FRIttata: Distributed Proof Generation of FRI-based SNARKs

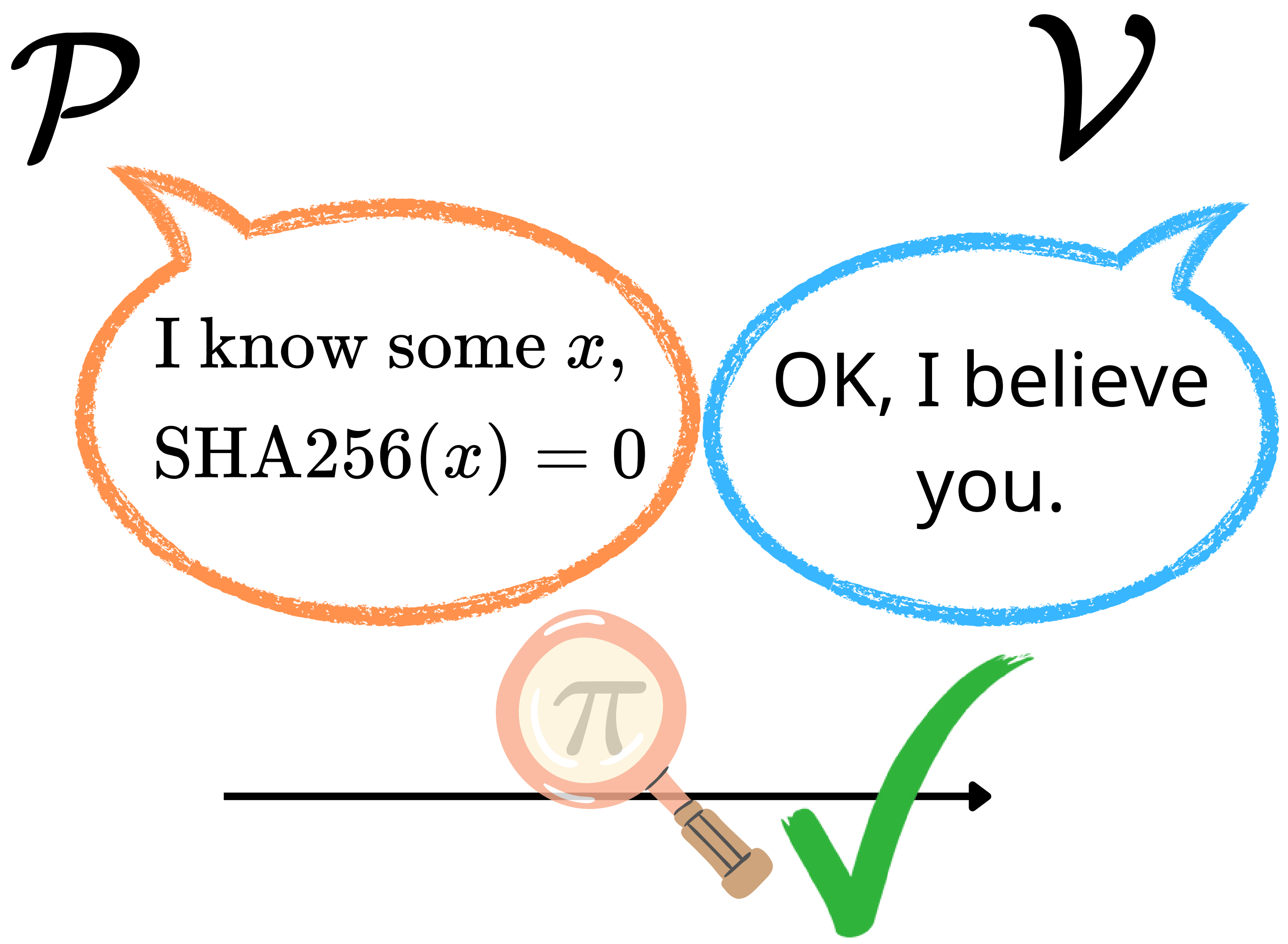
Hua Xu, Mariana Gama, Emad Heydari Beni, Jiayi Kang

Overview

- What is a SNARK + the Plonk proving system
- *Pianist*'s distributed proving system based on Plonk
- Background on FRI
- A distributed FRI-based PCS
- Distributed FRI
- Experiments & Future work

What is a SNARK?

A SNARK is a short "proof" that a computation was done correctly.



- **Succinct**: log(N) proof size and verification time, N = size of statement
- **Non-interactive**: single round of communication
- **ARgument of Knowledge**: being able to produce a proof ==> knowing the input x
- (zk-SNARK) **Zero-Knowledge**: V learns nothing about x beyond the claim

Plonk [GWC19]

Plonk Arithmetization

$a_0 + b_0 = o_0$

$a_1 \cdot b_1 = o_1$

$o_0 = b_1$

$\vdots$

Circuit constraints

$\Longrightarrow$

$g(X), p_0(X)$ and $p_1(X)$ vanish on Ω_X

Polynomial constraints

Plonk Polynomial IOP

$\mathcal{P}$ $\qquad\qquad$ $\mathcal{V}$

$\xrightarrow{\boxed{f_1}}$

$\xleftarrow{r_1}$

$\xrightarrow{\boxed{f_2}}$

$\xleftarrow{r_2}$

$\vdots$

$\xrightarrow{\boxed{f_n}}$ ✓/✗

A protocol for proving polynomial constraints

$\boxed{f}$: "oracles" to the function f

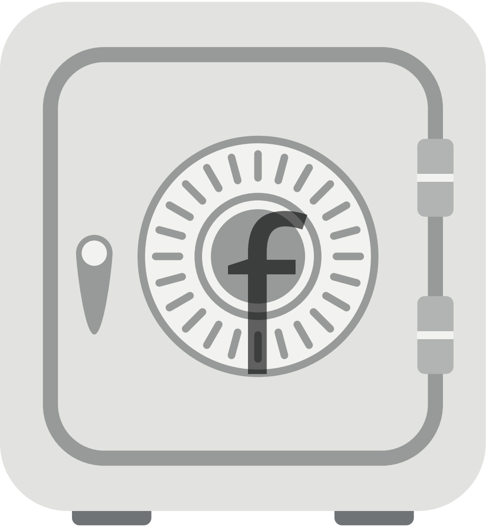
$\mathcal{V}$

What is $f(\alpha)$?

$f(\alpha) = z$

KZG polynomial commitment scheme [KZG10]

- A polynomial commitment scheme "instantiates" the oracle $\boxed{f}$.
- A polynomial commitment scheme allows a prover to
  - commit to a polynomial f



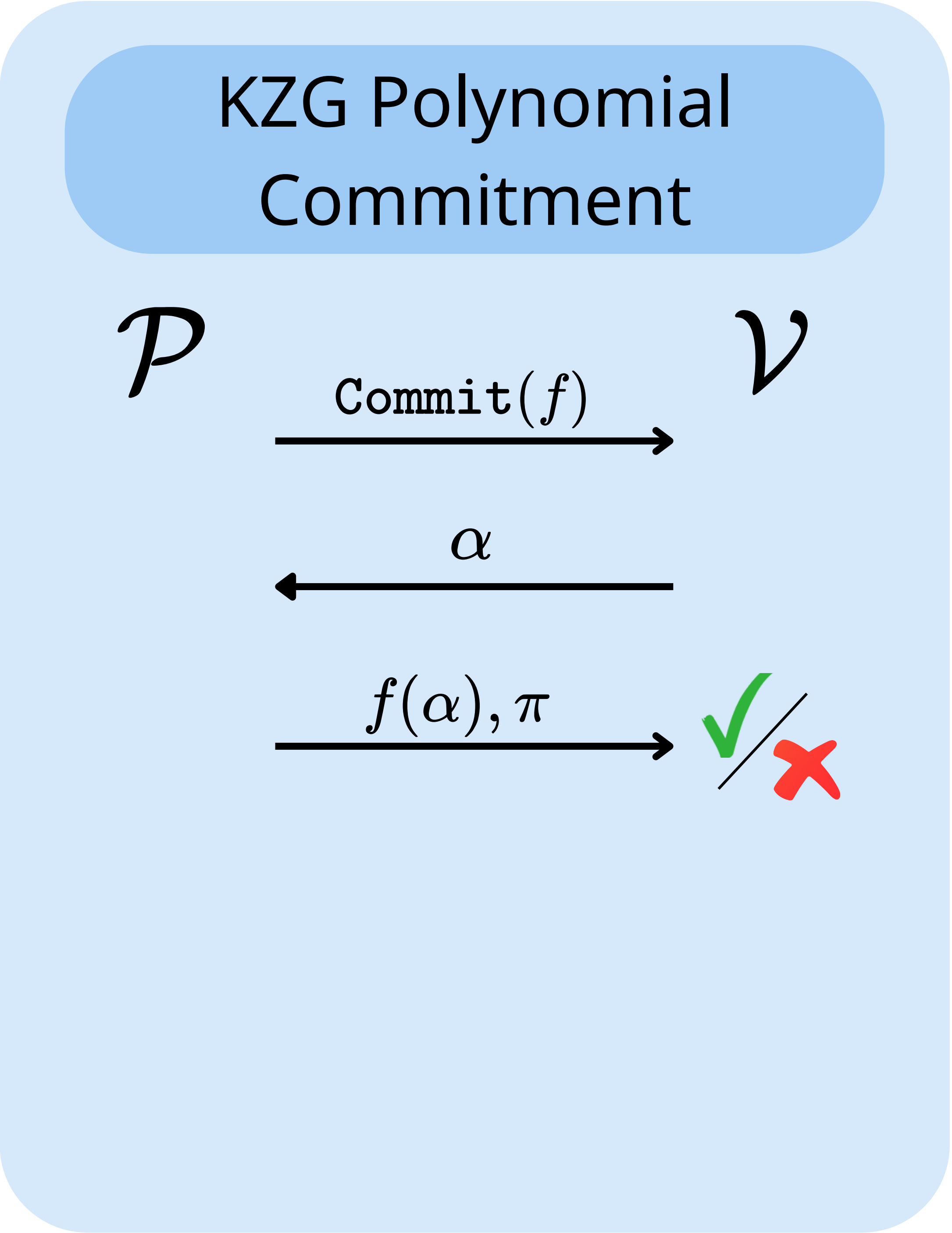
Committing f: "locking f in a box"

KZG Polynomial Commitment

$$\mathcal{P} \xrightarrow{\texttt{Commit}(f)} \mathcal{V}$$

KZG polynomial commitment scheme [KZG10]

- A polynomial commitment scheme "instantiates" the oracle $\boxed{f}$.
- A polynomial commitment scheme allows a prover to
  - commit to a polynomial f
  - later reveal its value at some point to the verifier while providing an evaluation proof.



