



Developing Optimal CUDA Kernels on Hopper Tensor Cores

Pradeep Ramani, Cris Cecka | March 22, 2023

CUTLASS

CUDA C++ Template Library for Deep Learning and High Performance Computing

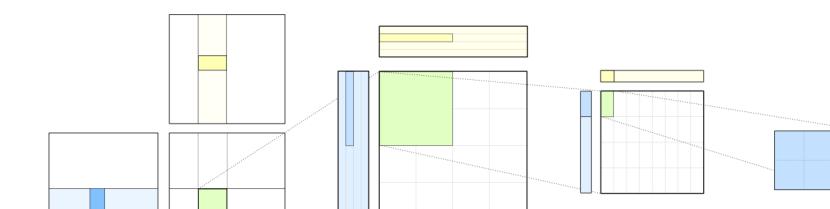
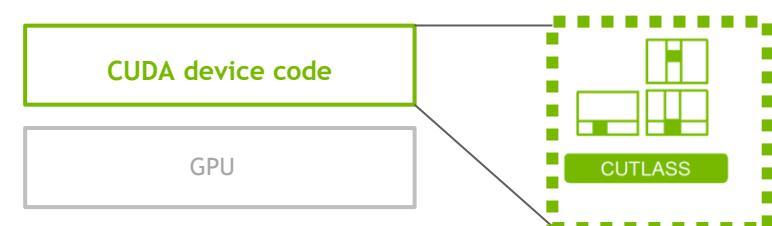


CUTLASS: optimal CUDA C++ templates for matrix computations at all scopes and scales

Device	{ GEMM, Convolution, Reductions , BLAS3 } x { all data types } x { SIMT, Tensor Cores } x { all architectures }
Kernel	GEMM, Batched GEMM, Convolution, Reduction, Fused output operations, Fused input operations
Collective	Pipelined Matrix Multiply, Epilogue, Collective access to tensors, Convolution matrix access
Atom	Tensor Core Multiply-Add operations, Efficient access to permuted tensor layouts
Thread	Numeric conversion, <functional> operators on arrays, complex<T>, fast math algorithms
Architecture intrinsic	Templates wrapping architecture-specific PTX instructions (e.g. mma, cp.async, ldmatrix, cvt)

Open source: <https://github.com/NVIDIA/cutlass> (new BSD license)

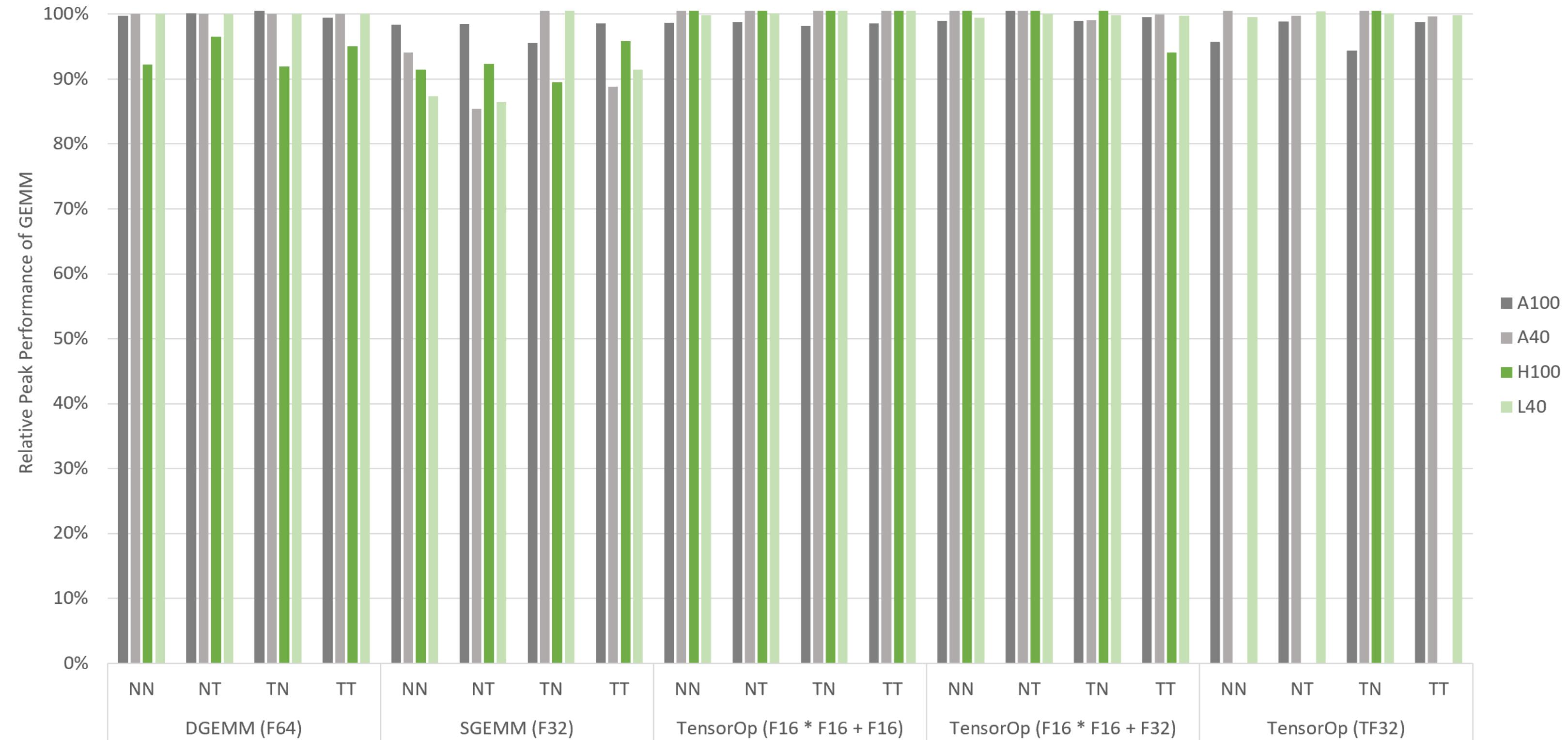
- Latest revision: CUTLASS 3.0
- Documentation: <https://github.com/NVIDIA/cutlass#documentation>
- Functionality: <https://github.com/NVIDIA/cutlass/blob/master/media/docs/functionality.md>
- Presented: [GTC'18](#), [GTC'19](#), [GTC'20](#), [GTC'21](#), [GTC'22](#) , [GTC'22](#)



TENSOR CORE PERFORMANCE ON NVIDIA HOPPER

CUTLASS 3.0 - CUDA 12.0 Toolkit - NVIDIA H100

CUTLASS 3.0 - Relative Peak Performance - GEMM
CUDA 12.0



Roadmap

Subject to change

	Q1 - 2023	Q2 - 2023	Q3 - 2023	Q4 - 2023
CUTLASS v2.x	<ul style="list-style-type: none">• Security bugs	<ul style="list-style-type: none">• Security bugs	<ul style="list-style-type: none">• Security bugs	<ul style="list-style-type: none">• Security bugs
CUTLASS v3.x	<ul style="list-style-type: none">• CuTe Integration• Hopper GEMMs• Batched GEMMs• Bias, Broadcast Fusions• v3.0 Transition Blog	<ul style="list-style-type: none">• Efficient Epilogues• Elementwise Fusions• Permute Fusions• Ptr-array GEMM• Stream-K• 2D CONV (Fprop, D/Wgrad)• fMHA (Backward Pass)*	<ul style="list-style-type: none">• Grouped GEMM• Reduction, B2B Fusions• FP16/INT8 FW fMHA• FP64 GEMM• Complex TF32 GEMM	<ul style="list-style-type: none">• Documentation Refresh• BLAS 3• 3D/4D• Sparsity• Grouped CONV• Separable CONV
Python (v3.x)	<ul style="list-style-type: none">• Prototyping and API design• Demos with stakeholders• Device-layer GEMMs w/ Epilogues• Jupyter Notebooks	<ul style="list-style-type: none">• Device-layer CONV w/ Epilogues• Jupyter Notebooks	<ul style="list-style-type: none">• Softmax/B2B Interface• Jupyter Notebooks	

* External contribution



Agenda

- Hopper Architecture
- CuTe
- CUTLASS 3.0
- CUTLASS Python
- Conclusion

Acknowledgements

CUTLASS GitHub Community

2.5K stars, 1.6M clones/month, 70+ contributors, and many active users.

Many contributions and PRs from outside of NVIDIA

Integrated into PyTorch, Oneflow, TVM, PaddlePaddle, AI Template, PyG, and 300 others github projects

CUTLASS Developers

Vijay Thakkar, Pradeep Ramani, Cris Cecka, Aniket Shivam, Jack Kosaian, Mark Hoemmen, Honghao Lu, Ethan Yan, Richard Cai, Haicheng Wu, Andrew Kerr, Dustyn Blasig, Fengqi Qiao, Duane Merrill, Yujia Zhai, Shang Zhang, Piotr Majcher, Paul Springer, Markus Hohnerbach, Jin Wang, Aditya Atluri

CuTe Developers

Cris Cecka, Vijay Thakkar

CUTLASS Product Management

Matthew Nicely

Contributors & Acknowledgements

Manish Gupta, Naila Farooqui, David Tanner, Manikandan Ananth, Zhaodong Chen, Chinmay Talegaonkar, Timothy Costa, Julien Demouth, Brian Fahs, Michael Garland, Michael Goldfarb, Mostafa Hagog, Fei Hu, Alan Kaatz, Tina Li, Timmy Liu, Wei Liu, Kevin Siu, Markus Tavenrath, John Tran, Vicki Wang, Junkai Wu, Fung Xie, Albert Xu, Yang Xu, Jack Yang, Scott Yokim, Xiuxia Zhang, Nick Zhao, Girish Bharambe, Luke Durant, Carter Edwards, Olivier Giroux, Stephen Jones, Rishkul Kulkarni, BalajiKrishna Atukuri, Bryce Lelbach, Joel McCormack, Kyrylo Perelygin, Sean Treichler

NVIDIA Hopper Architecture

NVIDIA - H100

New Faster Tensor Core Instructions

- 16b Floating-point Tensor Core operations **16x** and **32x** faster than F32 CUDA Cores
- 8b Floating-point Tensor Core operations **32x** and **64x** faster than F32 CUDA Cores
- Improved Integer Tensor Core operations **32x** and **64x** faster than F32 CUDA Cores
- Improved Tensor Float 32 Tensor Cores **8x** and **16x** faster than F32 CUDA Cores
- Improved IEEE double-precision Tensor Cores **2x** faster than F64 CUDA Cores

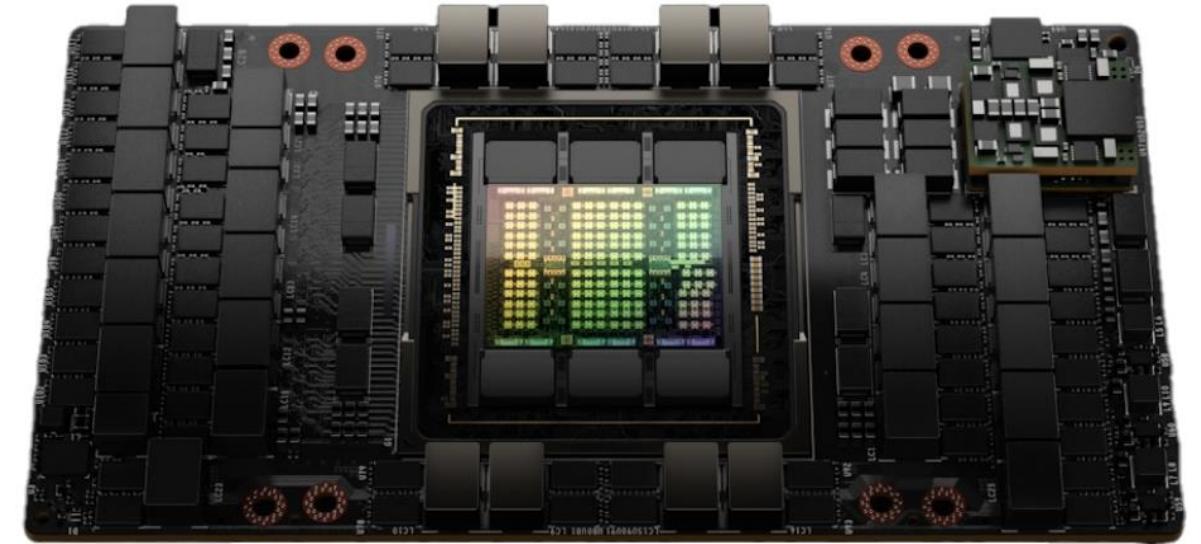


New Hierarchy of thread grouping – Thread Block Clusters

- Helps with optimal sharing of data amongst Thread Blocks

Additional Data Types

- 8bit Floating point types E5M2 and E4M3



Asynchronous Copy using TMA

- Perform address compute for global loads and swizzled shared memory stores with bounds check.
- Ability to multicast data across the Thread Block Cluster

Many additional new features – see "[NVIDIA H100 Tensor Core GPU Architecture](#)" whitepaper

NVIDIA HOPPER ARCHITECTURE - TENSOR CORE OPERATIONS

Operation	Data Types (A * B + C)	Shape	TFLOPS Hopper (H100)	TFLOPS Ampere (A100)	TFLOPS Volta (V100)
MMA wgmma, mma_sync	F16 * F16 + F16	64-by-N-by-16	1000	312	125
	F16 * F16 + F32				
	BF16 * BF16 + F32				
MMA wgmma, mma_sync	TF32 * TF32 + F32	64-by-N-by-8	500	156	N/A
MMA mma_sync	F64 * F64 + F64	16-by-8-by-4/8/16	60	19	N/A
MMA wgmma, mma_sync	S8 * S8 + S32	64-by-N-by-32	2000	624	N/A
MMA wgmma, mma_sync	F8 * F8 + F32	64-by-N-by-32	2000	N/A	N/A

<https://docs.nvidia.com/cuda/parallel-thread-execution/index.html#asynchronous-warpgroup-level-matrix-instructions>

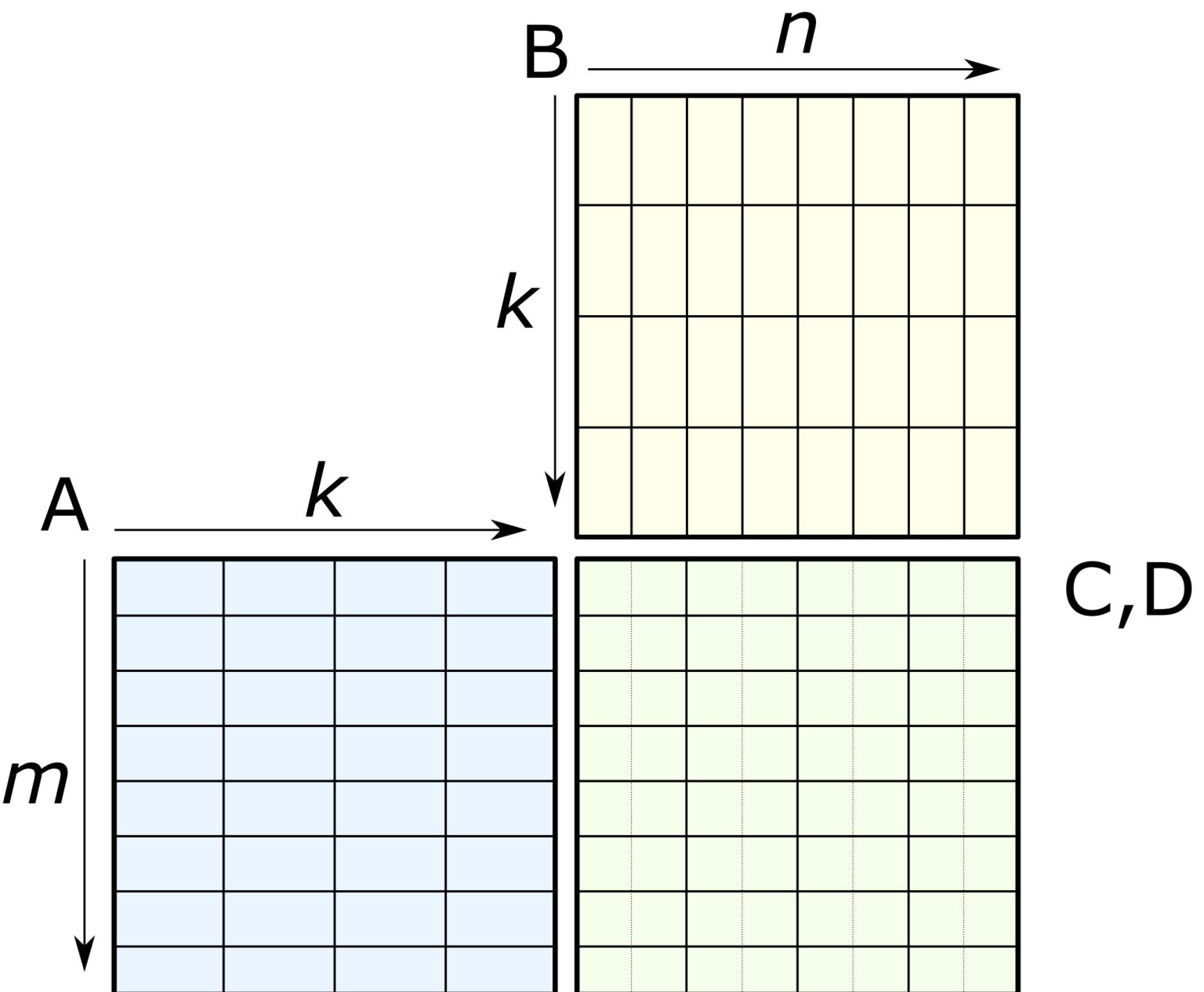
TENSOR CORES OPERATION : FUNDAMENTAL SHAPE

Matrix operations: $D = \text{op}(A, B) + D$

- Matrix multiply-add
- XOR-POPC

M -by- N -by- K matrix operation

- Warp-Group-wide, async. collective operation.
- 128 threads within warp-group collectively holds D operand.
- Operands A and B (not both) can optional be loaded from register memory. Loading directly from shared memory using descriptors most optimal.
- Valid value(s) of $M = 64$; $N \in [8, 256]$ and $K \in [8, 16, 32, 256]$ based on data type refer [ISA documentation](#) for details



F16 * F16 + F32

64-by-N-by-16

- **wmma.mma_async** issues a warp-group wide asynchronous MxNxK matrix multiply and accumulate operation,

$$D = A * B + D$$

Where A matrix is MxK, B matrix is KxN, and D matrix is MxN.

- Where $N \in [8, 256]$, and $M = 64$.

- A and B matrices are loaded directly from Shared Memory using descriptors.
- Transposition and scaling with certain immediate values supported as part of the instruction.

```
uint64_t descriptor_a, descriptor_b;
float D[64];
constexpr int scale_a, scale_b;           // -1 or +1
constexpr uint32_t scale_d;               // 0 or 1
constexpr uint32_t trans_a, trans_b; // 0 or 1

asm (
    "wgmma.mma_async.sync.aligned.m64n64k16.f32.f16."
f16 "
    "%0, %1, %2, %3, %4, %5, %6, %7,
    %8, %9, %10, %11, %12, %13, %14, %15,
    ...
    ...
    %56, %57, %58, %59, %60, %61, %62, %63},
    %64,
    %65,
    %66, %67, %68, %69, %70;\n"
: "+f"(D[0]), "+f"(D[1]), "+f"(D[2]), "+f"(D[3]),
  "+f"(D[4]), "+f"(D[5]), "+f"(D[6]), "+f"(D[7]),
  ...
  ...
  "+f"(D[60]), "+f"(D[61]), "+f"(D[62]), "+f"(D[63])
:
    "l"(descriptor_a), "l"(descriptor_b),
    "n"(scale_d), "n"(scale_a), "n"(scale_b),
    "n"(trans_a), "n"(trans_b)
);
```

Ref. [cute::SM90_64x128x16_F32F16F16_SS](#)

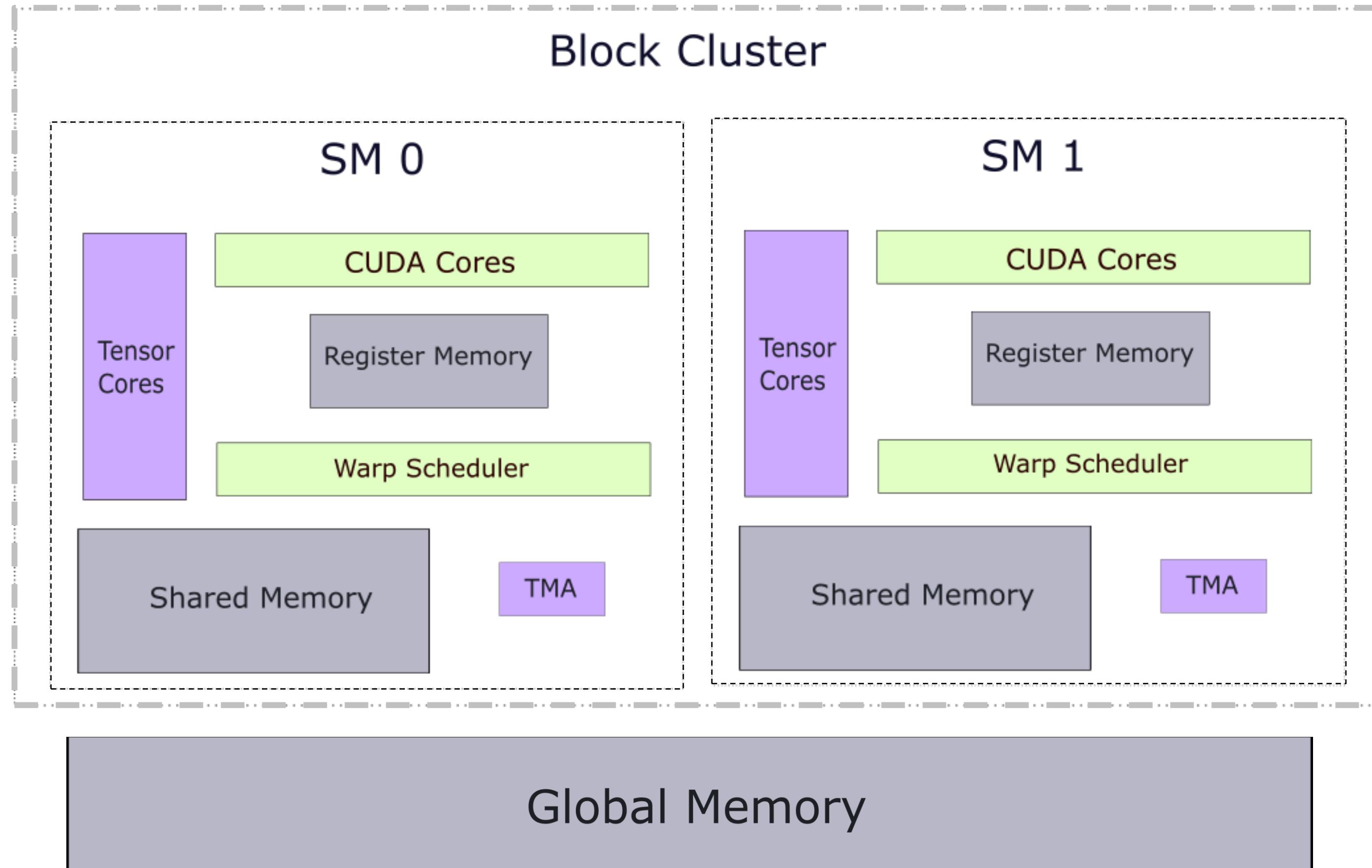
A new way to move data !

Async Copy using TMA

- A new mechanism to issue asynchronous copies from / to global memory to / from shared memory.
- Avoids extra roundtrip of going through register file, and instead directly writes / reads shared memory.
- Can Multicast data to other threadblocks in a cluster and update async barriers.
- Performs bounds checking on data and automatically zero fills into shared memory if out of bounds
- Capable of writing and reading data in several different swizzled formats in Shared needed
- Several flavors supported by Hopper :
 - 1D-5D Tensor Copy between Global and Shared Memory.
 - Two modes, TILED and IM2COL (useful for convolutions)
 - Variants which support reductions
 - Variants which support simplified bulk copies
- Information regarding the tensor which being copied to / from in global memory is passed to using descriptors, which are created using [driver APIs](#).

