



# Speaking Composable Kernel

**Haocong Wang**  
**(Representing the Composable Kernel Team @AMD)**

**Courtesy: Chao Liu**

**July 20<sup>th</sup>, 2024**

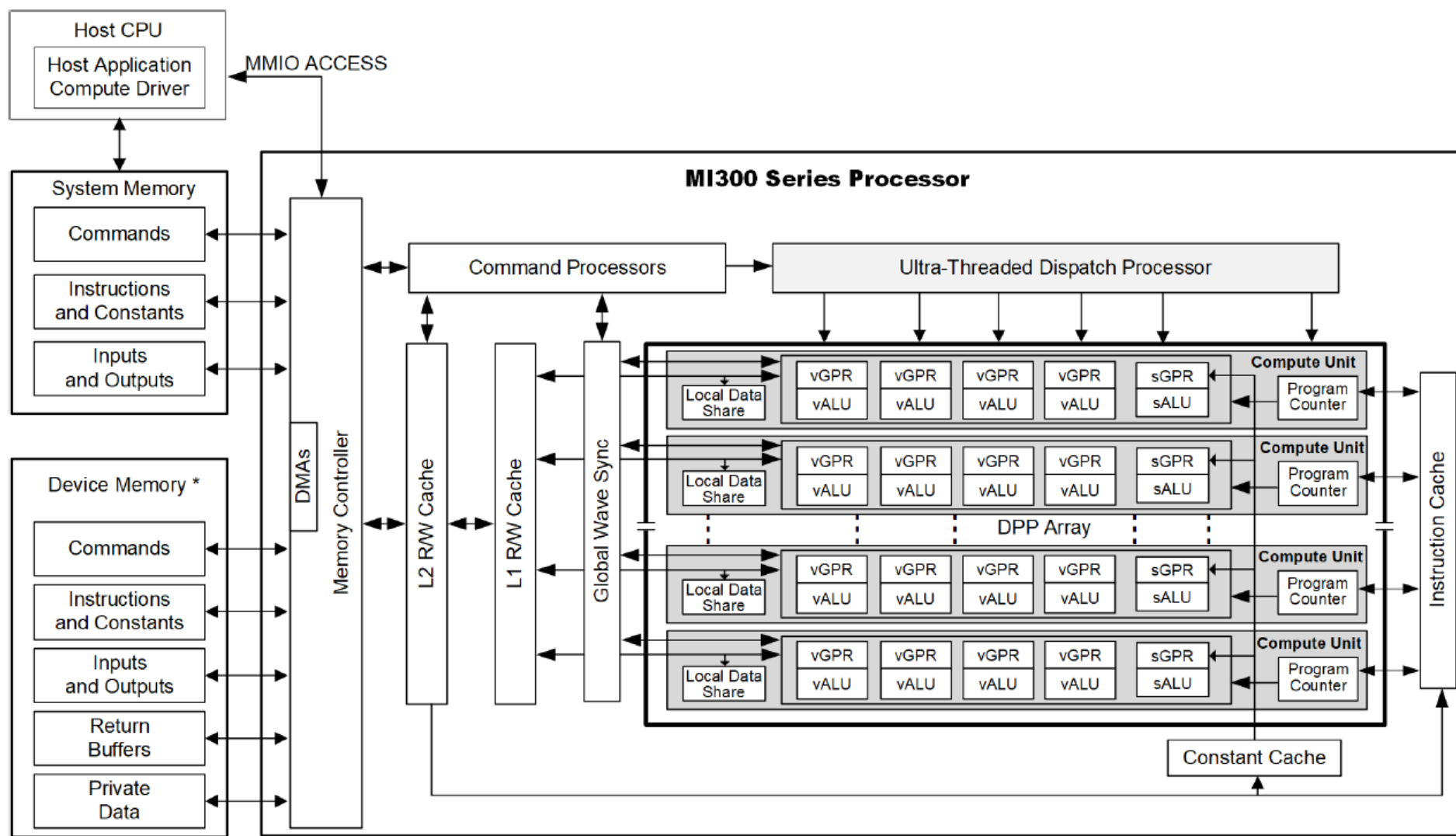
**AMD**   
together we advance\_

# Agenda

---

- ❖ **High GPU utilization is Challenging**
  - ❖ **Composable Kernel**
  - ❖ **Kernel customization**
  - ❖ **ROCm Flash-Attention**
  - ❖ **Q&A**
- 

**AMD**   
**ROCm**



**High GPU utilization is Challenging**

# Mapping custom AI workloads for high GPU utilization

- **Complexity of GPU programming model**
  - Complicated memory hierarchy, global memory, shared memory, register file, multiple-level cache...
  - Multiple types of compute units, VALU, DPP, matrix cores...
  - The rapid iteration of hardware, makes the above even harder for ordinary users, making it difficult to fully utilize the GPU's performance
- **Highly customizable algorithms**
  - CNN, high-dimensional, irregular image size, sophisticated mapping
  - Fused kernel, flash-attention for example, including multiple operations
  - The rapid development of machine learning has led to a surge in customized demands
- **Developer productivity**
  - Developers need a tool that is suitable for rapid development while also delivering high performance on GPU

Hardware Portability

Operator Extendability

Extract More Performance



Client APIs

- C++ APIs for precompile kernels
- (Planned) Python APIs

Instantiated  
Kernels and  
Invokers

Instantiate:

- Arbitrary point-wise activation/reduction functions
- Arbitrary datatypes
- Arbitrary tensor memory layouts

Templated  
Kernels and  
Invokers

Templates for Fused and Non-fused ML Kernels  
(Conv/GEMM/reduction, Attention,  
GEMM/Conv+GEMM/Conv, etc)

Templated  
Tile  
Operators

Fundamental Templated Tile Operators  
(GEMM/reduction/data transfer)

