# Cross-Architecture Lifter Synthesis

Rijnard van Tonder and Claire Le Goues

Carnegie Mellon University

squaresLab
Software Quality in Real Evolving Systems

isr institute for SOFTWARE RESEARCH

# Cross-Architecture Lifter Synthesis

Rijnard van Tonder and Claire Le Goues

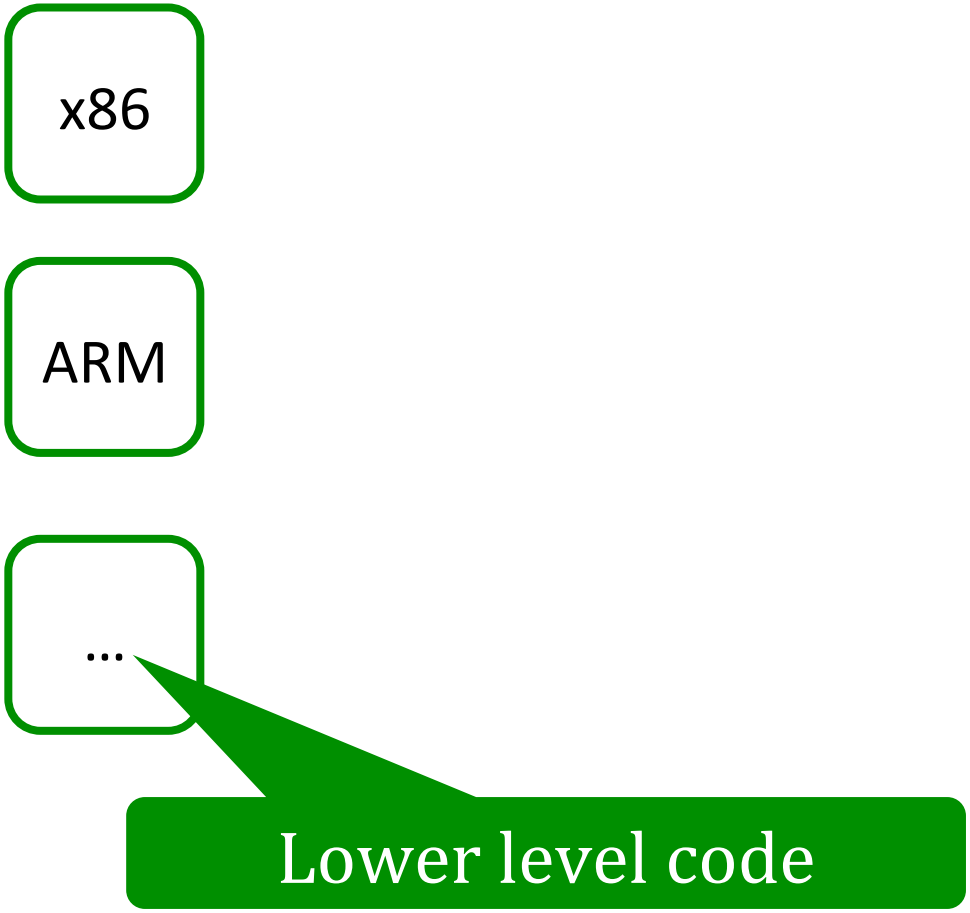
Carnegie Mellon University

squaresLab
Software Quality in Real Evolving Systems

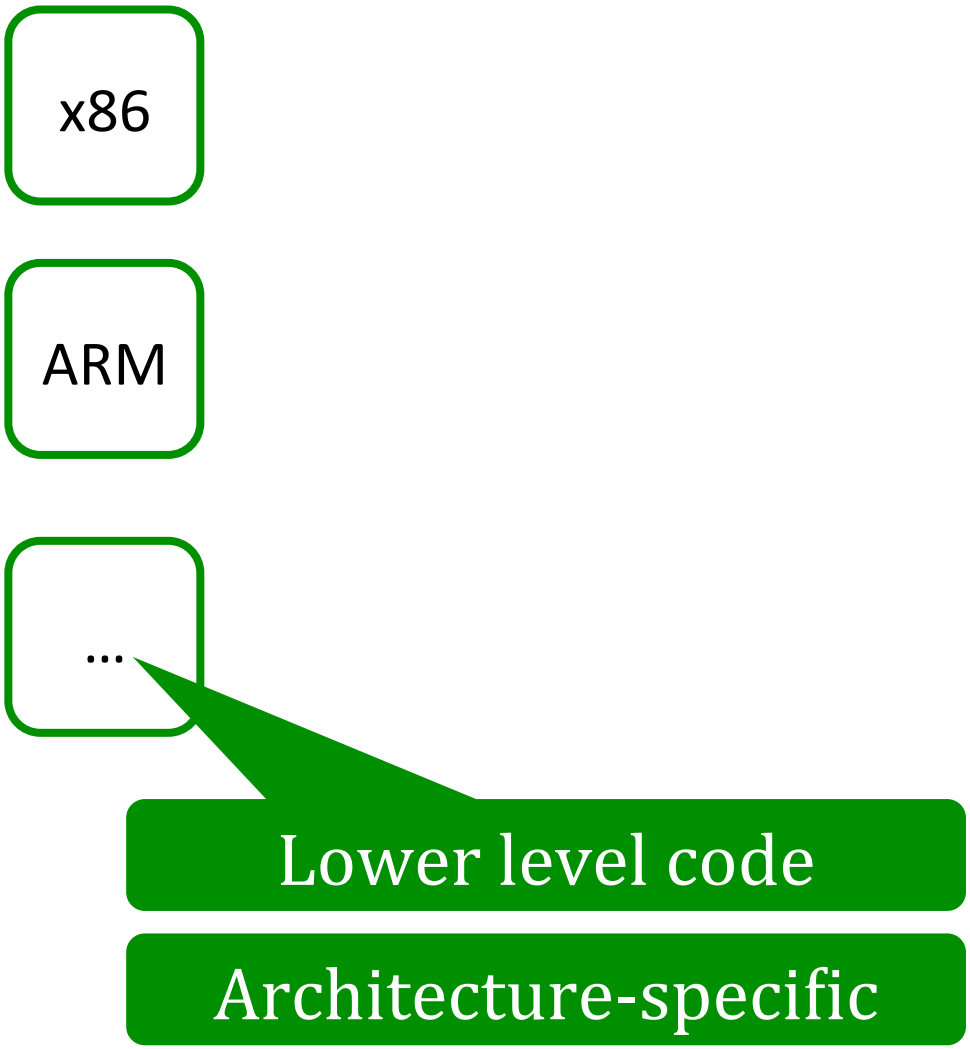
isr institute for SOFTWARE RESEARCH

# What is a Lifter?

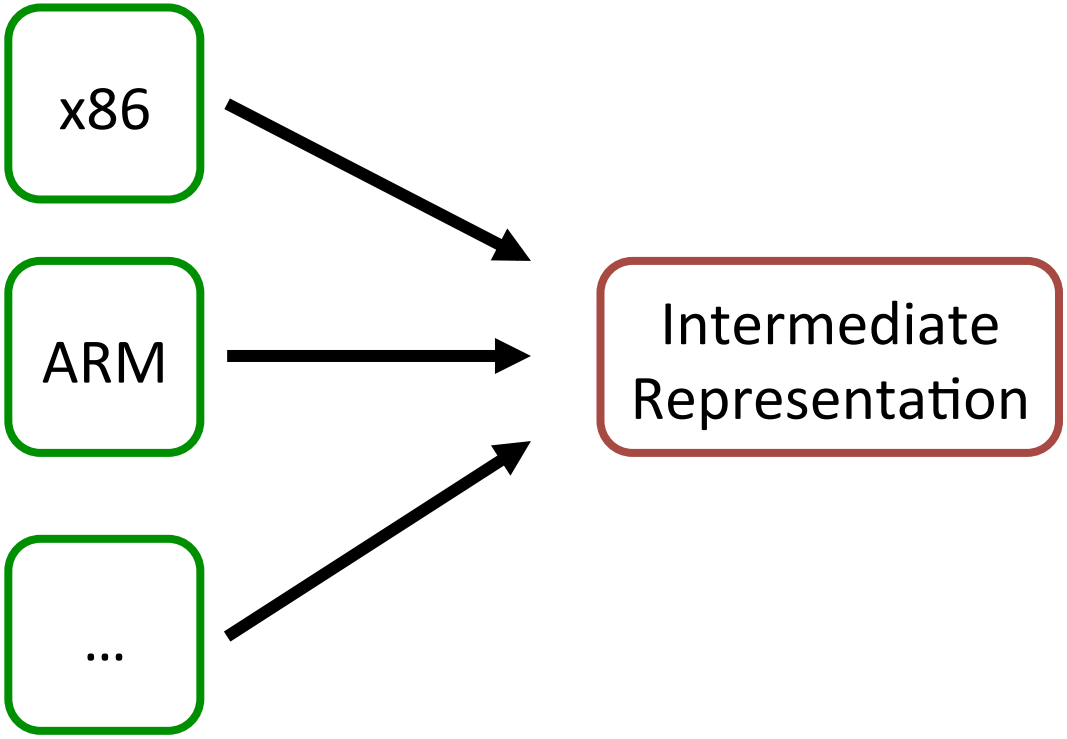
# What is a Lifter?



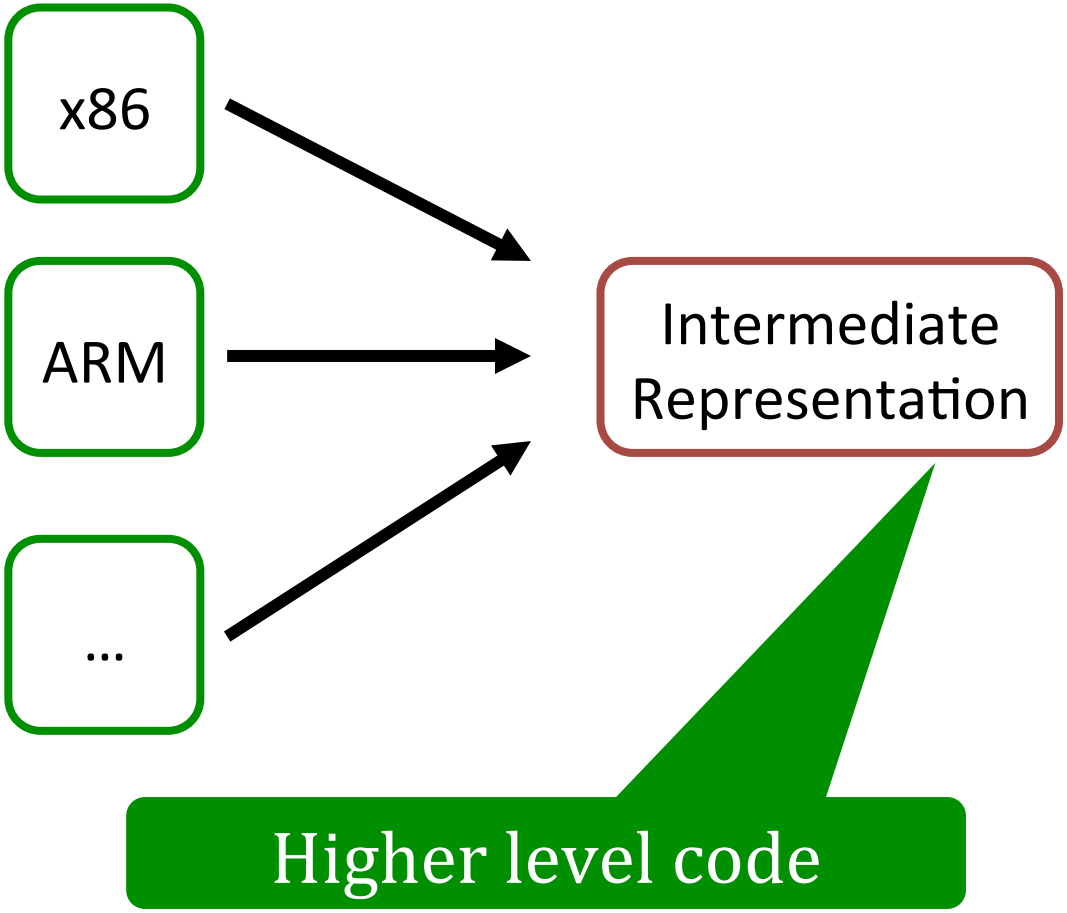
# What is a Lifter?



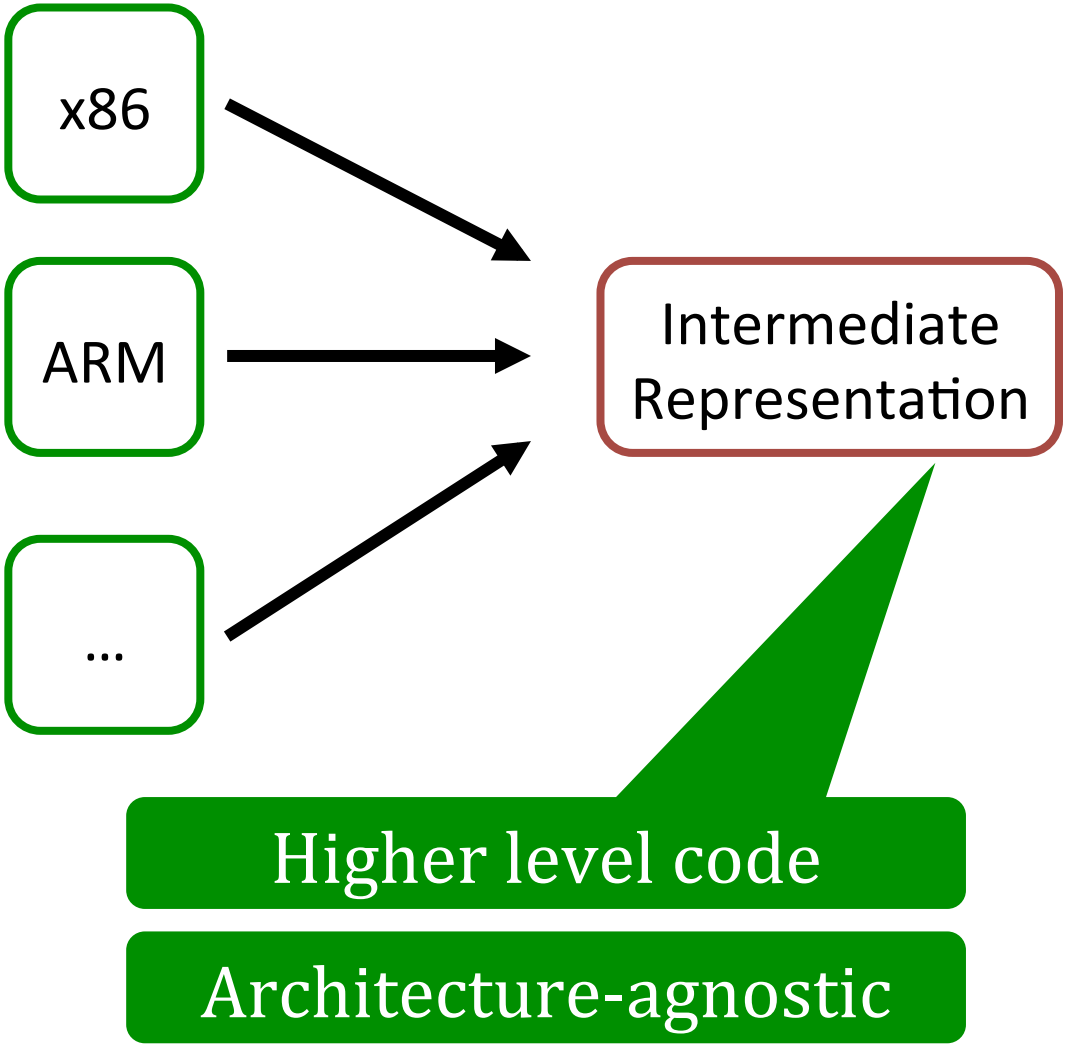
# What is a Lifter?



