Spectre Declassified:
Reading from the Right Place at the Wrong Time

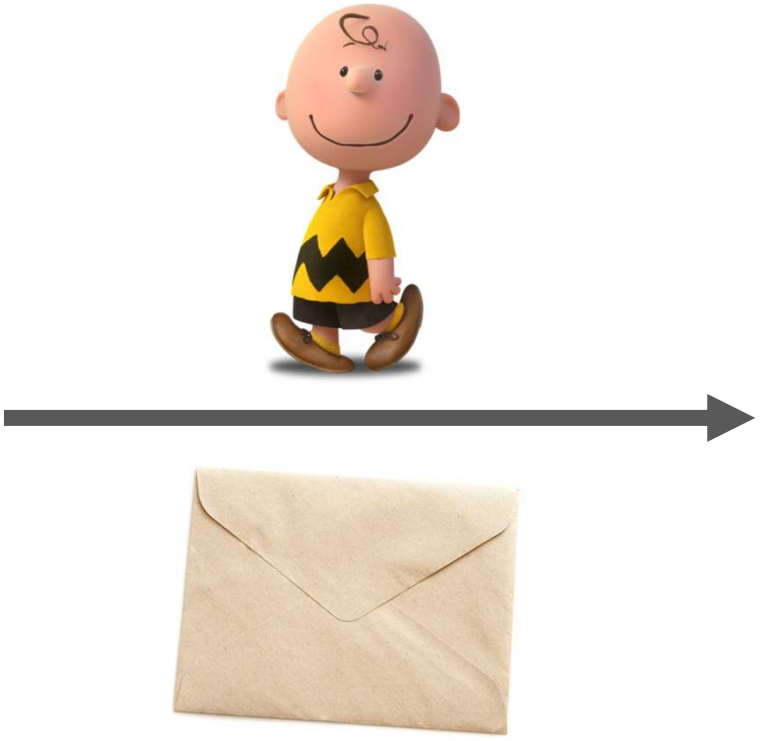
Chitchanok Chuengsatiansup

Joint work with:

Basavesh Ammanaghatta Shivakumar, Jack Barnes,
Gilles Barthe, Sunjay Cauligi, Daniel Genkin, Sioli O'Connell,
Peter Schwabe, Rui Qi Sim, and Yuval Yarom

Background

Background



Motivating Example

```
key = ...
state = message
for (i from 0 to 8) {
    bit = 1 << i
    if (key & bit) { state ^= bit }
}
```

Motivating Example

```
key = ...
state = message
for (i from 0 to 8) {
    bit = 1 << i
    if (key & bit) { state ^= bit }
}
```

Applies a one-time-pad one bit a time

Motivating Example

```
key = ...
state = message
for (i from 0 to 8) {
    bit = 1 << i
    if (key & bit) { state ^= bit }
}
```

Applies a one-time-pad one bit a time

Branches on key material

Background

Influences

Background

Influences

Infer

Constant-Time Programming

- Prevents Secret-Dependent
  - Memory Accesses
  - Control Flow
  - Instruction Timings

Motivating Example

```
key = ...
state = message
for (i from 0 to 8) {
    bit = 1 << i
    state ^= (key & bit)
}
```

No longer uses a branch

Motivating Example

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
```

Motivating Example

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    if (key & bit) { state ^= bit }
}
```

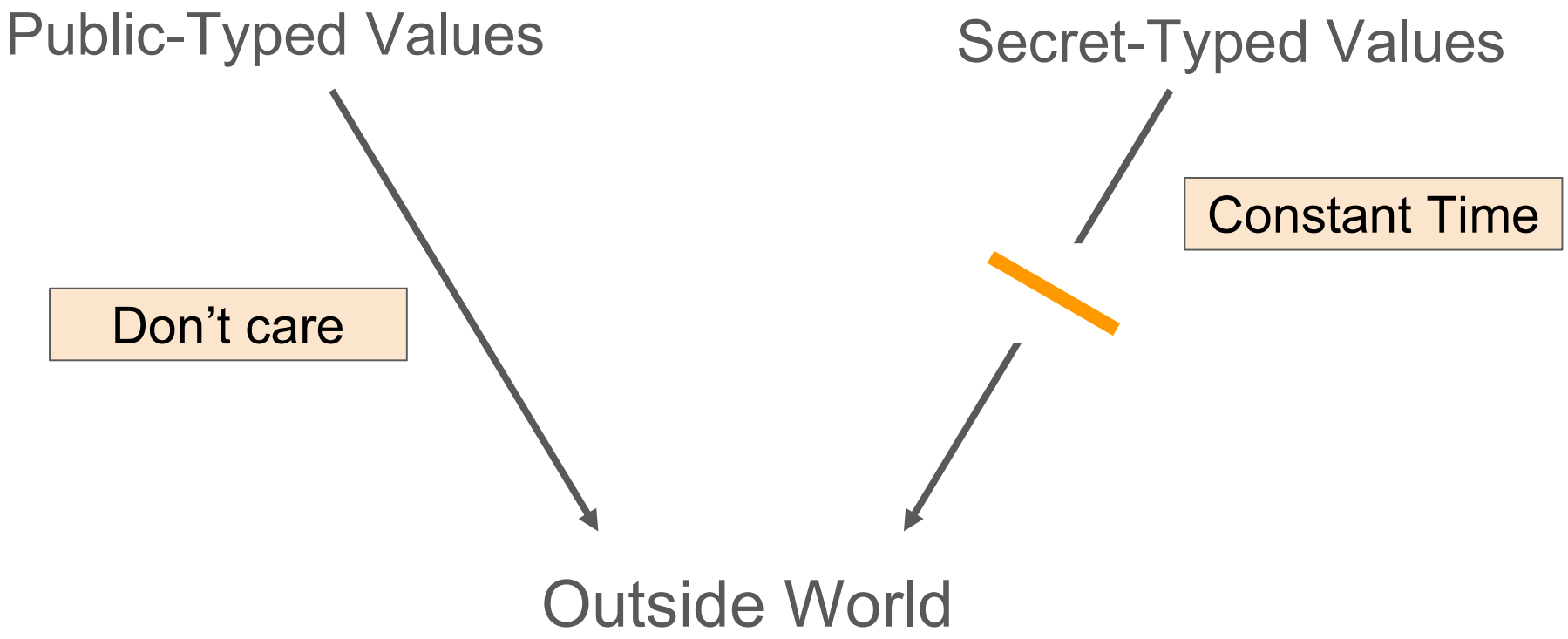
Motivating Example

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    if (key & bit) { state ^= bit }
}
```

Motivating Example

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
```

Flow of Values



Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
```

Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = state
```

