



Memory Tradeoffs in Spatial Programming of Embedded Accelerators

Dr. Stephen Neuendorffer

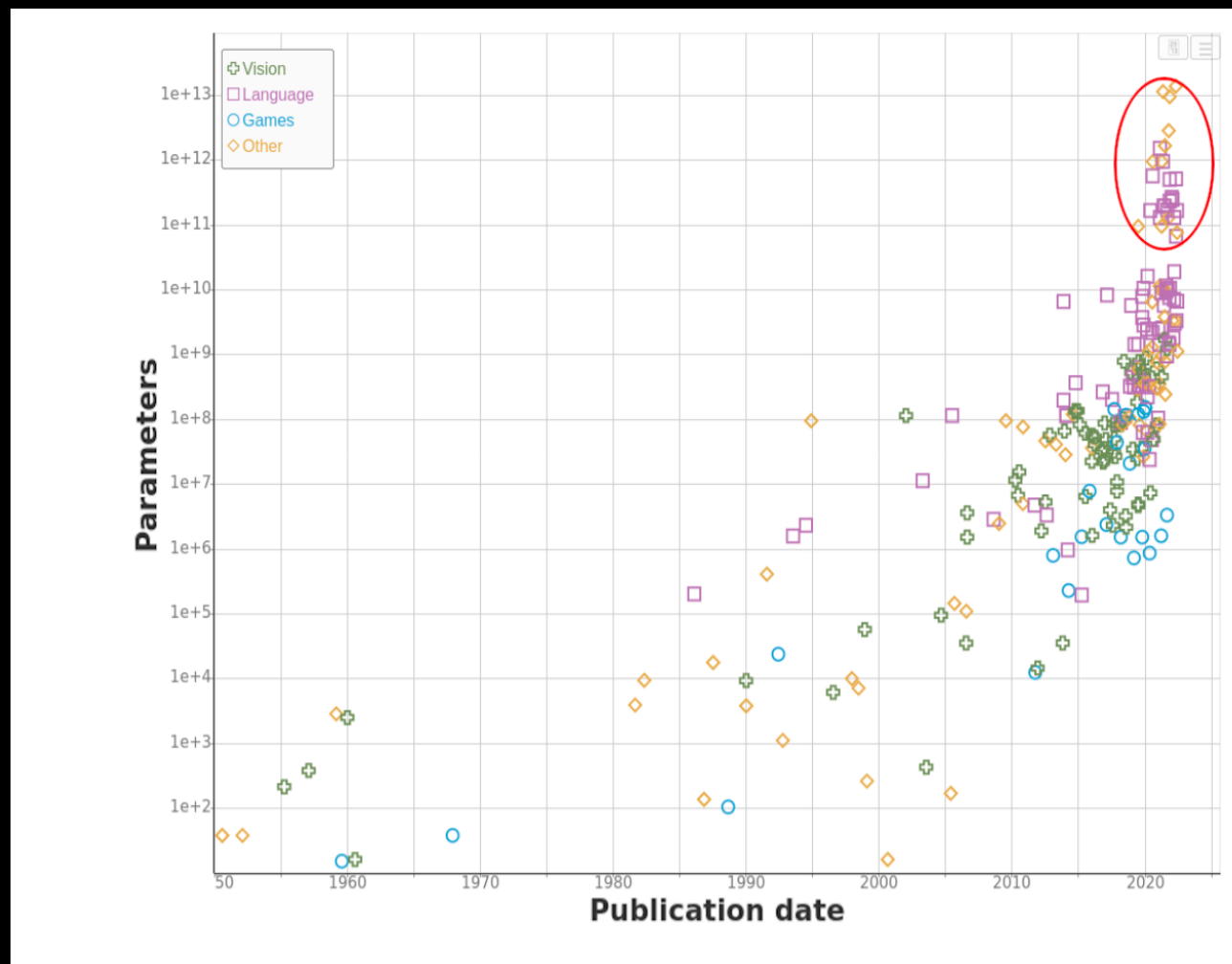
Fellow

Research and Advanced Development

With contributions from many other AMD colleagues

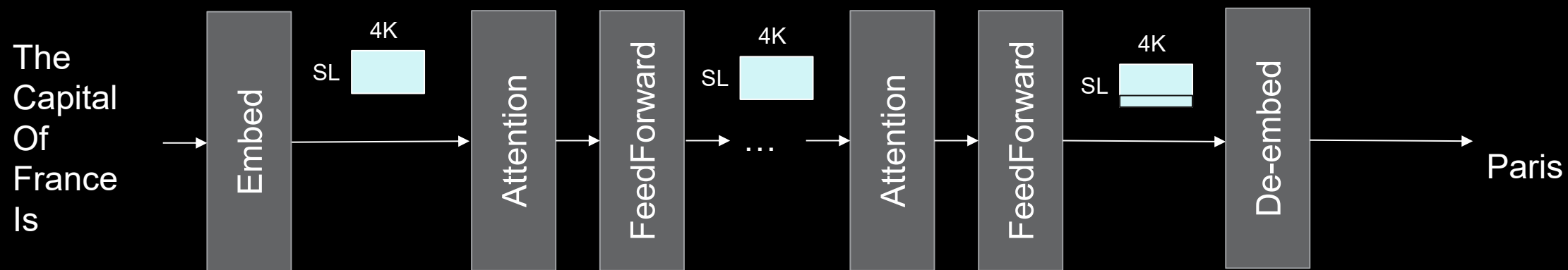
Sep 2026

AI Model Size

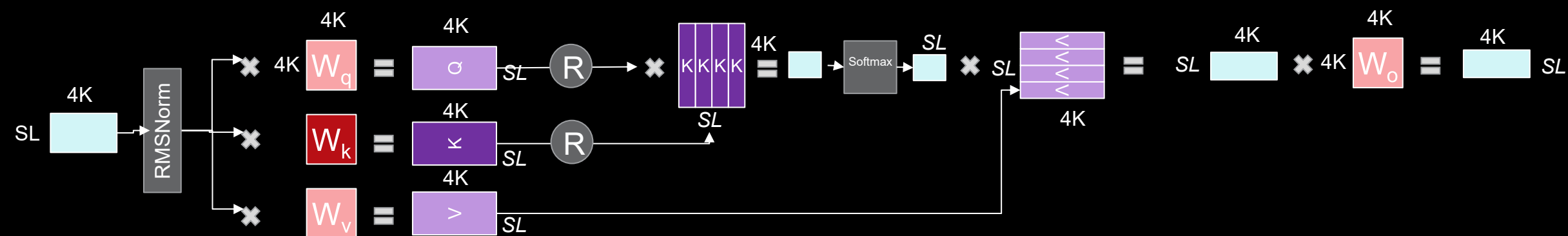


[Ref] Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn. 'Machine Learning Model Sizes and the Parameter Gap'. ArXiv [Cs.LG], 2022. arXiv. <http://arxiv.org/abs/2207.02852>.

A Simple LLM model (~Llama)



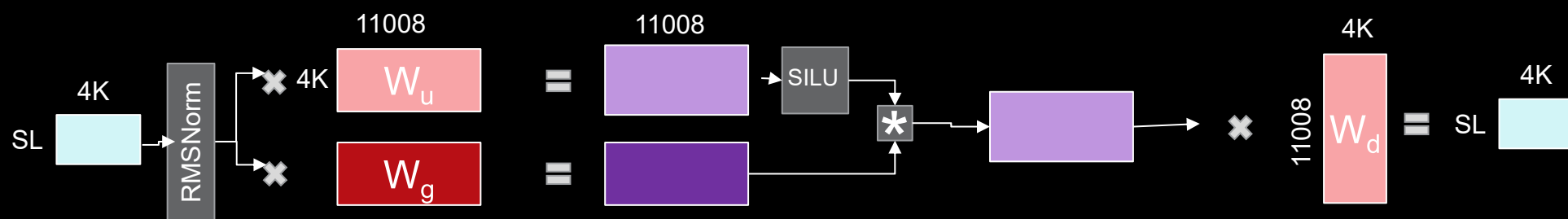
Single-Head Attention, Prefill Phase



Observations:

- 1) GEMM operations predominate during the prompt/encoder phase
- 2) Typically compute bound
- 3) With short sequence lengths, projections dominate compute and weights dominate external I/O
- 4) With long sequence lengths, attention dominates compute

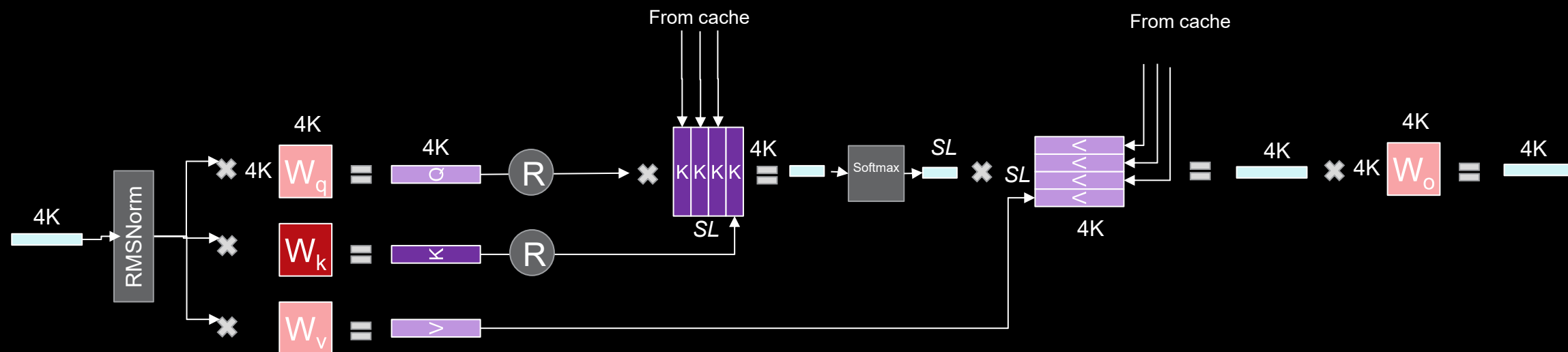
Feed Forward, Prefill Phase



Observations:

- 1) Larger projections than attention
- 2) Typically more compute bound than memory bound

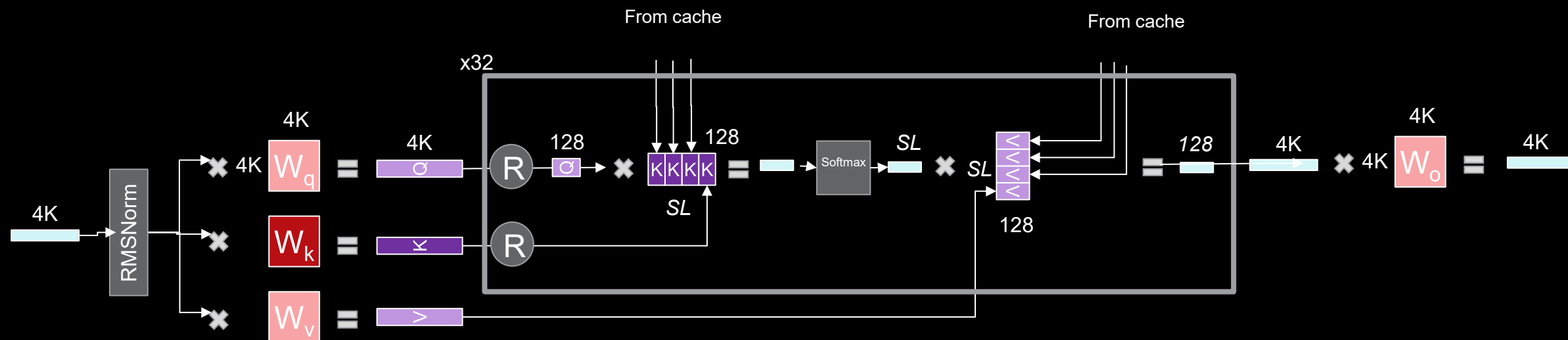
Single-Head Attention, Decode Phase with Cache



Observations:

- 1) GEMV operations predominate during the decode phase
- 2) Typically very I/O bound (Weights and KV Cache)

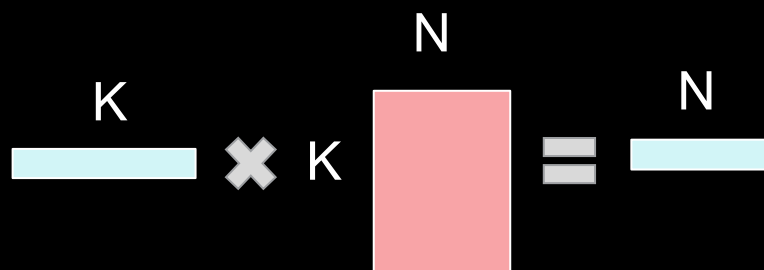
Multi-Head, Token Phase with Cache



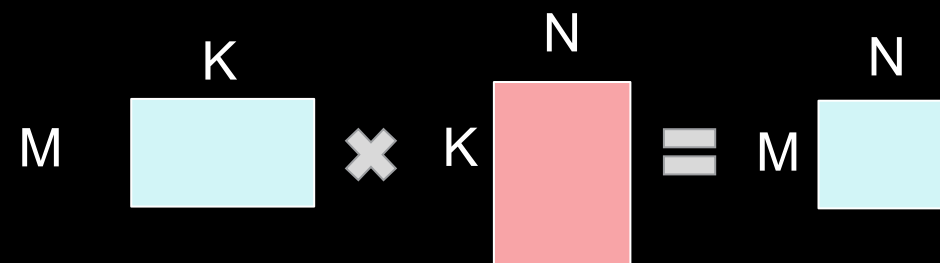
Lots of additional complexity in SOTA models:
 Grouped Query Attention
 Fast Attention
 Mixture of Experts

Lots of additional complexity in serving these models:
 Prefill/Decode disaggregation
 Multi-user Batching
 Large Models Don't Fit in 1 GPU
 Latency Constraints

Matrix Operations



GEMV:
 $O(K*N)$ Compute
 $O(K*N)$ I/O

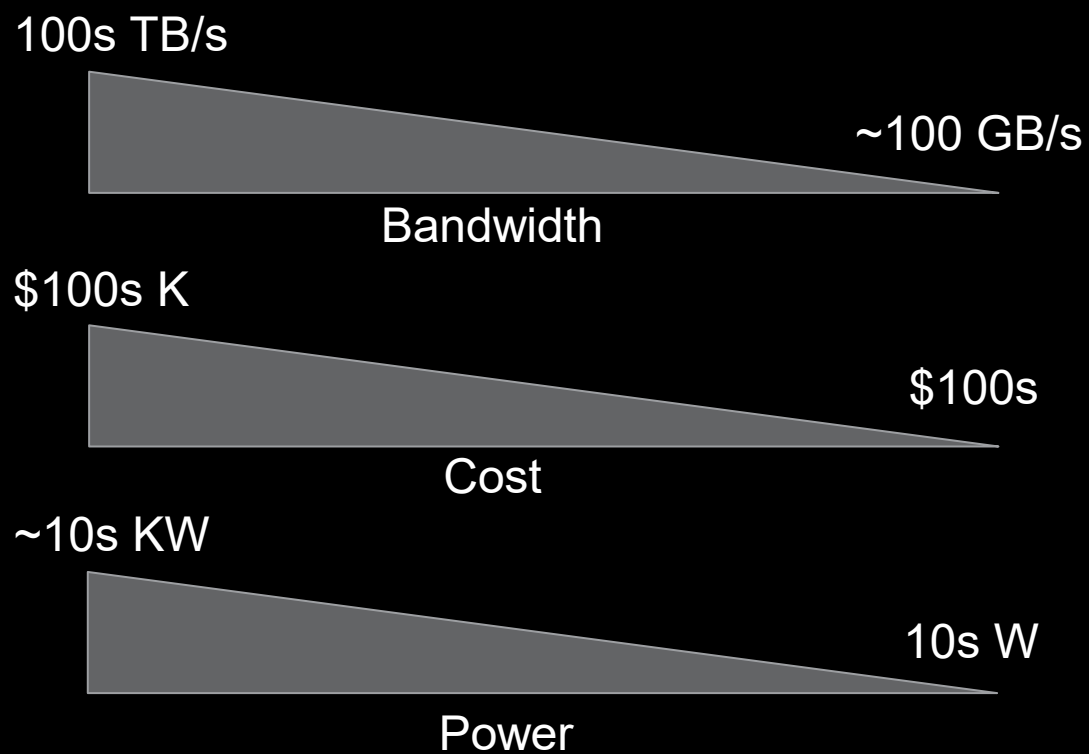


GEMM:
 $O(M*K*N)$ Compute
 $O(M*K) + O(K*N)$ I/O

Memory Considerations: Diverging Requirements



Datacenter

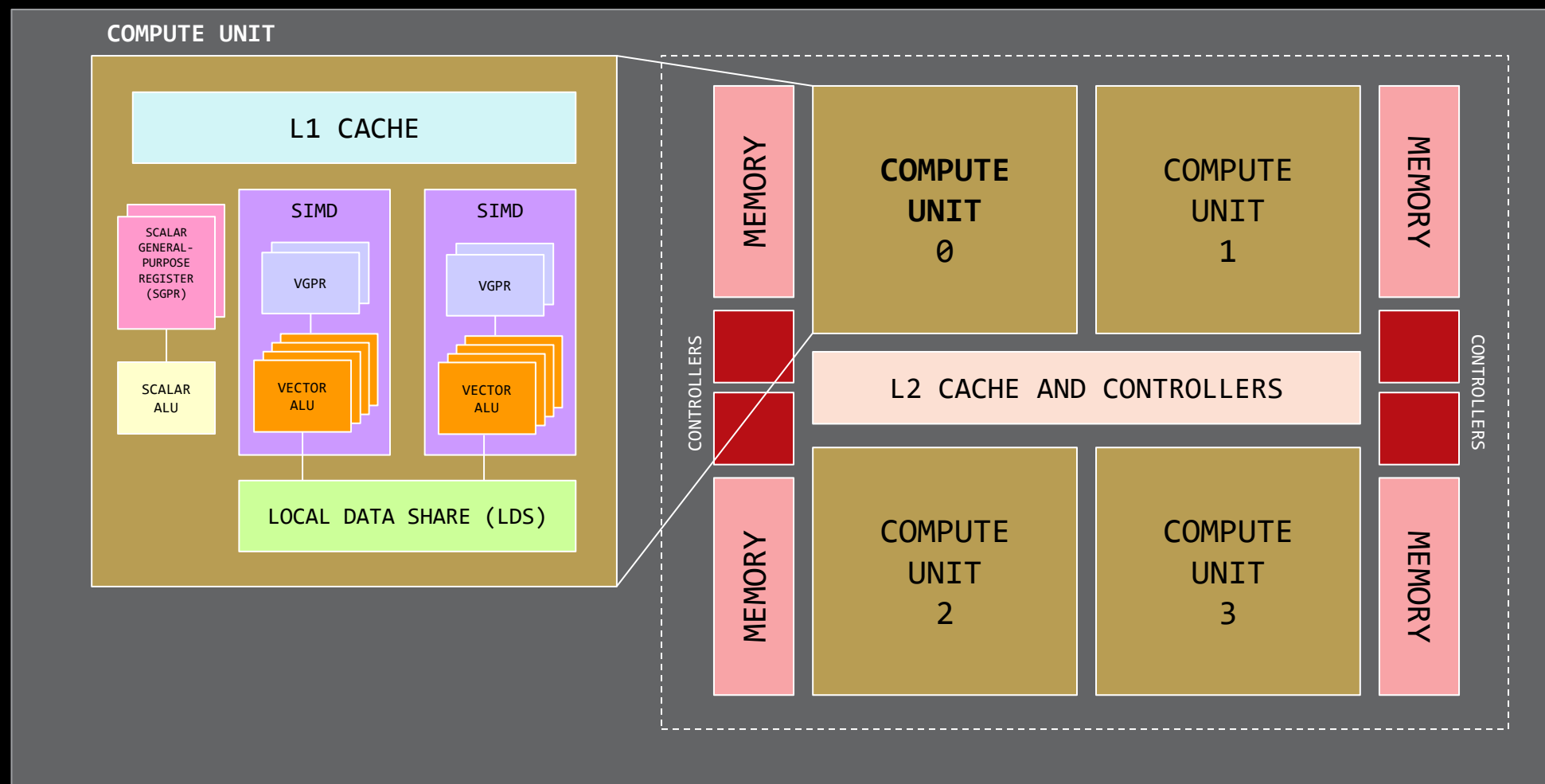


Client (Laptop)

