SNARGs Under LWE via Propositional Proofs

Zhengzhong Jin — MIT → Northeastern

Yael Tauman Kalai — Microsoft Research and MIT

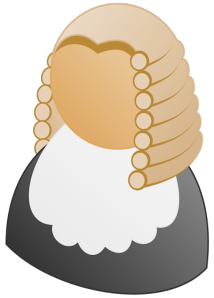
Alex Lombardi — Princeton

Vinod Vaikuntanathan — MIT

Succinct Non-Interactive Arguments (SNARGs)

CRS

CRS

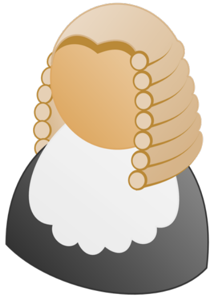


Succinct Non-Interactive Arguments (SNARGs)

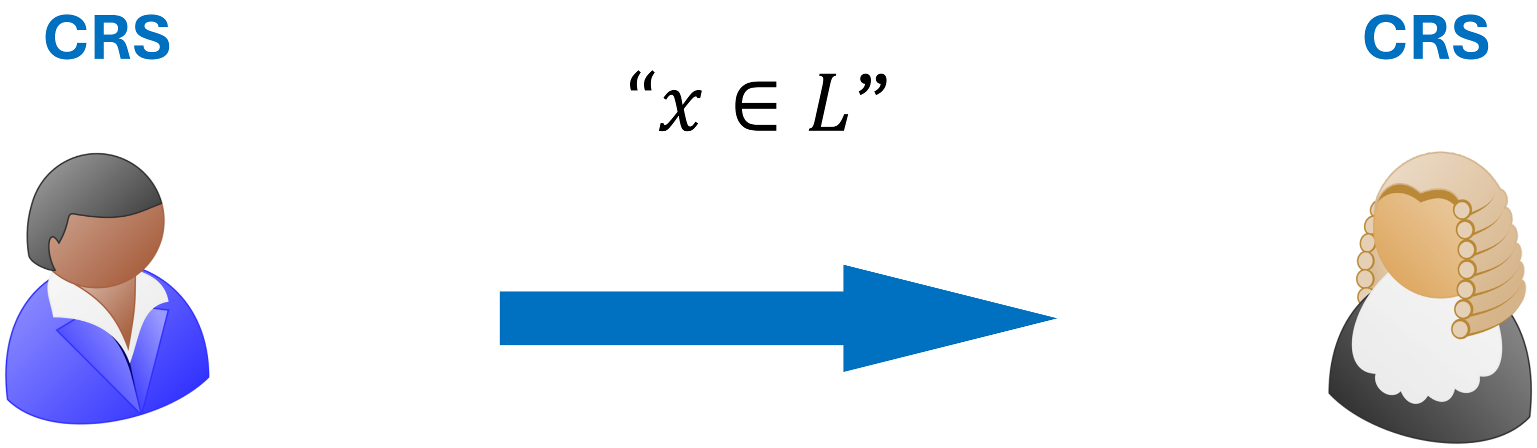
CRS

“$x \in L$”

CRS



Succinct Non-Interactive Arguments (SNARGs)



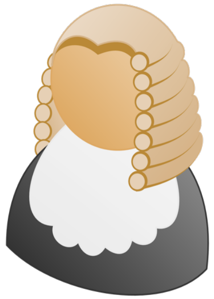
Succinct Non-Interactive Arguments (SNARGs)

CRS

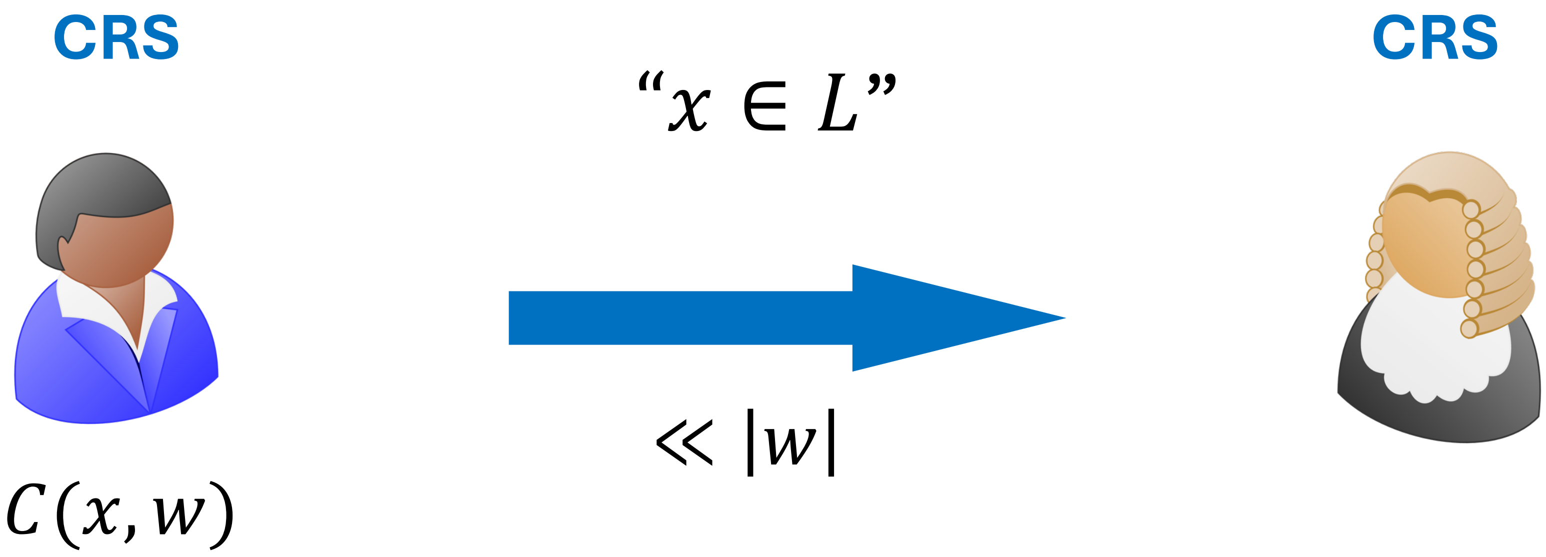
$C(x, w)$

“$x \in L$”

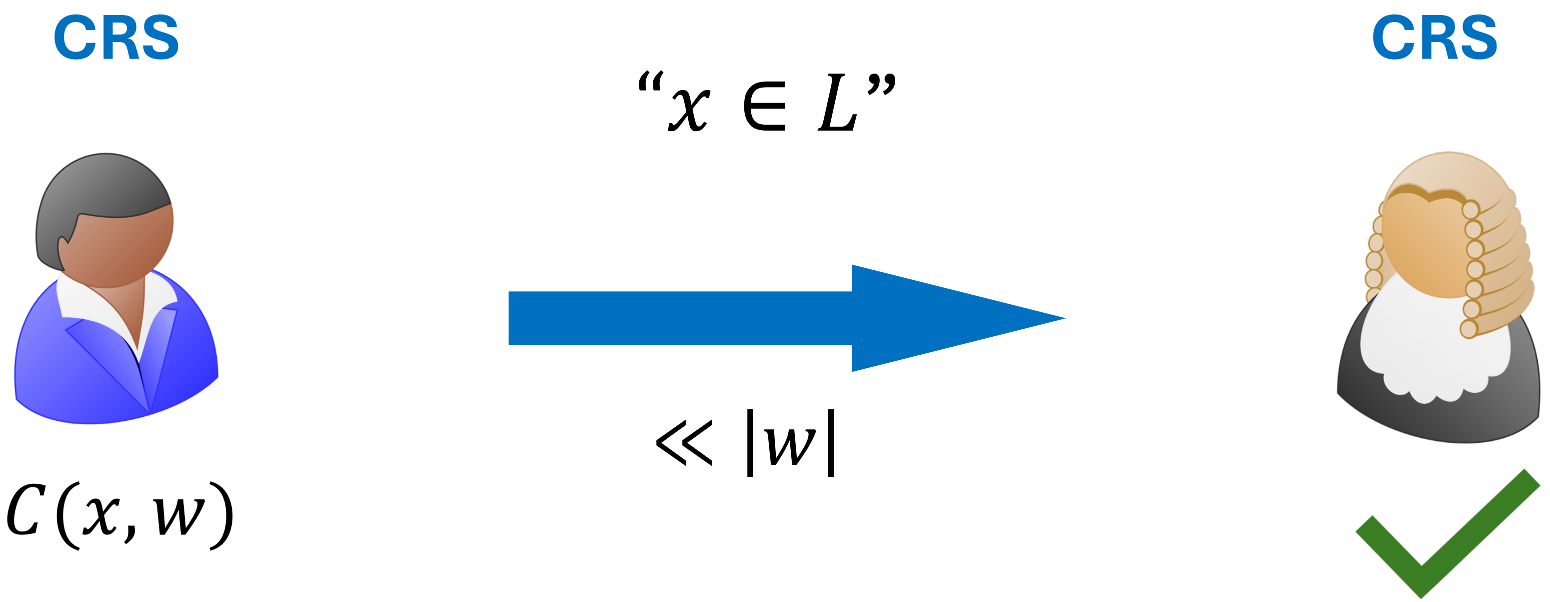
CRS



Succinct Non-Interactive Arguments (SNARGs)

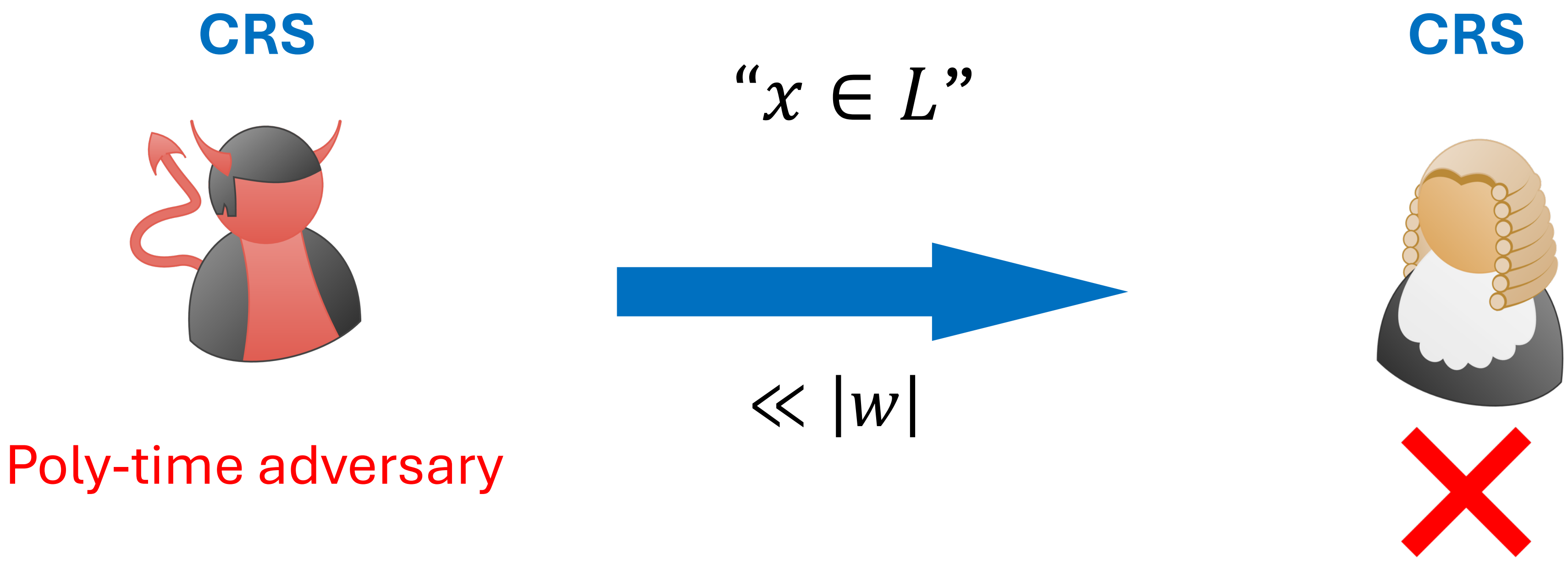


Succinct Non-Interactive Arguments (SNARGs)



- **Completeness:** $\forall x \in L$, the honestly generated proof is accepted.

Succinct Non-Interactive Arguments (SNARGs)



- **Completeness:** $\forall x \in L$, the honestly generated proof is accepted.
- **Soundness:** for any $x \notin L$, and any efficient adversary, the cheating proof should be rejected.

Can we build SNARGs for NP?

Can we build SNARGs for NP?

Micali’00, IKO’07, GKR’08, IKOS’09, Groth’10, SBW’11, SMBW’12, Lipmaa’12, CMT’12, DFH’12, SVPBBW’12, TRMP’12, GGPR’13, BCIOP’13, BCCT’13, Thaler’13, BCGTV’13, PHGR’13, BSCGT’13, BCGGMTV’14, BCCGP’16, Groth’16, GMO’16, GLRT’17, AHIV’17, BSBCGGHPRST’17, WJBSTWW’17, BBBPWM’18, BCGMMW’18, BSBHR’18, WTSTW’18, WZCPS’18, GMNO’18, FKL’18, BBCGI’19, BBHR’19, BCRSVW’19, BSCRSVW’19, CFQ19, GWC’19, KPV’19, KPY’19, MBKM’19, Nitulescu’19, XZZPS’19, Gabizon’19, BBS’20, BSCIKS’20, BFHVXZ’20, COS’20, CHMMVW’20, KZ’20, KPPS’20, SGKS’20, SL’20, Setty’20, ZXZS’20, BMMTV’21, GLSTW’21, GMN21, GPR’21, Sta’21, ZLWZSXZ’21, Bay’22, CBBZ’22, XZCZZJBS’22, XZS’22, …

Can we build SNARGs for NP?

Micali’00, IKO’07, GKR’08, IKOS’09, Groth’10, SBW’11, SMBW’12, Lipmaa’12, CMT’12, DFH’12, SVPBBW’12, TRMP’12, GGPR’13, BCIOP’13, BCCT’13, Thaler’13, BCGTV’13, PHGR’13, BSCGT’13, BCGGMTV’14, BCCGP’16, Groth’16, GMO’16, GLRT’17, AHIV’17, BSBCGGH[illegible] [illegible]CGMMW’18, BSBHR’18, WTSTW’18, W[illegible] [illegible]19, BCRSVW’19, BSCRSVW’19, CFQ19, GWC’19, KPV’19, KPY’19, MBKM’19, Nitulescu’19, XZZPS’19, Gabizon’19, BBS’20, BSCIKS’20, BFHVXZ’20, COS’20, CHMMVW’20, KZ’20, KPPS’20, SGKS’20, SL’20, Setty’20, ZXZS’20, BMMTV’21, GLSTW’21, GMN21, GPR’21, Sta’21, ZLWZSXZ’21, Bay’22, CBBZ’22, XZCZZJBS’22, XZS’22, …

Random Oracle Model, or Knowledge-type Assumptions

SNARGs for NP from **well-studied assumptions**?

SNARGs for NP from **well-studied assumptions**?

[Gentry-Wichs'12] Impossibility from falsifiable assumptions

(Caveat: require adaptive soundness and black-box soundness reduction)

SNARGs for NP from **well-studied assumptions?**

[Gentry-Wichs'12] Impossibility from falsifiable assumptions

For $L \in NP$ with "hardness" T (L needs time T to decide),
the proof size is unlikely $\ll \log T$.

(Caveat: require adaptive soundness and black-box soundness reduction)

SNARGs for NP from **well-studied assumptions?**

[Gentry-Wichs'12] Impossibility from falsifiable assumptions

For $L \in NP$ with "hardness" T (L needs time T to decide),
the proof size is unlikely $\ll \log T$.

For this Talk: A SNARG is "truly succinct" if proof size $\ll \log T$.
(SNARGs for NP must be truly succinct.)

(Caveat: require adaptive soundness and black-box soundness reduction)

SNARGs for NP from **well-studied assumptions**?

[Gentry-Wichs'12] Impossibility from falsifiable assumptions

For $L \in NP$ with "hardness" T (L needs time T to decide),
the proof size is unlikely $\ll \log T$.

Circumvent?

For this Talk: A SNARG is "truly succinct" if proof size $\ll \log T$.
(SNARGs for NP must be truly succinct.)

(Caveat: require adaptive soundness and black-box soundness reduction)

Prior Works Circumvent Gentry-Wichs

Prior Works Circumvent Gentry-Wichs

Not "Truly Succinct"

- Batch-NP [BHK17, CJJ21]
- P [KRR14, CJJ21, KVZ21]
- Bounded-space Non-deterministic comp. [KVZ21]
- Monotone policy Batch-NP [BBKLP23]

Assumption: standard (LWE, DDH, …)

Prior Works Circumvent Gentry-Wichs

Not "Truly Succinct"

- Batch-NP [BHK17, CJJ21]
- P [KRR14, CJJ21, KVZ21]
- Bounded-space Non-deterministic comp. [KVZ21]
- Monotone policy Batch-NP [BBKLP23]

Assumption: standard (LWE, DDH, …)

"Truly succinct"

- NP [SW14, WW24]
- Languages that have a "mathematical proof of non-membership" [JJ22]

Assumption: require indistinguishability obfuscation (iO)

Prior Works Circumvent Gentry-Wichs

Not "Truly Succinct"

- Batch-NP [BHK17, CJJ21]
- P [KRR14, CJJ21, KVZ21]
- Bounded-space Non-deterministic comp. [KVZ21]
- Monotone policy Batch-NP [BBKLP23]

Assumption: standard (LWE, DDH, …)

"Truly succinct"

- NP [SW14, WW24]
- Languages that have a "mathematical proof of non-membership" [JJ22]

Assumption: require indistinguishability obfuscation (iO)

We don't know ***any*** truly succinct SNARGs from standard assumptions!

Can we build **truly succinct** SNARGs from standard assumptions?

(Recall: truly succinct means proof size $\ll \log T$ for any NP languages that require time T to decide.)

Can we build **truly succinct** SNARGs from standard assumptions?

(Recall: truly succinct means proof size $\ll \log T$ for any NP languages that require time T to decide.)

(One step closer to SNARGs for NP from standard assumptions)

Our Results (I)

SNARGs from learning with errors (LWE) for NP languages that have a poly-size propositional proof of non-membership, with uniformly random CRS, where

- Proof size = poly(λ)
- CRS size = poly(prop. proof length, λ)

Construction doesn't need to use the propositional proof.

Our Results (I)

SNARGs from learning with errors (LWE) for NP languages that have a poly-size propositional proof of non-membership, with uniformly random CRS, where

- Proof size = poly(λ)
- CRS size = poly(prop. proof length, λ)

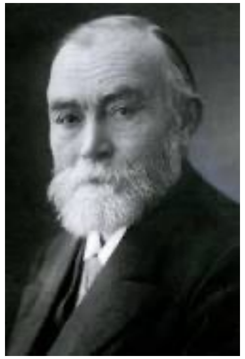
Construction doesn't need to use the propositional proof.

Propositional Proof of Non-Membership

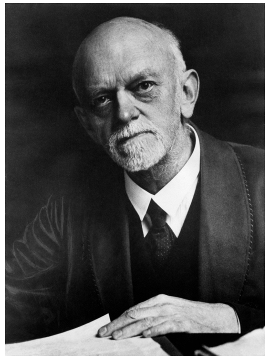
$\forall\, x \notin L$, there exists a poly-size propositional proof of

$$C(x,\cdot) = 0$$

Propositional Logic (Extended Frege)

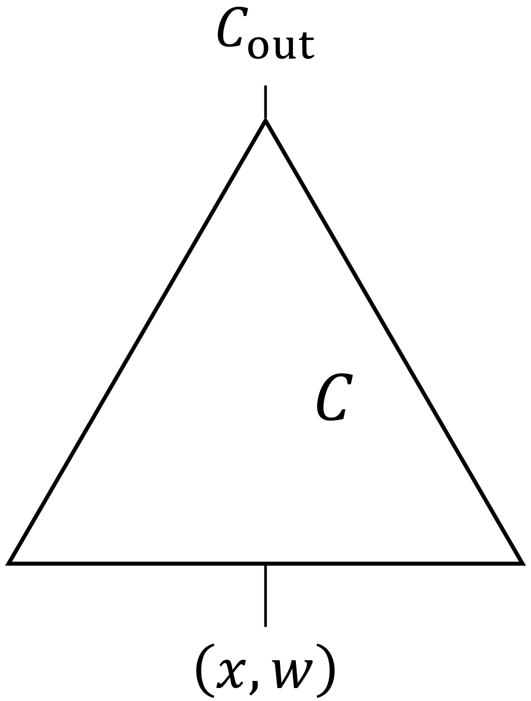


Frege



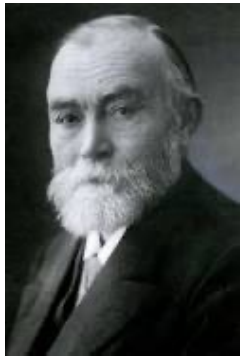
Hilbert

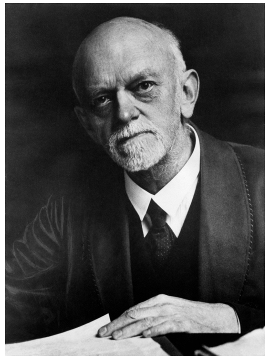
...



Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)


Frege


Hilbert

θ_1

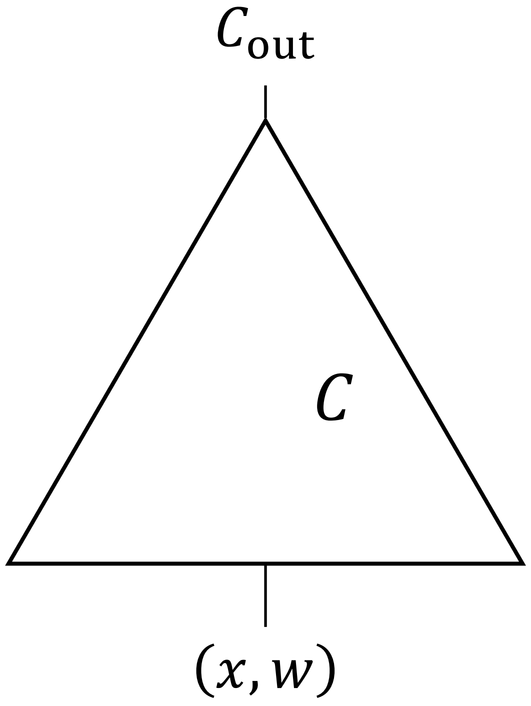
θ_2

…

θ_i

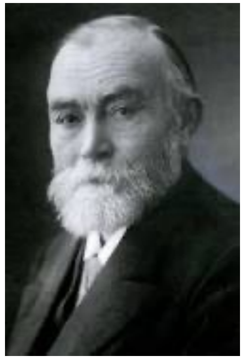
…

θ_T

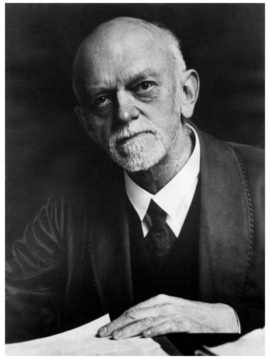


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

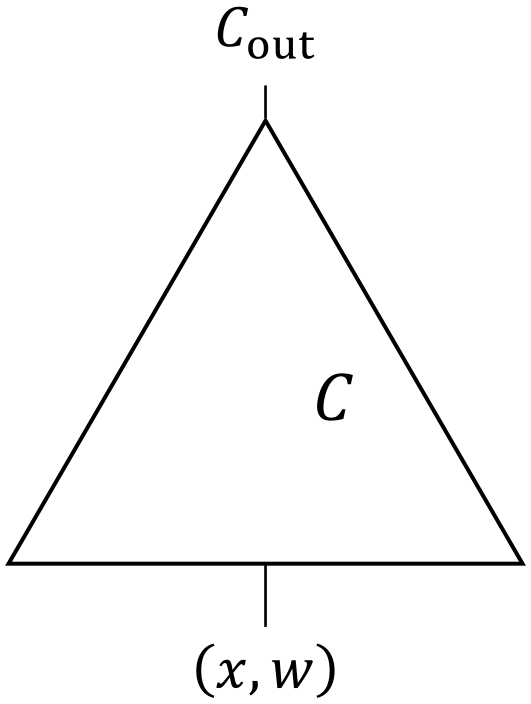
θ_2

...

θ_i

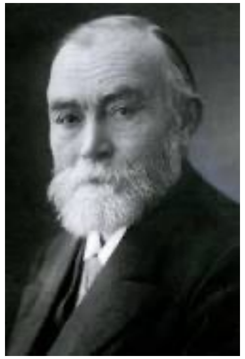
$\dots$

θ_T

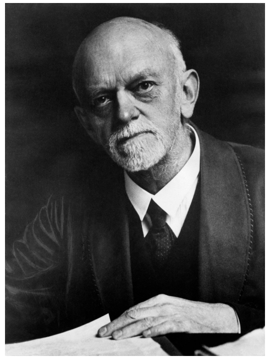


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

“lines” in the proof: boolean formulas

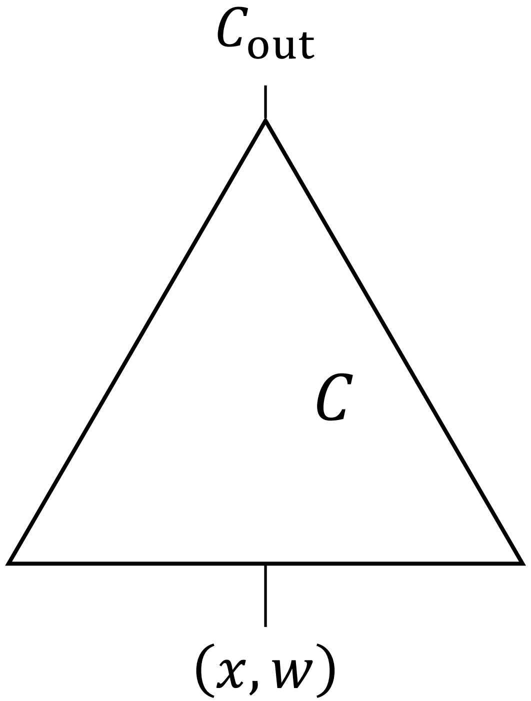
θ_2

…

θ_i

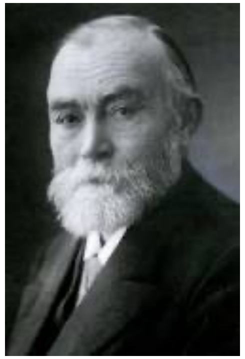
…

θ_T

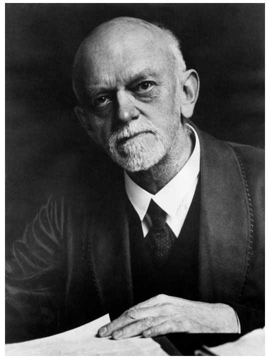


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

“lines” in the proof: boolean formulas

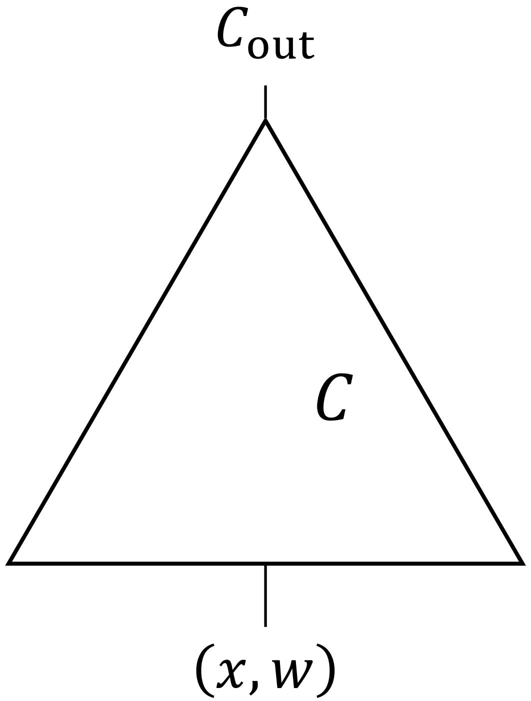
θ_2

...

θ_i

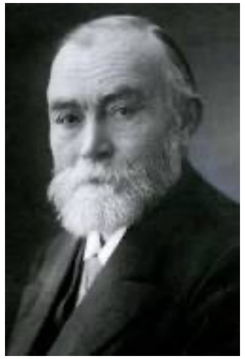
...

θ_T

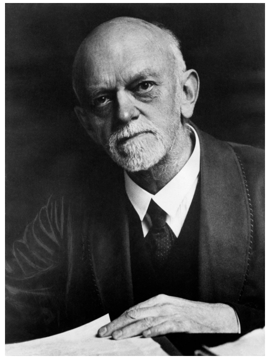


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

θ_2

...

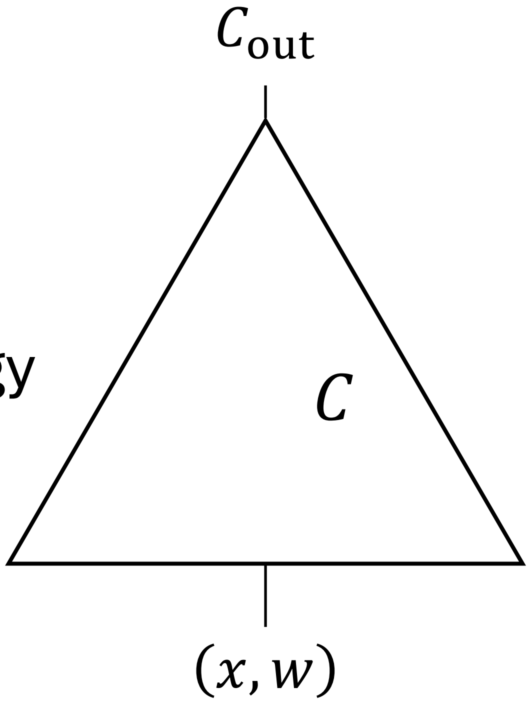
- Premise (i.e. gates in C)
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \rightarrow \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

θ_i

$$e \leftrightarrow \phi$$

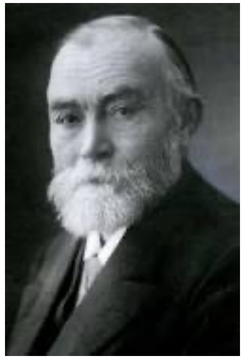
...

θ_T

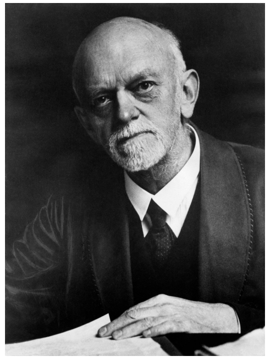


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

θ_2

...

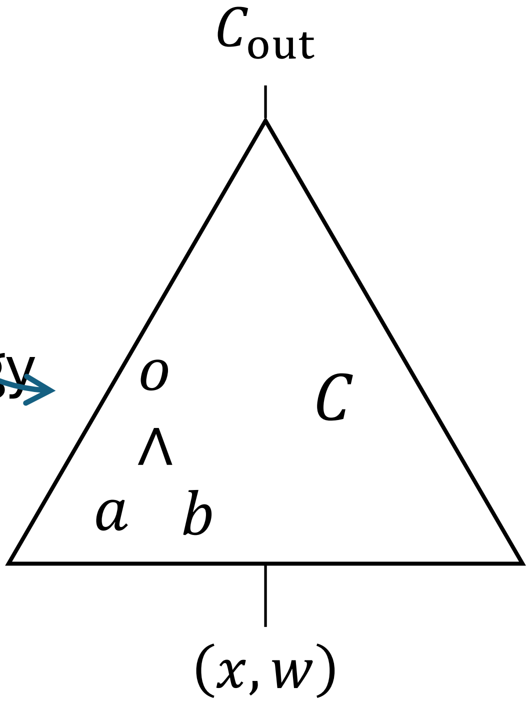
θ_i

- Premise (i.e. gates in C) e.g. "$o \leftrightarrow a \wedge b$"
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \to \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

  $e \leftrightarrow \phi$

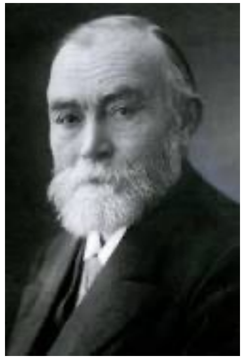
...

θ_T

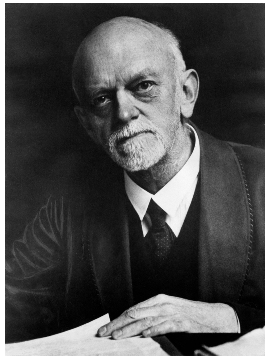


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

θ_2

…

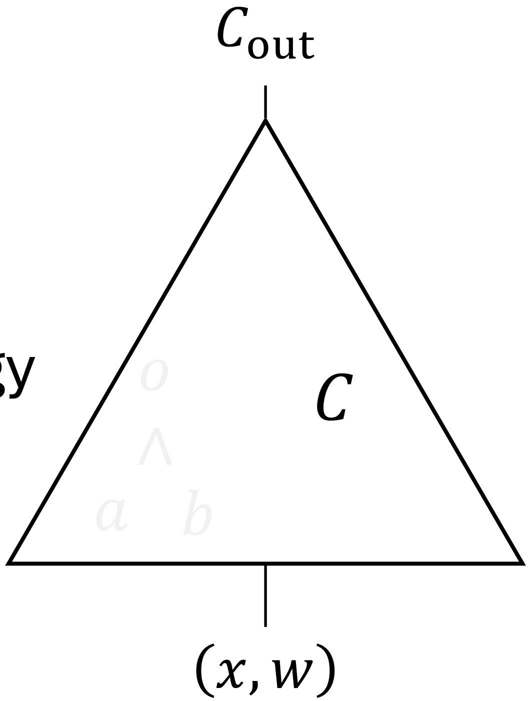
- Premise (i.e. gates in C)
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \to \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

  $e \leftrightarrow \phi$

θ_i

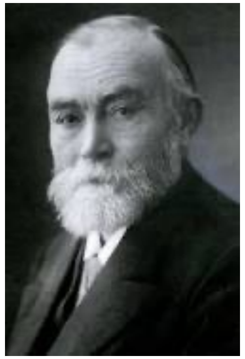
…

θ_T

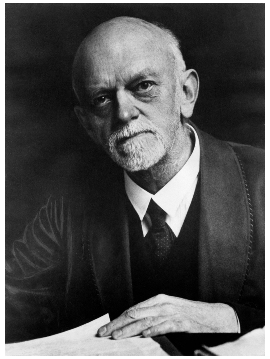


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

“lines” in the proof: boolean formulas

θ_2

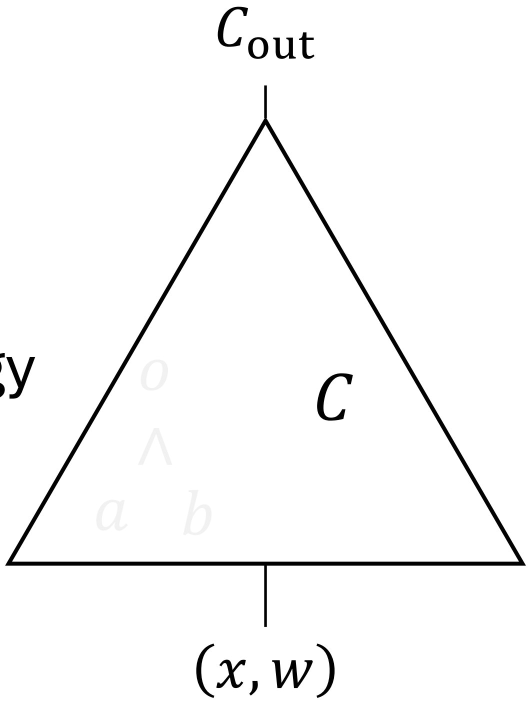
...

θ_i
- Premise (i.e. gates in C)
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \rightarrow \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. “e”

  $e \leftrightarrow \phi$

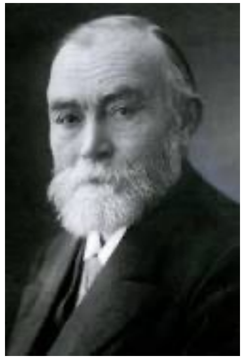
...

θ_T

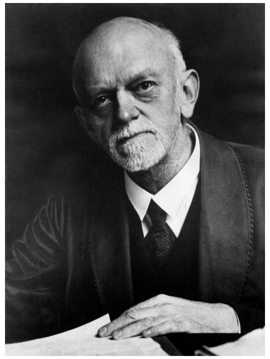


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

θ_2

$\theta_1{}'$

…

$\theta_c{}'$

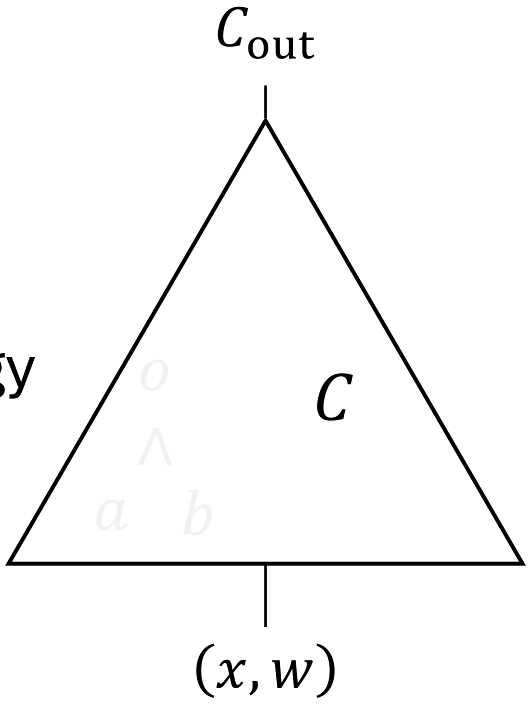
θ_i

- Premise (i.e. gates in C)
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \to \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

  $e \leftrightarrow \phi$

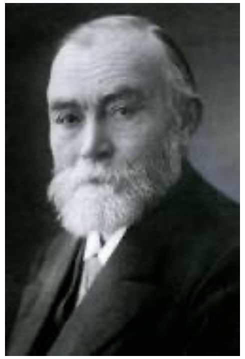
…

θ_T

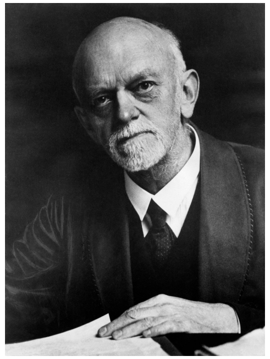


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

θ_2

...

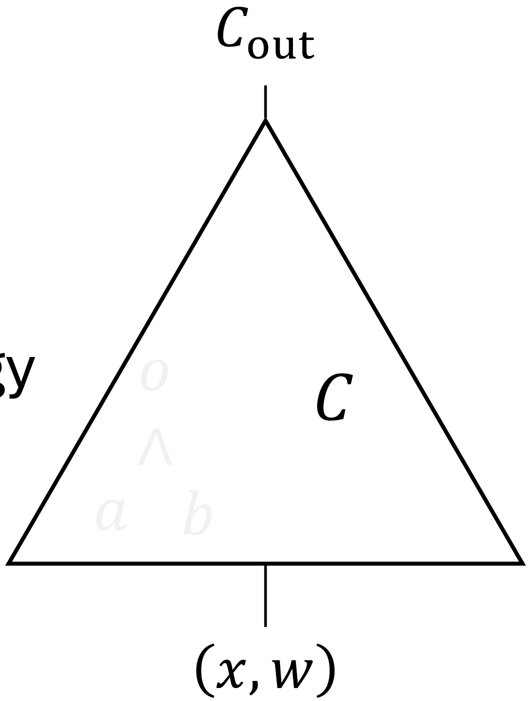
θ_i

- Premise (i.e. gates in C)
- Inference: $\theta'_1 \wedge \cdots \wedge \theta'_c \to \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

  $e \leftrightarrow \phi$

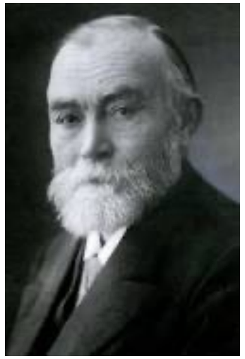
...

