Protecting online communication against eavesdropping at the program level

Defence of PhD thesis “Information Flow Techniques for Mitigating Traffic Analysis”

Jeppe Fredsgaard Blaabjerg

Advisor: Aslan Askarov

Aarhus University

AARHUS UNIVERSITET

Information hiding

Credit bicycling.com

Thesis

Two research topics

1. Mitigating traffic-analysis at the program level
   a. Towards Language-Based Mitigation of Traffic Analysis Attacks
      In Proceedings of the IEEE 34th Computer Security Foundations Symposium (CSF), 2021
   b. OblivIO: Securing reactive programs by oblivious execution with bounded traffic overheads
      In Proceedings of the IEEE 36th Computer Security Foundations Symposium (CSF), 2023
2. Precision of dynamic information-flow control
   a. On precision of dynamic fine-grained information-flow control

Thesis

Two research topics

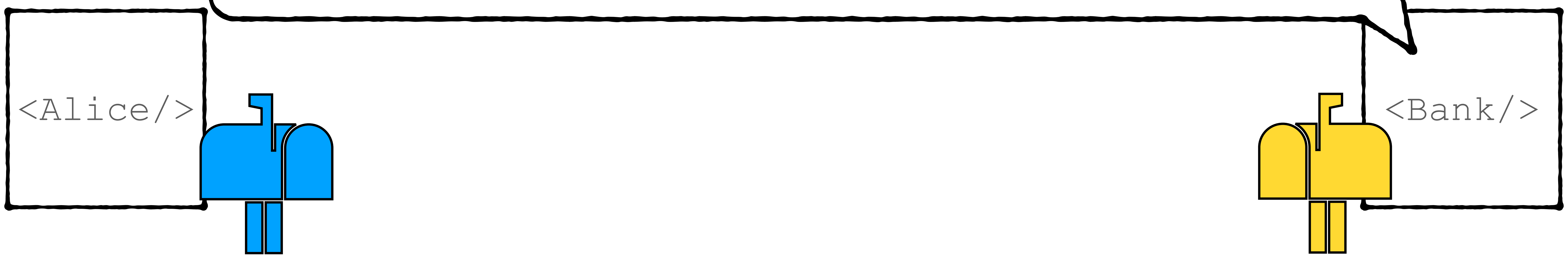
1. Mitigating traffic-analysis at the program level
   a. Towards Language-Based Mitigation of Traffic Analysis Attacks
      In Proceedings of the IEEE 34th Computer Security Foundations Symposium (CSF), 2021
   b. OblivIO: Securing reactive programs by oblivious execution with bounded traffic overheads
      In Proceedings of the IEEE 36th Computer Security Foundations Symposium (CSF), 2023
2. Precision of dynamic information-flow control
   a. On precision of dynamic fine-grained information-flow control

In this talk

Traffic analysis

Example

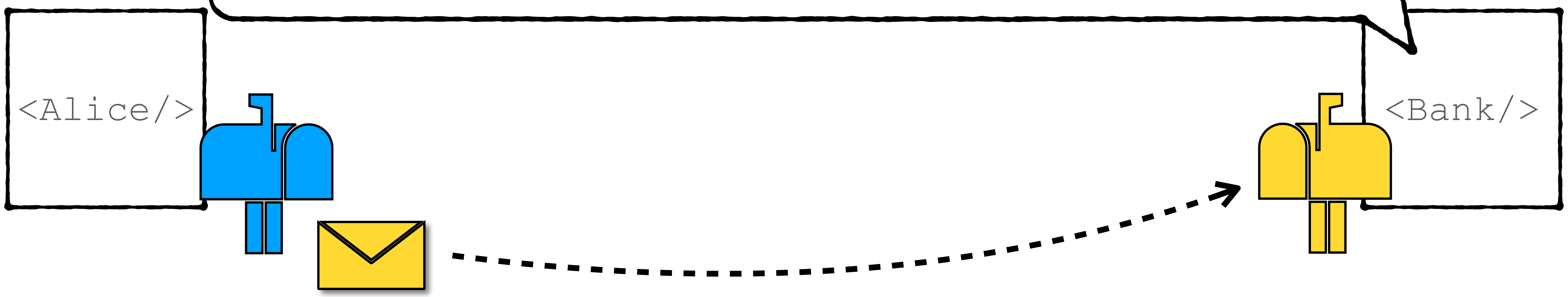
```
TRANSFER(from: int, amount: int, to: int) {
  if amount <= balance[from]
  then {
    balance[from] = balance[from] - amount;
    balance[to]   = balance[to]   + amount;
  }
  else send(ALICE, "ERROR!");
}
```



Traffic analysis

Example

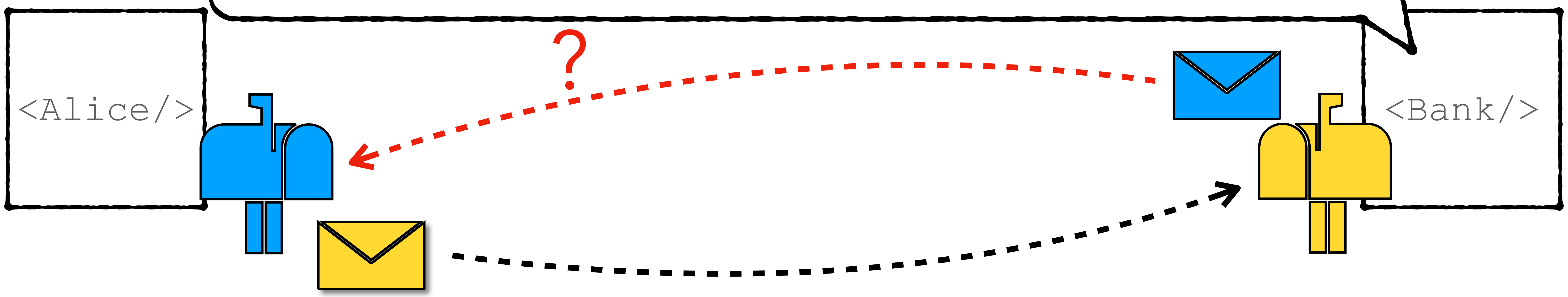
```
TRANSFER(from: int, amount: int, to: int) {
  if amount <= balance[from]
  then {
    balance[from] = balance[from] - amount;
    balance[to]   = balance[to]   + amount;
  }
  else send(ALICE, “ERROR!”);
}
```



Traffic analysis

Example

```
TRANSFER(from: int, amount: int, to: int) {
  if amount <= balance[from]
  then {
    balance[from] = balance[from] - amount;
    balance[to]   = balance[to]   + amount;
  }
  else send(ALICE, “ERROR!”);
}
```



Traffic analysis

Other observable properties of online communication

Traffic analysis

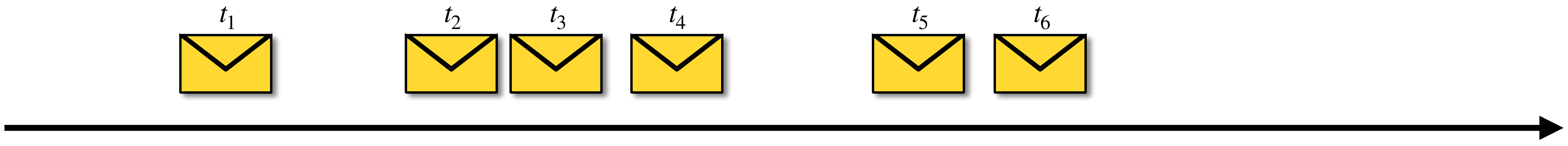
Other observable properties of online communication

- Message timing

Traffic analysis

Other observable properties of online communication

- Message timing

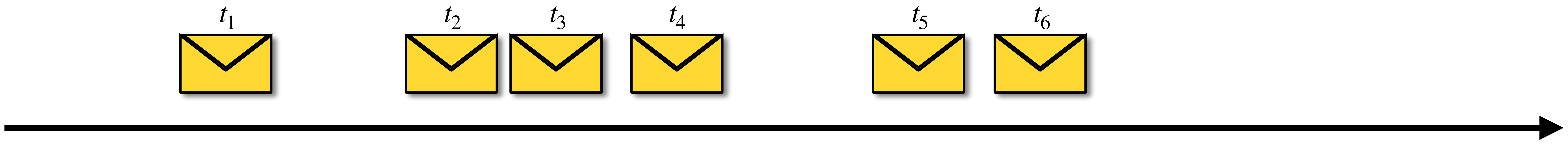


- Message size

Traffic analysis

Other observable properties of online communication

- Message timing

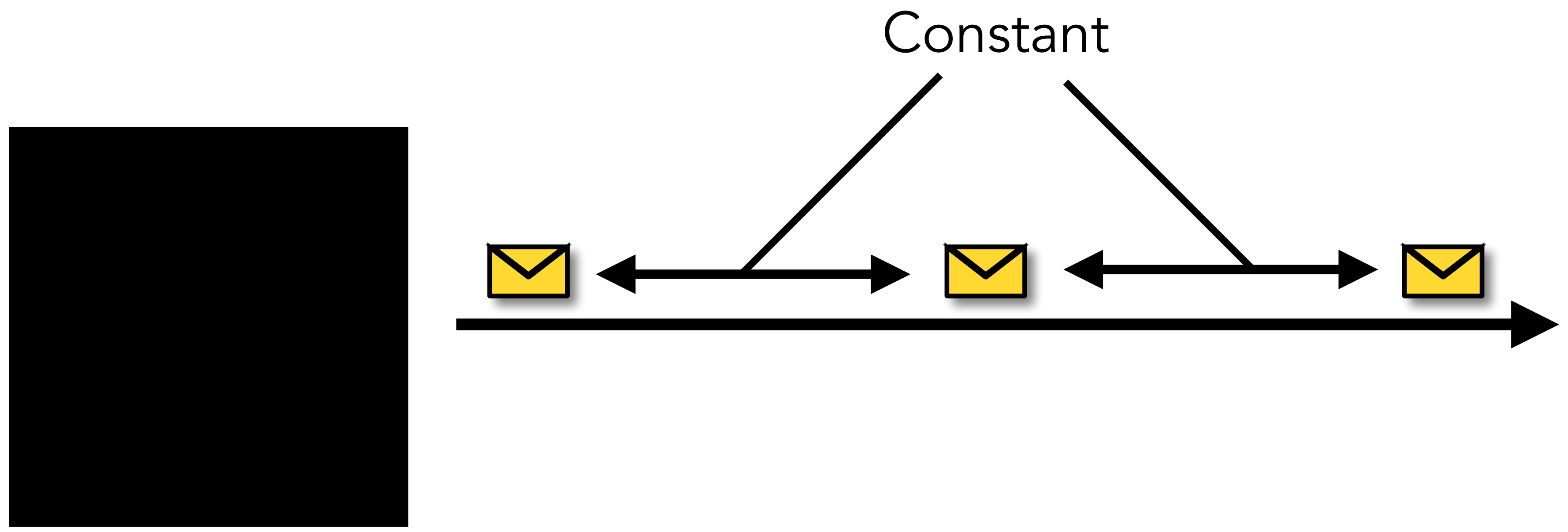


- Message size

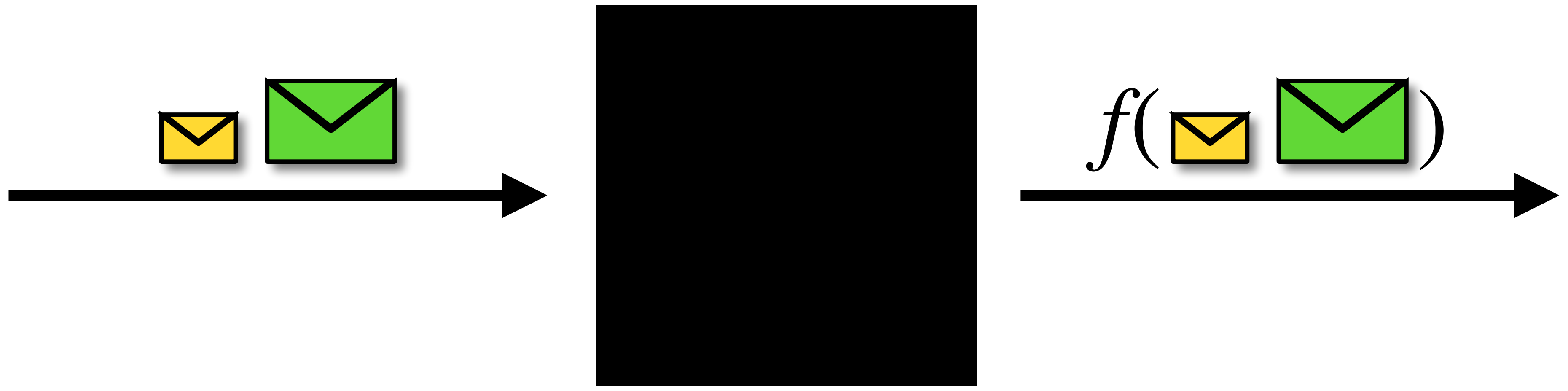
- Message recipient

Mitigating traffic analysis

Existing approaches: System-level mitigation



Independent link padding



Dependent link padding

- Treat program as black-box
- Two main approaches
  - Independent-link padding: Commonly, constant rate of fixed-size packets
  - Dependent-link padding: Shape of outgoing traffic computed from the shape of incoming traffic
- Prohibitive overheads in practice: traffic or latency[1]

[1] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in 2012 IEEE symposium on security and privacy. IEEE, 2012, pp. 332–346
D. Das, S. Meiser, E. Mohammadi, and A. Kate, "Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two," IACR Cryptology ePrint Archive, vol. 2017, p. 954, 2017.

Example

What is the right system-level bandwidth?

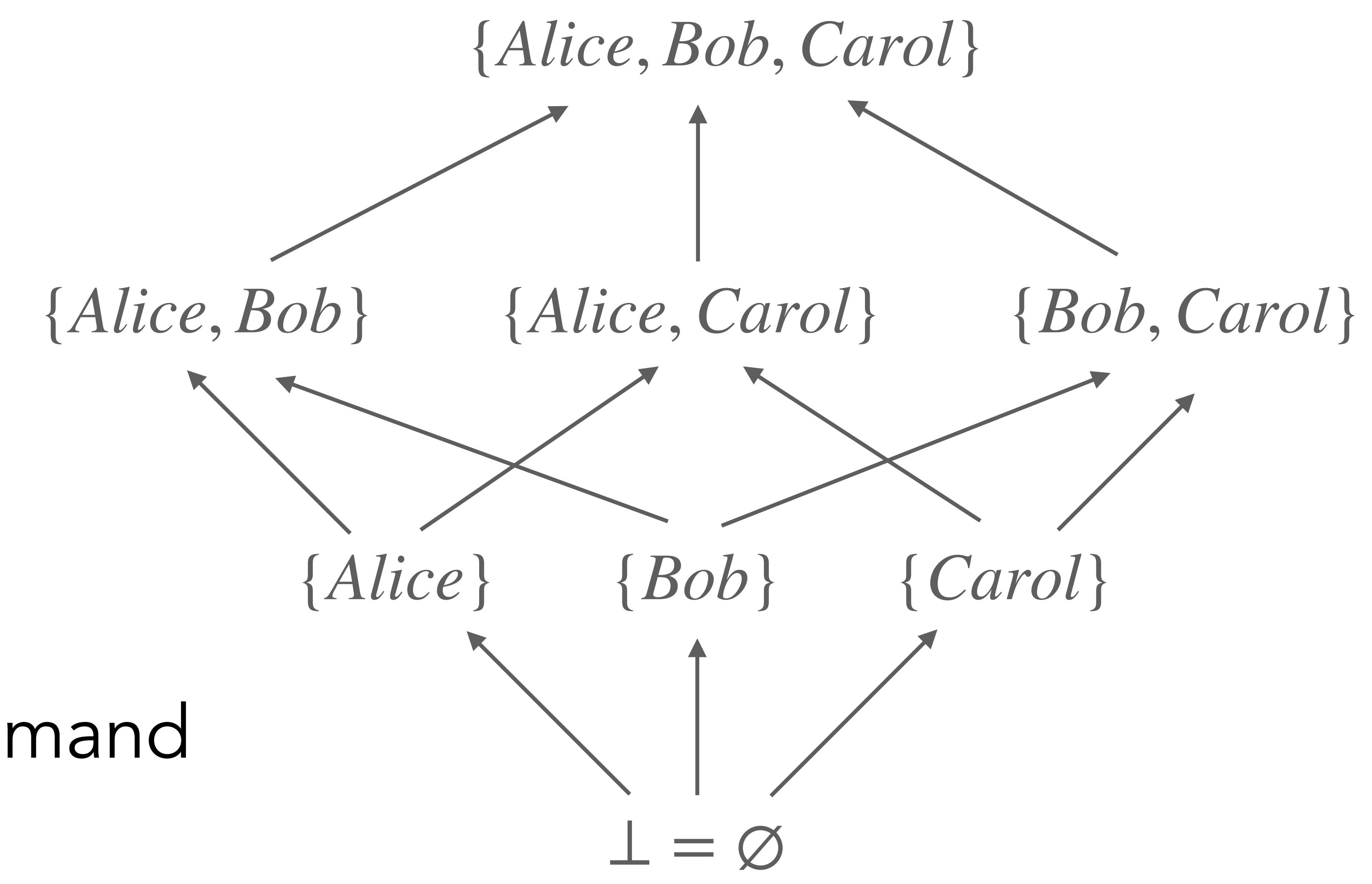
```
RELAY(x: int) {
  if cnd
  then send(FORWARD, x);
  else skip;
}
```

- Traffic padding only needed if `cnd` is secret
  - Not known at the system level
- Idea in my work: Use language-level techniques for mitigating traffic analysis
  - How: Information-flow control

Information-flow control

Background

- Label data with security levels ℓ drawn from a lattice
  - Distinguished *least* level $\bot$ *(public)*
  - Flows-to relation $\ell_1 \sqsubseteq \ell_2$
    - ℓ_2 may *learn* data at level ℓ_1
  - Join operation $\ell_1 \sqcup \ell_2 = \ell_3$
    - ℓ_3 is the *least* level that both ℓ_1 and ℓ_2 may flow to
- pc-label to track the sensitivity of executing a particular command



Powerset lattice

OblivIO

Securing reactive programs by oblivious execution with bounded traffic overheads

In Proceedings of the IEEE 36th Computer Security Foundations Symposium (CSF), 2023.

