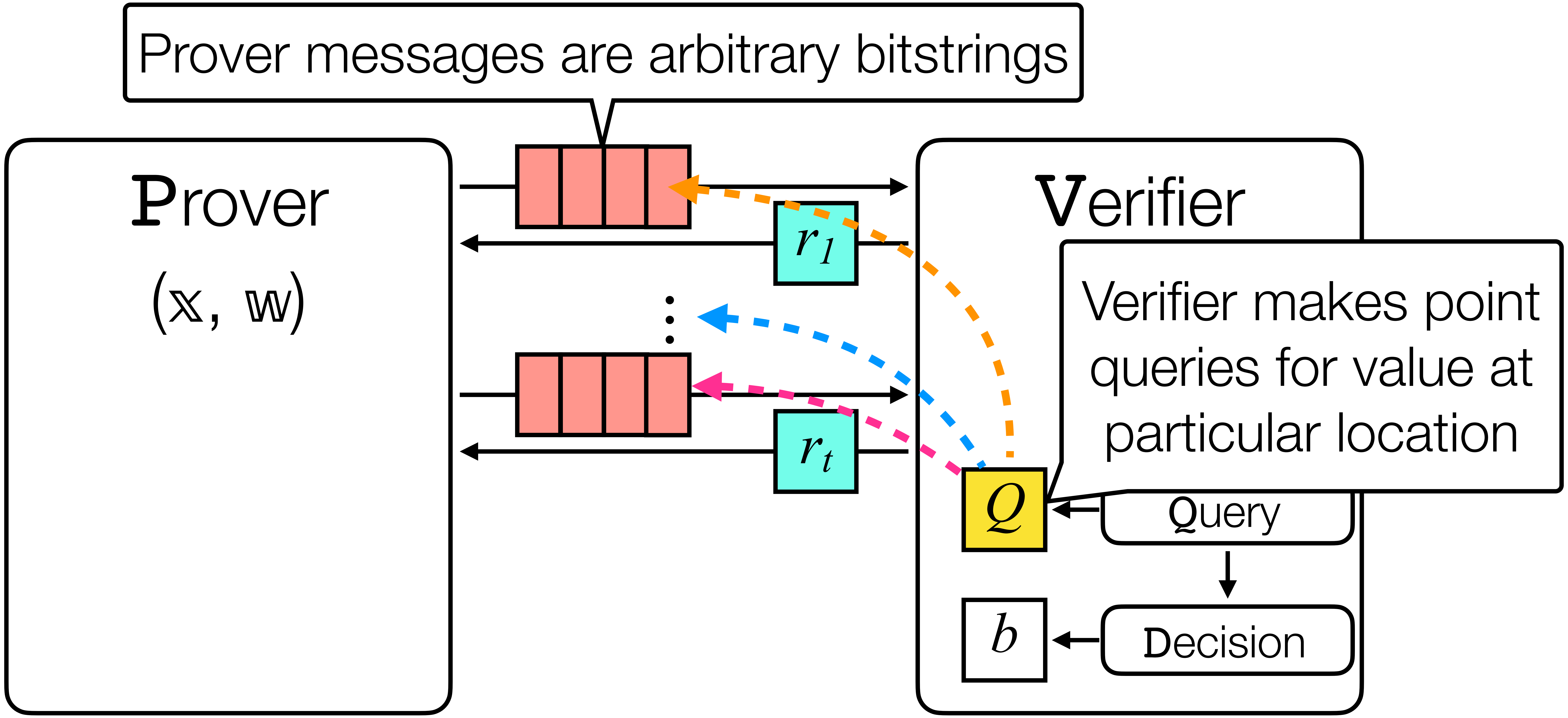
zkSNARKs from Polynomial Commitments

Pratyush Mishra
Aleo, UC Berkeley

Recall IOPs



What's inside common IOPs?

A: Polynomials!

