Global Instruction Selection

A Proposal

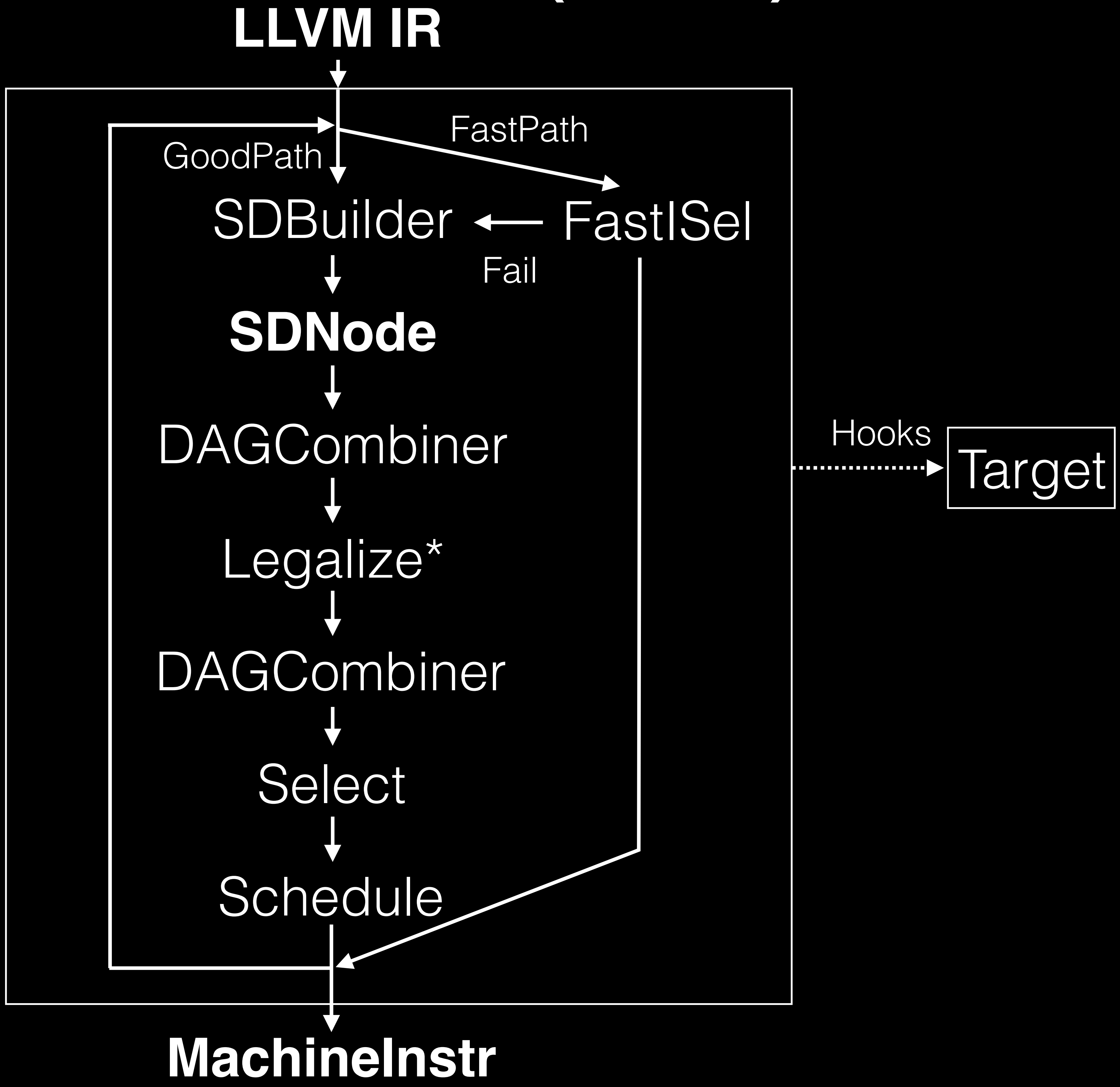
Quentin Colombet
Apple

What Is Instruction Selection?

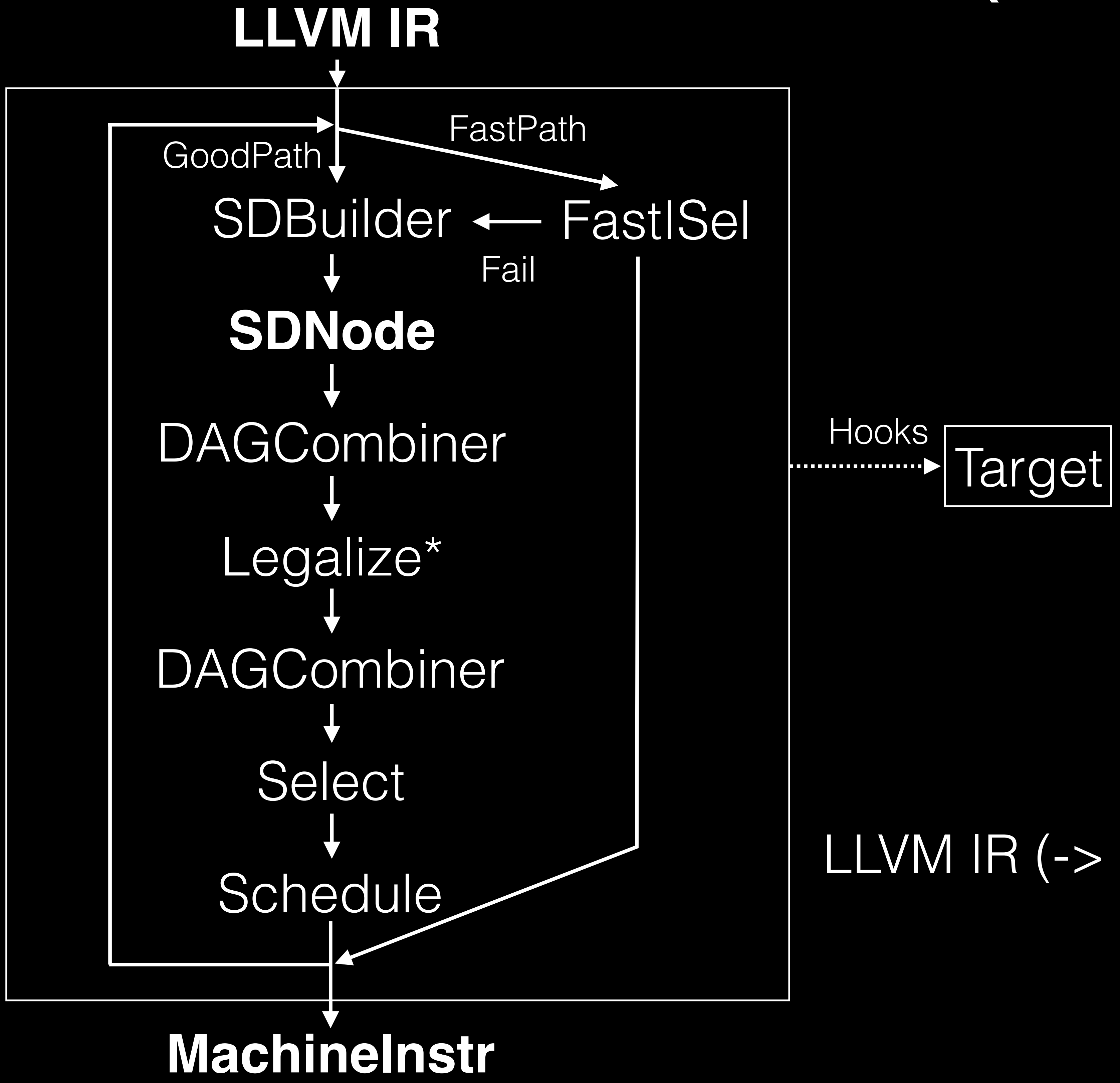
Translation from target independent intermediate representation (IR) to target specific IR.

