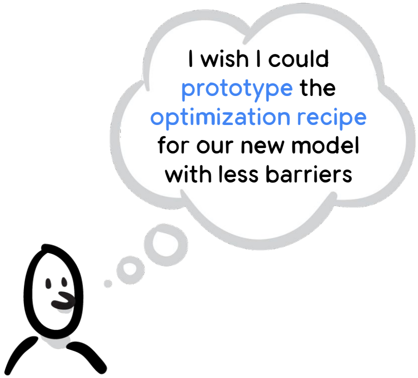
Driving MLIR Compilation from Python

Martin Lücke
Alex Zinenko
Ingo Müller
Matthias Springer

Whom is this for?

ML Researcher

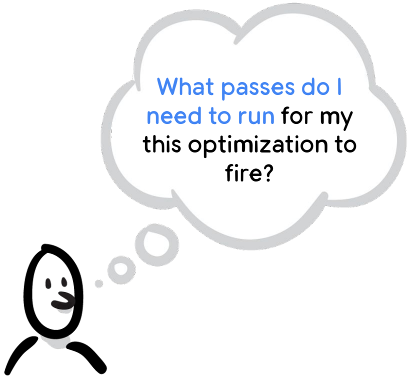
- Comfortable in Python
- Interested in (some) low level details


I wish I could prototype the optimization recipe for our new model with less barriers

Performance Engineer

Designs heuristics, e.g:

- When to fuse ops
- What tile size for this matmul?


What passes do I need to run for my this optimization to fire?

Compiler Engineer

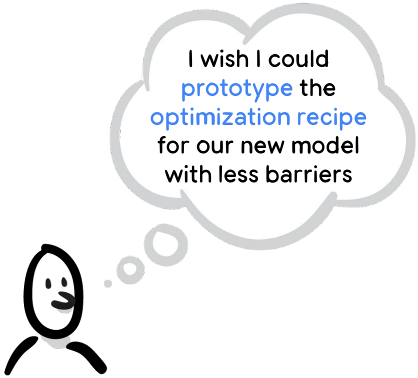
- Writes new optimizations
- Cares deeply about low level details

Whom is this for?

I want to try an idea how to tile my new custom op

ML Researcher

- Comfortable in Python
- Interested in (some) low level details

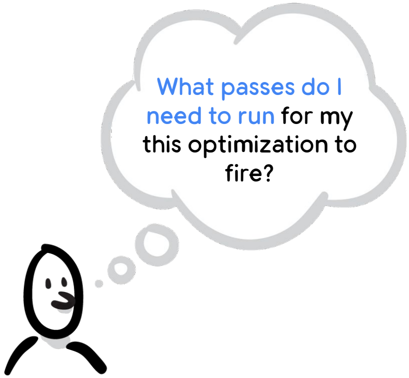


I wish I could prototype the optimization recipe for our new model with less barriers

Performance Engineer

Designs heuristics, e.g:

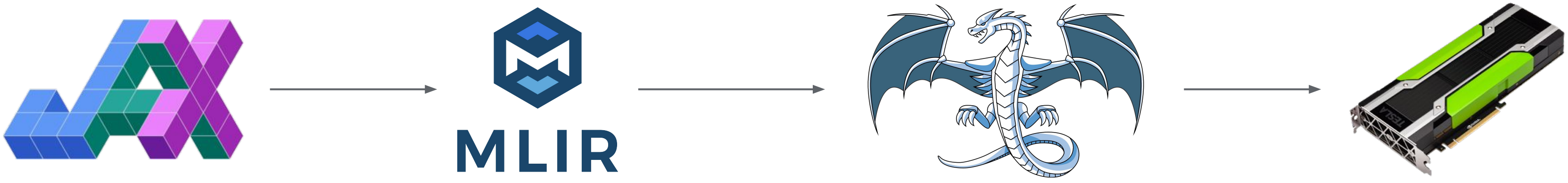
- When to fuse ops
- What tile size for this matmul?



What passes do I need to run for my this optimization to fire?

Compiler Engineer

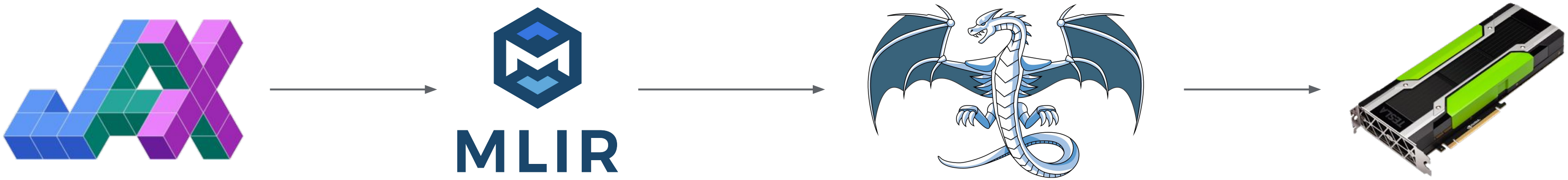
- Writes new optimizations
- Cares deeply about low level details



Compilation Flow: Batch Matmul

.py

```python
import jax

def batch_matmul (a: jax.Array[128, 80, 32],
                  b: jax.Array[128, 32, 320]) ->
                     jax.Array[128, 80, 320]:
  return jax.batch_matmul (a, b)
```



Compilation Flow: Batch Matmul

.py

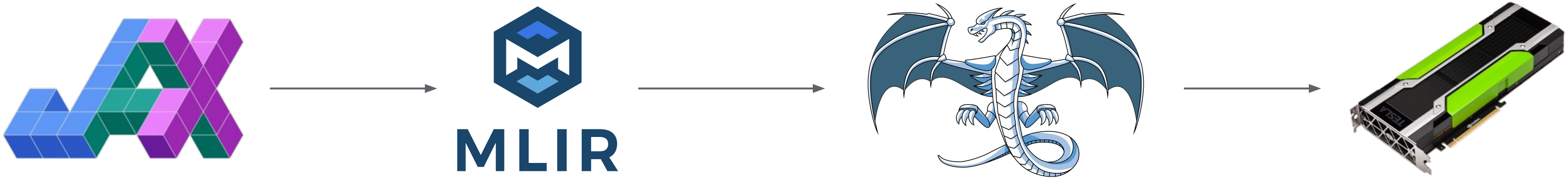
```python
import jax

def batch_matmul (a: jax.Array[128, 80, 32],
                  b: jax.Array[128, 32, 320]) ->
                     jax.Array[128, 80, 320]:
  return jax.batch_matmul (a, b)
```

.mlir

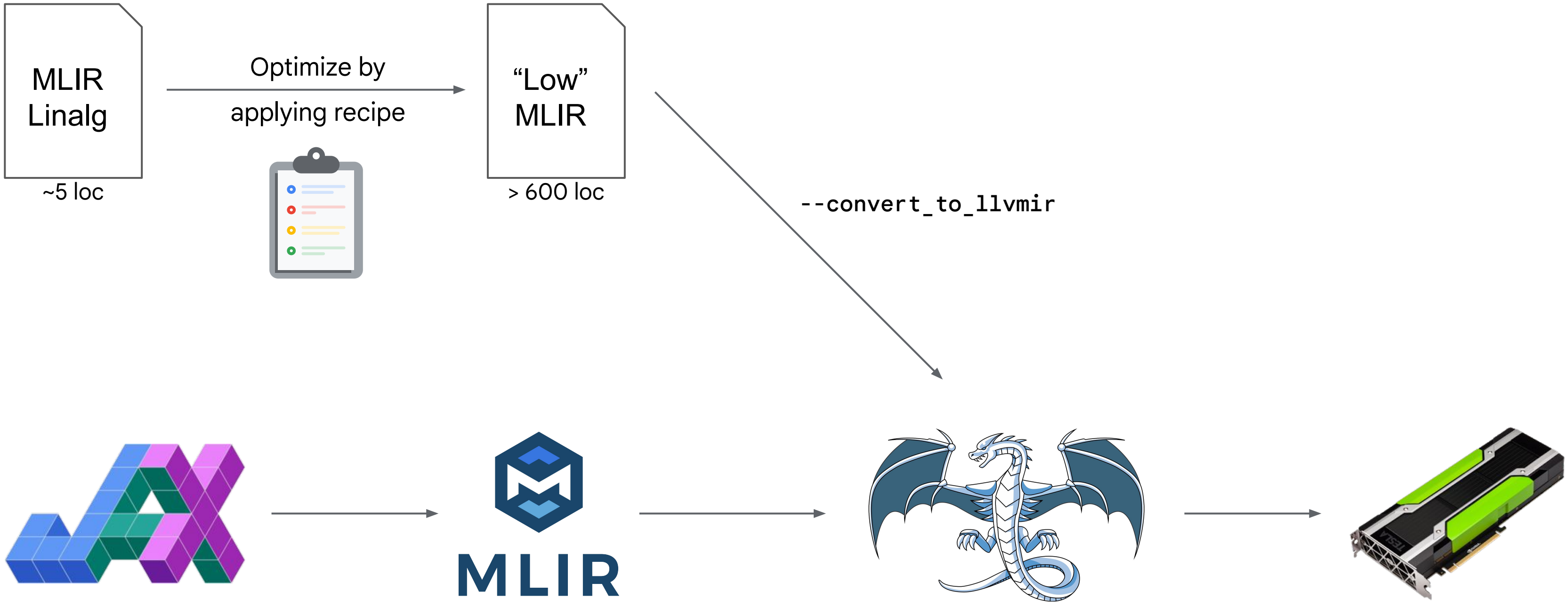
```mlir
func.func public @batch_matmul(%arg0: tensor<128x80x32xf32>,
                               %arg1: tensor<128x32x320xf32>) ->
                                     (tensor<128x80x320xf32>) {
    // prepare output
    %0   = tensor.empty() : tensor<128x80x320xf32>
    %cst = arith.constant 0.0 : f32
    %1   = linalg.fill ins(%cst) outs(%0)
    %2   = linalg.batch_matmul ins(%arg0, %arg1) outs(%1)
    return %2 : tensor<128x80x320xf32>
}
```

--convert_to_stable_hlo

--convert_to_linalg



Compilation Flow: Batch Matmul



MLIR Linalg

~5 loc

Optimize by applying recipe

"Low" MLIR

> 600 loc

`--convert_to_llvmir`

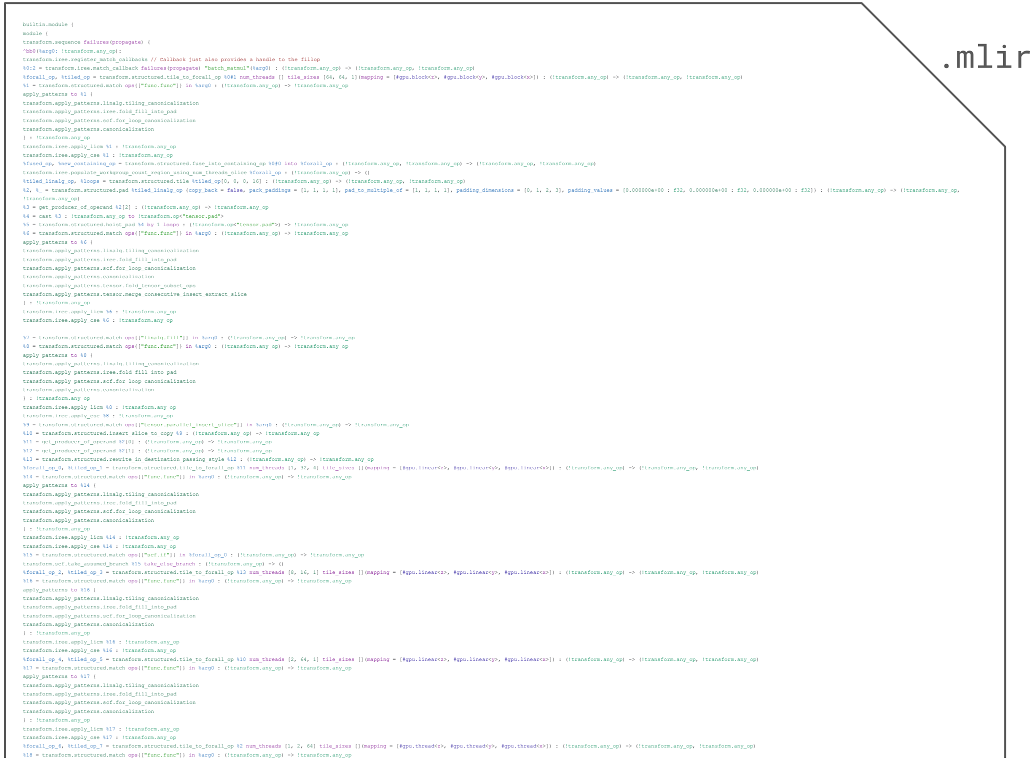
Expressing Optimization Recipe

- Available to expert users only
- Construction is error prone
- Hard to prototype new recipes

Level of abstraction too low for use cases

> 200 loc!

