Plonk Standardisation Workshop

Mary Maller, Ethereum Foundation and PQShield

ZKProof5.5 - 02-Aug-2023

ZKPROOF

About Me

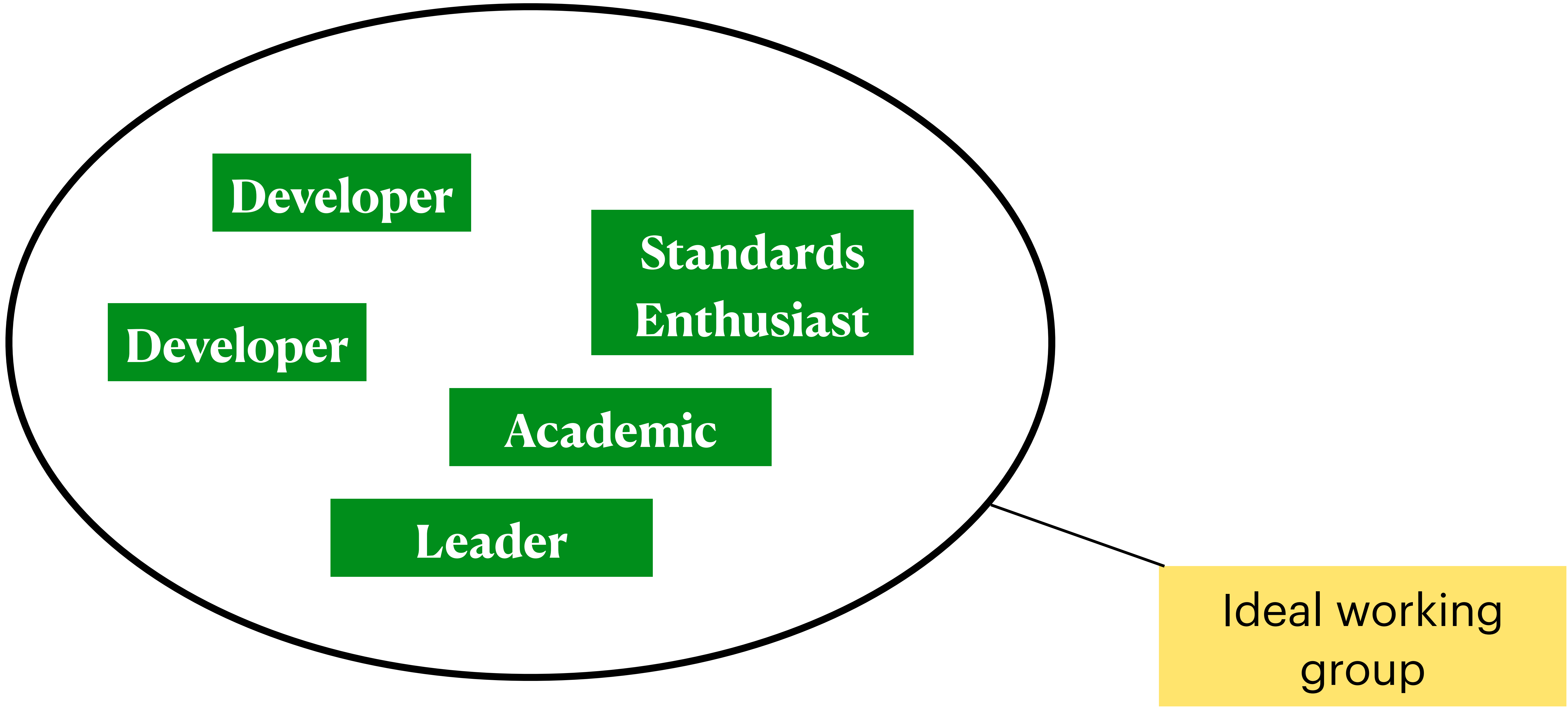
- Cryptography researcher in zero-knowledge proofs since 2015.
- Contributed to zk research projects including:
  - GM17, GKMMM18, Sonic, Marlin, IPPs, SnarkPack, Caulk, Baloo, KMV23
- Active member of ZKProof since 2021.
- Not an author on the Plonk paper.
- I care deeply about the ZKProof agenda.
- I have not yet solved the maze from Saturday's Crypto Lounge.

Workshop Outline

- Introduction to Plonk standardisation.
- The Plonkish constraint system.
- Writing constraints for key applications.
- The optimising system.
- Interactive oracle proofs.
- Polynomial commitment schemes.
- Fiat-Shamir compiler.

Not so Secret Agenda

We are inviting experts like you to form working groups and make magic happen.



Not so Secret Agenda

We are inviting experts like you to form working groups and make magic happen.

Good Developers

- You document your code.
- You like comments.
- You are using Plonk in your product.
- You are opinionated.

Not so Secret Agenda

We are inviting experts like you to form working groups and make magic happen.

Standards Enthusiasts

- You have written e.g. an RFC before.
- You want to see zero-knowledge be standardised.
- You are happy to educate zk experts about how to write specs.

Not so Secret Agenda

We are inviting experts like you to form working groups and make magic happen.

Good Academics

- You theoretically can code and can talk to developers.
- You have specific expertise in one piece of a ZKP.
- You care about readability.
- You understand this is more work than a research paper.

Not so Secret Agenda

We are inviting experts like you to form working groups and make magic happen.

Leader

- You are passionate and enthusiastic.
- You are organised and capable of herding cats.
- You will persistently remind members of calls and deliverables.
- You are a good presenter.

Not so Secret Agenda

We are inviting experts like you to form working groups and make magic happen.

Each working group should enjoy working together.

Why Plonk

- Active support from authors of the Halo2 library.
- Halo2 library used in production with a strong user base.
- Plonk has been under scrutiny since 2019.
- Folding approaches are still in early stages and less stable.

Components of a ZKP

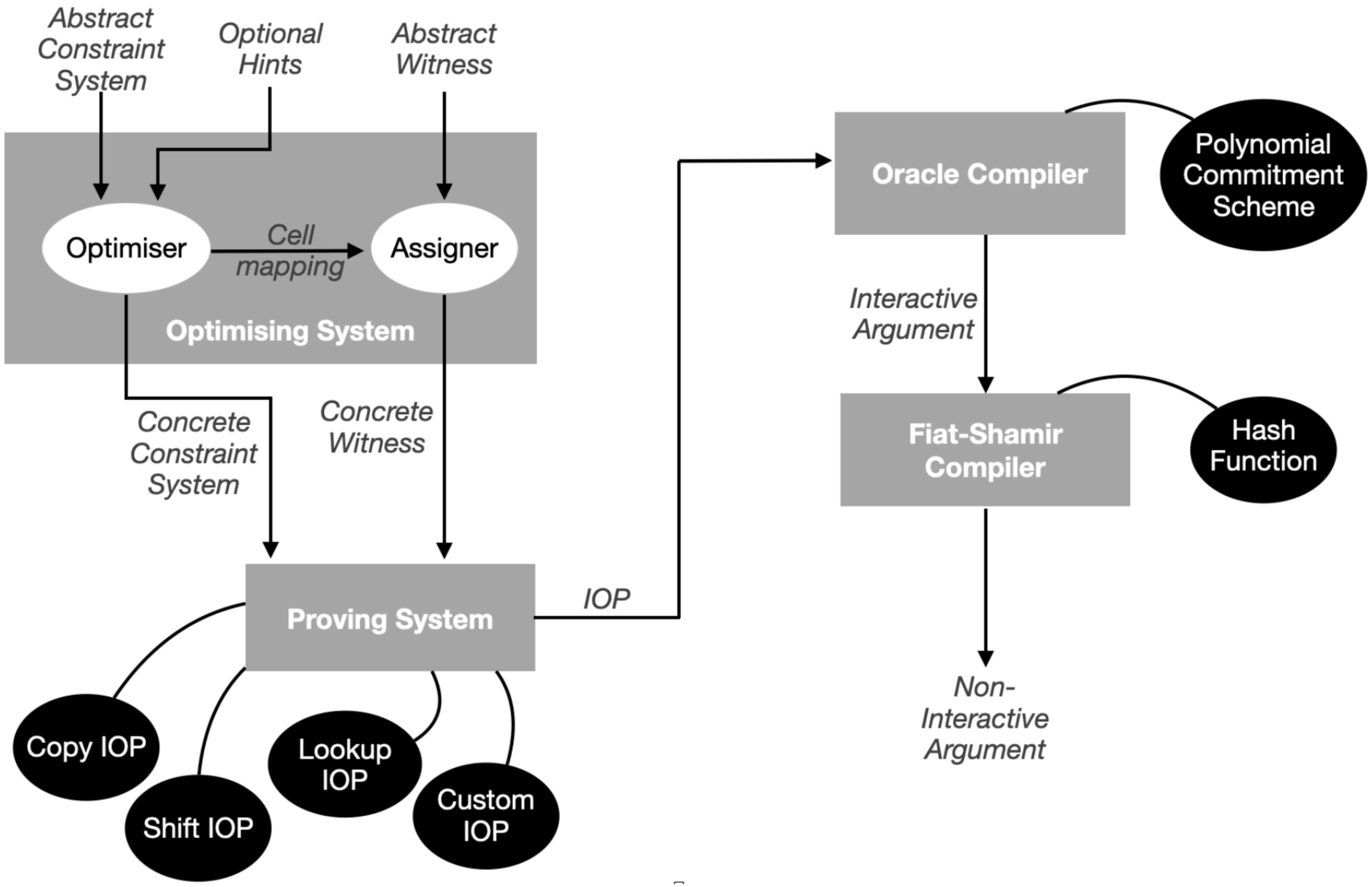
A zero-knowledge proof typically consists of the following components:

- Constraint system
- Optimising system
- Interactive oracle proof
- Polynomial commitments
- Fiat-Shamir

We will cover each of these components in detail throughout this workshop.

