# Deep Dive into the MLIR to LLVM IR Translation Mechanism

**Alex Zinenko (Brium Inc.)**

EuroLLVM 2025
Berlin,
April 15, 2025

# Deep Dive into the MLIR to LLVM IR Translation Mechanism

**Alex Zinenko (Brium Inc.)**

EuroLLVM 2025
Berlin,
April 15, 2025

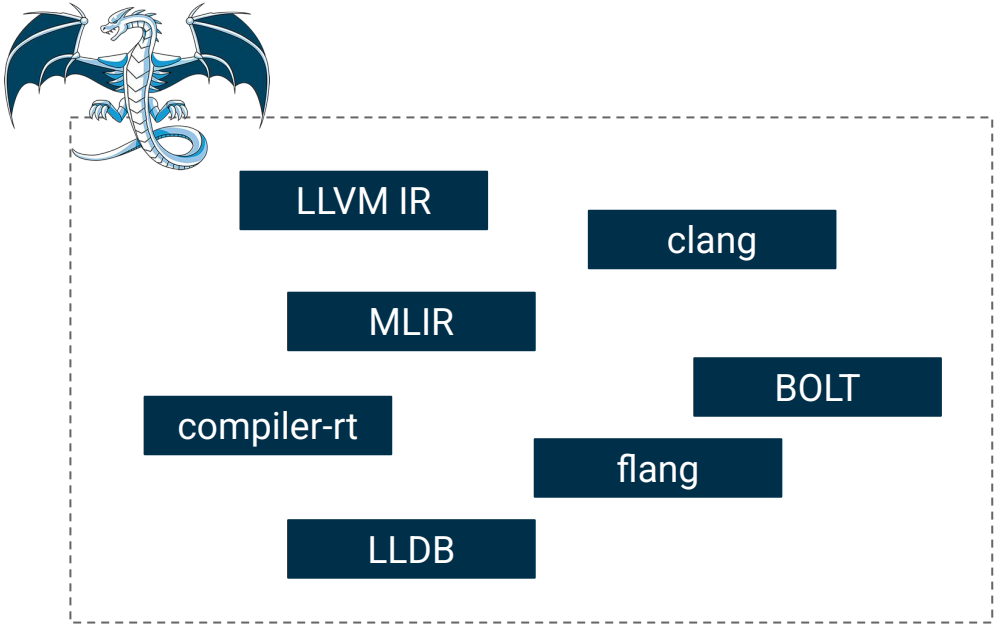
*Isn't this kind of boring for an opening keynote?*

# Past, Present and Future of the MLIR/LLVM Integration

## (and a deep dive into the MLIR to LLVM IR translation mechanism)

**Alex Zinenko (Brium Inc.)**

EuroLLVM 2025
Berlin,
April 15, 2025

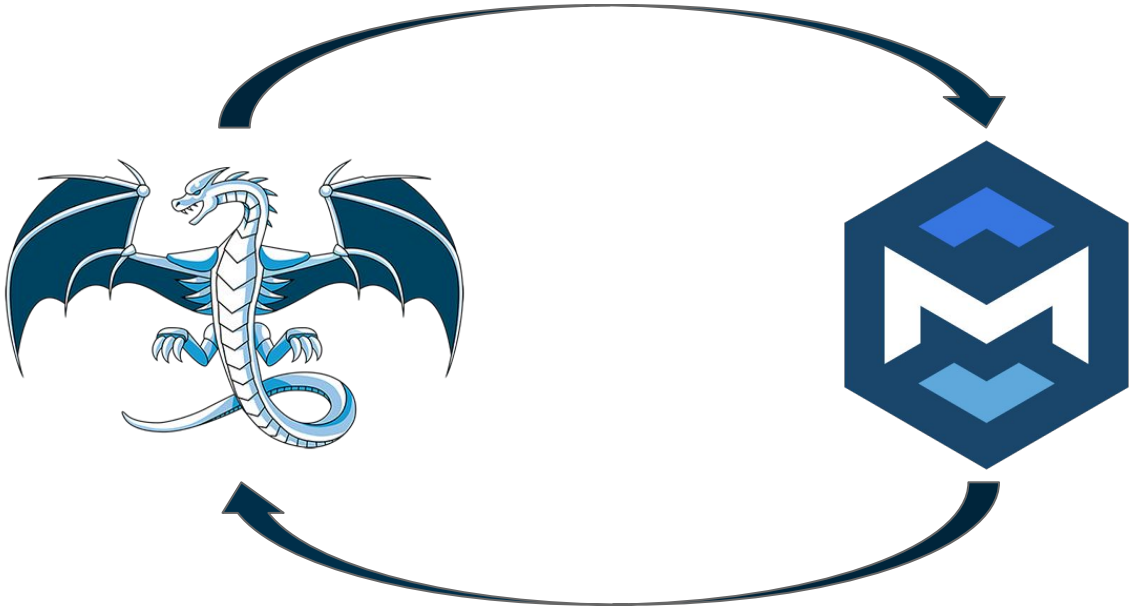



MLIR is part of the LLVM project

LLVM IR ~= SSA with useful instructions and types

MLIR IR ~= SSA + regions + BYO instructions and types

MLIR is a conceptual generalization (and a spiritual succession) of LLVM IR



There is a bi-directional connection between MLIR and LLVM IR

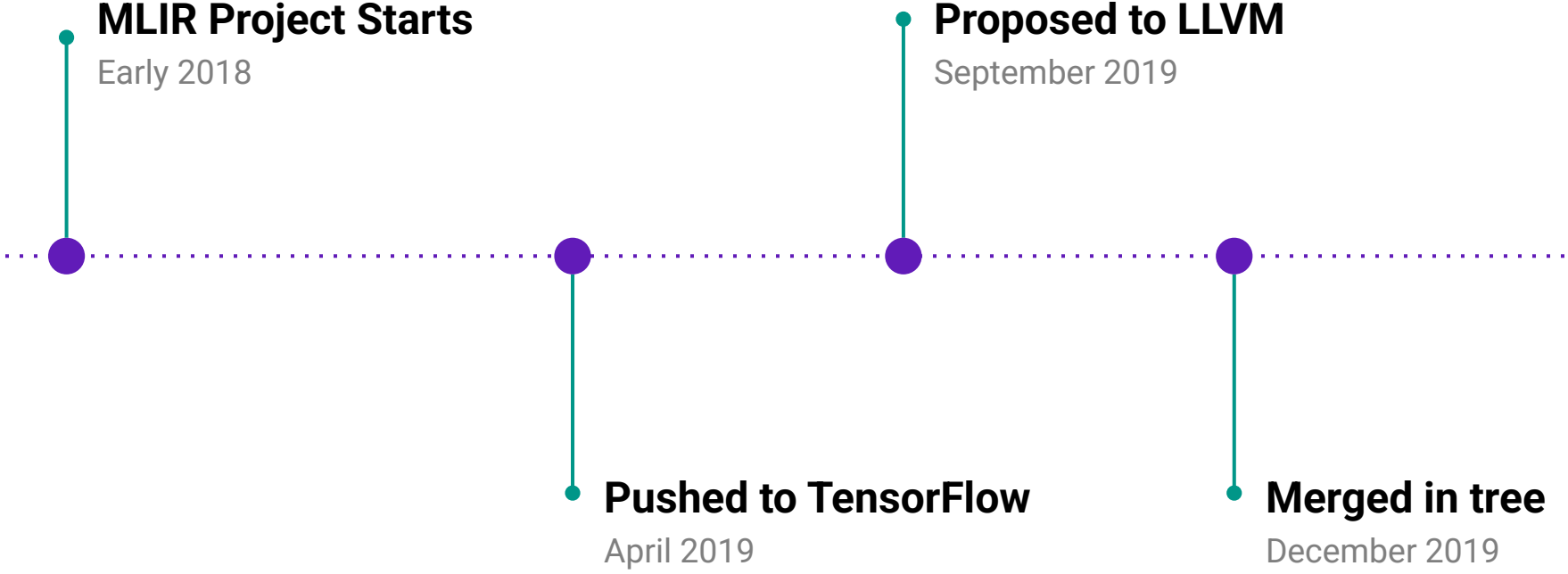
# The Short MLIR Origin Story

MLIR was born out of realization that a lot of work done in machine learning frameworks is secretly compiler work.