Transform Dialect



```
builtin.module {
module {
transform.sequence failures(propagate) {
^bb0(%arg0: !transform.any_op):
transform.iree.register_match_callbacks // Callback just also provides a handle to the fillop
%0:2 = transform.iree.match_callback failures(propagate) "batch_matmul"(%arg0) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
%forall_op, %tiled_op = transform.structured.tile_to_forall_op %0#1 num_threads [] tile_sizes [64, 64, 1](mapping = [#gpu.block<z>, #gpu.block<y>, #gpu.block<x>]) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
%1 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
apply_patterns to %1 {
transform.apply_patterns.linalg.tiling_canonicalization
transform.apply_patterns.iree.fold_fill_into_pad
transform.apply_patterns.scf.for_loop_canonicalization
transform.apply_patterns.canonicalization
} : !transform.any_op
transform.iree.apply_licm %1 : !transform.any_op
transform.iree.apply_cse %1 : !transform.any_op
%fused_op, %new_containing_op = transform.structured.fuse_into_containing_op %0#0 into %forall_op : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
transform.iree.populate_workgroup_count_region_using_num_threads_slice %forall_op : (!transform.any_op) -> ()
%tiled_linalg_op, %loops = transform.structured.tile %tiled_op[0, 0, 0, 16] : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
%2, %_ = transform.structured.pad %tiled_linalg_op {copy_back = false, pack_paddings = [1, 1, 1, 1], pad_to_multiple_of = [1, 1, 1, 1], padding_dimensions = [0, 1, 2, 3], padding_values = [0.000000e+00 : f32, 0.000000e+00 : f32, 0.000000e+00 : f32]} : (!transform.any_op) -> (!transform.any_op,
!transform.any_op)
%3 = get_producer_of_operand %2[2] : (!transform.any_op) -> !transform.any_op
%4 = cast %3 : !transform.any_op to !transform.op<"tensor.pad">
%5 = transform.structured.hoist_pad %4 by 1 loops : (!transform.op<"tensor.pad">) -> !transform.any_op
%6 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
apply_patterns to %6 {
transform.apply_patterns.linalg.tiling_canonicalization
transform.apply_patterns.iree.fold_fill_into_pad
transform.apply_patterns.scf.for_loop_canonicalization
transform.apply_patterns.canonicalization
transform.apply_patterns.tensor.fold_tensor_subset_ops
transform.apply_patterns.tensor.merge_consecutive_insert_extract_slice
} : !transform.any_op
transform.iree.apply_licm %6 : !transform.any_op
transform.iree.apply_cse %6 : !transform.any_op

%7 = transform.structured.match ops{["linalg.fill"]} in %arg0 : (!transform.any_op) -> !transform.any_op
%8 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
apply_patterns to %8 {
transform.apply_patterns.linalg.tiling_canonicalization
transform.apply_patterns.iree.fold_fill_into_pad
transform.apply_patterns.scf.for_loop_canonicalization
transform.apply_patterns.canonicalization
} : !transform.any_op
transform.iree.apply_licm %8 : !transform.any_op
transform.iree.apply_cse %8 : !transform.any_op
%9 = transform.structured.match ops{["tensor.parallel_insert_slice"]} in %arg0 : (!transform.any_op) -> !transform.any_op
%10 = transform.structured.insert_slice_to_copy %9 : (!transform.any_op) -> !transform.any_op
%11 = get_producer_of_operand %2[0] : (!transform.any_op) -> !transform.any_op
%12 = get_producer_of_operand %2[1] : (!transform.any_op) -> !transform.any_op
%13 = transform.structured.rewrite_in_destination_passing_style %12 : (!transform.any_op) -> !transform.any_op
%forall_op_0, %tiled_op_1 = transform.structured.tile_to_forall_op %11 num_threads [1, 32, 4] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
%14 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
apply_patterns to %14 {
transform.apply_patterns.linalg.tiling_canonicalization
transform.apply_patterns.iree.fold_fill_into_pad
transform.apply_patterns.scf.for_loop_canonicalization
transform.apply_patterns.canonicalization
} : !transform.any_op
transform.iree.apply_licm %14 : !transform.any_op
transform.iree.apply_cse %14 : !transform.any_op
%15 = transform.structured.match ops{["scf.if"]} in %forall_op_0 : (!transform.any_op) -> !transform.any_op
transform.scf.take_assumed_branch %15 take_else_branch : (!transform.any_op) -> ()
%forall_op_2, %tiled_op_3 = transform.structured.tile_to_forall_op %13 num_threads [8, 16, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
%16 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
apply_patterns to %16 {
transform.apply_patterns.linalg.tiling_canonicalization
transform.apply_patterns.iree.fold_fill_into_pad
transform.apply_patterns.scf.for_loop_canonicalization
transform.apply_patterns.canonicalization
} : !transform.any_op
transform.iree.apply_licm %16 : !transform.any_op
transform.iree.apply_cse %16 : !transform.any_op
%forall_op_4, %tiled_op_5 = transform.structured.tile_to_forall_op %10 num_threads [2, 64, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
%17 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
apply_patterns to %17 {
transform.apply_patterns.linalg.tiling_canonicalization
transform.apply_patterns.iree.fold_fill_into_pad
transform.apply_patterns.scf.for_loop_canonicalization
transform.apply_patterns.canonicalization
} : !transform.any_op
transform.iree.apply_licm %17 : !transform.any_op
transform.iree.apply_cse %17 : !transform.any_op
%forall_op_6, %tiled_op_7 = transform.structured.tile_to_forall_op %2 num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
%18 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
```

// [...] Less than half shown here

Python transforms

.py

```python
from mlir.dialects import linalg
import jax

def batch_matmul(a: jax.Array[128, 80, 32],
                 b: jax.Array[128, 32, 320]) ->
                   jax.Array[128, 80, 320]:
  return jax.batch_matmul(a, b)

def schedule(module: OpHandle) -> None:
  matmul   = module.match_ops(linalg.BatchMatmulOp)
  fill     = module.match_ops(linalg.FillOp)
  for_all  = matmul.tile_to_forall(tile_sizes=[64, 64, 1])
  fill.fuse_into(for_all)
  for_all2 = matmul.tile_to_forall(tile_sizes=[4, 32, 1])
  # ...

jit(batch_matmul, schedule, input)
```

.mlir

```mlir
func.func public @batch_matmul(%arg0: tensor<128x80x32xf32>,
                               %arg1: tensor<128x32x320xf32>)->
                              (tensor<128x80x320xf32>) {
    // prepare output
    %0   = tensor.empty() : tensor<128x80x320xf32>
    %cst = arith.constant 0.0 : f32
    %1   = linalg.fill ins(%cst) outs(%0)
    %2   = linalg.batch_matmul ins(%arg0, %arg1) outs(%1)
    return %2 : tensor<128x80x320xf32>
}
```

Python transforms

.py

```python
def schedule(module: OpHandle) -> None:
  matmul   = module.match_ops(linalg.BatchMatmulOp)
  fill     = module.match_ops(linalg.FillOp)
  for_all  = matmul.tile_to_forall(tile_sizes=[64, 64, 1])
  fill.fuse_into(for_all)
  for_all2 = matmul.tile_to_forall(tile_sizes=[4, 32, 1])
  # ...
```

.mlir

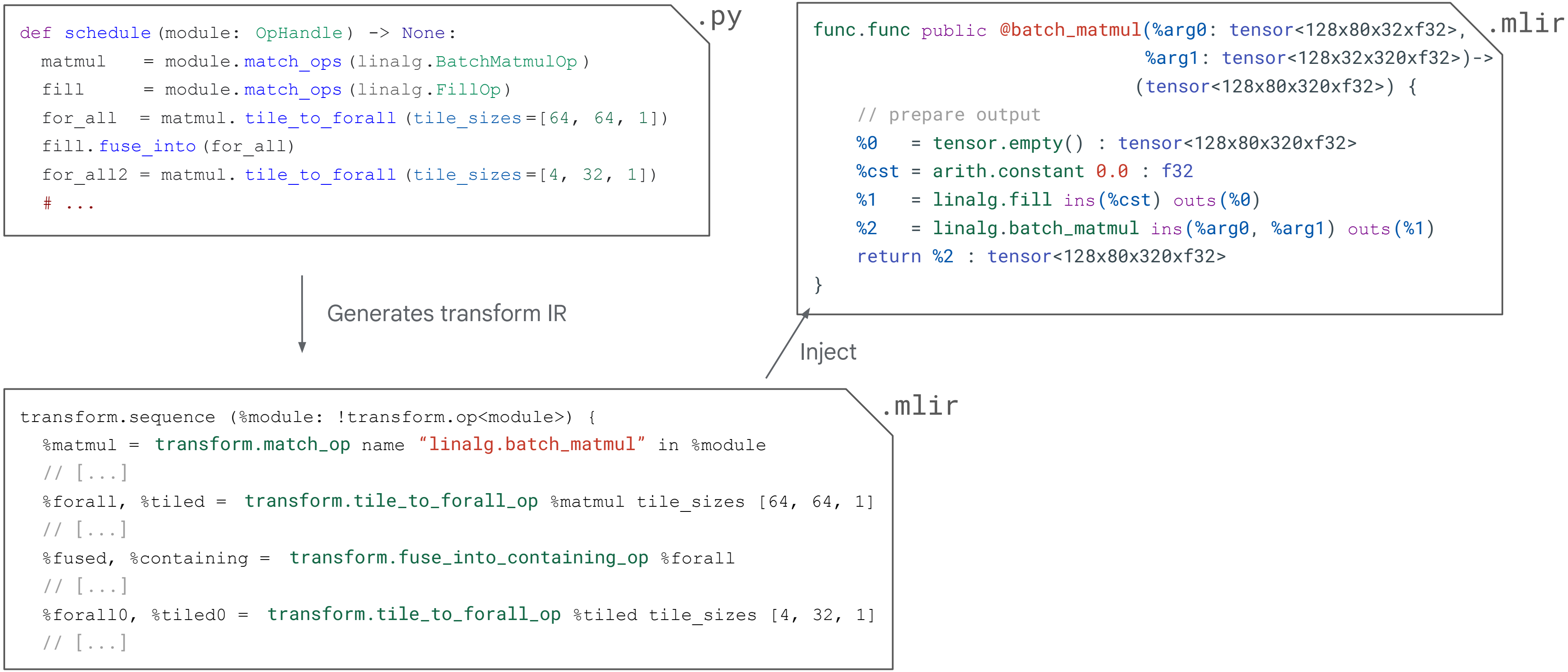
```mlir
func.func public @batch_matmul(%arg0: tensor<128x80x32xf32>,
                               %arg1: tensor<128x32x320xf32>)->
                              (tensor<128x80x320xf32>) {
    // prepare output
    %0   = tensor.empty() : tensor<128x80x320xf32>
    %cst = arith.constant 0.0 : f32
    %1   = linalg.fill ins(%cst) outs(%0)
    %2   = linalg.batch_matmul ins(%arg0, %arg1) outs(%1)
    return %2 : tensor<128x80x320xf32>
}
```

Generates transform IR

.mlir

```mlir
transform.sequence (%module: !transform.op<module>) {
  %matmul = transform.match_op name “linalg.batch_matmul” in %module
  // [...]
  %forall, %tiled = transform.tile_to_forall_op %matmul tile_sizes [64, 64, 1]
  // [...]
  %fused, %containing = transform.fuse_into_containing_op %forall
  // [...]
  %forall0, %tiled0 = transform.tile_to_forall_op %tiled tile_sizes [4, 32, 1]
  // [...]
```

Python transforms



.py

```python
def schedule(module: OpHandle) -> None:
  matmul   = module.match_ops(linalg.BatchMatmulOp)
  fill     = module.match_ops(linalg.FillOp)
  for_all  = matmul.tile_to_forall(tile_sizes=[64, 64, 1])
  fill.fuse_into(for_all)
  for_all2 = matmul.tile_to_forall(tile_sizes=[4, 32, 1])
  # ...
```

Generates transform IR

.mlir

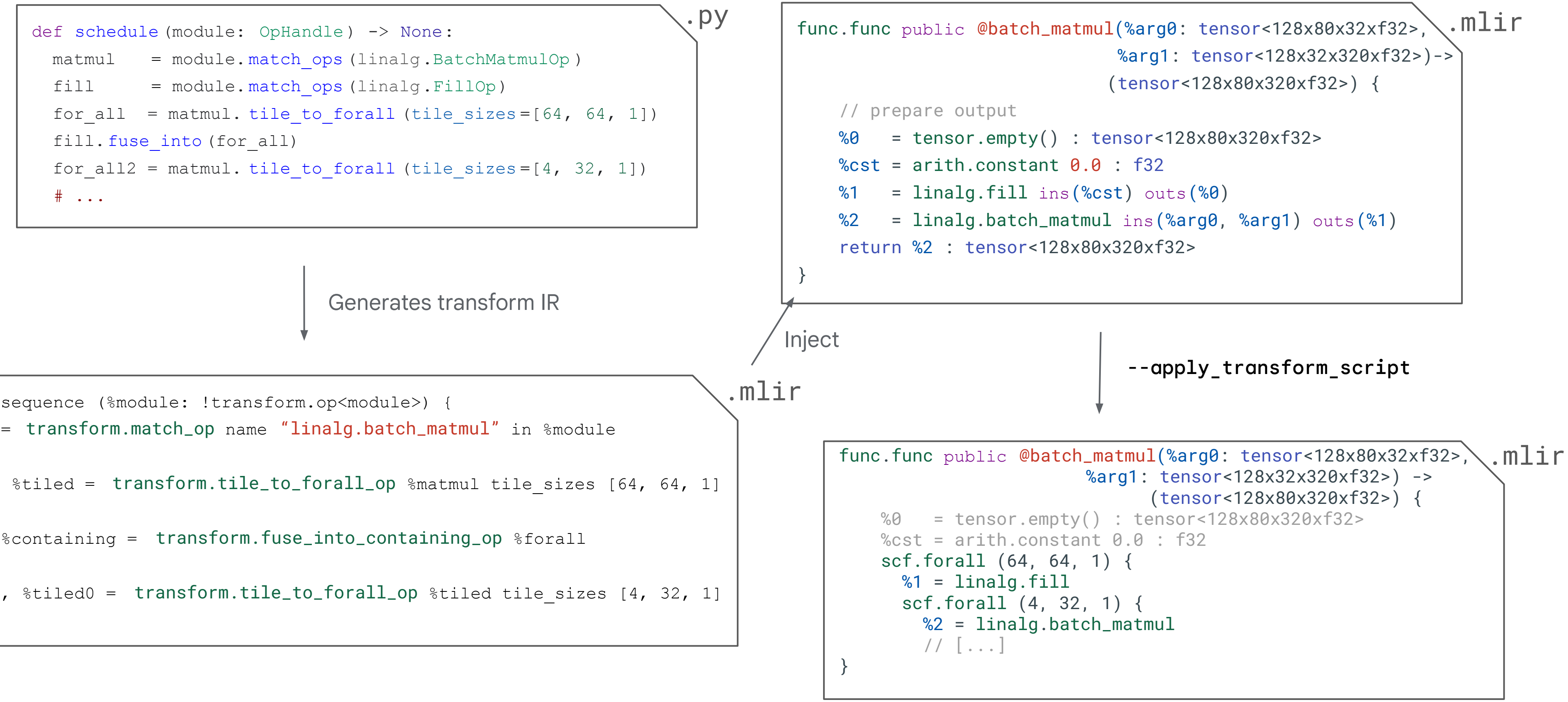
```
transform.sequence (%module: !transform.op<module>) {
  %matmul = transform.match_op name "linalg.batch_matmul" in %module
  // [...]
  %forall, %tiled = transform.tile_to_forall_op %matmul tile_sizes [64, 64, 1]
  // [...]
  %fused, %containing = transform.fuse_into_containing_op %forall
  // [...]
  %forall0, %tiled0 = transform.tile_to_forall_op %tiled tile_sizes [4, 32, 1]
  // [...]
```

Inject

.mlir

```
func.func public @batch_matmul(%arg0: tensor<128x80x32xf32>,
                               %arg1: tensor<128x32x320xf32>)->
                              (tensor<128x80x320xf32>) {
    // prepare output
    %0   = tensor.empty() : tensor<128x80x320xf32>
    %cst = arith.constant 0.0 : f32
    %1   = linalg.fill ins(%cst) outs(%0)
    %2   = linalg.batch_matmul ins(%arg0, %arg1) outs(%1)
    return %2 : tensor<128x80x320xf32>
}
```

Python transforms



.py

```
def schedule(module: OpHandle) -> None:
  matmul   = module.match_ops(linalg.BatchMatmulOp)
  fill     = module.match_ops(linalg.FillOp)
  for_all  = matmul.tile_to_forall(tile_sizes=[64, 64, 1])
  fill.fuse_into(for_all)
  for_all2 = matmul.tile_to_forall(tile_sizes=[4, 32, 1])
  # ...
```

Generates transform IR

.mlir

```
sequence (%module: !transform.op<module>) {
= transform.match_op name “linalg.batch_matmul” in %module

 %tiled = transform.tile_to_forall_op %matmul tile_sizes [64, 64, 1]

%containing = transform.fuse_into_containing_op %forall

, %tiled0 = transform.tile_to_forall_op %tiled tile_sizes [4, 32, 1]
```

Inject

.mlir

```
func.func public @batch_matmul(%arg0: tensor<128x80x32xf32>,
                               %arg1: tensor<128x32x320xf32>)->
                              (tensor<128x80x320xf32>) {
    // prepare output
    %0   = tensor.empty() : tensor<128x80x320xf32>
    %cst = arith.constant 0.0 : f32
    %1   = linalg.fill ins(%cst) outs(%0)
    %2   = linalg.batch_matmul ins(%arg0, %arg1) outs(%1)
    return %2 : tensor<128x80x320xf32>
}
```