Programmable  
for ML System  
Experts

Programmable  
for ML Kernel  
Experts

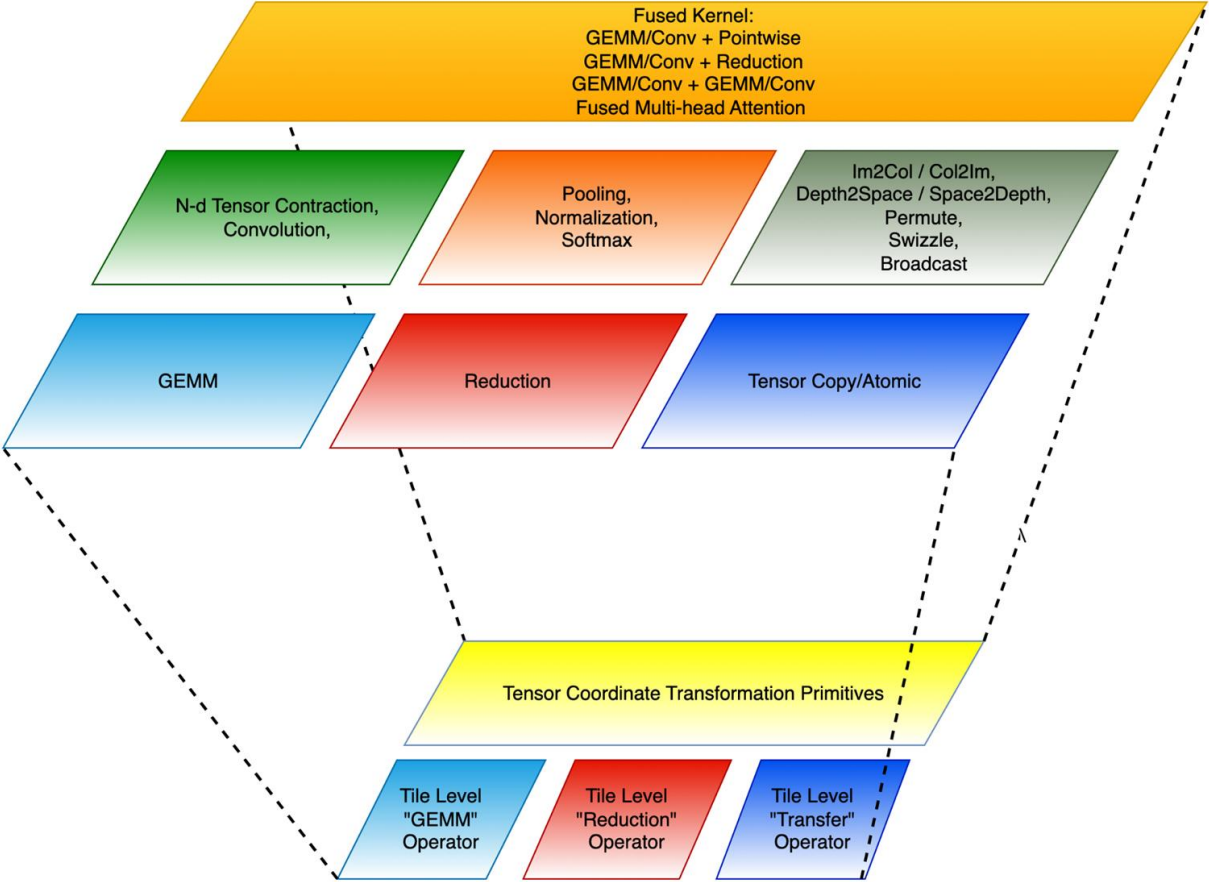


Composable Kernel

# Design Philosophies

- **At the core: a programming paradigm for Performance, Productivity & Portability**
  - **Systematic & Self-sufficient, Support all tensor operators, Express all optimization techniques, abstract all hardware**
  - **Composable & Reusable coding component**
- **User productivity**
  - **Optimized hierarchical API calls**
  - **C++ templated**
- **Vendor optimized high performance code**

# Key Abstractions in CK



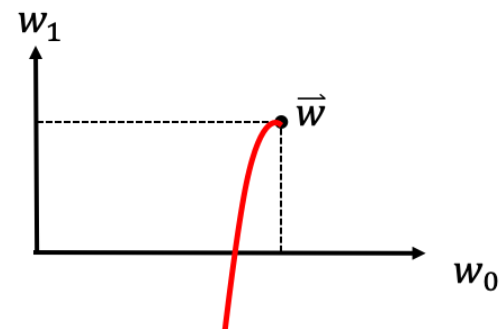
**Coordinate Transformation Primitives**  
Reduce algorithm complexity

**Tile Programming**  
Intuitive & Productive programming interface

# Coordinate Transformation Primitives

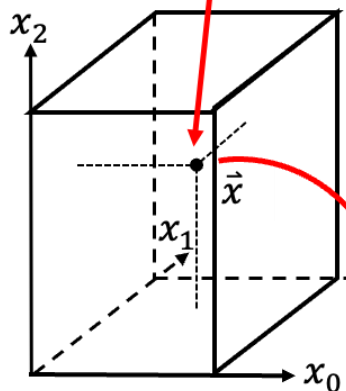
Transformed  
Coordinate Space W

Abstraction



Transformation function  
 $\vec{x} = f(\vec{w})$

Naive  
Coordinate Space X



Address function  
 $y = g(\vec{x})$

Raw Memory  
Coordinate Space Y

Address y	123	124	125	126	127	...	234	235	236
Data	1.0	2.5	1.3	6.2	2.1	...	0.7	1.3	3.5

CK API

make\_pad\_transform()  
make\_merge\_transform()  
make\_xor\_transform()  
make\_passthrough\_transform()  
make\_freeze\_transform()  
...

make\_naive\_tensor\_view<>()

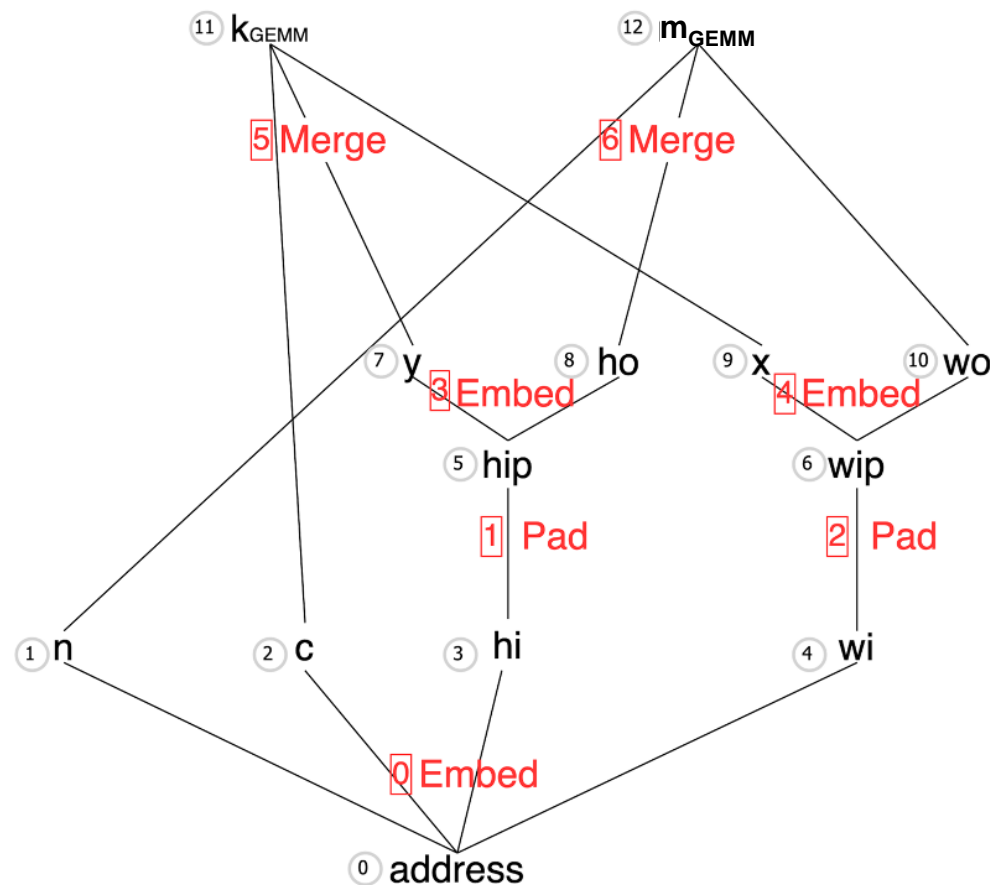
AddressSpaceEnum::Global  
AddressSpaceEnum::Lds  
AddressSpaceEnum::Vgpr...

