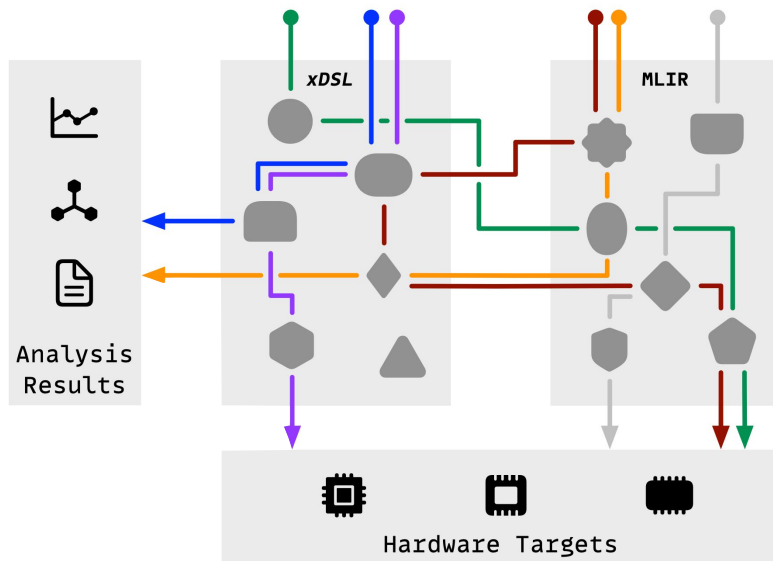


# xDSL: Prototyping MLIR in Python



Sasha Lopoukhine, Mathieu Fehr, and many more!