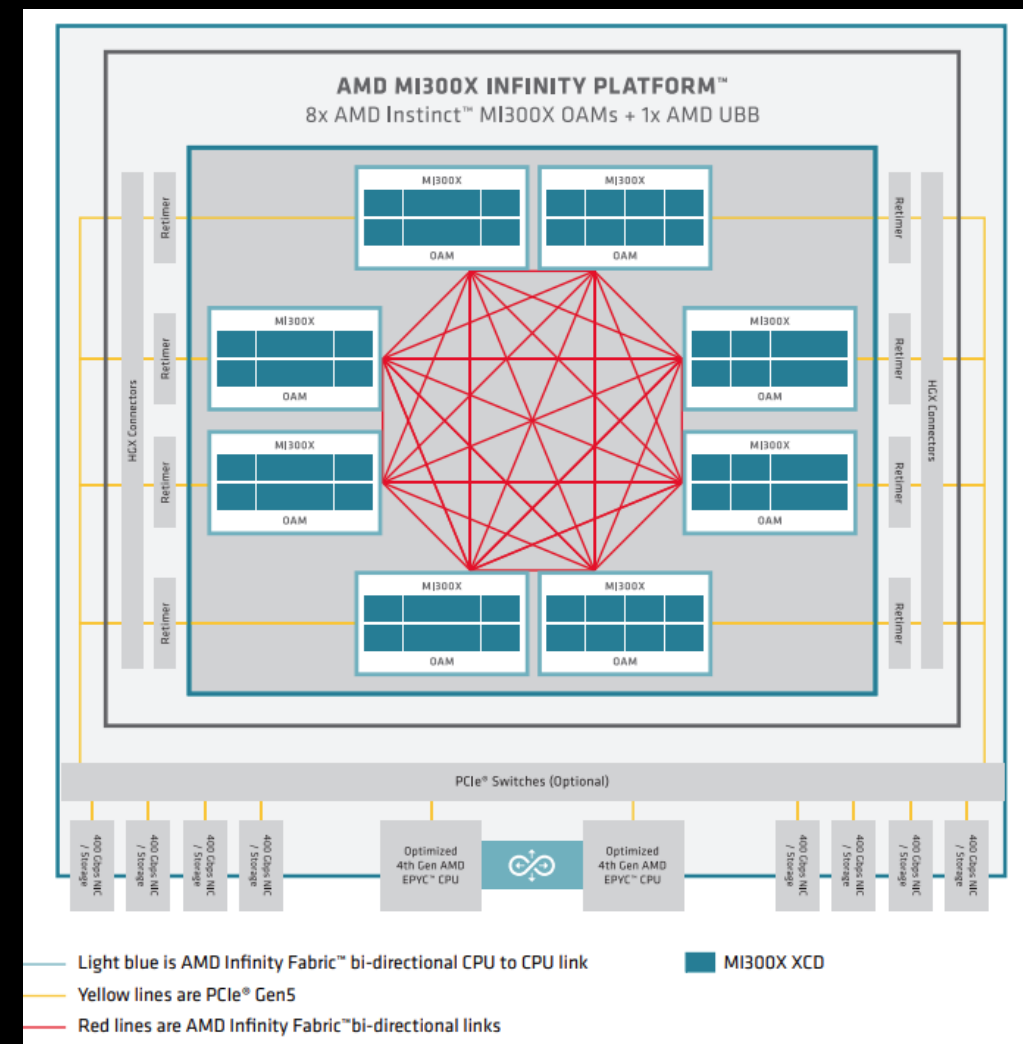
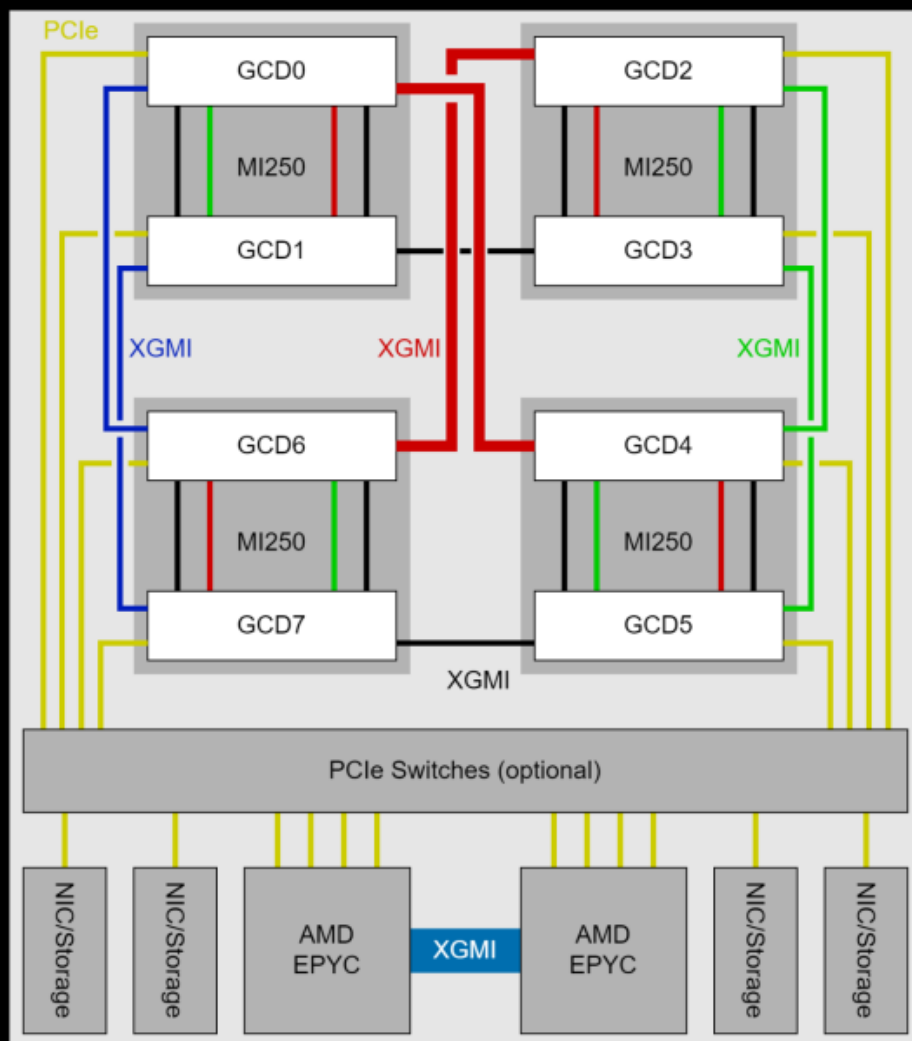
Claim

- Spatial Locality is increasingly important
- Cannot treat memory as an infinite, global resource
- These effects increasingly important within a device

High-level GPU Architecture

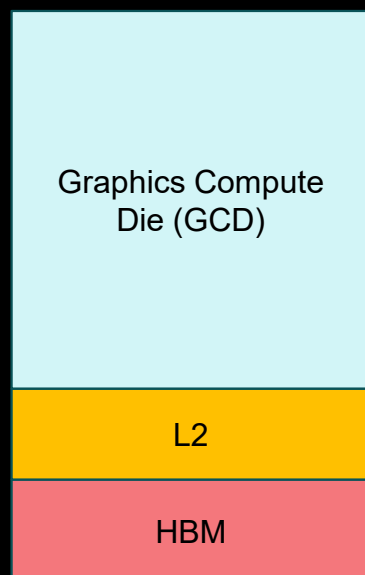


Locality in Distributed Systems is Important!

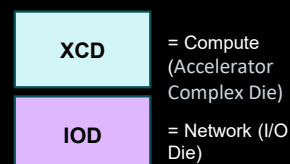
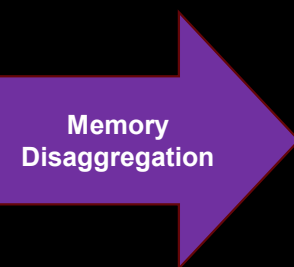


Spatial Locality at Device Scale

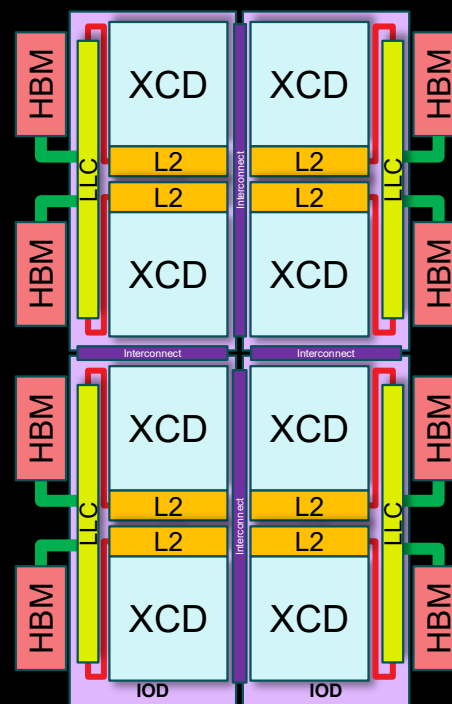
MI200



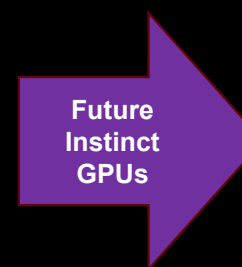
Aggregated Memory
No L2 Cache NUMA
No HBM NUMA



MI300



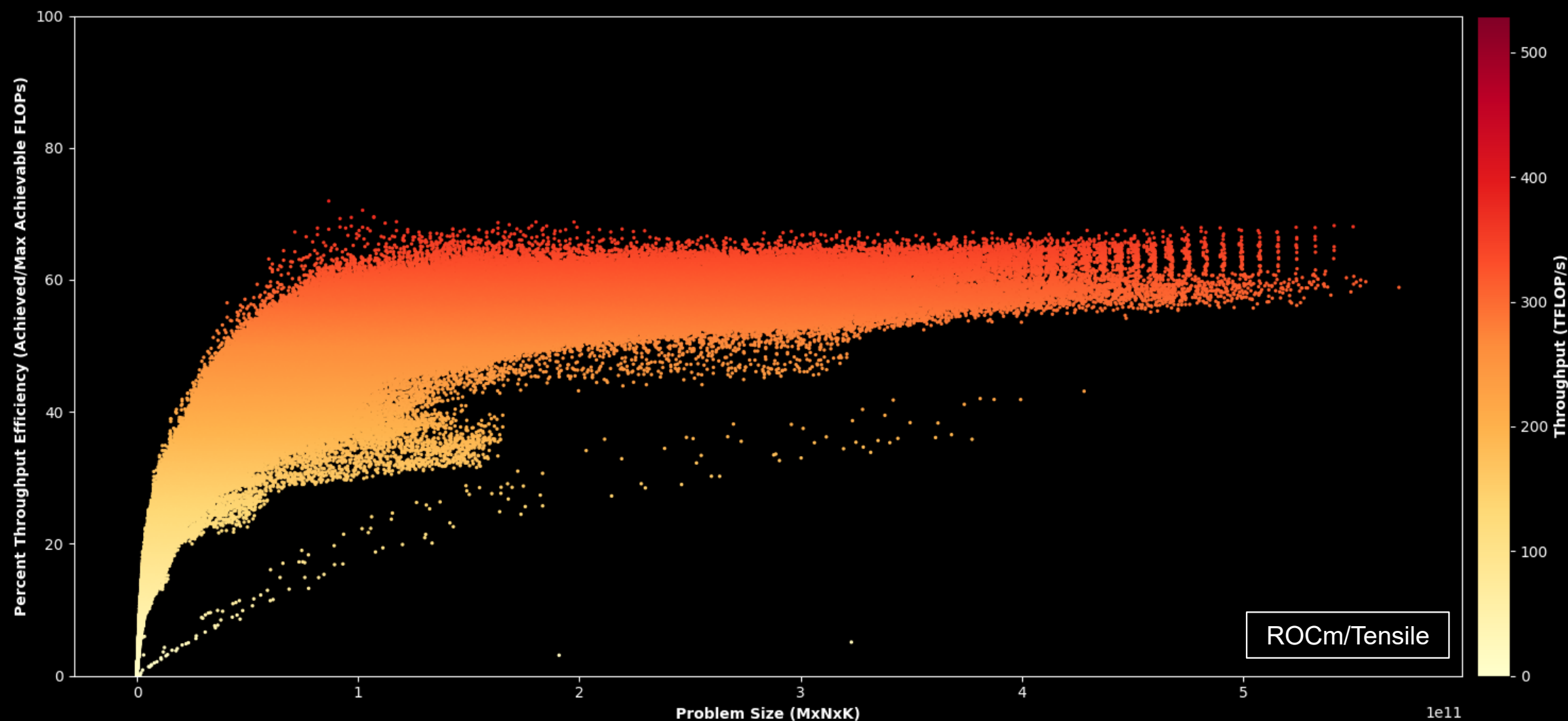
Disaggregated Memory
L2 Cache NUMA
HBM Stack NUMA



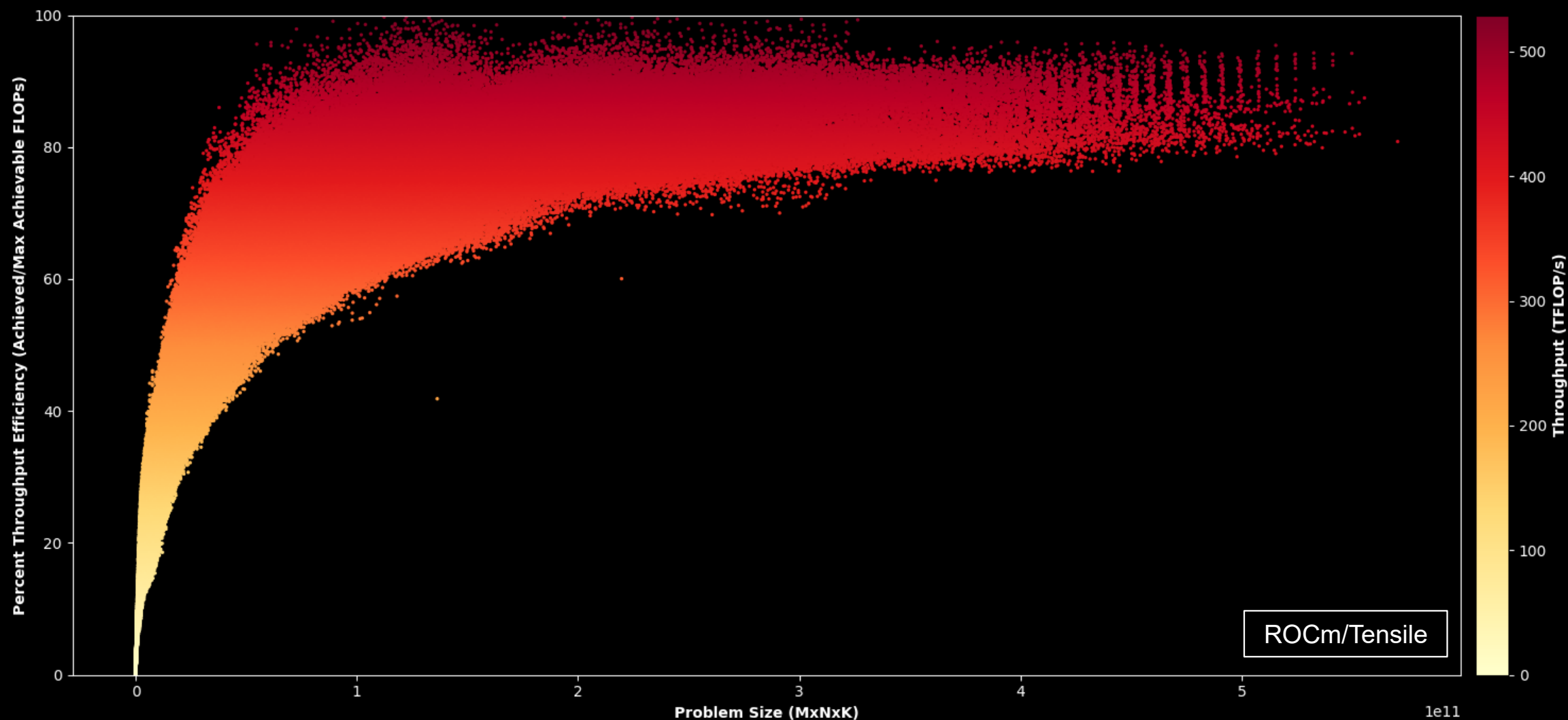
Disaggregated Memory
L2 Cache NUMA
DRAM Stack NUMA
Position within XCD NUMA
...

Architectural NUMA effects are inevitable; our algorithms and programming models must evolve to effectively program them.