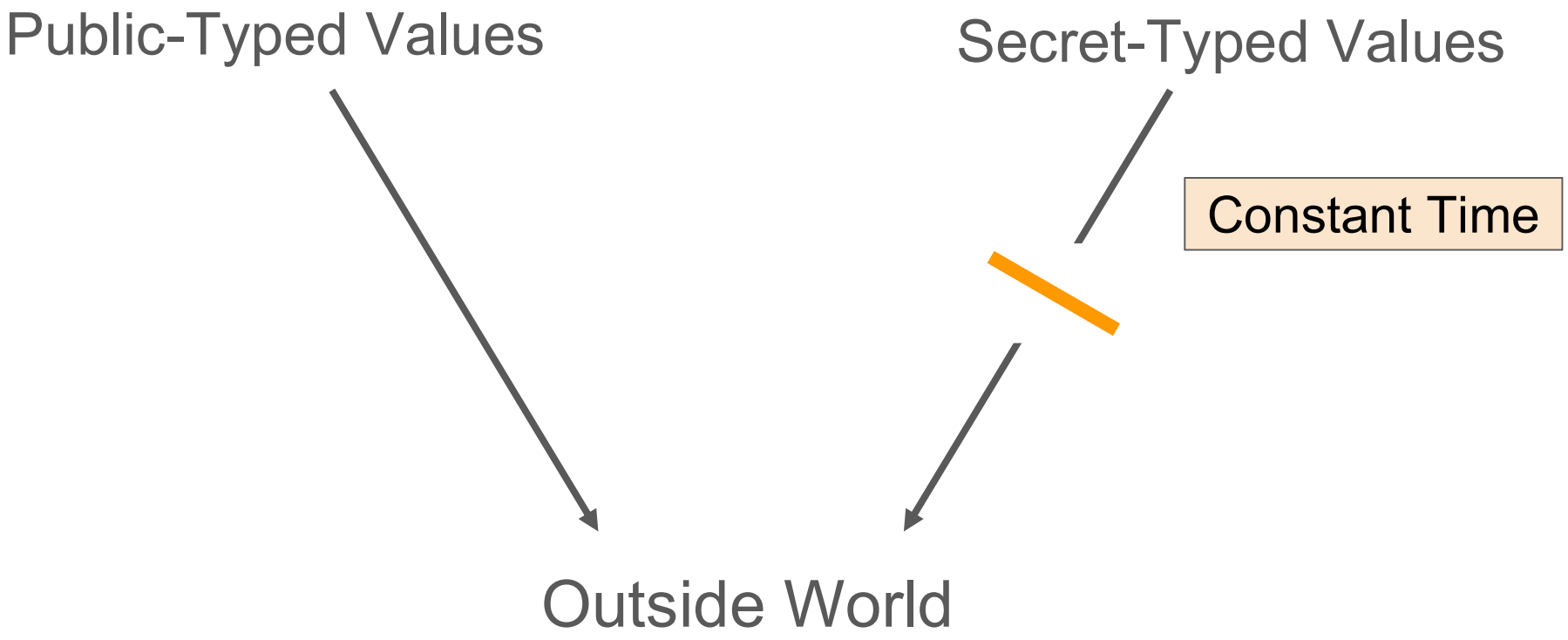
Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = state
```

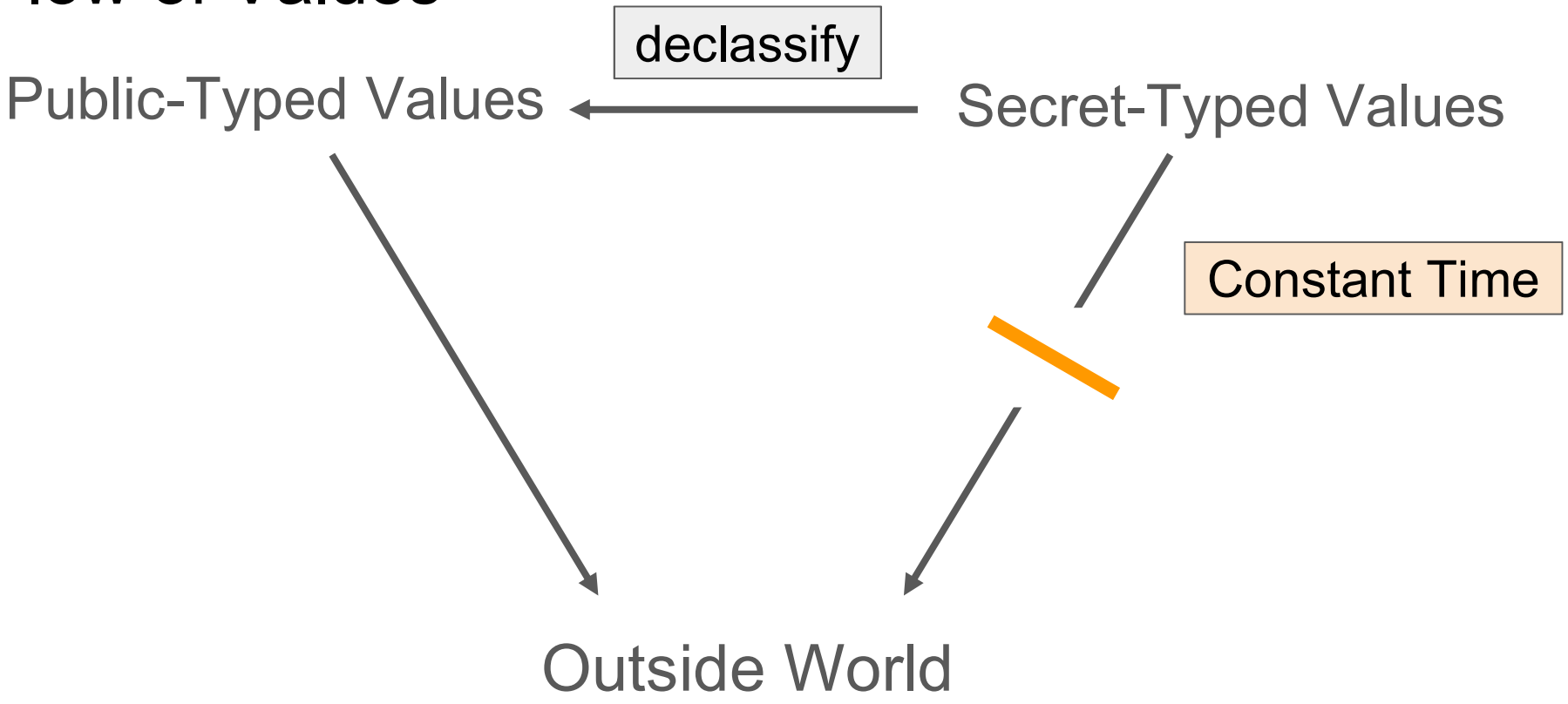
Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

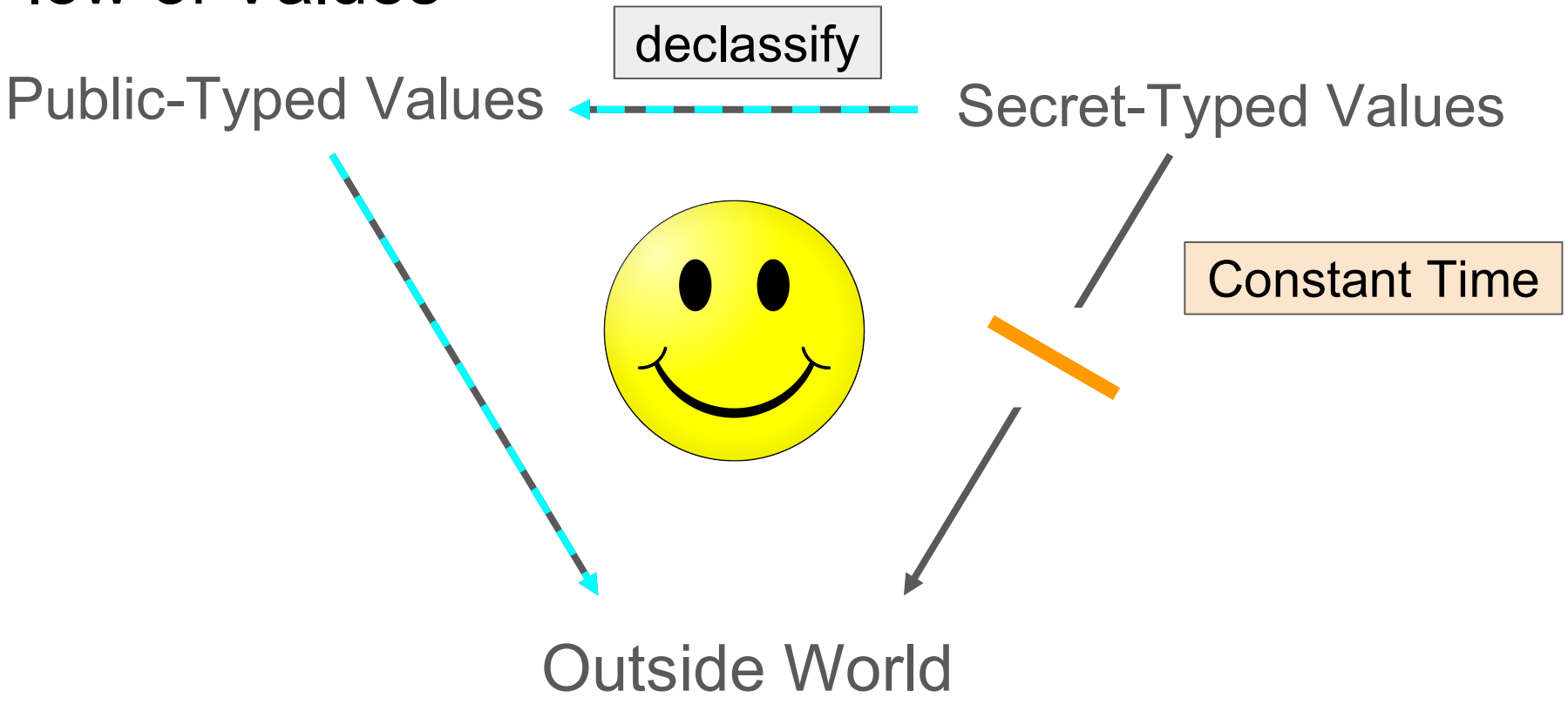
Flow of Values

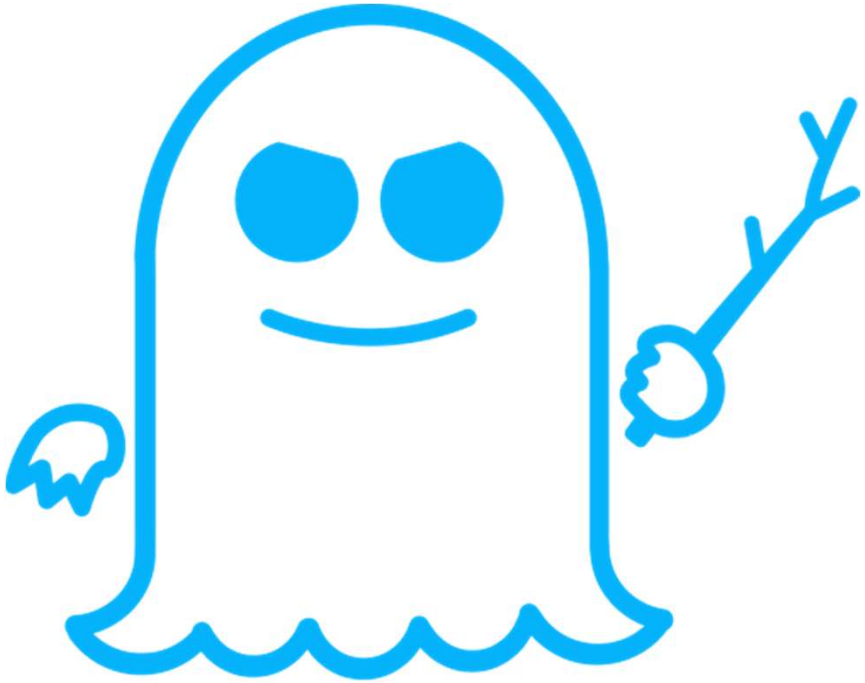


Flow of Values



Flow of Values





SPECTRE

Spectre Variant 1

