HyperNova

Recursive Arguments for Customizable Constraint Systems

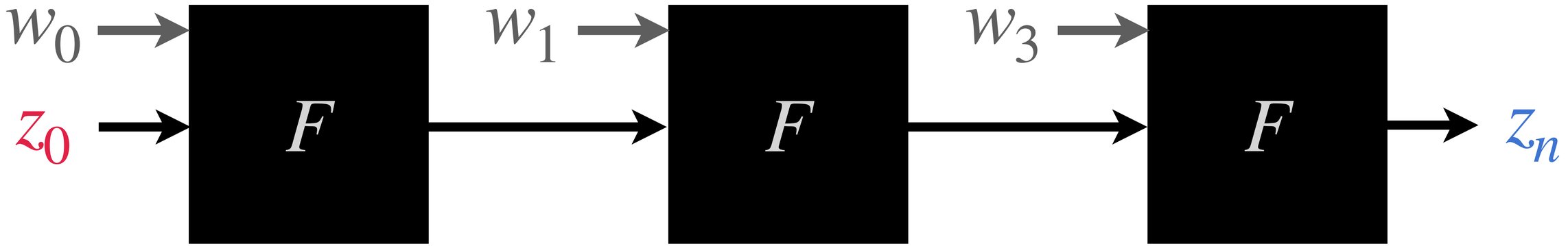
Abhiram Kothapalli CMU
Srinath Setty MSR

ia.cr/2023/573

Crypto 2024, UCSB

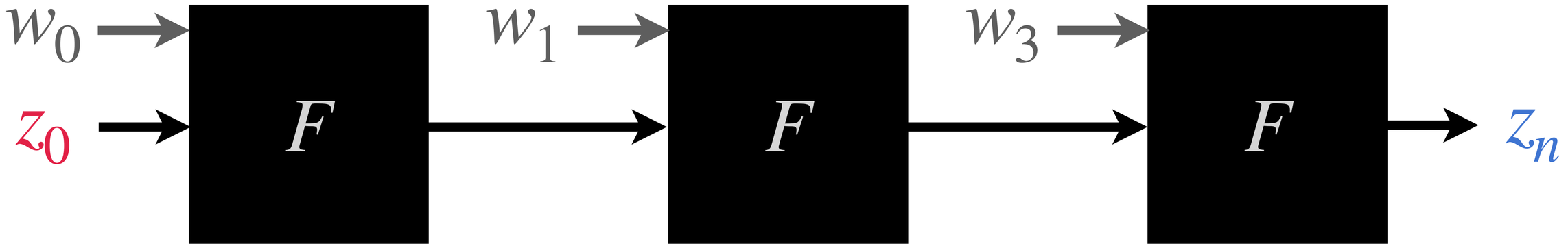
Goal: Practical zkSNARKs for Recursive Computation

Prove that (non-deterministic) function F applied n times to initial input z_0 results in z_n



Goal: Practical zkSNARKs for Recursive Computation

Prove that (non-deterministic) function F applied n times to initial input z_0 results in z_n

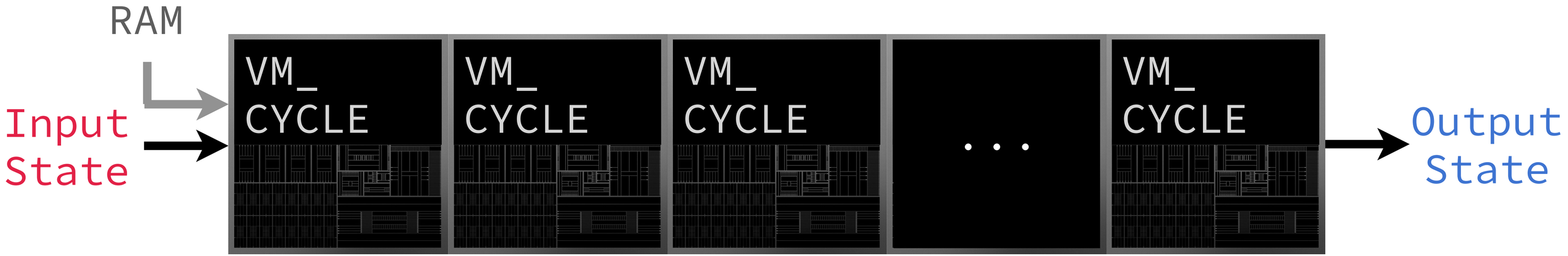


Example Applications

- Verifiable Delay Function: Let F be a delay function [BBBF19]
- ZK-rollups: Let F validate new blockchain transactions

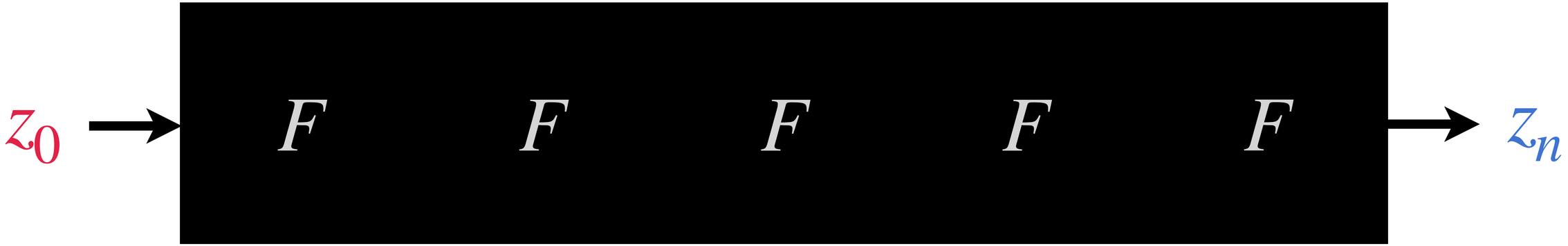
Converging Application: Zero-Knowledge Virtual Machines

Proofs for recursive computations imply proofs for VM execution.



Direct Approach: Memory-Bound [BCTV13,WSHRBW15,AST23]

Use a SNARK to check the entire unrolled computation trace.

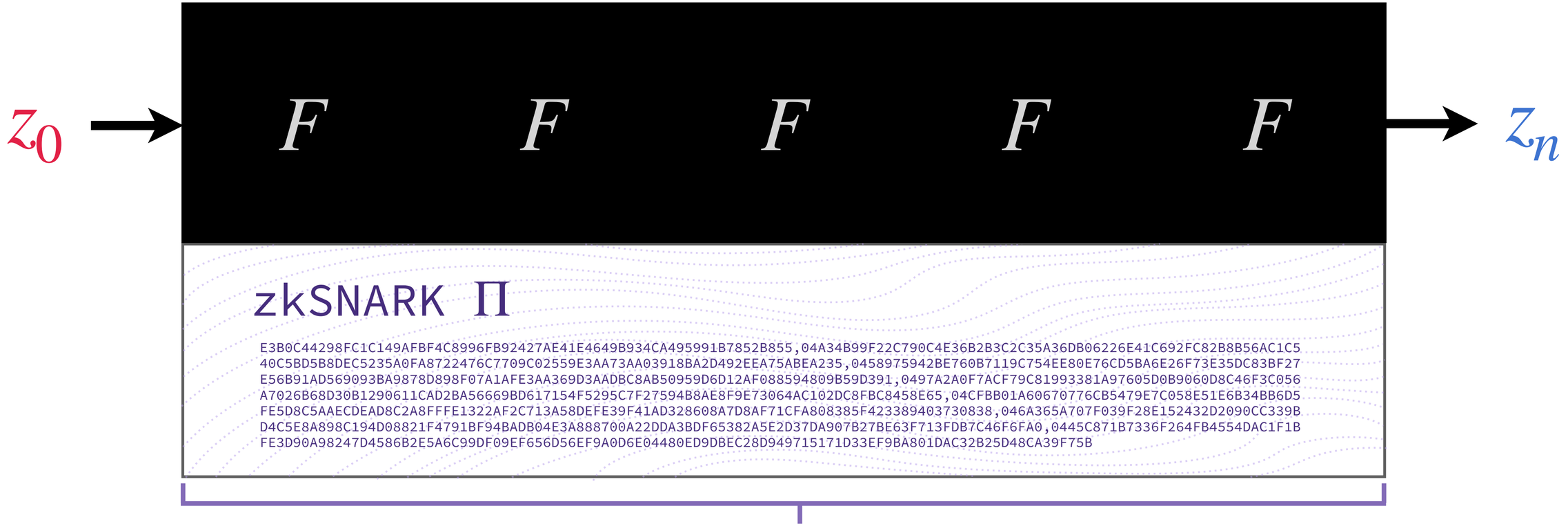


Direct Approach: Memory-Bound [BCTV13,WSHRBW15,AST23]

Use a SNARK to check the entire unrolled computation trace.

Direct Approach: Memory-Bound [BCTV13,WSHRBW15,AST23]

Use a SNARK to check the entire unrolled computation trace.



Jolt [AST23] proves 17M RISC-V cycles in 125 seconds using 512GB.

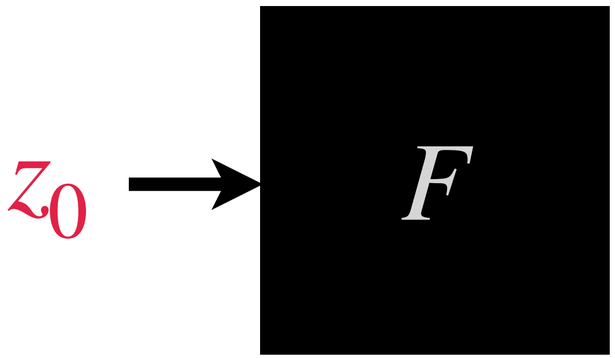
Limitation: Memory scales linearly with iterations.

Memory-Efficient Approach: IVC [Val08,BGH19,BCLMS21,KST21]

Solution: Locally process each chunk of execution and incrementally update the proof without increasing its size.

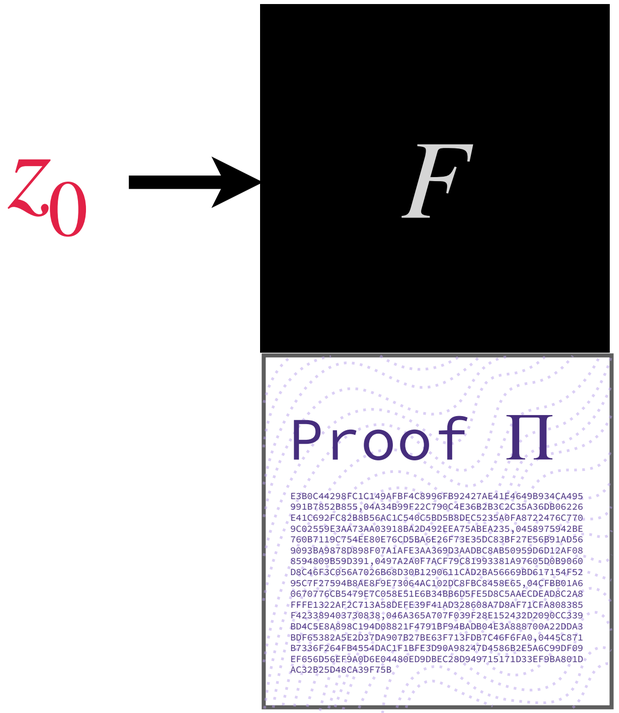
Memory-Efficient Approach: IVC [Val08,BGH19,BCLMS21,KST21]

Solution: Locally process each chunk of execution and incrementally update the proof without increasing its size.



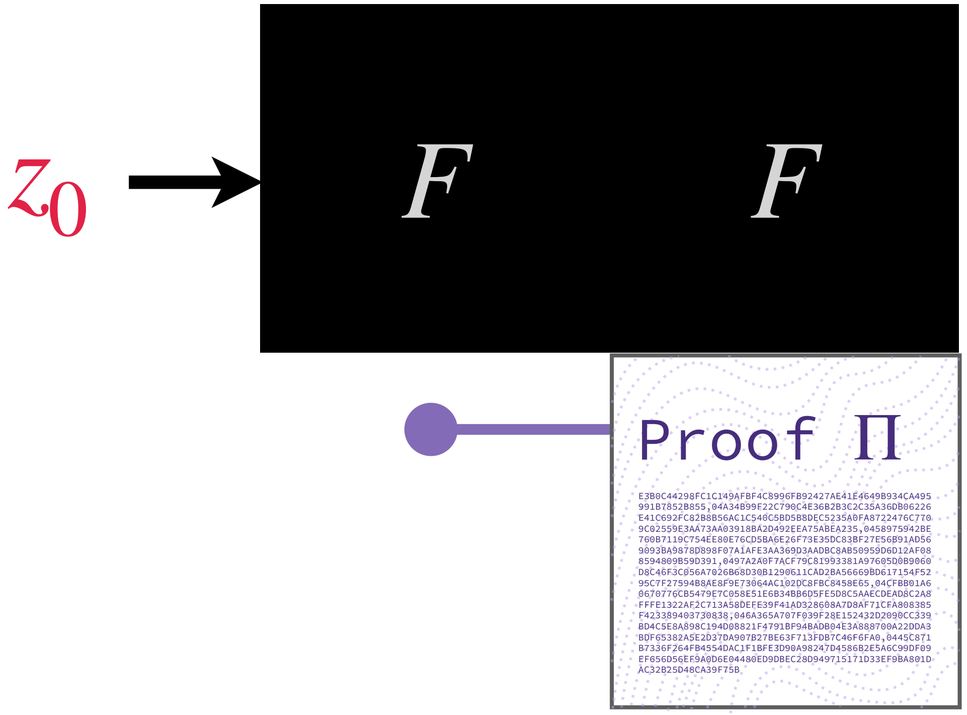
Memory-Efficient Approach: IVC [Val08,BGH19,BCLMS21,KST21]

Solution: Locally process each chunk of execution and incrementally update the proof without increasing its size.



