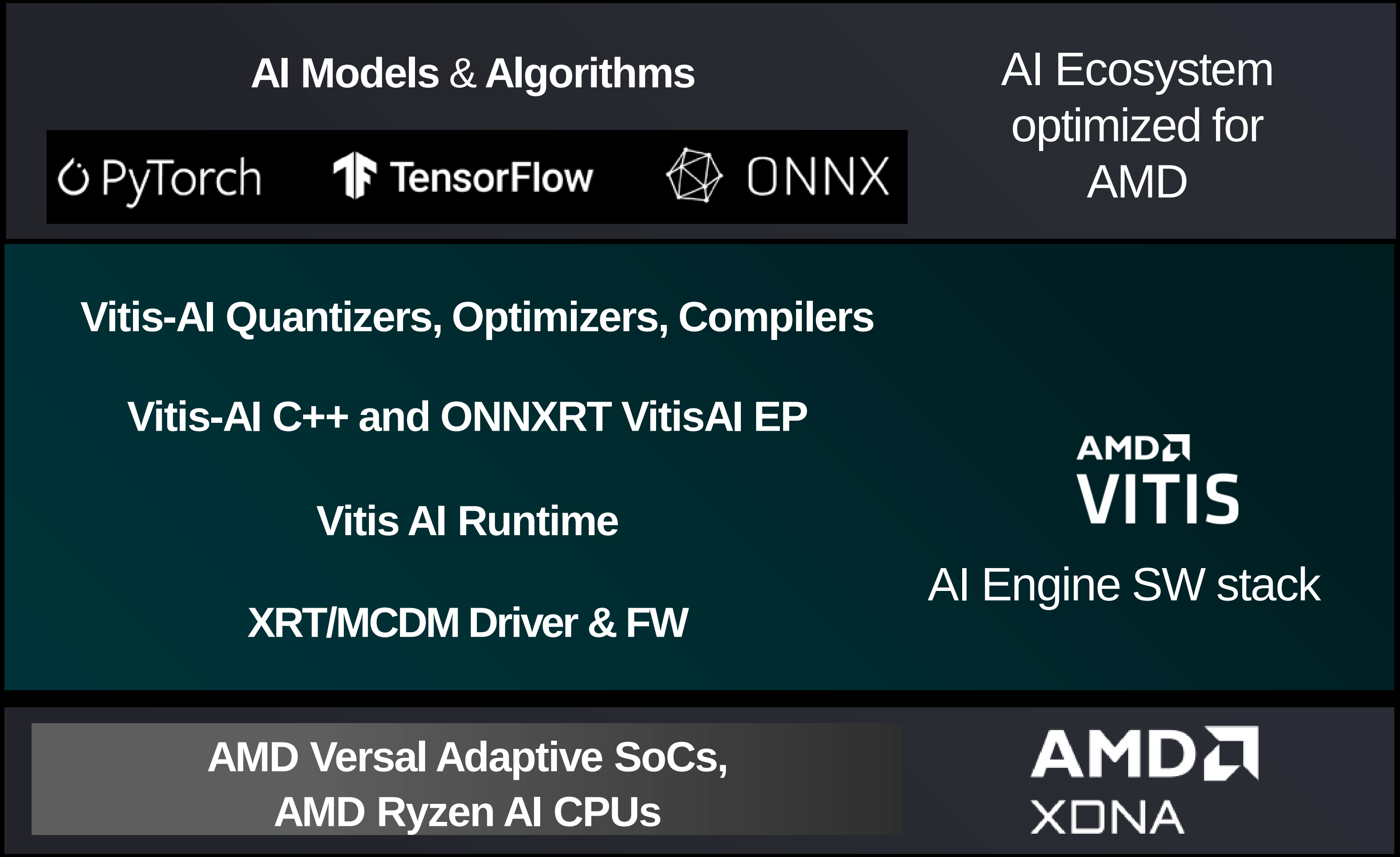
CNN loaded and running

ASR loaded and running;
CNN running uninterrupted

ASR paused; LSTM loaded and running
CNN running uninterrupted

XDNA Cloud to Client + Embedded SW SOLUTION

AMD VITIS



AI Models & Algorithms

PyTorch TensorFlow ONNX

AI Ecosystem optimized for AMD

Vitis-AI Quantizers, Optimizers, Compilers

Vitis-AI C++ and ONNXRT VitisAI EP

Vitis AI Runtime

XRT/MCDM Driver & FW

AMD VITIS

AI Engine SW stack

AMD Versal Adaptive SoCs,
AMD Ryzen AI CPUs

AMD XDNA

- Out-of-the-box support for broad models & operators
- Generative AI support
- Developer enablement & application support