LLVM IR -> MachineInstr

SelectionDAG (SD) ISel



SelectionDAG (SD) ISel



LLVM IR (-> SDNode) -> MachineInstr

Problems with SDISel

- Basic block scope
- SDNode IR, specific to instruction selection
- Monolithic

Goals

- Global
- Fast
- Shared code for fast and good paths
- IR that represents ISA concepts better
- More flexible pipeline
- Easier to maintain/understand
- Self contained representation
- No change to LLVM IR

Non-Goals for the Prototype

- Reuse of InstCombine
- Improve TableGen
- Support target specific features

Why Not Fix SDISel?

- Hard limitations of the underlying representation
- Would introduce SDNode IR to optimizers
- SDNode IR can be avoided
- Inherent overhead

Global ISel

LLVM IR

LLVM IR $\rightarrow \boxed{\text{IRTranslator}}$

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

- LLVM IR to generic (G) MachineInstr

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
```

- LLVM IR to generic (G) MachineInstr

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val = …
  addr = …
```

- LLVM IR to generic (G) MachineInstr

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val = …
  addr = …
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val = …
  addr = …
  loaded = gLD addr
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val = …
  addr = …
  loaded = gLD (double) addr
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

NEW MachineInstrs get a type

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val = …
  addr = …
  loaded(64) = gLD (double) addr
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

NEW MachineInstrs get a type

NEW Virtual registers get a size

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(64) = …
  addr(32) = …
  loaded(64) = gLD (double) addr
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

NEW MachineInstrs get a type

NEW Virtual registers get a size

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(64) = …
  addr(32) = …
  loaded(64) = gLD (double) addr
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

NEW MachineInstrs get a type

NEW Virtual registers get a size

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(64) = …
  addr(32) = …
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

NEW MachineInstrs get a type

NEW Virtual registers get a size

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(64) = …
  addr(32) = …
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

NEW MachineInstrs get a type

NEW Virtual registers get a size

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(64) = …
  addr(32) = …
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  … = and
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr

NEW MachineInstrs get a type

NEW Virtual registers get a size

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(64) = …
  addr(32) = …
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  … = and
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr
  - NEW MachineInstrs get a type
  - NEW Virtual registers get a size
  - ABI lowering

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(64) = … R0,R1
  addr(32) = …
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  … = and
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr
  - NEW MachineInstrs get a type
  - NEW Virtual registers get a size
  - ABI lowering

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = …
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  … = and
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr
  - NEW MachineInstrs get a type
  - NEW Virtual registers get a size
  - ABI lowering

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  … = and
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr
  - NEW MachineInstrs get a type
  - NEW Virtual registers get a size
  - ABI lowering

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr
  - NEW MachineInstrs get a type
  - NEW Virtual registers get a size
  - ABI lowering