--apply_transform_script

.mlir

```
func.func public @batch_matmul(%arg0: tensor<128x80x32xf32>,
                               %arg1: tensor<128x32x320xf32>) ->
                                  (tensor<128x80x320xf32>) {
    %0   = tensor.empty() : tensor<128x80x320xf32>
    %cst = arith.constant 0.0 : f32
    scf.forall (64, 64, 1) {
      %1 = linalg.fill
      scf.forall (4, 32, 1) {
        %2 = linalg.batch_matmul
        // [...]
}
```

```
builtin.module {
  module {
    transform.sequence  failures(propagate) {
    ^bb0(%arg0: !transform.any_op):
      transform.iree.register_match_callbacks // Callback just also provides a handle to the fillop
      %0:2 = transform.iree.match_callback failures(propagate) "batch_matmul"(%arg0) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %forall_op, %tiled_op = transform.structured.tile_to_forall_op %0#1   num_threads [] tile_sizes [64, 64, 1](mapping = [#gpu.block<z>, #gpu.block<y>, #gpu.block<x>]) : (!transform.any_op) ->
(!transform.any_op, !transform.any_op)
      %1 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %1 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %1 : !transform.any_op
      transform.iree.apply_cse %1 : !transform.any_op
      %fused_op, %new_containing_op = transform.structured.fuse_into_containing_op %0#0 into %forall_op : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
      transform.iree.populate_workgroup_count_region_using_num_threads_slice %forall_op : (!transform.any_op) -> ()
      %tiled_linalg_op, %loops = transform.structured.tile %tiled_op[0, 0, 0, 16] : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %2, %_ = transform.structured.pad %tiled_linalg_op {copy_back = false, pack_paddings = [1, 1, 1, 1], pad_to_multiple_of = [1, 1, 1, 1], padding_dimensions = [0, 1, 2, 3], padding_values =
[0.000000e+00 : f32, 0.000000e+00 : f32, 0.000000e+00 : f32]} : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %3 = get_producer_of_operand %2[2] : (!transform.any_op) -> !transform.any_op
      %4 = cast %3 : !transform.any_op to !transform.op<"tensor.pad">
      %5 = transform.structured.hoist_pad %4 by 1 loops : (!transform.op<"tensor.pad">) -> !transform.any_op
      %6 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %6 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
        transform.apply_patterns.tensor.fold_tensor_subset_ops
        transform.apply_patterns.tensor.merge_consecutive_insert_extract_slice
      } : !transform.any_op
      transform.iree.apply_licm %6 : !transform.any_op
      transform.iree.apply_cse %6 : !transform.any_op

      %7 = transform.structured.match ops{["linalg.fill"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      %8 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %8 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %8 : !transform.any_op
      transform.iree.apply_cse %8 : !transform.any_op
      %9 = transform.structured.match ops{["tensor.parallel_insert_slice"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      %10 = transform.structured.insert_slice_to_copy %9 : (!transform.any_op) -> !transform.any_op
      %11 = get_producer_of_operand %2[0] : (!transform.any_op) -> !transform.any_op
      %12 = get_producer_of_operand %2[1] : (!transform.any_op) -> !transform.any_op
      %13 = transform.structured.rewrite_in_destination_passing_style %12 : (!transform.any_op) -> !transform.any_op
      %forall_op_0, %tiled_op_1 = transform.structured.tile_to_forall_op %11   num_threads [1, 32, 4] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %14 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %14 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %14 : !transform.any_op
      transform.iree.apply_cse %14 : !transform.any_op
      %15 = transform.structured.match ops{["scf.if"]} in %forall_op_0 : (!transform.any_op) -> !transform.any_op
      transform.scf.take_assumed_branch %15 take_else_branch : (!transform.any_op) -> ()
      %forall_op_2, %tiled_op_3 = transform.structured.tile_to_forall_op %13   num_threads [8, 16, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %16 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %16 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %16 : !transform.any_op
      transform.iree.apply_cse %16 : !transform.any_op
      %forall_op_4, %tiled_op_5 = transform.structured.tile_to_forall_op %10   num_threads [2, 64, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %17 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %17 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %17 : !transform.any_op
      transform.iree.apply_cse %17 : !transform.any_op
      %forall_op_6, %tiled_op_7 = transform.structured.tile_to_forall_op %2   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %18 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %18 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %18 : !transform.any_op
      transform.iree.apply_cse %18 : !transform.any_op
      %forall_op_8, %tiled_op_9 = transform.structured.tile_to_forall_op %7   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %19 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %19 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %19 : !transform.any_op
      transform.iree.apply_cse %19 : !transform.any_op
```

```
 %20 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %20 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %20 : !transform.any_op
      transform.iree.apply_cse %20 : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_1 vector_sizes [64, 2, 4] : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_5 vector_sizes [32, 1, 1] : !transform.any_op
      %21 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %21 {
        transform.apply_patterns.vector.lower_masked_transfers
      } : !transform.any_op
      %22 = transform.structured.vectorize %21 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %22 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %22 : !transform.any_op
      transform.iree.apply_cse %22 : !transform.any_op
      %23 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %23 {
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %23 : !transform.any_op
      transform.iree.apply_cse %23 : !transform.any_op
      transform.iree.eliminate_empty_tensors %arg0 : (!transform.any_op) -> ()
      %24 = transform.iree.bufferize {target_gpu} %arg0 : (!transform.any_op) -> !transform.any_op

      %25 = transform.structured.match ops{["func.func"]} in %24 : (!transform.any_op) -> !transform.any_op
      transform.iree.apply_buffer_optimizations %25 : (!transform.any_op) -> () // NO effect here
      %26 = transform.structured.match ops{["func.func"]} in %24 : (!transform.any_op) -> !transform.any_op
      transform.iree.forall_to_workgroup %26 : (!transform.any_op) -> ()
      transform.iree.map_nested_forall_to_gpu_threads %26 workgroup_dims = [64, 2, 1] warp_dims = [2, 2, 1] : (!transform.any_op) -> ()
      %27 = transform.iree.eliminate_gpu_barriers %26 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %27 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %27 : !transform.any_op
      transform.iree.apply_cse %27 : !transform.any_op
      transform.iree.hoist_static_alloc %27 : (!transform.any_op) -> ()
      apply_patterns to %27 {
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      apply_patterns to %27 {
        transform.apply_patterns.memref.extract_address_computations
      } : !transform.any_op
      apply_patterns to %27 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %27 : !transform.any_op
      transform.iree.apply_cse %27 : !transform.any_op
      %28 = transform.structured.match ops{["scf.for"]} in %27 : (!transform.any_op) -> !transform.op<"scf.for">
      transform.iree.synchronize_loop %28 : (!transform.op<"scf.for">) -> ()
      %29 = transform.structured.hoist_redundant_vector_transfers %27 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %29 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %29 : !transform.any_op
      transform.iree.apply_cse %29 : !transform.any_op
      transform.iree.apply_buffer_optimizations %29 : (!transform.any_op) -> ()
      %30 = transform.iree.eliminate_gpu_barriers %29 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      %31 = transform.structured.match ops{["memref.alloc"]} in %30 : (!transform.any_op) -> !transform.op<"memref.alloc">
      %32 = transform.memref.multibuffer %31 {factor = 2 : i64, skip_analysis} : (!transform.op<"memref.alloc">) -> !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.vector.transfer_to_scf   max_transfer_rank = 1 full_unroll = true
      } : !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      transform.iree.create_async_groups %30 : (!transform.any_op) -> ()
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      %33 = transform.structured.match ops{[" vector.contract"]} in %30 : (!transform.any_op) -> !transform.any_op
      %34 = transform.loop.get_parent_for %33 : (!transform.any_op) -> !transform.any_op
      %35 = transform.iree.pipeline_shared_memory_copies %34 {depth = 2 : i64} : (!transform.any_op) -> !transform.any_op
```

```
builtin.module {
  module {
    transform.sequence  failures(propagate) {
    ^bb0(%arg0: !transform.any_op):
      transform.iree.register_match_callbacks // Callback just also provides a handle to the fillop
      %0:2 = transform.iree.match_callback failures(propagate) "batch_matmul"(%arg0) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %forall_op, %tiled_op = transform.structured.tile_to_forall_op %0#1   num_threads [] tile_sizes [64, 64, 1](mapping = [#gpu.block<z>, #gpu.block<y>, #gpu.block<x>]) : (!transform.any_op) ->
(!transform.any_op, !transform.any_op)
      %1 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %1 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %1 : !transform.any_op
      transform.iree.apply_cse %1 : !transform.any_op
      %fused_op, %new_containing_op = transform.structured.fuse_into_containing_op %0#0 into %forall_op : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
      transform.iree.populate_workgroup_count_region_using_num_threads_slice %forall_op : (!transform.any_op) -> ()
      %tiled_linalg_op, %loops = transform.structured.tile %tiled_op[0, 0, 0, 16] : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %2, %_ = transform.structured.pad %tiled_linalg_op {copy_back = false, pack_paddings = [1, 1, 1, 1], pad_to_multiple_of = [1, 1, 1, 1], padding_dimensions = [0, 1, 2, 3], padding_values =
[0.000000e+00 : f32, 0.000000e+00 : f32, 0.000000e+00 : f32]} : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %3 = get_producer_of_operand %2[2] : (!transform.any_op) -> !transform.any_op
      %4 = cast %3 : !transform.any_op to !transform.op<"tensor.pad">
      %5 = transform.structured.hoist_pad %4 by 1 loops : (!transform.op<"tensor.pad">) -> !transform.any_op
      %6 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %6 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patte
        transform.apply_patte
        transform.apply_patte
        transform.apply_patte
        transform.apply_patte
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse

      %7 = transform.structure
      %8 = transform.structure
      apply_patterns to %8 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %9 = transform.structure
      %10 = transform.structur
      %11 = get_producer_of_ope
      %12 = get_producer_of_ope
      %13 = transform.structur
      %forall_op_0, %tiled_op_
-> (!transform.any_op, !transf
      %14 = transform.structur
      apply_patterns to %14 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %15 = transform.structur
      transform.scf.take_assum
      %forall_op_2, %tiled_op_
-> (!transform.any_op, !transf
      %16 = transform.structur
      apply_patterns to %16 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %forall_op_4, %tiled_op_
-> (!transform.any_op, !transf
      %17 = transform.structur
      apply_patterns to %17 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %forall_op_6, %tiled_op_
-> (!transform.any_op, !transf
      %18 = transform.structur
      apply_patterns to %18 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %18 : !transform.any_op
      transform.iree.apply_cse %18 : !transform.any_op
      %forall_op_8, %tiled_op_9 = transform.structured.tile_to_forall_op %7   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %19 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %19 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %19 : !transform.any_op
      transform.iree.apply_cse %19 : !transform.any_op
```

