Hyperplonk: Plonk with linear-time prover and high degree gates

Binyi Chen
Espresso Systems

Benedikt Bünz
Espresso, Stanford, NYU

Dan Boneh
Stanford

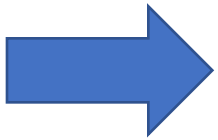
Zhenfei Zhang
Espresso Systems

Preprocessing (zk)-SNARKs

SNARK = A *succinct* proof showing that $\exists \boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

small size fast to verify

Why so popular recently? Outsourcing computation

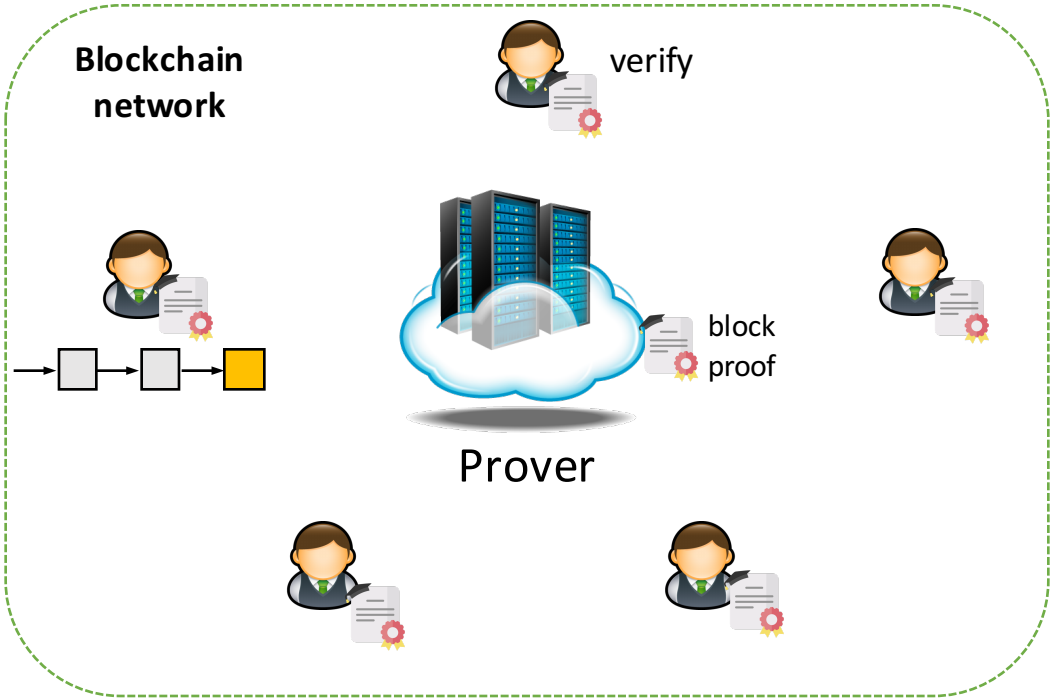


Computation happens locally

Computation goes remote/global

Application: Blockchain

SNARK = A *succinct* proof showing that $\exists \boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$



More applications: zkRollup, zkEVM, zkBridge, DSNs, zkML

Preprocessing (zk)-SNARKs

Call for:

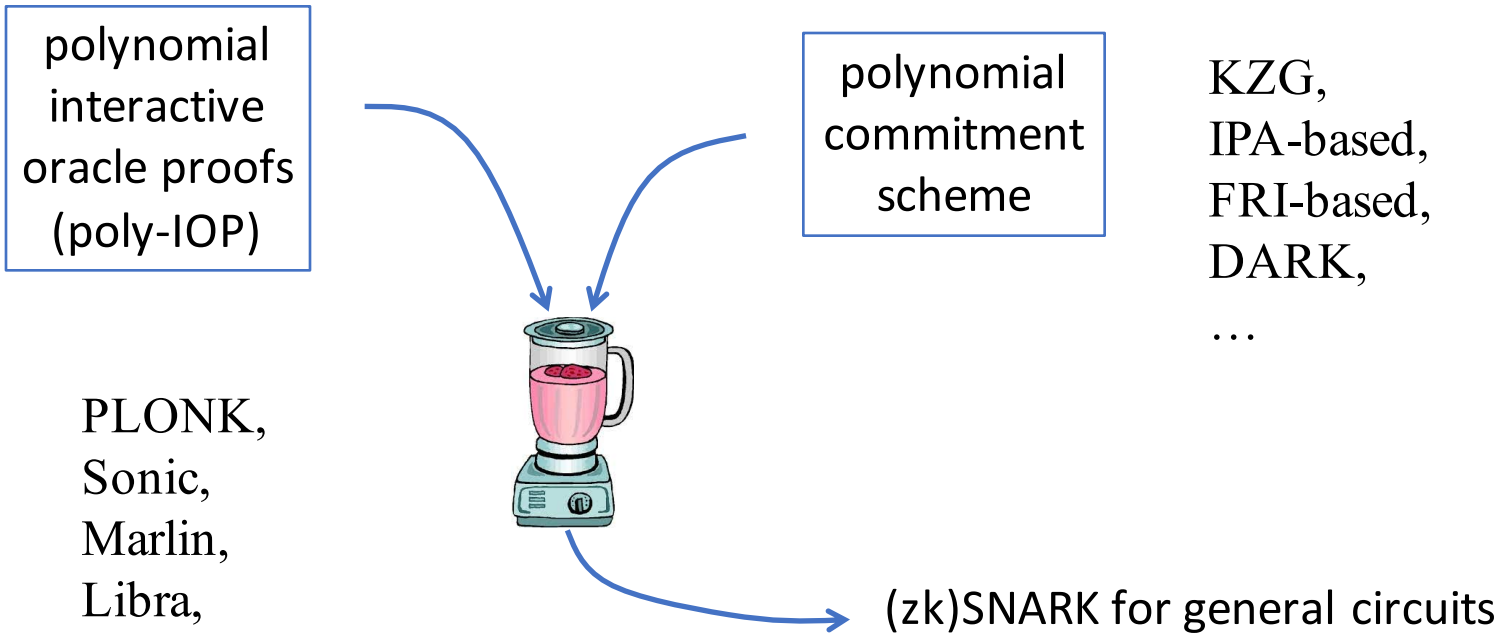
- *Fast* prover for *large* statements (e.g., as fast as computation)
- Small proof size and efficient verification
- Powerful circuit constraint systems (e.g., high-deg/lookup gates)
- Hardware-friendliness

