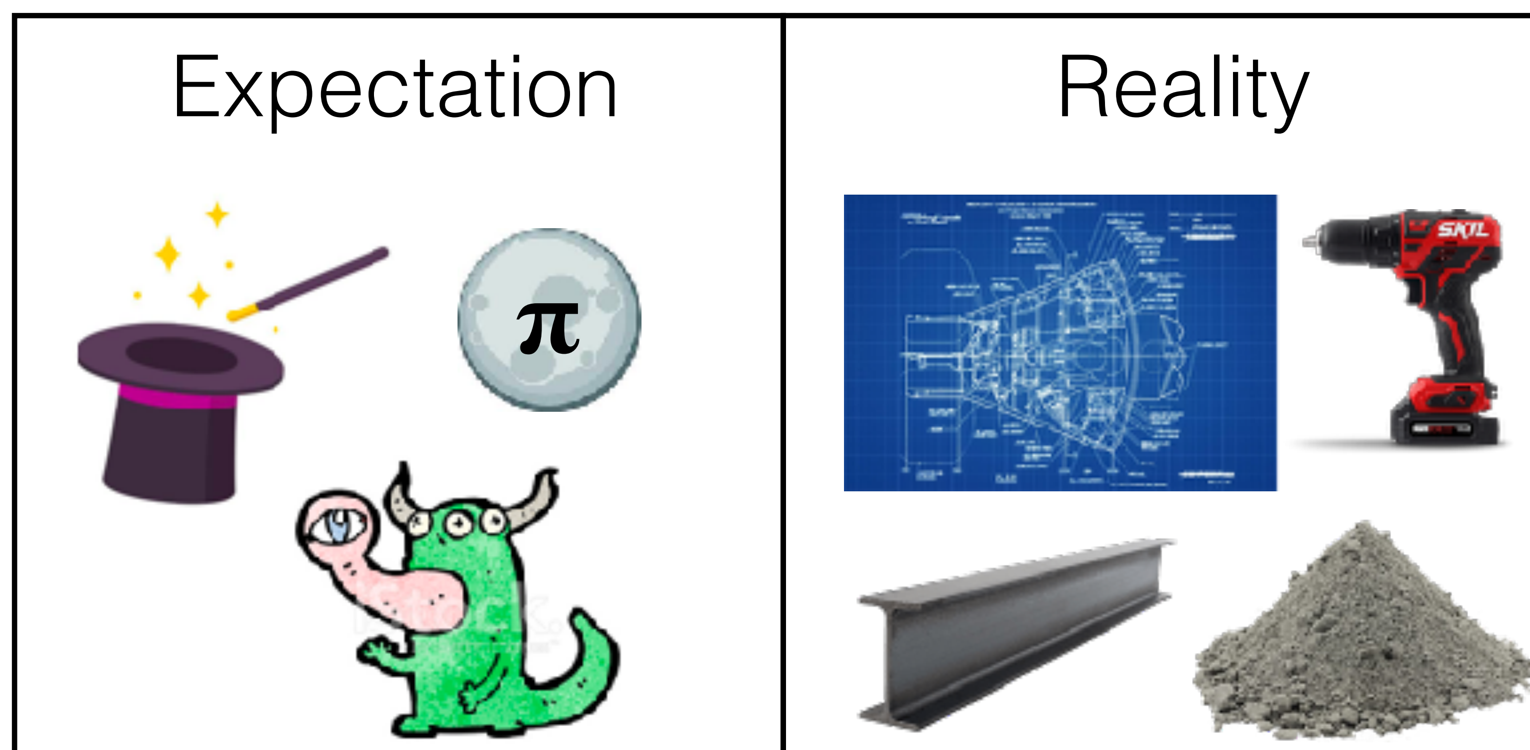
How to Make SNARKs

Alessandro Chiesa

EPFL

UC Berkeley

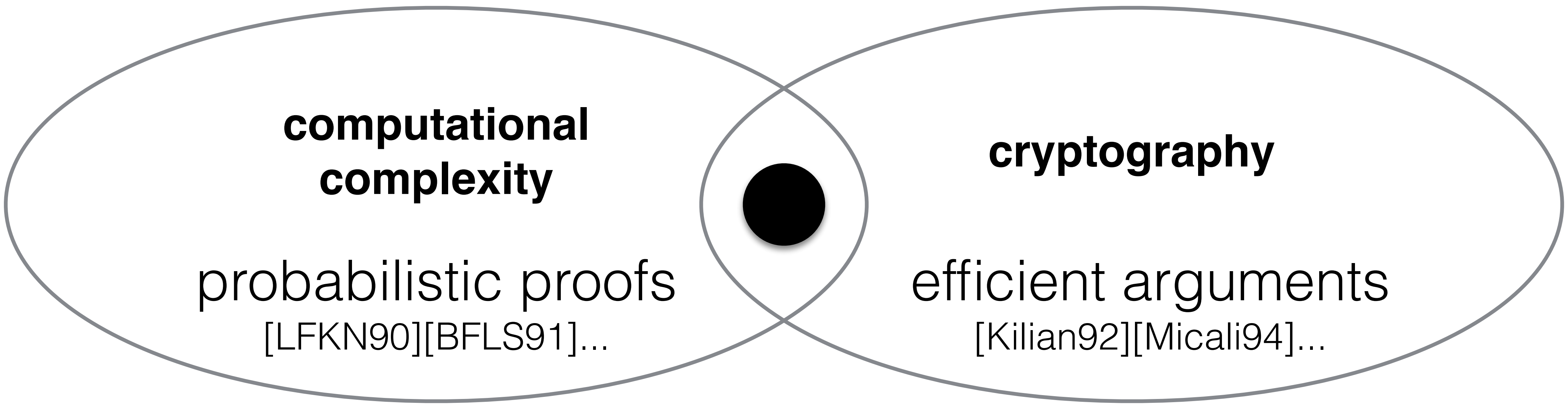
StarkWare



SNARKs