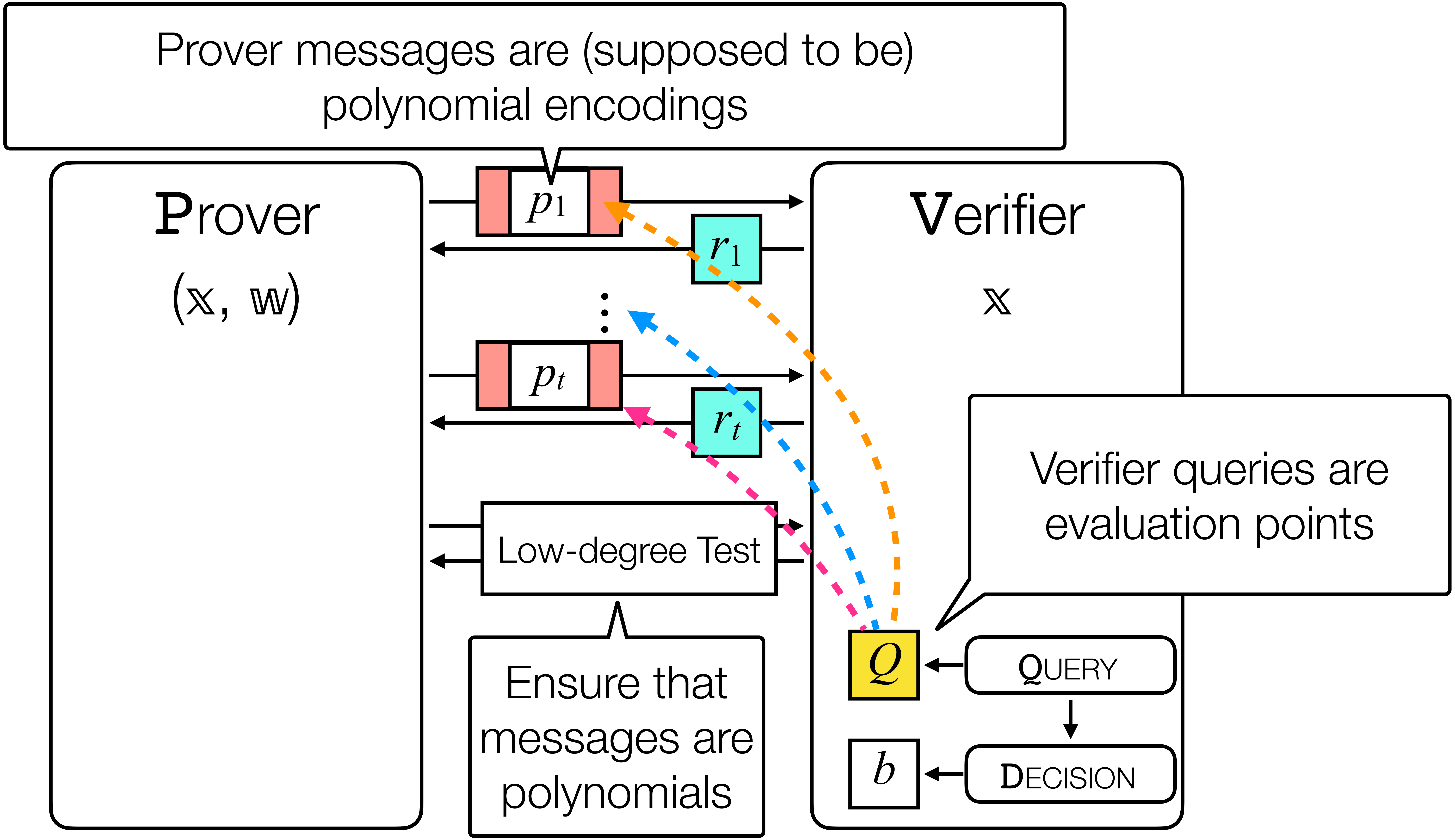
Benedikt Bünz
@benediktbuenz

Replying to @Zac_Aztec

Reed - Solomon code: Polynomial
Zero-Knwoledge Proof Systems: Polynomials
Secret Sharing: Polynomial Evaluations
Identity Testing: Polynomials equal?
FFTs: Polynomials
FRI: FFTs-> Polynomials
SNARK: Polynomials
STARK: SNARK
Security Parameter: Polynomial
Lagrange: Polynomial

11:28 AM · Oct 25, 2021 · Twitter Web App

A: Polynomials!



A: Polynomials!

Prover messages are (supposed to be) polynomial encodings

What if we just assume messages are polynomials?

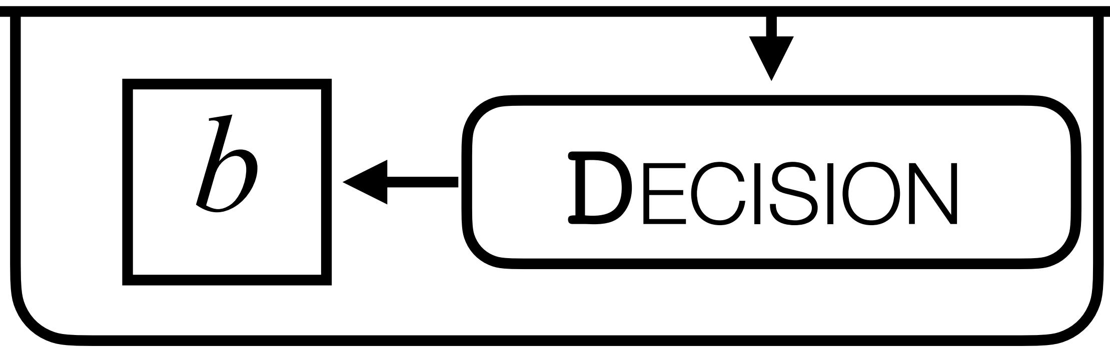
That is, no LDTs!