Components of a ZKP

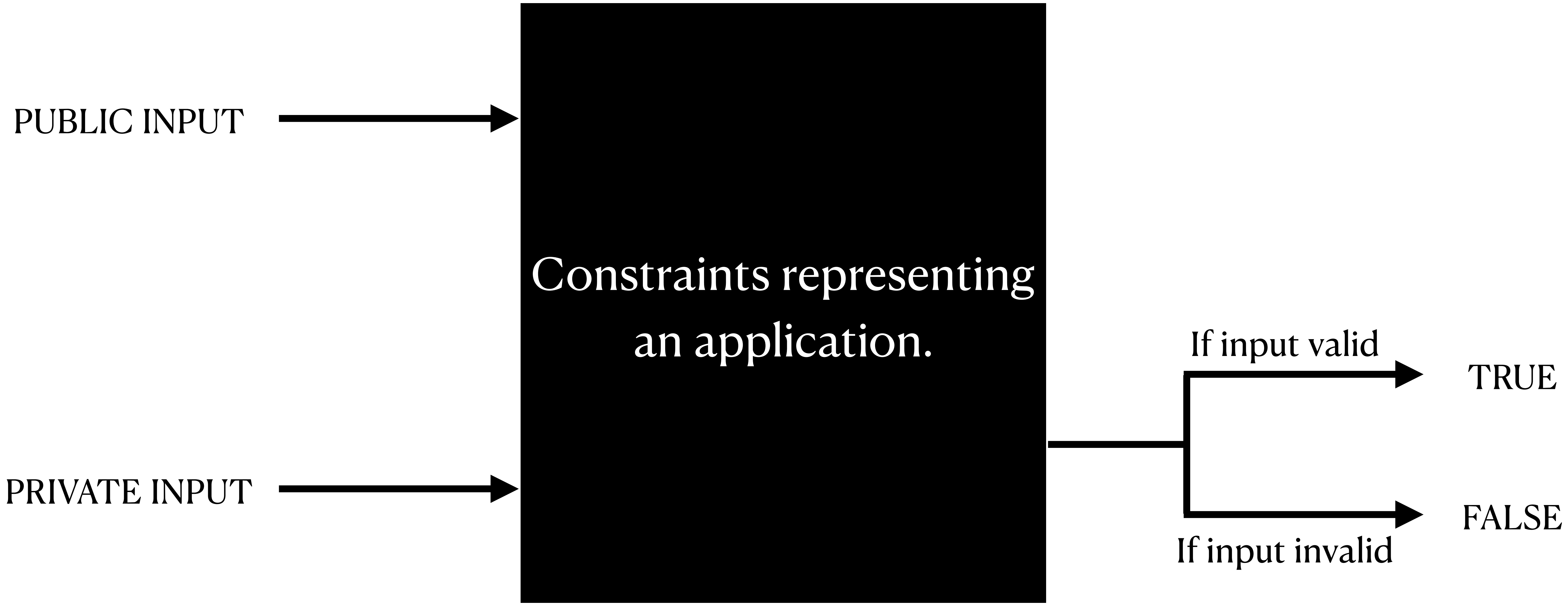
A zero-knowledge proof typically consists of the following components:



Aims of the Constraint System

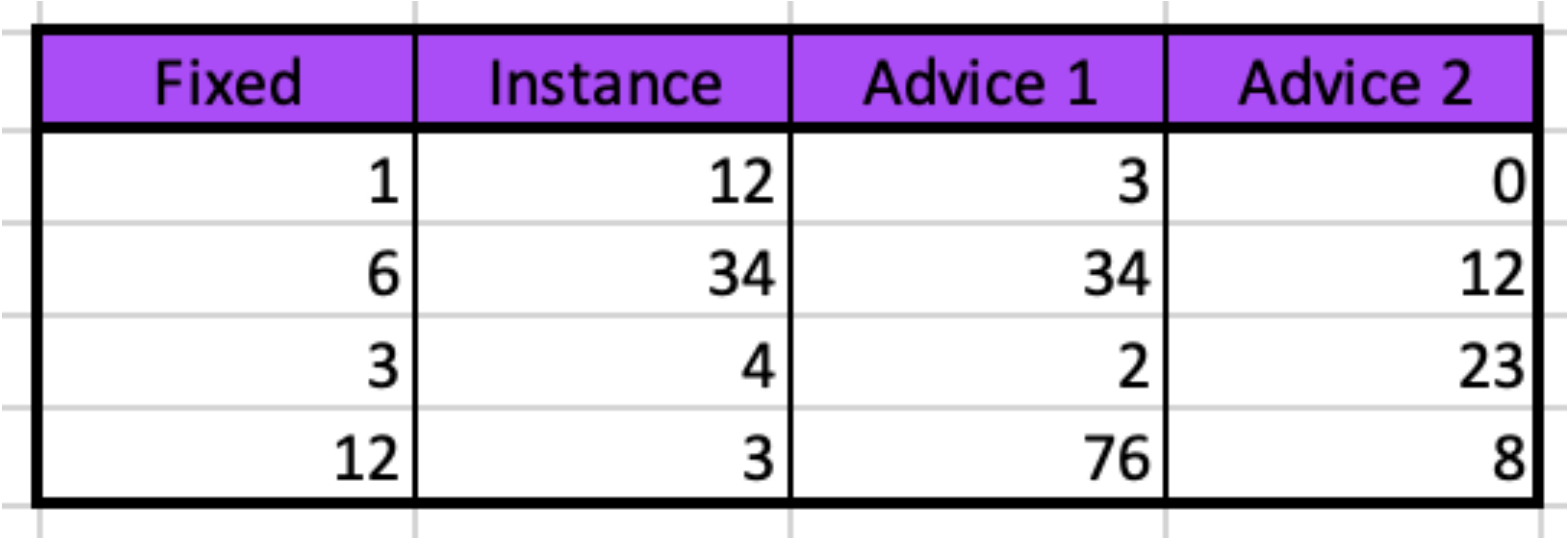
- Simplicity: A straight forward design for ease of understanding.
- Comprehensive: A clear set of rules that define the relation.
- Compatibility: Constraint system should not to be too “special”.

Constraint Systems



Maybe more intro slide first

Plonkish Constraints: Inputs

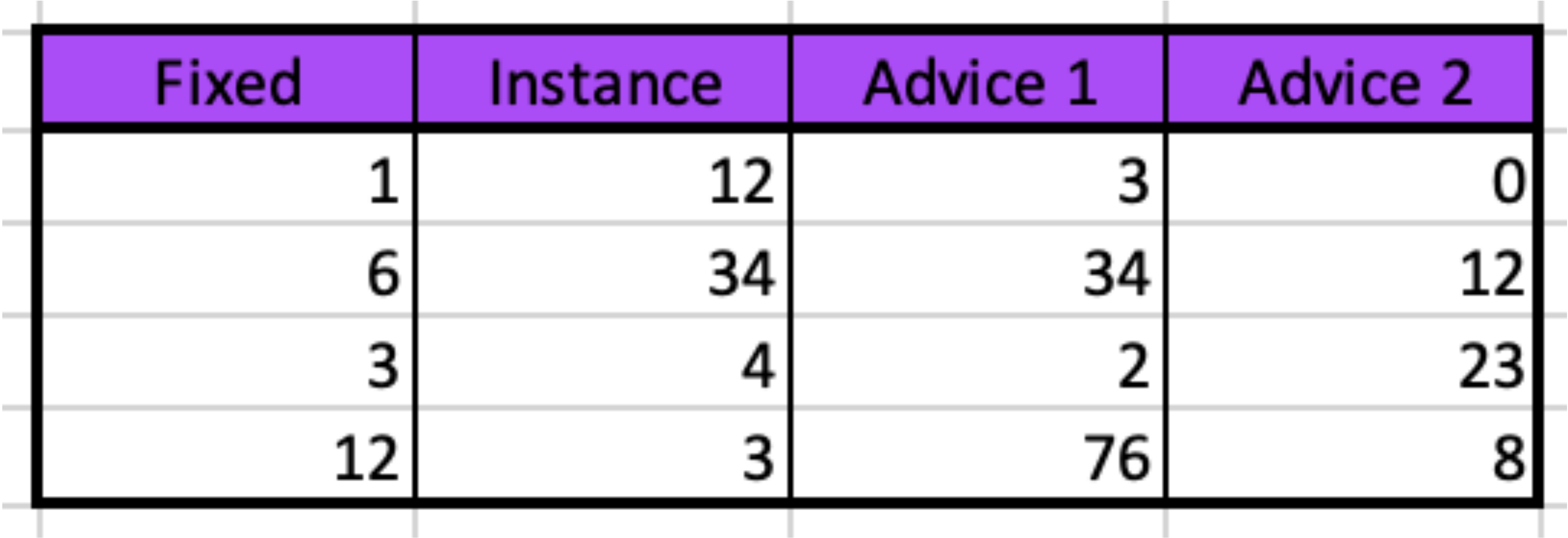


Fixed	Instance	Advice 1	Advice 2
1	12	3	0
6	34	34	12
3	4	2	23
12	3	76	8

- Fixed Columns: Given as part of the application design.
- Instance Columns: Public inputs for a specific proof.
- Advice Columns: Private inputs known only to the prover.