Cryptographic proofs for computation integrity
that are super short and are super fast to verify.

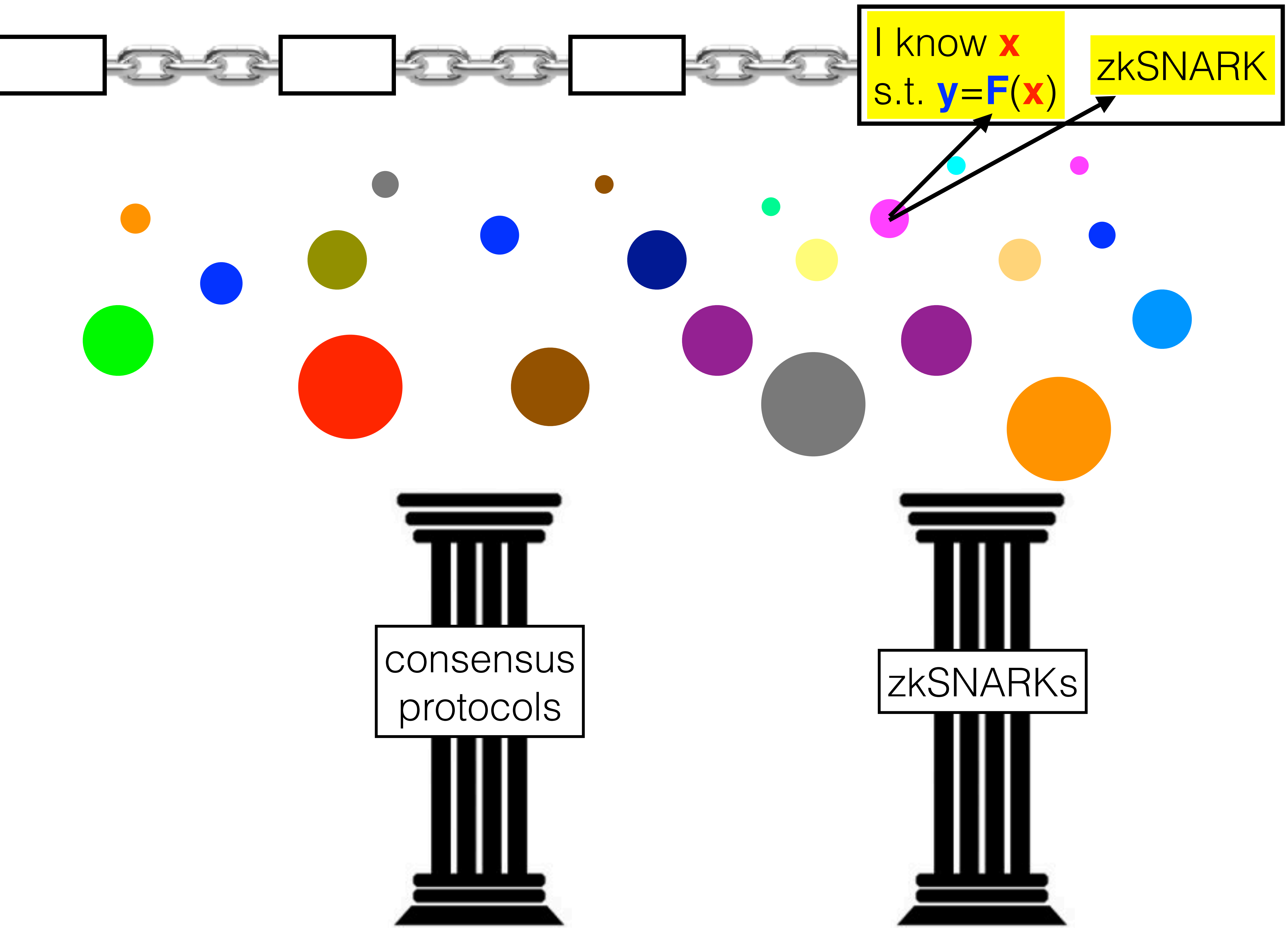
Origins in the 1990s:



In last 10 years, extraordinary progress in:

- cryptographic foundations
- efficient constructions
- implementations
- applications
- ...

Peer-to-Peer Protocols



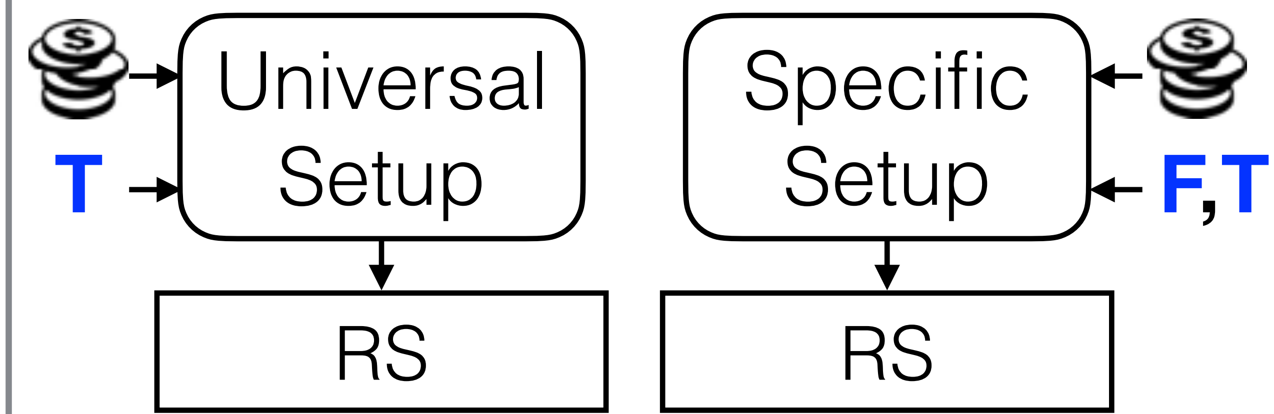
Succinct Non-Interactive Arguments

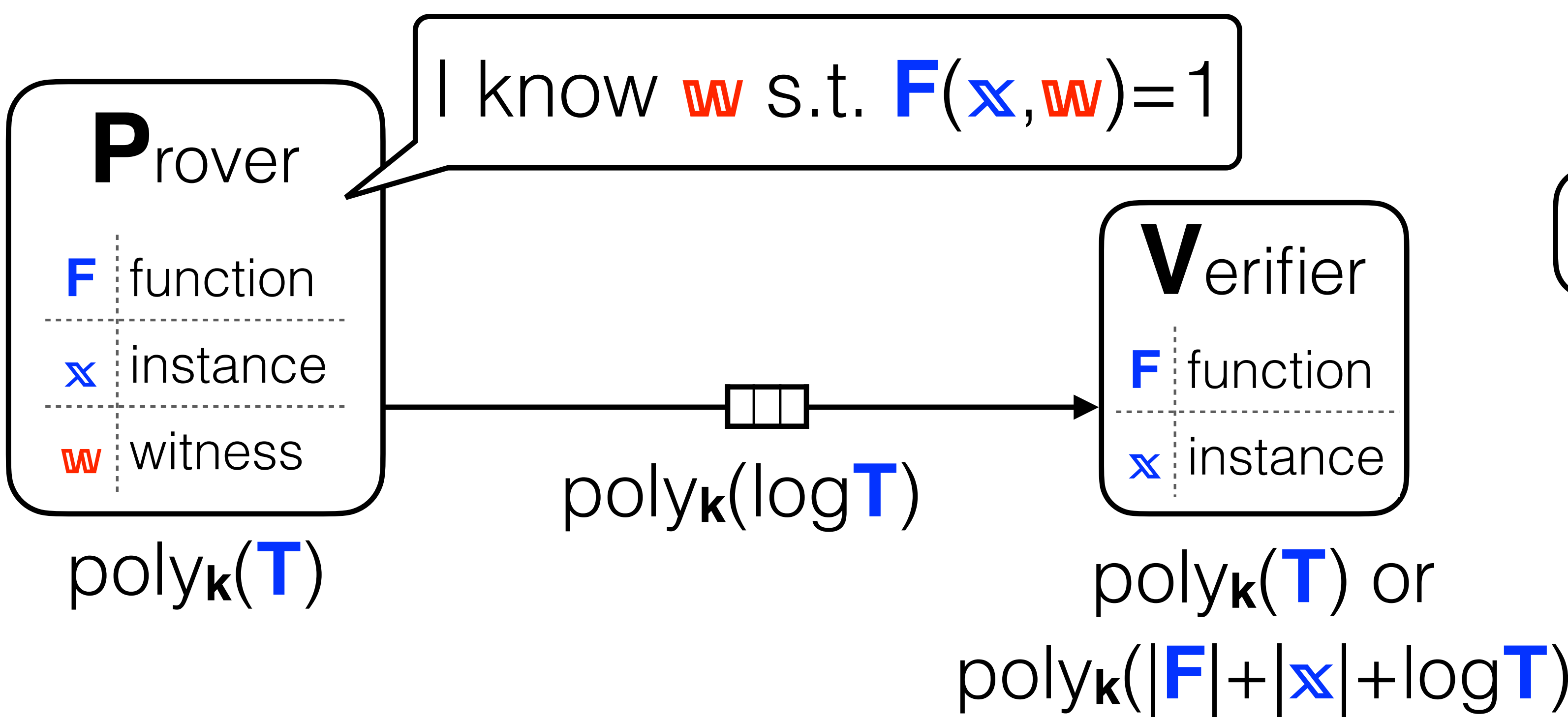
URS (uniform reference string)