θ_T

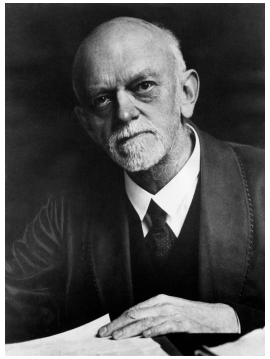


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

θ_2

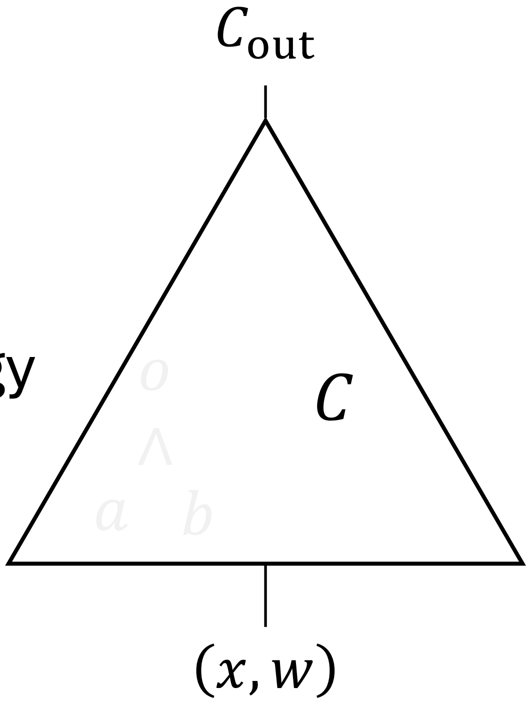
…

θ_i
- Premise (i.e. gates in C)
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \rightarrow \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

  $e \leftrightarrow \phi$

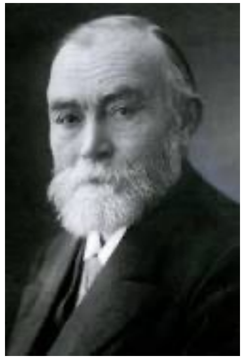
…

θ_T

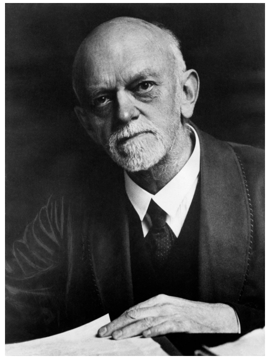


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

"lines" in the proof: boolean formulas

θ_2

...

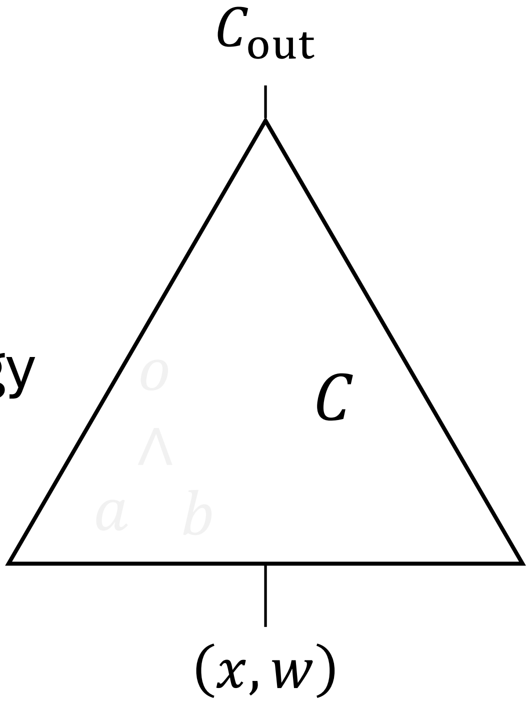
θ_i

- Premise (i.e. gates in C)
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \to \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

  $e \leftrightarrow \phi$

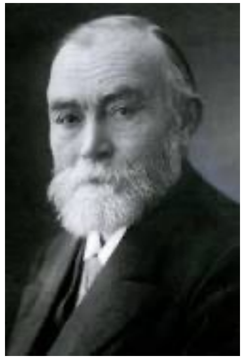
...

θ_T

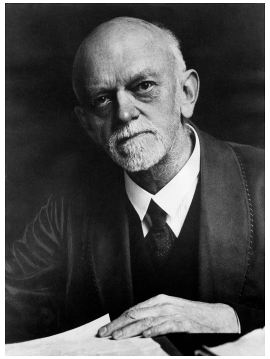


Propositional Logic (Extended Frege)

(Example: a non-membership proof of x)



Frege



Hilbert

θ_1

θ_2

"lines" in the proof: boolean formulas

…

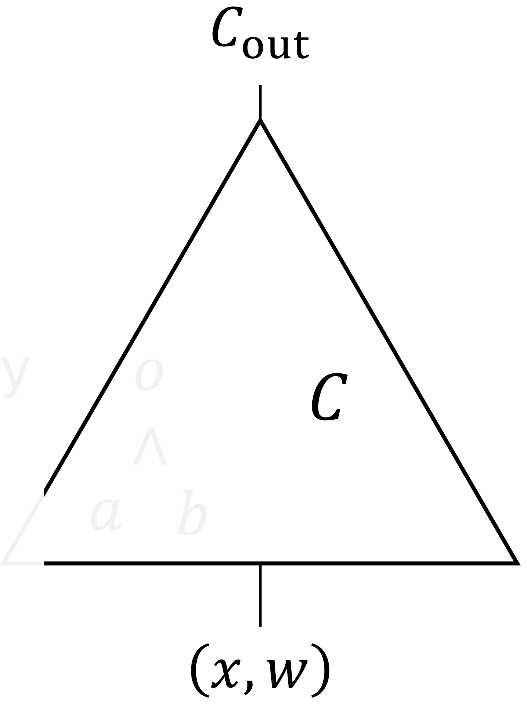
θ_i

- Premise (i.e. gates in C)
- Inference: $\theta_1' \wedge \cdots \wedge \theta_c' \rightarrow \theta_i$ is a $O(1)$-size tautology
- Extensions: define a new variable, e.g. "e"

  $e \leftrightarrow \phi$

…

θ_T : The theorem statement itself (i.e. "$C_{\text{out}} \leftrightarrow 0$")



Recall: Our Results (I)

SNARGs from learning with errors (LWE) for NP languages that

have a **poly-size** propositional proof of non-membership, where

- Proof size = poly(λ)
- CRS size = poly(prop. proof length, λ)

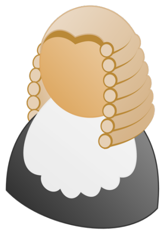
Recall: Our Results (I)

SNARGs from learning with errors (LWE) for NP languages that

have a **poly-size** propositional proof of non-membership, where

- Proof size = poly(λ)
- CRS size = poly(prop. proof length, λ)

Why should I care about propositional proofs?



(a cryptographer)

Context: **Duality** between Logic and Computation

Context: **Duality** between Logic and Computation

Computation

Context: **Duality** between Logic and Computation

Logic

Computation

Context: **Duality** between Logic and Computation

Logic

Computation

Formulas

Context: **Duality** between Logic and Computation

Logic	Computation
Frege	Formulas

Context: **Duality** between Logic and Computation

Logic

Frege

Computation

Formulas

Circuits

Context: **Duality** between Logic and Computation

Logic	Computation
Frege	Formulas
Extended Frege	Circuits

Context: **Duality** between Logic and Computation

Logic	Computation
Frege	Formulas
Extended Frege	Circuits
	Poly-size Circuits

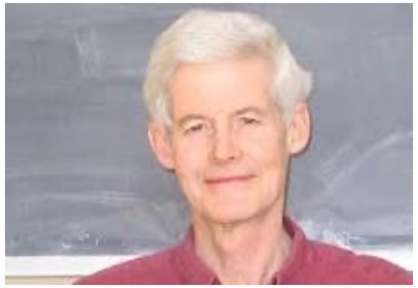
Context: **Duality** between Logic and Computation

Logic	Computation
Frege	Formulas
Extended Frege	Circuits
Poly-size Extended Frege	Poly-size Circuits

Context: **Duality** between Logic and Computation

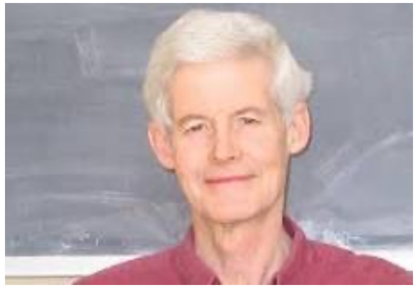
Logic	Computation
Frege	Formulas
Extended Frege	Circuits
Poly-size Extended Frege	Poly-size Circuits
	Poly-time Turing Machines

Context: **Duality** between Logic and Computation

Logic	Computation
Frege	Formulas
Extended Frege	Circuits
Poly-size Extended Frege	Poly-size Circuits
 Cook's Theory PV [1975]	Poly-time Turing Machines

Context: **Duality** between Logic and Computation

Logic	Computation
Frege	Formulas
Extended Frege	Circuits
Poly-size Extended Frege	Poly-size Circuits
Cook’s Theory PV [1975]	Poly-time Turing Machines
…	…



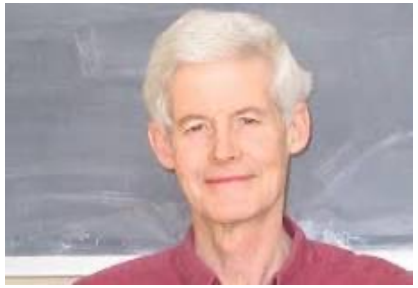
Context: **Duality** between Logic and Computation

Logic

Frege

Extended Frege

Poly-size Extended Frege



Cook's Theory PV [1975]

...

Computation

Formulas

Circuits

Computational resources (time/space) used to decide the language?

Poly-size Circuits

(Computational complexity)

Poly-time Turing Machines

...

Context: **Duality** between Logic and Computation

Logic

Frege

Extended Frege

Resources (sizes) of proofs used to prove co-NP statements?
(non-membership)

Poly-size Extended Frege

Proof complexity

Cook's Theory PV [1975]

...

Computation

Formulas

Circuits

Computational resources (time/space) used to decide the language?

Poly-size Circuits

(Computational complexity)

Poly-time Turing Machines

...

Context: **Duality** between Logic and Computation

Logic

Frege

Extended Frege

Resources (sizes) of proofs used to prove co-NP statements?
(non-membership)

Poly-size Extended Frege

Proof complexity

Cook's Theory PV [1975]

Example languages?

...

Computation

Formulas

Circuits

Computational resources (time/space) used to decide the language?

Poly-size Circuits

(Computational complexity)

Poly-time Turing Machines

...

Corollary (via Cook's Theory PV)

SNARGs from LWE for decisional Diffie-Hellman language

$L = \{(g, g^a, g^b, g^{ab})\}$ over n-bits standard group.

- Proof size = poly(λ), independent of n!
- CRS size = poly(λ, n)

Corollary (via Cook's Theory PV)

SNARGs from LWE for decisional Diffie-Hellman language

$L = \{(g, g^a, g^b, g^{ab})\}$ over n-bits standard group.

- Proof size = poly(λ), independent of n!
- CRS size = poly(λ, n)

Fully Succinct: L requires $2^{\Omega(n^{1/3})}$ time to decide, but SNARG proof size $\ll n$.

Our Results (II)

SNARGs from T-hardness of (LWE) for NP languages that have a propositional proof of non-membership of space S and (possibly super-poly) length T, where

- |Proof| = poly($\log T$, λ)
- |CRS| = poly($S, \log T, \lambda$)

Our Results (II)

SNARGs from T-hardness of (LWE) for NP languages that have a propositional proof of non-membership of space S and (possibly super-poly) length T, where

- |Proof| = poly($\log T,\ \lambda$)
- |CRS| = poly($S, \log T, \lambda$)

"Dual" of SNARGs for bounded space non-deterministic computation NTISP(S, T)
(Proof and CRS size = poly(S, $\log T$))

Main Challenge

Recall: Indistinguishability Obfuscation (iO)

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

$i\mathcal{O}$ →

```javascript
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();
```

Recall: Indistinguishability Obfuscation (iO)

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

iO

```
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();
```

- **Preserve Functionality:** $iO(C)$ preserves the functionality of C

Recall: Indistinguishability Obfuscation (iO)

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

iO

```
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();
```

- **Preserve Functionality:** $iO(C)$ preserves the functionality of C
- **Indistinguishability Security:** for any C_0, C_1 that compute the same function,

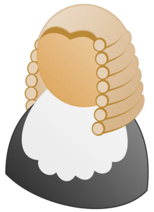
$$iO(C_0) \approx_c iO(C_1)$$

Starting Point: SW-SNARGs from iO

CRS

$$\widetilde{PK} = iO\left(\begin{array}{l} PK(x,w): \\ \quad \text{if } C(x,w) = 1: \\ \quad\quad \text{output } PRF_k(x) \end{array}\right)$$

$$iO\left(\begin{array}{l} VK(x,\sigma): \\ \quad \sigma \stackrel{?}{=} PRF_k(x) \end{array}\right) = \widetilde{VK}$$



Starting Point: SW-SNARGs from iO

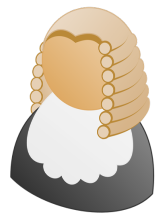
CRS

$$\widetilde{PK} = iO\left(\begin{array}{l} PK(x,w): \\ \quad \text{if } C(x,w) = 1: \\ \quad\quad \text{output } PRF_k(x) \end{array} \right)$$

$$iO\left(\begin{array}{l} VK(x,\sigma): \\ \quad \sigma \stackrel{?}{=} PRF_k(x) \end{array} \right) = \widetilde{VK}$$

(x, w)

σ



Starting Point: SW-SNARGs from iO

CRS

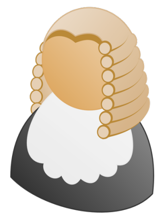
$$\widetilde{PK} = iO\left(\begin{array}{l} PK(x,w): \\ \quad \text{if } C(x,w) = 1: \\ \qquad \text{output } PRF_k(x) \end{array}\right)$$

$$iO\left(\begin{array}{l} VK(x,\sigma): \\ \quad \sigma \overset{?}{=} PRF_k(x) \end{array}\right) = \widetilde{VK}$$

(x, w)

σ

σ



Starting Point: SW-SNARGs from iO

CRS

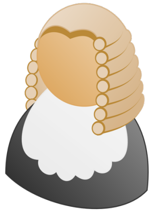
$\widetilde{PK} = iO\left(\begin{array}{l} PK(x,w): \\ \quad \text{if } C(x,w) = 1: \\ \quad\quad \text{output } PRF_k(x) \end{array}\right)$

$iO\left(\begin{array}{l} VK(x,\sigma): \\ \quad \sigma \stackrel{?}{=} PRF_k(x) \end{array}\right) = \widetilde{VK}$

(x, w)

σ

σ



Intuition for Soundness: $\forall\, x^* \notin L$, $\widetilde{PK}$ never outputs $PRF_k(x^*)$

$\Rightarrow$ iO hides $PRF_k(x^*)$

Starting Point: SW-SNARGs from iO

CRS

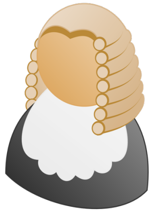
$\widetilde{PK} = iO\left(\begin{array}{l} PK(x,w): \\ \text{If } R(x,w)=1 \\ \quad \text{output } PRF_k(x) \end{array}\right)$ $\quad$ $iO\left(\begin{array}{l} VK(x,\sigma): \\ \sigma = PRF_k(x) \end{array}\right) = \widetilde{VK}$

Can we instantiate this template from LWE?

(x, w)

σ

σ

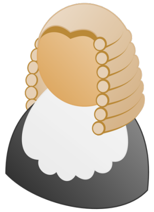


Intuition for Soundness: $\forall\, x^* \notin L$, $\widetilde{PK}$ never outputs $PRF_k(x^*)$

$\Rightarrow\ iO$ hides $PRF_k(x^*)$

Towards SNARGs from LWE: Replace iO with FHE

(FHE: fully homomorphic encryption)

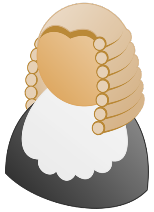


Towards SNARGs from LWE: Replace iO with FHE

(FHE: fully homomorphic encryption)

CRS

$$\mathrm{FHE}\left(\, C(x,\cdot) \,\right)$$



sk

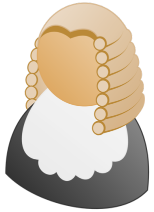
Towards SNARGs from LWE: Replace iO with FHE

(FHE: fully homomorphic encryption)

CRS

$$\mathrm{FHE}\left[\; C(x,\cdot) \;\right]$$

Eval. on w



sk

Towards SNARGs from LWE: Replace iO with FHE

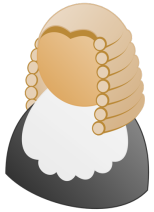
(FHE: fully homomorphic encryption)

CRS

$$\mathrm{FHE}\left[\, C(x,\cdot) \,\right]$$

Eval. on w

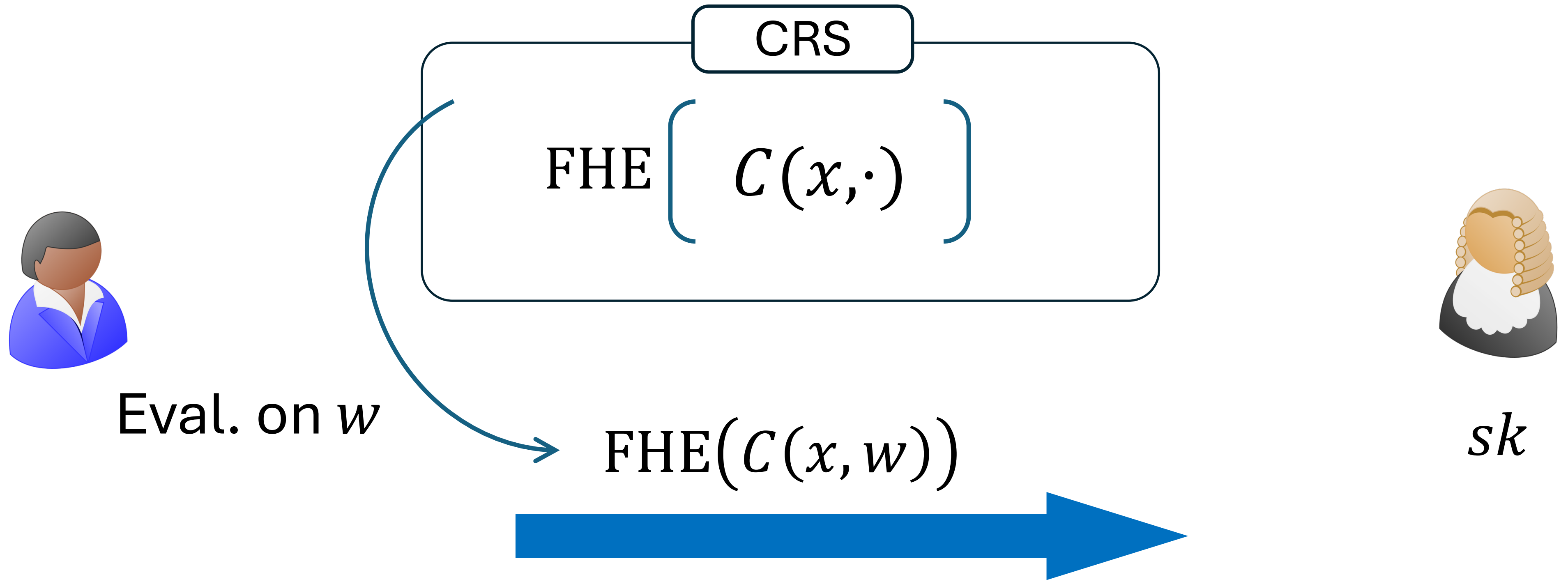
$$\mathrm{FHE}\big(C(x,w)\big)$$



sk

Towards SNARGs from LWE: Replace iO with FHE

(FHE: fully homomorphic encryption)



Towards SNARGs from LWE: Replace iO with FHE

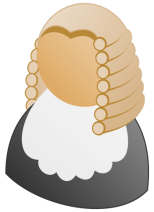
(FHE: fully homomorphic encryption)

CRS

$\mathrm{FHE}\left[\ C(x,\cdot)\ \right]$

Eval. on w

$\mathrm{FHE}\big(C(x,w)\big)$



sk

Dec and check whether it's 1

Towards SNARGs from LWE: Replace iO with FHE

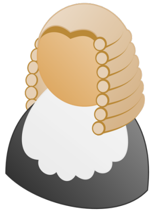
(FHE: fully homomorphic encryption)

CRS

$\mathrm{FHE}\left(C(x,\cdot) \right)$

Eval. on w

$\mathrm{FHE}\big(C(x,w)\big)$ π



sk

Dec and check whether it's 1

Towards SNARGs from LWE: Replace iO with FHE

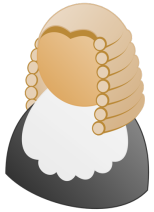
(FHE: fully homomorphic encryption)

CRS

$\mathrm{FHE}\left(C(x,\cdot) \right)$

Eval. on w

$\mathrm{FHE}\big(C(x,w)\big)$ π



sk

Dec and check whether it's 1

π proves "$\mathrm{FHE}\big(C(x,w)\big)$ is computed correctly from some w."

Towards SNARGs from LWE: Replace iO with FHE

(FHE: fully homomorphic encryption)

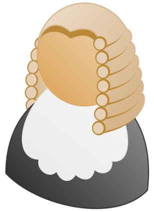
CRS

$\mathrm{FHE}\big(C(x,\cdot)\big)$

Eval. on w

We need a SNARG!

$\mathrm{FHE}\big(C(x,w)\big)$ π



sk

Dec and check whether it's 1

π proves "$\mathrm{FHE}\big(C(x,w)\big)$ is computed correctly from some w."

Towards SNARGs from LWE: Replace iO with FHE

(FHE: fully homomorphic encryption)

- CRS depends on x
- Designated Verifier

(will solve later)

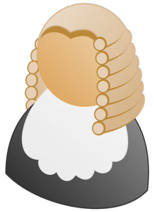
CRS

$\mathrm{FHE}\left(C(x,\cdot) \right)$

Eval. on w

$\mathrm{FHE}\big(C(x,w)\big)$ π

We need a SNARG!

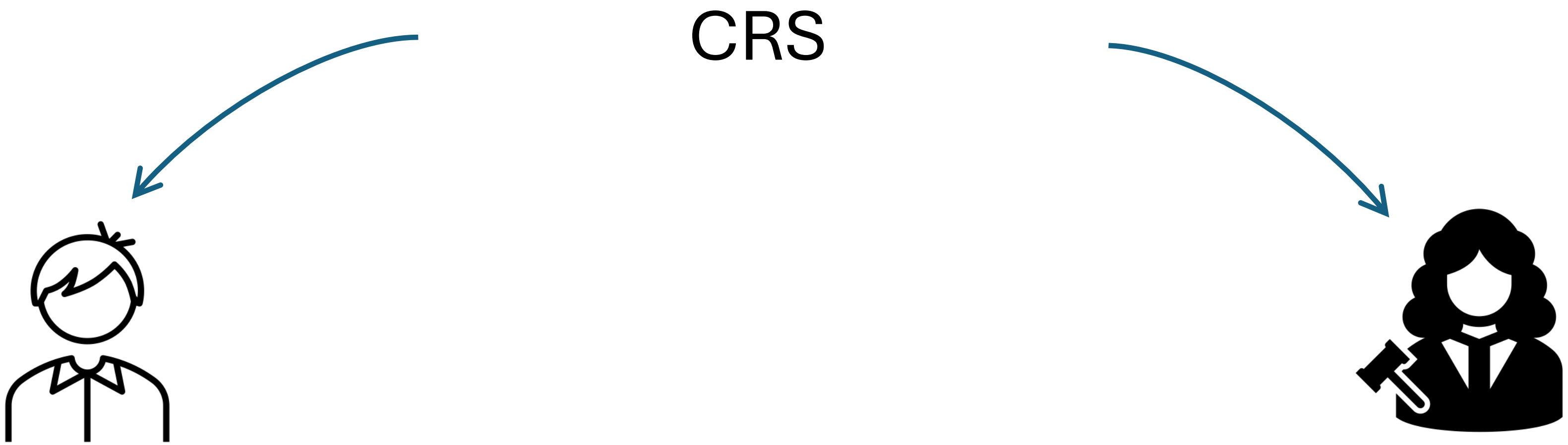


sk

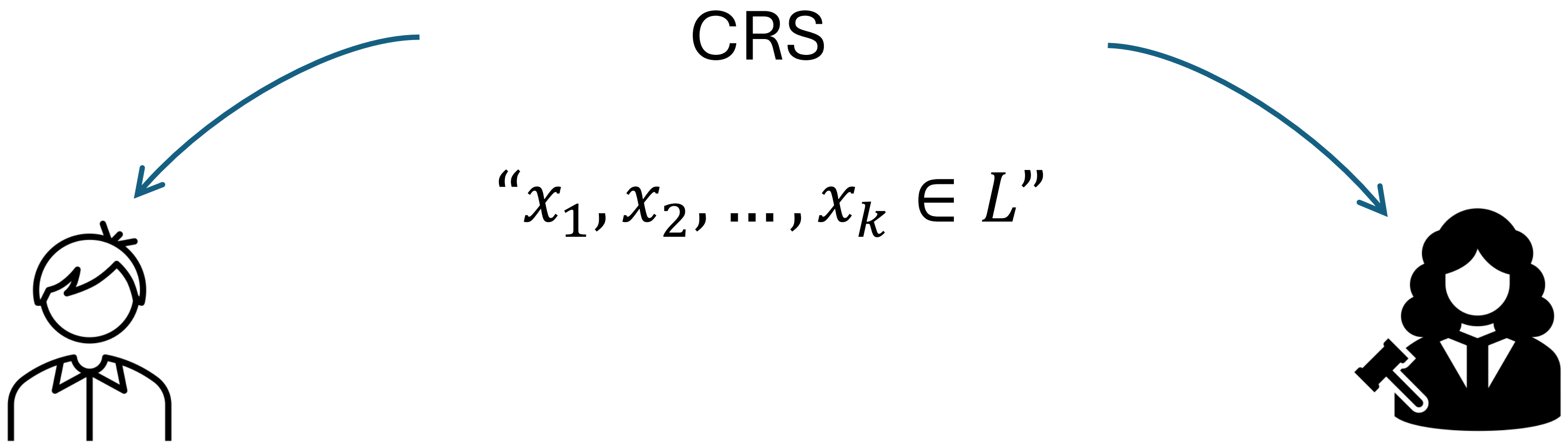
Dec and check whether it's 1

π proves "$\mathrm{FHE}\big(C(x,w)\big)$ is computed correctly from some w."

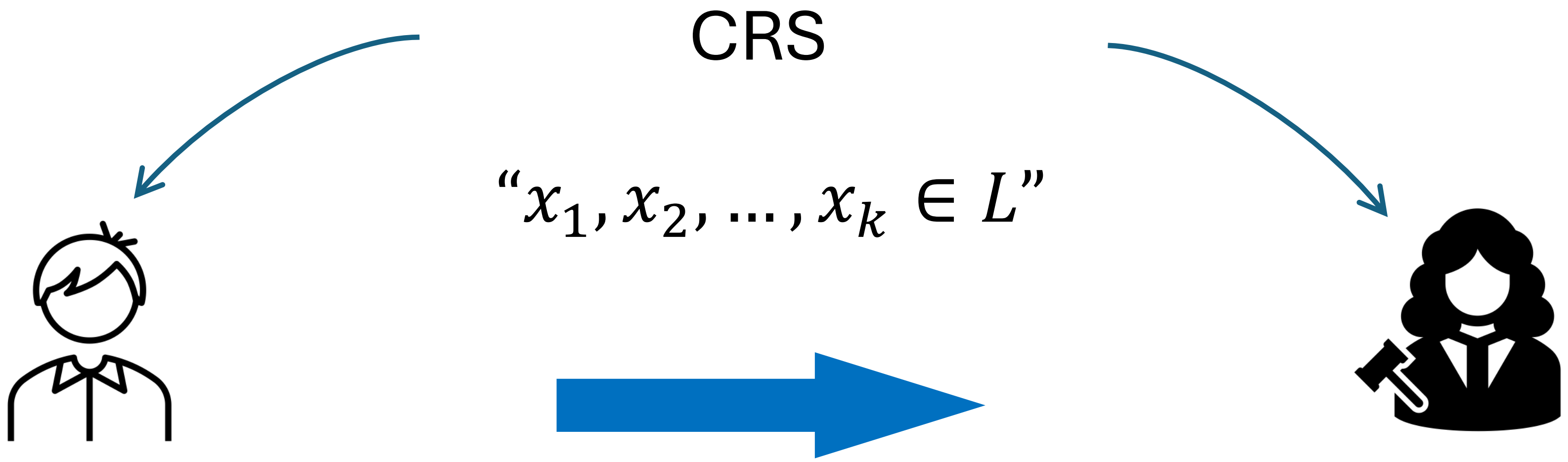
Recall: BARGs (Batch Arguments)[BHK17,CJJ21]



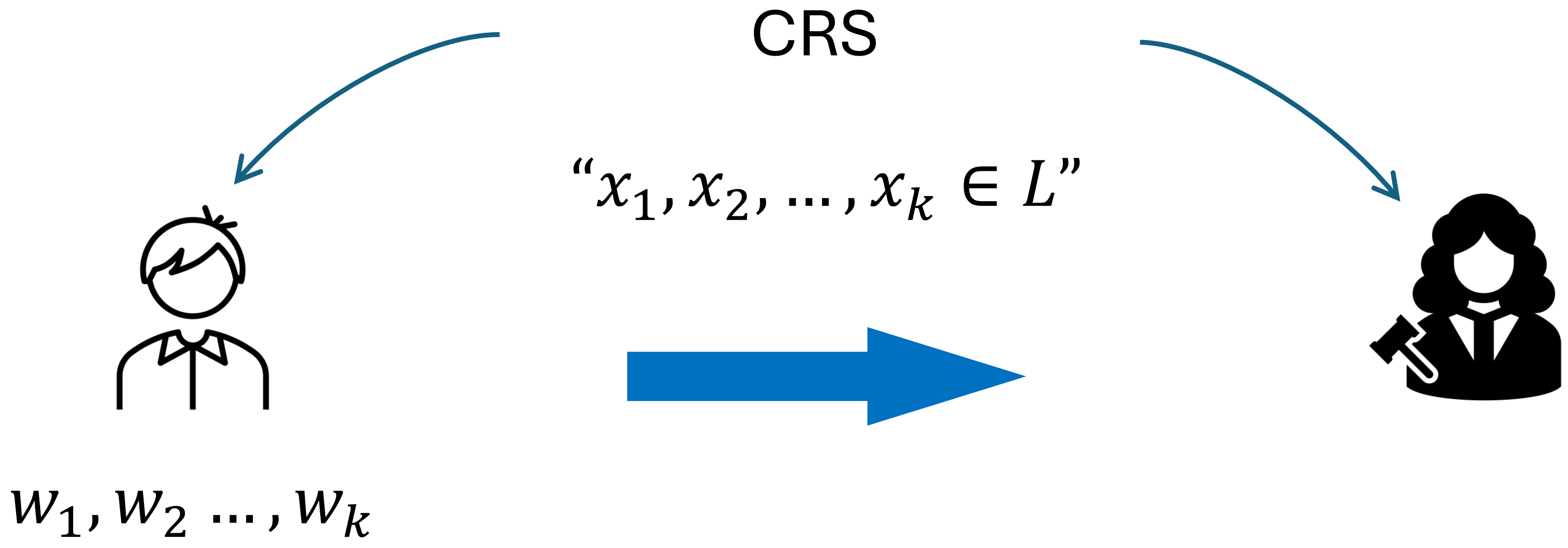
Recall: BARGs (Batch Arguments)[BHK17,CJJ21]



Recall: BARGs (Batch Arguments)[BHK17,CJJ21]



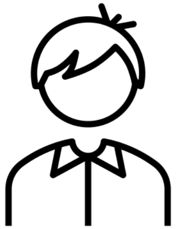
Recall: BARGs (Batch Arguments)[BHK17,CJJ21]



Recall: BARGs (Batch Arguments)[BHK17,CJJ21]

CRS

“$x_1, x_2, \ldots, x_k \in L$”

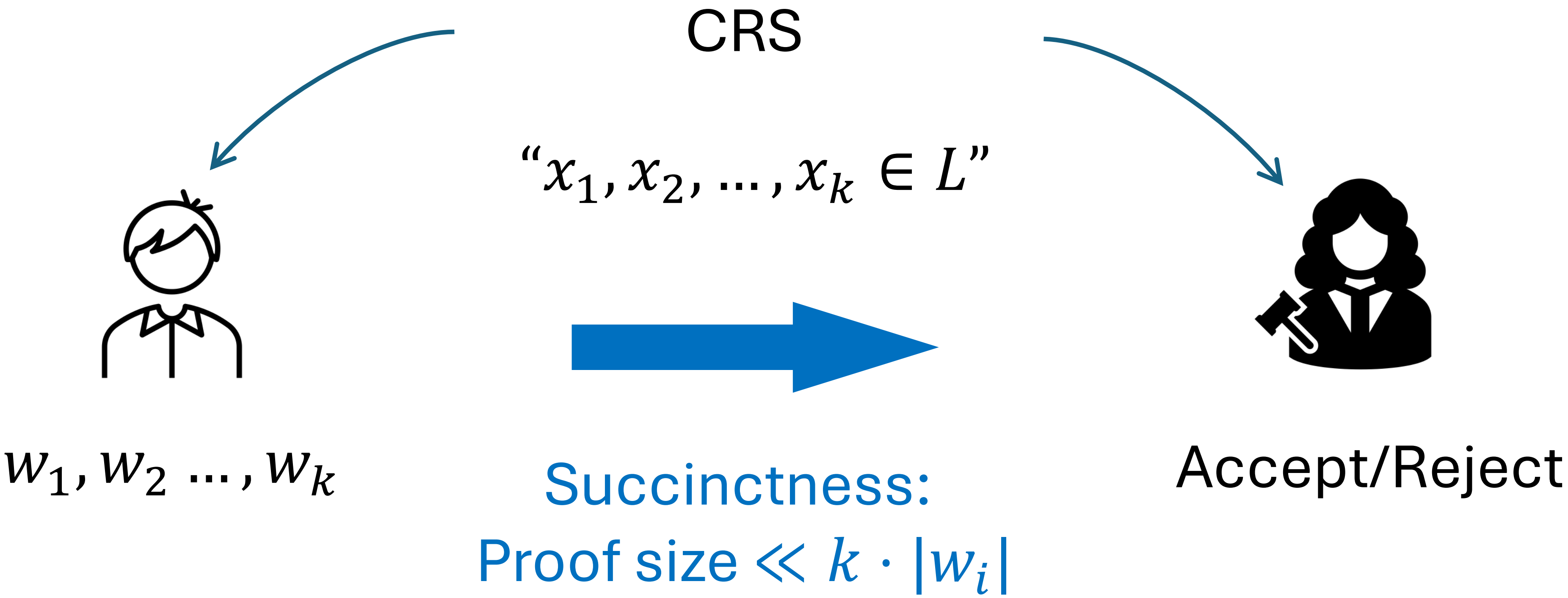


$w_1, w_2 \ldots, w_k$

Succinctness:

Proof size $\ll k \cdot |w_i|$

Recall: BARGs (Batch Arguments)[BHK17,CJJ21]



Recall: BARGs (Batch Arguments)[BHK17,CJJ21]

$\mathsf{CRS}(S) \approx_c U$

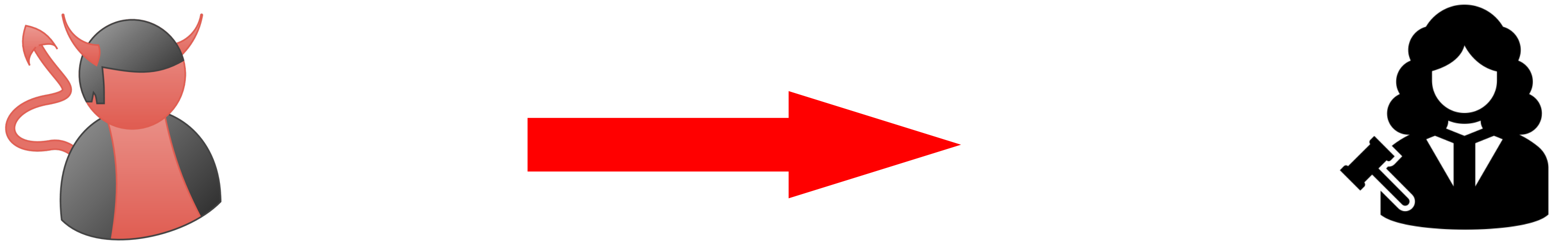
S: a **small** subset of $[k]$

Unbounded

Recall: BARGs (Batch Arguments)[BHK17,CJJ21]

$\mathrm{CRS}(S) \approx_c U$

S: a **small** subset of $[k]$



Unbounded

Recall: BARGs (Batch Arguments)[BHK17,CJJ21]

$\mathrm{CRS}(S) \approx_c U$

S: a **small** subset of $[k]$

Unbounded

Proof size $\approx |S| \cdot |w_i|$

Recall: BARGs (Batch Arguments)[BHK17,CJJ21]

$\text{CRS}(S) \approx_c U$

S: a **small** subset of $[k]$

Proof size $\approx |S| \cdot |w_i|$

Unbounded

Somewhere Statistical Soundness

If one of the instances in S is false, then any **unbounded-time** computed cheating proof should be rejected.

Recall: BARGs (Batch Arguments)[BHK17,CJJ21]

$\text{CRS}(S) \approx_c U$

S: a **small** subset of $[k]$

Unbounded

Proof size $\approx |S| \cdot |w_i|$

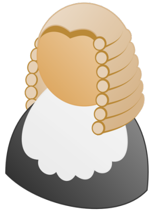
Somewhere Statistical Soundness

If one of the instances in S is false, then any **unbounded-time** computed cheating proof should be rejected.