GEMM Performance with Random Spatially-Unaware Dispatch

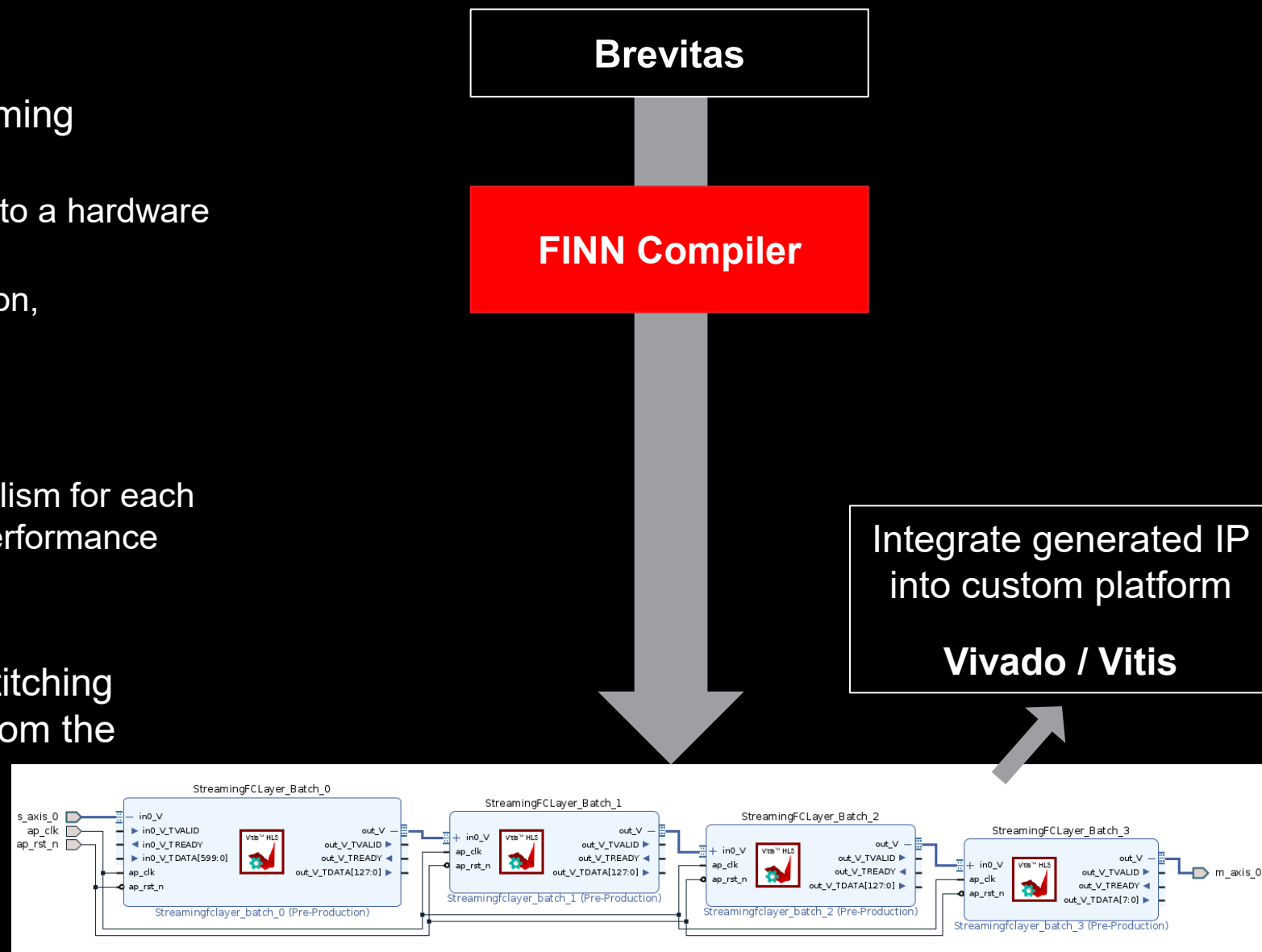


Significantly Better Performance with Spatially Aware Dispatch



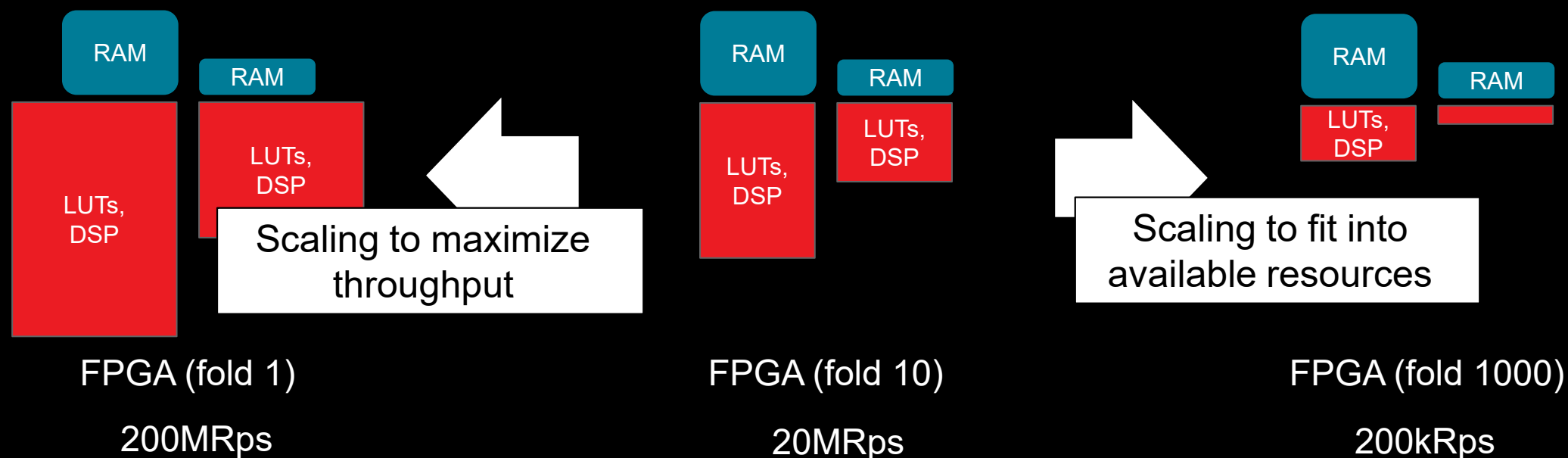
FINN Compiler

- Modular **graph compiler** performing transformations to
 - Incrementally lower ONNX graph to a hardware description
 - Perform **optimization**: Layer fusion, streamlining, ...
- Explores the **design space**
 - Calculates the degrees of parallelism for each kernel using resource cost and performance models
- Creates **DNN hardware IP** by stitching together parametrized kernels from the **kernel library**



Custom Dataflow Processing:

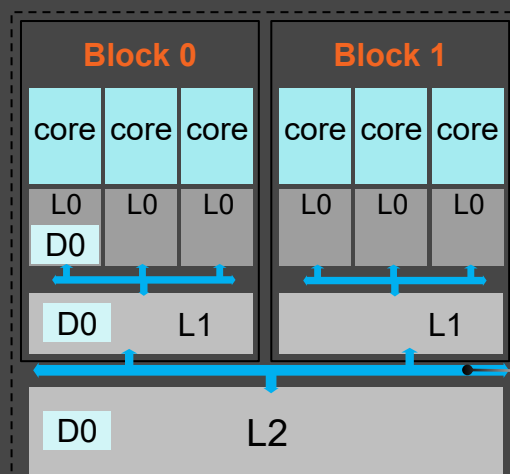
Scaling to Meet Performance and Resource Requirements



1. Scale performance and resources to meet the application requirements
2. If resources allow, we unfold completely, creating a circuit for inference at clock speed

Memory Management

Traditional Multi-core (cache-based architecture)



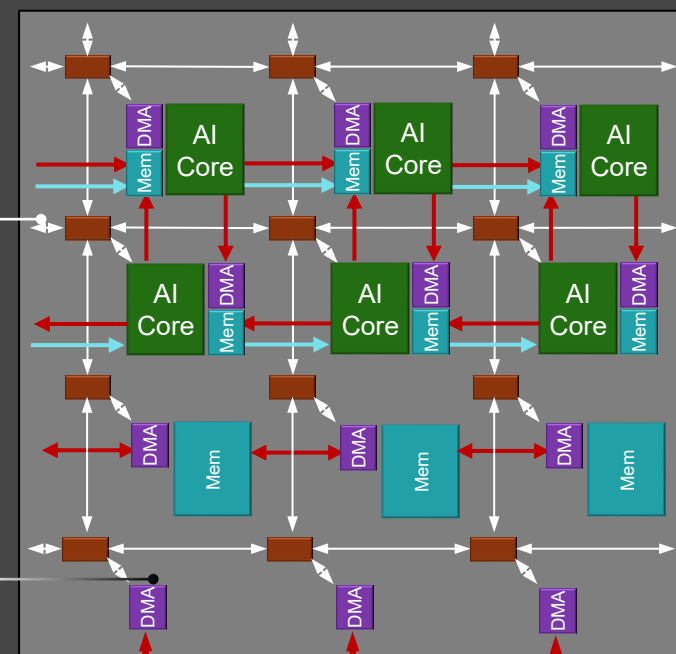
Fixed, shared Interconnect

- Blocking limits compute
- Timing not deterministic

Data Replicated

- Robs bandwidth
- Reduces capacity

AI Engine Array (intelligent engine)



Dedicated Interconnect

- Non-blocking
- Deterministic

Explicit Data Movement

- No cache misses
- Higher bandwidth
- Less capacity required

What is an AMD AI Engine (NPU)?

>1 GHz VLIW / SIMD Vector Processors

- Versatile: Supports ML and other advanced DSP workloads
- 10s to 100s of tiles per device for scalable compute (2D array)

Massive Memory Access Within AI Engine Array

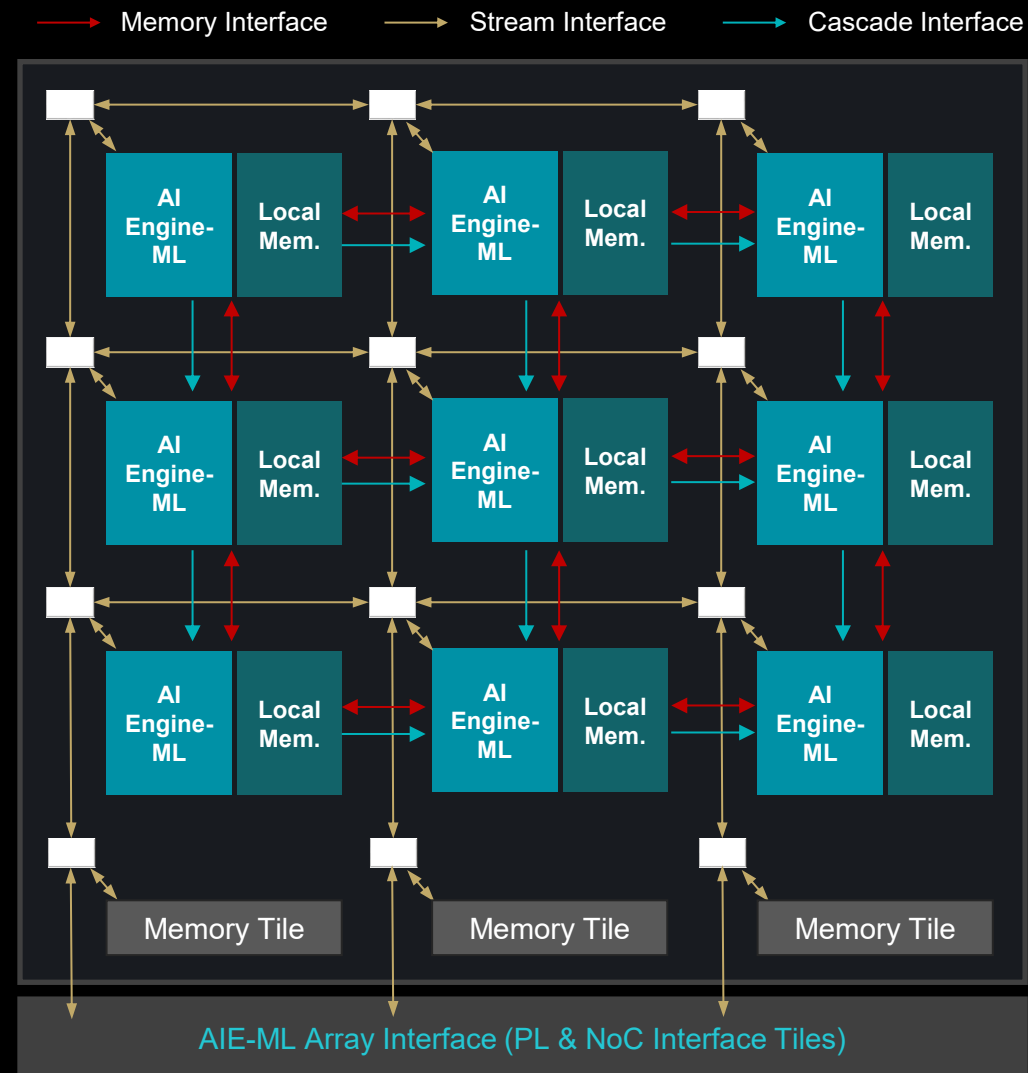
- Reduces data movement across device and need for DDR
- Minimizes power and latency

High-Bandwidth Interface to Programmable Logic or NoC

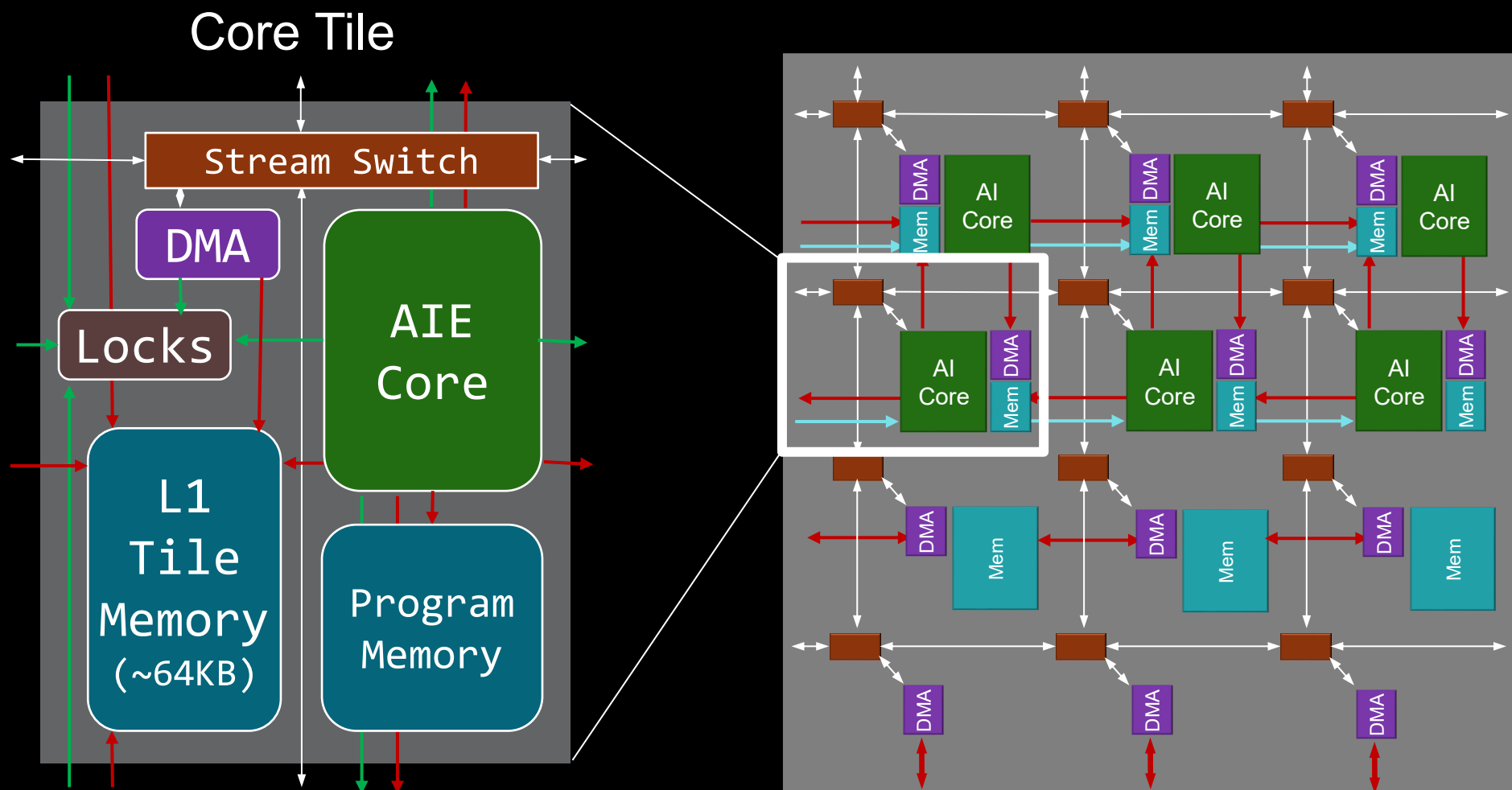
- Flexible connectivity: Stream data from PL or from DDR via NoC
- Optimize data flow for efficient end-to-end application acceleration

SW Programmable for Any Developer

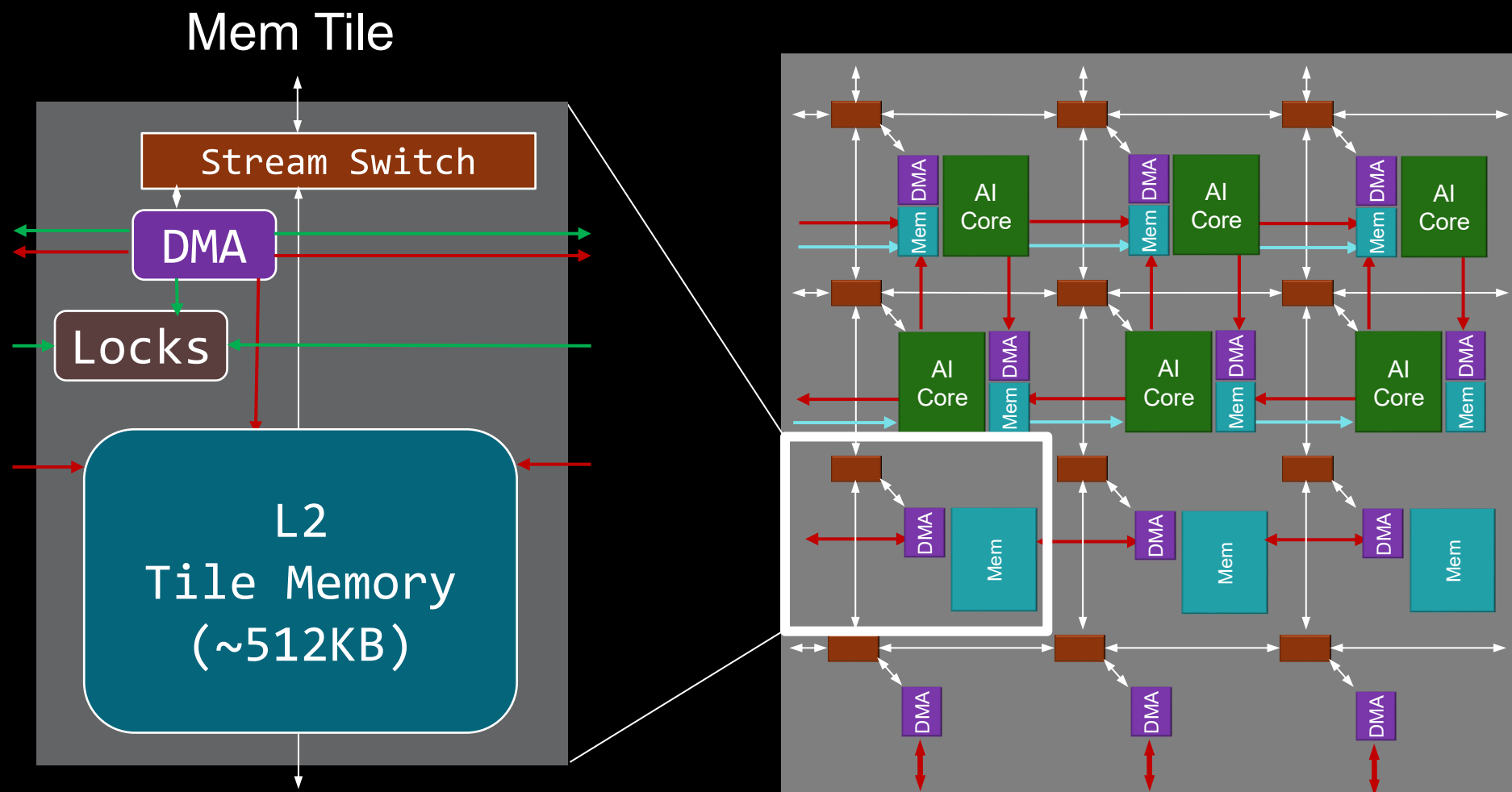
- C programmable, compile in minutes
- Multiple layers of abstraction from custom kernels to out-of-the-box ML framework/model support