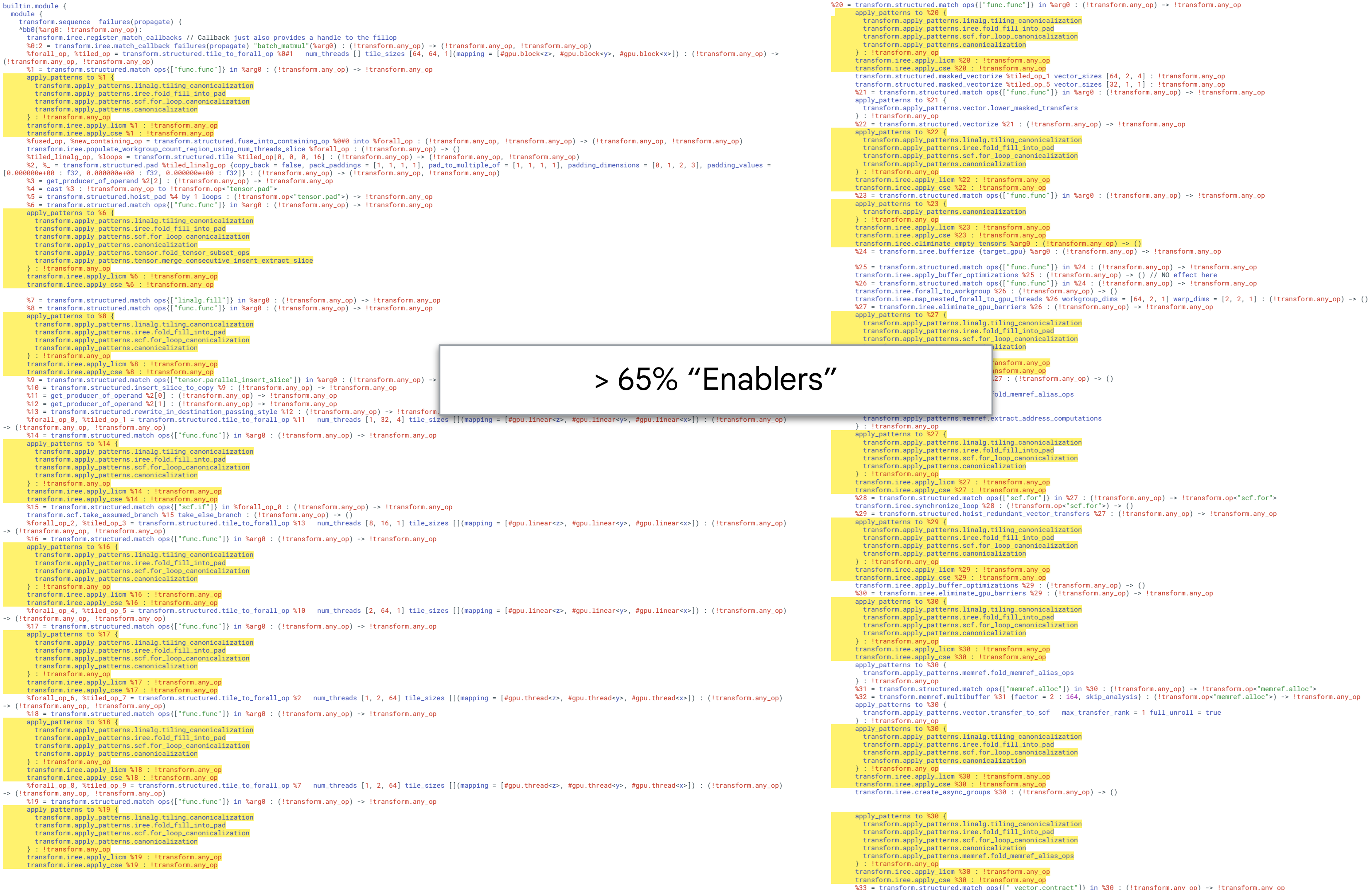
```
%20 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
     apply_patterns to %20 {
       transform.apply_patterns.linalg.tiling_canonicalization
       transform.apply_patterns.iree.fold_fill_into_pad
       transform.apply_patterns.scf.for_loop_canonicalization
       transform.apply_patterns.canonicalization
     } : !transform.any_op
     transform.iree.apply_licm %20 : !transform.any_op
     transform.iree.apply_cse %20 : !transform.any_op
     transform.structured.masked_vectorize %tiled_op_1 vector_sizes [64, 2, 4] : !transform.any_op
     transform.structured.masked_vectorize %tiled_op_5 vector_sizes [32, 1, 1] : !transform.any_op
     %21 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
     apply_patterns to %21 {
       transform.apply_patterns.vector.lower_masked_transfers
     } : !transform.any_op
     %22 = transform.structured.vectorize %21 : (!transform.any_op) -> !transform.any_op
     apply_patterns to %22 {
       transform.apply_patterns.linalg.tiling_canonicalization
       transform.apply_patterns.iree.fold_fill_into_pad
       transform.apply_patterns.scf.for_loop_canonicalization
       transform.apply_patterns.canonicalization
     } : !transform.any_op
     transform.iree.apply_licm %22 : !transform.any_op
     transform.iree.apply_cse %22 : !transform.any_op
     %23 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
     apply_patterns to %23 {
       transform.apply_patterns.canonicalization
     } : !transform.any_op

(!transform.any_op) -> ()

">

op

emref.alloc">
'>) -> !transform.any_op

       transform.apply_patterns.canonicalization
     } : !transform.any_op
     transform.iree.apply_licm %30 : !transform.any_op
     transform.iree.apply_cse %30 : !transform.any_op
     transform.iree.create_async_groups %30 : (!transform.any_op) -> ()
     apply_patterns to %30 {
       transform.apply_patterns.linalg.tiling_canonicalization
       transform.apply_patterns.iree.fold_fill_into_pad
       transform.apply_patterns.scf.for_loop_canonicalization
       transform.apply_patterns.canonicalization
       transform.apply_patterns.memref.fold_memref_alias_ops
     } : !transform.any_op
     transform.iree.apply_licm %30 : !transform.any_op
     transform.iree.apply_cse %30 : !transform.any_op
     %33 = transform.structured.match ops{[" vector.contract"]} in %30 : (!transform.any_op) -> !transform.any_op
     %34 = transform.loop.get_parent_for %33 : (!transform.any_op) -> !transform.any_op
     %35 = transform.iree.pipeline_shared_memory_copies %34 {depth = 2 : i64} : (!transform.any_op) -> !transform.any_op
```

```
%forall_op_0, %tiled_op_1 = transform.structured.tile_to_forall_op %11 num_threads [1, 32, 4]
%14 = transform.structured.match ops{["func.func"]} in %arg0
apply_patterns to %14 {
  transform.apply_patterns.linalg.tiling_canonicalization
  transform.apply_patterns.iree.fold_fill_into_pad
  transform.apply_patterns.scf.for_loop_canonicalization
  transform.apply_patterns.canonicalization
}
transform.iree.apply_licm %14
transform.iree.apply_cse %14
%15 = transform.structured.match ops{["scf.if"]} in %forall_op_0
transform.scf.take_assumed_branch %15 take_else_branch
%forall_op_2, %tiled_op_3 = transform.structured.tile_to_forall_op %13 num_threads [8, 16, 1]
%16 = transform.structured.match ops{["func.func"]} in %arg0
apply_patterns to %16 {
  transform.apply_patterns.linalg.tiling_canonicalization
  transform.apply_patterns.iree.fold_fill_into_pad
  transform.apply_patterns.scf.for_loop_canonicalization
  transform.apply_patterns.canonicalization
}
transform.iree.apply_licm %16
transform.iree.apply_cse %16
%forall_op_4, %tiled_op_5 = transform.structured.tile_to_forall_op %10 num_threads [2, 64, 1]
```

```
builtin.module {
  module {
    transform.sequence  failures(propagate) {
    ^bb0(%arg0: !transform.any_op):
      transform.iree.register_match_callbacks // Callback just also provides a handle to the fillop
      %0:2 = transform.iree.match_callback failures(propagate) "batch_matmul"(%arg0) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %forall_op, %tiled_op = transform.structured.tile_to_forall_op %0#1   num_threads [] tile_sizes [64, 64, 1](mapping = [#gpu.block<z>, #gpu.block<y>, #gpu.block<x>]) : (!transform.any_op) ->
(!transform.any_op, !transform.any_op)
      %1 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %1 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %1 : !transform.any_op
      transform.iree.apply_cse %1 : !transform.any_op
      %fused_op, %new_containing_op = transform.structured.fuse_into_containing_op %0#0 into %forall_op : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
      transform.iree.populate_workgroup_count_region_using_num_threads_slice %forall_op : (!transform.any_op) -> ()
      %tiled_linalg_op, %loops = transform.structured.tile %tiled_op[0, 0, 0, 16] : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %2, %_ = transform.structured.pad %tiled_linalg_op {copy_back = false, pack_paddings = [1, 1, 1, 1], pad_to_multiple_of = [1, 1, 1, 1], padding_dimensions = [0, 1, 2, 3], padding_values =
[0.000000e+00 : f32, 0.000000e+00 : f32, 0.000000e+00 : f32]} : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %3 = get_producer_of_operand %2[2] : (!transform.any_op) -> !transform.any_op
      %4 = cast %3 : !transform.any_op to !transform.op<"tensor.pad">
      %5 = transform.structured.hoist_pad %4 by 1 loops : (!transform.op<"tensor.pad">) -> !transform.any_op
      %6 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %6 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patter
        transform.apply_patt
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse

      %7 = transform.structured
      %8 = transform.structured
      apply_patterns to %8 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %9 = transform.structured
      %10 = transform.structure
      %11 = get_producer_of_ope
      %12 = get_producer_of_ope
      %13 = transform.structure
      %forall_op_0, %tiled_op_
-> (!transform.any_op, !transfo
      %14 = transform.structure
      apply_patterns to %14 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %15 = transform.structure
      transform.scf.take_assume
      %forall_op_2, %tiled_op_
-> (!transform.any_op, !transfo
      %16 = transform.structure
      apply_patterns to %16 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %forall_op_4, %tiled_op_
-> (!transform.any_op, !transfo
      %17 = transform.structure
      apply_patterns to %17 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse
      %forall_op_6, %tiled_op_
-> (!transform.any_op, !transfo
      %18 = transform.structure
      apply_patterns to %18 {
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
        transform.apply_patter
      } : !transform.any_op
      transform.iree.apply_lic
      transform.iree.apply_cse %18 : !transform.any_op
      %forall_op_8, %tiled_op_9 = transform.structured.tile_to_forall_op %7   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %19 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %19 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %19 : !transform.any_op
      transform.iree.apply_cse %19 : !transform.any_op
```

```
      %20 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %20 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %20 : !transform.any_op
      transform.iree.apply_cse %20 : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_1 vector_sizes [64, 2, 4] : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_5 vector_sizes [32, 1, 1] : !transform.any_op
      %21 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %21 {
        transform.apply_patterns.vector.lower_masked_transfers
      } : !transform.any_op
      %22 = transform.structured.vectorize %21 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %22 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %22 : !transform.any_op
      transform.iree.apply_cse %22 : !transform.any_op
      %23 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %23 {
        transform.apply_patterns.canonicalization
      } : !transform.any_op

 (!transform.any_op) -> ()

">
op

emref.alloc">
">) -> !transform.any_op

      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      transform.iree.create_async_groups %30 : (!transform.any_op) -> ()
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      %33 = transform.structured.match ops{[" vector.contract"]} in %30 : (!transform.any_op) -> !transform.any_op
      %34 = transform.loop.get_parent_for %33 : (!transform.any_op) -> !transform.any_op
      %35 = transform.iree.pipeline_shared_memory_copies %34 {depth = 2 : i64} : (!transform.any_op) -> !transform.any_op
```

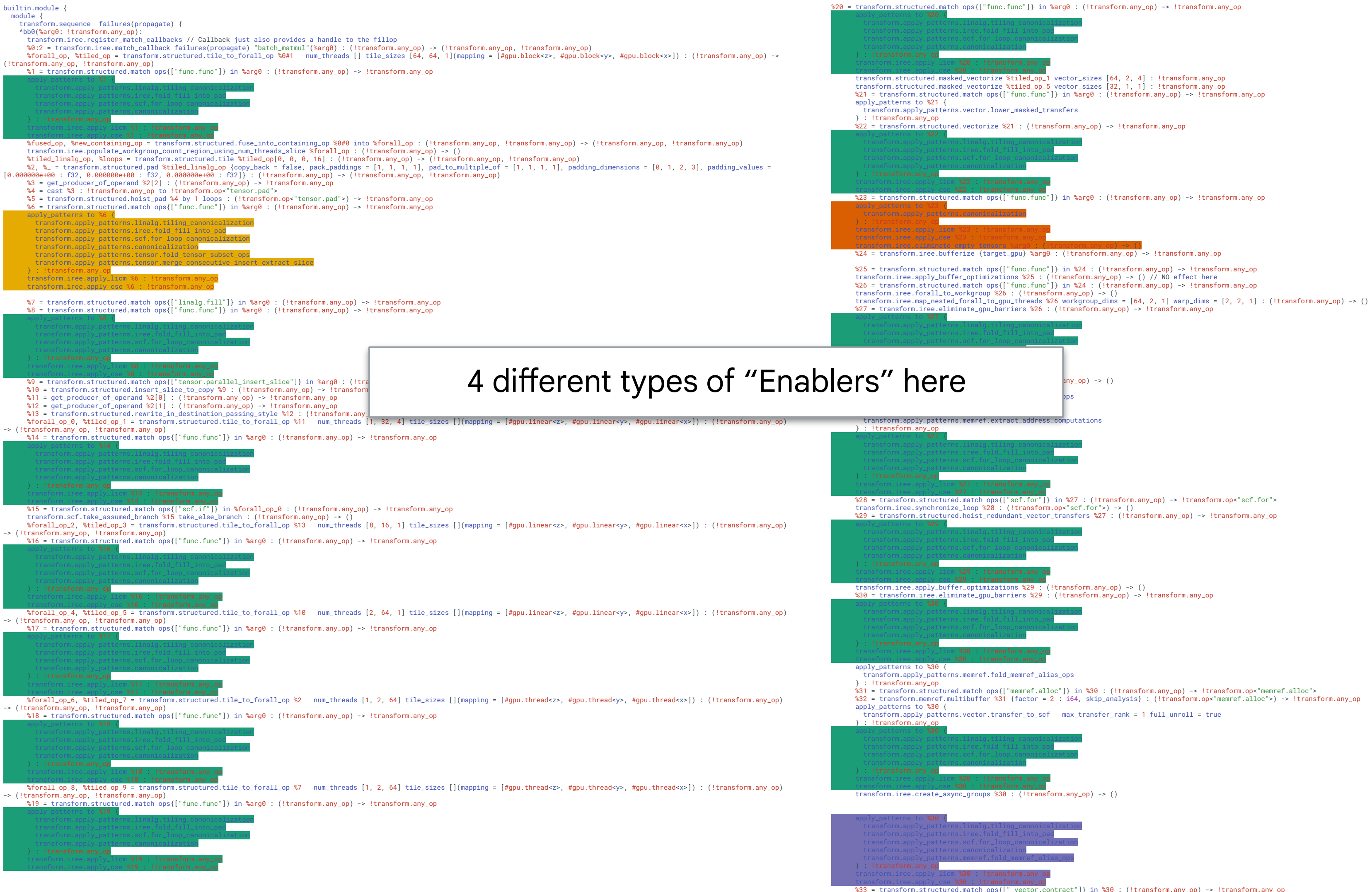
```
%forall_op_0, %tiled_op_1 = transform.structured.tile_to_forall_op %11 num_threads [1, 32, 4]
%14 = transform.structured.match ops{["func.func"]} in %arg0
apply_patterns to %14 {
  transform.apply_patterns.linalg.tiling_canonicalization
  transform.apply_patterns.iree.fold_fill_into_pad
  transform.apply_patterns.scf.for_loop_canonicalization
  transform.apply_patterns.canonicalization
}
transform.iree.apply_licm %14
transform.iree.apply_cse %14
%15 = transform.structured.match ops{["scf.if"]} in %forall_op_0
transform.scf.take_assumed_branch %15 take_else_branch
%forall_op_2, %tiled_op_3 = transform.structured.tile_to_forall_op %13 num_threads [8, 16, 1]
%16 = transform.structured.match ops{["func.func"]} in %arg0
apply_patterns to %16 {
  transform.apply_patterns.linalg.tiling_canonicalization
  transform.apply_patterns.iree.fold_fill_into_pad
  transform.apply_patterns.scf.for_loop_canonicalization
  transform.apply_patterns.canonicalization
}
transform.iree.apply_licm %16
transform.iree.apply_cse %16
%forall_op_4, %tiled_op_5 = transform.structured.tile_to_forall_op %10 num_threads [2, 64, 1]
```



```
builtin.module {
  module {
    transform.sequence  failures(propagate) {
    ^bb0(%arg0: !transform.any_op):
      transform.iree.register_match_callbacks // Callback just also provides a handle to the fillop
      %0:2 = transform.iree.match_callback failures(propagate) "batch_matmul"(%arg0) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %forall_op, %tiled_op = transform.structured.tile_to_forall_op %0#1   num_threads [] tile_sizes [64, 64, 1](mapping = [#gpu.block<z>, #gpu.block<y>, #gpu.block<x>]) : (!transform.any_op) ->
(!transform.any_op, !transform.any_op)
      %1 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %1 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %1 : !transform.any_op
      transform.iree.apply_cse %1 : !transform.any_op
      %fused_op, %new_containing_op = transform.structured.fuse_into_containing_op %0#0 into %forall_op : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
      transform.iree.populate_workgroup_count_region_using_num_threads_slice %forall_op : (!transform.any_op) -> ()
      %tiled_linalg_op, %loops = transform.structured.tile %tiled_op[0, 0, 0, 16] : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %2, %_ = transform.structured.pad %tiled_linalg_op {copy_back = false, pack_paddings = [1, 1, 1, 1], pad_to_multiple_of = [1, 1, 1, 1], padding_dimensions = [0, 1, 2, 3], padding_values =
[0.000000e+00 : f32, 0.000000e+00 : f32, 0.000000e+00 : f32]} : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %3 = get_producer_of_operand %2[2] : (!transform.any_op) -> !transform.any_op
      %4 = cast %3 : !transform.any_op to !transform.op<"tensor.pad">
      %5 = transform.structured.hoist_pad %4 by 1 loops : (!transform.op<"tensor.pad">) -> !transform.any_op
      %6 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %6 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
        transform.apply_patterns.tensor.fold_tensor_subset_ops
        transform.apply_patterns.tensor.merge_consecutive_insert_extract_slice
      } : !transform.any_op
      transform.iree.apply_licm %6 : !transform.any_op
      transform.iree.apply_cse %6 : !transform.any_op

      %7 = transform.structured.match ops{["linalg.fill"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      %8 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %8 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %8 : !transform.any_op
      transform.iree.apply_cse %8 : !transform.any_op
      %9 = transform.structured.match ops{["tensor.parallel_insert_slice"]} in %arg0 : (!transform.any_op) ->
      %10 = transform.structured.insert_slice_to_copy %9 : (!transform.any_op) -> !transform.any_op
      %11 = get_producer_of_operand %2[0] : (!transform.any_op) -> !transform.any_op
      %12 = get_producer_of_operand %2[1] : (!transform.any_op) -> !transform.any_op
      %13 = transform.structured.rewrite_in_destination_passing_style %12 : (!transform.any_op) -> !transform
      %forall_op_0, %tiled_op_1 = transform.structured.tile_to_forall_op %11   num_threads [1, 32, 4] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %14 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %14 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %14 : !transform.any_op
      transform.iree.apply_cse %14 : !transform.any_op
      %15 = transform.structured.match ops{["scf.if"]} in %forall_op_0 : (!transform.any_op) -> !transform.any_op
      transform.scf.take_assumed_branch %15 take_else_branch : (!transform.any_op) -> ()
      %forall_op_2, %tiled_op_3 = transform.structured.tile_to_forall_op %13   num_threads [8, 16, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %16 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %16 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %16 : !transform.any_op
      transform.iree.apply_cse %16 : !transform.any_op
      %forall_op_4, %tiled_op_5 = transform.structured.tile_to_forall_op %10   num_threads [2, 64, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %17 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %17 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %17 : !transform.any_op
      transform.iree.apply_cse %17 : !transform.any_op
      %forall_op_6, %tiled_op_7 = transform.structured.tile_to_forall_op %2   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %18 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %18 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %18 : !transform.any_op
      transform.iree.apply_cse %18 : !transform.any_op
      %forall_op_8, %tiled_op_9 = transform.structured.tile_to_forall_op %7   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %19 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %19 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %19 : !transform.any_op
      transform.iree.apply_cse %19 : !transform.any_op
```

```
%20 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %20 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %20 : !transform.any_op
      transform.iree.apply_cse %20 : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_1 vector_sizes [64, 2, 4] : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_5 vector_sizes [32, 1, 1] : !transform.any_op
      %21 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %21 {
        transform.apply_patterns.vector.lower_masked_transfers
      } : !transform.any_op
      %22 = transform.structured.vectorize %21 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %22 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %22 : !transform.any_op
      transform.iree.apply_cse %22 : !transform.any_op
      %23 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %23 {
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %23 : !transform.any_op
      transform.iree.apply_cse %23 : !transform.any_op
      transform.iree.eliminate_empty_tensors %arg0 : (!transform.any_op) -> ()
      %24 = transform.iree.bufferize {target_gpu} %arg0 : (!transform.any_op) -> !transform.any_op

      %25 = transform.structured.match ops{["func.func"]} in %24 : (!transform.any_op) -> !transform.any_op
      transform.iree.apply_buffer_optimizations %25 : (!transform.any_op) -> () // NO effect here
      %26 = transform.structured.match ops{["func.func"]} in %24 : (!transform.any_op) -> !transform.any_op
      transform.iree.forall_to_workgroup %26 : (!transform.any_op) -> ()
      transform.iree.map_nested_forall_to_gpu_threads %26 workgroup_dims = [64, 2, 1] warp_dims = [2, 2, 1] : (!transform.any_op) -> ()
      %27 = transform.iree.eliminate_gpu_barriers %26 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %27 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
                                                lization

                                               ansform.any_op
                                               nsform.any_op
                                               27 : (!transform.any_op) -> ()

                                               old_memref_alias_ops

        transform.apply_patterns.memref.extract_address_computations
      } : !transform.any_op
      apply_patterns to %27 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %27 : !transform.any_op
      transform.iree.apply_cse %27 : !transform.any_op
      %28 = transform.structured.match ops{["scf.for"]} in %27 : (!transform.any_op) -> !transform.op<"scf.for">
      transform.iree.synchronize_loop %28 : (!transform.op<"scf.for">) -> ()
      %29 = transform.structured.hoist_redundant_vector_transfers %27 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %29 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %29 : !transform.any_op
      transform.iree.apply_cse %29 : !transform.any_op
      transform.iree.apply_buffer_optimizations %29 : (!transform.any_op) -> ()
      %30 = transform.iree.eliminate_gpu_barriers %29 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      %31 = transform.structured.match ops{["memref.alloc"]} in %30 : (!transform.any_op) -> !transform.op<"memref.alloc">
      %32 = transform.memref.multibuffer %31 {factor = 2 : i64, skip_analysis} : (!transform.op<"memref.alloc">) -> !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.vector.transfer_to_scf   max_transfer_rank = 1 full_unroll = true
      } : !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      transform.iree.create_async_groups %30 : (!transform.any_op) -> ()


      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      %33 = transform.structured.match ops{["vector.contract"]} in %30 : (!transform.any_op) -> !transform.any_op
```