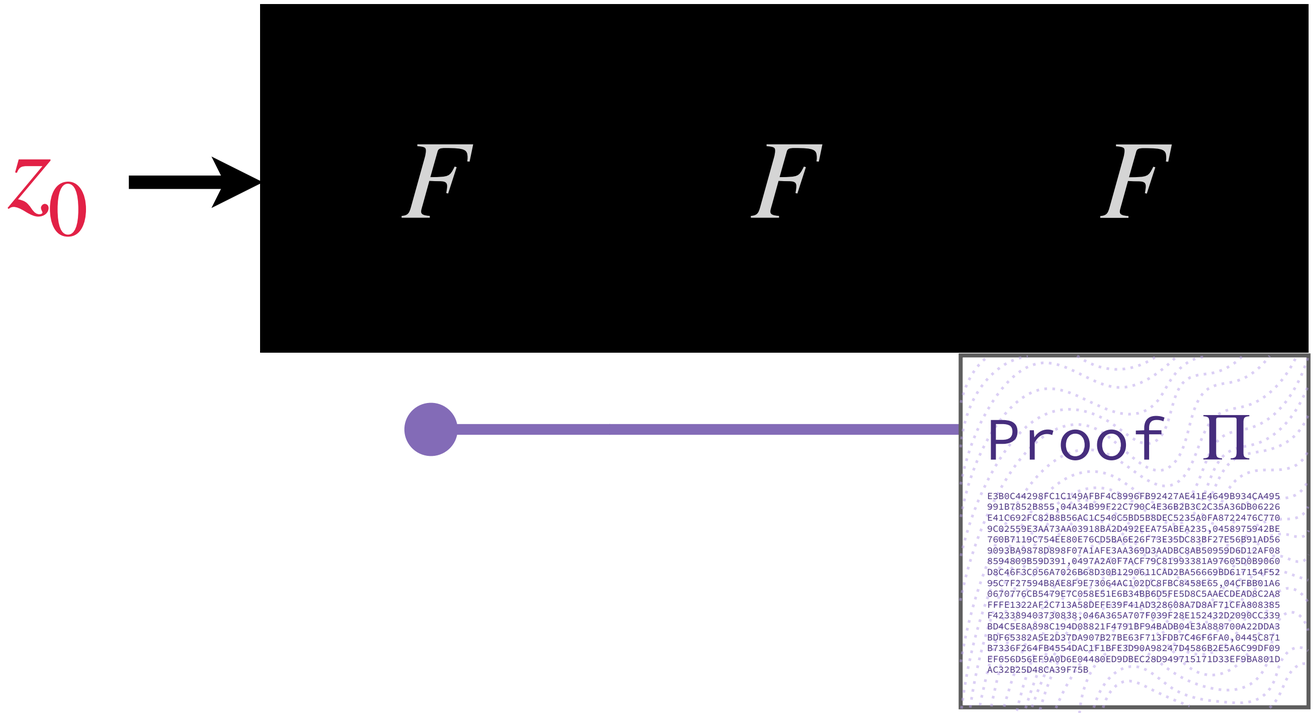
Memory-Efficient Approach: IVC [Val08,BGH19,BCLMS21,KST21]

Solution: Locally process each chunk of execution and incrementally update the proof without increasing its size.



Memory-Efficient Approach: IVC [Val08,BGH19,BCLMS21,KST21]

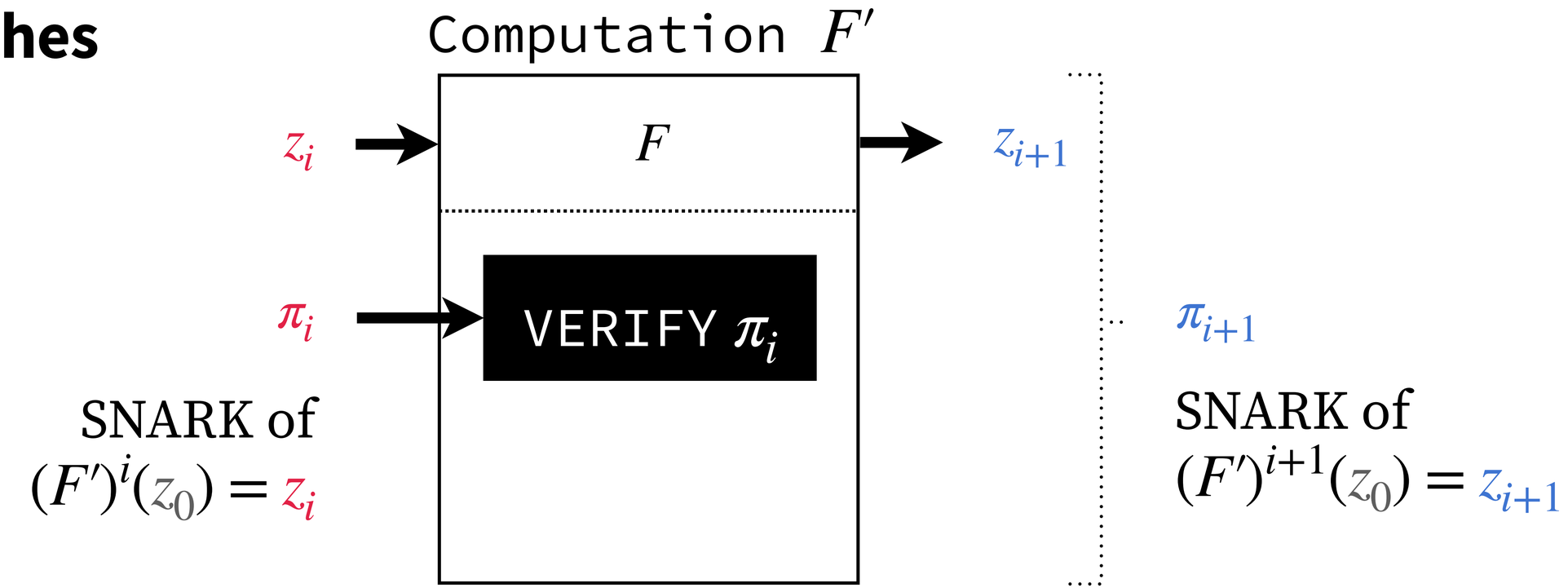
Solution: Locally process each chunk of execution and incrementally update the proof without increasing its size.



Memory-Efficient Approach: IVC [Val08,BGH19,BCLMS21,KST21]

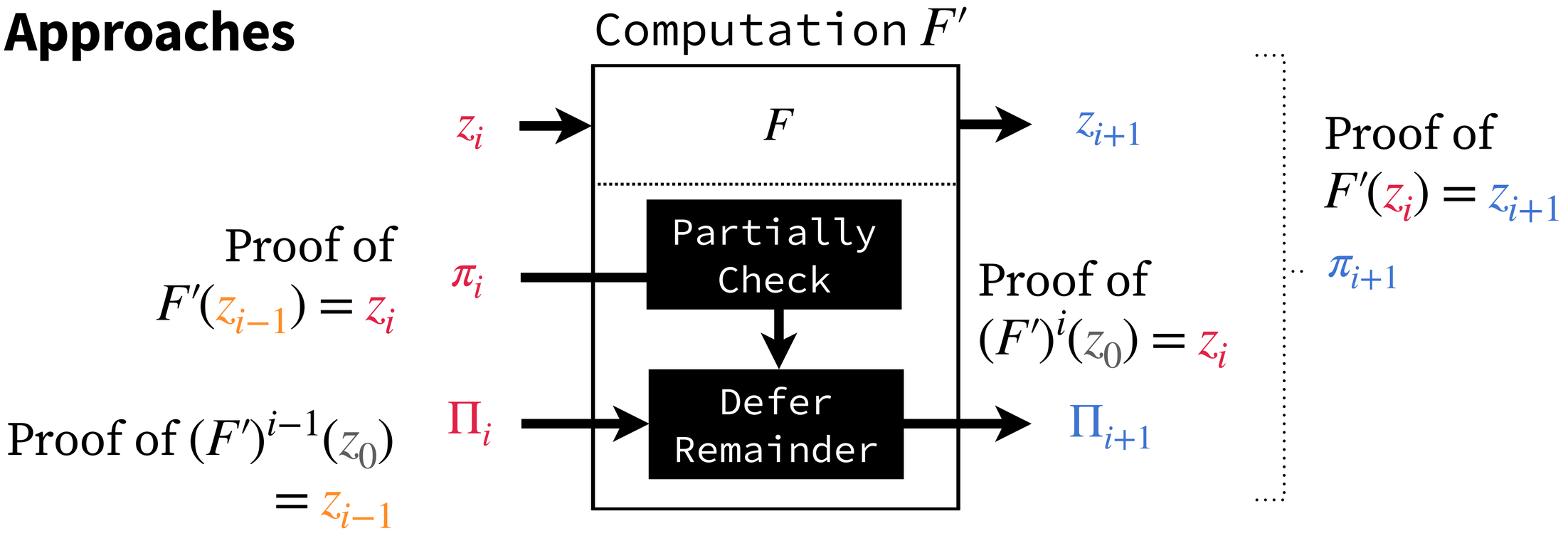
Solution: Locally process each chunk of execution and incrementally update the proof without increasing its size.

IVC Approaches



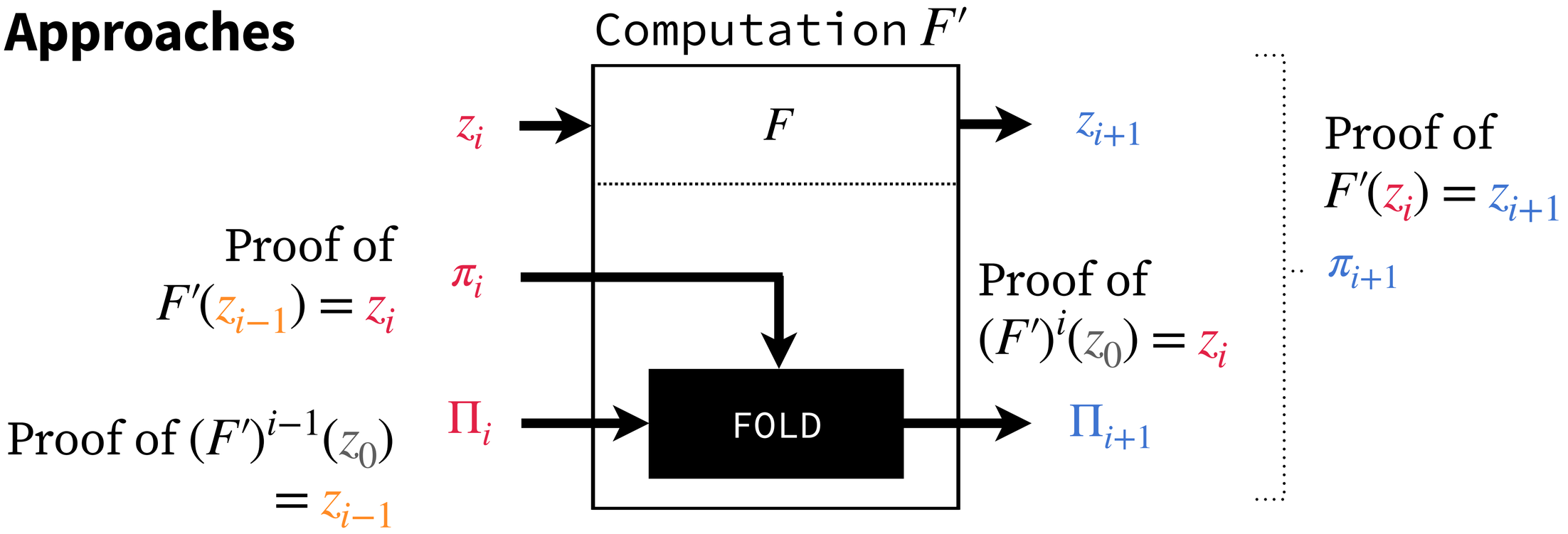
	Direct Recursion [Val08,BCTV14]	**Accumulation** [BGH19,BCMS20,BDFG21]	**Folding** [BCLMS21,KST21]
Prover	Produces SNARK [10MSMs on pairing cycle]	Produces SNARK [10MSMs on standard cycle]	Only commits to witness and auxiliary materials [2MSMs]
Verifier	Verifies entire SNARK	Partially verifies SNARK	Constant grp ops and hashes