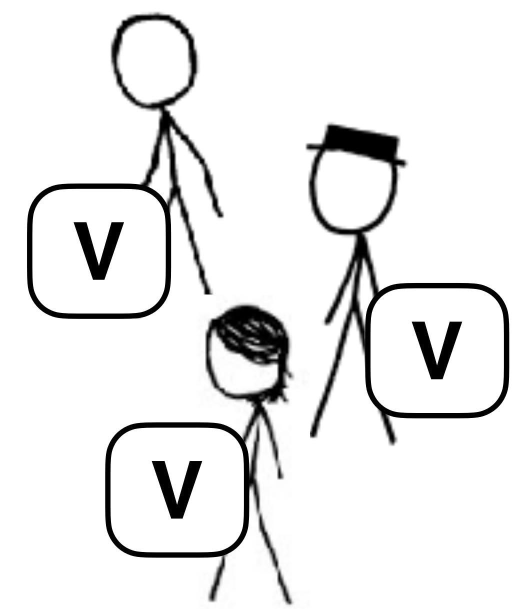
aka public/transparent setup

SRS (structured reference string)

aka private setup







SNAR**K** = SNARG of Knowledge

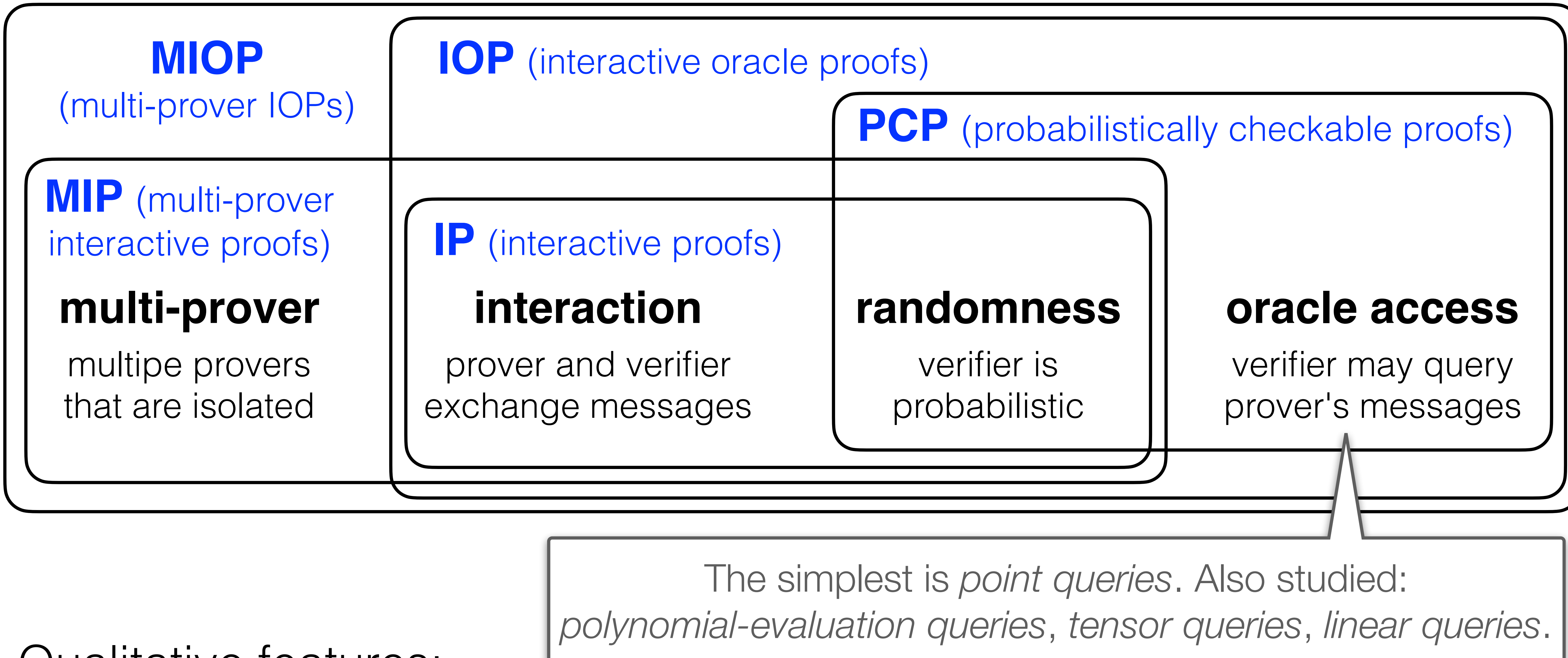
zkSNARG = Zero Knowledge SNARG