InMemoryDataOperationEnum::Set  
InMemoryDataOperationEnum::AtomicAdd



# Coordinate Transformation Primitives

## Implicit GEMM Convolution example



Im2Col Transformation Graph for  
Implicit GEMM Convolution

```

1  < const auto a_n_hi_wi_c = make_naive_tensor_view_packed<AddressSpaceEnum::Global>(
2      p_a_img, make_tuple(N, Hi, Wi, C), Number<32>{})
3
4  < const auto a_n_hip_wip_c = transform_tensor_view(
5      a_n_hi_wi_c,
6      < make_tuple(make_pass_through_transform(N),
7                  make_pad_transform(Hi, InLeftPadH, InRightPadH),
8                  make_pad_transform(Wi, InLeftPadW, InRightPadW),
9                  make_pass_through_transform(C)),
10     make_tuple(Sequence<0>{}, Sequence<1>{}, Sequence<2>{}, Sequence<3>{}),
11     make_tuple(Sequence<0>{}, Sequence<1>{}, Sequence<2>{}, Sequence<3>{}));
12
13 < const auto a_n_y_ho_x_wo_c = transform_tensor_view(
14     a_n_hip_wip_c,
15     < make_tuple(
16         make_pass_through_transform(N),
17         make_embed_transform(make_tuple(Y, Ho), make_tuple(ConvDilationH, ConvStrideH)),
18         make_embed_transform(make_tuple(X, Wo), make_tuple(ConvDilationW, ConvStrideW)),
19         make_pass_through_transform(C)),
20     make_tuple(Sequence<0>{}, Sequence<1>{}, Sequence<2>{}, Sequence<3>{}),
21     make_tuple(Sequence<0>{}, Sequence<1, 2>{}, Sequence<3, 4>{}, Sequence<5>{}));
22
23 < const auto src_gemmm_gemm_k =
24     transform_tensor_view(a_n_y_ho_x_wo_c,
25     < make_tuple(
26         make_merge_transform(make_tuple(N, Ho, Wo)),
27         make_merge_transform(make_tuple(Y, X, C)),
28         make_tuple(Sequence<0, 2, 4>{}, Sequence<1, 3, 5>{}),
29         make_tuple(Sequence<0>{}, Sequence<1>{}));

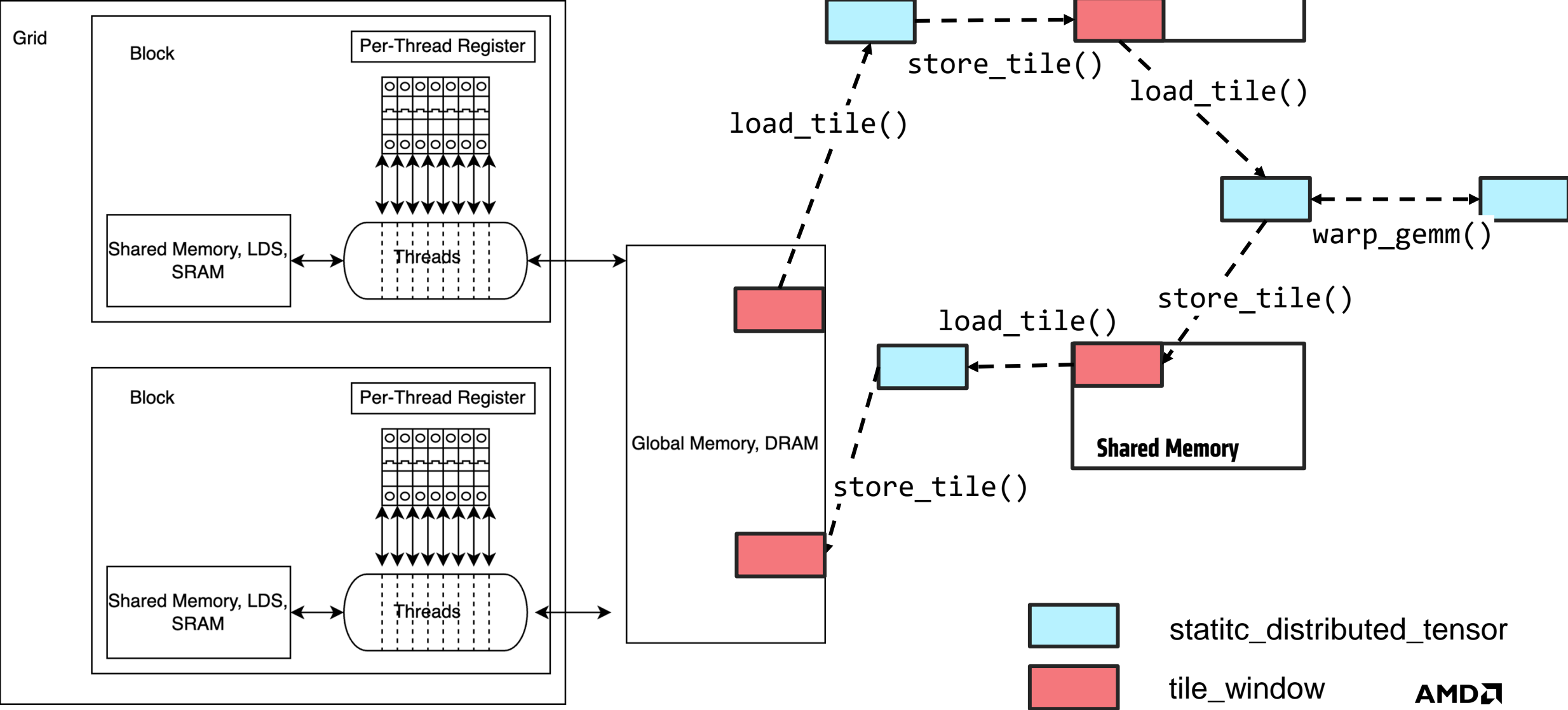
```

