ECLIPSE*:

Better Commit-and-Prove SNARKs with Universal SRS

Diego F. Aranha[1], Emil Madsen Bennedsen[2], **Matteo Campanelli**[3], Chaya Ganesh[4], Claudio Orlandi[1], Akira Takahashi[1]

*Enhanced Compiling Method for Pedersen-Committed zkSNARK Engines

https://ia.cr/2021/934

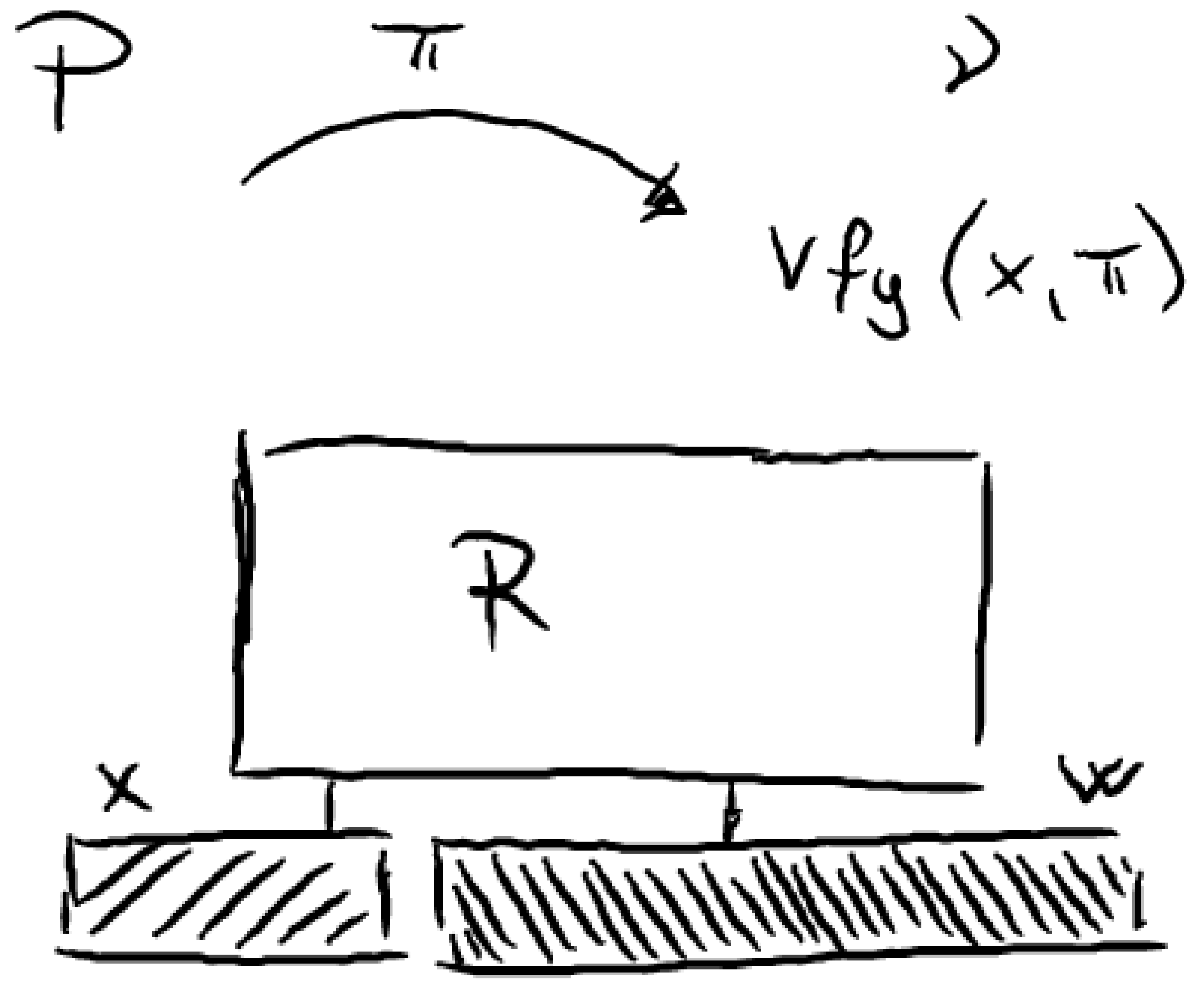
[1] Aarhus University
[2] Concordium
[3] Protocol Labs
[4] Indian Institute of Science

Our Setting

- Succinct and non-interactive ZK (SNARKs)
- Commit-and-Prove (CP-SNARK)
- Universal Trusted Setup

Succinct and Non-Interactive ZK

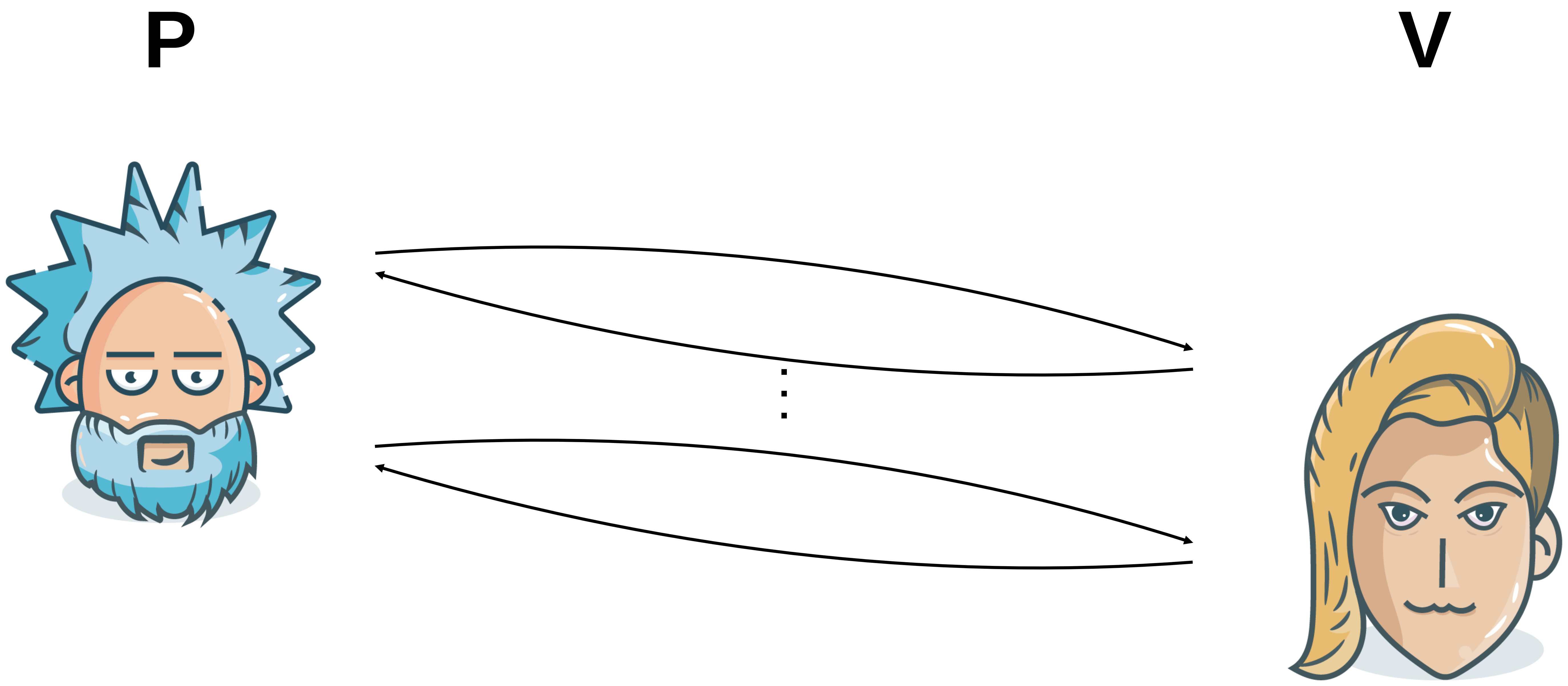
SNARK



e.g., x = msg, w = signature

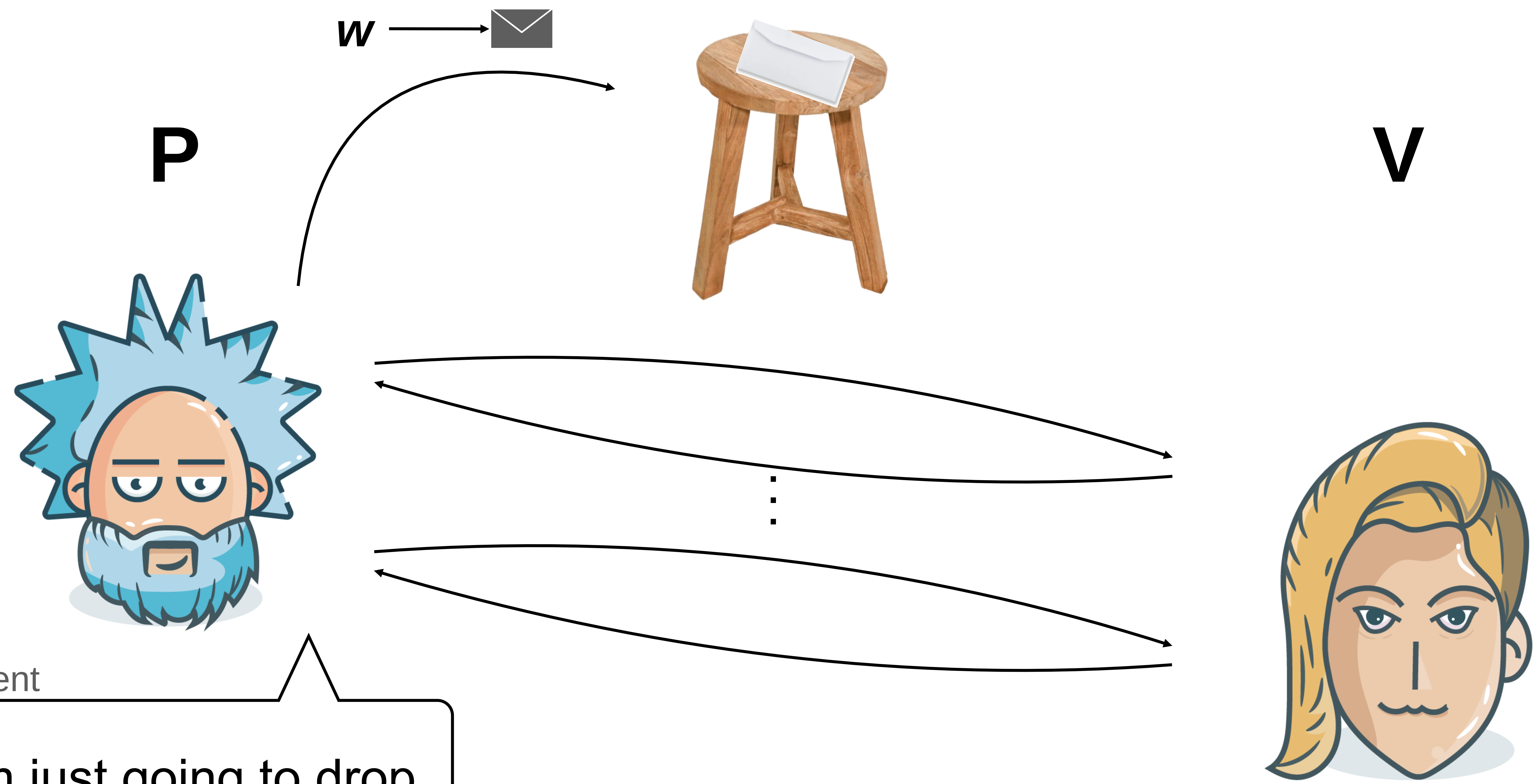
ZK

“Trusting someone else’s claims on data that you have not seen”



OK, P must *know* ***w*** such that ***R(w)*** holds.