Plonk [GWC19]

Plonk Arithmetization

$a_0 + b_0 = o_0$

$a_1 \cdot b_1 = o_1$

$o_0 = b_1$

$\vdots$

Circuit constraints

$g(X), p_0(X)$ and $p_1(X)$ vanish on Ω_X

Polynomial constraints

Plonk Polynomial IOP

$\mathcal{P}$ $\quad$ $\mathcal{V}$

$\xrightarrow{\boxed{f_1}}$

$\xleftarrow{r_1}$

$\xrightarrow{\boxed{f_2}}$

$\xleftarrow{r_2}$

$\vdots$

$\xrightarrow{\boxed{f_n}}$ ✓/✗

A protocol for proving polynomial constraints

KZG Polynomial Commitment

$\mathcal{P}$ $\quad$ $\mathcal{V}$

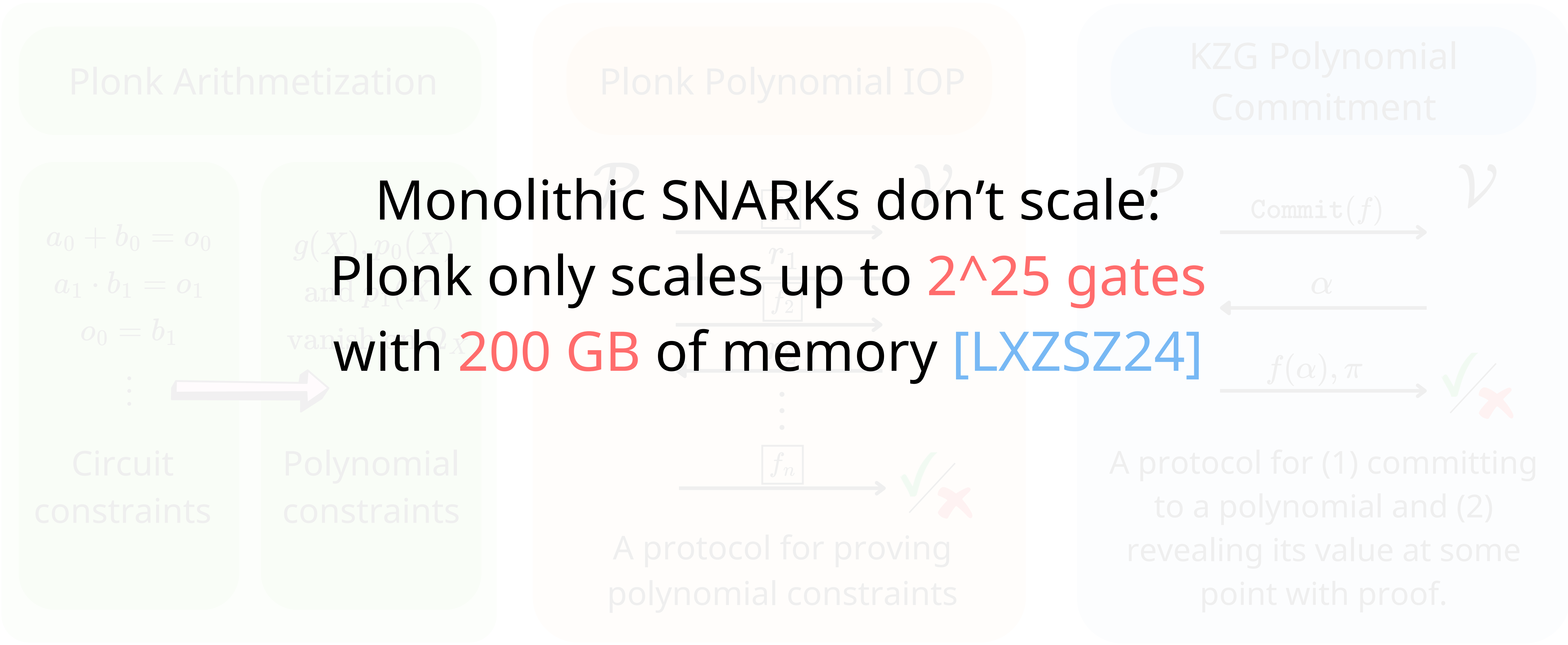
$\xrightarrow{\texttt{Commit}(f)}$

$\xleftarrow{\alpha}$

$\xrightarrow{f(\alpha), \pi}$ ✓/✗

A protocol for (1) committing to a polynomial and (2) revealing its value at some point with proof.

Plonk [GWC19]

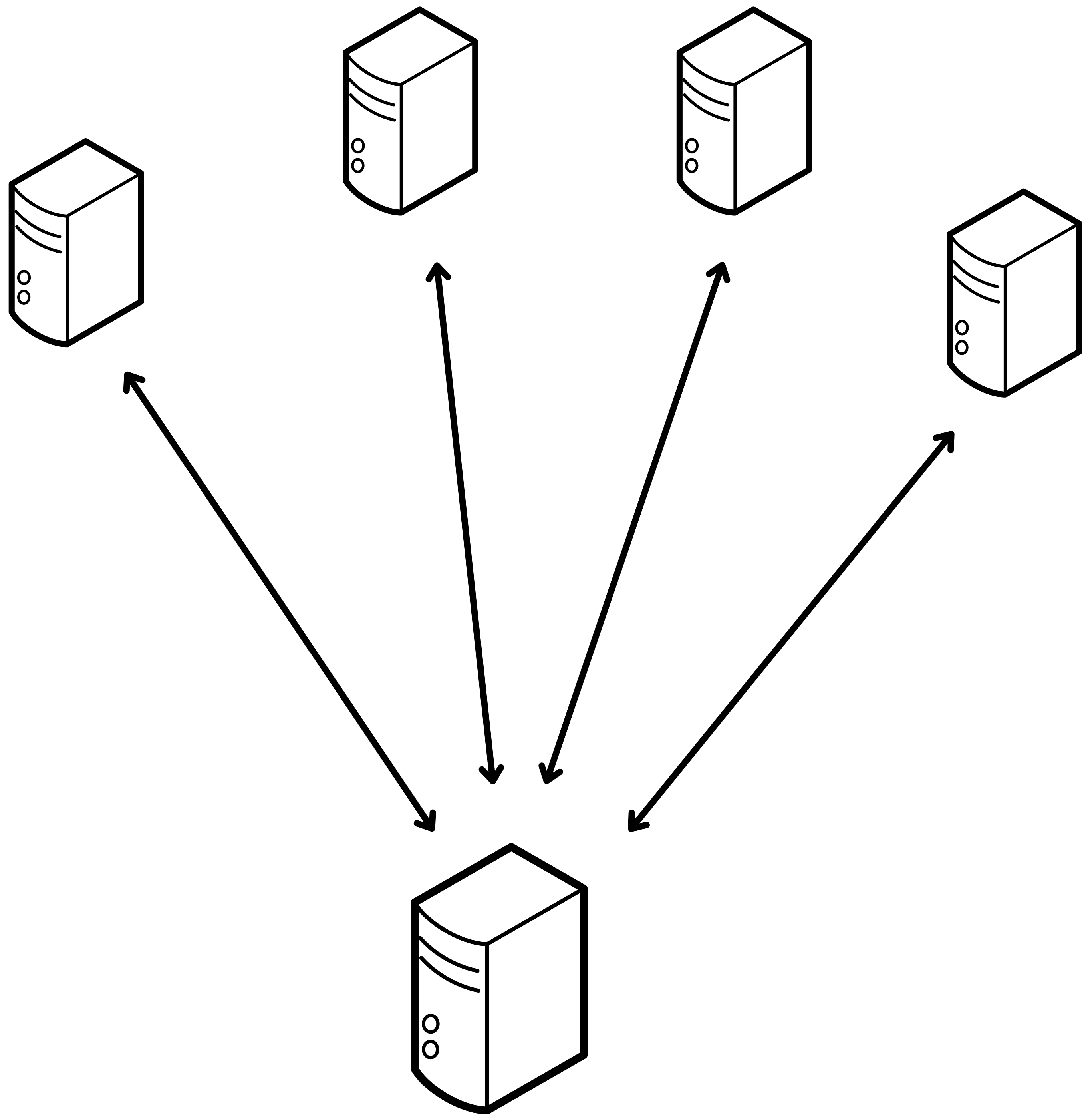


Monolithic SNARKs don't scale:
Plonk only scales up to 2^25 gates
with 200 GB of memory [LXZSZ24]

Pianist [LXZSZ24]

- A distributively computable version of Plonk
- One master node + several (2 ~ 64) worker nodes
- Divide a larger circuit evenly among worker nodes

	Plonk	Pianist
Proving time	$O(N \log N)$	$O(T \log T)$
Communication		$O(M)$
Proof size	$O(1)$	$O(1)$



$N=$ circuit size , $M=$ number of machines, $T=\dfrac{N}{M}=$ sub-circuit size ,

Pianist [LXZSZ24]

Pianist's Bivariate Arithmetization

$a_0 + b_0 = o_0$

$a_1 \cdot b_1 = o_1$

$o_0 = b_1$

$\vdots$

Circuit constraints

$g(X, Y), \ldots$
$p_3(X, Y)$ vanish on $\Omega_X \times \Omega_Y$

Polynomial constraints

Pianist's Bivariate Polynomial IOP

$\mathcal{P}$ $\qquad$ $\mathcal{V}$

$\xrightarrow{\boxed{F_1}}$

$\xleftarrow{r_1}$

$\xrightarrow{\boxed{F_2}}$

$\xleftarrow{r_2}$

$\vdots$

$\xrightarrow{\boxed{F_n}}$

A protocol for proving polynomial constraints

Distributed KZG

$\mathcal{P}$ $\qquad$ $\mathcal{V}$

$\xrightarrow{\texttt{Commit}(F)}$

$\xleftarrow{\alpha, \beta}$

$\xrightarrow{F(\alpha, \beta), \pi}$

A protocol for (1) committing to a polynomial and (2) revealing its value at some point with proof.

KZG [KZG10] vs. FRI-based polynomial commitment scheme [VP19]

KZG Polynomial Commitment

- $O(1)$ proof size*
- Requires trusted setup
- not post-quantum secure

FRI-based Polynomial Commitment

- Polylogarithmic proof size*
- No trusted setup
- Plausibly post-quantum

*: with respect to the number of gates

FRI low-degree test [BBHR18]

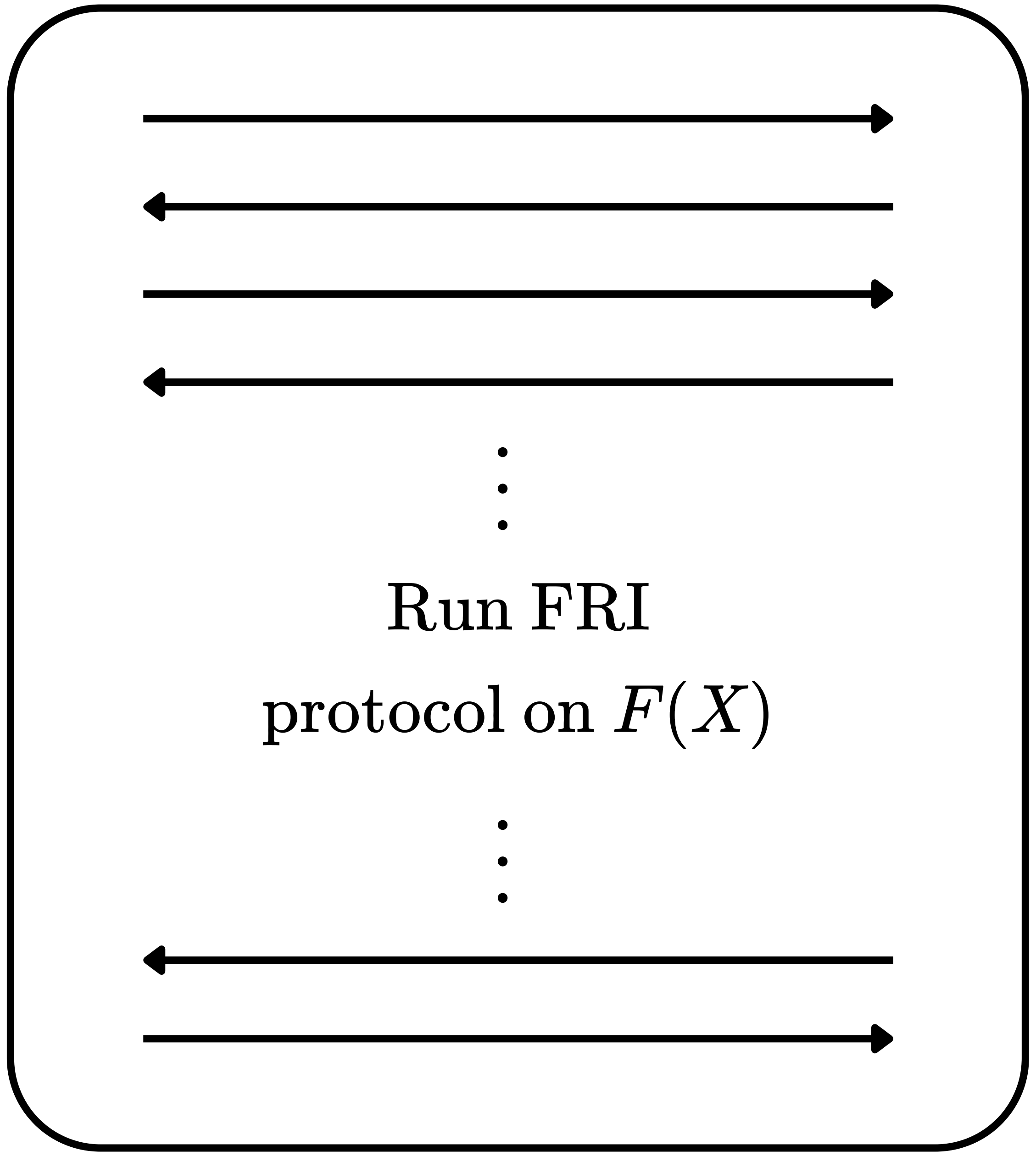
$\boxed{F}$

$\mathcal{P}$

F is close to polynomials of degree < N

$\mathcal{V}$

Prove it to me!



