Indistinguishability Obfuscation via Mathematical Proofs of Equivalence

Abhishek Jain

Johns Hopkins University

Zhengzhong Jin

Johns Hopkins University → MIT

Program Obfuscation

```
function main() {
  console.log('hello, world');
}
main()
```

```
function _0x19e6(_0x4d301f,_0xcaab53){var _0x3a4e72=_0x3a4e();return
_0x19e6=function(_0x19e691,_0x5809f0){_0x19e691=_0x19e691-0x14e;var
_0x16ee0b=_0x3a4e72[_0x19e691];return
_0x16ee0b;},_0x19e6(_0x4d301f,_0xcaab53);}function _0x3a4e(){var _0x3f0a9d=
['log','199381NCGrSa','2328491tAiNSg','18mVqyqS','4cVQTsk','6PuGzwR','107410
32WsiTVO','104321yYIIVM','370911DTLqdw','10uRQffV','2024504eEkwnt','114dOcOh
j','hello,\x20world','2634710IatlOd'];_0x3a4e=function(){return
_0x3f0a9d;};return _0x3a4e();}(function(_0x3d9e47,_0x360e03){var
_0x3afd0b=_0x19e6,_0x2928d3=_0x3d9e47();while(!![]){try{var _0x33cc3a=-
parseInt(_0x3afd0b(0x15a))/0x1*(-parseInt(_0x3afd0b(0x158))/0x2)+-
parseInt(_0x3afd0b(0x15b))/0x3*(-parseInt(_0x3afd0b(0x157))/0x4)+-
parseInt(_0x3afd0b(0x152))/0x5+parseInt(_0x3afd0b(0x150))/0x6*
(parseInt(_0x3afd0b(0x154))/0x7)+-parseInt(_0x3afd0b(0x14f))/0x8*(-
parseInt(_0x3afd0b(0x156))/0x9)+parseInt(_0x3afd0b(0x14e))/0xa*
(parseInt(_0x3afd0b(0x155))/0xb)+-
parseInt(_0x3afd0b(0x159))/0xc;if(_0x33cc3a===_0x360e03)break;else
_0x2928d3['push'](_0x2928d3['shift']());}catch(_0x437e27){_0x2928d3['push']
(_0x2928d3['shift']());}}}(_0x3a4e,0x42c94));function main(){var
_0x29ace6=_0x19e6;console[_0x29ace6(0x153)](_0x29ace6(0x151));}main();
```

Indistinguishability Obfuscation (iO)

C: Program (Circuit/Turing Machine)

Indistinguishability Obfuscation (iO)

C: Program (Circuit/Turing Machine)

iO: obfuscator $\quad iO(1^\lambda, C) \to C'$, $\quad$ (λ: Security parameter)

Indistinguishability Obfuscation (iO)

C: Program (Circuit/Turing Machine)

iO: obfuscator $iO(1^\lambda, C) \rightarrow C'$, ($\lambda$: Security parameter)

- **Preserve Functionality:**

$$\forall\, x, C'(x) = C(x)$$

Indistinguishability Obfuscation (iO)

C: Program (Circuit/Turing Machine)

iO: obfuscator $\quad iO(1^\lambda, C) \rightarrow C'$, $\quad$ (λ: Security parameter)

- **Preserve Functionality:**

$$\forall\, x, C'(x) = C(x)$$

- **Indistinguishability Security:** for any C_0, C_1

$$\forall\, x\ C_0(x) = C_1(x), \quad iO(1^\lambda, C_0) \approx_c iO(1^\lambda, C_1)$$

Indistinguishability Security, as a **Game**

Indistinguishability Security, as a **Game**

n.u.
Probabilistic
Poly.-time

Indistinguishability Security, as a **Game**

n.u.
Probabilistic
Poly.-time

Indistinguishability Security, as a **Game**

$C_0, C_1 \longrightarrow$

n.u.
Probabilistic
Poly.-time

Indistinguishability Security, as a **Game**

n.u.
Probabilistic
Poly.-time

$\xrightarrow{C_0, C_1}$

$b \leftarrow \{0,1\}$

Indistinguishability Security, as a **Game**

n.u.
Probabilistic
Poly.-time

C_0, C_1 →

← $iO(1^\lambda, C_b)$

$b \leftarrow \{0,1\}$

Indistinguishability Security, as a **Game**

n.u.
Probabilistic
Poly.-time

$$\xrightarrow{\color{red}{C_0}, \color{blue}{C_1}}$$

$$\xleftarrow{iO(1^\lambda, C_b)}$$

$$\xrightarrow{b' \in \{0,1\}}$$

$b \leftarrow \{0,1\}$

Indistinguishability Security, as a **Game**

C_0, C_1 →

n.u.
Probabilistic
Poly.-time

$b \leftarrow \{0,1\}$

← $iO(1^\lambda, C_b)$

$b' \in \{0,1\}$ →

If $\forall \boldsymbol{x}\ \boldsymbol{C_0}(\boldsymbol{x}) = \boldsymbol{C_1}(\boldsymbol{x}) \wedge$
$b = b'$, adversary wins

Indistinguishability Security, as a **Game**

n.u.
Probabilistic
Poly.-time

$C_0, C_1 \longrightarrow$

$iO(1^\lambda, C_b) \longleftarrow$

$b' \in \{0,1\} \longrightarrow$

$b \leftarrow \{0,1\}$

If $\forall \boldsymbol{x}\ \boldsymbol{C_0(x) = C_1(x)} \wedge$
$b = b'$, adversary wins

$$\boxed{\Pr[\text{Adversary } wins] \leq \text{negl}\ (\lambda)}$$

Indistinguishability Security, as a **Game**

n.u.
Probabilistic
Poly.-time

C_0, C_1 →

$iO(1^\lambda, C_b)$ ←

$b' \in \{0,1\}$ →

$b \leftarrow \{0,1\}$

If $\forall \boldsymbol{x}\ \boldsymbol{C_0(x) = C_1(x)} \wedge$
$b = b'$, adversary wins

<u>Non-Falsifiability</u>
The challenger (judger) is inefficient.
($2^{|x|}$-time)

Good Assumptions are **Falsifiable**

Good Assumptions are **Falsifiable**

<u>Example: Factoring</u>

Good Assumptions are **Falsifiable**

Example: Factoring

$p, q \leftarrow Primes$

Good Assumptions are **Falsifiable**

Example: Factoring

$p, q \leftarrow Primes$

$N = p \cdot q$

Good Assumptions are **Falsifiable**

<u>Example: Factoring</u>

$p, q \leftarrow Primes$

$N = p \cdot q$

$\longleftarrow$

p', q'

$\longrightarrow$

Good Assumptions are **Falsifiable**

Example: Factoring

$p, q \leftarrow Primes$

$N = p \cdot q$

$\longleftarrow$

p', q'

$\longrightarrow$

If $N = p' \cdot q'$, adversary wins

Good Assumptions are **Falsifiable**

Example: Factoring

$p, q \leftarrow Primes$

$N = p \cdot q$

$\longleftarrow$

p', q'

$\longrightarrow$

If $N = p' \cdot q'$, adversary wins

Falsifiable

The challenger is poly-time

Base iO on Good Assumptions?

Base iO on Good Assumptions?

- A long line of works:

  [Garg-Gentry-Halevi-Raykova-Sahai-Waters’13][Pass-Seth-Telang’14]
  [Gentry-Lewko-Sahai-Waters’15][Ananth-Jain’15][Bitansky-Vaikuntanathan’15]
  [Lin’16][Lin-Vaikuntanathan’16][Lin-Pass-Karn Seth-Telang’16]
  [Garg-Miles-Mukherjee-Sahai-Srinivasan-Zhandry’16][Ananth-Sahai’17][Lin’17]
  [Lin-Tessaro’17][Agrawal’19][Jain-Lin-Matt-Sahai’19][Brakerski-Dottling-Malavolta’20]…

Base iO on Good Assumptions?

- A long line of works:
  [Garg-Gentry-Halevi-Raykova-Sahai-Waters'13][Pass-Seth-Telang'14]
  [Gentry-Lewko-Sahai-Waters'15][Ananth-Jain'15][Bitansky-Vaikuntanathan'15]
  [Lin'16][Lin-Vaikuntanathan'16][Lin-Pass-Karn Seth-Telang'16]
  [Garg-Miles-Mukherjee-Sahai-Srinivasan-Zhandry'16][Ananth-Sahai'17][Lin'17]
  [Lin-Tessaro'17][Agrawal'19][Jain-Lin-Matt-Sahai'19][Brakerski-Dottling-Malavolta'20]…
- **iO from Well-Founded Assumptions** [Jain-Lin-Sahai'20]
  Based on **Sub-exponential Security** of Learning with Errors, and Learning Parity with Noise and more…

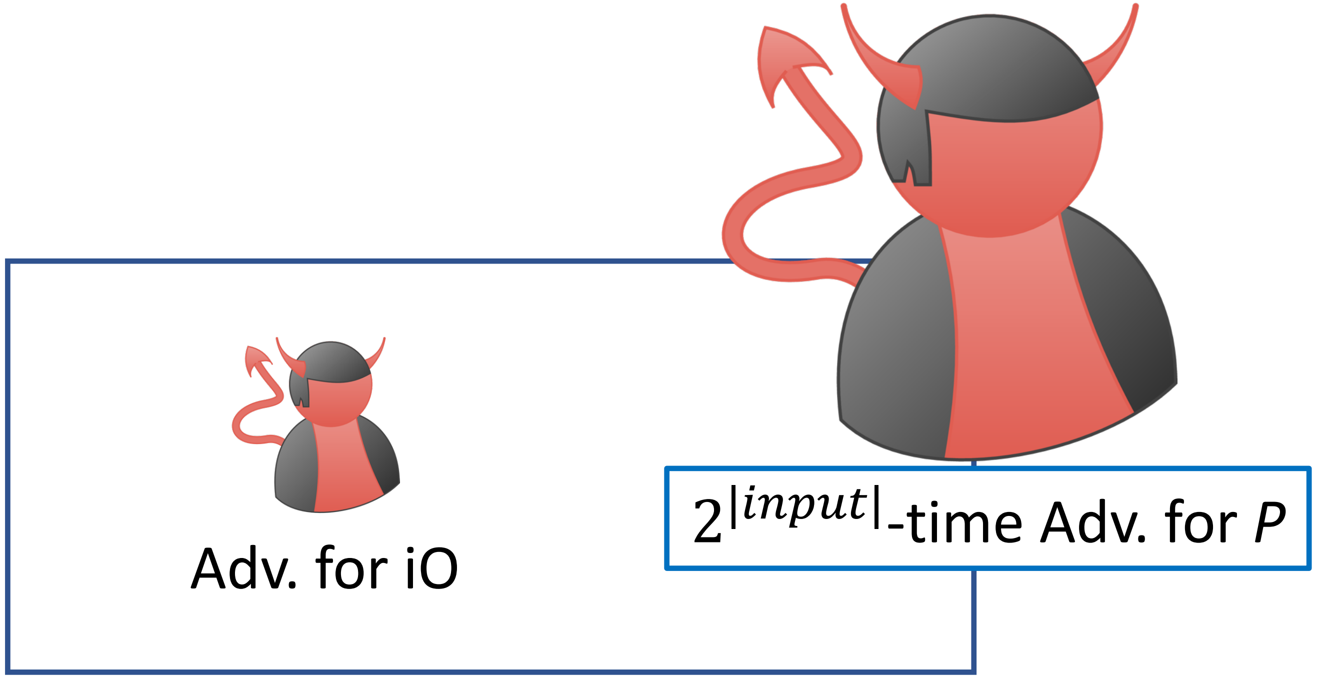
Sub-exponential Security

Of an Assumption P

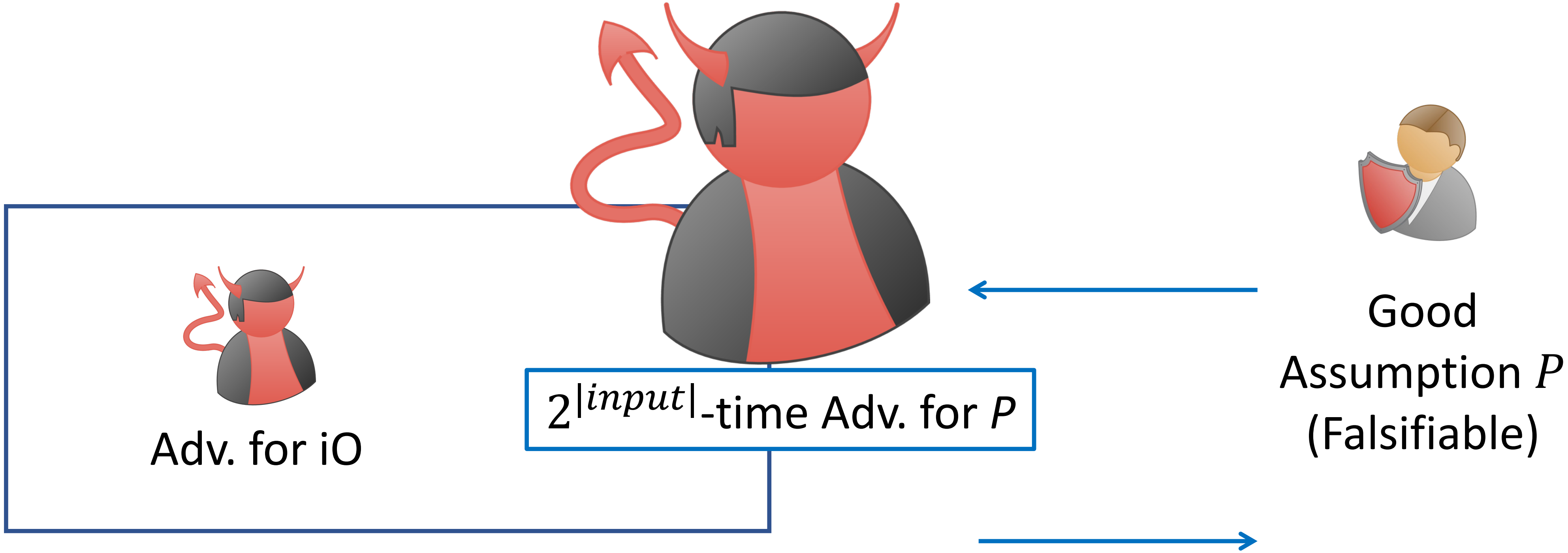
For any adversary that runs in 2^{λ^c}-time ($0 < c < 1$),
it can only break the Assumption P of size λ with negligible probability.

(P=Learning with Errors, Learning party with Noise, …)

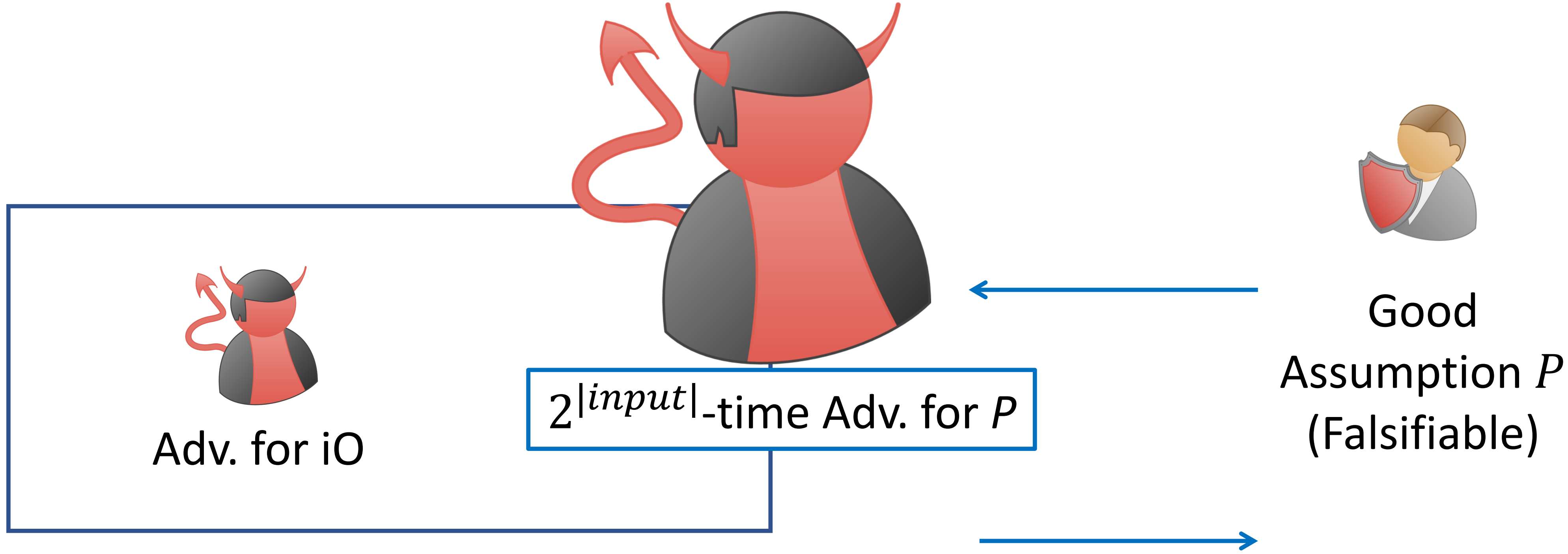
$2^{|input|}$-Loss in Reduction



$2^{|input|}$-Loss in Reduction

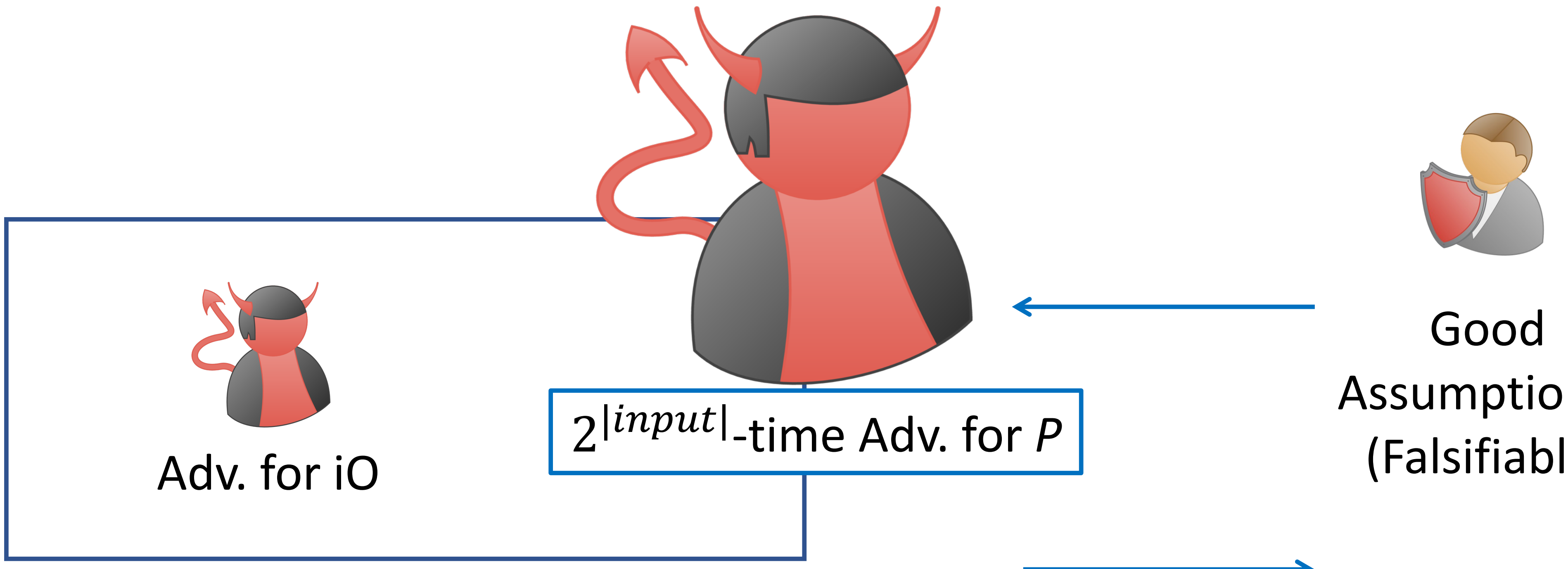


$2^{|input|}$-Loss in Reduction



Assume 2^{λ^c}-Security & set $2^{\lambda^c} > 2^{|input|}$

$2^{|input|}$-Loss in Reduction



Assume 2^{λ^c}-Security & set $2^{\lambda^c} > 2^{|input|}$

$|\boldsymbol{input}| < \boldsymbol{\lambda^c}$

$2^{|input|}$-Security Loss is Bad

$2^{|input|}$-Security Loss is Bad

iO for Turing Machines: M: a Turing Machine, $iO(1^\lambda, M) \to M'$

$2^{|input|}$-Security Loss is Bad

iO for Turing Machines: M: a Turing Machine, $iO(1^\lambda, M) \rightarrow M'$

Ideal: M' works for **unbounded input-length**

$2^{|input|}$-Security Loss is Bad

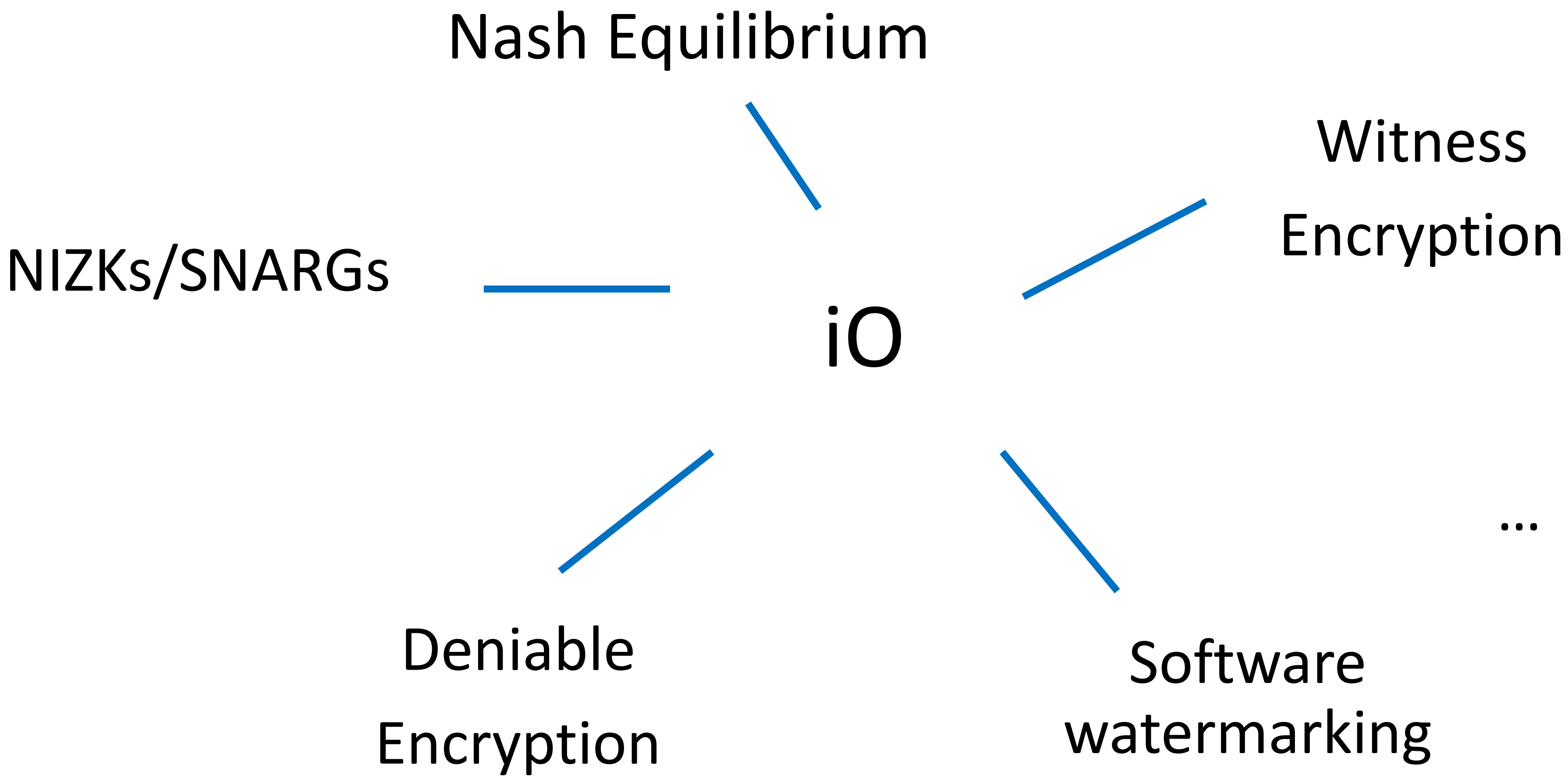
iO for Turing Machines: M: a Turing Machine, $iO(1^\lambda, M) \to M'$

Ideal: M' works for **unbounded input-length**

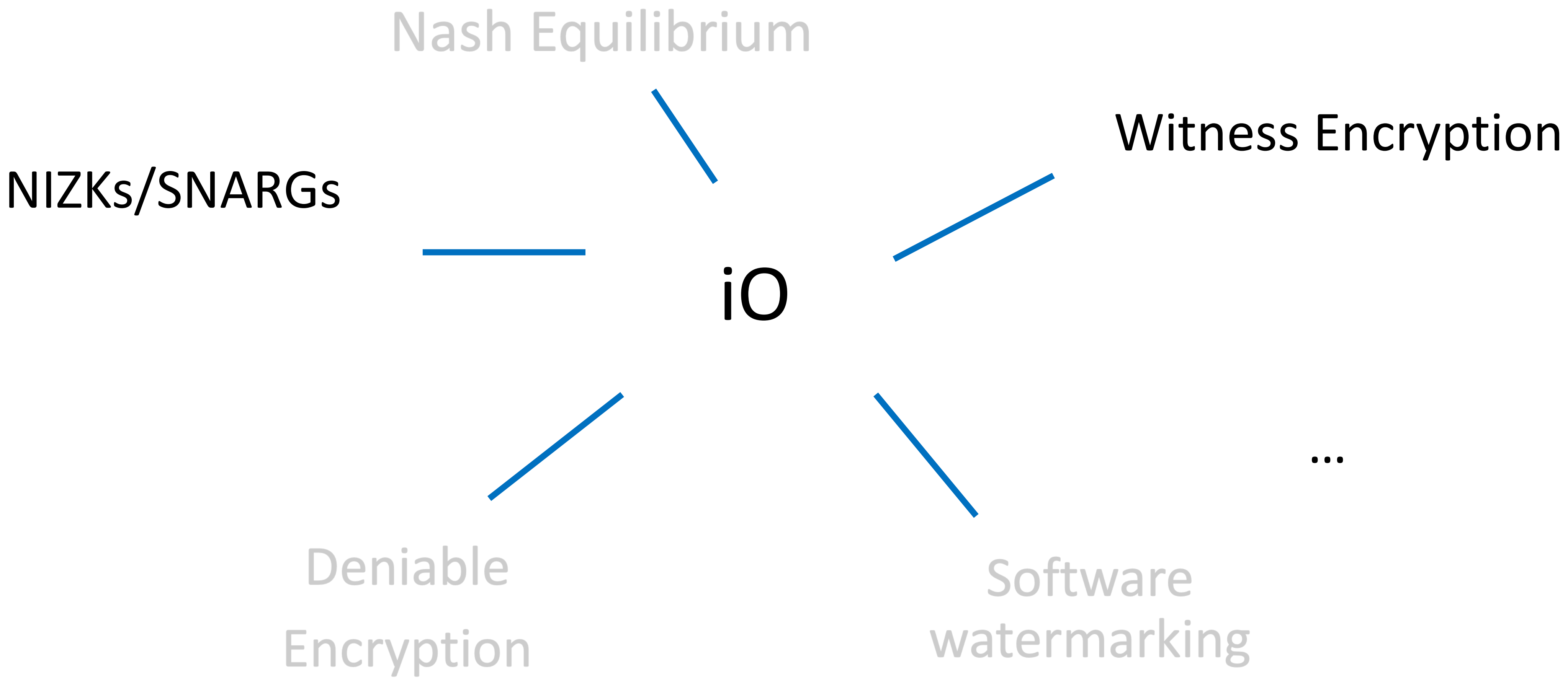
Reality:
Input length is **a-priori bounded (since** $|input| < \lambda^c$**)**

[Bitansky-Garg-Lin-Pass-Telang'15][Canetti-Holmgren-Jain-Vaikuntanathan'15][Koppula-Lewko-Waters'15]…

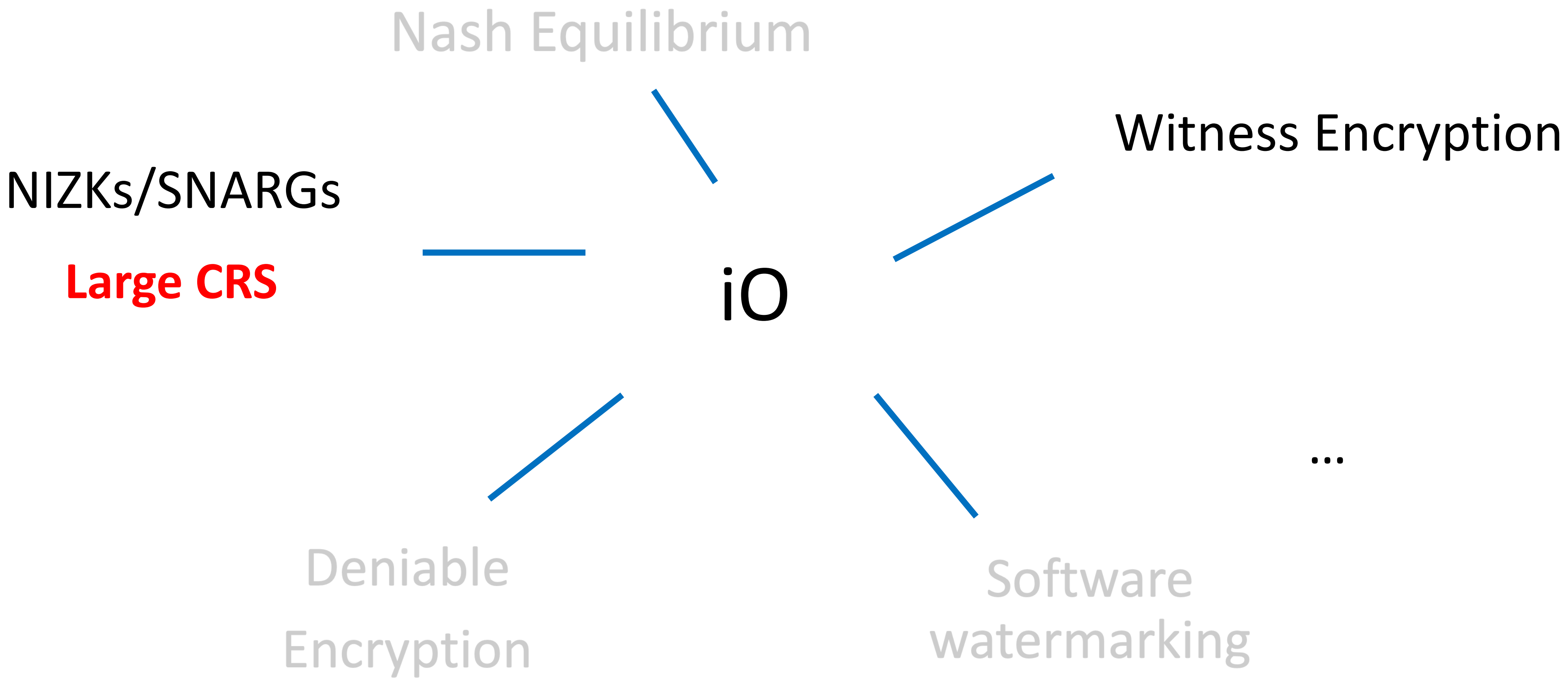
iO: the "Central Hub" [Sahai-Waters'13]



