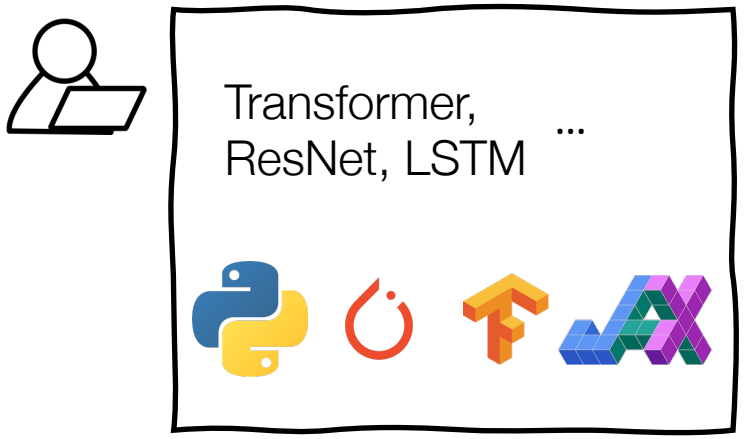
Machine Learning Compilation

Summary and Future Directions

Tianqi Chen

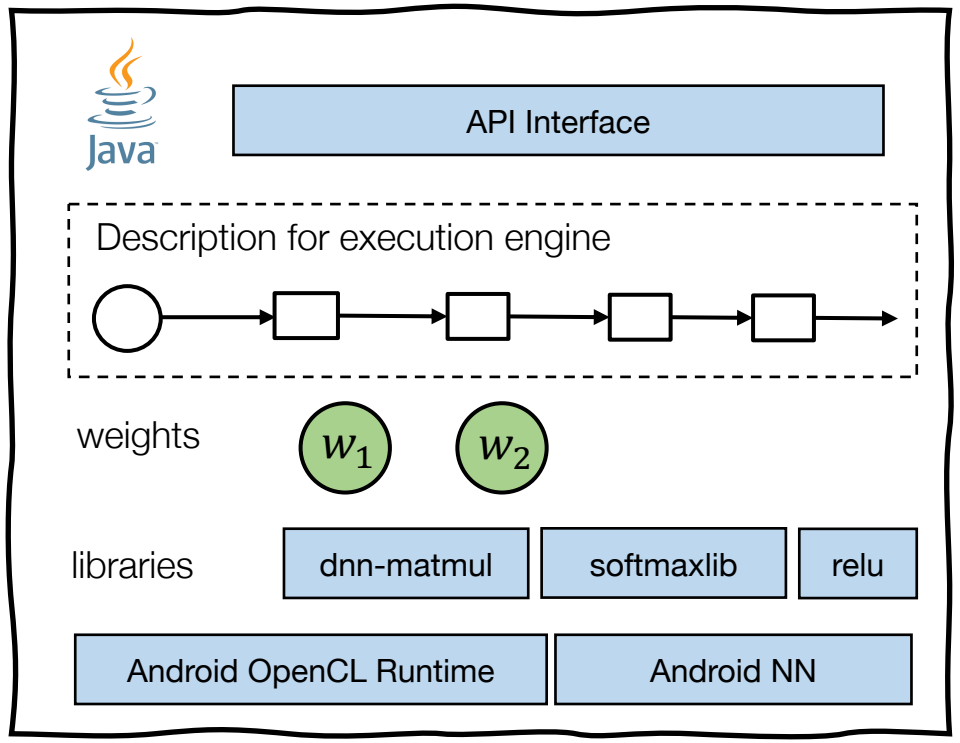
Machine Learning Compilation

Development Form



MLC Process

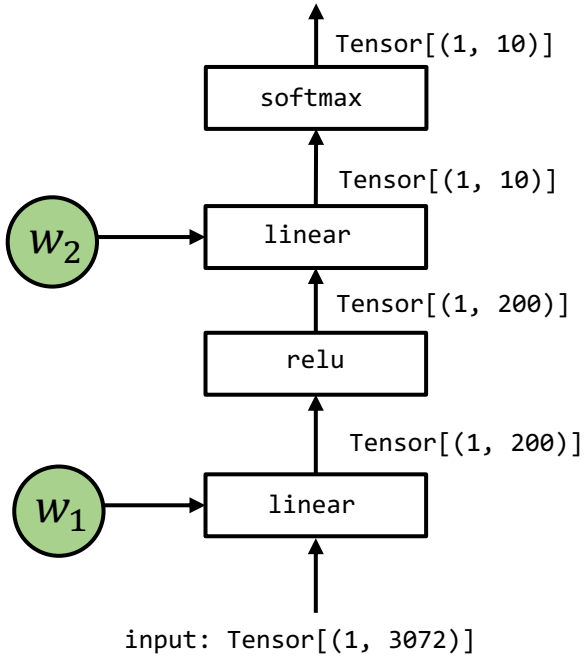
Deployment Form



Machine learning compilation (MLC) is the process to transform and optimize machine learning execution from its development form to deployment form.

