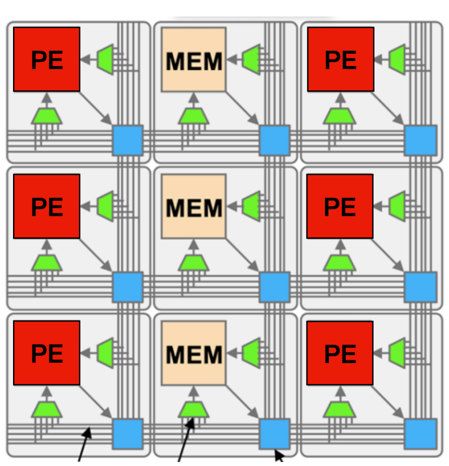
# Efficiently Synthesizing Lowest Cost Rewrite Rules for Instruction Selection

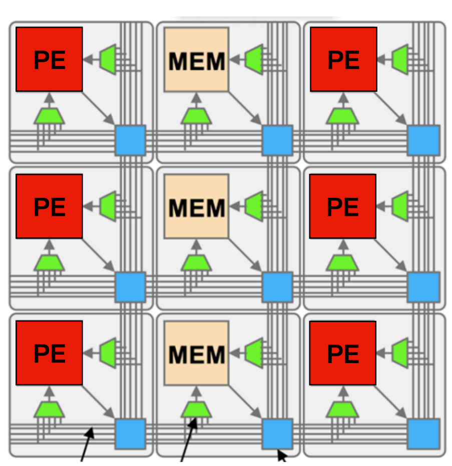
Ross Daly

# Create Domain Specific Accelerators by *Specializing* Processors

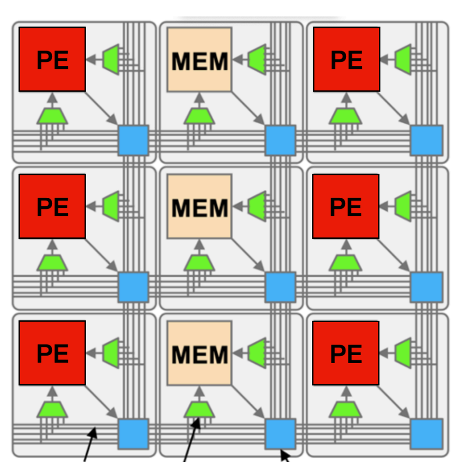
**Image Processing CGRA**




**ML Training CGRA**




**Large Language Model CGRA??**

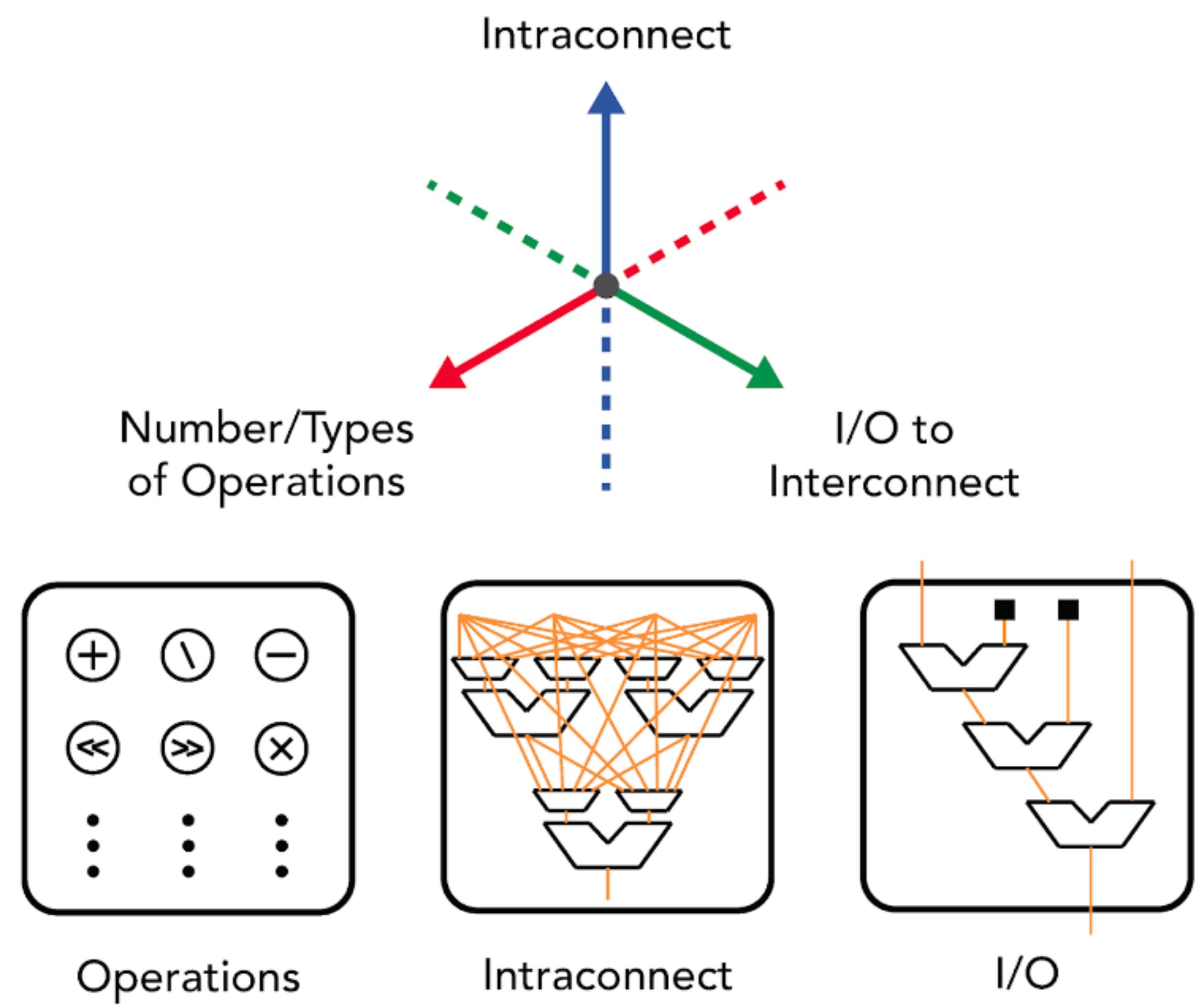



...

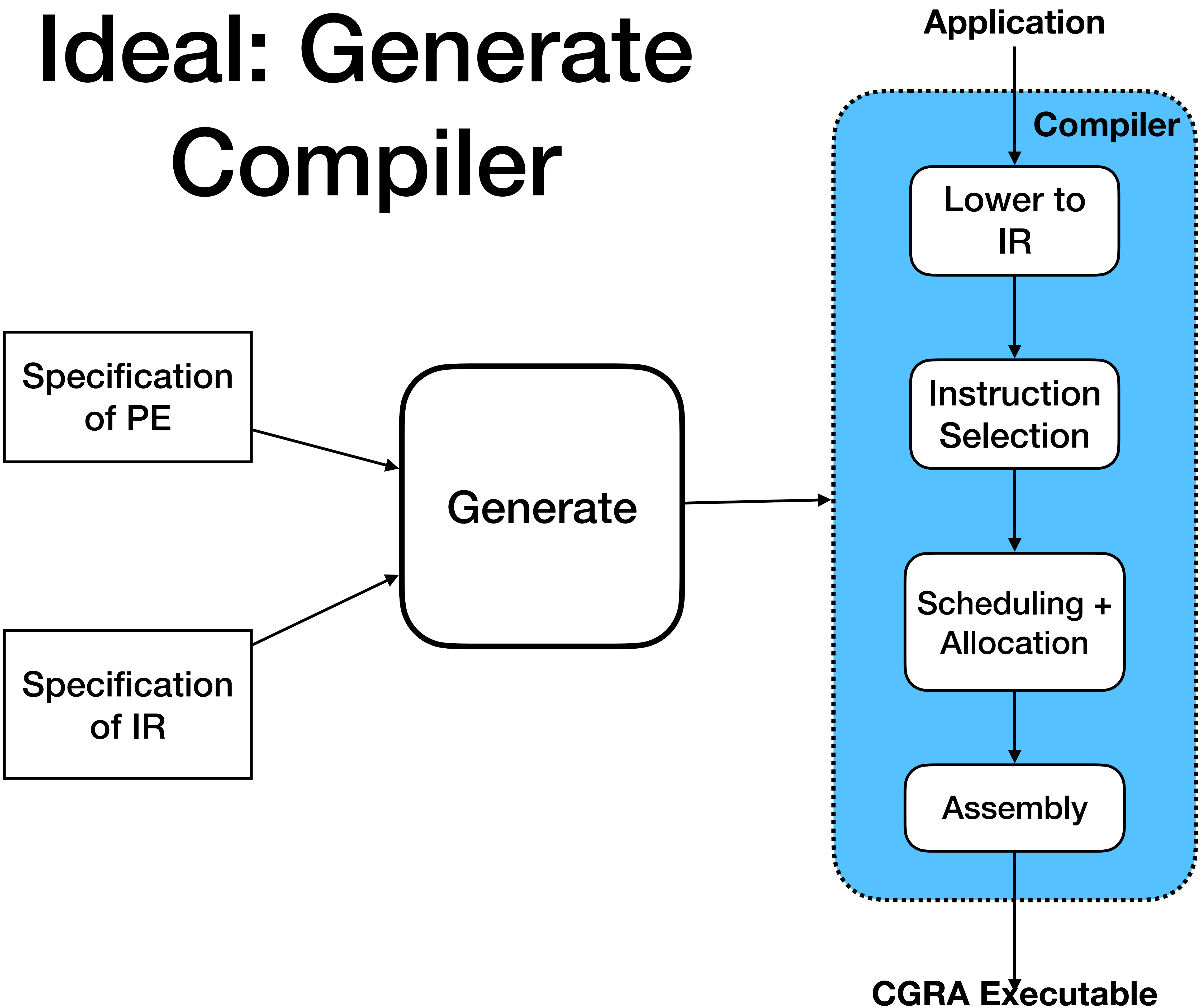
# Each Version Requires a Custom Compiler