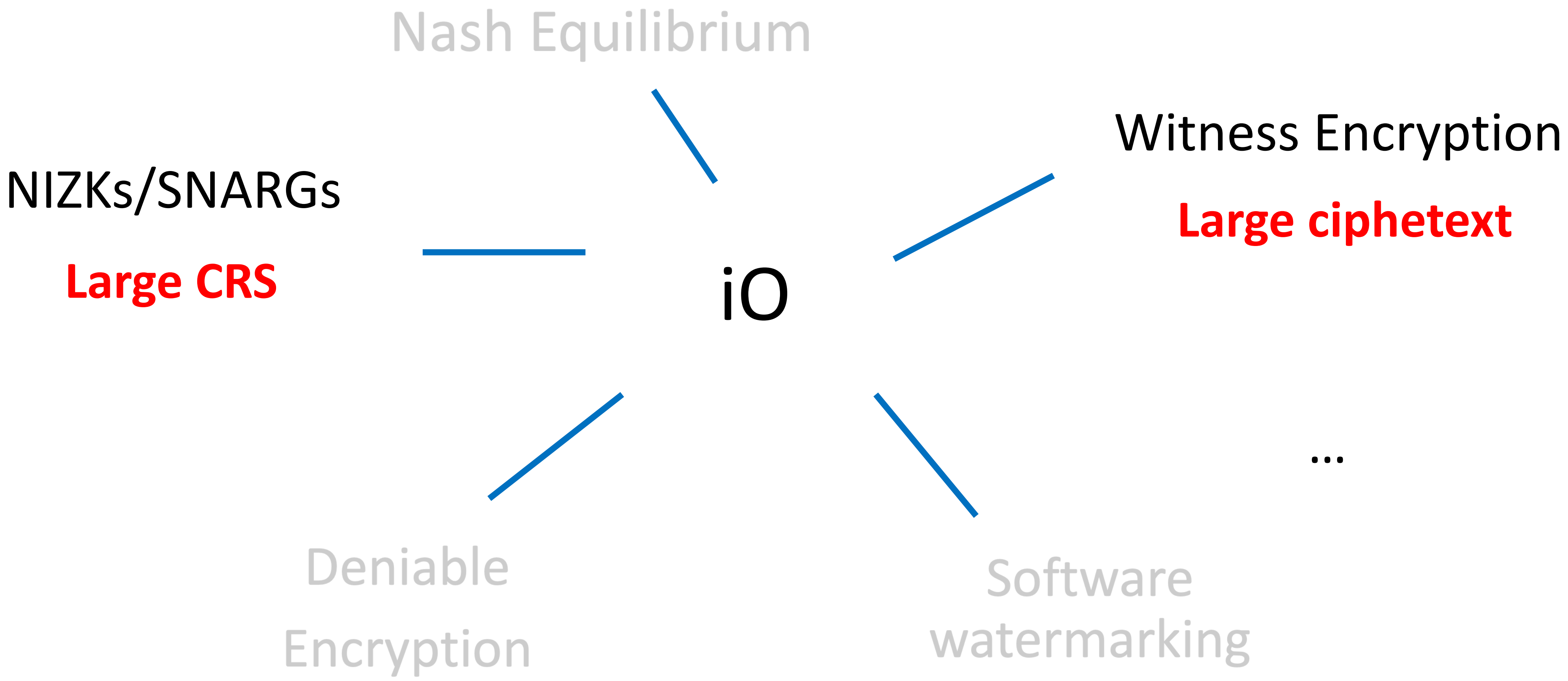
$2^{|input|}$-Security Loss "Spreads"



$2^{|input|}$-Security Loss “Spreads”

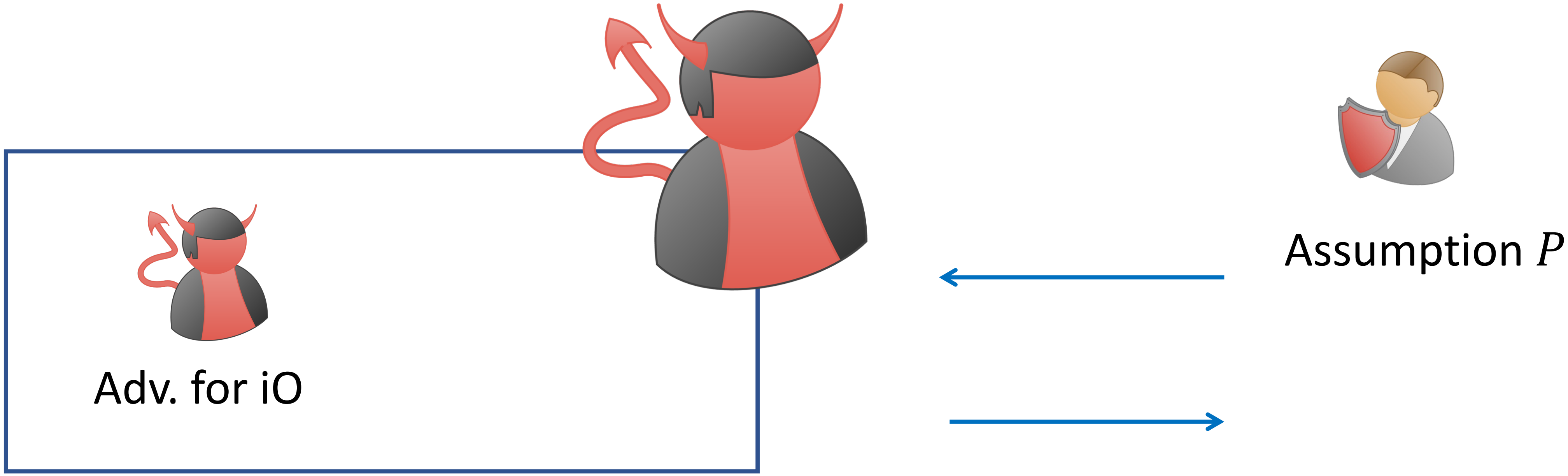


$2^{|input|}$-Security Loss "Spreads"

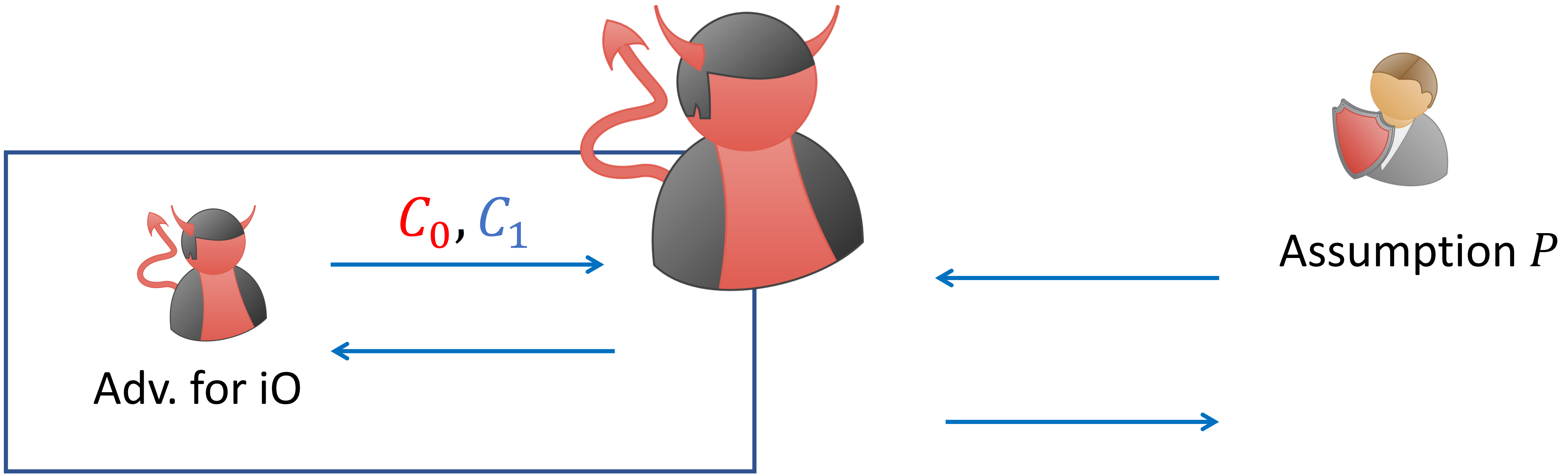


Question: Can we build iO with a security loss ***independent*** of the input length?

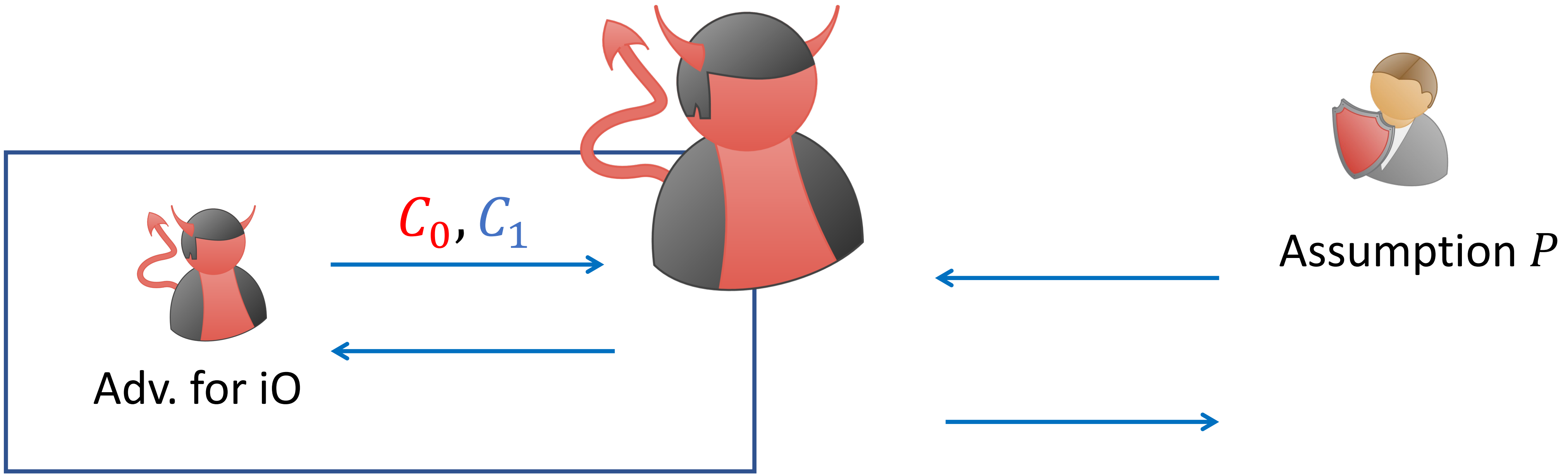
Is $2^{|input|}$-Loss Inherent? (folklore)



Is $2^{|input|}$-Loss Inherent? (folklore)

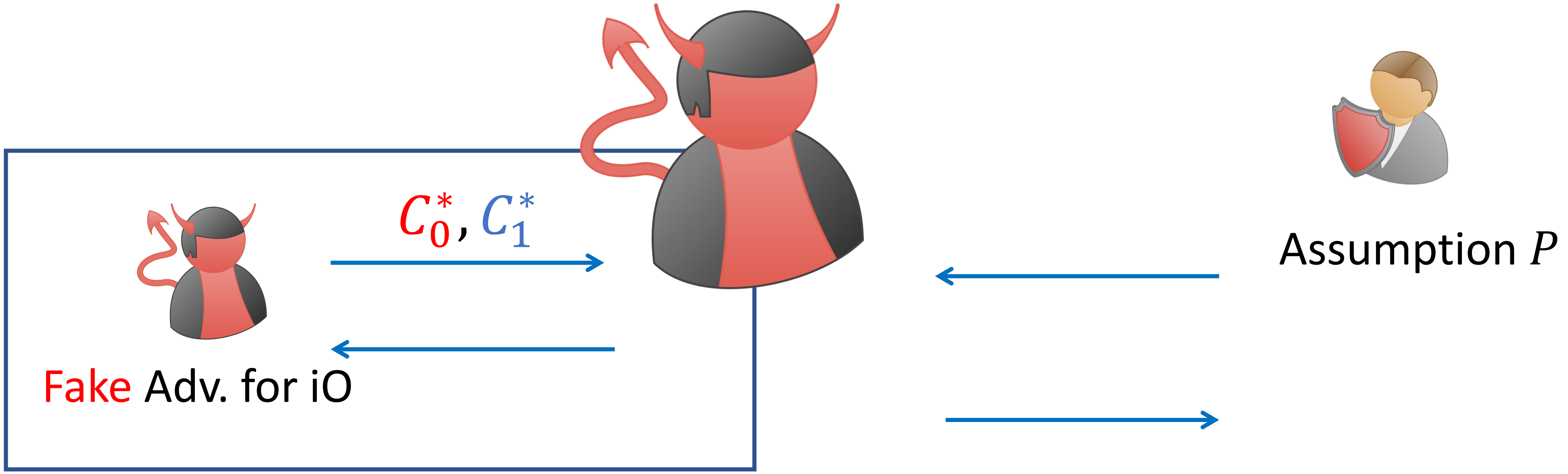


Is $2^{|input|}$-Loss Inherent? (folklore)

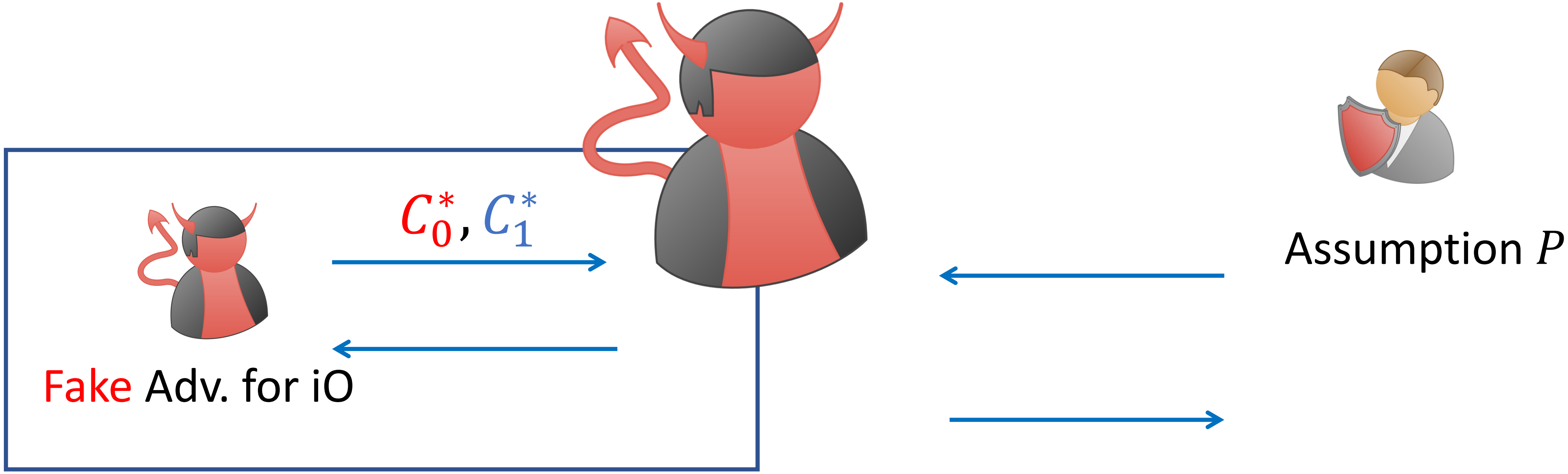


If $\forall x\ C_0(x) = C_1(x)$, then reduction break P.

Is $2^{|input|}$-Loss Inherent? (folklore)



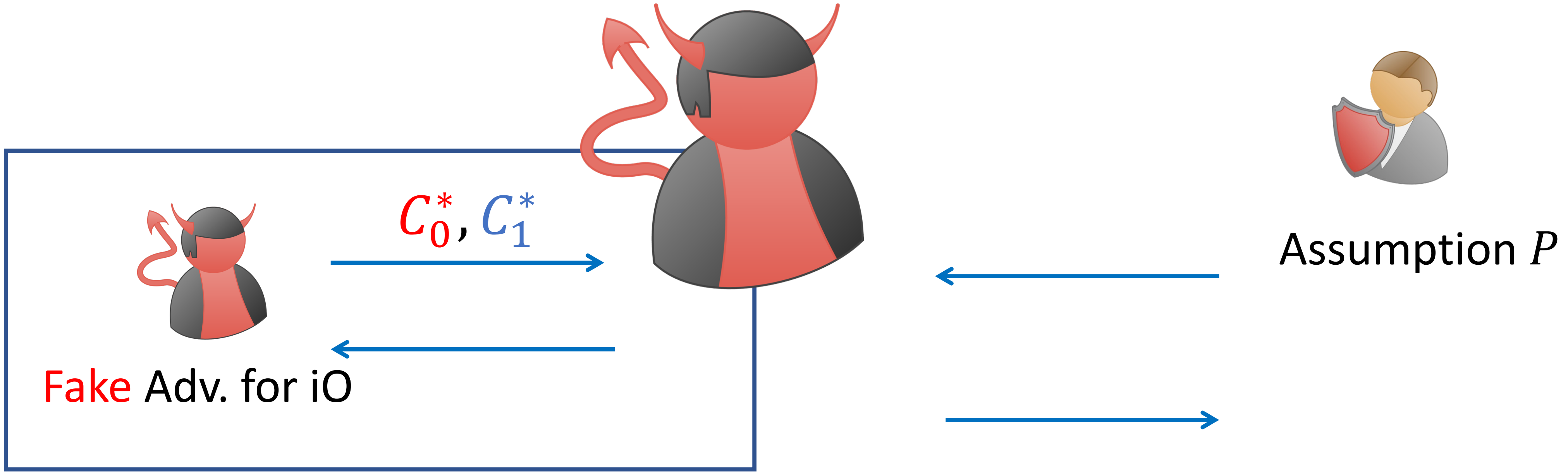
Is $2^{|input|}$-Loss Inherent? (folklore)



If C_0^*, C_1^* differ at some x^*, then reduction **shouldn't** break P.

Otherwise, P is broken unconditionally.

Is $2^{|input|}$-Loss Inherent? (folklore)



If C_0^*, C_1^* differ at some x^*, then reduction **shouldn't** break P.
Otherwise, P is broken unconditionally.

Reduction can't tell, unless it checks at x^*

Broader Perspective

Broader Perspective

Such non-falsifiability appears in many other places in crypto

Broader Perspective

Such non-falsifiability appears in many other places in crypto

Example: Non-Interactive Proofs (for $L \in NP$)

Broader Perspective

Such non-falsifiability appears in many other places in crypto

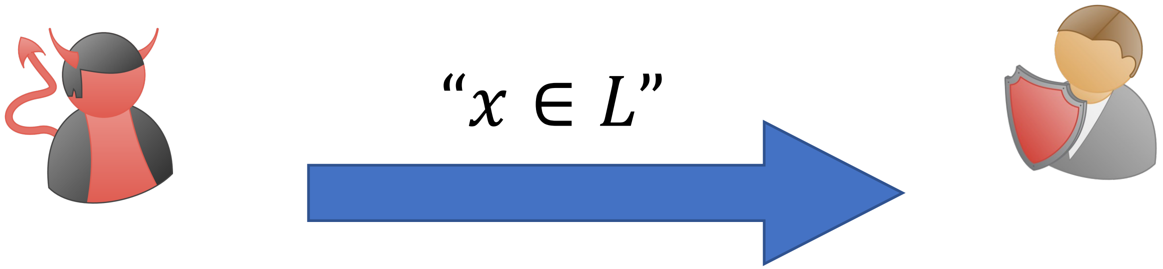
<u>Example:</u> Non-Interactive Proofs (for $L \in NP$)

"$x \in L$"

Broader Perspective

Such non-falsifiability appears in many other places in crypto

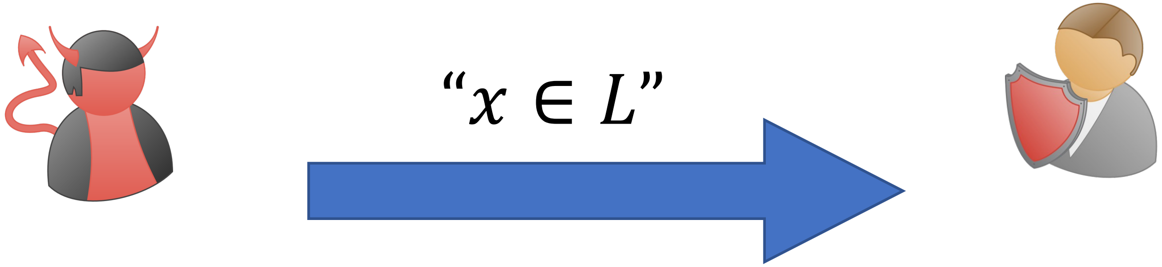
<u>Example:</u> Non-Interactive Proofs (for $L \in NP$)



Broader Perspective

Such non-falsifiability appears in many other places in crypto

<u>**Example:**</u> **Non-Interactive Proofs** (for $L \in NP$)

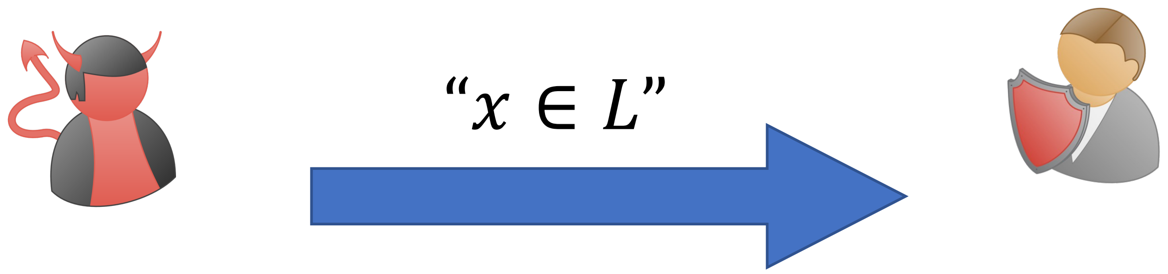


Soundness: If $x \notin L$, any cheating proof should be rejected

Broader Perspective

Such non-falsifiability appears in many other places in crypto

<u>Example:</u> Non-Interactive Proofs (for $L \in NP$)

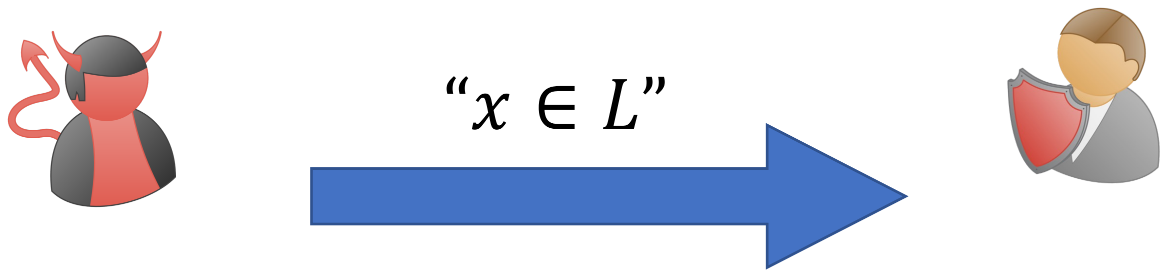


Soundness: $\boxed{\text{If } x \notin L,}$ any cheating proof should be rejected

Broader Perspective

Such non-falsifiability appears in many other places in crypto

<u>Example:</u> Non-Interactive Proofs (for $L \in NP$)



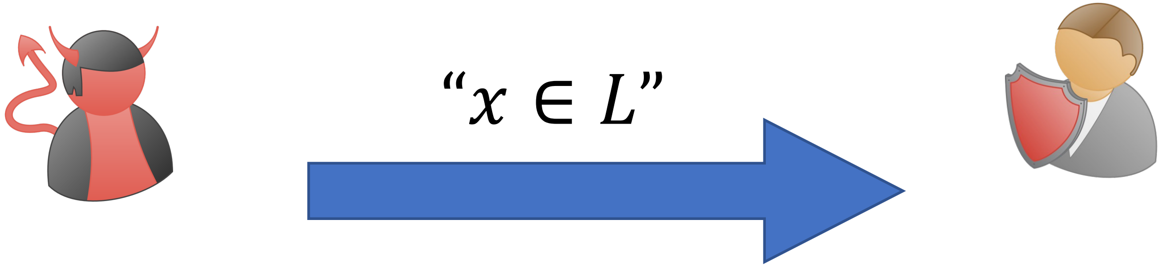
Non-Falsifiable!

Soundness: $\boxed{\text{If } x \notin L,}$ any cheating proof should be rejected

Broader Perspective

Such non-falsifiability appears in many other places in crypto

<u>Example:</u> Non-Interactive Proofs (for $L \in NP$)



Non-Falsifiable!

Soundness: If $x \notin L$, any cheating proof should be rejected

[Gentry-Wichs’10] impossibility for SNARGs

iO

iO

Reduction checks $C_0(x^*) = C_1(x^*)$ for every x^* with $2^{|x^*|}$-loss

Previous Works:

iO

Reduction checks $C_0(x^*) = C_1(x^*)$ for every x^* with $2^{|x^*|}$-loss

Previous Works:

"$\forall x\ C_0(x) = C_1(x)$" can be decided in **P**
[Garg-Pandey-Srinivasan'16, Garg-Srinivasan'16, Garg-Pandey-Srinivasan-Zhandry'17]
[Liu-Zhandry'17]

iO

Reduction checks $C_0(x^*) = C_1(x^*)$ for every x^* with $2^{|x^*|}$-loss

Previous Works:

"$\forall x\ C_0(x) = C_1(x)$" can be decided in **P**

[Garg-Pandey-Srinivasan'16, Garg-Srinivasan'16, Garg-Pandey-Srinivasan-Zhandry'17]

[Liu-Zhandry'17]

This Work:

Leverage **math. proofs** of "$\forall x\ C_0(x) = C_1(x)$" to avoid the $2^{|x|}$-time check

Why Such Math. Proofs Exist?

When iO is used in the security proof of other applications:

…

- Construct C_0, C_1
- **Write a math. proof** for $\forall\, x\; C_0(x) = C_1(x)$
- Apply iO security to derive $iO(C_0) \approx_c iO(C_1)$

…

The proof must be "**short**" (length $\ll 2^{|x|}$)

Otherwise, we (human brain) can't understand it.

Our Results I (for Propositional Logic)

$\mathcal{O}$ with security loss independent of $|input|$ for any ckts $\{C_\lambda^1\}_\lambda, \{C_\lambda^2\}_\lambda$ where $C_\lambda^1(x) \leftrightarrow C_\lambda^2(x)$ have **poly-size proofs** in *Extended Frege systems.*

Extended Frege System ($\mathcal{EF}$)

- **Variables**: $p, q, r, \ldots$
- **Formulas**: $p \to r, p \wedge q, \neg p, \ldots$
- **Axioms**:

$$p \to (q \to p)$$
$$(p \to (q \to r)) \to ((p \to q) \to (p \to r))$$
$$p \to \neg\neg p$$

- **Inference Rule:**

$$p, p \to q \vdash q$$

- **Extension Rule:**

$$e \leftrightarrow \phi$$

(assign a new variable e to an existing formula ϕ)

Our Results II (for Cook's Theory PV)

iO for any ***unbounded-input*** Turing machines M_1, M_2,
with $\vdash_{PV} M_1(x) = M_2(x)$.

Assumptions: sub-exponential security of LWE & iO for circuits.

Cook's Theory *PV* [Cook'75] *"Poly-time Reasoning"*

Terms: $f(x), g(x), h(x_1, x_2), \ldots$

Lines are Equations: $f(x) = h(x),\ f\big(x_1, g(x_2)\big) = h(x_1, x_2), \ldots$

Allow definition of any polynomial-time functions, e.g.

- Arithmetic: $+, -, \times, \div, \leq, <, \lfloor\cdot\rfloor, mod, \ldots$
- Logic Symbols: $\rightarrow, \neg, \wedge, \ldots$

Relation Between *PV* and $\mathcal{EF}$

Propositional Translation
[Cook'75]

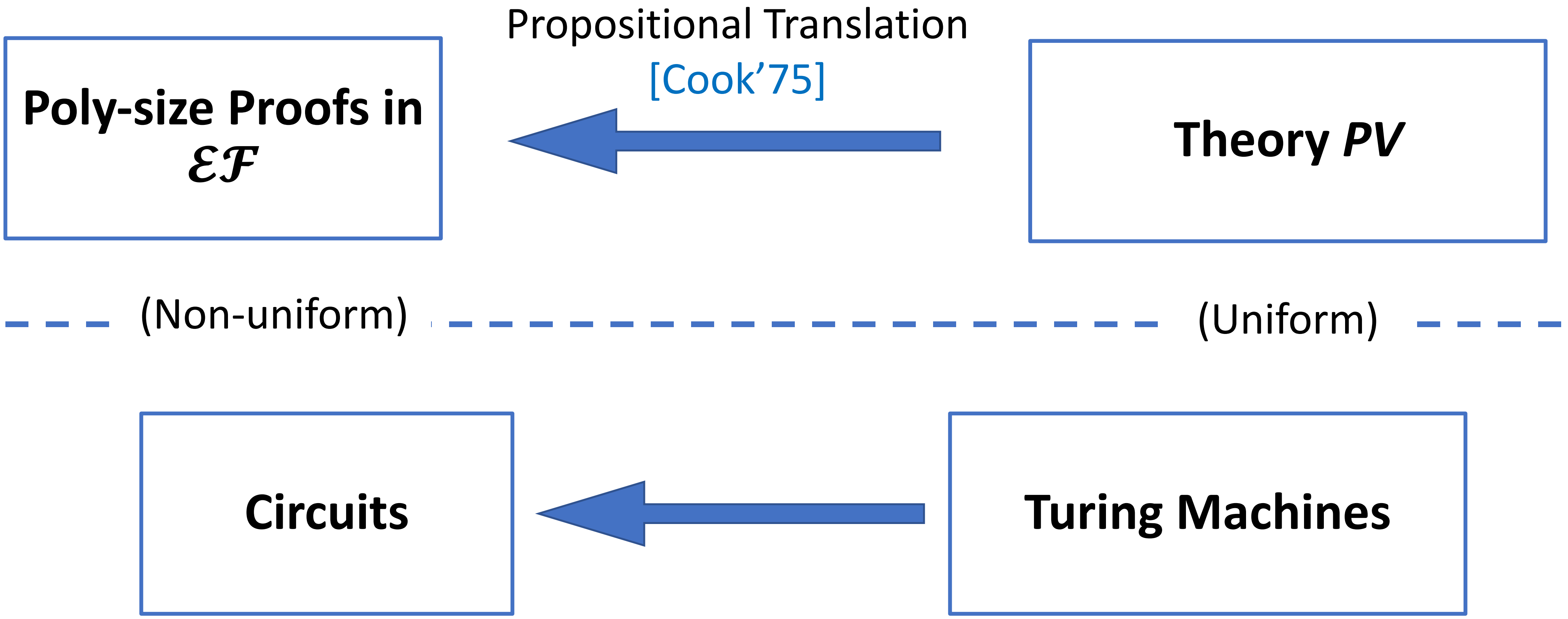
Poly-size Proofs in $\boldsymbol{\mathcal{EF}}$

(Non-uniform)

Theory *PV*

(Uniform)

Relation Between *PV* and $\mathcal{EF}$



What Can *PV* Prove?

What Can *PV* Prove?

- **Correctness** of "natural" poly-time algorithms

What Can *PV* Prove?

- **Correctness** of “natural” poly-time algorithms
- **Linear Algebra:**

  Matrix properties,

  Determinants,

  Cayley-Hamilton Theorem,

  …

What Can *PV* Prove?

- **Correctness** of "natural" poly-time algorithms
- **Linear Algebra:**

  Matrix properties,

  Determinants,

  Cayley-Hamilton Theorem,

  ...
- **Complexity Theorems:**

  Cook-Levin theorem, PCP theorem,

  …

What Can *PV* Prove?

- **Correctness** of “natural” poly-time algorithms
- **Linear Algebra:**

  Matrix properties,

  Determinants,

  Cayley-Hamilton Theorem,

  ...
- **Complexity Theorems:**

  Cook-Levin theorem, PCP theorem,

  …

This work:

- Many crypto algorithms are “natural”, e.g.

  ElGamal Encryption,

  Regev’s Encryption

  Commitments,

  Puncturable PRFs

Limitation of PV (Assuming Factoring is hard)

Limitation of *PV* (Assuming Factoring is hard)

- Fermat's Little Theorem

Limitation of PV (Assuming Factoring is hard)

- Fermat's Little Theorem

- Correctness of the algorithm that decides Primes

Limitation of *PV* (Assuming Factoring is hard)

- Fermat's Little Theorem

- Correctness of the algorithm that decides Primes

(Both from Witnessing Theorem)

How to leverage math. proofs?