Mitigating traffic analysis

What must be protected?

<cfg$_1$/>

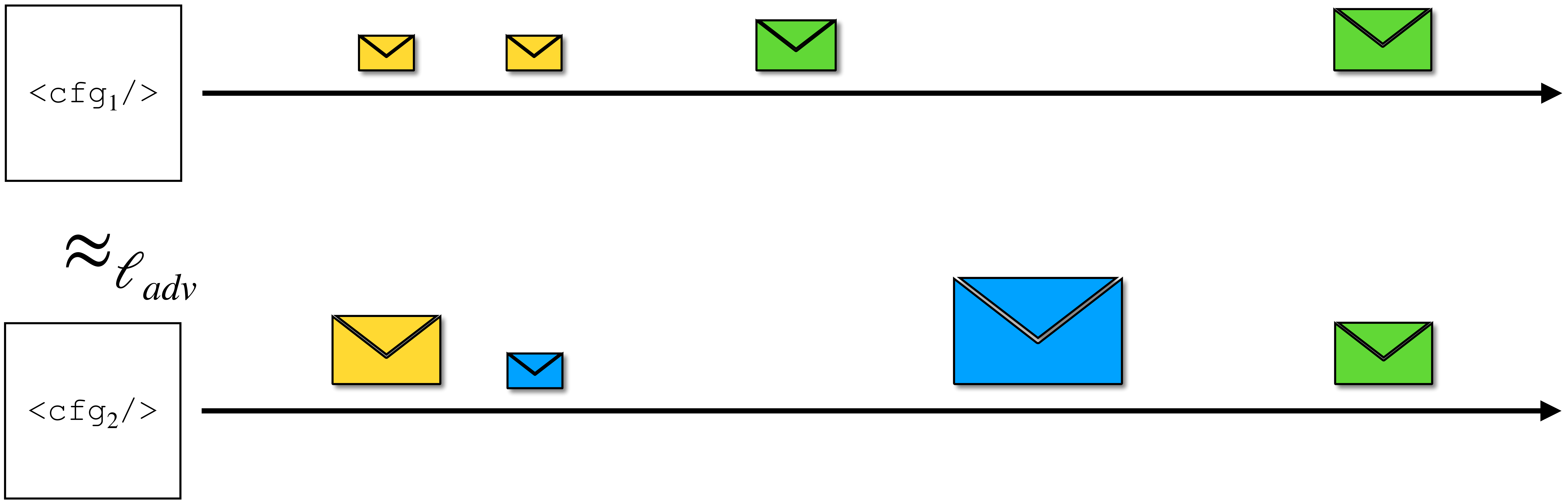
$$\approx_{\ell_{adv}}$$

<cfg$_2$/>

- All network nodes run OblivIO
- Attacker may be network level only or may be another node

Mitigating traffic analysis

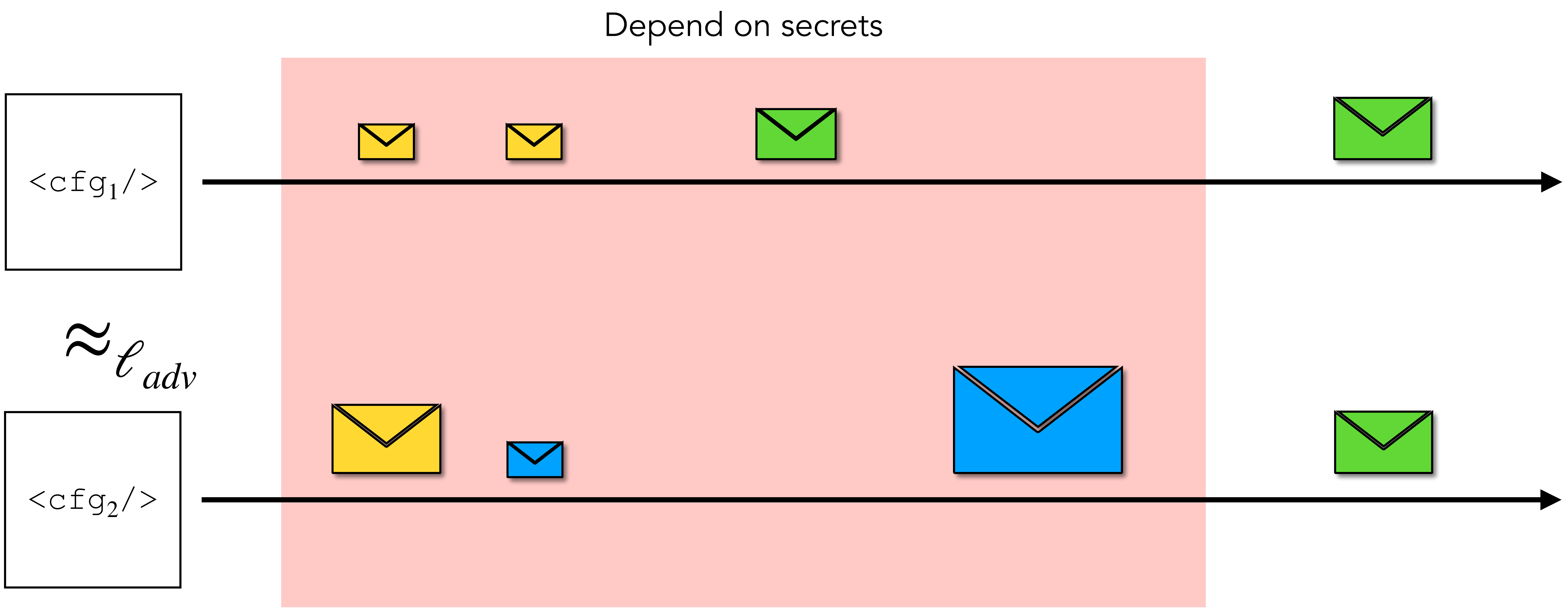
What must be protected?



- All network nodes run OblivIO
- Attacker may be network level only or may be another node

Mitigating traffic analysis

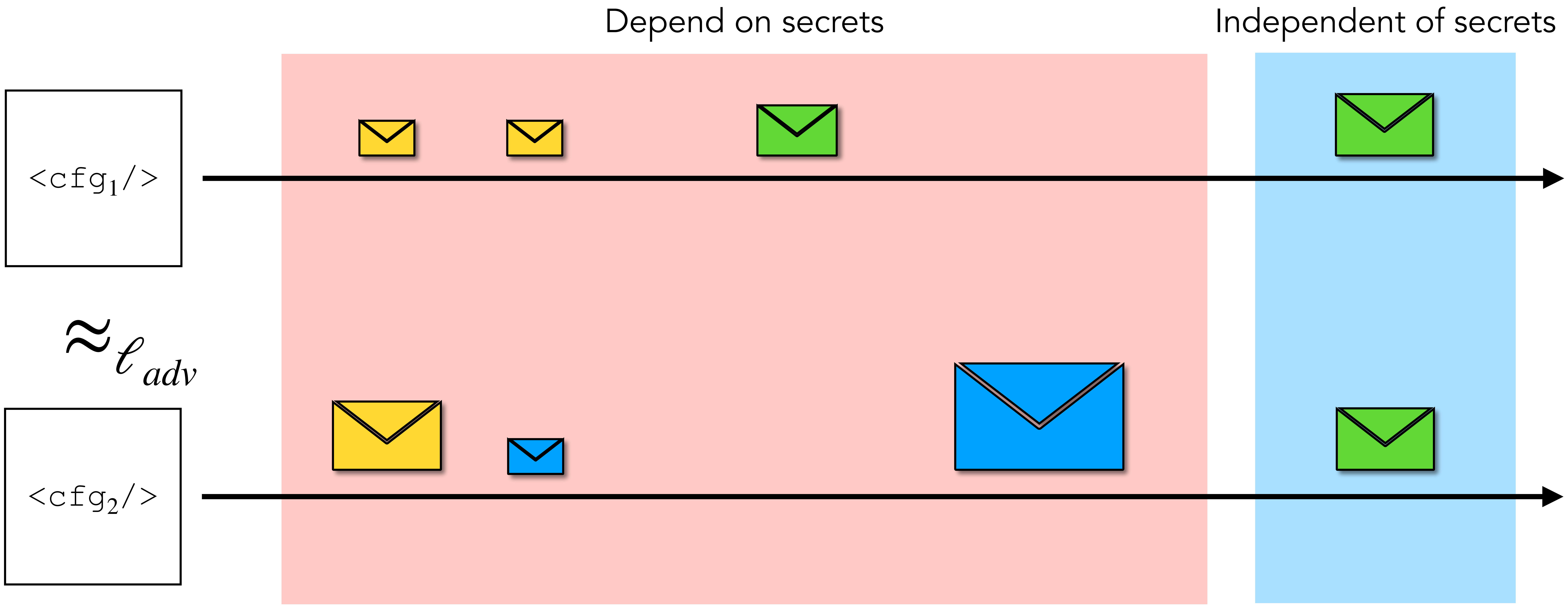
What must be protected?



- All network nodes run OblivIO
- Attacker may be network level only or may be another node

Mitigating traffic analysis

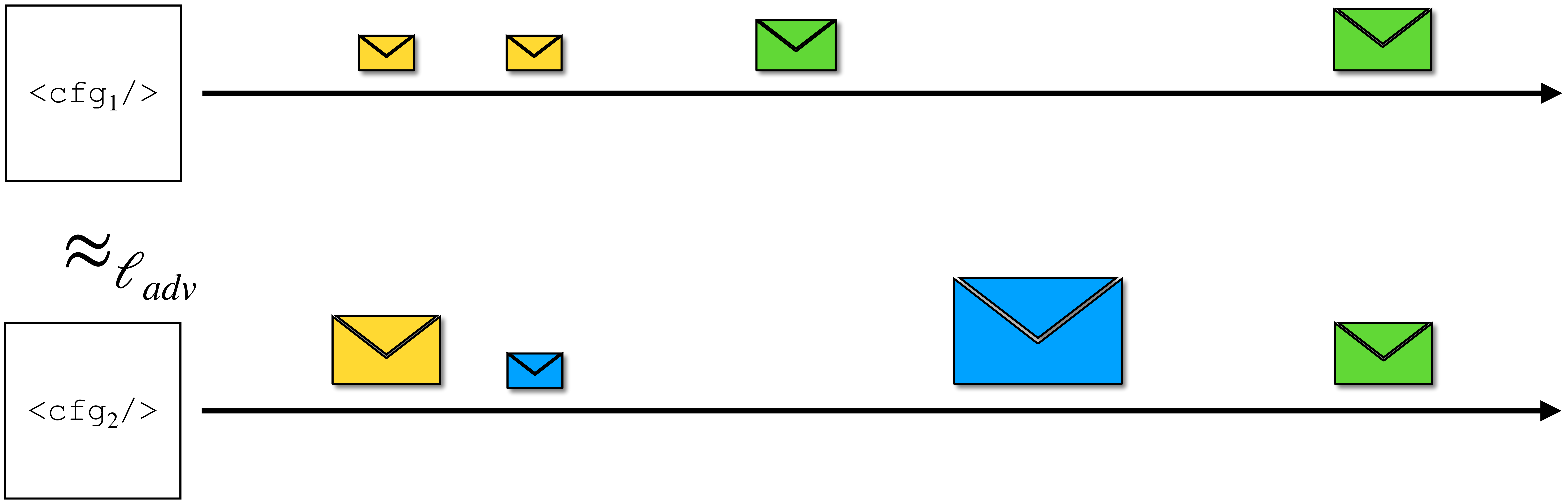
What must be protected?



- All network nodes run OblivIO
- Attacker may be network level only or may be another node

