Private outsourcing of zkSNARK proof construction

Mariana Gama

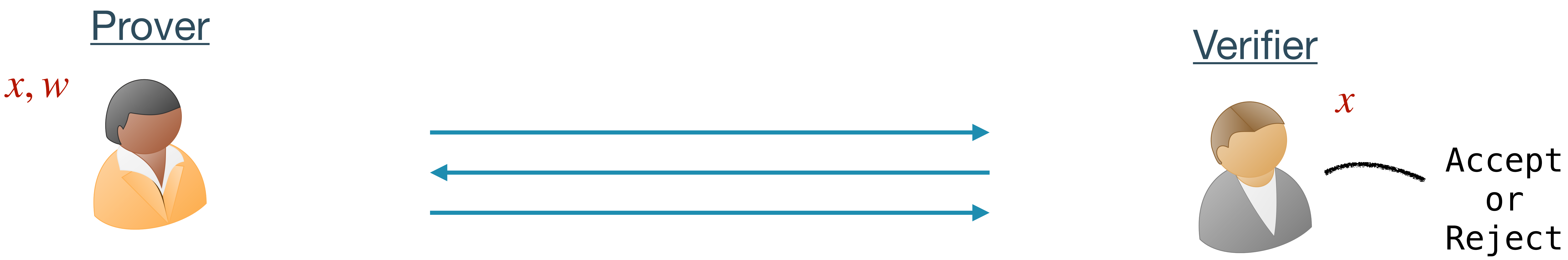
13 December 2024

KU LEUVEN

Zero-knowledge proofs (of knowledge)

- Relation R
- x is public (statement)
- w is private (witness)

Prover claims: I know w such that $(x, w) \in R$



Zero-knowledge proofs (of knowledge)

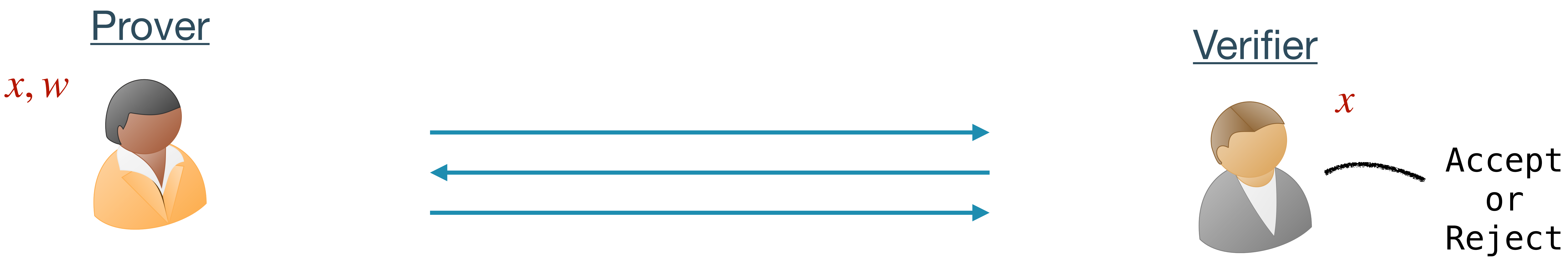
- Relation R
- x is public (statement)
- w is private (witness)

Prover claims: I know w such that $(x, w) \in R$

Completeness: honest V accepts proof from honest P

(Knowledge) Soundness: If P doesn't know w, V rejects

Zero-Knowledge: does not leak anything about w



zkSNARKs
<u>S</u>uccint <u>N</u>on-interactive <u>AR</u>gument of <u>K</u>nowledge

- Relation R
- x is public (statement)
- w is private (witness)

<u>Prover claims</u>: I know w such that $(x, w) \in R$

<u>Completeness</u>: honest V accepts proof from honest P

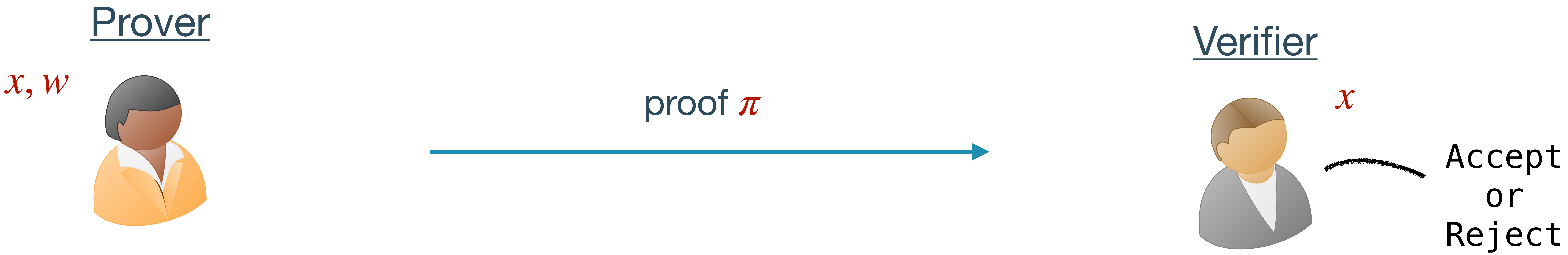
<u>(Knowledge) Soundness</u>: If P doesn't know w, V rejects

<u>Zero-Knowledge</u>: does not leak anything about w

<u>Non-interactive</u>: no exchange between prover and verifier

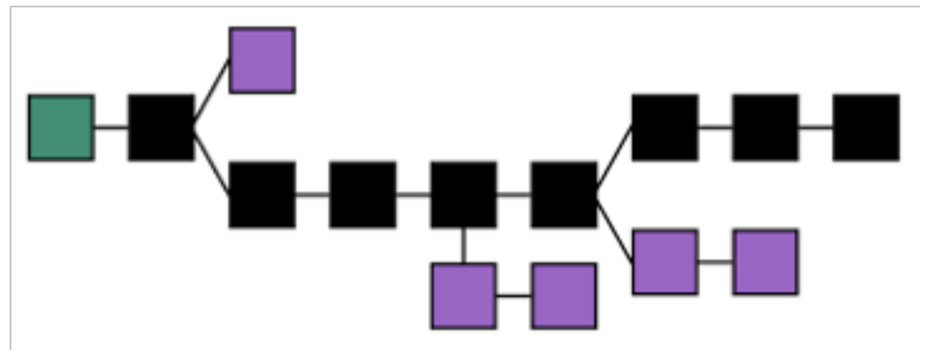
<u>Succinct</u>: - proof size independent (sublinear) of witness size

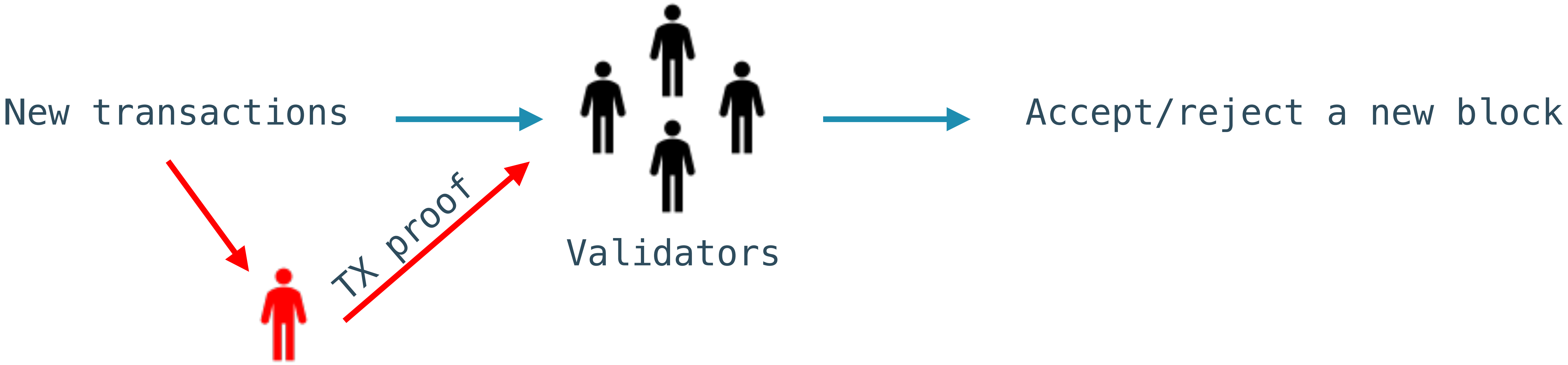
- fast verification



(zk)SNARKs: where are they used?

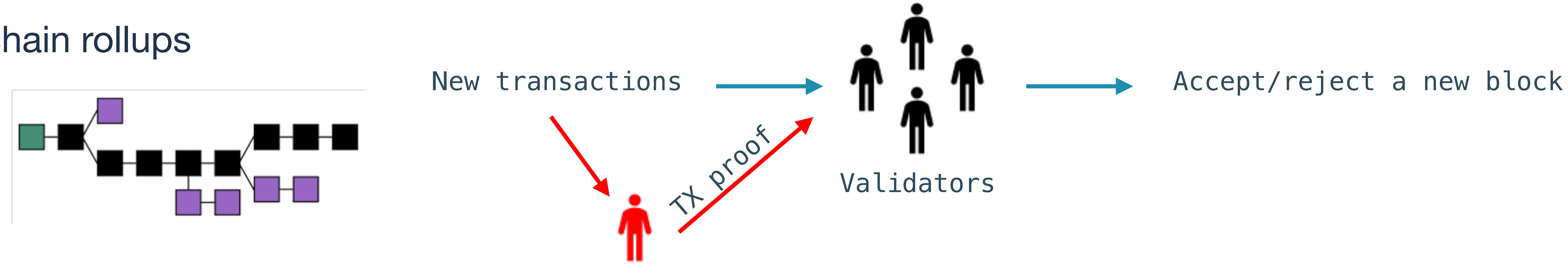
- Blockchain rollups





(zk)SNARKs: where are they used?

- Blockchain rollups

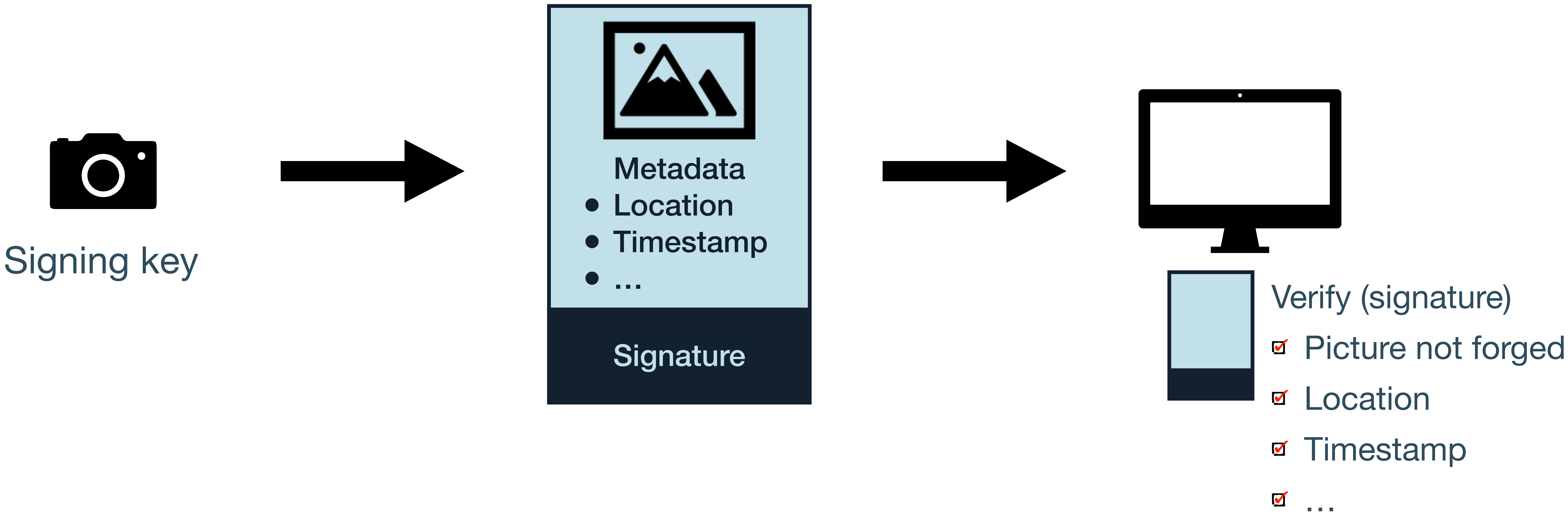


- zkML: proof of correct training / correct inference

- Sensors telemetry data

- Journalism (content provenance)

