OblivIO:

Securing reactive programs by oblivious execution with bounded traffic overheads

Jeppe Fredsgaard Blaabjerg

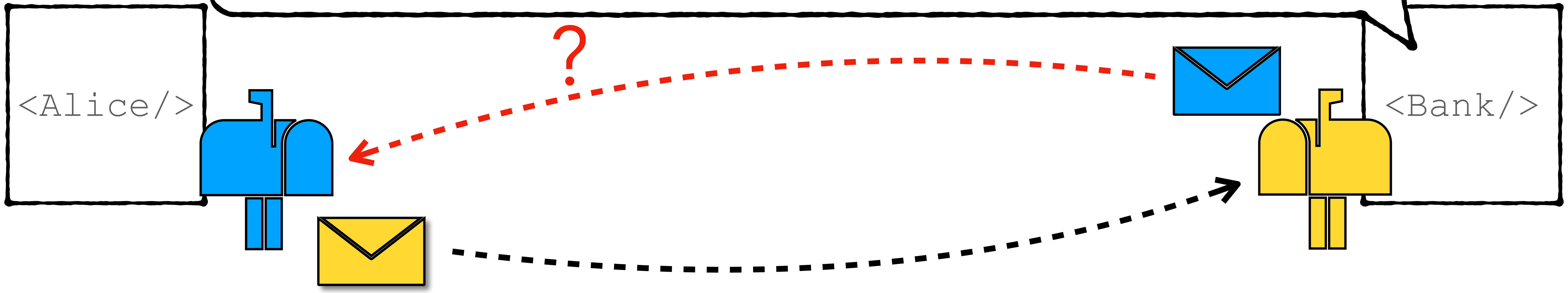
Aslan Askarov

Aarhus University

Traffic analysis

Example

```
channel ALICE/ERROR: (int*int);
var balance: int[];

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



Traffic analysis

Other observable properties of online communication