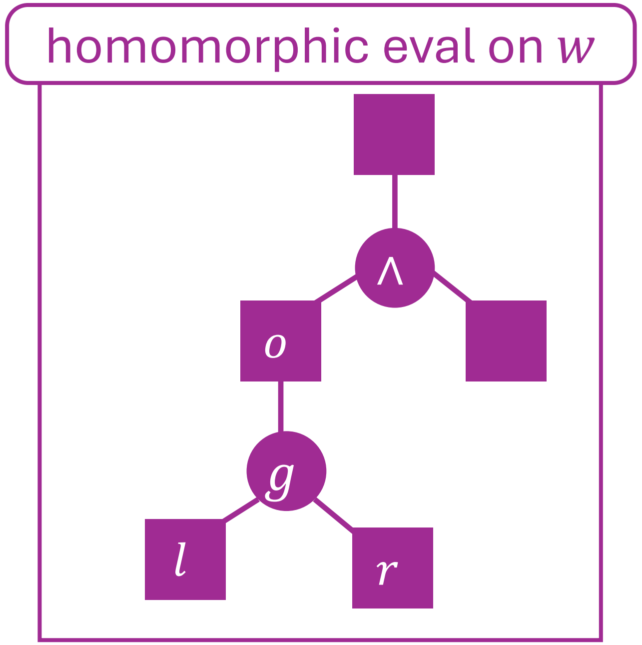
Applying BARGs

$$\text{CRS} = \text{FHE}\left[\, C(x,\cdot) \,\right]$$

$$\text{FHE}\big(C(x,w)\big)$$

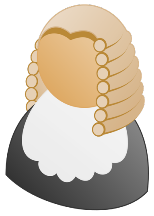


Applying BARGs



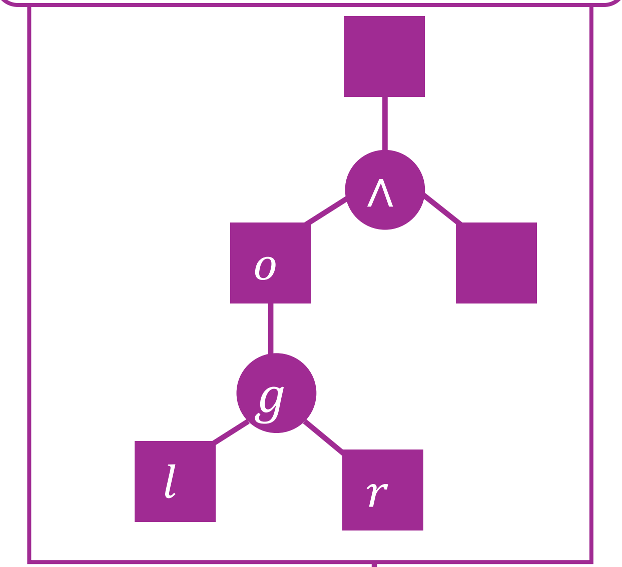
$$\mathrm{CRS} = \mathrm{FHE}\left[\,C(x,\cdot)\,\right]$$

$$\mathrm{FHE}\big(C(x,w)\big)$$



Applying BARGs

homomorphic eval on w

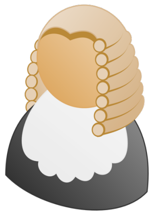


Merkle hash
ciphertexts of wires
(consistency)

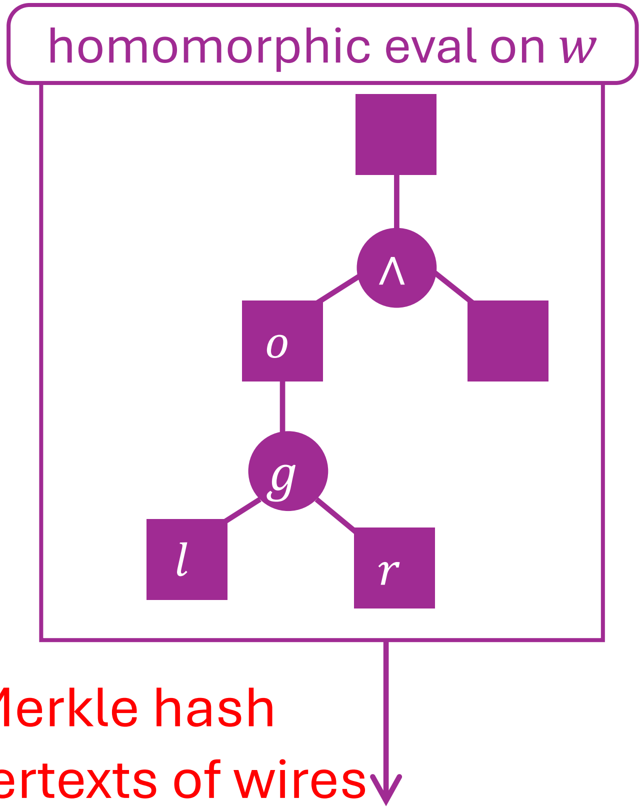
h

$$\text{CRS} = \text{FHE}\left[\, C(x,\cdot) \,\right]$$

$\text{FHE}\big(C(x,w)\big)$



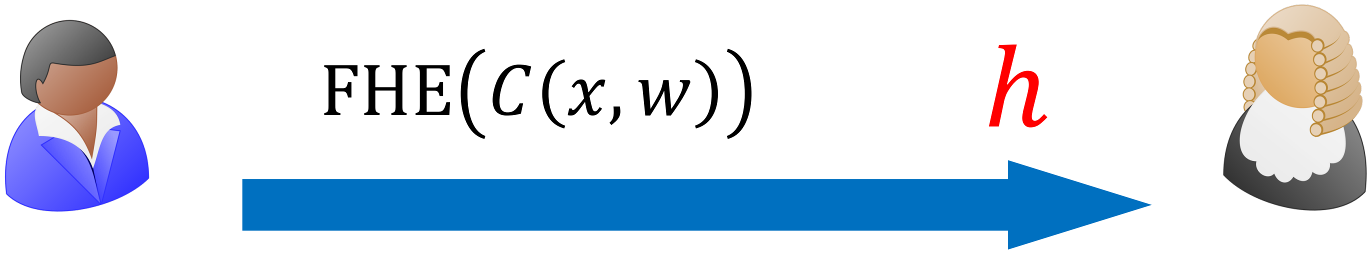
Applying BARGs



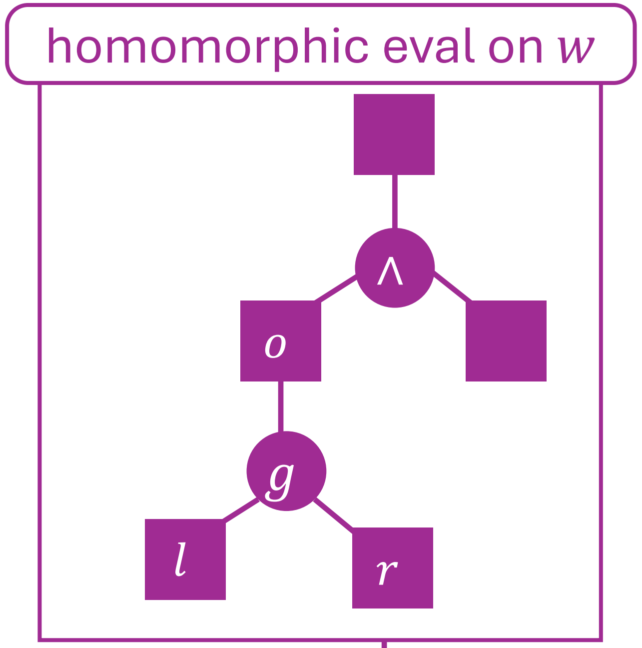
Merkle hash
ciphertexts of wires
(consistency)

h

$$\mathrm{CRS} = \mathrm{FHE}\left[C(x,\cdot) \right]$$



Applying BARGs

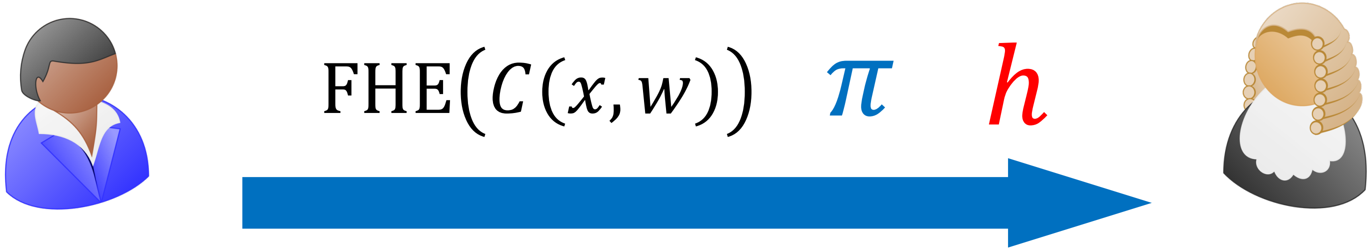


Merkle hash
ciphertexts of wires
(consistency)

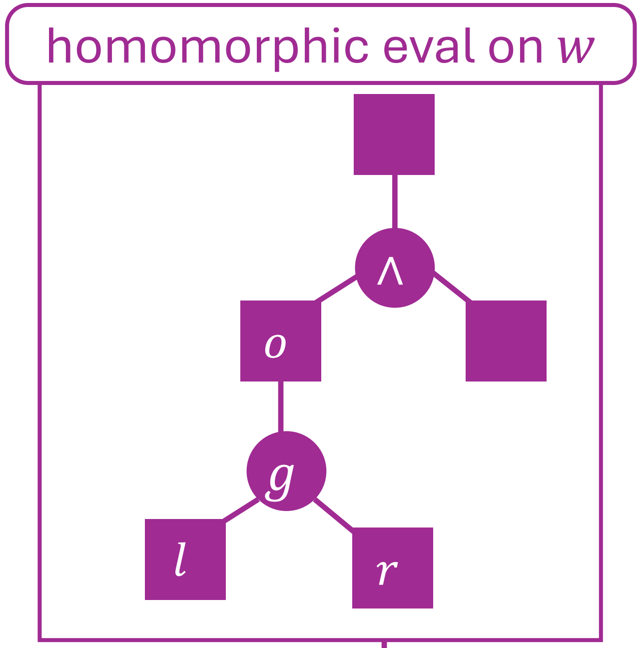
h

$\text{CRS} = \text{FHE}\left[\, C(x,\cdot) \,\right]$

$\text{FHE}\big(C(x,w)\big) \quad \pi \quad h$



Applying BARGs

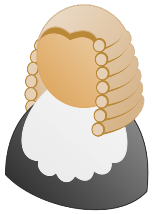


Merkle hash ciphertexts of wires (consistency)

h

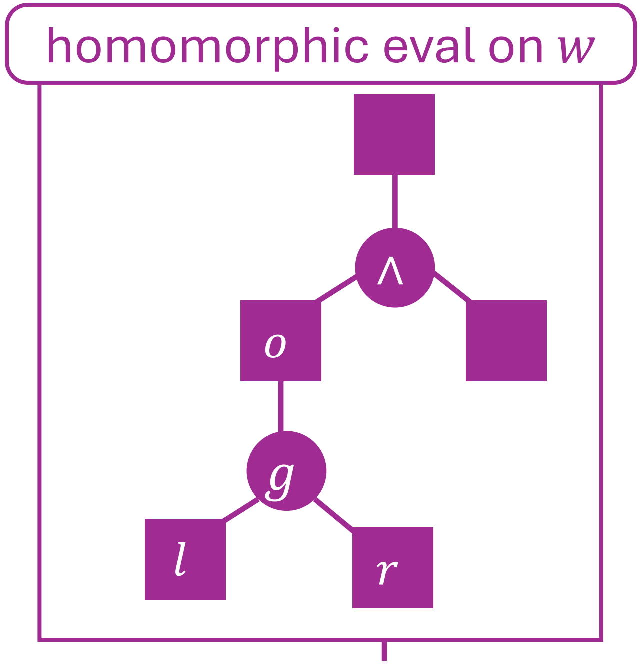
$\text{CRS} = \text{FHE}\left[C(x,\cdot) \right]$

$\text{FHE}(C(x,w))$ π h

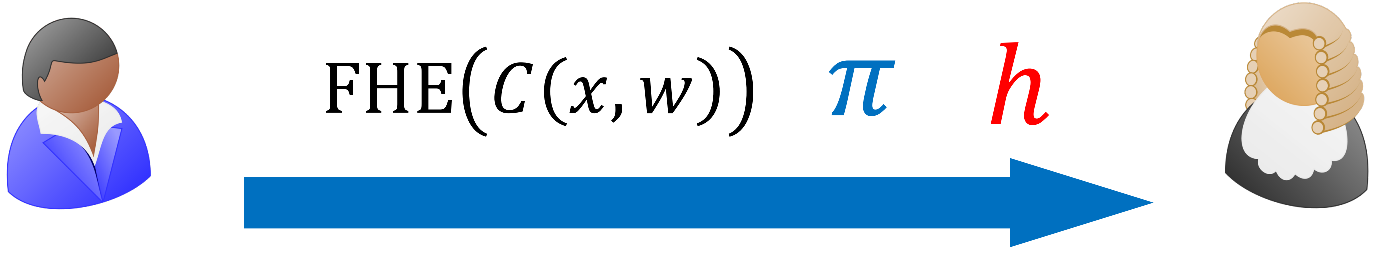


π proves "$\forall$ gate g, $\exists$ local openings ρ_l, ρ_r, ρ_o of ct_l, ct_r, ct_o w.r.t. h, and $ct_o = g(ct_l, ct_r)$" using BARG.

Applying BARGs



$$\mathrm{CRS} = \mathrm{FHE}\left[\,C(x,\cdot)\,\right]$$

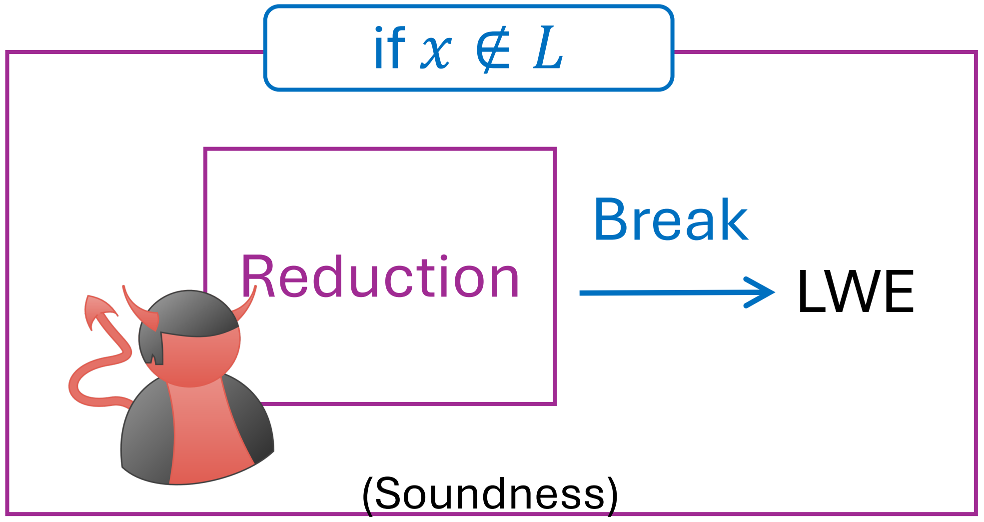


How to Prove the Soundness?

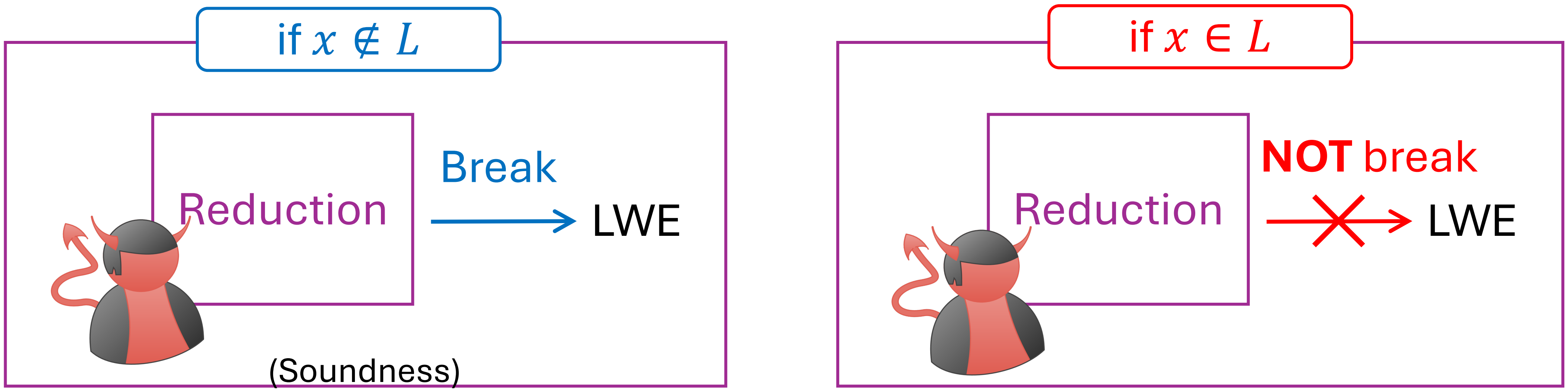
BARG $\Rightarrow$ only part of the evaluation is correct

(An Informal) **Barrier** in Soundness Reduction

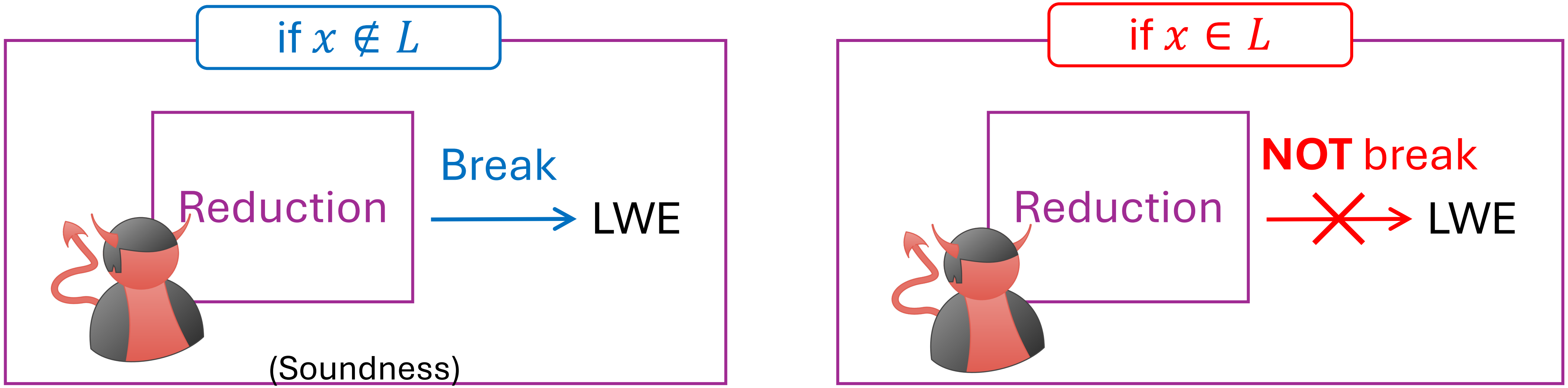
(An Informal) **Barrier** in Soundness Reduction



(An Informal) **Barrier** in Soundness Reduction



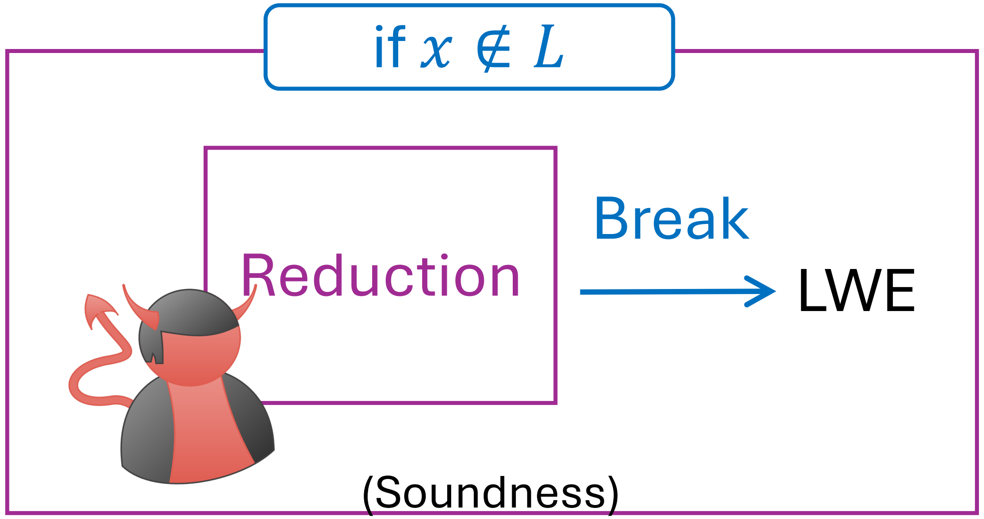
(An Informal) **Barrier** in Soundness Reduction

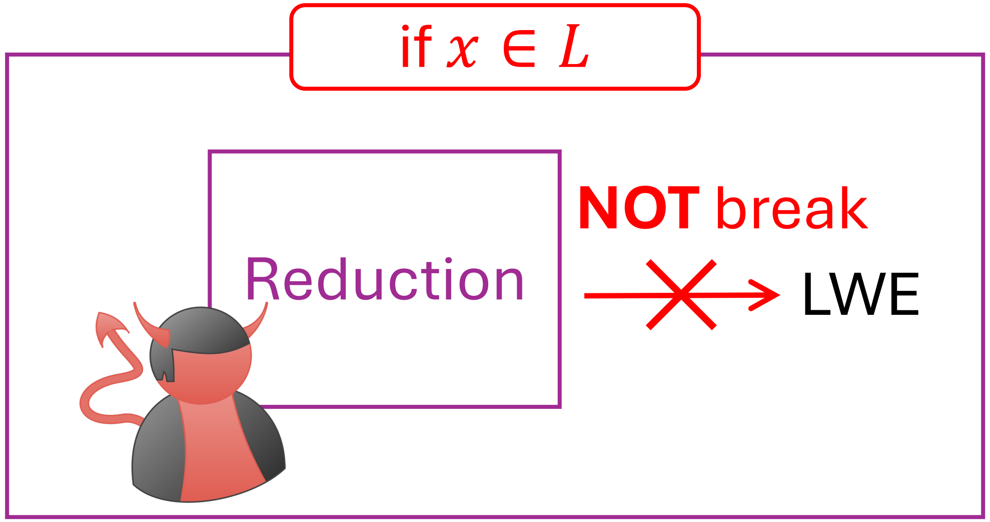


Reduction seems need to "tell" whether $x \notin L$ or $x \in L$.

(Gentry-Wichs: formalize this intuition (with caveat))

(An Informal) **Barrier** in Soundness Reduction





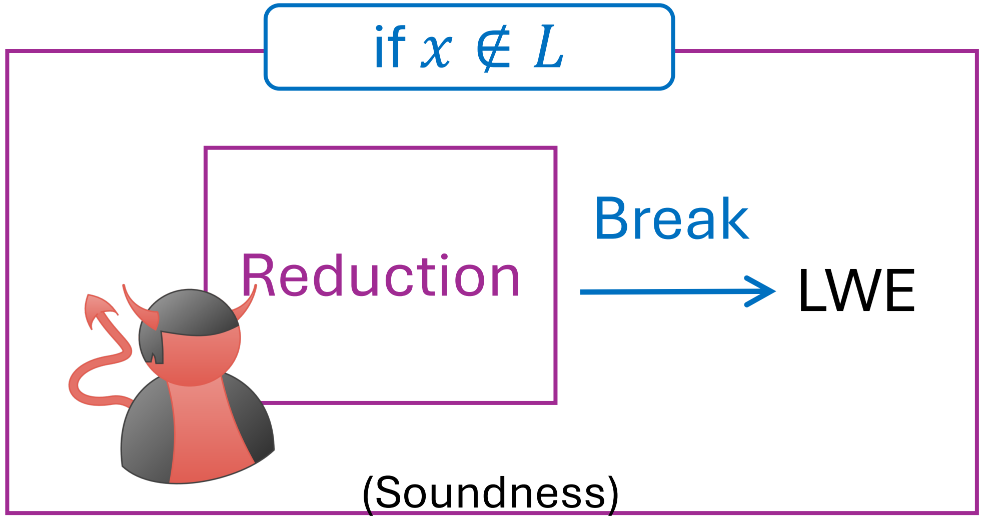
Reduction seems need to "tell" whether $x \notin L$ or $x \in L$.

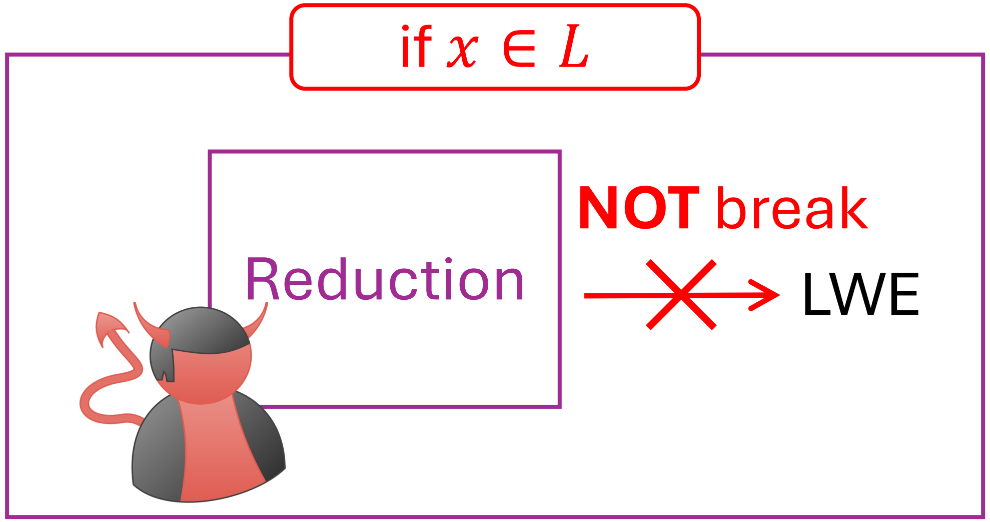
(Gentry-Wichs: formalize this intuition (with caveat))

If the reduction runs in $2^{|w|}$-time $\Rightarrow$ FHE security parameters $\geq |w|$

Not Succinct!

(An Informal) **Barrier** in Soundness Reduction





Reduction seems need to "tell" whether $x \notin L$ or $x \in L$.

(Gentry-Wichs: formalize this intuition (with caveat))

If the reduction runs in $2^{|w|}$-time ⇒ FHE security parameters $\geq |w|$

Not Succinct!

(poly-time reduction?)

Efficient Reduction via
Logical Structure of the Language [JJ'22]

Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$

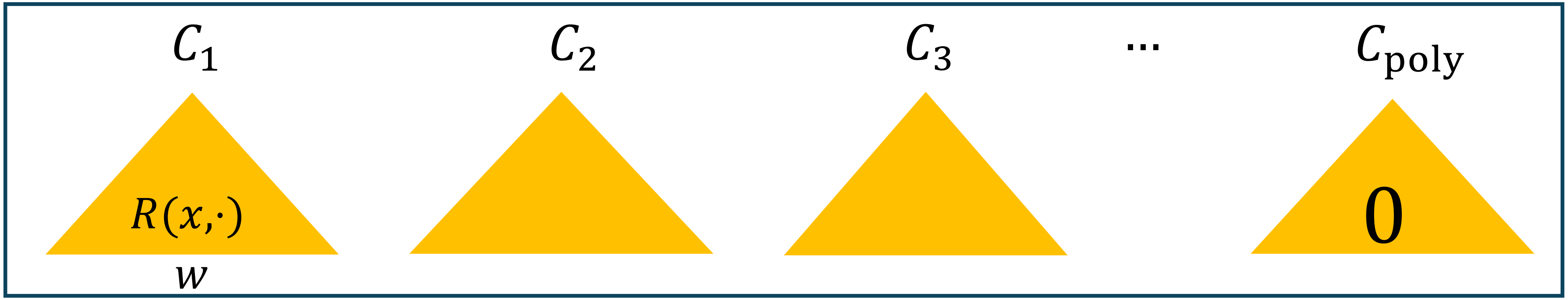
Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$

⬇

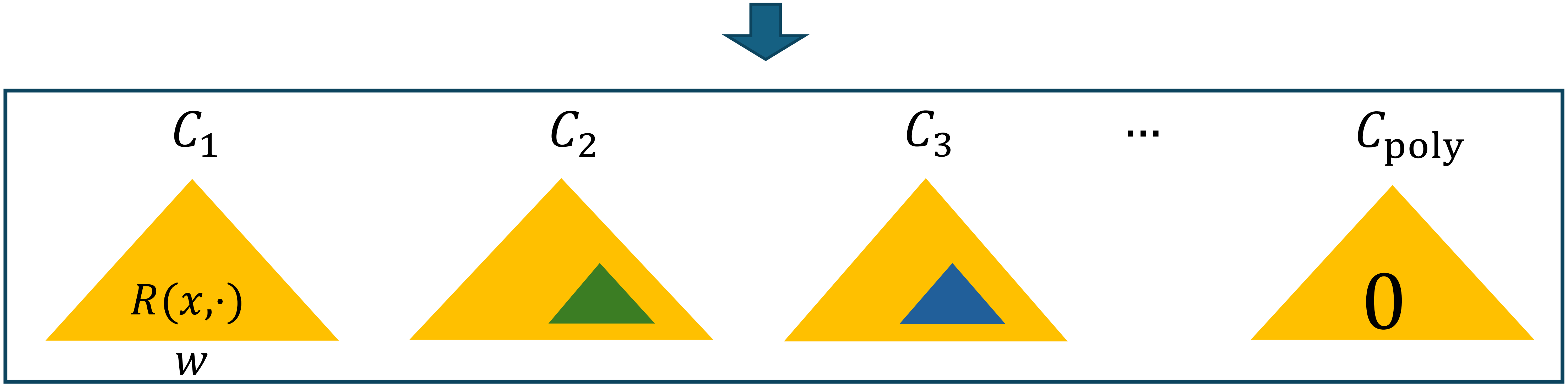
Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$



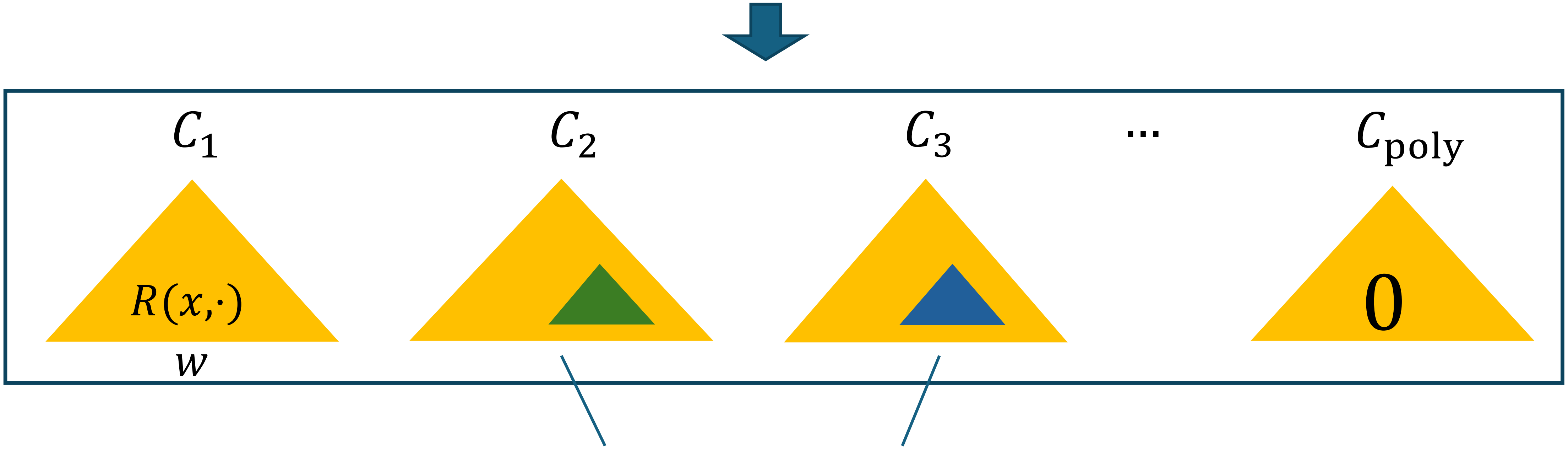
Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$



Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$

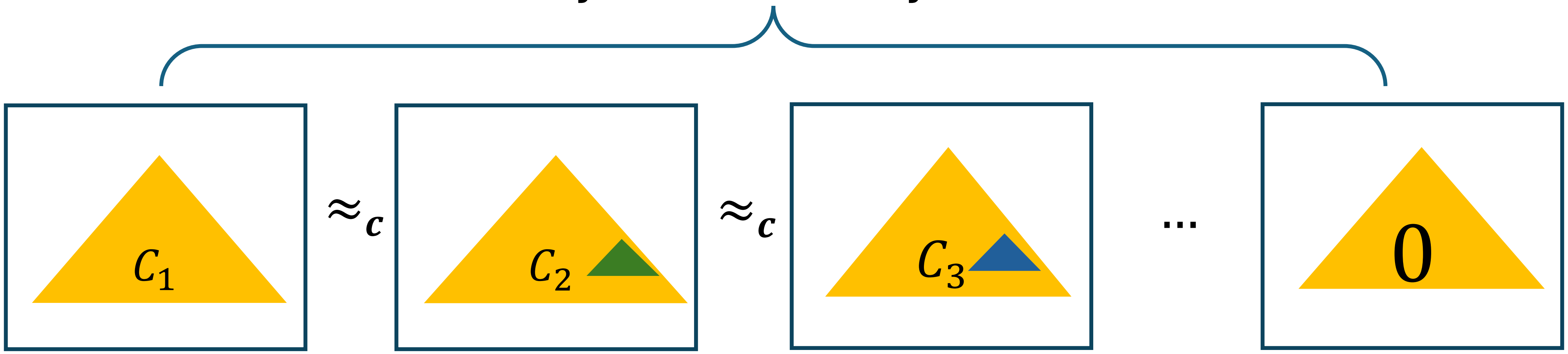


Adjacent pair of circuits are almost the same,
except for a ***O(log n) size*** sub-circuits of the same functionality.

Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$

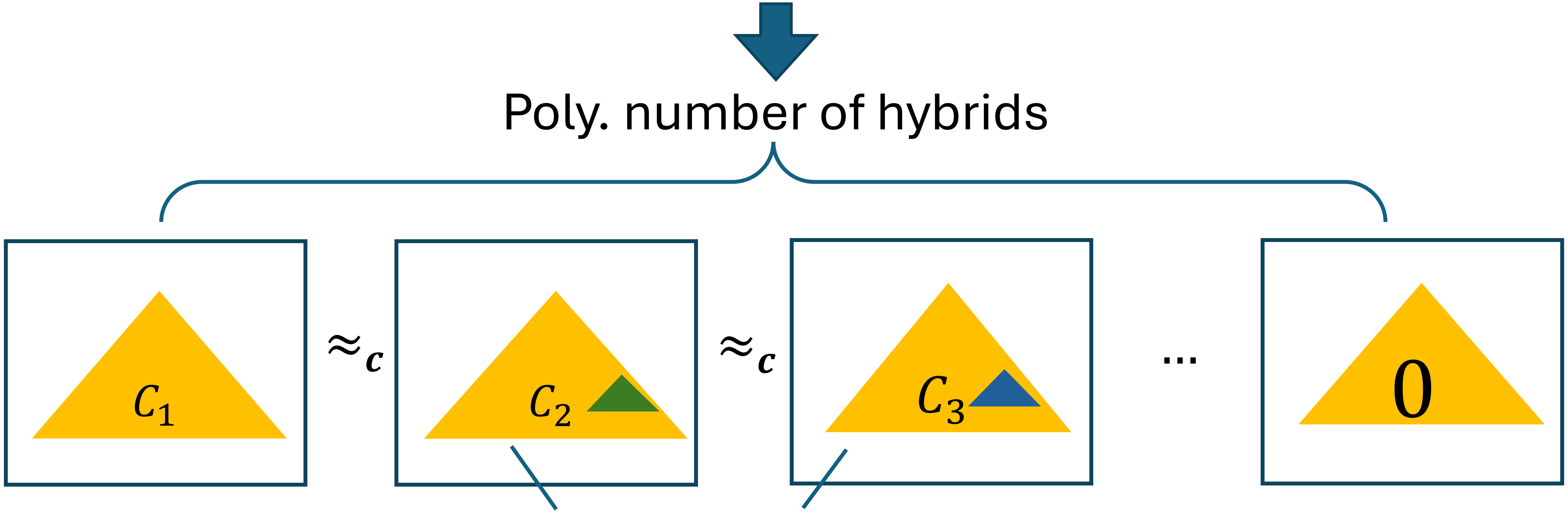
Poly. number of hybrids



Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$

Poly. number of hybrids

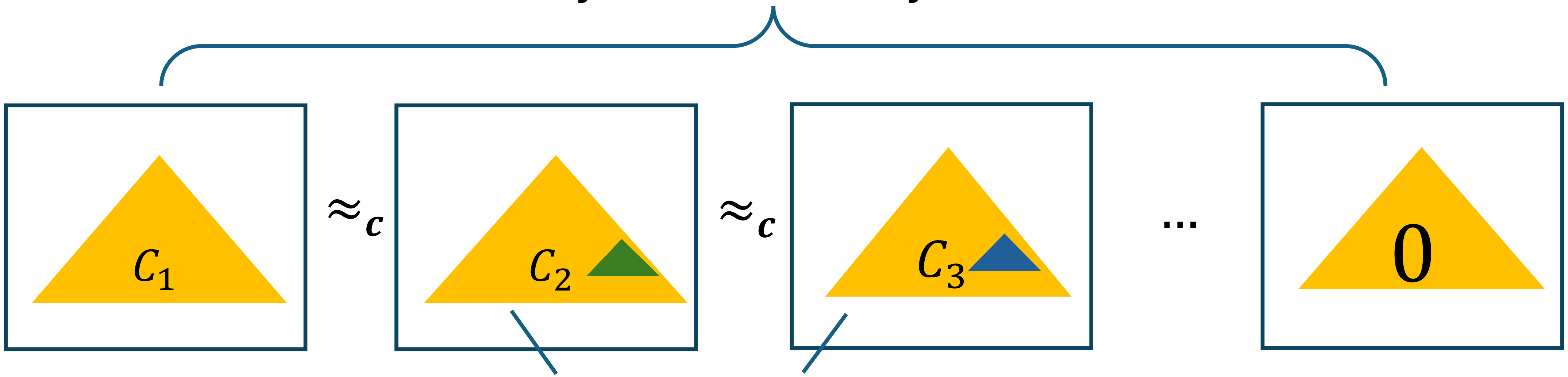


A "Puncturing Argument" with **quasi-polynomial time** reduction

Efficient Reduction via
Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$

Poly. number of hybrids



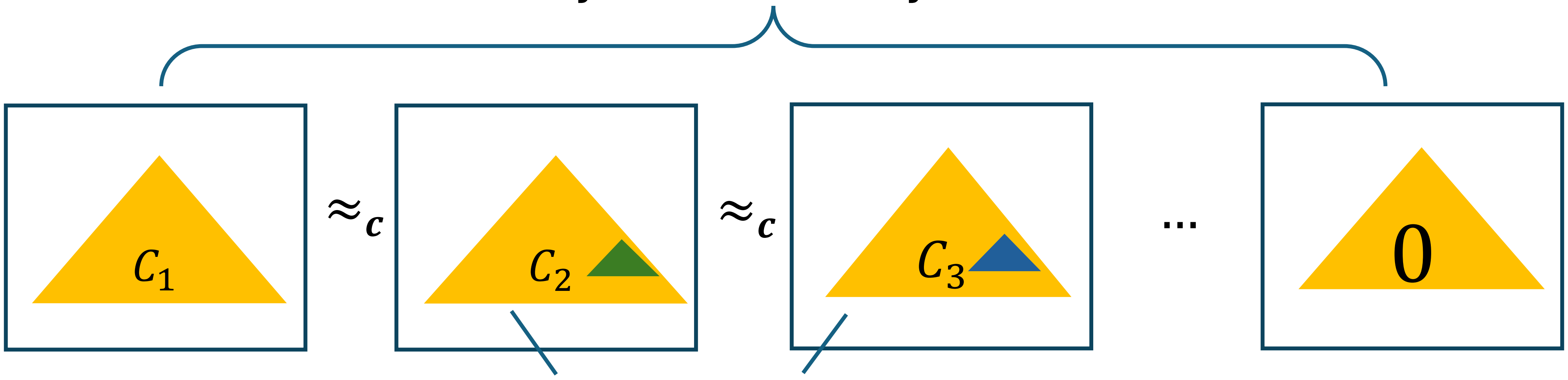
A "Puncturing Argument" with **quasi-polynomial time** reduction

$\Rightarrow$ Total reduction time = **quasi-poly**!

Efficient Reduction via Logical Structure of the Language [JJ'22]

Poly-size Extended Frege proof of $R(x,\cdot) = 0$

Poly. number of hybrids



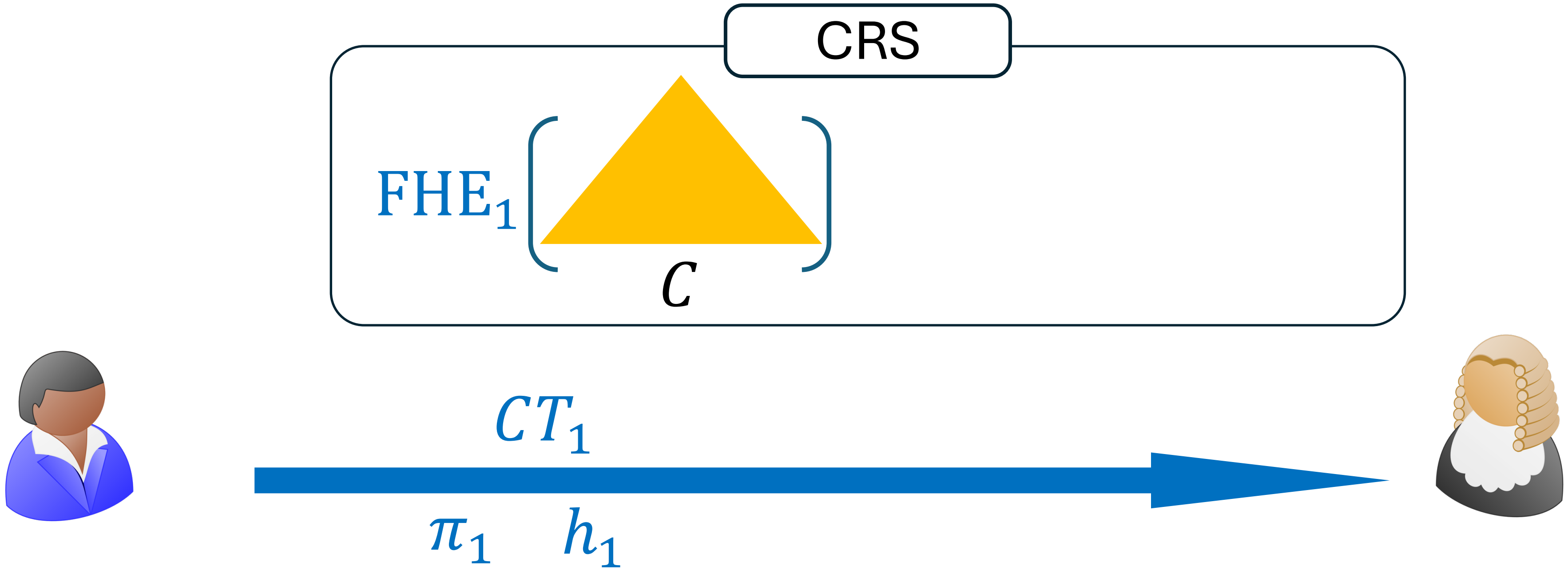
A "Puncturing Argument" with quasi-polynomial time reduction

JJ'22 realizes "puncturing argument" using iO

⇒ Total reduction time = **quasi-poly**!

