# Security Bounds for Proof-Carrying Data from Straightline Extractors

**Alessandro Chiesa, Ziyi Guan, Shahar Samocha, Eylon Yogev**

EPFL STARKWARE Bar-Ilan University אוניברסיטת בר־אילן

https://eprint.iacr.org/2023/1646

# TL;DR

# TL;DR

What is *proof-carrying data* (PCD)?

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It’s useful for efficiently verifying distributed computations.

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It's useful for efficiently verifying distributed computations.

**Problem:**

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It’s useful for efficiently verifying distributed computations.

**Problem:**
- PCD is deployed under the assumption "security of PCD" = "security of underlying SNARK".

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It's useful for efficiently verifying distributed computations.

**Problem:**
- PCD is deployed under the assumption "security of PCD" = "security of underlying SNARK".
- BUT existing security analyses show a huge gap in security ("PCD is far less secure than underlying SNARK").

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It's useful for efficiently verifying distributed computations.

**Problem:**
- PCD is deployed under the assumption "security of PCD" = "security of underlying SNARK".
- BUT existing security analyses show a huge gap in security ("PCD is far less secure than underlying SNARK").

**This work:**

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It's useful for efficiently verifying distributed computations.

**Problem:**
- PCD is deployed under the assumption "security of PCD" = "security of underlying SNARK".
- BUT existing security analyses show a huge gap in security ("PCD is far less secure than underlying SNARK").

**This work:**
- We propose an **idealized PCD** that models hash-based PCD in practice.

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It's useful for efficiently verifying distributed computations.

**Problem:**
- PCD is deployed under the assumption "security of PCD" = "security of underlying SNARK".
- BUT existing security analyses show a huge gap in security ("PCD is far less secure than underlying SNARK").

**This work:**
- We propose an **idealized PCD** that models hash-based PCD in practice.
- We prove that this idealized PCD is **as secure as its underlying SNARK**.

# What is proof-carrying data (PCD)? [1/2]

# What is proof-carrying data (PCD)? [1/2]

Proof-carrying data (PCD)

- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

# What is proof-carrying data (PCD)? [1/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

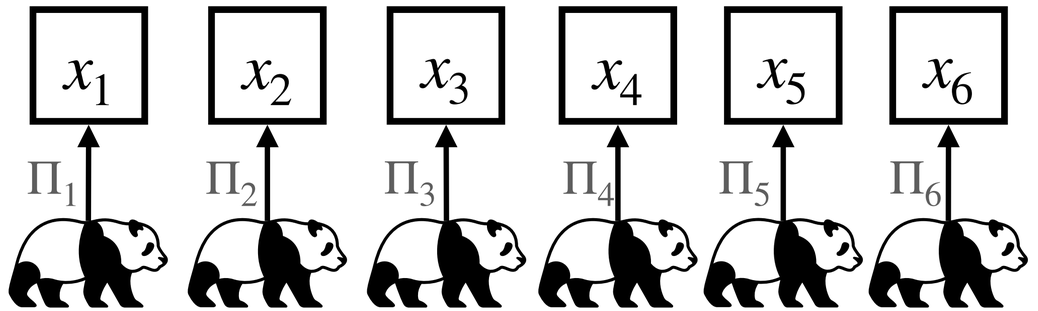
E.g. A simple distributed computation: summing six numbers



# What is proof-carrying data (PCD)? [1/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

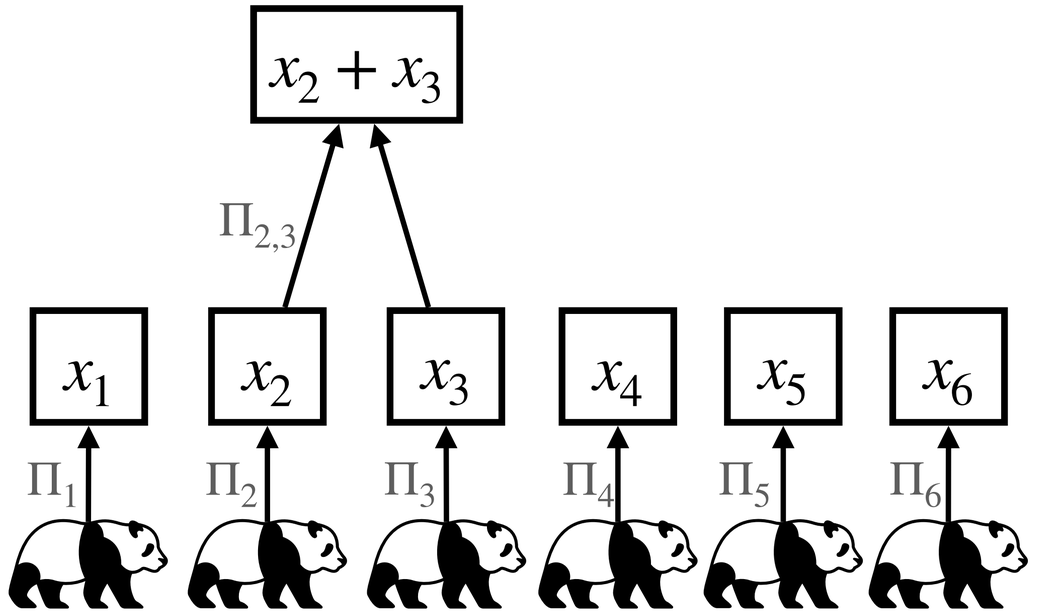
E.g. A simple distributed computation: summing six numbers



# What is proof-carrying data (PCD)? [1/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

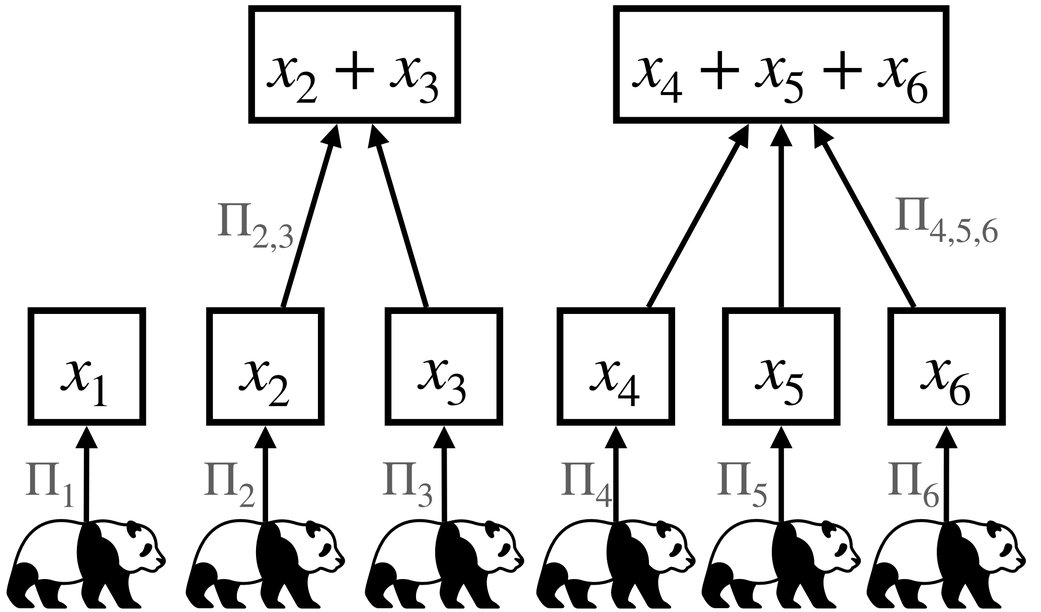
E.g. A simple distributed computation: summing six numbers



# What is proof-carrying data (PCD)? [1/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

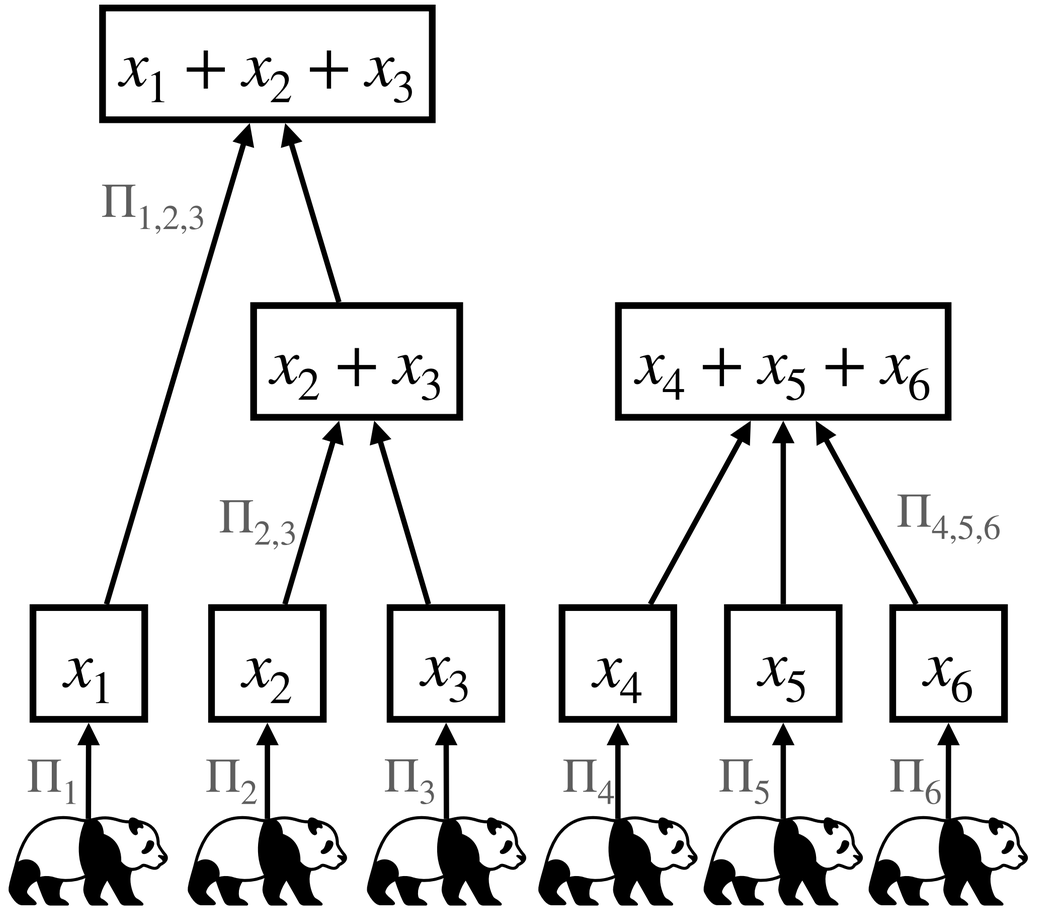
E.g. A simple distributed computation: summing six numbers



# What is proof-carrying data (PCD)? [1/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

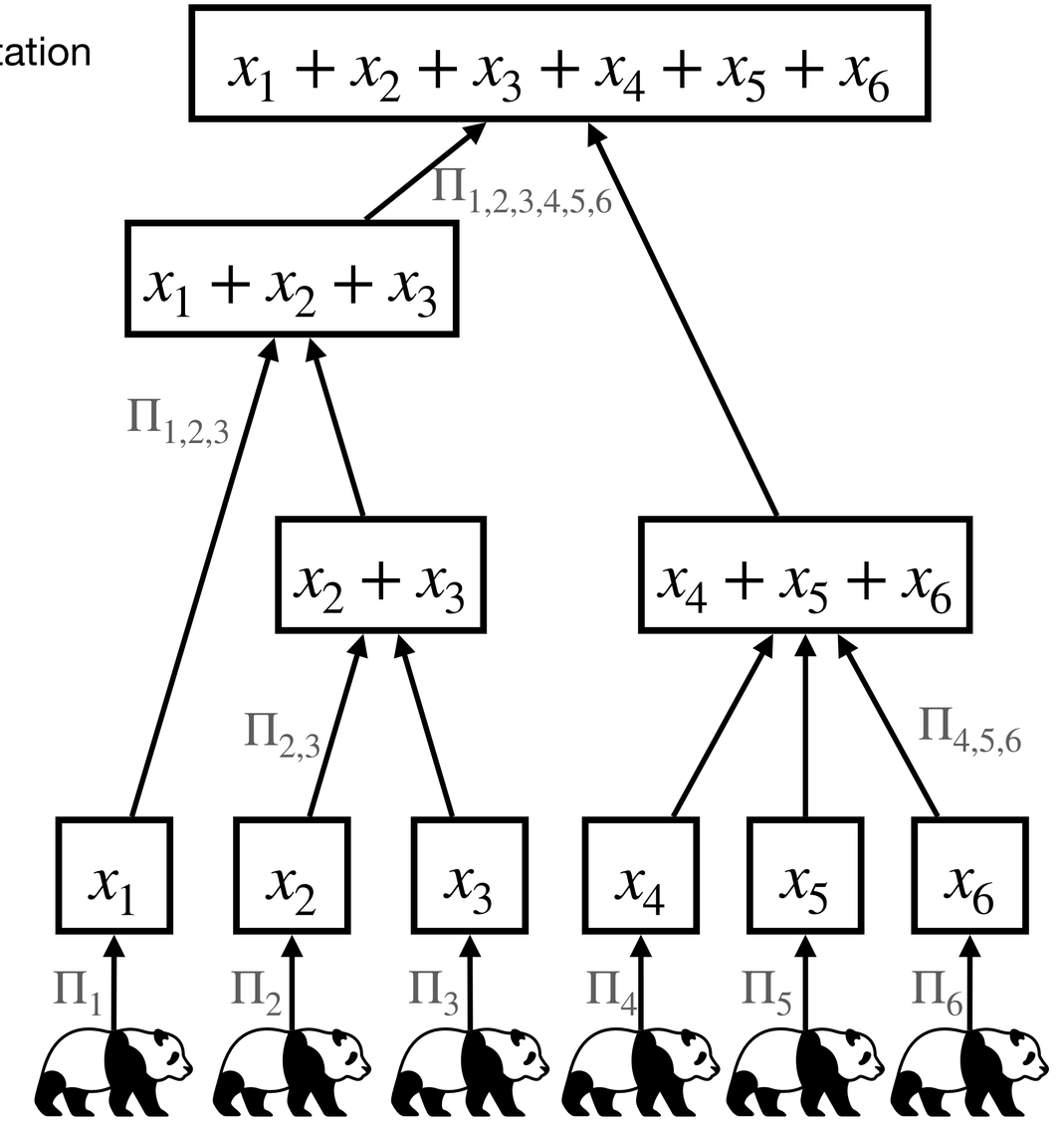
E.g. A simple distributed computation: summing six numbers



# What is proof-carrying data (PCD)? [1/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

E.g. A simple distributed computation: summing six numbers

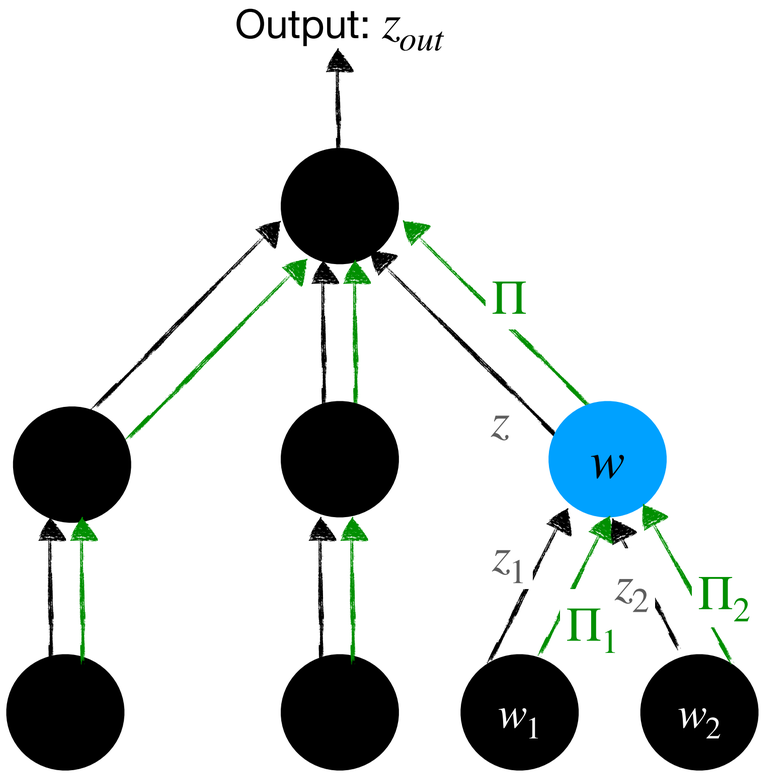


# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

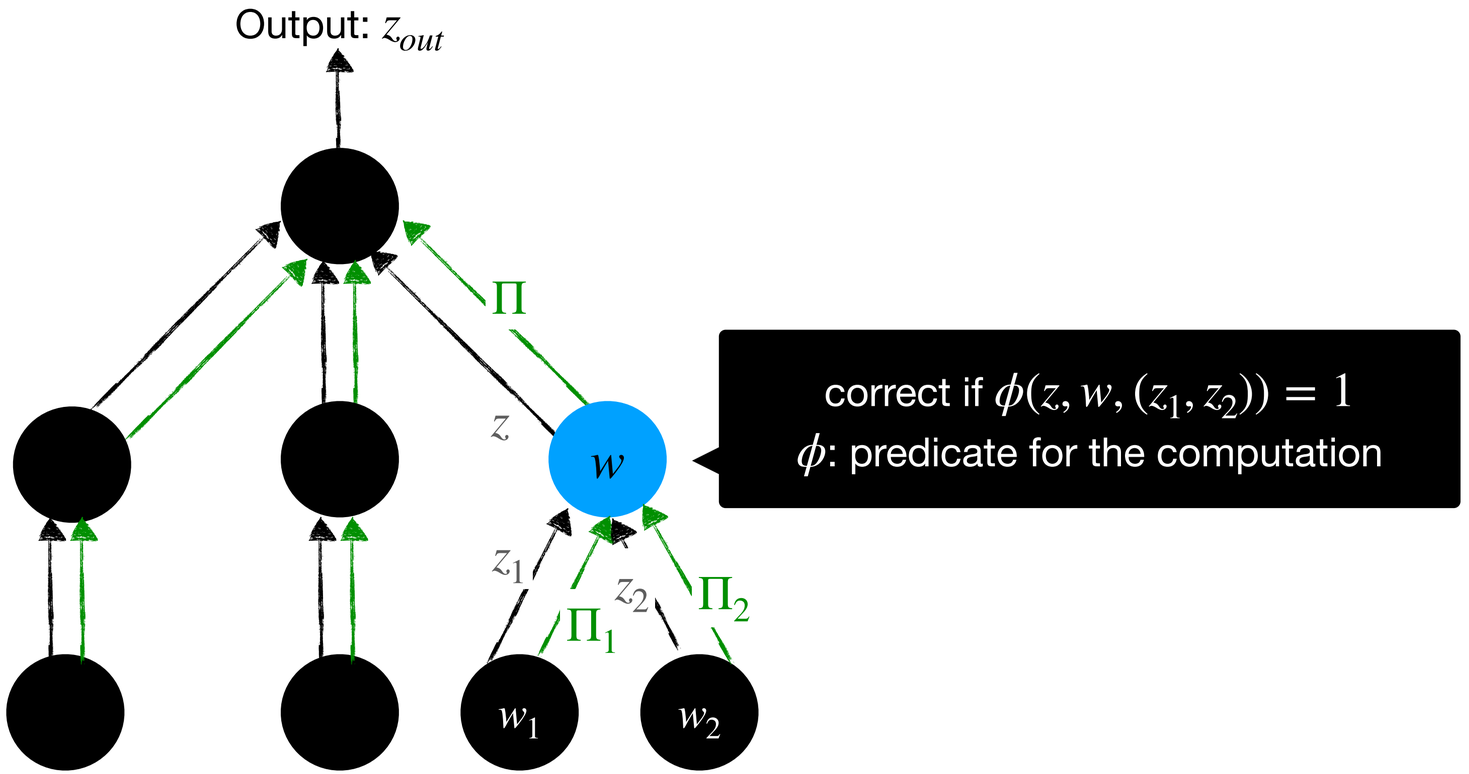



PCD transcript $T$

size N = 8,depth D = 3

# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

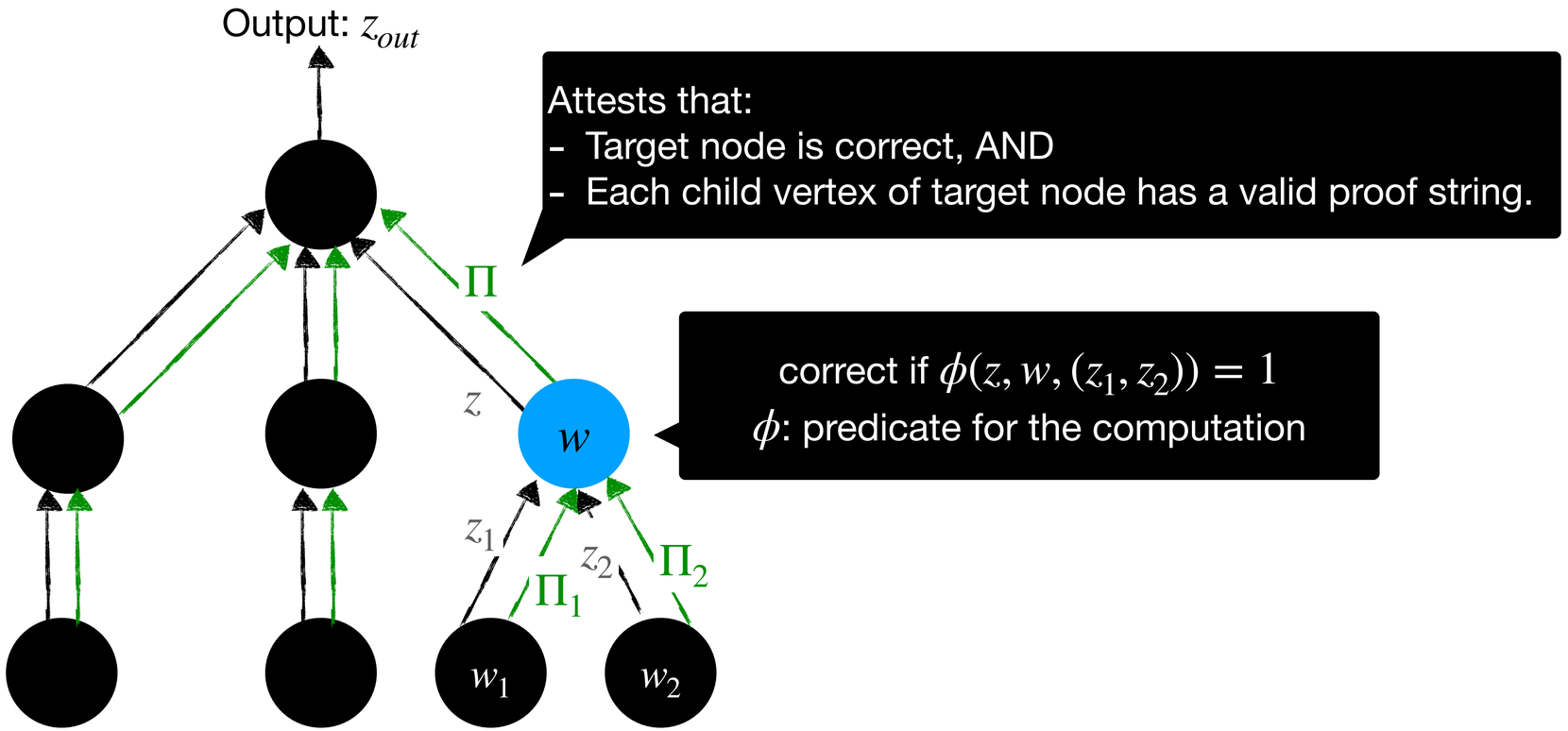



PCD transcript $T$

size N $= 8$,depth D $= 3$

# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

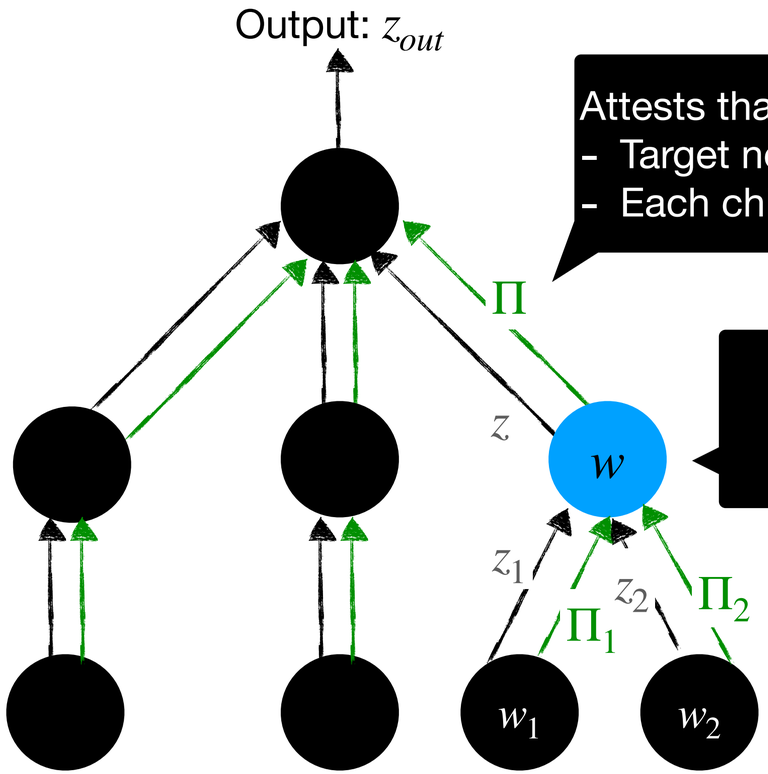



PCD transcript $T$

size N $= 8$,depth D $= 3$

# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

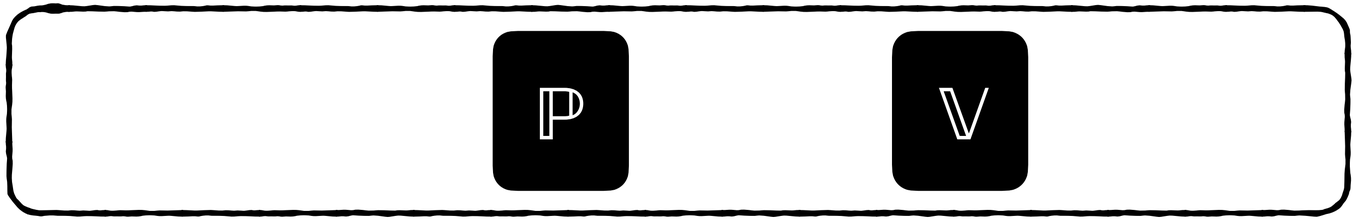



Attests that:
- Target node is correct, AND
- Each child vertex of target node has a valid proof string.

correct if $\phi(z, w, (z_1, z_2)) = 1$
$\phi$: predicate for the computation

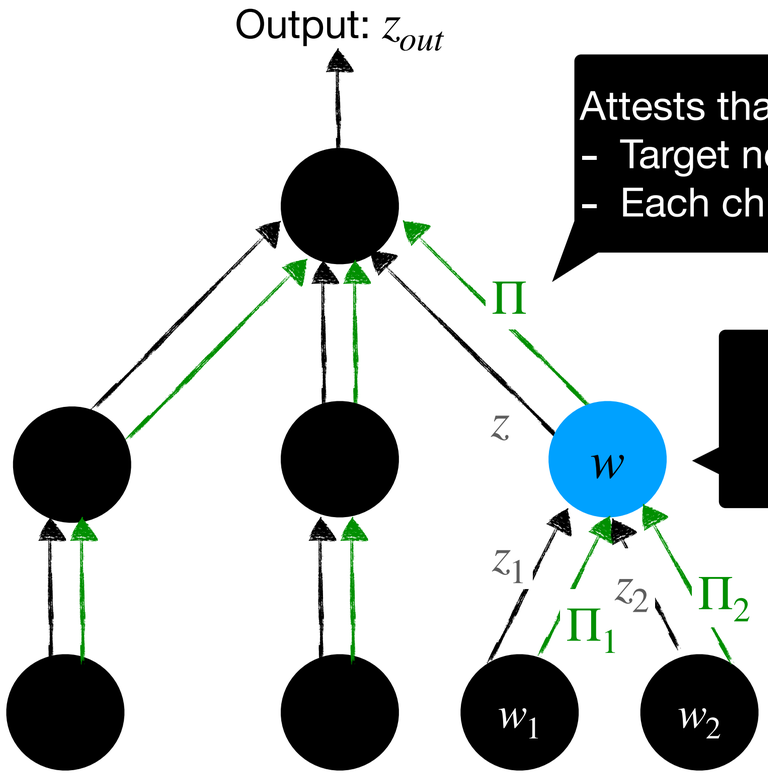
PCD transcript $T$
size N = 8,depth D = 3

PCD prover $\mathbb{P}$ and PCD verifier $\mathbb{V}$



# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

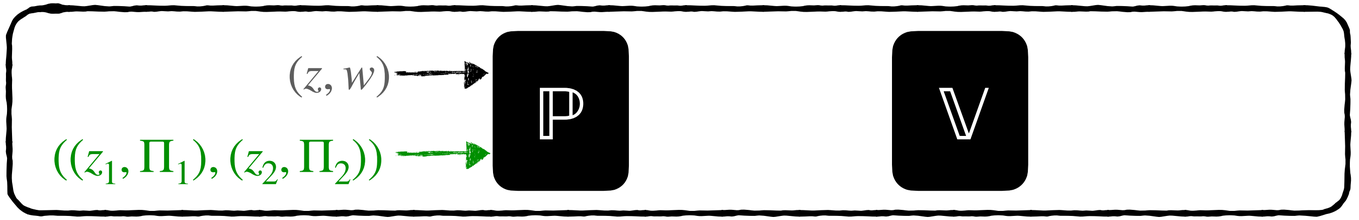



Attests that:
- Target node is correct, AND
- Each child vertex of target node has a valid proof string.

correct if $\phi(z, w, (z_1, z_2)) = 1$
$\phi$: predicate for the computation

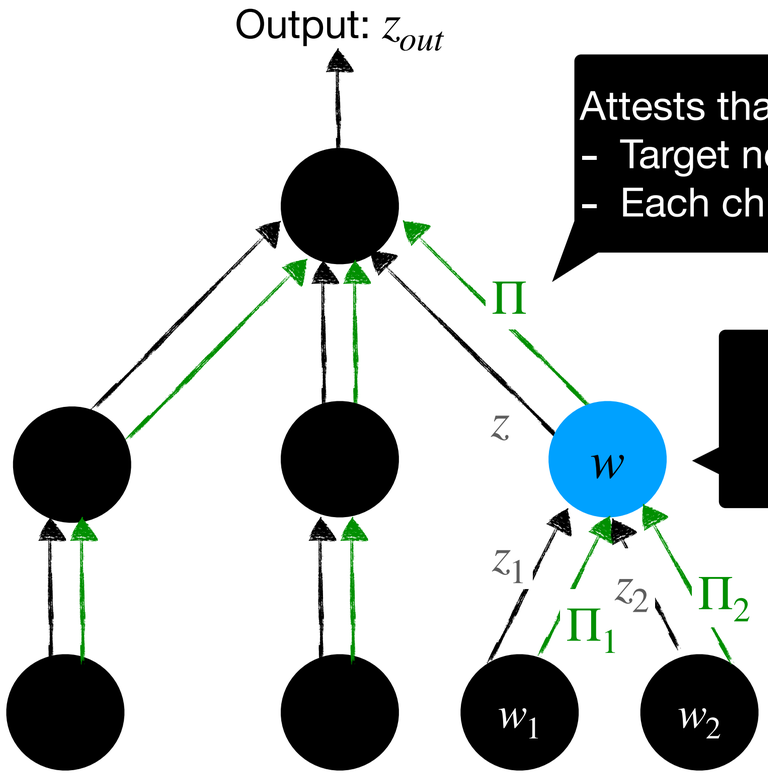
PCD transcript $T$
size N = 8,depth D = 3