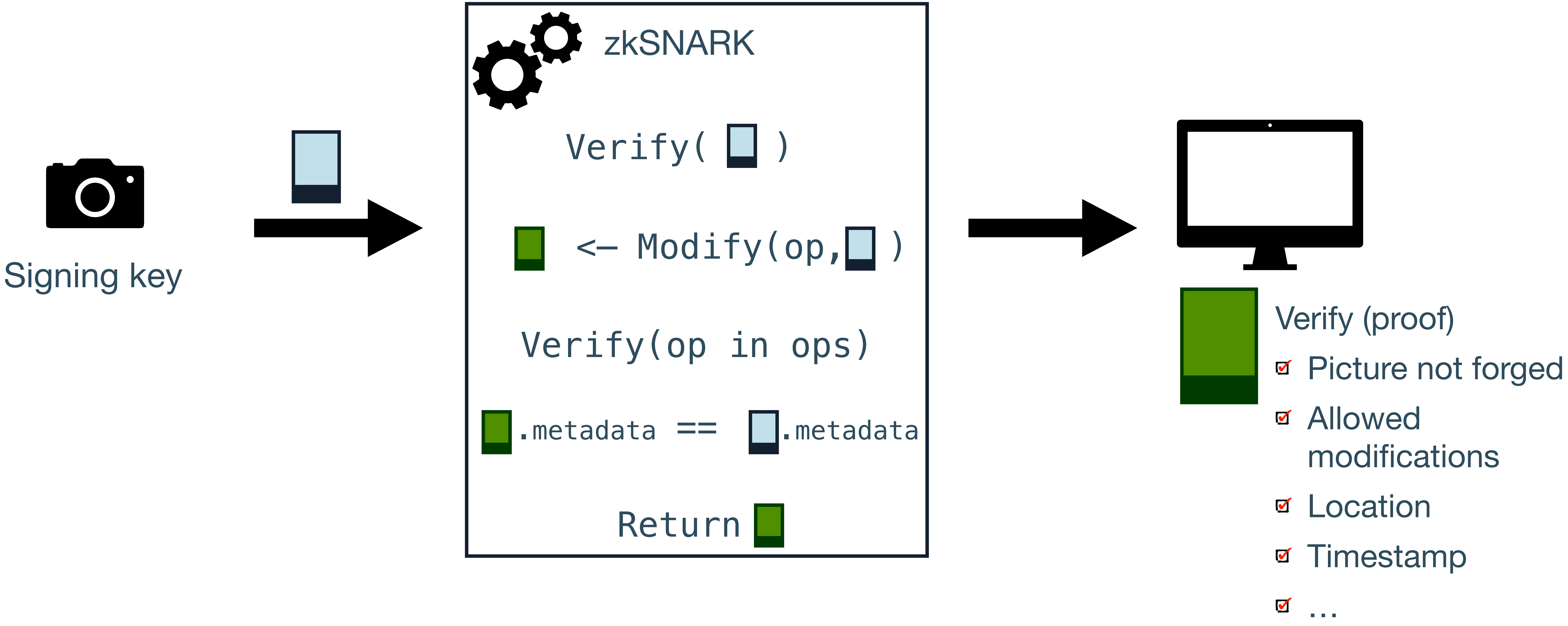
(zk)SNARKs: where are they used?
image attestation



- Any modifications: signature verification fails

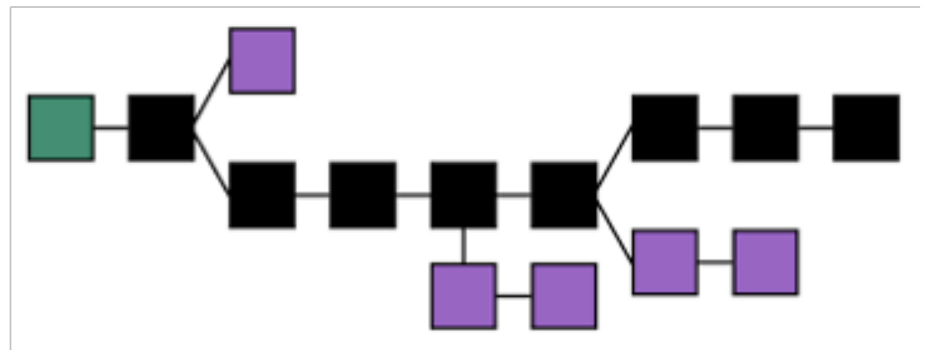
(zk)SNARKs: where are they used?
image attestation

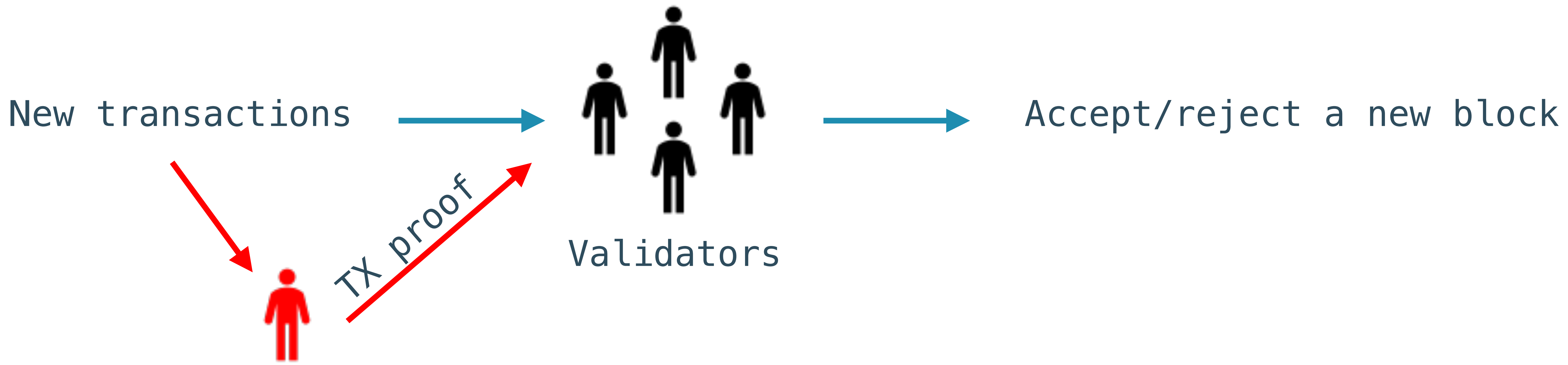
- zk-IMG **[KHSS22]**
- VerITAS **[DCB24]**



(zk)SNARKs: where are they used?

- Blockchain rollups





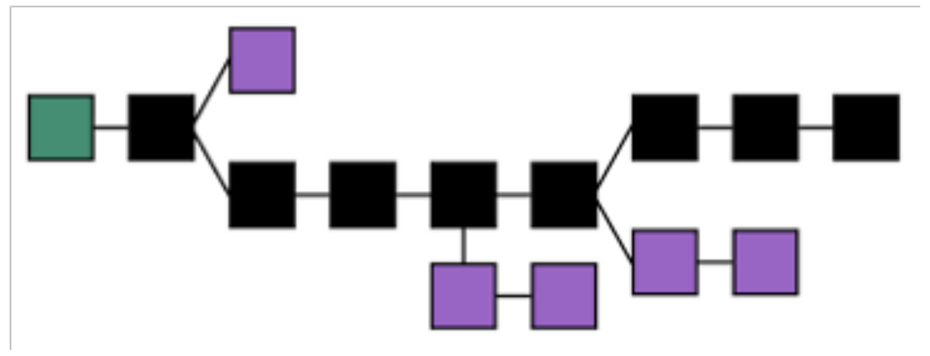
- zkML: proof of correct training / correct inference

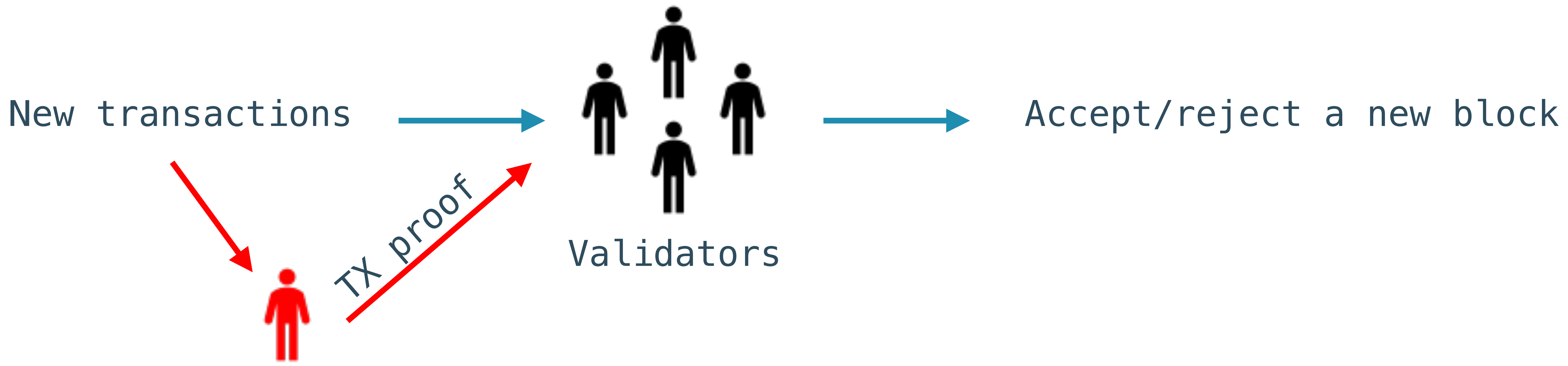
- Sensors telemetry data

- Journalism (content provenance)

(zk)SNARKs: where are they used?

- Blockchain rollups





- zkML: proof of correct training / correct inference
- Sensors telemetry data
- Journalism (content provenance)

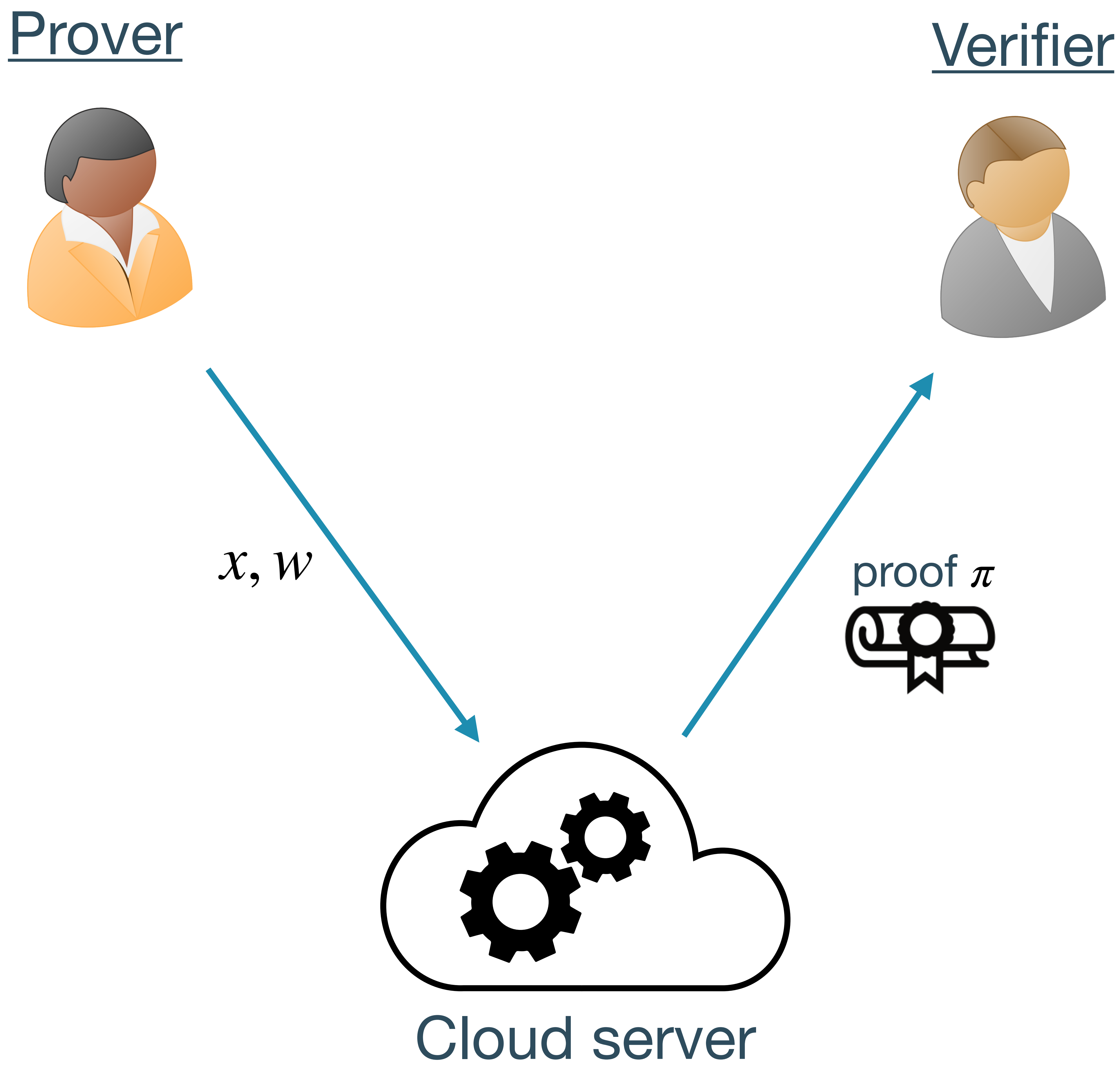
System Requirements

- zkProver: 1TB RAM with 128-core CPU

If you want to run a full-fledged zkProver on your own, you'll need at least 1TB of RAM.

polygon zkEVM

Computing zkSNARKs is expensive
… can we outsource it?

