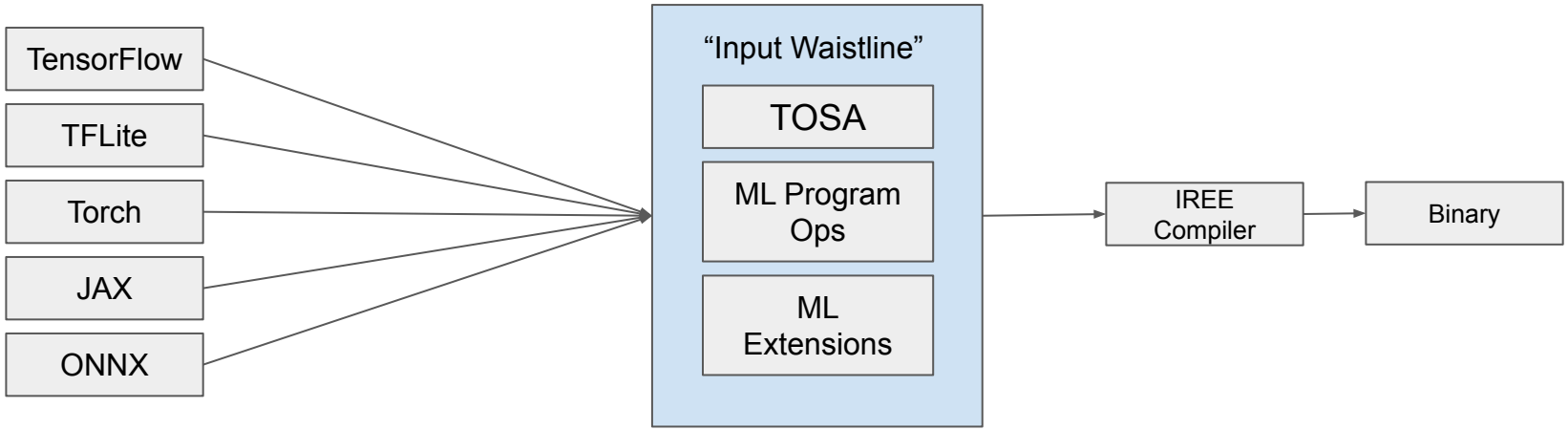
IREE's Input Dialect

Interfacing to ML Frontends

Who we are

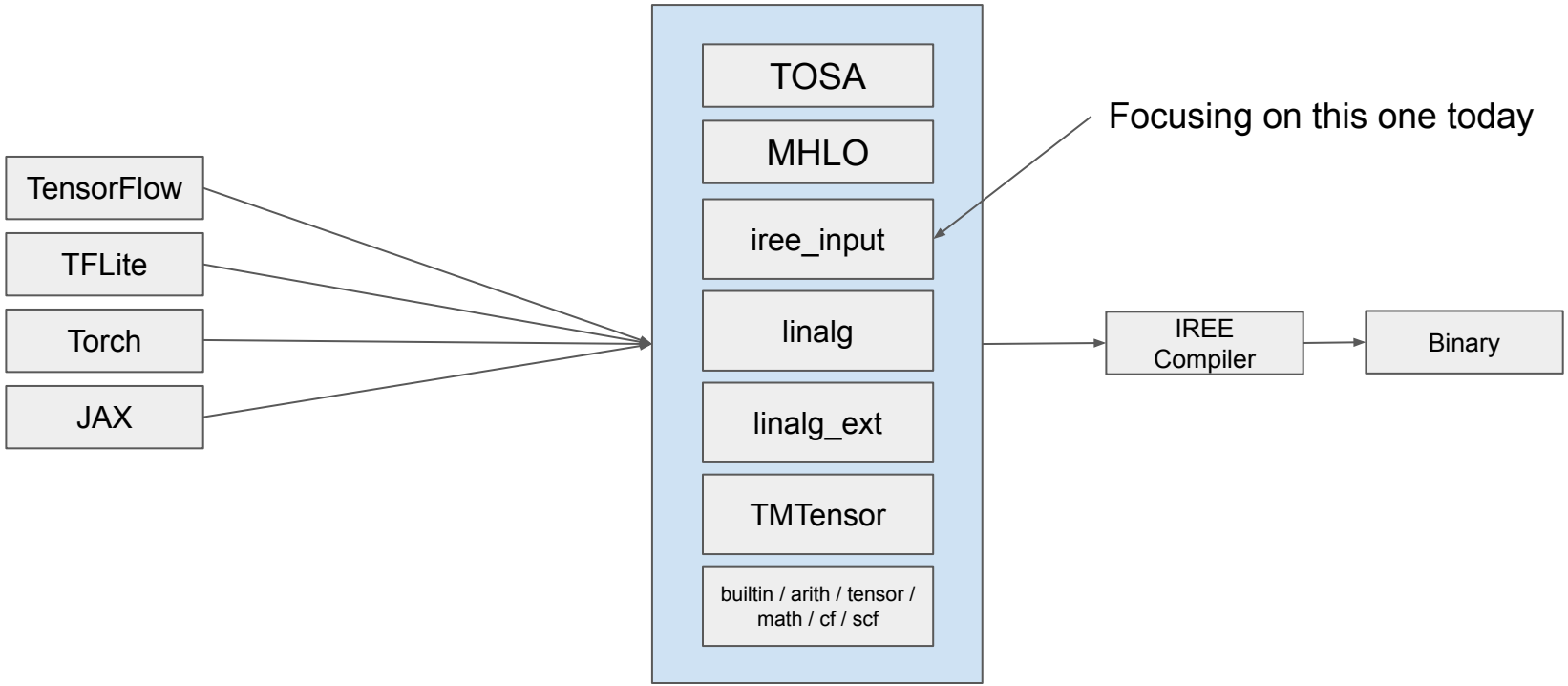
Preferred ML Compilation Pipeline



Why does a defined waistline matter?

- Ultimately desire some degree of program portability and stability guarantees
- Versioning/serialization/extension management requires a defined set
- Keeps the door accessible for top-level retargetability of entire compiler stacks
- Allows decoupling of system components and layers of the stack

What we actually have today – more of a kitchen sink



It’s not that dire – progress has been made

- TFLite fully legalizes to TOSA + iree_input *
- TensorFlow fully legalizes to MHLO + iree_input *
- JAX fully legalizes to MHLO + iree_input *
- Torch is growing legalizations to TOSA
- Rumors of better alignment with ONNX …

Once you see what iree_input is, you will understand that there isn’t much left. If we can agree on these final bits, we’re almost at completeness for the current generation of frontends.

* Plus builtin.module and builtin.func, but that is getting easier to abstract

What is the iree_input dialect?

- Frontends used to just target IREE's internal dialects directly.
- We boiled out the minimal useful set of things that didn't exist elsewhere and were needed for completeness.
- We wish we didn't need it and that there was just a common dialect upstream.
- Would also like "internal" upstream dialects to be distinguished from "interchange" dialects, which have stronger guarantees around versioning, serialization, etc.
    - Some of the things in iree_input are passingly similar to MLIR internal dialect constructs but we want them stabilized and specified concretely with different guarantees