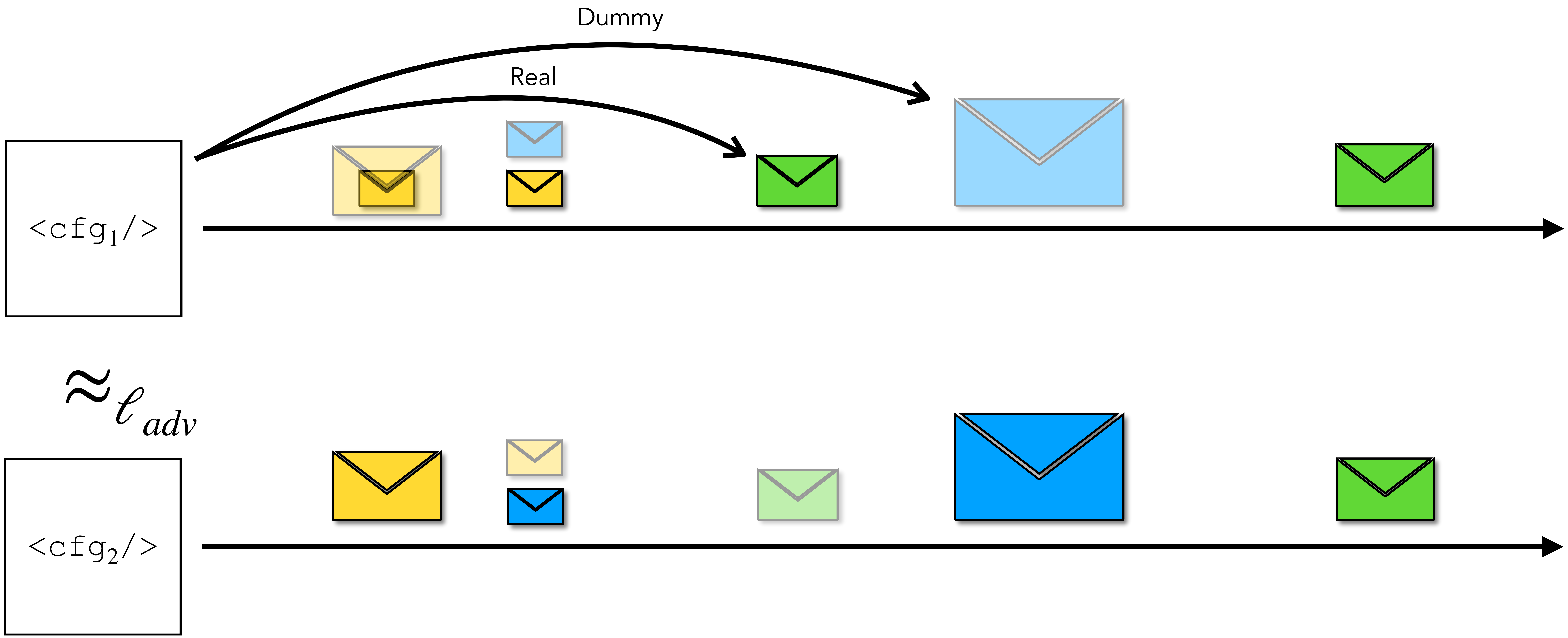
Mitigating traffic analysis

OblivIO: Traffic padding guided by program source



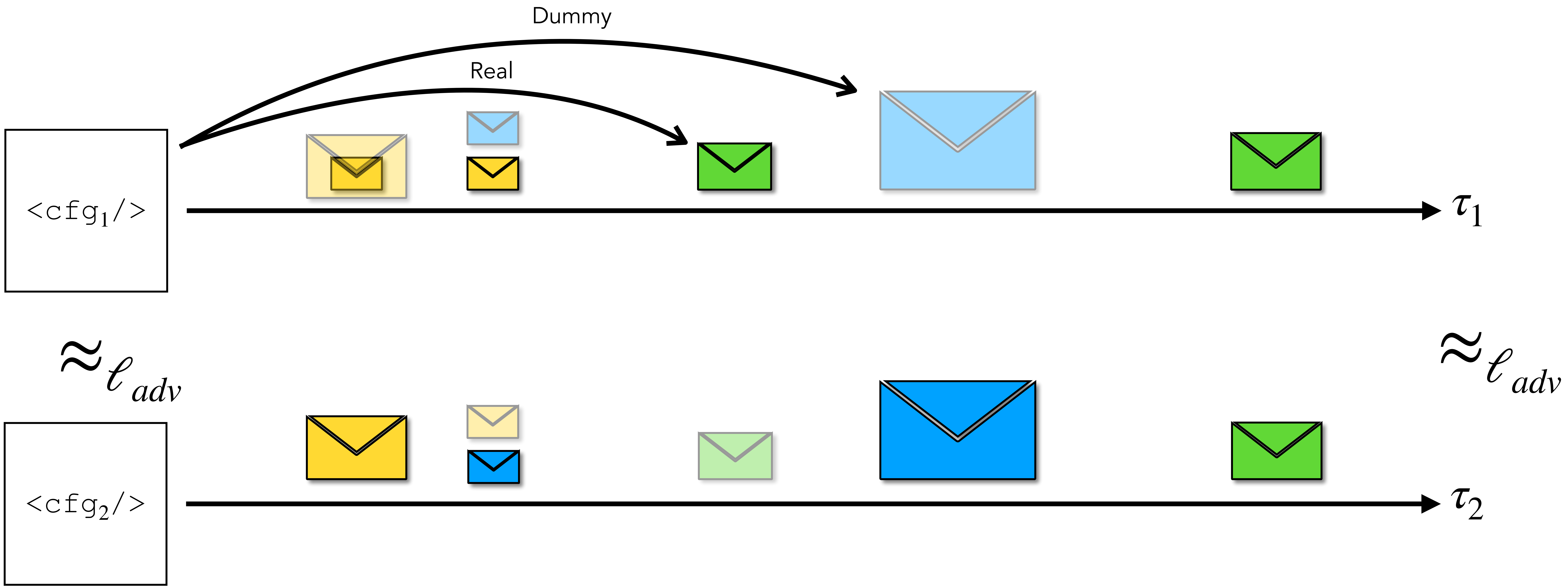
Mitigating traffic analysis

OblivIO: Traffic padding guided by program source



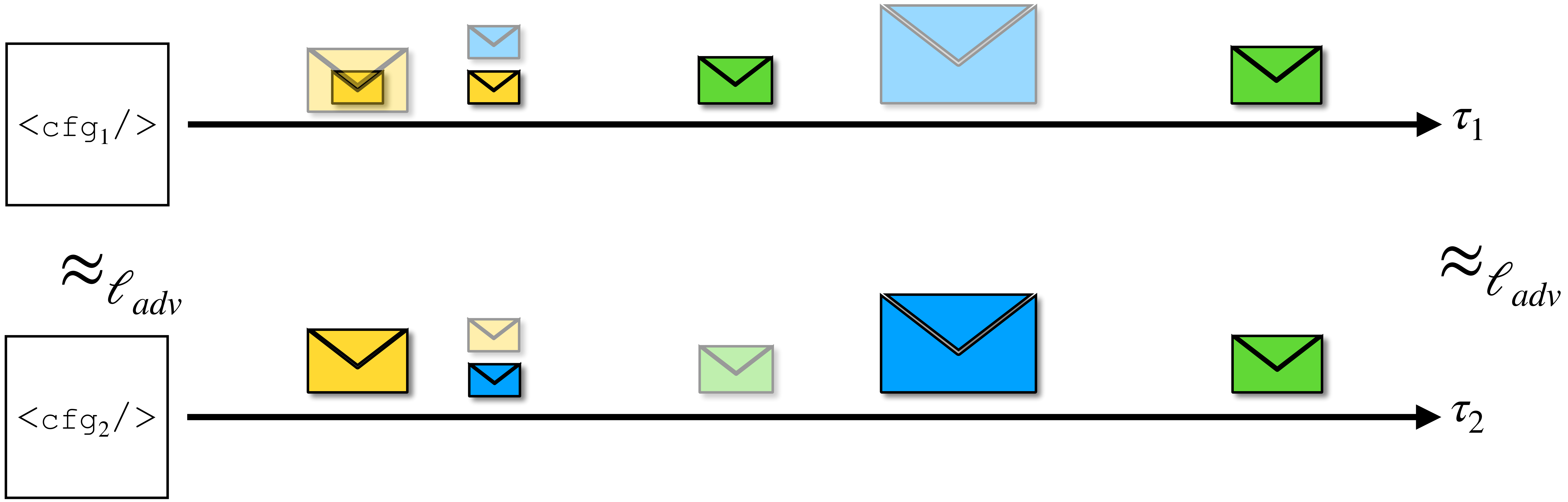
Mitigating traffic analysis

OblivIO: Traffic padding guided by program source



Mitigating traffic analysis

OblivIO: Traffic padding guided by program source



$$k(cfg_1, \tau_1, \ell_{adv}) \triangleq \{cfg_2 \mid cfg_1 \approx_{\ell_{adv}} cfg_2 \land cfg_2 \longrightarrow^*_{\tau_2} cfg_2' \land \tau_1 \approx_{\ell_{adv}} \tau_2\}$$

Attacker knowledge[2]

$$k(cfg_1, \tau_1 \cdot \alpha_1, \ell_{adv}) \supseteq k(cfg_1, \tau_1, \ell_{adv})$$

Progress-sensitive noninterference (PSNI)

[2] Askarov and A. Sabelfeld, "Gradual release: Unifying declassification, encryption and key release policies," 2007 IEEE Symposium on Security and Privacy.

OblivIO

Language and syntax

- Simple imperative language for reactive programs
- Two execution modes: *real* and *phantom*
  - Data-obliviousness[3] — control-flow is never secret
- Formal model includes computational history for computing timestamp[4]

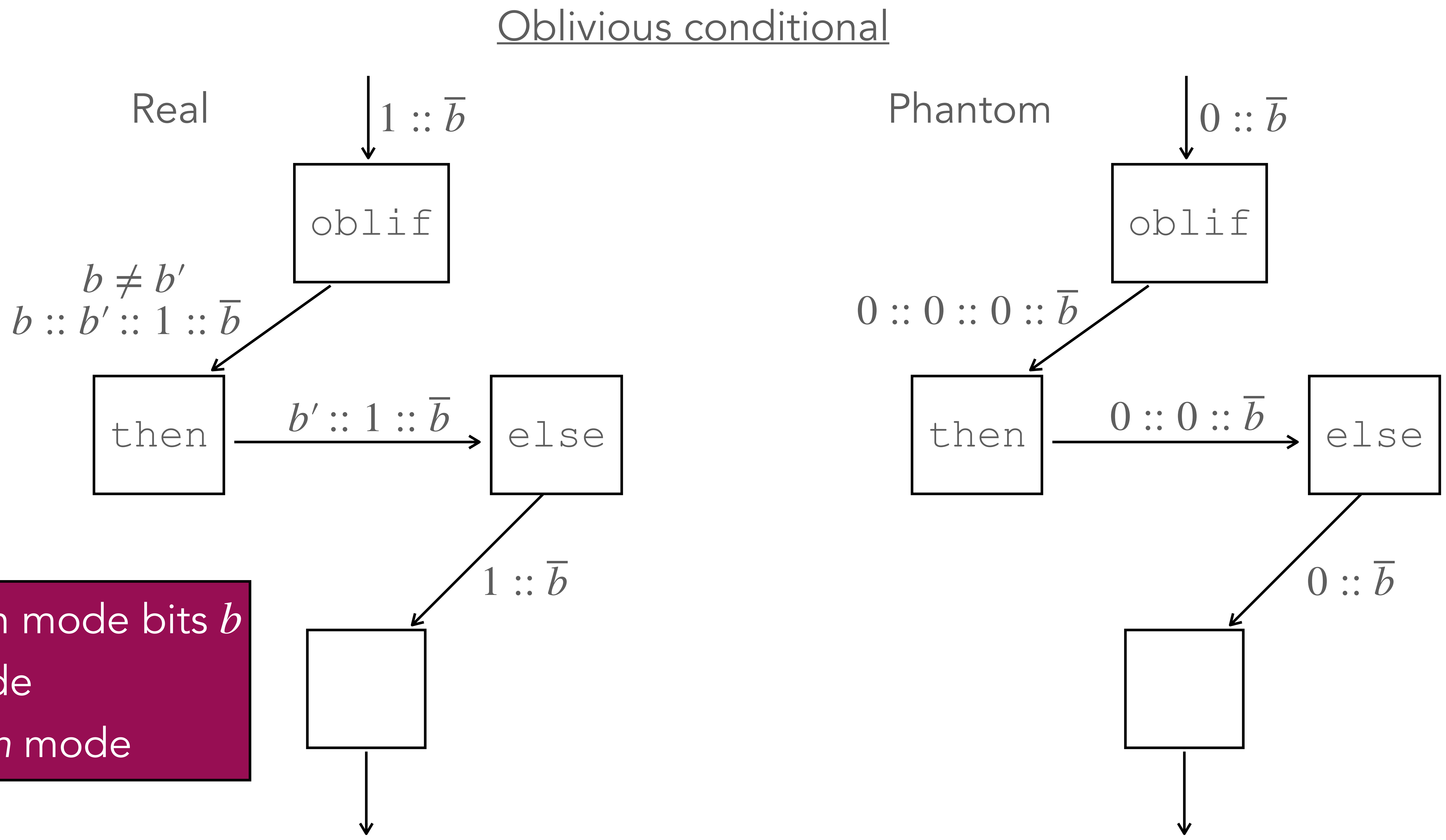
$$
\begin{aligned}
p &::= \cdot \mid ch(x)\{c\};p \\
c &::= \texttt{skip} \mid c_1;c_2 \mid x = e \mid \texttt{if}\ e\ \texttt{then}\ c\ \texttt{else}\ c \mid \texttt{while}\ e\ \texttt{do}\ c \mid \texttt{send}(ch,e) \\
&\quad \mid \texttt{oblif}\ e\ \texttt{then}\ c\ \texttt{else}\ c \quad \text{(* Oblivious conditional — executes both branches *)} \\
&\quad \mid x \mathrel{?=} e \quad \text{(* Oblivious, padding assignment *)} \\
&\quad \mid x \mathrel{?=} \texttt{input}(ch,e) \quad \text{(* Local input *)}
\end{aligned}
$$

[3] S. Zahur and D. Evans, "Obliv-c: A language for extensible data-oblivious computation," IACR Cryptol. ePrint Arch., p. 1153, 2015. [Online]. Available: http://eprint.iacr.org/2015/1153

[4] Daniel Hedin and David Sands. Timing aware information flow security for a javacard-like bytecode. *Electronic Notes in Theoretical Computer Science*, 141 (1):163–182, 2005.

