# Closing Internal Timing Channels by Code Transformation

Alejandro Russo[1]

(Andrei Sabelfeld[1], David Naumann[2], and John Hughes[1])

Work-in-progress!

**FOSAD ’06**

[1] *Chalmers University of Technology, Gothenborg, Sweden*

[2] *Stevens Institute of Technology, Hoboken, New Jersey, USA.*

# Language-based security

# Language-based security

- Field in computer science that deals with security related problems

# Language-based security

- Field in computer science that deals with security related problems (how?)

# Language-based security

- Field in computer science that deals with security related problems (how?)
- By analyzing the code of the program!

# Language-based security

- Field in computer science that deals with security related problems (how?)
- By analyzing the code of the program!
- In particular, we would like to guarantee the **confidentiality** of our data

# Language-based security

- Field in computer science that deals with security related problems (how?)
- By analyzing the code of the program!
- In particular, we would like to guarantee the **confidentiality** of our data
- Traditional run-time mechanisms are not enough (access control, etc)

# Language-based security

- Field in computer science that deals with security related problems (how?)
- By analyzing the code of the program!
- In particular, we would like to guarantee the **confidentiality** of our data
- Traditional run-time mechanisms are not enough (access control, etc) (why?)

# Information Flow

# Information Flow

- Our programs manage public and secret data (input/output)

# Information Flow

- Our programs manage public and secret data (input/output)
- The attacker **can only see** public output when run the program

# Information Flow

- Our programs manage public and secret data (input/output)
- The attacker **can only see** public output when run the program
- Our goal: we want programs where the attacker **cannot infer anything** about the secret data by looking the public output

# Information Flow

- Our programs manage public and secret data (input/output)
- The attacker **can only see** public output when run the program
- Our goal: we want programs where the attacker **cannot infer anything** about the secret data by looking the public output
- Those kind of programs are called non-interferent!

# Information Flow II

# Information Flow II

- How can a program reveal information to the attacker?

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h;$$

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h; \qquad\qquad h = 10 \leadsto l = 10$$

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h; \qquad h = 10 \leadsto l = 10$$
$$h = 5 \leadsto l = 5$$

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h; \qquad h = 10 \leadsto l = 10$$

$$h = 5 \leadsto l = 5 \qquad \textit{Explicit flow}$$

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h; \qquad h = 10 \leadsto l = 10$$

$$h = 5 \leadsto l = 5 \qquad \textit{Explicit flow}$$

$$\texttt{if } h > 10$$

$$\texttt{then } l := 1;$$

$$\texttt{else } l := 0;$$

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h; \qquad h = 10 \leadsto l = 10$$

$$h = 5 \leadsto l = 5 \qquad \textit{Explicit flow}$$

$$\texttt{if } h > 10 \qquad h > 10 \leadsto l = 1$$

$$\texttt{then } l := 1;$$

$$\texttt{else } l := 0;$$

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h; \qquad h = 10 \leadsto l = 10$$

$$h = 5 \leadsto l = 5 \qquad \textit{Explicit flow}$$

$$\texttt{if}\ h > 10 \qquad h > 10 \leadsto l = 1$$

$$\texttt{then}\ l := 1; \quad h \leq 10 \leadsto l = 0$$

$$\texttt{else}\ l := 0;$$

# Information Flow II

- How can a program reveal information to the attacker? (two cases)

$$l := h; \qquad h = 10 \leadsto l = 10$$

$$h = 5 \leadsto l = 5 \qquad \textit{Explicit flow}$$

$$\texttt{if}\ h > 10 \qquad h > 10 \leadsto l = 1$$

$$\texttt{then}\ l := 1; \quad h \leq 10 \leadsto l = 0 \qquad \textit{Implicit flow}$$

$$\texttt{else}\ l := 0;$$

# Internal Timing Covert Channel

- Framework: concurrent systems

# Internal Timing Covert Channel

- Framework: concurrent systems
- New covert channels are introduced (*ways to leak information*)

# Internal Timing Covert Channel

- Framework: concurrent systems
- New covert channels are introduced (*ways to leak information*)
- Number of created threads, internal timing, etc.