> 65% “Enablers”



```
builtin.module {
  module {
    transform.sequence  failures(propagate) {
    ^bb0(%arg0: !transform.any_op):
      transform.iree.register_match_callbacks // Callback just also provides a handle to the fillop
      %0:2 = transform.iree.match_callback failures(propagate) "batch_matmul"(%arg0) : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %forall_op, %tiled_op = transform.structured.tile_to_forall_op %0#1   num_threads [] tile_sizes [64, 64, 1](mapping = [#gpu.block<z>, #gpu.block<y>, #gpu.block<x>]) : (!transform.any_op) ->
(!transform.any_op, !transform.any_op)
      %1 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %1 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %1 : !transform.any_op
      transform.iree.apply_cse %1 : !transform.any_op
      %fused_op, %new_containing_op = transform.structured.fuse_into_containing_op %0#0 into %forall_op : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
      transform.iree.populate_workgroup_count_region_using_num_threads_slice %forall_op : (!transform.any_op) -> ()
      %tiled_linalg_op, %loops = transform.structured.tile %tiled_op[0, 0, 0, 16] : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %2, %_ = transform.structured.pad %tiled_linalg_op {copy_back = false, pack_paddings = [1, 1, 1, 1], pad_to_multiple_of = [1, 1, 1, 1], padding_dimensions = [0, 1, 2, 3], padding_values =
[0.000000e+00 : f32, 0.000000e+00 : f32, 0.000000e+00 : f32]} : (!transform.any_op) -> (!transform.any_op, !transform.any_op)
      %3 = get_producer_of_operand %2[2] : (!transform.any_op) -> !transform.any_op
      %4 = cast %3 : !transform.any_op to !transform.op<"tensor.pad">
      %5 = transform.structured.hoist_pad %4 by 1 loops : (!transform.op<"tensor.pad">) -> !transform.any_op
      %6 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %6 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
        transform.apply_patterns.tensor.fold_tensor_subset_ops
        transform.apply_patterns.tensor.merge_consecutive_insert_extract_slice
      } : !transform.any_op
      transform.iree.apply_licm %6 : !transform.any_op
      transform.iree.apply_cse %6 : !transform.any_op

      %7 = transform.structured.match ops{["linalg.fill"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      %8 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %8 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %8 : !transform.any_op
      transform.iree.apply_cse %8 : !transform.any_op
      %9 = transform.structured.match ops{["tensor.parallel_insert_slice"]} in %arg0 : (!tra
      %10 = transform.structured.insert_slice_to_copy %9 : (!transform.any_op) -> !transform
      %11 = get_producer_of_operand %2[0] : (!transform.any_op) -> !transform.any_op
      %12 = get_producer_of_operand %2[1] : (!transform.any_op) -> !transform.any_op
      %13 = transform.structured.rewrite_in_destination_passing_style %12 : (!transform.any_
      %forall_op_0, %tiled_op_1 = transform.structured.tile_to_forall_op %11   num_threads [1, 32, 4] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %14 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %14 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %14 : !transform.any_op
      transform.iree.apply_cse %14 : !transform.any_op
      %15 = transform.structured.match ops{["scf.if"]} in %forall_op_0 : (!transform.any_op) -> !transform.any_op
      transform.scf.take_assumed_branch %15 take_else_branch : (!transform.any_op) -> ()
      %forall_op_2, %tiled_op_3 = transform.structured.tile_to_forall_op %13   num_threads [8, 16, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %16 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %16 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %16 : !transform.any_op
      transform.iree.apply_cse %16 : !transform.any_op
      %forall_op_4, %tiled_op_5 = transform.structured.tile_to_forall_op %10   num_threads [2, 64, 1] tile_sizes [](mapping = [#gpu.linear<z>, #gpu.linear<y>, #gpu.linear<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %17 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %17 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %17 : !transform.any_op
      transform.iree.apply_cse %17 : !transform.any_op
      %forall_op_6, %tiled_op_7 = transform.structured.tile_to_forall_op %2   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %18 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %18 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %18 : !transform.any_op
      transform.iree.apply_cse %18 : !transform.any_op
      %forall_op_8, %tiled_op_9 = transform.structured.tile_to_forall_op %7   num_threads [1, 2, 64] tile_sizes [](mapping = [#gpu.thread<z>, #gpu.thread<y>, #gpu.thread<x>]) : (!transform.any_op)
-> (!transform.any_op, !transform.any_op)
      %19 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %19 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %19 : !transform.any_op
      transform.iree.apply_cse %19 : !transform.any_op
```

```
%20 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %20 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %20 : !transform.any_op
      transform.iree.apply_cse %20 : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_1 vector_sizes [64, 2, 4] : !transform.any_op
      transform.structured.masked_vectorize %tiled_op_5 vector_sizes [32, 1, 1] : !transform.any_op
      %21 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %21 {
        transform.apply_patterns.vector.lower_masked_transfers
      } : !transform.any_op
      %22 = transform.structured.vectorize %21 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %22 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %22 : !transform.any_op
      transform.iree.apply_cse %22 : !transform.any_op
      %23 = transform.structured.match ops{["func.func"]} in %arg0 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %23 {
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %23 : !transform.any_op
      transform.iree.apply_cse %23 : !transform.any_op
      transform.iree.eliminate_empty_tensors %arg0 : (!transform.any_op) -> ()
      %24 = transform.iree.bufferize {target_gpu} %arg0 : (!transform.any_op) -> !transform.any_op

      %25 = transform.structured.match ops{["func.func"]} in %24 : (!transform.any_op) -> !transform.any_op
      transform.iree.apply_buffer_optimizations %25 : (!transform.any_op) -> () // NO effect here
      %26 = transform.structured.match ops{["func.func"]} in %24 : (!transform.any_op) -> !transform.any_op
      transform.iree.forall_to_workgroup %26 : (!transform.any_op) -> ()
      transform.iree.map_nested_forall_to_gpu_threads %26 workgroup_dims = [64, 2, 1] warp_dims = [2, 2, 1] : (!transform.any_op) -> ()
      %27 = transform.iree.eliminate_gpu_barriers %26 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %27 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization

                                                                     any_op) -> ()

                                                                     ops

        transform.apply_patterns.memref.extract_address_computations
      } : !transform.any_op
      apply_patterns to %27 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %27 : !transform.any_op
      transform.iree.apply_cse %27 : !transform.any_op
      %28 = transform.structured.match ops{["scf.for"]} in %27 : (!transform.any_op) -> !transform.op<"scf.for">
      transform.iree.synchronize_loop %28 : (!transform.op<"scf.for">) -> ()
      %29 = transform.structured.hoist_redundant_vector_transfers %27 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %29 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %29 : !transform.any_op
      transform.iree.apply_cse %29 : !transform.any_op
      transform.iree.apply_buffer_optimizations %29 : (!transform.any_op) -> ()
      %30 = transform.iree.eliminate_gpu_barriers %29 : (!transform.any_op) -> !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      %31 = transform.structured.match ops{["memref.alloc"]} in %30 : (!transform.any_op) -> !transform.op<"memref.alloc">
      %32 = transform.memref.multibuffer %31 {factor = 2 : i64, skip_analysis} : (!transform.op<"memref.alloc">) -> !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.vector.transfer_to_scf   max_transfer_rank = 1 full_unroll = true
      } : !transform.any_op
      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      transform.iree.create_async_groups %30 : (!transform.any_op) -> ()


      apply_patterns to %30 {
        transform.apply_patterns.linalg.tiling_canonicalization
        transform.apply_patterns.iree.fold_fill_into_pad
        transform.apply_patterns.scf.for_loop_canonicalization
        transform.apply_patterns.canonicalization
        transform.apply_patterns.memref.fold_memref_alias_ops
      } : !transform.any_op
      transform.iree.apply_licm %30 : !transform.any_op
      transform.iree.apply_cse %30 : !transform.any_op
      %33 = transform.structured.match ops{[" vector.contract"]} in %30 : (!transform.any_op) -> !transform.any_op
```

4 different types of “Enablers” here

Enabler Categories by example: Loop Interchange

Observation: Interchanging the loops here might increase locality

```
scf.for %j = 0 to 4096 {
  %hoistable = ...
  scf.for %i = 0 to 4096 {
    %res = memref.load %values[%i, %j]
    func.call @use(%res, %hoistable)
  }
}
```

```
outer_for.interchange(inner_for)
```

- Only safe if we have a perfect loop nest

Enabler Categories by example: Loop Interchange

Observation: Interchanging the loops here might increase locality

```
scf.for %j = 0 to 4096 {
  %hoistable = ...
  scf.for %i = 0 to 4096 {
    %res = memref.load %values[%i, %j]
    func.call @use(%res, %hoistable)
  }
}
```

Not Interchangeable ✗

```
outer_for.interchange(inner_for)
```

- Only safe if we have a perfect loop nest

Enabler Categories by example: Loop Interchange

Observation: Interchanging the loops here might increase locality

Not Interchangeable ✗

```
scf.for %j = 0 to 4096 {
  %hoistable = ...
  scf.for %i = 0 to 4096 {
    %res = memref.load %values[%i, %j]
    func.call @use(%res, %hoistable)
  }
}
```

→

Interchangeable ✓

```
%hoistable = ...
scf.for %j = 0 to 4096 {

  scf.for %i = 0 to 4096 {
    %res = memref.load %values[%i, %j]
    func.call @use(%res, %hoistable)
  }
}
```

```
# adhoc solution for this specific payload program
outer_for.apply_licm() # loop invariant code motion
outer_for.interchange(inner_for)
```

- Only safe if we have a perfect loop nest
- Every user: “What canonicalizations do I have to apply to this specific payload?”

Enabler Categories by example

```
with handle.apply_patterns():
  structured.ApplyTilingCanonicalizationPatternsOp()
  loop.      ApplyForLoopCanonicalizationPatternsOp()
  transform. ApplyCanonicalizationPatternsOp()

handle.apply_licm()
handle.apply_cse()
```

~~Enabler Categories~~ by example

```
with handle.apply_patterns():
  structured.ApplyTilingCanonicalizationPatternsOp()
  loop.      ApplyForLoopCanonicalizationPatternsOp()
  transform. ApplyCanonicalizationPatternsOp()

handle.apply_licm()
handle.apply_cse()
```

Normalforms by example

Inspired by term rewriting

```python
class PerfectForNestForm(Normalform):
  def apply(cls, handle: OpHandle) -> None:
    with handle.apply_patterns():
      structured.ApplyTilingCanonicalizationPatternsOp()
      loop.      ApplyForLoopCanonicalizationPatternsOp()
      transform. ApplyCanonicalizationPatternsOp()

    handle.apply_licm()
    handle.apply_cse()
```

- Explicitly capture the structure we expect in the IR
- Defined by the transforms to reach this specific IR structure

Normalforms by example

```
scf.for %j = 0 to 4096 {
  %hoistable = ...
  scf.for %i = 0 to 4096 {
    %res = memref.load %values[%i, %j]
    func.call @use(%res, %hoistable)
  }
}
```

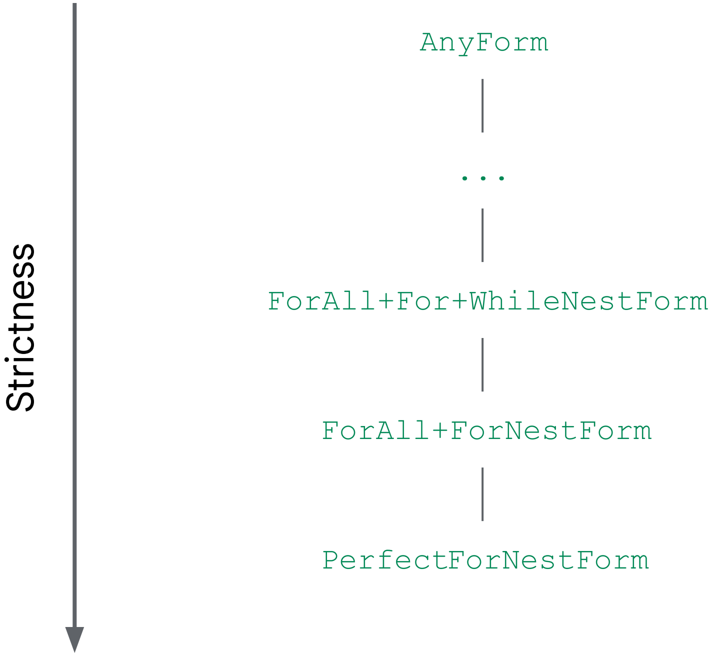
Not Interchangeable ✗

```
%hoistable = ...
scf.for %j = 0 to 4096 {

  scf.for %i = 0 to 4096 {
    %res = memref.load %values[%i, %j]
    func.call @use(%res, %hoistable)
  }
}
```

Interchangeable ✓

```
# General solution for loop interchange
outer_for.normalize(PerfectForNestForm)
outer_for.interchange(inner_for)
```

Normalforms by example: Hierarchy



Strictness

AnyForm

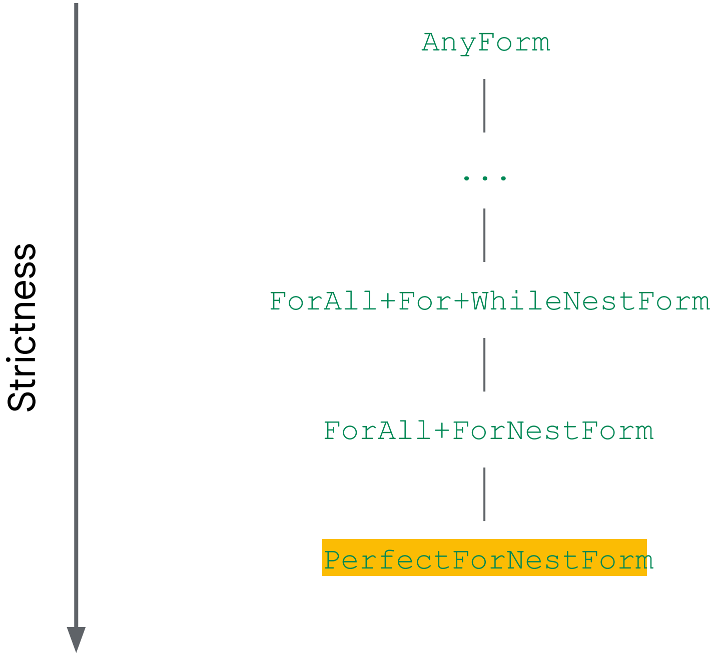
...

