# Global Instruction Selection

## Status

Ahmed Bougacha
Quentin Colombet
Tim Northover

LLVM DevMeeting
November 2016
San Jose, CA

2015 Recap

# Global ISel Recap

Initial Proposal:
http://lists.llvm.org/pipermail/llvm-dev/2015-November/092566.html

# Overview

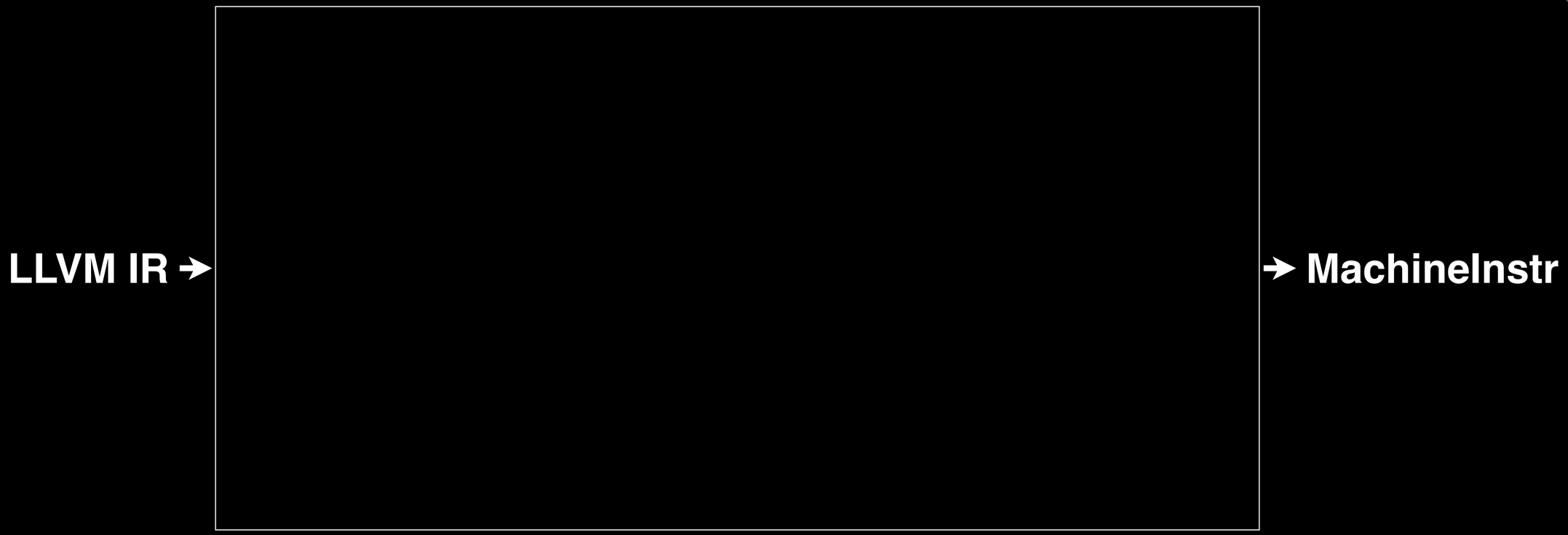
## SelectionDAG

2015 Recap

# Overview

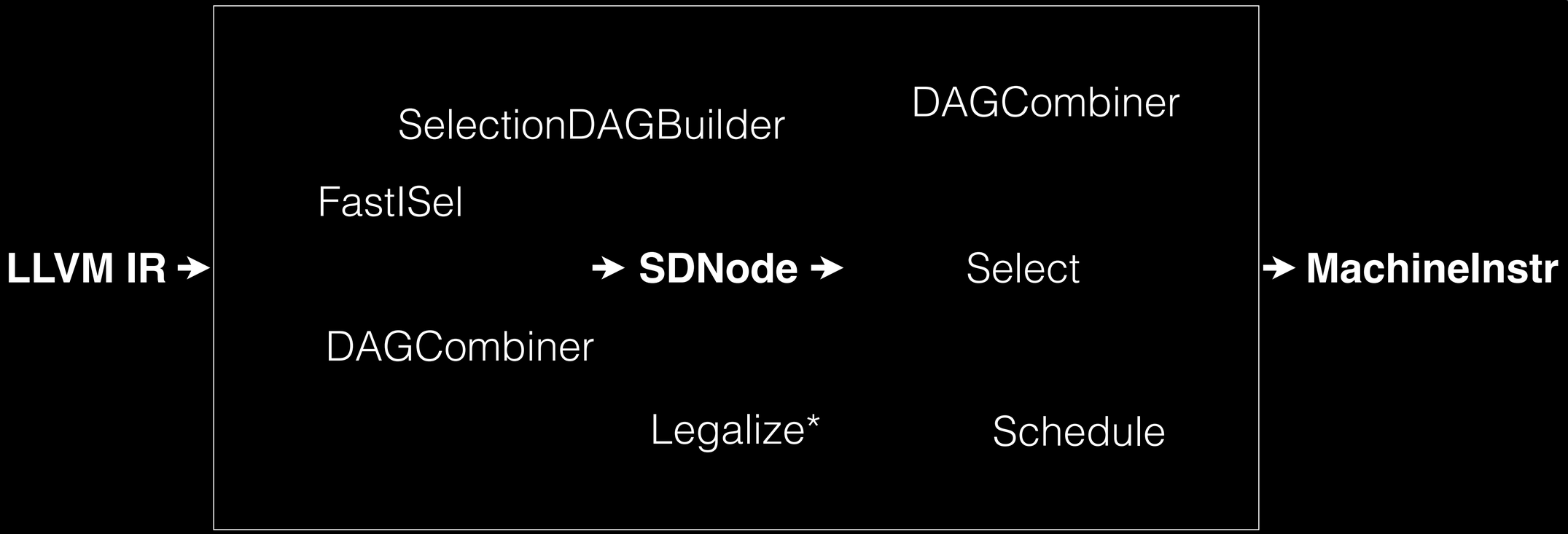
## SelectionDAG

2015 Recap



# Overview
## SelectionDAG





# Overview

## Global ISel

2015 Recap

# Overview

## Global ISel

2015 Recap


**Generic MachineInstr**

**MachineInstr**

2015 Recap


LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Instruction Select
MI

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

- LLVM IR to generic (G) MachineInstr

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%2(_,s1) = G_LOAD %0(_,p0)
```

- LLVM IR to generic (G) MachineInstr

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%2(_,s1) = G_LOAD %0(_,p0)
```

- LLVM IR to generic (G) MachineInstr:
  G_ADD, G_PTRTOINT

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%2(_,s1) = G_LOAD %0(_,p0)
```

- LLVM IR to generic (G) MachineInstr
- Virtual registers have a type

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%2(_,s1) = G_LOAD %0(_,p0)
```

- LLVM IR to generic (G) MachineInstr
- Virtual registers have a type:
  LowLevelType (LLT): Replacement of EVT

| | | |
|---|---|---|
| Scalar: | s#bit | `s8, s32` |
| Vector: | <#lane x s#bit> | `<2 x s8>, <3 x s48>` |
| Pointer: | p#addrspace | `p0, p256` |

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%2(_,s1) = G_LOAD %0(_,p0)
```

- LLVM IR to generic (G) MachineInstr
- Virtual registers have a type
- Virtual registers might not have a register class

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%2(_,s1) = G_LOAD %0(_,p0)
```

- LLVM IR to generic (G) MachineInstr
- Virtual registers have a type
- Virtual registers might not have a register class
- ABI lowering

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%2(_,s1) = G_LOAD %0(_,p0)



%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

- LLVM IR to generic (G) MachineInstr
- Virtual registers have a type
- Virtual registers might not have a register class
- ABI lowering

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%2(_,s1) = G_LOAD %0(_,p0)




%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

- LLVM IR to generic (G) MachineInstr
- Virtual registers have a type
- Virtual registers might not have a register class
- ABI lowering

# IRTranslator



```
define i64 @foo(i1* %addr1,
                <2 x i32>* %addr2) {
  %tmp0 = load i1, i1* %addr1
  %tmp1 = zext i1 %tmp0 to i64
  %tmp2 = bitcast i64 %tmp1 to <2 x i32>
  %tmp3 = load <2 x i32>, <2 x i32>* %addr2
  %tmp4 = or <2 x i32> %tmp2, %tmp3
  %tmp5 = bitcast <2 x i32> %tmp4 to i64
  ret i64 %tmp5
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%2(_,s1) = G_LOAD %0(_,p0)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

- LLVM IR to generic (G) MachineInstr
- Virtual registers have a type
- Virtual registers might not have a register class
- ABI lowering

2015 Recap


LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Instruction Select
MI

2015 Recap


LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Instruction Select
MI

# Legalizer



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%2(_,s1) = G_LOAD %0(_,p0)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

# Legalizer



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%2(_,s1) = G_LOAD %0(_,p0)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Illegal (G)MachineInstr to legal (G)MachineInstr

# Legalizer



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%2(_,s1) = G_LOAD %0(_,p0)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Illegal (G)MachineInstr to legal (G)MachineInstr

# Legalizer



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%8(_,s8) = G_LOAD %0(_,p0)
%2(_,s1) = G_TRUNC %8(_,s8)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Illegal (G)MachineInstr to legal (G)MachineInstr

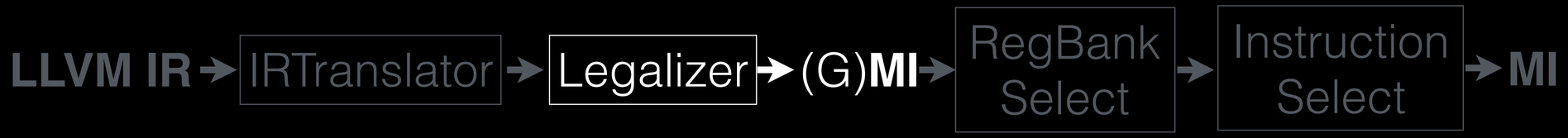
# Legalizer



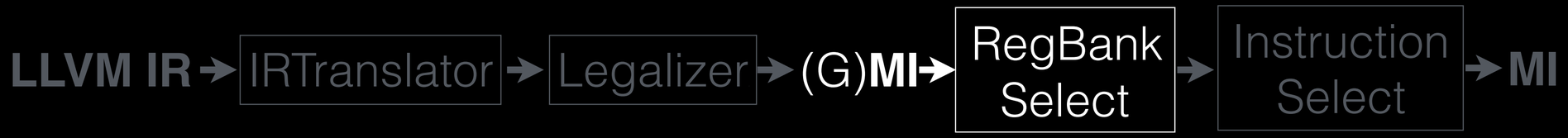
```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%8(_,s8) = G_LOAD %0(_,p0)
%2(_,s1) = G_TRUNC %8(_,s8)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Illegal (G)MachineInstr to legal (G)MachineInstr

2015 Recap


LLVM IR
IRTranslator
Legalizer
(G)MI
RegBank Select
Instruction Select
MI

2015 Recap


LLVM IR
IRTranslator
Legalizer
(G)MI
RegBank Select
Instruction Select
MI

# RegBankSelect



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%8(_,s8) = G_LOAD %0(_,p0)
%2(_,s1) = G_TRUNC %8(_,s8)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

# RegBankSelect



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%8(_,s8) = G_LOAD %0(_,p0)
%2(_,s1) = G_TRUNC %8(_,s8)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

# RegBankSelect



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%8(_,s8) = G_LOAD %0(_,p0)
%2(_,s1) = G_TRUNC %8(_,s8)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

# RegBankSelect



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%8(_,s8) = G_LOAD %0(_,p0)
%2(_,s1) = G_TRUNC %8(_,s8)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(_,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

# RegBankSelect



```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%8(_,s8) = G_LOAD %0(_,p0)
%2(_,s1) = G_TRUNC %8(_,s8)
%3(_,s64) = G_ZEXT %2(_,s1)
%4(_,<2 x s32>) = G_BITCAST %3(_,s64)
%5(FPR,<2 x s32>) = G_LOAD %1(_,p0)
%6(_,<2 x s32>) = G_OR %4, %5
%7(_,s64) = G_BITCAST %6(_,<2 x s32>)
%x0 = COPY %7(_,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

# RegBankSelect



```
%0(GPR,p0) = COPY %x0
%1(GPR,p0) = COPY %x1
%8(GPR,s8) = G_LOAD %0(GPR,p0)
%2(GPR,s1) = G_TRUNC %8(GPR,s8)
%3(GPR,s64) = G_ZEXT %2(GPR,s1)
%4(FPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(FPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(FPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(FPR,<2 x s32>)
%x0 = COPY %7(GPR,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

# RegBankSelect



```
%0(GPR,p0) = COPY %x0
%1(GPR,p0) = COPY %x1
%8(GPR,s8) = G_LOAD %0(GPR,p0)
%2(GPR,s1) = G_TRUNC %8(GPR,s8)
%3(GPR,s64) = G_ZEXT %2(GPR,s1)
%4(FPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(FPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(FPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(FPR,<2 x s32>)
%x0 = COPY %7(GPR,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

# RegBankSelect



```
%0(GPR,p0) = COPY %x0
%1(GPR,p0) = COPY %x1
%8(GPR,s8) = G_LOAD %0(GPR,p0)
%2(GPR,s1) = G_TRUNC %8(GPR,s8)
%3(GPR,s64) = G_ZEXT %2(GPR,s1)
%4(GPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(GPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(GPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(GPR,<2 x s32>)
%x0 = COPY %7(GPR,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

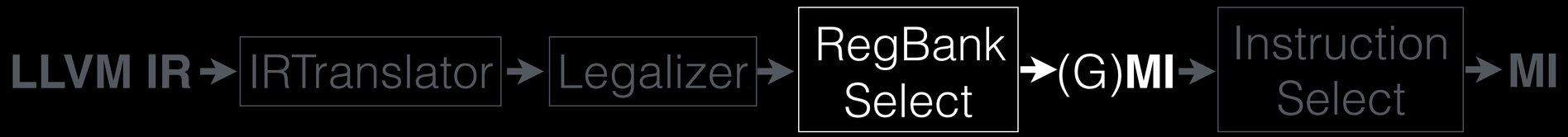
# RegBankSelect



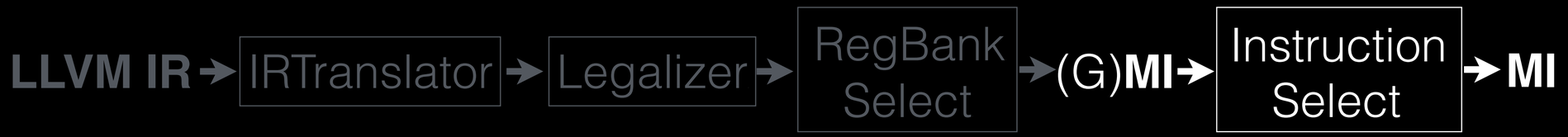
```
%0(GPR,p0) = COPY %x0
%1(GPR,p0) = COPY %x1
%8(GPR,s8) = G_LOAD %0(GPR,p0)
%2(GPR,s1) = G_TRUNC %8(GPR,s8)
%3(GPR,s64) = G_ZEXT %2(GPR,s1)
%4(GPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(GPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(GPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(GPR,<2 x s32>)
%x0 = COPY %7(GPR,s64)
RET_ReallyLR implicit %x0
```