```
if (index < array.length) {

    public value = array[index]
    leak(value)
}
```

Spectre Variant 1

Body executed even if condition is false

```
if (index < array.length) {

    public value = array[index]
    leak(value)
}
```

Spectre Variant 1

Body executed even if condition is false

```
if (index < array.length) {
    public value = array[index]
    leak(value)
}
```

Could load a secret

Spectre Variant 1

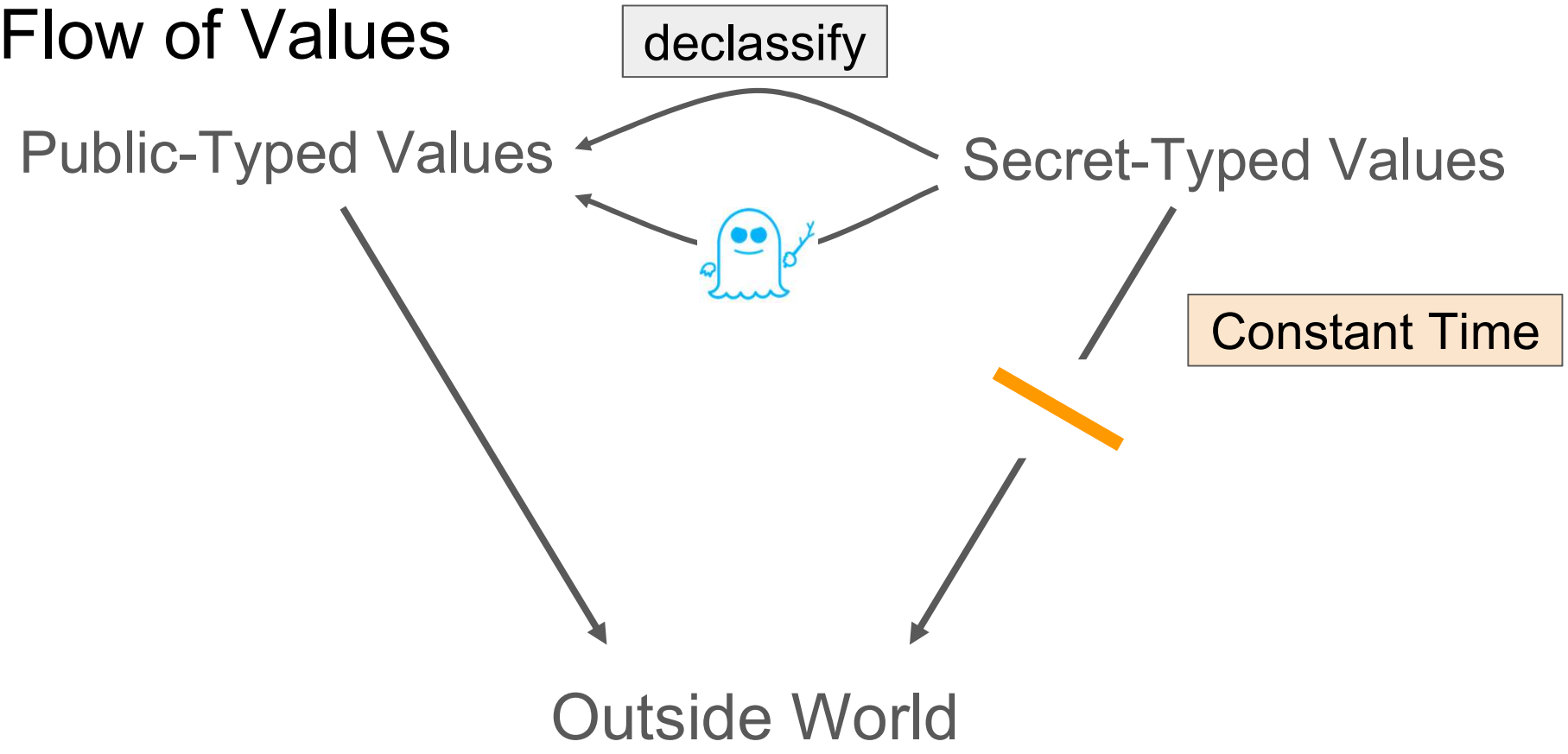
Body executed even if condition is false