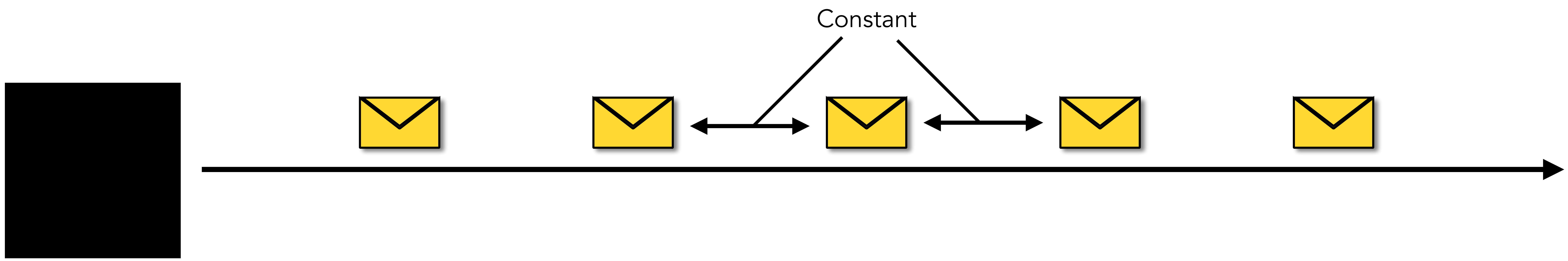
- Message timing

- Message size

- Message recipient

Mitigating traffic analysis

System-level mitigation



- Black-box
- Constant rate traffic of fixed-size packets
- 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

Example

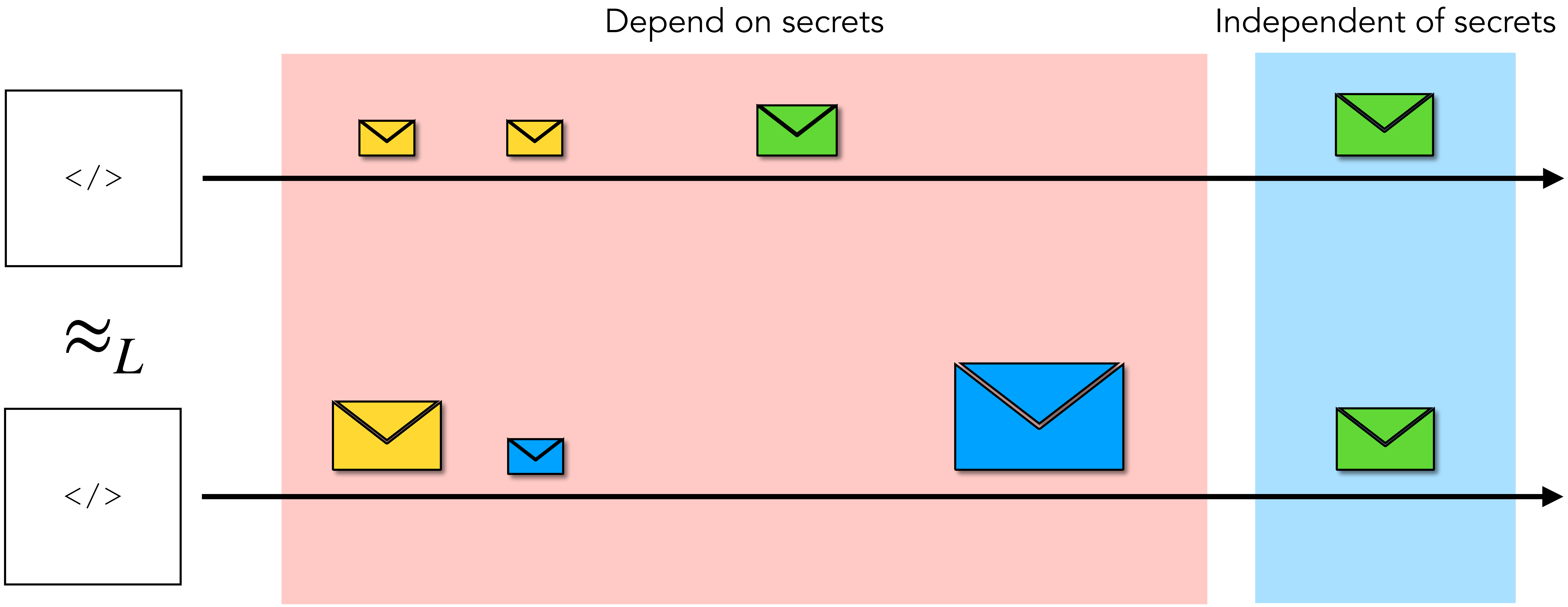
What is the right system-level bandwidth?

```
channel FORWARD: int;
var cnd: int;