Our Contributions

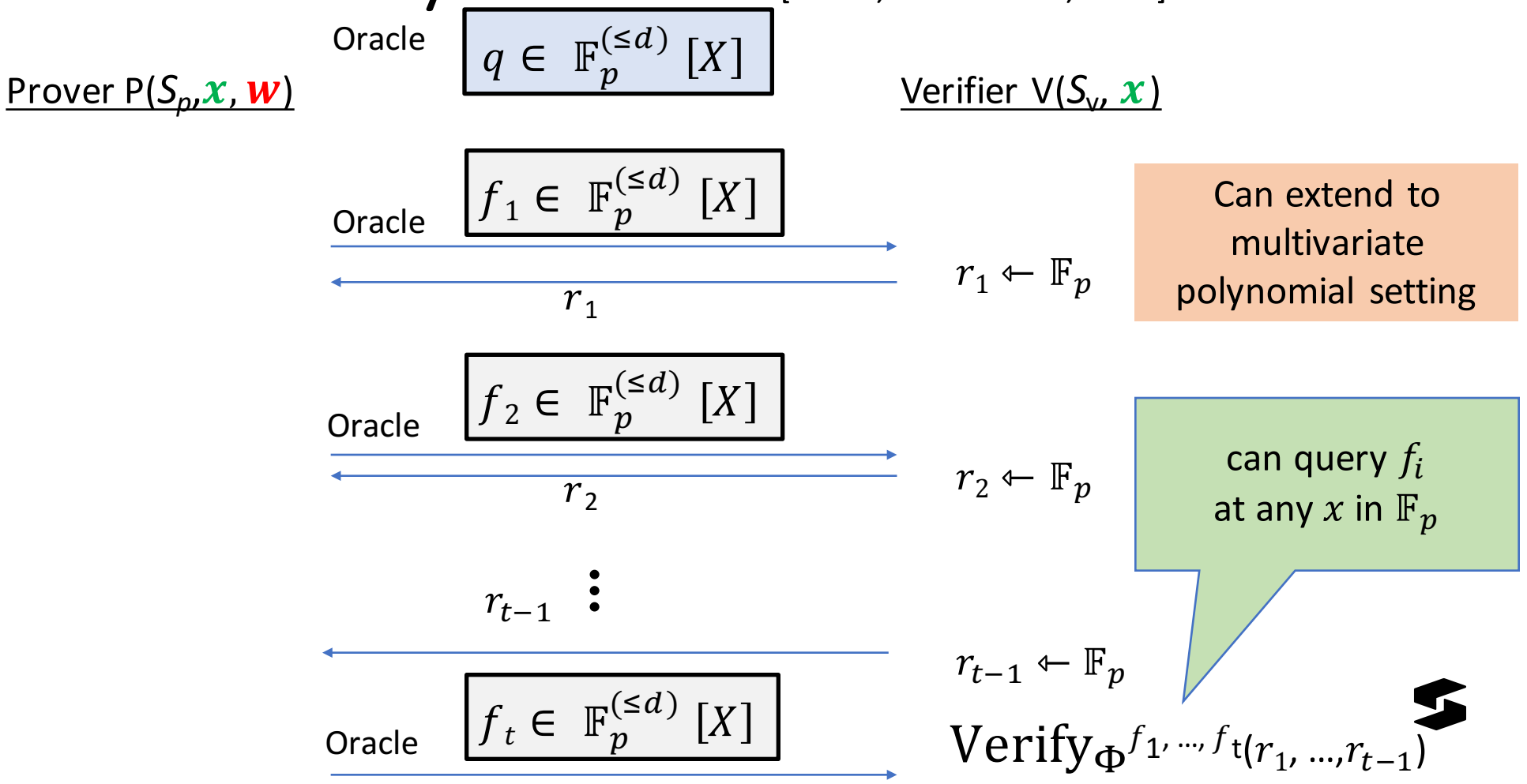
- **HyperPlonk+ PolyIOP**
  - Linear-time prover, no use of FFT-friendly field
    - The *first* linear-time SNARK that has expressive gate support and small proof size
    - Hardware-optimization friendly
  - Native high-degree custom gate support
    - Better support than Plonk [GWC19]
  - Lookup gate support
    - The *first linear-time* PolyIOP for the lookup relation!
- **Towards strict linear-time SNARKs: Orion+ PCS**
  - Improve opening size of the state-of-the-arts (Orion & Brakedown)
    - 5MB proof -> <10KB proof
- **Active industry deployment/development**
  - E.g., ETH Foundation/Scroll actively develop it for future zkEVM solutions

General paradigm for modern SNARKs

Many SNARKs are built in two steps:



Polynomial IOP [BCS16,CHMMVW19,BFS19]



Polynomial Commitment [KZG10]

Committing phase

pp ← **Setup**(d)

pp, $f \in \mathbb{F}_p^{(\leq d)}[X]$, r

Prover

commit(*pp*, f, r) ↠ *"short"* com_f

pp

Verifier

Binding: cannot output two valid openings $(f_1, r_1), (f_2, r_2)$ for com_f.

Optional: commitment com_f is **hiding**

Polynomial Commitment

Evaluation phase

Goal: "convince" V that $f(z) = y$ and $\boldsymbol{com_f}$ =*commit*(*pp*, f, r)

$\boldsymbol{x} = (pp, \boldsymbol{com_f}, z, y)$, $\boldsymbol{w} = (\boldsymbol{f}, \boldsymbol{r})$

Prover

proof π

$\boldsymbol{x} = (pp, \boldsymbol{com_f}, z, y)$

Verifier

Accept/Reject

Eval is **knowledge sound**:

- V accepts $\Rightarrow$ P "knows" $\boldsymbol{f} \in \mathbb{F}_p^{(\leq d)}[X], \boldsymbol{r}$ s.t. $f(z) = y$ and $\boldsymbol{com_f}$ =*commit*(*pp*, f, r)

Optional: **Eval** is **zero knowledge**: π "reveals nothing" about f.

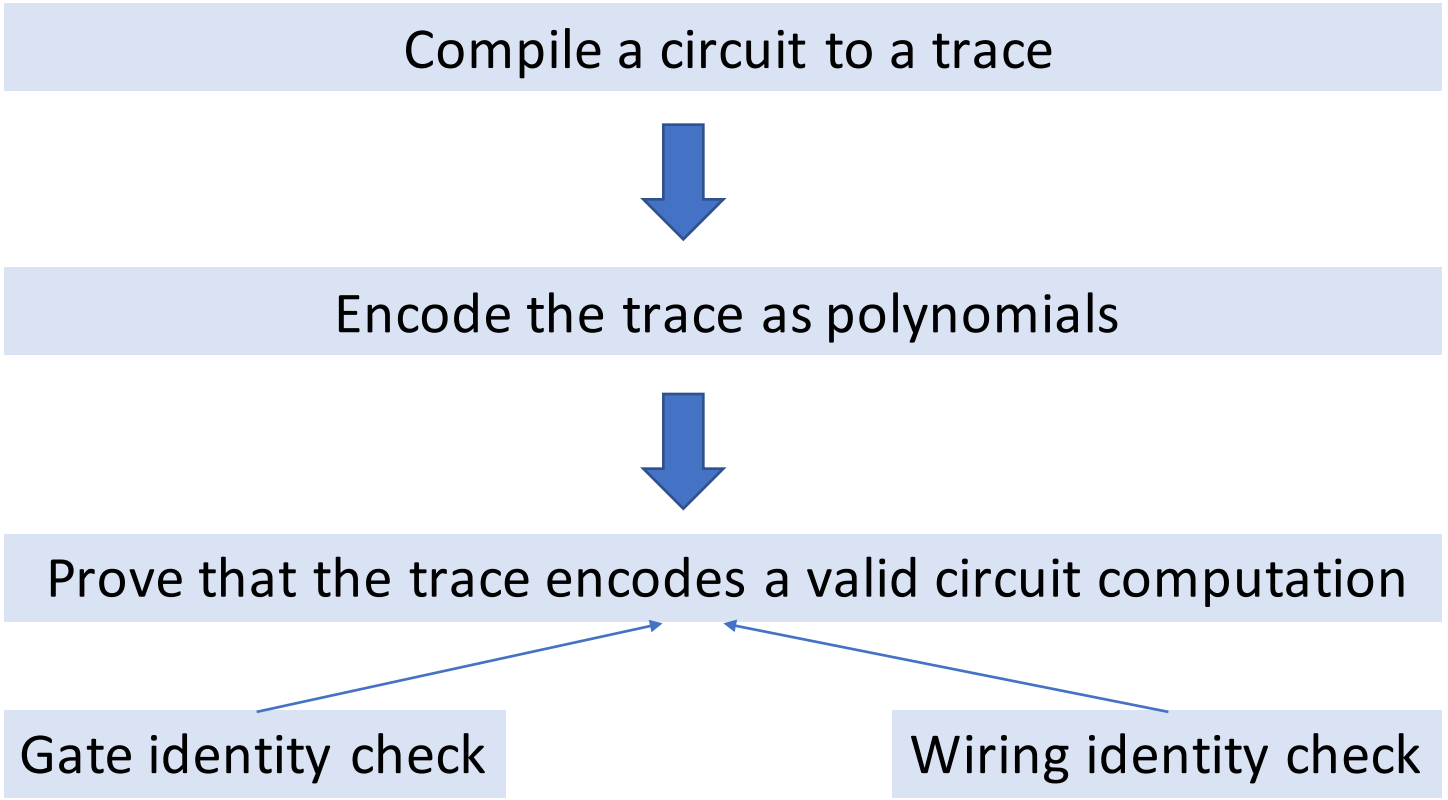
Outline

- A generic framework for proving circuit relations
- High-degree gates support
- Hyperplonk+: support lookup on the Boolean hypercube
- Evaluations