Source Links

- iree-dialects/Dialect/Input/InputDialect.td
- iree-dialects/Dialect/Input/InputOps.td

Types

Types

- *buffer* – Reference counted, bag of bits (no dtype or shape) *
- *buffer_view* – Reference counted, view over a buffer, adding dtype and shape metadata
- *list* – Reference counted linear list, parameterized to contain either: ref-objects, primitives, or variants
- *variant* – Holds any value or ref-object
- *ptr* – Parameterized to point to a type (currently used to support global indirection).

* Not yet in input dialect, only in internal, but needed for some calling conventions.

Ops

Global Ops

- IREE globals store a value type or ref-counted object, which includes BufferViews. At the high level, they support tensor-types, which can be thought of as unique BufferViews whose contents are never modified (the compiler relaxes this constraint if able).
- A reference to a global can be taken, supporting indirect access.
- Globals are either initialized with an inline value or can have an initial store performed in a module initializer.

Global Ops – continued

- *global* : Defines a module-level, symbolic global
- *global.address* : Converts a global-symbol into a reference (Ptr)
- *global.load* : Loads a value from a global
- *global.store* : Stores a value into a global
- *global.load.indirect* : Loads a value from a global reference
- *global.store.indirect* : Stores a value into a global reference

BufferView Ops

- *buffer_view.rank* : Returns the rank of a BufferView
- *buffer_view.dim* : Returns the dim of a BufferView

Upstream Recommendation:

Model reference-counted types in MLIR core and make BufferView an implementation of them.