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

- Traffic padding only needed if `cnd` is secret
  - Not known at system level

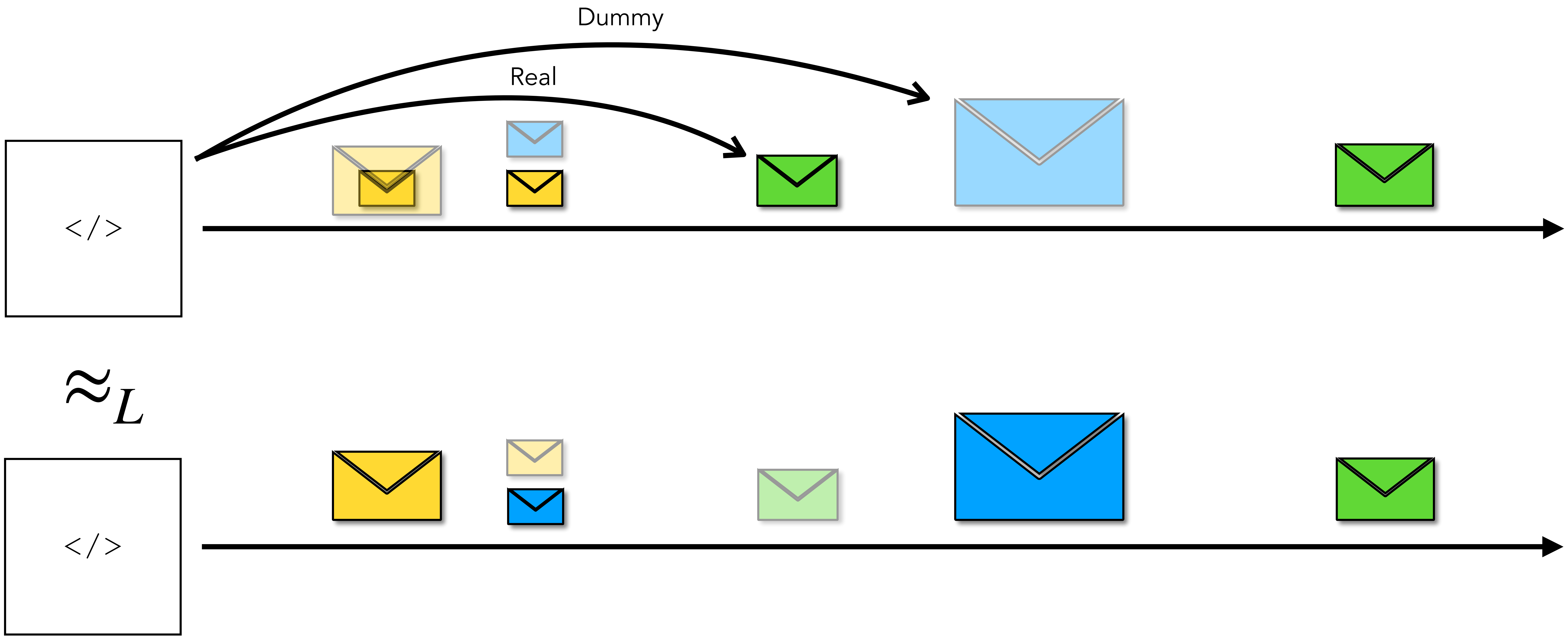
Mitigating traffic analysis

Which messages are sensitive?



Mitigating traffic analysis

Idea: Traffic padding guided by program source



OblivIO

Language and syntax

- Simple imperative language for reactive programs
- Data-oblivious execution model[2] — Control-flow is never secret
  - Execution mode *real* or *phantom* can be secret
- Formal model includes computational history for computing timestamp[3]

$$p ::= \cdot \mid ch(x)\{c\}; p$$

$$c ::= \text{skip} \mid c_1; c_2 \mid x = e \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c \mid \text{send}(ch, e)$$

$$\mid x \mathrel{?=} e \qquad \text{(* Oblivious, padding assignment *)}$$

$$\mid \text{oblif } e \text{ then } c \text{ else } c \qquad \text{(* Oblivious conditional — executes both branches *)}$$

