Cirrus: Performant and Accountable Distributed SNARK*

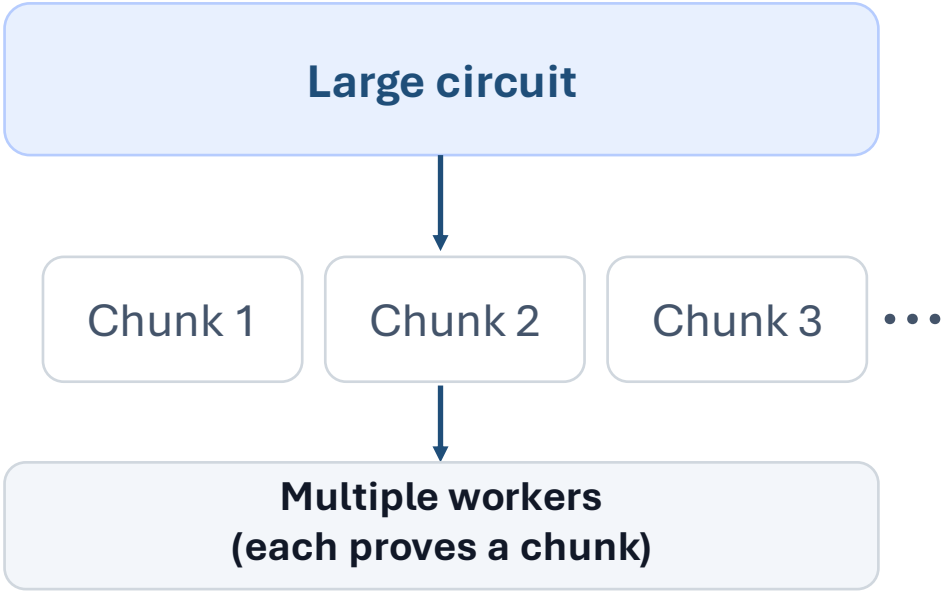
Wenhao Wang[1,4], Fangyan Shi[2], Dani Vilardell[3,4], Fan Zhang[1,4]

[1]**Yale University**, [2]Tsinghua University, [3]Cornell University, [4]IC3

Yale Cornell University

SNARKs call for distribution

- Large real-world SNARK applications produce massive circuits
  - Proving becomes a bottleneck
- Single-machine proving hits **time** & **memory** limits
  - High memory footprint
  - Long proving time
- Idea: split the proving workload across many workers (horizontal scaling)



Example single-machine pain point

~200 GB RAM
> 500 s proving

What makes distributed SNARKs hard?

Not just “distributed”, but distributed well

(1) Low overhead

- Linear scalability: Worker compute ∝ circuit chuck size
- Low coordination cost
- Low communication per worker (sublinear in circuit chuck size)

(2) Accountability

- Detect faults when proof fails
- Locate faulty workers
- Enables incentives (pay / slash) in prover markets
- A property that most prior works omit

(3) Universal setup

- Avoid per-circuit trusted setup ceremonies
- Support multiple applications & varying worker counts at once
- Critical for real deployments and outsourcing

Prior work: no one got all 3 properties

- Existing systems sacrifice at least one of scalability, accountability, or universal setup

Scheme	Overhead	Accountable?	Setup
DIZK	High Comm.	🔴 NO	Circuit-specific
deVirgo	High Comm.	🔴 NO	Transparent
Pianist	Low	🔴 NO*	Universal
HyperPianist	Low	🔴 NO	Universal
Hekaton	Low	🟢 YES	Circuit-specific
Cirrus	**Low**	🟢 **YES**	**Universal**

* Pianist offers limited accountability for data-parallel circuits only.