ForAll+For+WhileNestForm

ForAll+ForNestForm

PerfectForNestForm

Normalforms by example: Hierarchy



Strictness

AnyForm

...

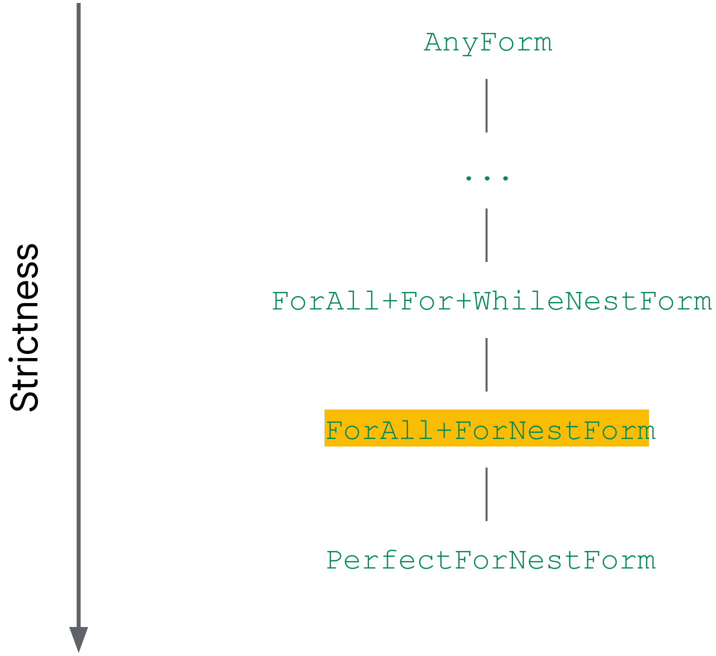
ForAll+For+WhileNestForm

ForAll+ForNestForm

PerfectForNestForm

```
scf.for {
  scf.for {
    //...
  }
}
```

Normalforms by example: Hierarchy



Strictness

AnyForm

...

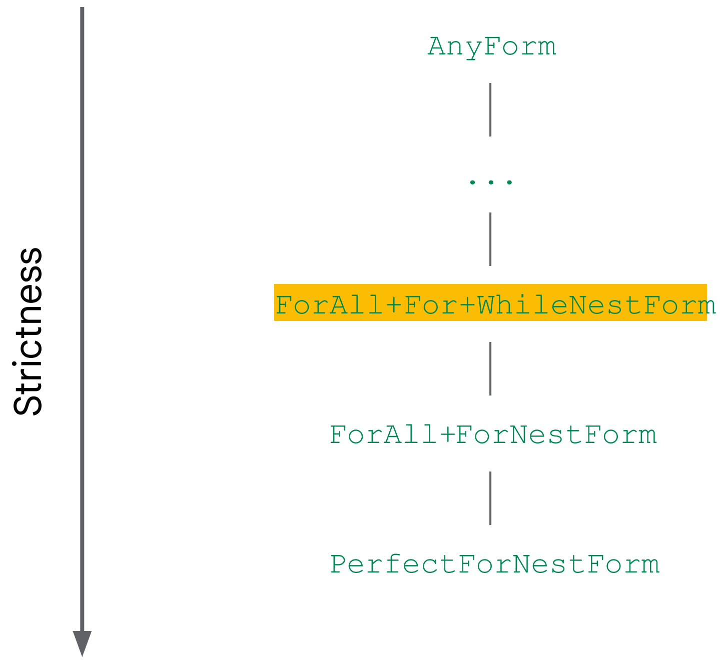
ForAll+For+WhileNestForm

ForAll+ForNestForm

PerfectForNestForm

```
scf.for {
  scf.forAll {
    //...
  }
}
```

Normalforms by example: Hierarchy



Strictness

AnyForm

...

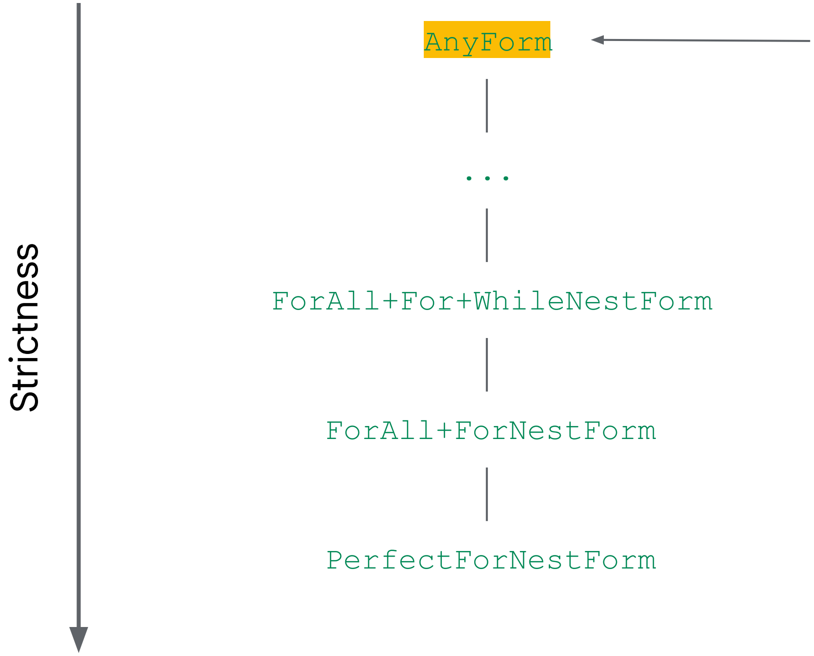
ForAll+For+WhileNestForm

ForAll+ForNestForm

PerfectForNestForm

```
scf.for {
  scf.forAll {
    scf.while {
      //...
    }
  }
}
```

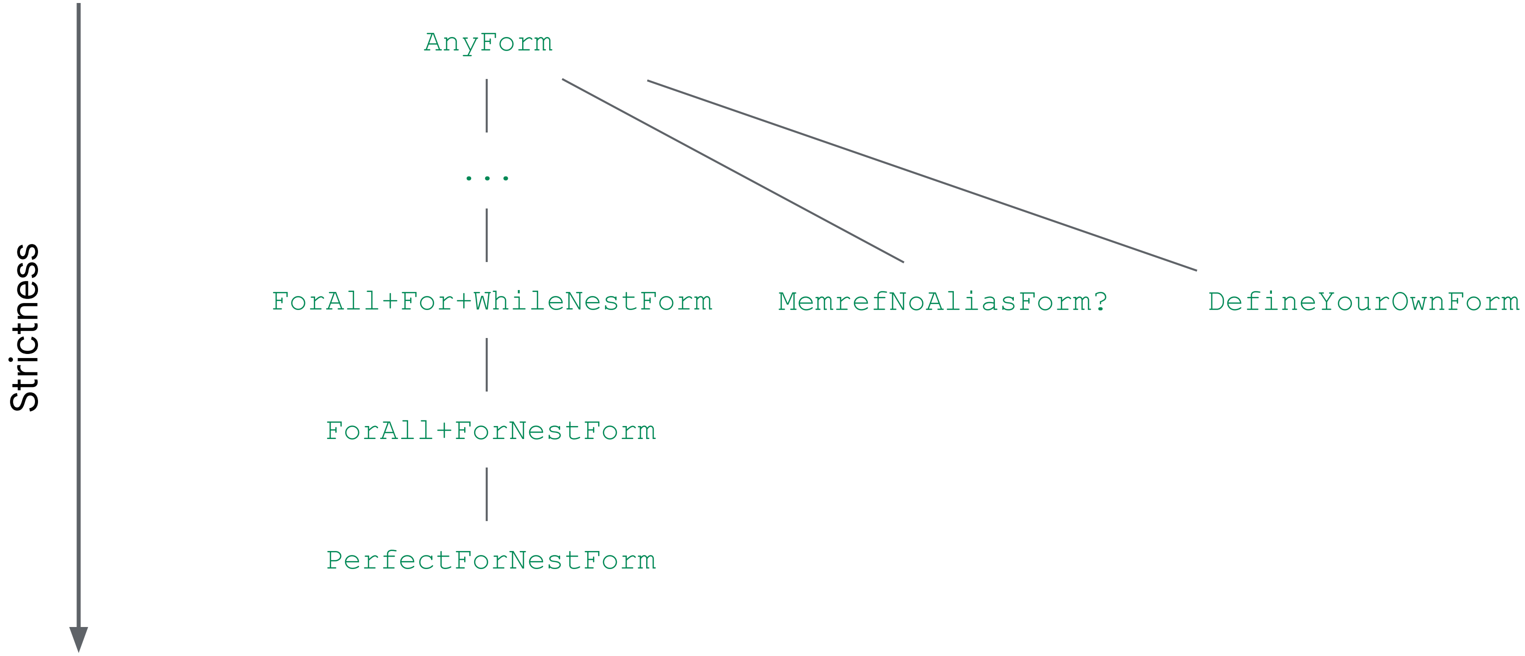
Normalforms by example: Hierarchy



Weakest form. All valid IR can be considered to be in AnyForm

```
scf.for {
  arith.constant
  scf.forAll {
    scf.while {
      //...
    }
  }
}
```

Normalforms by example: Hierarchy



Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition

Schedule

```
def schedule(module: OpHandle) -> None:
```

Example Payload IR

Normalform

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition

Schedule

```
def schedule(module: OpHandle) -> None:
```

Example Payload IR

```
module {
  scf.for %j = 0 to 4096 {
    %hoistable = ...
    scf.for %i = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

```
AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition

Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)
```

Example Payload IR

```
module {
  scf.for %j = 0 to 4096 {
    %hoistable = ...
    scf.for %i = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

```
AnyForm
  AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition

Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
```

Example Payload IR

```
module {
  scf.for %j = 0 to 4096 {
    %hoistable = ...
    scf.for %i = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

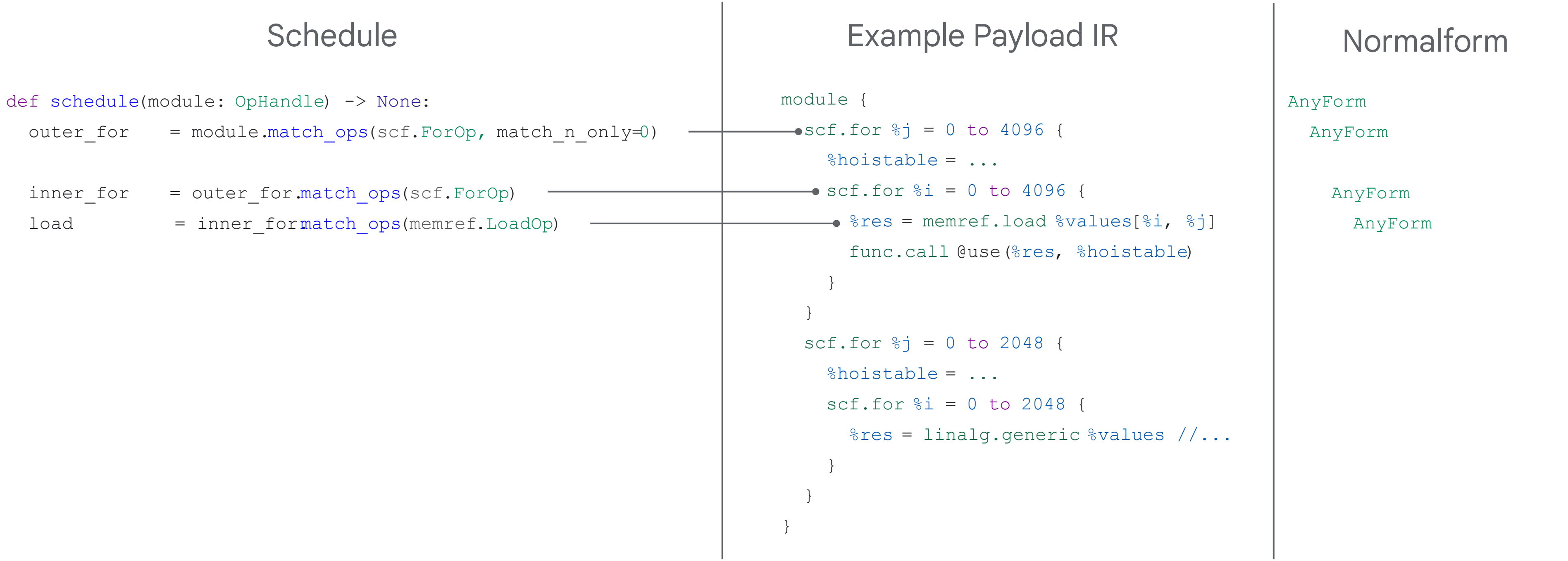
```
AnyForm
  AnyForm

    AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition



Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_for.match_ops(memref.LoadOp)
```

Example Payload IR

```
module {
  scf.for %j = 0 to 4096 {
    %hoistable = ...
    scf.for %i = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

```
AnyForm
  AnyForm

    AnyForm
      AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition

Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_formatch_ops(memref.LoadOp)
  outer_for.interchange(inner_for)
```

