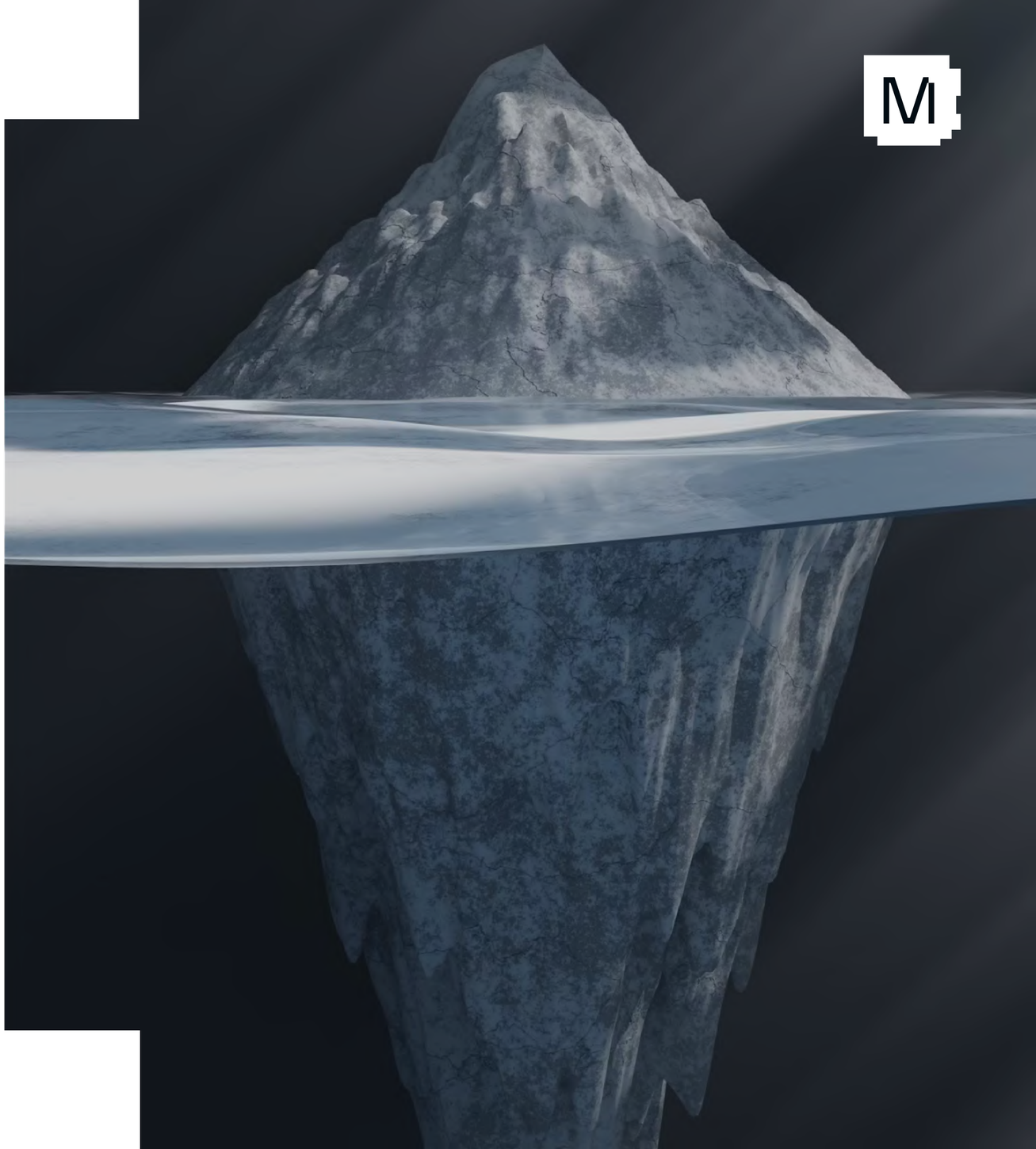
AI Systems Performance Engineering

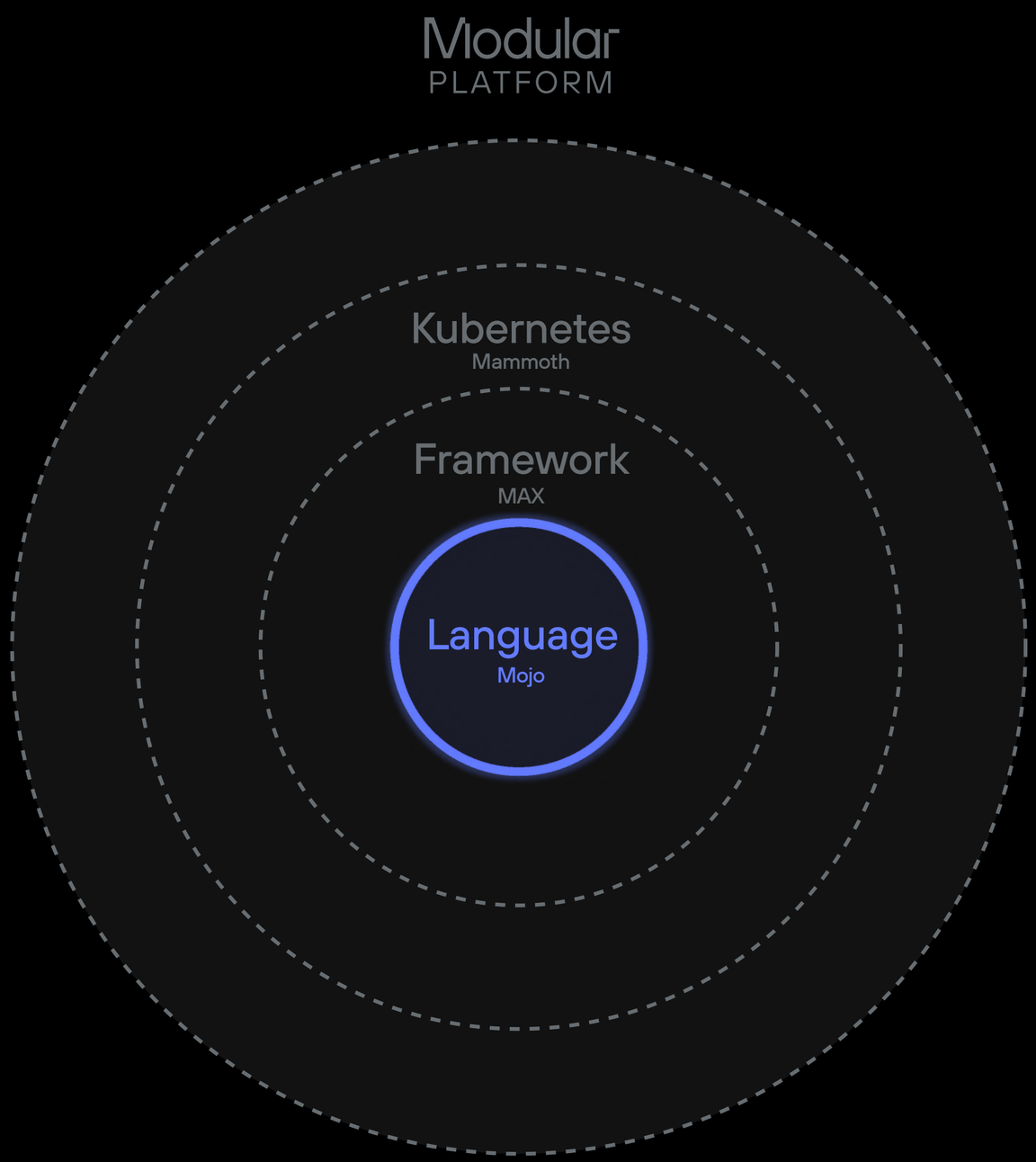
Modular

Abdul Dakkak
Head of GenAI

Agenda

- Familiarize with Modular technology Stack
- Learn Mojo programing basics
- Understand the benefits of Modular proposal
- Showcase performance of Modular on Blackwell systems
- Understand how to write a fast Matrix multiplication for Blackwell



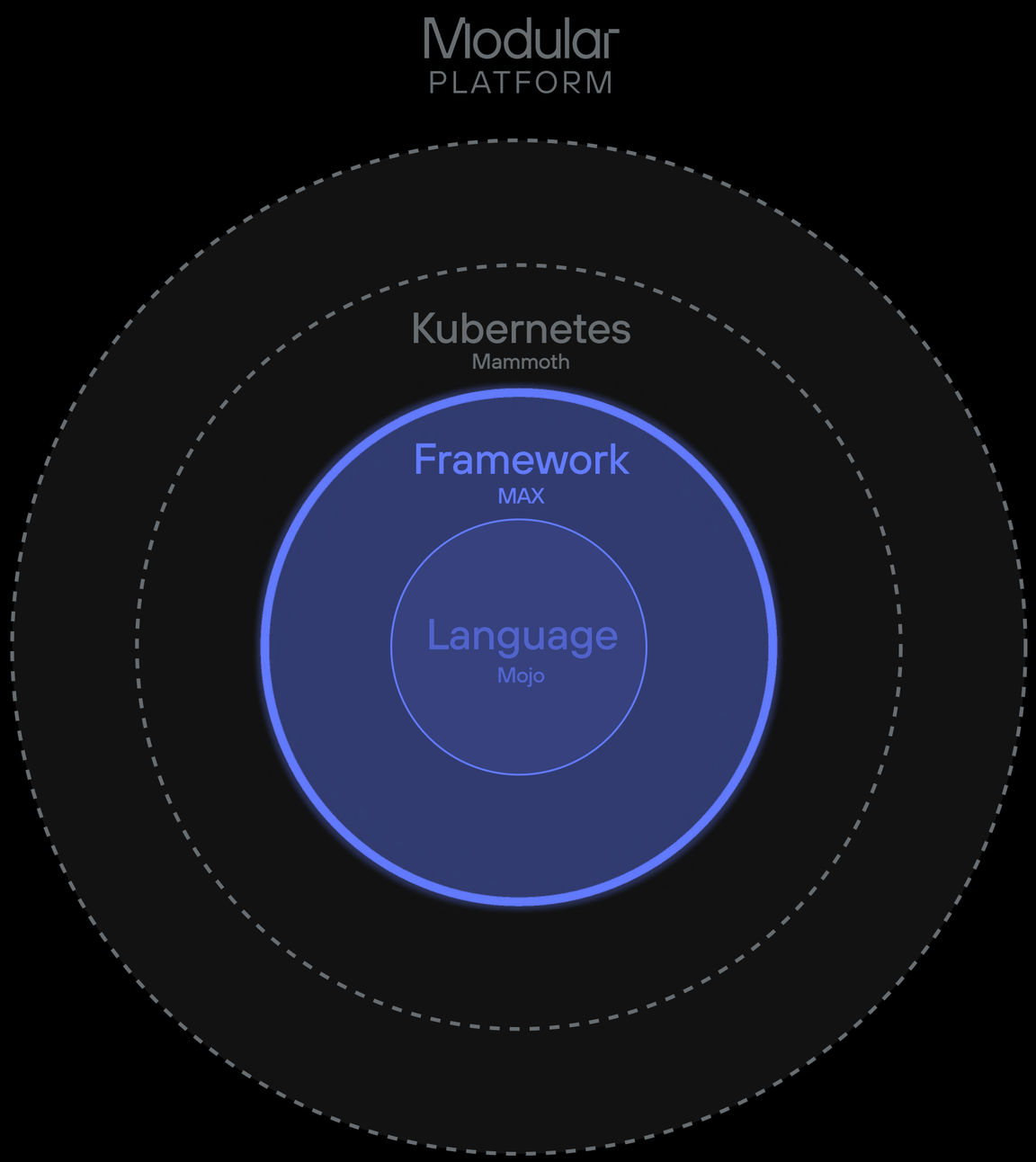


Mammoth Cluster

MAX Framework

Mojo🔥 Language

Code for a single device
Get the full power of the hardware
Easy to learn and scale



Mammoth Cluster

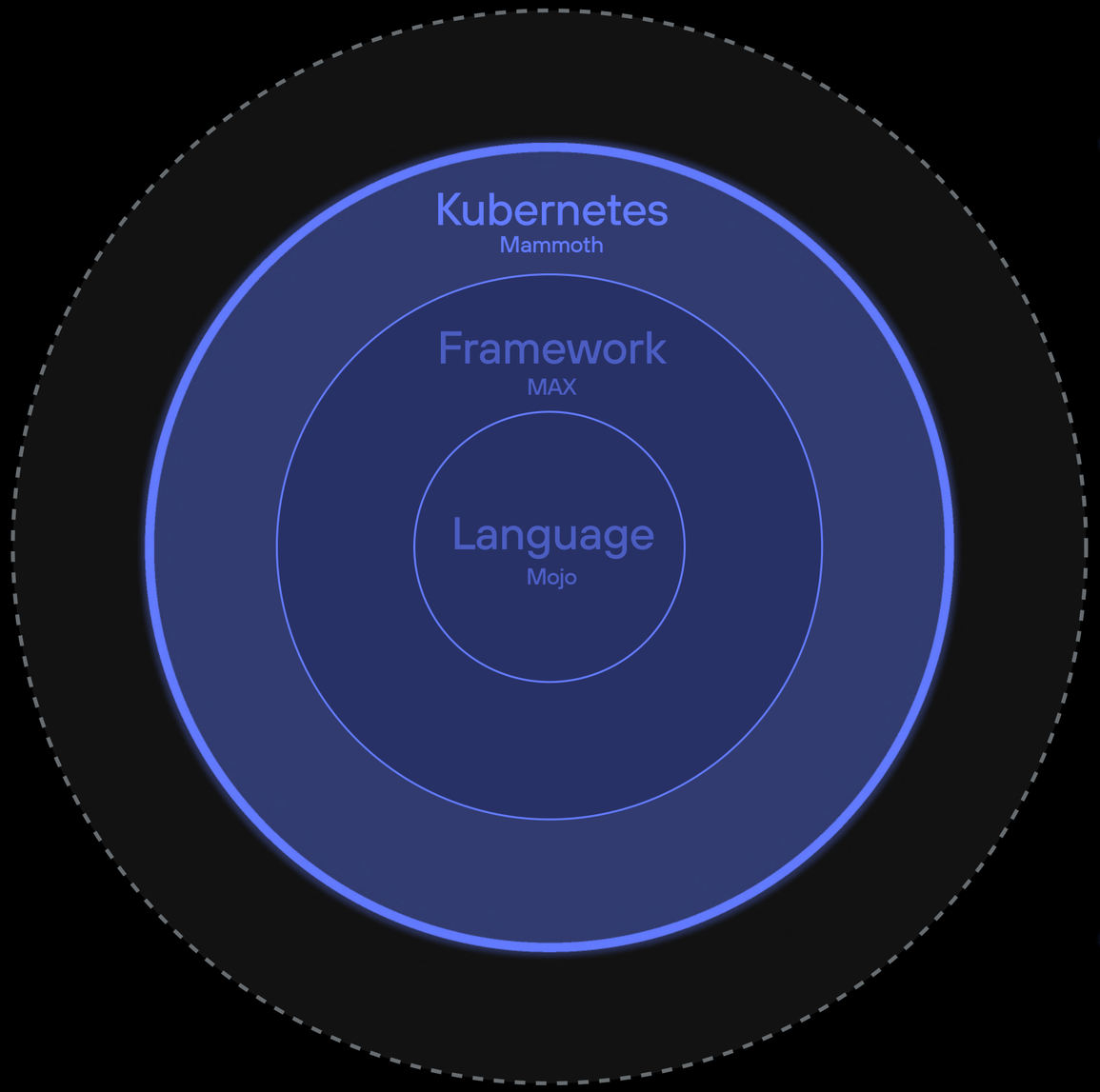
MAX 👩‍🚀 Framework

Deploy SOTA GenAI services on one node
AI Inference Framework, Graph Compiler + APIs