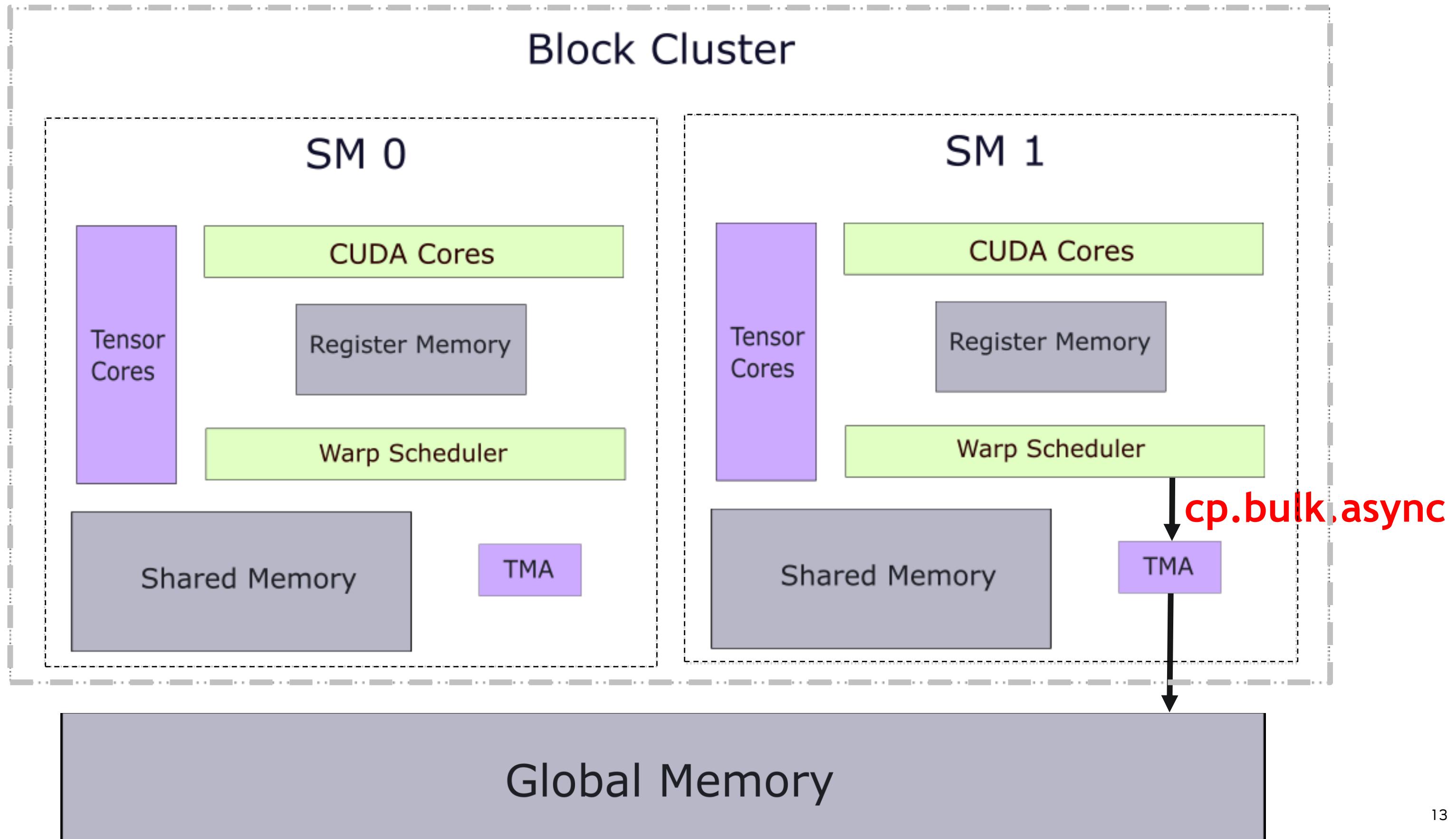
A new way to move data !

TMA Multicast



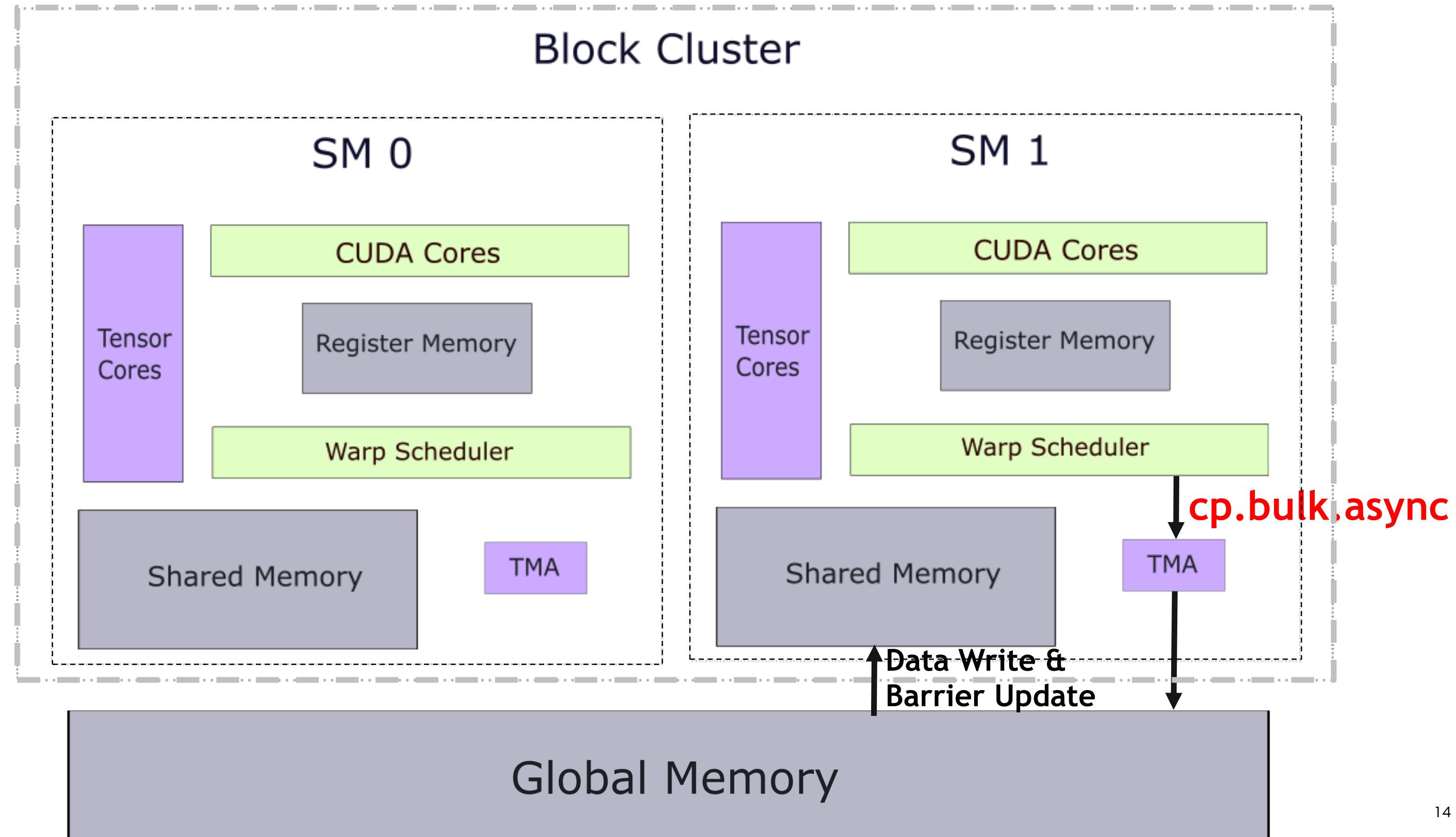
A new way to move data !

TMA Multicast



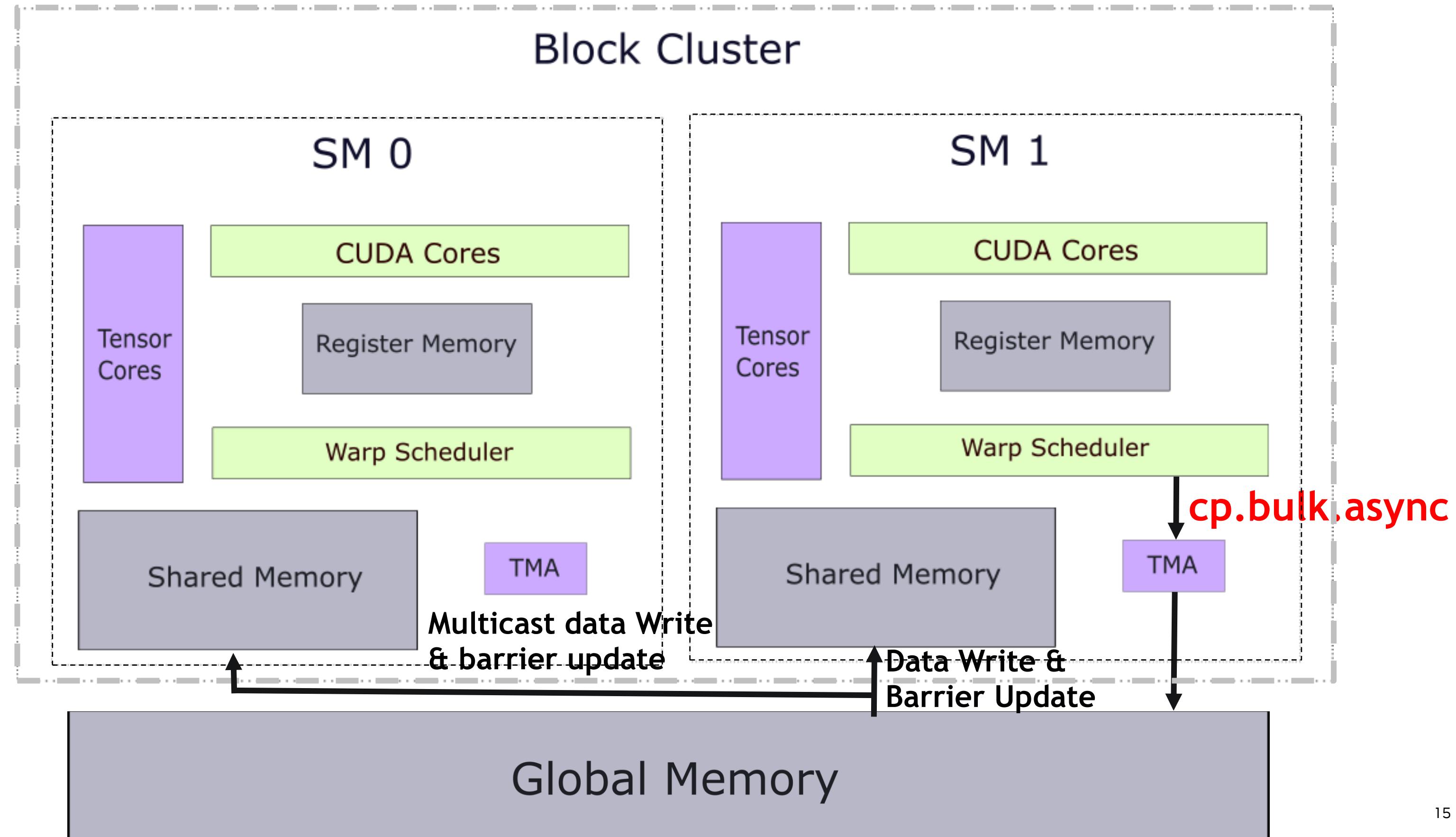
A new way to move data !

TMA Multicast



A new way to move data !

TMA Multicast



copy.async.bulk

2D, tiled mode, multicast

- **Copy.async.bulk** family of instructions issues a warp uniform asynchronous copy operation.
- Copy can be Shared -> Global (or) Global -> Shared
- Multicast mask specifies which CTAs in the Cluster also need this data to be sent to them (applies only for Global -> Shared).
- Descriptors are created on the host using CUDA driver APIs [cuTensorMapEncode*](#) and passed to the device as grid private constants.

```
uint32_t smem_data_ptr;
uint64_t gmem_data_desc;
uint32_t smem_mbar_ptr;
uint16_t multicast_mask,
int32_t crd0;
int32_t crd1;

asm
(
    "cp.async.bulk.tensor.2d.shared::cluster.global.mbarrier::complete_tx::bytes.multicast::cluster"
    " [%0], [%1, {%4, %5}], [%2], %3;"
    :
    :
    "r"(smem_data_ptr),
    "l"(gmem_data_desc),
    "r"(smem_mbar_ptr),
    "h"(multicast_mask),
    "r"(crd0), "r"(crd1)
);
```

Ref. [cute::SM90_TMA_LOAD_2D_MULTICAST](#)



Agenda

- Hopper Architecture
- CuTe
- CUTLASS 3.0
- CUTLASS Python
- Conclusion

What is CuTe?

- CuTe is for CUDA Tensors
- CuTe provides **Layout** and **Tensor** types
 - Compactly package the **type**, **shape**, memory **space**, and **layout** of data
 - Perform the complicated indexing for the user
 - Abstractions for defining and operating on hierarchically multidimensional layouts
 - Shapes and strides can be fully static, dynamic, or mixed
- A formal algebra over **Layout**
 - Layouts can be combined and manipulated
 - Build complicated layouts from simple arch invariant primitive layouts
 - Partition layouts across other layouts to preserve logical consistency at every level

Why is CuTe?

Layouts and Tensors EVERYWHERE

- Layout subsumes every CUTLASS-2 iterator
 - Condense complexity to a single impl.
- Formal algebra to manipulate Layout
 - composition
 - complement
 - right_inverse, left_inverse
 - “product”
 - “divide”
- Layout for both threads and data

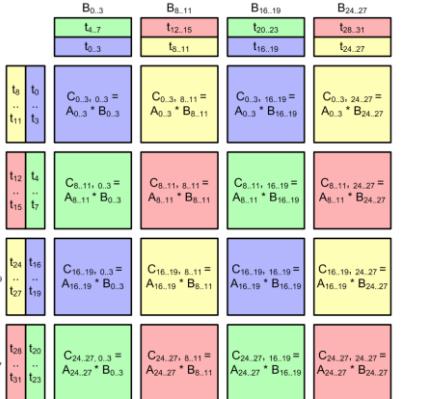


Figure 2a: Composing sparse $A[16,4] * B[4,16] = C[16,16]$
warp-wide multiply from four $A[8,4] * B[4,8] = C[8,8]$ 8-thread multiplies.

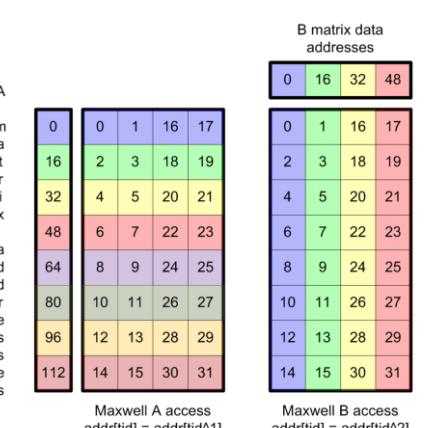


Figure 3b: Maxwell thread assignments for LDS.U matching

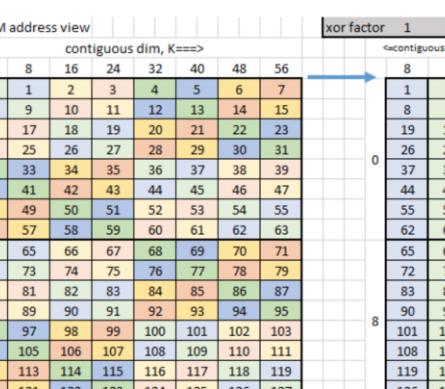


Figure 1b: HMMA.884.F32.TN data layout.
Again, ignore the apparent isomorphism
between thread numbers and matrix indices.

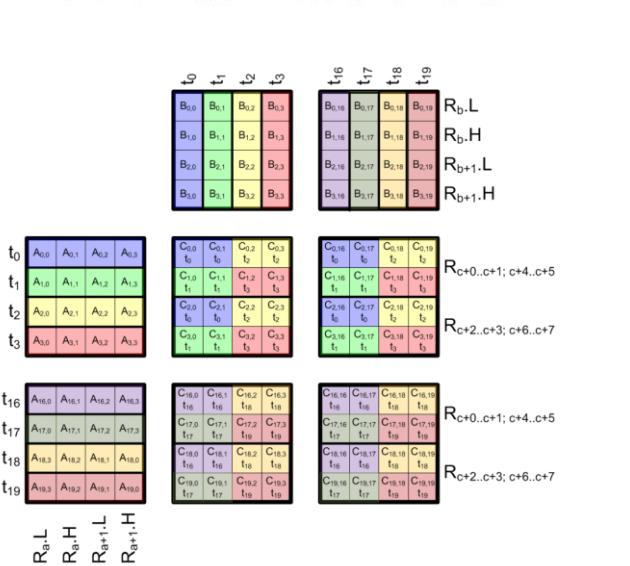
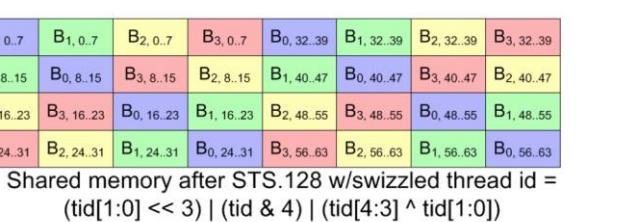
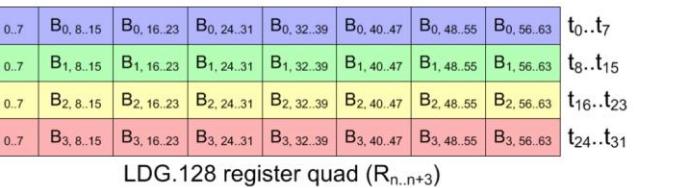


Figure 1b: HMMA.884.F32.TN data layout.
Again, ignore the apparent isomorphism
between thread numbers and matrix indices.

CUTLASS 2.x

RowMajor

ColumnMajor

RowMajorInterleaved

ColumnMajorInterleaved

PitchLinear

TensorNCHW

VoltaTensorOpMultiplicandCongruous

ColumnMajorVoltaTensorOpMultiplicandCongruous

RowMajorVoltaTensorOpMultiplicandCongruous

VoltaTensorOpMultiplicandBCongruous

ColumnMajorVoltaTensorOpMultiplicandBCongruous

RowMajorVoltaTensorOpMultiplicandBCongruous

VoltaTensorOpMultiplicandCrosswise

ColumnMajorVoltaTensorOpMultiplicandCrosswise

RowMajorVoltaTensorOpMultiplicandCrosswise

Many, many more ...

tridao commented 3 weeks ago

Author



...

Amazing! Thanks a lot for the explanation @thakkarV.

I've always found smem swizzling to be the trickiest part of writing a fast gemm. It's crazy that you can replace thousands of lines of smem swizzling code with a few lines of well-designed and composable abstractions.

I'm excited to start hacking on CuTe!



1

CUTLASS 3.x



Layout<Shape, Stride>



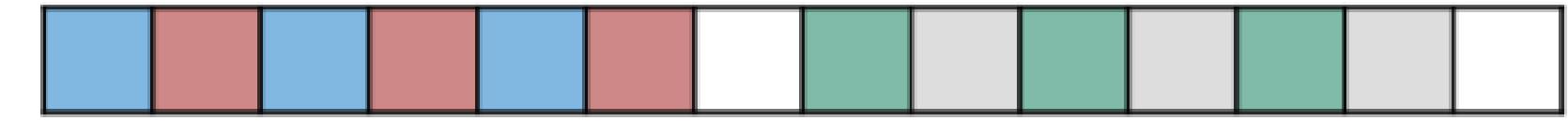
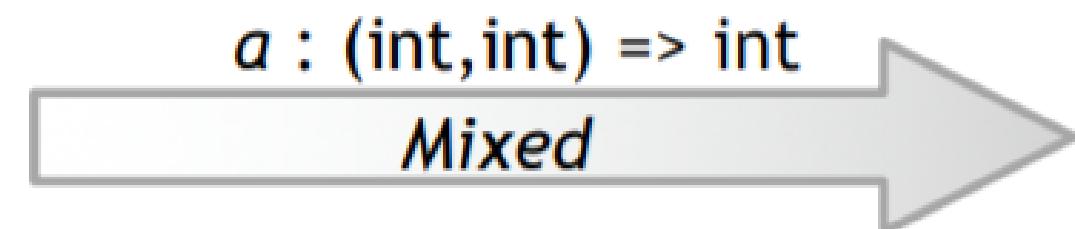
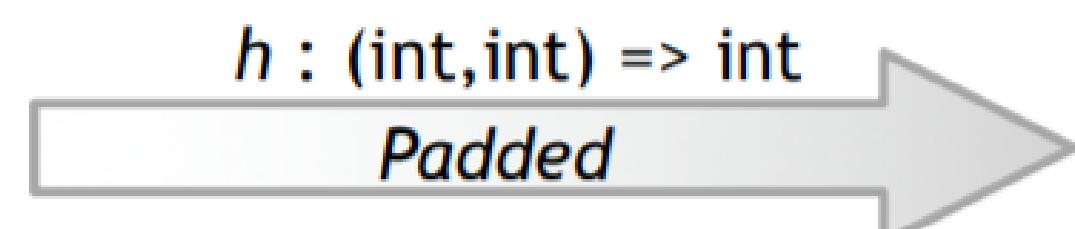
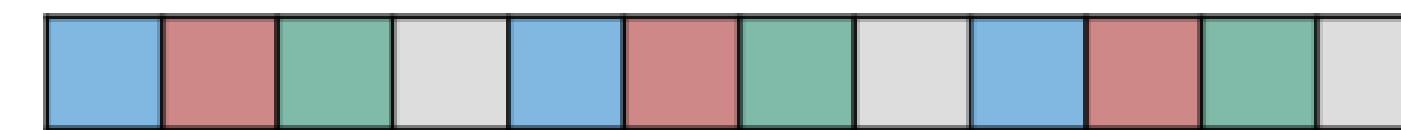
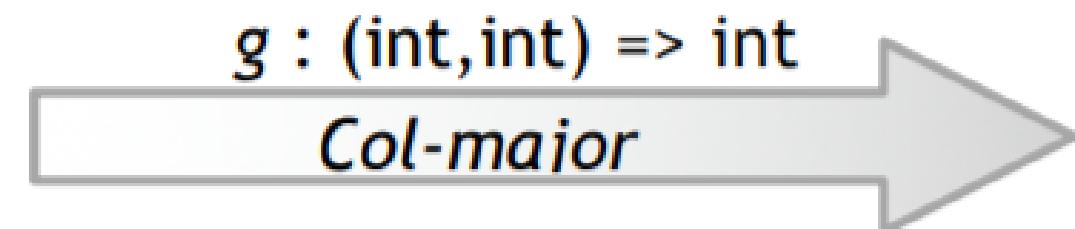
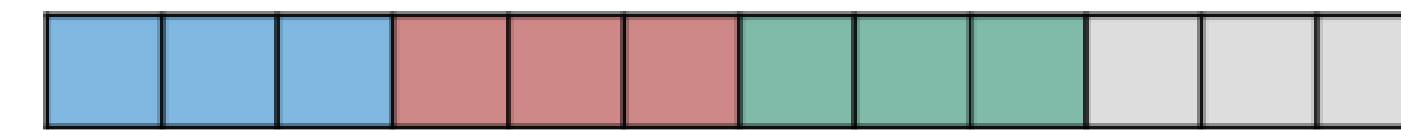
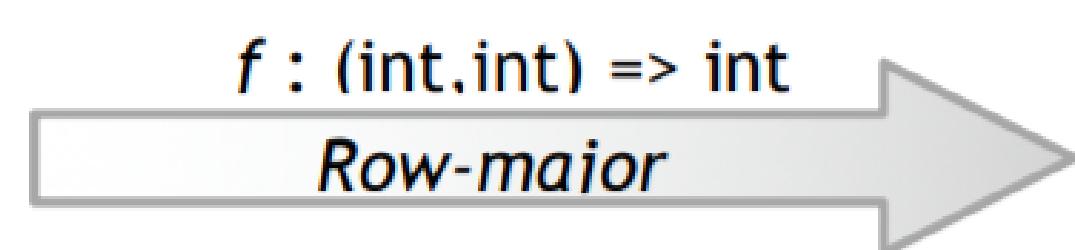
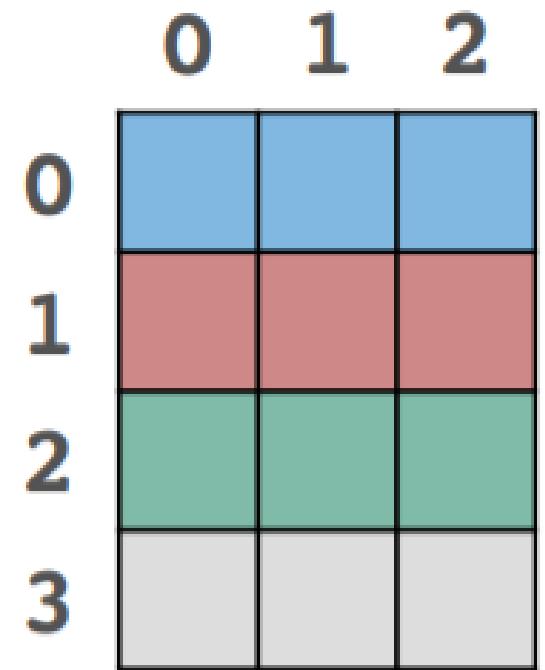
CuTe in CUTLASS-3.0

