

Closing Internal Timing Channels by Code Transformation

Alejandro Russo¹

(Andrei Sabelfeld¹, David Naumann², and John Hughes¹)

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**)

$$l := h; \quad h = 10 \rightsquigarrow l = 10$$

Information Flow II

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

$$l := h; \quad h = 10 \rightsquigarrow l = 10$$

$$h = 5 \rightsquigarrow l = 5$$

Information Flow II

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

$l := h; \quad h = 10 \rightsquigarrow l = 10$

$h = 5 \rightsquigarrow l = 5 \quad \textit{Explicit flow}$

Information Flow II

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

$l := h;$ $h = 10 \rightsquigarrow l = 10$

$h = 5 \rightsquigarrow l = 5$ *Explicit flow*

```
if  $h > 10$ 
then  $l := 1;$ 
else  $l := 0;$ 
```

Information Flow II

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

$l := h; \quad h = 10 \rightsquigarrow l = 10$

$h = 5 \rightsquigarrow l = 5$ *Explicit flow*

if $h > 10$ $h > 10 \rightsquigarrow l = 1$

then $l := 1;$

else $l := 0;$

Information Flow II

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

$l := h;$ $h = 10 \rightsquigarrow l = 10$

$h = 5 \rightsquigarrow l = 5$ *Explicit flow*

if $h > 10$ $h > 10 \rightsquigarrow l = 1$

then $l := 1;$ $h \leq 10 \rightsquigarrow l = 0$

else $l := 0;$

Information Flow II

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

$l := h;$ $h = 10 \rightsquigarrow l = 10$

$h = 5 \rightsquigarrow l = 5$ *Explicit flow*

if $h > 10$ $h > 10 \rightsquigarrow l = 1$

then $l := 1;$ $h \leq 10 \rightsquigarrow l = 0$ *Implicit flow*

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 : if h then skip; skip else skip;
 $l := 1$

c_2 : skip; skip; $l := 0$

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;
 $l := 1$

c_2 : skip; skip; $l := 0$

- Both threads are secure in isolation

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; 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 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : 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

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing Leak: Example

c_1 : if h then skip; skip else skip;

$l := 1$

||

c_2 : skip; skip; $l := 0$

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

Internal Timing leak: Magnified

```
 $p := 0;$   
while  $n \geq 0$  do  
     $k := 2^{n-1};$   
    fork(skip; skip;  $l := 0$ );  
    if  $h \geq k$  then skip; skip else skip;  
     $l := 1;$   
    if  $l = 1$  then  $h := h - k; p := p + k$   
        else skip;  
     $n := n - 1$ 
```

