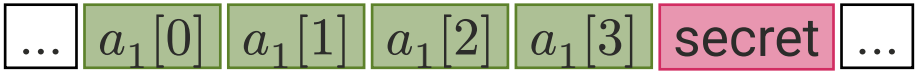
Towards More Efficient and Trustworthy Formally Secure Compilation Against Speculative Side-Channel Attacks

Jonathan Baumann

Supervised by Cătălin Hriţcu

Formally Verified Security group, MPI-SP, Germany

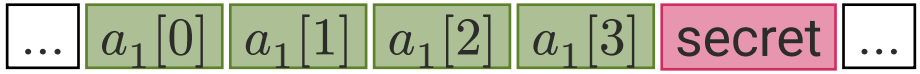
Spectre (v1)




```
if i < size(a1) then
    j ← a1[i];
    x ← a2[j]

else
    ...
```

Spectre (v1)

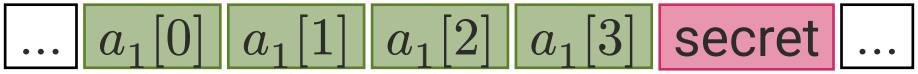



```
if i < size(a1) then
  j ← a1[i];
  x ← a2[j]

else
  ...
```

👀 $\mathtt{i} < size(\mathtt{a}_1)$

Spectre (v1)



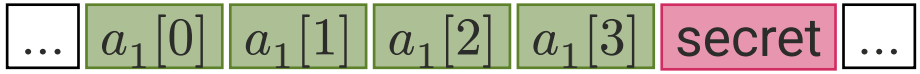

```
if i < size(a1) then
  j ← a1[i];
  x ← a2[j]

else
  ...
```

👀 $\mathtt{i} < size(\mathtt{a}_1)$

👀 $\mathtt{i}$

Spectre (v1)




```
if i < size(a1) then
  j ← a1[i];
  x ← a2[j]

else
  ...
```

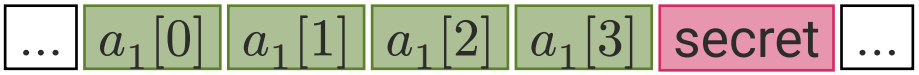
👀 $\mathtt{i} < size(\mathtt{a}_1)$

👀 $\mathtt{i}$

👀 $\mathtt{a}_1[\mathtt{i}]$

Spectre (v1)

let $\mathtt{i} = 4$

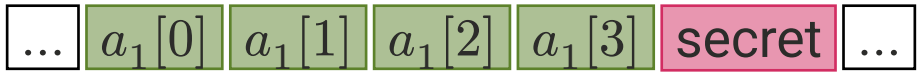



```
if i < size(a1) then
    j ← a1[i];
    x ← a2[j]

else
    ...
```

Spectre (v1)

let $\mathtt{i} = 4$

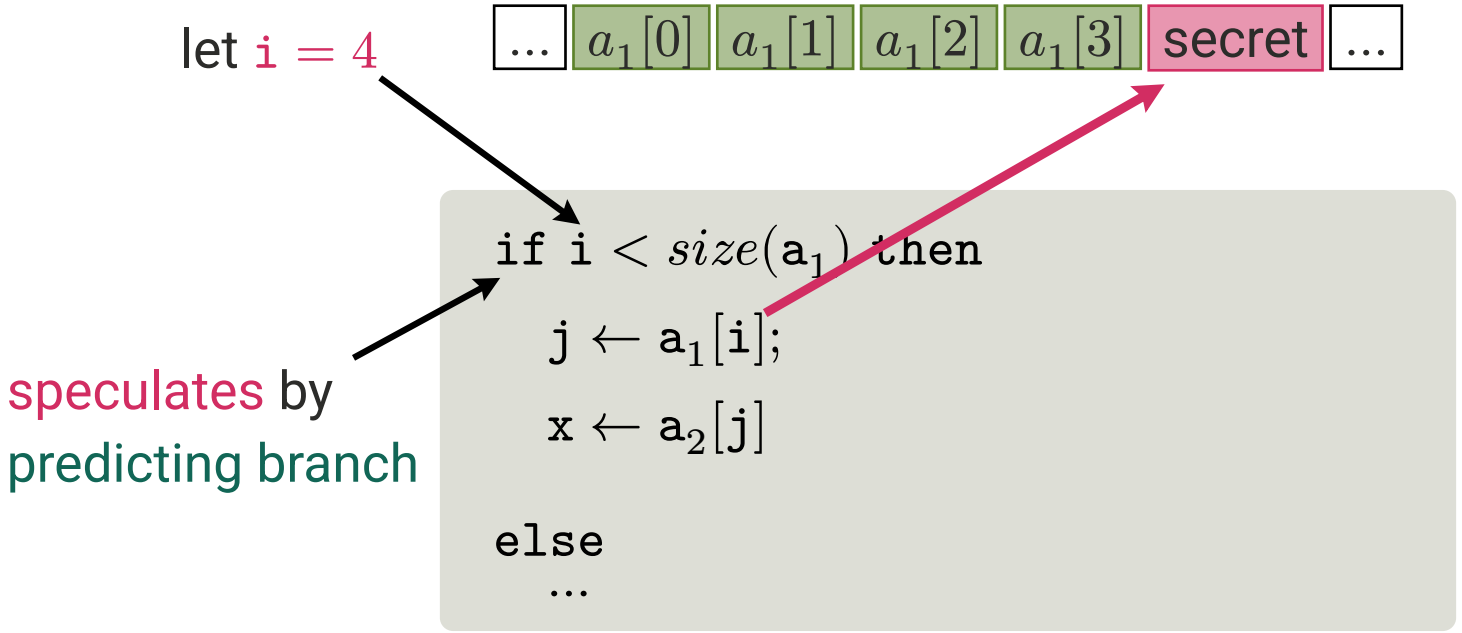


speculates by predicting branch

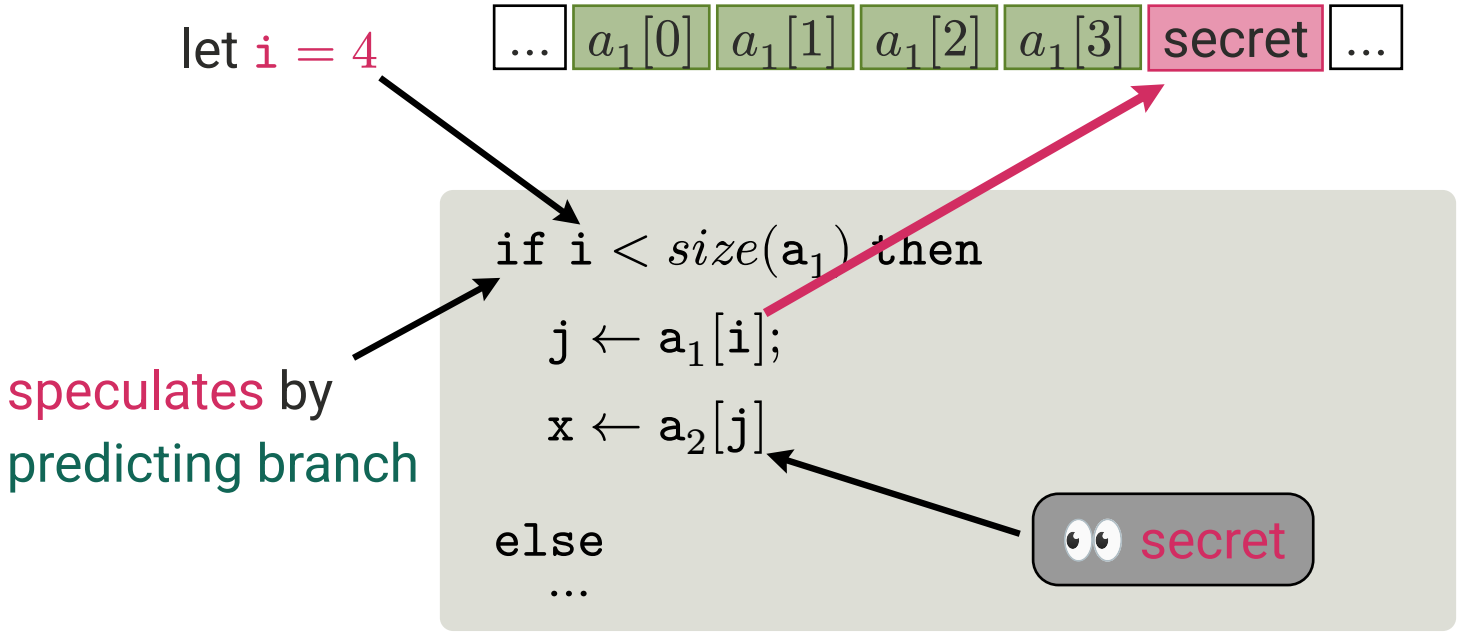
```
if i < size(a1) then
    j ← a1[i];
    x ← a2[j]

else
    ...
```

Spectre (v1)


let i = 4
... $a_1[0]$ $a_1[1]$ $a_1[2]$ $a_1[3]$ secret ...
if i < $size(a_1)$ then
j ← a_1[i];
x ← a_2[j]
else
...
speculates by
predicting branch

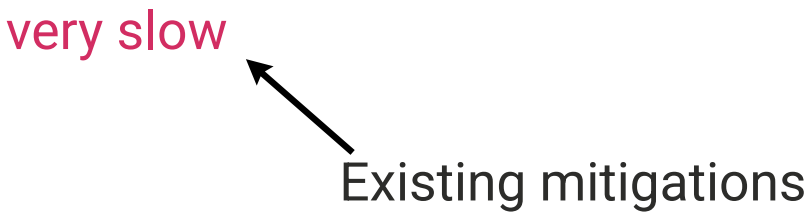
Spectre (v1)


let i = 4
... $a_1[0]$ $a_1[1]$ $a_1[2]$ $a_1[3]$ secret ...
if i < $size(a_1)$ then
j ← a_1[i];
x ← a_2[j]
else
...
speculates by
predicting branch
secret