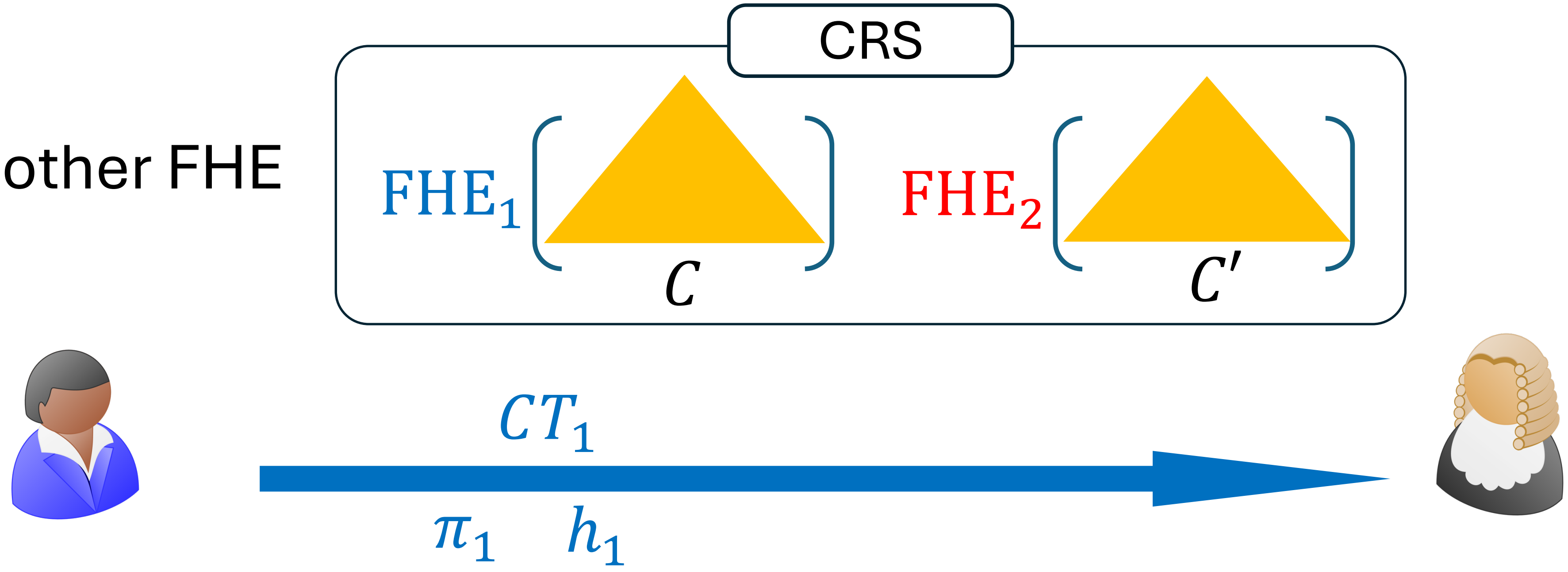
Our Approach

Puncturing Argument, via FHE! (Informal)



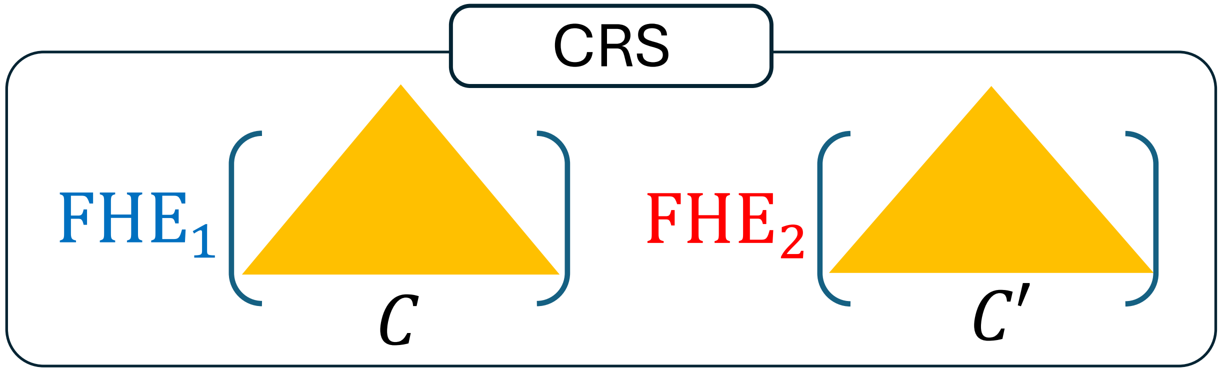
Puncturing Argument, via FHE! (Informal)

Add another FHE



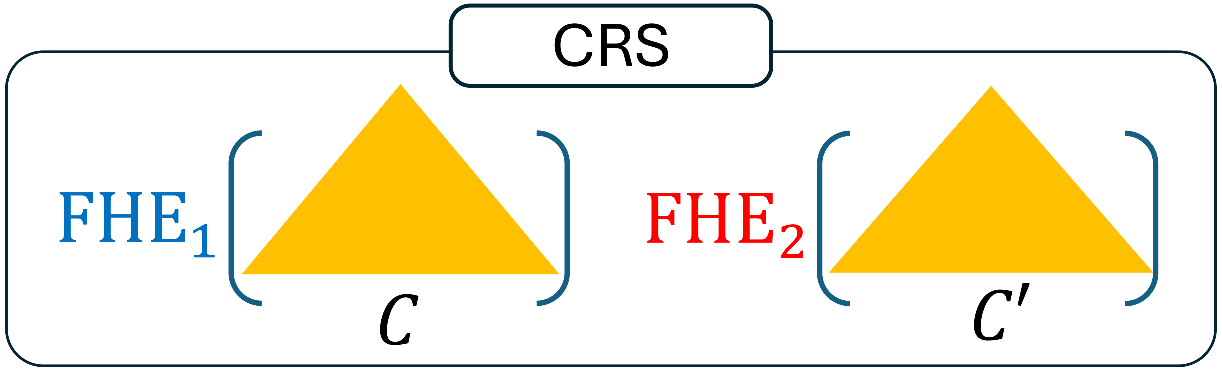
Puncturing Argument, via FHE! (Informal)

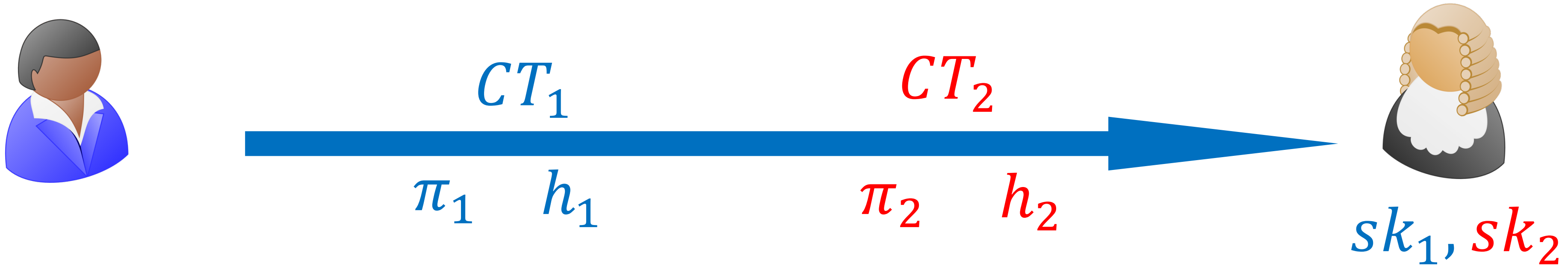
Add another FHE



Puncturing Argument, via FHE! (Informal)

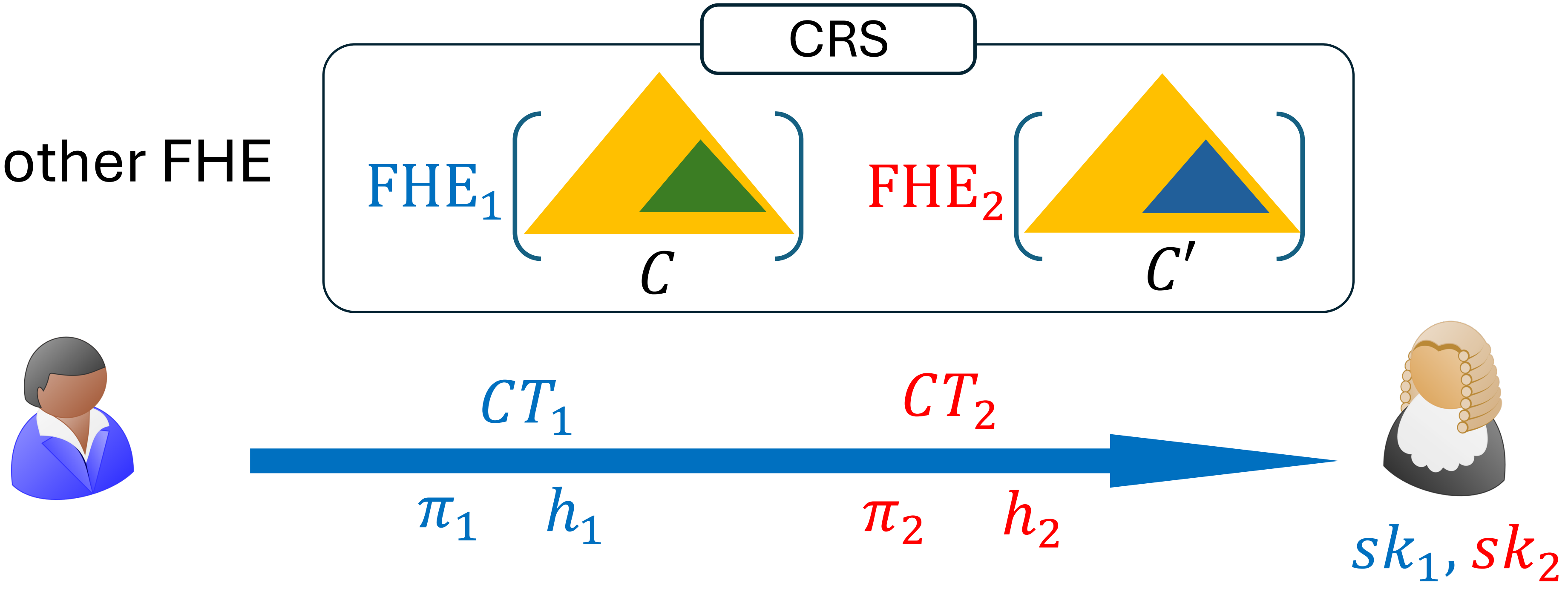
Add another FHE





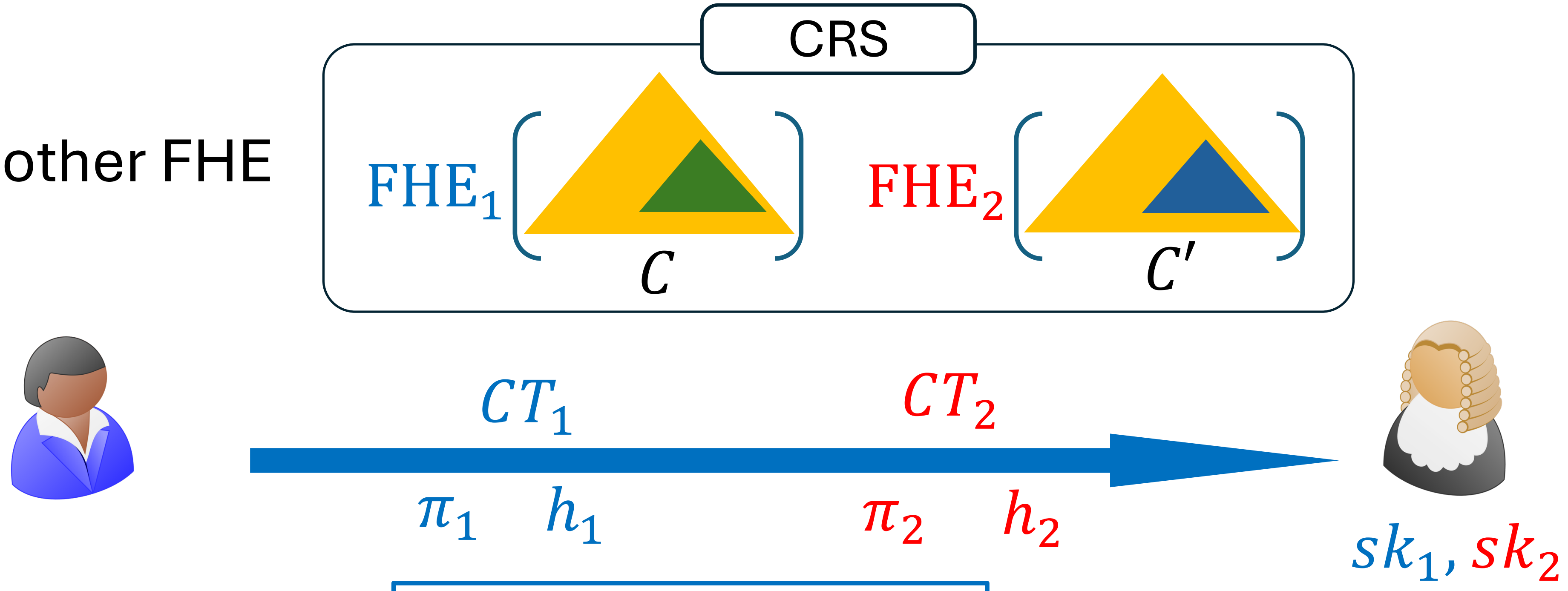
Puncturing Argument, via FHE! (Informal)

Add another FHE



Puncturing Argument, via FHE! (Informal)

Add another FHE



Puncturing Argument

If C and C' are almost the same except for a functionality equivalent ***O(log n)*** sub-ckt, then $\mathrm{Dec}(sk_1, CT_1) = 1$ implies $\mathrm{Dec}(sk_2, CT_2) = 1$, using **poly-secure BARG.**

Proving Soundness via Puncturing Argument

...

Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

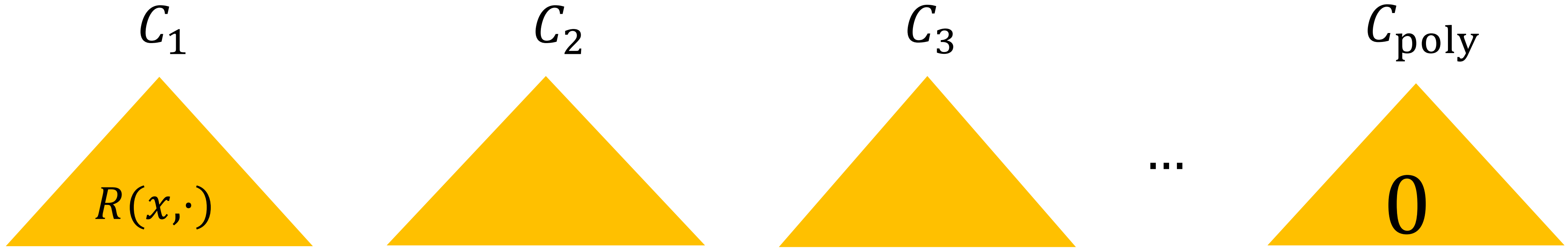
$\Downarrow$ [JJ22]

...

Proving Soundness via Puncturing Argument

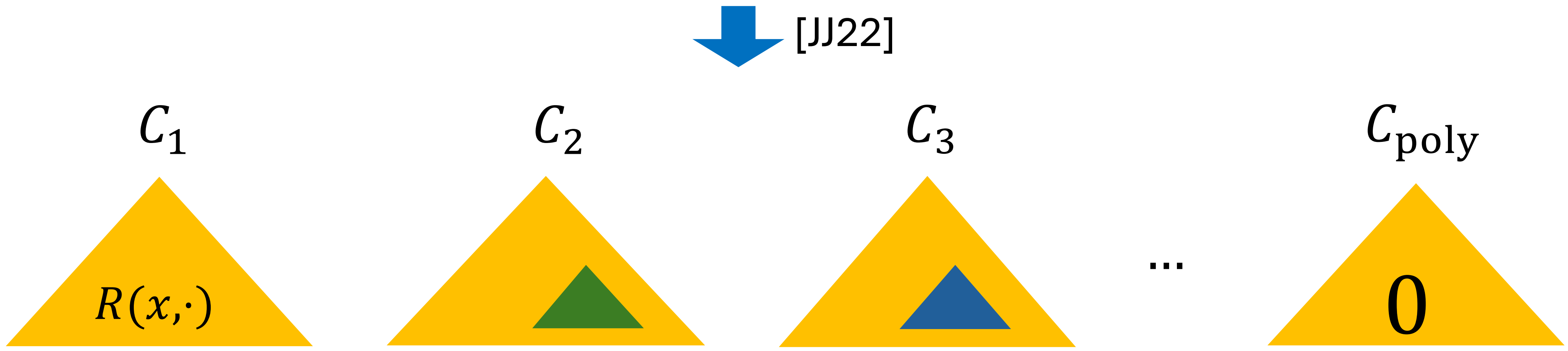
Poly-size Extended Frege proof of $R(x, w) = 0$

[JJ22]



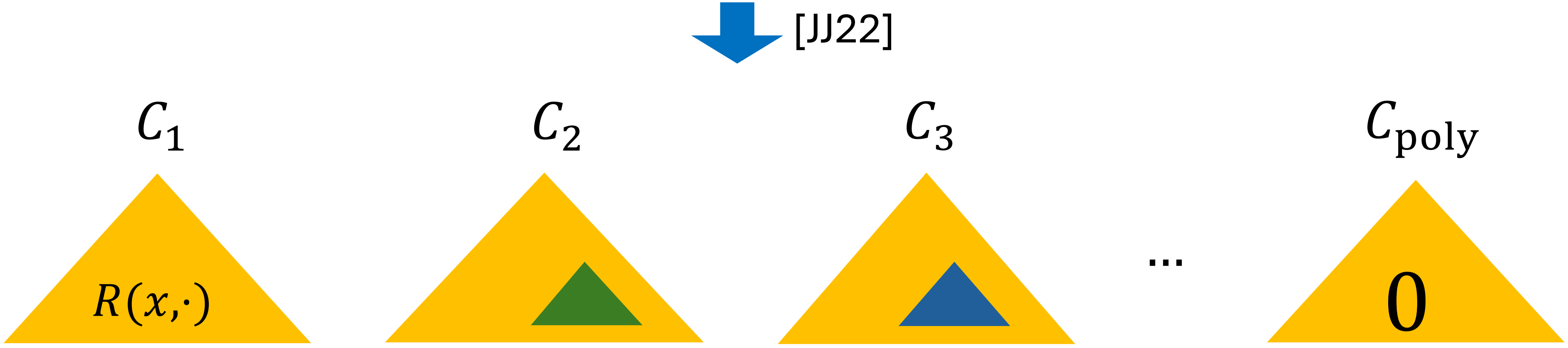
Proving Soundness via Puncturing Argument

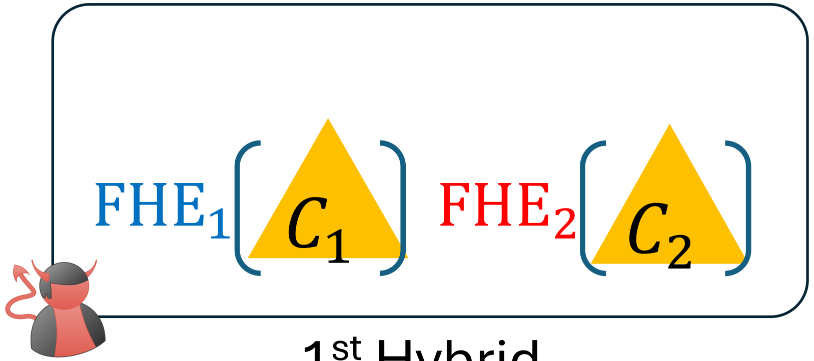
Poly-size Extended Frege proof of $R(x, w) = 0$



Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

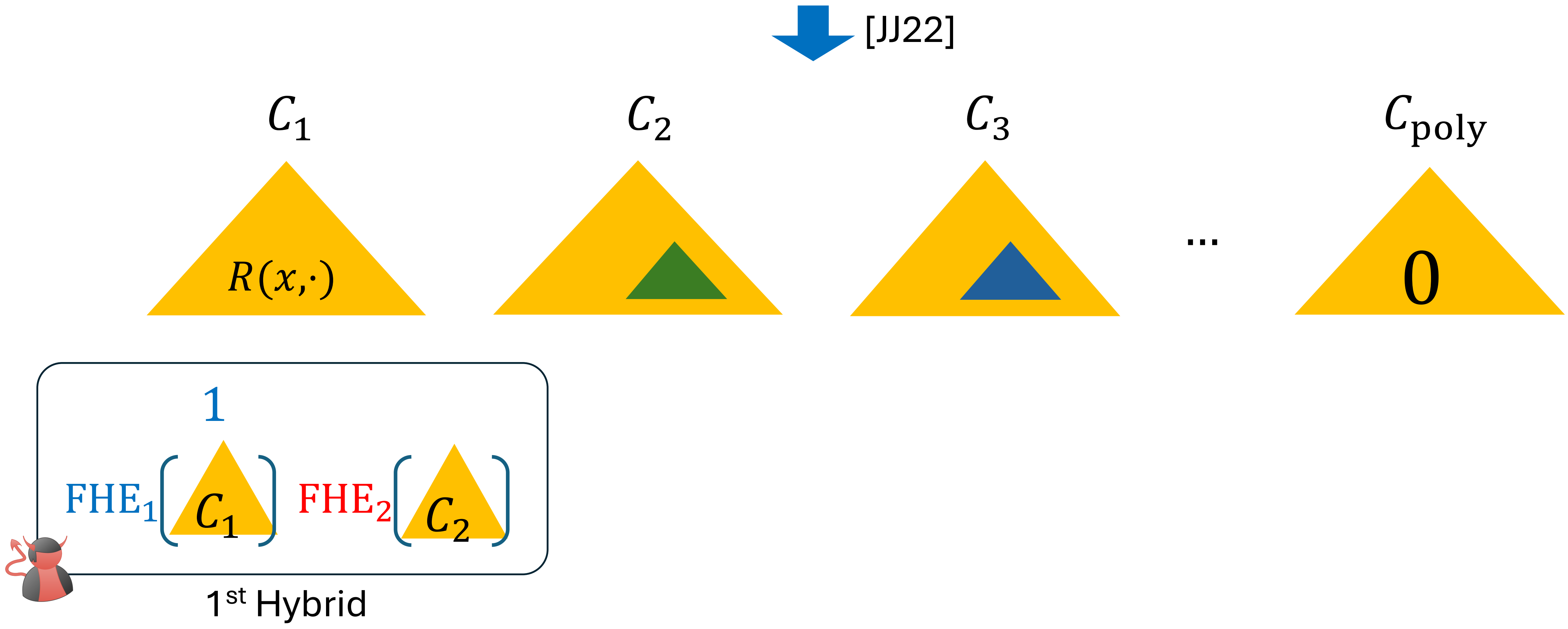




1st Hybrid

Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x,w) = 0$



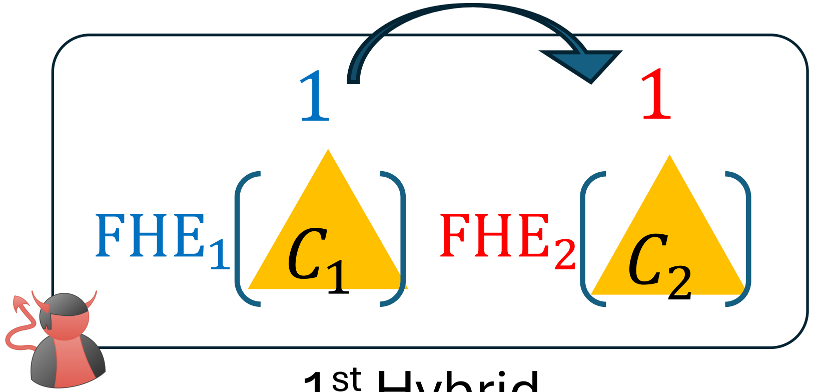
Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

[JJ22]

...

Puncturing Argument



1^{st} Hybrid

Proving Soundness via Puncturing Argument

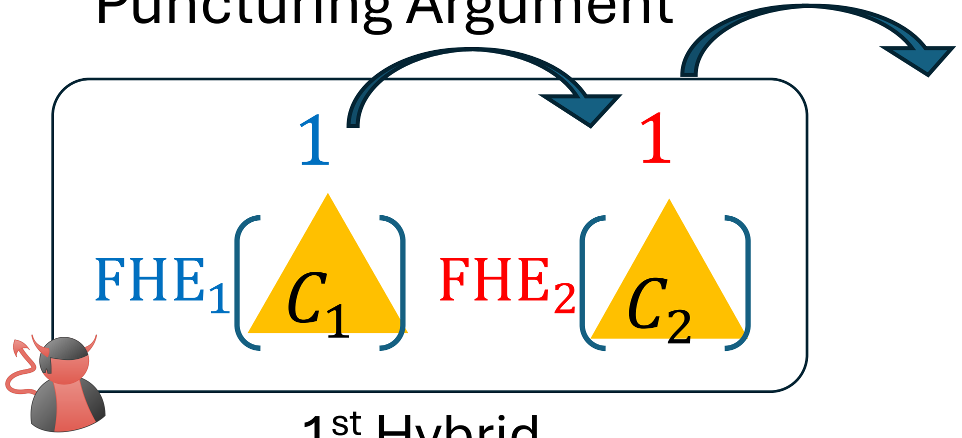
Poly-size Extended Frege proof of $R(x, w) = 0$

[JJ22]

...

FHE_1 semantic security

Puncturing Argument

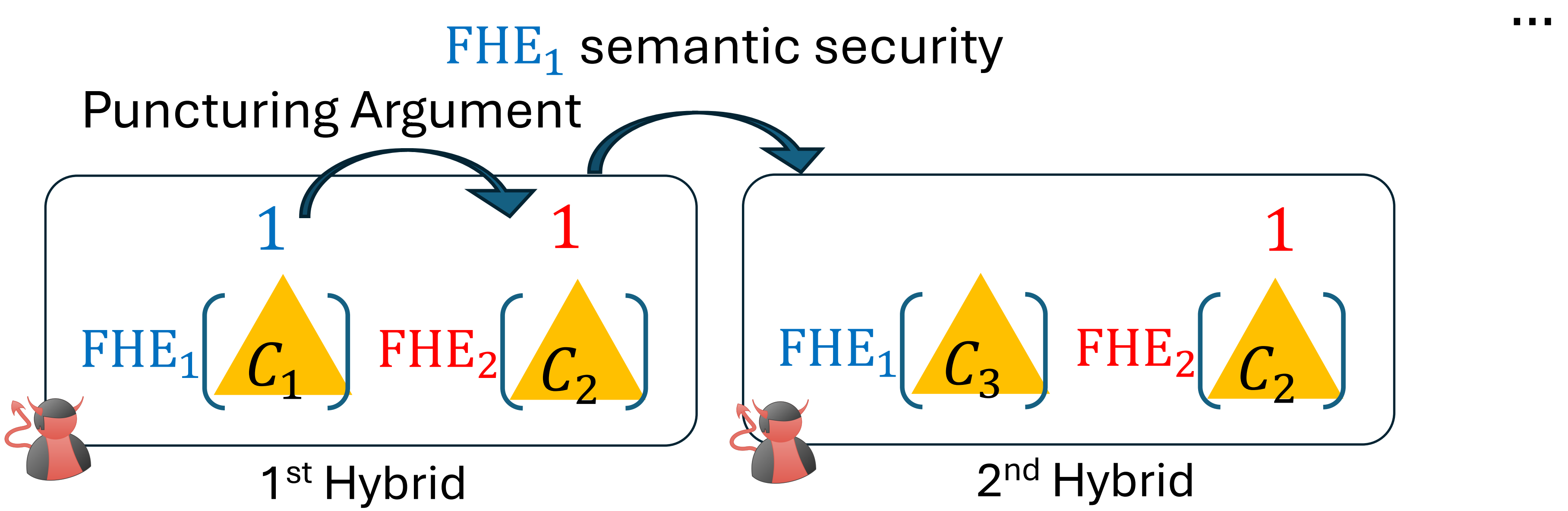


1st Hybrid

Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

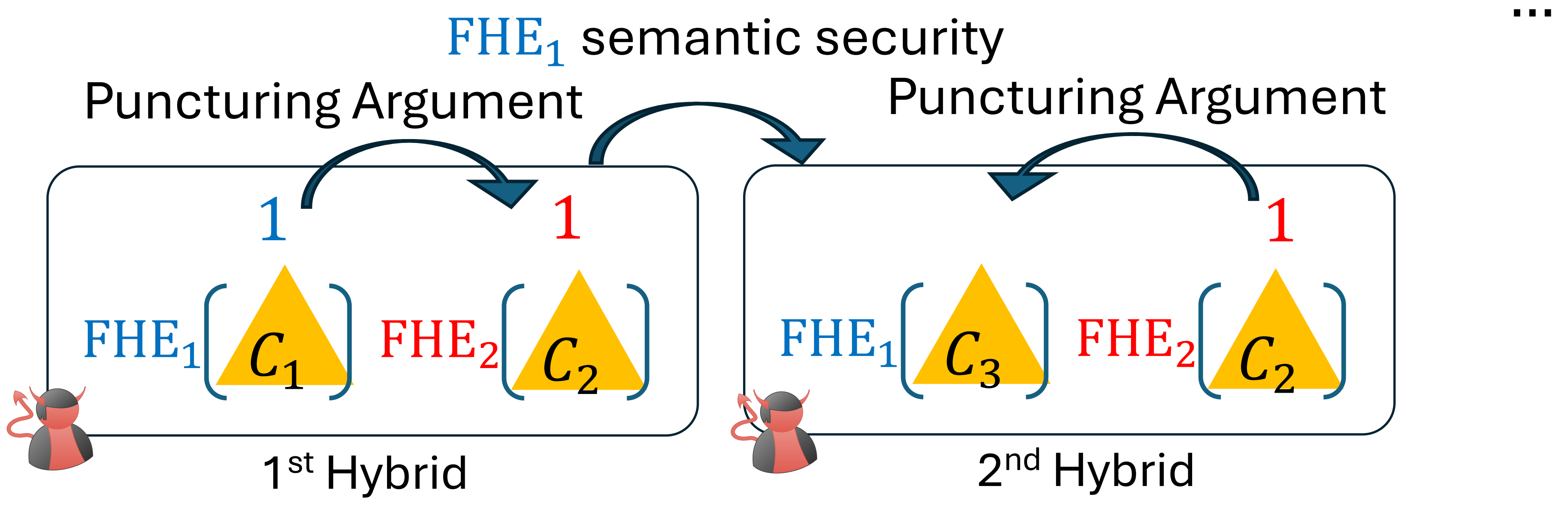
$\Downarrow$ [JJ22]



Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

[JJ22]

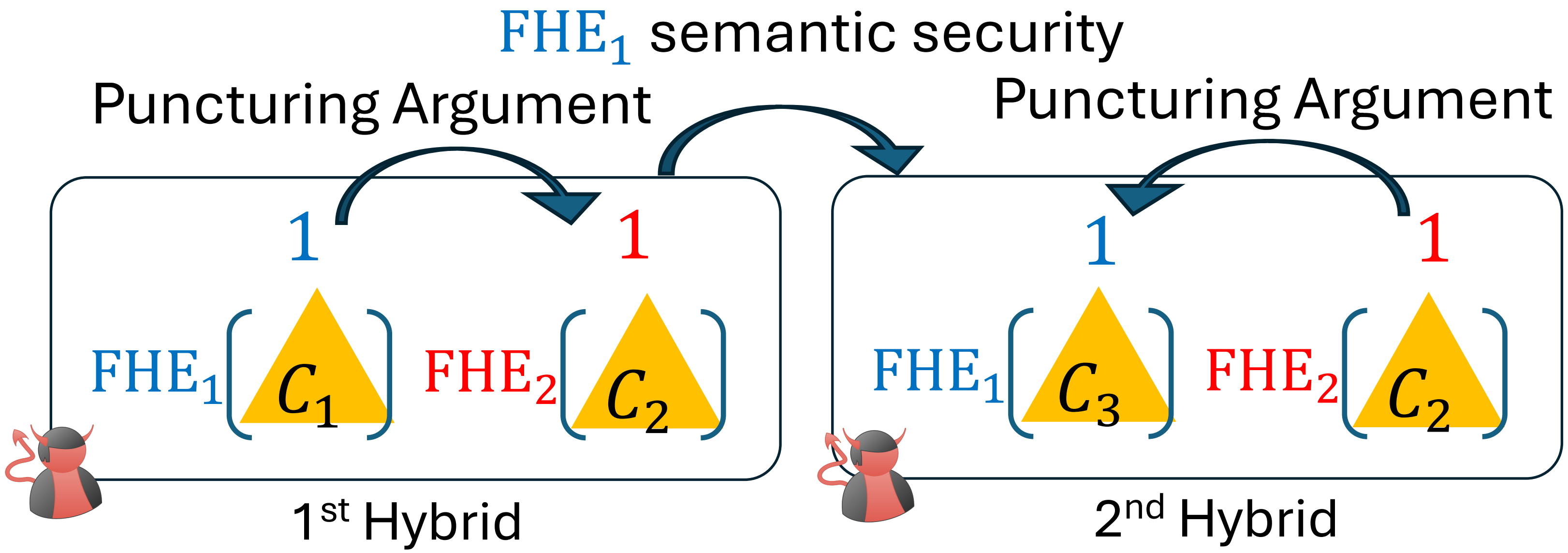


Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

[JJ22]

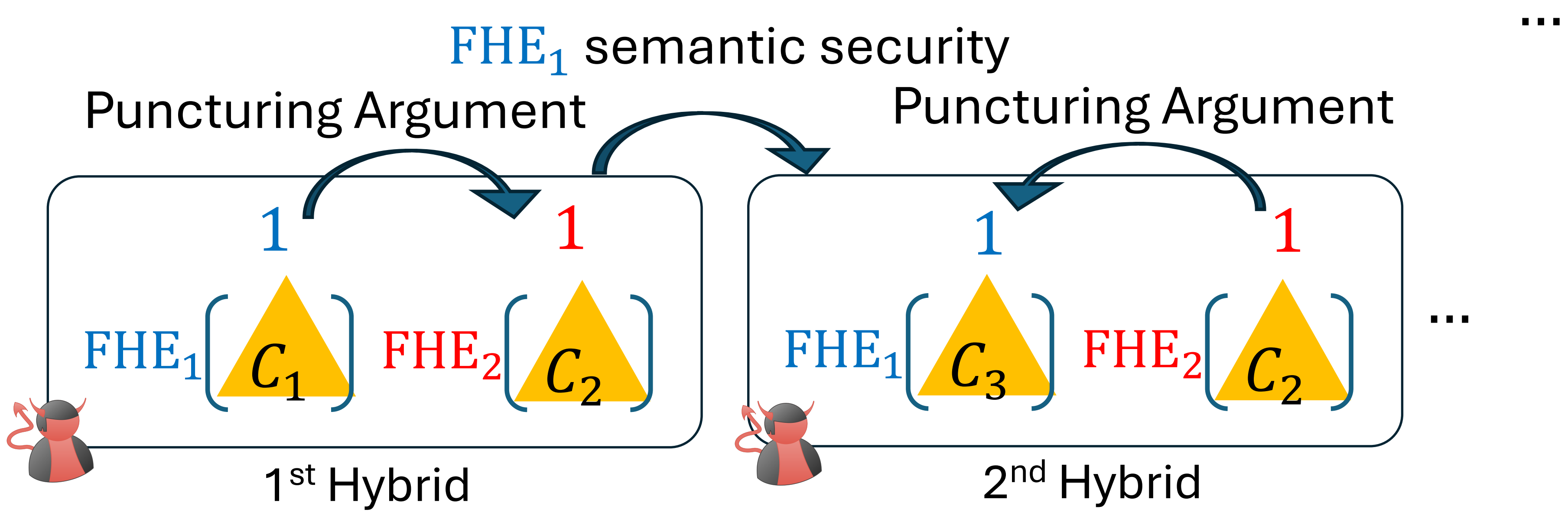
...



Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

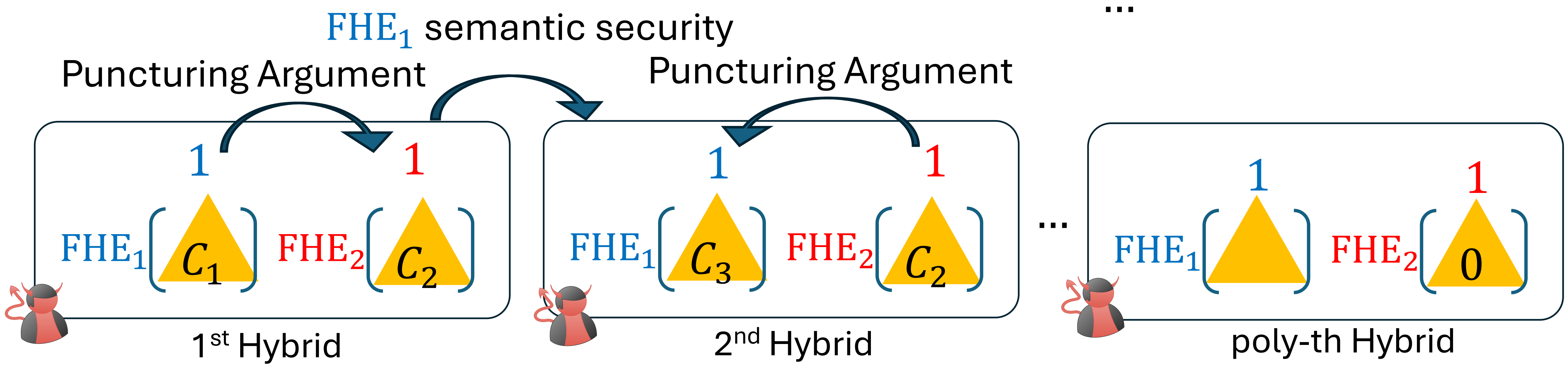
[JJ22]



Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

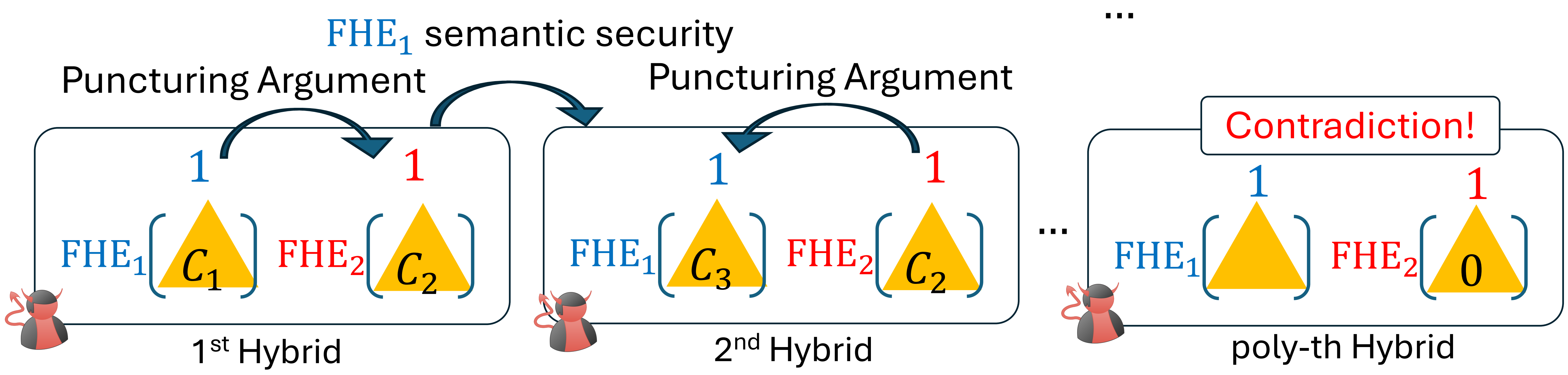
[JJ22]



Proving Soundness via Puncturing Argument

Poly-size Extended Frege proof of $R(x, w) = 0$

[JJ22]



Rest of the Talk

- Proof of Puncturing Argument
- Achieving Public Verification & Random CRS
- Discussion

Rest of the Talk

- **Proof of Puncturing Argument**
- Achieving Public Verification & Random CRS
- Discussion

Recall: Somewhere Statistical Binding (SSB) Hash

[Hubacek-Wichs'15, Okamoto-Pietrzak-Waters-Wichs'15]

Recall: Somewhere Statistical Binding (SSB) Hash

[Hubacek-Wichs'15, Okamoto-Pietrzak-Waters-Wichs'15]

Hash Key: $K(S \subseteq [n])$ (Pseudorandom)

Recall: Somewhere Statistical Binding (SSB) Hash

[Hubacek-Wichs'15, Okamoto-Pietrzak-Waters-Wichs'15]

Hash Key: $K(S \subseteq [n])$ (Pseudorandom)

$h \leftarrow \mathrm{SSB}(K, m_1, m_2, \ldots, m_n)$ $|h| \approx |S|$

Recall: Somewhere Statistical Binding (SSB) Hash

[Hubacek-Wichs'15, Okamoto-Pietrzak-Waters-Wichs'15]

Hash Key: $K(S \subseteq [n])$ (Pseudorandom)

$h \leftarrow \mathrm{SSB}(K, m_1, m_2, \ldots, m_n)$ $|h| \approx |S|$

Extraction: $m_S \leftarrow \mathrm{Ext}(\mathrm{td}, h)$

Recall: Somewhere Statistical Binding (SSB) Hash

[Hubacek-Wichs'15, Okamoto-Pietrzak-Waters-Wichs'15]

Hash Key: $K(S \subseteq [n])$ (Pseudorandom)

$h \leftarrow \mathrm{SSB}(K, m_1, m_2, \ldots, m_n)$ $|h| \approx |S|$

Extraction: $m_S \leftarrow \mathrm{Ext}(\mathrm{td}, h)$

No-Signaling Property: for any two subsets S_1, S_2,

Recall: Somewhere Statistical Binding (SSB) Hash

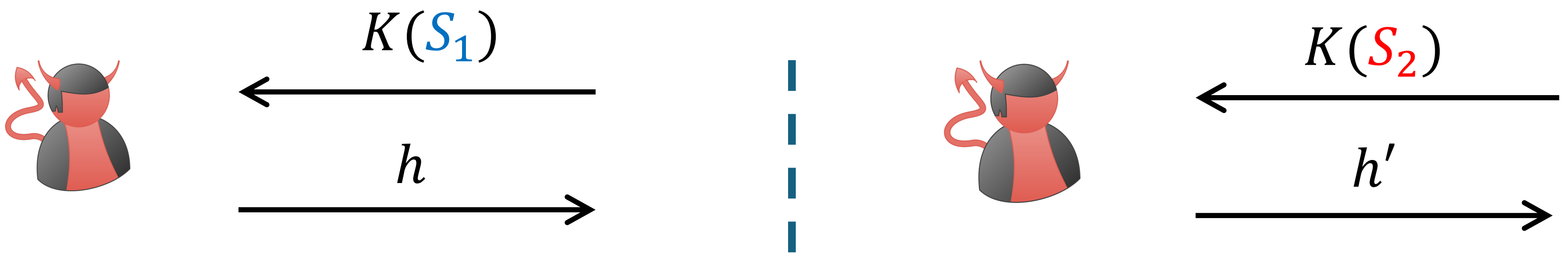
[Hubacek-Wichs'15, Okamoto-Pietrzak-Waters-Wichs'15]

Hash Key: $K(S \subseteq [n])$ (Pseudorandom)

$h \leftarrow \mathrm{SSB}(K, m_1, m_2, \ldots, m_n)$ $|h| \approx |S|$

Extraction: $m_S \leftarrow \mathrm{Ext}(\mathrm{td}, h)$

No-Signaling Property: for any two subsets S_1, S_2,



Recall: Somewhere Statistical Binding (SSB) Hash

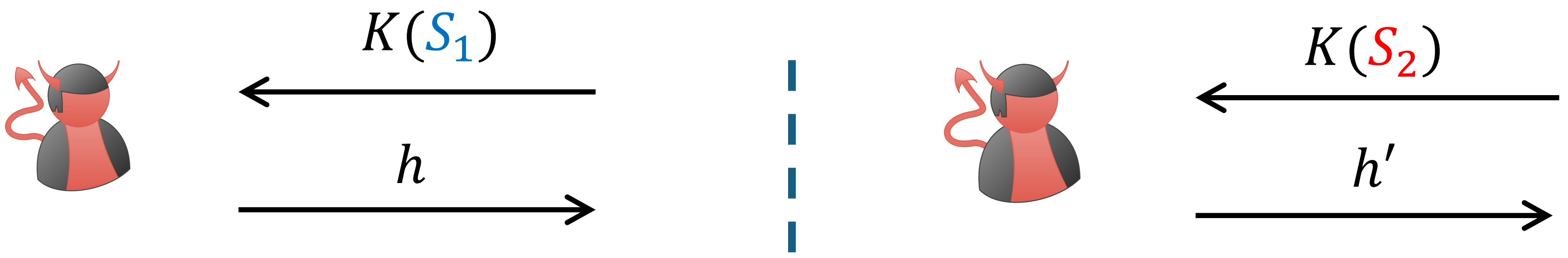
[Hubacek-Wichs'15, Okamoto-Pietrzak-Waters-Wichs'15]

Hash Key: $K(S \subseteq [n])$ (Pseudorandom)

$h \leftarrow \mathrm{SSB}(K, m_1, m_2, \ldots, m_n)$ $\quad |h| \approx |S|$

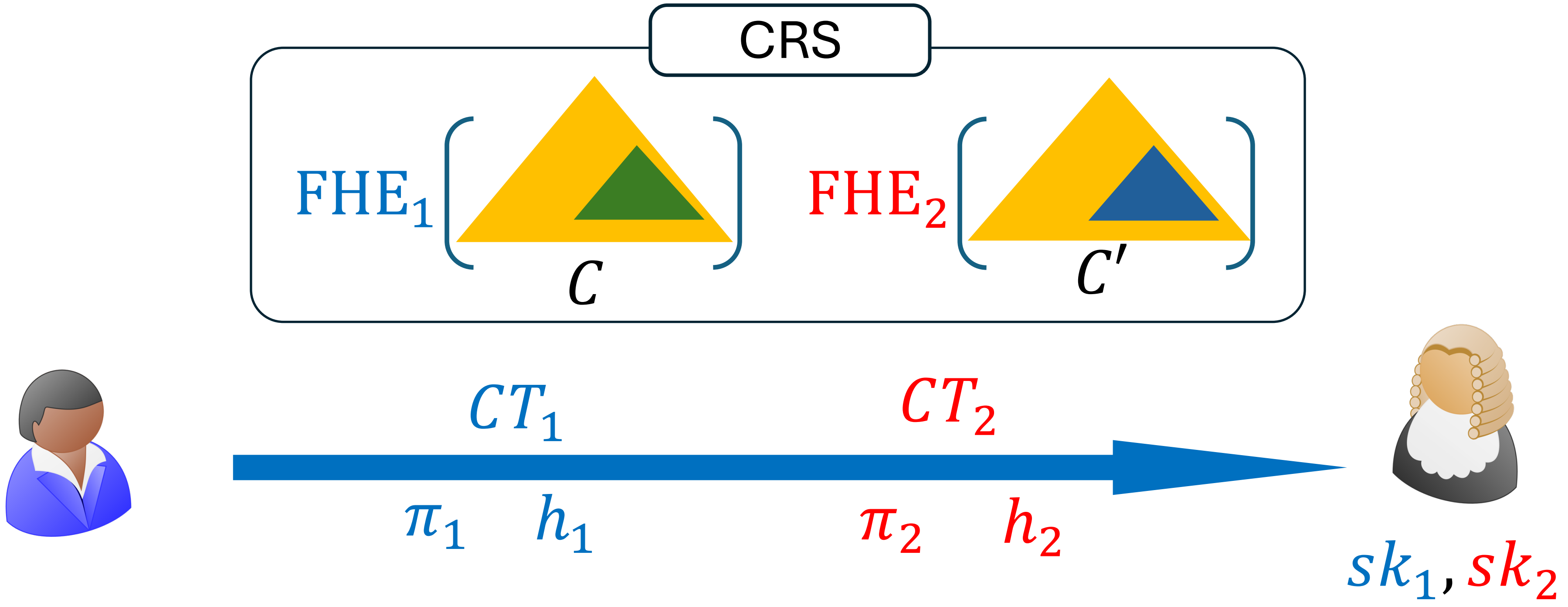
Extraction: $m_S \leftarrow \mathrm{Ext}(\mathrm{td}, h)$

No-Signaling Property: for any two subsets S_1, S_2,



$$m_{S_1 \cap S_2} \approx_c m'_{S_1 \cap S_2}$$

Recall: Puncturing Argument



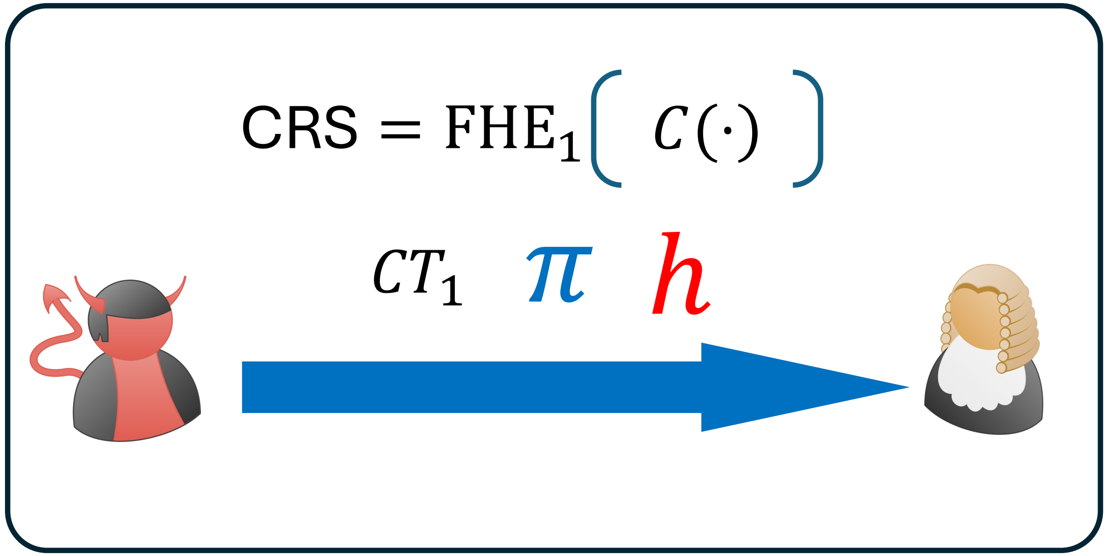
If C and C' are almost the same except for a functionality equivalent ***O(log n)*** sub-ckt, then $\mathrm{Dec}(sk_1, CT_1) = 1$ implies $\mathrm{Dec}(sk_2, CT_2) = 1$, assuming **poly-secure BARG.**

A Simplified View via Local Assignment Generator

[BHK17]

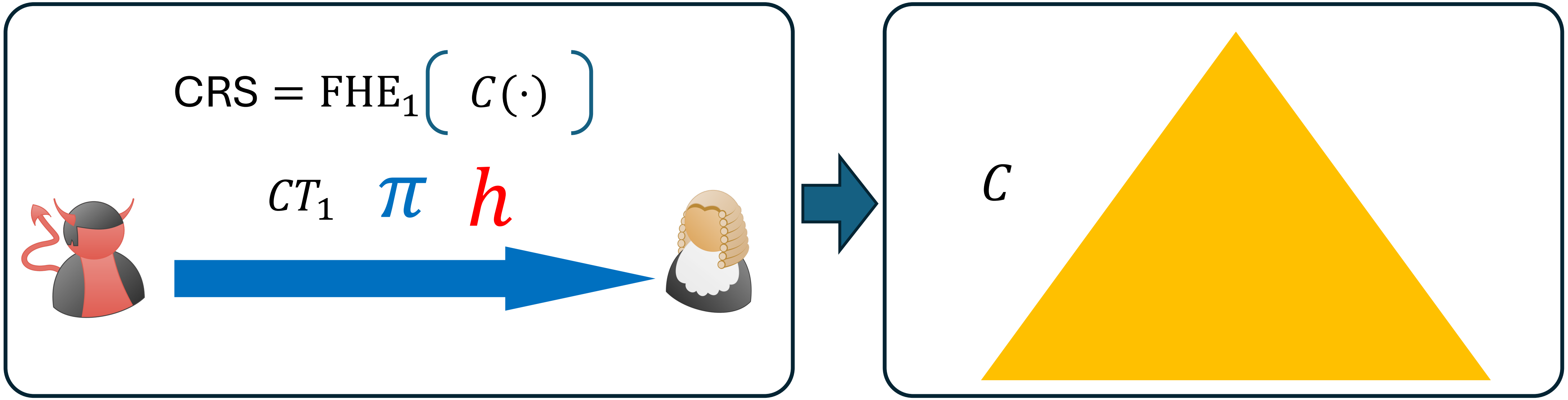
A Simplified View via Local Assignment Generator

[BHK17]



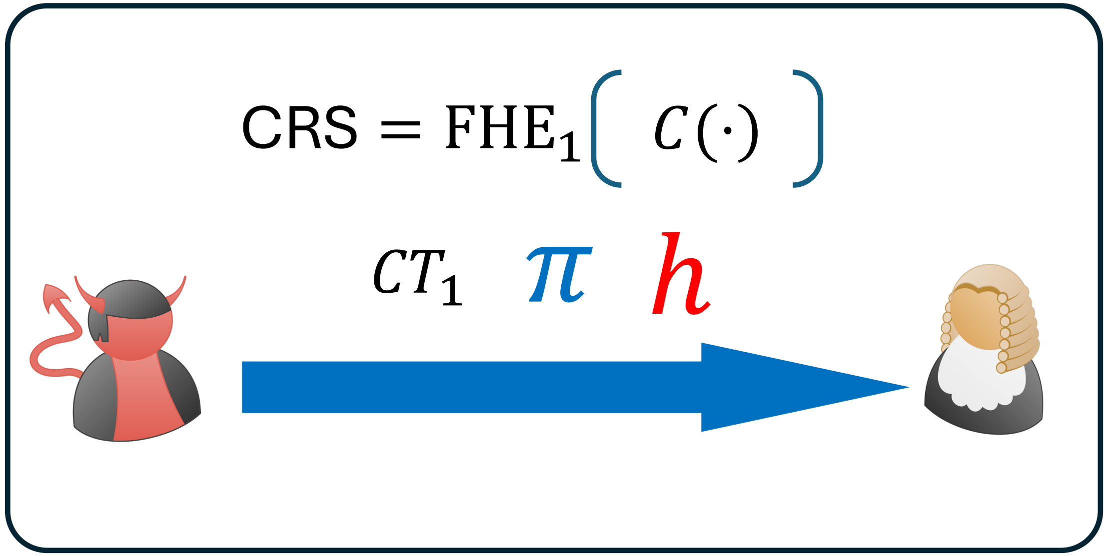
A Simplified View via Local Assignment Generator

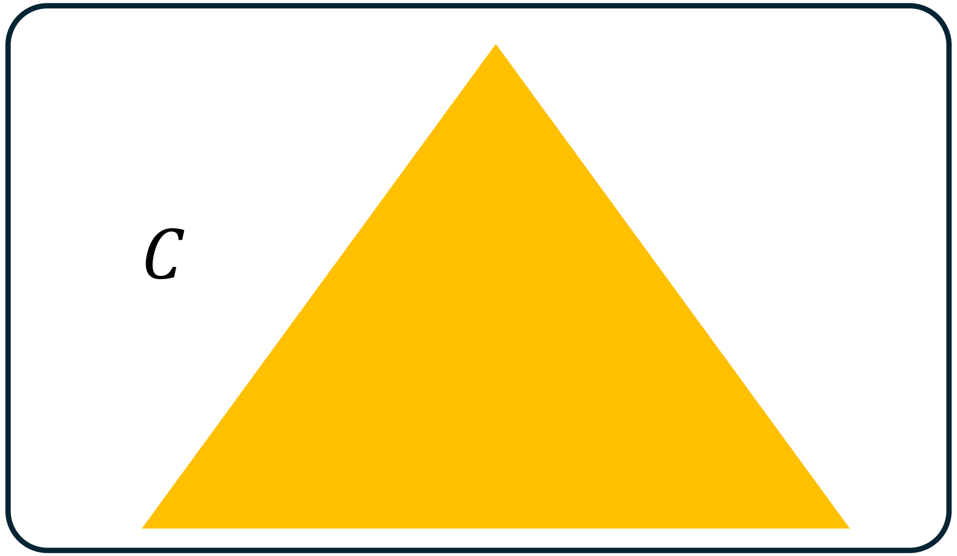
[BHK17]



A Simplified View via Local Assignment Generator

[BHK17]

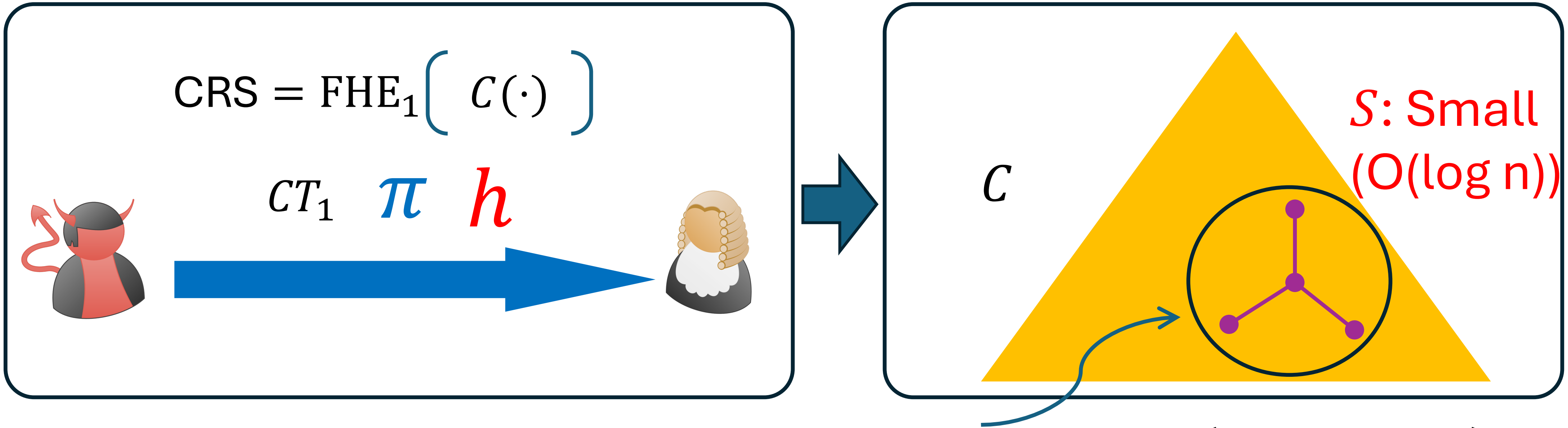




Local Assignment Generator: $\mathrm{Gen}(S) := \mathrm{FHE}_1.\mathrm{Dec}\big(sk, \mathrm{Ext}(\mathrm{td}, h)\big)$

A Simplified View via Local Assignment Generator

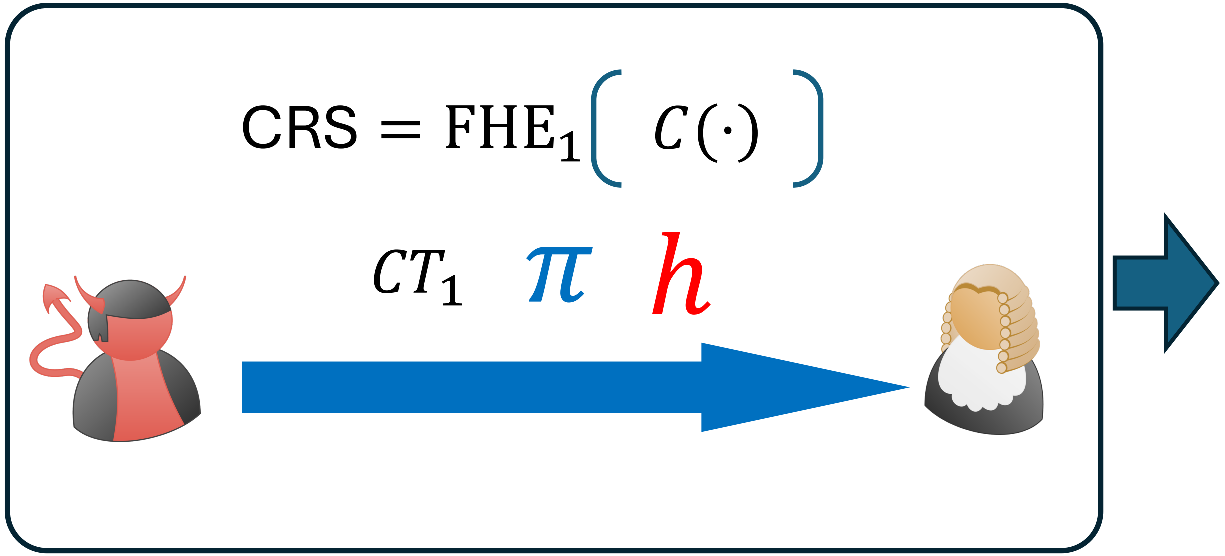
[BHK17]

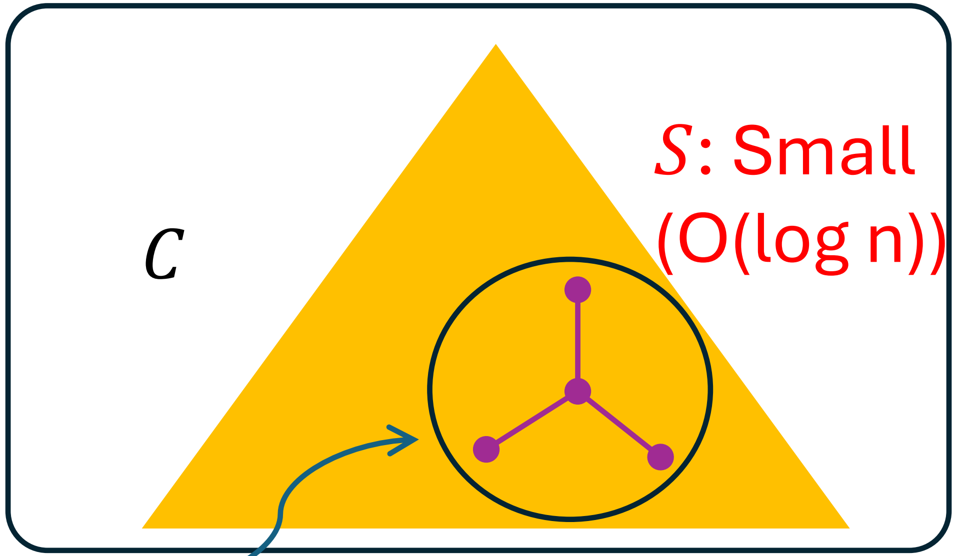


Local Assignment Generator: $\text{Gen}(S) \coloneqq \text{FHE}_1.\text{Dec}\big(sk, \text{Ext}(\text{td}, h)\big)$

A Simplified View via Local Assignment Generator

[BHK17]



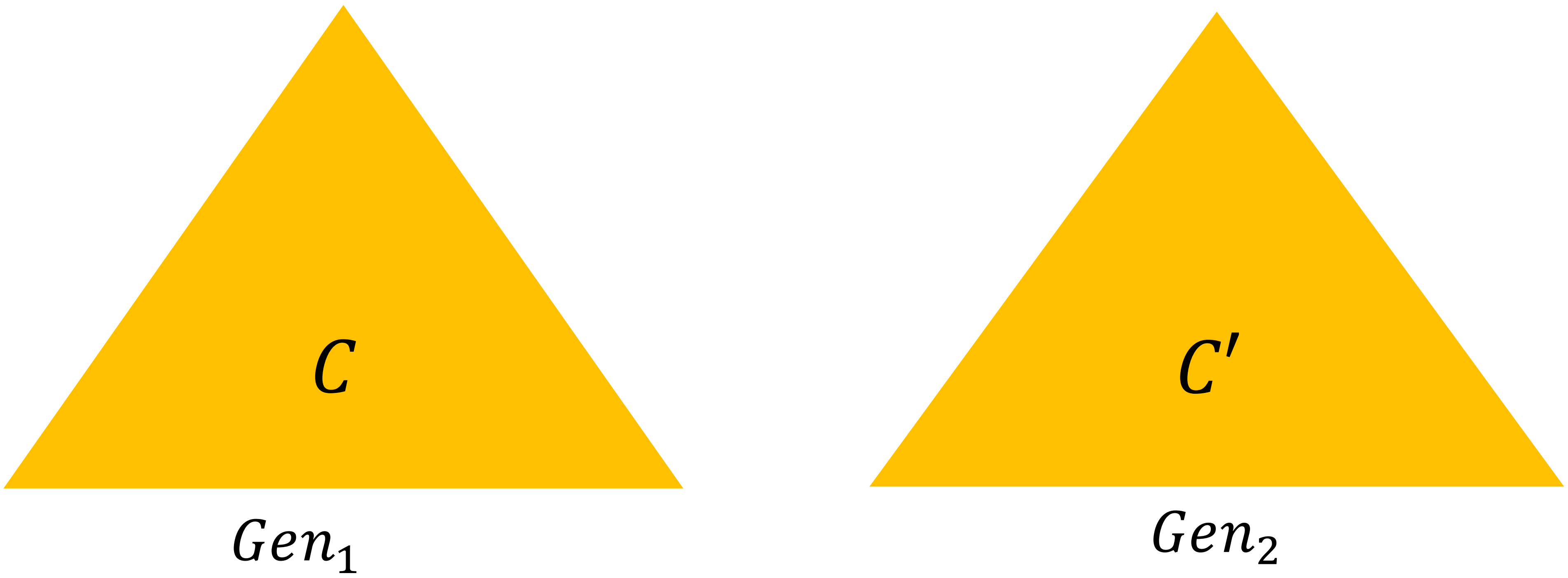


Local Assignment Generator: $\text{Gen}(S) \coloneqq \text{FHE}_1.\text{Dec}\big(sk, \text{Ext}(\text{td}, h)\big)$

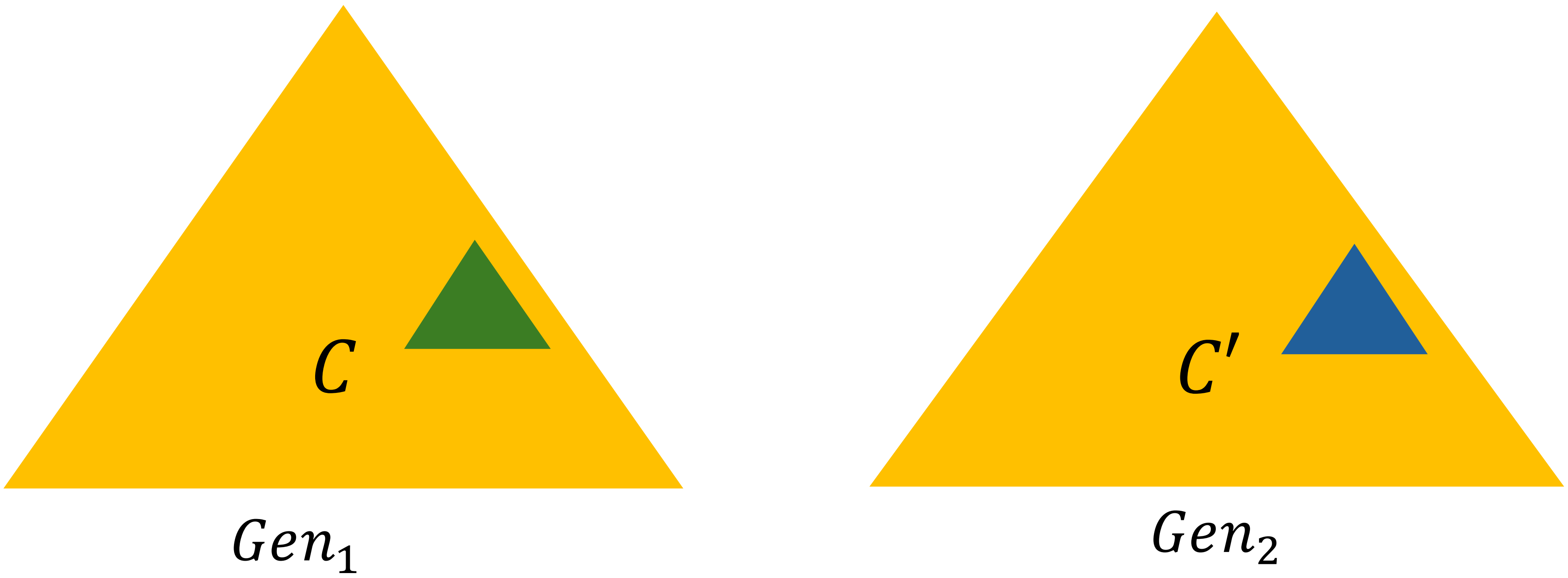
Properties of Gen

- No-Signaling (from No-Signaling property of SSB)
- Extracted wire values satisfy the gates in S (via Soundness of BARGs)