```
if (index < array.length) {
    public value = array[index]
    leak(value)
}
```

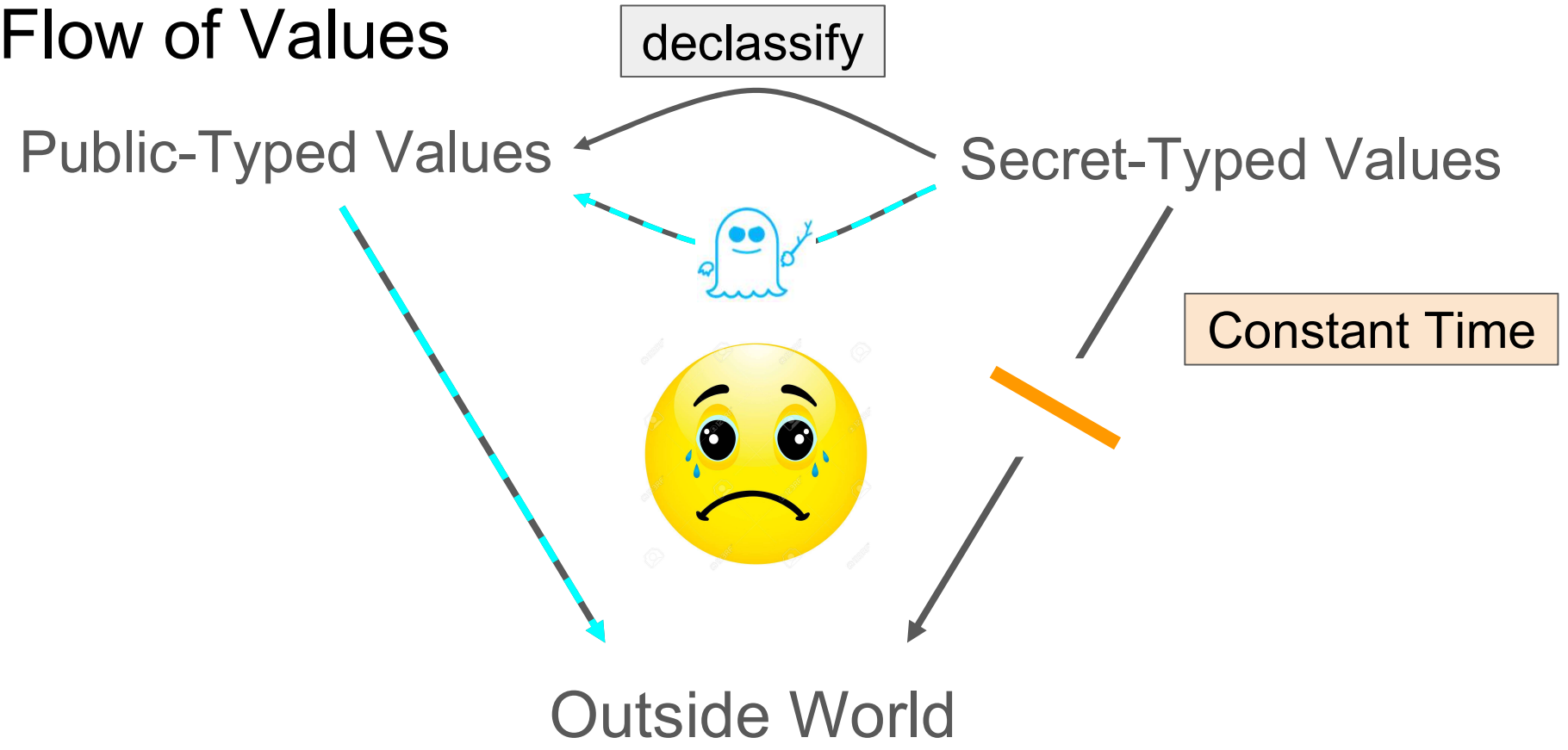
Can leak

Could load a secret

Flow of Values



Flow of Values



Speculative Load Hardening (SLH)

- Compiler instruments program
- Detects misspeculation
- Poisons loaded values under misspeculation

Speculative Load Hardening (SLH)

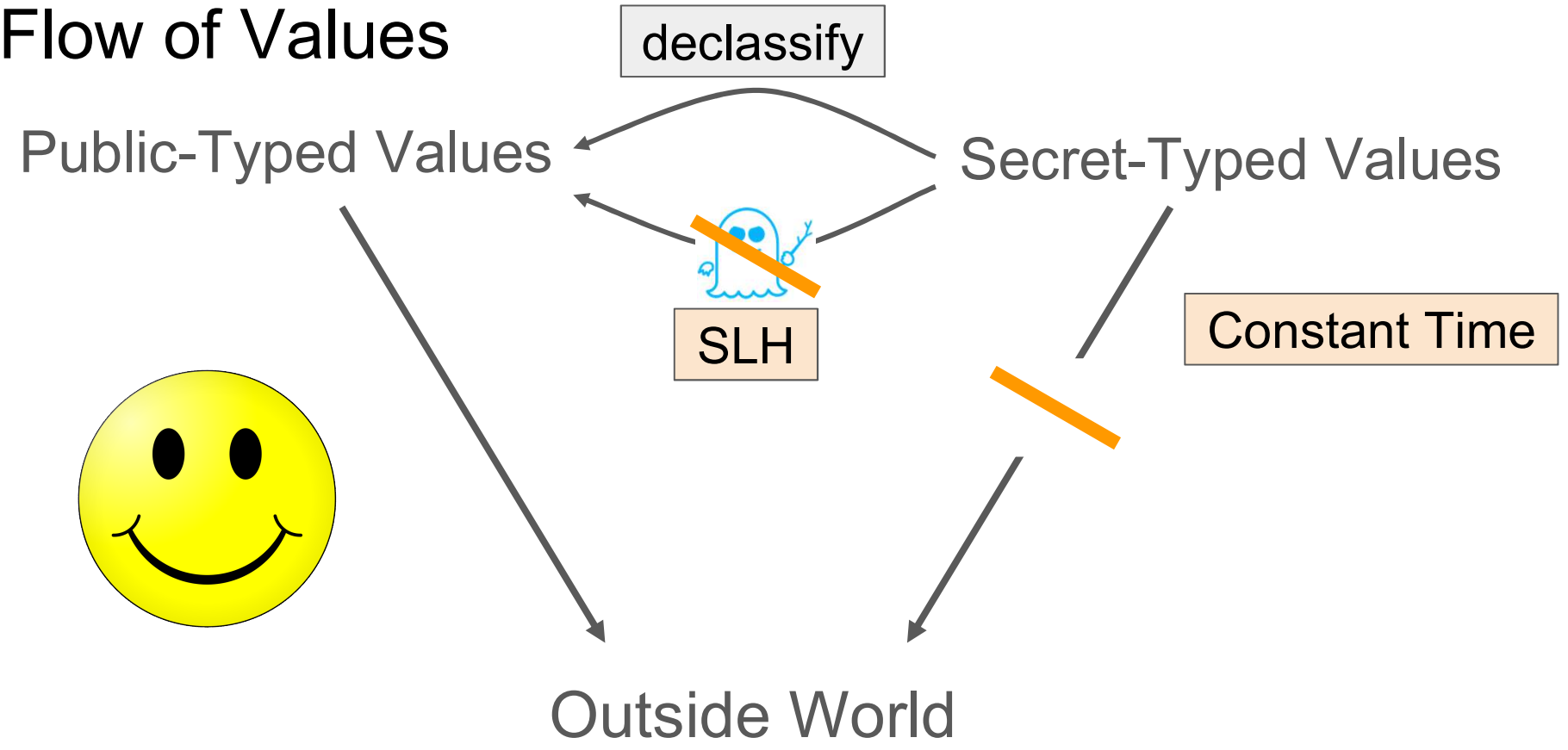
```
if (index < array.length) {