Assigns register banks

2015 Recap


LLVM IR
IRTranslator
Legalizer
RegBank Select
(G)MI
Instruction Select
MI

2015 Recap


LLVM IR
IRTranslator
Legalizer
RegBank Select
(G)MI
Instruction Select
MI

# InstructionSelect



```
%0(GPR,p0) = COPY %x0
%1(GPR,p0) = COPY %x1
%8(GPR,s8) = G_LOAD %0(GPR,p0)
%2(GPR,s1) = G_TRUNC %8(GPR,s8)
%3(GPR,s64) = G_ZEXT %2(GPR,s1)
%4(FPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(FPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(FPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(FPR,<2 x s32>)
%x0 = COPY %7(GPR,s64)
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

# InstructionSelect



```
%0(GPR,p0) = COPY %x0
%1(GPR,p0) = COPY %x1
%8(GPR,s8) = G_LOAD %0(GPR,p0)
%2(GPR,s1) = G_TRUNC %8(GPR,s8)
%3(GPR,s64) = G_ZEXT %2(GPR,s1)
%4(FPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(FPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(FPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(FPR,<2 x s32>)
%x0 = COPY %7(GPR,s64)
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

# InstructionSelect



```
%0(GPR,p0) = COPY %x0
%1(GPR,p0) = COPY %x1
%8 = LDRBBui %0, 0
%2(GPR,s1) = G_TRUNC %8(GPR,s8)
%3(GPR,s64) = G_ZEXT %2(GPR,s1)
%4(FPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(FPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(FPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(FPR,<2 x s32>)
%x0 = COPY %7(GPR,s64)
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

# InstructionSelect



```
%0 = COPY %x0
%1 = COPY %x1
%8 = LDRBBui %0, 0
%2 = COPY %8
%9 = SUBREG_TO_REG 0, %2, 15
%3 = UBFMXri %9, 0, 0
%4 = COPY %3
%5 = LDRDui %1, 0
%6 = ORRv8i8 %4, %5
%7 = COPY %6
%x0 = COPY %7
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

# InstructionSelect



```
%0 = COPY %x0
%1 = COPY %x1
%8 = LDRBBui %0, 0
%2 = COPY %8
%9 = SUBREG_TO_REG 0, %2, 15
%3 = UBFMXri %9, 0, 0
%4 = COPY %3
%5 = LDRDui %1, 0
%6 = ORRv8i8 %4, %5
%7 = COPY %6
%x0 = COPY %7
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

# InstructionSelect



```
%0 = COPY %x0
%1 = COPY %x1
%8 = LDRBBui %0, 0
%2 = COPY %8
%9 = SUBREG_TO_REG 0, %2, 15
%3 = UBFMXri %9, 0, 0
%4(FPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(FPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(FPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(FPR,<2 x s32>)
%x0 = COPY %7
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

# InstructionSelect



```
%0 = COPY %x0
%1 = COPY %x1
%8 = LDRBBui %0, 0
%2 = COPY %8
%9 = SUBREG_TO_REG 0, %2, 15
%3 = UBFMXri %9, 0, 0
%4(GPR,<2 x s32>) = G_BITCAST %3(GPR,s64)
%5(GPR,<2 x s32>) = G_LOAD %1(GPR,p0)
%6(GPR,<2 x s32>) = G_OR %4, %5
%7(GPR,s64) = G_BITCAST %6(GPR,<2 x s32>)
%x0 = COPY %7
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

# InstructionSelect



```
%0 = COPY %x0
%1 = COPY %x1
%8 = LDRBBui %0, 0
%2 = COPY %8
%9 = SUBREG_TO_REG 0, %2, 15
%3 = UBFMXri %9, 0, 0
%4 = COPY %3
%5 = LDRXui %1, 0
%6 = ORRXrr %4, %5
%7 = COPY %6
%x0 = COPY %7
RET_ReallyLR implicit %x0
```

Generic MachineInstr to MachineInstr

2015 Recap

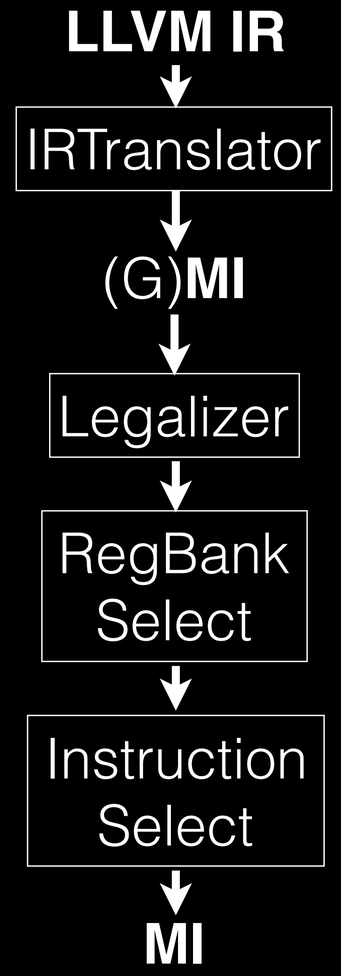

LLVM IR
IRTranslator
Legalizer
RegBank Select
Instruction Select
MI

2015 Recap


LLVM IR
IRTranslator
Legalizer
RegBank Select
Instruction Select
MI

# Global ISel Recap

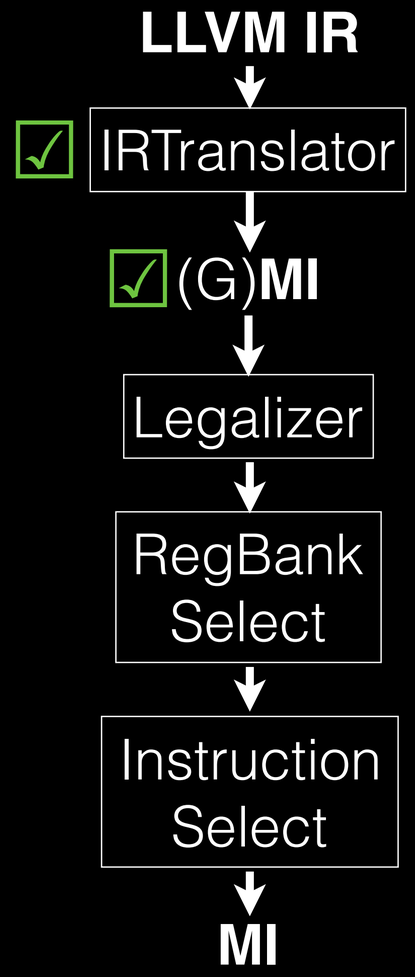
## Initial Plan & Prototype Status



# Global ISel Recap

## Initial Plan & Prototype Status

1. Proof-of-concept
   - ☑ Perform the translation
   - ☑ Lower the ABI
   - ☑ Support simple instructions

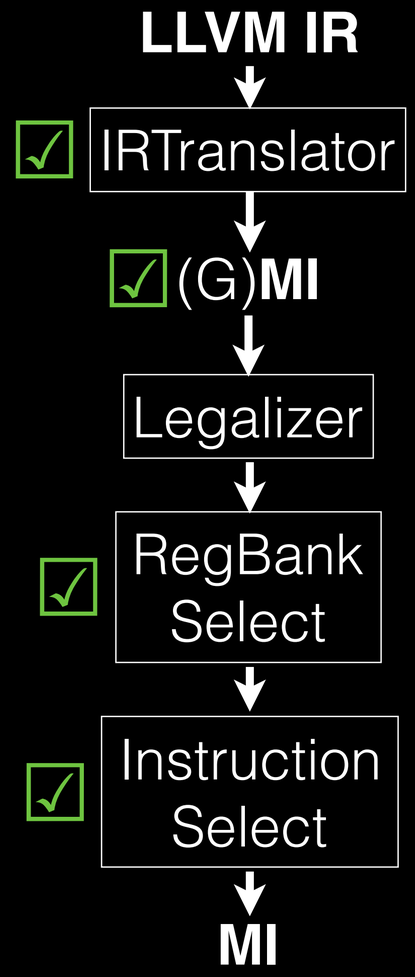


# Global ISel Recap

## Initial Plan & Prototype Status

1. Proof-of-concept
   - ☑ Perform the translation
   - ☑ Lower the ABI
   - ☑ Support simple instructions

2. Basic selector
   - ☑ Abort or fallback to SDAG on illegal ops
   - ☑ Selector patterns written in C++
   - ☑ Simple bank selection



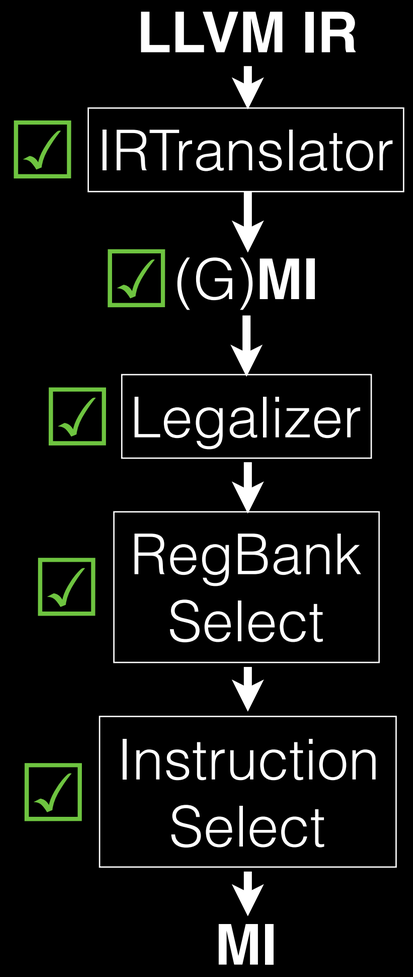
# Global ISel Recap

## Initial Plan & Prototype Status

1. Proof-of-concept
   - ☑ Perform the translation
   - ☑ Lower the ABI
   - ☑ Support simple instructions

2. Basic selector
   - ☑ Abort or fallback to SDAG on illegal ops
   - ☑ Selector patterns written in C++
   - ☑ Simple bank selection

3. Simple legalization
   - ☑ Scalar operations
   - ☑ Some vector operations

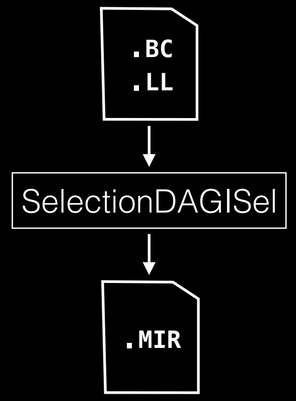


# Global ISel In Depth

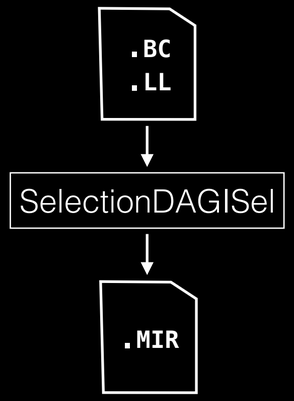
# Global ISel In Depth

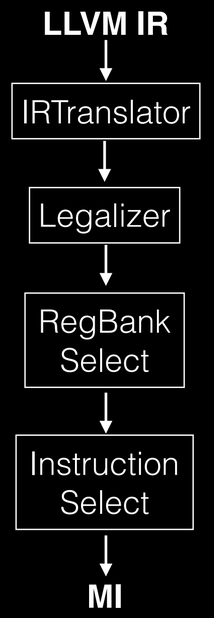
## Testability

# Testability

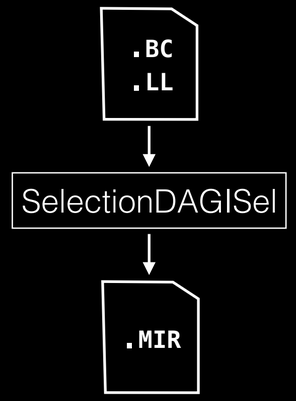


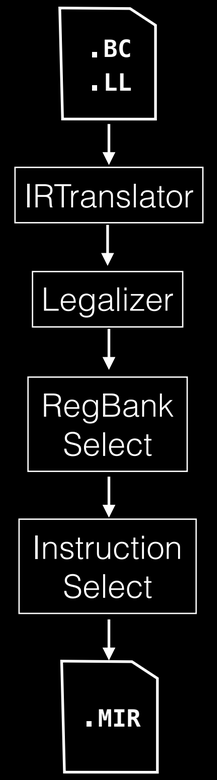
# Testability



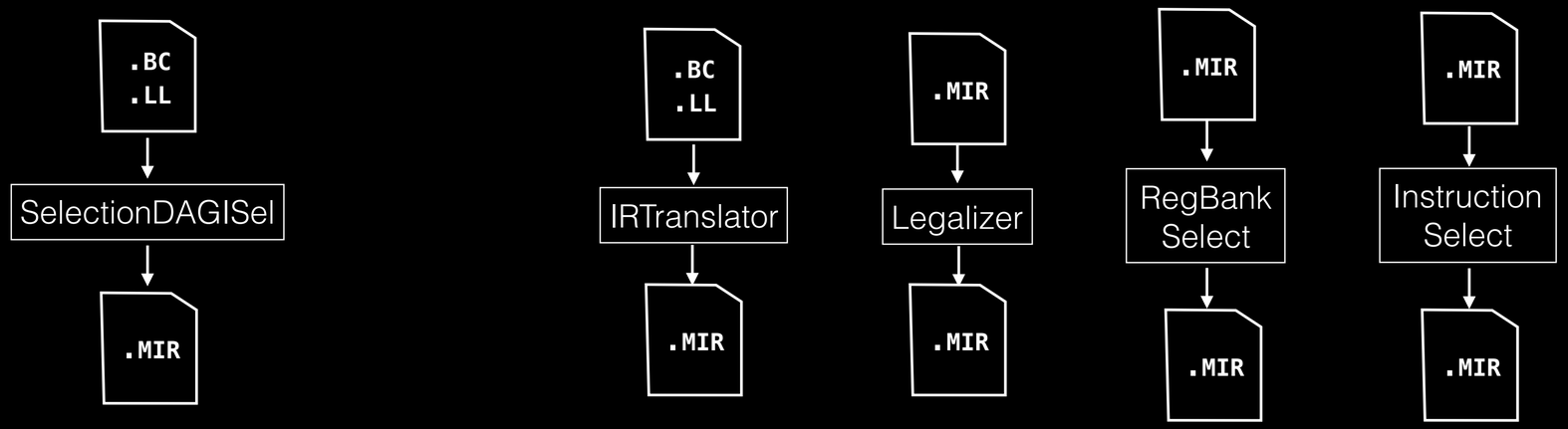



# Testability



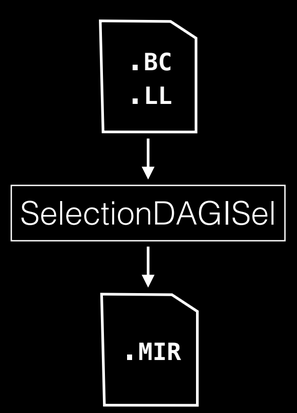



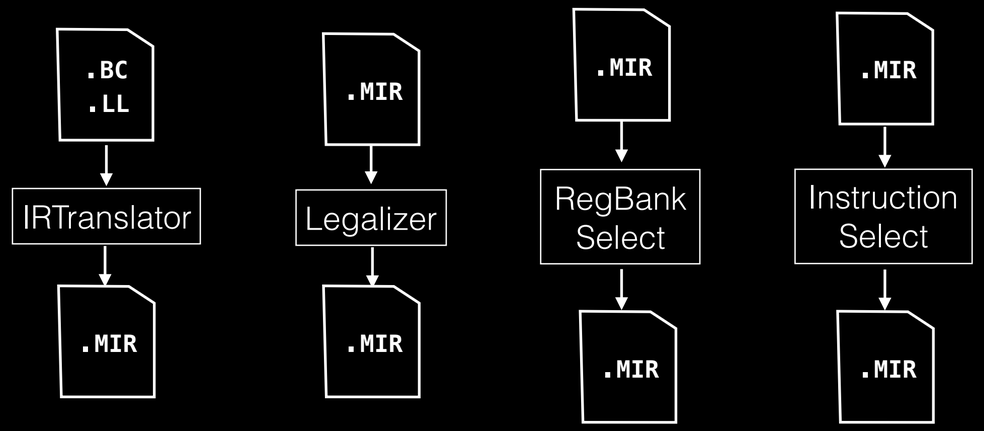
# Testability




Each pass is individually testable

# Testability

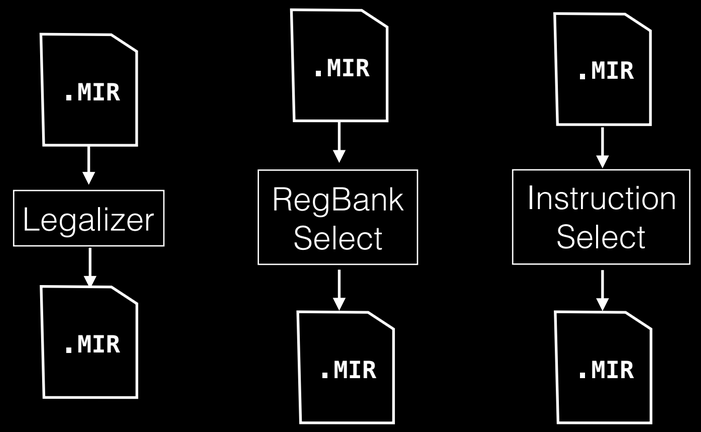