# Processing Element (PE) Design Space Exploration

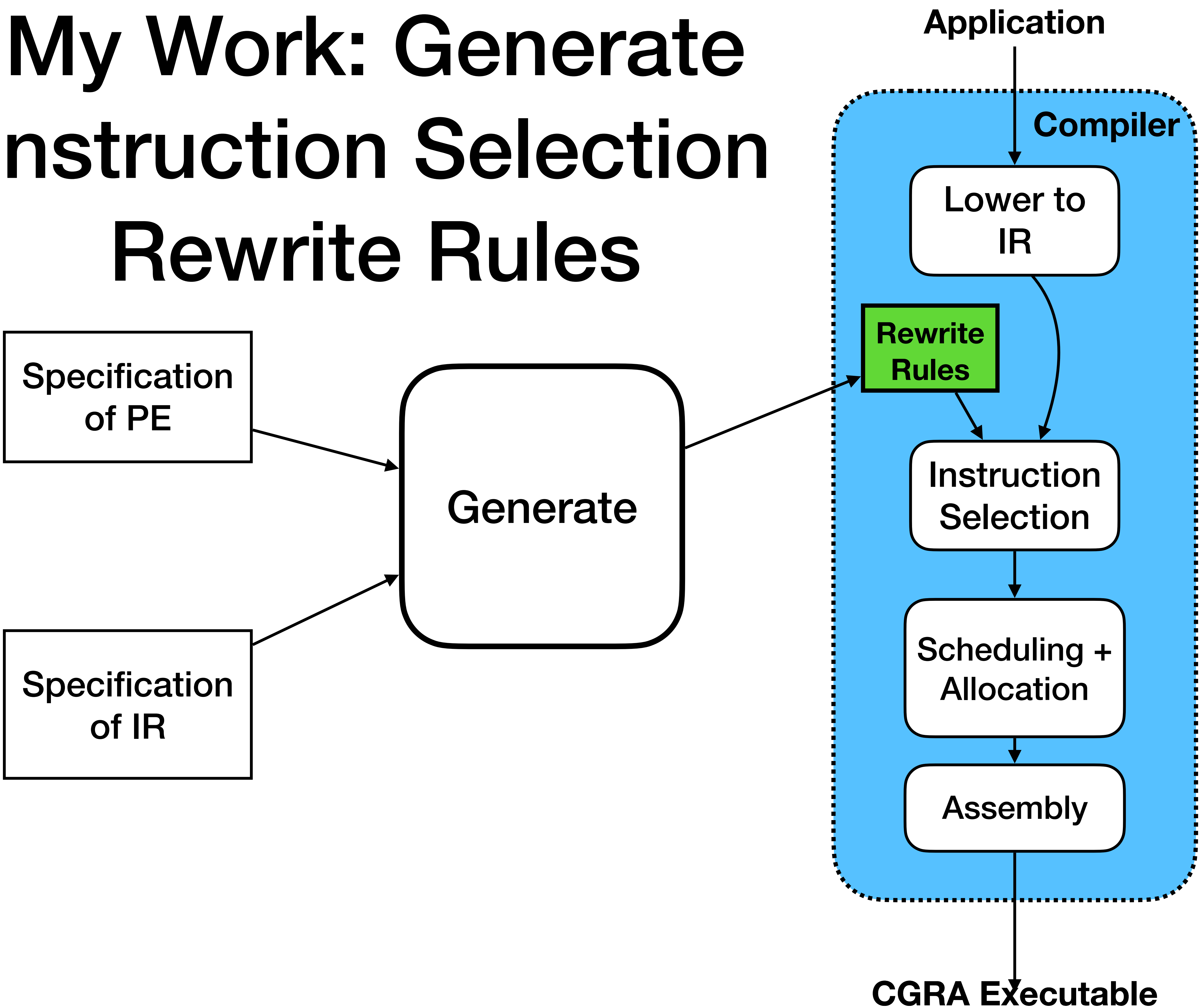
- **For a domain**: Large space of possible PE designs
- Generate 100s of PEs
- Benchmark and compare each PE
  - **Requires a working compiler per PE!**