A protocol for proving F(X) is close to polynomials of deg < N

FRI-based polynomial commitment scheme [VP19]

$\mathcal{P}$ $\quad\quad\quad\quad$ $\mathcal{V}$

$F(\alpha) = z$

What is $F(\alpha)$?

$$\xrightarrow{\texttt{Commit}(F)}$$

$$\xleftarrow{\alpha}$$

$$\xrightarrow{z}$$

Run FRI on $\dfrac{F(X) - z}{X - a}$

Useful fact:

$$\begin{array}{c} F(X) \in \mathbb{F}[X]_{<N} \\ \text{with } F(\alpha) = z \end{array} \iff \frac{F(X) - z}{X - a} \in \mathbb{F}[X]_{<N-1}$$

This paper

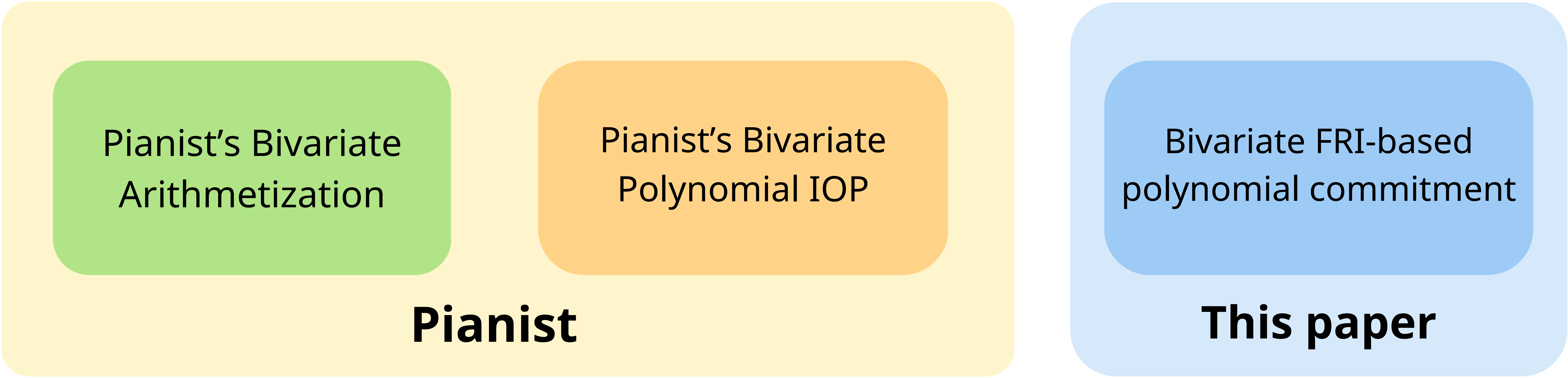
Pianist's Bivariate Arithmetization

Pianist's Bivariate Polynomial IOP

Pianist

Question: Can we construct a bivariate FRI-based polynomial commitment scheme that is efficient and horizontally scalable?

This paper



Question: Can we construct a bivariate FRI-based polynomial commitment scheme that is efficient and horizontally scalable?

DFRI (Distributed FRI)

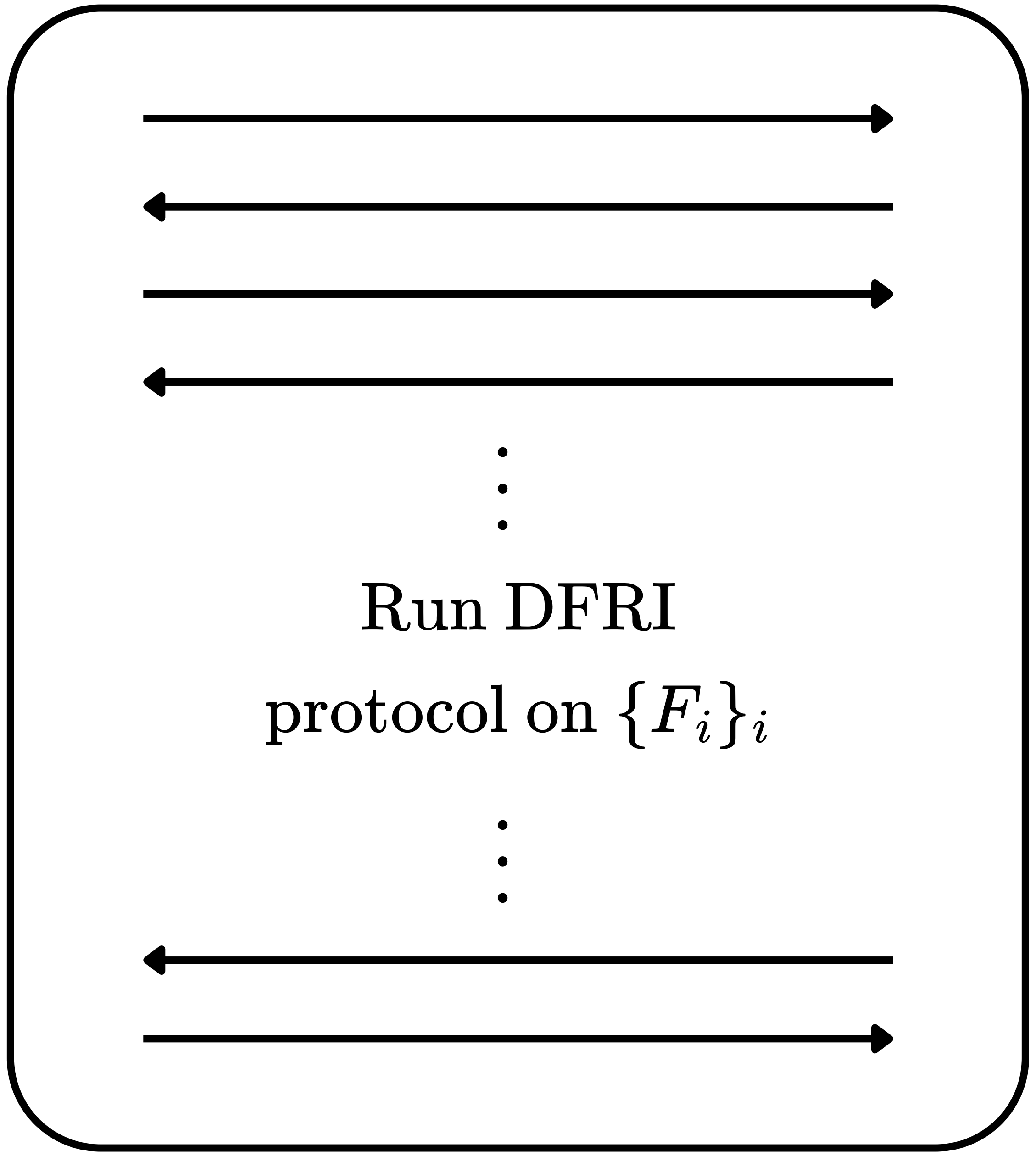
$\boxed{F_0}, \ldots, \boxed{F_{M-1}}$

$\mathcal{P}$

F_i's are close to polynomials of degree < N

$\mathcal{V}$

Prove it to me!



A protocol for proving $F_0(X), \ldots, F_{M-1}(X)$ are close to polynomials of degree $< N$

Bivariate FRI-based PCS: Commit

$$F(X,Y) = \sum_{i=0}^{M-1} F_i(X) R_i(Y)$$

$F_i(X)$ is held by $\mathcal{P}_i$

$R_i(Y)$ is public

$$\mathcal{P} \xrightarrow{\quad \boxed{F_0}, \ldots, \boxed{F_{M-1}} \quad} \mathcal{V}$$

The prover commits to the polynomial $F(X,Y)$ by sending the verifier Merkle commitments of $F_0, \ldots, F_{M-1}$

Bivariate FRI-based PCS: Open (Simplified version)

$\boxed{F_0}, \ldots, \boxed{F_{M-1}}$

$$F(X,Y) = \sum_{i=0}^{M-1} F_i(X) R_i(Y)$$

$\mathcal{P}$ $\qquad$ $\mathcal{V}$

$\xleftarrow{\quad \alpha \quad}$

$\xrightarrow{z_0, \ldots, z_{M-1}}$

$F_i(\alpha) = z_i$

Run DFRI on $\left\{ \dfrac{F_i(X) - z_i}{X - \alpha} \right\}_i$ ✓

Now I believe $F_i(\alpha) = z_i$

Computes

$$F(\alpha, \beta) = \sum_{i=0}^{M-1} z_i R_i(\beta)$$

Simplified version:

- Simpler protocol
- Verification: O(M^2) on top of verifying DFRI

Bivariate FRI-based PCS: Cost

- Prover:
  - O(TlogT) for commiting F(X, Y) + cost of proving DFRI
  - O(TlogT) is the cost for computing the evaluation vectors for FRI
- Verifier: O(M^2) + cost of verifying DFRI
  - O(M^2) is the cost for computing $F(\alpha, \beta) = \sum_{i=0}^{M-1} z_i R_i(\beta)$

T = size of sub-circuit for each worker

M = number of workers

Bivariate FRI-based PCS: Open (Complete version)

$\boxed{F_0}, \ldots, \boxed{F_{M-1}}$

$$F(X,Y) = \sum_{i=0}^{M-1} F_i(X) R_i(Y)$$

$\mathcal{P}$

$\mathcal{V}$

$F_i(\alpha) = z_i$

α, β ←

$z_0, \ldots, z_{M-1}$ →

Why should I believe you?

Run DFRI on $\left\{ \dfrac{F_i(X) - z_i}{X - \alpha} \right\}_i$

✓/✗

Complete version:

- Uses pre-computation
- Verification: O(M) on top of verifying DFRI

Bivariate FRI-based PCS: Open (Complete version)

$$F(X,Y) = \sum_{i=0}^{M-1} F_i(X) R_i(Y)$$

$\mathcal{P}$ $\mathcal{V}$

$F(\alpha, \beta) = z$

$\xrightarrow{\alpha, \beta}$

$\xleftarrow{z_0, \ldots, z_{M-1}}$

how do I check this?

Run DFRI on $\left\{ \dfrac{F_i(X) - z_i}{X - \alpha} \right\}_i$ ✓

$\xrightarrow{z}$

Complete version:
- Uses pre-computation
- Verification: O(M) on top of verifying DFRI

Bivariate FRI-based PCS: Open

$$F(X,Y) = \sum_{i=0}^{M-1} F_i(X) R_i(Y)$$

$\mathcal{P}$ $\mathcal{V}$

$F(\alpha, \beta) = z$

$\xrightarrow{\alpha, \beta}$

$\xleftarrow{z_0, \ldots, z_{M-1}}$

how do I check this?

Run DFRI on $\left\{ \dfrac{F_i(X) - z_i}{X - \alpha} \right\}_i$ ✓

$\xrightarrow{z}$

Useful fact:

$$F(\alpha, \beta) = z$$
$$\Updownarrow$$
$$F(\alpha, Y) - z = H(Y)(Y - \beta)$$
for some $H(Y) \in \mathbb{F}[Y]_{<M-1}$

Bivariate FRI-based PCS: Open

$$F(X,Y) = \sum_{i=0}^{M-1} F_i(X) R_i(Y)$$

$\mathcal{P}$ $\mathcal{V}$

$F(\alpha, \beta) = z$

$\mathcal{P} \xrightarrow{\alpha, \beta} \mathcal{V}$

$\mathcal{P} \xleftarrow{z_0, \ldots, z_{M-1}} \mathcal{V}$

I can check that myself!