Commit-and-Prove (CP) ZK



"Trusting someone else's claims on data that you have not seen ***but that can be pointed to"***

Commitment

First, I'm just going to drop this here…

OK, P must *know* ***w*** such that ***R(w)*** holds…

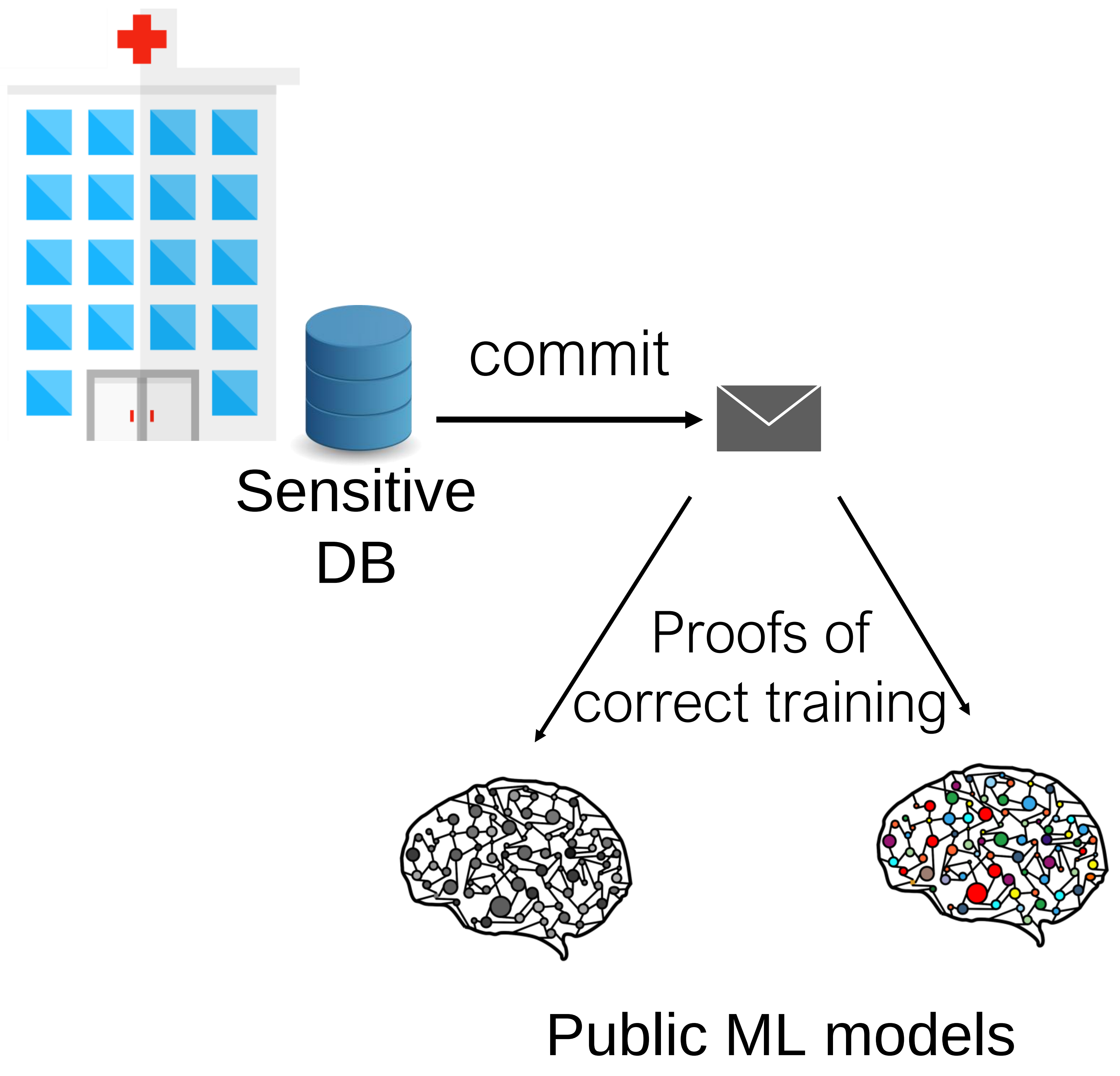
and such that

✉ $\xrightarrow{\texttt{opens}}$ w

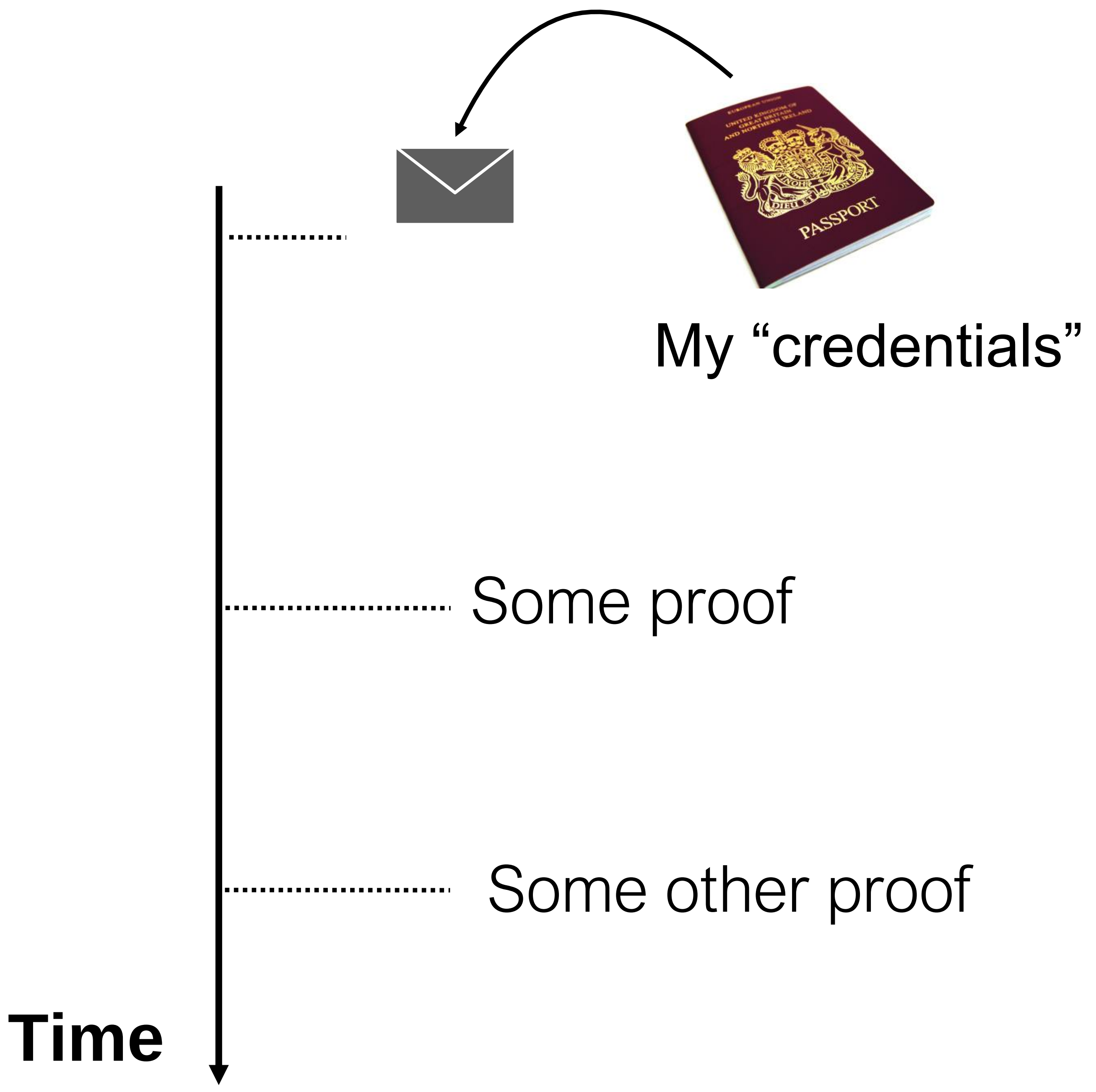
In CP-ZK we prove R
and we open a commitment

Motivation for CP

Compression/ Fingerprinting

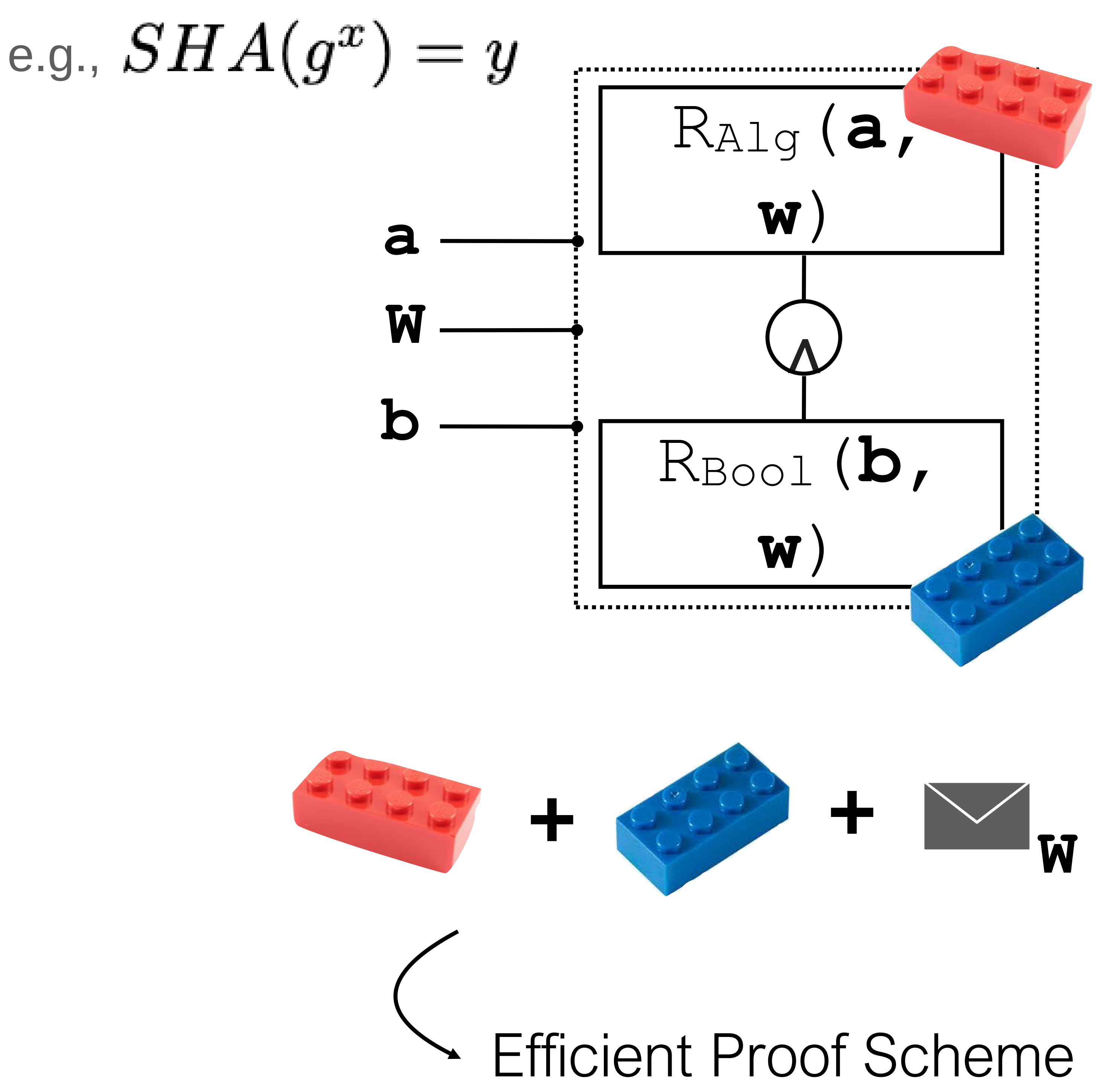


Commit-ahead-of-time



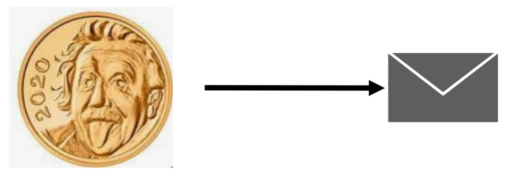
Modular/efficient composition of proofs

[AGM18,**C**FQ19]