# Internal Timing Covert Channel

- Framework: concurrent systems
- New covert channels are introduced (*ways to leak information*)
- Number of created threads, internal timing, etc.
- Our focus: **internal timing covert channel**

# Internal Timing Covert Channel

- Framework: concurrent systems
- New covert channels are introduced (*ways to leak information*)
- Number of created threads, internal timing, etc.
- Our focus: **internal timing covert channel** (why?)

# Internal Timing Covert Channel

- Framework: concurrent systems
- New covert channels are introduced (*ways to leak information*)
- Number of created threads, internal timing, etc.
- Our focus: **internal timing covert channel** (why?)
- Motivating example: mobile devices (Geo-localization)

# Internal Timing Leak: Example

$$c_1 : \quad \texttt{if}\ h\ \texttt{then}\ \texttt{skip}; \texttt{skip}\ \texttt{else}\ \texttt{skip};$$
$$l := 1$$

$$c_2 : \quad \texttt{skip}; \texttt{skip}; l := 0$$

# Internal Timing Leak: Example

$$c_1 : \quad \texttt{if}\ h\ \texttt{then}\ \texttt{skip}; \texttt{skip}\ \texttt{else}\ \texttt{skip};$$
$$l := 1$$

$$c_2 : \quad \texttt{skip}; \texttt{skip}; l := 0$$

- Both threads are secure in isolation

# Internal Timing Leak: Example

$$c_1: \quad \texttt{if}\ h\ \texttt{then}\ \texttt{skip}; \texttt{skip}\ \texttt{else}\ \texttt{skip};$$
$$l := 1$$
$$\|$$
$$c_2: \quad \texttt{skip}; \texttt{skip}; l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)

# Internal Timing Leak: Example

$$c_1 : \quad \texttt{if}\ h\ \texttt{then}\ \texttt{skip}; \texttt{skip}\ \texttt{else}\ \texttt{skip};$$
$$l := 1$$
$$\|$$
$$c_2 : \quad \texttt{skip}; \texttt{skip}; l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):

# Internal Timing Leak: Example

$$c_1 : \ \texttt{if}\ h\ \texttt{then skip}; \texttt{skip else skip};$$
$$l := 1$$
$$\|$$
$$c_2 : \ \texttt{skip}; \texttt{skip}; l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \ \rightsquigarrow \ l$

# Internal Timing Leak: Example

$$c_1 : \ \texttt{if } h \texttt{ then skip; skip else skip;}$$
$$l := 1$$
$$\|$$
$$c_2 : \ \texttt{skip; skip; } l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \ \rightsquigarrow \ l = 1$

# Internal Timing Leak: Example

$$
\begin{array}{ll}
c_1: & \texttt{if } h \texttt{ then skip; skip else skip;} \\
& l := 1 \\
& \| \\
c_2: & \texttt{skip; skip; } l := 0
\end{array}
$$

- Both threads are secure in isolation ($c_1 \| c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \rightsquigarrow l = 1$ ( $l := 0$, then $l := 1$)

# Internal Timing Leak: Example

$$c_1: \ \texttt{if}\ h\ \texttt{then skip; skip else skip;}$$
$$l := 1$$
$$\|$$
$$c_2: \ \texttt{skip; skip;}\ l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \quad \leadsto \quad l = 1$ ( $l := 0$, then $l := 1$)
  - $h < 0 \quad \leadsto \quad l$

# Internal Timing Leak: Example

$$c_1: \quad \texttt{if } h \texttt{ then skip; skip else skip;}$$
$$l := 1$$
$$\|$$
$$c_2: \quad \texttt{skip; skip; } l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \quad \rightsquigarrow \quad l = 1$ ( $l := 0$, then $l := 1$)
  - $h < 0 \quad \rightsquigarrow \quad l = 0$