Run DFRI on $\left\{ \dfrac{F_i(X) - z_i}{X - \alpha} \right\}_i$ ✓

$\mathcal{P} \xrightarrow{z} \mathcal{V}$

Useful fact:

$$F(\alpha, \beta) = z$$

$$\Updownarrow$$

$$F(\alpha, Y) - z = H(Y)(Y - \beta)$$

for some $H(Y) \in \mathbb{F}[Y]_{<M-1}$

Bivariate FRI-based PCS

- V needs to check $F(\alpha, Y) - z = H(Y)(Y - \beta)$ for some $H(Y) \in \mathbb{F}[Y]_{<M-1}$
- V already knows the evaluation of H at M different points:

$$H(\omega^i) = \frac{F(\alpha, \omega^i) - z}{\omega^i - \beta} = \frac{z_i - z}{\omega^i - \beta}, \ i = 0, \ldots, M - 1$$

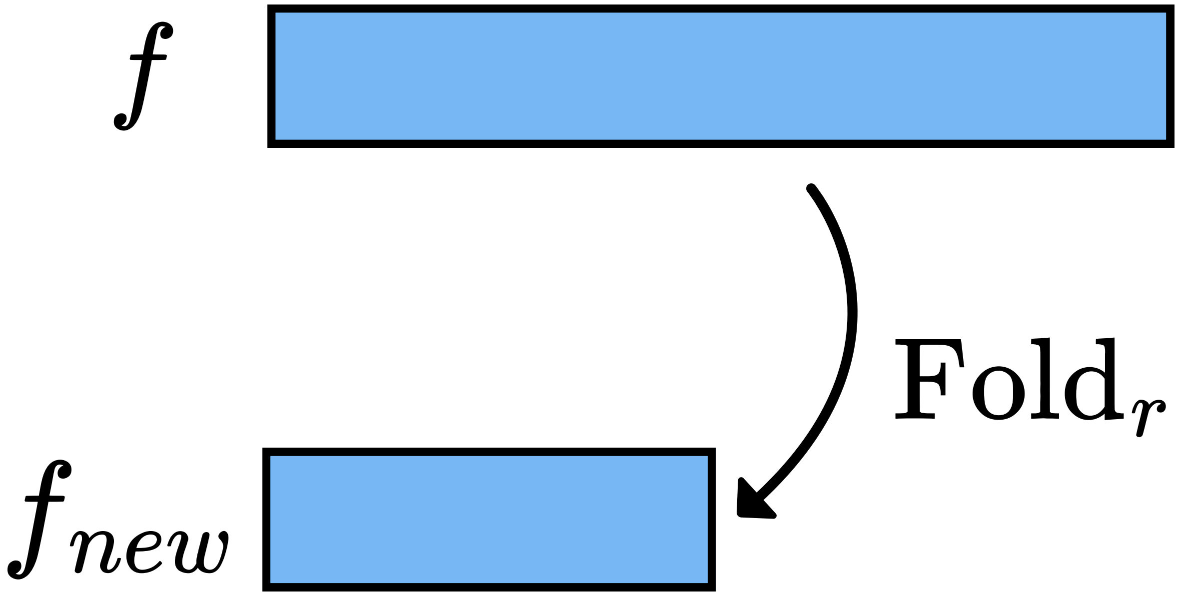
- V constructs H(Y) by "interpolating" M - 1 points
- V checks that indeed $F(\alpha, Y) - z = H(Y)(Y - \beta)$ by evaluating at ω^i

Cost O(M) operations

DFRI (Distributed FRI)

- Problem: How to prove multiple instances of FRI distributively?
- Setup:
  - Each FRI instance f_i(X) is stored by a worker node.
  - Together, they need to efficiently prove that all instances are low-degree.
- Ideally, we want:
  - Prover time and memory costs to be small.
  - Communication between prover nodes are small.
  - Prover has horizontal scalability, i.e. prover time / memory costs decreases linearly as we increase the number of machines.
  - Short verification time and small proof sizes.

FRI low-degree test [BBHR18]: Folding operation

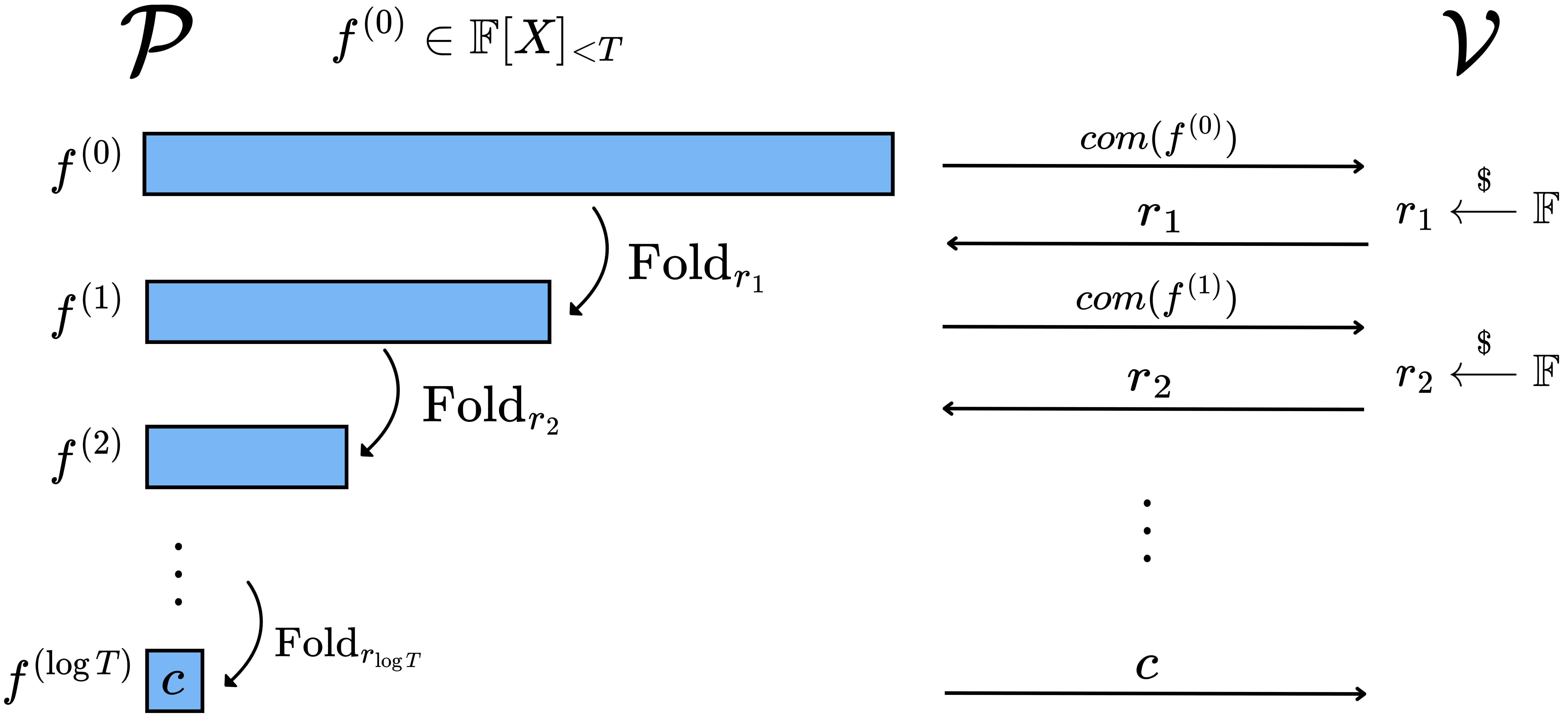


$$f(X) = \underbrace{f_{even}(X^2)}_{\text{even degree terms}} + \underbrace{X \cdot f_{odd}(X^2)}_{\text{odd degree terms}}$$

$$f_{new}(X) = f_{even}(X) + r \cdot f_{odd}(X)$$

- **Fold_r**:
  - A “folding” operation on polynomials
  - Uses a verifier-provided random value r
- **Degree reduction**:
  - If a polynomial f has degree < d, then the new polynomial has degree < d / 2.

FRI low-degree test [BBHR18]: Folding phase



FRI low-degree test [BBHR18]: Query phase

$\mathcal{P}$ $\qquad f^{(0)} \in \mathbb{F}[X]_{<8}$ $\qquad \mathcal{V}$

$f^{(0)}$ $\boxed{g}$ $\boxed{-g}$

FRI query set Q $\longleftarrow$

$f^{(1)}$ $\boxed{g^2}$ $\boxed{-g^2}$

$f^{(2)}$ $\boxed{g^4}$ $\boxed{-g^4}$

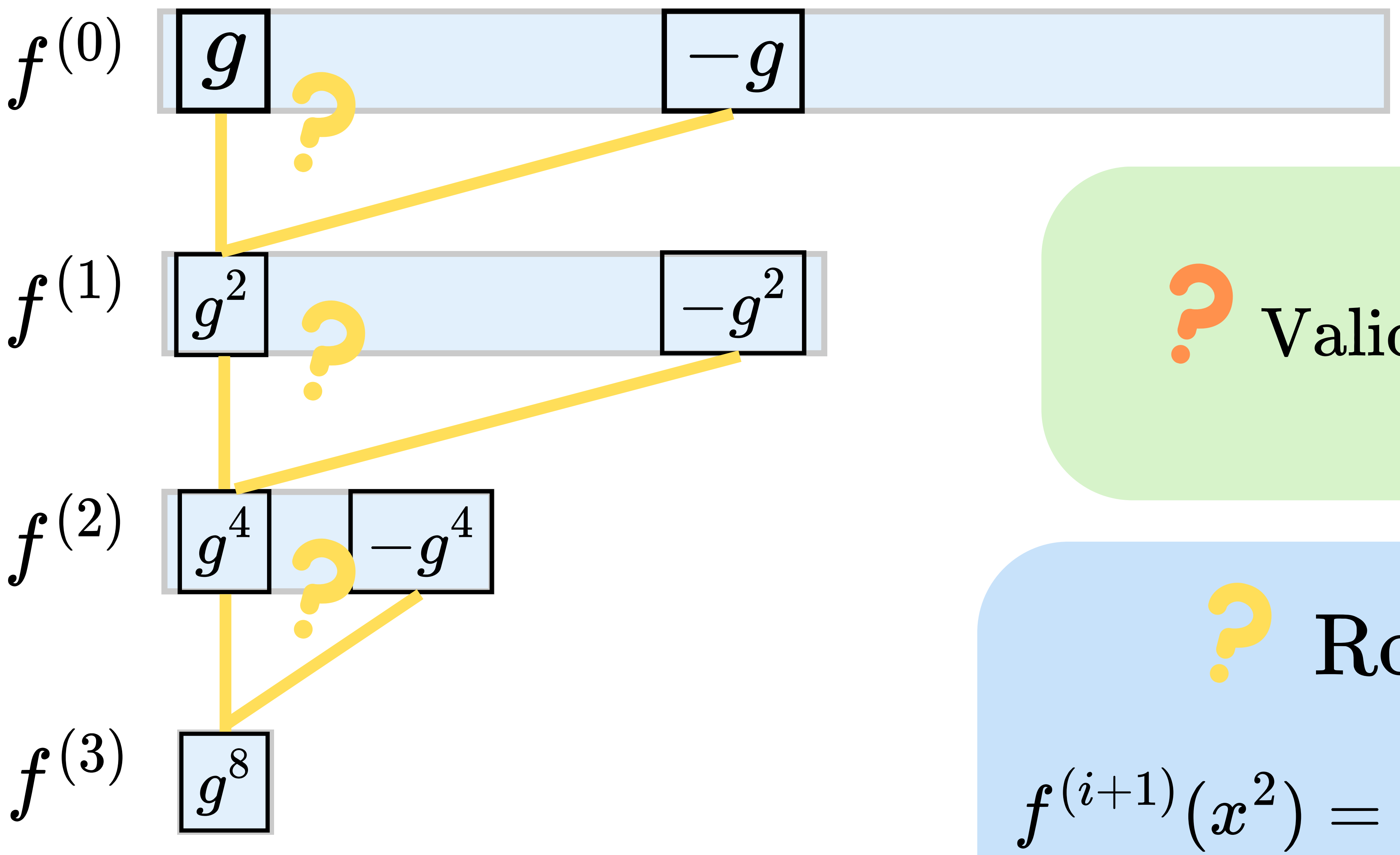
$f^{(3)}$ $\boxed{g^8}$

$f^{(j)}(\pm g^{2^j})$ and opening proofs

for all j and for all $g \in Q$ $\longrightarrow$

FRI low-degree test [BBHR18]: Query phase

$\mathcal{P}$ $\quad f^{(0)} \in \mathbb{F}[X]_{<T}$ $\quad \mathcal{V}$



Validity of Merkle opening proofs

Round consistency checks:

$$f^{(i+1)}(x^2) = \frac{x + r_{i+1}}{2x} f^{(i)}(x) + \frac{x - r_{i+1}}{2x} f^{(i)}(-x)$$