The Tensor Expression Language: Describing GEMM

HIGH LEVEL DESCRIPTION

```
a = relay.var(‘a', shape=(1024, 1024))
b = relay.var(‘b', shape=(1024, 1024))
c = relay.nn.dense(a, b)
```

TENSOR EXPRESSION

```
A = te.placeholder((1024, 1024), name=‘A’)
B = te.placeholder((1024, 1024), name=‘B’)
k = te.reduce_axis((0, 1024), "k")
C = te.compute((1024, 1024), lambda x, y: te.sum(A[x, k] * B[k, y], axis=k), name=“C”)
s = te.create_schedule(C.op)
```

GENERATED TENSOR IR

```
for (x: int32, 0, 1024) {
  for (y: int32, 0, 1024) {
    C[((x*1024) + y)] = 0f32
    for (k: int32, 0, 1024) {
      C[((x*1024) + y)] = ((float32*)C[((x*1024) + y)]
        + ((float32*)A[((x*1024) + k)]*(float32*)B[((k*1024) + y)]))
    }
  }
}
```

The Tensor Expression Language: Scheduling the Operator

TENSOR EXPRESSION

```
…
C = te.compute((1024, 1024), lambda x, y: te.sum(A[x, k] * B[k, y], axis=k), name=“C”)
s = te.create_schedule(C.op)
xo, yo, xi, yi = s[C].tile(C.op.axis[0], C.op.axis[1], 32, 32)
(k,) = s[C].op.reduce_axis
ko, ki = s[C].split(k, factor=4)
s[C].reorder(xo, yo, ko, ki, xi, yi)
```

GENERATED TENSOR IR

```
for (x.outer: int32, 0, 32)
  for (y.outer: int32, 0, 32)
    for (x.inner.i: int32, 0, 32)
      for (y.inner.i: int32, 0, 32)
        C[(((x.outer*32768) + (x.inner.i*1024)) + (y.outer*32)) + y.inner.i] = 0f32
  for (k.outer: int32, 0, 256)
    for (k.inner: int32, 0, 4)
      for (x.inner: int32, 0, 32)
        for (y.inner: int32, 0, 32)
          C[(((x.outer*32768) + (x.inner*1024)) + (y.outer*32)) + y.inner] +=
            (float32*)A[(((x.outer*32768) + (x.inner*1024)) + (k.outer*4)) + k.inner]
           *(float32*)B[(((k.outer*4096) + (k.inner*1024)) + (y.outer*32)) + y.inner]
```

Scheduling operators
Tiling
Reordering
Iterator splitting/fusion
Vectorization
Tensorization
Inserting caches
Inserting pragmas
…

