System for Artificial Intelligence

# NPU Architecture & Ascend Programming

Siyuan Feng
Shanghai Innovation Institute

# Recap: Overview of Machine Learning Systems

上海创智学院
Shanghai Innovation Institute

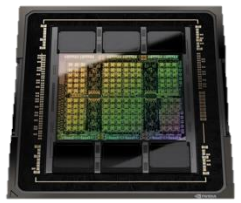
## ML Models

Automatic Differentiation

Graph Optimization

Parallelism / Distributed

Hardware Acceleration



NVIDIA GPU


HUAWEI NPU

This Lecture

Mobile devices

# OUTLINE

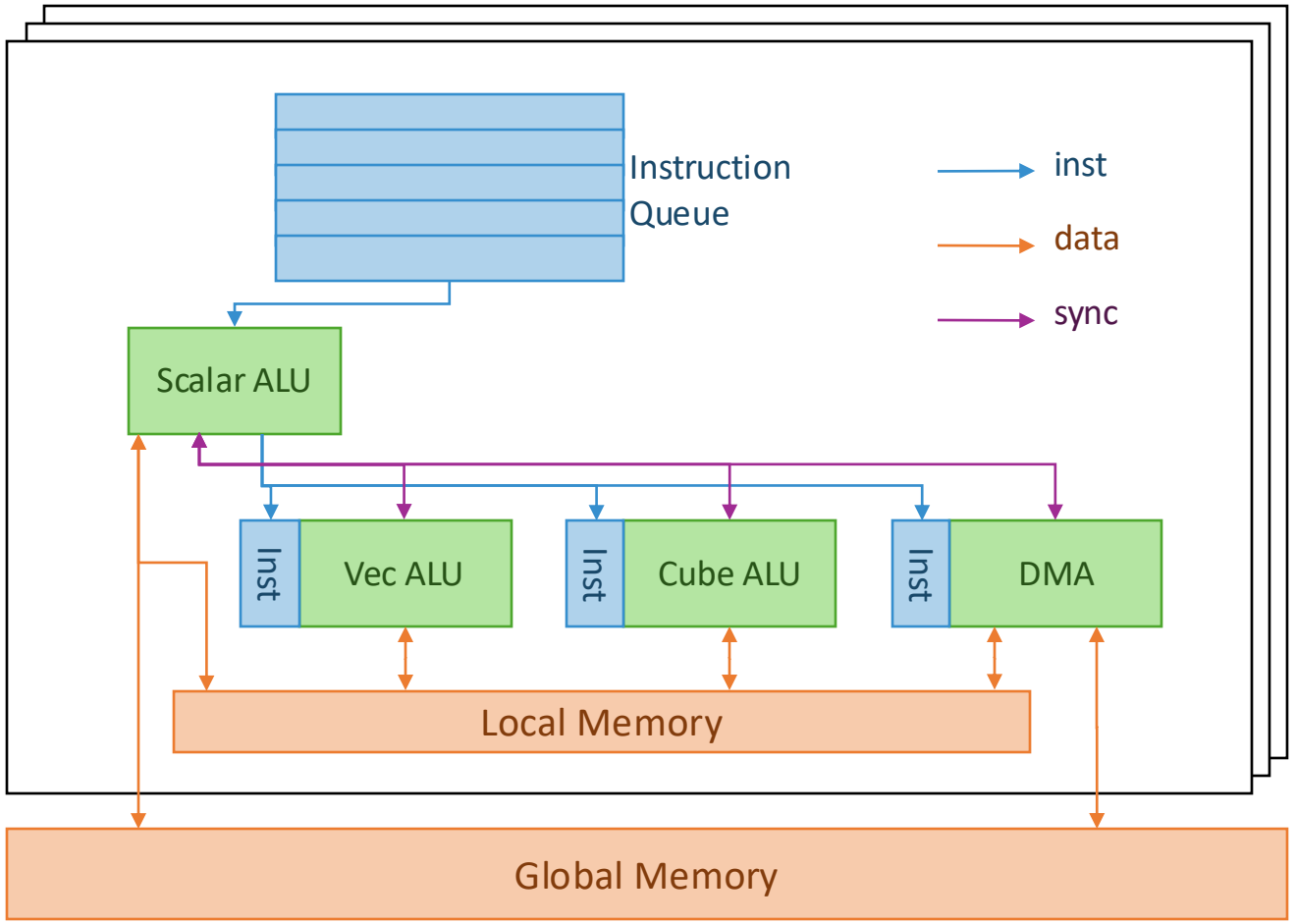
01 HUWEI NPU Architectures

02 Ascend C Programming

01

# HUWEI NPU Architectures

# NPU AI Core Architecture




Scalar Unit:

```
c = a + b
```

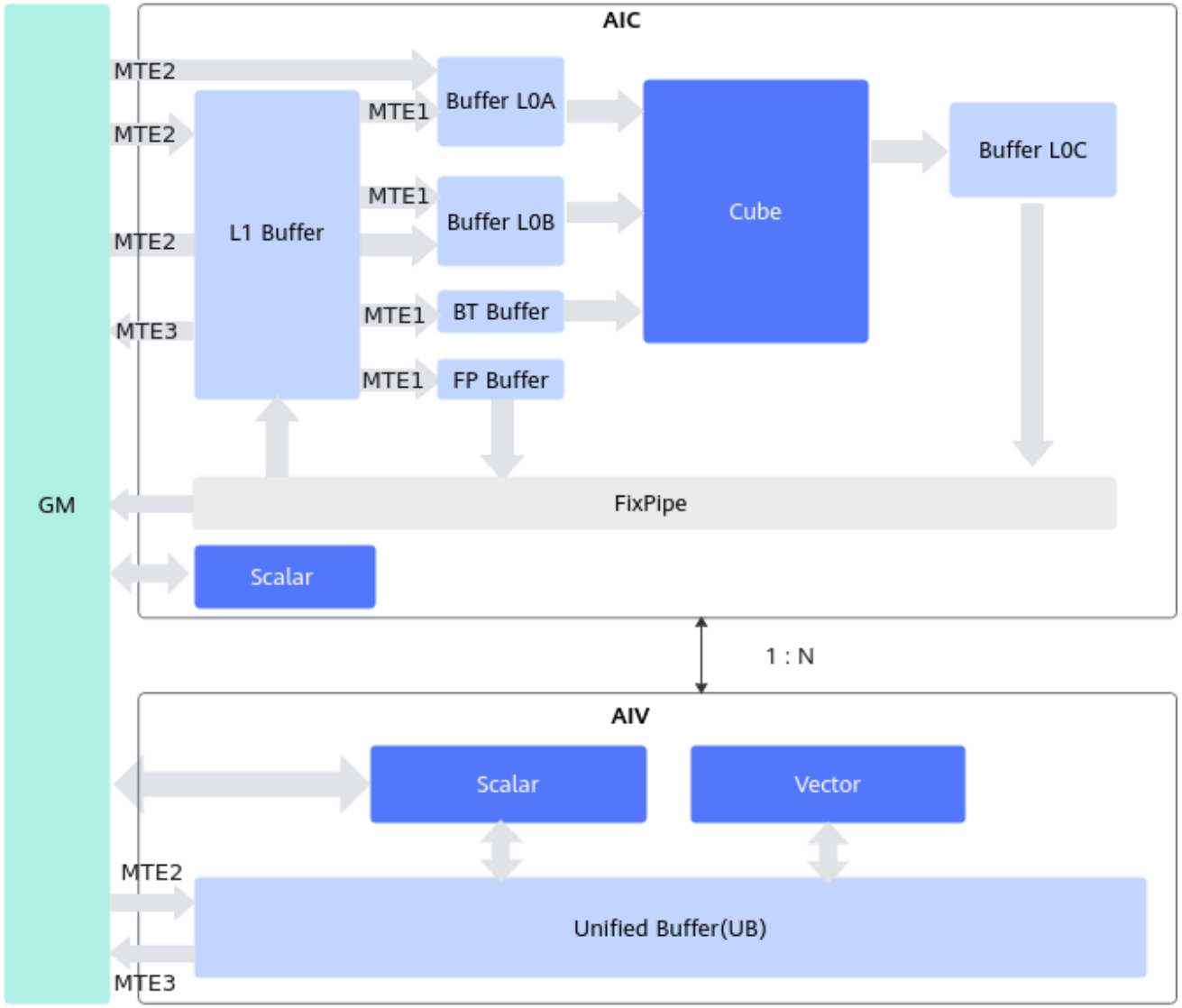
Vector Unit:

```
C[0:1024] = A[0:1024] + B[0:1024]
```

Cube Unit:

```
C[0:16, 0:16] +=
    A[0:16, 0:16] * B[0:16, 0:16]
```

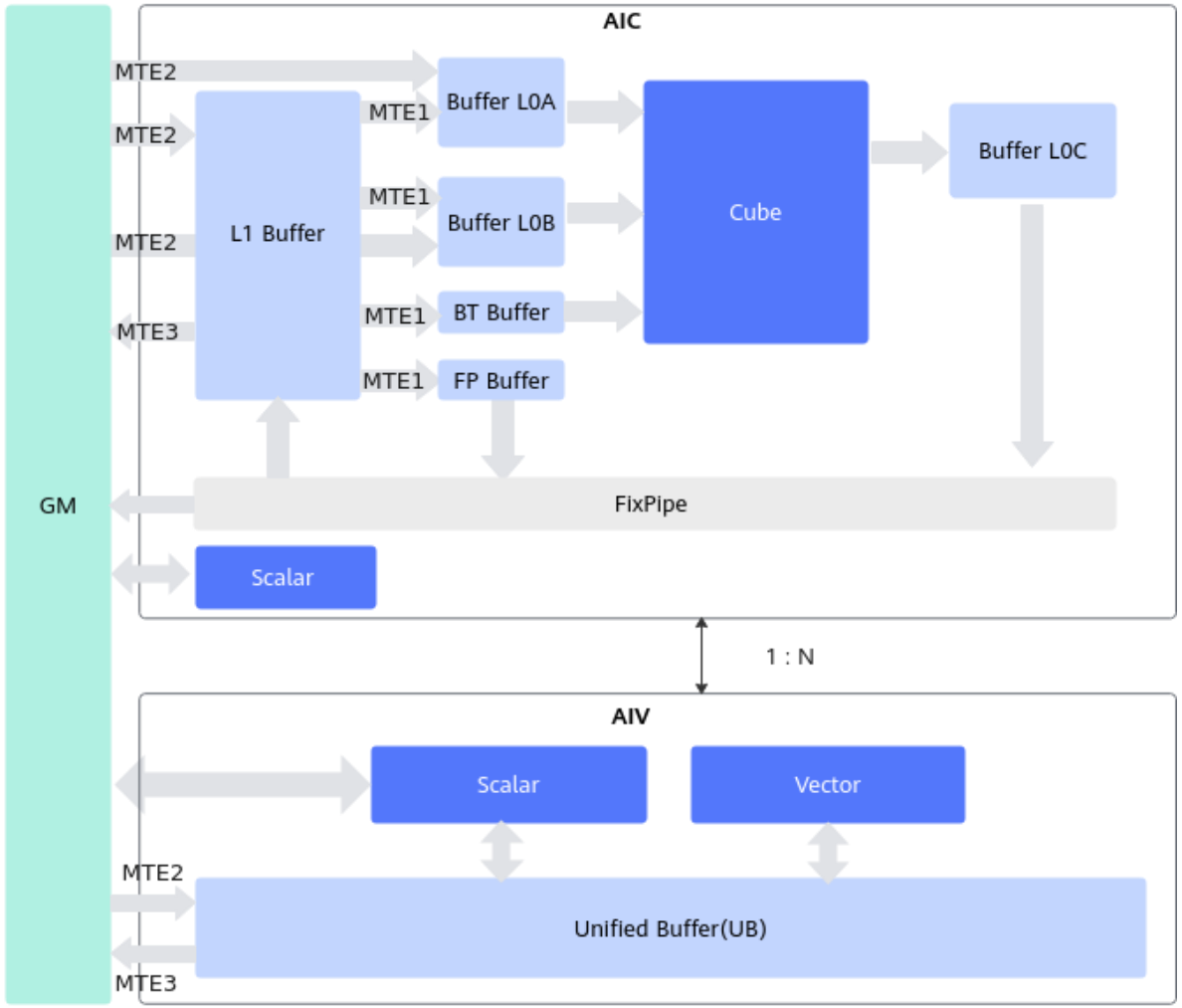
# Disaggregated AI Core Architecture (910 B / C)