PCD prover $\mathbb{P}$ and PCD verifier $\mathbb{V}$



# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

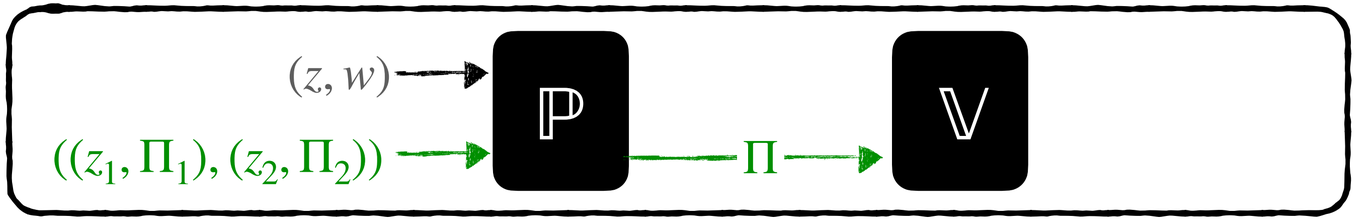



Attests that:
- Target node is correct, AND
- Each child vertex of target node has a valid proof string.

correct if $\phi(z, w, (z_1, z_2)) = 1$
$\phi$: predicate for the computation

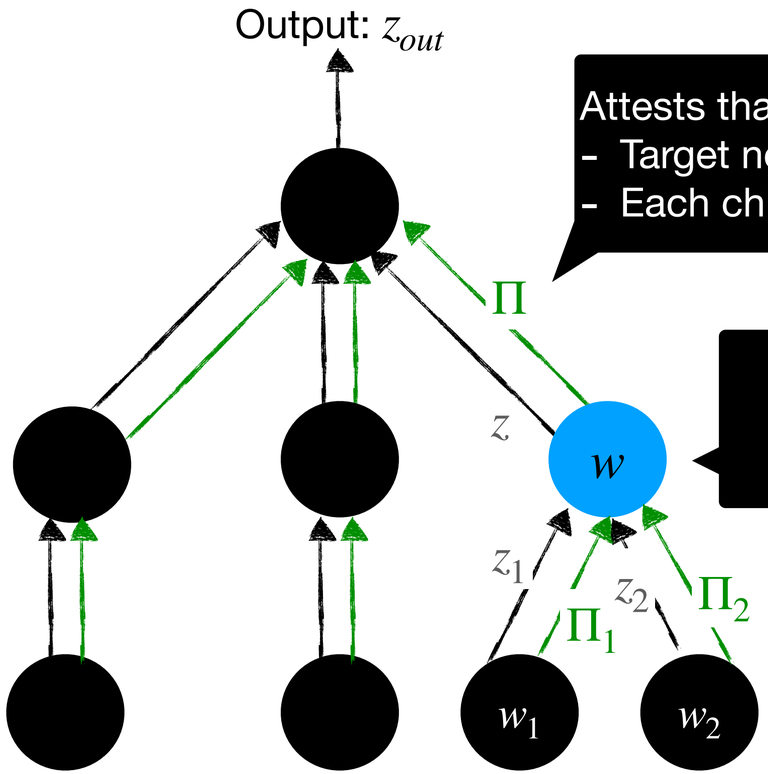
PCD transcript $T$
size N = 8,depth D = 3

PCD prover $\mathbb{P}$ and PCD verifier $\mathbb{V}$



# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

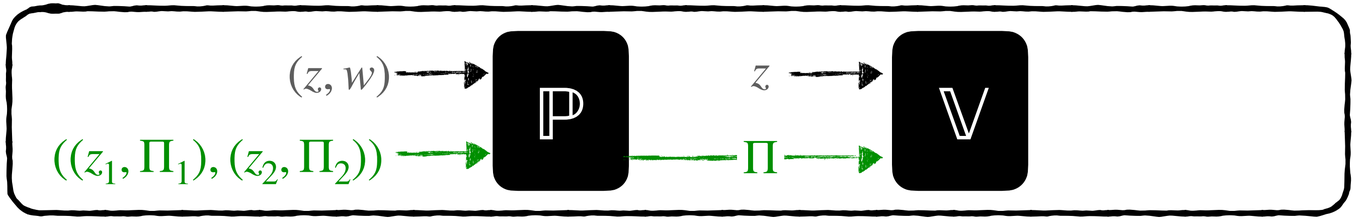



Attests that:
- Target node is correct, AND
- Each child vertex of target node has a valid proof string.

correct if $\phi(z, w, (z_1, z_2)) = 1$
$\phi$: predicate for the computation

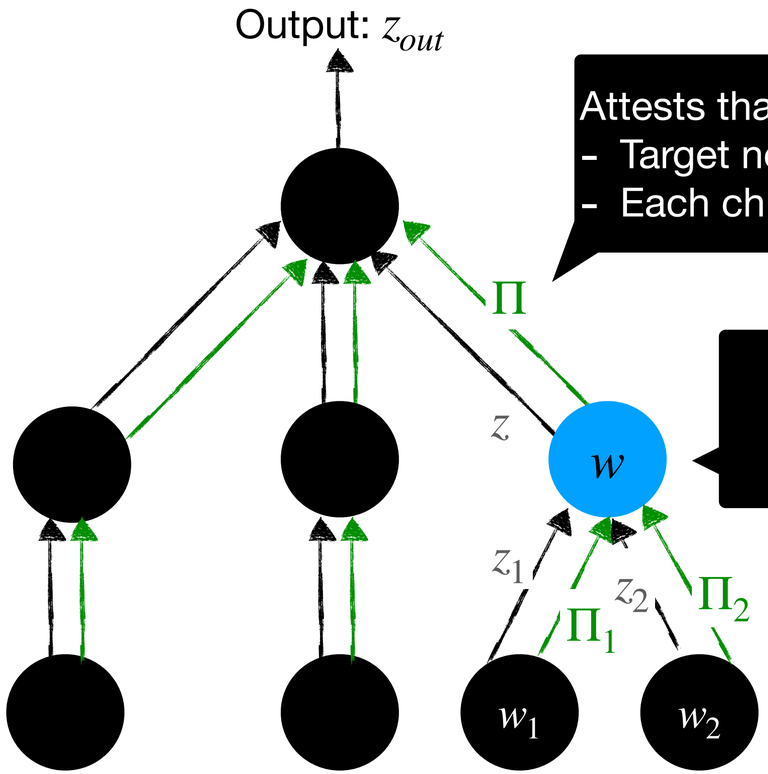
PCD transcript $T$
size N = 8,depth D = 3

PCD prover $\mathbb{P}$ and PCD verifier $\mathbb{V}$



# What is proof-carrying data (PCD)? [2/2]

Proof-carrying data (PCD)
- Enables mutually distrustful parties to perform a distributed computation
- The correctness of each step can be **verified efficiently**

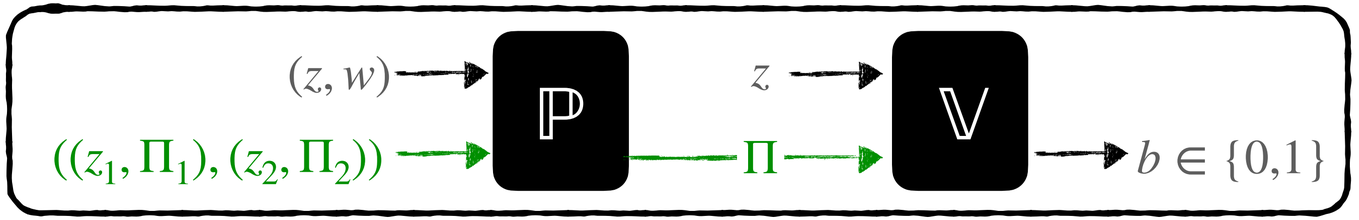



Attests that:
- Target node is correct, AND
- Each child vertex of target node has a valid proof string.

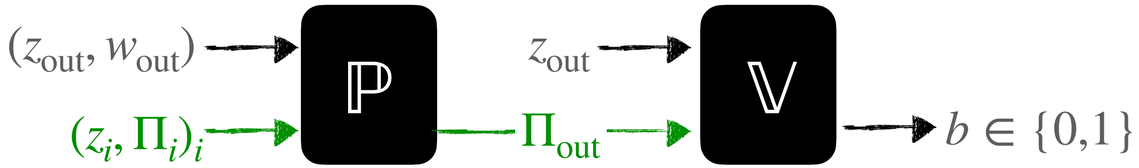
correct if $\phi(z, w, (z_1, z_2)) = 1$
$\phi$: predicate for the computation

PCD transcript $T$
size N = 8,depth D = 3

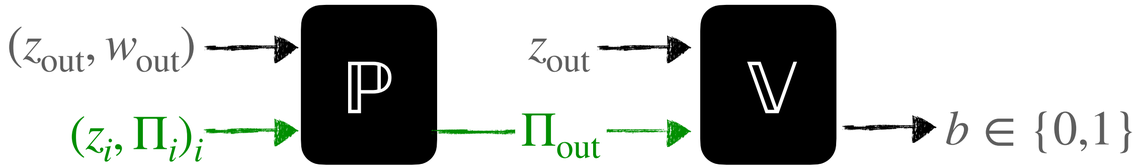
PCD prover $\mathbb{P}$ and PCD verifier $\mathbb{V}$



# Security guarantee of PCD



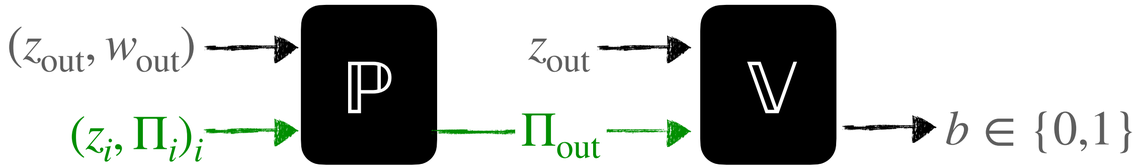
# Security guarantee of PCD




$T$: computation transcript
$\lambda$: security parameter
$D$: maximum transcript depth
N: maximum transcript size

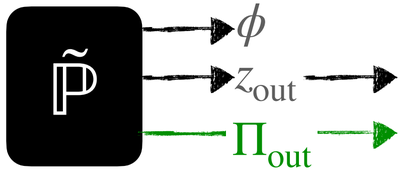
Knowledge soundness: ∀ bounded $\tilde{\mathbb{P}}$, ∃ an efficient extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ who outputs $T$ such that the following happens with probability at most $\kappa(\lambda, \mathsf{D}, \mathsf{N})$:

# Security guarantee of PCD

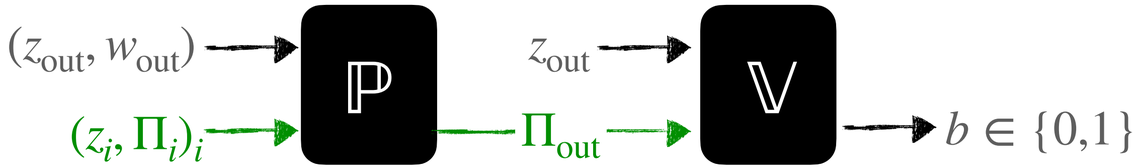



$T$: computation transcript
$\lambda$: security parameter
$D$: maximum transcript depth
N: maximum transcript size

Knowledge soundness: ∀ bounded $\tilde{\mathbb{P}}$, ∃ an efficient extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ who outputs $T$ such that the following happens with probability at most $\kappa(\lambda, \mathsf{D}, \mathsf{N})$:

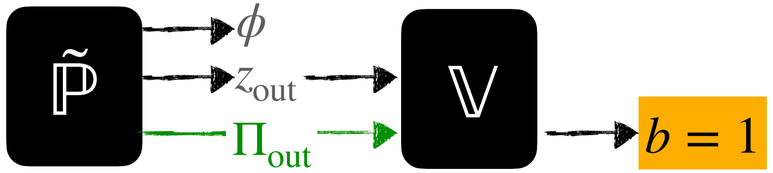


# Security guarantee of PCD

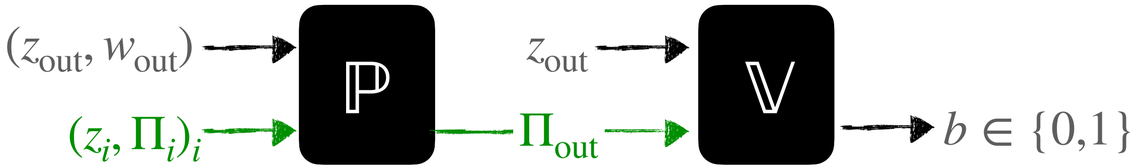



$T$: computation transcript
$\lambda$: security parameter
$D$: maximum transcript depth
N: maximum transcript size

Knowledge soundness: ∀ bounded $\tilde{\mathbb{P}}$, ∃ an efficient extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ who outputs $T$ such that the following happens with probability at most $\kappa(\lambda, D, N)$:

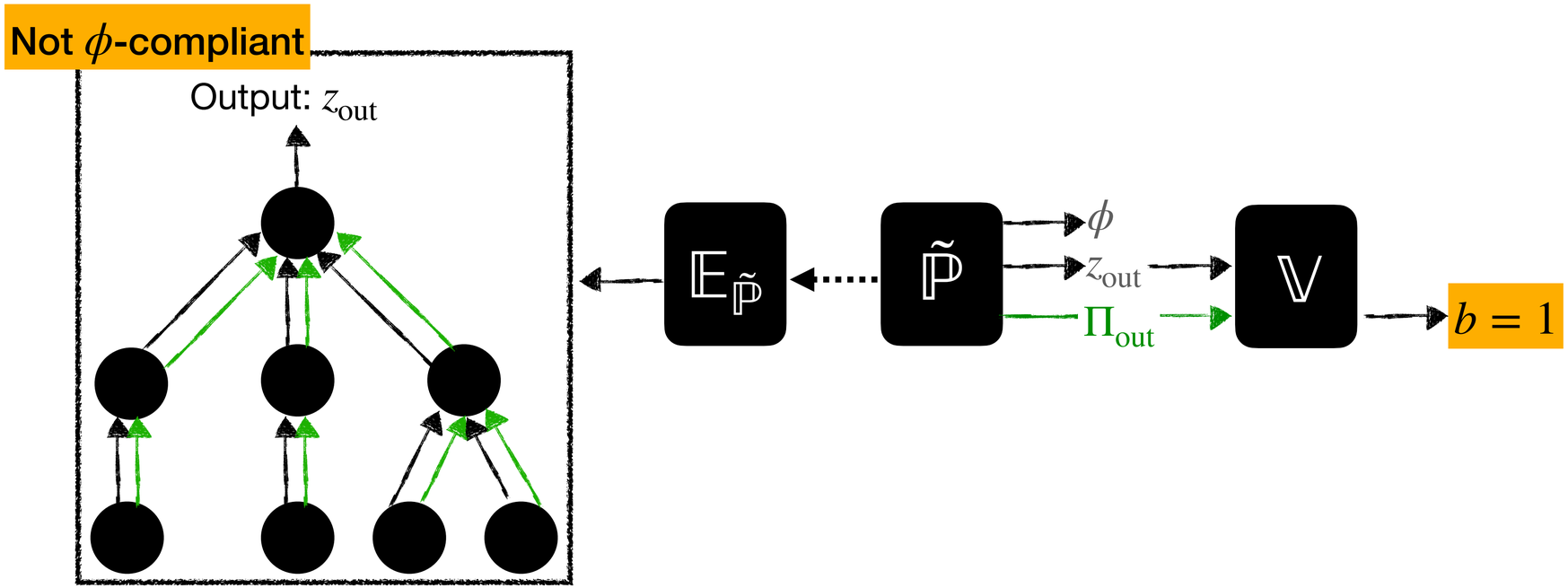


# Security guarantee of PCD




$T$: computation transcript
$\lambda$: security parameter
$D$: maximum transcript depth
N: maximum transcript size

Knowledge soundness: ∀ bounded $\tilde{\mathbb{P}}$, ∃ an efficient extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ who outputs $T$ such that the following happens with probability at most $\kappa(\lambda, D, N)$:



# Review: SNARK

# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

# Review: SNARK

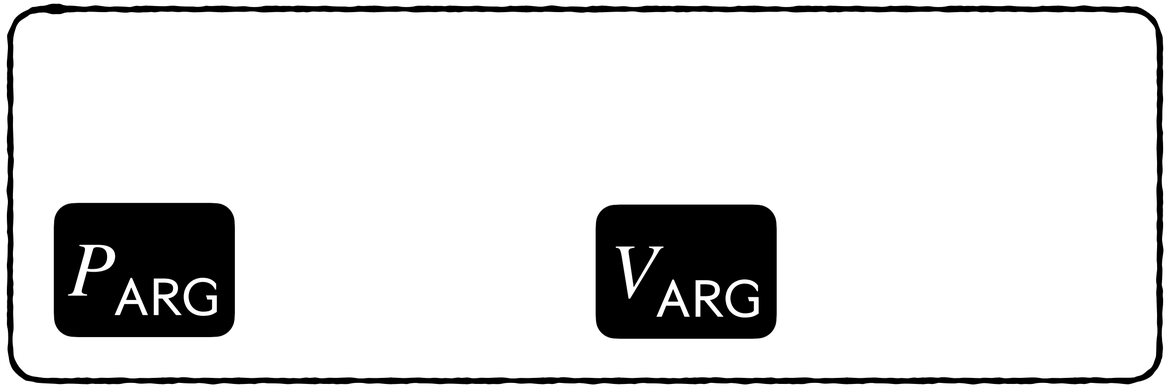
PCD can be constructed from a SNARK (e.g., for CSAT).

$$\text{CSAT} := \{(C, a) : C(a) = 1\}$$

# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

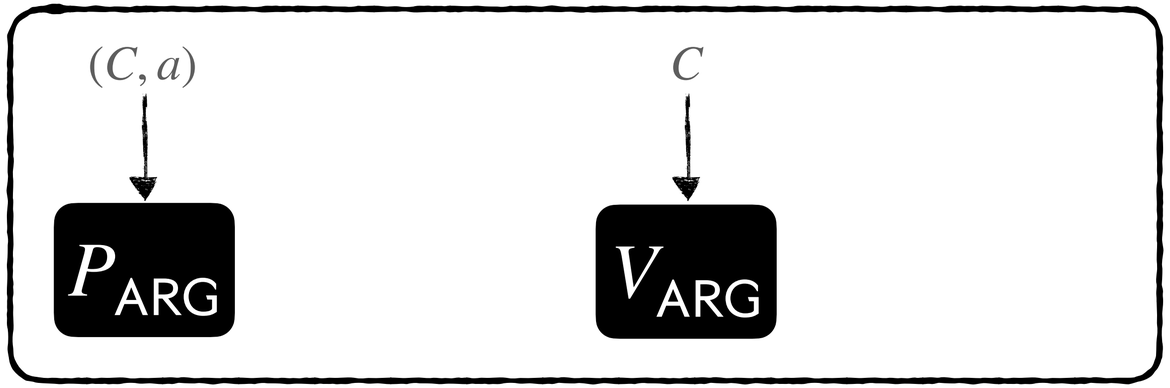
$$\mathsf{CSAT} := \{(C, a) : C(a) = 1\}$$



# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

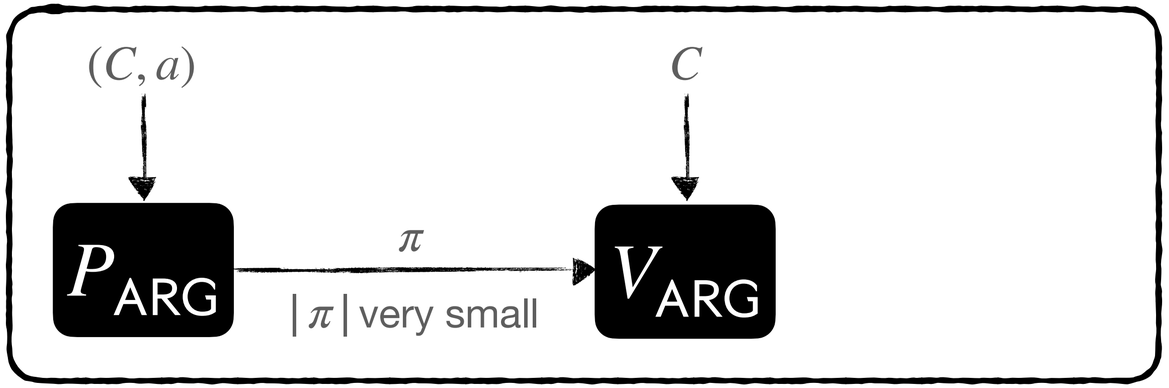
$$\mathsf{CSAT} := \{(C, a) : C(a) = 1\}$$



# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

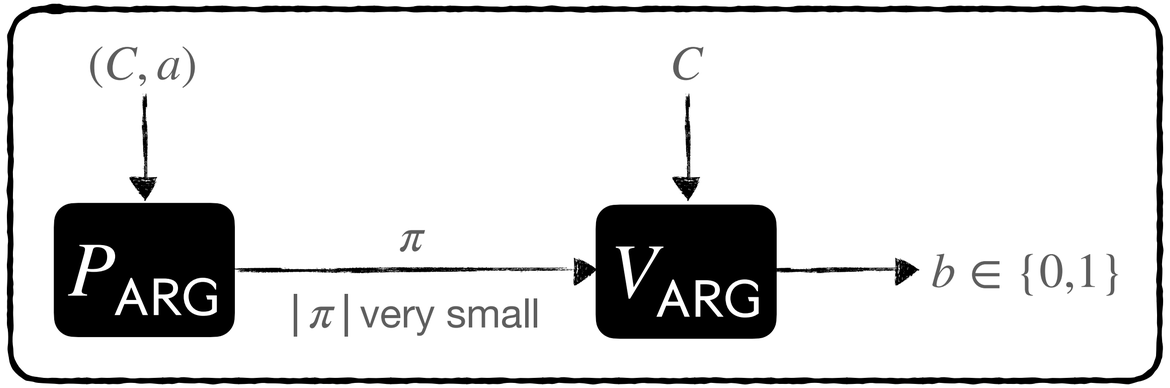
$$\text{CSAT} := \{(C, a) : C(a) = 1\}$$



# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

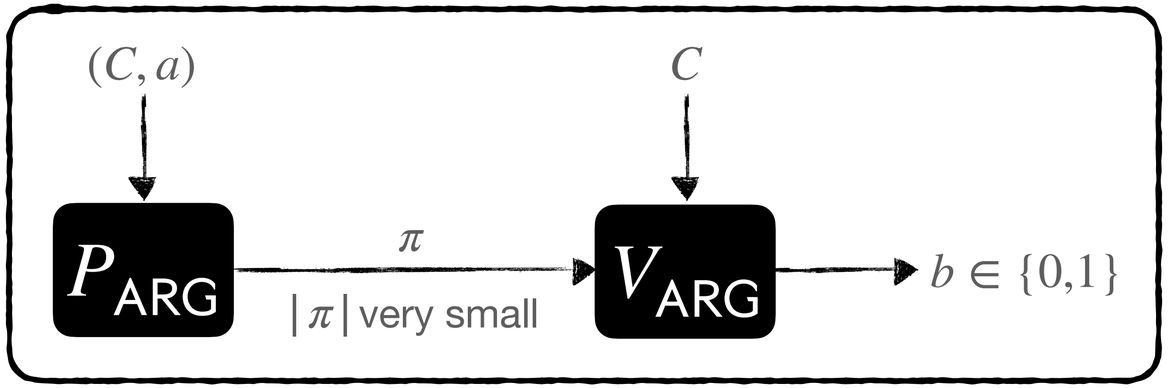
$$\text{CSAT} := \{(C, a) : C(a) = 1\}$$



# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

$$\text{CSAT} := \{(C, a) : C(a) = 1\}$$




$\lambda$: security parameter

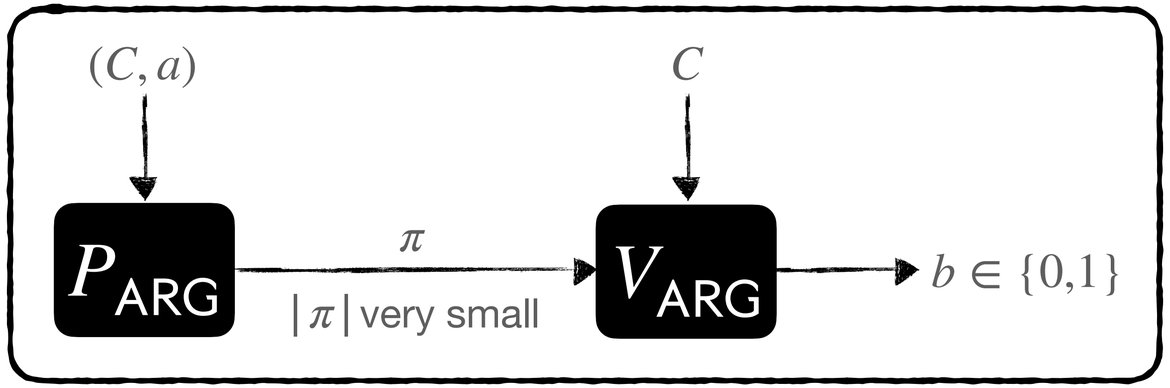
$n$: upper bound on size of $C$

Knowledge soundness: ∀ bounded $\tilde{P}_{\text{ARG}}$, ∃ an efficient extractor $E_{\tilde{P}_{\text{ARG}}}$ who outputs an assignment $a$ such that the following happens with probability at most $\kappa_{\text{ARG}}(\lambda, n)$:

# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

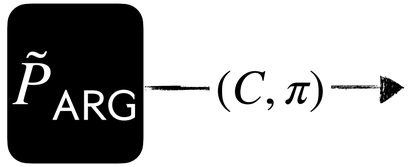
$$\text{CSAT} := \{(C, a) : C(a) = 1\}$$




$\lambda$: security parameter

$n$: upper bound on size of $C$

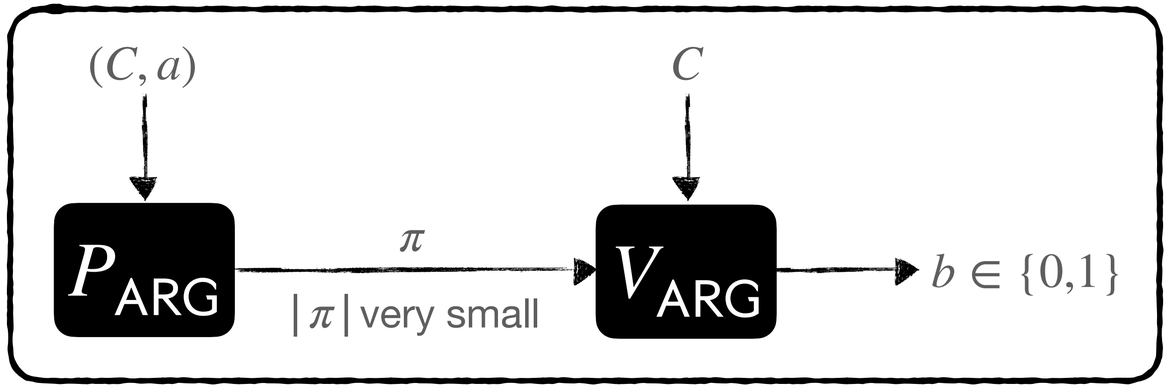
Knowledge soundness: ∀ bounded $\tilde{P}_{\text{ARG}}$, ∃ an efficient extractor $E_{\tilde{P}_{\text{ARG}}}$ who outputs an assignment $a$ such that the following happens with probability at most $\kappa_{\text{ARG}}(\lambda, n)$:



# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

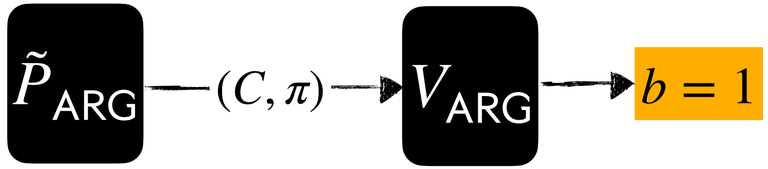
$$\text{CSAT} := \{(C, a) : C(a) = 1\}$$




$\lambda$: security parameter

$n$: upper bound on size of $C$

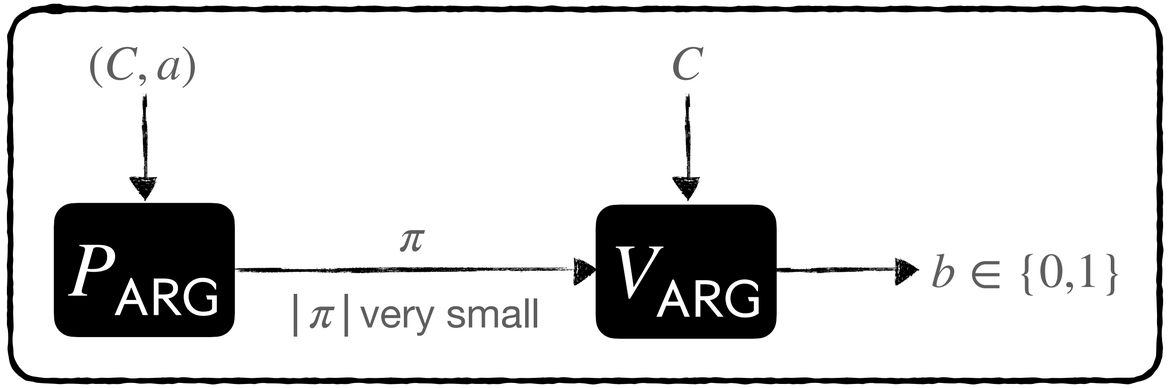
Knowledge soundness: ∀ bounded $\tilde{P}_{\text{ARG}}$, ∃ an efficient extractor $E_{\tilde{P}_{\text{ARG}}}$ who outputs an assignment $a$ such that the following happens with probability at most $\kappa_{\text{ARG}}(\lambda, n)$:



# Review: SNARK

PCD can be constructed from a SNARK (e.g., for CSAT).