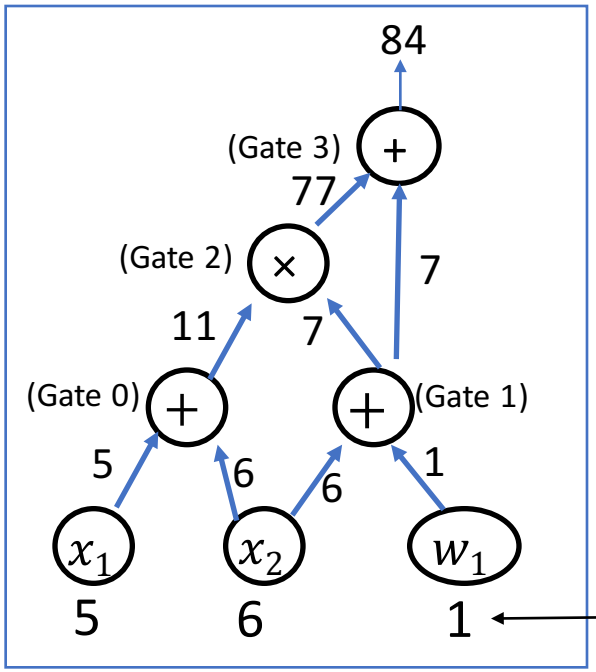
Outline

- A generic framework for proving circuit relations
- High-degree gates support
- Hyperplonk+: support lookup on the Boolean hypercube
- Evaluations

Workflow [GWC19]



PLONK's Circuit Encoding



example input

The computation trace:

inputs:	5,	6,	1
Claimed output: 84			
Gate 0:	5 ,	6 ,	11
Gate 1:	6 ,	1 ,	7
Gate 2:	11,	7,	77
Gate 3:	77 ,	7 ,	84

left inputs

right inputs

outputs

Trace to Polynomial

The computation trace:

Gate 0	Gate 1	Gate 2	Gate 3
5	6	11	77
6	1	7	7
11	7	77	84

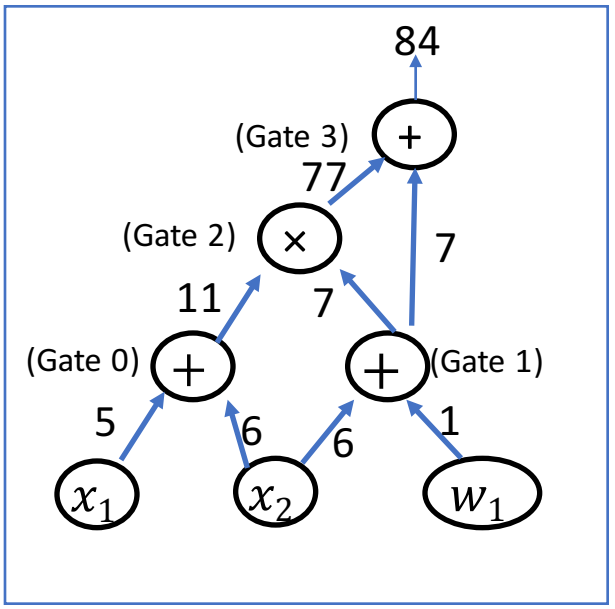
The polynomials:

L($\mathbf{H}$) =	**5**	**6**	**11**	**77**
R($\mathbf{H}$) =	6	1	7	7
O($\mathbf{H}$) =	11	7	77	84

Options of $\mathbf{H}$:

- $\mathbf{H} = \{1, \omega, \omega^2, \omega^3\}$
  - L, R, O are univariate polys
  - Interpolating L, R, O needs FFT
- $\mathbf{H} = \{00,01,10,11\}$
  - L, R, O are multilinear polys
  - Polys are in eval form, no interpolation
  - Free embedding!

Gate Check [GWC19]



The polynomials:

L(**H**) =	**5**	**6**	**11**	**77**
R(**H**) =	6	1	7	7
O(**H**) =	11	7	77	84
S(**H**) =	1	1	0	1

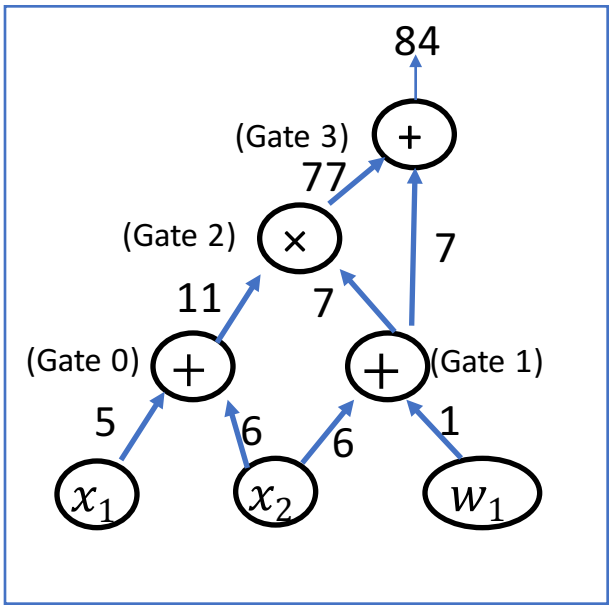
Idea: encode gate types using a *selector* polynomial S(X)

$\forall\ \ell = 0, \ldots, |C| - 1$:

S(H_ℓ) = 1 if gate #l is an addition gate

S(H_ℓ) = 0 if gate #l is a multiplication gate

Gate Check [GWC19]



The polynomials:

L(**H**) =	**5**	**6**	**11**	**77**
R(**H**) =	6	1	7	7
O(**H**) =	11	7	77	84
S(**H**) =	1	1	0	1

The zero-check: $\forall\ y \in H$:

$$S(y) \cdot [L(y) + R(y)] + \big(1 - S(y)\big) \cdot [L(y) \cdot R(y)] = O(y)$$

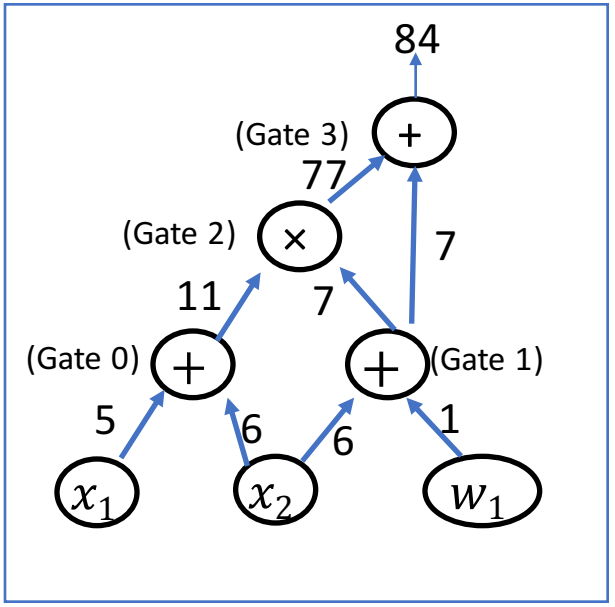
Addition gate

$1 \cdot [L(y) + R(y)] + (1 - 1) \cdot \quad = O(y)$

Multiplication gate

$0 \cdot \quad + (1 - 0) \cdot [L(y) \cdot R(y)] = O(y)$

Gate Check [GWC19]



The polynomials:

L(**H**) =	**5**	**6**	**11**	**77**
R(**H**) =	6	1	7	7
O(**H**) =	11	7	77	84
S(**H**) =	1	1	0	1

ZeroCheck

$f(y) = 0 \forall y \in H$

SumCheck

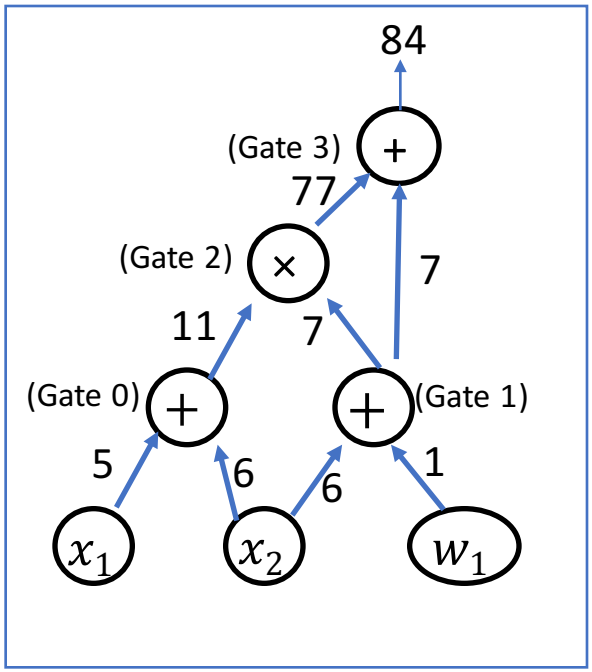
$\sum_{y \in H} f(y) \cdot eq_r(y) = 0$

Intuition: rand linear combine

Univariate Sumcheck: requires FFTs

Multivariate Sumcheck: linear prover time!