`SERVE` | `ENGINE` | `KERNELS`

Mojo Language

Modular
PLATFORM



Mammoth Cluster

Scalable GenAI cluster inference

SELF-HOSTED | JOINTLY-MANAGED | SCALABLE

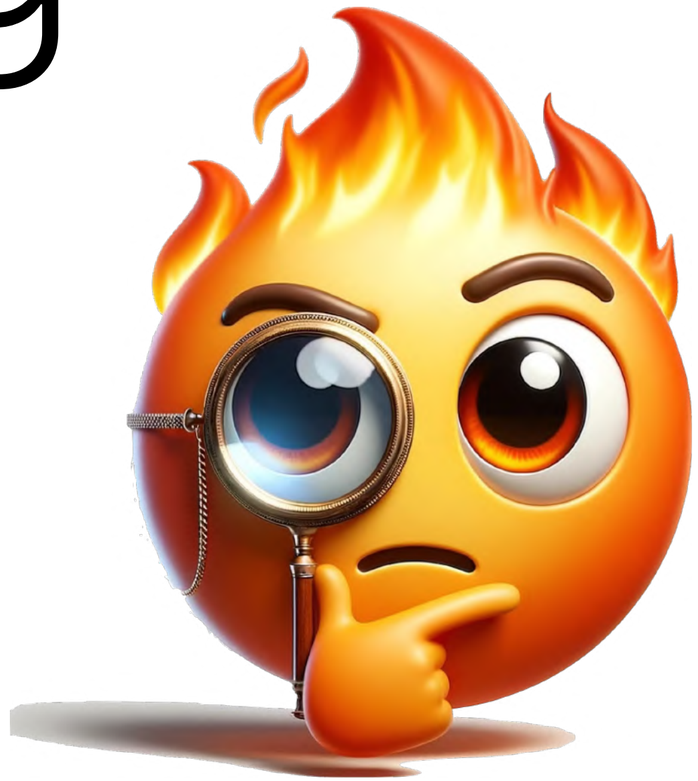
MAX Framework

Mojo Language

Digging into Mojo 🔥

The Industry has a Problem

A look at existing performance libraries



Many are written in assembly...

... so much for abstraction even across members of an architectural family ...

```
lea     rax,[rdx+r8*2]
vpmovzxbw ymm4,XMMWORD PTR [rdx]
vpmovzxbw ymm5,XMMWORD PTR [rdx+r8]
vpmovzxbw ymm6,XMMWORD PTR [rax]
vpmovzxbw ymm7,XMMWORD PTR [rax+r8]
lea     rax,[rcx+r11*4]
vmovdqu YMMWORD PTR [rcx],ymm4
vmovdqu YMMWORD PTR [rcx+r11*2],ymm5
vmovdqu YMMWORD PTR [rax],ymm6
vmovdqu YMMWORD PTR [rax+r11*2],ymm7
vpaddw  ymm0,ymm0,ymm4
vpaddw  ymm1,ymm1,ymm5
vpaddw  ymm2,ymm2,ymm6
vpaddw  ymm3,ymm3,ymm7
add     rdx,16
add     rcx,16*2
sub     rbx,16
```

C++ Templates

```
static constexpr auto GemmDefault =
    ck::tensor_operation::device::GemmSpecialization::Default;

using DeviceGemmInstance = ck::tensor_operation::device::DeviceGemmXdl<
    ADataType, BDataType, CDataType, AccDataType, ALayout, BLayout, CLayout,
    AElementOp, BElementOp, CElementOp, GemmDefault, 256, 128, 128, 4, 2, 16,
    16, 4, 4, S<4, 64, 1>, S<1, 0, 2>, S<1, 0, 2>, 2, 2, 2, true, S<4, 64, 1>,
    S<1, 0, 2>, S<1, 0, 2>, 2, 2, 2, true, 7, 1>;

using ReferenceGemmInstance =
    ck::tensor_operation::host::ReferenceGemm<ADataType, BDataType, CDataType,
                                              AccDataType, AElementOp,
                                              BElementOp, CElementOp>;

#include "run_gemm_example.inc"
```

Source: Composable Kernels

C++ DSL for ASM

```
L(labels[4]);
test(K, 2);
jle(labels[5], T_NEAR);
innerkernel2(unroll_m, unroll_n, isLoad1Unmasked, isLoad2Unmasked, isDirect,
             isCopy, useFma, reg00, reg01, reg02, reg03, reg04, reg05,
             reg06, reg07, reg08, reg09, reg10, reg11, reg12, reg13, reg14,
             reg15, reg16, reg17, reg18, reg19, reg20, reg21, reg22, reg23);
align(16);

L(labels[5]);
if (unroll_m == 16) {
    if (unroll_n <= 3) {
        vaddps(reg00, reg00, reg12);
        vaddps(reg01, reg01, reg13);
        vaddps(reg02, reg02, reg14);
        vaddps(reg06, reg06, reg18);
        vaddps(reg07, reg07, reg19);
        vaddps(reg08, reg08, reg20);
    }
}
```

Source: OneDNN

Python program to generate ASM

```
for iui in range(0, innerUnroll):
    for idx1 in range(0, kernel["ThreadTile1"]):
        for idx0 in range(0, kernel["ThreadTile0"]):
            vars["idx0"] = idx0
            vars["idx1"] = idx1
            vars["a"] = idx0 if writer.tPB["tile01Idx"] else idx1
            vars["b"] = idx1 if writer.tPB["tile01Idx"] else idx0
            vars["iui"] = iui

            vars["cStr"] = "v[vgprValuC + {idx0} + {idx1}*{ThreadTile0}]".format_map(vars)
            vars["aStr"] = "v[vgprValuA_X{m}_I{iui} + {a}]".format_map(vars)
            vars["bStr"] = "v[vgprValuB_X{m}_I{iui} + {b}]".format_map(vars)

            if instruction == "v_fma_f32":
                kStr += "v_fma_f32 {cStr}, {aStr}, {bStr}, {cStr}{endLine}".format_map(vars)
            else:
                kStr += "{instruction} {cStr}, {aStr}, {bStr}{endLine}".format_map(vars)

            kStr += priority(writer, 1, "Raise priority while processing macs")
```

Source: Tensile

Python template to generate C++

```
const __m128i vsign_mask =
    _mm_load_si128((const __m128i*)params->${PARAMS_STRUCT}.sign_mask);
const __m256 vsat_cutoff = _mm256_load_ps(params->${PARAMS_STRUCT}.sat_cutoff);
const __m256 vlog2e = _mm256_load_ps(params->${PARAMS_STRUCT}.log2e);
const __m256 vmagic_bias = _mm256_load_ps(params->${PARAMS_STRUCT}.magic_bias);
const __m256 vminus_ln2 = _mm256_load_ps(params->${PARAMS_STRUCT}.minus_ln2);
$for i in reversed(range(2, P + 1))
    : const __m256 vc${i} = _mm256_load_ps(params->${PARAMS_STRUCT}.c${i});
$if P != H + 1 : const __m256 vminus_one =
    _mm256_load_ps(params->${PARAMS_STRUCT}.minus_one);
const __m256 vtwo = _mm256_load_ps(params->${PARAMS_STRUCT}.two);
$if P == H + 1 : const __m256 vminus_one =
    _mm256_load_ps(params->${PARAMS_STRUCT}.minus_one);
```

Source: XNNPack

C++ Metaprogramming

```
Building CUDA object
examples/46_depthwise_simt_conv2dfprop/CMakeFiles/46_depthwise_simt_conv2dfprop.dir/depthwise_simt_conv2dfprop.cu.
o /home/clattner/personal/cutlass/include/cutlass/conv/kernel/direct_convolution.h(95): error: incomplete type is
not allowed detected during: instantiation of class "cutlass::conv::kernel::DirectConvolutionParams<Mma_,
Epilogue_, ThreadblockSwizzle_, ConvOperator, Arguments_, ConvOutputIteratorParameter_, ConvProblemSize_,
GroupMode_, ThreadBlockOutputShape_> [with
Mma_=cutlass::conv::threadblock::DepthwiseFpropDirectConvMultipleStage<ThreadblockShape,
cutlass::conv::threadblock::DepthwiseFpropActivationDirect2dConvTileAccessIteratorFixedStrideDilation<cutlass::Mat
rixShape<64, 64>, ThreadBlockOutputShape, StrideShape, DilationShape, cutlass::conv::TensorNHWCShape<1, 10, 10,
64>, ElementInputA, LayoutInputA,
cutlass::transform::PitchLinearStripminedThrea          ss::PitchLinearShape<64, 100>, 128, 4>,
cutlass::AlignedArray<ElementInputA, 4, 16>
cutlass::transform::threadblock::RegularTi              rectConv<cutlass::MatrixShape<100, 64>,
ElementInputA, cutlass::layout::RowMajor,
cutlass::transform::PitchLinearStripminedThre           s::PitchLinearShape<64, 100>, 128, 4>, false, 16>,
cutlass::arch::CacheOperation::Global,
cutlass::conv::threadblock::DepthwiseFpropFilterDirectConvTileAccessIteratorOptimized<cutlass::MatrixShape<64,
9>, ElementInputB, LayoutInputB, cutlass::transform::PitchLinearStripminedThreadMap<cutlass::PitchLinearShape<64,
9>, 128, 4>, cutlass::AlignedArray<ElementInputA, 4, 16>>,
cutlass::transform::threadblock::RegularTileAccessIteratorDirectConv<cutlass::MatrixShape<9, 64>, ElementInputB,
cutlass::layout::RowMajor, 0, cutlass::transform::PitchLinearStripminedThreadMap<cutlass::PitchLinearShape<64,
9>, 128, 4>, false, 16>, cutlass::arch::CacheOperation::Global,
cutlass::conv::threadblock::DepthwiseDirectConvMmaPolicy<cutlass::conv::warp::MmaDepthwiseDirectConvSimt<WarpShape
, FilterShape,
```

Pervasive suffering

Maintainability, debugging, tooling, ...

Enter Mojo 🔥

Mojo 🔥 at a glance

Pythonic systems programming language

- CLI toolchain + VS Code support
- Strong Python and C interop
- Extensive generic programming, type system, and memory safety

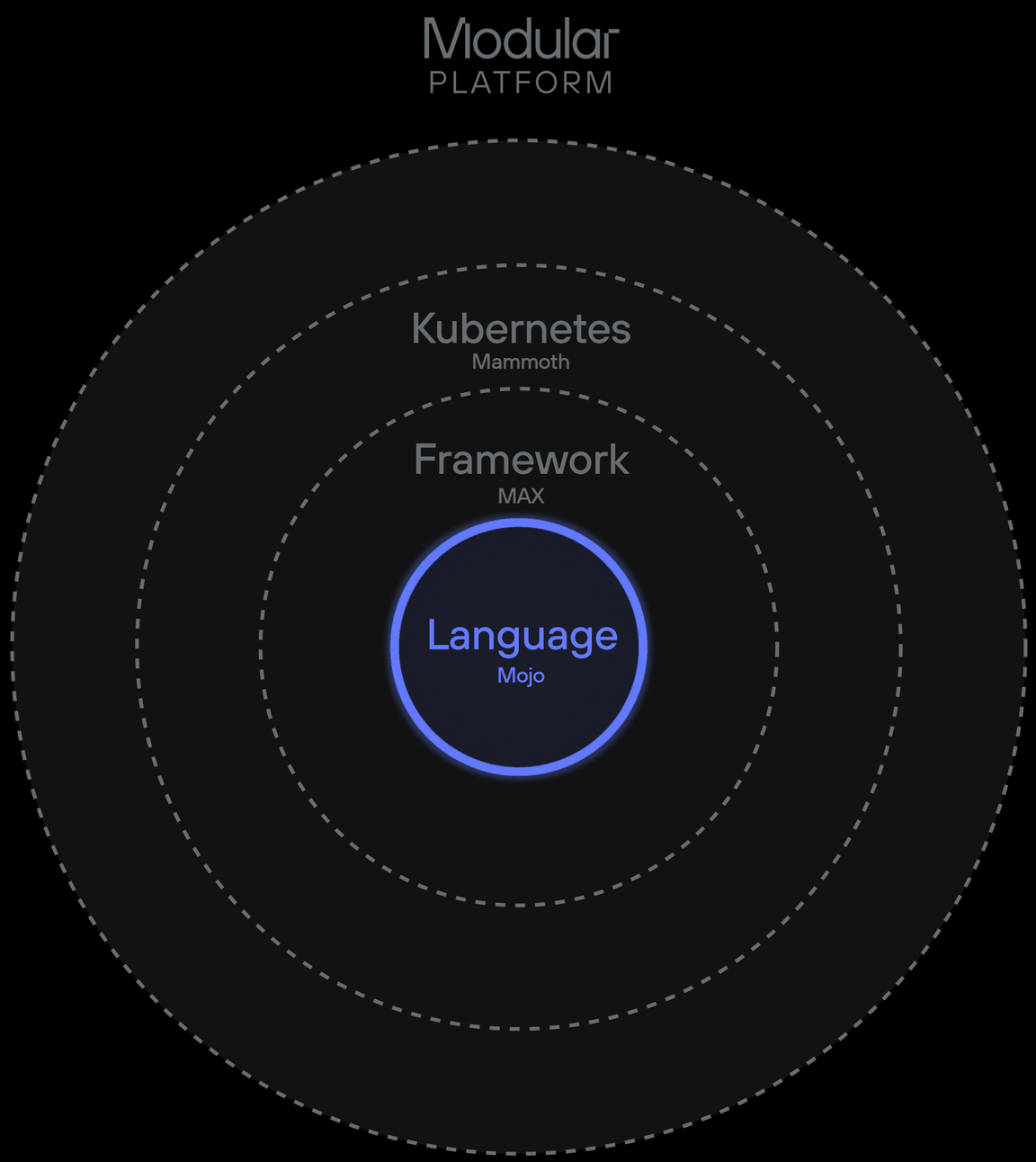
Unlocking performance for heterogeneous compute

- Blazing fast
- Unified programming CPU + GPU
- Scalable GPU kernels for Nvidia, AMD, Apple ...