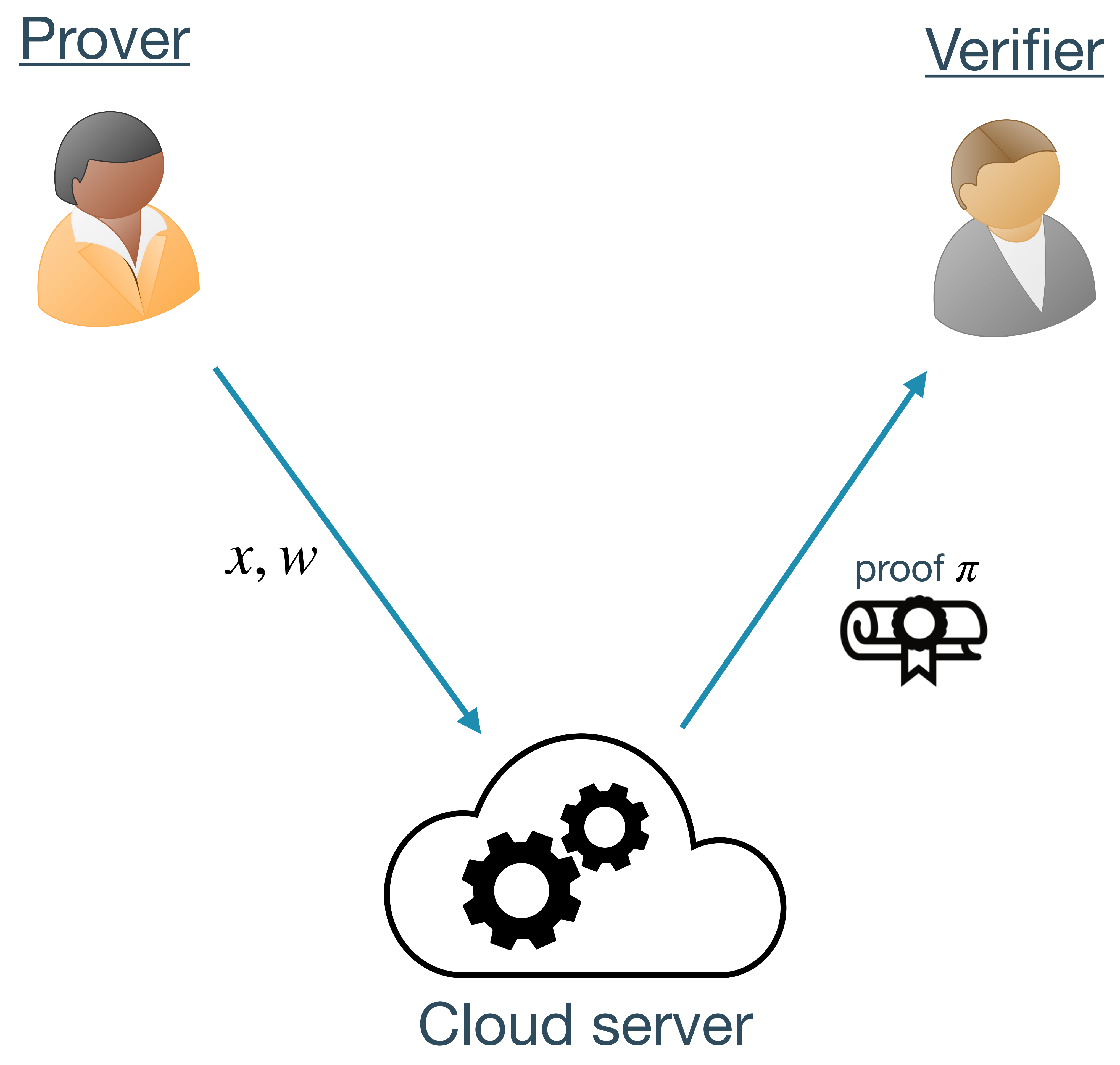


Computing zkSNARKs is expensive
… can we outsource it?

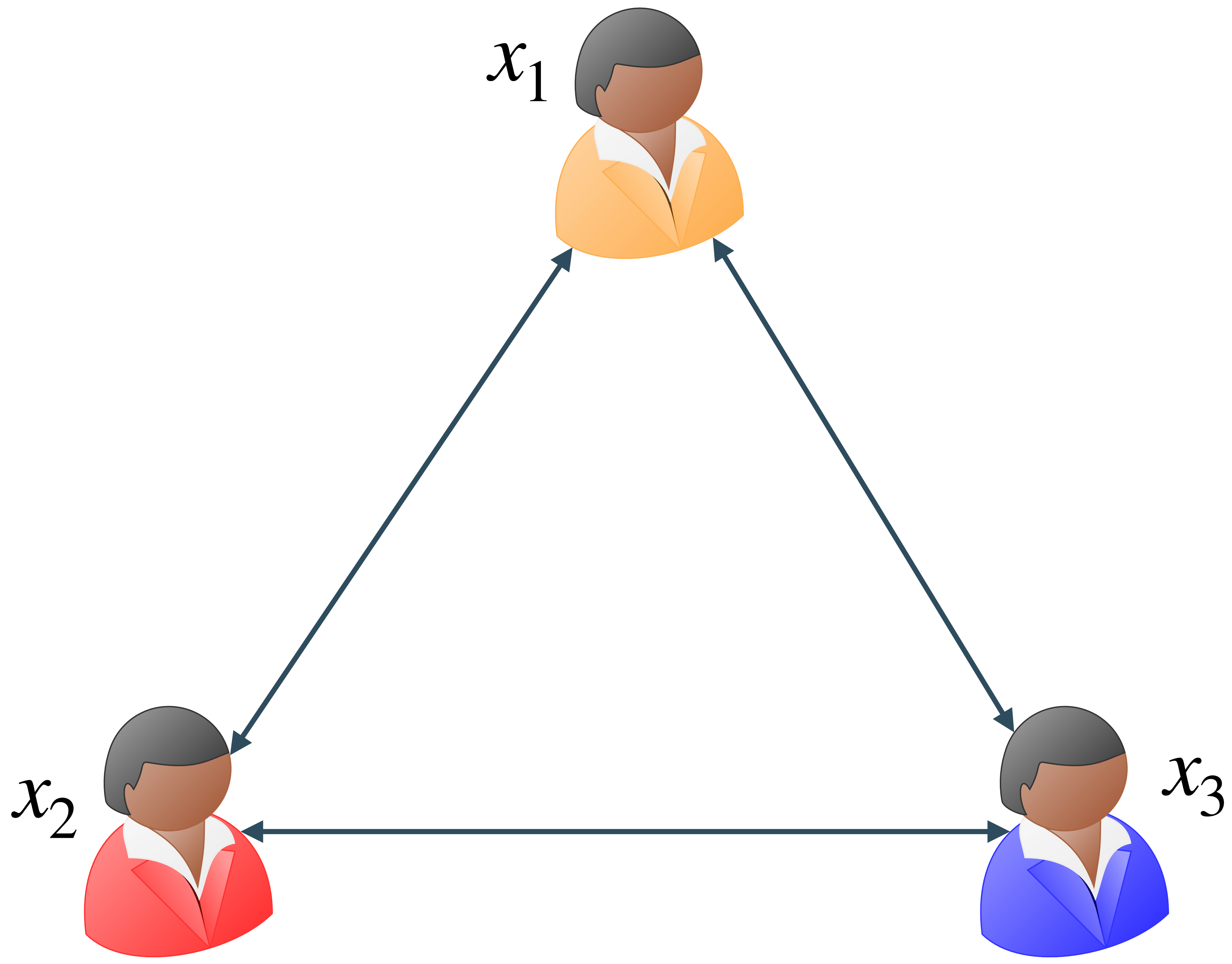
Horizontally scalable zkSNARKs:

- DIZK **[WZCPS18]**
- Pianist **[LXZSZ24]**
- Hekaton **[RMHMM24]**
- …

More worker nodes —> less work per node
(but nodes learn the witness)



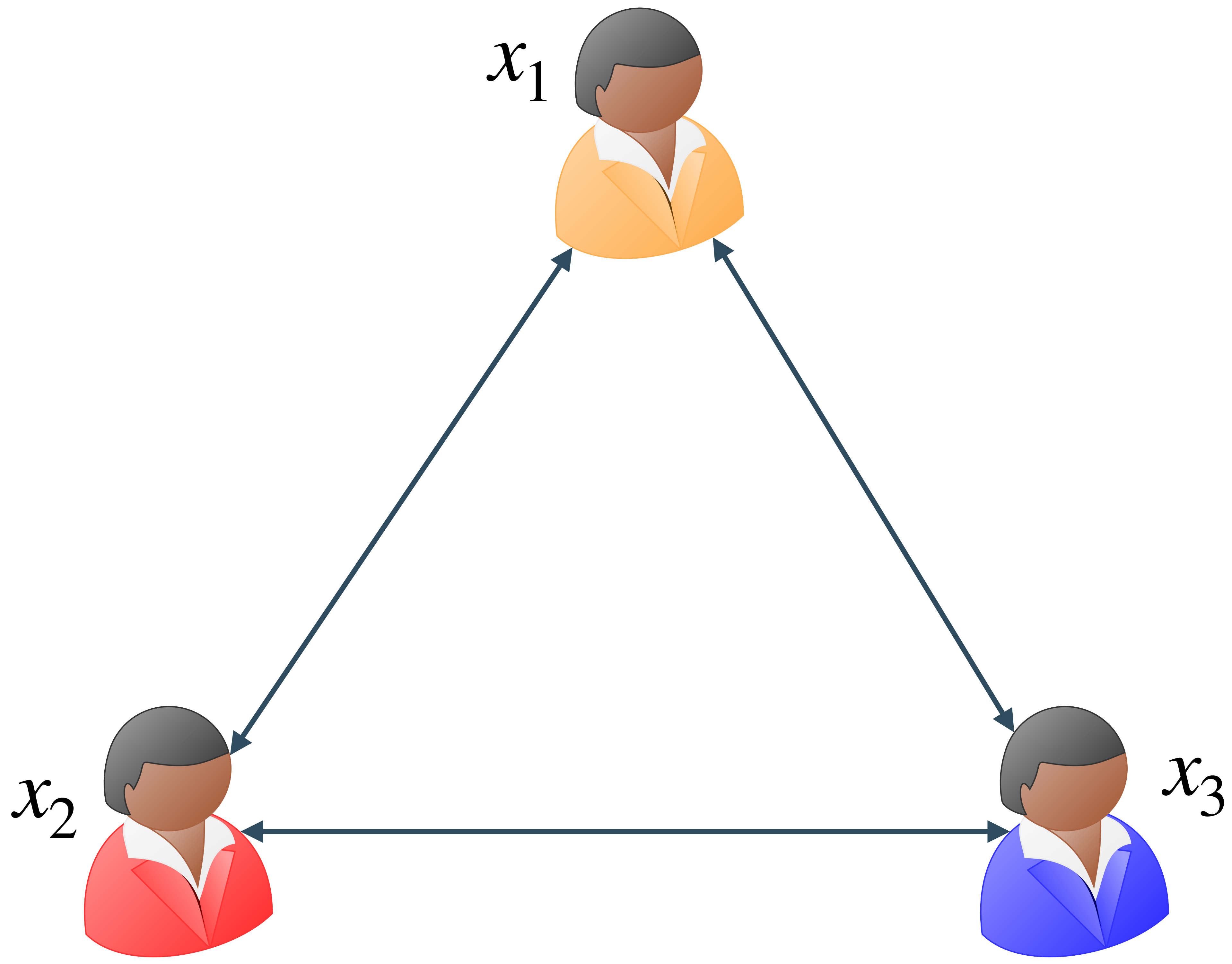
Proof outsourcing with Multiparty Computation (MPC)



- Technique for computing over encrypted data.
- Achieves privacy by distributing the computation.

> Adversary corrupting a percentage of the parties will still learn nothing but the output,
>
> $$y = f(x_1, x_2, x_3)$$

Proof outsourcing with Multiparty Computation (MPC)



- Technique for computing over encrypted data.
- Achieves privacy by distributing the computation.