zkSNARK = ...

Origin of Succinctness: Probabilistic Proofs

Mind the Model

Different models depending on the "powers" granted to the verifier:



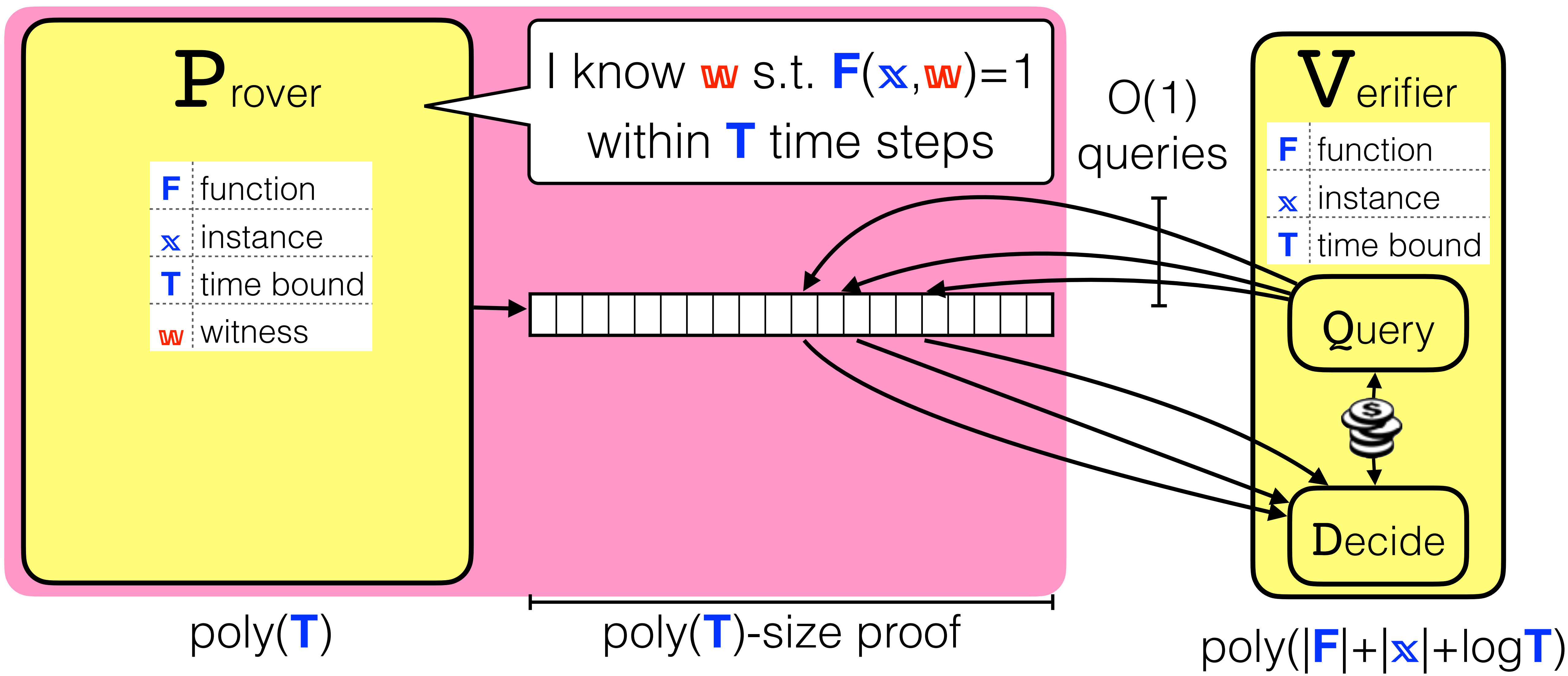
Qualitative features:

- **IP**: primarily sub-routines (e.g. sumcheck) to other probabilistic proofs
- **PCP**: pedagogically useful but mostly inefficient (e.g. with point queries)
- **MIP** (& **MIOP**): attractive features (e.g. space efficiency) but hard to use
- **IOP**: underlie most efficient SNARKs

Probabilistically Checkable Proofs

[BFLS91]
[FGLSS91]
[AS92]
[ALMSS92]

The verifier is probabilistic and has oracle access to 1 prover message:

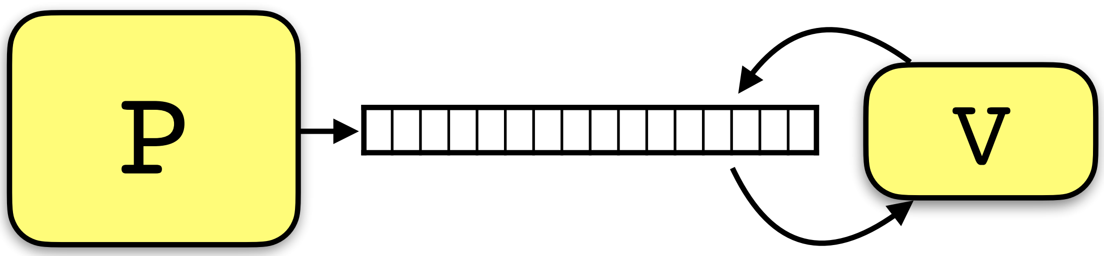


Note: PCP ≠ Succinct Argument!
It is insecure for the verifier to just ask the prover to answer a few queries.

PCPs are Inefficient

Notable Exception:
PCPs with linear queries are efficient

PCP
(probabilistically checkable proof)



Celebrated Results [BFLS91]...[BGHSV06]...

Every T-step non-deterministic computation has PCP whose prover runs in $T \cdot \mathrm{poly}(\log T)$ time and whose verifier runs in $\mathrm{poly}(\log T)$ time.

1990s - 2010: PCPs are galactic (asymptotically efficient but concretely useless)

2010 - 2013: galactic → expensive

PCP-based SNARGs where argument size is **10s of MBs**
& non-trivial for large (but not galactic) values of T

