

# Leveraging MLIR to Design for AI Engines

@FCCM2023 Andra Bisca, Stephen Neuendorffer, Erwei Wang

@our\_team

Samuel Bayliss, Kristof Denolf, Jeff Fifield, Phil James-Roxby, Alireza Khodamoradi, Jack Lo, Joseph Melber, Eddie Richter, Mark Sears, Ralph Wittig

**AMD Research and Advanced Development**

**AMD**   
together we advance\_

# Agenda: Leveraging MLIR to Design for AI Engines

## Thursday May 11<sup>th</sup>, 2023, 09:00-12:00 PST

Time	Topic	Presenter	HandsOn Online tutorial #
09:00 – 09:05	Welcome and Logistics	Steve	
09:05 – 09:30	Introduction to MLIR and AIEngine	Erwei	
9:30 – 10:15	Physical MLIR-AIE	Steve	Tutorial 1
10:15 – 10:30	Break + AWS Troubleshoot		
10:30 – 11:15	Host Code, Simulation and Performance Counters	Andra	Tutorial 2 (a,b,c)
11:15 – 11:45	Logical MLIR-AIE: Communication	Andra	Tutorials 3,4,5
11:45 – 11:59	Advanced Topics and Next Steps	Erwei	Tutorial 11 or design examples
11:59 – 12:00	Wrap-Up	Steve	

<https://github.com/Xilinx/mlir-aie/tree/main/tutorials>

# Hands on: AWS Login and VCK-190s in the Back

- AWS Login details (details for login are sent to your email)

1. <https://console.aws.amazon.com/ec2/>

- Account ID – xilinx-aws-f1-developer-labs
- IAM user name – user# that was sent to your email (e.g. user5)
- Password – xup\_aie2023

2. Click on (1) “Instances”, select (2) “user#”, select (3) “Instance state” and “Start Instance”

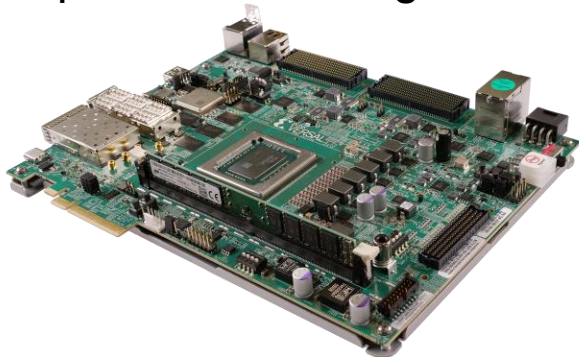
3. Download NICE Desktop Cloud Visualization Client

- <https://download.nice-dcv.com/>

4. Connect to (4) IP Address via NICE DCV

- Username: ubuntu, Password: xup\_aie2023

- Optional: Experience running code on a VCK-190 board



aws

Sign in as IAM user

Account ID (12 digits) or account alias

xilinx-aws-f1-developer-labs

IAM user name

user5

Password

.....

Remember this account

Sign in

[Sign in using root user email](#)

[Forgot password?](#)

aws Services Search [ALT+S] [N. Vir]

New EC2 Experience Tell us what you think

EC2 Dashboard

EC2 Global View

Events

Tags

Limits

**Instances**

Instance types

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts

Scheduled Instances

Capacity Reservations

Images

Instances (1/8) Info

Find instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
pavan-test	i-0b74a11b8e12rcurr	Stopped	t2.micro	-	User: arn:aws:iam::123456789012:user:pavan-test
flexlm-license-...	i-0bcf1eb1b6af12a74	Running	t2.micro	2/2 checks passed	User: arn:aws:iam::123456789012:user:flexlm-license-...
user31	i-0431284e3037e28a2	Stopped	c5a.2xlarge	-	User: arn:aws:iam::123456789012:user:user31
user31	i-0a906a9ad31b9d524	Stopped	c5a.2xlarge	-	User: arn:aws:iam::123456789012:user:user31
user4	i-0710549abb1376cf3	Stopped	m5a.4xlarge	-	User: arn:aws:iam::123456789012:user:user4
user3	i-07f2efda52a4e9794	Stopped	m5a.4xlarge	-	User: arn:aws:iam::123456789012:user:user3
<b>user5</b>	<b>i-0e9c3961ad0712597</b>	<b>Running</b>	<b>m5a.4xlarge</b>	<b>2/2 checks passed</b>	<b>User: arn:aws:iam::123456789012:user:user5</b>
user2	i-0f37ef15bab8c74e7	Stopped	m5a.4xlarge	-	User: arn:aws:iam::123456789012:user:user2

Instance: i-0e9c3961ad0712597 (user5)

Details Security Networking Storage Status checks Monitoring Tags

Instance summary Info

Instance ID i-0e9c3961ad0712597 (user5)

Public IPv4 address 3.239.97.192 | Open address

Private IPv4 address 172.31.78.102

# Introduction to AIEngines

- Welcome & Logistics
- Introduction to AIEngines
- Physical MLIR-AIE
- Host Code, Simulation, Performance Counters
- Logical MLIR-AIE: Communication
- Advanced Topics and Next Steps
- Wrap Up

# Versal ACAP Architecture



**Adaptable Engines**  
2X compute density



## Scalar Engines

- Platform Control
- Edge Compute



## Protocol Engines

- Integrated 600G cores
- 4X encrypted bandwidth



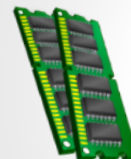
## Programmable I/O

- Any interface or sensor
- Includes 4.2Gb/s MIPI



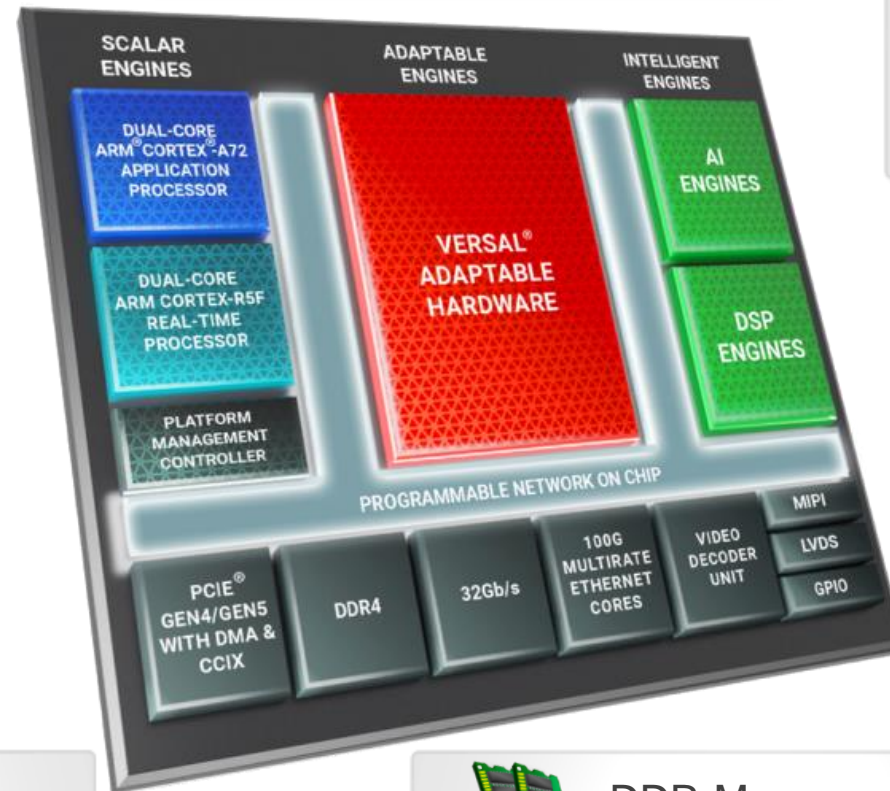
## PCIe & CCIX

- 2X PCIe & DMA bandwidth
- Cache-coherent interface to accelerators



## DDR Memory

- 3200-DDR4, 3200-LPDDR4
- 2X bandwidth/pin



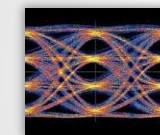
## AI Engines

- AI Compute
- Diverse DSP workloads



## Network-on-Chip

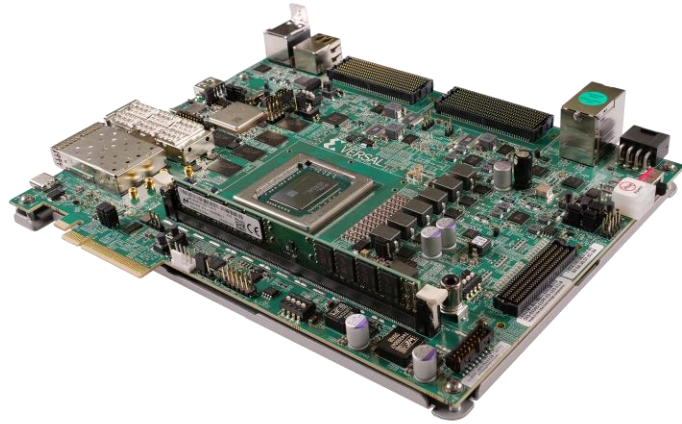
- Guaranteed Bandwidth
- Enables SW Programmability



## Transceivers

- Broad range, 25G →112G
- 58G in mainstream devices

# Yes, It's Really Here



VCK190  
1<sup>st</sup> gen AIE



VCK5000  
1<sup>st</sup> gen AIE



Announcing  
**AMD RYZEN™ 7040 Series**  
The Ultimate Ultrathin Processor

<b>ZEN 4</b> PROCESSOR ARCHITECTURE	<b>AMD RDNA 3</b> GRAPHICS ARCHITECTURE
<b>AMD XDNA</b> AI ENGINE ARCHITECTURE	<b>4nm</b> MANUFACTURING TECHNOLOGY

Featuring Ryzen™ AI



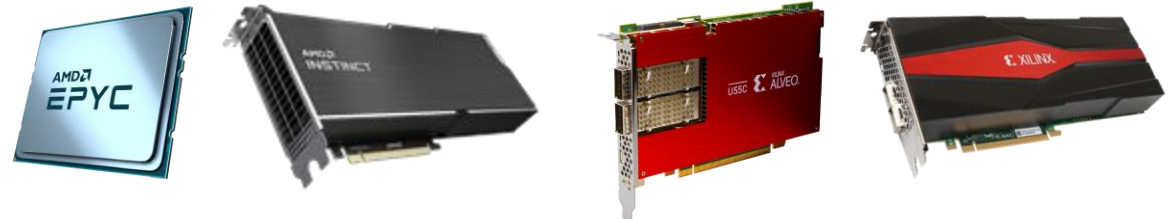
V70  
AMD XDNA  
2<sup>nd</sup> gen AIE-ML

# Heterogenous Accelerated Compute Clusters (HACC)

- Extend scope to conduct state-of-the-art research on heterogeneous accelerated computing by incorporating GPUs into CPU+FPGA/AIE systems
- The scope still encompasses systems, architecture, tools and applications
- Internal senior technical advisors now across the broader scope
- Heterogeneous reference systems
  - On-premise at the HACC sites (infrastructure research and innovation)
- Tightly integrated reference system
- Per node (SMC 4124GS)
  - 2 EPYC Milan/X CPUs
  - 2-4 MI210 GPUs
  - 2 Alveo U55C FPGA with HBM
  - 2 VCK-5000 ACAP/Versal with AIEs
  - Direct inter-board connections where appropriate
  - Run-time via ROCm, XRT
  - SW development via HIP, Vitis, frameworks



How can we target all of these heterogeneous devices?

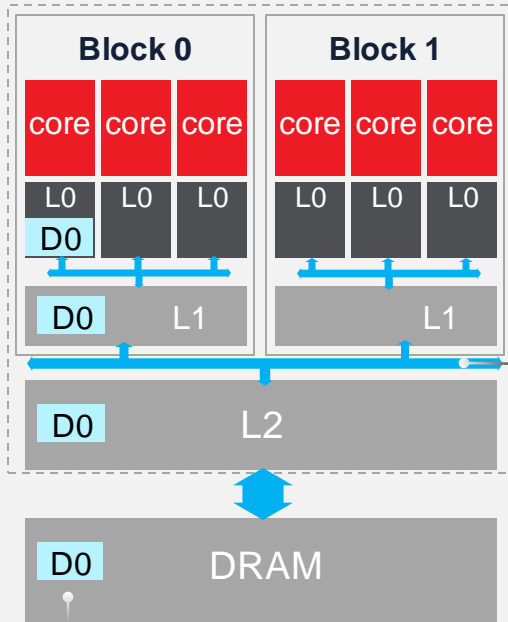


# **AI Engine Architecture**



# AI Engine: Multi-Core Compute with Dedicated Memory

## Traditional Multi-core (cache-based architecture)



Fixed, Shared Interconnect

- Blocking limits compute
- Timing not deterministic

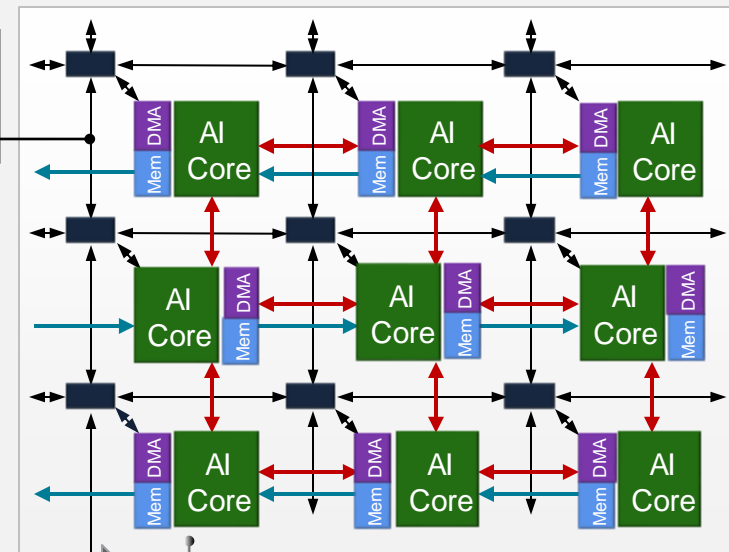
Data Replicated

- Robs bandwidth
- Reduces capacity

## AI Engine Array (adaptable distributed architecture)

Dedicated Interconnect

- Non-blocking
- Deterministic



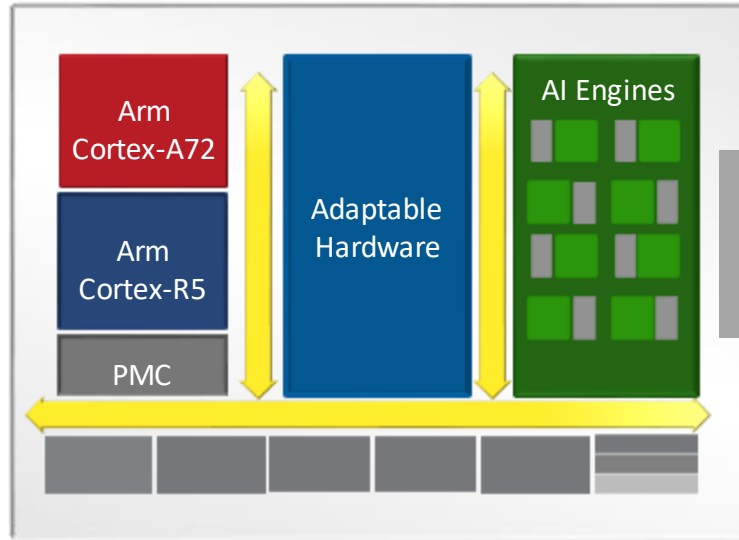
Memory Tile  
(Up to 38MB)

Local, Distributed Memory

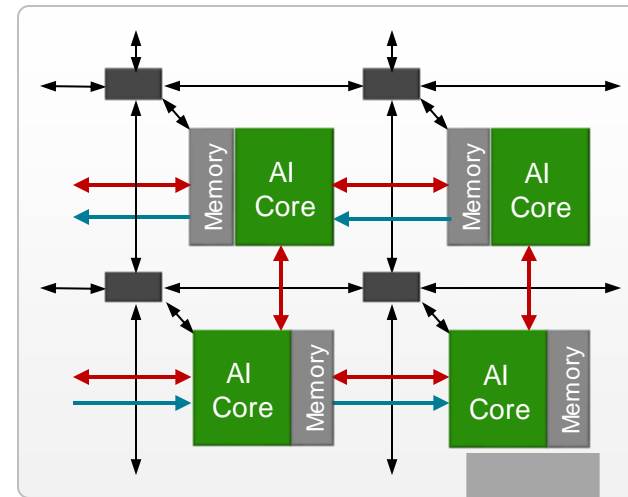
- No cache misses
- Higher bandwidth
- Less capacity required