Internal Timing Leak: Transformation

Low Code

```
if ...
```

High Code

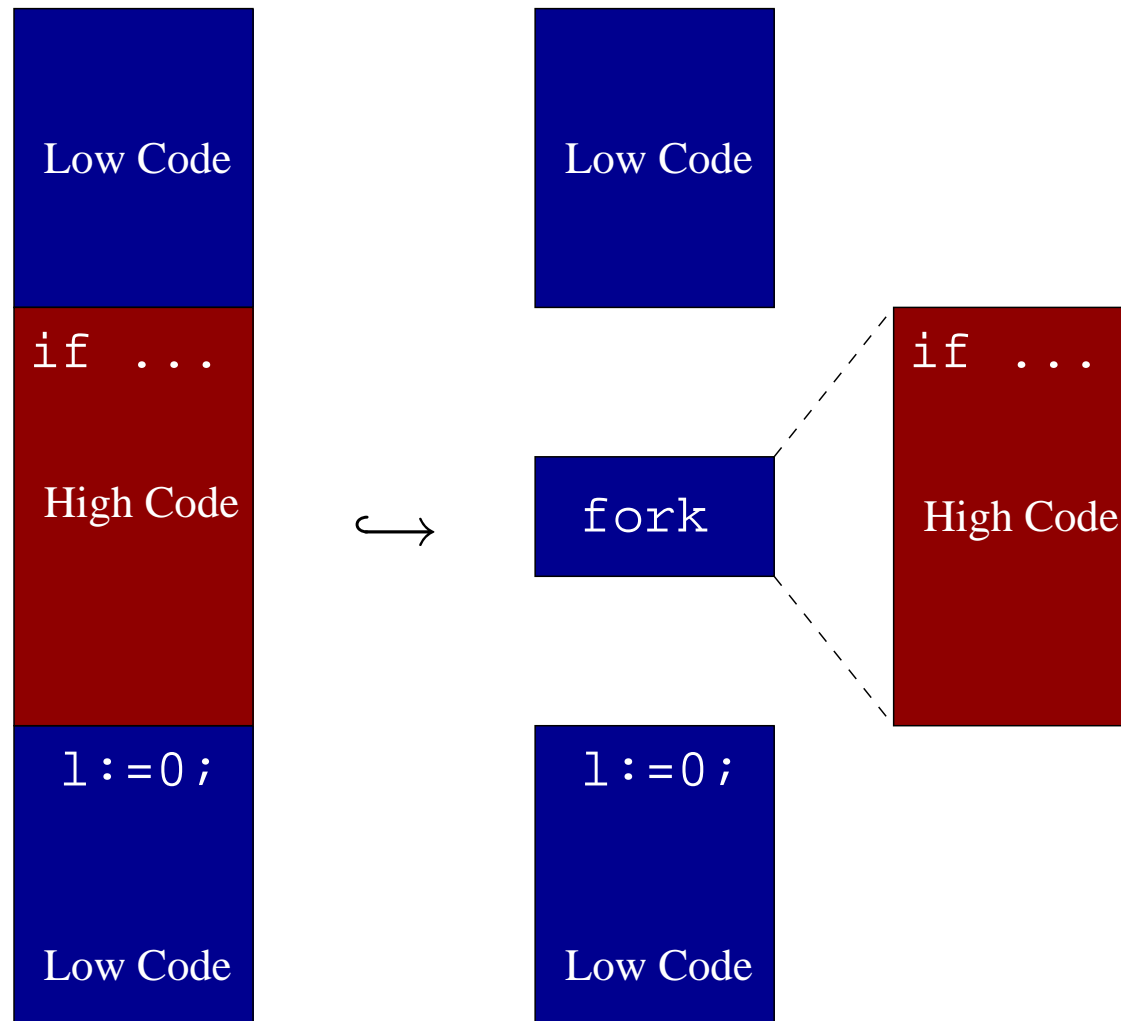
```
l := 0 ;
```

Low Code

Internal Timing Leak: Transformation



Internal Timing Leak: Transformation



Transformation: Example I

c_1 : if h then skip; skip else skip ;
 $l := 1$
 ||
 c_2 : skip; skip; $l := 0$

Transformation: Example I

c_1 : if h then skip; skip else skip ;
 $l := 1$
 ||
 c_2 : skip; skip; $l := 0$

- Spawn high computations in dedicated threads

Transformation: Example I

```
 $c_1$  : fork(if  $h$  then skip; skip else skip);  
       $l := 1$   
      ||  
 $c_2$  : skip; skip;  $l := 0$ 
```

- Spawn high computations in dedicated threads

Transformation: Example I

```
 $c_1$  : fork(if  $h$  then skip; skip else skip);  
       $l := 1$   
      ||  
 $c_2$  : skip; skip;  $l := 0$ 
```

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

Transformation: Example I

```
 $c_1$  : fork(if  $h$  then skip; skip else skip);  
       $l := 1$   
      ||  
 $c_2$  : skip; skip;  $l := 0$ 
```

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

Transformation: Example II

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

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

Transformation: Example II

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

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

- Final value of $h_2 = 0$

Transformation: Example II

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

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

- Final value of $h_2 = 0$

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

Transformation: Example II

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

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

- Final value of $h_2 = 0$

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

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

Transformation: Example II

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

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

- Final value of $h_2 = 0$

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

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

Transformation: Example II

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

(if h_1 then $h_2 := 2 * h_2 + l$; skip else skip); $l := 1$ || c_2

- Final value of $h_2 = 0$

fork(if h_1 then $h_2 := 2 * h_2 + l$; skip else skip); $l := 1$
|| c_2

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

Transformation: Example II

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

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

- Final value of $h_2 = 0$

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

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

Transformation: Example II

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

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

- Final value of $h_2 = 0$

$\text{fork}((\lambda \hat{l}. \text{if } h_1 \text{ then } h_2 := 2 * h_2 + \hat{l}; \text{skip else skip})@l); l := 1$
 $\parallel c_2$

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

Transformation: Example III

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

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

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

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

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

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

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

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

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

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

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

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

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

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

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

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

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

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

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

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

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

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

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

$l := 3$ || c_2

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

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

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

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

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

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

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

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

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

Transformation: Example III

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

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

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

- Final value of $h_2 = 1$

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

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

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

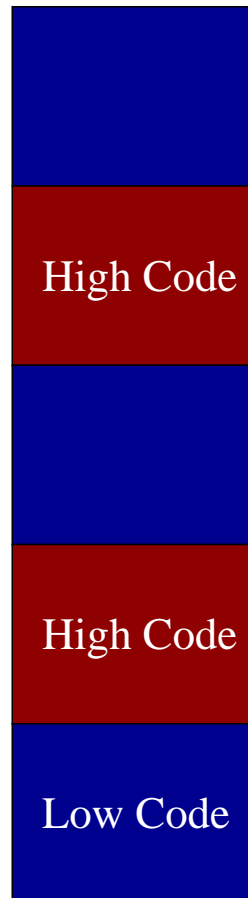
$s := \text{newSem}(0)$;

