Modular

Apple GPU Support in Mojo

Kolya Panchenko

EuroLLVM 2026

Why Apple GPU ?

- Large share of developers on Apple Silicon
- Unified Memory as a compelling compute advantage
- Good next GPU to enable that also has LLVM-based compiler

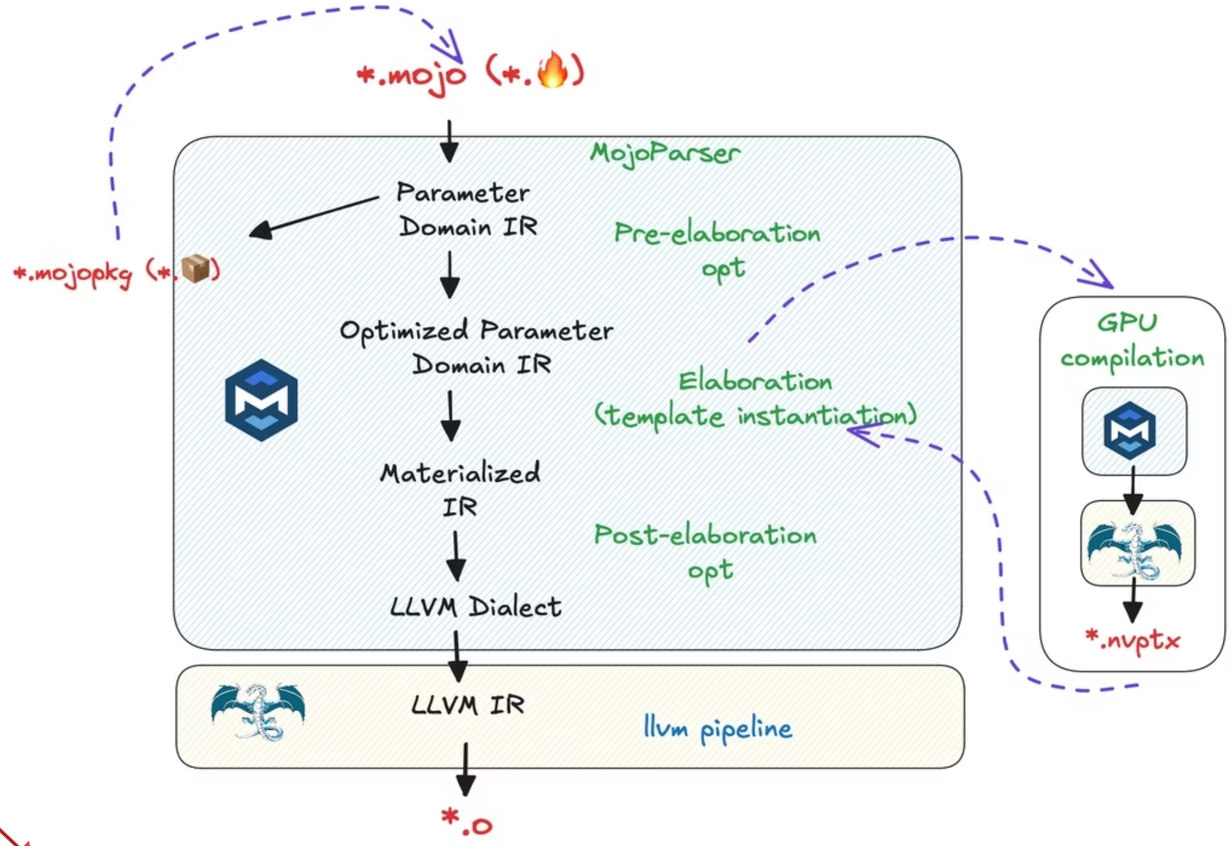
Mojo GPU Compilation Flow

```mojo
def main() raises:
    var ctx = DeviceContext()
    # ...

    # Launching a kernel
    ctx.enqueue_function[vector_addition](
        ...
        out_ptr,
        grid_dim=grid_dim,
        block_dim=5,
        target=NvidiaSM100a(),
    )

    # ....


def vector_addition(
    #...
    lhs_ptr: UnsafePointer[Float32, MutAnyOrigin],
):
    var global_tid = Int(global_idx.x)
    if global_tid < 10:
        out_ptr[global_tid] = lhs_ptr[global_tid] + rhs_ptr[global_tid]
```