Previous Spectre Mitigations

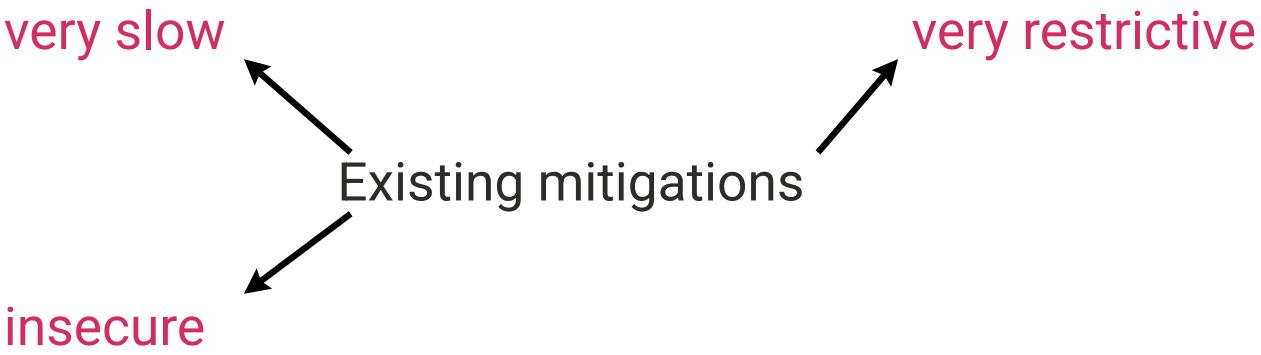
Existing mitigations

Previous Spectre Mitigations

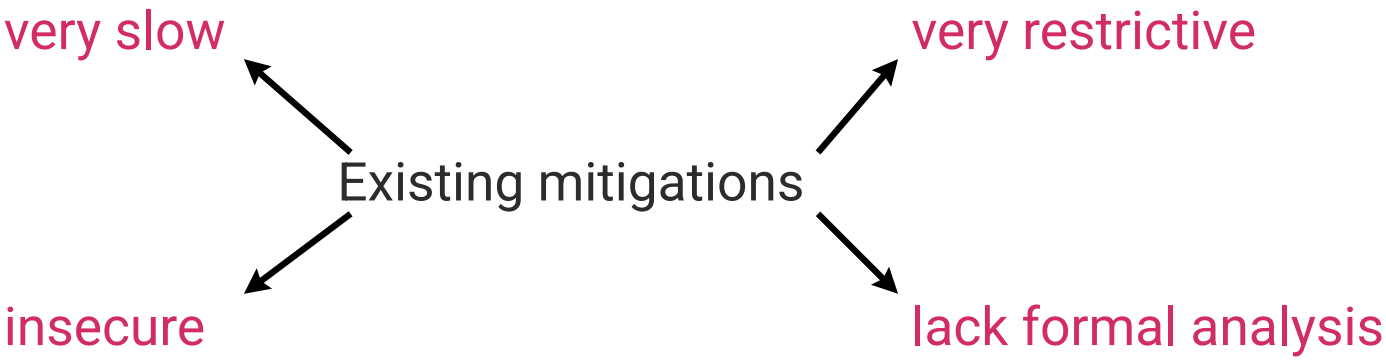


Previous Spectre Mitigations

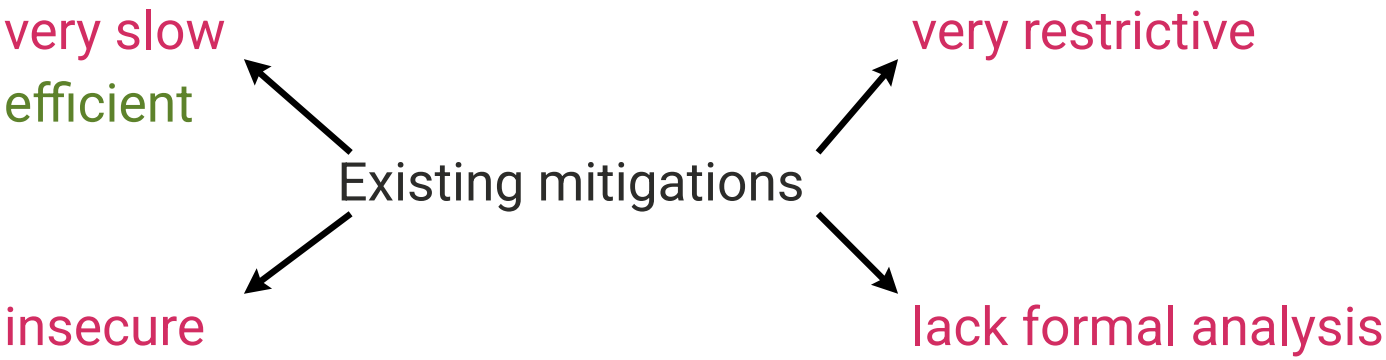
Previous Spectre Mitigations



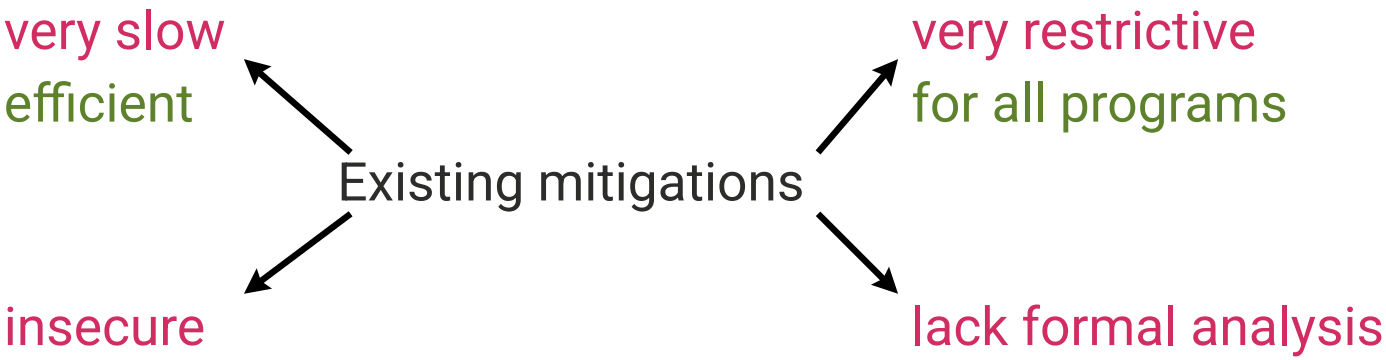
Previous Spectre Mitigations



Previous Spectre Mitigations

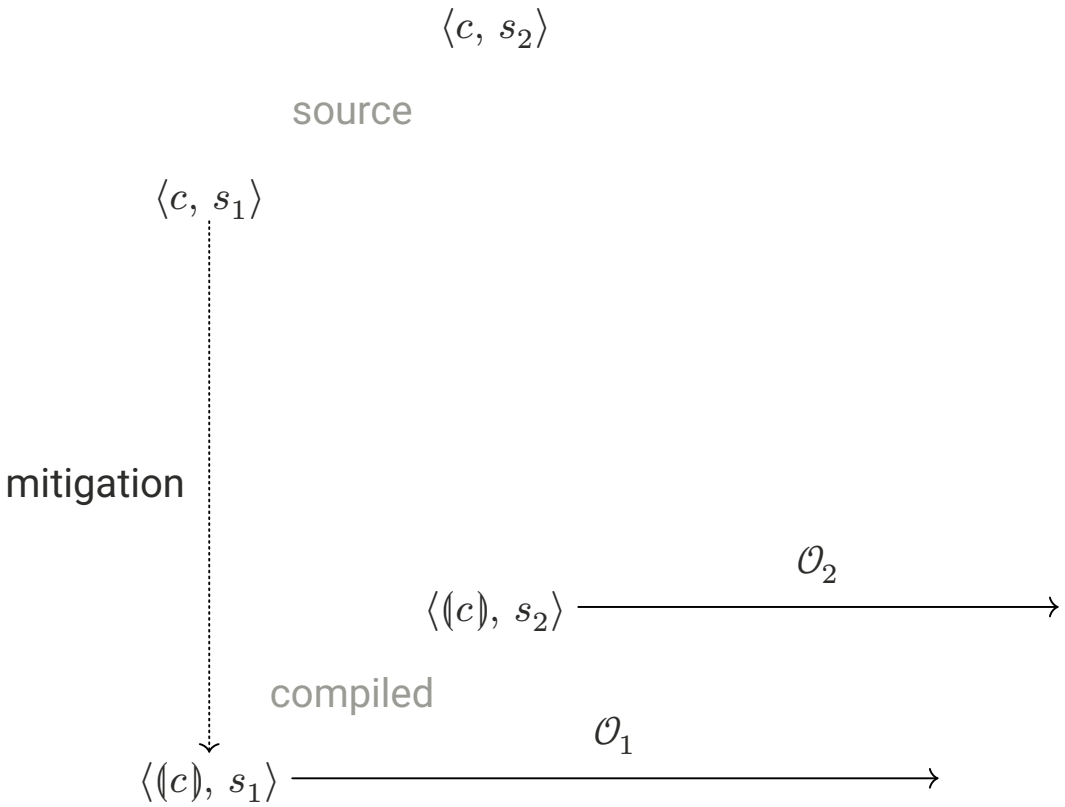


Previous Spectre Mitigations

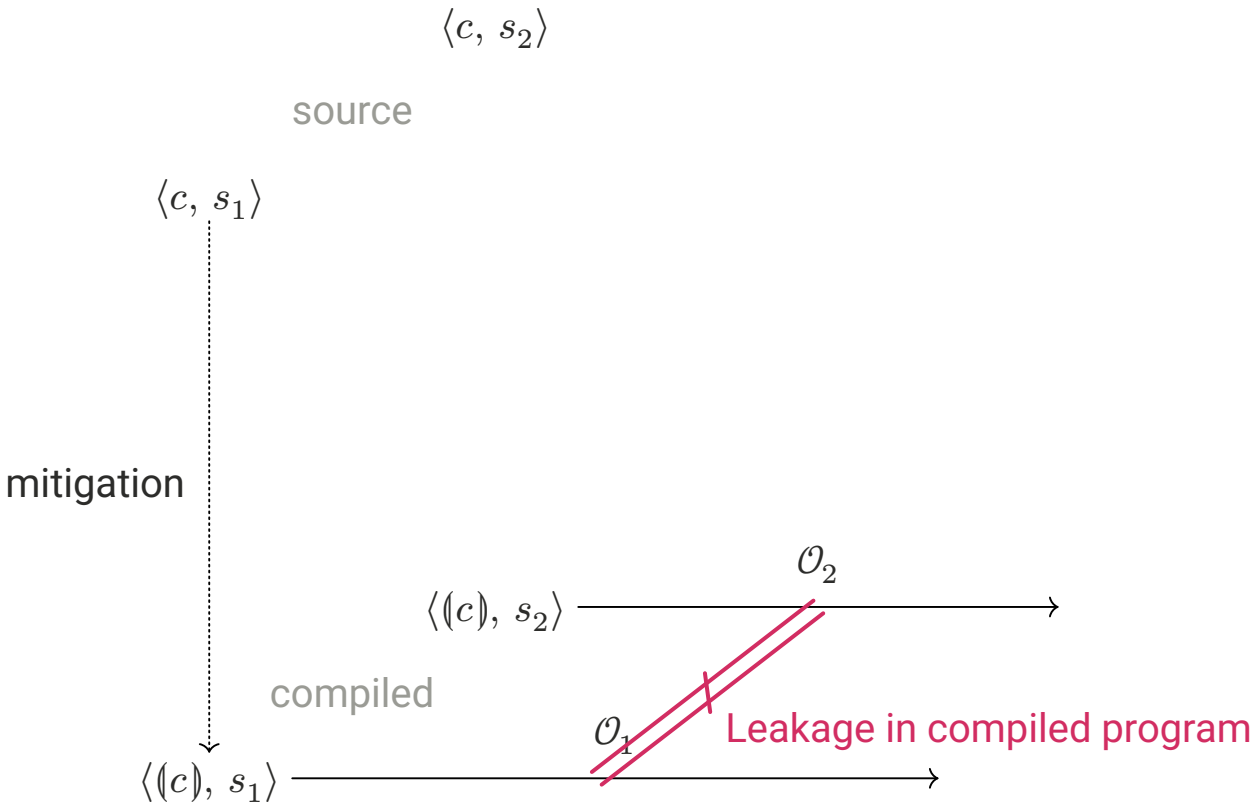


Previous Spectre Mitigations

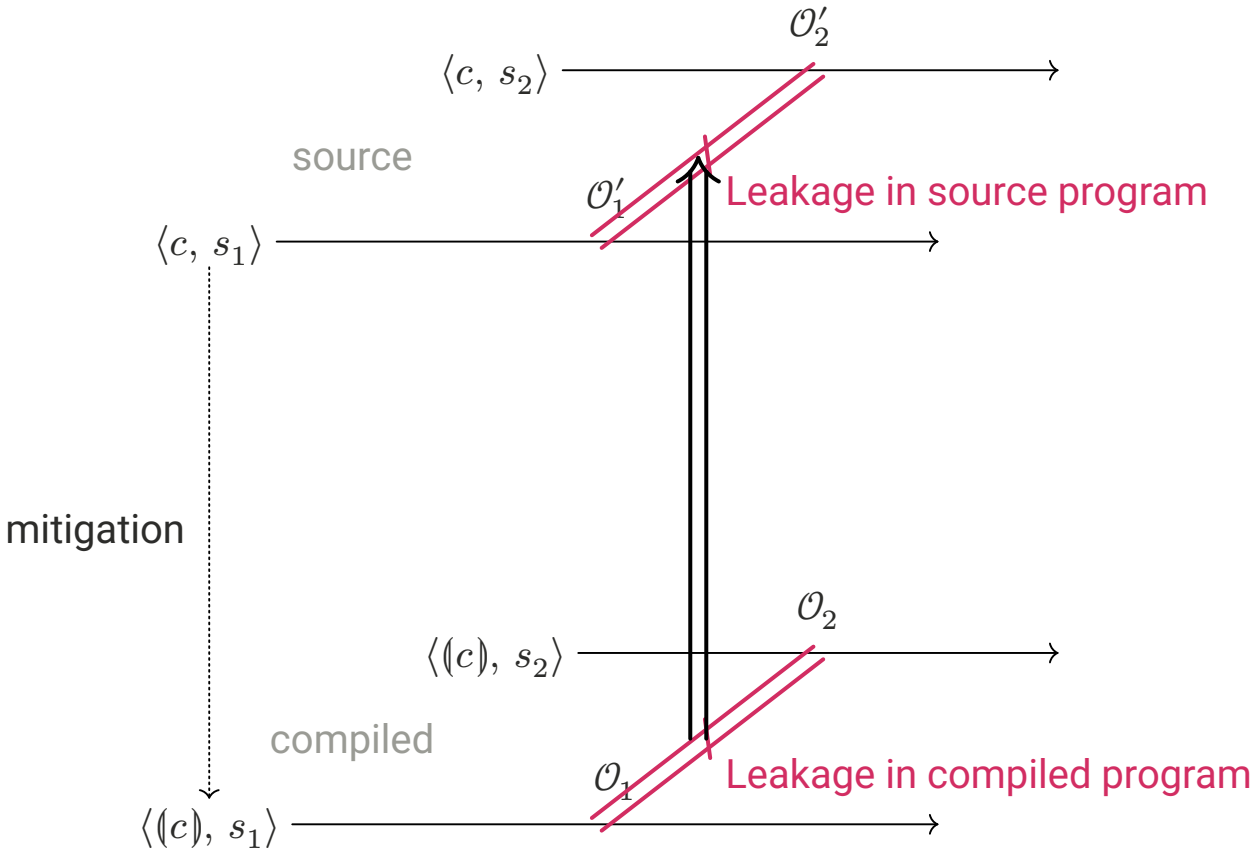
Security for Arbitrary Programs



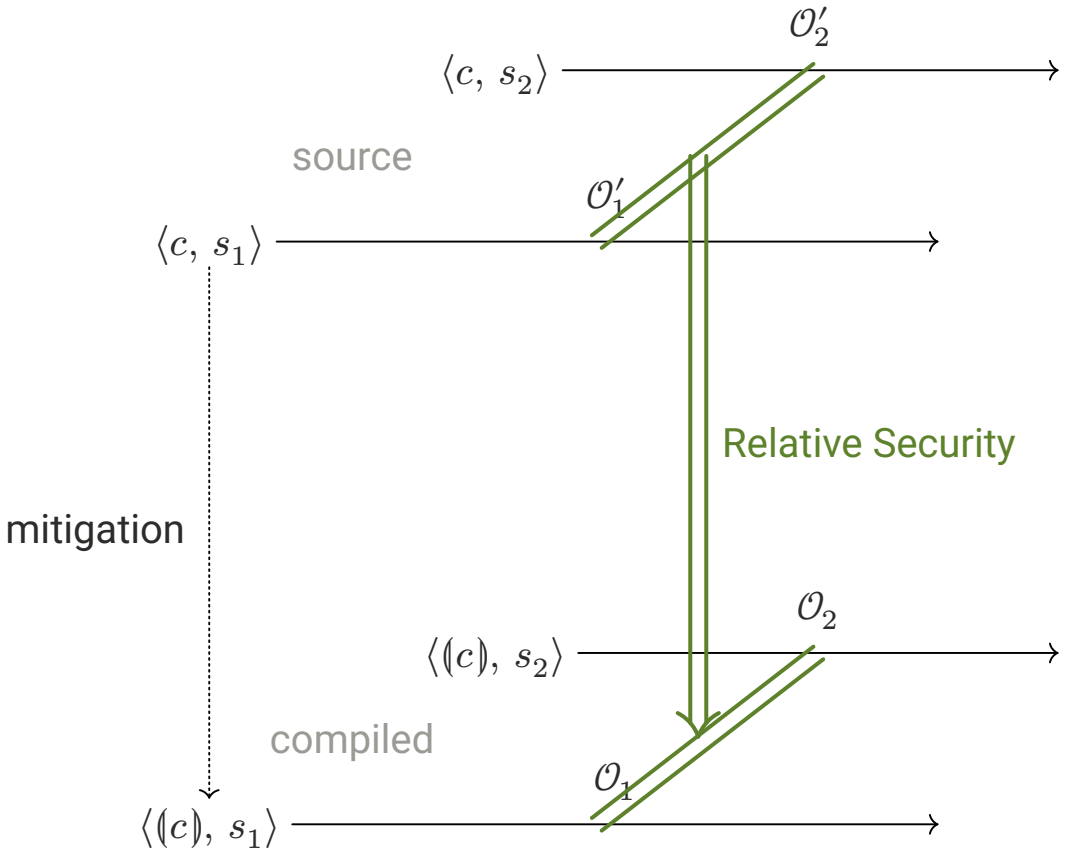
Security for Arbitrary Programs

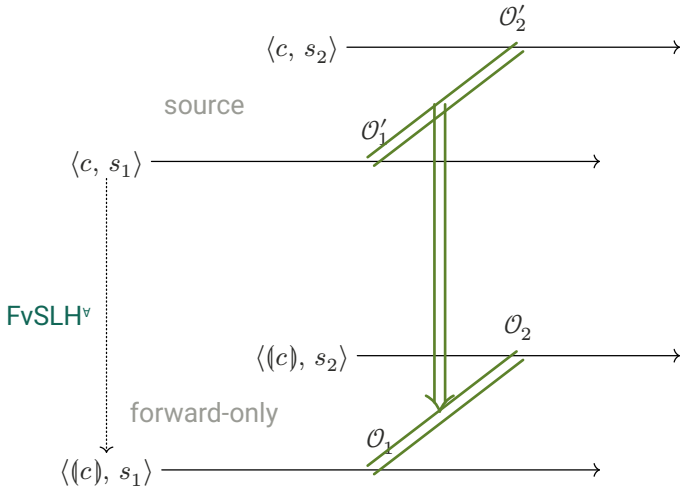


Security for Arbitrary Programs



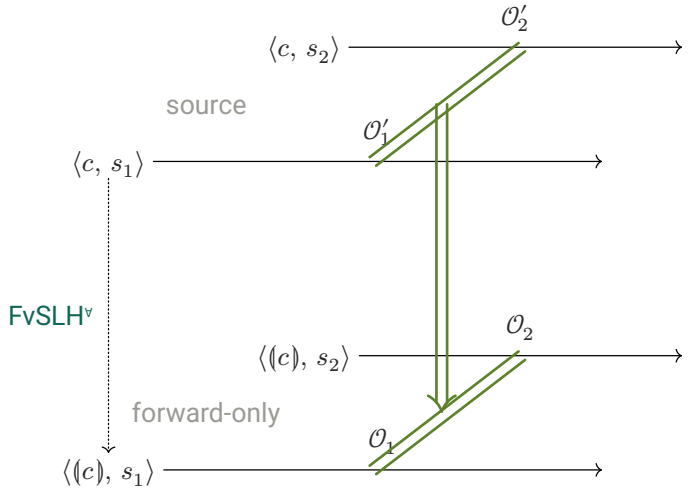
Security for Arbitrary Programs




$\mathcal{O}_2'$
$\langle c,\ s_2 \rangle$
source
$\mathcal{O}_1'$
$\langle c,\ s_1 \rangle$
FvSLH$^\forall$
$\mathcal{O}_2$
$\langle (\!|c|\!),\ s_2 \rangle$
forward-only
$\mathcal{O}_1$
$\langle (\!|c|\!),\ s_1 \rangle$

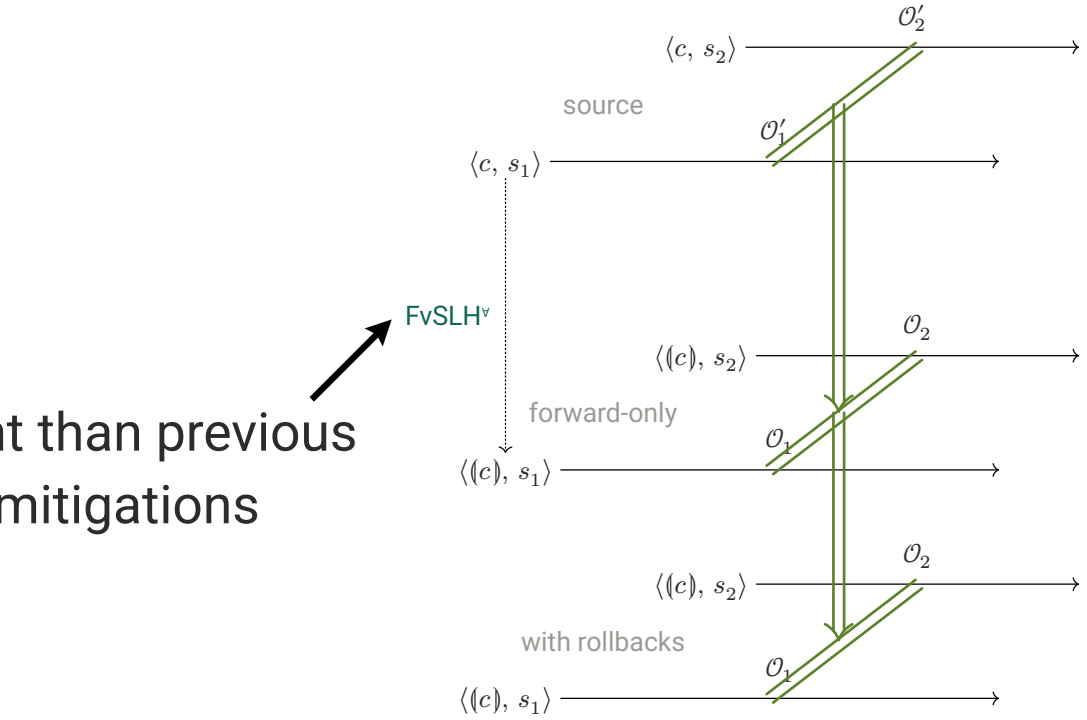
More Efficient and Trustworthy Formally Secure Compilation

more efficient than previous general mitigations



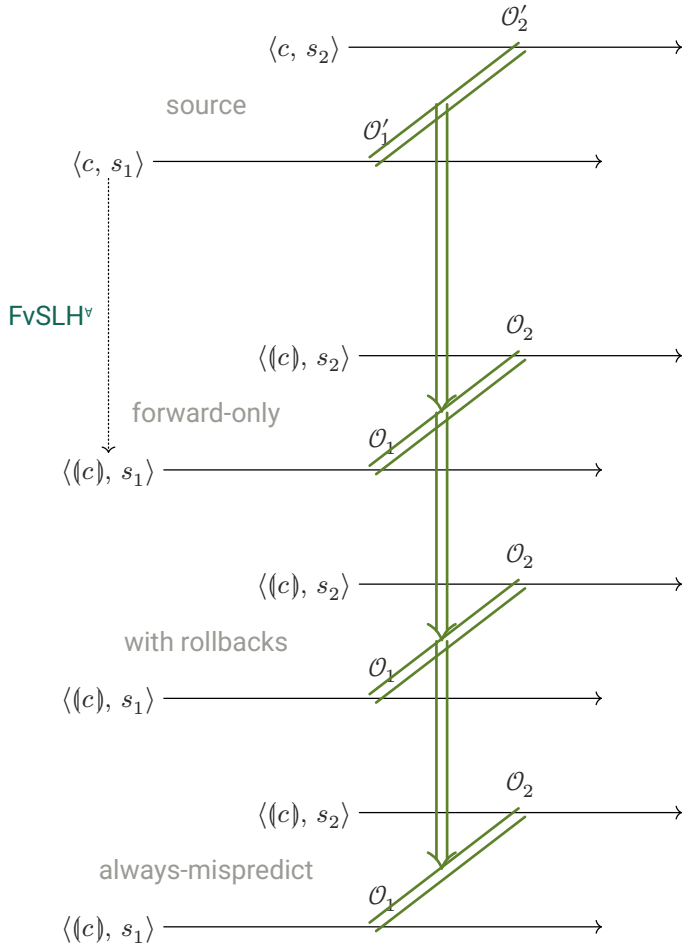
More Efficient and Trustworthy Formally Secure Compilation

more efficient than previous general mitigations



More Efficient and Trustworthy Formally Secure Compilation

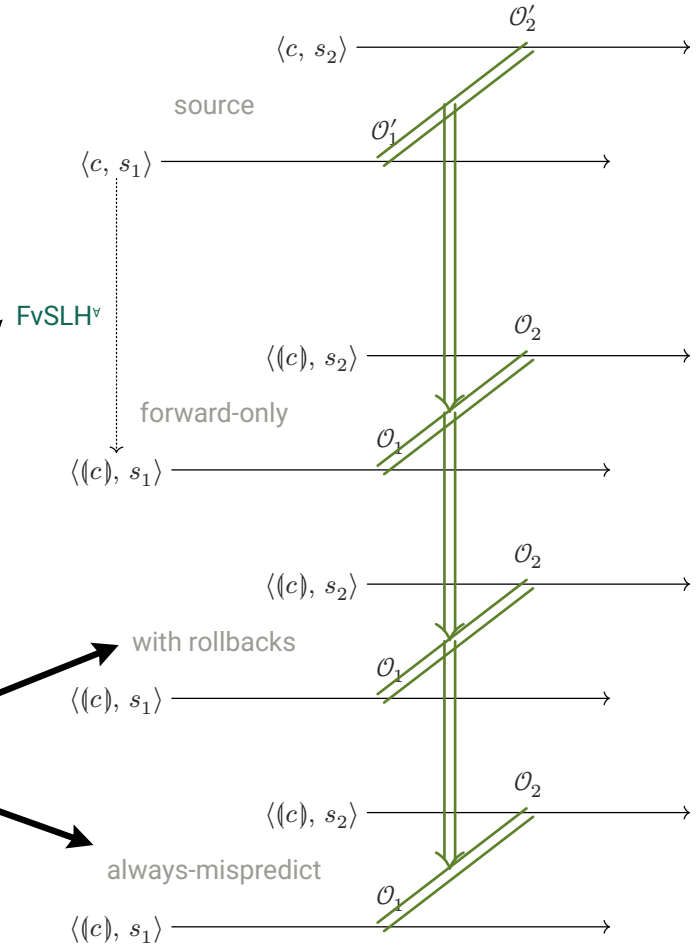
more efficient than previous general mitigations



More Efficient and Trustworthy Formally Secure Compilation

more efficient than previous general mitigations

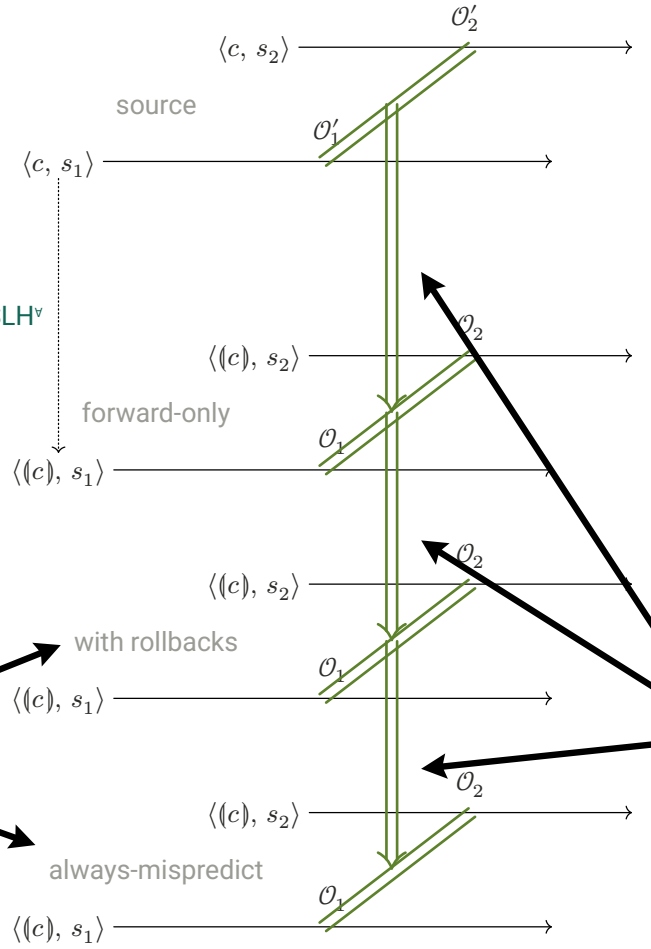
more precise models



More Efficient and Trustworthy Formally Secure Compilation

more efficient than previous general mitigations

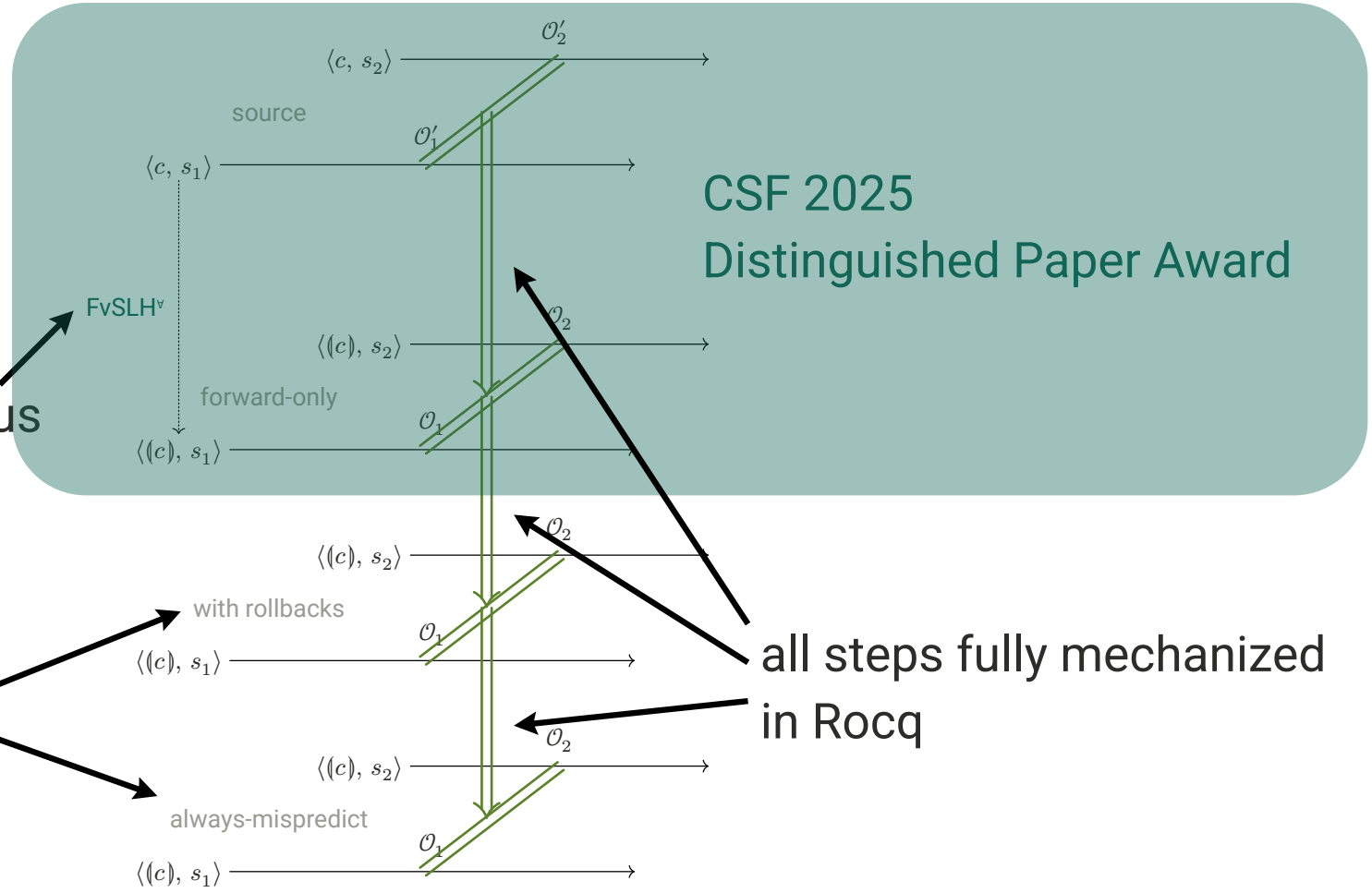
more precise models



all steps fully mechanized in Rocq

More Efficient and Trustworthy Formally Secure Compilation

MAX PLANCK INSTITUTE
FOR SECURITY AND PRIVACY



more efficient than previous general mitigations

more precise models

all steps fully mechanized in Rocq

Flexible SLH:
Providing Efficient Protections To All Programs

Previous Work: Selective SLH (Shivakumar et al. 2023)

$$\texttt{if i} \;< size(\texttt{a}_1)\; \texttt{then}$$

$$\texttt{j} \leftarrow \texttt{a}_1\,[\texttt{i}\,];$$
$$\texttt{x} \leftarrow \texttt{a}_2\,[\texttt{j}\,]$$

$$\texttt{else}$$

Previous Work: Selective SLH (Shivakumar et al. 2023)

$$\texttt{if } \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \texttt{ then}$$

$$\mathtt{j}_{\mathbb{T}} \leftarrow \mathtt{a}_{1\mathbb{T}}[\mathtt{i}_{\mathbb{T}}];$$

$$\mathtt{x}_{\mathbb{F}} \leftarrow \mathtt{a}_{2\mathbb{F}}[\mathtt{j}_{\mathbb{T}}]$$

$$\texttt{else}$$

- CCT type system:
  - variables and arrays public or secret

Previous Work: Selective SLH (Shivakumar et al. 2023)


```
if i_T < size(a1)_T then
    j_T ← a1_T[i_T];
    x_F ← a2_F[j_T];
    y ← a3_T[x_F];
    if y < 10 then ... else ...
else
    b := i_T < size(a1)_T ? 1 : b
```