Best way to extend Python to CPUs and GPUs
Backbone for MAX 🧑‍🚀

```
def mandelbrot_kernel[
    width: Int  # SIMD Width
](c: ComplexSIMD[float, width]) ->
   SIMD[int, width]:

   """A vectorized implementation of
   the inner mandelbrot computation."""
   z = ComplexSIMD[float, width](0, 0)

   iters = SIMD[DType.index, width](0)
   mask = SIMD[DType.bool, width](True)

   for i in range(MAX_ITERS):
    if not any(mask):
       break

    mask = z.squared_norm() <= 4
    iters = mask.select(iters + 1, iters)
    z = z.squared_add(c)
   return iters
```


Mojo 🔥: A system programming language for heterogenous computing
LLVM Dev Meeting 2023
Mojo : Programming language for all of AI
Mojo: A novel programming language for AI ACAT 202



Mammoth Cluster

MAX Framework

Mojo🔥 Language

Code for a single device
Get the full power of the hardware
Easy to learn and scale

Diving into the Technology

01

Foundations through an example

Demo of Vector Addition in Mojo

02

Mojo Libraries

Overview, common patterns, and code examples

03

Kernels

Matmul on GPU and measuring performance

Structured Kernels

Mojo 🔥 needs ...

... what Python 🐍 has

Powerful metaprogramming:

- Decorators
- Metaclasses
- Reflection

But ... Runtime based is slow - it will never run on the accelerator!

💡 Let's do it at compile time! 💡

Hardware Portability

Same language / no emulation

- The warp size / simd width is a property of the target hardware

Everything is JIT compiled

- Fast compile time, publish device agnostic packages

```
fn add_one[target: String](
  out: OutputTensor,
  input: InputTensor[type=out.type, rank=out.rank],
  ctx: DeviceContextPtr) raises:

  fn add_one_impl[width: Int](idx: IndexList[x.rank])
                -> SIMD[x.type, width]:
    return input.load[width](idx) + 1

  foreach[add_one_impl, target=target](out, ctx)
```

We don't believe in "magic"

- Mojo gives you the tools to build portable abstractions
- It doesn't pretend different hardware is the same

GPU Programming in Mojo

GPU kernel in Mojo

Library imports

constants

GPU function

GPU tensor representation

GPU global offset

```mojo
from math import ceildiv
from sys import has_accelerator
from gpu import global_idx
from gpu.host import DeviceContext
from layout import Layout, LayoutTensor

alias float_dtype = DType.float32
alias VECTOR_WIDTH = 10
alias BLOCK_SIZE = 5
alias layout = Layout.row_major(VECTOR_WIDTH)

fn vector_addition(
    lhs_tensor: LayoutTensor[float_dtype, layout, MutableAnyOrigin],
    rhs_tensor: LayoutTensor[float_dtype, layout, MutableAnyOrigin],
    out_tensor: LayoutTensor[float_dtype, layout, MutableAnyOrigin],
    size: Int,
):
    """The calculation to perform across the vector on the GPU."""
    var global_tid = global_idx.x
    if global_tid < UInt(size):
        out_tensor[global_tid] = lhs_tensor[global_tid] + rhs_tensor[global_tid]
```

CPU host code in Mojo

static assertion

GPU DeviceContext

GPU device buffers

GPU tensors

compile and launch GPU kernel

device buffer to host

```mojo
def main():
    constrained[has_accelerator(), "This example requires a supported GPU"]()

    # Get context for the attached GPU
    var ctx = DeviceContext()

    # Allocate data on the GPU address space
    var lhs_buffer = ctx.enqueue_create_buffer[float_dtype](VECTOR_WIDTH)
    var rhs_buffer = ctx.enqueue_create_buffer[float_dtype](VECTOR_WIDTH)
    var out_buffer = ctx.enqueue_create_buffer[float_dtype](VECTOR_WIDTH)

    # Fill in values across the entire width
    _ = lhs_buffer.enqueue_fill(1.25)
    _ = rhs_buffer.enqueue_fill(2.5)

    # Wrap the device buffers in tensors
    var lhs_tensor = LayoutTensor[float_dtype, layout](lhs_buffer)
    var rhs_tensor = LayoutTensor[float_dtype, layout](rhs_buffer)
    var out_tensor = LayoutTensor[float_dtype, layout](out_buffer)

    # Calculate the number of blocks needed to cover the vector
    var grid_dim = ceildiv(VECTOR_WIDTH, BLOCK_SIZE)

    # Launch the vector_addition function as a GPU kernel
    ctx.enqueue_function_checked[vector_addition, vector_addition](
        lhs_tensor,
        rhs_tensor,
        out_tensor,
        VECTOR_WIDTH,
        grid_dim=grid_dim,
        block_dim=BLOCK_SIZE,
    )

    # Map to host so that values can be printed from the CPU
    with out_buffer.map_to_host() as host_buffer:
        var host_tensor = LayoutTensor[float_dtype, layout](host_buffer)
        print("Resulting vector:", host_tensor)
```

https://github.com/modular/modular/blob/main/examples/mojo/gpu-functions/vector_addition.mojo

GPU Programming in Mojo

- Unified programming for CPU + GPU
- The full power of standard CUDA/ROCm
  - Threads, warps, sync primitives
  - All the WMMA instructions
- "Without CUDA"
  - All GPU kernels written in Mojo.
  - Generate PTX directly without using CUDA toolkits or libraries
  - GPU kernels compiled on demand with small release container size
- GPU kernels for Nvidia, AMD, Apple in one language
- Library driven with "simple" compiler support

```
# ===-----------------------------------------------------------------------===#
# thread_idx
# ===-----------------------------------------------------------------------===#


@register_passable("trivial")
struct _ThreadIdx(Defaultable):
    """ThreadIdx provides static methods for getting the x/y/z coordinates of
    a thread within a block."""

    @always_inline("nodebug")
    fn __init__(out self):
        return

    @always_inline("nodebug")
    @staticmethod
    fn _get_intrinsic_name[dim: StringLiteral]() -> StaticString:
        @parameter
        if is_nvidia_gpu():
            return "llvm.nvvm.read.ptx.sreg.tid." + dim
        elif is_amd_gpu():
            return "llvm.amdgcn.workitem.id." + dim
        elif is_apple_gpu():
            return "llvm.air.thread_position_in_threadgroup." + dim
        else:
            return CompilationTarget.unsupported_target_error[
                StaticString,
                operation="thread_idx field access",
            ]()

    @always_inline("nodebug")
    fn __getattr__[dim: StringLiteral](self) -> UInt:
        """Gets the `x`, `y`, or `z` coordinates of a thread within a block.

        Returns:
            The `x`, `y`, or `z` coordinates of a thread within a block.
        """
        _verify_xyz[dim]()
        alias intrinsic_name = Self._get_intrinsic_name[dim]()
        return UInt(
            llvm_intrinsic[intrinsic_name, UInt32, has_side_effect=False]()
        )


alias thread_idx = _ThreadIdx()
```

compile-time condition for different HW

llvm intrinsics

https://github.com/modular/modular/blob/main/mojo/stdlib/stdlib/gpu/intrinsics.mojo

Mojo Parameter Syntax

```
# Struct with parameters
struct SIMD[dtype: DType, width: Int]:
    ...


# "alias" declaration -> parameter
alias Float32 = SIMD[DType.f32, 1]
```

```
# Bind function parameters to type
fn first_class_simd[width: Int](
    x: SIMD[DType.float32, width]):
  pass
```

~= C++ templates

Simple meta programing example

Strings, lists, trees all "just work" at compile time

Function that builds a list with a "for" loop

```
fn fill(lb: Int, ub: Int) -> List[Int]:
    var values = List[Int]()
    for i in range(lb, ub):
        values.append(i)
    return values

fn comptime_vector():
    alias vec = fill(15, 20)
    for e in vec:
        print(e)
```

Vector with heap allocation

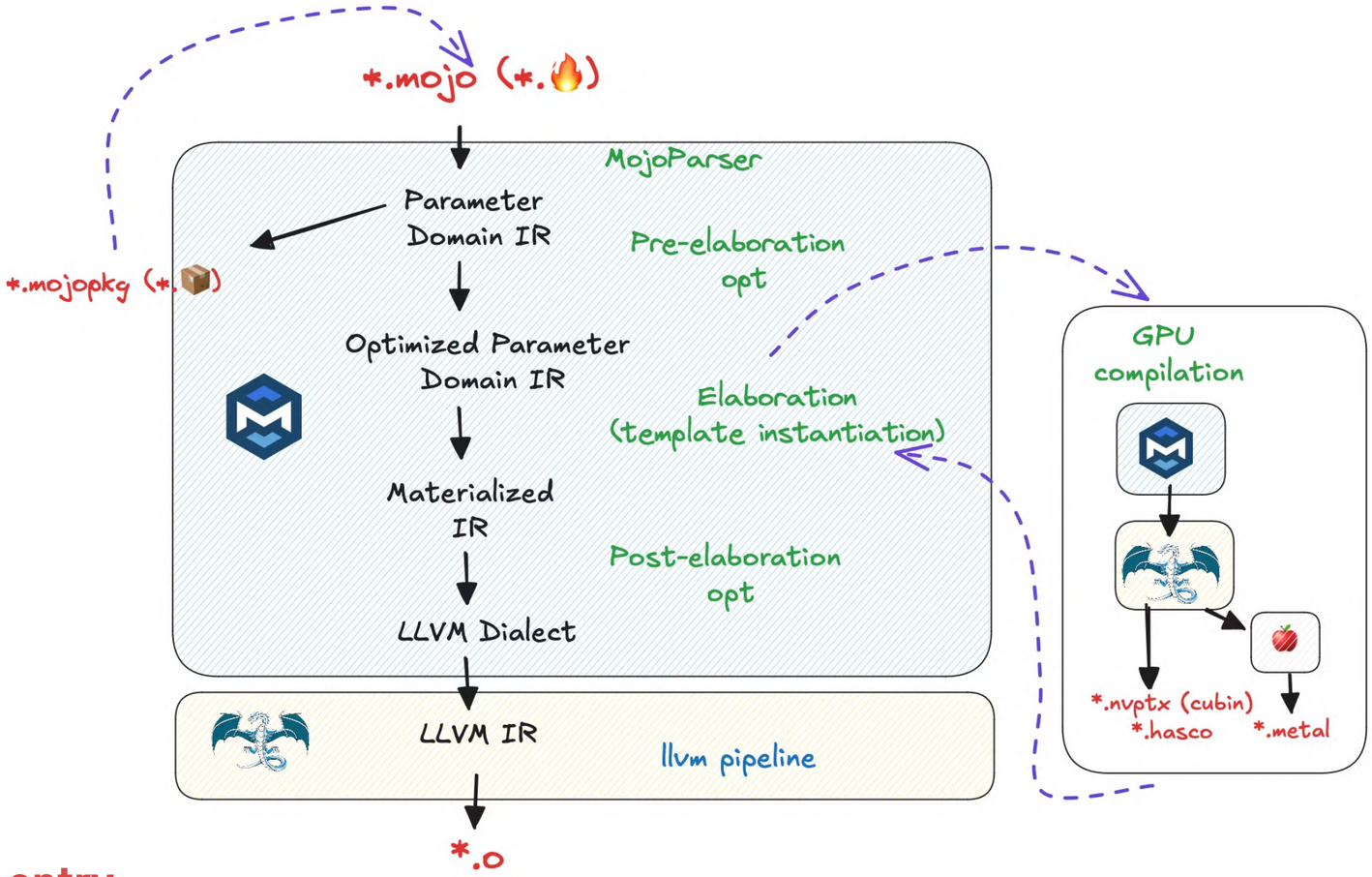
Vector computed at compile-time... ***comptime malloc***.

Result materialized into a runtime value!

Mojo GPU Compilation Flow

```mojo
def main():
    constrained[has_accelerator(), "This example requires a supported GPU"]()

    # Get context for the attached GPU
    var ctx = DeviceContext()

    # Allocate data on the GPU address space
    var lhs_buffer = ctx.enqueue_create_buffer[float_dtype](VECTOR_WIDTH)
    var rhs_buffer = ctx.enqueue_create_buffer[float_dtype](VECTOR_WIDTH)
    var out_buffer = ctx.enqueue_create_buffer[float_dtype](VECTOR_WIDTH)

    # Fill in values across the entire width
    _ = lhs_buffer.enqueue_fill(1.25)
    _ = rhs_buffer.enqueue_fill(2.5)

    # Wrap the device buffers in tensors
    var lhs_tensor = LayoutTensor[float_dtype, layout](lhs_buffer)
    var rhs_tensor = LayoutTensor[float_dtype, layout](rhs_buffer)
    var out_tensor = LayoutTensor[float_dtype, layout](out_buffer)

    # Calculate the number of blocks needed to cover the vector
    var grid_dim = ceildiv(VECTOR_WIDTH, BLOCK_SIZE)

    # Launch the vector_addition function as a GPU kernel
    ctx.enqueue_function_checked[vector_addition, vector_addition](
        lhs_tensor,
        rhs_tensor,
        out_tensor,
        VECTOR_WIDTH,
        grid_dim=grid_dim,
        block_dim=BLOCK_SIZE,
    )

    # Map to host so that values can be printed from the CPU
    with out_buffer.map_to_host() as host_buffer:
        var host_tensor = LayoutTensor[float_dtype, layout](host_buffer)
        print("Resulting vector:", host_tensor)
```

GPU entry function as a parameter



Resources

1. Source code: https://github.com/modular/modular
   - `examples/mojo`

2. Documentation:
   https://docs.modular.com/mojo/manual/

3. GPU puzzles:
   https://puzzles.modular.com/introduction.html

4. Forum: https://forum.modular.com
   Discord: https://discord.gg/modular

5. Modular presentations and tech talks:
   https://www.youtube.com/@modularinc

Diving into the Technology

01

Foundations through an example

Walk through vector addition kernel

02

Mojo Libraries

Overview, common patterns, and code examples

03

Kernels

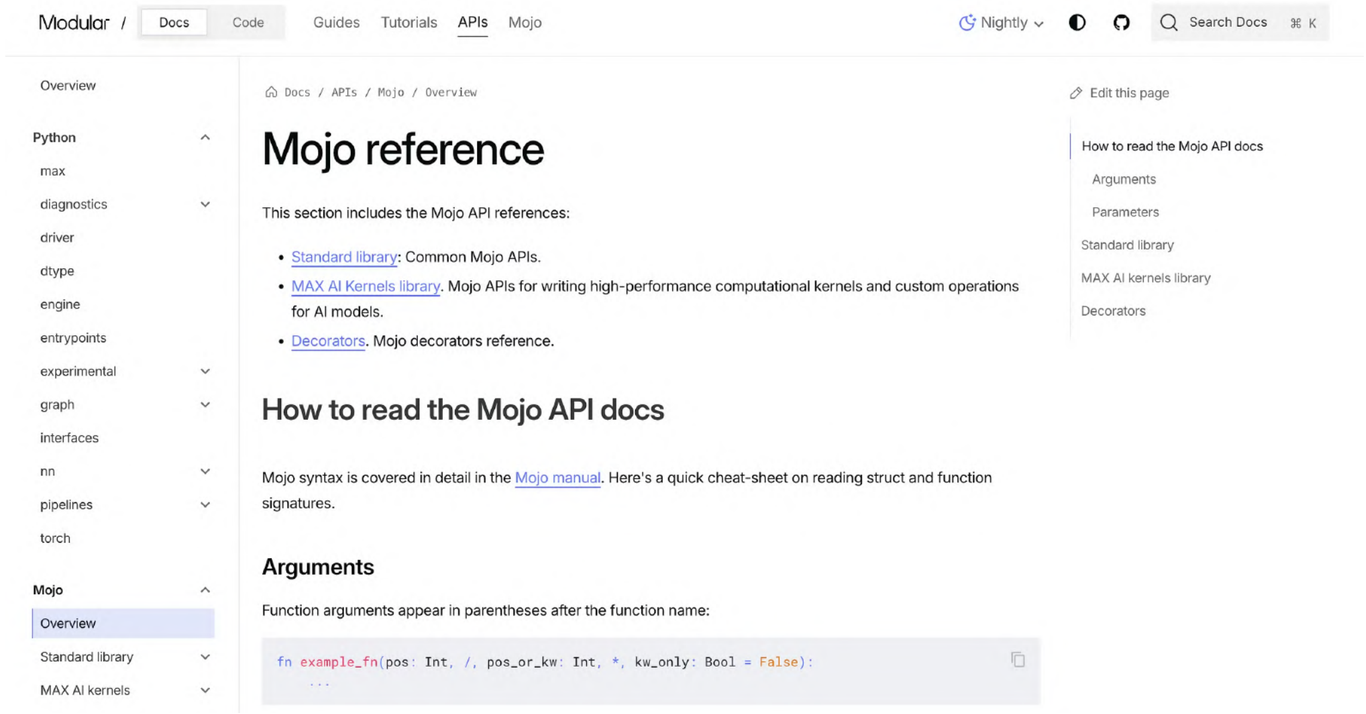
Matmul on GPU and measuring performance

Structured Kernels

Qualcomm-specific kernels

Documentation

- Stdlib and other APIs have auto-generated docs
- All of the stdlib + kernels code is open source!
- Good resource for browsing the modules and getting a feel for things
- There's a dedicated Mojo manual as well - check it out!