ies are

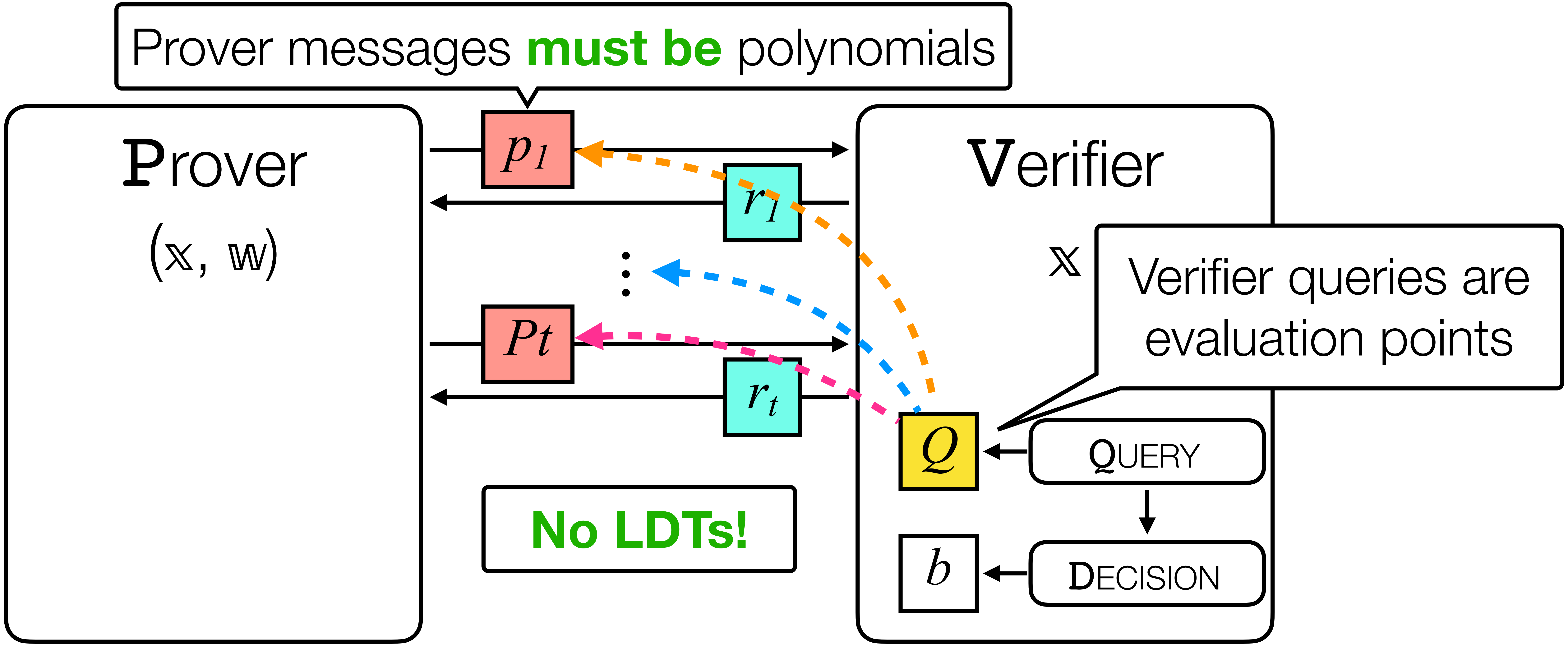
points

Pro

Large proof size due to many queries



Polynomial IOPs [GWC19, CHMMVW20, BFS20]



- **Completeness**: Whenever $(\mathbb{x}, \mathbb{w}) \in R$, there is a strategy for $\mathbf{P}$ that outputs **only polynomials**, and which causes $\mathbf{V}$ to accept.
- **Knowledge Soundness**: Whenever $\mathbf{V}$ accepts against a $\mathbf{P}$ that outputs **only polynomials**, then $\mathbf{P}$ “knows” $\mathbb{w}$ such that $(\mathbb{x}, \mathbb{w}) \in R$.

A Selection of PIOP Constructions

Many PIOPs with many different properties:

- Linear degree of oracles
- Linear prover time
- Sublinear verification for repeated circuits
- …

This has been achieved by leveraging a variety of underlying techniques:

- Univariate and multivariate sum checks
- Grand product checks
- Permutation checks and lookup tables
- …

Overall, PIOPs provide a strong foundation for constructing SNARKs with interesting properties and strong efficiency.

So, how to get SNARKs from PIOPs?

Compiling PIOPs to SNARKs via analogy

Can the IOP-to-SNARK compiler teach us how to construct a PIOP-to-SNARK compiler?

K (see Nick's talk)

Need to

1. succinctly **commit** to a **vector**
2. **open** the commitment at **arbitrary locations**

P

Vector Commitment (eg: MT)

SNARK

Prover
1. VC.Commit(IOP oracles)
2. VC.Open(query set)

Verifier
1. VC.Check(opening proof)

PIOP to

Need to

1. succinctly **commit** to a **polynomial**
2. **open** the commitment at **arbitrary points**