    public value = array[index]
    leak(value)
}
```

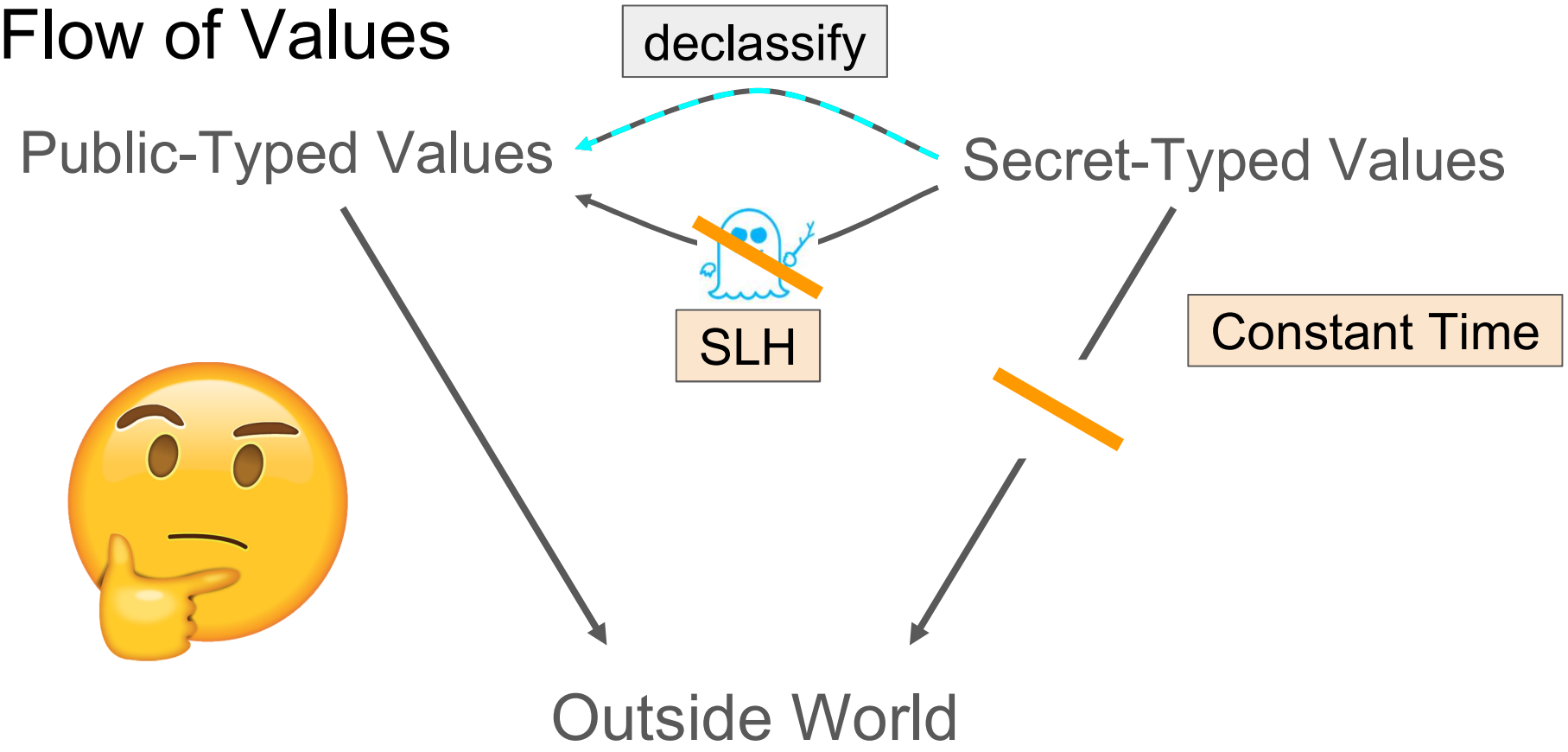
Speculative Load Hardening (SLH)