An example instance of deployment form

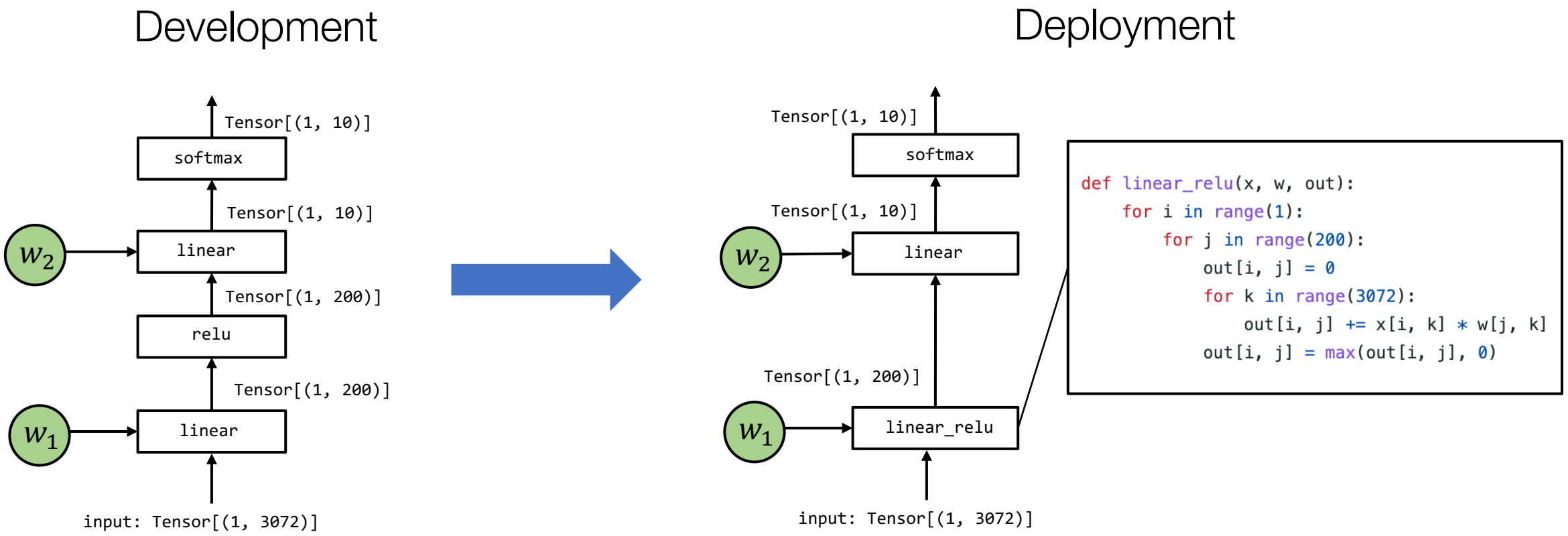
Key Elements in Machine Learning Compilation



Tensor multi-dimensional array that stores the input, output and intermediate results of model executions.

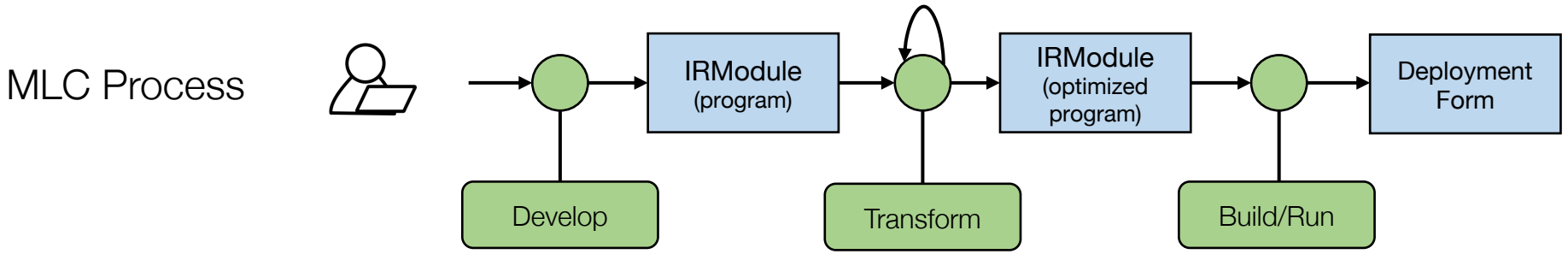
Tensor Functions that encodes computations among the input/output. Note that a tensor function can contain multiple operations