- CCT type system:
  - variables and arrays public or secret
  - secret values may not be used as indices or branch conditions

Previous Work: Selective SLH (Shivakumar et al. 2023)


```
if i_T < size(a1)_T then
    b := i_T < size(a1)_T ? b : 1;
    j_T ← a1_T[i_T];
    x_F ← a2_F[j_T];
    y ← a3_T[x_F];
    if y < 10 then ... else ...
else
    b := i_T < size(a1)_T ? 1 : b
```

- CCT type system:
  - variables and arrays public or secret
  - secret values may not be used as indices or branch conditions
- maintain a misspeculation flag

Previous Work: Selective SLH (Shivakumar et al. 2023)


```
if i_T < size(a1)_T then
  b := i_T < size(a1)_T ? b : 1;
  j_T ← a1_T[i_T];
  x_F ← a2_F[j_T];
  y ← a3_T[x_F];
  if y < 10 then ... else ...
else
  b := i_T < size(a1)_T ? 1 : b
```

- CCT type system:
  - variables and arrays public or secret
  - secret values may not be used as indices or branch conditions
- maintain a misspeculation flag
  - updated with constant-time conditionals

Previous Work: Selective SLH (Shivakumar et al. 2023)


```
if i_T < size(a1)_T then
    b := i_T < size(a1)_T ? b : 1;
    j_T ← a1_T[i_T]; j_T := b ? 0 : j_T;
    x_F ← a2_F[j_T];
    y ← a3_T[x_F];
    if y < 10 then ... else ...
else
    b := i_T < size(a1)_T ? 1 : b
```

- CCT type system:
  - variables and arrays public or secret
  - secret values may not be used as indices or branch conditions
- maintain a misspeculation flag
  - updated with constant-time conditionals
- mask reads to public variables

Previous Work: Selective SLH (Shivakumar et al. 2023)


```
if i_T < size(a1)_T then
  b := i_T < size(a1)_T ? b : 1;
  j_T ← a1_T[i_T]; j_T := b ? 0 : j_T;
  x_F ← a2_F[j_T];
  y ← a3_T[x_F];
  if y < 10 then ... else ...
else
  b := i_T < size(a1)_T ? 1 : b
```

- CCT type system:
  - variables and arrays public or secret
  - secret values may not be used as indices or branch conditions
- maintain a misspeculation flag
  - updated with constant-time conditionals
- mask reads to public variables
  - secret variables can not leak anyway

Flexible SLH: Removing Restrictions


```
if i_T < size(a_1)_T then
    b := i_T < size(a_1)_T ? b : 1;
    j_T ← a_1T[i_T]; j_T := b ? 0 : j_T;
    x_F ← a_2F[j_T];
    y ← a_3T[x_F];
    if y < 10 then ... else ...
else
    b := i_T < size(a_1)_T ? 1 : b
```

- Static Information-Flow Analysis

Flexible SLH: Removing Restrictions


```
if i_T < size(a1)_T then
    b := i_T < size(a1)_T ? b : 1;
    j_T ← a1_T[i_T]; j_T := b ? 0 : j_T;
    x_F ← a2_F[j_T];
    y_F ← a3_T[x_F];
    if y_F < 10 then ... else ...
else
    b := i_T < size(a1)_T ? 1 : b
```

- Static Information-Flow Analysis
  - annotates expressions with labels

Flexible SLH: Removing Restrictions

$$\texttt{if } \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \texttt{ then}$$

$$\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, \mathtt{b} : 1;$$

$$\quad \mathtt{j}_{\mathbb{T}} \leftarrow \mathtt{a}_{1\mathbb{T}}[\mathtt{i}_{\mathbb{T}}]; \mathtt{j}_{\mathbb{T}} := \mathtt{b} \,?\, 0 : \mathtt{j}_{\mathbb{T}};$$

$$\quad \mathtt{x}_{\mathbb{F}} \leftarrow \mathtt{a}_{2\mathbb{F}}[\mathtt{j}_{\mathbb{T}}];$$

$$\quad \mathtt{y}_{\mathbb{F}} \leftarrow \mathtt{a}_{3\mathbb{T}}[\mathtt{x}_{\mathbb{F}}];$$

$$\quad \texttt{if } \mathtt{y}_{\mathbb{F}} < 10 \texttt{ then } \ldots \texttt{ else } \ldots$$

$$\texttt{else}$$

$$\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, 1 : \mathtt{b}$$

- Static Information-Flow Analysis
  - annotates expressions with labels
  - does not prevent the use of secrets

Flexible SLH: Removing Restrictions

$$
\begin{array}{l}
\texttt{if } \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \texttt{ then} \\
\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, \mathtt{b} : 1; \\
\quad \mathtt{j}_{\mathbb{T}} \leftarrow \mathtt{a}_{1\mathbb{T}}[\mathtt{i}_{\mathbb{T}}]; \mathtt{j}_{\mathbb{T}} := \mathtt{b} \,?\, 0 : \mathtt{j}_{\mathbb{T}}; \\
\quad \mathtt{x}_{\mathbb{F}} \leftarrow \mathtt{a}_{2\mathbb{F}}[\mathtt{j}_{\mathbb{T}}]; \\
\quad \mathtt{y}_{\mathbb{F}} \leftarrow \mathtt{a}_{3\mathbb{T}}[\mathtt{x}_{\mathbb{F}}]; \\
\quad \texttt{if } \mathtt{y}_{\mathbb{F}} < 10 \texttt{ then } \ldots \texttt{ else } \ldots \\
\texttt{else} \\
\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, 1 : \mathtt{b}
\end{array}
$$

- Static Information-Flow Analysis
  - annotates expressions with labels
  - does not prevent the use of secrets
  - accepts all programs

Flexible SLH: Removing Restrictions


```
if i_T < size(a_1)_T then
  b := i_T < size(a_1)_T ? b : 1;
  j_T ← a_1T[i_T]; j_T := b ? 0 : j_T;
  x_F ← a_2F[j_T];
  y_F ← a_3T[x_F];
  if y_F < 10 then ... else ...
else
  b := i_T < size(a_1)_T ? 1 : b
```

- Static Information-Flow Analysis
  - annotates expressions with labels
  - does not prevent the use of secrets
  - accepts all programs
- More masking required:

Flexible SLH: Removing Restrictions

$$\texttt{if } \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \texttt{ then}$$

$$\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, \mathtt{b} : 1;$$

$$\quad \mathtt{j}_{\mathbb{T}} \leftarrow \mathtt{a}_{1\mathbb{T}}[\mathtt{i}_{\mathbb{T}}]; \mathtt{j}_{\mathbb{T}} := \mathtt{b} \,?\, 0 : \mathtt{j}_{\mathbb{T}};$$

$$\quad \mathtt{x}_{\mathbb{F}} \leftarrow \mathtt{a}_{2\mathbb{F}}[\mathtt{j}_{\mathbb{T}}];$$

$$\quad \mathtt{y}_{\mathbb{F}} \leftarrow \mathtt{a}_{3\mathbb{T}}[\mathtt{b} \,?\, 0 : \mathtt{x}];$$

$$\quad \texttt{if } \mathtt{y}_{\mathbb{F}} < 10 \texttt{ then } \ldots \texttt{ else } \ldots$$

$$\texttt{else}$$

$$\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, 1 : \mathtt{b}$$

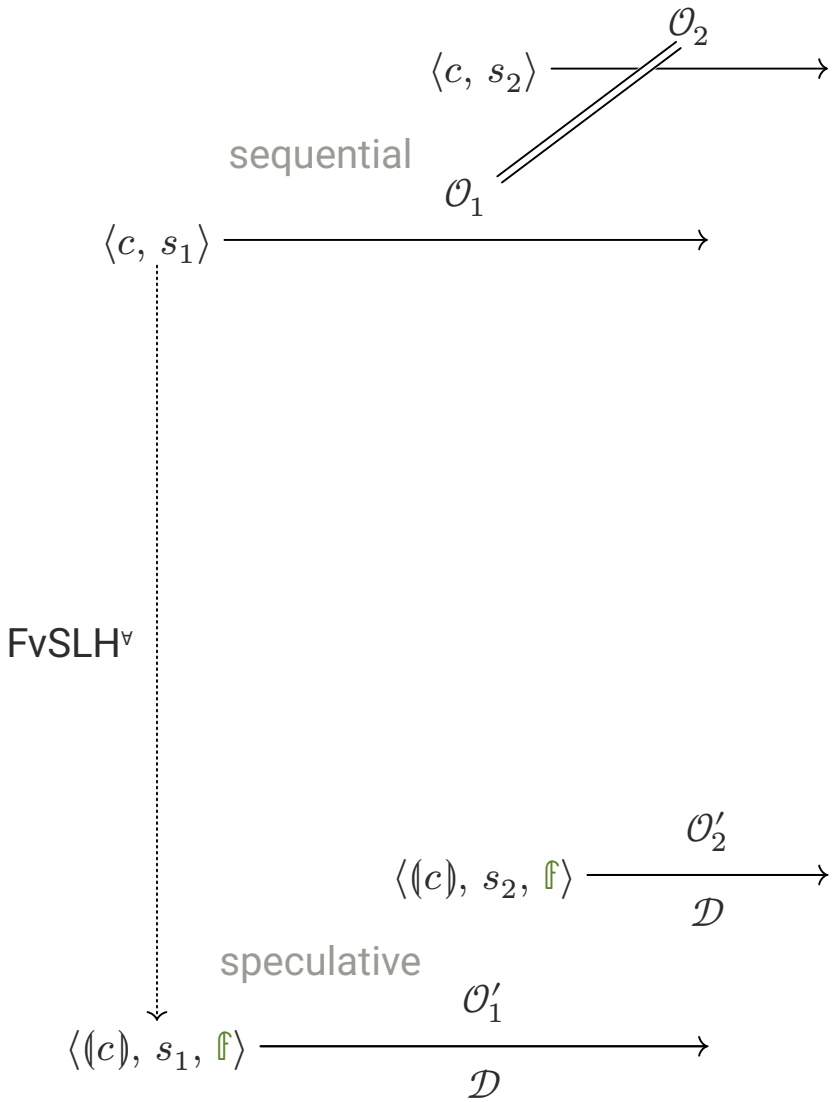
- Static Information-Flow Analysis
  - annotates expressions with labels
  - does not prevent the use of secrets
  - accepts all programs
- More masking required:
  - secret indices

Flexible SLH: Removing Restrictions

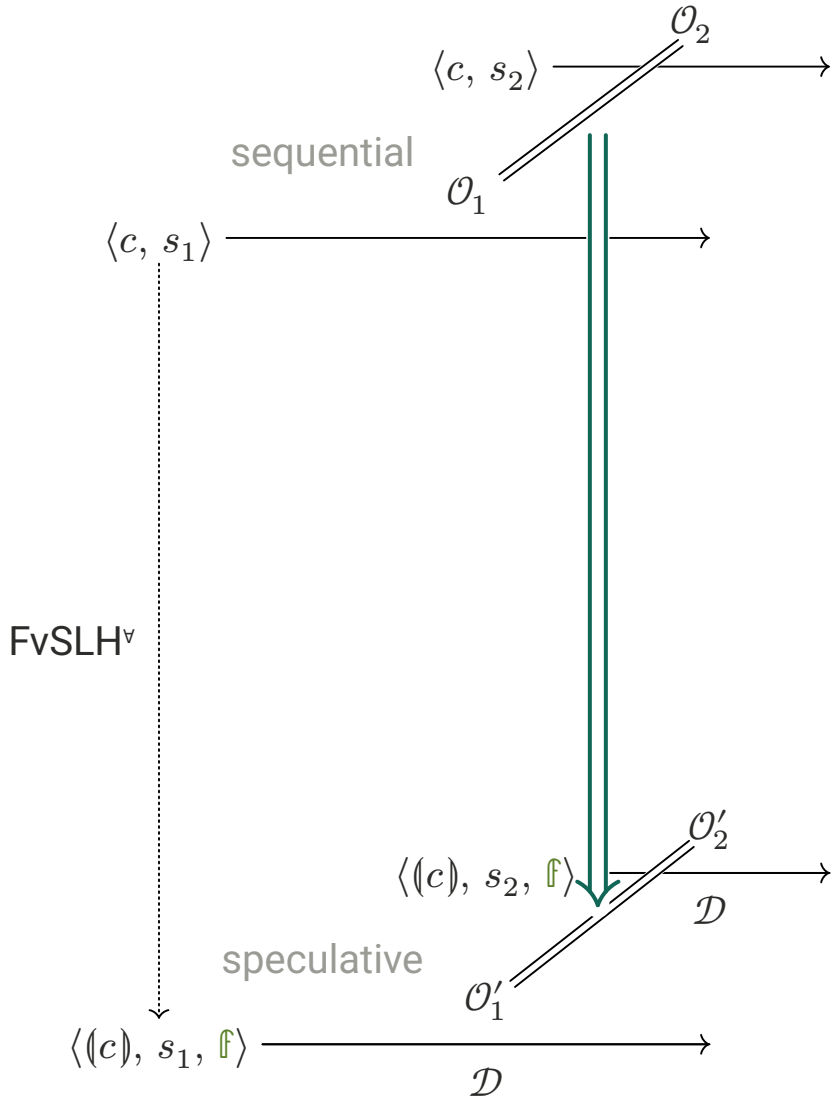
$$
\begin{array}{l}
\texttt{if } \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \texttt{ then} \\
\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, \mathtt{b} : 1; \\
\quad \mathtt{j}_{\mathbb{T}} \leftarrow \mathtt{a}_{1\mathbb{T}}[\mathtt{i}_{\mathbb{T}}]; \mathtt{j}_{\mathbb{T}} := \mathtt{b} \,?\, 0 : \mathtt{j}_{\mathbb{T}}; \\
\quad \mathtt{x}_{\mathbb{F}} \leftarrow \mathtt{a}_{2\mathbb{F}}[\mathtt{j}_{\mathbb{T}}]; \\
\quad \mathtt{y}_{\mathbb{F}} \leftarrow \mathtt{a}_{3\mathbb{T}}[\mathtt{b} \,?\, 0 : \mathtt{x}]; \\
\quad \texttt{if } \mathtt{b} \,\&\&\, \mathtt{y}_{\mathbb{F}} < 10 \texttt{ then } \ldots \texttt{ else } \ldots \\
\texttt{else} \\
\quad \mathtt{b} := \mathtt{i}_{\mathbb{T}} < size(\mathtt{a}_1)_{\mathbb{T}} \,?\, 1 : \mathtt{b}
\end{array}
$$

- Static Information-Flow Analysis
  - annotates expressions with labels
  - does not prevent the use of secrets
  - accepts all programs
- More masking required:
  - secret indices
  - secret branch conditions