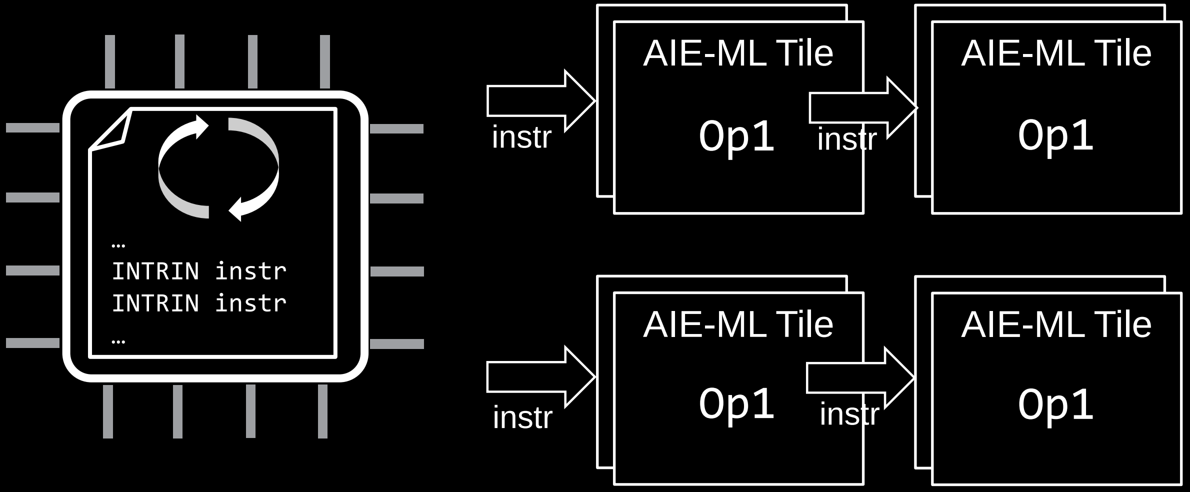
The Tensor Expression Language: Tensorization

- Insert calls to implemented AIE kernel intrinsics (e.g. 64x128x64 GEMM, ReLU etc.)
- This intrinsic call will be handled and initiated by the AIE uController

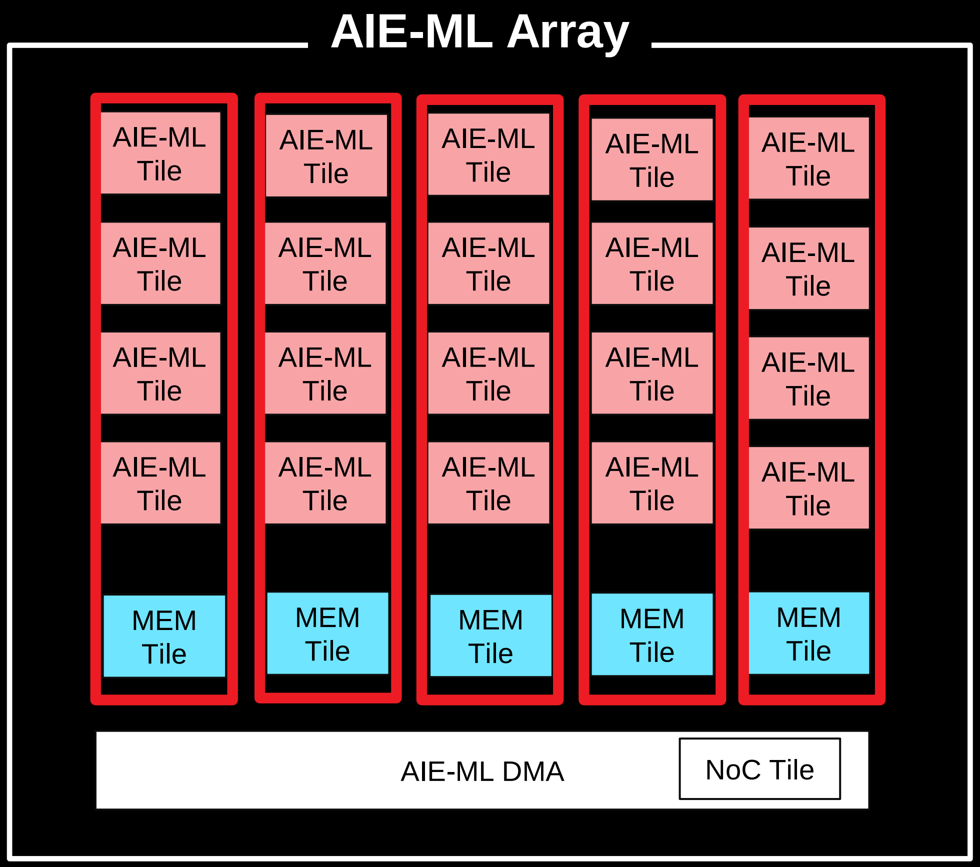
```
for (int xo=0; xo<2; xo++)
  for (int yo=0; yo<2; yo++)
   ⎧for (int xi=0; xi<2; xi++)
 I ⎨  for (int yi=0; yi<2; yi++)
   ⎩    out[xo*2+xi, yo*2+yi] = …
```