Concise Im2Col implementation code with  
Coordinate Transformation Primitives

# Tile Programming

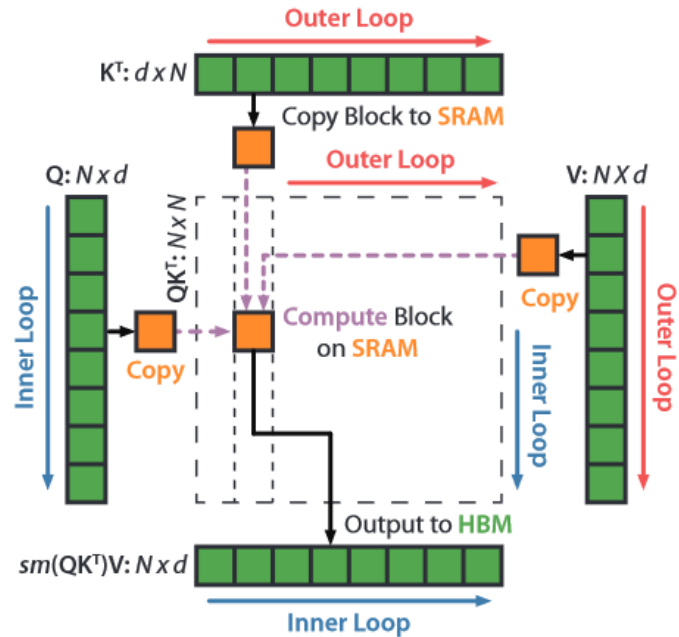
- **Goal: Non-compromised Productivity and Performance**
  - **Non-GPU experts can write functional kernels**
  - **GPU experts can hyper-optimize kernels**
  - **Architectural Independent code**
- **Design decision**
  - **Only tile level interface (both tensor operator and operand) to hide GPU complexity**
  - **Interface reflects conceptual data layout for math (not implementation)**
  - **Able to express all optimization**
- **Solution**
  - **Distributed Tensor - Tile level view data structure - regardless of private memory space**
  - **Generalize a set of reusable tile level operators and APIs**
  - **Policy - To inject optimization**

# Tile Programming



# Tile Programming

## Flash Attention forward pass example



for  $1 \leq j \leq T_c$  do

Load  $K_j, V_j$  from HBM to on-chip SRAM.

On chip, compute  $S_i^{(j)} = Q_i K_j^T \in \mathbb{R}^{B_r \times B_c}$ .

On chip, compute  $m_i^{(j)} = \max(m_i^{(j-1)}, \text{rowmax}(S_i^{(j)})) \in \mathbb{R}^{B_r}$

$$\tilde{P}_i^{(j)} = \exp(S_i^{(j)} - m_i^{(j)}) \in \mathbb{R}^{B_r \times B_c}$$

$$\ell_i^{(j)} = e^{m_i^{j-1} - m_i^{(j)}} \ell_i^{(j-1)} + \text{rowsum}(\tilde{P}_i^{(j)}) \in \mathbb{R}^{B_r}$$

On chip, compute  $O_i^{(j)} = \text{diag}(e^{m_i^{(j-1)} - m_i^{(j)}})^{-1} O_i^{(j-1)} + \tilde{P}_i^{(j)} V_j$ .