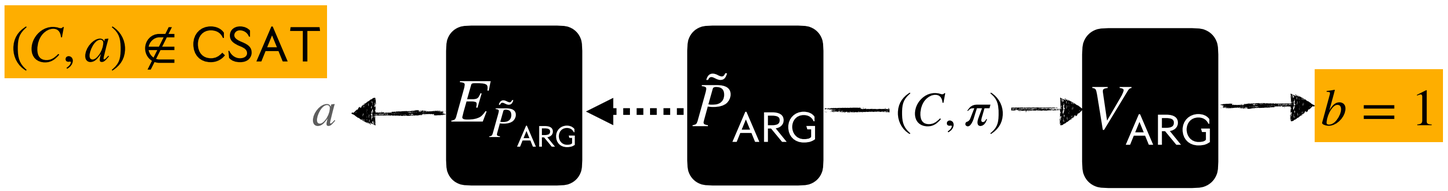
$$\text{CSAT} := \{(C, a) : C(a) = 1\}$$




$\lambda$: security parameter

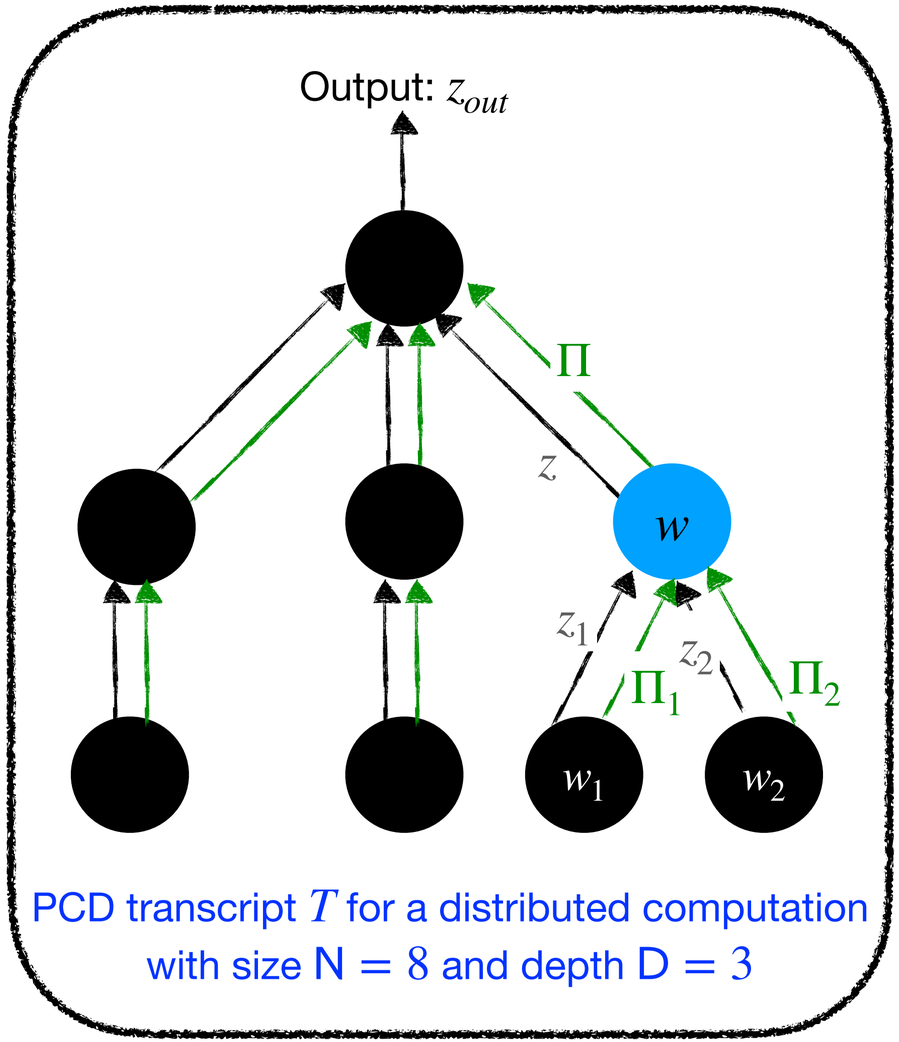
$n$: upper bound on size of $C$

Knowledge soundness: ∀ bounded $\tilde{P}_{\text{ARG}}$, ∃ an efficient extractor $E_{\tilde{P}_{\text{ARG}}}$ who outputs an assignment $a$ such that the following happens with probability at most $\kappa_{\text{ARG}}(\lambda, n)$:



# Naive PCD construction

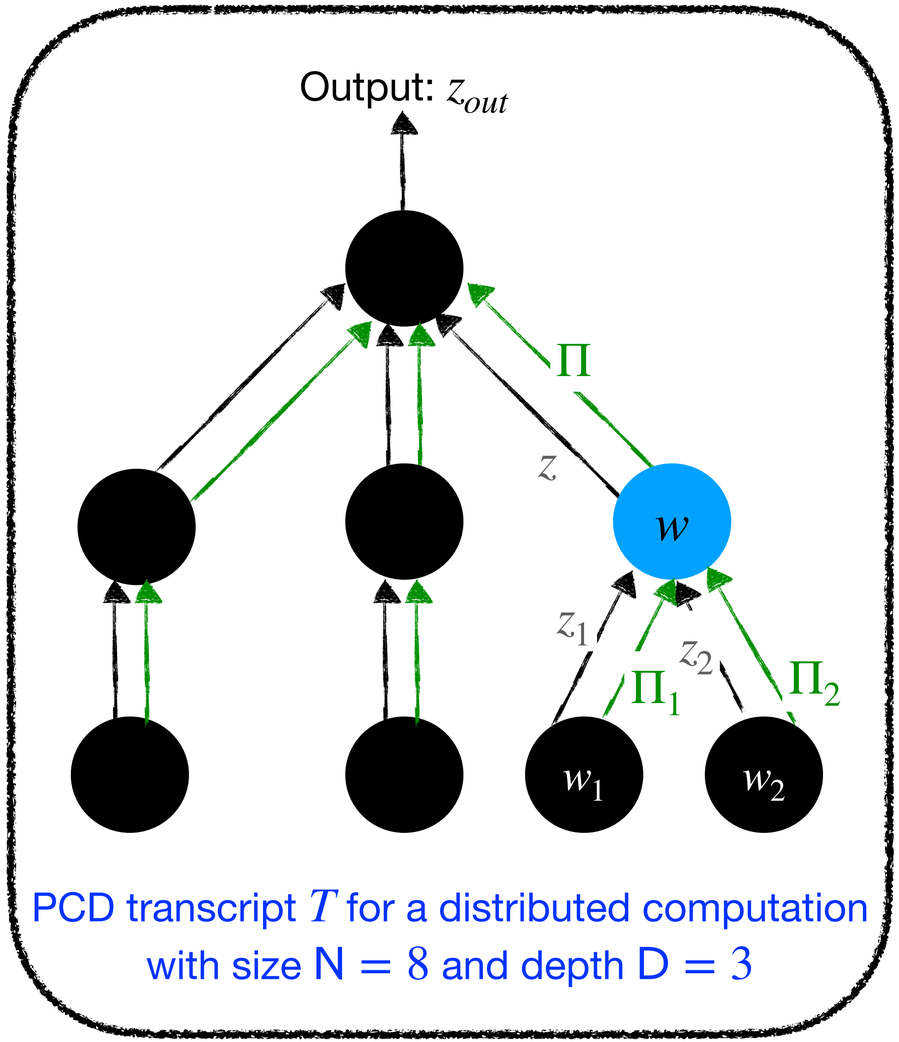
## Concatenate SNARK proofs



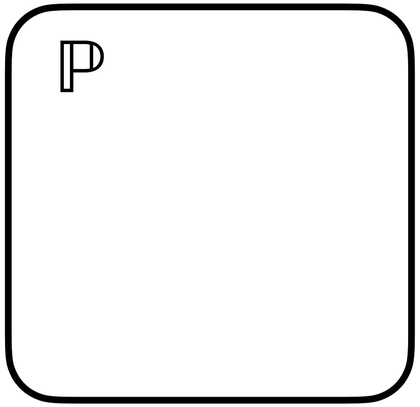

PCD transcript $T$ for a distributed computation
with size N $= 8$ and depth D $= 3$

# Naive PCD construction

## Concatenate SNARK proofs

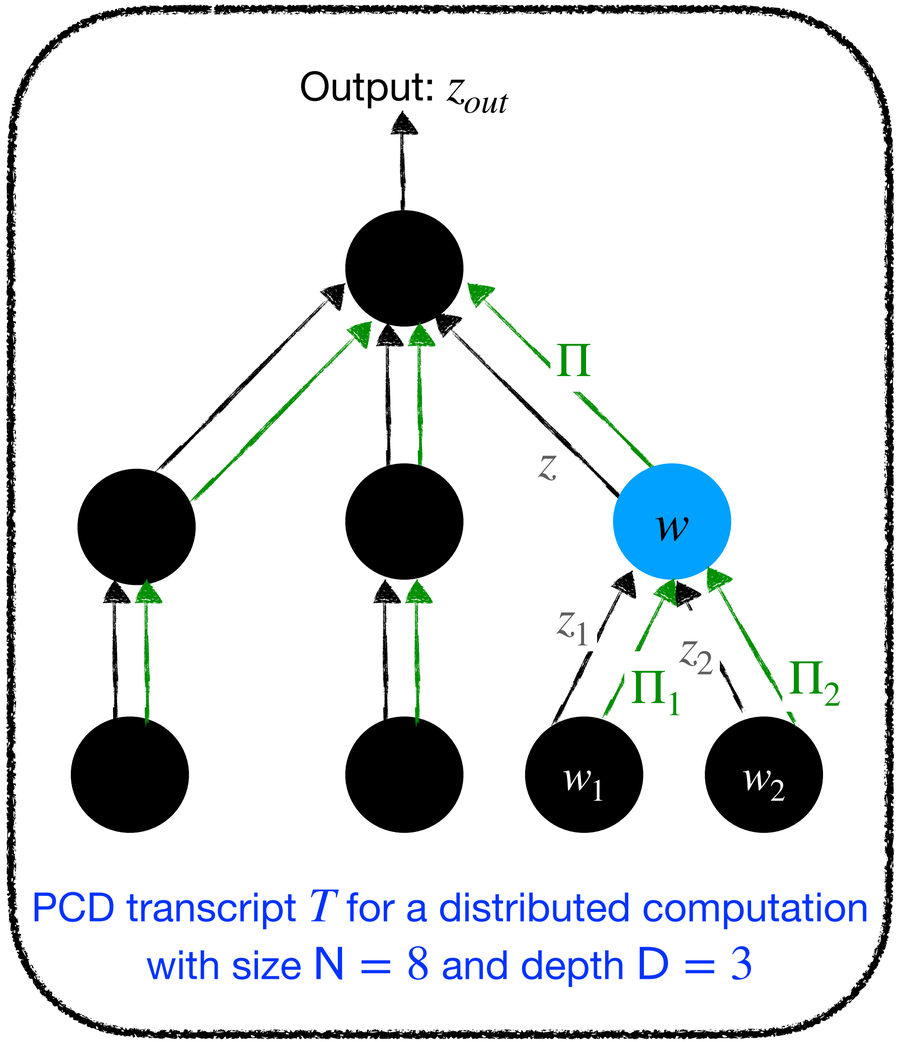



PCD transcript $T$ for a distributed computation with size $N = 8$ and depth $D = 3$



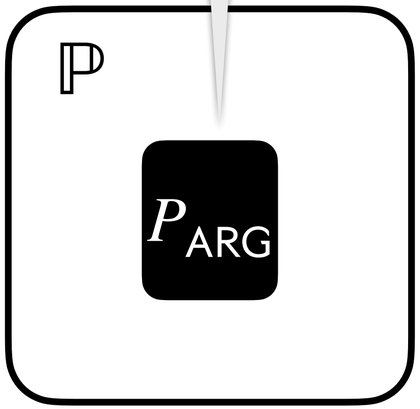
# Naive PCD construction

## Concatenate SNARK proofs



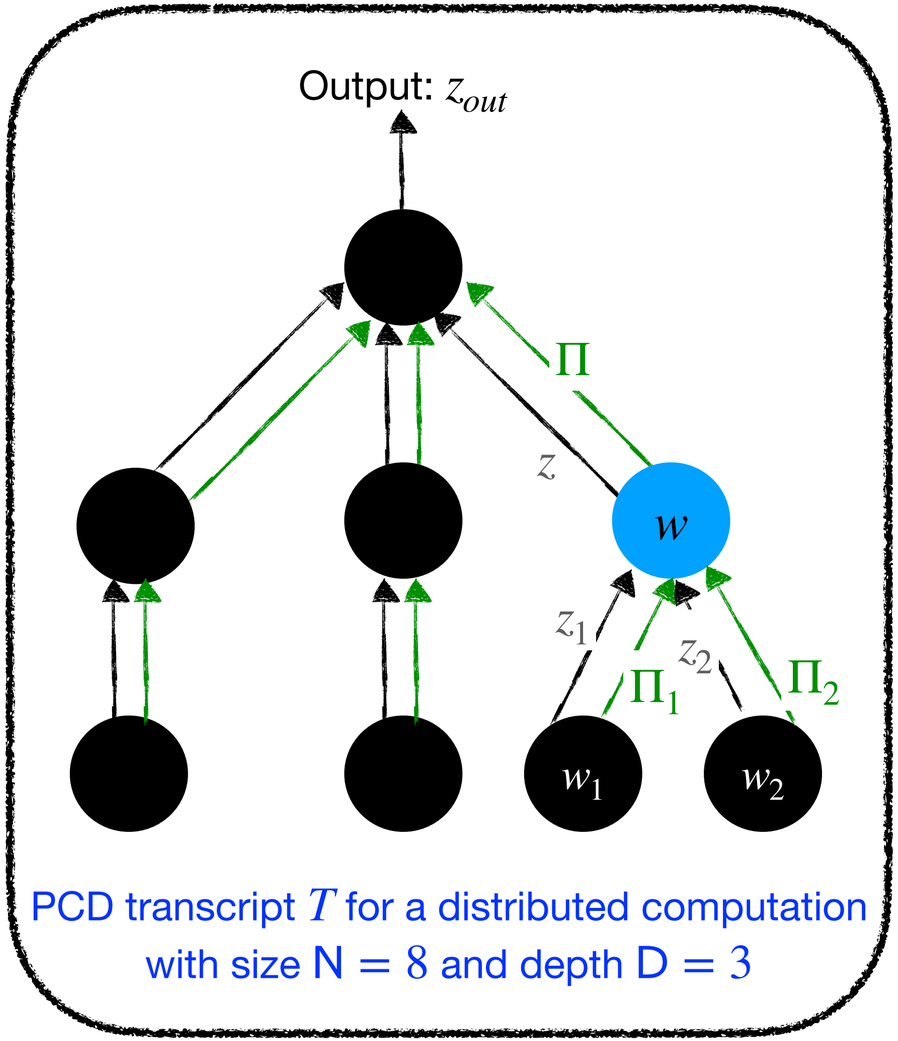

PCD transcript $T$ for a distributed computation with size N $= 8$ and depth D $= 3$

SNARK prover for compliance predicate $\phi$



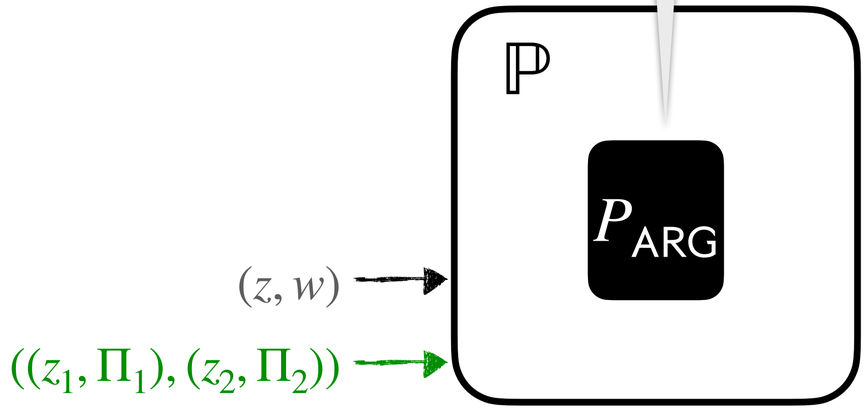
# Naive PCD construction

## Concatenate SNARK proofs



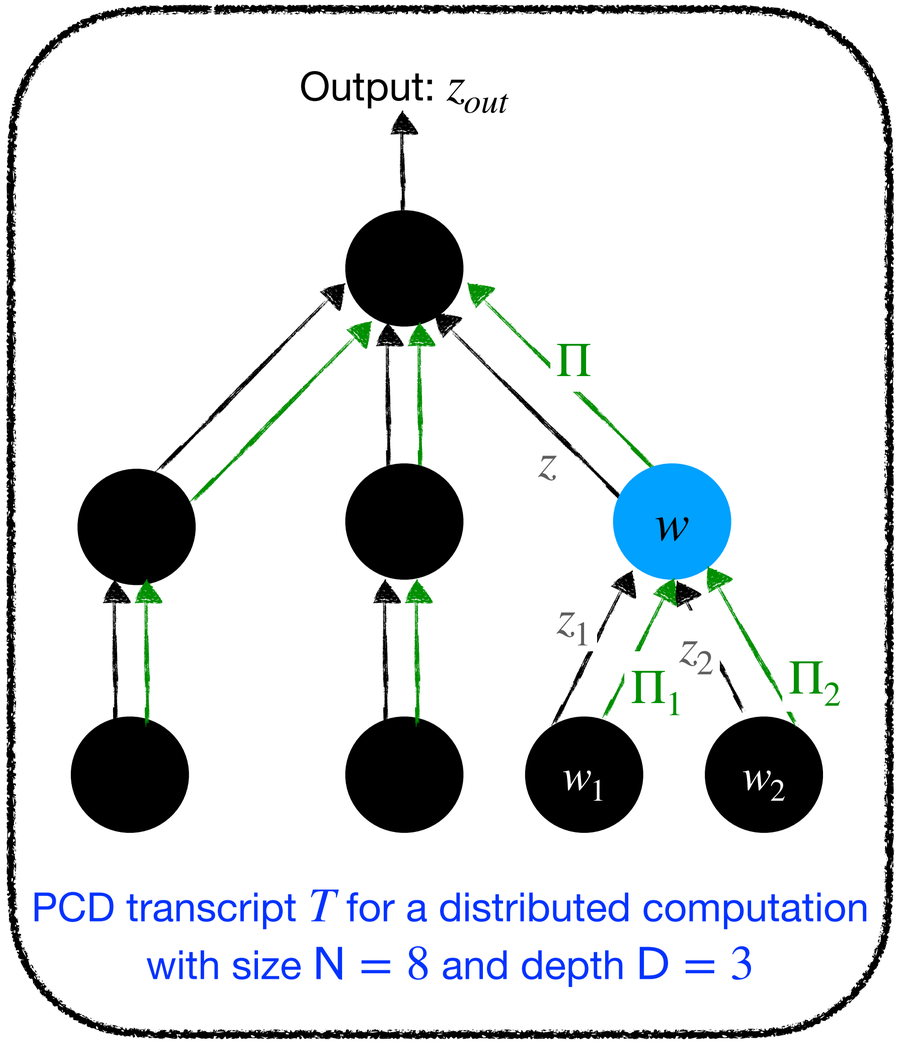

PCD transcript $T$ for a distributed computation with size N $= 8$ and depth D $= 3$

SNARK prover for compliance predicate $\phi$



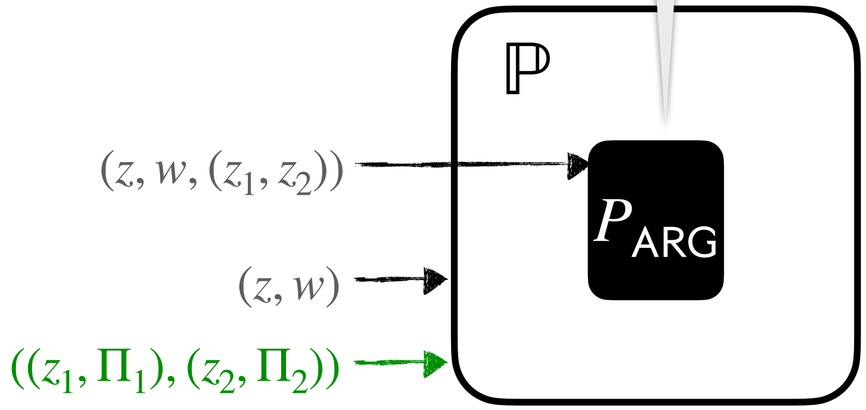
# Naive PCD construction

## Concatenate SNARK proofs



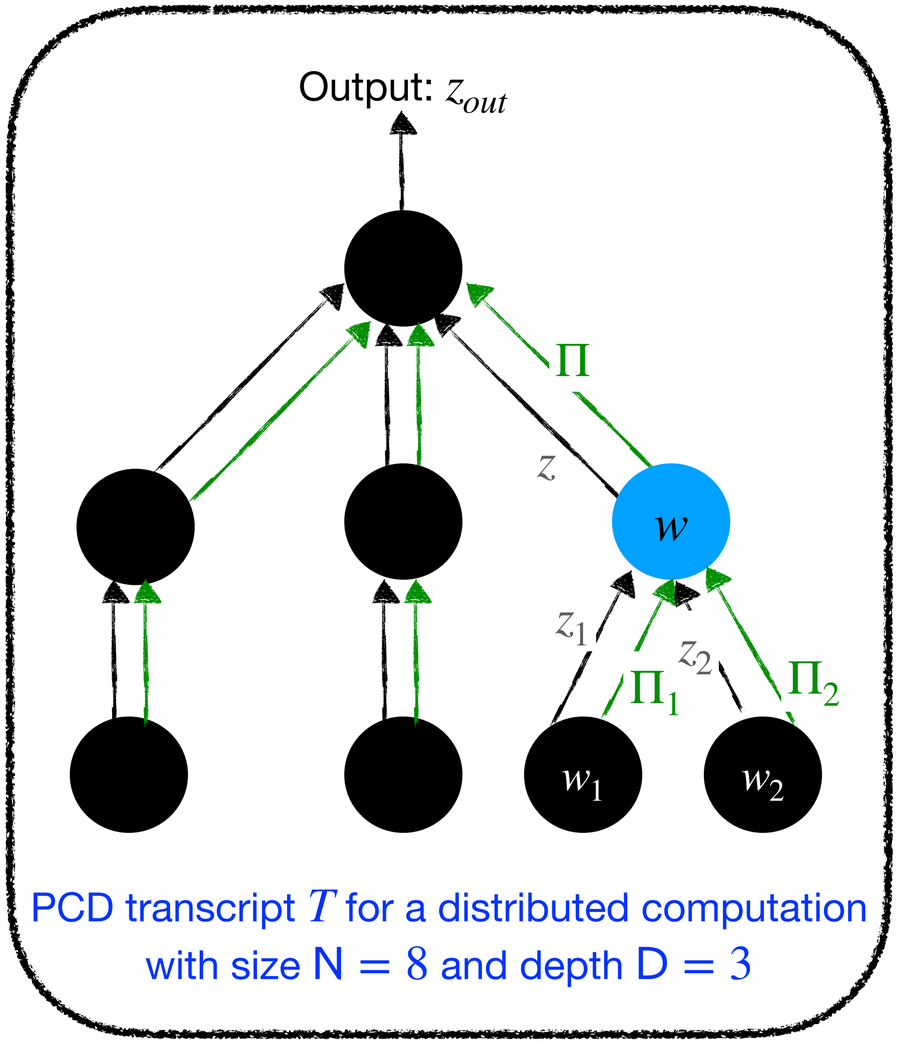

PCD transcript $T$ for a distributed computation with size N $= 8$ and depth D $= 3$

SNARK prover for compliance predicate $\phi$



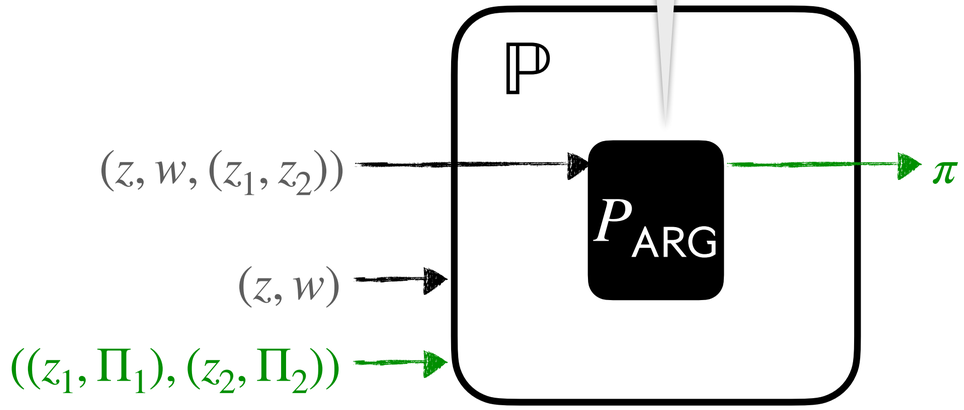
# Naive PCD construction

## Concatenate SNARK proofs



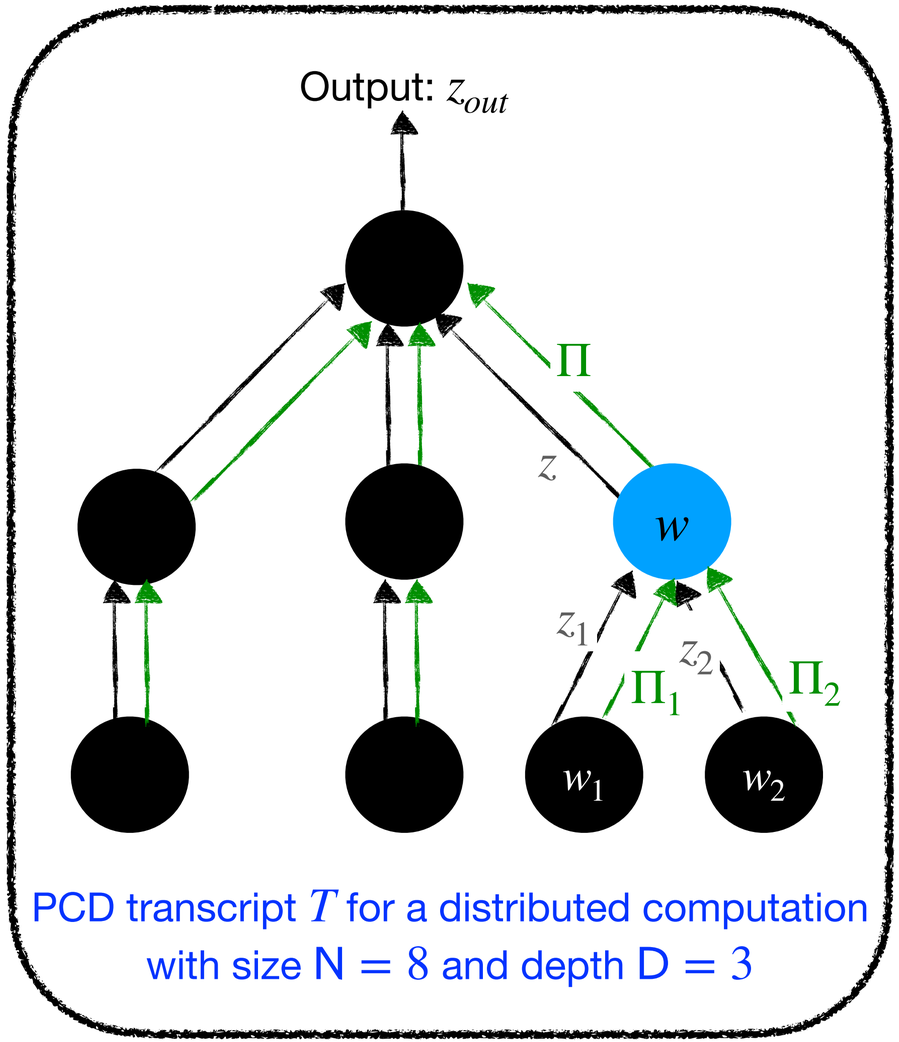

PCD transcript $T$ for a distributed computation with size N $= 8$ and depth D $= 3$

SNARK prover for compliance predicate $\phi$



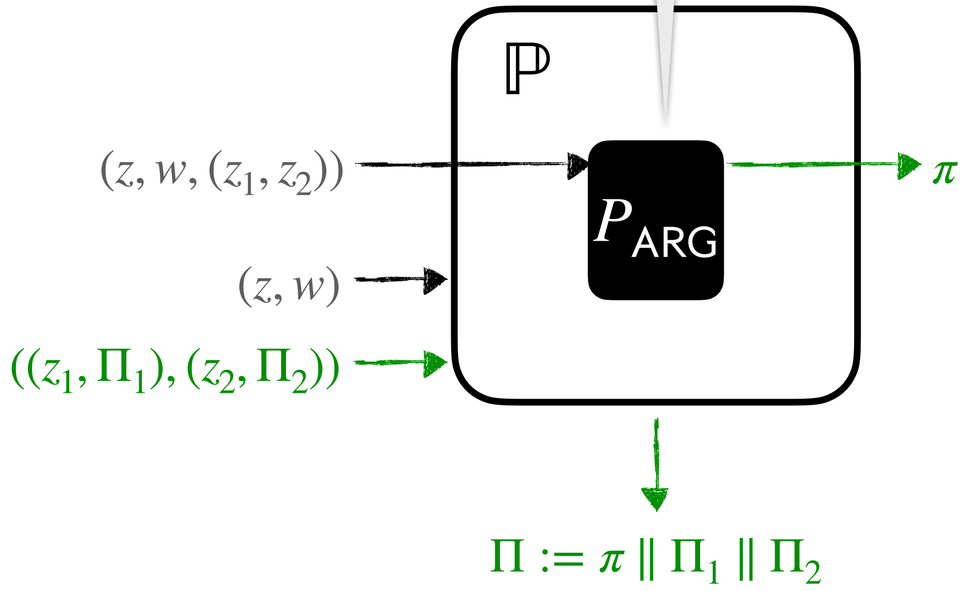
# Naive PCD construction

## Concatenate SNARK proofs



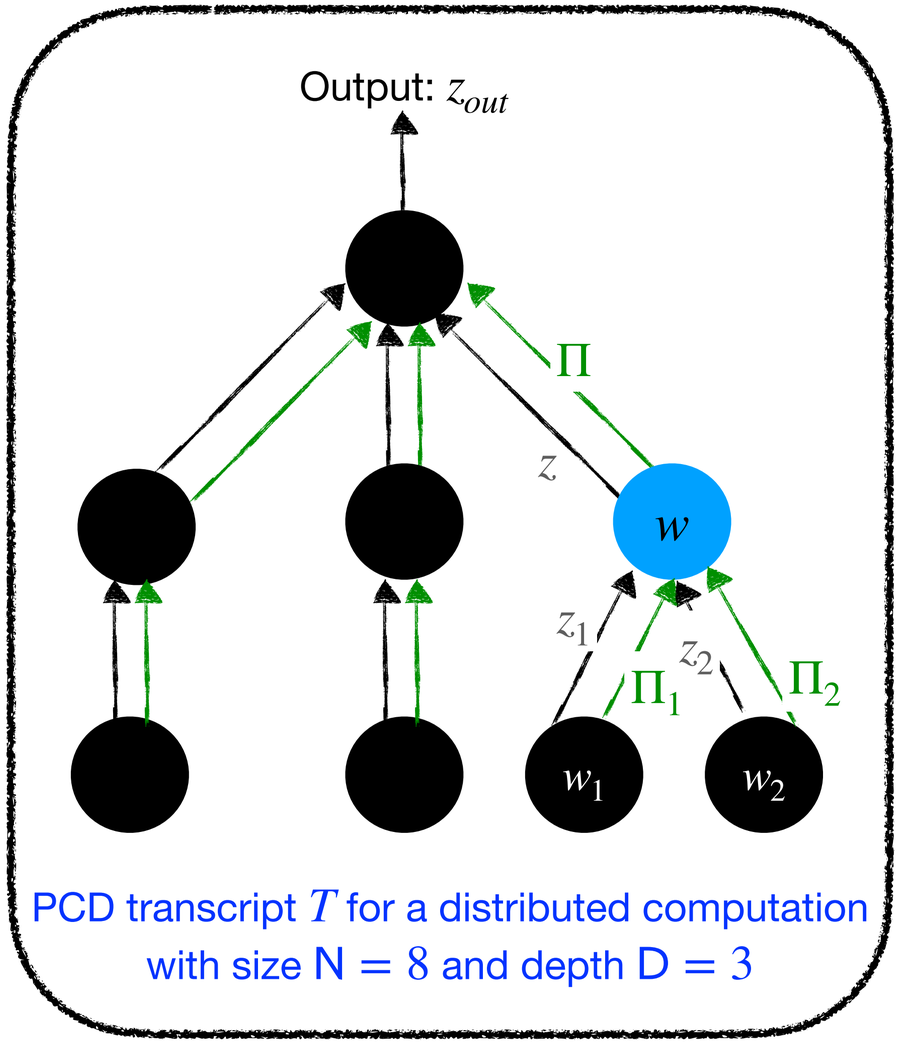

PCD transcript $T$ for a distributed computation with size N $= 8$ and depth D $= 3$