(An overview)

Key Observation: Math. Proofs are "Local"

Key Observation: Math. Proofs are "**Local**"

Truth of each line follows from $O(1)$ previous lines

Key Observation: Math. Proofs are "**Local**"

Truth of each line follows from $O(1)$ previous lines

e.g., in $\mathcal{EF}$, Each line either follows from an **axiom** or **modus ponens**
$(p, p \to q \vdash q)$

Key Observation: Math. Proofs are "**Local**"

Truth of each line follows from $O(1)$ previous lines

e.g., in $\mathcal{EF}$, Each line either follows from an **axiom** or **modus ponens** $(p, p \to q \vdash q)$

<u>Example:</u> Proof of $A \to A$ in *EF*

1. $A \to ((B \to A) \to A)$ (instance of (A1))
2. $(A \to ((B \to A) \to A)) \to ((A \to (B \to A)) \to (A \to A))$ (instance of (A2))
3. $(A \to (B \to A)) \to (A \to A)$ (from (1) and (2) by modus ponens)
4. $A \to (B \to A)$ (instance of (A1))
5. $A \to A$ (from (4) and (3) by modus ponens)

Key Observation: Math. Proofs are "**Local**"

Truth of each line follows from $O(1)$ previous lines

e.g., in $\mathcal{EF}$, Each line either follows from an **axiom** or **modus ponens** $(p, p \to q \vdash q)$

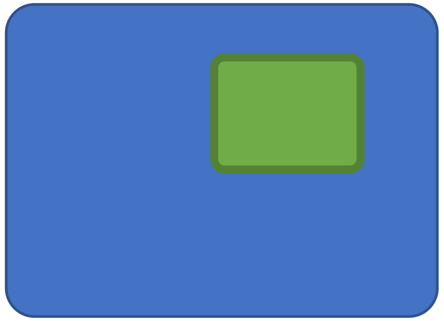
Example: Proof of $A \to A$ in *EF*

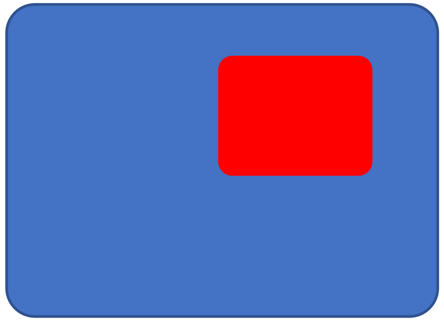
1. $A \to ((B \to A) \to A)$ (instance of (A1))
2. $(A \to ((B \to A) \to A)) \to ((A \to (B \to A)) \to (A \to A))$ (instance of (A2))
3. $(A \to (B \to A)) \to (A \to A)$ (from (1) and (2) by modus ponens)
4. $A \to (B \to A)$ (instance of (A1))
5. $A \to A$ (from (4) and (3) by modus ponens)

How do we leverage localness?

“Local” Equivalence for **Circuits**

“Local” Equivalence for **Circuits**

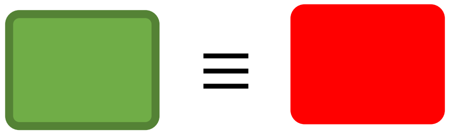
C: 

 : C'

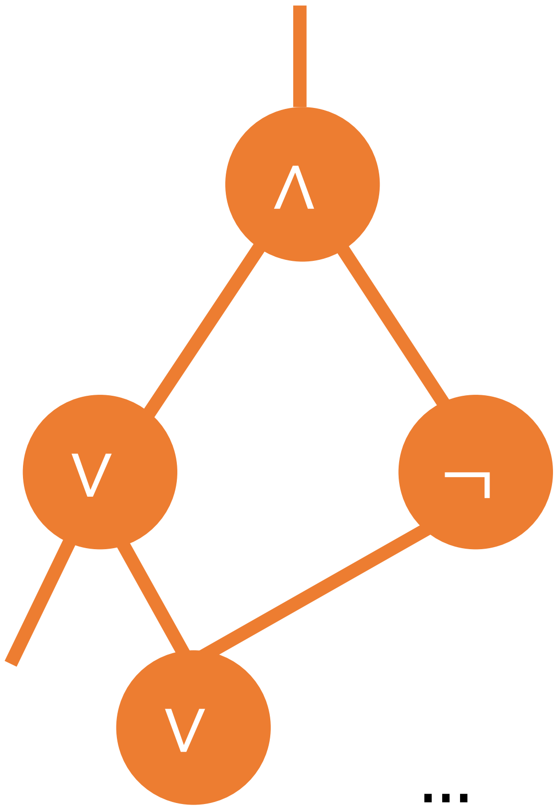
“Local” Equivalence for **Circuits**

C and C' are δ-equivalent, if C and $C'$are almost the same,

except for a **functionality equivalent *<u>sub-circuit</u>*** of size $O(\log n)$

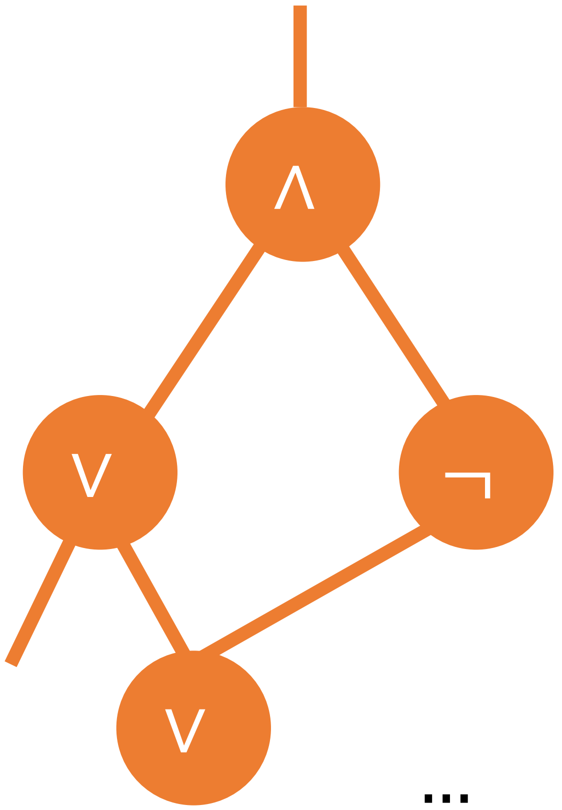


Sub-Circuits: Defined as Subsets of Gates



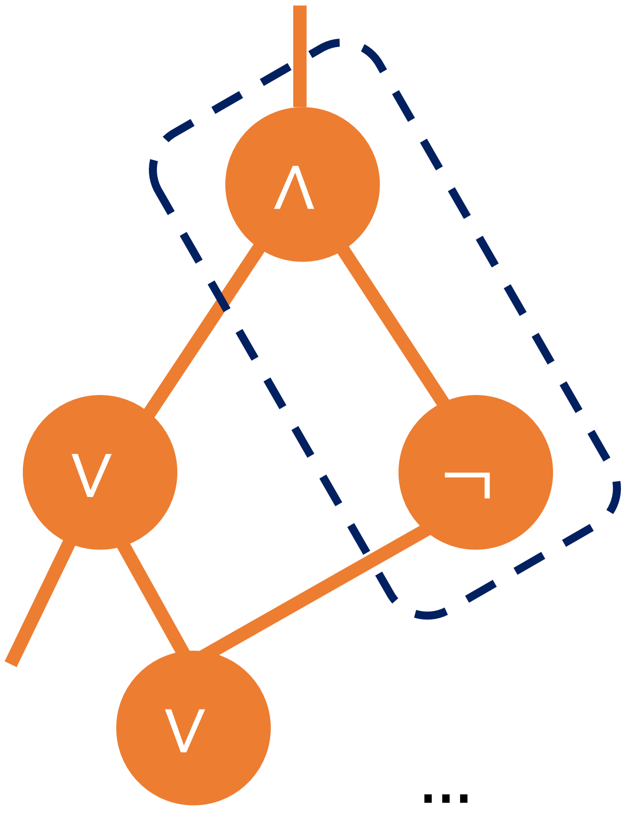
…

Sub-Circuits: Defined as Subsets of Gates



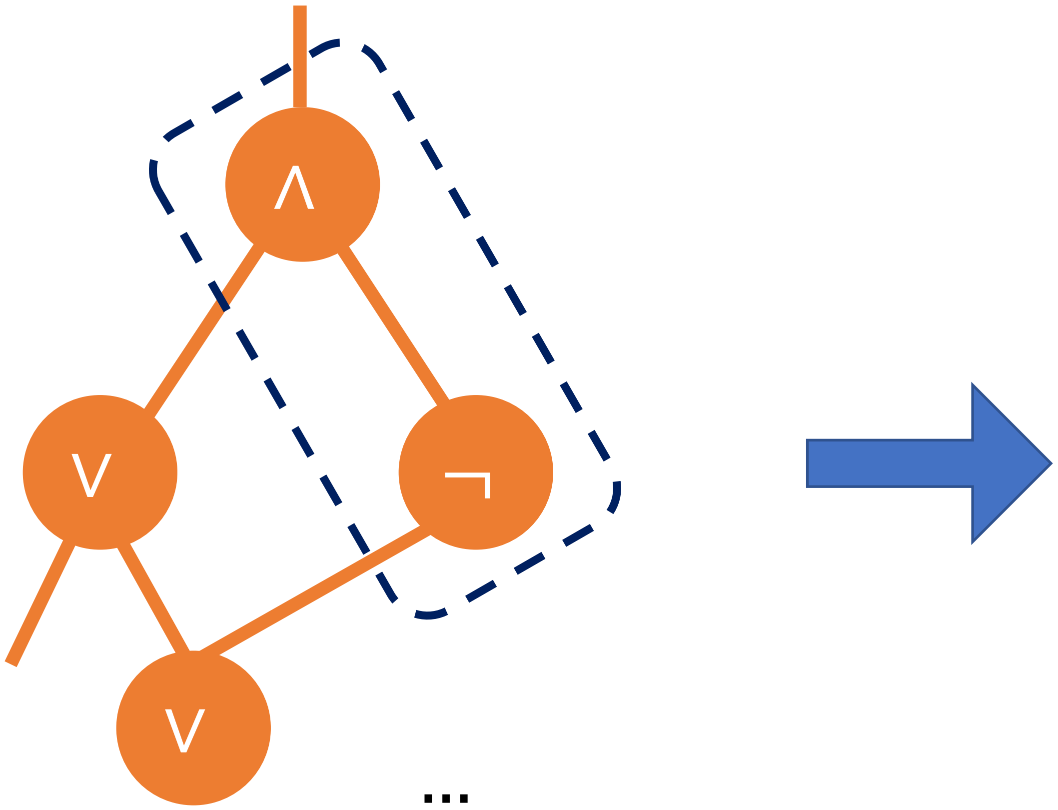
Directed Acyclic Graph

Sub-Circuits: Defined as Subsets of Gates



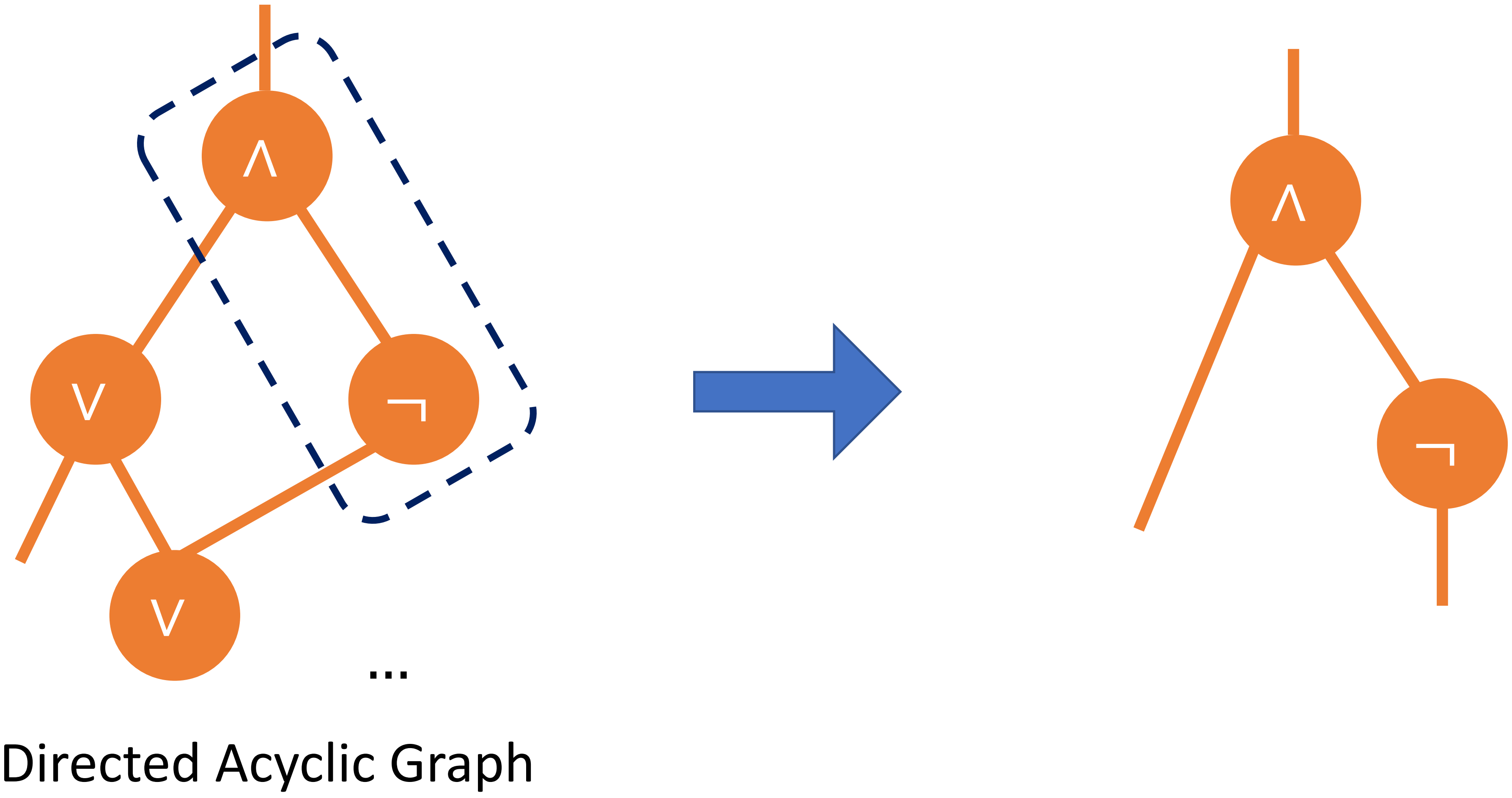
Directed Acyclic Graph

Sub-Circuits: Defined as Subsets of Gates

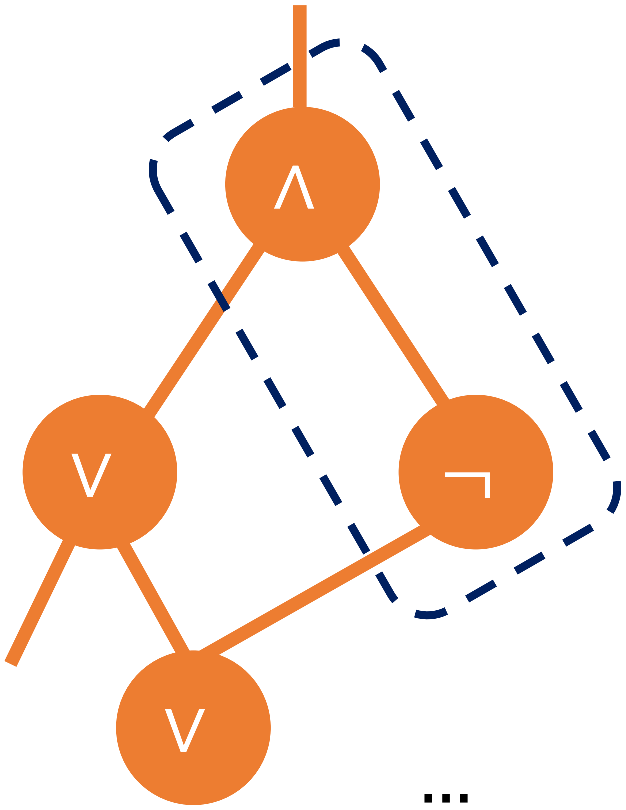


Directed Acyclic Graph

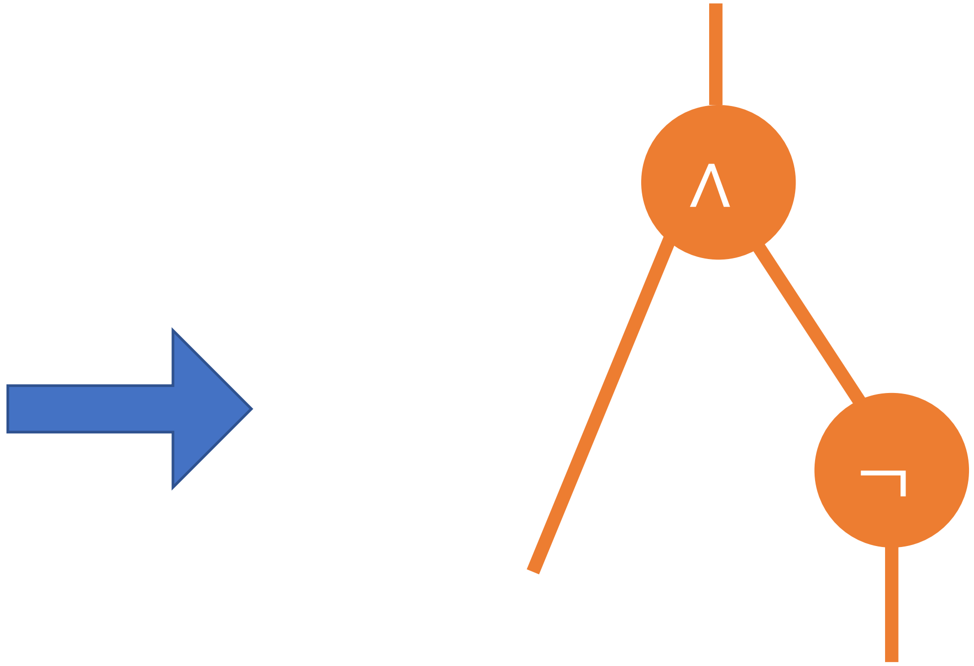
Sub-Circuits: Defined as Subsets of Gates



Sub-Circuits: Defined as Subsets of Gates

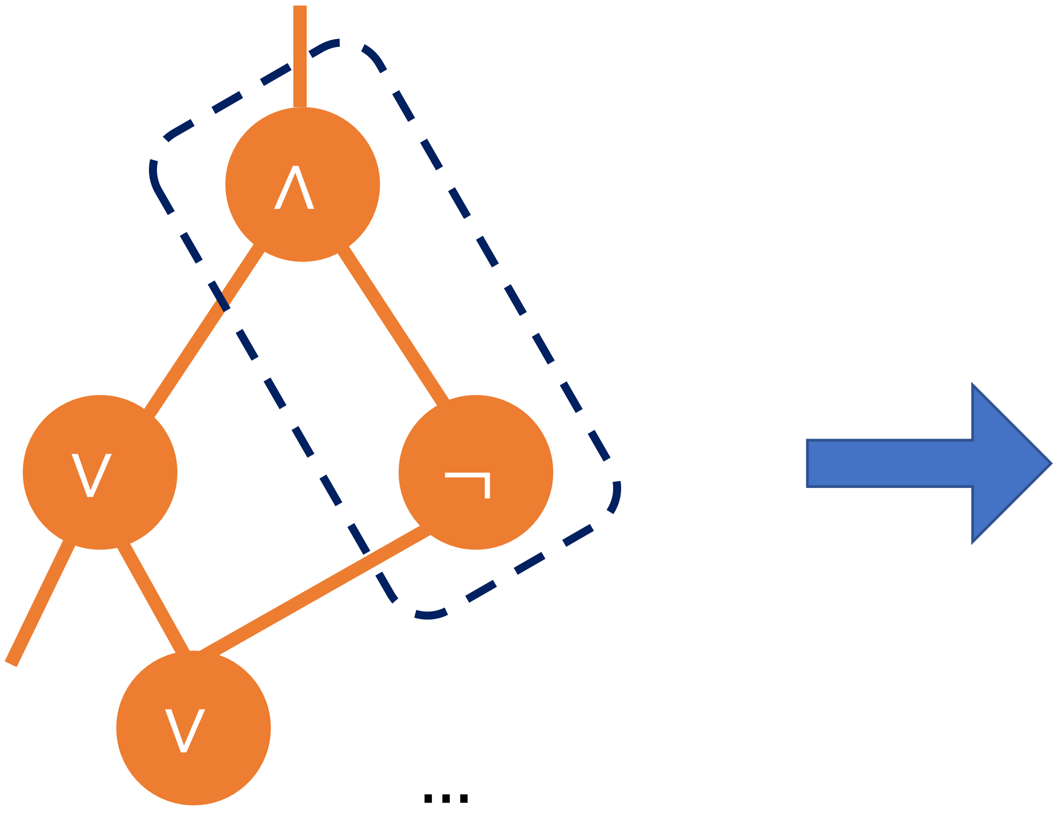


Directed Acyclic Graph

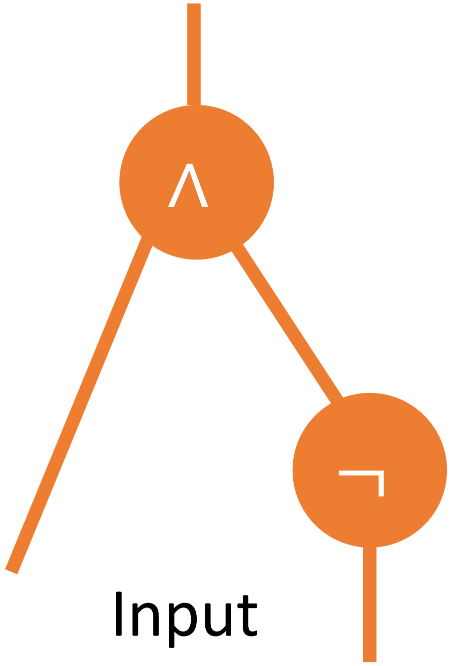


<u>**Sub-Circuit**</u>

Sub-Circuits: Defined as Subsets of Gates

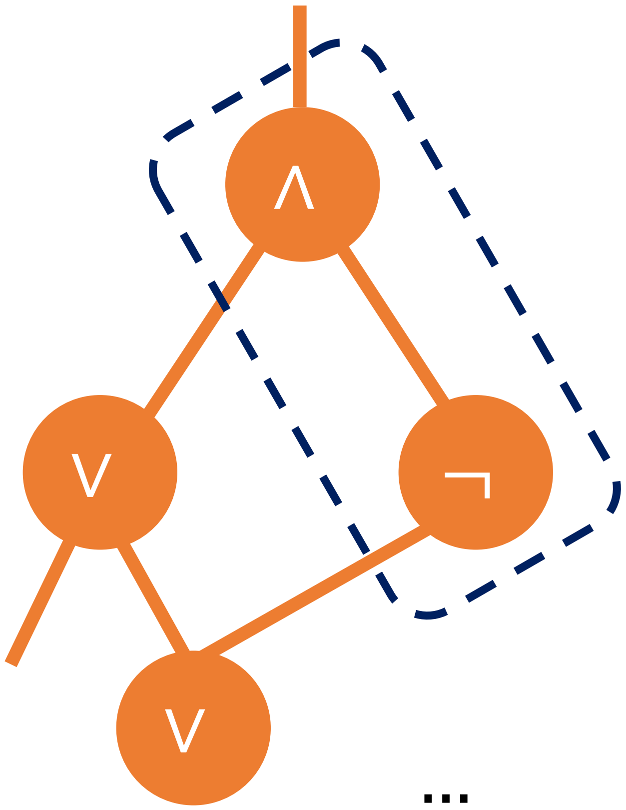


Directed Acyclic Graph

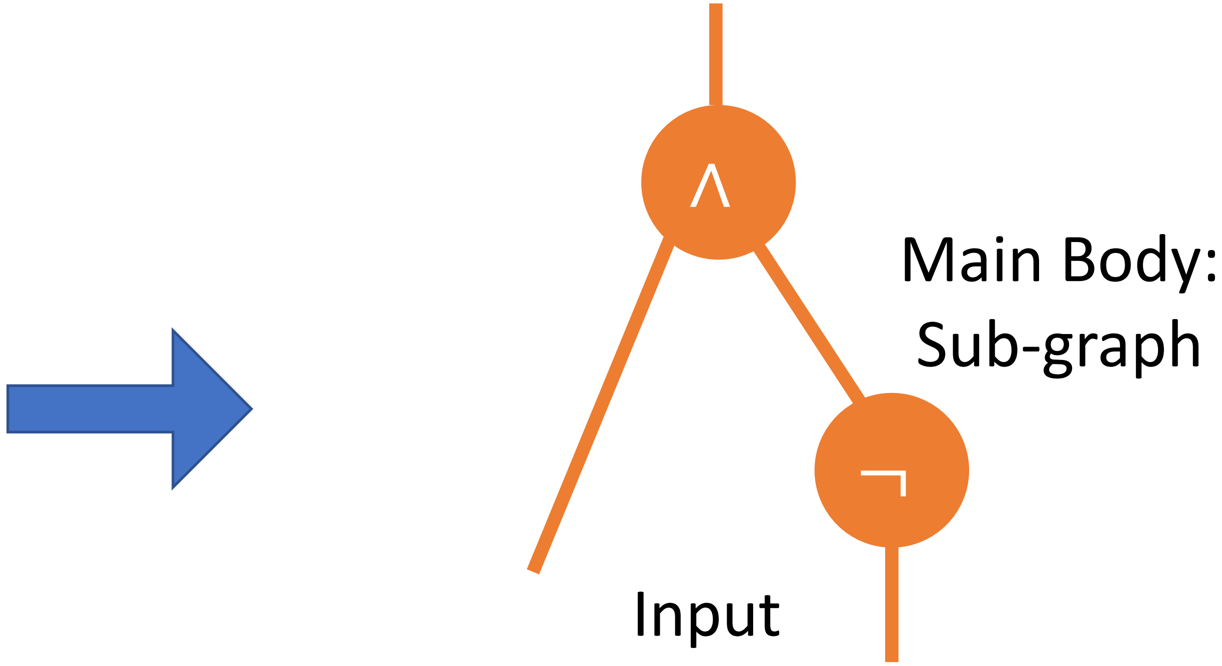


Sub-Circuit

Sub-Circuits: Defined as Subsets of Gates

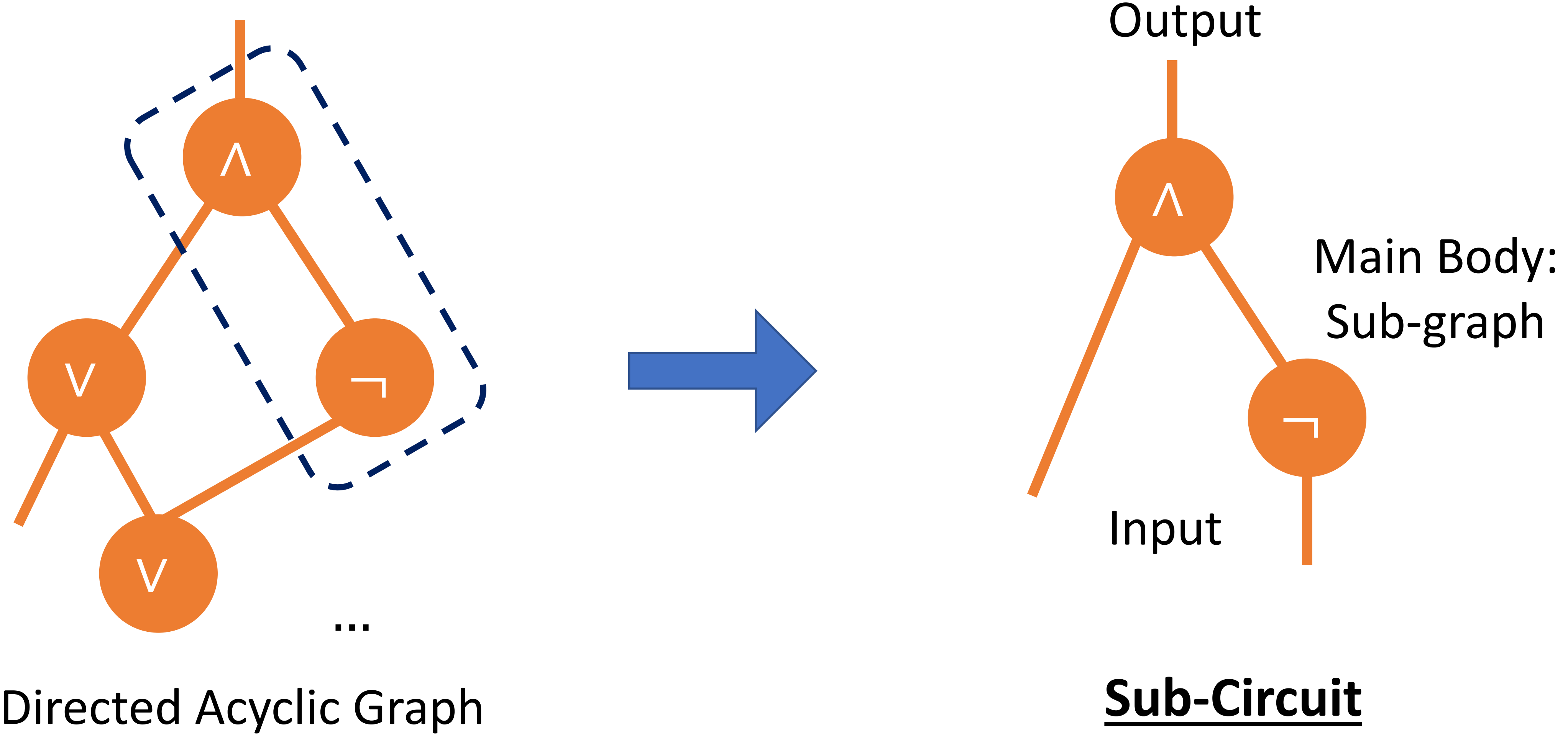


Directed Acyclic Graph

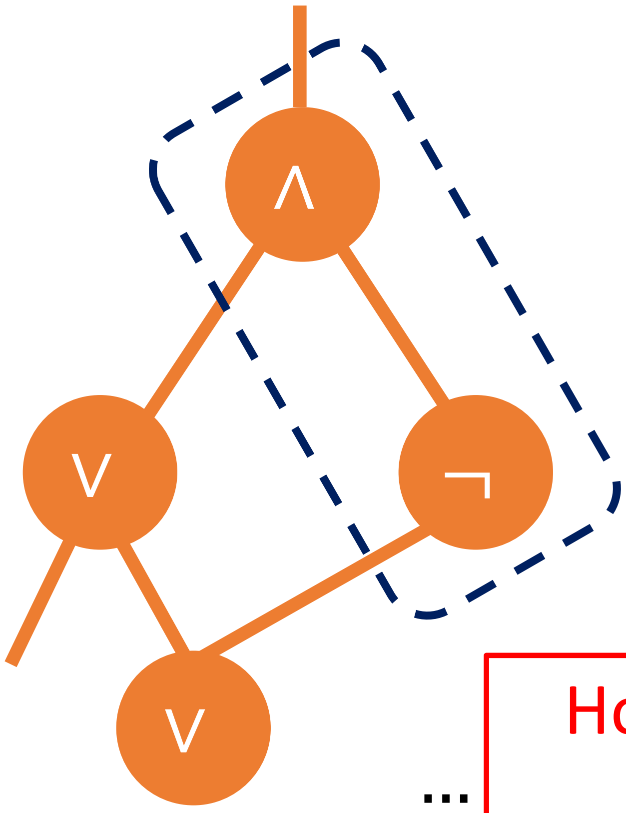


<u>Sub-Circuit</u>

Sub-Circuits: Defined as Subsets of Gates

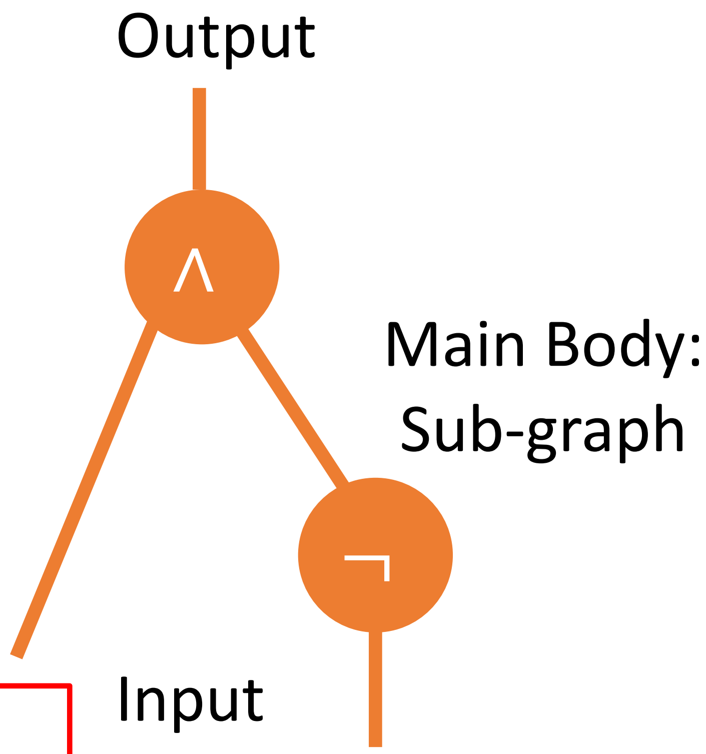


Sub-Circuits: Defined as Subsets of Gates



Directed Acyclic Graph

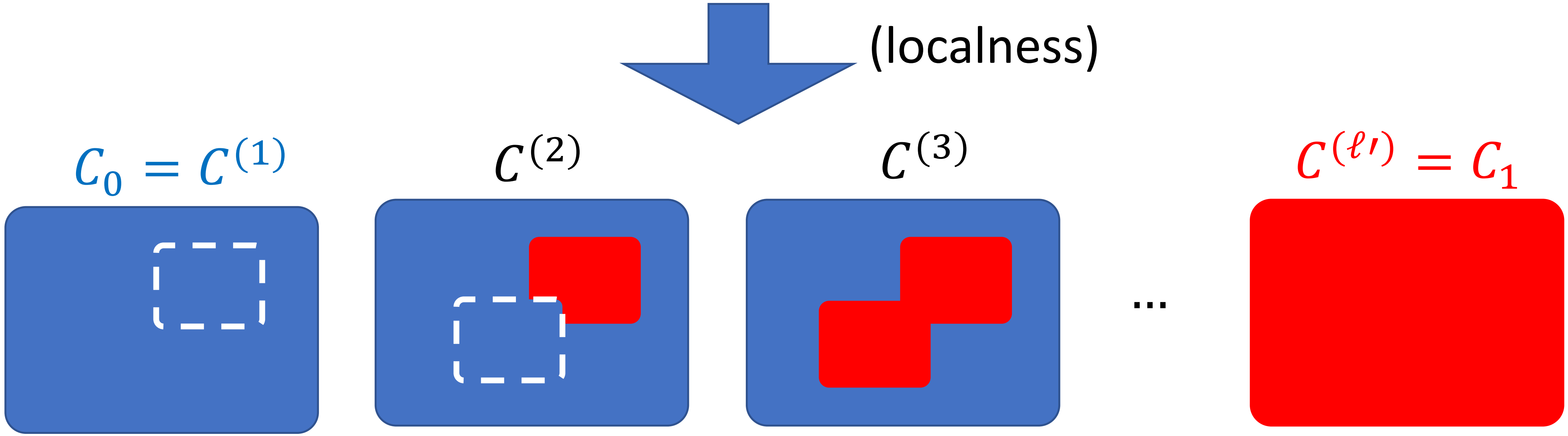
How does δ-Equivalence relate to proofs?