List Ops

- list.create : Creates a list with an initial capacity
- list.size : Current size of a list
- list.resize : Resizes a list
- list.get : Gets an element in a list
- list.set : Sets an element in a list

Upstream Recommendation:

Define reference-counted types upstream (lists are ref-counted here).

Tensor Ops

- Sometimes mirrors ops in core dialects, but often in a more specific way.
- All ops in this category are parameterized with explicit rank and values for any dynamic dims (i.e. they exist as the *result* of lowering a program requiring shape inference to discover all shape relationships and make them concrete).
- All shape dims are loose scalars passed in variadics.
- Externally, IREE allows commingling of implicit-shape dialects, so long as they have enough notion of shape inference built-in to concretize. Internally, IREE's invariant is that all tensor ops carry, self contained explicit shapes, and these ops bridge the divide in user programs we have discovered.

Tensor Ops – continued

- tensor.reshape : Reshapes to a new shape without modifying contents
- tensor.load : Loads an element from a tensor into a scalar or vector
- tensor.store : Returns a tensor with the element at the given index set to the given value.
- tensor.splat : Returns a tensor with all elements initialized to a given value.
- tensor.clone : Clones the input tensor to a tensor with an identical value (used in various boundary/interop cases to introduce explicit shape).
- tensor.slice : Slices out a sub-region of a tensor.
- tensor.update : Updates a sub-region of tensor, returning the result.
- tensor.trace : Logs/traces a set of tensors with an identifying key

Tensor Ops – continued

Recommendation:

Define explicit “ranked tensor” ops as first order concepts and ensure that they capture explicit dimensions (in parallel to the higher level, implicit dimension, generic tensor ops).

Concurrency Ops

- IREE directly exposes concurrency as dispatch of a SIMT program across an nd-grid.
- Subset of “GPU” programming model exposed in the “gpu” dialect.
- Suitable for GPU and CPU.

Recommendation:

Define a new “simt” dialect as a higher level abstraction suitable for both GPU/CPU and lower the user programming model into it.

General Recommendations

Dialect upstream?

- Wouldn't just "move" iree_dialects upstream:
  - It is an amalgam which was derived experimentally by reducing multiple ML frontends and working to compile real programs from them
- It (and IREE generally) highlights some general purpose gaps upstream:
  - No modeling of reference-counted types
  - Explicit shaped, ranked tensor manipulators solve important problems
  - List types and transformations would be really valuable
  - Generalized SIMT concurrency primitives have been useful
- Even if every op/type here existed in a core dialect, we probably want a dedicated dialect to represent present-day ML program modules.
  - All current frontends reduce to a reasonable subset
  - In combination with serialization/versioning of TOSA, could provide a complete waistline
  - Would promote ops from iree_input, evaluating as we go

Define an “ml_program” dialect?

- ML frontends have moved fast in the past but have largely converged in terms of top-level constructs.
- Having a stable target for them would help further convergence and sharing when connecting them to the compiler ecosystem.
- There will always be a next thing, but that thing can have its own dialect when it emerges.
- If “ml_program” brought common structural elements, this would nicely complement more op-centric dialects like TOSA/MHLO/etc, driving further convergence.
- Would give us a place to work on other features (i.e. out-of-line constants, mechanisms of program sharding, whole program analyses, etc).

Extra bits

- There are still some bits that are inherited from core dialects or not fully mirrored in IREE's input dialect.
- An "ml_program" dialect could also include:
  - *module / func / call* : Allowing ML programs to bring their own module/func would aid in making them self contained, also allowing further differentiation with respect to different kinds of code units (graph vs cfg, etc).
  - *initializer* : IREE internally has a FunctionLike *initializer* op, which is a () -> () function run at module initialization time. This has not yet been mirrored in the input dialect.

Where to place it?

- Assuming that repository re-layout happens: https://discourse.llvm.org/t/rfc-restructuring-of-the-mlir-repo/4927/50
- We already have “targets” that represent defined egress points in terms of self-contained dialects that bridge out.
- Can have “sources” that represent ingress points and are held to standards of compatibility, serializability, etc in line with higher expectations.
  - sources/MlProgram
  - sources/Tosa
  - sources/MlPrimitives? (hunting for a name for structured ops that form the basis for higher order ops and augment the more algebraic kernel centric ops focused on in dialects like Tosa – contractions, sort, scan, etc)
- If defined right, we could start to get stable interchange points for sourcing programs from present day ML systems.