Plonkish Constraints

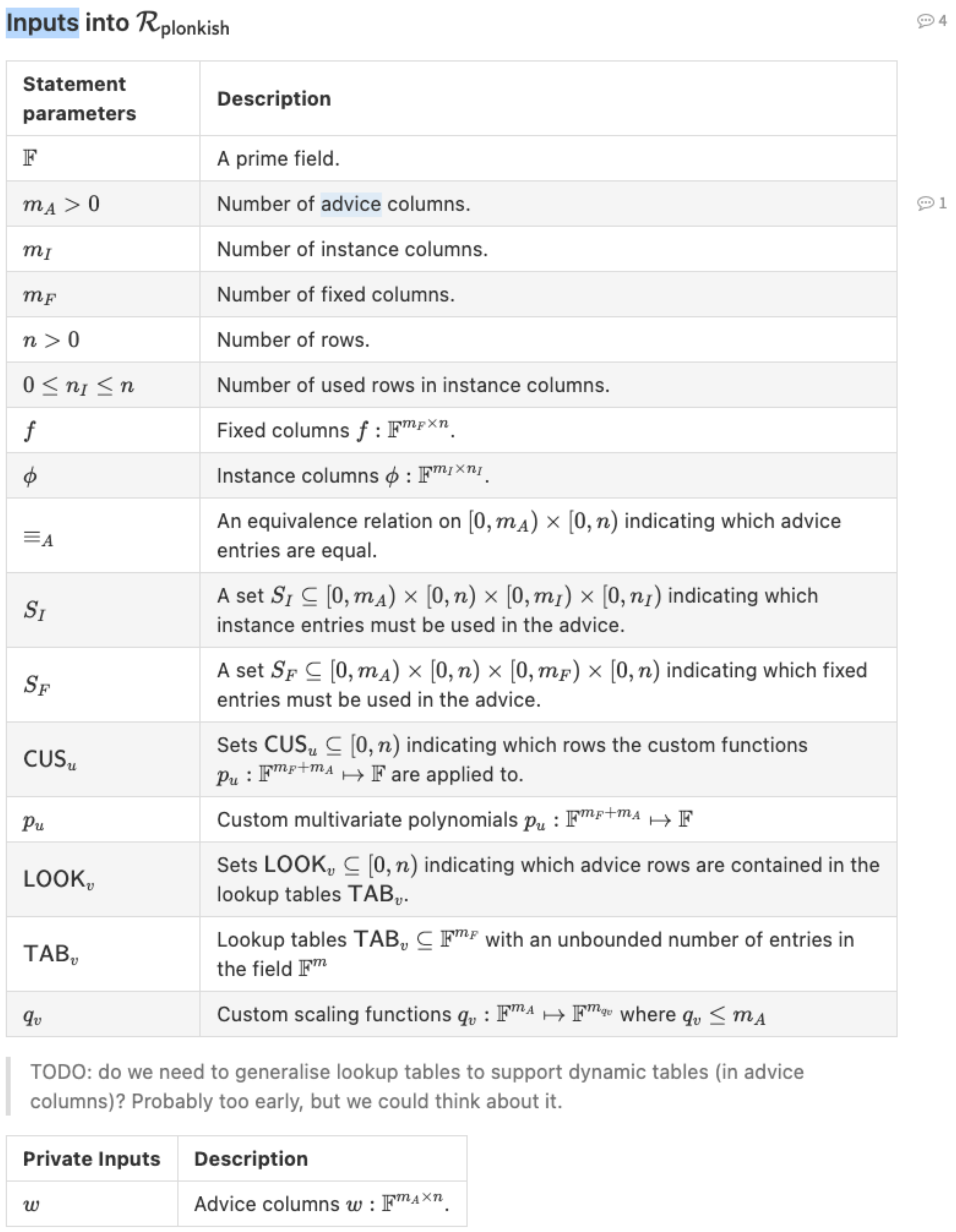
Note: I have not said rotation yet.



Fixed	Instance	Advice 1	Advice 2
1	12	3	0
6	34	34	12
3	4	2	23
12	3	76	8

- Copy: E.g. Cell[2,0] is a copy of Cell[0,2].
- Custom: E.g. (Cell[i,0] + Cell[i,1]) x Cell[i,2] x Cell[i,3] = 0
- Lookup: E.g. Cell[i,3] is included in [0, 4, 8, 12]

Plonkish Constraints



Inputs into $\mathcal{R}_{\text{plonkish}}$

Statement parameters	Description
$\mathbb{F}$	A prime field.
$m_A > 0$	Number of advice columns.
m_I	Number of instance columns.
m_F	Number of fixed columns.
$n > 0$	Number of rows.
$0 \leq n_I \leq n$	Number of used rows in instance columns.
f	Fixed columns $f : \mathbb{F}^{m_F \times n}$.
ϕ	Instance columns $\phi : \mathbb{F}^{m_I \times n_I}$.
$\equiv_A$	An equivalence relation on $[0, m_A) \times [0, n)$ indicating which advice entries are equal.
S_I	A set $S_I \subseteq [0, m_A) \times [0, n) \times [0, m_I) \times [0, n_I)$ indicating which instance entries must be used in the advice.
S_F	A set $S_F \subseteq [0, m_A) \times [0, n) \times [0, m_F) \times [0, n)$ indicating which fixed entries must be used in the advice.
CUS_u	Sets $\mathsf{CUS}_u \subseteq [0, n)$ indicating which rows the custom functions $p_u : \mathbb{F}^{m_F + m_A} \mapsto \mathbb{F}$ are applied to.
p_u	Custom multivariate polynomials $p_u : \mathbb{F}^{m_F + m_A} \mapsto \mathbb{F}$
LOOK_v	Sets $\mathsf{LOOK}_v \subseteq [0, n)$ indicating which advice rows are contained in the lookup tables TAB_v.
TAB_v	Lookup tables $\mathsf{TAB}_v \subseteq \mathbb{F}^{m_F}$ with an unbounded number of entries in the field $\mathbb{F}^m$
q_v	Custom scaling functions $q_v : \mathbb{F}^{m_A} \mapsto \mathbb{F}^{m_{q_v}}$ where $q_v \leq m_A$

> TODO: do we need to generalise lookup tables to support dynamic tables (in advice columns)? Probably too early, but we could think about it.

Private Inputs	Description
w	Advice columns $w : \mathbb{F}^{m_A \times n}$.

Draft specification precise about all inputs.

Popular Constraint Systems

R1CS
(Rank 1 Constraint System)

Plonkish

AIR
(Algebraic Intermediate Representation)

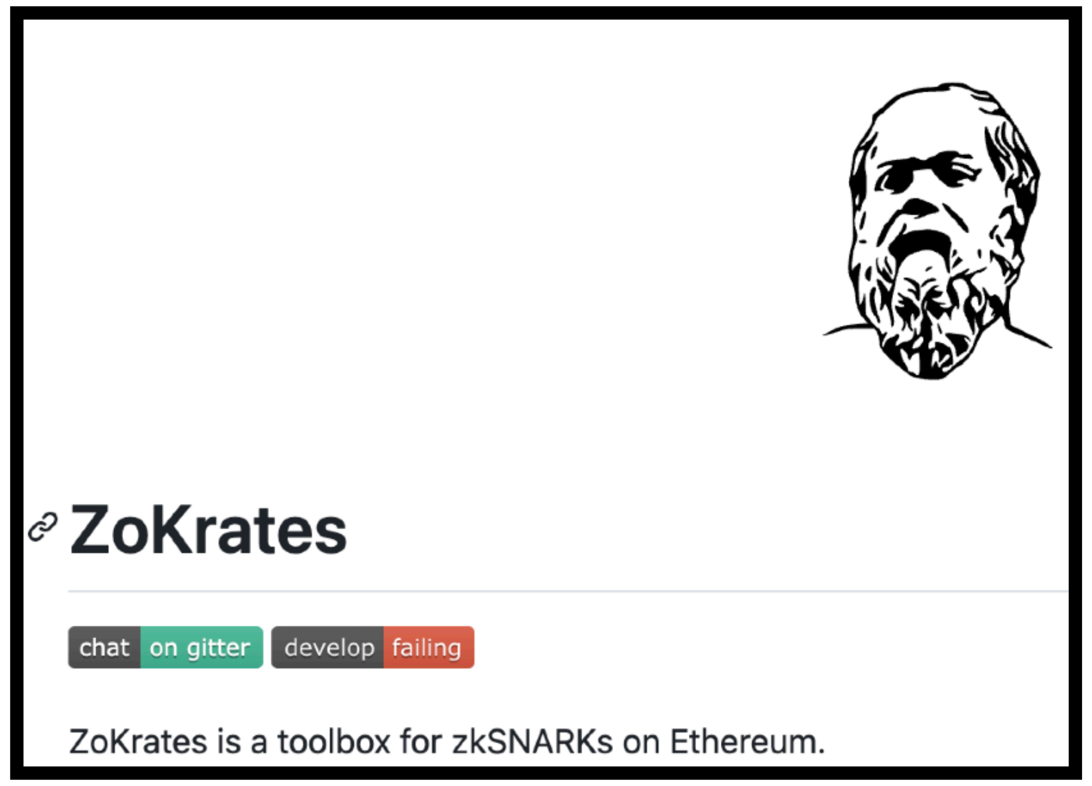
- I am currently prioritising Plonkish.
- There have been prior attempts to write specifications for R1CS but they are not currently active.
- Specifications for R1CS and AIR would be nice to have.