Data Type System

- `DType`'s purpose: data type definitions and conversions
- Supported types:
  - Integers: int8 → int256
  - Unsigned: uint8 → uint256
  - Floating point: float16, float32, float64, various fp8 types
- **DType is an enum**, so you can't write: `var my_int: DType.uint8 = 1`
  - `DType.uint8` is **not** a type: it's a **value**

```
@register_passable("trivial")
struct DType(…):
    alias _mlir_type = __mlir_type.`!kgen.dtype`
    var _mlir_value: Self._mlir_type

    alias uint8 = DType( mlir_value=__mlir_attr.`#kgen.dtype.constant<ui8> : !kgen.dtype`)
    alias int8 = DType( mlir_value=__mlir_attr.`#kgen.dtype.constant<si8> : !kgen.dtype`)
    # ... other bitwidths too
    alias float8_e3m4 = DType(
        mlir_value=__mlir_attr.`#kgen.dtype.constant<f8e3m4> : !kgen.dtype`
    )
    """Represents an 8-bit `e3m4` floating point format, encoded as
    `s.eee.mmmm`:

    - (s)ign: 1 bit
    - (e)xponent: 3 bits
    - (m)antissa: 4 bits
    - exponent bias: 3
    - nan: {0,1}.111.1111
    - fn: finite (no inf or -inf encodings)
    - -0: 1.000.0000
    """
    alias float8_e4m3fn = DType(
        mlir_value=__mlir_attr.`#kgen.dtype.constant<f8e4m3fn> : !kgen.dtype`
    )
    alias float8_e4m3fnuz = DType(
        mlir_value=__mlir_attr.`#kgen.dtype.constant<f8e4m3fnuz> : !kgen.dtype`
    )
    alias float8_e5m2 = DType(
        mlir_value=__mlir_attr.`#kgen.dtype.constant<f8e5m2> : !kgen.dtype`
    )
    alias float8_e5m2fnuz = DType(
        mlir_value=__mlir_attr.`#kgen.dtype.constant<f8e5m2fnuz> : !kgen.dtype`
    )
    alias bfloat16 = DType(
        mlir_value=__mlir_attr.`#kgen.dtype.constant<bf16> : !kgen.dtype`
    )
```

SIMD

- Key Features:
  - Explicit control over alignment and load/store patterns
  - Direct access to SIMD intrinsics
  - Configurable SIMD widths
- The foundational numeric type in Mojo. Important part is `Scalar` is in terms of `SIMD`.
- Every numeric primitive is just an alias to SIMD

```
alias Scalar = SIMD[_, size=1]
"""Represents a scalar dtype."""

alias Int8 = Scalar[DType.int8]
"""Represents an 8-bit signed scalar integer."""
alias UInt8 = Scalar[DType.uint8]
"""Represents an 8-bit unsigned scalar integer."""
alias Int16 = Scalar[DType.int16]
"""Represents a 16-bit signed scalar integer."""
alias UInt16 = Scalar[DType.uint16]
"""Represents a 16-bit unsigned scalar integer."""
alias Int32 = Scalar[DType.int32]
"""Represents a 32-bit signed scalar integer."""
alias UInt32 = Scalar[DType.uint32]
"""Represents a 32-bit unsigned scalar integer."""
alias Int64 = Scalar[DType.int64]
"""Represents a 64-bit signed scalar integer."""
alias UInt64 = Scalar[DType.uint64]
"""Represents a 64-bit unsigned scalar integer."""
alias Int128 = Scalar[DType.int128]
"""Represents a 128-bit signed scalar integer."""
alias UInt128 = Scalar[DType.uint128]
"""Represents a 128-bit unsigned scalar integer."""
alias Int256 = Scalar[DType.int256]
"""Represents a 256-bit signed scalar integer."""
alias UInt256 = Scalar[DType.uint256]
"""Represents a 256-bit unsigned scalar integer."""

@register_passable("trivial")
struct SIMD[dtype: DType, size: Int](
    # ...
):
    ...
```

Memory Management and Pointers

- One of several pointer types available to indirectly reference locations in memory.
- Note the address space and mutability embedded in the type. We'll also talk about origins in a bit.
- Dedicated section in the Mojo manual if you'd like to read more

```mojo
@register_passable("trivial")
struct UnsafePointer[
    type: AnyType,
    *,
    address_space: AddressSpace = AddressSpace.GENERIC,
    mut: Bool = True,
    origin: Origin[mut] = Origin[mut].cast_from[MutableAnyOrigin],
]
    # ===-------------------------------------------------------------------===#
    # Aliases
    # ===-------------------------------------------------------------------===#

    # Fields
    alias _mlir_type = __mlir_type[
        `!kgen.pointer<`,
        type,
        `, `,
        address_space._value._mlir_value,
        `>`,
    ]
    var address: Self._mlir_type

    @always_inline("builtin")
    fn __init__(out self):
        """Create a null pointer."""
        self.address = __mlir_attr[`#interp.pointer<0> : `, Self._mlir_type]

    @doc_private
    @always_inline("builtin")
    @implicit
    fn __init__(out self, value: Self._mlir_type):
        """Create a pointer from a low-level pointer primitive.

        Args:
            value: The MLIR value of the pointer to construct with.
        """
        self.address = value

    @always_inline("nodebug")
    fn __init__(
        out self, *, ref [origin, address_space._value._mlir_value]to: type
    ):
        """Constructs a Pointer from a reference to a value.

        Args:
            to: The value to construct a pointer to.
        """
        self = Self(__mlir_op.`lit.ref.to_pointer`(__get_mvalue_as_litref(to)))
```

https://github.com/modular/modular/blob/main/mojo/stdlib/stdlib/memory/unsafe_pointer.mojo

Kernels & Stdlib are OSS

GPU kernels

modul.ar/kernels

```
[modular/max/nn] ls *.mojo | pr -3 -t
__init__.mojo                gather_scatter.mojo          pad.mojo
_amd_flash_attention_gp      image.mojo                   pool.mojo
_ragged_utils.mojo           index_tensor.mojo            rand_uniform.mojo
activations.mojo             irfft.mojo                   randn.mojo
arange.mojo                  kv_cache_ragged.mojo         repeat_interleave.mojo
arg_nonzero.mojo             kv_cache.mojo                reshape.mojo
argmaxmin_gpu.mojo           mha_cross.mojo               resize.mojo
argmaxmin.mojo               mha_mask.mojo                roi_align.mojo
argsort.mojo                 mha_operand.mojo             sampling.mojo
bicubic.mojo                 mha_score_mod.mojo           shapes.mojo
broadcast.mojo               mha_sm90.mojo                slice.mojo
concat.mojo                  mha_tile_scheduler.mojo      softmax.mojo
conv_transpose.mojo          mha_utils.mojo               split.mojo
conv_utils.mojo              mha.mojo                     tile.mojo
conv.mojo                    mla.mojo                     topk.mojo
cumsum.mojo                  moe.mojo                     toppminp_gpu.mojo
flash_attention.mojo         nms.mojo                     toppminp.mojo
fold.mojo                    normalization.mojo
fused_qk_rope.mojo           pad_gpu.mojo
```

About 1 million lines of OSS code

LayoutTensor

LayoutTensor

A portable advancement for CUTLASS / CUTE

Defines a shared memory & register tiled Matmul operation in a succinct manner

Works on multiple GPUs + CPUs as well

modul.ar/mat-mul-recipe
modul.ar/layouts-intro

```
var col = thread_idx.x % UInt(BN)
var row = thread_idx.x // UInt(BN)

var dst = c.tile[BM, BN](block_idx.y, block_idx.x).tile[TM, 1](row, col)

alias SharedTensor = LayoutTensor[
    _, _, MutableAnyOrigin, address_space = AddressSpace.SHARED
]
var a_smem = SharedTensor[dtype, Layout.row_major(BM, BK)].stack_allocation()
var b_smem = SharedTensor[dtype, Layout.row_major(BK, BN)].stack_allocation()

var dst_reg = LayoutTensor[
    dtype,
    Layout(TM),
    MutableAnyOrigin,
    address_space = AddressSpace.LOCAL,
].stack_allocation()
dst_reg.copy_from(dst)

for block in range(b.dim[0]() // BK):
    alias load_a_layout = Layout.row_major(NUM_THREADS // BK, BK)
    alias load_b_layout = Layout.row_major(BK, NUM_THREADS // BK)

    var a_tile = a.tile[BM, BK](block_idx.y, block)
    var b_tile = b.tile[BK, BN](block, block_idx.x)

    copy_dram_to_sram_async[thread_layout=load_a_layout](a_smem, a_tile)
    copy_dram_to_sram_async[thread_layout=load_b_layout](b_smem, b_tile)

    async_copy_wait_all()
    barrier()

    @parameter
    for k in range(BK):
        var a_tile = a_smem.tile[TM, 1](row, k)
        var b_tile = b_smem.tile[1, BN](k, 0)
        var b_val = b_tile[0, col]

        @parameter
        for t in range(TM):
            dst_reg[t] += a_tile[t, 0] * b_val

    barrier()

dst.copy_from(dst_reg)
```

Inside Layout

- A function logical (n-d) coords → linear coords / Integer
- Defined by {IntTuple(shape), IntTuple(stride)}
- Layout(coords) → dot(coords, stride)
- Only used at compile-time

```
struct Layout:
    var shape: IntTuple
    var stride: IntTuple

    fn __init__(out self, shape: IntTuple, stride: IntTuple):
        self.shape = shape.owned_copy()
        if len(stride) == 0:
            self.stride = prefix_product(self.shape)
        else:
            self.stride = stride.owned_copy()

    @always_inline("nodebug")
    fn idx2crd(self, idx: IntTuple) -> IntTuple:
        return idx2crd(idx, self.shape, self.stride)


    @always_inline("nodebug")
    fn __call__(self, idx: IntTuple) -> Int:
        return crd2idx(idx, self.shape, self.stride)

    @staticmethod
    fn col_major(*dims: Int) -> Layout:
        var shape = IntTuple(dims)
        return Self.col_major(shape)
```

Diving into the Technology

01

Foundations through an example

Walk through vector addition kernel

02

Mojo Libraries

Overview, common patterns, and code examples

03

Kernels

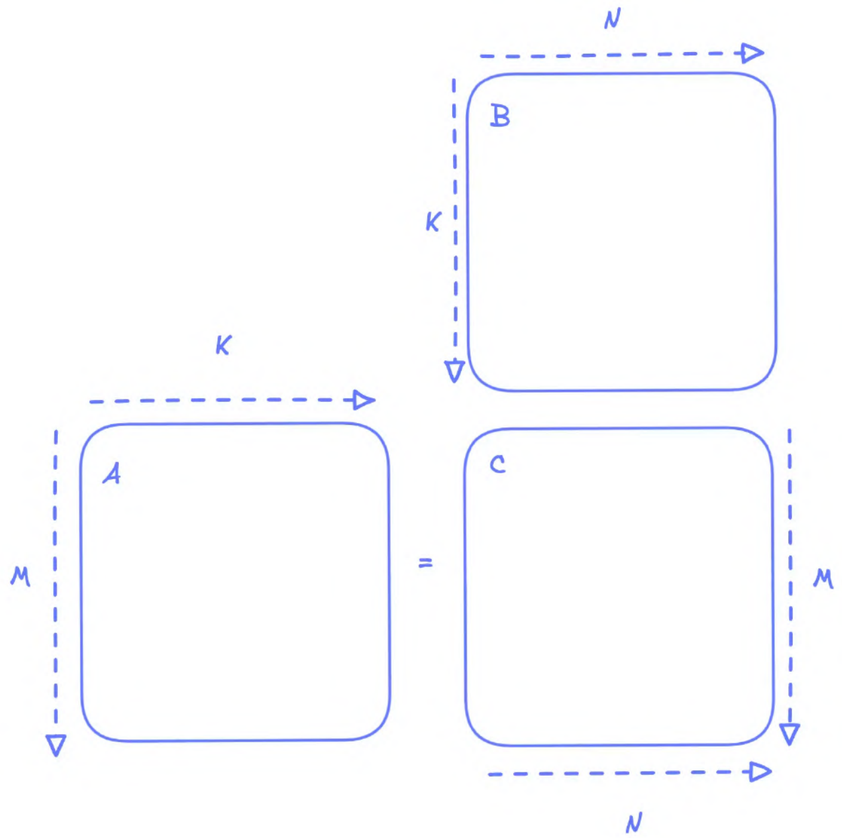
Matmul on GPU

Structured Kernels

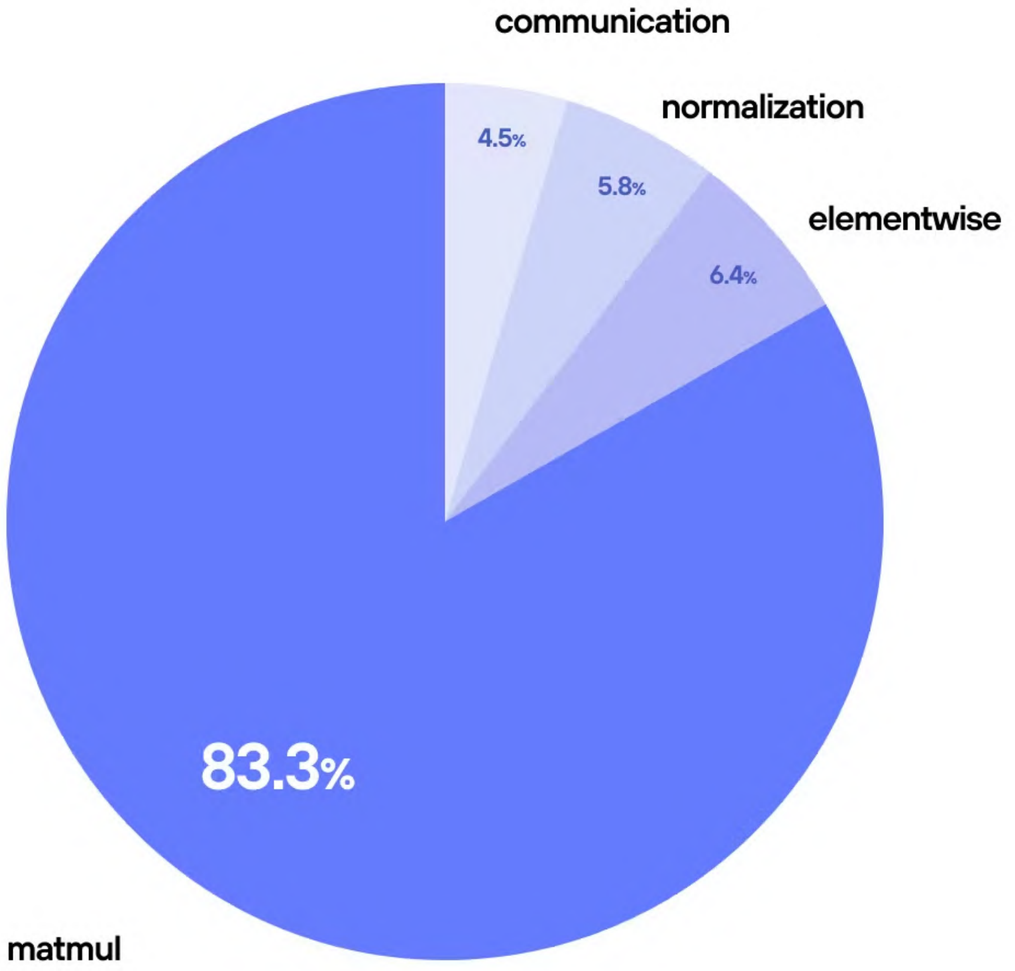
Matrix Multiplication
Structured kernels

