Securing Cryptographic Software via Typed Assembly Language

Shixin Song*, **Tingzhen Dong***, Kosi Nwabueze, Julian Zanders,

Andres Erbsen, Adam Chlipala, Mengjia Yan

Cryptographic Software

Timing Side Channels

Cryptographic Software

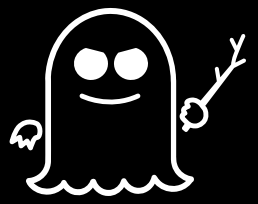
Timing Side Channels

Constant-time Coding Guidelines

```
e.g., a = array[secret]
```

Cryptographic Software

Timing Side Channels



Spectre-like Attacks

Constant-time Coding Guidelines

`e.g., a = array[secret]` 

Speculative Execution violates 

Software

Hardware

[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Software

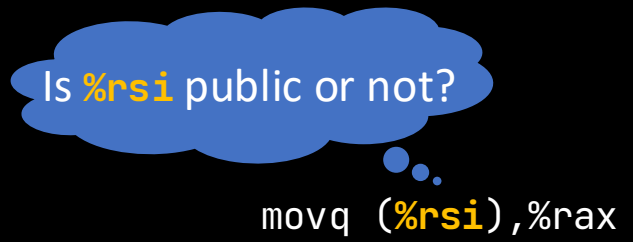
Hardware

```
movq (%rsi),%rax
```


[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

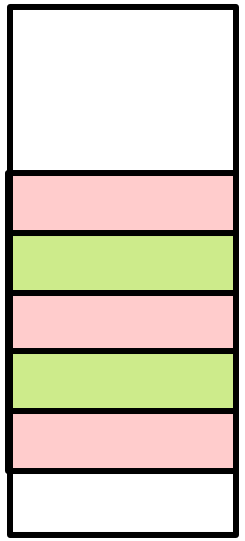
Software

Hardware



[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Software

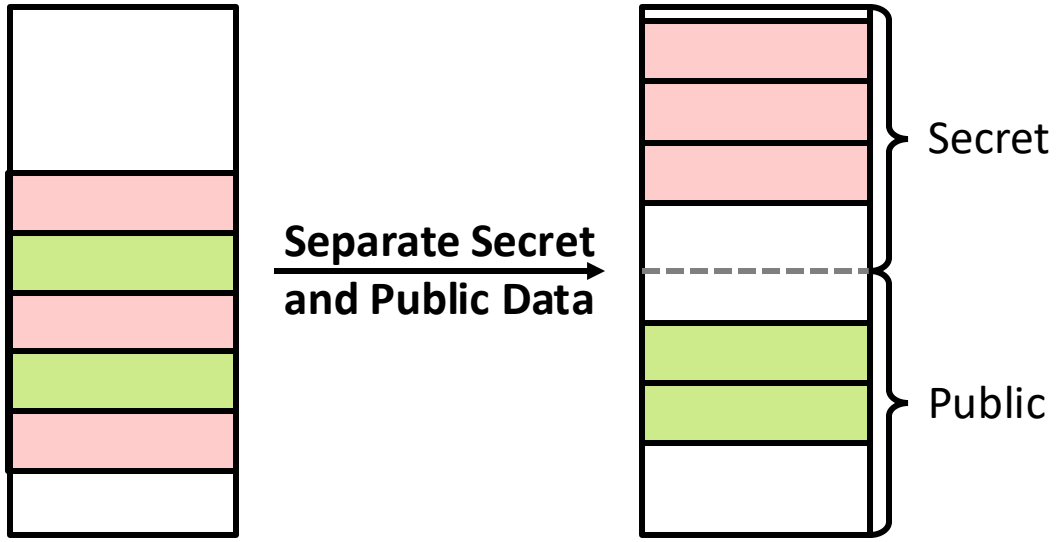


Hardware


```
movq (%rdi),%rsi; movq (%rsi),%rax
```


[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Software



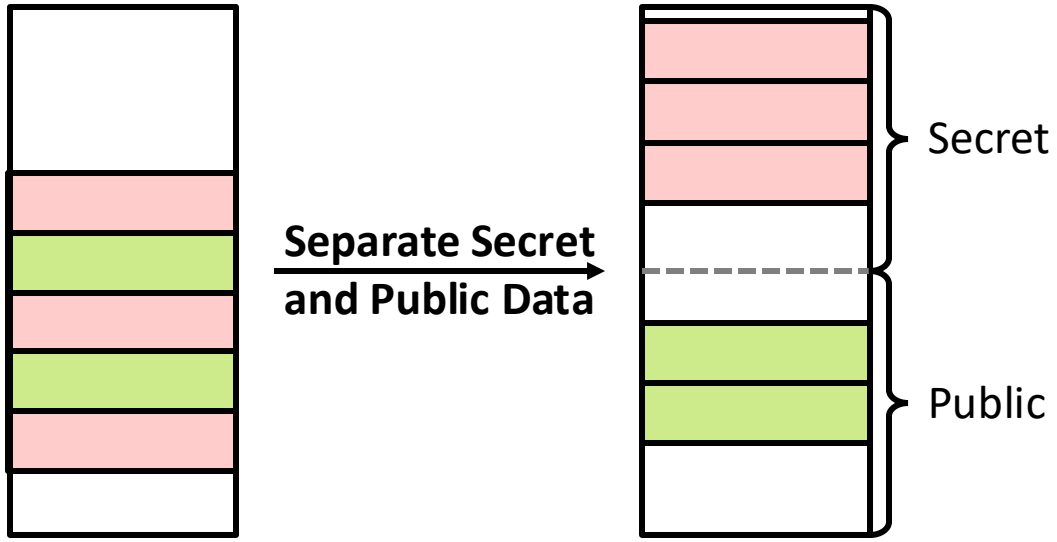
Hardware

Is %rsi public or not?

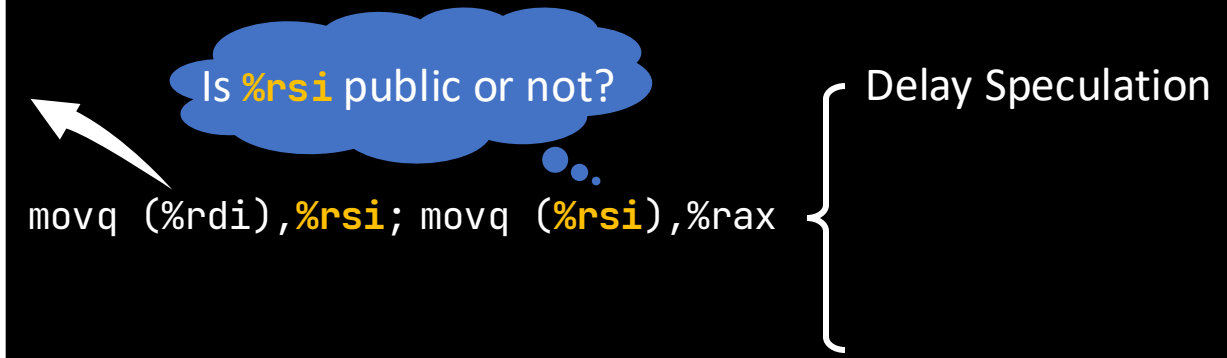
```
movq (%rdi),%rsi; movq (%rsi),%rax
```