```
$ llc -run-pass <PassName>
```

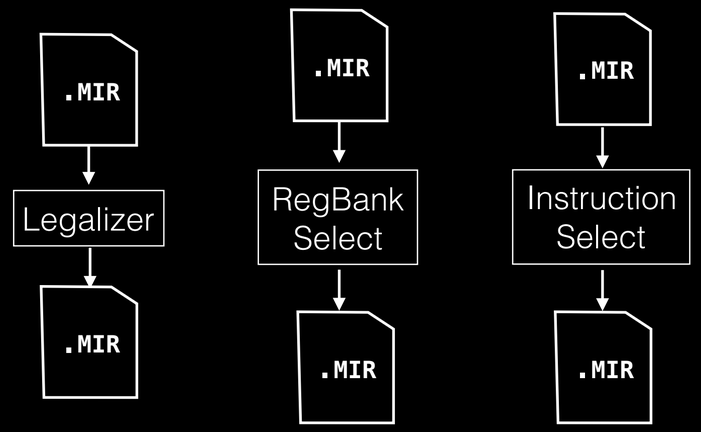
Each pass is individually testable

# Testability



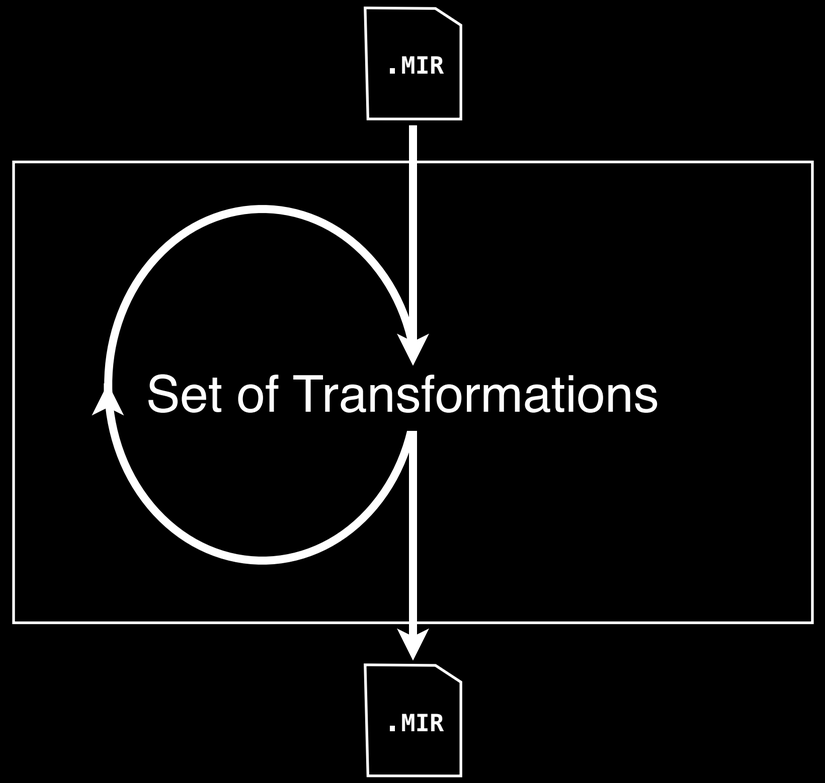

Each pass is individually testable

# Testability

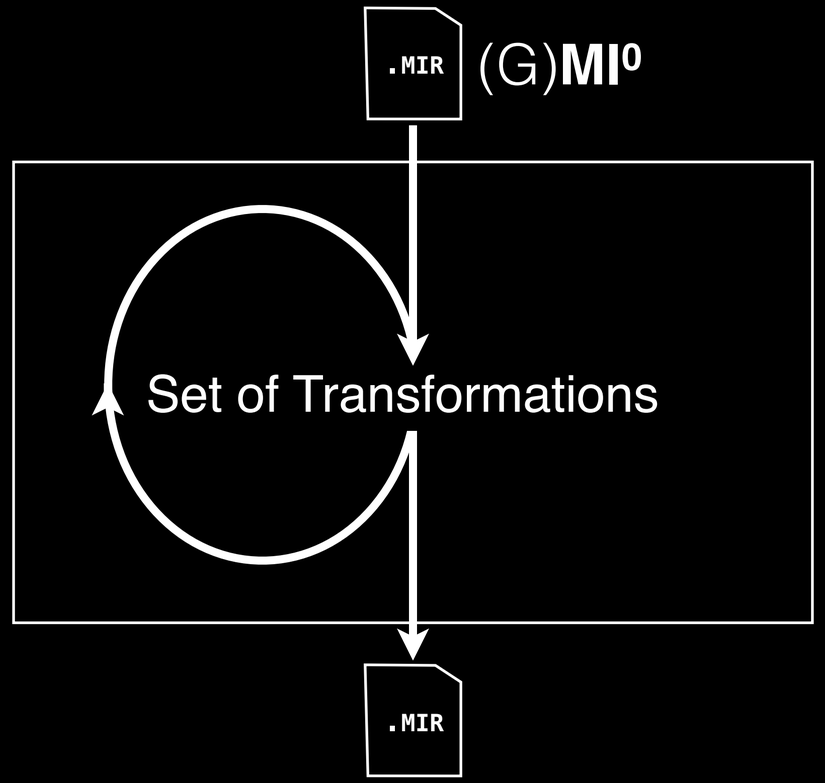



Each pass is individually testable

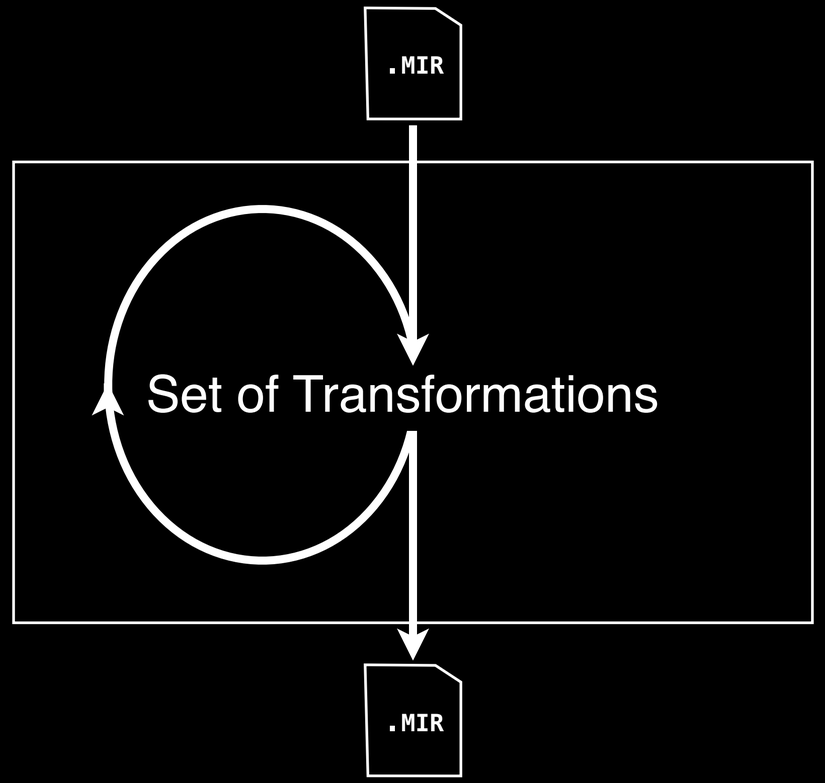
# Testability



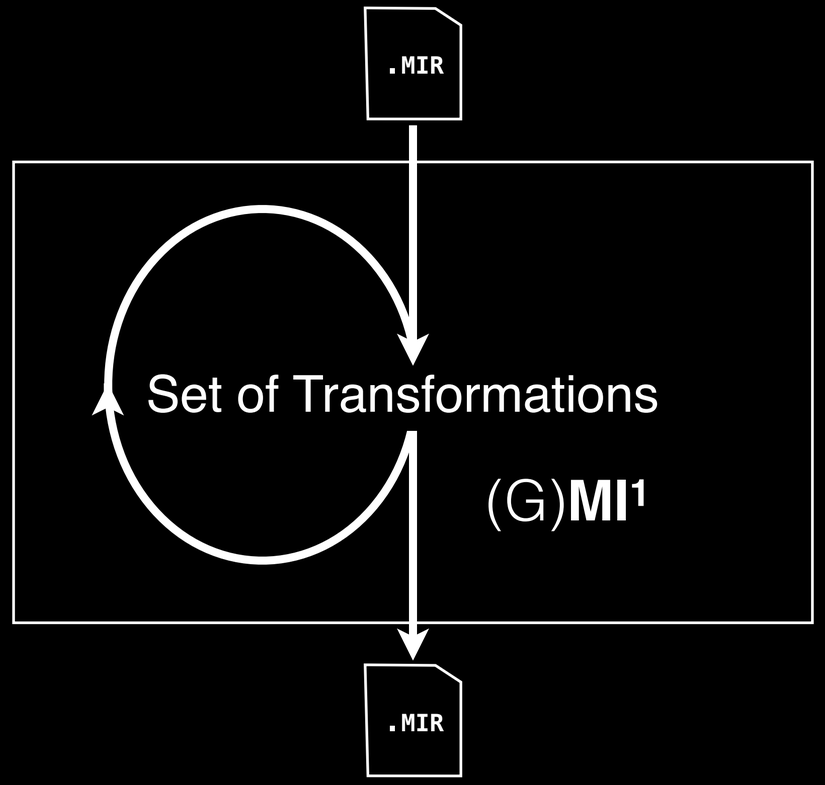
# Testability



# Testability

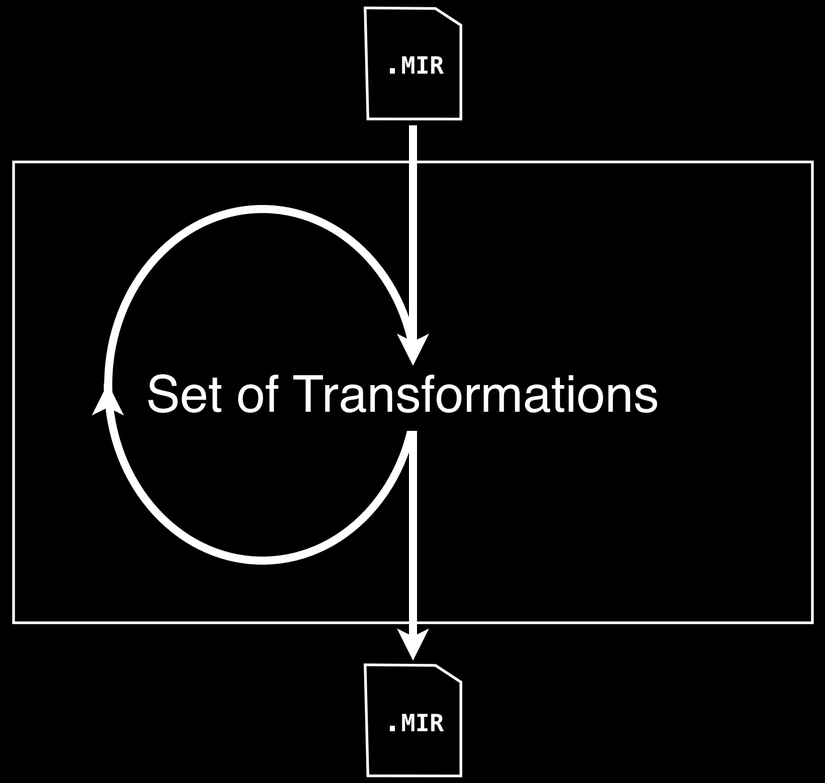


# Testability



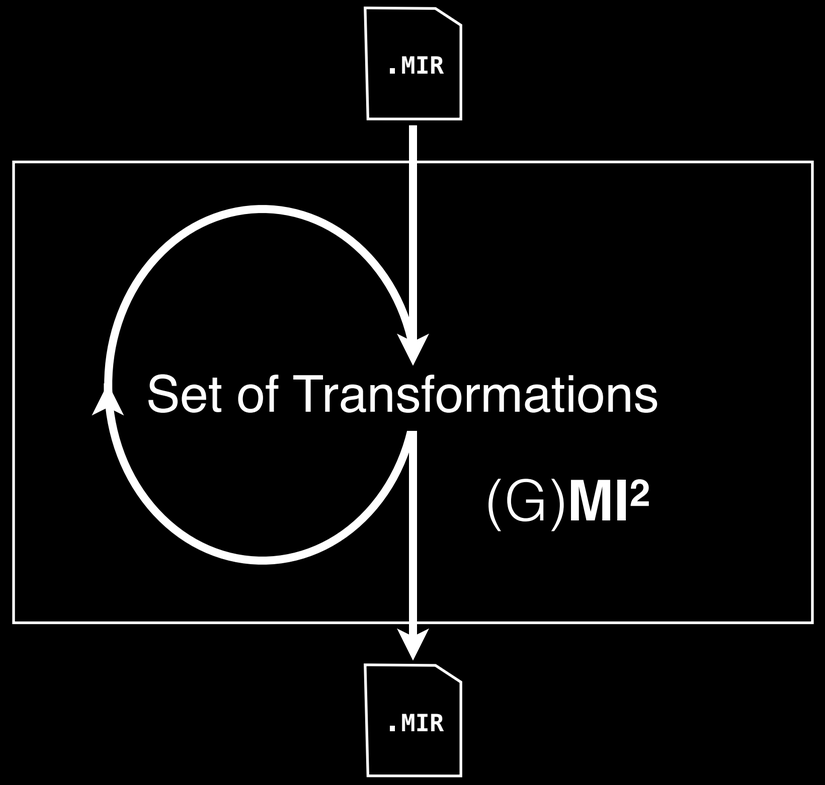

State expressed in the IR

# Testability



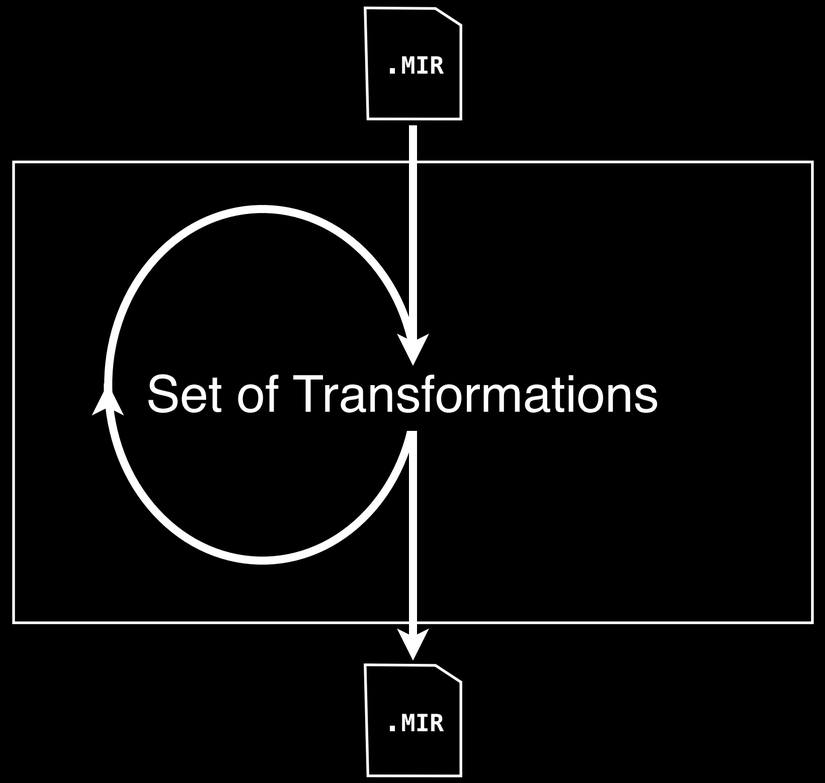

State expressed in the IR

# Testability



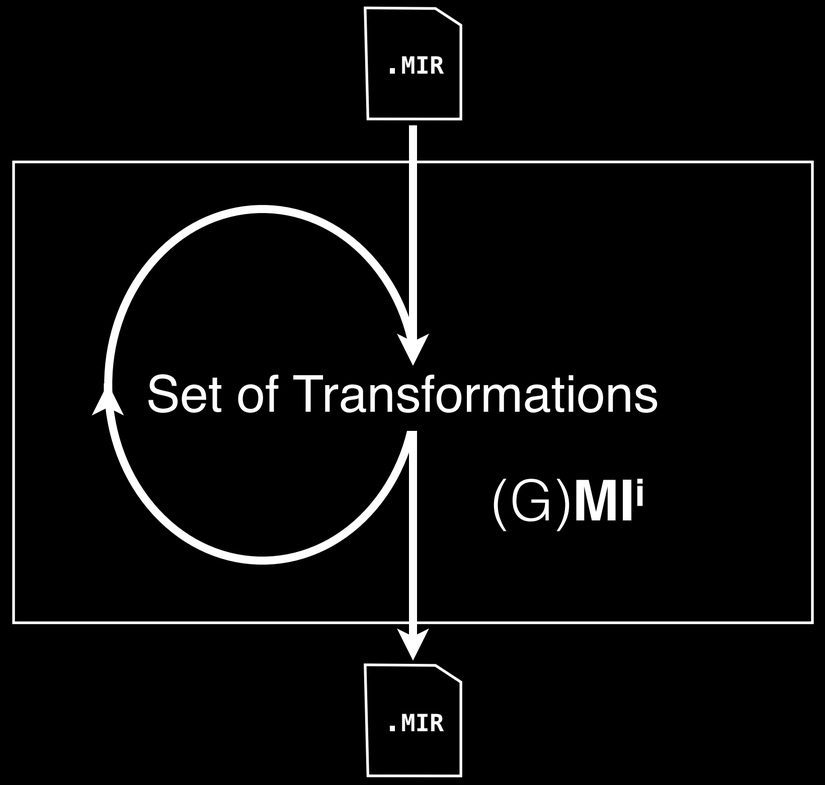

State expressed in the IR

# Testability



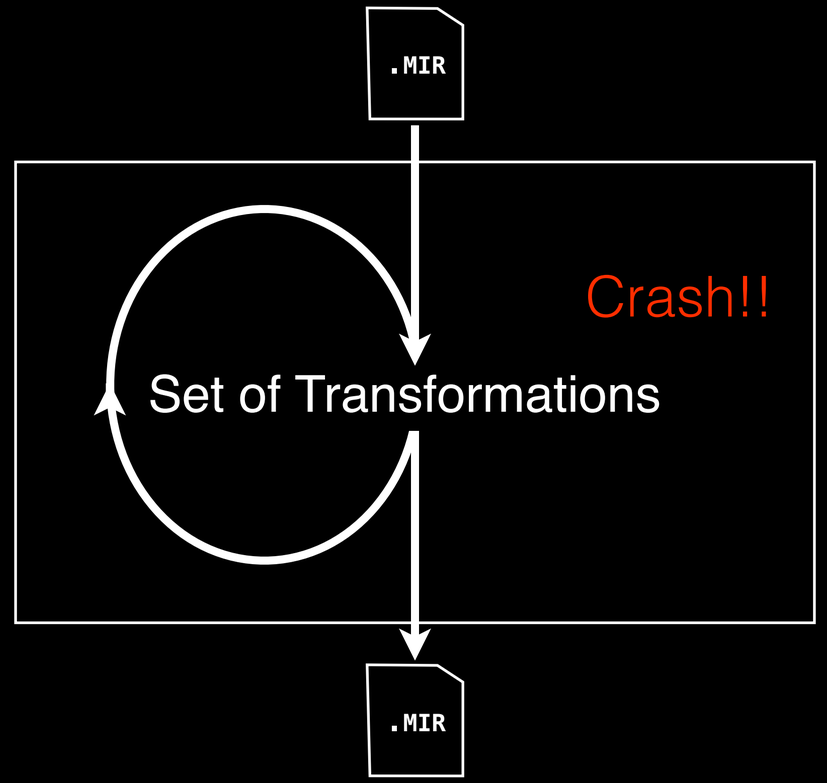

State expressed in the IR

# Testability



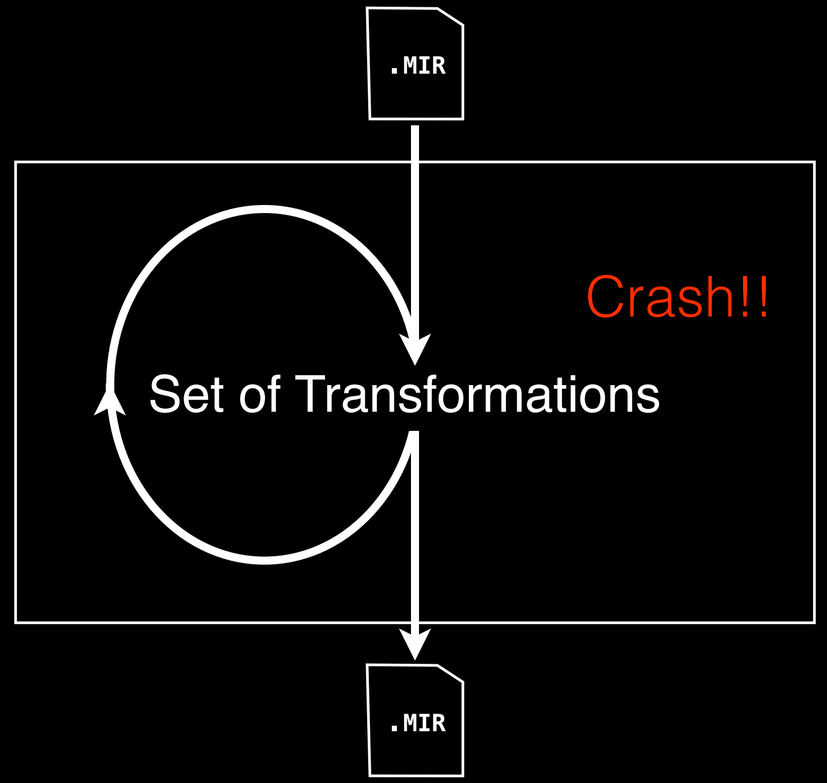

State expressed in the IR

# Testability



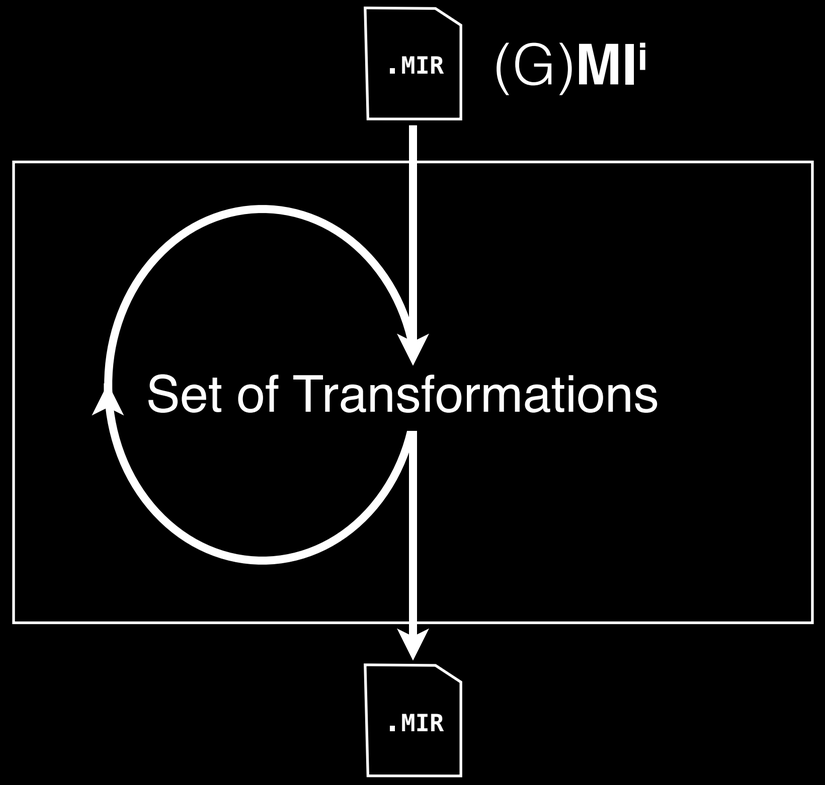

State expressed in the IR

# Testability



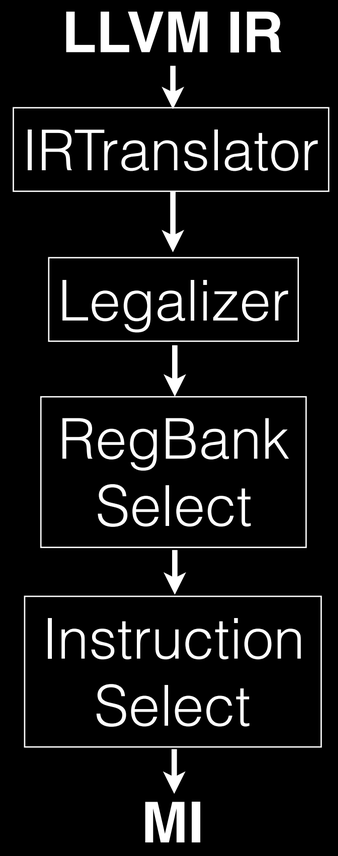

State expressed in the IR

# Testability



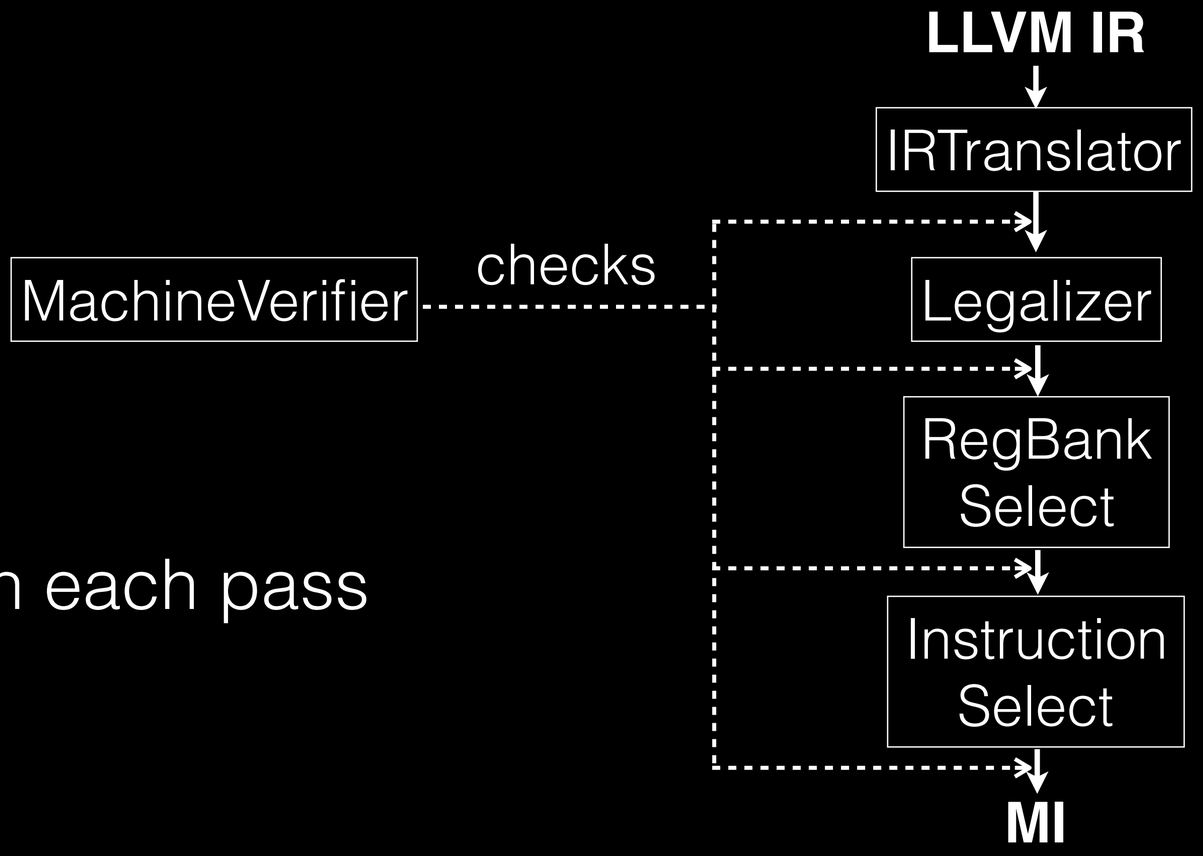

State expressed in the IR

# Testability

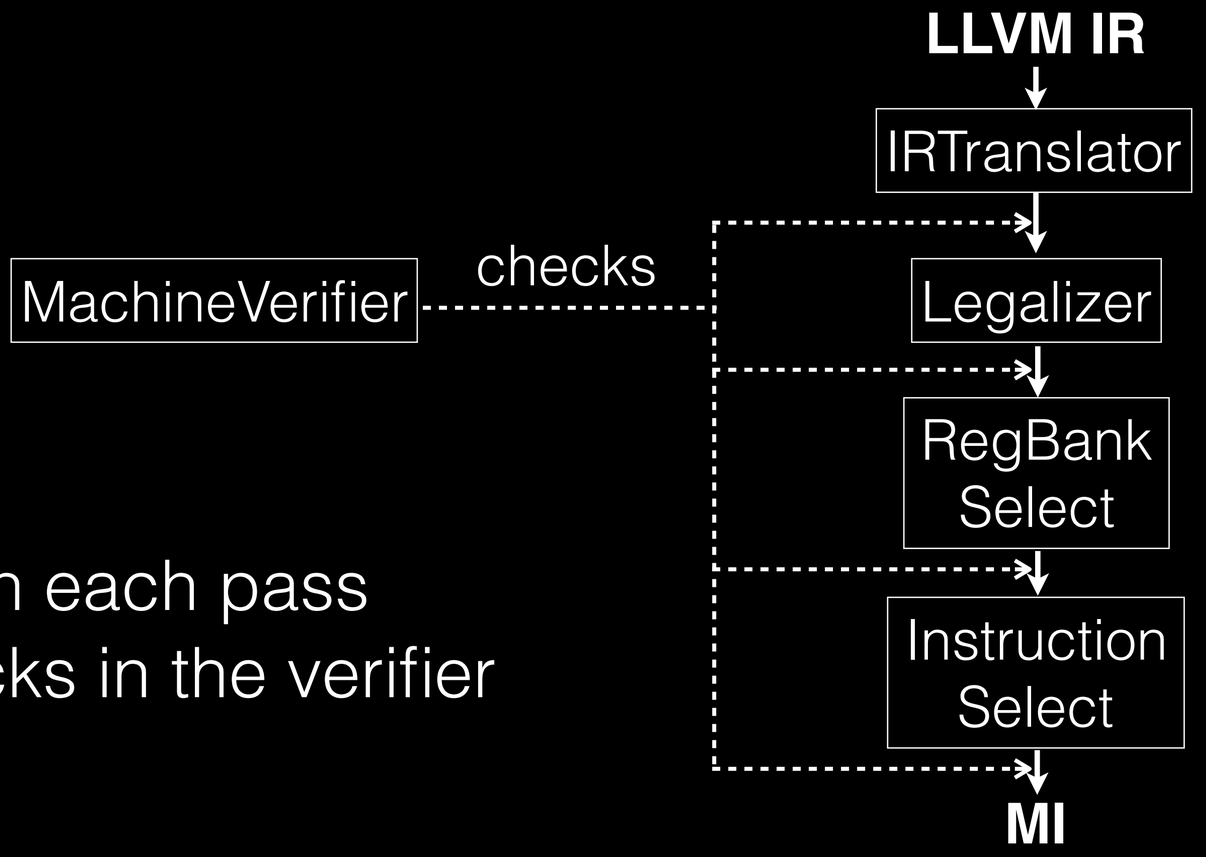


# Testability

- Verifier between each pass



# Testability




- Verifier between each pass

TODO Add more checks in the verifier

# Global ISel In Depth

## Passes

# Global ISel In Depth

## Passes: IRTranslator

# IRTranslator

# IRTranslator


- IRTranslator: Target independent translation

# IRTranslator


- IRTranslator: Target independent translation
- CallLowering: Provide hooks for ABI lowering

# IRTranslator

## CallLowering

TARGET API

# IRTranslator

## CallLowering

TARGET API

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

# IRTranslator

## CallLowering

TARGET API

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
lowerReturn(Value, VReg)
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
lowerReturn(Value, VReg)
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
%x0 = COPY %2(_,s64)
RET_ReallyLR implicit %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
lowerReturn(Value, VReg)
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
%x0 = COPY %2(_,s64)
RET_ReallyLR implicit %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
lowerReturn(Value, VReg)
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
%x0 = COPY %2(_,s64)
RET_ReallyLR implicit %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
lowerReturn(Value, VReg)
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
%x0 = COPY %2(_,s64)
RET_ReallyLR implicit %x0
```