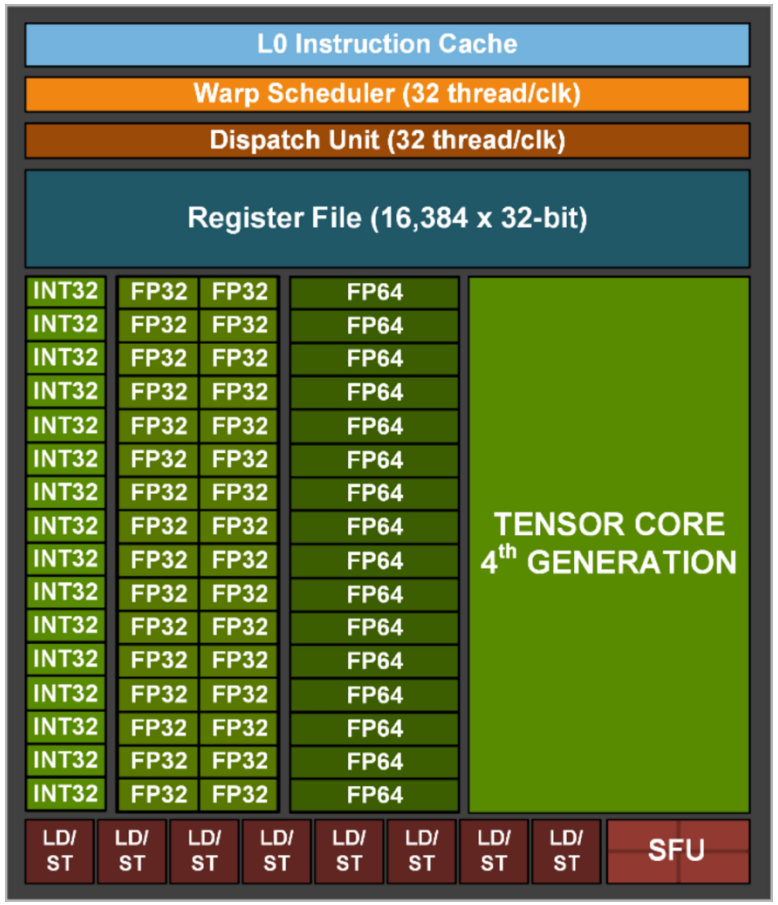

- Multi-level Memory Scope
- **Not "full-mesh"** access across ALU and memory

# Discussion: comparison between NPU and GPU

上海创智学院
Shanghai Innovation Institute




HUAWEI NPU Architecture



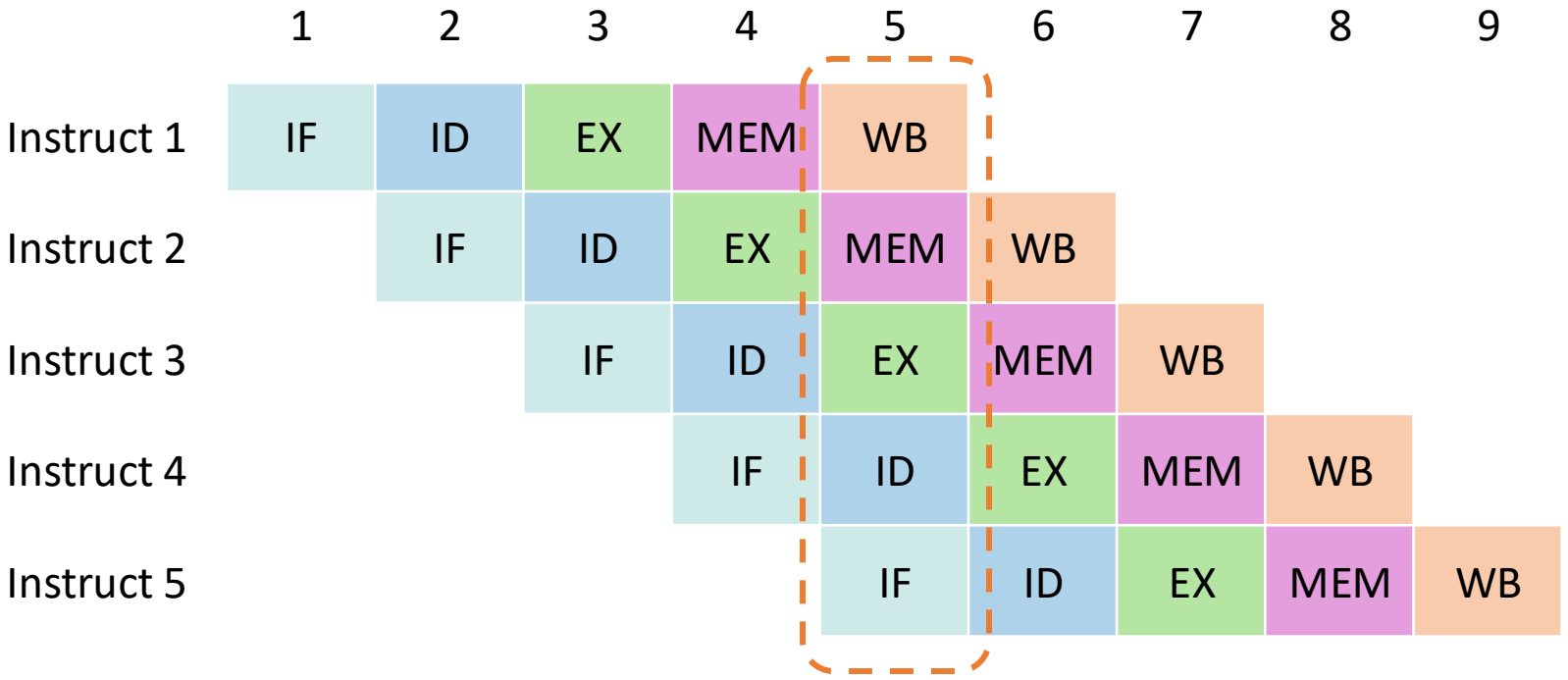

NVIDIA GPU Architecture

02

# Ascend C Programming

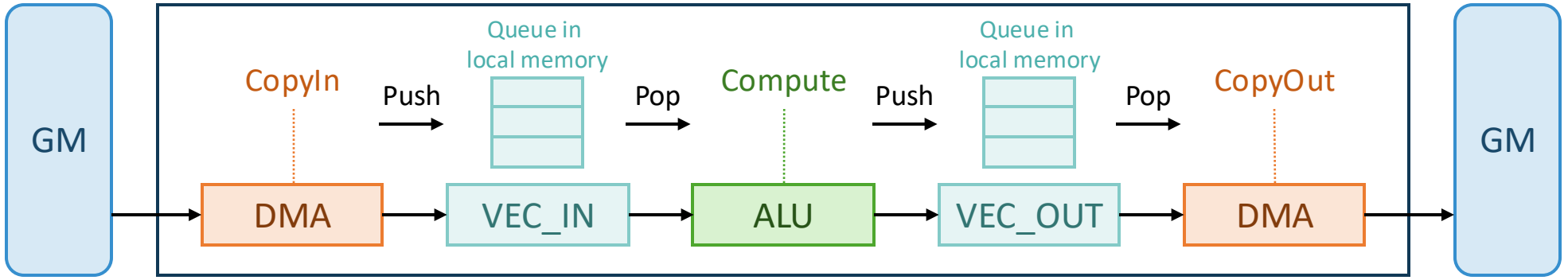