[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Software

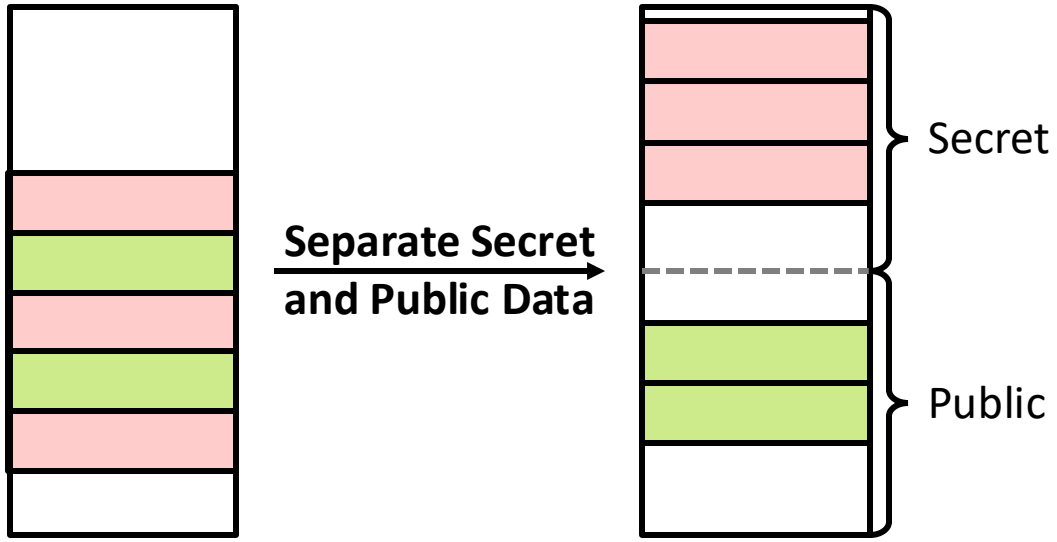


Hardware

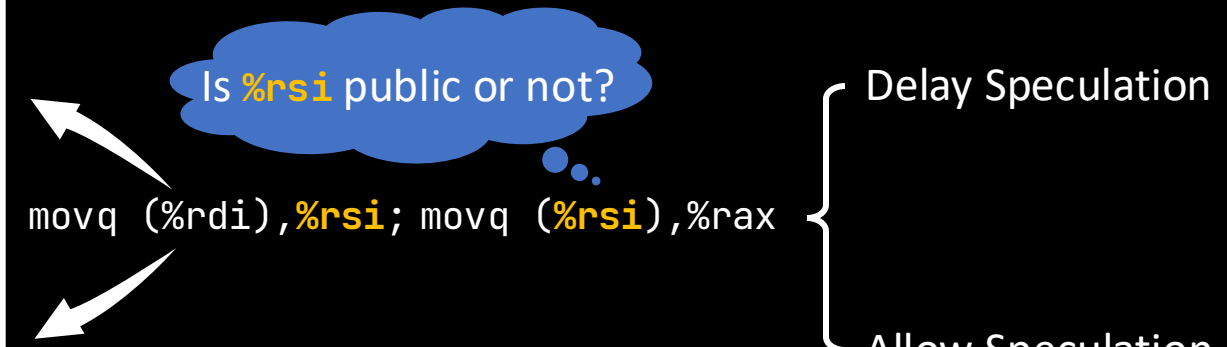


[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Software

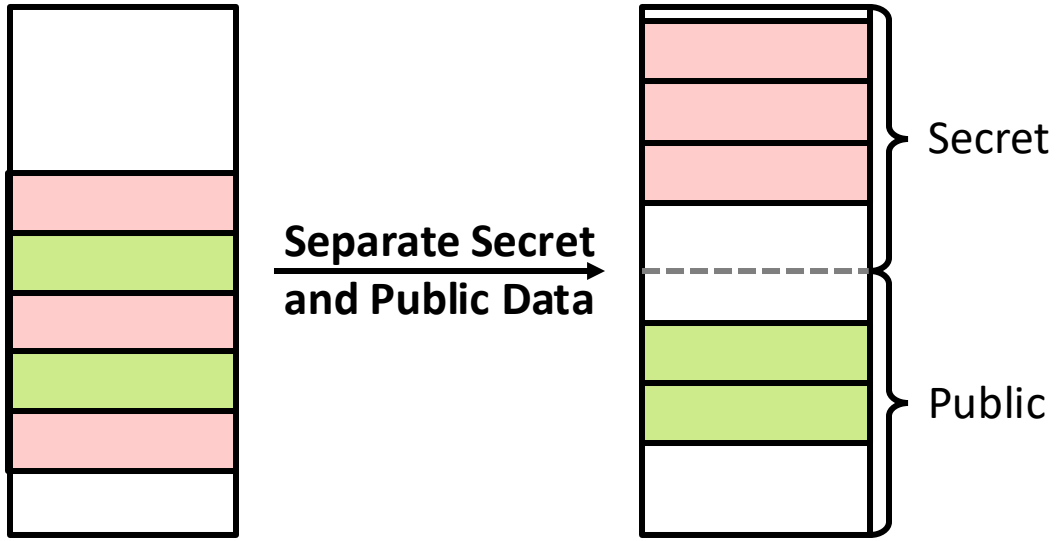


Hardware

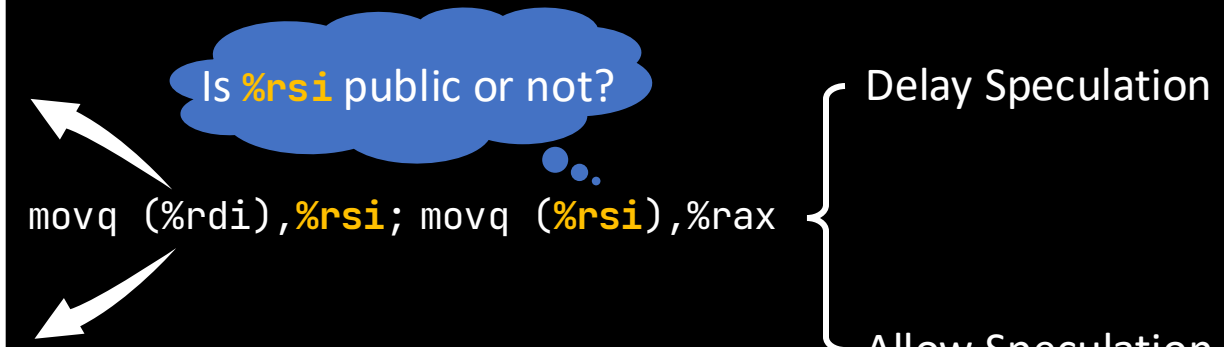


[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Software



Hardware

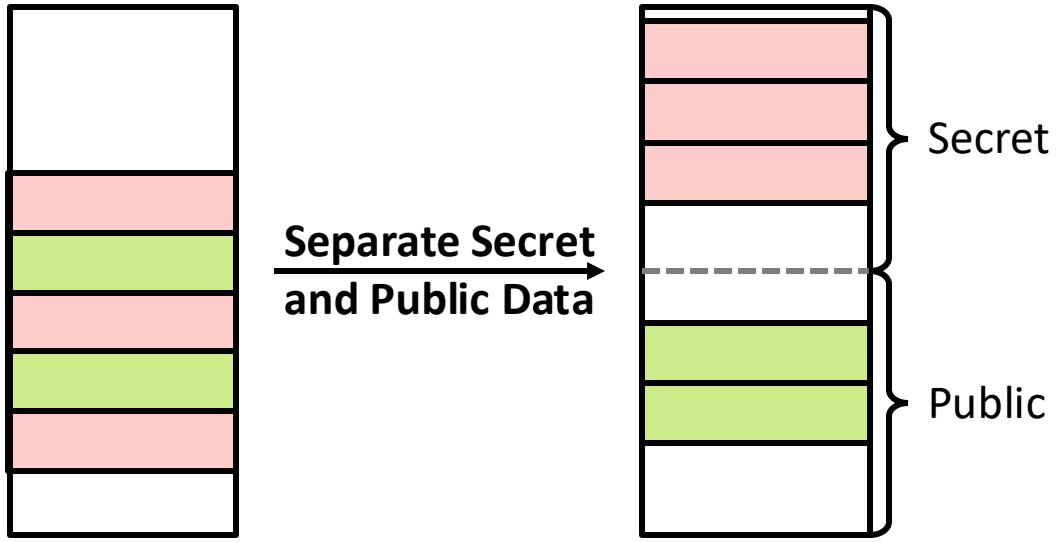


Existing Solution: ProSpeCT[1]

Low hardware cost

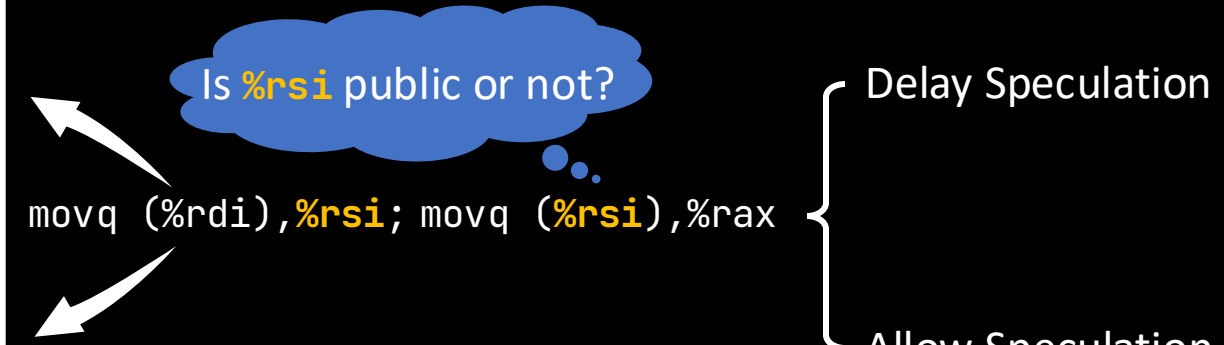
[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Software



Existing solutions **cannot** separate secret and public **stack** data **accurately**

Hardware



Existing Solution: ProSpeCT[1]

Low hardware cost

[1] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. 2023. ProSpeCT: Provably Secure Speculation for the Constant-Time Policy. In 32nd USENIX Security Symposium (USENIX Security 23). 7161–7178.

Separating Secret and Public Stack Data

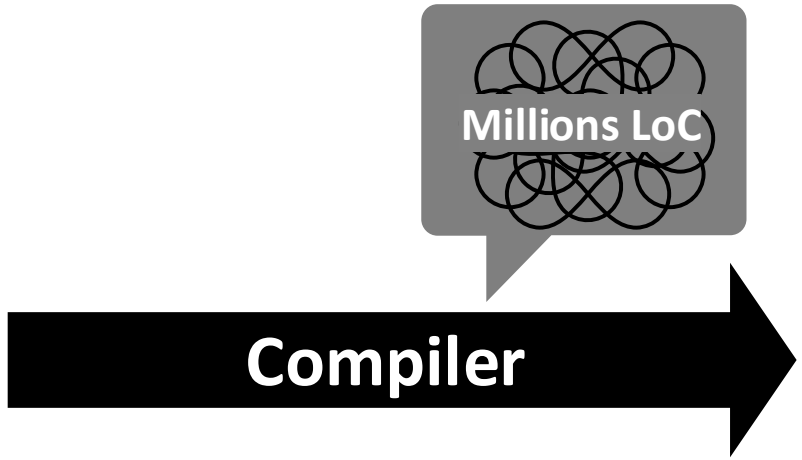
```
void salsa20_words
  (uint32_t *d, uint32_t s[16]) {

  uint32_t x[4][4];
  int i;
  for (i=0; i<16; ++i)
    x[i/4][i%4] = s[i];
  // ...
}
```

Separating Secret and Public Stack Data

```
void salsa20_words
  (uint32_t *d, uint32_t s[16]) {
__attribute__((section("sec"))) static
  uint32_t x[4][4];
  int i;
  for (i=0; i<16; ++i)
    x[i/4][i%4] = s[i];
  // ...
}
```

Separating Secret and Public Stack Data

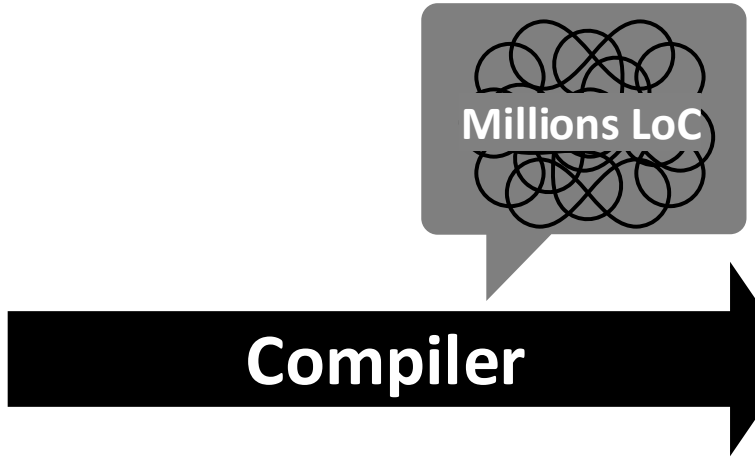
```
void salsa20_words
  (uint32_t *d, uint32_t s[16]) {
__attribute__((section("sec"))) static
  uint32_t x[4][4];
  int i;
  for (i=0; i<16; ++i)
    x[i/4][i%4] = s[i];
  // ...
}
```




```
salsa20_words:
  movq %rdi, -64(%rsp)
  movl (%rsi), %r12d
  movl 4(%rsi), %r11d
  movl 8(%rsi), %r9d
  ...
  // Secret Stack Spill
  movq %rax, -88(%rsp)
  ...
```

Separating Secret and Public Stack Data

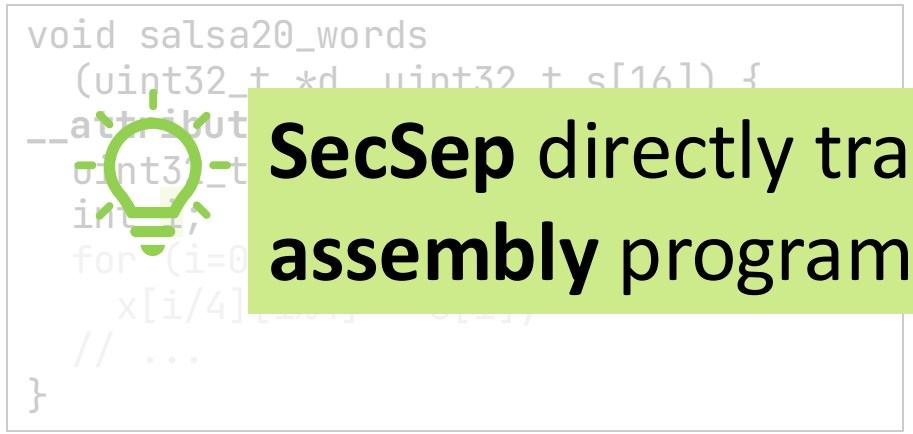
```
void salsa20_words
  (uint32_t *d, uint32_t s[16]) {
__attribute__((section("sec"))) static
  uint32_t x[4][4];
  int i;
  for (i=0; i<16; ++i)
    x[i/4][i%4] = s[i];
  // ...
}
```




```
salsa20_words:
  movq %rdi, -64(%rsp)
  movl (%rsi), %r12d
  movl 4(%rsi), %r11d
  movl 8(%rsi), %r9d
  ...
  // Secret Stack Spill
  movq %rax, -88(%rsp)
  ...
```

Source code annotation cannot separate secret and public **stack** data accurately.

Separating Secret and Public Stack Data



SecSep directly transforms the compiled **assembly** program.

```
salsa20_words:
  movq %rdi, -64(%rsp)
  movl (%rsi), %r12d
  movl 4(%rsi), %r11d
  movl 8(%rsi), %r9d
  ...
  // Secret Stack Spill
  movq %rax, -88(%rsp)
  ...
```

- Accurate stack separation
- Smaller TCB

SecSep: Overview

Compiler-generated assembly

```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq (%rdi), %rax
```

Challenge: Track memory access to secret and public data at the **assembly level**.

SecSep: Overview

Compiler-generated assembly

```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq (%rdi), %rax
```

Typed assembly: Octal

```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq (%rdi), %rax
```

Challenge: Track memory access to secret and public data at the **assembly level**.

Solution: Octal type system + type inference + transformation

SecSep: Overview

Compiler-generated assembly

```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq (%rdi), %rax
```

Type inference

Typed assembly: Octal

```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq (%rdi), %rax
```

Challenge: Track memory access to secret and public data at the **assembly level**.

Solution: Octal type system + type inference + transformation

SecSep: Overview

Compiler-generated assembly

```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq (%rdi), %rax
```

Type inference

Typed assembly: Octal

```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq (%rdi), %rax
```

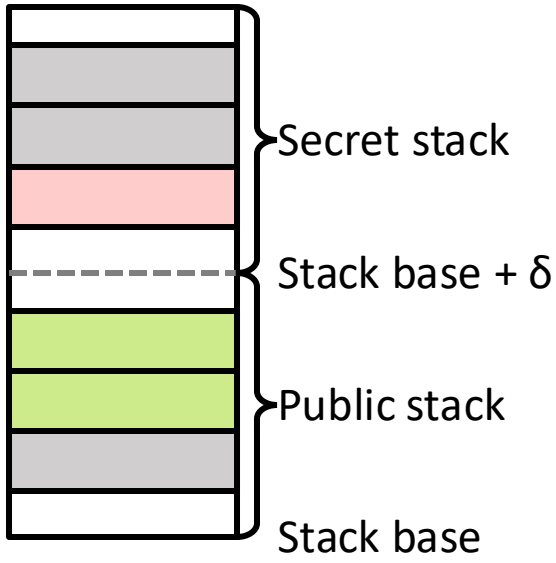
Transformation

Transformed assembly

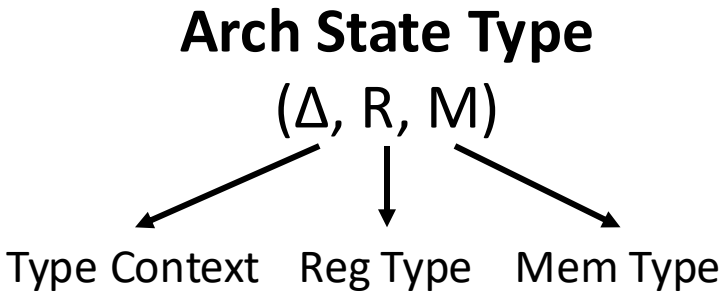
```
foo:
  ...
  movq %rdi, (%rsp)
  ...
  movq (%rsp), %rdi
  movq δ(%rdi), %rax
```

Challenge: Track memory access to secret and public data at the **assembly level**.