Some Applications

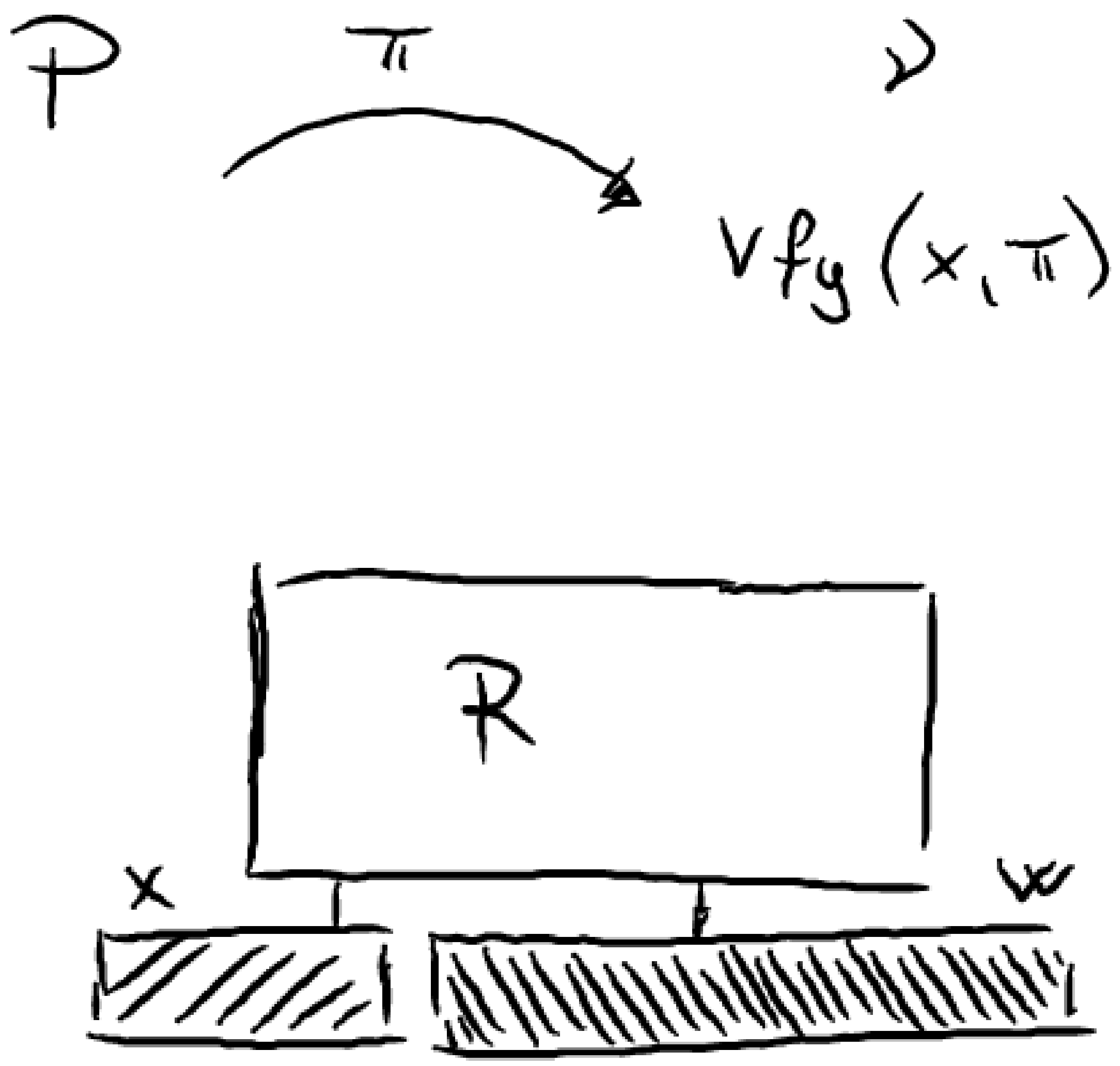
- **Anonymous Credentials** 

- **Blockchains:**
  - with privacy properties 
  - proofs on data posted on blockchains

- Generally: anywhere data need to be referenced to (privately or succinctly)

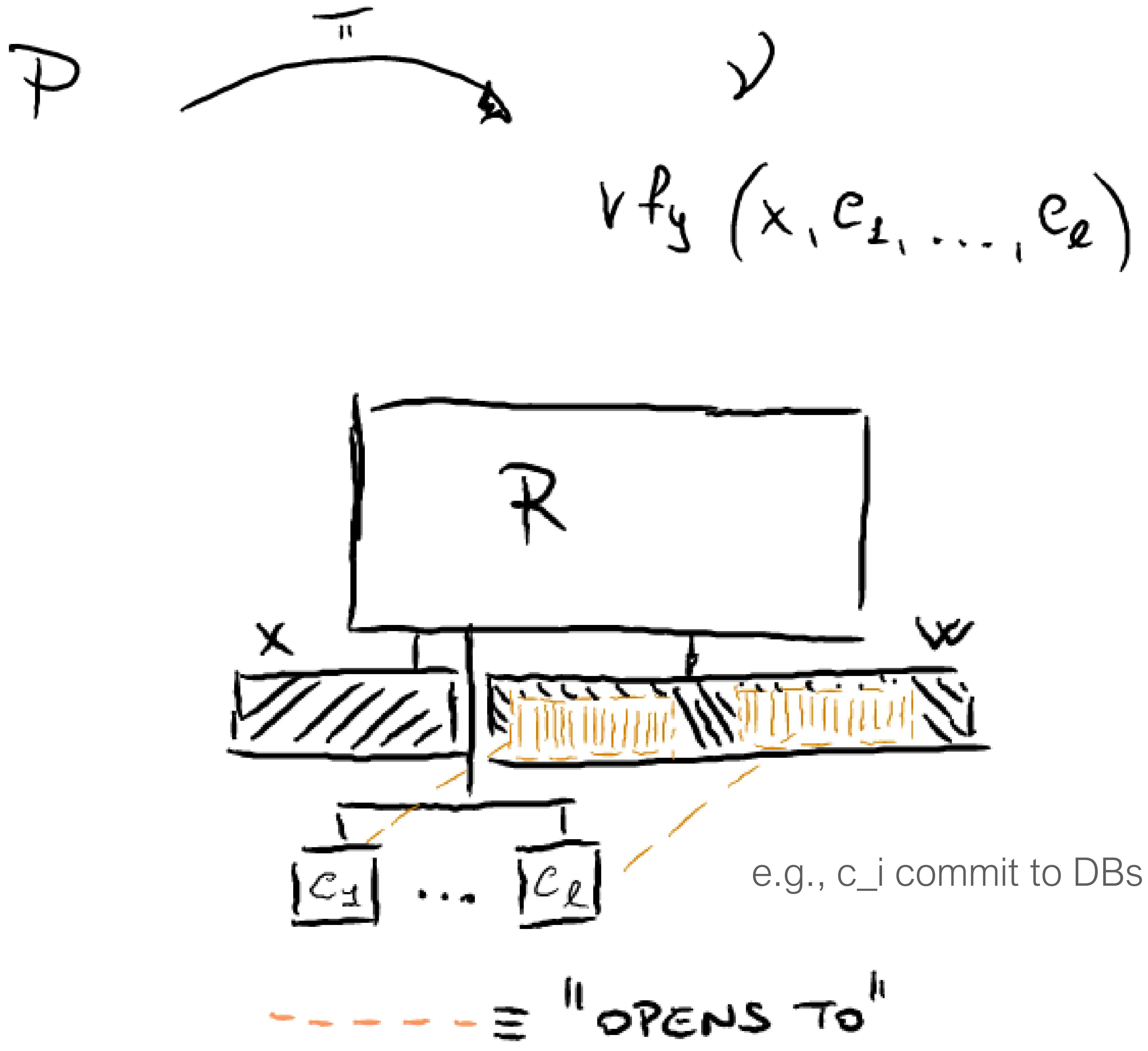
Syntax: SNARKs vs CP-SNARKs

SNARK



e.g., x = msg, w = signature

CP-SNARK



e.g., c_i commit to DBs

Setting on the right is a special case of the other. Then why care?? **Efficiency & interoperability**