PI

??? Commitment

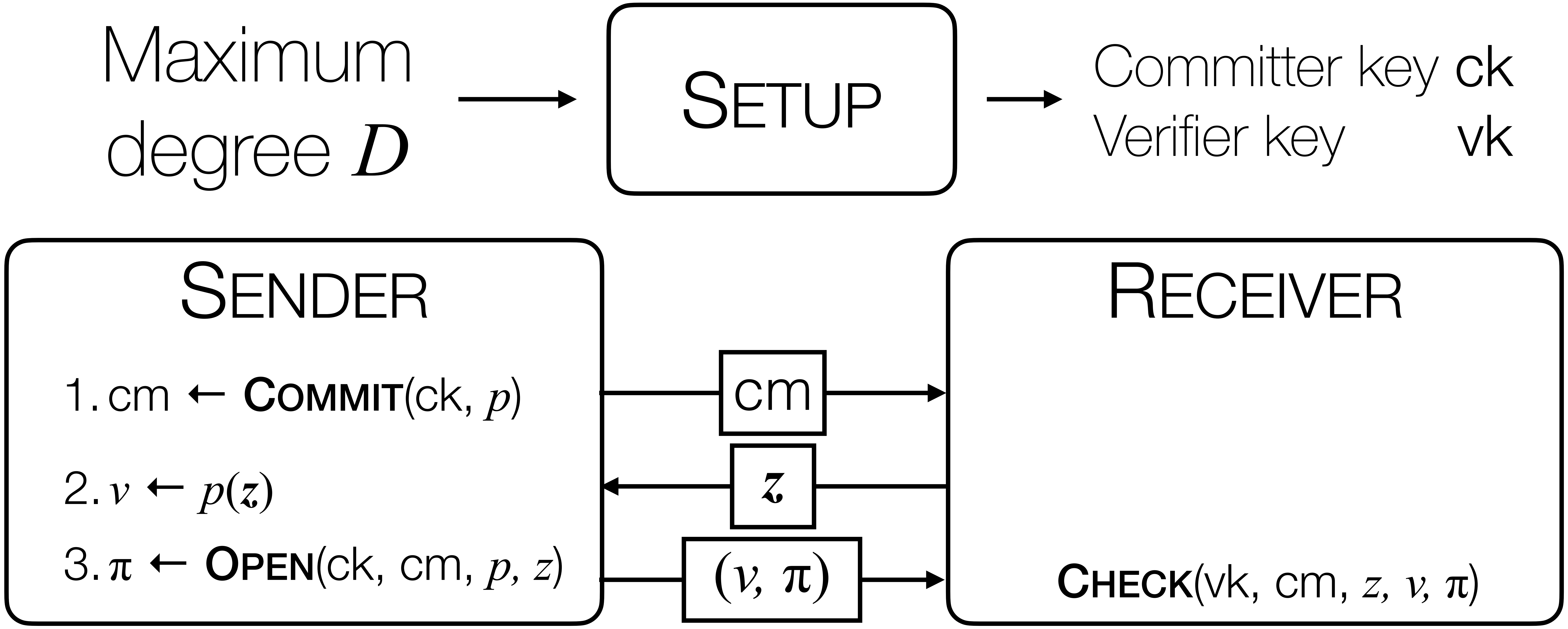
SNARK

Prover
1. ??.Commit(PIOP oracles)
2. ??.Open(query set)

Verifier
1. ??.Check(eval. proof)