AI Engine: Array Architecture

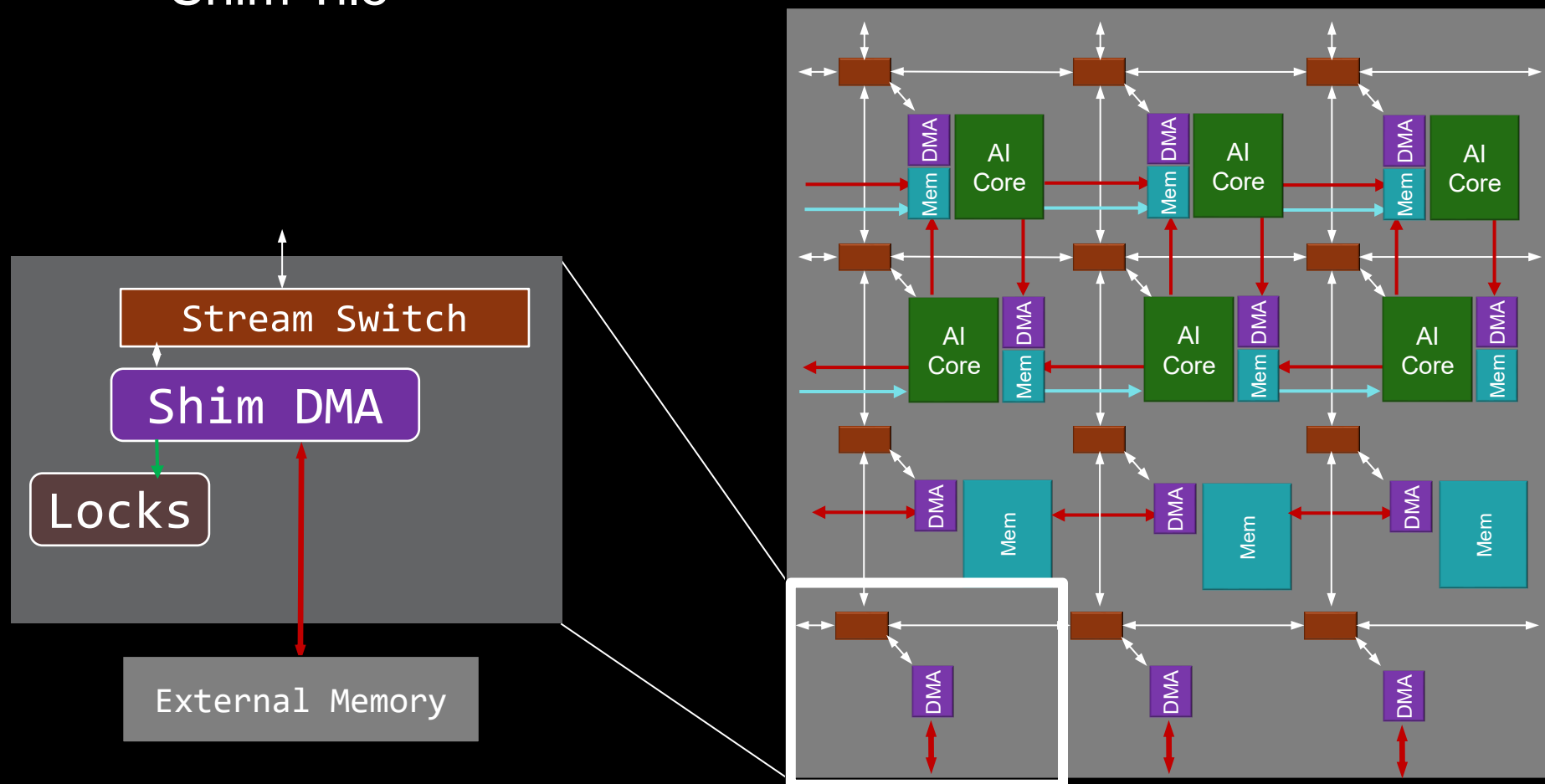


AI Engine: Array Architecture



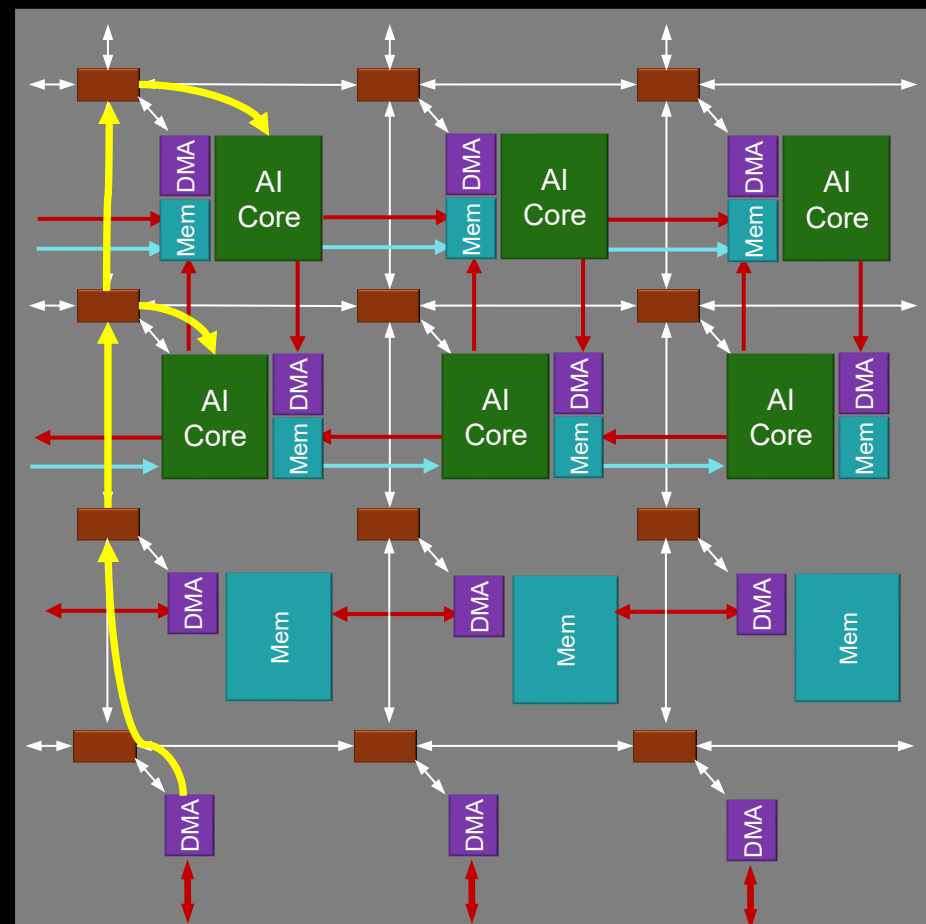
AI Engine: Array Architecture

Shim Tile



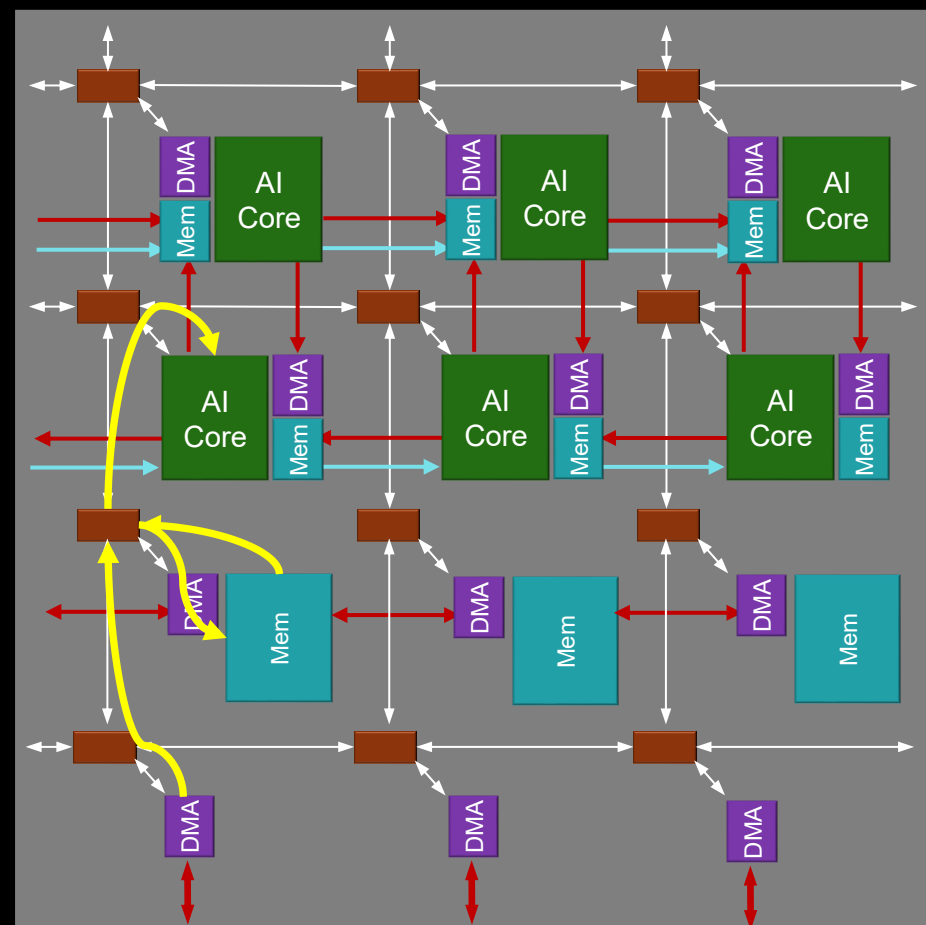
Spatial Reuse

- Read Data from external memory once
- Process using multiple cores



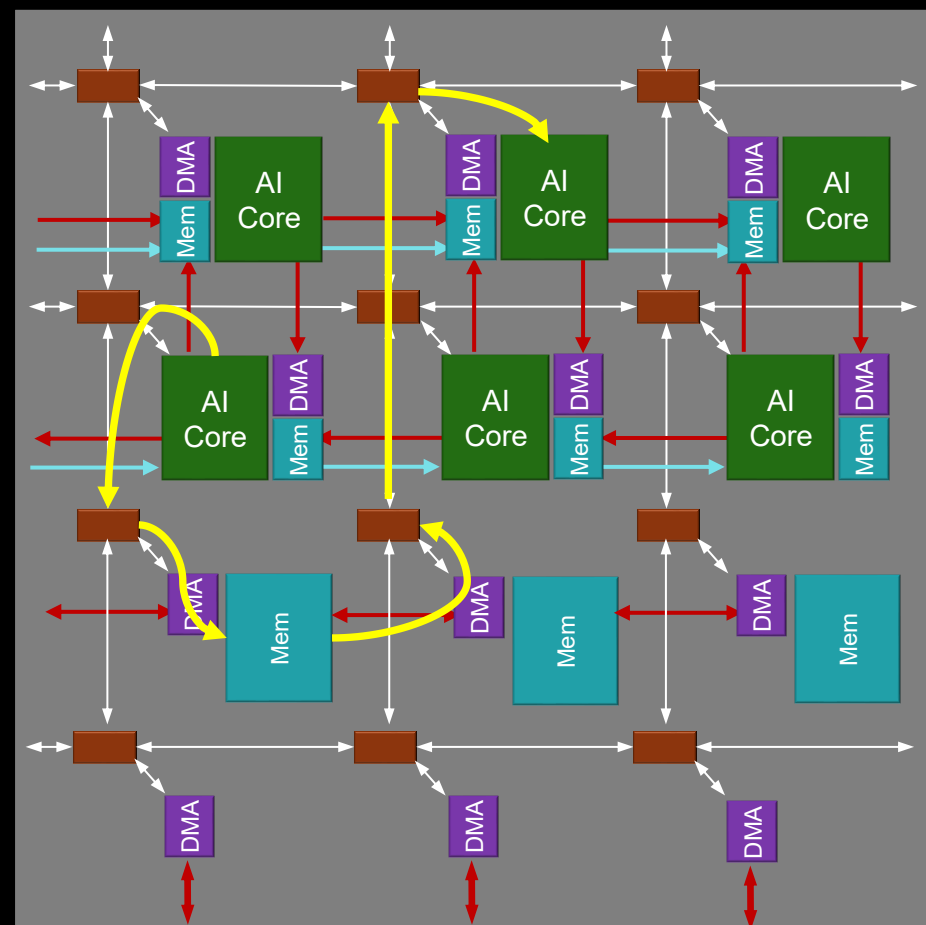
Temporal Reuse

- Read Data from external memory once
- Keep on chip until needed again

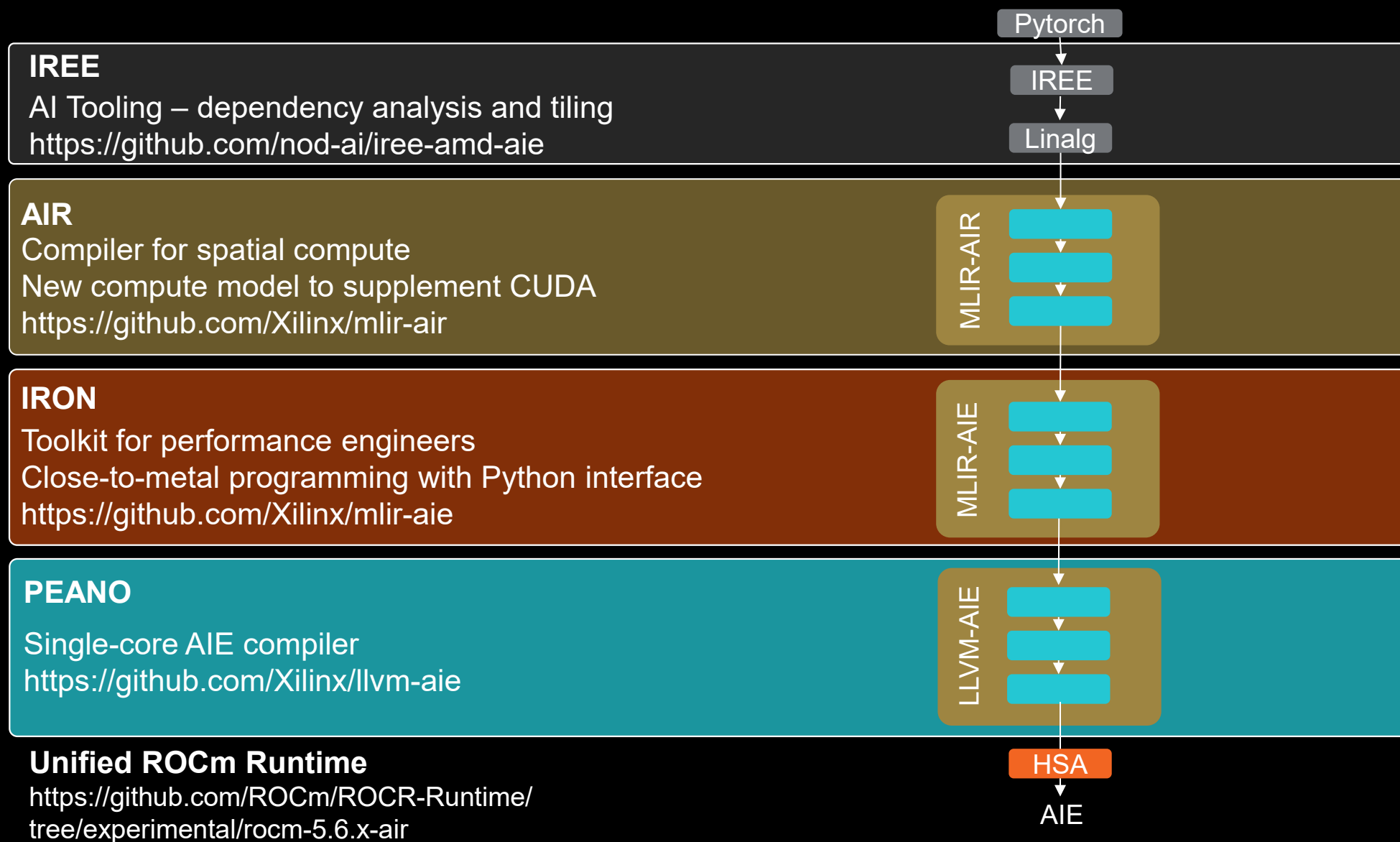


On-chip Buffering

- Keep data on chip
- Keep on chip until needed again



Enabling an Open Source MLIR Compiler Stack



SnortML Model

```

16 # Prepare Data
17 maxlen = 1024
18 X = []
19 Y = []
20
21 for item in data:
22     arr = decode_query(item['str'])[:maxlen]
23     arrlen = len(arr)
24     seq = [0] * maxlen
25     for i in range(arrlen):
26         seq[maxlen - arrlen + i] = arr[i]
27     X.append(seq)
28     Y.append(item['attack'])
29
30 #
31 # Build Model (Simple LSTM)
32 #
33 model = tf.keras.Sequential([
34     layers.Input(shape=(maxlen,)), batch_size=1),
35     layers.Embedding(256, 32),
36     layers.LSTM(16),
37     layers.Dense(1, activation='sigmoid')])
38
39 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
40
41 model.summary()
42
43 # Train Model
44 #
45 model.fit(np.asarray(X).astype(np.float32),
46         np.asarray(Y).astype(np.float32),
47         epochs=100, batch_size=1)
48
49 #
50 # Save Model
51 #
52 model.export('model')
53

```

Snort Blog: Talos launching new machine learning-based exploit detection engine

Model Summary

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(1, 1024, 32)	8,192
lstm (LSTM)	(1, 16)	3,136
dense (Dense)	(1, 1)	17