# AI Engine: Terminology

Versal ACAP

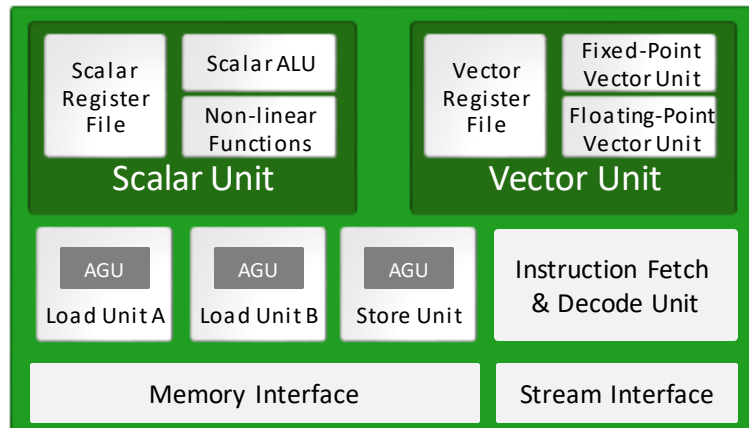


AI Engine Array

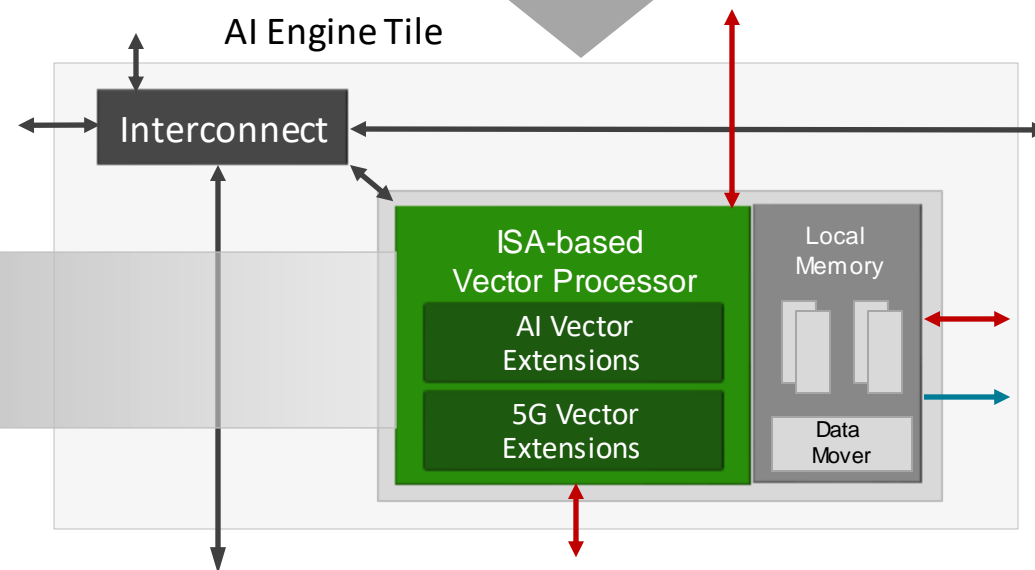


- Memory Interface
- Stream Interface
- Cascade Interface

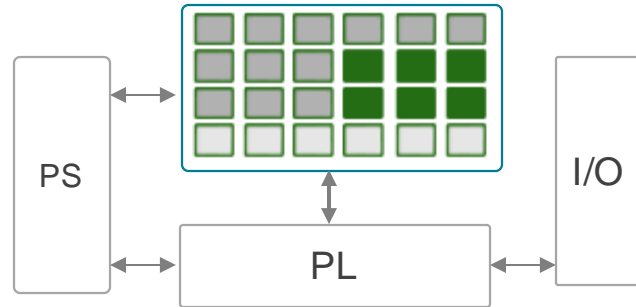
AI Engine Core



AI Engine Tile

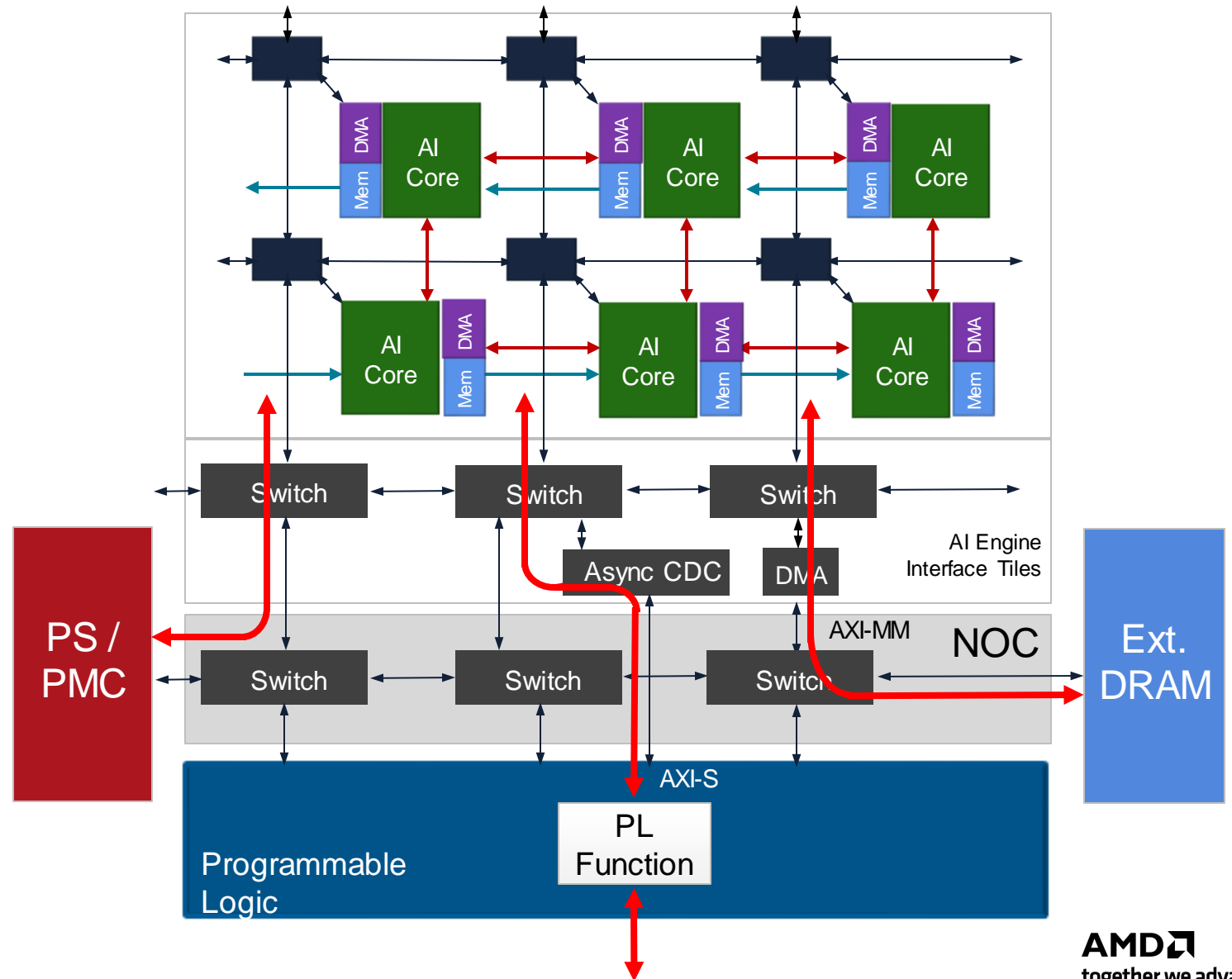


# AI Engine Integration with Versal ACAP

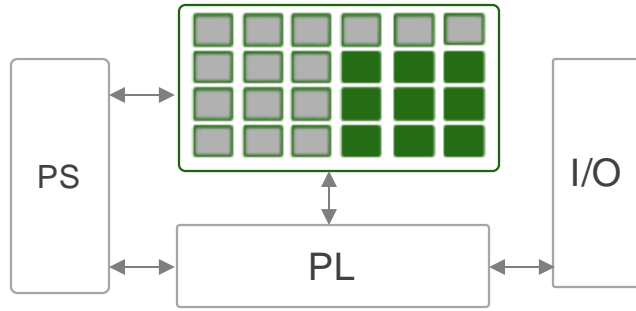


- TB/s of Interface Bandwidth
  - AI Engine to Programmable Logic
  - AI Engine to NOC
- Leveraging NOC connectivity
  - PS manages Config/Debug/Trace
  - AI Engine to DRAM (no PL req'd)

- ➔ Memory Interface
- ➔ Stream Interface
- ➔ Cascade Interface



# AI Engine: Array Architecture



## Modular and scalable architecture

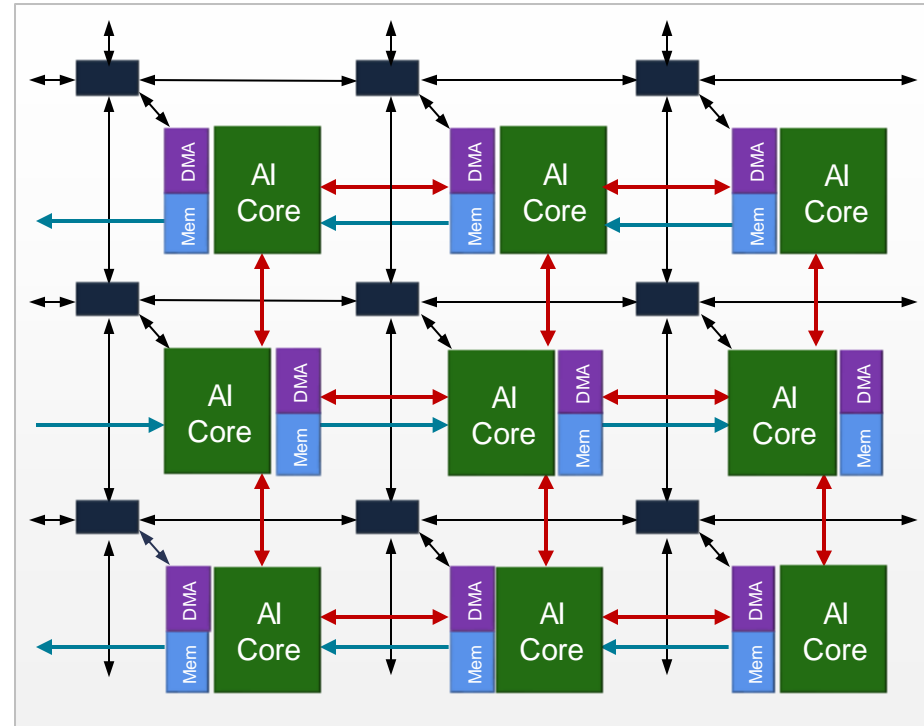
- More tiles = more compute
- Up to 400 per device
  - Versal AI Core VC1902 device

## Distributed memory hierarchy

Maximize memory bandwidth

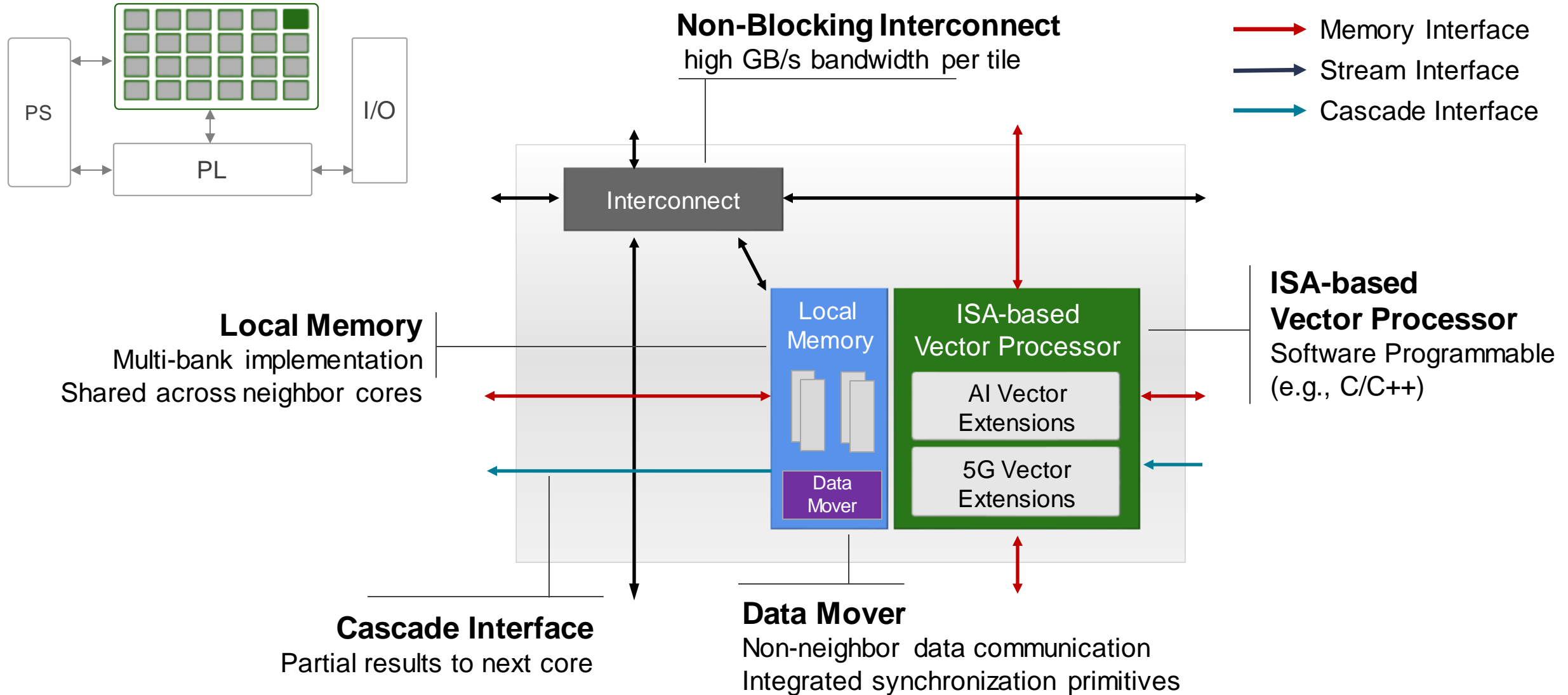
## Array of AI Engines

- Increase in compute, memory and communication bandwidth



Deterministic Performance & Low Latency

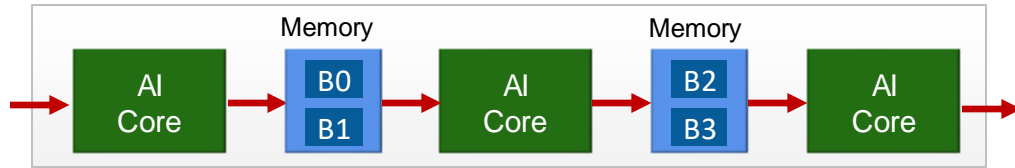
# AI Engine: Tile-Based Architecture



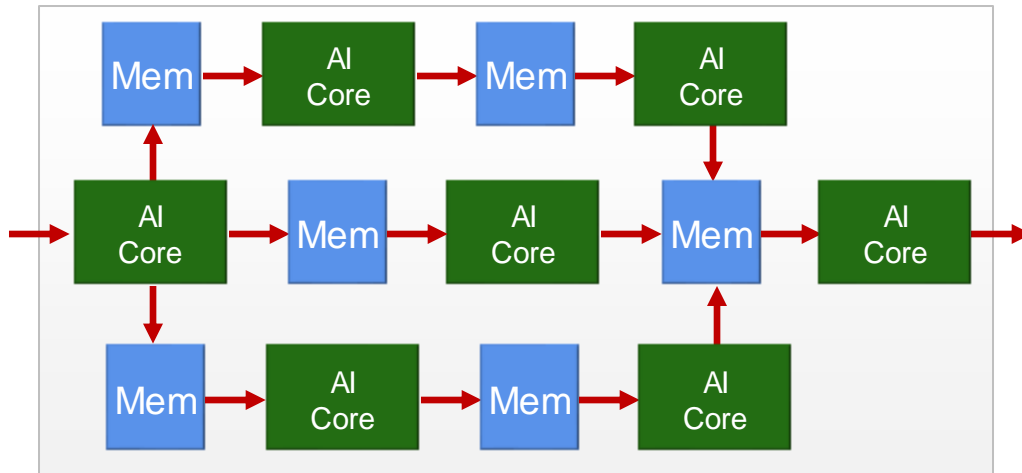
# Data Movement Architecture

## Memory Communication

Dataflow Pipeline



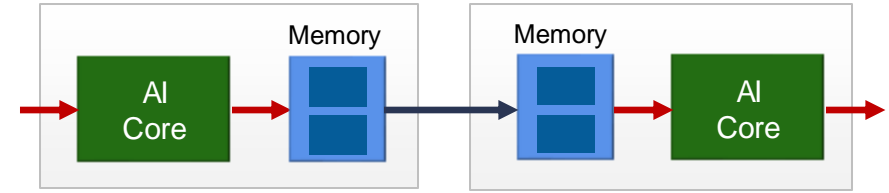
Dataflow Graph



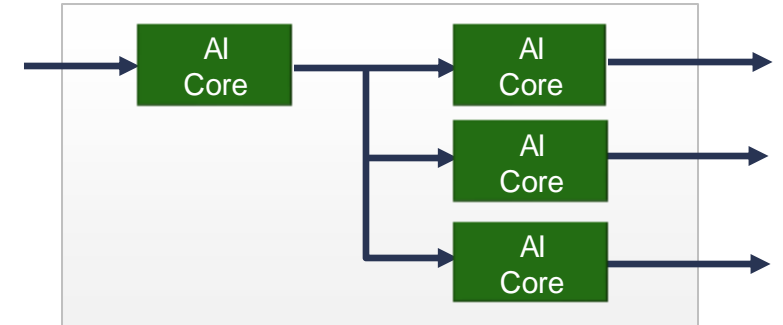
- Memory Interface
- Stream Interface
- Cascade Interface