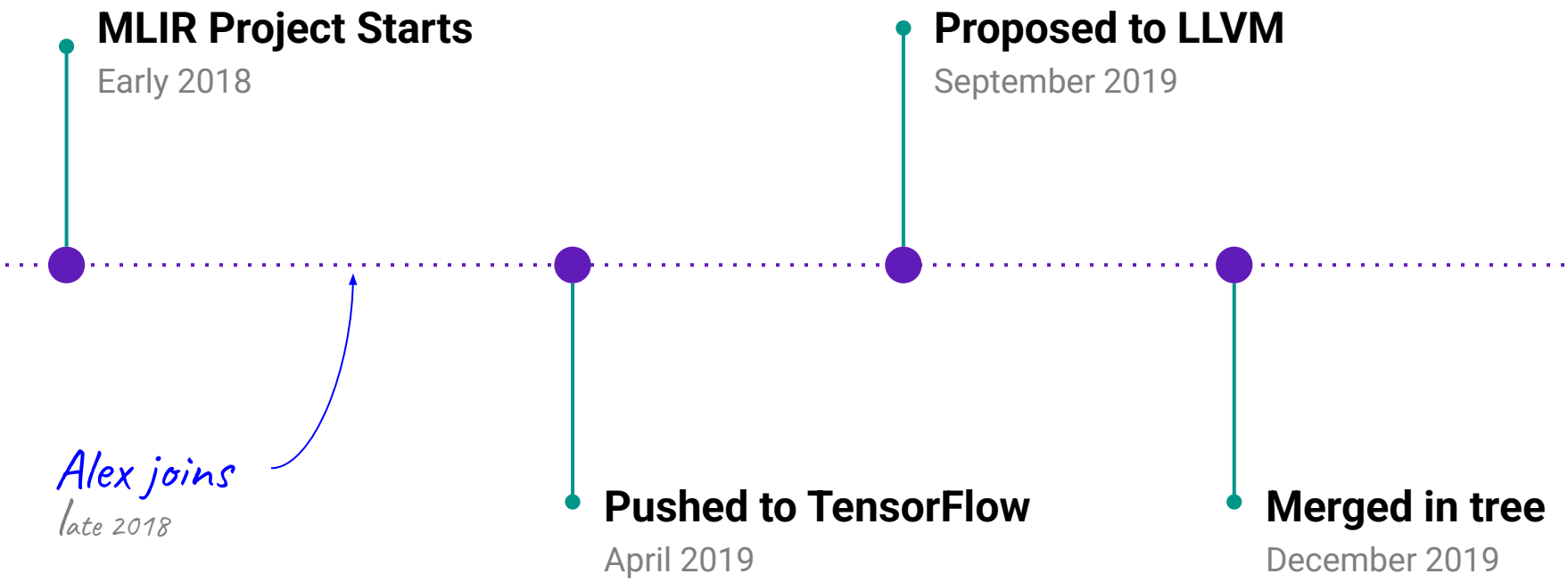
Without using ~~proper~~ LLVM-style compiler design.


* AI still doesn't understand topology

# Early Days of MLIR



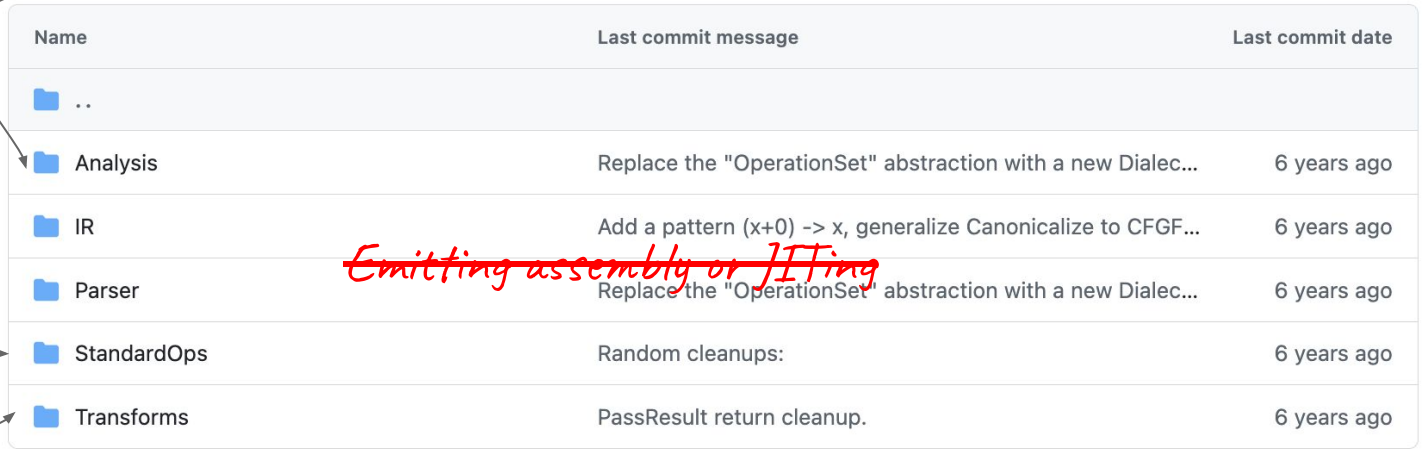
# Early Days of MLIR



# State of MLIR as of late 2018

Module, MLFunction, CFGFunction, Block, Statement (loop, conditional), Instruction

Dominance, First-party Polyhedral (yay!)



| Name | Last commit message | Last commit date |
|---|---|---|
| .. | | |
| Analysis | Replace the "OperationSet" abstraction with a new Dialec... | 6 years ago |
| IR | Add a pattern (x+0) -> x, generalize Canonicalize to CFGF... | 6 years ago |
| Parser | Replace the "OperationSet" abstraction with a new Dialec... | 6 years ago |
| StandardOps | Random cleanups: | 6 years ago |
| Transforms | PassResult return cleanup. | 6 years ago |

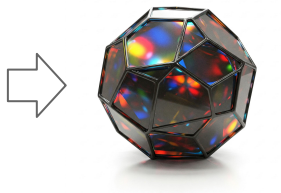
~~Emitting assembly or JITing~~

17 Operations, mostly arithmetic borrowed from LLVM IR

Canonicalizer, Affine Loop Transforms, Vectorizer

https://github.com/tensorflow/mlir/tree/acdf59d8c02bb3db8d9b3d65e00608c80ede6965/lib