SNARK prover for compliance predicate $\phi$

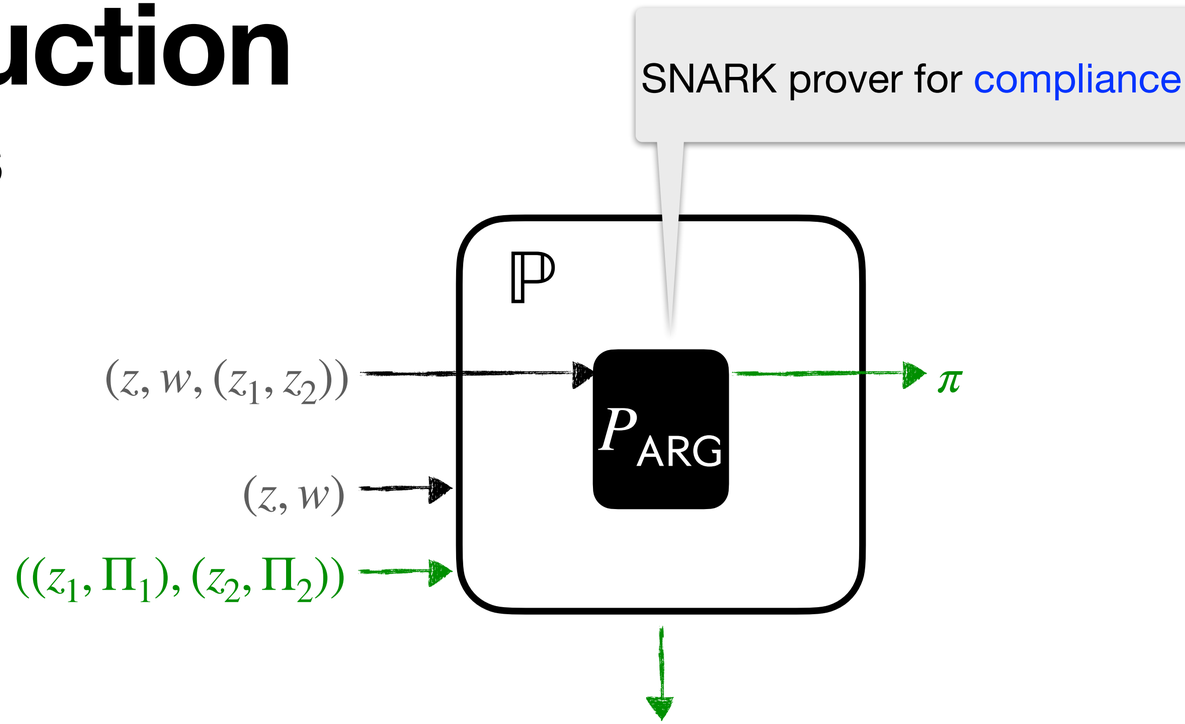


# Naive PCD construction

## Concatenate SNARK proofs




PCD transcript $T$ for a distributed computation with size $N = 8$ and depth $D = 3$



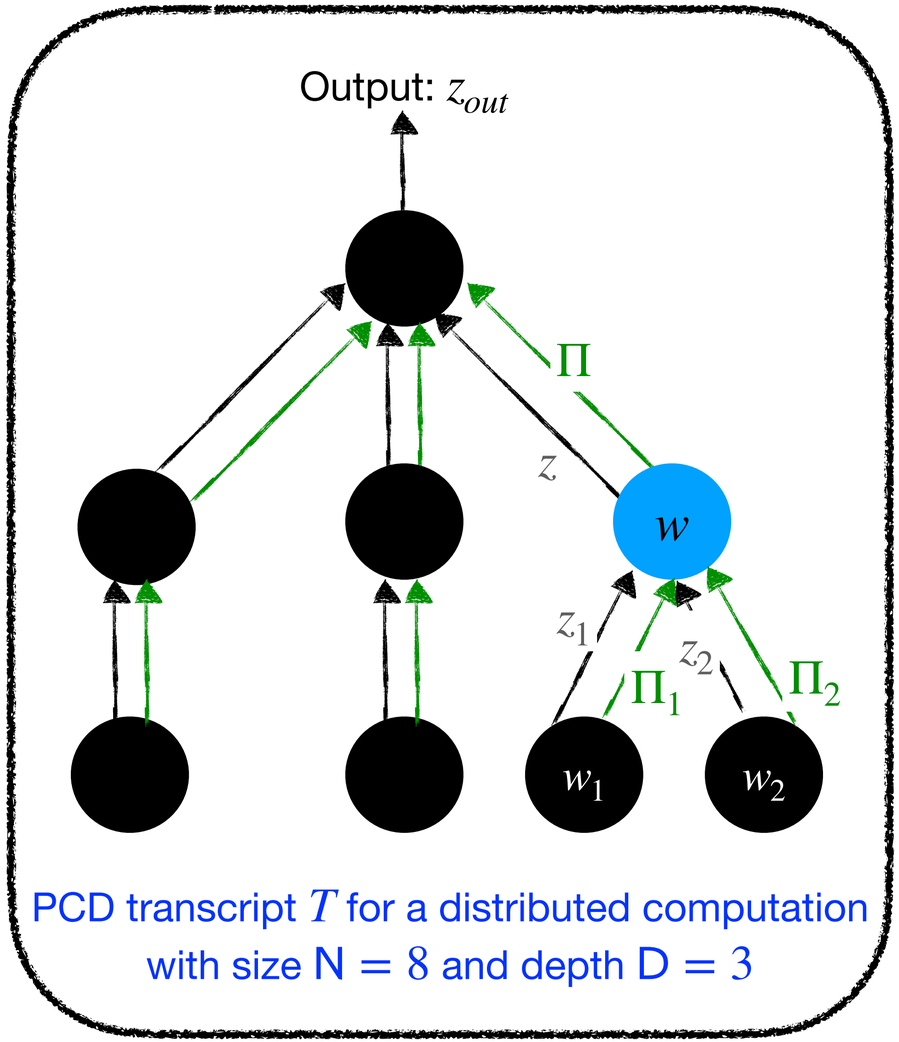

$\phi(z, w, (z_1, z_2)) = 1$

$\phi(z_1, w_1) = 1$

$\phi(z_2, w_2) = 1$

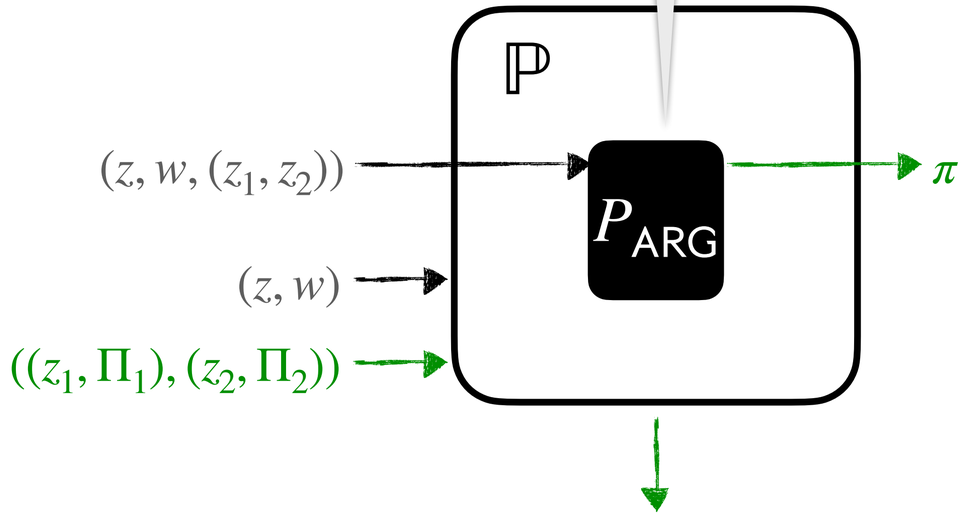
# Naive PCD construction

## Concatenate SNARK proofs




PCD transcript $T$ for a distributed computation with size N $= 8$ and depth D $= 3$

SNARK prover for compliance predicate $\phi$




$\Pi := \pi \parallel \Pi_1 \parallel \Pi_2$

$\phi(z, w, (z_1, z_2)) = 1$

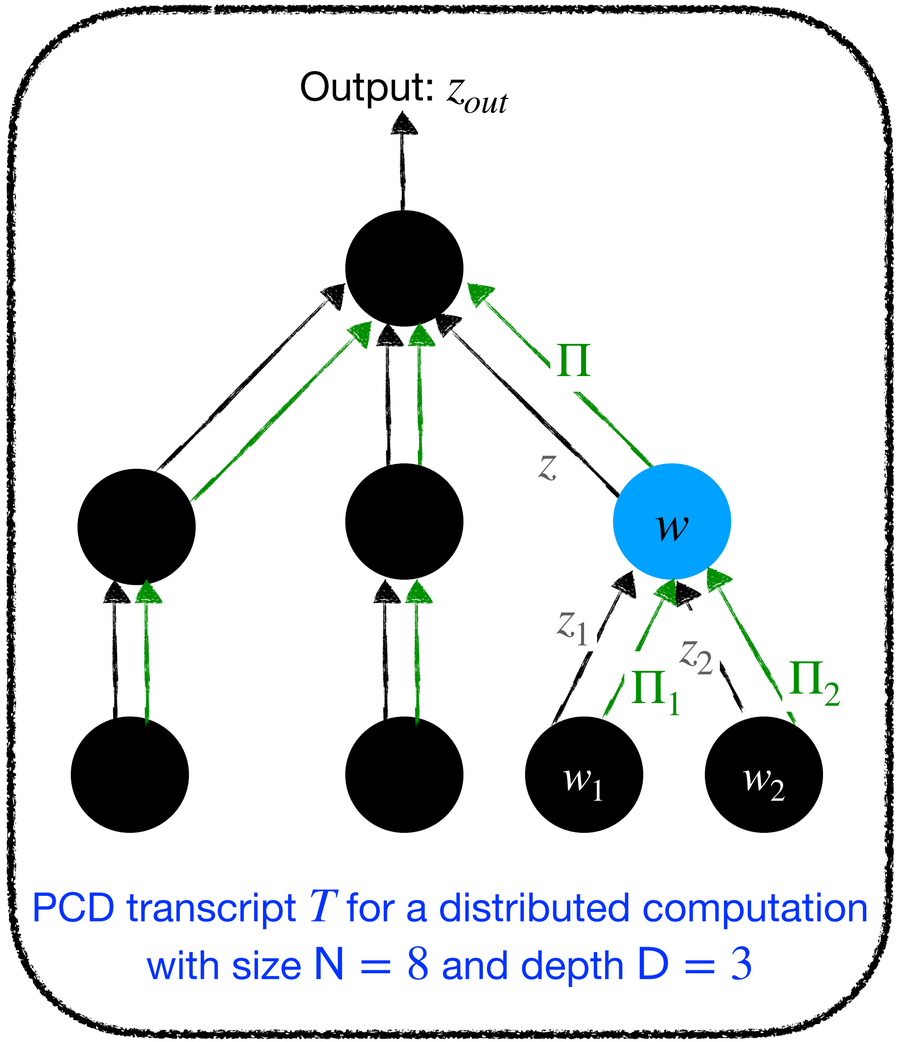
$\phi(z_1, w_1) = 1$

$\phi(z_2, w_2) = 1$

**Pro: as secure as underlying SNARKs!**

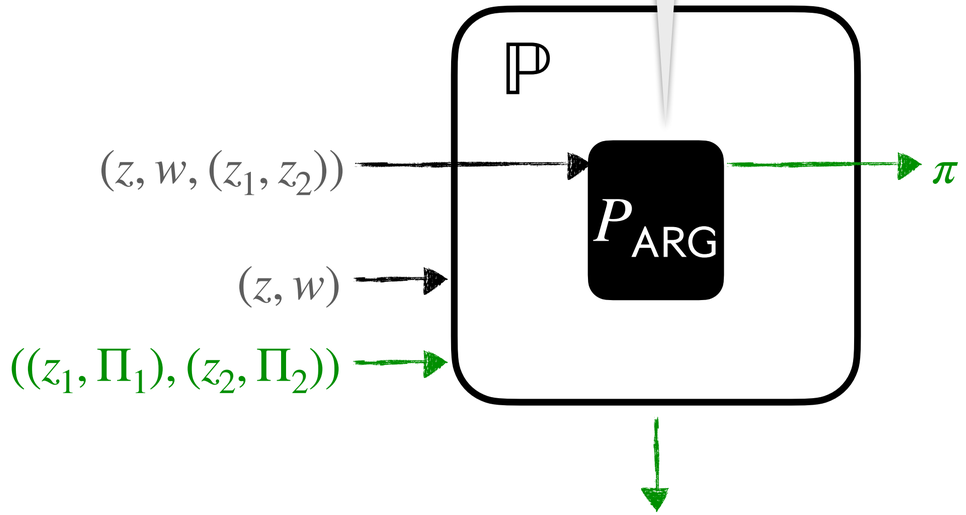
# Naive PCD construction

## Concatenate SNARK proofs




PCD transcript $T$ for a distributed computation with size N $= 8$ and depth D $= 3$

SNARK prover for compliance predicate $\phi$




$\Pi := \pi \parallel \Pi_1 \parallel \Pi_2$

$\phi(z, w, (z_1, z_2)) = 1$

$\phi(z_1, w_1) = 1$

$\phi(z_2, w_2) = 1$

**Pro: as secure as underlying SNARKs!**

**Con: $\Pi$ is NOT succinct (linear in number of vertices)**

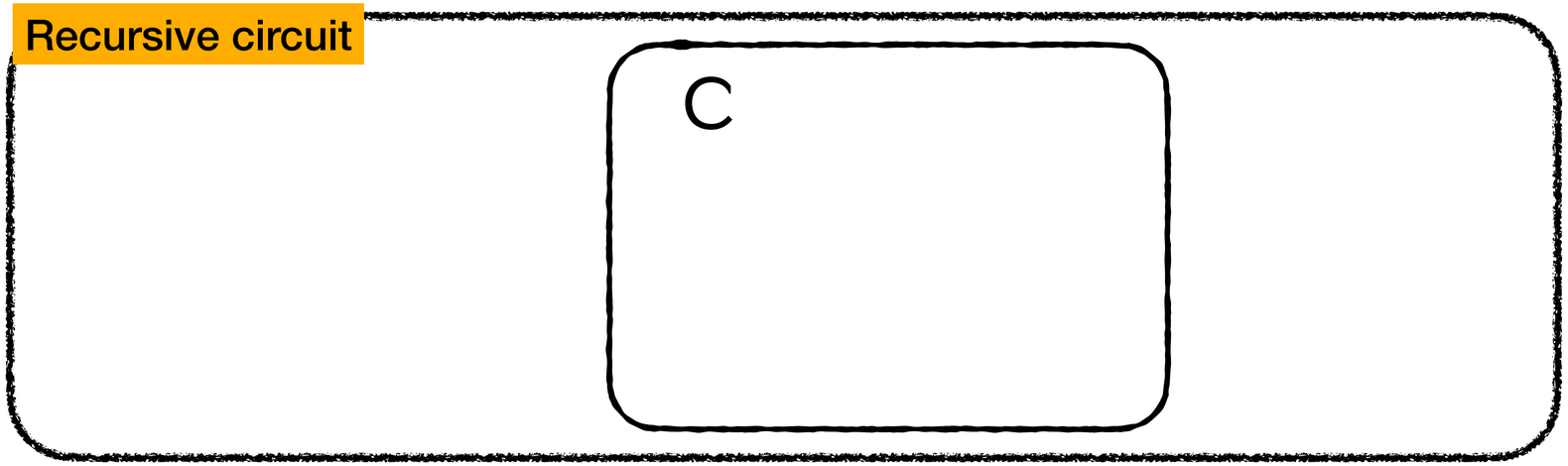
# Recursively compose the SNARK proofs

# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for $\mathsf{CSAT}$) for the recursive circuit $C$.

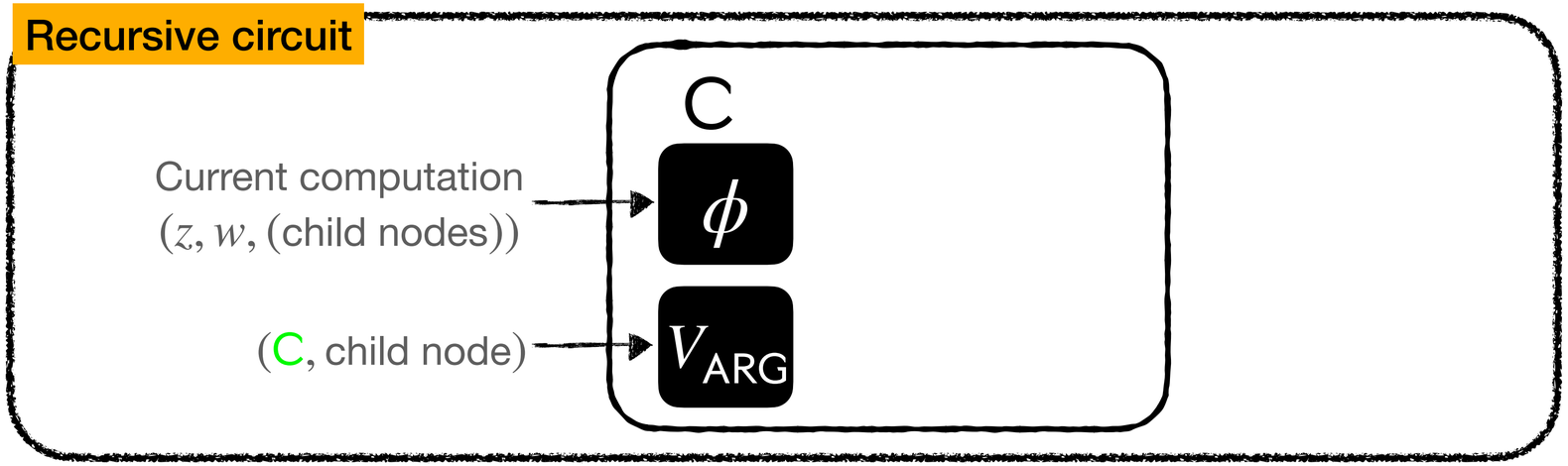
# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



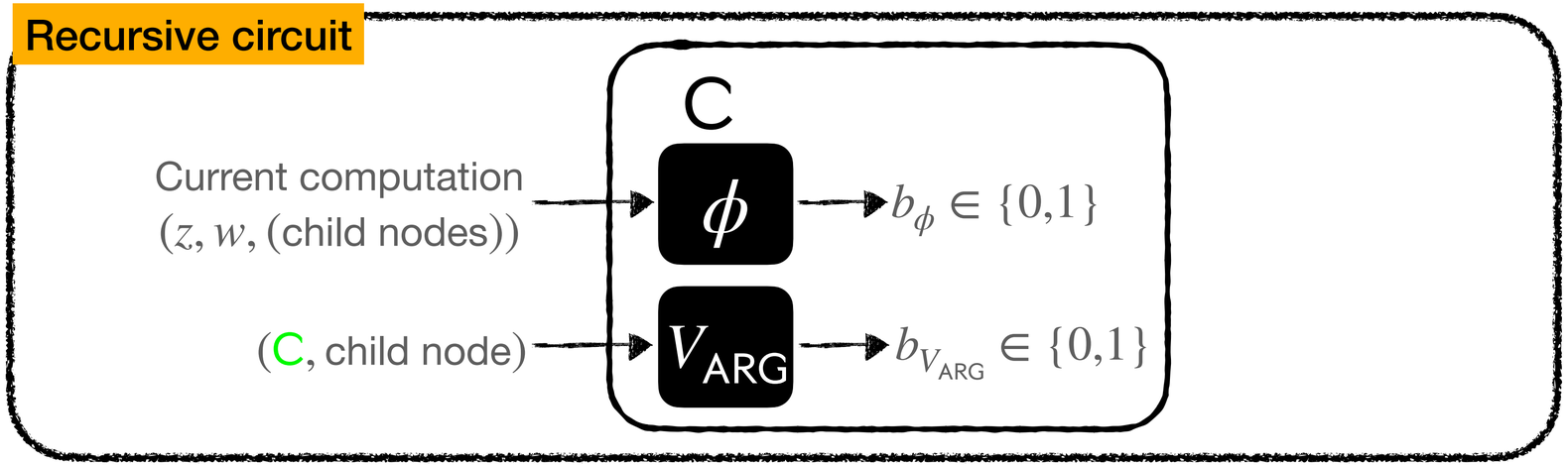
# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



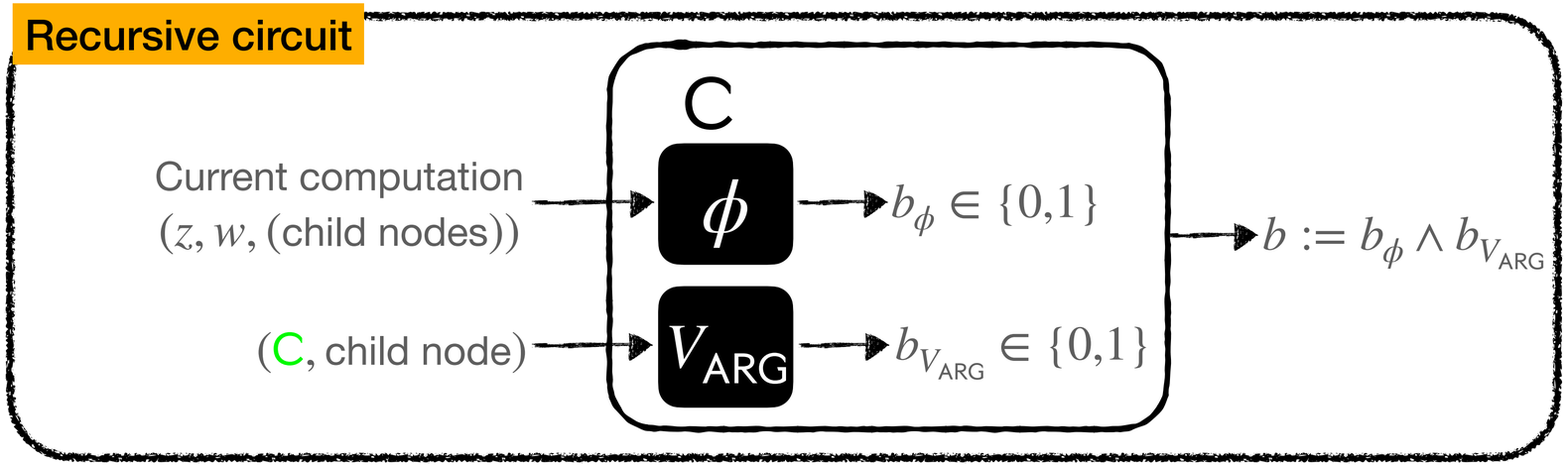
# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



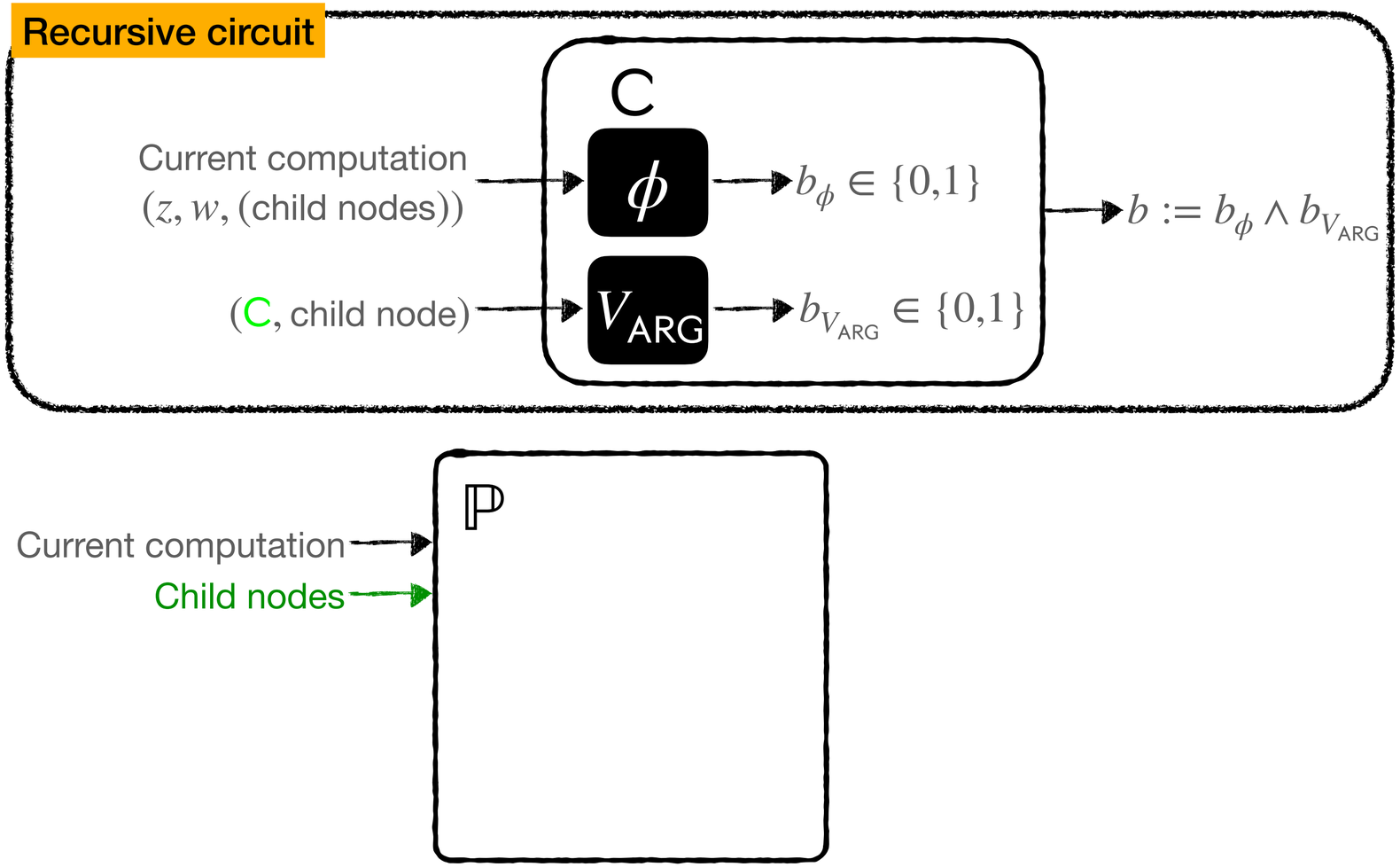
# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



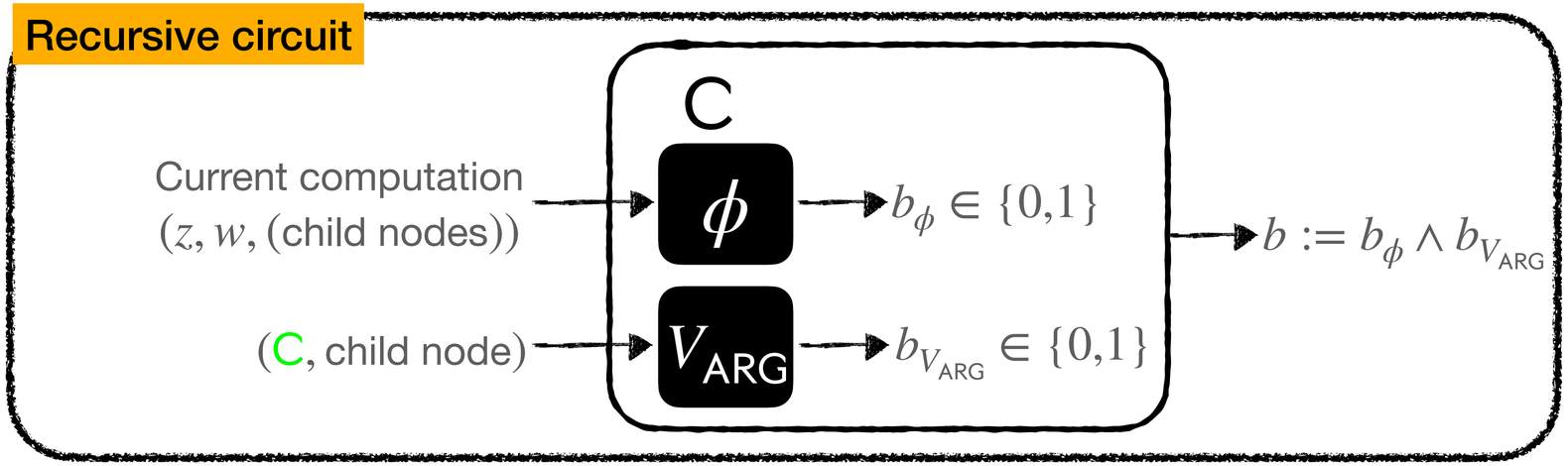
# Recursively compose the SNARK proofs

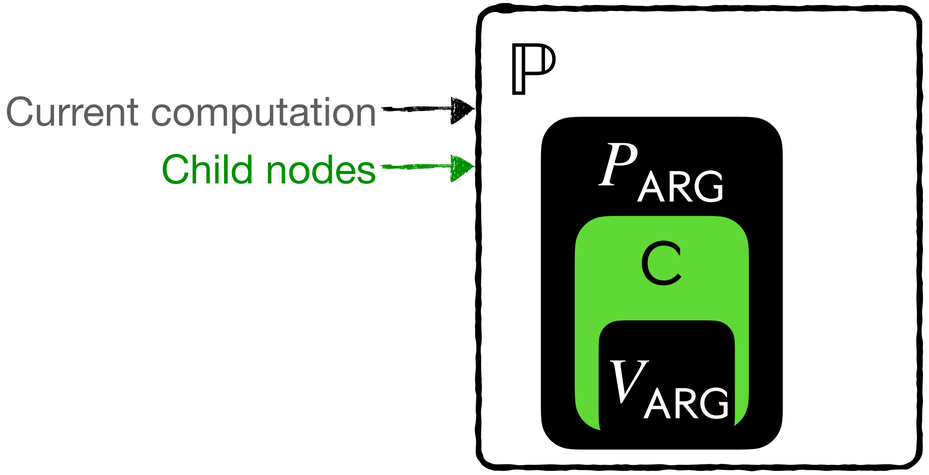
PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