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

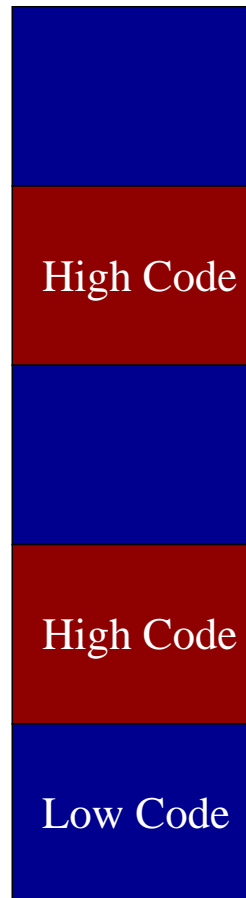
$w := s$; $l := 3$ || c_2

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

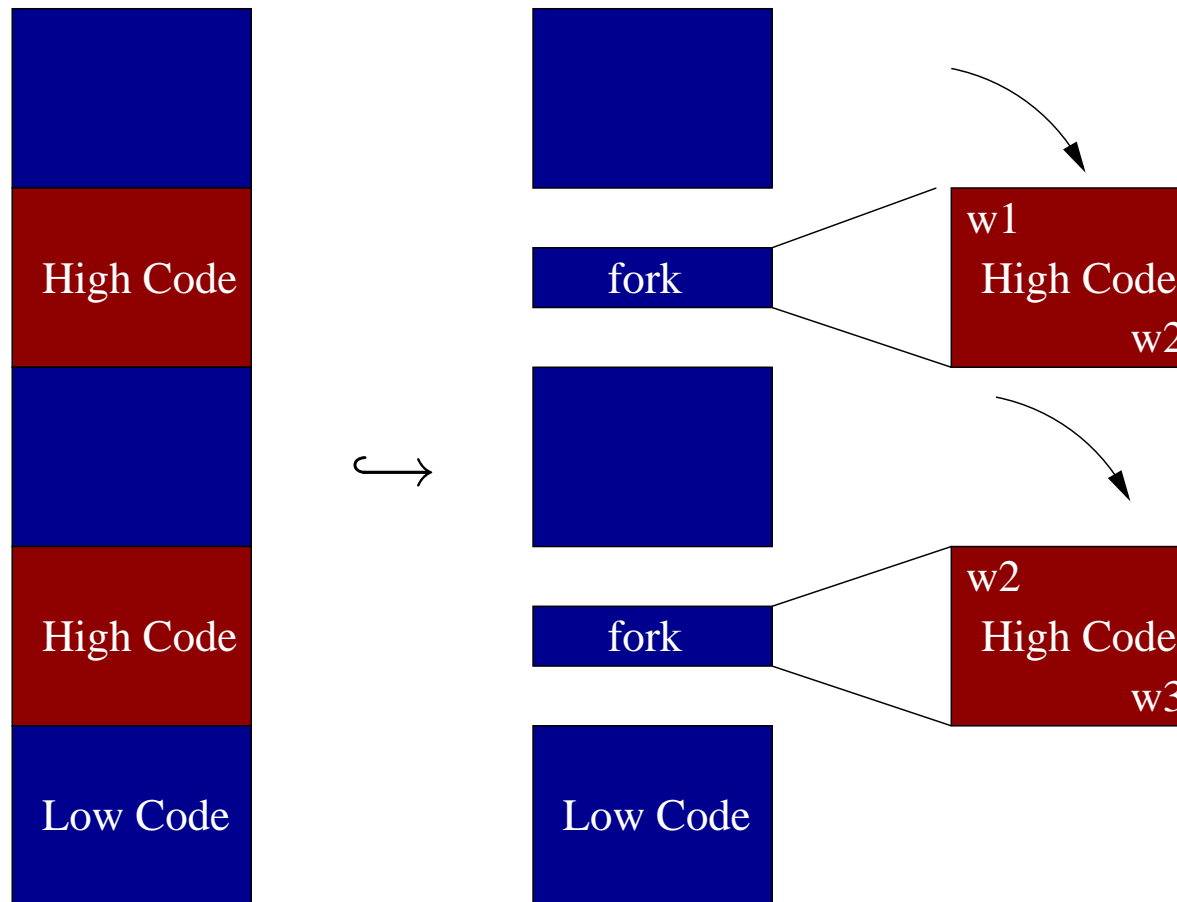
Transformation: Example III



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