# What is a Lifter?



# What is a Lifter?



# Why Lifters?

# Why Lifters?

- Binary analysis

# Why Lifters?

- Binary analysis
- Simpler semantic abstraction
  - E.g., symbolic execution

# Why Lifters?

- Binary analysis
- Simpler semantic abstraction
  - E.g., symbolic execution
- Reuse analysis components
  - Control flow graph construction
  - Single Static Assignment (SSA) conversion

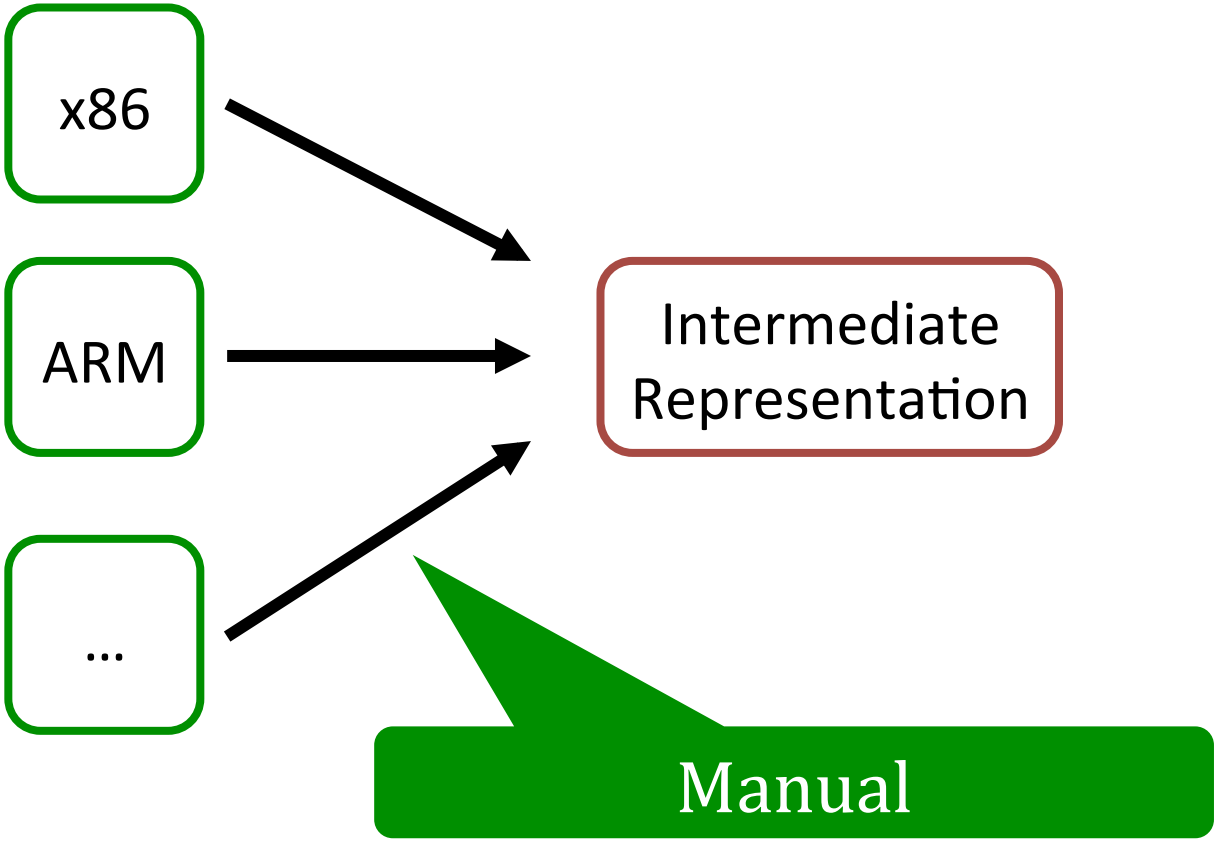
# Compilers and Decompilers

- Compilers lose information
  - Translation is not a bijection

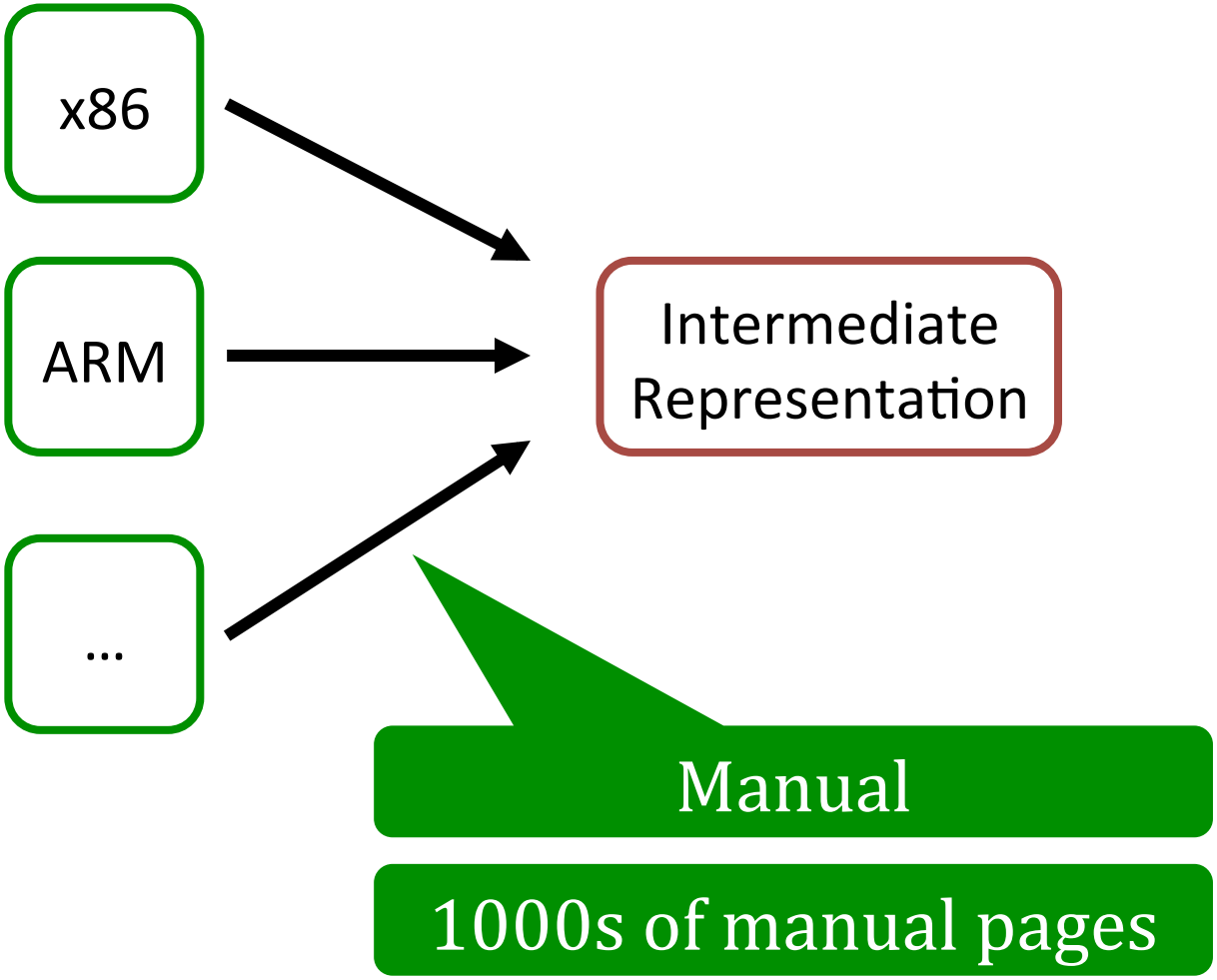
# Compilers and Decompilers

- Compilers lose information
  - Translation is not a bijection
- Decompilers recover more features
  - Often architecture-specific (e.g., ABI)

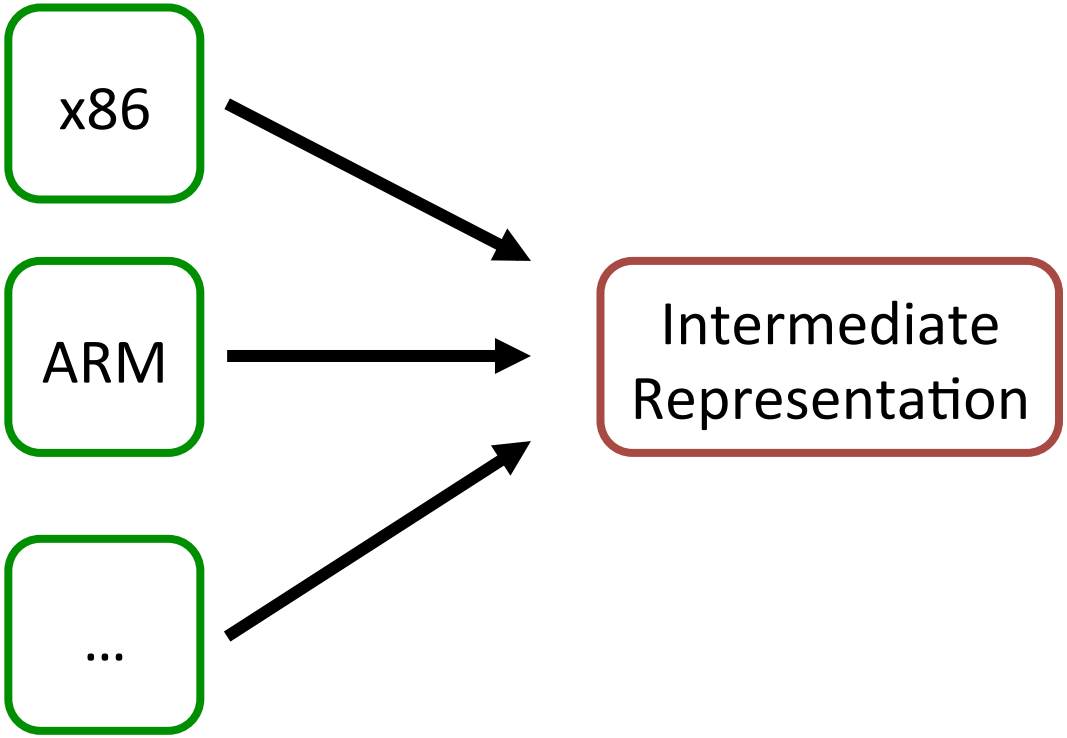
# The Problem: Translating Multiple Architectures



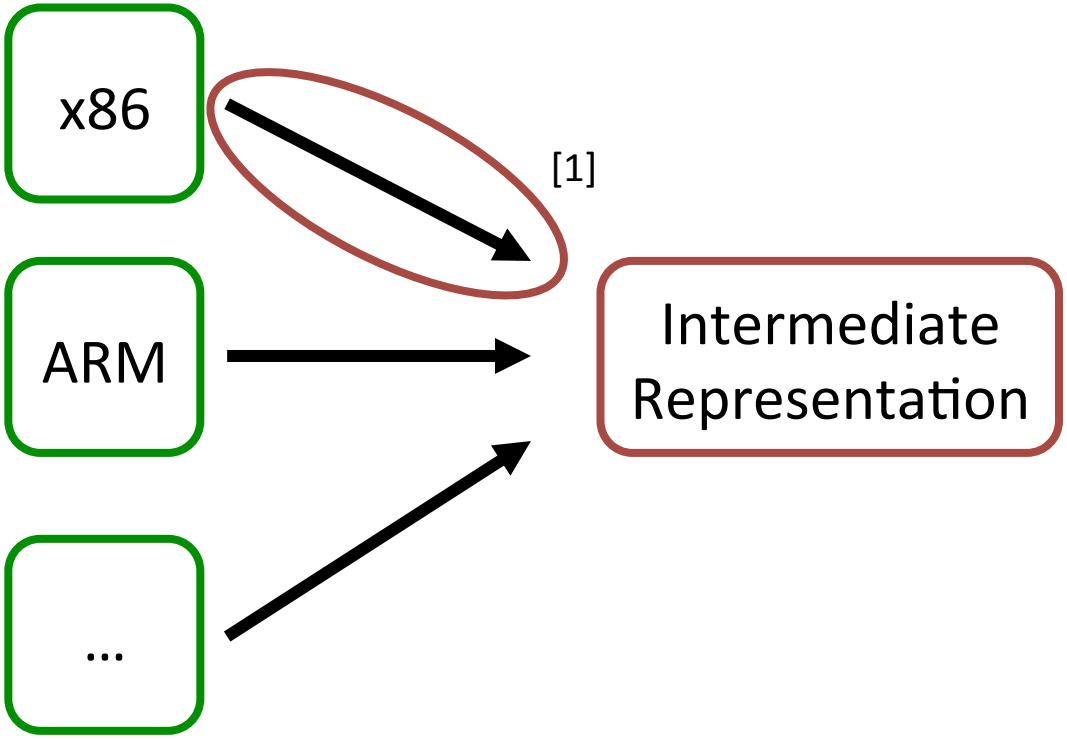
# The Problem: Translating Multiple Architectures



