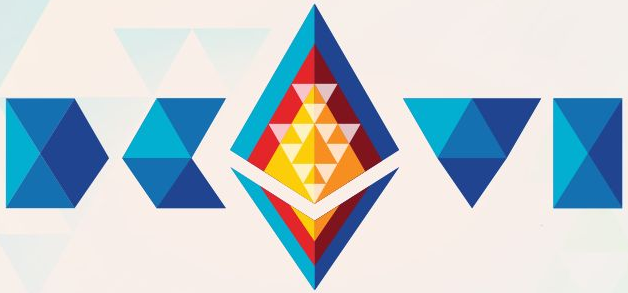


recursive proofs:

applications and affordances

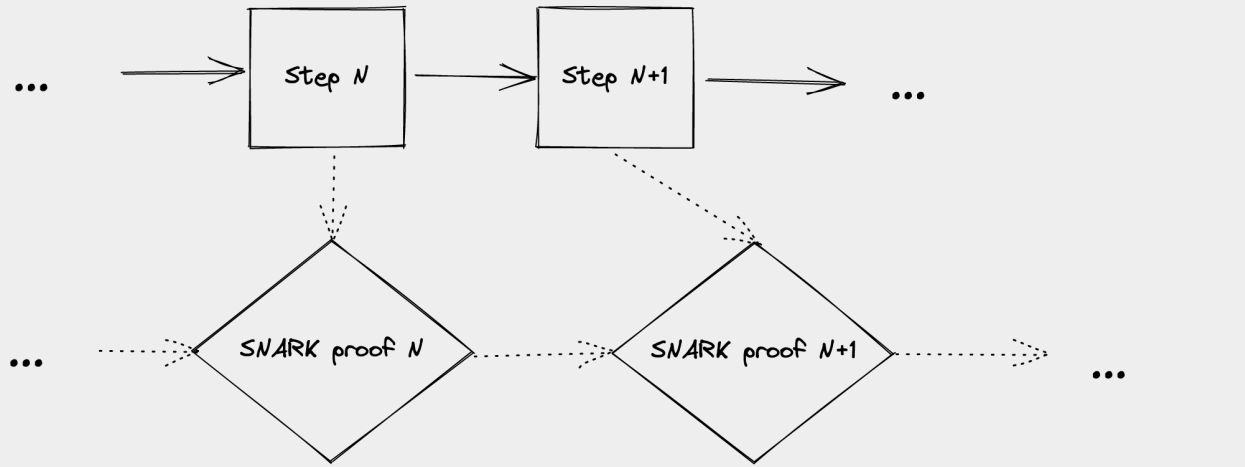
Ying Tong

Core Engineer, Electric Coin Company

Nalin

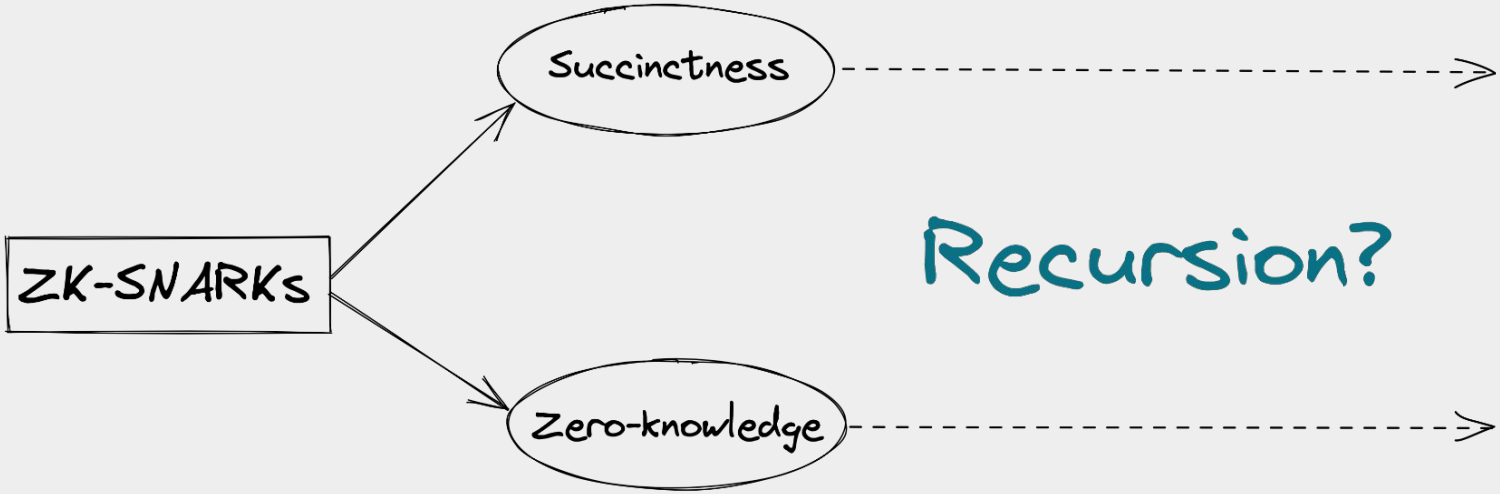
Person, 0xPARC

what are recursive proofs?

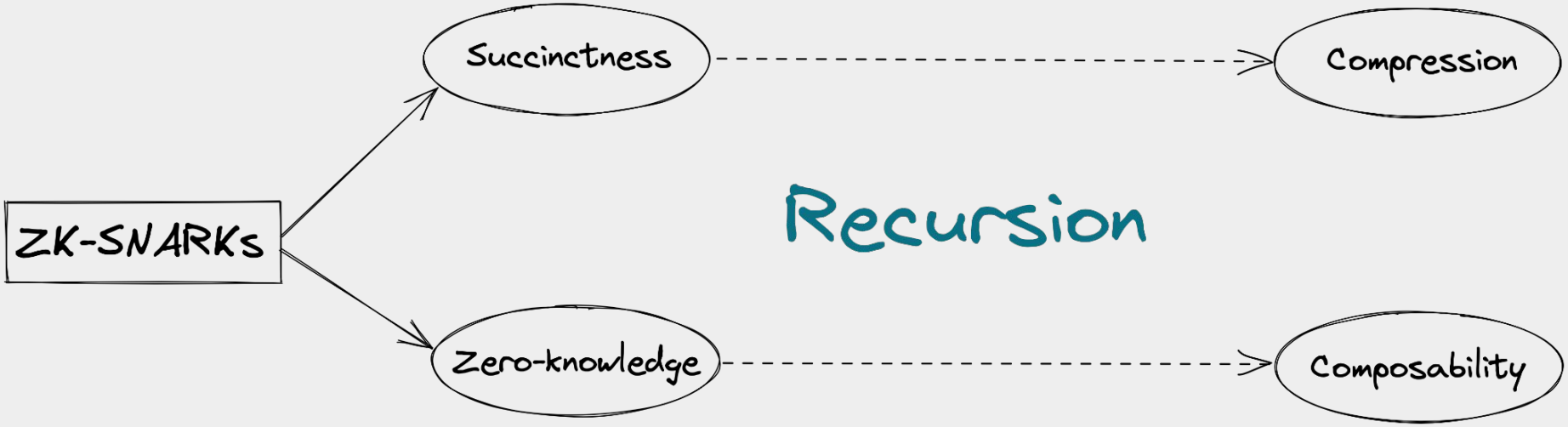


Verify SNARK proof N and Step N+1
(instead of verifying all N steps again)

unlocks from recursion



unlocks from recursion



Section 1

compression

(supercharging succinctness)

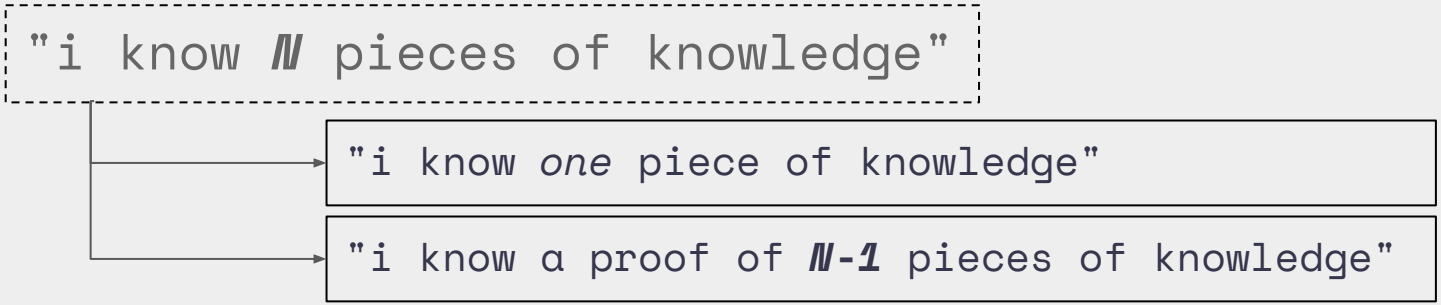
compression: supercharging succinctness

for a low verifier cost, a prover can show:

"i know N pieces of knowledge"