What's Matmul?

```
for row in range(M):
  for col in range(N):
    C[row][col] = 0
    for inner in range(K):
      C[row][col] += A[row][inner]*B[inner][col]
```



Why do we care about Matmul?



matmul makes up more than 80% of Llama 8B execution

Classical Optimizations for Matmuls

Tiling

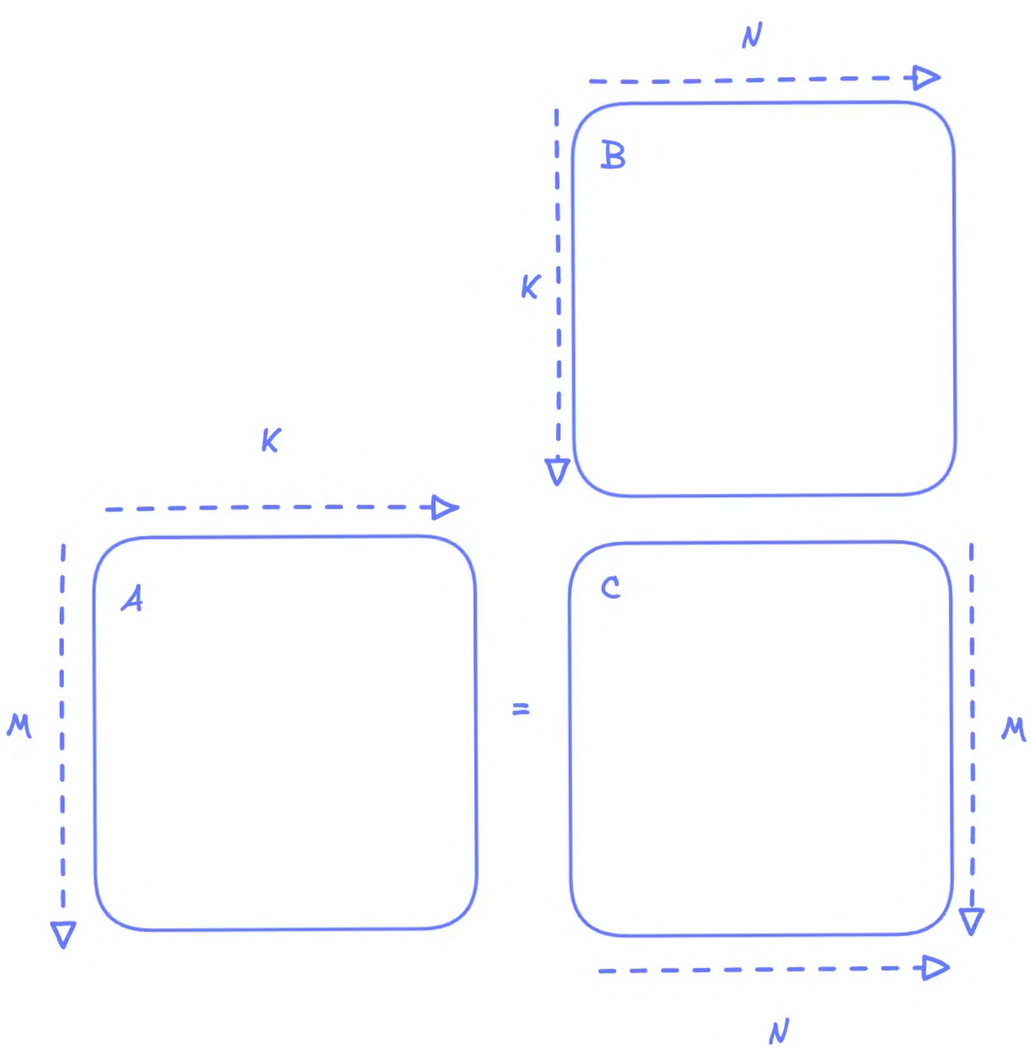
- Partition data to increase reuse

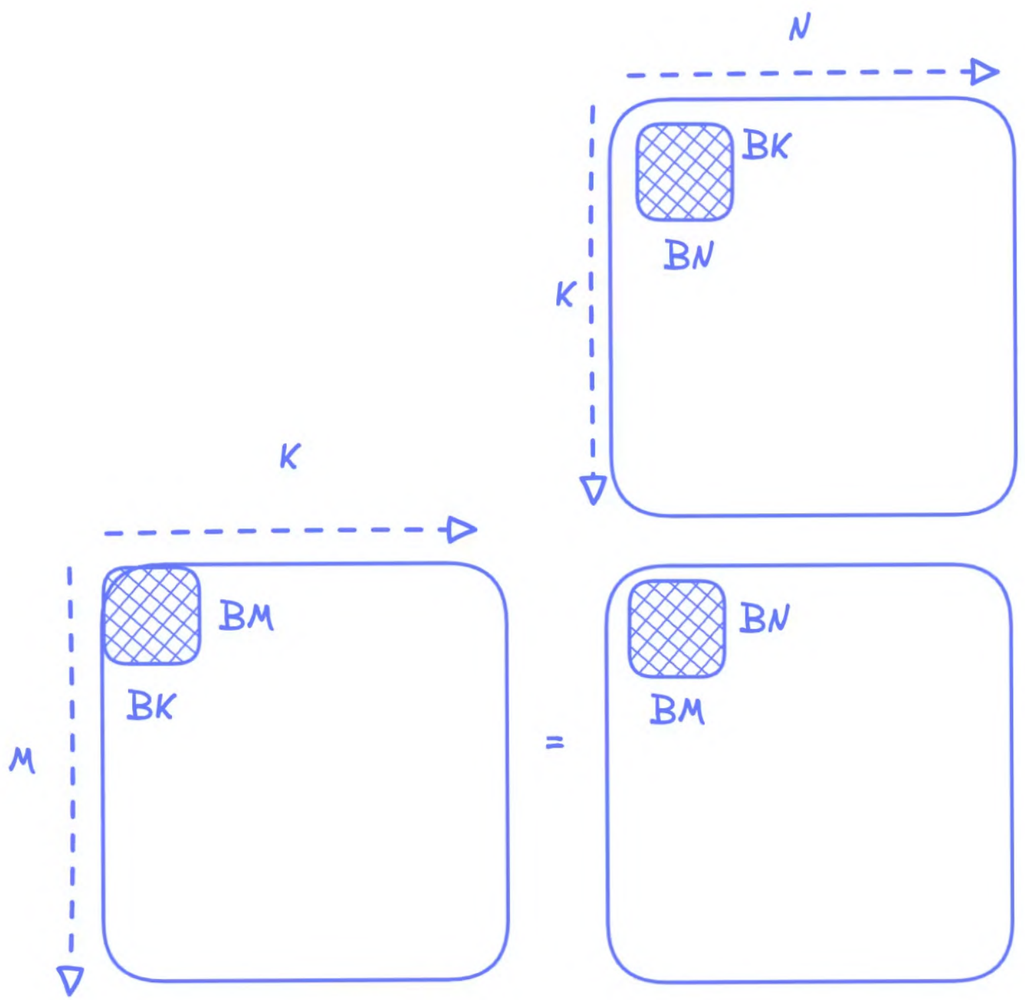
Pipelining

- Pingpong buffering
- More deep pipelining

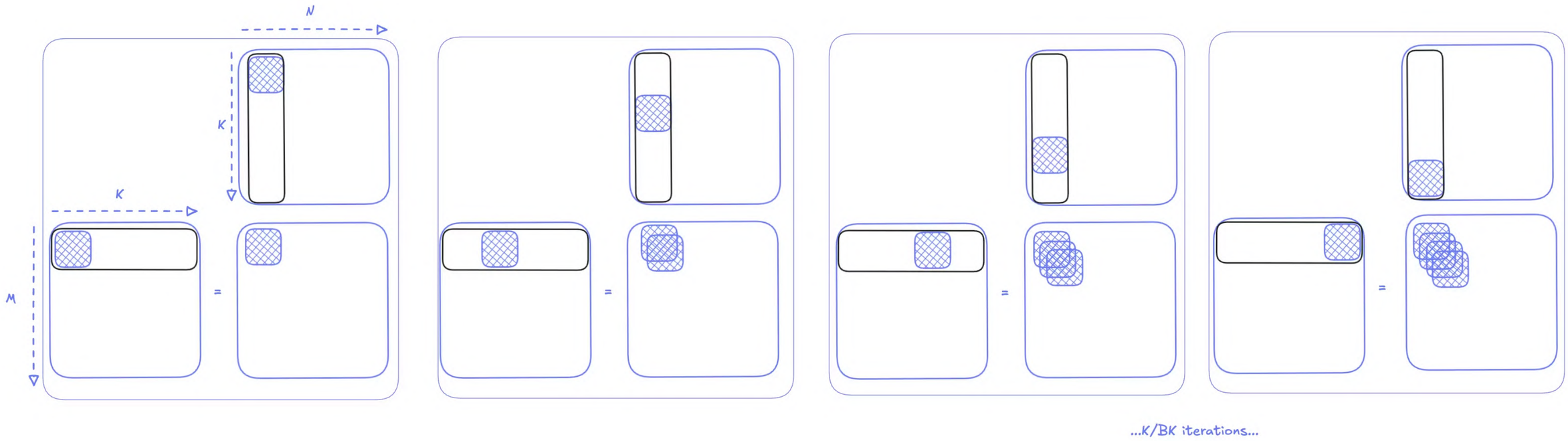
... and use the TensorCore

Matmul Tiling

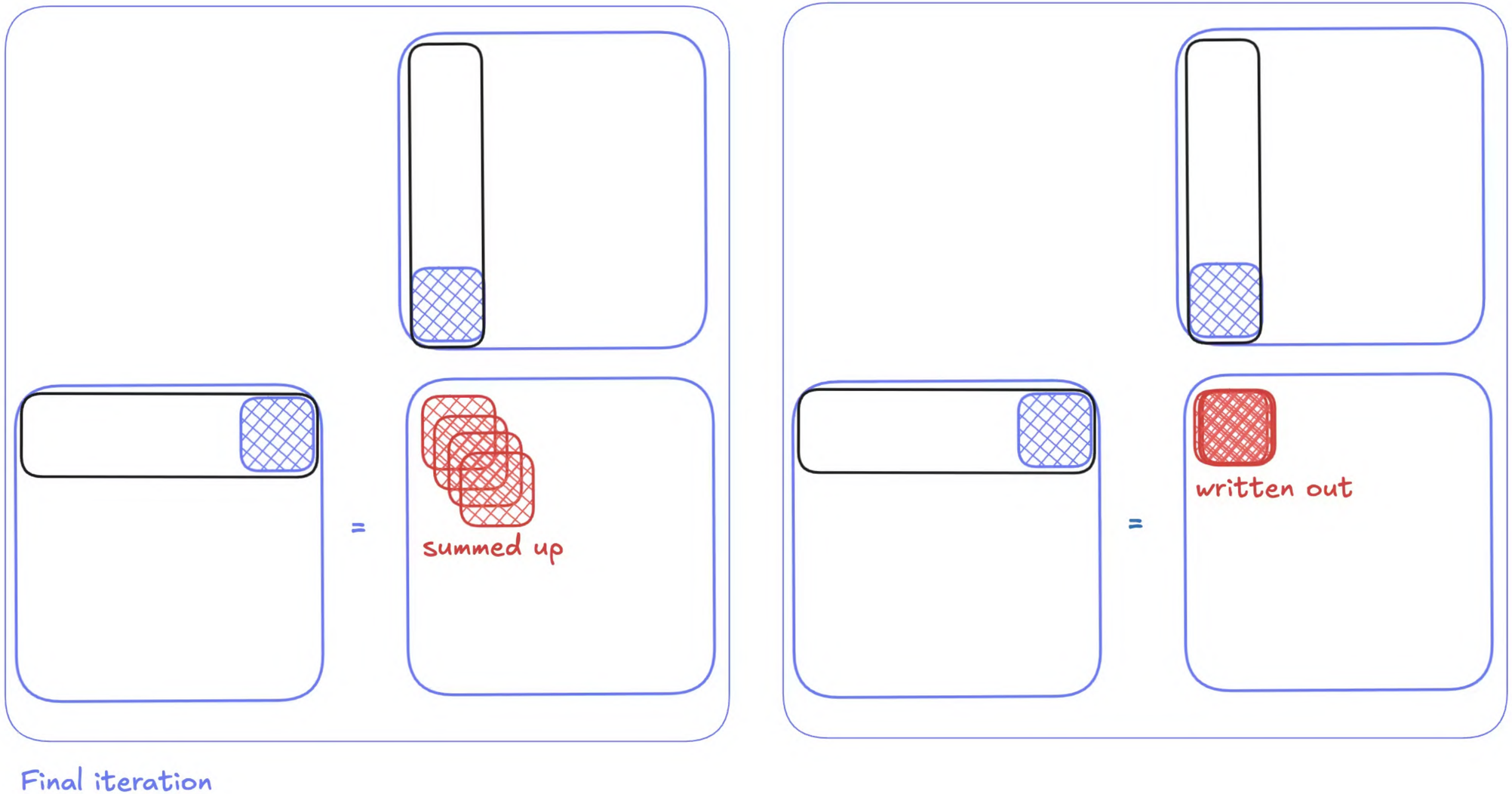




Matmul Tiling



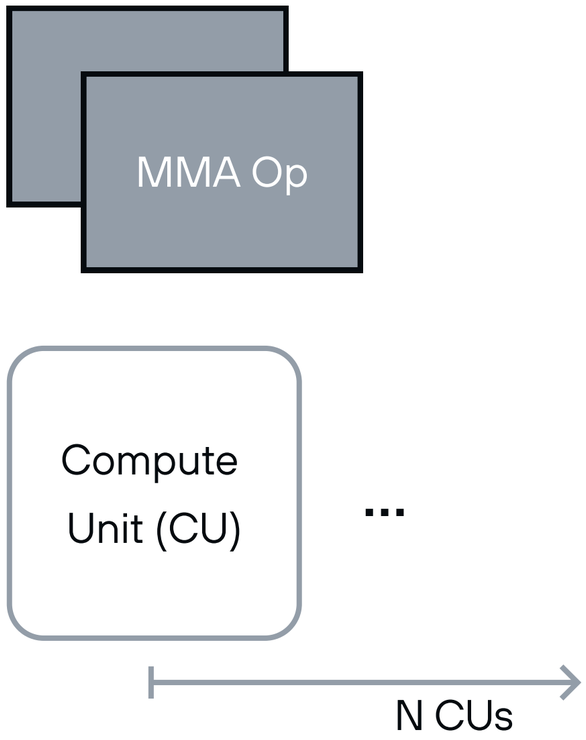
Matmul Tiling



MatMul pipelining depends on GPU

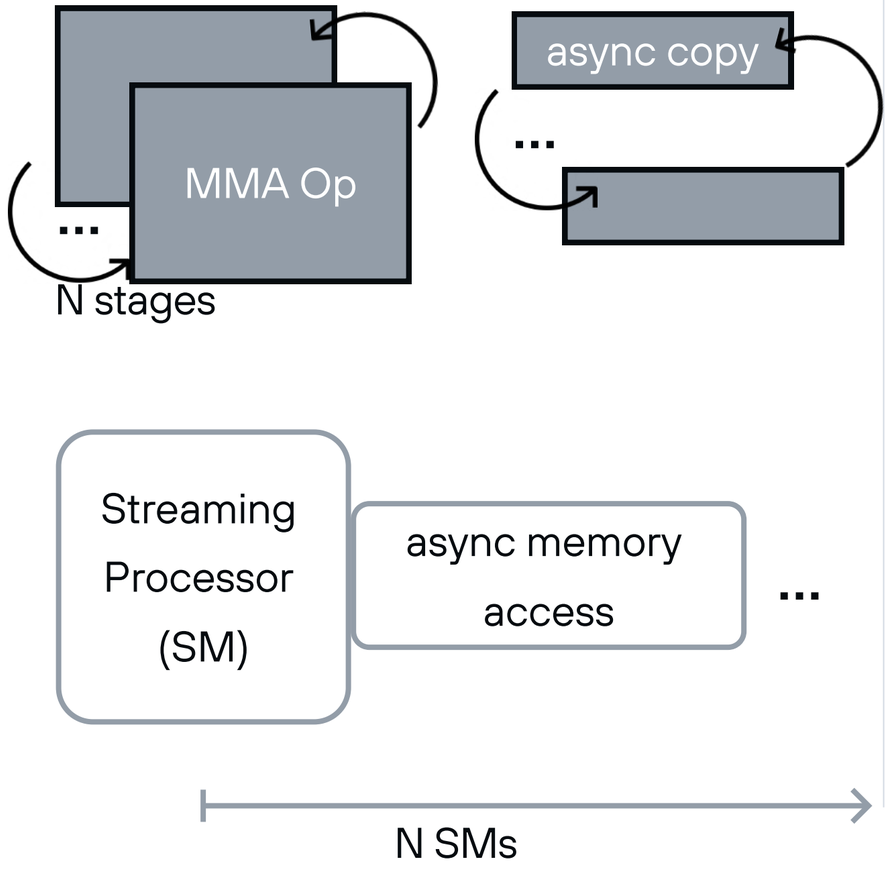
AMD MI355

Double buffer loop



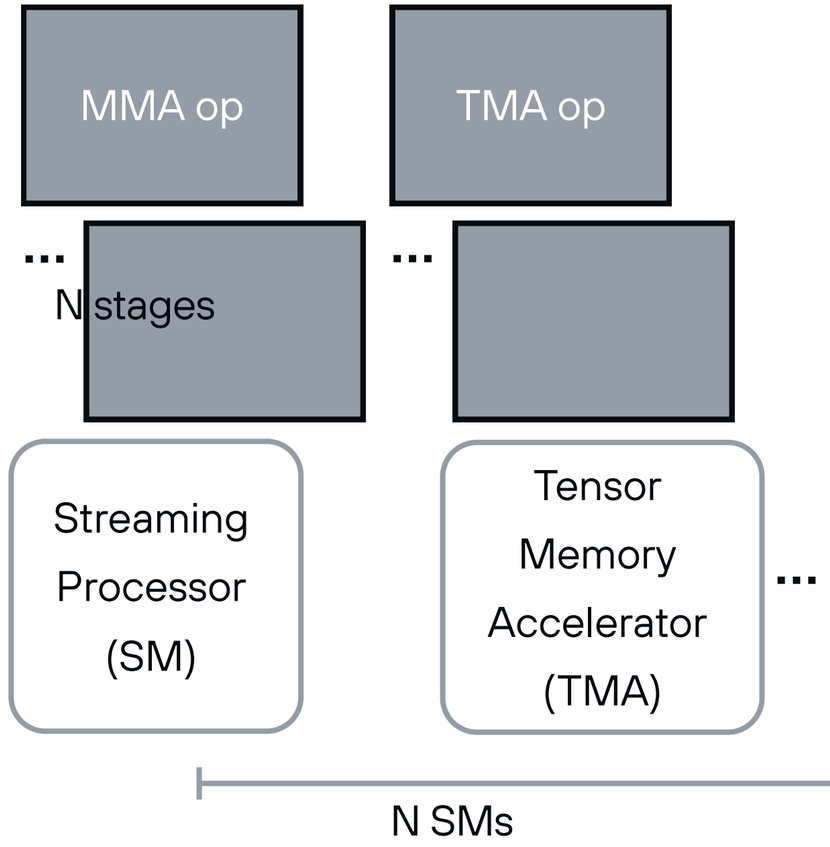