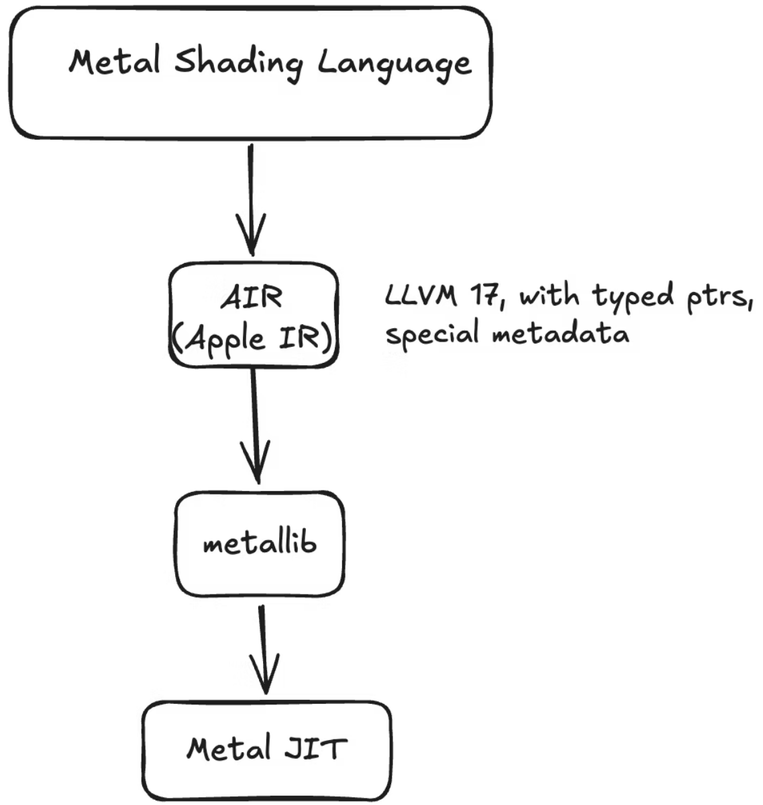
Host

Device

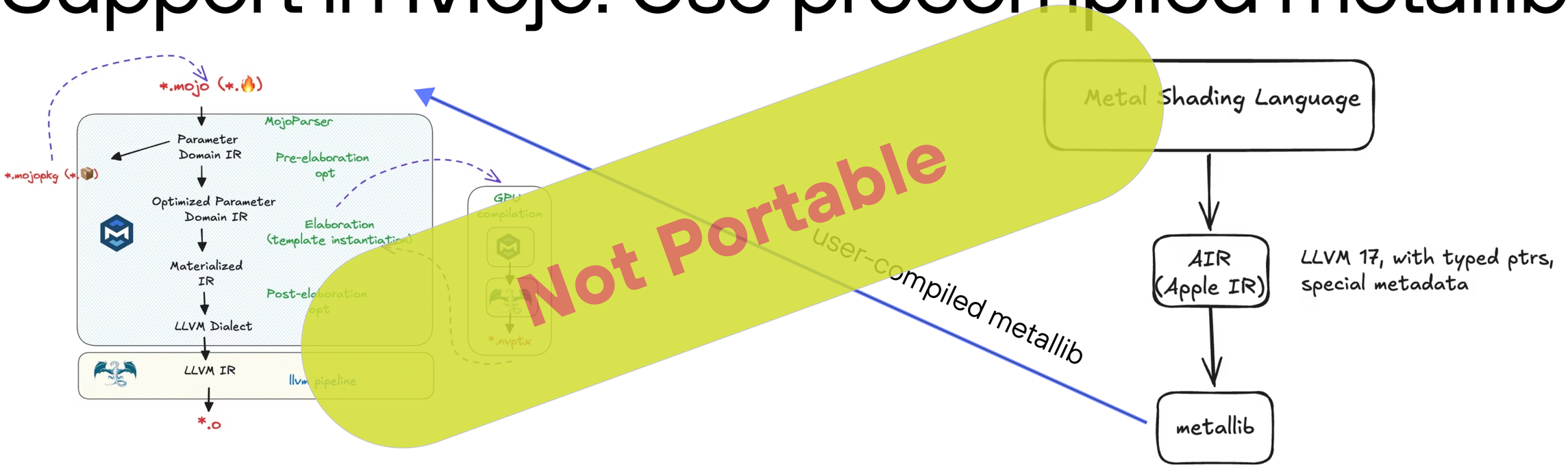
function to run on GPU



Metal Compiler in Xcode 16.4



Support in Mojo: Use precompiled metallib