# Recursively compose the SNARK proofs

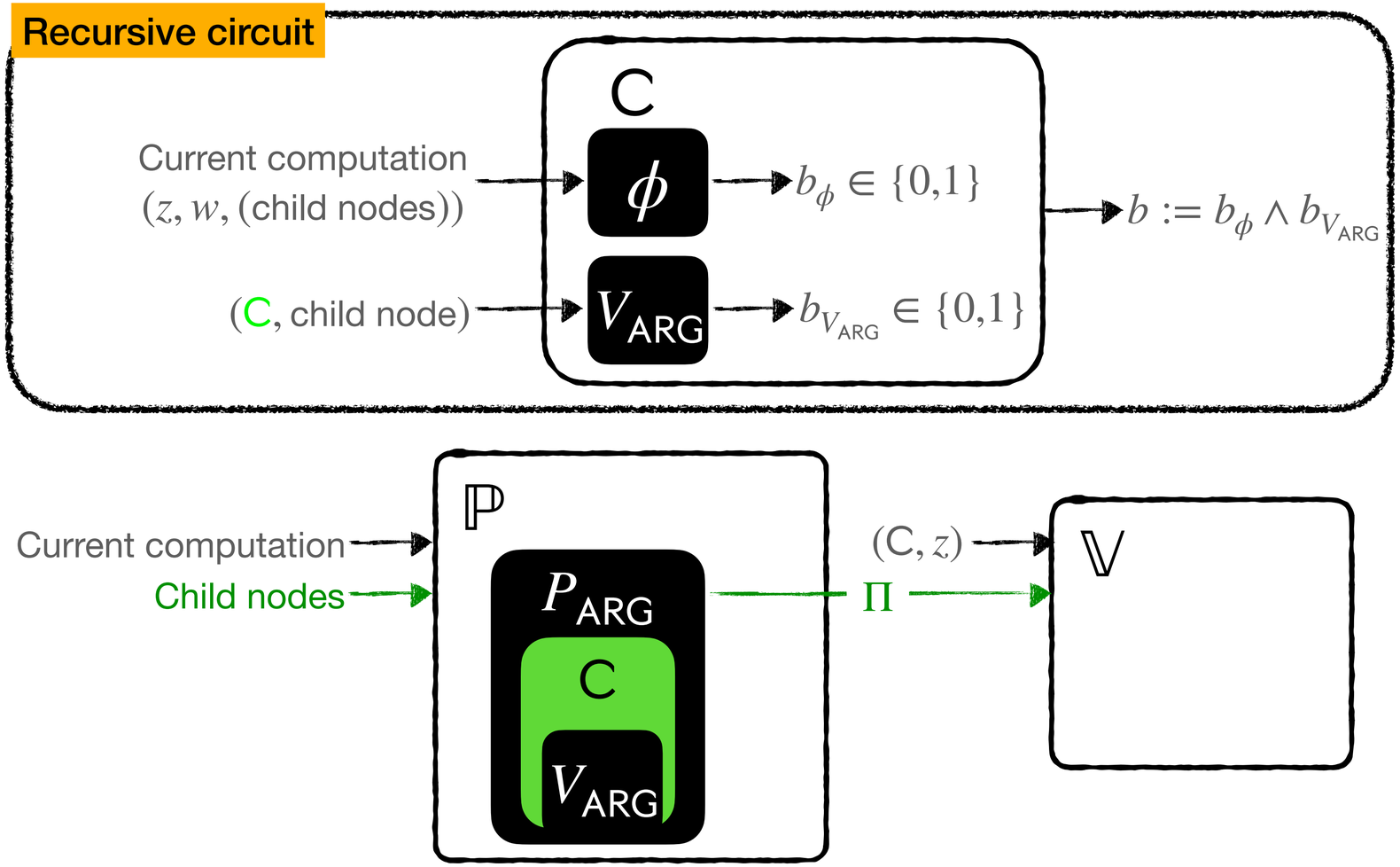
PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



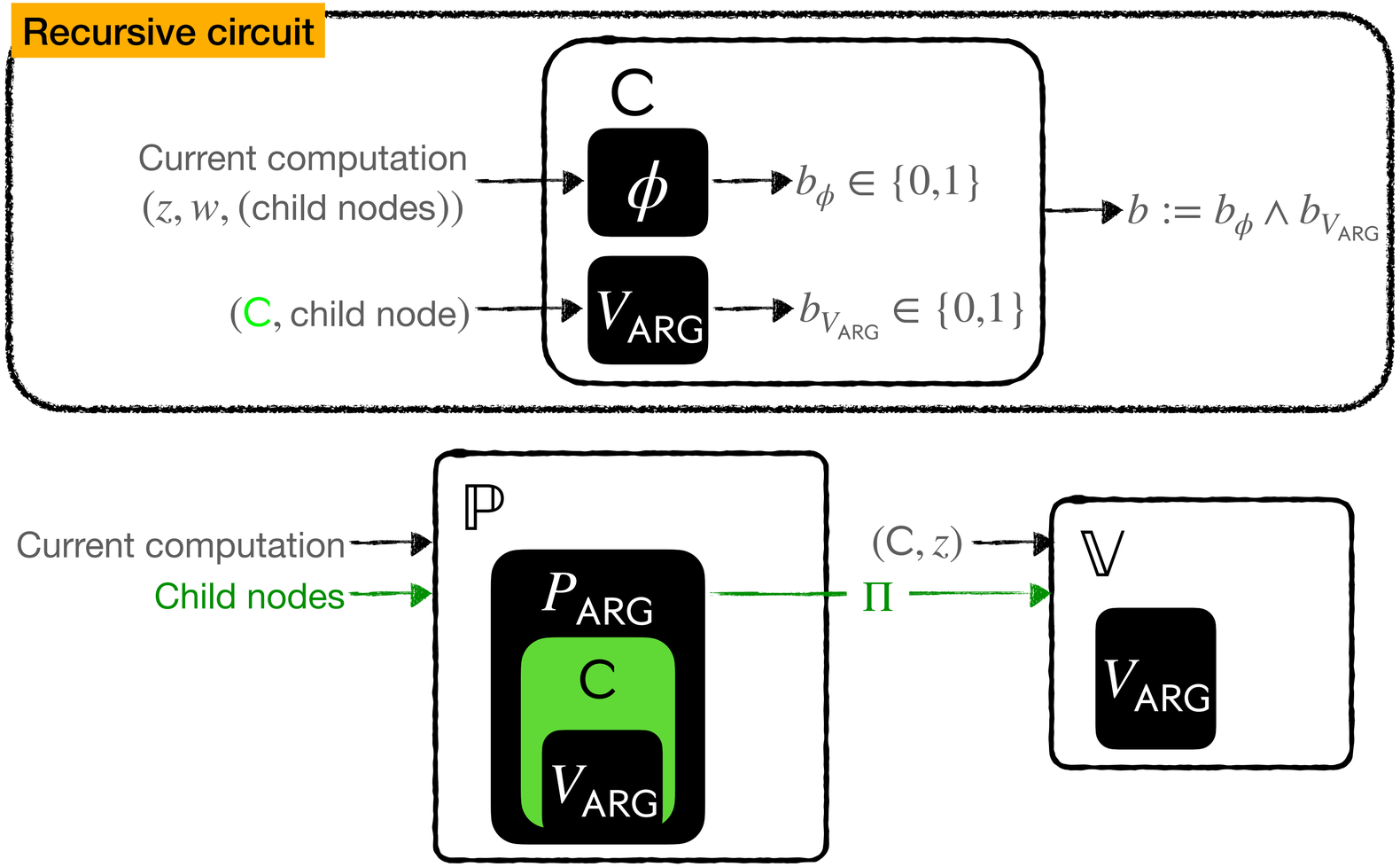
# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



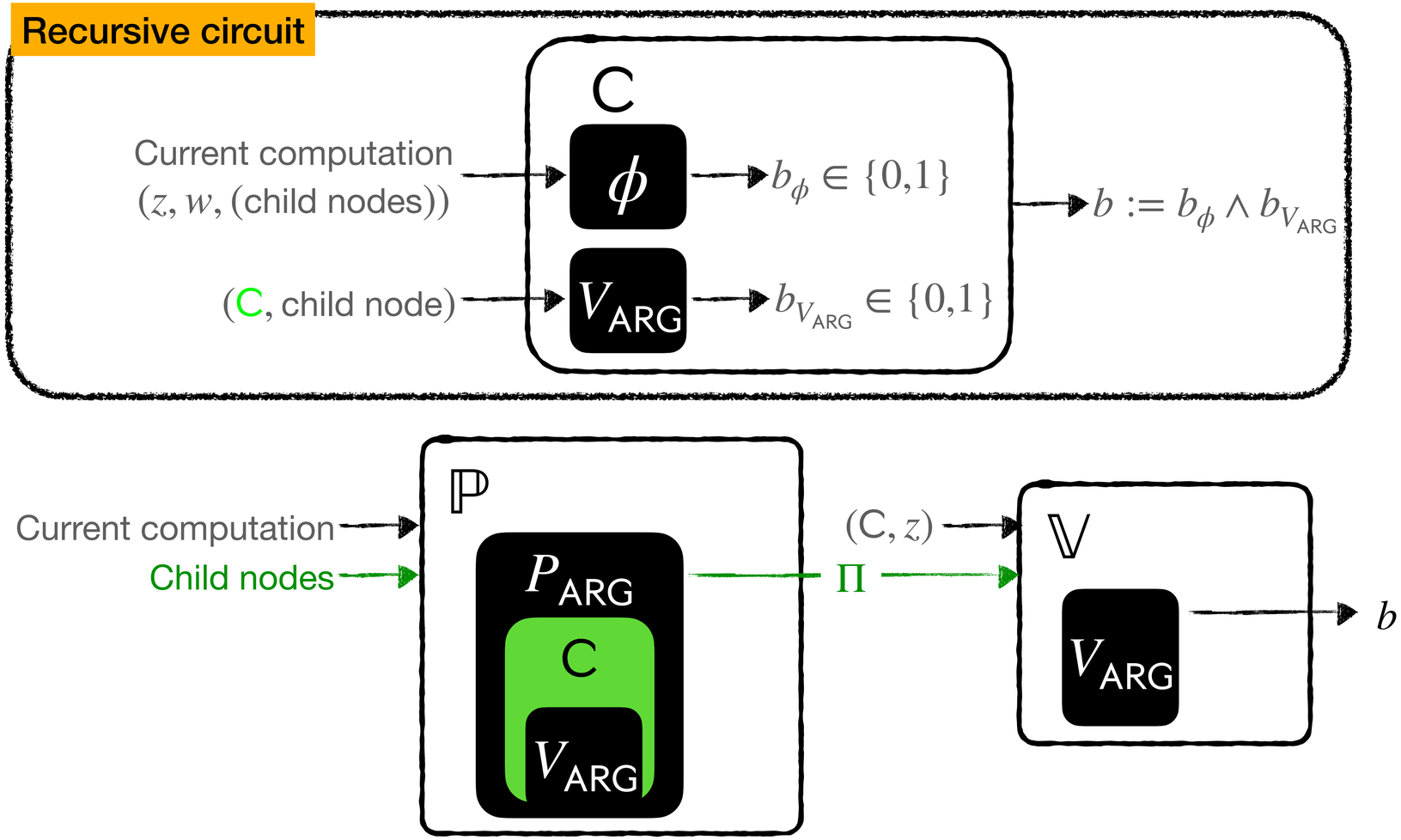
# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



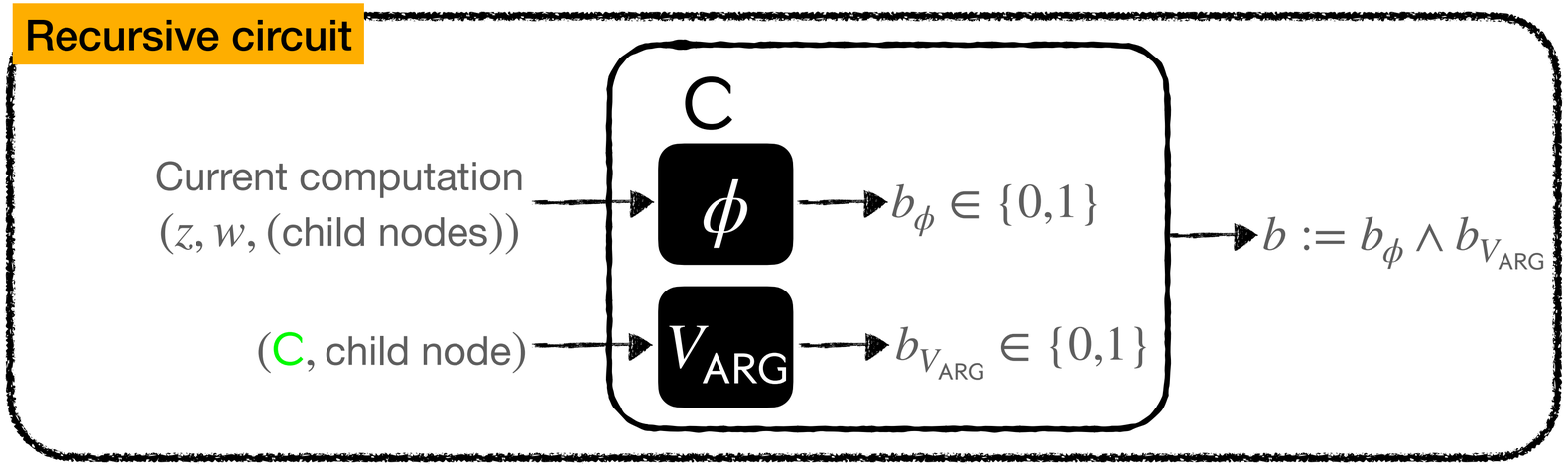
# Recursively compose the SNARK proofs

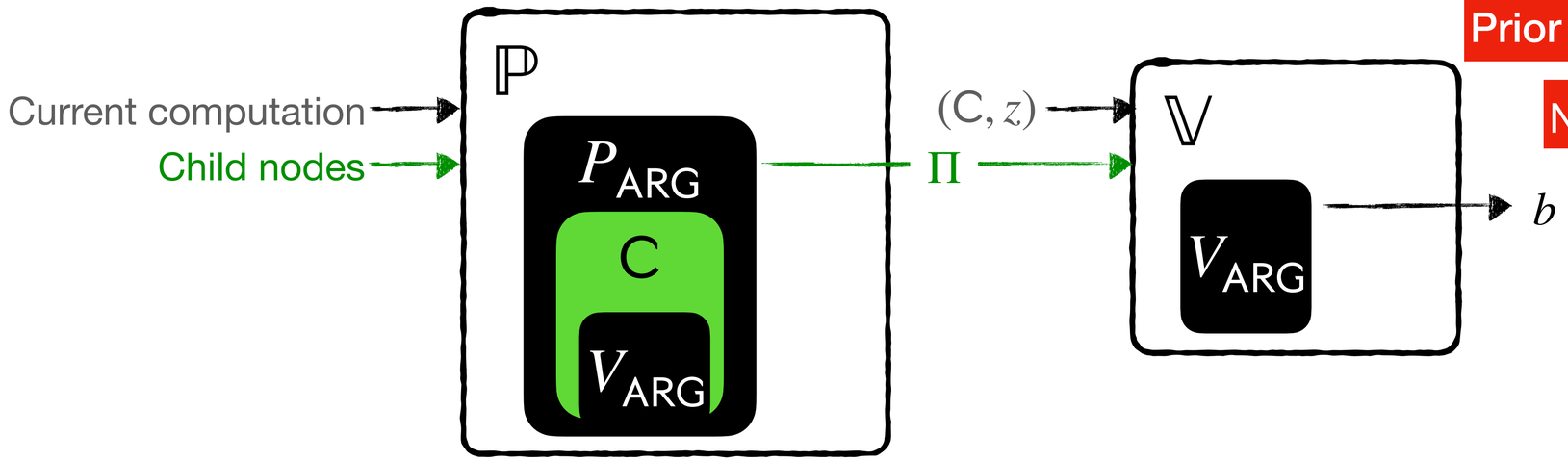
PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



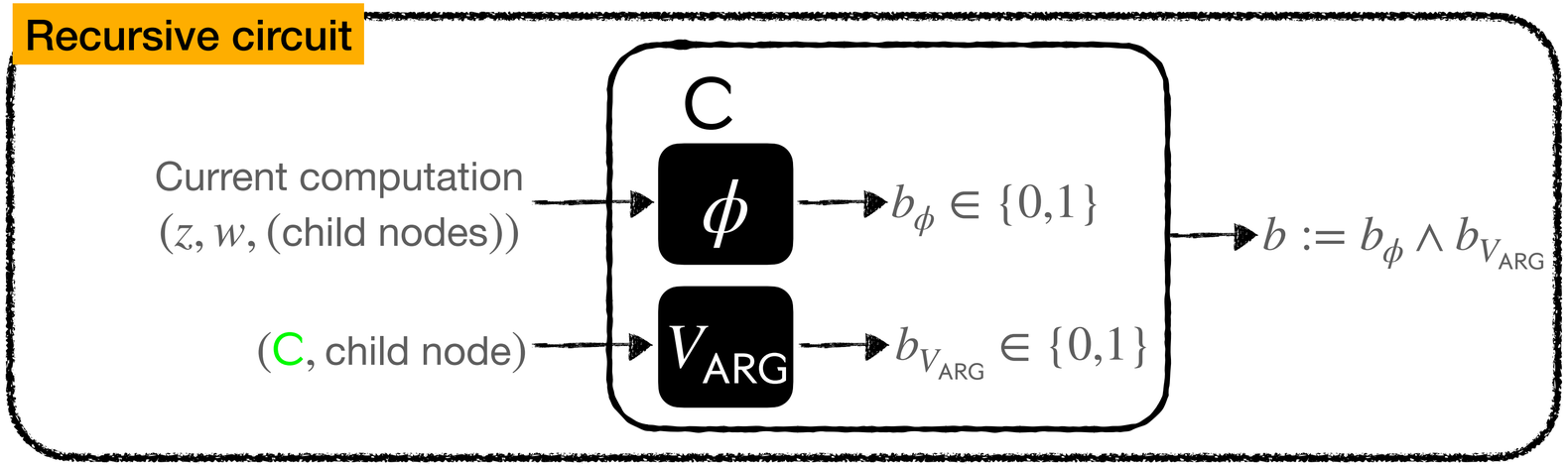



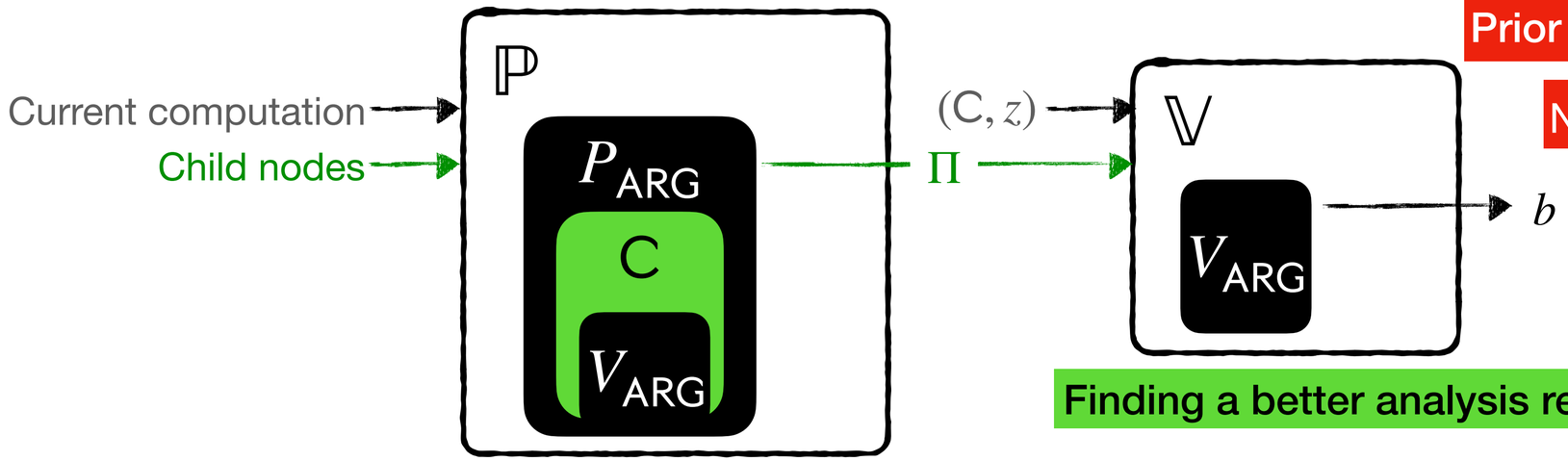

**Prior works:** $\kappa(\lambda, q, D, N) \leq \exp(D) \cdot \kappa_{\text{ARG}}(\lambda, q, N)$

No security when D is larger than constant.

# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.



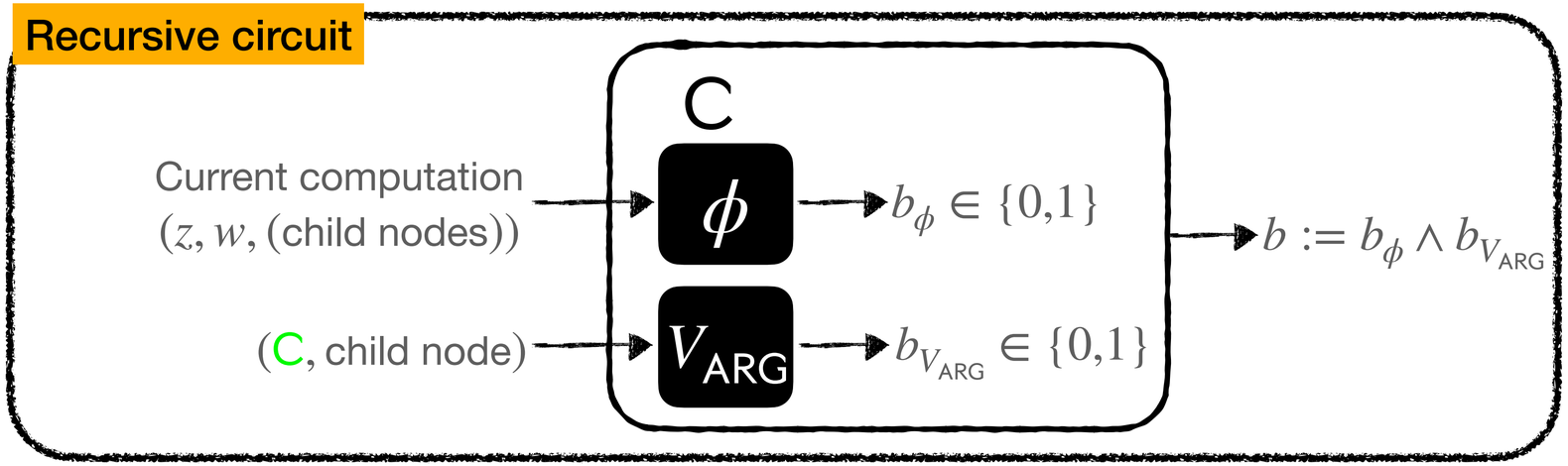

**Prior works:** $\kappa(\lambda, q, D, N) \leq \exp(D) \cdot \kappa_{\text{ARG}}(\lambda, q, N)$

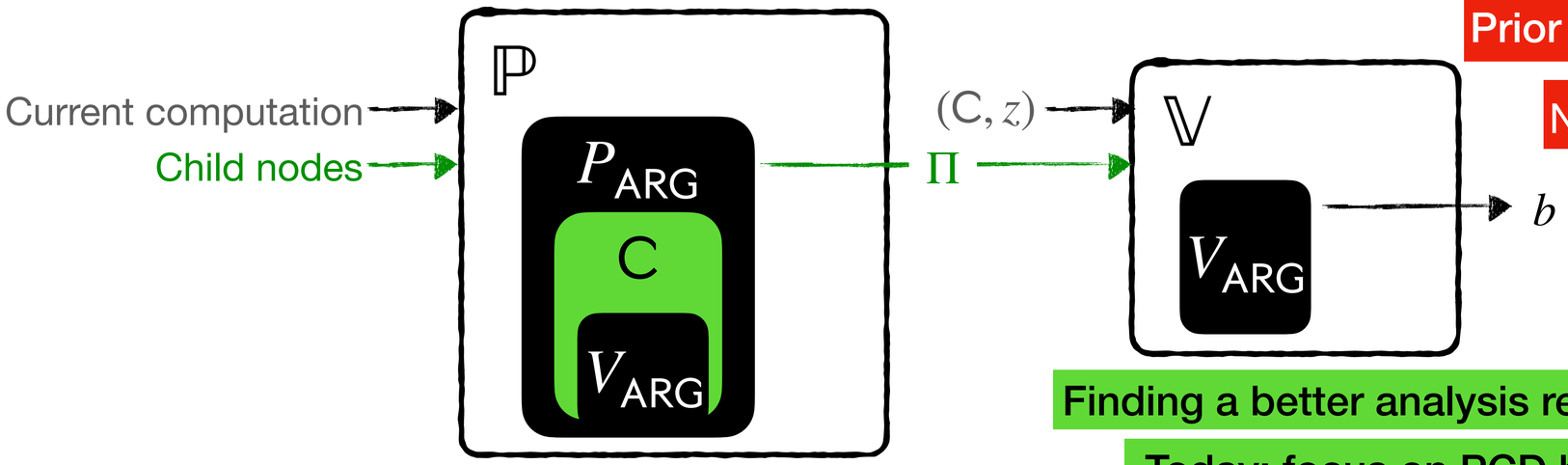
No security when D is larger than constant.

**Finding a better analysis remains a MAJOR open problem in this area.**

# Recursively compose the SNARK proofs

PCD = recursive proof composition of a SNARK: PCD invokes SNARK (for CSAT) for the recursive circuit C.




**Prior works:** $\kappa(\lambda, q, D, N) \leq \exp(D) \cdot \kappa_{\text{ARG}}(\lambda, q, N)$

No security when D is larger than constant.

Finding a better analysis remains a MAJOR open problem in this area.

Today: focus on PCD based on SNARKs with "strong" extraction.

# Our result

# Our result

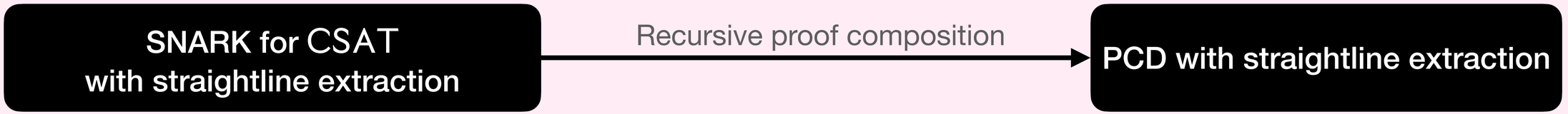
**Theorem.**

# Our result

**Theorem.** We prove a significantly improved security bound for PCD based on SNARKs with **straightline extraction**:

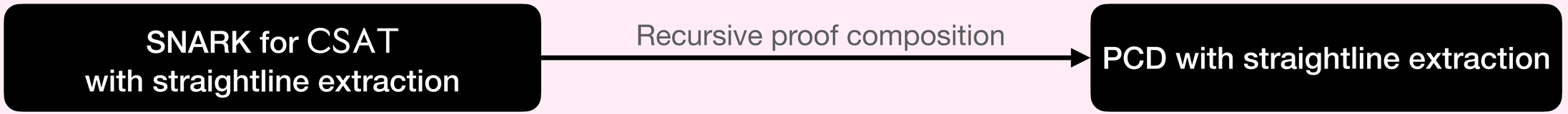
# Our result

**Theorem.** We prove a significantly improved security bound for PCD based on SNARKs with **straightline extraction**:



# Our result

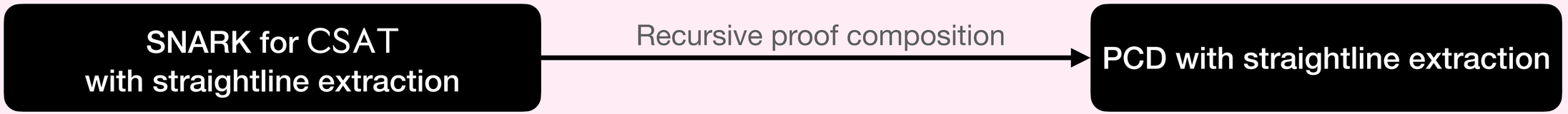
**Theorem.** We prove a significantly improved security bound for PCD based on SNARKs with **straightline extraction**:




$$\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N})$$

# Our result

**Theorem.** We prove a significantly improved security bound for PCD based on SNARKs with **straightline extraction**:

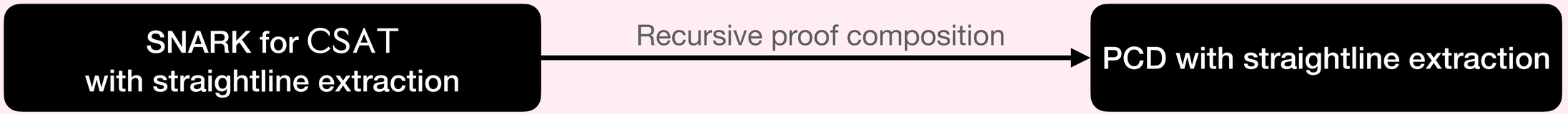



$$\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N})$$

**Significant improvement:**
**$\kappa$ doesn't grow exponentially with $\mathsf{D}$**

# Our result

**Theorem.** We prove a significantly improved security bound for PCD based on SNARKs with **straightline extraction**:



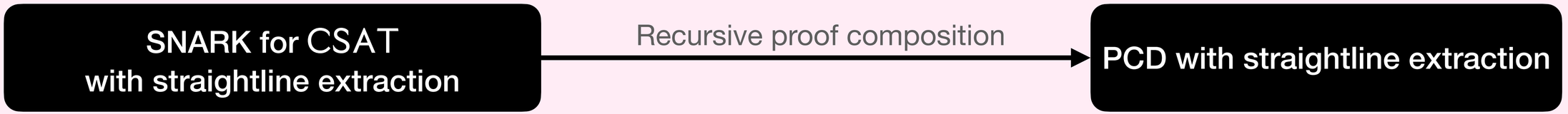

$$\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N})$$

**Significant improvement:**
**$\kappa$ doesn't grow exponentially with D**

**In practice, SNARKs have non-black-box knowledge soundness.**

# Our result

**Theorem.** We prove a significantly improved security bound for PCD based on SNARKs with **straightline extraction**:



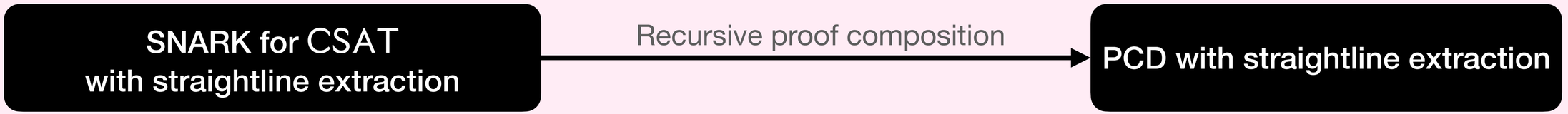

$$\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N})$$

**Significant improvement:**
**$\kappa$ doesn't grow exponentially with D**

**In practice, SNARKs have non-black-box knowledge soundness.**
**Straightline extraction only exists in idealized models.**

# Our result

**Theorem.** We prove a significantly improved security bound for PCD based on SNARKs with **straightline extraction**:




$$\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N})$$

**Significant improvement:**
**$\kappa$ doesn't grow exponentially with D**

**In practice, SNARKs have non-black-box knowledge soundness.**
**Straightline extraction only exists in idealized models.**
**How can we apply our theorem in practice then?**

# Applications

# Applications

**Application 1 [main].**

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

- [CT10]: SNARK with straightline extraction in the SROM (*signed random oracle model*).