Oblivious semantics

Control flow



$\overline{b}$ is a stack of execution mode bits b

$b = 1$ denotes *real* mode

$b = 0$ denotes *phantom* mode

Oblivious semantics

Assignment

Oblivious assignment

Real

$1 :: \overline{b}, x \mapsto (\text{"Goodbye"})_7$

`x ?= “Hello”;`

$1 :: \overline{b}, x \mapsto (\text{"Hello"})_7$

Phantom

$0 :: \overline{b}, x \mapsto (\text{"Hello"})_5$

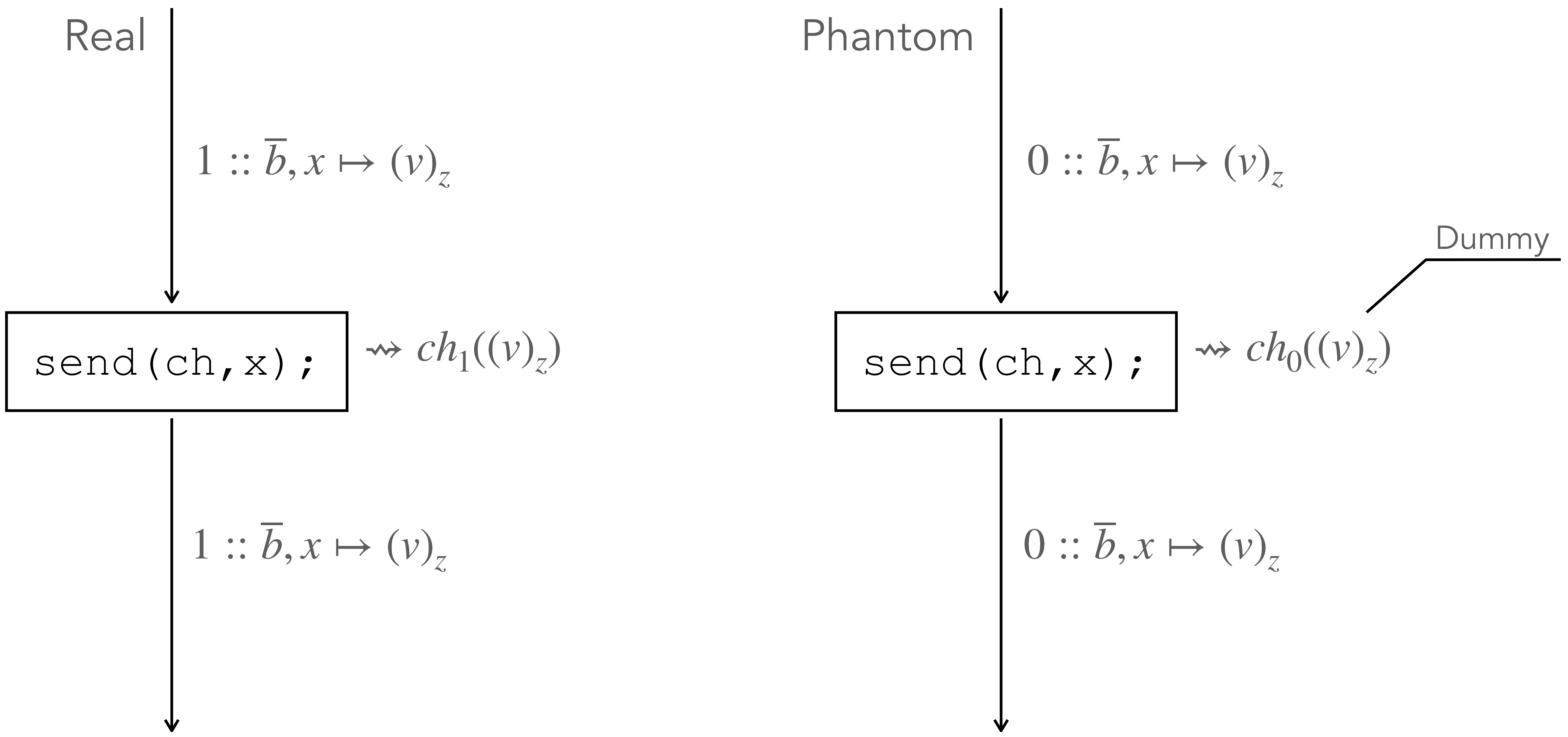
`x ?= “Goodbye”;`

$0 :: \overline{b}, x \mapsto (\text{"Hello"})_7$

Oblivious semantics

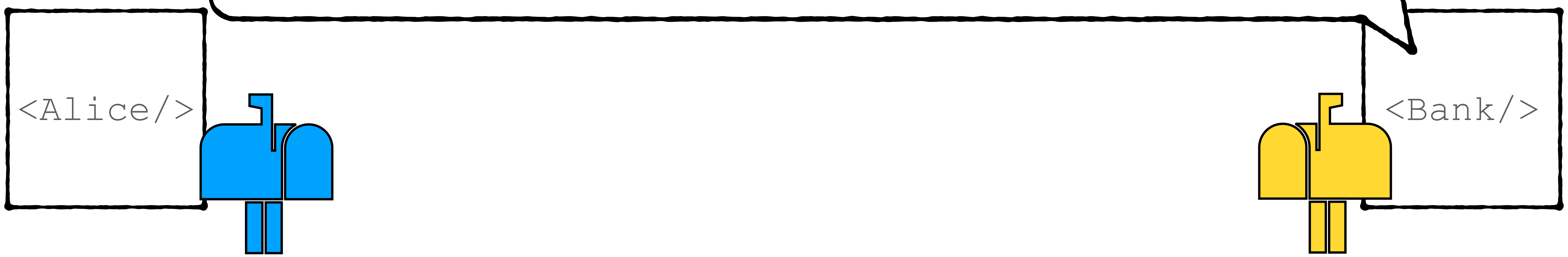
Sending

Send



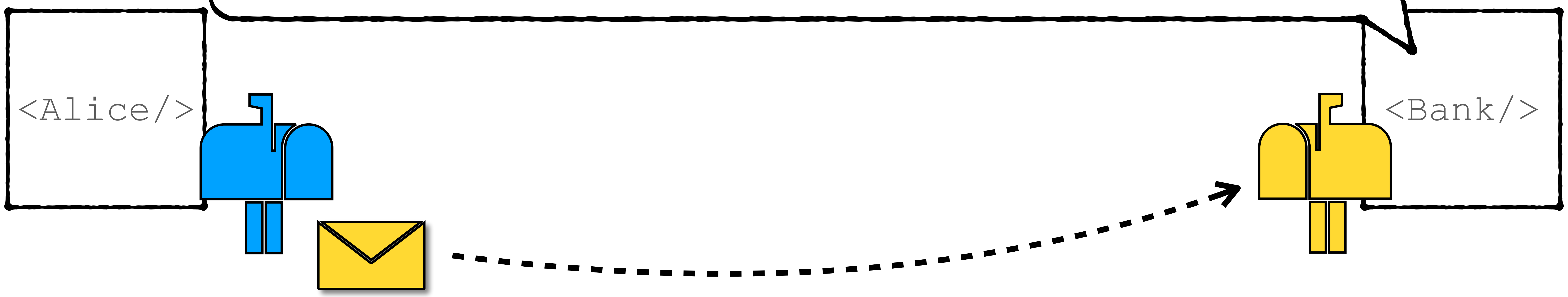
Intro example in OblivIO

```
TRANSFER(from: int, amount: int, to: int) {
  oblif amount <= balance[from]
  then {
    balance[from] ?= balance[from] - amount;
    balance[to]   ?= balance[to]   + amount;
  }
  else send(ALICE, "ERROR!");
}
```



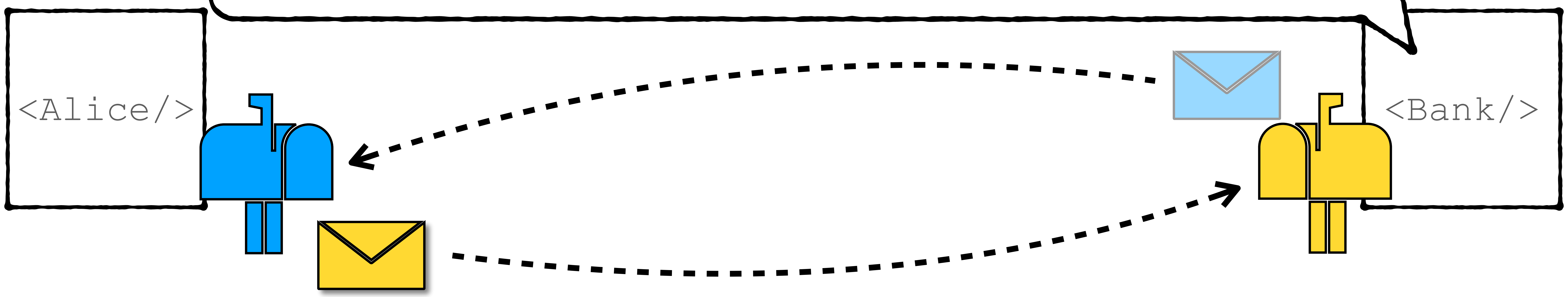
Intro example in OblivIO

```
TRANSFER(from: int, amount: int, to: int) {
  oblif amount <= balance[from]
  then {
    balance[from] ?= balance[from] - amount;
    balance[to]   ?= balance[to]   + amount;
  }
  else send(ALICE, “ERROR!”);
}
```



Intro example in OblivIO

```
TRANSFER(from: int, amount: int, to: int) {
  oblif amount <= balance[from]
  then {
    balance[from] ?= balance[from] - amount;
    balance[to]   ?= balance[to]   + amount;
  }
  else send(ALICE, "ERROR!");
}
```



Type system

Part a

T-If

$$\frac{\Gamma; \Delta \vdash e : int@\bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash c_2}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-OblivIf

$$\frac{\Gamma; \Delta \vdash e : int@\ell \qquad \ell \neq \bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash c_2}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \texttt{oblif}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-Assign

$$\frac{x \notin dom(\Delta) \qquad \Gamma(x) = \sigma@\ell_x \qquad \Gamma; \Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqsubseteq \ell_x}{\Gamma, \Pi, \Lambda; \Delta; \bot \vdash x = e}$$

T-OblivAssign

$$\frac{x \notin dom(\Delta) \qquad \Gamma(x) : \sigma@\ell_x \qquad \Gamma; \Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqcup pc \sqsubseteq \ell_x}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash x \mathrel{?=} e}$$

T-Send

$$\frac{\Gamma; \Delta \vdash e : \sigma@\ell_e \qquad \Lambda(ch) = \sigma@\ell_{mode}; \ell_{val} \qquad pc \sqsubseteq \ell_{mode} \qquad \ell_e \sqsubseteq \ell_{val}}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \texttt{send}(ch, e)}$$

Type system

Part a

Public guard

T-If

$$\frac{\Gamma;\Delta \vdash e : int@\bot \qquad \Gamma,\Pi,\Lambda;\Delta;pc \vdash c_1 \qquad \Gamma,\Pi,\Lambda;\Delta;pc \vdash c_2}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

Non-public guard

T-OblivIf

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash e : int@\ell \\ \ell \neq \bot \qquad \Gamma,\Pi,\Lambda;\Delta;pc \sqcup \ell \vdash c_1 \qquad \Gamma,\Pi,\Lambda;\Delta;pc \sqcup \ell \vdash c_2\end{array}}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{oblif}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-Assign

$$\frac{x \notin dom(\Delta) \qquad \Gamma(x) = \sigma@\ell_x \qquad \Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqsubseteq \ell_x}{\Gamma,\Pi,\Lambda;\Delta;\bot \vdash x = e}$$

T-OblivAssign

$$\frac{\begin{array}{c}x \notin dom(\Delta) \\ \Gamma(x) : \sigma@\ell_x \qquad \Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqcup pc \sqsubseteq \ell_x\end{array}}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash x\ ?\!= e}$$

T-Send

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \Lambda(ch) = \sigma@\ell_{mode};\ell_{val} \\ pc \sqsubseteq \ell_{mode} \qquad \ell_e \sqsubseteq \ell_{val}\end{array}}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{send}(ch, e)}$$

Type system

Part a

Public guard

T-If

$$\frac{\Gamma;\Delta \vdash e : int@\bot \qquad \Gamma,\Pi,\Lambda;\Delta;pc \vdash c_1 \qquad \Gamma,\Pi,\Lambda;\Delta;pc \vdash c_2}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

Non-public guard

T-OblivIf

$$\frac{\Gamma;\Delta \vdash e : int@\ell \qquad \ell \neq \bot \qquad \Gamma,\Pi,\Lambda;\Delta;pc \sqcup \ell \vdash c_1 \qquad \Gamma,\Pi,\Lambda;\Delta;pc \sqcup \ell \vdash c_2}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{oblif}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-Assign

$$\frac{x \notin dom(\Delta) \qquad \Gamma(x) = \sigma@\ell_x \qquad \Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqsubseteq \ell_x}{\Gamma,\Pi,\Lambda;\Delta;\bot \vdash x = e}$$

Public pc

T-OblivAssign

$$\frac{x \notin dom(\Delta) \qquad \Gamma(x) : \sigma@\ell_x \qquad \Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqcup pc \sqsubseteq \ell_x}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash x\ ?\!= e}$$

Any pc

T-Send

$$\frac{\Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \Lambda(ch) = \sigma@\ell_{mode};\ell_{val} \qquad pc \sqsubseteq \ell_{mode} \qquad \ell_e \sqsubseteq \ell_{val}}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{send}(ch, e)}$$

Type system

Part a

Public guard

T-If

$$\frac{\Gamma;\Delta \vdash e : int@\bot \qquad \Gamma,\Pi,\Lambda;\Delta;pc \vdash c_1 \qquad \Gamma,\Pi,\Lambda;\Delta;pc \vdash c_2}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

Non-public guard

T-OblivIf

$$\frac{\ell \neq \bot \qquad \begin{array}{c}\Gamma;\Delta \vdash e : int@\ell \\ \Gamma,\Pi,\Lambda;\Delta;pc \sqcup \ell \vdash c_1 \qquad \Gamma,\Pi,\Lambda;\Delta;pc \sqcup \ell \vdash c_2\end{array}}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{oblif}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-Assign

$$\frac{x \notin dom(\Delta) \qquad \Gamma(x) = \sigma@\ell_x \qquad \Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqsubseteq \ell_x}{\Gamma,\Pi,\Lambda;\Delta;\bot \vdash x = e}$$

Public pc

T-OblivAssign

$$\frac{\begin{array}{c} x \notin dom(\Delta) \\ \Gamma(x) : \sigma@\ell_x \qquad \Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \ell_e \sqcup pc \sqsubseteq \ell_x\end{array}}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash x\ ?{=}\ e}$$

Any pc

T-Send

$$\frac{\begin{array}{c}\Gamma;\Delta \vdash e : \sigma@\ell_e \qquad \Lambda(ch) = \sigma@\ell_{mode};\ell_{val} \\ pc \sqsubseteq \ell_{mode} \qquad \ell_e \sqsubseteq \ell_{val}\end{array}}{\Gamma,\Pi,\Lambda;\Delta;pc \vdash \texttt{send}(ch, e)}$$

Soundness Theorem:

Well-typed OblivIO programs do not leak by their traffic patterns

$$k(cfg, \tau \cdot \alpha, \ell_{adv}) \supseteq k(cfg, \tau, \ell_{adv})$$

Secure, but at what cost…

A pitfall of oblivious execution

PING₁(1)

Message queue

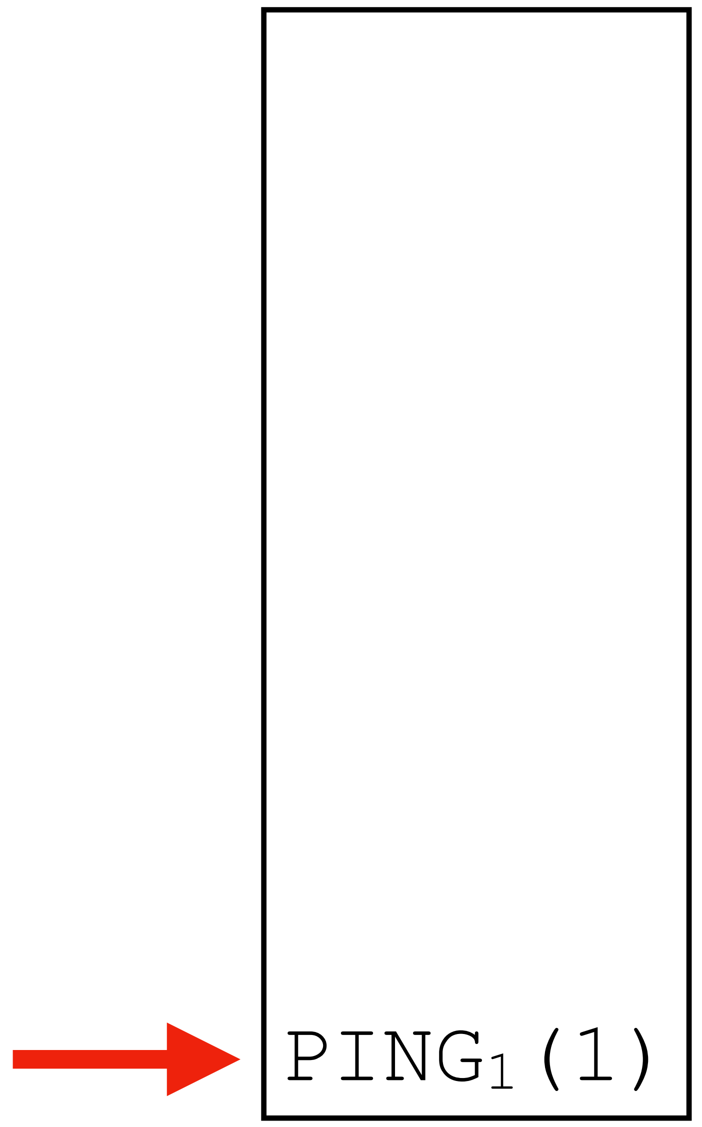
```
PING_H (x: int_H) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

```
PONG_H (x: int_H) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```

Message queue

Secure, but at what cost…

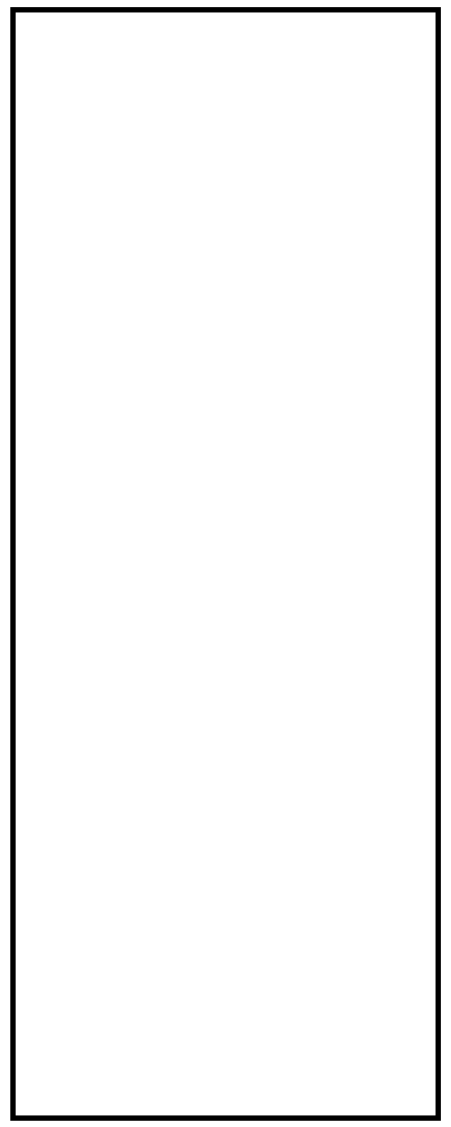
A pitfall of oblivious execution



Message queue

```
PING_H (x: int_H) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

```
PONG_H (x: int_H) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```



Message queue

Secure, but at what cost...