```
mask = 0
if (index < array.length) {
    mask = index < array.length ? mask : -1
    public value = array[index] | mask
    leak(value)
}
```

Flow of Values



Flow of Values



Speculative Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

Speculative Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

Skipped

Speculative Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

Skipped

Unencrypted

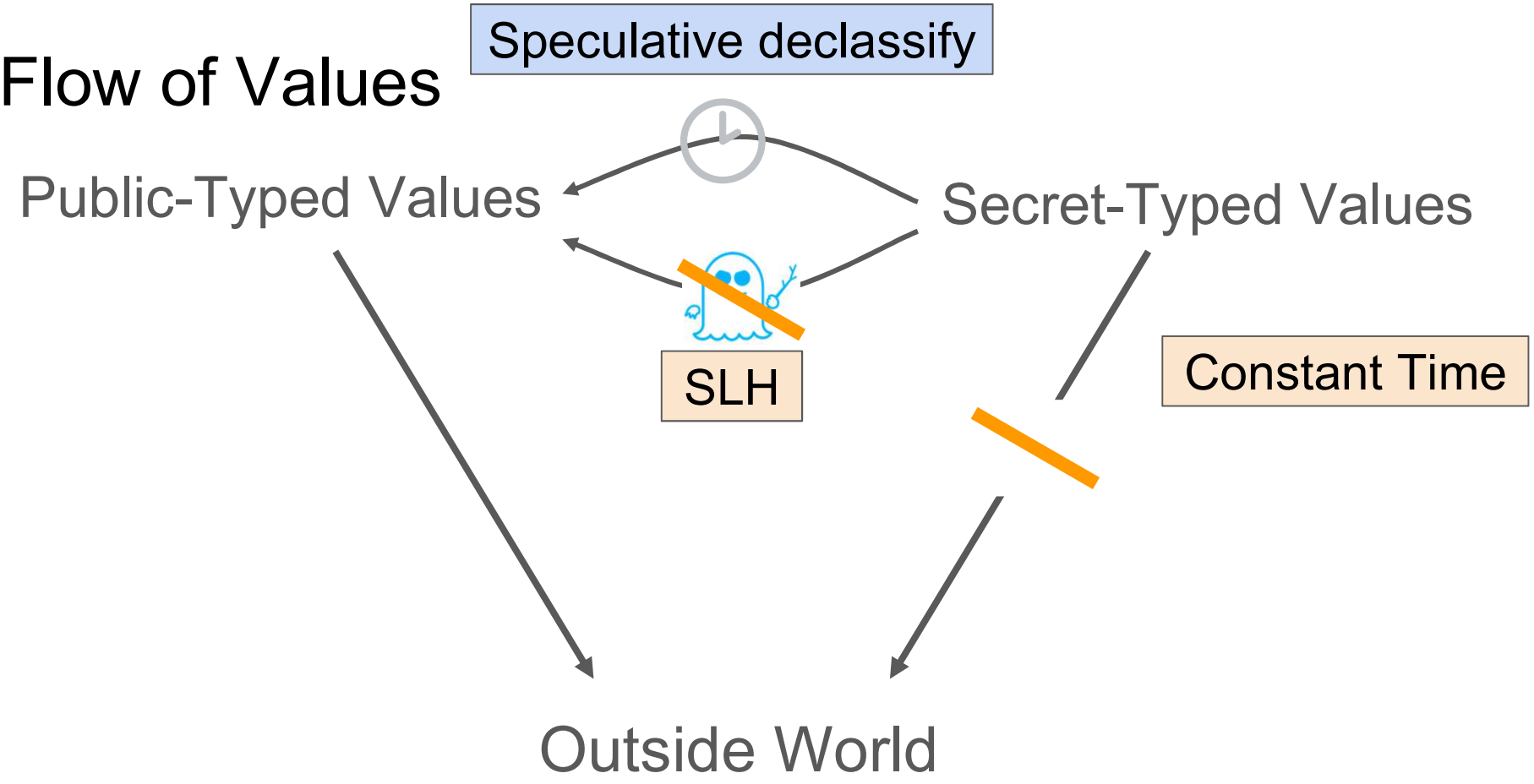
Speculative Declassification

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

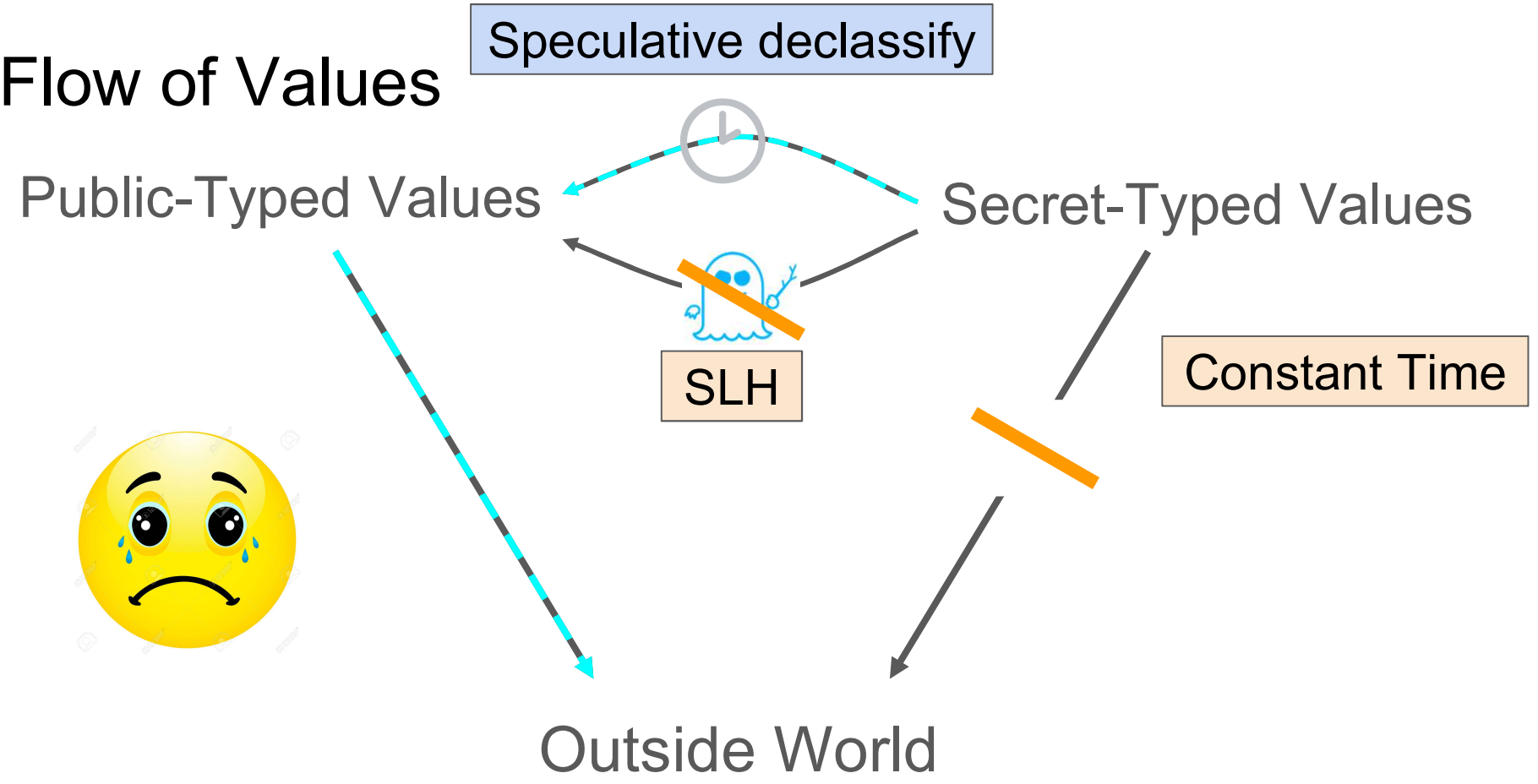
Skipped

Unencrypted

Flow of Values



Flow of Values



Real-World Example

```
secret state = plaintext
state = aes_round(state, *key++)
   ...
state = aes_round(state, *key++)
if (rounds == 12) {
    ...
}
state = aes_last_round(state, *key++)
public ciphertext = declassify(state)
```

Real-World Example

```
secret state = plaintext
state = aes_round(state, *key++)
   ...
state = aes_round(state, *key++)
if (rounds == 12) {
   ...
}
state = aes_last_round(state, *key++)
public ciphertext = declassify(state)
```

Skipped

Real-World Example