# Making MLIR Executable



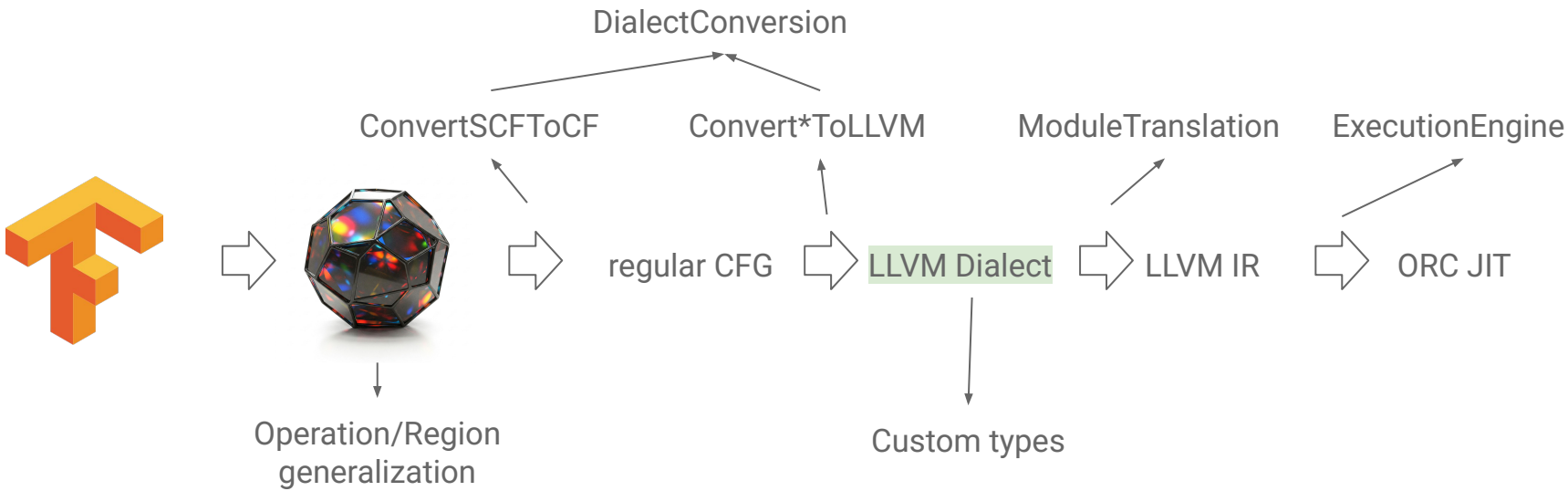
*Except we don't have branches*

*Except memref has arbitrarily complex layouts and no ABI*

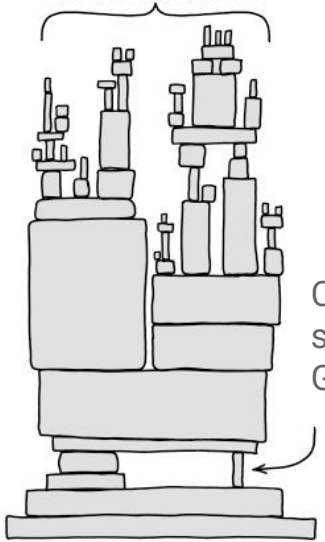
regular CFG → LLVM IR → ORC JIT

*Arithmetic on tensors requires loops*

# Introduction of the LLVM Dialect



All modern ML compiler infrastructure



Code written in 2018 by some junior-ish SWEs at Google as a side project

Adapted from https://xkcd.com/2347/

# LLVM Dialect

Design objective: reflect LLVM IR in MLIR as closely as possible while using MLIR core abstractions.

Corollary: LLVM dialect to LLVM IR translation is straightforward and the infrastructure is simple.

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}

llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}

declare void @bar(i64) #0

attributes #0 = { noinline }
```

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}

llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}

declare void @bar(i64) #0

attributes #0 = { noinline }
```

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}

llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}

declare void @bar(i64) #0

attributes #0 = { noinline }
```

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}

llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}

declare void @bar(i64) #0

attributes #0 = { noinline }
```

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}

llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}

declare void @bar(i64) #0

attributes #0 = { noinline }
```

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}

llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}

declare void @bar(i64) #0

attributes #0 = { noinline }
```

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}
```

```
llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}
```

```
declare void @bar(i64) #0

attributes #0 = { noinline }
```

# LLVM Dialect

```
llvm.func @foo(%arg0: i1, %arg1: i64) {
  %0 = llvm.mlir.constant(0 : i64) : i64
  %1 = llvm.mlir.constant(1 : i64) : i64
  llvm.cond_br %arg0, ^bb1(%0 : i64), ^bb1(%1 : i64)
^bb1(%2: i64):
  %3 = llvm.add %2, %arg1 : i64
  llvm.call @bar(%3) : (i64) -> ()
  llvm.return
}
```

*Functions are operations, not first-class entities*

```
llvm.func @bar(i64) attributes {no_inline}
```

```
define void @foo(i1 %0, i64 %1) {
 br i1 %0, label %3, label %6

3:
 %4 = phi i64 [ %7, %6 ], [ 0, %2 ]
 %5 = add i64 %4, %1
 call void @bar(i64 %5)
 ret void

6:
 %7 = phi i64 [ 1, %2 ]
 br label %3
}

declare void @bar(i64) #0

attributes #0 = { noinline }
```

# Translation to LLVM IR

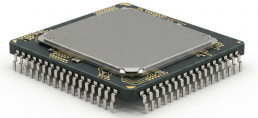
```
for f in functions:
```

# Translation to LLVM IR

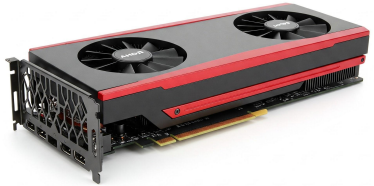
```
for f in functions:
 signatures[f] = translate_signature(f)

for g in globals:
 translate_global(g)

# same for certain kinds of metadata...

for f in functions:
 for b in f.body.blocks:
   blocks += create_block(b)
   for o in without_terminator(b.operations):
     translate_operation(o)
 connect_blocks_via_phis(blocks)
```

```
@_ZN4mlir2OpINS_4LLVM10LLVMFuncOpEJNS_7OpTrait9OneRegionENS3_11ZeroResultsENS3_14ZeroSuccessorsENS3_12ZeroOperandsENS3_12O
pInvariantsENS_19BytecodeOpInterface5TraitENS3_24AutomaticAllocationScopeENS3_19IsIsolatedFromAboveENS_17SymbolOpInterface
5TraitENS_19CallableOpInterface5TraitENS_19FunctionOpInterface5TraitEEE13getPropertiesIS2_EERNS_18PropertiesSelectorIT_vE4
typeEv(ptr noundef %4)
 %6 = getelementptr inbounds %"struct.mlir::LLVM::detail::LLVMFuncOpGenericAdaptorBase::Properties", ptr %5, i32 0, i32 25
 %7 = call i64 @_ZN4llvm16dyn_cast_or_nullIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %6)
 %8 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %9 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %8, i32 0, i32 0
 %10 = getelementptr inbounds %"class.mlir::Attribute", ptr %9, i32 0, i32 0
 %11 = inttoptr i64 %7 to ptr
 store ptr %11, ptr %10, align 8
 %12 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %13 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %12, i32 0, i32 0
 %14 = getelementptr inbounds %"class.mlir::Attribute", ptr %13, i32 0, i32 0
 %15 = load ptr, ptr %14, align 8
 %16 = ptrtoint ptr %15 to i64
 ret i64 %16
}

; Function Attrs: noinline optnone ssp uwtable(sync)
define linkonce_odr ptr @_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEC1ES1_NS_10ModRefInfoE(ptr noundef returned %0,
i32 noundef %1, i8 noundef zeroext %2) unnamed_addr #4 align 2 {
 %4 = alloca ptr, align 8
 %5 = alloca i32, align 4
 %6 = alloca i8, align 1
 store ptr %0, ptr %4, align 8
 store i32 %1, ptr %5, align 4
 store i8 %2, ptr %6, align 1
 %7 = load ptr, ptr %4, align 8
 %8 = load i32, ptr %5, align 4
 %9 = load i8, ptr %6, align 1
 %10 = call ptr @_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEC2ES1_NS_10ModRefInfoE(ptr noundef %7, i32 noundef %8, i8
noundef zeroext %9)
 ret ptr %7
}

; Function Attrs: mustprogress noinline nounwind optnone ssp uwtable(sync)
define linkonce_odr noundef nonnull align 4 dereferenceable(4) ptr
@_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEoRES2_(ptr noundef %0, i64 %1) #1 align 2 {
 %3 = alloca %"class.llvm::MemoryEffectsBase", align 4
 %4 = alloca ptr, align 8
 %5 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %3, i32 0, i32 0
 %6 = trunc i64 %1 to i32
 store i32 %6, ptr %5, align 4
 store ptr %0, ptr %4, align 8
 %7 = load ptr, ptr %4, align 8
 %8 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %3, i32 0, i32 0
 %9 = load i32, ptr %8, align 4
 %10 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %7, i32 0, i32 0
 %11 = load i32, ptr %10, align 4
 %12 = or i32 %11, %9
 store i32 %12, ptr %10, align 4
 ret ptr %7
}

; Function Attrs: mustprogress noinline nounwind optnone ssp uwtable(sync)
define linkonce_odr hidden zeroext i1
@_ZNKSt3__123__optional_storage_baseIN4mlir4LLVM17MemoryEffectsAttrELb0EE9has_valueB8ue170006Ev(ptr noundef %0) #1 align 2
{
 %2 = alloca ptr, align 8
 store ptr %0, ptr %2, align 8
 %3 = load ptr, ptr %2, align 8
 %4 = getelementptr inbounds %"struct.std::__1::__optional_destruct_base.1739", ptr %3, i32 0, i32 1
 %5 = load i8, ptr %4, align 8
 %6 = trunc i8 %5 to i1
 ret i1 %6
}