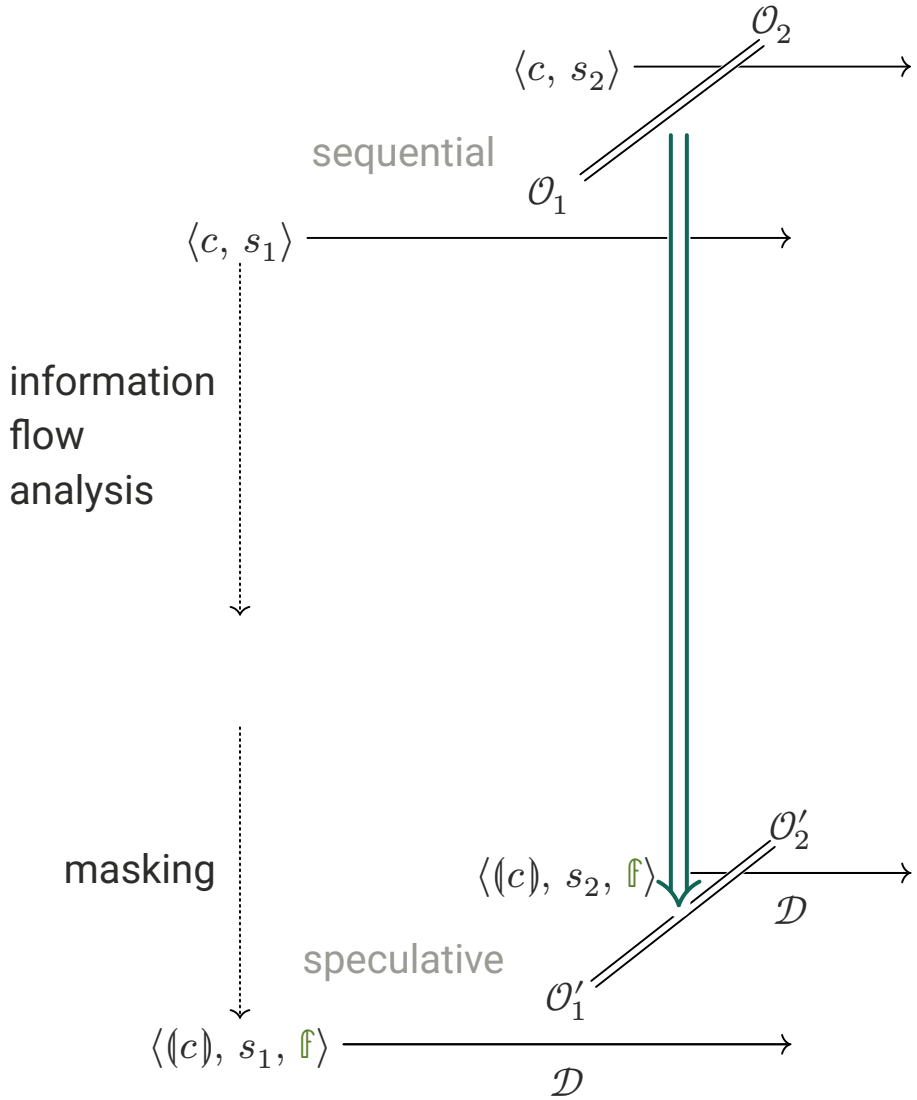
Proving Relative Security



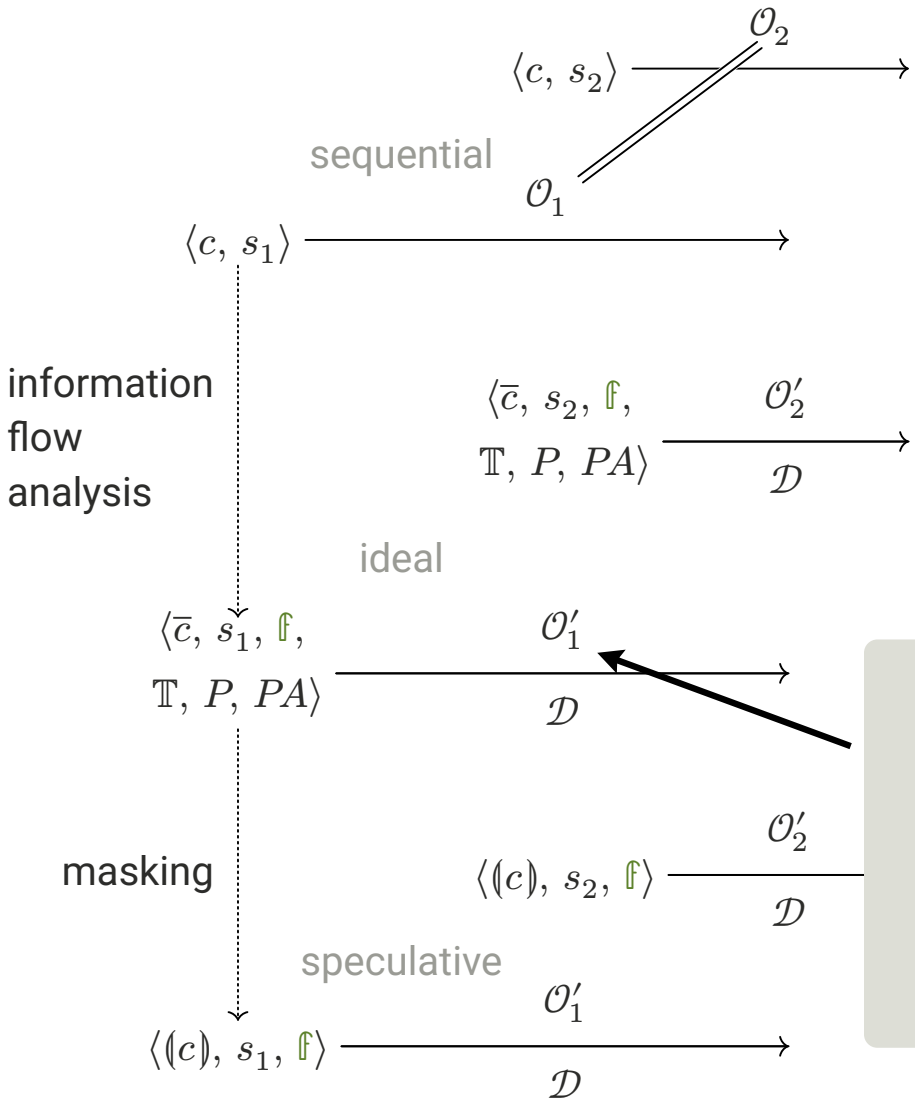
Proving Relative Security



Proving Relative Security

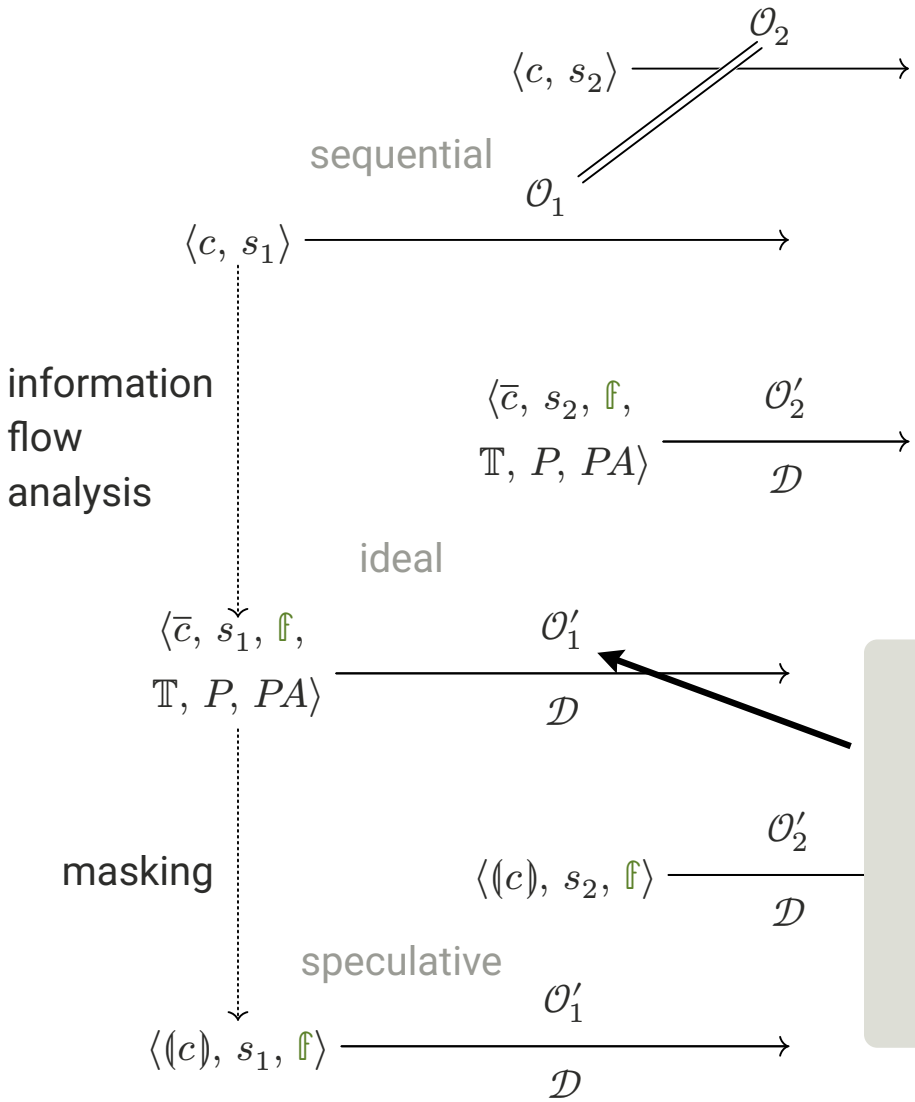


Proving Relative Security



Ideal semantics:

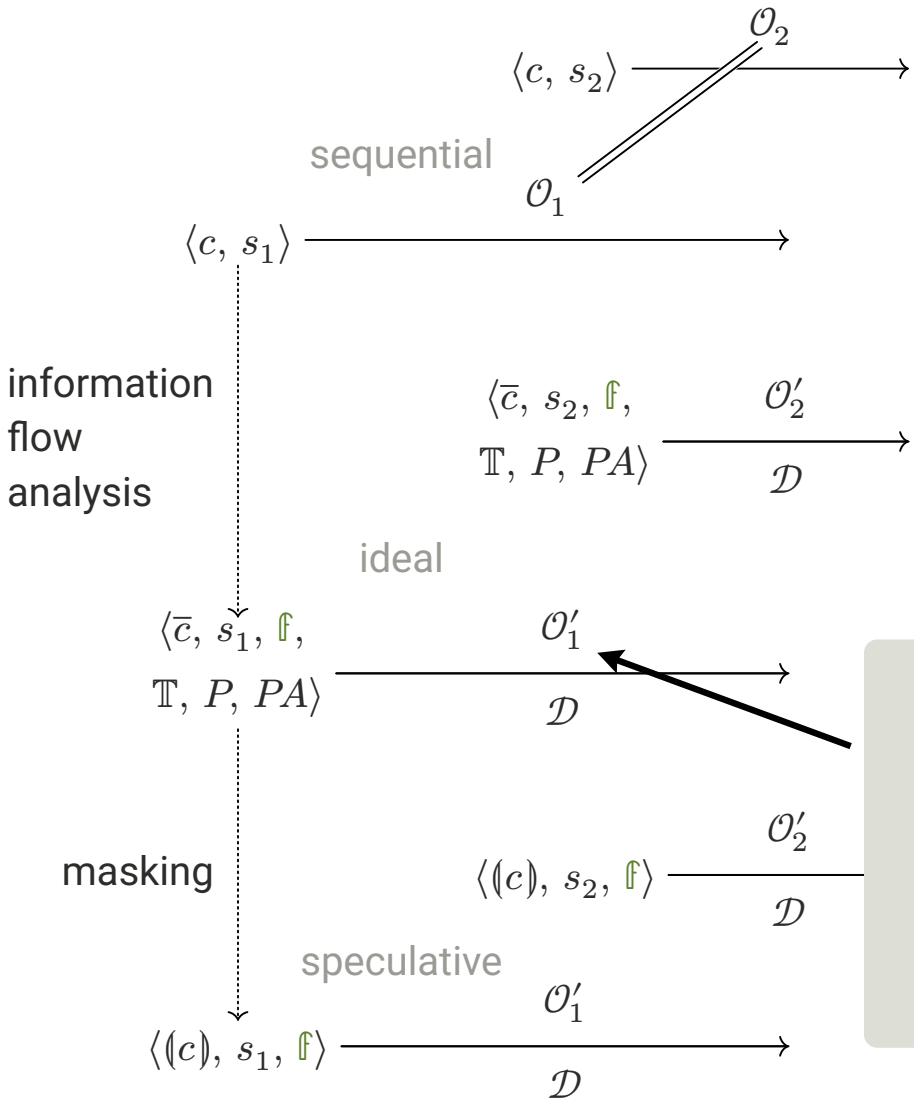
Proving Relative Security



Ideal semantics:

- speculative execution

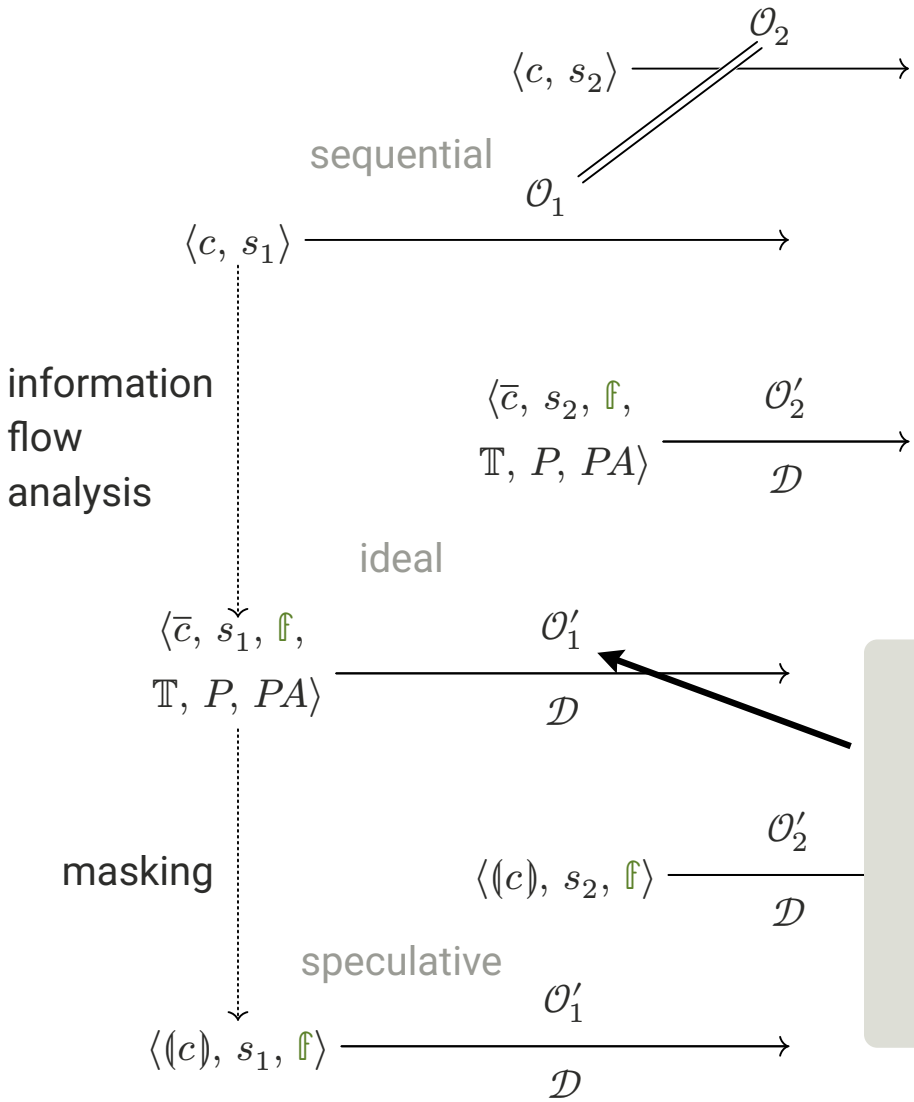
Proving Relative Security



Ideal semantics:

- speculative execution
- with masking in semantics

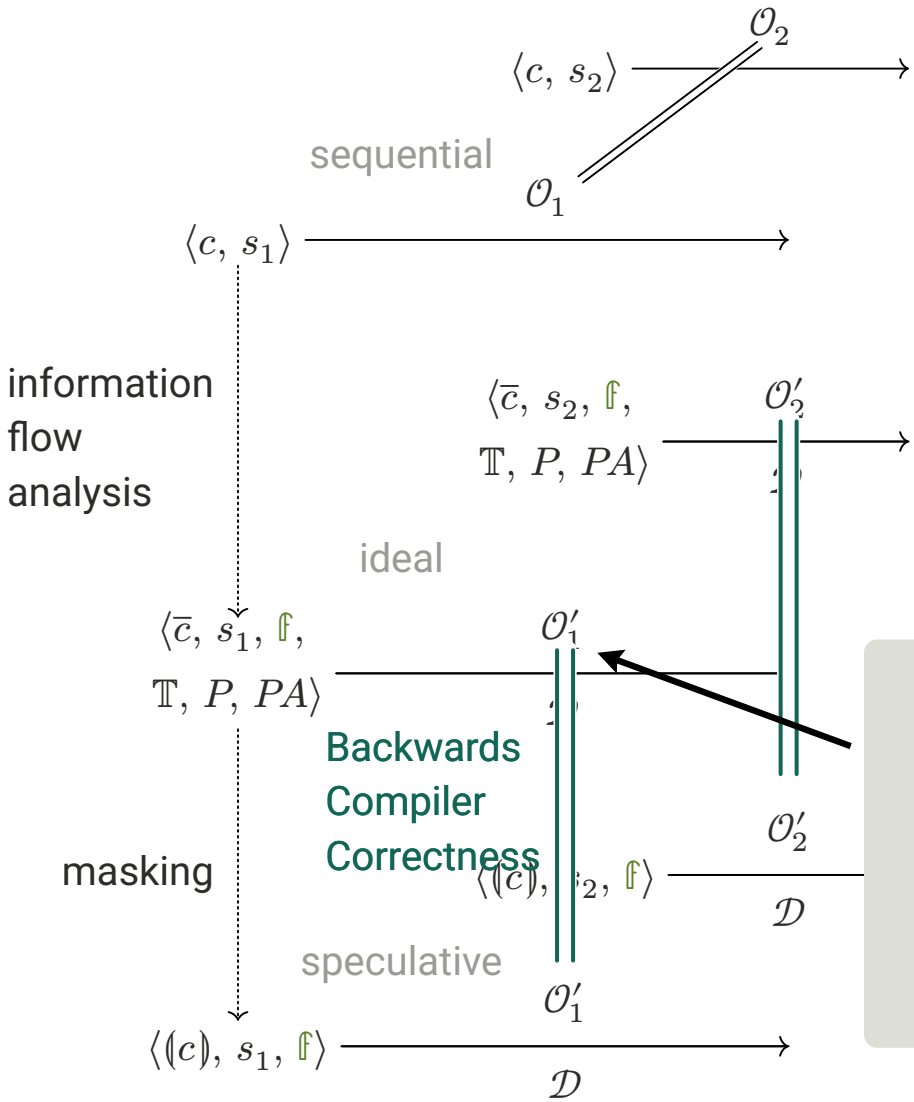
Proving Relative Security



Ideal semantics:

- speculative execution
- with masking in semantics
  - matches behaviour of compiled program

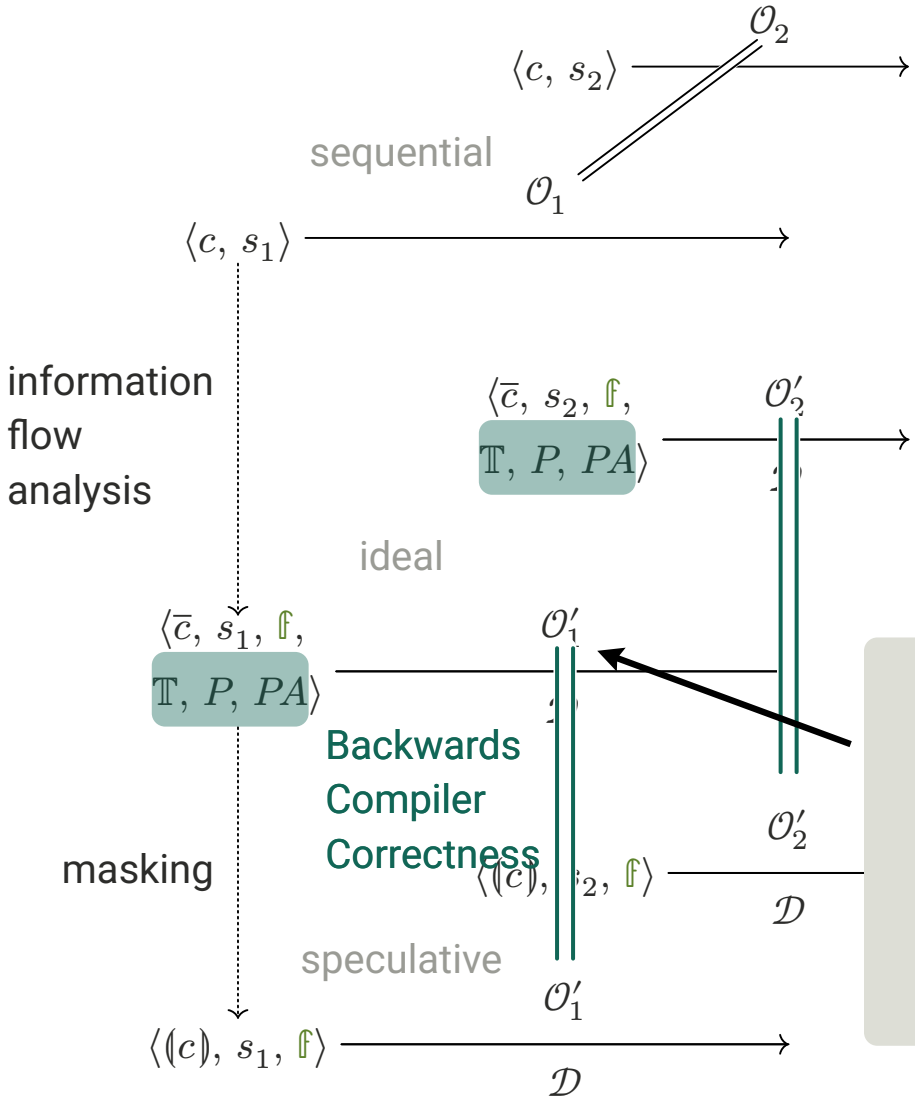
Proving Relative Security



Ideal semantics:

- speculative execution
- with masking in semantics
  - matches behaviour of compiled program

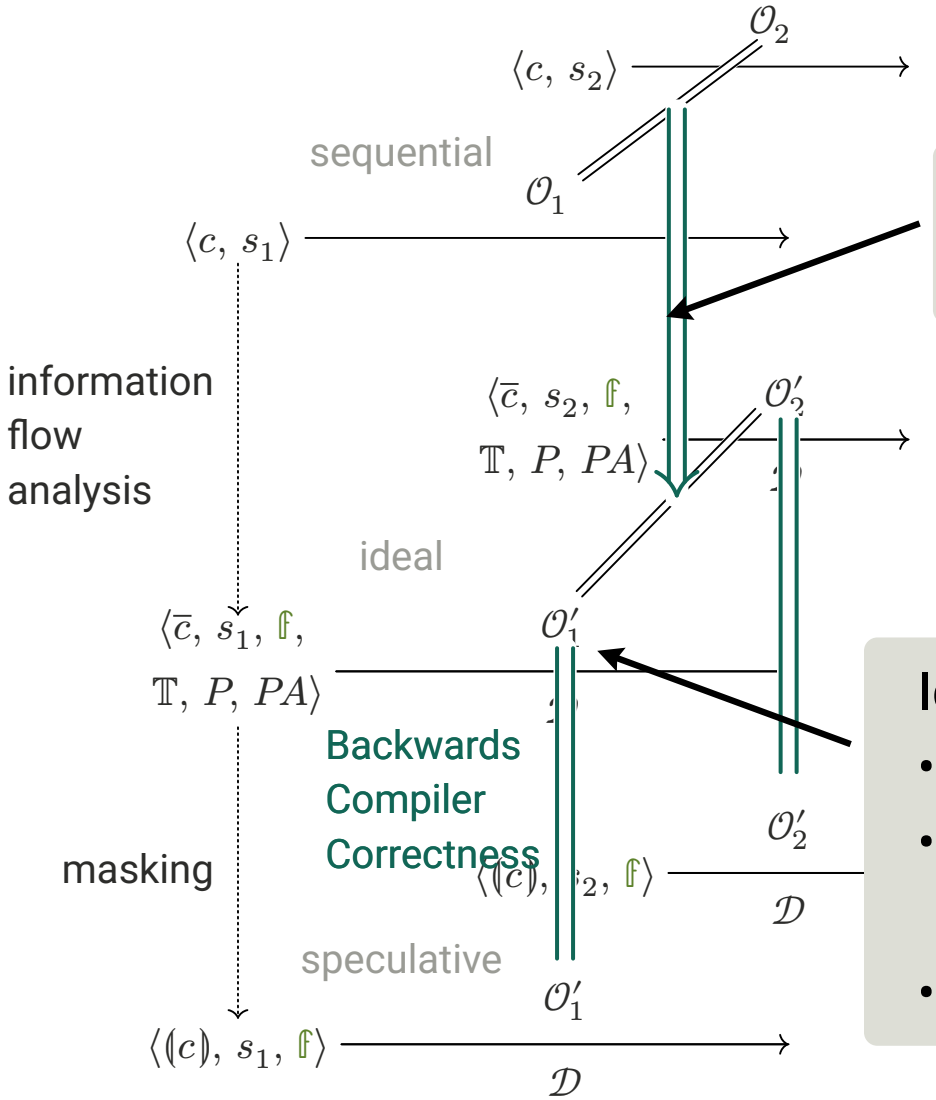
Proving Relative Security



Ideal semantics:

- speculative execution
- with masking in semantics
  - matches behaviour of compiled program
- with dynamic information-flow tracking

Proving Relative Security

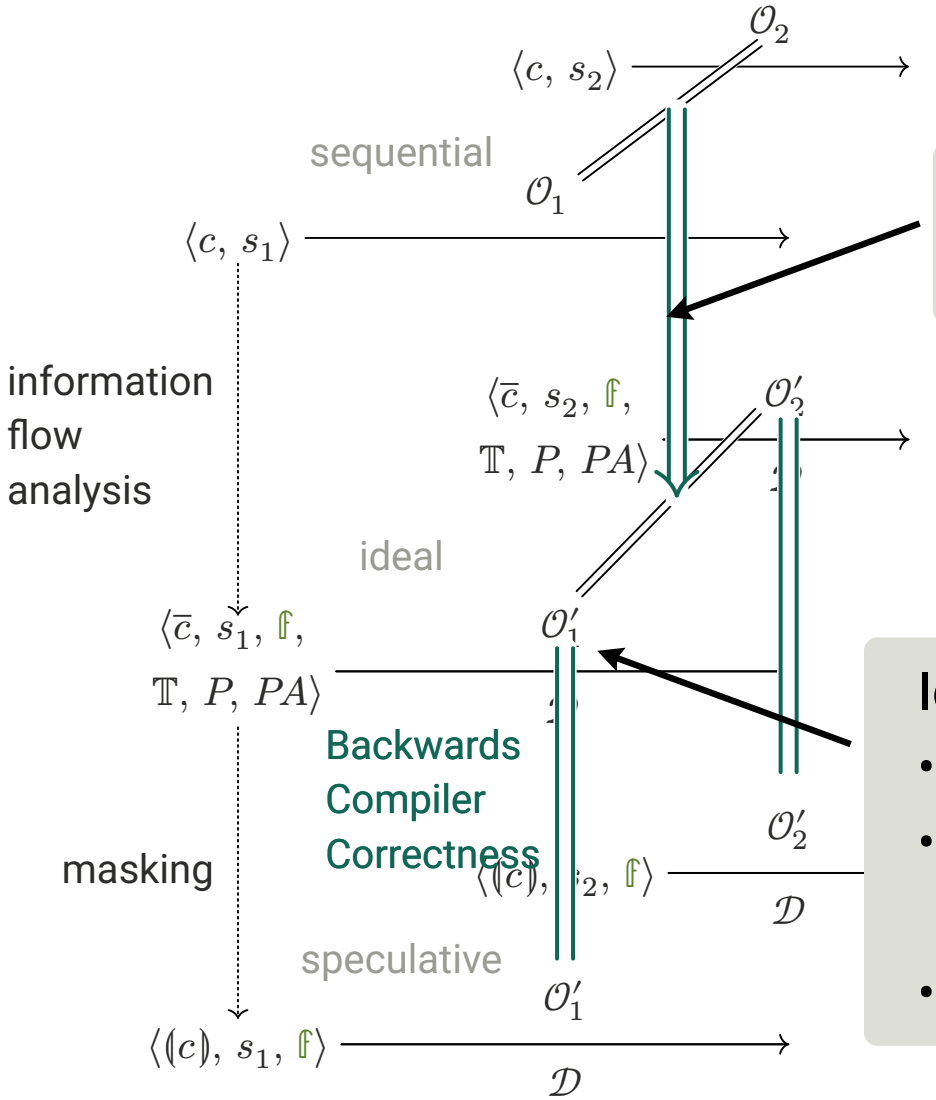


Relative Security of ideal semantics:

Ideal semantics:

- speculative execution
- with masking in semantics
  - matches behaviour of compiled program
- with dynamic information-flow tracking

Proving Relative Security



Relative Security of ideal semantics:

⚠️ depends on correctness of annotations

Ideal semantics:

- speculative execution
- with masking in semantics
  - matches behaviour of compiled program
- with dynamic information-flow tracking

Well-Labeledness

- Relative security requires correct annotations during execution

Well-Labeledness

- Relative security requires correct annotations during execution
- Annotations are produced by static analysis on the initial program

Well-Labeledness

- Relative security requires correct annotations during execution
- Annotations are produced by static analysis on the initial program
  - not suitable for preservation

Well-Labeledness

- Relative security requires correct annotations during execution
- Annotations are produced by static analysis on the initial program
  - not suitable for preservation
- Introduce a *typing-like* well-labeledness predicate:

Well-Labeledness

- Relative security requires correct annotations during execution
- Annotations are produced by static analysis on the initial program
  - not suitable for preservation
- Introduce a *typing-like* well-labeledness predicate:

$$P, PA \rightsquigarrow P'', PA'' \vdash_{pc} \overline{c}$$

Well-Labeledness

- Relative security requires correct annotations during execution
- Annotations are produced by static analysis on the initial program
  - not suitable for preservation
- Introduce a *typing-like* well-labeledness predicate:

$$P, PA \rightsquigarrow P'', PA'' \vdash_{pc} \overline{c}$$

annotated command

Well-Labeledness

- Relative security requires correct annotations during execution
- Annotations are produced by static analysis on the initial program
  - not suitable for preservation
- Introduce a *typing-like* well-labeledness predicate:

$$P, PA \rightsquigarrow P'', PA'' \vdash_{pc} \overline{c}$$

initial labeling

annotated command

Well-Labeledness

- Relative security requires correct annotations during execution
- Annotations are produced by static analysis on the initial program
  - not suitable for preservation
- Introduce a *typing-like* well-labeledness predicate:

$$P, PA \rightsquigarrow P'', PA'' \vdash_{pc} \overline{c}$$

initial labeling

final labeling

annotated command

Well-Labeledness

Lemma

The information-flow analysis produces well-labeled programs.

$$\langle\!\langle c \rangle\!\rangle_{pc}^{P,PA} = (\overline{c}, P', PA') \Rightarrow P, PA \rightsquigarrow P', PA' \vdash_{pc} \overline{c}$$

ng execution

e initial program

- not suitable for preservation

- Introduce a *typing-like* well-labeledness predicate:

$$P, PA \rightsquigarrow P'', PA'' \vdash_{pc} \overline{c}$$

initial labeling

final labeling

annotated command

Lemma

The information-flow analysis produces well-labeled programs.

$$\langle\!\langle c \rangle\!\rangle_{pc}^{P,PA} = (\overline{c}, P', PA') \Rightarrow P, PA \rightsquigarrow P', PA' \vdash_{pc} \overline{c}$$

ng execution

e initial program

- not suitable for preser
- Introduce a *typing-like* we

initial labeli

Lemma

Ideal execution preserves well-labeledness.

$$P, PA \rightsquigarrow P'', PA'' \vdash_{pc} \overline{c} \quad \Rightarrow$$

$$\langle \overline{c}, \rho, \mu, b, pc, P, PA \rangle \xrightarrow[\mathcal{D}]{\mathcal{O}}_{\mathrm{i}} \langle \overline{c'}, \rho, \mu, b, pc', P', PA' \rangle \Rightarrow$$

$$P', PA' \rightsquigarrow P'', PA'' \vdash_{pc'} \overline{c'}$$

Lemma

The information-flow analysis produces well-labeled programs.

$$\langle\!\langle c \rangle\!\rangle_{pc}^{P,PA} = (\overline{c}, P', PA') \Rightarrow P, PA \rightsquigarrow P', PA' \vdash_{pc} \overline{c}$$

ng execution

e initial program

- not suitable for preser
- Introduce a *typing-like* we

initial labeli

Lemma

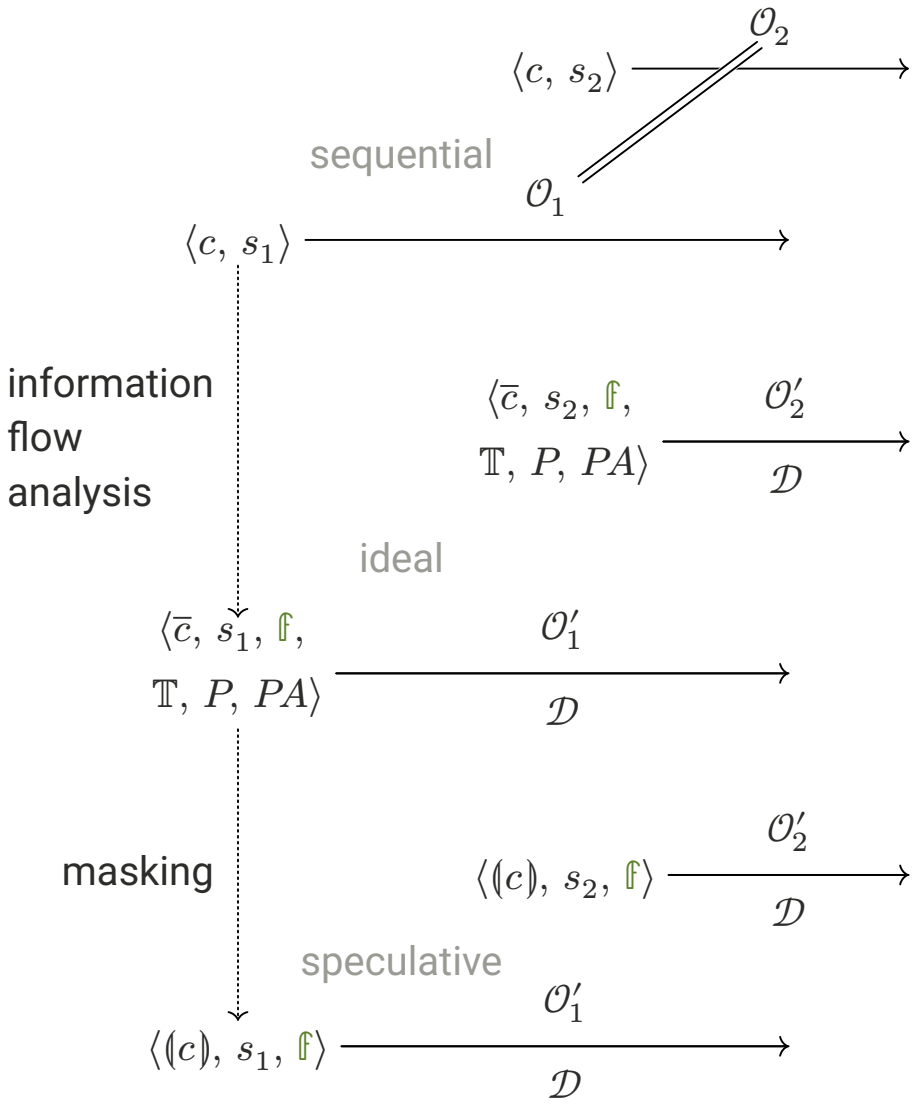
Ideal execution preserves well-labeledness.

$$P, PA \rightsquigarrow P'', PA'' \vdash_{pc} \overline{c} \Rightarrow$$

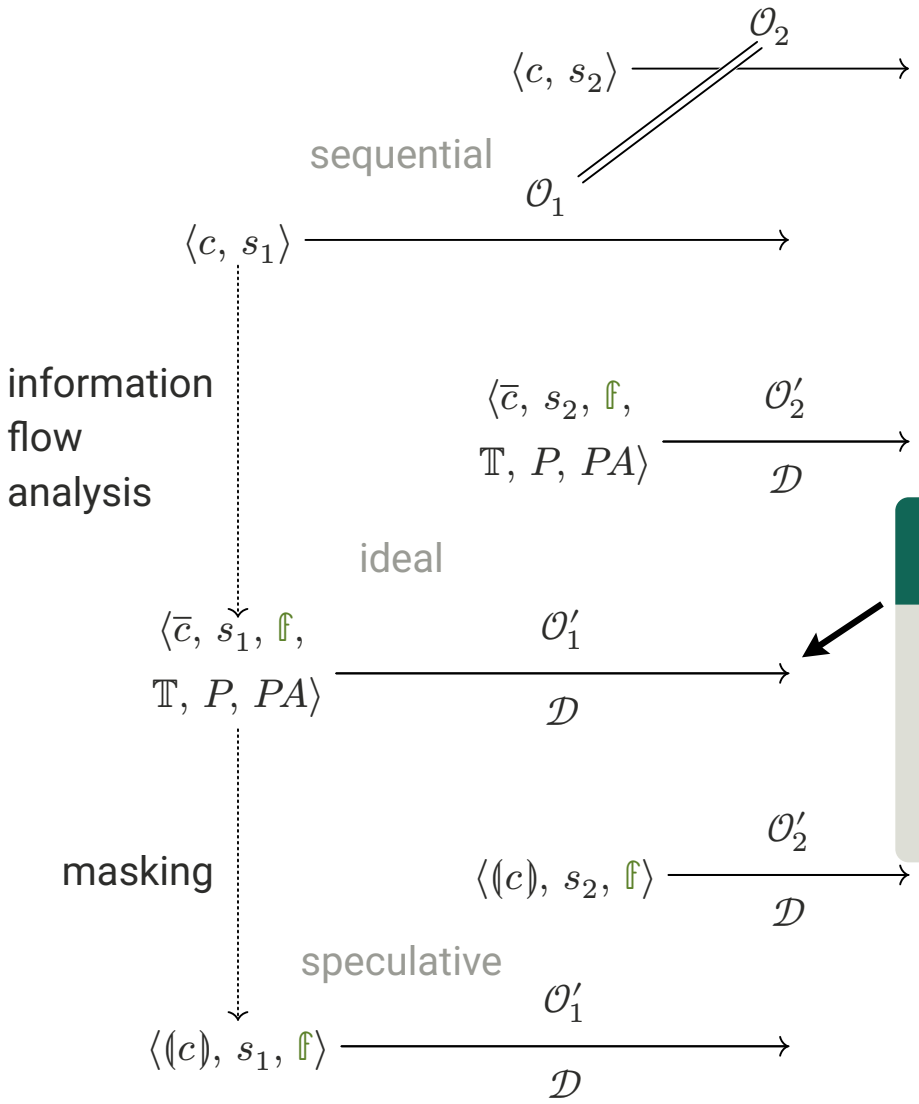
$$\langle \overline{c}, \rho, \mu, b, pc, P, PA \rangle \xrightarrow[\mathcal{D}]{\mathcal{O}}_{\mathrm{i}} \langle \overline{c'}, \rho, \mu, b, pc', P', PA' \rangle \Rightarrow$$

$$P', PA' \rightsquigarrow P'', PA'' \vdash_{pc'} \overline{c'}$$

Proving Relative Security



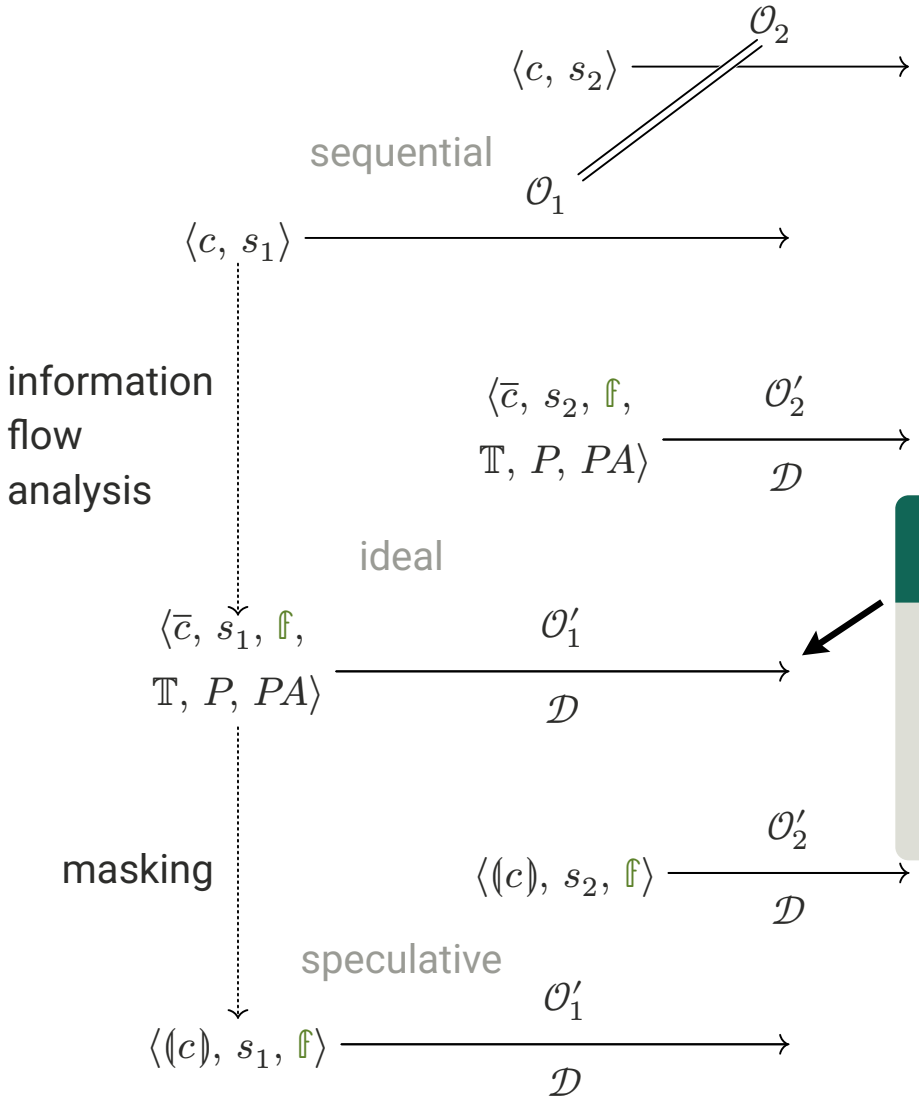
Proving Relative Security



Unwinding

During misspeculation:

Proving Relative Security

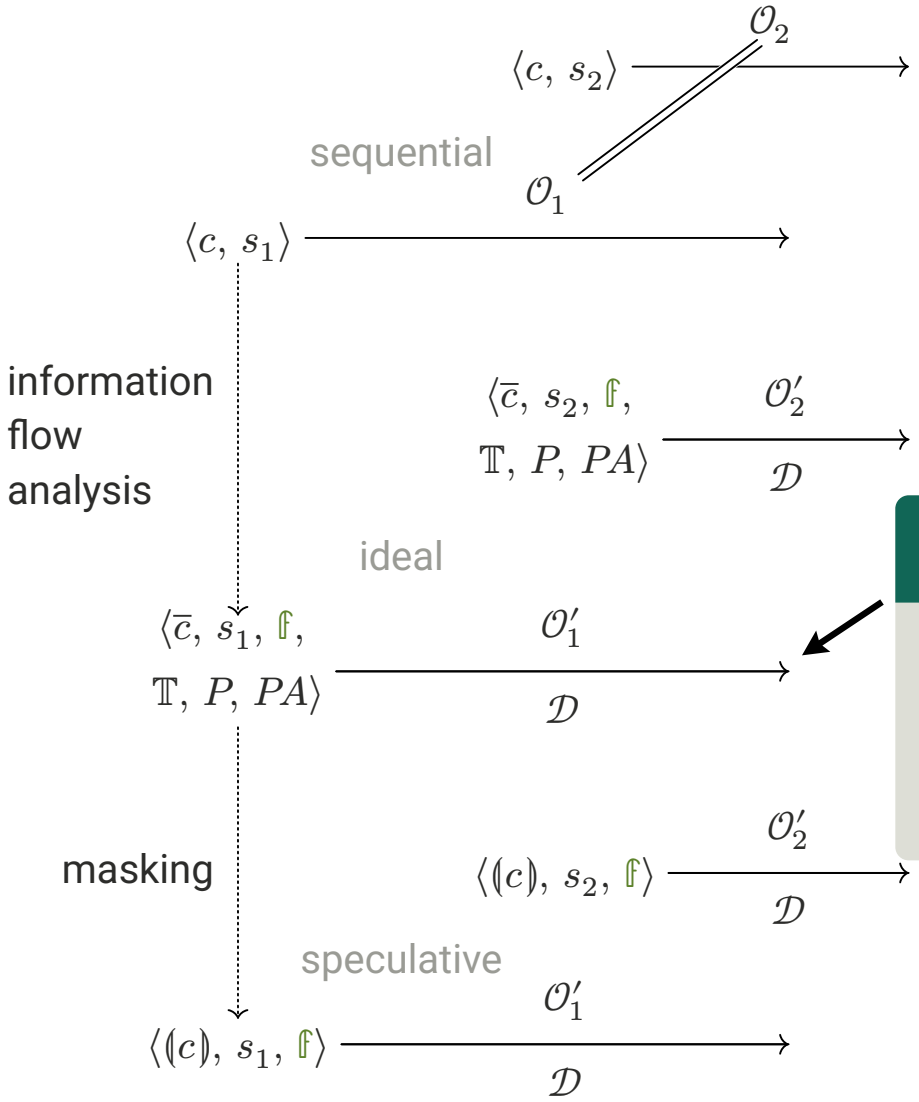


Unwinding

During misspeculation:

- all secret values are masked

Proving Relative Security

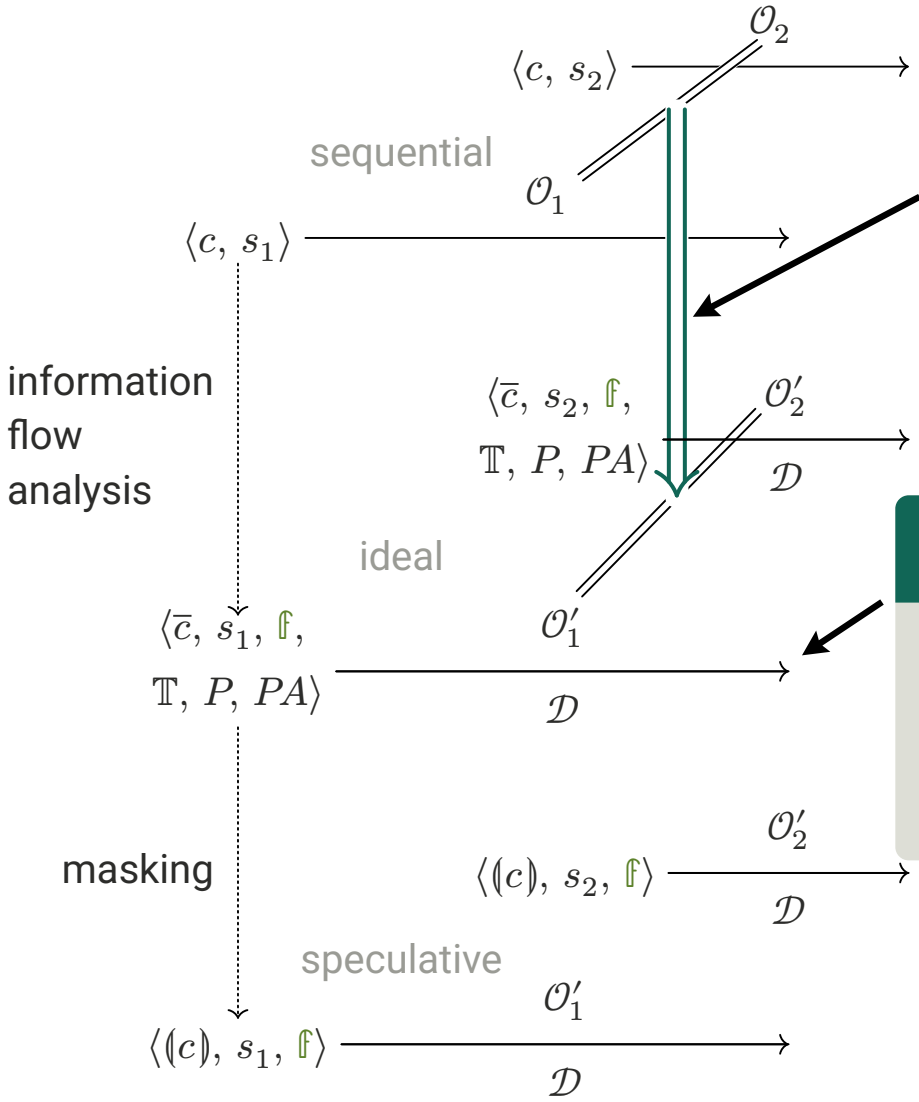


Unwinding

During misspeculation:

- all secret values are masked
- all public values are equal in both executions

Proving Relative Security



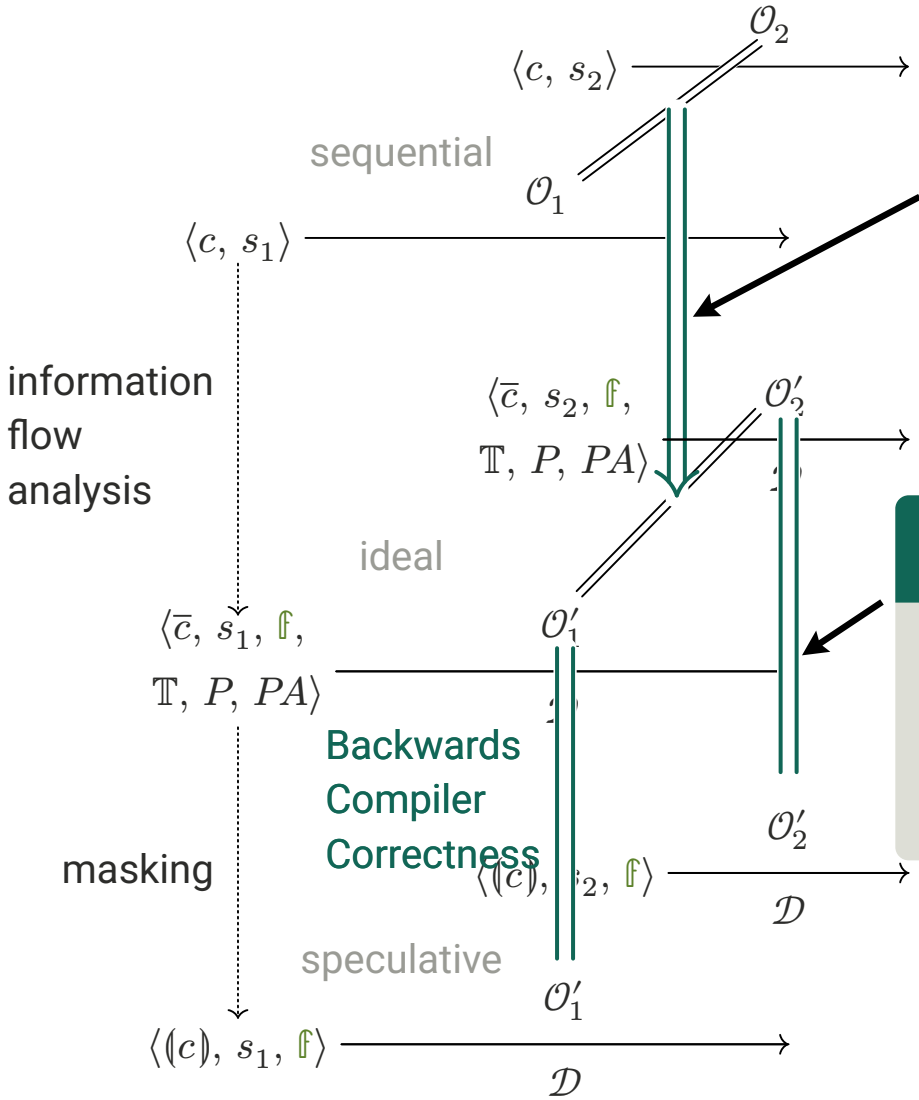
Same behaviour before misspeculation

Unwinding

During misspeculation:

- all secret values are masked
- all public values are equal in both executions

Proving Relative Security



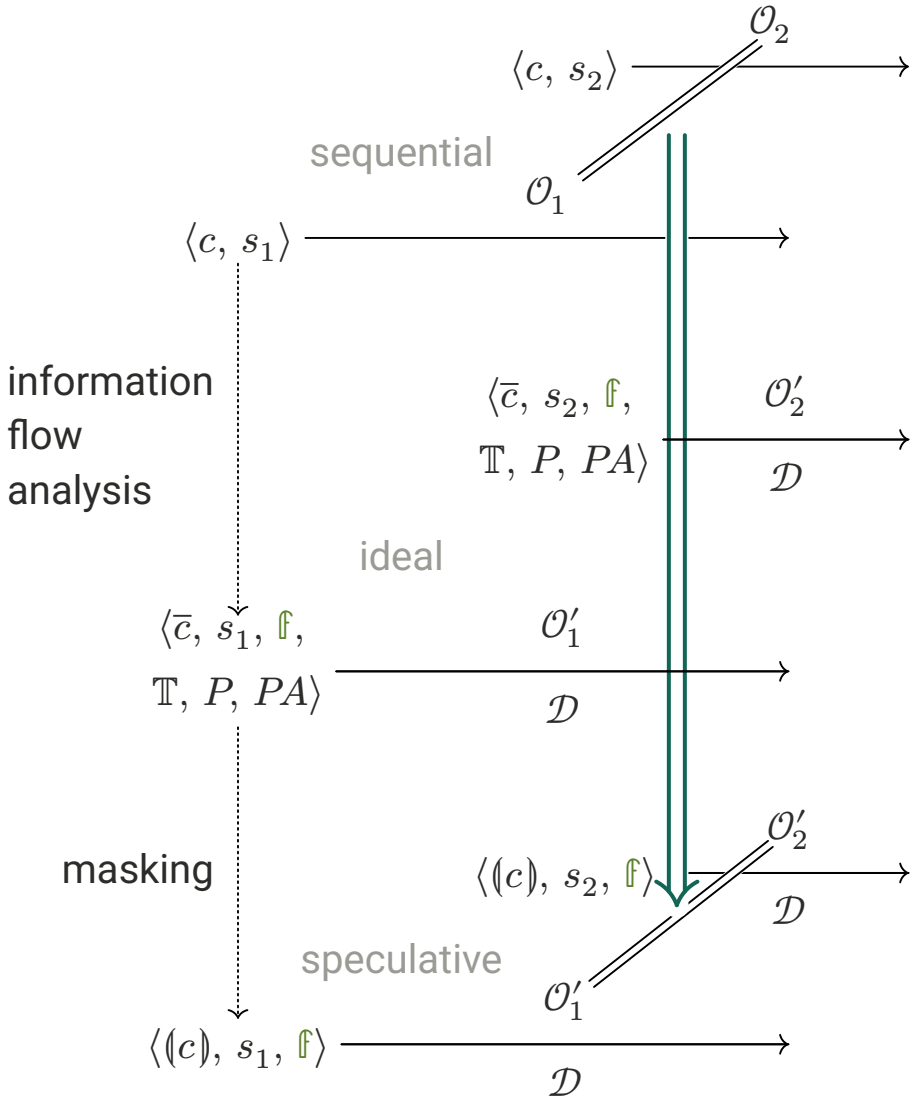
Same behaviour before misspeculation

Unwinding

During misspeculation:

- all secret values are masked
- all public values are equal in both executions

Proving Relative Security



Flexible SLH: Recap

- secure: fully mechanized relative security proof in Rocq

Flexible SLH: Recap

- secure: fully mechanized relative security proof in Rocq
- general: accepts all programs

Flexible SLH: Recap

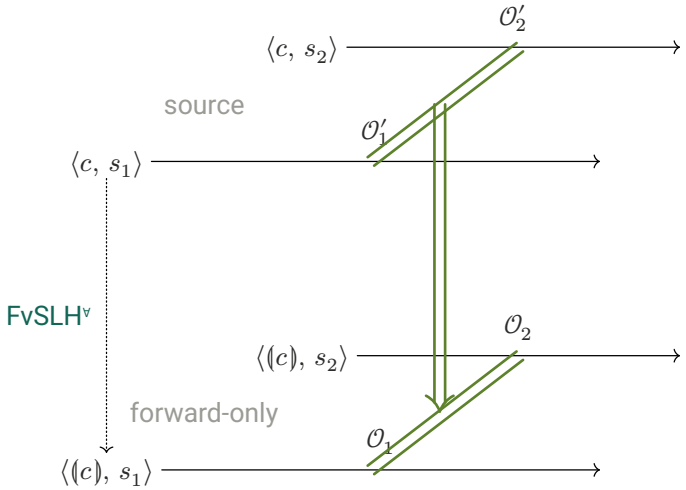
- secure: fully mechanized relative security proof in Rocq
- general: accepts all programs
- efficient: only inserts protections where needed

Flexible SLH: Recap

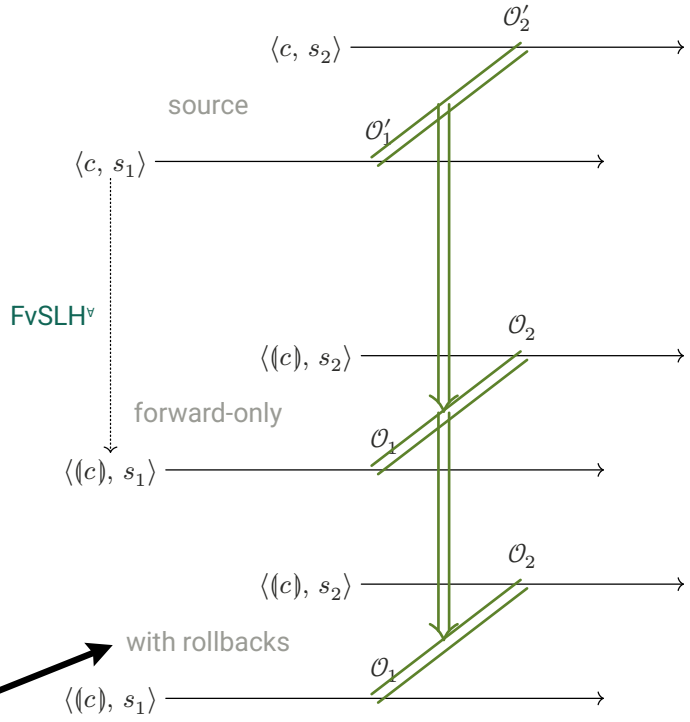
- secure: fully mechanized relative security proof in Rocq
- general: accepts all programs
- efficient: only inserts protections where needed
- no real-world implementation yet

More Realistic Models of Speculation

Translation to More Realistic Models

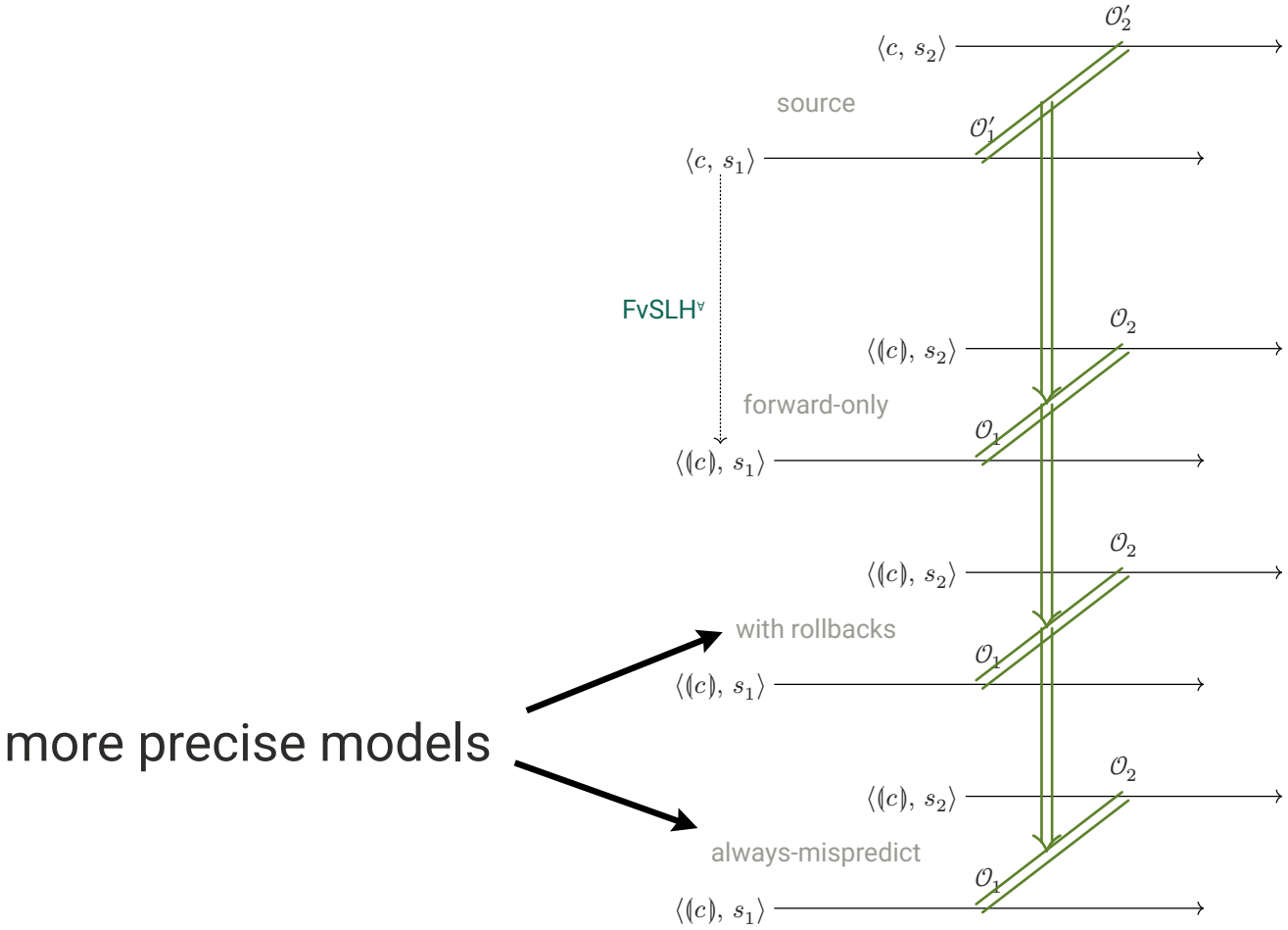


Translation to More Realistic Models



more precise models

Translation to More Realistic Models

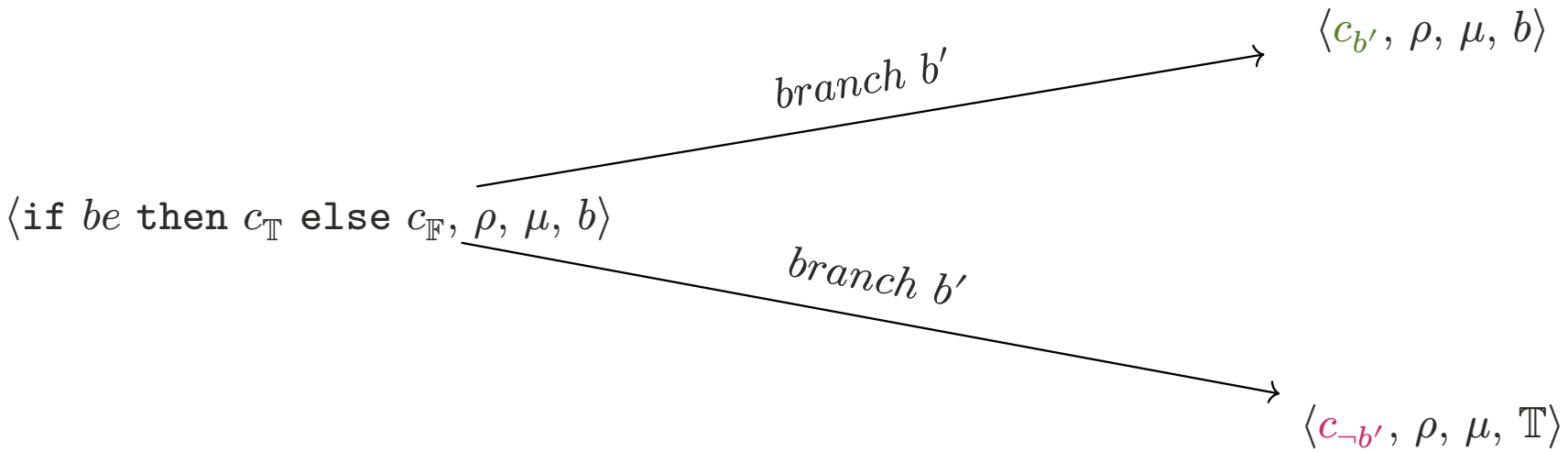


Forward-Only Semantics to Rollbacks

The model for the FSLH security proof is very abstract:

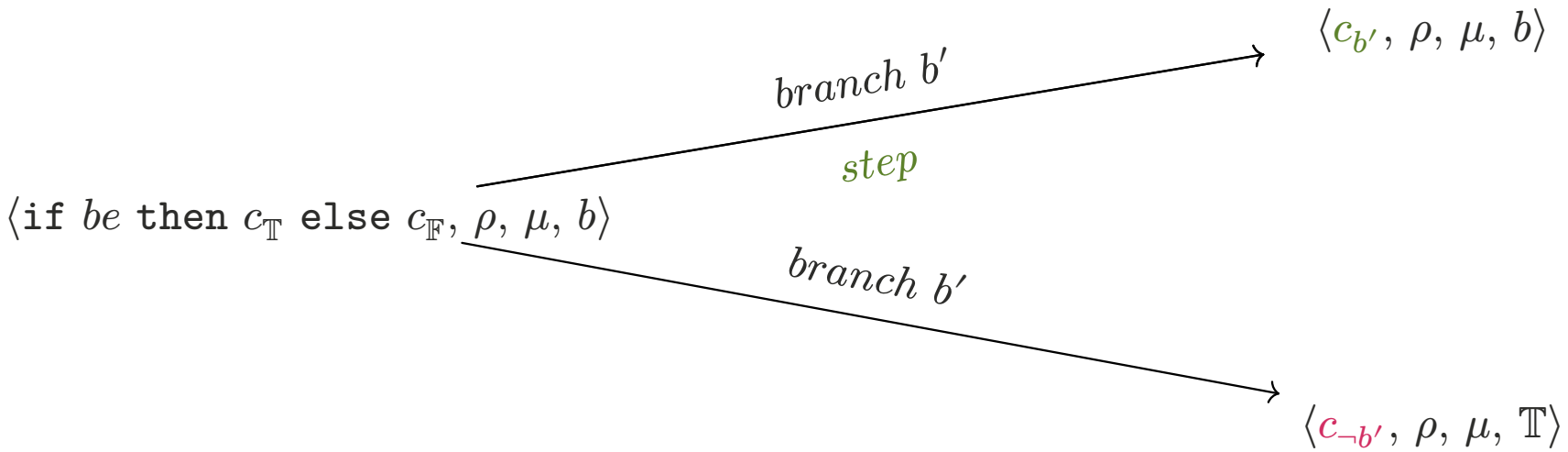
The model for the FSLH security proof is very abstract:

- Directives control branch prediction



The model for the FSLH security proof is very abstract:

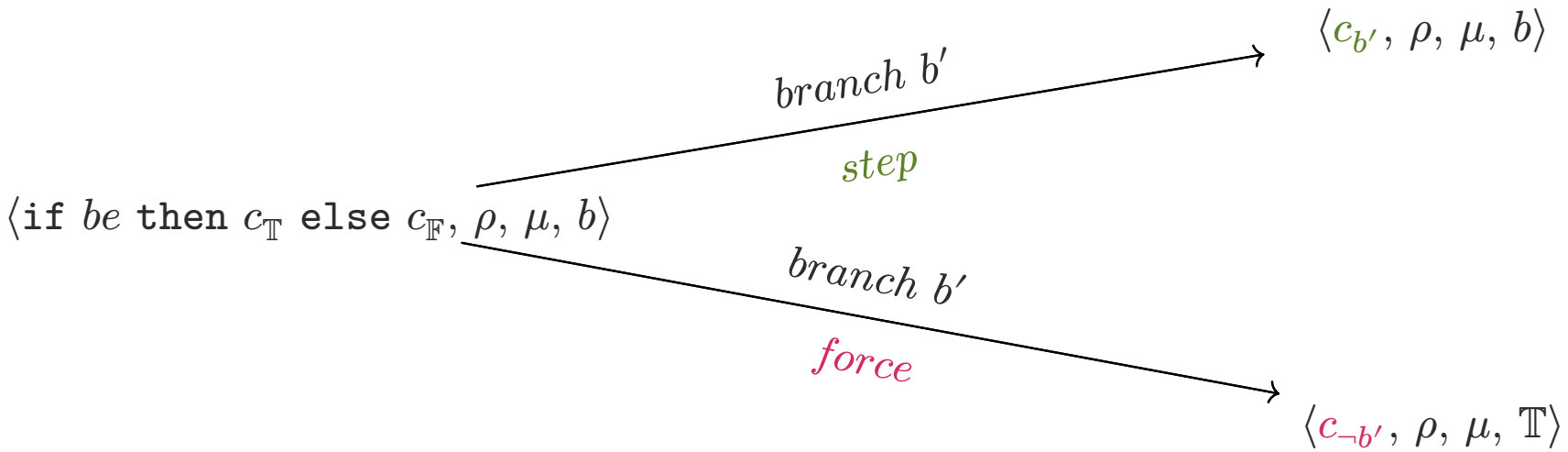
- Directives control branch prediction



Forward-Only Semantics to Rollbacks

The model for the FSLH security proof is very abstract:

- Directives control branch prediction



Forward-Only Semantics to Rollbacks

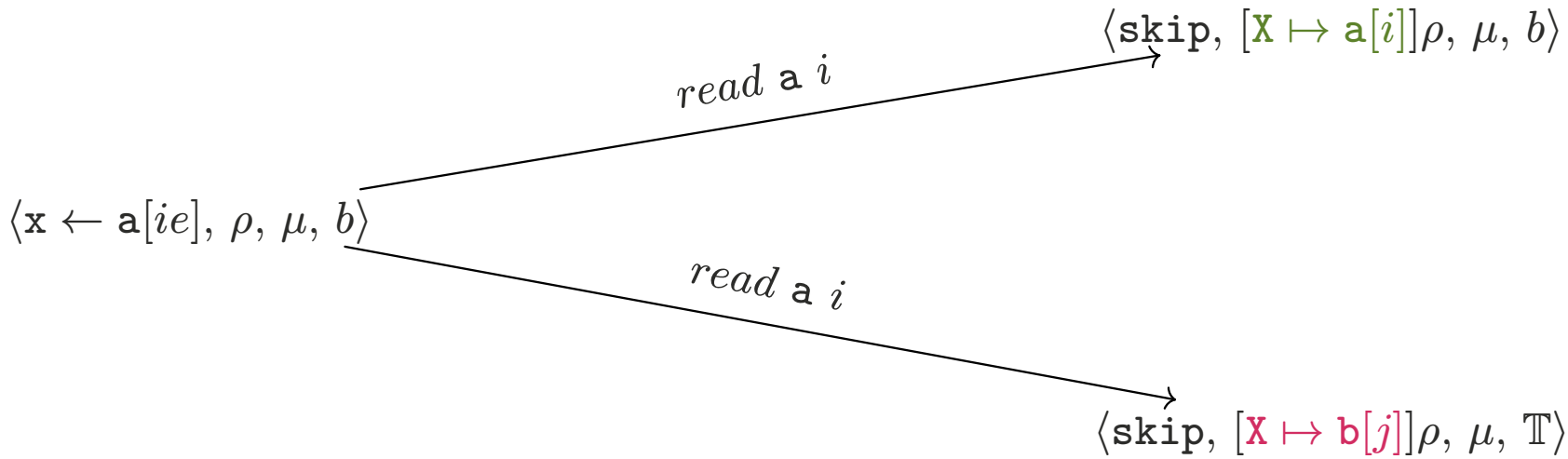
The model for the FSLH security proof is very abstract:

- Directives control branch prediction
- Attacker chooses location for out-of-bounds accesses

Forward-Only Semantics to Rollbacks

The model for the FSLH security proof is very abstract:

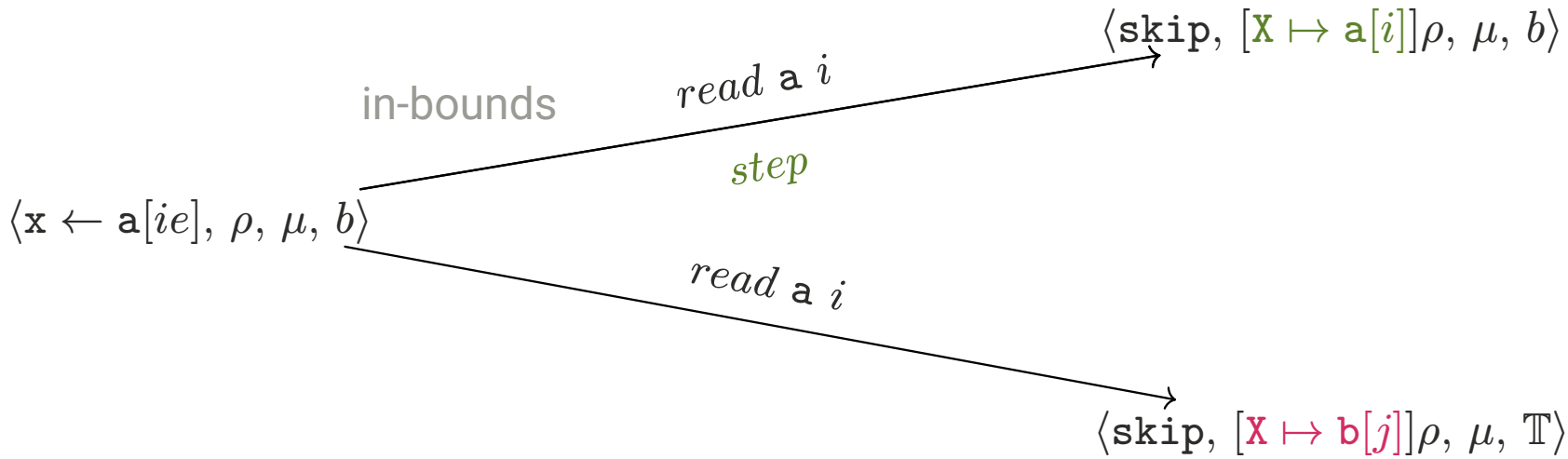
- Directives control branch prediction
- Attacker chooses location for out-of-bounds accesses



Forward-Only Semantics to Rollbacks

The model for the FSLH security proof is very abstract:

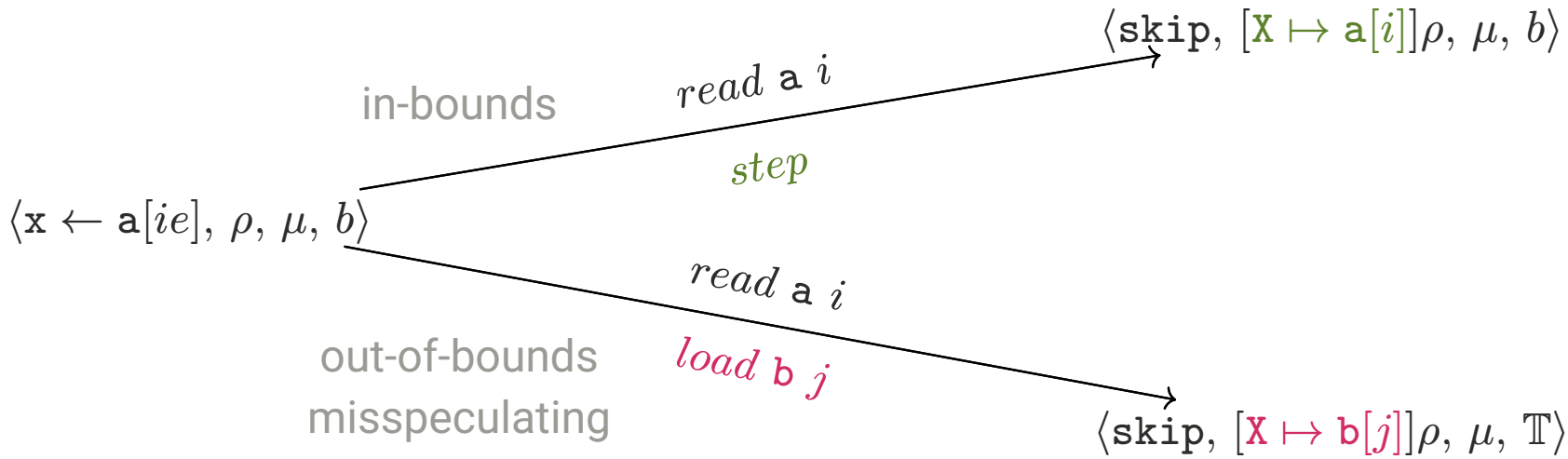
- Directives control branch prediction
- Attacker chooses location for out-of-bounds accesses



Forward-Only Semantics to Rollbacks

The model for the FSLH security proof is very abstract:

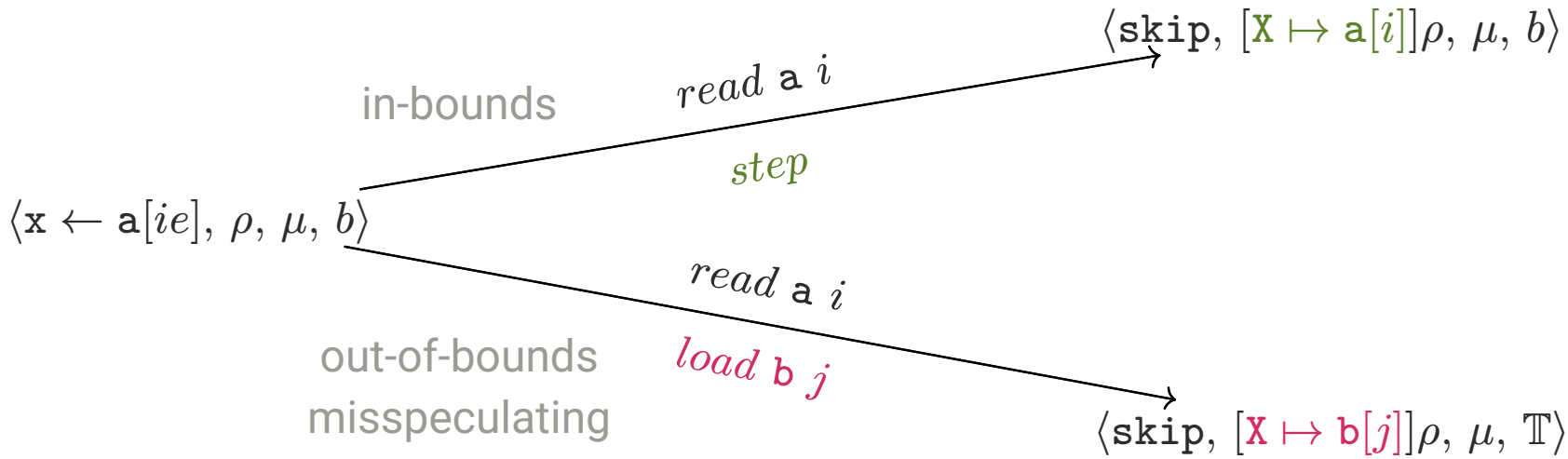
- Directives control branch prediction
- Attacker chooses location for out-of-bounds accesses



Forward-Only Semantics to Rollbacks

The model for the FSLH security proof is very abstract:

- Directives control branch prediction
- Attacker chooses location for out-of-bounds accesses
- Forward-only: No rollback mechanism, cannot leave misspeculation



Adding Rollbacks

$$\textsc{Rb\_If\_Force}\ \frac{b' = [\![be]\!]_\rho}{\langle \texttt{if}\ be\ \texttt{then}\ c_{\mathbb{T}}\ \texttt{else}\ c_{\mathbb{F}},\ \rho,\ \mu,\ b\rangle \xrightarrow[force]{branch\ b'}_{\mathrm{rb}} \langle c_{\neg b'},\ \rho,\ \mu,\ \mathbb{T}\rangle}$$

Adding Rollbacks

$$\textsc{Rb\_If\_Force}\ \frac{b' = [\![be]\!]_\rho}{\langle \texttt{if}\ be\ \texttt{then}\ c_{\mathbb{T}}\ \texttt{else}\ c_{\mathbb{F}},\ \rho,\ \mu,\ b\rangle \xrightarrow[force]{branch\ b'}_{\mathrm{rb}} \langle c_{\neg b'},\ \rho,\ \mu,\ \mathbb{T}\rangle}$$

Adding Rollbacks

$$\textsc{Rb\_If\_Force}\ \frac{b' = [\![be]\!]_\rho}{\langle \texttt{if } be \texttt{ then } c_{\mathbb{T}} \texttt{ else } c_{\mathbb{F}},\ \rho,\ \mu,\ b\rangle \xrightarrow[force]{branch\ b'}_{\mathrm{rb}} \langle c_{\neg b'},\ \rho,\ \mu,\ \mathbb{T}\rangle} \quad \langle c_{b'},\ \rho,\ \mu\rangle$$

Adding Rollbacks

$$\textsc{Rb\_If\_Force}\ \frac{b' = [\![be]\!]_\rho}{\langle \texttt{if } be \texttt{ then } c_{\mathbb{T}} \texttt{ else } c_{\mathbb{F}},\ \rho,\ \mu,\ b\rangle \xrightarrow[force]{branch\ b'}_{\mathrm{rb}} \langle c_{\neg b'},\ \rho,\ \mu,\ \mathbb{T}\rangle}\quad \langle c_{b'},\ \rho,\ \mu\rangle \ \vdots$$

$$\textsc{Rb\_Rollback}\ \frac{\langle c,\ \rho,\ \mu,\ b\rangle}{\langle c',\ \rho',\ \mu',\ b'\rangle \xrightarrow[rollback]{rollback}_{\mathrm{rb}} \langle c',\ \rho',\ \mu',\ b'\rangle}\quad \vdots$$

Rollbacks Are Not Needed (Barthe et al. 2021)

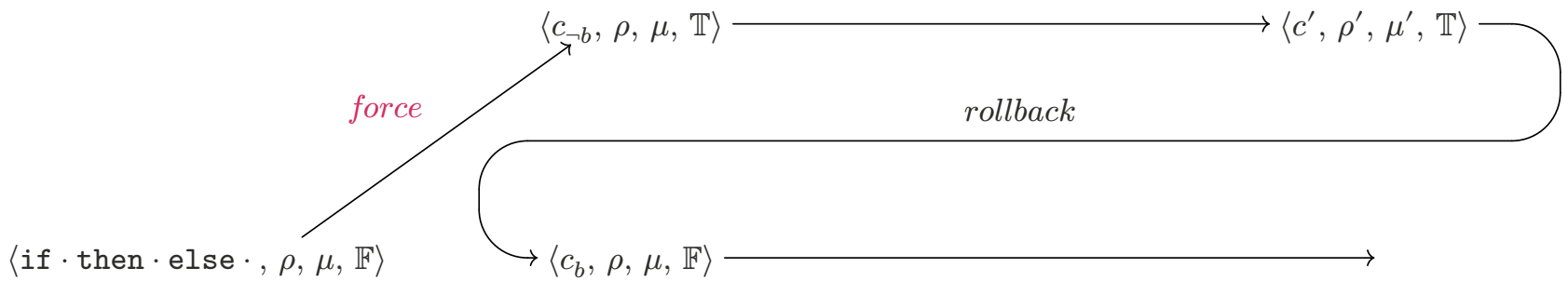
- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?

Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage

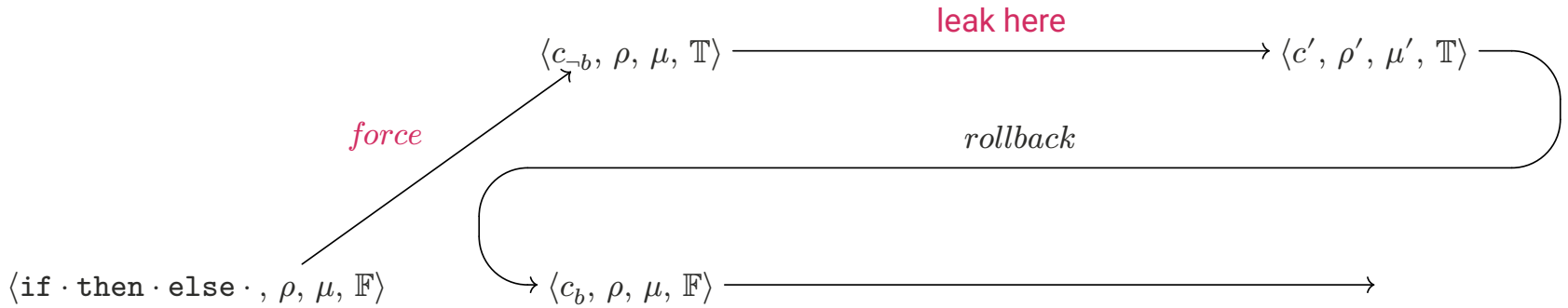
Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage



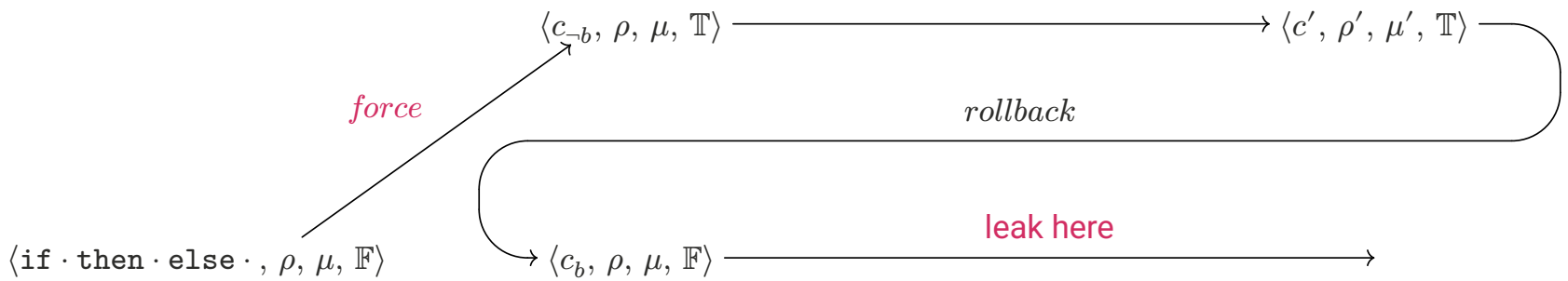
Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage



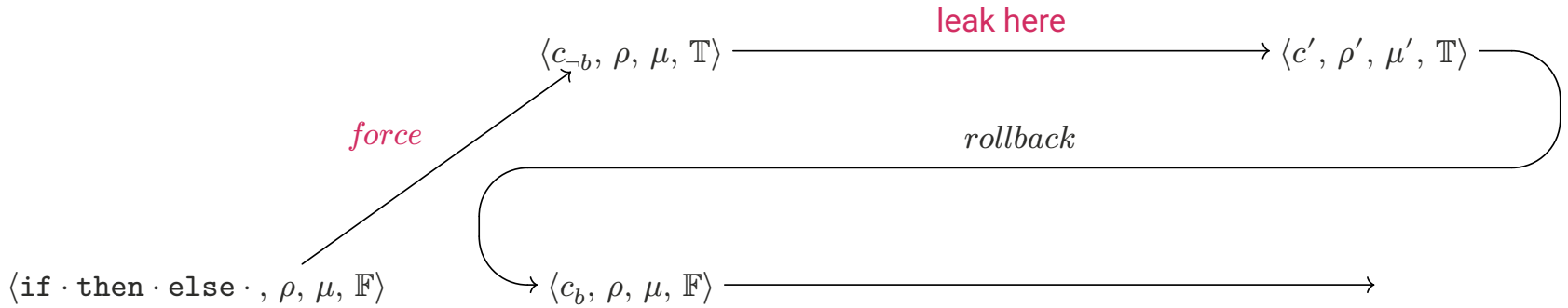
Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage



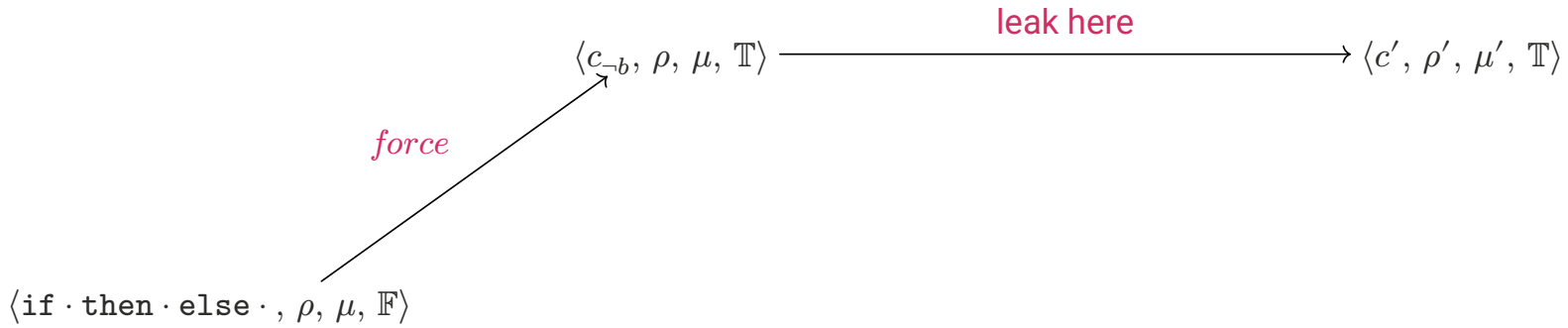
Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage



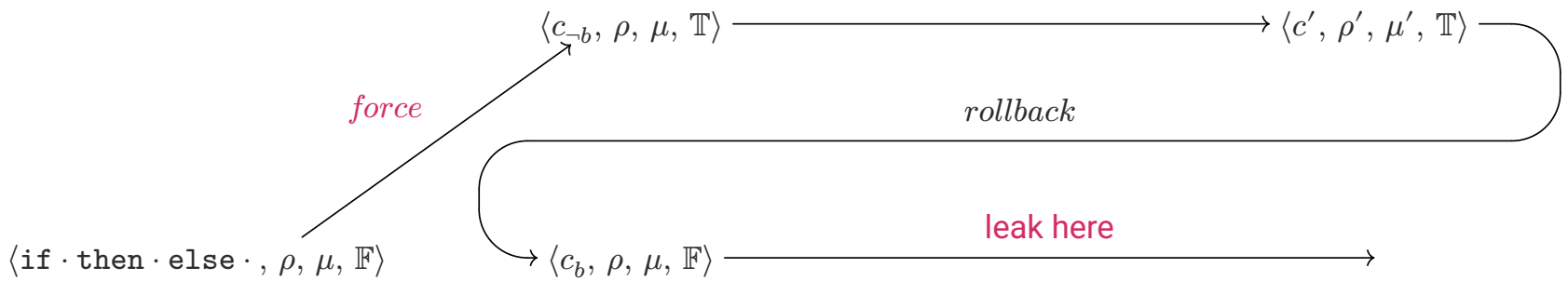
Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage



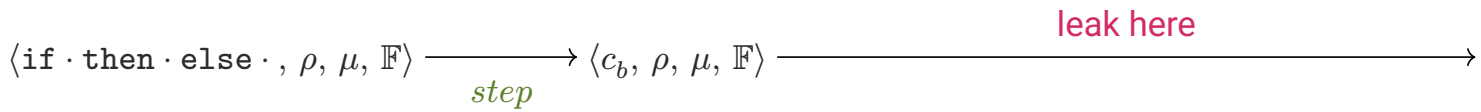
Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage



Rollbacks Are Not Needed (Barthe et al. 2021)

- Equal leakage in forward-only semantics $\Rightarrow$ equal leakage with rollbacks?
- Different leakage with rollbacks $\Rightarrow$ different forward-only leakage



Always-Mispredict Semantics

- unlimited misspeculation

Always-Mispredict Semantics

- unlimited misspeculation
- limited speculation window

Always-Mispredict Semantics

- unlimited misspeculation
- attacker-chosen memory locations

- limited speculation window

Always-Mispredict Semantics

- unlimited misspeculation
- attacker-chosen memory locations

- limited speculation window
- flat memory layout

Always-Mispredict Semantics

- unlimited misspeculation
- attacker-chosen memory locations
- attacker-controlled misspeculation

- limited speculation window
- flat memory layout

Always-Mispredict Semantics

- unlimited misspeculation
- attacker-chosen memory locations
- attacker-controlled misspeculation

- limited speculation window
- flat memory layout
- always mispredict

Always-Mispredict Semantics

- unlimited misspeculation
- attacker-chosen memory locations
- attacker-controlled misspeculation

- limited speculation window
- flat memory layout
- always mispredict

How is this more realistic?