Example Payload IR

```
module {
  scf.for %j = 0 to 4096 {
    %hoistable = ...
    scf.for %i = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

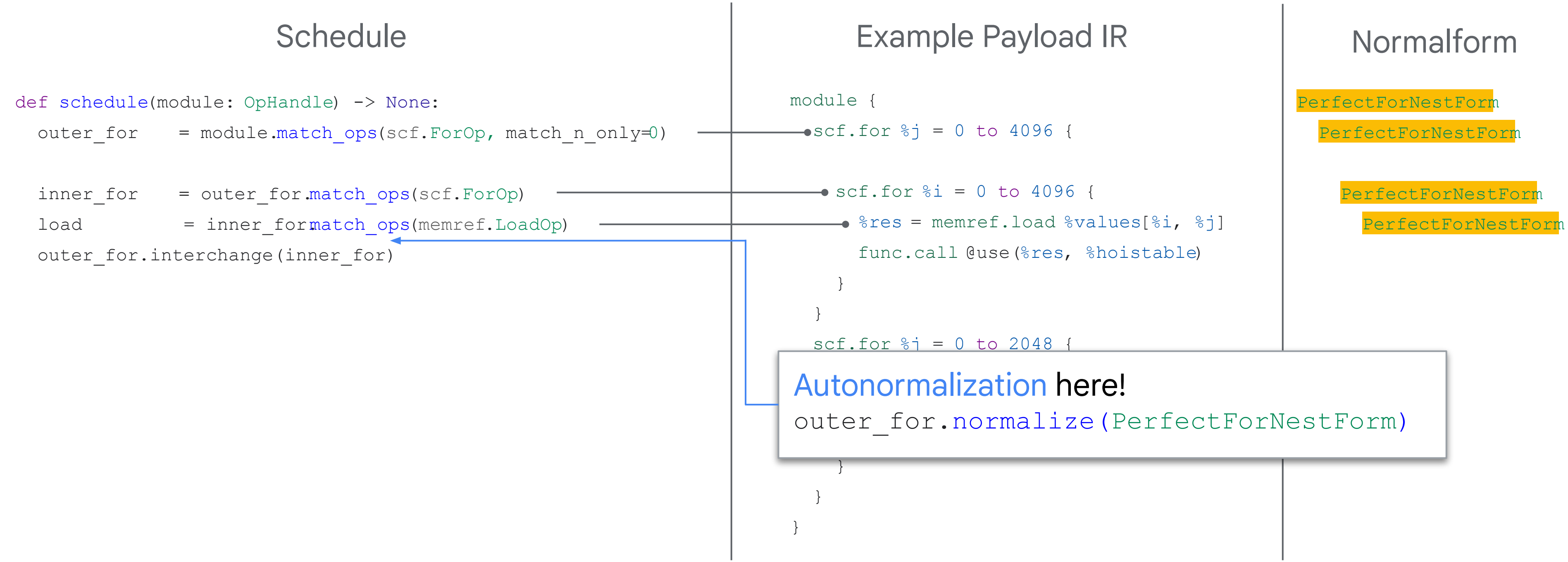
```
AnyForm
  AnyForm

    AnyForm
      AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition



Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_for.match_ops(memref.LoadOp)
  outer_for.interchange(inner_for)
```

Example Payload IR

```
module {
  scf.for %j = 0 to 4096 {

    scf.for %i = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {




    }
  }
}
```

Normalform

PerfectForNestForm

PerfectForNestForm

PerfectForNestForm

PerfectForNestForm

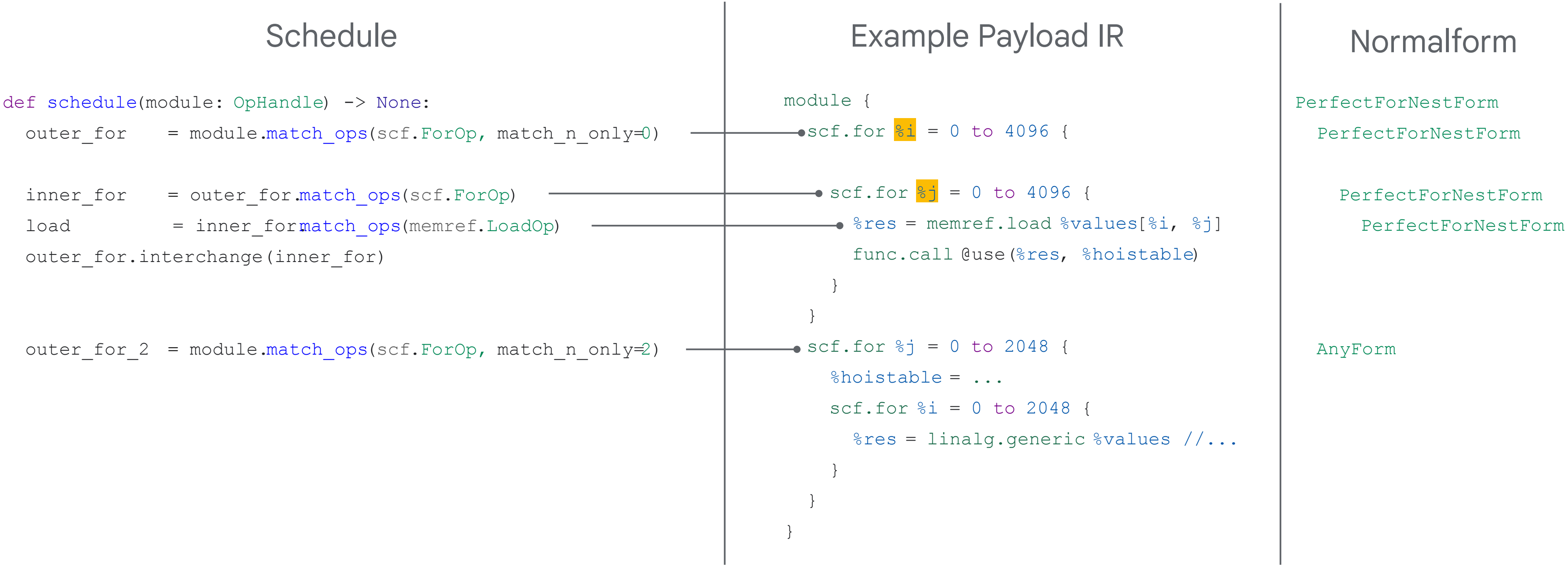
Autonormalization here!

```
outer_for.normalize(PerfectForNestForm)
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition
Postcondition



Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_for.match_ops(memref.LoadOp)
  outer_for.interchange(inner_for)


  outer_for_2  = module.match_ops(scf.ForOp, match_n_only=2)
```

Example Payload IR

```
module {
  scf.for %i = 0 to 4096 {

    scf.for %j = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

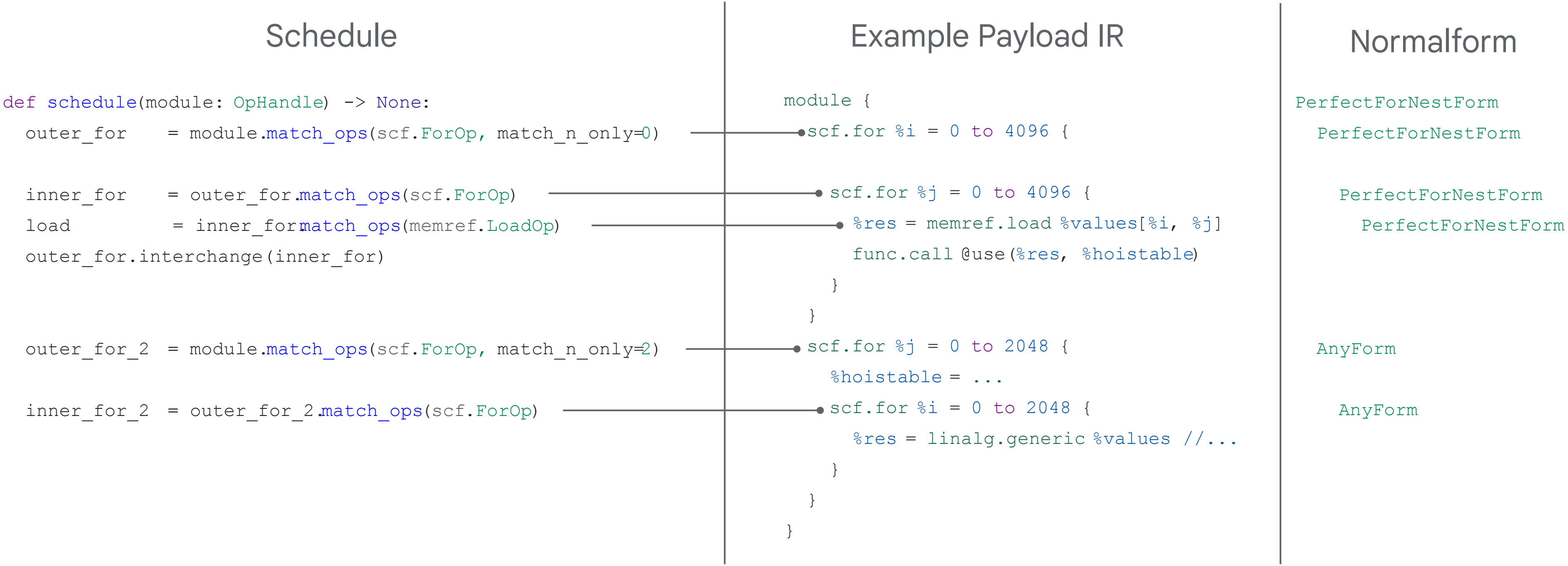
```
PerfectForNestForm
  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm




  AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition



Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_for.match_ops(memref.LoadOp)
  outer_for.interchange(inner_for)


  outer_for_2  = module.match_ops(scf.ForOp, match_n_only=2)

  inner_for_2  = outer_for_2.match_ops(scf.ForOp)
```

Example Payload IR

```
module {
  scf.for %i = 0 to 4096 {

    scf.for %j = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

```
PerfectForNestForm
  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm



  AnyForm

    AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,        Precondition
           enforced_normalform=PerfectForNestForm)        Postcondition
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_formatch_ops(memref.LoadOp)
  outer_for.interchange(inner_for)


  outer_for_2  = module.match_ops(scf.ForOp, match_n_only=2)

  inner_for_2  = outer_for_2.match_ops(scf.ForOp)
  linalg_op    = inner_for_2.match_ops(linalg.GenericOp)
```

Example Payload IR

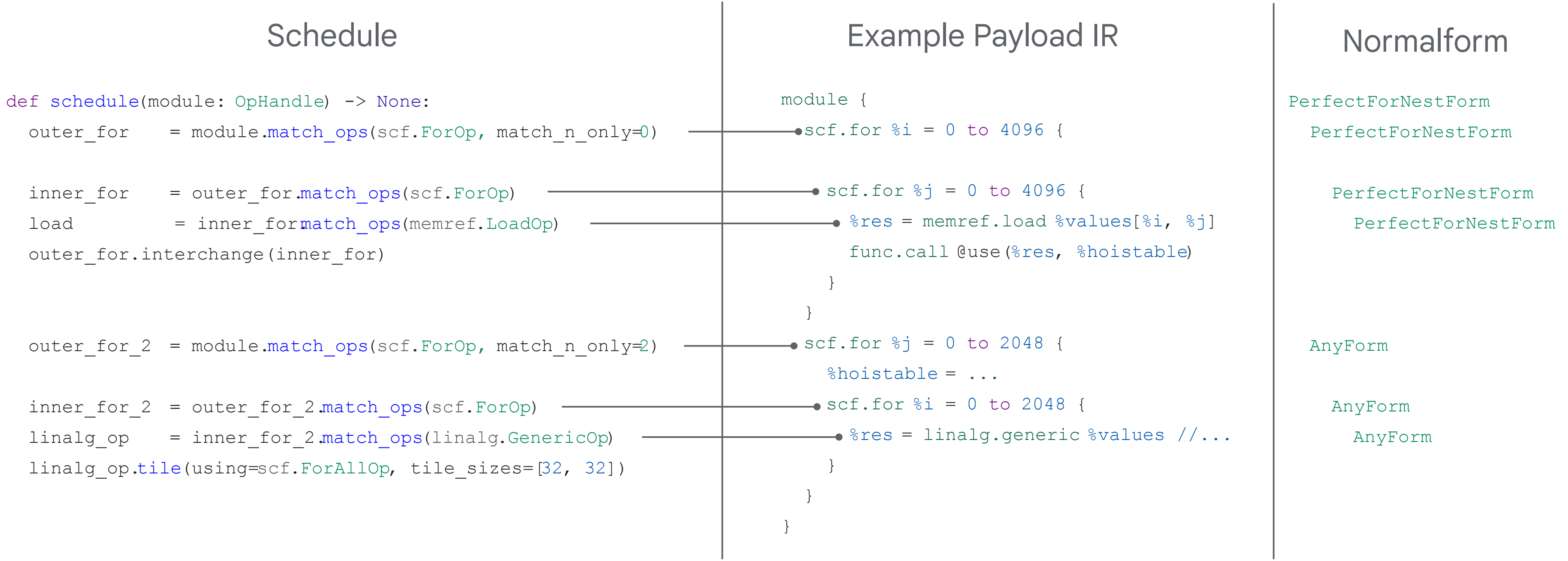
```
module {
  scf.for %i = 0 to 4096 {

    scf.for %j = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

```
PerfectForNestForm
  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm




  AnyForm

    AnyForm
      AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,        Precondition
           enforced_normalform=PerfectForNestForm)        Postcondition
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```



Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_formatch_ops(memref.LoadOp)
  outer_for.interchange(inner_for)


  outer_for_2  = module.match_ops(scf.ForOp, match_n_only=2)

  inner_for_2  = outer_for_2.match_ops(scf.ForOp)
  linalg_op    = inner_for_2.match_ops(linalg.GenericOp)
  linalg_op.tile(using=scf.ForAllOp, tile_sizes=[32, 32])
```

Example Payload IR

```
module {
  scf.for %i = 0 to 4096 {

    scf.for %j = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {
    %hoistable = ...
    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

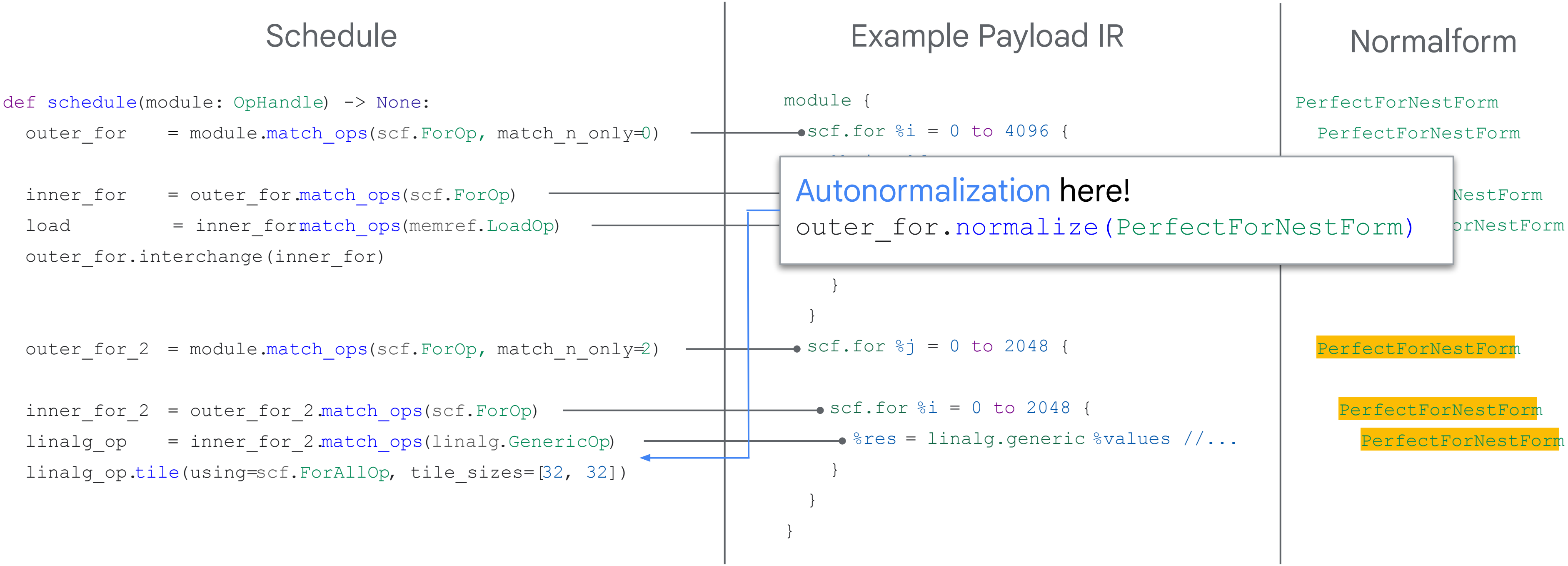
```
PerfectForNestForm
  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm



  AnyForm

    AnyForm
      AnyForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition



Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_formatch_ops(memref.LoadOp)
  outer_for.interchange(inner_for)


  outer_for_2  = module.match_ops(scf.ForOp, match_n_only=2)

  inner_for_2  = outer_for_2.match_ops(scf.ForOp)
  linalg_op    = inner_for_2.match_ops(linalg.GenericOp)
  linalg_op.tile(using=scf.ForAllOp, tile_sizes=[32, 32])
```

Example Payload IR

```
module {
  scf.for %i = 0 to 4096 {

    }
  }
  scf.for %j = 0 to 2048 {

    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

```
PerfectForNestForm
  PerfectForNestForm
      NestForm
      orNestForm


  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm
```

Autonormalization here!