# Ideal: Generate Compiler

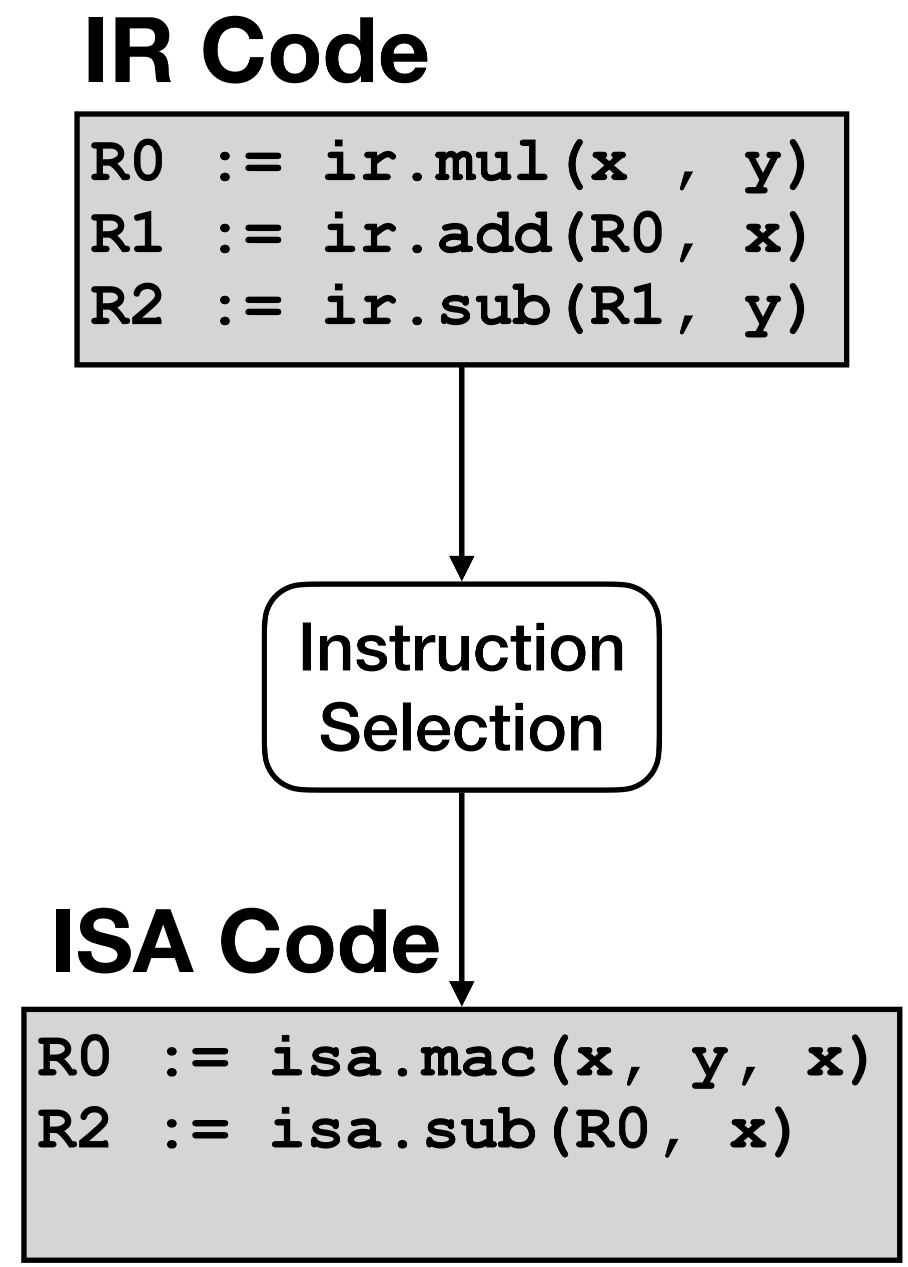


# My Work: Generate Instruction Selection Rewrite Rules



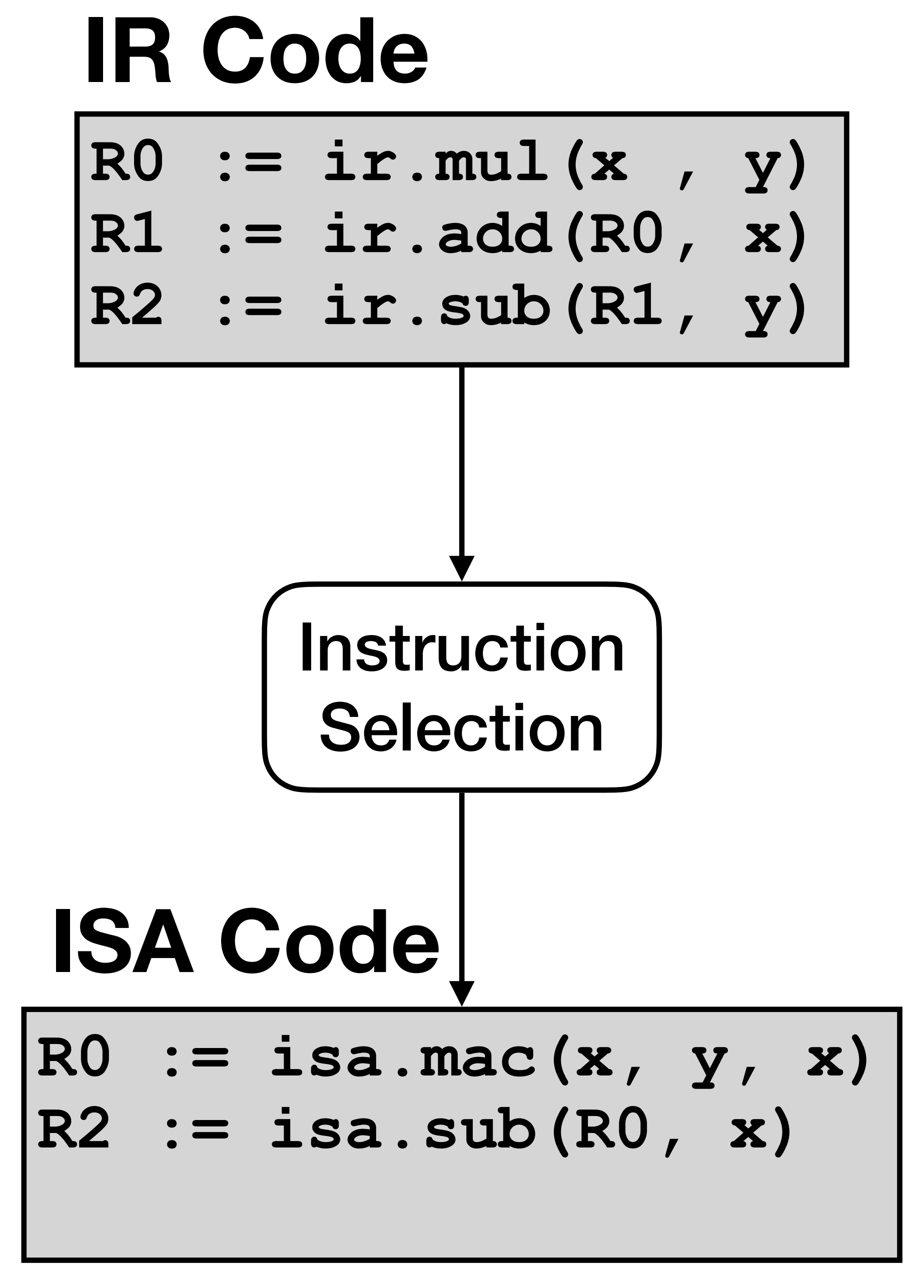
# Instruction Selection
# (i.e. ‘Mapping’)

- Instruction Selection Task:
    - Translate intermediate instructions (IR) to architecture-specific instructions (ISA)
    - Translation must preserve equivalence
    - Produce optimal ISA code



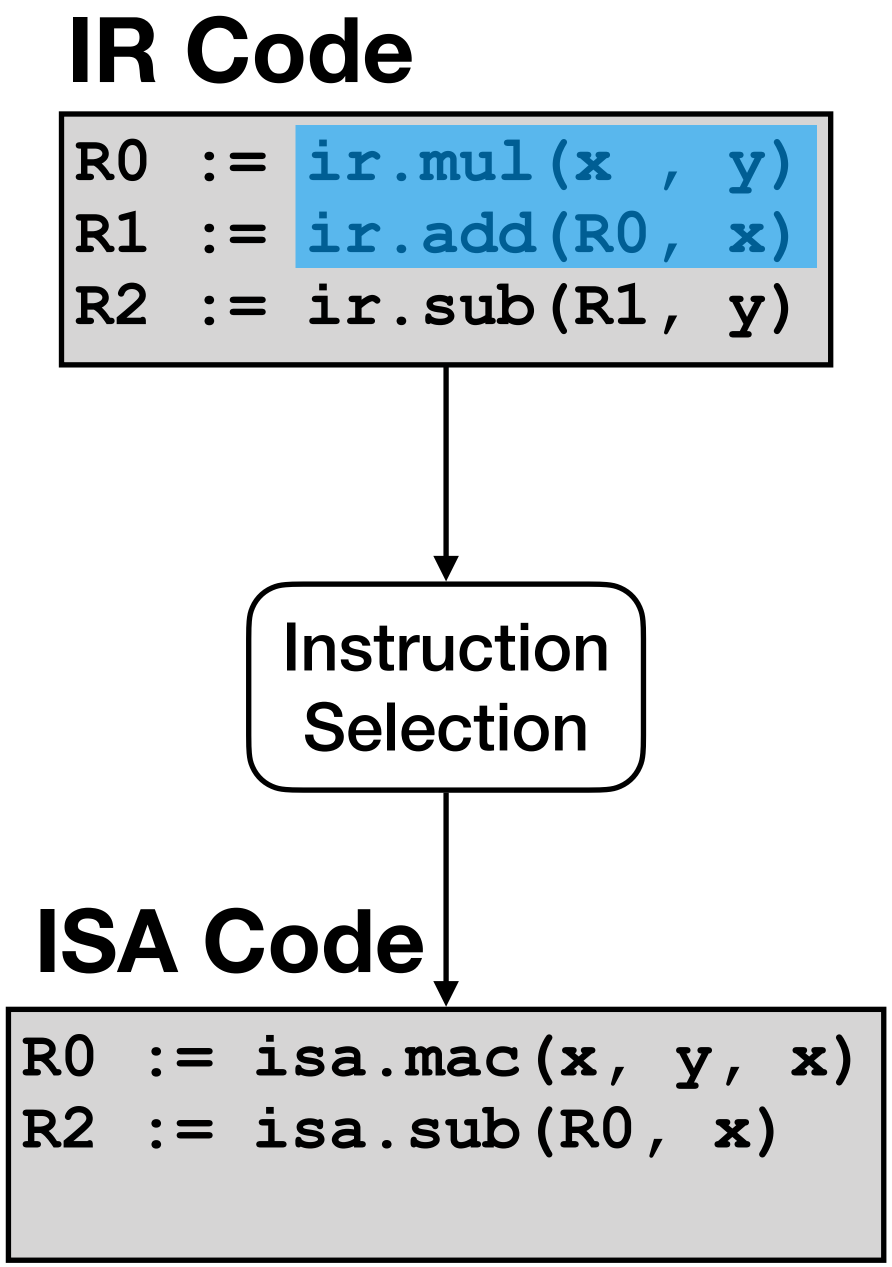
# Rewrite Rules

| IR Pattern | ISA Pattern | Cost |
|---|---|---|
| ir.add(x,y) | isa.add(x,y) | 1 |
| ir.sub(x,y) | isa.sub(y,x) | 1 |
| ir.mul(x,y) | isa.mul(x,y) | 1 |
| ir.add(x,<br>  ir.mul(y,z)<br>) | isa.mac(x,y,z) | 1 |