A pitfall of oblivious execution

```
PINGH (x: intH) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

```
PONGH (x: intH) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```

Message queue

$\mathtt{PONG}_0\mathtt{(0)}$
$\mathtt{PONG}_1\mathtt{(1)}$

Message queue

Secure, but at what cost…

A pitfall of oblivious execution

Message queue

```
PINGH (x: intH) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

```
PONGH (x: intH) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```

$\mathrm{PONG}_0(0)$

$\mathrm{PONG}_1(1)$

Message queue

Secure, but at what cost…

A pitfall of oblivious execution

$PING_0(0)$
$PING_1(1)$

Message queue

```
PING_H (x: int_H) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

```
PONG_H (x: int_H) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```

$PONG_0(0)$

Message queue

Secure, but at what cost…

A pitfall of oblivious execution

⋮
PING$_0$(0)
PING$_0$(1)
PING$_0$(0)
PING$_0$(1)
PING$_0$(0)
PING$_0$(1)
PING$_0$(0)
PING$_0$(1)
PING$_0$(0)
PING$_0$(1)

Message queue

```
PINGH (x: intH) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

```
PONGH (x: intH) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```

⋮
PONG$_0$(1)
PONG$_0$(0)
PONG$_0$(1)
PONG$_0$(0)
PONG$_0$(1)
PONG$_0$(0)
PONG$_0$(1)
PONG$_0$(0)
PONG$_1$(1)
PONG$_0$(0)

Message queue

Secure, but at what cost…

A pitfall of oblivious execution

```
⋮
PING₀(0)
PING₀(1)
PING₀(0)
PING₀(1)
PING₀(0)
PING₀(1)
PING₀(0)
PING₀(1)
PING₀(0)
PING₀(1)
```

Message queue

```
PINGH (x: intH) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

```
PONGH (x: intH) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```

```
⋮
PONG₀(1)
PONG₀(0)
PONG₀(1)
PONG₀(0)
PONG₀(1)
PONG₀(0)
PONG₀(1)
PONG₀(0)
PONG₁(1)
PONG₀(0)
```

Message queue

Idea:

Statically restrict the amount of dummy traffic produced by a program

Restricting the amount of dummy traffic

Resource awareness[5]

- Declare integer *potential* q of a handler
  - Spend potential when sending obliviously
  - Oblivious send on channel with potential q costs $1 + q$
    - 1 to pay for the message itself
    - q to pay for the potential of the handler
- Instrument typing judgements with potentials

[5] J. Hoffmann and M. Hofmann, "Amortized resource analysis with polynomial potential," in European Symposium on Programming. Springer, 2010, pp. 287–306.
J. Hoffmann, K. Aehlig, and M. Hofmann, "Resource aware ml," in International Conference on Computer Aided Verification. Springer, 2012, pp. 781–786.

Adding potentials

T-If

$$\frac{\Gamma; \Delta \vdash e : int@\bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash c_2}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2}$$

T-OblivIf

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash e : int@\ell \\ \ell \neq \bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash c_2\end{array}}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \texttt{oblif } e \texttt{ then } c_1 \texttt{ else } c_2}$$

T-Send

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash e : \sigma@\ell_e \qquad \Lambda(ch) = \sigma@\ell_{mode}; \ell_{val} \\ pc \sqsubseteq \ell_{mode} \qquad \ell_e \sqsubseteq \ell_{val}\end{array}}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash \texttt{send}(ch, e)}$$

Adding potentials

T-If

$$\frac{\Gamma; \Delta \vdash e : \mathit{int}@\bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q} c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q} c_2}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q} \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-OblivIf

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash e : \mathit{int}@\ell \\ \ell \neq \bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash^{q_1} c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash^{q_2} c_2\end{array}}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q_1+q_2} \texttt{oblif}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-Send

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash e : \sigma@\ell_e \qquad \Lambda(ch) = \sigma@\ell_{mode}; \ell_{val}; r \\ pc \sqsubseteq \ell_{mode} \qquad \ell_e \sqsubseteq \ell_{val} \qquad q' = \begin{cases} 0 & \mathit{if}\ pc = \bot \\ 1 + r & \mathit{otherwise} \end{cases}\end{array}}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q+q'} \texttt{send}(ch, e)}$$

Adding potentials

T-If

$$\frac{\Gamma; \Delta \vdash e : int@\bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q} c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q} c_2}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q} \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-OblivIf

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash e : int@\ell \\ \ell \neq \bot \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash^{q_1} c_1 \qquad \Gamma, \Pi, \Lambda; \Delta; pc \sqcup \ell \vdash^{q_2} c_2\end{array}}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q_1+q_2} \texttt{oblif}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

T-Send

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash e : \sigma@\ell_e \qquad \Lambda(ch) = \sigma@\ell_{mode}; \ell_{val}; r \\ pc \sqsubseteq \ell_{mode} \qquad \ell_e \sqsubseteq \ell_{val} \qquad q' = \begin{cases} 0 & \textit{if } pc = \bot \\ 1 + r & \textit{otherwise} \end{cases}\end{array}}{\Gamma, \Pi, \Lambda; \Delta; pc \vdash^{q+q'} \texttt{send}(ch, e)}$$

Overhead Theorem:

- Given
  - (System-wide) OblivIO trace τ_1
  - (System-wide) Unpadded trace τ_2
    - Without *dummy* messages
- Then
  - $|\tau_1| \leq |\tau_2| * c$

Example

Example revisited

```
PINGH $N (x: intH) {
  oblif x
  then send(PONG,1);
  else send(PONG,0);
}
```

$N ≥ 2+2*$M

```
PONGH $M (x: intH) {
  oblif x
  then send(PING,1);
  else send(PING,0);
}
```

$M ≥ 2+2*$N

Example: Round auction

```
var round_counter: int_L = 500;
var leader: string_H = "";
var leading_bid: int_H = 0;

BID_H $0 (name: string_H, bid: int_H) {
  oblif leading_bid < bid
  then {
    leader ?= name;
    leading_bid ?= bid;
  }
  else skip;
}

TICK_L $0 (dmy: int_L) {
  if round_counter > 0
  then {
    round_counter = round_counter - 1;
    send(AUCTIONTIMER/BEGIN, 2000);
    ... // send AUCTION_STATUS to all users
  } else {
    ... // send AUCTION_OVER to all users
  }
}
```

AUCTIONHOUSE

```
var max_bid: int_H = 432;

AUCTION_STATUS_L $1 (name: string_H, bid: int_H) {
    oblif bid < max_bid && name != "Alice"
    then send(AUCTIONHOUSE/BID, ("Alice", bid + 1));
    else skip;
}

AUCTION_OVER_L $0 (winner: string_H, winning_bid: int_H) {
    ...
}
```

ALICE

```
var c: int_L = 0;

BEGIN_L $0 (i: int_L) {
    c = i;
    while (c > 0) do {
        c = c - 1;
    }
    send(AUCTIONHOUSE/TICK, 0);
}
```

AUCTIONTIMER

Discussion

Limitations

Discussion

Limitations

- Events are network messages only
  - Cannot react to events with secret presence

Discussion

Limitations

- Events are network messages only
  - Cannot react to events with secret presence
- Constant-time implementation of all operations

Discussion

Limitations

- Events are network messages only
  - Cannot react to events with secret presence
- Constant-time implementation of all operations
- Programs are static
  - No dynamically registered handlers
  - Functions not first-class

Discussion

Limitations

- Events are network messages only
  - Cannot react to events with secret presence
- Constant-time implementation of all operations
- Programs are static
  - No dynamically registered handlers
  - Functions not first-class
- Channels not first-class

```
oblif secret
then ch ?= ALICE/GREET;
else ch ?= BOB/GREET;
send(ch,”Hello");
```

Conclusion

OblivIO Takeaways

- Secures reactive programs by oblivious execution
  - Well-typed programs do not leak by their traffic pattern (Soundness theorem)
- Bounds the traffic overhead produced by the enforcement
  - Every real message generates at most c dummy messages (Overhead theorem)

How OblivIO secures observable properties of communication:

Message presence
Sending dummy messages under phantom mode

Message size
Padding value size at oblivious assignments

Message timing
Constant-time execution through data-obliviousness

Message recipient
Channels are given in program text

IFC Precision

On precision of dynamic fine-grained
information-flow control

Dynamic information flow control

Motivation and background

- Many popular web-languages are dynamic, e.g., JavaScript and Python
  - Dynamic enforcement via runtime monitor allows for precise reasoning
- Monitors are typically fail-safe and termination-insensitive
  - Stop program execution before insecure action
- Two approaches to monitors, both use security labels
  - Fine-grained: track labels on values
  - Coarse-grained: track labels on computation

Fine-grained IFC

- All values are intrinsically labelled v^{ℓ}

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 7^{\perp}]$$

$$x + y \rightarrow 5^{\{Alice\}} + 7^{\perp} \rightarrow (5+7)^{\{Alice\} \sqcup \perp} \rightarrow 12^{\{Alice\}}$$

- pc-label tracks sensitivity of executing a particular command

$pc = \perp$

$pc = \{Alice\}$

$pc = \perp$

```
if x
  then 1
  else 2
```

$\rightarrow \quad 1^{\{Alice\}}$

Coarse-grained IFC

Fine-grained IFC

- All values are intrinsically labelled v^{ℓ}

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 7^{\perp}]$$

$$x + y \to 5^{\{Alice\}} + 7^{\perp} \to (5+7)^{\{Alice\} \sqcup \perp} \to 12^{\{Alice\}}$$

- pc-label tracks sensitivity of executing a particular command

$pc = \perp$

$pc = \{Alice\}$

```
if x
  then 1
  else 2
```

$\to \quad 1^{\{Alice\}}$

$pc = \perp$

Coarse-grained IFC

- Computation has a *floating-label* pc

Fine-grained IFC

- All values are intrinsically labelled v^{ℓ}

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 7^{\perp}]$$

$$x + y \to 5^{\{Alice\}} + 7^{\perp} \to (5+7)^{\{Alice\} \sqcup \perp} \to 12^{\{Alice\}}$$

- pc-label tracks sensitivity of executing a particular command

$pc = \perp$

```
if x
  then 1
  else 2
```

$pc = \{Alice\}$ $\to$ $1^{\{Alice\}}$

$pc = \perp$

Coarse-grained IFC

- Computation has a *floating-label* pc
- Values are not labelled
  - Secrets are boxed with a label and require unboxing before being used

$$m = [x \mapsto \boxed{5}^{\{Alice\}}]$$

Fine-grained IFC

- All values are intrinsically labelled v^{ℓ}

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 7^{\perp}]$$

$$x + y \to 5^{\{Alice\}} + 7^{\perp} \to (5+7)^{\{Alice\} \sqcup \perp} \to 12^{\{Alice\}}$$

- pc-label tracks sensitivity of executing a particular command

$pc = \perp$

$pc = \{Alice\}$

```
if x
  then 1
  else 2
```

$\to \quad 1^{\{Alice\}}$

$pc = \perp$

Coarse-grained IFC

- Computation has a *floating-label* pc
- Values are not labelled
  - Secrets are boxed with a label and require unboxing before being used

$$m = [x \mapsto \boxed{5}^{\{Alice\}}]$$

Cannot access boxed value

```
if x
  then 1
  else 2
```

Fine-grained IFC

- All values are intrinsically labelled v^{ℓ}

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 7^{\perp}]$$

$$x + y \rightarrow 5^{\{Alice\}} + 7^{\perp} \rightarrow (5+7)^{\{Alice\} \sqcup \perp} \rightarrow 12^{\{Alice\}}$$

- pc-label tracks sensitivity of executing a particular command

$pc = \perp$

$pc = \{Alice\}$

```
if x
  then 1
  else 2
```

$\rightarrow$ $1^{\{Alice\}}$

$pc = \perp$

Coarse-grained IFC

- Computation has a *floating-label* pc
- Values are not labelled
  - Secrets are boxed with a label and require unboxing before being used

$$m = [x \mapsto \boxed{5}^{\{Alice\}}, x' \mapsto 5]$$

$pc = \perp$

$pc = \{Alice\}$

```
let x' = unlabel x in
if x'
  then 1
  else 2
```

Raises floating-label

Fine-grained IFC

- All values are intrinsically labelled v^{ℓ}

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 7^{\perp}]$$

$$x + y \rightarrow 5^{\{Alice\}} + 7^{\perp} \rightarrow (5+7)^{\{Alice\} \sqcup \perp} \rightarrow 12^{\{Alice\}}$$

- pc-label tracks sensitivity of executing a particular command

$pc = \perp$

$pc = \{Alice\}$

```
if x
  then 1
  else 2
```

$\rightarrow \quad 1^{\{Alice\}}$

$pc = \perp$

Coarse-grained IFC

- Computation has a *floating-label* pc
- Values are not labelled
  - Secrets are boxed with a label and require unboxing before being used

$$m = [x \mapsto \boxed{5}^{\{Alice\}}, x' \mapsto 5]$$

Provides computational scope

$pc = \{Alice\}$

```
toLabeled(
  let x' = unlabel x in
  if x'
    then 1
    else 2
)
```

Raises floating-label

$pc = \perp$

Fine-grained IFC ⟷ Coarse-grained IFC

Vassena et al. [POPL19]: fine- and coarse-grained dynamic IFC are equally expressive

- Formal setup: Two calculi
  - Fine- and coarse-grained
- Theorem: The two calculi are equally expressive
  - Shown by a pair of semantic preserving translations

- Assumptions
  1. Termination-insensitive security
  2. Programs are well-typed (in a security unaware way)
  3. The fine-grained calculus is standard

Lifting the assumption

What is this work about?

- Novel fine-grained IFC techniques for cases where the assumptions do not hold
  1. Disjunctive precision (Novel fine-grained semantics, PSNI)
  2. Refinement labels (Dynamically typed, PSNI)
- We show that the techniques have no translation to coarse-grained IFC
  - Fine- and coarse-grained dynamic IFC are not equivalent

Disjunctive precision

Standard expression semantics

- Results are tainted by the sensitivity of both operands

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}]$$

$$x * y \rightarrow 5^{\{Alice\}} * 0^{\perp} \rightarrow (5 * 0)^{\{Alice\} \sqcup \perp} \rightarrow 0^{\{Alice\}}$$

Disjunctive precision

Standard expression semantics

- Results are tainted by the sensitivity of both operands

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}]$$

$$x * y \to 5^{\{Alice\}} * 0^{\perp} \to (5 * 0)^{\{Alice\} \sqcup \perp} \to 0^{\{Alice\}}$$

- Does this result actually depend on the value of x?

Disjunctive precision

Standard expression semantics

- Results are tainted by the sensitivity of both operands

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}]$$

$$x * y \rightarrow 5^{\{Alice\}} * 0^{\perp} \rightarrow (5 * 0)^{\{Alice\} \sqcup \perp} \rightarrow 0^{\{Alice\}}$$

Semantics lacks precision

- Does this result actually depend on the value of x?

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\bot}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\overset{pc = \bot}{\underset{pc = \bot}{\texttt{x*y}}} \quad \to \quad 0^{\bot}$$

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\overset{pc = \perp}{\underset{pc = \perp}{\texttt{x*y}}} \quad \rightarrow \quad 0^{\perp}$$

- Non-trivial case?

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\overset{pc = \perp}{\underset{pc = \perp}{\texttt{x*y}}} \quad \to \quad 0^{\perp}$$

- Non-trivial case?

$$\overset{pc = \perp}{\underset{pc = \perp}{\texttt{x*z}}} \quad \to \quad 0^{\{Bob\}}?$$

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\bot}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\underset{pc = \bot}{\overset{pc = \bot}{\texttt{x*y}}} \quad \rightarrow \quad 0^{\bot}$$

- Non-trivial case?

Not safe!

$$\underset{pc = \bot}{\overset{pc = \bot}{\texttt{x*z}}} \quad \rightarrow \quad \cancel{0^{\{Bob\}}}?$$

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\texttt{x*y} \xrightarrow[pc=\perp]{pc=\perp} 0^{\perp}$$

- Non-trivial case?

$$\texttt{x*z} \xrightarrow[pc=\perp]{pc=\perp} \cancel{0^{\{Bob\}}}?$$

Not safe!

$$\left.\begin{array}{l} 5^{\{Alice\}} * 0^{\{Bob\}} \\ 0^{\{Alice\}} * 0^{\{Bob\}} \\ 0^{\{Alice\}} * 5^{\{Bob\}} \end{array}\right\}$$

Results must have same label

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\frac{pc = \perp}{pc = \perp} \quad \texttt{x*y} \quad \rightarrow \quad 0^{\perp}$$

- Non-trivial case?

Not safe!

$$\frac{pc = \perp}{pc = \perp} \quad \texttt{x*z} \quad \rightarrow \quad \cancel{0^{\{Bob\}}}?$$

Results must have same label

$$\left.\begin{array}{l} 5^{\{Alice\}} * 0^{\{Bob\}} \\ 0^{\{Alice\}} * 0^{\{Bob\}} \\ 0^{\{Alice\}} * 5^{\{Bob\}} \end{array}\right\} \rightarrow 0^{\{Alice,Bob\}}$$

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\frac{pc = \perp}{pc = \perp} \quad \texttt{x*y} \quad \to \quad 0^{\perp}$$

- Non-trivial case?