## Streaming Communication

Non-Neighbor



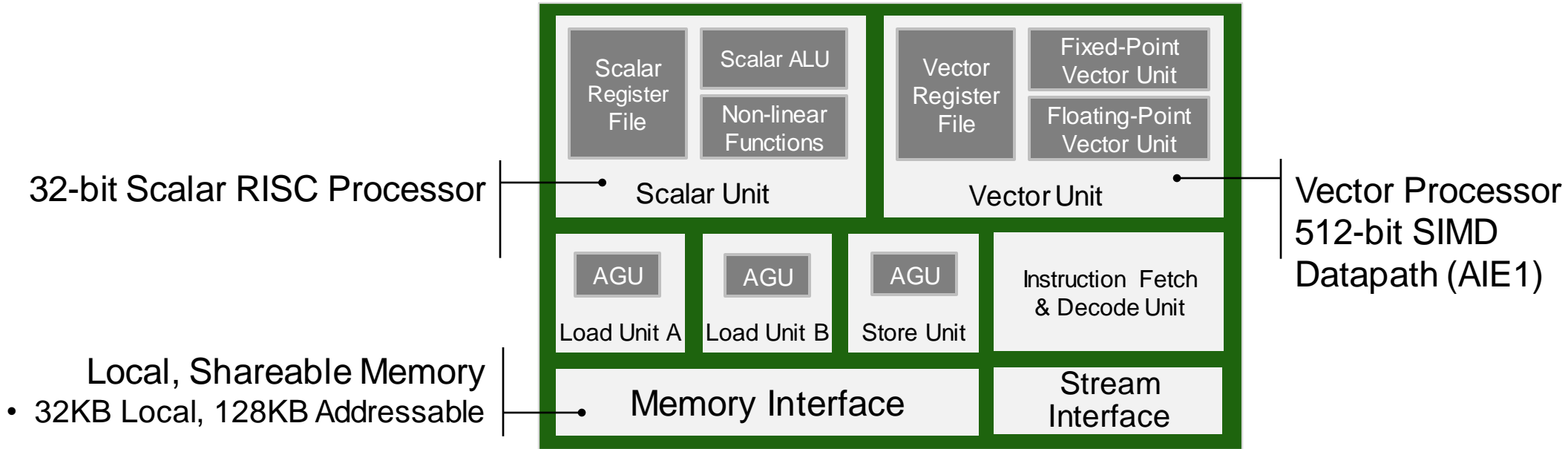
Streaming Multicast



Cascade Streaming



# AI Engine: Processor Core



## Instruction Parallelism: VLIW

7+ operations / clock cycle

- 2 Vector Loads / 1 Mult / 1 Store
- 2 Scalar Ops / Stream Access

Highly Parallel

## Data Parallelism: SIMD

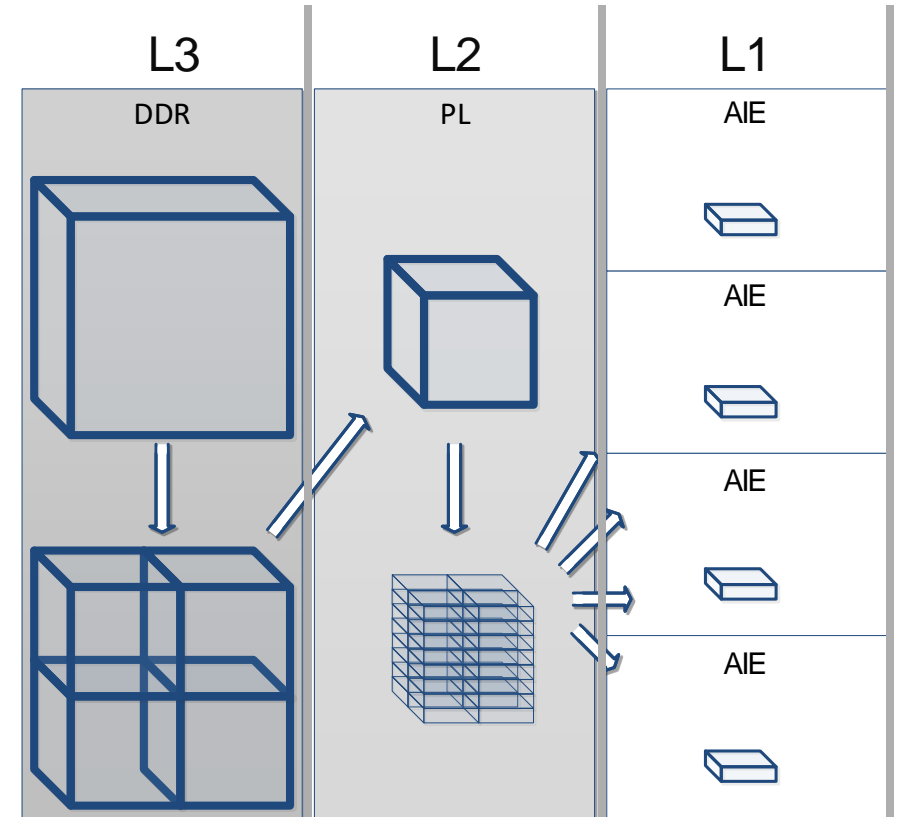
Multiple vector lanes

- Vector Datapath
- 8 / 16 / 32-bit & SPFP operands

Up to 128 MACs / Clock Cycle per Core (INT 8)

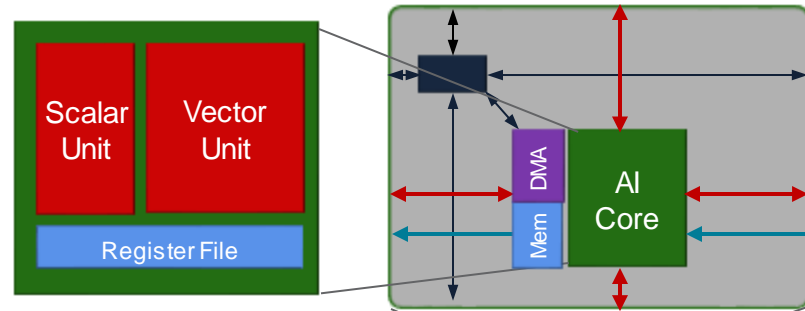
# Challenge: Handling Large Data Sets

- Many applications can't easily be streamed
  - 3D Volumetric Data -> Weather Prediction
  - 2D Batched Data -> Image-based Machine Learning
  - 1D Time-Series Data -> 5G Beamforming/SAR
- Often large data sets (~100GB)
  - Doesn't fit on-chip
- Data must be tiled for processing on AIE
- Data must be prefetched and streamed
- Data reuse must be considered at multiple levels





# Memory Reuse is Critical!

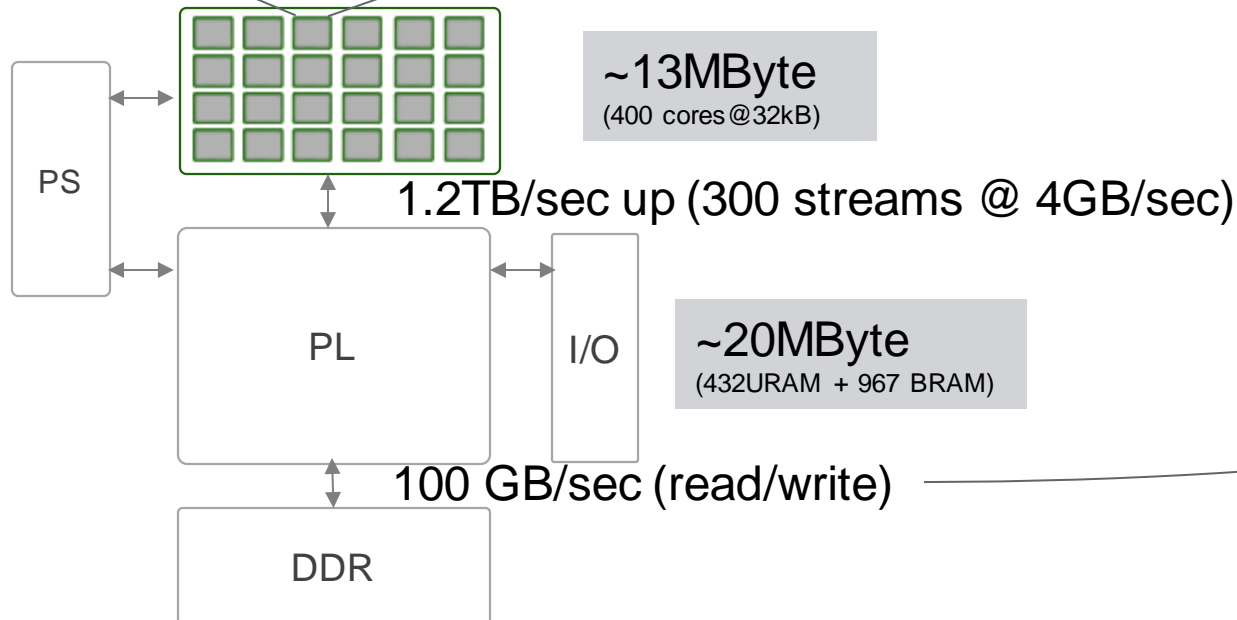


102TB/sec  
(400 cores @ 2x128 bytes per instruction)

25.6TB/sec  
(400 cores @ 2x 256-bit load per instruction)

4x register file reuse

21.3x core memory reuse (no broadcast)  
2.7x core memory reuse (8x broadcast)



~13MByte  
(400 cores @ 32kB)

~20MByte  
(432URAM + 967 BRAM)

1.2TB/sec up (300 streams @ 4GB/sec)

100 GB/sec (read/write)

12x on-chip reuse

# Adaptability Where You Want It, Efficiency Where You Need It

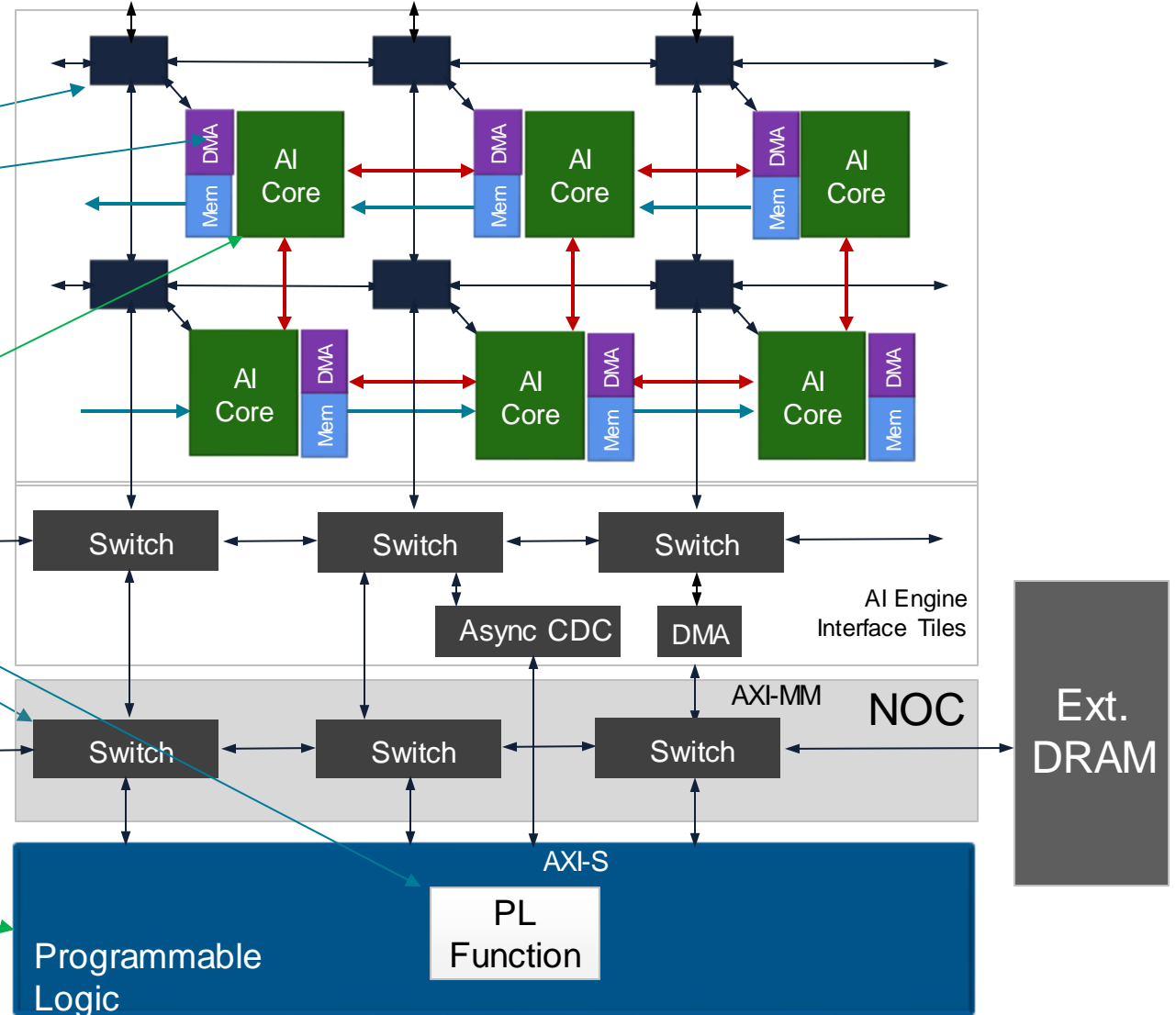
- Accessible

- Adaptable

- Compute

- ➔ Memory Interface
- ➔ Stream Interface
- ➔ Cascade Interface

PS /  
PMC



Ext.  
DRAM

# MLIR-AIE

- Welcome & Logistics
- Introduction to AIEngines
- Physical MLIR-AIE
- Host Code, Simulation, Performance Counters
- Logical MLIR-AIE: Communication
- Advanced Topics and Next Steps
- Wrap Up

# Various Toolflows

- Vitis AI
  - Audience: ML programmers
  - ML programming model
  - Details of architecture abstracted using an overlay
- Vitis
  - Audience: DSP programmers
  - Graph programming model
  - Some architecture concepts exposed
- **MLIR-AIE: Open-Source Multi-level IR toolflow**
  - Audience: Researchers and Tool developers
  - Enable multiple programming abstractions
  - Architectural details exposed and accessible

<https://github.com/Xilinx/mlir-aie>

## MLIR-based AIEngine toolchain

build passing open pull requests 4



This repository contains an [MLIR-based](#) toolchain for Xilinx Versal AIEngine-based devices. This can be used to generate low-level configuration for the AIEngine portion of the device, including processors, stream switches, TileDMA and ShimDMA blocks. Backend code generation is included, targeting the LibXAIE library. This project is primarily intended to support tool builders with convenient low-level access to devices and enable the development of a wide variety of programming models from higher level abstractions. As such, although it contains some examples, this project is not intended to represent end-to-end compilation flows or to be particularly easy to use for system design.

[Full Documentation](#)

You can:

- Assemble AIEngine designs in a 'structural' way
- Compile designs relatively quickly
- Build your own higher-level tools
- Contribute to ongoing open-source development

# MLIR: Multi-Level Intermediate Representation

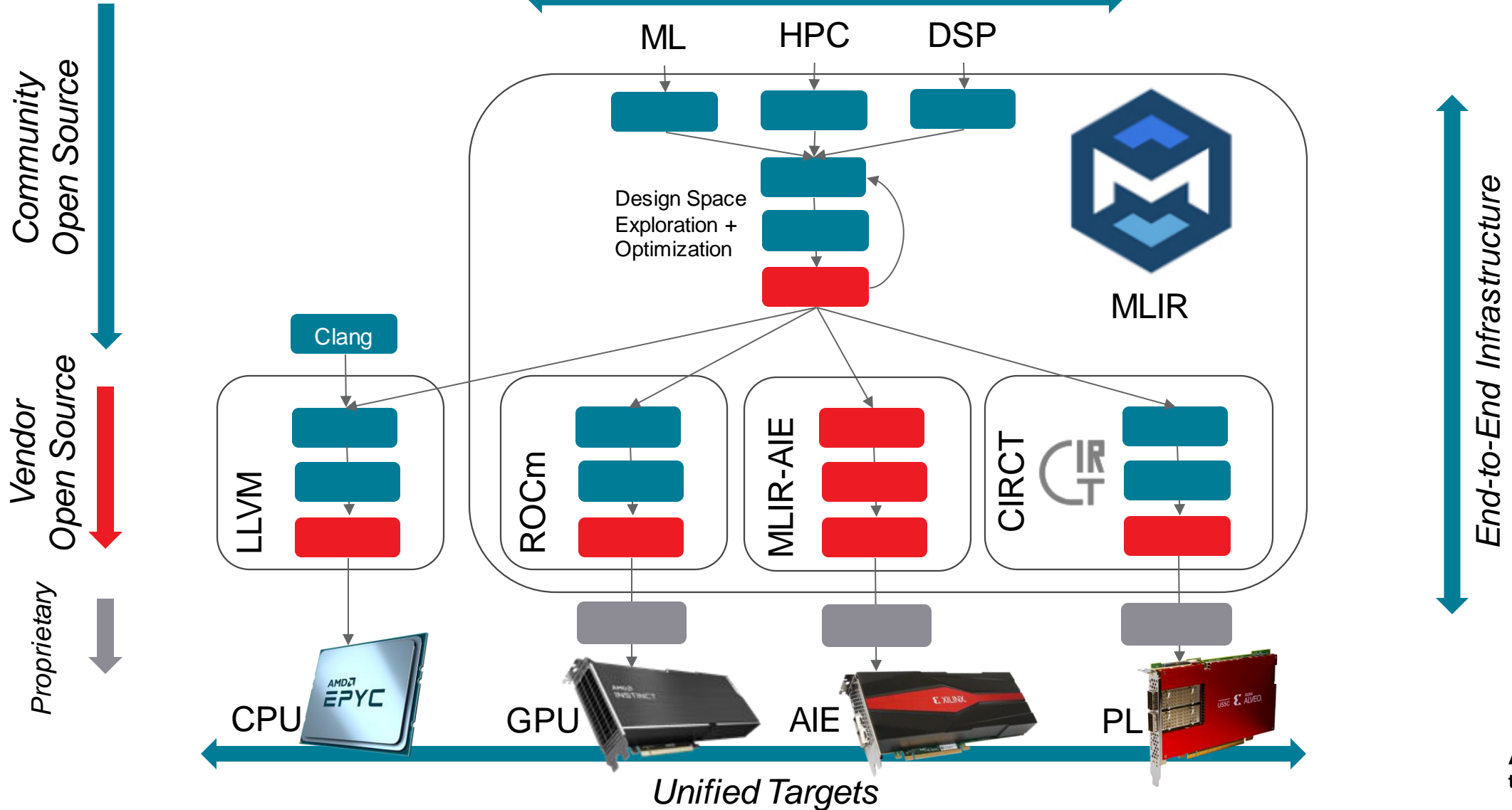
- Next generation **open source** compiler infrastructure
  - LLVM core project
- Key idea: Fundamentally Extensible
- Lots of 'Batteries Included'
  - Math, Loops, and Tensor *Dialects*
  - Many Optimizations
- Lots of Connections
  - ML Frontends, LLVM Backends