Clarifying (our) CP-SNARK Setting

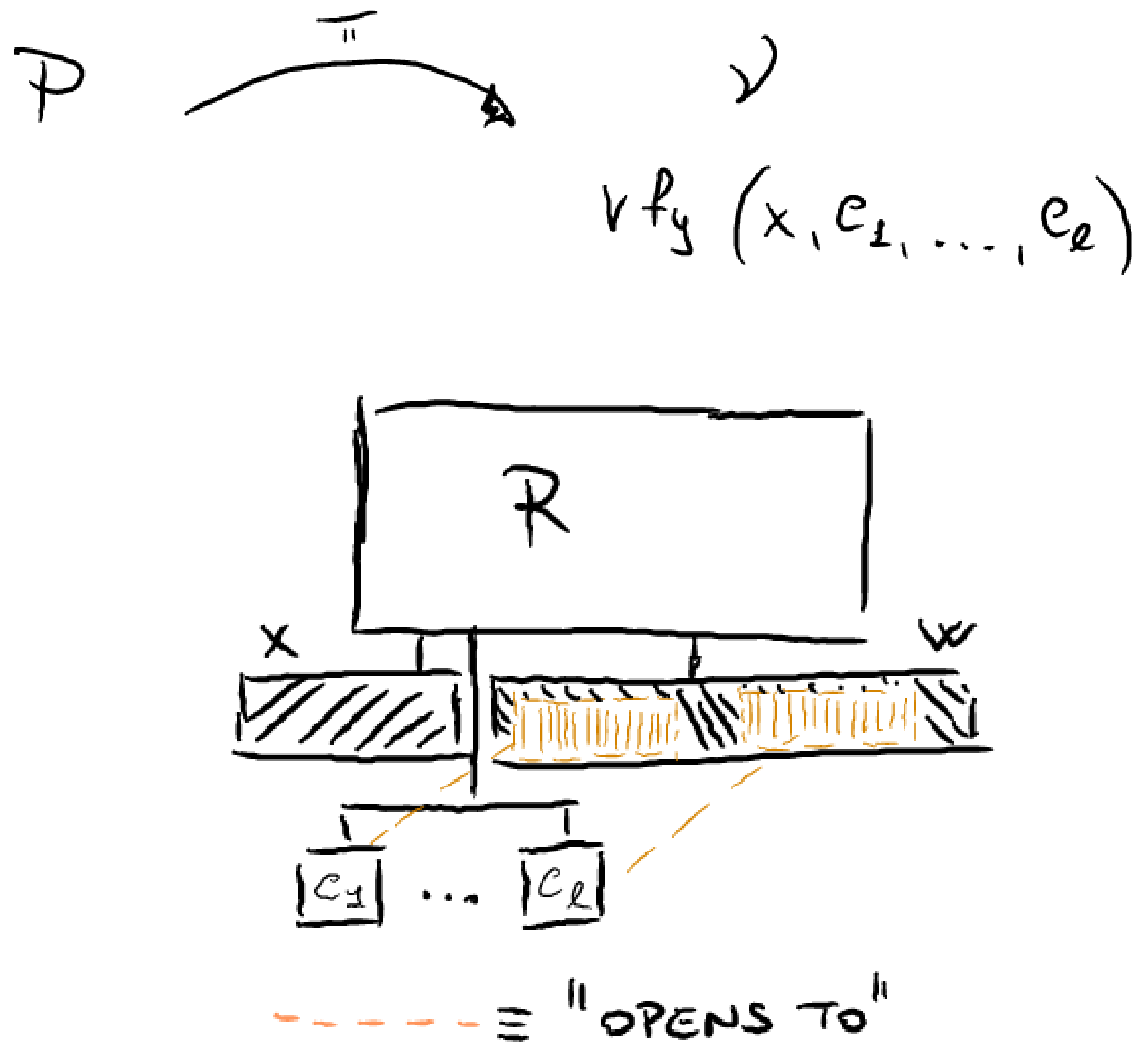
Desiderata ([CFQ18,ZKProof]:
- Efficient ZK opening
- Interoperable commitments (as standard as possible)

UNSATISFACTORY SOLUTIONS:

- USE MERKLE TREES OR PEDERSEN TO COMMIT, THEN 'OPEN' IN CIRCUIT

STANDARD COMMITMENTS

EXPENSIVE



Clarifying (our) CP-SNARK Setting

Desiderata ([CFQ18,ZKProof]:

- Efficient ZK opening
- Interoperable commitments (as standard as possible)

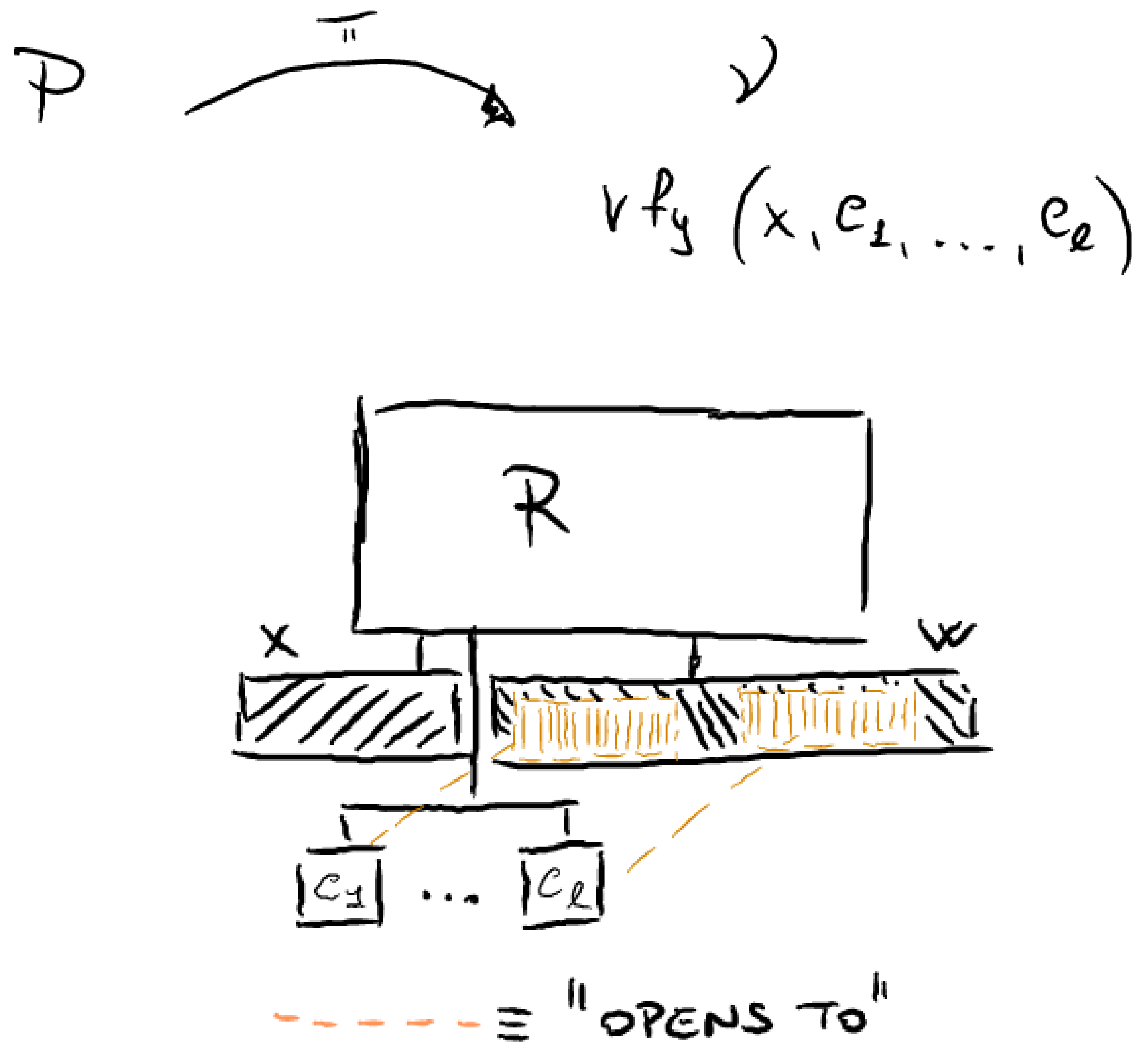
UNSATISFACTORY SOLUTIONS:

- USE MERKLE TREES OR PEDERSEN TO COMMIT, THEN OPEN IN CIRCUIT

  ☺ STANDARD COMMITMENTS ☹ EXPENSIVE

- AS ABOVE BUT WITH SMART ARITHMETIC / ELLIPTIC CURVES (e.g. ZCASH (JUBJUB), [COCO], [VEKSEL] (JABBERWOCKY))

  ☺ (MORE) EFFICIENT ☹ CURVE DEPENDENT



Trust Models in SNARKs (and CP-SNARKs)

- **Transparent :-)))** (Bulletproofs,Hyrax,DARK…)
  - no trusted setup
- **SRS (Structured Reference String) :-|** (Pinocchio,Groth16…)
  - Keygen(R) -> srs_R
- **Universal SRS (USRS) :-)** (GKMM18,LegoSNARK,Sonic,Marlin,PLONK,…)
  - Keygen(maxSize) -> srs_gen
  - Specialize(srs_gen, R) -> srs_R
  - Often also **updatable** (anyone can rerandomize srs_gen)

Eclipse results from 10^9 feet:

new ways to construct CP-SNARKs with a Universal SRS generically

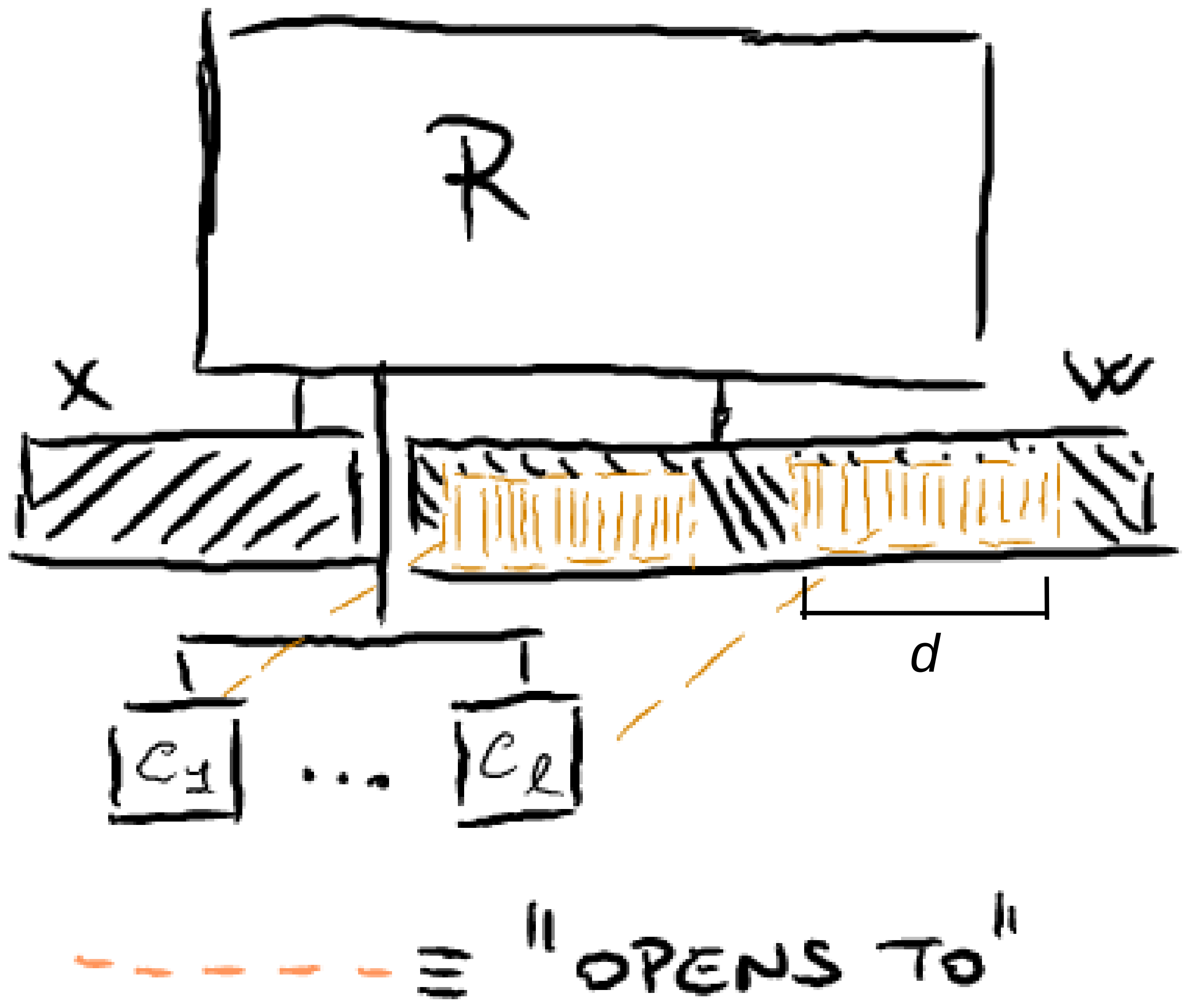
Summary of Our Results

- **General Compiler into CP-SNARKs with Universal SRS**
  - Your favorite SNARK* with USRS -> CP-SNARK
  - * in "information-theoretic" form (more on that later)
- **CP versions of Marlin, PLONK, and SONIC**
  - commitment type = Pedersen
- **All with small overhead** (next slide)