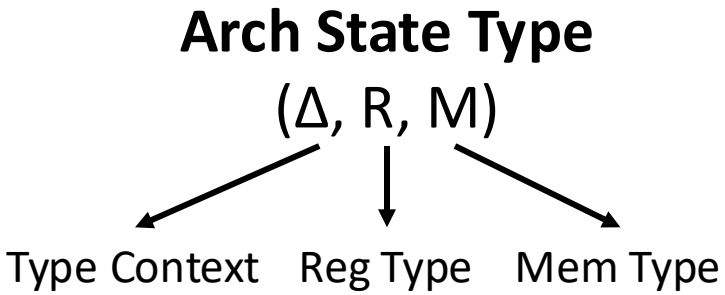
Solution: Octal type system + type inference + transformation

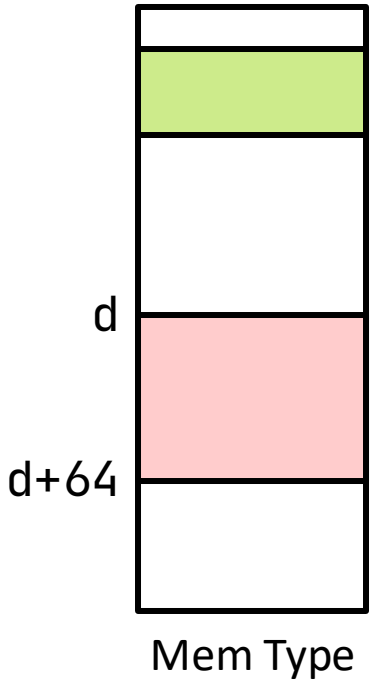


Octal: Typed Assembly Language

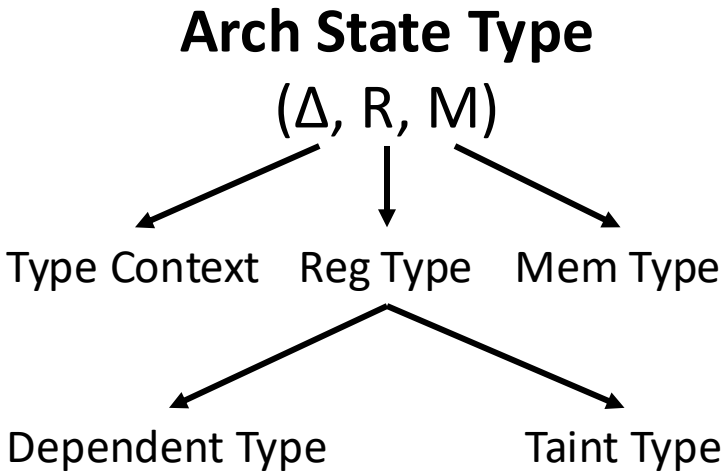


Octal: Typed Assembly Language

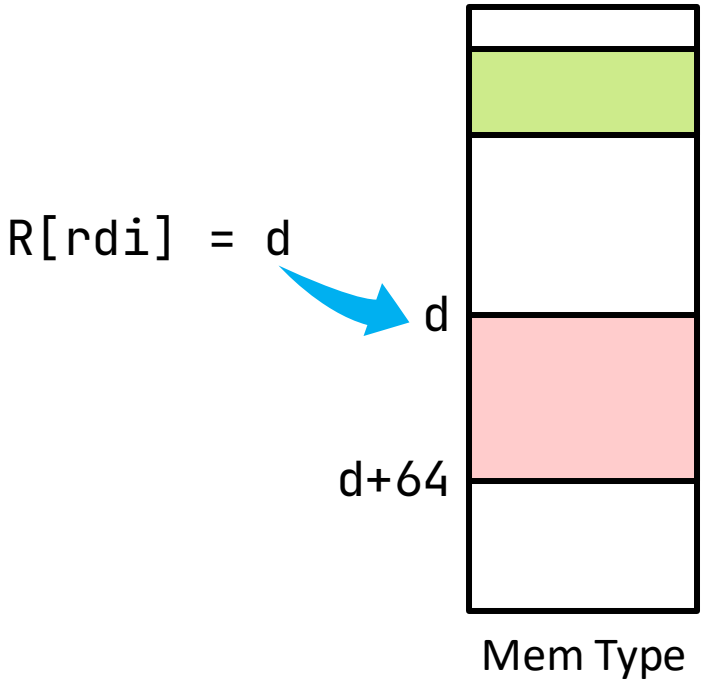




Octal: Typed Assembly Language




```
.L0:
movq (%rdi), %rsi
...
```



Octal: Typed Assembly Language

Arch State Type

(Δ, R, M)

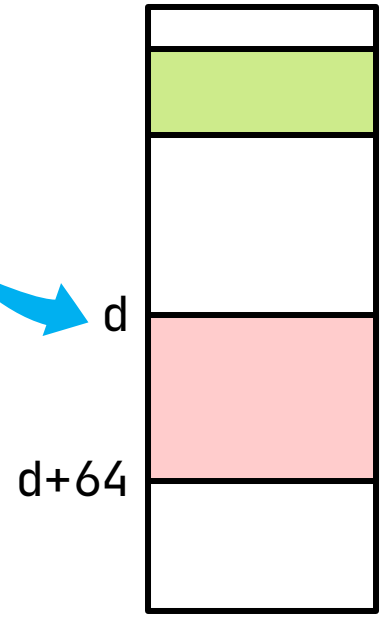
Is subtype of …
(type soundness)

(Δ’, R’, M’)

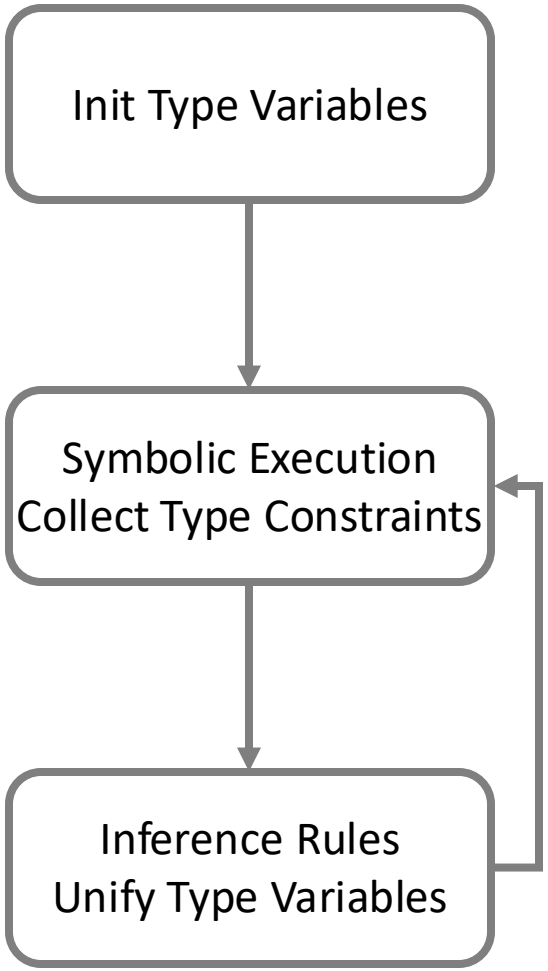
```
.L0:
movq (%rdi), %rsi
...
```

```
.L1:
...
```

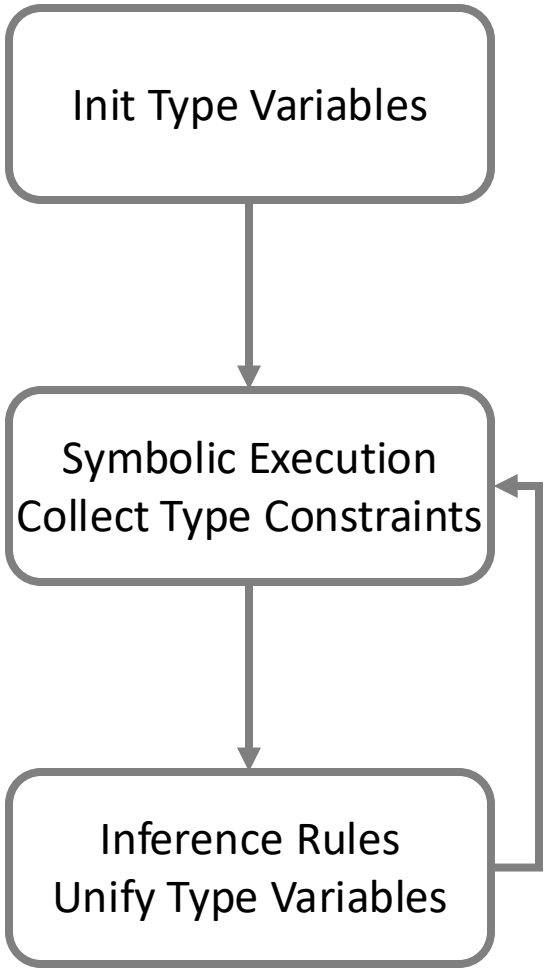
R[rdi] = d



Infer Octal Types for Cryptographic Assembly

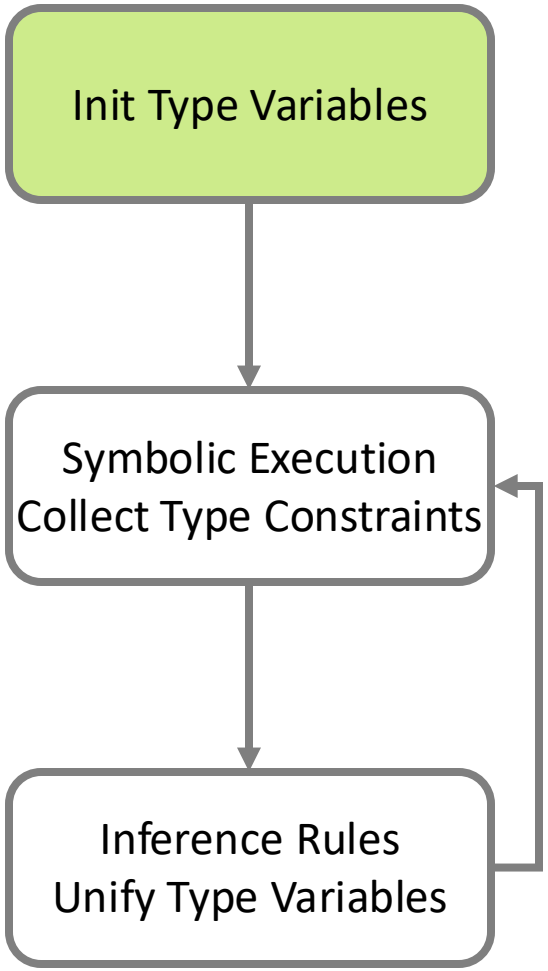


Infer Octal Types for Cryptographic Assembly



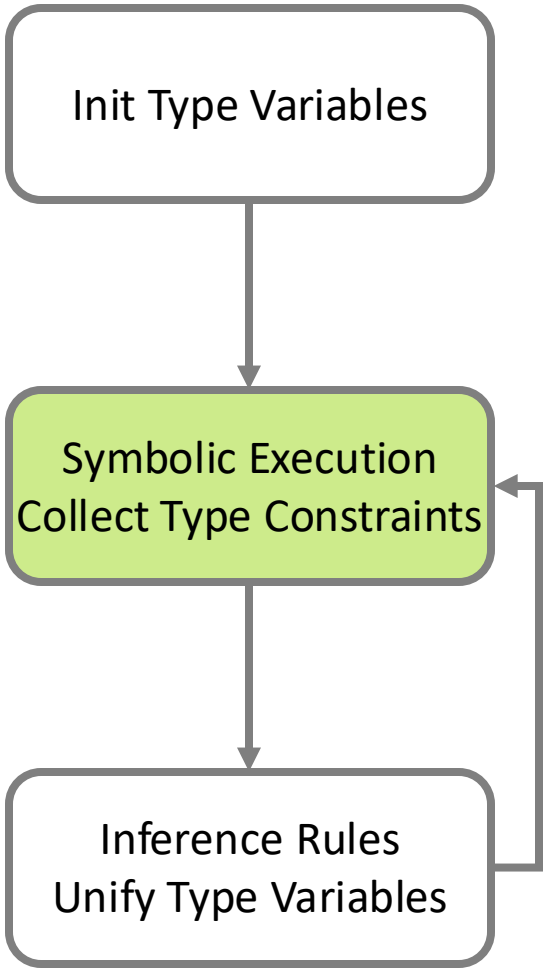

```
.foo:
  movq  (❓)    → ❓
  addq   ❓, $1 → ❓
  cmpq   ❓, $8
  addq   ❓, ❓ → ❓
  jne    .foo
  ...
```