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

- [CT10]: SNARK with straightline extraction in the SROM (*signed random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

- [CT10]: SNARK with straightline extraction in the SROM (*signed random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.
- Our bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, t + \mathsf{N})$.

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

- [CT10]: SNARK with straightline extraction in the SROM (*signed random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.
- Our bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, t + \mathsf{N})$.

**Application 3.**

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

- [CT10]: SNARK with straightline extraction in the SROM (*signed random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.
- Our bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, t + \mathsf{N})$.

**Application 3.**

- [CCGOS23]: SNARK with straightline extraction in the AROM (*arithmetized random oracle model*).

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

- [CT10]: SNARK with straightline extraction in the SROM (*signed random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.
- Our bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, t + \mathsf{N})$.

**Application 3.**

- [CCGOS23]: SNARK with straightline extraction in the AROM (*arithmetized random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.

# Applications

**Application 1 [main].**

- We propose a new idealization of hash-based PCD used in practice as a “PCD” in the ROM.
- We apply our theorem: $\kappa(\lambda, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda)$.
- First justification for current choice of parameters of hash-based PCD in practice! [Polygon, Sharp]

**Application 2.**

- [CT10]: SNARK with straightline extraction in the SROM (*signed random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.
- Our bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, t + \mathsf{N})$.

**Application 3.**

- [CCGOS23]: SNARK with straightline extraction in the AROM (*arithmetized random oracle model*).
- Their bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \mathsf{N} \cdot \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, \mathsf{N} \cdot t)$.
- Our bound: $\kappa(\lambda, \mathsf{D}, \mathsf{N}, t) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{N}, t + \mathsf{N})$.

# What is straightline extraction?

# SNARKs with straightline extraction

# SNARKs with straightline extraction

**SNARKs in an oracle model (e.g. ROM):**

# SNARKs with straightline extraction

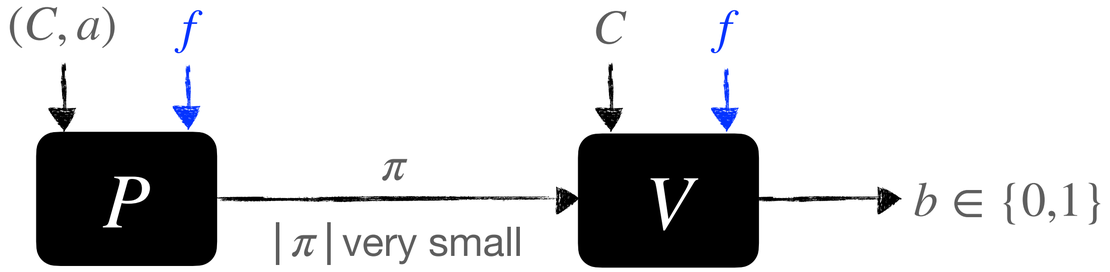
**SNARKs in an oracle model (e.g. ROM):**

# SNARKs with straightline extraction

**SNARKs in an oracle model (e.g. ROM):**

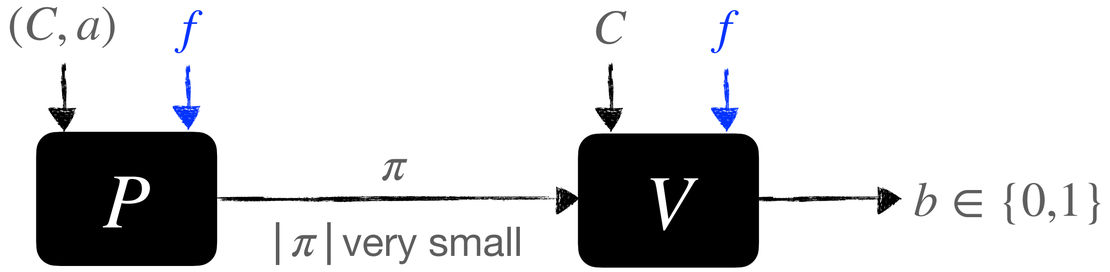
# SNARKs with straightline extraction

**SNARKs in an oracle model (e.g. ROM):**



# SNARKs with straightline extraction

**SNARKs in an oracle model (e.g. ROM):**



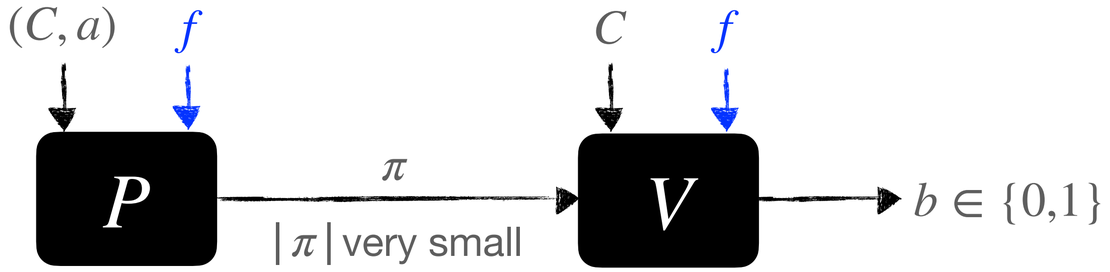

$\lambda$: security parameter

q: adversary query bound

**Straightline knowledge soundness**: ∃ deterministic extractor $E$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q})$:

# SNARKs with straightline extraction

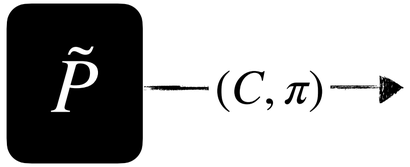
**SNARKs in an oracle model (e.g. ROM):**




$\lambda$: security parameter

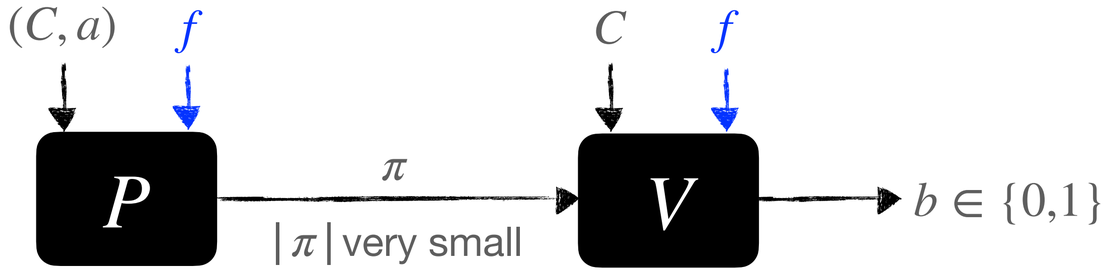
q: adversary query bound

**Straightline knowledge soundness**: ∃ deterministic extractor $E$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q})$:



# SNARKs with straightline extraction

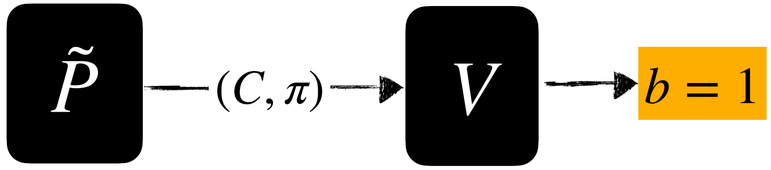
**SNARKs in an oracle model (e.g. ROM):**




$\lambda$: security parameter

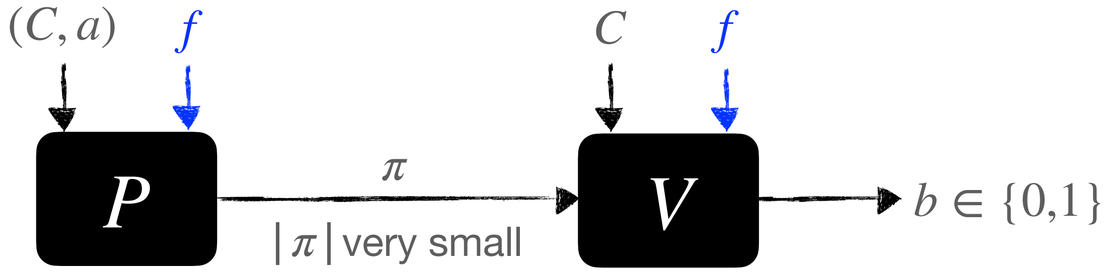
q: adversary query bound

**Straightline knowledge soundness**: ∃ deterministic extractor $E$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q})$:



# SNARKs with straightline extraction

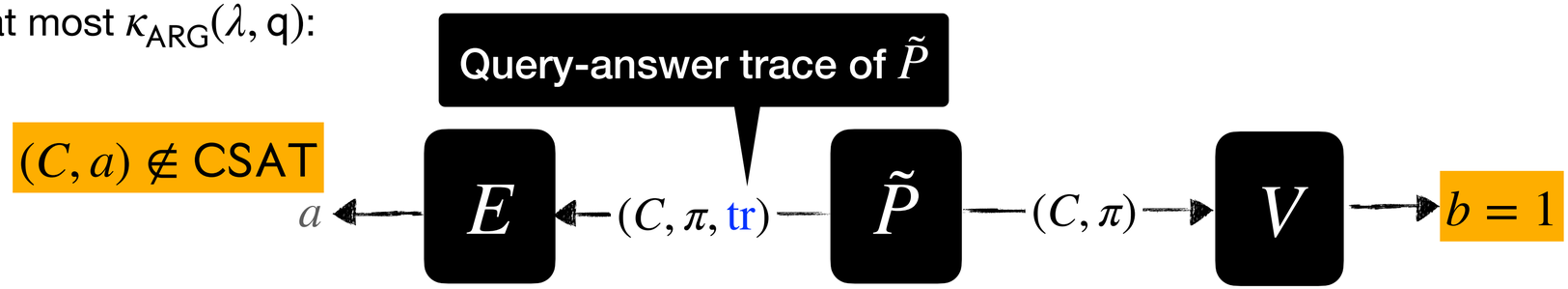
**SNARKs in an oracle model (e.g. ROM):**




$\lambda$: security parameter

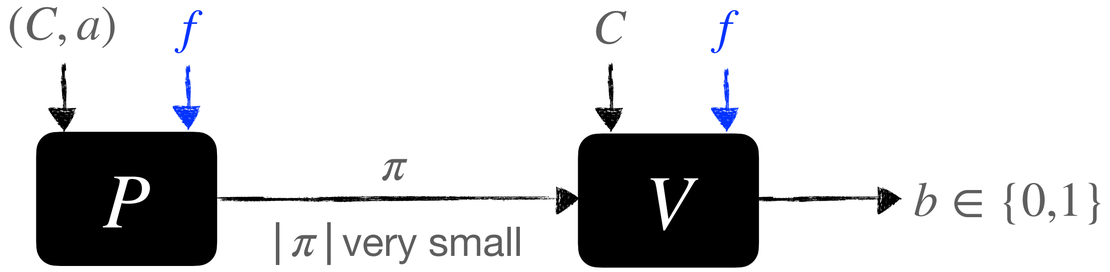
q: adversary query bound

**Straightline knowledge soundness**: ∃ deterministic extractor $E$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q})$:



# SNARKs with straightline extraction

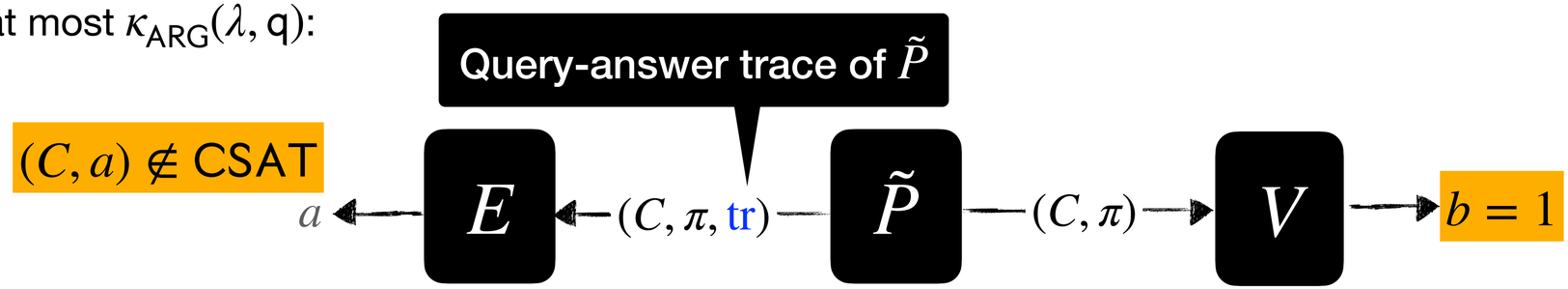
**SNARKs in an oracle model (e.g. ROM):**




$\lambda$: security parameter

q: adversary query bound

**Straightline knowledge soundness**: ∃ deterministic extractor $E$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa_{\mathsf{ARG}}(\lambda, \mathsf{q})$:

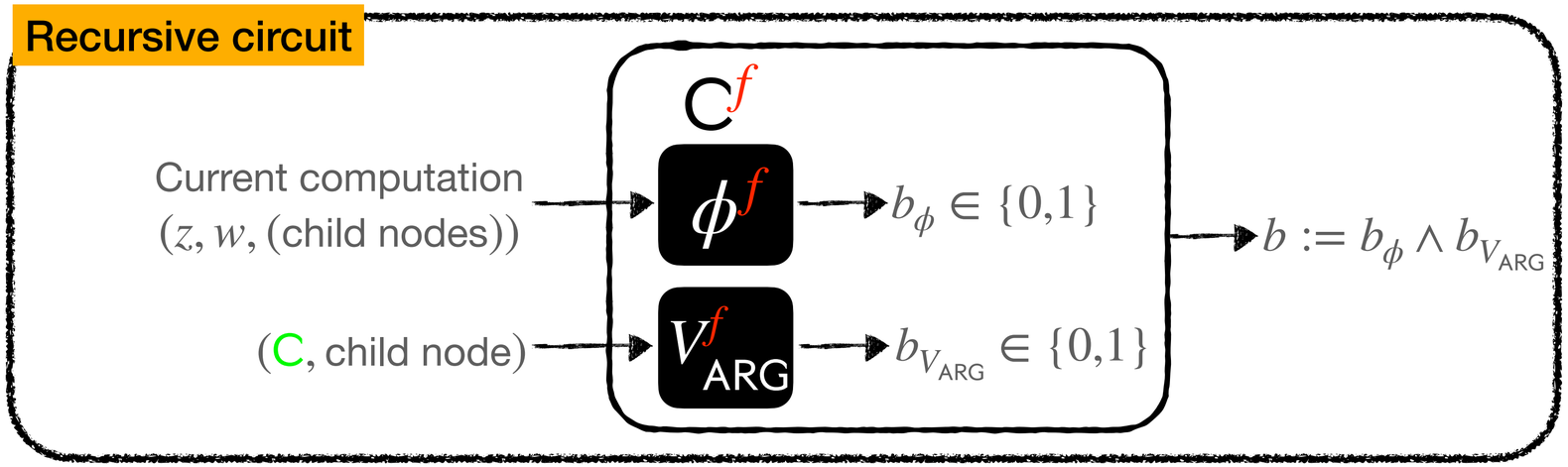



Wonderful Fact: in the ROM (and other interesting oracle models) there are SNARKs of interest with straightline extraction!

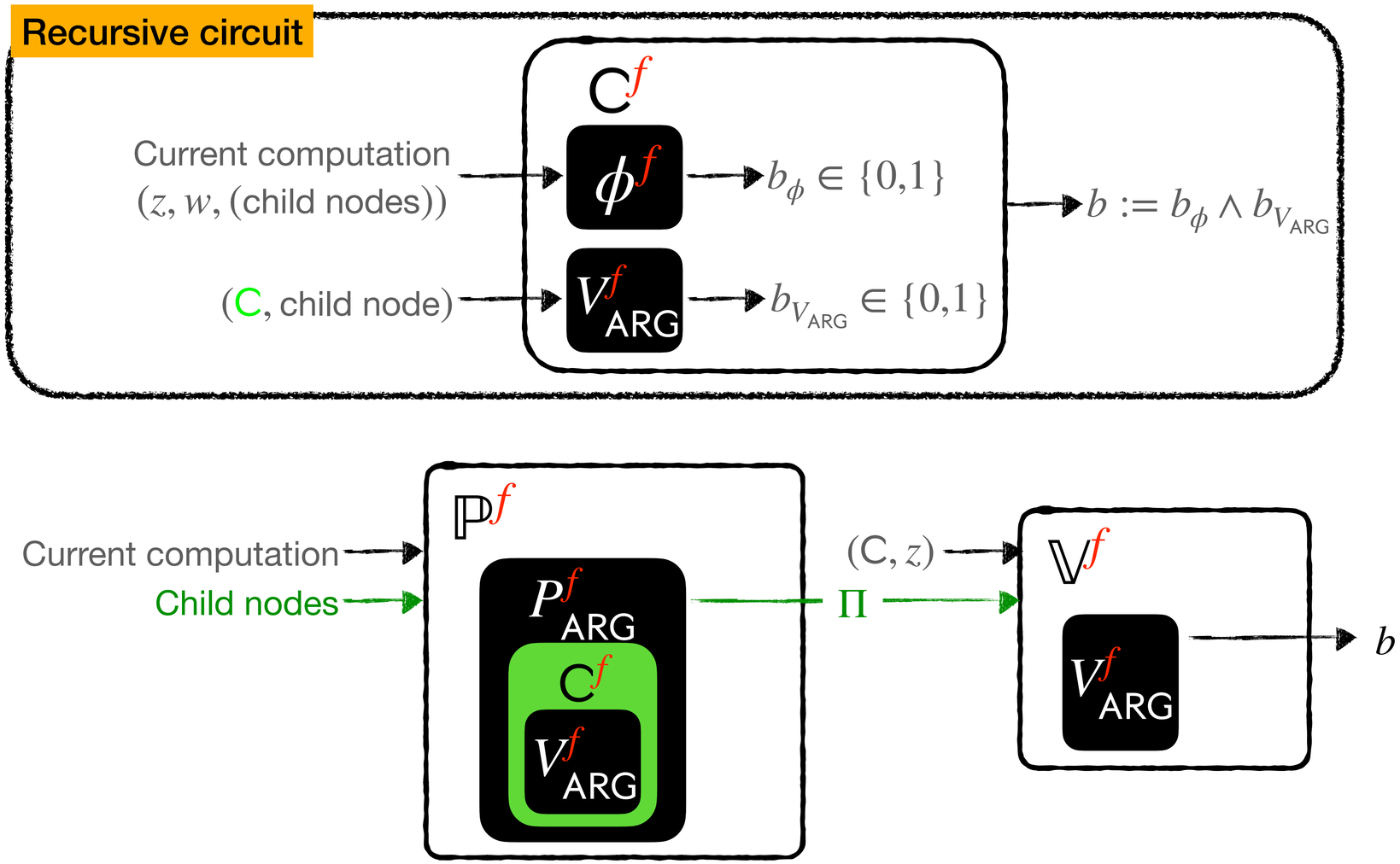
(E.g., the Micali SNARK and BCS SNARK and related constructions.)

# Can’t we use the previous recursive composition?

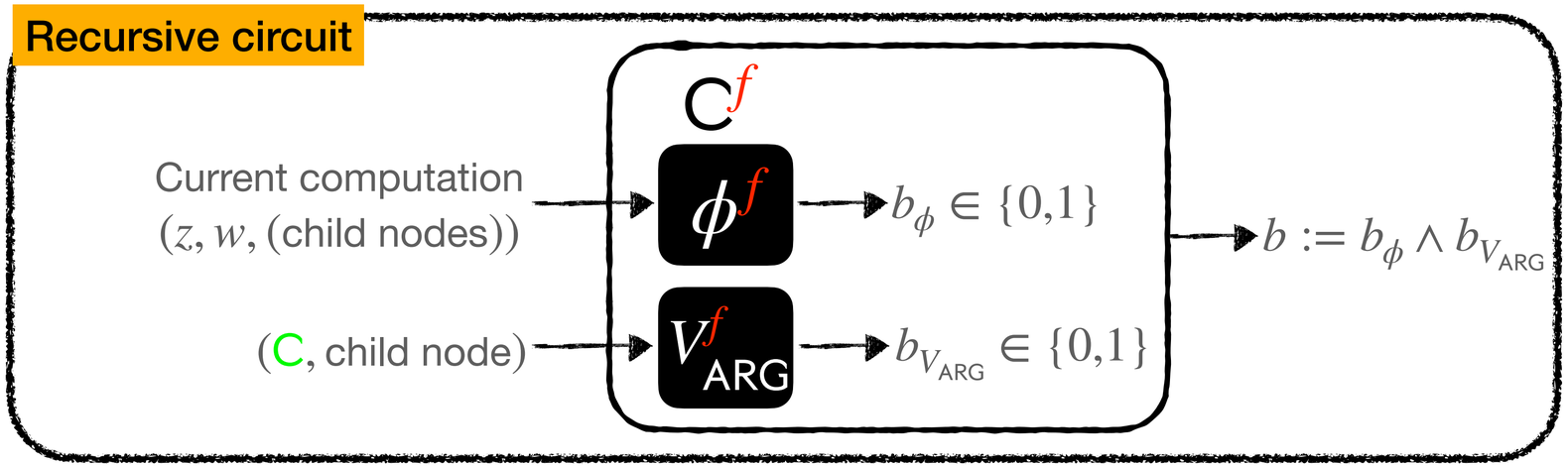
# Can’t we use the previous recursive composition?

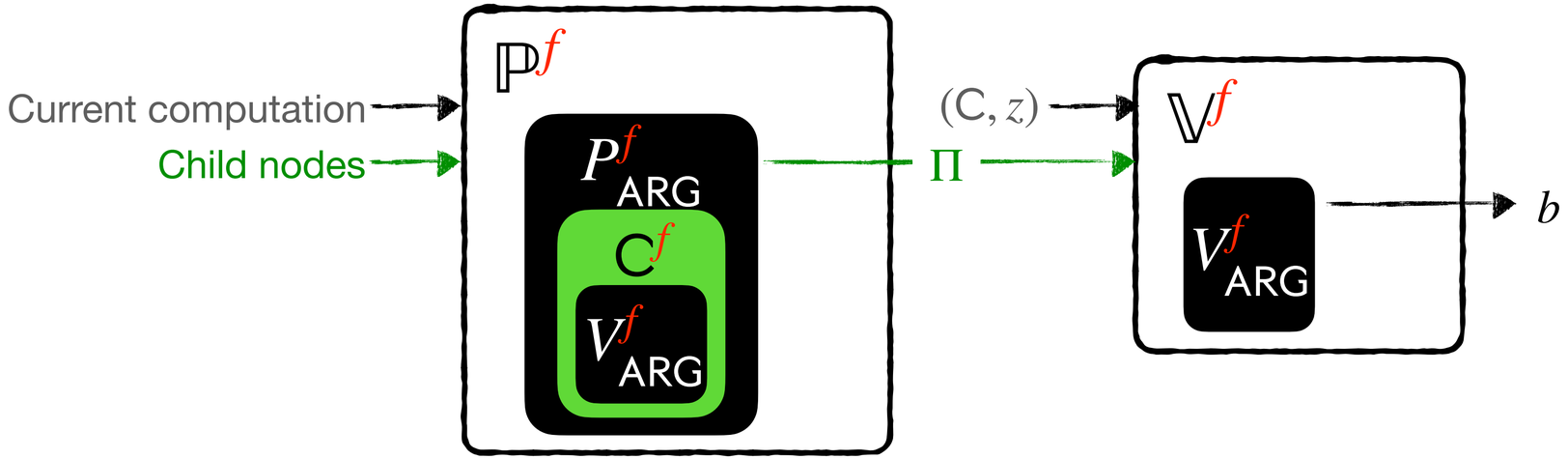


# Can’t we use the previous recursive composition?



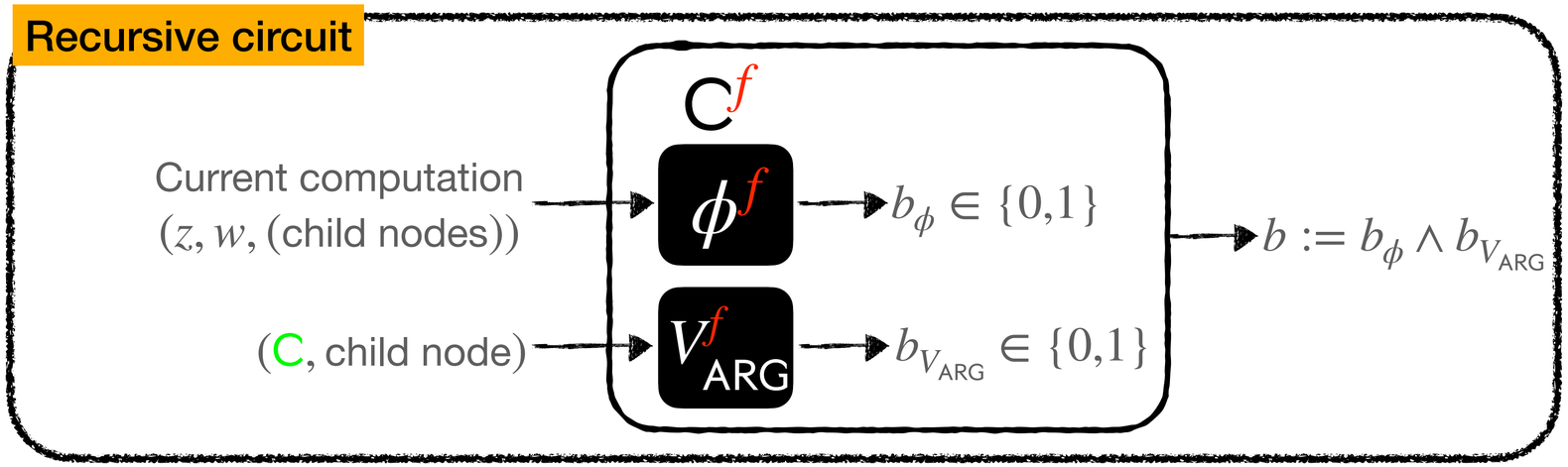
# Can’t we use the previous recursive composition?