IRTranslator

```
define double @foo(double %val,
                   double* %addr) {
  %intval = bitcast double %val to i64
  %loaded = load double, double* %addr
  %mask = bitcast double %loaded to i64
  %and = and i64 %intval, %mask
  %res = bitcast i64 %and to double
  ret double %res
}
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- LLVM IR to generic (G) MachineInstr
  - One IR instruction to 0..* (G) MachineInstr
  - NEW MachineInstrs get a type
  - NEW Virtual registers get a size
  - ABI lowering

$$\textbf{LLVM IR} \rightarrow \boxed{\text{IRTranslator}}$$

$$\begin{tikzcd} \textbf{LLVM IR} \arrow[r] & \boxed{\text{IRTranslator}} \arrow[r] & \text{(G)}\textbf{MI} \end{tikzcd}$$

$$\begin{tikzcd}\textbf{LLVM IR} \arrow[r] & \boxed{\text{IRTranslator}} \arrow[r] & \text{(G)}\textbf{MI} \arrow[r] & \boxed{\text{Legalizer}}\end{tikzcd}$$

Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

NEW No illegal types, just illegal operations

- Illegal (G)MachineInstr to legal (G)MachineInstr

Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

NEW No illegal types, just illegal operations

- Illegal (G)MachineInstr to legal (G)MachineInstr

Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

NEW No illegal types, just illegal operations

- Illegal (G)MachineInstr to legal (G)MachineInstr

Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = extract val
  low(32),high(32) = extract loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = build_sequence land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

New No illegal types, just illegal operations

- Illegal (G)MachineInstr to legal (G)MachineInstr

Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = extract val
  low(32),high(32) = extract loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = build_sequence land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

NEW No illegal types, just illegal operations

- Illegal (G)MachineInstr to legal (G)MachineInstr
  - NEW State expressed in the IR (extract, build_sequence)
  - Iterative process
  - Set of transformations
  - Iterate until no more changes

Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  and(64) = gAND (i64) val, loaded
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = extract val
  low(32),high(32) = extract loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = build_sequence land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

NEW No illegal types, just illegal operations

- Illegal (G)MachineInstr to legal (G)MachineInstr
  - NEW State expressed in the IR (extract, build_sequence)
  - Iterative process
  - Set of transformations
  - Iterate until no more changes

Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = extract val
  low(32),high(32) = extract loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = build_sequence land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

NEW Introduce a “LegalizerToolkit” for (custom) lowering of legalization

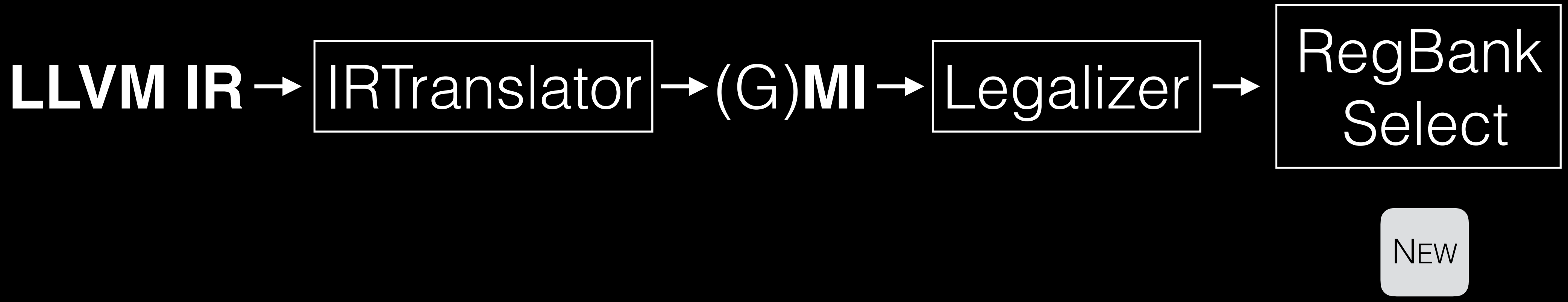
Legalizer

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = extract val
  low(32),high(32) = extract loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = build_sequence land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

NEW Introduce a “LegalizerToolkit” for (custom) lowering of legalization

$$\begin{tikzcd}\textbf{LLVM IR} \arrow[r] & \boxed{\text{IRTranslator}} \arrow[r] & \text{(G)}\textbf{MI} \arrow[r] & \boxed{\text{Legalizer}}\end{tikzcd}$$


LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
NEW

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- Assign virtual registers to register bank
- Avoid cross domain penalties
- Aware of register pressure

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(GPR,32) = COPY R2
  loaded(FPR,64) = gLD (double) addr
  lval(GPR,32),hval(GPR,32) = VMOVRRD val
  low(GPR,32),high(GPR,32) = VMOVRRD loaded
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and(FPR,64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- Assign virtual registers to register bank
- Avoid cross domain penalties
- Aware of register pressure

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1    {(FPR,0xFF…FF),1}
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1    {(FPR,0xFF…FF),1} {(GPR,0xFFFF…0000)(GPR,0x0000…FFFF),0}
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1    {(FPR,0xFF…FF),1} {(GPR,0xFFFF…0000)(GPR,0x0000…FFFF),0}
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val   {(FPR,0xFF…FF),1}
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1    {(FPR,0xFF…FF),1} {(GPR,0xFFFF…0000)(GPR,0x0000…FFFF),0}
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val   {(FPR,0xFF…FF),1} {(GPR,0xFFFF…0000)(GPR,0x0000…FFFF),0}
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1    {(FPR,0xFF…FF),1} {(GPR,0xFFFF…0000)(GPR,0x0000…FFFF),0}
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val   {(FPR,0xFF…FF),1} {(GPR,0xFFFF…0000)(GPR,0x0000…FFFF),0}
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode
  - Assigns register banks to avoid cross bank copies
  - May use the LegalizerToolkit to change the code

RegBankSelect

NEW

```
foo:
  val(FPR,64) = VMOVDRR R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = VMOVRRD val
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode
  - Assigns register banks to avoid cross bank copies
  - May use the LegalizerToolkit to change the code

RegBankSelect

NEW

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode
  - Assigns register banks to avoid cross bank copies
  - May use the LegalizerToolkit to change the code

RegBankSelect

NEW

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = COPIES loaded1,loaded2
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode
  - Assigns register banks to avoid cross bank copies
  - May use the LegalizerToolkit to change the code

RegBankSelect

NEW

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded(64) = gLD (double) addr
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = VMOVRRD loaded
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = COPIES loaded1,loaded2
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode
  - Assigns register banks to avoid cross bank copies
  - May use the LegalizerToolkit to change the code

RegBankSelect

NEW

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = COPIES loaded1,loaded2
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode
  - Assigns register banks to avoid cross bank copies
  - May use the LegalizerToolkit to change the code