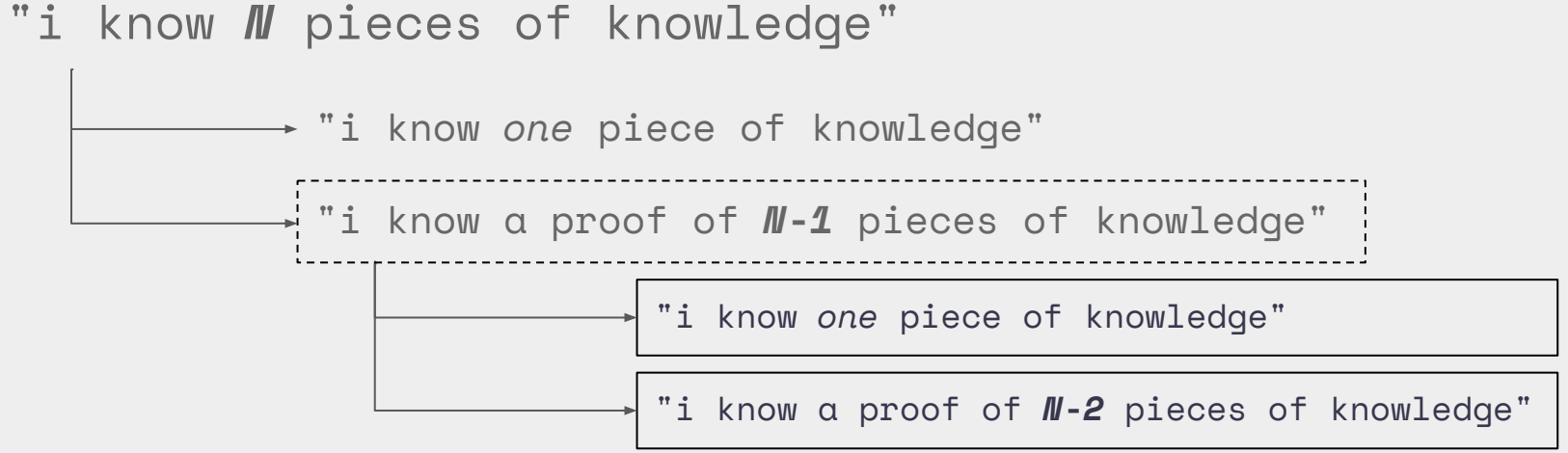
compression: supercharging succinctness

for a low verifier cost, a prover can show



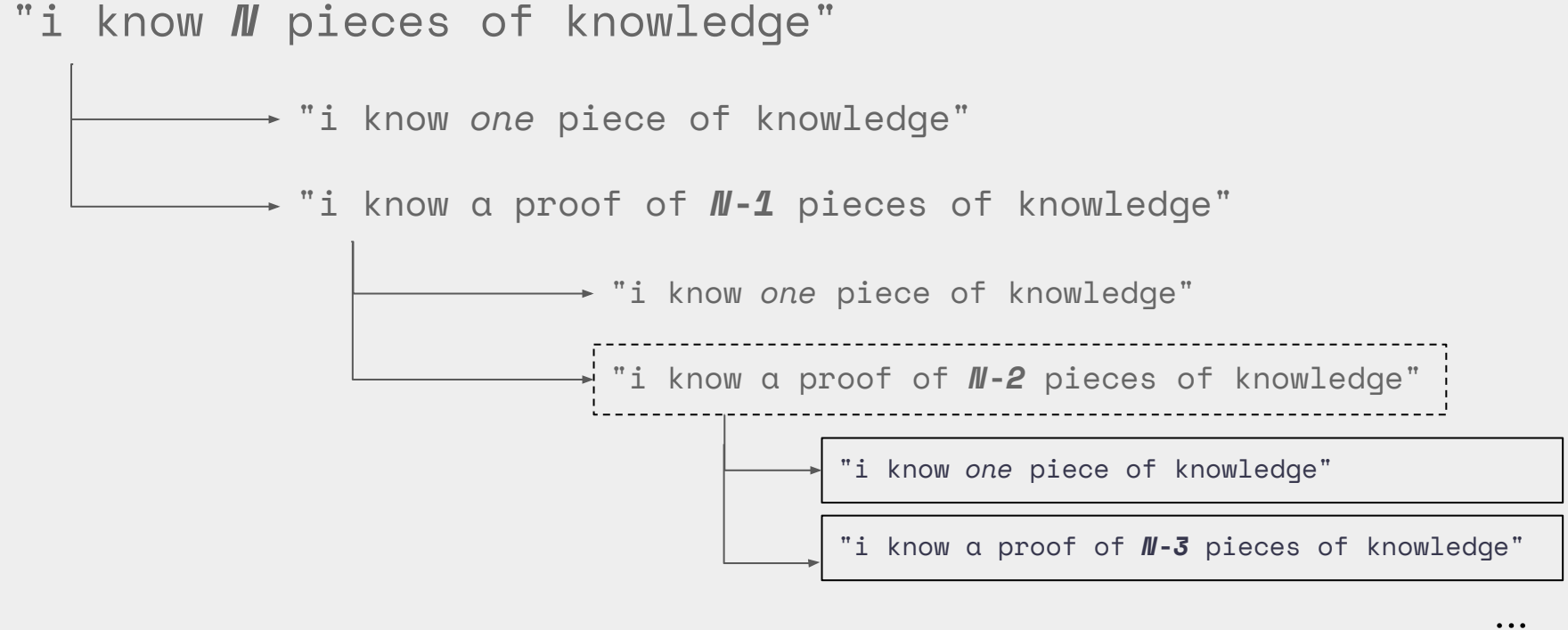
compression: supercharging succinctness

for a low verifier cost, a prover can show



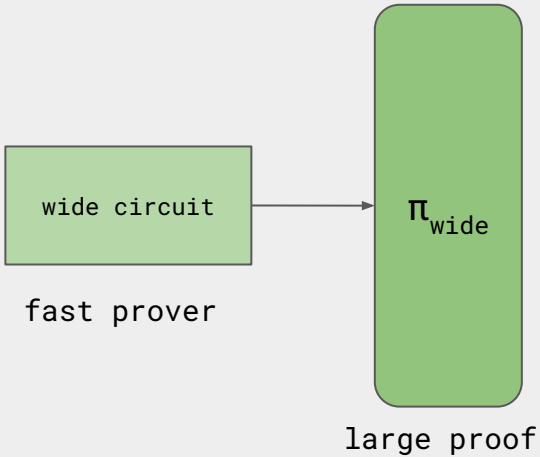
compression: supercharging succinctness

for a low verifier cost, a prover can show



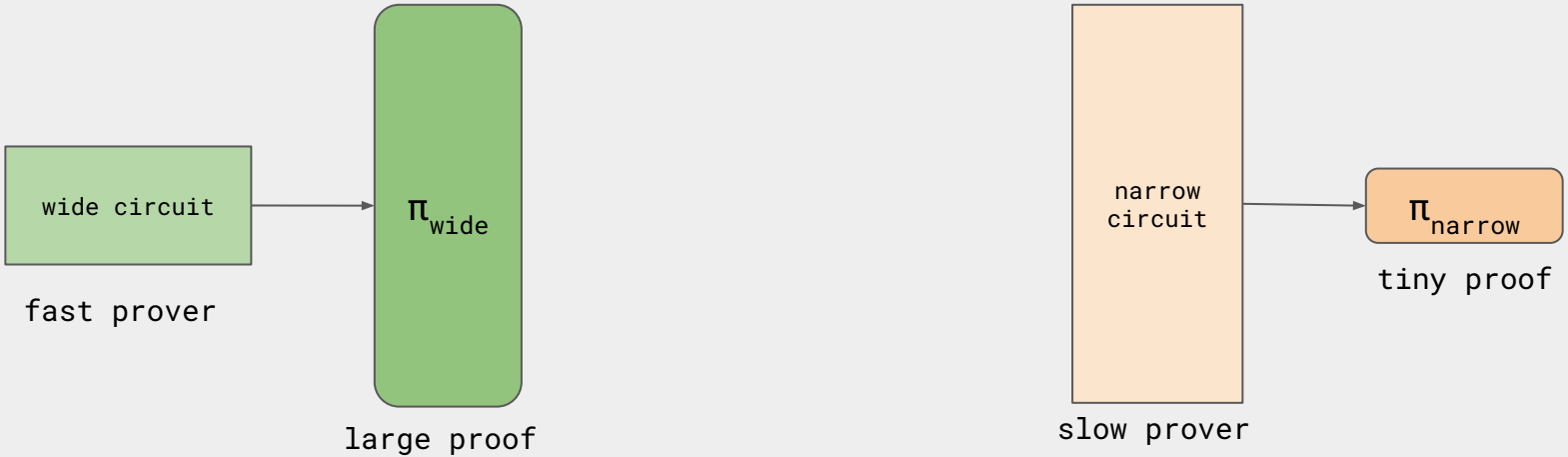
prover / verifier complexity tradeoffs 🍰