```
secret state = plaintext
state = aes_round(state, *key++)
   ...
state = aes_round(state, *key++)
if (rounds == 12) {
    ...
}
state = aes_last_round(state, *key++)
public ciphertext = declassify(state)
```

Skipped

Incorrectly encrypted

Real-World Example

```
secret state = plaintext
state = aes_round(state, *key++)
   ...
state = aes_round(state, *key++)
if (rounds == 12) {
   ...
}
state = aes_last_round(state, *key++)
public ciphertext = declassify(state)
```

Skipped

Incorrectly encrypted

Countermeasure

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

Countermeasure

```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

Insert speculation fence

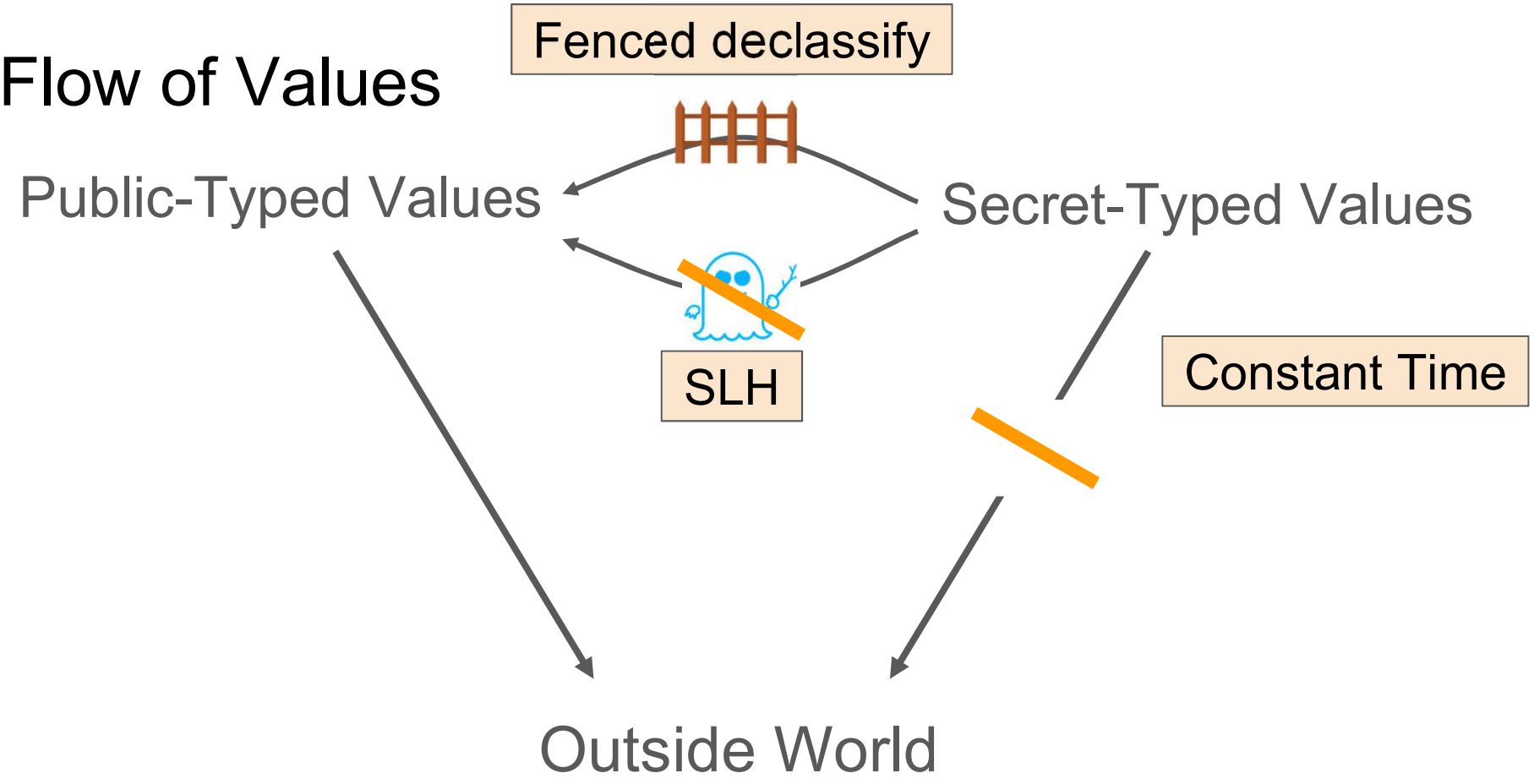
Countermeasure

Earlier instructions must execute correctly

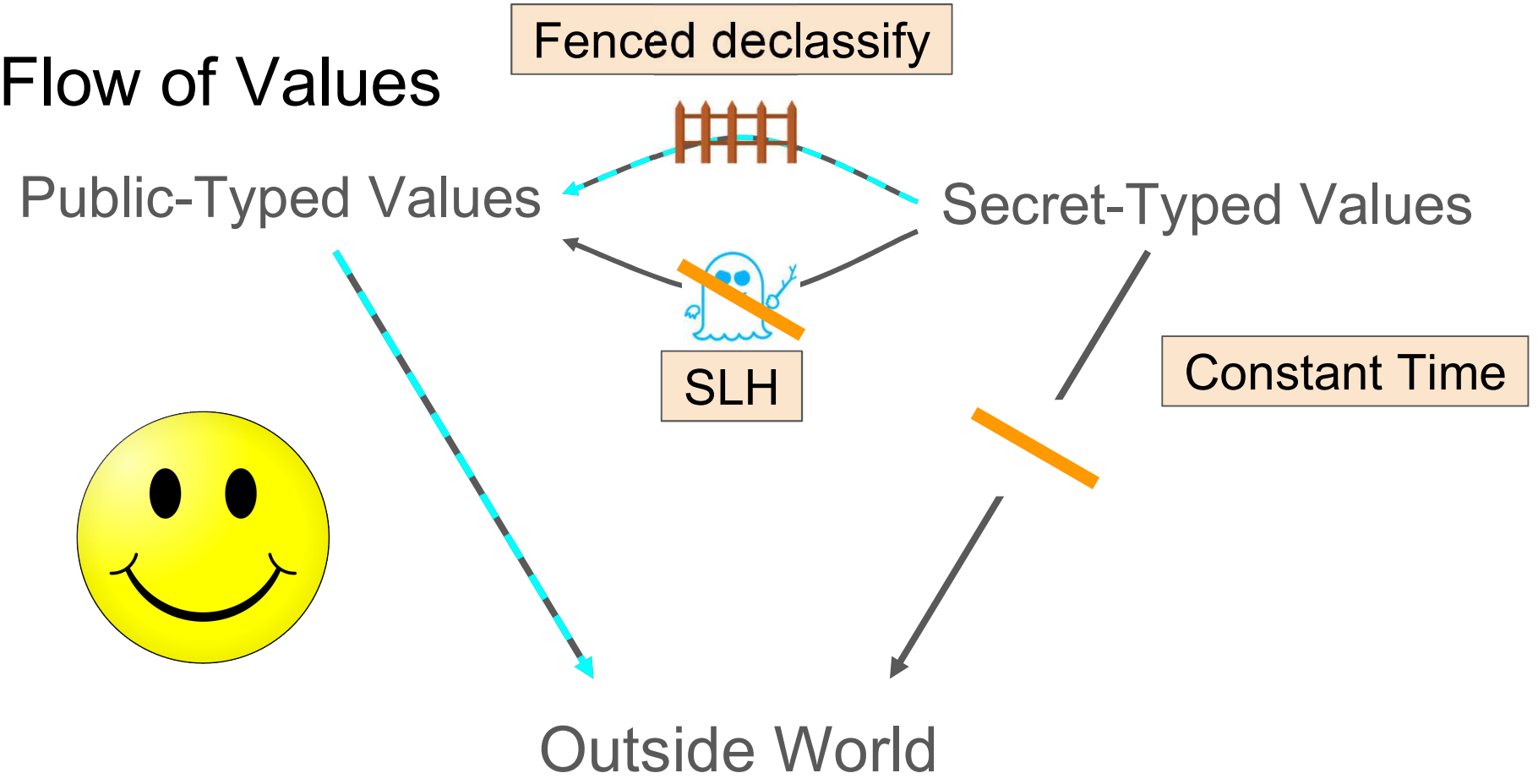
```
secret key = ...
secret state = message
for (i from 0 to 8) {
    public bit = 1 << i
    state ^= (key & bit)
}
public ciphertext = declassify(state)
```