THE UNIVERSITY  
*of* EDINBURGH



**Research**



**Open-Source Development**



**Teaching**  
(150 students / year)





SSA and Operations

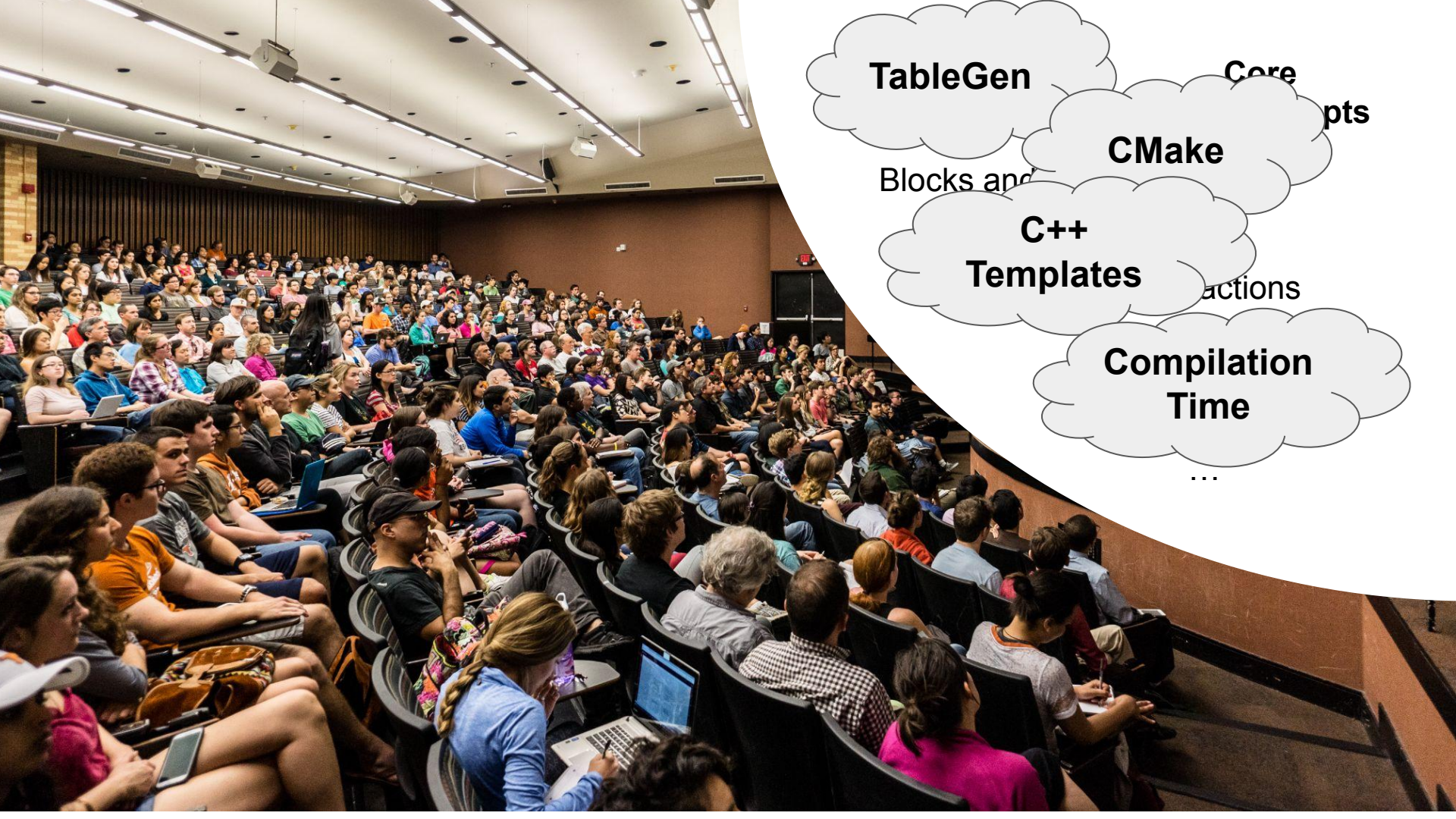
**Core  
Concepts**

Blocks and Regions

Dialects as abstractions

Peephole rewrites

...



**TableGen**

**Core  
pts**

**CMake**

**Blocks and**

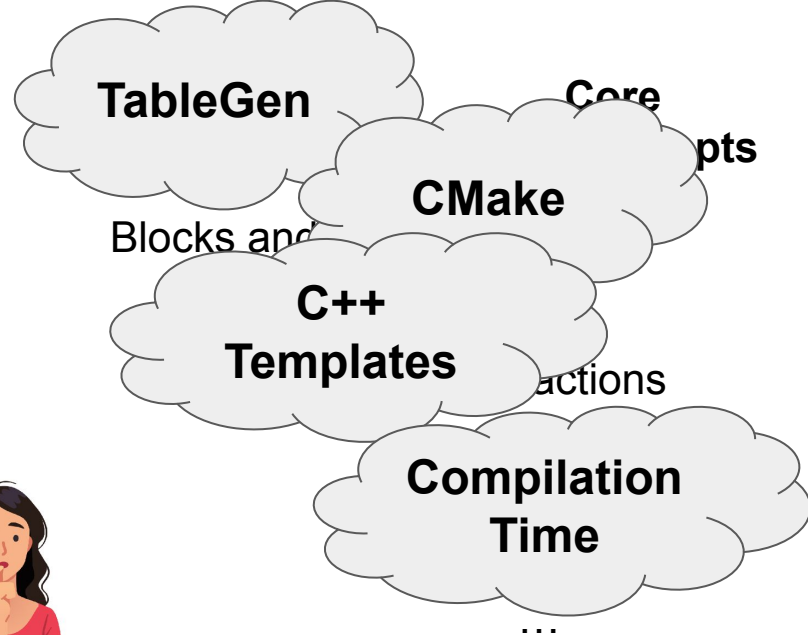
**C++  
Templates**

**actions**

**Compilation  
Time**

...

# How to teach MLIR?



*MLIR is hard to get started with!*

# xDSL



*What if we reimplemented MLIR in Python?*

# Operation definition in xDSL

```
@irdl_op_definition
class AddOp(IRDLOperation):
    name = "toy.add"
    lhs: Annotated[Operand, AnyTensorTypeF64]
    rhs: Annotated[Operand, AnyTensorTypeF64]
    res: Annotated[OpResult, AnyTensorTypeF64]
```



# Module builders in xDSL

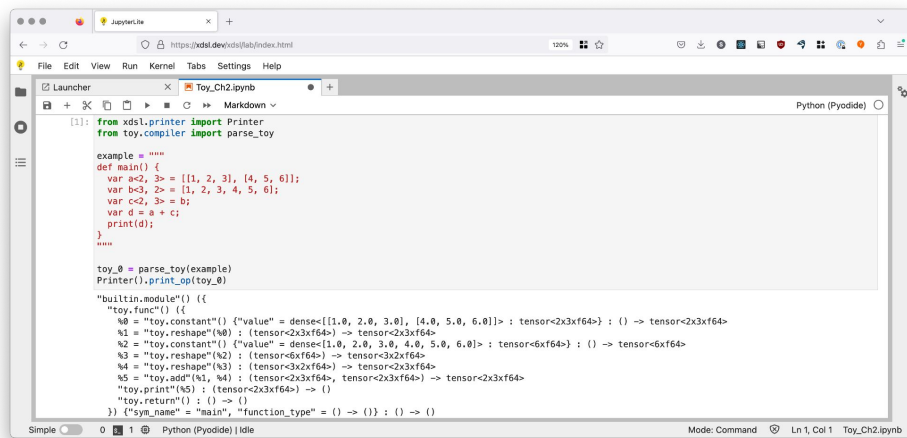
```
@Builder.implicit_region
def main() → None:
    x = toy.ConstantOp.from_list([1, 2, 3, 4, 5, 6], [2, 3]).res
    y = toy.AddOp(x, x).res
    toy.PrintOp(y)
    toy.ReturnOp()

toy.FuncOp("main", main_type, main)
```