	fast prover	small proof / fast verifier
"wide" proof	✔	❌



prover / verifier complexity tradeoffs 🍰

	fast prover	small proof / fast verifier
"wide" proof	✔	❌
"narrow" proof	❌	✔

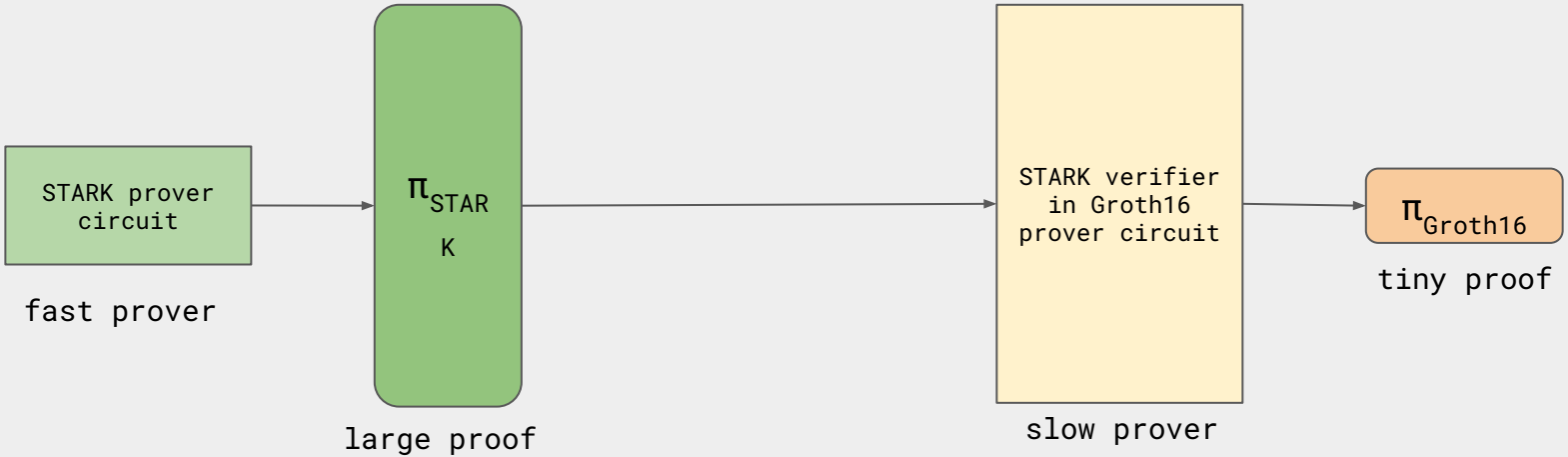


prover / verifier complexity tradeoffs 🍰

	fast prover	small proof / fast verifier
"wide" proof	✔	❌
"narrow" proof	❌	✔

interoperability between proof systems

	fast prover	small proof / fast verifier
STARK	✔	❌
Groth16	❌	✔



compression: examples

a rollup of...

Signatures

Plumo: An Ultralight Blockchain Client

AXIOM

light client proofs

MINA

polygon

Hermez

txes

Section 2

composability

(taking zk a step further)

composability

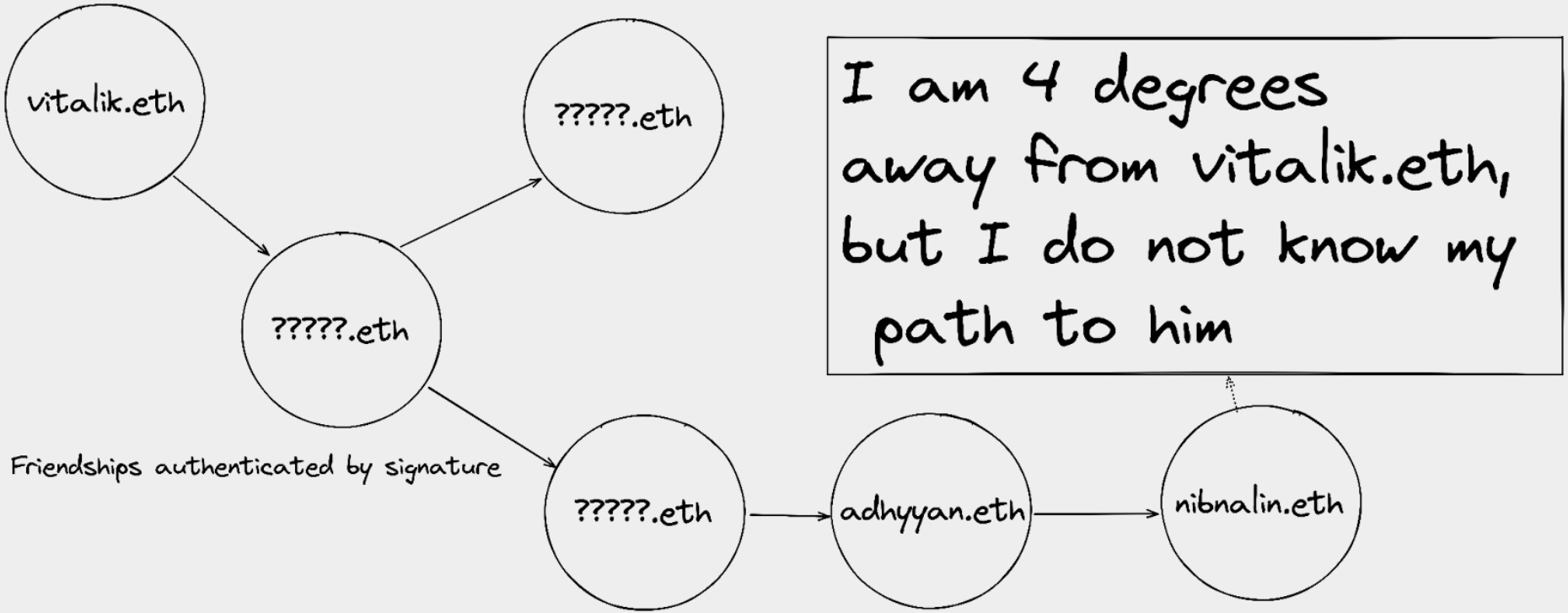
typically, zk proofs are thought of in the context:

> *"a prover shows knowledge to a verifier,* ***without revealing the underlying fact.****"*

recursive zk proofs, in fact, unlock a stronger property:

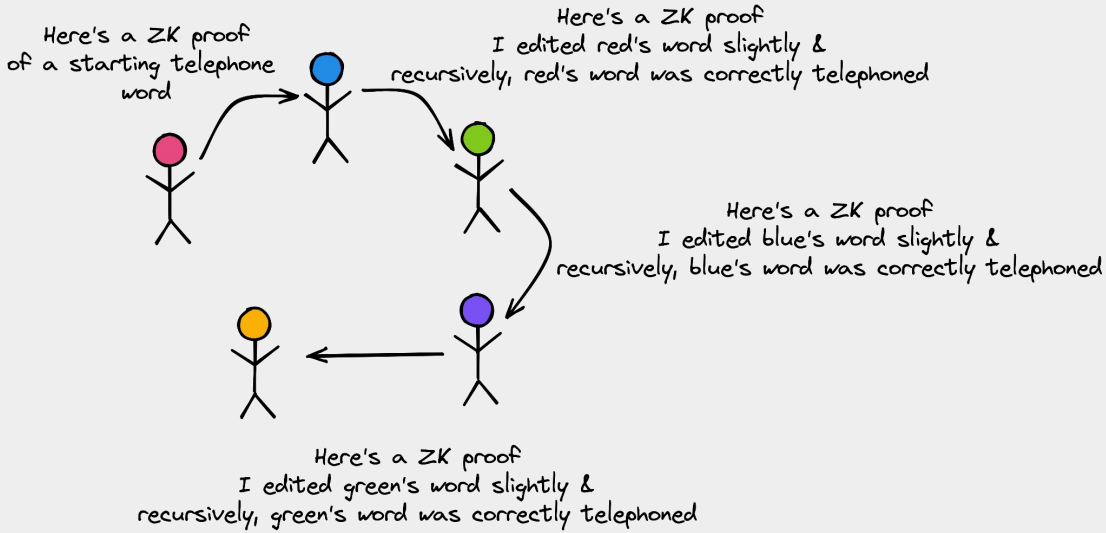
> *"a prover shows knowledge to a verifier,* ***without fully knowing the underlying facts themselves****"*