# Internal Timing Leak: Example

$$c_1: \quad \texttt{if}\ h\ \texttt{then skip; skip else skip;}$$
$$l := 1$$
$$\|$$
$$c_2: \quad \texttt{skip; skip;}\ l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \quad \rightsquigarrow \quad l = 1$ ( $l := 0$, then $l := 1$)
  - $h < 0 \quad \rightsquigarrow \quad l = 0$ ( $l := 1$, then $l := 0$)

# Internal Timing Leak: Example

$$c_1 : \quad \texttt{if } h \texttt{ then skip; skip else skip;}$$
$$l := 1$$
$$\|$$
$$c_2 : \quad \texttt{skip; skip; } l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \quad \leadsto \quad l = 1$ ( $l := 0$, then $l := 1$)
  - $h < 0 \quad \leadsto \quad l = 0$ ( $l := 1$, then $l := 0$)
- The low race is affected by the secret!

# Internal Timing Leak: Example

$$c_1: \quad \texttt{if } h \texttt{ then skip; skip else skip;}$$
$$l := 1$$
$$\|$$
$$c_2: \quad \texttt{skip; skip; } l := 0$$

- Both threads are secure in isolation ($c_1 \parallel c_2$?)
- One-step RR scheduler (starting at $c_1$):
  - $h \geq 0 \quad \rightsquigarrow \quad l = 1$ ( $l := 0$, then $l := 1$)
  - $h < 0 \quad \rightsquigarrow \quad l = 0$ ( $l := 1$, then $l := 0$)
- The low race is affected by the secret! (how?)

# Internal Timing leak: Magnified

$p := 0;$

$\texttt{while}\ n \geq 0\ \texttt{do}$

$\quad k := 2^{n-1};$

$\quad \texttt{fork}(\texttt{skip}; \texttt{skip}; l := 0);$

$\quad \texttt{if}\ h \geq k\ \texttt{then}\ \texttt{skip}; \texttt{skip}\ \texttt{else}\ \texttt{skip};$

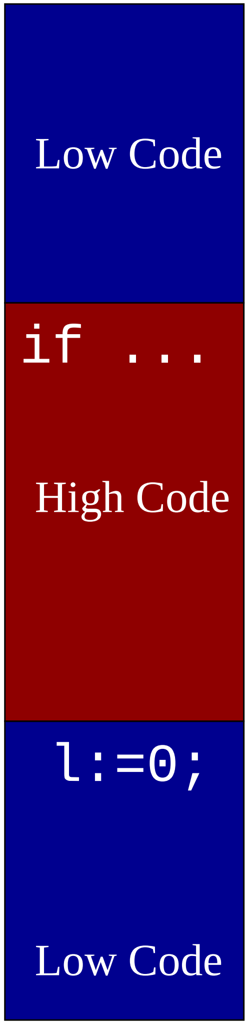
$\quad l := 1;$

$\quad \texttt{if}\ l = 1\ \texttt{then}\ h := h - k; p := p + k$

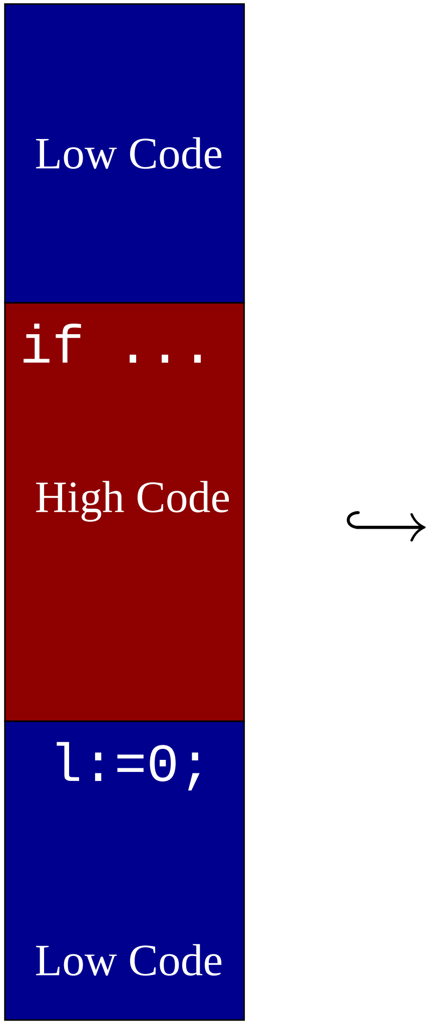
$\qquad\qquad \texttt{else}\ \texttt{skip};$