end for

```
// Sacc{j} = Q * K{j}
const auto s_acc =
    gemm0_pipeline(q_dram_window, k_dram_window, K0 / kK0PerBlock, smem_ptr);

// S{j}
const auto s =
    tile_elementwise_in(type_convert<SMPLComputeDataType, SaccDataType>, s_acc);

// m_local = rowmax(S{j})
auto m_local = block_tile_reduce<SMPLComputeDataType>(
    s, Sequence<1>{}, f_max, NumericLimits<SMPLComputeDataType>::Lowest());

block_tile_reduce_sync(m_local, f_max);

// m{j-1}
const auto m_old = m;

// m{j}
tile_elementwise_inout(
    [](auto& e0, auto e1, auto e2) { e0 = max(e1, e2); }, m, m_old, m_local);

// Pcompute{j}
auto p_compute =
    make_static_distributed_tensor<SMPLComputeDataType>(s.GetTileDistribution());

constexpr auto p_spans = decltype(p_compute)::GetDistributedSpans();

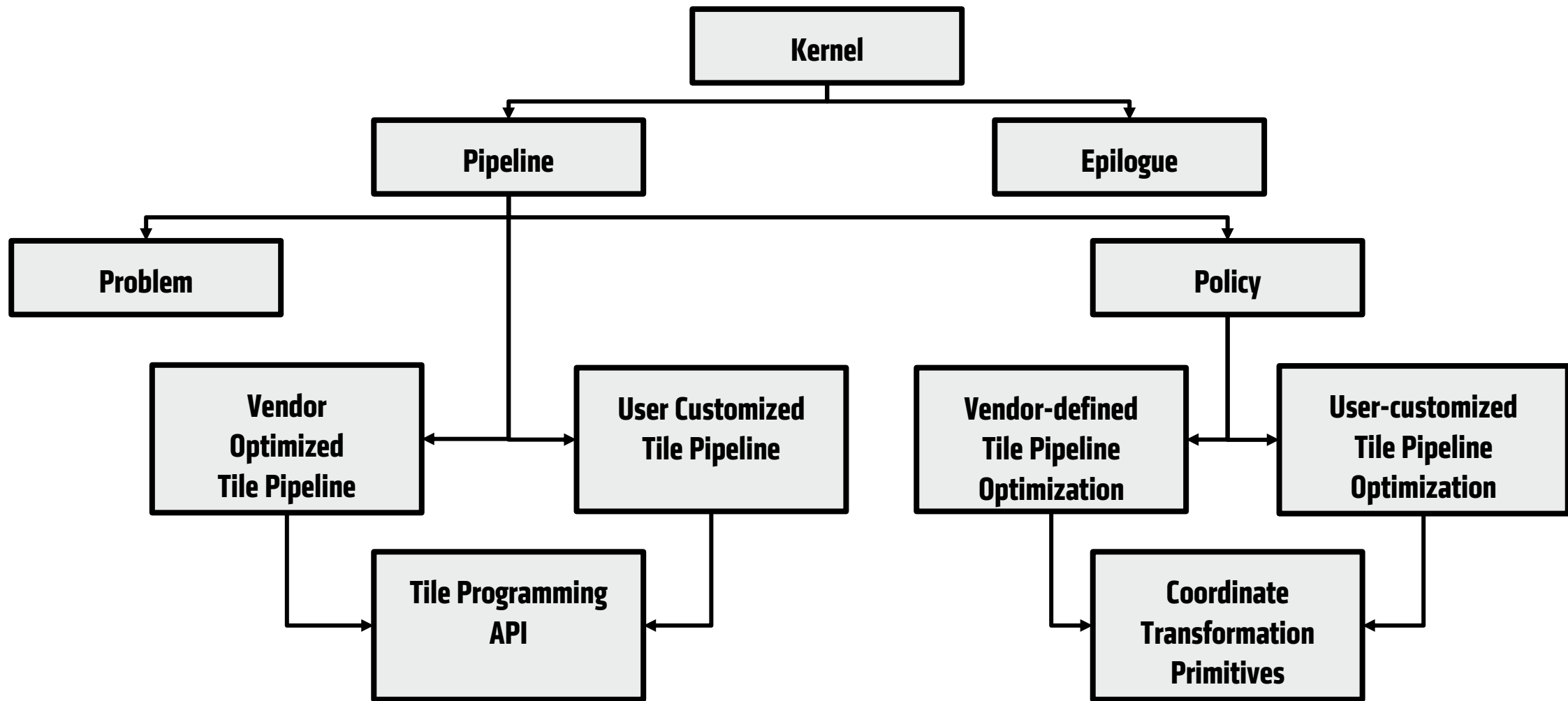
sweep_tile_span(p_spans[I0], [&](auto idx0) {
    constexpr auto i_idx = make_tuple(idx0);

    sweep_tile_span(p_spans[I1], [&](auto idx1) {
        constexpr auto i_j_idx = make_tuple(idx0, idx1);

        p_compute(i_j_idx) = math::exp(s[i_j_idx] - m[i_idx]);
    });
});

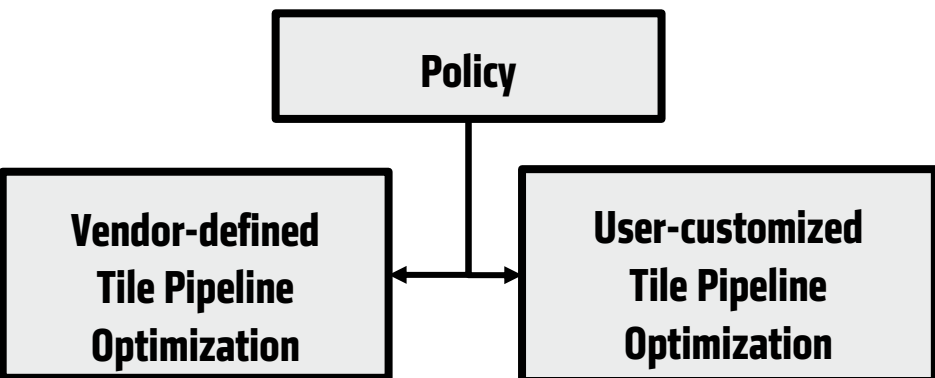
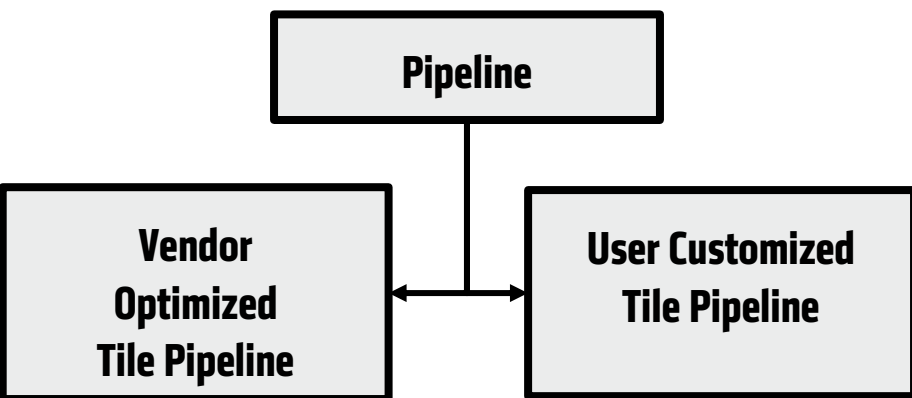
// rowsum(Pcompute{j})
auto rowsum_p = block_tile_reduce<SMPLComputeDataType>(
    p_compute, Sequence<1>{}, f_sum, SMPLComputeDataType{0});

block_tile_reduce_sync(rowsum_p, f_sum);
```