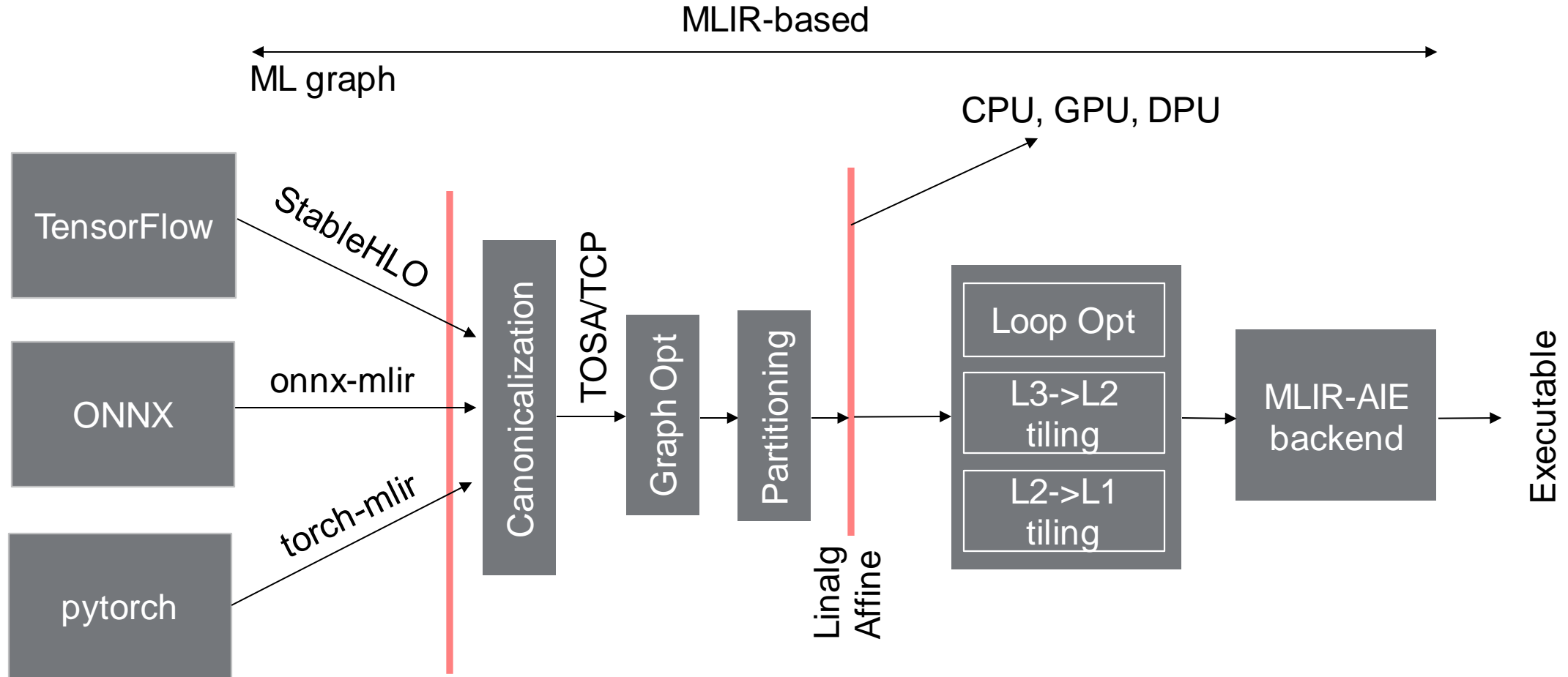
```
module {  
  %foo, %bar = myDialect.myOp(%biz, %baz) {  
    %1 = math.add(%biz, %baz) : i32  
    ^bb1:  
      myDialect.myOtherOp(%1)  
    ^bb2:  
      myDialect.goto ^bb1  
  }  
}
```

# Future Heterogeneous Programming

Multiple Domains

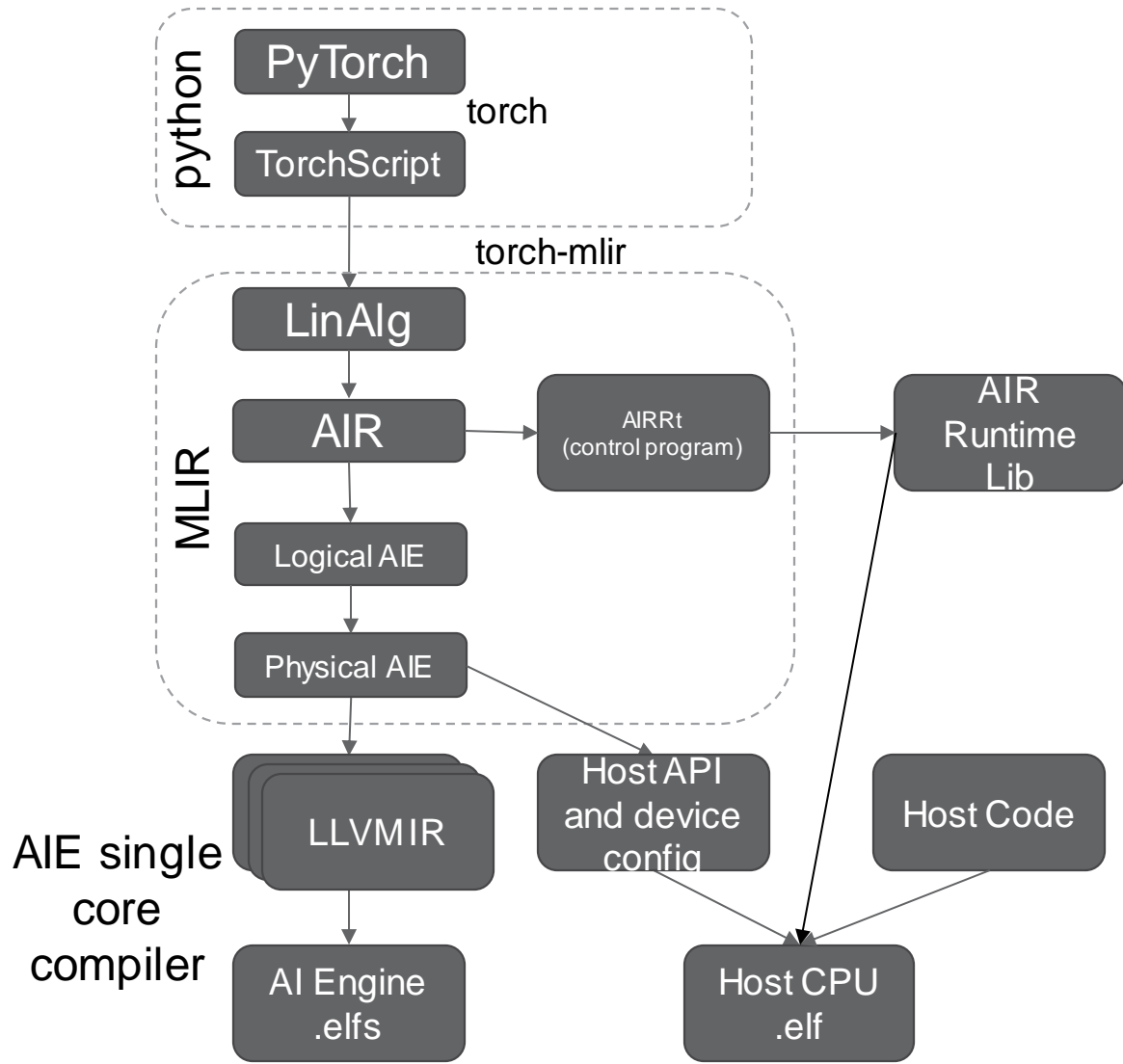


# Machine Learning in MLIR



See also: <https://iree-org.github.io/iree/>

# End-to-End Research Flow



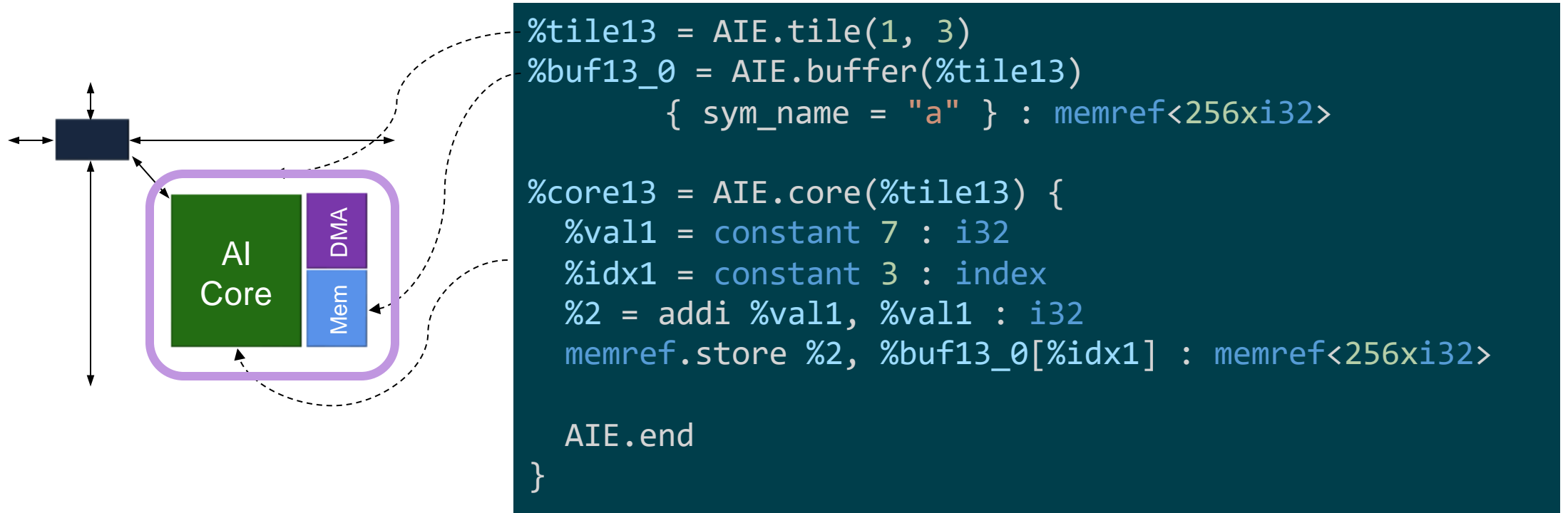
Most of this is already open source, except for the single-core compiler

This tutorial (mostly)



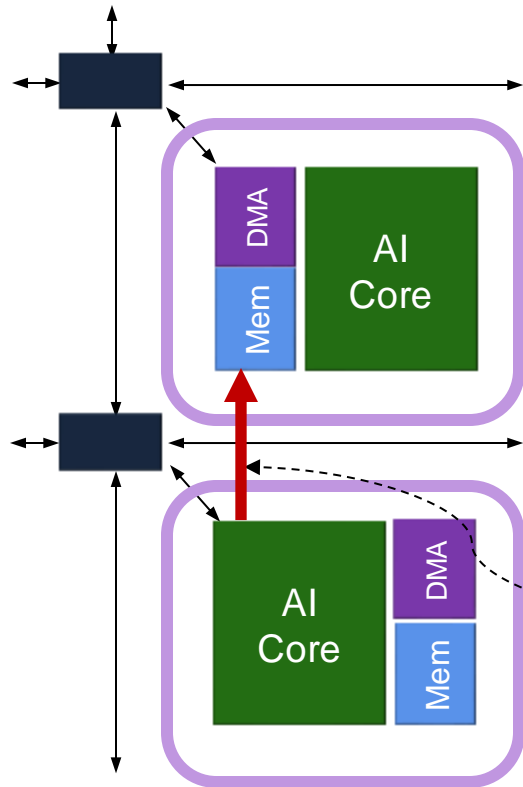
# **Physical MLIR-AIE**

# Running Code on a Core



- Memory Interface
- Stream Interface

# Moving Buffers



➔ Memory Interface  
➔ Stream Interface

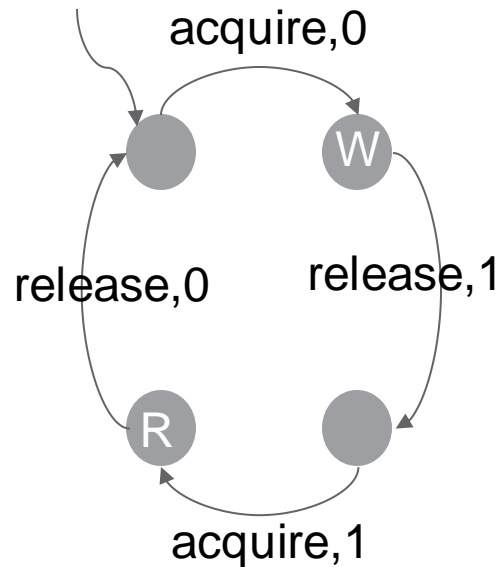
```
%tile14 = AIE.tile(1, 4)
%buf = AIE.buffer(%tile14)
      { sym_name = "a" } : memref<256xi32>

%tile13 = AIE.tile(1, 3)
%core13 = AIE.core(%tile13) {
  %val1 = constant 7 : i32
  %idx1 = constant 3 : index
  %2 = addi %val1, %val1 : i32

  memref.store %2, %buf[%idx1] : memref<256xi32>

  AIE.end
}
```

# Synchronization with Locks



```
%tile14 = AIE.tile(1, 4)
%lock = AIE.lock(%tile14) { sym_name = "a_lock" }
%buf = AIE.buffer(%tile14)
      { sym_name = "a" } : memref<512xi32>

%tile13 = AIE.tile(1, 3)
%core13 = AIE.core(%tile13) {
  AIE.useLock(%lock, "Acquire", 0)
  memref.store %2, %buf[%idx1] : memref<512xi32>
  AIE.useLock(%lock, "Release", 1)
}

%core14 = AIE.core(%tile14) {
  AIE.useLock(%lock, "Acquire", 1)
  %data = memref.load %buf[%idx1] : memref<512xi32>
  AIE.useLock(%lock, "Release", 0)
}
```

# Tutorial 1 Focus

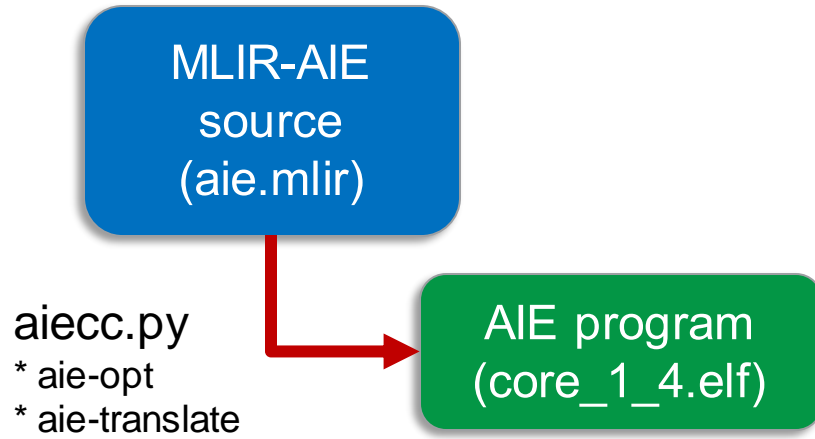
- Tutorial 1 – Modules, tile, buffer, core, lock
  - aie.mlir
  - test.cpp
  - Makefile
  - README.md
  - answers/

<https://github.com/Xilinx/mlir-aie/tree/main/tutorials>

# Host Code, Simulation and Performance Counters

- Welcome & Logistics
- Introduction to AIEngines
- Physical MLIR-AIE
- Host Code, Simulation, Performance Counters
- Logical MLIR-AIE: Communication
- Advanced Topics and Next Steps
- Wrap Up