<u>Sub-Circuit</u>

$\mathcal{EF}$-Proofs imply δ-Equivalence

$\theta_1,\ \theta_2,\ \ldots,\ \theta_\ell$: $\mathcal{EF}$-proof of $C_0(x) \leftrightarrow C_1(x)$



$C^{(i)}$ and $C^{(i+1)}$ are δ-equivalent

Focus on δ-Equivalent Ckts

Focus on δ-Equivalent Ckts

Assume iO for δ-Equivalent Ckts exists: δiO

Focus on δ-Equivalent Ckts

Assume iO for δ-Equivalent Ckts exists: δiO

$\delta iO(C^{(1)})$ $\delta iO(C^{(2)})$ $\delta iO(C^{(3)})$ … $\delta iO(C^{(\ell')})$

Focus on δ-Equivalent Ckts

Assume iO for δ-Equivalent Ckts exists: δiO

$$\delta iO(C^{(1)}) \quad \approx \quad \delta iO(C^{(2)}) \quad \approx \quad \delta iO(C^{(3)}) \quad \dots \quad \delta iO(C^{(\ell')})$$

Focus on δ-Equivalent Ckts

Assume iO for δ-Equivalent Ckts exists: δiO

$$\delta iO(C^{(1)}) \quad \approx \quad \delta iO(C^{(2)}) \quad \approx \quad \delta iO(C^{(3)}) \quad \ldots \quad \delta iO(C^{(\ell')})$$

$$\Rightarrow \delta iO(C_0) \approx_c \delta iO(C_1)$$

Focus on δ-Equivalent Ckts

Assume iO for δ-Equivalent Ckts exists: δiO

$\delta iO(C^{(1)}) \quad \approx \quad \delta iO(C^{(2)}) \quad \approx \quad \delta iO(C^{(3)}) \quad \ldots \quad \delta iO(C^{(\ell')})$

$$\Rightarrow \delta iO(C_0) \approx_c \delta iO(C_1)$$

<u>Total Security Loss</u> = ℓ' ·Loss of δiO $\quad (\ell' = poly)$

Focus on δ-Equivalent Ckts

Assume iO for δ-Equivalent Ckts exists: δiO

$$\delta iO(C^{(1)}) \quad \approx \quad \delta iO(C^{(2)}) \quad \approx \quad \delta iO(C^{(3)}) \quad \ldots \quad \delta iO(C^{(\ell')})$$

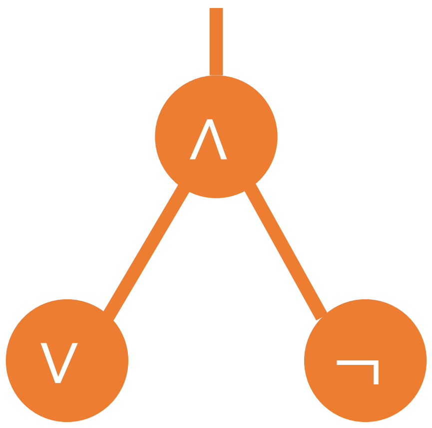
$$\Rightarrow \delta iO(C_0) \approx_c \delta iO(C_1)$$

Total Security Loss = ℓ' $\cdot$Loss of δiO $\qquad (\ell' = poly)$

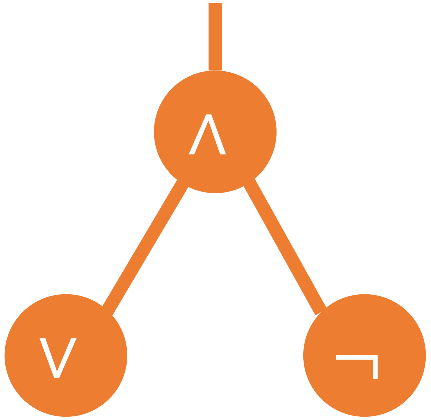
If loss of δiO is independent of $|input|$, so is the total loss.

Constructing δiO

Constructing δiO



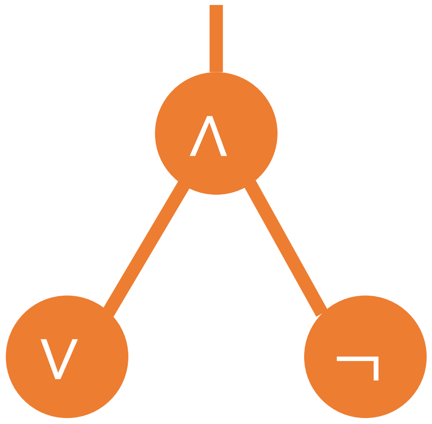
Constructing δiO



"Gate-by-Gate" Obfuscation:

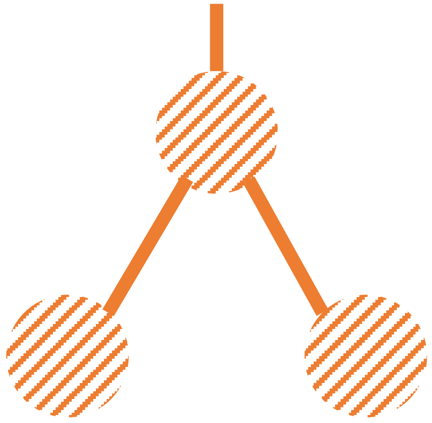
Obfuscate each gate separately

Constructing δiO

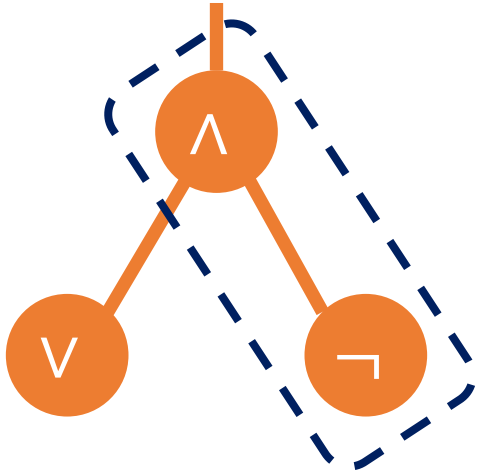


"Gate-by-Gate" Obfuscation:

Obfuscate each gate separately

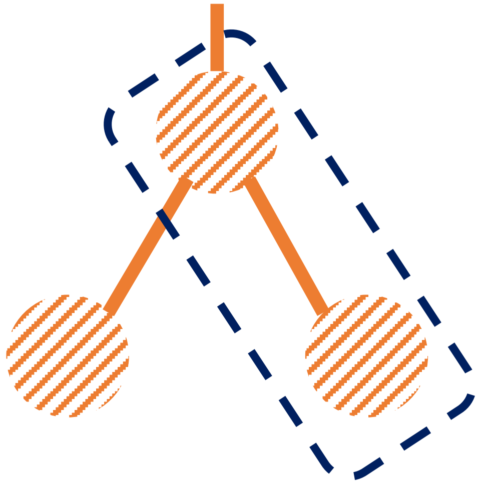


Constructing δiO



"Gate-by-Gate" Obfuscation:

Obfuscate each gate separately



Security loss is independent of $|input|$, why?

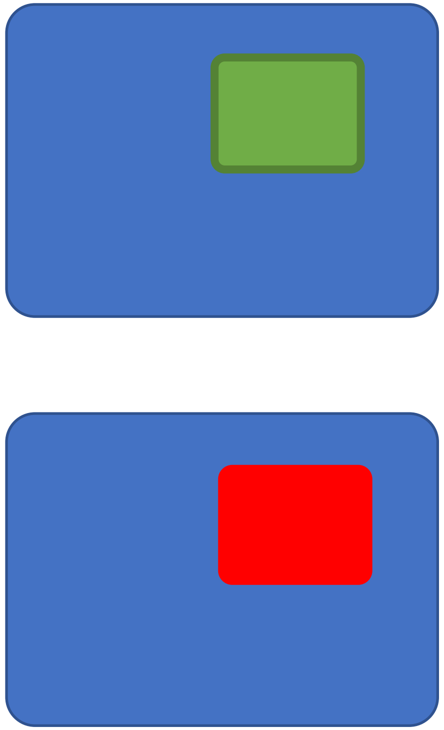
We can "**cut**" the obfuscated program!

Security Proof for δiO

Security Proof for δiO

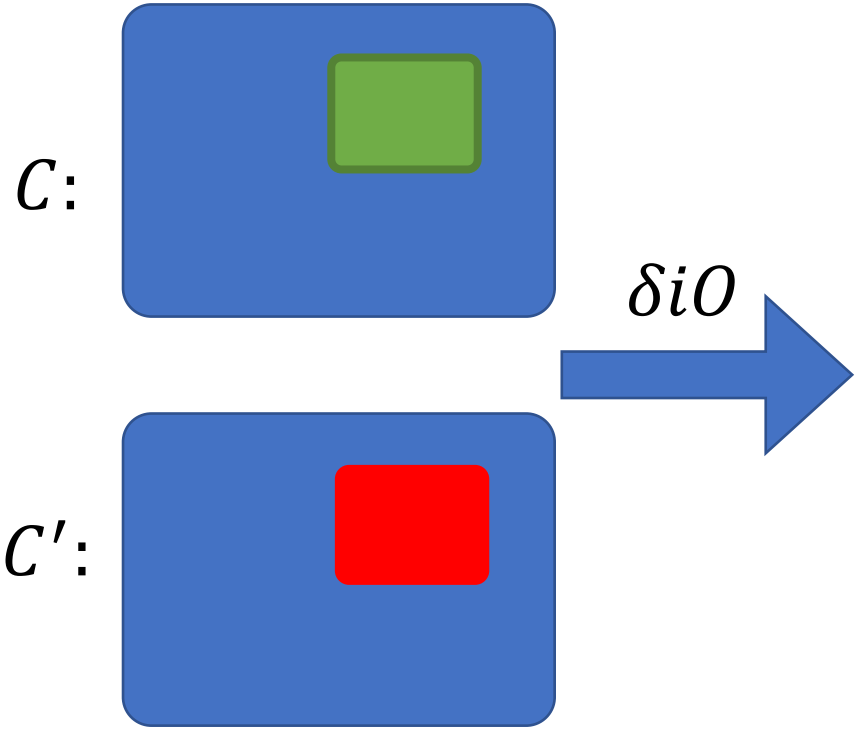
C:

C':



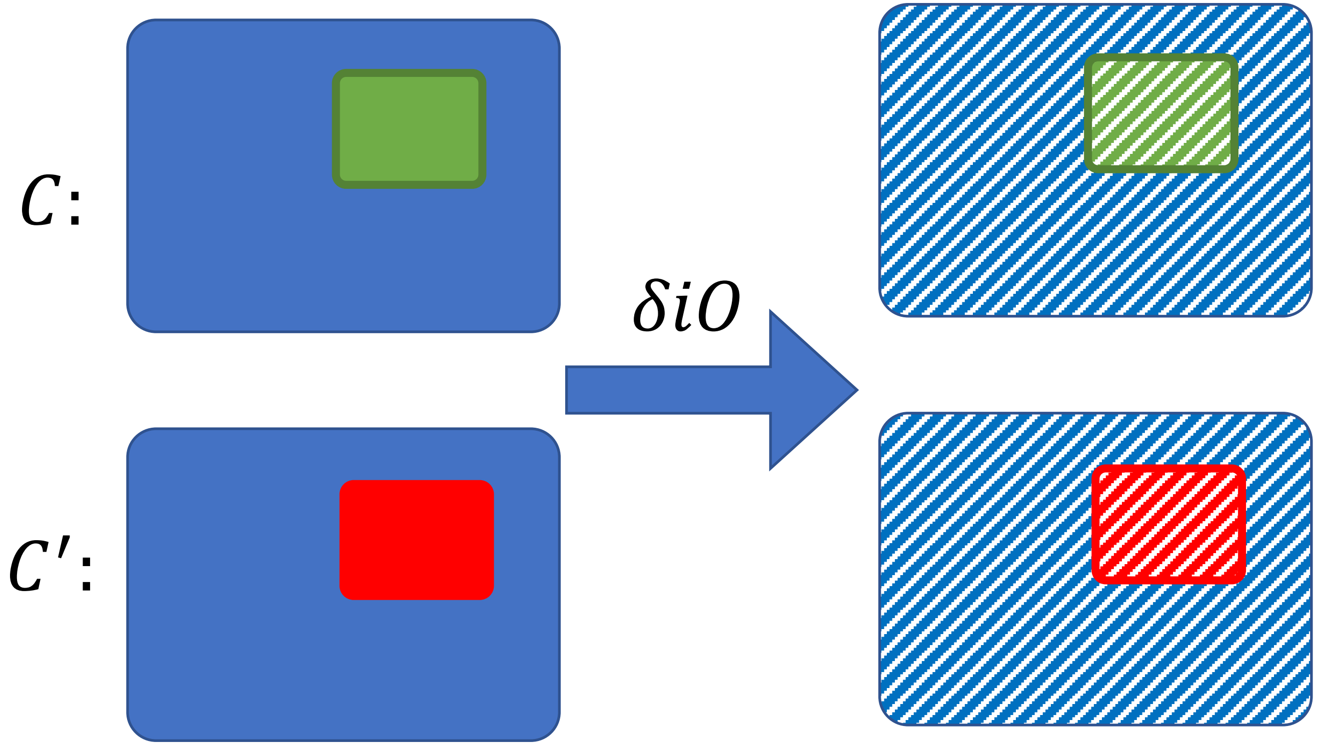
C, C': δ-Equivalent

Security Proof for δiO



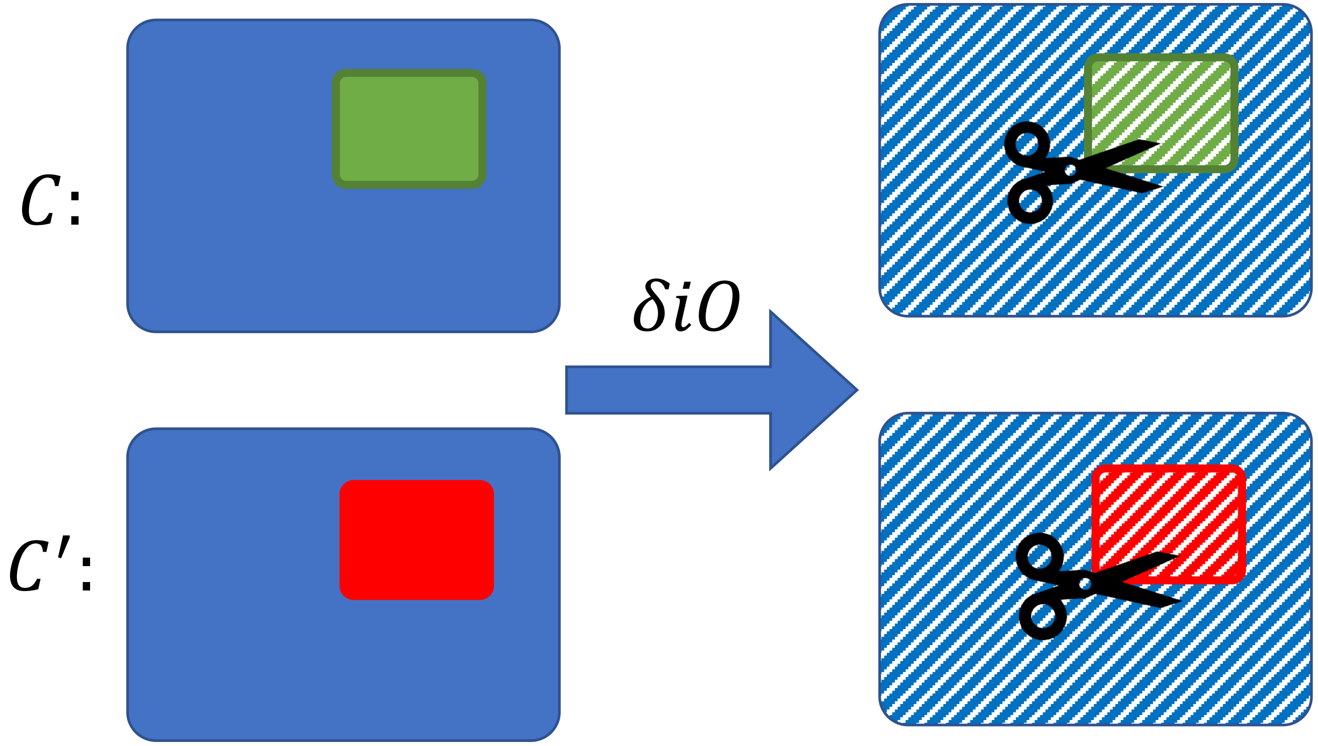
C, C': δ-Equivalent

Security Proof for δiO



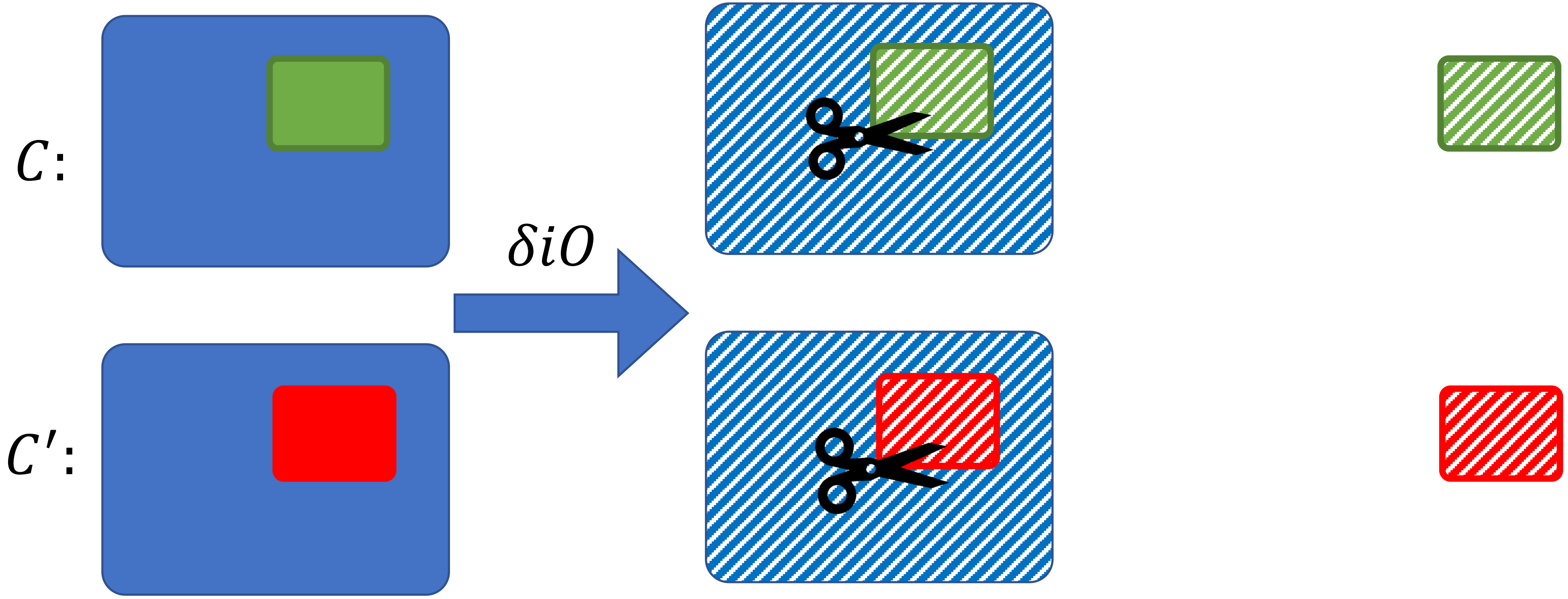
C, C': δ-Equivalent

Security Proof for δiO



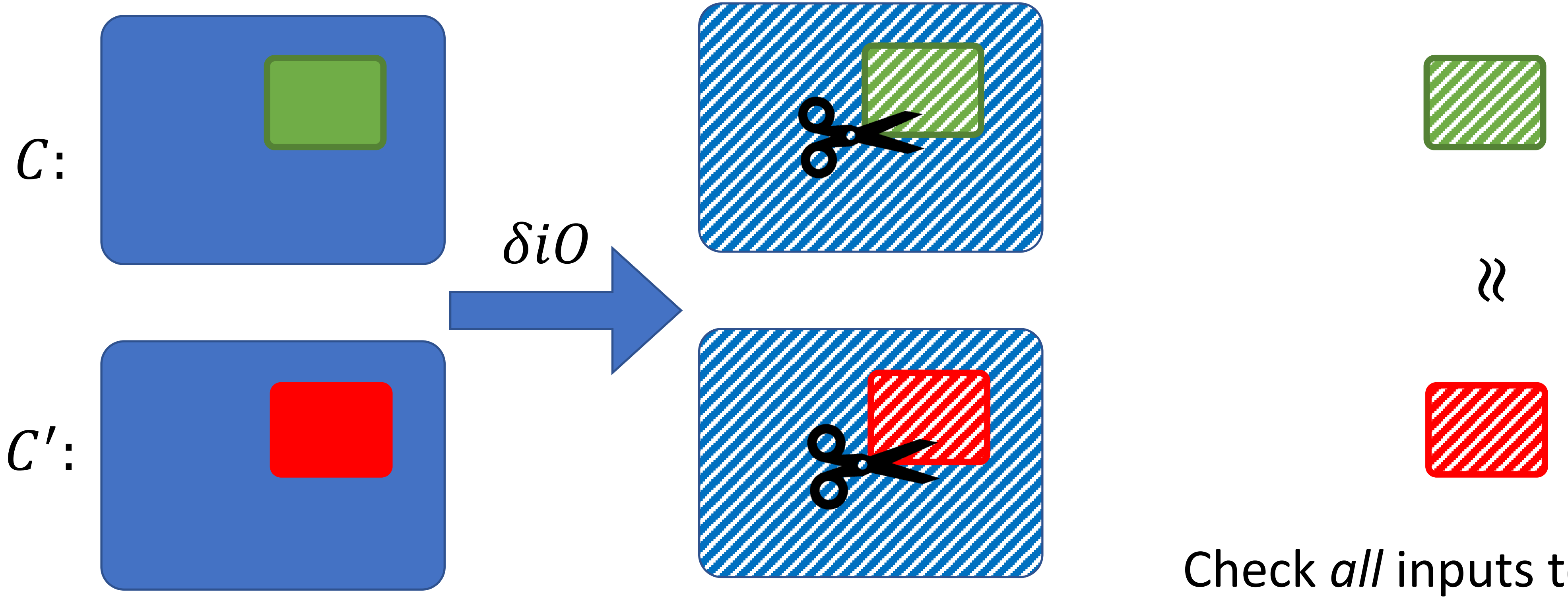
C, C': δ-Equivalent

Security Proof for δiO



C, C': δ-Equivalent

Security Proof for δiO



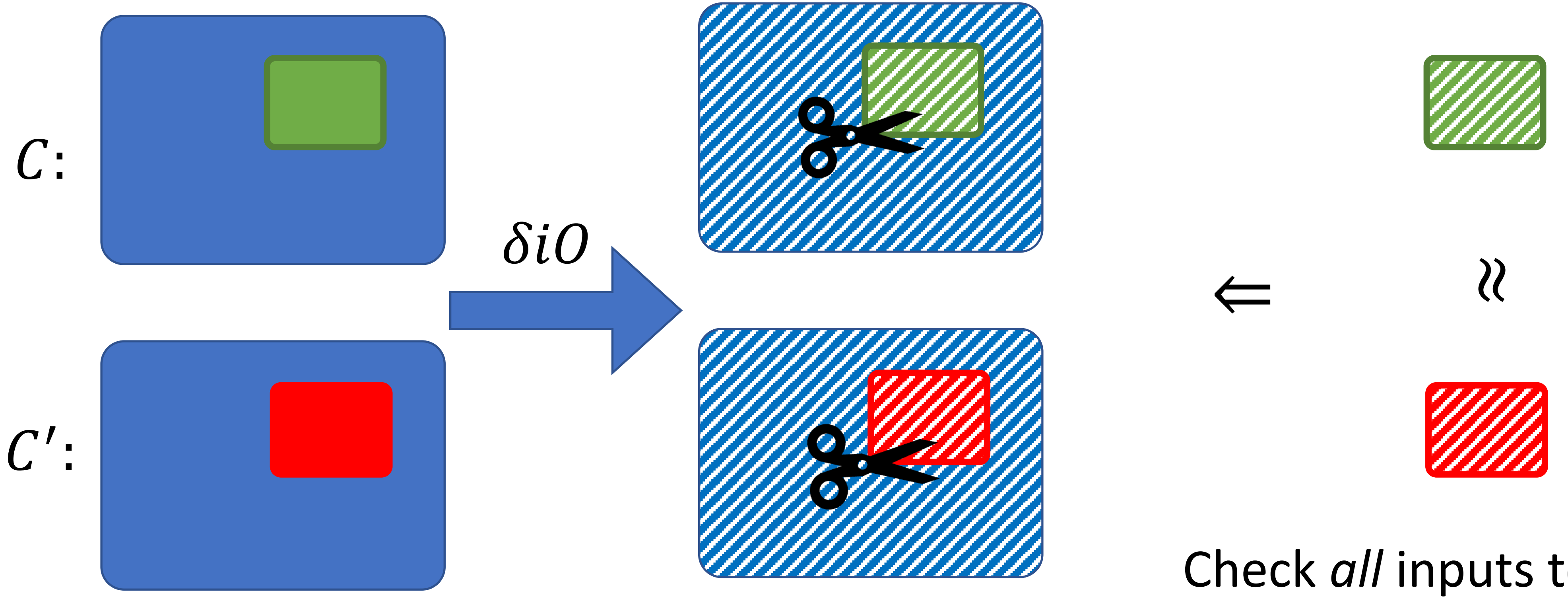
C, C': δ-Equivalent

Check *all* inputs to **Sub-ckt**

Security Loss: $2^{|subckt\ input|}$

$= 2^{O(\log n)} = poly$!

Security Proof for δiO



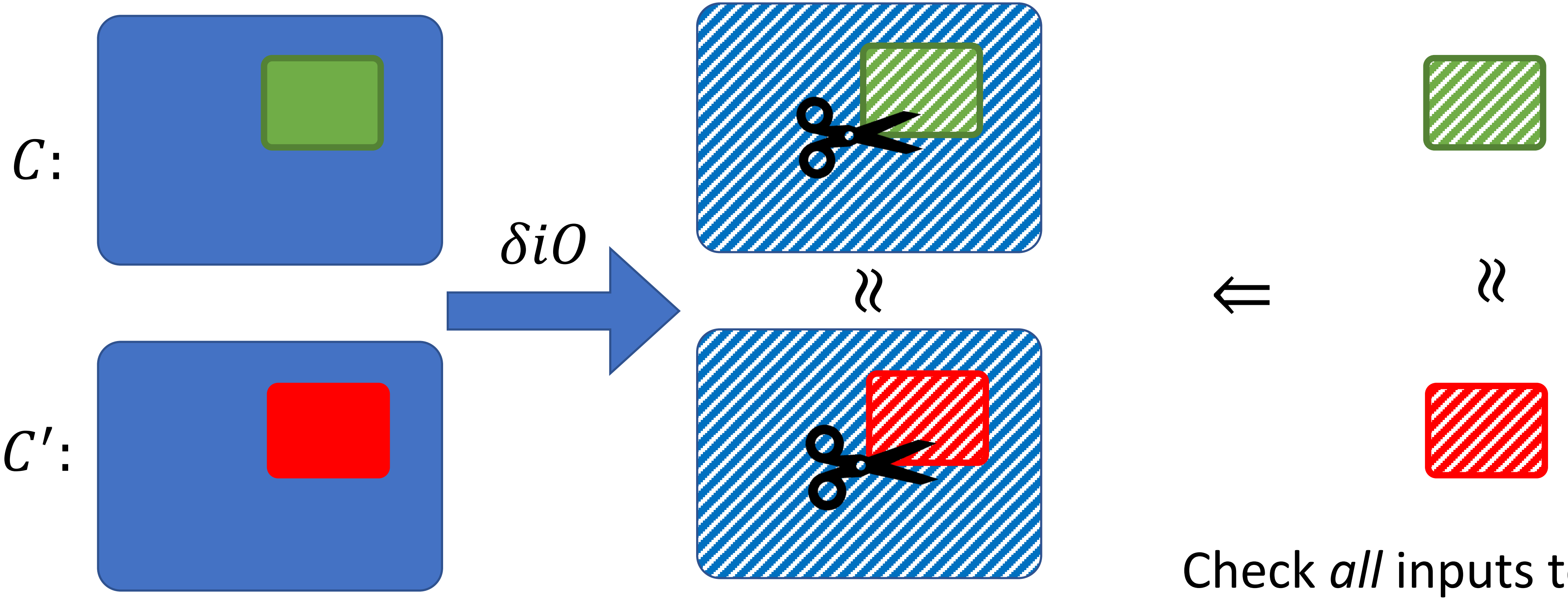
C, C': δ-Equivalent

Check *all* inputs to **Sub-ckt**

Security Loss: $2^{|subckt\ input|}$

$= 2^{O(\log n)} = poly!$

Security Proof for δiO



C, C': δ-Equivalent

Check *all* inputs to **Sub-ckt**

Security Loss: $2^{|subckt\ input|}$

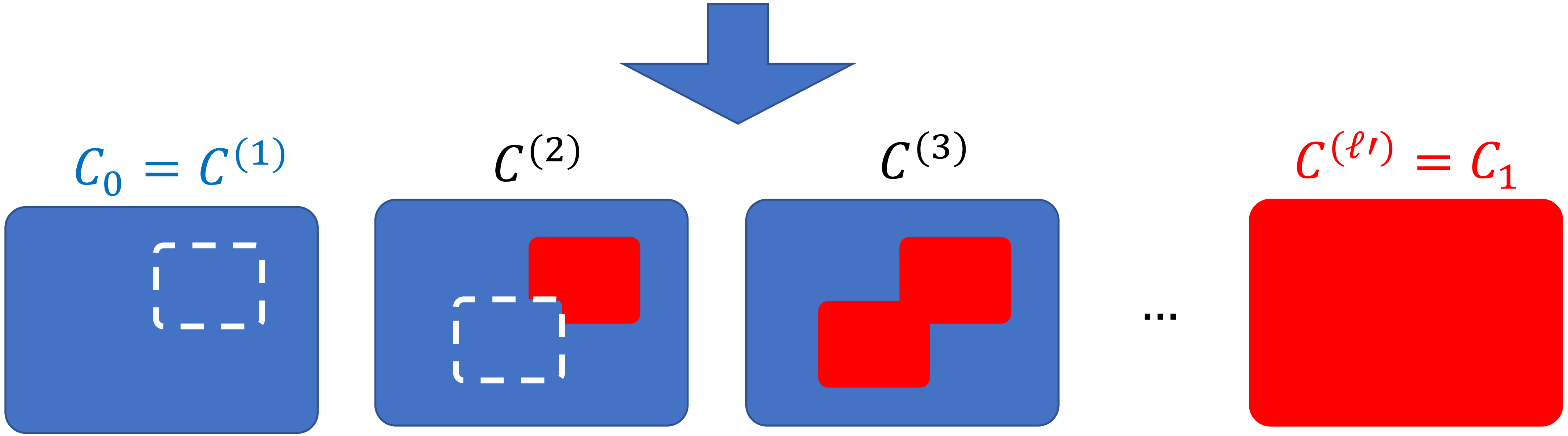
$= 2^{O(\log n)} = poly$!