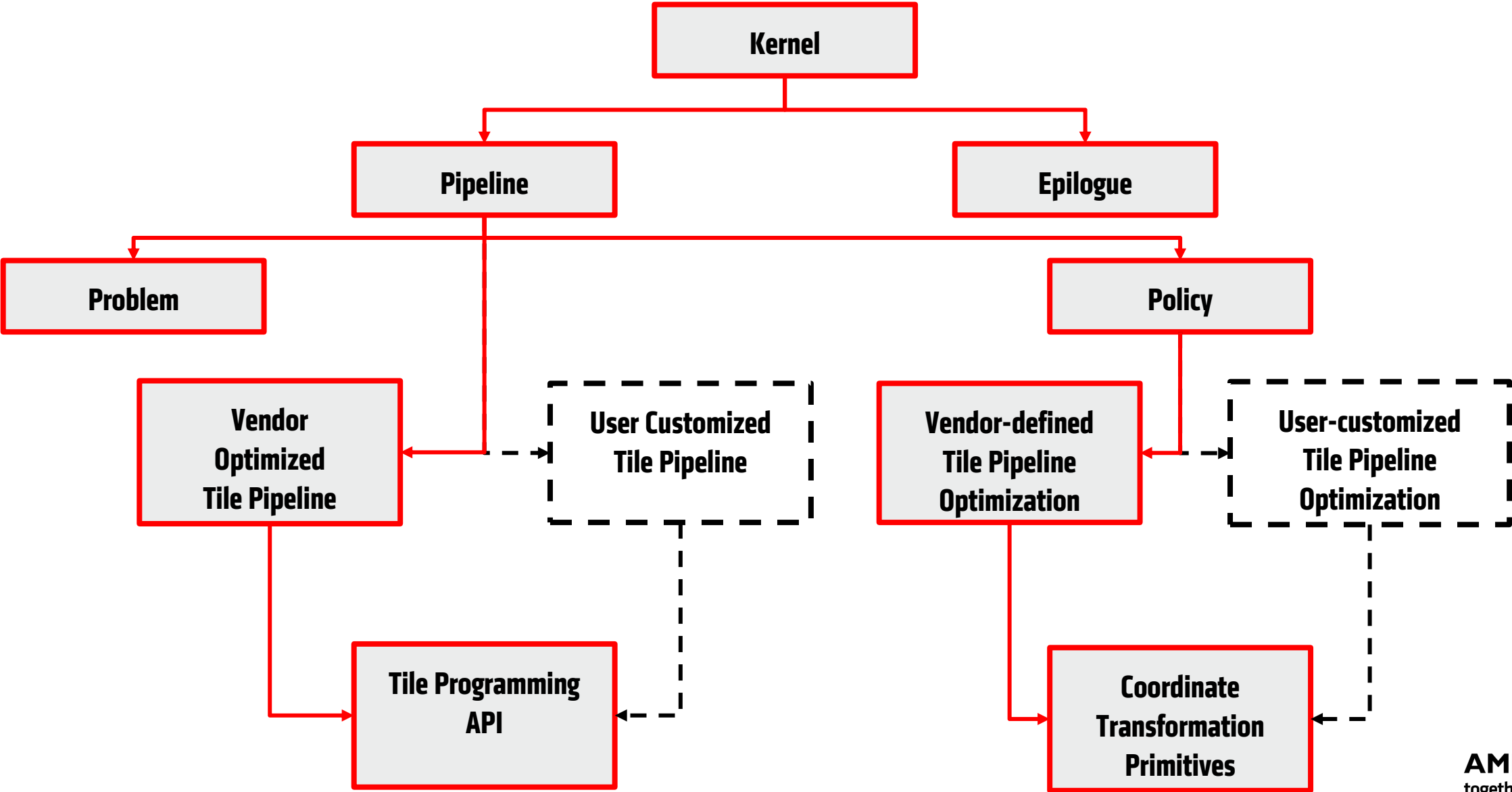
**Kernel customization**

# Kernel customization

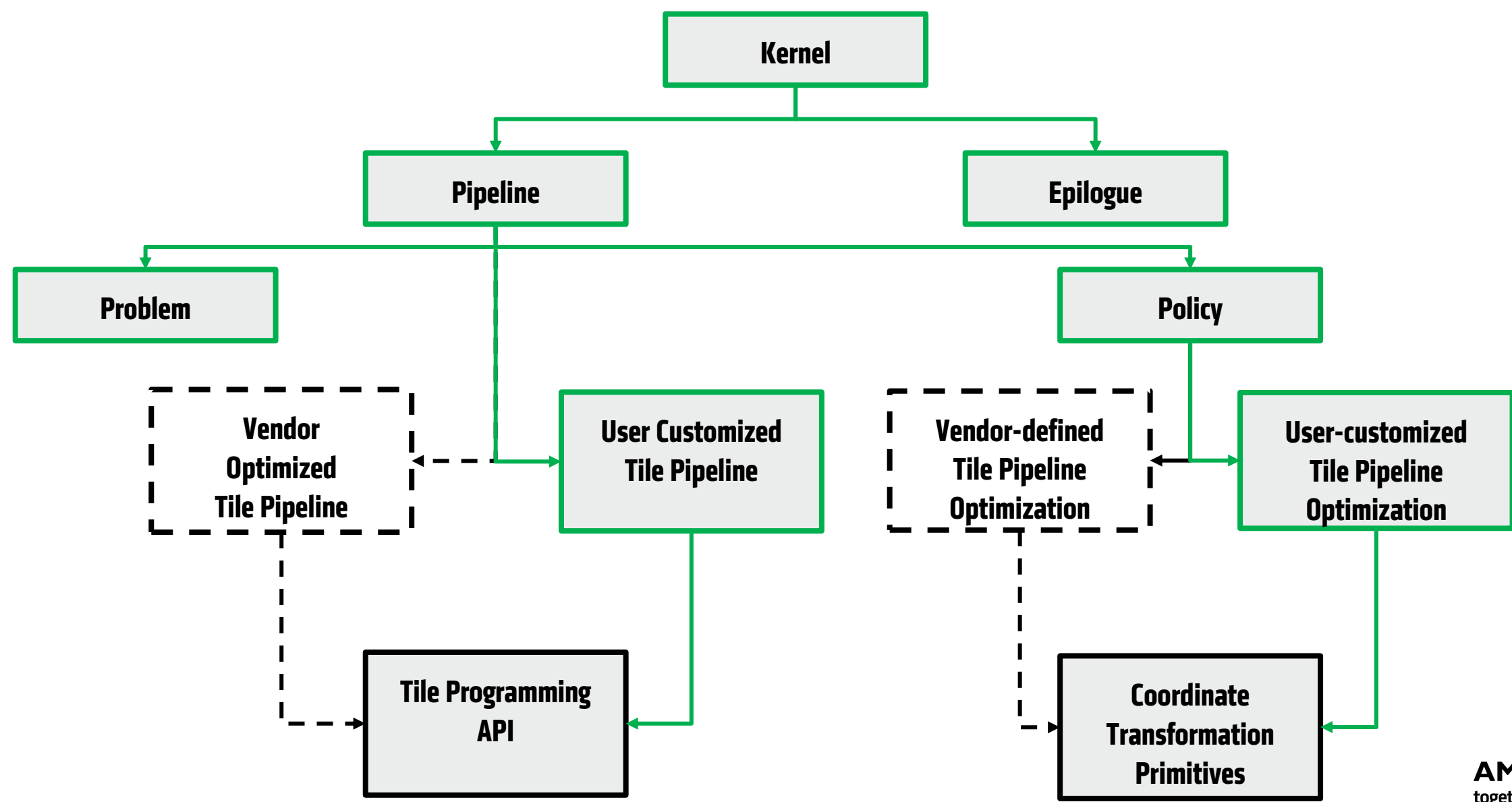


- **Pipeline**
  - Compose data movement and computation component
  - Tile Programming API to provide Intuitive & Productive programming interface
- **Policy**
  - Optimization applied to specific pipeline
  - Define data movement and computation component
  - Coordinate Transformation Primitives to hide complexity