CSS
Customisable Constraint System

Recent

Constraints for Key Applications

- Concrete applications: Require concrete constraints. I cannot prove preimage of a hash with just a constraint system. I need the constraints.
- Fully general is too hard: I do not know how to write specifications for Circom, zkVMs, or other constraint writers at this point in time.

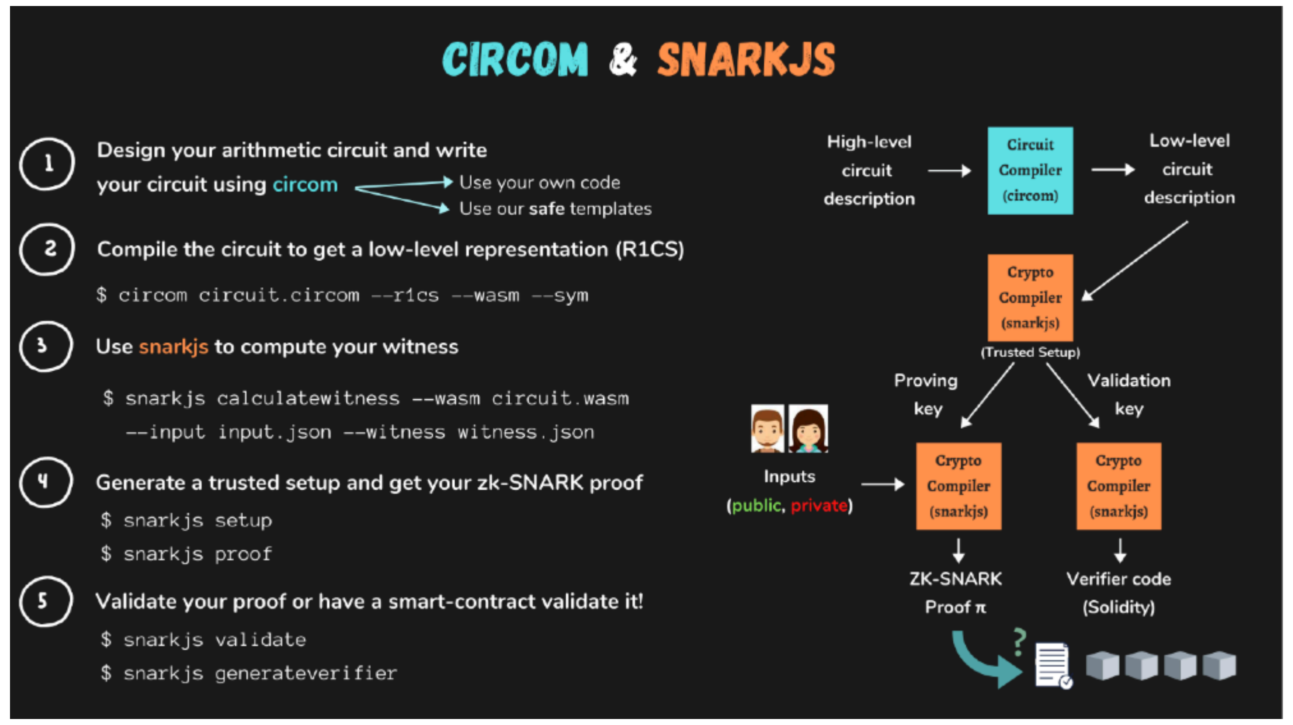


ZoKrates

ZoKrates is a toolbox for zkSNARKs on Ethereum.

Version: 0.7.1

Introducing Noir



CIRCOM & SNARKJS

1. Design your arithmetic circuit and write your circuit using circom → Use your own code / Use our safe templates
2. Compile the circuit to get a low-level representation (R1CS)

```
$ circom circuit.circom --r1cs --wasm --sym
```

3. Use snarkjs to compute your witness

```
$ snarkjs calculatewitness --wasm circuit.wasm
  --input input.json --witness witness.json
```

4. Generate a trusted setup and get your zk-SNARK proof

```
$ snarkjs setup
$ snarkjs proof
```

5. Validate your proof or have a smart-contract validate it!

```
$ snarkjs validate
$ snarkjs generateverifier
```

High-level circuit description
Circuit Compiler (circom)
Low-level circuit description
Crypto Compiler (snarkjs)
(Trusted Setup)
Proving key
Validation key
Inputs (public, private)
Crypto Compiler (snarkjs)
Crypto Compiler (snarkjs)
ZK-SNARK Proof π
Verifier code (Solidity)
?

So what's the plan?

Constraints for Key Applications

- Specific constraints doable: We could, theoretically, have a concrete list of “approved” applications that have “approved constraints.”
- Which applications to target: There are many choices. But. There is also a very juicy NIST target.

NIST threshold draft call

So what's the plan?

Constraints for Key Applications

- NIST Draft Threshold Call: Document requesting for threshold schemes for encryption, signatures and more.
- That doesn't sound ZK: They have explicitly expressed interest in zero-knowledge proofs of knowledge of secret key for a selection of schemes.
- So?: Please could we have certified constraints for the relations in Table 12 (Page 53)

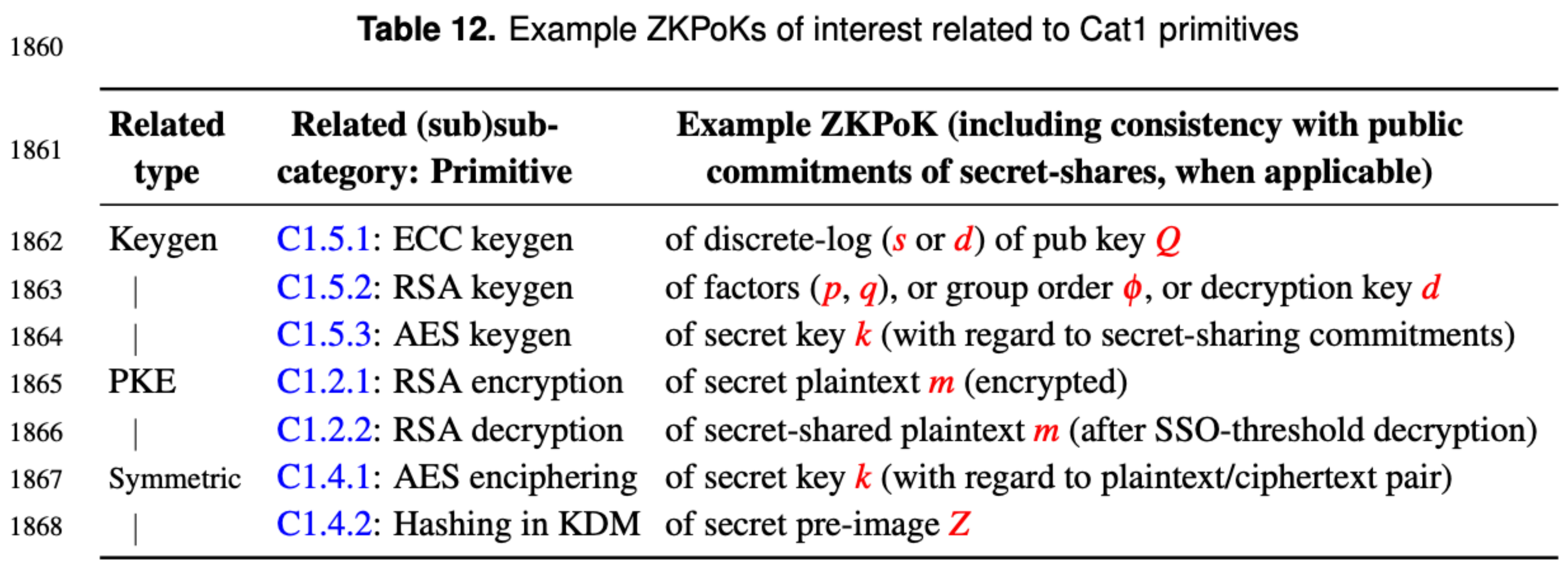


Table 12. Example ZKPoKs of interest related to Cat1 primitives

Related type	Related (sub)sub-category: Primitive	Example ZKPoK (including consistency with public commitments of secret-shares, when applicable)
Keygen	C1.5.1: ECC keygen	of discrete-log (s or d) of pub key Q
|	C1.5.2: RSA keygen	of factors (p, q), or group order ϕ, or decryption key d
|	C1.5.3: AES keygen	of secret key k (with regard to secret-sharing commitments)
PKE	C1.2.1: RSA encryption	of secret plaintext m (encrypted)
|	C1.2.2: RSA decryption	of secret-shared plaintext m (after SSO-threshold decryption)
Symmetric	C1.4.1: AES enciphering	of secret key k (with regard to plaintext/ciphertext pair)
|	C1.4.2: Hashing in KDM	of secret pre-image Z

NIST threshold draft call

Aims of the Constraint System

- Simplicity: A straight forward design for ease of understanding.
- Comprehensive: A clear set of rules that define the relation.
- Compatibility: Constraint system should not to be too “special”.
-

The Optimising System

- Simplicity: A straight forward design for ease of understanding.
- Comprehensive: A clear set of rules that define the relation.
- Compatibility: Constraint system should not to be too “special”.
- Applicability: Must be optimal enough that people actual use it…

The optimising system should contain all the optimisations that are too complicated for the main constraint system.