⇩

```
for (int xo=0; xo<2; xo++)
  for (int yo=0; yo<2; yo++)
    call_aie_intrin(
      op1, xo, yo, …
    )
```



Case Study: GEMM



- Maximize bandwidth to achieve peak TOPS
- Asymmetry in N/S vs E/W connections
- Think of array as set of column processors
  - Fuse columns if partitioning compute
- Compute vertically and cascade horizontally
  - Broadcast shared activations vertically

Spatial/Temporal Tiling

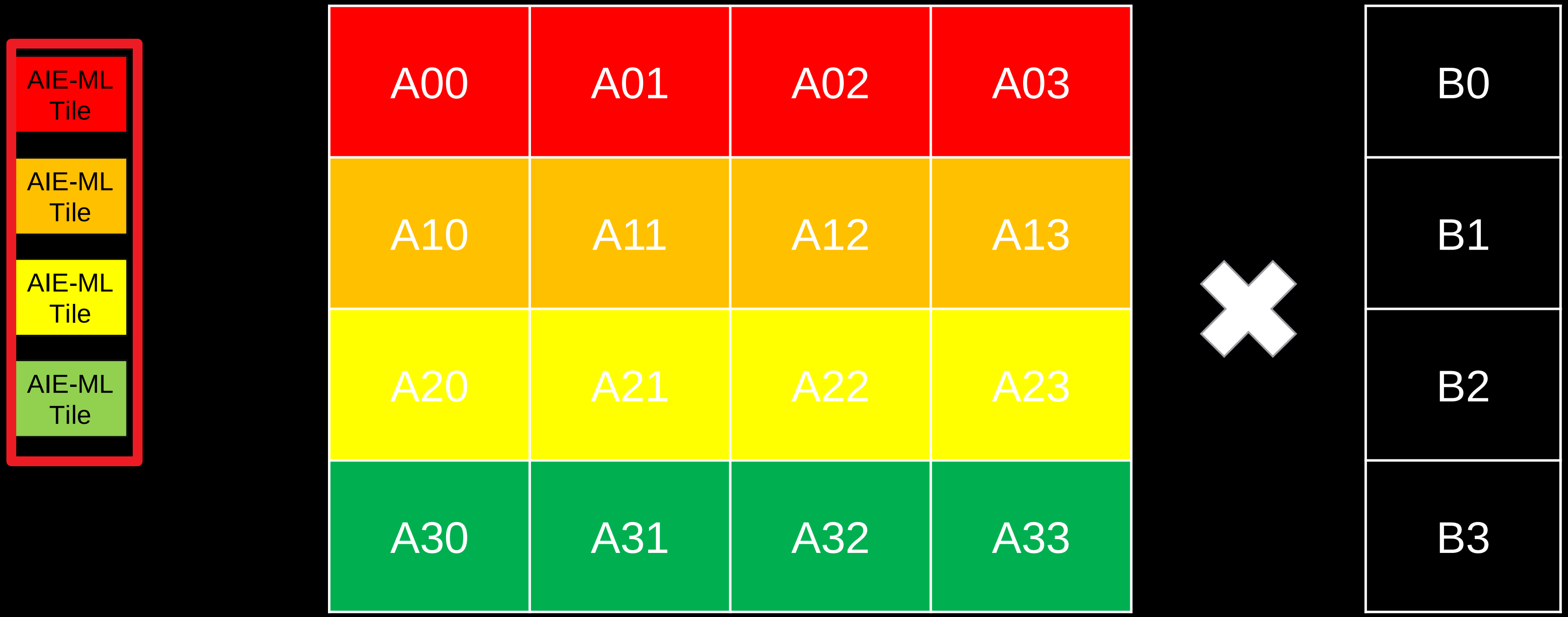
A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