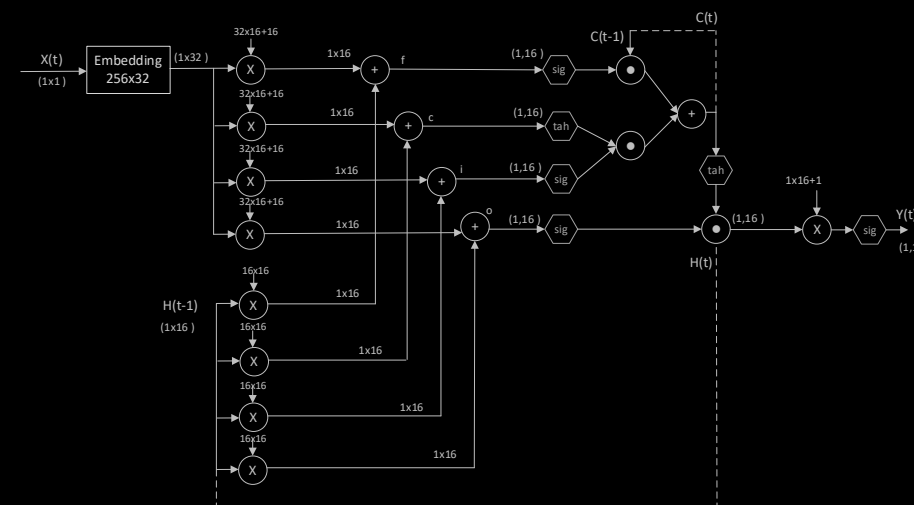
Total params: 11,345 (44.32 KB)

Trainable params: 11,345 (44.32 KB)

Non-trainable params: 0 (0.00 B)

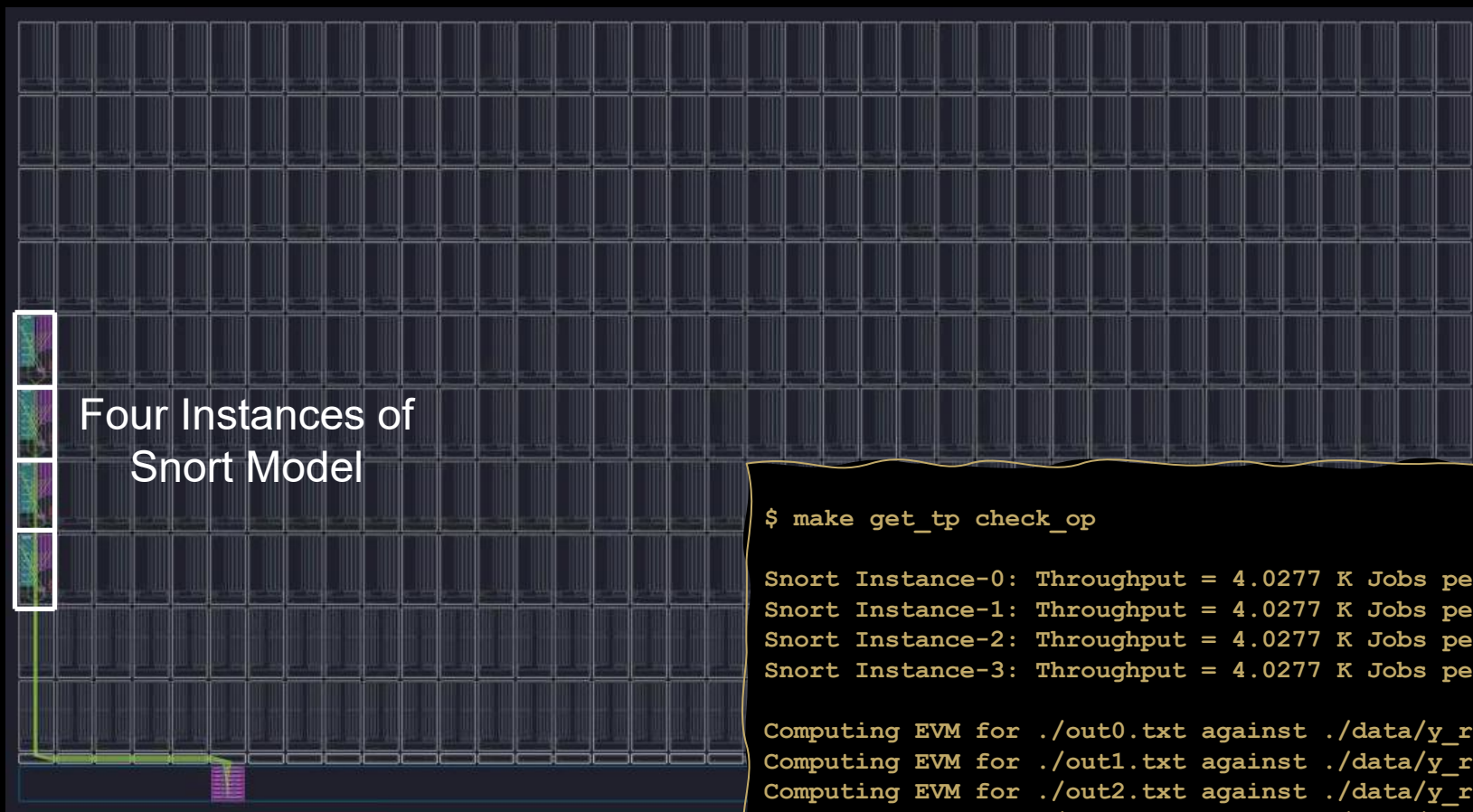
All Weights fit on chip
in 1 AIE core!

Model Compute Data Flow



AIE Implementation Result (4 Instances)

VE2802-2MS @ 1.25GHz



Four Instances of
Snort Model

```
$ make get_tp check_op
```

```
Snort Instance-0: Throughput = 4.0277 K Jobs per second
```

```
Snort Instance-1: Throughput = 4.0277 K Jobs per second
```

```
Snort Instance-2: Throughput = 4.0277 K Jobs per second
```

```
Snort Instance-3: Throughput = 4.0277 K Jobs per second
```

```
Computing EVM for ./out0.txt against ./data/y_ref_float.txt ...MSE = -49.1 dBc
```

```
Computing EVM for ./out1.txt against ./data/y_ref_float.txt ...MSE = -49.1 dBc
```

```
Computing EVM for ./out2.txt against ./data/y_ref_float.txt ...MSE = -49.1 dBc
```

```
Computing EVM for ./out3.txt against ./data/y_ref_float.txt ...MSE = -49.1 dBc
```

Magika v3.3 Model

Unquantized. Use bfloat16 data type.

Idx	Operator	MACs	Weights (KB)
1	Group0	524288	34.9
2	Group1	332922880	1311.7
3	Group2	462592	231.9
Total		333909760	1578.6

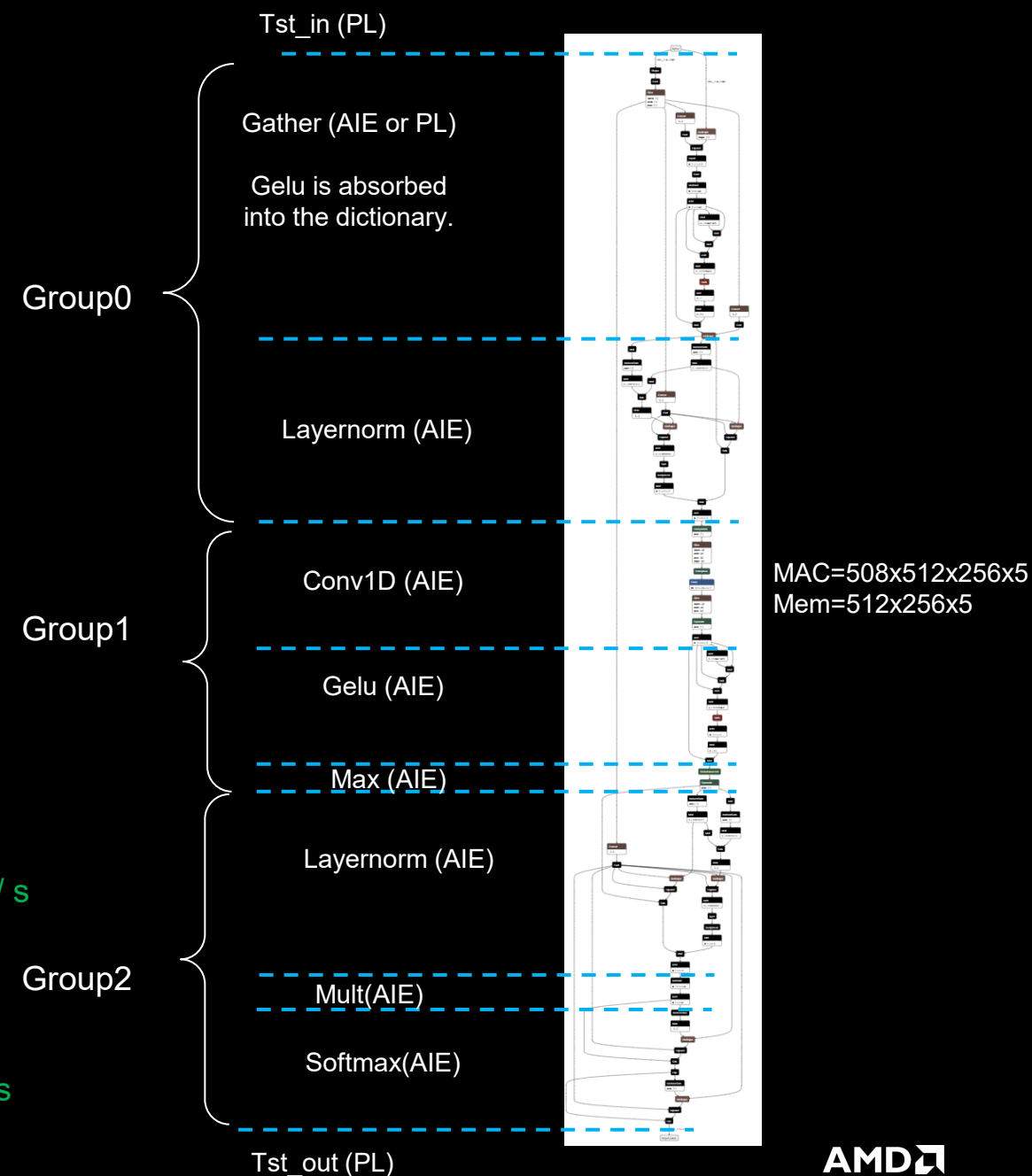
Need ~50 AIEs to fit all the weights in L1

For VE2802 (304 AIEs. Weights in AIE):

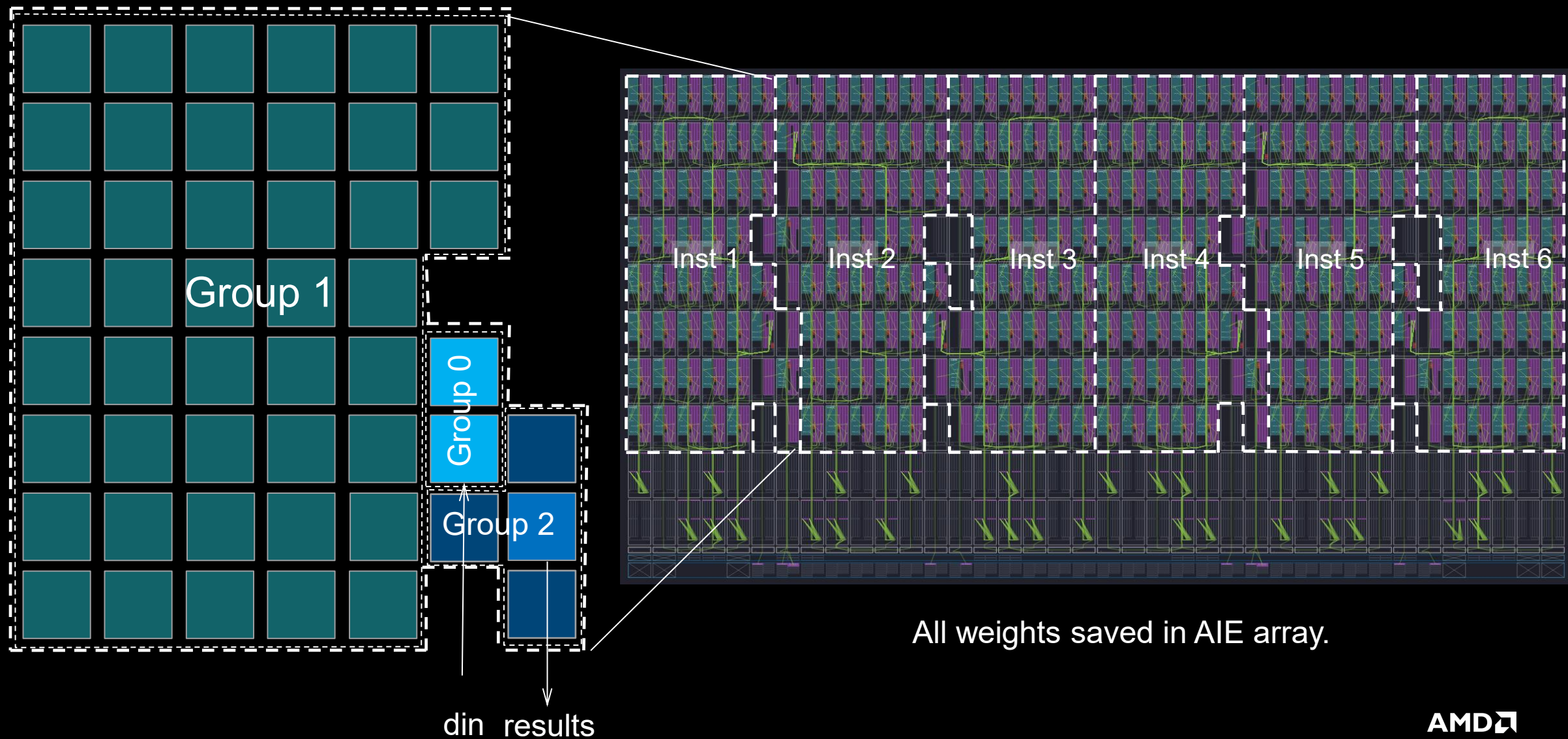
304 AIEs x 128 MACs / cycle x 1.25 GHz = 48.64 T MACs
 48.64 T MACs x 60% overall efficiency / 334 M MACs = 87.4K Inferences / s

For VE2302 (23 AIEs. Weights in PL):

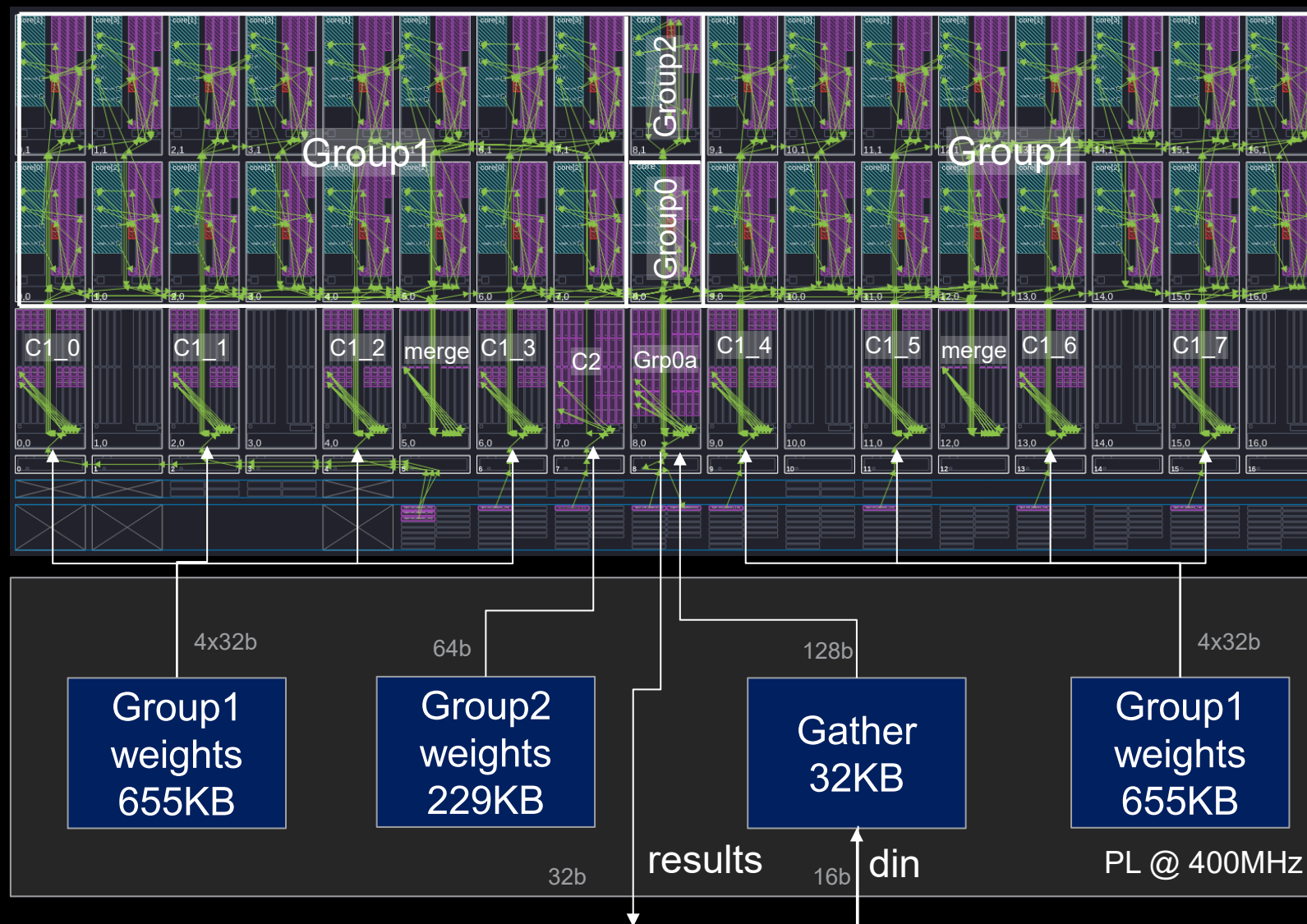
34 AIEs x 128 MACs / cycle x 1.25 GHz = 5.44 T MACs
 5.44 T MACs x 66% overall efficiency / 334 M MACs = 10.7K Inferences / s



AIE Floorplan for VE2802

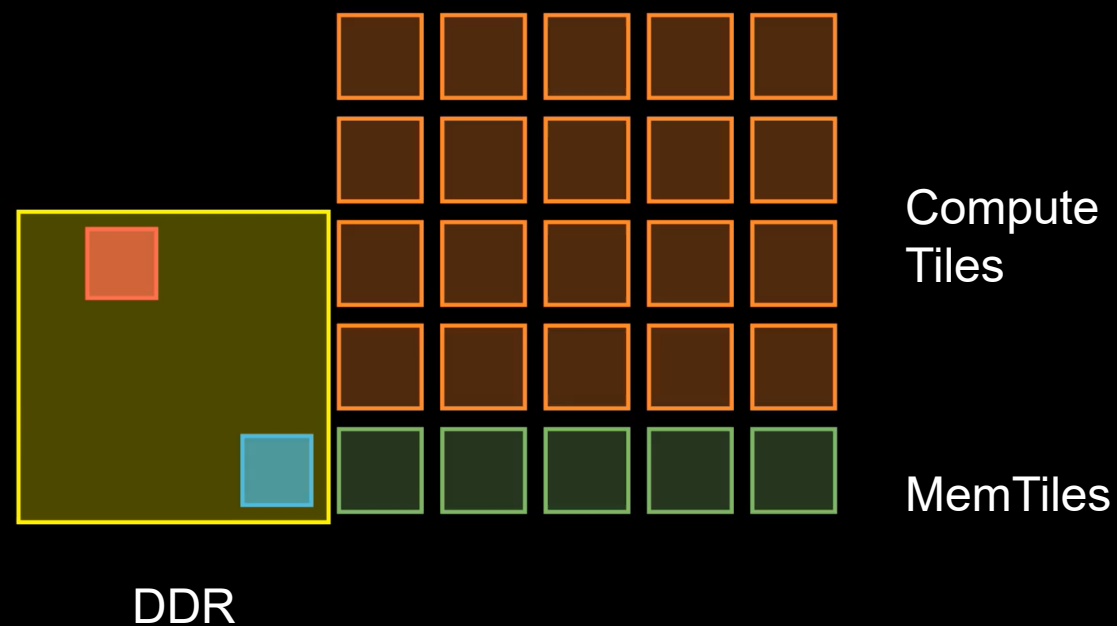


AIE Implementation on VE2302 (1 instance)