RegBankSelect

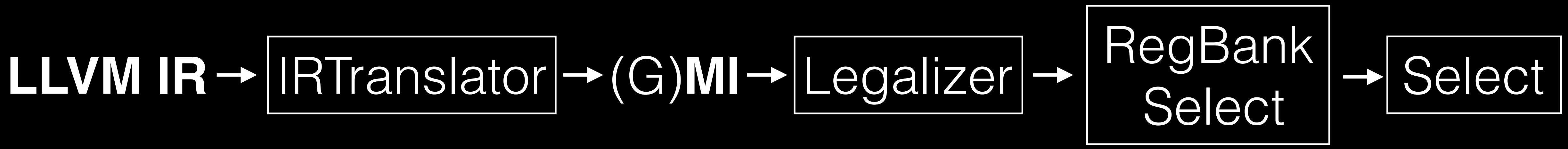
NEW

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(32),hval(32) = COPIES val1, val2
  low(32),high(32) = COPIES loaded1,loaded2
  land(32) = gAND (i32) lval, low
  hand(32) = gAND (i32) hval, high
  and(64) = VMOVDRR land, hand
  R0,R1 = VMOVRRD and
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- RegBankSelect:
  - Asks the target what register banks are supported for a given opcode
  - Assigns register banks to avoid cross bank copies
  - May use the LegalizerToolkit to change the code

$$\begin{tikzcd}
\textbf{LLVM IR} \arrow[r] & \boxed{\text{IRTranslator}} \arrow[r] & \text{(G)}\textbf{MI} \arrow[r] & \boxed{\text{Legalizer}} \arrow[r] & \boxed{\begin{array}{c}\text{RegBank}\\ \text{Select}\end{array}}
\end{tikzcd}$$


LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Select

Select

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- (G)MachineInstr to MachineInstr

Select

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- (G)MachineInstr to MachineInstr
  - NEW In-place morphing

Select

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1,
  low(GPR,32),high(GPR,32) = COPIES loaded1
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land,
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- (G)MachineInstr to MachineInstr
  - NEW In-place morphing

Select

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1,
  low(GPR,32),high(GPR,32) = COPIES loaded1
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = t2ANDrr (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land,
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- (G)MachineInstr to MachineInstr
  NEW In-place morphing

Select

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1,
  low(GPR,32),high(GPR,32) = COPIES loaded1
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = t2ANDrr (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land,
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- (G)MachineInstr to MachineInstr
  - NEW In-place morphing
  - NEW State expressed in the IR
  - State machine
  - Iterate until everything is selected
  - Combines across basic blocks

Select

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = t2LDRi12 (i32) addr
  loaded2(GPR,32) = t2LDRi12 (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1,
  low(GPR,32),high(GPR,32) = COPIES loaded1
  land(GPR,32) = t2ANDrr (i32) lval, low
  hand(GPR,32) = t2ANDrr (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land,
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

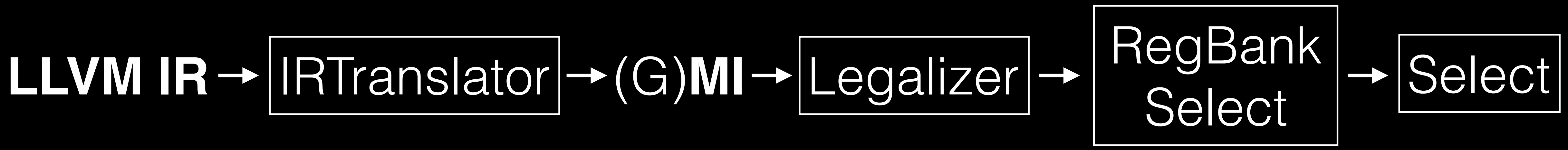
- (G)MachineInstr to MachineInstr
  - NEW In-place morphing
  - NEW State expressed in the IR
    - State machine
    - Iterate until everything is selected
  - NEW Combines across basic blocks