- Layouts

- Algebra

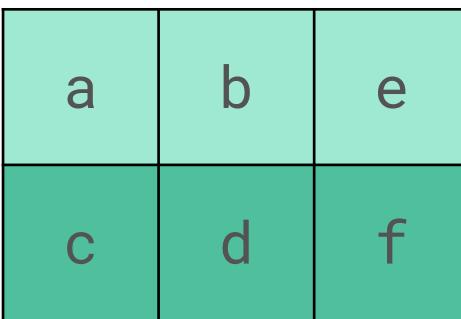
- Tensor Cores

Layouts Map Coordinates to Storage

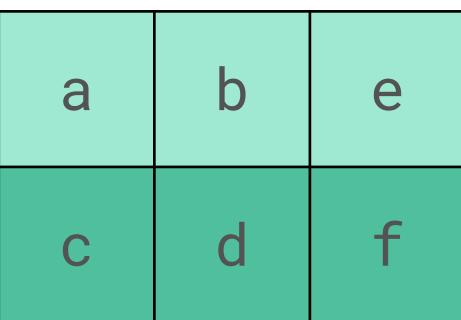
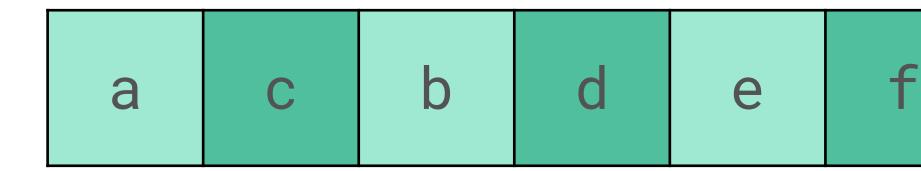


Layout Representation

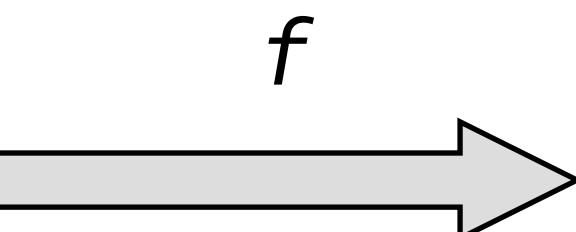
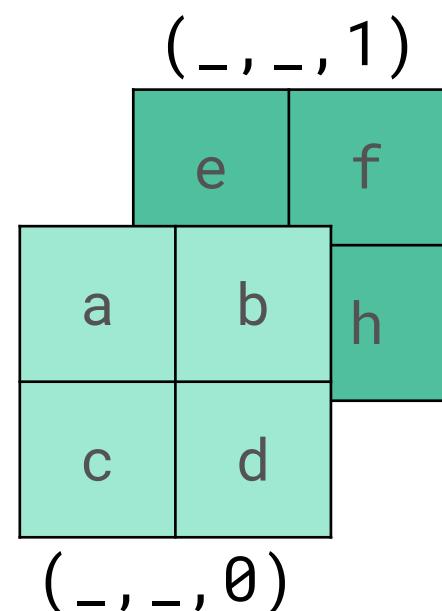
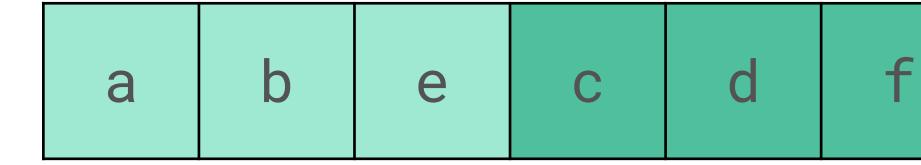
Shapes and Strides



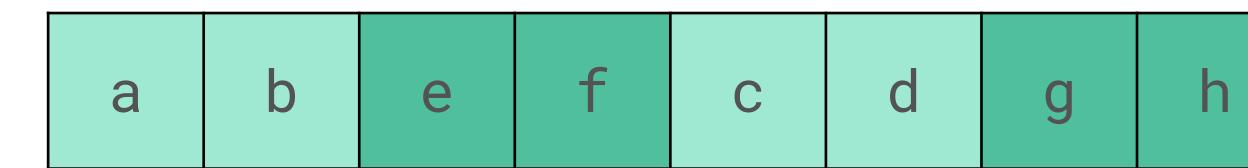
Shape: (2, 3)
Stride: (1, 2)



Shape: (2, 3)
Stride: (3, 1)



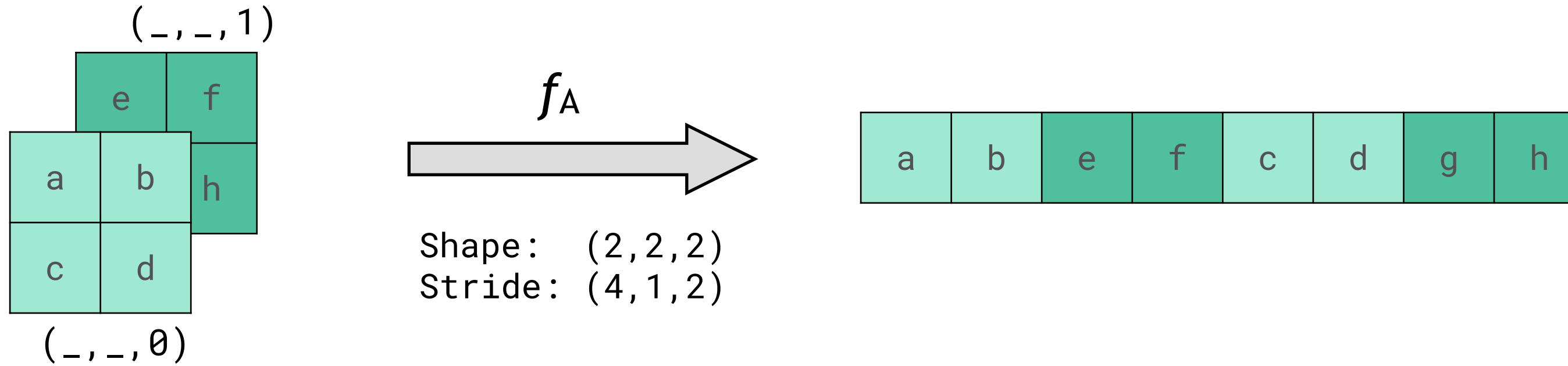
Shape: (2, 2, 2)
Stride: (4, 1, 2)



$f(\text{coord}) = \text{inner_product}(\text{coord}, \text{stride})$

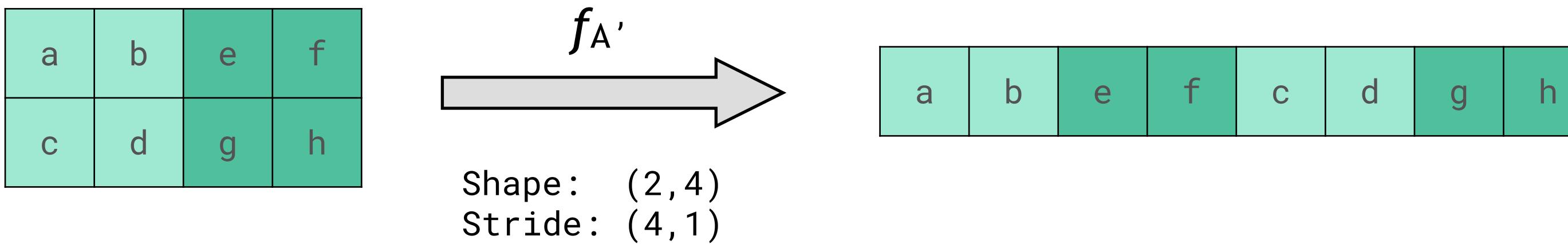
Hierarchical Layouts

Tensors and Folded Modes



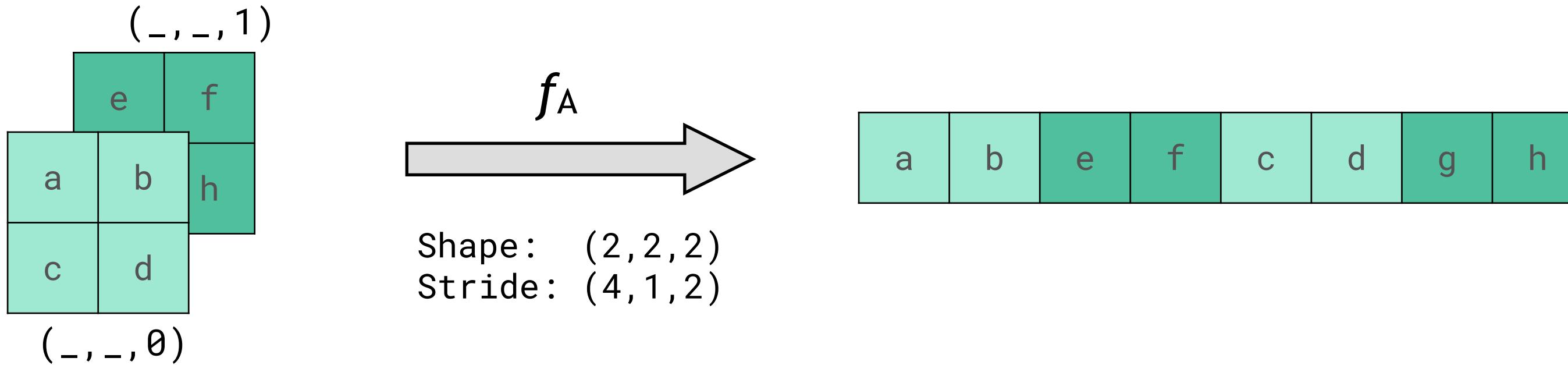
Which can be folded and viewed as a matrix

$f(\text{coord}) = \text{inner_product}(\text{coord}, \text{stride})$



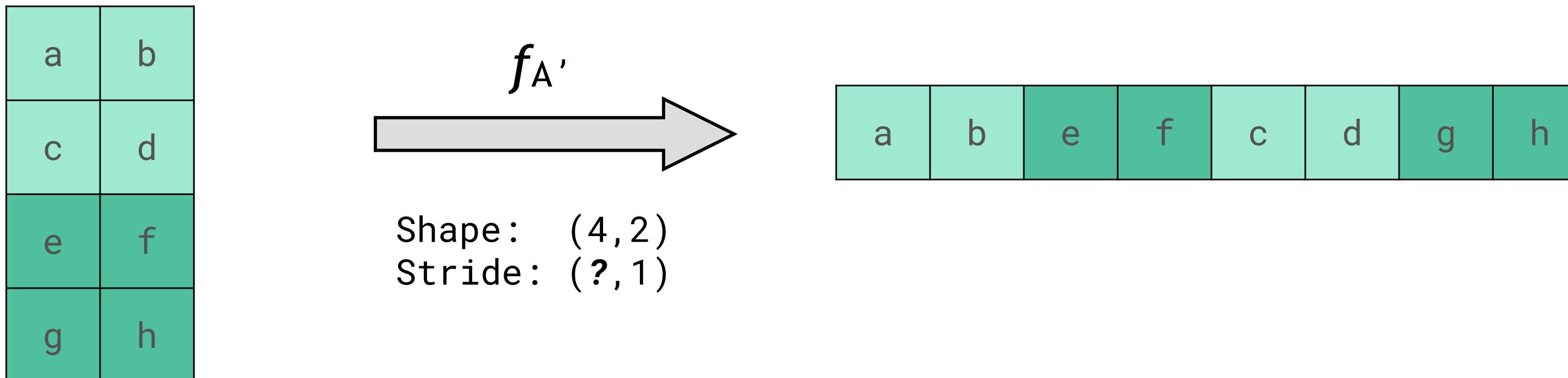
Hierarchical Layouts

Tensors and Folded Modes



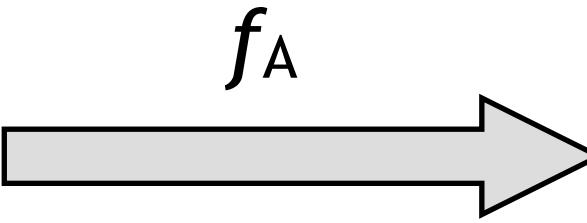
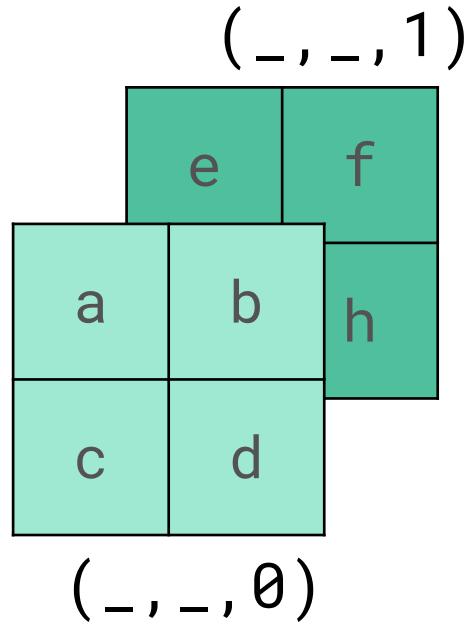
Which can be folded and viewed as a matrix

$$f(\text{coord}) = \text{inner_product}(\text{coord}, \text{stride})$$

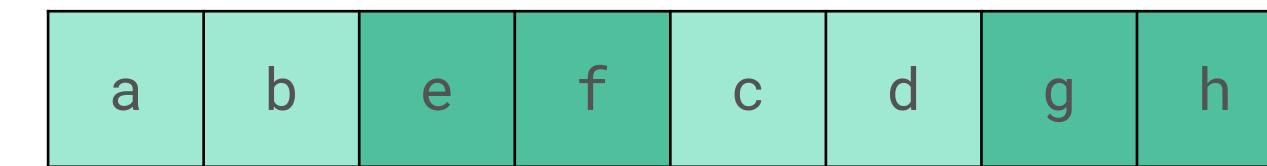


Hierarchical Layouts

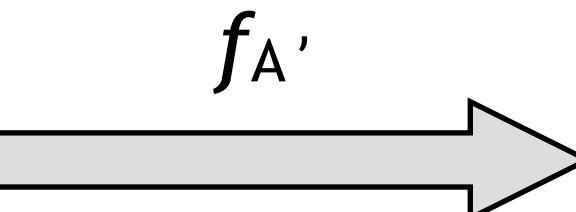
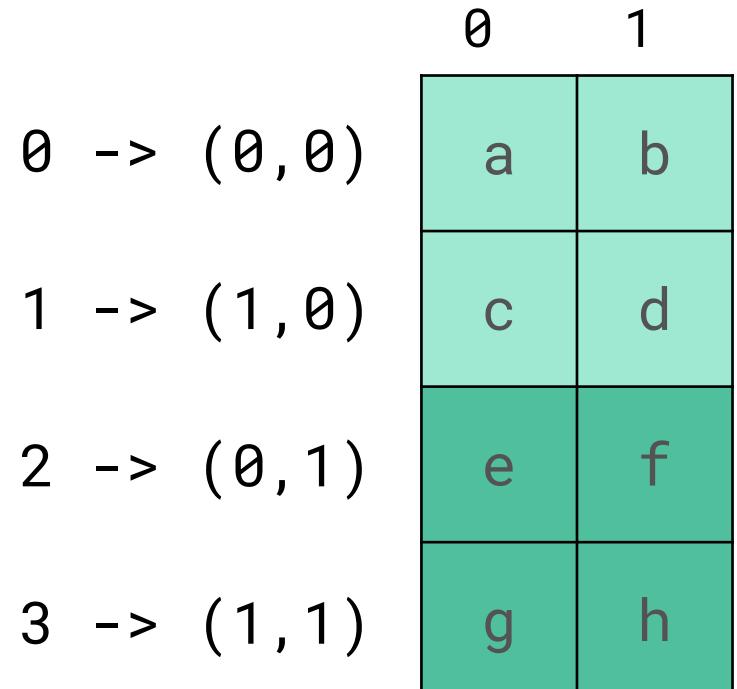
Tensors and Folded Modes



Shape: (2, 2, 2)
Stride: (4, 1, 2)

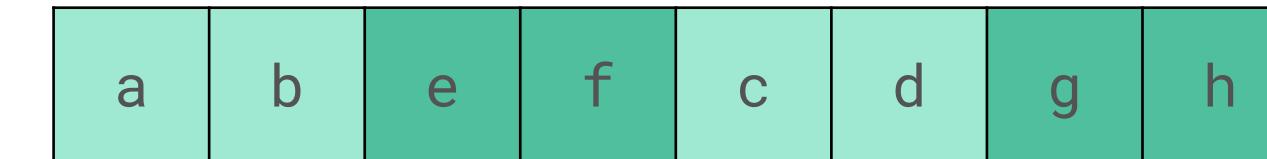


Which can be folded and viewed as a nested shape



Shape: ((2, 2), 2)
Stride: ((4, 2), 1)

$f(\text{coord}) = \text{inner_product}(\text{coord}, \text{stride})$



Key design ingredients

CuTe Tensors

Shape

- Concept `IntTuple`: int or tuple of `IntTuples`.

N (N, M) (N, M, P) $((N_1, N_2), N_3)$ $((N_a, N_b), (N_2, (N_3, N_4, N_5)))$

Stride

- Concept `DTuple`: D or tuple of `DTuples`.

Layout<Shape, Stride>

- A map between n-D logical coordinates and 1-D index coordinates and inverse (if available)

$$(I) \Leftrightarrow (i, j) \Leftrightarrow (i, (j_1, j_2)) \leftrightarrow [k]$$

Tensor<Ptr, Layout>

- Composition of Layout with underlying random access iterator.

`T[], T*, smem_ptr<T>, gmem_ptr<T>, random_access_iterator`

Algebra of Layouts:

- `concatenation` (Layouts...) \rightarrow Layout
- `composition` (LayoutA, LayoutB) \rightarrow Layout
- `complement` (Layout, M) \rightarrow Layout
- `right_inverse` (Layout) \rightarrow Layout

Layout Mappings

Logical Coordinates, Linear Index

Shape: $(4, (2, 2))$

Stride: $(2, (1, 8))$

Logical 1-D
coordinate

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15



$A(I)$

Logical n-D
coordinate

0, 0	0, 1	0, 2	0, 3
1, 0	1, 1	1, 2	1, 3
2, 0	2, 1	2, 2	2, 3
3, 0	3, 1	3, 2	3, 3

$A(i, j)$

Logical h-D
coordinate

0, (0, 0)	0, (1, 0)	0, (0, 1)	0, (1, 1)
1, (0, 0)	1, (1, 0)	1, (0, 1)	1, (1, 1)
2, (0, 0)	2, (1, 0)	2, (0, 1)	2, (1, 1)
3, (0, 0)	3, (1, 0)	3, (0, 1)	3, (1, 1)

$A(i, (j_1, j_2))$

Linear 1-D
storage index

0	1	8	9
2	3	10	11
4	5	12	13
6	7	14	15

$A[k]$

Shape defines the *coordinate* mappings

$$(I) \Leftrightarrow (i, j) \Leftrightarrow (i, (j_1, j_2))$$

Stride defines the *index* mapping

$$(i, (j_1, j_2)) \Leftrightarrow [k]$$

Example Layout

More operations

```
shape(A) = ((2, (2, 2)), (2, (2, 2)))
stride(A) = ((1, (4, 16)), (2, (8, 32)))
```

i\j	0	1	2	3	4	5	6	7
0	0	2	8	10	32	34	40	42
1	1	3	9	11	33	35	41	43
2	4	6	12	14	36	38	44	46
3	5	7	13	15	37	39	45	47
4	16	18	24	25	48	50	56	58
5	17	19	26	27	49	51	57	59
6	20	22	28	30	52	54	60	62
7	21	23	29	31	53	55	61	63

```
Layout morton1 = make_layout(Shape<_2,_2>{}, Stride<_1,_2>{})
Layout morton2 = blocked_product(morton1, morton1);
Layout morton3 = blocked_product(morton1, morton2);
Tensor A = make_tensor(counting_iterator<int>{}, morton3);
```

- Shape

size(A) = 64

size<0>(A) = 8

size<1>(A) = 8

- Logical coordinates are 1D, 2D, hD

A(37) = 49



A(5, 4) = 49

A((1, 2), (0, 2)) = 49

A((1, (0, 1)), (0, (0, 1))) = 49

- Slice along logical sub-boundaries



A(_, 2) = [8; 9; 12; 13; 24; 26; 28; 29]



A((_, 1), (_, 2)) = [36, 38;

37, 39]

Layout Transcription Examples

ColumnMajor<> + Padding

0	4	8	12	16
1	5	9	13	17
2	6	10	14	18

(3,5)
(1,4)

Layout<Shape <_3,_5>,
Stride<_1,_4>>

PitchLinear<>

0
3
6
9
1
4
7
10
2
5
8
11

((4,3))
((3,1))

Layout<Shape <Shape <_4,_3>>,
Stride<Stride<_3,_1>>>

ColumnMajorInterleaved<4>

0	1	2	3	16	17	18	19
4	5	6	7	20	21	22	23
8	9	10	11	24	25	26	27
12	13	14	15	28	29	30	31

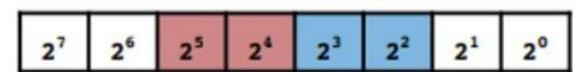
(4,(4, 2))
(4,(1,16))

Layout<Shape <_4,Shape <_4, _2>>,
Stride<_4,Stride<_1,_16>>>

Swizzle Layout Examples

0	1	2	3	20	21	22	23	40	41	42	43	60	61	62	63
4	5	6	7	16	17	18	19	44	45	46	47	56	57	58	59
8	9	10	11	28	29	30	31	32	33	34	35	52	53	54	55
12	13	14	15	24	25	26	27	36	37	38	39	48	49	50	51

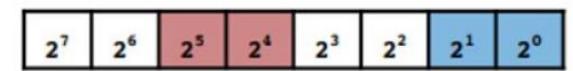
Swizzle<2,2,2>



Layout<Shape <_4,Shape <_4, _4>>,
Stride<_4,Stride<_1,_16>>>

0	4	8	12	17	21	25	29	34	38	42	46	51	55	59	63
1	5	9	13	16	20	24	28	35	39	43	47	50	54	58	62
2	6	10	14	19	23	27	31	32	36	40	44	49	53	57	61
3	7	11	15	18	22	26	30	33	37	41	45	48	52	56	60

Swizzle<2,0,2>



Layout<Shape <_4,_16>,
Stride<_1, _4>>

0	1	2	3	4	5	6	7
9	8	11	10	13	12	15	14
18	19	16	17	22	23	20	21
27	26	25	24	31	30	29	28
36	37	38	39	32	33	34	35
45	44	47	46	41	40	43	42
54	55	52	53	50	51	48	49
63	62	61	60	59	58	57	56