# Vendor optimized kernel

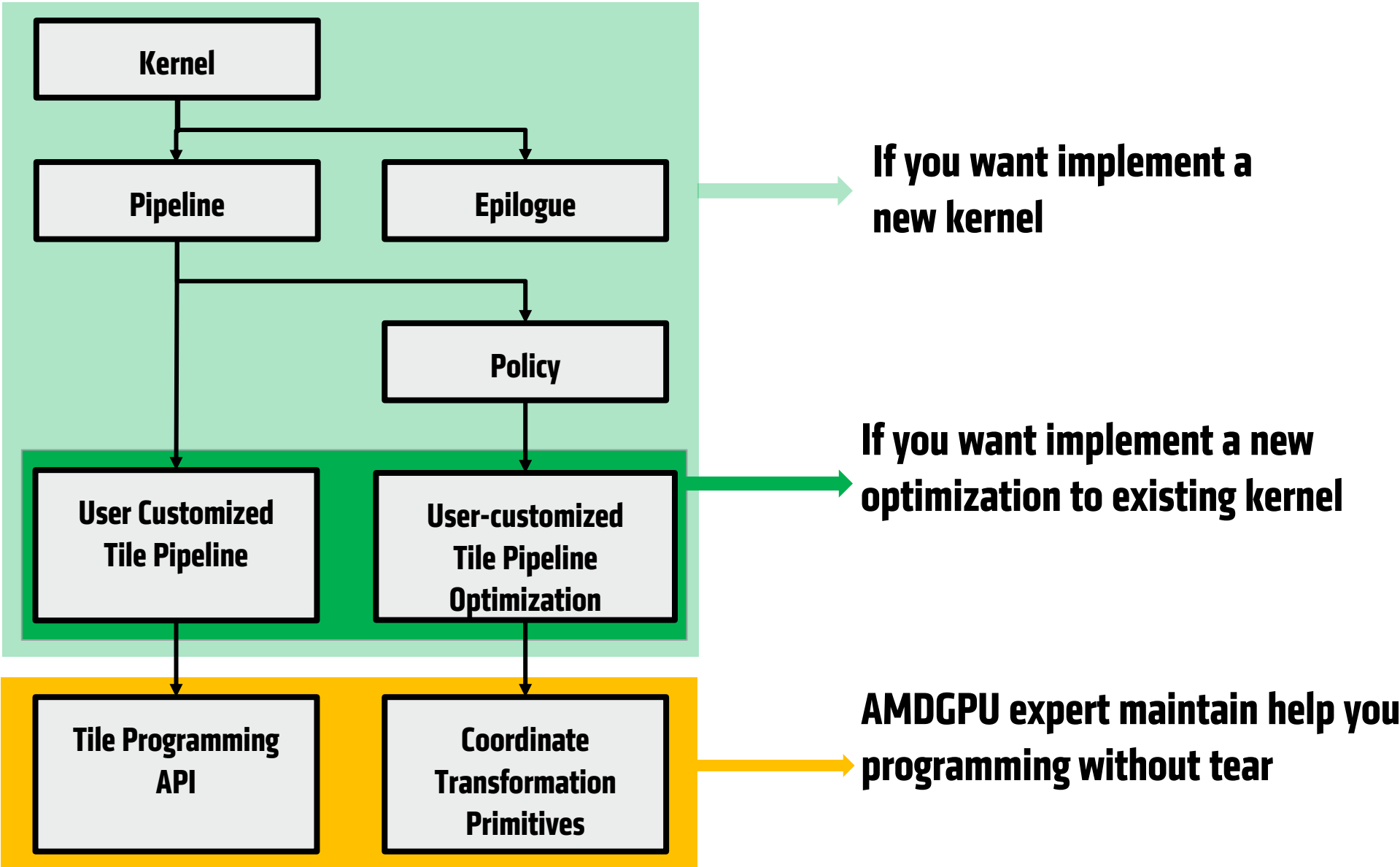


# User customized kernel





# User customized kernel



**ROCm Flash-Attention**

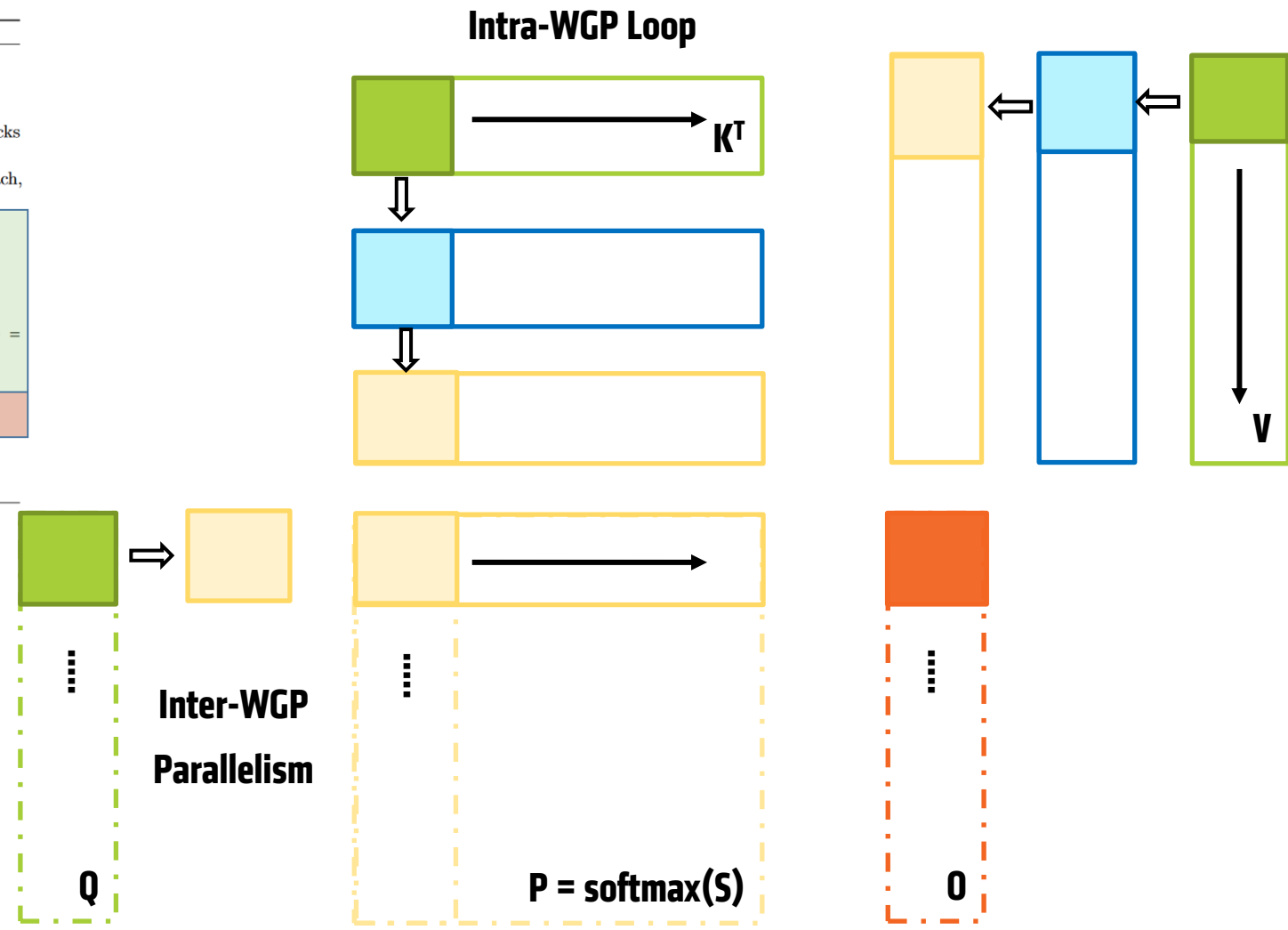
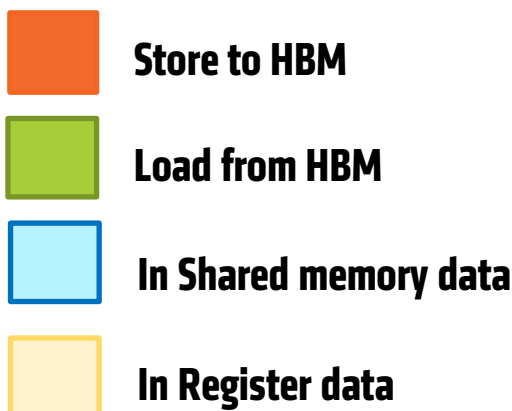