; Function Attrs: mustprogress noinline optnone ssp uwtable(sync)
define linkonce_odr i64 @_ZN4llvm16dyn_cast_or_nullIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %0) #0 {
 %2 = alloca %"class.mlir::LLVM::MemoryEffectsAttr", align 8
 %3 = alloca ptr, align 8
 store ptr %0, ptr %3, align 8
 %4 = load ptr, ptr %3, align 8
 %5 = call i64 @_ZN4llvm19dyn_cast_if_presentIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %4)
 %6 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %7 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %6, i32 0, i32 0
 %8 = getelementptr inbounds %"class.mlir::Attribute", ptr %7, i32 0, i32 0
 %9 = inttoptr i64 %5 to ptr
 store ptr %9, ptr %8, align 8
 %10 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
```

clang
llc



*These exist ?!*



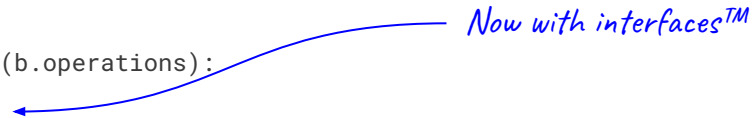
# Accelerator Support

The dialect system allows to add support for accelerators with little overhead.

NVVM dialect: 148 operations
ROCDL dialect: 153 operations
ArmSVE dialect: 37 operations
AMX dialect: 14 operations
and many, many more…

# Translation to LLVM IR

```
for f in functions:
 signatures[f] = translate_signature(f)

for g in globals:
 translate_global(g)

# same for certain kinds of metadata...

for f in functions:
 for b in f.body.blocks:
   blocks += create_block(b)
   for o in without_terminator(b.operations):
     translate_operation(o)
 connect_blocks_via_phis(blocks)
```



# LLVM Translation Dialect Interface

```
class LLVMTranslationDialectInterface : public DialectInterface::Base<...> {
public:
 // ...

 virtual LogicalResult
 convertOperation(Operation *op, llvm::IRBuilderBase &builder,
                  LLVM::ModuleTranslation &moduleTranslation) const {
   // ...
 }
```

*Operation to convert*

*Mappings between values, locations, functions, etc.*

*The usual way to build LLVM IR*

# LLVM Translation Dialect Interface

```
class LLVMTranslationDialectInterface : public DialectInterface::Base<...> {
public:
 // ...

 virtual LogicalResult
 convertOperation(...) const

 virtual LogicalResult
 amendOperation(Operation *op, ArrayRef<llvm::Instruction *> instructions,
                NamedAttribute attribute,
                LLVM::ModuleTranslation &moduleTranslation) const {
   // ...
 }
```

Set of produced instructions

Original operation

Mappings between values, locations, functions, etc.

Attribute that belongs to this dialect

# LLVM Translation Dialect Interface

```
class LLVMTranslationDialectInterface : public DialectInterface::Base<...> {
public:
 // ...

 virtual LogicalResult
 convertOperation(...) const

 virtual LogicalResult
 amendOperation(...) const

 virtual LogicalResult
 convertParameterAttr(LLVM::LLVMFuncOp function, int argIdx,
                      NamedAttribute attr,
                      LLVM::ModuleTranslation &moduleTranslation) const {
     // ...
 }
```

*Converted function*

*Argument index*

*Attribute that belongs to this dialect*

*Mappings between values, locations, functions, etc.*

```
@_ZN4mlir2OpINS_4LLVM10LLVMFuncOpEJNS_7OpTrait9OneRegionENS3_11ZeroResultsENS3_14ZeroSuccessorsENS3_12ZeroOperandsENS3_12O
pInvariantsENS_19BytecodeOpInterface5TraitENS3_24AutomaticAllocationScopeENS3_19IsIsolatedFromAboveENS_17SymbolOpInterface
5TraitENS_19CallableOpInterface5TraitENS_19FunctionOpInterface5TraitEEE13getPropertiesIS2_EERNS_18PropertiesSelectorIT_vE4
typeEv(ptr noundef %4)
 %6 = getelementptr inbounds %"struct.mlir::LLVM::detail::LLVMFuncOpGenericAdaptorBase::Properties", ptr %5, i32 0, i32 25
 %7 = call i64 @_ZN4llvm16dyn_cast_or_nullIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %6)
 %8 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %9 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %8, i32 0, i32 0
 %10 = getelementptr inbounds %"class.mlir::Attribute", ptr %9, i32 0, i32 0
 %11 = inttoptr i64 %7 to ptr
 store ptr %11, ptr %10, align 8
 %12 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %13 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %12, i32 0, i32 0
 %14 = getelementptr inbounds %"class.mlir::Attribute", ptr %13, i32 0, i32 0
 %15 = load ptr, ptr %14, align 8
 %16 = ptrtoint ptr %15 to i64
 ret i64 %16
}

; Function Attrs: noinline optnone ssp uwtable(sync)
define linkonce_odr ptr @_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEC1ES1_NS_10ModRefInfoE(ptr noundef returned %0,
i32 noundef %1, i8 noundef zeroext %2) unnamed_addr #4 align 2 {
 %4 = alloca ptr, align 8
 %5 = alloca i32, align 4
 %6 = alloca i8, align 1
 store ptr %0, ptr %4, align 8
 store i32 %1, ptr %5, align 4
 store i8 %2, ptr %6, align 1
 %7 = load ptr, ptr %4, align 8
 %8 = load i32, ptr %5, align 4
 %9 = load i8, ptr %6, align 1
 %10 = call ptr @_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEC2ES1_NS_10ModRefInfoE(ptr noundef %7, i32 noundef %8, i8
noundef zeroext %9)
 ret ptr %7
}

; Function Attrs: mustprogress noinline nounwind optnone ssp uwtable(sync)
define linkonce_odr noundef nonnull align 4 dereferenceable(4) ptr
@_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEoRES2_(ptr noundef %0, i64 %1) #1 align 2 {
 %3 = alloca %"class.llvm::MemoryEffectsBase", align 4
 %4 = alloca ptr, align 8
 %5 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %3, i32 0, i32 0
 %6 = trunc i64 %1 to i32
 store i32 %6, ptr %5, align 4
 store ptr %0, ptr %4, align 8
 %7 = load ptr, ptr %4, align 8
 %8 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %3, i32 0, i32 0
 %9 = load i32, ptr %8, align 4
 %10 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %7, i32 0, i32 0
 %11 = load i32, ptr %10, align 4
 %12 = or i32 %11, %9
 store i32 %12, ptr %10, align 4
 ret ptr %7
}

; Function Attrs: mustprogress noinline nounwind optnone ssp uwtable(sync)
define linkonce_odr hidden zeroext i1
@_ZNKSt3__123__optional_storage_baseIN4mlir4LLVM17MemoryEffectsAttrELb0EE9has_valueB8ue170006Ev(ptr noundef %0) #1 align 2
{
 %2 = alloca ptr, align 8
 store ptr %0, ptr %2, align 8
 %3 = load ptr, ptr %2, align 8
 %4 = getelementptr inbounds %"struct.std::__1::__optional_destruct_base.1739", ptr %3, i32 0, i32 1
 %5 = load i8, ptr %4, align 8
 %6 = trunc i8 %5 to i1
 ret i1 %6
}

; Function Attrs: mustprogress noinline optnone ssp uwtable(sync)
define linkonce_odr i64 @_ZN4llvm16dyn_cast_or_nullIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %0) #0 {
 %2 = alloca %"class.mlir::LLVM::MemoryEffectsAttr", align 8
 %3 = alloca ptr, align 8
 store ptr %0, ptr %3, align 8
 %4 = load ptr, ptr %3, align 8
 %5 = call i64 @_ZN4llvm19dyn_cast_if_presentIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %4)
 %6 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %7 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %6, i32 0, i32 0
 %8 = getelementptr inbounds %"class.mlir::Attribute", ptr %7, i32 0, i32 0
 %9 = inttoptr i64 %5 to ptr
 store ptr %9, ptr %8, align 8
 %10 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
```

clang
llc

Fortran needs this?!


Affine too?!!

# Dialects to the Rescue

```
llvm.func @test_omp_parallel_1() -> () {
  omp.parallel {
    omp.barrier
    omp.terminator
  }

  llvm.return
}
```

```
@0 = private unnamed_addr constant [23 x i8] c";unknown;unknown;0;0;;\00", align 1
@1 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 34, i32 0, i32 22, ptr @0 }, align 8
@2 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 2, i32 0, i32 22, ptr @0 }, align 8

define void @test_omp_parallel_1() {
  %omp_global_thread_num = call i32 @__kmpc_global_thread_num(ptr @2)
  call void (ptr, i32, ptr, ...)
    @__kmpc_fork_call(ptr @2, i32 0, ptr @test_omp_parallel_1..omp_par)
  ret void
}

define internal void @test_omp_parallel_1..omp_par(ptr noalias %tid.addr,
    ptr noalias %zero.addr) #0 {
  %tid.addr.local = alloca i32, align 4
  %0 = load i32, ptr %tid.addr, align 4
  store i32 %0, ptr %tid.addr.local, align 4
  %tid = load i32, ptr %tid.addr.local, align 4