✖

B0
B1
B2
B3

Spatial/Temporal Tiling

Example computing inner products



AIE-ML Tile

AIE-ML Tile

AIE-ML Tile

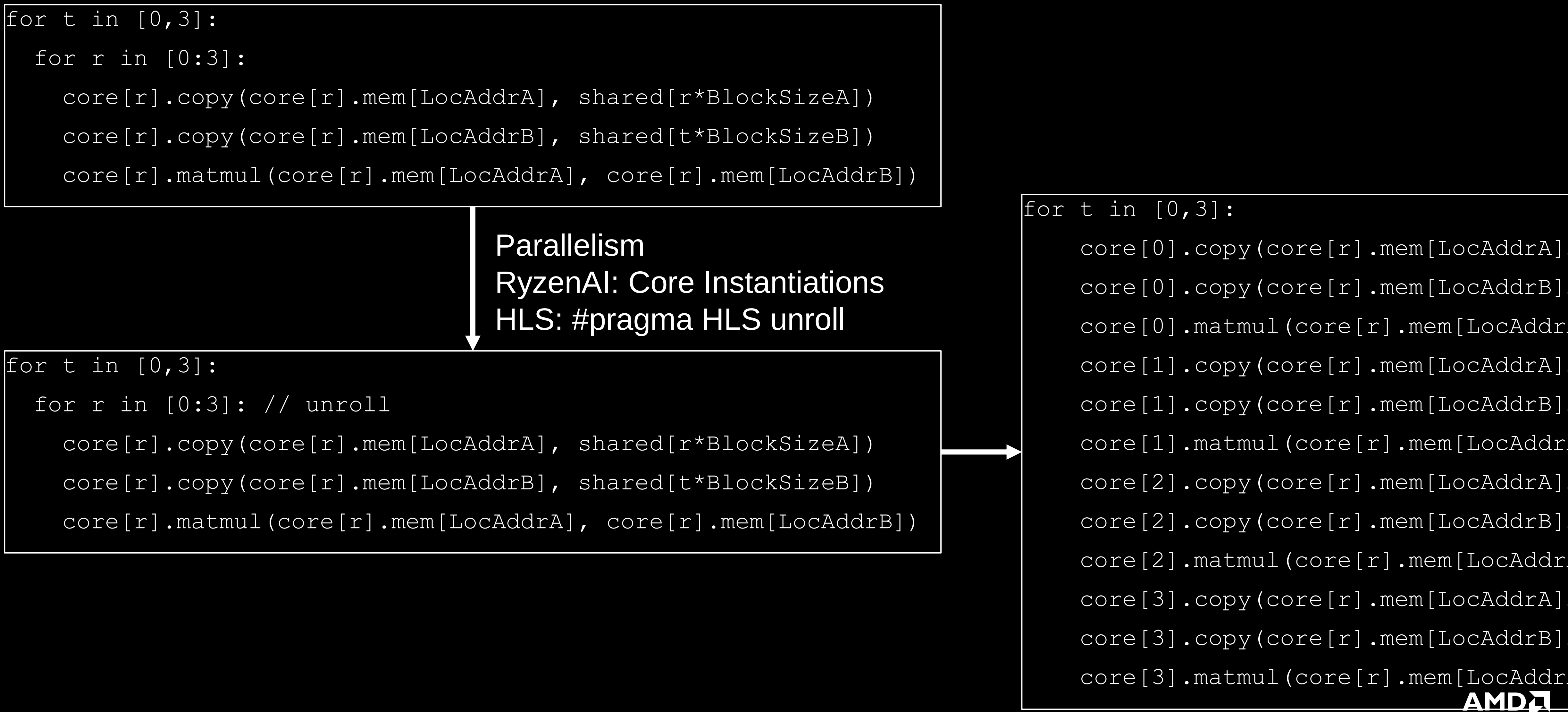
AIE-ML Tile

A00	A01	A02	A03
A10	A11	A12	A13
A20	A21	A22	A23
A30	A31	A32	A33

×

B0
B1
B2
B3

RyzenAI Array Programming / HLS Similarities



```
for t in [0,3]:
  for r in [0:3]:
    core[r].copy(core[r].mem[LocAddrA], shared[r*BlockSizeA])
    core[r].copy(core[r].mem[LocAddrB], shared[t*BlockSizeB])
    core[r].matmul(core[r].mem[LocAddrA], core[r].mem[LocAddrB])
```