$\quad n := n - 1$

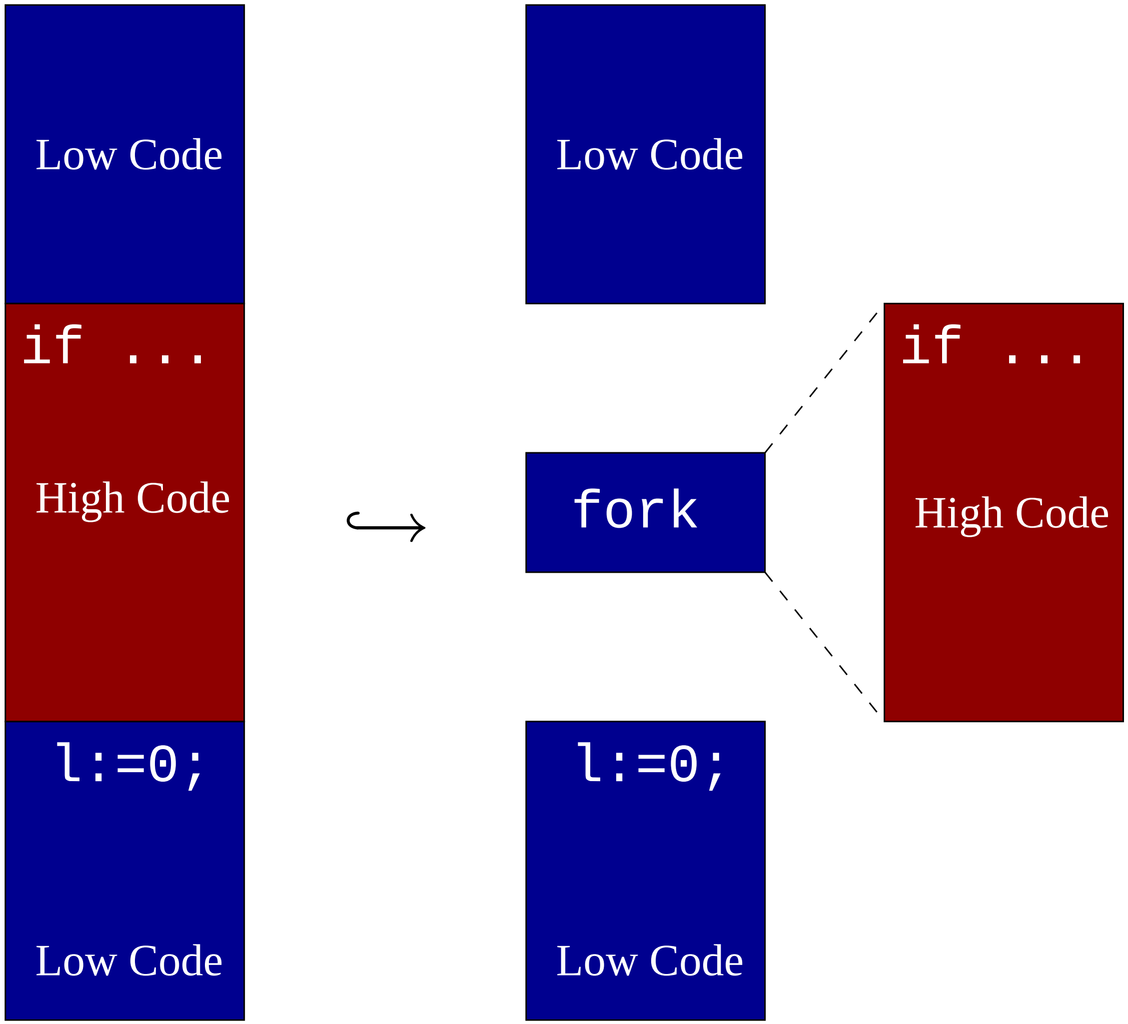
# Internal Timing Leak: Transformatio



# Internal Timing Leak: Transformatio



# Internal Timing Leak: Transformatio



# Transformation: Example I

$$c_1 : \quad \texttt{if}\ h\ \texttt{then skip; skip else skip}\ ;$$

$$l := 1$$

$$\|$$

$$c_2 : \quad \texttt{skip; skip;}\ l := 0$$

# Transformation: Example I

$$c_1 : \quad \texttt{if } h \texttt{ then skip; skip else skip } ;$$

$$l := 1$$

$$\|$$

$$c_2 : \quad \texttt{skip; skip; } l := 0$$

- Spawn high computations in dedicated threads

# Transformation: Example I

$$
\begin{aligned}
c_1: \quad & \texttt{fork}(\texttt{if}\ h\ \texttt{then}\ \texttt{skip}; \texttt{skip}\ \texttt{else}\ \texttt{skip}); \\
& l := 1 \\
& \| \\
c_2: \quad & \texttt{skip}; \texttt{skip}; l := 0
\end{aligned}
$$

- Spawn high computations in dedicated threads

# Transformation: Example I

$$c_1: \quad \texttt{fork}(\texttt{if } h \texttt{ then skip; skip else skip});$$
$$l := 1$$
$$\|$$
$$c_2: \quad \texttt{skip}; \texttt{skip}; l := 0$$

- Spawn high computations in dedicated threads
  - Good news: no internal timing leaks!

# Transformation: Example I

$$
\begin{array}{rl}
c_1: & \texttt{fork}(\texttt{if}\ h\ \texttt{then skip}; \texttt{skip else skip}); \\
 & l := 1 \\
 & \| \\
c_2: & \texttt{skip}; \texttt{skip}; l := 0
\end{array}
$$

- Spawn high computations in dedicated threads
  - Good news: no internal timing leaks!
  - Bad news: it may introduce new races between variables!

# Transformation: Example II

$\{h_2 = 0, l = 0\}$

$(\texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip}); l := 1 \parallel c_2$

# Transformation: Example II

$\{h_2 = 0, l = 0\}$

$(\texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip}); l := 1 \parallel c_2$

- Final value of $h_2 = 0$

# Transformation: Example II

$\{h_2 = 0, l = 0\}$