# Our Goal:
# Automate across Architectures

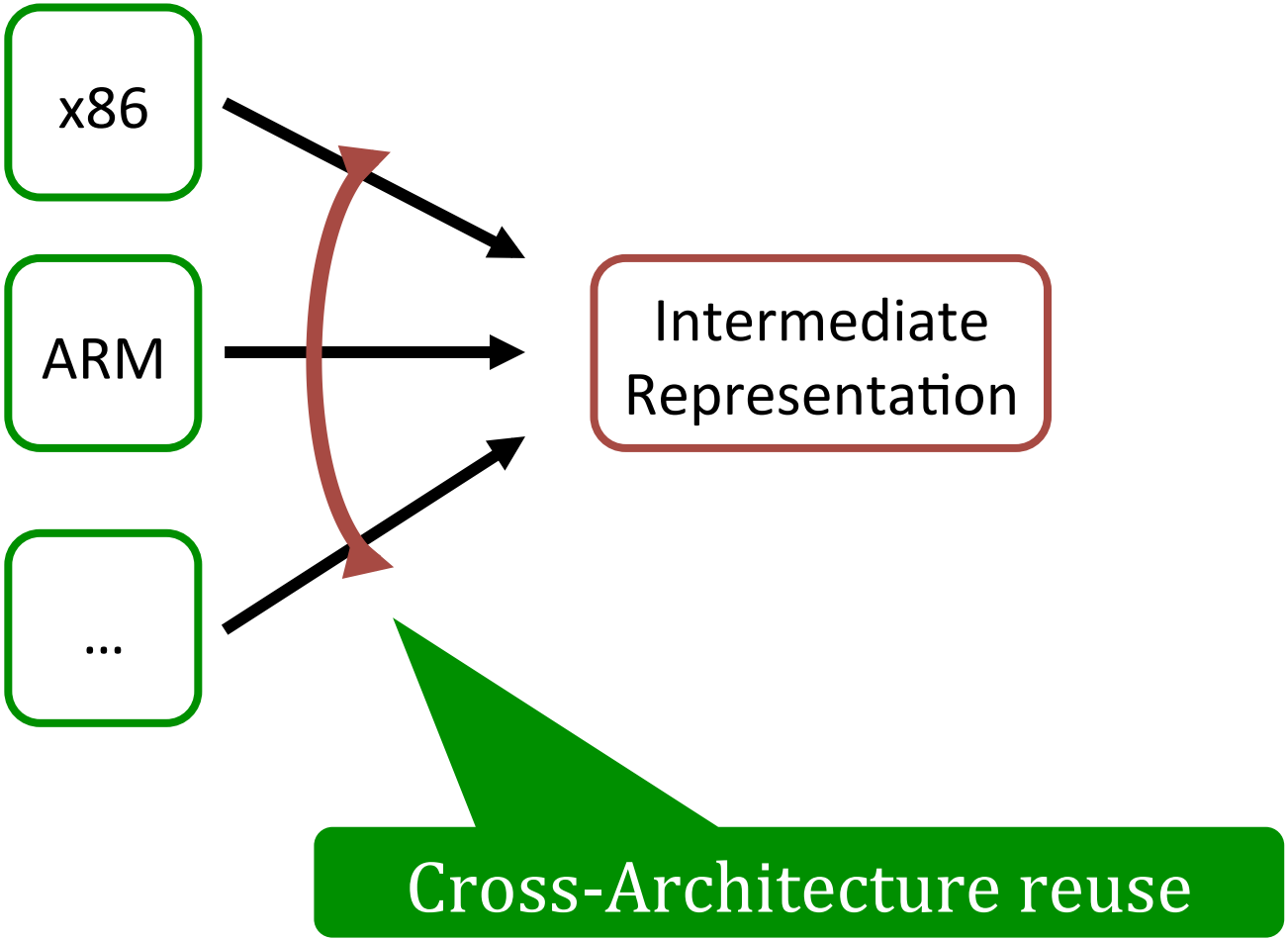


# Our Goal:
# Automate across Architectures

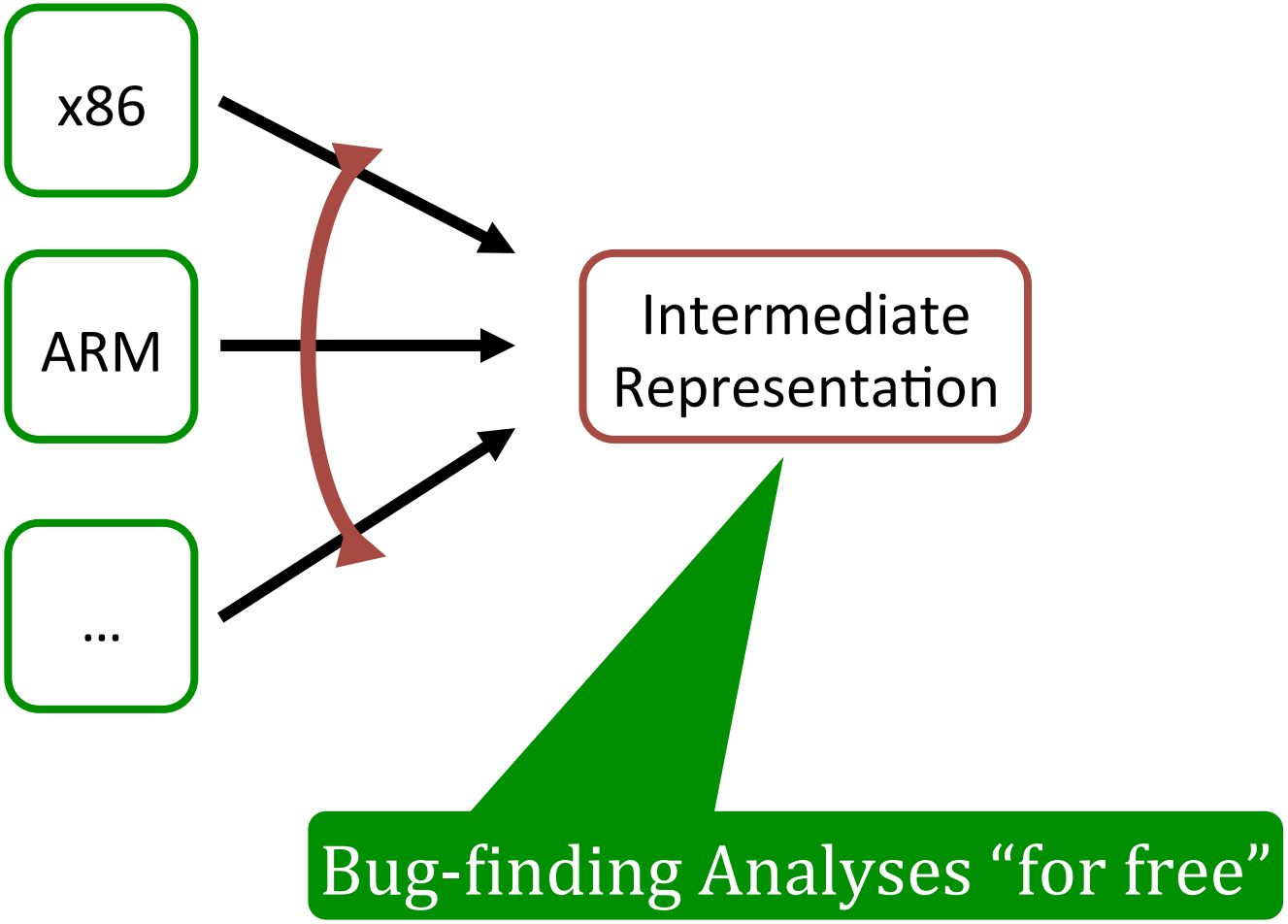



[1] Stratified synthesis: automatically learning the x86-64 instruction set. Heule et al.

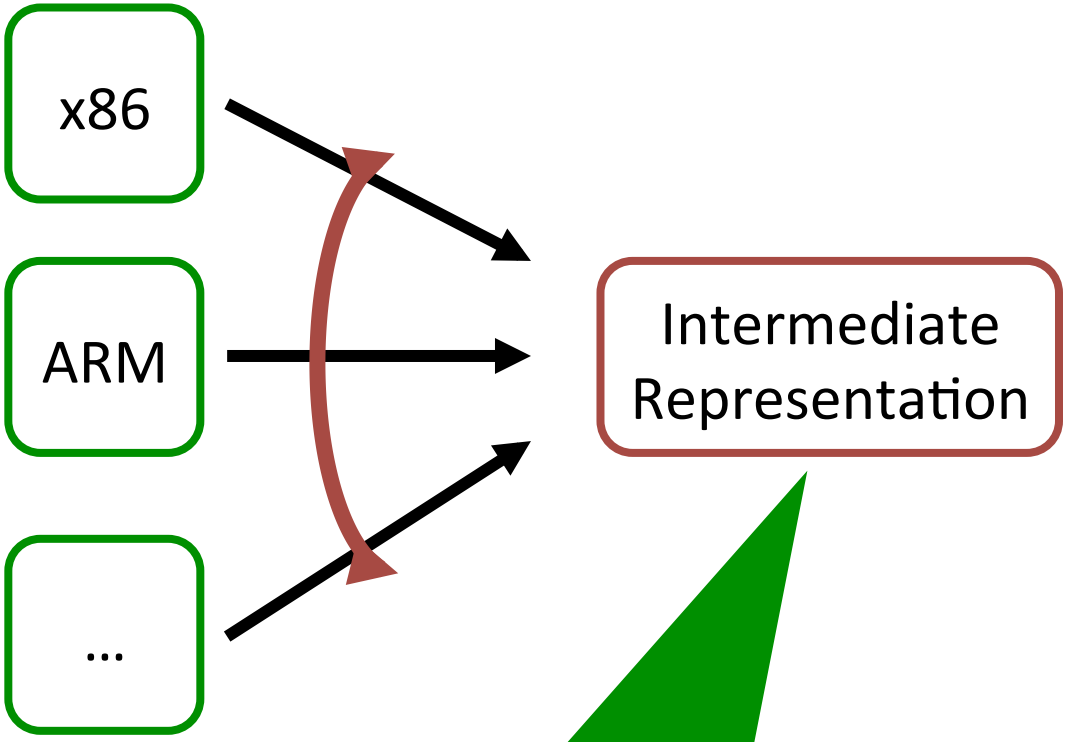
# Our Goal: Automate across Architectures



# Our Goal: Automate across Architectures



# Our Goal: Automate across Architectures




Dataflow framework

Bug-finding Analyses "for free"

VC Generation

Taint analysis

# What We Do

# What We Do

- IR translation as a Syntax-Guided Synthesis problem

# What We Do

- IR translation as a Syntax-Guided Synthesis problem

$$\forall x \,.\, \varphi(x, P(x))$$

# What We Do

- IR translation as a Syntax-Guided Synthesis problem

$$\forall x \,.\, \varphi(x, P(x)) \equiv \forall x \,.\, \text{oracle}(x) = P(x)$$

# What We Do

- IR translation as a Syntax-Guided Synthesis problem

$$\forall x \,.\, \varphi(x, P(x)) \equiv \forall x \,.\, \text{oracle}(x) = P(x)$$

Correctness specification

# What We Do

- IR translation as a Syntax-Guided Synthesis problem

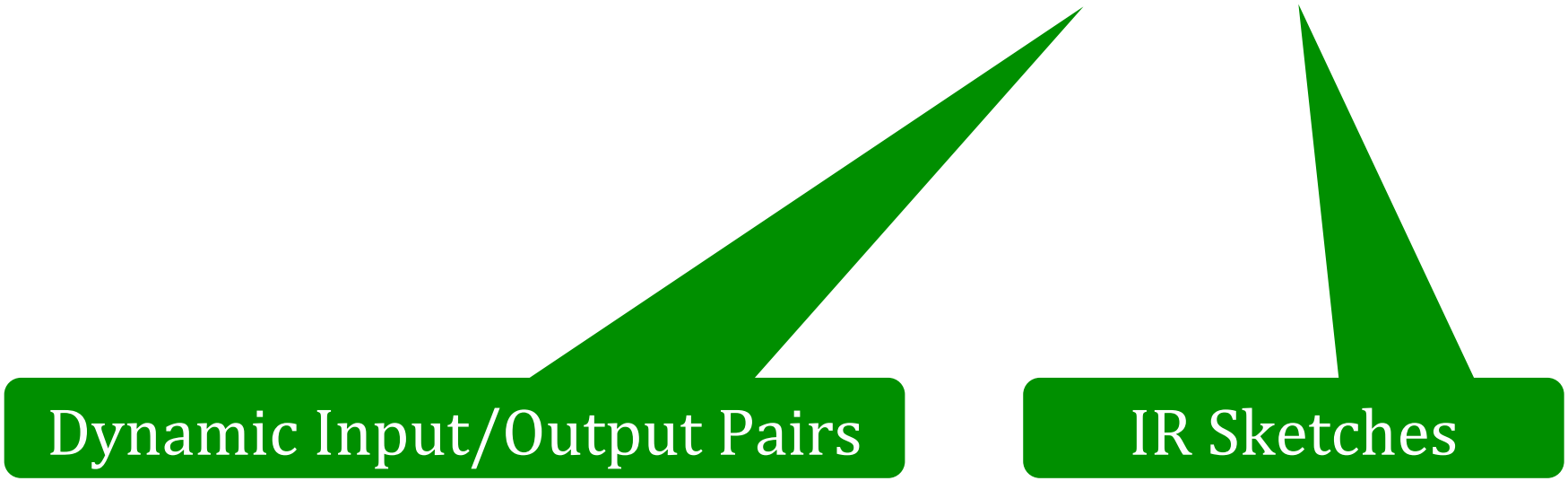
$$\forall x \,.\, \varphi(x, P(x)) \equiv \forall x \,.\, \text{oracle}(x) = P(x)$$

Dynamic Input/Output Pairs

# What We Do

- IR translation as a Syntax-Guided Synthesis problem

$$\forall x \,.\, \varphi(x, P(x)) \equiv \forall x \,.\, \text{oracle}(x) = P(x)$$



# What We Do

- IR translation as a Syntax-Guided Synthesis problem

$$\forall x \,.\, \varphi(x, P(x)) \equiv \forall x \,.\, \text{oracle}(x) = P(x)$$