Not safe!

$$\frac{pc = \perp}{pc = \perp} \quad \texttt{x*z} \quad \to \quad \cancel{0^{\{Bob\}}}?$$

$$\left.\begin{aligned} &5^{\{Alice\}} * 0^{\{Bob\}} \\ &0^{\{Alice\}} * 0^{\{Bob\}} \\ &0^{\{Alice\}} * 5^{\{Bob\}} \end{aligned}\right\} \to 0^{\{Alice,Bob\}}$$

Results must have same label

No non-trivial cases?

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\texttt{x*y} \xrightarrow[pc = \perp]{pc = \perp} 0^{\perp}$$

- Non-trivial case

Precise if $pc = \{Bob\}$

$$\texttt{x*z} \xrightarrow[pc = \{Bob\}]{pc = \{Bob\}} 0^{\{Bob\}}$$

Disjunctive precision

Precise expression semantics

- Setup: Integer values n and binary operations $x_1 \oplus x_2$
  - Precise multiplication if either x_1 or x_2 is zero

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 0^{\perp}, z \mapsto 0^{\{Bob\}}]$$

- Trivial case

$$\langle \texttt{x*y} \rangle_{pc=\perp}^{pc=\perp} \rightarrow 0^{\perp}$$

- Non-trivial case

Precise if $pc = \{Bob\}$

$$\langle \texttt{x*z} \rangle_{pc=\{Bob\}}^{pc=\{Bob\}} \rightarrow 0^{\{Bob\}}$$

$$\left.\begin{array}{l} 5^{\{Alice\}} * 0^{\{Bob\}} \\ 0^{\{Alice\}} * 0^{\{Bob\}} \end{array}\right\} \rightarrow 0^{\{Bob\}}$$

$$0^{\{Alice\}} * 5^{\{Bob\}} \rightarrow 0^{\{Alice,Bob\}}$$

Refinement labels*

Standard PSNI in a dynamically-typed setting

- Setup: Unit value () and integer values n and ternary conditional operator $x \mathbin{?} x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x \mathbin{?} y : z \rightarrow 5^{\{Alice\}} \mathbin{?} 42^{\perp} : 84^{\perp} \rightarrow 42^{\{Alice\}}$$

Refinement labels*

Standard PSNI in a dynamically-typed setting

- Setup: Unit value () and integer values n and ternary conditional operator $x \mathbin{?} x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x \mathbin{?} y : z \to 5^{\{Alice\}} \mathbin{?} 42^{\perp} : 84^{\perp} \to 42^{\{Alice\}}$$

Does the following program satisfy PSNI?

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in output(⊥, ”Done!”)
```

Refinement labels*

Standard PSNI in a dynamically-typed setting

- Setup: Unit value () and integer values n and ternary conditional operator $x \mathrel{?} x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x \mathrel{?} y : z \to 5^{\{Alice\}} \mathrel{?} 42^{\perp} : 84^{\perp} \to 42^{\{Alice\}}$$

Does the following program satisfy PSNI?

a is always an integer

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in output(⊥, ”Done!”)
```

Refinement labels*

Standard PSNI in a dynamically-typed setting

- Setup: Unit value () and integer values n and ternary conditional operator $x \mathbin{?} x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x \mathbin{?} y : z \to 5^{\{Alice\}} \mathbin{?} 42^{\perp} : 84^{\perp} \to 42^{\{Alice\}}$$

Does the following program satisfy PSNI?

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in output(⊥, ”Done!”)
```

a is always an integer

output is always reachable

Refinement labels*

Standard PSNI in a dynamically-typed setting

- Setup: Unit value () and integer values n and ternary conditional operator $x \mathbin{?} x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x \mathbin{?} y : z \to 5^{\{Alice\}} \mathbin{?} 42^{\perp} : 84^{\perp} \to 42^{\{Alice\}}$$

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in output(⊥, ”Done!”)
```

a is always an integer

output is always reachable

Program satisfies PSNI!

Refinement labels*

Standard IFC monitors lacks precision

- Setup: Unit value () and integer values n and ternary conditional operator $x \;?\; x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x \;?\; y : z \to 5^{\{Alice\}} \;?\; 42^{\perp} : 84^{\perp} \to 42^{\{Alice\}}$$

$pc = \perp$

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in output(⊥, "Done!")
```

Refinement labels*

Standard IFC monitors lacks precision

- Setup: Unit value () and integer values n and ternary conditional operator $x \ ? \ x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x \ ? \ y : z \rightarrow 5^{\{Alice\}} \ ? \ 42^{\perp} : 84^{\perp} \rightarrow 42^{\{Alice\}}$$

a is labelled $\{Alice\}$

$pc = \perp$

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in output(⊥, ”Done!”)
```

Refinement labels*

Standard IFC monitors lacks precision

- Setup: Unit value () and integer values n and ternary conditional operator $x\ ?\ x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x\ ?\ y : z \rightarrow 5^{\{Alice\}}\ ?\ 42^{\perp} : 84^{\perp} \rightarrow 42^{\{Alice\}}$$

a is labelled $\{Alice\}$

Addition may fail so pc is tainted by the label of its operands

$pc = \perp$

$pc = \{Alice\}$

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in output(⊥, "Done!")
```

Refinement labels*

Standard IFC monitors lacks precision

- Setup: Unit value () and integer values n and ternary conditional operator $x\ ?\ x_1 : x_2$

$$m = [x \mapsto 5^{\{Alice\}}, y \mapsto 42^{\perp}, z \mapsto 84^{\perp}, w \mapsto ()^{\perp}]$$

$$x\ ?\ y : z \rightarrow 5^{\{Alice\}}\ ?\ 42^{\perp} : 84^{\perp} \rightarrow 42^{\{Alice\}}$$

a is labelled $\{Alice\}$

Addition may fail so pc is tainted by the label of its operands

$pc = \perp$

$pc = \{Alice\}$

```
let a = x ? y : z
    b = a + 1 (* dynamic type error if a is unit *)
in ~~output(⊥, "Done!")~~
```

Public side-effect when $pc = \{Alice\}$

Refinement labels*

Tracking the sensitivity of types

- Two-label approach: v^{ℓ^v / ℓ^t}

Refinement labels*

Tracking the sensitivity of types

Value label

- Two-label approach: v^{ℓ^v/ℓ^t}

Refinement labels*

Tracking the sensitivity of types

Value label Type label

- Two-label approach: v^{ℓ^v / ℓ^t}

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

$pc = \bot$

```
let a = x ? y : z   →   42^{Alice}/⊥
    b = a + 1
in output(⊥, ”Done!”)
```

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

$\bot$ since $42 \overset{type}{=} 84$

$pc = \bot$

```
let a = x ? y : z   →   42^{Alice}/⊥
    b = a + 1
in output(⊥, "Done!")
```

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

$\{Alice\}$ since $42 \neq 84$

$\bot$ since $42 \overset{type}{=} 84$

$pc = \bot$

```
let a = x ? y : z   →   42^{Alice}/⊥
    b = a + 1
in output(⊥, ”Done!”)
```

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

$\{Alice\}$ since $42 \neq 84$

$\bot$ since $42 \overset{type}{=} 84$

$pc = \bot$

```
let a = x ? y : z   →   42{Alice}/⊥
    b = a + 1
in output(⊥, "Done!")
```

Public that a is integer and operation succeeds

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

$\{Alice\}$ since $42 \neq 84$

$\bot$ since $42 \overset{type}{=} 84$

$pc = \bot$

```
let a = x ? y : z   →   42^{Alice}/⊥
    b = a + 1
in output(⊥, ”Done!”)
```

$pc = \bot$

Public that a is integer and operation succeeds

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

$\{Alice\}$ since $42 \neq 84$

$\bot$ since $42 \overset{type}{=} 84$

$pc = \bot$

```
let a = x ? y : z   →   42{Alice}/⊥
    b = a + 1
in output(⊥, ”Done!”)
```

$pc = \bot$

Public that a is integer and operation succeeds

Success!

Refinement labels*

Tracking the sensitivity of types

Value label

Type label

- Two-label approach: v^{ℓ^v/ℓ^t}

Always the case that $\ell^t \sqsubseteq \ell^v$

$$m = [x \mapsto 5^{\{Alice\}/\bot}, y \mapsto 42^{\bot/\bot}, z \mapsto 84^{\bot/\bot}, w \mapsto ()^{\bot/\bot}]$$

$\{Alice\}$ since $42 \neq 84$

$\bot$ since $42 \overset{type}{=} 84$

$pc = \bot$

$pc = \bot$