Cirrus: the high-level workflow

1) Partition circuit

Split gates into chunks

2) Local proving

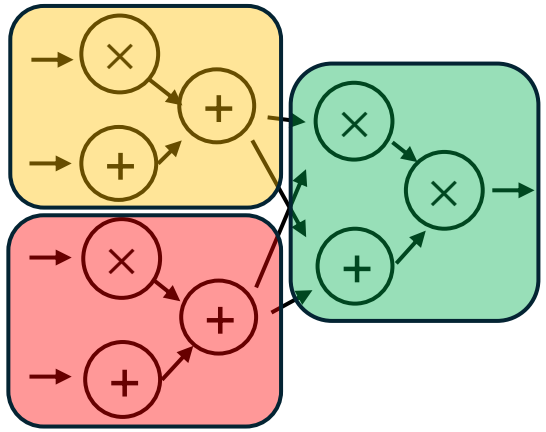
Each worker proves its chunk (linear work in chunk size)

3) Aggregate

Coordinator combines messages & relays challenges

4) Verify succinctly

Verifier checks a short proof (Logarithmic in circuit size)



Worker 1
Worker 2
Worker 3

↔

Coordinator

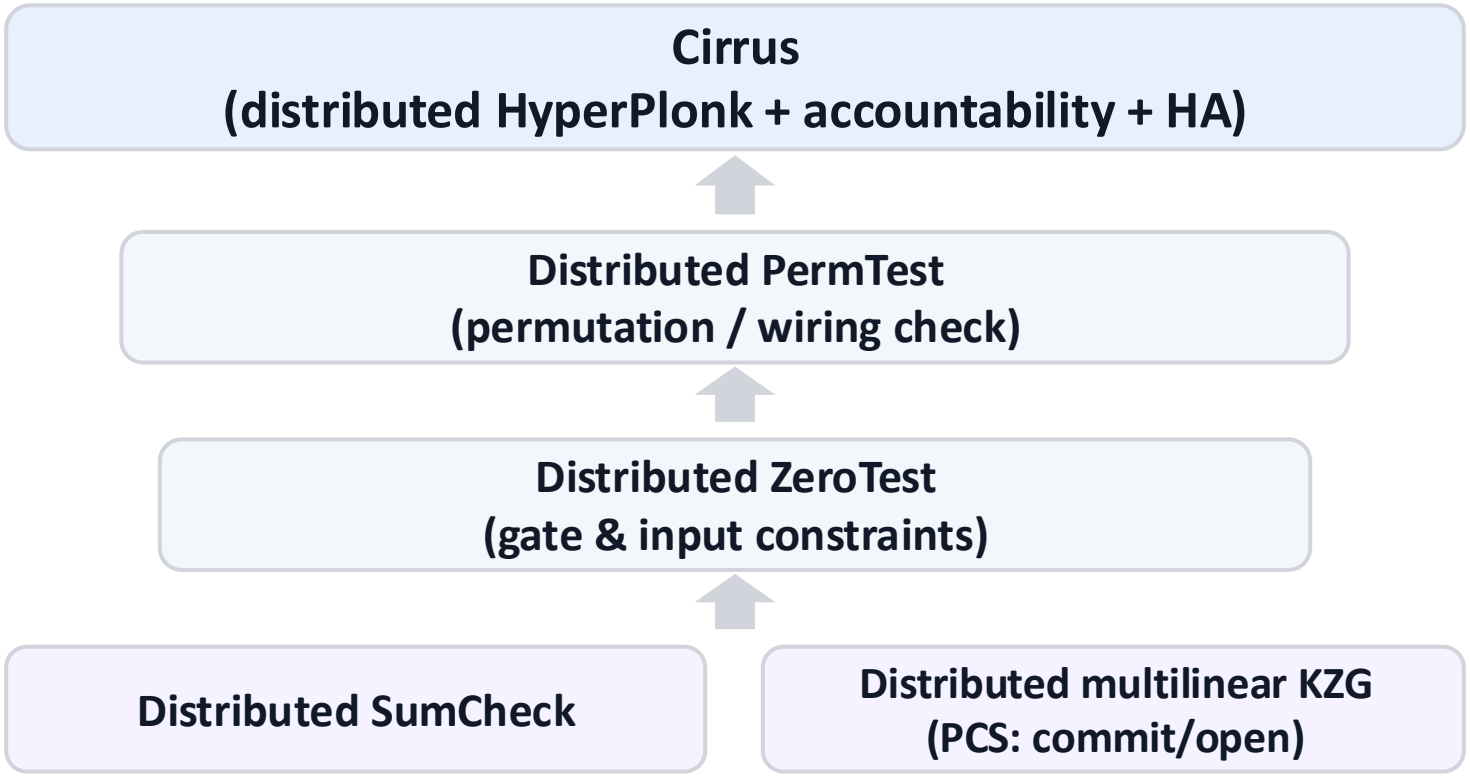
Aggregate results

→

Verifier

Goal: linear work per worker + small communication, while retaining universal setup

Components of Cirrus



Built on HyperPlonk's universal setup; designed for scalable proving across many workers

Distributed Multilinear KZG

- **Multilinear KZG (PCS)**: commit to a multilinear polynomial and open it at one point succinctly $f(\boldsymbol{\alpha}, \boldsymbol{\beta}) = z$

Commit (distributed)

- Worker $\mathcal{P}_{\boldsymbol{b}}$ commits to its local chunk polynomial $f^{(\boldsymbol{b})}(\boldsymbol{x})$
- Coordinator multiplies commitments to get a single commitment $\mathsf{com}_{\boldsymbol{b}}$

$$\mathsf{com} = \prod \mathsf{com}_{\boldsymbol{b}}$$

- Reuses HyperPlonk universal CRS

Open at (α, β) (distributed)

- Worker computes local $z_{\boldsymbol{b}} = f_{\boldsymbol{b}}(\boldsymbol{\alpha})$ and opening proof $\boldsymbol{\pi}_{\boldsymbol{b}}$