# Recap: Five-Stage Pipeline in CPU




**Discussion: Why?** **Utilize different hardware components at the same time**

# Ascend C Programming for AIV




Three tasks:

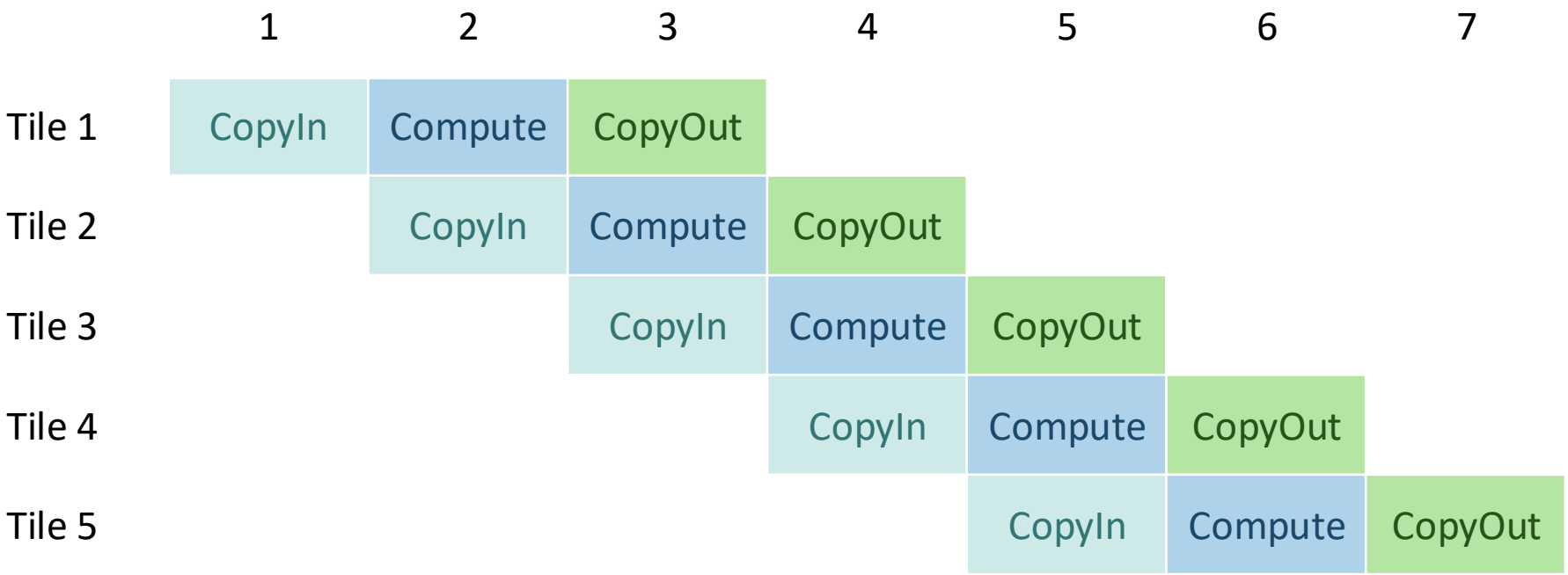
1. **CopyIn:** load data from global memory to local memory and push to queue VEC_IN
2. **Compute:** load from queue VEC_IN, do computation, and then store to VEC_OUT
3. **CopyOut:** load data from queue VEC_IN, and store data to VEC_OUT