Resulting USRS CP-SNARKs—Efficiency

	$\lvert\pi\rvert$	Prove (time)	Verify (time)
ECLIPSE [ABC+21]	$O(\log(\ell \cdot d))$	$O(n + \ell \cdot d)$	$O(\ell \cdot d)$
Lunar [CFF+20]	$O(\ell)$	$O(n + \ell \cdot d)$	$O(\ell)$
LegoUAC [CFQ19]	$O(\ell \log^2(n))$	$O(n) + \ell \cdot \tilde{O}(d)$	$O(\ell \log^2(n))$

Time is in group operations. Above, n is roughly # of multiplication gates

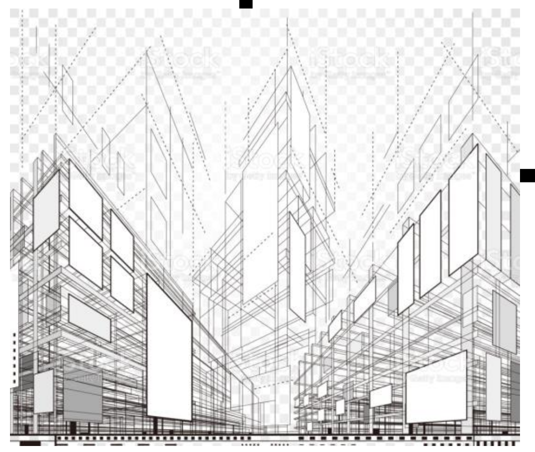


In practice the two family of systems show a tradeoff in verification time/proof size.

Constructing (USRS) SNARKs

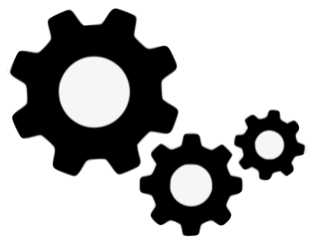
Compilers from idealized information-theoretic objects

Information-theoretic Object



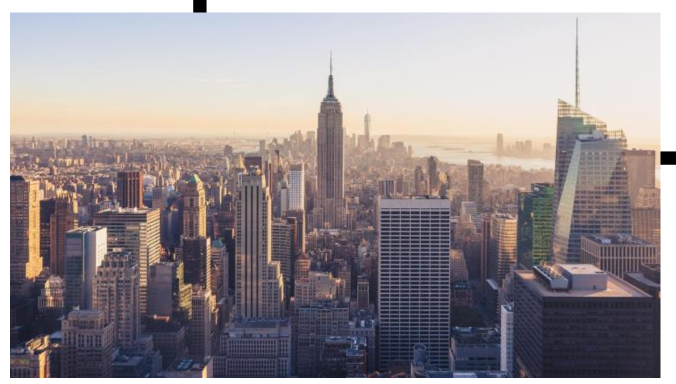
+

Crypto primitive



Compilation →

Cryptographic Proof System

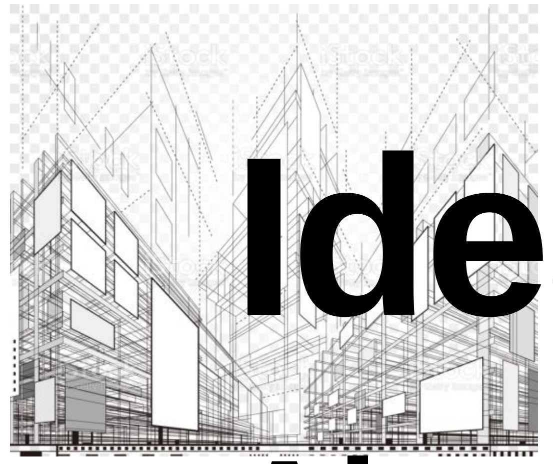


Practical* SNARKs with Universal SRS

zkSNARK		size		time	
		$\lvert vk_R \rvert$	$\lvert \pi \rvert$	Prove	Verify
Sonic [46]	$\mathbb{G}_1$	—	20	$273n$	7 pairings
	$\mathbb{G}_2$	3	—	—	
	$\mathbb{F}$	—	16	$O(m \log m)$	$O(\ell + \log m)$
MARLIN [20]	$\mathbb{G}_1$	12	13	$14n+8m$	2 pairings
	$\mathbb{G}_2$	2	—	—	
	$\mathbb{F}$	—	8	$O(m \log m)$	$O(\ell + \log m)$
PLONK (small proof) [28]	$\mathbb{G}_1$	8	7	$11n+11a$	2 pairings
	$\mathbb{G}_2$	1	—	—	
	$\mathbb{F}$	—	7	$O((n+a)\log(n+a))$	$O(\ell + \log(n+a))$
PLONK (fast prover) [28]	$\mathbb{G}_1$	8	9	$9n+9a$	2 pairings
	$\mathbb{G}_2$	1	—	—	
	$\mathbb{F}$	—	7	$O((n+a)\log(n+a))$	$O(\ell + \log(n+a))$

Roughly:
- n: # MUL gates
- a: # ADD gates
- m: # wires

*practical + focus is on O(1) proof size

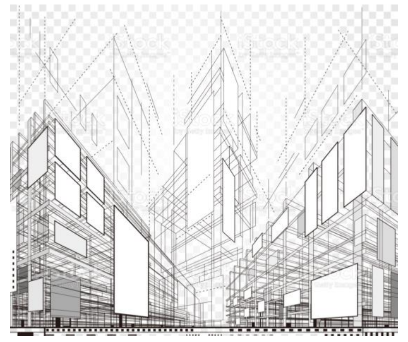


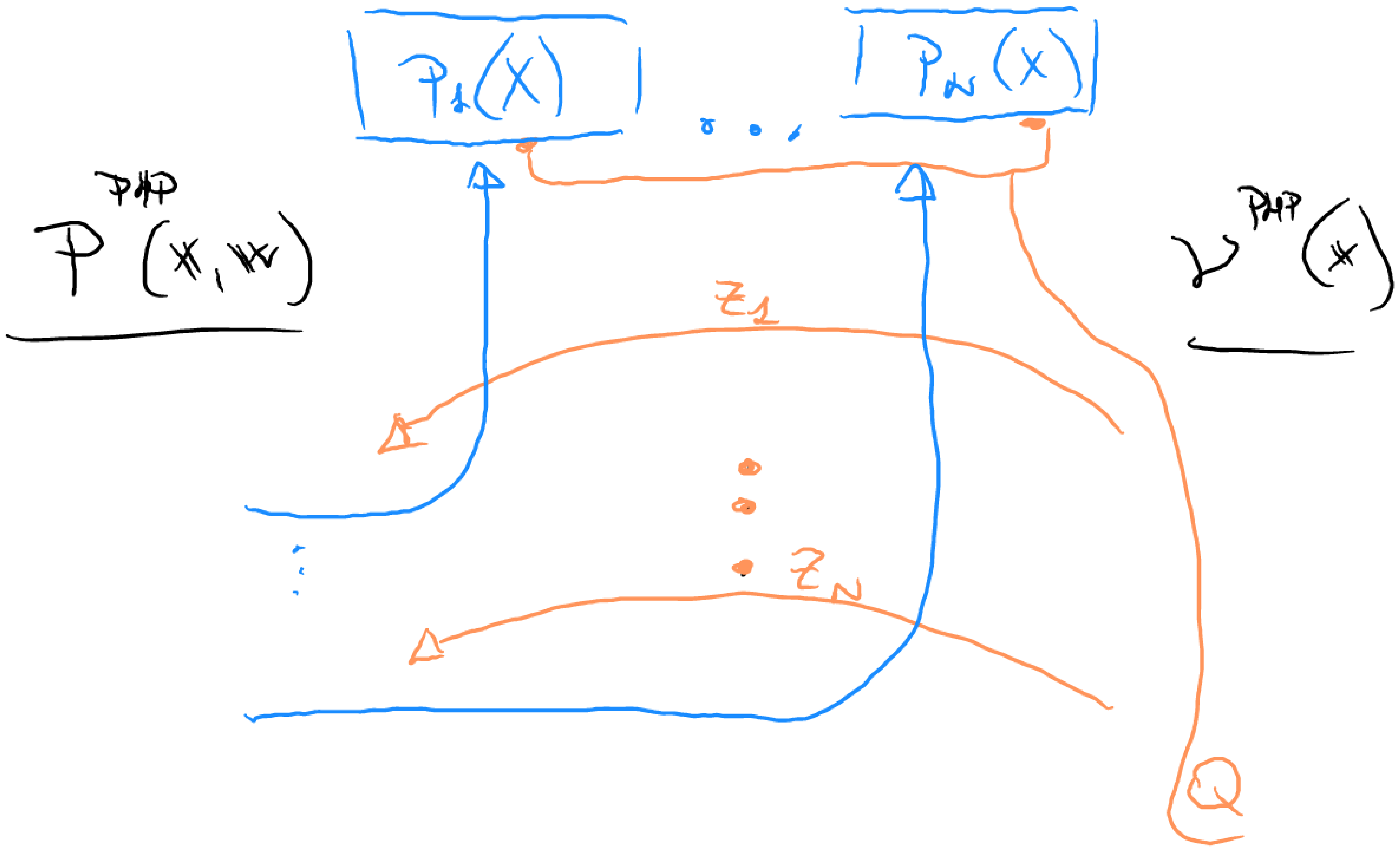
Idealized protocols for USRS SNARKs

Algebraic Holographic Proofs (AHPs)

- Interactive
- Prover holds polynomials "encoding" the witness
- It gives oracle access to their evaluations

A picture of the idealized protocol

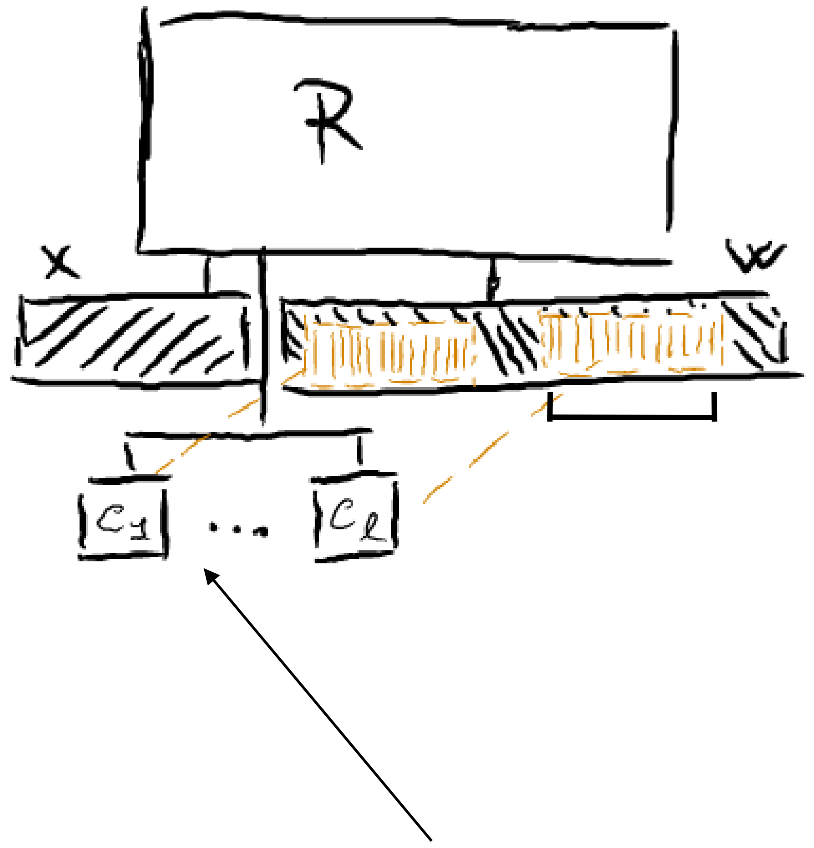




Queries Q:
Evaluations of polynomial (e.g. p1(x*) == t*)