Key Insight:
Learn IR Sketches from existing Lifter Productions

IR Sketches

# Learning Sketch Templates: Example

# Learning Sketch Templates: Example

**ARM**

**add R3, R0**

# Learning Sketch Templates: Example


**SOURCE**

**Native Instruction**

# Learning Sketch Templates: Example

**ARM**

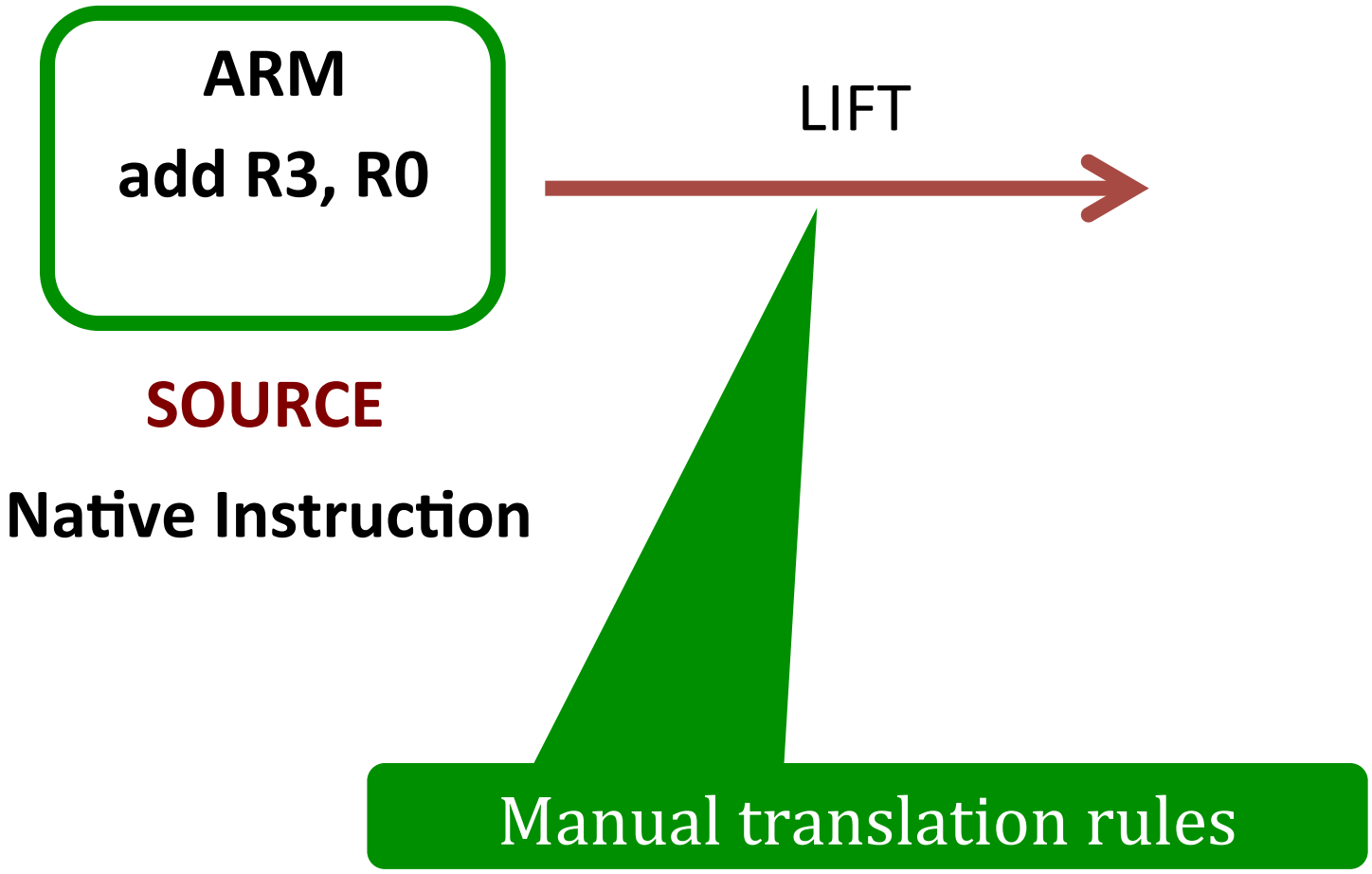
**add R3, R0**

**SOURCE**

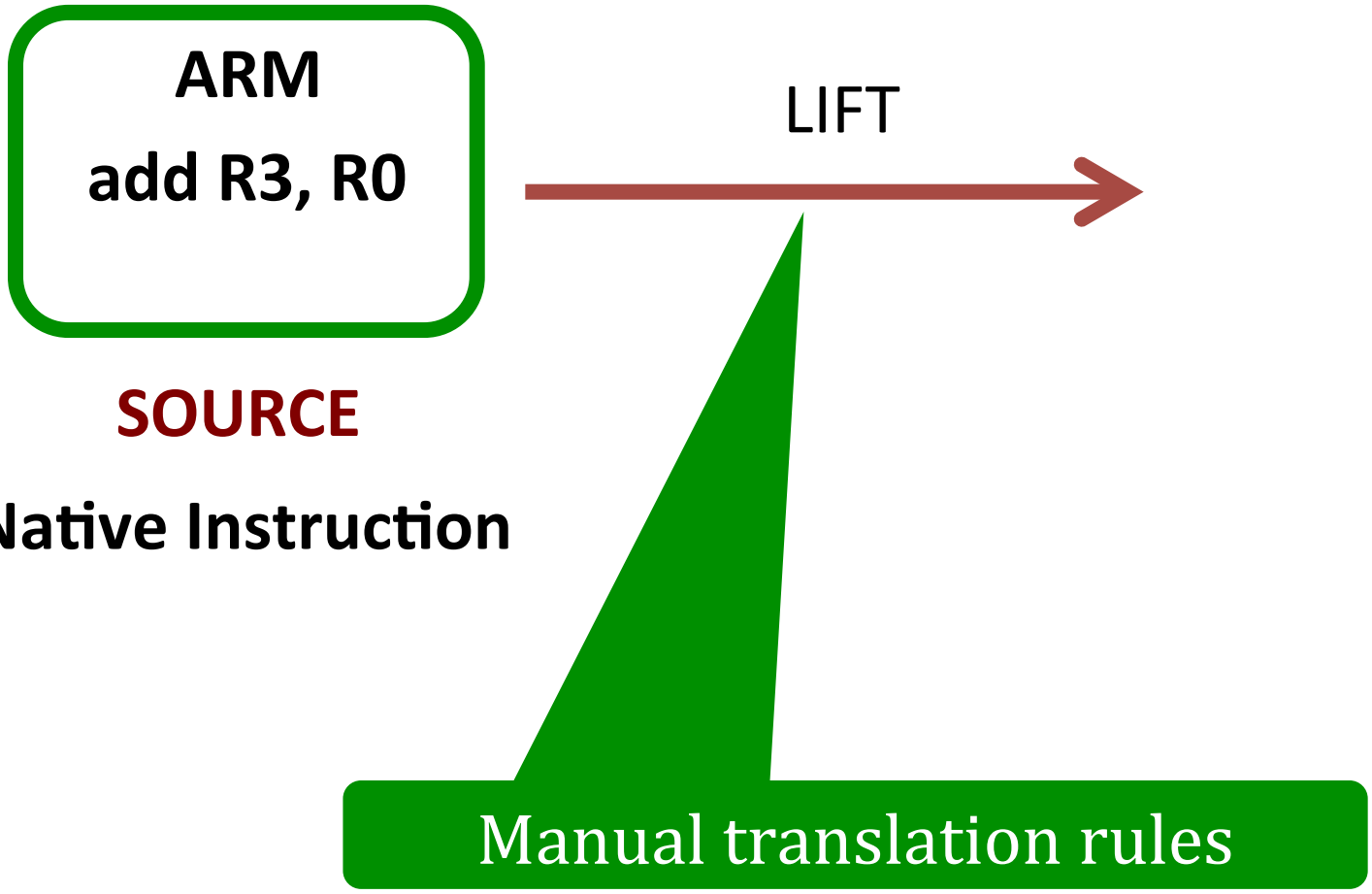
**Native Instruction**

Op codes and register names

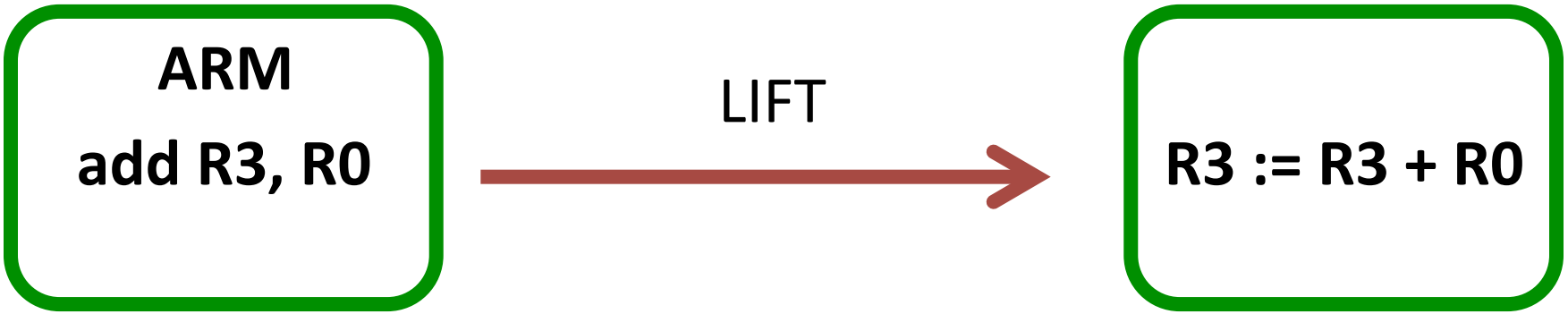
# Learning Sketch Templates: Example



# Learning Sketch Templates: Example



# Learning Sketch Templates: Example



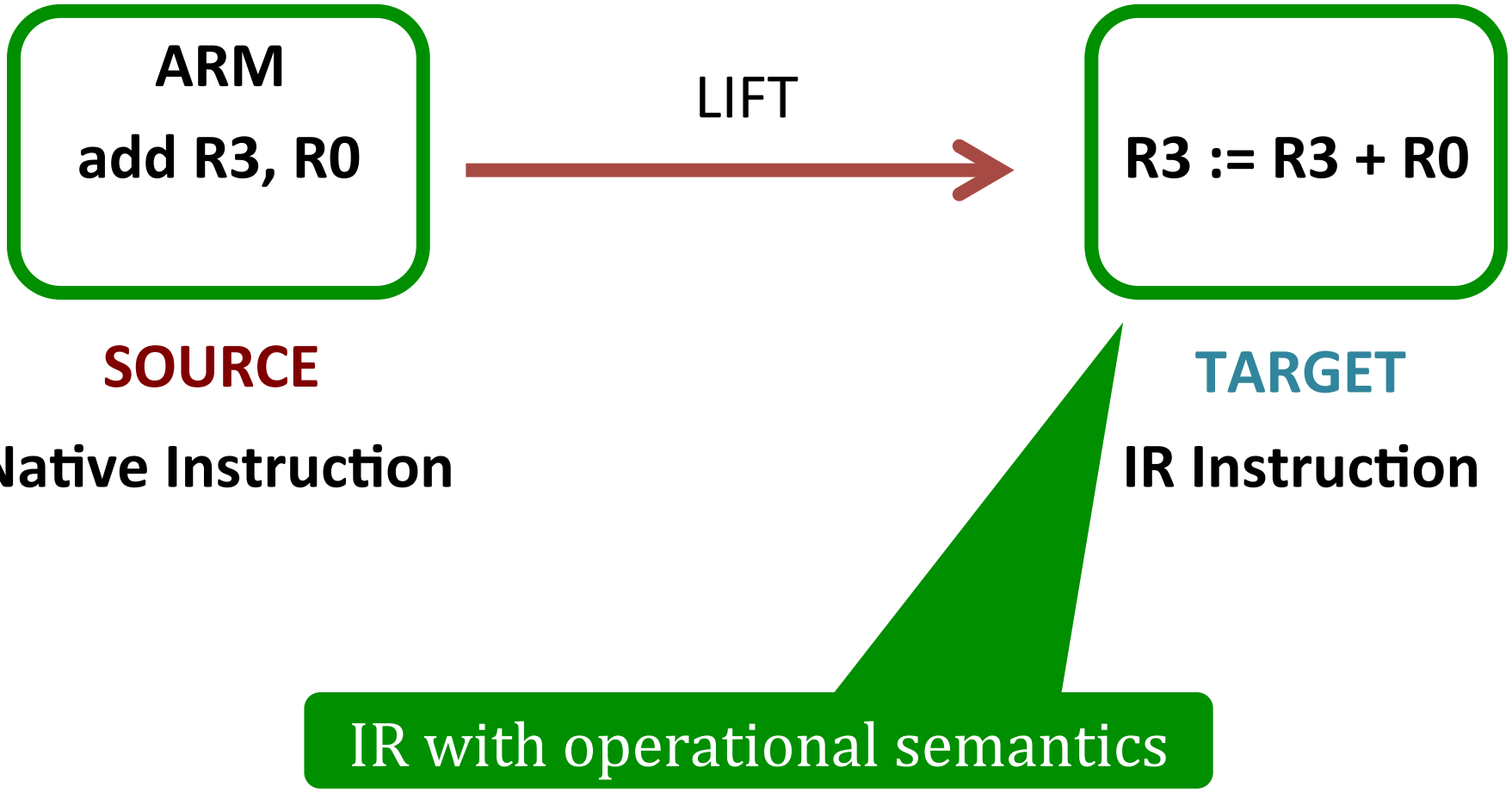

**SOURCE**

**Native Instruction**

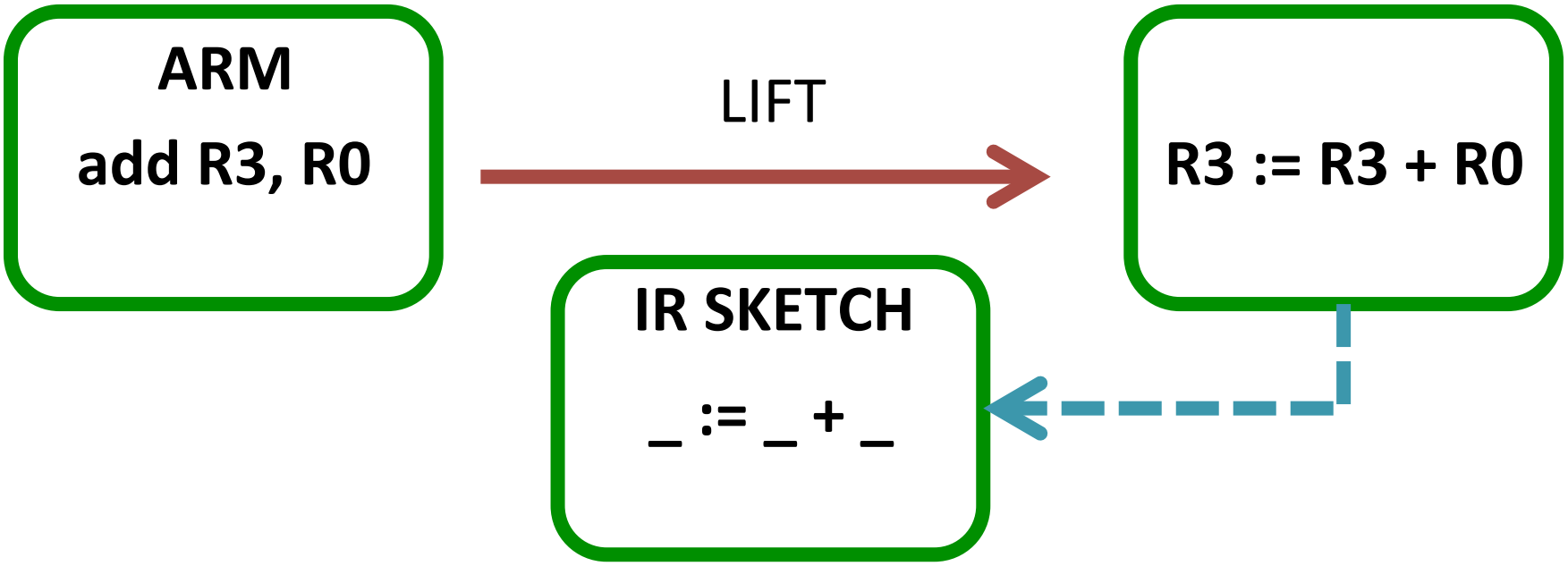
**TARGET**

**IR Instruction**

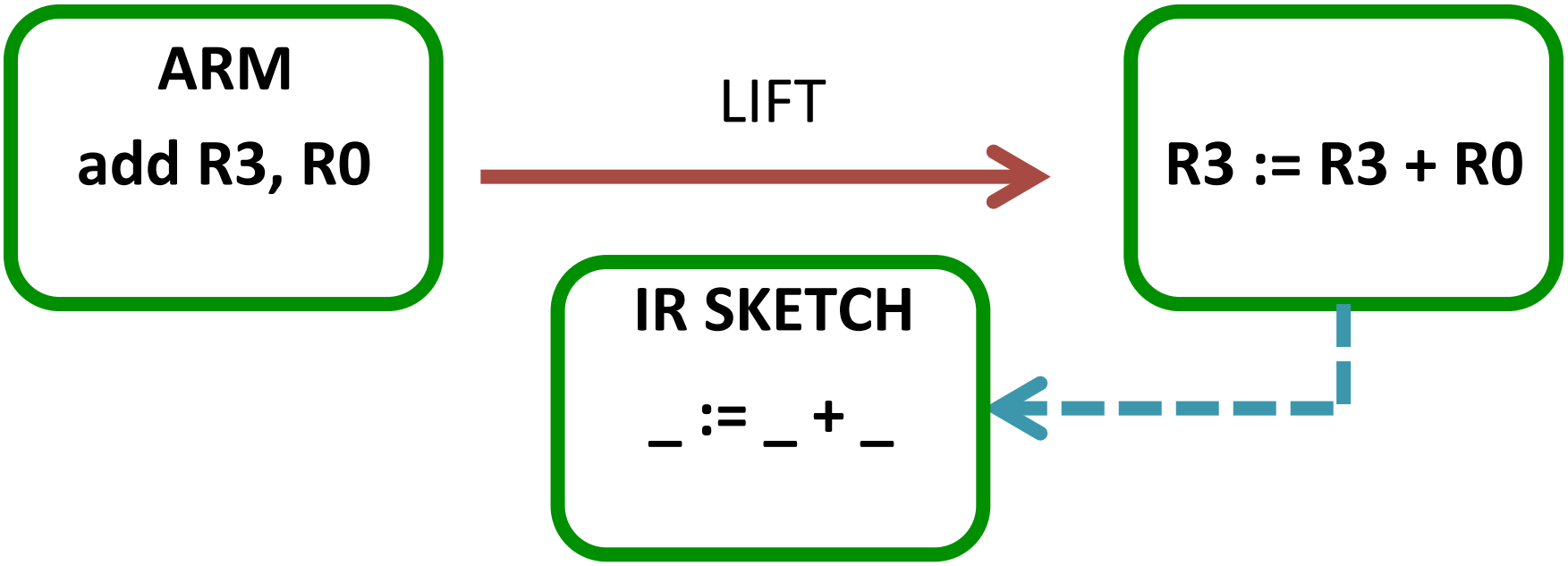
# Learning Sketch Templates: Example



# Learning Sketch Templates: Example



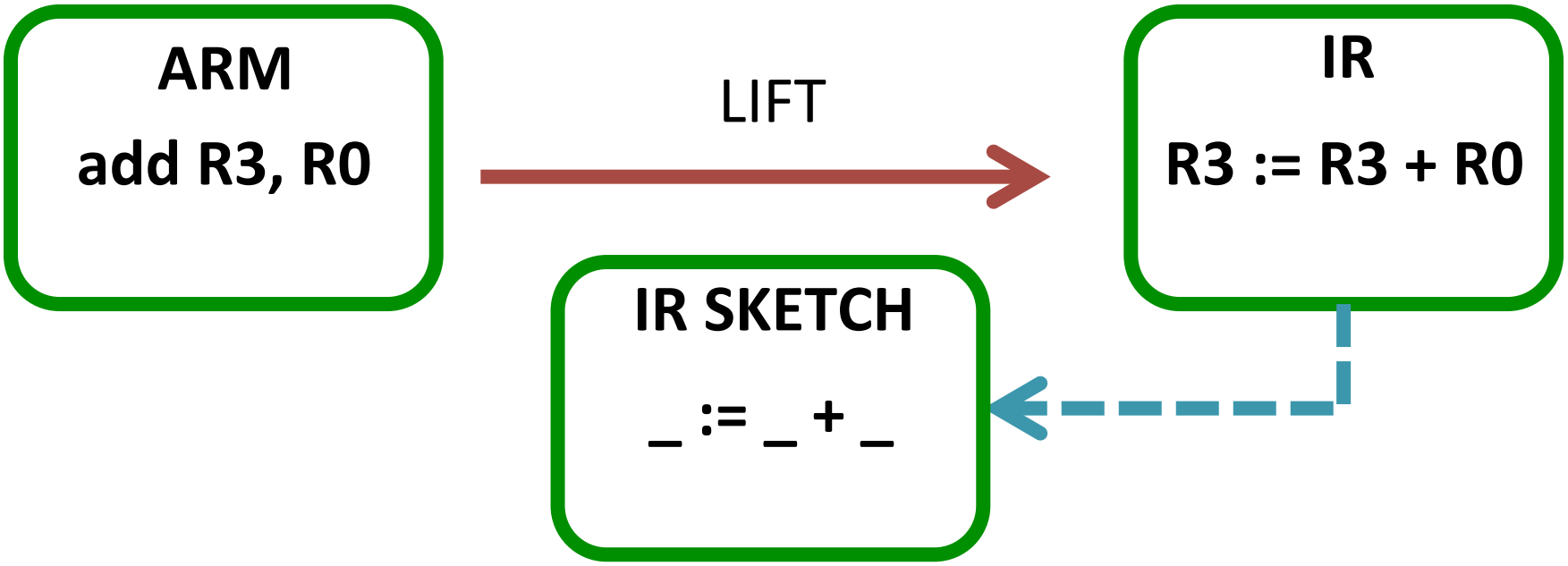
# Learning Sketch Templates: Example



# Learning Sketch Templates: Example

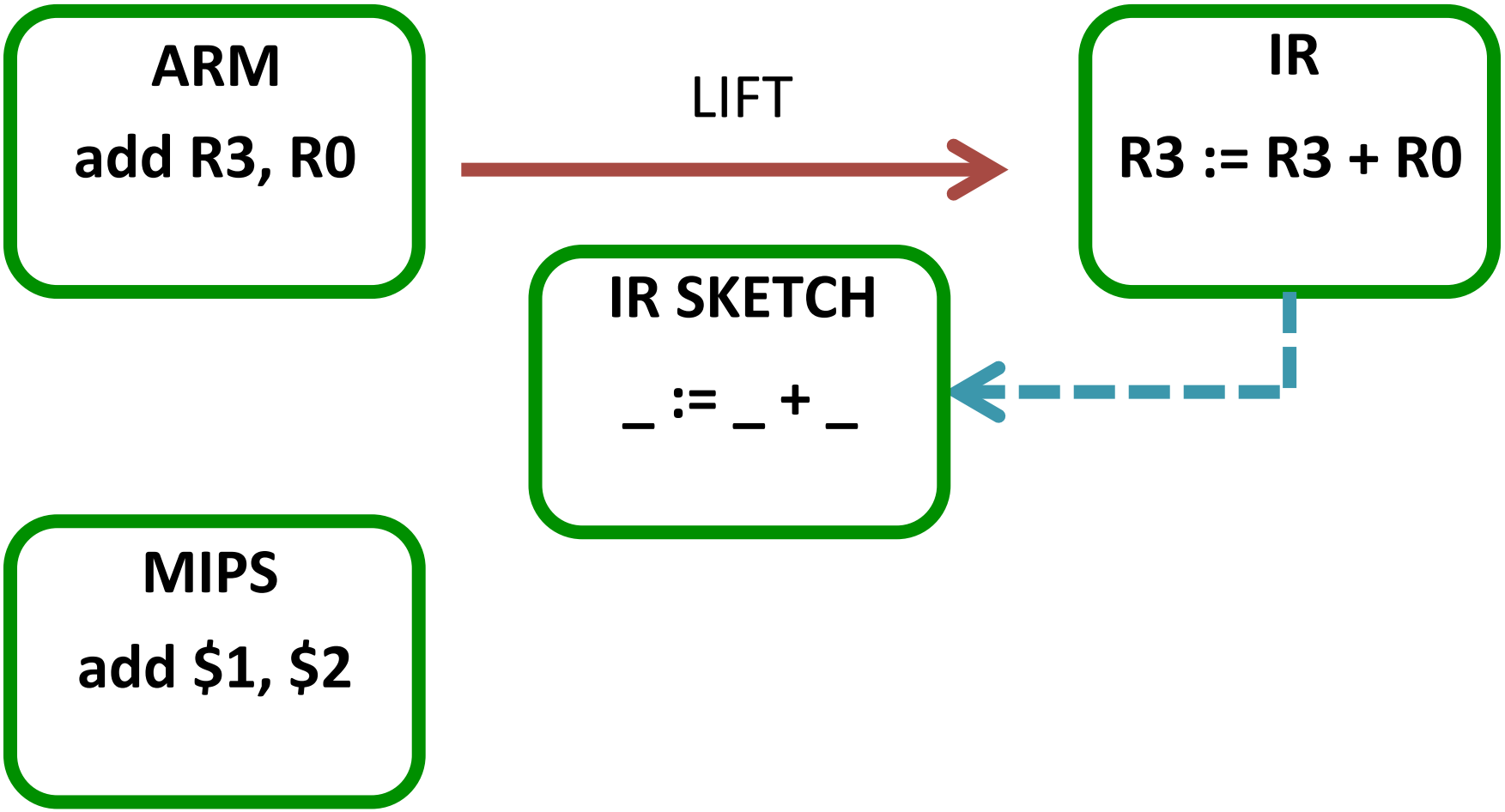
- Learn templates from lifter output for a **supported** architecture to synthesize one for an **unsupported** architecture