Parallelism
RyzenAI: Core Instantiations
HLS: #pragma HLS unroll

```
for t in [0,3]:
  for r in [0:3]: // unroll
    core[r].copy(core[r].mem[LocAddrA], shared[r*BlockSizeA])
    core[r].copy(core[r].mem[LocAddrB], shared[t*BlockSizeB])
    core[r].matmul(core[r].mem[LocAddrA], core[r].mem[LocAddrB])
```

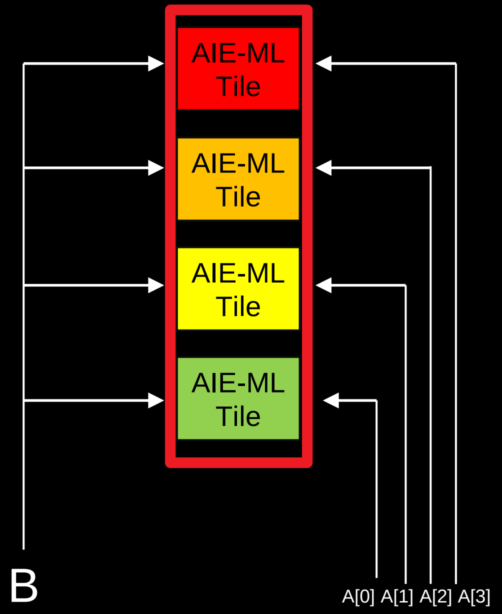
```
for t in [0,3]:
    core[0].copy(core[r].mem[LocAddrA]
    core[0].copy(core[r].mem[LocAddrB]
    core[0].matmul(core[r].mem[LocAddr
    core[1].copy(core[r].mem[LocAddrA]
    core[1].copy(core[r].mem[LocAddrB]
    core[1].matmul(core[r].mem[LocAddr
    core[2].copy(core[r].mem[LocAddrA]
    core[2].copy(core[r].mem[LocAddrB]
    core[2].matmul(core[r].mem[LocAddr
    core[3].copy(core[r].mem[LocAddrA]
    core[3].copy(core[r].mem[LocAddrB]
    core[3].matmul(core[r].mem[LocAddr
```

RyzenAI Array Programming / HLS Similarities

```
for t in [0,3]:
  for r in [0:3]:
    core[r].copy(core[r].mem[LocAddrA], shared[r*BlockSizeA])
    core[r].copy(core[r].mem[LocAddrB], shared[t*BlockSizeB])
    core[r].matmul(core[r].mem[LocAddrA], core[r].mem[LocAddrB])
```

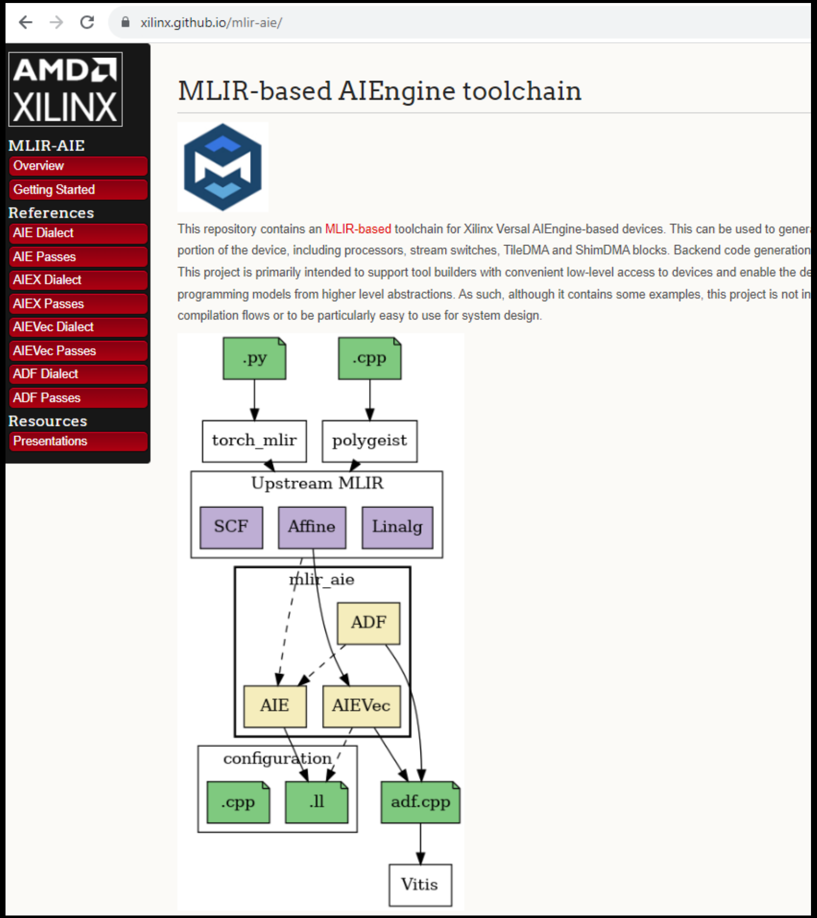
Broadcast
RyzenAI: stream broadcast
HLS: Wire Fanout

```
for t in [0,3]:
 for r in [0:3]:
    core[r].copy(core[r].mem[LocAddrB], shared[t*BlockSizeB])
for r in [0:3]:
    core[r].copy(core[r].mem[LocAddrA], shared[r*BlockSizeA])
    core[r].copy(core[r].mem[LocAddrB], shared[t*BlockSizeB])
    core[r].matmul(core[r].mem[LocAddrA], core[r].mem[LocAddrB])
```