Infer Octal Types for Cryptographic Assembly



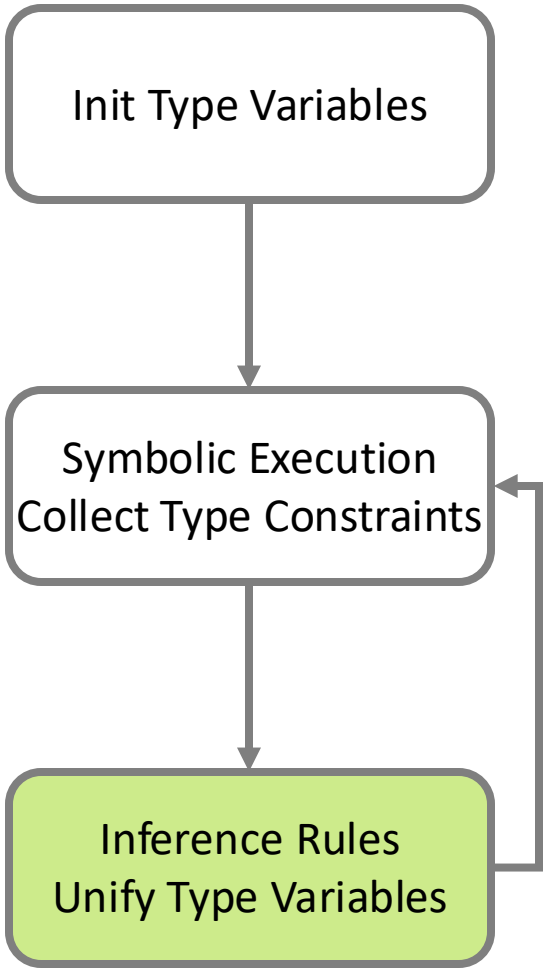

```
.foo:
  movq  (?)    → ?
  addq   ?, $1 → ?
  cmpq   ?, $8
  addq   ?, ? → ?
  jne    .foo
  ...
```

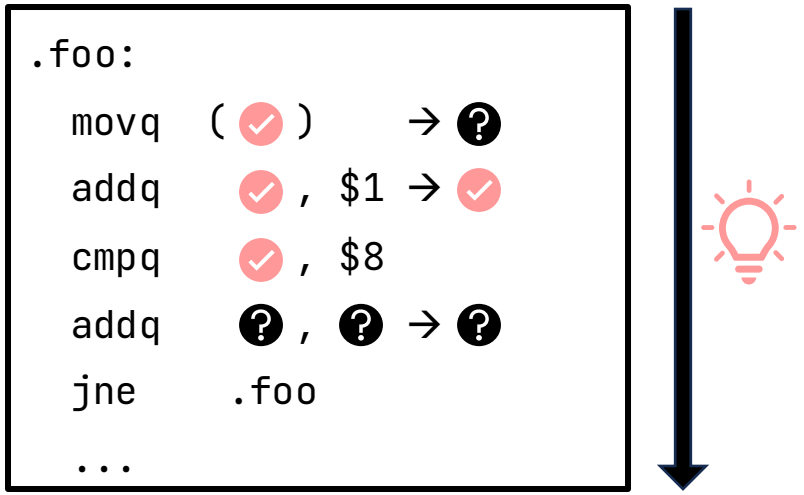
Infer Octal Types for Cryptographic Assembly



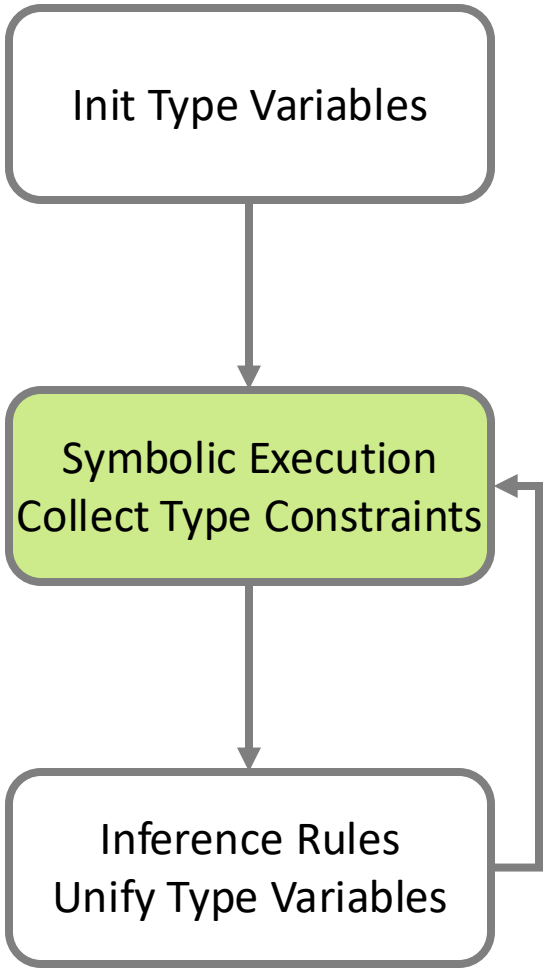

```
.foo:
  movq  (❓)    → ❓
  addq   ❓, $1 → ❓
  cmpq   ❓, $8
  addq   ❓, ❓ → ❓
  jne    .foo
  ...
```

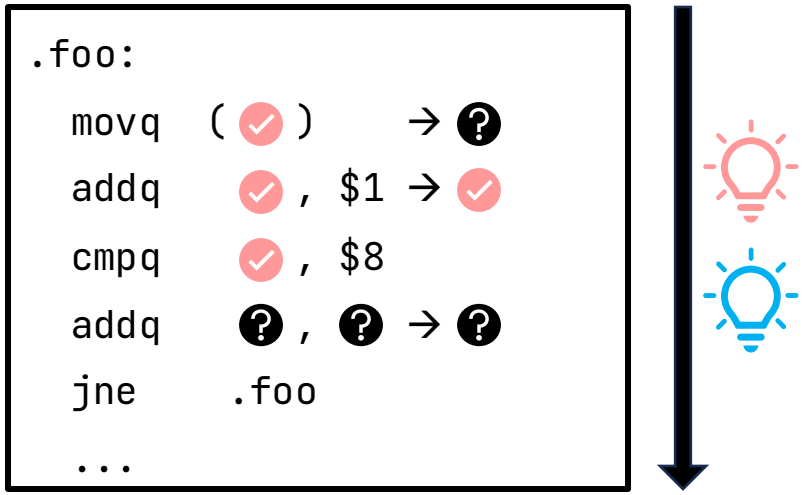
Infer Octal Types for Cryptographic Assembly



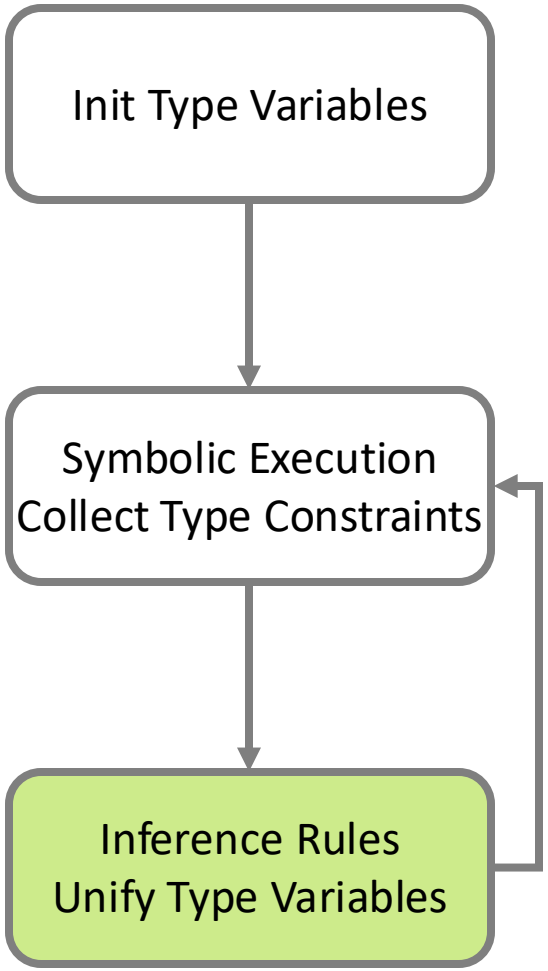


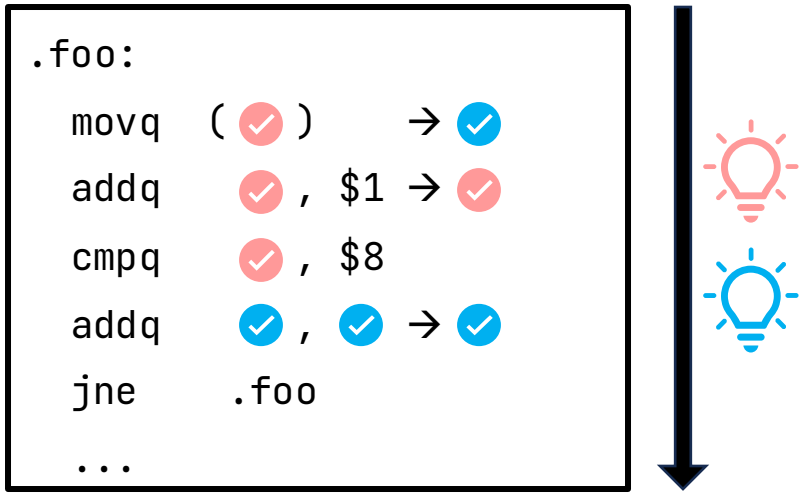
Infer Octal Types for Cryptographic Assembly



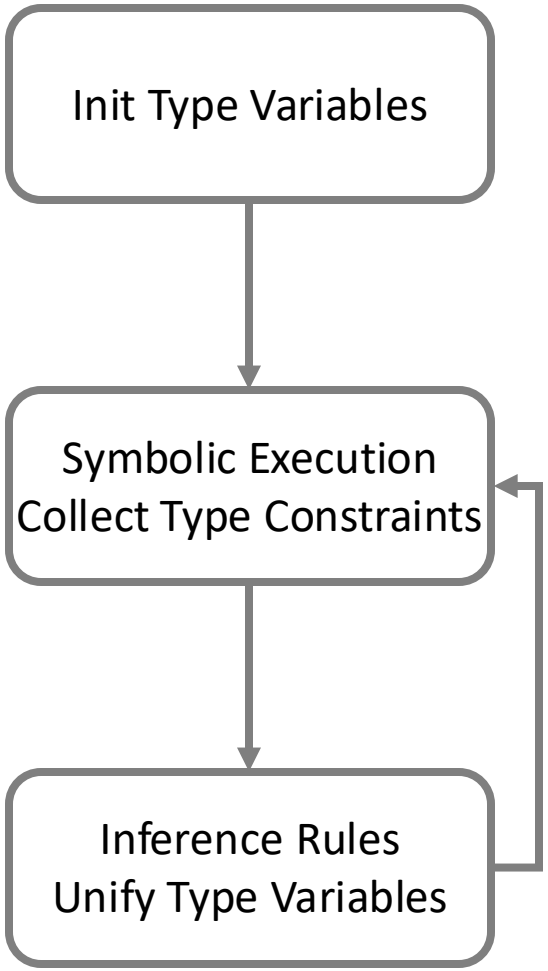


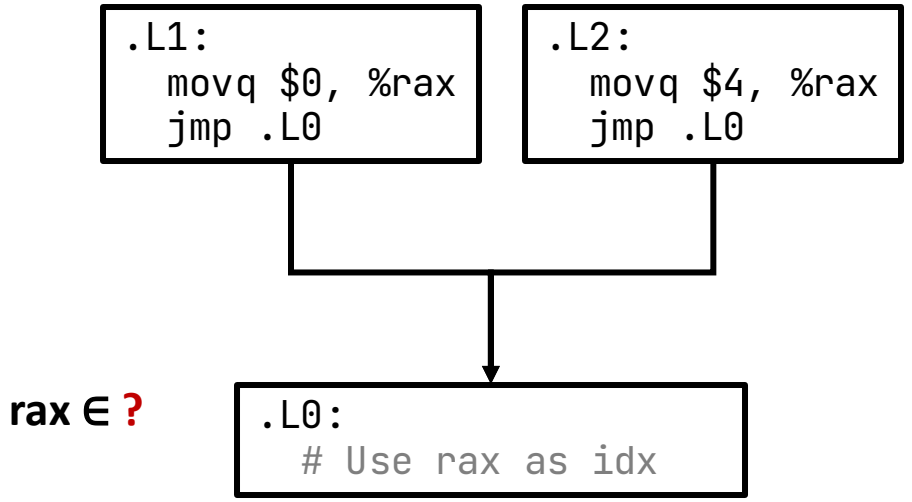
Infer Octal Types for Cryptographic Assembly