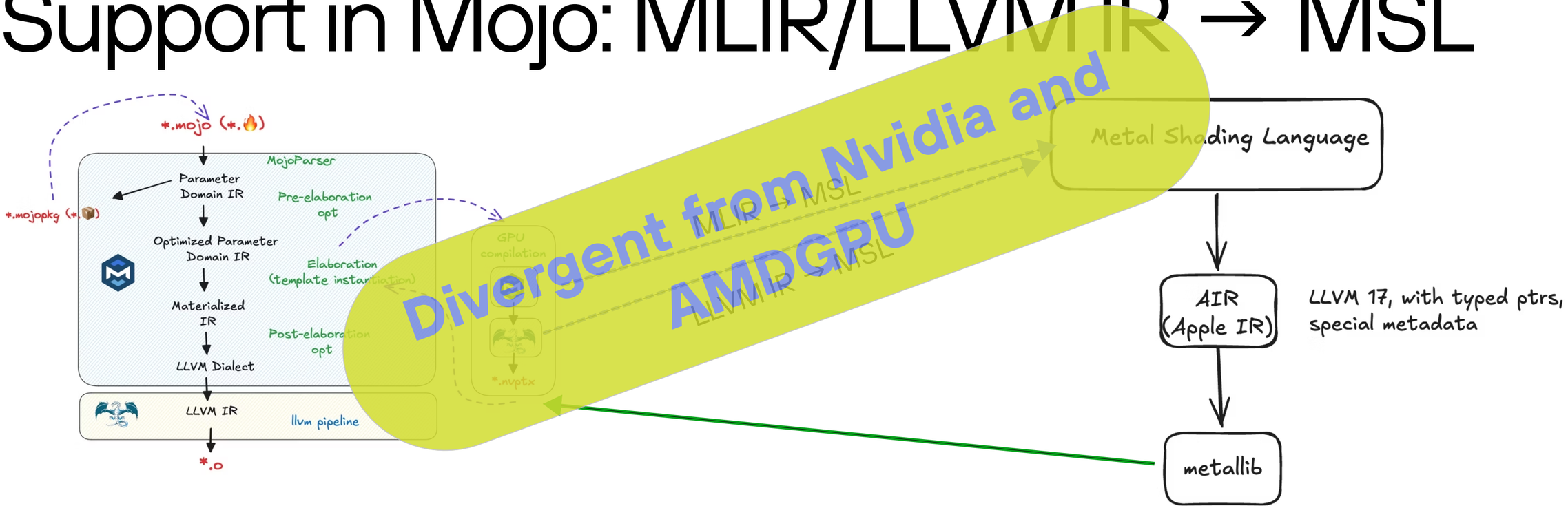
Pros

- very easy to implement

Cons

- Not portable
- Requires manual artifacts managing
- User has to use 2 different languages