# Reshape definition in xDSL

```
class ReshapeReshapeOpPattern(RewritePattern):
    @op_type_rewrite_pattern
    def match_and_rewrite(self, op: ReshapeOp, rewriter: PatternRewriter):
        """Reshape(Reshape(x)) = Reshape(x)"""
        if isinstance(op.arg, OpResult) and isinstance(op.arg.op, ReshapeOp):
            t = cast(TensorType[Float64Type], op.res.typ)
            new_op = ReshapeOp(op.arg.op.arg, t)
            rewriter.replace_matched_op(new_op)
```

# Easy to get started



```
from xdsl.printer import Printer
from toy.compiler import parse_toy

example = """
def main() {
  var a<2, 3> = [[1, 2, 3], [4, 5, 6]];
  var b<3, 2> = [[1, 2, 3, 4, 5, 6];
  var c<2, 3> = b;
  var d = a + c;
  print(d);
}
"""

toy_0 = parse_toy(example)
Printer().print_op(toy_0)

"builtin.module"() {
  "toy.func"() {
    %0 = "toy.constant"() ("value" = dense<[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]> : tensor<2x3xf64>) : () -> tensor<2x3xf64>
    %1 = "toy.reshape"(%0) : (tensor<2x3xf64>) -> tensor<2x3xf64>
    %2 = "toy.constant"() ("value" = dense<[[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]]> : tensor<6xf64>) : () -> tensor<6xf64>
    %3 = "toy.reshape"(%2) : (tensor<6xf64>) -> tensor<3x2xf64>
    %4 = "toy.reshape"(%3) : (tensor<3x2xf64>) -> tensor<2x3xf64>
    %5 = "toy.add"(%1, %4) : (tensor<2x3xf64>, tensor<2x3xf64>) -> tensor<2x3xf64>
    "toy.print"(%5) : (tensor<2x3xf64>) -> ()
  }
  "toy.return"() : () -> ()
} ("sym_name" = "main", "function_type" = () -> () : () -> ())
```