```
let a = x ? y : z   →  42{Alice}/⊥
    b = a + 1
in output(⊥, "Done!")
```

Public that a is integer and operation succeeds

Success!

Non-trivial cases of $x \mathbin{?} x_1 : x_2$

$$\ell^t = \begin{cases} \ell^t_i \sqcup pc & \text{if } v_1 \overset{type}{=} v_2 \wedge pc \sqcup \ell^t_1 = pc \sqcup \ell^t_2 \\ \ell^v_x \sqcup \ell^t_i \sqcup pc & \text{otherwise} \end{cases}$$

$$\ell^v = \begin{cases} \ell^v_i \sqcup \ell^t \sqcup pc & \text{if } v_1 = v_2 \wedge pc \sqcup \ell^v_1 = pc \sqcup \ell^v_2 \\ \ell^v_x \sqcup \ell^v_i \sqcup \ell^t \sqcup pc & \text{otherwise} \end{cases}$$

* Semantics of $x \mathbin{?} x_1 : x_2$ makes use of disjunctive precision

Our Fine-grained IFC ⟷ ? Coarse-grained IFC

Values v		Translated values $\llbracket v \rrbracket$
Memories m	Translation $\llbracket \cdot \rrbracket$	Translated memories $\llbracket m \rrbracket$
Expressions e		Translated expressions $\llbracket e \rrbracket$

Jeppe Fredsgaard Blaabjerg, Aarhus University

Our Fine-grained IFC ⟷ ? ⟷ Coarse-grained IFC

Values v

Memories m

Expressions e

Translation $[\![\cdot]\!]$

How can we show that no $[\![\cdot]\!]$ exists?

Translated values $[\![v]\!]$

Translated memories $[\![m]\!]$

Translated expressions $[\![e]\!]$

Our Fine-grained IFC $\longleftrightarrow$? Coarse-grained IFC

Values v		Translated values $[\![v]\!]$
Memories m	Translation $[\![\cdot]\!]$	Translated memories $[\![m]\!]$
Expressions e		Translated expressions $[\![e]\!]$

How can we show that no $[\![\cdot]\!]$ exists?

- Translation $[\![\cdot]\!]$
  - Source language: Fine-grained calculus with disjunctive precision
  - Target language: Sequential coarse-grained calculus for PSNI
- Cannot use **toLabeled** for PSNI [Stefan et al., ICFP'12]
- Modify the coarse-grained calculus of Vassena et al. [POPL19]
  - Replace **toLabeled** with **label**
  - Add integer values n and binary expressions $e_1 \oplus e_2$

Our Fine-grained IFC $\longleftrightarrow$? Coarse-grained IFC

Values v

Memories m

Expressions e

Translation $[\![\cdot]\!]$

How can we show that no $[\![\cdot]\!]$ exists?

Translated values $[\![v]\!]$

Translated memories $[\![m]\!]$

Translated expressions $[\![e]\!]$

- Translation $[\![\cdot]\!]$
  - Source language: Fine-grained calculus with disjunctive precision
  - Target language: Sequential coarse-grained calculus for PSNI
- Cannot use **toLabeled** for PSNI [Stefan et al., ICFP'12]
- Modify the coarse-grained calculus of Vassena et al. [POPL19]
  - Replace **toLabeled** with **label**

    Does not restore floating-label after evaluation
  - Add integer values n and binary expressions $e_1 \oplus e_2$

Translating disjunctive precision

Proof strategy

- What does the translation $[\![\cdot]\!]$ look like?
  - On values: $[\![v]\!]$
  - On memories: $[\![m]\!]$
  - On expressions: $[\![e]\!]$

Translating disjunctive precision

Proof strategy

- What does the translation $\llbracket \cdot \rrbracket$ look like?
  - On values: $\llbracket v \rrbracket$
  - On memories: $\llbracket m \rrbracket$
  - On expressions: $\llbracket e \rrbracket$

Could in principle have any shape

Translating disjunctive precision

Proof strategy

- What does the translation $[\![\cdot]\!]$ look like?
  - On values: $[\![v]\!]$
  - On memories: $[\![m]\!]$
  - On expressions: $[\![e]\!]$

Could in principle have any shape

- Strategy: Define 4 properties that translations must satisfy

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e \rangle \Downarrow^{m} \langle pc', v \rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!] \rangle \longrightarrow^{*} \langle m', pc', [\![v]\!] \rangle$

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e\rangle \Downarrow^m \langle pc', v\rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!]\rangle \longrightarrow^* \langle m', pc', [\![v]\!]\rangle$

Property 2: Translation of values

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e \rangle \Downarrow^m \langle pc', v \rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!] \rangle \longrightarrow^* \langle m', pc', [\![v]\!] \rangle$

What does $[\![n^\ell]\!]$ look like?

Property 2: Translation of values

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e\rangle \Downarrow^m \langle pc', v\rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!]\rangle \longrightarrow^* \langle m', pc', [\![v]\!]\rangle$

What does $[\![n^\ell]\!]$ look like?

Property 2: Translation of values

$$[\![n^\ell]\!] = \boxed{n}^\ell ?$$

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e\rangle \Downarrow^m \langle pc', v\rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!]\rangle \longrightarrow^* \langle m', pc', [\![v]\!]\rangle$

What does $[\![n^\ell]\!]$ look like?

Property 2: Translation of values

$$[\![n^\ell]\!] = \boxed{n}^\ell?$$
$$= (42, \boxed{n}^\ell)?$$

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e\rangle \Downarrow^m \langle pc', v\rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!]\rangle \longrightarrow^* \langle m', pc', [\![v]\!]\rangle$

What does $[\![n^\ell]\!]$ look like?

Property 2: Translation of values

$$[\![n^\ell]\!] = \boxed{n}^\ell?$$
$$= (42, \boxed{n}^\ell)?$$
$$= (\lambda x\,.\,e, m)?$$

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e\rangle \Downarrow^m \langle pc', v\rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!]\rangle \longrightarrow^* \langle m', pc', [\![v]\!]\rangle$

What does $[\![n^\ell]\!]$ look like?

Property 2: Translation of values

$$\begin{aligned}[\![n^\ell]\!] &= \boxed{n}^\ell ?\\ &= (42, \boxed{n}^\ell)?\\ &= (\lambda x\,.\,e, m)?\\ &\ldots\end{aligned}$$

Translating disjunctive precision

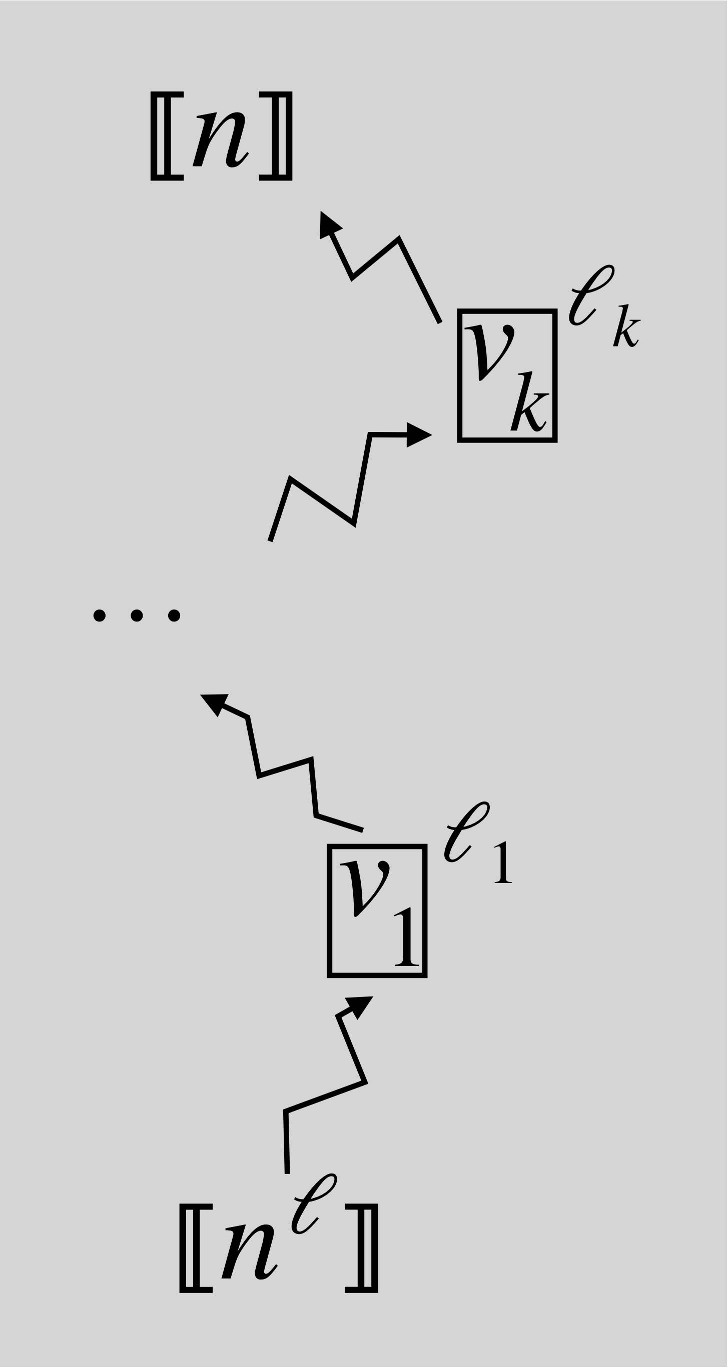
Property 1: Semantics preserving

- If $\langle pc, e \rangle \Downarrow^m \langle pc', v \rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!] \rangle \longrightarrow^* \langle m', pc', [\![v]\!] \rangle$

Property 2: Translation of values

What does $[\![n^\ell]\!]$ look like?

$$[\![n^\ell]\!] = \boxed{n}^\ell ?$$
$$= (42, \boxed{n}^\ell)?$$
$$= (\lambda x \,.\, e, m)?$$
$$\ldots$$



Translating disjunctive precision

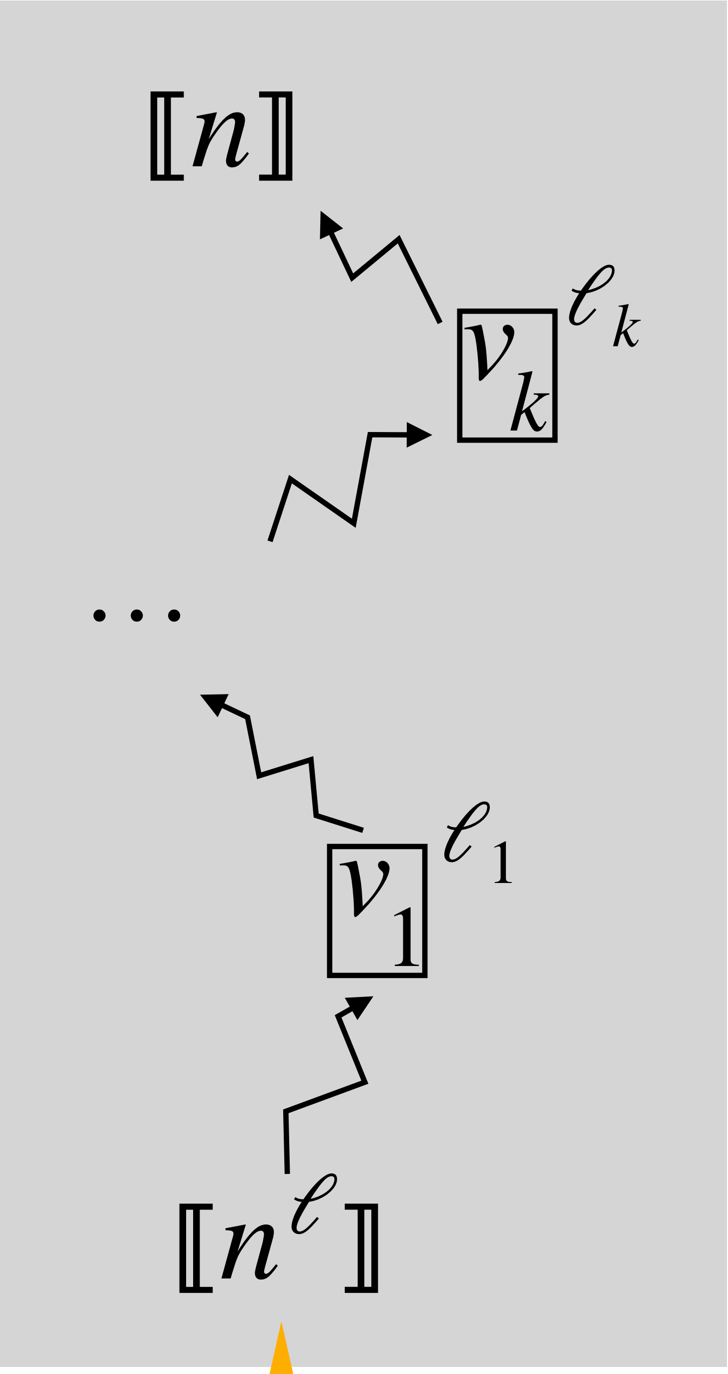
Property 1: Semantics preserving

- If $\langle pc, e \rangle \Downarrow^m \langle pc', v \rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!] \rangle \longrightarrow^* \langle m', pc', [\![v]\!] \rangle$

Property 2: Translation of values

What does $[\![n^\ell]\!]$ look like?

$$
\begin{aligned}
[\![n^\ell]\!] &= \boxed{n}^{\ell}? \\
&= (42, \boxed{n}^{\ell})? \\
&= (\lambda x \,.\, e, m)? \\
&\ldots
\end{aligned}
$$



If there is a *path* from $[\![n^\ell]\!]$ to $[\![n]\!]$ and if $\ell' = \ell_1 \sqcup \ldots \sqcup \ell_k$ is the *least sensitive boxing* along any such path, we say that $[\![n]\!]$ is included in $[\![n^\ell]\!]$ under label ℓ'

Translating disjunctive precision

Property 1: Semantics preserving

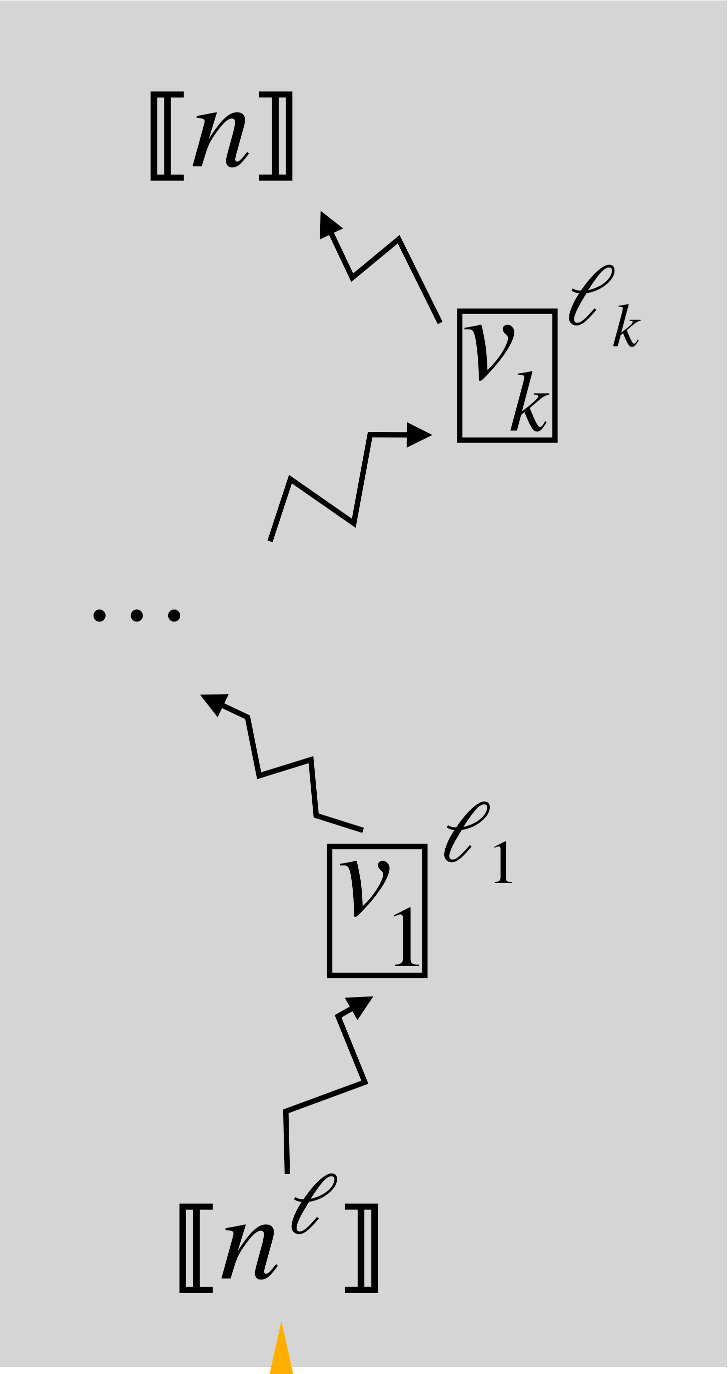
- If $\langle pc, e \rangle \Downarrow^m \langle pc', v \rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!] \rangle \longrightarrow^* \langle m', pc', [\![v]\!] \rangle$

Property 2: Translation of values

- $[\![n]\!]$ is included in $[\![n^\ell]\!]$ under label ℓ

What does $[\![n^\ell]\!]$ look like?

$$
\begin{aligned}
[\![n^\ell]\!] &= \boxed{n}^\ell ? \\
&= (42, \boxed{n}^\ell)? \\
&= (\lambda x \,.\, e, m)? \\
&\dots
\end{aligned}
$$



If there is a *path* from $[\![n^\ell]\!]$ to $[\![n]\!]$ and if $\ell' = \ell_1 \sqcup \dots \sqcup \ell_k$ is the *least sensitive boxing* along any such path, we say that $[\![n]\!]$ is included in $[\![n^\ell]\!]$ under label ℓ'

Translating disjunctive precision

Property 1: Semantics preserving

- If $\langle pc, e\rangle \Downarrow^m \langle pc', v\rangle$
- Then $\langle [\![m]\!], pc, [\![e]\!]\rangle \longrightarrow^* \langle m', pc', [\![v]\!]\rangle$

What does $[\![n^\ell]\!]$ look like?

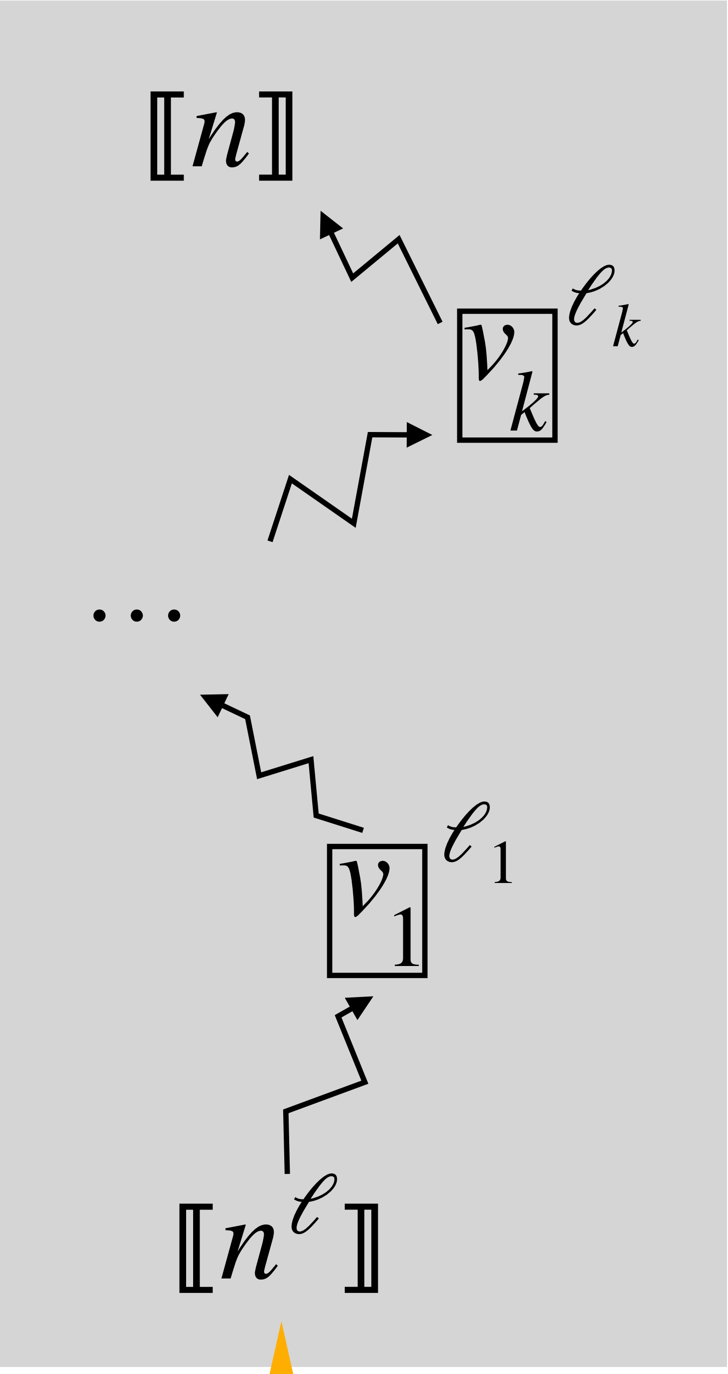
Property 2: Translation of values

- $[\![n]\!]$ is included in $[\![n^\ell]\!]$ under label ℓ

$$
\begin{aligned}
[\![n^\ell]\!] &= \boxed{n}^{\ell}? \\
&= (42, \boxed{n}^{\ell})? \\
&= (\lambda x \,.\, e, m)? \\
&\dots
\end{aligned}
$$

Property 3: Translation of memories

- Point-wise, i.e., $[\![m]\!] = \lambda x \,.\, [\![m(x)]\!]$



If there is a *path* from $[\![n^\ell]\!]$ to $[\![n]\!]$ and if $\ell' = \ell_1 \sqcup \dots \sqcup \ell_k$ is the *least sensitive boxing* along any such path, we say that $[\![n]\!]$ is included in $[\![n^\ell]\!]$ under label ℓ'

Translating disjunctive precision

Property 4: Translation of binary operations

Translating disjunctive precision

What does $[\![x_1 \oplus x_2]\!]$ look like?

Property 4: Translation of binary operations

Translating disjunctive precision

What does $[\![x_1 \oplus x_2]\!]$ look like?

Property 4: Translation of binary operations

- The values of the operands are necessary for computing the result

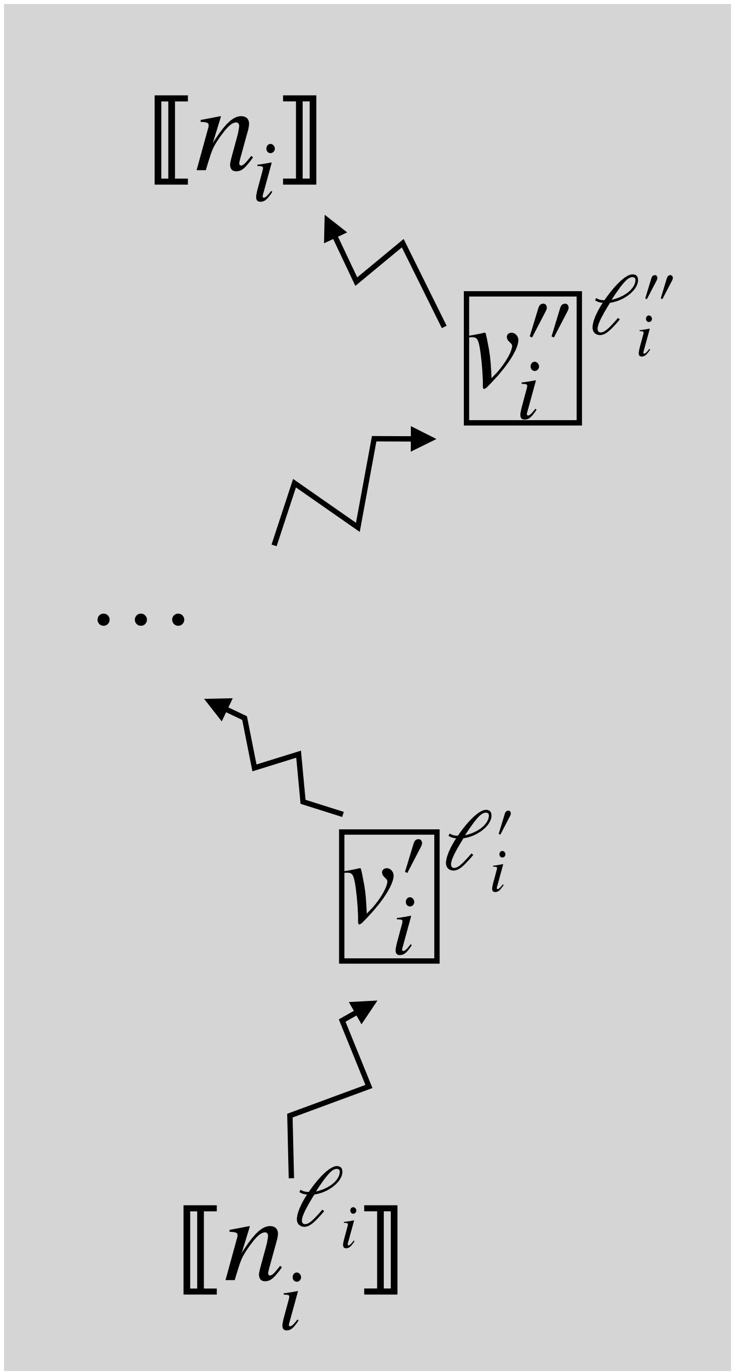
Translating disjunctive precision

What does $[\![x_1 \oplus x_2]\!]$ look like?

$$m = [x_1 \mapsto n_1^{\ell_1}, x_2 \mapsto n_2^{\ell_2}]$$

Property 4: Translation of binary operations

- The values of the operands are necessary for computing the result



Translation of $n_i^{\ell_i}$

Translating disjunctive precision

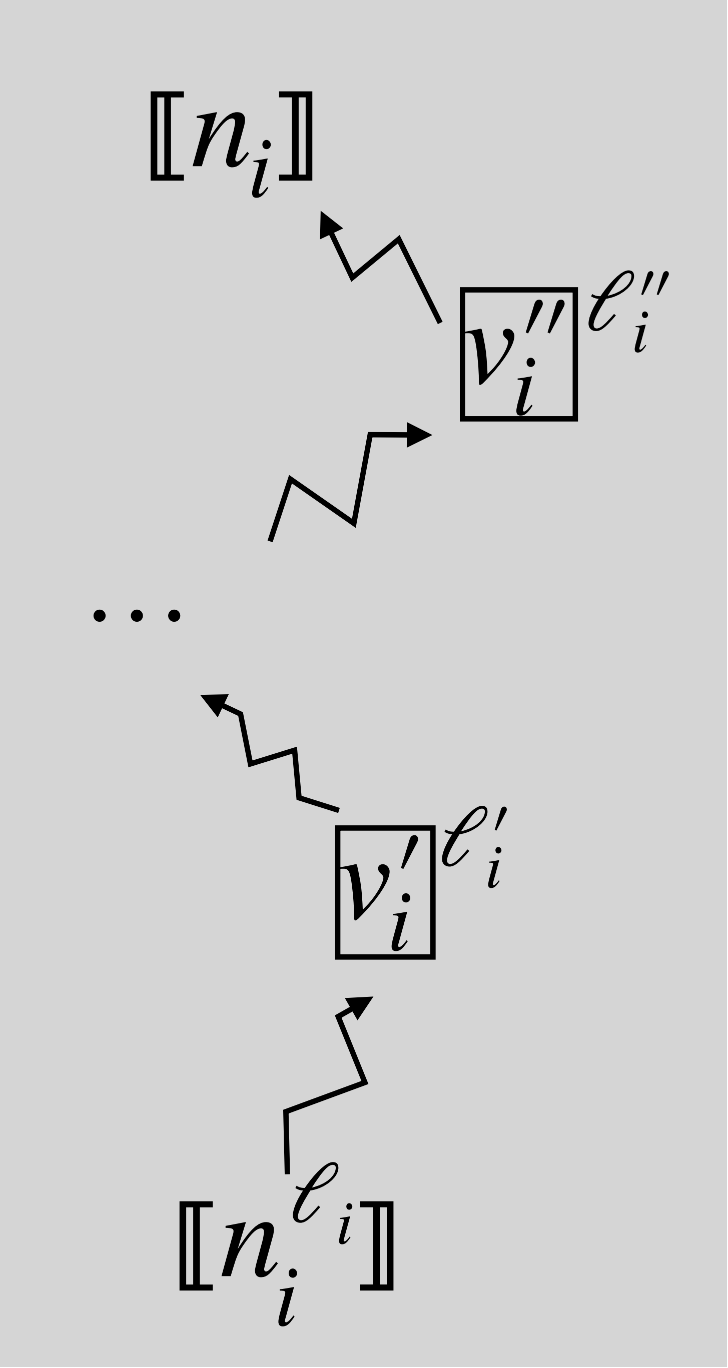
What does $[\![x_1 \oplus x_2]\!]$ look like?

$$m = [x_1 \mapsto n_1^{\ell_1}, x_2 \mapsto n_2^{\ell_2}]$$

Property 4: Translation of binary operations

- The values of the operands are necessary for computing the result

Recursively unlabels $[\![n_i]\!]$ from $[\![n_i^{\ell_i}]\!]$



Translation of $n_i^{\ell_i}$

Translating disjunctive precision

What does $[\![x_1 \oplus x_2]\!]$ look like?

$$m = [x_1 \mapsto n_1^{\ell_1}, x_2 \mapsto n_2^{\ell_2}]$$

Property 4: Translation of binary operations

- The values of the operands are necessary for computing the result

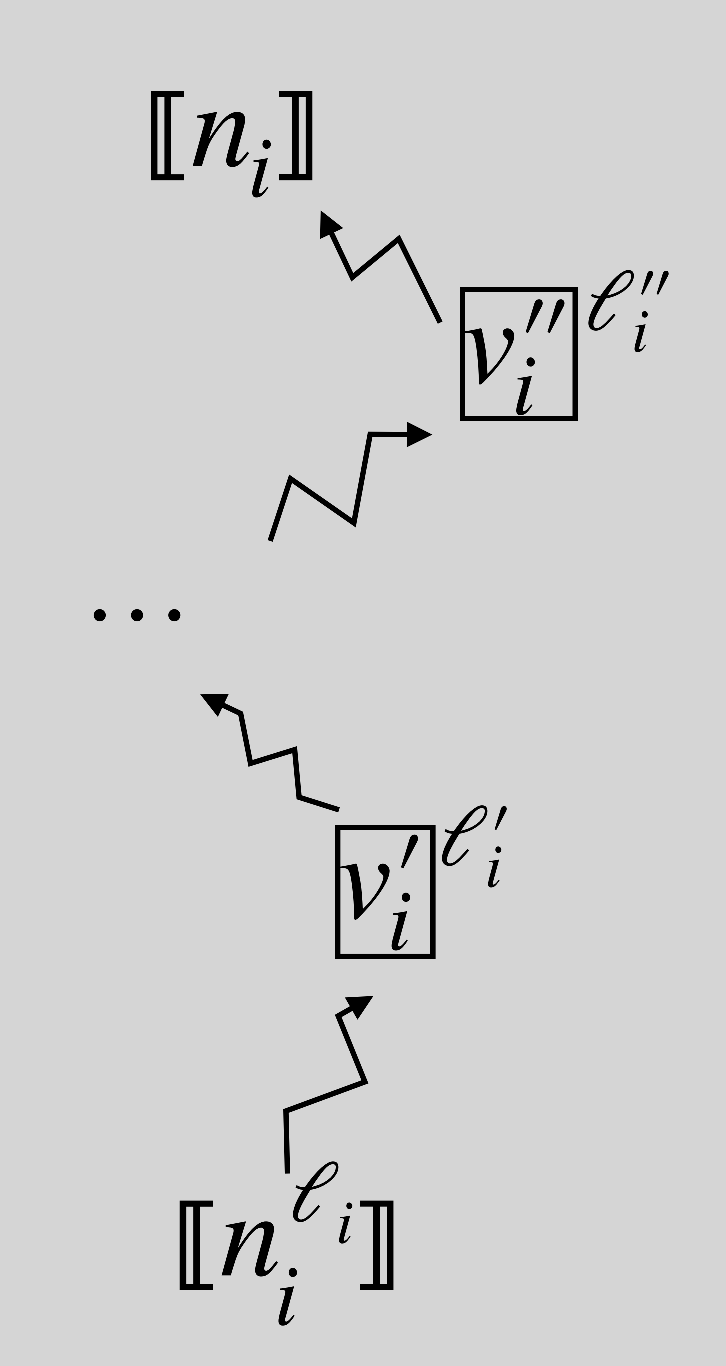
Recursively unlabels $[\![n_i]\!]$ from $[\![n_i^{\ell_i}]\!]$

$$\langle [\![m]\!], pc, [\![x_1 \oplus x_2]\!] \rangle \longrightarrow^* \langle m', pc', \mathbf{unlabel}(\boxed{v_i'}^{\ell_i'}) \rangle$$

$$\ldots$$

$$\longrightarrow^* \langle m'', pc'', \mathbf{unlabel}(\boxed{v_i''}^{\ell_i''}) \rangle$$

$$\longrightarrow^* c'$$



Translation of $n_i^{\ell_i}$

Translating disjunctive precision

What does $[\![x_1 \oplus x_2]\!]$ look like?

$m = [x_1 \mapsto n_1^{\ell_1}, x_2 \mapsto n_2^{\ell_2}]$

Property 4: Translation of binary operations

- The values of the operands are necessary for computing the result

Recursively unlabels $[\![n_i]\!]$ from $[\![n_i^{\ell_i}]\!]$

$$\langle [\![m]\!], pc, [\![x_1 \oplus x_2]\!]\rangle \longrightarrow^* \langle m', pc', \textbf{unlabel}(\boxed{v_i'}^{\ell_i'})\rangle$$

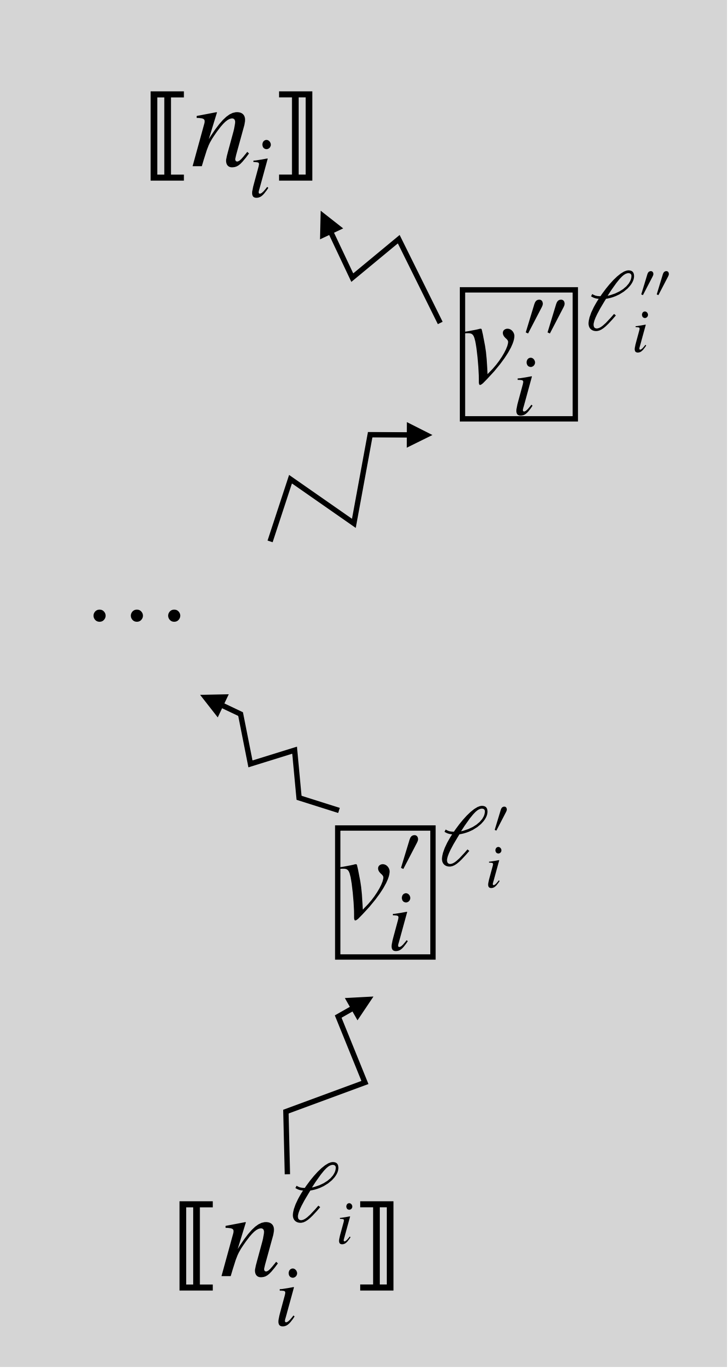
$$\ldots$$

$$\longrightarrow^* \langle m'', pc'', \textbf{unlabel}(\boxed{v_i''}^{\ell_i''})\rangle$$

$$\longrightarrow^* c'$$

Impossibility Theorem:

No translation $[\![\cdot]\!]$ satisfies Properties 1, 2, 3, 4.



Translation of $n_i^{\ell_i}$

Why this coarse-grained calculus?

What about other calculi?

- Why translate to sequential coarse-grained for PSNI?

Why this coarse-grained calculus?

What about other calculi?

- Why translate to sequential coarse-grained for PSNI?
  - To reason about taint from unlabelling