Technical Details

- **$\boldsymbol{\delta}$-Equivalence from $\mathcal{EF}$-proofs**
- δiO Construction
- iO for Turing Machines

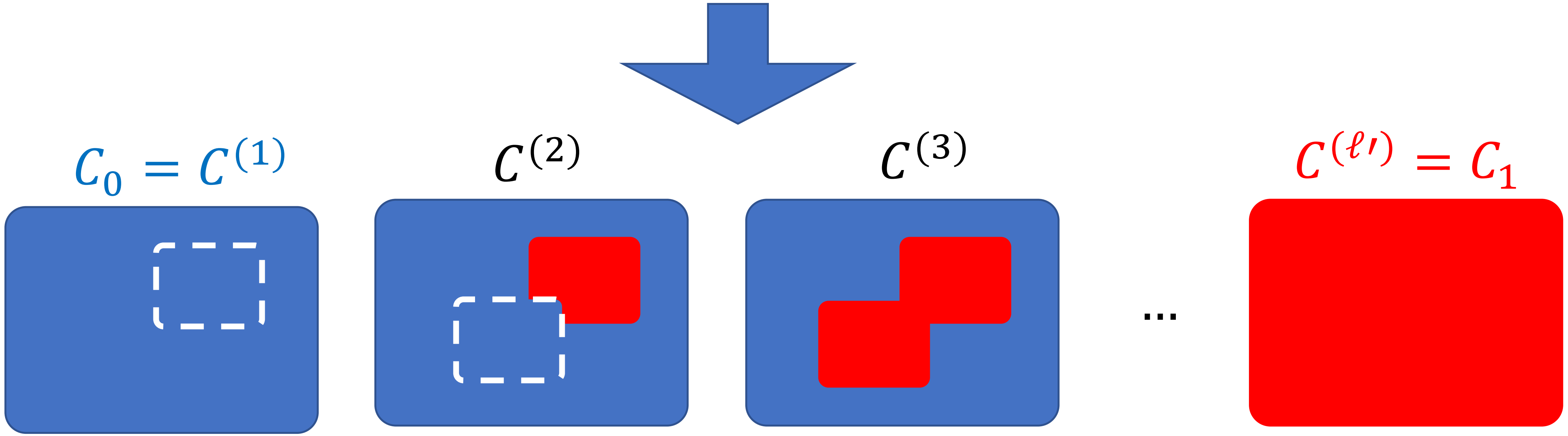
Recall: δ-Equivalence via $\mathcal{EF}$-Proofs

$\theta_1,\ \theta_2,\ \ldots,\ \theta_\ell$: $\mathcal{EF}$-proof of $C_0(x) \leftrightarrow C_1(x)$



Recall: δ-Equivalence via $\mathcal{EF}$-Proofs

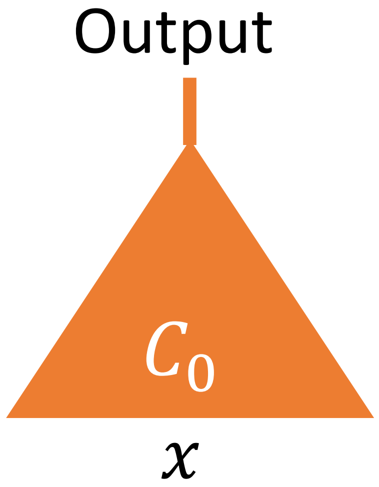
$\theta_1, \theta_2, \ldots, \theta_\ell$: $\mathcal{EF}$-proof of $C_0(x) \leftrightarrow C_1(x)$



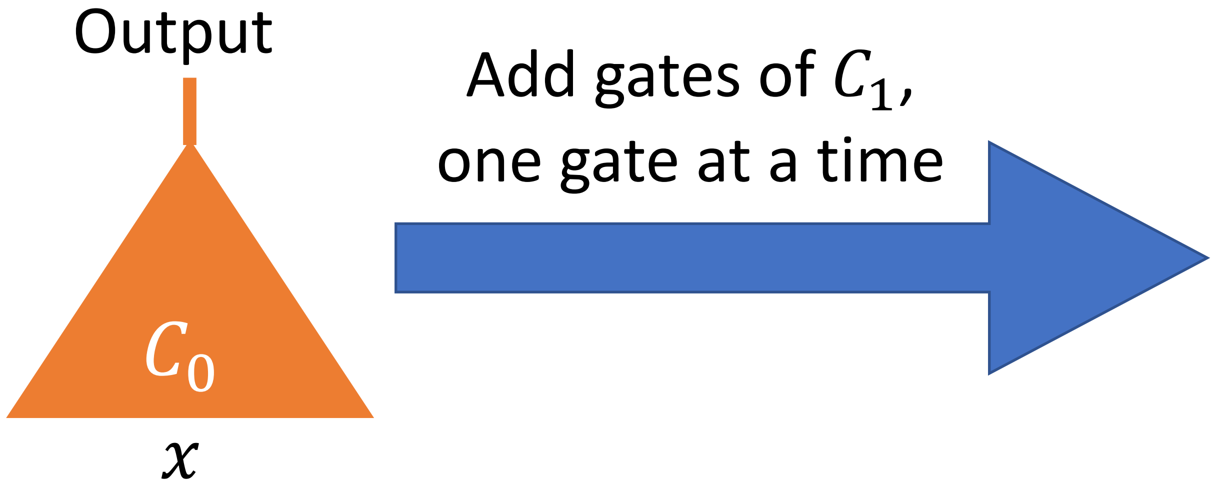
A sequence of **incremental** changes from C_0 to C_1

$C_0 \to C_1$, Stage I: “Grow” C_1

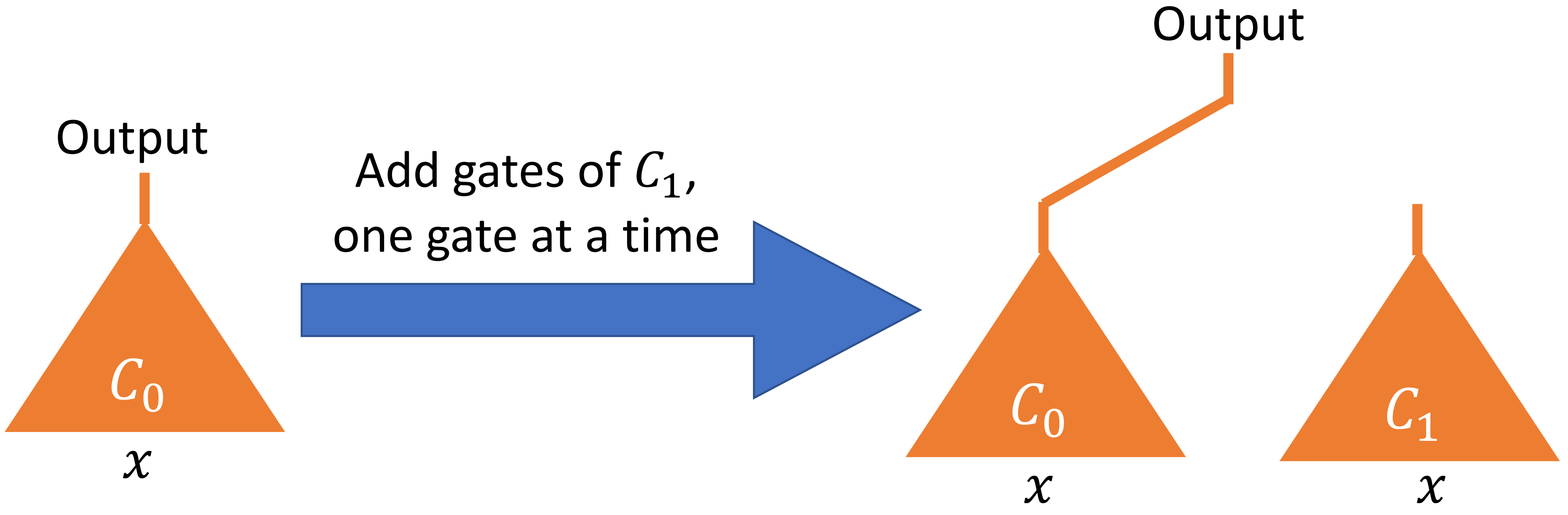
$C_0 \to C_1$, Stage I: "Grow" C_1



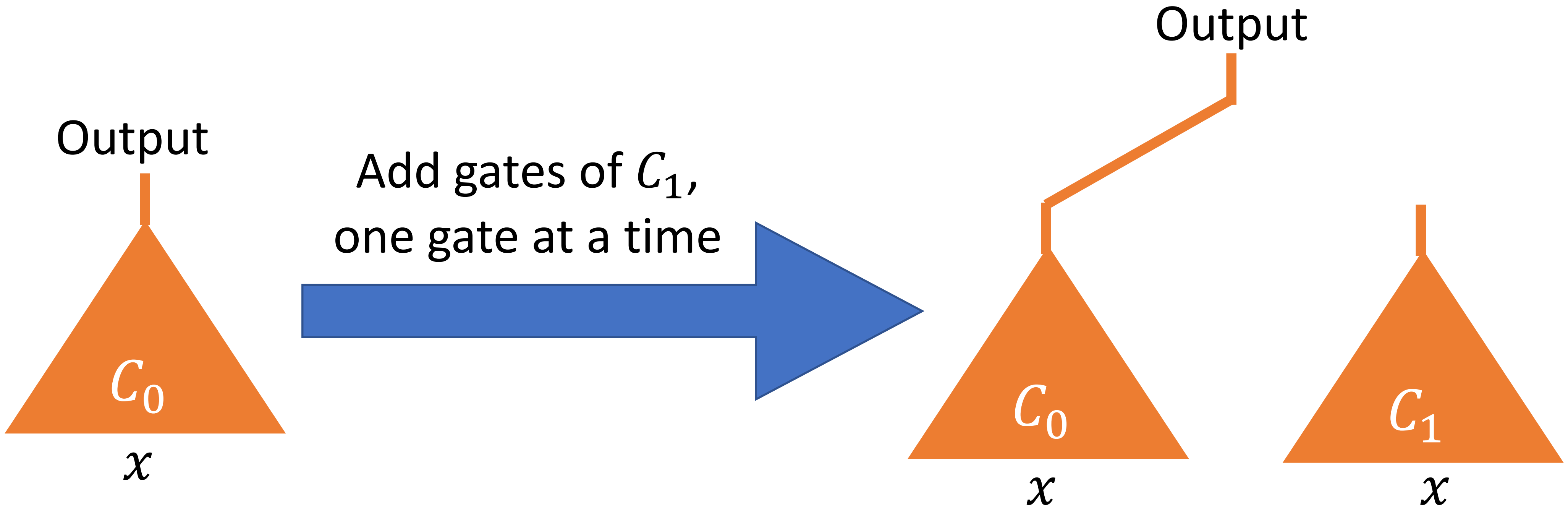
$C_0 \to C_1$, Stage I: "Grow" C_1



$C_0 \to C_1$, Stage I: “Grow” C_1



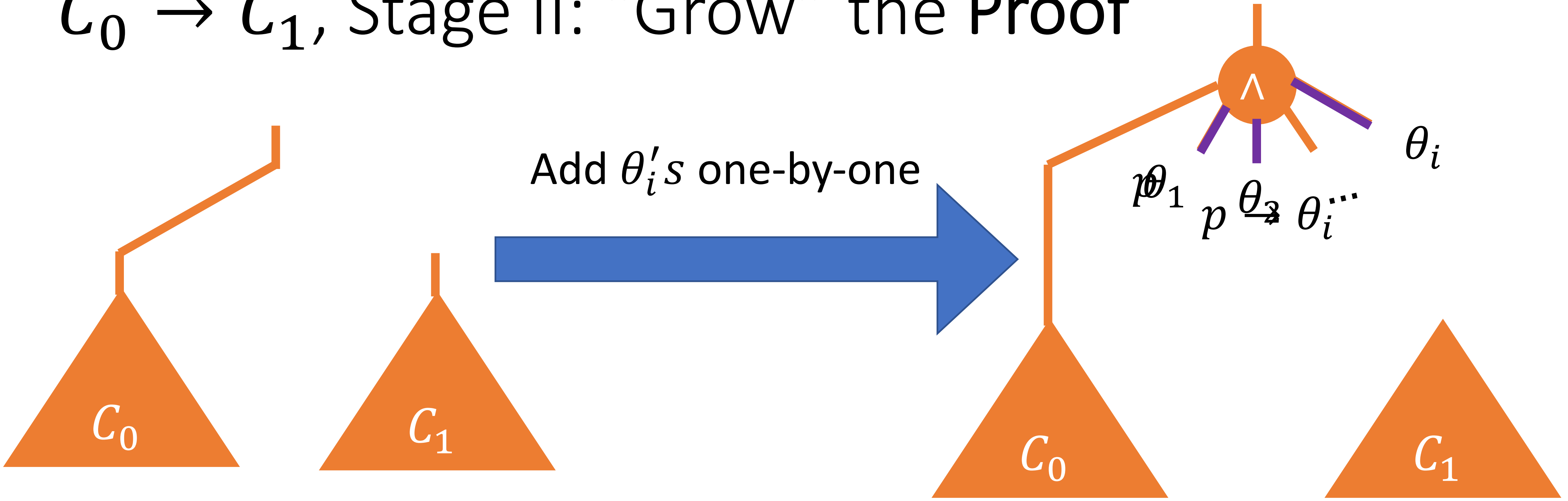
$C_0 \to C_1$, Stage I: "Grow" C_1



δ-Equivalence:

- We only add 1 gate at a time
- Gate we add doesn't affect output

$C_0 \to C_1$, Stage II: "Grow" the **Proof**

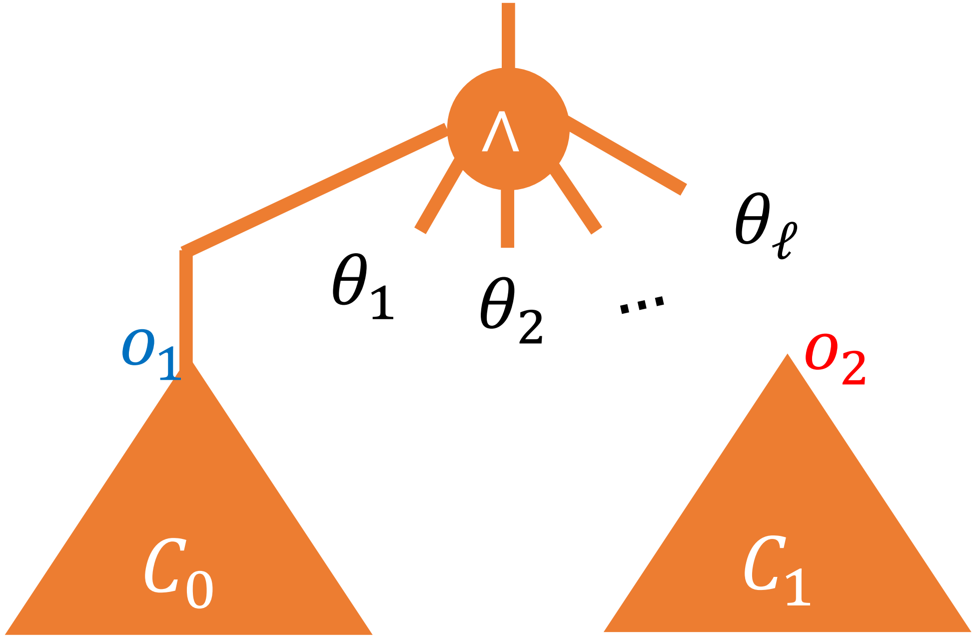


***δ*-Equivalence**: θ_i is from...

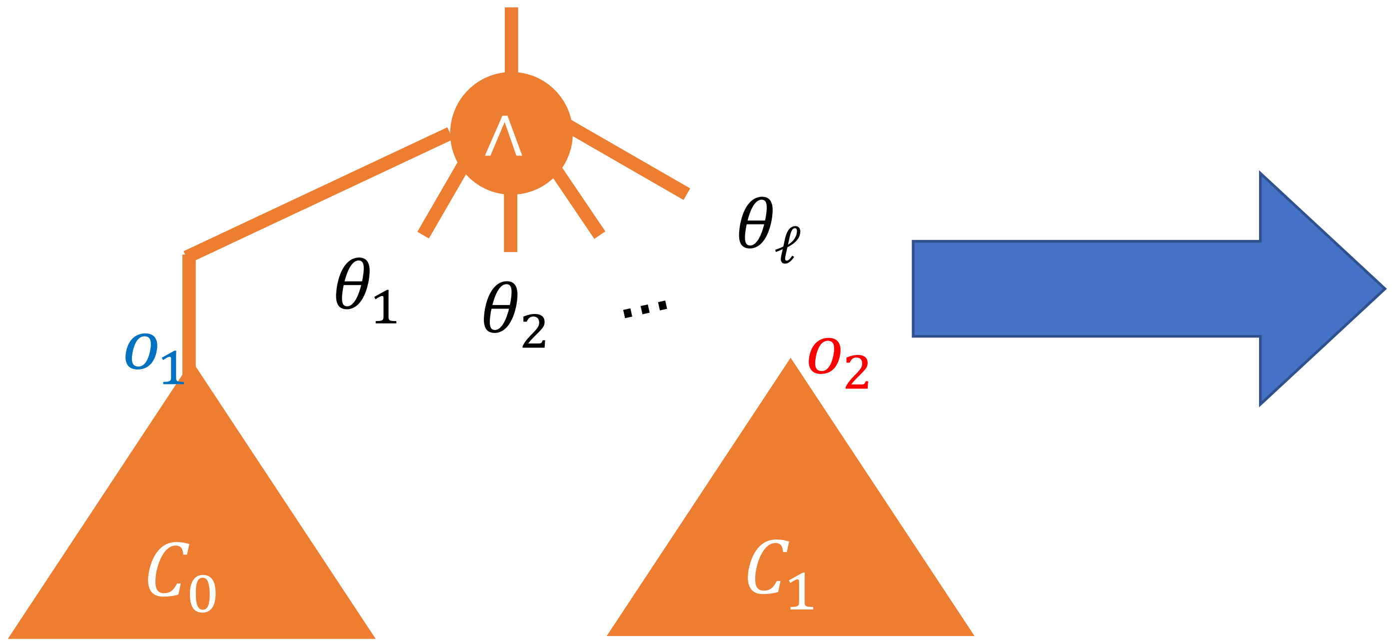
- Axiom: θ_i a tautology, itself is the sub-ckt
- Modus Ponens: $p \wedge (p \to \theta_i) = p \wedge (p \to \theta_i) \wedge \theta_i$

$C_0 \to C_1$, Stage III: **Switch** the Output

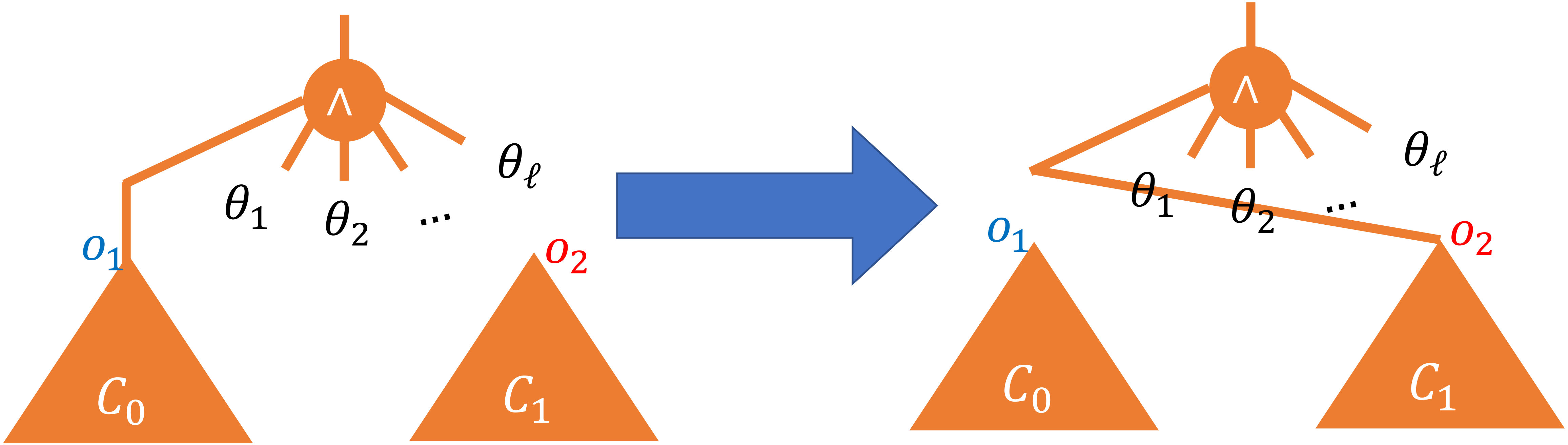
$C_0 \rightarrow C_1$, Stage III: **Switch** the Output



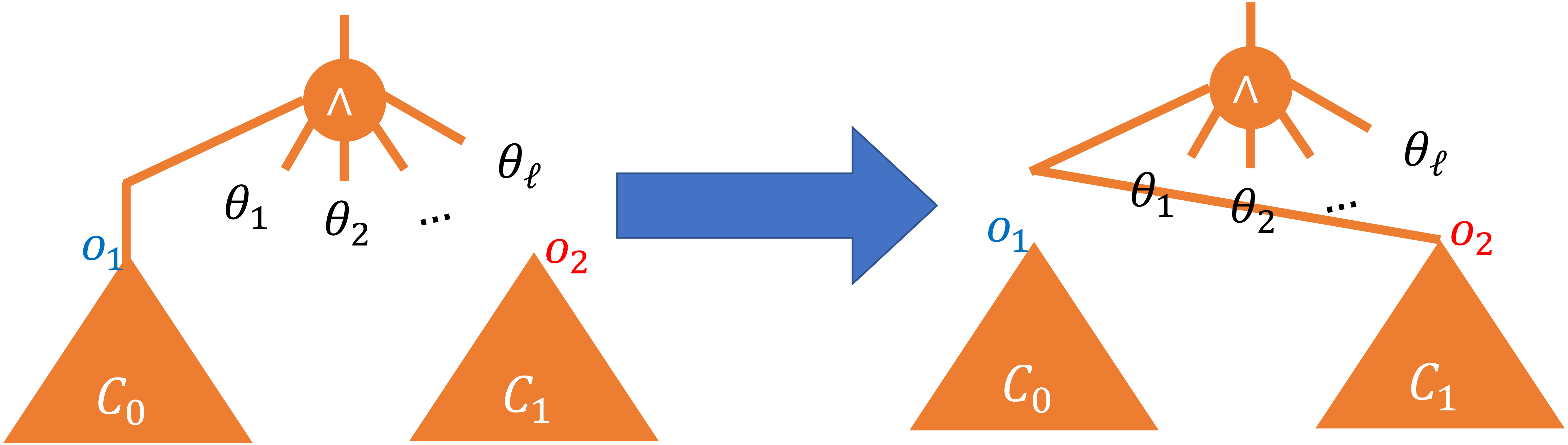
$C_0 \to C_1$, Stage III: **Switch** the Output



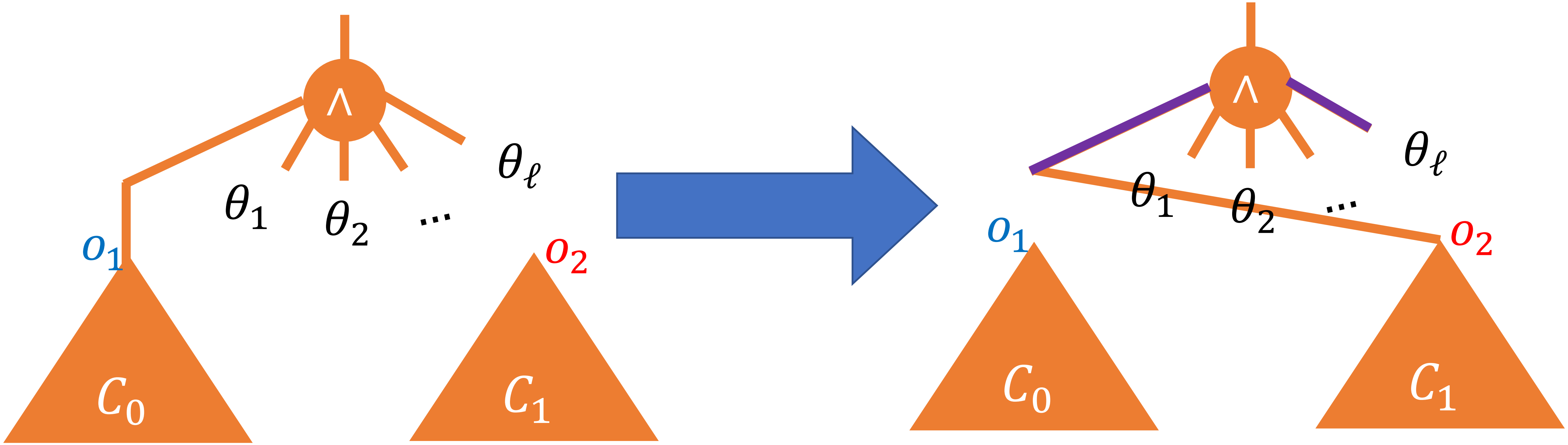
$C_0 \rightarrow C_1$, Stage III: **Switch** the Output



$C_0 \to C_1$, Stage III: **Switch** the Output



$C_0 \to C_1$, Stage III: **Switch** the Output

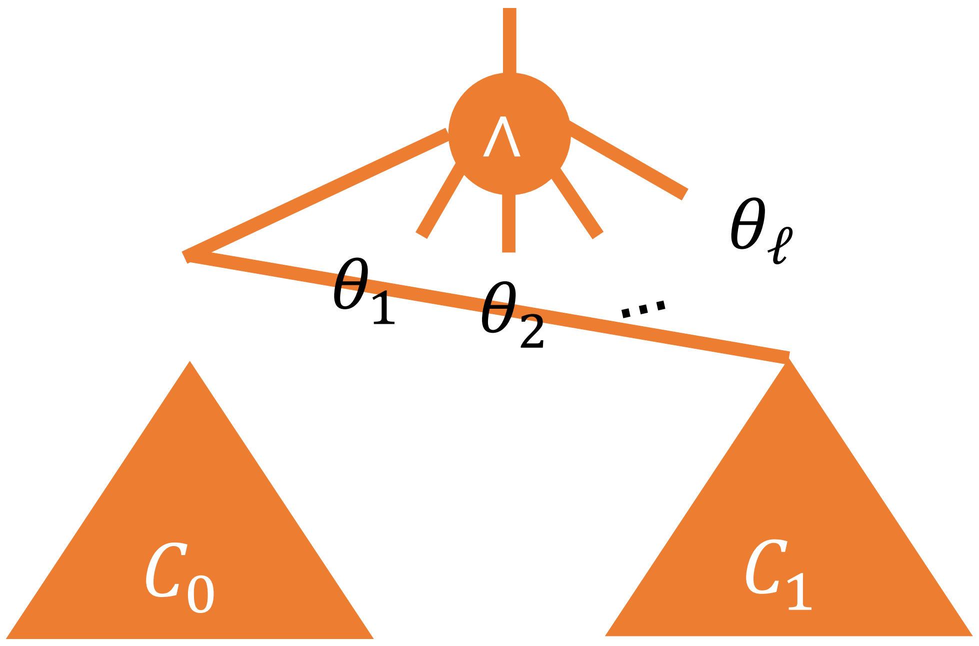


***δ*-Equivalence**: $\theta_\ell =$ "$o_1 \leftrightarrow o_2$",

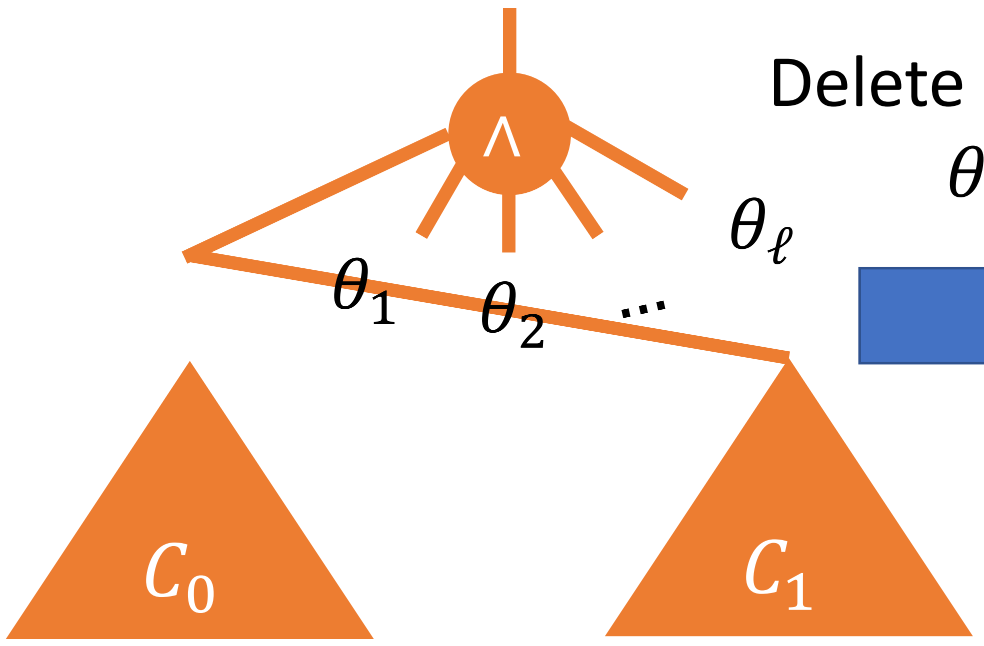
$$o_1 \wedge (o_1 \leftrightarrow o_2) = o_2 \wedge (o_1 \leftrightarrow o_2)$$