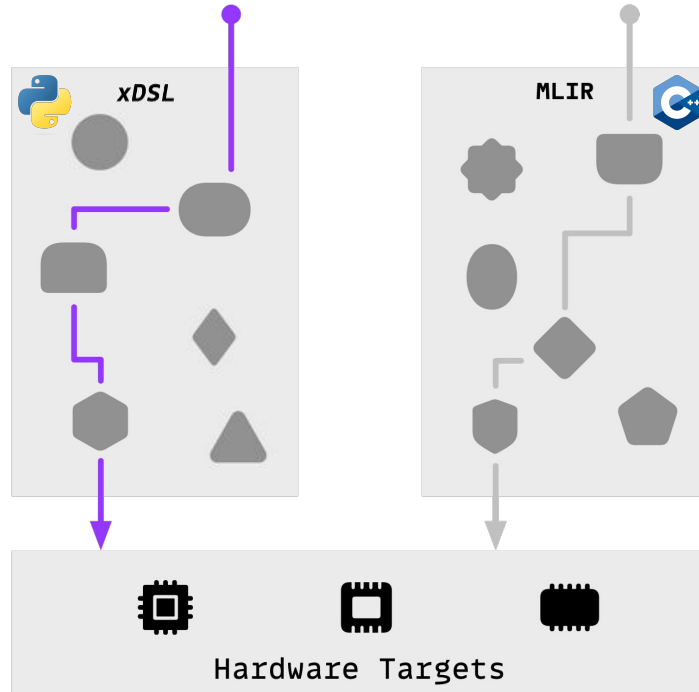
pip install xdsl

Fast iteration loop

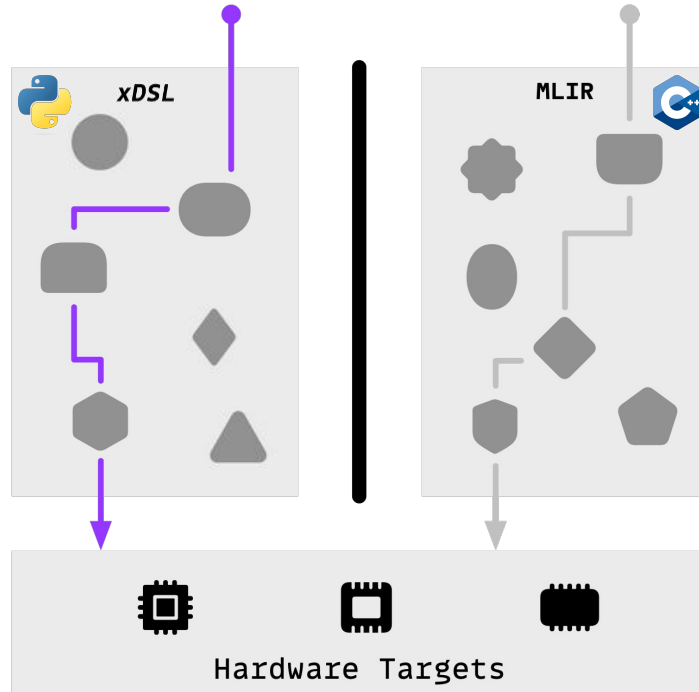
Fully hackable

[xdsl.dev/xdsl](https://xdsl.dev/xdsl)

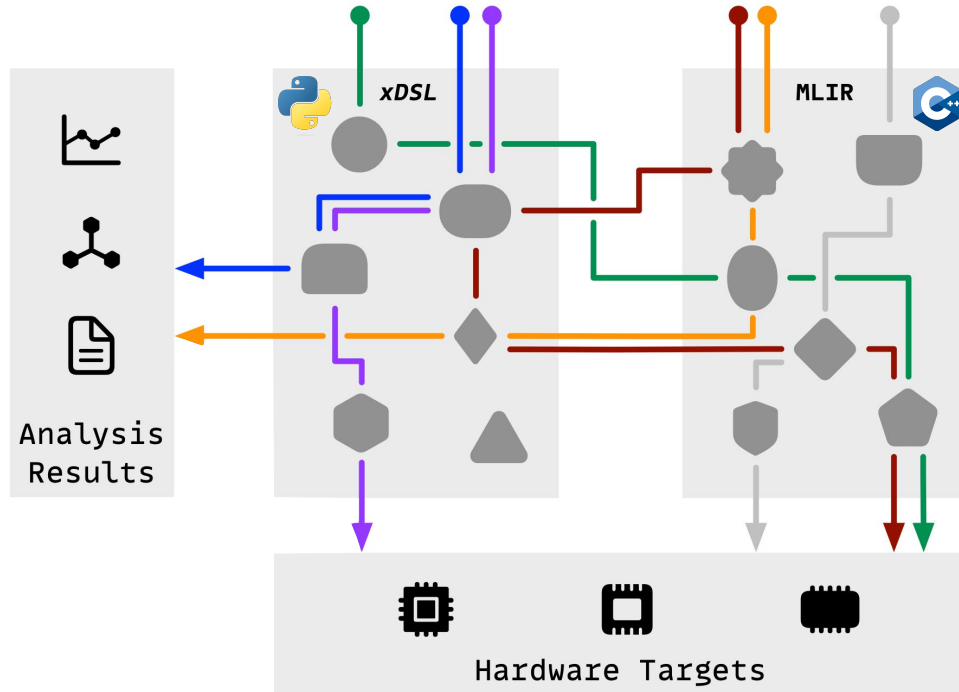
# xDSL: A Python framework replicating MLIR



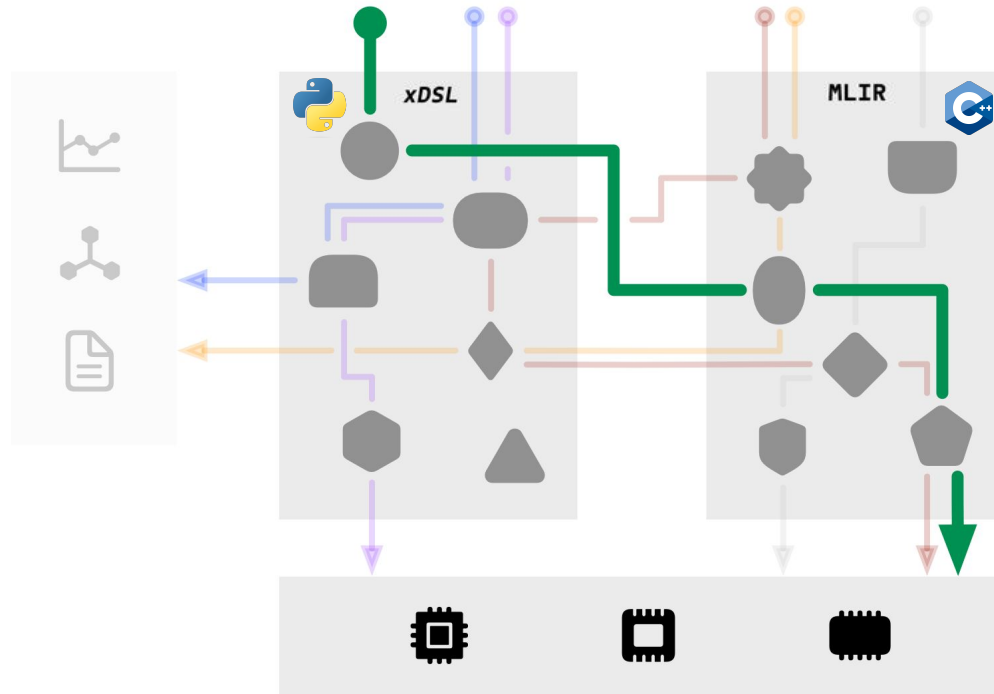
# xDSL: A Python framework replicating MLIR