Swizzle<3,0,0>



Layout<Shape <_8,_8>,
Stride<_8,_1>>

Composition Power

Example



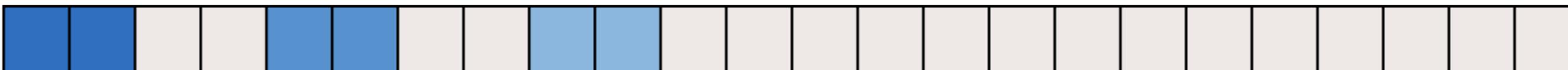
Composition Power

Example



Composition Power

Example



$((2,3))$

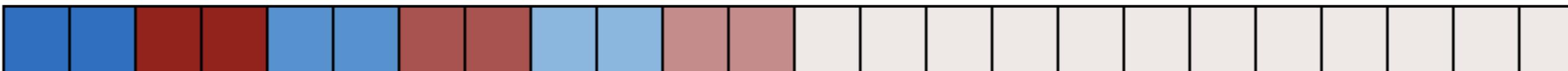
$((1,4))$

Values

0	1	4	5	8	9
---	---	---	---	---	---

Composition Power

Example



$((2,3))$

$((1,4))$

Values

0	1	4	5	8	9
2	3	6	7	10	11

Composition Power

Example



$((2,3))$

$((1,4))$

Values

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

Composition Power

Example



$((2,3))$

$((1,4))$

Values

$((2, 2))$

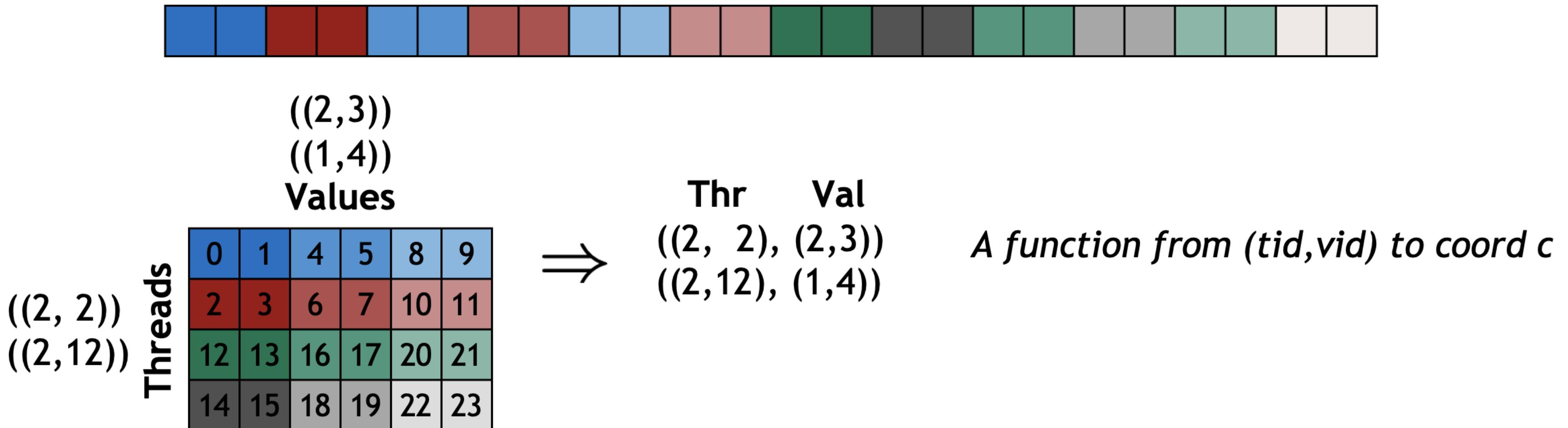
$((2,12))$

Threads

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

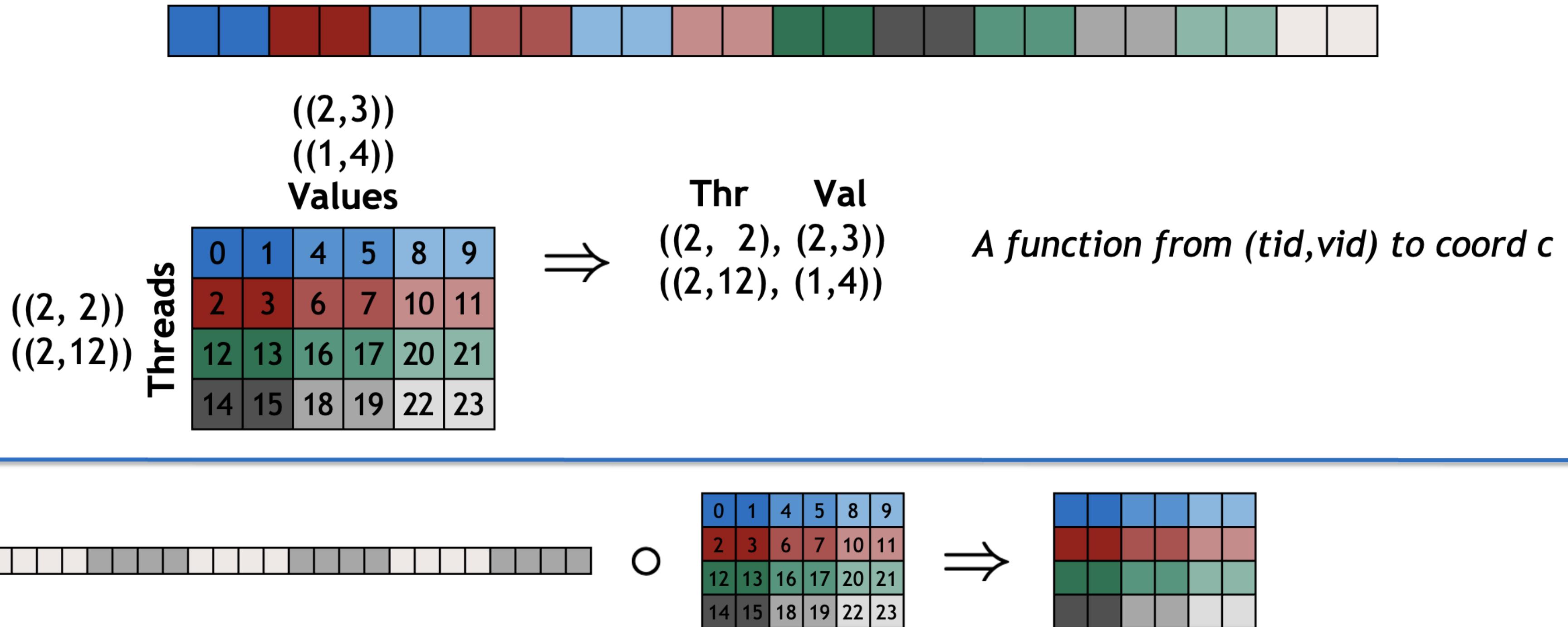
Composition Power

Example



Composition Power

Example



Composition Power

Example

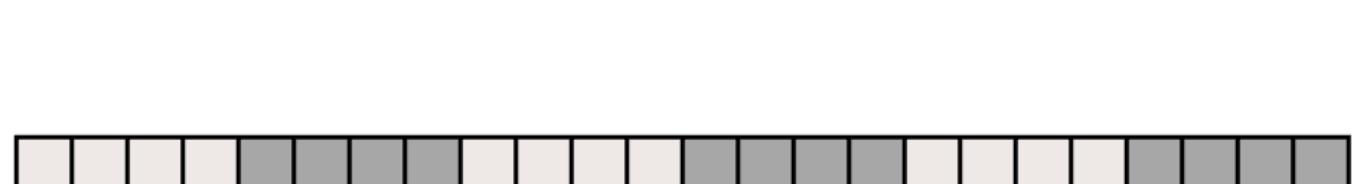


Threads	Values
$((2, 2))$	$(0, 1, 4, 5, 8, 9)$
$((2, 12))$	$(2, 3, 6, 7, 10, 11)$
	$(12, 13, 16, 17, 20, 21)$
	$(14, 15, 18, 19, 22, 23)$

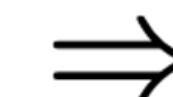
\Rightarrow **Thr Val**
 $((2, 2), (2, 3))$
 $((2, 12), (1, 4))$

A function from (tid, vid) to coord c

```
Tensor input      = make_tensor(...);
Tensor input_TV   = composition(input, thr_val);
Tensor thr_input = input_TV(tid, _);
```



0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

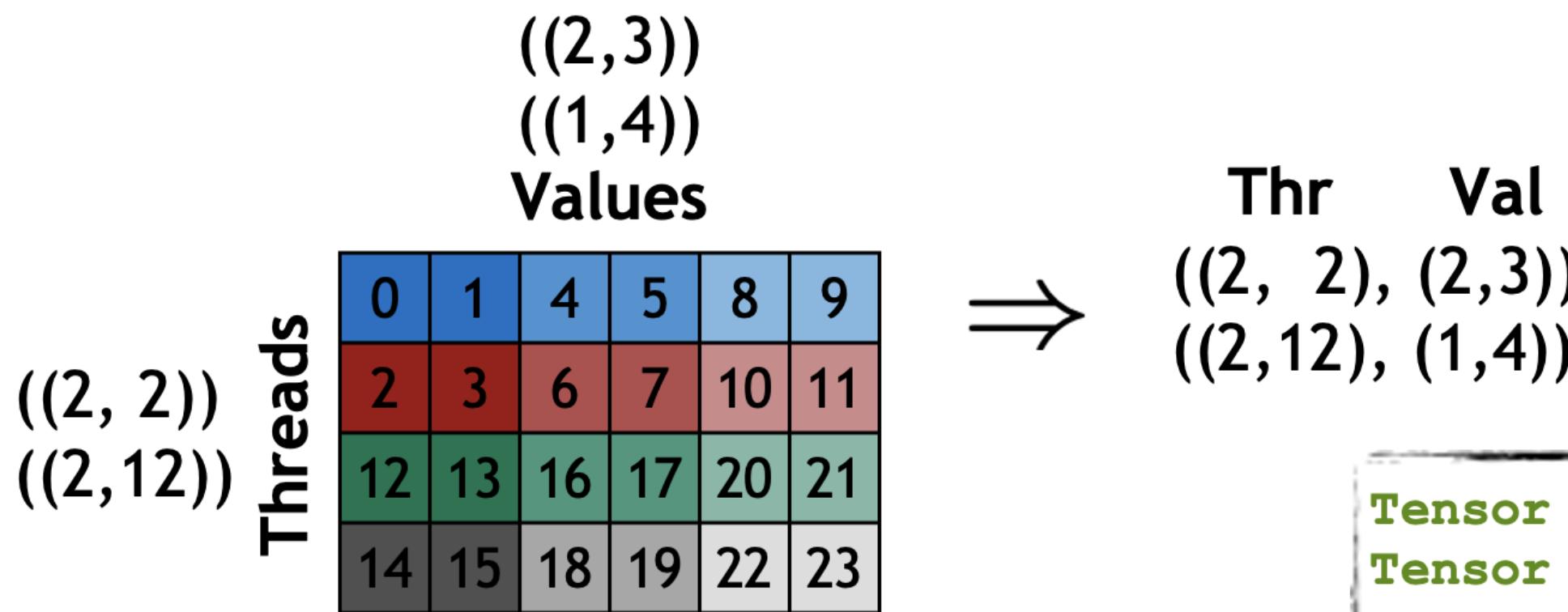


0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

0	1	4	5	8	9
2	3	6	7	10	11
12	13	16	17	20	21
14	15	18	19	22	23

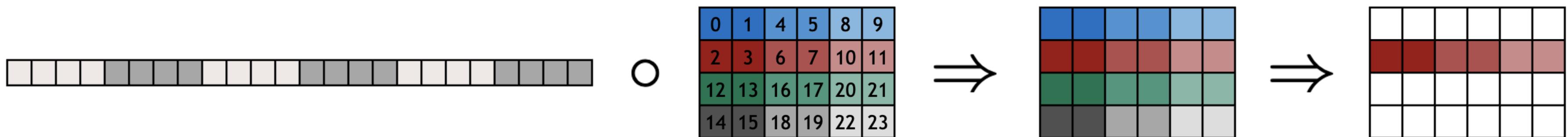
Composition Power

Example



Moral of the story:
Given a map $(\text{thr}, \text{val}) \rightarrow \text{coord}$,
partitioning is simply functional
composition followed by **slicing**

```
Tensor input      = make_tensor(...);  
Tensor input_TV  = composition(input, thr_val);  
Tensor thr_input = input_TV(tid, _);
```



MMA Traits

Volta FP16 8x8x4 Metadata

```
template <>
struct MMA_Traits<SM70_8x8x4_F32F16F16F32_NT>
{
    using ElementDVal = float;
    using ElementAVal = half_t;
    using ElementBVal = half_t;
    using ElementCVal = float;

    using Shape_MNK = Shape<_8, _8, _4>;

    using ThrID = Layout<Shape <_4, _2>,
                        Stride<_1, _16>>;
    // (T8, V4) -> (M8, K4)
    using ALayout = Layout<Shape <_4, _2>, _4>,
                          Stride<Stride<_8, _4>, _1>>;
    // (T8, V4) -> (N8, K4)
    using ALayout = Layout<Shape <_4, _2>, _4>,
                          Stride<Stride<_8, _4>, _1>>;
    // (T8, V8) -> (M8, N8)
    using CLayout = Layout<Shape <_2, _2, _2>, Shape <_2, _2, _2>,
                           Stride<Stride<_1, _16, _4>, Stride<_8, _2, _32>>>;
};
```

	0	1	2	3	4	5	6	7
0	T0 V0	T0 V1	T0 V2	T0 V3	T16 V0	T16 V1	T16 V2	T16 V3
1	T1 V0	T1 V1	T1 V2	T1 V3	T17 V0	T17 V1	T17 V2	T17 V3
2	T2 V0	T2 V1	T2 V2	T2 V3	T18 V0	T18 V1	T18 V2	T18 V3
3	T3 V0	T3 V1	T3 V2	T3 V3	T19 V0	T19 V1	T19 V2	T19 V3
	0	1	2	3	4	5	6	7
0	T0 V0	T1 V0	T2 V0	T3 V0	T0 V4	T0 V5	T2 V4	T2 V5
1	T0 V1	T1 V1	T2 V1	T3 V1	T1 V4	T1 V5	T3 V4	T3 V5
2	T0 V2	T1 V2	T2 V2	T3 V2	T0 V6	T0 V7	T2 V6	T2 V7
3	T0 V3	T1 V3	T2 V3	T3 V3	T1 V6	T1 V7	T3 V6	T3 V7
4	T16 V0	T17 V0	T18 V0	T19 V0	T16 V4	T16 V5	T18 V4	T18 V5
5	T16 V1	T17 V1	T18 V1	T19 V1	T17 V4	T17 V5	T19 V4	T19 V5
6	T16 V2	T17 V2	T18 V2	T19 V2	T16 V6	T16 V7	T18 V6	T18 V7
7	T16 V3	T17 V3	T18 V3	T19 V3	T17 V6	T17 V7	T19 V6	T19 V7

SM70_8x8x4_F32F16F16F32_NT

MMA Traits

Ampere FP64 8x8x4 Metadata

```
template <>
struct MMA_Traits<SM80_8x8x4_F64F64F64F64_TN>
{
    using ElementDVal = double;
    using ElementAVal = double;
    using ElementBVal = double;
    using ElementCVal = double;

    using Shape_MNK = Shape<_8,_8,_4>;

    using ThrID = Layout<_32>;
    // (T32,V1) -> (M8,K4)
    using ALayout = Layout<Shape<Shape<_4,_8,>,_1>,
                           Stride<Stride<_8,_1>,_0>>;
    // (T32,V1) -> (N8,K4)
    using BLayout = Layout<Shape<Shape<_4,_8,>,_1>,
                           Stride<Stride<_8,_1>,_0>>;
    // (T32,V2) -> (M8,N8)
    using CLayout = Layout<Shape<Shape<_4,_8,>,_2>,
                           Stride<Stride<_16,_1>,_8>>;
};


```

	0	1	2	3	4	5	6	7
0	T0 V0	T4 V0	T8 V0	T12 V0	T16 V0	T20 V0	T24 V0	T28 V0
1	T1 V0	T5 V0	T9 V0	T13 V0	T17 V0	T21 V0	T25 V0	T29 V0
2	T2 V0	T6 V0	T10 V0	T14 V0	T18 V0	T22 V0	T26 V0	T30 V0
3	T3 V0	T7 V0	T11 V0	T15 V0	T19 V0	T23 V0	T27 V0	T31 V0

	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
0	T0 V0	T1 V0	T2 V0	T3 V0	T0 V0	T0 V1	T1 V0	T1 V1	T2 V0	T2 V1	T3 V0	T3 V1	T0 V0	T0 V1	T1 V0	T1 V1
1	T4 V0	T5 V0	T6 V0	T7 V0	T4 V0	T4 V1	T5 V0	T5 V1	T6 V0	T6 V1	T7 V0	T7 V1	T8 V0	T8 V1	T9 V0	T9 V1
2	T8 V0	T9 V0	T10 V0	T11 V0	T8 V0	T8 V1	T9 V0	T9 V1	T10 V0	T10 V1	T11 V0	T11 V1	T12 V0	T12 V1	T13 V0	T13 V1
3	T12 V0	T13 V0	T14 V0	T15 V0	T12 V0	T12 V1	T13 V0	T13 V1	T14 V0	T14 V1	T15 V0	T15 V1	T16 V0	T16 V1	T17 V0	T17 V1
4	T16 V0	T17 V0	T18 V0	T19 V0	T16 V0	T16 V1	T17 V0	T17 V1	T18 V0	T18 V1	T19 V0	T19 V1	T20 V0	T20 V1	T21 V0	T21 V1
5	T20 V0	T21 V0	T22 V0	T23 V0	T20 V0	T20 V1	T21 V0	T21 V1	T22 V0	T22 V1	T23 V0	T23 V1	T24 V0	T24 V1	T25 V0	T25 V1
6	T24 V0	T25 V0	T26 V0	T27 V0	T24 V0	T24 V1	T25 V0	T25 V1	T26 V0	T26 V1	T27 V0	T27 V1	T28 V0	T28 V1	T29 V0	T29 V1
7	T28 V0	T29 V0	T30 V0	T31 V0	T28 V0	T28 V1	T29 V0	T29 V1	T30 V0	T30 V1	T31 V0	T31 V1				

SM80_8x8x4_F64F64F64F64_TN

MMA Traits

Ampere FP16 16x8x8 Metadata

```

template <>
struct MMA_Traits<SM80_16x8x8_F32F16F16F32_TN>
{
    using ElementDVal = float;
    using ElementAVal = half_t;
    using ElementBVal = half_t;
    using ElementCVal = float;

    using Shape_MNK = Shape<_16,_8,_8>;
    using ThrID = Layout<_32>;
    // (T32,V4) -> (M16,K8)
    using ALayout = Layout<Shape <Shape <_4,_8>,_1>,
                           Stride<Stride<_8,_1>,_0>>;
    // (T32,V2) -> (N8,K8)
    using BLayout = Layout<Shape <Shape <_4,_8>,_1>,
                           Stride<Stride<_8,_1>,_0>>;
    // (T32,V4) -> (M16,N8)
    using CLayout = Layout<Shape <Shape <_4,_8>,_2>,
                           Stride<Stride<_16,_1>,_8>>;
};

```

	0	1	2	3	4	5	6	7
0	T0 V0	T4 V0	T8 V0	T12 V0	T16 V0	T20 V0	T24 V0	T28 V0
1	T0 V1	T4 V1	T8 V1	T12 V1	T16 V1	T20 V1	T24 V1	T28 V1
2	T1 V0	T5 V0	T9 V0	T13 V0	T17 V0	T21 V0	T25 V0	T29 V0
3	T1 V1	T5 V1	T9 V1	T13 V1	T17 V1	T21 V1	T25 V1	T29 V1
4	T2 V0	T6 V0	T10 V0	T14 V0	T18 V0	T22 V0	T26 V0	T30 V0
5	T2 V1	T6 V1	T10 V1	T14 V1	T18 V1	T22 V1	T26 V1	T30 V1
6	T3 V0	T7 V0	T11 V0	T15 V0	T19 V0	T23 V0	T27 V0	T31 V0
7	T3 V1	T7 V1	T11 V1	T15 V1	T19 V1	T23 V1	T27 V1	T31 V1
	0	1	2	3	4	5	6	7
0	T0 V0	T0 V1	T1 V0	T1 V1	T2 V0	T2 V1	T3 V0	T3 V1
1	T4 V0	T4 V1	T5 V0	T5 V1	T6 V0	T6 V1	T7 V0	T7 V1
2	T8 V0	T8 V1	T9 V0	T9 V1	T10 V0	T10 V1	T11 V0	T11 V1
3	T12 V0	T12 V1	T13 V0	T13 V1	T14 V0	T14 V1	T15 V0	T15 V1
4	T16 V0	T16 V1	T17 V0	T17 V1	T18 V0	T18 V1	T19 V0	T19 V1
5	T20 V0	T20 V1	T21 V0	T21 V1	T22 V0	T22 V1	T23 V0	T23 V1
6	T24 V0	T24 V1	T25 V0	T25 V1	T26 V0	T26 V1	T27 V0	T27 V1
7	T28 V0	T28 V1	T29 V0	T29 V1	T30 V0	T30 V1	T31 V0	T31 V1
	0	1	2	3	4	5	6	7
8	T0 V2	T0 V3	T1 V2	T1 V3	T2 V2	T2 V3	T3 V2	T3 V3
9	T4 V2	T4 V3	T5 V2	T5 V3	T6 V2	T6 V3	T7 V2	T7 V3
10	T8 V2	T8 V3	T9 V2	T9 V3	T10 V2	T10 V3	T11 V2	T11 V3
11	T12 V2	T12 V3	T13 V2	T13 V3	T14 V2	T14 V3	T15 V2	T15 V3
12	T16 V2	T16 V3	T17 V2	T17 V3	T18 V2	T18 V3	T19 V2	T19 V3
13	T20 V2	T20 V3	T21 V2	T21 V3	T22 V2	T22 V3	T23 V2	T23 V3
14	T24 V2	T24 V3	T25 V2	T25 V3	T26 V2	T26 V3	T27 V2	T27 V3
15	T28 V2	T28 V3	T29 V2	T29 V3	T30 V2	T30 V3	T31 V2	T31 V3

SM80_16x8x8_F32F16F16F32_TN

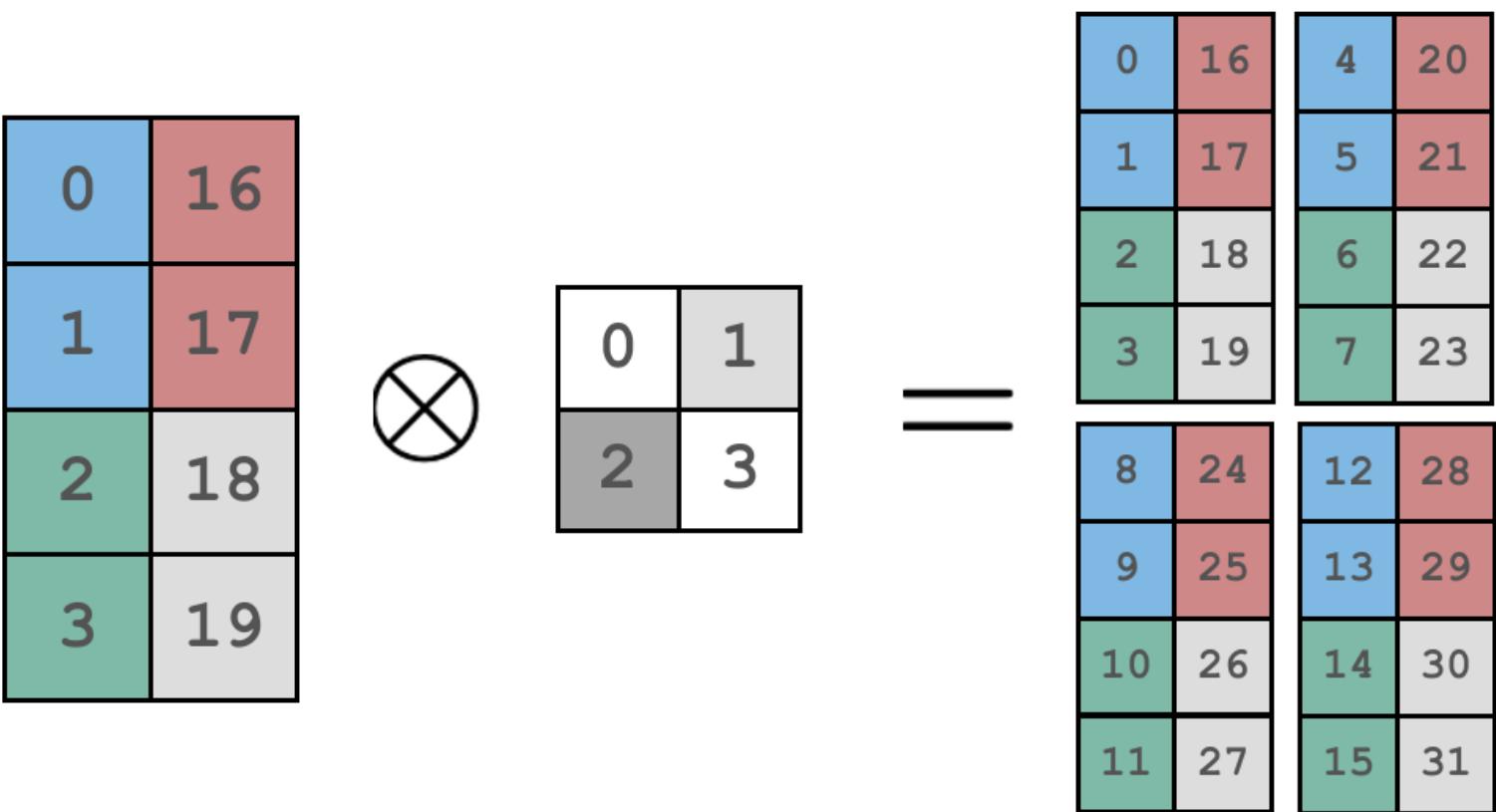
Layout Algebra

Logical Product

$$f_A \otimes g_B = (f_A, f_A^* \circ g_B) \rightarrow (f_A, h_{B'})$$

"Produce a layout where every element of layout B is a layout A."