  ; Concurrently executed code goes here
  %omp_global_thread_num2 = call i32 @__kmpc_global_thread_num(ptr @2)
  call void @__kmpc_barrier(ptr @1, i32 %omp_global_thread_num2)
  ret void
}
```

# Dialects to the Rescue

*Location information*

```
@0 = private unnamed_addr constant [23 x i8] c";unknown;unknown;0;0;;\00", align 1
@1 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 34, i32 0, i32 22, ptr @0 }, align 8
@2 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 2, i32 0, i32 22, ptr @0 }, align 8

define void @test_omp_parallel_1() {
  %omp_global_thread_num = call i32 @__kmpc_global_thread_num(ptr @2)
  call void (ptr, i32, ptr, ...)
    @__kmpc_fork_call(ptr @2, i32 0, ptr @test_omp_parallel_1..omp_par)
  ret void
}

define internal void @test_omp_parallel_1..omp_par(ptr noalias %tid.addr,
    ptr noalias %zero.addr) #0 {
  %tid.addr.local = alloca i32, align 4
  %0 = load i32, ptr %tid.addr, align 4
  store i32 %0, ptr %tid.addr.local, align 4
  %tid = load i32, ptr %tid.addr.local, align 4

  ; Concurrently executed code goes here
  %omp_global_thread_num2 = call i32 @__kmpc_global_thread_num(ptr @2)
  call void @__kmpc_barrier(ptr @1, i32 %omp_global_thread_num2)
  ret void
}
```

```
llvm.func @test_omp_parallel_1() -> () {
  omp.parallel {
    omp.barrier
    omp.terminator
  }

  llvm.return
}
```

# Dialects to the Rescue

```
llvm.func @test_omp_parallel_1() -> () {
  omp.parallel {
    omp.barrier
    omp.terminator
  }

  llvm.return
}
```

*Runtime call to create threads*

*Location information*

```
@0 = private unnamed_addr constant [23 x i8] c";unknown;unknown;0;0;;\00", align 1
@1 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 34, i32 0, i32 22, ptr @0 }, align 8
@2 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 2, i32 0, i32 22, ptr @0 }, align 8

define void @test_omp_parallel_1() {
  %omp_global_thread_num = call i32 @__kmpc_global_thread_num(ptr @2)
  call void (ptr, i32, ptr, ...)
    @__kmpc_fork_call(ptr @2, i32 0, ptr @test_omp_parallel_1..omp_par)
  ret void
}

define internal void @test_omp_parallel_1..omp_par(ptr noalias %tid.addr,
    ptr noalias %zero.addr) #0 {
  %tid.addr.local = alloca i32, align 4
  %0 = load i32, ptr %tid.addr, align 4
  store i32 %0, ptr %tid.addr.local, align 4
  %tid = load i32, ptr %tid.addr.local, align 4

  ; Concurrently executed code goes here
  %omp_global_thread_num2 = call i32 @__kmpc_global_thread_num(ptr @2)
  call void @__kmpc_barrier(ptr @1, i32 %omp_global_thread_num2)
  ret void
}
```

# Dialects to the Rescue

```
llvm.func @test_omp_parallel_1() -> () {
  omp.parallel {
    omp.barrier
    omp.terminator
  }

  llvm.return
}
```

Runtime call to create threads

Callback

```
Location information
@0 = private unnamed_addr constant [23 x i8] c";unknown;unknown;0;0;;\00", align 1
@1 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 34, i32 0, i32 22, ptr @0 }, align 8
@2 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 2, i32 0, i32 22, ptr @0 }, align 8

define void @test_omp_parallel_1() {
  %omp_global_thread_num = call i32 @__kmpc_global_thread_num(ptr @2)
  call void (ptr, i32, ptr, ...)
    @__kmpc_fork_call(ptr @2, i32 0, ptr @test_omp_parallel_1..omp_par)
  ret void
}

define internal void @test_omp_parallel_1..omp_par(ptr noalias %tid.addr,
    ptr noalias %zero.addr) #0 {
  %tid.addr.local = alloca i32, align 4
  %0 = load i32, ptr %tid.addr, align 4
  store i32 %0, ptr %tid.addr.local, align 4
  %tid = load i32, ptr %tid.addr.local, align 4

  ; Concurrently executed code goes here
  %omp_global_thread_num2 = call i32 @__kmpc_global_thread_num(ptr @2)
  call void @__kmpc_barrier(ptr @1, i32 %omp_global_thread_num2)
  ret void
}
```

# Dialects to the Rescue

```
llvm.func @test_omp_parallel_1() -> () {
  omp.parallel {
    omp.barrier
    omp.terminator
  }

  llvm.return
}
```



```
@0 = private unnamed_addr constant [23 x i8] c";unknown;unknown;0;0;;\00", align 1
@1 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 34, i32 0, i32 22, ptr @0 }, align 8
@2 = private unnamed_addr constant %struct.ident_t
     { i32 0, i32 2, i32 0, i32 22, ptr @0 }, align 8

define void @test_omp_parallel_1() {
  %omp_global_thread_num = call i32 @__kmpc_global_thread_num(ptr @2)
  call void (ptr, i32, ptr, ...)
    @__kmpc_fork_call(ptr @2, i32 0, ptr @test_omp_parallel_1..omp_par)
  ret void
}

define internal void @test_omp_parallel_1..omp_par(ptr noalias %tid.addr,
    ptr noalias %zero.addr) #0 {
  %tid.addr.local = alloca i32, align 4
  %0 = load i32, ptr %tid.addr, align 4
  store i32 %0, ptr %tid.addr.local, align 4
  %tid = load i32, ptr %tid.addr.local, align 4

  ; Concurrently executed code goes here
  %omp_global_thread_num2 = call i32 @__kmpc_global_thread_num(ptr @2)
  call void @__kmpc_barrier(ptr @1, i32 %omp_global_thread_num2)
  ret void
}
```

Regions

Outlining

Globals

Runtime library knowledge

# OpenMPIRBuilder to the Rescue

```
$ wc -l llvm/include/llvm/Frontend/OpenMP/* llvm/lib/Frontend/OpenMP/*
     4     llvm/include/llvm/Frontend/OpenMP/CMakeLists.txt
     1346  llvm/include/llvm/Frontend/OpenMP/ClauseT.h
     1189  llvm/include/llvm/Frontend/OpenMP/ConstructDecompositionT.h
     61    llvm/include/llvm/Frontend/OpenMP/OMP.h
     2628  llvm/include/llvm/Frontend/OpenMP/OMP.td
     55    llvm/include/llvm/Frontend/OpenMP/OMPAssume.h
     298   llvm/include/llvm/Frontend/OpenMP/OMPConstants.h
     213   llvm/include/llvm/Frontend/OpenMP/OMPContext.h
     31    llvm/include/llvm/Frontend/OpenMP/OMPDeviceConstants.h
     137   llvm/include/llvm/Frontend/OpenMP/OMPGridValues.h
     3718  llvm/include/llvm/Frontend/OpenMP/OMPIRBuilder.h
     1420  llvm/include/llvm/Frontend/OpenMP/OMPKinds.def
     26    llvm/lib/Frontend/OpenMP/CMakeLists.txt
     234   llvm/lib/Frontend/OpenMP/OMP.cpp
     629   llvm/lib/Frontend/OpenMP/OMPContext.cpp
    10107 llvm/lib/Frontend/OpenMP/OMPIRBuilder.cpp
   22096 total
```

# Accessing Conversion Functionality

```
// This is available to any `convertOperaiton` and callees.
class ModuleTranslation {
public:
  // Returns an OpenMP IR Builder object.
  llvm::OpenMPIRBuilder *getOpenMPBuilder();
```

# Accessing Conversion Functionality

```
// This is available to any `convertOperaiton` and callees.
class ModuleTranslation {
public:
  // Returns an OpenMP IR Builder object.
  llvm::OpenMPIRBuilder *getOpenMPBuilder();

  // Translates an MLIR block into an LLVM IR basic block by
  // calling the interface for every operation.
  LogicalResult convertBlock(Block &bb, bool ignoreArguments,
                             llvm::IRBuilderBase &builder);
};
```

*Note: block arguments become unconnected PHI nodes*

*Entry blocks shouldn't have PHI nodes!*

# Accessing Conversion Functionality