The Plonkish Optimising System

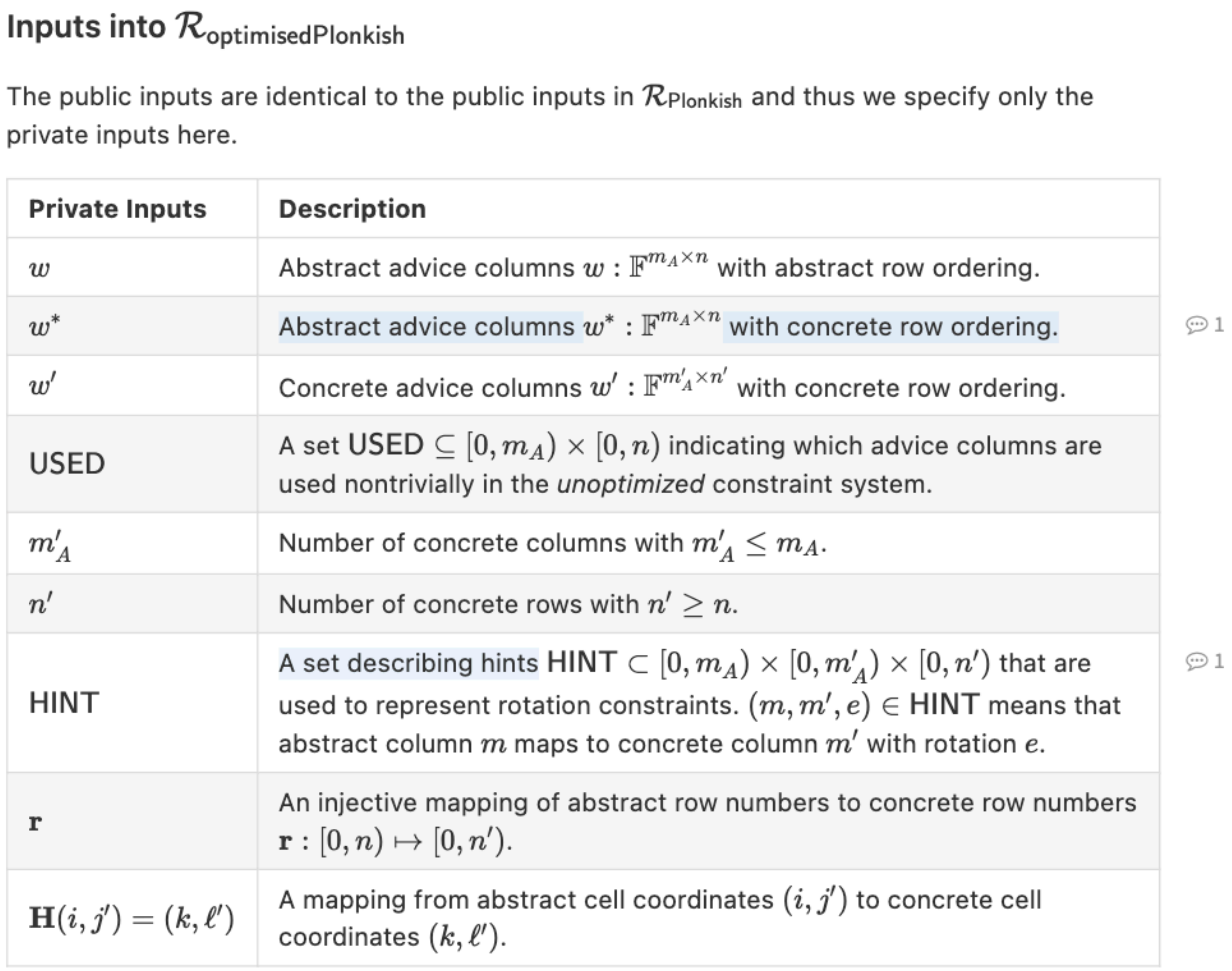
- Rotations/ Shifts: Rotations can be a useful alternative to copy constraints.
- Reorgs: Key optimisation reorders the table rows to maximise the number of rotations.
- Permutations: Developers like to have explicit permutation constraints.

The optimising system should not change the meaning expressed in the constraint system.

The Plonkish Optimising System

- Working group: The working group on the Plonkish constraint system are simultaneously tackling this step.
- It's hard: We expect, once we have the specification written, to go through many steps of community feedback.
- Simplicity still matters: I want to prioritise only important optimisations.

The Plonkish Optimising System

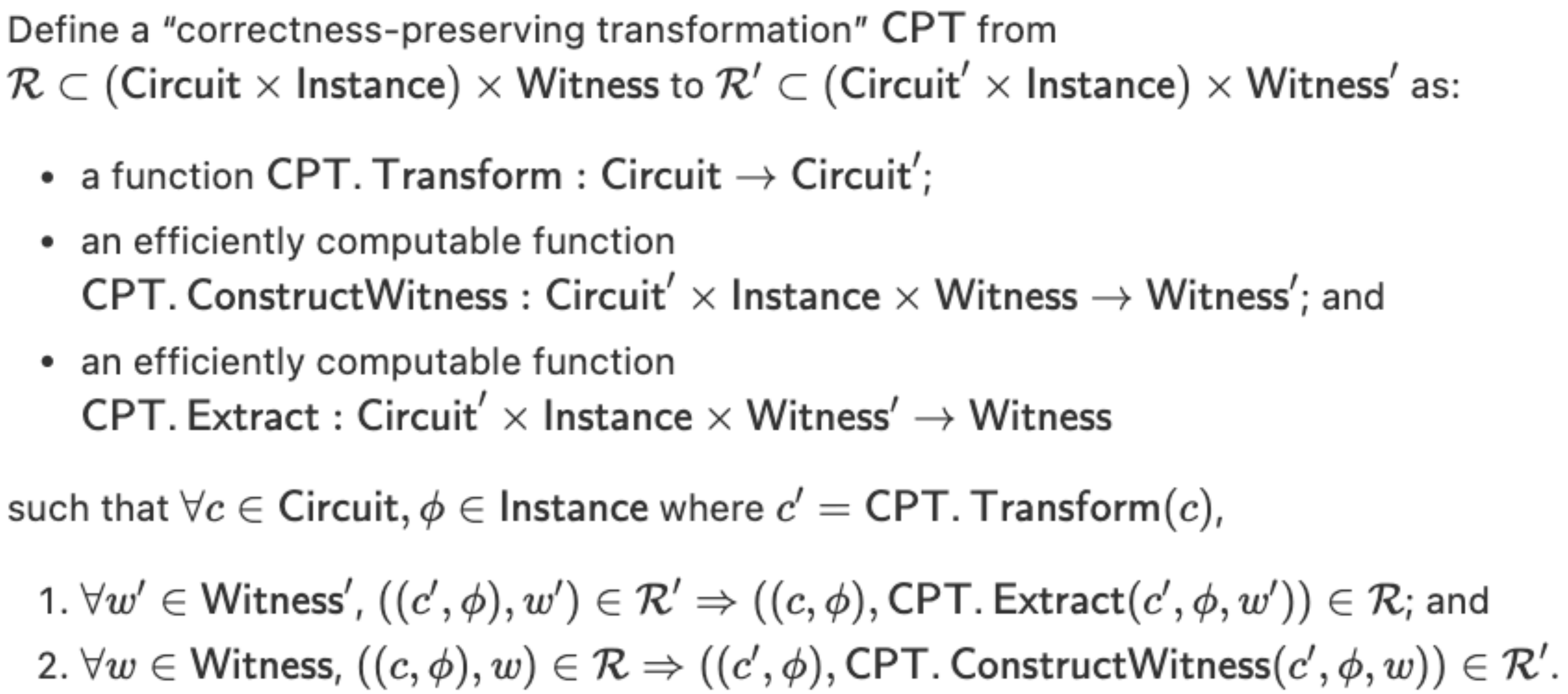


Inputs into $\mathcal{R}_{\text{optimisedPlonkish}}$

The public inputs are identical to the public inputs in $\mathcal{R}_{\text{Plonkish}}$ and thus we specify only the private inputs here.

Private Inputs	Description
w	Abstract advice columns $w : \mathbb{F}^{m_A \times n}$ with abstract row ordering.
w^*	Abstract advice columns $w^* : \mathbb{F}^{m_A \times n}$ with concrete row ordering.
w'	Concrete advice columns $w' : \mathbb{F}^{m'_A \times n'}$ with concrete row ordering.
USED	A set $\text{USED} \subseteq [0, m_A) \times [0, n)$ indicating which advice columns are used nontrivially in the *unoptimized* constraint system.
m'_A	Number of concrete columns with $m'_A \leq m_A$.
n'	Number of concrete rows with $n' \geq n$.
HINT	A set describing hints $\text{HINT} \subset [0, m_A) \times [0, m'_A) \times [0, n')$ that are used to represent rotation constraints. $(m, m', e) \in \text{HINT}$ means that abstract column m maps to concrete column m' with rotation e.
$\mathbf{r}$	An injective mapping of abstract row numbers to concrete row numbers $\mathbf{r} : [0, n) \mapsto [0, n')$.
$\mathbf{H}(i, j') = (k, \ell')$	A mapping from abstract cell coordinates (i, j') to concrete cell coordinates (k, ℓ').