Batched FRI [BCI+20]

$\mathcal{P}$

$\boxed{f_0}, \boxed{f_1}, \ldots, \boxed{f_{M-1}}$

$\mathcal{V}$

$\theta \xleftarrow{\$} \mathbb{F}$

$\xleftarrow{\quad\theta\quad}$

$$f := \sum_{i=0}^{M-1} \theta^i f_i$$

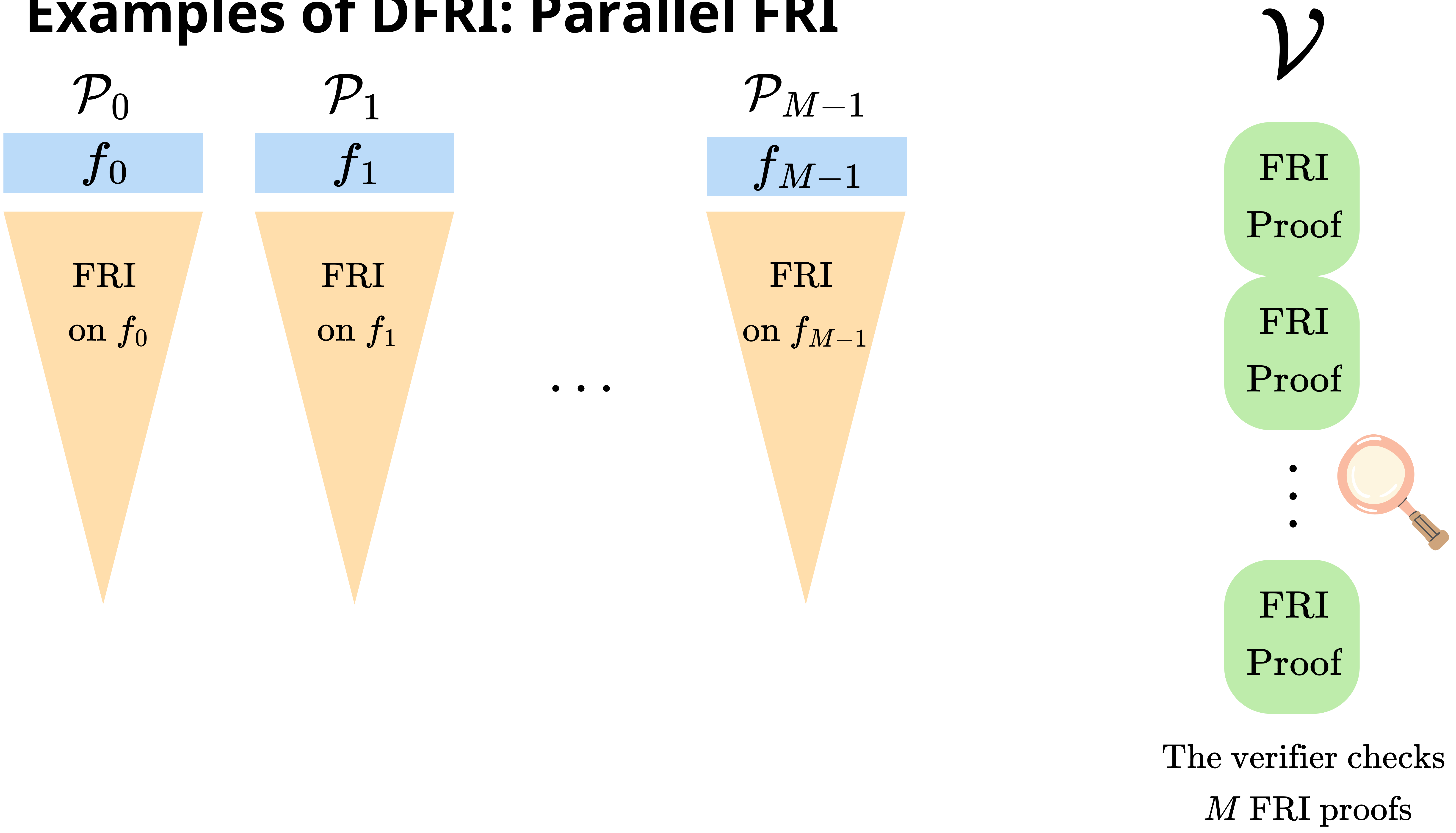
Run FRI on f

$f_i(x)$ for all $x \in Query$
$+$ opening proofs $\longrightarrow$

Linear combination check: $f(x) = \sum_i \theta^i f_i(x)$ for each $x \in Query$

Batched FRI soundness: If the verifier accepts, then with overwhelming probability, all f_i's are (close to) low degree polynomials.

Examples of DFRI: Parallel FRI

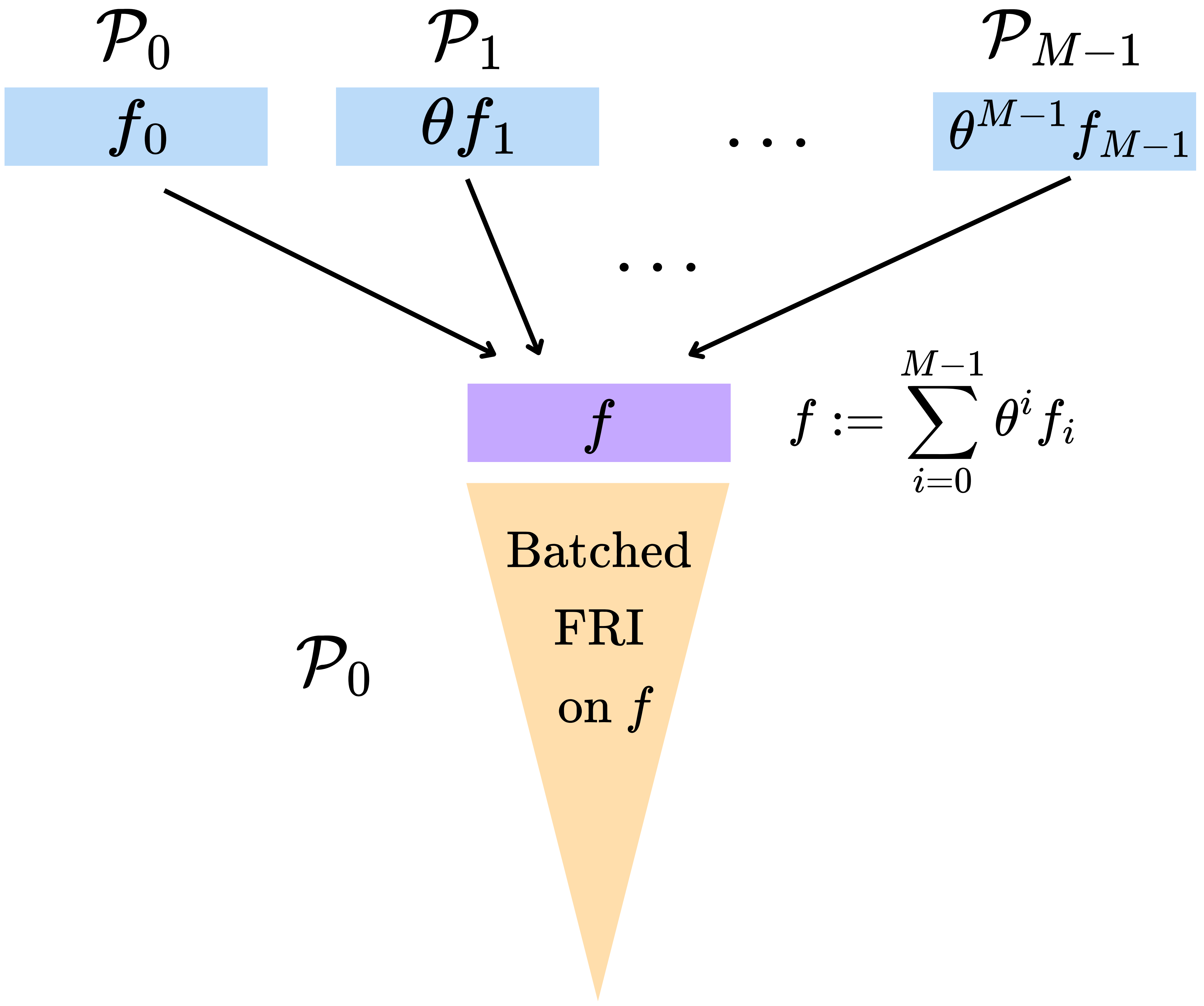


Examples of DFRI: Parallel FRI

- Method:
  - Each worker node runs a separate instance of FRI.
  - No master node needed.
  - The verifier checks M FRI proofs where M = num of machines.
- Efficiency:
  - Perfectly scalable in terms of prover time/memory.
  - No communication between prover nodes.
  - Most expensive for the verifier among DFRI methods.

Examples of DFRI: Distributed Batched FRI [=nil; Research]