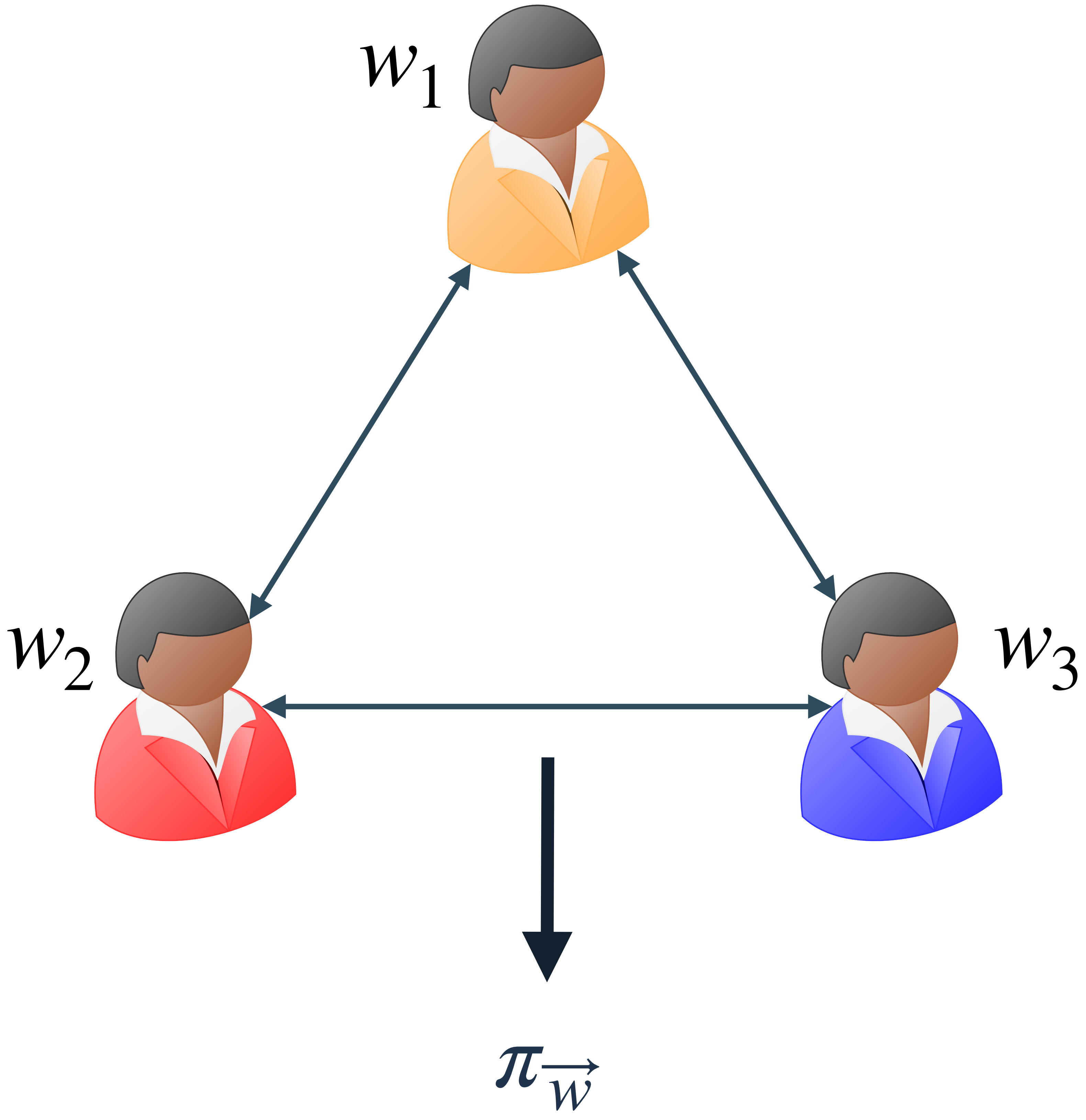
Adversary corrupting a percentage of the parties will still learn nothing but the output,

$$y = f(x_1, x_2, x_3)$$

Linear operations on private data can be done locally

- non-linear operations require communication

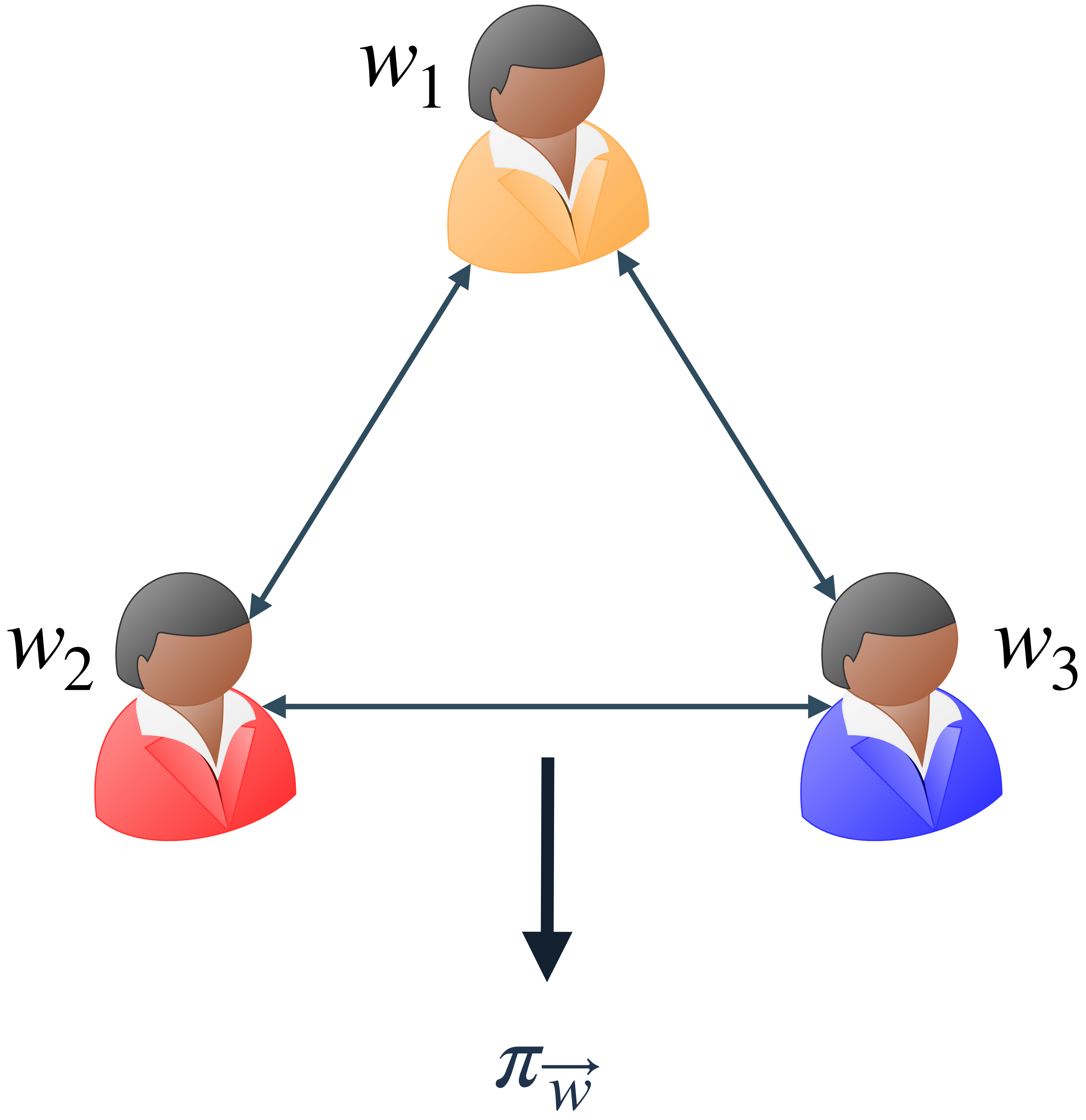
Proof outsourcing with Multiparty Computation (MPC)



<u>Collaborative zkSNARKs</u> **[OB22]**

- zkSNARKs for distributed secrets
- Groth16, Marlin, Plonk (and Fractal)

Proof outsourcing with Multiparty Computation (MPC)



Collaborative zkSNARKs [OB22]

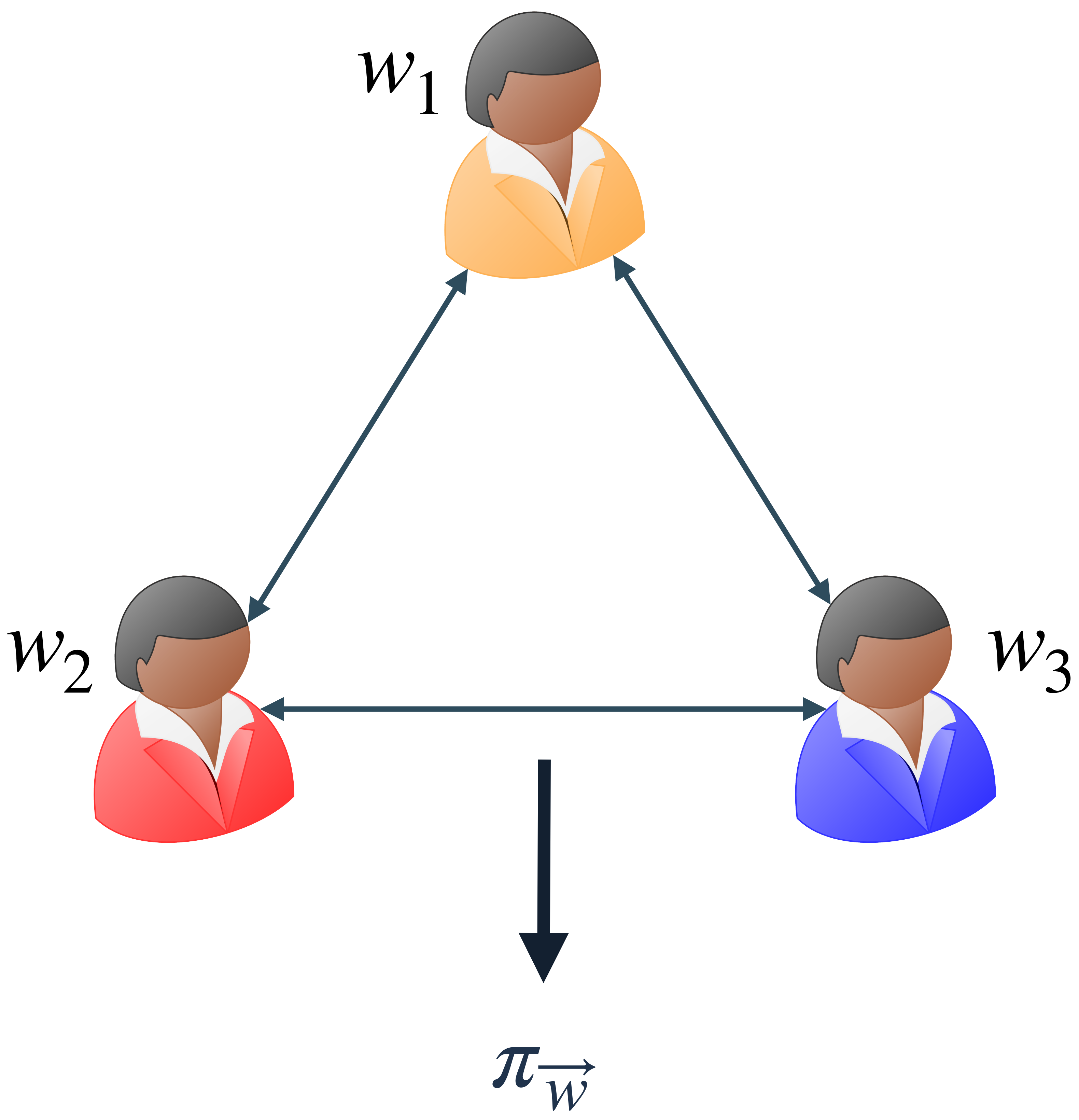
- zkSNARKs for distributed secrets
- Groth16, Marlin, Plonk (and Fractal)

Traditional zkSNARK bottlenecks

- FFTs
- MSMs (multi-scalar multiplications):

  $\sum_i \gamma_i \cdot g_i$ for scalars γ_i and elliptic curve points g_i

Proof outsourcing with Multiparty Computation (MPC)



Collaborative zkSNARKs **[OB22]**

- zkSNARKs for distributed secrets
- Groth16, Marlin, Plonk (and Fractal)