```
// This is available to any `convertOperaiton` and callees.
class ModuleTranslation {
public:
  // Returns an OpenMP IR Builder object.
  llvm::OpenMPIRBuilder *getOpenMPBuilder();

  // Translates an MLIR block into an LLVM IR basic block by
  // calling the interface for every operation.
  LogicalResult convertBlock(Block &bb, bool ignoreArguments,
                             llvm::IRBuilderBase &builder);
};

namespace detail {
  // Connects PHI node targets.
  void connectPHINodes(Region &region, const ModuleTranslation &state);
}
```

*Note: block arguments become unconnected PHI nodes*

*Note: call this to connect PHI nodes to target blocks*

# MLIR Supports Arbitrary nesting

```
omp.parallel   {
  // ...
  omp.wsloop {
    omp.loop_nest (%arg2) : i64 = (%2) to (%1) step (%0) {
      omp.parallel {
        omp.wsloop {
          omp.loop_nest (%arg3) : i64 = (%2) to (%0) step (%0) {
            // ...
            omp.yield
          }
        }
        omp.terminator
      }
      // ...
      omp.yield
    }
  }
  omp.terminator
}
```

# MLIR Supports Arbitrary nesting

```
omp.parallel   {
  // ...
  omp.wsloop {
    omp.loop_nest (%arg2) : i64 = (%2) to (%1) step (%0) {
      omp.parallel {
        omp.wsloop {
          omp.loop_nest (%arg3) : i64 = (%2) to (%0) step (%0) {
            // ...
            omp.yield
          }
        }
        omp.terminator
      }
      // ...
      omp.yield
    }
  }
  omp.terminator
}
```

# MLIR Supports Arbitrary nesting

```
omp.parallel   {
  // ...
  omp.wsloop {
    omp.loop_nest (%arg2) : i64 = (%2) to (%1) step (%0) {
      omp.parallel {
        omp.wsloop {
          omp.loop_nest (%arg3) : i64 = (%2) to (%0) step (%0) {
            // ...
            omp.yield
          }
        }
        omp.terminator
      }
      // ...
      omp.yield
    }
  }
  omp.terminator
}
```

# MLIR Supports Arbitrary nesting

```
omp.parallel   {
  // ...
  omp.wsloop {
    omp.loop_nest (%arg2) : i64 = (%2) to (%1) step (%0) {
      omp.parallel {
        omp.wsloop {
          omp.loop_nest (%arg3) : i64 = (%2) to (%0) step (%0) {
            // ...
            omp.yield
          }
        }
        omp.terminator
      }
      // ...
      omp.yield
    }
  }
  omp.terminator
}
```

# MLIR Supports Arbitrary nesting

```
omp.parallel   {
  // ...
  omp.wsloop {
    omp.loop_nest (%arg2) : i64 = (%2) to (%1) step (%0) {
      omp.parallel {
        omp.wsloop {
          omp.loop_nest (%arg3) : i64 = (%2) to (%0) step (%0) {
            // ...
            omp.yield
          }
        }
        omp.terminator
      }
      // ...
      omp.yield
    }
  }
  omp.terminator
}
```

*Extensible stack frame object*

```
class StackFrame {
 // ...
 public:
   virtual ~StackFrame()
};
```

# MLIR Supports Arbitrary nesting

```
omp.parallel   {
  // ...
  omp.wsloop {
    omp.loop_nest (%arg2) : i64 = (%2) to (%1) step (%0) {
      omp.parallel {
        omp.wsloop {
          omp.loop_nest (%arg3) : i64 = (%2) to (%0) step (%0) {
            // ...
            omp.yield
          }
        }
        omp.terminator
      }
      // ...
      omp.yield
    }
  }
  omp.terminator
}
```

*Extensible stack frame object*

*Works with isa/cast via MLIR TypeID*

```
class StackFrame {
 // ...
 public:
   virtual ~StackFrame()
};


template <typename Derived>
class StackFrameBase : public StackFrame {
  // ...
};
```

# MLIR Supports Arbitrary nesting

```
omp.parallel   {
  // ...
  omp.wsloop {
    omp.loop_nest (%arg2) : i64 = (%2) to (%1) step (%0) {
      omp.parallel {
        omp.wsloop {
          omp.loop_nest (%arg3) : i64 = (%2) to (%0) step (%0) {
            // ...
            omp.yield
          }
        }
        omp.terminator
      }
      // ...
      omp.yield
    }
  }
  omp.terminator
}
```

*Extensible stack frame object*

*Works with isa/cast via MLIR TypeID*

```
class StackFrame {
 // ...
 public:
   virtual ~StackFrame()
};


template <typename Derived>
class StackFrameBase : public StackFrame {
  // ...
};


class OpenMPLoopInfoStackFrame
   : public StackFrameBase<OpenMPLoopInfoStackFrame>
public:
  MLIR_DEFINE_EXPLICIT_INTERNAL_INLINE_TYPE_ID(
    OpenMPLoopInfoStackFrame)
  llvm::CanonicalLoopInfo *loopInfo = nullptr;
};
```

# MLIR Supports Arbitrary nesting

```
// This is available to any `convertOperaiton` and callees.
class ModuleTranslation {
public:

  // RAII object to be created before translating a nested
  // region from MLIR to LLVM IR.
  template <typename T>
  struct SaveStack {
    template <typename... Args>
    explicit SaveStack(ModuleTranslation &m, Args &&...args);
  };
```

```
class StackFrame {
 // ...
 public:
   virtual ~StackFrame()
};


template <typename Derived>
class StackFrameBase : public StackFrame {
  // ...
};


class OpenMPLoopInfoStackFrame
   : public StackFrameBase<OpenMPLoopInfoStackFrame>
public:
  MLIR_DEFINE_EXPLICIT_INTERNAL_INLINE_TYPE_ID(
    OpenMPLoopInfoStackFrame)
  llvm::CanonicalLoopInfo *loopInfo = nullptr;
};
```

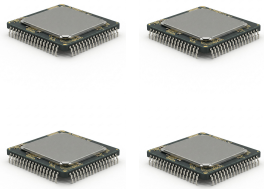
# MLIR Supports Arbitrary nesting

```
// This is available to any `convertOperaiton` and callees.
class ModuleTranslation {
public:

  // RAII object to be created before translating a nested
  // region from MLIR to LLVM IR.
  template <typename T>
  struct SaveStack {
    template <typename... Args>
    explicit SaveStack(ModuleTranslation &m, Args &&...args);
  };

  // Visitor for stack frames of a certain kind: they can
  // be interleaved freely.
  template <typename T>
  WalkResult stackWalk(
    function_ref<WalkResult(T &)> callback);
};
```

```
class StackFrame {
 // ...
 public:
   virtual ~StackFrame()
};


template <typename Derived>
class StackFrameBase : public StackFrame {
  // ...
};


class OpenMPLoopInfoStackFrame
   : public StackFrameBase<OpenMPLoopInfoStackFrame>
public:
  MLIR_DEFINE_EXPLICIT_INTERNAL_INLINE_TYPE_ID(
    OpenMPLoopInfoStackFrame)
  llvm::CanonicalLoopInfo *loopInfo = nullptr;
};
```

```
@_ZN4mlir2OpINS_4LLVM10LLVMFuncOpEJNS_7OpTrait9OneRegionENS3_11ZeroResultsENS3_14ZeroSuccessorsENS3_12ZeroOperandsENS3_12O
pInvariantsENS_19BytecodeOpInterface5TraitENS3_24AutomaticAllocationScopeENS3_19IsIsolatedFromAboveENS_17SymbolOpInterface
5TraitENS_19CallableOpInterface5TraitENS_19FunctionOpInterface5TraitEEE13getPropertiesIS2_EERNS_18PropertiesSelectorIT_vE4
typeEv(ptr noundef %4)
 %6 = getelementptr inbounds %"struct.mlir::LLVM::detail::LLVMFuncOpGenericAdaptorBase::Properties", ptr %5, i32 0, i32 25
 %7 = call i64 @_ZN4llvm16dyn_cast_or_nullIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %6)
 %8 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %9 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %8, i32 0, i32 0
 %10 = getelementptr inbounds %"class.mlir::Attribute", ptr %9, i32 0, i32 0
 %11 = inttoptr i64 %7 to ptr
 store ptr %11, ptr %10, align 8
 %12 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %13 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %12, i32 0, i32 0
 %14 = getelementptr inbounds %"class.mlir::Attribute", ptr %13, i32 0, i32 0
 %15 = load ptr, ptr %14, align 8
 %16 = ptrtoint ptr %15 to i64
 ret i64 %16
}