# Lay of the Land (Tutorial – 1)



Pre-built libs

User source

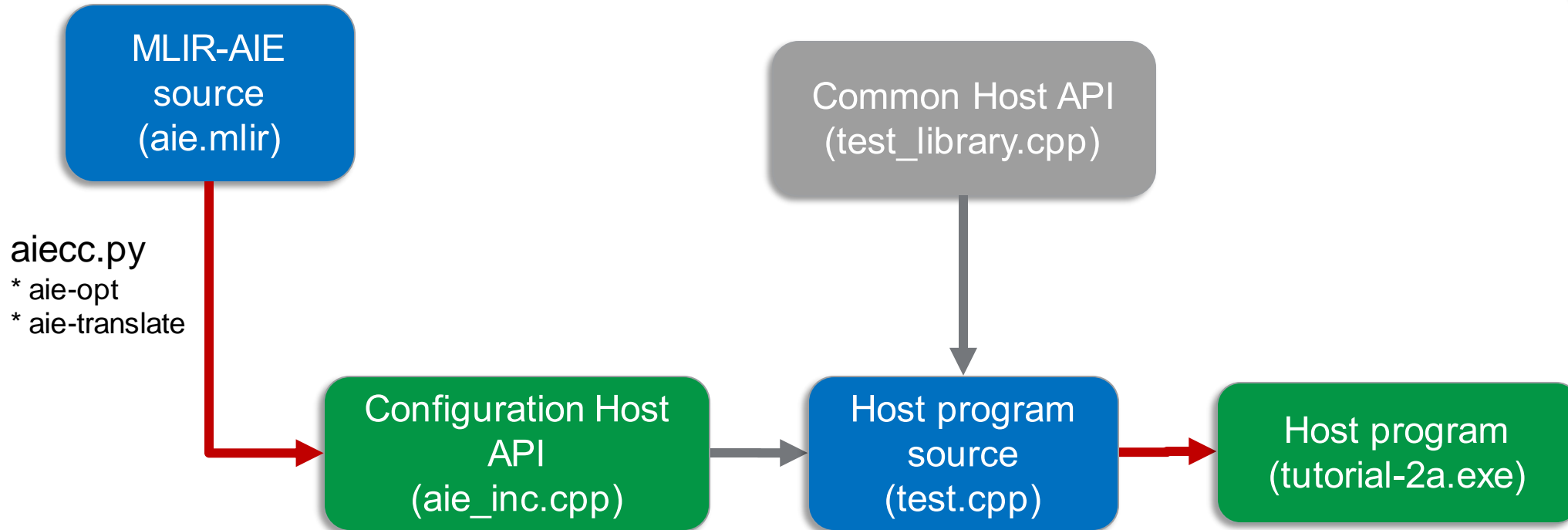
Generated files

# Lay of the Land (Tutorial 2a – Host Code Configuration)

Pre-built libs

User source

Generated files



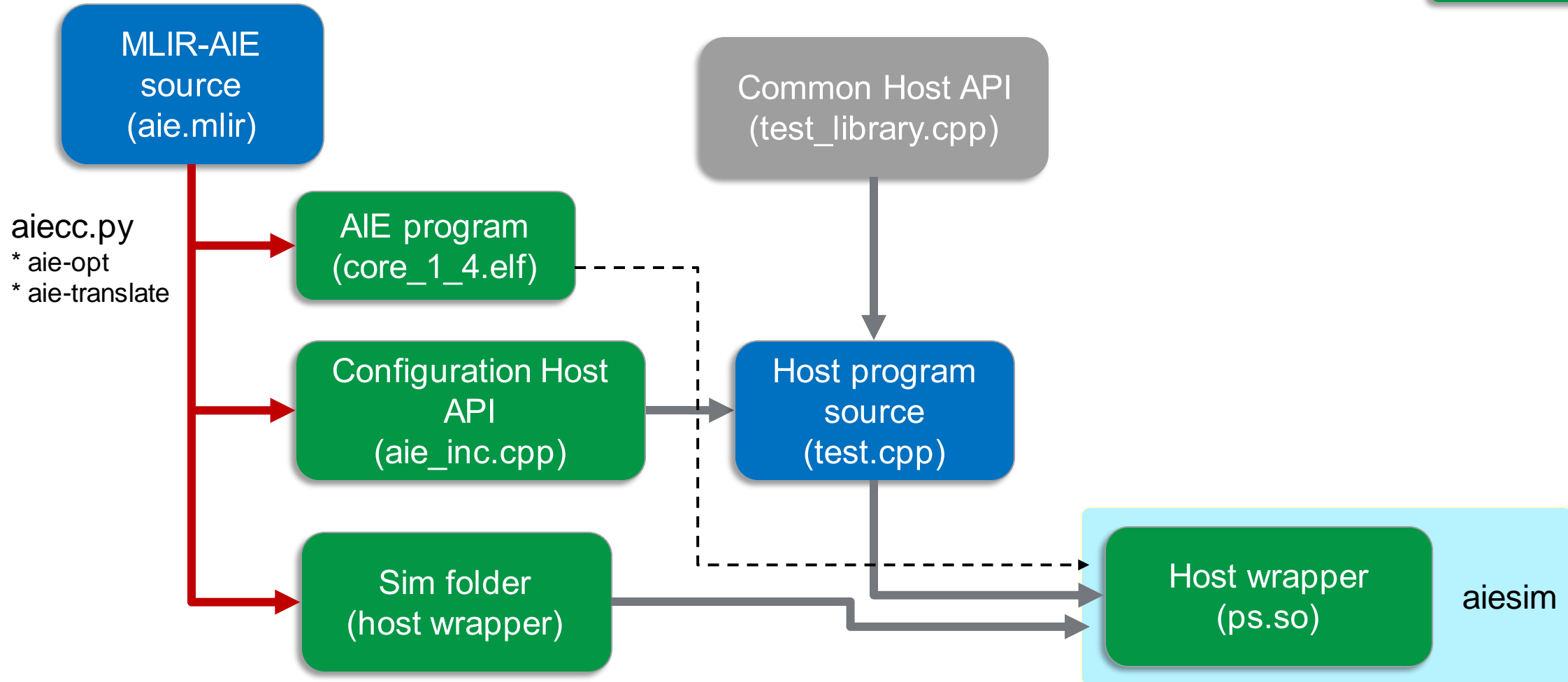


Pre-built libs

User source

Generated files

# Lay of the Land (Tutorial 2b – Simulation)

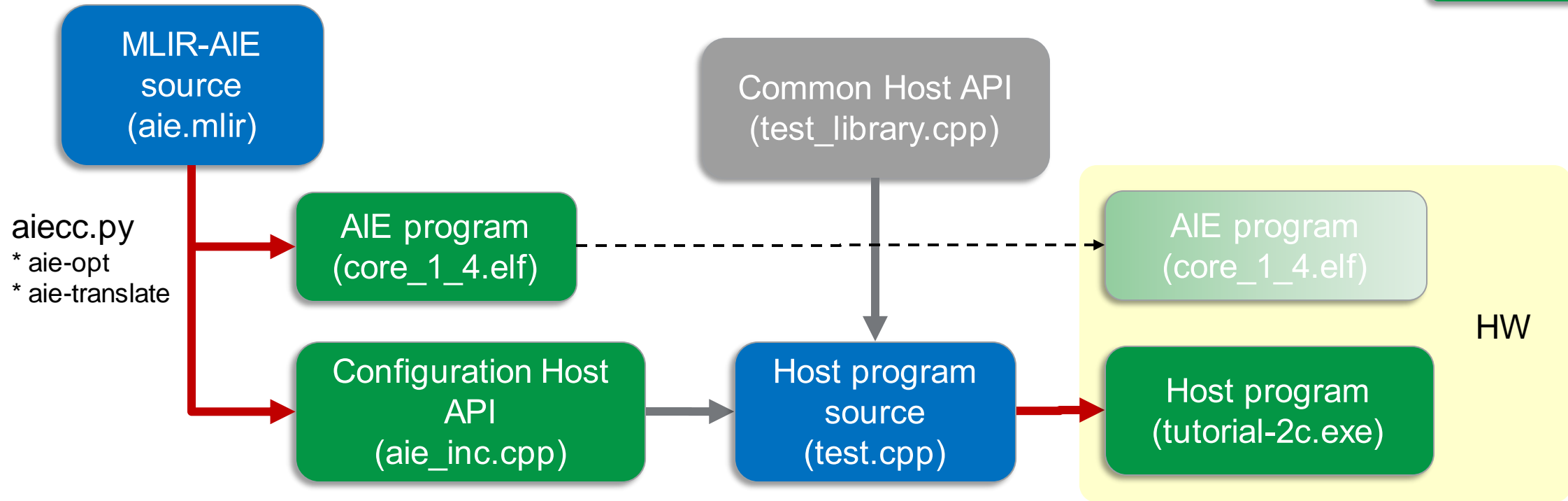


# Lay of the Land (Tutorial 2c – Hardware & Performance)

Pre-built libs

User source

Generated files

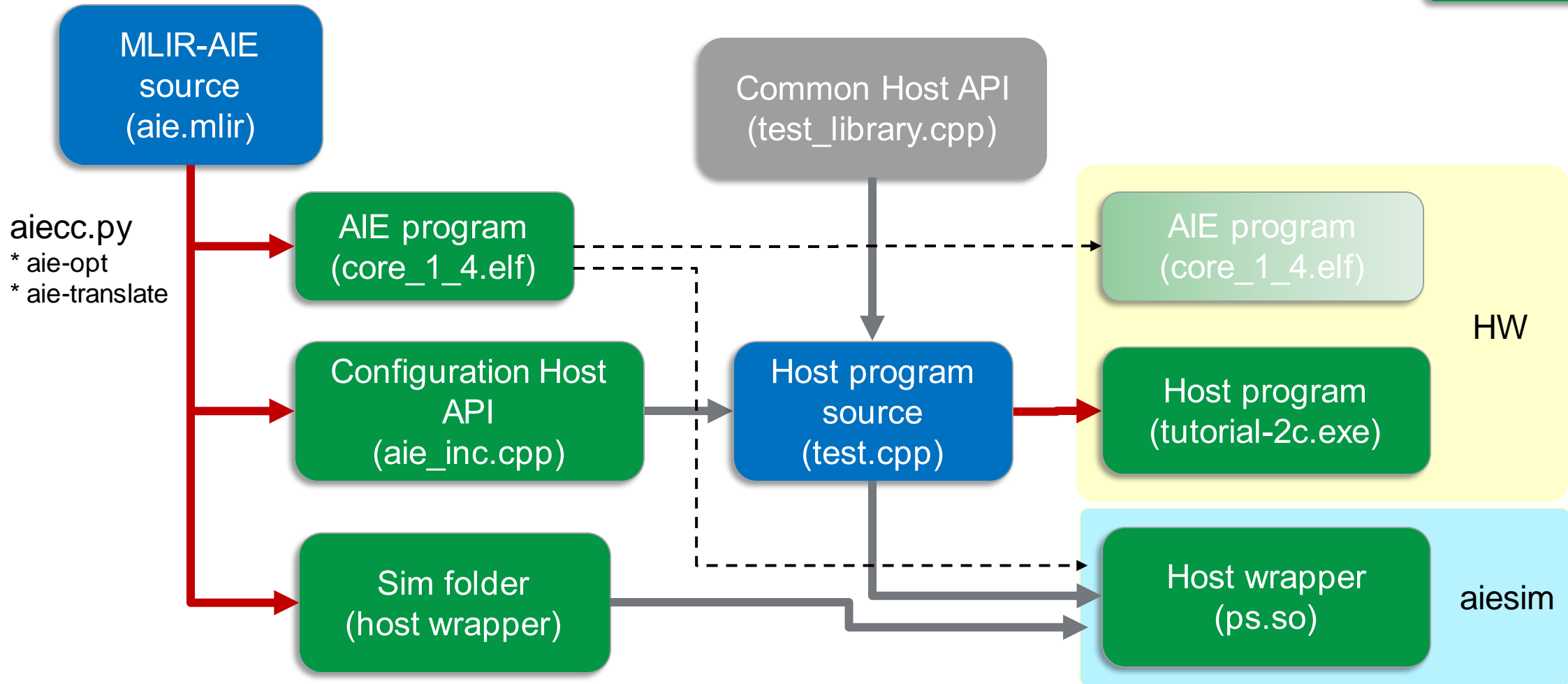


# Lay of the Land (Summary)

Pre-built libs

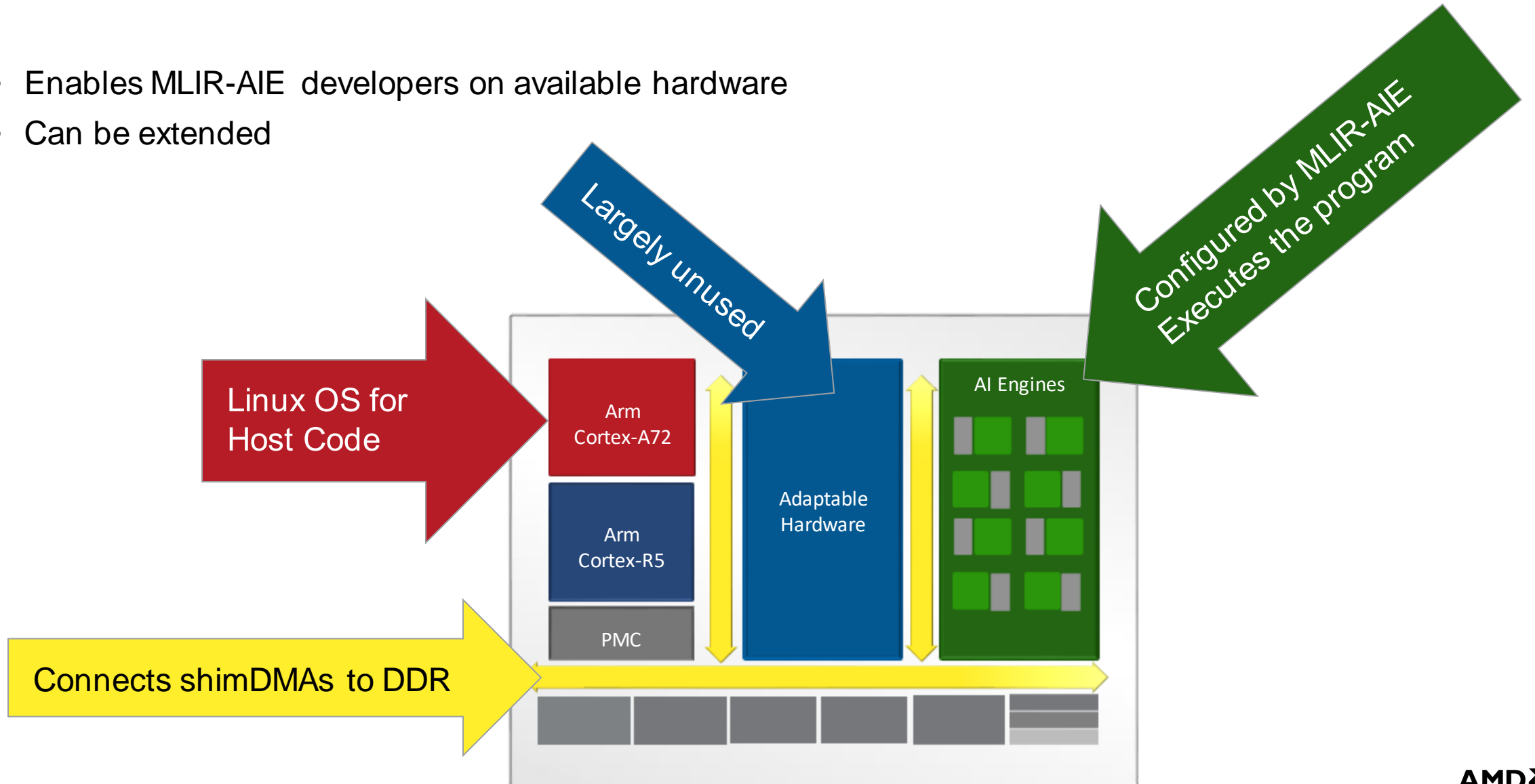
User source

Generated files



# MLIR-AIE VCK190 Platform Infrastructure

- Enables MLIR-AIE developers on available hardware
- Can be extended



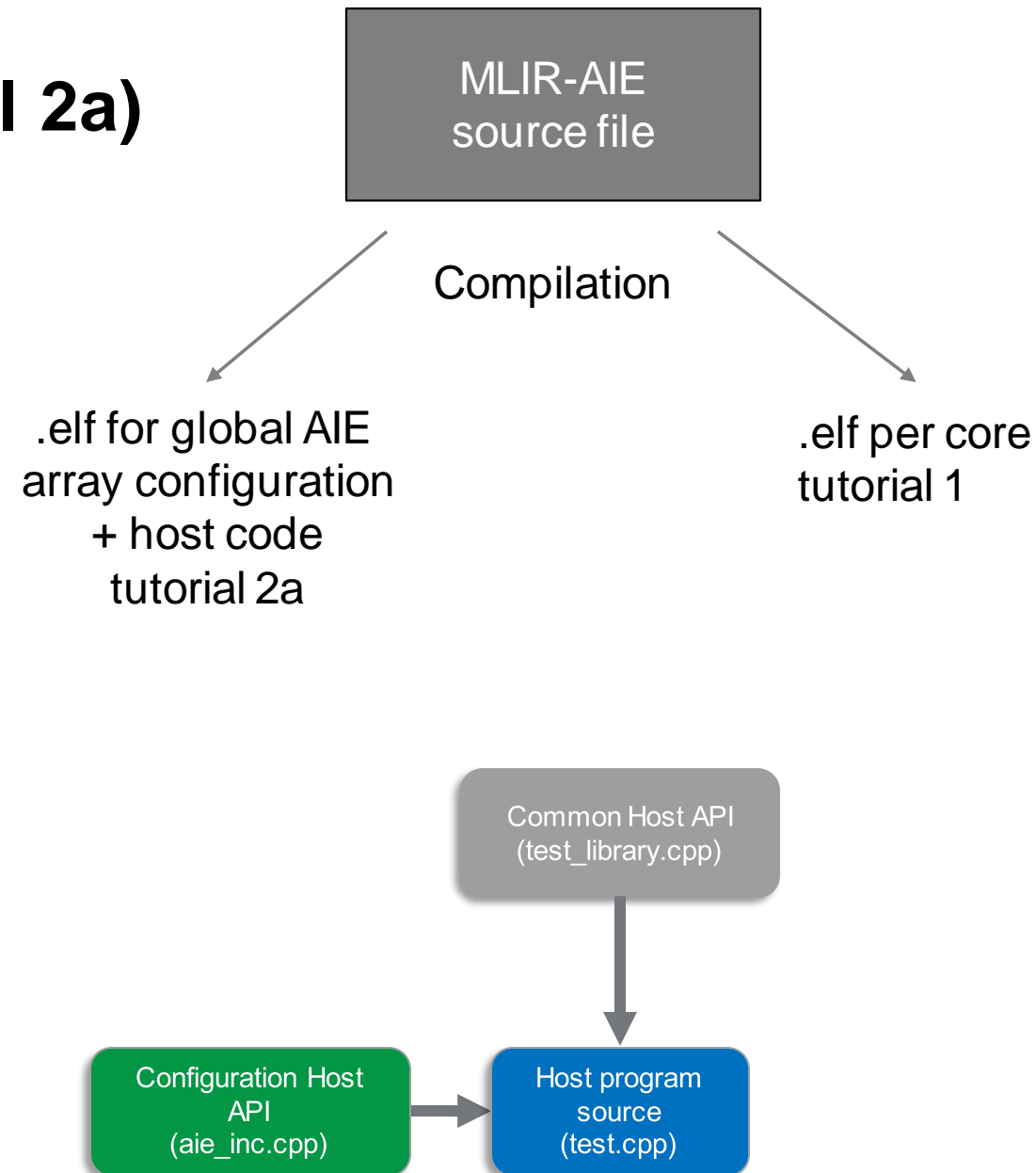
# Host Code Configuration API (Tutorial 2a)

- AI Engine configuration and initialization – 2 ways
  1. Control processor running host code (e.g. ARM)
  2. Bootup (CDO files)
- Tutorial examples uses (1) with host code API
  - Host program (test.cpp) references host code API
  - Configuration host API (aie\_inc.cpp)
    - Configure/ initialize AIE array design

```
aie_libxaie_ctx_t *_xaie = mlir_aie_init_libxaie();
mlir_aie_init_device(_xaie);
mlir_aie_configure_cores(_xaie);
mlir_aie_configure_switchboxes(_xaie);
mlir_aie_configure_dmas(_xaie);
mlir_aie_initialize_locks(_xaie);

mlir_aie_clear_tile_memory(_xaie, 1, 4); // clear local data memory for tile(1,4)
```

- Common host code API (test\_library.cpp)
  - Read/write data memory
  - Acquire/release locks
  - Check/compare values



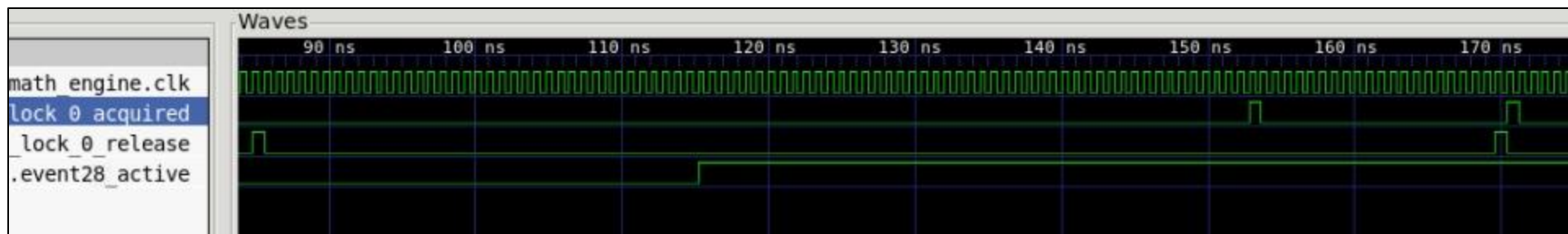
# Simulation (Tutorial 2b)