IVC Approaches



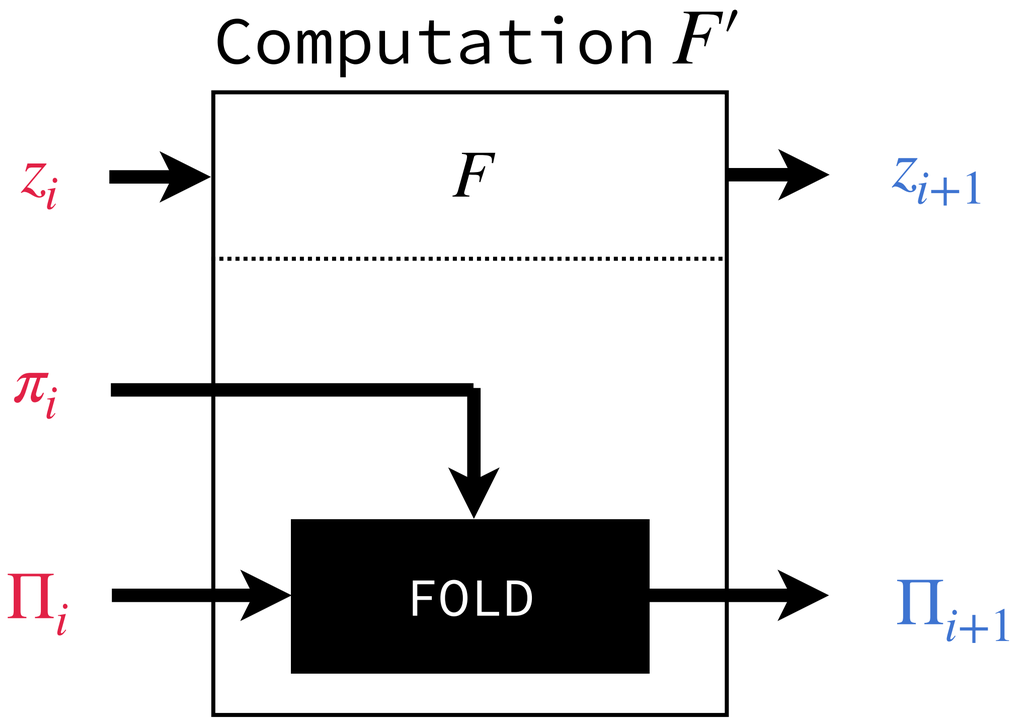
	Direct Recursion [Val08,BCTV14]	**Accumulation** [BGH19,BCMS20,BDFG21]	**Folding** [BCLMS21,KST21]
Prover	Produces SNARK [10MSMs on pairing cycle]	Produces SNARK [10MSMs on standard cycle]	Only commits to witness and auxiliary materials [2MSMs]
Verifier	Verifies entire SNARK	Partially verifies SNARK	Constant grp ops and hashes

IVC Approaches

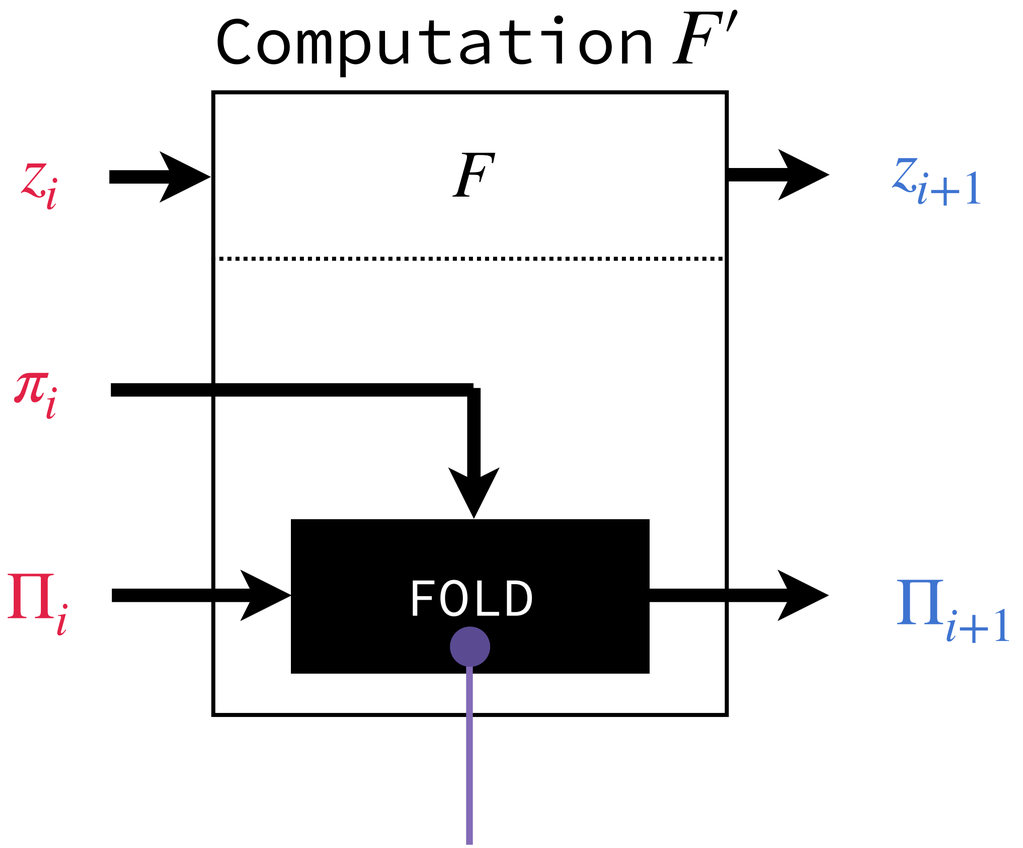


	Direct Recursion [Val08,BCTV14]	**Accumulation** [BGH19,BCMS20,BDFG21]	**Folding** [BCLMS21,KST21]
Prover	Produces SNARK [10MSMs on pairing cycle]	Produces SNARK [10MSMs on standard cycle]	Only commits to witness and auxiliary materials [2MSMs]
Verifier	Verifies entire SNARK	Partially verifies SNARK	Constant grp ops and hashes

Limitations with Prior Folding-Based IVC Schemes

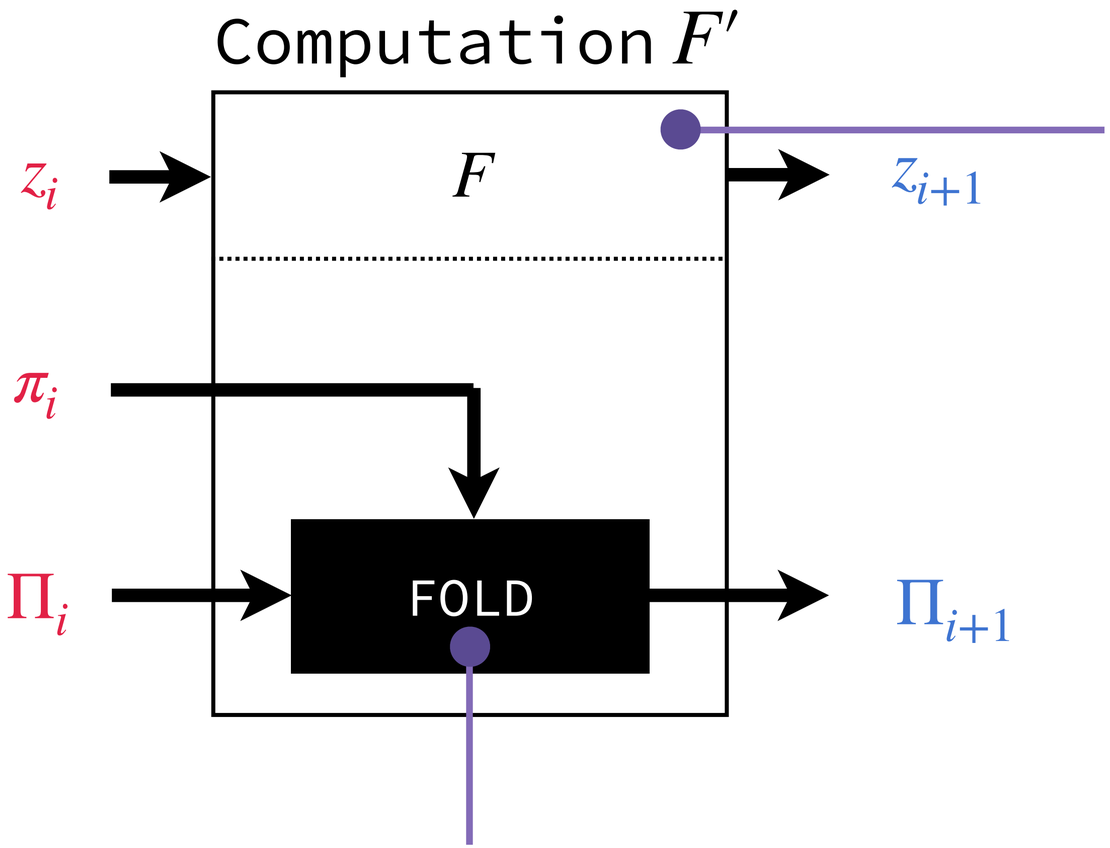


Limitations with Prior Folding-Based IVC Schemes



(1) No efficient method for folding **high-degree** constraints.

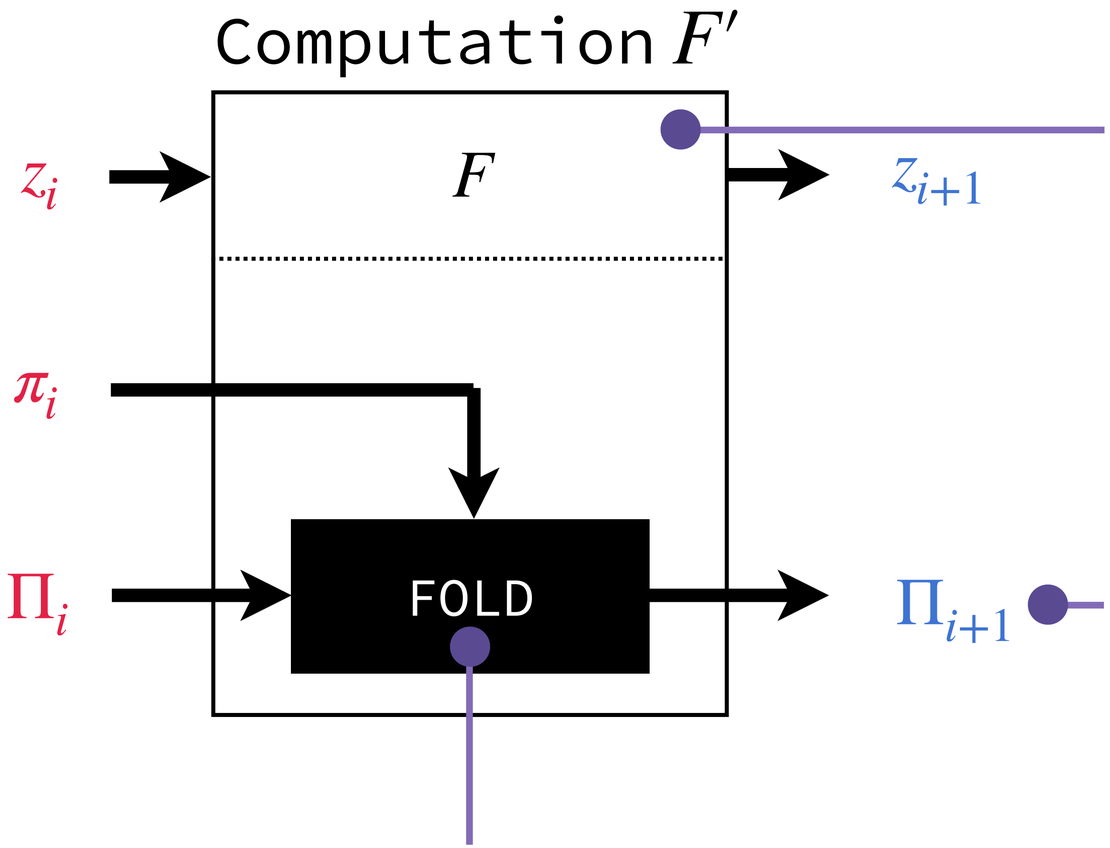
Limitations with Prior Folding-Based IVC Schemes



(2) IVC limited to repeated applications of a **single** function.

(1) No efficient method for folding **high-degree** constraints.