Always-Mispredict Semantics

- unlimited misspeculation
- attacker-chosen memory locations
- attacker-controlled misspeculation

- limited speculation window
- flat memory layout
- always mispredict

How is this more realistic?

- proposed for use in Hardware-Software Contracts (Guarnieri et al. 2021)

Always-Mispredict Semantics

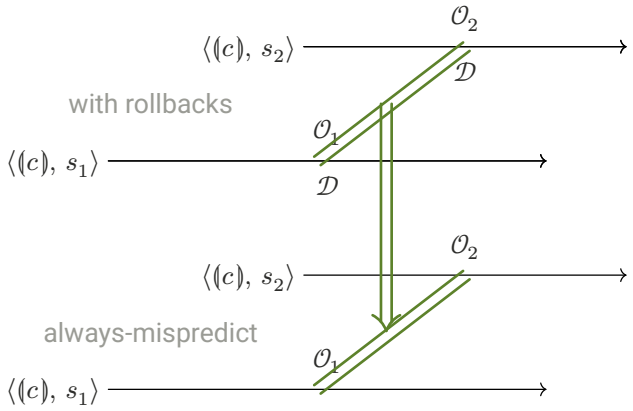
- unlimited misspeculation
- attacker-chosen memory locations
- attacker-controlled misspeculation

- limited speculation window
- flat memory layout
- always mispredict

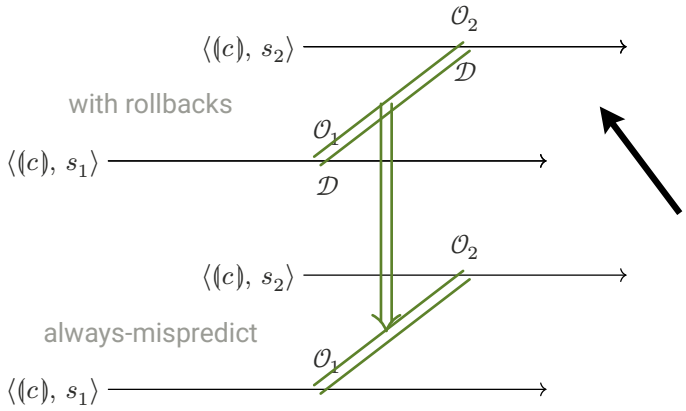
How is this more realistic?

- proposed for use in Hardware-Software Contracts (Guarnieri et al. 2021)
  - *idea:* vendor-guaranteed leakage model

From Directives to Always-Mispredict

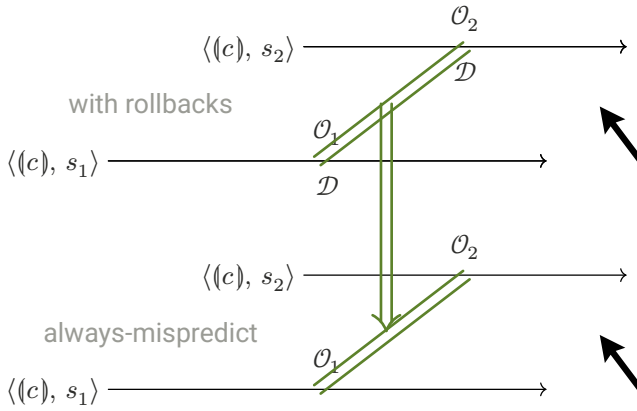


From Directives to Always-Mispredict



out-of-bounds accesses only in misspeculating execution

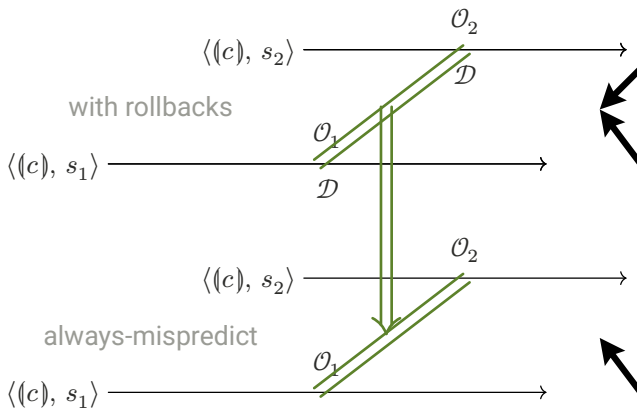
From Directives to Always-Mispredict



out-of-bounds accesses only in misspeculating execution

flattened memory layout independent of speculation

From Directives to Always-Mispredict

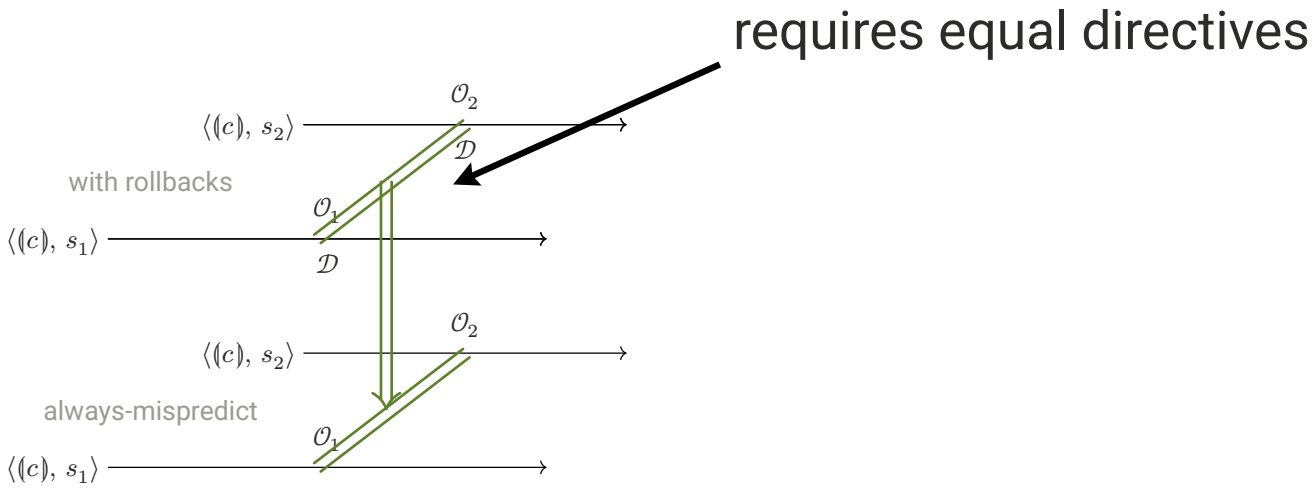


must assume sequential execution does not get stuck!

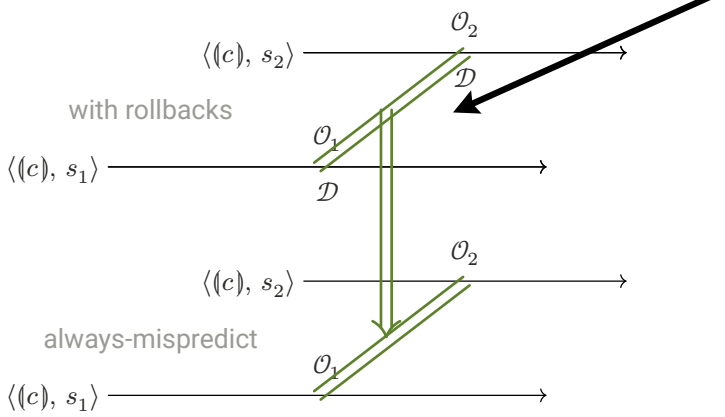
out-of-bounds accesses only in misspeculating execution

flattened memory layout independent of speculation

From Directives to Always-Mispredict



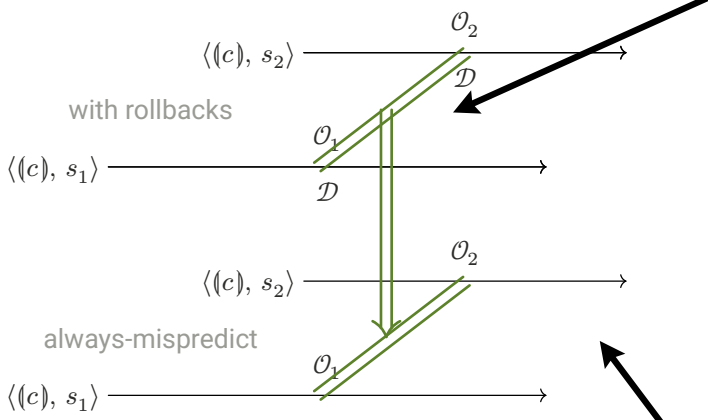
From Directives to Always-Mispredict



requires equal directives

- same out-of-bounds index must result in same location

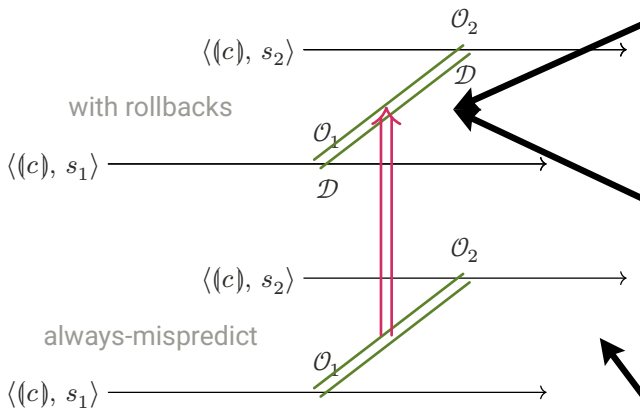
From Directives to Always-Mispredict



requires equal directives

memory access:
One might be in-bounds
and the other out-of-bounds

From Directives to Always-Mispredict

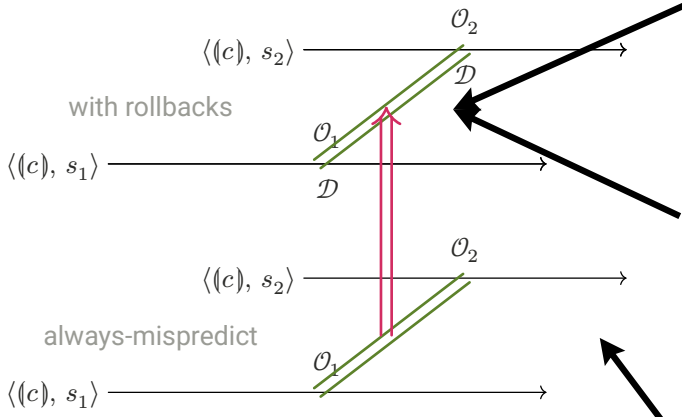


requires equal directives

no equality assumption
when different directives are used

memory access:
One might be in-bounds
and the other out-of-bounds

From Directives to Always-Mispredict



requires equal directives

one directive must allow
both in-bounds and out-of-bounds access

memory access:
One might be in-bounds
and the other out-of-bounds

From Directives to Always-Mispredict: Lessons Learned

- Models can make implicit safety assumptions

From Directives to Always-Mispredict: Lessons Learned

- Models can make implicit safety assumptions
  - undefined behaviour might be intentionally out-of-scope

From Directives to Always-Mispredict: Lessons Learned

- Models can make implicit safety assumptions
  - undefined behaviour might be intentionally out-of-scope
  - but this should be explicit!

From Directives to Always-Mispredict: Lessons Learned

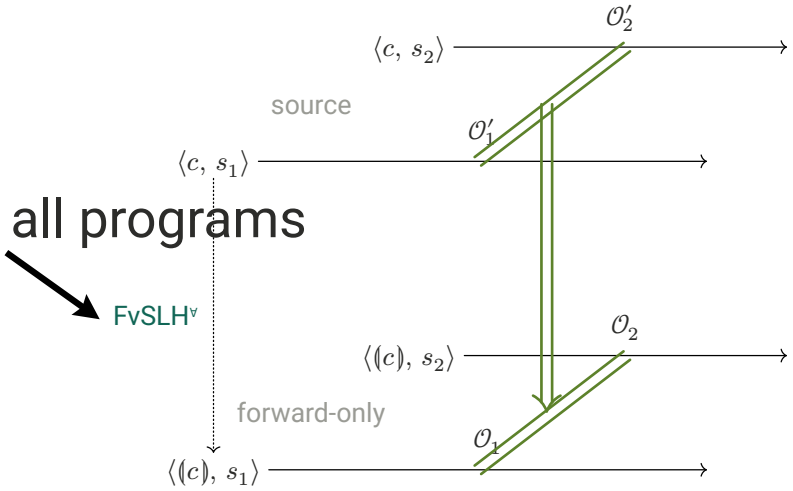
- Models can make implicit safety assumptions
  - undefined behaviour might be intentionally out-of-scope
  - but this should be explicit!
- Directive-based models must have a directive allowing both in-bounds and out-of-bounds access

From Directives to Always-Mispredict: Lessons Learned

- Models can make implicit safety assumptions
  - undefined behaviour might be intentionally out-of-scope
  - but this should be explicit!
- Directive-based models must have a directive allowing both in-bounds and out-of-bounds access
  - Easy to miss! Affects proofs of Selective and Flexible SLH

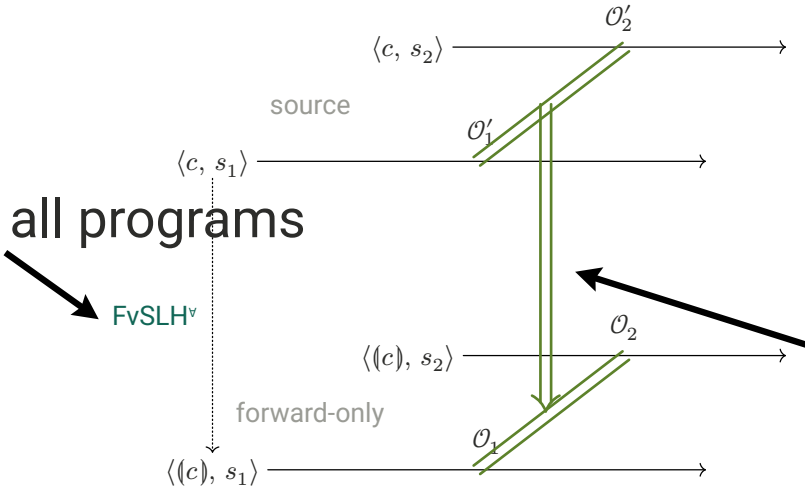
Conclusions

Efficient Mitigation for all programs



Conclusions

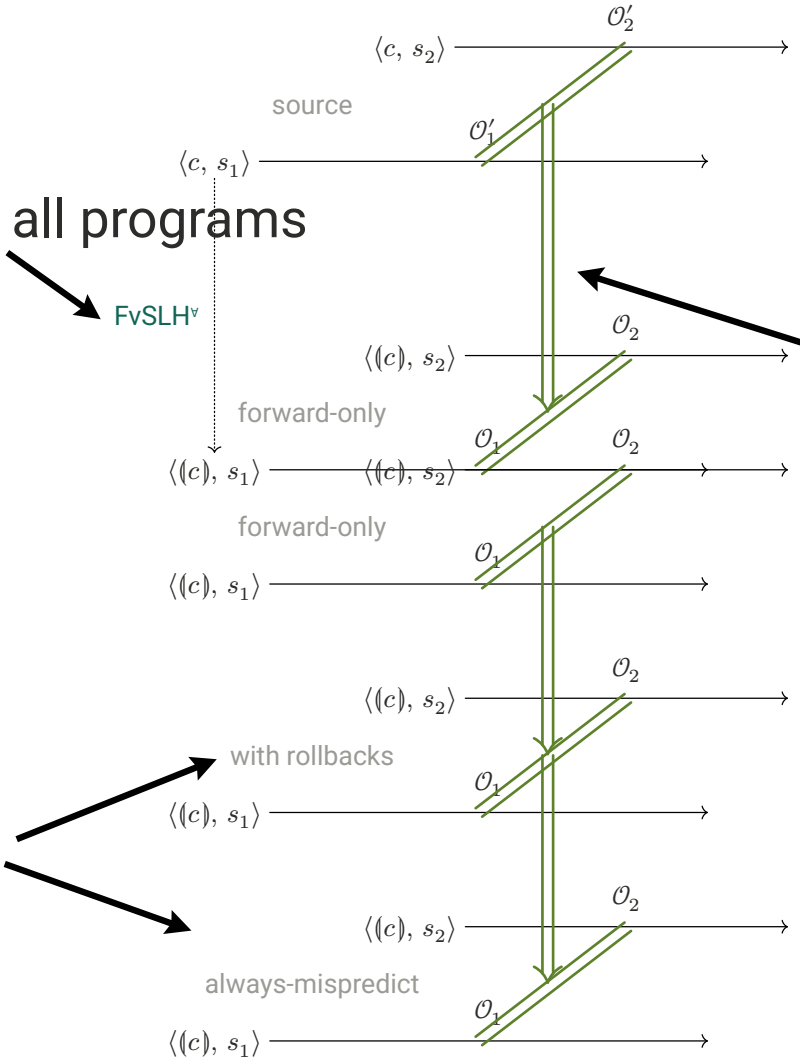
Efficient Mitigation for all programs



Formal relative security proof in Rocq

Conclusions

Efficient Mitigation for all programs



Formal relative security proof in Rocq

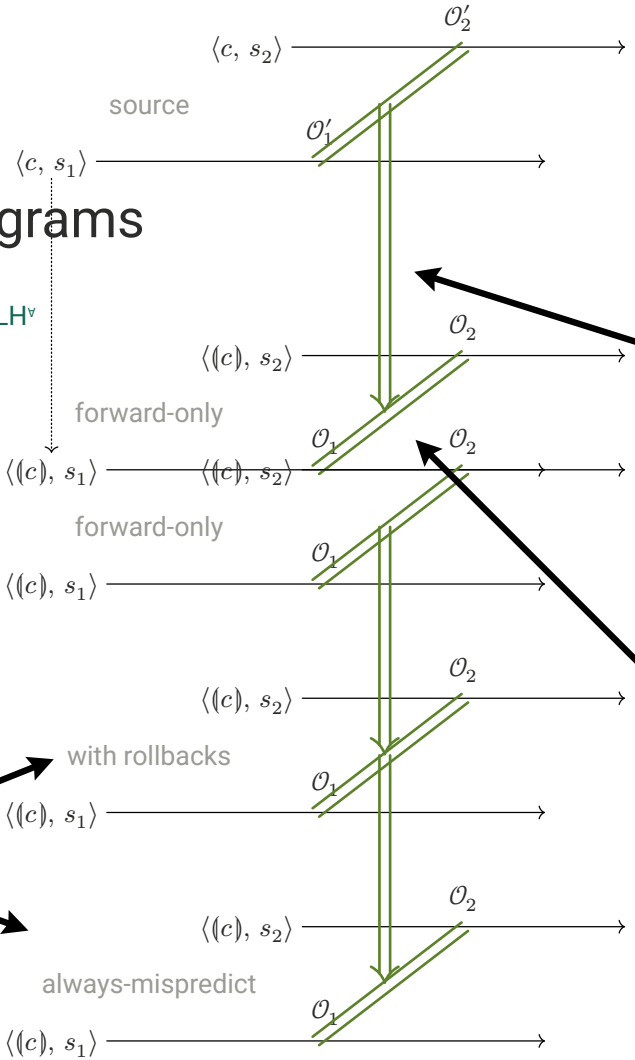
Proofs for more precise models (also in Rocq)

Conclusions

Efficient Mitigation for all programs

FvSLH$^{\forall}$

Proofs for more precise models (also in Rocq)



Formal relative security proof in Rocq

work needed to connect here

Future Work

- Real-world implementation of Flexible SLH

Future Work

- Real-world implementation of Flexible SLH
  - open questions: when during compilation to perform analysis?

Future Work

- Real-world implementation of Flexible SLH
  - open questions: when during compilation to perform analysis?
- Mitigations for other SPECTRE variants

Future Work

- Real-world implementation of Flexible SLH
  - open questions: when during compilation to perform analysis?
- Mitigations for other SPECTRE variants
  - e.g. prediction of indirect branch targets and return addresses

Future Work

- Real-world implementation of Flexible SLH
  - open questions: when during compilation to perform analysis?
- Mitigations for other SPECTRE variants
  - e.g. prediction of indirect branch targets and return addresses
- Ever more accurate models

Future Work

- Real-world implementation of Flexible SLH
  - open questions: when during compilation to perform analysis?
- Mitigations for other SPECTRE variants
  - e.g. prediction of indirect branch targets and return addresses
- Ever more accurate models
  - Hardware-Software Contracts?

Future Work

- Real-world implementation of Flexible SLH
  - open questions: when during compilation to perform analysis?
- Mitigations for other SPECTRE variants
  - e.g. prediction of indirect branch targets and return addresses
- Ever more accurate models
  - Hardware-Software Contracts?
  - dynamic attackers?

Bibliography

Barthe, Gilles, Sunjay Cauligi, Benjamin Grégoire, Adrien Koutsos, Kevin Liao, Tiago Oliveira, Swarn Priya, Tamara Rezk, and Peter Schwabe. 2021. “High-Assurance Cryptography in the Spectre Era”. In *42nd IEEE Symposium on Security and Privacy, SP*, 1884–1901. IEEE. https://doi.org/10.1109/SP40001.2021.00046.

Guarnieri, Marco, Boris Köpf, Jan Reineke, and Pepe Vila. 2021. “Hardware-Software Contracts for Secure Speculation”. In *42nd IEEE Symposium on Security and Privacy, SP*, 1868–83. IEEE. https://doi.org/10.1109/SP40001.2021.00036.

Shivakumar, Basavesh Ammanaghatta, Jack Barnes, Gilles Barthe, Sunjay Cauligi, Chitchanok Chuengsatiansup, Daniel Genkin, Sioli O'Connell, Peter Schwabe, Rui Qi Sim, and Yuval Yarom. 2023. “Spectre Declassified: Reading from the Right Place at the Wrong Time”. In *44th IEEE Symposium on Security and Privacy, SP*, 1753–70. IEEE. https://doi.org/10.1109/SP46215.2023.10179355.