ETHdos: Erdős numbers on social graphs



composability: incomplete information games

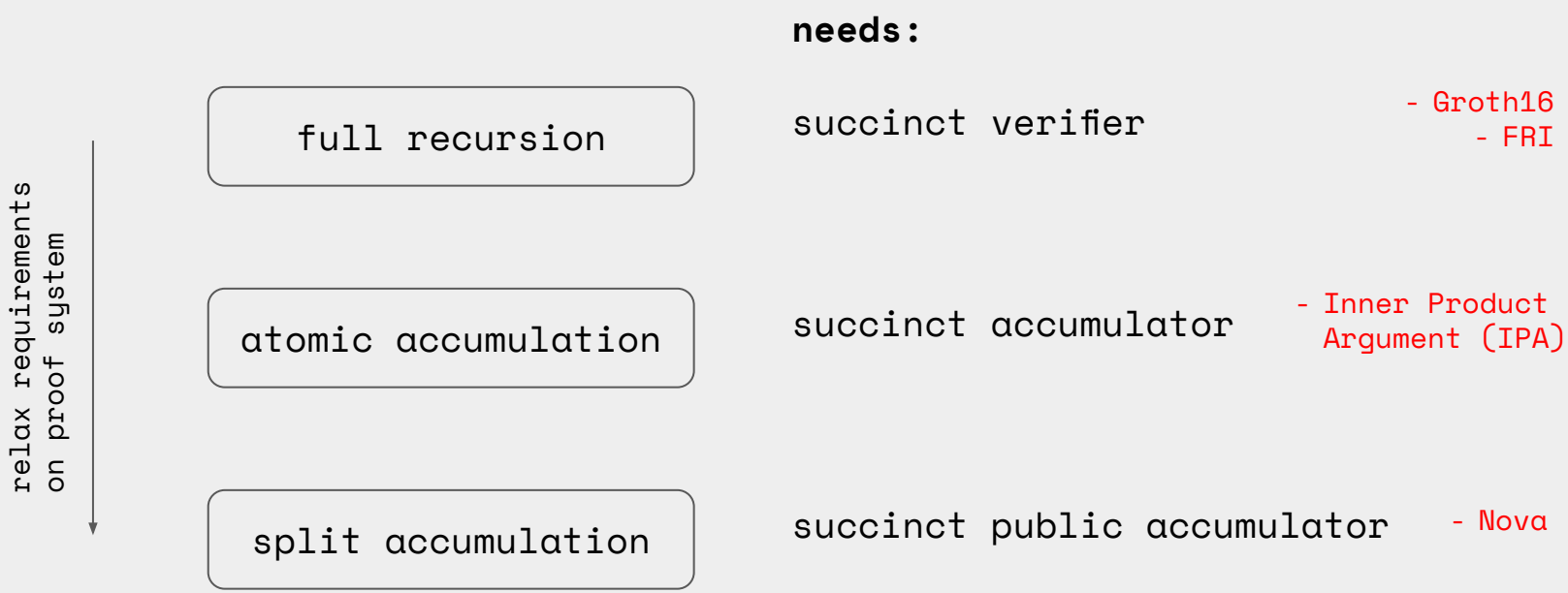
- telephone/chinese whispers
- mafia
- private state channels



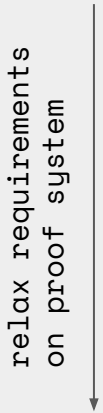
Section 3

implementations

schemes for recursive proof composition



schemes for recursive proof composition



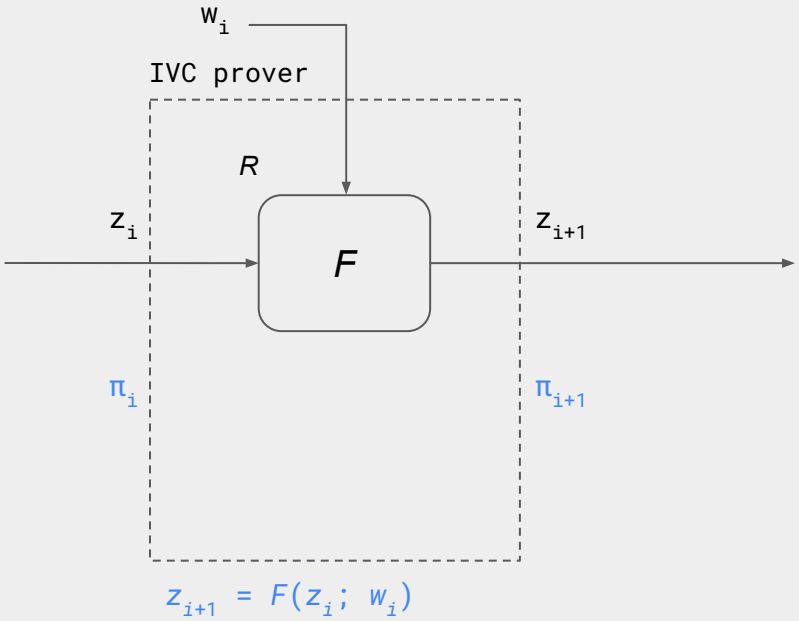
full recursion

needs:

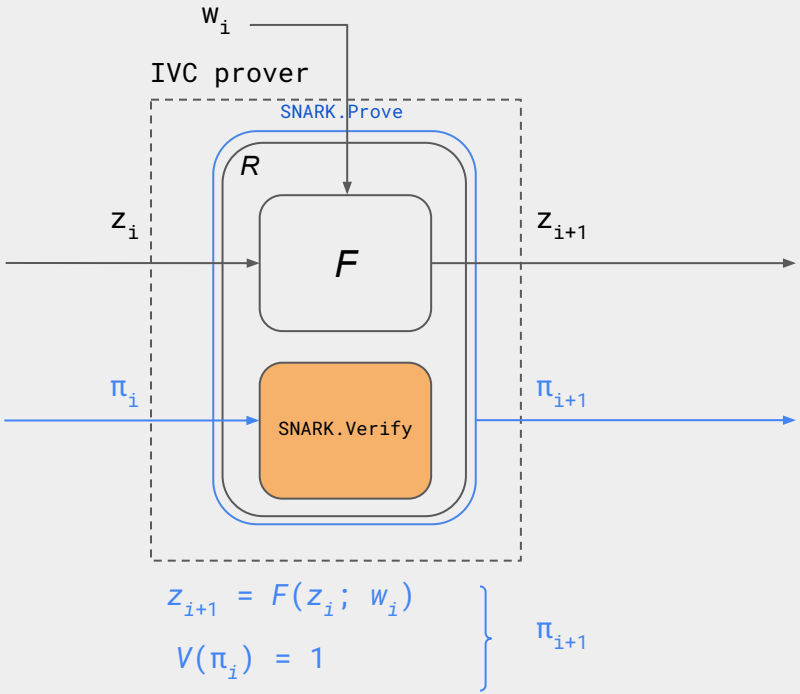
succinct verifier

- Groth16
- FRI

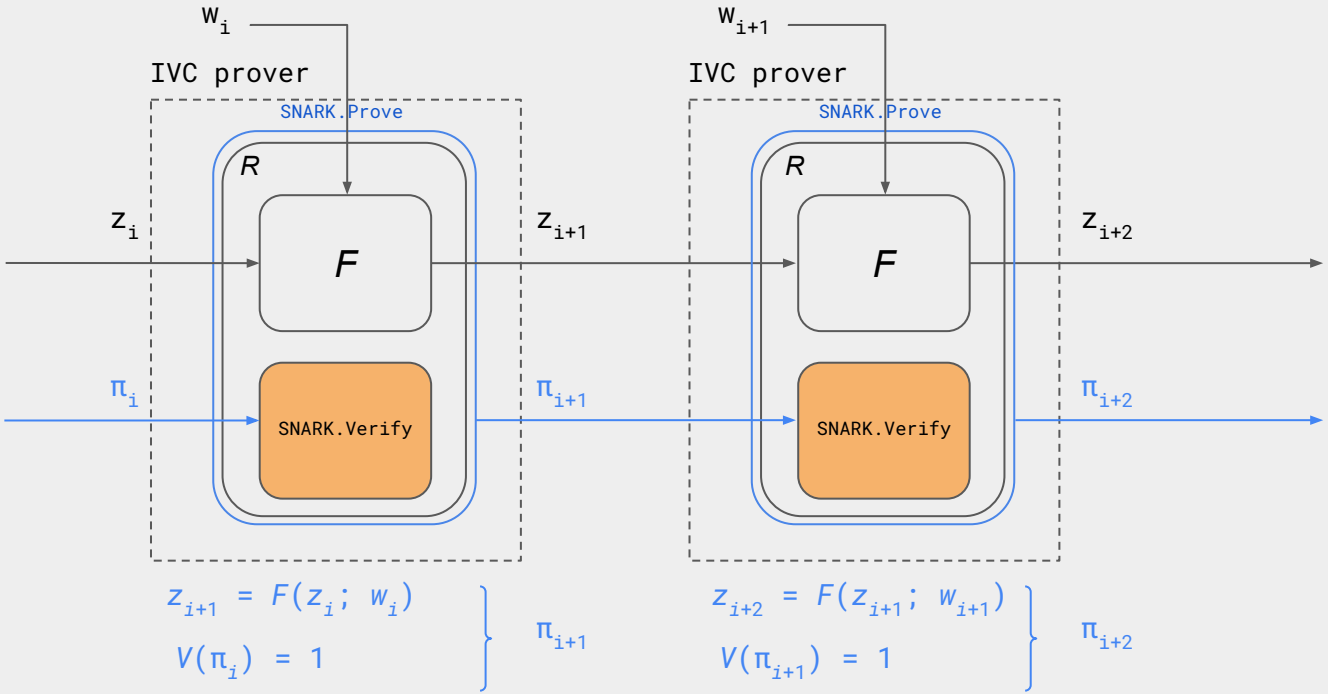
IVC from SNARKs with sublinear verification