Insert speculation fence

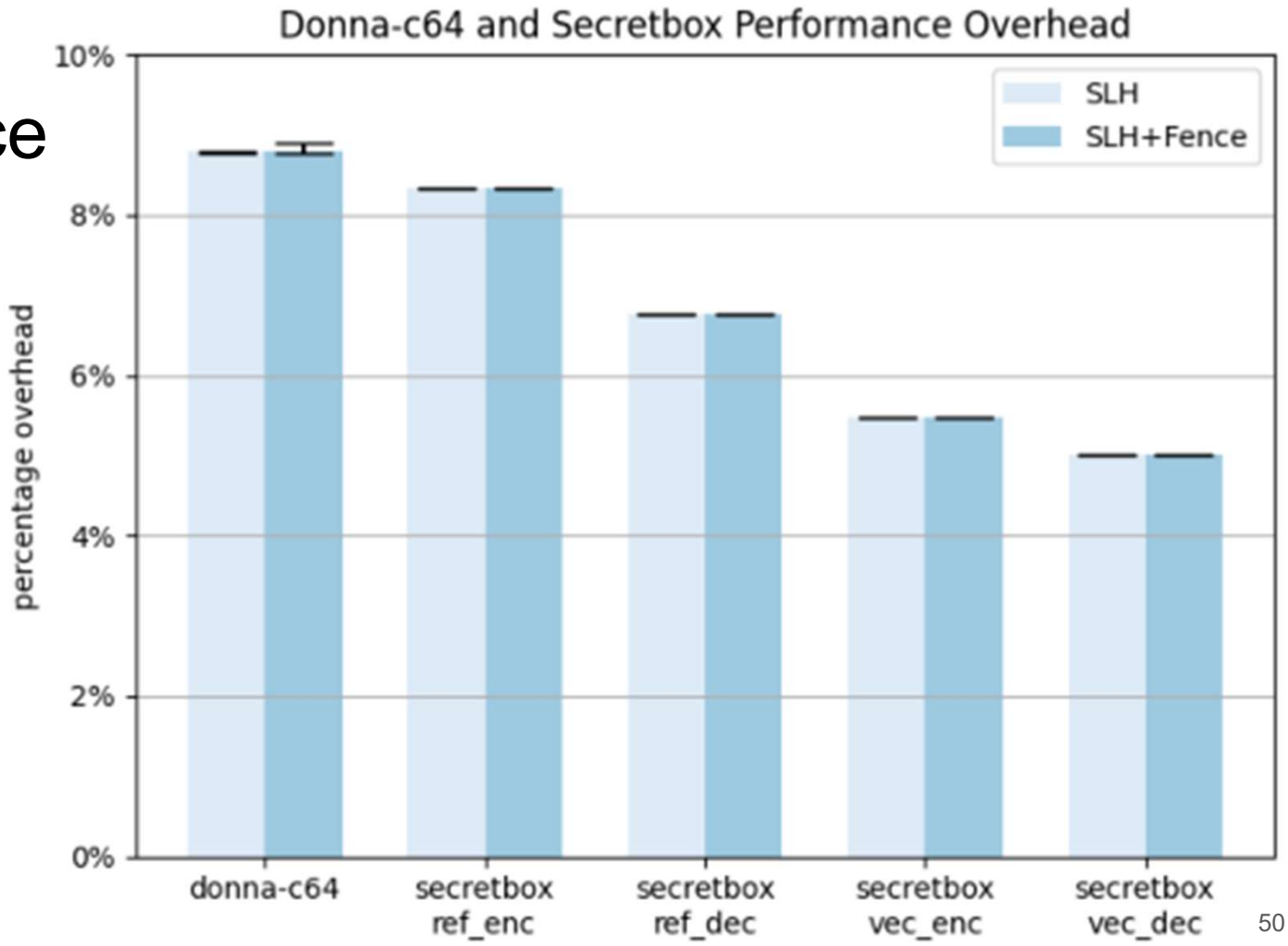
Flow of Values



Flow of Values



Performance Impact



Selective SLH

```
if (index < array.length) {

    ...
    public value = array[index]
    ...
    leak(value)
}
```

Selective SLH

```
mask = 0
if (index < array.length) {
    mask = index < array.length ? mask : -1
    ...
    public value = array[index] | mask
    ...
    leak(value)
}
```

Selective SLH

```
mask = 0
if (index < array.length) {
    mask = index < array.length ? mask : -1
    ...
    secret value = array[index] | mask
    ...
    leak(value)
}
```

Selective SLH

```
mask = 0
if (index < array.length) {
    mask = index < array.length ? mask : -1
    ...
    secret value = array[index] | mask
    ...
    leak(value)
}
```

Selective SLH

```
mask = 0
if (index < array.length) {
    mask = index < array.length ? mask : -1
    ...
    secret value = array[index] | mask
    ...
    leak(value)
}
```

Out of bounds access

Selective SLH

```
mask = 0
if (index < array.length) {
    mask = index < array.length ? mask : -1
    ...
    secret value = array[index] | mask
    ...
    leak(value)
}
```

Out of bounds access

Selective SLH

```
public array[]

if (index < array.length) {

    ...
    public value = array[index]
    ...
    leak(value)
}
```

Selective SLH

```
public array[]
mask = 0
if (index < array.length) {
    mask = index < array.length ? mask : -1
    ...
    public value = array[index] | mask
    ...
    leak(value)
}
```

What if (public) array never leaks?

Selective SLH

```
secret array[]

if (index < array.length) {

    ...
    secret value = array[index]
    ...
    leak(value)
}
```

Tagging public values that do not leak as secret **reduces overhead!!!**

Selective SLH

Algorithm	**SLH Masking Removed**
ChaCha20 (1024B)	~80%
Donna-c64	~95%

Summary

- Declassification requires protections
- SLH + fence has negligible performance impact
- Selective SLH significantly reduces overhead

- See the paper for more details: https://eprint.iacr.org/2022/426