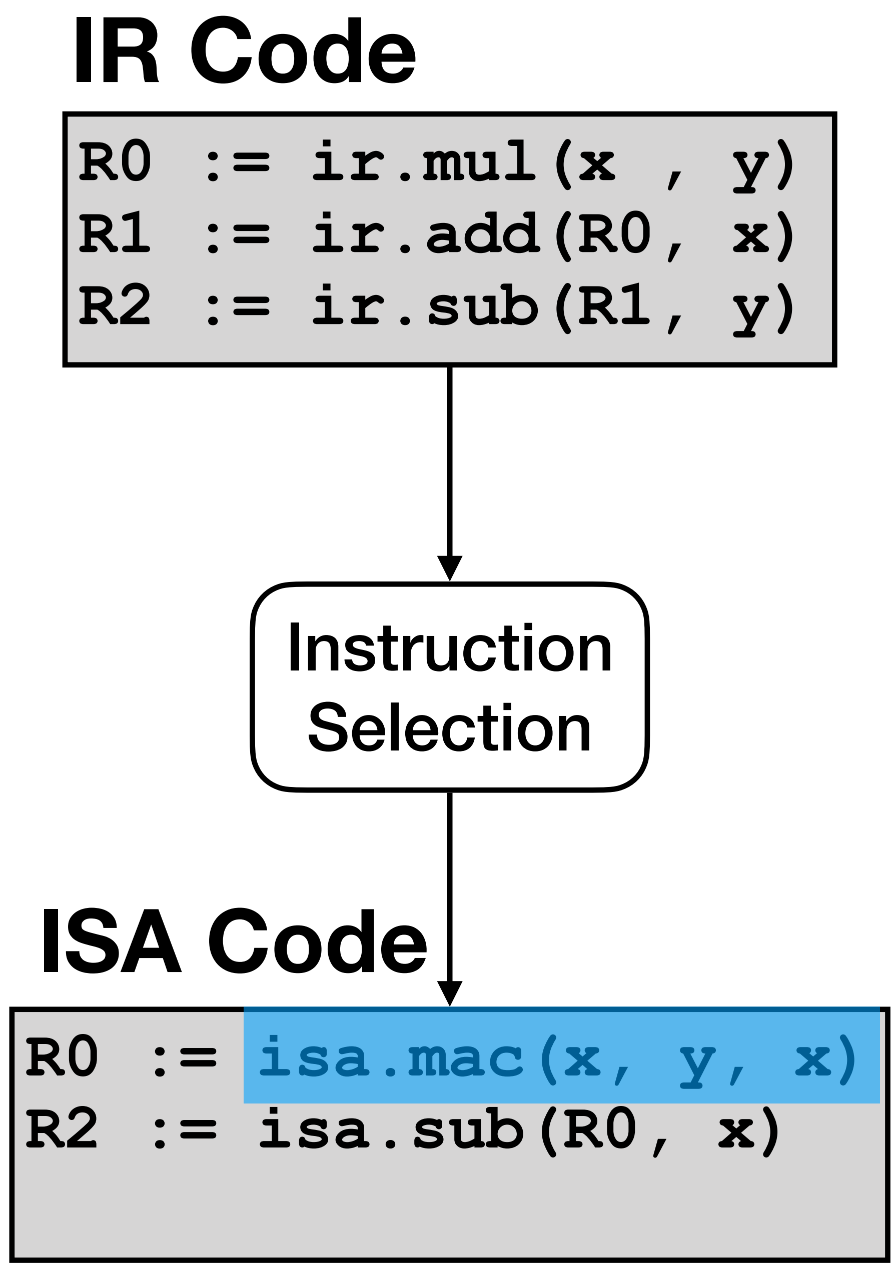
# Rewrite Rules

| IR Pattern | ISA Pattern | Cost |
|---|---|---|
| ir.add(x,y) | isa.add(x,y) | 1 |
| ir.sub(x,y) | isa.sub(y,x) | 1 |
| ir.mul(x,y) | isa.mul(x,y) | 1 |
| ir.add(x, ir.mul(y,z) ) | isa.mac(x,y,z) | 1 |



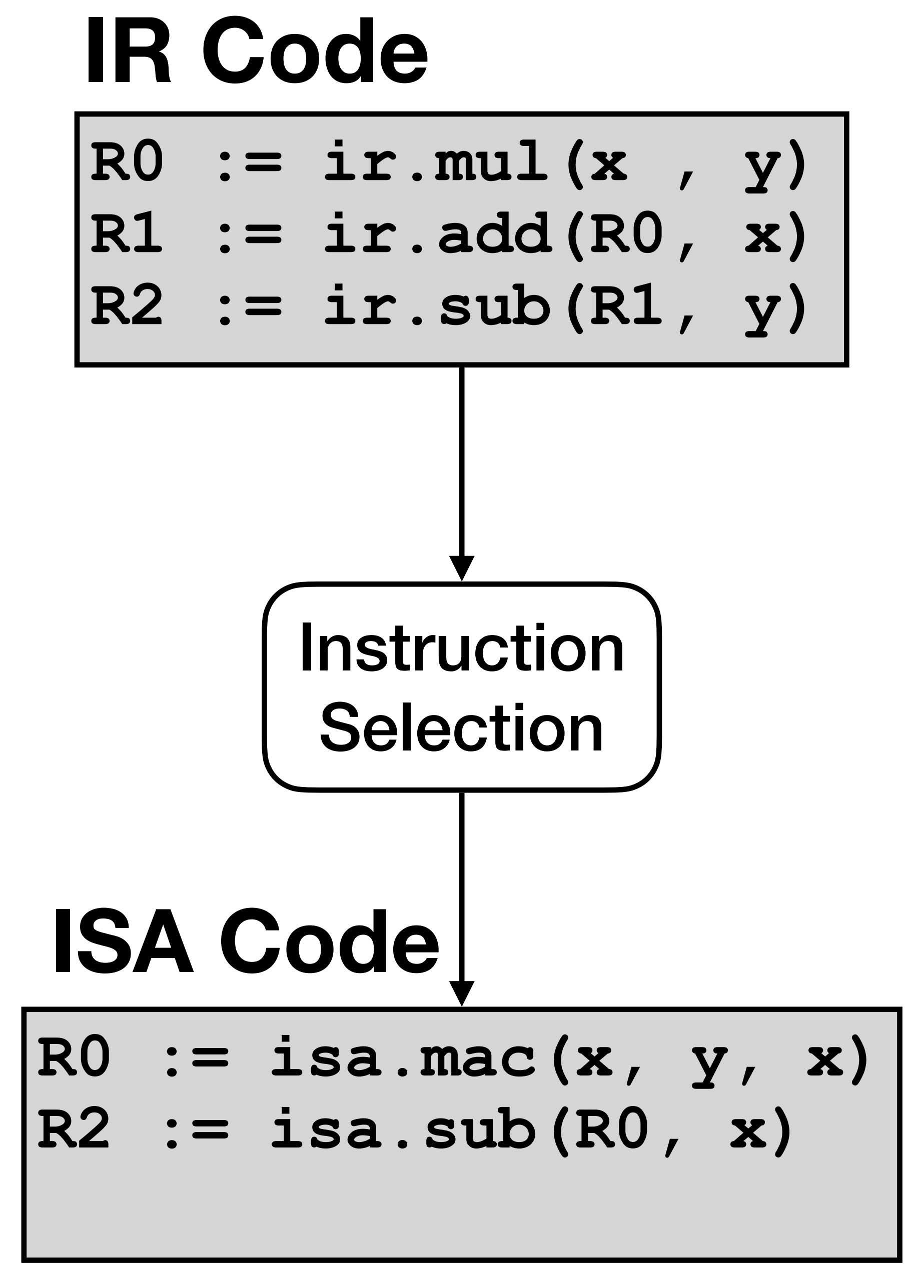
# Rewrite Rules

| IR Pattern | ISA Pattern | Cost |
|---|---|---|
| `ir.add(x,y)` | `isa.add(x,y)` | 1 |
| `ir.sub(x,y)` | `isa.sub(y,x)` | 1 |
| `ir.mul(x,y)` | `isa.mul(x,y)` | 1 |
| `ir.add(x,`<br>`ir.mul(y,z)`<br>`)` | `isa.mac(x,y,z)` | 1 |