2013 - now: no PCP improvements

PCPs remain concretely inefficient, and asymptotically sub-optimal

E.g. we have no PCPs of size $O(T)$

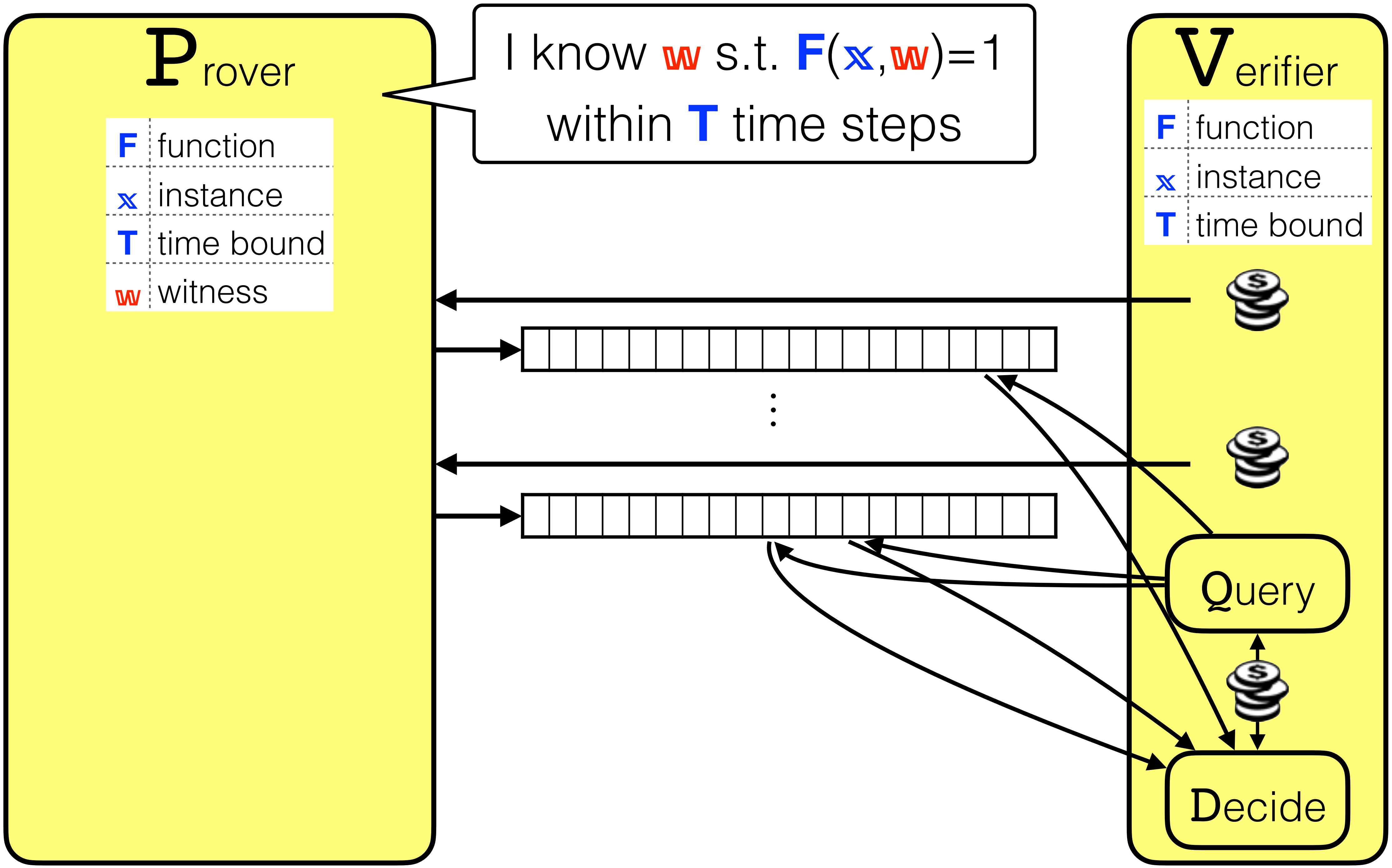
What do we do? 😭

Interactive Oracle Proofs

[BCS16]
[RRR16]

aka Probabilistically Checkable Interactive Proofs

The verifier can simultaneously leverage
randomness, **interaction**, and **oracle access**:



Constructions of IOPs

Flurry of IOP research in the past few years:

- quasilinear-time ZK [BCGV16][BCFGRS17]
- linear-size proof length [BCGRS17][RR20]
- linear-time prover [BCGGHJ17][BCG20][BCL20]
- linear-time proximity proofs [BBGR16][BBHR18][BKS18][BGKS20][BCIKS20][BN20]
- efficient implementations [BBC+16][BBHR19][BCRSVW19][COS20]

all these features are out of reach for known PCPs

Many new techniques: interactive proof composition, univariate sumcheck, out of domain sampling, algebraic linking, ...

In sum: IOPs offer much improved efficiency (asymptotically & concretely).

Probabilistic proofs (PCPs, IOPs, ...) are a **beautiful** but **technical** area.

Want to learn more?

- **Online course**: `http://bit.do/probabilistic-proofs`
- **Summer school** (w/ problem sets!): `http://bit.do/summer-proofs`

Realizing Proof Models: Cryptography

Examples of SNARK Recipes

Probabilistic Proof	+ Cryptography	→ SNARK
linear PCP (and 2-message linear IP)	linear encoding: pairings, lattices, ...	[G10][L11][BCIOP13] [GGPR13][PGHR13] ... [G16][GM17]
PCP and IOP	vector commitment: hash functions, pairings, ...	Ligero, Aurora, Fractal, SCI, STARK, ... **Nick Spooner's talk.**
polynomial PCP & IOP	polynomial commitment: PO groups, UO groups, pairings	Sonic, Marlin, Plonk, Spartan Supersonic-RSA, Hyrax, vSQL, vRAM, Libra, ... **Pratyush Mishra's talk.**

time/space efficiency is inherited from PP & Crypto

Probabilistic Proof determines ↓ type of computation (e.g., circuit vs machine)

Cryptography determines ↓

- cryptographic costs (in prover and in verifier)
- pre-quantum or post-quantum
- setup (public or private, specific or universal)

Notable Crossover: **cryptographic sumchecks** (Bulletproofs and follow-up work)