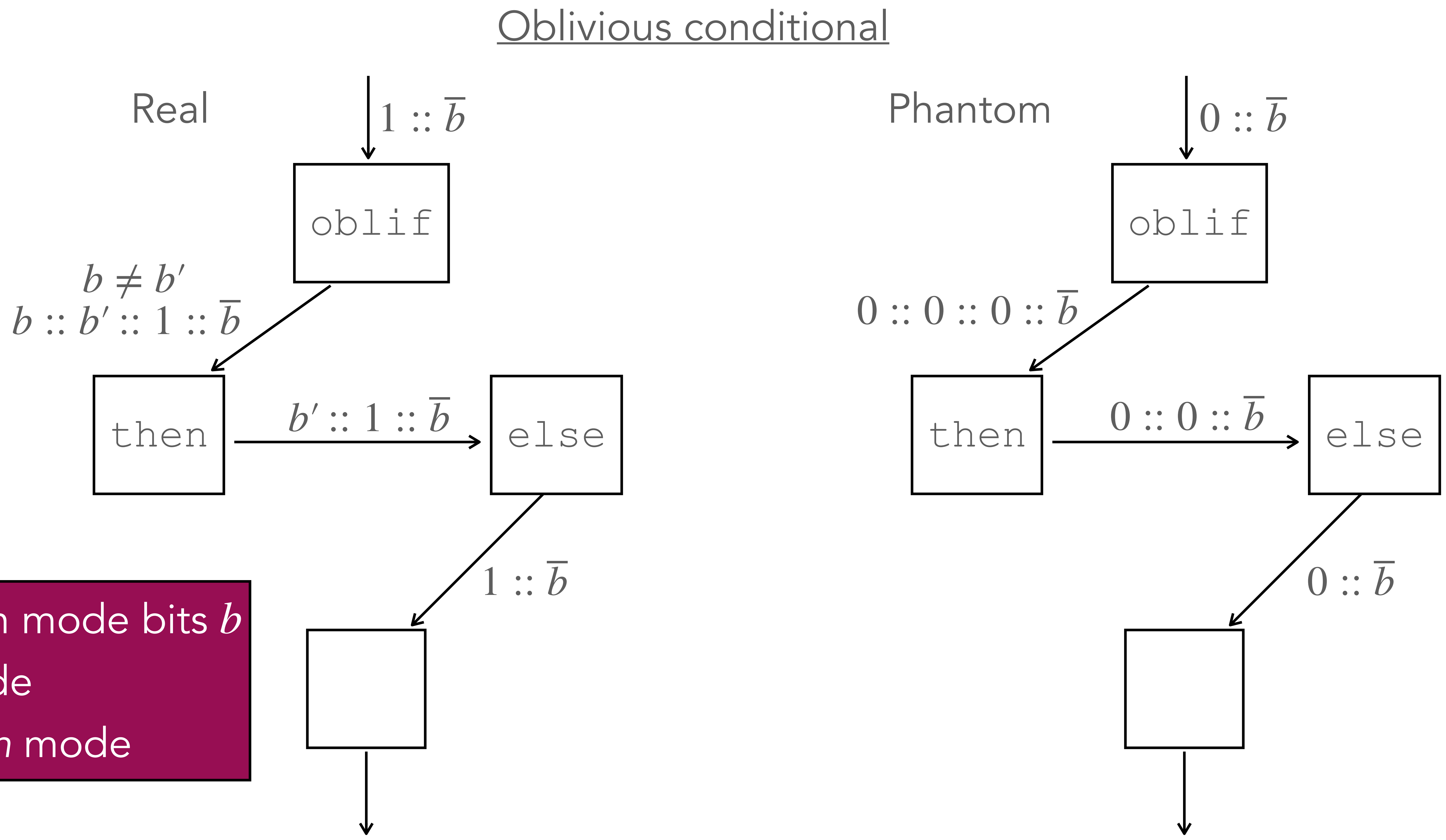
$$\mid x \mathrel{?=} \text{input}(ch, e) \qquad \text{(* Local input *)}$$

[2] 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

[3] I. Bastys, M. Balliu, T. Rezk, and A. Sabelfeld, "Clockwork: Tracking remote timing attacks," in 2020 IEEE 33rd Computer Security Foundations Symposium (CSF). IEEE, 2020, pp. 350–365.