- We generate a `sim` folder every time `aiecc.py` is run
  - Contains stubs to work with command line AI Engine simulator (`aiesimulator`)
- `aiesimulator`
  - Cycle-accurate simulator per AI Engine core
  - Transaction level System-C simulator for array communication
  - Wrapper for host code programs
    - Models transfer between host side memory (DDR) and AI Engine Array data movers (shimDMA through NOC)
  - Useful for testing smaller designs to verify functionality
  - Larger designs can be run directly in hardware
- `gtkwave`
  - `aiesimulator` generates `foo.vcd` file which user can use `gtkwave` to view simulation waveforms

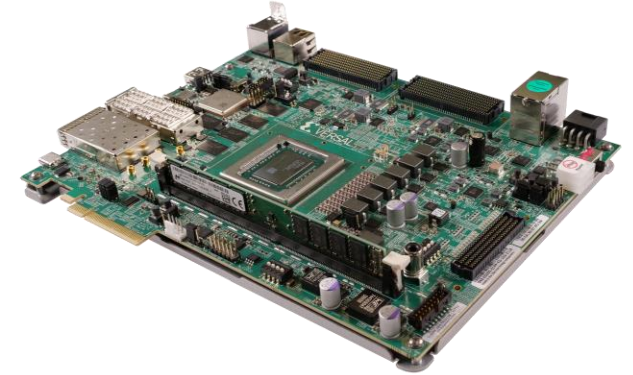
make -C sim

(1) Recompile  
test.cpp+aie\_inc.cpp  
test\_library.cpp  
host wrapper

(2) Run aiesimulator



# Running on Hardware (Tutorial 2c)

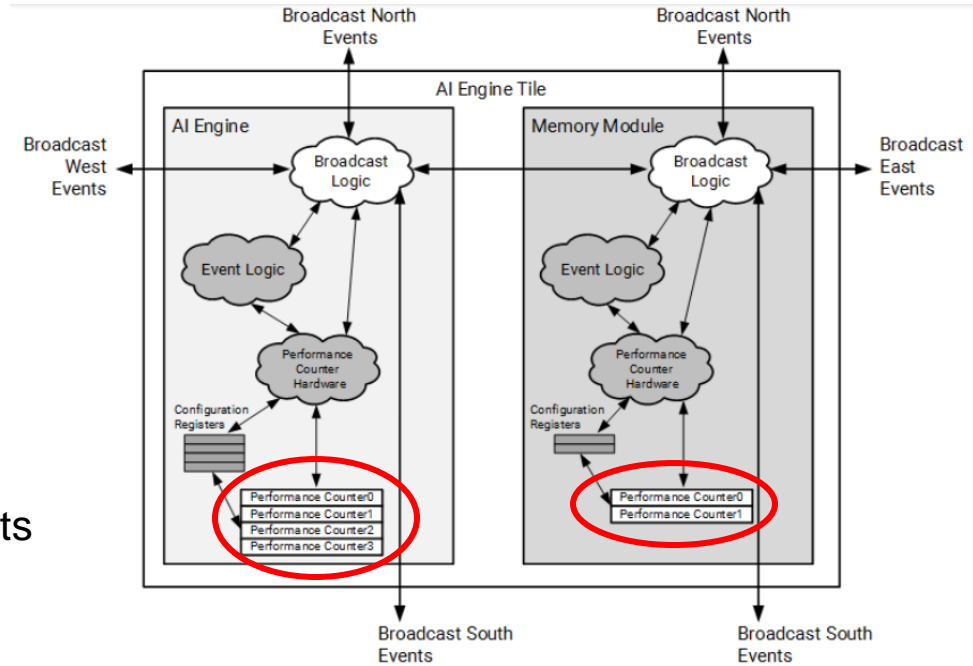


- Steps for running on vck190
  1. Copy .elf and .exe files to board (e.g., `scp *.elf xilinx@<board ip>/home/xilinx/.`)
  2. `ssh xilinx@<board ip>`, pw: xilinx
  3. Run executable with sudo permissions (`sudo ./tutorial-2c.exe`)
- Time to verify designs on hardware is fast (no need to run synthesis, place & route)
  - Leverage a Vivado built platform design (PYNQ enabled) which configures the NOC to support all shim DMA connections between the AI Engine array and DDR memory (vck190)
    - Linux OS on Arm core for host code
  - Especially fast for very large designs (as compared to simulation)
- Simulation vs. Hardware
  - Timing relationship between host code and AIE programs changes between simulation vs. hardware
    - Simulation – Fast host code, “Slower” AIE programs (cycle accurate simulation)
    - Hardware – Fast AIE programs (1 GHz), “Slower” host code (100s of AIE cycles per host code API function call)
  - Examples
    - `usleep` can work for waiting for AIE programs to complete but does not work in simulation
    - Instead, use locks to synchronize between host and AI Engine simulation

# Performance Counters (Tutorial 2c)



- Useful for counting cycles between events
  - Core (4 counters), Memory (2 counters)
  - Start, Stop, Reset – works like a stop watch
- Example events
  - <https://docs.xilinx.com/r/en-US/am009-versal-ai-engine/AI-Engine-Events>
  - Program addresses
  - Lock acquire, release (by ID but not value)
  - Broadcast (by ID)
- Tutorial-2c introduces performance counters with program address event triggers
- Tutorial-3 extends events triggers to locks
- Tutorial-4 extends event triggers to broadcast events



X22904-022119



# Tutorial 2 Focus

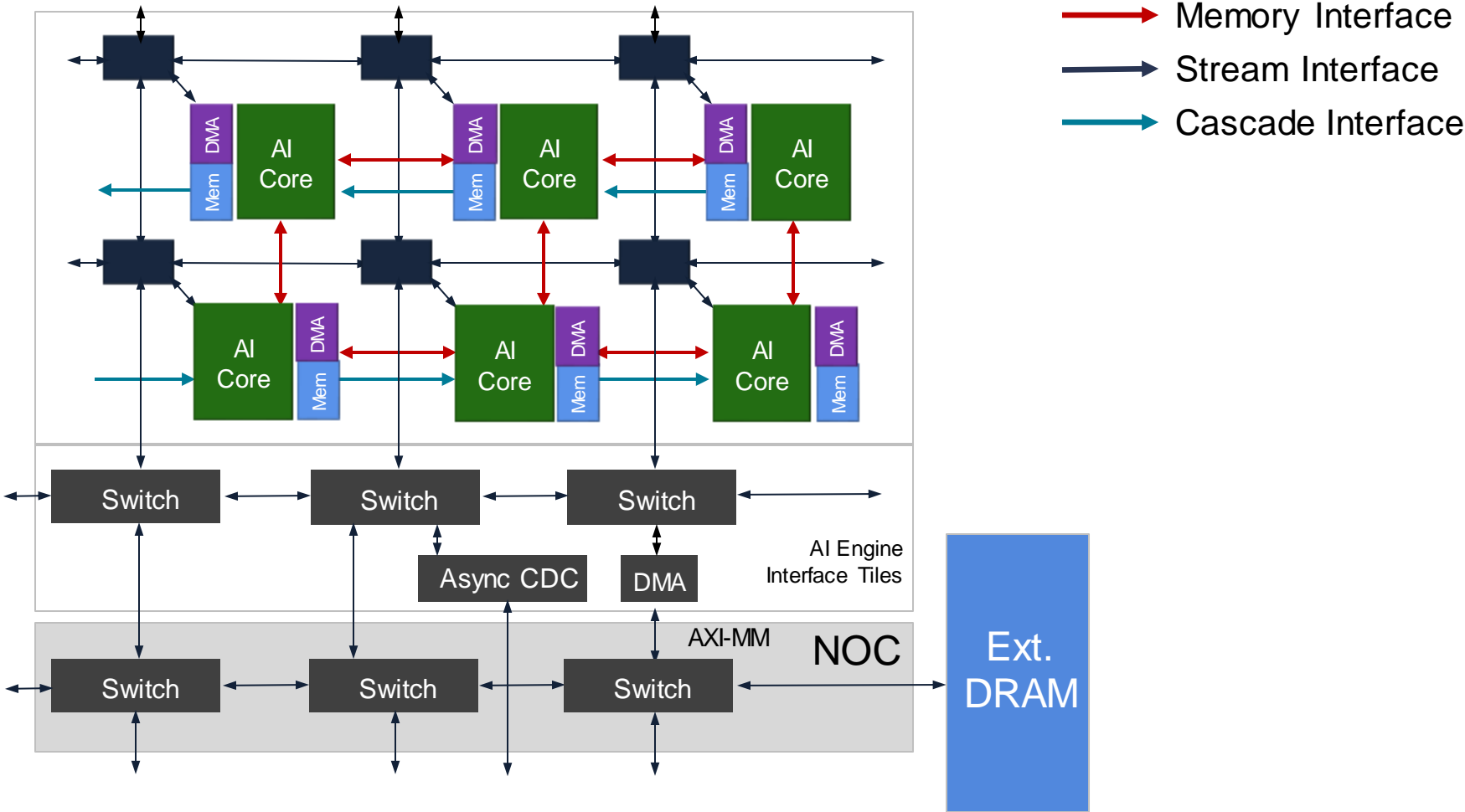
- Tutorial 2 – Host code configuration, simulation, hardware performance
  - Tutorial-2a - Host code configuration
    - aie.mlir
    - test.cpp
    - Makefile
    - README.md
  - Tutorial-2b - Simulation
    - aie.mlir
    - test.cpp
    - Makefile
    - README.md
  - Tutorial-2c - Running on hardware and measuring performance
    - aie.mlir
    - test.cpp
    - Makefile
    - README.md

<https://github.com/Xilinx/mlir-aie/tree/main/tutorials>

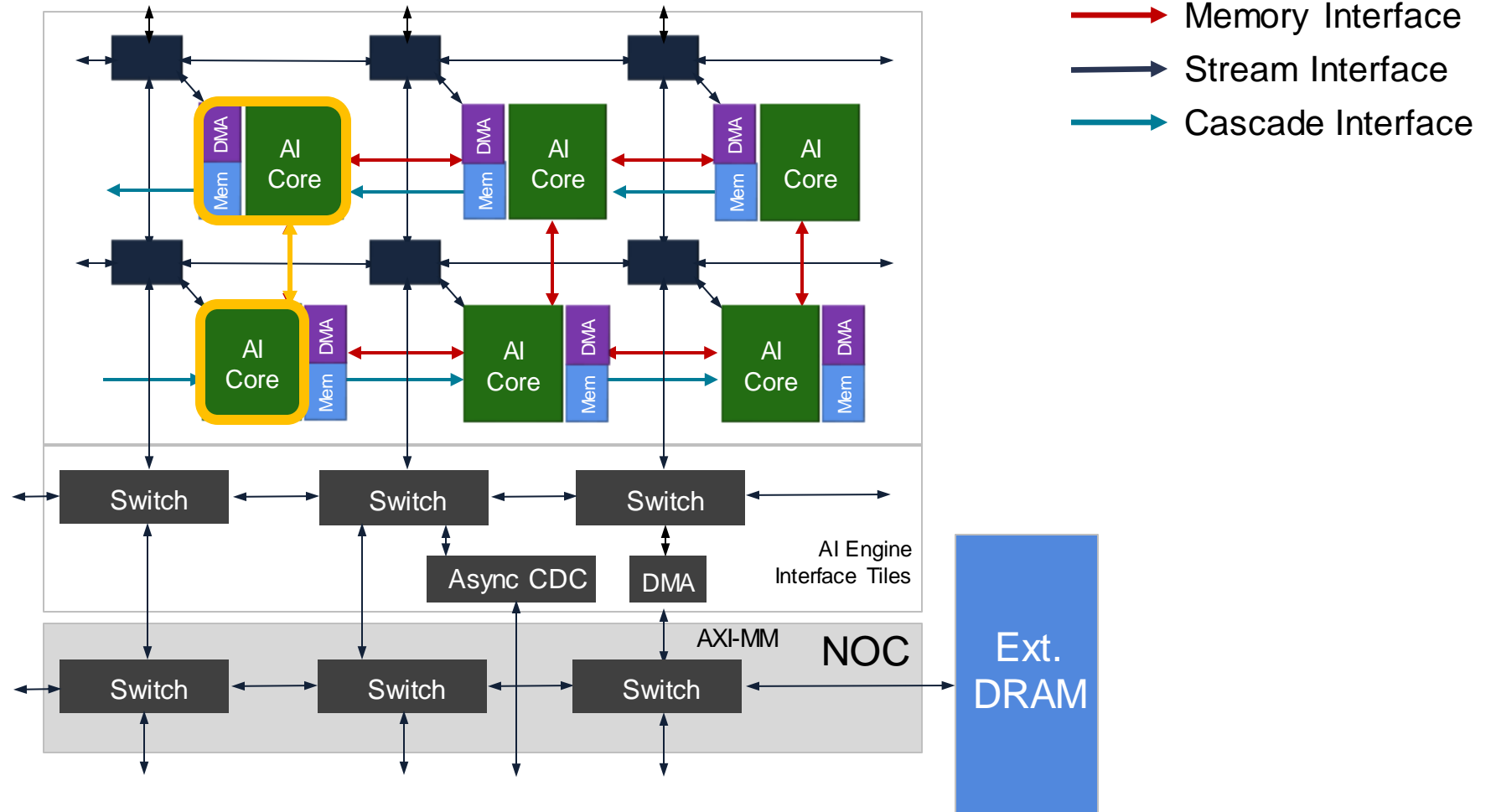
# Logical MLIR-AIE: Communication

- Welcome & Logistics
- Introduction to AIEngines
- Physical MLIR-AIE
- Host Code, Simulation, Performance Counters
- Logical MLIR-AIE: Communication
- Advanced Topics and Next Steps
- Wrap Up

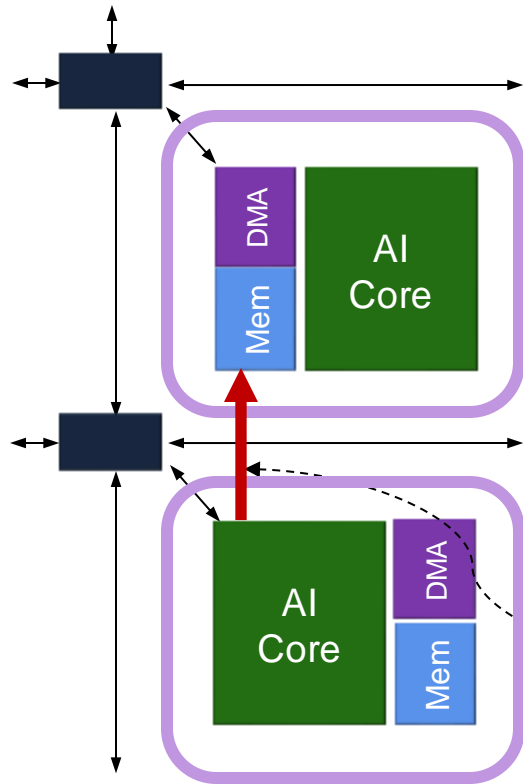
# MLIR-AIE Communication Achieved by Configuring Different Components