# IRTranslator

## CallLowering

TARGET API

```
lowerFormalArguments(Function, VRegs[])
lowerCall(Call, ResVReg, ArgVRegs[])
lowerReturn(Value, VReg)
```

```
define i64 @baz(i1* %addr1,
                <2 x i32> *%addr2) {
 %res = call i64 bar(i1* %addr1,
                     <2 x i32> *%addr2)
 ret i64 %res
}
```

```
%0(_,p0) = COPY %x0
%1(_,p0) = COPY %x1
%x0 = COPY %0(_,p0)
%x1 = COPY %1(_,p0)
BL @bar, csr_aarch64_aapcs, implicit-defs...
%2(_,s64) = COPY %x0
%x0 = COPY %2(_,s64)
RET_ReallyLR implicit %x0
```

# IRTranslator

## Aggregates

IN DEPTH

# IRTranslator

## Aggregates

IN DEPTH

```
%struct.AB = type {        i32,          i8                  }
```

# IRTranslator

## Aggregates

IN DEPTH

```
%struct.AB = type {        i32,          i8                }
```

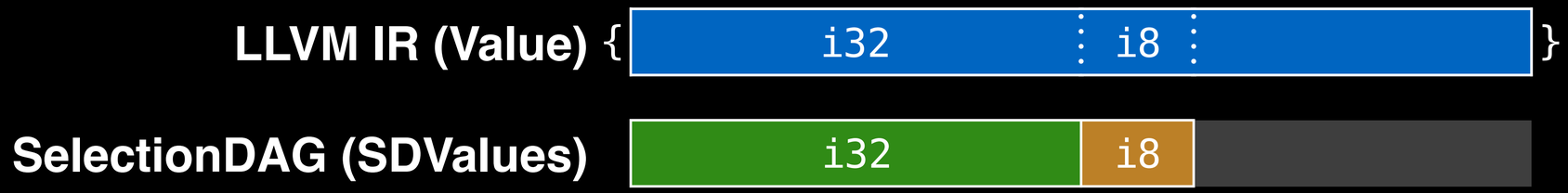
**LLVM IR (Value)** { i32 | i8 | } }

# IRTranslator

## Aggregates

IN DEPTH

```
%struct.AB = type { i32, i8 }
```

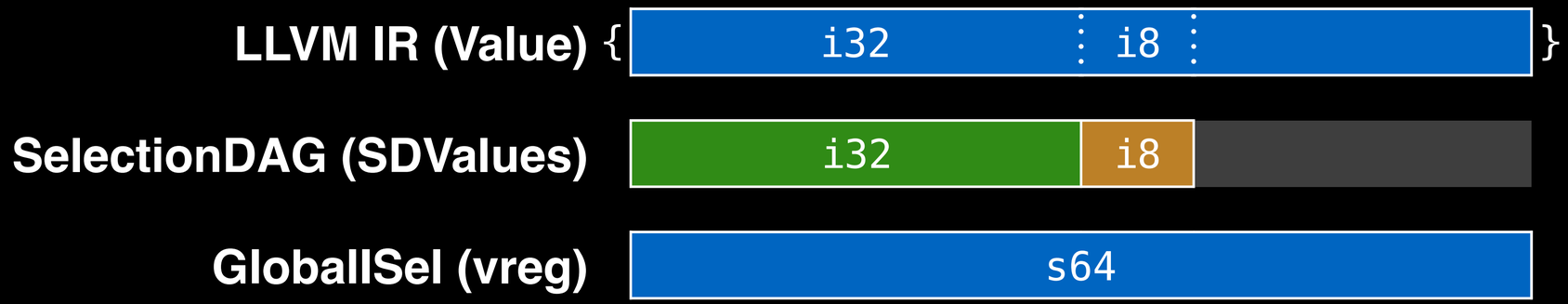


# IRTranslator

## Aggregates

IN DEPTH

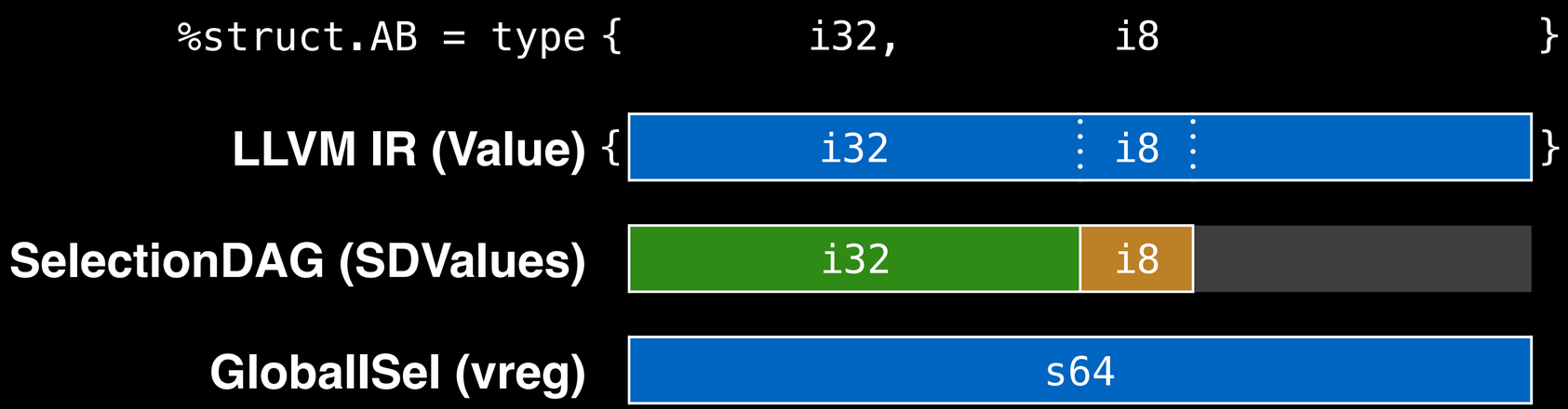
```
%struct.AB = type { i32, i8 }
```



# IRTranslator

## Aggregates

IN DEPTH

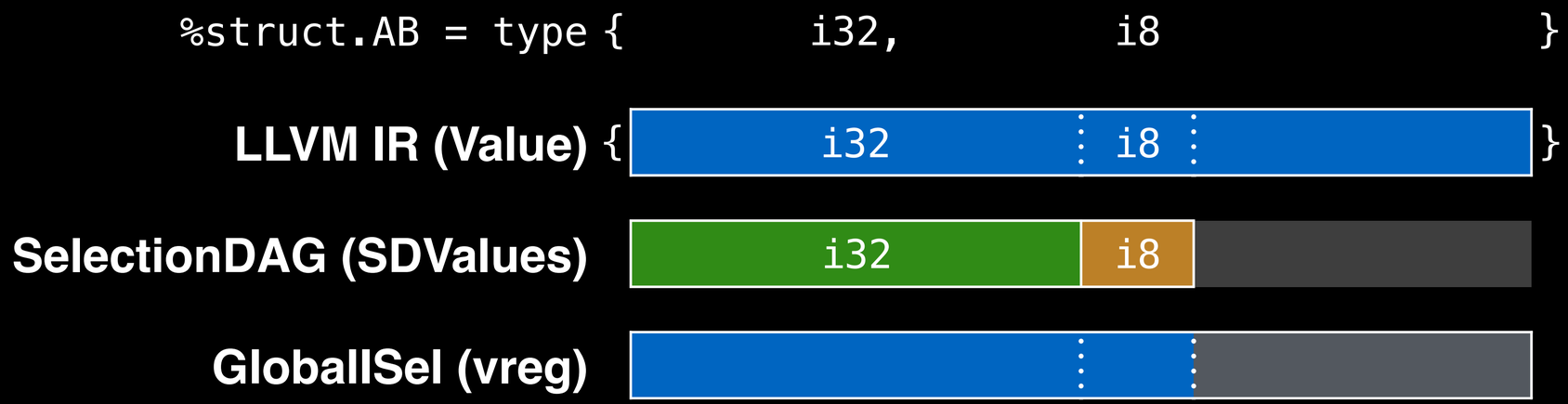



- One scalar vreg for aggregate type

# IRTranslator

## Aggregates

IN DEPTH

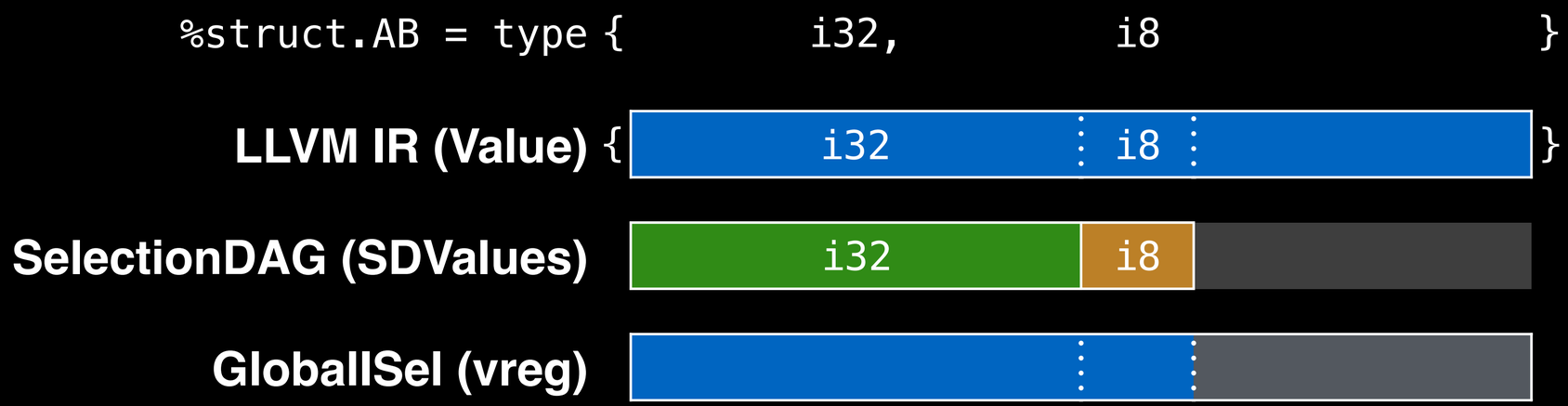



- One scalar vreg for aggregate type

# IRTranslator

## Aggregates

IN DEPTH



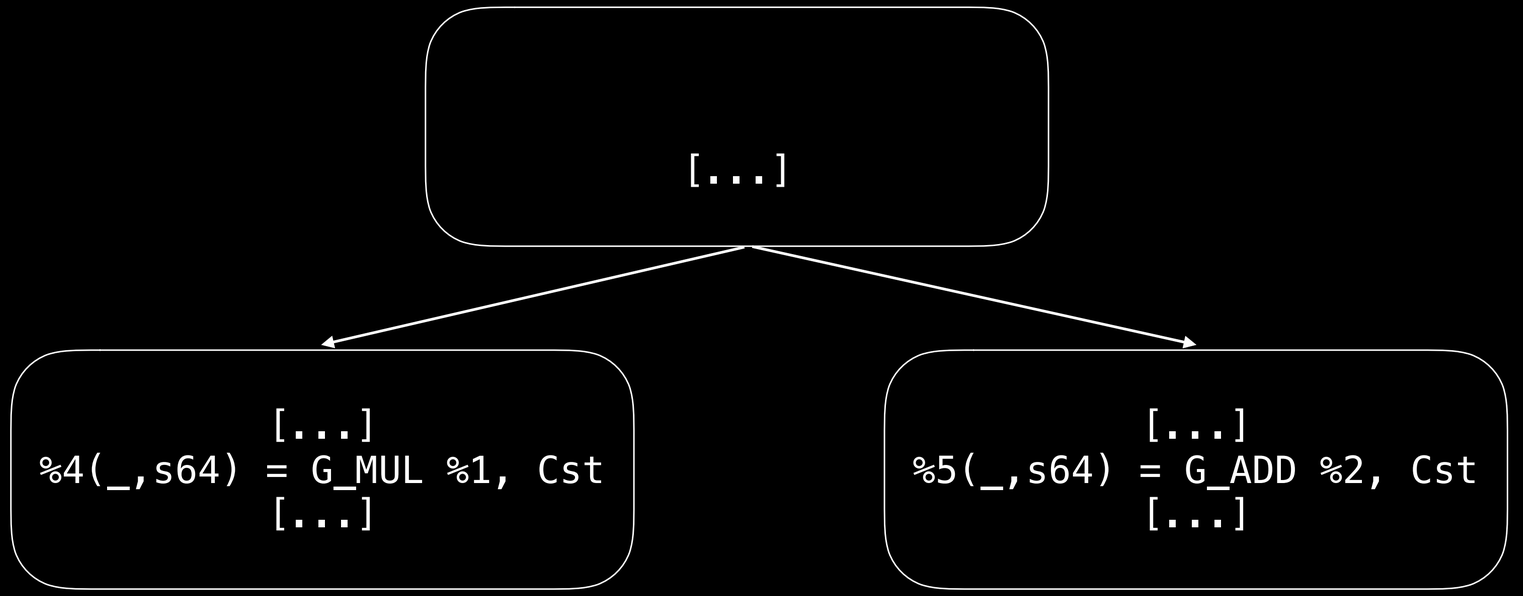

- One scalar vreg for aggregate type

TODO Express that some bits are garbage

# IRTranslator

## Constants

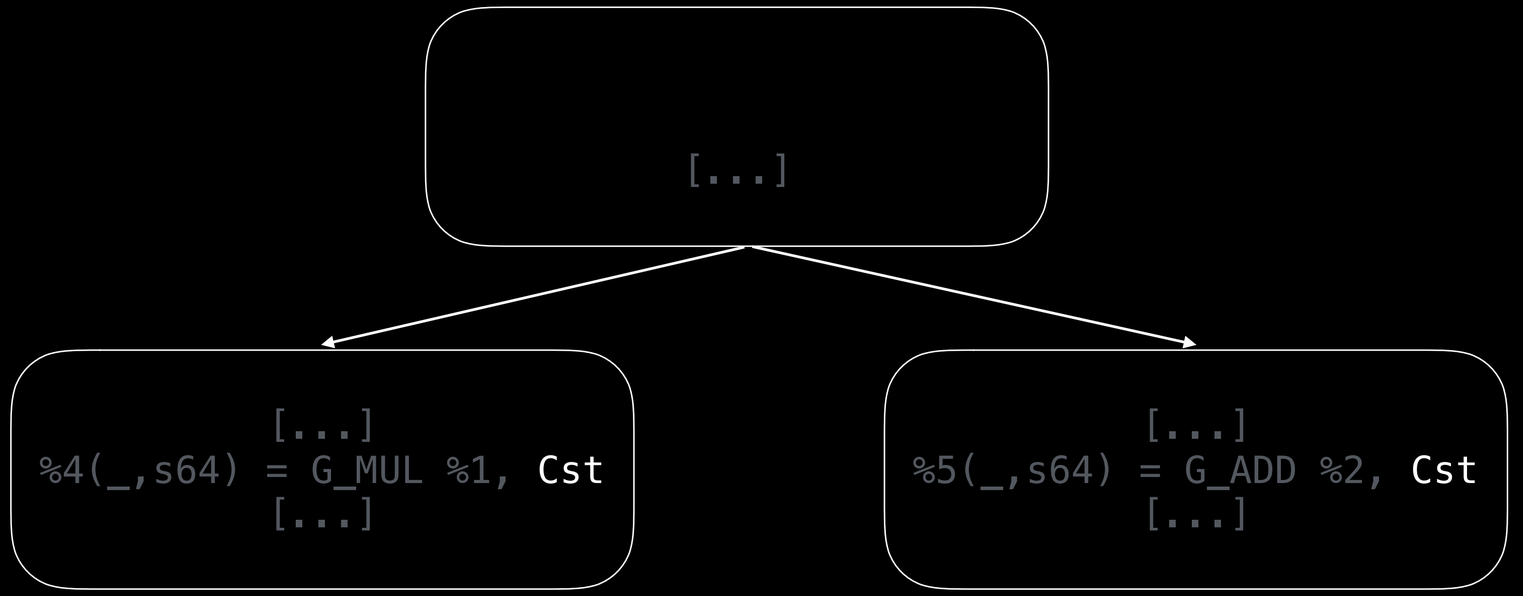
IN DEPTH



# IRTranslator

## Constants

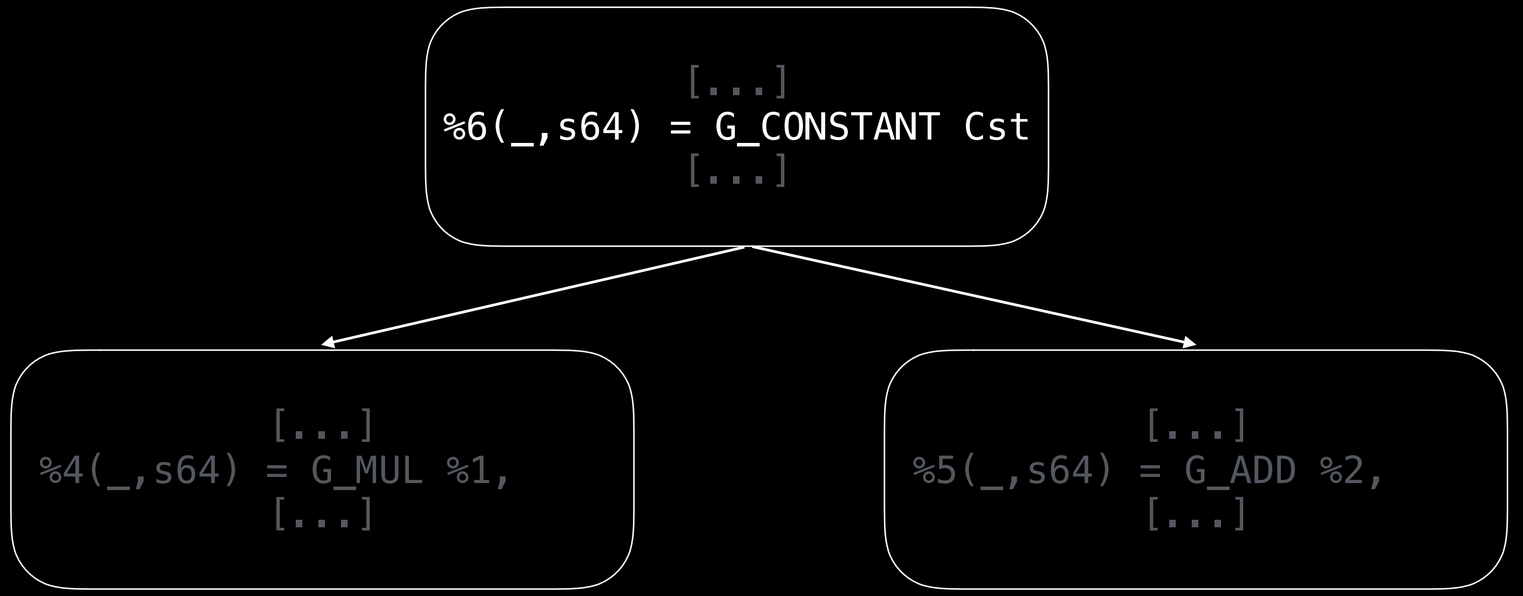
IN DEPTH



# IRTranslator

## Constants

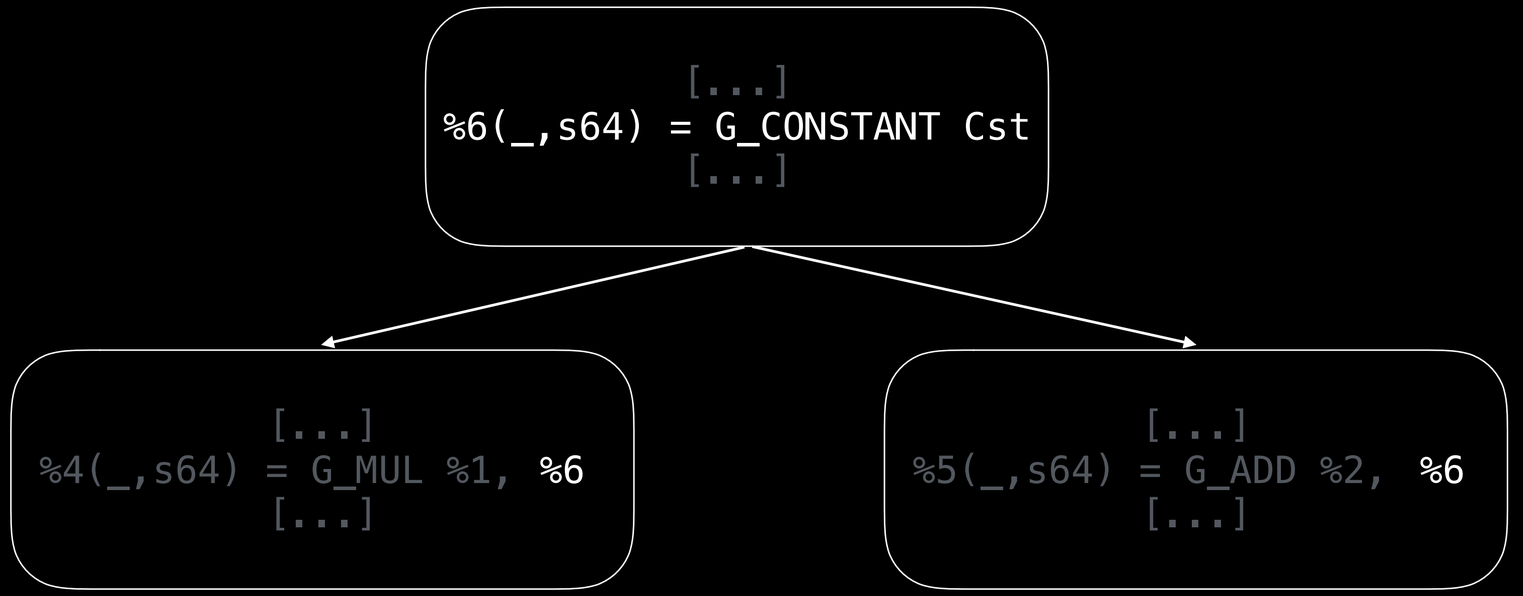
IN DEPTH



# IRTranslator

## Constants

IN DEPTH

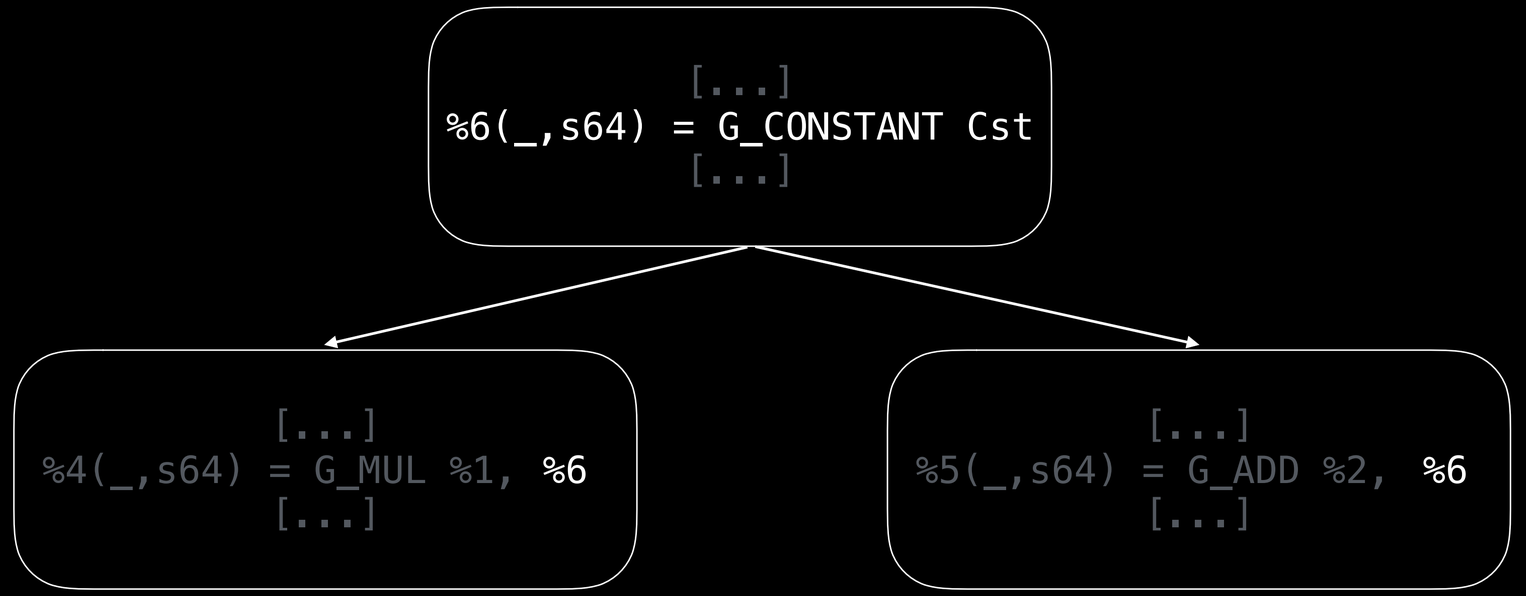



- Constants in entry block

# IRTranslator

## Constants

IN DEPTH



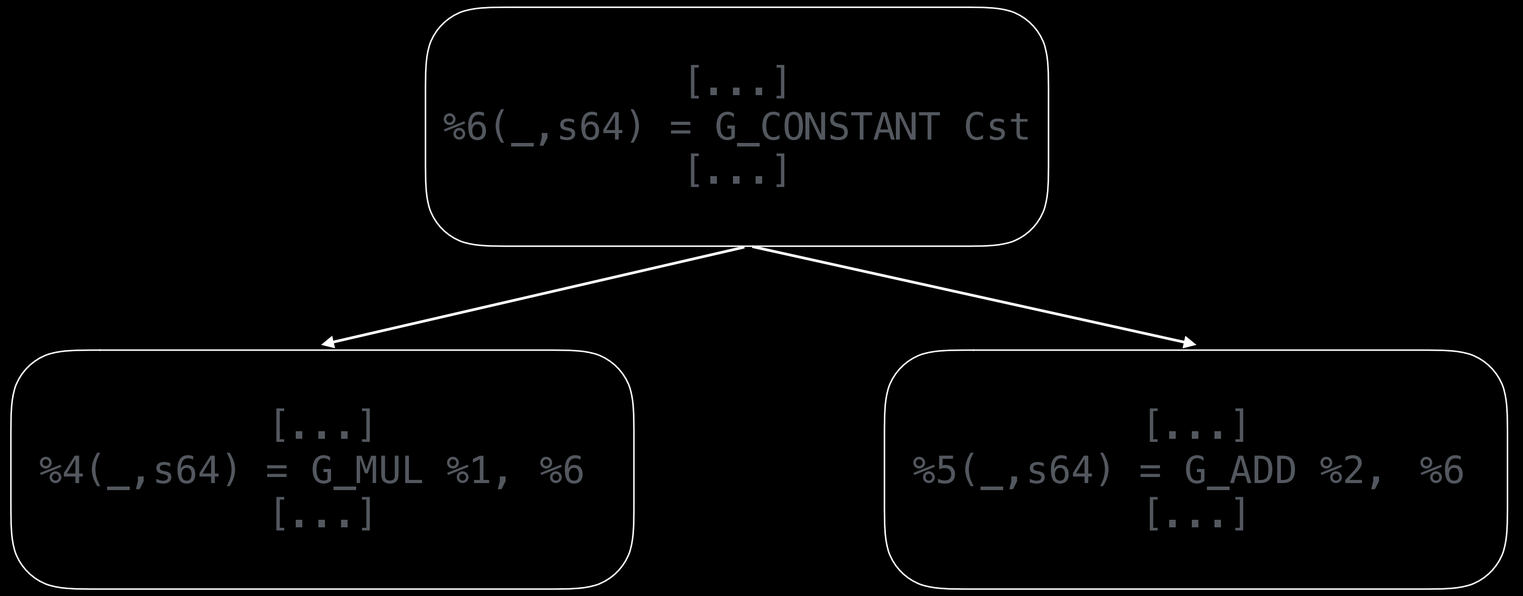

- Constants in entry block

TODO Investigate better constants placement

# IRTranslator

## Constants

IN DEPTH




- Constants in entry block

TODO Investigate better constants placement

# Global ISel In Depth

## Passes: Legalizer

# Legalizer

# Legalizer

Legalizer

- Legalizer Pass: Iterate and legalize

# Legalizer


- Legalizer Pass: Iterate and legalize
- LegalizerInfo: Drive the legalization process

# Legalizer


- Legalizer Pass: Iterate and legalize
- LegalizerInfo: Drive the legalization process
- LegalizerHelper: Implement the common legalization actions (NarrowScalar, Widen, etc.)

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...

%3(_,s1) = G_ICMP(eq) %2(_,s16), %2
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...
%4(_,s32) = G_ZEXT %2(_,s16)
%3(_,s1) = G_ICMP(eq) %4(_,s32), %4
```

# Legalizer

## LegalizerInfo

TARGET API

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

```
setAction({s1, 0, Legal}, G_ICMP)
setAction({s32, 1, Legal}, G_ICMP)
setAction({s16, 1, WidenScalar}, G_ICMP)
```

```
%0(_,s32) = ...
%1(_,s1) = G_ICMP(eq) %0(_,s32), %0

%2(_,s16) = ...
%4(_,s32) = G_ZEXT %2(_,s16)
%3(_,s1) = G_ICMP(eq) %4(_,s32), %4
```

# Legalizer

## LegalizerInfo

IN DEPTH

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

# Legalizer

## LegalizerInfo

IN DEPTH

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