$\mathcal{V}$

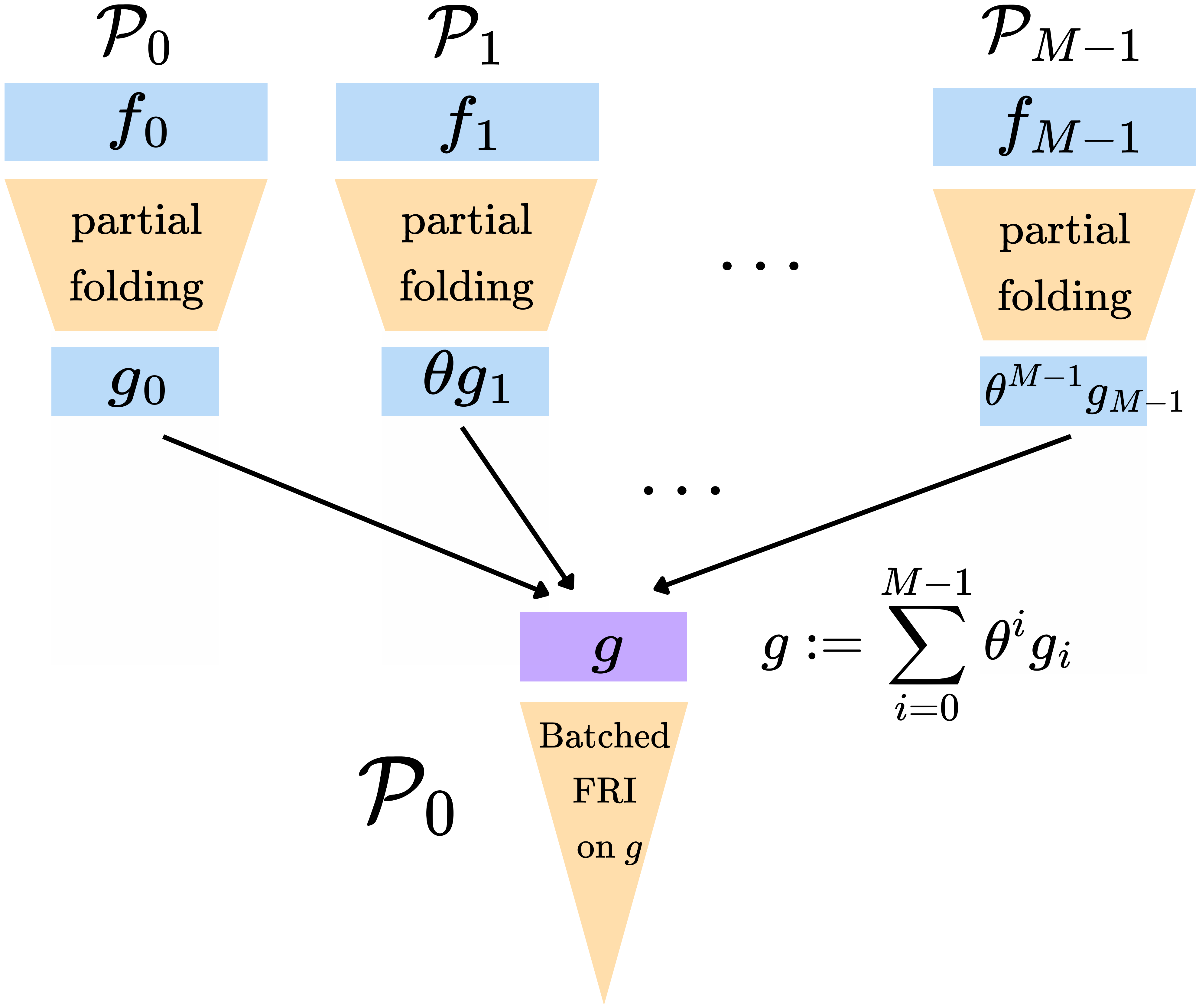


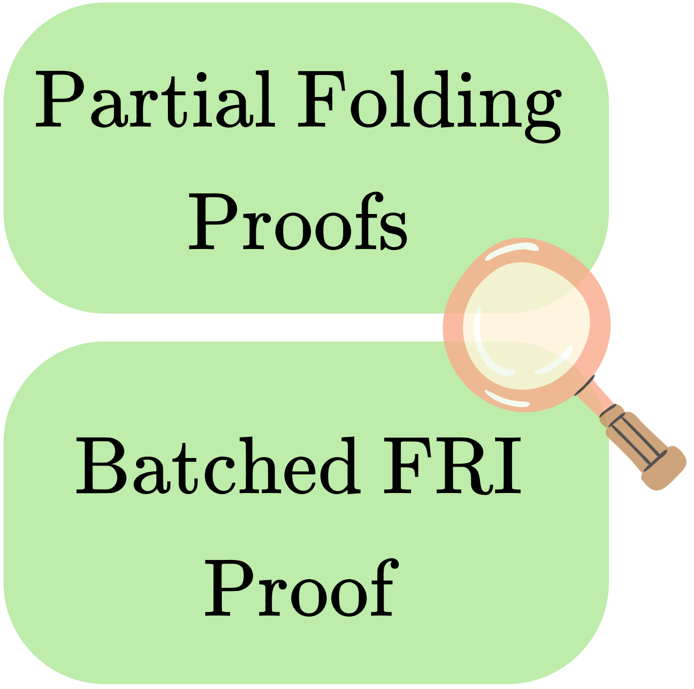
The verifier checks the batched FRI proof

Examples of DFRI: Distributed Batched FRI [=nil; Research]

- Method:
  - Each worker node maintains S small witnesses.
  - The worker nodes batch all their local witnesses using a random challenge from the verifier.
  - Each worker sends the batched eval vector to the master node.
  - The master node performs batched FRI.
  - In this work, we focus on the case where S = 1 to match the setup for the Pianist arithmetization.
- Efficiency:
  - Small proof size: a single batched FRI proof.
  - Large communication cost (linear in instance size) regardless of number of worker nodes.
  - Large memory cost not relieved by increasing the number of workers.

Examples of DFRI: Fold-and-Batch (This paper) $\mathcal{V}$





The verifier checks the partial folding proofs and the batched FRI proof

Examples of DFRI: Fold-and-Batch (This paper)

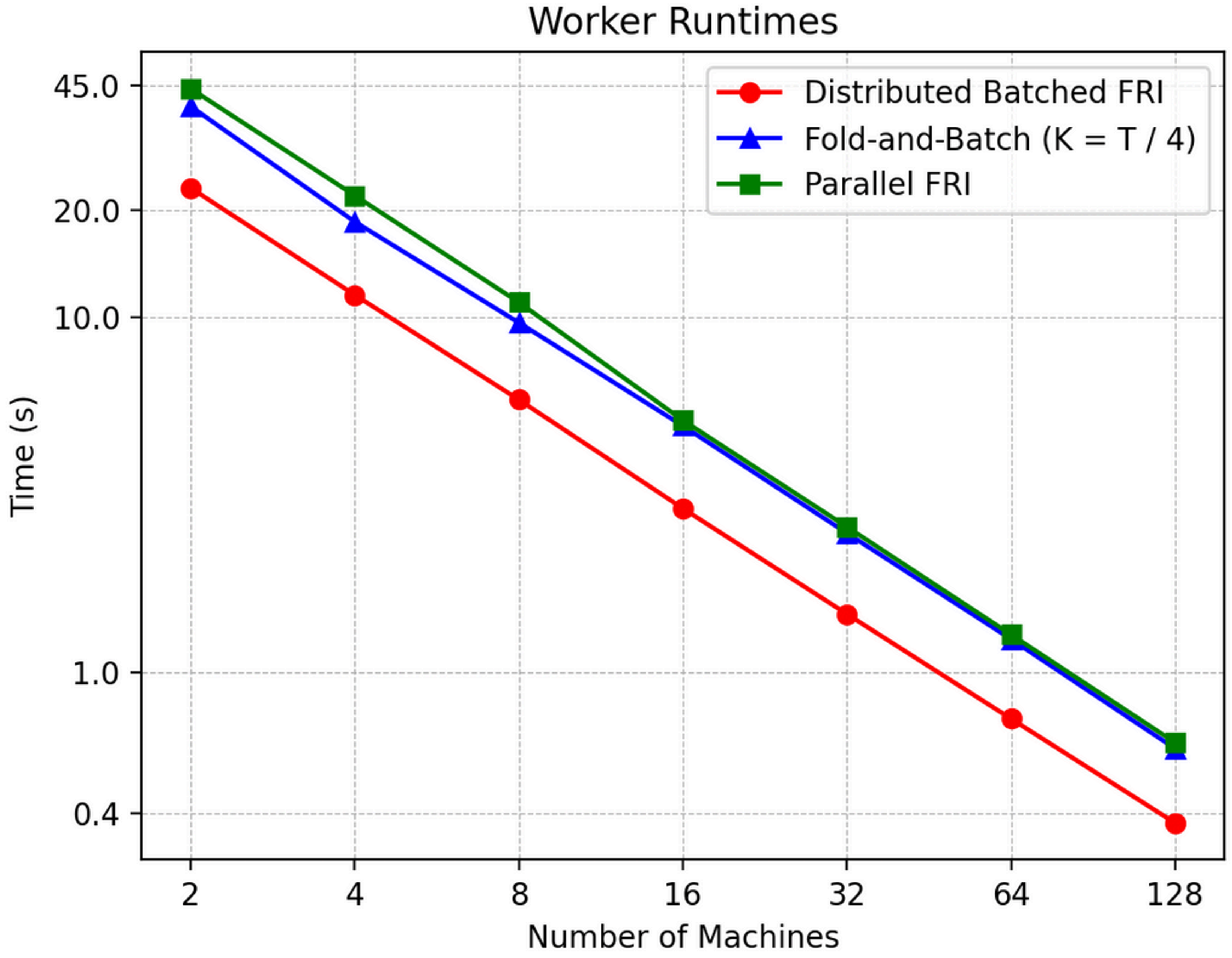
- A combination of local FRI folding and distributed batched FRI.
- Method:
  - Each worker node folds its local eval vector for a few rounds (adjustable).
  - Then proceed as in distributed batched FRI.
- Efficiency (compared to Distributed Batched FRI):
  - Larger proof size, as the proofs from local FRI foldings are not batched.
  - Smaller communication and memory costs for the prover.
  - Can be adjusted to have an efficiency profile closer to either Parallel FRI or Distributed Batched FRI.

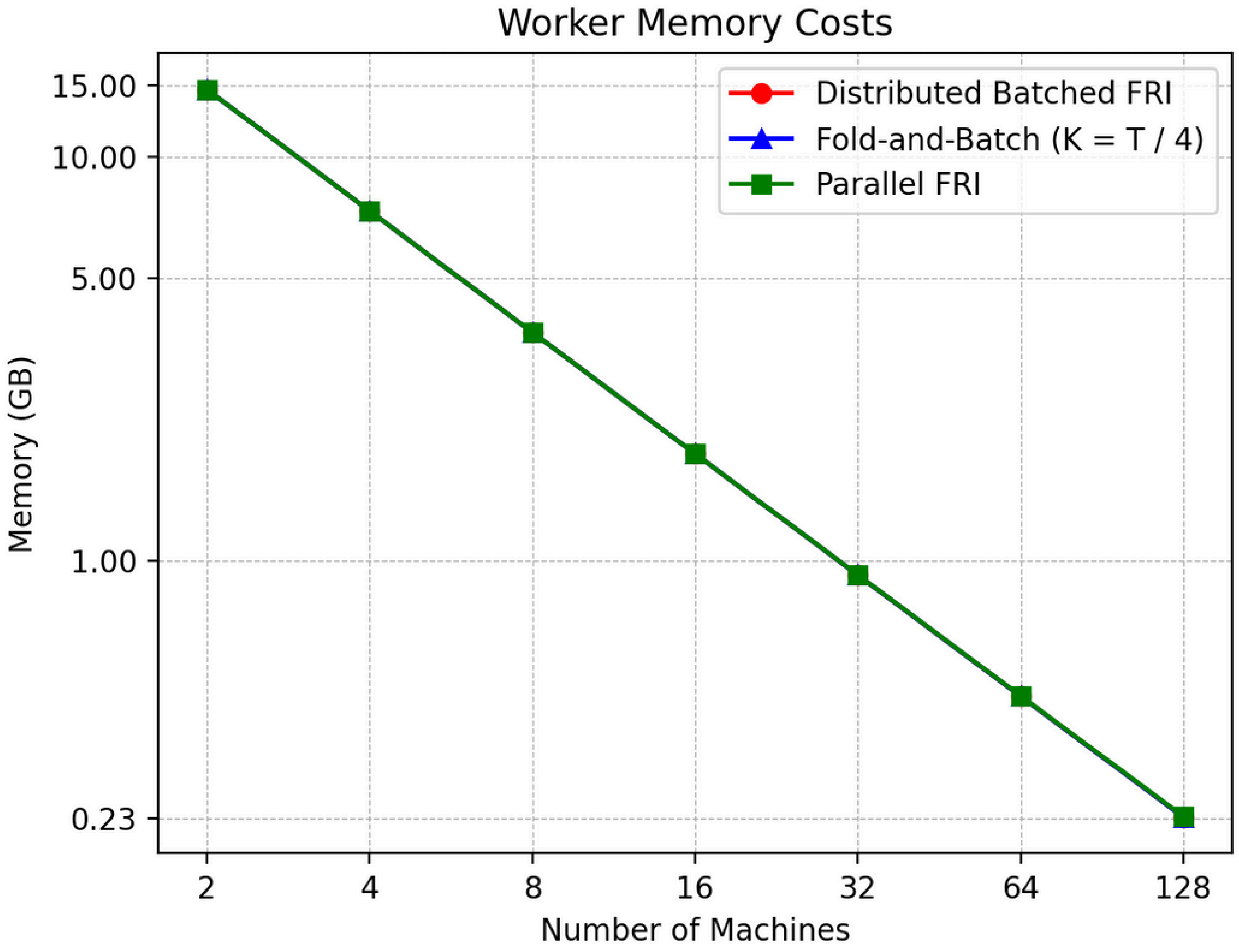
Implementation & Evaluation

- Implementation:
    - Based on **winterfell**, a STARK implementation written in Rust from Facebook.
- Setup:
    - Simulated all the prover nodes on a single server with 377 GiB RAM.
    - Runtime does not include time spent on communication.
    - Communication cost is computed and shown separately.
    - Merkle trees are instantiated with BLAKE3 with 256-bit output.