Oblivious semantics

Control flow



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

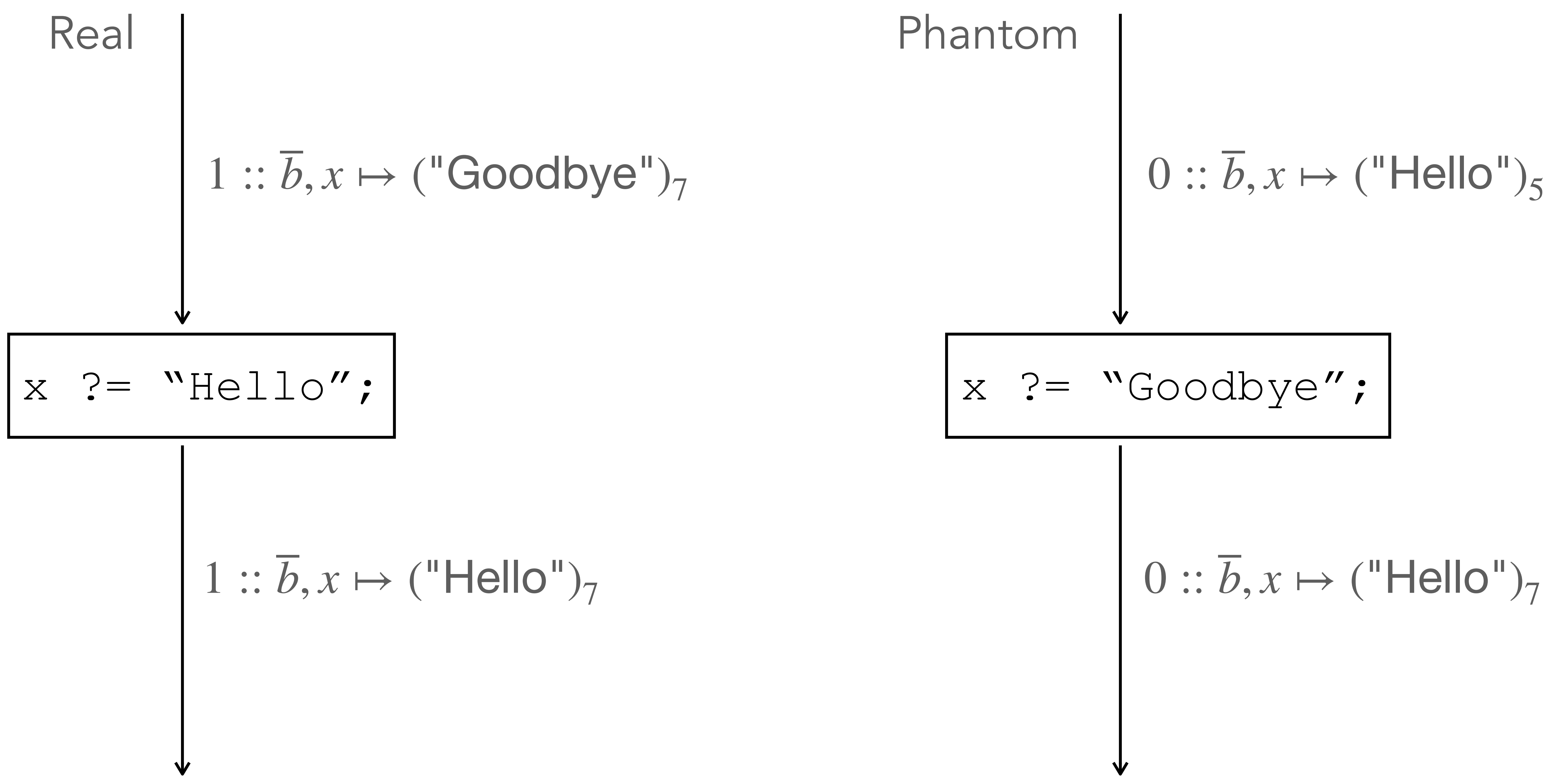
$b = 1$ denotes *real* mode

$b = 0$ denotes *phantom* mode

Oblivious semantics

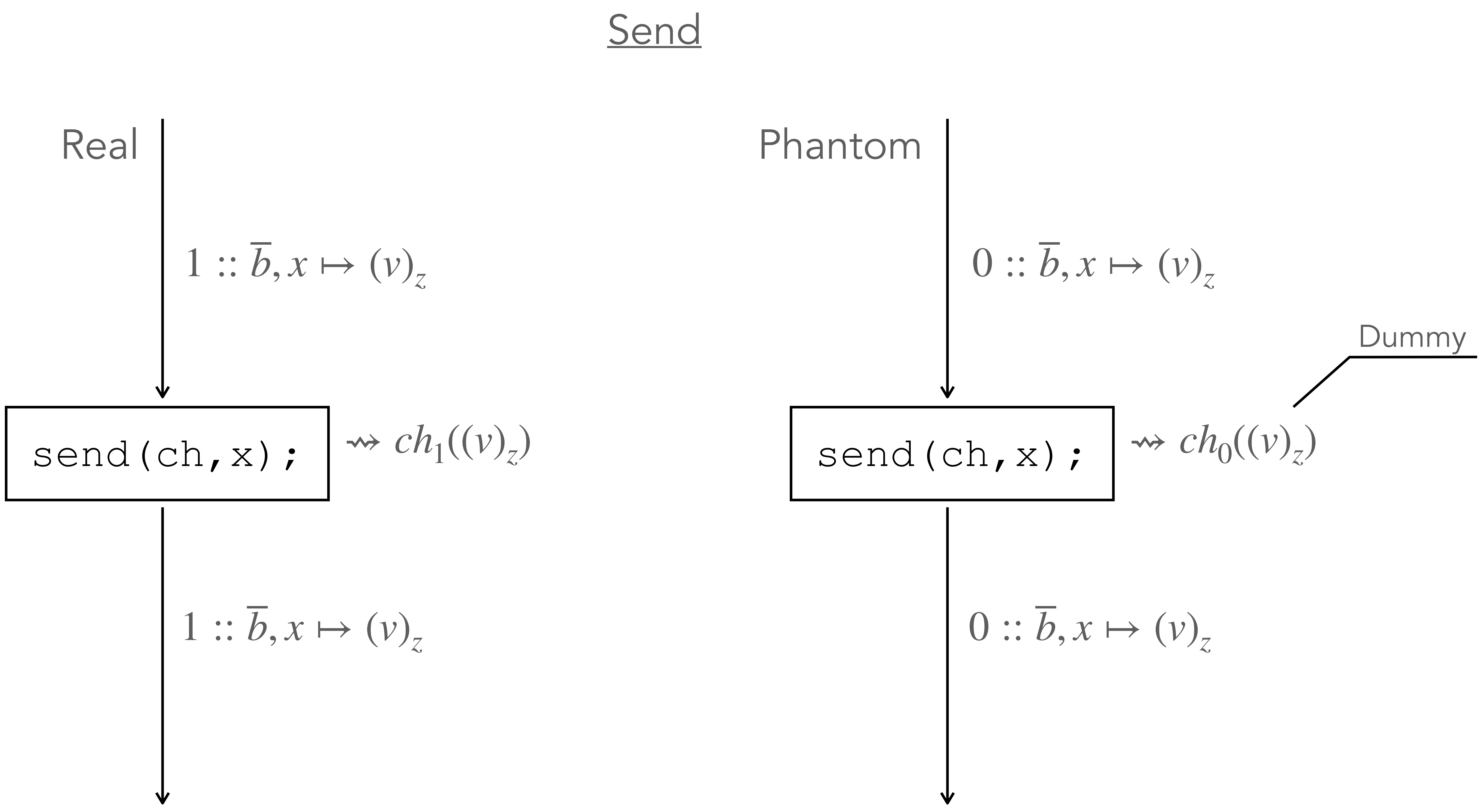
Assignment

Oblivious assignment



Oblivious semantics

Sending



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

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

Public pc

T-OblivAssign

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

Any pc

T-Send

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

Theorem

Soundness

$$k(cfg, \tau, \ell) \triangleq \{cfg_2 \mid cfg \approx_\ell cfg_2 \wedge cfg_2 \longrightarrow^*_{\tau_2} cfg_2' \wedge \tau \approx_\ell \tau_2\}$$

Attacker knowledge[4]

$$k(cfg, \tau \cdot \alpha, \ell) \supseteq k(cfg, \tau, \ell)$$

Security condition

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

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

Example

Example 1 revisited

```
channel ERROR_H: (int_H*int_H)_H;
var balance: int_H[]_H;

TRANSFER_L(from: int_H, amount: int_H, to: int_H) {
  oblif amount <= balance[from]
  then {
    balance[from] -= amount;
    balance[to] += amount;
  }
  else send(ERROR,(amount, balance[from]));
}
```

Problem

Unbounded number of dummy messages

$PING_1(1)$

Message queue

```
channel pong_H: int_H;

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

```
channel ping_H: int_H;