# Rewrite Rules

| IR Pattern | ISA Pattern | Cost |
|---|---|---|
| `ir.add(x,y)` | `isa.add(x,y)` | 1 |
| `ir.sub(x,y)` | `isa.sub(y,x)` | 1 |
| `ir.mul(x,y)` | `isa.mul(x,y)` | 1 |
| `ir.add(x, ir.mul(y,z) )` | `isa.mac(x,y,z)` | 1 |



# Large Diversity in Possible PEs

- ISA for the PE can be CISC-like compared to the IR
  - E.g. PE specialization using APEX Design Space Exploration
  - Requires “many-to-1” rewrite rules
    - Each IR pattern has multiple instructions

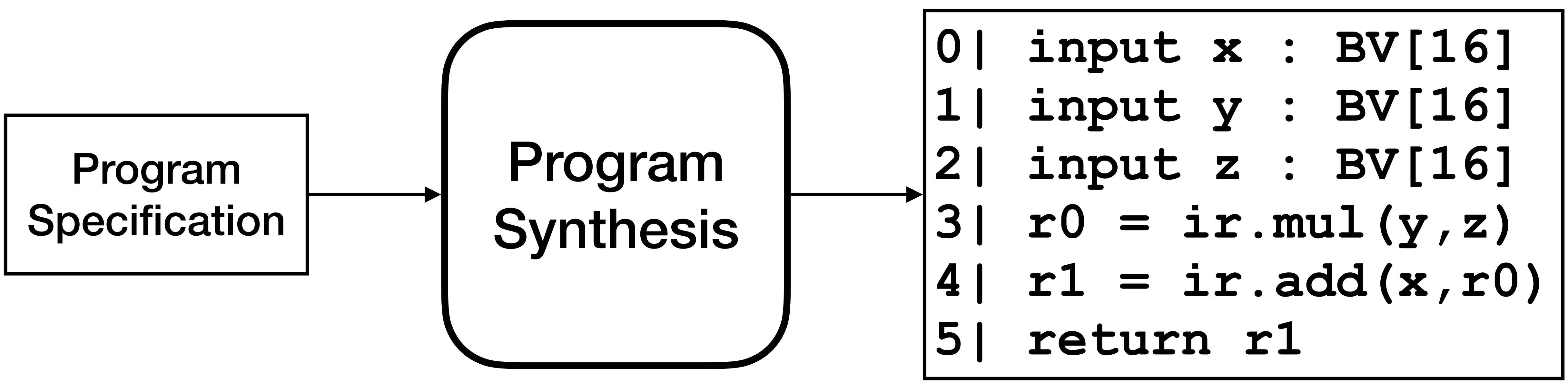
# Large Diversity in Possible PEs

- ISA for the PE can be RISC-like compared to the IR
  - Requires “1-to-many” rewrite rules
    - Each ISA pattern can has multiple instructions

A rewrite rule generation tool ***must*** be capable of generating 'many-to-many' rewrite rules

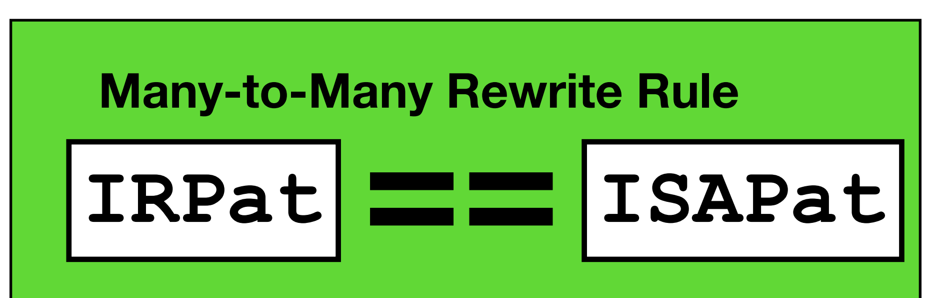
# How? Use Program Synthesis!

# Program Synthesis



# How to Synthesize a Many-to-Many Rewrite Rule?

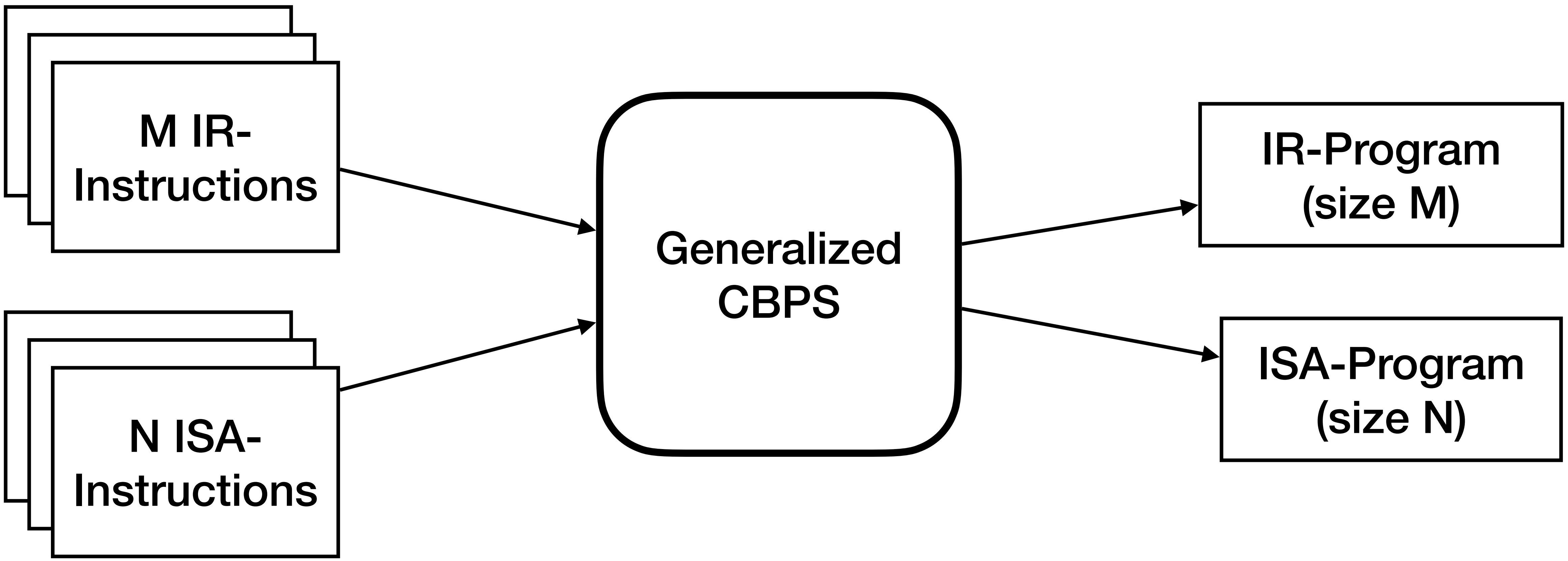
- Search for two patterns (IRPat, ISAPat)
- Such that:
  - IRPat is composed of 1 or more IR instructions
  - ISAPat is composed of 1 or more ISA instructions
  - IRPat is functionally equivalent to ISAPat