Puncturing Argument, Rephrased

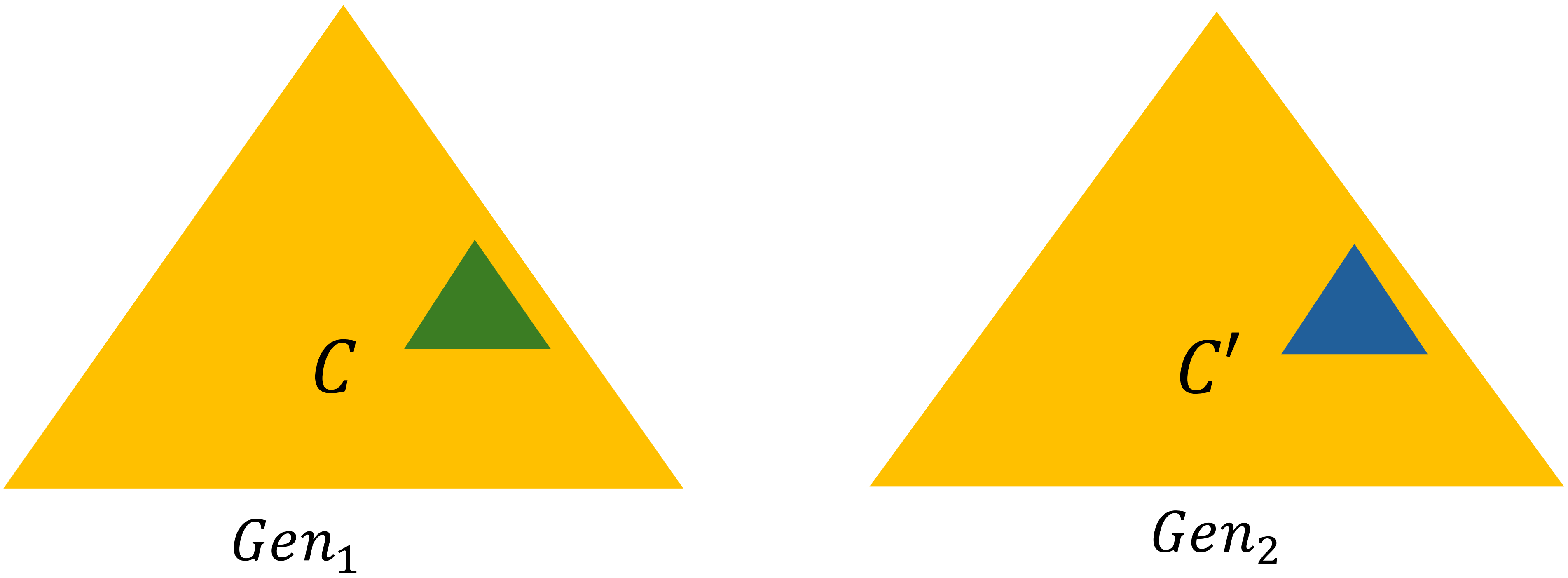


Puncturing Argument, Rephrased

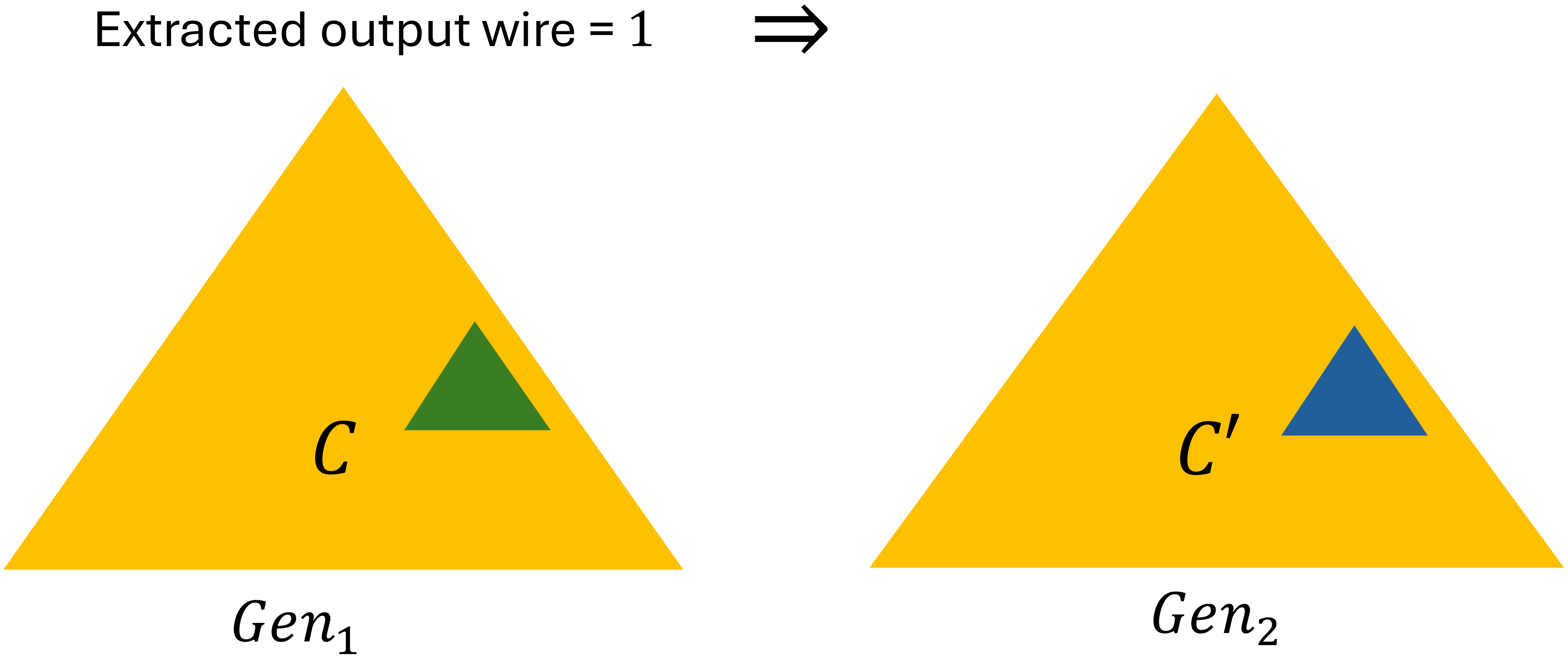


Puncturing Argument, Rephrased

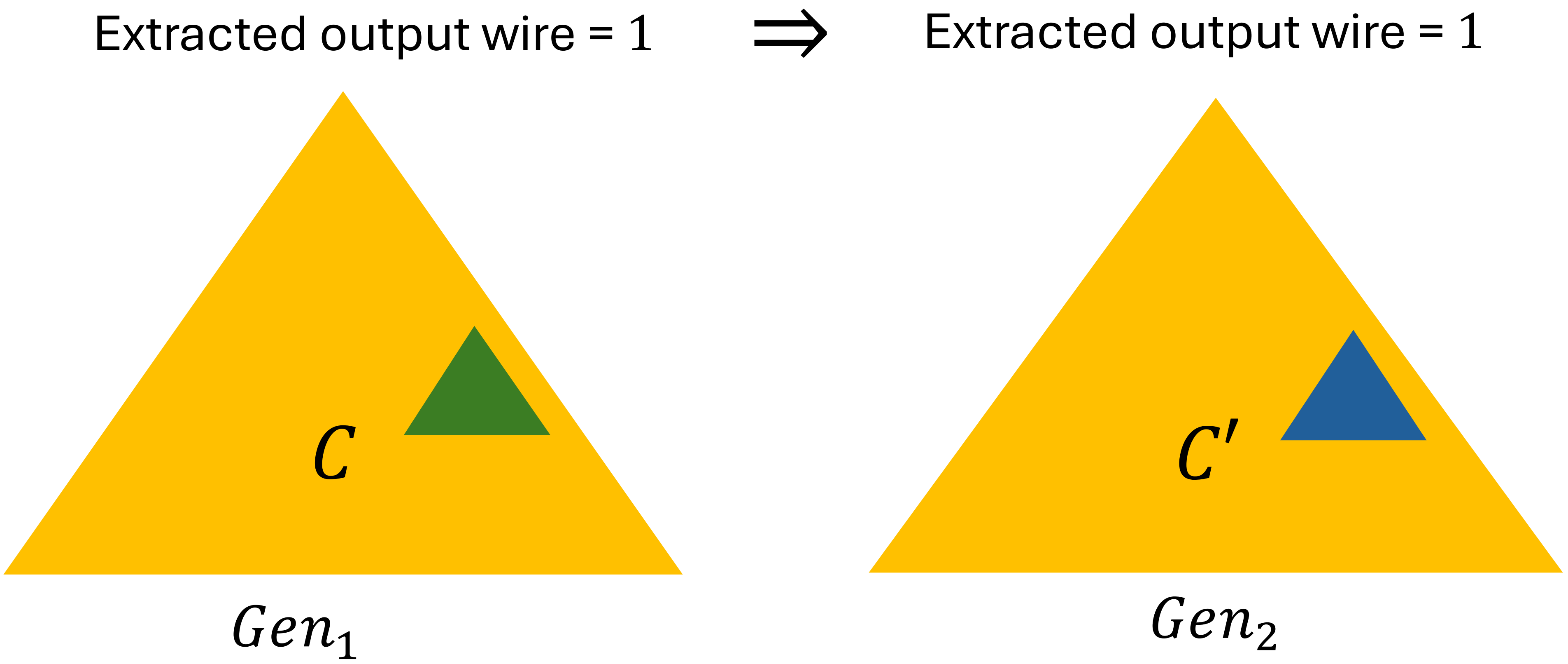
Extracted output wire = 1



Puncturing Argument, Rephrased



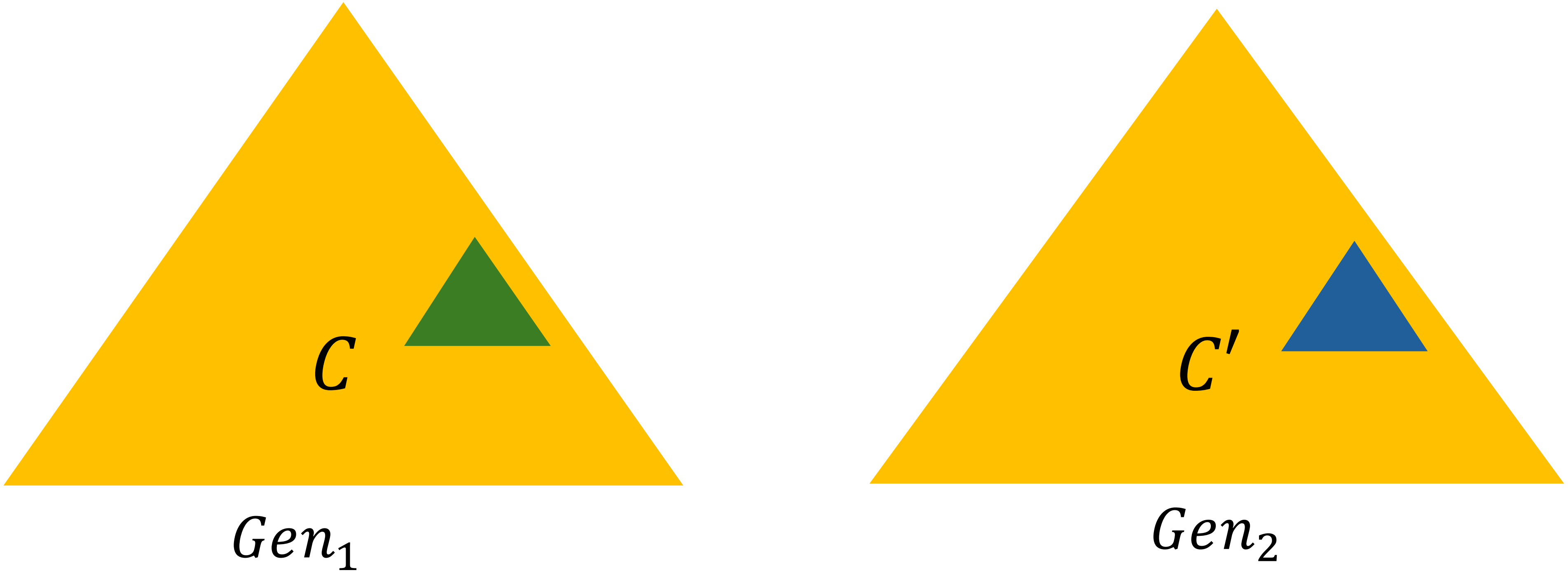
Puncturing Argument, Rephrased



Puncturing Argument, Rephrased

We want to prove

Extracted output wire = 1 $\Rightarrow$ Extracted output wire = 1

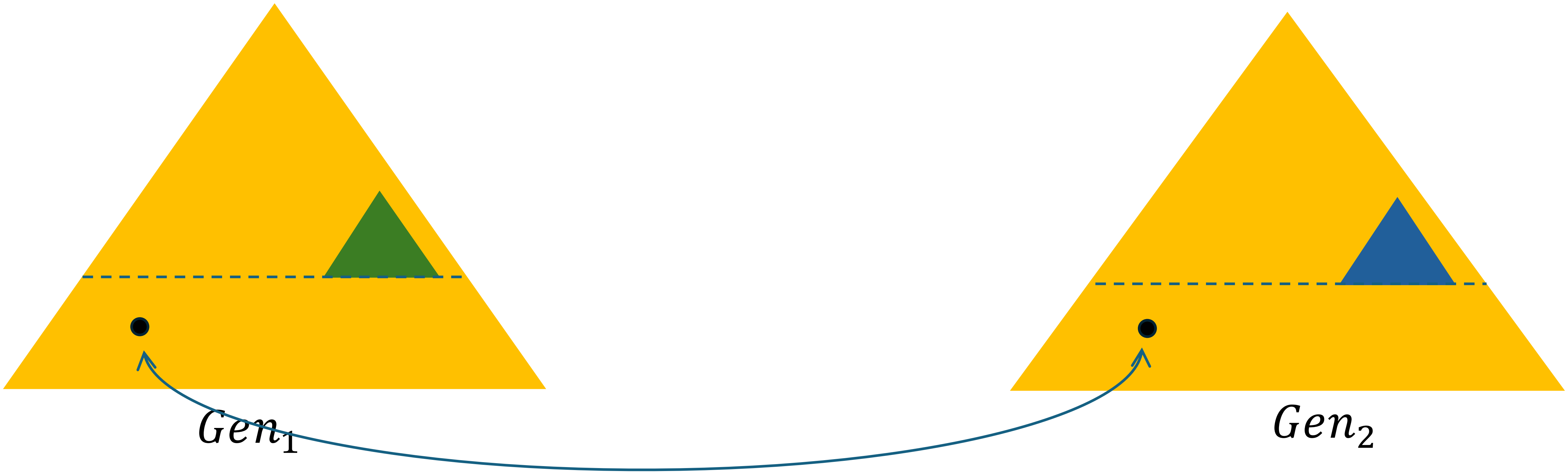


Blueprint of the Proof

Blueprint of the Proof

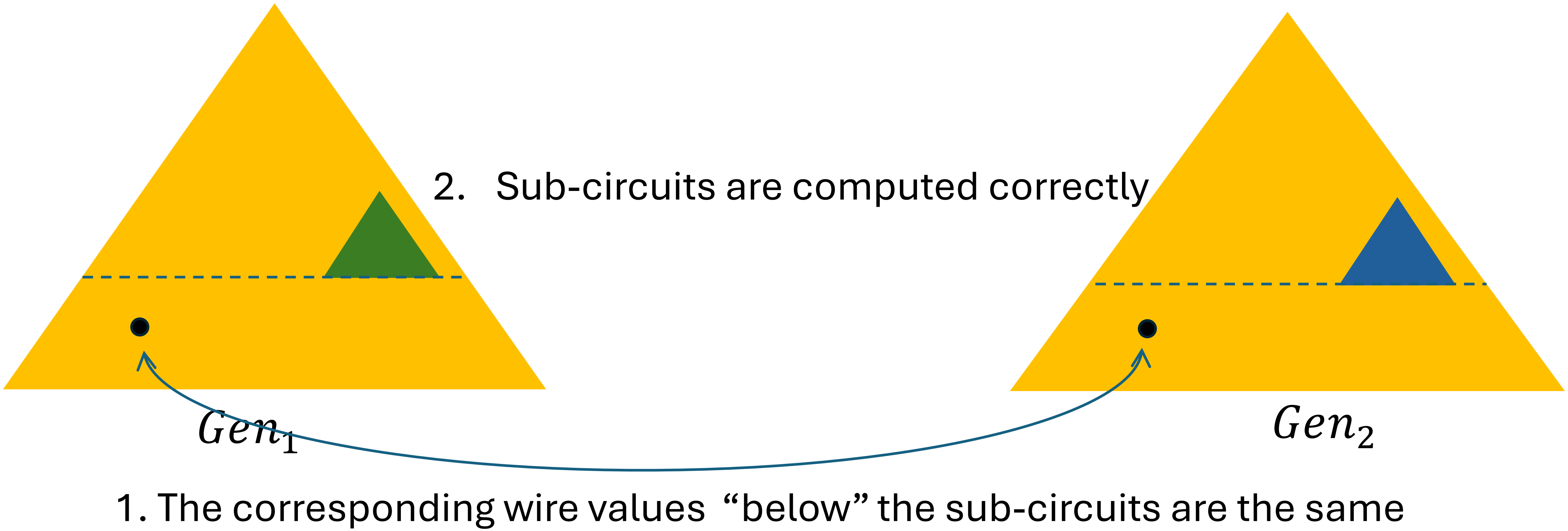
Gen_1 Gen_2

Blueprint of the Proof



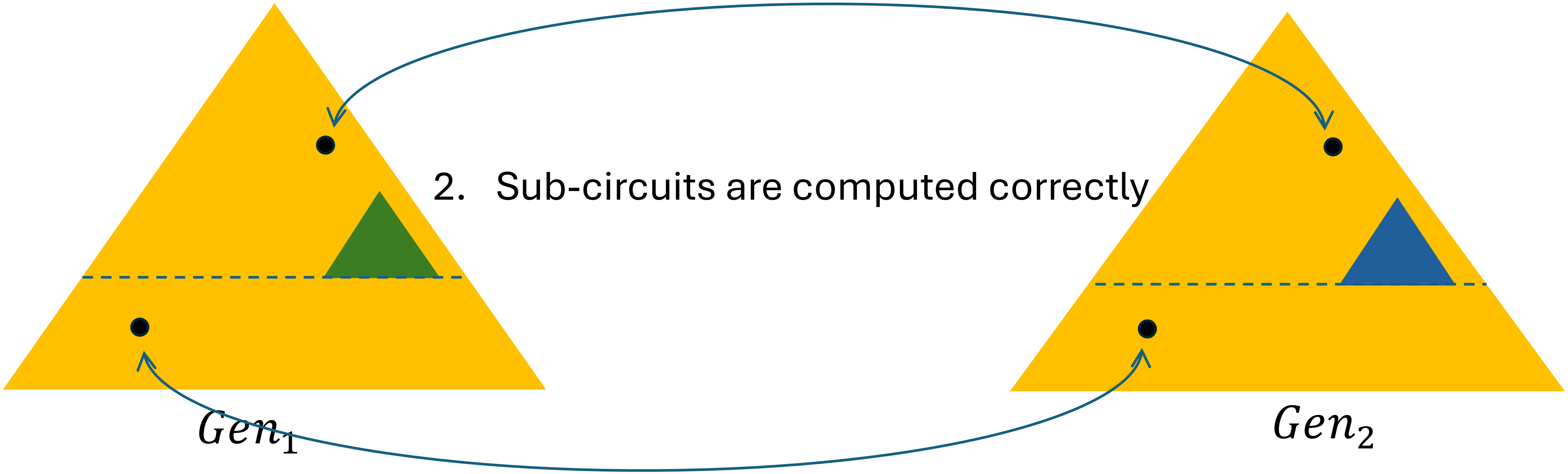
1. The corresponding wire values “below” the sub-circuits are the same

Blueprint of the Proof



Blueprint of the Proof

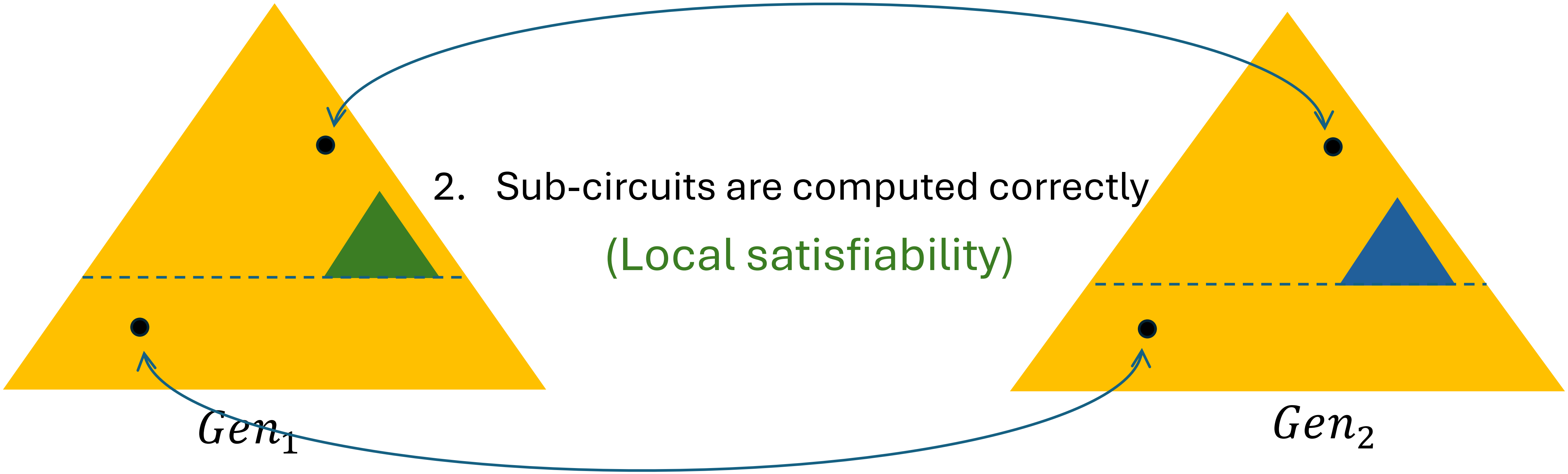
3. The corresponding wire values “above” the sub-circuits are the same.



1. The corresponding wire values “below” the sub-circuits are the same

Blueprint of the Proof

3. The corresponding wire values “above” the sub-circuits are the same.

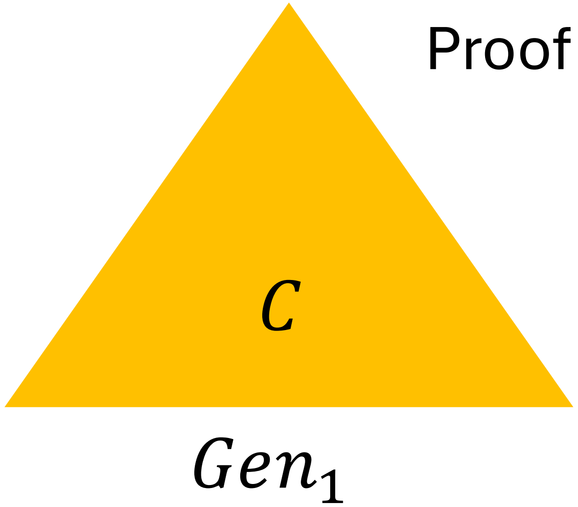


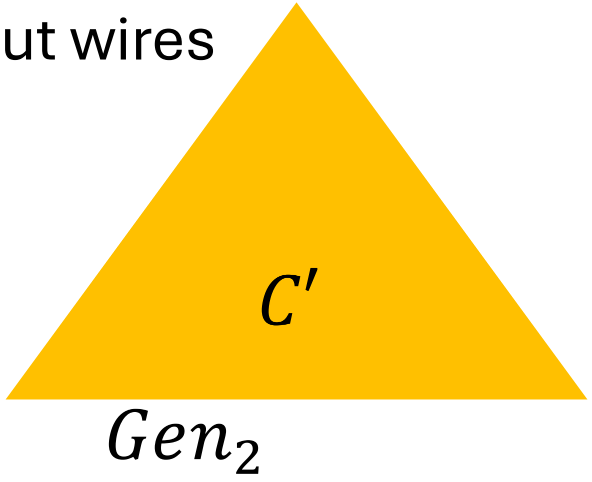
1. The corresponding wire values “below” the sub-circuits are the same

Proof of Puncturing Argument: Blow the Subckt

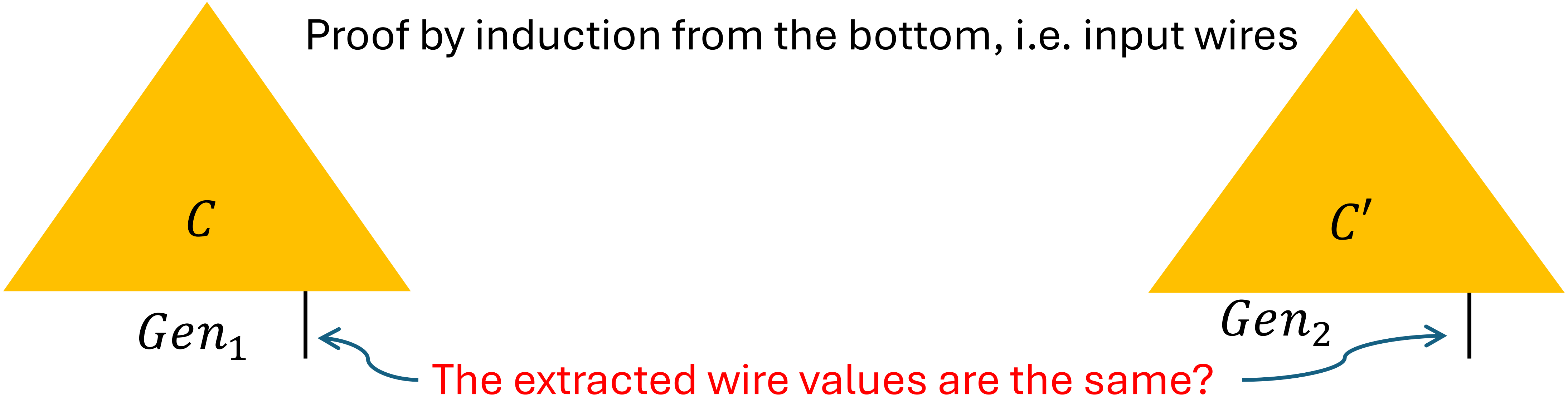
Proof of Puncturing Argument: Blow the Subckt

Proof by induction from the bottom, i.e. input wires



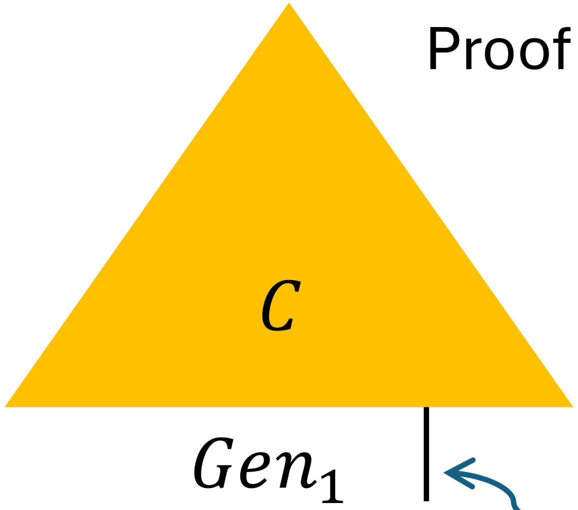


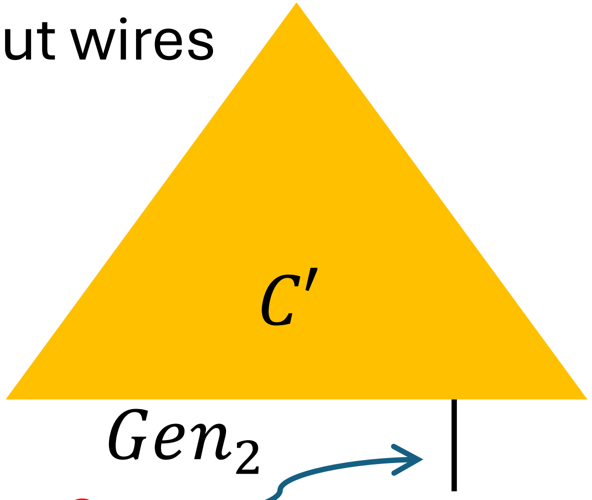
Proof of Puncturing Argument: Blow the Subckt



Proof of Puncturing Argument: Blow the Subckt

Proof by induction from the bottom, i.e. input wires





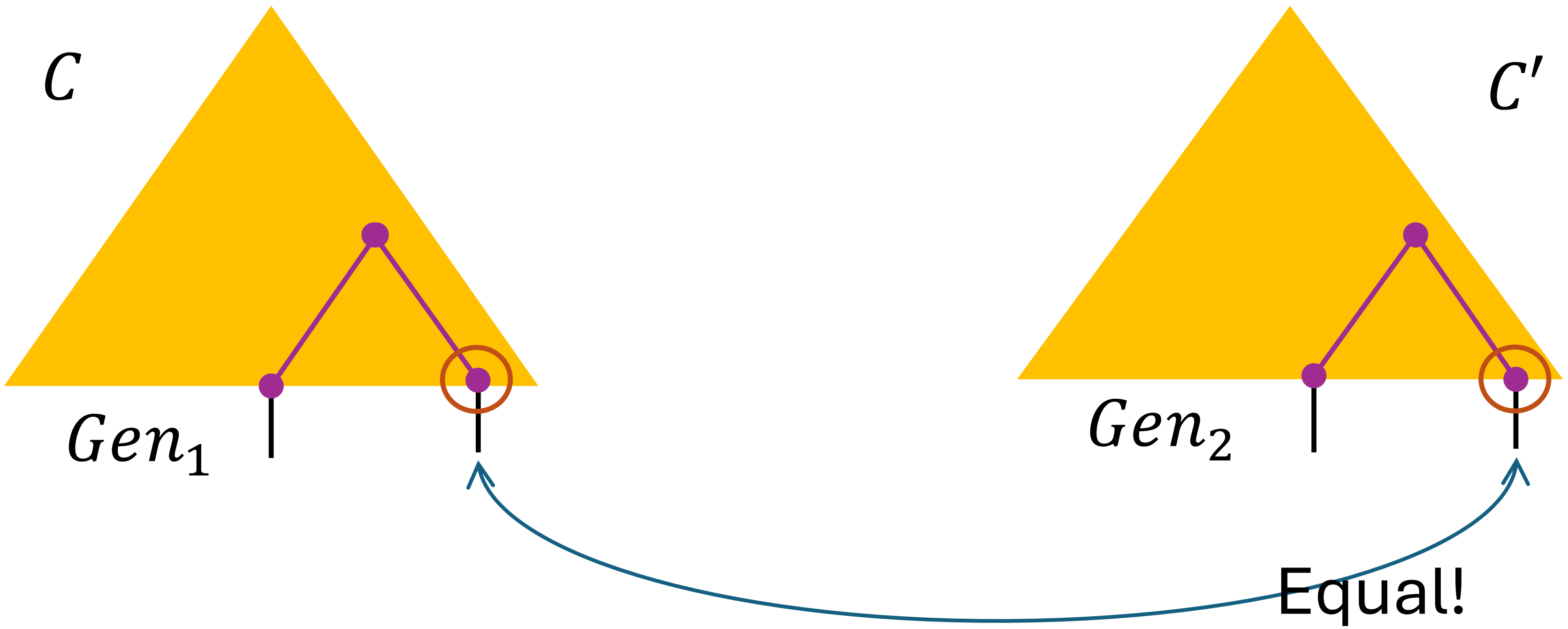
The extracted wire values are the same?

Add Statements that BARG Proves: $\forall i,\ \exists$ local openings of the i-th input wire w.r.t. h_1, h_2, and the wire values $w_i, {w_i}'$ where $w_i = {w_i}'$.

Moving Up in the Layers

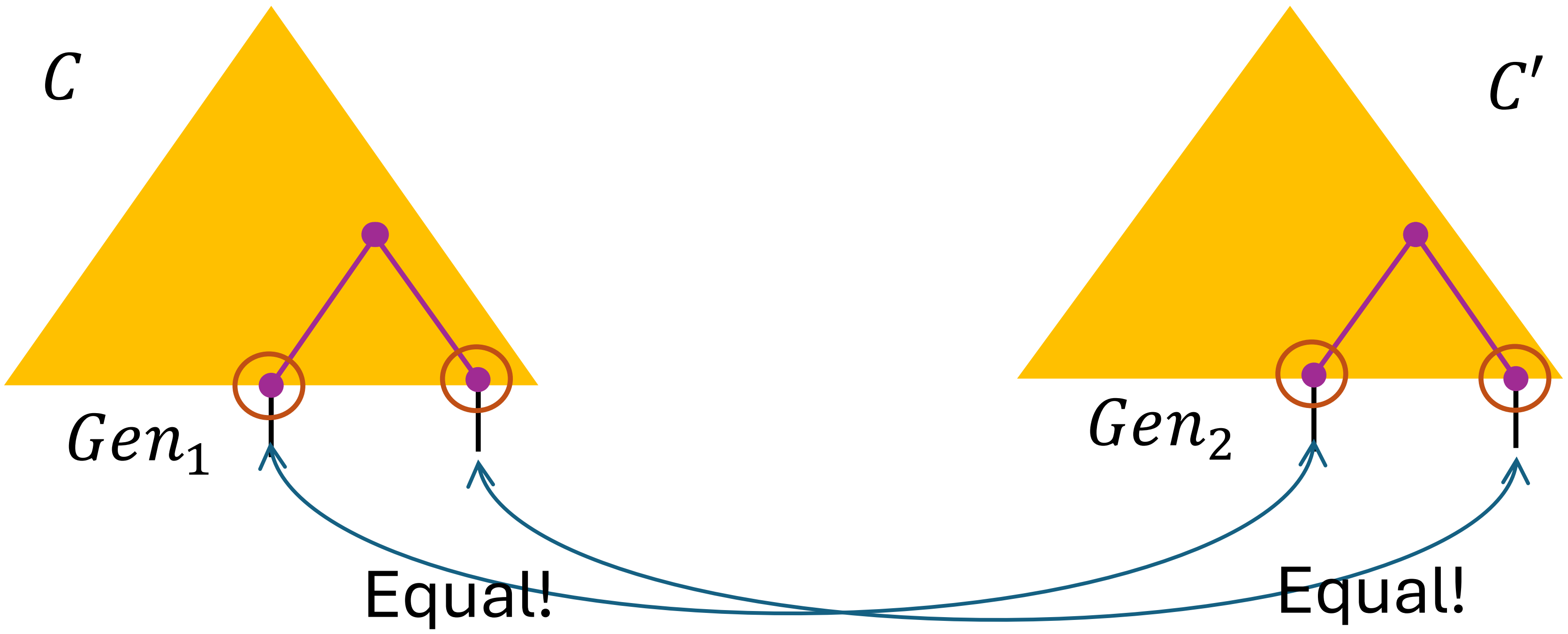
: extracted gate outputs in local assignment generators

Moving Up in the Layers



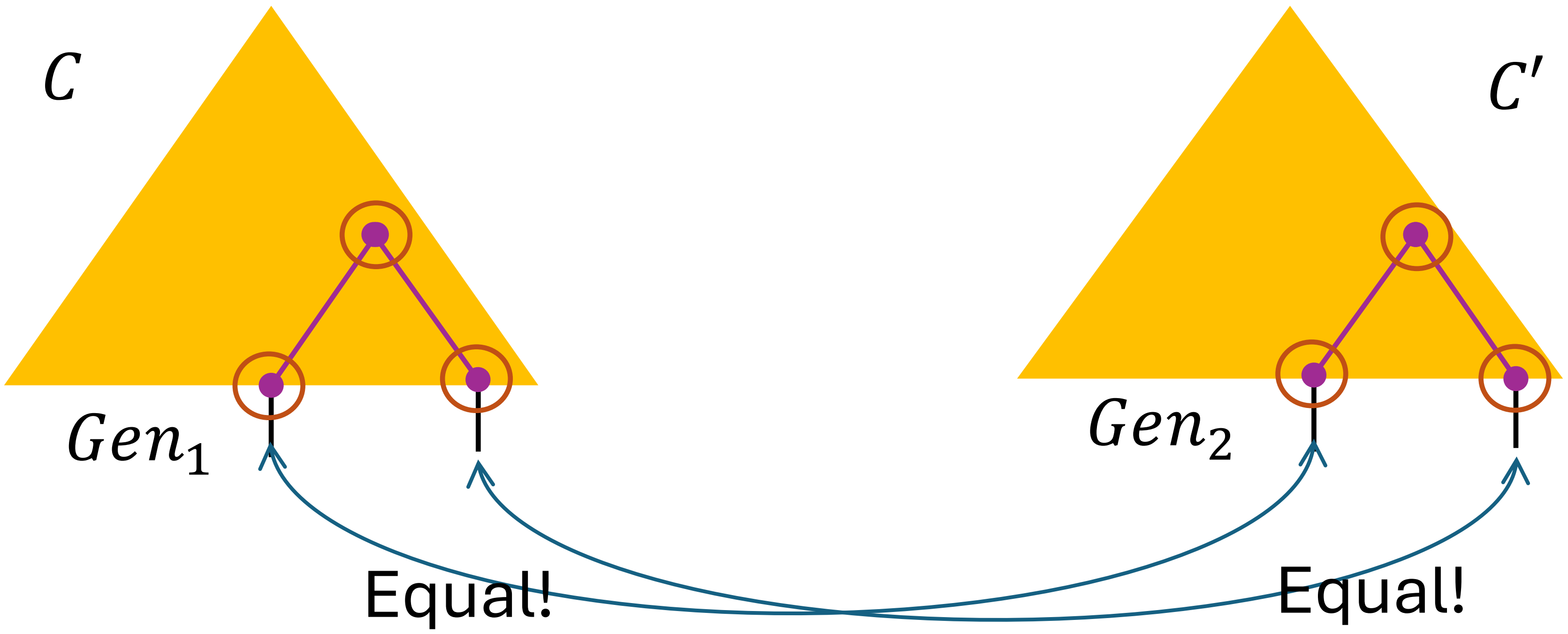
: extracted gate outputs in local assignment generators

Moving Up in the Layers



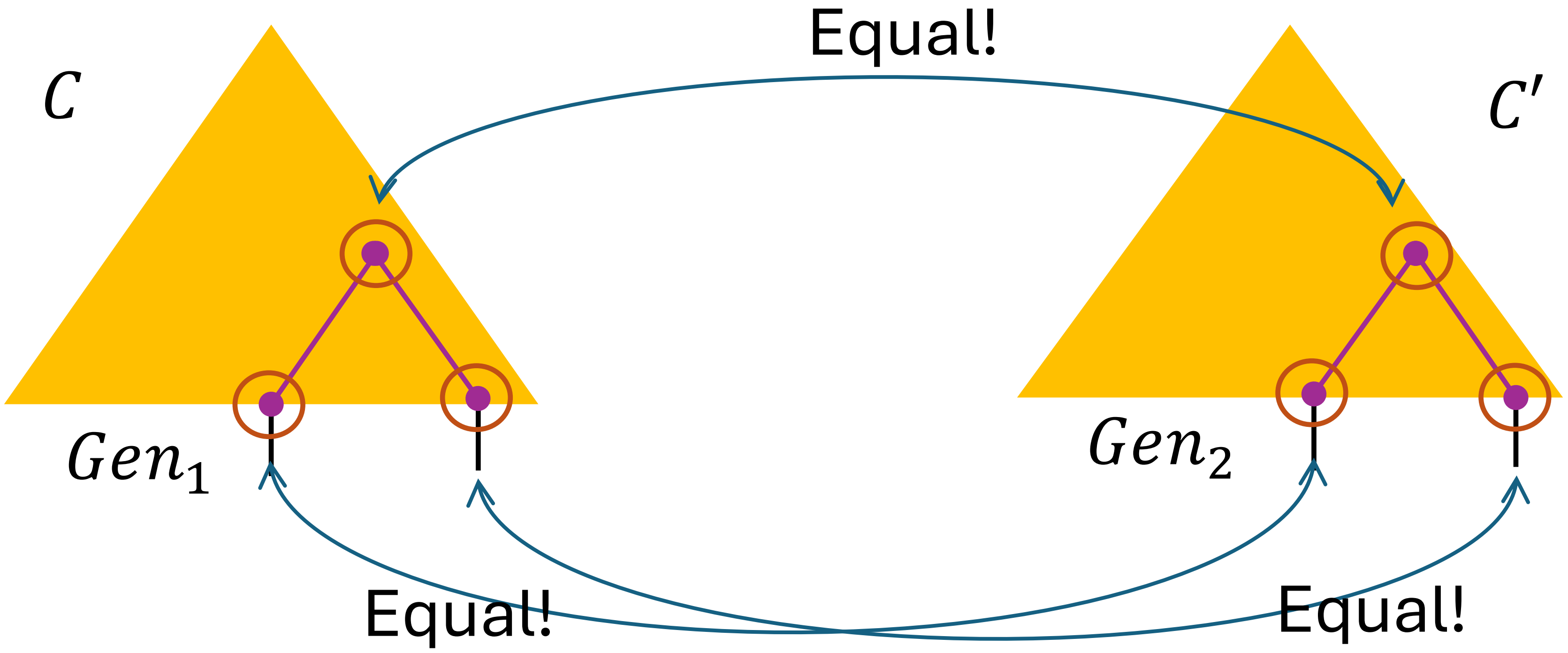
: extracted gate outputs in local assignment generators

Moving Up in the Layers



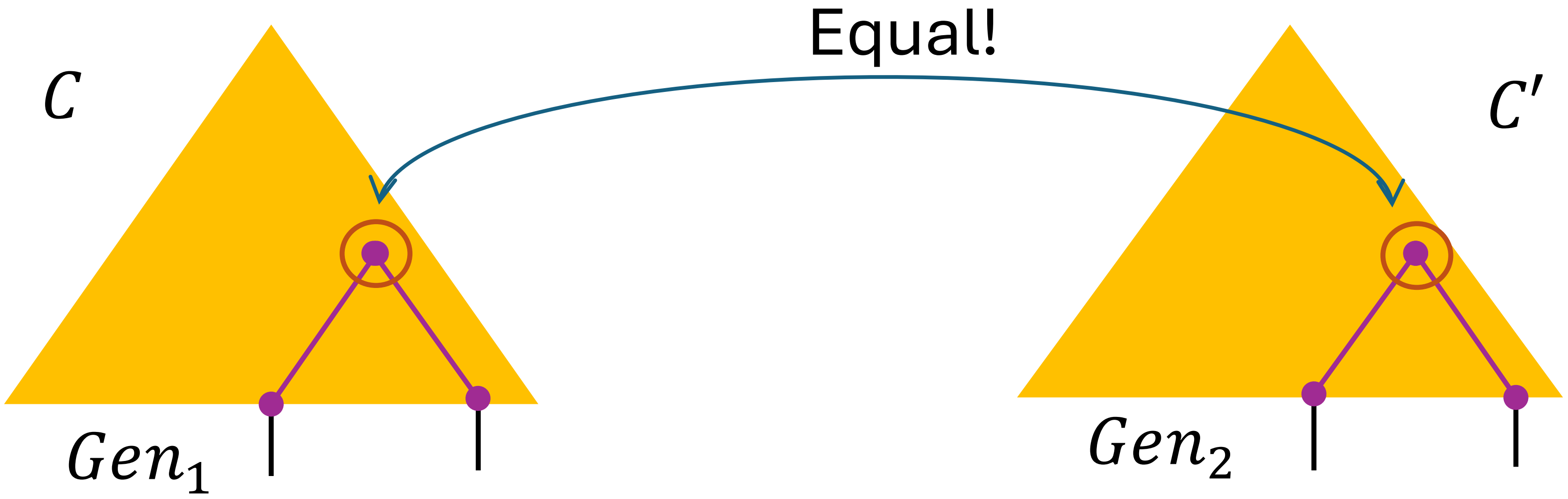
: extracted gate outputs in local assignment generators

Moving Up in the Layers



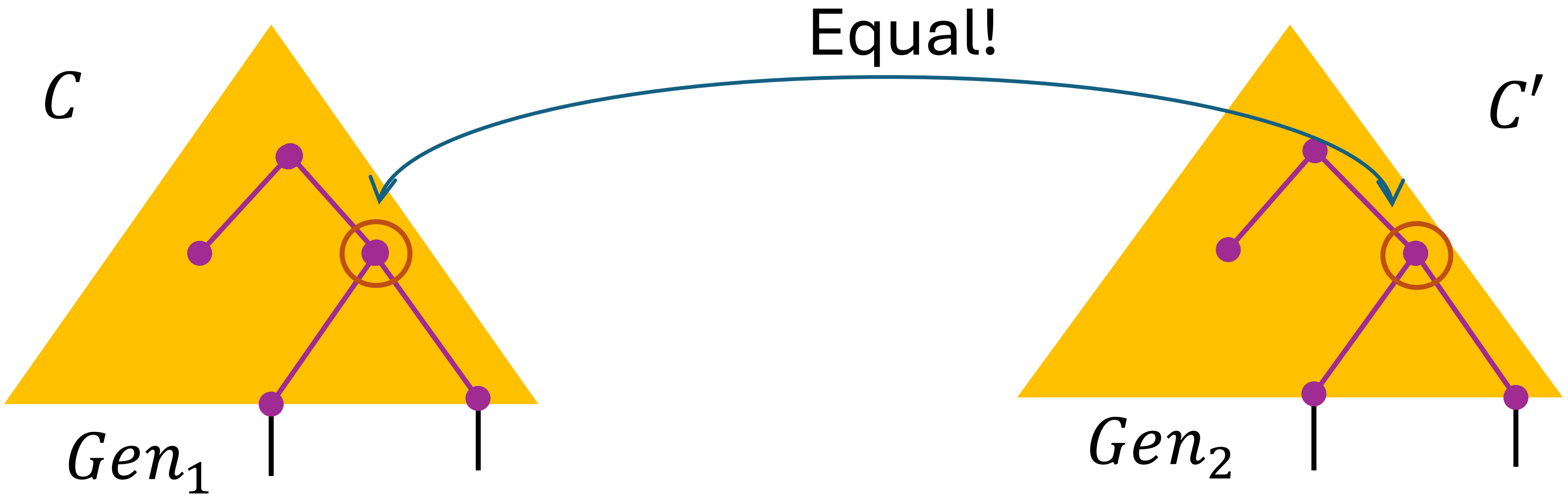
: extracted gate outputs in local assignment generators

Moving Up in the Layers



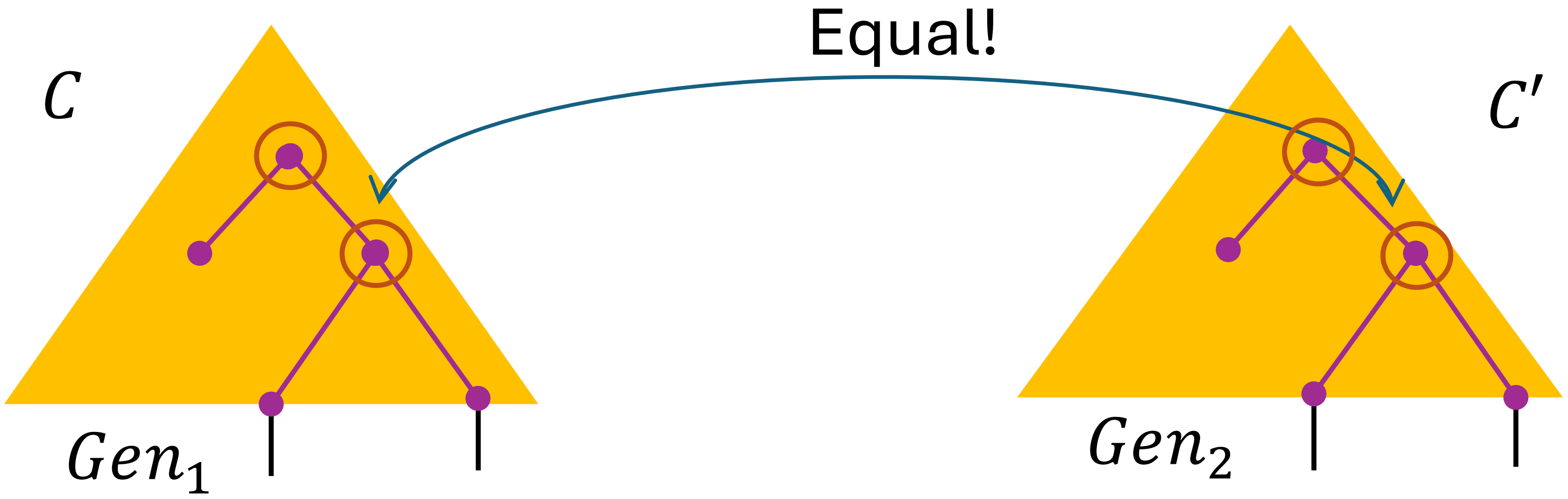
: extracted gate outputs in local assignment generators

Moving Up in the Layers



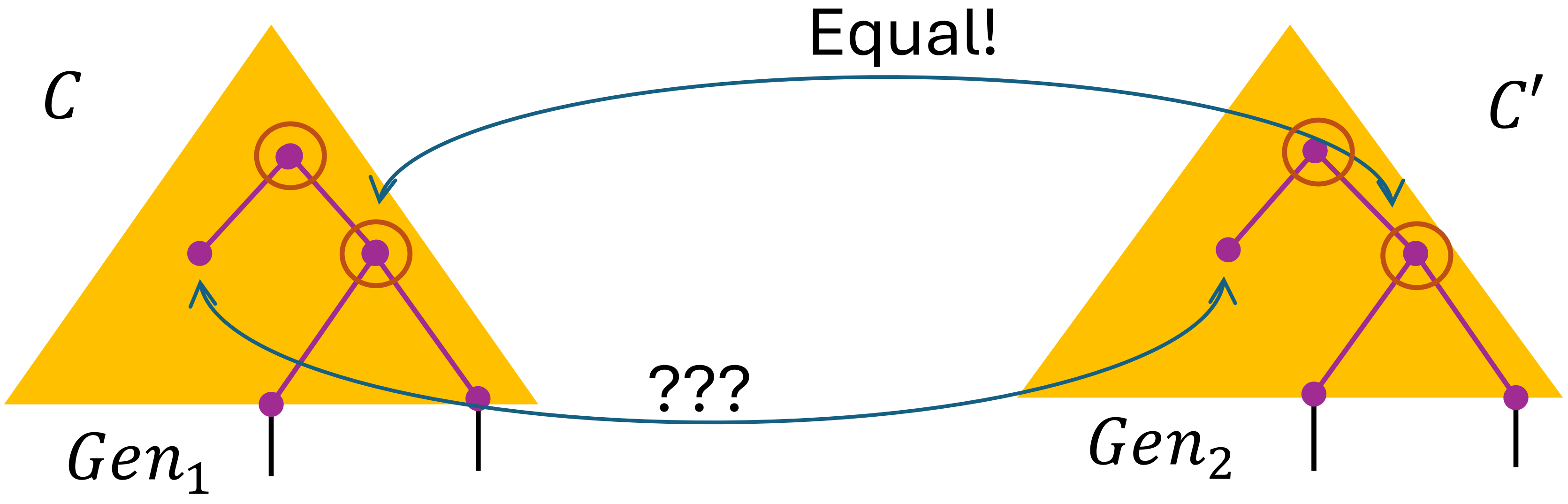
: extracted gate outputs in local assignment generators