Support in Mojo: MLIR/LLVM IR → MSL



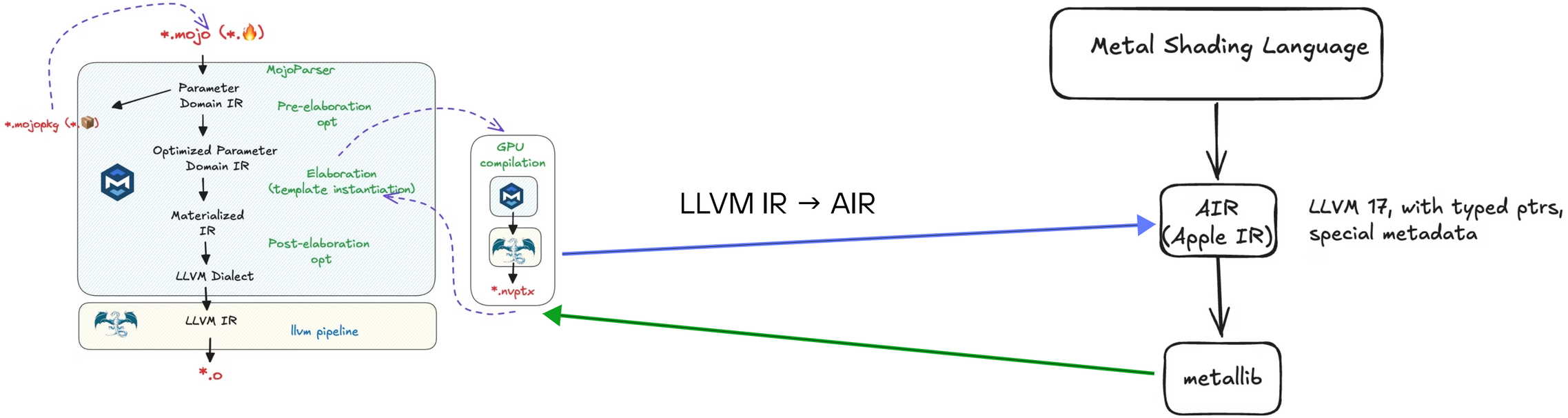
Pros

- Semantics checking by Metal
- Nicer error message

Cons

- Divergent from Nvidia and AMDGPU
- Lifting IR to higher-level representation is challenging