# Prior Work on Instruction Selection Rewrite Rule Synthesis

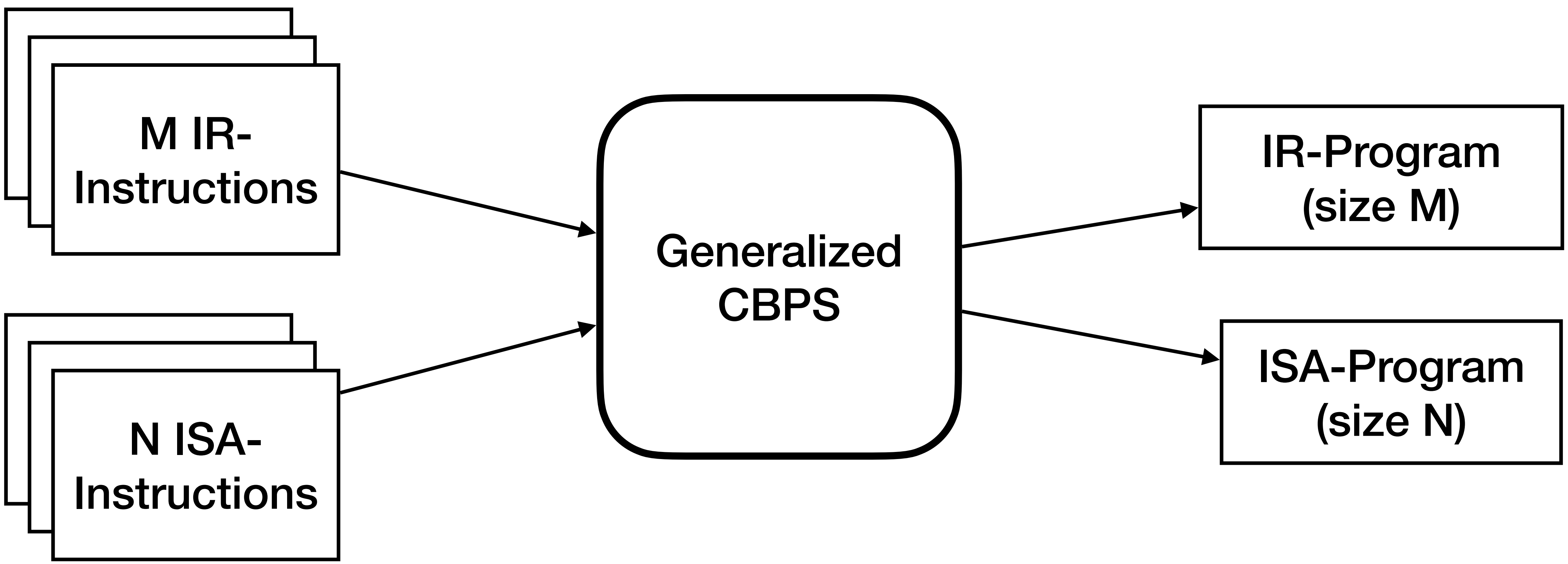
- Buchwald et al. Synthesizing an Instruction Selection Rule Library from Semantic Specifications
  - Uses Component-Based Program Synthesis to synthesize ‘many-to-1’ rewrite rules.
    - IR pattern with one or more instruction (many)
    - ISA pattern with single instruction (1)

# Generalized Component-Based Program Synthesis (GCBPS)



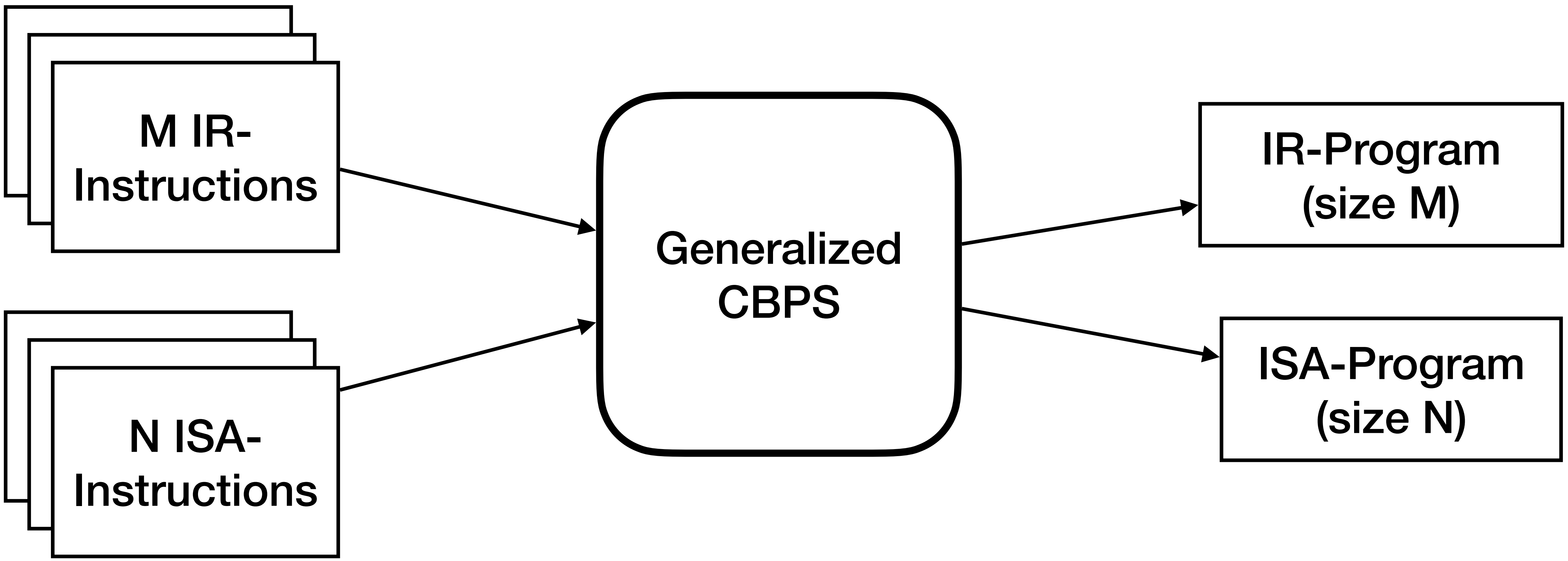

**New Program Synthesis Technique!**

# Generalized Component-Based Program Synthesis (GCBPS)




$$\phi_{synth} := \exists \mathbf{L}^a, \mathbf{L}^b . \forall \mathbf{I}^a, \mathbf{I}^b, O^a, O^b, \mathbf{W}^a, \mathbf{W}^b$$
$$\psi_{wfp}(\mathbf{L}^a) \wedge \psi_{wfp}(\mathbf{L}^b) \wedge$$
$$((\psi_{prog}(\mathbf{L}^a, \mathbf{I}^a, O^a, \mathbf{W}^a) \wedge \psi_{prog}(\mathbf{L}^b, \mathbf{I}^b, O^b, \mathbf{W}^b)) \implies$$
$$((\wedge_i \ \mathbf{I}_i^a = \mathbf{I}_i^b) \implies O^a = O^b))$$