TODO Support non-power of 2 types

# Legalizer

## LegalizerInfo

IN DEPTH

```
setAction({Type, [OpIdx,] Action},
          Opcode)
```

There are no illegal types, only illegal operations

- TODO Support non-power of 2 types
- TODO Infer legality from TableGen

# Global ISel In Depth

## Passes: RegBankSelect

# RegBankSelect

# RegBankSelect


- RegBankSelect: Main pass

# RegBankSelect

RegBankSelect

- RegBankSelect: Main pass
  - Perform top-down register bank assignments

# RegBankSelect

RegBankSelect

- RegBankSelect: Main pass
  - Perform top-down register bank assignments
  - Support two modes: Fast and Greedy

# RegBankSelect

RegBankSelect

- RegBankSelect: Main pass
  - Perform top-down register bank assignments
  - Support two modes: Fast and Greedy
- TODO Improve Greedy

# RegBankSelect

RegBankSelect

- RegBankSelect: Main pass
  - Perform top-down register bank assignments
  - Support two modes: Fast and Greedy
- TODO Improve Greedy
- TODO Add a Global mode

# RegBankSelect


- RegBankSelect: Main pass
  - Perform top-down register bank assignments
  - Support two modes: Fast and Greedy

TODO Improve Greedy

TODO Add a Global mode

- RegisterBankInfo: Provide RegisterBank related information

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
```

- Coverage of the RegisterClasses by the RegisterBanks

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
```

```
FPRRegBank covers FPR32RegClass
```

- Coverage of the RegisterClasses by the RegisterBanks

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
```

```
FPRRegBank covers FPR32RegClass
```

- Coverage of the RegisterClasses by the RegisterBanks

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
```

```
FPRRegBank covers FPR32RegClass
FPRRegBank covers FPR64RegClass
```

- Coverage of the RegisterClasses by the RegisterBanks

# RegBankSelect

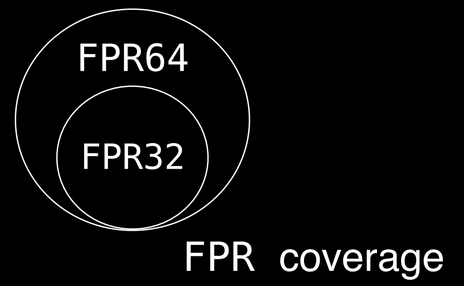
## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
```

```
FPRRegBank covers FPR32RegClass
FPRRegBank covers FPR64RegClass
```

- Coverage of the RegisterClasses by the RegisterBanks



# RegBankSelect

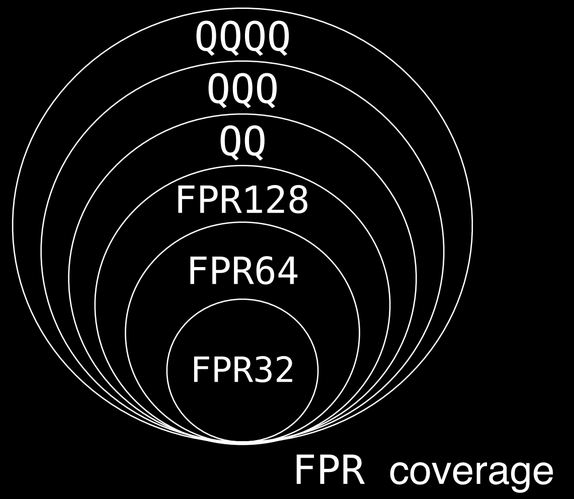
## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
```

- Coverage of the RegisterClasses by the RegisterBanks

```
FPRRegBank covers FPR32RegClass
FPRRegBank covers FPR64RegClass
FPRRegBank covers QQQQRegClass
```



# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
```

- Coverage of the RegisterClasses by the RegisterBanks

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
```

```
%0(RBDst,Size) = COPY %1(RBSrc,Size)
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
getInstrMapping(MachineInstr)
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies
- Mapping of the Instructions

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
getInstrMapping(MachineInstr)
```

```
%0(_,s64) = G_OR %1, %2
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies
- Mapping of the Instructions

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
getInstrMapping(MachineInstr)
```

```
%0(_,s64) = G_OR %1, %2

{/*ID*/ DefaultMappingID,
 /*Cost*/ 1,
 /*Opd0*/ {[0,63], GPR},
 /*Opd1*/ {[0,63], GPR},
 /*Opd2*/ {[0,63], GPR}}
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies
- Mapping of the Instructions

# RegBankSelect

## RegisterBankInfo

TARGET API