The Plonkish Optimising System



Define a "correctness-preserving transformation" CPT from $\mathcal{R} \subset (\mathsf{Circuit} \times \mathsf{Instance}) \times \mathsf{Witness}$ to $\mathcal{R}' \subset (\mathsf{Circuit}' \times \mathsf{Instance}) \times \mathsf{Witness}'$ as:

- a function $\mathsf{CPT.Transform} : \mathsf{Circuit} \to \mathsf{Circuit}'$;
- an efficiently computable function $\mathsf{CPT.ConstructWitness} : \mathsf{Circuit}' \times \mathsf{Instance} \times \mathsf{Witness} \to \mathsf{Witness}'$; and
- an efficiently computable function $\mathsf{CPT.Extract} : \mathsf{Circuit}' \times \mathsf{Instance} \times \mathsf{Witness}' \to \mathsf{Witness}$

such that $\forall c \in \mathsf{Circuit}, \phi \in \mathsf{Instance}$ where $c' = \mathsf{CPT.Transform}(c)$,

1. $\forall w' \in \mathsf{Witness}'$, $((c', \phi), w') \in \mathcal{R}' \Rightarrow ((c, \phi), \mathsf{CPT.Extract}(c', \phi, w')) \in \mathcal{R}$; and
2. $\forall w \in \mathsf{Witness}$, $((c, \phi), w) \in \mathcal{R} \Rightarrow ((c', \phi), \mathsf{CPT.ConstructWitness}(c', \phi, w)) \in \mathcal{R}'$.

Correctness-preserving transform that maintains knowledge-soundness, correctness, and zero-knowledge.

Summary of the Plonkish Working Group

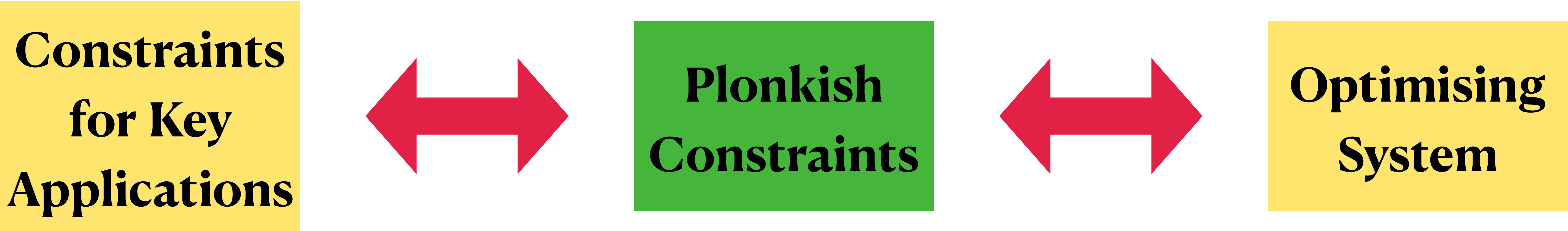
Jack Grigg (ECC)

Daira Emma Hopwood (ECC)

Me

- Write specification for Plonkish constraint system.
- Write specification for optimising system.
- Theoretically, our next step is to implement Plonkish constraint system in Hacspec.
  Please don't hold me to this. I do not know Hacspec, I just heard a really good pitch for it.

Working Groups that must Communicate

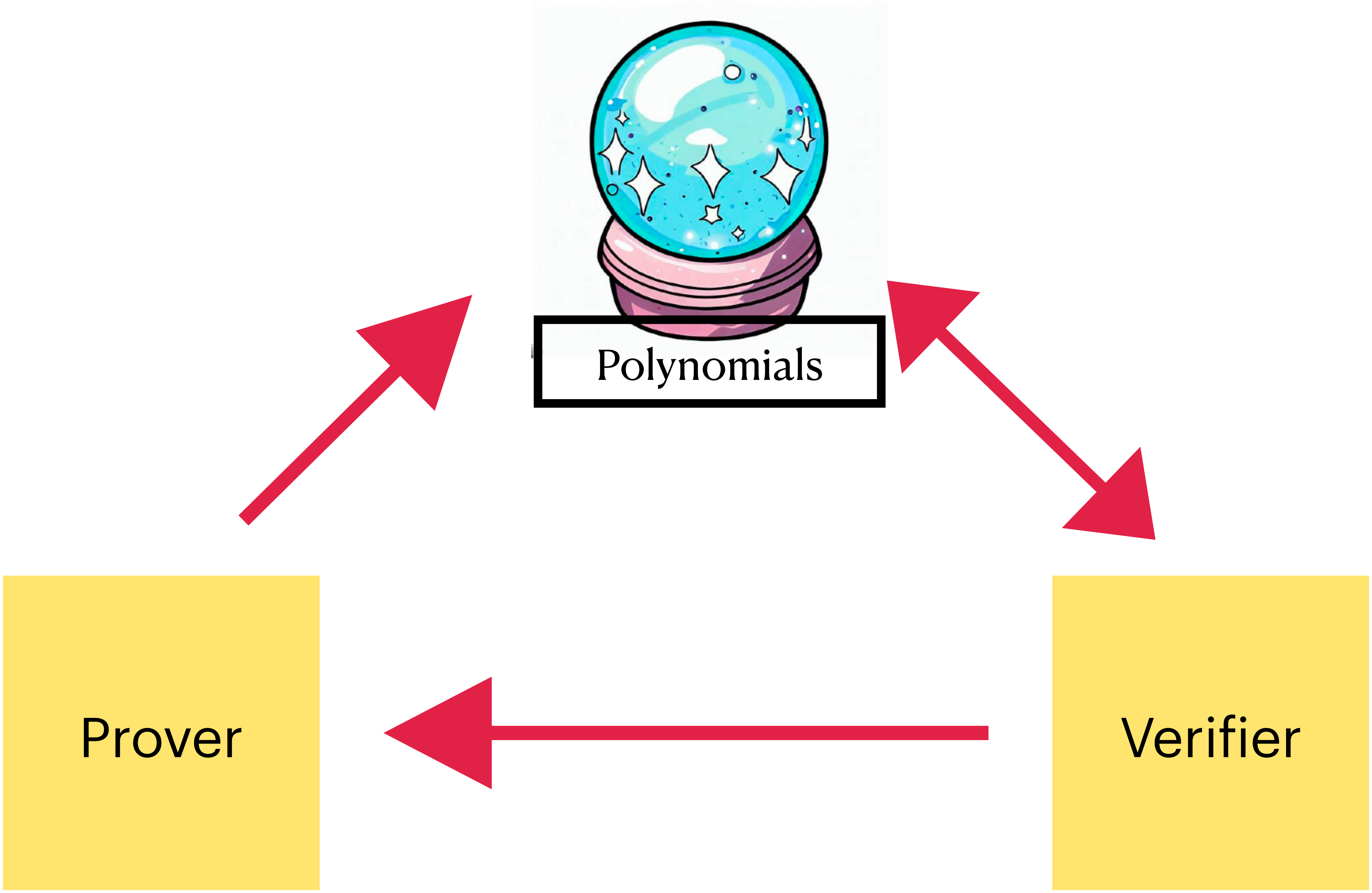


Constraints for Key Applications

Plonkish Constraints

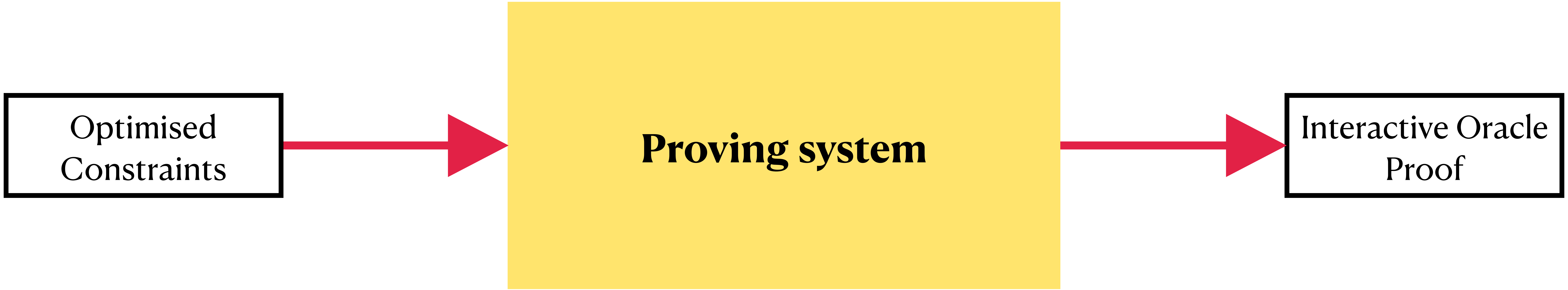
Optimising System

The Interactive Oracle Proof



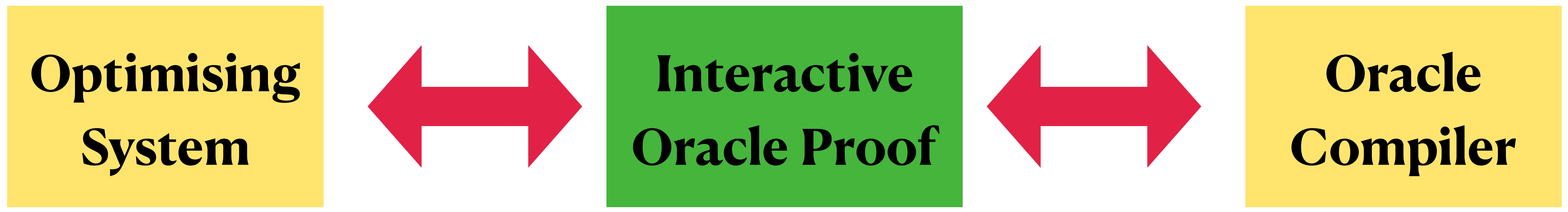
Interactive oracle proofs are the information theoretical part of a ZKP.
There is no cryptography at this step.