# Tutorial 3: MLIR-AIE Engine-to-Engine Communication Through Shared Memory



# Buffers in Shared Memory Offer Straightforward Communication



➔ Memory Interface  
➔ Stream Interface

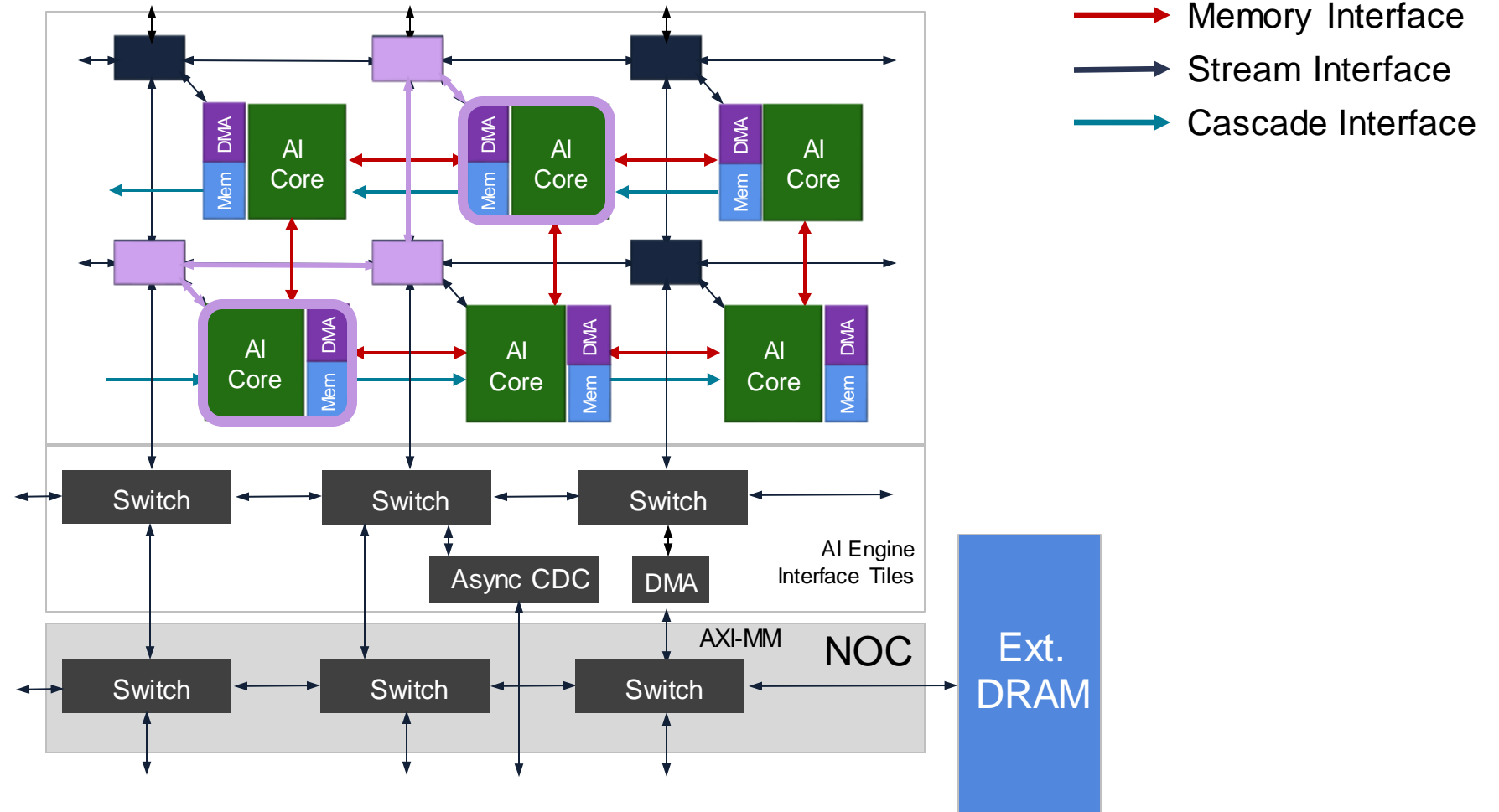
```
%tile14 = AIE.tile(1, 4)
%buf = AIE.buffer(%tile14)
      { sym_name = "a" } : memref<256xi32>

%tile13 = AIE.tile(1, 3)
%core13 = AIE.core(%tile13) {
  %val1 = constant 7 : i32
  %idx1 = constant 3 : index
  %2 = addi %val1, %val1 : i32

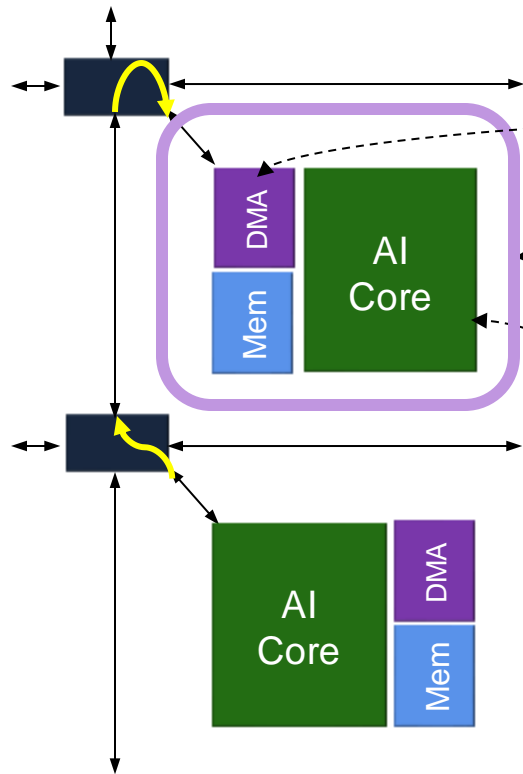
  memref.store %2, %buf[%idx1] : memref<256xi32>

  AIE.end
}
```

# Tutorial 4: MLIR-AIE Engine-to-Engine Communication Through Tile DMAs and Stream Switches



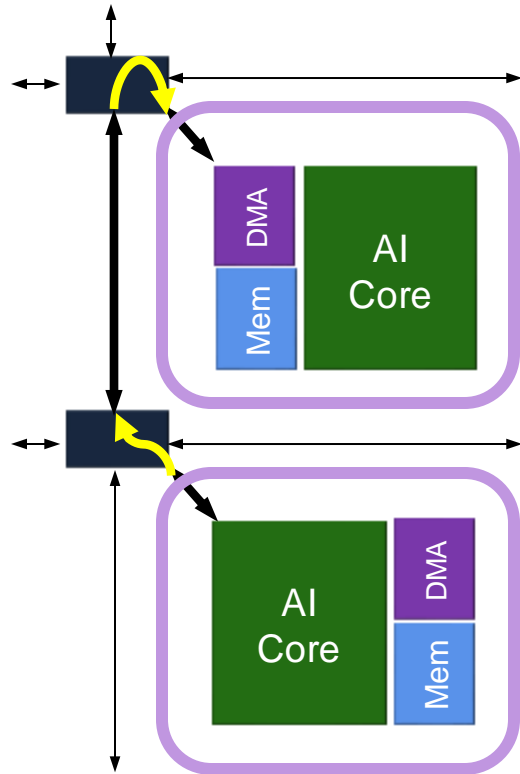
# Configuring Tile DMAs is Tedious



→ Memory Interface  
→ Stream Interface

```
%tile14 = AIE.tile(1, 4)
%buf14 = AIE.buffer(%tile14) { sym_name = "a14" } : memref<256xi32>
%lock14_6 = AIE.lock(%tile14, 6) { sym_name = "lock_a14_6" }
%mem14 = AIE.mem(%tile14) {
  AIE.dmaStart("MM2S", 0, ^bd0, ^end)
^bd0:
  AIE.useLock(%lock14_6, "Acquire", 1)
  AIE.dmaBd(<%buf14 : memref<256xi32>, 0, 256>, 0)
  AIE.useLock(%lock14_6, "Release", 0)
  AIE.nextBd ^end
^end:
  AIE.end
}
%core14 = AIE.core(%tile14) {
  AIE.useLock(%lock14_6, "Acquire", 0)
  %val = arith.constant 14 : i32
  %idx = arith.constant 3 : index
  memref.store %val, %buf14[%idx] : memref<256xi32>
  AIE.useLock(%lock14_6, "Release", 1)
  AIE.end
}
```

# Configuring Stream Switches is Tedious



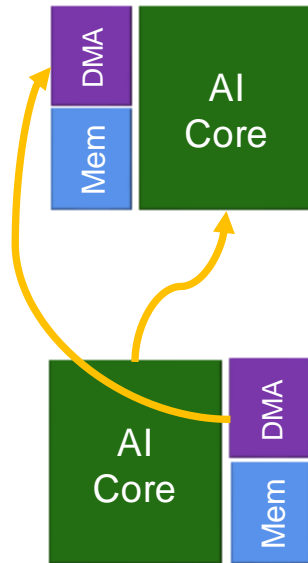
→ Memory Interface  
→ Stream Interface

```
%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>
}
```



# The Flow Higher Abstraction Absorbs Some of the Configuration...



→ Memory Interface  
→ Flows

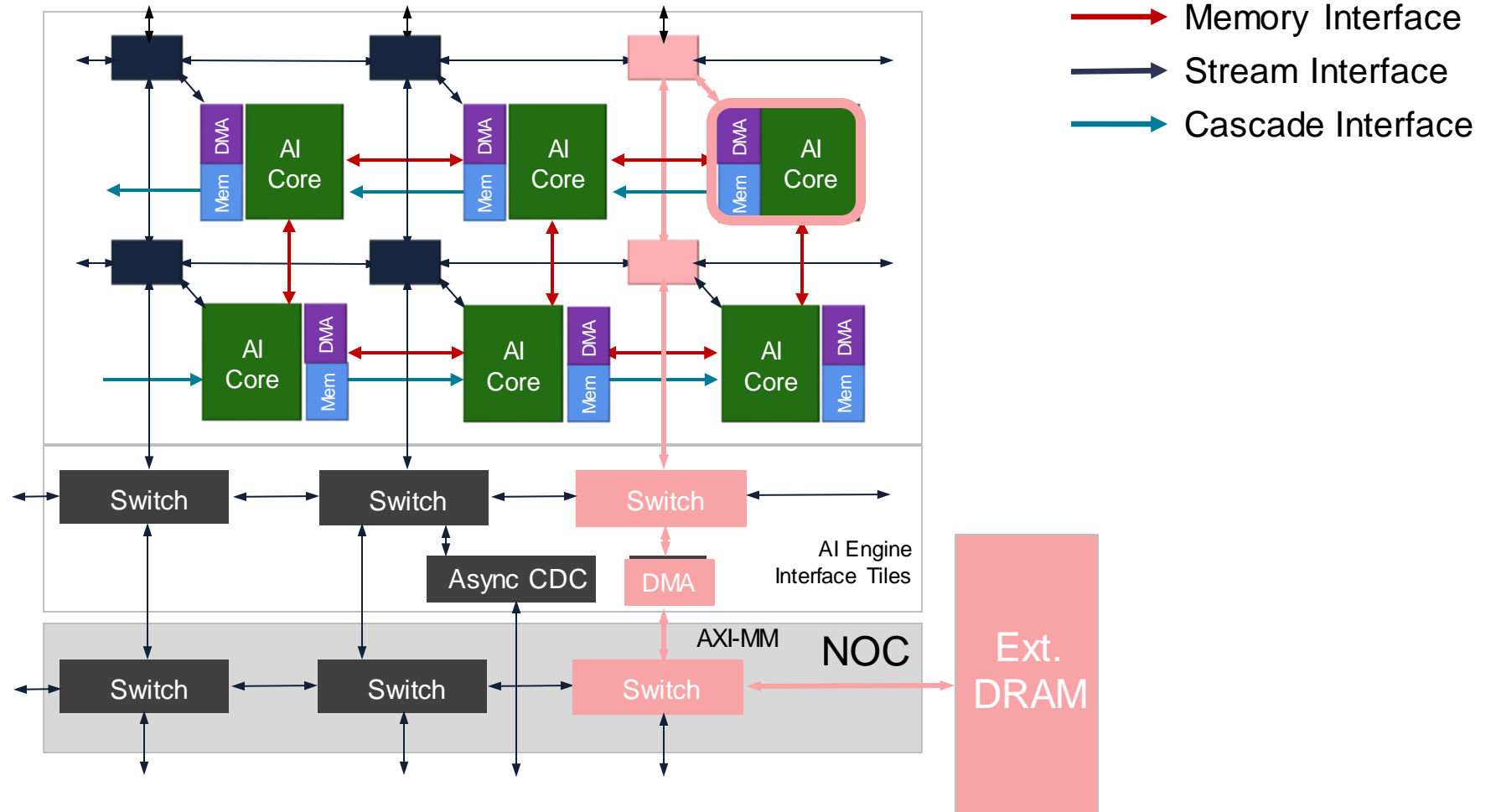
```
AIE.flow(%tile14, "DMA" : 0, %tile13, "DMA" : 0)  
AIE.flow(%tile14, "Core" : 0, %tile13, "Core" : 0)
```



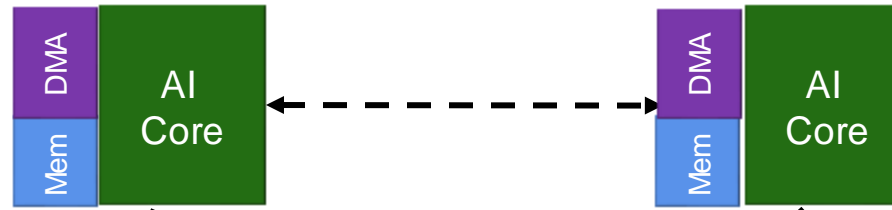
aie-opt --aie-create-flows

```
%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>  
}
```

# Tutorial 5: MLIR-AIE Engine-to-External Memory Communication Through Tile DMAs, Stream Switches and NOC



# The Object FIFO Abstracts Both the Data Allocation and the Data Movement

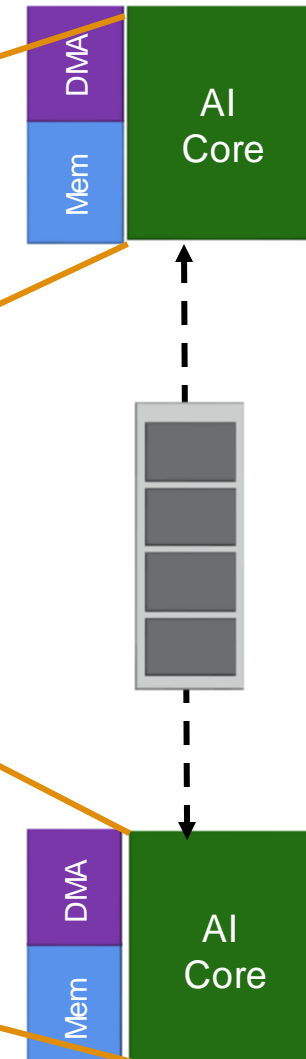


```
%objFifo = AIE.objectFifo.createObjectFifo(%tile14, {%tile24}, 2) : !AIE.objectFifo<memref<256xi32>>
```

```
%buff24_0 = AIE.buffer(%tile24) {sym_name = "of_0_buff_0"} : memref<256xi32>  
%lock24_0 = AIE.lock(%tile24, 0) {sym_name = "of_0_lock_0"}  
%buff24_1 = AIE.buffer(%tile24) {sym_name = "of_0_buff_1"} : memref<256xi32>  
%lock24_1 = AIE.lock(%tile24, 1) {sym_name = "of_0_lock_1"}
```

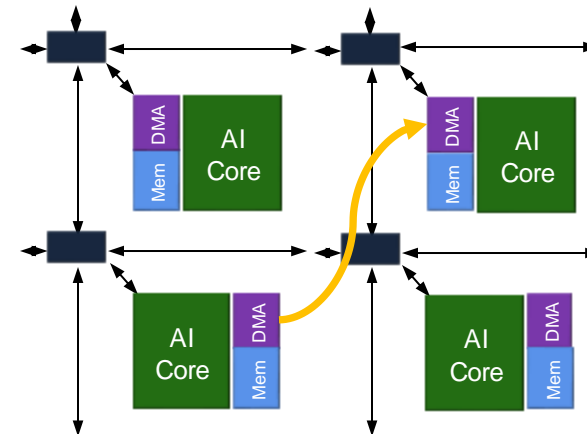
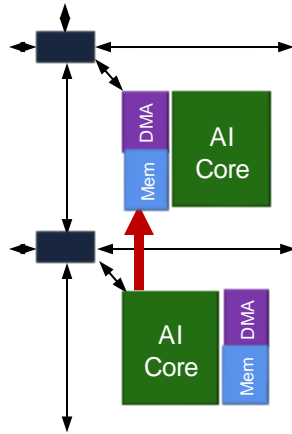
# Synchronized Accesses to the Object FIFO Ensure a Deadlock-Free Schedule

```
%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
}
```



# Object FIFO Lowers to Communication Components Based on the Endpoints

aie-opt --aie-objectFifo-stateful-transform



```
%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)
```

# Tutorials 3, 4, 5 Focus

- Tutorial 3 – Communication (Local Memory)

- aie.mlir
- test.cpp
- Makefile
- README.md
- answers/
- objectFifo\_ver/

- Tutorial 4 – Communication via objectFifo (tile DMA, logical routing)

- aie.mlir
- test.cpp
- Makefile
- README.md
- flow/
- switchbox/

- Tutorial 5 – Communication via objectFifo (shim DMA, external memory aka DDR)

- aie.mlir
- test.cpp
- Makefile
- README.md
- flow/

<https://github.com/Xilinx/mlir-aie/tree/main/tutorials>

# Advanced Topics and Next Steps

- Welcome & Logistics
- Introduction to AIEngines
- Physical MLIR-AIE
- Host Code, Simulation, Performance Counters
- Logical MLIR-AIE: Communication
- Advanced Topics and Next Steps
- Wrap Up

# Generating MLIR-AIE from Higher Level Tools

- Generate Textual MLIR-AIE ←
- Python Bindings for MLIR-AIE
- Higher Level Dialects (e.g., MLIR-AIR)

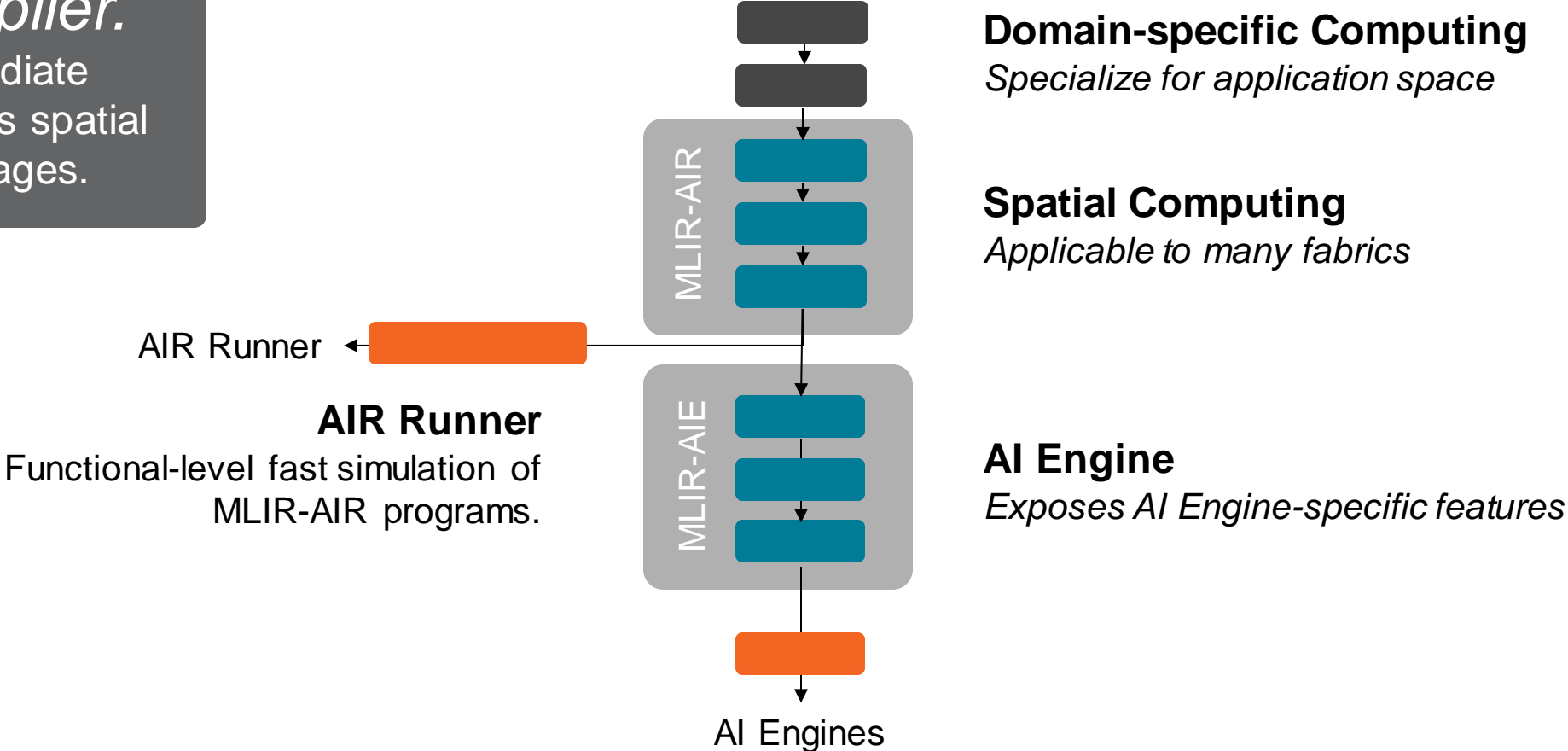
reference\_designs/horizontal\_diffusion

- README.md
- HDIFF\_single\_AIE\_objectFIFO\_ping\_pong
- HDIFF\_dual\_AIE\_objectFIFO\_ping\_pong
- HDIFF\_tri\_AIE\_objectFIFO\_ping\_pong
- HDIFF\_tri\_AIE\_objectFIFO\_ping\_pong\_scaled

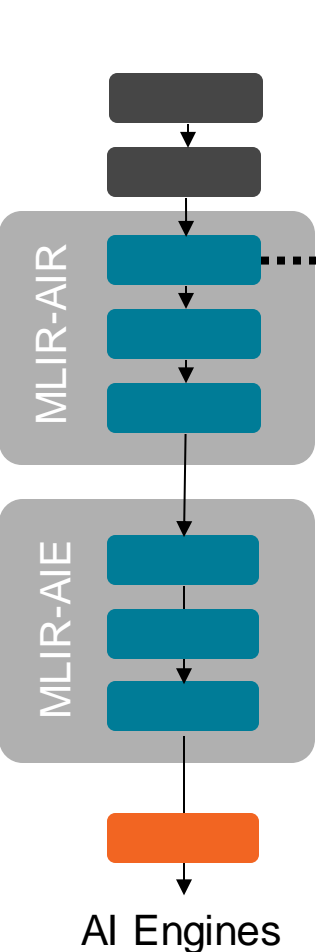


# Introducing MLIR-AIR

*AIR is a compiler.*  
A portable intermediate language that exposes spatial computing advantages.