; Function Attrs: noinline optnone ssp uwtable(sync)
define linkonce_odr ptr @_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEC1ES1_NS_10ModRefInfoE(ptr noundef returned %0,
i32 noundef %1, i8 noundef zeroext %2) unnamed_addr #4 align 2 {
 %4 = alloca ptr, align 8
 %5 = alloca i32, align 4
 %6 = alloca i8, align 1
 store ptr %0, ptr %4, align 8
 store i32 %1, ptr %5, align 4
 store i8 %2, ptr %6, align 1
 %7 = load ptr, ptr %4, align 8
 %8 = load i32, ptr %5, align 4
 %9 = load i8, ptr %6, align 1
 %10 = call ptr @_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEC2ES1_NS_10ModRefInfoE(ptr noundef %7, i32 noundef %8, i8
noundef zeroext %9)
 ret ptr %7
}

; Function Attrs: mustprogress noinline nounwind optnone ssp uwtable(sync)
define linkonce_odr noundef nonnull align 4 dereferenceable(4) ptr
@_ZN4llvm17MemoryEffectsBaseINS_13IRMemLocationEEoRES2_(ptr noundef %0, i64 %1) #1 align 2 {
 %3 = alloca %"class.llvm::MemoryEffectsBase", align 4
 %4 = alloca ptr, align 8
 %5 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %3, i32 0, i32 0
 %6 = trunc i64 %1 to i32
 store i32 %6, ptr %5, align 4
 store ptr %0, ptr %4, align 8
 %7 = load ptr, ptr %4, align 8
 %8 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %3, i32 0, i32 0
 %9 = load i32, ptr %8, align 4
 %10 = getelementptr inbounds %"class.llvm::MemoryEffectsBase", ptr %7, i32 0, i32 0
 %11 = load i32, ptr %10, align 4
 %12 = or i32 %11, %9
 store i32 %12, ptr %10, align 4
 ret ptr %7
}

; Function Attrs: mustprogress noinline nounwind optnone ssp uwtable(sync)
define linkonce_odr hidden zeroext i1
@_ZNKSt3__123__optional_storage_baseIN4mlir4LLVM17MemoryEffectsAttrELb0EE9has_valueB8ue170006Ev(ptr noundef %0) #1 align 2
{
 %2 = alloca ptr, align 8
 store ptr %0, ptr %2, align 8
 %3 = load ptr, ptr %2, align 8
 %4 = getelementptr inbounds %"struct.std::__1::__optional_destruct_base.1739", ptr %3, i32 0, i32 1
 %5 = load i8, ptr %4, align 8
 %6 = trunc i8 %5 to i1
 ret i1 %6
}

; Function Attrs: mustprogress noinline optnone ssp uwtable(sync)
define linkonce_odr i64 @_ZN4llvm16dyn_cast_or_nullIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %0) #0 {
 %2 = alloca %"class.mlir::LLVM::MemoryEffectsAttr", align 8
 %3 = alloca ptr, align 8
 store ptr %0, ptr %3, align 8
 %4 = load ptr, ptr %3, align 8
 %5 = call i64 @_ZN4llvm19dyn_cast_if_presentIN4mlir4LLVM17MemoryEffectsAttrES3_EEDaRT0_(ptr noundef nonnull align 8
dereferenceable(8) %4)
 %6 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
 %7 = getelementptr inbounds %"class.mlir::detail::StorageUserBase.1694", ptr %6, i32 0, i32 0
 %8 = getelementptr inbounds %"class.mlir::Attribute", ptr %7, i32 0, i32 0
 %9 = inttoptr i64 %5 to ptr
 store ptr %9, ptr %8, align 8
 %10 = getelementptr inbounds %"class.mlir::LLVM::MemoryEffectsAttr", ptr %2, i32 0, i32 0
```

clang
llc




Nothing to do!
OpenACC

# Translation Summary

Dialects can implement `LLVMTranslationDialectInterface` to convert operations and attributes they contain.

Implementations can use the `ModuleTranslation` object to query the state of the translation.

They can also trigger translation recursively for nested regions and manage customizable stack frames.

No support for custom type translation.

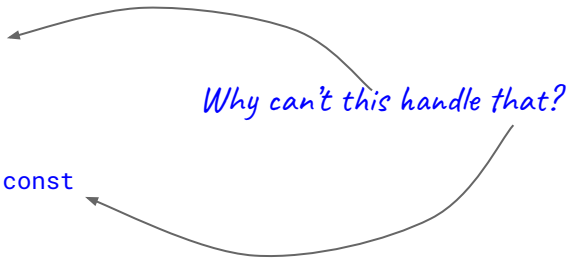
# Is It General Enough?

```
for f in functions:
 signatures[f] = translate_signature(f)

for g in globals:
 translate_global(g)

# same for certain kinds of metadata...

for f in functions:
 for b in f.body.blocks:
   blocks += create_block(b)
   for o in without_terminator(b.operations):
     translate_operation(o)
 connect_blocks_via_phis(blocks)
```

*Why do these exist?*

# Is It General Enough

```
class LLVMTranslationDialectInterface : public DialectInterface::Base<...> {
public:
 // ...

 virtual LogicalResult
 convertOperation(...) const

 virtual LogicalResult
 amendOperation(...) const

 virtual LogicalResult
 convertParameterAttr(...) const
```



# The Cost of Legacy Design

Commit `ac6bfa6`

**ftynse** authored and **jpienaar** committed on Mar 29, 2019

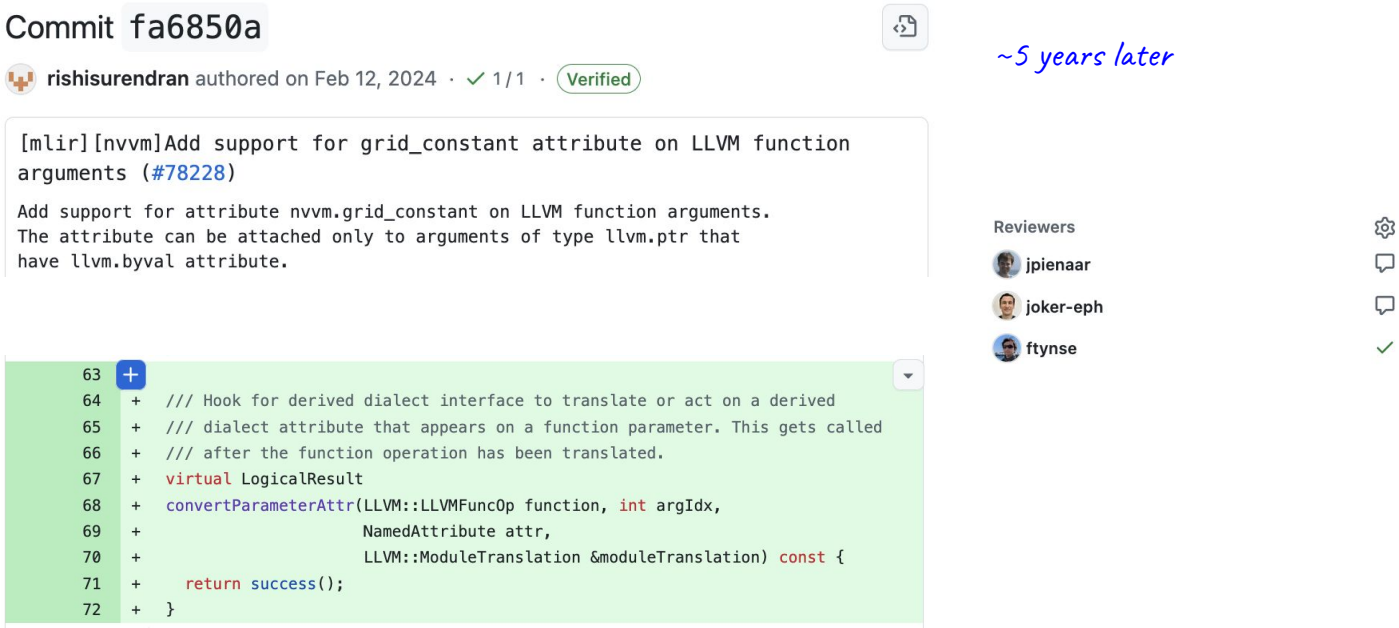
*This date is wrong, btw!*