# Learning Sketch Templates: Example

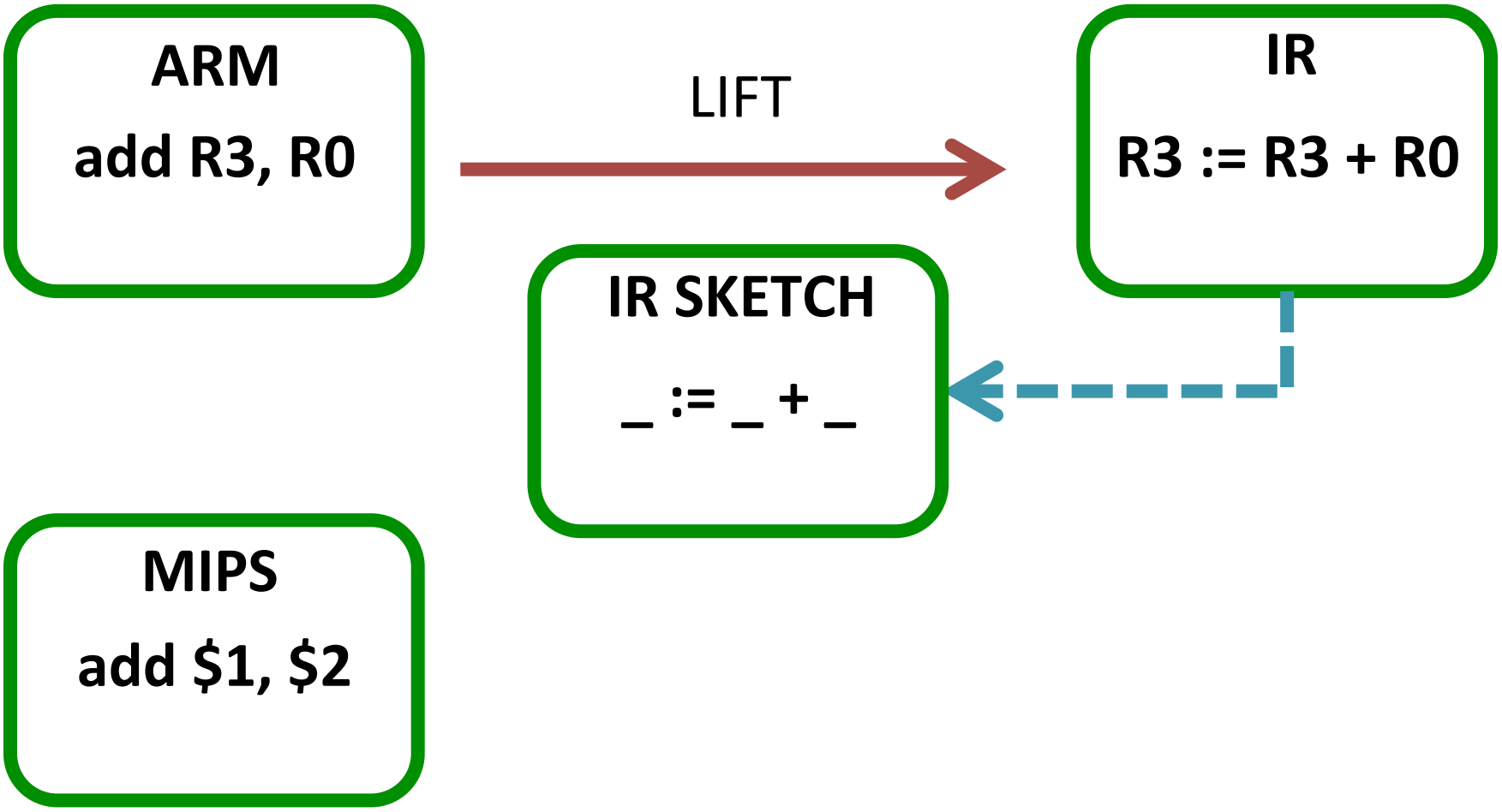



Create sketches from concrete expressions

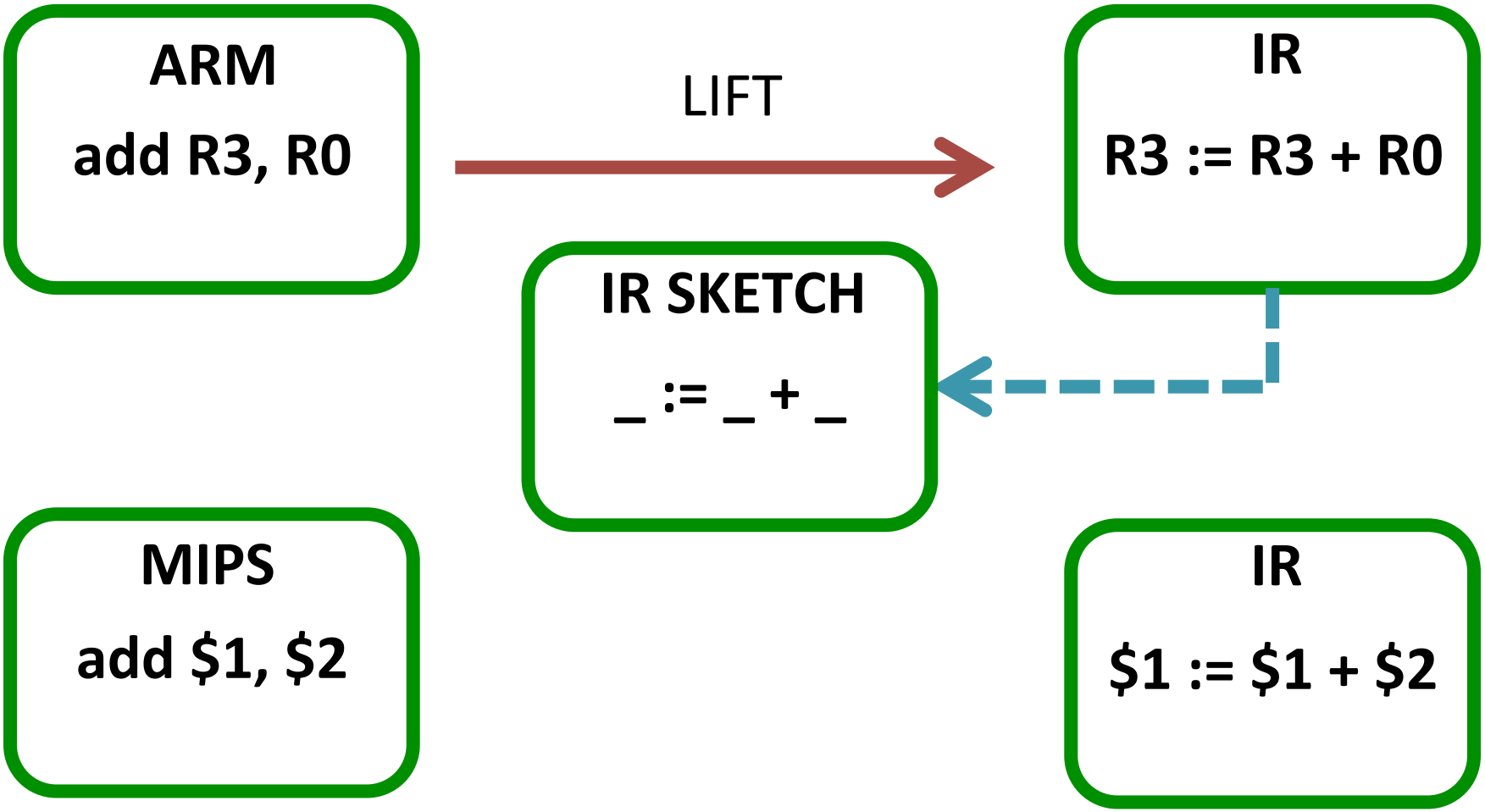
# Learning Sketch Templates: Example



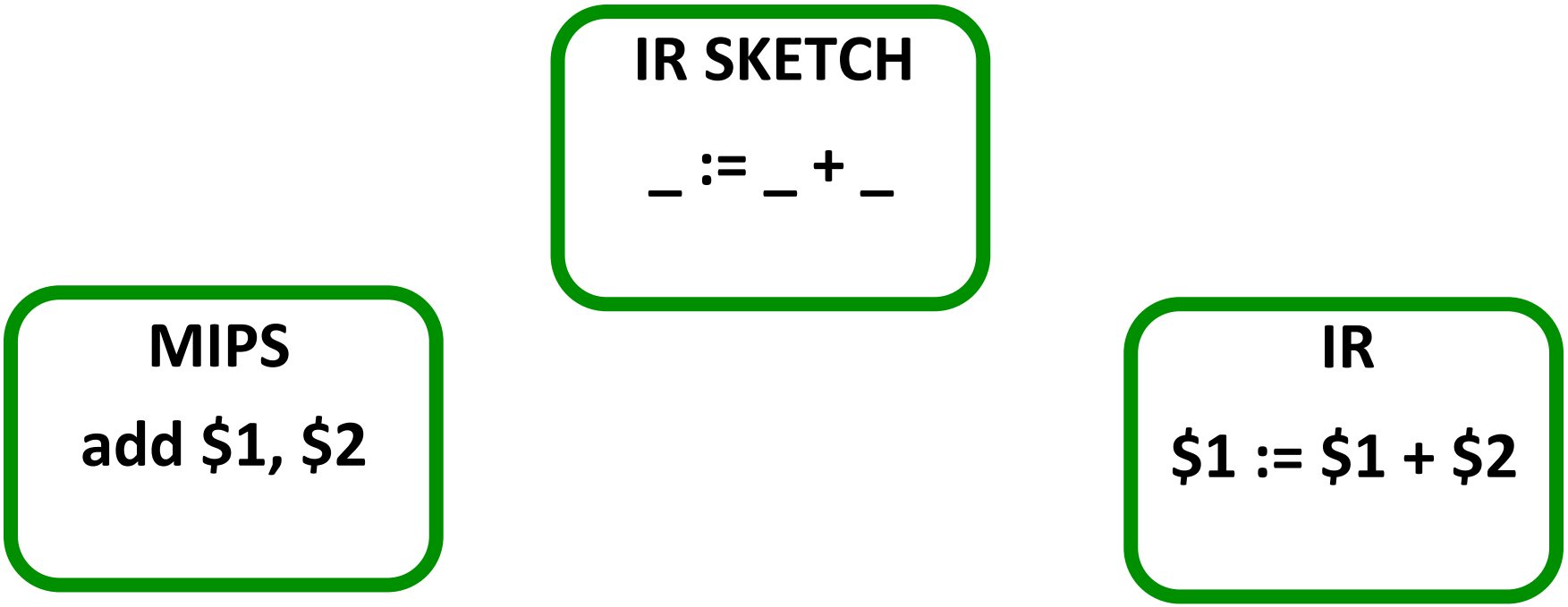
# Learning Sketch Templates: Example



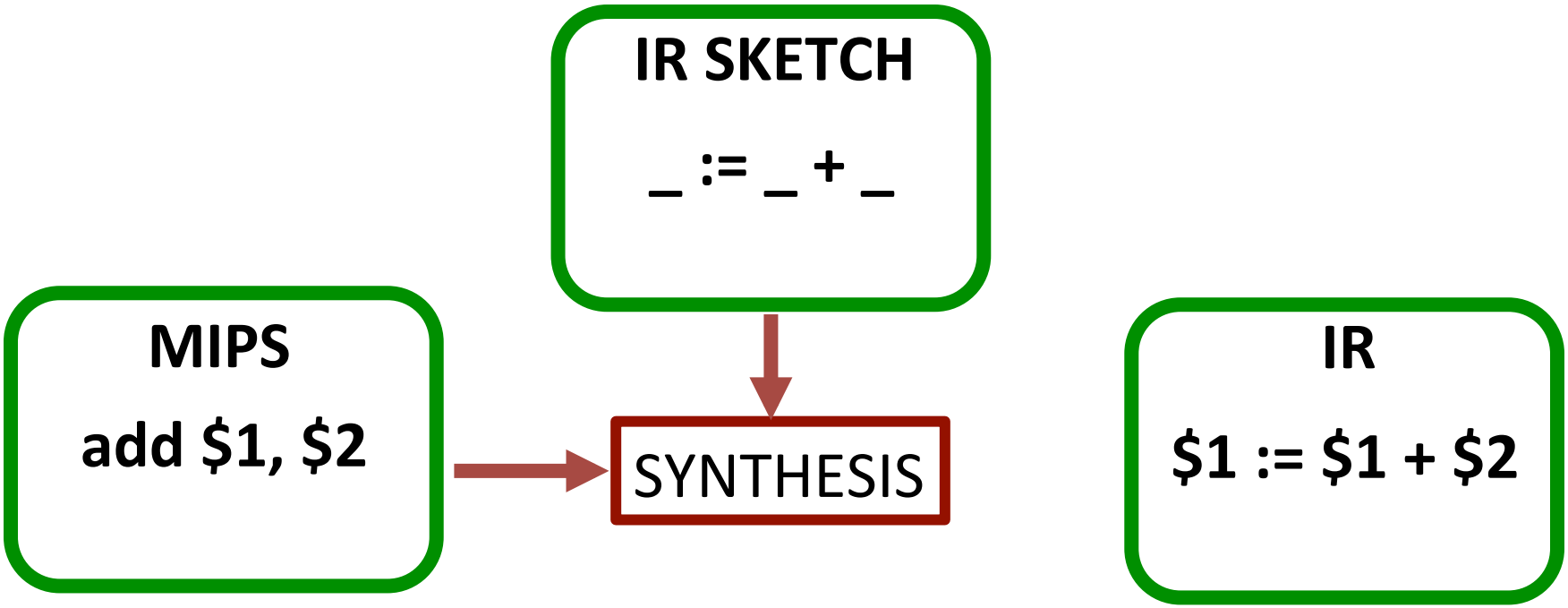
# Learning Sketch Templates: Example



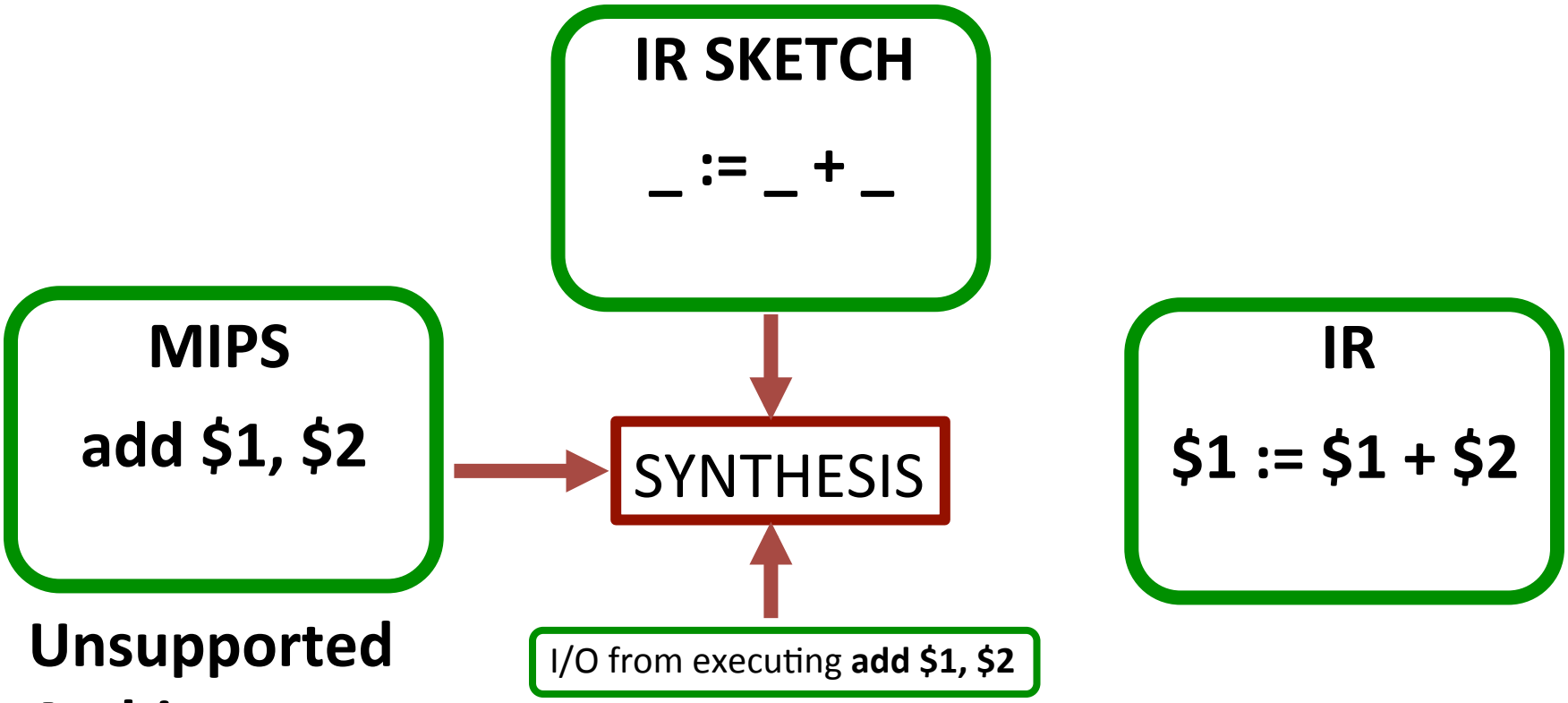
# Learning Sketch Templates: Example



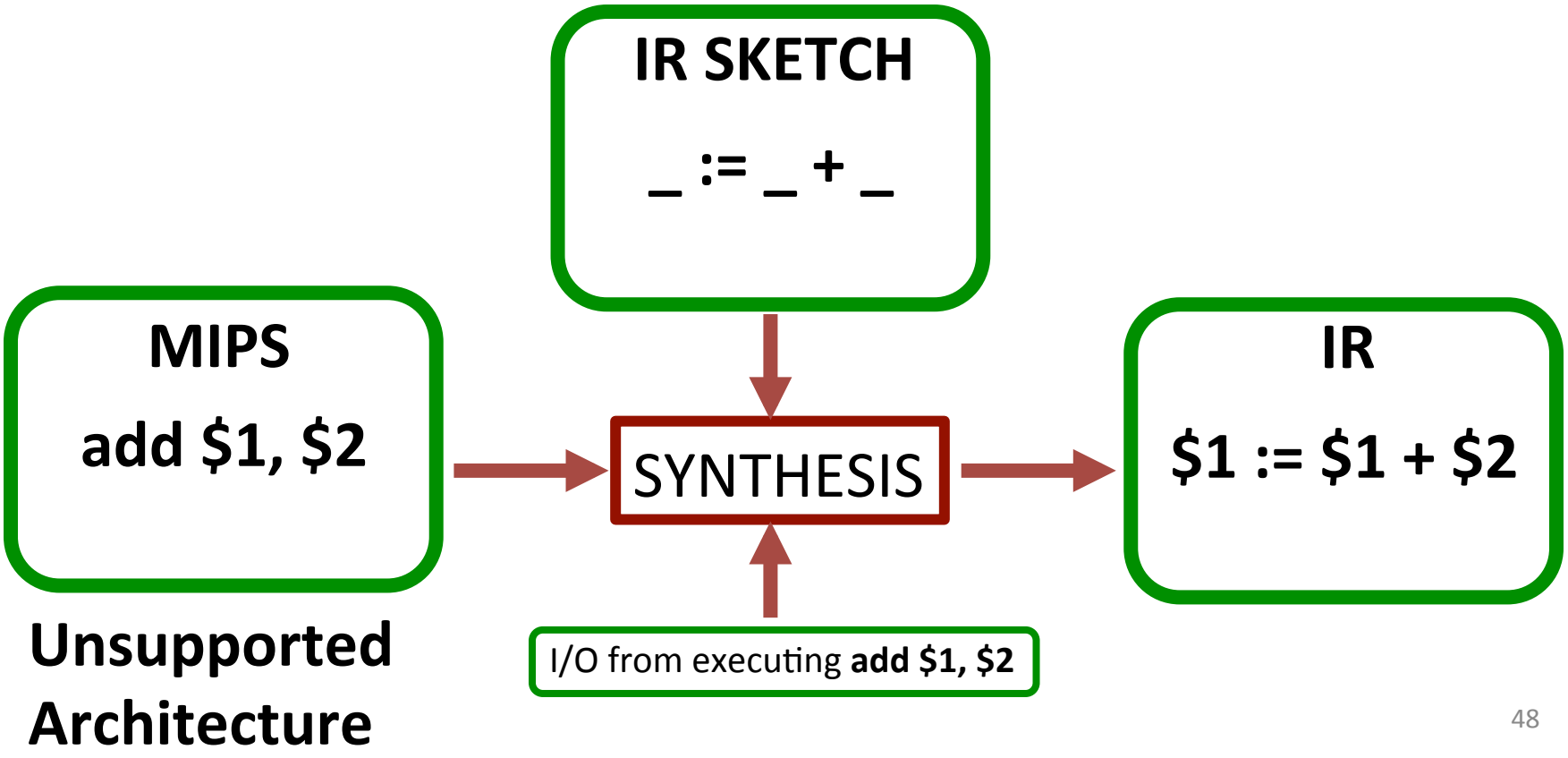
# Learning Sketch Templates: Example



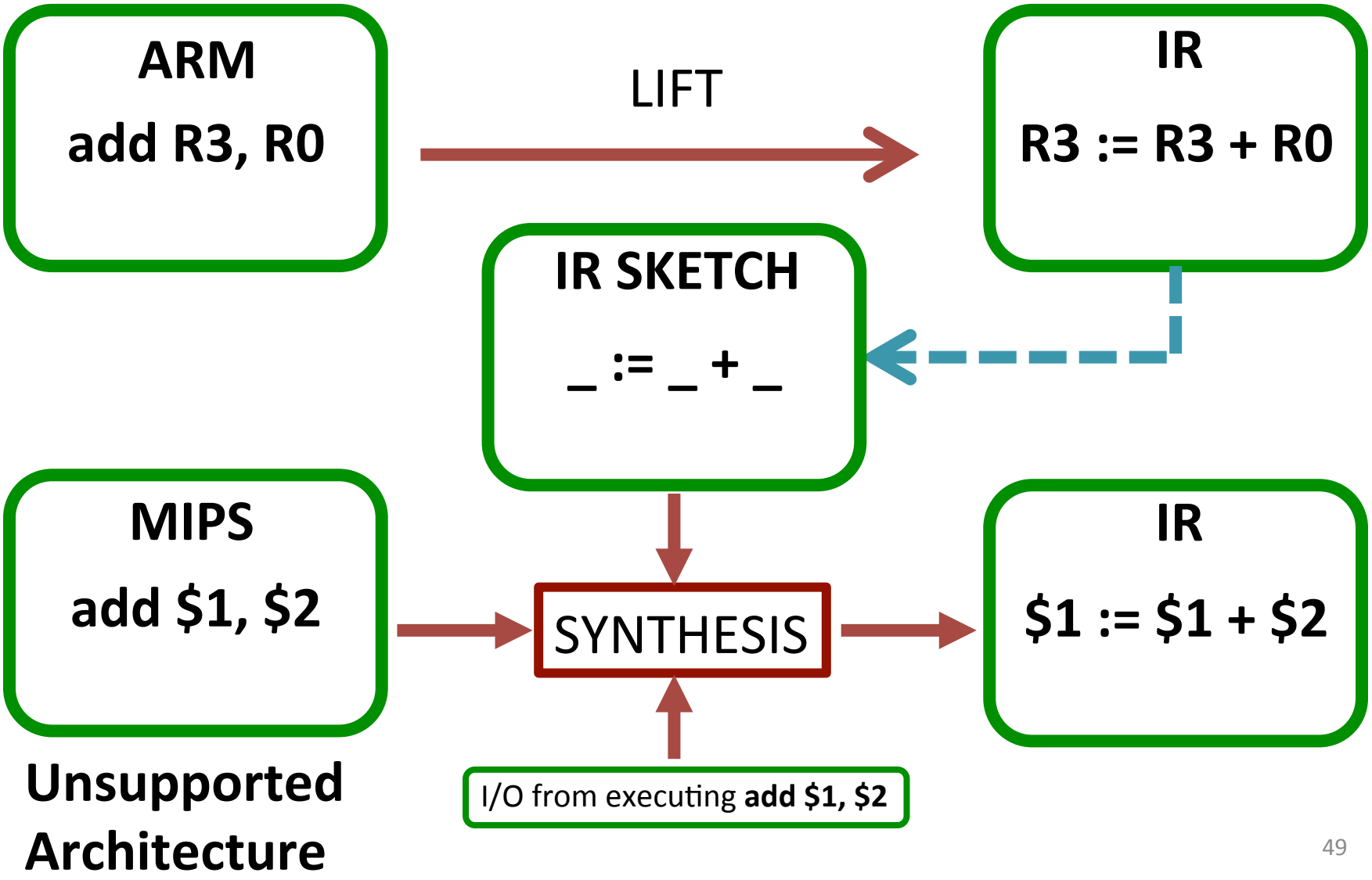
# Learning Sketch Templates: Example



# Learning Sketch Templates: Example



# Learning Sketch Templates: Example



# Learning Sketch Templates: Example

- Learn templates from lifter output for a **supported** architecture to synthesize one for an **unsupported** architecture

# Learning Sketch Templates: Example

- Learn templates from lifter output for a **supported** architecture to synthesize one for an **unsupported** architecture

- Code is "natural" ⟶ shared semantic properties

# Learning Sketch Templates: Example

- Learn templates from lifter output for a **supported** architecture to synthesize one for an **unsupported** architecture

- Code is "natural" ⟶ shared semantic properties

- Sketch templates exploit structural qualities to guide synthesis

# Synthesis Approach

# Synthesis Approach

## Native Behavior

- I/O pairs from native execution

# Synthesis Approach

## Native Behavior

- I/O pairs from native execution
- QEMU and PIN traces

# Synthesis Approach

## Native Behavior

- I/O pairs from native execution
- QEMU and PIN traces

## IR Target

- Binary Analysis Platform (BAP) IR

# Synthesis Approach

## Native Behavior

- I/O pairs from native execution
- QEMU and PIN traces

## IR Target

- Binary Analysis Platform (BAP) IR
- IR Interpreter

# Synthesis Approach

## Native Behavior

- I/O pairs from native execution
- QEMU and PIN traces

$$\langle \sigma_{\mathbb{T}}, \mathcal{I}_{\mathbb{T}}, \varnothing \rangle \overset{\mathbb{T}}{\rightsquigarrow} \langle \sigma'_{\mathbb{T}}, -, \mathcal{E}_{\mathbb{T}} \rangle$$