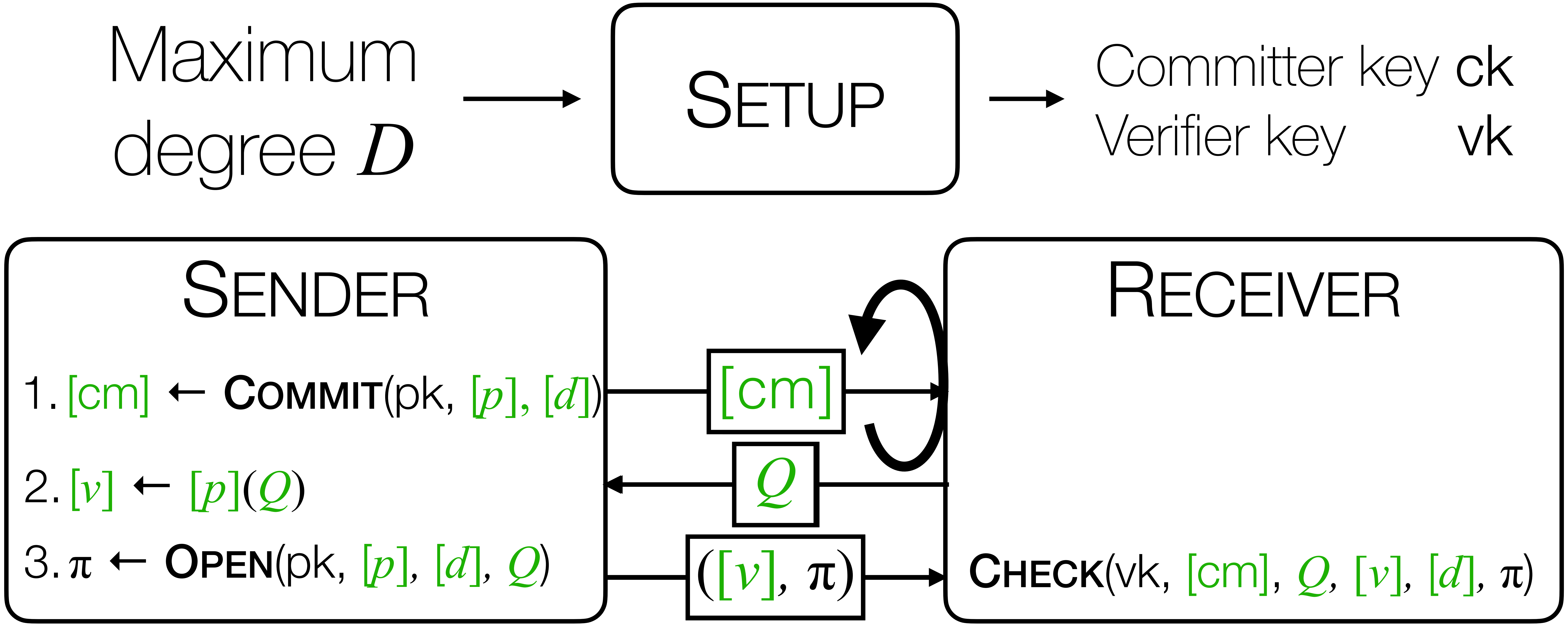
Polynomial Commitments

Polynomial Commitments



- **Completeness**: Whenever $p(z) = v$, **R** accepts.
- **Extractability**: Whenever **R** accepts, **S**'s commitment **cm** "contains" a polynomial p of degree at most D.

Polynomial Commitments



For efficiency improvements, you need

- Batch commitment
- Batch opening

A selection of constructions

In the last 10 years, several constructions with different

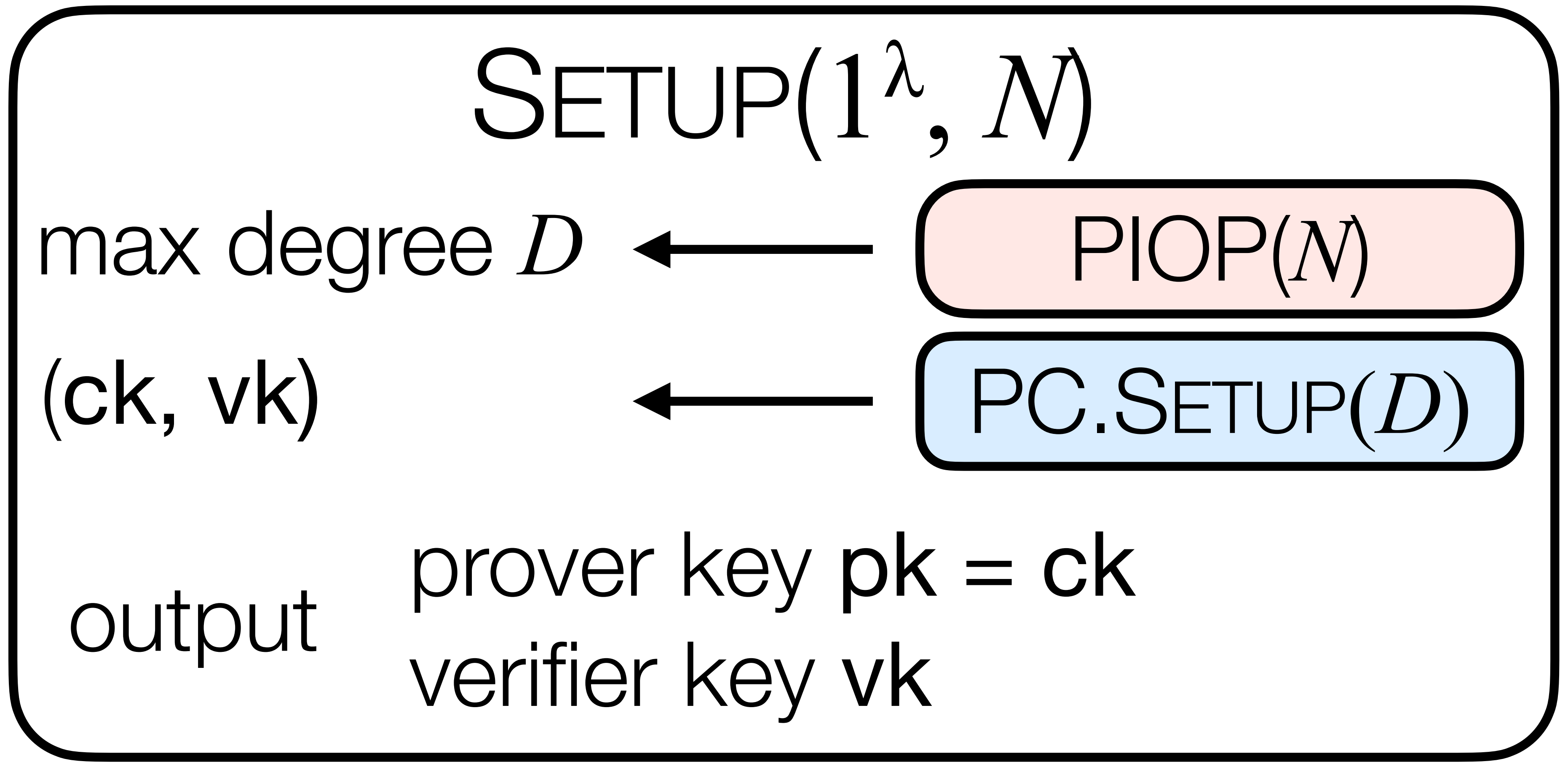
- Cryptographic assumptions
- Prover and verifier efficiency and proof sizes
- Homomorphism and batching properties