Traditional zkSNARK bottlenecks

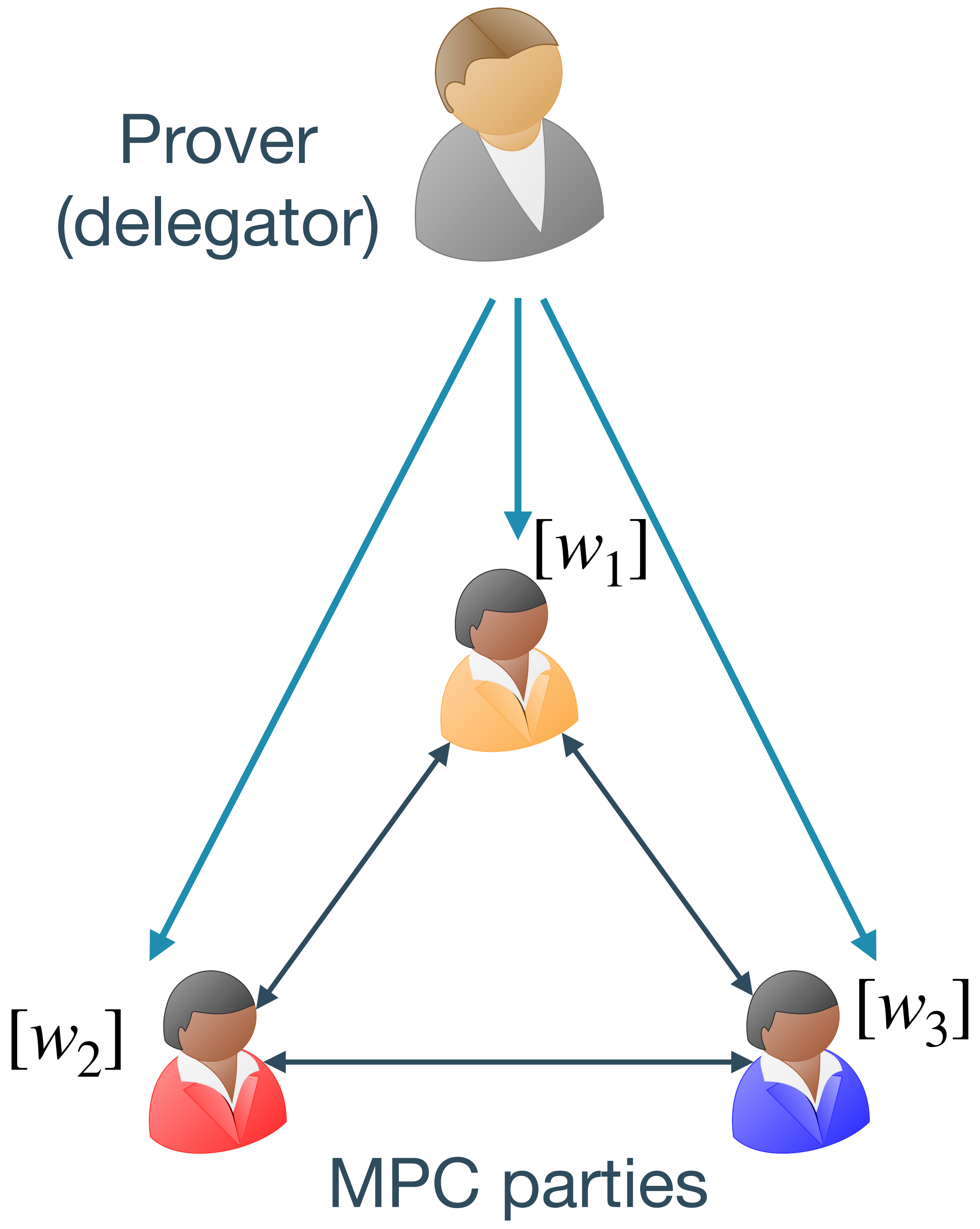
- FFTs
- MSMs (multi-scalar multiplications):

  $\sum_i \gamma_i \cdot g_i$ for scalars γ_i and elliptic curve points g_i

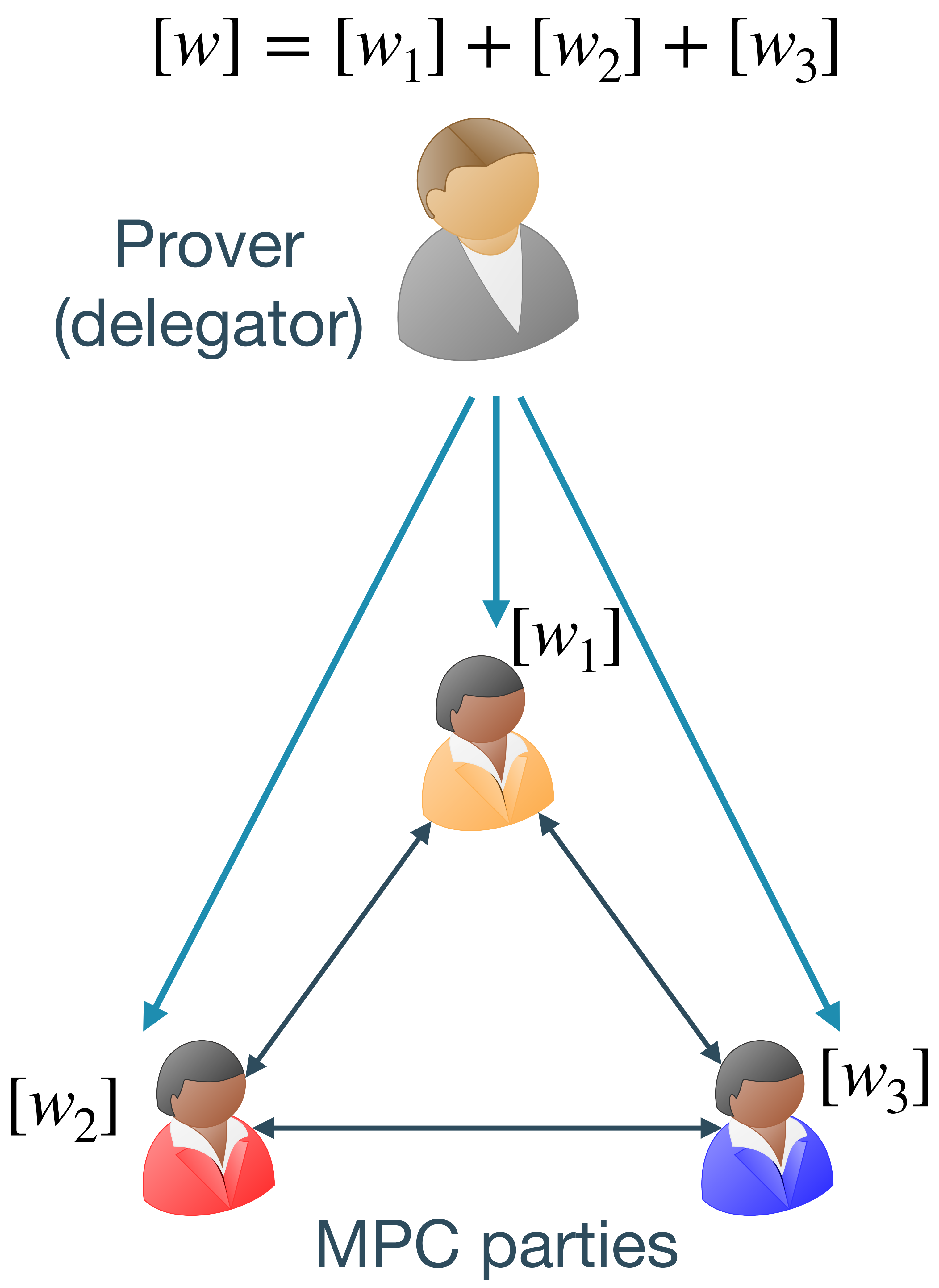
—> Both are linear operations on the witness-dependent data

Proof outsourcing with Multiparty Computation (MPC)

$[w] = [w_1] + [w_2] + [w_3]$



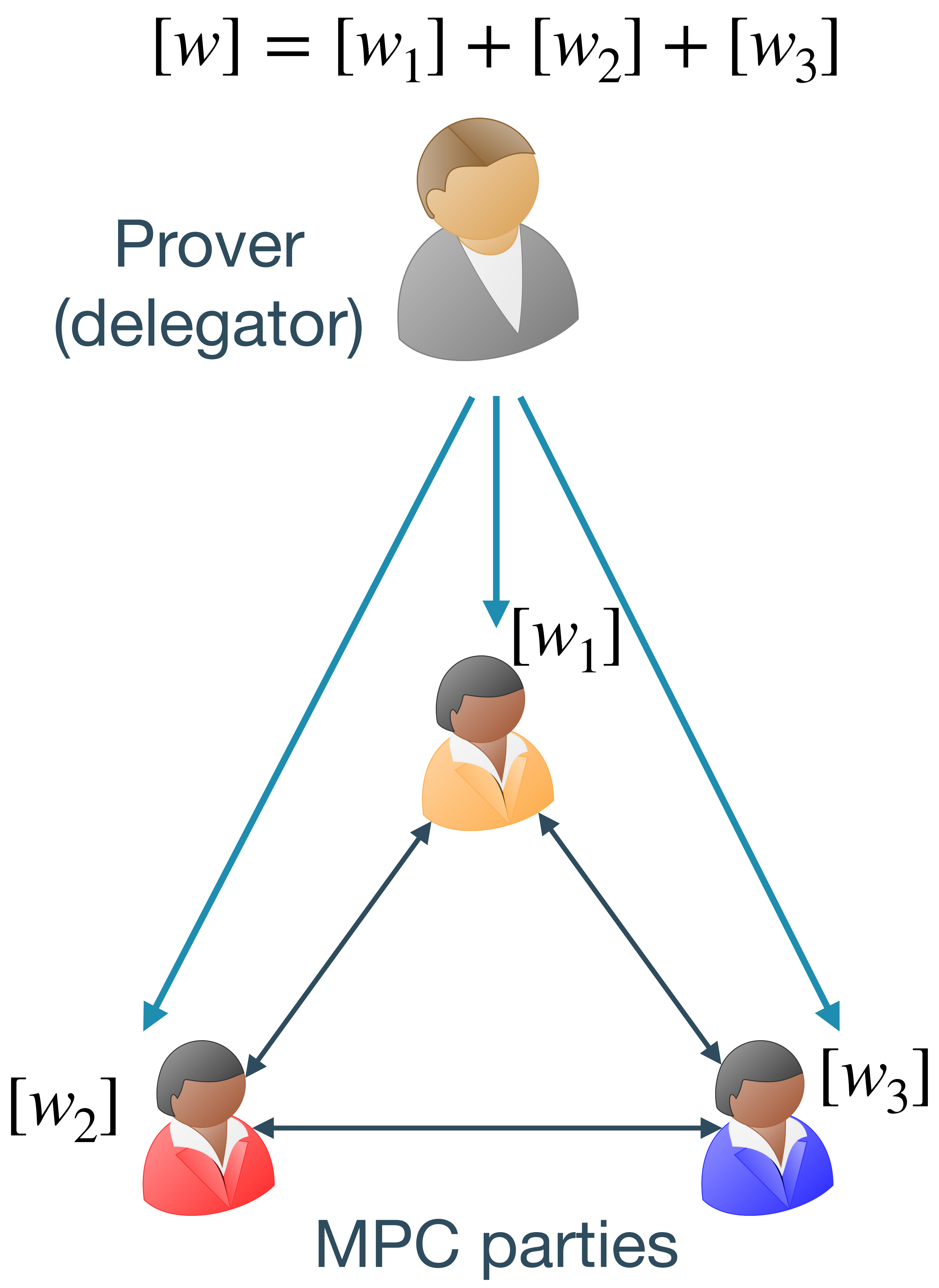
Proof outsourcing with Multiparty Computation (MPC)



Eos [CLMZ23]

- Optimises distributed Marlin proof for outsourcing
- Leverages delegator as a trusted third party
  - to generate correlated randomness
  - to enforce malicious security

Proof outsourcing with Multiparty Computation (MPC)



Eos **[CLMZ23]**

- Optimises distributed Marlin proof for outsourcing
- Leverages delegator as a trusted third party
  - to generate correlated randomness
  - to enforce malicious security

zkSaaS **[GGJPS23]**

- Uses packed secret sharing for SIMD operations
  (at the cost of lower corruption threshold)

Blind zkSNARKs
Private Proof Delegation and Verifiable Computation over Encrypted Data