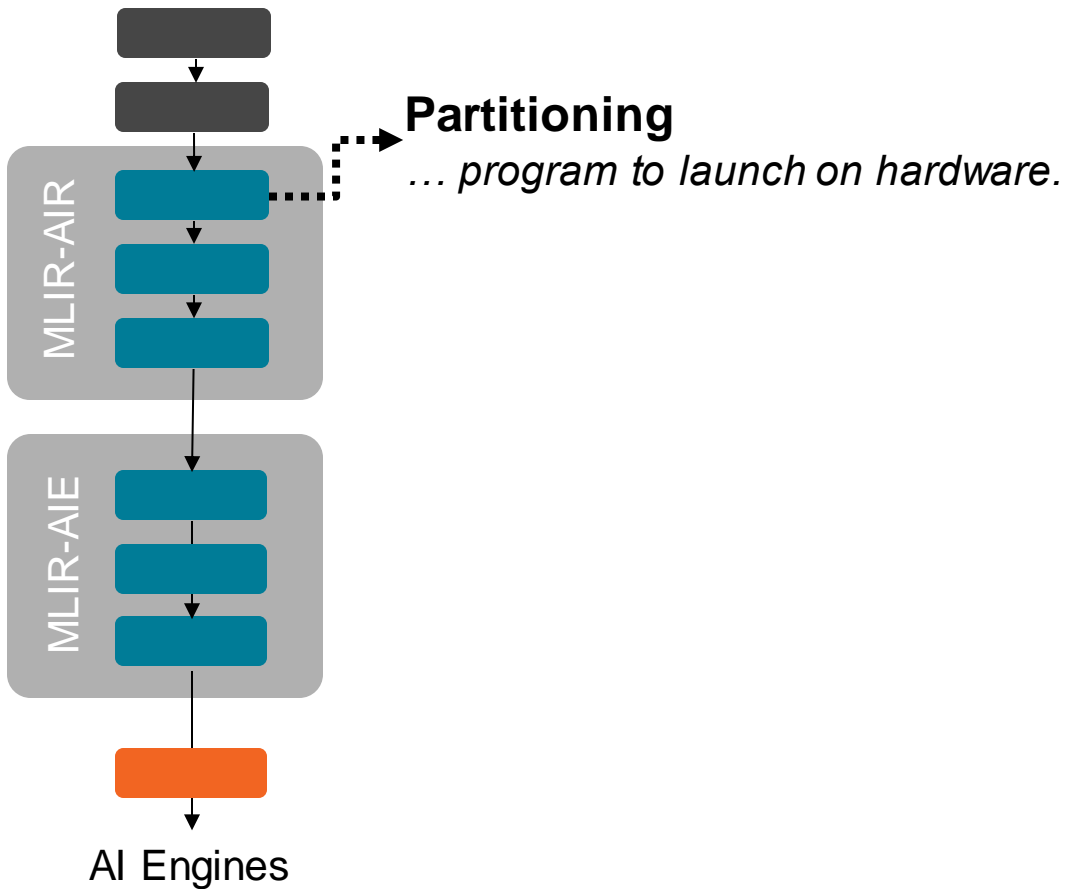
# Introducing MLIR-AIR



**Tiling**  
*... a program to be executed concurrently in space..*

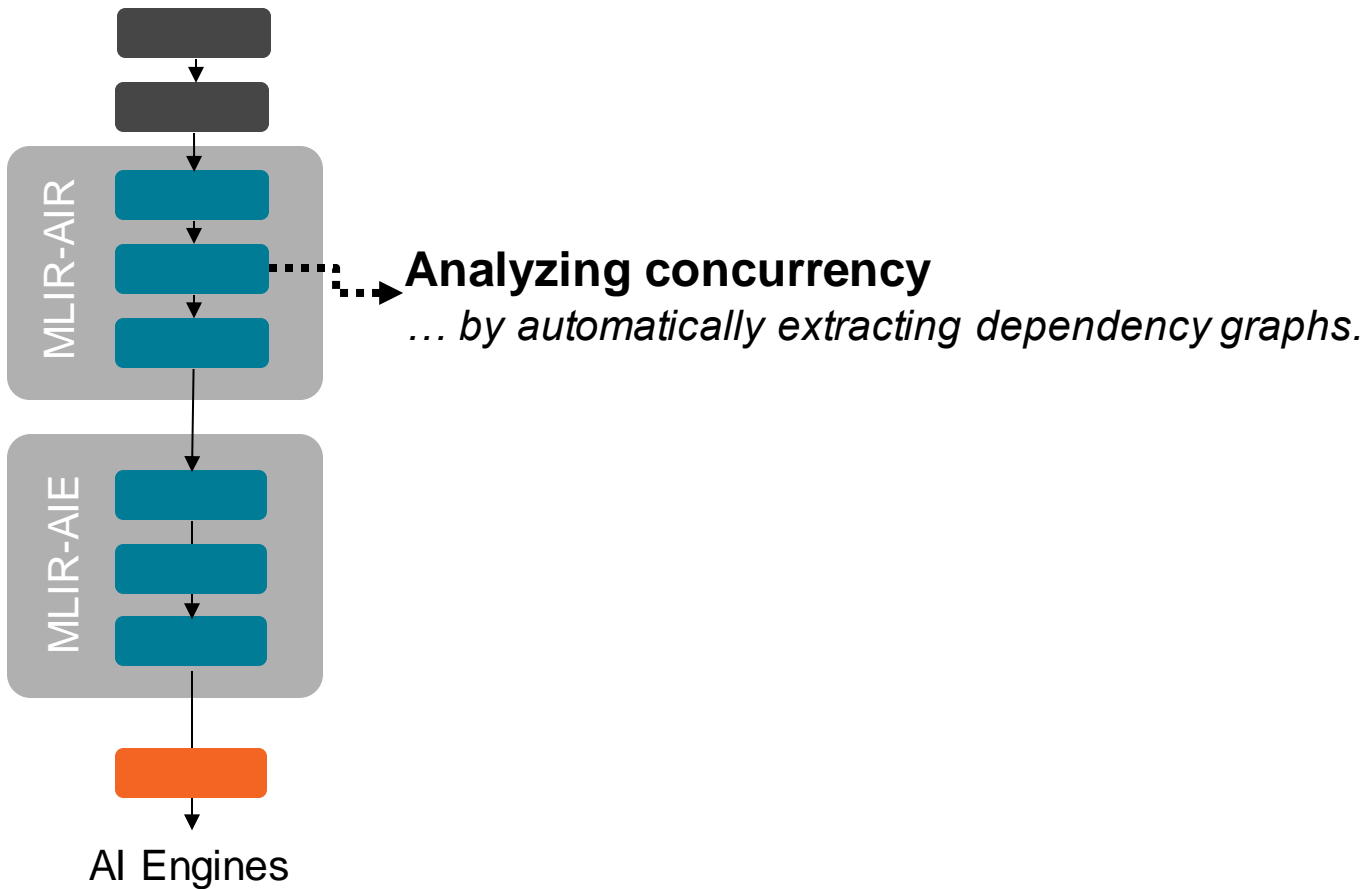
*Scheduling in space*

# Introducing MLIR-AIR



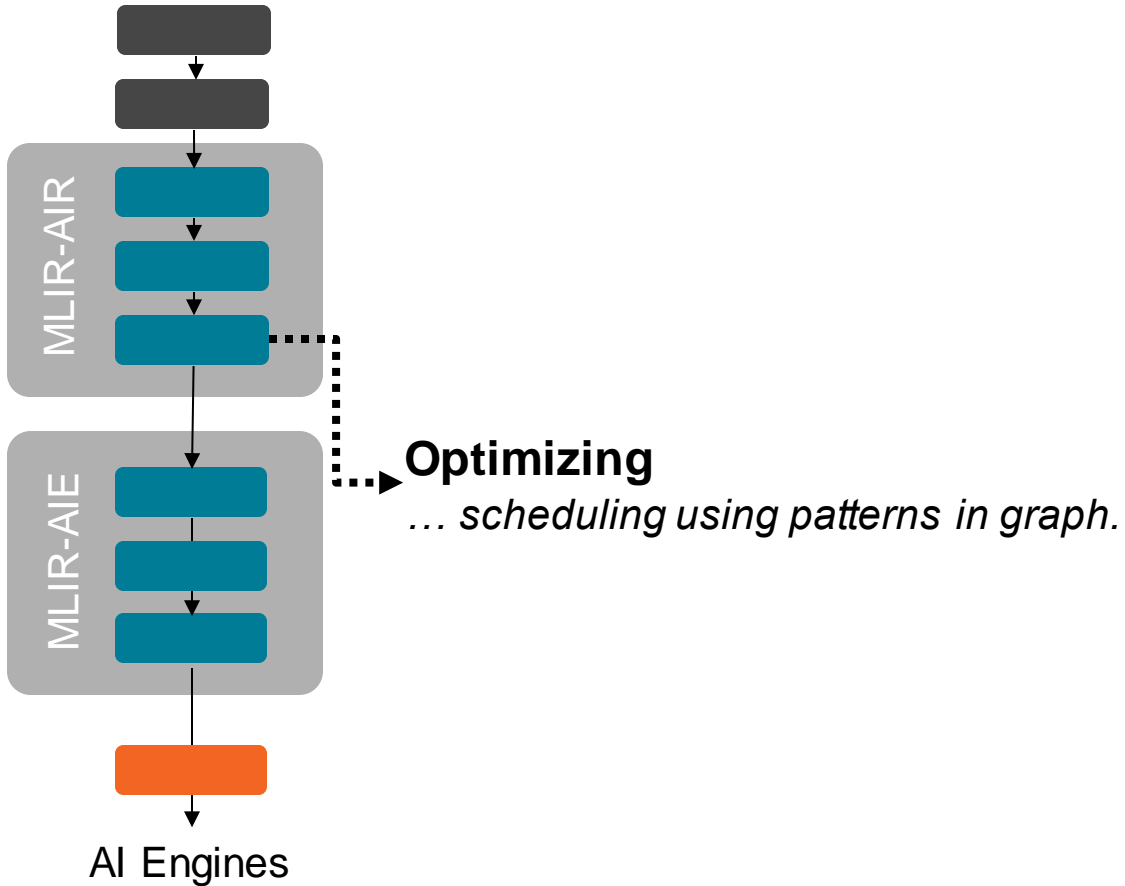
*Scheduling in time*

# Introducing MLIR-AIR



*Reasoning about performance*

# Introducing MLIR-AIR



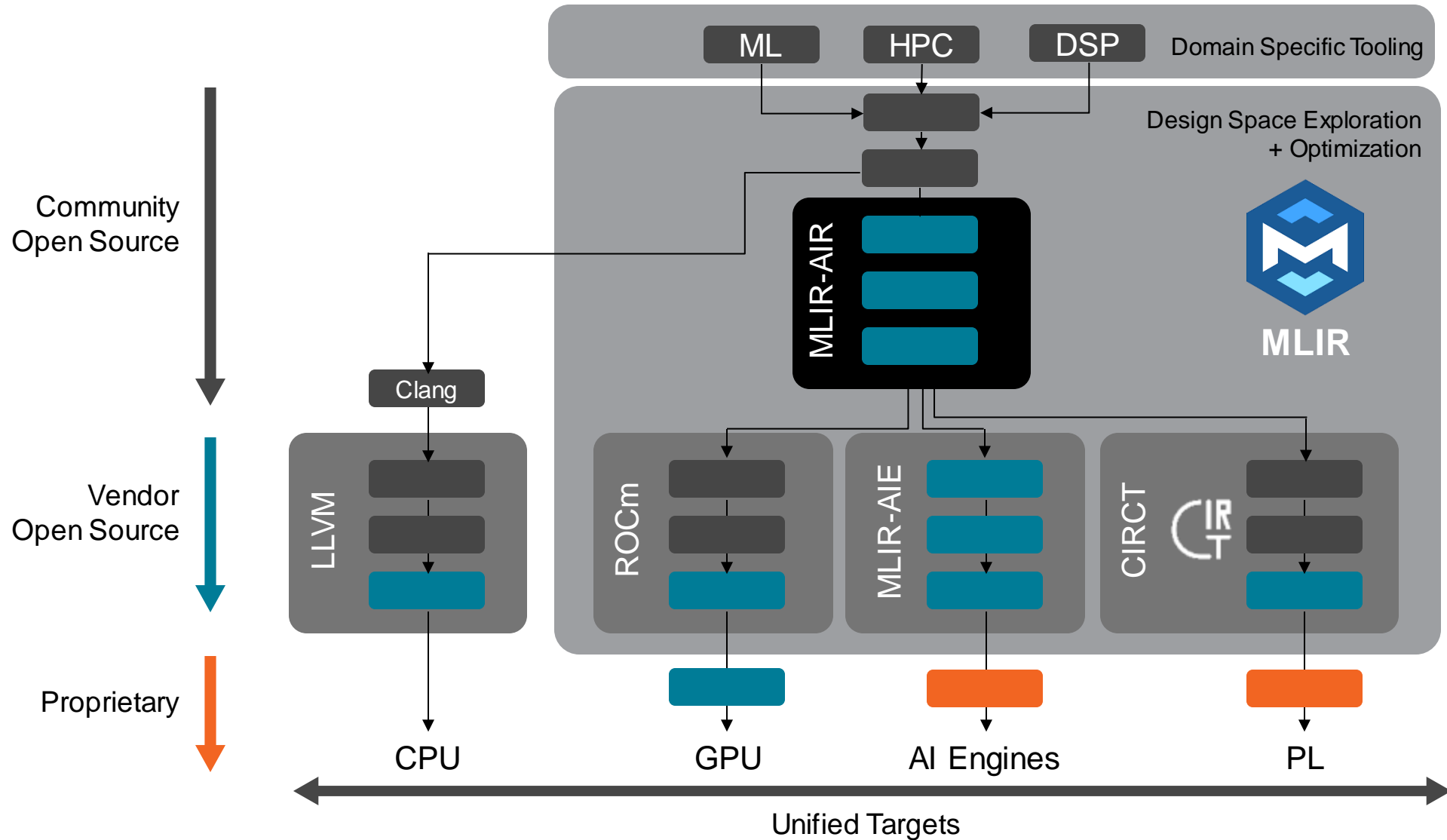
*Optimizing for  
performance*

# Tools Built around AIR dialect

*AIR is a template.*

A template for scalable ML platforms using multiple fabrics, including AIE and GPUs.

# Tools Built around AIR dialect



# Building on our MLIR tools

- Our tools are open sourced on GitHub
  - forks, PRs and issues are welcome!
- Stable code interface.
  - IR validation and unit testing infrastructure.
- Progressive MLIR lowering passes at per-operation granularity.
  - Sub-passes callable in isolation; open to per-pass customization.
  - Declarative MLIR rewrite patterns.
  - MLIR code reusability for upstream dialects.

***And we value your feedback!***

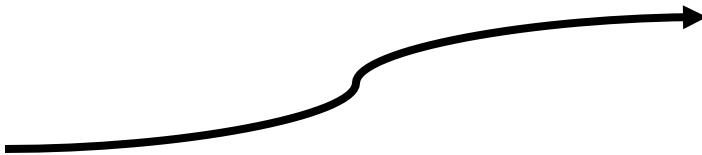


# Wrap Up

- Welcome & Logistics
- Introduction to AIEngines
- Physical MLIR-AIE
- Host Code, Simulation, Performance Counters
- Logical MLIR-AIE: Communication
- Advanced Topics and Next Steps
- Wrap Up

# Thank You

- Thank you for joining!
- Want to experiment more:
  - Clone <http://github.com/Xilinx/mlir-aie.git>
  - Clone <http://github.com/Xilinx/mlir-air.git>
  - Tutorials are online: <https://github.com/Xilinx/mlir-aie/tree/main/tutorials>  
No board needed!
  - Get access to HACCC: <https://www.amd-haccs.io> with pre-installed tools (and boards)
  - Contact: [xup@amd.com](mailto:xup@amd.com)
  - Internships
    - Interested in modelling new devices ?
    - Interested in targeting new applications ?
- Feedback – QR code link for feedback form
  - [Web link](#)



**AMD** 