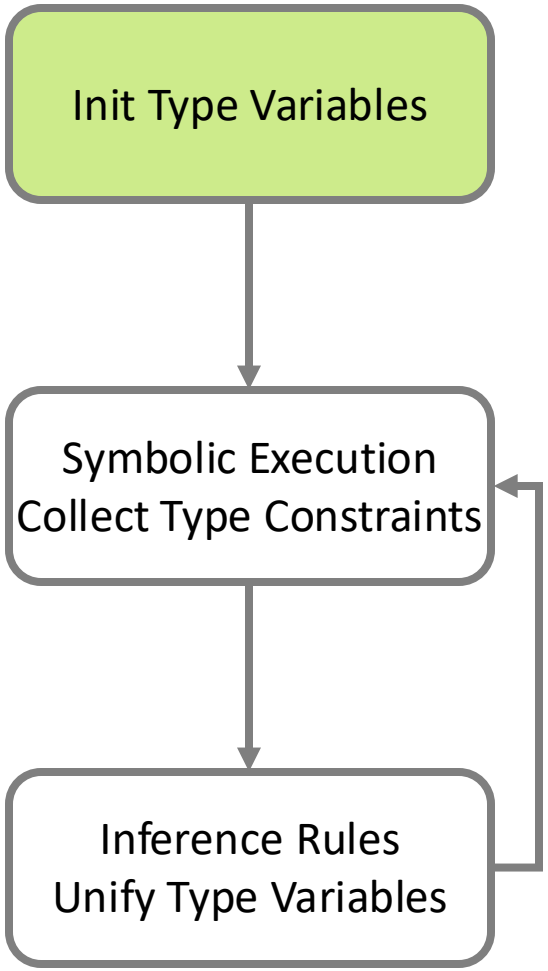


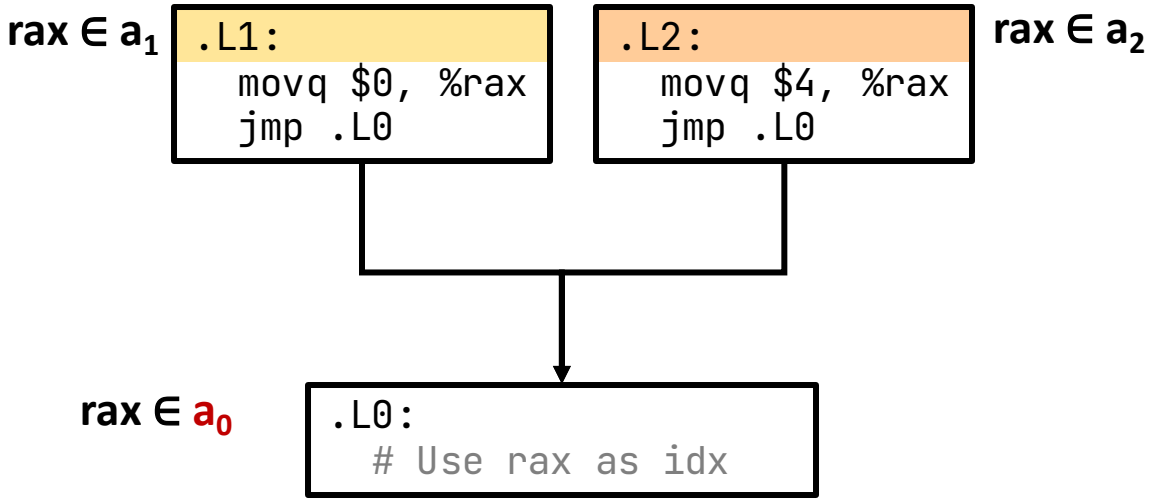
Infer Octal Types for Cryptographic Assembly



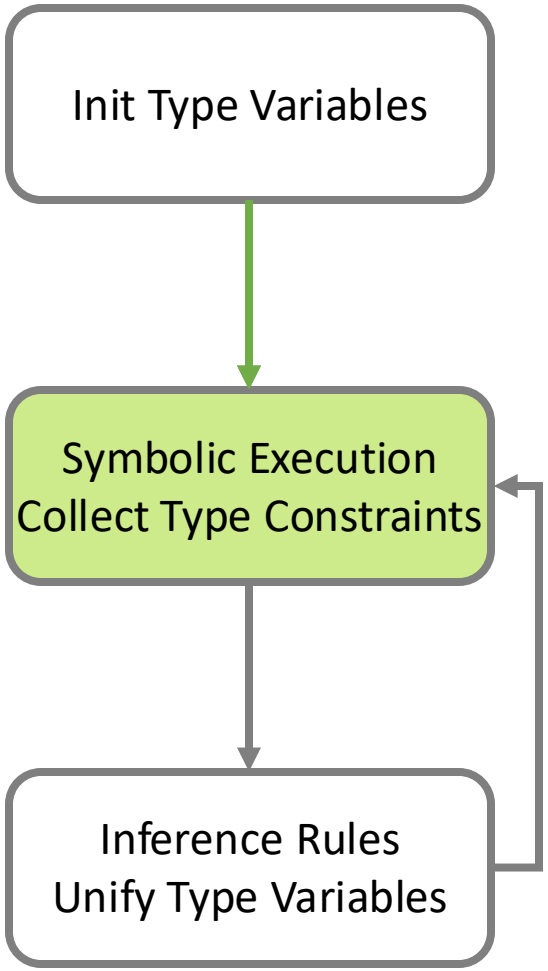


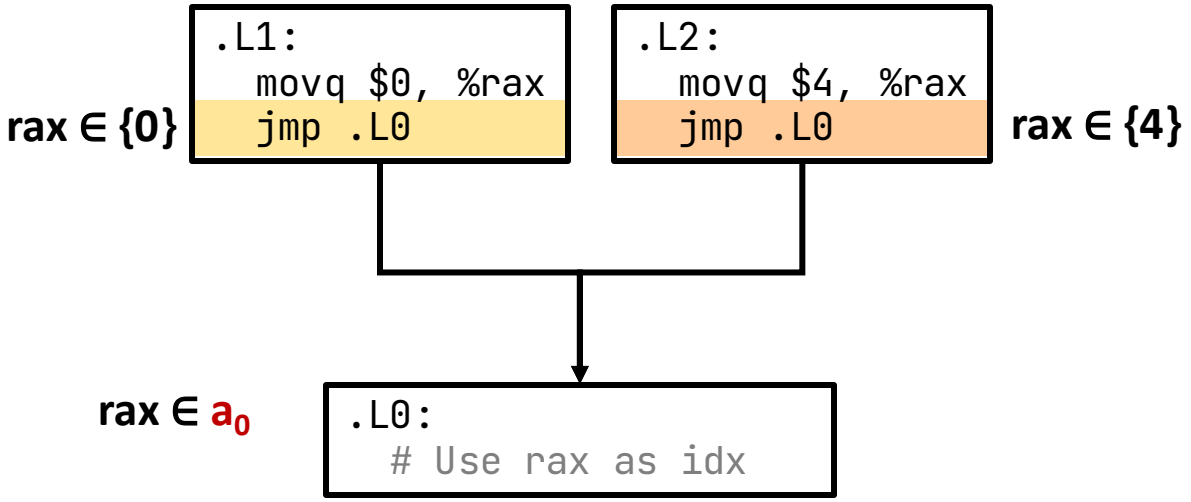
Infer Octal Types for Cryptographic Assembly



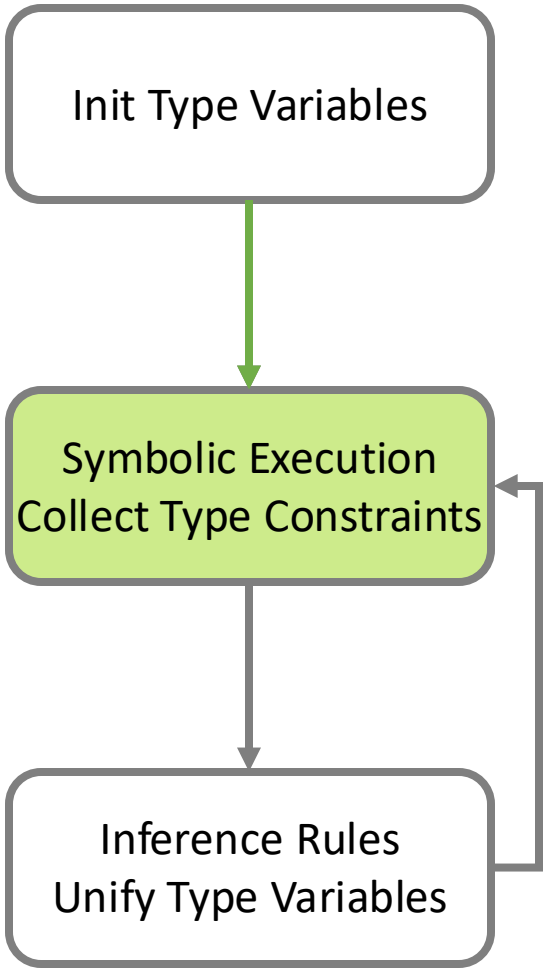


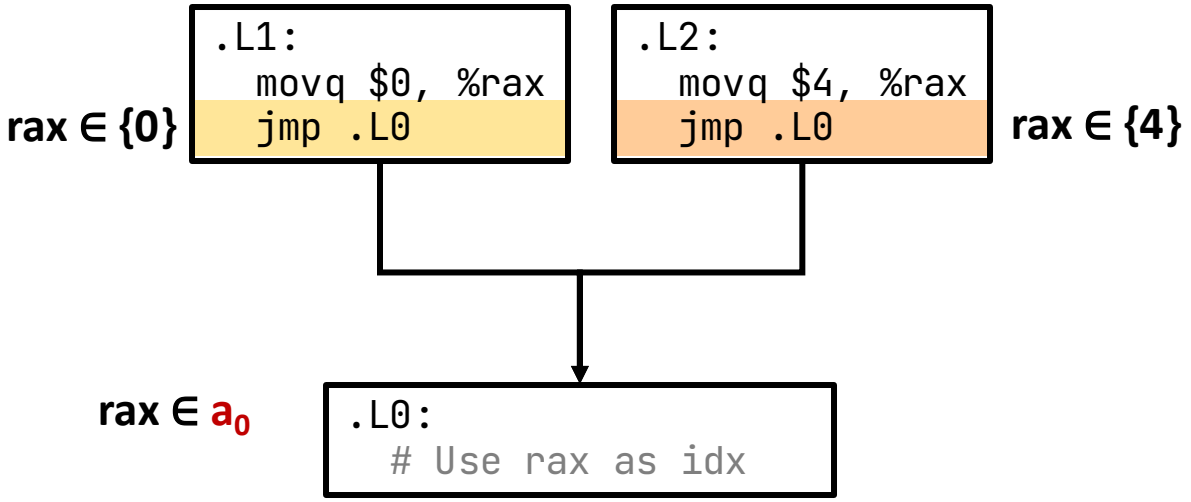
Infer Octal Types for Cryptographic Assembly