$C_0 \to C_1$, Stage IV: "**Shrink**" the Proof

$C_0 \to C_1$, Stage IV: "**Shrink**" the Proof

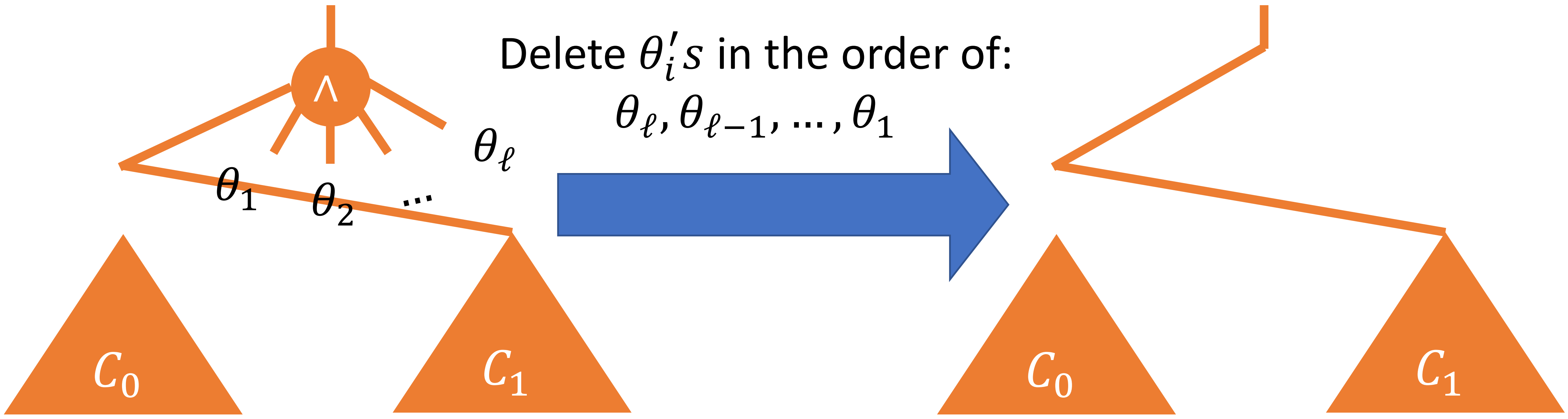


$C_0 \rightarrow C_1$, Stage IV: "**Shrink**" the Proof

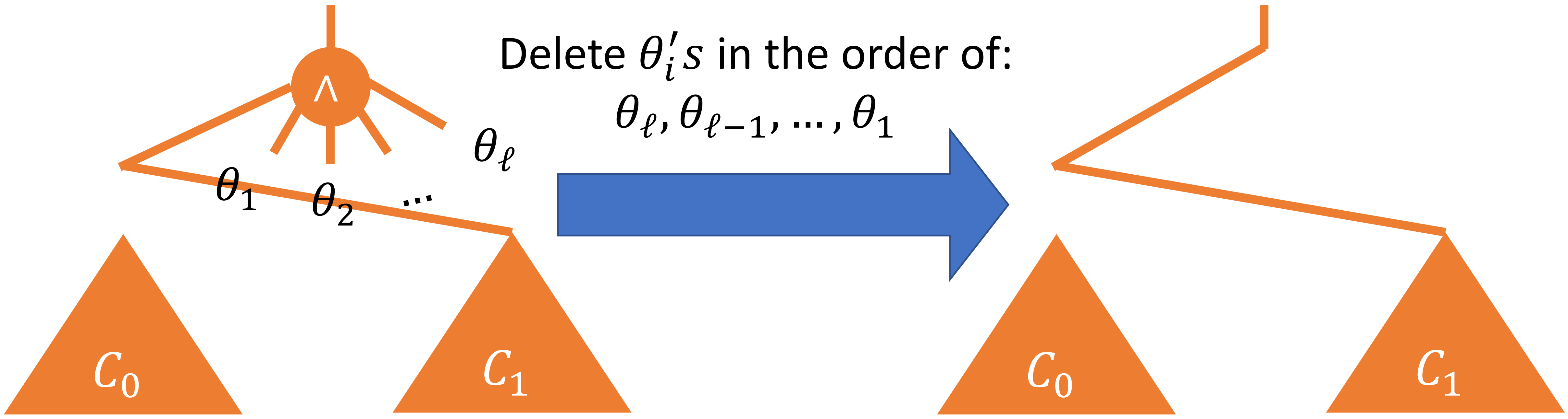


Delete $\theta_i' s$ in the order of:
$\theta_\ell, \theta_{\ell-1}, \dots, \theta_1$

$C_0 \to C_1$, Stage IV: "**Shrink**" the Proof



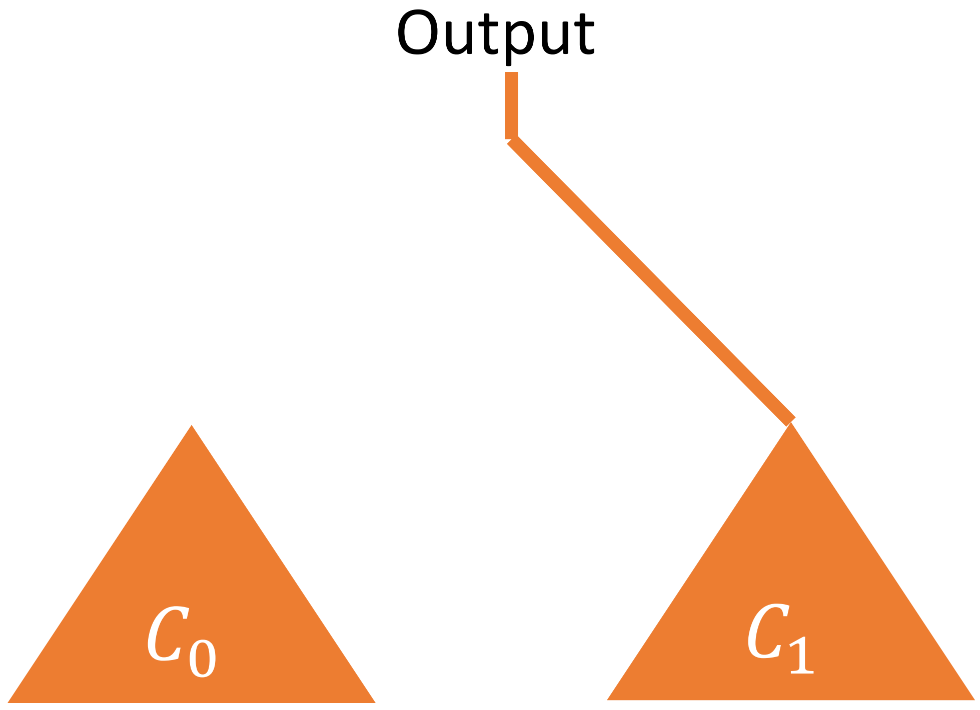
$C_0 \to C_1$, Stage IV: "**Shrink**" the Proof



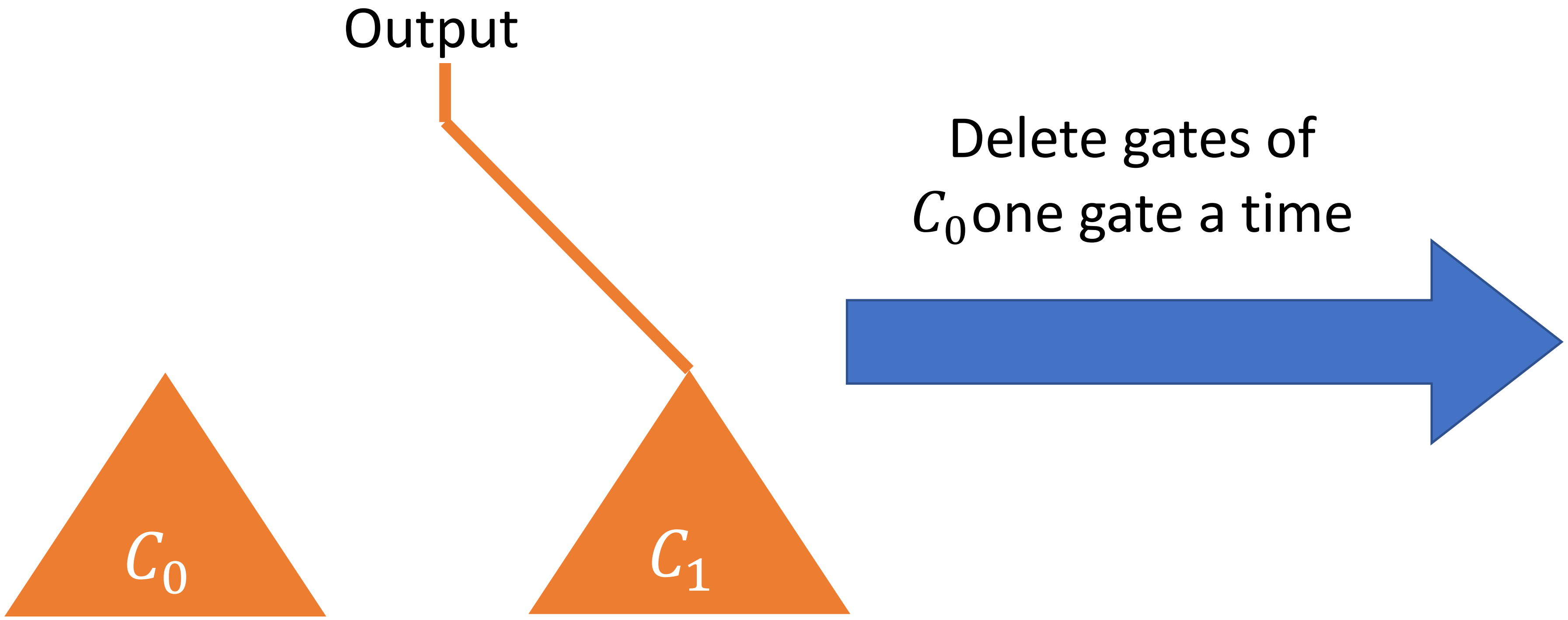
δ-Equivalence: same as the growing phase

$C_0 \to C_1$, Stage V: “Shrink” C_0

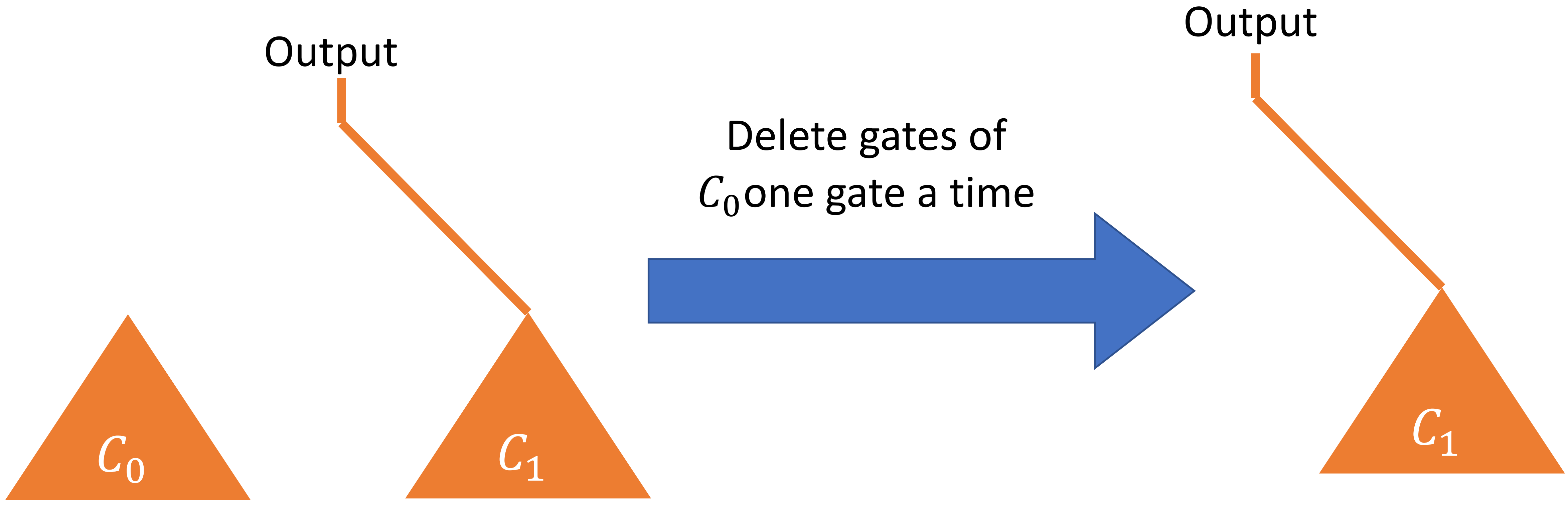
$C_0 \to C_1$, Stage V: "Shrink" C_0



$C_0 \rightarrow C_1$, Stage V: "Shrink" C_0

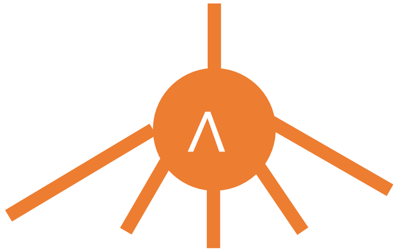


$C_0 \to C_1$, Stage V: “Shrink” C_0

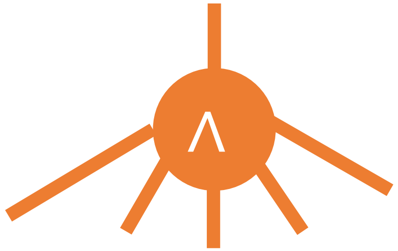


More Details (I): Multi-arity ∧-Gate?

More Details (I): Multi-arity ∧-Gate?

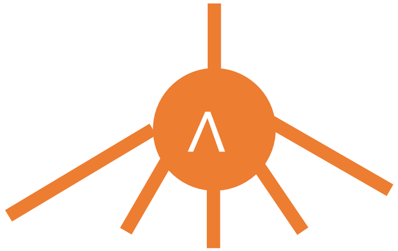


More Details (I): Multi-arity ∧-Gate?



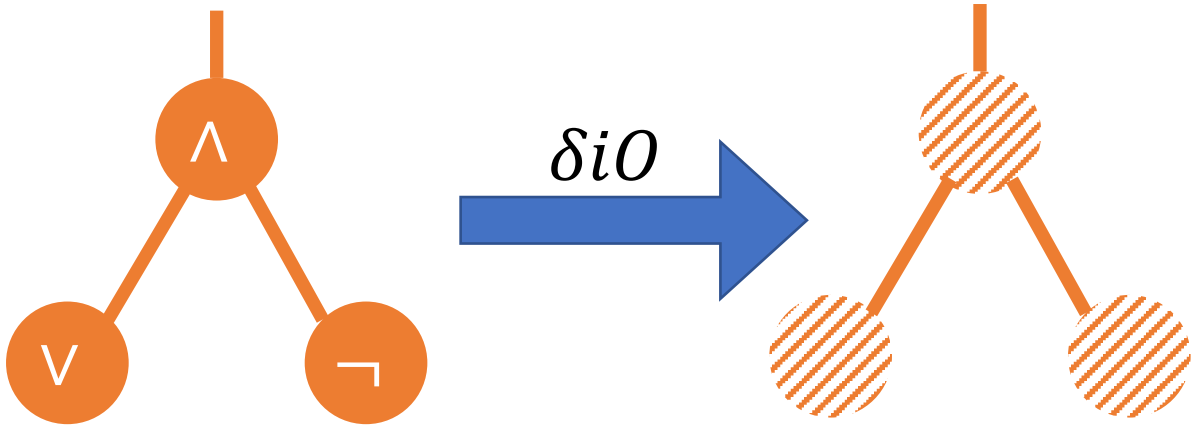
We need small arity, because

More Details (I): Multi-arity ∧-Gate?

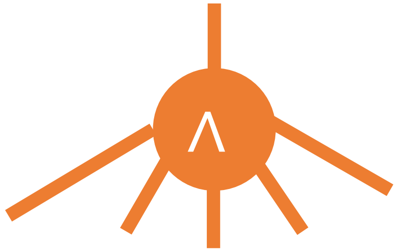


We need small arity, because

δiO relies on **iO for ckt** to obfuscate each gate

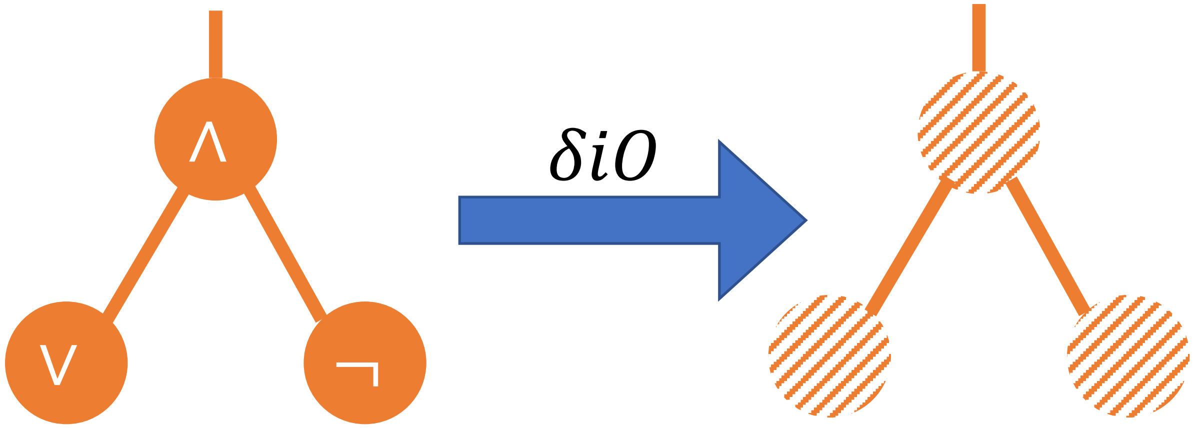


More Details (I): Multi-arity ∧-Gate?



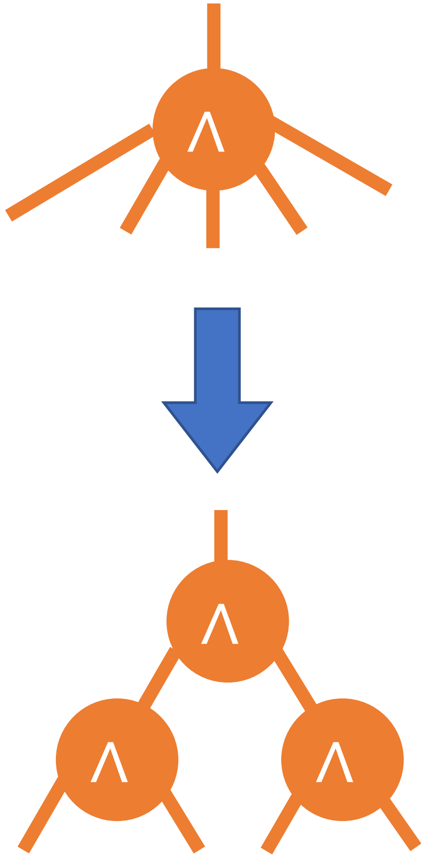
We need small arity, because

δiO relies on **iO for ckt** to obfuscate each gate



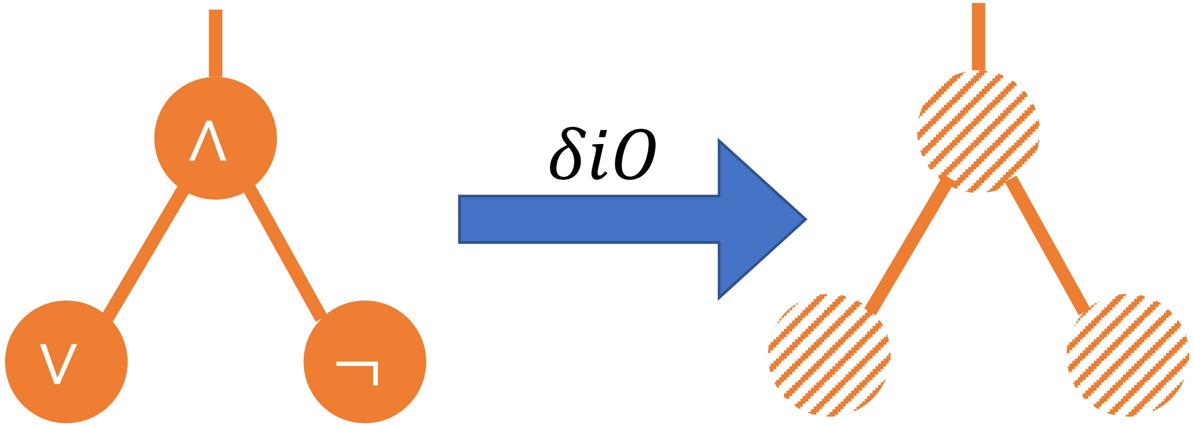
Security Loss $\geq 2^{|arity|}$

More Details (I): Multi-arity ∧-Gate?



We need small arity, because

δiO relies on **iO for ckt** to obfuscate each gate



Security Loss $\geq 2^{|arity|}$

More Details (II): Change of the Topology?

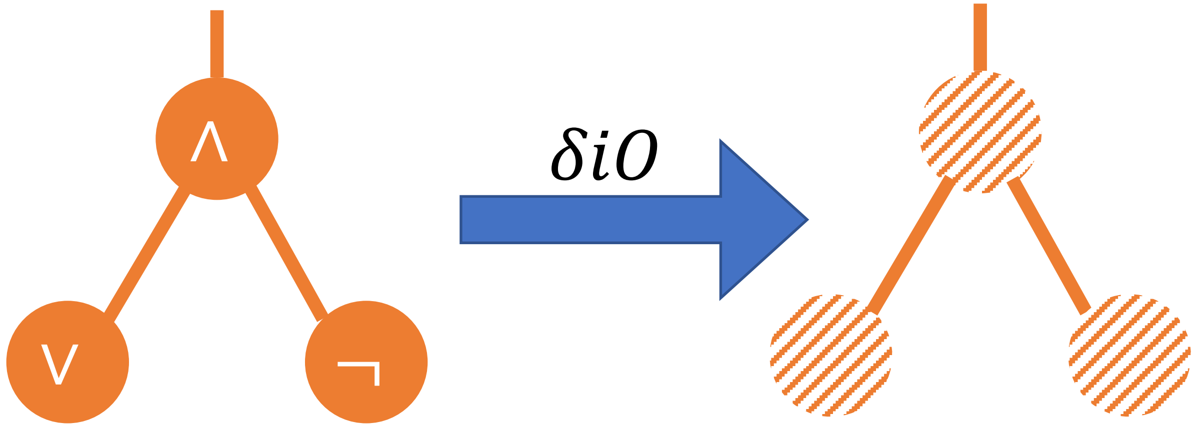
More Details (II): Change of the Topology?

We add/delete gates, but…

More Details (II): Change of the Topology?

We add/delete gates, but…

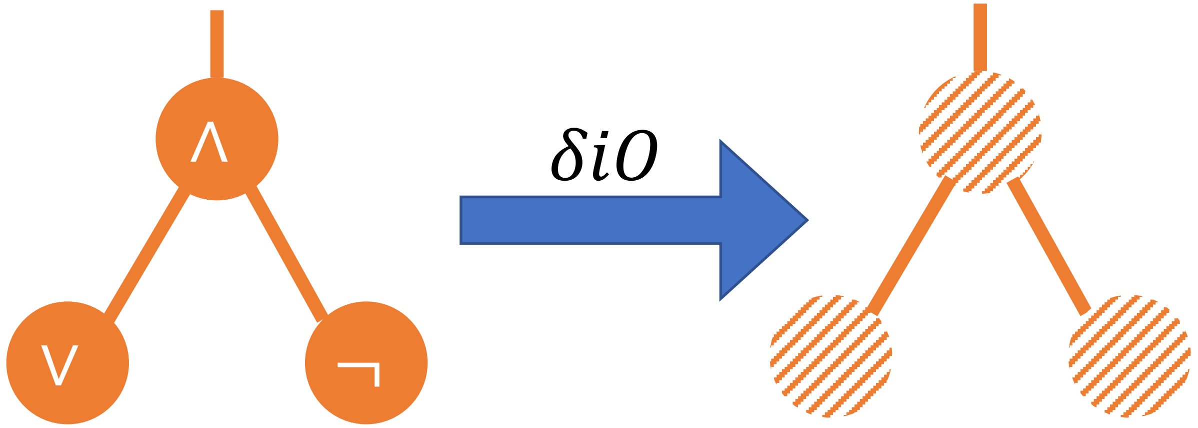
Recall: δiO doesn't hide topology



More Details (II): Change of the Topology?

We add/delete gates, but…

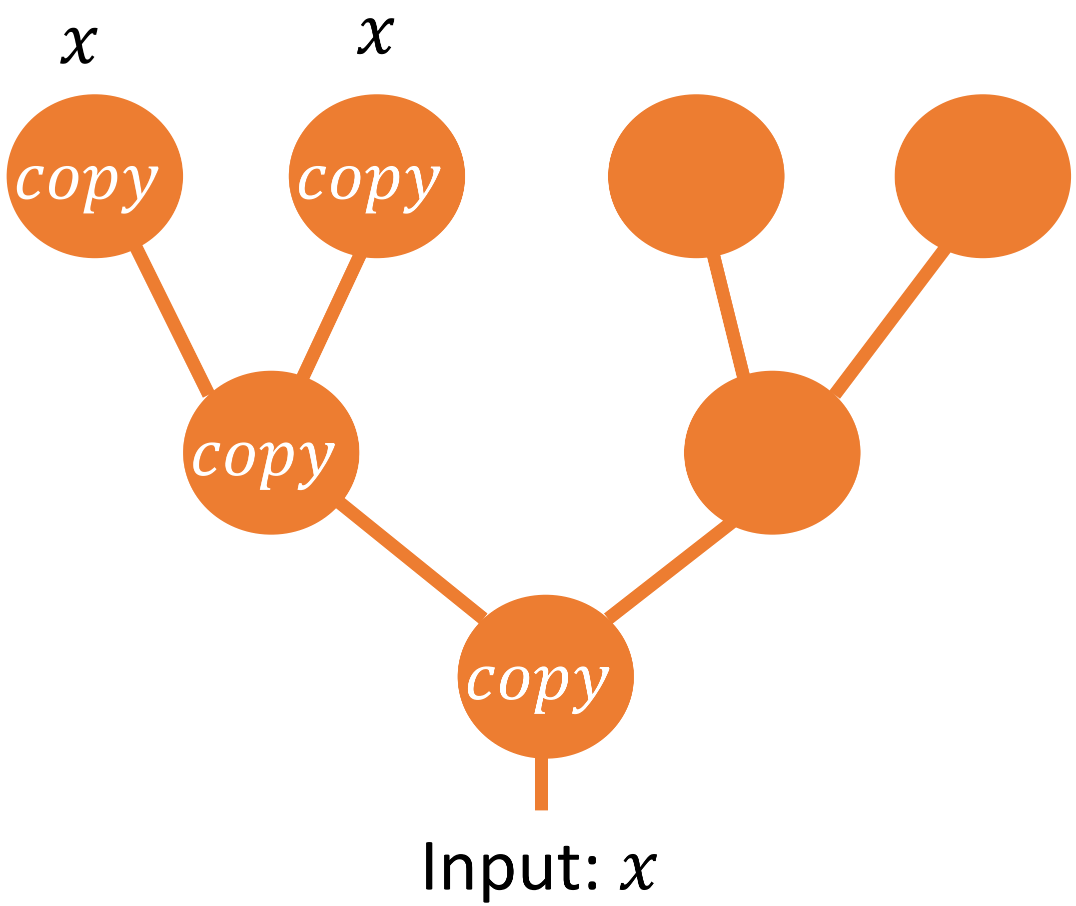
<u>Recall:</u> δiO doesn't hide topology



We can't **change** the **topology**, otherwise we can't apply the security of δiO.