$(\texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip}); l := 1 \parallel c_2$

- Final value of $h_2 = 0$

$\texttt{fork}(\quad \texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip} \quad); l := 1$

$\parallel c_2$

# Transformation: Example II

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip}); l := 1 \parallel c_2$$

- Final value of $h_2 = 0$

$$\texttt{fork}(\quad \texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip} \quad); l := 1$$

$$\parallel c_2$$

- Final value of $h_2 \in \{0, 1\}$

# Transformation: Example II

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1 \parallel c_2$$

- Final value of $h_2 = 0$

$$\texttt{fork}( \quad \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip} \quad ); l := 1$$

$$\parallel c_2$$

- Final value of $h_2 \in \{0, 1\}$ (why?)

# Transformation: Example II

$\{h_2 = 0, l = 0\}$

$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1 \parallel c_2$

- Final value of $h_2 = 0$

$\texttt{fork}(\quad \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip} \quad); l := 1$

$\parallel c_2$

- Final value of $h_2 \in \{0, 1\}$ (why?) (solution?)

# Transformation: Example II

$\{h_2 = 0, l = 0\}$

$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1 \parallel c_2$

- Final value of $h_2 = 0$

$\texttt{fork}(\quad \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip} \quad); l := 1$

$\parallel c_2$

- Final value of $h_2 \in \{0, 1\}$ (why?) (solution?)
- Take snapshots of low variables when fork

# Transformation: Example II

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip}); l := 1 \parallel c_2$$

- Final value of $h_2 = 0$

$$\texttt{fork}((\lambda\hat{l}.\texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + \hat{l}; \texttt{skip}\ \texttt{else}\ \texttt{skip})@l); l := 1$$

$$\parallel c_2$$

- Final value of $h_2 \in \{0, 1\}$ (why?) (solution?)
- Take snapshots of low variables when fork