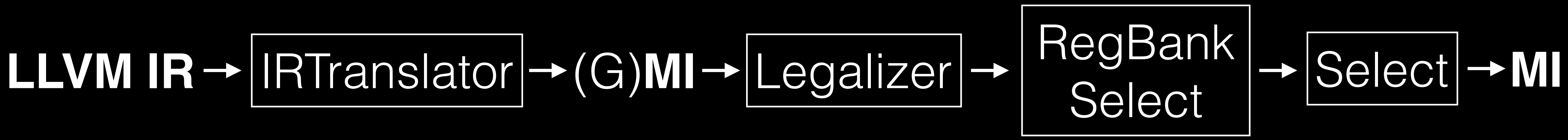
Select

```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = gLD (i32) addr
  loaded2(GPR,32) = gLD (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1, val2
  low(GPR,32),high(GPR,32) = COPIES loaded1,loaded2
  land(GPR,32) = gAND (i32) lval, low
  hand(GPR,32) = gAND (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land, hand
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

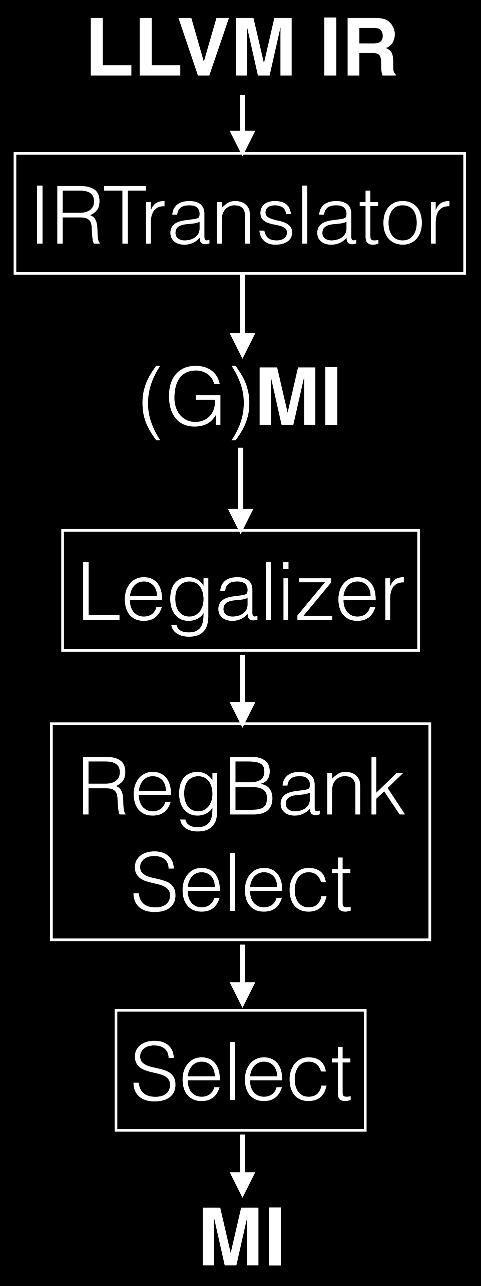
```
foo:
  val1(GPR,32),val2(GPR,32) = COPIES R0,R1
  addr(GPR,32) = COPY R2
  loaded1(GPR,32) = t2LDRi12 (i32) addr
  loaded2(GPR,32) = t2LDRi12 (i32) addr, #4
  lval(GPR,32),hval(GPR,32) = COPIES val1,
  low(GPR,32),high(GPR,32) = COPIES loaded1
  land(GPR,32) = t2ANDrr (i32) lval, low
  hand(GPR,32) = t2ANDrr (i32) hval, high