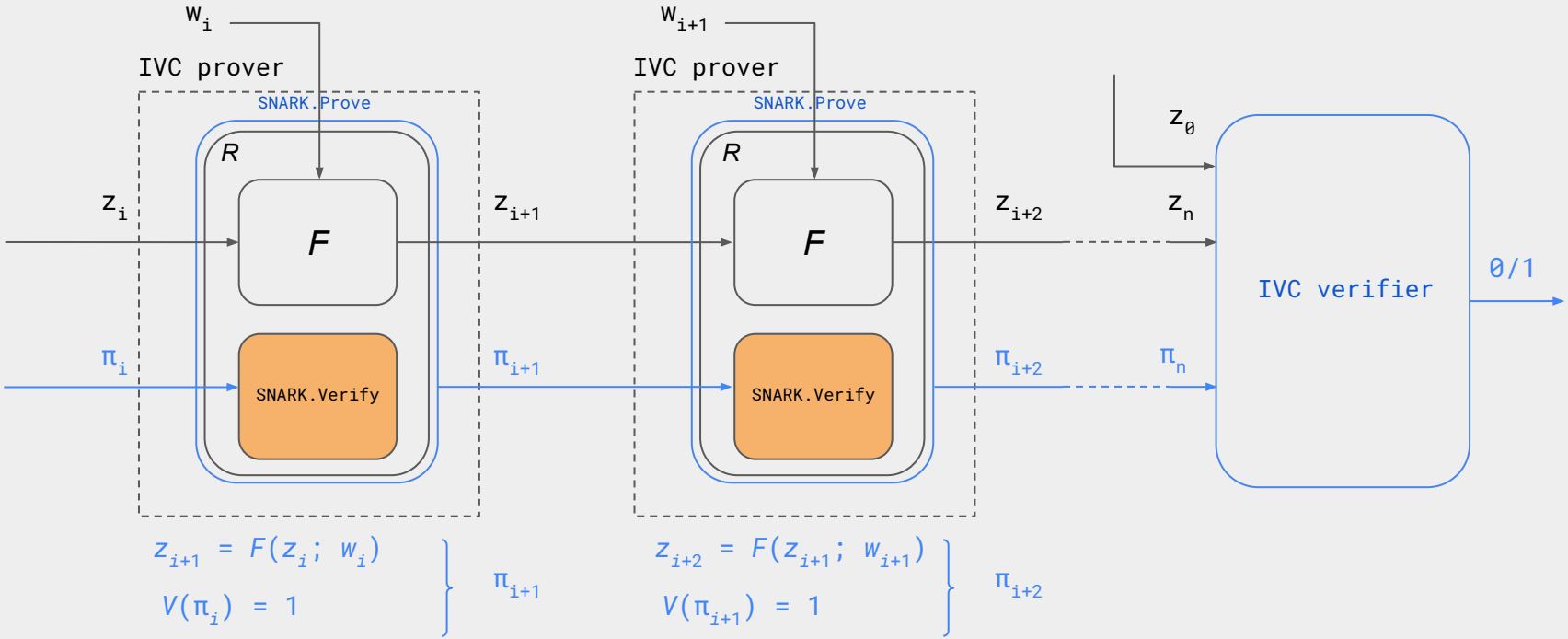
IVC from SNARKs with sublinear verification



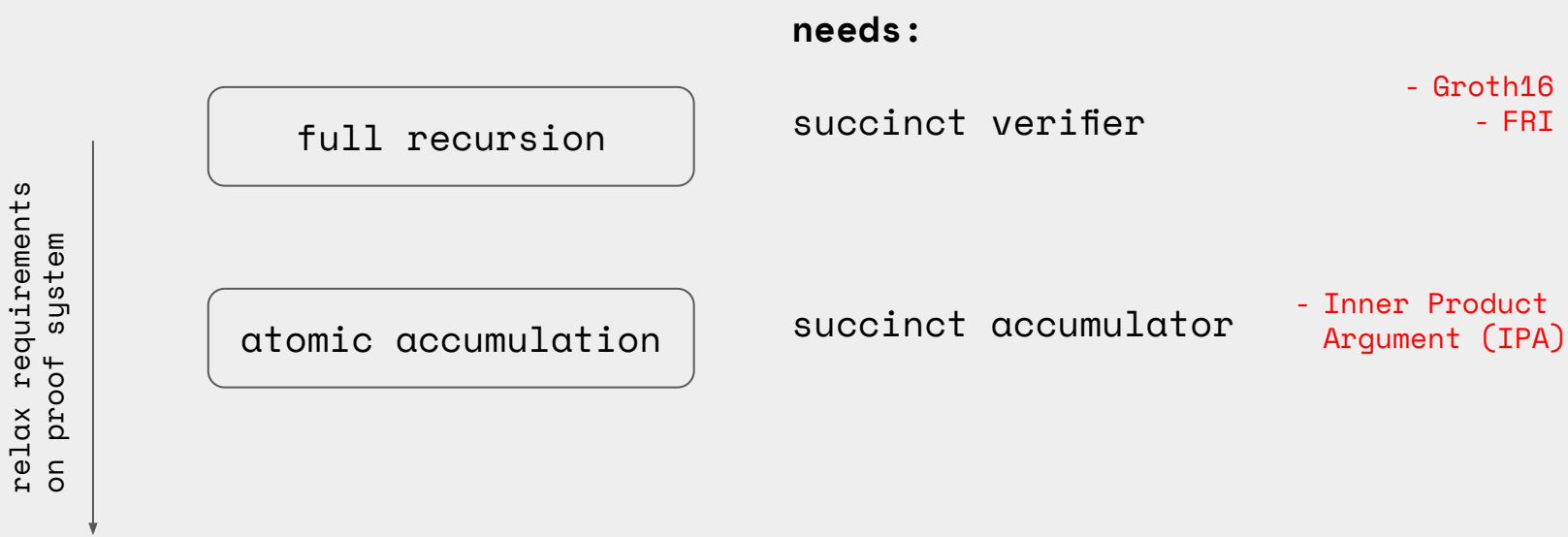
IVC from SNARKs with sublinear verification



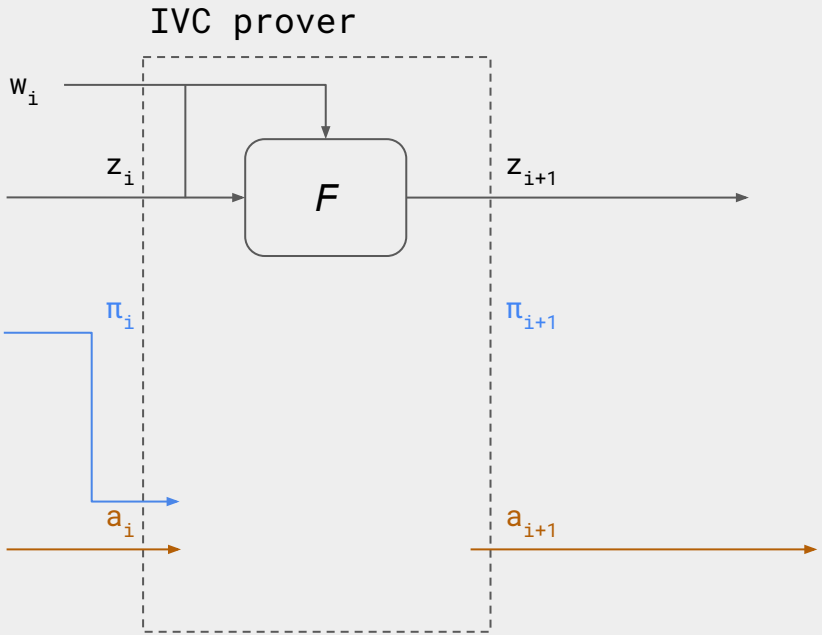
IVC from SNARKs with sublinear verification