Open Source for RyzenAI / AIE-ML

- MLIR dialects from AMD Research Labs
- Look for upcoming open source release with tutorials on directly programming RyzenAI

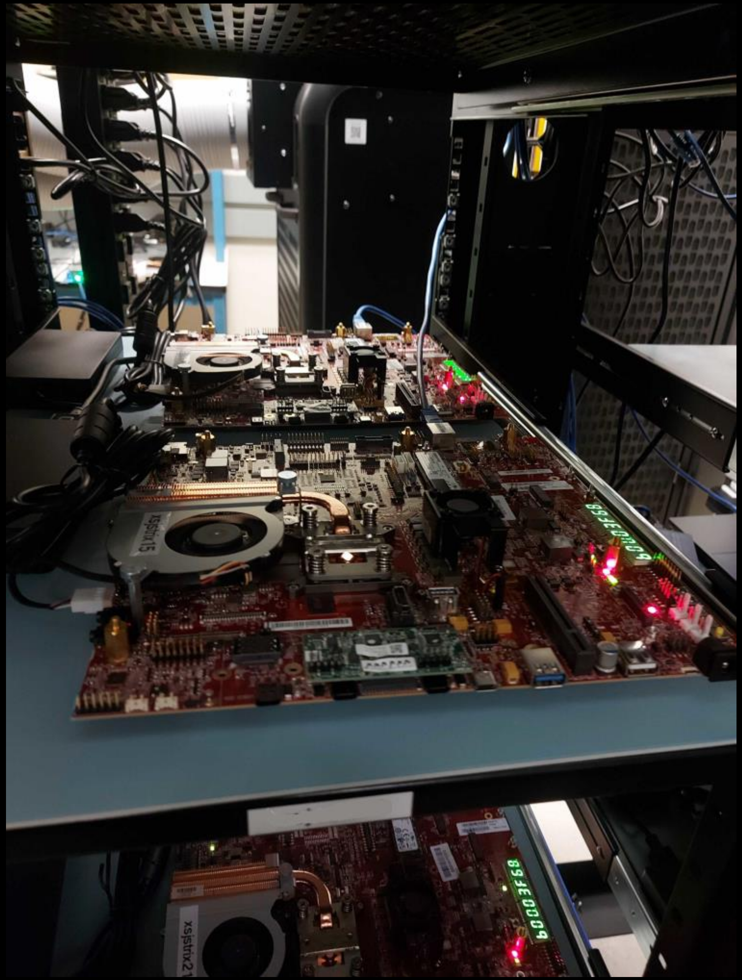


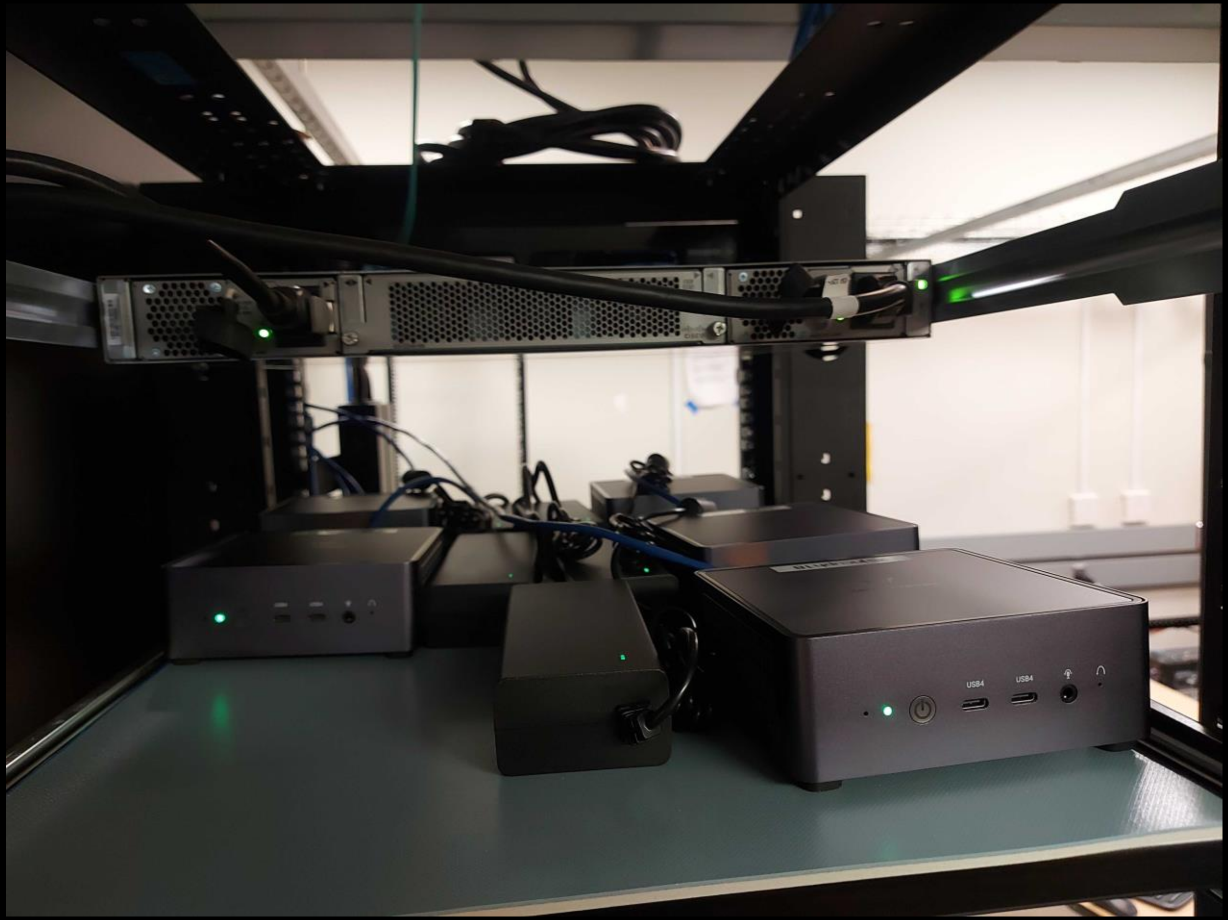
MLIR-based AIEngine toolchain

This repository contains an MLIR-based toolchain for Xilinx Versal AIEngine-based devices. This can be used to gener portion of the device, including processors, stream switches, TileDMA and ShimDMA blocks. Backend code generation This project is primarily intended to support tool builders with convenient low-level access to devices and enable the de programming models from higher level abstractions. As such, although it contains some examples, this project is not in compilation flows or to be particularly easy to use for system design.

https://xilinx.github.io/mlir-aie/

Join our team!





Compilers, Kernel Optimization, Frameworks
Contact for openings in my team:
elliott.delaye@amd.com
https://careers.amd.com/

AMD