- logical_product
- blocked_product
- raked_product
- tile_to_shape

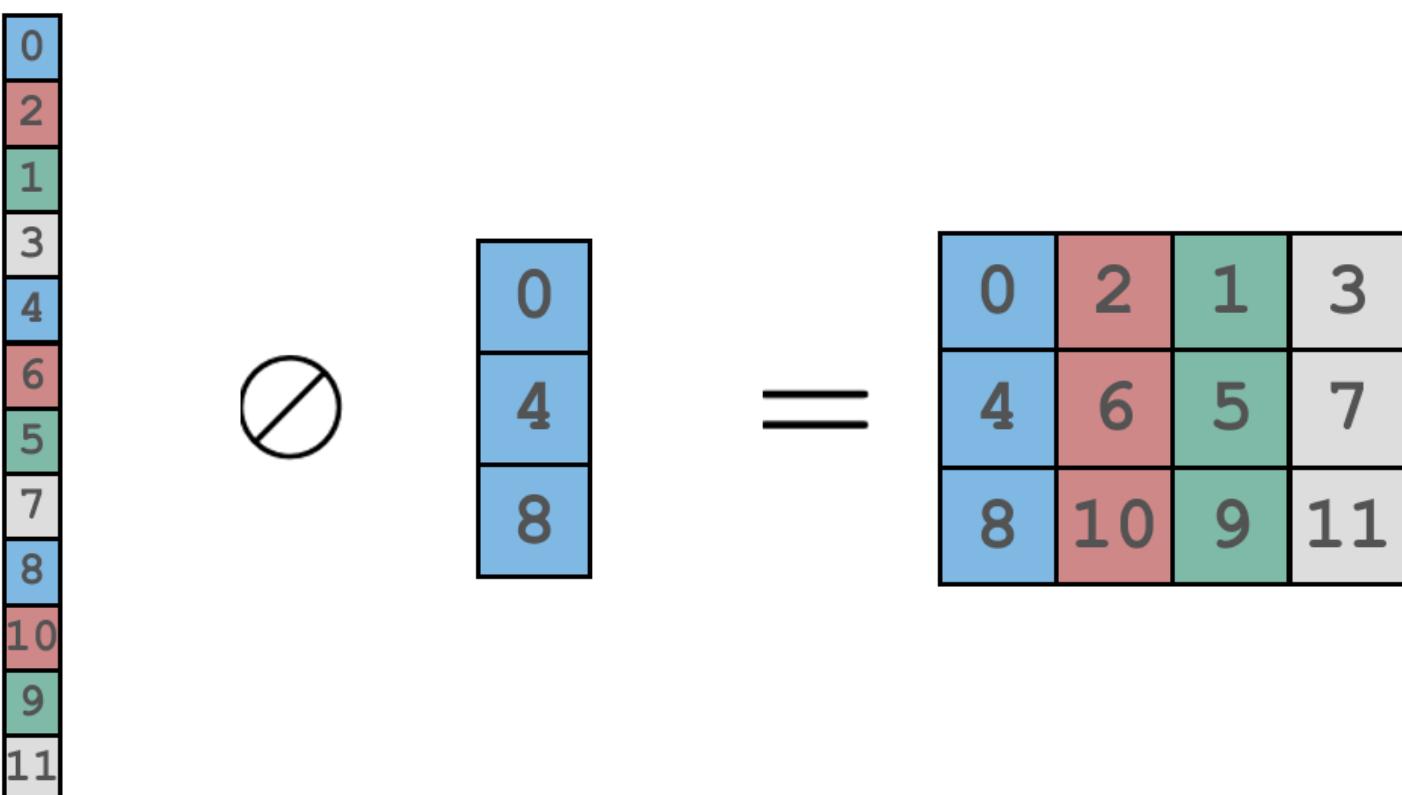


Logical Divide

$$f_A \oslash g_B = f_A \circ (g_B, g_B^*) \rightarrow (h_{B'}, \ell_C)$$

"Split a layout A into elements pointed to by layout B and the rest."

- logical_divide
- zipped_divide
- tiled_divide



See CuTe documentation for discussion and examples

<https://github.com/NVIDIA/cutlass/tree/master/media/docs/cute>



Agenda

- Hopper Architecture
- CuTe
- CUTLASS 3.0
- CUTLASS Python
- Conclusion

CUTLASS 3

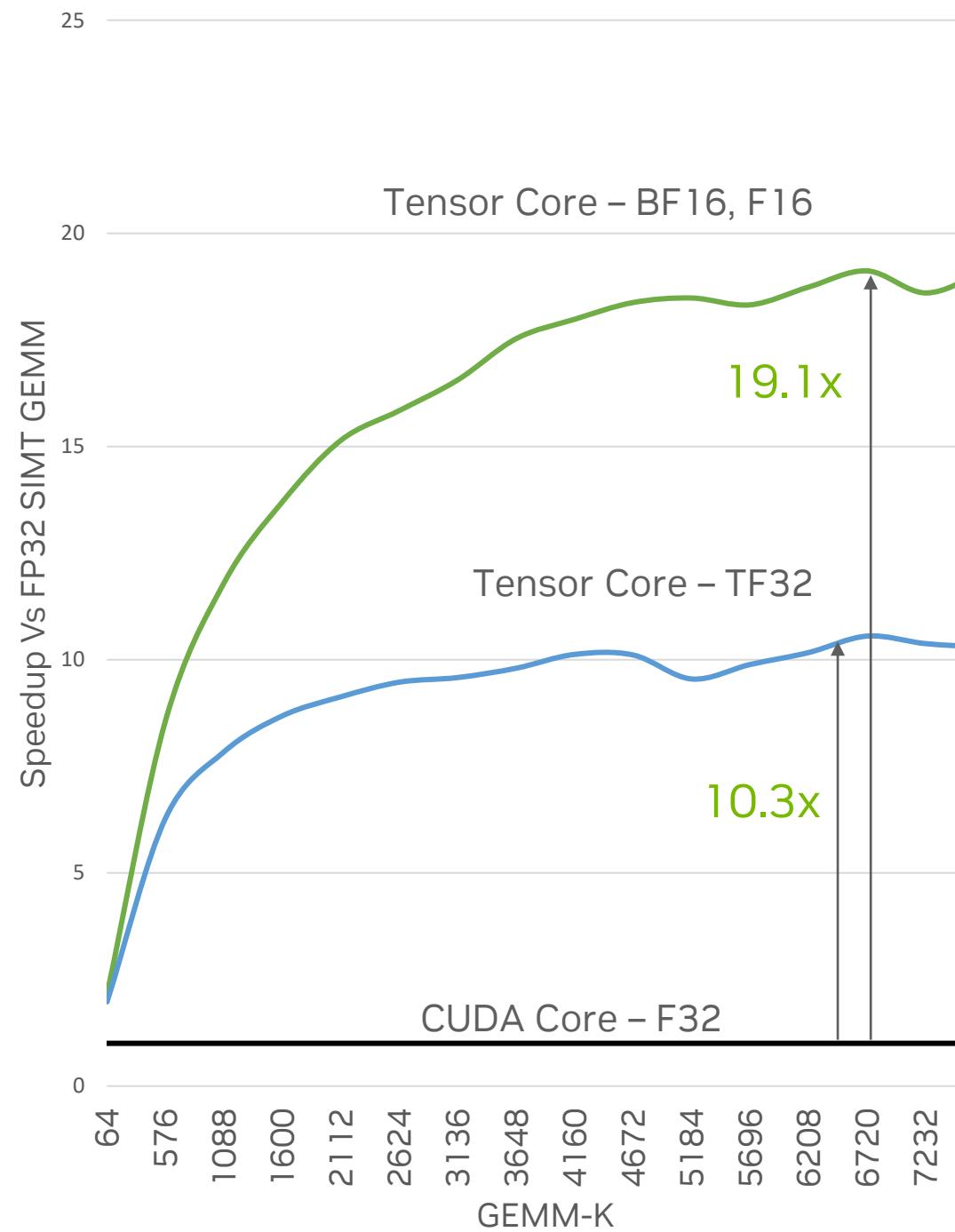
What's new ?

- **CUTLASS 3.1 :**
 - Register backed WGMMA Kernels for TF32.
 - New Pythonic Interface for CUTLASS
 - Efficient Epilogues with fusions
- **CUTLASS 3.0 :** A significant refactoring leveraging CuTe backend
 - Efficient Tensor Core Hopper Instructions & async. copy using TMA
 - Warp Specialized & Persistent kernel implementations
 - Collective Builders, Documentation, profiler support, PyCUTLASS integration, SDK Samples and more...
- **CUTLASS 2.11**
 - Fused MHA – for Ampere kernels
 - Stream-K – A new generalized Split-K implementation
 - BLAS3 Functionality with New Hopper double precision instructions

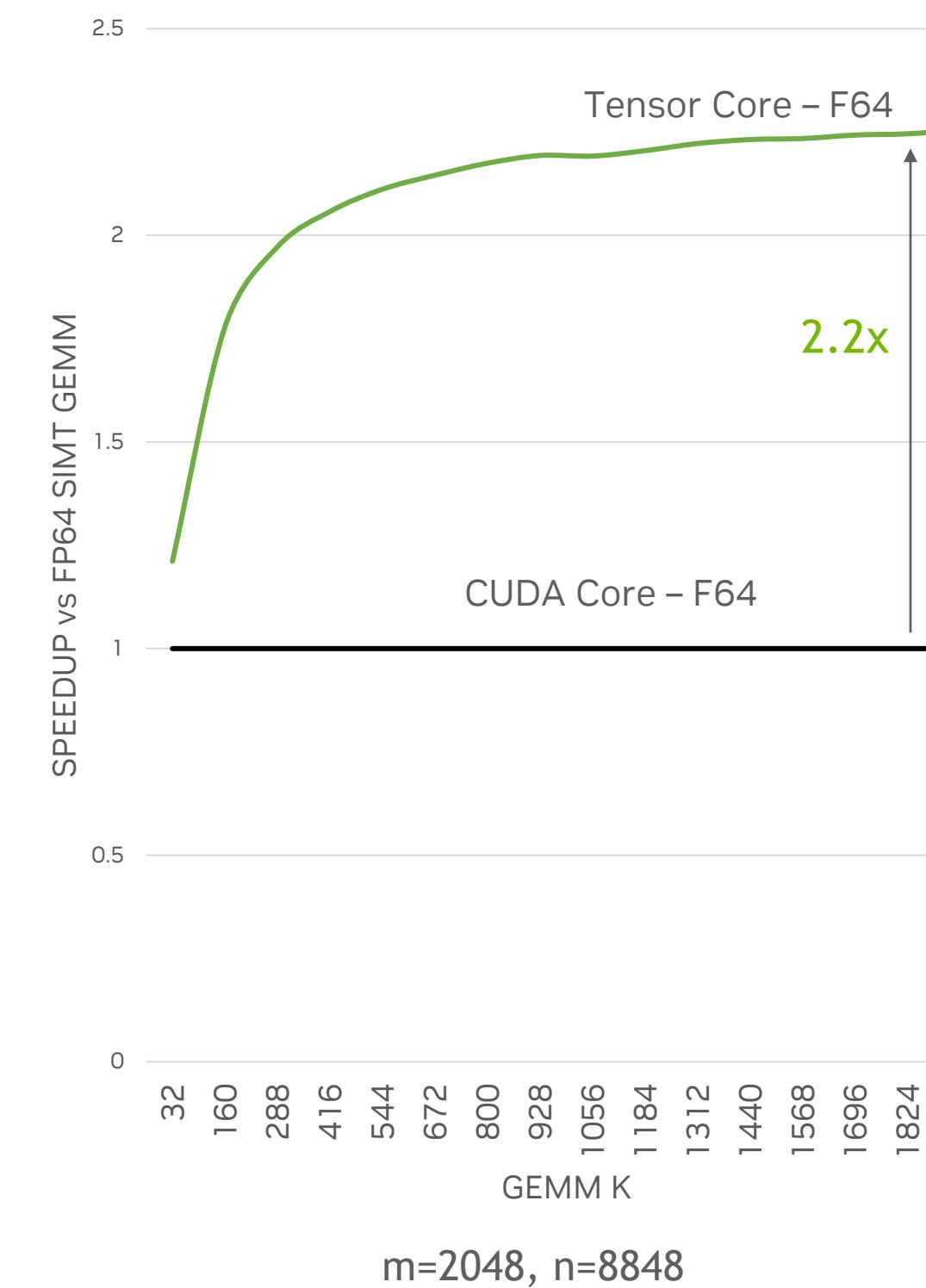
TENSOR CORE PERFORMANCE ON NVIDIA HOPPER

CUTLASS 3.0 - CUDA 12.0 Toolkit - NVIDIA H100

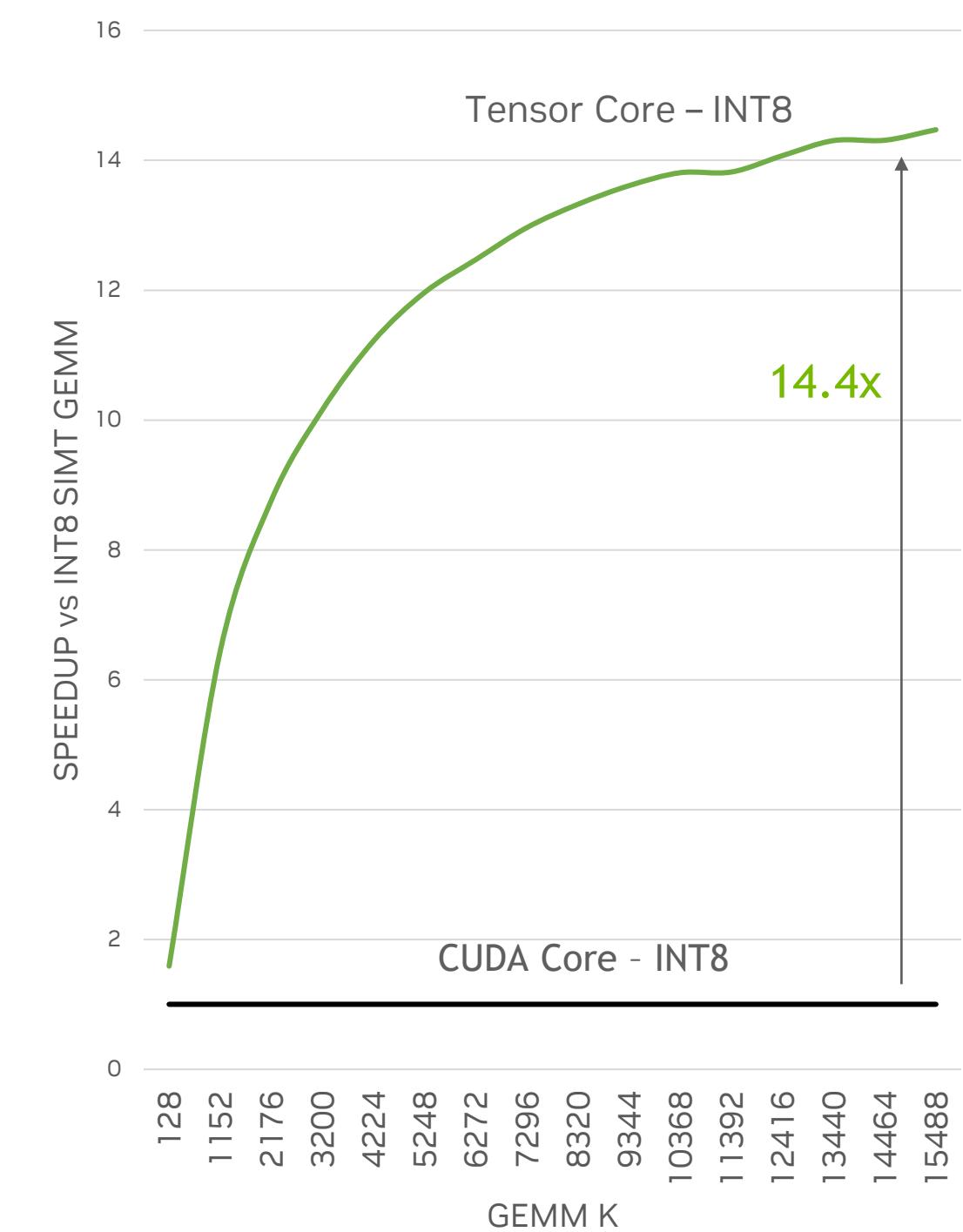
Mixed Precision Floating Point



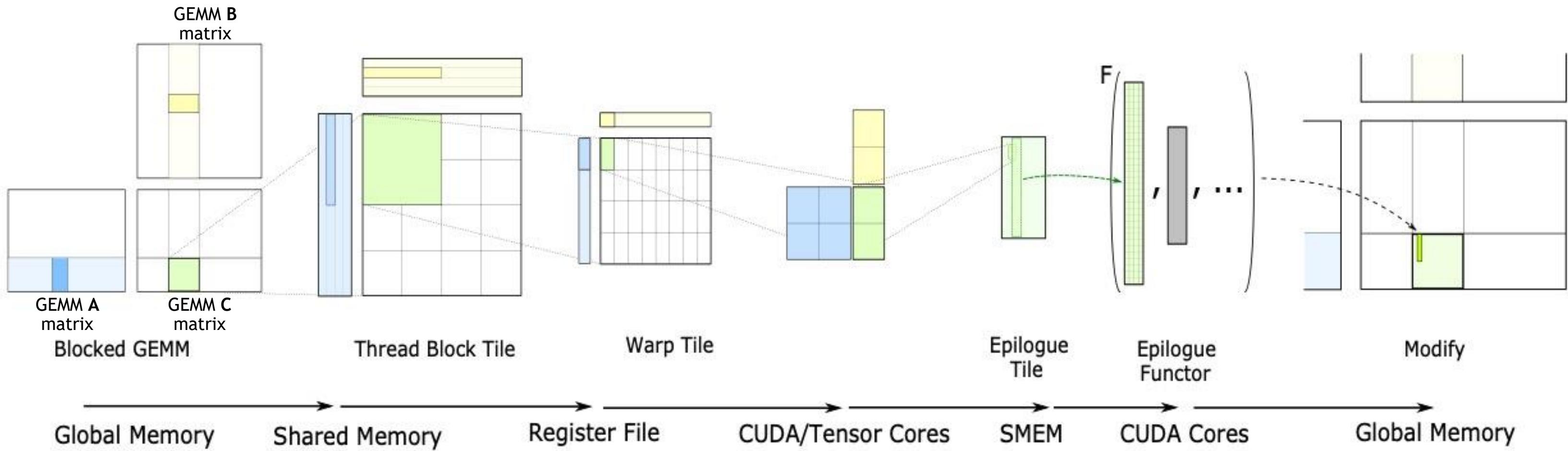
Double Precision Floating Point



Mixed Precision Integer



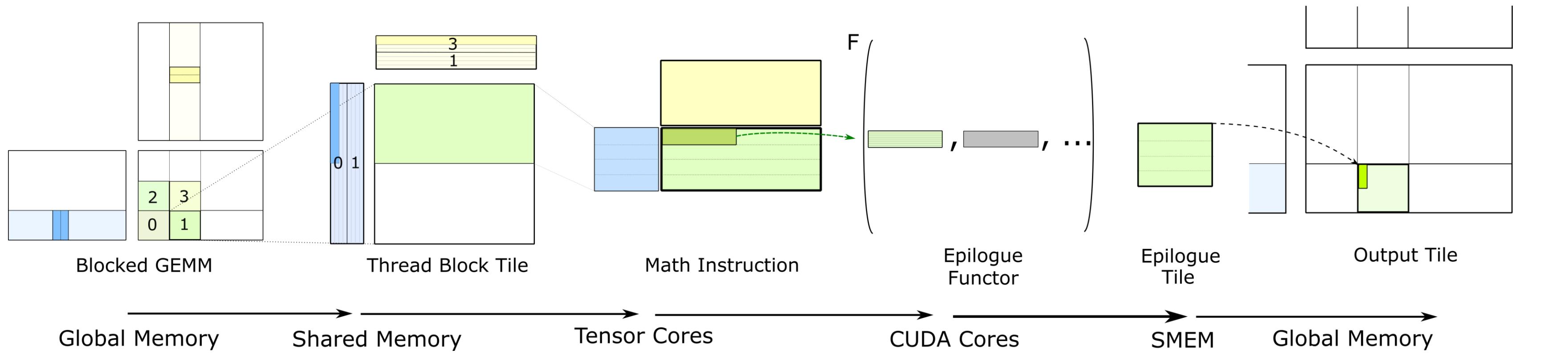
BLOCKED GEMM RECAP



Tiled, hierarchical model: reuse data in Shared Memory and in Registers

See [CUTLASS GTC 2018](#) and [2020](#) talks for more details about this model

WHAT'S CHANGED IN HOPPER



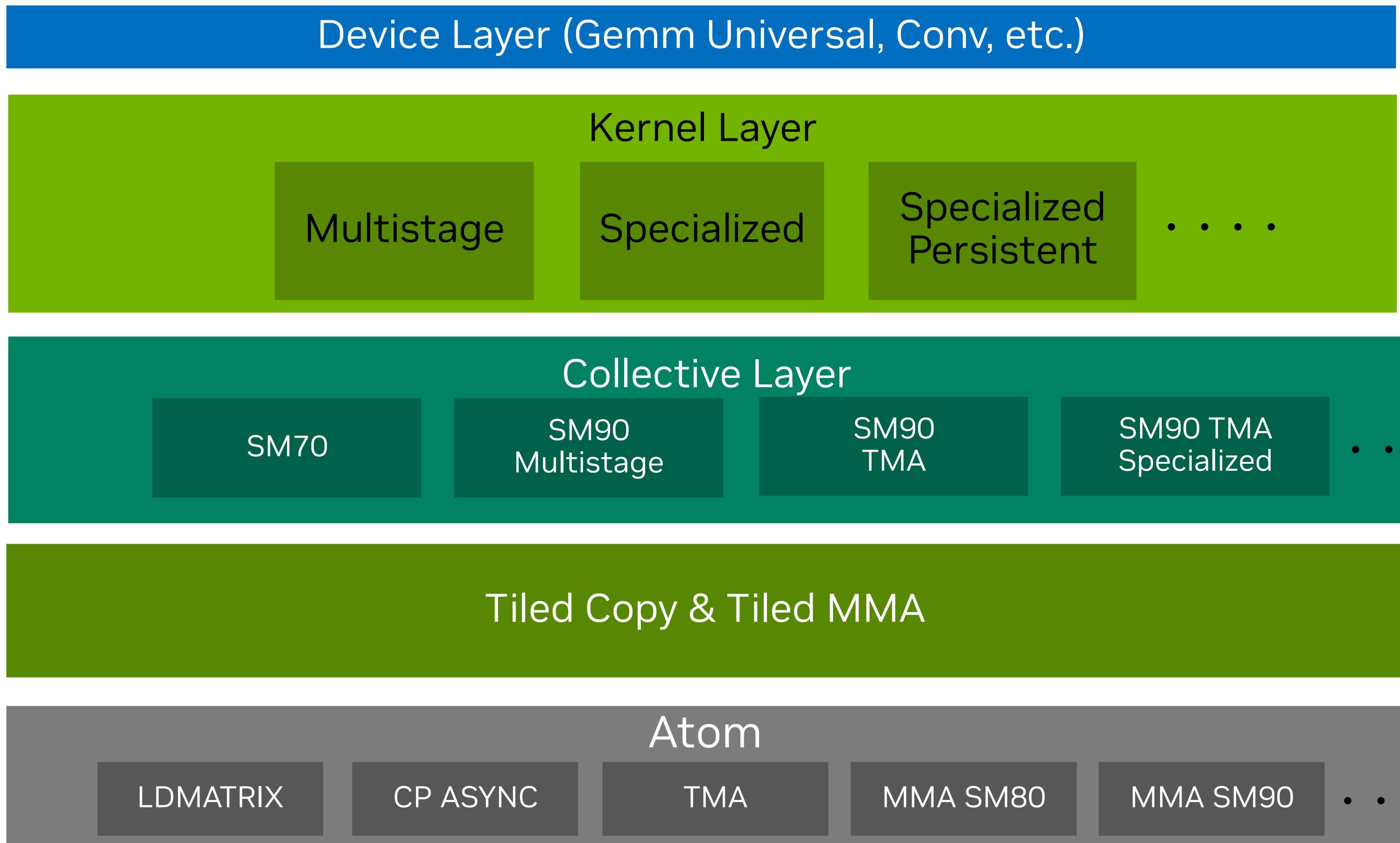
Block Cluster Tiled, Bulk Copy in and out of global memory, re-use data directly from Shared Memory

CUTLASS 3 Conceptual GEMM Hierarchy

Not centered around the hardware hierarchy

- **Device layer:** host side setup and interface
- **Kernel layer:** Launch API, grid planning logic, load balancing schedules, and kernel thread marshalling
 - Conceptually: A collection of all threadblock/clusters in the grid
- **Collective layer:** Mainloops that orchestrate the copy/math micro-kernels with arch specific synchronization
 - Conceptually: Dynamic loops invoking the math/copy micro-kernels that compute the accumulated inner product of outer products
- **Tiled MMA/Copy:** The GPU micro-kernel interface
 - Conceptually: The largest collection of threads over which the GPU micro-kernel is constructed for math/copy
- **Atom layer:** Architecture instructions and their associated meta-information
 - Conceptually: The fewest number of threads and values that must participate in an architecture accelerated specified math/copy op