Limitations with Prior Folding-Based IVC Schemes

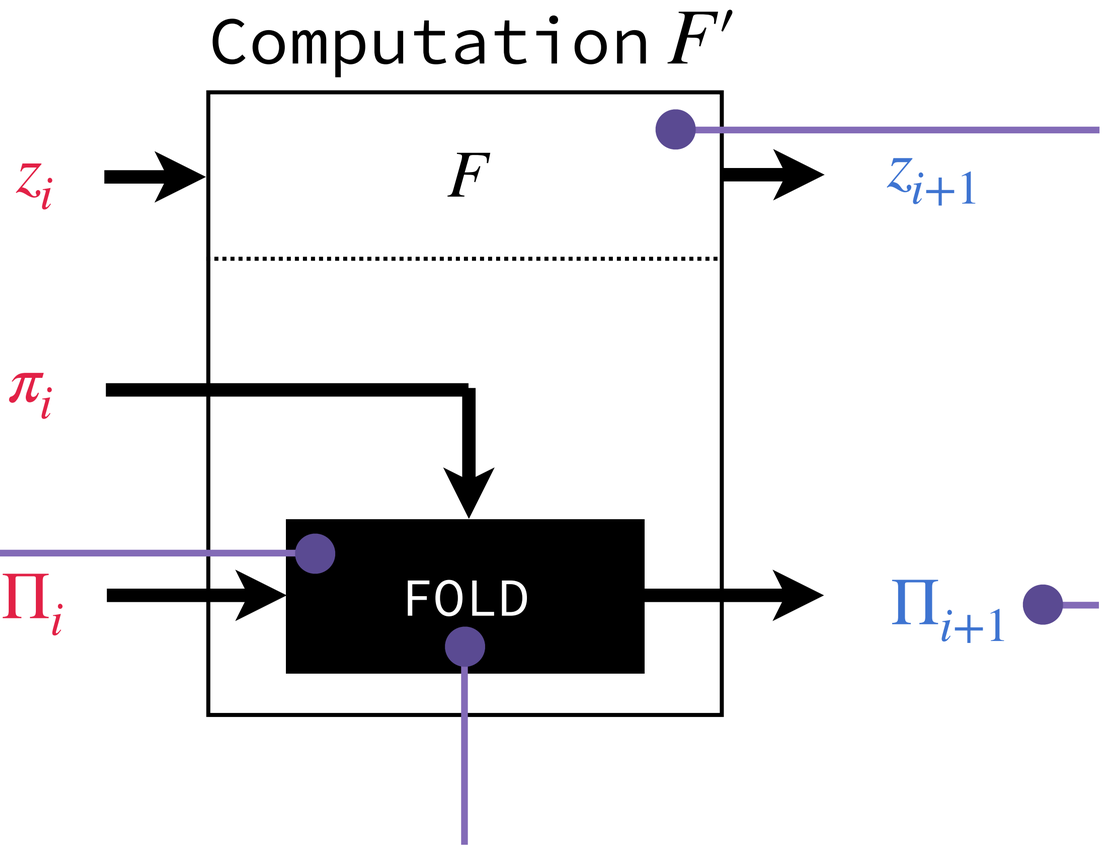


(2) IVC limited to repeated applications of a **single** function.

(3) Zero-knowledge requires zk(S)NARKs or expensive overhead per step.

(1) No efficient method for folding **high-degree** constraints.

Limitations with Prior Folding-Based IVC Schemes



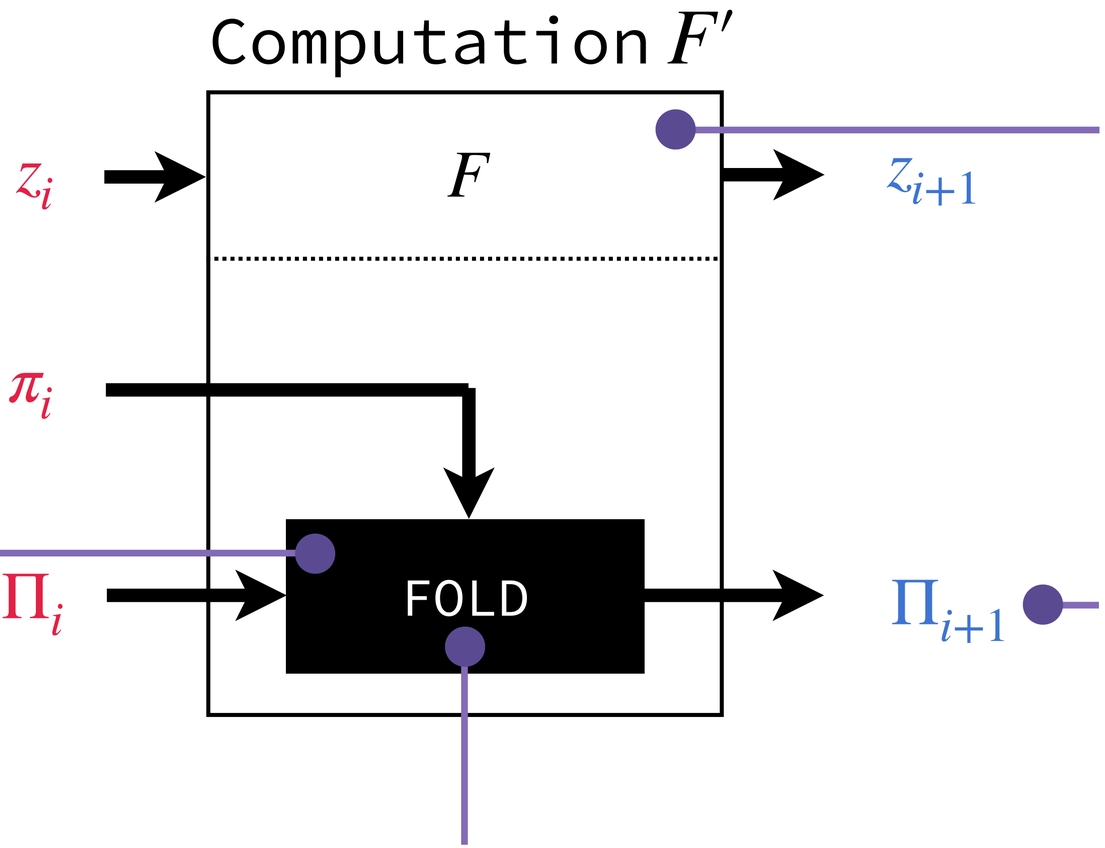
(2) IVC limited to repeated applications of a **single** function.

(3) Zero-knowledge requires zk(S)NARKs or expensive overhead per step.

(4) instantiation over a cycle of curves is complex and less efficient.

(1) No efficient method for folding **high-degree** constraints.

HyperNova Contributions



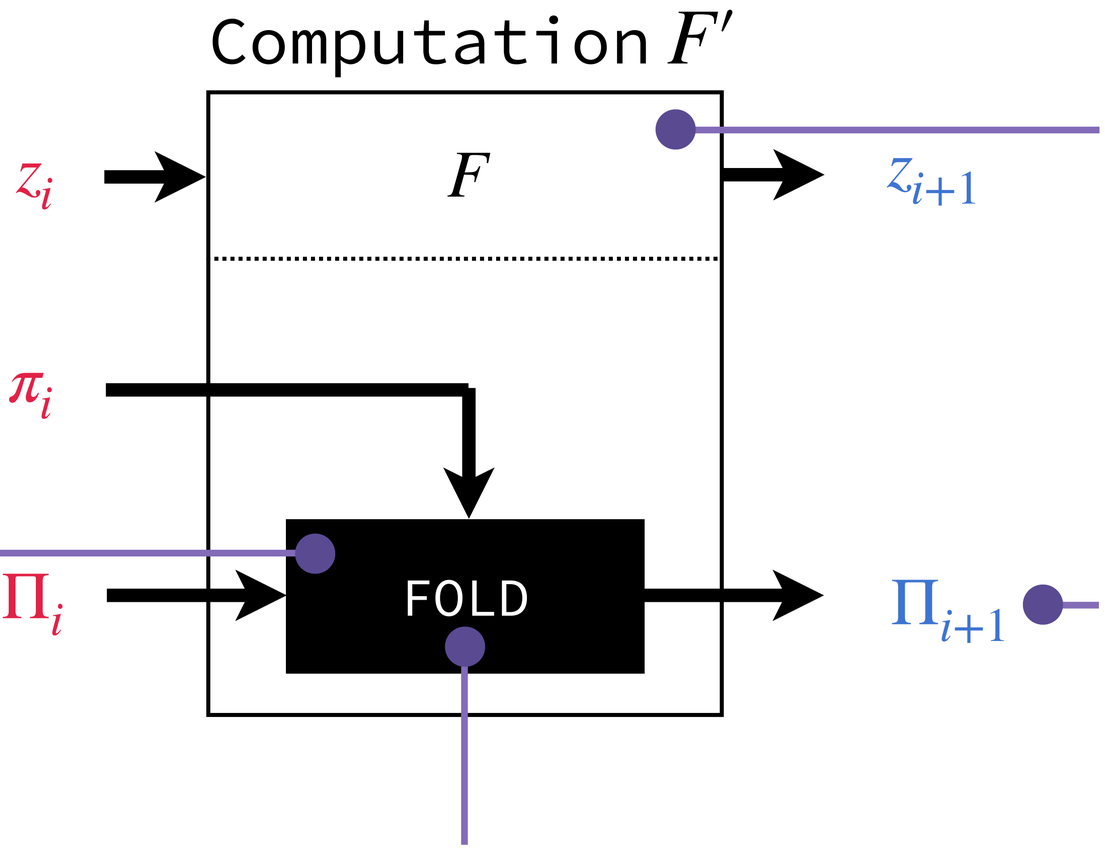
(2) IVC limited to repeated applications of a **single** function.

(3) Zero-knowledge requires zk(S)NARKs or expensive overhead per step.

(4) instantiation over a cycle of curves is complex and less efficient.

(1) We design an efficient folding scheme for high-degree constraints (CCS [STW23])

HyperNova Contributions



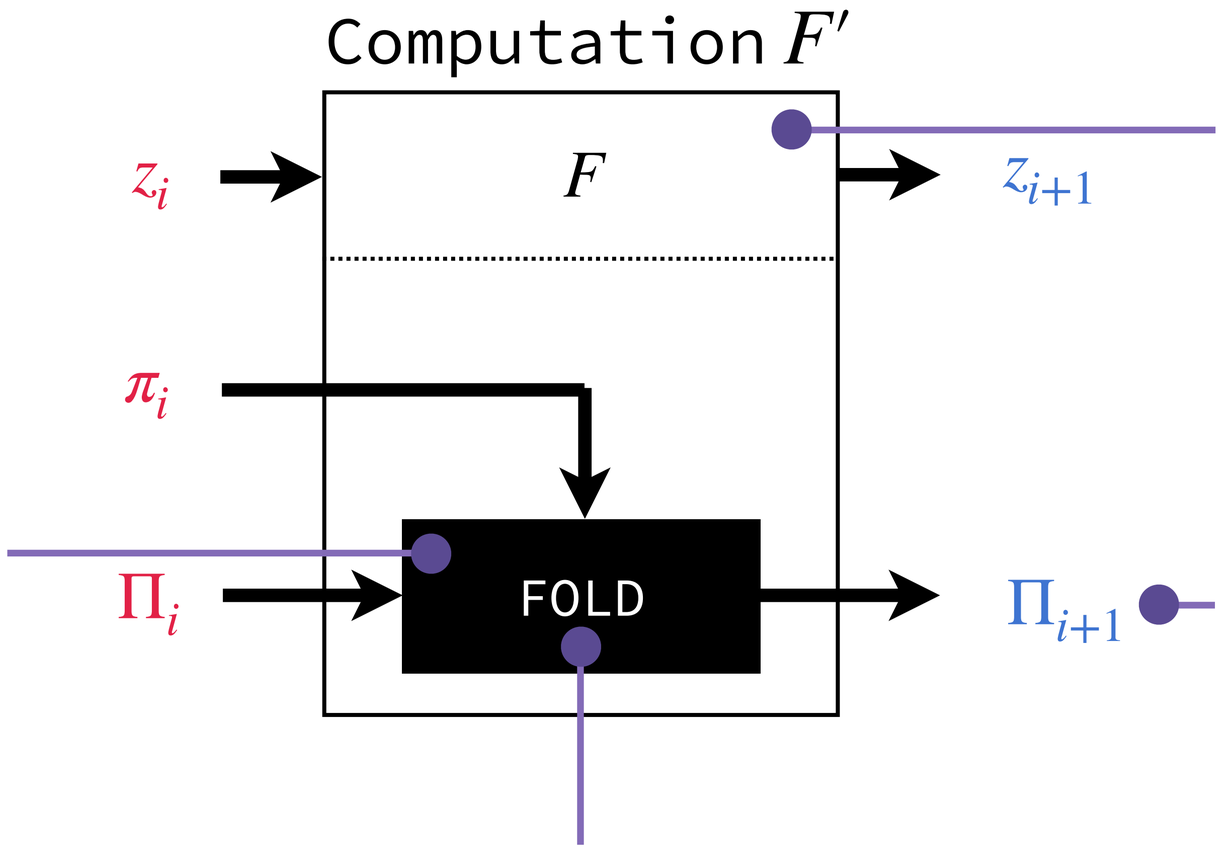
(2) We introduce and achieve **NIVC**, which extends IVC to multiple functions.