# Transformation: Example III

$\{h_2 = 0, l = 0\}$

$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$

$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$

# Transformation: Example III

$\{h_2 = 0, l = 0\}$

$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$

$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$

- Final value of $h_2 = 1$

# Transformation: Example III

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$$

$$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$$

- Final value of $h_2 = 1$

$$\texttt{fork}((\lambda \quad \hat{l}. \quad \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + \hat{l}; \texttt{skip else skip}; \quad )@ \quad l);$$
$$l := 1;$$

$$\texttt{fork}((\lambda \quad \hat{l}. \quad h_2 := h_2 + 1; \quad )@ \quad l);$$
$$l := 3 \quad \| \; c_2$$

# Transformation: Example III

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$$

$$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$$

- Final value of $h_2 = 1$

$$\texttt{fork}((\lambda \quad \hat{l}. \qquad \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + \hat{l}; \texttt{skip else skip}; \qquad )@ \quad l);$$
$$l := 1;$$

$$\texttt{fork}((\lambda \quad \hat{l}. \qquad h_2 := h_2 + 1; \qquad )@ \quad l);$$
$$l := 3 \quad \| \; c_2$$

- Final value of $h_2 \in \{1, 2\}$

# Transformation: Example III

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$$

$$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$$

- Final value of $h_2 = 1$

$$\texttt{fork}((\lambda \quad \hat{l}. \qquad \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + \hat{l}; \texttt{skip else skip}; \qquad )@ \quad l);$$
$$l := 1;$$

$$\texttt{fork}((\lambda \quad \hat{l}. \qquad h_2 := h_2 + 1; \qquad )@ \quad l);$$
$$l := 3 \quad \| \; c_2$$

- Final value of $h_2 \in \{1, 2\}$ (why?)

# Transformation: Example III

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$$
$$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$$

- Final value of $h_2 = 1$

$$\texttt{fork}((\lambda \quad \hat{l}. \qquad \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + \hat{l}; \texttt{skip else skip}; \qquad )@ \quad l);$$
$$l := 1;$$

$$\texttt{fork}((\lambda \quad \hat{l}. \qquad h_2 := h_2 + 1; \qquad )@ \quad l);$$
$$l := 3 \quad \| \; c_2$$

- Final value of $h_2 \in \{1, 2\}$ (why?) (solution?)

# Transformation: Example III

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$$

$$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$$

- Final value of $h_2 = 1$

$w := \texttt{newSem}(1); s := \texttt{newSem}(0);$

$\texttt{fork}((\lambda \hat{w}\hat{s}\hat{l}.\texttt{P}(\hat{w}); \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + \hat{l}; \texttt{skip else skip}; \texttt{V}(\hat{s}))@wsl);$

$w := s;\ l := 1;$

$\texttt{fork}((\lambda \quad \hat{l}. \qquad h_2 := h_2 + 1; \qquad )@ \quad l);$

$l := 3 \quad \| \; c_2$

- Final value of $h_2 \in \{1, 2\}$ (why?) (solution?)

# Transformation: Example III

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + l; \texttt{skip else skip}); l := 1;$$

$$h_2 := h_2 + 1; l := 3 \quad \| \; c_2$$

- Final value of $h_2 = 1$

$w := \texttt{newSem}(1); s := \texttt{newSem}(0);$

$\texttt{fork}((\lambda \hat{w}\hat{s}\hat{l}.\texttt{P}(\hat{w}); \texttt{if } h_1 \texttt{ then } h_2 := 2 * h_2 + \hat{l}; \texttt{skip else skip}; \texttt{V}(\hat{s}))@wsl);$

$w := s;\ l := 1;$

$s := \texttt{newSem}(0);$

$\texttt{fork}((\lambda \hat{w}\hat{s}\hat{l}.\texttt{P}(\hat{w}); h_2 := h_2 + 1; \texttt{V}(\hat{s}))@wsl);$

$w := s;\ l := 3 \quad \| \; c_2$

- Final value of $h_2 \in \{1, 2\}$ (why?) (solution?)