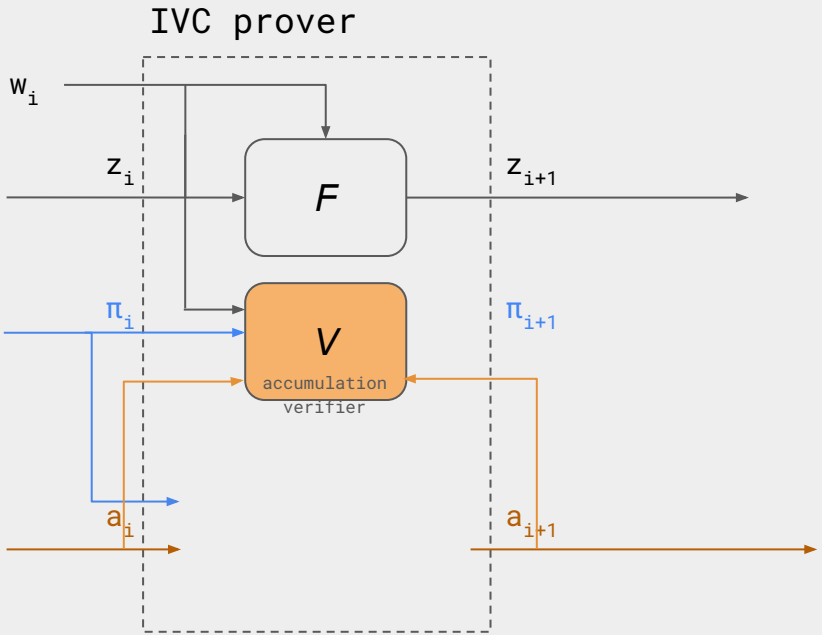
schemes for recursive proof composition



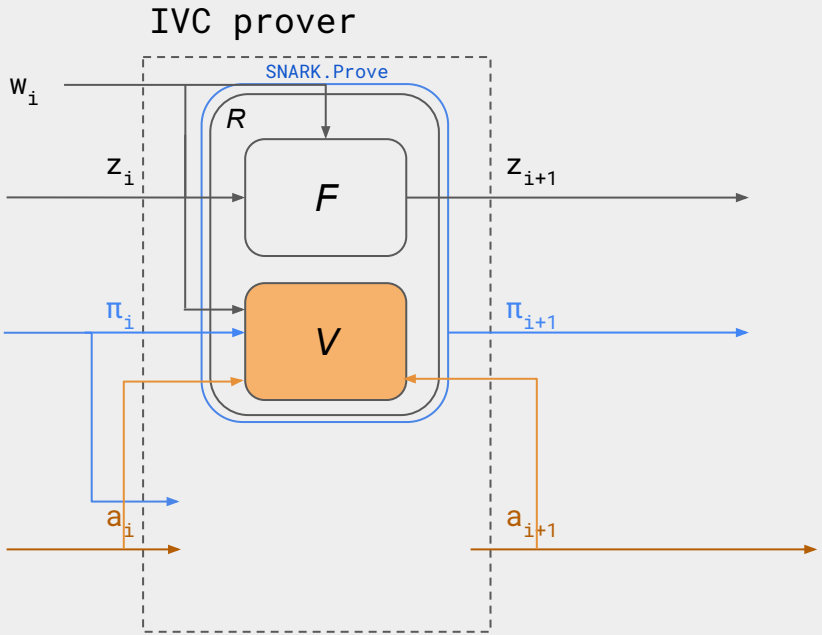
IVC from atomic accumulation



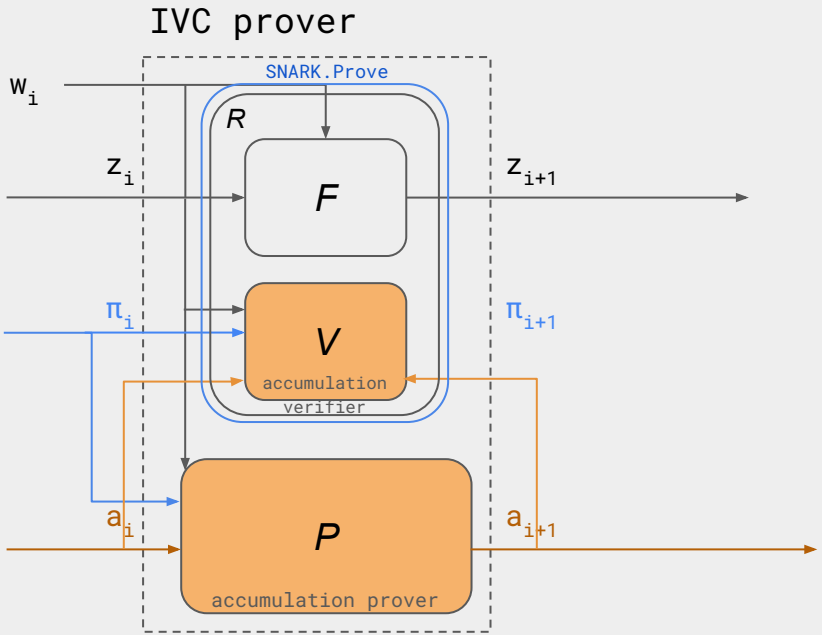
IVC from atomic accumulation



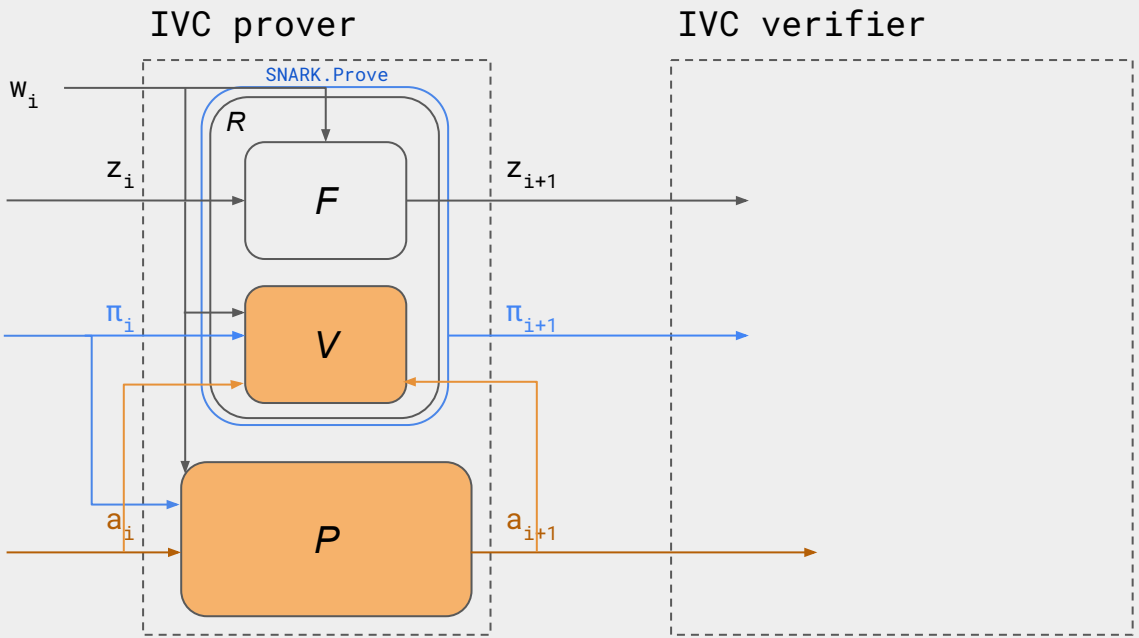
IVC from atomic accumulation



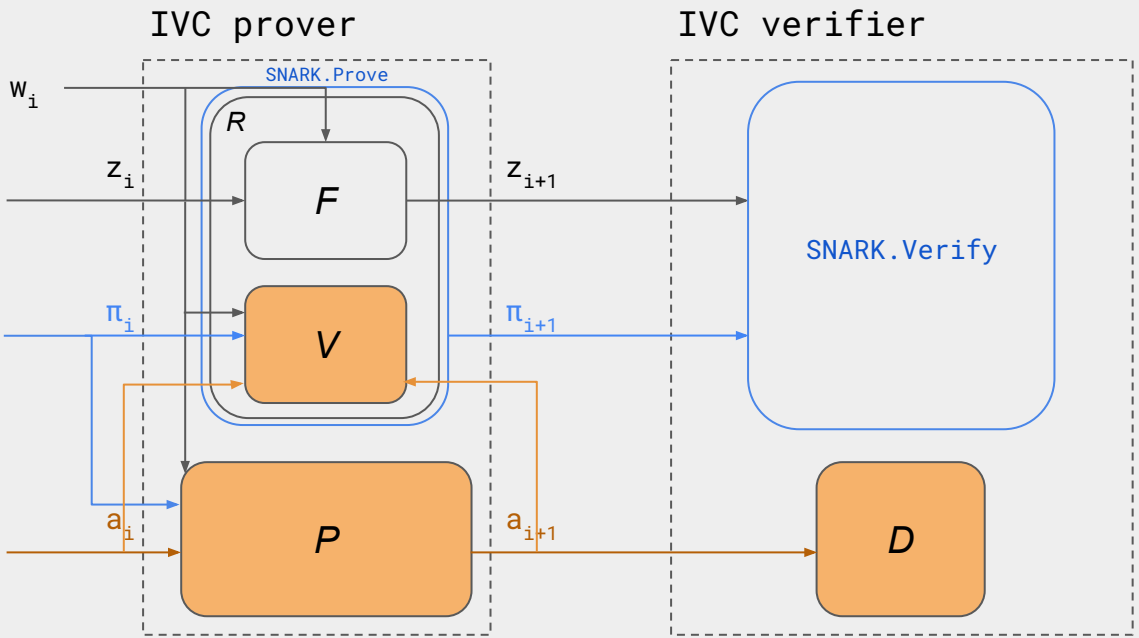
IVC from atomic accumulation



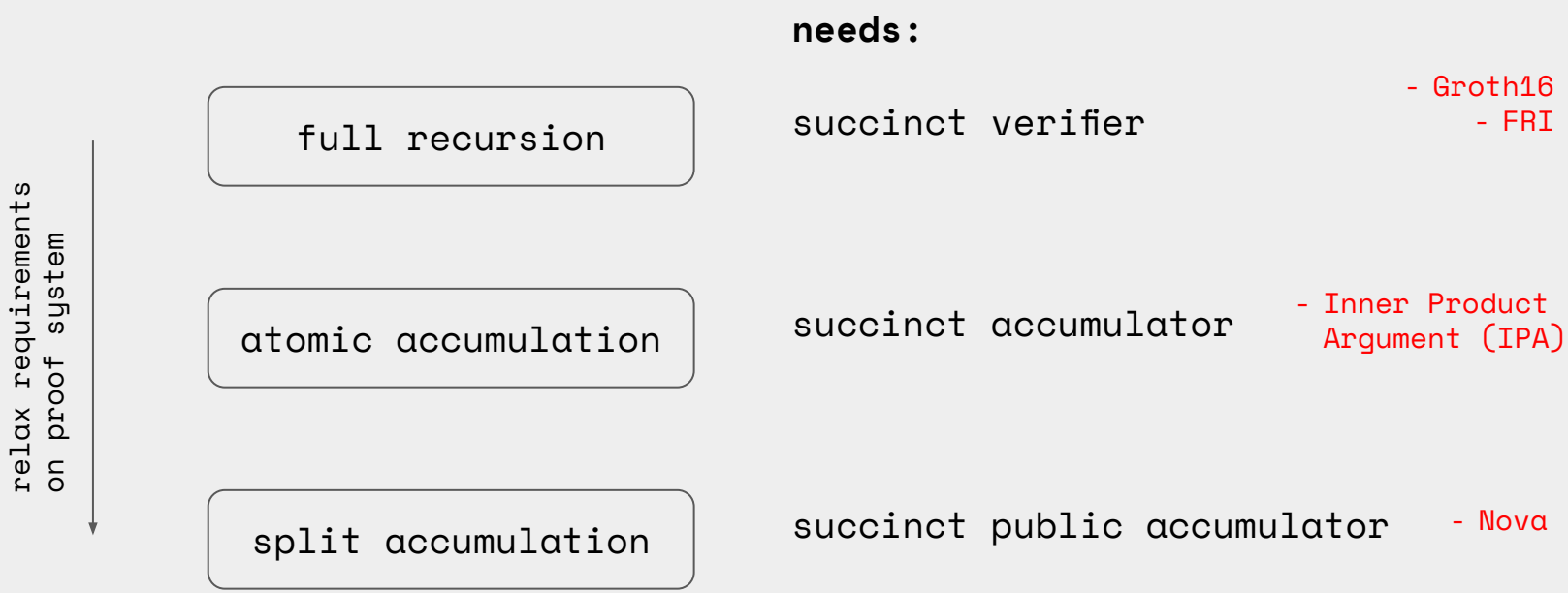
IVC from atomic accumulation



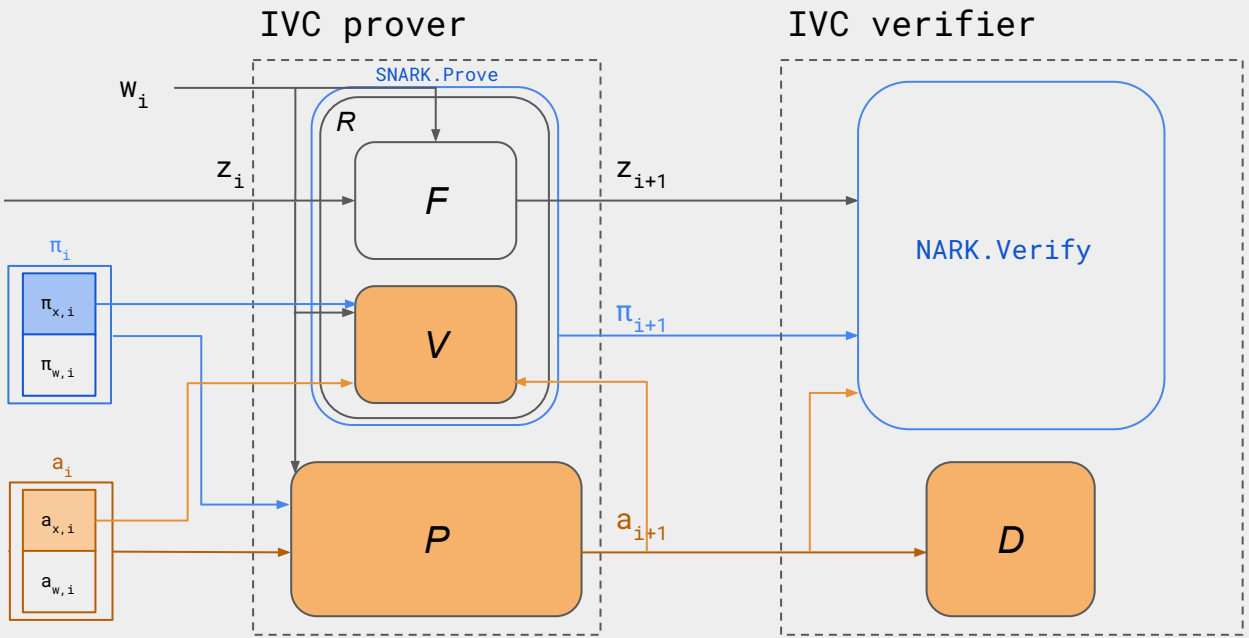
IVC from atomic accumulation



schemes for recursive proof composition



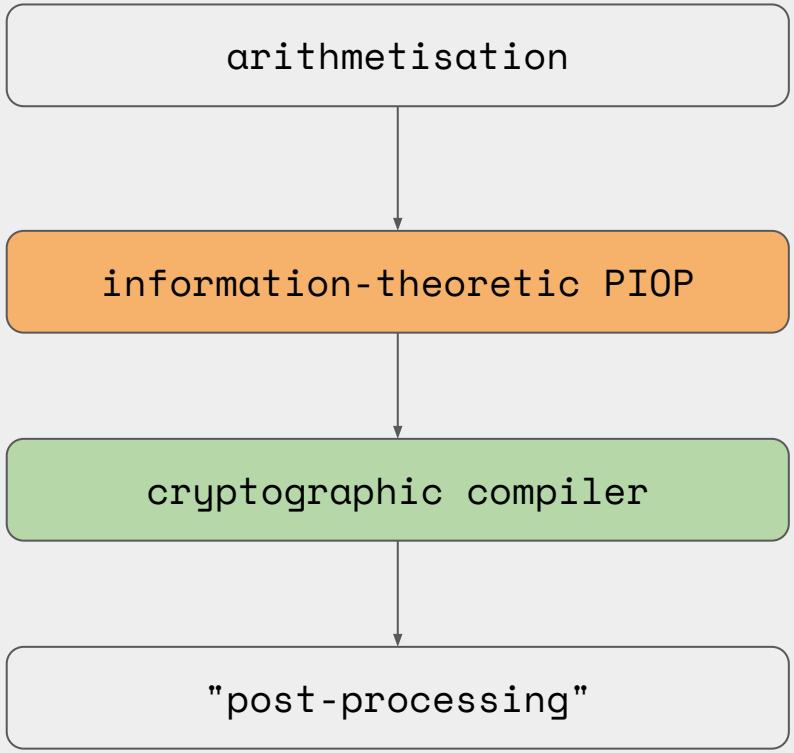
IVC from split accumulation [BCLMS20]