Performance: Worker Nodes

Circuit size $= 2^{25}$

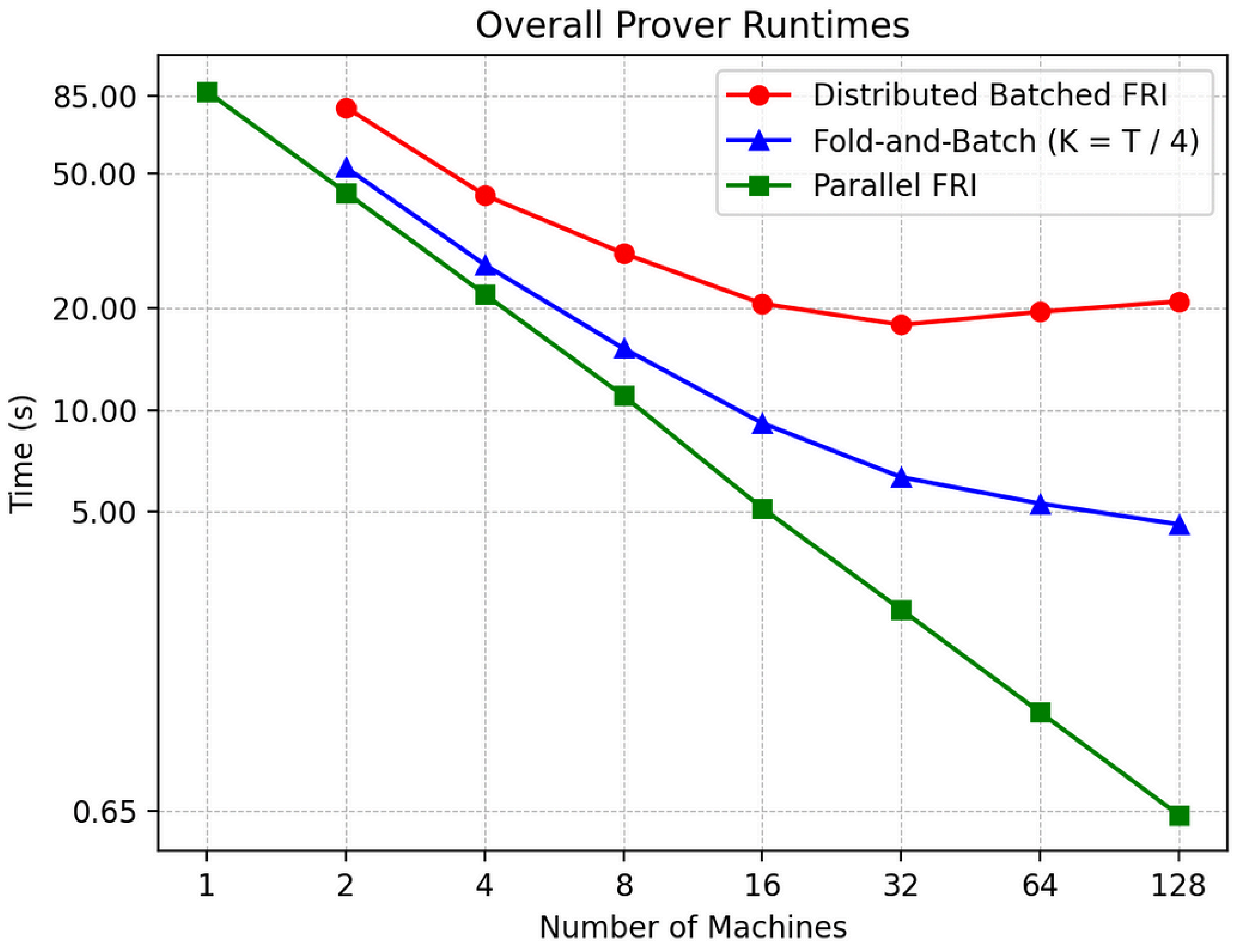


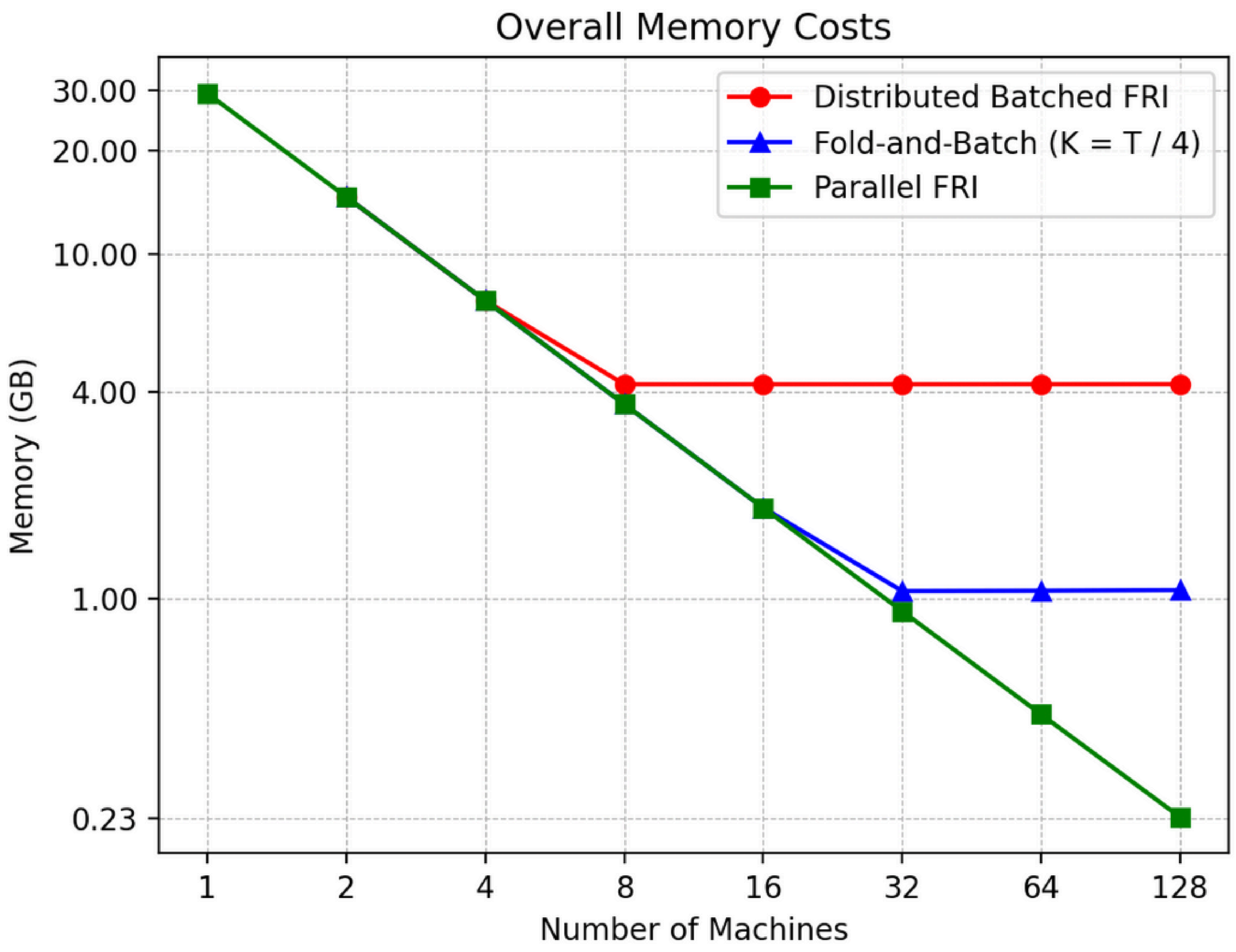


Worker runtime and memory costs are horizontally scalable.

Performance: Prover Overall

Circuit size $= 2^{25}$

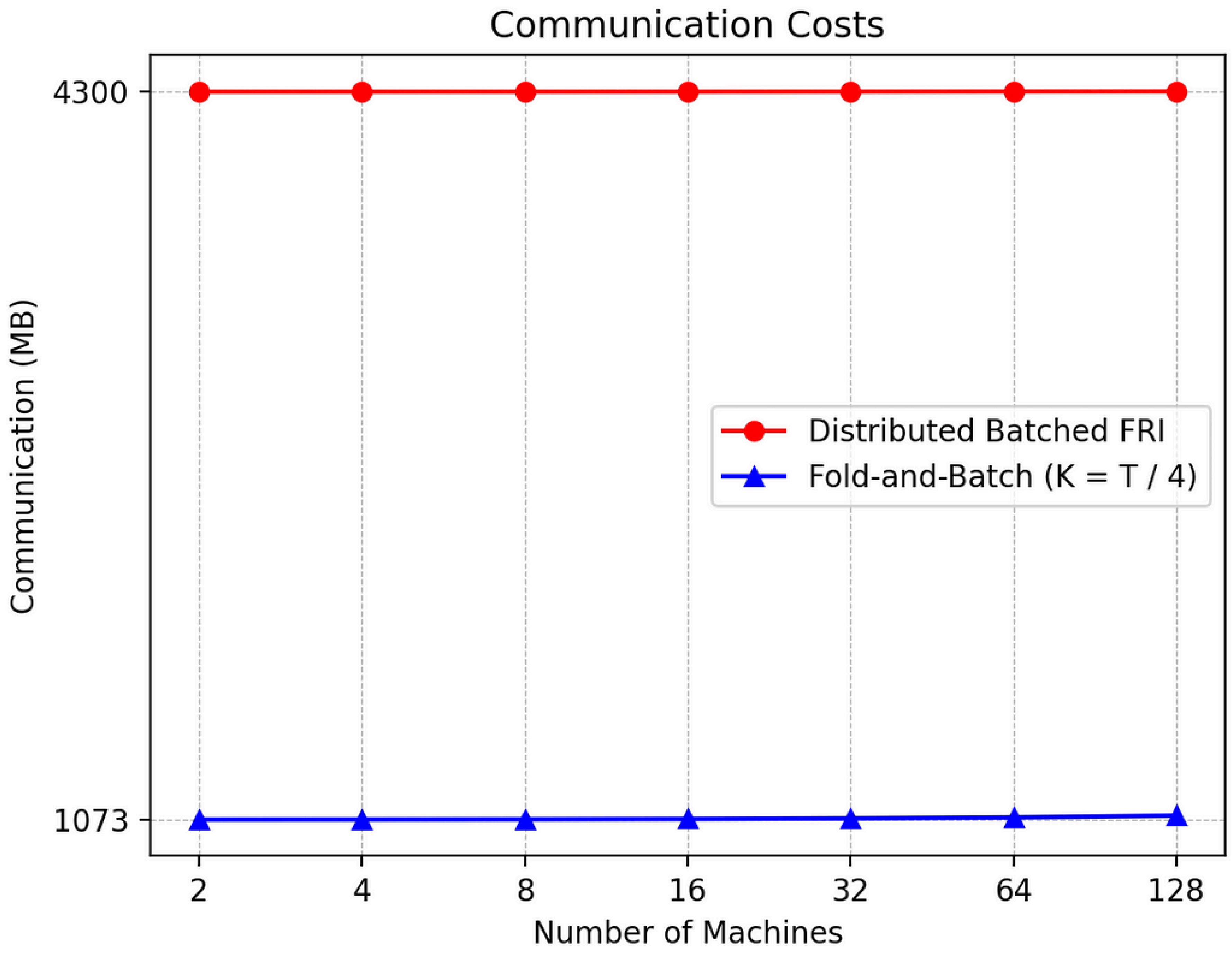




The overall prover runtime and memory costs have limited scalability.