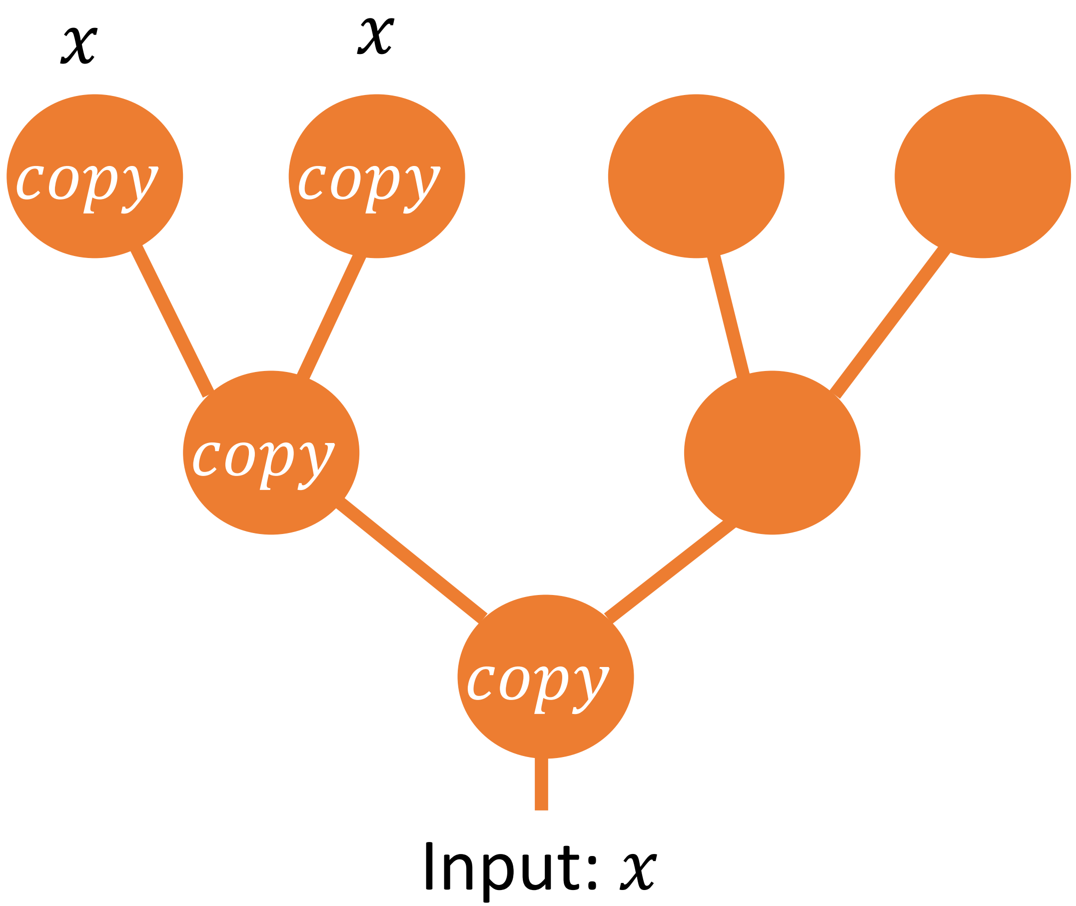
Build “Helper” Sub-Circuits

Build "Helper" Sub-Circuits

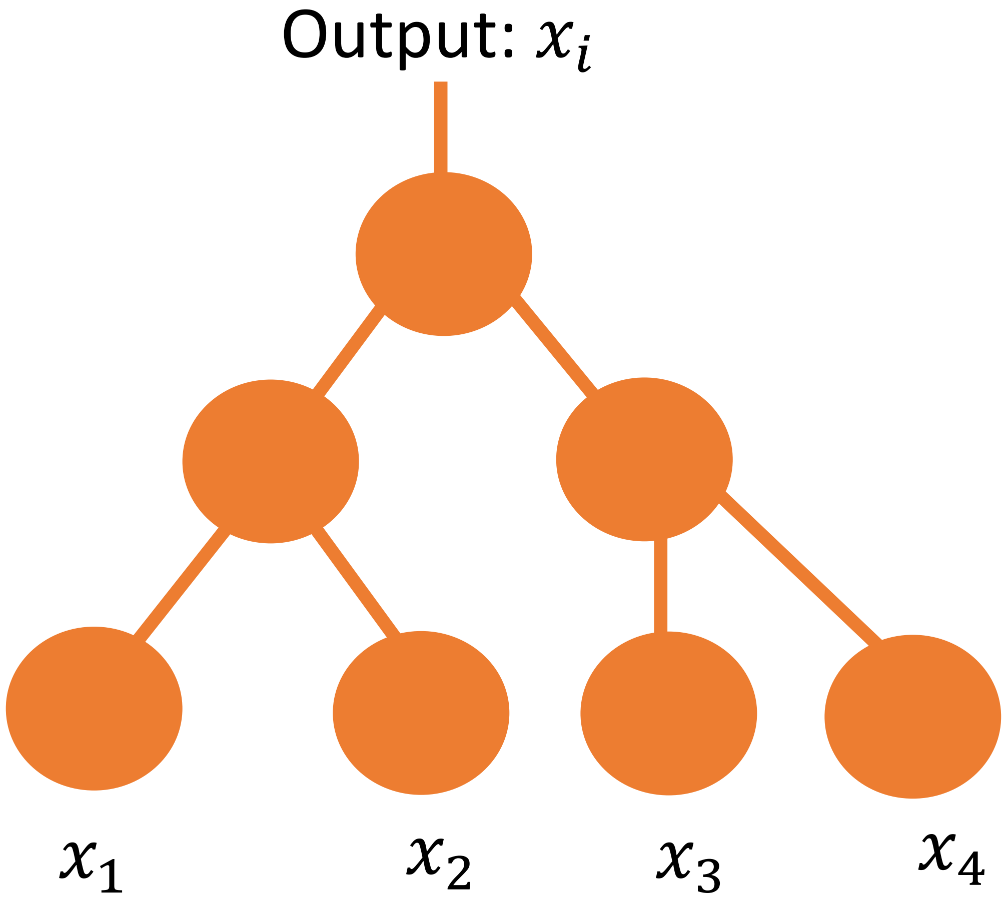


<u>"Copy" circuit</u>

Build "Helper" Sub-Circuits



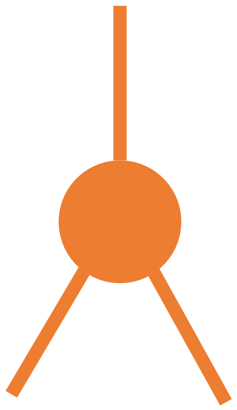
"Copy" circuit



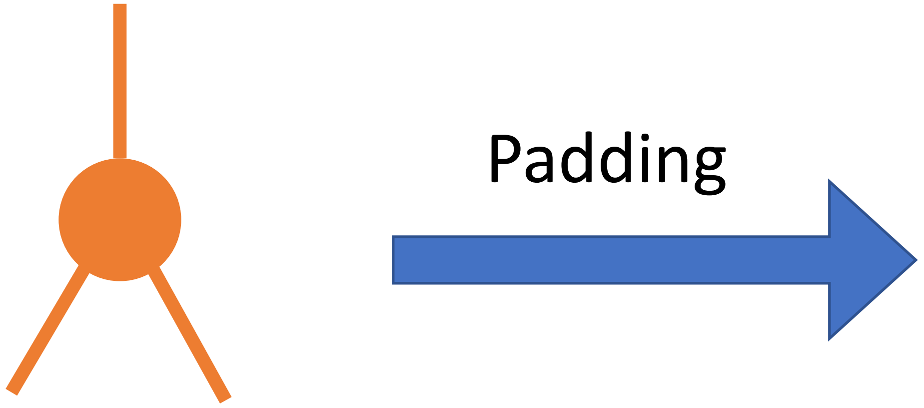
"Project" circuit

Pad the Circuit

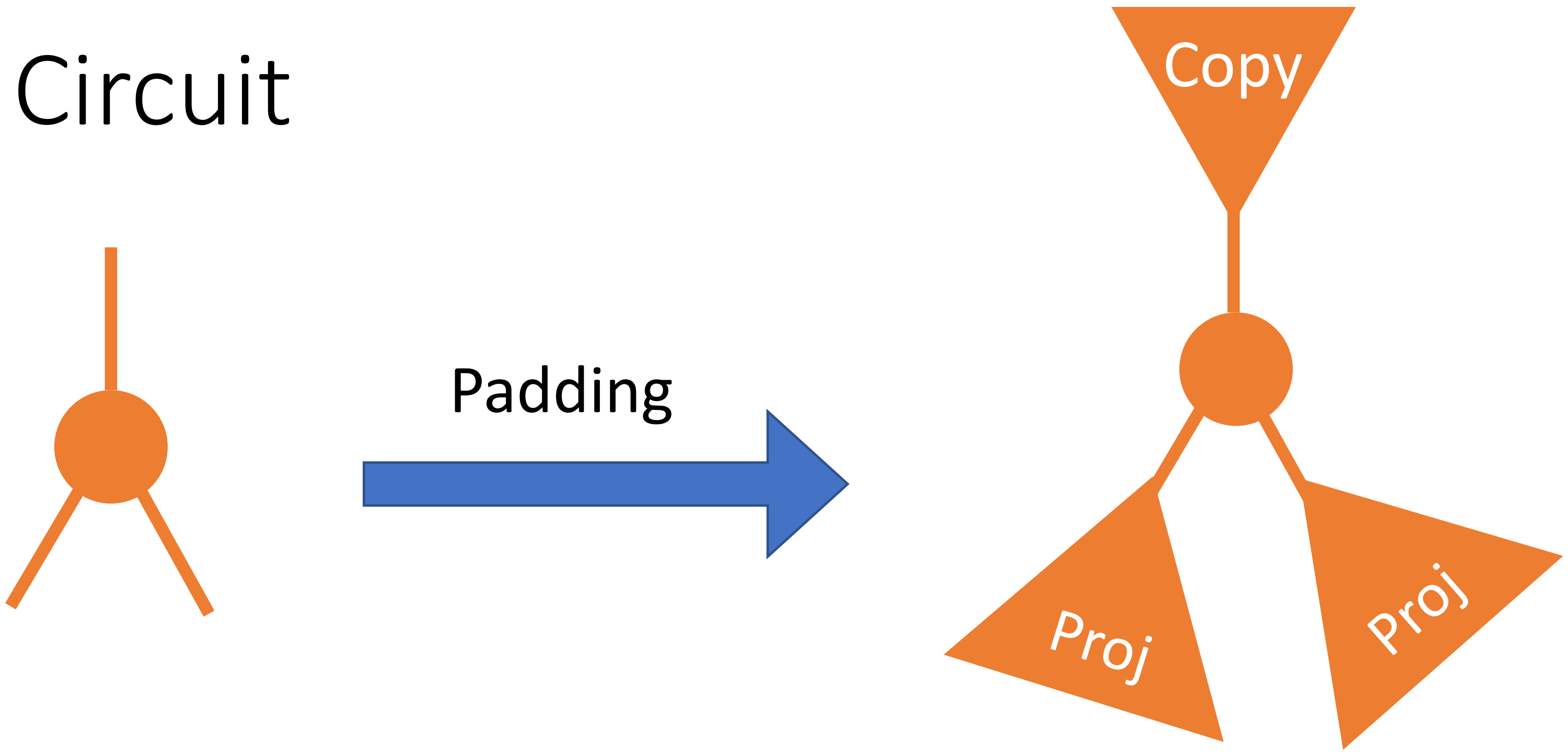
Pad the Circuit



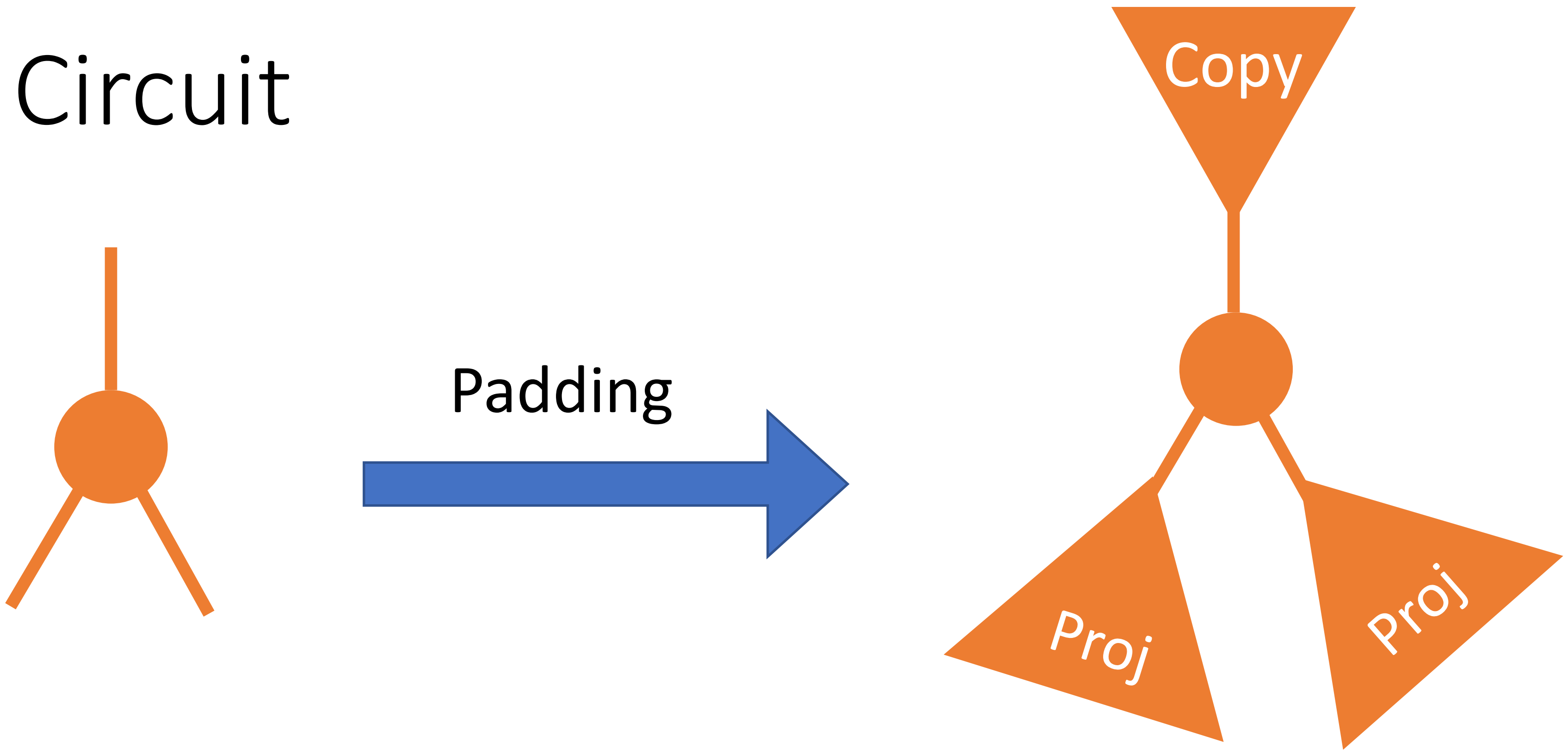
Pad the Circuit



Pad the Circuit

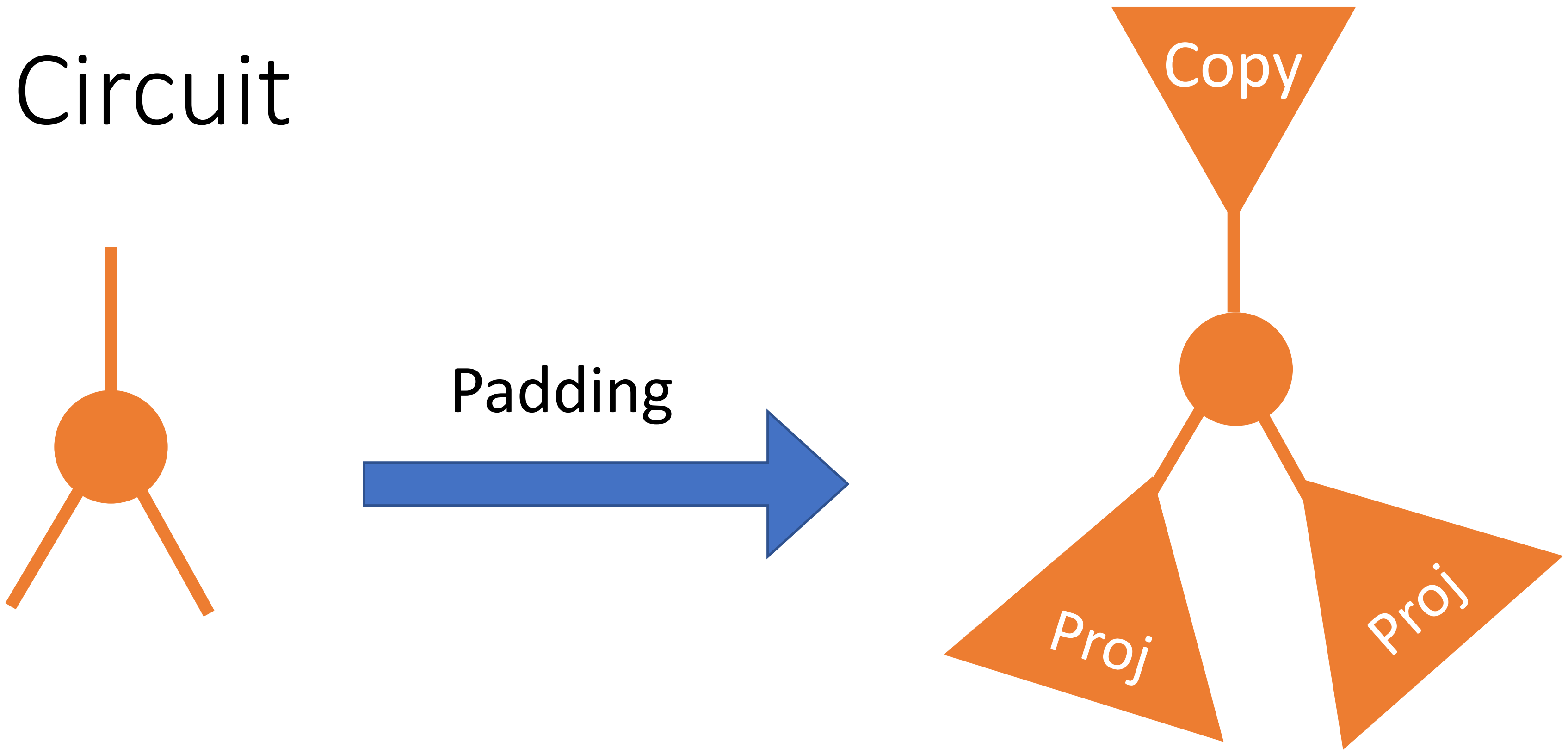


Pad the Circuit



Topology changing operations (Adding/Deleting Gate) becomes changing the functionality of Copy, Proj sub-circuits.

Pad the Circuit



Topology changing operations (Adding/Deleting Gate) becomes changing the functionality of Copy, Proj sub-circuits.

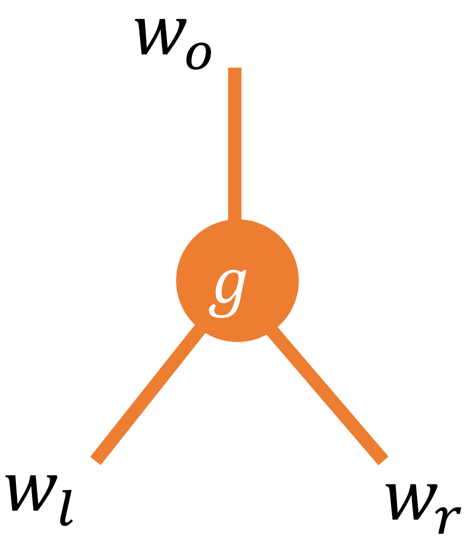
The change is “local” due to the tree structure

Technical Details

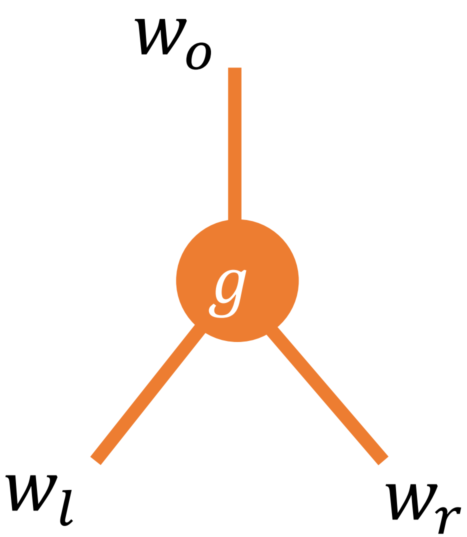
- δ-Equivalence from $\mathcal{EF}$-proofs
- $\boldsymbol{\delta i\mathcal{O}}$ **Construction**
- iO for Turing Machines

Construct δiO: Initial Idea

Construct δiO: Initial Idea

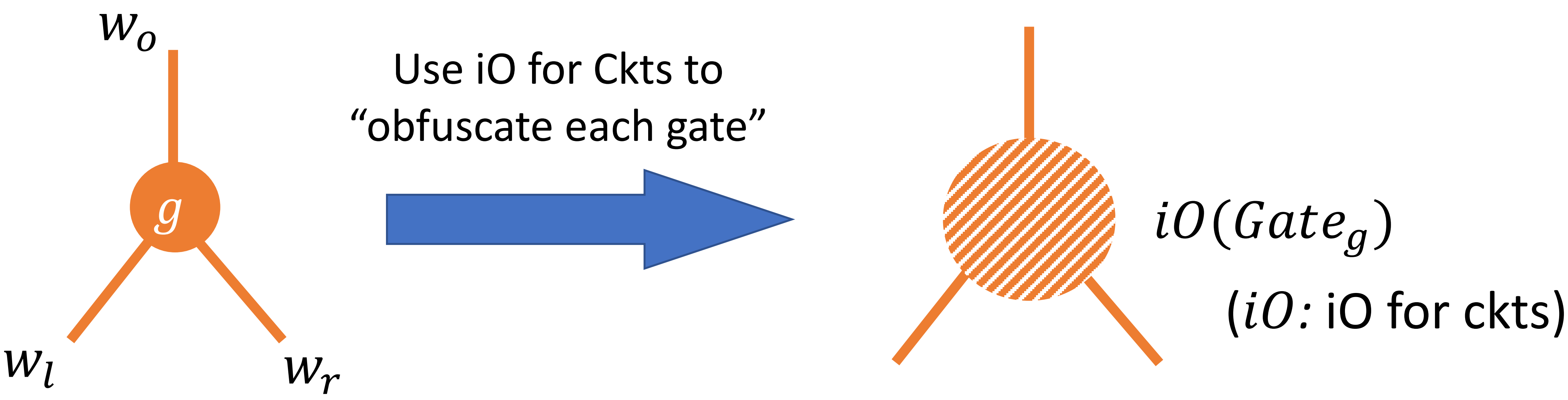


Construct δiO: Initial Idea

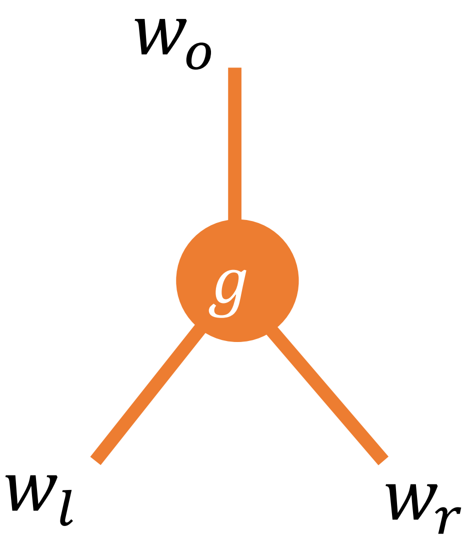


Use iO for Ckts to
"obfuscate each gate"

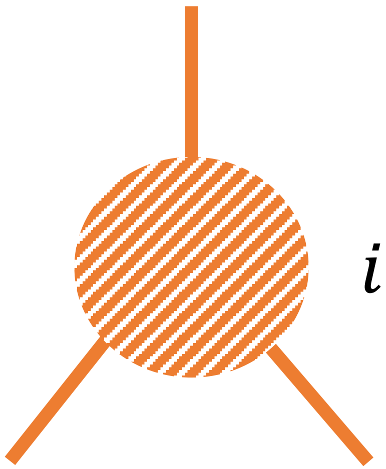
Construct δiO: Initial Idea



Construct δiO: Initial Idea



Use iO for Ckts to
"obfuscate each gate"

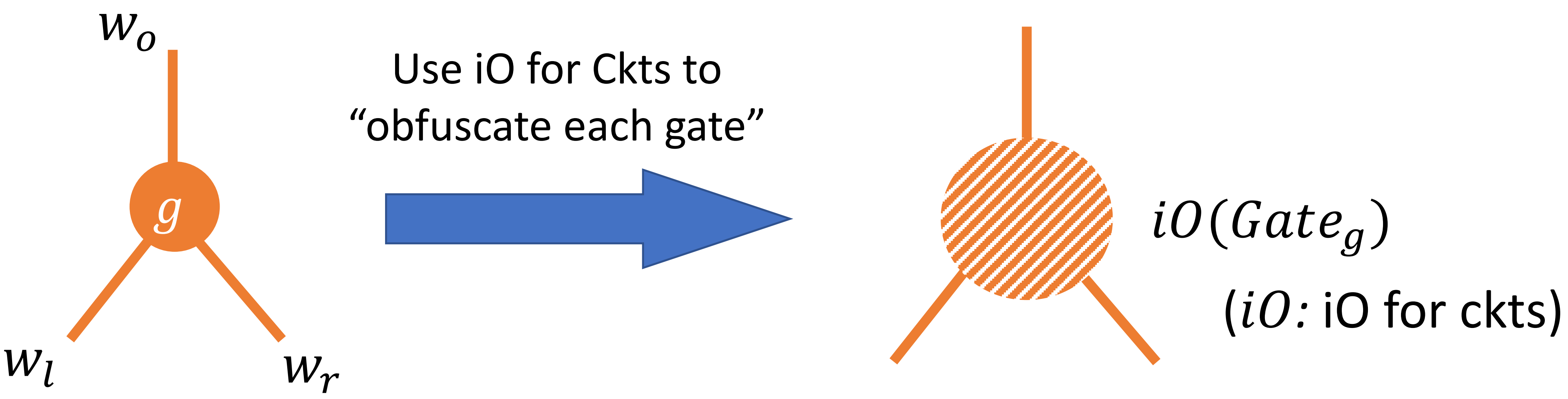


$iO(Gate_g)$

(iO: iO for ckts)

$Gate_g(w_l, w_r)$: Output $w_o = g(w_l, w_r)$

Construct δiO: Initial Idea

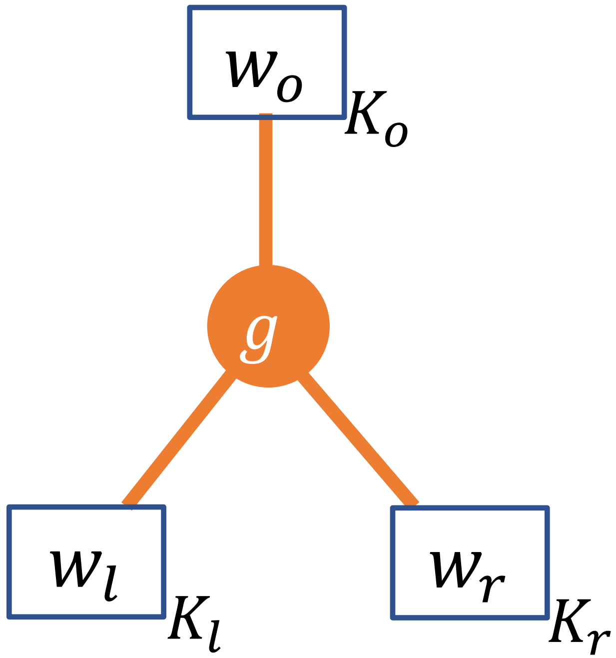


$Gate_g(w_l, w_r)$: Output $w_o = g(w_l, w_r)$

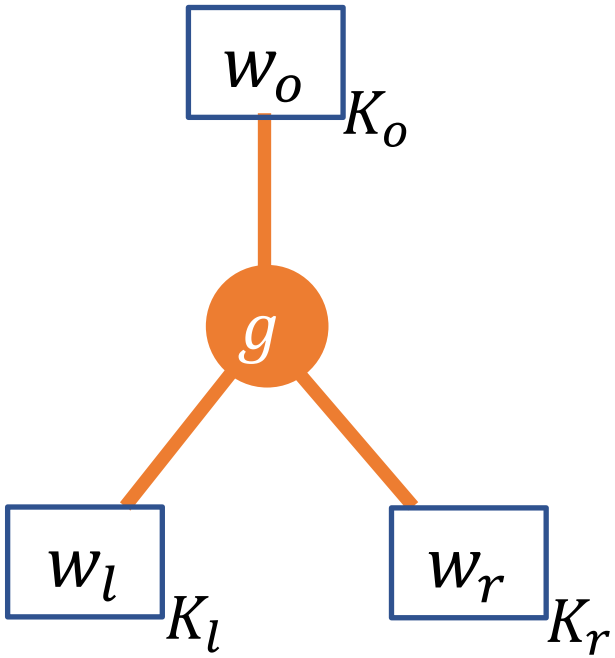
The adversary can learn the gate from its truth table.

Encrypt Wire Values

Encrypt Wire Values

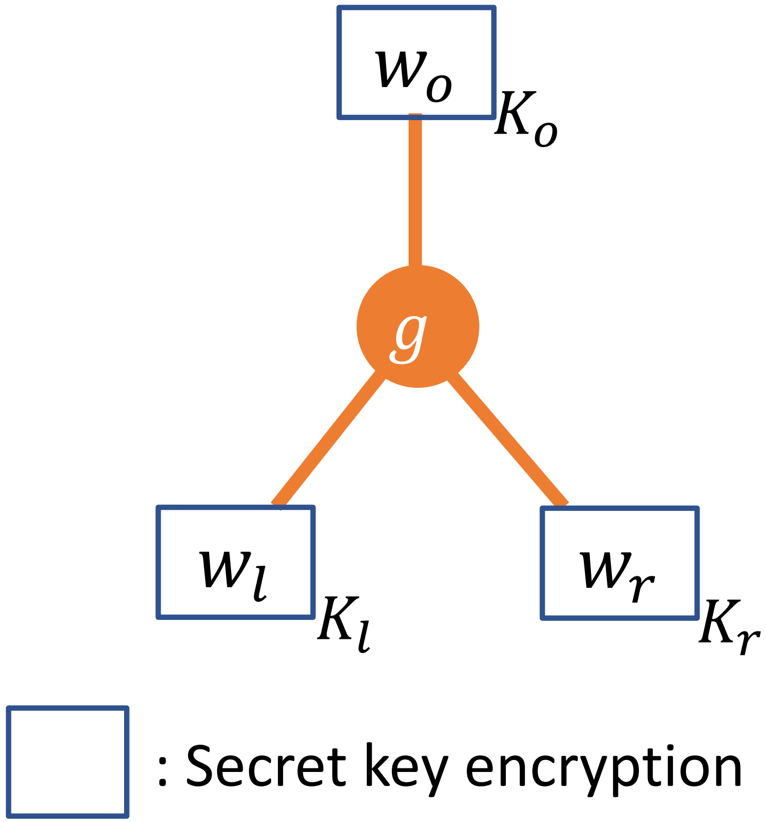


Encrypt Wire Values



: Secret key encryption

Encrypt Wire Values



$\underline{Gate_g}$:

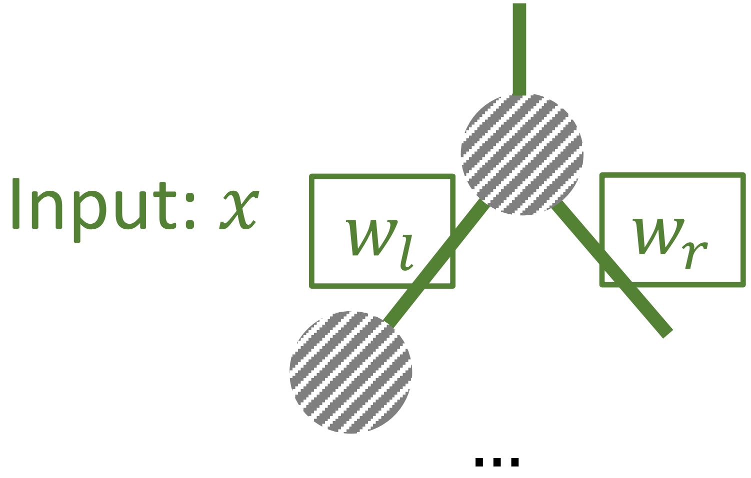
- **Input**: Ciphertexts of w_l, w_r

  Decrypt the input wires w_l, w_r
  Compute gate g: $w_o = g(w_l, w_r)$
  Encrypt the output wire w_o