Wiring Identity Check



The computation trace:

	L	R	O
inputs:	5,	6,	1
Gate 0:	5 ,	6 ,	11
Gate 1:	6 ,	1 ,	7
Gate 2:	11,	7,	77
Gate 3:	77,	7 ,	84

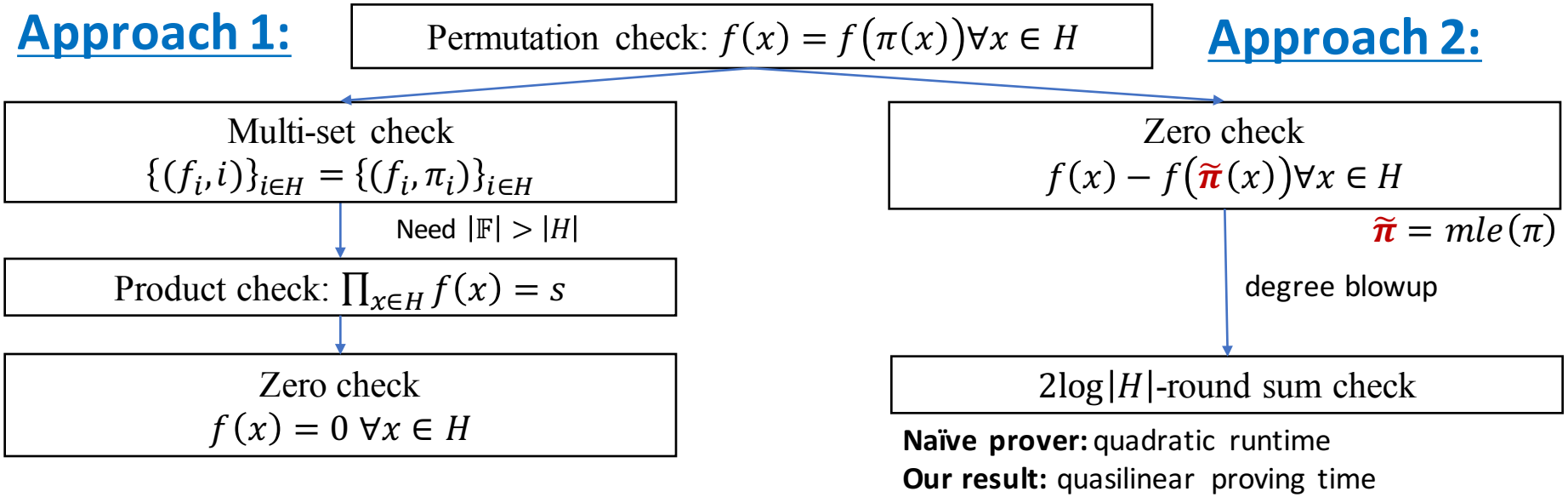
All slots with the same color should have identical values!

$$(L, R, O) = \pi(L, R, O)$$

for some **fixed permutation** π

Permutation Check

Prove $(L, R, O) = \pi(L, R, O)$ for some **fixed permutation** π



Naïve prover: quadratic runtime
Our result: quasilinear proving time

Advantage: Linear prover time
Tradeoff: Not super fit for "small" fields

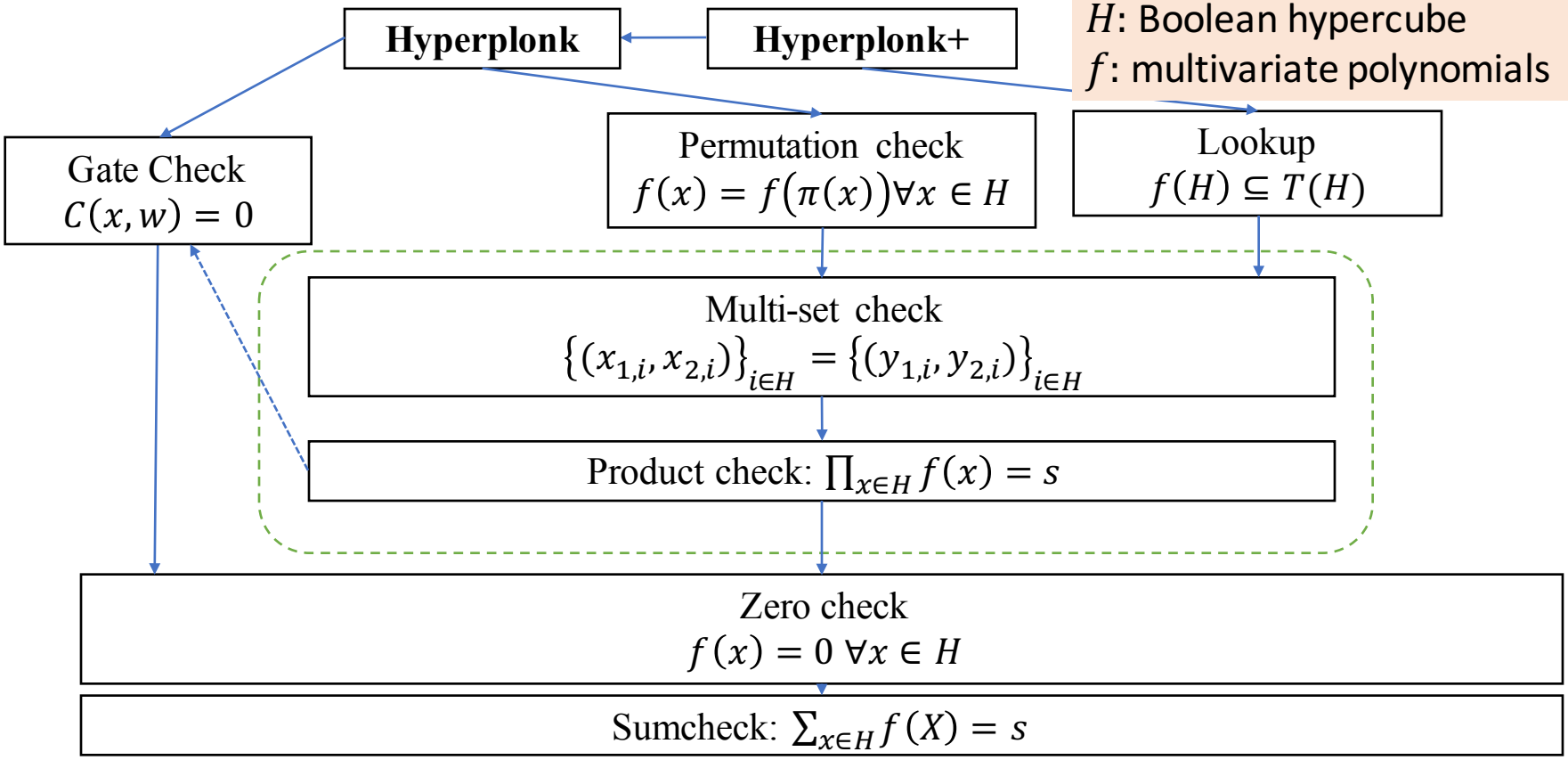
Advantage: soundness err = $(\log|H|)^2/|\mathbb{F}|$
Tradeoff: quasilinear prover