NVIDIA A100

N stages & async copy via circular buffer

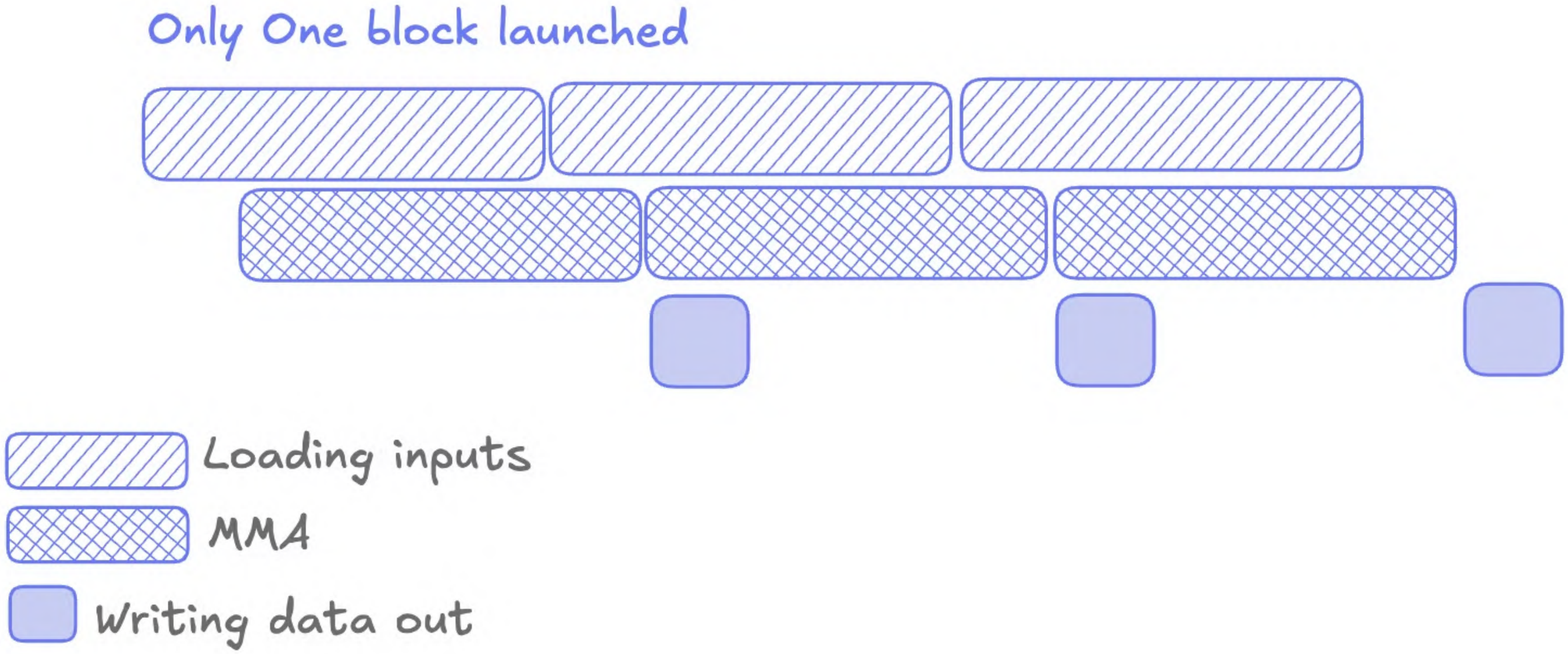


NVIDIA H100 / B200

Producer/Consumer & warp-specialized



Blackwell Pipelining



Blackwell Matmuls

Leverage the hardware

- Use Grid Clusters to perform 2SM optimization
- Maximize overlap between compute and communication

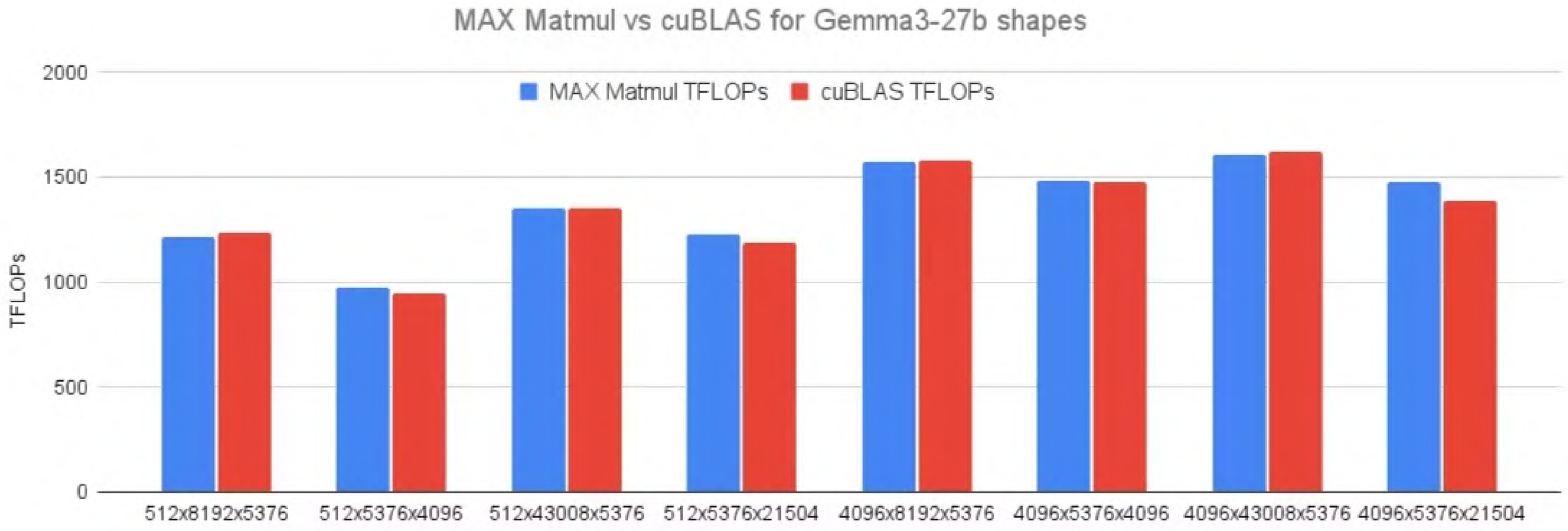
Persistent

- Launch as many blocks as there are SMs
- Perform own scheduling to process next tile

Use other hardware features such as TMA, TMEM, etc.

https://www.modular.com/matrix-multiplication-on-blackwell

Upto 6% faster than cuBLAS



https://www.modular.com/matrix-multiplication-on-blackwell

Small M

The M dimension

- Corresponds to the batch size

What is small?

- For our definition < 128

What is the issue?