# Pseudocode for AIV Programming

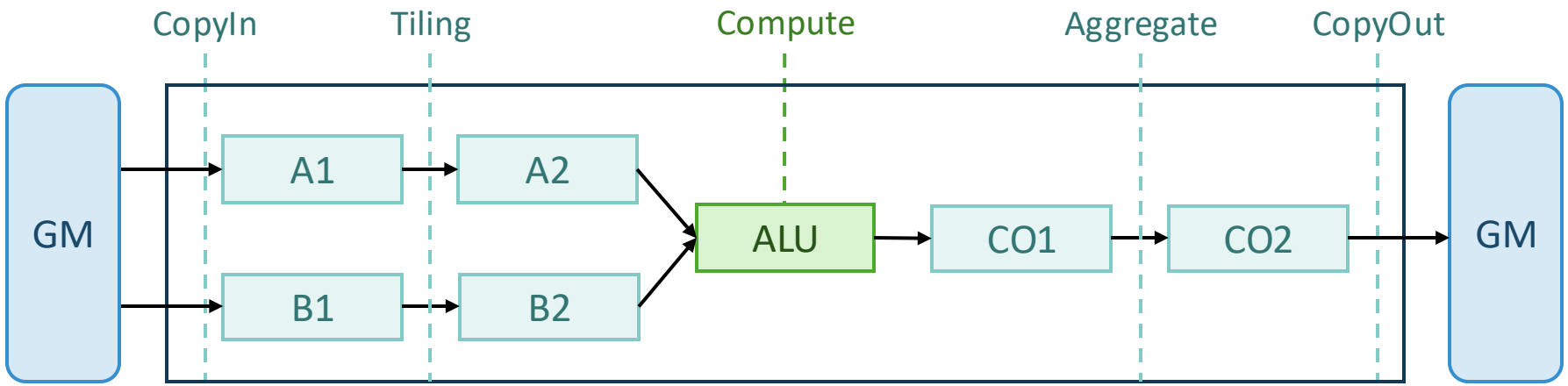
上海创智学院
Shanghai Innovation Institute

```
TQue<VecIn, 1> queIn;
TQue<VecOut, 1> queOut;

for-loop {
  // CopyIn Stage
  auto tensor = queIn.AllocTensor<half>();        // Allocate tensor at local for input
  DataCopy(tensor, gm, len);                      // Copy data from gm to local
  queIn.EnQue(tensor);                            // push tensor to queIn after load

  // Compute Stage
  auto tensor = queIn.DeQue<half>();              // Get tensor from queIn when ready
  auto tensorOut = queOut.AllocTensor<half>();    // Allocate tensor at local for output
  OP(tensorOut, tensor, 1024);                    // Main computation
  queIn.FreeTensor(tensor);                       // Free input tensor at local
  queOut.EnQue(tensorOut);                        // Push output tensor to queOut

  // CopyOut Stage
  auto tensor = queOut.DeQue<half>();             // Get tensor from queIn when ready
  DataCopy(gmOut, tensor, 1024);                  // Copy data from local to gm
  queOut.FreeTensor(tensor);                      // Free output tensor at local
}
```

# Pipeline of AIV



# Ascend C Programming for AIC




A1: L2 cache for matrix A
A2: L1 cache for matrix A
B1: L2 cache for matrix B
B2: L1 cache for matrix B

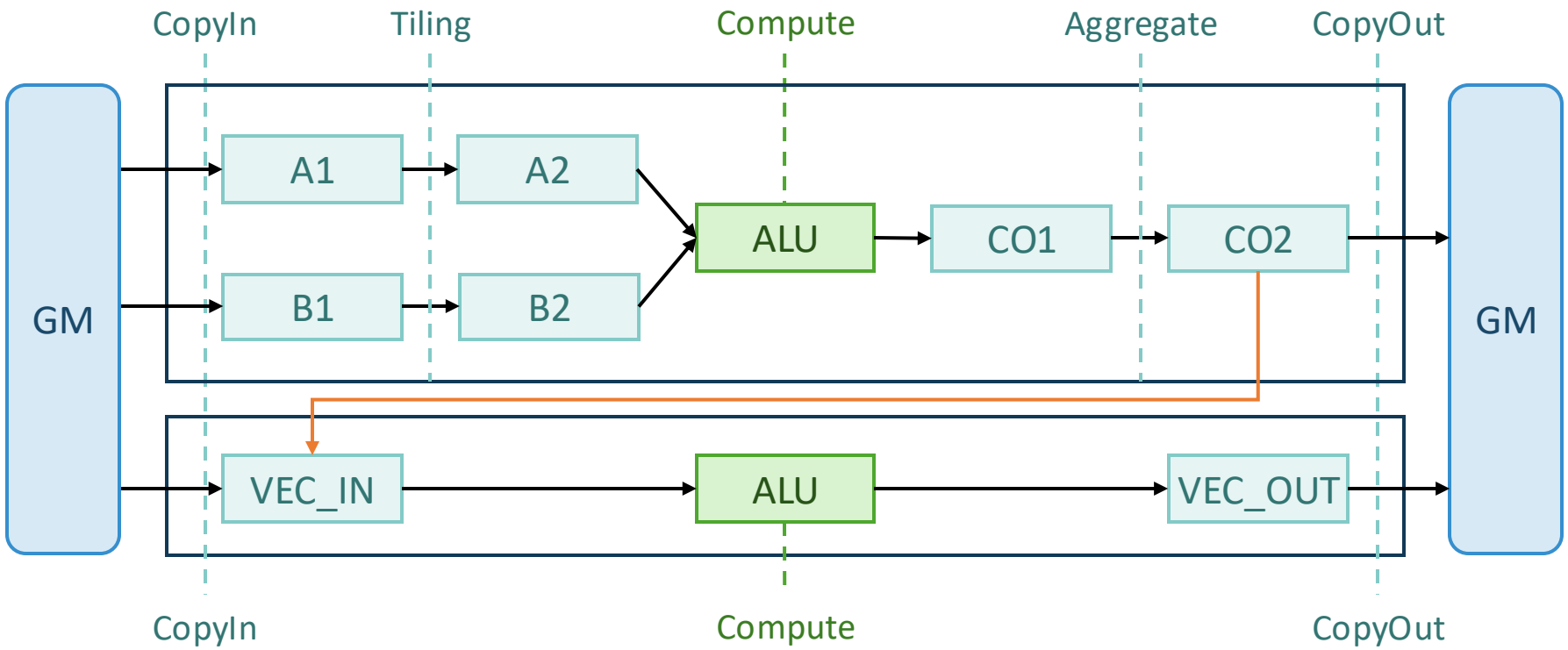
CO1: Output buffer for tiled C
CO2: Output buffer for C