Moving to Larger Models

- Larger Models are often more constrained with respect to external memory bandwidth
- Leverage Temporal and Spatial Reuse to build optimized compute overlays for common functions (e.g. GEMM or Fused Attention)
- Higher-level tools can automatically map models into these common functions



AI Engine Array Architecture: Understanding NPU compute blocks

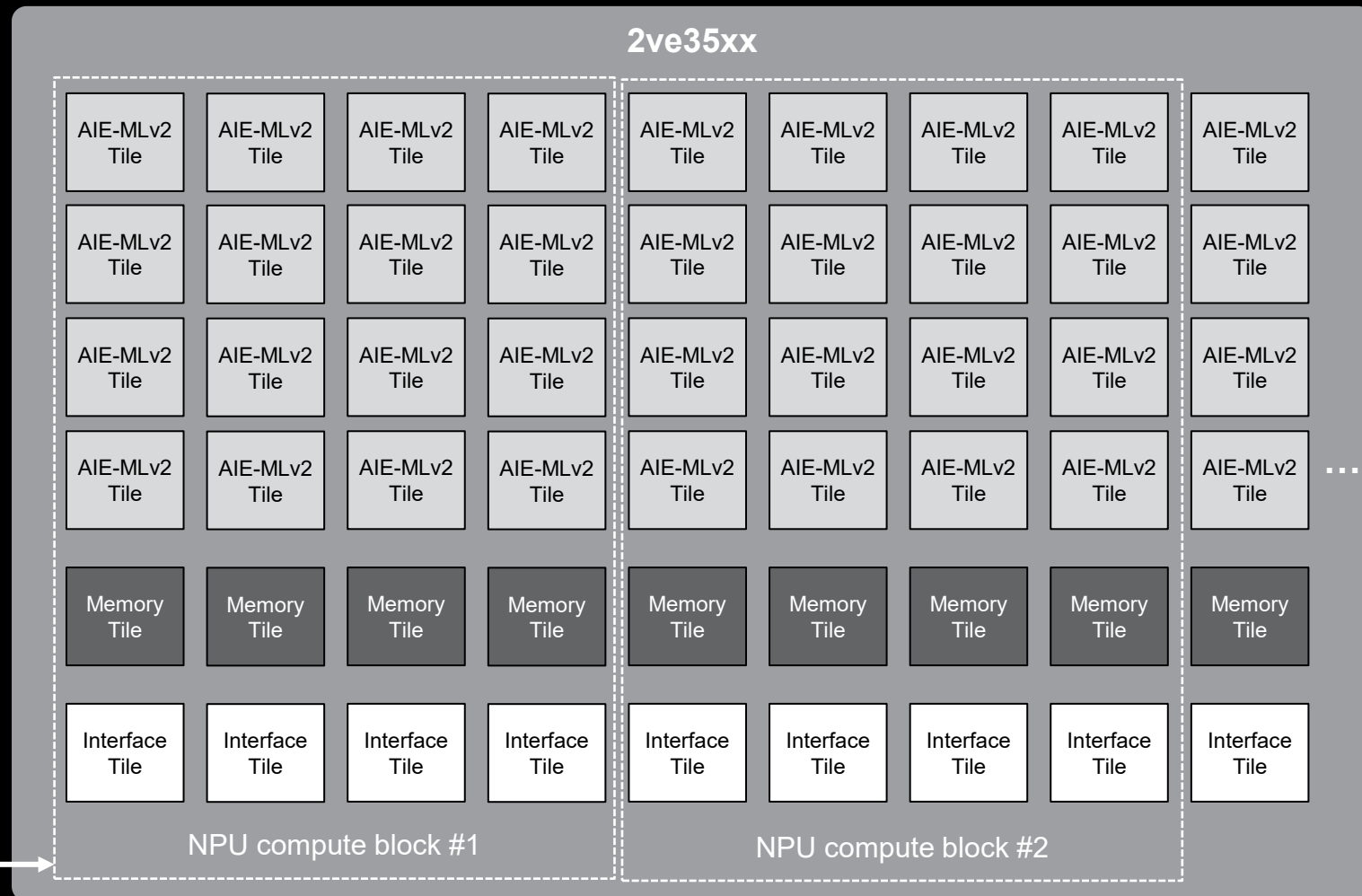
NPU compute block

Lowest granularity compute block for VAIML compiler

- Compiler replicates and combines the **NPU compute block(s)** to build variable size NPU
- One **NPU compute block** contains a mix of tiles:

Tiles (rows x columns)	2ve38xx Up to 36 cols	2ve35xx Up to 24 cols	2ve33xx Up to 12 cols
AIE-MLv2	4x4	4x4	2x4
Memory	1x4	1x4	1x4
Interface	1x4*	1x4*	1x4*

* Shared across NPU compute block when batching



Versal™ AI Edge Series Gen 2 Architecture Overview

Up to 10X Scalar Compute¹

- 8x A78AE cores – Up to 200k DMIPS
- 10x R52 cores – Up to 23k DMIPS

Improved Safety & Security

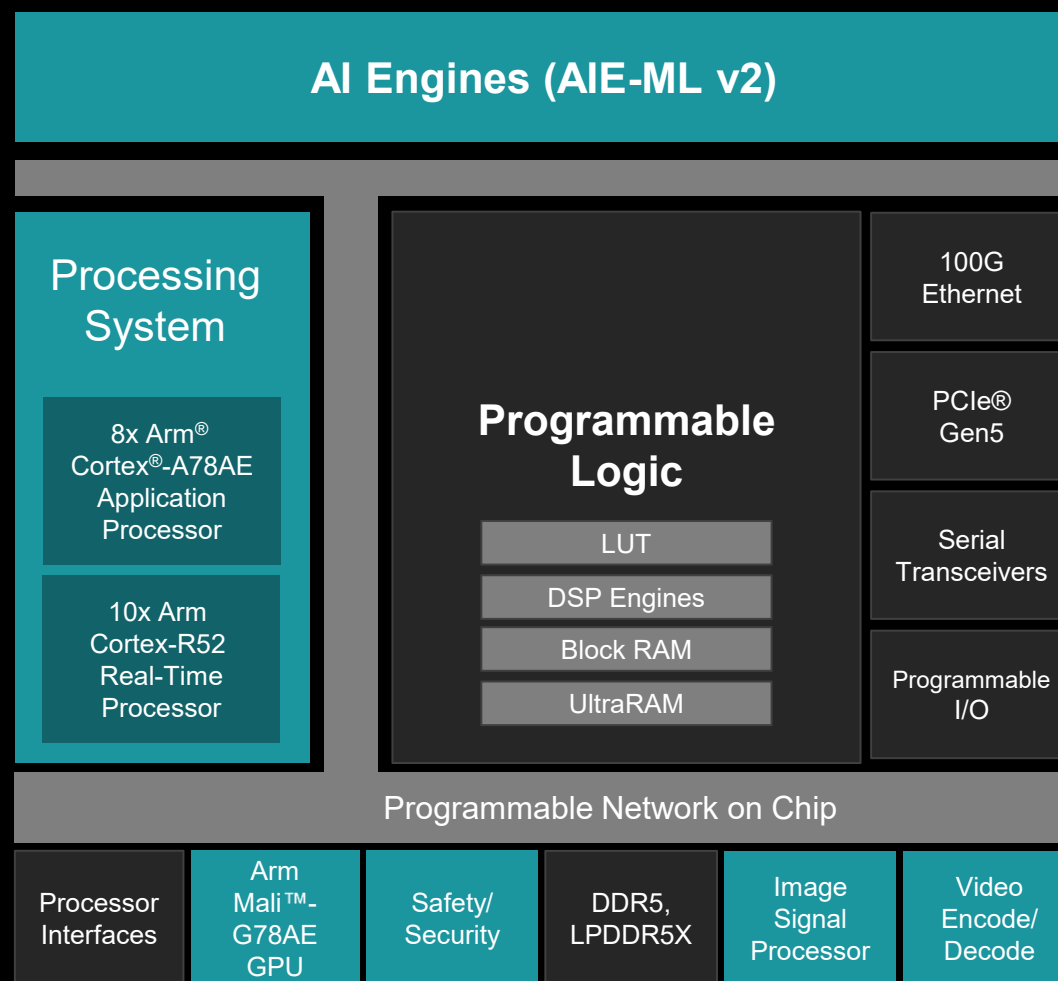
- ASIL D on APU and RPU w/Lock-Step
- Application Security Unit

High-Speed PS Connectivity

- PCIe® Gen5 w/ DMA, 10 Gb Ethernet
- USB 3.2, DP 1.4, UFS 3.1

High-Performance Memory

- 6400-DDR5, 8533-LPDDR5X
- Selectable inline encryption/decryption



Next-Generation AI Engines

- Up to 3X TOPS/Watt vs. AIE-ML¹
- Advanced Data Types (MX6, MX9)

Hard Video Processing

- Image Signal Processors (ISPs)
- Video Codec Units (VCUs)

Integrated GPU

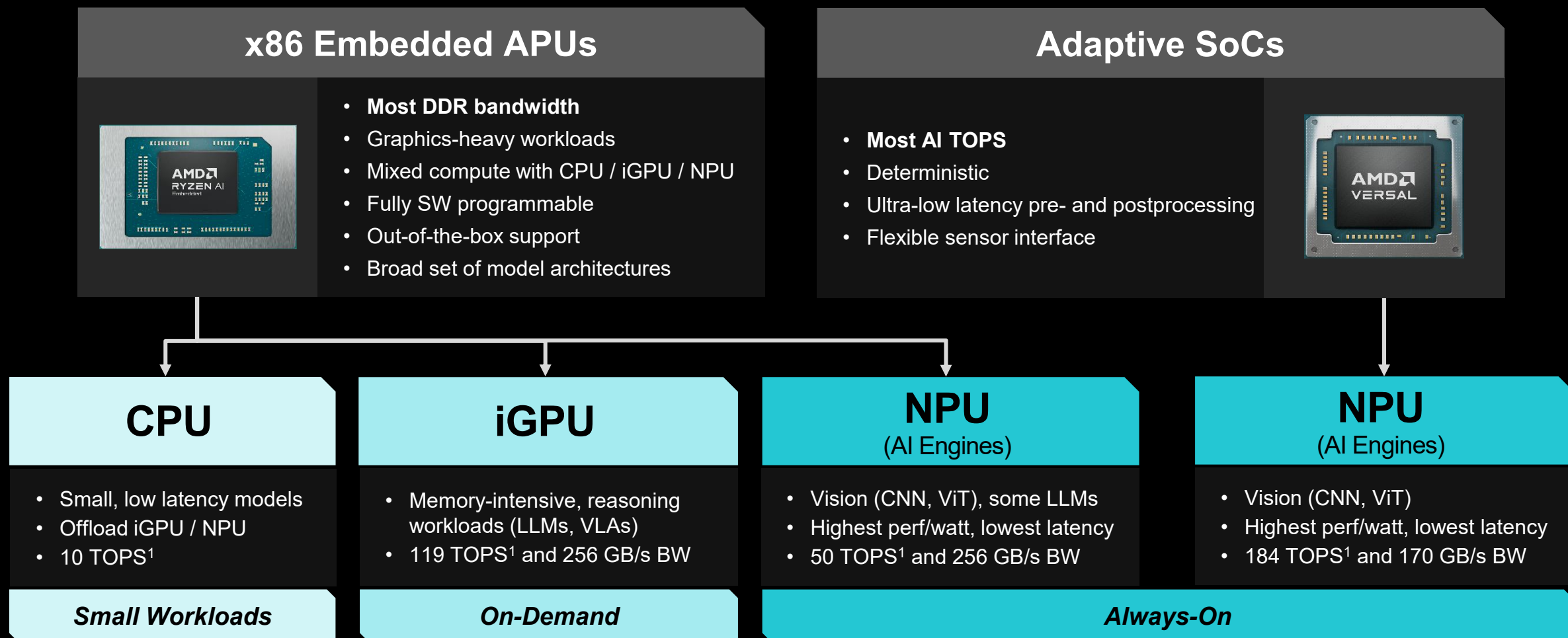
- Arm® Mali™-G78AE for rendering/HMI
- 2D/3D GUI rendering at 4k60

High-Performance I/O

- 1.2 LVDS up to 1.6 Gb/s
- 10 Gb/s MIPI C-PHY, 4.5 Gb/s D-PHY

1: Pre-silicon estimated performance vs. prior generation. See Endnotes VER-023, VER-027.

Compute Engines on AMD Embedded AI Platforms



AMD Ryzen™ Embedded 8000 Series Processor

Next-Generation AMD “Zen 4” x86 CPU Cores

- Up to 8 “Zen 4” Cores
- AVX512 support

Next-Generation AMD RDNA™ 3 iGPU

- Full-featured AMD Radeon™ RDNA 3, up to 6 workgroups
- 4 Display Pipes; 4 DSC

High Performance NPU

- XDNA™ Neural Processing Unit (NPU)

High Bandwidth Infinity Fabric & Memory System

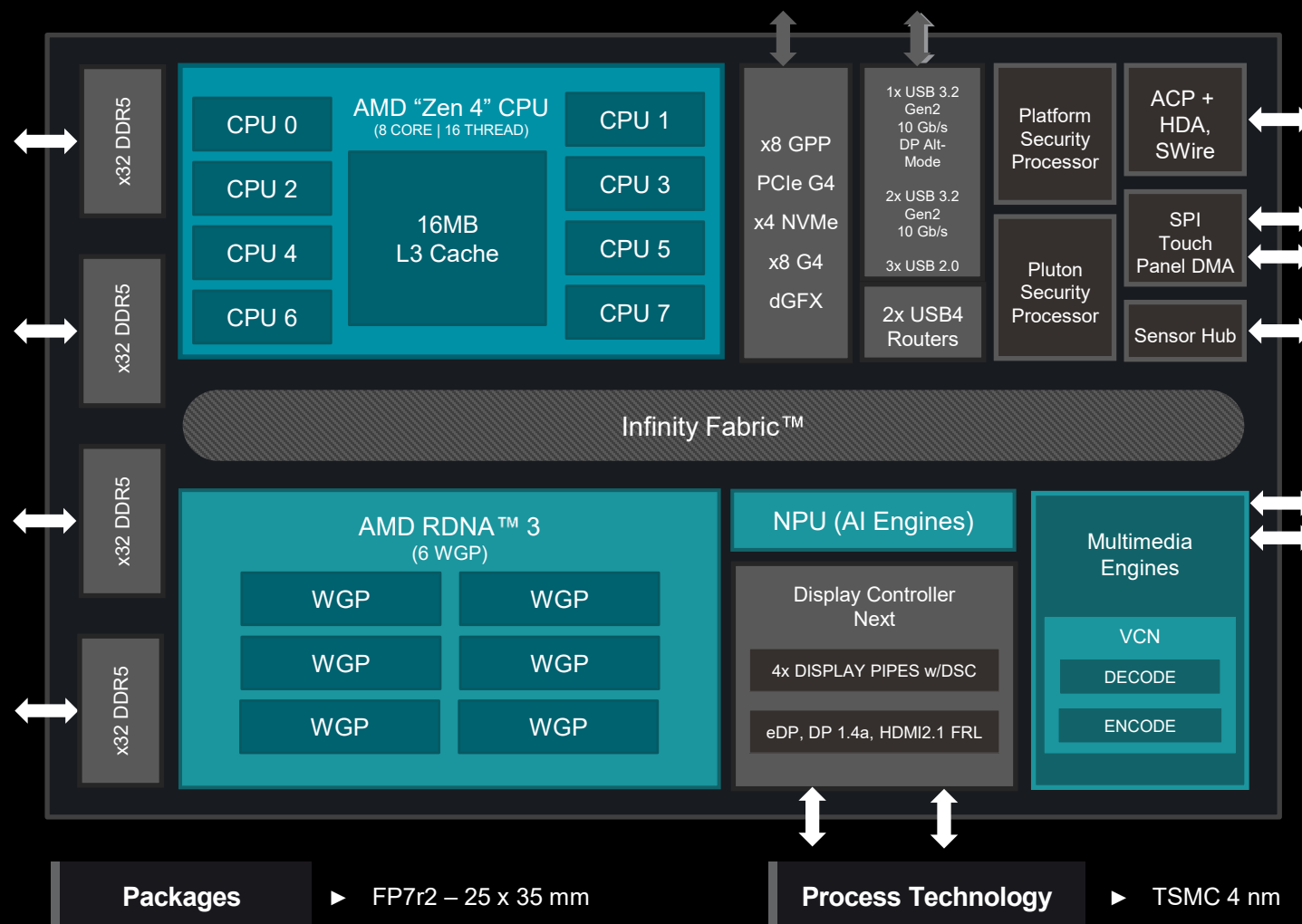
- 4x 32b Channels DDR5

Accelerated Multimedia Experience

- Video codec engine (VCN) with AV1 encode/decode
- Enhanced Audio Engine (ACP) WoV, LPAP