Infer Octal Types for Cryptographic Assembly

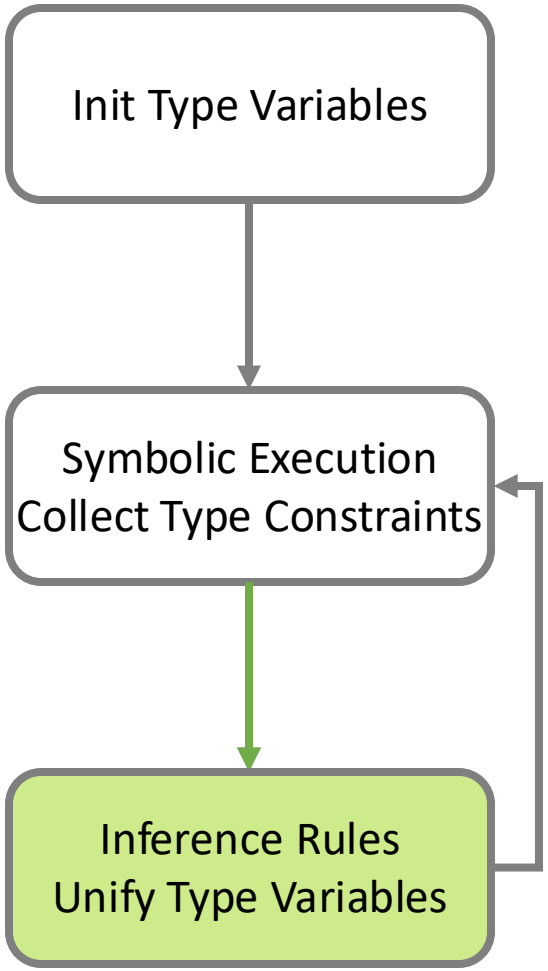


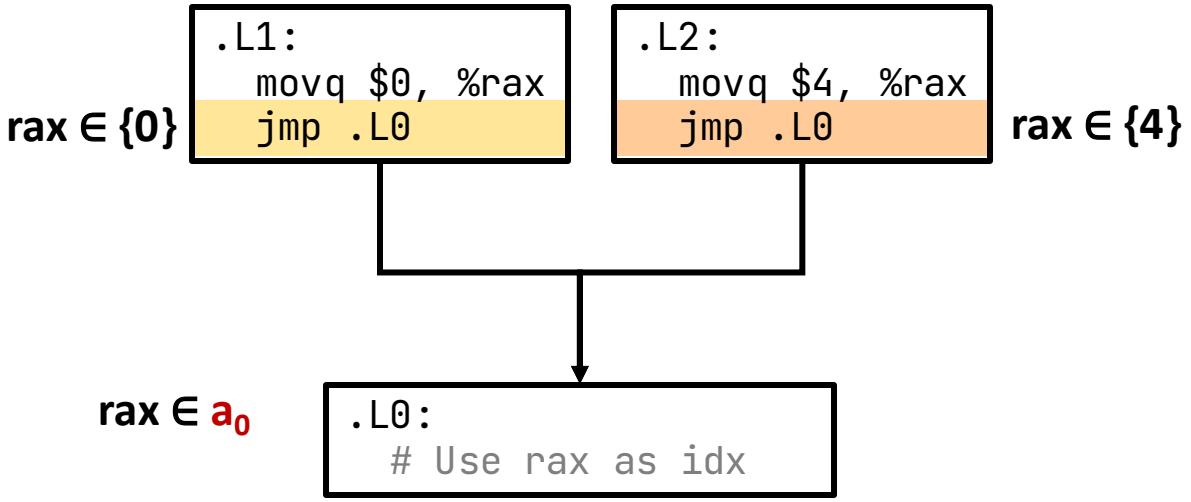


Type constraints to satisfy the tying rules

$\{0\} \subseteq a_0$
$\{4\} \subseteq a_0$

Infer Octal Types for Cryptographic Assembly





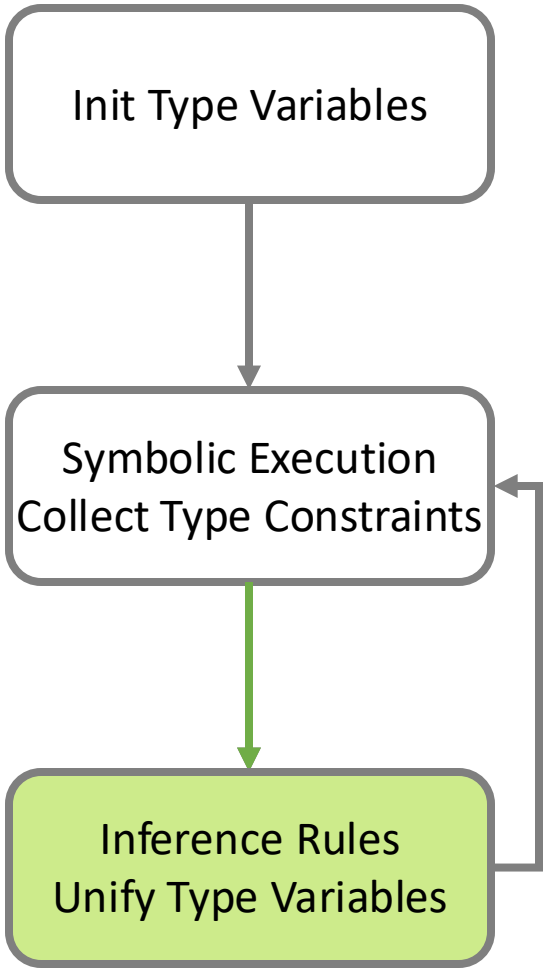
Type constraints to satisfy the tying rules

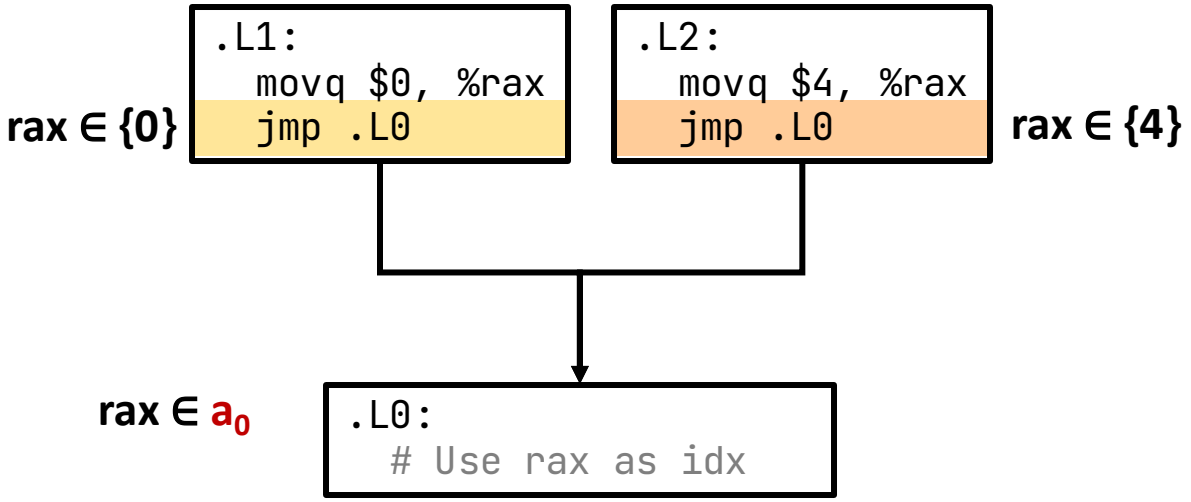
$\{0\} \subseteq a_0$
$\{4\} \subseteq a_0$

Inferred type

$a_0 = \{0, 4\}$

Infer Octal Types for Cryptographic Assembly





Type constraints to satisfy the tying rules

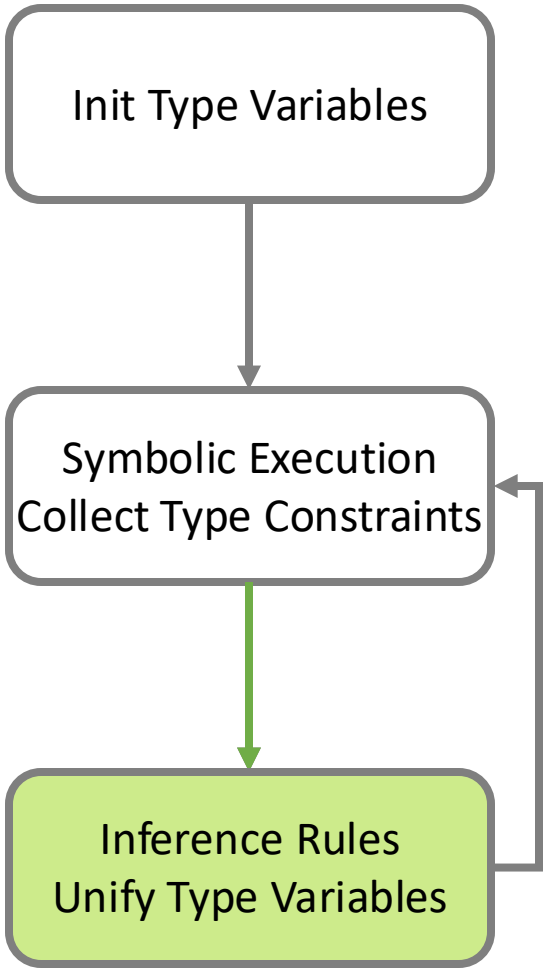
Inferred type

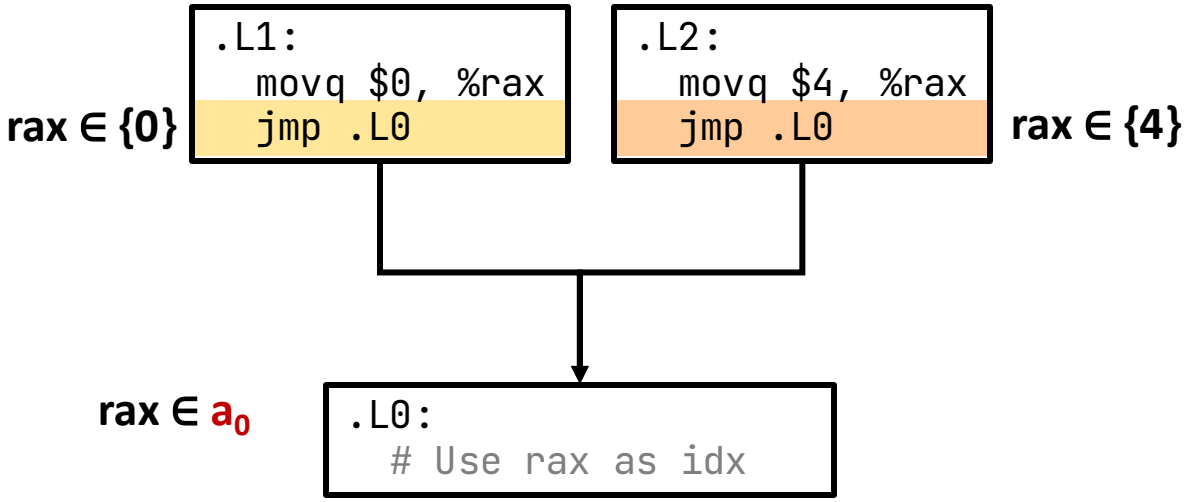
$\{0\} \subseteq a_0$
$\{4\} \subseteq a_0$
→ $a_0 = \{0, 4\}$

$\{0\} \subseteq b_0$
$\{b_0+4\} \subseteq b_0$
$b_0 < 64$
→ $b_0 = \text{range}(0, 64, \text{step}=4)$

And more inference rules…

Infer Octal Types for Cryptographic Assembly





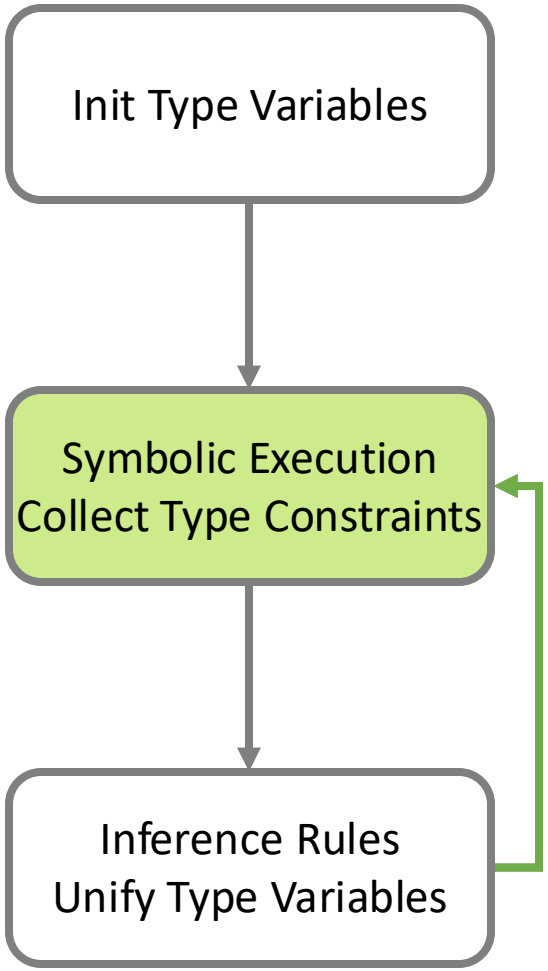
Type constraints to satisfy the tying rules