Moving Up in the Layers



: extracted gate outputs in local assignment generators

Moving Up in the Layers

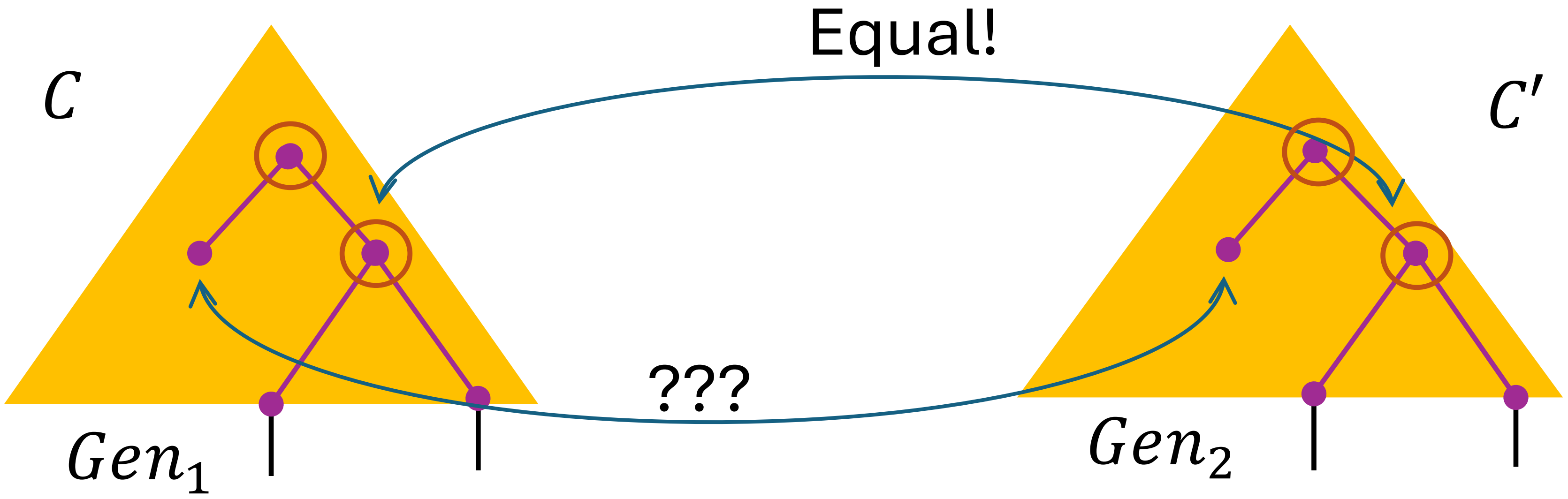


: extracted gate outputs in local assignment generators

Moving Up in the Layers

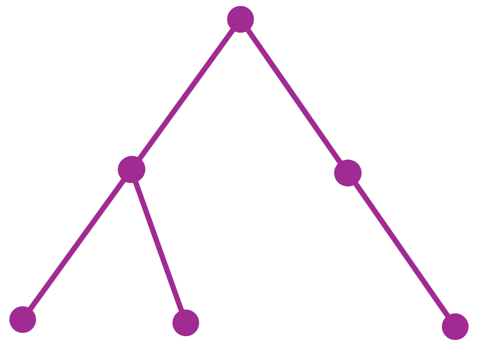
We need to extract their children first…

and the children of their children first…

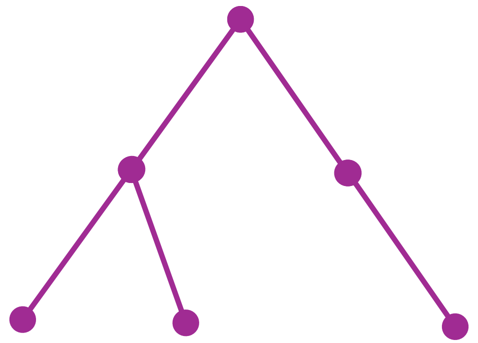


: extracted gate outputs in local assignment generators

Abstracting as a Pebbling Game

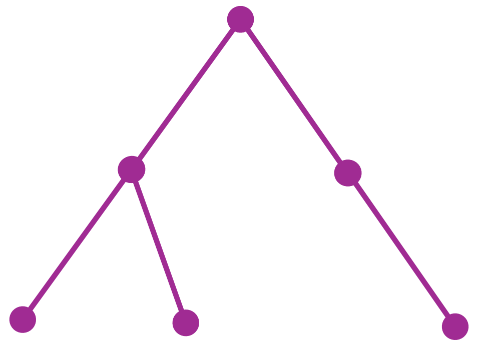


Abstracting as a Pebbling Game



Pebbles: ◯ the extracted gates in local assignment generator

Abstracting as a Pebbling Game

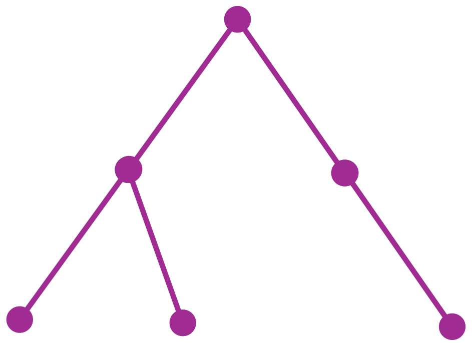


Pebbles: ◯ the extracted gates in local assignment generator

Rules:

- Place a pebble at an input wire
- Place a pebble if both children is pebbled
- Free to delete a pebble

Abstracting as a Pebbling Game

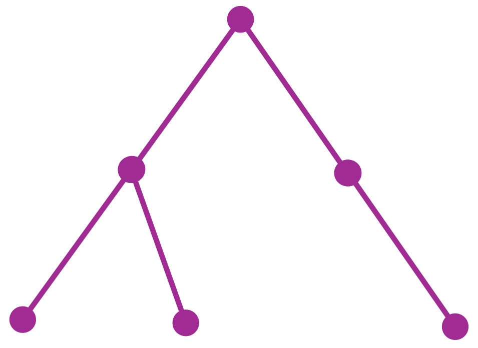


Pebbles: the extracted gates in local assignment generator

Rules:

- Place a pebble at an input wire
- Place a pebble if both children is pebbled
- Free to delete a pebble

Abstracting as a Pebbling Game

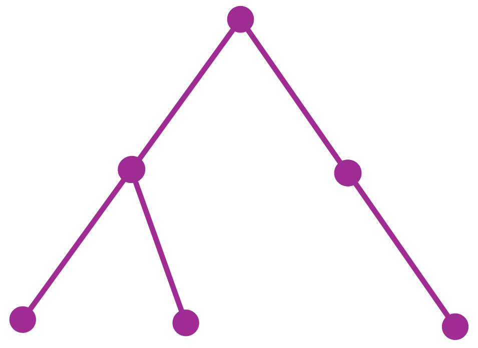


Pebbles: ◯ the extracted gates in local assignment generator

Rules:

- Place a pebble at an input wire
- Place a pebble if both children is pebbled
- Free to delete a pebble

Abstracting as a Pebbling Game

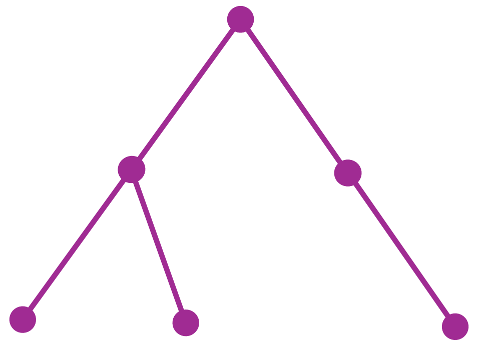


Pebbles: the extracted gates in local assignment generator

Rules:

- Place a pebble at an input wire
- Place a pebble if both children is pebbled
- Free to delete a pebble

Abstracting as a Pebbling Game



Pebbles: the extracted gates in local assignment generator

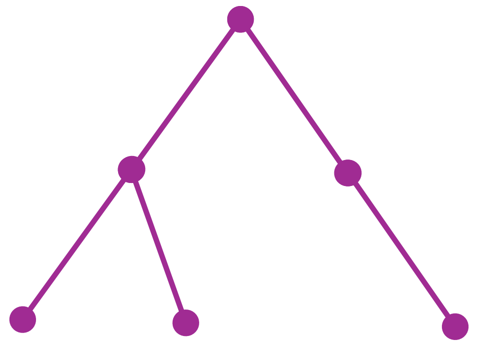
Rules:

- Place a pebble at an input wire
- Place a pebble if both children is pebbled
- Free to delete a pebble

Seems need 2^{depth} moves in general if we only have $O(\log n)$ pebbles

Abstracting as a Pebbling Game

Pebbles: the extracted gates in local assignment generator



Rules:

- Place a pebble at an input wire
- Place a pebble if both children is pebbled
- Free to delete a pebble

Seems need 2^{depth} moves in general if we only have $O(\log n)$ pebbles

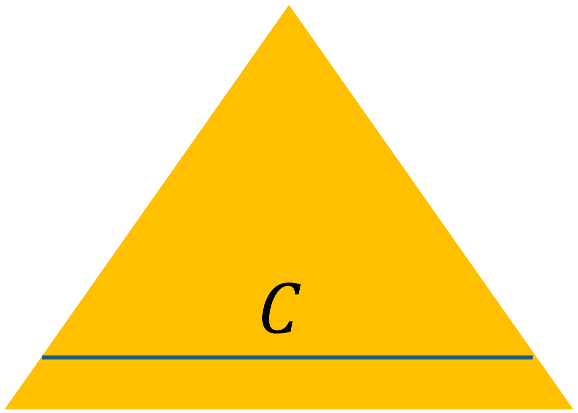
Reduction time: 2^{depth}! More efficient reduction?

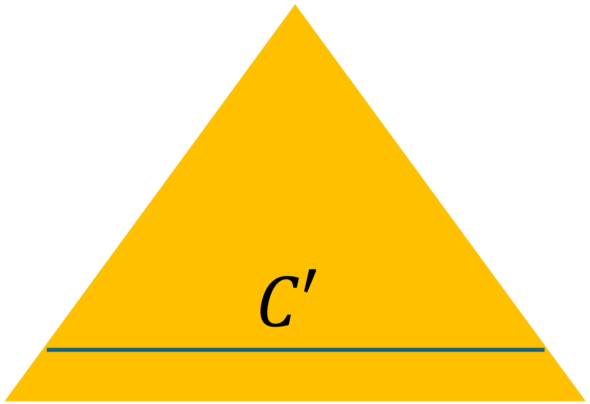
Augmenting Ckts with Collision-Resistance Hashes

: extracted gate values in local assignment generators

Augmenting Ckts with Collision-Resistance Hashes

Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)

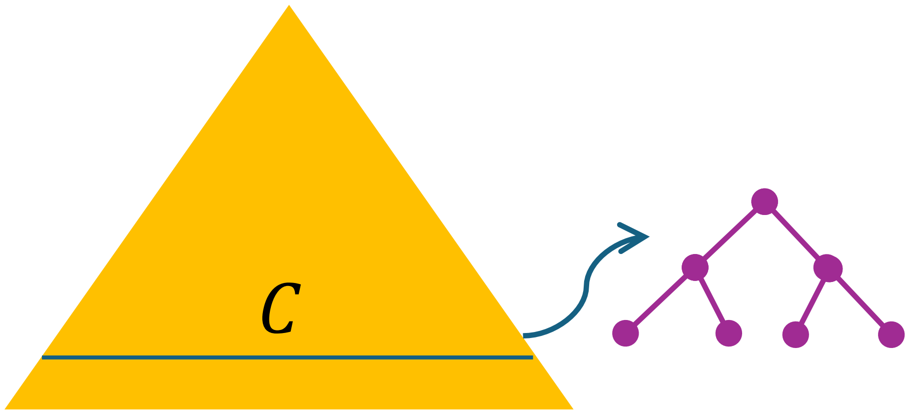


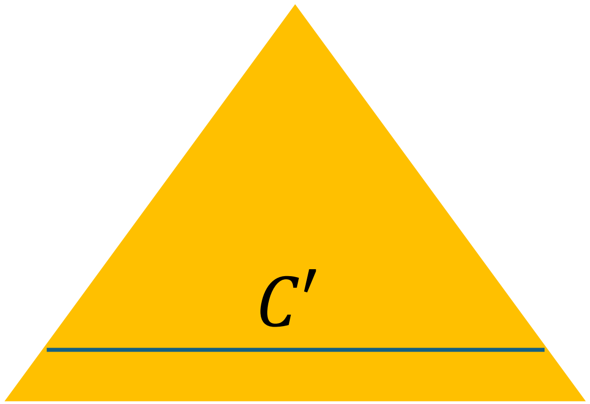


: extracted gate values in local assignment generators

Augmenting Ckts with Collision-Resistance Hashes

Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)

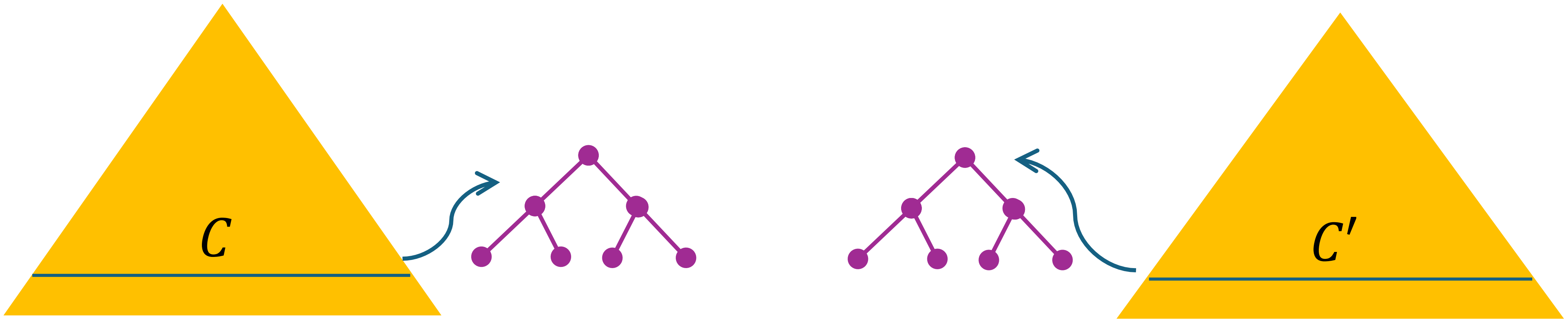




: extracted gate values in local assignment generators

Augmenting Ckts with Collision-Resistance Hashes

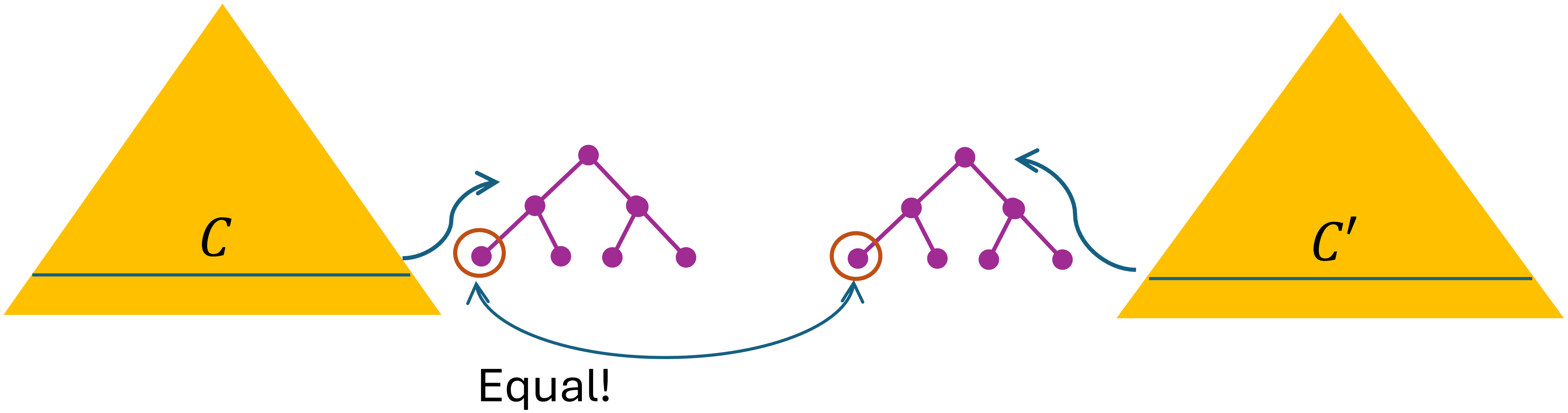
Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)



: extracted gate values in local assignment generators

Augmenting Ckts with Collision-Resistance Hashes

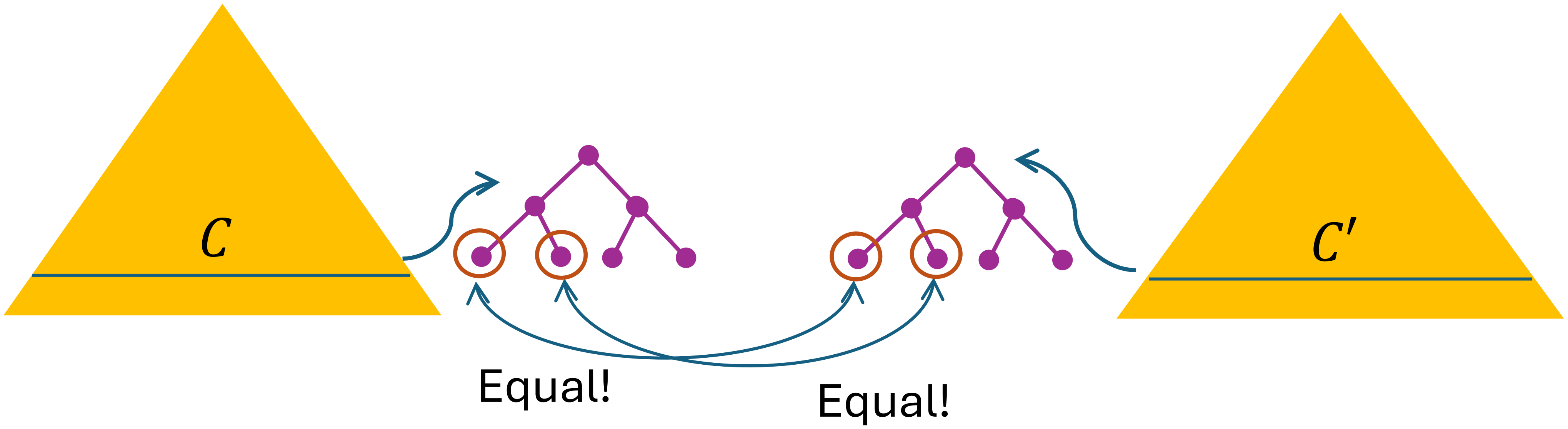
Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)



: extracted gate values in local assignment generators

Augmenting Ckts with Collision-Resistance Hashes

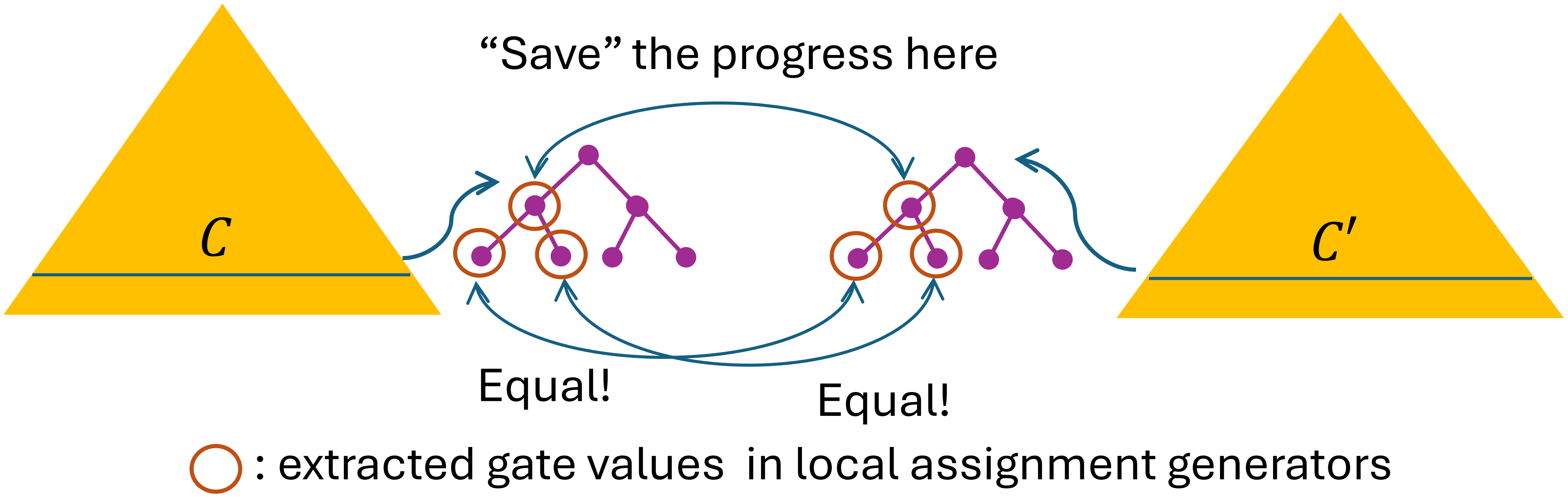
Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)



: extracted gate values in local assignment generators

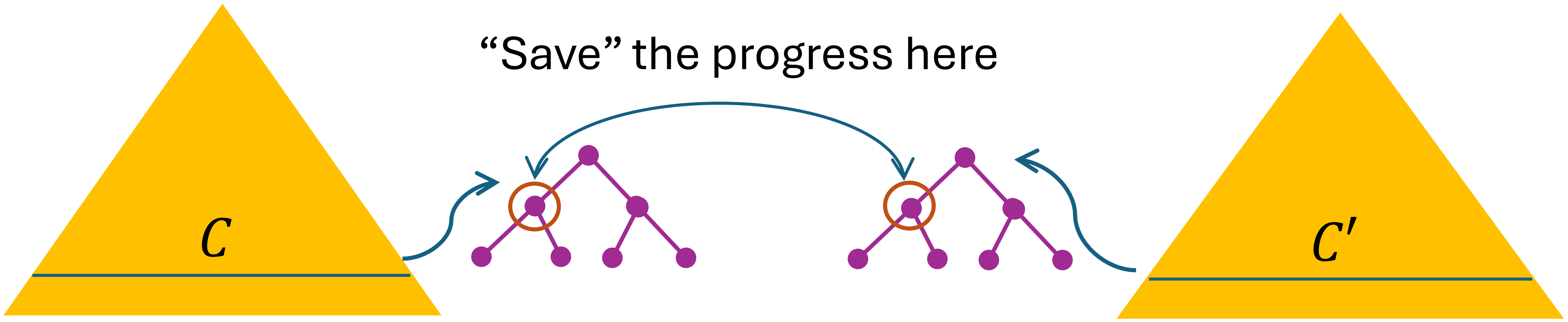
Augmenting Ckts with Collision-Resistance Hashes

Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)



Augmenting Ckts with Collision-Resistance Hashes

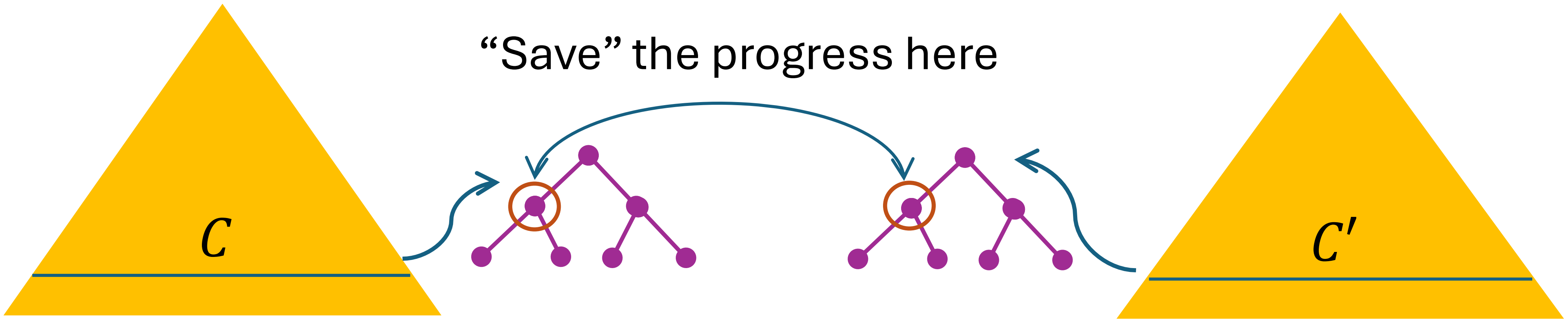
Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)



: extracted gate values in local assignment generators

Augmenting Ckts with Collision-Resistance Hashes

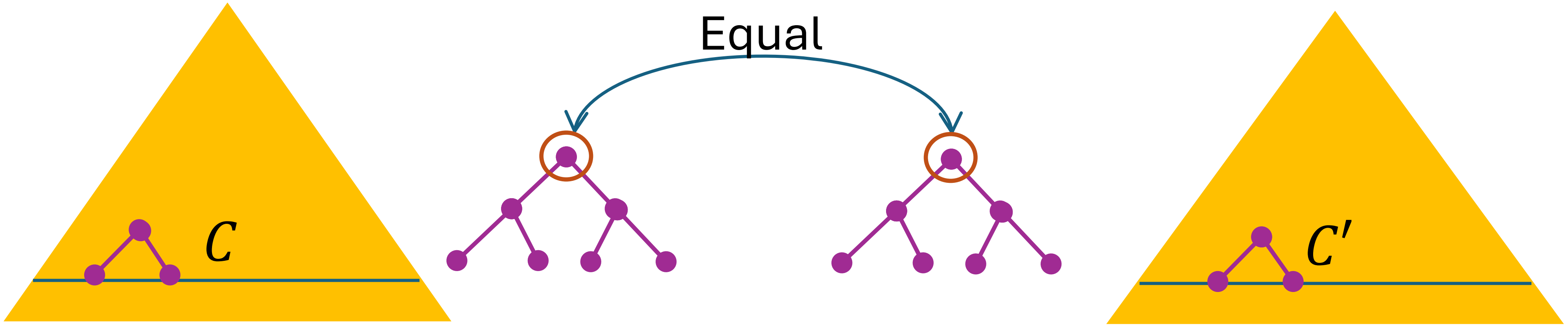
Merkle hash the gate values at each layer via Collision-Resistance Hash (CRHF)



: extracted gate values in local assignment generators

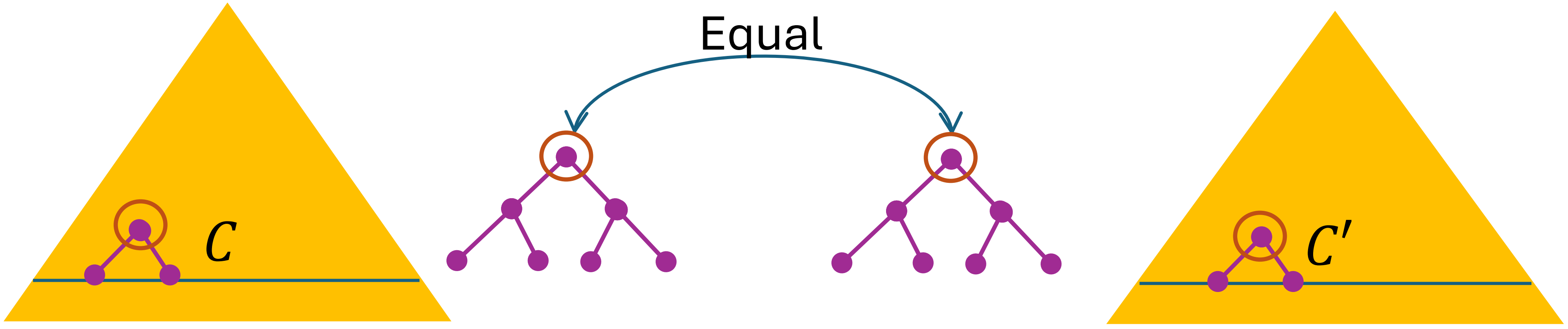
The entire layer is “equal” $\Longleftrightarrow$ the roots of CRHF trees are “equal”.

Moving Up in the Layers



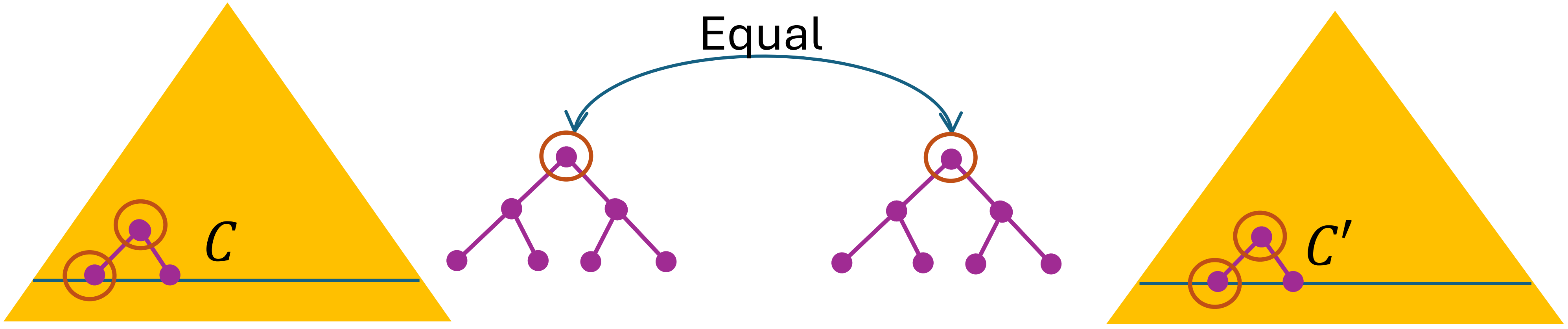
: extracted gate values in local assignment generators

Moving Up in the Layers



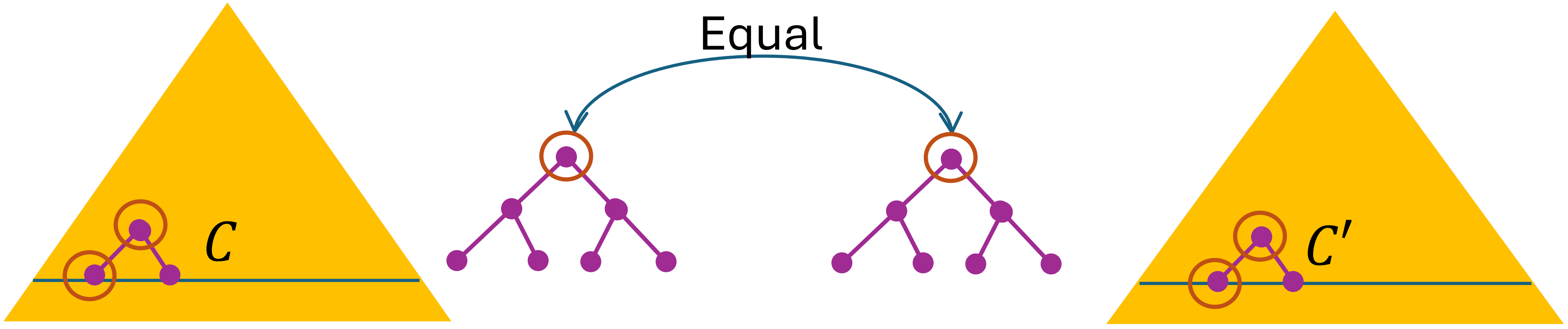
: extracted gate values in local assignment generators

Moving Up in the Layers



: extracted gate values in local assignment generators

Moving Up in the Layers

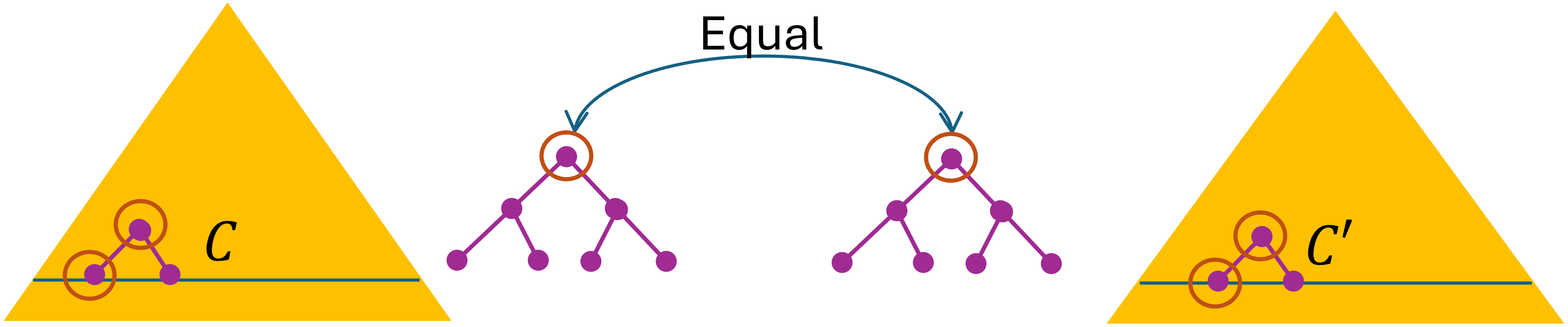


Suppose the left child is the 2nd node in the layer

: extracted gate values in local assignment generators

Moving Up in the Layers

"Load" the saved progress from the roots

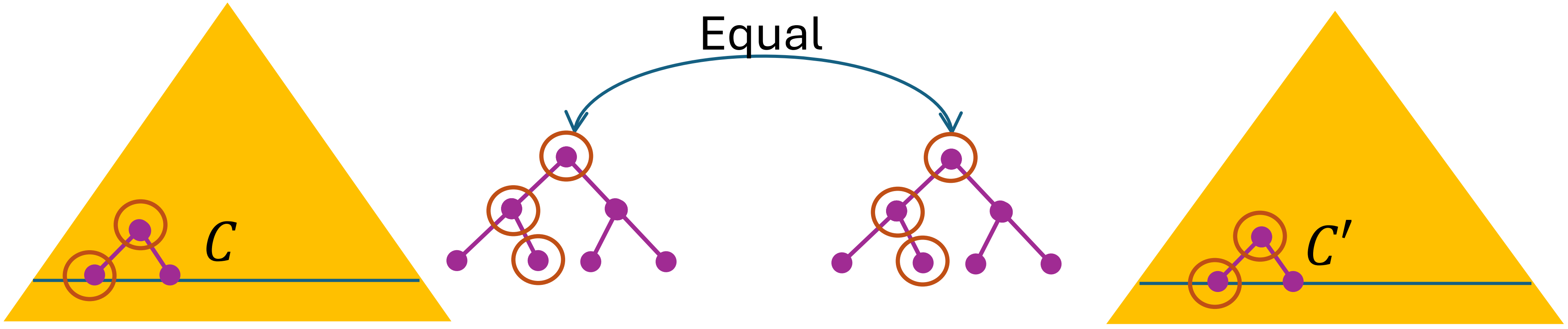


Suppose the left child is the 2nd node in the layer

: extracted gate values in local assignment generators

Moving Up in the Layers

"Load" the saved progress from the roots

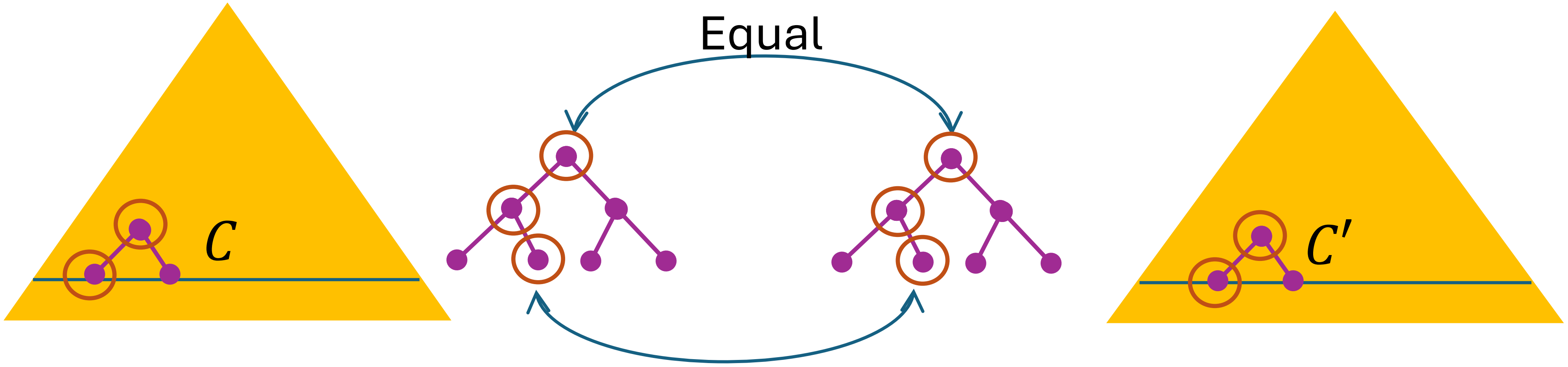


Suppose the left child is the 2nd node in the layer

 : extracted gate values in local assignment generators

Moving Up in the Layers

"Load" the saved progress from the roots



Suppose the left child is the 2nd node in the layer

◯ : extracted gate values in local assignment generators

Save the Progress in the Middle of the Subckt

◯ : extracted gate values in local assignment generators

Save the Progress in the Middle of the Subckt

Merkle-Hash the layer, but “puncture” the-root-to-leaf path for leaves in subcircuit, and use the roots of remaining subtrees to save the “equality” progress so far.

: extracted gate values in local assignment generators

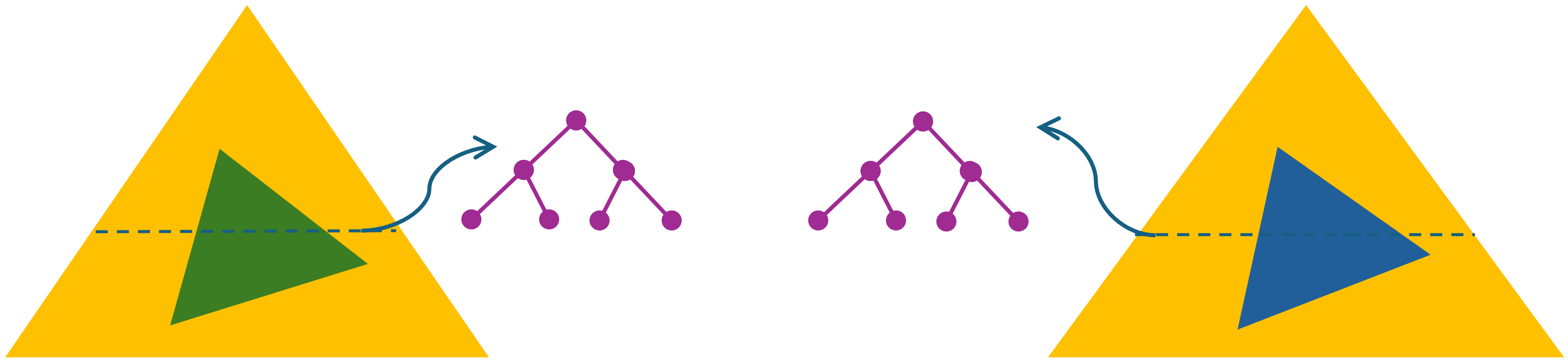
Save the Progress in the Middle of the Subckt

Merkle-Hash the layer, but "puncture" the-root-to-leaf path for leaves in subcircuit, and use the roots of remaining subtrees to save the "equality" progress so far.

◯ : extracted gate values in local assignment generators

Save the Progress in the Middle of the Subckt

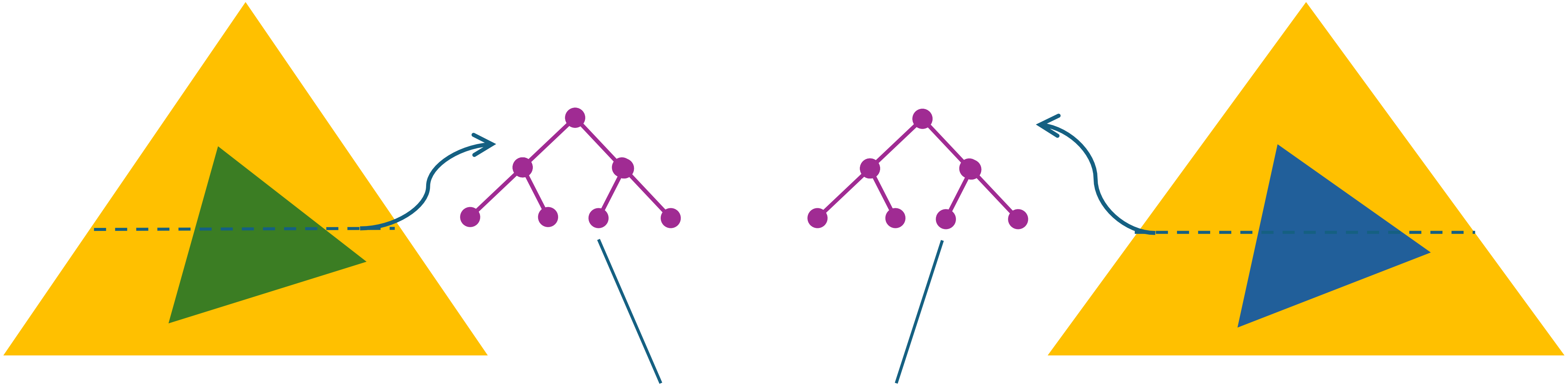
Merkle-Hash the layer, but "puncture" the-root-to-leaf path for leaves in subcircuit, and use the roots of remaining subtrees to save the "equality" progress so far.



◯ : extracted gate values in local assignment generators

Save the Progress in the Middle of the Subckt

Merkle-Hash the layer, but "puncture" the-root-to-leaf path for leaves in subcircuit, and use the roots of remaining subtrees to save the "equality" progress so far.