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
getInstrMapping(MachineInstr)
getInstrAlternativeMappings(MachineInstr)
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies
- Mapping of the Instructions

```
%0(_,s64) = G_OR %1, %2

{/*ID*/ DefaultMappingID,
 /*Cost*/ 1,
 /*Opd0*/ {[0,63], GPR},
 /*Opd1*/ {[0,63], GPR},
 /*Opd2*/ {[0,63], GPR}}

{/*ID*/ VecOR,
 /*Cost*/ 1,
 /*Opd0*/ {[0,63], FPR},
 /*Opd1*/ {[0,63], FPR},
 /*Opd2*/ {[0,63], FPR}}
```

# RegBankSelect

## RegisterBankInfo

IN DEPTH

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
getInstrMapping(MachineInstr)
getInstrAlternativeMappings(MachineInstr)
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies
- Mapping of the Instructions

TODO Merge the API related to instruction mappings

# RegBankSelect

## RegisterBankInfo

IN DEPTH

```
addRegBankCoverage(RegBank, RegClass)
copyCost(RegBankDst, RegBankSrc, Size)
getInstrMapping(MachineInstr)
getInstrAlternativeMappings(MachineInstr)
```

- Coverage of the RegisterClasses by the RegisterBanks
- Cost of cross register bank copies
- Mapping of the Instructions

TODO Merge the API related to instruction mappings

TODO Infer mappings from TableGen

# Global ISel In Depth

## Passes: InstructionSelect

# InstructionSelect

# InstructionSelect


- InstructionSelect Pass: ISel engine

# InstructionSelect

InstructionSelect

- InstructionSelect Pass: ISel engine
  - Traverse blocks bottom-up

# InstructionSelect


- InstructionSelect Pass: ISel engine
  - Traverse blocks bottom-up
  - Provide dead code elimination for free

# InstructionSelect


- InstructionSelect Pass: ISel engine
  - Traverse blocks bottom-up
  - Provide dead code elimination for free
- InstructionSelector: Translate (G)MI to MI

# InstructionSelect

## InstructionSelector

TARGET API

```
select(MachineInstr)
```

# InstructionSelect

## InstructionSelector

TARGET API

```
select(MachineInstr)
```

```
%6(GPR,<2 x s32>) = G_OR %4, %5
```

# InstructionSelect

## InstructionSelector

TARGET API

```
select(MachineInstr)
```

```
%6(GPR,<2 x s32>) = ORRXrr %4, %5
```

- Switch to target specific opcode

# InstructionSelect

## InstructionSelector

TARGET API

```
select(MachineInstr)
```

```
%6(GPR64) = ORRXrr %4, %5
```

- Switch to target specific opcode

# InstructionSelect

## InstructionSelector

TARGET API

```
select(MachineInstr)
```

```
%6(GPR64) = ORRXrr %4, %5
```

- Switch to target specific opcode
- Set proper RegisterClass
  `InstructionSelector::constrainSelectedInstRegOperands`

# InstructionSelect

## InstructionSelector

IN DEPTH

```
select(MachineInstr)
```

```
%6(GPR64) = ORRXrr %4, %5
```

- Switch to target specific opcode
- Set proper RegisterClass
  `InstructionSelector::constrainSelectedInstRegOperands`

# InstructionSelect

## InstructionSelector

IN DEPTH

```
select(MachineInstr)
```

```
%6(GPR64) = ORRXrr %4, %5
```

- Switch to target specific opcode
- Set proper RegisterClass
  `InstructionSelector::constrainSelectedInstRegOperands`
- InstructionSelector bound to subtarget

# InstructionSelect

## InstructionSelector

IN DEPTH

```
select(MachineInstr)
```

```
%6(GPR64) = ORRXrr %4, %5
```

- Switch to target specific opcode
- Set proper RegisterClass
  `InstructionSelector::constrainSelectedInstRegOperands`
- InstructionSelector bound to subtarget