  and1(GPR,32), and2(GPR,32) = COPIES land,
  R0,R1 = COPIES and1, and2
  tBX_RET R0<imp-use>,R1<imp-use>
```

- (G)MachineInstr to MachineInstr
  - NEW In-place morphing
  - NEW State expressed in the IR
    - State machine
    - Iterate until everything is selected
  - NEW Combines across basic blocks

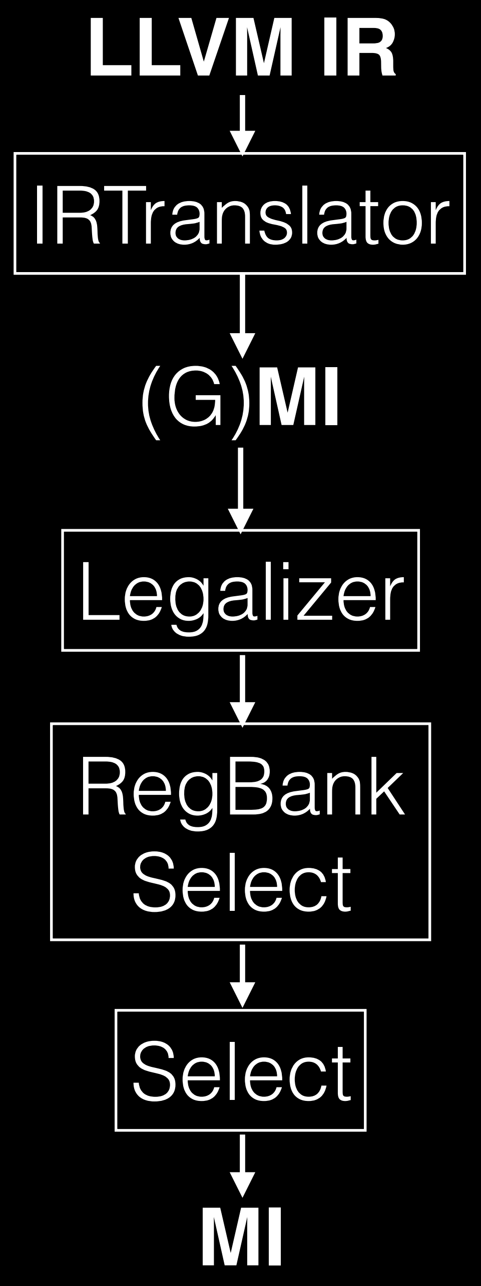

LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Select


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

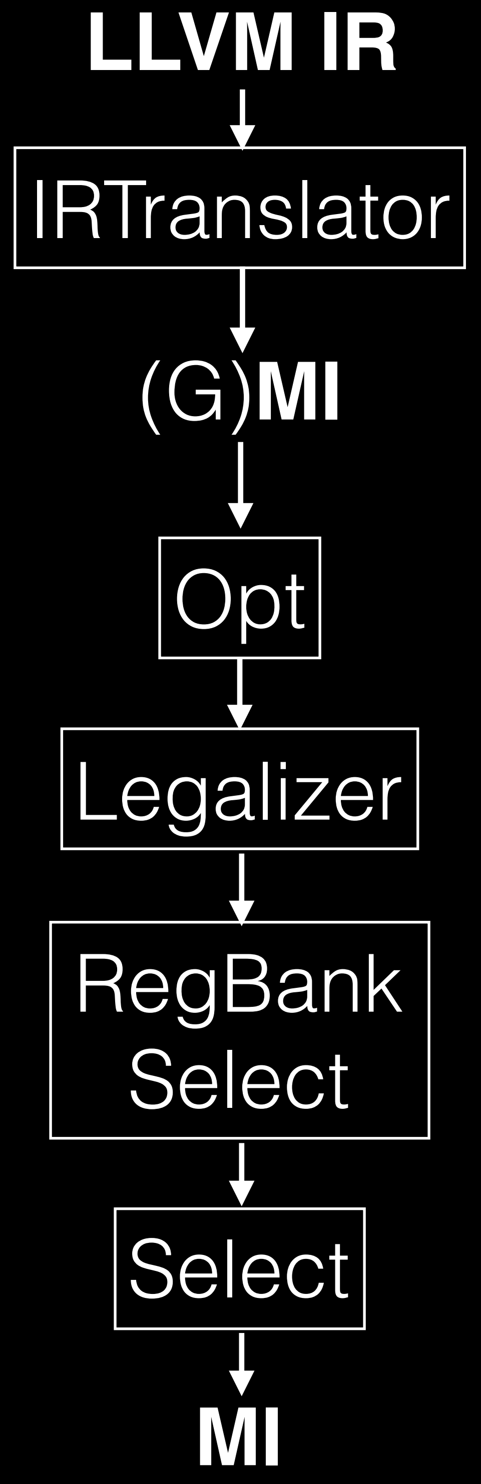
Summary


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

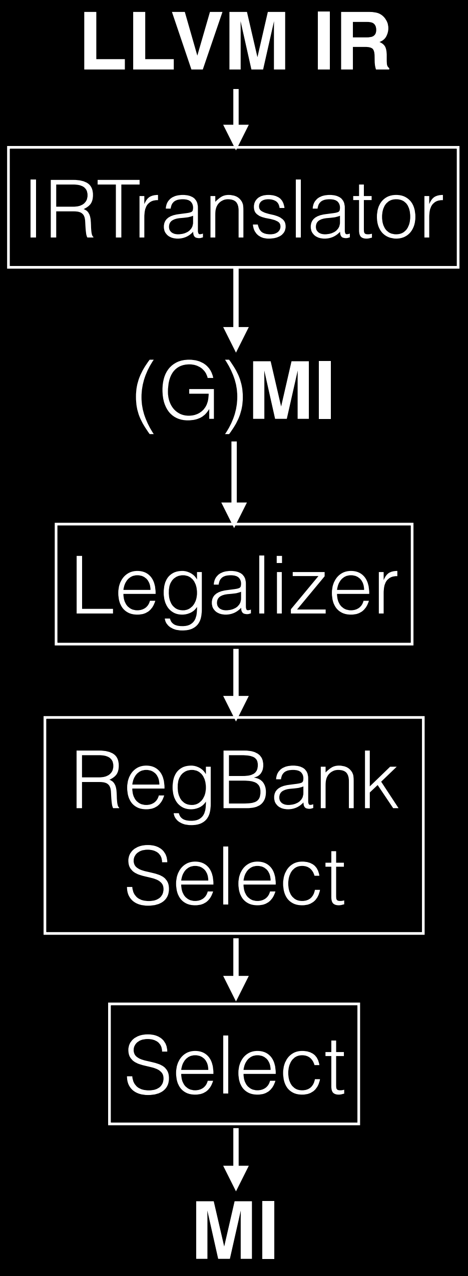
Summary


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

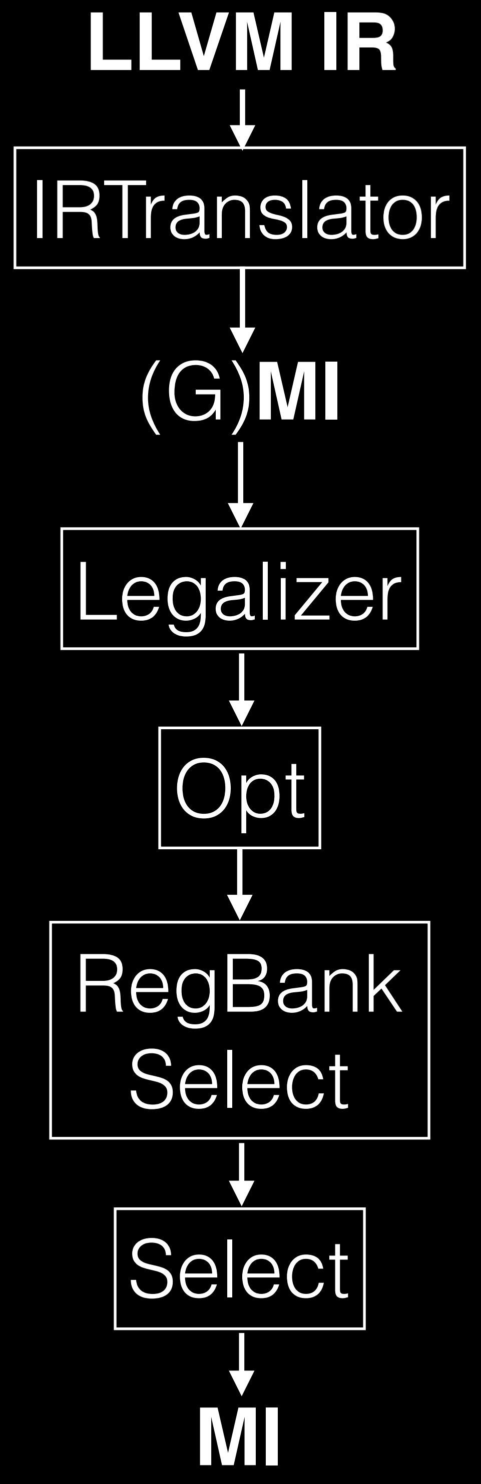
Summary


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

Summary

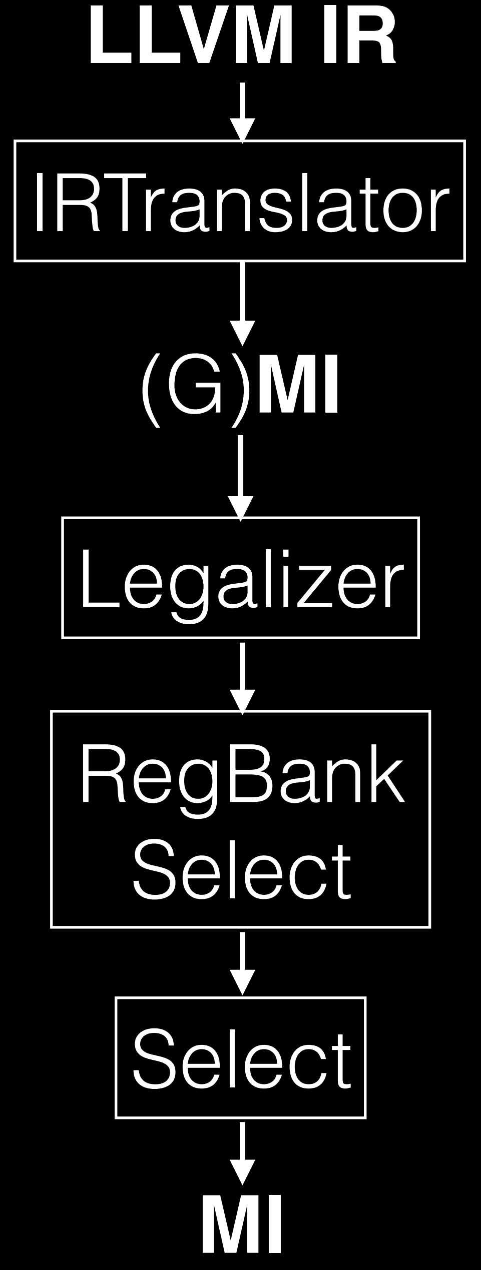

LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Select
MI

Summary

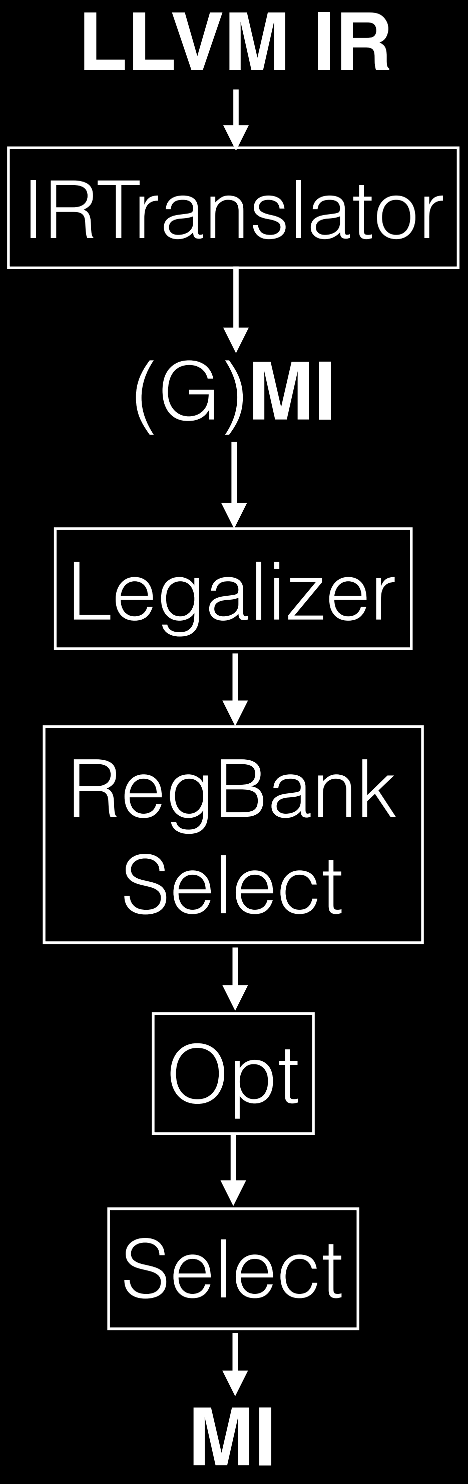


LLVM IR
IRTranslator
(G)MI
Legalizer
Opt
RegBank Select
Select
MI

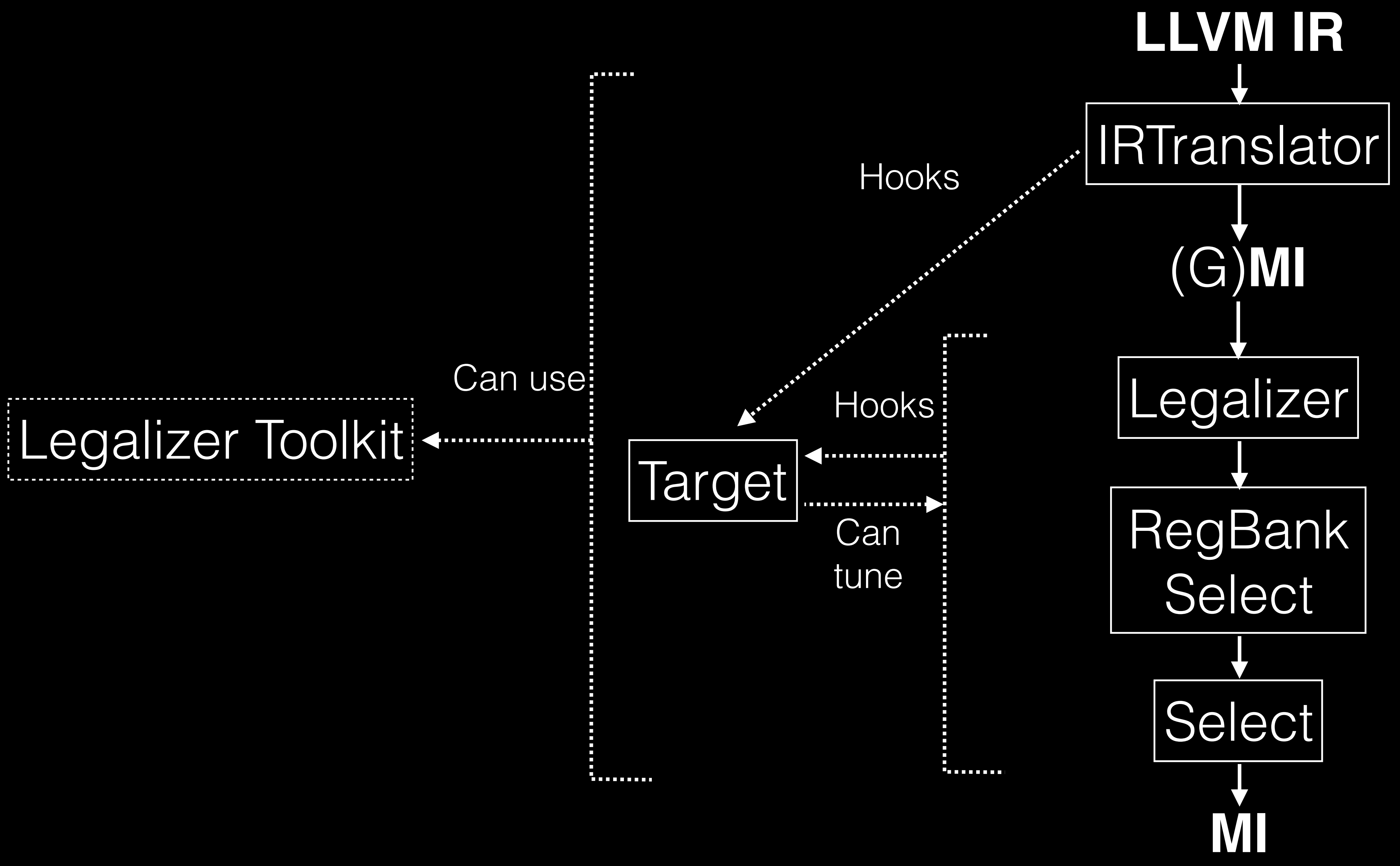
Summary


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

Summary

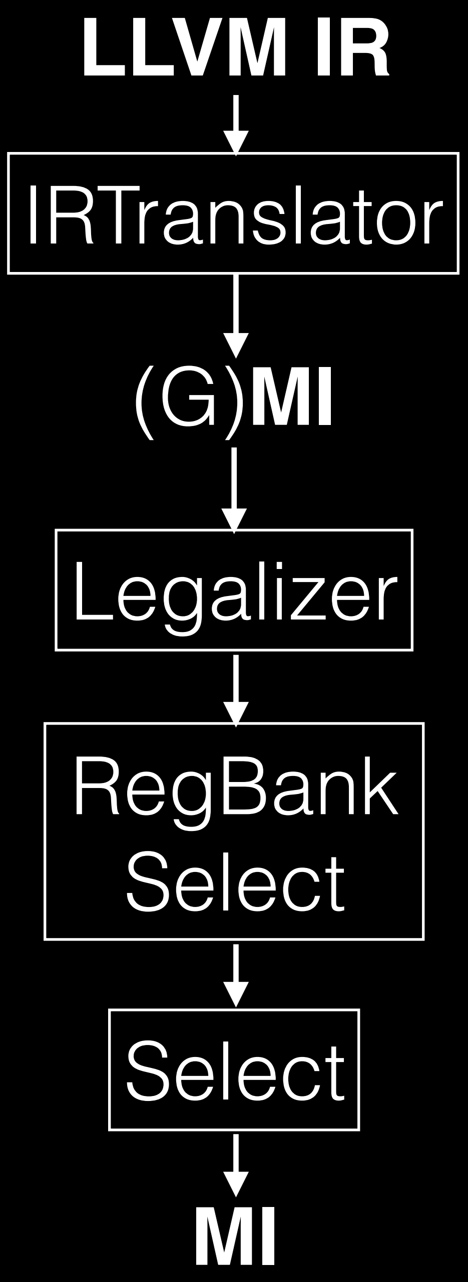

LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Opt
Select
MI

Summary


LLVM IR
IRTranslator
(G)MI
Legalizer
RegBank Select
Select
MI
Hooks
Hooks
Can tune
Target
Can use
Legalizer Toolkit

Global ISel

- Work at function scope
  - Global
- Break down the process in passes
  - More flexible pipeline
  - Easier to understand/maintain
  - Shared code for fast and good paths
- Introduce new generic opcodes for MachineInstr
  - Fast
  - IR that represents ISA concepts better