# Solved using Counter Example Guided Inductive Synthesis (CEGIS)



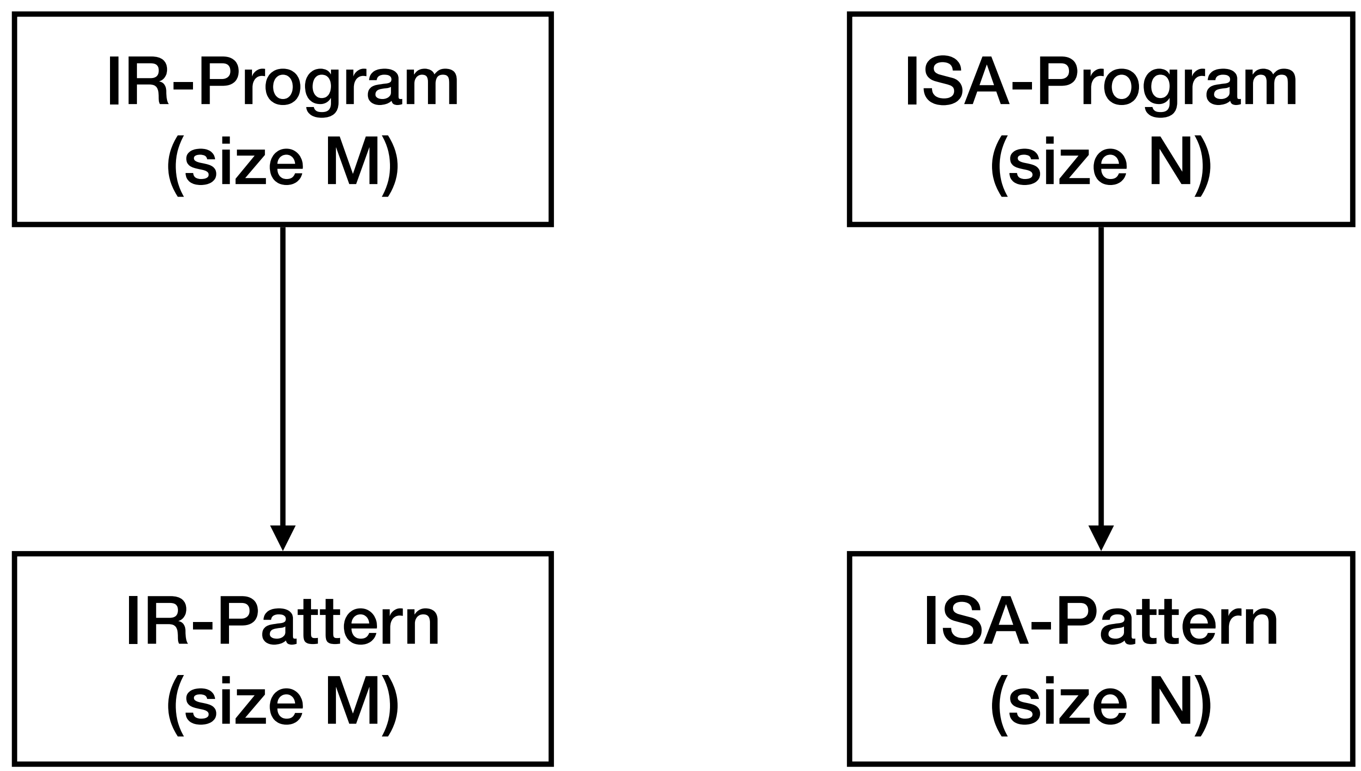

$$
\begin{aligned}
&\phi_{synth} := \exists \mathbf{L}^a, \mathbf{L}^b . \forall \mathbf{I}^a, \mathbf{I}^b, O^a, O^b, \mathbf{W}^a, \mathbf{W}^b \\
&\psi_{wfp}(\mathbf{L}^a) \wedge \psi_{wfp}(\mathbf{L}^b) \wedge \\
&\big((\psi_{prog}(\mathbf{L}^a, \mathbf{I}^a, O^a, \mathbf{W}^a) \wedge \psi_{prog}(\mathbf{L}^b, \mathbf{I}^b, O^b, \mathbf{W}^b)\big) \implies \\
&\big((\wedge_i \ \mathbf{I}_i^a = \mathbf{I}_i^b) \implies O^a = O^b\big)\big)
\end{aligned}
$$

# Given a Satisfying Solution

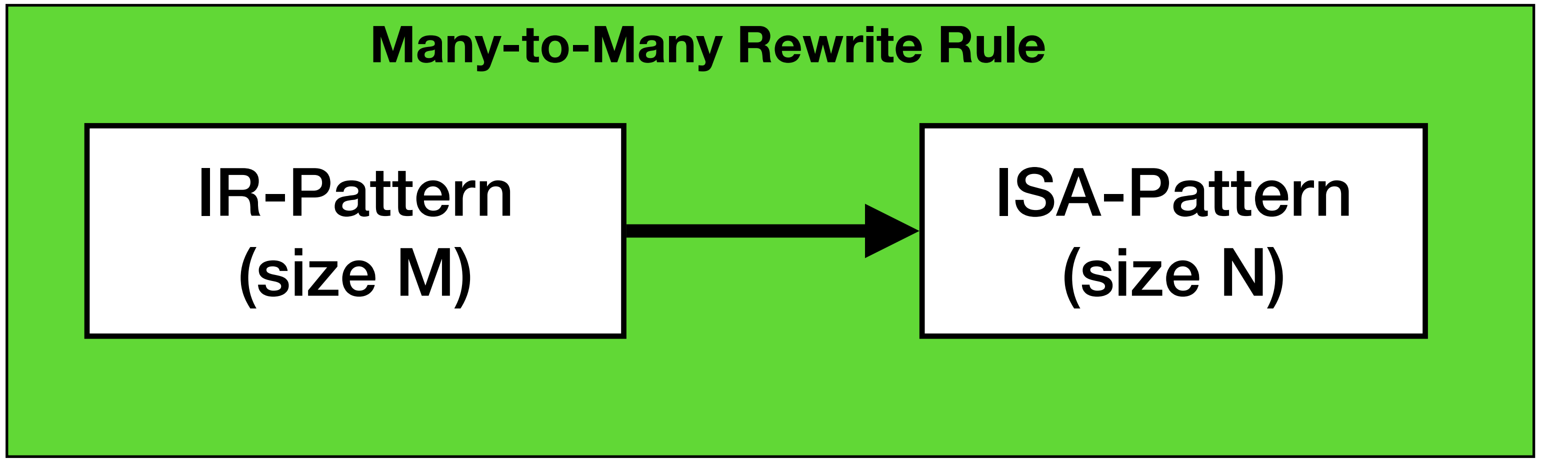
IR-Program
(size M)

ISA-Program
(size N)

# Programs are Interpreted as Patterns



# Pair of Patterns Interpreted as an Instruction Selection Rewrite Rule



# Complete: Synthesizing All Rewrite Rules

# Complete

- Why? Having a complete set of rewrite rules will improve the code quality of instruction selection!
- Iterative Algorithm (inspired by Buchwald et al.):
  - Iterate over all combinations of instructions
    - Iterate over all possible number of inputs
      - Find all satisfying solutions to GCBPS

# Number of Synthesized Many-to-Many Rewrite Rules

**Number of instructions in ISA Pattern** (columns) / **Number of instructions in IR Pattern** (rows)

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 17 | 71 | X | X |
| 2 | 89 | 3945 | X | X |

# Number of Synthesized Many-to-Many Rewrite Rules

**Number of instructions in ISA Pattern**

**Number of instructions in IR Pattern**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 17 | 71 | X | X |
| 2 | 89 | 3945 | X | X |

**Timed Out!**

# Problem: Are These the Same Rule?

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.sub(x,y)
3| return t0
```

=>

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.neg(y)
3| t1 = isa.add(x,t0)
4| return t1
```

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.sub(y,x)
3| return t0
```

=>

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.neg(x)
3| t1 = isa.add(y,t0)
4| return t1
```

# Yes! But x and y are Swapped

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.sub(x,y)
3| return t0
```

=>

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.neg(y)
3| t1 = isa.add(x,t0)
4| return t1
```

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.sub(y,x)
3| return t0
```

=>

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.neg(x)
3| t1 = isa.add(y,t0)
4| return t1
```

# Duplicates

- Different kinds of duplicates
  - Input variable permutations
  - Commutativity of instructions
  - Same-kind instruction
  - Topological sorts of data dependency graph

# Preventing Synthesis of Duplicates

- **Key idea**: Don’t use program synthesis to enumerate duplicates!
- Formally define **equivalence relations** for rewrite rules
- Updated Algorithm: Given a new rewrite rule:
    - All duplicates are programmatically enumerated and excluded from search space using that rules’ equivalence class.

# Number of Unique Many-to-Many Rewrite Rules

**Number of instructions in ISA Pattern**

**Number of instructions in IR Pattern**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 9 | 51 | 959 | X |
| 2 | 78 | 1004 | 19278 | X |

# Number of Unique Many-to-Many Rewrite Rules

**Number of instructions in ISA Pattern**

| Number of instructions in IR Pattern | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 9 | 51 | 959 | X |
| 2 | 78 | 1004 | 19278 | X |

**Does not time out!**

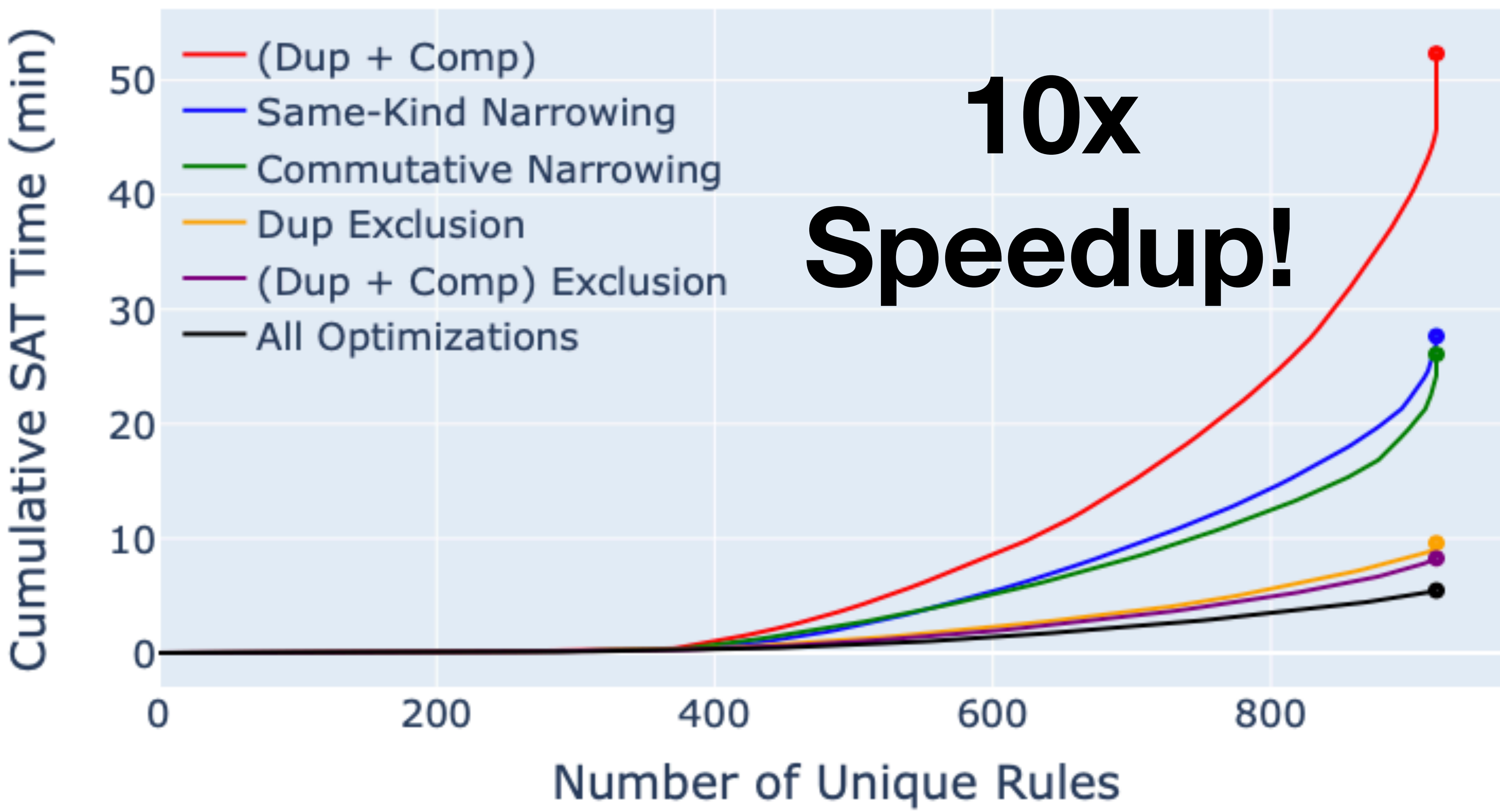
# Number of Unique Many-to-Many Rewrite Rules