Support in Mojo: LLVM IR → AIR



Pros

- Similar path to Nvidia and AMDGPU

Cons

- May generate code that is not expected by Metal Compiler

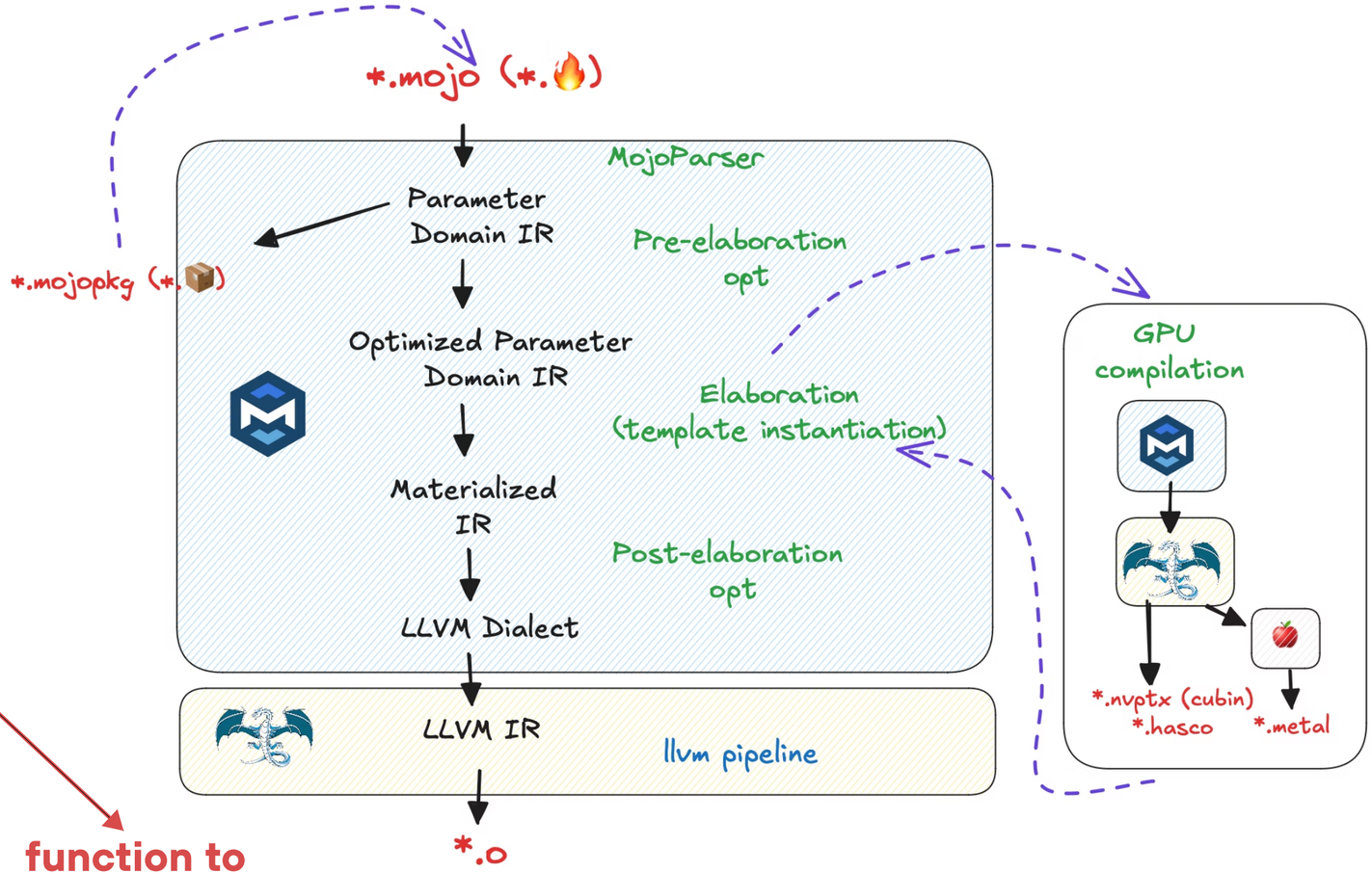
Mojo GPU Compilation Flow with Apple GPU

```mojo
def main() raises:
    var ctx = DeviceContext()
    # ...

    # Launching a kernel
    ctx.enqueue_function[vector_addition](
        ...
        out_ptr,
        grid_dim=grid_dim,
        block_dim=5,
        target=AppleGPU_M1(),
    )

    # ....


def vector_addition(
    #...
    lhs_ptr: UnsafePointer[Float32, MutAnyOrigin],
):
    var global_tid = Int(global_idx.x)
    if global_tid < 10:
        out_ptr[global_tid] = lhs_ptr[global_tid] + rhs_ptr[global_tid]
```

Host

Device

function to run on GPU



Problem: divergent programming model

// CUDA/ROCm

```
int tx = threadIdx.x;  // Thread X within block
int bx = blockIdx.x;   // Block X within grid
int bdx = blockDim.x;  // Block width (threads per block in X)
int gdx = gridDim.x;   // Grid width (blocks per grid in X)
```

// Metal

```
kernel void kernel(
    // Thread position within threadgroup
    uint3 tid [[thread_position_in_threadgroup]],
    // Threadgroup position within grid
    uint3 bid [[threadgroup_position_in_grid]],
    // Threadgroup dimensions
    uint3 tg_dim [[threads_per_threadgroup]],
    // Grid dimensions
    uint3 grid_dim [[threadgroups_per_grid]],
    // Global thread position
    uint3 gid [[thread_position_in_grid]]
) {
```

Solution: custom AIR intrinsic + transforms

```
def kernel():
    _ = thread_idx.x
```