- The GPU was not fully utilized

Following the blog post you'd only get 80% of the perf of cuBLAS on small M

Full detailed will be explained in a Part 5 of the blog post

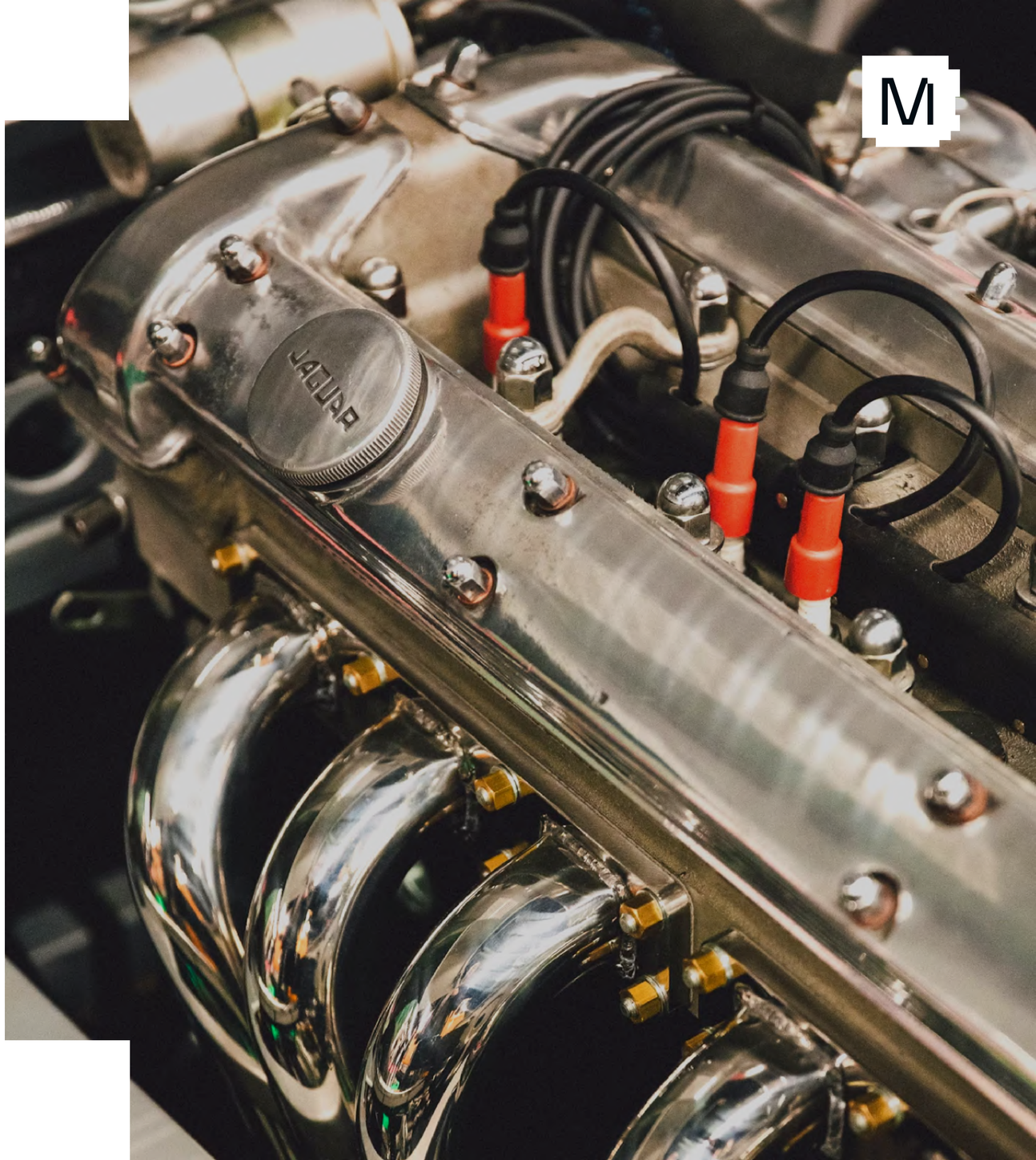
Poor GPU Utilization

Tensor Core (MMA) shapes

- `mma_m x mma_n x 16`
- `mma_m` is 64, 128 per SM
- `mma_n` is range(8, 257, 8) or range(16, 257, 16)

A real world case

- We have MxNxK = 16x14336x4096 in llama3's work load and 148 SMs on B200
- Use 64x128x16 mma ⇒ 112 tasks, waste 75% of mma throughput and shared memory usage



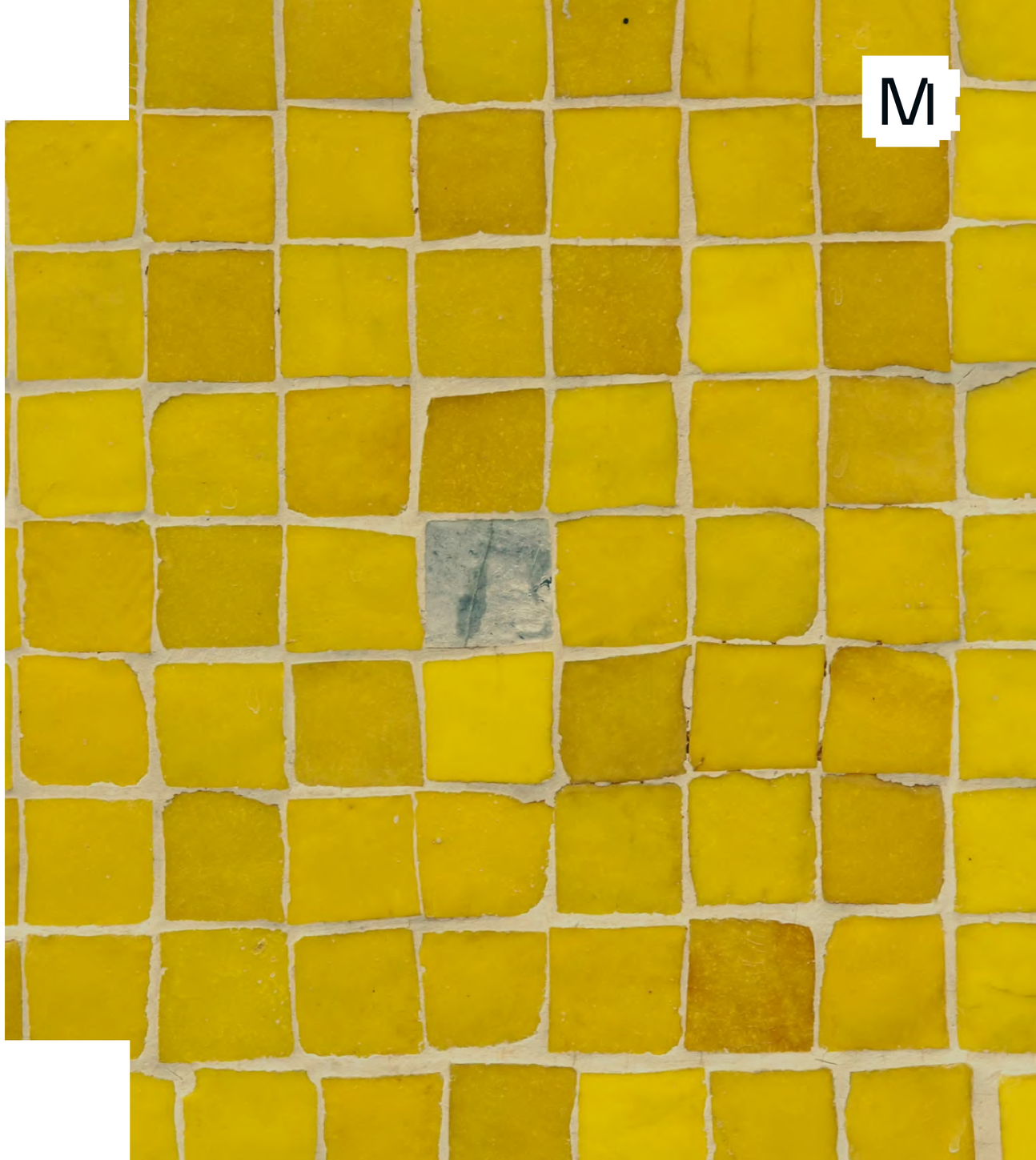
Improving Utilization

MMA is more fine grained in the N dimension

- `mma_m x mma_n x 16`
- `mma_m` is 64, 128 per SM
- `mma_n` is range(8, 257, 8) or range(16, 257, 16)

What can we do?

- Remember that NN workloads compute linear (MLP) blocks
- i.e. $A.B^T$



Swap AB

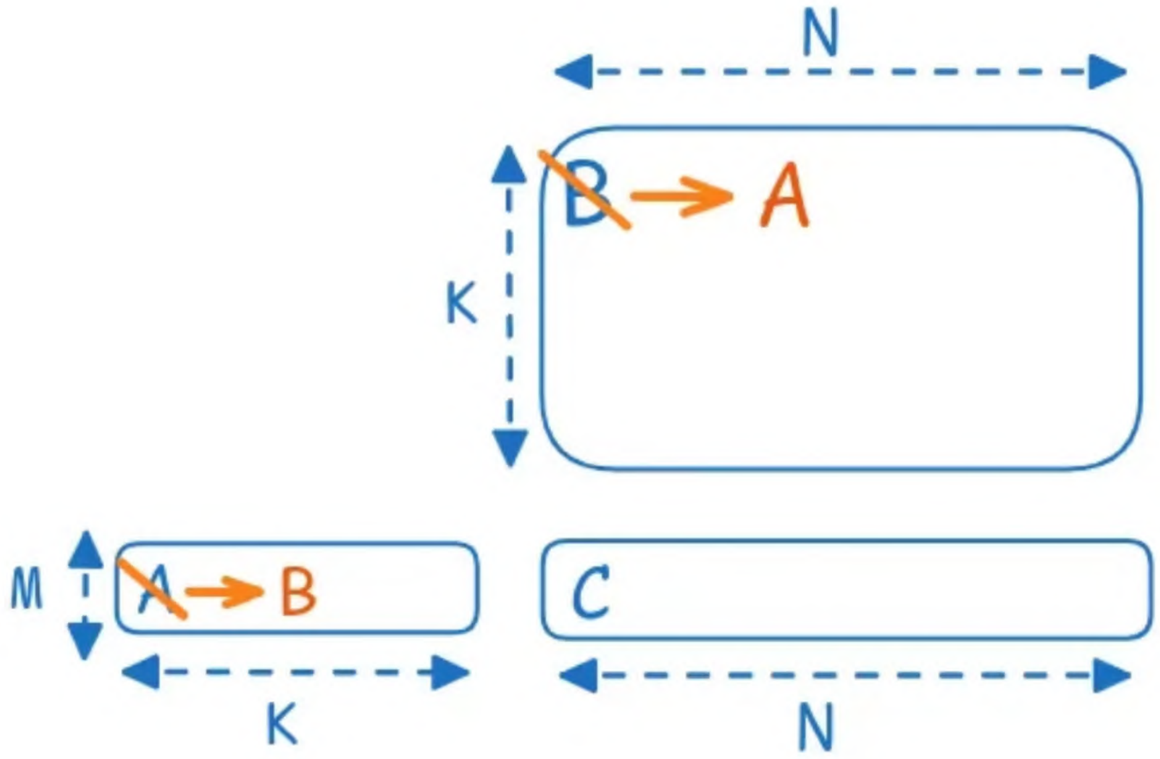
M

Recall

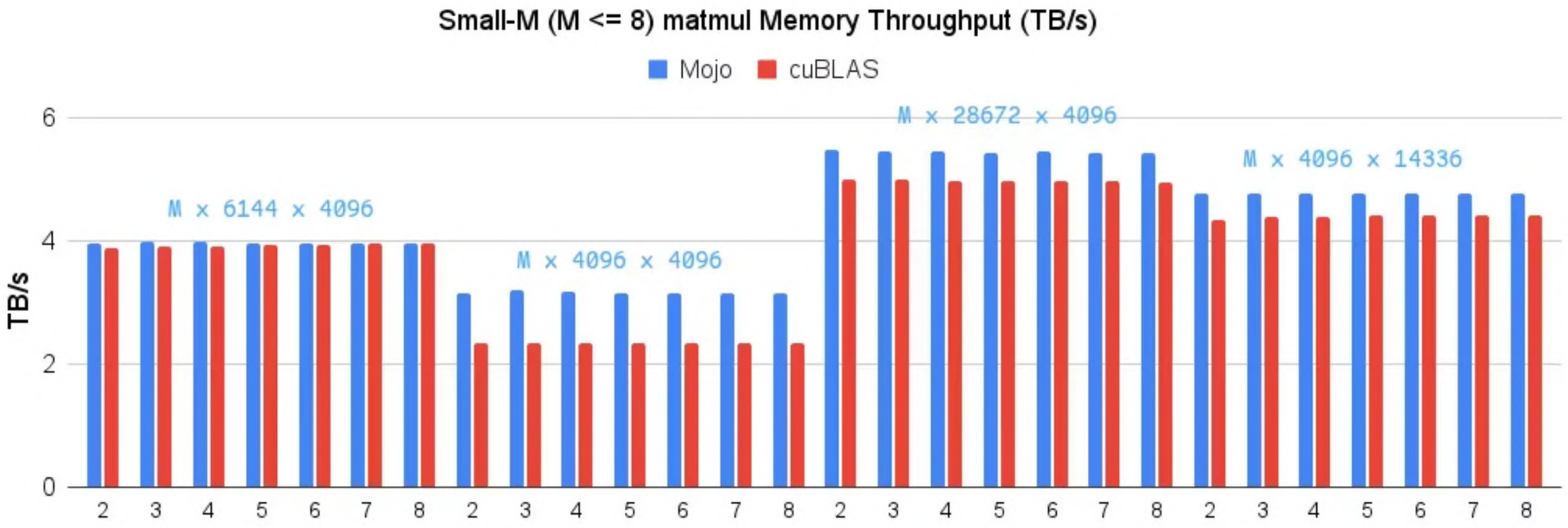
- A linear layer is defined as $C = A.B^T$
- From linear algebra $C^T = (A.B)^T = B^T.A^T$

What can we do?

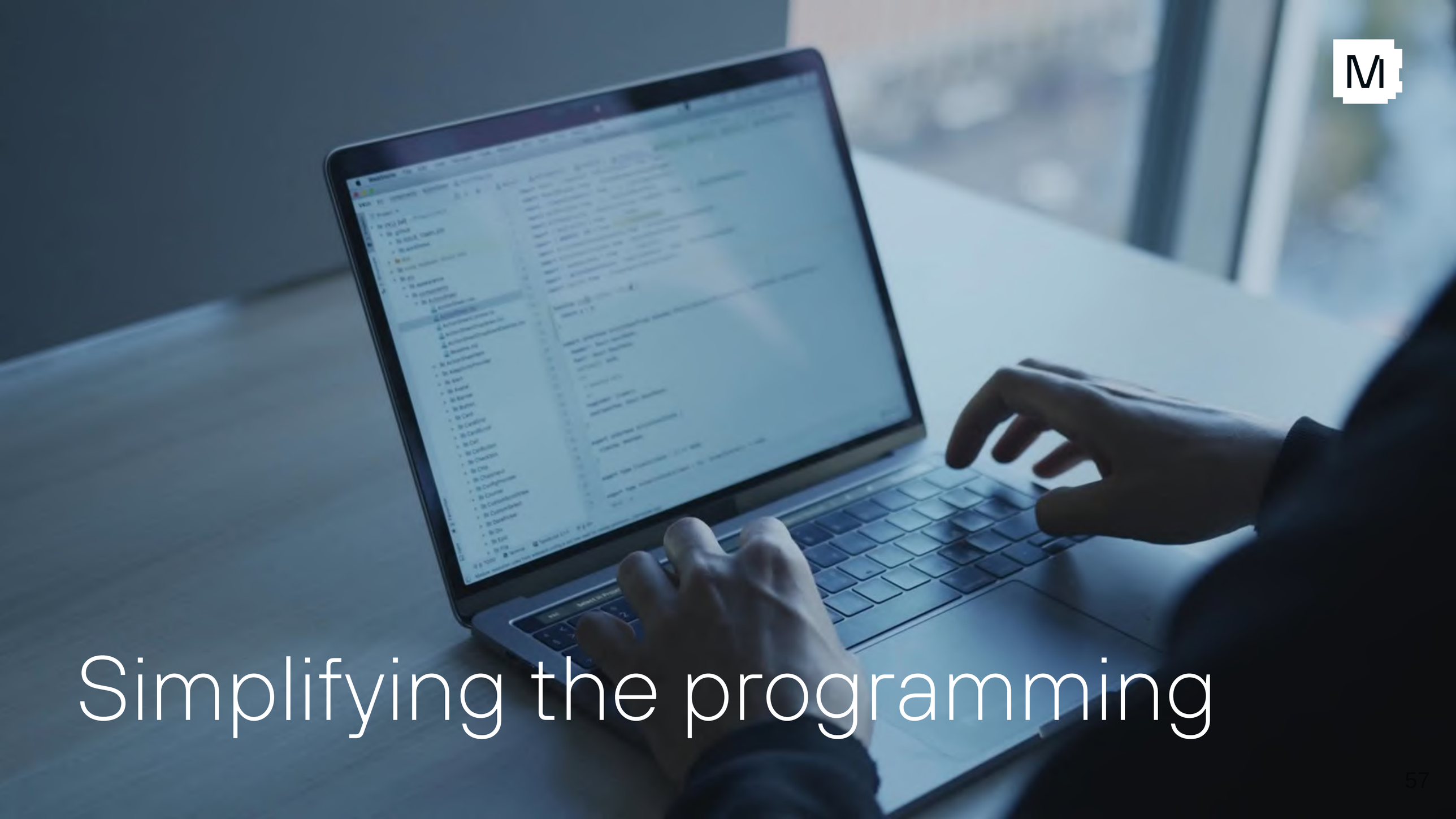
- Swap the arguments to the linear block
- Write in a transposed fashion