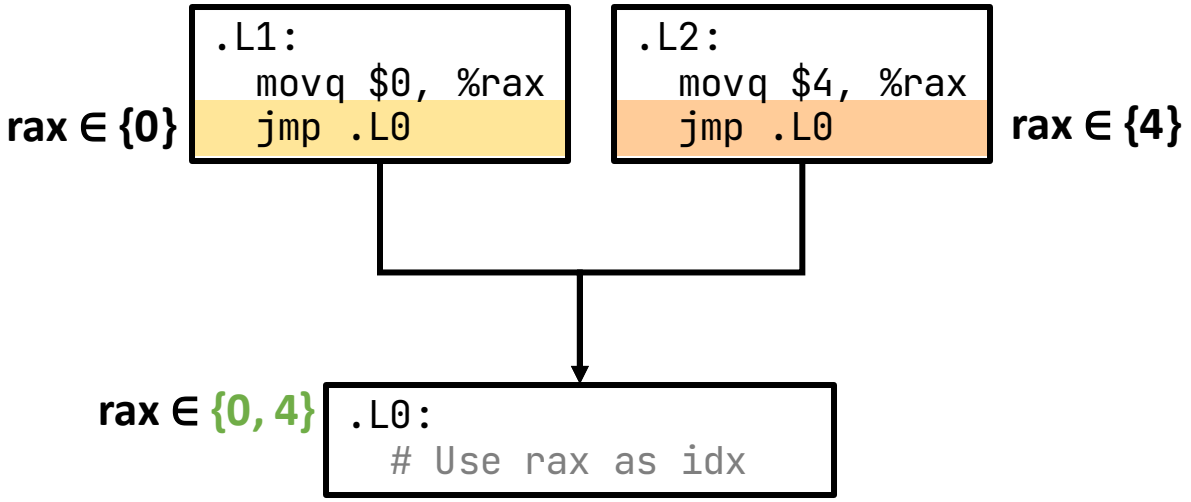
$\{0\} \subseteq a_0$
$\{4\} \subseteq a_0$

Inferred type

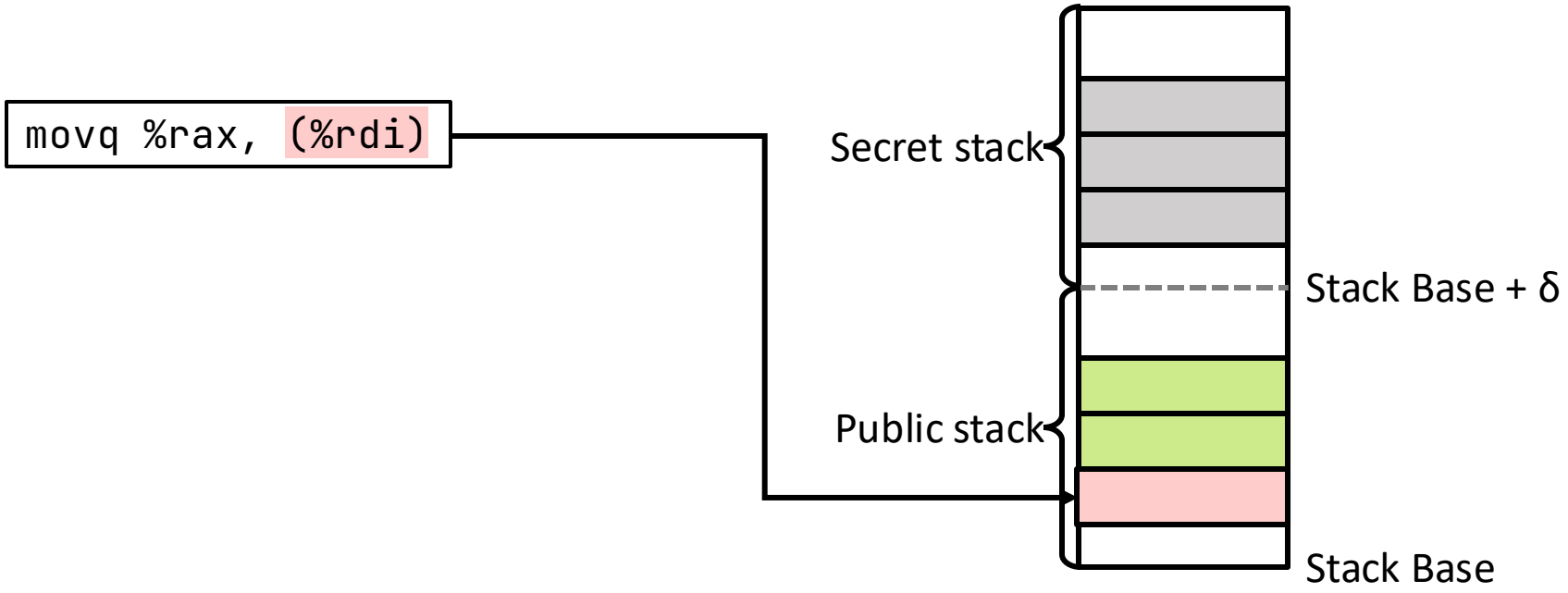
$a_0 = \{0, 4\}$

Infer Octal Types for Cryptographic Assembly



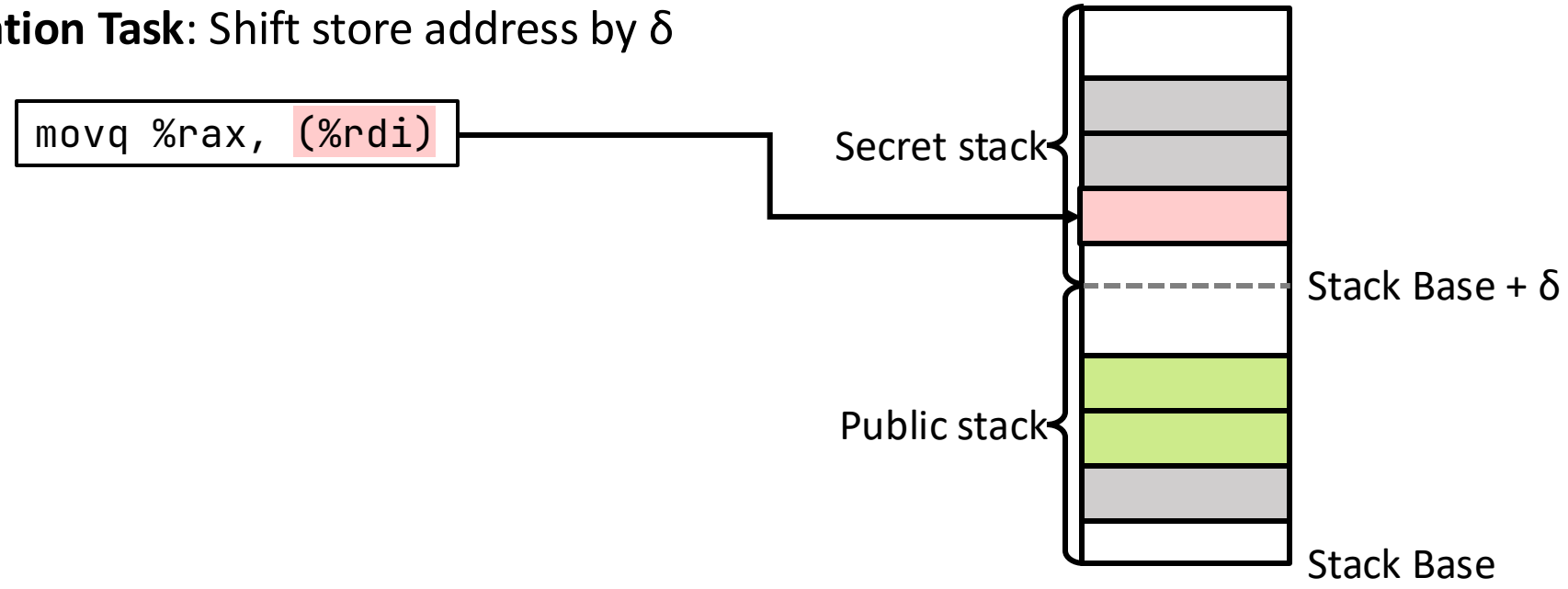


Two Transformation Strategies

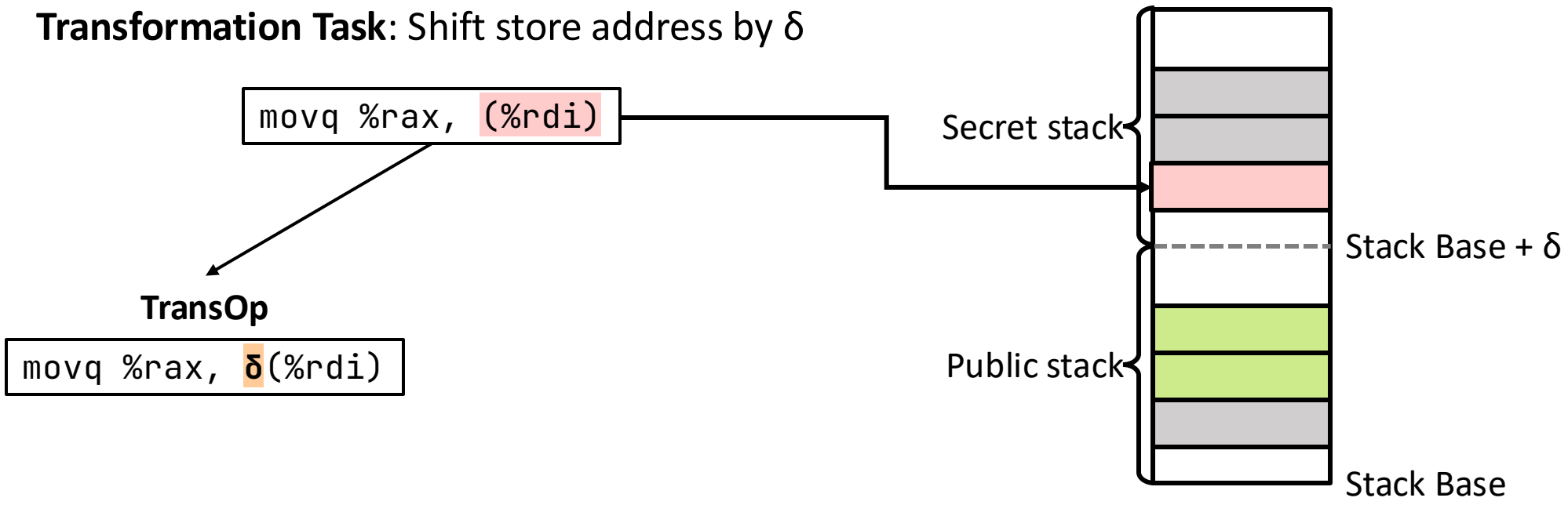


Two Transformation Strategies

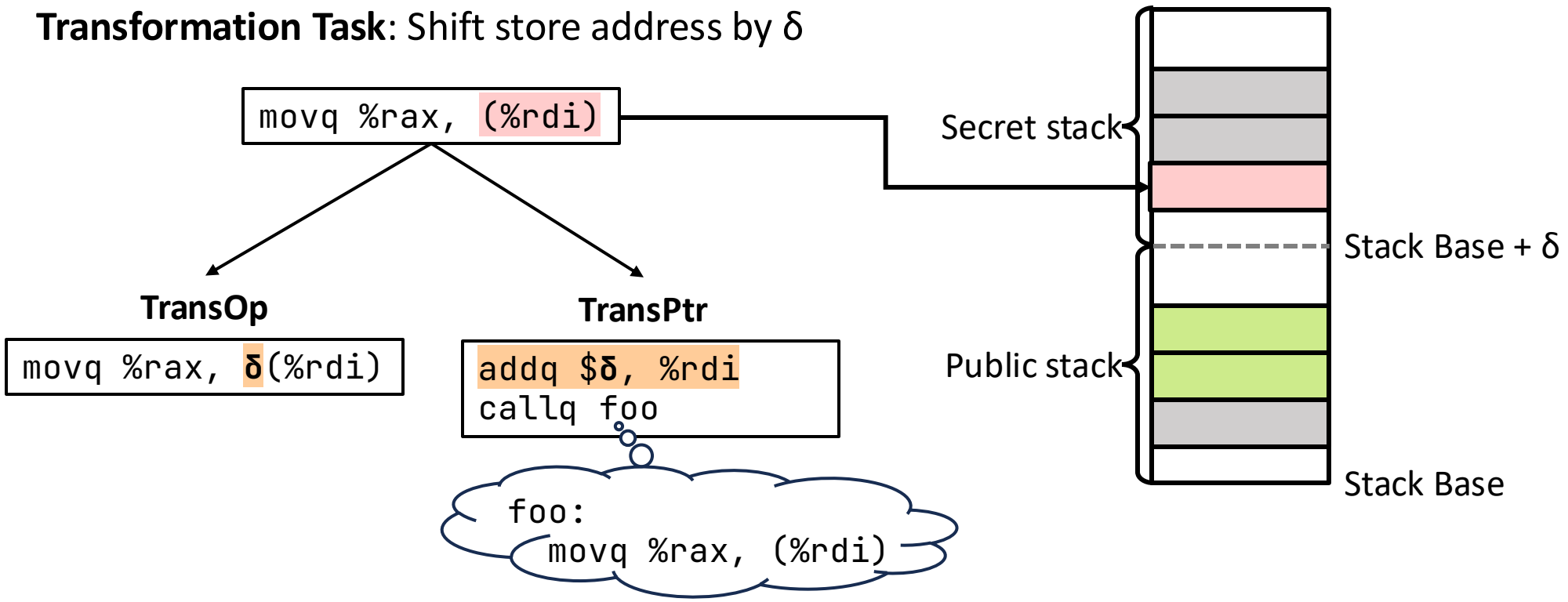
Transformation Task: Shift store address by δ