Mariana Gama
KU Leuven

Emad Heydari Beni
KU Leuven
Nokia Bell Labs

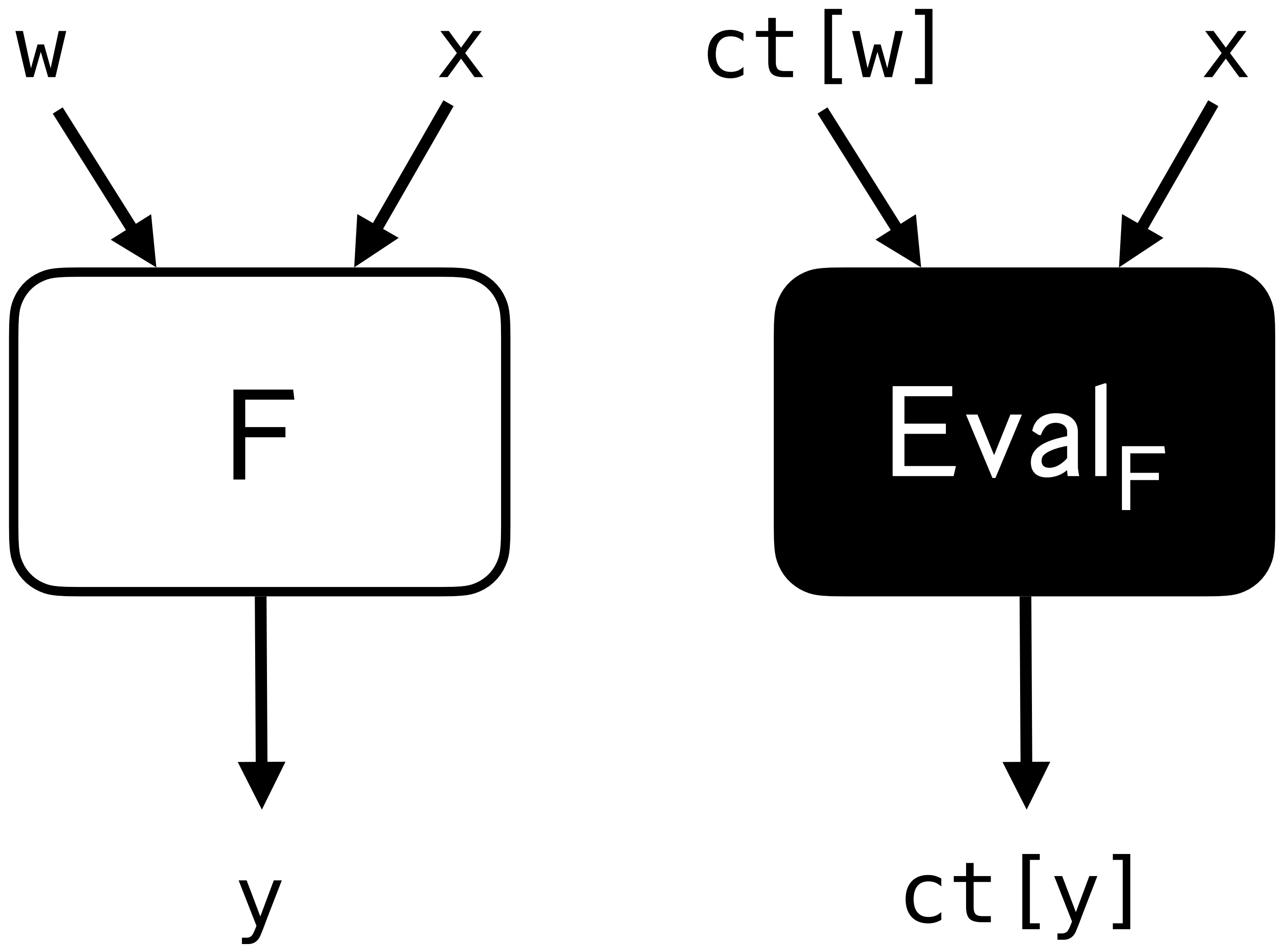
Jiayi Kang
KU Leuven

Jannik Spiessens
KU Leuven

Frederik Vercauteren
KU Leuven

Homomorphic Encryption (HE)

- Transforms arithmetic circuit F into homomorphic circuit $\mathsf{Eval}_{\mathsf{F}}$
- Encrypts inputs with secret key sk such that the other party can **blindly** compute F
- Ciphertext space $\mathbb{Z}_q[X]/\Phi(X)$ homomorphic to plaintext space $\mathscr{P}$: vectors on finite field $\mathbb{F}$



$\mathtt{Dec(ct[y])\ =\ y}$

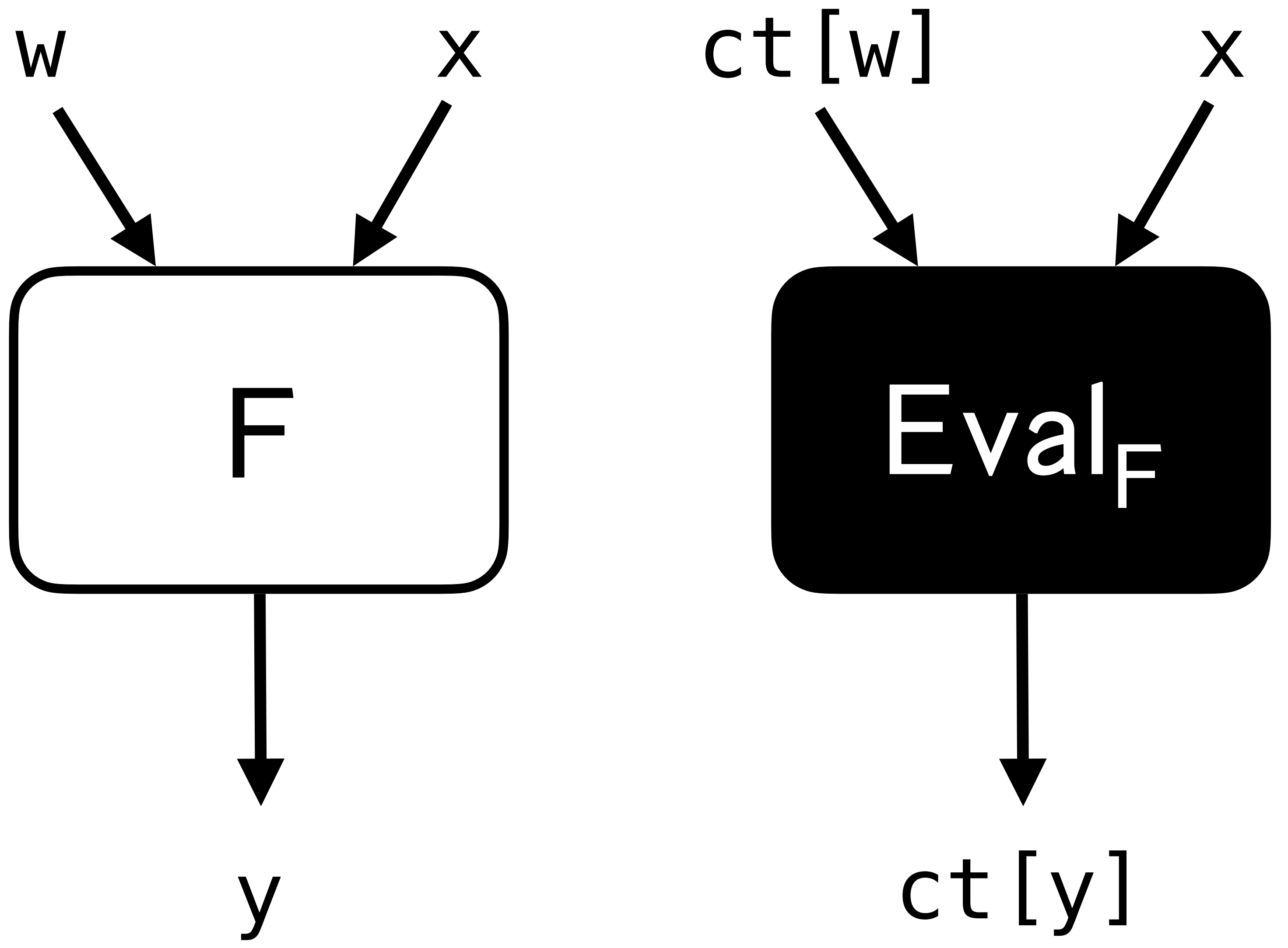
Homomorphic Encryption (HE)

- Transforms arithmetic circuit F into homomorphic circuit Eval_F
- Encrypts inputs with secret key sk such that the other party can **blindly** compute F
- Ciphertext space $\mathbb{Z}_q[X]/\Phi(X)$ homomorphic to plaintext space $\mathscr{P}$: vectors on finite field $\mathbb{F}$

Operations

- Element-wise addition (`pt` or `ct`)
- Element-wise multiplication (`pt` or `ct`)
- Permutation in vector