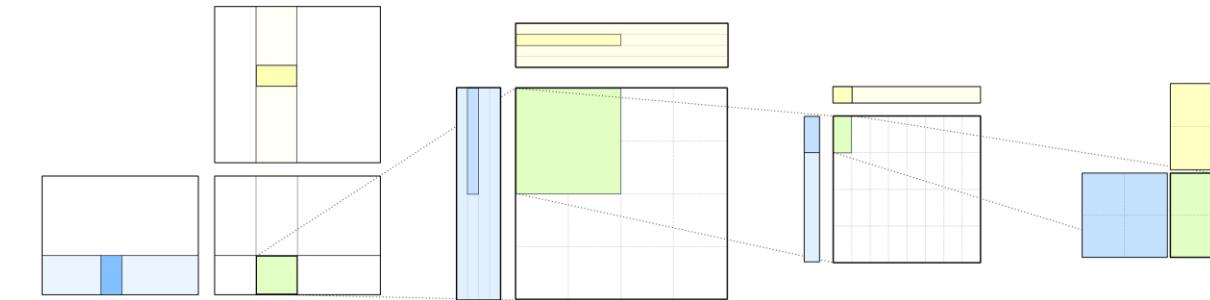
CUTLASS 3 Conceptual GEMM Hierarchy



CUTLASS 3.x API Entry Points

Reduced API Surface Area

- Device layer: `cutlass::gemm::device::GemmUniversalAdapter<>`
 - Can be used with 2.x or 3.x API kernels
 - A stateless wrapper around a kernel type
- Kernel layer: `cutlass::gemm::kernel::GemmUniversal<>`
 - Treats GEMM as a composition of a collective mainloop and a collective epilogue
 - Each new kernel schedule is a specialization dispatched against with schedule tags
- Collective layer: `cutlass::gemm::collective::CollectiveMma<>`
 - Dispatched against by policies that also define the set of kernel schedules they can be composed with
- Microkernel layer: `cute::TiledMma<>` and `cute::TiledCopy<>`
 - Robust representation power across a wide range GPU architectures
- Static asserts everywhere to guard against invalid compositions or incorrect layouts



CUTLASS 3: Kernel API

`cutlass::gemm::kernel::GemmUniversal<>`

- Each GEMM is a fusion of a mainloop and an epilogue
- Block/Cluster wide swizzling, grid planning logic, load balancing schedules
- Thread marshalling for warp-specialized collectives
- Selection via dispatch policy's `Schedule` tag
- Rule of thumb:
 - Each mainloop variant gets its own policy type for dispatch
 - Can be composed with a set of kernel schedules that it specifies and checks against

This dispatch policy example shows that warp specialized mainloop can be composed with both persistent and non-persistent kernel schedules.

```
template<
    int Stages_,
    class ClusterShape_ = Shape<-1,-1,-1>,
    class KernelSchedule = KernelTmaWarpSpecialized // or KernelTmaWarpSpecializedPersistent
>
struct MainloopSm90TmaGmmaWarpSpecialized {
    constexpr static int Stages = Stages_;
    using ClusterShape = ClusterShape_;
    using ArchTag = arch::Sm90;
    using Schedule = KernelSchedule;
};
```

CUTLASS 3: Collective API

`cutlass::gemm::collective::Gemm<>`
`cutlass::epilogue::collective::Epilogue<>`

Threadblock Cluster wide operand and work ownership

Dispatch policies allow for freely composing with any kernel schedule while proving guard rails

```
// n-buffer in smem, pipelined with Hopper GMMA and TMA
template<
    int Stages_,
    class ClusterShape_ = Shape<_1,_1,_1>,
    int PipelineAsyncMmaStages_ = 1
>
struct MainloopSm90TmaGmma {
    constexpr static int Stages = Stages_;
    using ClusterShape = ClusterShape_;
    constexpr static int PipelineAsyncMmaStages = PipelineAsyncMmaStages_;
    using ArchTag = arch::Sm90;
    using Schedule = KernelTma;
};

// n-buffer in smem, pipelined with Hopper GMMA and TMA, warp-specialized
template<
    int Stages_,
    class ClusterShape_ = Shape<_1,_1,_1>,
    class KernelSchedule = KernelTmaWarpSpecialized
>
struct MainloopSm90TmaGmmaWarpSpecialized {
    constexpr static int Stages = Stages_;
    using ClusterShape = ClusterShape_;
    using ArchTag = arch::Sm90;
    using Schedule = KernelSchedule;
};
```

```
template <
    class DispatchPolicy,
    class TileShape,
    class ElementA,
    class SmemLayoutA,
    class ElementB,
    class SmemLayoutB,
    class ElementC,
    class ArchTag,
    class TiledMma,
    class GmemCopyAtomA,
    class SmemCopyAtomA,
    class GmemCopyAtomB,
    class SmemCopyAtomB
>
```

```
struct CollectiveMma;
```

CUTLASS 3: Collective Builders

`cutlass::gemm::collective::CollectiveBuilder<>`

A convenience interface on top of the primary collective API

Meta-programs that map common 2.x style args to generate 3.x types

Replaces `DefaultXConfiguration` specializations of 2.x

“I just want a Hopper mainloop”:

```
using CollectiveOp = typename cutlass::gemm::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    half_t, LayoutA, 8,
    half_t, LayoutB, 8,
    float,
    Shape< 128, 128, 64>, Shape< 1, 2, 1>,
    gemm::collective::StageCountAuto,
    gemm::collective::KernelScheduleAuto
>::CollectiveOp;
```

“I want a Hopper mainloop but with persistent scheduling and 5 stages”:

```
using CollectiveOp = typename cutlass::gemm::collective::CollectiveBuilder<
    arch::Sm90, arch::OpClassTensorOp,
    half_t, LayoutA, 8,
    half_t, LayoutB, 8,
    float,
    Shape< 128, 128, 64>, Shape< 1, 2, 1>,
    gemm::collective::StageCount<5>,
    gemm::KernelTmaWarpSpecializedPersistent
>::CollectiveOp;
```

```
template <
    class ArchTag,
    class OpClass,
    class ElementA,
    class GmemLayoutA,
    int AlignmentA,
    class ElementB,
    class GmemLayoutB,
    int AlignmentB,
    class ElementAccumulator,
    class TileShape_MNK,
    class ClusterShape_MNK,
    class StageCountType,
    class KernelScheduleType,
    class Enable = void
>
struct CollectiveBuilder;
```

Recipe for Peak Performance

Attack of the asynchronous machines

- **Global memory accesses using TMA:**

- TMA with programmatic multicast necessary for L2 bandwidth efficiency
- Saves predicate and address arithmetic ops.
- TMA issuing threads must issue only TMA instructions for minimizing latency coverage needs

- **MMA operation :**

- Issue multiple asynchronous MMA instructions in flight synchronized against TMA barriers
- MMA issuing threads must only issue MMA for peak issue rate

- **Latency hiding of data movement for peak math throughput**

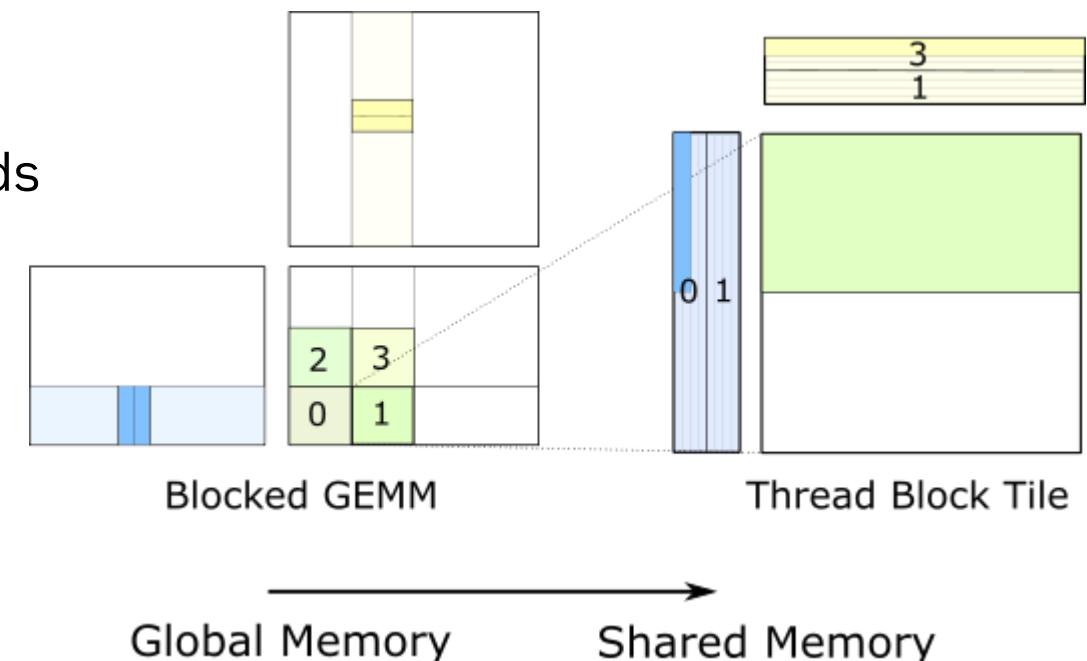
- Efficient synchronization of async operations at peak performance.
- Software pipelining of gmem->smem through buffers in shared memory

- **Must also hide software pipeline head & tail induced bubbles**

- MMA instructions have strong scaled a lot
- Persistent GEMMs are unavoidable and also require CTA reconfiguration.

- **Must still swizzle data in smem**

- Now prescribed by MMA and implemented by TMA



Async. Pipelines

Managing Arrive wait &
Transaction Barriers

- Hopper H100, relies heavily on deep async software pipelines for peak performance.
- With a dozen+ circular buffer stages, managing Async. pipelines directly by manipulating barriers is tedious across multiple sets of producers and consumers.
- CUTLASS hence adds support for Async Pipeline classes which provide a functional abstraction API for achieving synchronization using underlying hardware features.

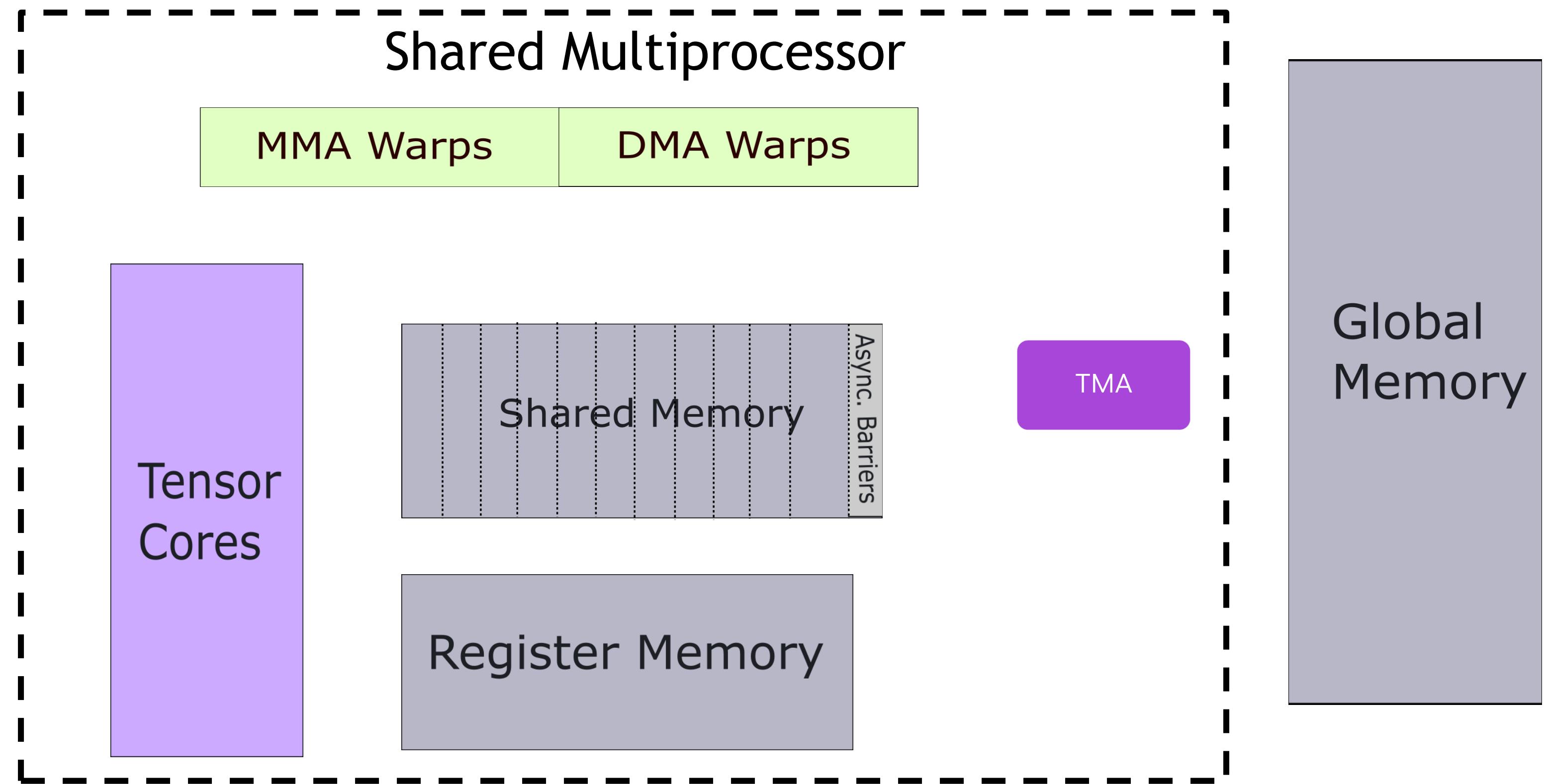
```
template <int Stages, class ClusterShape>
class PipelineTmaAsync {
    // Acquire a stage in Smem before writing to it
    void producer_acquire(PipelineState<Stages> state);

    // Commit a stage after writing to Smem (optional)
    void producer_commit(PipelineState<Stages> state);

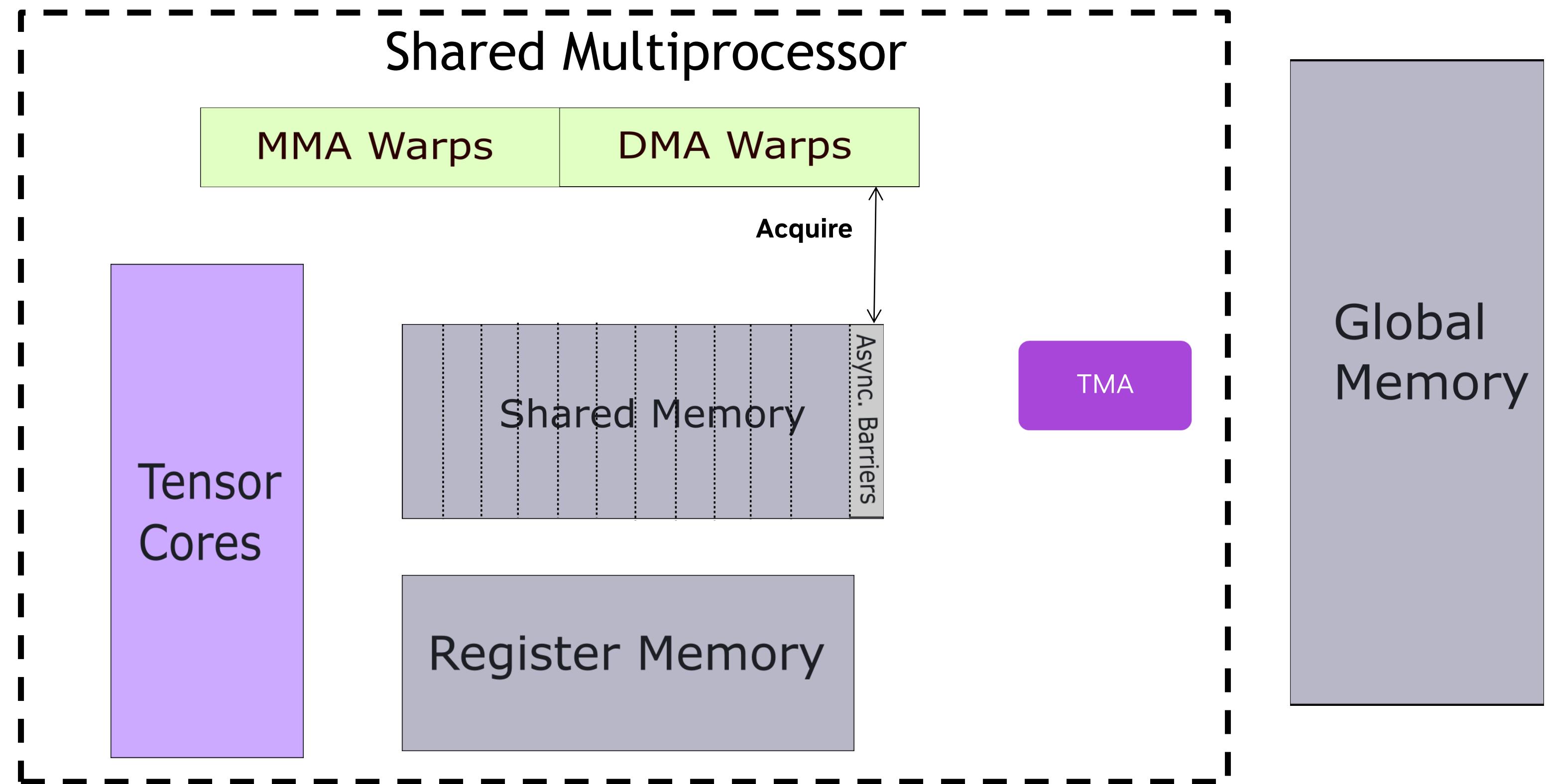
    // Wait for Commit before consuming a stage in Smem
    void consumer_wait(PipelineState<Stages> state);

    // Notify end of consumption of Smem stage
    void consumer_release(PipelineState<Stages> state);
};
```