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

Message queue

Problem

Unbounded number of dummy messages

Message queue

```
channel pong_H: int_H;

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

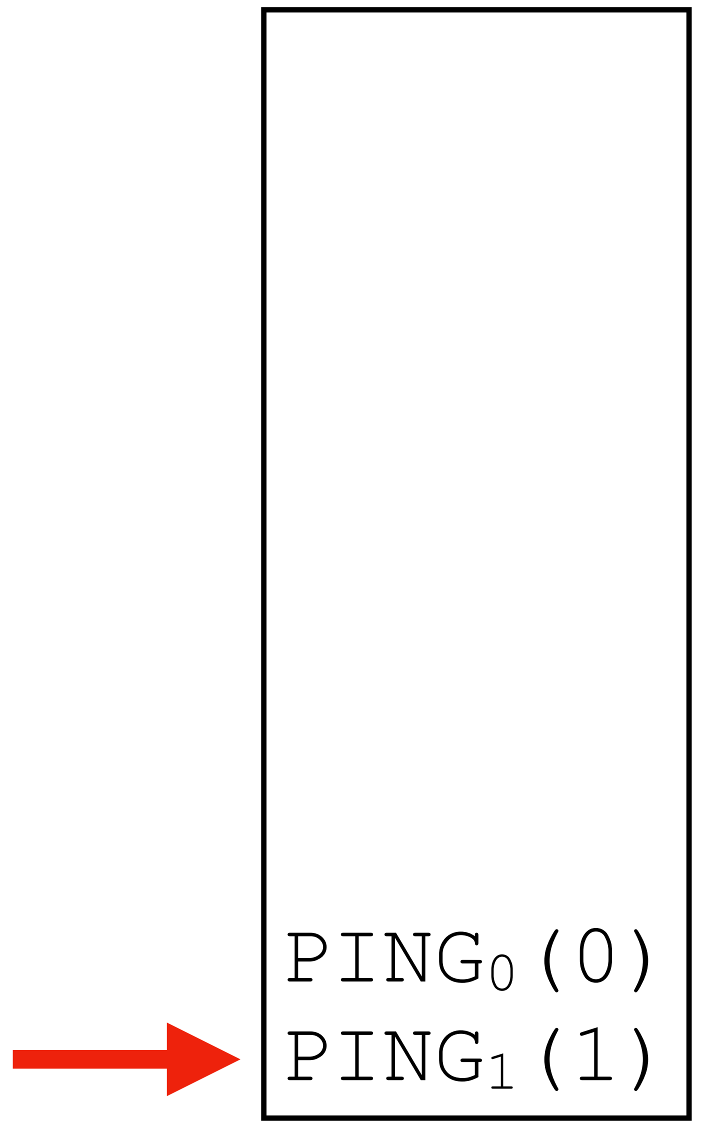
```
channel ping_H: int_H;

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

$PONG_0(0)$
$PONG_1(1)$

Message queue

Problem

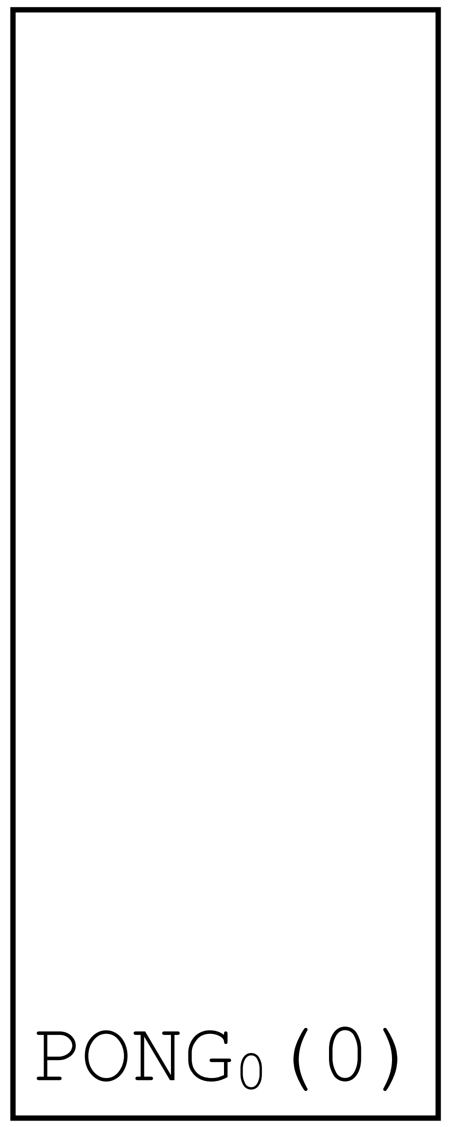
Unbounded number of dummy messages



Message queue

```
channel pongH: intH;

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