Compiling to USRS SNARKs: Ingredients

- (Underlying compiler in Marlin/DARK/Lunar/PLONK)
- Main tool is a **Polynomial Commitment** PC: 
  - with *compressing* commitment to polynomials
  - Allows proving efficiently (and succinctly) in ZK:
    - p(x) = y (evaluation)
    - (plus degree bounds: deg(p) <= Dbound)

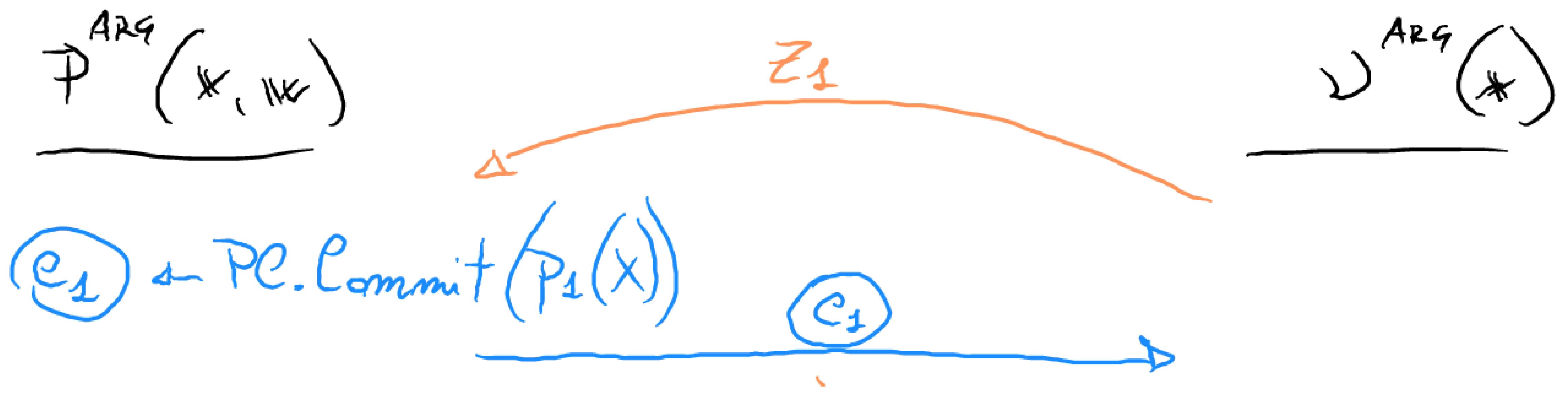


NB: different from these commitments!

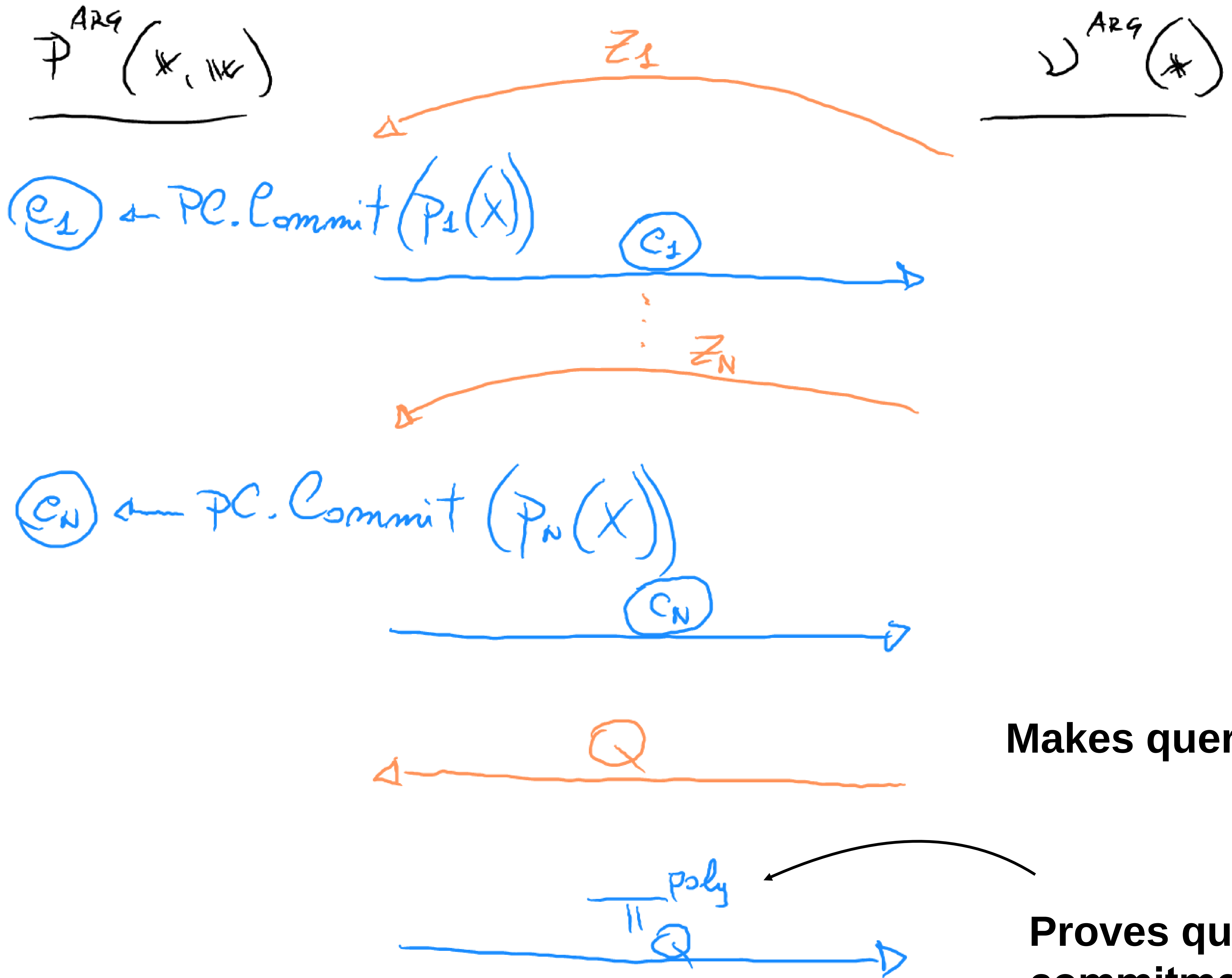
Notation (circles for polynomial commitment)

$c_1 \leftarrow PC.Commit(p_1(X))$

Compiling to USRS SNARKs



Compiling to USRS SNARKs



Makes queries Q.

Proves queries Q are satisfied by poly commitments c1,…,cN

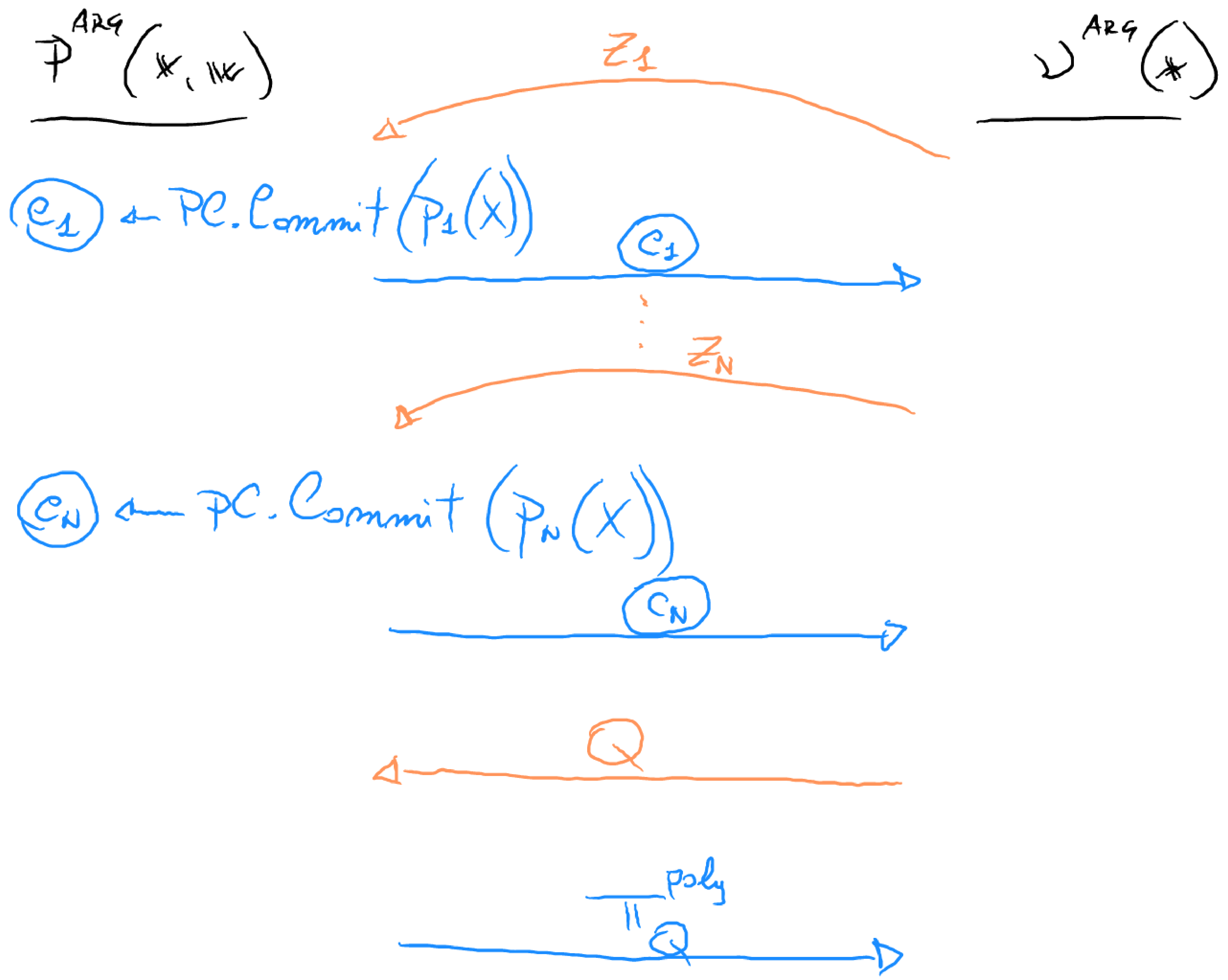
The Resulting USRS SNARKs

- Use Fiat-Shamir for non-interaction
- Why is the SRS *Universal*?
  - Because we can define