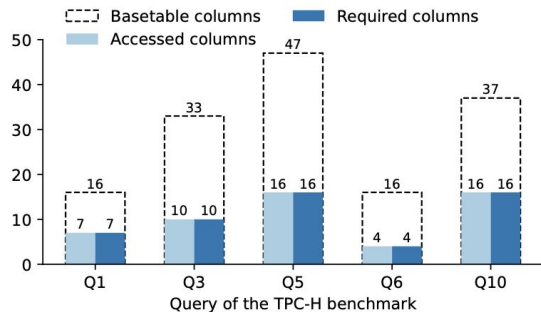
# xDSL: A Python *Sidekick* for MLIR



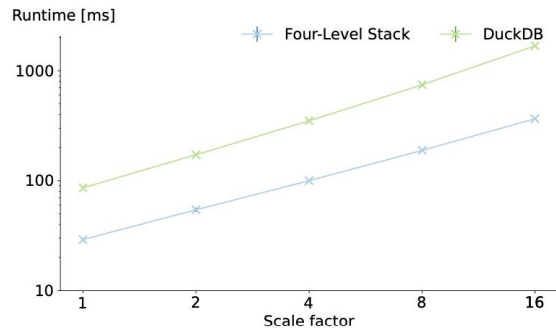
# Prototyping MLIR dialects in xDSL



# Prototyping MLIR dialects in xDSL



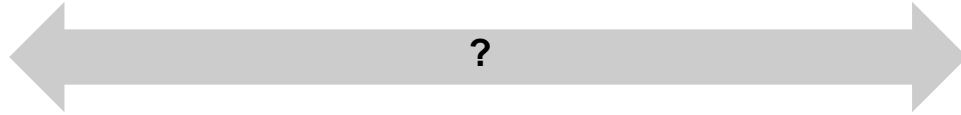
xDSL pass reducing accessed columns



Improvement over database compiler DuckDB



# How do we connect with MLIR?



# How do we connect with MLIR?



```
%0 = arith.constant 42 : i32  
%1 = arith.constant 12 : i32  
%2 = arith.addi %0, %1 : i32
```



# How do we connect with MLIR?



```
%0 = arith.constant 42 : i32  
%1 = arith.constant 12 : i32  
%2 = arith.addi %0, %1 : i32
```

What is arith?



# How do we connect with MLIR? (WIP)



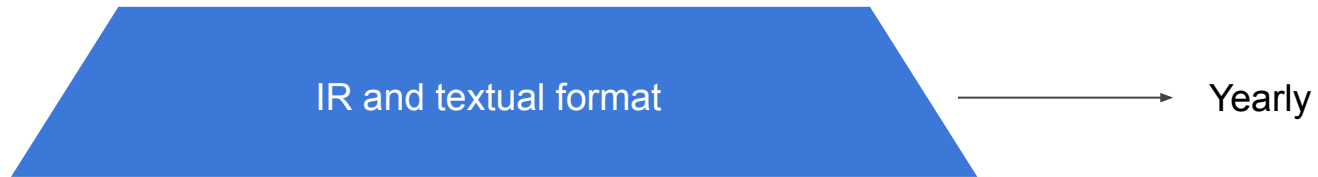
```
irdl.operation @arith.addi {  
  %0 = irdl.is_type : f32  
  %1 = irdl.is_type : f64  
  %2 = irdl.any_of(%0, %1)  
  irdl.operands(%2, %2)  
  irdl.results(%2)  
}
```