- Coordinator aggregates $(z_{\boldsymbol{b}}, \boldsymbol{\pi}_{\boldsymbol{b}})$ into $(z, \boldsymbol{\pi})$

$$z = \sum z_{\boldsymbol{b}} \qquad \boldsymbol{\pi} = \left(\underbrace{\prod \boldsymbol{\pi}_{\boldsymbol{b}}}_{\boldsymbol{\alpha} \text{ coordinates}} , \underbrace{\boldsymbol{\pi}'}_{\boldsymbol{\beta} \text{ coordinates}} \right)$$

- Verifier checks a constant-size pairing equation

Each worker sends only O(log T) group elements per opening; coordinator just aggregates.

Distributed SumCheck

- **SumCheck:** Prove a sum over the Boolean hypercube without sending all evaluations

$$\sum_{z \in \{0,1\}^n} f(z) = v$$

Standard SumCheck (one variable per round)

$$r_i(X) = \sum_{w \in \{0,1\}^{n-i}} f(\alpha_1, \ldots, \alpha_{i-1}, X, w)$$

- Verifier checks: $v_{i-1} = r_i(0) + r_i(1)$
- Verifier samples random challenge α_i
- Next target becomes $v_i = r_i(\alpha_i)$
- At the end, open f at one random point

Cirrus: distribute it across workers

$$r_i(X) = \sum_{b \in \{0,1\}^{\xi}} r_i^{(b)}(X)$$

- Worker b computes its local $r_i^b(X)$ from its chunk
- Coordinator adds them up to form global $r_i(X)$
- This repeats for $\log T$ rounds (T = chunk size)
- At the end, workers open polynomials at the same random point

Reducing polynomial degree in SumCheck

- Problem with distributed polynomial opening: only supports degree-1 polynomials