Number of instructions in ISA Pattern

| Number of instructions in IR Pattern | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 9 | 51 | 959 | X |
| 2 | 78 | 1004 | 19278 | X |

**75% of synthesized of previous rewrite rules were duplicates! (3945 -> 1004)**

# Speedup by Removing Duplicates



# Problem: Different Rules for Same IR Pattern?

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.add(x,y)
3| return t0
```

=>

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.add(x,y)
4| return t1
```

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.add(x,y)
3| return t0
```

=>

```
0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.add(x,0)
3| t1 = isa.add(t0,y)
4| return t1
```

# High Cost Rules

- There are many valid rewrite rules for the same IR pattern
- Only one will be the lowest cost for a given cost metric
- All higher cost rules will ***never*** be useful in instruction selection

# Preventing Synthesis of High Cost Rewrite Rules

- **Key idea**: Don’t use program synthesis to enumerate high cost rules!
- **Key Idea**: Knowing instruction selection cost metric at synthesis time allows preventing synthesis of higher cost rules
- Try all lower cost ISA instruction combinations before higher cost ones
    - Guarantees the first rule found is lowest-cost
- Exclude IR patterns with a lowest cost rule from search space

# Number of Lowest-Cost Many-to-Many Rewrite Rules

**Number of instructions in ISA Pattern**

**Number of instructions in IR Pattern**

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 7 | 3 | 0 | 0 |
| 2 | 52 | 112 | 69 | 9 |

# Number of Lowest-Cost Many-to-Many Rewrite Rules

**Number of instructions in ISA Pattern** (columns); **Number of instructions in IR Pattern** (rows)

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 7 | 3 | 0 | 0 |
| 2 | 52 | 112 | 69 | 9 |

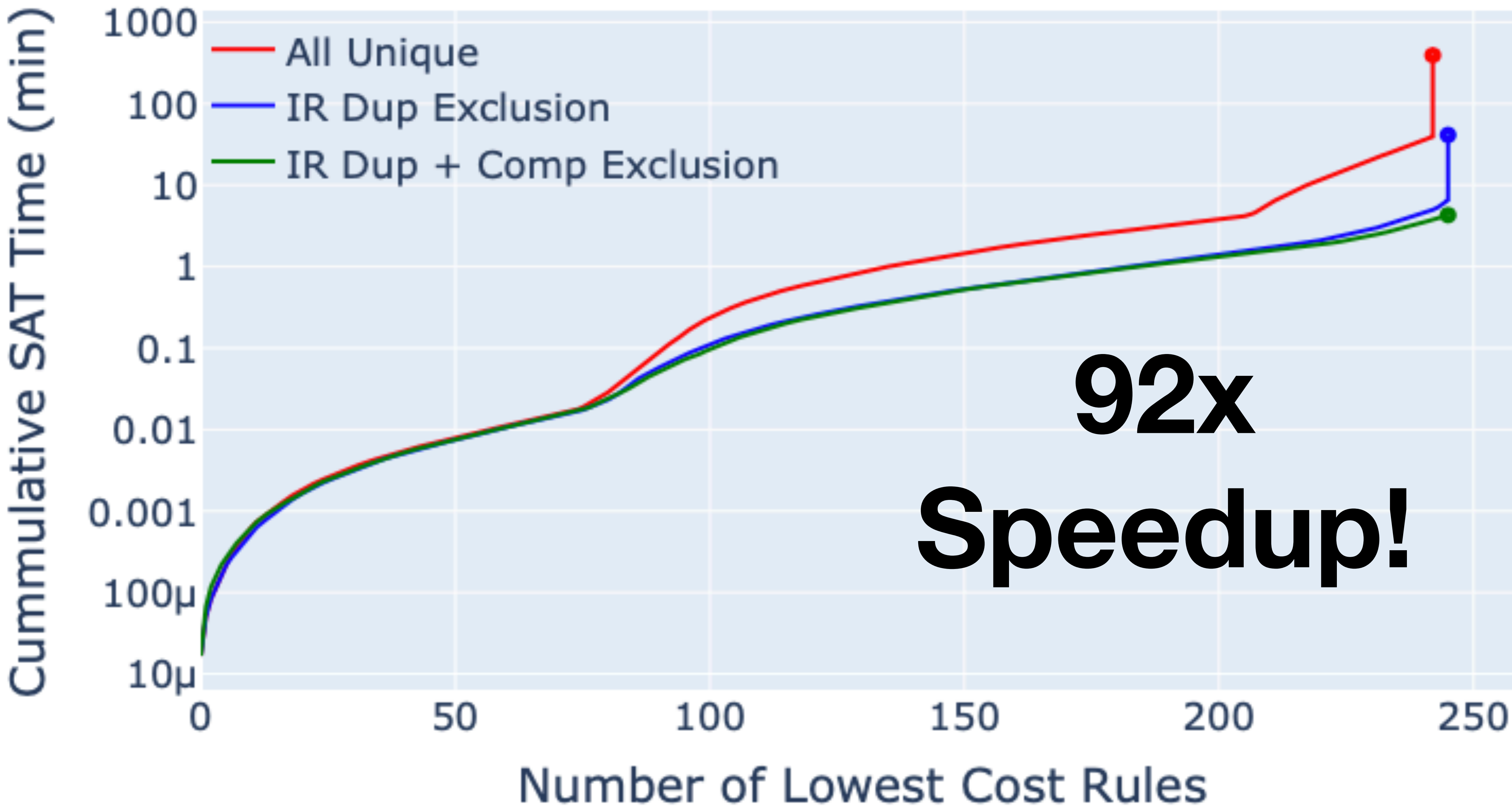
**Does not time out!**

# Number of Lowest-Cost Many-to-Many Rewrite Rules

| Number of instructions in IR Pattern \ Number of instructions in ISA Pattern | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 7 | 3 | 0 | 0 |
| 2 | 52 | 112 | 69 | 9 |

**99.6% of unique rewrite rules are high cost!! (19278 -> 69)**

# Speedup by Removing High Cost



# Different Cost Metrics Synthesize Different Rules

| ISA | Rewrite Rule Size (IR, ISA) | Unique (CS) | Unique (E) | Common |
|---|---|---|---|---|
| 1a | (2, 5) | 109 | 145 | 82 |
| 1b | (2, 4) | 147 | 182 | 105 |
| 2 | (3, 2) | 104 | 171 | 1099 |

**CS: Code Size**
**E: Energy**

# Summary

- New program syntheses technique to synthesize **many-to-many** rewrite rules for Instruction Selection

- Iterative algorithm to generate **complete** set of many-to-many rewrite rules

- Only **unique** rules using equivalence relations

    - Resulting in large speedups

- Only **lowest-cost** rules using a cost metric

    - Resulting in further large speedups

# A Necessary Step in Automatically Generating Application Compilers

# Questions?

# Backup