The Interactive Oracle Proof



- Working group needed
- Interoperability: The inputs must match the format of the optimised constraint system. The outputs match the format of the "oracle compiler."
- Further modularity: Maybe include building blocks e.g. lookup IOPs as optional black box.

Working Groups that must Communicate

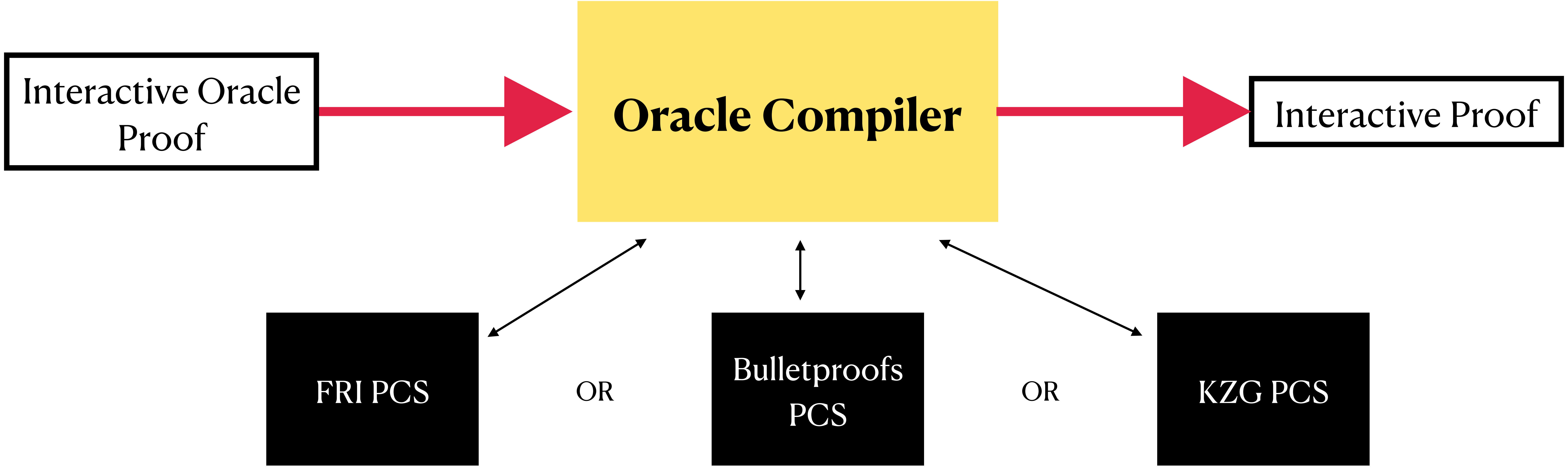


Optimising System

Interactive Oracle Proof

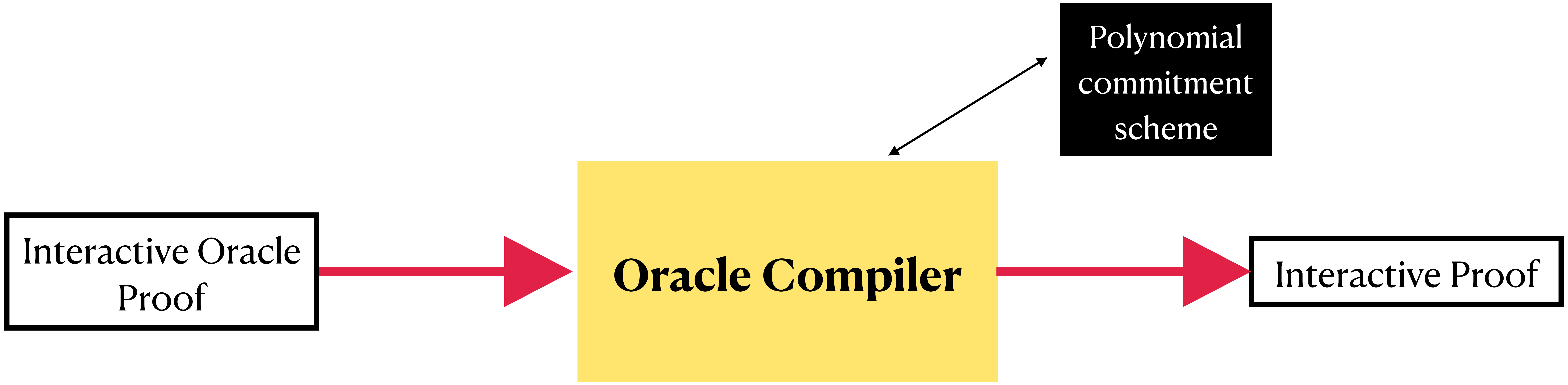
Oracle Compiler

The Oracle Compiler



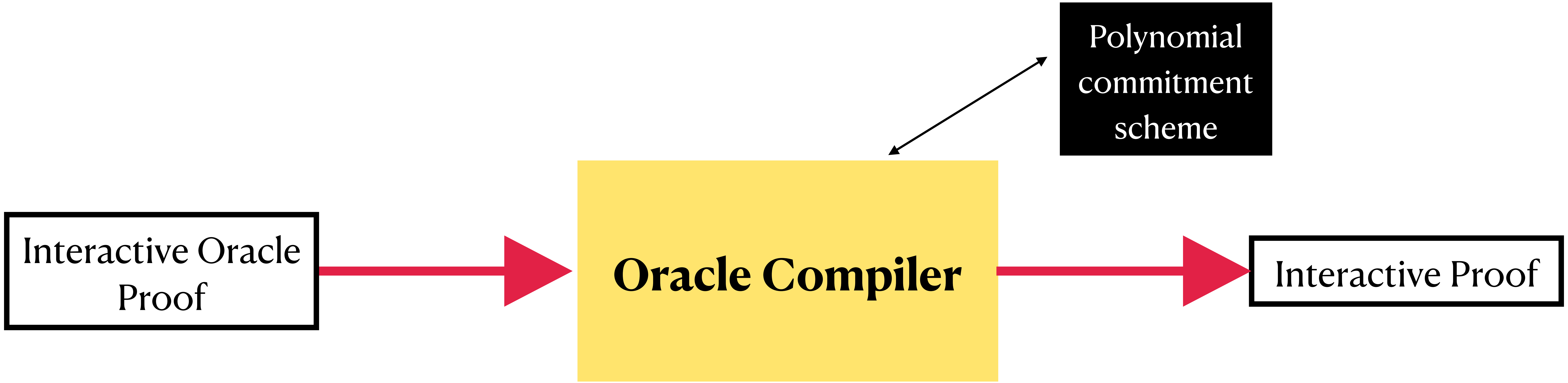
We have choices for our PCS that determine security assumptions, trusted setup, proof size, and verifier time. We want to be able to have a unifying framework for all of them.

The Oracle Compiler



- Working group needed
- APIs: This group needs to define the APIs and security requirements for a polynomial commitment scheme.
- Abstraction: Need that the field size is determined by the polynomial commitment scheme. Must support both large and small fields.
- Interoperability: Inputs must be consistent with proving system. Outputs formats must be consistent with Fiat-Shamir compiler.