# Why not Python Bindings?

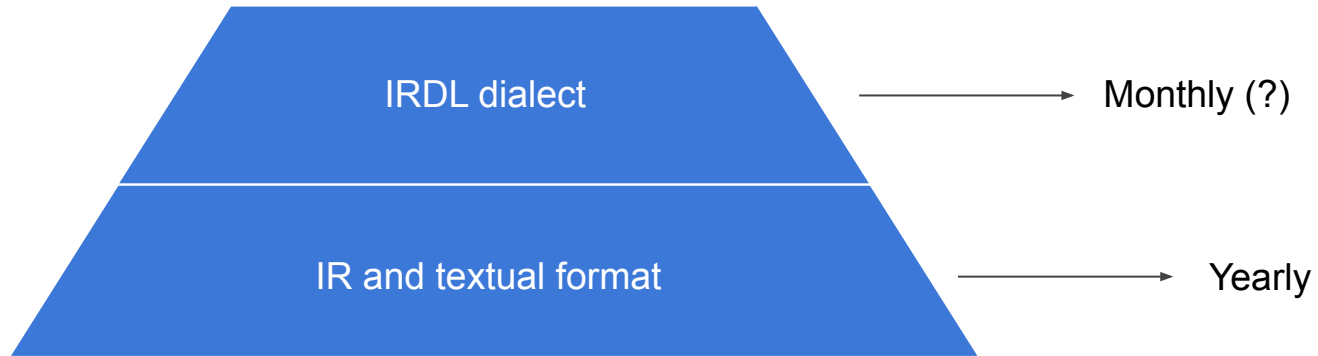


# The Pyramid of MLIR Compatibility

# The Pyramid of MLIR Compatibility

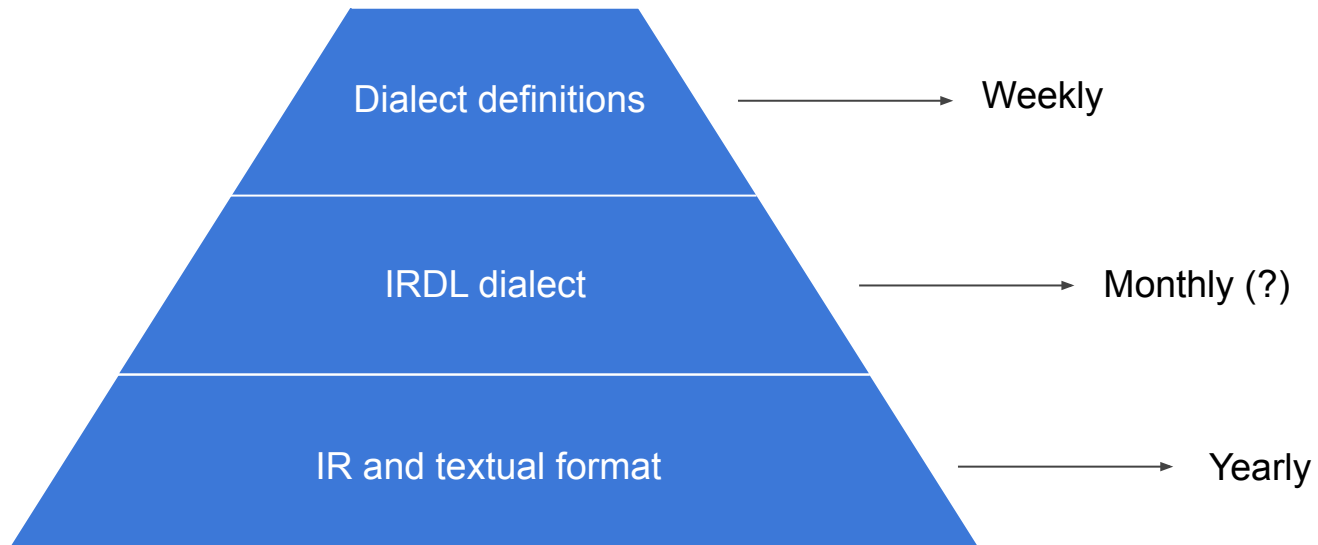


# The Pyramid of MLIR Compatibility

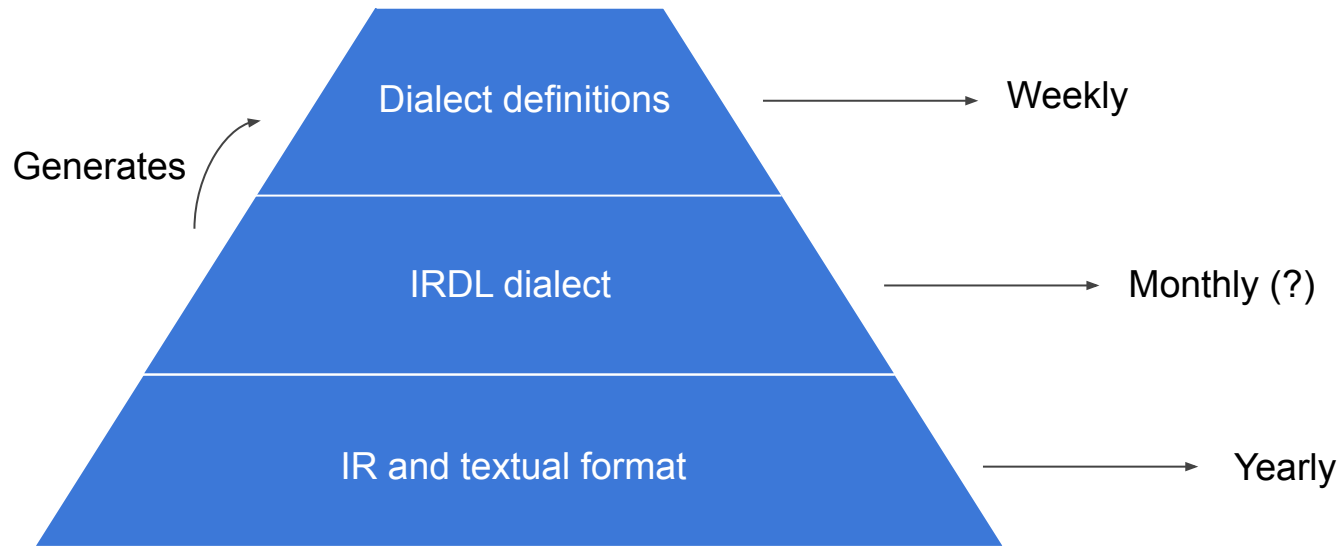