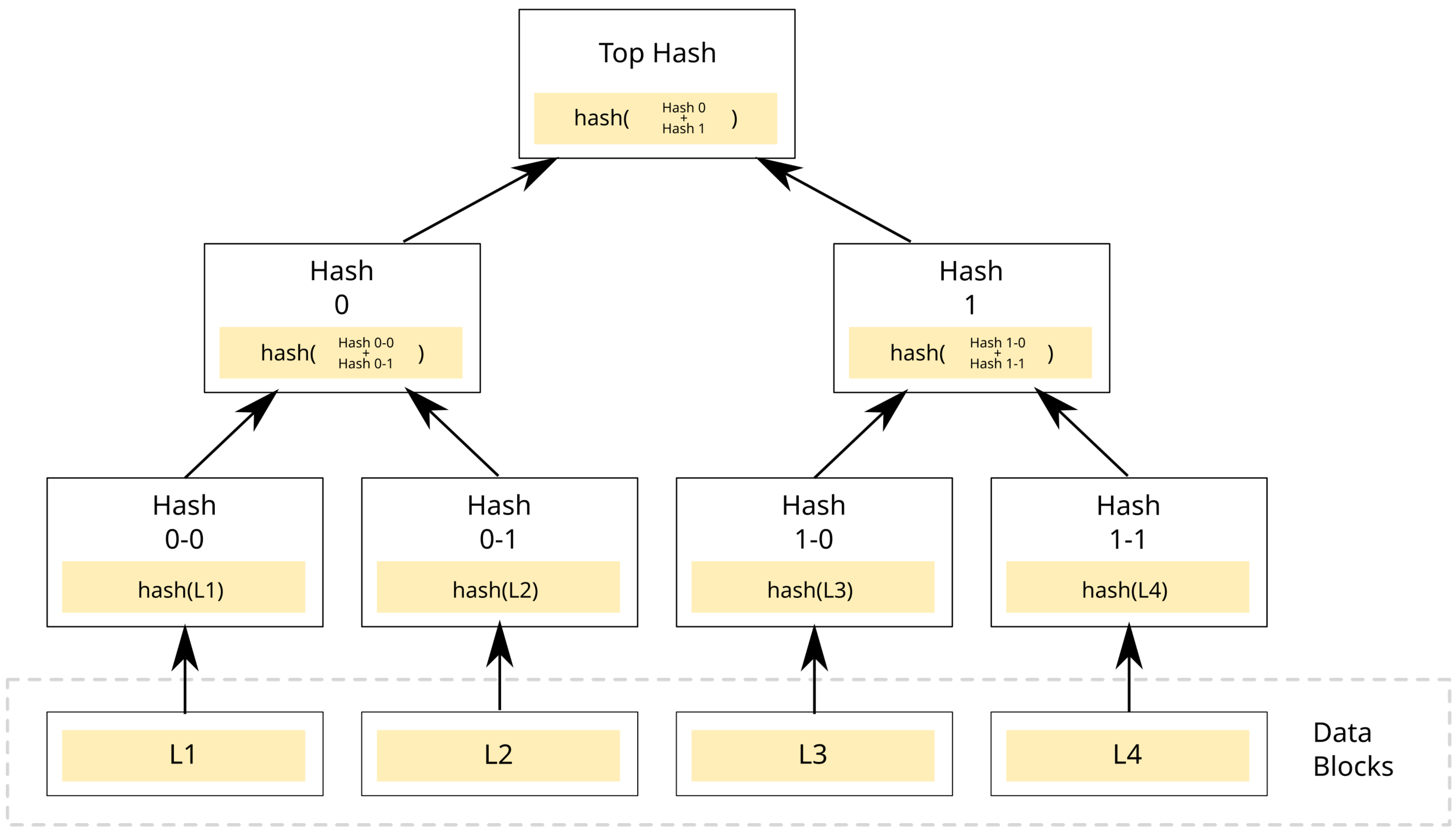
Noise grows with each operation



`Dec(ct[y]) = y`

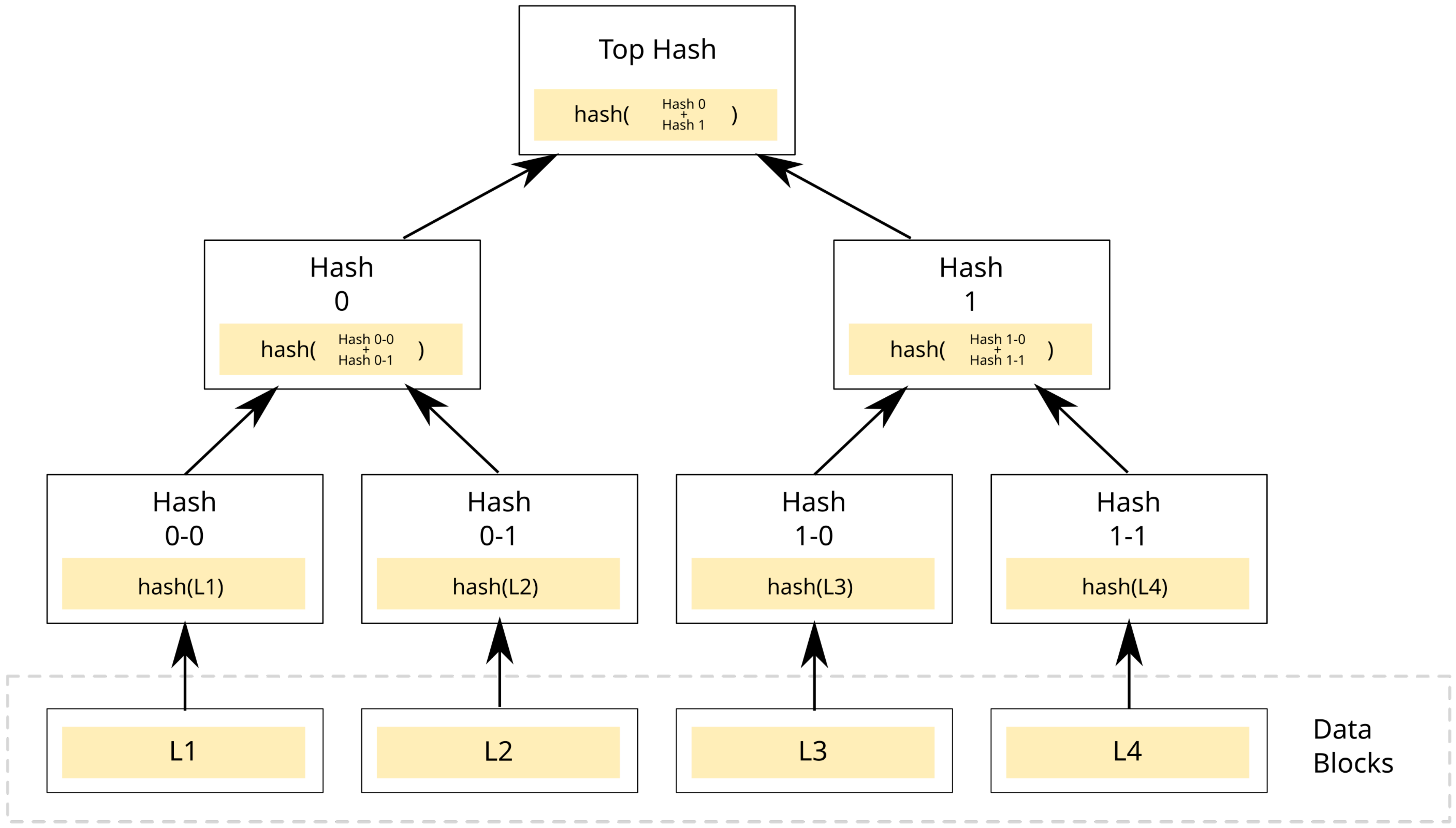
zkSNARK proof delegation with FHE

- The Fractal zkSNARK scheme uses the FRI low degree test, which requires computing Merkle trees.
- Merkle trees are binary trees of hash evaluations —> extremely non-linear



zkSNARK proof delegation with FHE

- The Fractal zkSNARK scheme uses the FRI low degree test, which requires computing Merkle trees.
- Merkle trees are binary trees of hash evaluations —> extremely non-linear



<u>Dealing with Merkle trees</u>

1. Commit to *ciphertext values*
   i.e., hash the ciphertexts “in the clear”

2. Append Proof of Decryption for the queried plaintext/ciphertext pairs

Previous work

How to prove statements obliviously? **[GGW24]**

- First work proposing zkSNARK proof outsourcing with FHE
- Does not specify parameters / how to optimise the computation
- Proposes using homomorphic zkSNARK computation for

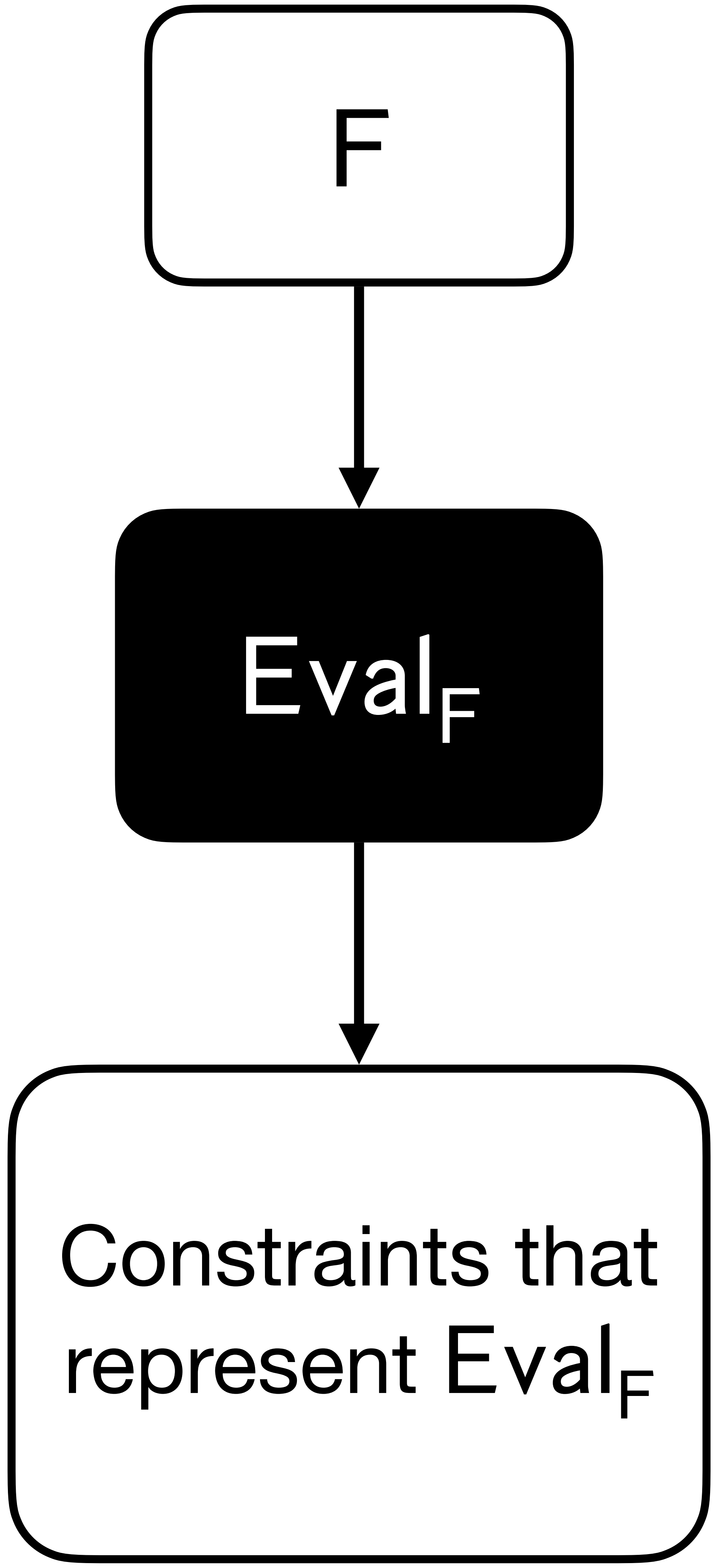
  verifiable computation over encrypted data

Previous work

How to prove statements obliviously? **[GGW24]**

- First work proposing zkSNARK proof outsourcing with FHE
- Does not specify parameters / how to optimise the computation
- Proposes using homomorphic zkSNARK computation for **verifiable computation over encrypted data**

Usual vFHE approach (expensive)

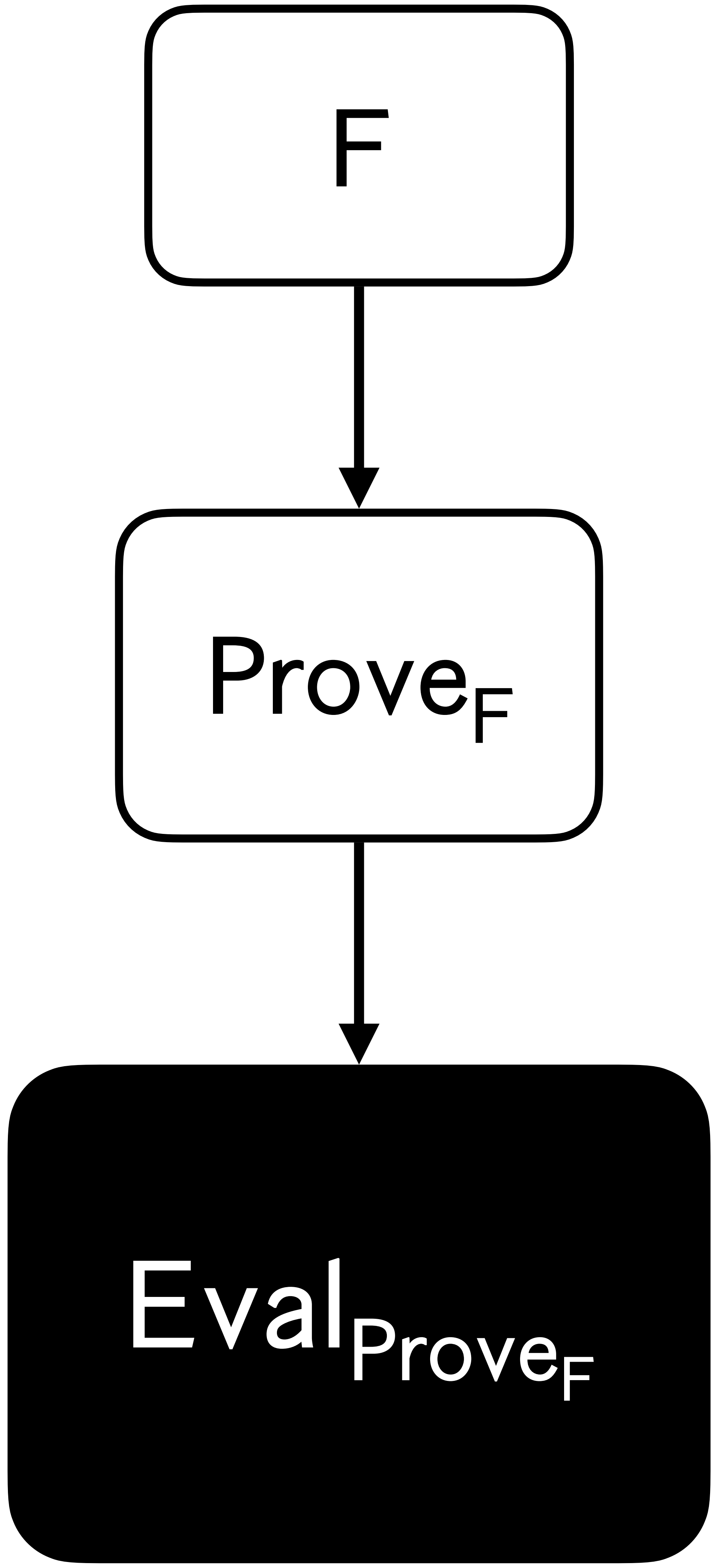


Previous work

How to prove statements obliviously? **[GGW24]**

- First work proposing zkSNARK proof outsourcing with FHE
- Does not specify parameters / how to optimise the computation
- Proposes using homomorphic zkSNARK computation for **verifiable computation over encrypted data**