Upto 20% faster than cuBLAS



The E2E perf is much higher due to fusion



Simplifying the programming

Structured Kernels

Kernel Initialization

- Common structure across all GPUs:
  - Op definition,
  - Shared Memory structure,
  - RingBuffer initialization,
  - Data loaders, ...

```
alias K = b_layout.shape[1].value()
alias num_k_iters = ceildiv(K, Self.BK)

# Initialize WgmmaOp and SMem first
var wgmma_op = Self.WgmmaOp()
var smem = Self.SMem()

# Common initialization
var (
    warp_group_idx,
    warp_group_thread_idx,
    rank_m,
    rank_n,
    warp_id,
    lane_predicate,
) = Self.common_kernel_init()

# Create ring buffer
var ring_buffer = Self.build_ring_buffer(smem, warp_group_thread_idx)

# Create TileLoaderTMA loaders
var a_loader, b_loader = Self.build_tma_loaders(
    a_tma_op, b_tma_op, rank_m, rank_n
)

Self.pipeline_init()

# Calculate block swizzle
var block_idx_swizzle = Self.get_block_swizzle(lut_ptr)
var m_coord = block_idx_swizzle[1] * Self.BM
var n_coord = block_idx_swizzle[0] * Self.BN
```

Structured Kernels

Warp Group Specialization

- The kernel splits in producer and consumer, each with a generic main loop.

```
# Split thread blocks into producer and consumer warp groups
if warp_group_idx == 0:
    # Producer warp group
    _ = Self.setup_producer()

    if warp_id == 0 and lane_predicate:
        Self.producer_main_loop[num_k_iters=num_k_iters](
            UInt(m_coord),
            UInt(n_coord),
            0,  # k_start,
            a_loader,
            b_loader,
            ring_buffer,
        )
else:
    # Consumer warp groups
    var local_warp_group_idx, c_reg_tile, final_c_reg_tile = (
        Self.setup_consumer(warp_group_idx)
    )

    # Enter consumer mode
    with ring_buffer.consumer() as consumer:
        Self.consumer_main_loop[num_k_iters=num_k_iters](
            wgmma_op,
            local_warp_group_idx,
            final_c_reg_tile,
            c_reg_tile,
            consumer,
        )
```

Structured Kernels

Producer Main Loop

- The RingBuffer object orchestrates the interaction between Producer and Consumer warps.
- The Producer waits for available tiles to fill and delegates to the tile loaders for data movement.
- Notice the tile iteration structure at the bottom.

```
@always_inline
@parameter
fn producer_loop[
    num_pipeline_stages_to_unroll: Int,
](k_iter: Int):
    @parameter
    for j in range(num_pipeline_stages_to_unroll):
        var k_offset = UInt(
            k_coord + UInt(k_iter * num_pipeline_stages + j)
        )

        # Get the next available tile slot from the ring buffer.
        # The context manager ensures proper barrier synchronization.
        with ring_buffer.producer() as producer:
            with producer.get_tiles() as tiles:
                a_loader.load_tile(
                    tiles.a_tile,
                    tiles.barrier,
                    (m_coord, k_offset),
                )
                b_loader.load_tile(
                    tiles.b_tile,
                    tiles.barrier,
                    (n_coord, k_offset),
                )
@parameter
if num_remaining_k_iters == 0:
    for k_iter in range(num_full_k_iters):
        producer_loop[num_pipeline_stages](k_iter)
else:
    for k_iter in range(num_full_k_iters - 1):
        producer_loop[num_pipeline_stages](k_iter)
    producer_loop[num_remaining_k_iters](num_full_k_iters - 1)
```

Structured Kernels

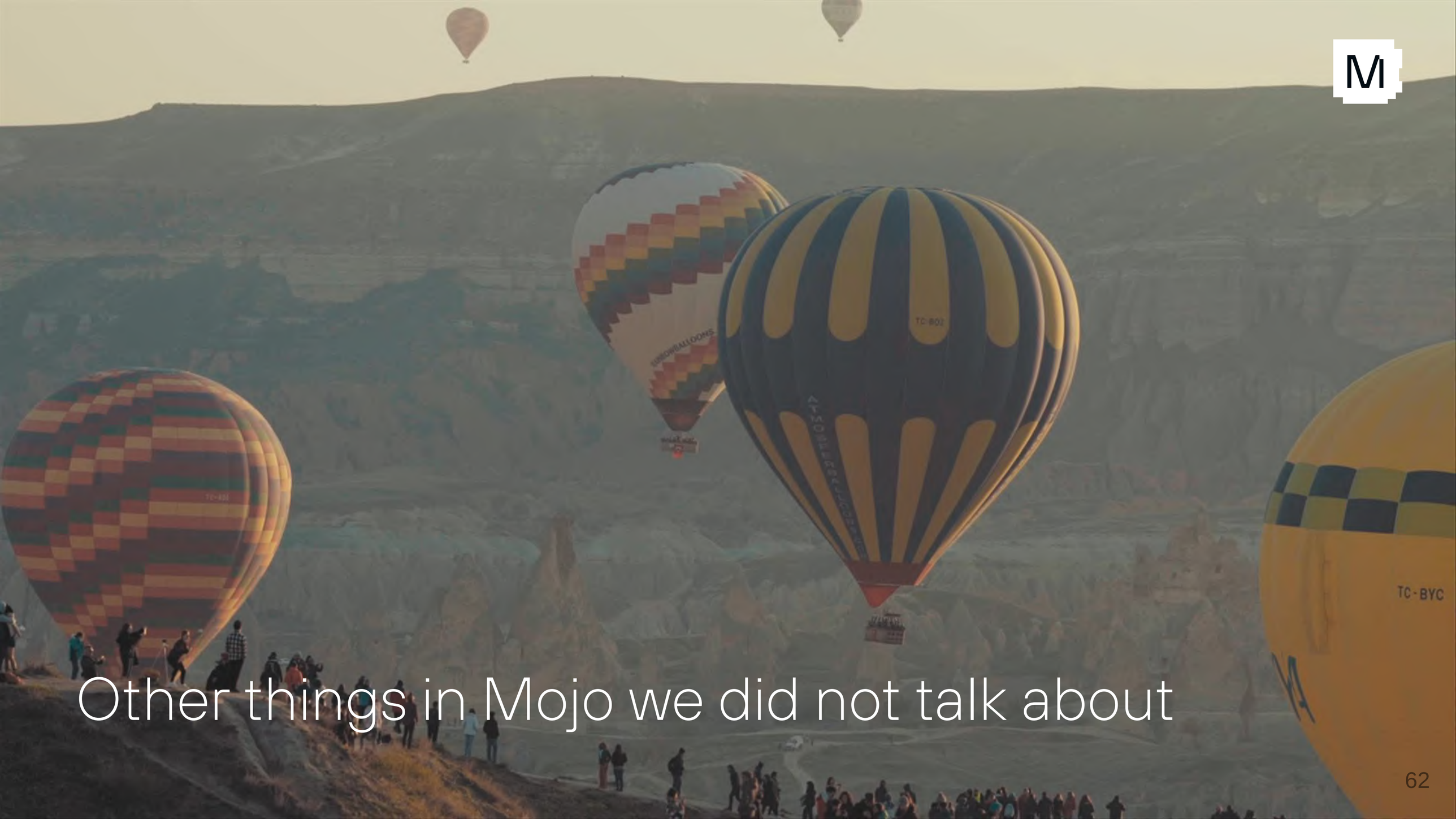
Consumer Main Loop

- Symmetric to the Producer
- Wait for available tiles
- Use Tiled Matmul Op (wgmma)
- Do the right thing for FP8
- Same tile iteration structure

Structured Kernels Deep Dive

- Stay tuned for an upcoming blog post

```
@always_inline
@parameter
fn consumer_loop[
    num_pipeline_stages_to_unroll: Int,
](k_iter: Int):
    @parameter
    for j in range(num_pipeline_stages_to_unroll):
        # Get the next available tile slot from the ring buffer.
        # The context manager ensures proper barrier synchronization.
        with ring_buffer.get_tiles() as tiles:
            Self.wgmma(
                wgmma_op,
                local_warp_group_idx,
                tiles.a_tile,
                tiles.b_tile,
                c_reg_tile,
            )

        @parameter
        if a_type is DType.float8_e4m3fn:
            fp8_promotion_iter += 1
            if fp8_promotion_iter == promotion_frequency:
                Self.promote_to_cuda_cores(c_reg_tile, final_c_reg_tile)
                fp8_promotion_iter -= promotion_frequency

@parameter
if num_remaining_k_iters == 0:
    for k_iter in range(num_full_k_iters):
        consumer_loop[num_pipeline_stages](k_iter)
else:
    for k_iter in range(num_full_k_iters - 1):
        consumer_loop[num_pipeline_stages](k_iter)
    consumer_loop[num_remaining_k_iters](num_full_k_iters - 1)
```



Other things in Mojo we did not talk about

Pythonic Syntax Native SIMD Type **Runs on CPU + GPU** Systems Programming

Metaprogramming Graph Compiler Safer Than C/C++ Faster Than Rust

GPU Agnostic Primitives **Tile Programming Model**

Tensor Core Access LayoutTensor DeviceContext

Zero Cost Abstractions Block/Thread Indexing

Python & C Interop New MLIR-based Compiler

VS Code Extension Call into LLVM & MLIR

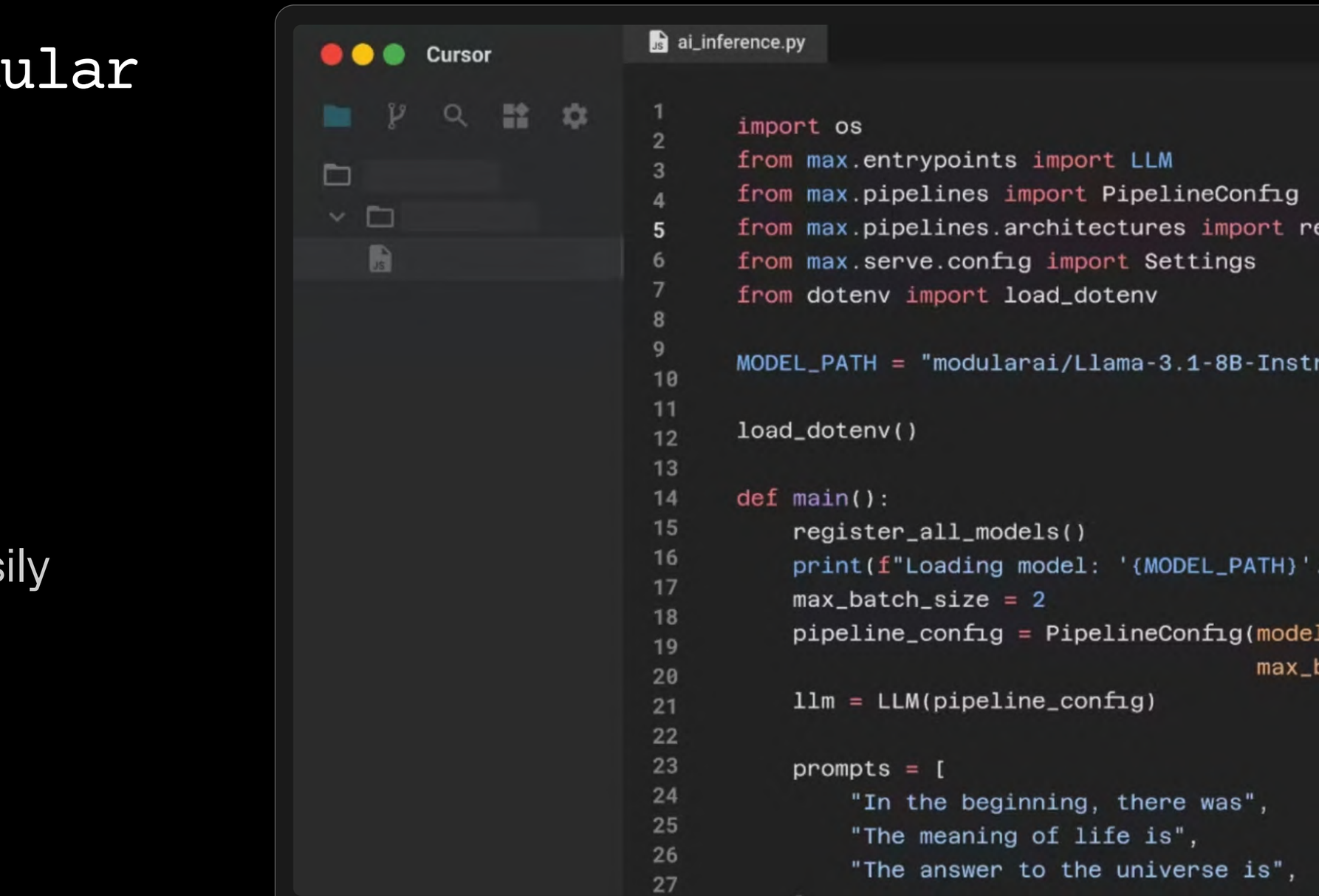
Dynamic Shapes Traits Open Source Libraries **Open Source AI Kernels**

Compile-time Parameters Python & C Interop Export to PTX & ASM


Get started today

pip install modular

- PyPi Support
- Easily Integrate
- Import Modules Easily



We're just getting started

Looking ahead

Near future directions for Modular

Platform: Next Gen AMD/NVIDIA HW, start into ASICs, add consumer GPUs for community support

Mojo: Expand from "performance" into "application level" programming

MAX: More models, expanded MoE support, Multi-node, and more performance optimizations

Mammoth: Multi-node and cluster level optimizations, scalability of models, users and hardaware, durablity of deployment.

Community: Keep teaching Mojo and MAX, enable a wide range of use-cases

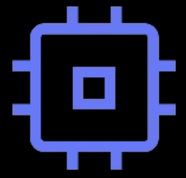
Modular is growing rapidly to support this!

Private & Confidential 2025

Let's build the most open GenAI platform!

Mojo Std Lib

Complete library & docs



GPU Kernels

With full version history

Model Pipelines

500+ open models supported

MAX Framework

Portable, hackable serving & more…
now open!

Mojo Compiler

Coming soon: 2026 or earlier

github.com/modular/modular = Apache2

Modular


Modular
THANK YOU

Q&A