Hyperplonk: Components

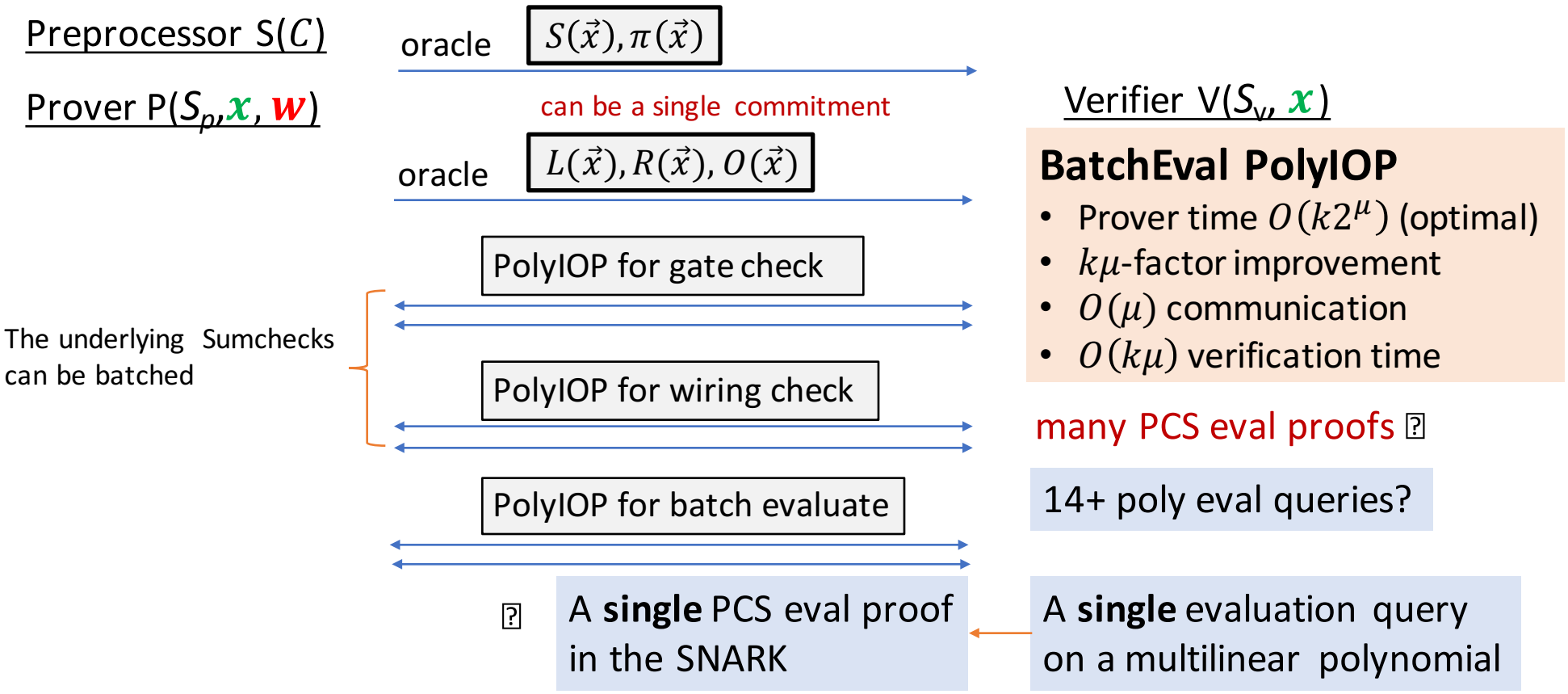
Recipe for linear prover & avoid FFTs

H: Boolean hypercube

f: multivariate polynomials



The last mile to success: batching



Outline

- Adapt Plonk to the Boolean hypercube
- High-degree gates support
- Hyperplonk+: support lookup on the Boolean hypercube
- Evaluations

Hyperplonk: High-degree Gates

$$\mathrm{S}(y) \cdot [L(y) + R(y)] + \big(1 - S(y)\big) \cdot [L(y) \cdot R(y)] = O(y)\ \forall y \in H_{gates}$$

The gate formula can be more general
e.g. more selectors and more different custom gates

high degree, e.g., $L(y)^5 + R(y)$

$$S(y) \cdot [L(y) + R(y)] + \big(1 - S(y)\big) \cdot [\boldsymbol{G(L(y), R(y))}] = O(y)\ \forall y \in H_{gates}$$

Benefit: reduce the size of the circuit/witness -> faster prover

E.g.: ECC addition, Rescue hash, etc…

Hyperplonk: High-degree Gates

Plonk: more expensive quotient check

- Quotient polynomial $\boldsymbol{q(X)}$ has degree $O(dn)$
- Commit $\boldsymbol{q(X)}$ -> increase group operations
- Higher degree poly-mul -> larger-sized FFTs
- In practice, deg ≤ 8

Quotient Check

$f(X) = Zero_H(X) \cdot \boldsymbol{q(X)}$

V.S.

Hyperplonk: efficient multivariate sumcheck

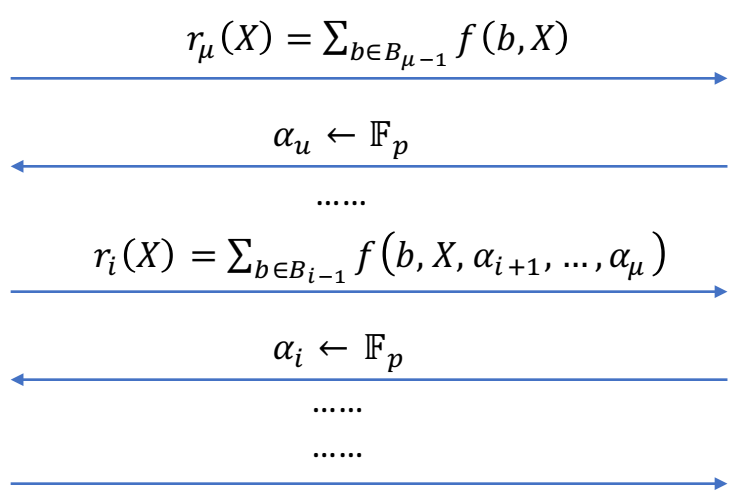
- Only $\tilde{O}(dn)$ field-operations
- Allows much higher degree

Sumcheck

$\sum_{y \in B_\mu} f(y) \cdot eq\ (y, r) = 0$

Efficient Sumcheck for High-degree Polynomials

Goal: "convince" V that $\sum_{x \in B_\mu} f(X) = s$

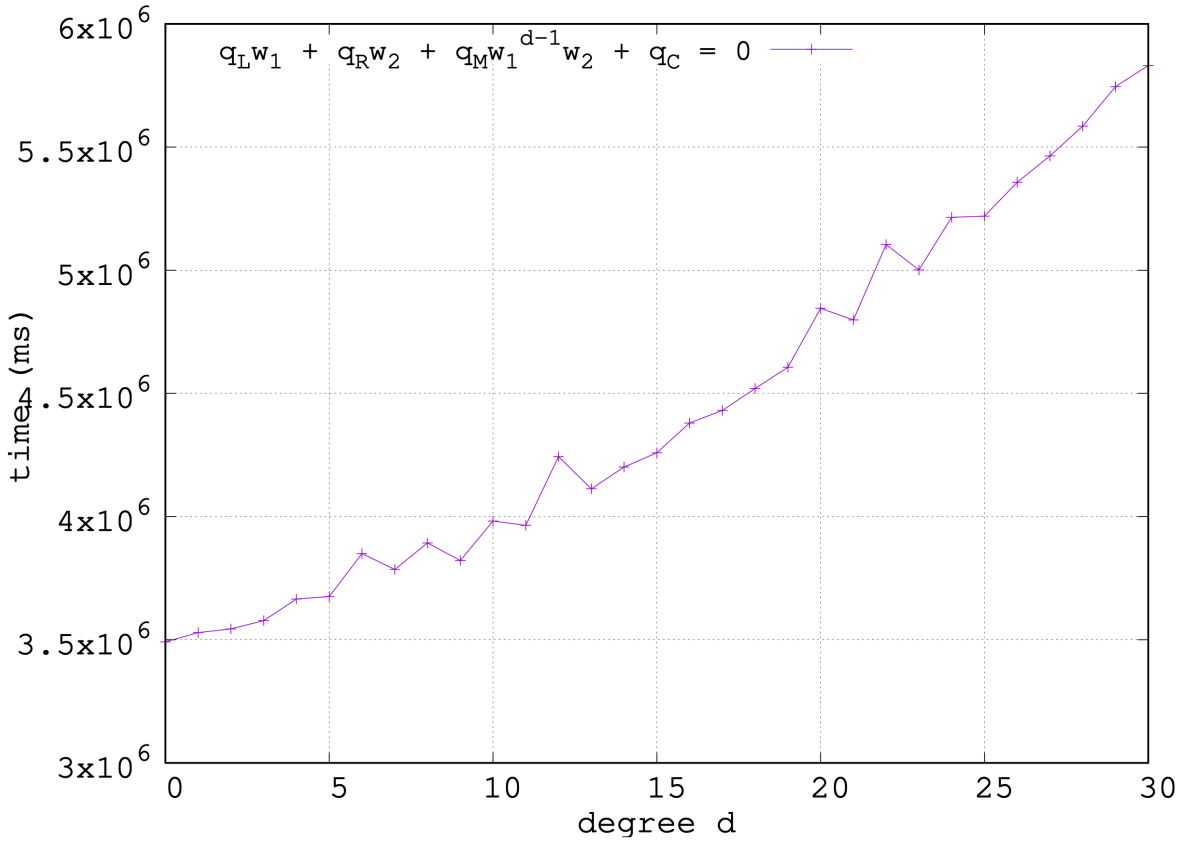


Optimizations to the classic Sumcheck [LFKN92]