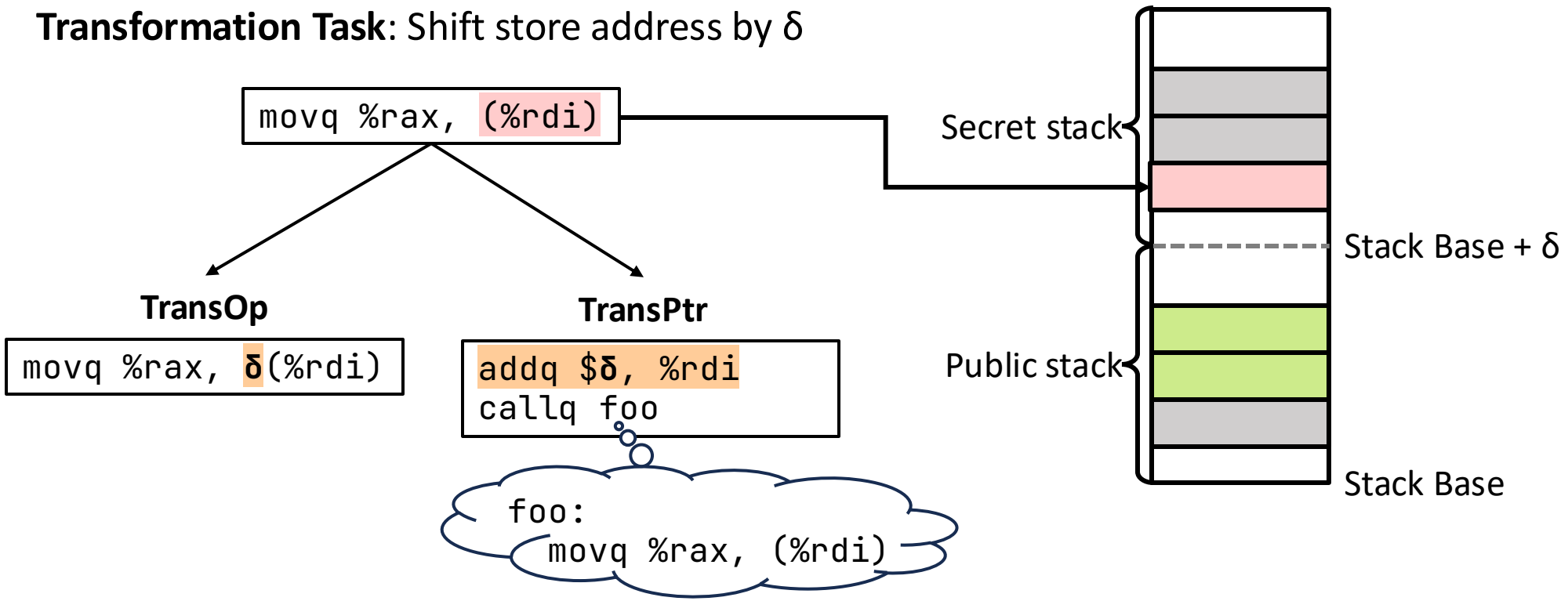
Two Transformation Strategies



Two Transformation Strategies



Two Transformation Strategies



? Which transformation strategy should we use?

Two Transformation Strategies

Example 1: Pointer is a function argument

Example 2: Base pointer references secret and public data (e.g., struct)

Two Transformation Strategies

Example 1: Pointer is a function argument

`rdi`: sometimes points to **secret**, sometimes **public**

```
// memset(rdi, rsi, size)
memset:
  movb %rsi, (%rdi)
```

Example 2: Base pointer references secret and public data (e.g., struct)

Two Transformation Strategies

Example 1: Pointer is a function argument

`rdi`: sometimes points to **secret**, sometimes **public**

```
// memset(rdi, rsi, size)
memset:
  movb %rsi, (%rdi)
```

✗ **TransOp**

✓ **TransPtr** at call site

```
// At call site
  addq $δ, %rdi
  callq memset
```

Example 2: Base pointer references secret and public data (e.g., struct)

Two Transformation Strategies

Example 1: Pointer is a function argument

`rdi`: sometimes points to **secret**, sometimes **public**

```
// memset(rdi, rsi, size)
memset:
  movb %rsi, (%rdi)
```

TransOp

TransPtr at call site

```
// At call site
  addq $δ, %rdi
  callq memset
```

Example 2: Base pointer references secret and public data (e.g., struct)

```
// rdi points to a struct
  movq $sec, (%rdi)
  movq $pub, 8(%rdi)
```

Two Transformation Strategies

Example 1: Pointer is a function argument

`rdi`: sometimes points to **secret**, sometimes **public**

```
// memset(rdi, rsi, size)
memset:
  movb %rsi, (%rdi)
```

TransOp

TransPtr at call site

```
// At call site
  addq $δ, %rdi
  callq memset
```

Example 2: Base pointer references secret and public data (e.g., struct)

```
// rdi points to a struct
  movq $sec, (%rdi)
  movq $pub, 8(%rdi)
```

TransOp

TransPtr

Two Transformation Strategies

Example 1: Pointer is a function argument

`rdi`: sometimes points to **secret**, sometimes **public**

```
// memset(rdi, rsi, size)
memset:
  movb %rsi, (%rdi)
```

TransOp

TransPtr at call site

```
// At call site
  addq $δ, %rdi
  callq memset
```

Example 2: Base pointer references secret and public data (e.g., struct)

```
// rdi points to a struct
  movq $sec, (%rdi)
  movq $pub, 8(%rdi)
```

TransOp

TransPtr

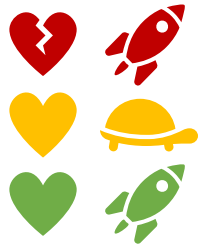
```
// rdi points to a struct
  movq $sec, δ(%rdi)
  movq $pub, 8(%rdi)
```

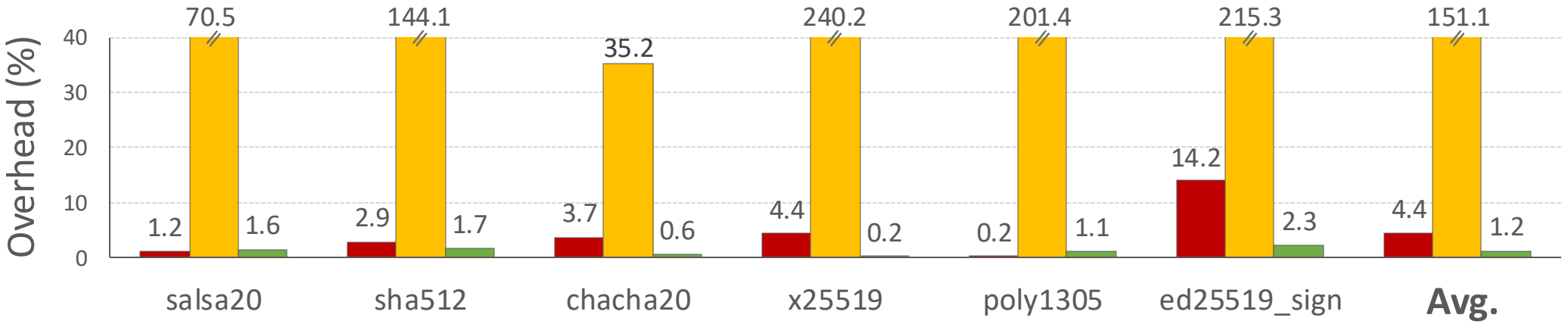
Performance Evaluation Results[1]

ProSpeCT (public stack): Compiler can spill secret data to the public stack

ProSpeCT (secret stack): Conservatively mark public stack data as secret

SecSep: Accurately separate public and secret stack data

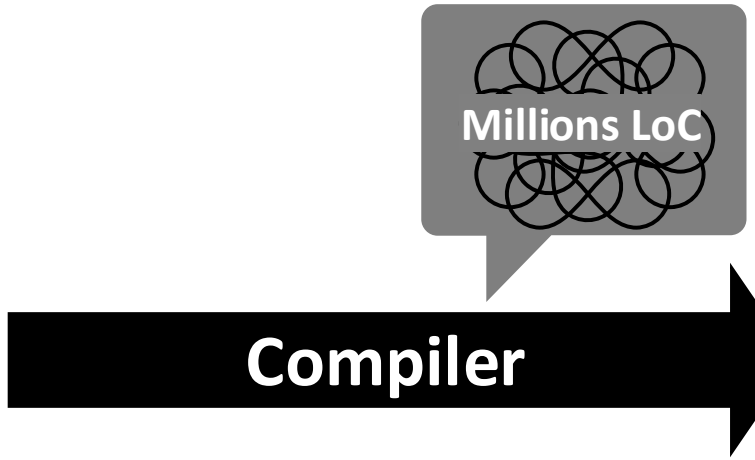




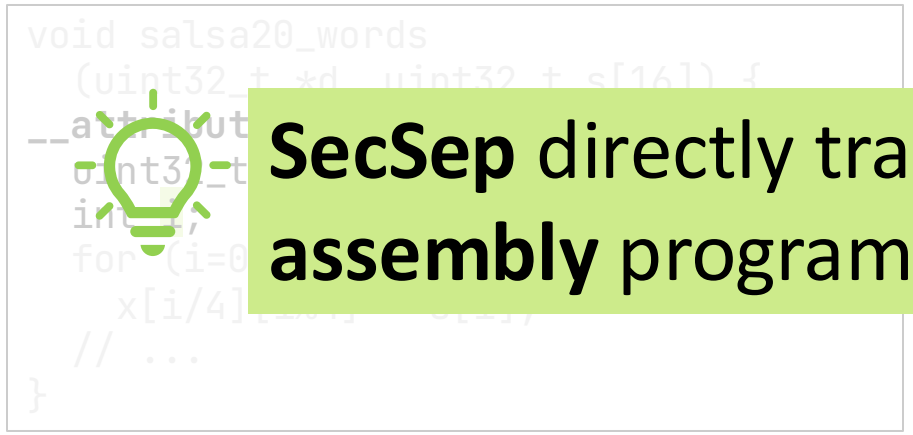
SecSep introduces a low overhead of **1.2%.**

[1] Gem5 is used to implement the CPU with defense and perform evaluation.

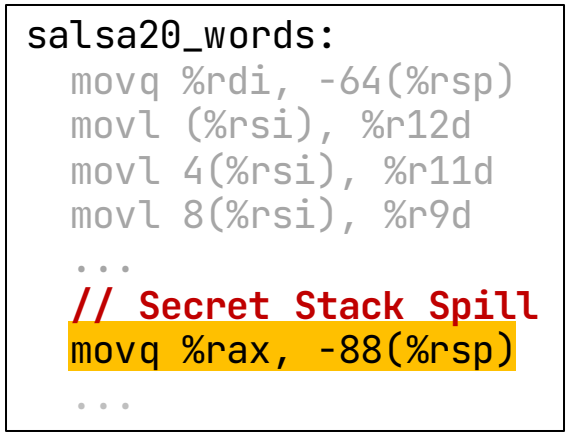
```
void salsa20_words
  (uint32_t *d, uint32_t s[16]) {
__attribute__((section("sec"))) static
  uint32_t x[4][4];
  int i;
  for (i=0; i<16; ++i)
    x[i/4][i%4] = s[i];
  // ...
}
```




```
salsa20_words:
  movq %rdi, -64(%rsp)
  movl (%rsi), %r12d
  movl 4(%rsi), %r11d
  movl 8(%rsi), %r9d
  ...
  // Secret Stack Spill
  movq %rax, -88(%rsp)
  ...
```



Millions LoC



SecSep directly transforms the compiled **assembly** program.

Accurate stack separation

Smaller TCB