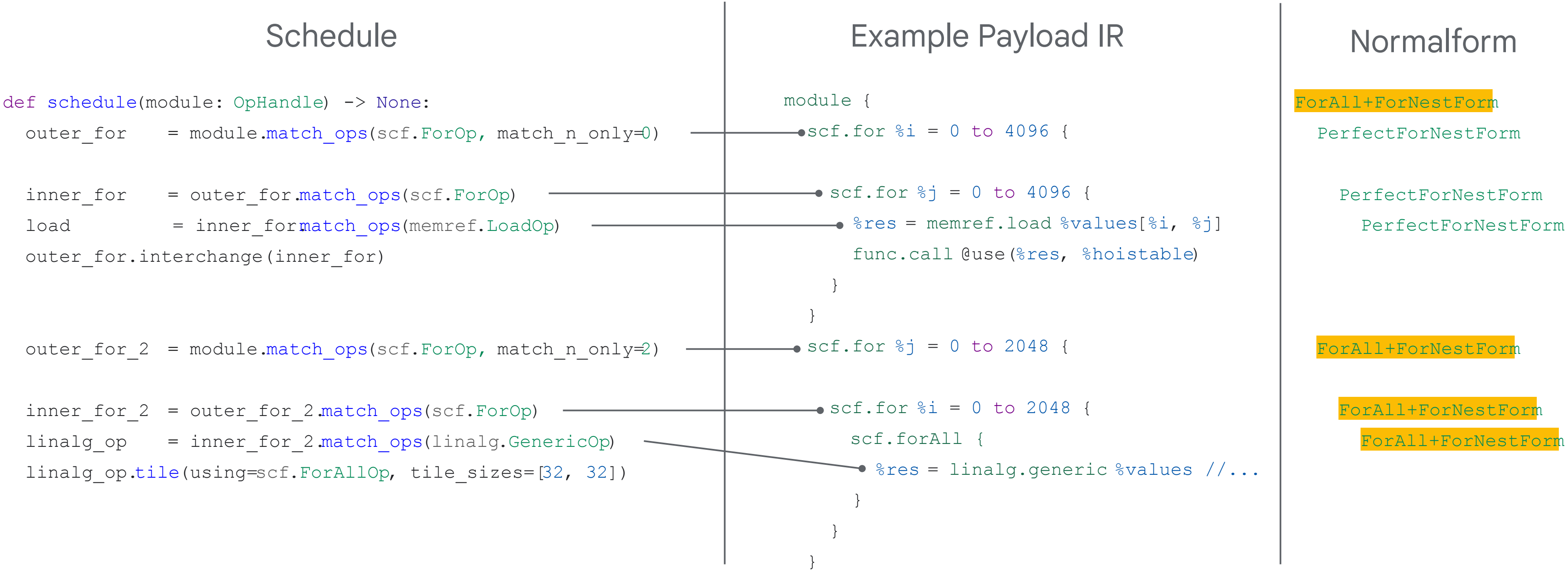
`outer_for.normalize(PerfectForNestForm)`

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition



Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load          = inner_for.match_ops(memref.LoadOp)
  outer_for.interchange(inner_for)


  outer_for_2  = module.match_ops(scf.ForOp, match_n_only=2)

  inner_for_2  = outer_for_2.match_ops(scf.ForOp)
  linalg_op    = inner_for_2.match_ops(linalg.GenericOp)
  linalg_op.tile(using=scf.ForAllOp, tile_sizes=[32, 32])
```

Example Payload IR

```
module {
  scf.for %i = 0 to 4096 {

    scf.for %j = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {

    scf.for %i = 0 to 2048 {
      scf.forAll {
        %res = linalg.generic %values //...
      }
    }
  }
```

Normalform

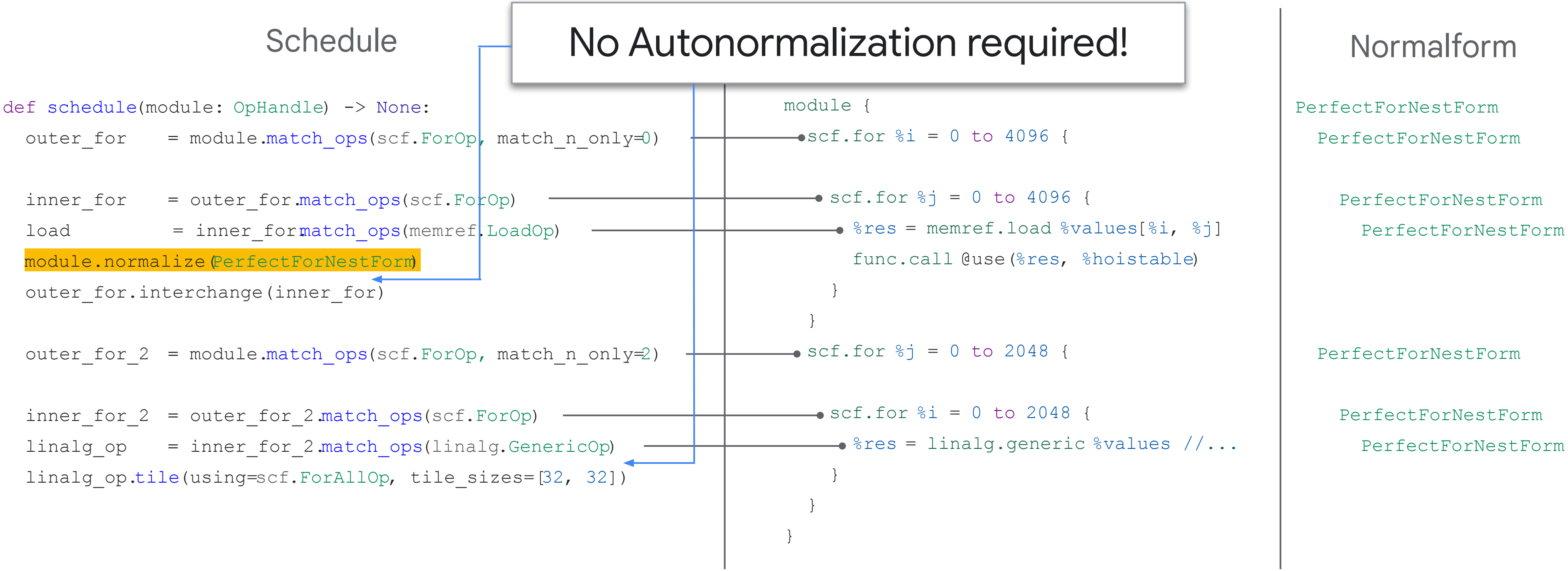
```
ForAll+ForNestForm
  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm




  ForAll+ForNestForm

    ForAll+ForNestForm
      ForAll+ForNestForm
```

Normalforms by example: Autonormalization

```
@transform(required_normalform=PerfectForNestForm,
           enforced_normalform=PerfectForNestForm)
def interchange(self: OpHandle, other_loop: OpHandle) -> OpHandle:
  # [...]
```

Precondition

Postcondition



No Autonormalization required!

Schedule

```
def schedule(module: OpHandle) -> None:
  outer_for    = module.match_ops(scf.ForOp, match_n_only=0)

  inner_for    = outer_for.match_ops(scf.ForOp)
  load         = inner_for.match_ops(memref.LoadOp)
  module.normalize(PerfectForNestForm)
  outer_for.interchange(inner_for)

  outer_for_2  = module.match_ops(scf.ForOp, match_n_only=2)

  inner_for_2  = outer_for_2.match_ops(scf.ForOp)
  linalg_op    = inner_for_2.match_ops(linalg.GenericOp)
  linalg_op.tile(using=scf.ForAllOp, tile_sizes=[32, 32])
```

```
module {
  scf.for %i = 0 to 4096 {

    scf.for %j = 0 to 4096 {
      %res = memref.load %values[%i, %j]
      func.call @use(%res, %hoistable)
    }
  }
  scf.for %j = 0 to 2048 {

    scf.for %i = 0 to 2048 {
      %res = linalg.generic %values //...
    }
  }
}
```

Normalform

```
PerfectForNestForm
  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm


  PerfectForNestForm

    PerfectForNestForm
      PerfectForNestForm
```

Designer of the transform thinks of the expected IR structure once, instead of every user every time

Parametric Schedules: Autotuning

.py

```python
def parametric_schedule(matmul: jasc.OpHandle) -> None:
  outer_tile_x = param()
  outer_tile_y = param(range: [1, 2, 4, 8])
  forall   = matmul.tile(tile_sizes=[outer_tile_x, outer_tile_y])
  forall_2 = matmul.tile(tile_sizes=[param(divides: outer_tile_x),
                                     param(divides: outer_tile_y)])
```

Generates parametric transform IR

.mlir

```mlir
transform.sequence (%matmul) {
  %outer_tile_x = transform.param
  %outer_tile_y = transform.param range [1, 2, 4, 8]
  %tiled        = transform.structured.tile %matmul[%outer_tile_x, %outer_tile_y]
  %inner_tile_x = transform.param divides[%outer_tile_x]
  %inner_tile_y = transform.param divides[%outer_tile_y]
  %inner_tiled  = transform.structured.tile %tiled[%inner_tile_x, %inner_tile_y]
}
```

Parametric schedule enables:

- Ship a parametric schedule, tune on user device
- Want to keep your model sizes secret but still collaborate? -> Model sizes become params

Autoscheduling enabled by Normalforms

- Autoscheduler does not have to generate “enabling” transforms anymore
- Easier to generate a valid schedule
- Extensible autoscheduling beyond just built-ins

Final schedule:

.py

```python
def batch_matmul_schedule(module: OpHandle) -> None:
  func = module.match_ops(func.FuncOp)
  matmul = module.match_ops(linalg.BatchMatmulOp)
  for_all = matmul.tile_to_forall(tile_sizes=[64, 64, 1], mapping=block_mapping)
  func.match_ops(linalg.FillOp).fuse_into(for_all).tile_to_forall(num_threads=[64, 2, 1])
  padded_input0, padded_input1, copy_op= matmul.tile([0, 0, 0, 16]).tiled_linalg_op.pad()
  padded_input0.tile_to_forall(num_threads=[1, 32, 4]).tiled_op.masked_vectorize([64, 2, 4])
  padded_input1.tile_to_forall(num_threads=[8, 16, 1])
  matmul.tile_to_forall(num_threads=[1, 2, 64])
  copy_op.tile_to_forall(num_threads=[2, 64, 1]).tiled_op.masked_vectorize([32, 1, 1])
  func.lower_vector_masked_transfers().generalize_named_ops().vectorize().bufferize()
  gpu_launch_op = module.gpu_lowering()
  gpu_launch_op.match_ops(scf.ForOp).synchronize_loop()
  func.hoist_redundant_vector_transfers()
  gpu_launch_op.barrier_elimination()
  gpu_launch_op.multibuffer()
  gpu_launch_op.create_async_groups()
  gpu_launch_op.pipeline_shared_memory_copies()
  func.lower_tensor_masks()
  # lower to llvm
```

Rough steps:

1. Tiling
2. Vectorization
3. Lower to required level
4. GPU specific transforms

↓

Transforms on different levels of abstraction expressed

Well, actually!

1. Schedule completely drives the compiler

```
def schedule(module: OpHandle) -> None:
  # [...]
  # lower to llvm is actually:
  module.convert_linalg_to_loops_pass()
  module.convert_scf_to_cf_pass()
  module.lower_affine_pass()
  module.convert_vector_to_llvm_pass()
  module.convert_math_to_llvm_pass()
  module.finalize_memref_to_llvm_conversion_pass()
  module.func_to_llvm_pass()
  module.reconcile_unrealized_casts_pass()
```

Every pass can be initiated through this interface

```
  module.run_pass(”MyPassName”)
```

Well, actually!

1. Schedule completely drives the compiler

```
def schedule(module: OpHandle) -> None:
  # [...]
  # lower to llvm is actually:
  module.convert_linalg_to_loops_pass()
  module.convert_scf_to_cf_pass()
  module.lower_affine_pass()
  module.convert_vector_to_llvm_pass()
  module.convert_math_to_llvm_pass()
  module.finalize_memref_to_llvm_conversion_pass()
  module.func_to_llvm_pass()
  module.reconcile_unrealized_casts_pass()
```

Every pass can be initiated through this interface

```
  module.run_pass(”MyPassName”)
```

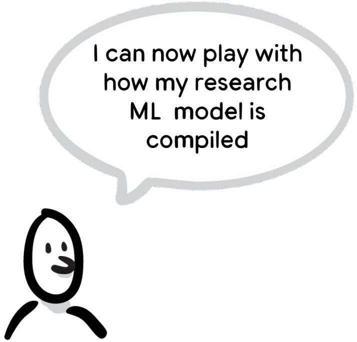
2. Constructing new Passes on-the-fly

```
with handle.apply_patterns():
  structured.ApplyTilingCanonicalizationPatternsOp()
  loop.      ApplyForLoopCanonicalizationPatternsOp()
  transform. ApplyCanonicalizationPatternsOp()
```

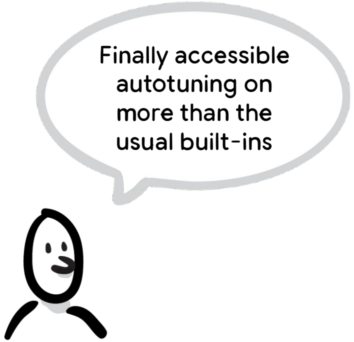
- Not possible with MLIR out-of-the-box
- Combination of patterns does not have to be known statically

-> We can precisely choose only the patterns we actually need

Contributions



ML Researcher


Finally accessible autotuning on more than the usual built-ins

Performance Engineer

Compiler Engineer

Martin Lücke

Alex Zinenko

Ingo Müller

Matthias Springer