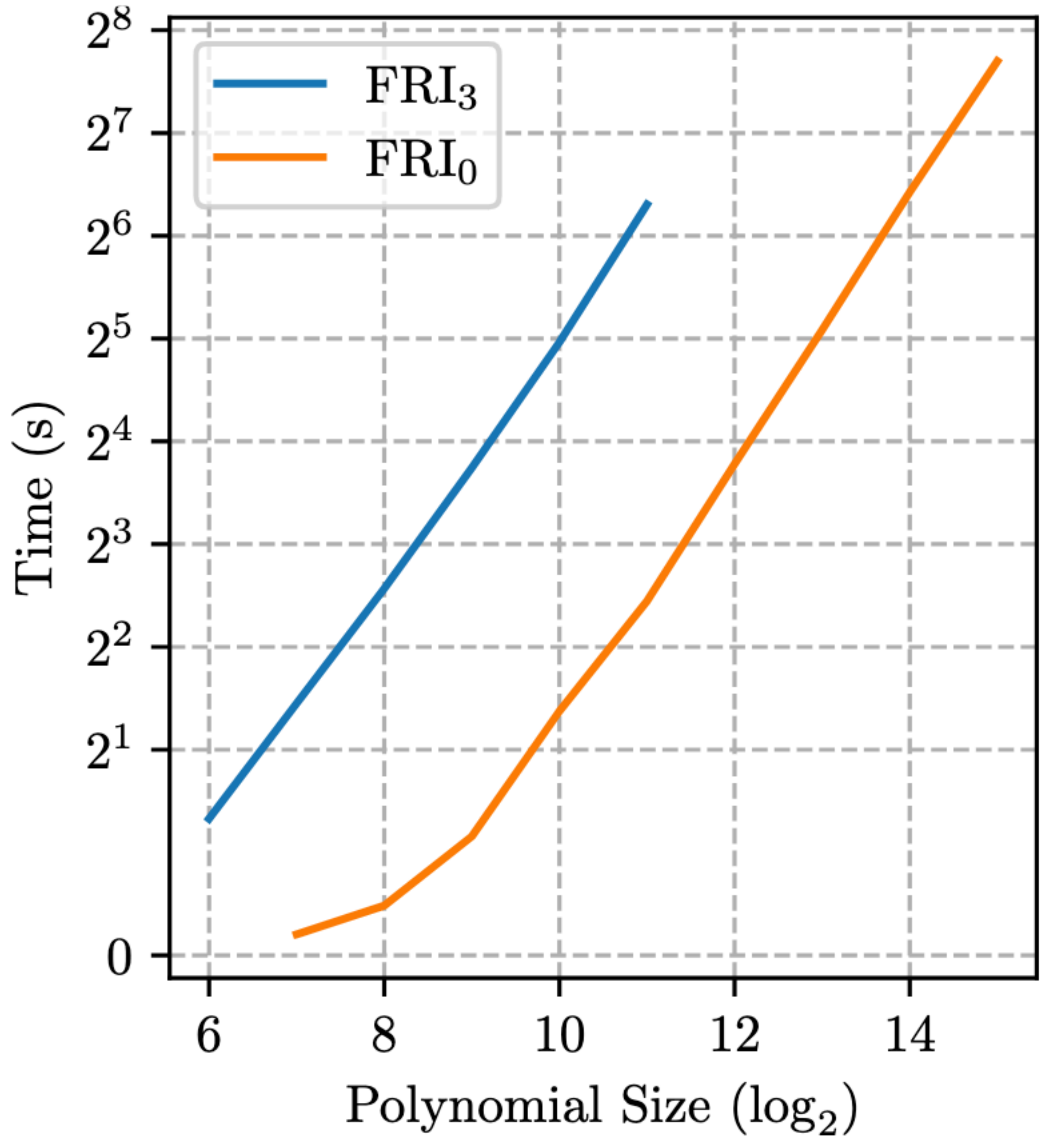
The opposite approach
vCOED



Previous work

HELIOPOLIS [ACGS24]

- Proposes concrete FHE parameters and optimises proof computation
- First **implementation** of homomorphic FRI computation
- Prover executes FRI for polynomials with degree bound 2^{15} in 207 seconds



(a) Prover (32 threads)

Generalised BFV [GV24]

- Ciphertext space $\mathbb{Z}_q[X]/\Phi(X)$ homomorphic to plaintext space $\mathscr{P}$: vectors on finite field $\mathbb{F}$

- Supports SIMD operations (as BGV/BFV **[FV12]**)
- Supports high precision arithmetic (as CLPX **[CLPX18]**)
- We select $\mathscr{P} = \mathbb{F}_{p^2}^{96}$ for $p = 2^{64} - 2^{32} + 1$

Computing Fractal

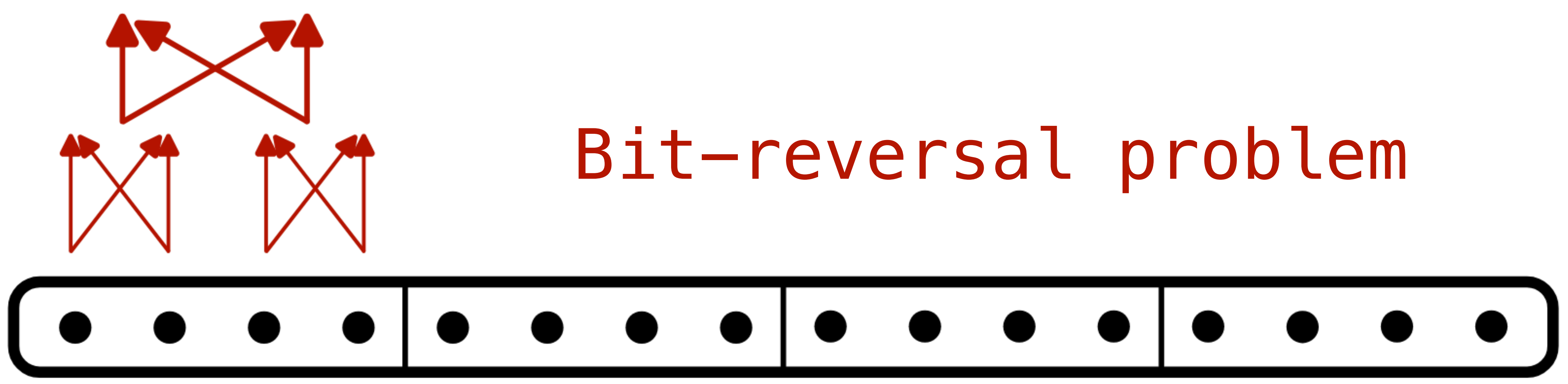
Generally a trade-off between number of operations and noise depth

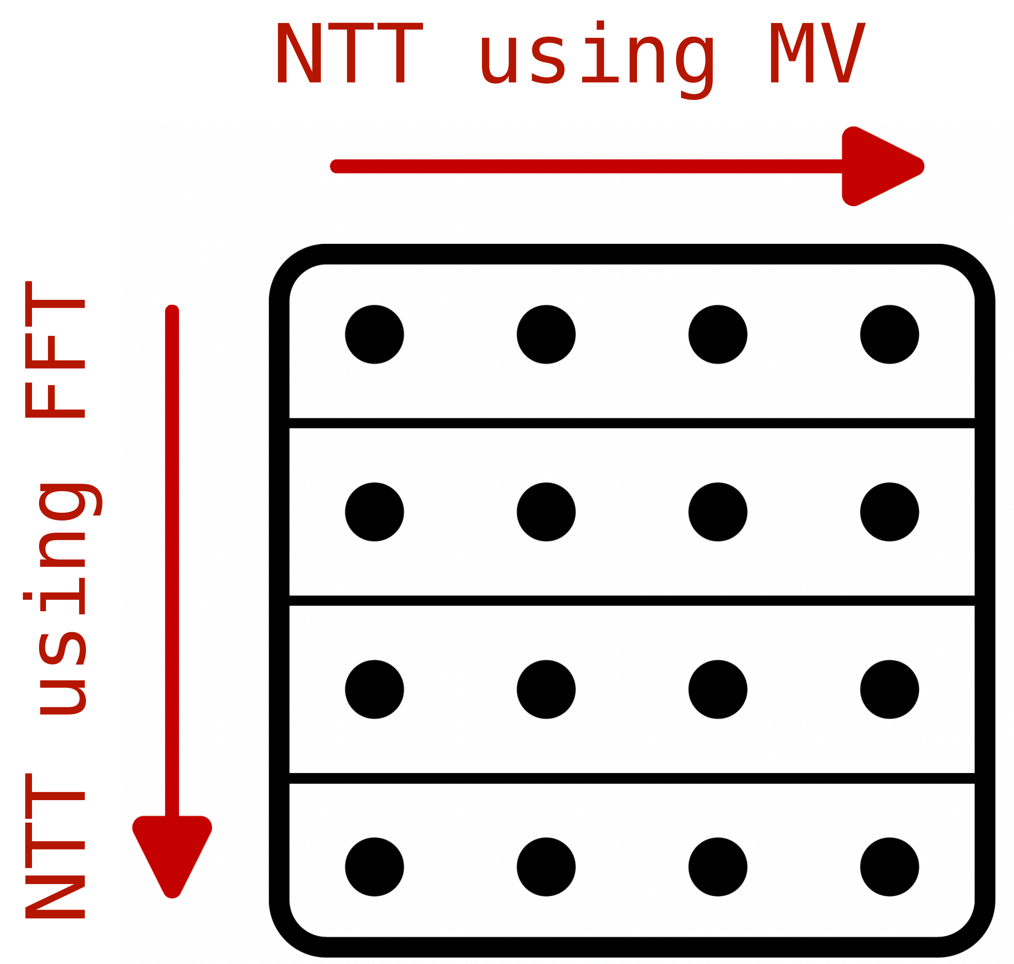
- e.g., domain extensions: compute $f|_L$ from $f|_H$

  Min. number of operations: $f|_L = \mathrm{NTT}\,(\,\mathrm{iNTT}(f|_H))$ using FFT

  Min. noise growth: $f|_L = V_L V_H^{-1} f|_H$

  Solution: 2D NTT





Computing Fractal

Fully parallel on 96 cores: 18min

Example estimate: R1CS with 2^{20} constraints

Computation	Noise (bits)	$C_{\texttt{add}}$	$C_{\texttt{ptct}}$	$C_{\texttt{aut}}$	$C_{\texttt{ctct}}$
Unpacking	9	0	16416	16416	0
Computing $\texttt{ct}[Mz]$	31	196602	163872	163872	0
Computing $\texttt{ct}[\vec{f_z}]/\texttt{ct}[\vec{f_{Mz}}]$	164	6389922	6455412	6455328	0
Computing $\texttt{ct}[\vec{g}]$	298	10633448	10780823	10649632	0
Computing $\texttt{ct}[\vec{f_{\mathsf{FRI}}}]$	298	10895592	11042967	10649632	32768
Computing FRI	318	11354345	11075735	10649632	32768

Operation count and noise estimates for computing blind Fractal

Proof of Decryption

- Proves that $\|c_0 + c_1 \cdot \mathsf{sk} - \lfloor \Delta \cdot m \rceil \|_\infty \leq B$ w.r.t. to committed sk

Proof of Decryption

- Proves that $\|c_0 + c_1 \cdot \mathsf{sk} - \lfloor \Delta \cdot m \rceil \|_\infty \leq B$ w.r.t. to committed sk

- Based on **[LNP22]** Approximate Range Proofs
  - work over $\mathbb{Z}_{q'}[X]/(X^{64}+1)$ instead of $\mathbb{Z}_q[X]/\Phi_m(X)$
  - requires relaxation factor $\approx$ noise space

Proof of Decryption

- Proves that $\|c_0 + c_1 \cdot \mathsf{sk} - \lfloor \Delta \cdot m \rceil \|_\infty \leq B$ w.r.t. to committed sk

- Based on **[LNP22]** Approximate Range Proofs
  - work over $\mathbb{Z}_{q'}[X]/(X^{64}+1)$ instead of $\mathbb{Z}_q[X]/\Phi_m(X)$
  - requires relaxation factor $\approx$ noise space

- Introduce new protocol for batching r PoDs
  - reduces prover cost $O(rn^2) \to O(n^2 + rn\log n)$
  - at the cost of $\approx 6 + \log r$ bits of noise

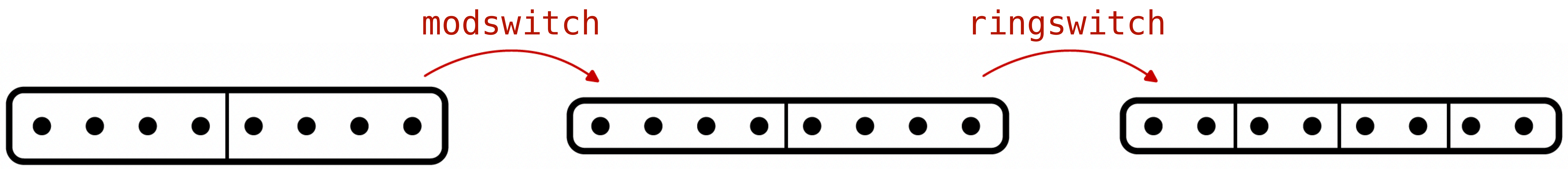
Proof of Decryption

Optimised using HE operations

- Modswitch
  From "FHE-friendly" $\mathbb{Z}_q / \Phi_m(X)$ to "LNP22-friendly" $\mathbb{Z}_{q'} / \Phi_m(X)$
  i.e., from 398 bits to 97 bits

- Ringswitch
  From "efficient" $\mathbb{Z}_{q'} / \Phi_{2^{11}\cdot 3\cdot 7}(X)$ to "small" $\mathbb{Z}_{q'} / \Phi_{2^8\cdot 3\cdot 7}(X)$
  i.e., from 96 slots to 24 slots

MS and RS performed again inside batching protocol



Proof of Decryption

- Implemented in C
- Built upon the LaZer library **[LSS24]**
- Our parameters: blind zkSNARK for 2^{20} R1CS gates
  - Proof size: 12 kB
  - Prover runtime: 2.65s (1 thread) or 0.7s (8 threads)

Main takeaways

- Delegating zkSNARK provers with MPC / FHE is efficient
- Homomorphically computing zkSNARKs enables verifiable computation over encrypted data

Blind zkSNARKs

- Appending a proof of decryption enables public verifiability
- Efficient instantiation using GBFV + PoD adapted from **[LNP22]**