# ROCm Flash-Attention

Algorithm 1 FLASHATTENTION

**Require:** Matrices  $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$  in HBM, on-chip SRAM of size  $M$ .

- 1: Set block sizes  $B_c = \lceil \frac{M}{4d} \rceil$ ,  $B_r = \min(\lceil \frac{M}{4d} \rceil, d)$ .
- 2: Initialize  $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}$ ,  $\ell = (0)_N \in \mathbb{R}^N$ ,  $m = (-\infty)_N \in \mathbb{R}^N$  in HBM.
- 3: Divide  $\mathbf{Q}$  into  $T_r = \lceil \frac{N}{B_r} \rceil$  blocks  $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$  of size  $B_r \times d$  each, and divide  $\mathbf{K}, \mathbf{V}$  into  $T_c = \lceil \frac{N}{B_c} \rceil$  blocks  $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$  and  $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$ , of size  $B_c \times d$  each.
- 4: Divide  $\mathbf{O}$  into  $T_r$  blocks  $\mathbf{O}_1, \dots, \mathbf{O}_{T_r}$  of size  $B_r \times d$  each, divide  $\ell$  into  $T_r$  blocks  $\ell_1, \dots, \ell_{T_r}$  of size  $B_r$  each, divide  $m$  into  $T_r$  blocks  $m_1, \dots, m_{T_r}$  of size  $B_r$  each.
- 5: **for**  $1 \leq j \leq T_c$  **do**
- 6:   Load  $\mathbf{K}_j, \mathbf{V}_j$  from HBM to on-chip SRAM.
- 7:   **for**  $1 \leq i \leq T_r$  **do**
- 8:     Load  $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$  from HBM to on-chip SRAM.
- 9:     On chip, compute  $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$ .
- 10:    On chip, compute  $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$  (pointwise),  $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$ .
- 11:    On chip, compute  $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$ ,  $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$ .
- 12:    Write  $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$  to HBM.
- 13:    Write  $\ell_i \leftarrow \ell_i^{\text{new}}$ ,  $m_i \leftarrow m_i^{\text{new}}$  to HBM.
- 14:   **end for**
- 15: **end for**
- 16: Return  $\mathbf{O}$ .



# Impact

AMD Community > Blogs > AI > AMD Composable Kernel library: efficient fused ker...

AMD Composable Kernel library: efficient fused kernels for AI apps with just a few lines of code

[Liu Chao](#)  
Staff

 6  0  14K

10-26-2022 02:42 PM

CLASS xformers.ops.fmha.ck.FwOp [SOURCE]

xFormers' MHA kernel based on Composable Kernel.

CLASS xformers.ops.fmha.ck.BwOp [SOURCE]

xFormers' MHA kernel based on Composable Kernel.

ECOSYSTEM

by [Abhi Venigalla](#)  
on June 30, 2023

SHARE    

## Training LLMs with AMD MI250 GPUs and MosaicML


With the release of PyTorch 2.0 and ROCm 5.4, we are excited to announce that LLM training works out of the box on AMD MI250 accelerators with zero code changes and at high performance!


```
pytorch / torch / _inductor / config.py

Code Blame 1024 lines (787 loc) · 37.5 KB Raw Copy Download Edit

254 # Specify candidate backends for gemm autotune.
255 # Possible choices are combinations of: ATen, Triton, CUTLASS, CK, CPP.
256 # ATen: default Pytorch ATen kernels.
257 # Triton: Triton templates defined in torch inductor (AMD and NVidia GPUs).
258 # CUTLASS: Cutlass templates and kernels (NVidia GPUs only).
259 # CK: Composable Kernel templates and kernels (AMD Instinct GPUs only).
260 # CPP: CPP templates and kernels for CPU.
```

## Support AMD ROCm on FlashAttention 2 #1010

 Open rocking5566 wants to merge 34 commits into Dao-AILab:main from ROCm:ck\_tile



[HOME](#) [SOFTWARE](#) [DOCS](#)

[Home » News » Announcement](#)

### AMD support for Microsoft® DirectML optimization of Stable Diffusion

May 23, 2023

AMD Community > Blogs > Instinct Accelerators > Competitive performance claims and industry le

Competitive performance claims and industry leading Inference performance on AMD Instinct MI300X

[Ian Ferreira](#)  
Staff

 10  0  30.5K

## Accelerating Stable Diffusion Inference with ONNX Runtime

[Tianlei Wu](#) · Follow

Published in Microsoft Azure · 4 min read · May 10

# Reference

1. Dao, Tri, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness.” arXiv, June 23, 2022. <https://doi.org/10.48550/arXiv.2205.14135>.

**To get access of Composable Kernel:**

[https://github.com/ROCm/composable\\_kernel](https://github.com/ROCm/composable_kernel)

**For Composable Kernel document:**

[https://rocm.docs.amd.com/projects/composable\\_kernel/en/latest/](https://rocm.docs.amd.com/projects/composable_kernel/en/latest/)

**AMDGPU architecture document:**

<https://gpuopen.com/amd-gpu-architecture-programming-documentation/>



**Q&A**

# COPYRIGHT AND DISCLAIMER

©2024 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, [\[insert all other AMD trademarks used in the material IN ALPHABETICAL ORDER here per AMD's Guidelines on Using Trademark Notice and Attribution\]](#) and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

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 releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. 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.