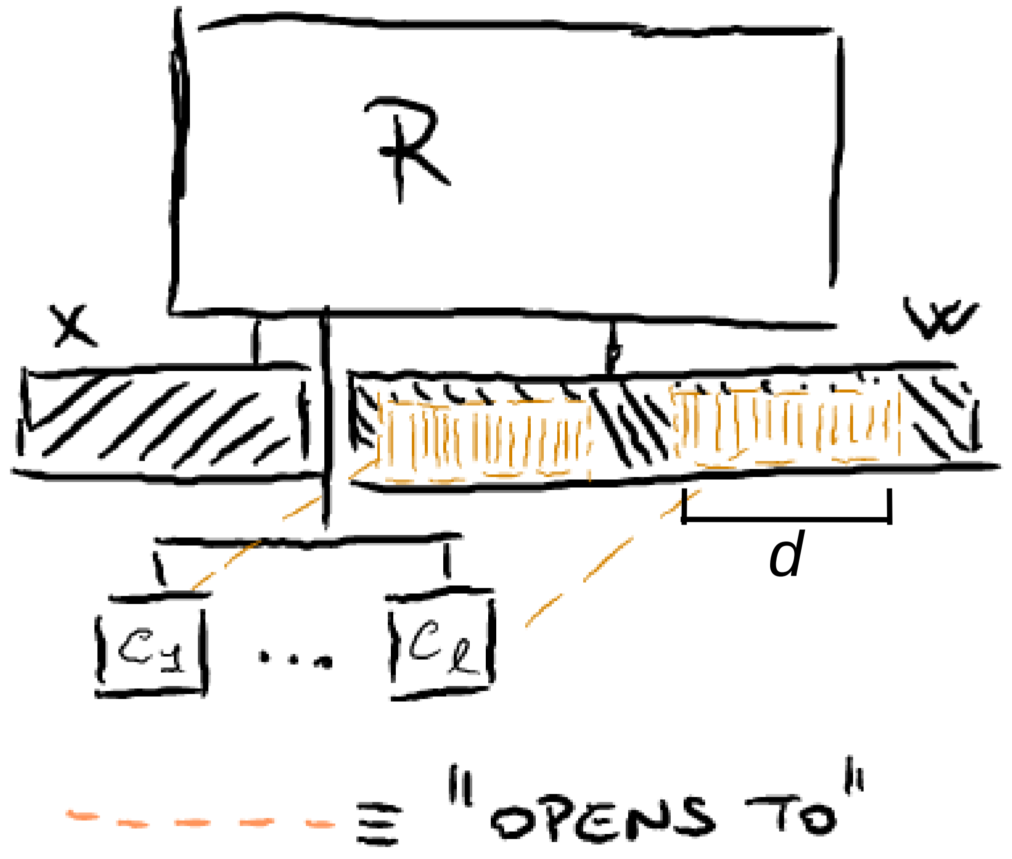
    SNARK.Setup(maxSize) ->

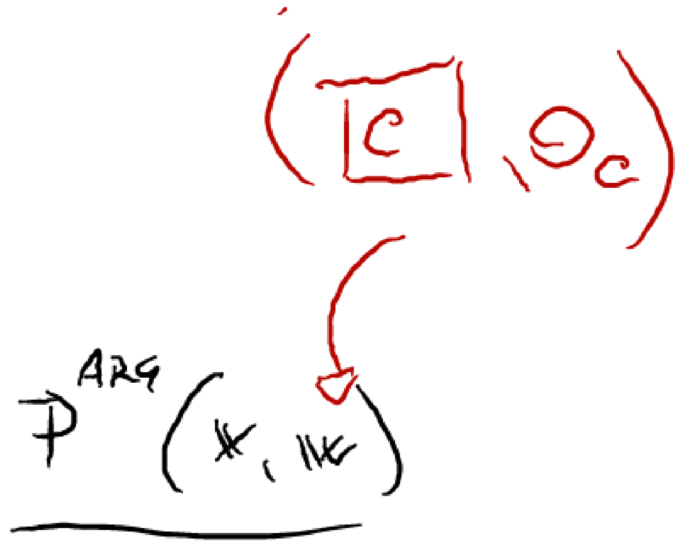
    srs_gen := PC.Setup(maxPolyDeg)