# Pseudocode for AIC Programming

上海创智学院
Shanghai Innovation Institute

```
// Init
typedef MatmulType<TPosition::GM, CubeFormat::ND, half> aType;
typedef MatmulType<TPosition::GM, CubeFormat::ND, half> bType;
typedef MatmulType<TPosition::GM, CubeFormat::ND, float> cType;
typedef MatmulType<TPosition::GM, CubeFormat::ND, float> biasType;
Matmul<aType, bType, cType, biasType> mm;

REGIST_MATMUL_OBJ(&pipe, GetSysWorkSpacePtr(), mm, &tiling);

// CopyIn stage
mm.SetTensorA(gm_a);
mm.SetTensorB(gm_b);
mm.SetBias(gm_bias);

// Compute stage
while (mm.Iterate()) {
    // CopyOut Stage
    mm.GetTensorC(gm_c);
}
mm.End();
```

# Ascend C Programming for Fused Operator




A1: L2 cache for matrix A
A2: L1 cache for matrix A
B1: L2 cache for matrix B
B2: L1 cache for matrix B

CO1: Output buffer for tiled C
CO2: Output buffer for C
VEC_IN: input buffer for AIV
VEC_OUT: output buffer for AIC

# Pseudocode for Fused Operator

上海创智学院
Shanghai Innovation Institute

```
template<typename aType, typename bType, typename cType, typename biasType>
__aicore__ inline void MatmulLeakyKernel<aType, bType, cType, biasType>::Process() {
    // Step 1. Init
    REGIST_MATMUL_OBJ(&pipe, GetSysWorkSpacePtr(), matmulObj);
    matmulObj.Init(&tiling);
    matmulObj.SetTensorA(aGlobal);
    matmulObj.SetTensorB(bGlobal);
    matmulObj.SetBias(biasGlobal);

    // Step 2. iterate the tiling
    while (matmulObj.template Iterate<true>()) {
        // Step 3. Move the output to AIV
        reluOutLocal = reluOutQueue_.AllocTensor<cType>();
        matmulObj.template GetTensorC<true>(reluOutLocal, false, true);
       // Step 4. Compute on AIV
        AscendC::LeakyRelu(reluOutLocal, reluOutLocal, (cType)alpha, tiling.baseM * tiling.baseN);
        reluOutQueue_.EnQue(reluOutLocal);
        // Step 5. Write result back to GM
        reluOutQueue_.DeQue<cType>();
        ...
        AscendC::DataCopy(cGlobal[startOffset], reluOutLocal, copyParam);
        reluOutQueue_.FreeTensor(reluOutLocal);
    }
    matmulObj.End();
}
```

# Programming Paradigm

上海创智学院
Shanghai Innovation Institute

- **Utilize hardware by applying instruction pipeline**
- Each stage of tasks:
  - Allocate local memory or get local memory from upstream queue
  - Compute or copy data
  - Push processed data to downstream queue
  - Free useless local memory

# Appendix A: The Processing Flow of Enque / Deque

上海创智学院 Shanghai Innovation Institute

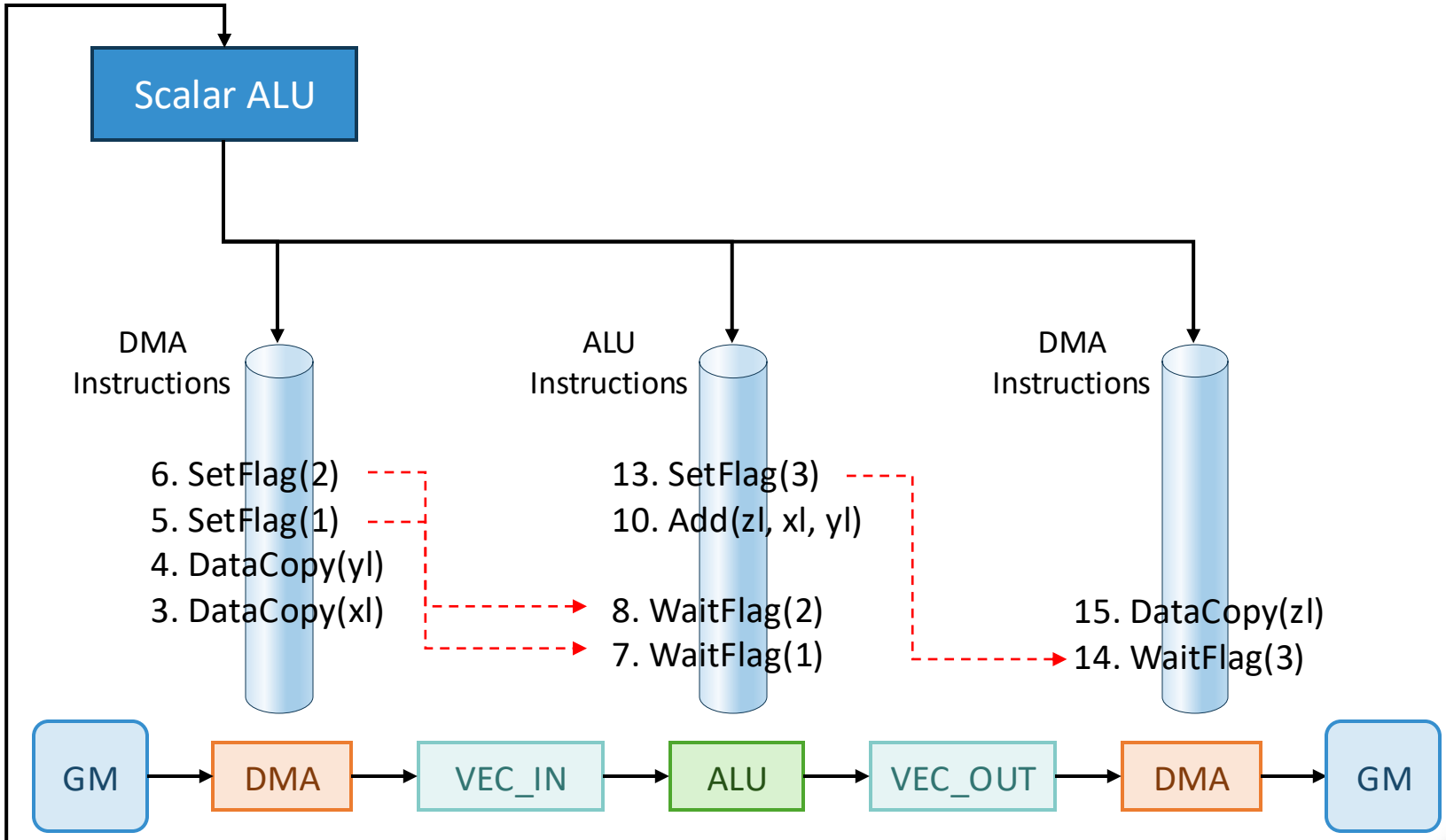
1. Scalar unit read the instruction
2. The instructions are dispatched to the corresponding unit's queues
3. The various execution units execute these instruction in parallel.
4. Enque/Deque resolves the **Write-After-Read (WAR) hazard** for memory:
    1. An **Enque** call issues a synchronization instruction (**set**), sending a signal to activate the wait.
    2. A **Deque** call issues a synchronization instruction (**wait**), waiting for the data write to complete.
    3. The wait must wait for the set signal before it can execute; otherwise, it is blocked.