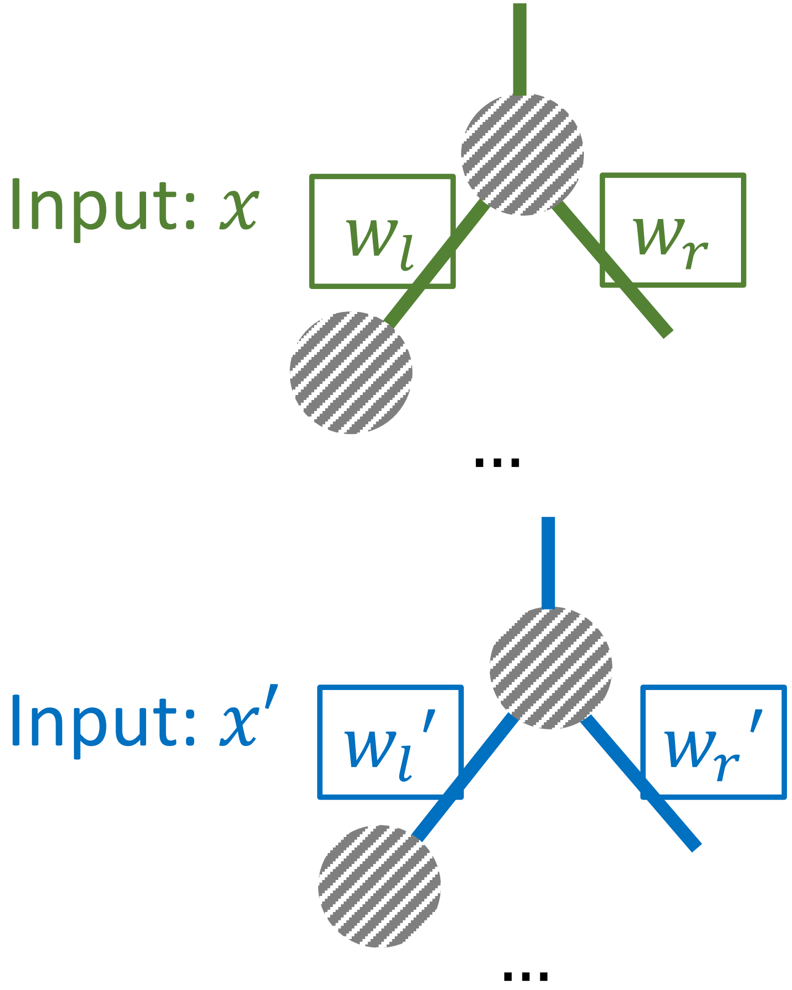
- **Output**: Ciphertext of w_o,

Mix-and-Match Attack

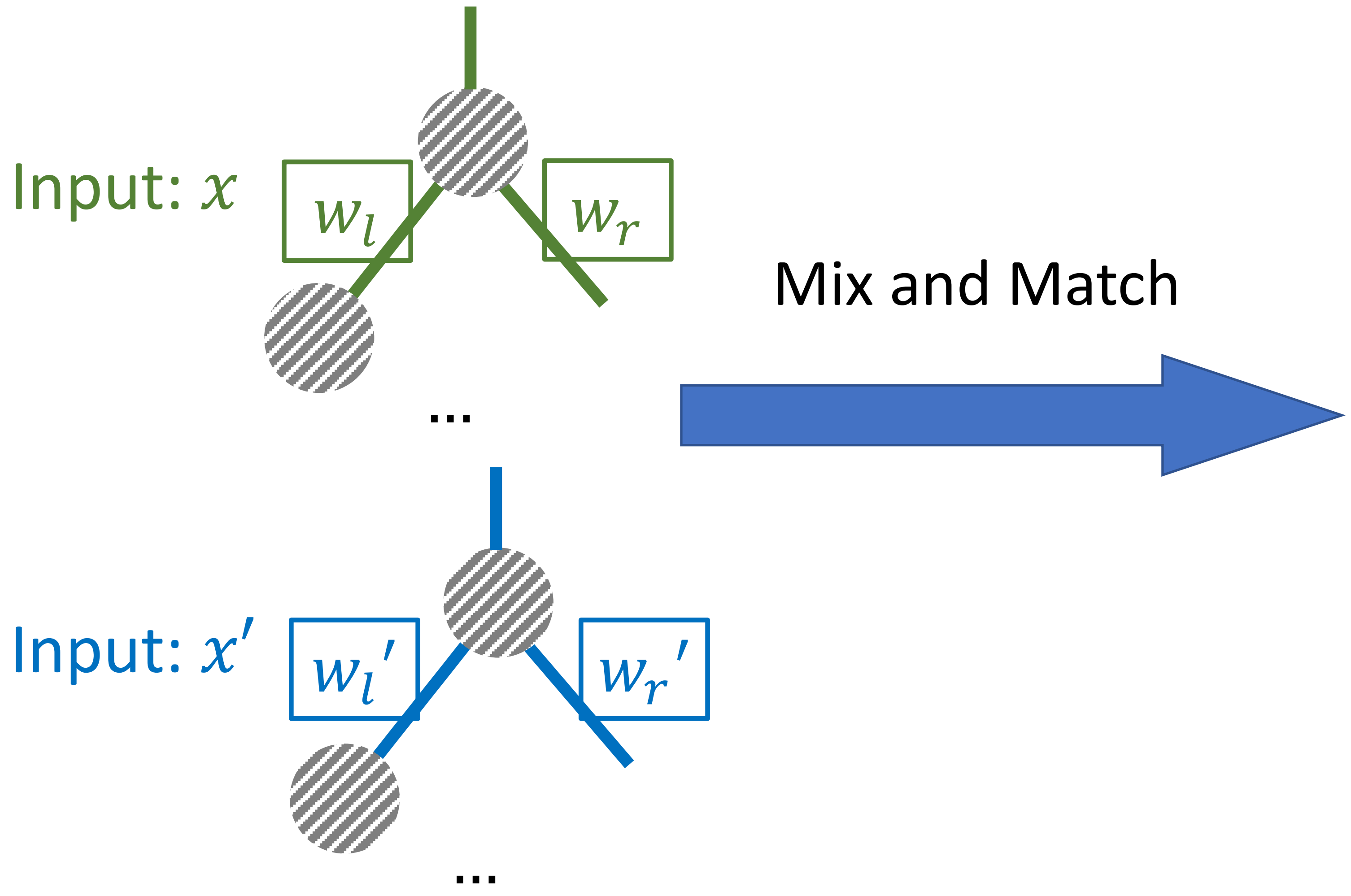
Mix-and-Match Attack



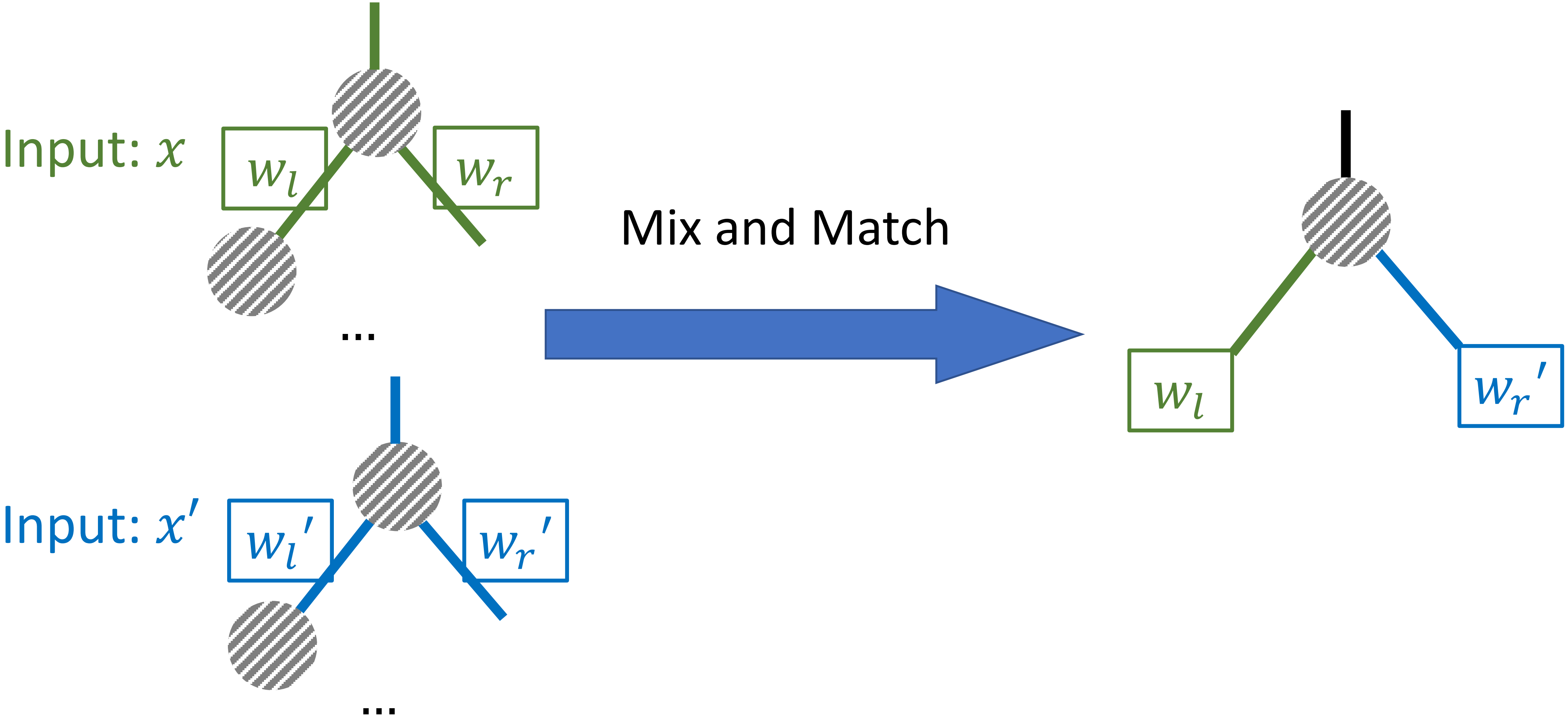
Mix-and-Match Attack



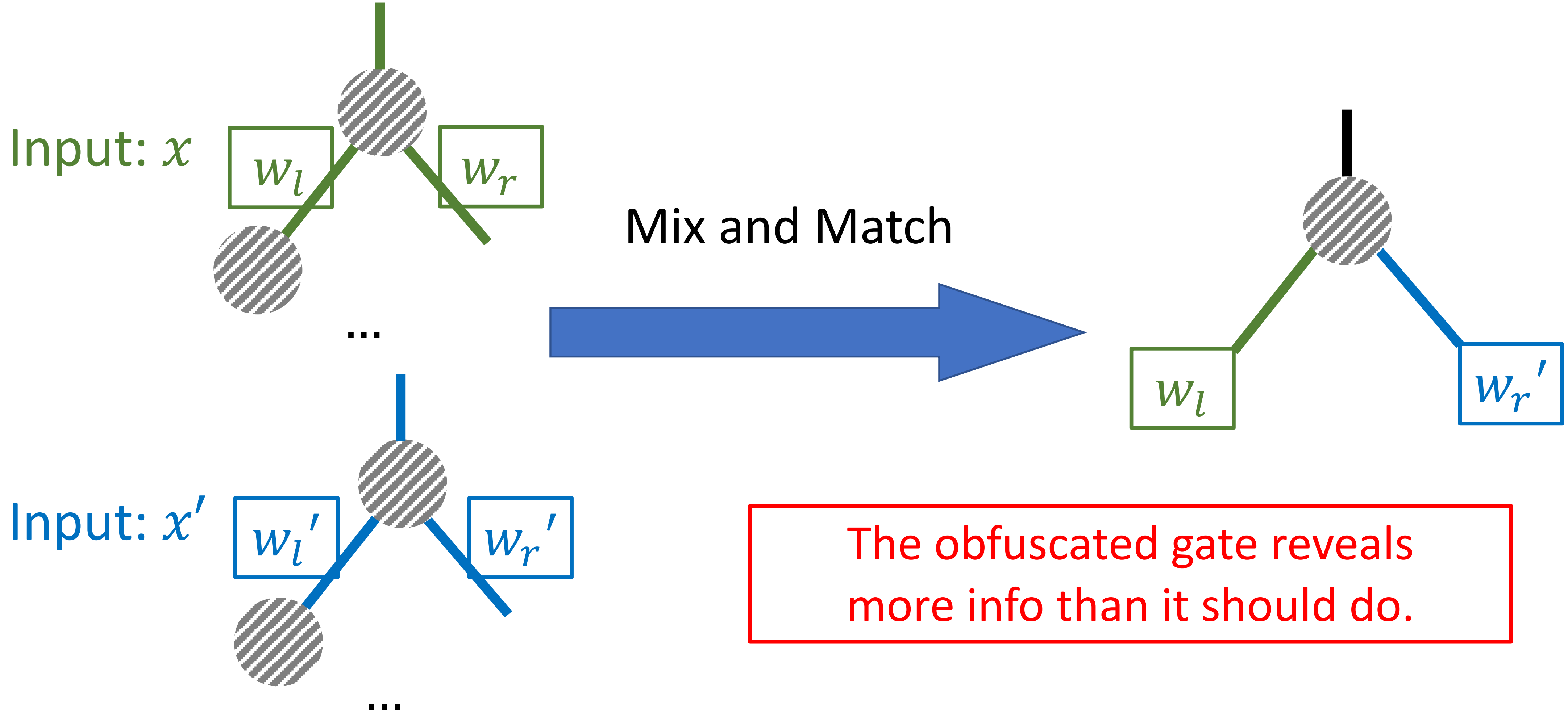
Mix-and-Match Attack



Mix-and-Match Attack

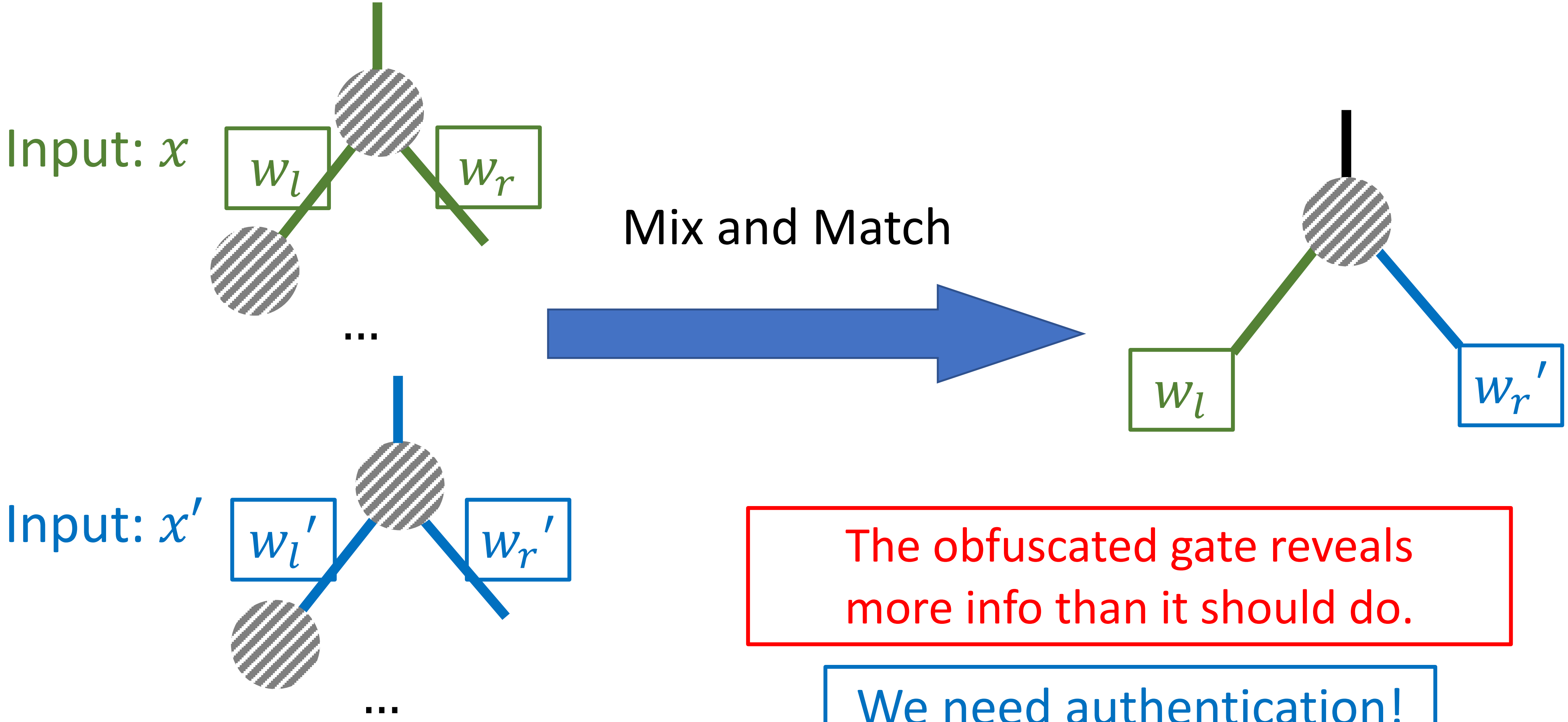


Mix-and-Match Attack



The obfuscated gate reveals more info than it should do.

Mix-and-Match Attack

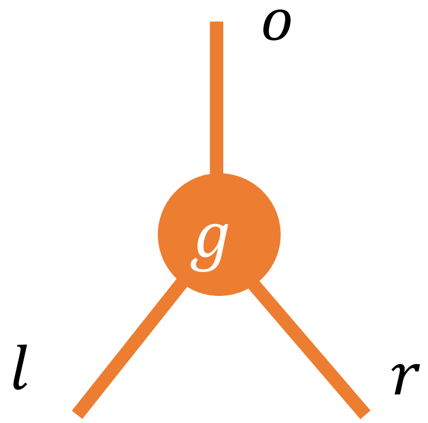


The obfuscated gate reveals more info than it should do.

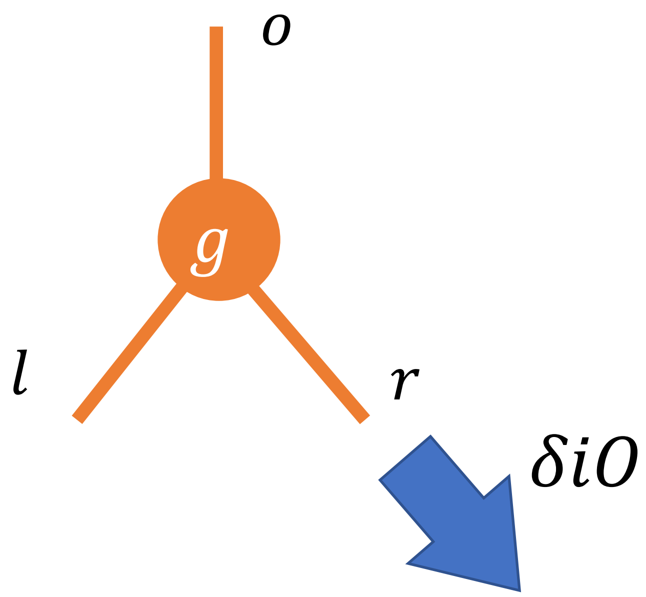
We need authentication!

δiO Construction: Super High Level

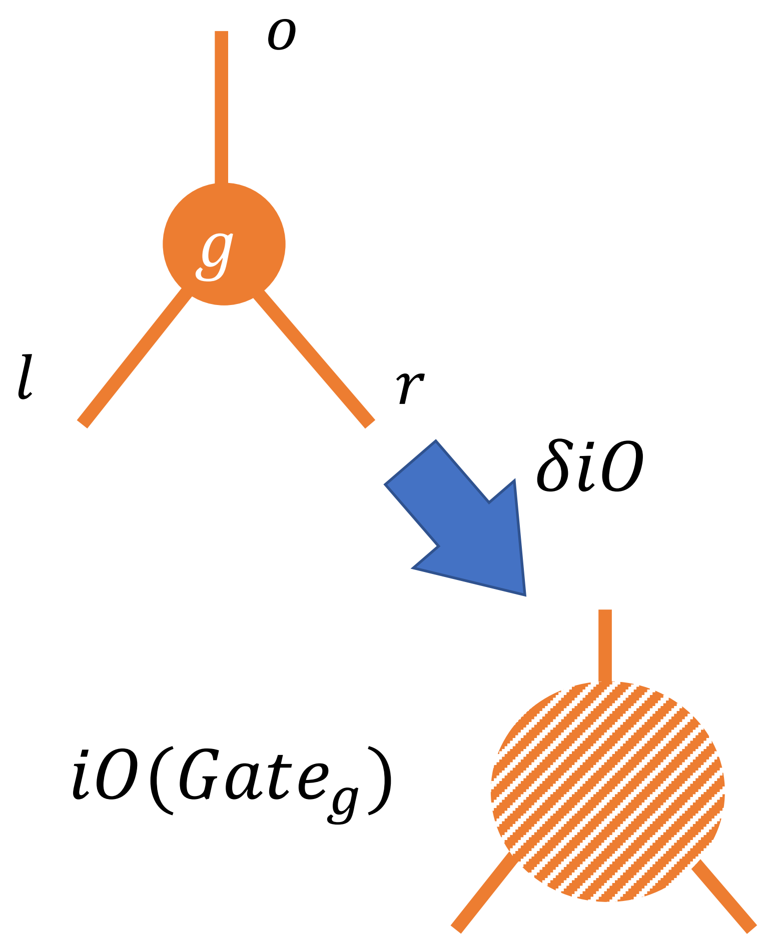
δiO Construction: Super High Level



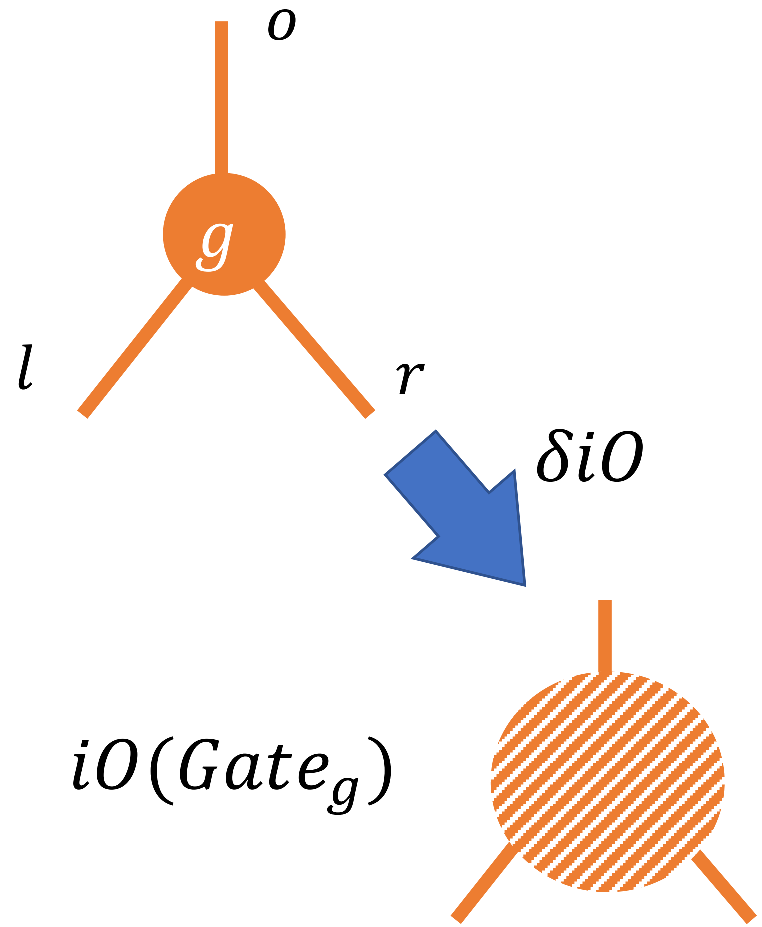
δiO Construction: Super High Level



δiO Construction: Super High Level



δiO Construction: Super High Level



$\underline{Gate_g}$

Input: Ciphertext of input wire values,
Authentication info of l, r.

Verification of Authentication
Decrypt input wires
Compute gate g
Encrypt output wire

Output: Ciphertext of output wire
Authentication info of o.

Technical Details

- δ-Equivalence from $\mathcal{EF}$-proofs
- δiO Construction
- **iO for Turing Machines**

Recall: Cook's Translation

Recall: Cook's Translation

Poly-size Proofs in $\mathcal{EF}$ $\xleftarrow{\text{Cook's Translation}}$ **Proofs in *PV***

Recall: Cook's Translation

Poly-size Proofs in $\mathcal{EF}$	$\xleftarrow{\text{Cook's Translation}}$	Proofs in *PV*

$$\vdash_{PV} M_1(x) = M_2(x)$$

Recall: Cook's Translation

Poly-size Proofs in $\mathcal{EF}$ $\xleftarrow{\text{Cook's Translation}}$ **Proofs in *PV***

Fix Input length n

$\longleftarrow$ $\vdash_{PV} M_1(x) = M_2(x)$

Recall: Cook's Translation

Poly-size Proofs in $\mathcal{EF}$ $\longleftarrow$ Cook's Translation $\longleftarrow$ **Proofs in *PV***

Fix Input length n

$$\vdash_{\mathcal{EF}} C_{M_1,n}(x) \leftrightarrow C_{M_2,n}(x) \longleftarrow \vdash_{PV} M_1(x) = M_2(x)$$

$C_{M_i,n}(x)$: Circuit that computes M_i for input $|x| \leq n$.

Recall: Cook's Translation

Poly-size Proofs in $\mathcal{EF}$ $\xleftarrow{\text{Cook's Translation}}$ **Proofs in *PV***

Fix Input length n

$$\vdash_{\mathcal{EF}} C_{M_1,n}(x) \leftrightarrow C_{M_2,n}(x) \longleftarrow \vdash_{PV} M_1(x) = M_2(x)$$

$C_{M_i,n}(x)$: Circuit that computes M_i for input $|x| \leq n$.

We know how to build iO for circuits
of poly-size $\mathcal{EF}$-proof of equivalence : δiO

iO for TMs from δiO

iO for TMs from δiO

M

Turing Machine

iO for TMs from δiO

$(N_0 = \lambda^{\log \lambda})$

M

Turing Machine $\xrightarrow[\;]{\text{All Input length } n \le N_0}$

iO for TMs from δiO

$(N_0 = \lambda^{\log \lambda})$

All Input length
$n \leq N_0$

M
Turing Machine $\xrightarrow{\hspace{2em}}$

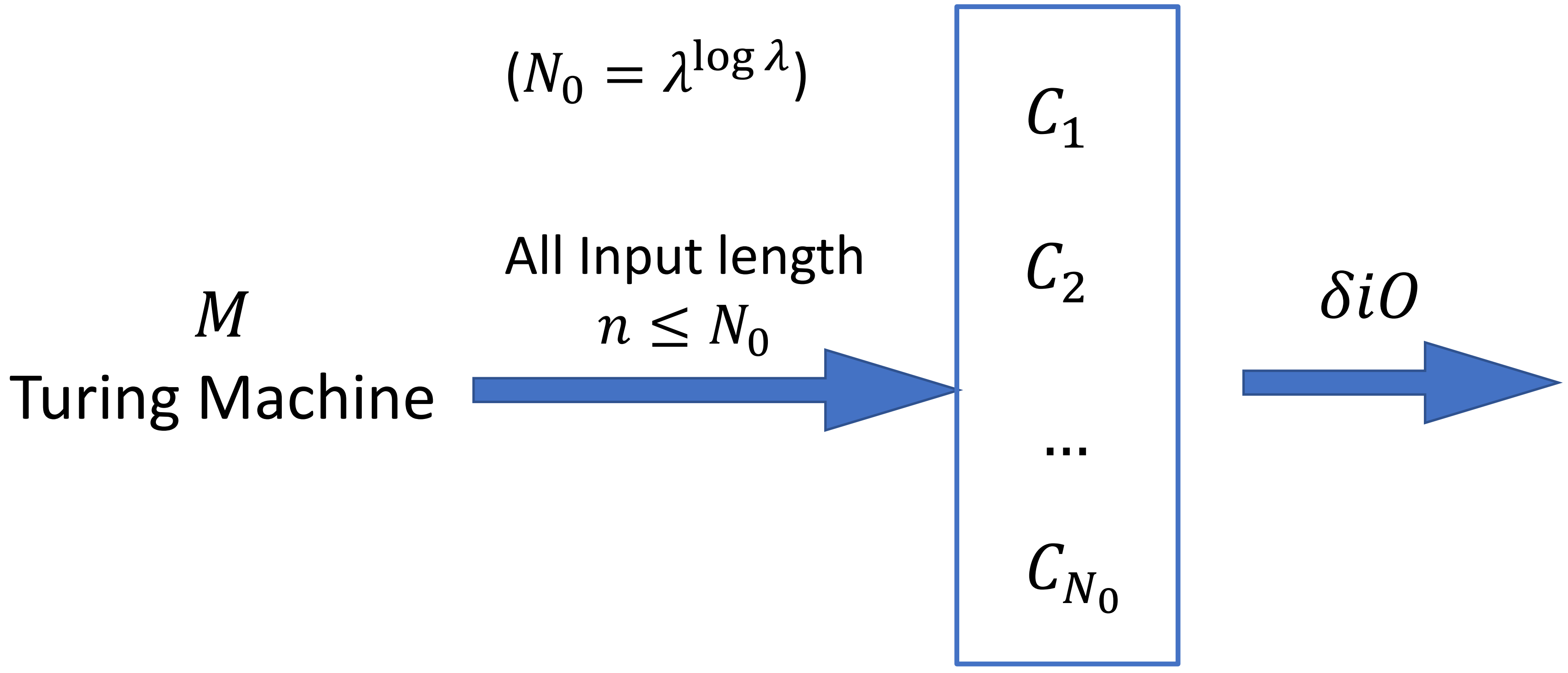
C_1

C_2

...

C_{N_0}

iO for TMs from δiO



iO for TMs from δiO

$(N_0 = \lambda^{\log \lambda})$

M

Turing Machine

All Input length

$n \leq N_0$

C_1

C_2

...

C_{N_0}

δiO

$\delta iO(C_1)$

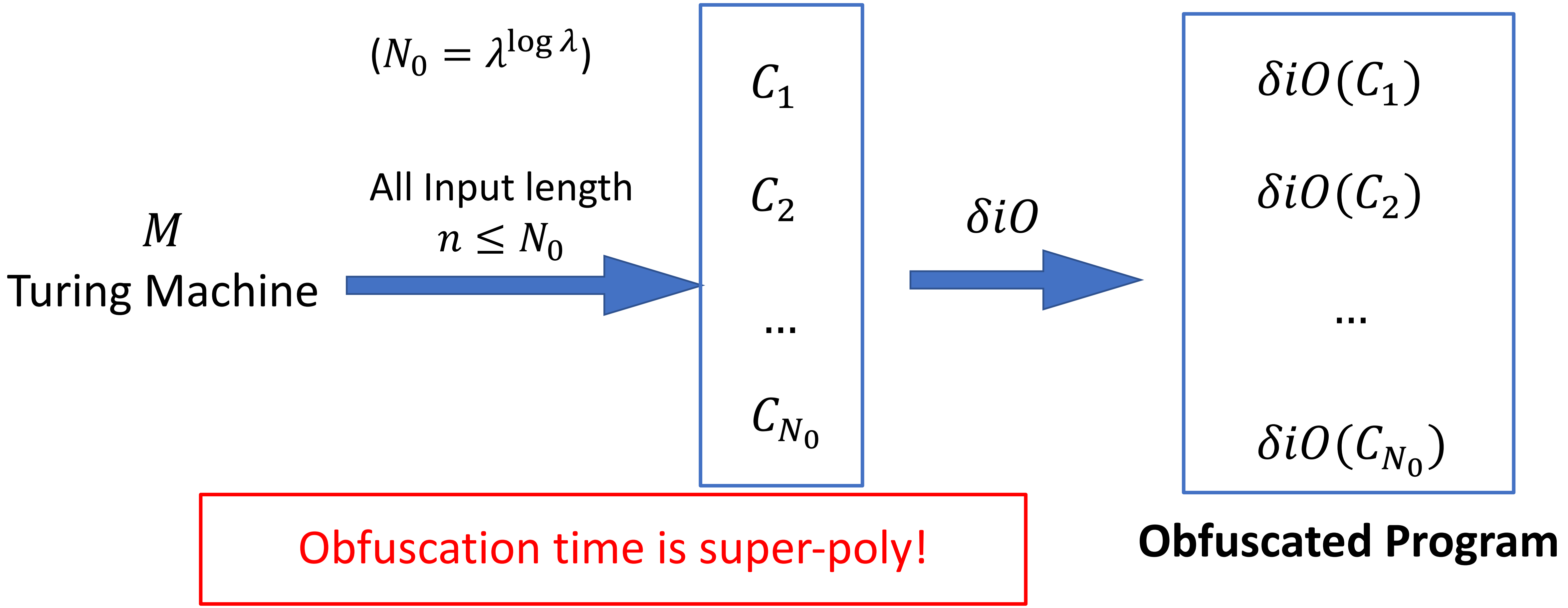
$\delta iO(C_2)$

...

$\delta iO(C_{N_0})$

Obfuscated Program

iO for TMs from δiO



Efficient Construction

Efficient Construction

$C_1, C_2, \dots, C_{N_0}$ have a succinct description

Efficient Construction

$C_1, C_2, \ldots, C_{N_0}$ have a succinct description

i.e. $\exists$ circuit $[M](\cdot,\cdot)$, s.t. $[M](n, i)$ outputs the description of i-th gate in C_n

Efficient Construction

$C_1, C_2, \ldots, C_{N_0}$ have a succinct description

i.e. $\exists$ circuit $[M](\cdot,\cdot)$, s.t. $[M](n, i)$ outputs the description of i-th gate in C_n

How do we generate $\delta iO\ \ C\ i$, given $[M](\cdot,\cdot)$?

, given $[M](\cdot,\cdot)$?

Efficient Construction

$C_1, C_2, \ldots, C_{N_0}$ have a succinct description

i.e. $\exists$ circuit $[M](\cdot,\cdot)$, s.t. $[M](n,i)$ outputs the description of i-th gate in C_n

How do we generate $\delta iO\ \ C\ i$, given $[M](\cdot,\cdot)$?

, given $[M](\cdot,\cdot)$?

Recall: δiO Construction

Gates $g_1, g_2, \ldots, g_{|C|}$ $\xrightarrow{\delta iO}$

$iO(Gate_{g_1})$

$iO(Gate_{g_2})$

…

$iO(Gate_{g_{|C|}})$

Generate $Gate_g$ from $[M](\cdot,\cdot)$

Efficient Construction

Efficient Construction

M

Turing Machine

Efficient Construction

M
Turing Machine

Efficient Construction

M
Turing Machine

$$\Rightarrow$$

$UGate(\cdot,\cdot,\cdot)$

Efficient Construction

(iO for small circuit)

M
Turing Machine $\Rightarrow$ $UGate(\cdot,\cdot,\cdot)$ $\Rightarrow$ $iO(UGate)$

Efficient Construction

(iO for small circuit)

M
Turing Machine → $UGate(\cdot,\cdot,\cdot)$ → $iO(UGate)$

"Uniform" Gate $UGate(n, i, input')$

Get description of i-th gate:

$$g \leftarrow [M](n, i)$$

Compute and output

$$Gate_g(input')$$

Future Directions

- iO for Turing machines with proof of equivalence in ZFC?

Future Directions

- iO for Turing machines with proof of equivalence in ZFC?

  iO for Turing machines for other logic systems, e.g., Buss's theory?

Future Directions

- iO for Turing machines with proof of equivalence in ZFC?

  iO for Turing machines for other logic systems, e.g., Buss's theory?

  Can we use other proof notions e.g. interactive proof systems?

Future Directions

- iO for Turing machines with proof of equivalence in ZFC?

  iO for Turing machines for other logic systems, e.g., Buss's theory?

  Can we use other proof notions e.g. interactive proof systems?

  Unprovability of cryptographic problems?