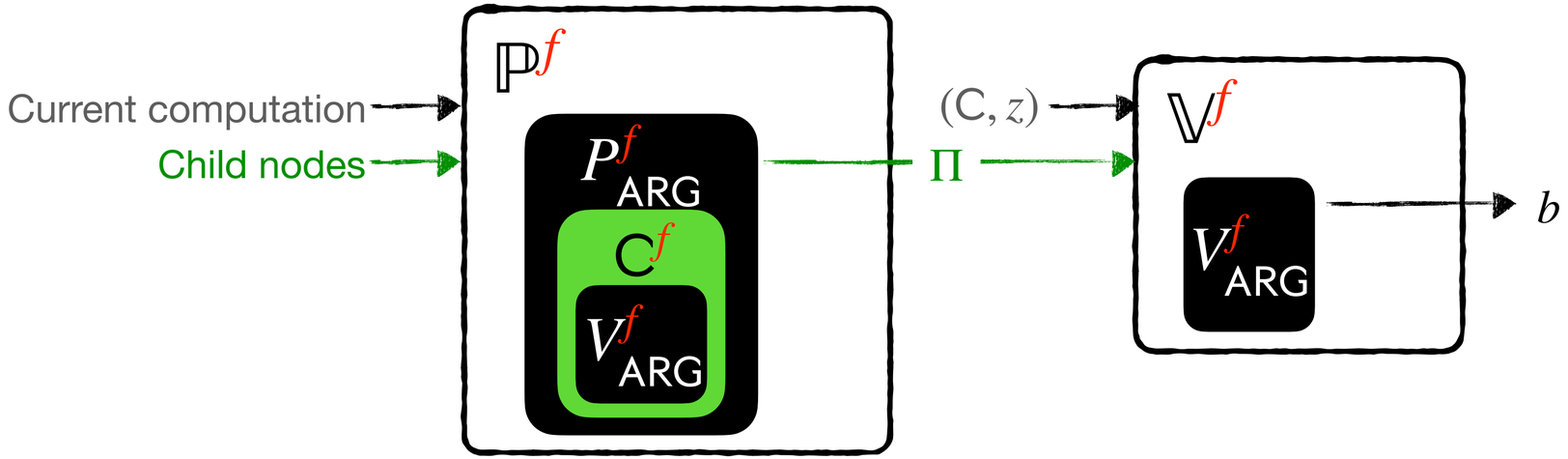



**ISSUE!** C has oracle access to $f$.

# Can’t we use the previous recursive composition?

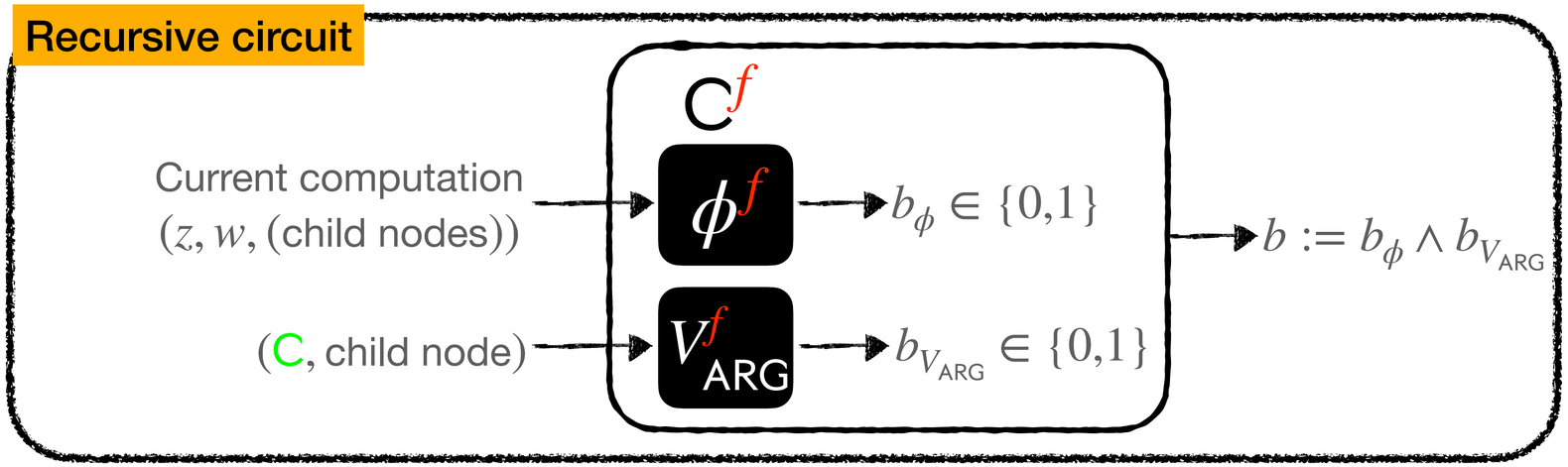


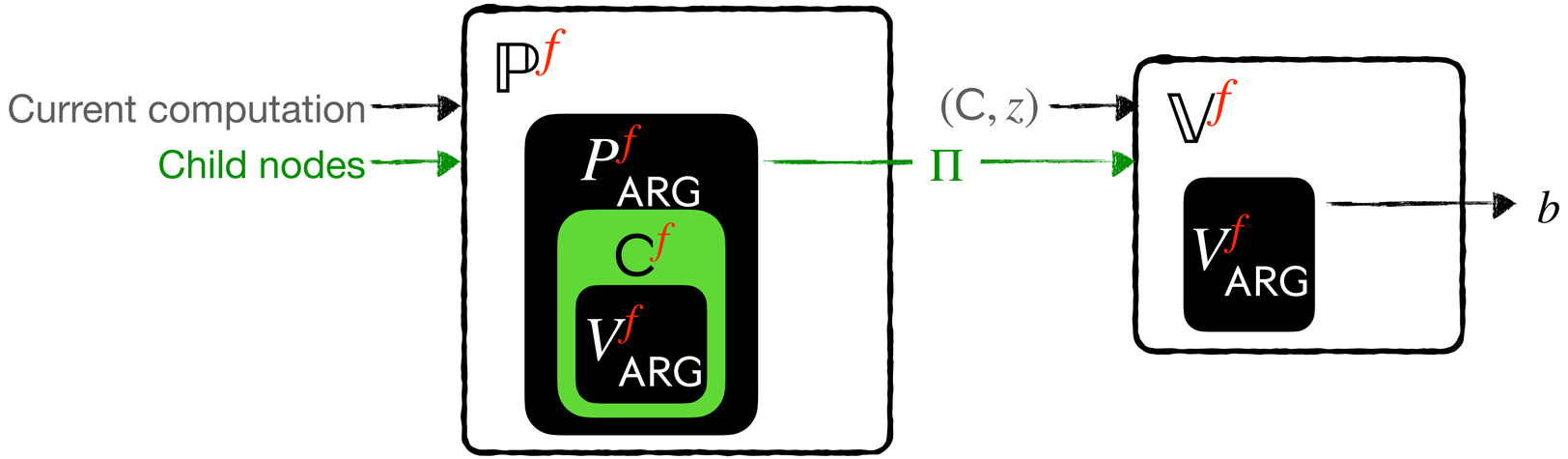




**ISSUE!** C has oracle access to $f$.

$\text{CSAT} := \{(C, a) : C(a) = 1\}$

# Can’t we use the previous recursive composition?



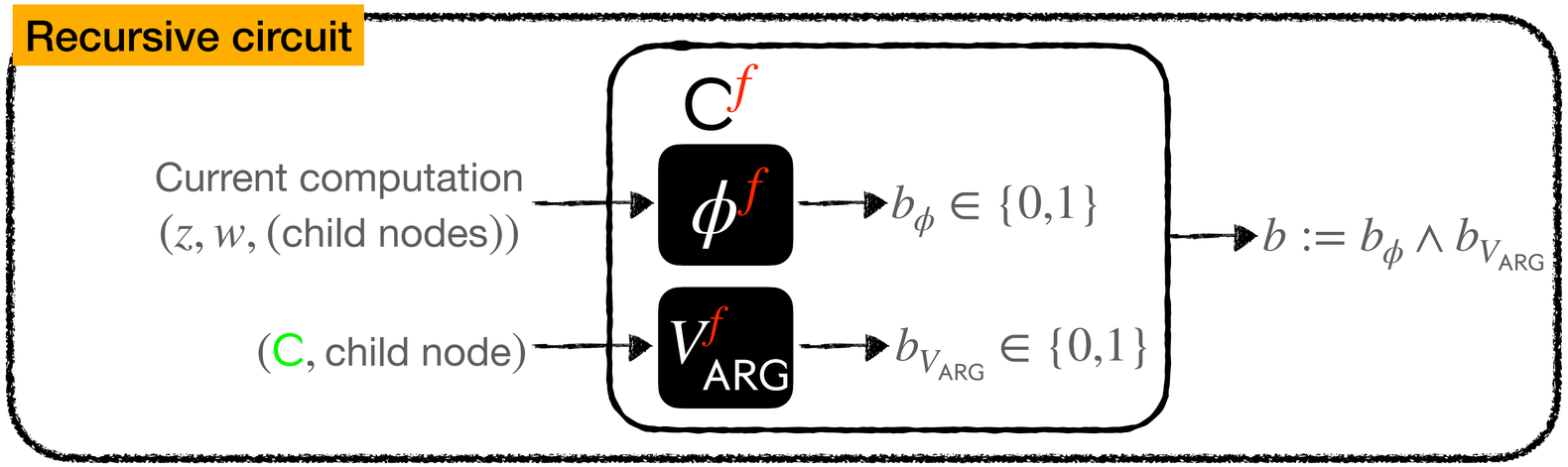



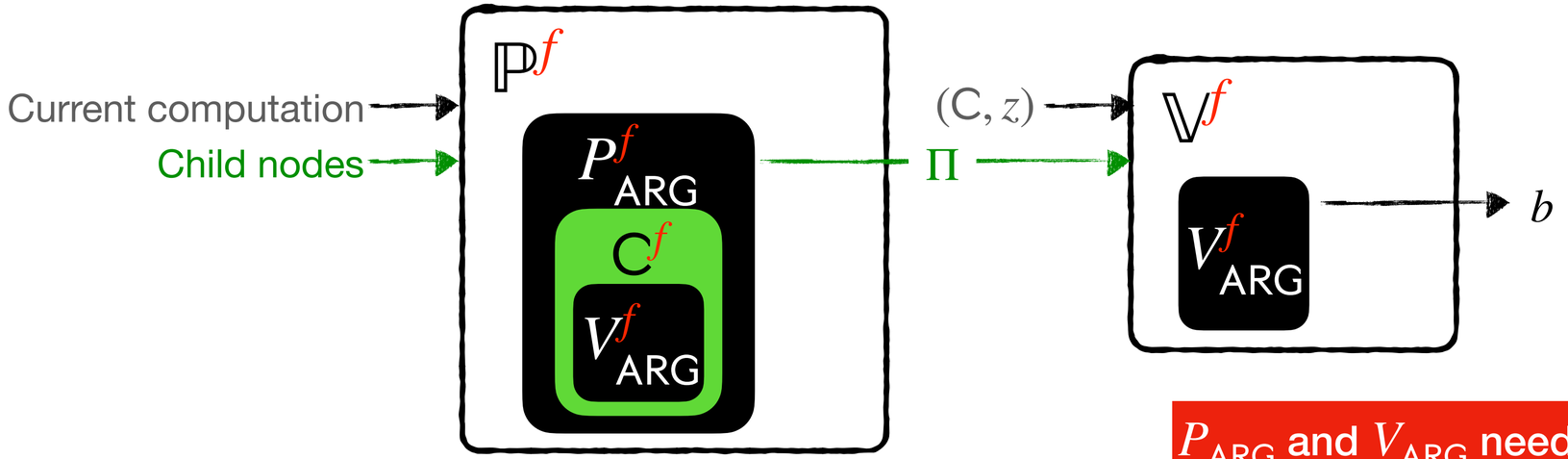

**ISSUE!** C has oracle access to $f$.

$\text{CSAT} := \{(C, a) : C(a) = 1\}$

$\text{CSAT}^f := \{(C, a) : C^f(a) = 1\}$

# Can’t we use the previous recursive composition?



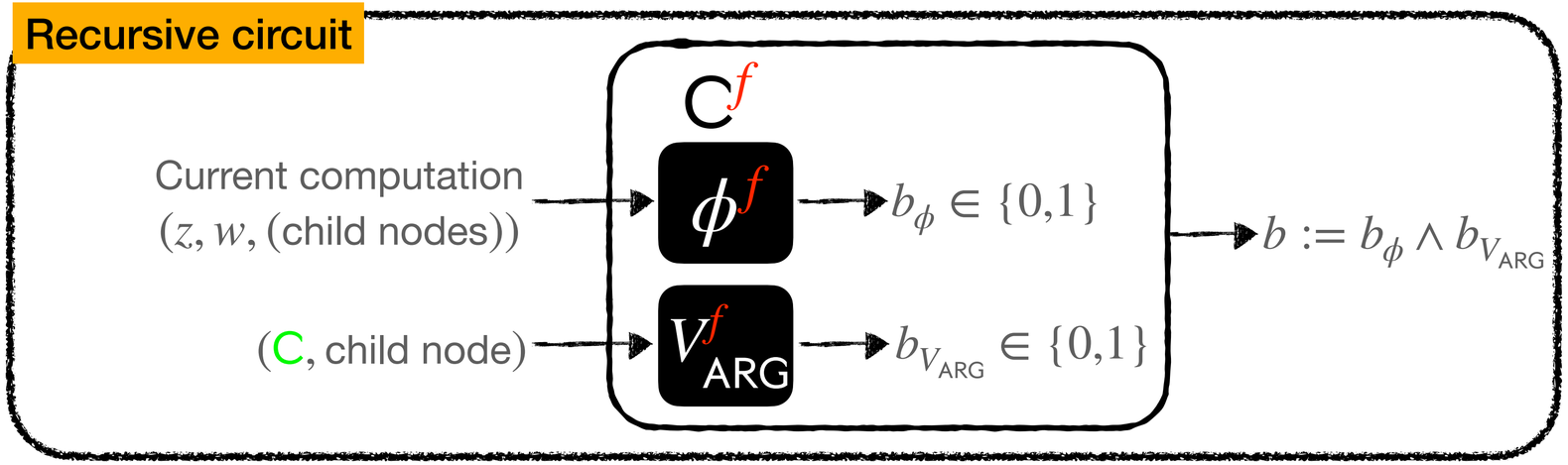

**ISSUE!** C has oracle access to $f$.

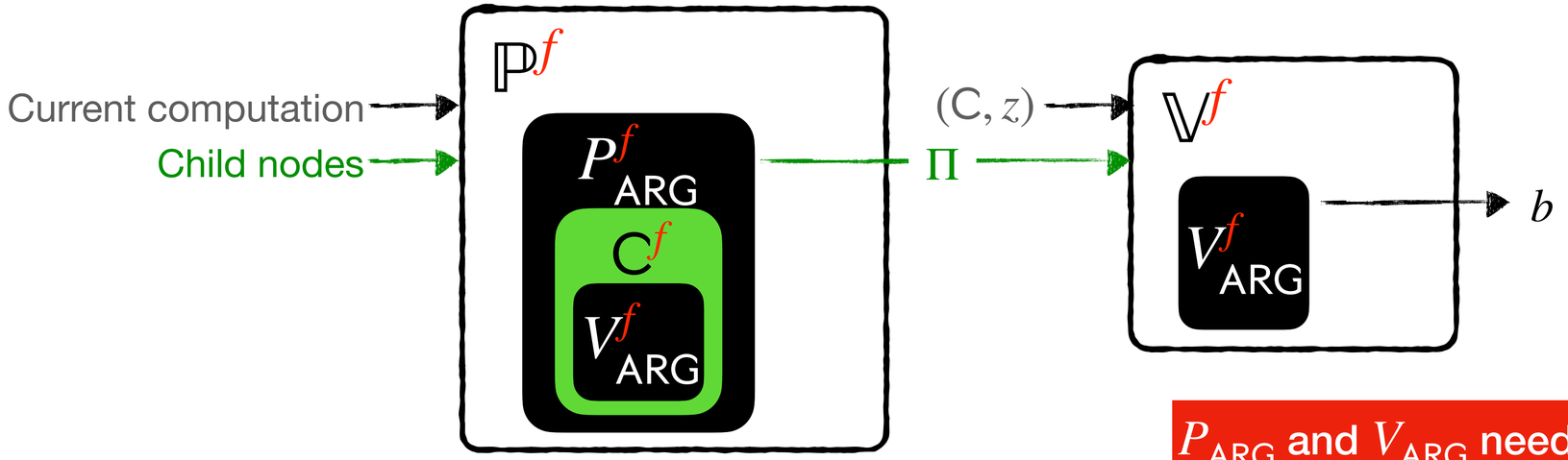
$\text{CSAT} := \{(C, a) : C(a) = 1\}$

$\text{CSAT}^f := \{(C, a) : C^f(a) = 1\}$

$P_{\text{ARG}}$ and $V_{\text{ARG}}$ need to prove computations involving oracle $f$.

# Can’t we use the previous recursive composition?




**ISSUE!** C has oracle access to $f$.

$\text{CSAT} := \{(C, a) : C(a) = 1\}$

$\text{CSAT}^f := \{(C, a) : C^f(a) = 1\}$

$P_{\text{ARG}}$ and $V_{\text{ARG}}$ need to prove computations involving oracle $f$.

**Solution: Relativized SNARKs!**

# Relativized SNARKs in an oracle model

# Relativized SNARKs in an oracle model

$\mathsf{CSAT}^f := \{(C, a) : C^f(a) = 1\}$

# Relativized SNARKs in an oracle model

$\mathsf{CSAT}^f := \{(C, a) : C^f(a) = 1\}$


PCD straightline knowledge soundness: ∃ deterministic extractor $\mathbb{E}$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa(\lambda, \mathsf{q}, \mathsf{N})$.

$\lambda$: security parameter
N: maximum transcript size
q: adversary query bound

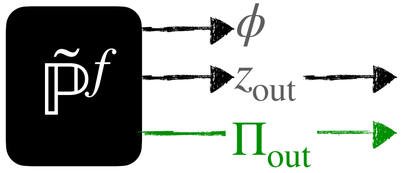
# Relativized SNARKs in an oracle model

$\mathsf{CSAT}^f := \{(C, a) : C^f(a) = 1\}$

**Relativized SNARK for $\mathsf{CSAT}^f$** → Recursive proof composition → **PCD with straightline extraction**

PCD straightline knowledge soundness: ∃ deterministic extractor $\mathbb{E}$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa(\lambda, \mathsf{q}, \mathsf{N})$.

$\lambda$: security parameter
N: maximum transcript size
q: adversary query bound



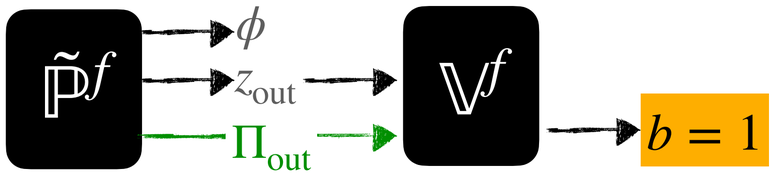
# Relativized SNARKs in an oracle model

$\mathsf{CSAT}^f := \{(C, a) : C^f(a) = 1\}$

**Relativized SNARK for $\mathsf{CSAT}^f$** → Recursive proof composition → **PCD with straightline extraction**

PCD straightline knowledge soundness: ∃ deterministic extractor $\mathbb{E}$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa(\lambda, \mathsf{q}, \mathsf{N})$.

$\lambda$: security parameter
N: maximum transcript size
q: adversary query bound



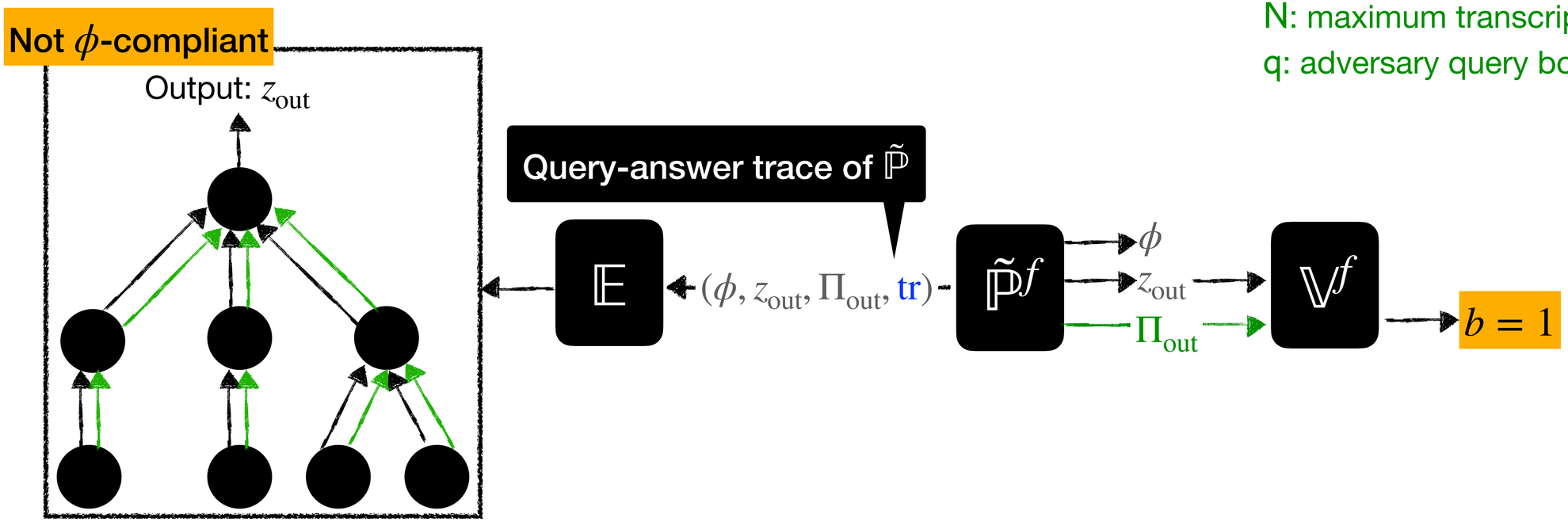
# Relativized SNARKs in an oracle model

$\mathsf{CSAT}^f := \{(C, a) : C^f(a) = 1\}$

**Relativized SNARK for $\mathsf{CSAT}^f$** → Recursive proof composition → **PCD with straightline extraction**

PCD straightline knowledge soundness: ∃ deterministic extractor $\mathbb{E}$, ∀ bounded adversary $\tilde{P}$, the following happens with probability at most $\kappa(\lambda, \mathsf{q}, \mathsf{N})$.

$\lambda$: security parameter
N: maximum transcript size
q: adversary query bound



# Application:
## Set security for hash-based PCD

# Warm-up: analyzing hash-based SNARKs

# Warm-up: analyzing hash-based SNARKs

**Three-step recipe:**

# Warm-up: analyzing hash-based SNARKs

**Three-step recipe:**

Step 1. Model the hash function as "ideal": a random function.

# Warm-up: analyzing hash-based SNARKs

**Three-step recipe:**

Step 1. Model the hash function as "ideal": a random function.

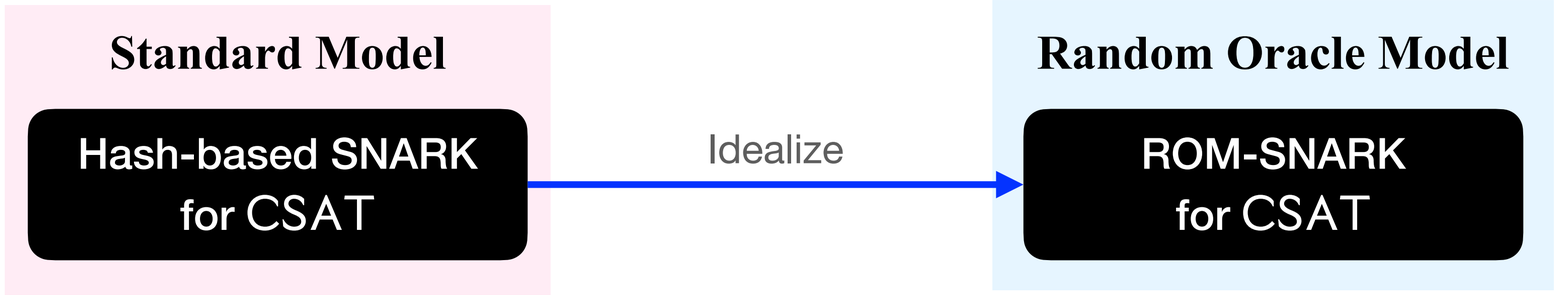
- the hash-based SNARK is idealized as **a SNARK in the random oracle model (ROM-SNARK)**.

# Warm-up: analyzing hash-based SNARKs

**Three-step recipe:**

Step 1. Model the hash function as "ideal": a random function.

- the hash-based SNARK is idealized as **a SNARK in the random oracle model (ROM-SNARK)**.



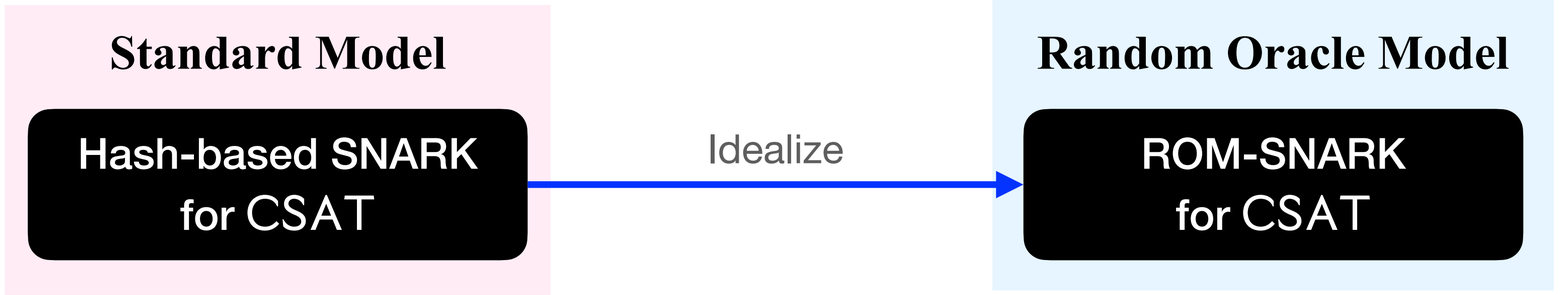
# Warm-up: analyzing hash-based SNARKs

**Three-step recipe:**

Step 1. Model the hash function as "ideal": a random function.

- the hash-based SNARK is idealized as **a SNARK in the random oracle model (ROM-SNARK)**.

Step 2. Establish **concrete** security bounds for the ROM-SNARK.



# Warm-up: analyzing hash-based SNARKs

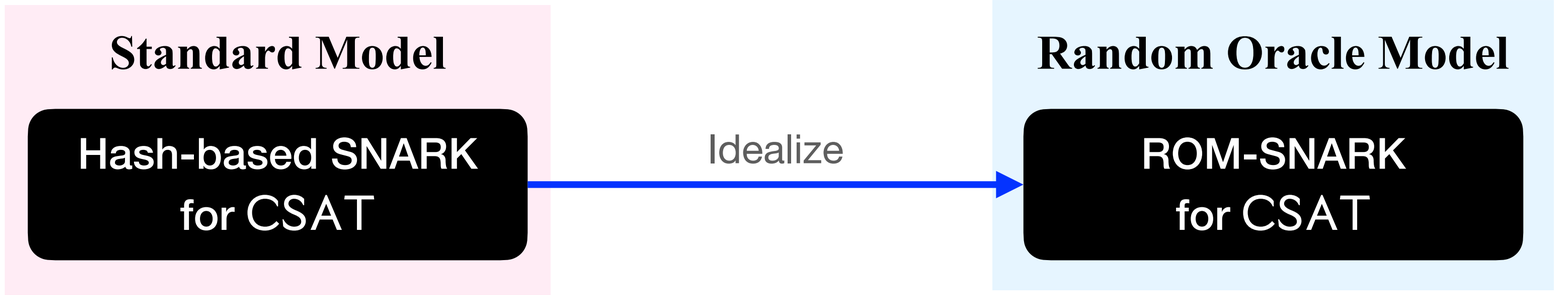
**Three-step recipe:**

Step 1. Model the hash function as "ideal": a random function.

- the hash-based SNARK is idealized as **a SNARK in the random oracle model (ROM-SNARK)**.

Step 2. Establish **concrete** security bounds for the ROM-SNARK.

Step 3. Set security parameters of the hash-based SNARK accordingly.



# Warm-up: analyzing hash-based SNARKs

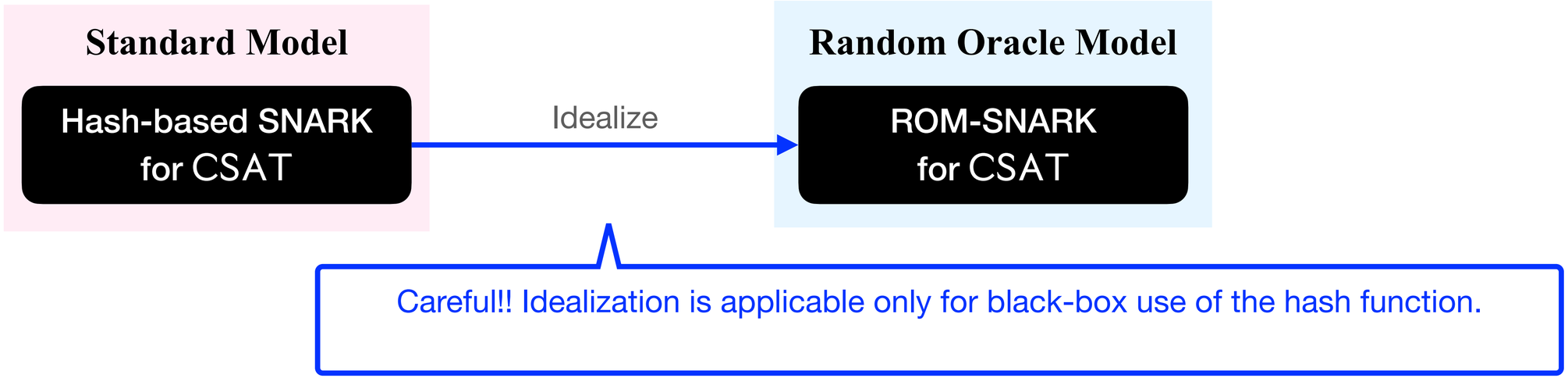
**Three-step recipe:**

Step 1. Model the hash function as "ideal": a random function.

- the hash-based SNARK is idealized as **a SNARK in the random oracle model (ROM-SNARK)**.

Step 2. Establish **concrete** security bounds for the ROM-SNARK.

Step 3. Set security parameters of the hash-based SNARK accordingly.



# Warm-up: analyzing hash-based SNARKs

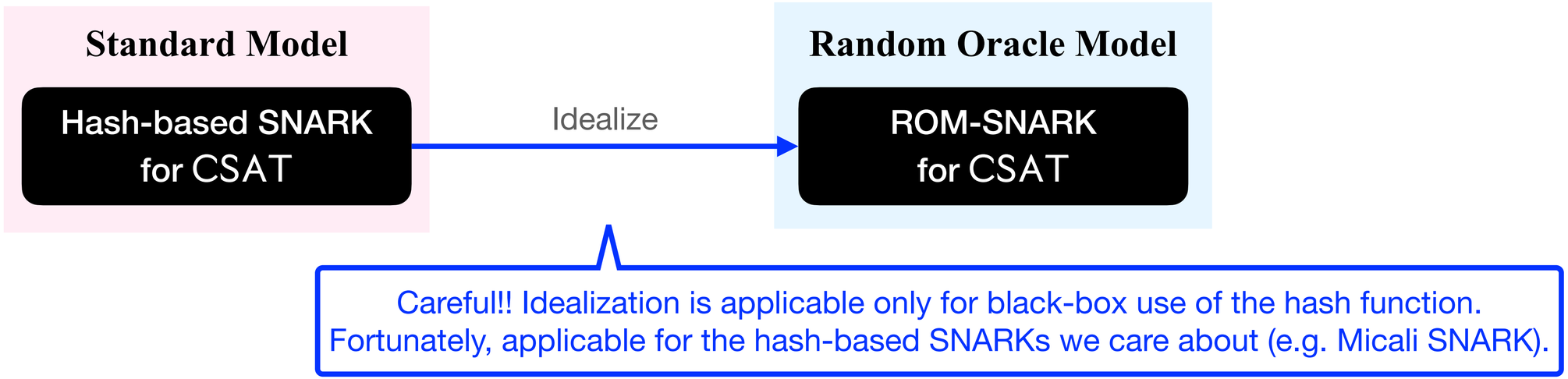
**Three-step recipe:**

Step 1. Model the hash function as "ideal": a random function.

- the hash-based SNARK is idealized as **a SNARK in the random oracle model (ROM-SNARK)**.

Step 2. Establish **concrete** security bounds for the ROM-SNARK.

Step 3. Set security parameters of the hash-based SNARK accordingly.



# First attempt for idealization of hash-based PCD

# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.

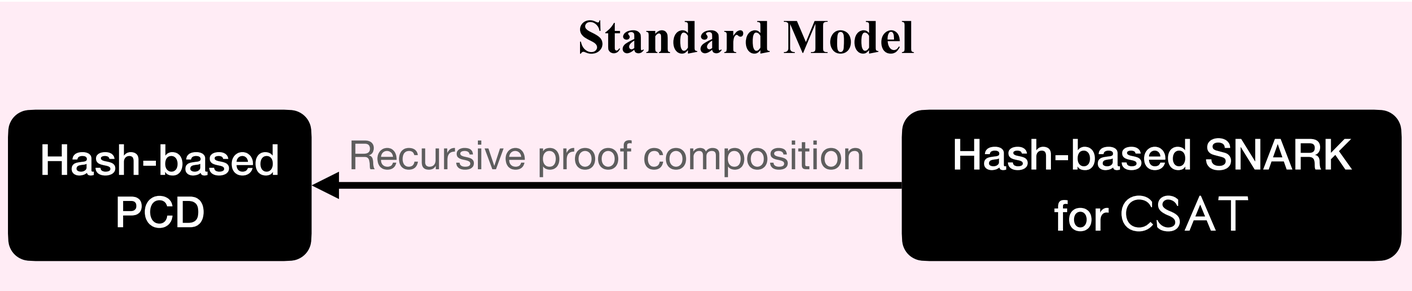
# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.

**Standard Model**

# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.

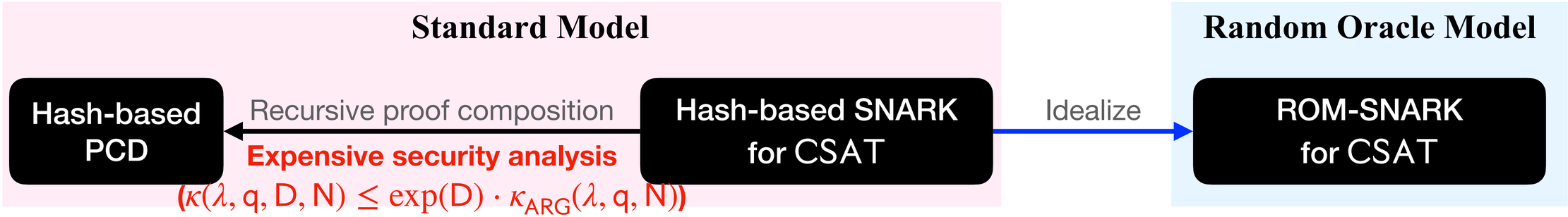


# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.