(4) instantiation over a cycle of curves is complex and less efficient.

(3) Zero-knowledge requires zk(S)NARKs or expensive overhead per step.

(1) We design an efficient folding scheme for high-degree constraints (CCS [STW23])

HyperNova Contributions



(2) We introduce and achieve **NIVC**, which extends IVC to multiple functions.

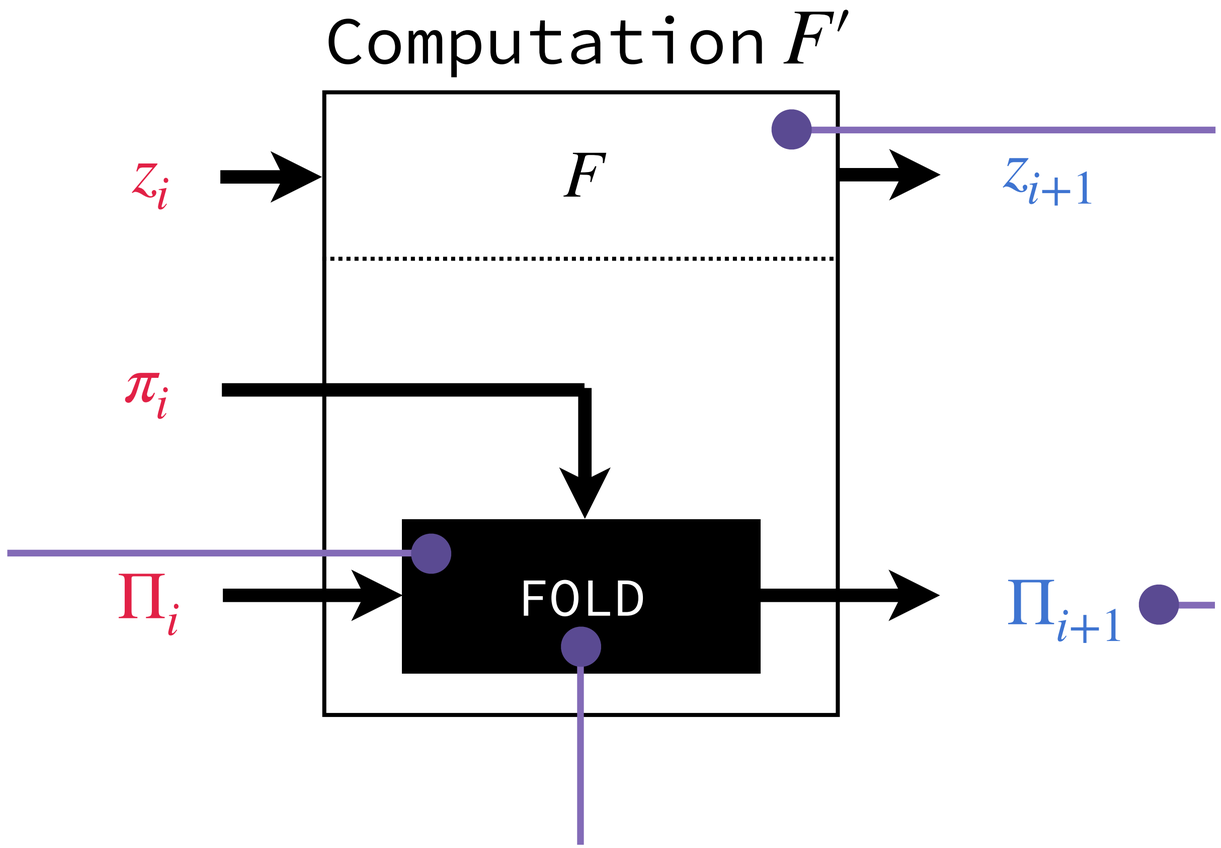
(3) We show how folding schemes can be used to blind the final proof.

(4) instantiation over a cycle of curves is complex and less efficient.

(1) We design an efficient folding scheme for high-degree constraints (CCS [STW23])

HyperNova Contributions

(4) We introduce CycleFold, a general methodology to efficiently instantiate folding schemes over cycles of curves.

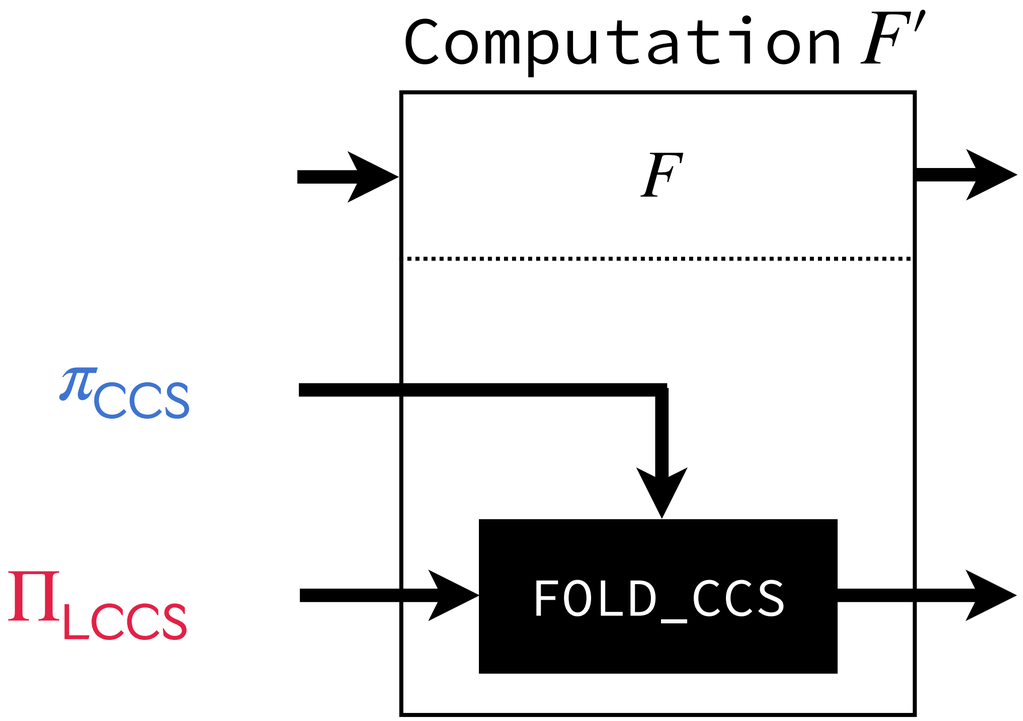


(2) We introduce and achieve **NIVC**, which extends IVC to multiple functions.

(3) We show how folding schemes can be used to blind the final proof.

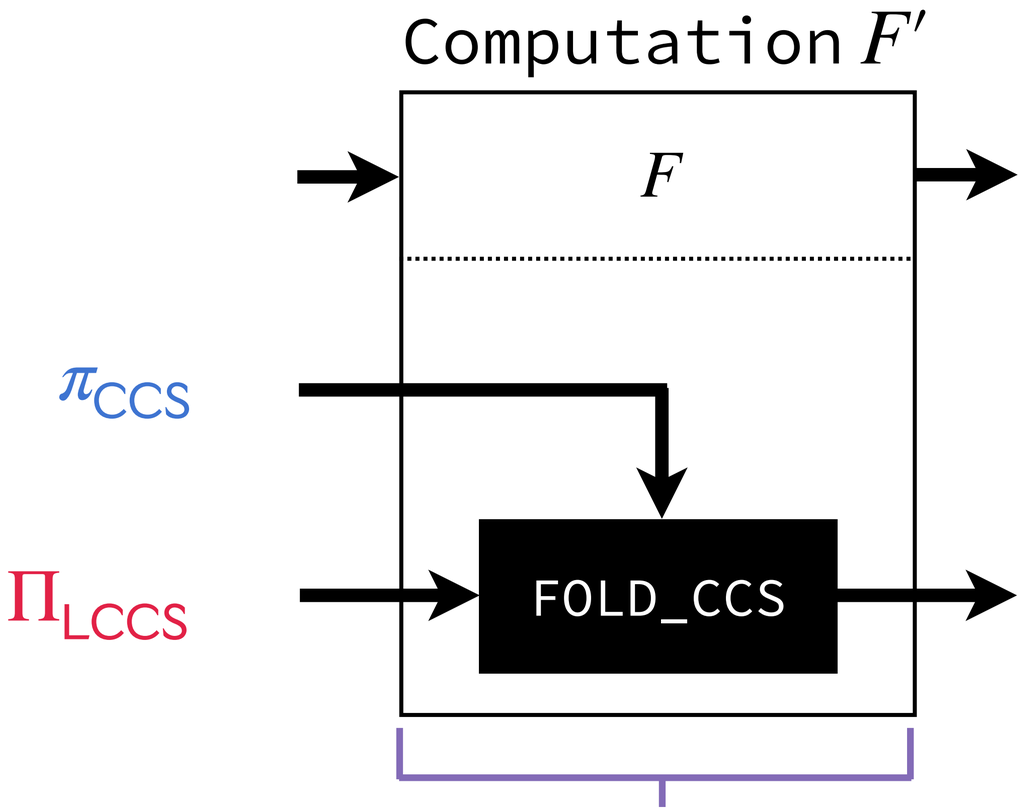
(1) We design an efficient folding scheme for high-degree constraints (CCS [STW23])

Folding High-Degree CCS Constraints



The core idea is to use the sumcheck protocol to reduce non-linear constraints into linear constraints.

Folding High-Degree CCS Constraints

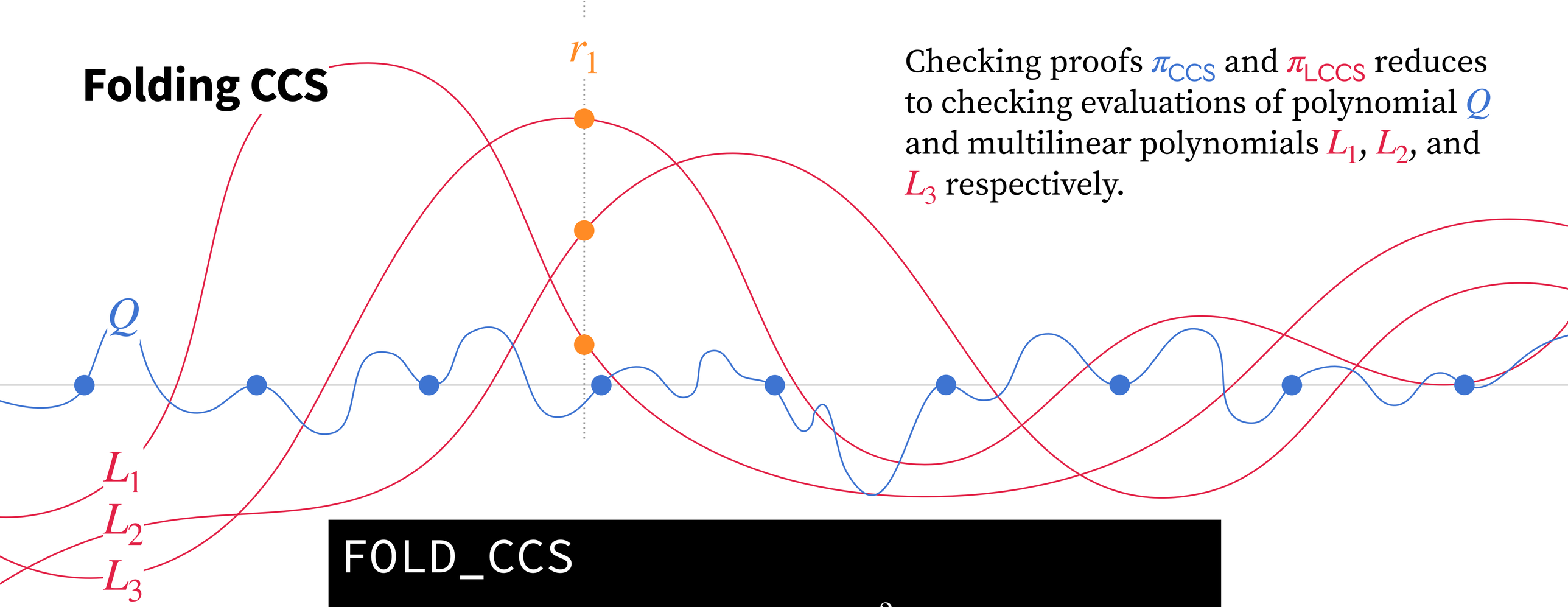


The core idea is to use the sumcheck protocol to reduce non-linear constraints into linear constraints.

We achieve a recursion overhead dominated by 1 GSM and $d \log |F'|$ CHRFs.

Folding CCS

Folding CCS



Checking proofs π_{CCS} and π_{LCCS} reduces to checking evaluations of polynomial Q and multilinear polynomials L_1, L_2, and L_3 respectively.