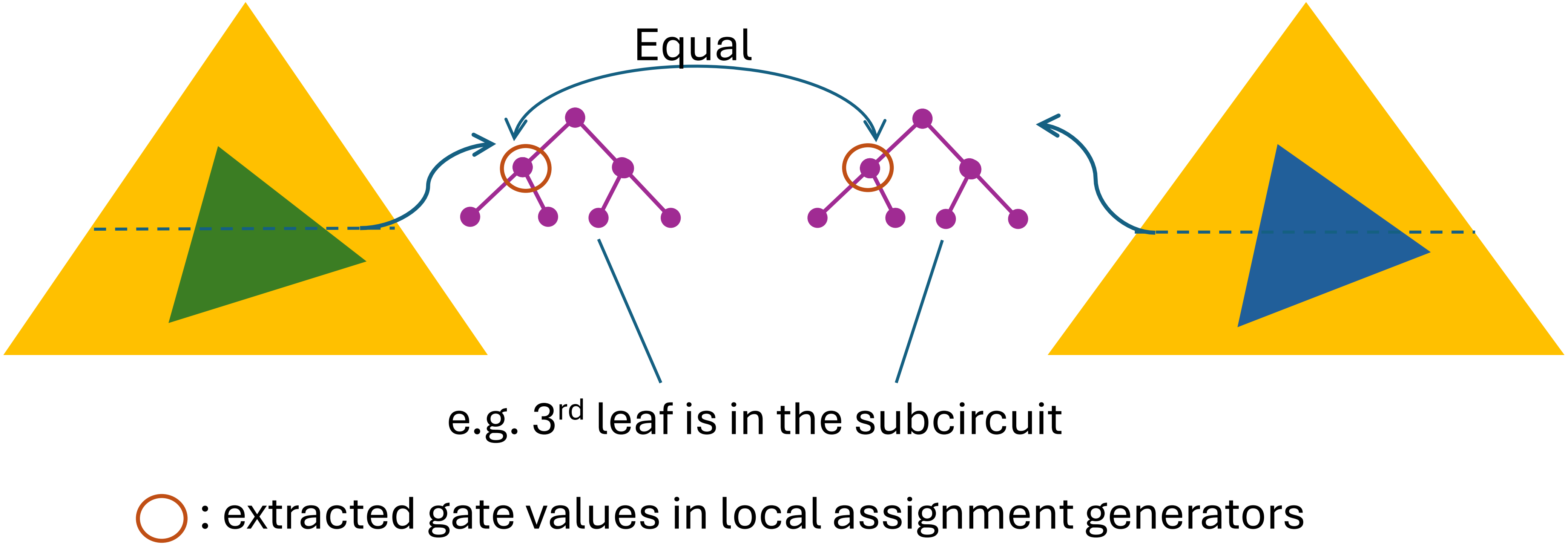


e.g. 3rd leaf is in the subcircuit

○ : extracted gate values in local assignment generators

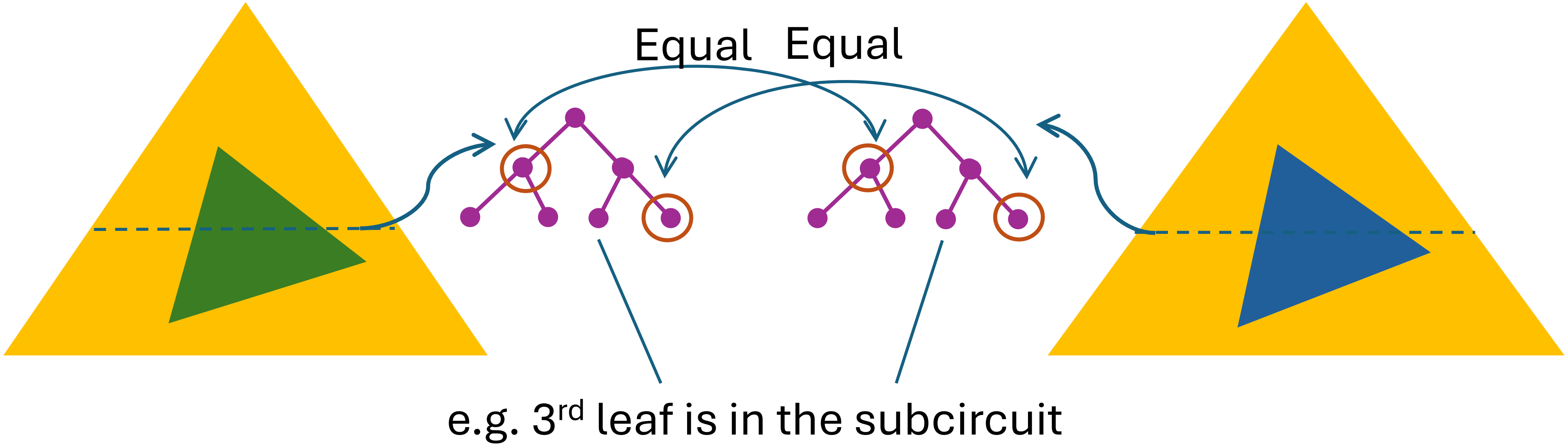
Save the Progress in the Middle of the Subckt

Merkle-Hash the layer, but "puncture" the-root-to-leaf path for leaves in subcircuit, and use the roots of remaining subtrees to save the "equality" progress so far.



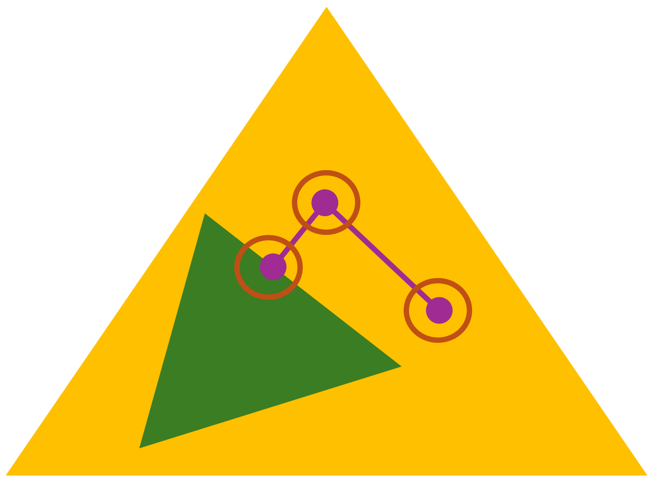
Save the Progress in the Middle of the Subckt

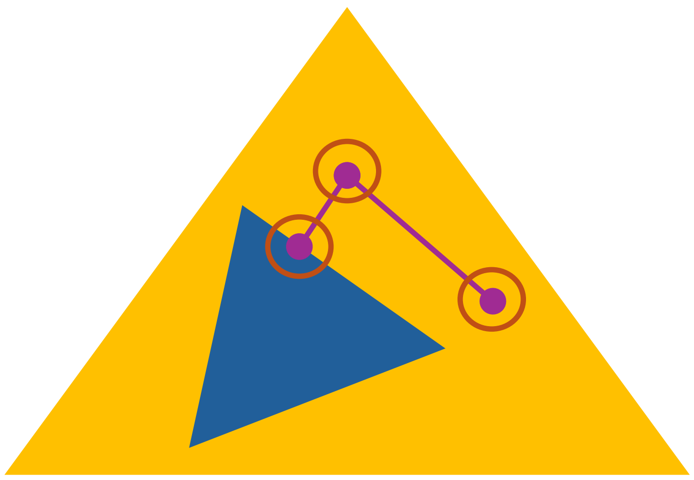
Merkle-Hash the layer, but "puncture" the-root-to-leaf path for leaves in subcircuit, and use the roots of remaining subtrees to save the "equality" progress so far.



 : extracted gate values in local assignment generators

Finishing the Proof: “Above” the Subckts

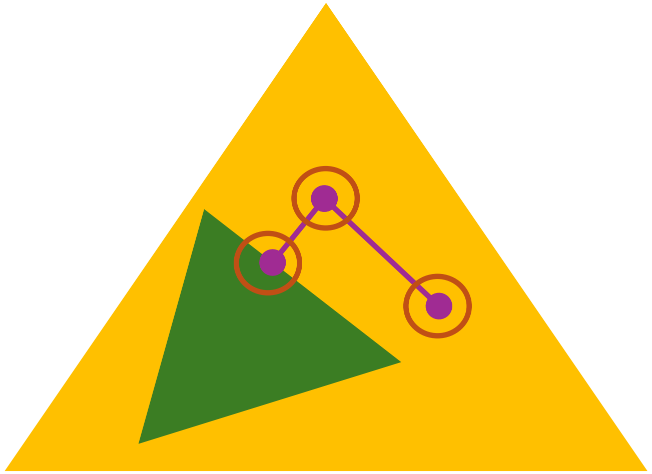


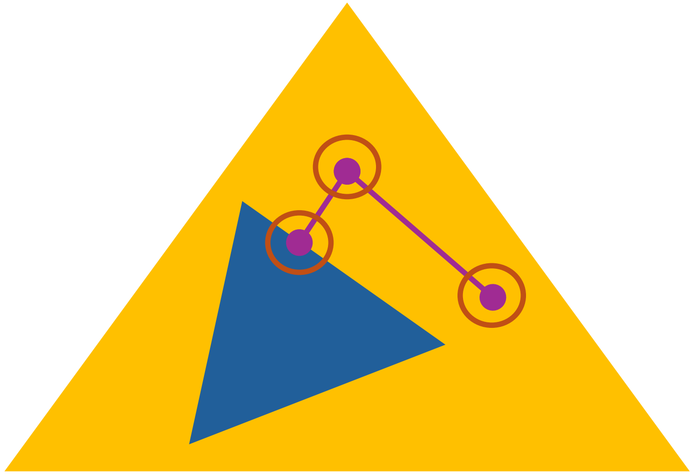


: extracted gate values in local assignment generators

Finishing the Proof: “Above” the Subckts

A general gate: Left Child: an output wire of the subcircuit
Right Child: a wire outside of the subcircuit

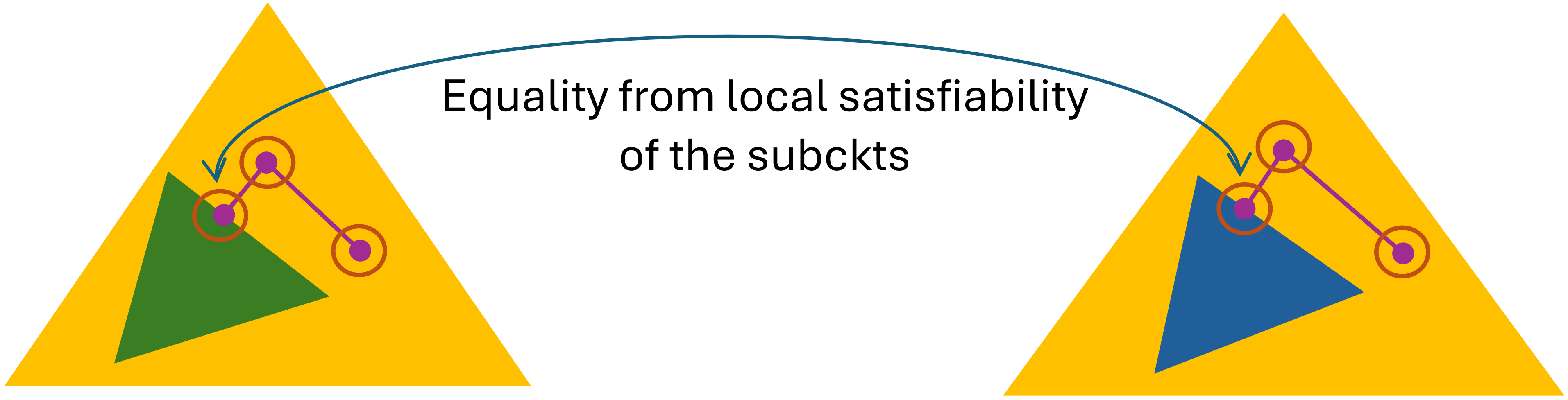




 : extracted gate values in local assignment generators

Finishing the Proof: "Above" the Subckts

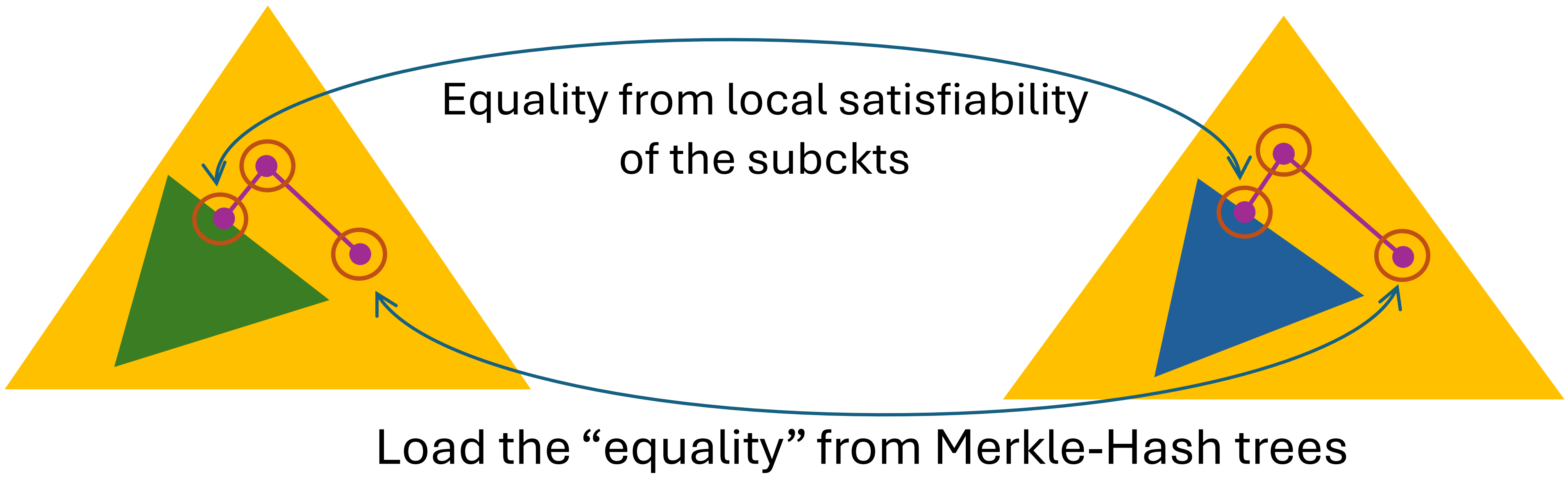
A general gate:
Left Child: an output wire of the subcircuit
Right Child: a wire outside of the subcircuit



: extracted gate values in local assignment generators

Finishing the Proof: “Above” the Subckts

A general gate: Left Child: an output wire of the subcircuit
Right Child: a wire outside of the subcircuit

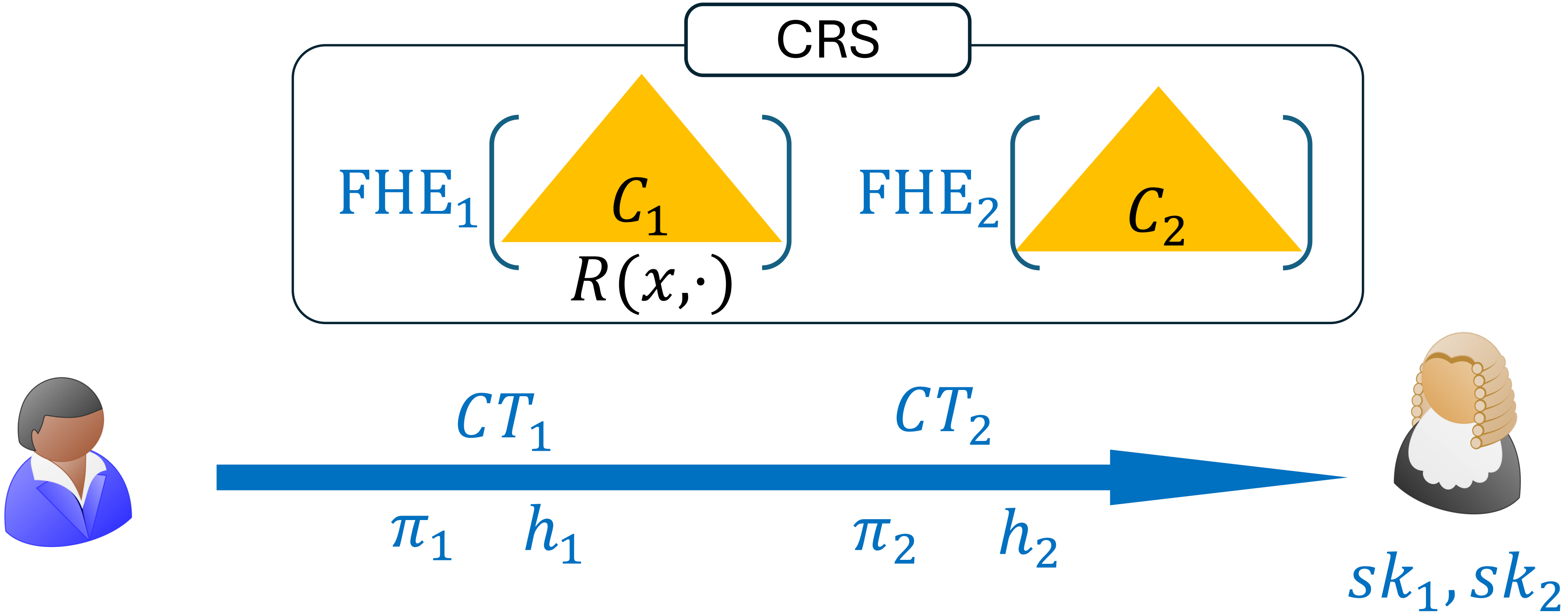


◯ : extracted gate values in local assignment generators

Rest of the Talk

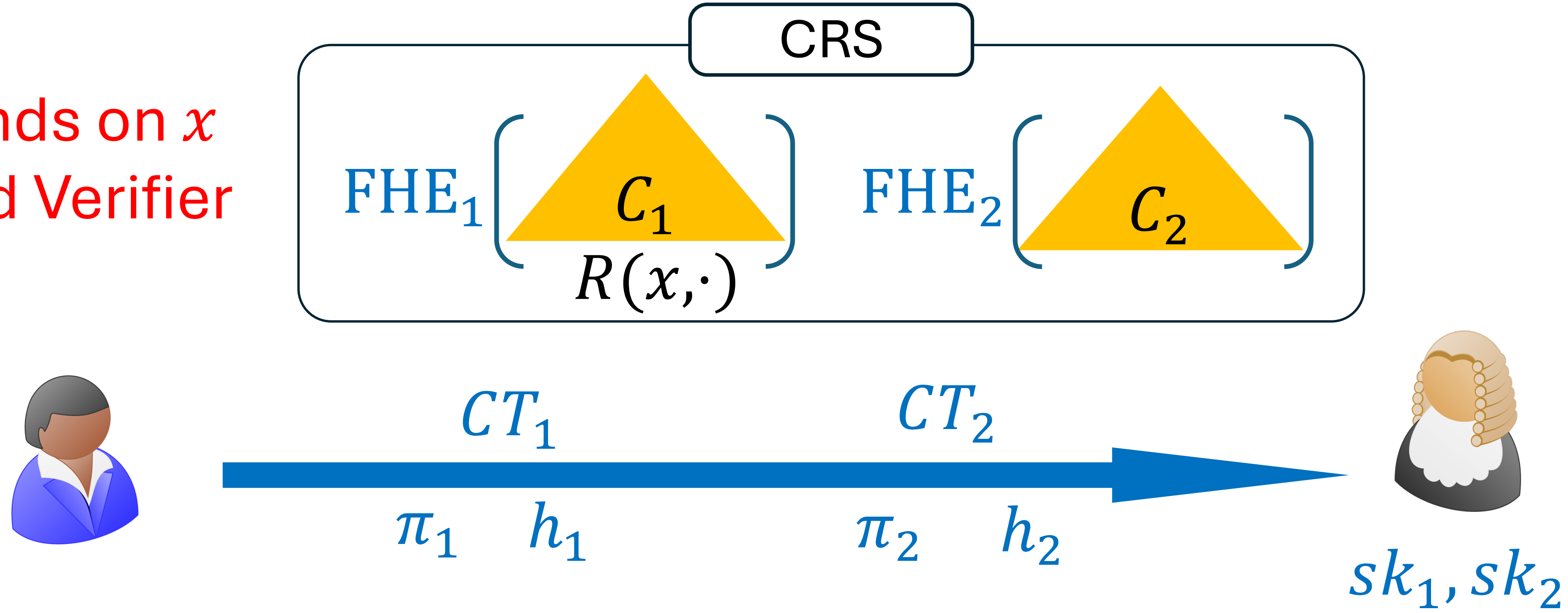
- **Achieving Public Verification & Random CRS**
- Discussion

Construction So Far



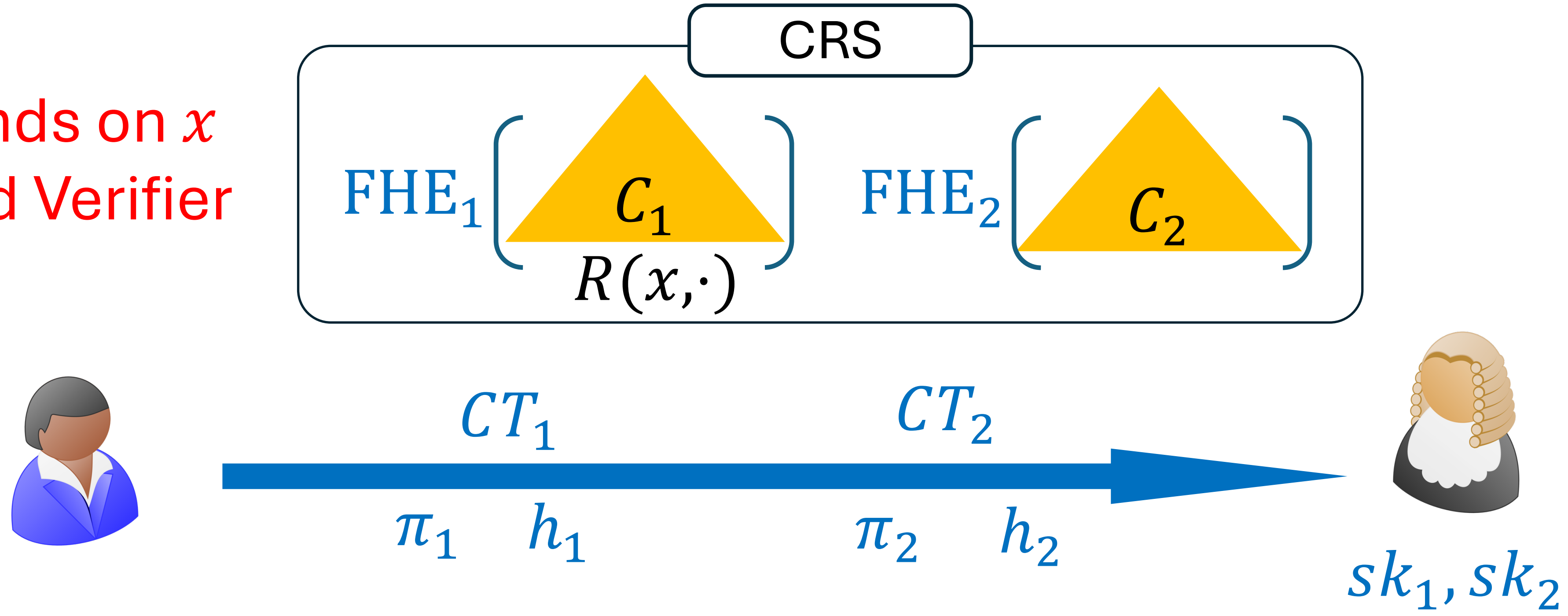
Construction So Far

- CRS depends on x
- Designated Verifier



Construction So Far

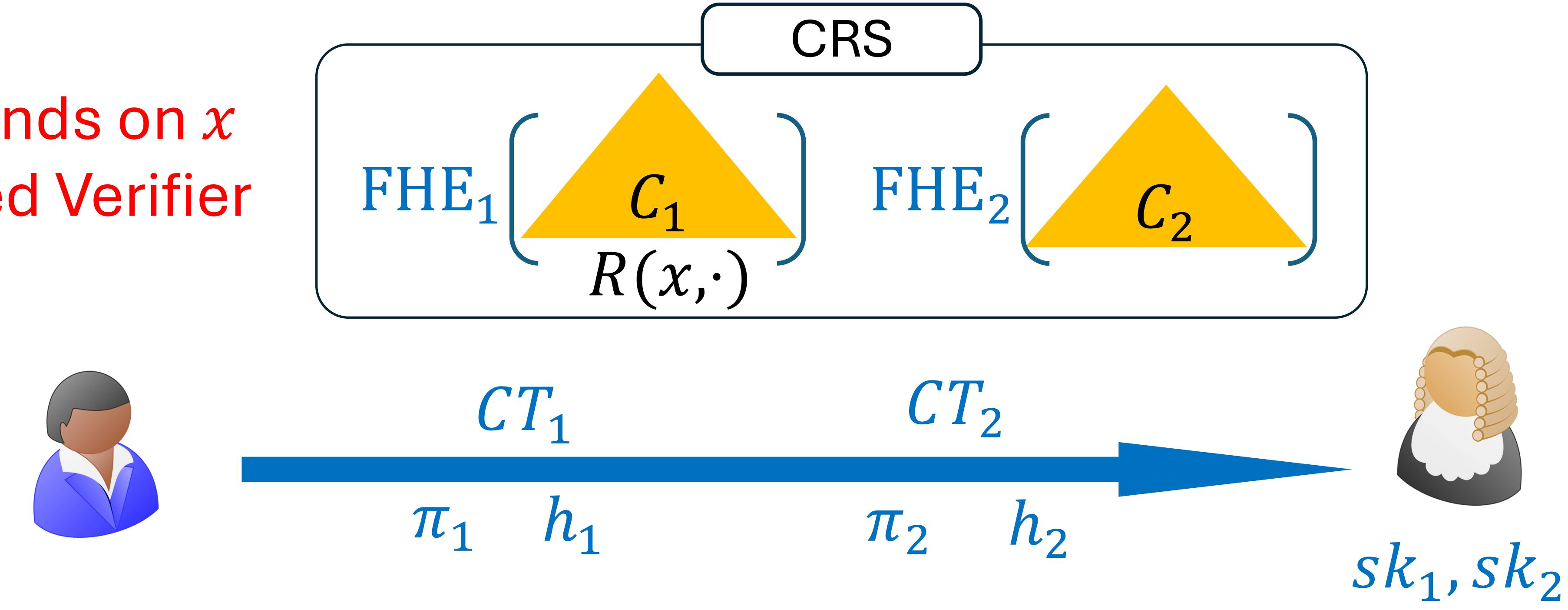
- CRS depends on x
- Designated Verifier



Public Verification: give out sk_1?

Construction So Far

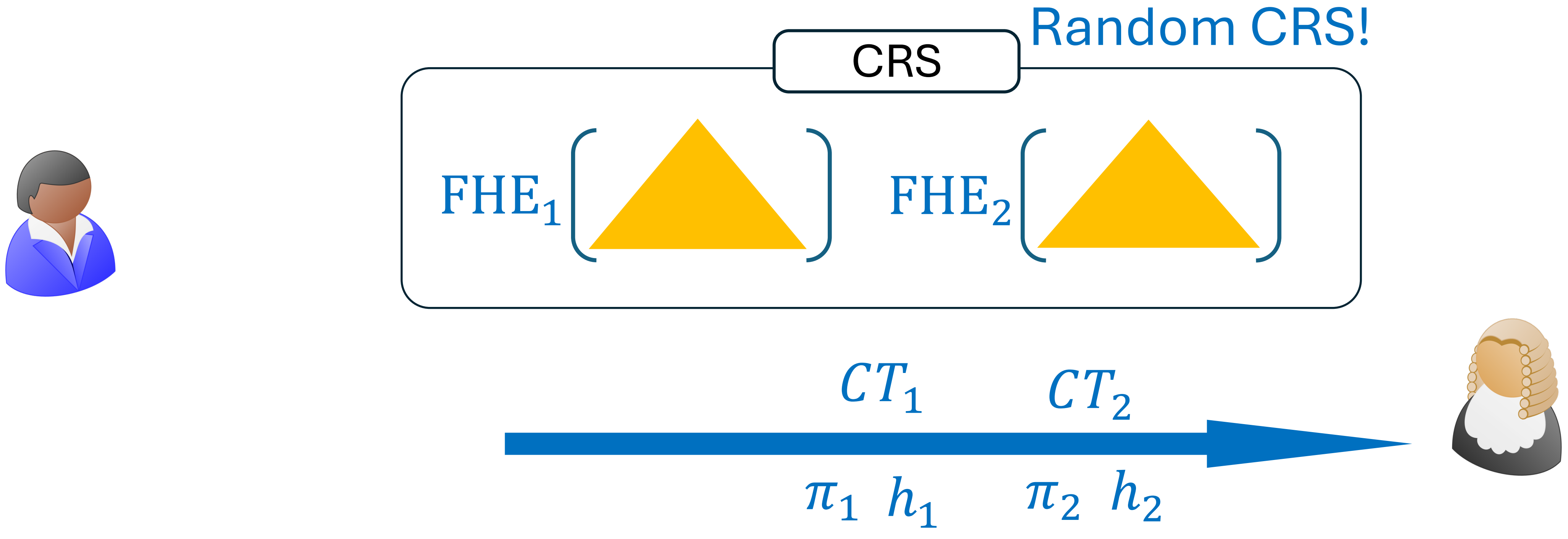
- CRS depends on x
- Designated Verifier



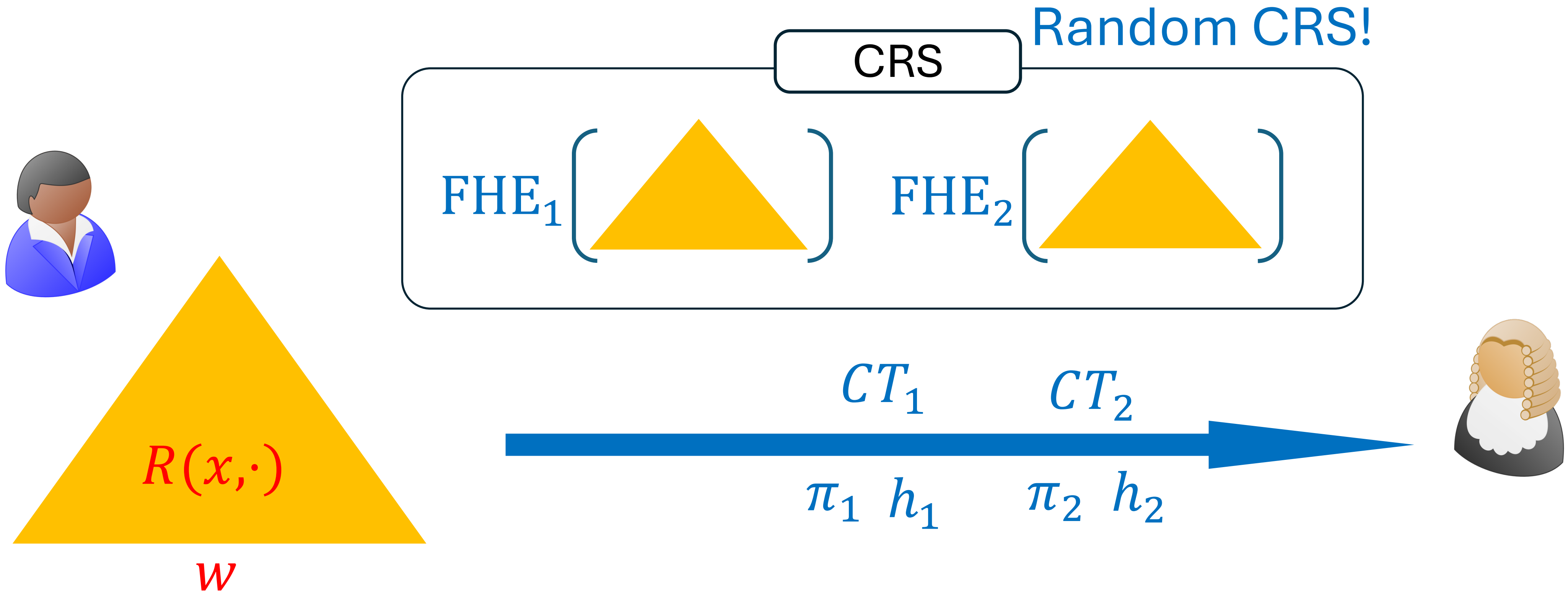
Public Verification: give out sk_1?

We don't need to encrypt $R(x,\cdot)$!

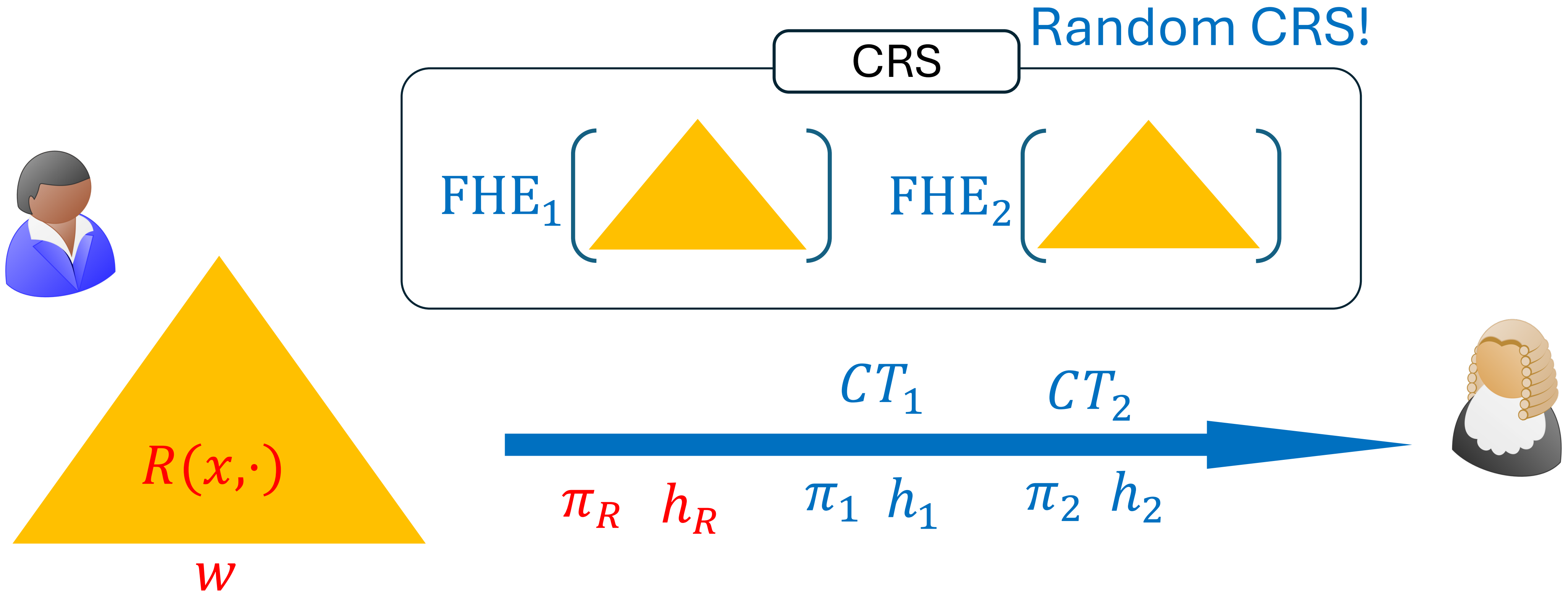
Achieving Public Verification



Achieving Public Verification

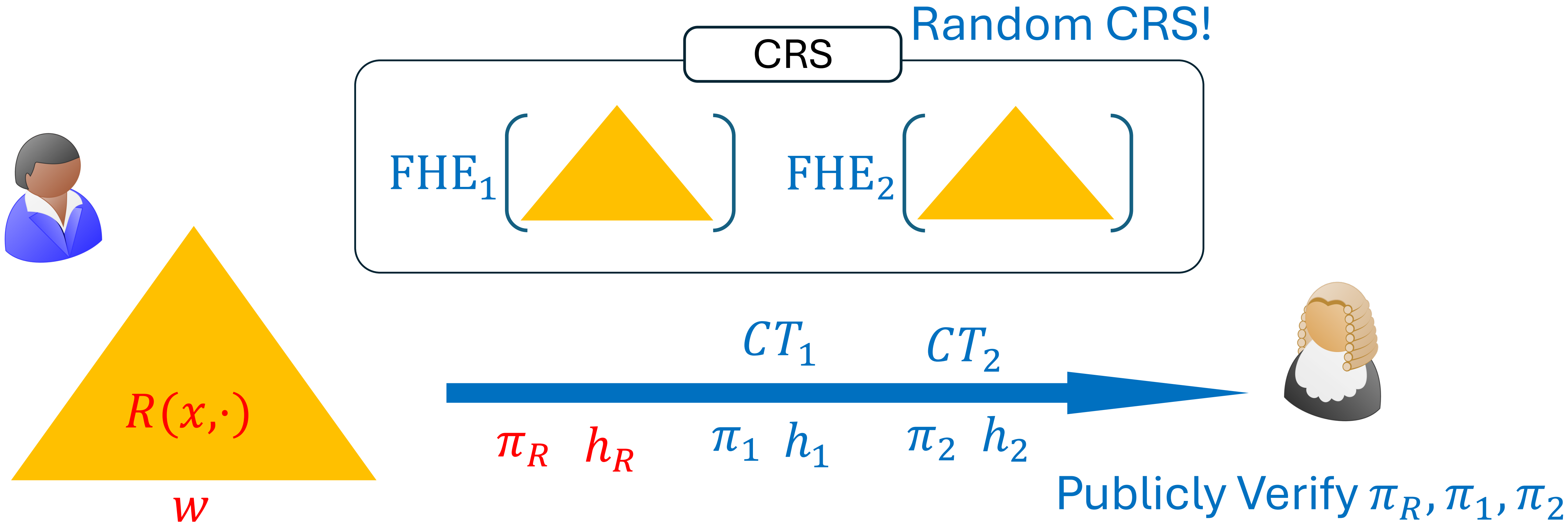


Achieving Public Verification



- h_R : Merkle-Hash of the wires in $R(x,\cdot)$
- π_R : BARGs for "each gate in $R(x,\cdot)$ is computed correctly".

Achieving Public Verification



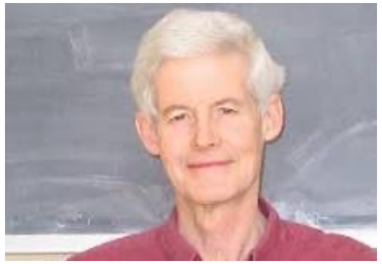
- h_R : Merkle-Hash of the wires in $R(x,\cdot)$
- π_R : BARGs for "each gate in $R(x,\cdot)$ is computed correctly".

Discussion

(Perspective on this line of ‘logic-based’ approach)

Recall: **Duality** between Logic and Computation

Proof Complexity	Computational Complexity
Frege	Formulas
Extended Frege	Circuits
Poly-size Extended Frege	Poly-size Circuits
Cook’s Theory PV	Poly-time Turing Machines
…	…



Logic: A Forgotten Structure in Cryptography?

Logic: A Forgotten Structure in Cryptography?

Non-black-box use of functions, i.e. use their circuits

Logic: A Forgotten Structure in Cryptography?

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization,
…

Logic: A Forgotten Structure in Cryptography?

"Non-black-box use" of *mathematical claims*

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization,
...

Logic: A Forgotten Structure in Cryptography?

“Non-black-box use” of *mathematical claims*

- iO via math proofs of equivalence [JJ’22]
- SNARGs via propositional logic [This work]
- …

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization,
…

Logic: A Forgotten Structure in Cryptography?

“Non-black-box use” of *mathematical claims*

- iO via math proofs of equivalence [JJ’22]
- SNARGs via propositional logic [This work]
- …

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization,
…

Logic: A Forgotten Structure in Cryptography?

“Non-black-box use” of *mathematical claims*

- iO via math proofs of equivalence [JJ’22]
- SNARGs via propositional logic [This work]
- …

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization,
…

Logic: A Forgotten Structure in Cryptography?

“Non-black-box use” of *mathematical claims*

- iO via math proofs of equivalence [JJ’22]
- SNARGs via propositional logic [This work]
- …

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization,
…

Is non-black-box techniques ***necessary***?

Logic: A Forgotten Structure in Cryptography?

“Non-black-box use” of *mathematical claims*

- iO via math proofs of equivalence [JJ’22]
- SNARGs via propositional logic [This work]
- …

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization,
…

Is non-black-box techniques ***necessary***?

Black-box separations

Logic: A Forgotten Structure in Cryptography?

“Non-black-box use” of *mathematical claims*

- iO via math proofs of equivalence [JJ’22]
- SNARGs via propositional logic [This work]
- …

Non-black-box use of functions, i.e. use their circuits

many examples:
FHE, garbled circuits, arithmetization, …

Is non-black-box techniques ***necessary***?

Impossibility?

Black-box separations

Thank you!

Q & A