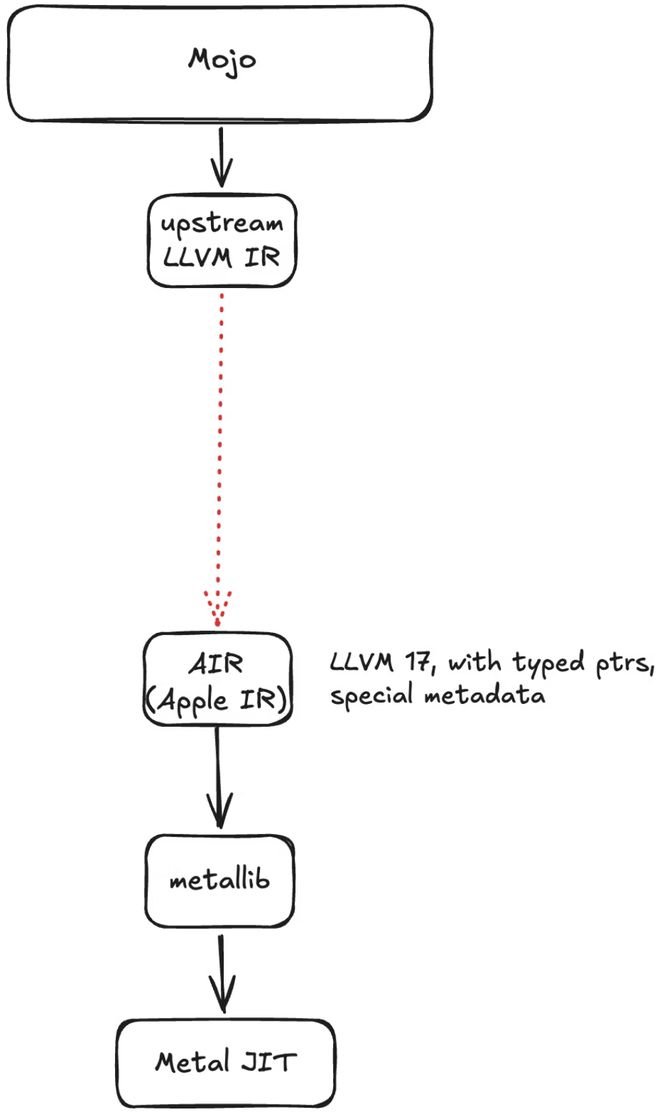
```
define void @kernel() {
  %tp = call i32 @air.thread_position_in_threadgroup.x()
  ret void
}
```

```
struct _ThreadIdx[ResultType: _FromInt](Defaultable, TrivialRegisterPassable):
    @always_inline("nodebug")
    @staticmethod
    def _get_intrinsic_name[dim: StringLiteral]() -> StaticString:
        comptime 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:
            CompilationTarget.unsupported_target_error[
                operation=__get_current_function_name(),
            ]()
```

```
define void @kernel(<3 x i32> %tid) {
  %1 = extractelement <3 x i32> %tid, i64 0
  ret void
}

!3 = !{i32 0, !"air.threadgroup_position_in_grid",
!"air.arg_type_name", !"uint3", !"air.arg_name",
!"threadgroup_position_in_grid",
!"air.arg_unused"}
```