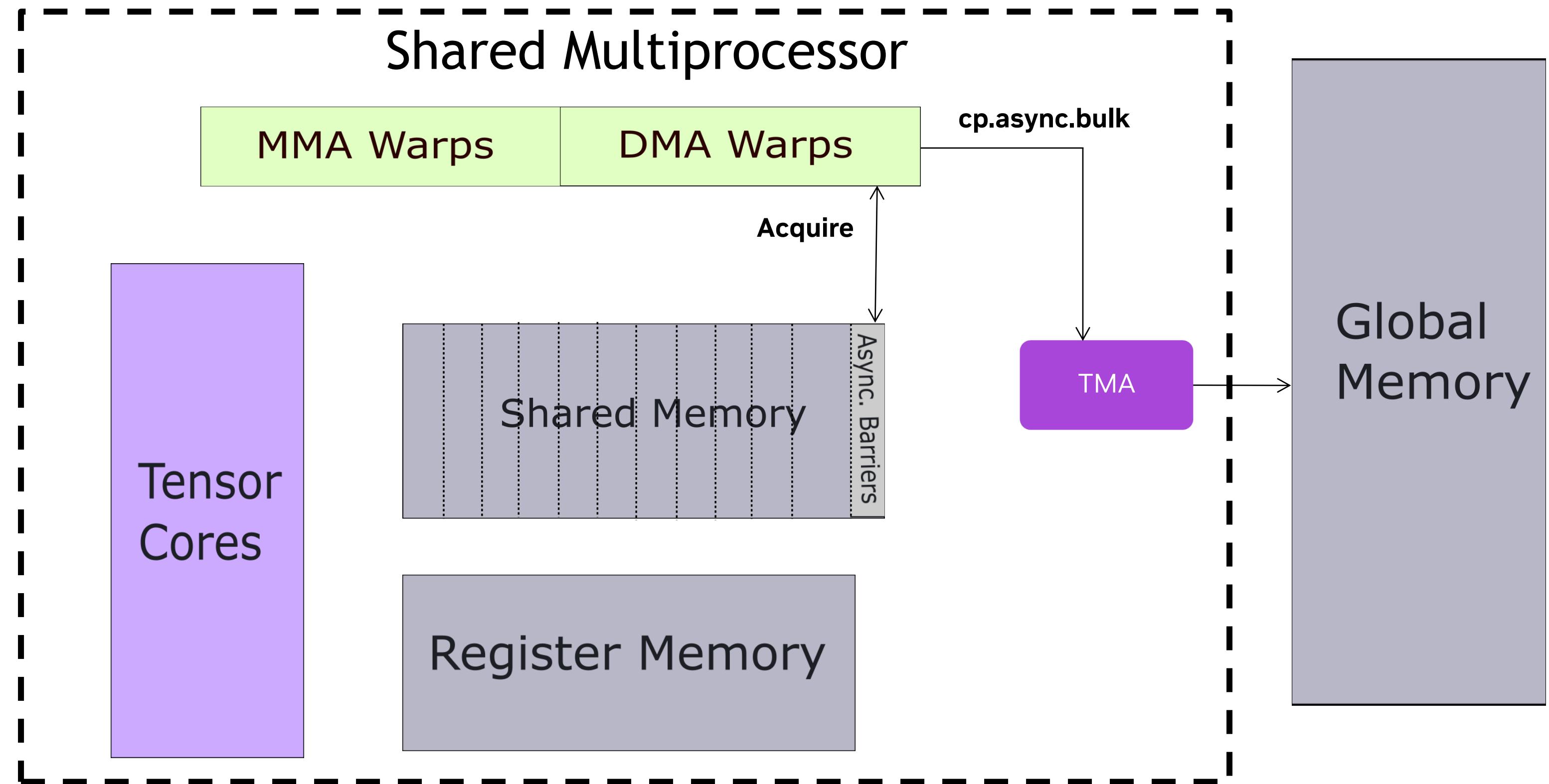
Async. Warp Specialized Kernel



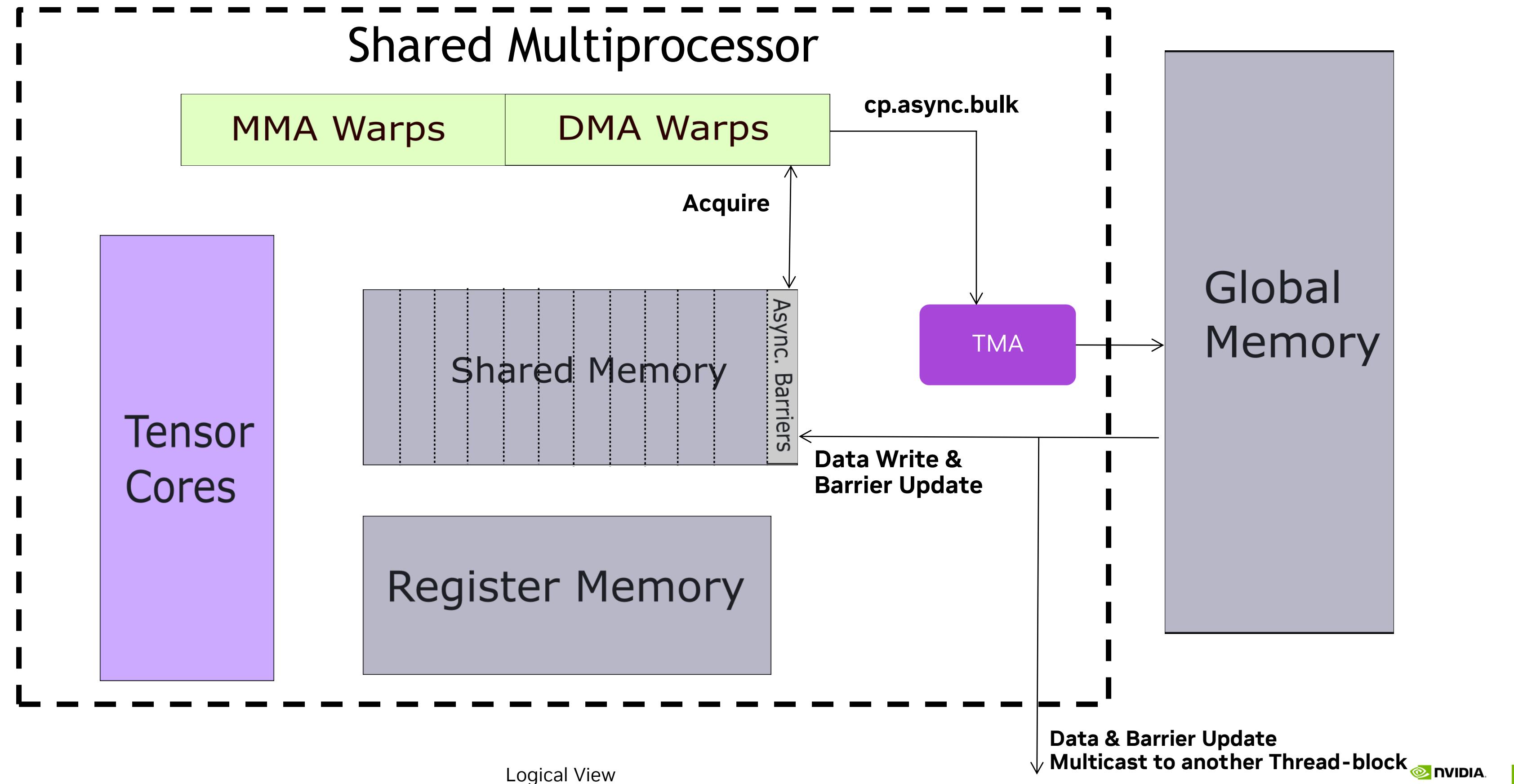
Async. Warp Specialized Kernel



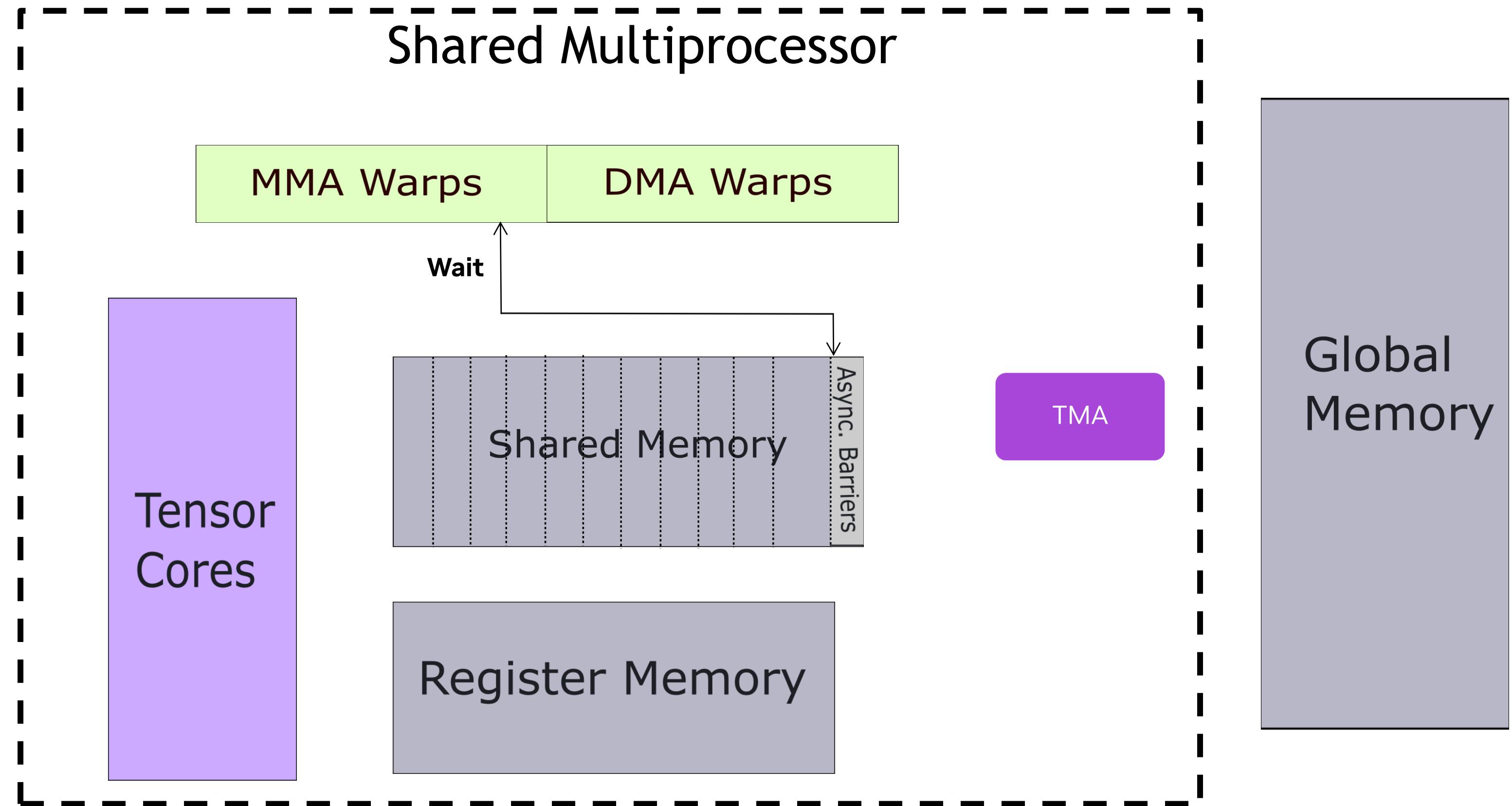
Async. Warp Specialized Kernel



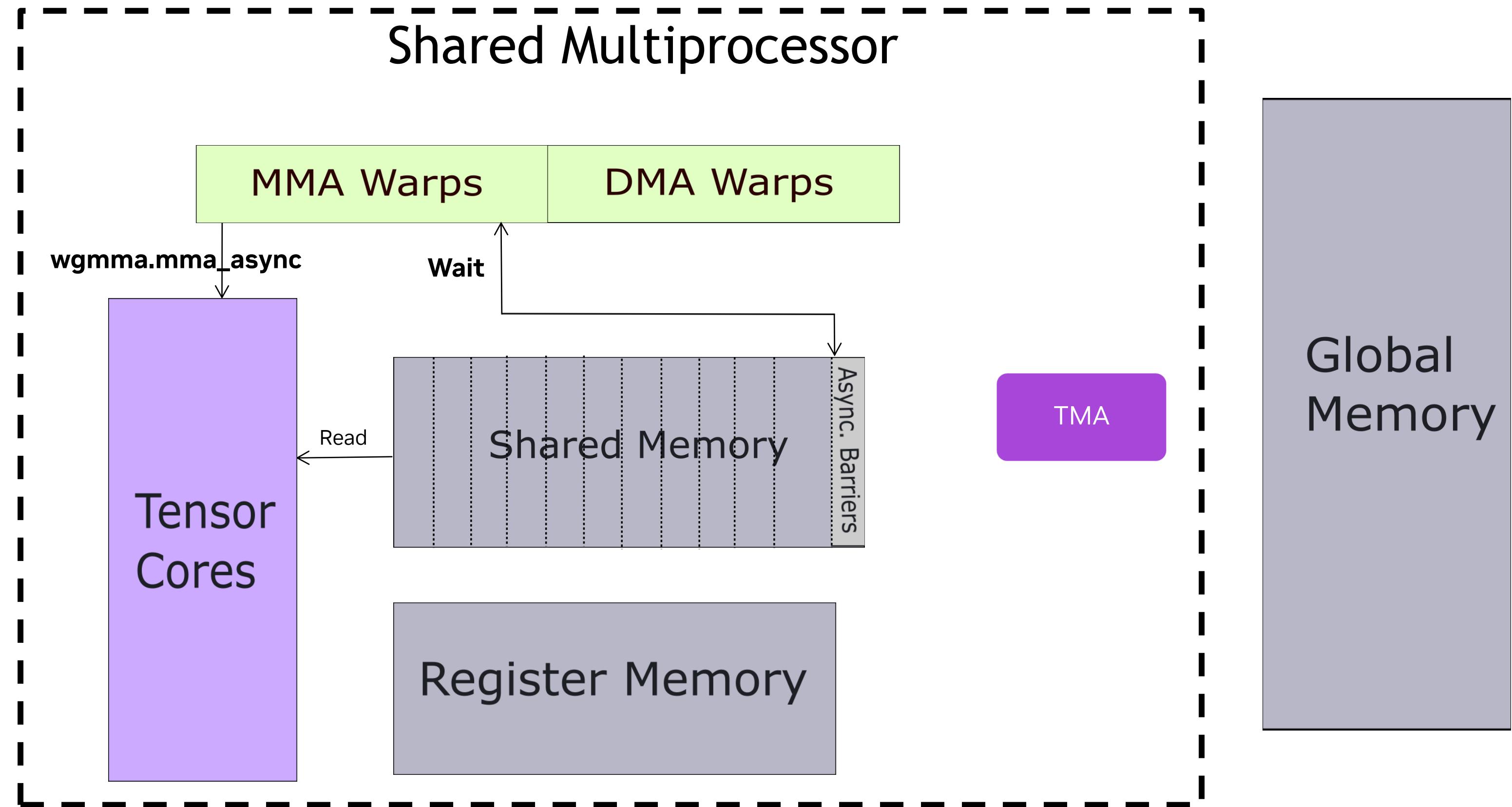
Async. Warp Specialized Kernel



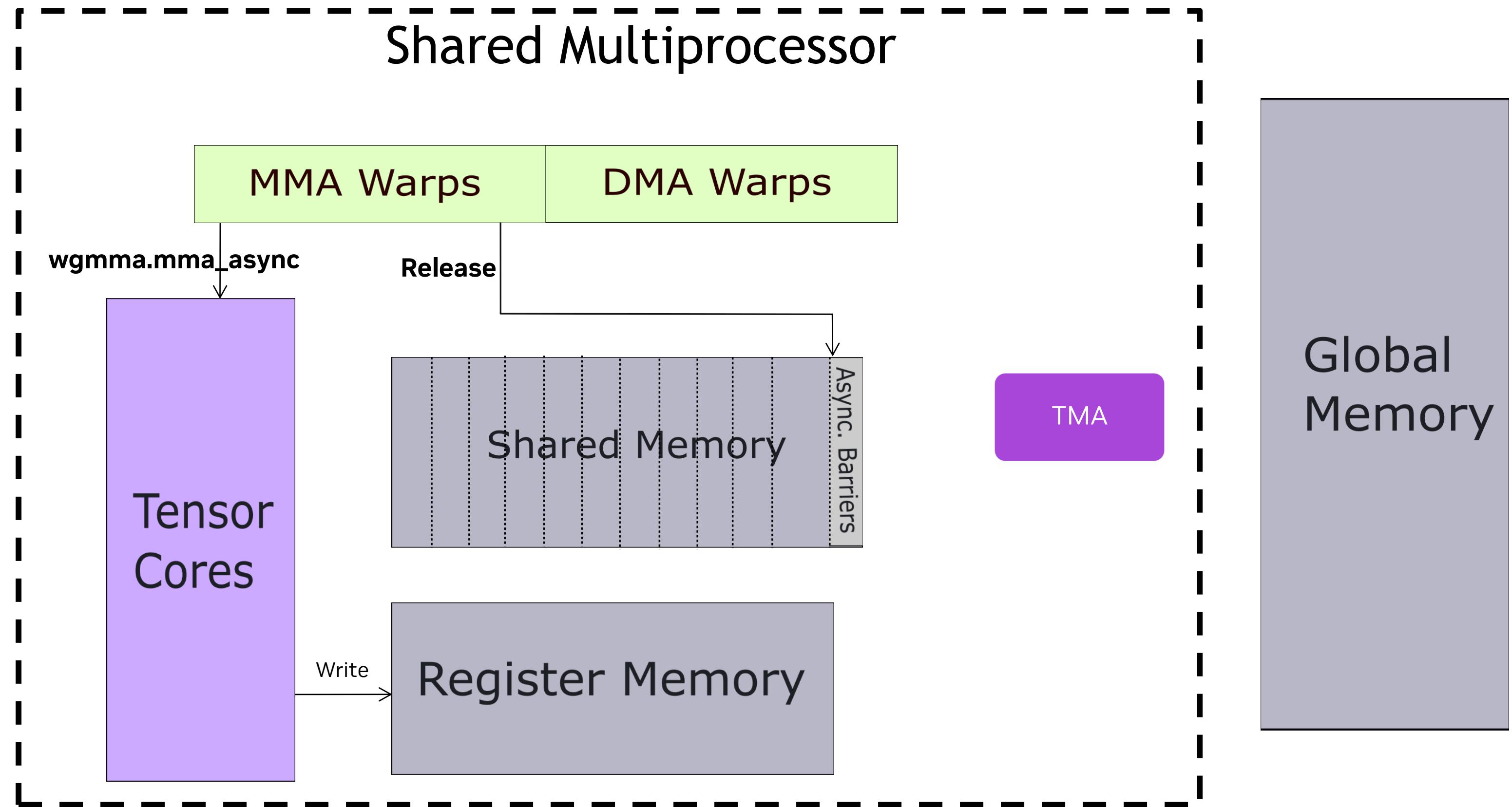
Async. Warp Specialized Kernel



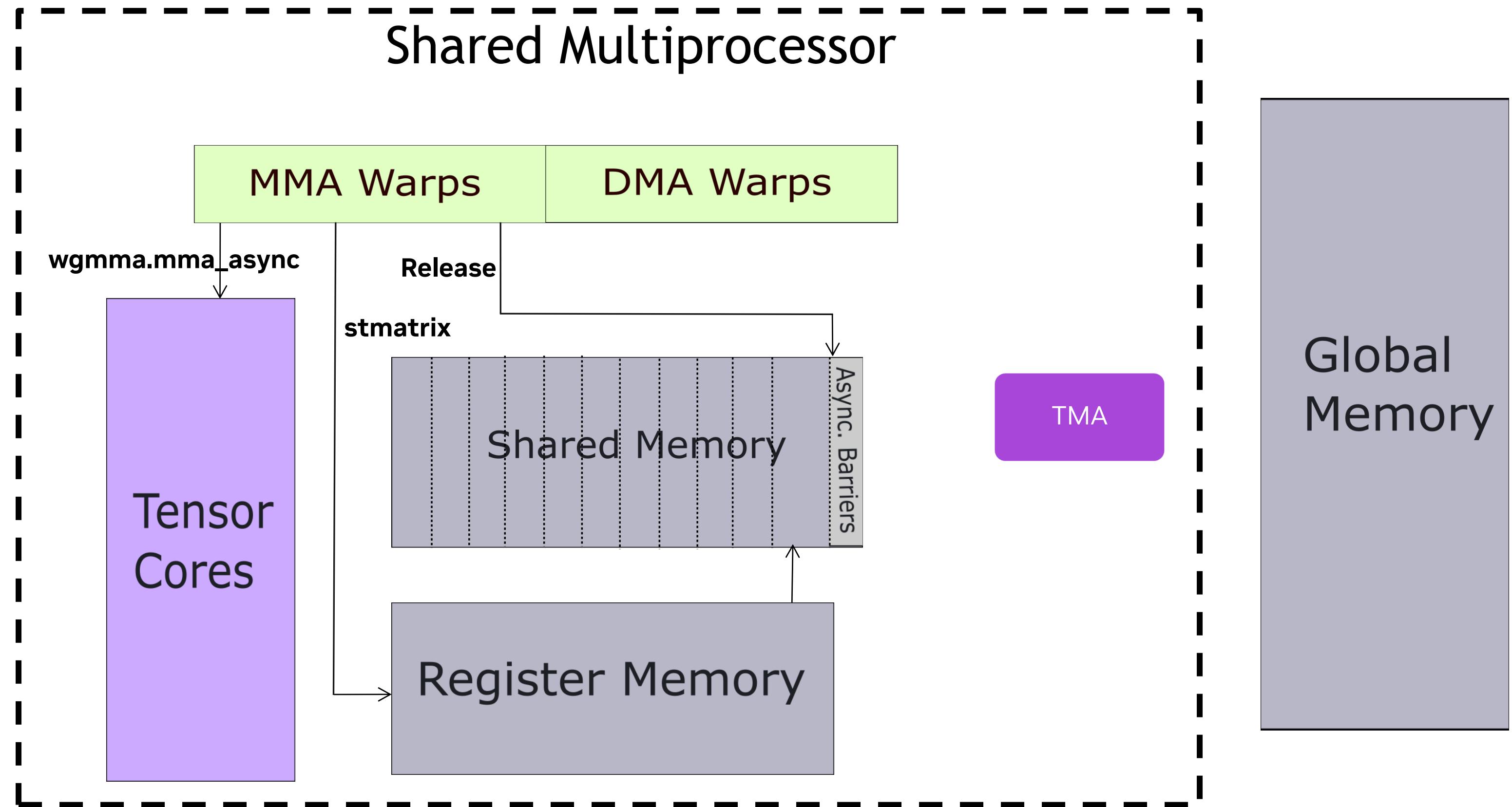
Async. Warp Specialized Kernel



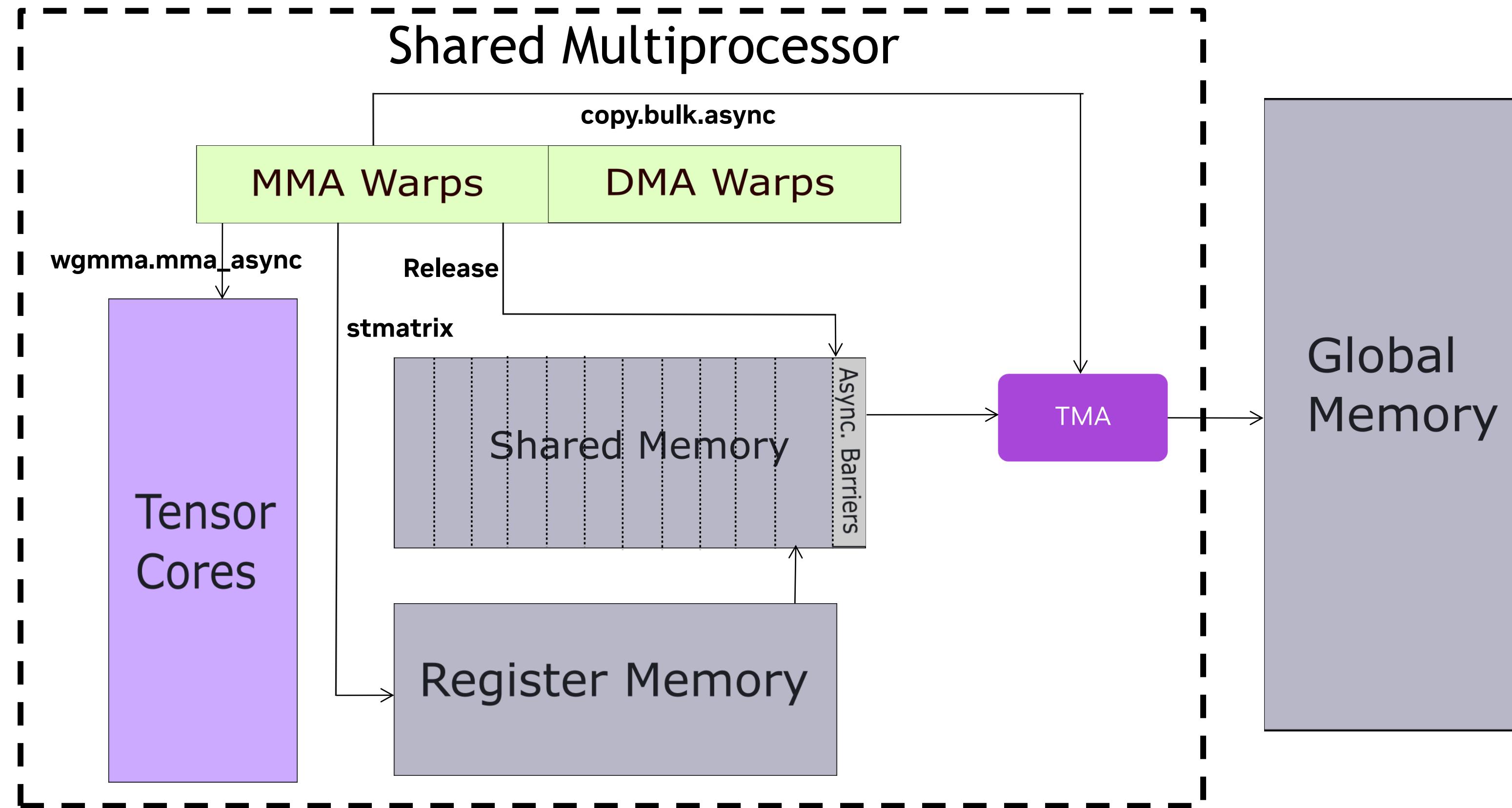
Async. Warp Specialized Kernel



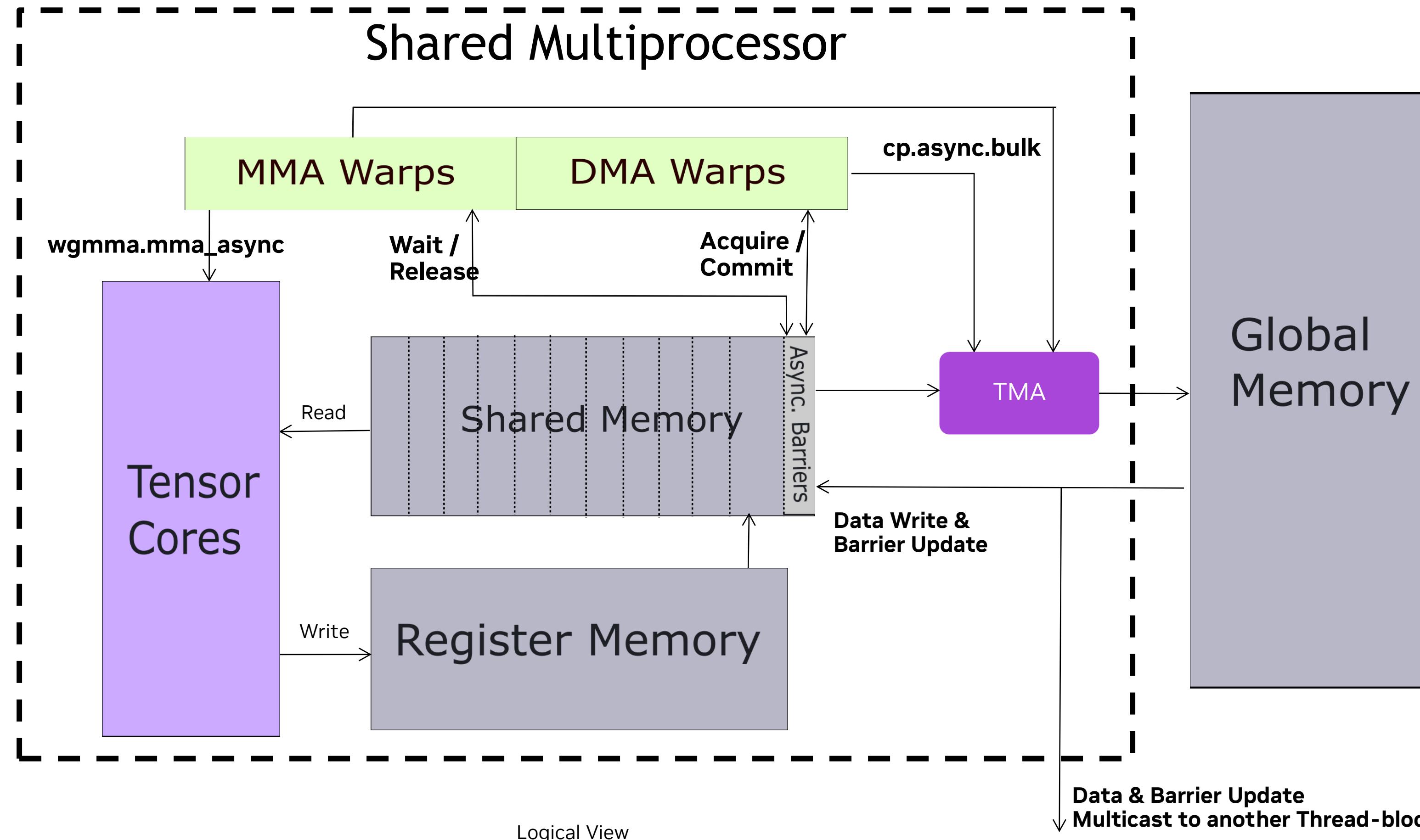
Async. Warp Specialized Kernel



Async. Warp Specialized Kernel



Async. Warp Specialized Kernel



Compact Mainloop Representation

```
// DMA Mainloop
for ( ; k_tile_count > 0; --k_tile_count) {
    // LOCK smem_pipe_write for _writing_
    pipeline.producer_acquire(smem_pipe_write);

    // Copy gmem to smem for *k_tile_iter
    int write_stage = smem_pipe_write.index();
    using BarrierType = typename
        MainloopPipeline::ValueType;
    BarrierType* tma_barrier =
        pipeline.producer_get_barrier(write_stage);

    copy(tma_load_a.with(*tma_barrier, mcast_mask_a),
          tAgA(_,_,*k_tile_iter), tAsA(_,_,_write_stage));
    copy(tma_load_b.with(*tma_barrier, mcast_mask_b),
          tBgB(_,_,*k_tile_iter), tBsB(_,_,_write_stage));
    ++k_tile_iter;

    // Advance smem_pipe_write
    ++smem_pipe_write;
}
```

```
// MMA Mainloop
for ( ; k_tile_count > 0; --k_tile_count) {
    // WAIT on smem_pipe_read until its data are available
    pipeline.consumer_wait(smem_pipe_read);

    // Compute on k_tile
    int read_stage = smem_pipe_read.index();
    warpgroup_fence_operand(accum);
    warpgroup_arrive();
    cute::gemm(tiled_mma, tCrA(_,_,_read_stage), tCrB(_,_,_read_stage),
               accum);
    warpgroup_commit_batch();

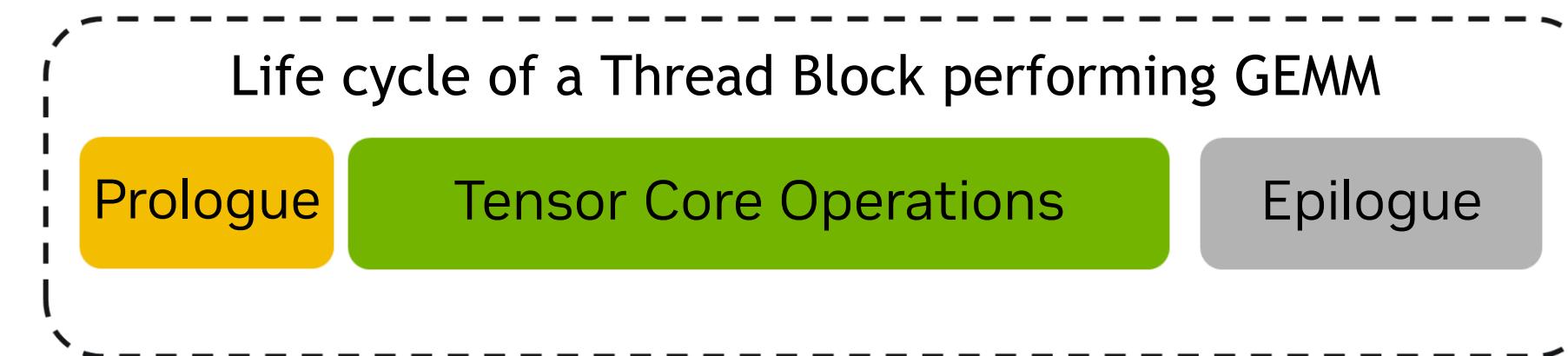
    // Wait on the GMMA barrier for K_PIPE_MMAS (or fewer) outstanding
    warpgroup_wait<K_PIPE_MMAS>();
    warpgroup_fence_operand(accum);

    // UNLOCK smem_pipe_release, done_computing_on it
    pipeline.consumer_release(smem_pipe_release);

    // Advance smem_pipe_read and smem_pipe_release
    ++smem_pipe_read;
    ++smem_pipe_release;
}
```