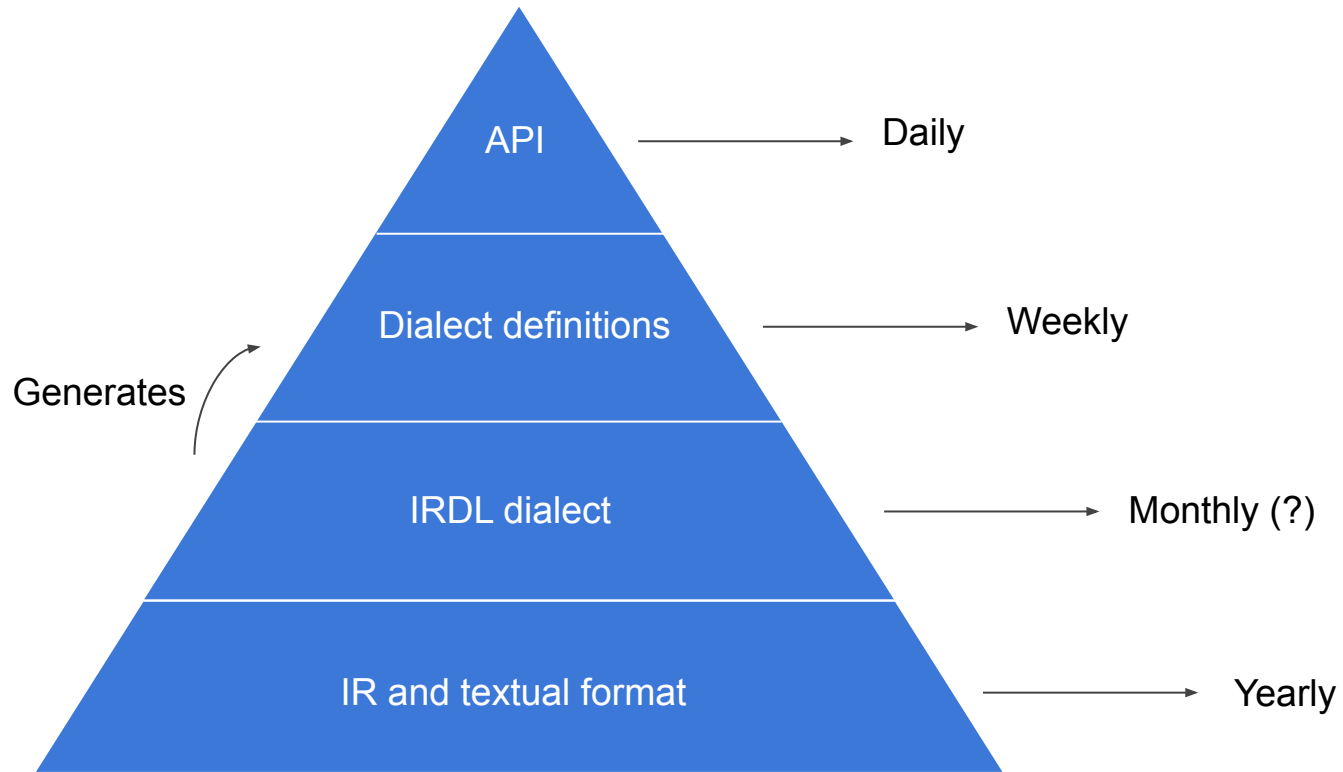
# The Pyramid of MLIR Compatibility



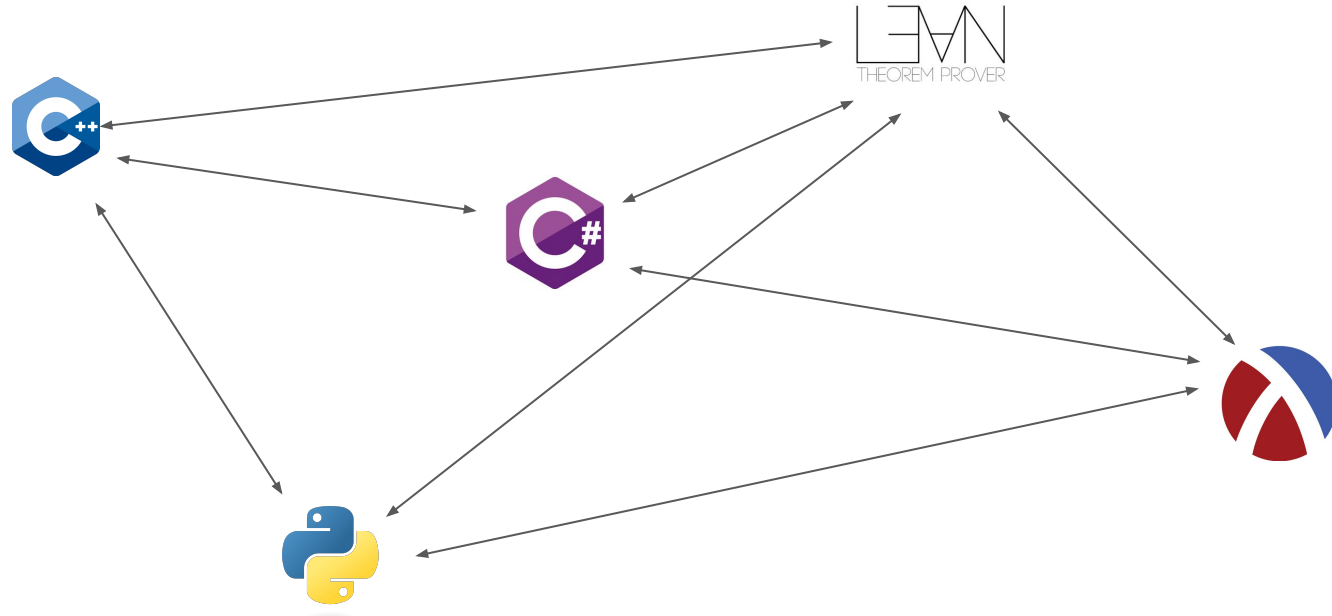
# The Pyramid of MLIR Compatibility



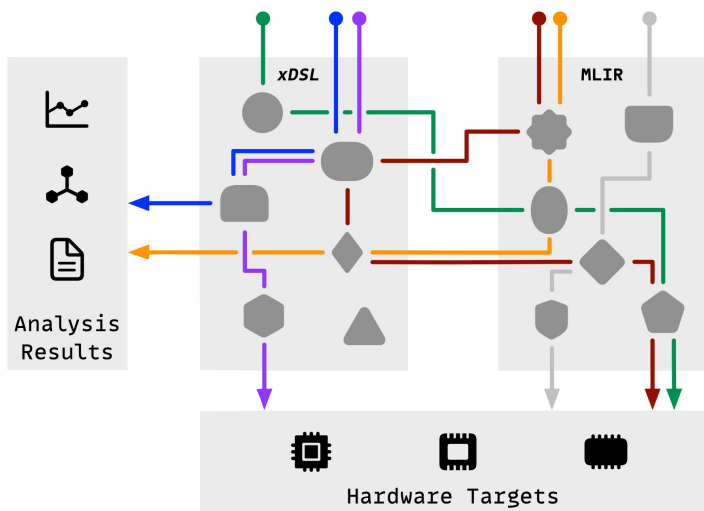
# The Pyramid of MLIR Compatibility



# Other *Sidekick* Frameworks?



# xDSL: A Compiler Infrastructure for Python DSLs



[xdsl.dev](https://xdsl.dev)

 [github.com/xdslproject/xdsl/](https://github.com/xdslproject/xdsl/)

Backup slides

# xDSL has reasonable overheads compared to MLIR