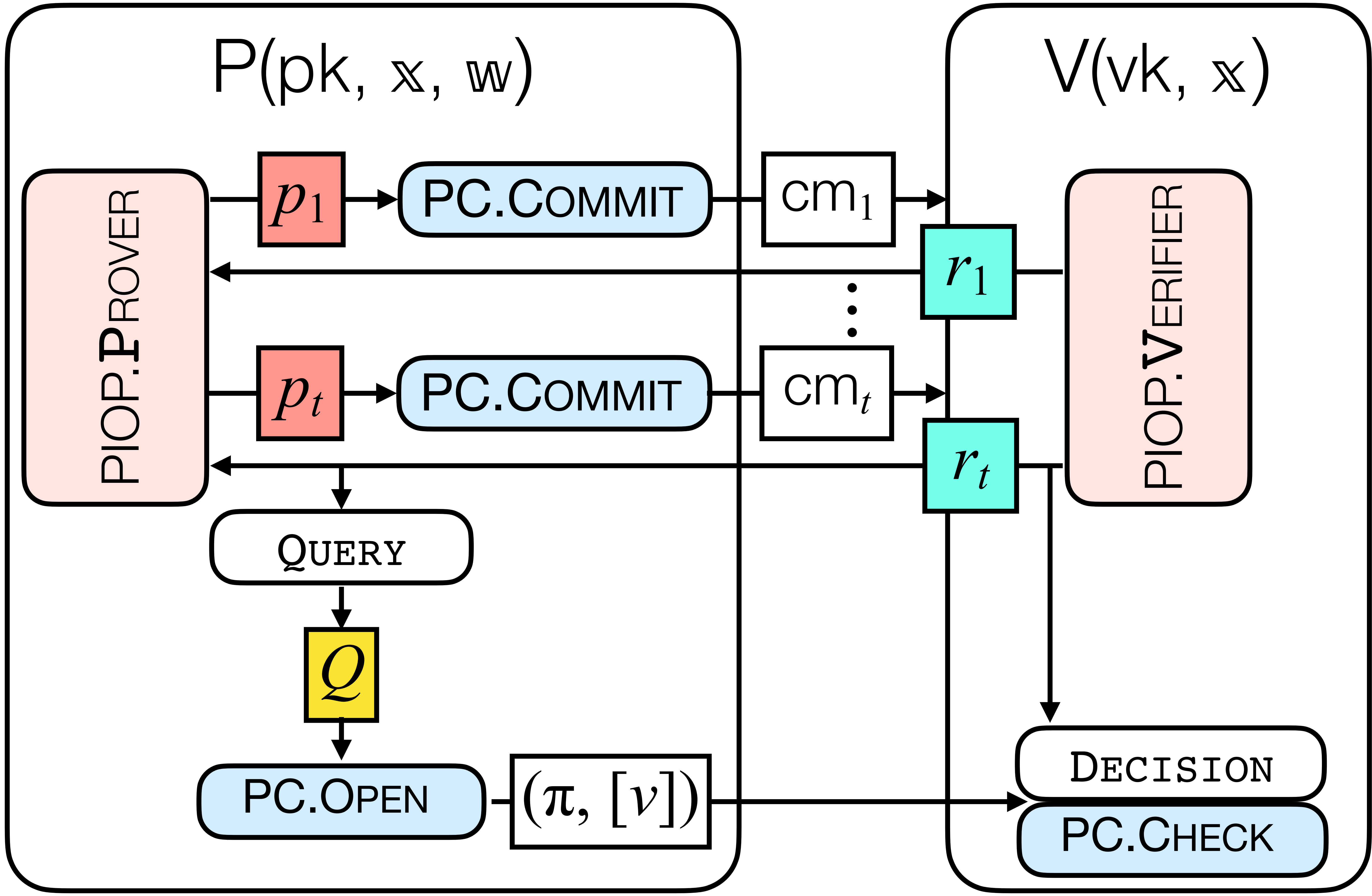
Looking ahead, this enables SNARKs with many different properties

	KZG10	**PST13**	**IPA**	**Hyrax**	**Dory**	**BFS20**
crypto	Pairings	Pairings	DLog + RO	DLog + RO	Pairing + RO	GUO + RO
# variables	1	m	1	m	1	1
setup type	Private	Private	Public	Public	Public	Public
commitment size	$O(1)$ G	$O(1)$ G	$O(1)$ G	$O(2^{m/2})$ G	$O(1)$ G	$O(1)$ G
proof size	$O(1)$ G	$O(m)$ G	$O(\log d)$ G	$O(2^{m/2})$ G	$O(\log d)$ G	$O(\log d)$ G
verifier time	$O(1)$ G	$O(m)$ G	$O(d)$ G	$O(2^{m/2})$ G	$O(\log d)$ G	$O(\log d)$ G