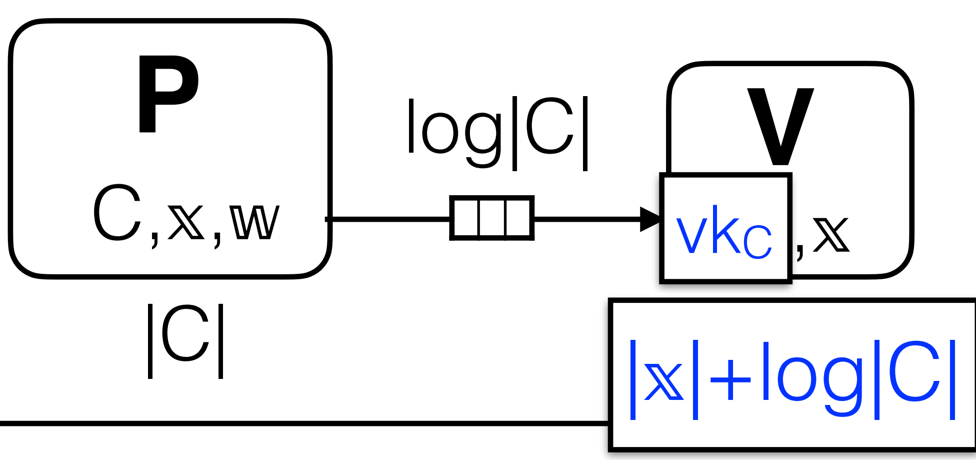
Whence Fast Verification?

Fast verification in SNARGs is achieved in different ways, depending on the *type of computation* being checked.

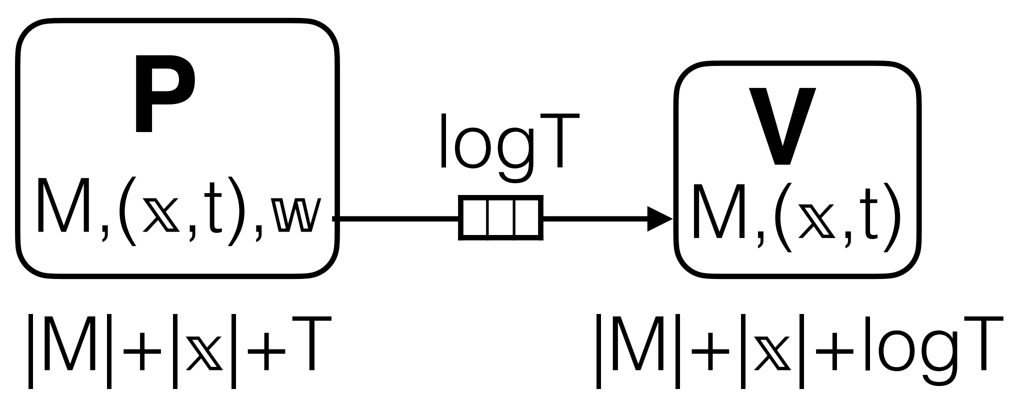
circuit computations	**machine computations**
$\mathrm{CSAT} = \{ (C,\mathbb{x},\mathbb{w}) : C(\mathbb{x},\mathbb{w})=0 \}$	$\mathrm{MSAT} = \{ (M,(\mathbb{x},T),\mathbb{w}) : M(\mathbb{x},\mathbb{w})=0 \text{ in } T \text{ steps} \}$
$\lvert C\rvert+\lvert\mathbb{x}\rvert \sim \lvert C\rvert$	$\lvert M\rvert+\lvert\mathbb{x}\rvert+\log T \ll \lvert M\rvert\cdot T$

preprocessing

SNARG for CSAT



SNARG for MSAT



all computations **with** preprocessing

some computations **without** preprocessing

Frontend vs. Backend

More about this in Alex Ozdemir's talk.

Frontends:

reductions from "high-level" languages to "low-level" languages

explicit reductions

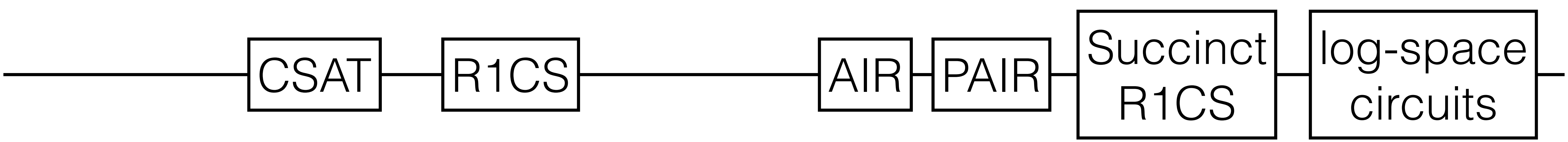
TinyRAM to circuits
gadgetlib
bellman
Pantry
ZoKrates
xJsnark
Circom
arkworks
snarky
...

succinct reductions

AirScript
Cairo
...

NP-complete

PSPACE/NEXP-complete



Backends:

argument systems for "low-level" languages

Thanks!