$\pi_{\mathsf{CCS}} \rightarrow$

$\Pi_{\mathsf{LCCS}} \rightarrow$

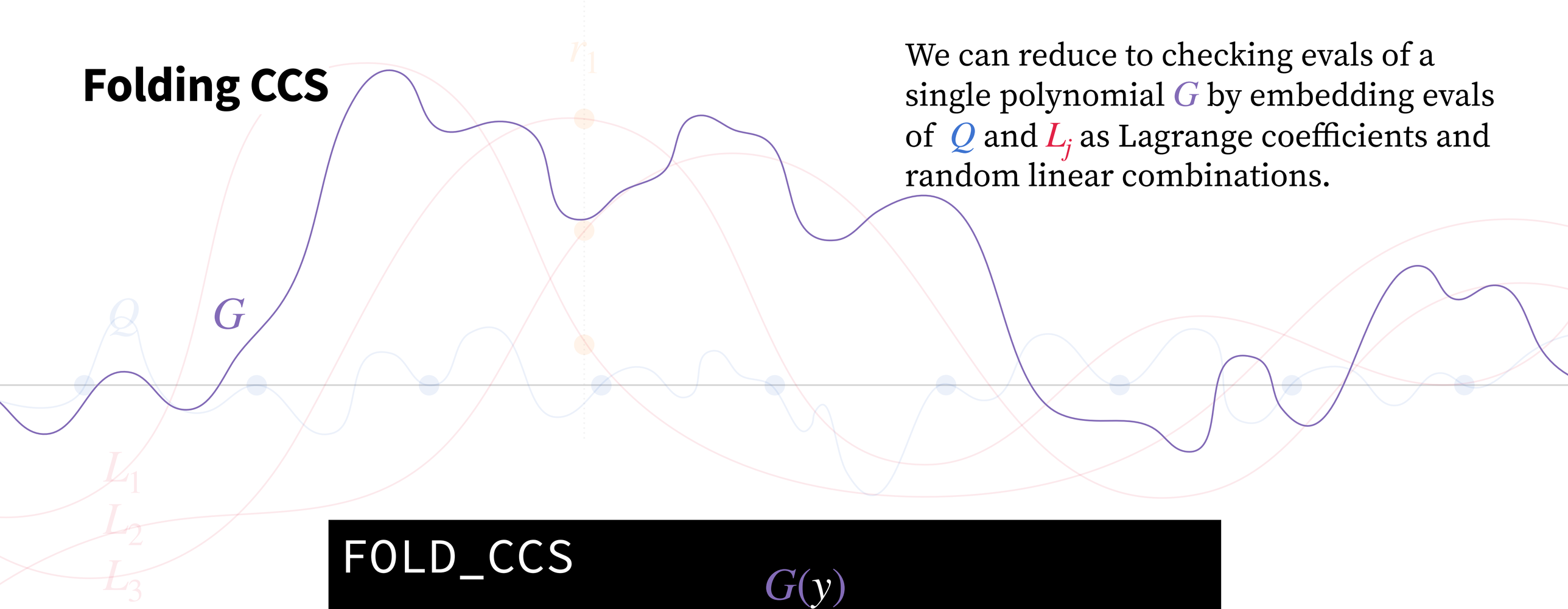
FOLD_CCS

$$\forall x \in [n] \,.\, Q(x) \stackrel{?}{=} 0$$

$$\forall j \in [t] \,.\, L_j(r_1) \stackrel{?}{=} v_j$$

Folding CCS

We can reduce to checking evals of a single polynomial G by embedding evals of Q and L_j as Lagrange coefficients and random linear combinations.



FOLD_CCS

$G(y)$

π_{CCS} →

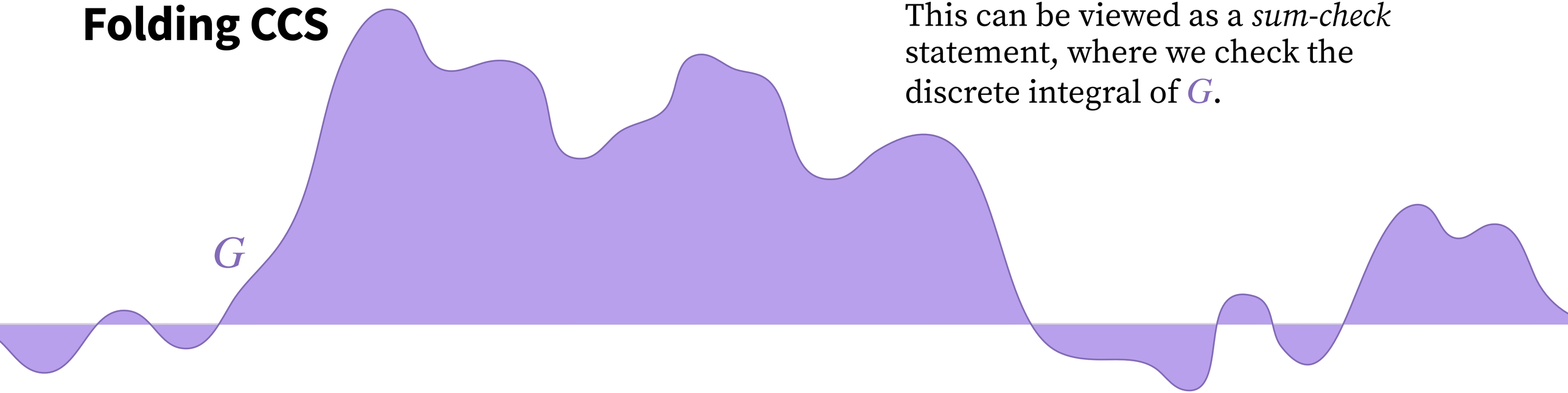
$$\sum_{y} \gamma^{t+1} \cdot \mathrm{eq}(r_2, y) \cdot Q(y) + \mathrm{eq}(r_1, y) \sum_{j \in [t]} \gamma^{j} \cdot L_j(y)$$

Π_{LCCS} →

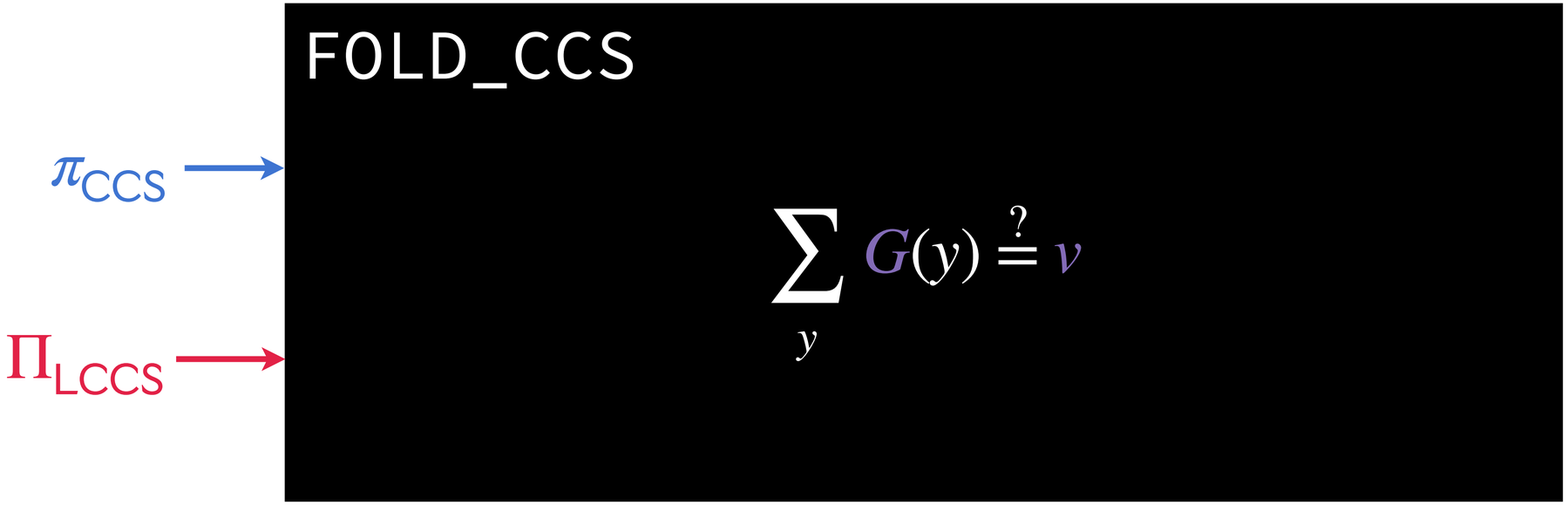
$$\stackrel{?}{=} \sum_{j \in [t]} \gamma^{j} \cdot v_j \quad v$$

For random γ, r_2

Folding CCS

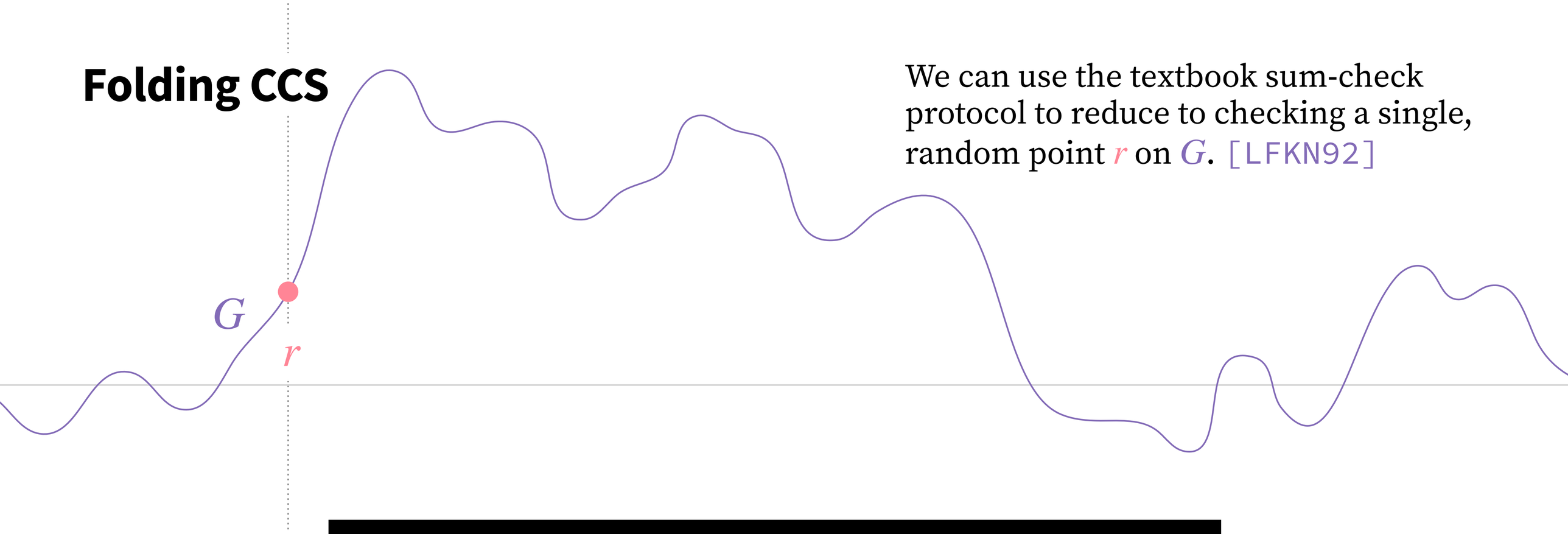


This can be viewed as a *sum-check* statement, where we check the discrete integral of G.