Combining PIOPs and PC schemes

PIOPs + PC Schemes → SNARK





+ Fiat—Shamir to get non-interactivity

Argument size comparison

Asymptotic

	Oracle commitments	Oracle values	Opening proofs
IOP-based	$t \times \lvert\mathrm{cm}\rvert$	$+ \lvert Q\rvert \times \log\lvert\mathbb{F}\rvert +$	$\lvert Q\rvert \times 2\kappa \log(D)$
PIOP-based	$t \times \lvert\mathrm{cm}\rvert$	$+ \lvert Q\rvert \times \log\lvert\mathbb{F}\rvert +$	$\lvert\pi_{\mathrm{PC}}\rvert$

Concrete

	Argument size
FRI-based IOP	~100kB
Univariate PIOP w/ IPA	~ 5 kB
Univariate PIOP w/ KZG	~ 600 B

- D is max degree
- t is number of polys
- $\mathbb{F}$ is field of definition
- $\lvert Q\rvert$ is number of queries

Why the big difference?

1. $\lvert Q_{\mathrm{IOP}}\rvert \gg \lvert Q_{\mathrm{PIOP}}\rvert$, as no LDT.
   Hence
   $\lvert Q_{\mathrm{IOP}}\rvert \times \log\lvert\mathbb{F}\rvert \gg \lvert Q_{\mathrm{PIOP}}\rvert \times \log\lvert\mathbb{F}\rvert$

2. $\lvert\pi_{\mathrm{PC}}\rvert \ll \lvert Q_{\mathrm{IOP}}\rvert\, 2\kappa \log(D)$
   for many PC schemes due to algebraic structure and batching. Eg: for KZG, eval. proof requires only 1 G per unique point

Summary

SNARKs from PIOPs and PC schemes are efficient and have much smaller argument size than IOP-based SNARKs (but assume structured crypto)

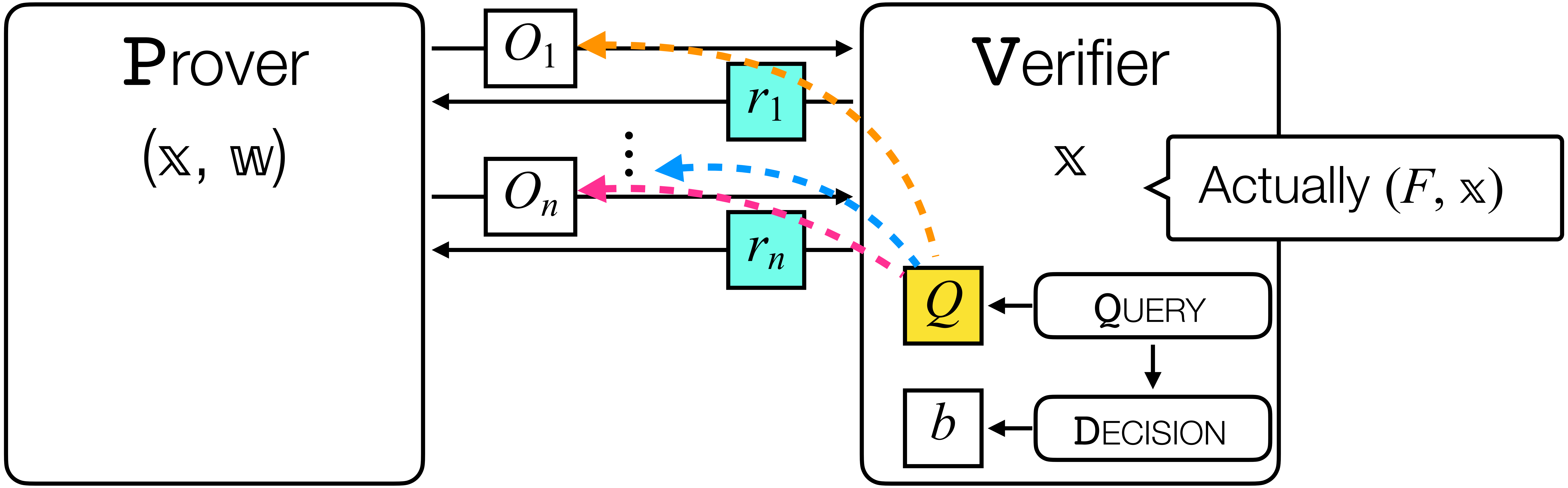
Sublinear verification for (P)IOP-based SNARKs

Verifier Complexity of (P)IOP-based SNARKs

$$T(\text{SNARK.V}) = T(\text{CHECK}) + T(\text{IOP.}\mathbf{V})$$

Can make this sublinear (eg: MT or KZG)

What about this?