# Appendix A: The Processing Flow of Enque / Deque

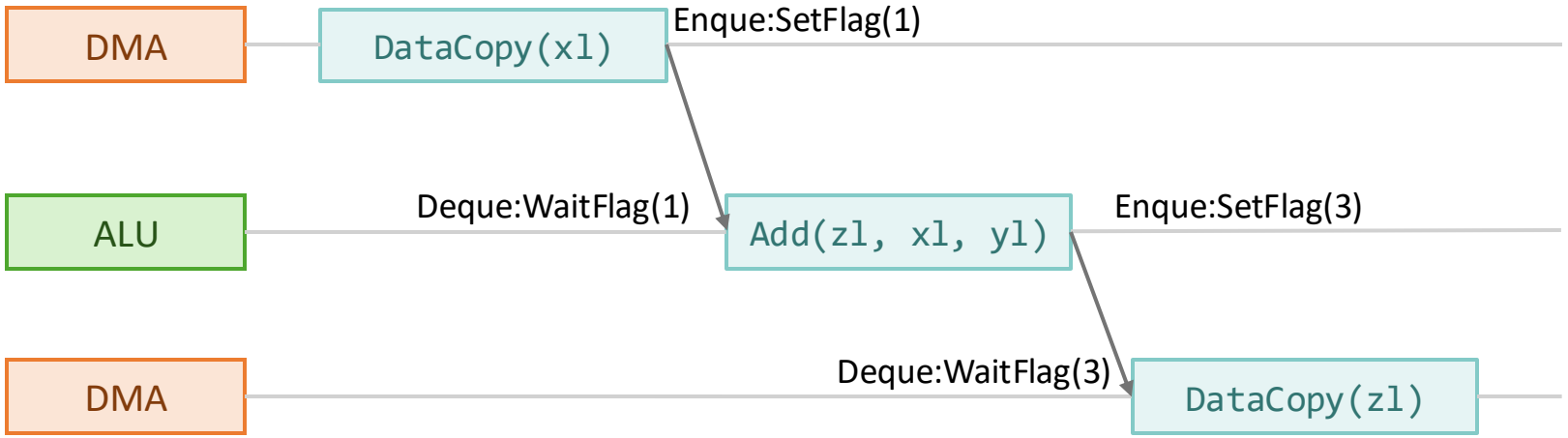
上海创智学院 Shanghai Innovation Institute

**Operator Instruction Sequence**

```
16. z.FreeTensor(zl)
15. DataCopy(zl)
14. zl = z.Deque()
13. z.Enque(zl)
12. y.FreeTensror(yl)
11. x.FreeTensror(xl)
10. Add (zl, xl, yl)
9.  zl = z.AllocTensor()
8.  yl = y.Deque()
7.  xl = x.Deque()
6.  y.Enque(yl)
5.  x.Enque(xl)
4.  DataCopy(yl)
3.  DataCopy(xl)
2.  yl = y.AllocTensor()
1.  xl = x.AllocTensor()
```