Enhanced I/O

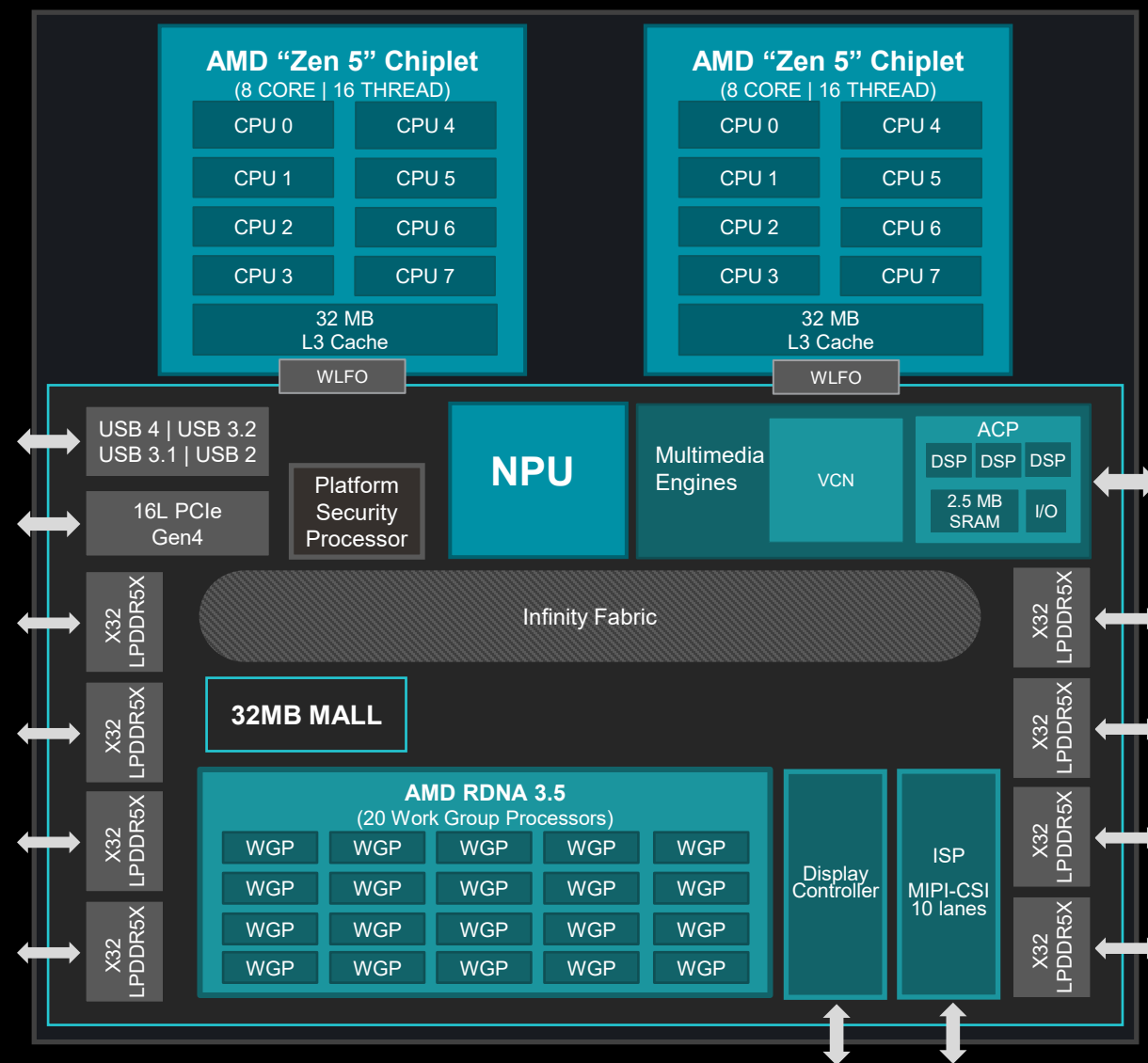
- PCIe® Gen4 (20 lanes, up to 10 host ports)
- USB-C integration of USB4 and DP 1.4a Alt mode
- USB 3.2 Gen2 10 Gb/s, NVMe, eSPI, dual QSPI for touch



AMD Ryzen™ AI Embedded X100 Series Processor

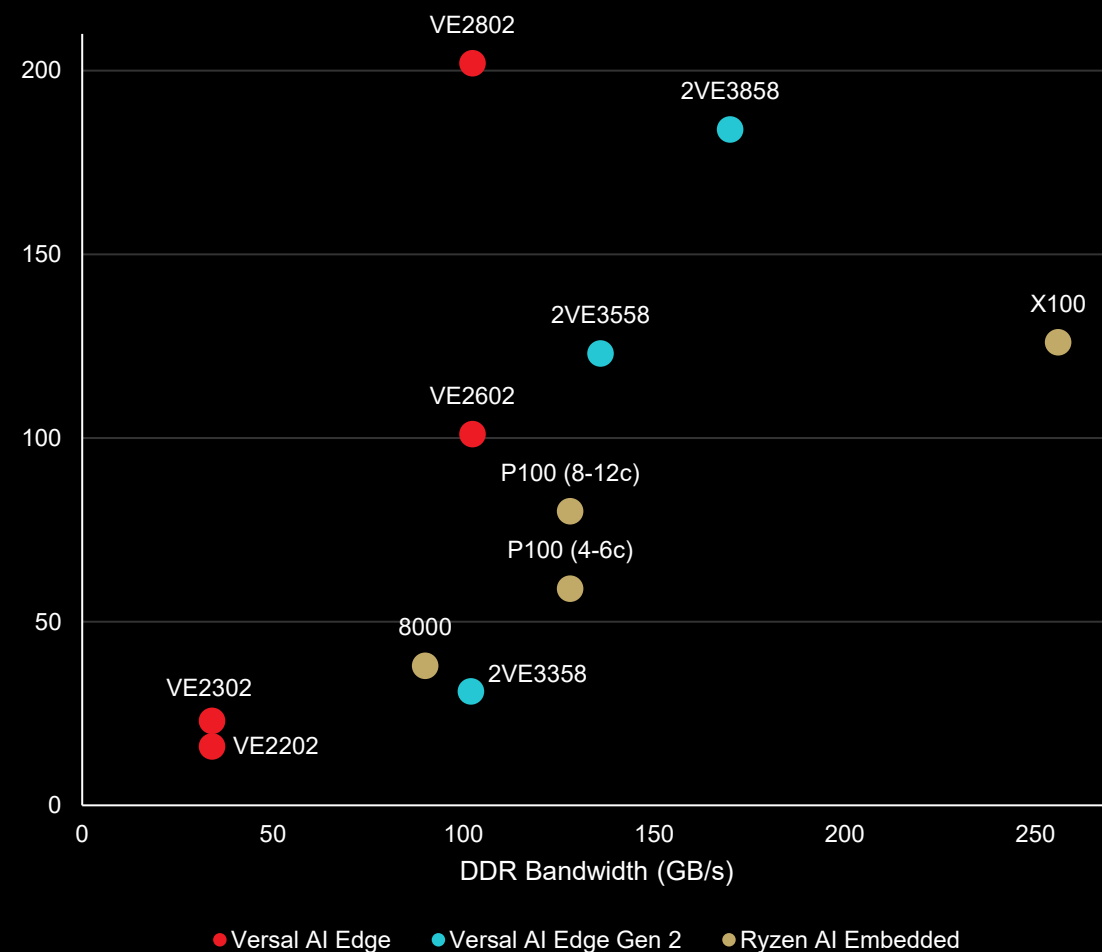
Roadmap Item – “Glacier Peak”

High Performance APU	
AMD “Zen 5” x86 CPU Cores	- Up to 16 cores / 32 threads, 2 CCD 1 MB L2\$ per core, 64 MB L3\$ cache - AVX512-ISA including VNNI, BFLOAT16
High Bandwidth SoC Fabric & Memory System	256b channels LPDDR5X-8000 (up to 256 GB/s)
Next-Generation AMD iGPU	- RDNA™ 3.5, 8 RB+, 2 MB L2, UHD render 20 WGP 4x 4K120 displays, 3 overlays, HDMI, eDP, DP, USB-C
Accelerated Multimedia Experience	- Video codec engine (VCN) with AV1 decode/encode - ACP audio offload w/2x Hifi5 DSPs, 1x Fusion F1
AI Inference Acceleration	XDNA™ 2 NPU 4x4 AIE2p, 50 TOPs INT8
Enhanced I/O	16L PCIe® Gen4 3x USB 2.0, 2x USB 3.1, 1x USB 3.2, 2x USB 4.0 w/ DP-alt-mode - MIPI-CSI2 10 lanes, x4 + x4 + x2 - eSPI, QSPI, I2C/I3C, GPIO
Power Management	Deep C-State support (fast save/resume)
Package, Power & Process	- FP11 40 x 40 x 1.38 mm @ 0.65 mm pitch BGA 45 W to 120 W TDP - N4P

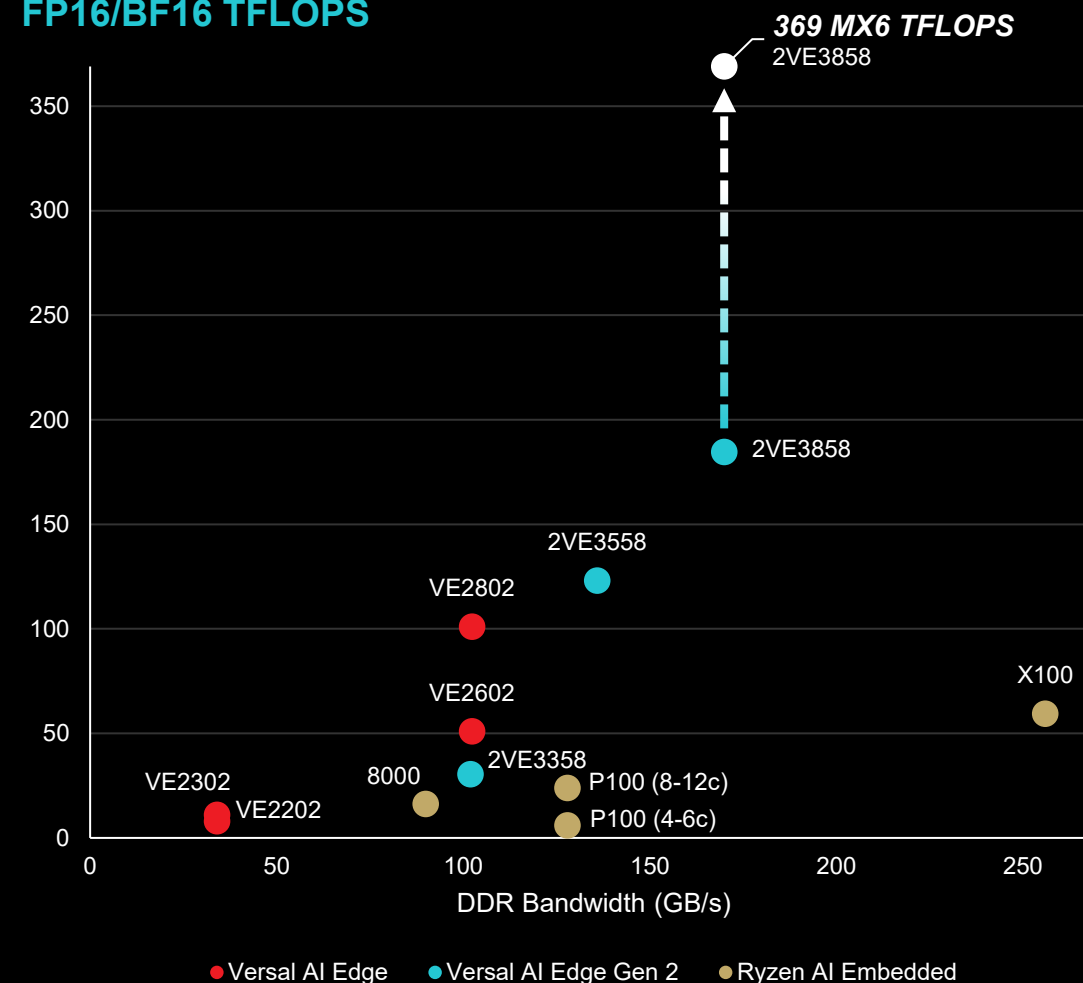


AI Compute & Memory Bandwidth Comparison

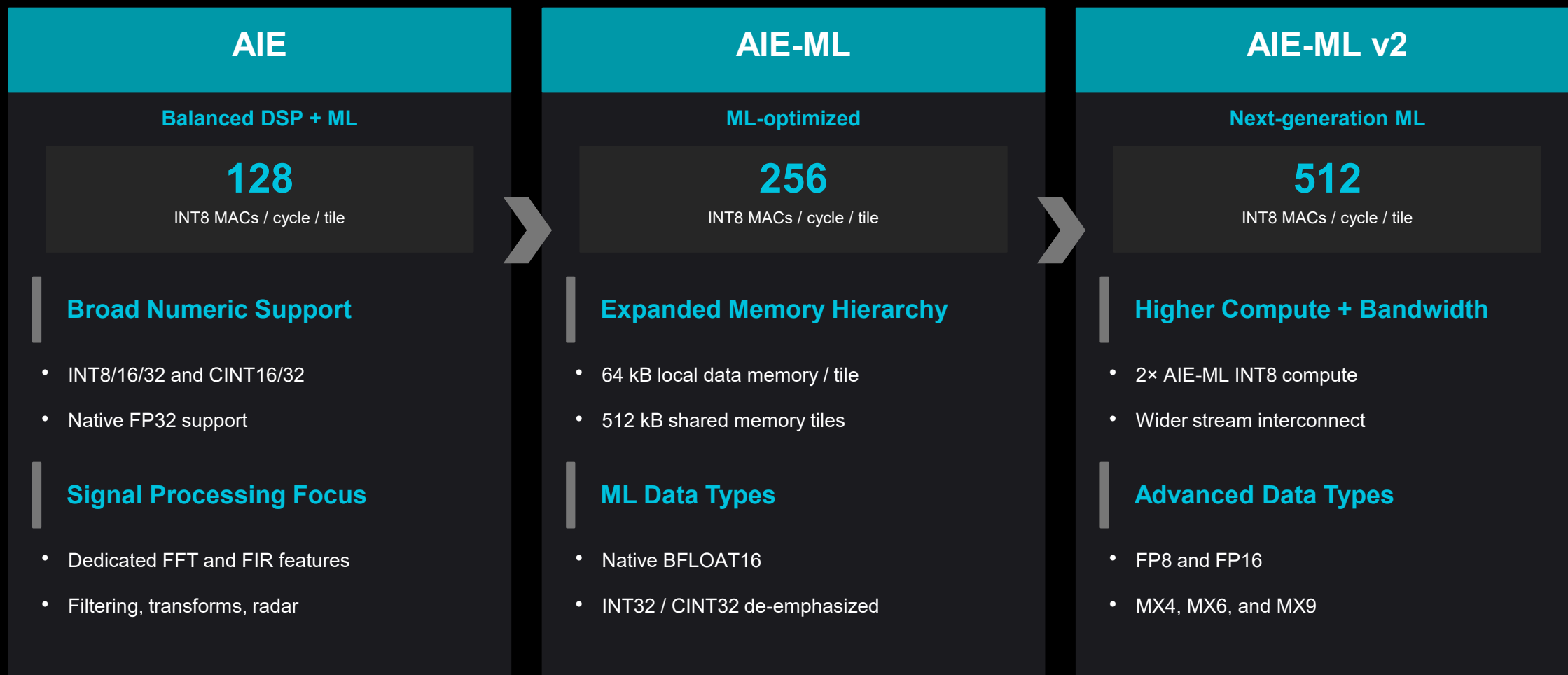
Dense INT8 TOPS



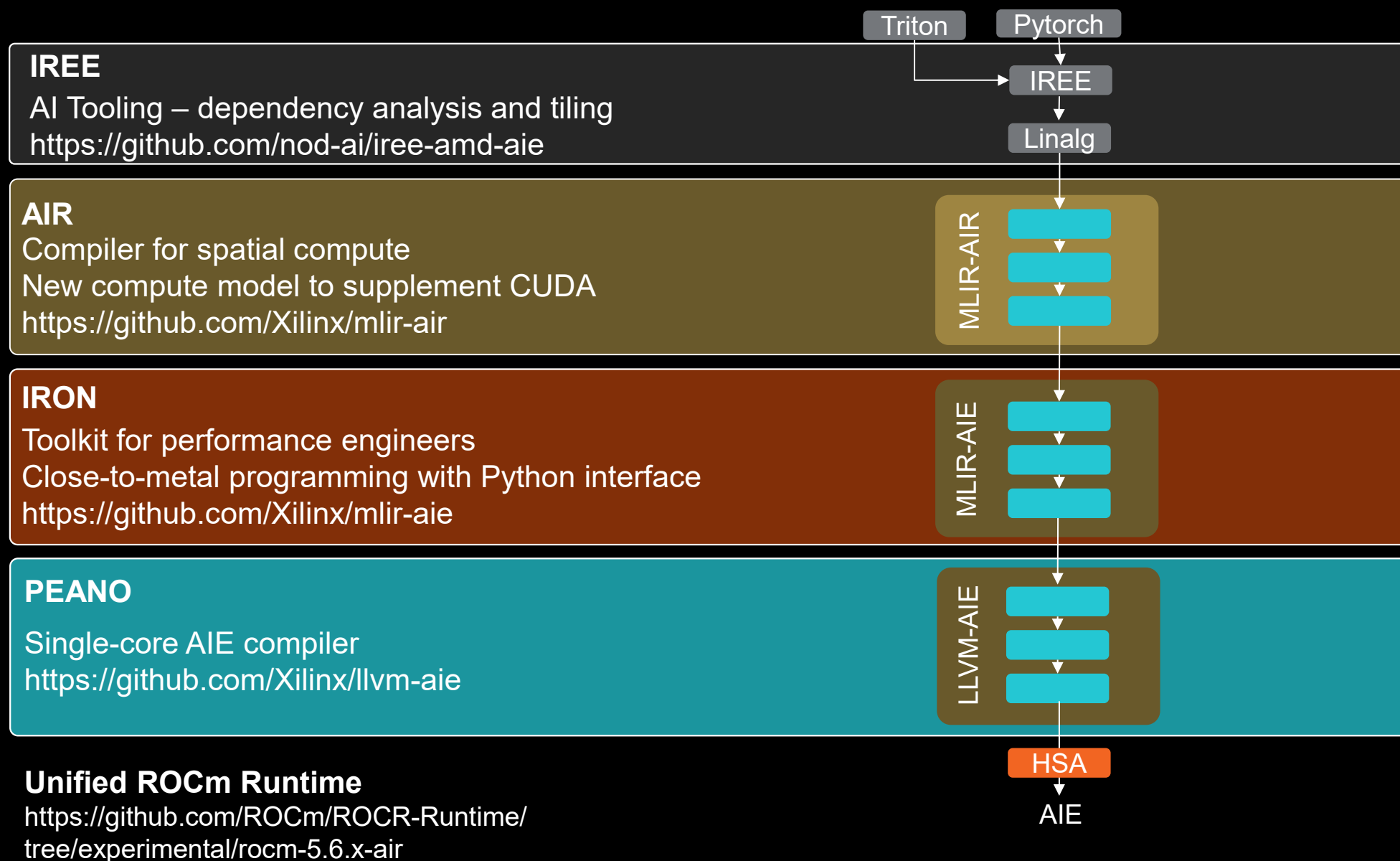
FP16/BF16 TFLOPS



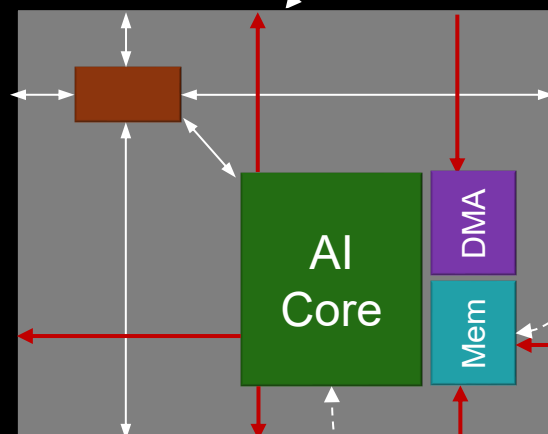
AI Engine Architecture Comparison



Enabling an Open Source MLIR Compiler Stack



Running Code on a Core

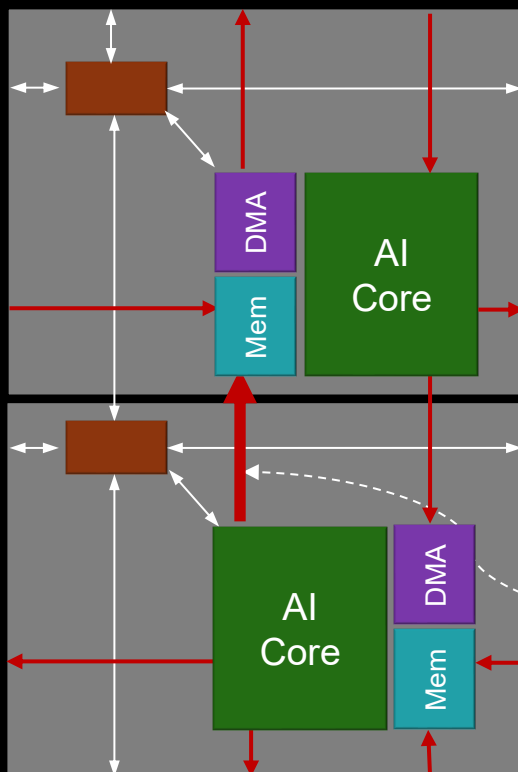


```
- %tile13 = AIE.tile(1, 3)
- %buf13_0 = AIE.buffer(%tile13)
    { sym_name = "a" } : memref<256xi32>

%core13 = AIE.core(%tile13) {
    %val1 = constant 7 : i32
    %idx1 = constant 3 : index
    %2 = addi %val1, %val1 : i32
    memref.store %2, %buf13_0[%idx1] : memref<256xi32>

    AIE.end
}
```