Persistent Warp Specialization

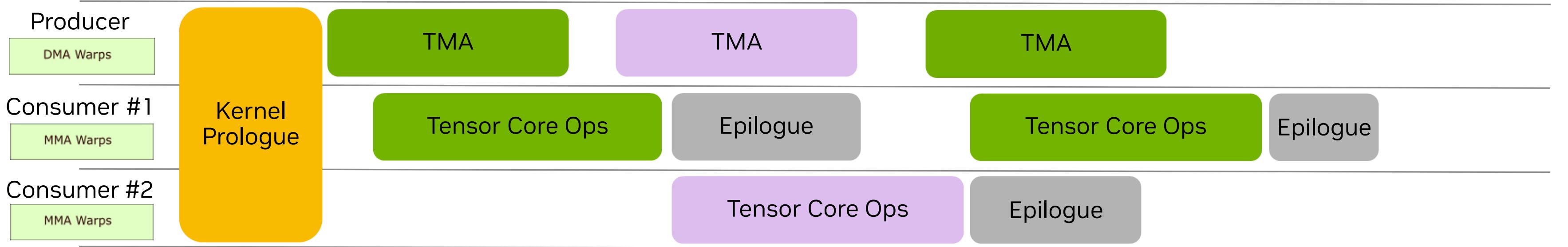
A way to hide non-Tensor Core Operations



- The Prologue and Epilogue are components of the GEMM kernel which involve non-tensor core operations and often latency and/or bandwidth.
- A typical solution to hiding these in past architectures has been to run multiple thread-blocks per SM.
- Increases in Tensor Core throughput relatively, has led to deeper software pipelines which limit our ability to run multiple thread-blocks per SM.
- Thus, in Hopper we introduce a new way of tackling this problem via persistent thread blocks issuing collective Mainloops across multiple output tiles and multiple warp-groups (WGs) with specialized functionality.
- Each WG is assigned a certain role, such as Data Producer or Data Consumer.

Warp-Specialized and Persistent Kernels

- Warp specialization: kernel marshals threads into totally separate execution paths
 - TMA-issuing threads only perform copies
 - MMA-issuing threads only perform math & Epilogues
- Persistent grids amortize launch & prologue costs in addition to overlapping MMA mainloop and epilogue execution.



Producer	Consumer #1	Consumer #2
PersistentTileSchedulerSm90 scheduler(problem_shape, blk_shape, cluster_shape)		
// Data in via TMA while (work_tile_info.is_valid_tile) { collective_mainloop.dma() scheduler.advance_to_next_work() work_tile_info = scheduler.get_current_work() }	// Mainloop, epilogue, and data out while (work_tile_info.is_valid_tile) { collective_mainloop.mma() scheduler.advance_to_next_work(NumConsumers) work_tile_info = scheduler.get_current_work() }	



Agenda

- Hopper Architecture
- CuTe
- CUTLASS 3.0
- CUTLASS Python
- Conclusion

CUTLASS Python interface

- Experimental release coming in CUTLASS 3.1
- Goals:
 - Easy declaration, emission, and compilation of CUTLASS kernels
 - Catch common compilation and runtime errors in Python to ease debugging
 - Easily integrate CUTLASS kernels into DL frameworks (e.g., PyTorch)

Goal: easy declaration, emission, compilation of CUTLASS kernels

C++ interface

```
using Gemm =  
typename gemm::kernel::DefaultGemmUniversal<  
    half_t, layout::RowMajor,  
    ComplexTransform::kNone, 8,  
    half_t, layout::RowMajor,  
    ComplexTransform::kNone, 8,  
    half_t, layout::RowMajor,  
    half_t, arch::OpClassTensorOp, arch::Sm80,  
    gemm::GemmShape<256, 128, 64>,  
    gemm::GemmShape<64, 64, 64>,  
    gemm::GemmShape<16, 8, 16>,  
    epilogue::thread::LinearCombination<  
        half_t, 8, half_t, half_t>,  
        gemm::threadblock::GemmIdentityThreadblockSwizzle<1>,  
        3, arch::OpMultiplyAdd  
>::GemmKernel;
```

Python interface

```
plan = cutlass.op.Gemm(element=torch.float16,  
                      layout=cutlass.LayoutType.RowMajor)  
  
plan.swizzling_functor =  
    cutlass.swizzle.ThreadblockSwizzleStreamK  
  
plan.activation = cutlass.epilogue.relu
```



Change swizzling functor

Add fused ReLU

Goal: catch common compilation and runtime errors in Python

C++ compilation

```
mma_tensor_op_tile_iterator.h(1163): error: incomplete type is not
allowed
    using Fragment = typename Base::Fragment;
        ^
detected during:
instantiation of class
"cutlass::gemm::warp::MmaTensorOpMultiplicandTileIterator<Shape_>,
Operand_, Element_,
cutlass::layout::RowMajorTensorOpMultiplicandCongruous<cutlass::sizeof_bis
ts<Element_>::value, <expression>>, InstructionShape_, OpDelta_, 32,
PartitionsK_> [with Shape_=cutlass::MatrixShape<32, 64>,
Operand_=cutlass::gemm::Operand::kB, Element_=ElementInputB,
InstructionShape_=cutlass::MatrixShape<32, 8>, OpDelta_=1,
PartitionsK_=1]" at line include/cutlass/gemm/warp/mma_tensor_op.h
instantiation of class
"cutlass::gemm::warp::MmaTensorOp<Shape_, ElementA_, LayoutA_, ElementB_>,
LayoutB_, ElementC_, LayoutC_, Policy_, PartitionsK_,
AccumulatorsInRowMajor, Enable> [with Shape_=ShapeMMAWarp,
ElementA_=ElementInputA,
LayoutA_=cutlass::layout::RowMajorTensorOpMultiplicandCrosswise<8, 32>,
ElementB_=ElementInputB,
LayoutB_=cutlass::layout::RowMajorTensorOpMultiplicandCongruous<8, 128>,
ElementC_=ElementAccumulator, LayoutC_=LayoutOutput,
Policy_=cutlass::gemm::warp::MmaTensorOpPolicy<cutlass::arch::Mma<cutlass
::gemm::GemmShape<16, 8, 32>, 32, int8_t, cutlass::layout::RowMajor,
int8_t, cutlass::layout::ColumnMajor, int, cutlass::layout::RowMajor,
cutlass::arch::OpMultiplyAddSaturate>, cutlass::MatrixShape<1, 1>>,
PartitionsK_=1, AccumulatorsInRowMajor=false, Enable=bool]" at line
include/cutlass/gemm/threadblock/mma_base.h
```

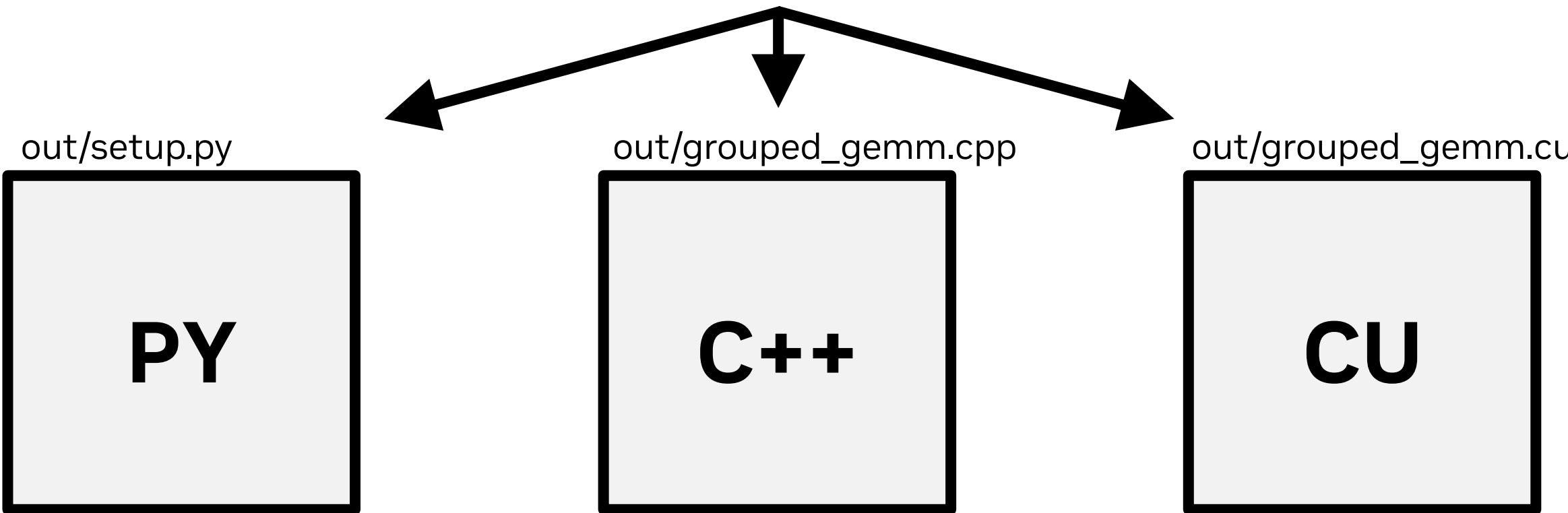
Python runtime exception

```
Exception: Unsupported operation class
OpcodeClass.TensorOp for CC 80 and data type combination
(DataType.s8, DataType.s8, DataType.s8) and layout
combination (LayoutType.RowMajor, LayoutType.RowMajor).
```

Goal: easy integration into DL frameworks (e.g., PyTorch)

```
plan = cutlass.GroupedGemm(element=torch.float16,  
                           layout=cutlass.LayoutType.RowMajor)
```

```
cutlass.emit.pytorch(plan.construct(), name='grouped_gemm',  
                     cc=80, sourcedir='out')
```



```
> python setup.py install
```

```
import grouped_gemm  
Ds = grouped_gemm.run([A0, A1], [B0, B1])
```



Agenda

- Hopper Architecture
- CuTe
- CUTLASS 3.0
- CUTLASS Python
- Conclusion

Conclusion

CUTLASS

• CUTLASS Roadmap

- CUTLASS 2.11 was released November 2022 – will serve as the last CUTLASS 2.x release
- CUTLASS 3.0 was released January 2023
- CUTLASS 3.1 to become available in April 2023.

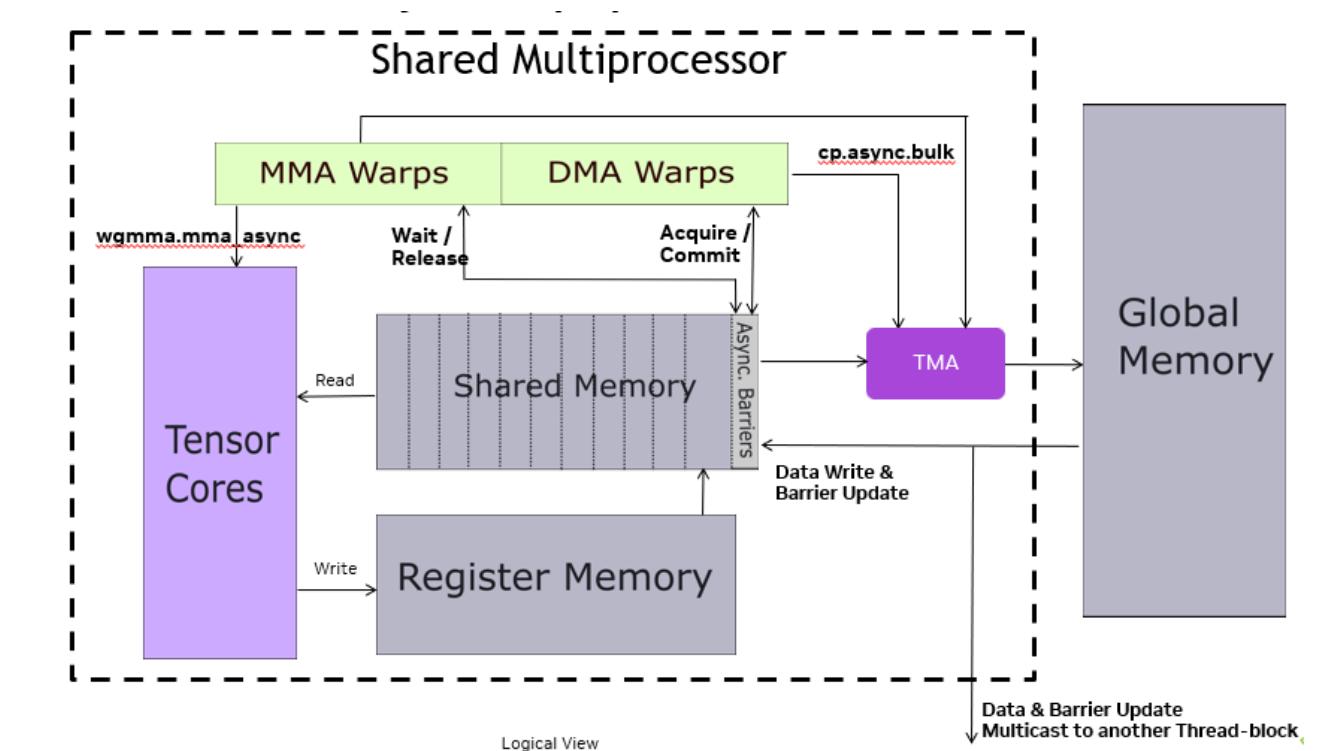
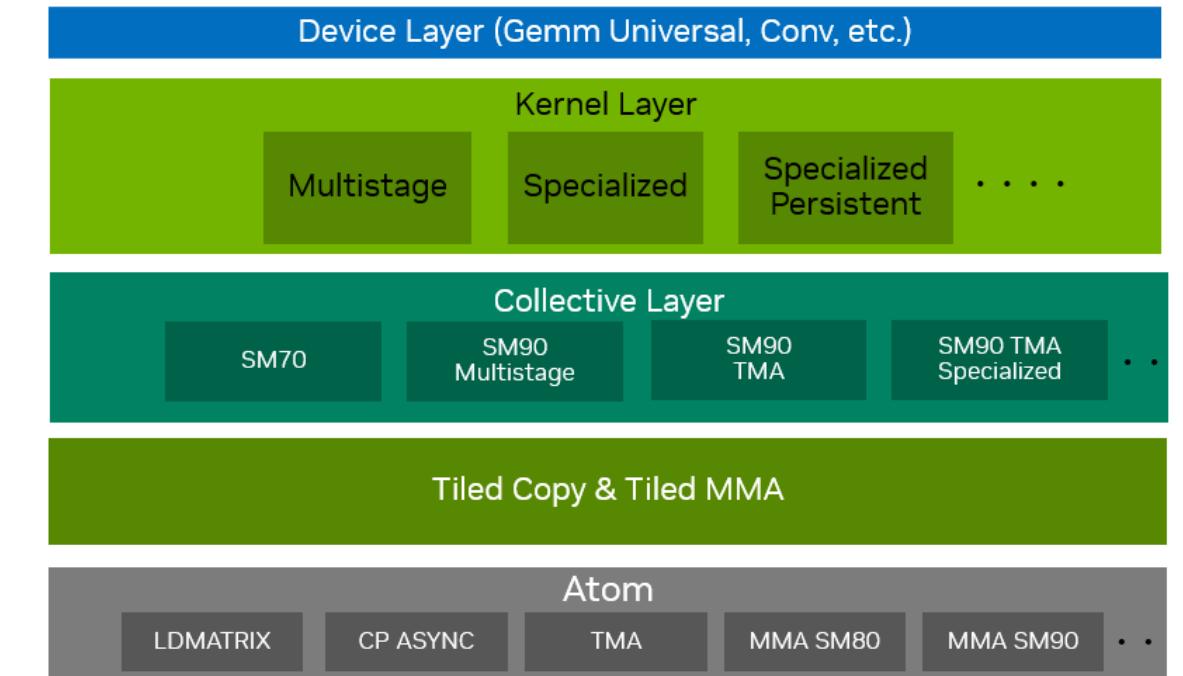
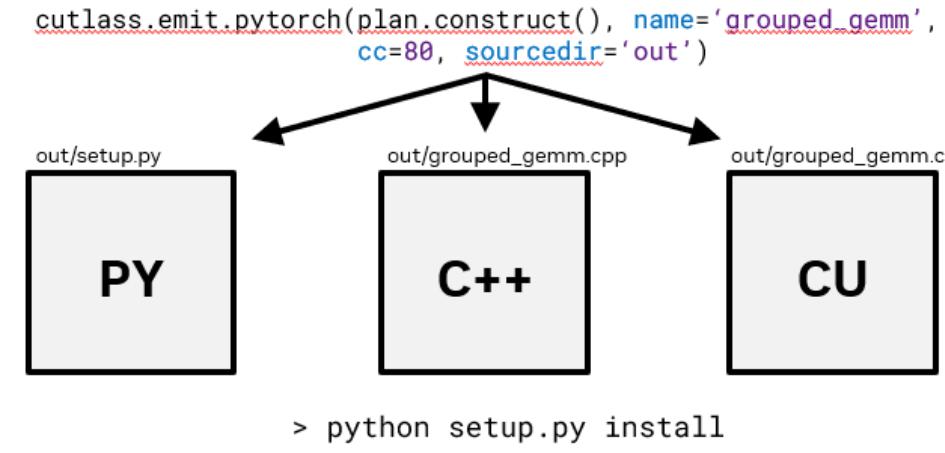
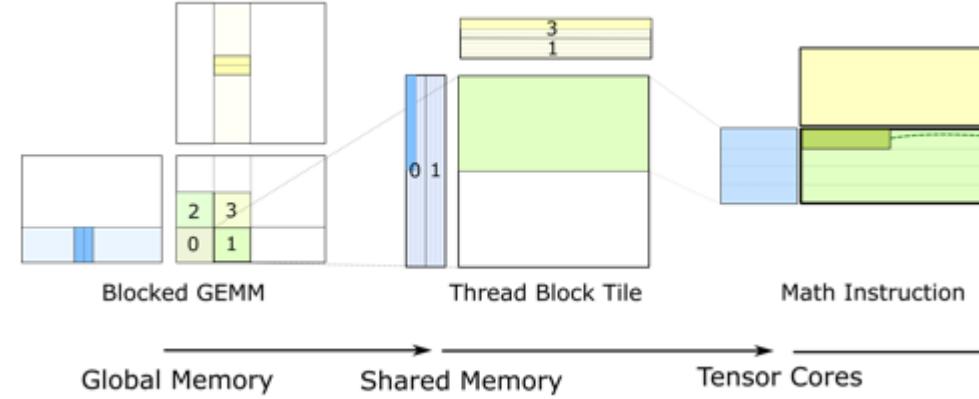
• CUTE

- A new way to think about Tensor and Layouts
 - Greatly simplifies address computation logic
- Is the backend for CUTLASS 3

• CUTLASS 3.0

- Provides flexible abstractions for users to compose customized kernels and collectives
- Implements optimal computations using 4th generation Tensor Cores.
- Aids async Persistent producer consumer model of synchronization.

<https://github.com/NVIDIA/cutlass>



REFERENCES

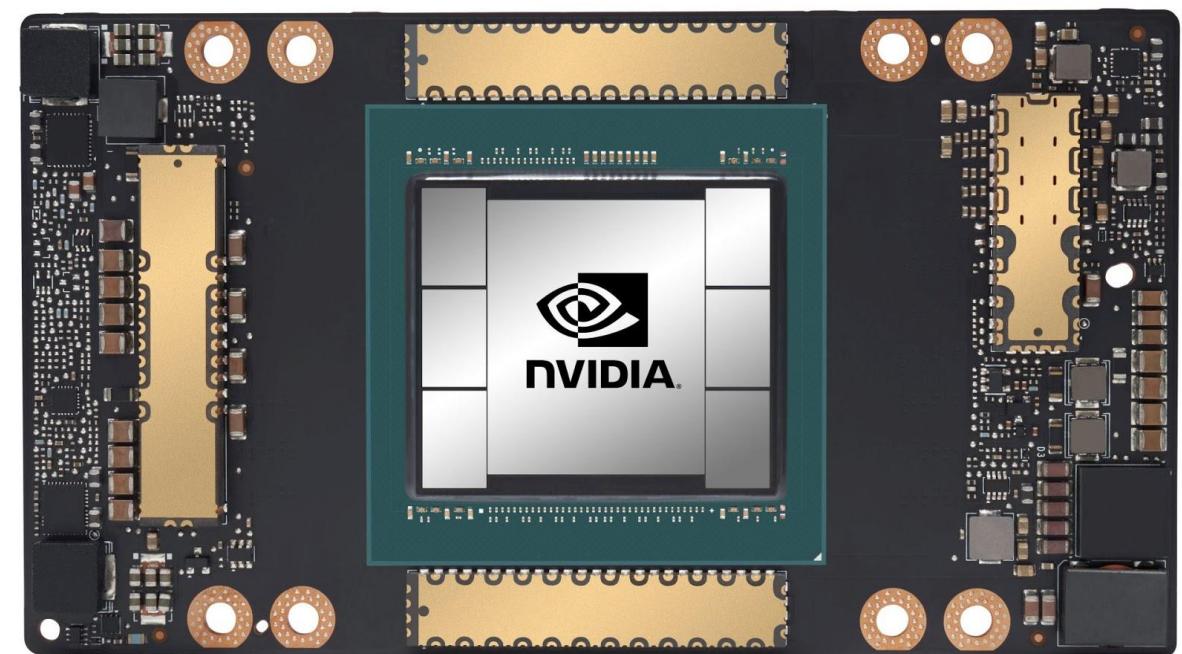
NVIDIA Hopper Architecture & CUDA :

“Inside the NVIDIA Hopper Architecture” ([GTC 2022](#))

“CUDA New Features and Beyond” ([GTC 2022](#), GTC 2023)

“Optimizing Applications for Hopper Architecture” (GTC 2023)

“NVIDIA Hopper Architecture In-Depth” ([blog post](#))



PTX ISA

The programming guide to using Parallel Thread Execution and Instruction Set Architecture.
([CUDA documentation](#)).

CUTLASS

<https://github.com/NVIDIA/cutlass> (open source software, New BSD license)

- CUTLASS Parallel For All blog post

Prior GTC CUTLASS Talks : [GTC'18](#), [GTC'19](#), [GTC'20](#), [GTC'21](#), [GTC'22](#) , [GTC'22](#)