MLC as Tensor Function Transformation

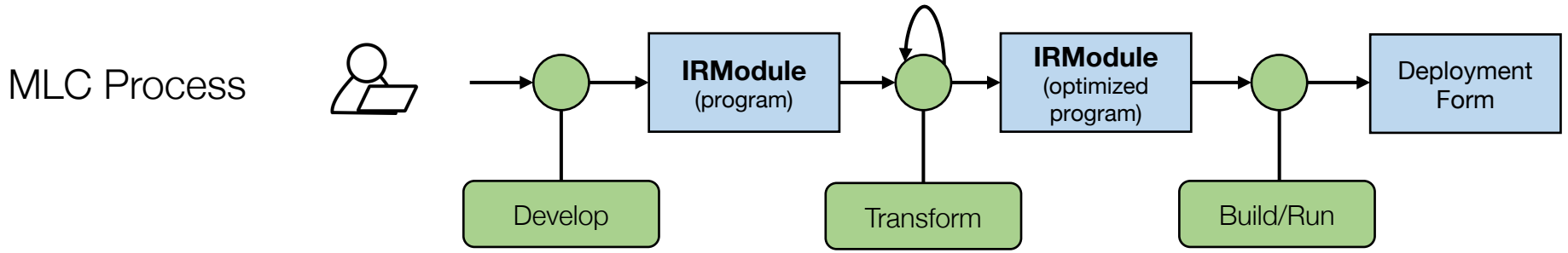


Most MLC process can be viewed as transformation among tensor functions (that can be represented with different abstractions).

Overall MLC Process



IRModule: "noun" in MLC Process



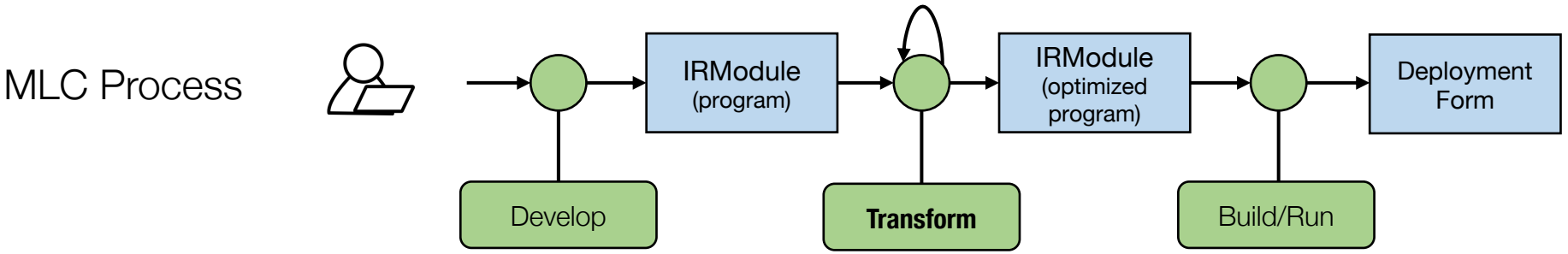

```python
@tvm.script.ir_module
class MyModuleMixture:
    @T.prim_func
    def linear0(X: T.Buffer[(1, 784), "float32"],
                W: T.Buffer[(128, 784), "float32"],
                Y: T.Buffer[(1, 128), "float32"]):
        T.func_attr({"global_symbol": "linear0", "tir.noalias": True})
        for i, j, k in T.grid(1, 128, 784):
            with T.block("Y"):
                vi, vj, vk = T.axis.remap("SSR", [i, j, k])
                with T.init():
                    Y[vi, vj] = T.float32(0)
                Y[vi, vj] = Y[vi, vj] + X[vi, vk] * W[vj, vk]

    @R.function
    def main(x: Tensor((1, 784), "float32"),
             w0: Tensor((128, 784), "float32"),
             w1: Tensor((10, 128), "float32")):
        with R.dataflow():
            lv0 = R.call_tir(linear0, (x, w0), (1, 128), dtype="float32")
            lv1 = R.nn.relu(lv0)
            out = R.call_tir("env.linear", (lv1, w1), (1, 10), dtype="float32")
            R.output(out)
        return out
```

TensorIR multi-dimensional loops, tensor programs.

Library function functions to fallback to existing manual implementations

Interactions among computational graph, TensorIR and library functions.

Transformations: Center of MLC Processes



TensorIR-level transformations among primitive tensor functions.

Computational graph transformations: operator fusion.

Mapping to library functions.