TODO Generate select code from TableGen

# Global ISel In Depth

## Targeting Overview

# Targeting

## TargetPassConfig

TARGET API

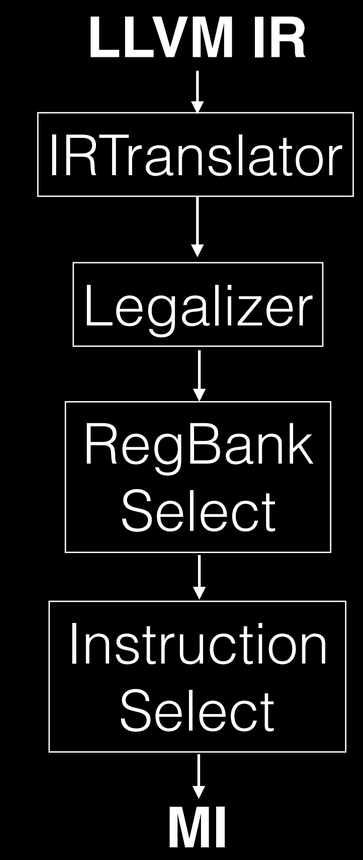
# Targeting

## TargetPassConfig

TARGET API

- Create the Global ISel pipeline

# Targeting
## TargetPassConfig

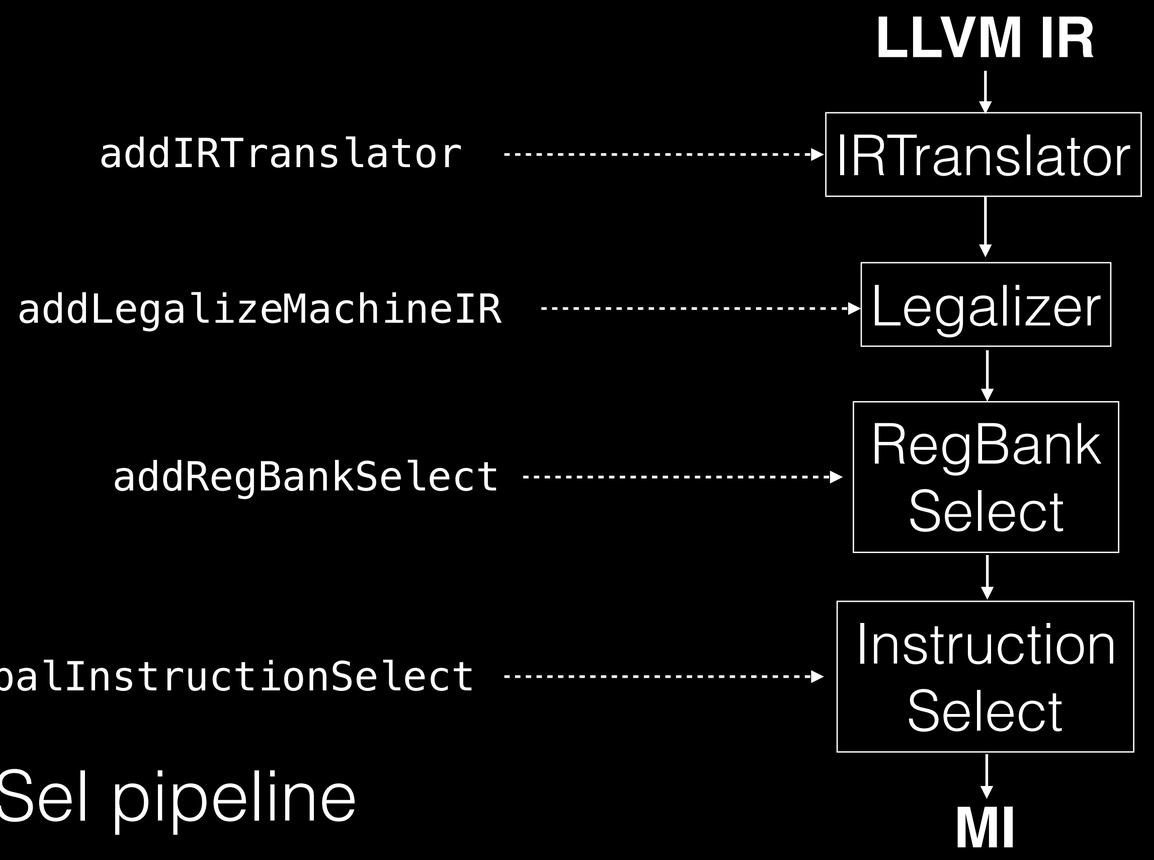



TARGET API

- Create the Global ISel pipeline

# Targeting

## TargetPassConfig



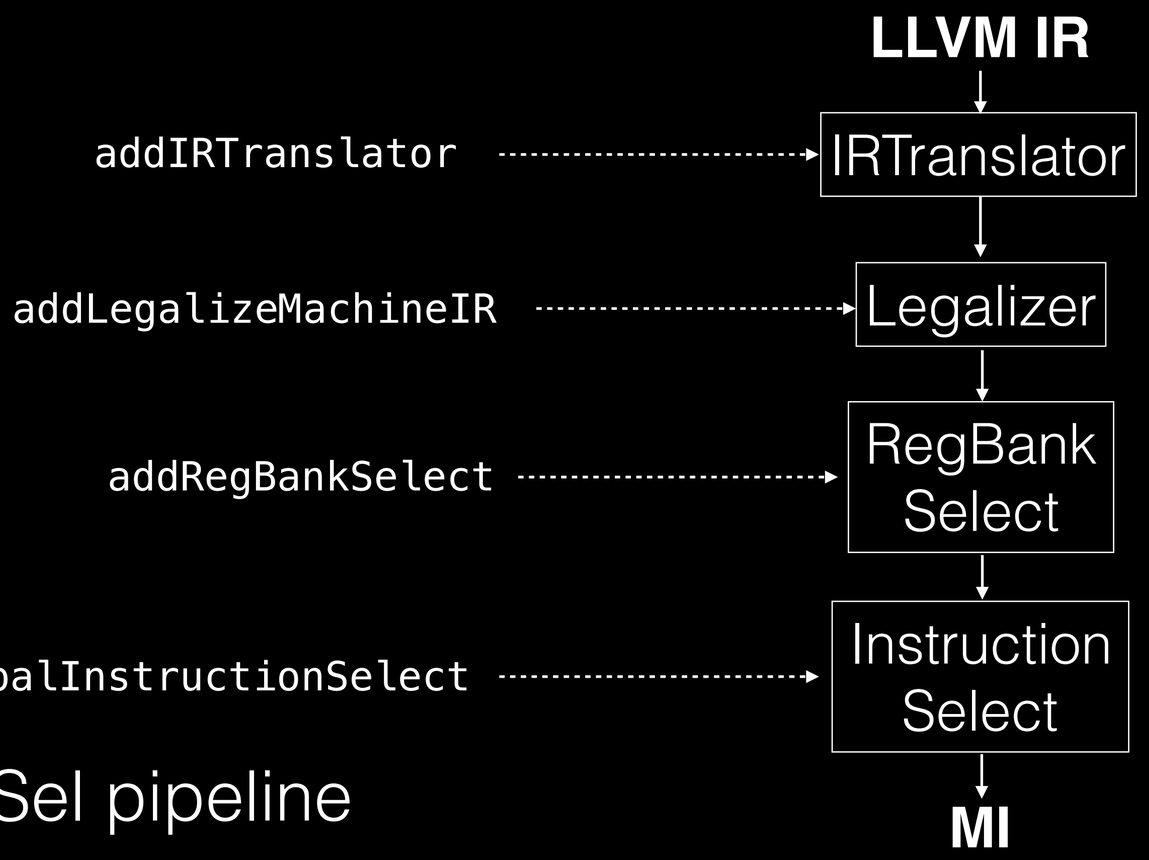

- Create the Global ISel pipeline

TARGET API

# Targeting

## TargetPassConfig

TARGET API

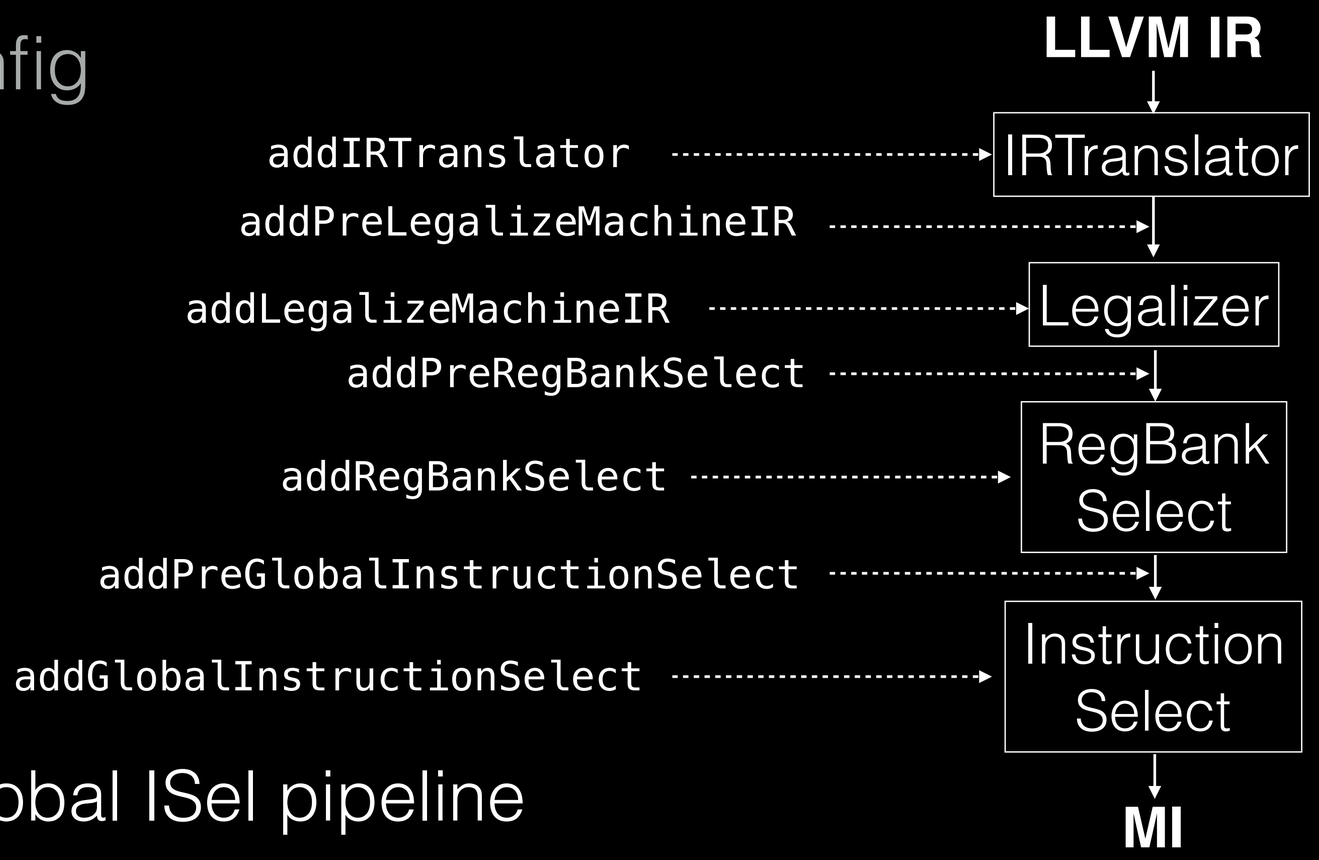



- Create the Global ISel pipeline
- Inject additional target specific passes

# Targeting

## TargetPassConfig

TARGET API

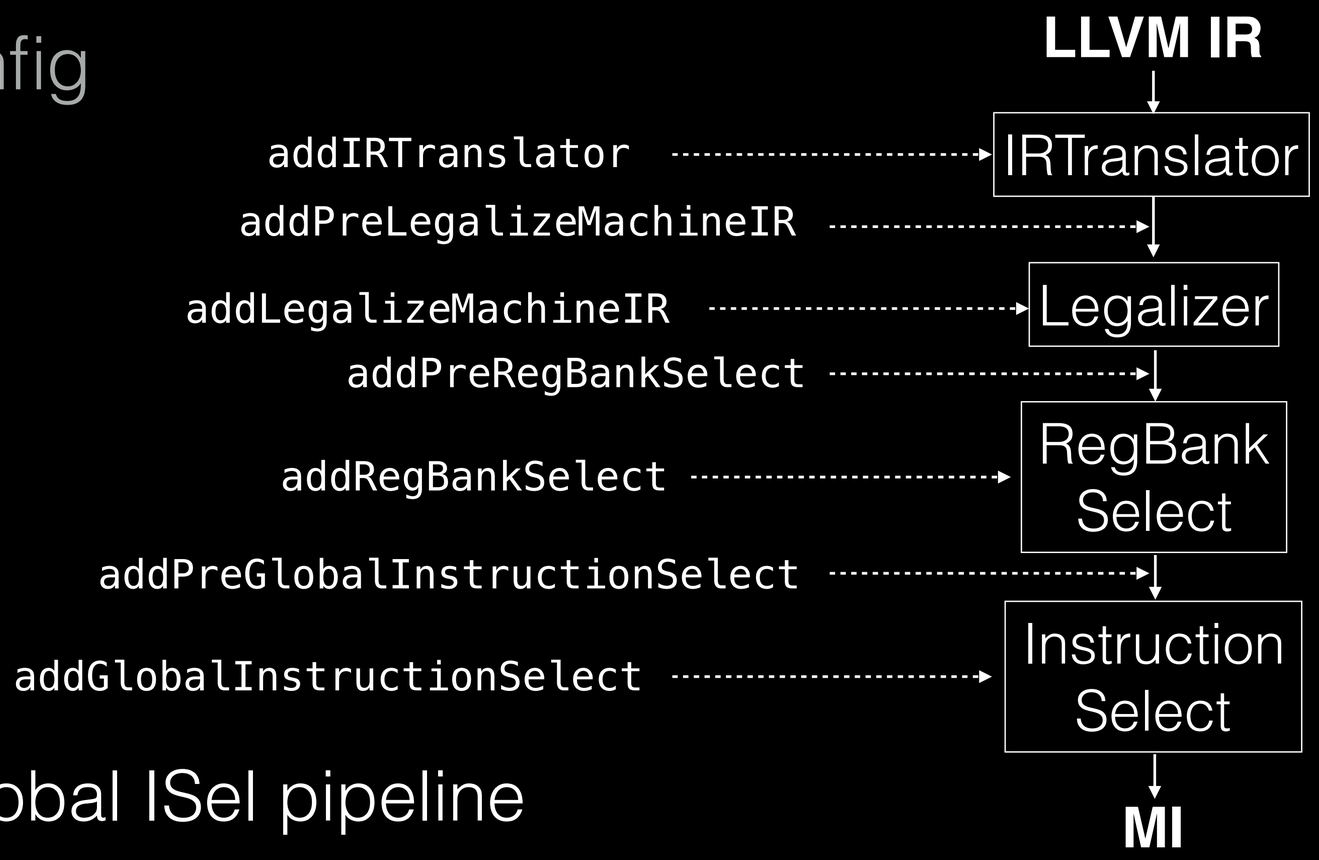



- Create the Global ISel pipeline
- Inject additional target specific passes

# Targeting

## TargetPassConfig

TARGET API




- Create the Global ISel pipeline
- Inject additional target specific passes
- GISelAccessor available as a proxy

# Targeting

## Summary

TARGET API

# Targeting
## Summary

TARGET API

- TargetPassConfig

# Targeting
## Summary

TARGET API

- TargetPassConfig
- CallLowering

# Targeting
## Summary

TARGET API

- TargetPassConfig
- CallLowering
- LegalizerInfo

# Targeting
## Summary

TARGET API

- TargetPassConfig
- CallLowering
- LegalizerInfo
- RegisterBankInfo

# Targeting

## Summary

TARGET API

- TargetPassConfig
- CallLowering
- LegalizerInfo
- RegisterBankInfo
- InstructionSelector

# Global ISel

## Status

# Non Goals

# Non Goals

☒ Self contained representation

# Goals

# Goals

- ☑ Global

# Goals

- ☑ Global
- ☑ Fast

# Goals

- ☑ Global
- ☑ Fast
- ☑ Shared code path for fast and good ISel

# Goals

- ☑ Global
- ☑ Fast
- ☑ Shared code path for fast and good ISel
- ☑ No change to LLVM IR

# Goals

- ☑ Global
- ☑ Fast
- ☑ Shared code path for fast and good ISel
- ☑ No change to LLVM IR
- ⍰ More configurable

# Goals

- ☑ Global
- ☑ Fast
- ☑ Shared code path for fast and good ISel
- ☑ No change to LLVM IR
- ? More configurable
- ? Easier to maintain/understand

# Future Work

# Future Work

- Work on supporting all IR

# Future Work

- Work on supporting all IR
- Work on compile time and runtime performance

# Future Work

- Work on supporting all IR
- Work on compile time and runtime performance
- Implement TableGen support

# Future Work

- Work on supporting all IR
- Work on compile time and runtime performance
- Implement TableGen support
- Deliver documentation

# Future Work

- Work on supporting all IR
- Work on compile time and runtime performance
- Implement TableGen support
- Deliver documentation
- Think about a transition plan

# Future Work

- Work on supporting all IR
- Work on compile time and runtime performance
- Implement TableGen support
- Deliver documentation
- Think about a transition plan
- Implement more backends

# Questions?

http://llvm.org/docs/GlobalISel.html