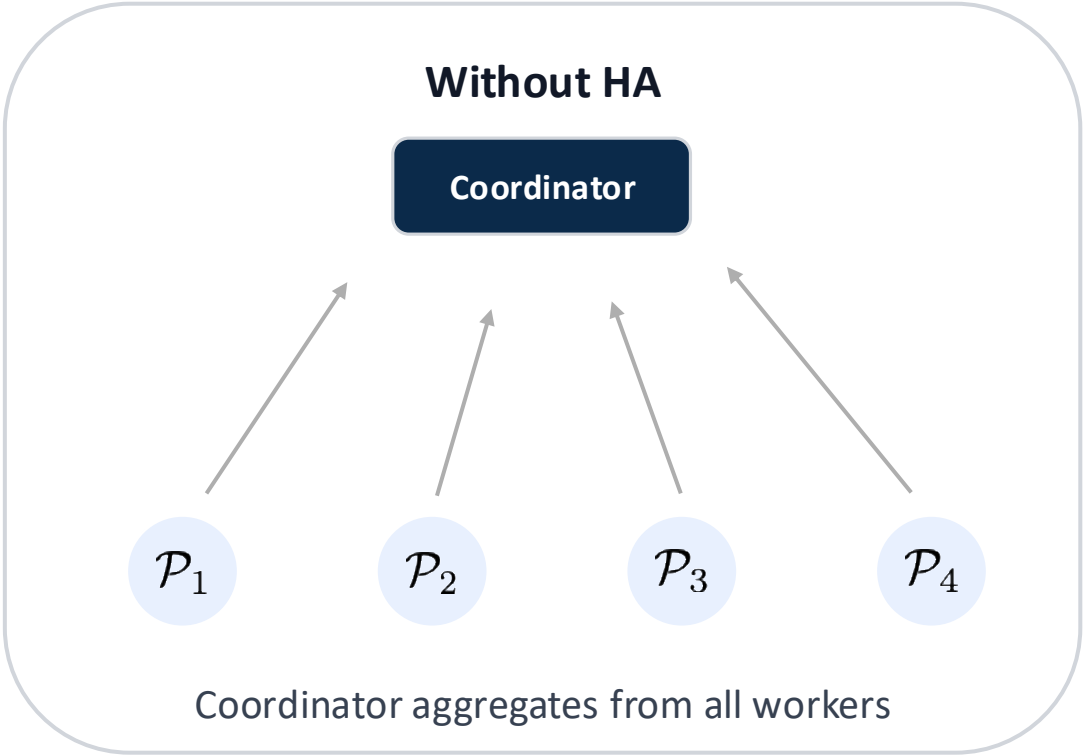
Our compatibility trick: reduce the higher-degree polynomial into a small number of multilinear polynomials, so multilinear KZG openings work cleanly