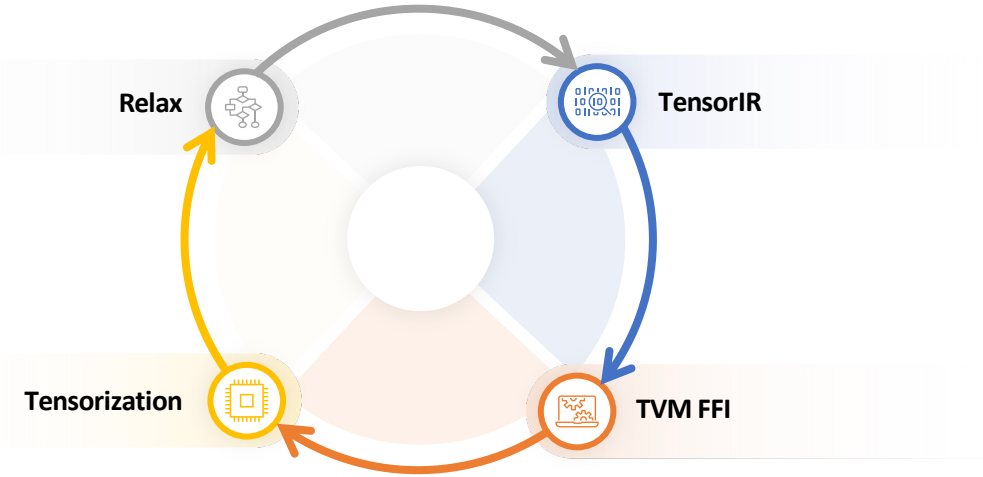
Example Case Study: Machine Learning Framework

Import into computational graph

Simple computational graph rewriting (e.g. fusion replacement)

Map to library function calls.

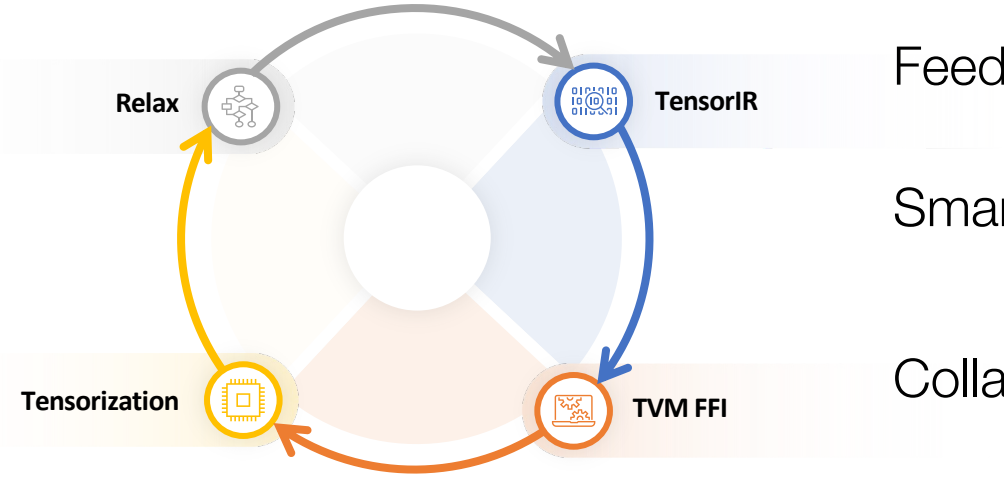
This Class: Interactive Transformation Process



Four categories of abstractions in the same IRModule, interacting with each other.

```python
@tvm.script.ir_module
class MyModuleMixture:
    @T.prim_func
    def linear0(X: T.Buffer[(1, 784), "float32"],
                W: T.Buffer[(128, 784), "float32"],
                Y: T.Buffer[(1, 128), "float32"]):
        T.func_attr({"global_symbol": "linear0", "tir.noalias": True})
        for i, j, k in T.grid(1, 128, 784):
            with T.block("Y"):
                vi, vj, vk = T.axis.remap("SSR", [i, j, k])
                with T.init():
                    Y[vi, vj] = T.float32(0)
                Y[vi, vj] = Y[vi, vj] + X[vi, vk] * W[vj, vk]

    @R.function
    def main(x: Tensor((1, 784), "float32"),
             w0: Tensor((128, 784), "float32"),
             w1: Tensor((10, 128), "float32")):
        with R.dataflow():
            lv0 = R.call_tir(linear0, (x, w0), (1, 128), dtype="float32")
            lv1 = R.nn.relu(lv0)
            out = R.call_tir("env.linear", (lv1, w1), (1, 10), dtype="float32")
            R.output(out)
        return out
```

Remarks Future Directions



Feedback and cross-layer optimizations.

Smarter automatic approaches for specialized hardware.

Collaboration between engineers and compilation system.

Many ideas can be achieved through incremental develop, transform, and build.

Continue to follow this course at mlc.ai and other related course (dlsyscourse.org)