Communication

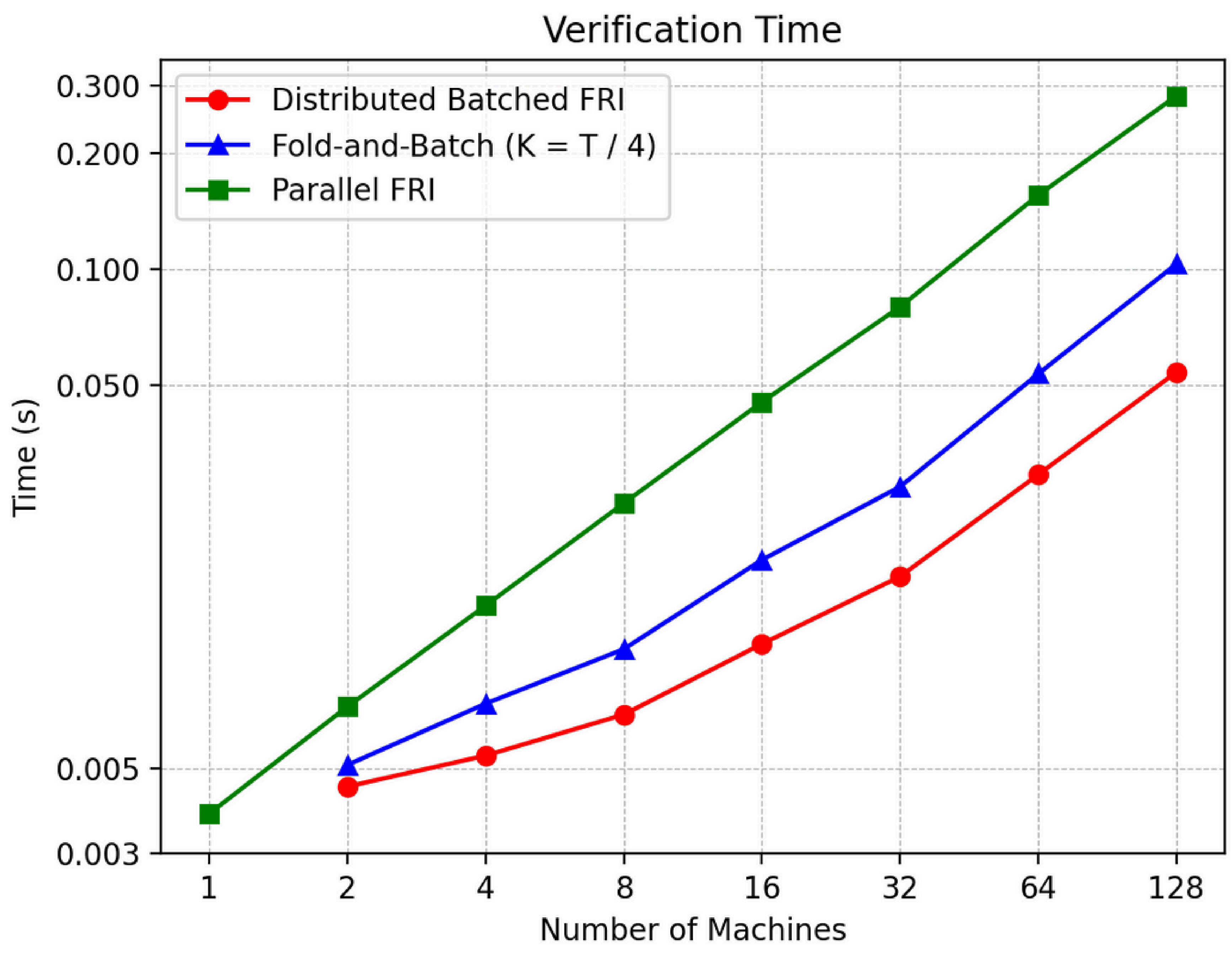
Circuit size $= 2^{25}$

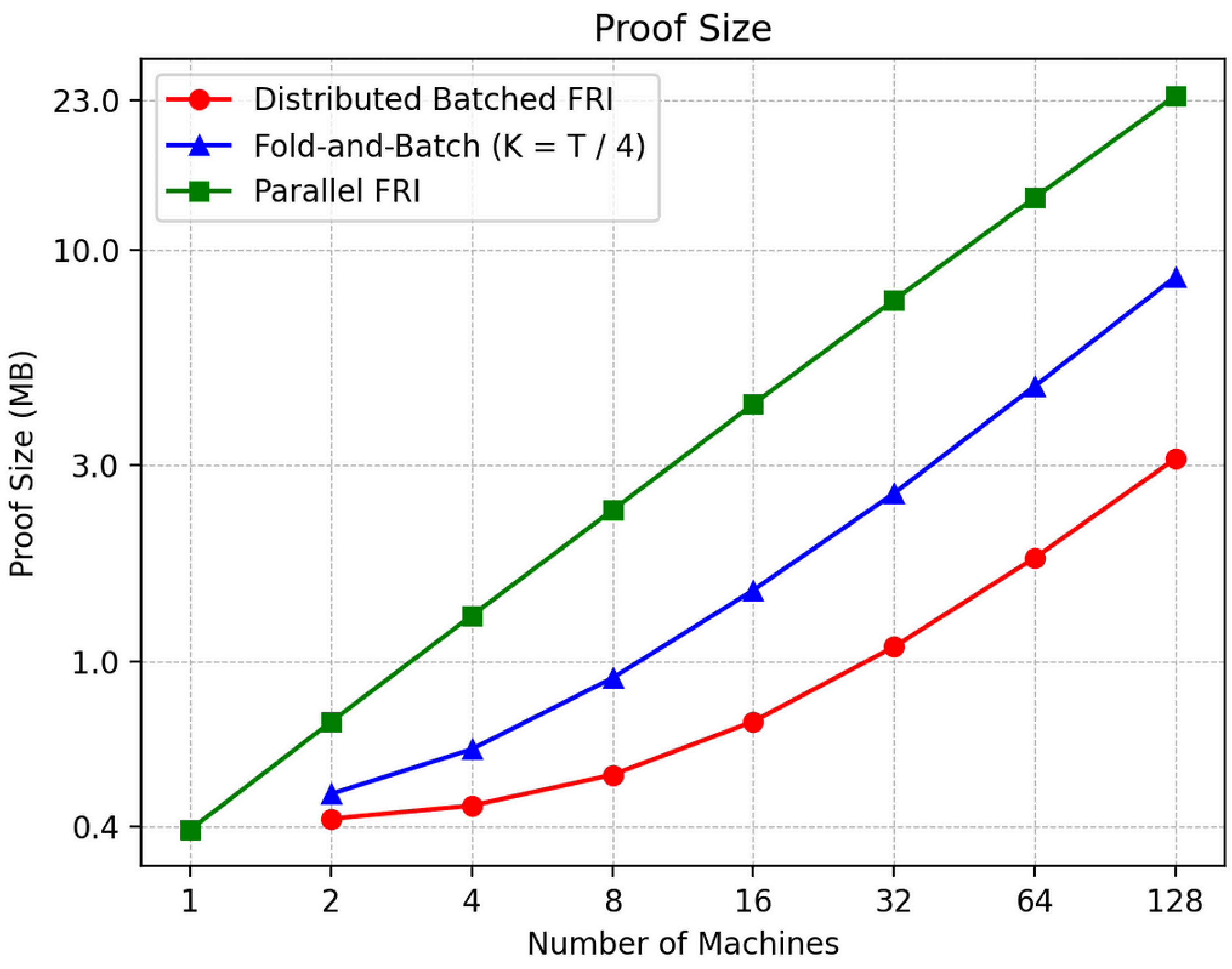


- Distributed Batched FRI:
  - Constant amount of ~4GB of communication
  - From all the uncompressed eval vectors sent from all worker nodes
- Fold-and-Batch:
  - Constant amount of ~1GB of communication
  - Eval vectors are folded by a factor of 4 and sent from all worker nodes

Performance: Verification

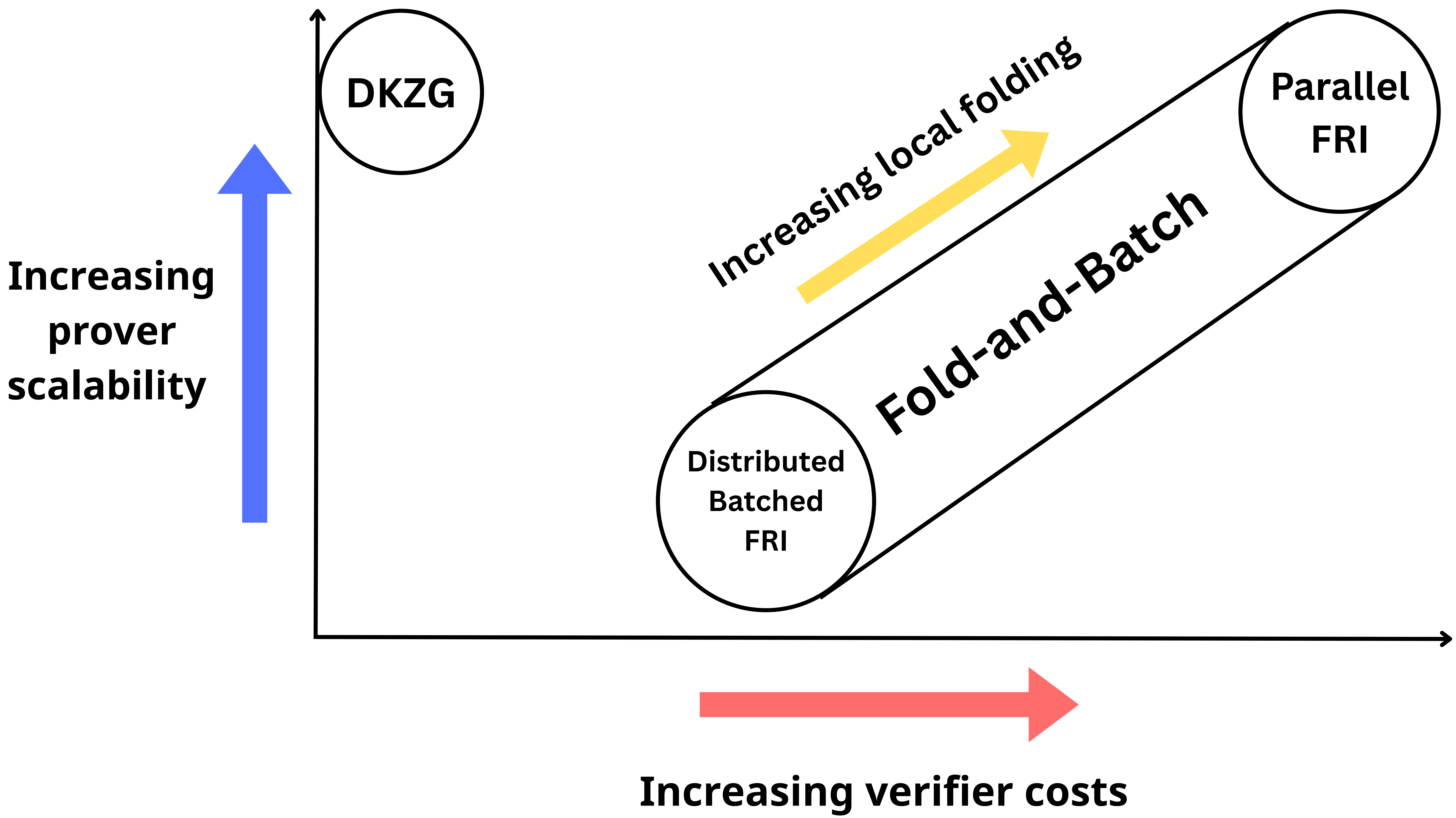
Circuit size $= 2^{25}$





- Verifier Costs: Parallel FRI > Fold-and-Batch > Distributed Batched FRI
- **Verification time and proof size increase with the number of workers for all methods.**

Prover/Verifier Cost Trade-off



Scalability bottlenecks

- **Bottleneck 1: Master runtimes/memory costs not linearly scalable.**
  - This happens for Distributed batched FRI and Fold-and-Batch.
  - Due to the handling of uncompressed local witnesses.

- **Bottleneck 2: Verifier costs increase with the number of machines.**
  - This happens for all three methods.
  - For batched FRI, this is due to the lack of homomorphism for Merkle tree commitments:
    - Batched FRI proof = commitments + opening proofs
    - Number of opening proofs for each FRI query: $M + \log T$

References

- **[GWC19]** Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Per- mutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953. 2019. url: https://eprint.iacr.org/2019/953.
- **[KZG10]** Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Advances in Cryptology – ASIACRYPT 2010. pp. 177–194
- **[LXZSZ24]** Tianyi Liu et al. Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2023/1271. 2023. url: https://eprint.iacr.org/2023/1271.
- **[VP19]** Alexander Vlasov and Konstantin Panarin. Transparent Polynomial Commitment Scheme with Polylogarithmic Communication Complexity. Cryptology ePrint Archive, Paper 2019/1020. 2019. url: https://eprint.iacr.org/2019/1020.

References

- **[BBHR18]** Eli Ben-Sasson et al. “Fast reed-solomon interactive oracle proofs of proximity”. In: 45th international colloquium on automata, languages, and programming (icalp 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.
- **[BCI+20]** Eli Ben-Sasson et al. Proximity Gaps for Reed-Solomon Codes. Cryptology ePrint Archive, Paper 2020/654. 2020. url: https://eprint.iacr.org/2020/654.
- **[=nil; Research]** Alisa Cherniaeva. On Distributed FRI-based Proof Generation-HackMD. HackMD, Oct. 2024. url: https://hackmd.io/@nil-research/rJ_NVyiRA (visited on 01/24/2025).