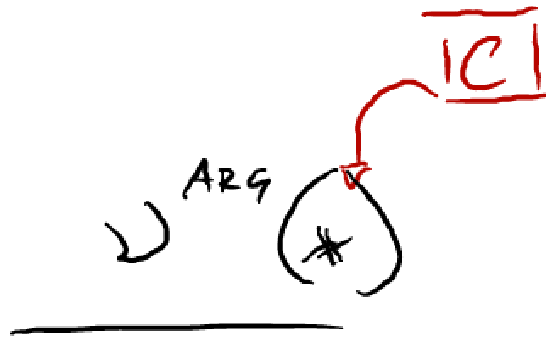
  - Where maxPolyDeg depends on maxSize



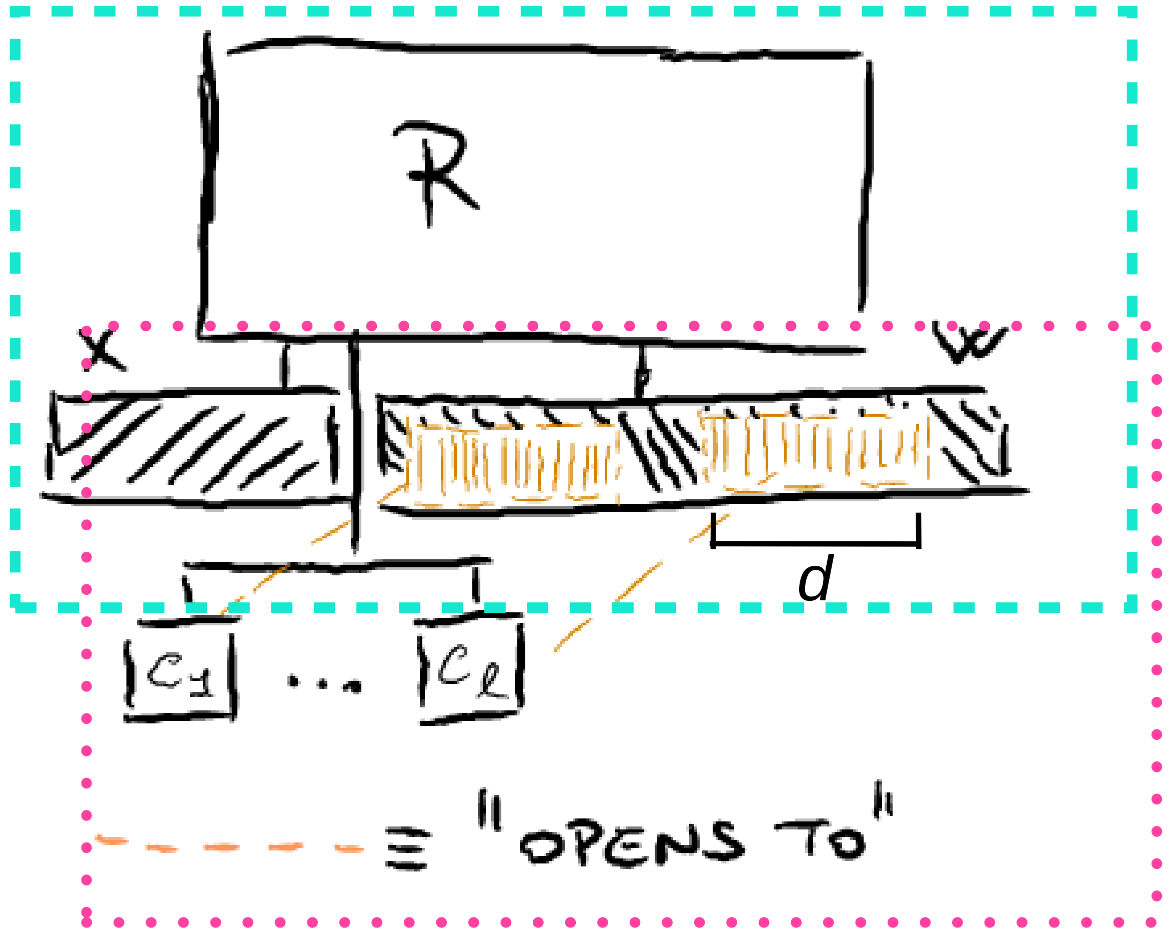
Compiling into CP-SNARKs

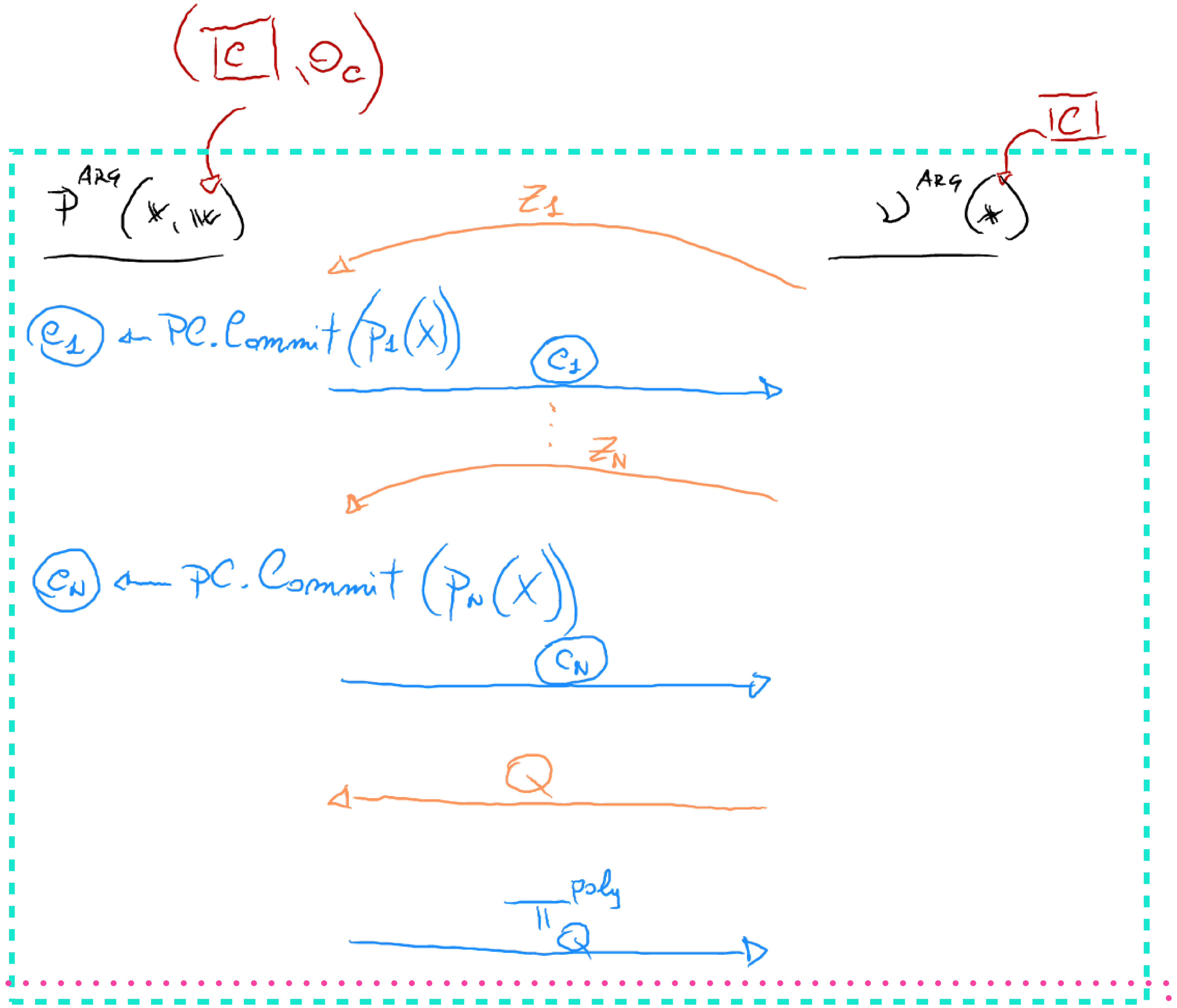






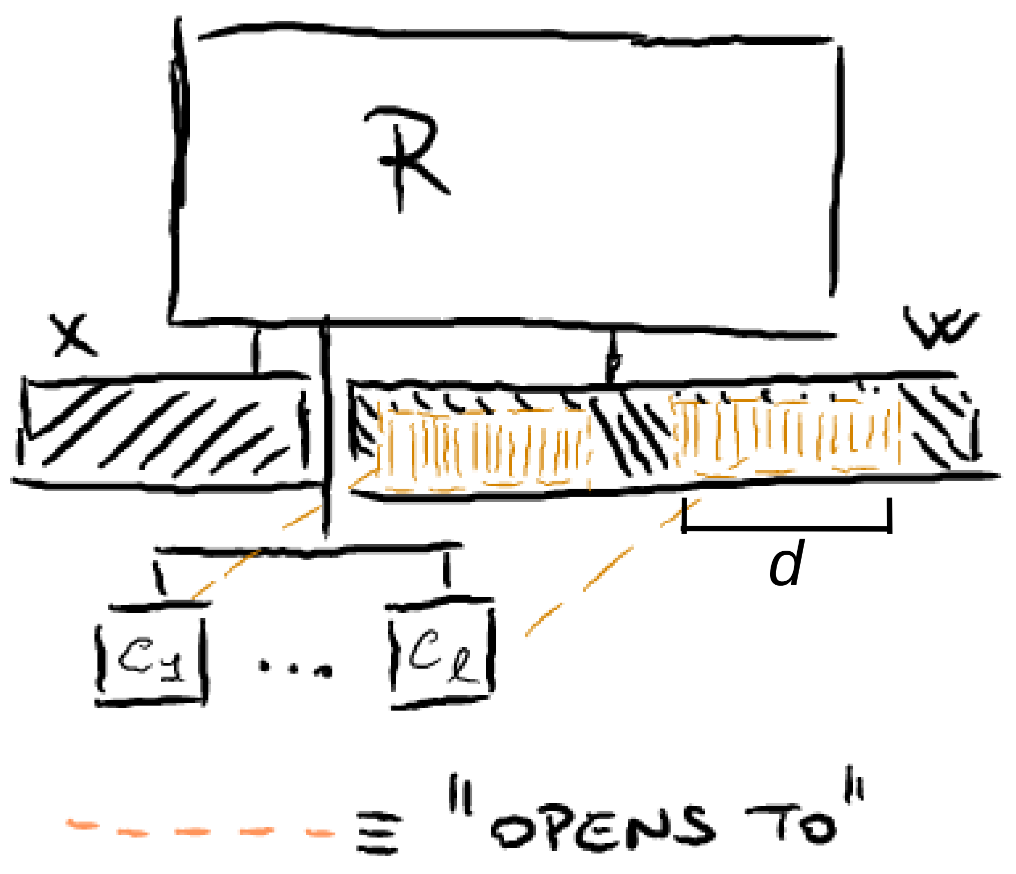
Compiling into CP-SNARKs

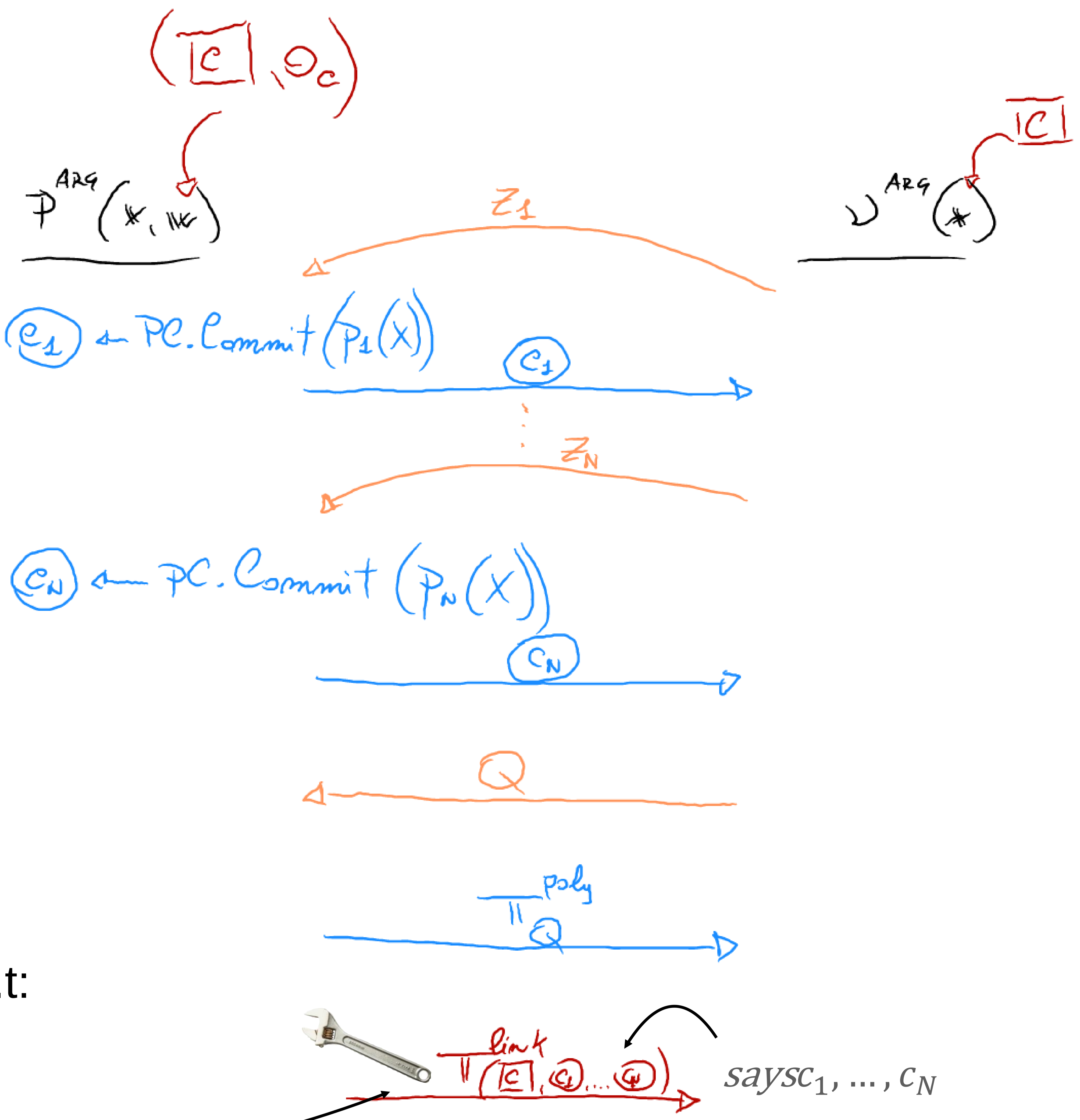




???

Compiling into CP-SNARKs





It proves "linking", or: knowledge of **w** s.t:

1) [c] opens to (parts of) w
2) (c_i) opens to p_i, forall i
3) w is "consistent with the execution"

Challenge 1: depending on only part of the witness

From previous slide

It proves "linking", or: knowledge of **w** s.t:
1) [c] opens to (**parts of**) w
2) (c_i) **opens to p_i**, forall i
3) w is "consistent with the execution"

Our solution:
showing that (c_i) can be additively decomposed in our SNARKs of interest

Definition 9 (Decomposable witness-carrying polynomials). *Let W be an index set of witness-carrying polynomials of* AHP. *We say that polynomials* $(p_{i,j}(X))_{(i,j)\in W}$ *of* AHP *are* decomposable *if there exists an efficient function* $\mathsf{Decomp}((p_{i,j}(X))_{(i,j)\in W}, I) \to (p^{(1)}_{i,j}(X), p^{(2)}_{i,j}(X))_{(i,j)\in W}$ *such that it satisfies the following properties for any* $I \subset [n]$.

– Additive decomposition: $p_{i,j}(X) = p^{(1)}_{i,j}(X) + p^{(2)}_{i,j}(X)$ *for* $(i,j) \in W$.

– Degree preserving: $\deg(p^{(1)}_{i,j}(X))$ *and* $\deg(p^{(2)}_{i,j}(X))$ *are at most* $\deg(p_{i,j}(X))$ *for* $(i,j) \in W$.

– Non-overlapping: *Let* $\mathsf{w} = \mathsf{WitExt}((p_{i,j}(X))_{(i,j)\in W})$, $\mathsf{w}^{(1)} = \mathsf{WitExt}((p^{(1)}_{i,j}(X))_{(i,j)\in W})$, *and* $\mathsf{w}^{(2)} = \mathsf{WitExt}((p^{(2)}_{i,j}(X))_{(i,j)\in W})$. *Then*

$$(\mathsf{w}_i)_{i\in I} = (\mathsf{w}^{(1)}_i)_{i\in I} \quad (\mathsf{w}_i)_{i\notin I} = (\mathsf{w}^{(2)}_i)_{i\notin I} \quad (\mathsf{w}^{(1)}_i)_{i\notin I} = 0 \quad (\mathsf{w}^{(2)}_i)_{i\in I} = 0$$

Challenge 2: efficient and succinct proof of linking

From previous slide

It proves "linking", or: knowledge of **w** s.t:
1) [c] opens to (parts of) w
2) (c_i) opens to p_i, forall i
3) w is "consistent with the execution"

- **Our solution:**
  - Prove through an (amortized) Sigma-protocol a "squashing" of the input commitments

$$C = \boldsymbol{g}^{\mathbf{w}}\mathbf{h}^{\boldsymbol{\alpha}}, \hat{C}_i = \mathbf{G}^{\mathbf{w}_i}\mathbf{H}^{\boldsymbol{\beta}_i},\ \mathbf{w} = [\mathbf{w}_1, \ldots, \mathbf{w}_\ell]$$

  - naively requires O(|w| · #commitments) communication, but we then compress it through Compressed-Sigma techniques [AC20] to O(log(|w| · #commitments))

Comparison with Lunar (CFFQ21)

- Similar blueprint
- Lunar uses a different pairing-based protocol for "linking"
- different tradeoffs in efficiency (see also table in the next slide)
- Lunar uses a more general formalization (PHP); our work can be easily formalized in the same framework

Open Questions

- **Better asymptotics:**
  - O(\ell) is inherent in verification time, but can we achieve constant proof size?
  - Maybe with one-level of (specialised) recursion?
- **Different techniques for “linking” and/or finding other applications for those in ECLIPSE?**

	$\lvert\pi\rvert$	Prove (time)	Verify (time)
ECLIPSE [ABC+21]	$O(\log(\ell \cdot d))$	$O(n + \ell \cdot d)$	$O(\ell \cdot d)$
Lunar [CFF+20]	$O(\ell)$	$O(n + \ell \cdot d)$	$O(\ell)$
Future?	$O(1)$		$O(\ell)$

https://ia.cr/2021/934

Thanks!

For questions/contacts:
matteo@protocol.ai
binarywhales.com