- Sending r_i as univariate oracles (d field elems -> 1 group elem)
- Decrease the number of queries per round: 3 queries -> 1 query
- More efficient algorithm for computing Sumcheck for high-degree polynomials
  - $O(2^u d^2) \rightarrow O(2^\mu d \log^2 d)$ (and $O(2^\mu d \log d)$ for certain custom gates)
  - In practice, replace FFTs with Karatsuba

No proof size/verifier dependence on d anymore

Evaluations: High-degree Gates



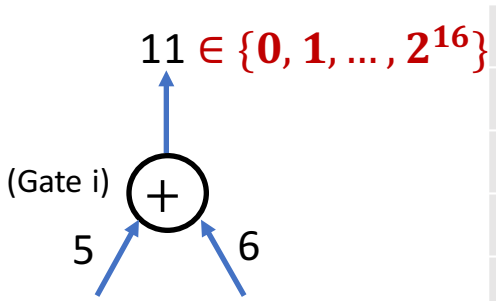
- Degree 32 is only 30% more expensive than degree 2

Outline

- Adapt Plonk to the Boolean hypercube
- High-degree gates support
- Hyperplonk+: support lookup on the Boolean hypercube
- Evaluations

Lookup Gates

Range check: **Expensive to express with arithmetic gates**

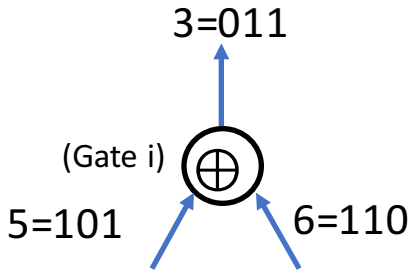


0
1
2
…
2^{16}-1

$x \in [0..2^{16})$

1 lookup gate suffices!

Bit operation:



(0,0,0)
…
(5,6,3)
…
(15,8,7)

precomputed

$(x, y, z) \in \{0,1\}^8 \times \{0,1\}^8 \times \{0,1\}^8$

Why so useful:

- Efficient random memory accessing
- Reuse precomputed correlated tuples

The Idea of Plookup [GW20]

Lookup relation: $\vec{f} \subseteq \vec{T}$ for some preprocessed table T and some witness vector f

Lookup relation

	f		T
1	1		1
ω	1	$\subseteq$	2
ω^2	3		3
ω^3	2		4

H

f	f
1	1
1	1
3	3
2	2

T	$T_\circlearrowleft$
1	2
2	3
3	4
4	1

$=$

h	$h_\circlearrowleft$
1	1
1	1
1	2
2	2
2	3
3	3
3	4
4	1

Multiset Equality Check Poly-IOP

- It's easy to shift on H for univariate polynomials: $f(H) \to f(\omega H)$
- How to do it in the Boolean hypercube for multilinear polynomials?

Shift in the Boolean Hypercube

Step 1: Shift on the Boolean hypercube from a multiplication in $GF(2^\mu)$

Cool connection w/ short PCP [Ben-Sasson, Sudan 05]

Let $p_\mu(X) \in \mathbb{F}_2^\mu[X]$ be a **primitive polynomial**

$\mathbb{F}_2[X]/(p_u) \cong GF(2^\mu)$ and X is a **generator**!

E.g. $p_\mu(X) = \mathrm{X}^4 + \mathrm{X} + 1$

$$b_3X^3 + b_2X^2 + b_1X + b_0$$

$$u = (b_3, b_2, b_1, b_0) \in \mathbb{F}_{2^4} \setminus \{0\}$$

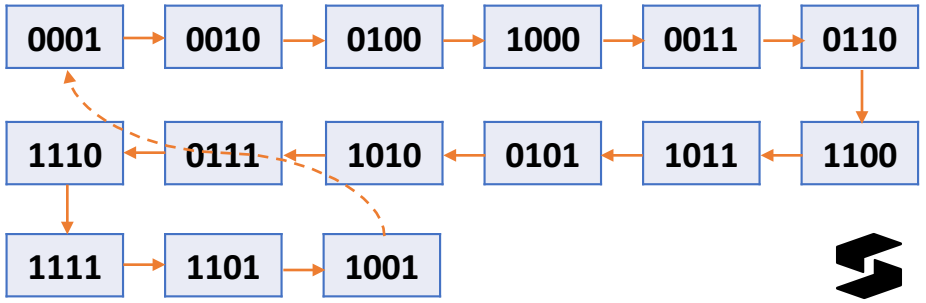
multiply X

$$b_3X^4 + b_2X^3 + b_1X^2 + b_0X$$
$$= b_2X^3 + b_1X^2 + (b_0 \oplus b_3)X + b_3$$

$$\mathrm{next}(u) = (b_2, b_1, b_0 \oplus b_3, b_3) \in \mathbb{F}_{2^4} \setminus \{0\}$$

Caveats

- $b_0 \oplus b_3 = b_0 + b_3 - 2b_0b_3$ is quadratic
- Individual deg of $f(\mathrm{next}(X)) = 2$
- Naïve MSET check on $f(\mathrm{next}(X))$
  - $\Theta(n^2)$ sumcheck prover



Shift in the Boolean Hypercube

Step 2: Simulate $f(\text{next}(X))$ on the Boolean hypercube using the multilinear $f(X)$!

$$u = (b_3, b_2, b_1, b_0) \in \mathbb{F}_{2^4} \setminus \{0\} \xrightarrow{\text{multiply X}} \text{next}(u) = (b_2, b_1, b_0 \oplus b_3, b_3) \in \mathbb{F}_{2^4} \setminus \{0\}$$

Idea: enumerate b_3 and linearize next()!

Define

$$f_{\circlearrowleft}(X_3, \dots, X_0) \coloneqq \boldsymbol{X_3} \cdot f(X_2, X_1, 1 - X_0, 1) + \boldsymbol{(1 - X_3)} \cdot f(X_2, X_1, X_0, 0)$$

Property 1: $f_{\circlearrowleft}(X) = f(\text{next}(X))$ on the Boolean hypercube!

Case $b_3 = 1$:

$$f_{\circlearrowleft}(b_3, \dots, b_0) \coloneqq \mathbf{1} \cdot f(b_2, b_1, 1 - b_0, 1) + \mathbf{(1 - 1)} = f(b_2, b_1, b_0 \oplus b_3, b_3) = f\big(\text{next}(b_3, \dots, b_0)\big)$$

Case $b_3 = 0$:

$$f_{\circlearrowleft}(b_3, \dots, b_0) \coloneqq \mathbf{0} \cdot + \mathbf{(1 - 0)} \cdot f(b_2, b_1, b_0, 0) = f(b_2, b_1, b_0 \oplus b_3, b_3) = f\big(\text{next}(b_3, \dots, b_0)\big)$$

Shift in the Boolean Hypercube

Step 2: Simulate $f(\text{next}(X))$ on the Boolean hypercube using the multilinear $f(X)$!

$$u = (b_3, b_2, b_1, b_0) \in \mathbb{F}_{2^4} \setminus \{0\} \xrightarrow{\text{multiply X}} \text{next}(u) = (b_2, b_1, b_0 \oplus b_3, b_3) \in \mathbb{F}_{2^4} \setminus \{0\}$$

Idea: enumerate b_3 and linearize next()!