# Transformation: Example III

$$\{h_2 = 0, l = 0\}$$

$$(\texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + l; \texttt{skip}\ \texttt{else}\ \texttt{skip}); l := 1;$$

$$h_2 := h_2 + 1; l := 3 \quad \| \ c_2$$

- Final value of $h_2 = 1$

$w := \texttt{newSem}(1); s := \texttt{newSem}(0);$

$\texttt{fork}((\lambda \hat{w}\hat{s}\hat{l}.\texttt{P}(\hat{w}); \texttt{if}\ h_1\ \texttt{then}\ h_2 := 2 * h_2 + \hat{l}; \texttt{skip}\ \texttt{else}\ \texttt{skip}; \texttt{V}(\hat{s}))@wsl);$

$w := s;\ l := 1;$

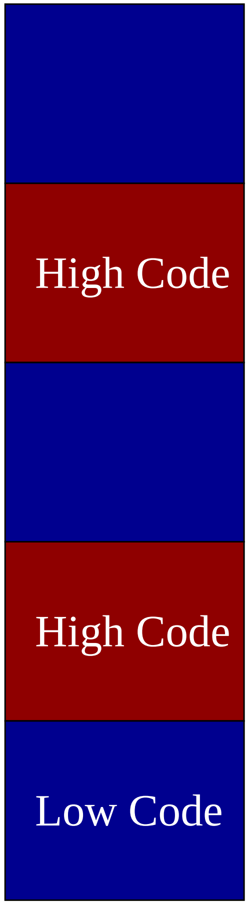
$s := \texttt{newSem}(0);$

$\texttt{fork}((\lambda \hat{w}\hat{s}\hat{l}.\texttt{P}(\hat{w}); h_2 := h_2 + 1; \texttt{V}(\hat{s}))@wsl);$

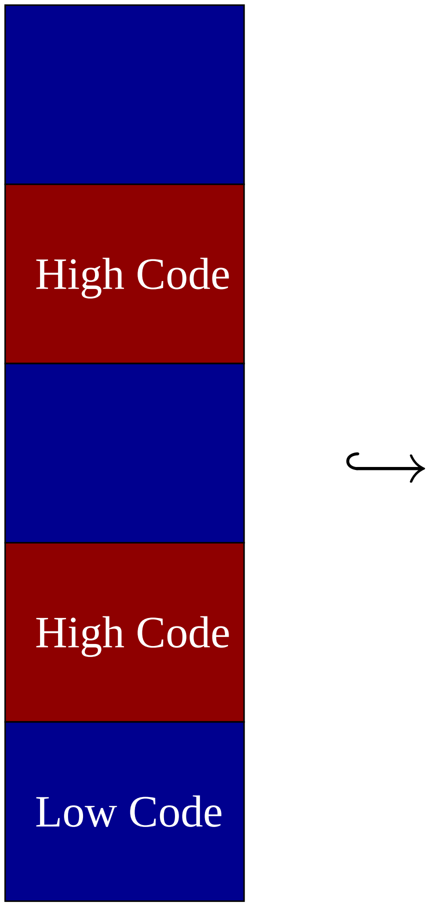
$w := s;\ l := 3 \quad \| \ c_2$

- Final value of $h_2 \in \{1, 2\}$ (why?) (solution?)
- Synchronize the spawned dedicated threads