## IR Target

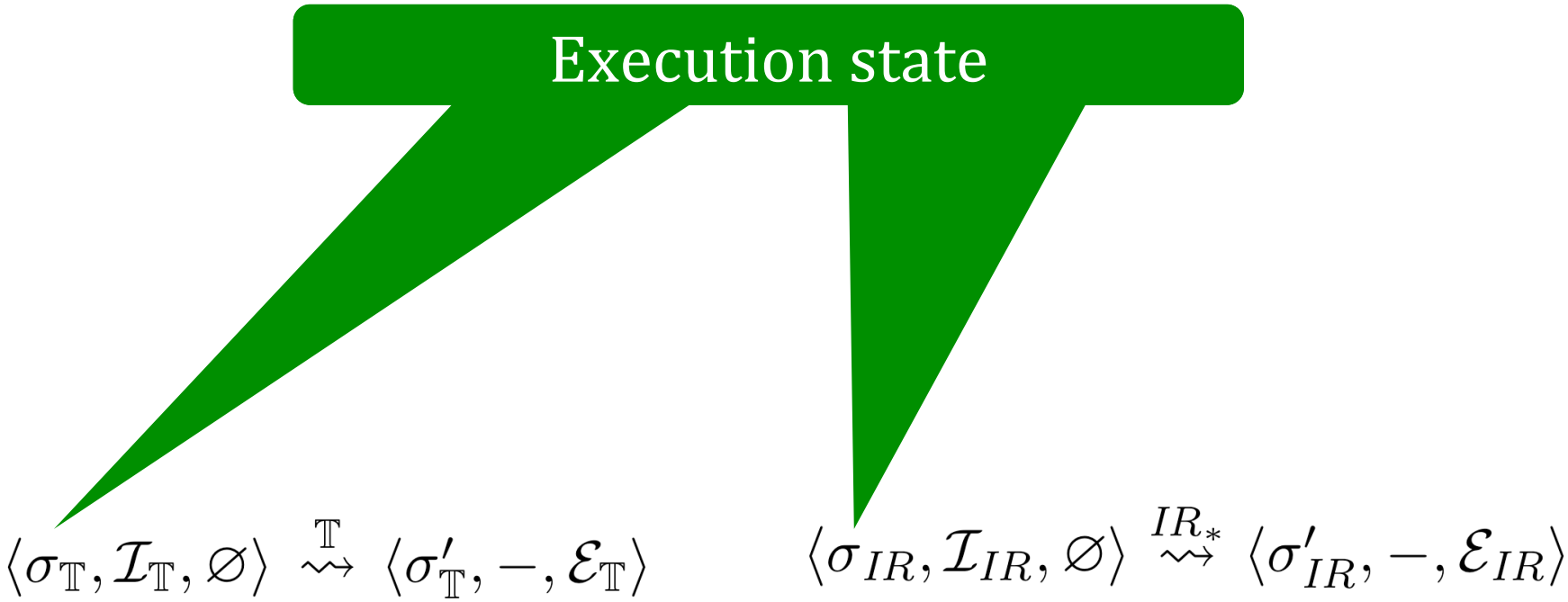
- Binary Analysis Platform (BAP) IR
- IR Interpreter

$$\langle \sigma_{IR}, \mathcal{I}_{IR}, \varnothing \rangle \overset{IR_*}{\rightsquigarrow} \langle \sigma'_{IR}, -, \mathcal{E}_{IR} \rangle$$

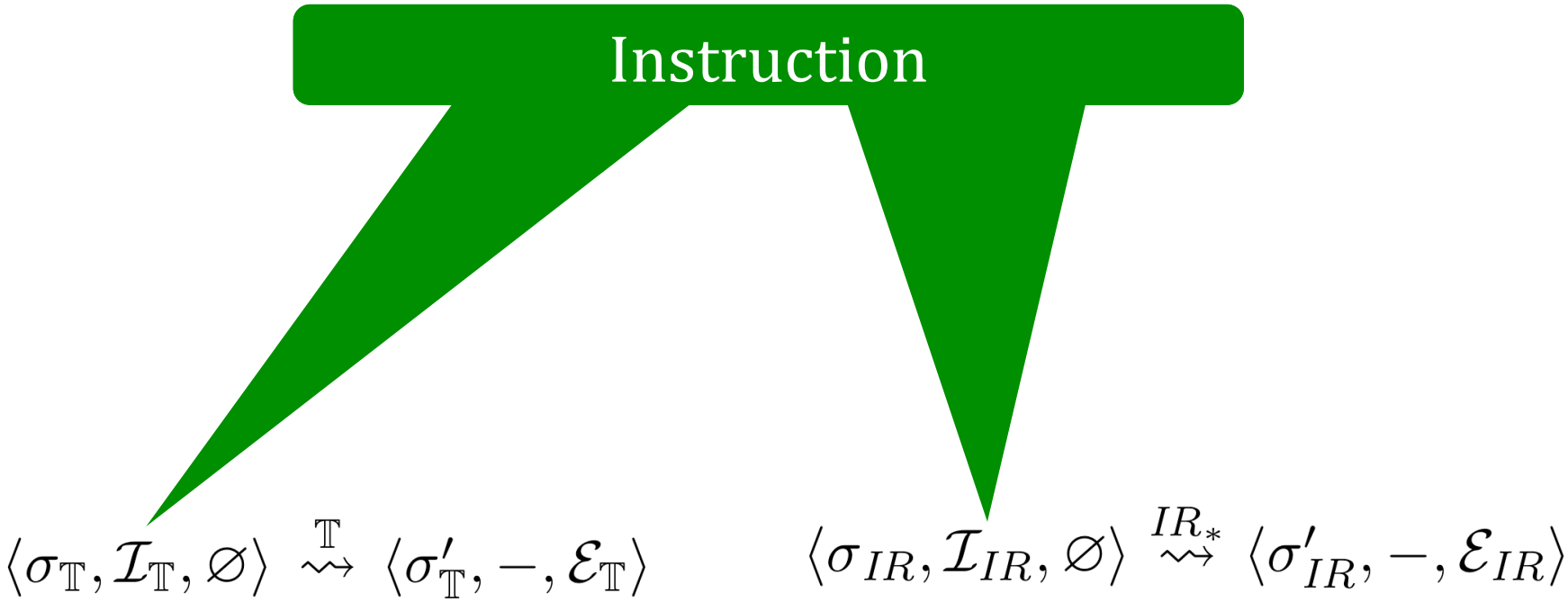
# Synthesis Approach

$$\langle \sigma_{\mathbb{T}}, \mathcal{I}_{\mathbb{T}}, \varnothing \rangle \overset{\mathbb{T}}{\rightsquigarrow} \langle \sigma'_{\mathbb{T}}, -, \mathcal{E}_{\mathbb{T}} \rangle \qquad \langle \sigma_{IR}, \mathcal{I}_{IR}, \varnothing \rangle \overset{IR_*}{\rightsquigarrow} \langle \sigma'_{IR}, -, \mathcal{E}_{IR} \rangle$$

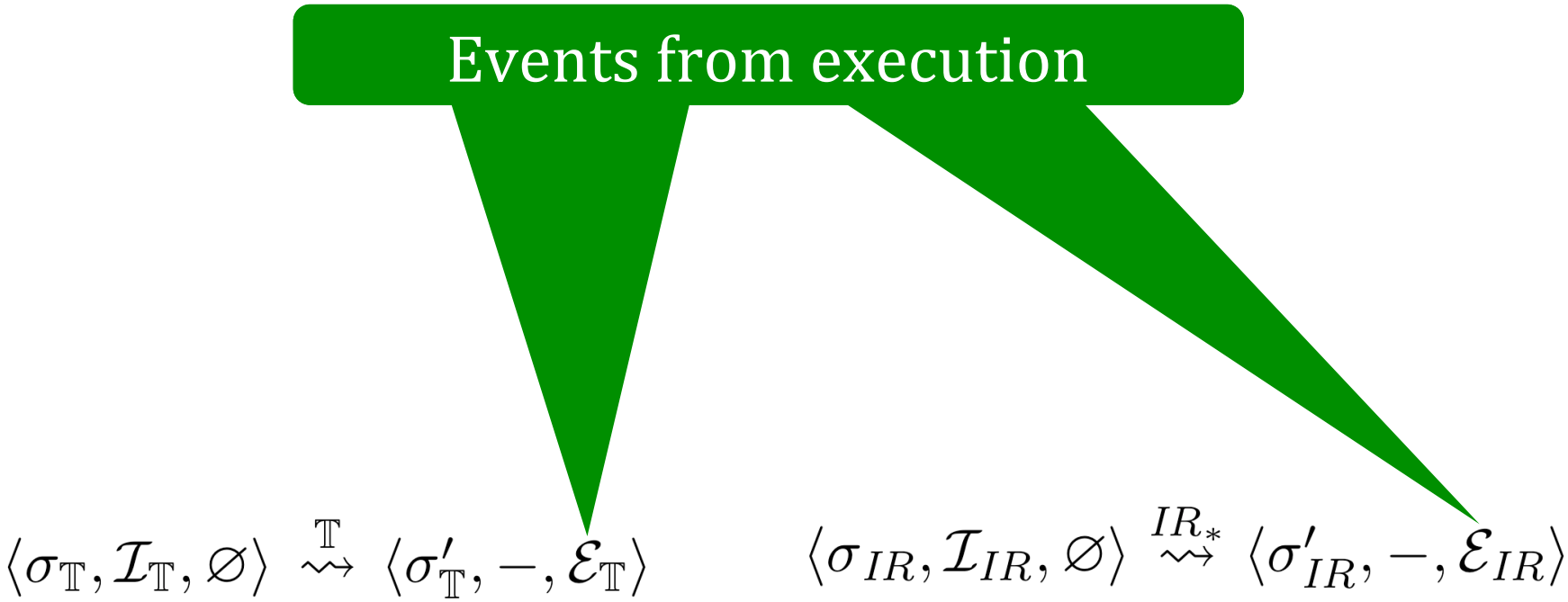
# Synthesis Approach



# Synthesis Approach



# Synthesis Approach



# Synthesis Approach

$$R2 := R0$$

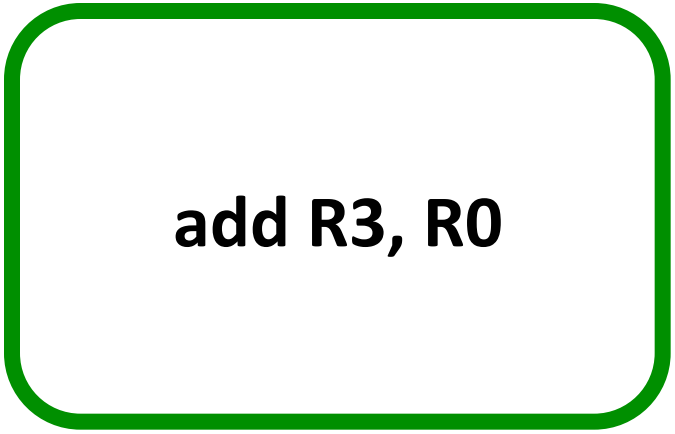
$$[(\texttt{R},\texttt{REG},\texttt{R0},\texttt{0x1}),(\texttt{W},\texttt{REG},\texttt{R2},\texttt{0x1})]$$


$$\langle \sigma_{\mathbb{T}}, \mathcal{I}_{\mathbb{T}}, \varnothing \rangle \overset{\mathbb{T}}{\rightsquigarrow} \langle \sigma'_{\mathbb{T}}, -, \mathcal{E}_{\mathbb{T}} \rangle \qquad \langle \sigma_{IR}, \mathcal{I}_{IR}, \varnothing \rangle \overset{IR_*}{\rightsquigarrow} \langle \sigma'_{IR}, -, \mathcal{E}_{IR} \rangle$$