  - No change to LLVM IR
  - Self contained machine representation



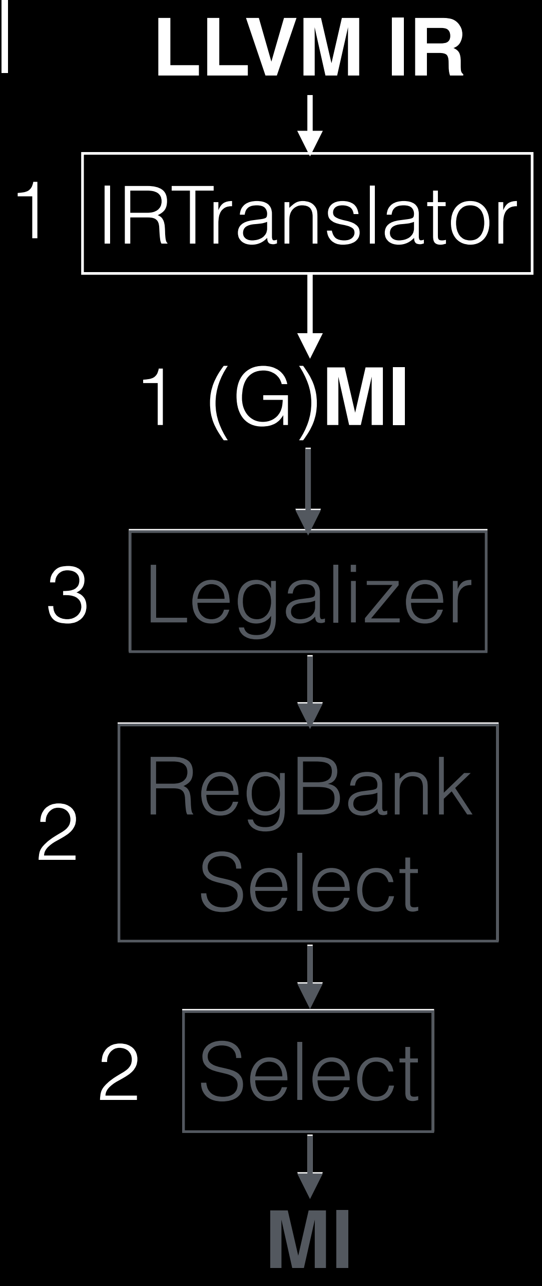
How Do We Get There?

1. Start of prototyping
1. Rename the SDISel into LegacyISel
    - Perform the translation
    - Lower the ABI
    - Complex instructions are not supported

2. Basic selector
    - Abort on illegal types
    - Selector patterns written in C++
    - Simple bank selection

3. Simple legalization
    - Scalar operations
    - Some vector operations

End of prototyping



Then What?

- Productize on what we learnt
- Discuss timeline to remove:
  - SDISel, FastISel
  - CodeGenPrepare, ConstantHoisting
  - ExeDepsFix, PeepholeOptimizer
- Present a status report next year

References

- Jakob’s initial proposal for global-isel: http://lists.llvm.org/pipermail/llvm-dev/2013-August/064696.html

Questions?