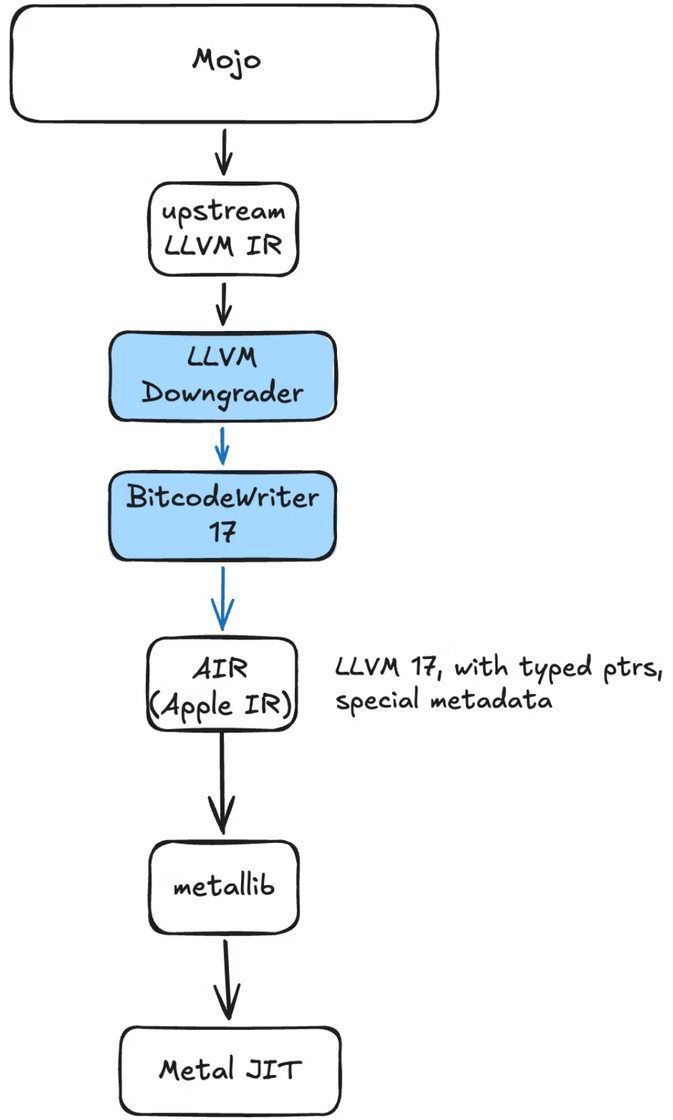
Problem: LLVM IR version mismatch

- Mojo weekly pulldowns
- Metal compiler ships with Xcode 16, which was based on LLVM 17.



Problem: LLVM IR version mismatch

- **Solution**: add LLVM Downgrader and Bitcode Writer for LLVM 17



Problem: undocumented backend

Some LLVM intrinsics are not supported by Metal

- *llvm.vector.[de]interleave.**
- *llvm.vector.reduce.**
- *llvm.masked.scatter*
- *llvm.stepvector*

Solution: replace them with simpler LLVM instruction before AIR

Problem: undocumented backend

fptrunc <8 x float> to <8 x bfloat> Works fine on M3, but crashes on M2

Troubleshooting: Used LLM to generate all supported conversions

xcrun metal -mllvm -print-after-all

Solution: Generate *@air.convert.f.X.f.Y* intrinsic for fptrunc

```
var bf16 = external_call["air.convert.f.v8f32.f.v8bf16", SIMD[DType.bfloat16, 8]](v8f32)
```

Vector Add on Apple GPU

```
def main() raises:
    var ctx = DeviceContext()
    # ...

    # Launching a kernel
    ctx.enqueue_function[vector_addition](
        ...
        out_ptr,
        grid_dim=grid_dim,
        block_dim=5,
        target=AppleGPU_M1(),
    )

    # ....


def vector_addition(
    #...
    lhs_ptr: UnsafePointer[Float32, MutAnyOrigin],
):
    var global_tid = Int(global_idx.x)
    if global_tid < 10:
        out_ptr[global_tid] = lhs_ptr[global_tid] + rhs_ptr[global_tid]
```

% mojo run vector_addition.mojo

Resulting vector with unsafe pointers: 3.75 3.75 3.75 3.75 3.75 3.75 3.75 3.75 3.75 3.75

What is enabled

- Apple GPU M2-M5, Xcode16+, Metal 3.2
- Almost all Mojo GPU Puzzles (excluding PyTorch and Nvidia-specific puzzles)
- GPT-2 can be executed on Apple GPU

Remaining work

- Performance tuning
- Support debugging

Conclusion

- Support target that are not in LLVM upstream
  - Need to have old LLVM Downgrader, BitcodeWriter
- Handle custom “intrinsics”
  - Treat them as external call and rely on BitcodeWriter and BitcodeReader
- Engage community to help to bringup hardware
  - After initial bringup is done, get help to test and provide implementation to some functions, like store_release

Thank you!

Acknowledgements:
Amir Nassereldine and The Mojo Language Team

Links

[0] 2025 US LLVM Developers' Meeting: Mojo GPU Compilation

[1] Mojo GPU Puzzles

[2] Apple Silicon GPU support in Mojo