```
Lower scalar parts of CFG functions to LLVM IR

Initial restricted implementaiton of the MLIR to LLVM IR translation.
Introduce a new flow into the mlir-translate tool taking an MLIR module
containing CFG functions only and producing and LLVM IR module.  The MLIR
features supported by the translator are as follows:
- primitive and function types;
- integer constants;
- cfg and ext functions with 0 or 1 return values;
- calls to these functions;
- basic block conversion translation of arguments to phi nodes;
- conversion between arguments of the first basic block and function
arguments;
- (conditional) branches;
- integer addition and comparison operations.
```

*convertFunctions is introduced*

```
52  + private:
53  +   bool convertBasicBlock(const BasicBlock &bb, bool ignoreArguments = false);
54  +   bool convertCFGFunction(const CFGFunction &cfgFunc, llvm::Function &llvmFunc);
55  +   bool convertFunctions(const Module &mlirModule, llvm::Module &llvmModule);
56  +   bool convertInstruction(const Instruction &inst);
57  +
```

# The Cost of Legacy Design




~5 years later

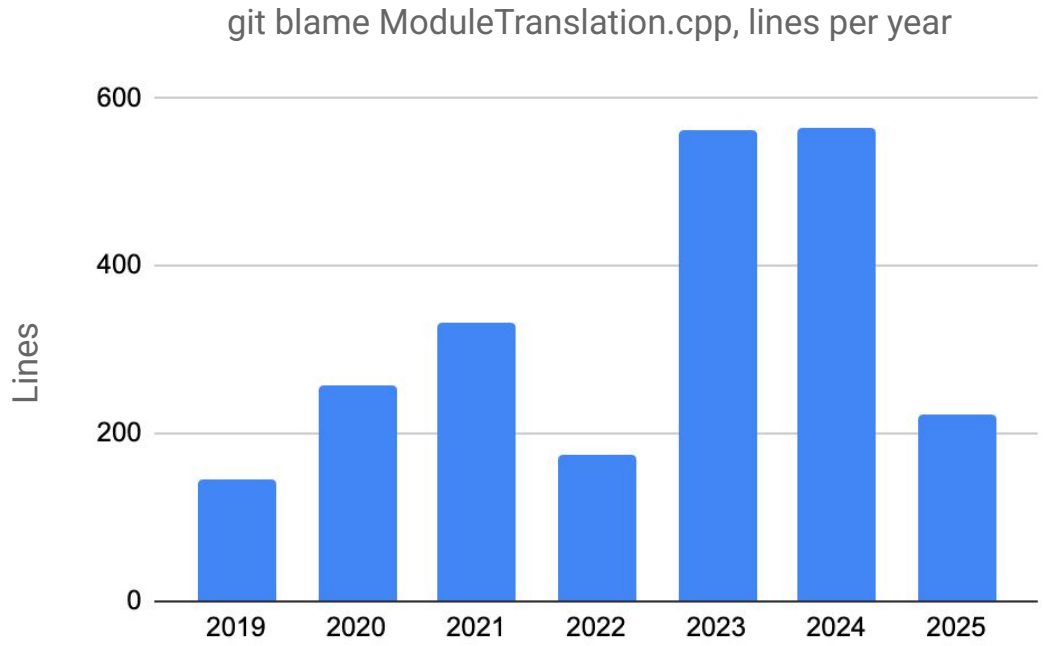
Reviewers

jpienaar

joker-eph

ftynse

# Legacy Design != Legacy Code



A lot of design decisions in MLIR, especially around LLVM connection, date back to early days.

Yet we keep adhering to them.

# The Monkey Ladder Experiment

*I have found no scientific paper that actually reports on such an experiment

# The Monkey Ladder Experiment

*I have found no scientific paper that actually reports on such an experiment

# Call to Action

Easy: take 5% more time to try and understand the rationale behind certain design (or the lack thereof).

- Do a git blame / log of relevant files. Check forums / mailing lists around those dates.
- Ask around, assume good intention.

# Call to Action

Medium: document new design decisions better.

- RFCs are 100-s of posts and the summary is rarely stated
- Summarize design in comments or commit messages
- Even without RFCs
- Consider committing “accepted” RFCs into git

# Call to Action

Hard: challenge legacy design decisions that no longer make sense.

- Acknowledge it may have made sense at the time!
- Devise and clearly describe a migration strategy.

All translation from MLIR to LLVM IR must go through the LLVM dialect to keep translation simple!

# Understand the Rationale

LLVM dialect design objective: reflect LLVM IR in MLIR as closely as possible while using MLIR core abstractions.

*But we have multiple dialects now!*

*Yes, we should make the dependency on libFrontendOpenMP optional!*

Corollary: ~~LLVM~~ dialect to LLVM IR translation ~~is straightforward and the infrastructure is simple.~~ *"pay only for what you use".*

*Recursion with implicit state, explicit stack frames, templates, dyn_cast...*

New corollary: introduce a dialect when it is worth it

# To Transform or Not To Transform

~~Forcing factor for MLIR development~~

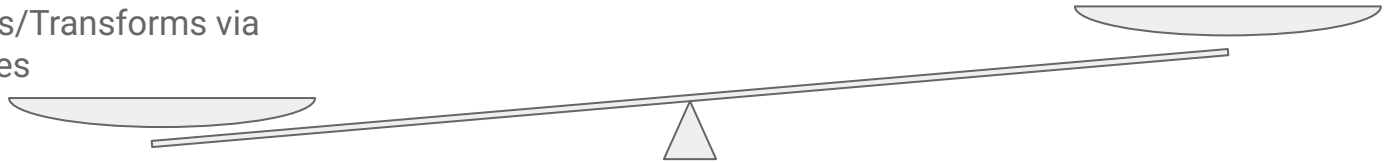
MLIR Context supports concurrency natively

Operations > Intrinsics

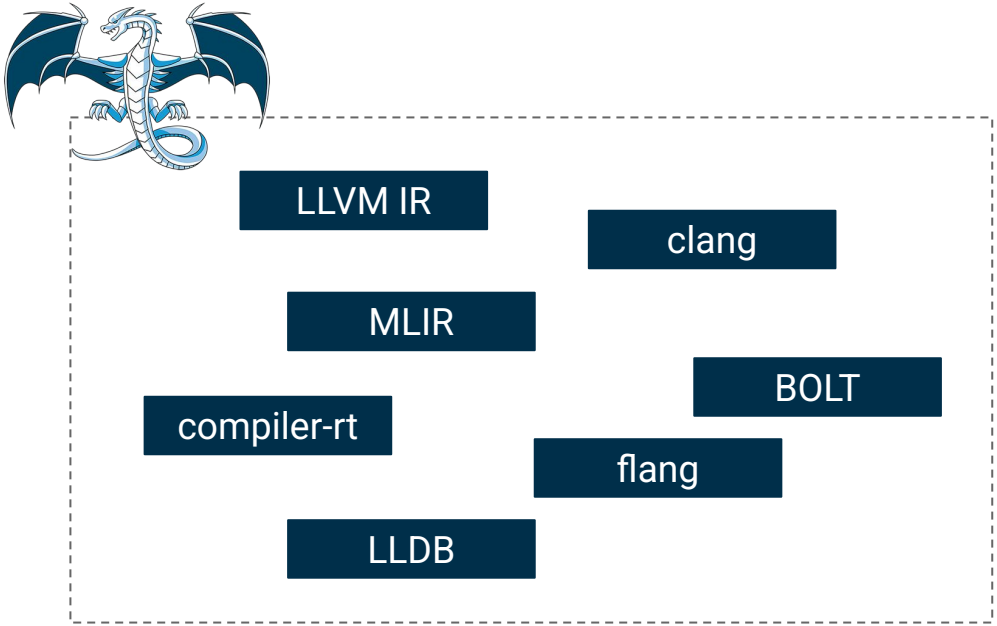
Analyses/Transforms via Interfaces

Extra dialects increase cost for everyone

MLIR manipulation is 3-4x slower than LLVM IR on average*



Is it worth it having an extra dialect + conversion to the LLVM dialect?




MLIR is part of the LLVM project

# Design Decisions

Understand

Document

Challenge

*Don't forget to remain friendly! Not everyone is a dragon here!*