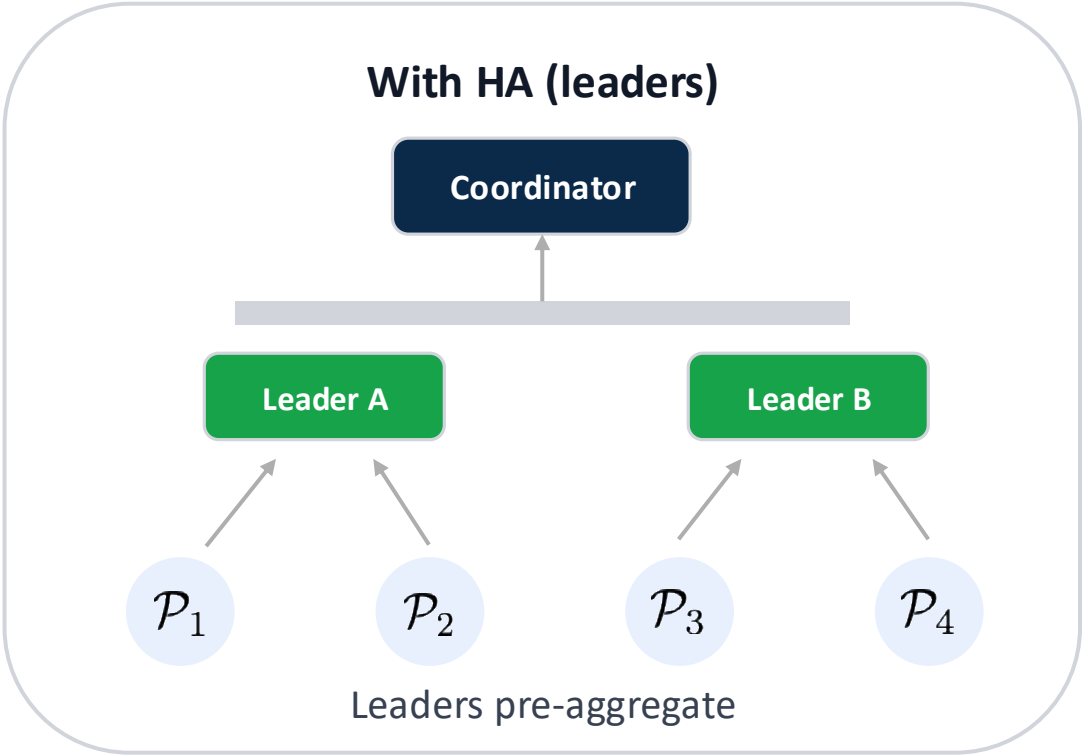
Given a high-degree polynomial h, we can split it into multiple linear polynomials, and then commit to each linear polynomials instead

$$\forall \boldsymbol{y} \in B_\xi, \qquad \tilde{f}(\boldsymbol{y}) = h(\tilde{g}_1(\boldsymbol{y}), \ldots, \tilde{g}_c(\boldsymbol{y}))$$
$$\equiv \sum_{\boldsymbol{b} \in B_\xi} h(g_1^{(\boldsymbol{b})}(\boldsymbol{y}), \ldots, g_c^{(\boldsymbol{b})}(\boldsymbol{y})) \cdot \chi_{\boldsymbol{b}}(\boldsymbol{y}) = f(\boldsymbol{\alpha}, \boldsymbol{y})$$

The equality showing that polynomial splitting does not break equality when evaluated one the Boolean hypercube

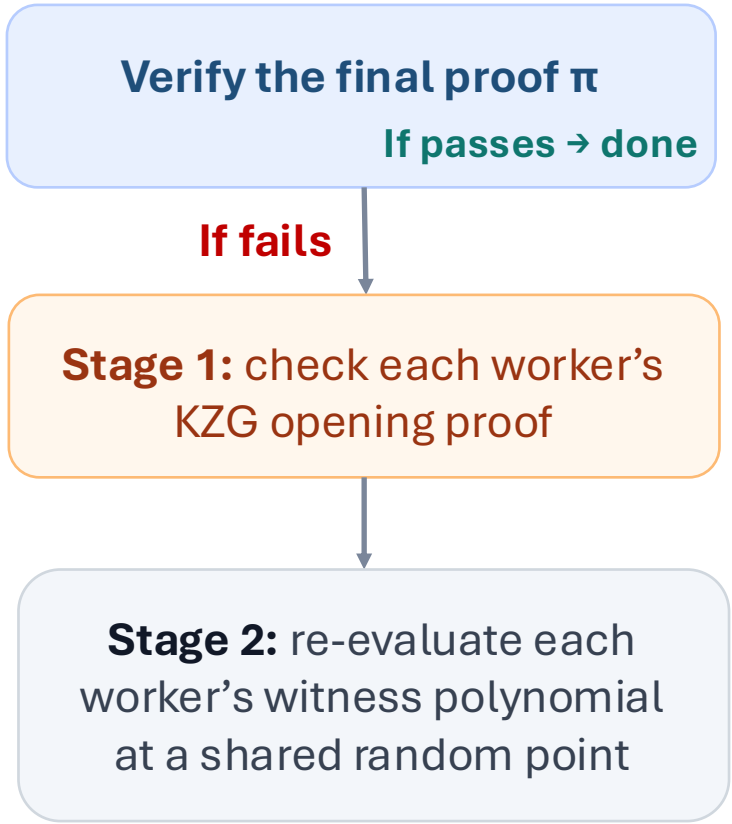
Hierarchical Aggregation (HA)