# Appendix A: The Processing Flow of Enque / Deque



# Appendix B: The Processing Flow of AllocTensor / FreeTensor



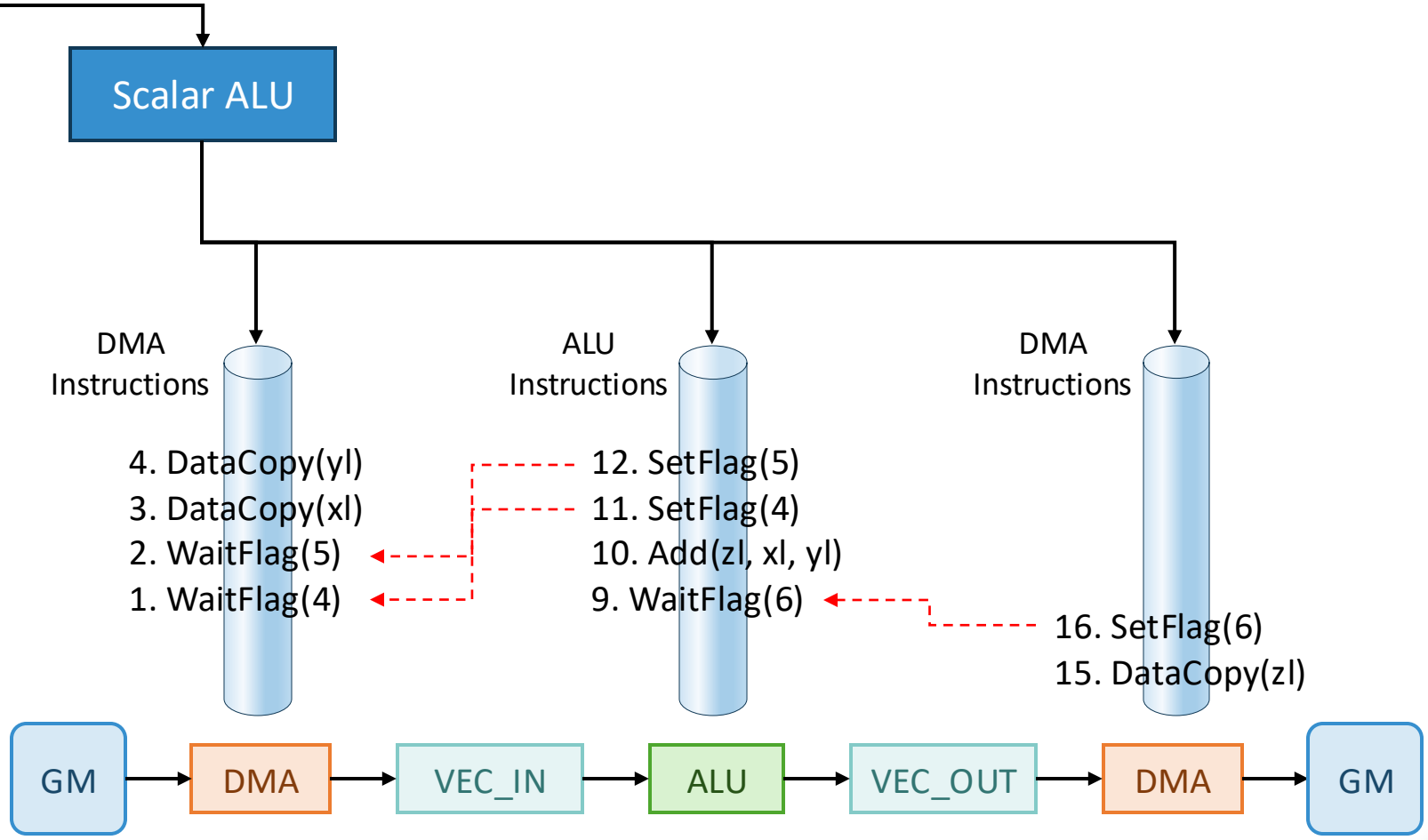
1. Scalar unit read the instruction
2. The instructions are dispatched to the corresponding unit's queues
3. The various execution units execute these instruction in parallel.
4. AllocTensor/FreeTensor resolves the **Read-After-Write (RAW)** hazard:
    1. An **AllocTensor** call issues a synchronization instruction (**wait**), waiting for the memory to finish being read.
    2. A **FreeTensor** call issues a synchronization instruction (**set**), signaling that the memory is released and can be repeatedly written to.
    3. The wait must wait for the set signal before it can execute; otherwise, it is blocked.

# Appendix B: The Processing Flow of AllocTensor / FreeTensor

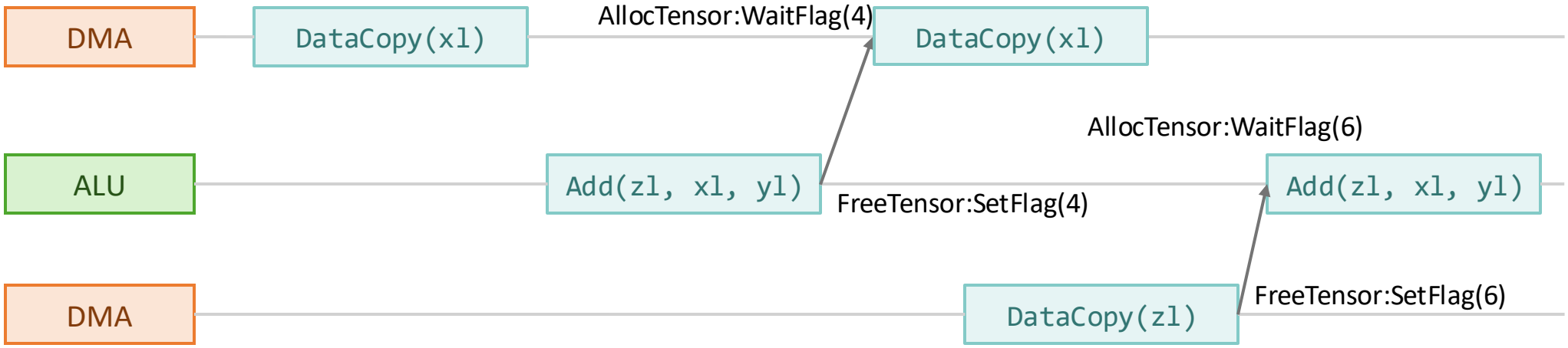
上海创智学院 Shanghai Innovation Institute

**Operator Instruction Sequence**

```
16. z.FreeTensor(zl)
15. DataCopy(zl)
14. zl = z.Deque()
13. z.Enque(zl)
12. y.FreeTensror(yl)
11. x.FreeTensror(xl)
10. Add (zl, xl, yl)
9.  zl = z.AllocTensor()
8.  yl = y.Deque()
7.  xl = x.Deque()
6.  y.Enque(yl)
5.  x.Enque(xl)
4.  DataCopy(yl)
3.  DataCopy(xl)
2.  yl = y.AllocTensor()
1.  xl = x.AllocTensor()
```



# Appendix B: The Processing Flow of AllocTensor / FreeTensor



# Acknowledgement

The development of this course, including its structure, content, and accompanying presentation slides, has been significantly influenced and inspired by the excellent work of instructors and institutions who have shared their materials openly. We wish to extend our sincere acknowledgement and gratitude to the following courses, which served as invaluable references and a source of pedagogical inspiration:

- Machine Learning Systems[15-442/15-642], by **Tianqi Chen** and **Zhihao Jia** at **CMU**.
- Advanced Topics in Machine Learning (Systems)[CS6216], by **Yao Lu** at **NUS**

While these materials provided a foundational blueprint and a wealth of insightful examples, all content herein has been adapted, modified, and curated to meet the specific learning objectives of our curriculum. Any errors, omissions, or shortcomings found in these course materials are entirely our own responsibility. We are profoundly grateful for the contributions of the educators listed above, whose dedication to teaching and knowledge-sharing has made the creation of this course possible.

System for Artificial Intelligence

# Thanks

Siyuan Feng
Shanghai Innovation Institute