The Oracle Compiler



Polynomial commitment scheme

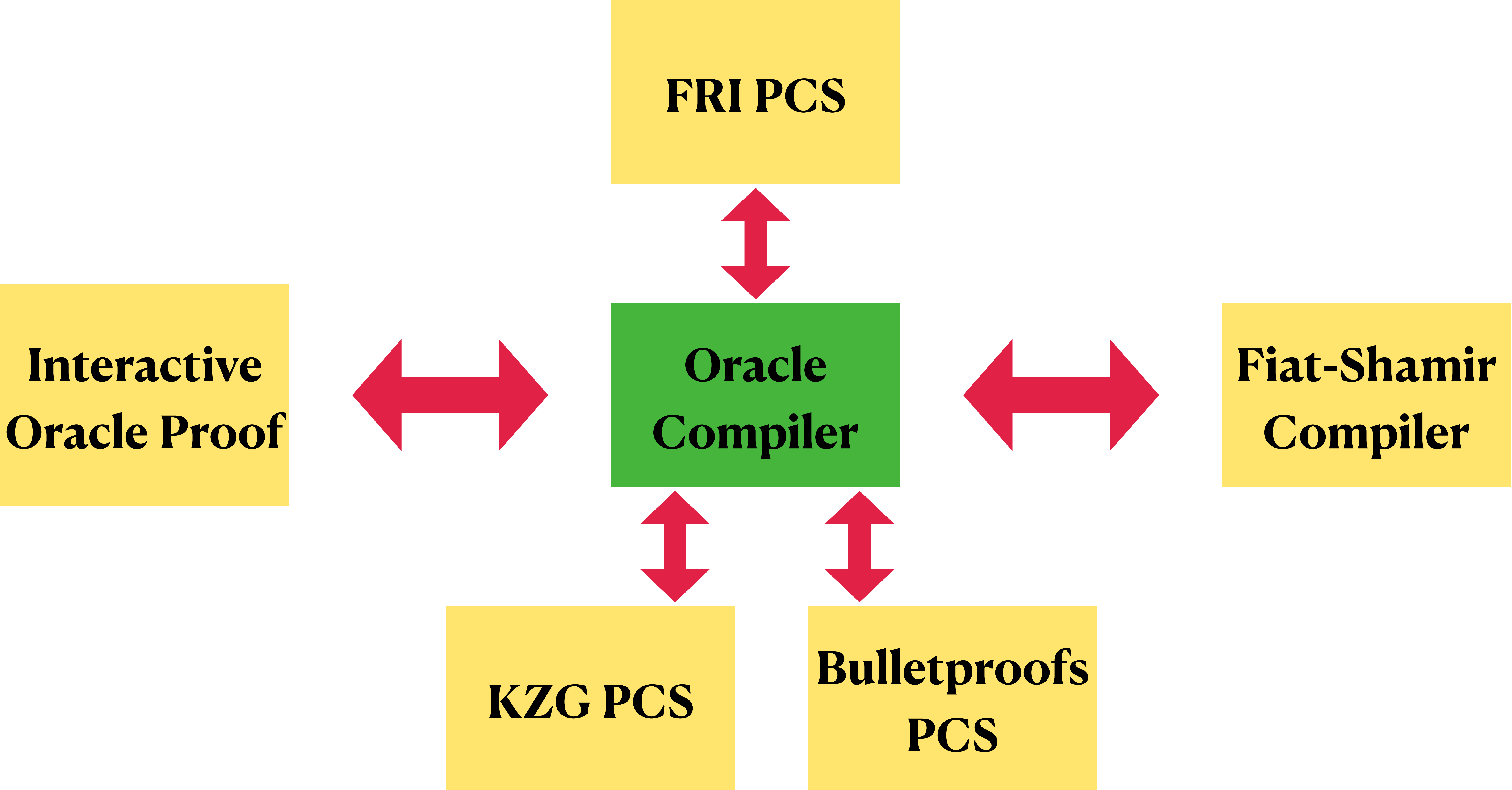
Interactive Oracle Proof

Oracle Compiler

Interactive Proof

Outputs here could be independent from the constraint system? :)

Working Groups that must Communicate



FRI PCS

Interactive Oracle Proof

Oracle Compiler

Fiat-Shamir Compiler

KZG PCS

Bulletproofs PCS

Polynomial Commitment Schemes (PCS)

FRI PCS

Bulletproofs PCS

KZG PCS

- Working groups needed
- APIs: Must use APIs that are consistent with the oracle compiler.
- Random Oracles: Specify usage of hash functions (e.g. by reference to Sigma Protocols spec)

Polynomial Commitment Schemes (PCS)

Bulletproofs
PCS

- Working group needed: Simplest PCS. Easy target for specs. Maybe some already exist?
- Modularity: PCS specification for any group where DL holds with any “Random Oracle” hash.
- Groups: Recommend at least one concrete group.

Polynomial Commitment Schemes (PCS)

FRI PCS

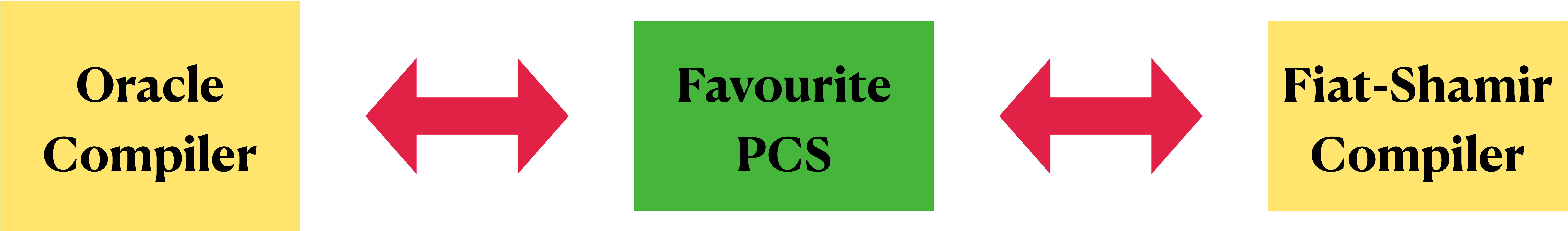
- Working group needed: FRIs are leading player for PCS and we really want this.
- Security: Target at least 128 bits of security in the random oracle model **(grinding attacks)**.
- Challenge: Are we okay using **algebraic hash functions**? Or only SHA2/ SHA3 for now?

Polynomial Commitment Schemes (PCS)

KZG PCS

- Working group needed: Anca Nitulescu (Protocol Labs) is interested in leading this working group.
- Modularity: PCS specification for any bilinear group with any “Random Oracle” hash.
- Pairings: **Cannot abstract.** Working group requires one person who knows how pairings work.

Working Groups that must Communicate

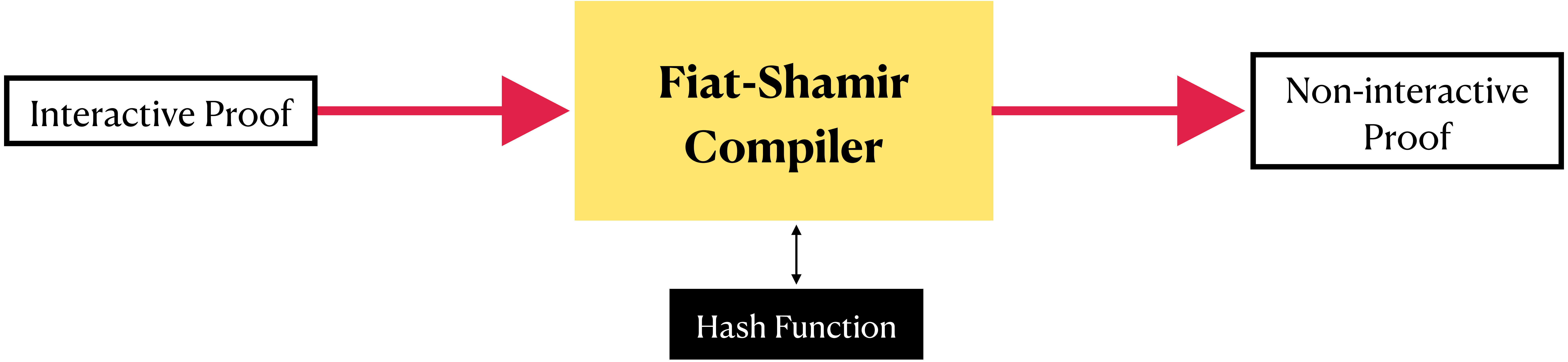


Oracle Compiler

Favourite PCS

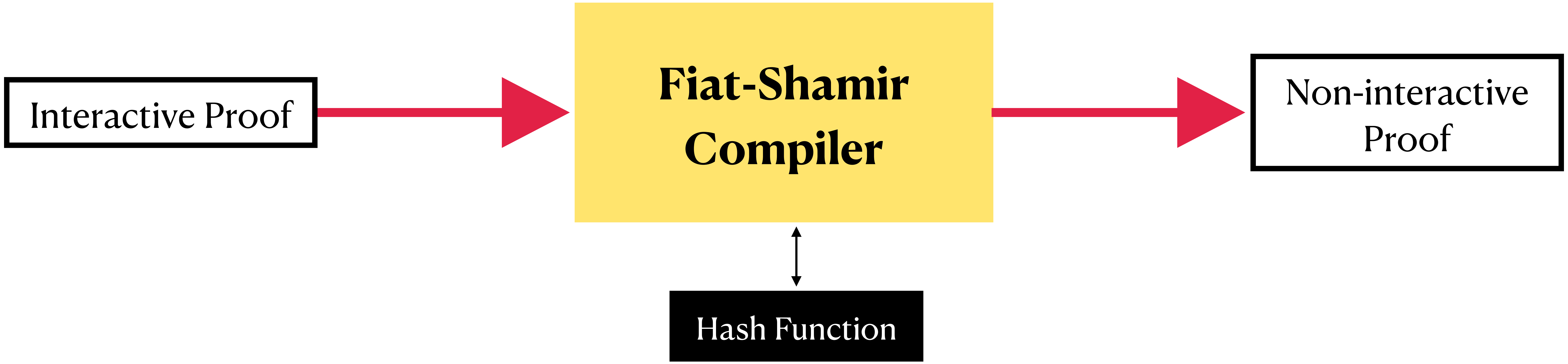
Fiat-Shamir Compiler

Fiat-Shamir Compiler



- Sigma protocols working group: Led by Michele Orru.
- Interoperability: Inputs must be consistent with oracle compiler. Output is the final proof.

Fiat-Shamir Compiler



- Considerations:
  - Domain Separation: Need a clear format.
  - Padding issues: Sometimes padding is required for uniqueness of inputs.
  - Number of outputs: Good and bad ways of squeezing multiple field elements from same input.

Sigma specification

Companies we want Involved

- zkSync
- Protocol Labs
- Web3 Foundation
- Aleo
- Anoma
- Risc Zero
- Aztec
- Manta Network
- Matterlabs
- Ethereum PSE team
- Apple
- Lurk Labs
- Espresso Systems
- Scroll
- ZK Validator
- Microsoft
- 0xParc
- Zcash
- Polygon
- Qedit
- Nethermind
- Polkadot
- Least Authority
- O(1) Labs
- Dusk Network
- Starkware
- Cloudflare
- Others?

Aim Now

Please split into groups to discuss e.g. the following topics.

Constraints for Key Applications

Interactive Oracle Proofs

Oracle Compiler

Bulletproofs PCS

KZG PCS

FRI PCS

- Who: Who are the ideal participants and how do we contact them (are they in the room?).
- What: What is the ideal output? Is there something I haven't thought of?
- When: What is the timeline? Do other working groups need to produce first?
- How: How do we make it happen?