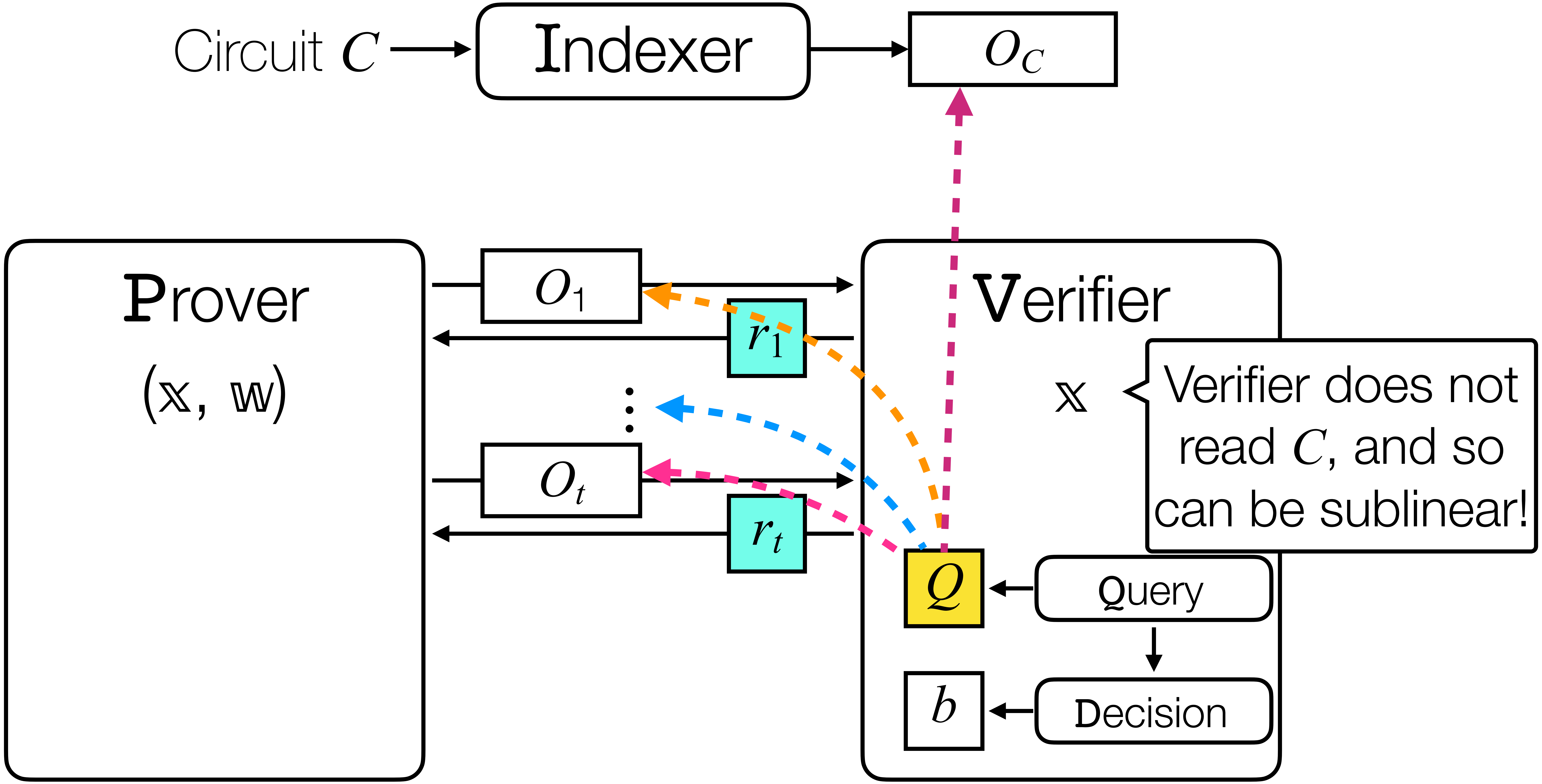


IOP Verifier has to **at least** read $(F, \mathbb{x})$

- When size of $F \ll$ size of computation (eg machine computations), TIME($\mathbb{V}$) **is sublinear.**
- When size of $F =$ size of computation (eg circuit computations), TIME($\mathbb{V}$) **is linear!**

Holographic (P)IOPs [CHMMVW20, COS20]

Introduce a new algorithm to preprocess the circuit



Holographic (P)IOPs + PC Schemes → Preprocessing SNARKs

Prover answers queries to circuit oracles too

+ Fiat—Shamir to get non-interactivity

Verifier Complexity of Holographic (P)IOP-based SNARKs

$$T(\mathrm{SNARK.V}) = T(\textsc{Check}) + T(\mathrm{HIOP}.\mathbf{V})$$

Now sublinear!

Holography enables sublinear verification for arbitrary circuits computations!

Thanks!

Lots of exciting future directions:

- PIOPs:
  - reduce prover memory,
  - total poly degree,
  - cheaper holography

- PC schemes:
  - efficient constructions from new assumptions (eg: lattices)
  - better constructions from existing assumptions (eg: succinct verification from DL)

- Applications:
  - Eg: accumulation for PC schemes and IOPs → efficient recursive SNARKs