Moving buffers

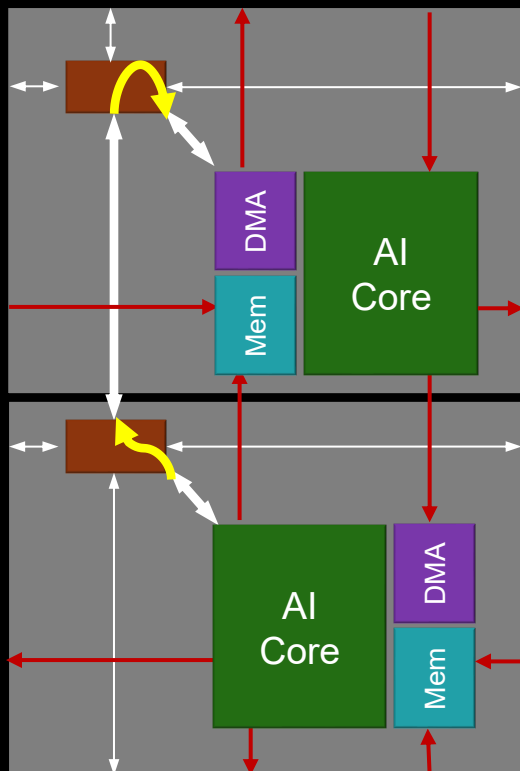


```
%tile14 = AIE.tile(1, 4)
%buf = AIE.buffer(%tile14)
      { sym_name = "a" } : memref<256xi32>

%tile13 = AIE.tile(1, 3)
%core13 = AIE.core(%tile13) {
  %val1 = constant 7 : i32
  %idx1 = constant 3 : index
  %2 = addi %val1, %val1 : i32
  AIE.useLock(%lock, "Acquire", 1)
  memref.store %2, %buf[%idx1] : memref<256xi32>
  AIE.useLock(%lock, "Release", 0)

  AIE.end
}
```

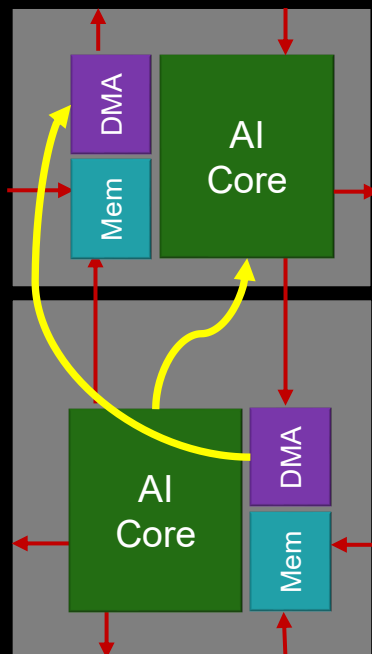
Connecting Streams



```
%tile14 = AIE.tile(1, 4)
AIE.switchbox(%tile14) {
    AIE.connect<"South" : 3, "Core" : 0>
    AIE.connect<"South" : 4, "DMA" : 0>
}

%tile13 = AIE.tile(1, 3)
AIE.switchbox(%tile13) {
    AIE.connect<"Core" : 0, "North" : 3>
    AIE.connect<"DMA" : 0 , "North" : 4>
}
```

Using Flows



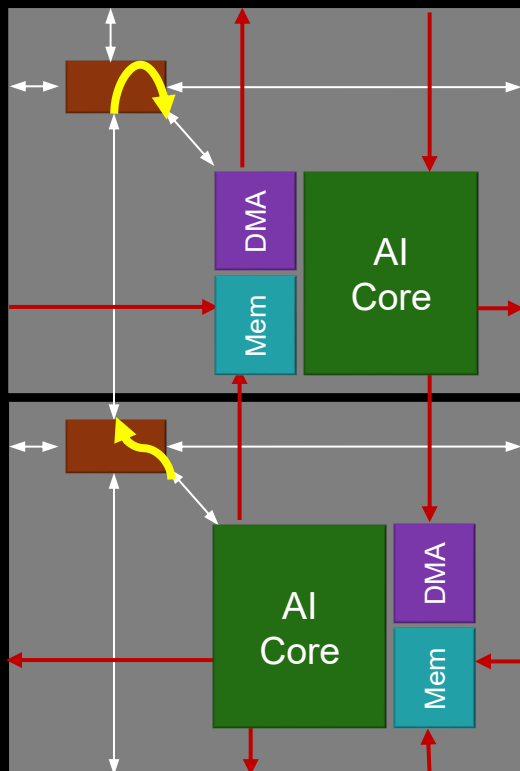
```
AIE.flow(%tile14, "DMA" : 0, %tile13, "DMA" : 0)
AIE.flow(%tile14, "Core" : 0, %tile13, "Core" : 0)
```

↓ aie-opt --aie-create-flows

```
%tile14 = AIE.tile(1, 4)
AIE.switchbox(%tile14) {
  AIE.connect<"South" : 3, "Core" : 0>
  AIE.connect<"South" : 4, "DMA" : 0>
}

%tile13 = AIE.tile(1, 3)
AIE.switchbox(%tile13) {
  AIE.connect<"Core" : 0, "North" : 3>
  AIE.connect<"DMA" : 0, "North" : 4>
}
```

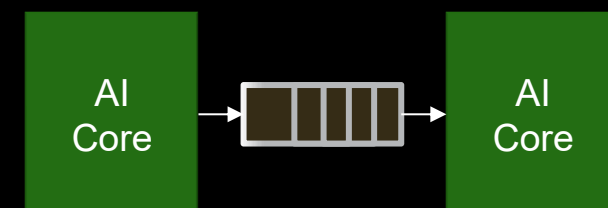
Using DMAs



```
%tile14 = AIE.tile(1, 4)
%mem14 = AIE.mem(%tile14) {
  %dma0 = AIE.dmaStart("S2MM", 0, ^bd0, ^end)
  ^bd0:
    AIE.useLock(%lock14_0, "Acquire", 1)
    AIE.dmaBd(<%buf: memref<512xi32>, 0, 512>, 0)
    AIE.useLock(%lock14_0, "Release", 0)
    br ^bd0
  ^end:
  AIE.end
}
%core14 = AIE.core(%tile14) {
  AIE.useLock(%lock, "Acquire", 0, 0)
  %data = memref.load %buf[%idx1] : memref<512xi32>
  AIE.useLock(%lock, "Release", 1, 0)
}
```

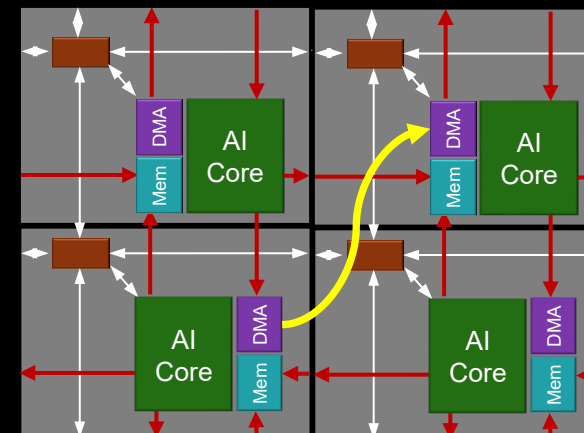
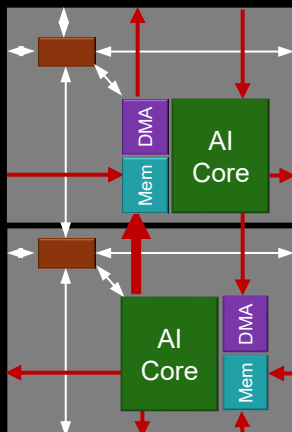
Abstract Communication with Object Fifo

```
%f = AIE.objectFifo.createObjectFifo(%src, {%dst}, 2)
AIE.core(%src) {
    scf.for %indexInLine = %c0 to %c16 step %c1 {
        %v1 = AIE.objectFifo.acquire<Produce>(%objFifo, 1)
        ...
        AIE.objectFifo.release<Produce>(%objFifo, 1)
    }
    AIE.end
}
AIE.core(%dst) {
    scf.for %indexInLine = %c0 to %c16 step %c1 {
        %v1 = AIE.objectFifo.acquire<Consume>(%objFifo, 1)
        ...
        AIE.objectFifo.release<Consume>(%objFifo, 1)
    }
    AIE.end
}
```



ObjectFifo Implementation

aie-opt --aie-register-objectFifos --aie-objectFifo-stateful-transform --aie-lower-multicast



```
%4 = AIE.buffer(%0) : memref<16xi32>
%5 = AIE.lock(%0, 0)
%6 = AIE.buffer(%0) : memref<16xi32>
%7 = AIE.lock(%0, 1)
%8 = AIE.core(%0) { ... }
%9 = AIE.core(%1) { ... }
```

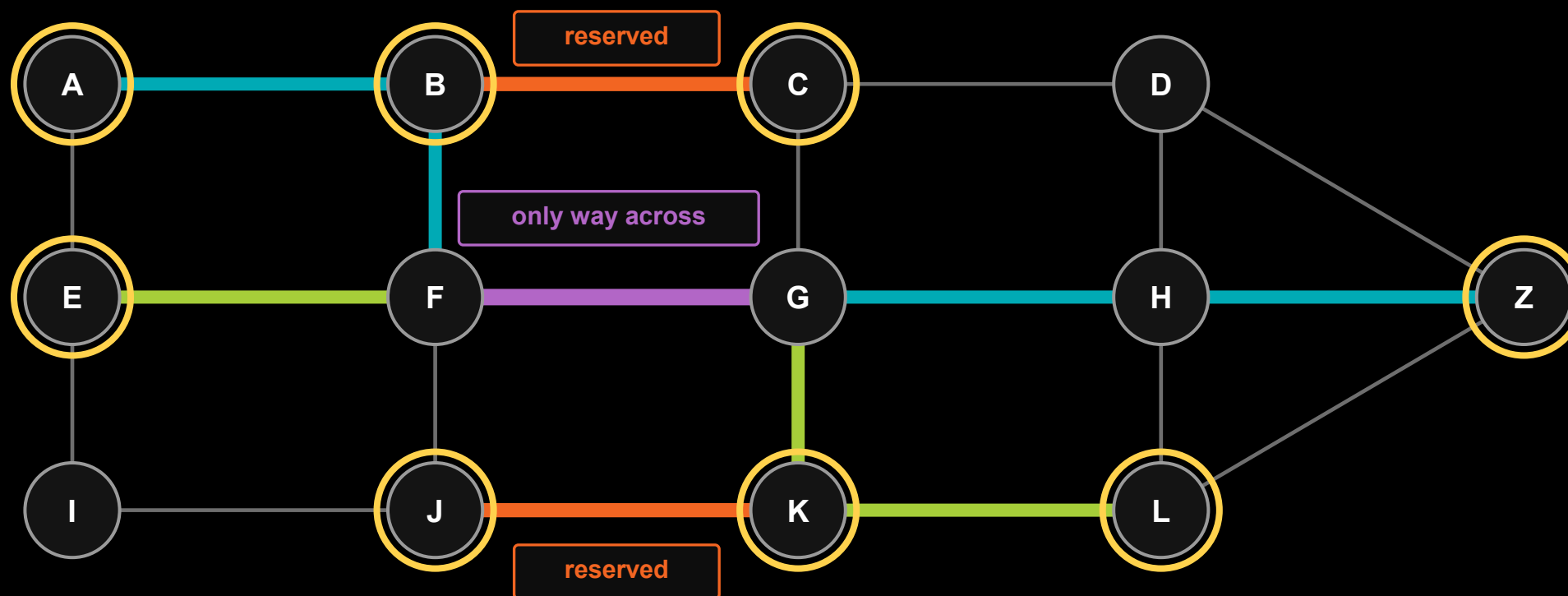
```
%6 = AIE.buffer(%0) : memref<16xi32>
%7 = AIE.lock(%0, 0)
%8 = AIE.buffer(%0) : memref<16xi32>
%9 = AIE.lock(%0, 1)
%10 = AIE.buffer(%2) : memref<16xi32>
%11 = AIE.lock(%2, 1)
%12 = AIE.buffer(%2) : memref<16xi32>
%13 = AIE.lock(%2, 2)
%14 = AIE.core(%0) { ... }
%15 = AIE.core(%2) { ... }
%16 = AIE.mem(%0) { ... }
%17 = AIE.mem(%2) { ... }
AIE.flow(%0, DMA : 0, %1, DMA : 0)
```

Congestion: circuit flows leave one way across

Circuit $B \rightarrow C$ and $J \rightarrow K$

Packet $A \rightarrow Z$

Packet $E \rightarrow L$

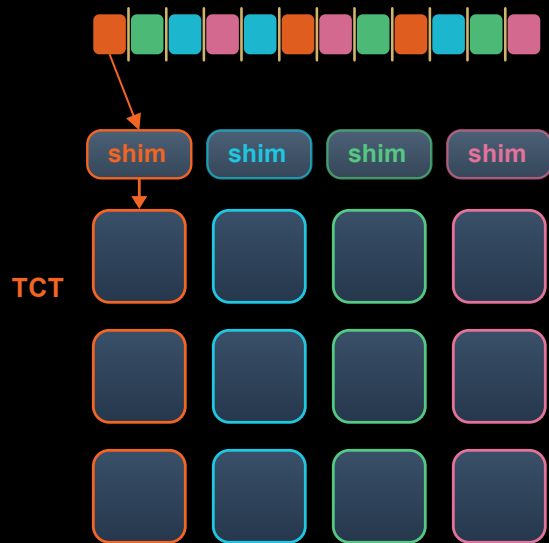


Three channels cross the middle: $B \rightarrow C$, $F \rightarrow G$ and $J \rightarrow K$. $B \rightarrow C$ and $J \rightarrow K$ are reserved, so everything else crossing the middle must take $F \rightarrow G$. Both packet flows merge there at $M/2$ and fan out at G — the alternative is not routing at all.

Reconfiguring in parallel

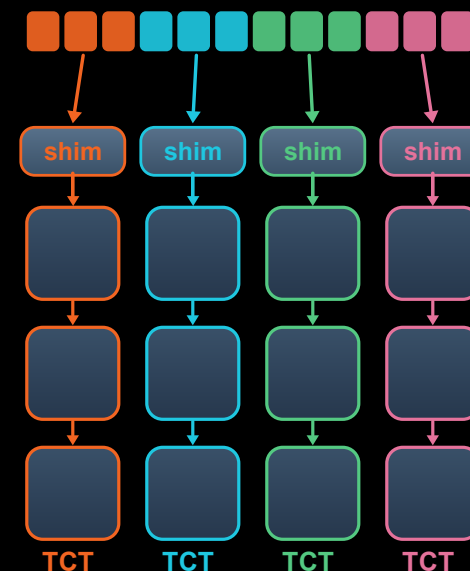
- Each column configures its own column with its own shim DMA → can and should be parallel
 - Default payload ordering = program order, i.e., “as-emitted”
- Solution: stable sort the payload by destination column → each column’s payload = contiguous DRAM read
 - Intra-column is still serial because each tile needs a TLAST signal sent by its own BD

Serial: push + sync, program order



Host pushes one tile and waits for Task Completion Token (TCT)

Parallel: one BD chain per column, all at once



Sync on last BD's (TCT)

Results (Strix)

Per-reconfiguration latency: 4×8 cores, CONFIGS=16

(Log scale)

ctrlpkt (parallel)

128 μ s

write32 (local)

337 μ s

write32 (full)

413 μ s

ctrlpkt (serial)

837 μ s

warm

2.17 ms

loadpdi

2.43 ms

cold

80.5 ms

100 μ s

1ms

10ms

100ms

Parallel ctrlpkts: ~17× over warm

~3.2× over --expand-load-pdis

Localized effect: ~1.2× over full

--expand-load-pdis: ~5.3× over warm

But pays cold cost on cap

Sublinear parallel speedup: ~6.5×

■ [1 binary : N configs] ×1

■ [1 binary : 1 config] ×N – 32 context cap

■ Baseline

Conclusion

- Data Locality is a fundamental concept
- Although very different, GPUs, FPGAs, and NPUs have some similar programming challenges
 - Especially for AI!
- Picking the right embedded target often comes down to data types, interfaces, power efficiency, etc.

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2026 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Infinite Cache, CDNA, Instinct, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Python logo and any related marks are trademarks of The Python Software Foundation. **All other product names used in this presentation are for identification purposes only and may be trademarks of their respective owners.**

ENDNOTES

VER-023

Based on AMD internal pre-silicon performance estimates and power projections for the AIE-ML v2 compute tile architecture featured in the Versal AI Edge Series Gen 2 using the MX6 data type compared to production performance specifications and AMD Power Design Manager power estimates for the AIE-ML compute tile architecture featured in the first generation Versal AI Edge series using the INT8 data type. Operating conditions: 1 GHz Fmax, 0.7V AIE operating voltage, 85C junction temperature, typical process, 60% vector load, % activations = 0 < 10%. Actual performance will vary when final products are released in market. Performance projections as of February 2024.

VER-024

Based on the product specification of 1024 INT8 ops/clock for the AIE-ML v2 compute tile architecture featured in the Versal AI Edge Series Gen 2 compared to published product specifications of 512 INT8 ops/clock for the AIE-ML compute tile architecture featured in the first generation Versal AI Edge series. Performance projections as of February 2024.