Define

$$f_{\circlearrowleft}(X_3, \ldots, X_0) := \boldsymbol{X_3} \cdot f(X_2, X_1, 1 - X_0, 1) + (\mathbf{1} - \boldsymbol{X_3}) \cdot f(X_2, X_1, X_0, 0)$$

Property 2: Can simulate eval of $f_{\circlearrowleft}$ from 2 evals of f

The lookup PolyIOP can run sumcheck given oracles T, h

No need to build/check $T(\text{next}(X)), h(\text{next}(X))$

lookup PIOP with linear prover time!

Outline

- Adapt Plonk to the Boolean hypercube
- High-degree gates support
- Hyperplonk+: support lookup on the Boolean hypercube
- Evaluations

Evaluations

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
HP (Vanilla Gate)	0.07	0.1	0.14	0.2	0.3	0.5	0.8	1.4	2.5	5.1	9.6
HP (Jellyfish Gate)	0.1	0.13	0.18	0.27	0.4	0.67	1.2	2	3.7	7.3	13.5
Jellyfish Plonk	0.07	0.1	0.15	0.25	0.46	0.78	1.4	2.7	5.5	10.8	22
Ark-Spartan	0.51	0.72	0.9	1.4	1.9	3.1	4.7	8.3	13.7	27	44

Table 6: 64-thread prover's performance (in seconds) for varying number of constraints under different schemes.

Application	$\mathcal{R}_{\text{R1}CS}$	Ark-Spartan	$\mathcal{R}_{\text{PLONK+}}$	Jellyfish	HyperPlonk
3-to-1 Rescue Hash	288 [1]	422 ms	144 [71]	40 ms	88 ms
PoK of Exponent	3315 [63]	902 ms	783 [63]	64 ms	105 ms
ZCash circuit	2^{17} [55]	8.3 s	2^{15} [42]	0.8 s	0.6 s
Zexe's recursive circuit	2^{22} [81]	6 min	2^{17} [81]	13.1 s	5.1 s
Rollup of 50 private tx	2^{25}	39 min[b]	2^{20} [71]	110 s	38.2 s
zkEVM circuit[a]	N/A	N/A	2^{27}	1 hour[b,c]	25 min[b,c]

Proof size (μ=20)(with KZG PCS)

- Plonk ~1kb
- HP 4.7kb
- Spartan 40kb

Implementation

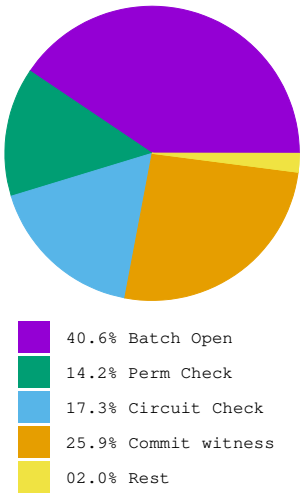
- Jellyfish: highly optimized
- HP: many possible improvements

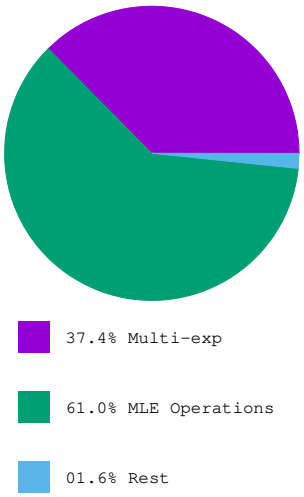
- Prover time
  - HP outperforms Plonk at 2^{14}; at 2^{20} HP is 60% faster then Plonk
  - Fewer constraints needed than Spartan for the same application

Cost Breakdown

Bottleneck

PCS commitments and batch openings





Potential optimizations

- many eval points and polys are identical
- Batching all zero-checks

Further improvements on hardware

- Multi-exps and sumchecks are highly parallelizable and hardware-friendly

Summary & Open problems

- **HyperPlonk+ PolyIOP**
  - Linear-time prover, no use of FFT-friendly field
  - Native high-degree custom gate support
  - Lookup gate support with linear-time prover
  - Concretely efficient
- **Open problems**
  - Degree $k > 1$ shifts?
    - Our technique: needs 2^k evals on f to simulate an eval on $f_{\circlearrowleft}$
  - Linear-time permutation argument for fields with small characteristics

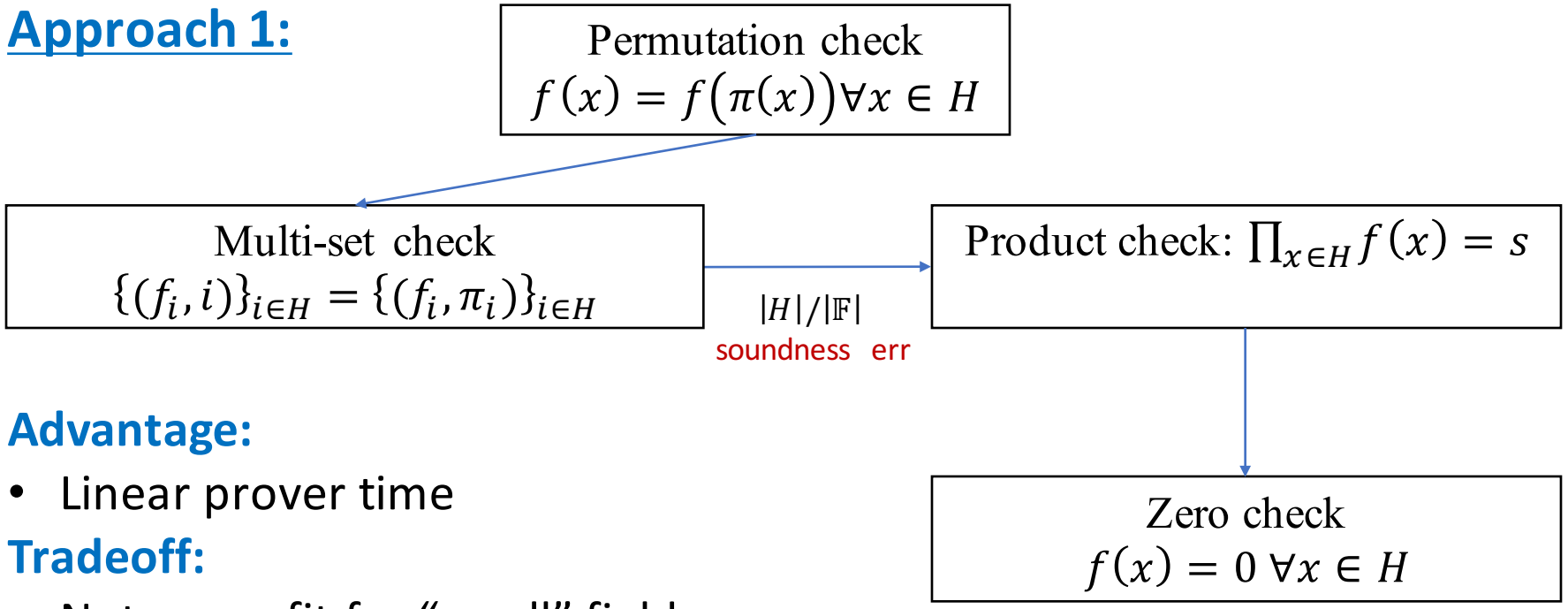
Thank you!

https://eprint.iacr.org/2022/1355.pdf

Permutation Check

Prove $(L, R, O) = \pi(L, R, O)$ for some **fixed permutation** π

Approach 1:



Advantage:

- Linear prover time

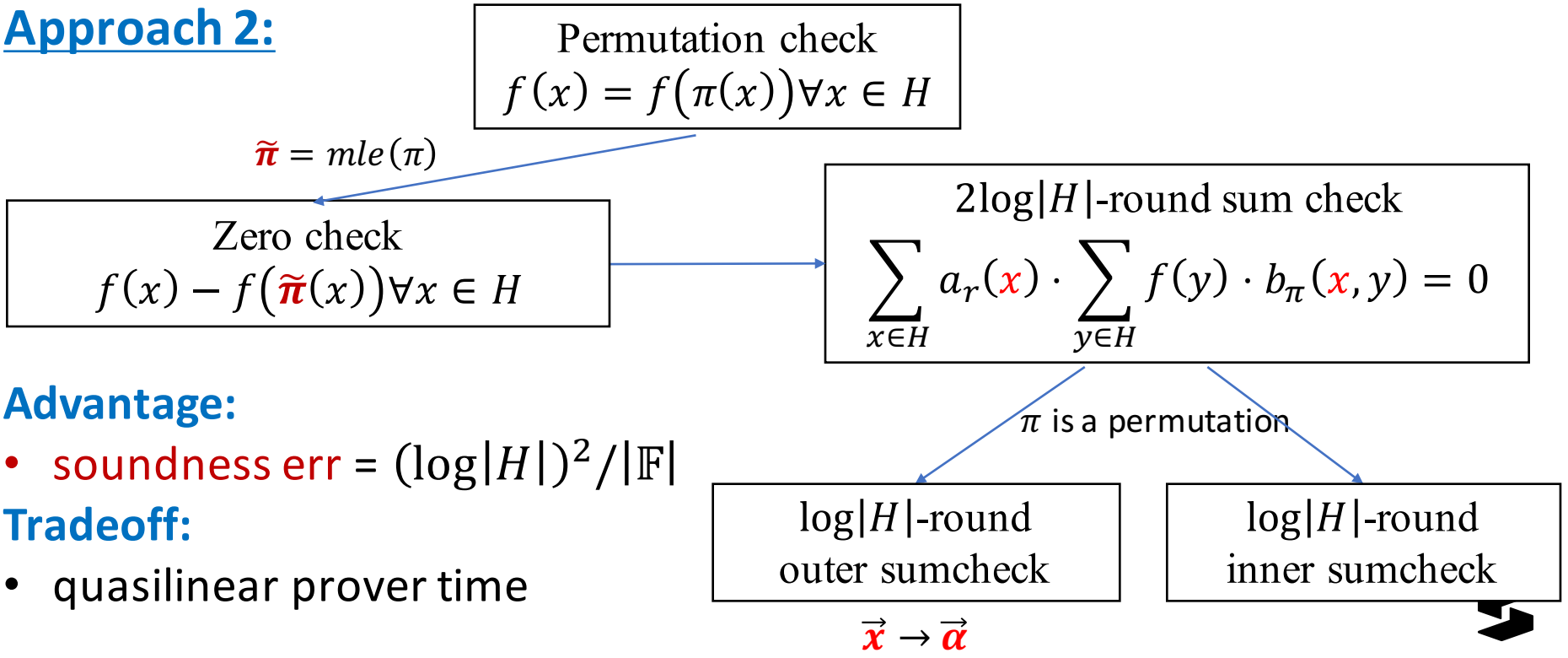
Tradeoff:

- Not super fit for "small" fields

Permutation Check

Prove $(L, R, O) = \pi(L, R, O)$ for some **fixed permutation** π

Approach 2:



Advantage:

- soundness err = $(\log|H|)^2/|\mathbb{F}|$

Tradeoff:

- quasilinear prover time

Efficient Batch Evaluations

Goal: "convince" V that $f_i(z_i) = y_i \forall i \in [k]$ ($f_i \in \mathbb{F}_p[X_1, \dots, X_\mu]$)

Prover P(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$)

PolyIOP for batch evaluate

Verifier V(S_v, $\boldsymbol{x}$)

A **single** evaluation query on a multilinear polynomial

Requirements

- Prover time ≈ the time complexity of k poly evaluations
- Logarithmic communication/rounds and verification time

Our results

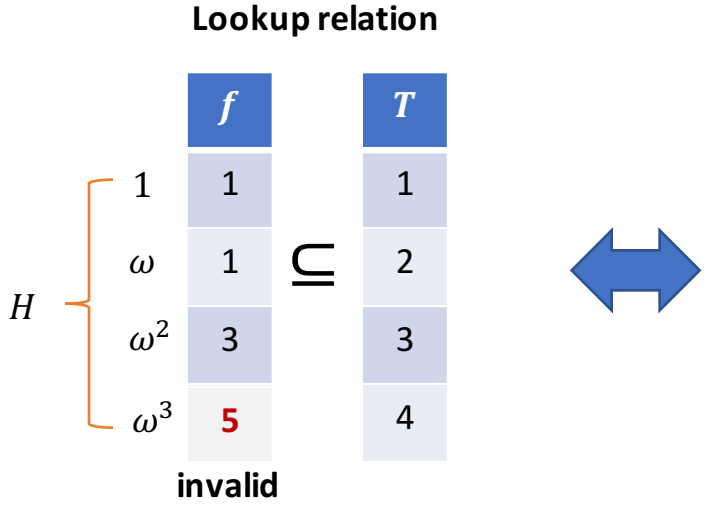
- Prover time $O(k2^\mu)$ (optimal)
- $k\mu$-factor improvement (k>13 in HP)
- $O(\mu)$ communication
- $O(k\mu)$ verification time

State-of-the-arts

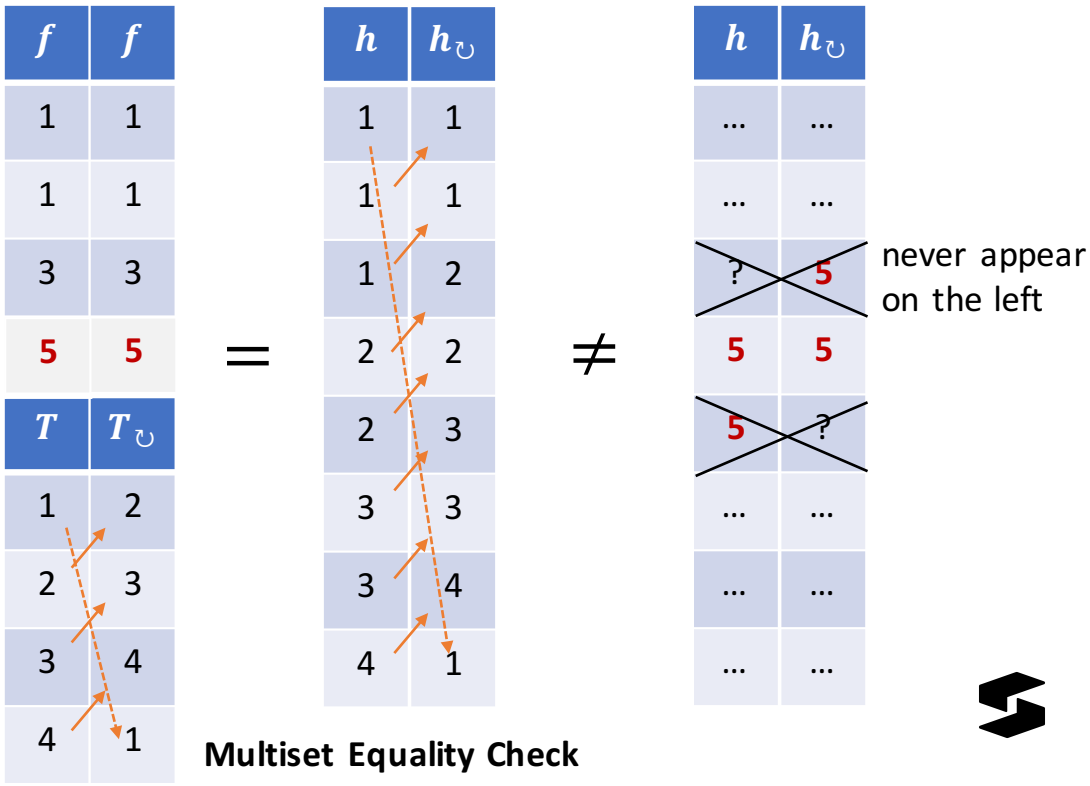
- Halo-infinite [BDFG20] only for univariate polys
- Thaler's solution has prover time $O(k^2\mu \cdot 2^\mu)$

The Idea of Plookup [GW20]

Lookup relation: $f(H) \subseteq T(H)$ for some preprocessed T and some witness f



- It's easy to perform rotation in H for univariate polynomials: $H \to \omega H$
- How to do it in the Boolean hypercube for multilinear polynomials?



HyperPlonk+: Open Problems

- **Shifting by k > 1 is useful in some applications (e.g. Halo2)**
- **Shifting by k is not as easy as shifting by 1**
  - The linearization trick has an exponential blow-up on the shift distance
  - Distance-k shift: 2^k evals on f to simulate an eval on $f_{\circlearrowleft}$
- **Can we implement a distance-k shift more efficiently?**