future of proving systems

- modular design → customisable proving stack
- recursion → composition across proof systems

modular proof systems: Halo 2



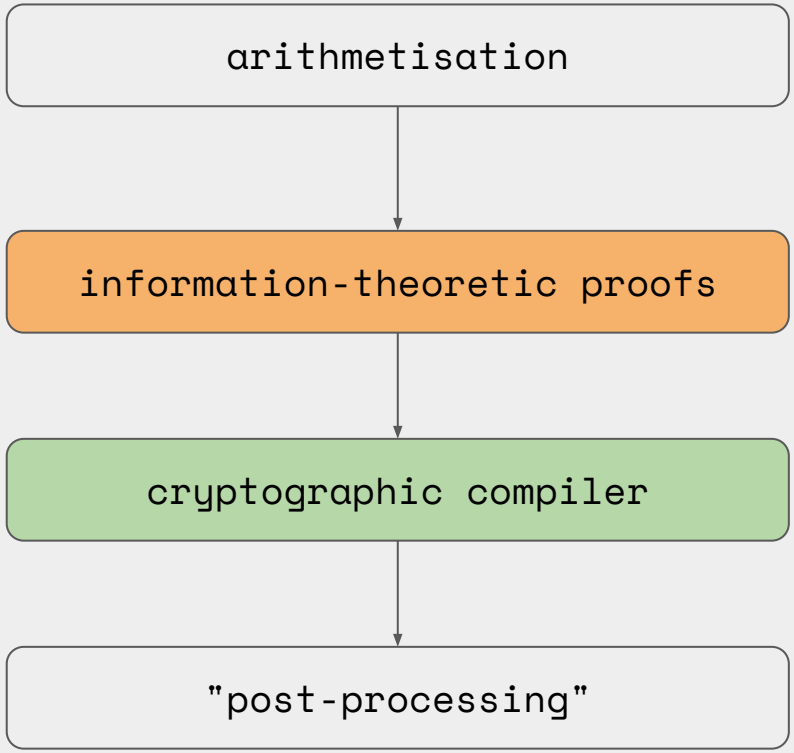
encode values in the Lagrange basis;
constraints as polynomial identities

vanishing argument;
multipoint opening argument

inner product argument;
Fiat-Shamir transform

atomic accumulation

modular proof systems



turns a relation into a **constraint system** involving native operations over a finite field

provides soundness and zk guarantees even when prover and verifier are **computationally unbounded**