Folding CCS

We can use the textbook sum-check protocol to reduce to checking a single, random point r on G. [LFKN92]



π_{CCS} →

Π_{LCCS} →

FOLD_CCS

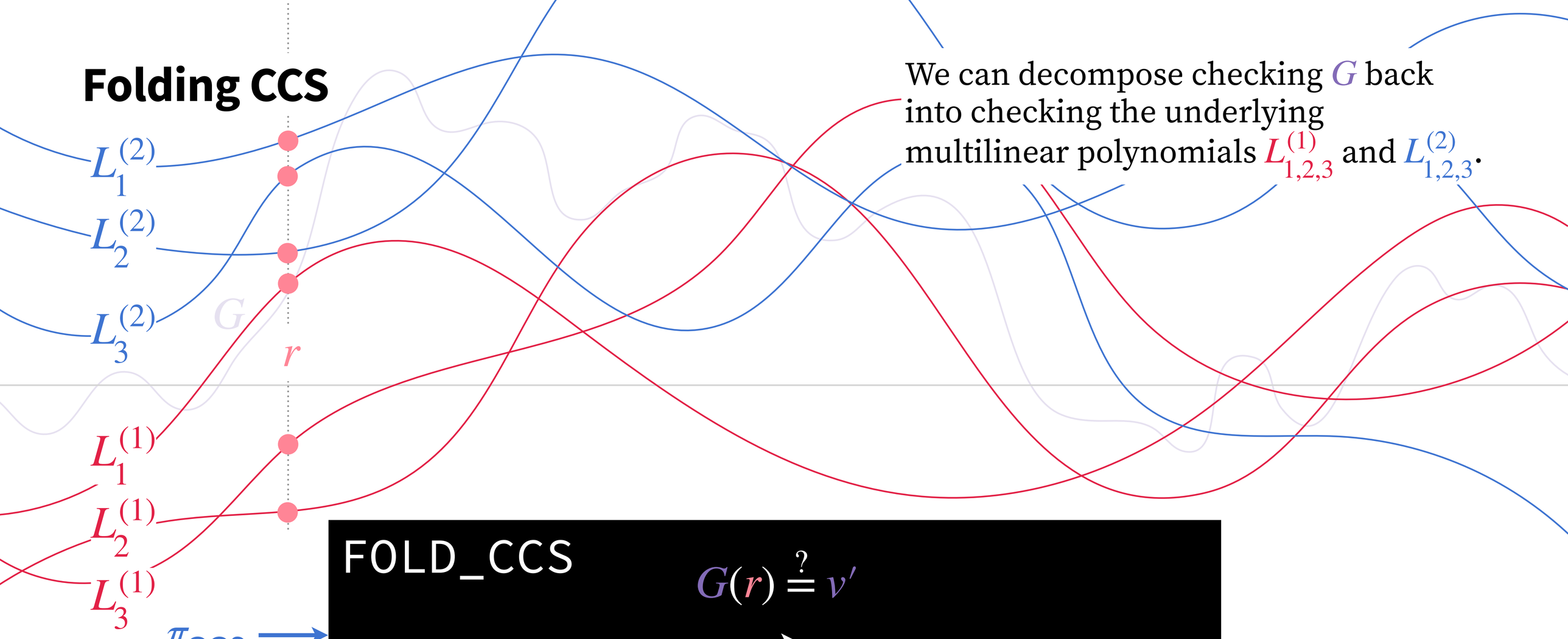
$$\sum_y G(y) \stackrel{?}{=} v$$

→

$$G(r) \stackrel{?}{=} v'$$

Folding CCS

We can decompose checking G back into checking the underlying multilinear polynomials $L^{(1)}_{1,2,3}$ and $L^{(2)}_{1,2,3}$.



π_{CCS} →

Π_{LCCS} →

FOLD_CCS

$$G(r) \stackrel{?}{=} v'$$

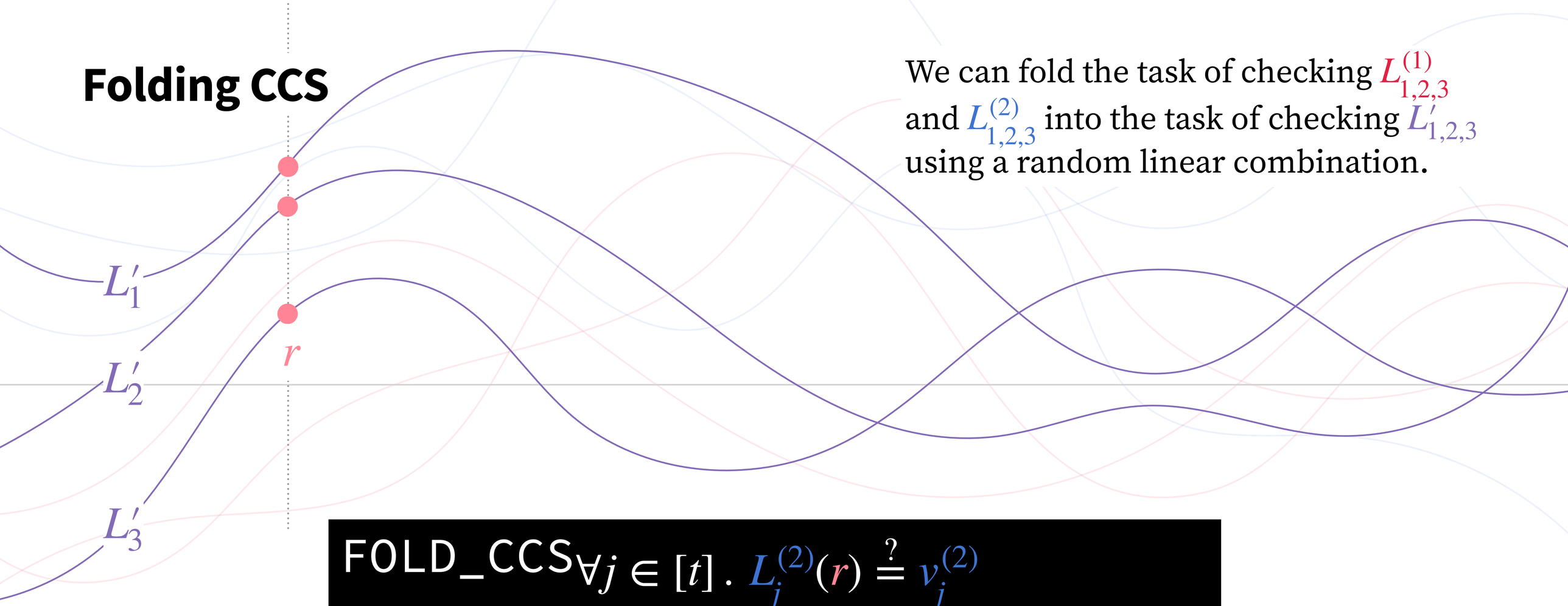
→

$$\forall j \in [t] \,.\, L_j^{(2)}(r) \stackrel{?}{=} v_j^{(2)}$$

$$\forall j \in [t] \,.\, L_j^{(1)}(r) \stackrel{?}{=} v_j^{(1)}$$

Folding CCS

We can fold the task of checking $L^{(1)}_{1,2,3}$ and $L^{(2)}_{1,2,3}$ into the task of checking $L'_{1,2,3}$ using a random linear combination.



π_{CCS} →

Π_{LCCS} →

FOLD_CCS

$\forall j \in [t] \,.\, L_j^{(2)}(r) \stackrel{?}{=} v_j^{(2)}$

$\forall j \in [t] \,.\, L_j^{(1)}(r) \stackrel{?}{=} v_j^{(1)}$

→

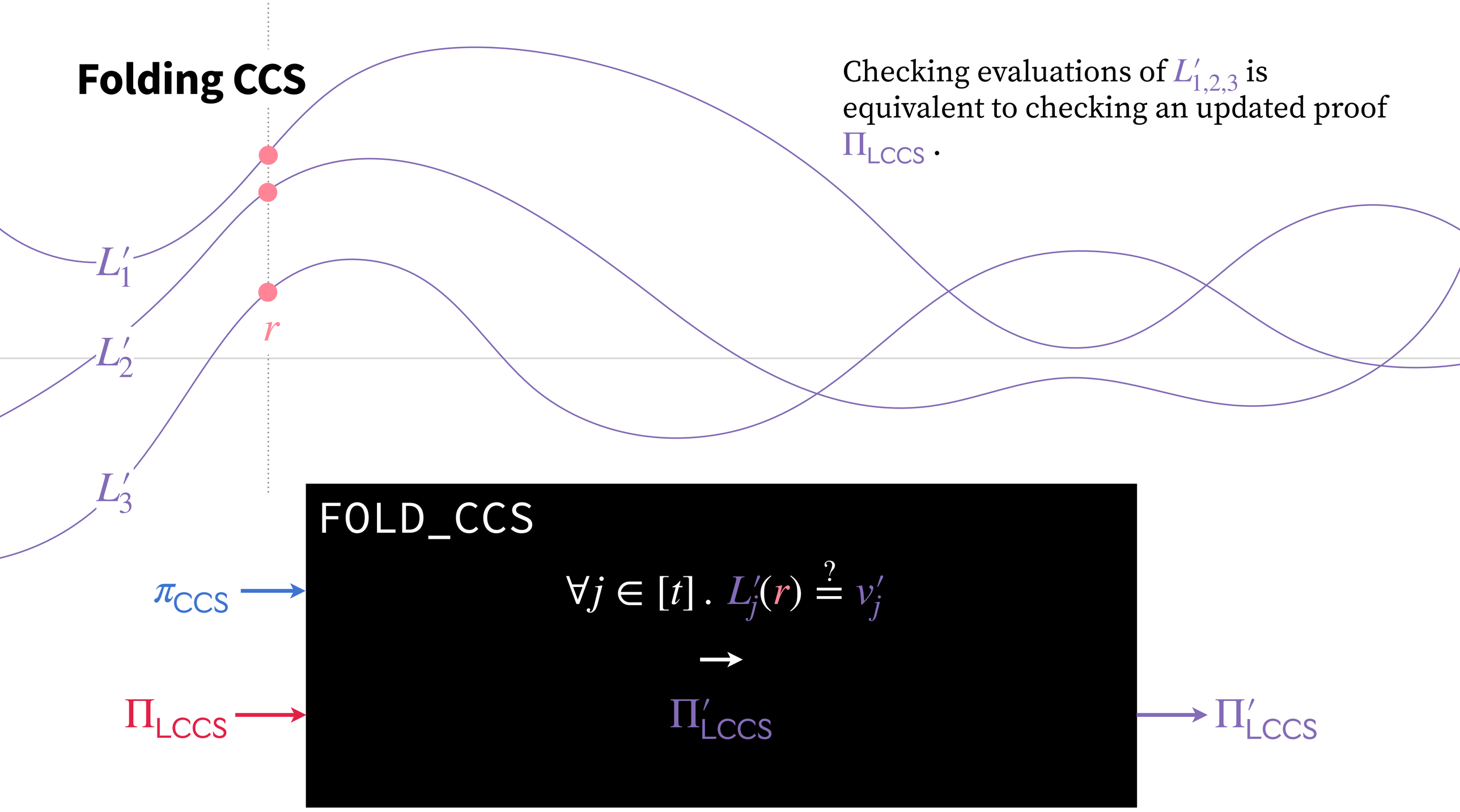
$\forall j \in [t] \,.\, L'_j(r) \stackrel{?}{=} v'_j$

For $L'_j = L_j^{(1)} + \rho \cdot L_j^{(2)}$

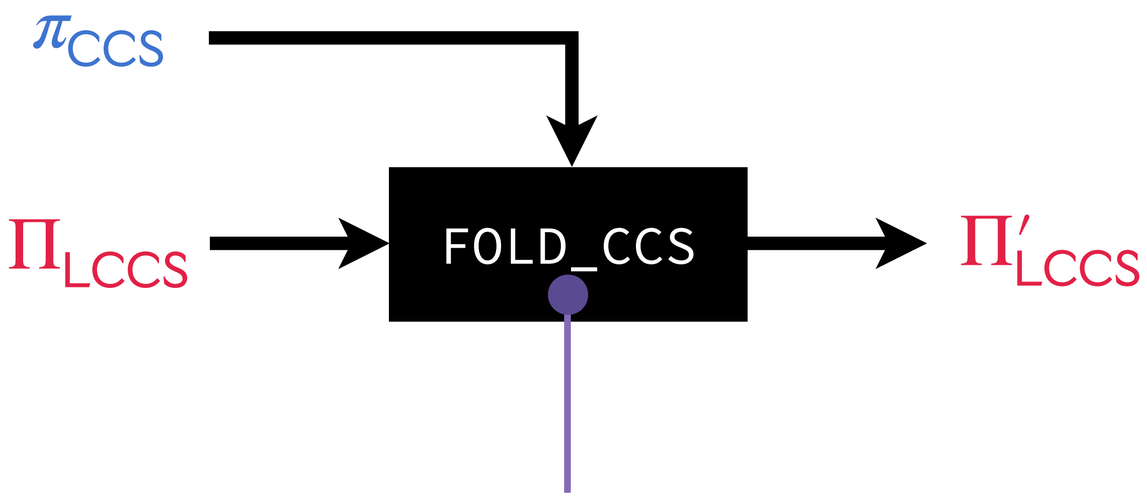
For random ρ

Folding CCS

Checking evaluations of $L'_{1,2,3}$ is equivalent to checking an updated proof Π_{LCCS}.