# Inferring Semantics from I/O pairs

**SOURCE**

**Native Instruction**



# Inferring Semantics from I/O pairs

**SOURCE**

**Native Instruction**

**add R3, R0**

## Execution

**Input**

R3 = 2
R0 = 2

**Output**

R3 = 4
R0 = 2

# Inferring Semantics from I/O pairs

**SOURCE**

**Native Instruction**

**add R3, R0**

## Execution

**Input**

R3 = 2
R0 = 2

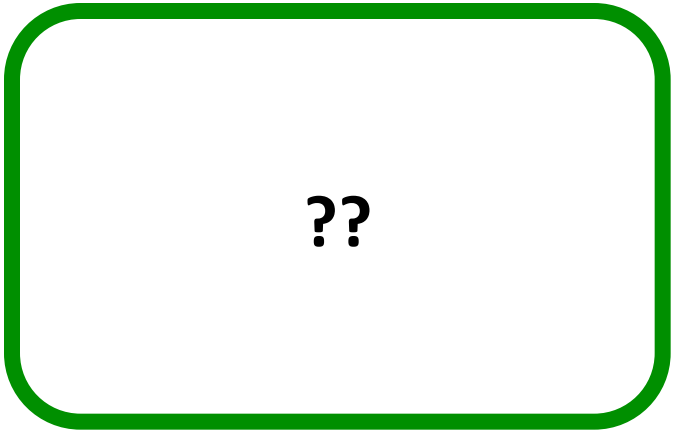
**Output**

**R3 = 4**
R0 = 2

# Synthesizing Translation

**TARGET**

**IR Instruction**




## Execution

**Input**

R3 = 2
R0 = 2

**Output**

R3 = 4
R0 = 2

# Synthesizing Translation

**TARGET**

**IR Instruction**

**R3, R0, R3**

## Execution

**Input**

R3 = 2
R0 = 2

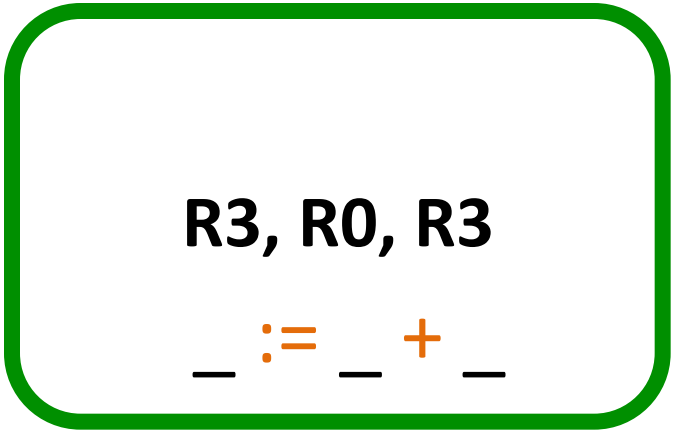
**Output**

R3 = 4
R0 = 2

# Syntax-guided Synthesis

**TARGET**

**IR Instruction**



# Syntax-guided Synthesis

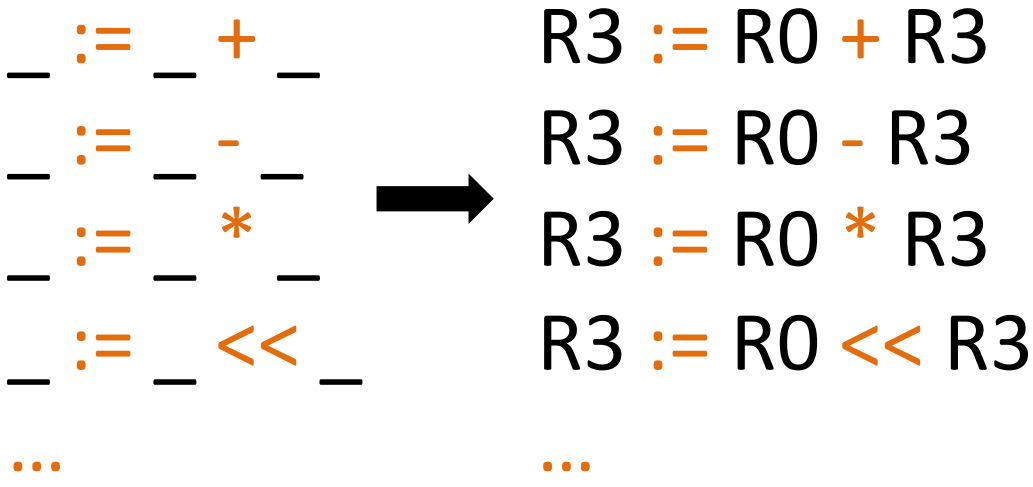
**TARGET**

**IR Instruction**

**R3, R0, R3**

_ := _ + _

**Sketch Templates**



# Syntax-guided Synthesis

**TARGET**

**IR Instruction**

**R3, R0, R3**

_ := _ + _

## Sketch Templates

_ := _ + _
_ := _ - _
_ := _ * _
_ := _ << _
…

R3 := R0 + R3
R3 := R0 - R3
R3 := R0 * R3
R3 := R0 << R3
…

Exhaustive Enumeration

# Syntax-guided Synthesis

**TARGET**

**IR Instruction**

**R3, R0, R3**

_ := _ + _

## Sketch Templates

_ := _ + _
_ := _ - _
_ := _ * _
_ := _ << _
…

R3 := R0 + R3
R3 := R0 - R3
R3 := R0 * R3
R3 := R0 << R3
…

Exhaustive Enumeration

Permute adjacent operands

# Syntax-guided Synthesis

**TARGET**

**IR Instruction**

**R3 := R0 + R3**

## IR Interpreter

**Input**

R3 = 2
R0 = 2

**Output**

R3 = 4
R0 = 2

# Syntax-guided Synthesis

## Native Execution

**Input**

R3 = 2
R0 = 2

**Expected Output**

R3 = 4
R0 = 2

## IR Interpreter

**Input**

R3 = 2
R0 = 2

**Output**

R3 = 4
R0 = 2

# Validating Correctness

**TARGET**

**IR Instruction**

**R3 := R0 * R3**

## IR Interpreter

**Input**

R3 = 2
R0 = 2

**Expected Output**

R3 = 4
R0 = 2

# Validating Correctness

**TARGET**

**IR Instruction**

**R3 := R0 * R3**

## IR Interpreter

**Input**

R3 = 2
R0 = 2

**Expected Output**

R3 = 4
R0 = 2

**Oops…**

# Validating Correctness

**TARGET**

**IR Instruction**

**R3 := R0 * R3**

**Oops…**

# Validating Correctness

**TARGET**

**IR Instruction**

**R3 := R0 * R3**

## Native Execution

**Input**

**R3 = 3**
R0 = 2

**Expected Output**

**R3 = 5**
R0 = 2

# Validating Correctness

**TARGET**

**IR Instruction**

**R3 := R0 * R3**

R3 = 6

## Native Execution

**Input**

**R3 = 3**
R0 = 2

**Expected Output**

**R3 = 5**
R0 = 2

# Validating Correctness

**TARGET**

**IR Instruction**

**R3 := R0 * R3**

R3 = 6 → Invalidate sketch

## Native Execution

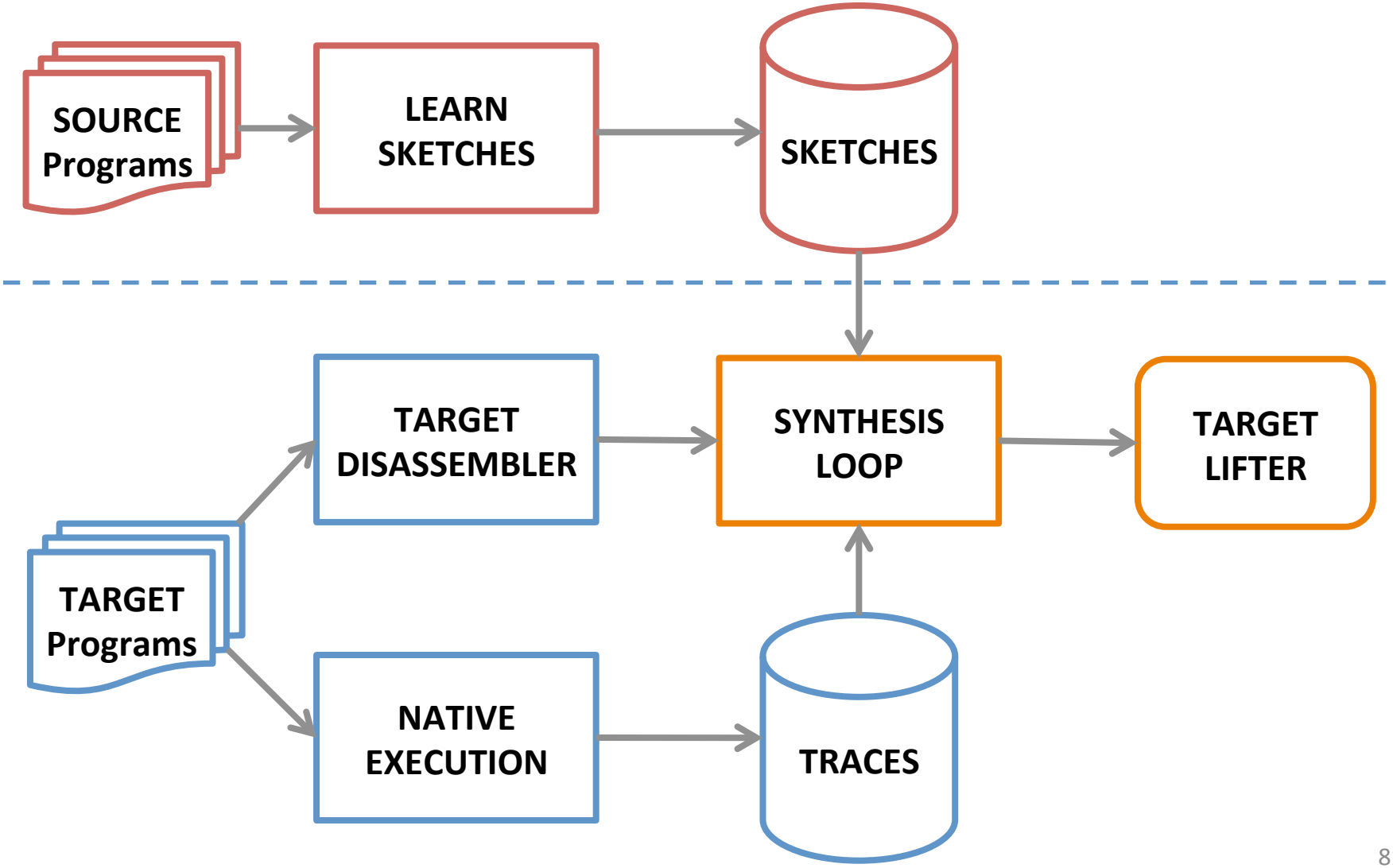
**Input**

**R3 = 3**
R0 = 2

**Expected Output**

**R3 = 5**
R0 = 2

# Lifter Synthesis System



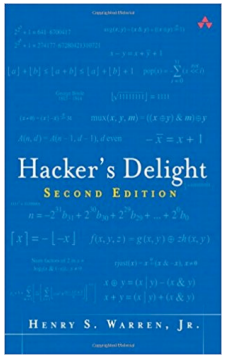
# Experimental Setup

# Experimental Setup

- Synthesize unsupported lifter target (MIPS)

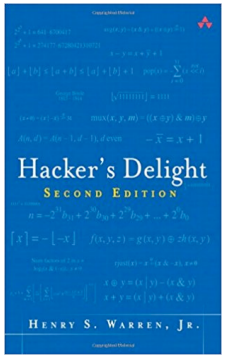
# Experimental Setup

- Synthesize unsupported lifter target (MIPS)
- Collect dynamic traces from arithmetic-heavy benchmark
  - 5 Hacker's delight programs



# Experimental Setup

- Synthesize unsupported lifter target (MIPS)
- Collect dynamic traces from arithmetic-heavy benchmark
  - 5 Hacker's delight programs




- Mine IR sketches from Coreutils, lifted from ARM and x86

# Results: Analysis Reuse

# Results: Analysis Reuse

- Taint analysis for warn-unused-result bugs

# Results: Analysis Reuse

- Taint analysis for warn-unused-result bugs
  - Forget to check memory is allocated
  - Forget to check privileges are dropped

# Results: Analysis Reuse

- Taint analysis for warn-unused-result bugs
  - Forget to check memory is allocated
  - Forget to check privileges are dropped

- Ran on 30 binaries of COTS D-Link router

# Results: Analysis Reuse

- Taint analysis for warn-unused-result bugs
  - Forget to check memory is allocated
  - Forget to check privileges are dropped
- Ran on 30 binaries of COTS D-Link router
- 29 bugs, 2 false positives

# Results: Analysis Reuse

| Name | # | Functions |
|---|---|---|
| **pppd** | 1 | fwrite |
| **iptables** | 3 | fwrite |
| **rdnssd** | 1 | setsockopt |
| **ntpclient** | 1 | send |
| **speedtest** | 2 | system, fgets |
| **timer** | 2 | read, shutdown |
| **wakeOnLanProxy** | 1 | shutdown |
| **wcnd** | 8 | system |

# Results: Analysis Reuse

| Name | # | Functions |
|---|---|---|
| **pppd** | 1 | fwrite |
| **iptables** | 3 | fwrite |
| **rdnssd** | 1 | setsockopt |
| **ntpclient** | 1 | send |
| **speedtest** | 2 | system, fgets |
| **timer** | 2 | read, shutdown |
| **wakeOnLanProxy** | 1 | shutdown |
| **wcnd** | 8 | system |

```
uint32_t data[12];
struct timeval now;

memset(data, 0, sizeof(data));
data[0] = htonl (
        ( LI << 30 ) | ( VN << 27 ) | (
        ( STRATUM << 16) | ( POLL << 8
data[1] = htonl(1<<16);  /* Root Delay
data[2] = htonl(1<<16);  /* Root Disper
gettimeofday(&now,NULL);
data[10] = htonl(now.tv_sec + JAN_1970)
data[11] = htonl(NTPFRAC(now.tv_usec));
send(sd,data,48,0);
```

# Results: Analysis Reuse

| Name | # | Functions |
|---|---|---|
| **pppd** | 1 | fwrite |
| **iptables** | 3 | fwrite |
| **rdnssd** | 1 | setsockopt |
| **ntpclient** | 1 | send |
| **speedtest** | 2 | system, fgets |
| **timer** | 2 | read, shutdown |
| **wakeOnLanProxy** | 1 | shutdown |
| **wcnd** | 8 | system |

```
uint32_t data[12];
struct timeval now;

memset(data, 0, sizeof(data));
data[0] = htonl (
        ( LI << 30 ) | ( VN << 27 ) | (
        ( STRATUM << 16) | ( POLL << 8
data[1] = htonl(1<<16);  /* Root Delay
data[2] = htonl(1<<16);  /* Root Disper
gettimeofday(&now,NULL);
data[10] = htonl(now.tv_sec + JAN_1970)
data[11] = htonl(NTPFRAC(now.tv_usec));
send(sd,data,48,0);
```

No false positives on OpenSSL

# Results: Synthesis

# Results: Synthesis

- End-to-end synthesis takes 58 seconds
  - mining sketches, processing traces, and lifter synthesis
- 29 sketches per instruction, on average
  - Synthesis converges after 2 input output pairs, on average
- Lifts 84.8% of native instructions.
  - Missed example: “load upper immediate”

# Generalizing across Architectures

# Generalizing across Architectures

- IR Sketches mined from ARM and x86 are common to both

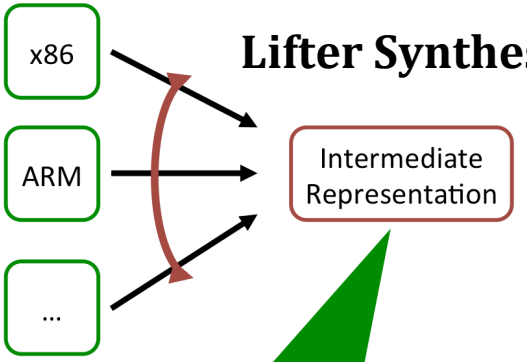
| % | ARM-IR | % | X86-IR |
|---|---|---|---|
| **20.9** | _v := _v | **11.9** | _v := _i |
| **14.5** | _v := _i | **8.6** | jmp _i |
| **8.9** | jmp _i | **5.1** | _v := mem[_v + _i] |
| **7.0** | _v := mem[_v + _i] | **4.4** | mem[_v + _i] := _v |
| **5.6** | _v := _v = _i | **4.2** | _v := _v = _i |
| **5.6** | _v := hi : 1[_v] | **4.2** | _v := hi : 1[_v] |
| **5.5** | mem[_v + _i] := _v | **4.0** | mem[_v + _i] := _v |
| **4.6** | _v := _v - _i | **3.9** | _v := _v - _i |

- Similar MIPS accuracy from either ARM or x86 sketches

# Discussion

- How good “out-of-box”?
- Improve recovery with nondeterministic sketch search and generation
- Expand to more architectures
  - SPARC, PPC, …
- Complement automatic synthesis with (reduced) manual effort
  - Verify and fix manual translation

# Summary




**Lifter Synthesis**

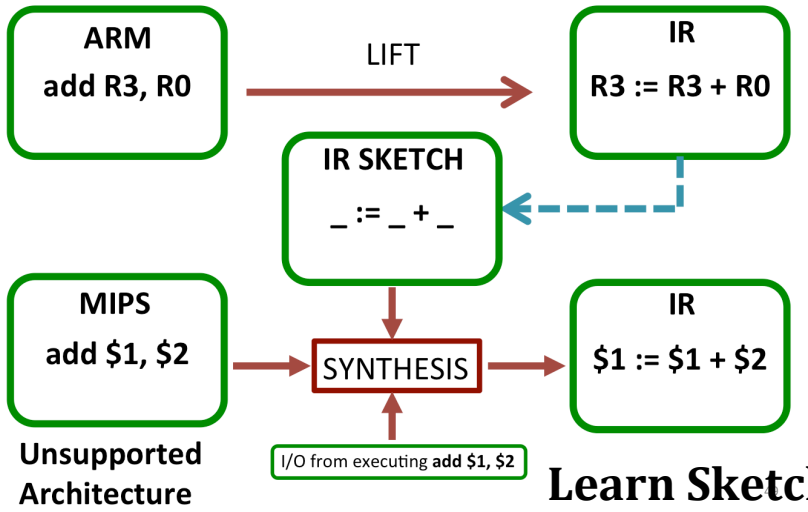
Dataflow framework

Bug-finding Analyses "for free"

VC Generation

Taint analysis






**Learn Sketches**

## Verify with Dynamic Traces

$$R2 := R0$$

$$[(\mathtt{R},\mathtt{REG},\mathtt{R0},\mathtt{0x1}),(\mathtt{W},\mathtt{REG},\mathtt{R2},\mathtt{0x1})]$$

Events from execution

$$\langle \sigma_{\mathbb{T}}, \mathcal{I}_{\mathbb{T}}, \varnothing \rangle \overset{\mathbb{T}}{\rightsquigarrow} \langle \sigma'_{\mathbb{T}}, -, \mathcal{E}_{\mathbb{T}} \rangle \qquad \langle \sigma_{IR}, \mathcal{I}_{IR}, \varnothing \rangle \overset{IR_*}{\rightsquigarrow} \langle \sigma'_{IR}, -, \mathcal{E}_{IR} \rangle$$

| Name | # | Functions |
|---|---|---|
| pppd | 1 | fwrite |
| iptables | 3 | fwrite |
| rdnssd | 1 | setsockopt |
| ntpclient | 1 | send |
| speedtest | 2 | system, fgets |
| timer | 2 | read, shutdown |
| wakeOnLanProxy | 1 | shutdown |
| wcnd | 8 | system |

```
uint32_t data[12];
struct timeval now;

memset(data, 0, sizeof(data));
data[0] = htonl (
        ( LI << 30 ) | ( VN << 27 ) | (
        ( STRATUM << 16) | ( POLL << 8
data[1] = htonl(1<<16);  /* Root Delay
data[2] = htonl(1<<16);  /* Root Disper
gettimeofday(&now,NULL);
data[10] = htonl(now.tv_sec + JAN_1970)
data[11] = htonl(NTPFRAC(now.tv_usec));
send(sd,data,48,0);
```

**Analysis Reuse for Previously Unsupported Instruction Set**