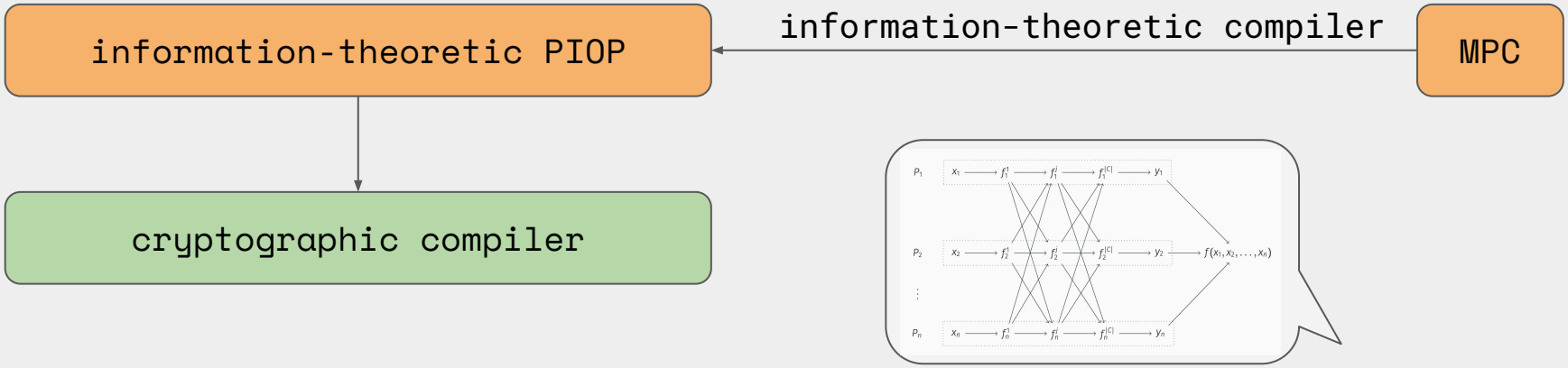
transforms proof system into concrete protocol involving **direct interaction** between prover and verifier

compositional schemes for the protocol (e.g. distributed proving, streaming prover, aggregation, accumulation, recursion)

types of proof composition

1. information-theoretic compilers

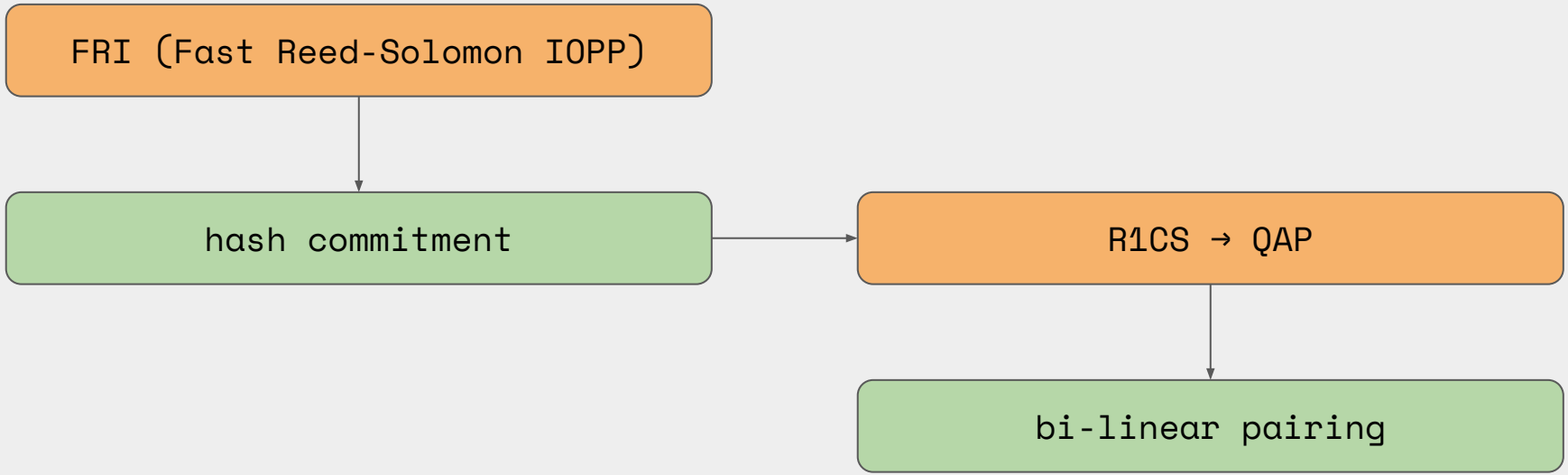
e.g. "MPC-in-the-head" [IKOS07]



types of proof composition

2. composing cryptographic compilers

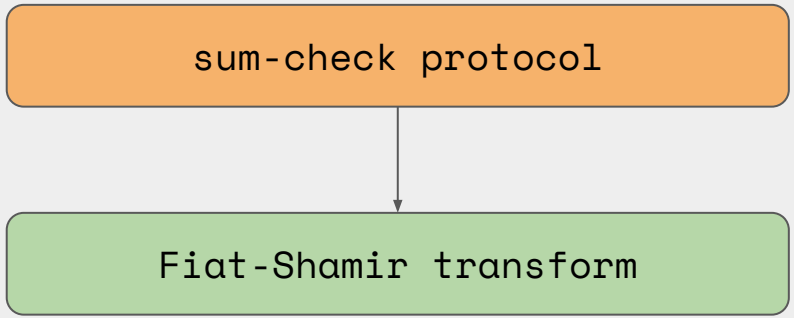
e.g. STARK verifier in Groth16 prover



types of proof composition

3. tailor-made cryptographic compilers

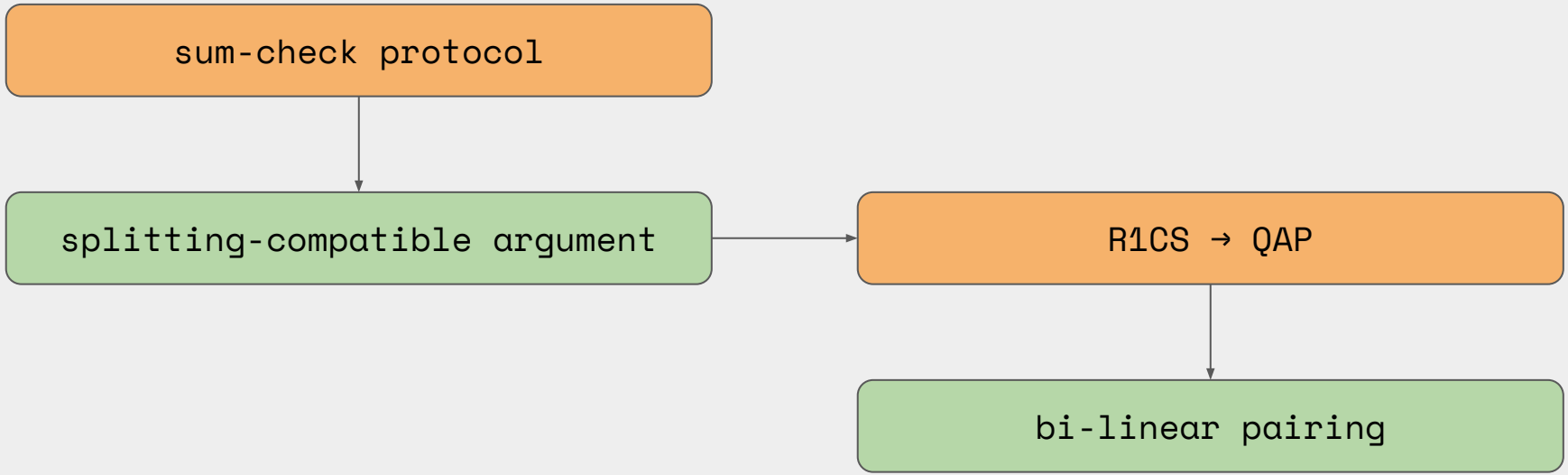
e.g. GKR verifier in Groth16 prover [BSB22]



types of proof composition

3. tailor-made cryptographic compilers

e.g. GKR verifier in Groth16 prover [BSB22]



future of proving systems

- modular design → customisable proving stack
- recursion → composition across proof systems

can we systematise the composition of proof protocols?

future of proving systems

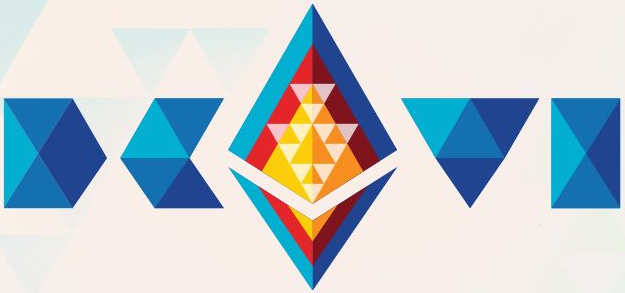
- modular design → customisable proving stack
- recursion → composition across proof systems

can we systematise the composition of proof protocols?

- better benchmarks for primitives (e.g. hashes, bigint, signatures, range proofs, …)
- standardised criteria for comparing different compositions
- auditing / formal verification when composing proof systems

Section 4

recursion, aggregation, composition task force



Thank you!

Ying Tong
Core Engineer, Electric Coin Company
yingtong@z.cash
@therealyingtong

Nalin
Person, 0xPARC
devcon@nibnalin.me
@nibnalin