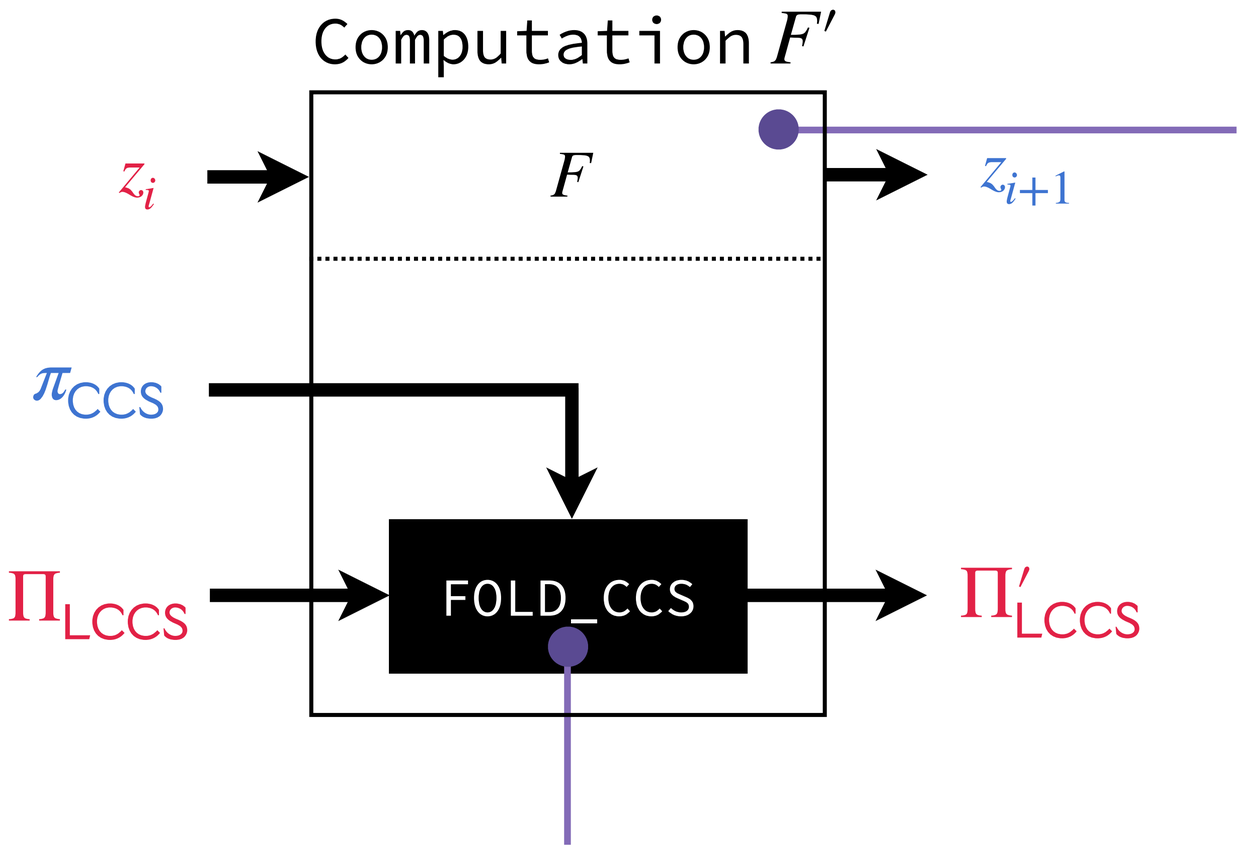
HyperNova: Efficient (N)IVC over High-Degree Constraints



(1) We design an efficient folding scheme
for high-degree constraints (CCS [STW23])

akothapalli@cmu.edu

HyperNova: Efficient (N)IVC over High-Degree Constraints

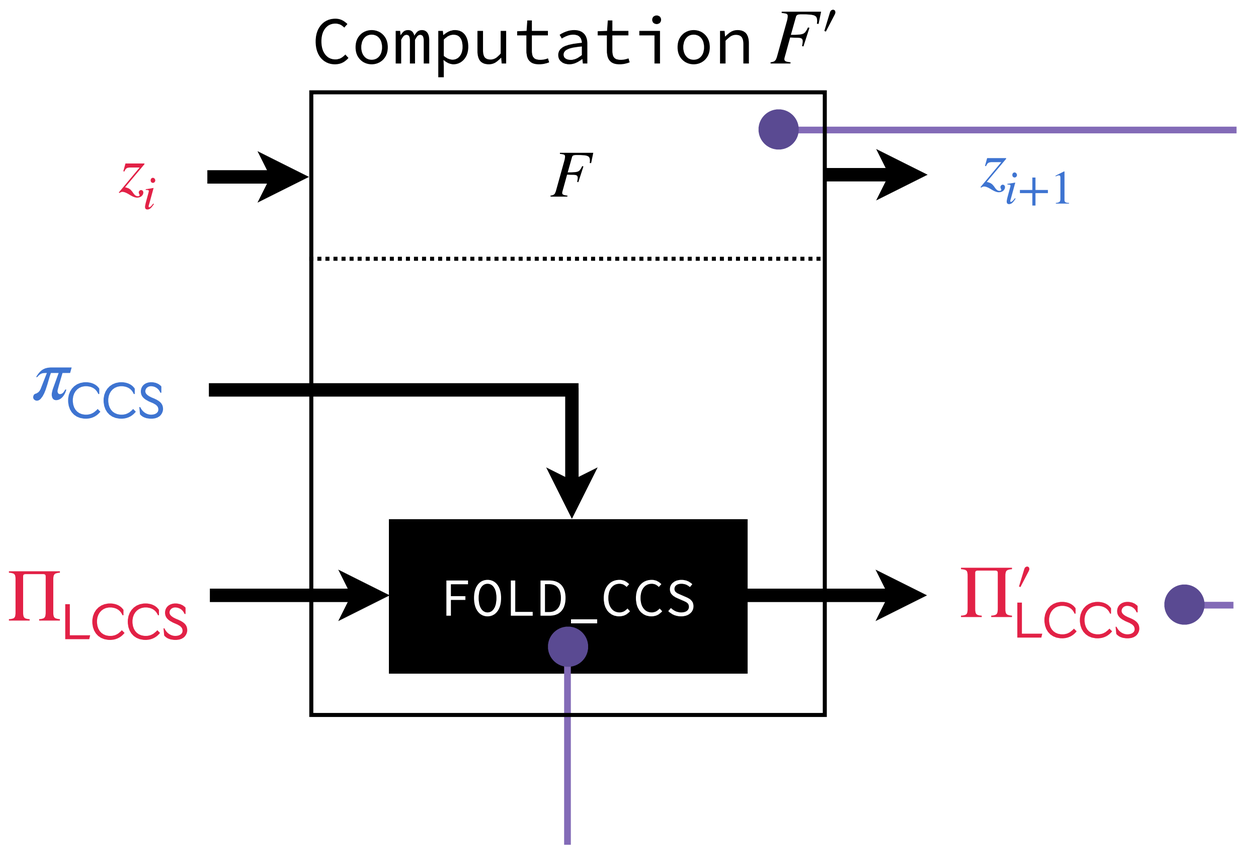


(2) We introduce and achieve **NIVC**, which extends IVC to multiple functions.

(1) We design an efficient folding scheme for high-degree constraints (CCS [STW23])

akothapalli@cmu.edu

HyperNova: Efficient (N)IVC over High-Degree Constraints



(1) We design an efficient folding scheme for high-degree constraints (CCS [STW23])

(2) We introduce and achieve **NIVC**, which extends IVC to multiple functions.

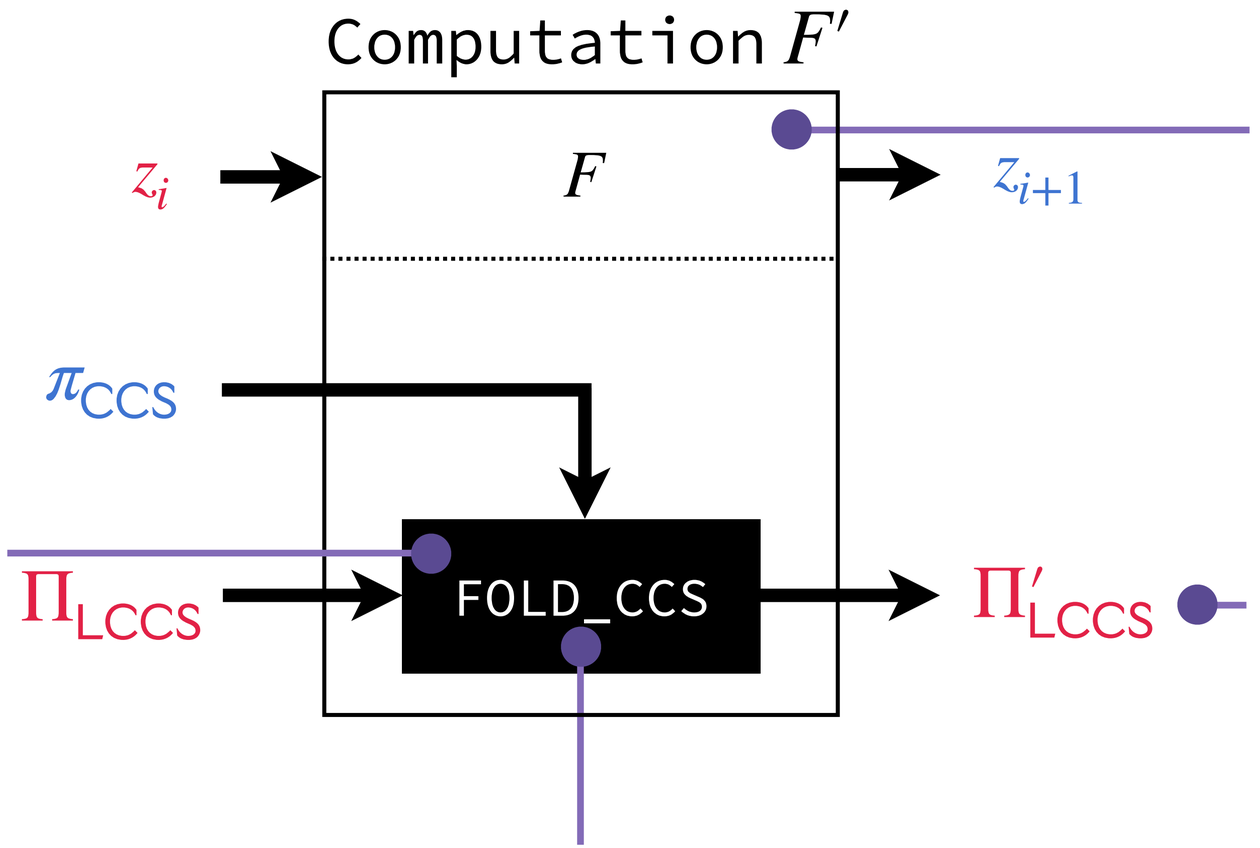
(3) We show how folding schemes can be used to blind the final IVC proof.

ia.cr/2023/573

akothapalli@cmu.edu

HyperNova: Efficient (N)IVC over High-Degree Constraints

(4) We introduce CycleFold, a general methodology to efficiently instantiate folding schemes over cycles of curves.



(2) We introduce and achieve **NIVC**, which extends IVC to multiple functions.

(3) We show how folding schemes can be used to blind the final IVC proof.

(1) We design an efficient folding scheme for high-degree constraints (CCS [STW23])

ia.cr/2023/573

akothapalli@cmu.edu

References

[BCLMS21] Bünz, Chiesa, Lin, Mishra, Spooner. Proof Carrying Data without Succinct Arguments.

[BDFG21] Boneh, Drake, Fisch, Gabizon. Halo Infinite: Recursive zkSNARKs from any Additive Polynomial Commitment Scheme.

[BCCGP16] Bootle, Cerulli, Chaidos, Groth, Petit. Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting.

[RZ21] Ràfols and Zapico. An Algebraic Framework for Universal and Updatable SNARKs.

[KST22] Kothapalli, Setty, Tzialla. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes.

[CNRZZ22] Campanelli, Nitulescu, Rafols, Zacharakis, Zapico. Linear-map vector commitments and their practical applications.

[LFKN92] Lund, Fortnow, Karloff, Nisan. Algebraic methods for interactive proof systems.

[BBBPWM18] Bünz, Bootle, Boneh, Poelstra, Wuille, and Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More.

[BMMTV21] Bünz, Maller, Mishra, Tyagi, and Vesely. Proofs for inner pairing products and applications.

[WTSTW18] Wahby, Tzialla, Shelat, Thaler, and Walfish. Doubly-efficient zkSNARKs without trusted setup.

[BTVW14] Blumberg, Thaler, Vu, and Walfish. Verifiable computation using multiple provers.

[GMR85] Goldwasser, Micali, and Rackoff. The knowledge complexity of interactive proof systems.

[Lee21] Lee. Dory: Efficient, Transparent arguments for Generalised Inner Products and Polynomial Commitments.

[BZ12] Bayer, and Groth. Efficient zero-knowledge argument for correctness of a shuffle.

[CBBZ22] Chen, Bünz, Boneh, Zhang. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates.

[Set20] Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup.

[BCHO22] Gemini: Bootle, Chiesa, Hu, Orrù. Elastic SNARKs for Diverse Environments.

[Bay13] Bayer. Practical Zero-Knowledge Protocols Based on the Discrete Logarithm Assumption.

[BCS21] Bootle, Chiesa, and Sotiraki. Sumcheck Arguments and their Applications.

[RZ22] Ràfols, and Zacharakis. Folding Schemes with Selective Verification.

[KS23] Kothapalli, and Setty. HyperNova: Recursive arguments for customizable constraint systems.

[Val08] Valiant. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency.

[BGH19] Bowe, Grigg, Hopwood. Recursive proof composition without a trusted setup.

[AC20] Attema and Cramer. Compressed-protocol theory and practical application to plug & play secure algorithmics.

[ACR21] Attema, Cramer, and Rambaud. Compressed Sigma protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures.

[GKR15] Goldwasser, Tauman Kalai, and Rothblum. Delegating computation: interactive proofs for muggles

[BFLS91] Babai, Fortnow, Levin, and Szegedy. Checking computations in polylogarithmic time.

[BC23] Bünz, and Chen. ProtoStar: Generic Efficient Accumulation/ Folding for Special Sound Protocols