```
let _ = toLabeled(unlabel(x)) in
let _ = toLabeled(unlabel(y)) in
e
```

Example translation of fine-grained program $x + y$ (TINI)

Why this coarse-grained calculus?

What about other calculi?

- Why translate to sequential coarse-grained for PSNI?
  - To reason about taint from unlabelling
- What do we think for translating to sequential coarse-grained for TINI or concurrent coarse-grained for PSNI [Stefan et al., ICFP'12]?

```
let _ = toLabeled(unlabel(x)) in
let _ = toLabeled(unlabel(y)) in
e
```

Example translation of fine-grained program $x + y$ (TINI)

Why this coarse-grained calculus?

What about other calculi?

- Why translate to sequential coarse-grained for PSNI?
  - To reason about taint from unlabelling

```
let _ = toLabeled(unlabel(x)) in
let _ = toLabeled(unlabel(y)) in
e
```

Example translation of fine-grained program $x + y$ (TINI)

- What do we think for translating to sequential coarse-grained for TINI or concurrent coarse-grained for PSNI [Stefan et al., ICFP'12]?
  - Conjecture: Translating disjunctive precision is impossible

Cannot inspect boxed values or their labels without tainting floating-label

Why this coarse-grained calculus?

What about other calculi?

- Why translate to sequential coarse-grained for PSNI?
  - To reason about taint from unlabelling

```
let _ = toLabeled(unlabel(x)) in
let _ = toLabeled(unlabel(y)) in
e
```

Example translation of fine-grained program $x + y$ (TINI)

- What do we think for translating to sequential coarse-grained for TINI or concurrent coarse-grained for PSNI [Stefan et al., ICFP'12]?
  - Conjecture: Translating disjunctive precision is impossible
  - Translation of refinement labels may be possible*

Cannot inspect boxed values or their labels without tainting floating-label

Scope each sensitive computation by **toLabeled/fork**

* Semantics of $x \ ? \ x_1 : x_2$ makes use of disjunctive precision

Conclusion

Takeaway: Fine- and coarse-grained dynamic IFC are not equally expressive

- Coarse-grained IFC cannot do disjunctive reasoning
  - Operations specialised using fine-grained information
- Refinement labels improve the precision of fine-grained dynamic IFC
  - Main refinement example: types
  - Other possible refinements: aliasing, semantic equivalence, predicates (e.g., isEven/isOdd)

Conclusion / Future work

Conclusion

- We can mitigate traffic analysis effectively using language-level techniques
  - Language design + runtime makes enforcement more permissive
  - Type-system bounds the traffic overhead
- Fine- and coarse-grained dynamic IFC are not equally expressive
  - Coarse-grained IFC cannot do disjunctive reasoning
  - Refinement labels improve the precision of fine-grained dynamic IFC

Future work

- Traffic analysis
  - Language features not supported by OblivIO
    - Bounding leaks from features that cannot be supported natively
  - Large design space, relatively little explored
- Dynamic fine-grained precision
  - Explore techniques where fine-grained reasoning can be applied
    - Other instances where disjunctive reasoning can apply
  - Prove the impossibility conjectures for translations to TINI/Concurrent PSNI