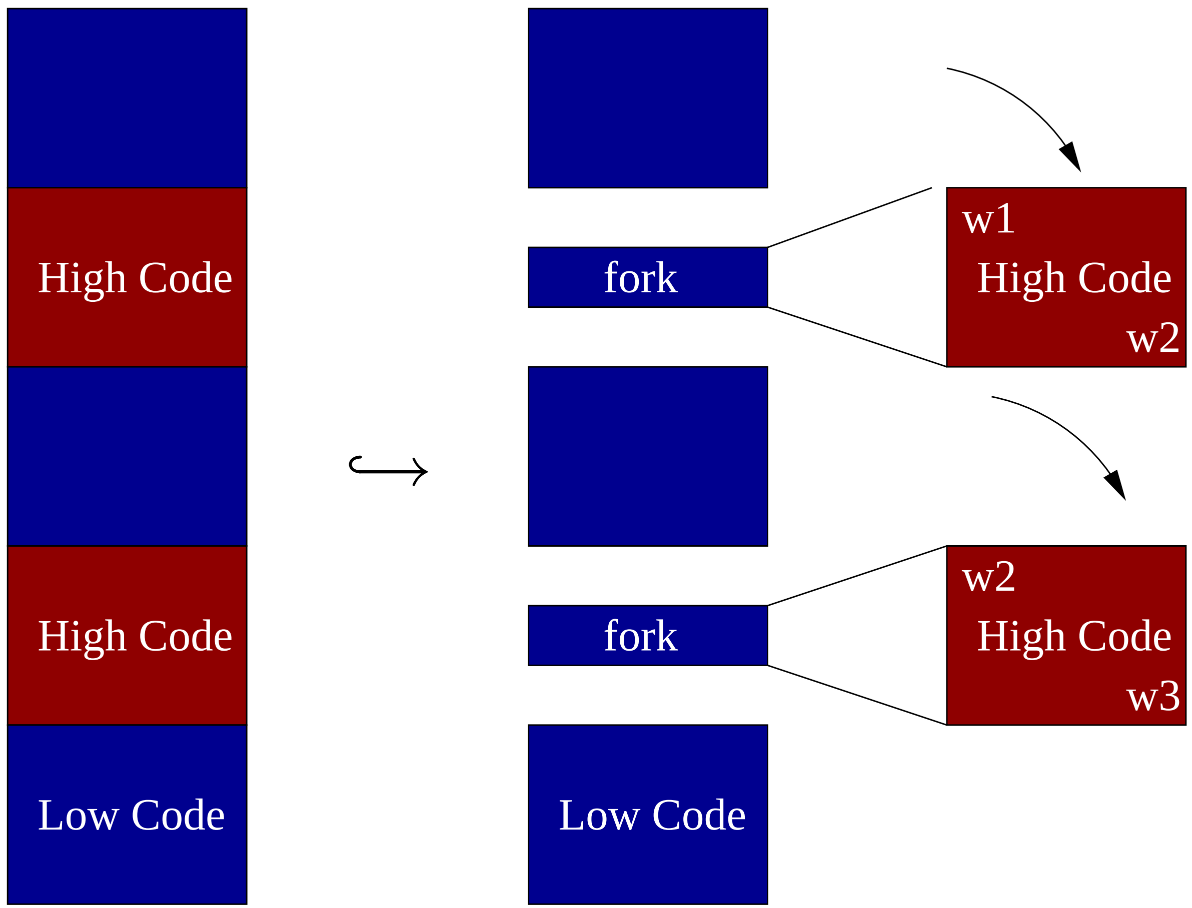
# Transformation: Example III



# Transformation: Example III



# Transformation: Example III



# Results... but technically

# Results... but technically

- **Security**: If $\Gamma \vdash c \hookrightarrow_t c'$ then $c'$ is secure under round-robin scheduling.

# Results... but technically

- **Security**: If $\Gamma \vdash c \hookrightarrow_t c'$ then $c'$ is secure under round-robin scheduling.
- **Refinement**: Suppose $\Gamma \vdash c \hookrightarrow_t c'$ and $g'_1$ and $g'_2$ are global memories for $c'$ such that $(c', g'_1) \Downarrow g'_2$ using the nondeterministic scheduler $ND$. Let $g_1$ and $g_2$ be the restrictions of $g'_1$ and $g'_2$ to the globals of $c$. Then $(c, g_1) \Downarrow g_2$ using $ND$.

# To sum up...

- Transformation that closes internal timing channels
- Dynamic thread creation in the source language
- No need to change the environment (schedulers, etc)
- Transformation only reject programs with illegal flows inherent to sequential computations