```
channel pingH: intH;

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



Message queue

Problem

Unbounded number of dummy messages

⋮
$\texttt{PING}_0$(0)
$\texttt{PING}_0$(1)
$\texttt{PING}_0$(0)
$\texttt{PING}_0$(1)
$\texttt{PING}_0$(0)
$\texttt{PING}_0$(1)
$\texttt{PING}_0$(0)
$\texttt{PING}_0$(1)
$\texttt{PING}_0$(0)
$\texttt{PING}_0$(1)

Message queue

```
channel pong_H: int_H;

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

```
channel ping_H: int_H;

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

⋮
$\texttt{PONG}_0$(1)
$\texttt{PONG}_0$(0)
$\texttt{PONG}_0$(1)
$\texttt{PONG}_0$(0)
$\texttt{PONG}_0$(1)
$\texttt{PONG}_0$(0)
$\texttt{PONG}_0$(1)
$\texttt{PONG}_0$(0)
$\texttt{PONG}_1$(1)
$\texttt{PONG}_0$(0)

Message queue

Solution

Resource tracking in type-system

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

Type system

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

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

T-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

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

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

Theorem

Overhead

- 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 2 revisited

```
channel pong_H $M: int_H;

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

```
channel ping_H $N: int_H;

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

$N=2+2*$M

$M=2+2*$N

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");
```

Summary

Mitigating traffic analysis with OblivIO

- Message presence
  - Sending dummy messages under *phantom* mode
- Message timing
  - Data-obliviousness ensuring constant-time execution
- Message size
  - Padding value size at oblivious assignments
- Message recipient
  - Channels given in program text

Conclusion

Takeaways

- OblivIO
  - Secures reactive programs by oblivious execution
    - Well-typed programs do not leak by traffic patterns (Theorem 1)
  - Bound on the traffic overhead
    - Every real message generates at most c dummy messages (Theorem 2)

Thank you!

jfblaa@cs.au.dk

Related work

Traffic analysis

- S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in 2010 IEEE Symposium on Security and Privacy. IEEE, 2010, pp. 191–206.
- G. Cherubin, J. Hayes, and M. Juárez, "Website finger-printing defenses at the application layer." Proc. Priv. Enhancing Technol., vol. 2017, no. 2, pp. 186–203, 2017.
- 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.

Related work

Constant-time execution and data-obliviousness

- 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
- C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "Oblivm: A programming framework for secure computation," in 2015 IEEE Symposium on Security and Privacy. IEEE, 2015, pp. 359–376.
- G. Barthe, B. Grégoire, and V. Laporte, "Secure compilation of side-channel countermeasures: the case of cryptographic "constant-time"," in 2018 IEEE 31st Computer Security Foundations Symposium (CSF). IEEE, 2018, pp. 328–343.
- S. Cauligi, G. Soeller, B. Johannesmeyer, F. Brown, R. S. Wahby, J. Renner, B. Grégoire, G. Barthe, R. Jhala, and D. Stefan, "Fact: a dsl for timing-sensitive computation," in Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2019, pp. 174–189.

Related work

Resource analysis

- 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.
- N. R. Krishnaswami, N. Benton, and J. Hoffmann, "Higher-order functional reactive programming in bounded space," ACM SIGPLAN Notices, vol. 47, no. 1, pp. 45–58, 2012
- M. Dehesa-Azuara, M. Fredrikson, J. Hoffmann et al., "Verifying and synthesizing constant-resource implementations with types," in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, pp. 710–728.