Saves coordinator fan-in and aggregation work: from $O(M \cdot \log T) \rightarrow O(M)$
(Coordinator runtime independent of sub-circuit size)

Accountability with low overhead



Optimistic idea:

- If the final proof verifies → All good
- Only localize faults when verification fails
- Goal: Make the accountability protocol fast
  - Naïve approach: redo all workers' computation

Our solution: 2-Stage localization

- Observation: Verifying a few intermediate results suffices; no need to redo all work.
- Result: With just 2 checks, the coordinator detects faulty workers

End-to-end scalability

HyperPlonk baseline: proves 4M gates in more than 100s (runs out of memory beyond that)

33M-gate circuits proved in ≈39 seconds

using 32 machines × 8 cores (AWS t3.2xlarge)

Per-worker communication

≈ 40 KB

Proof size

≈ 17 KB

Accountability

< 4 s

Why distribution helps

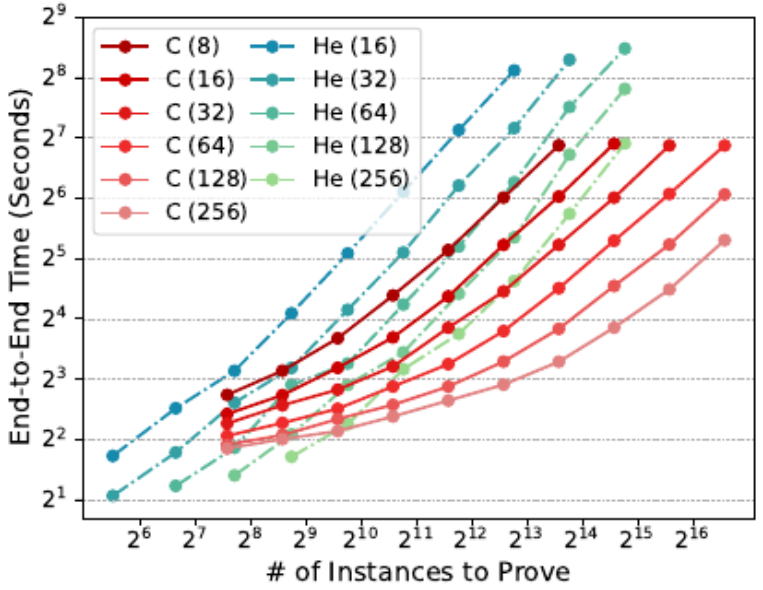
- Cirrus keeps per-worker memory bounded by the subcircuit size T
- Low coordination overhead → near-linear scaling with more workers for large circuits

Comparison with Hekaton

Cirrus
PLONK • Universal setup

Hekaton
R1CS • Circuit-specific setup

- **Apples-to-oranges:** different arithmetizations
- **But on PLONK-friendly tasks** Cirrus is faster
- **Pedersen hash circuits** > 7× faster proof generation



(a) Comparison between Cirrus and Hekaton to prove Pedersen

Fig. 3(a) from Cirrus paper: Pedersen hash benchmark
(C = Cirrus, He = Hekaton; cores in parentheses)

Takeaways

- Why this matters
  - Unlocks larger ZK workloads: zkML, zkVMs
  - Enables outsourced proving

- What Cirrus achieves
  - ✓ Linear-time workers and lightweight coordinator
  - ✓ Fast accountability (localize faulty workers)
  - ✓ Universal trusted setup

Personal Website: wenhao-wang0.github.io

Link to full paper

Link to personal website

Thank you!