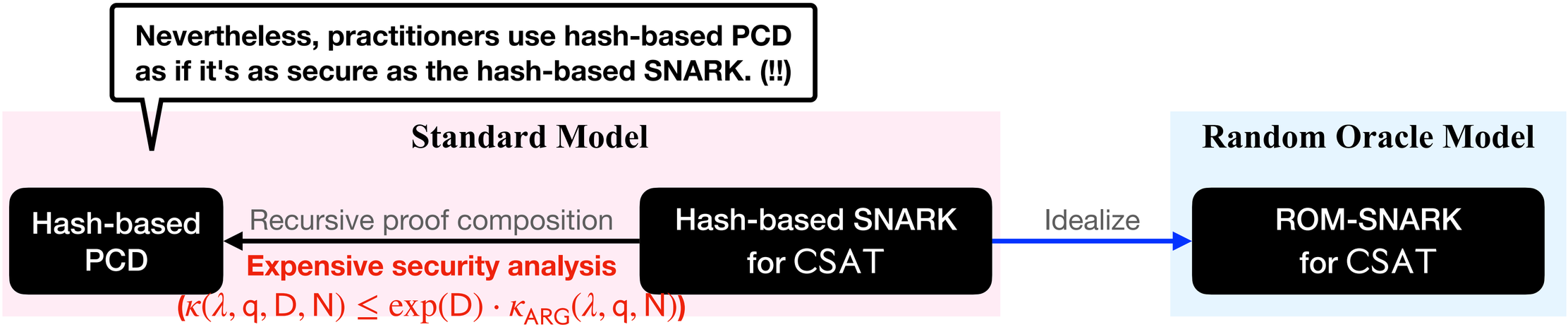
# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.



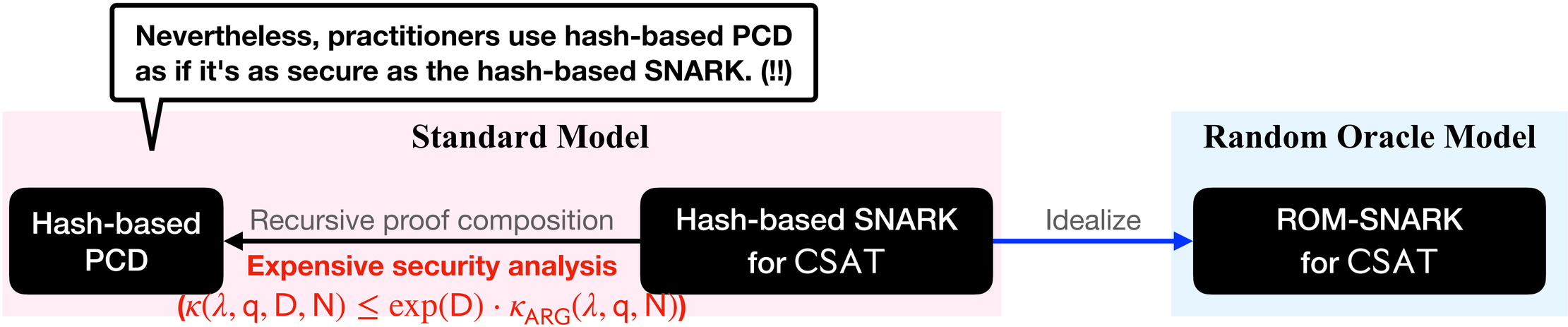
# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.



# First attempt for idealization of hash-based PCD

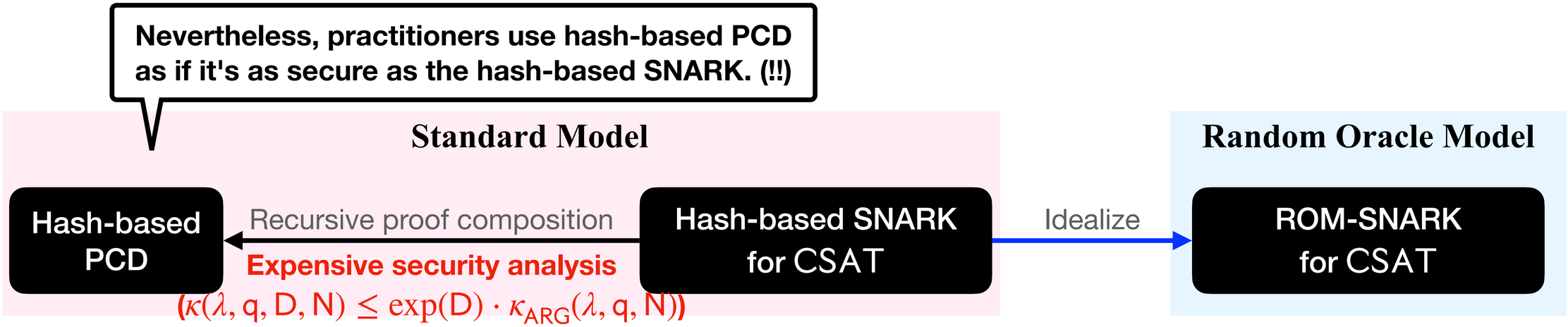
PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.




Can our new analysis justify the above practice?

# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.

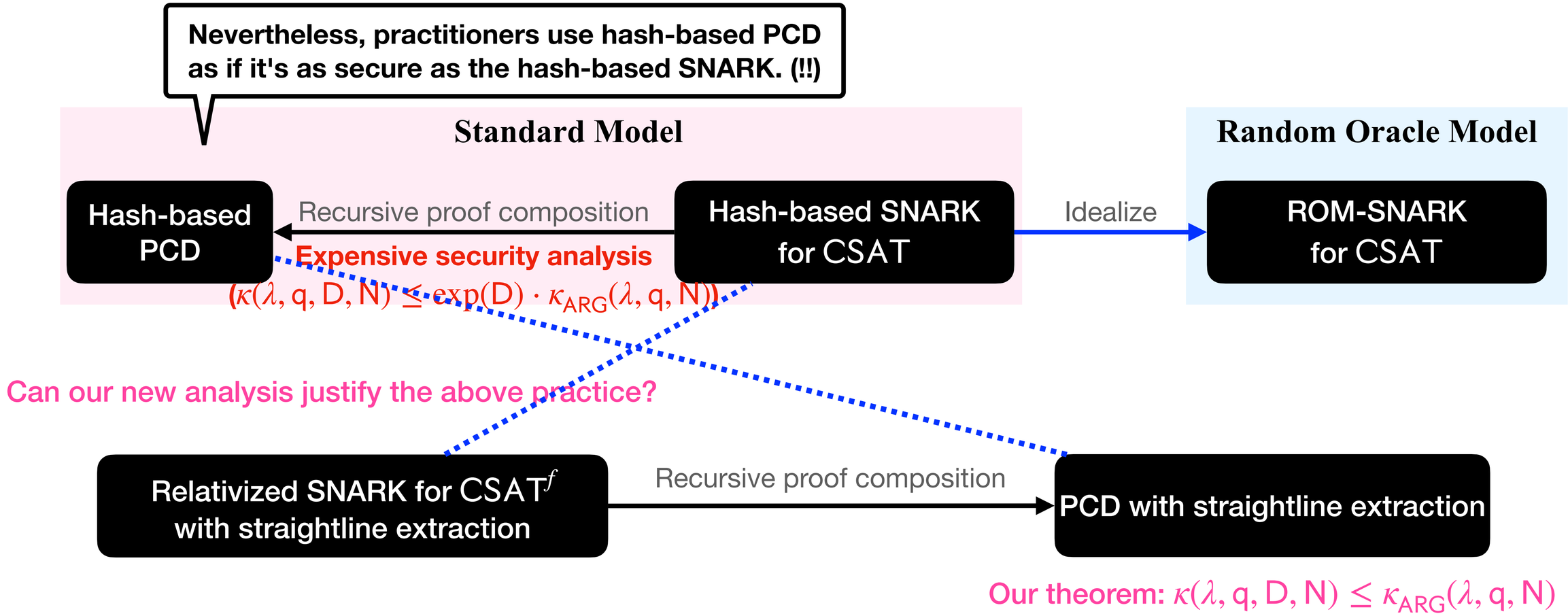



Can our new analysis justify the above practice?


Our theorem: $\kappa(\lambda, q, D, N) \leq \kappa_{ARG}(\lambda, q, N)$

# First attempt for idealization of hash-based PCD

PCDs are deployed based on various approaches. A popular approach is **hash-based PCD**.



# Second attempt for idealization of hash-based PCD

# Second attempt for idealization of hash-based PCD

What we hope to do

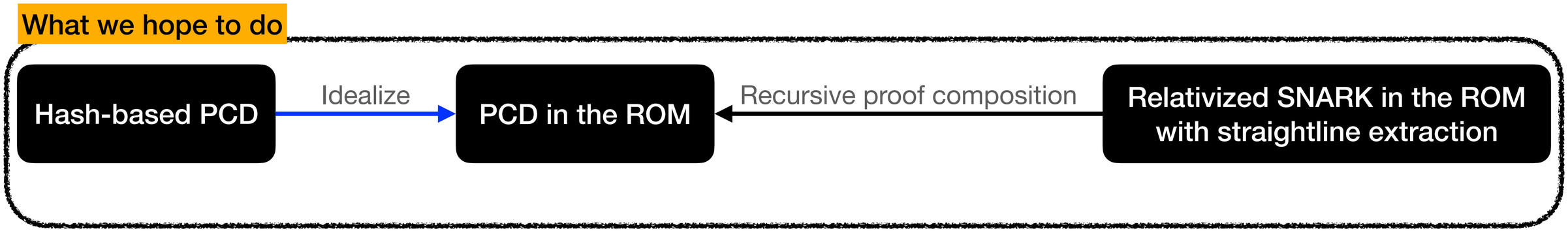
# Second attempt for idealization of hash-based PCD

What we hope to do

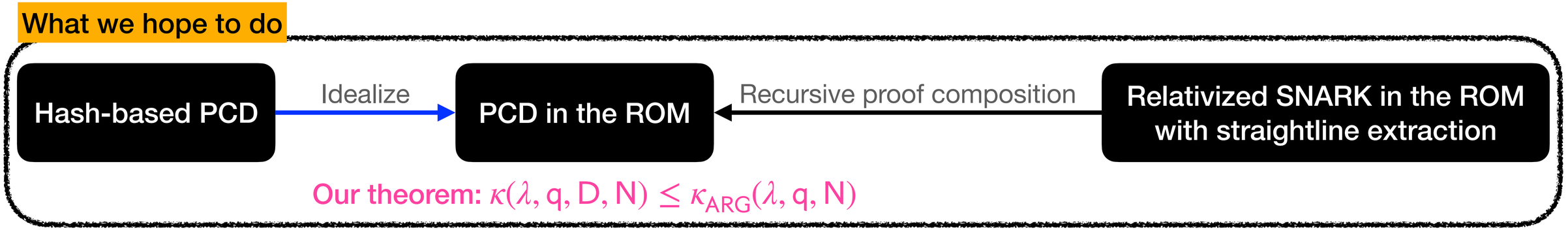
Hash-based PCD

# Second attempt for idealization of hash-based PCD

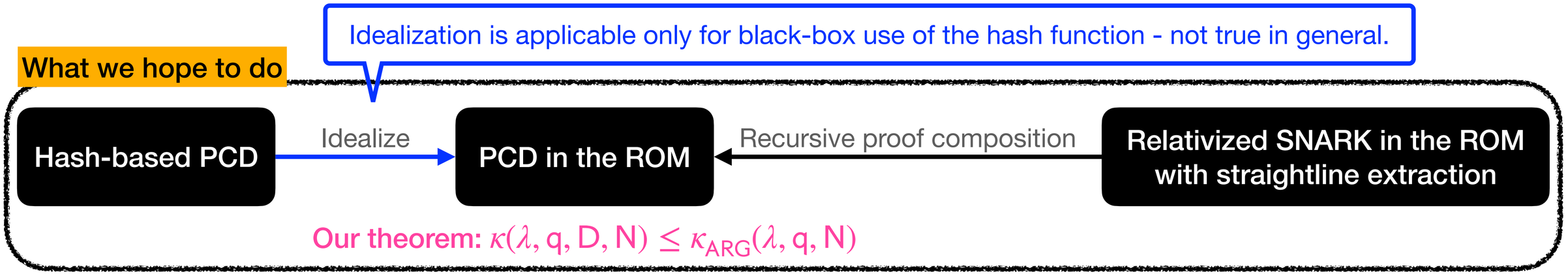
# Second attempt for idealization of hash-based PCD



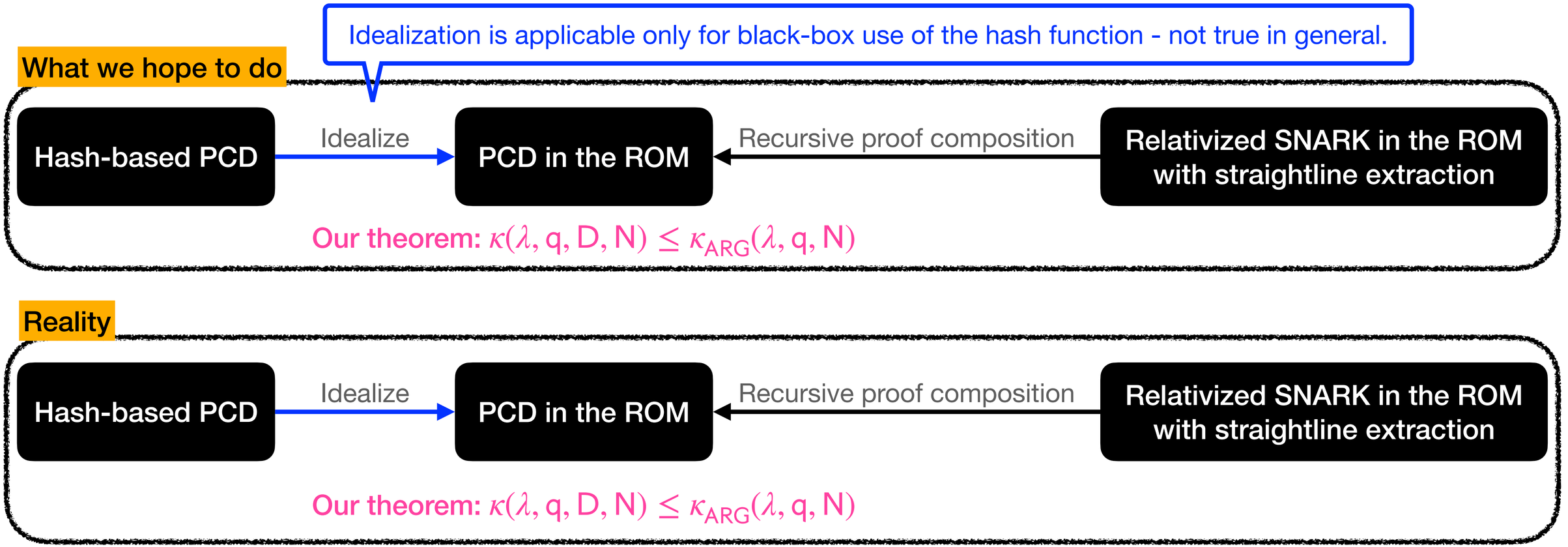
# Second attempt for idealization of hash-based PCD



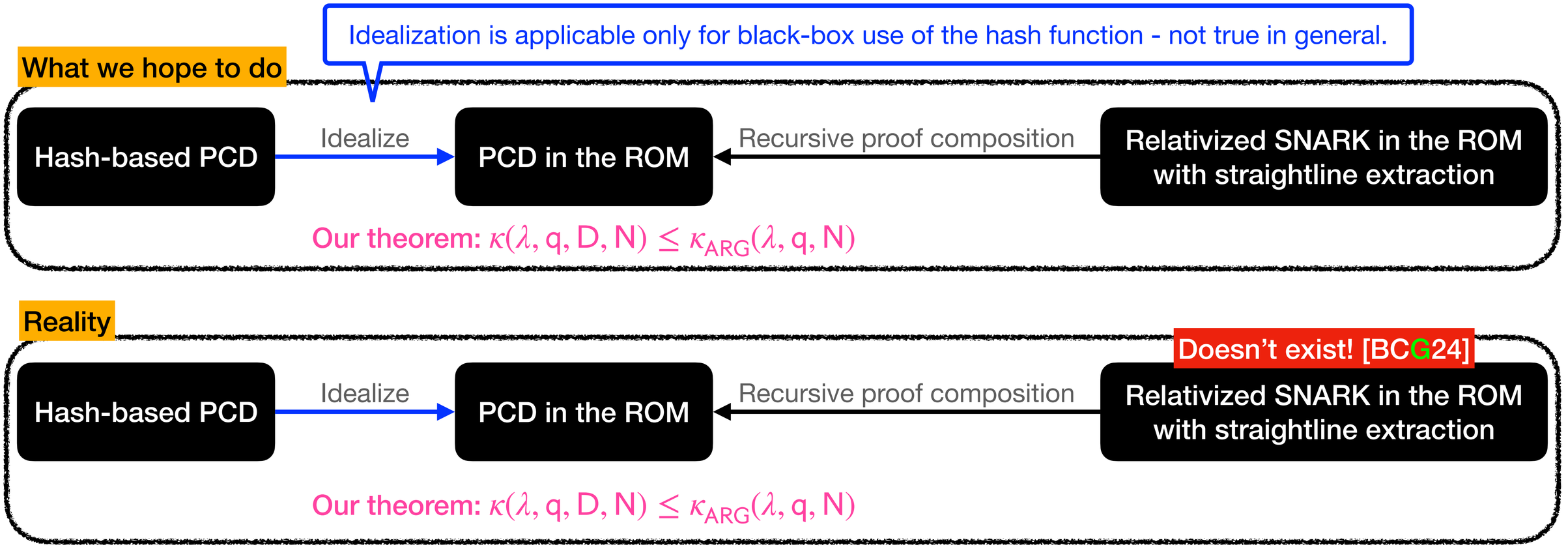
# Second attempt for idealization of hash-based PCD



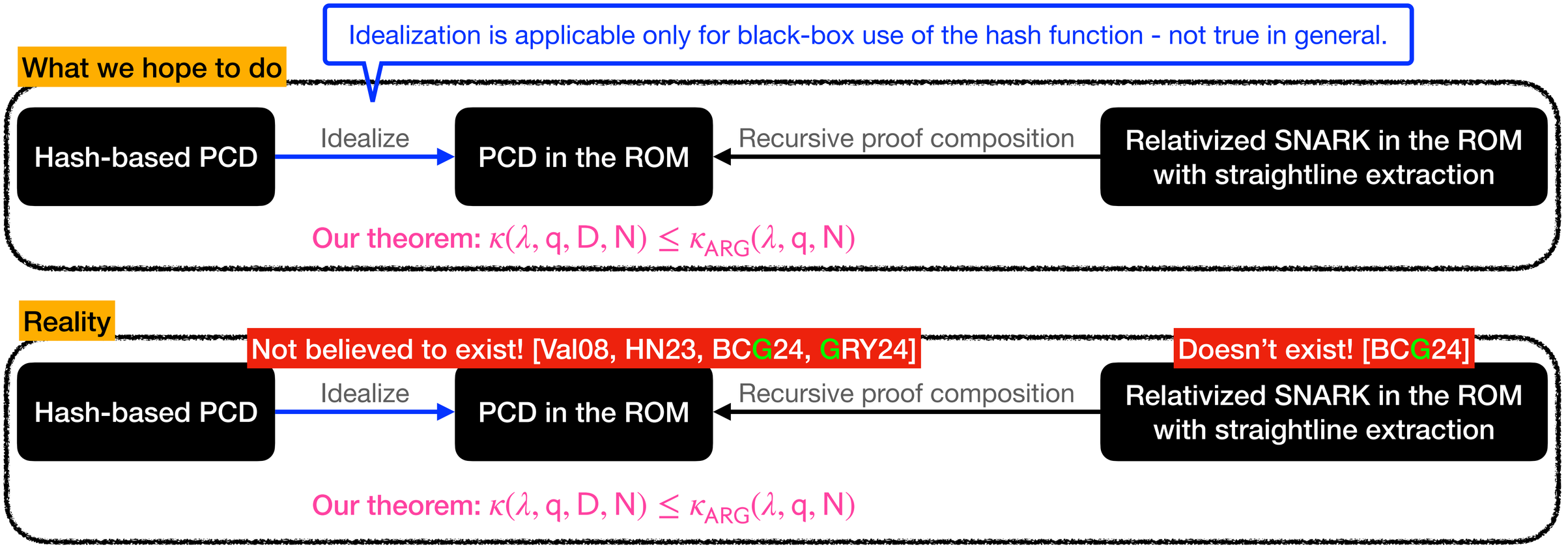
# Second attempt for idealization of hash-based PCD



# Second attempt for idealization of hash-based PCD



# Second attempt for idealization of hash-based PCD



# Second attempt for idealization of hash-based PCD

Idealization is applicable only for black-box use of the hash function - not true in general.

**What we hope to do**

Hash-based PCD —Idealize→ PCD in the ROM ←Recursive proof composition— Relativized SNARK in the ROM with straightline extraction

Our theorem: $\kappa(\lambda, q, D, N) \leq \kappa_{ARG}(\lambda, q, N)$

**Reality**

Not believed to exist! [Val08, HN23, BCG24, GRY24]

Doesn't exist! [BCG24]

Hash-based PCD —Idealize→ PCD in the ROM ←Recursive proof composition— Relativized SNARK in the ROM with straightline extraction

Can't apply Our theorem: $\kappa(\lambda, q, D, N) \leq \kappa_{ARG}(\lambda, q, N)$ ☹️

# Our idealization for hash-based PCD

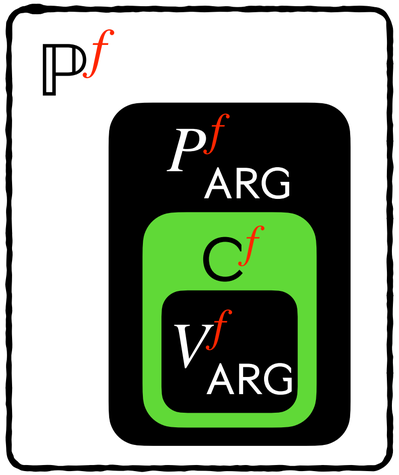
# Our idealization for hash-based PCD

Issue: Hash-based PCD uses hash function in a non-black-box way.

# Our idealization for hash-based PCD

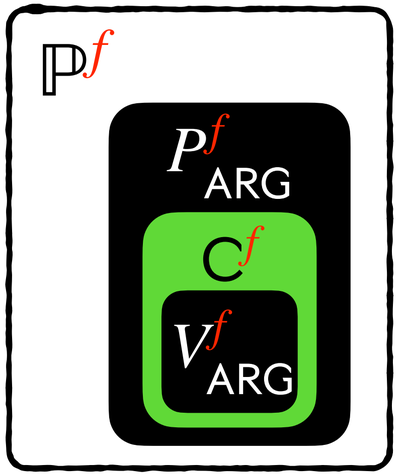
Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn’t “destroy” the hash function.

# Our idealization for hash-based PCD



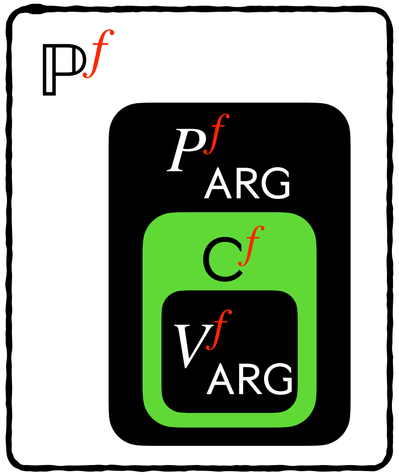

Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\text{ARG}}$ make oracle queries.

# Our idealization for hash-based PCD




Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\text{ARG}}$ make oracle queries.
Solution: Forward all the queries of $\mathsf{C}$ by asking $P_{\text{ARG}}$ to attach $\mathsf{C}$'s "query-answer trace" in the proof.

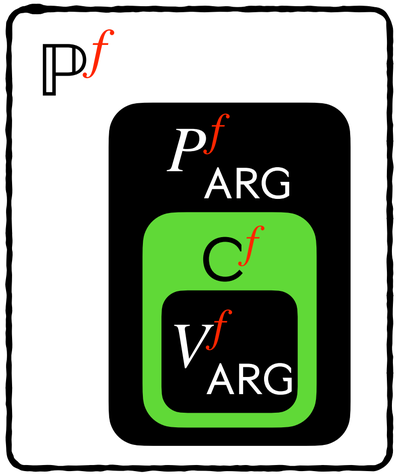
# Our idealization for hash-based PCD




Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\text{ARG}}$ make oracle queries.
Solution: Forward all the queries of $\mathsf{C}$ by asking $P_{\text{ARG}}$ to attach $\mathsf{C}$'s "query-answer trace" in the proof.

Forwarding the queries makes the proof non-succinct

# Our idealization for hash-based PCD

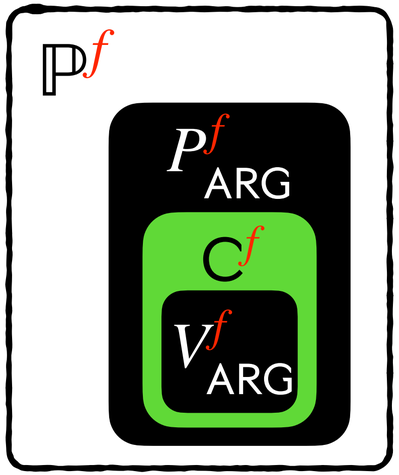



Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\text{ARG}}$ make oracle queries.
Solution: Forward all the queries of $\mathsf{C}$ by asking $P_{\text{ARG}}$ to attach $\mathsf{C}$'s "query-answer trace" in the proof.

Forwarding the queries makes the proof non-succinct

**Hash-based PCD**

# Our idealization for hash-based PCD

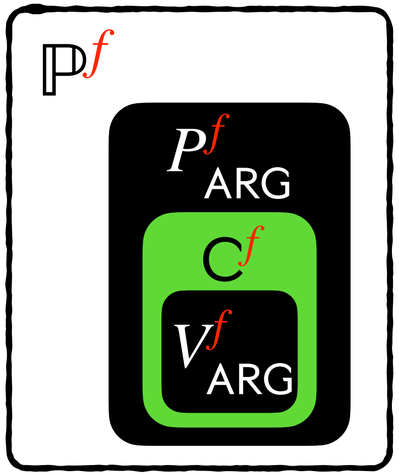



Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\mathsf{ARG}}$ make oracle queries.
Solution: Forward all the queries of $\mathsf{C}$ by asking $P_{\mathsf{ARG}}$ to attach $\mathsf{C}$'s "query-answer trace" in the proof.

Forwarding the queries makes the proof non-succinct

Hash-based PCD → Idealize

# Our idealization for hash-based PCD

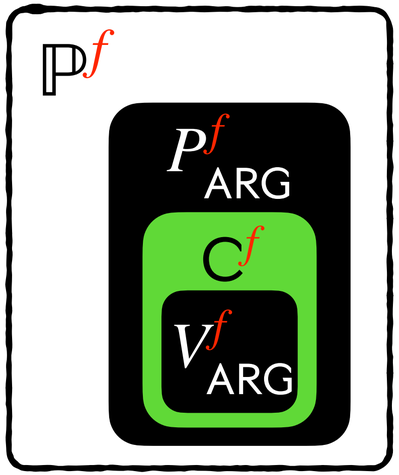



Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\mathsf{ARG}}$ make oracle queries.
Solution: Forward all the queries of $\mathsf{C}$ by asking $P_{\mathsf{ARG}}$ to attach $\mathsf{C}$'s "query-answer trace" in the proof.

Forwarding the queries makes the proof non-succinct

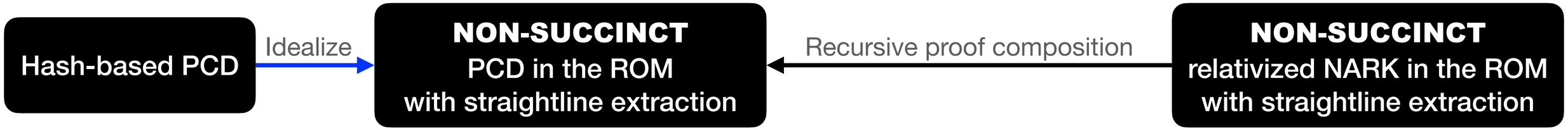
Hash-based PCD → Idealize → **NON-SUCCINCT** PCD in the ROM with straightline extraction

# Our idealization for hash-based PCD

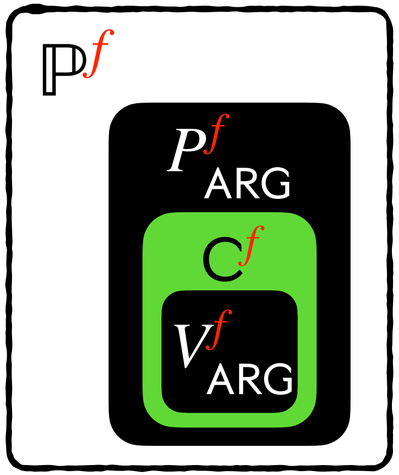



Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\mathsf{ARG}}$ make oracle queries.
Solution: Forward all the queries of $\mathsf{C}$ by asking $P_{\mathsf{ARG}}$ to attach $\mathsf{C}$'s "query-answer trace" in the proof.

Forwarding the queries makes the proof non-succinct



# Our idealization for hash-based PCD




Issue: Hash-based PCD uses hash function in a non-black-box way.
Observation 1: PCD looks at hash function to check the correctness, it doesn't "destroy" the hash function.
Observation 2: $\mathsf{C}$ is an oracle circuit because $V_{\mathsf{ARG}}$ make oracle queries.
Solution: Forward all the queries of $\mathsf{C}$ by asking $P_{\mathsf{ARG}}$ to attach $\mathsf{C}$'s "query-answer trace" in the proof.

Forwarding the queries makes the proof non-succinct

Hash-based PCD → (Idealize) → **NON-SUCCINCT** PCD in the ROM with straightline extraction ← (Recursive proof composition) ← **NON-SUCCINCT** relativized NARK in the ROM with straightline extraction

Our theorem: $\kappa(\lambda, \mathsf{q}, \mathsf{D}, \mathsf{N}) \leq \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q}, \mathsf{N}) = \kappa_{\mathsf{ARG}}(\lambda, \mathsf{q})$

# TL;DR

What is *proof-carrying data* (PCD)?
- Recursive compositions of SNARKs.
- It's useful for efficiently verifying distributed computations.

**Problem:**
- PCD is deployed under the assumption "security of PCD" = "security of underlying SNARK".
- BUT existing security analyses show a huge gap in security ("PCD is far less secure than underlying SNARK").

**This work:**
- We propose **an idealized PCD** that models hash-based PCD in practice.
- We prove that this idealized PCD is **as secure as its underlying SNARK**.

**Thank you!**