Lunar

a toolbox for more efficient universal and updatable
zkSNARKs and commit-and-prove extensions

SNARKs

Succinct Non Interactive ARguments of Knowledge

- NIZK for circuit SAT
- no Fiat-Shamir
- constant proof + linear prover
- circuit-specific setup
- linear trusted CRS
- linear verification

Pinocchio

* preprocessing SNARK
* no Fiat-Shamir
* 3G proof + linear prover
* circuit-specific setup
* linear trusted CRS
* constant verification

Pinocchio

’16

Groth16

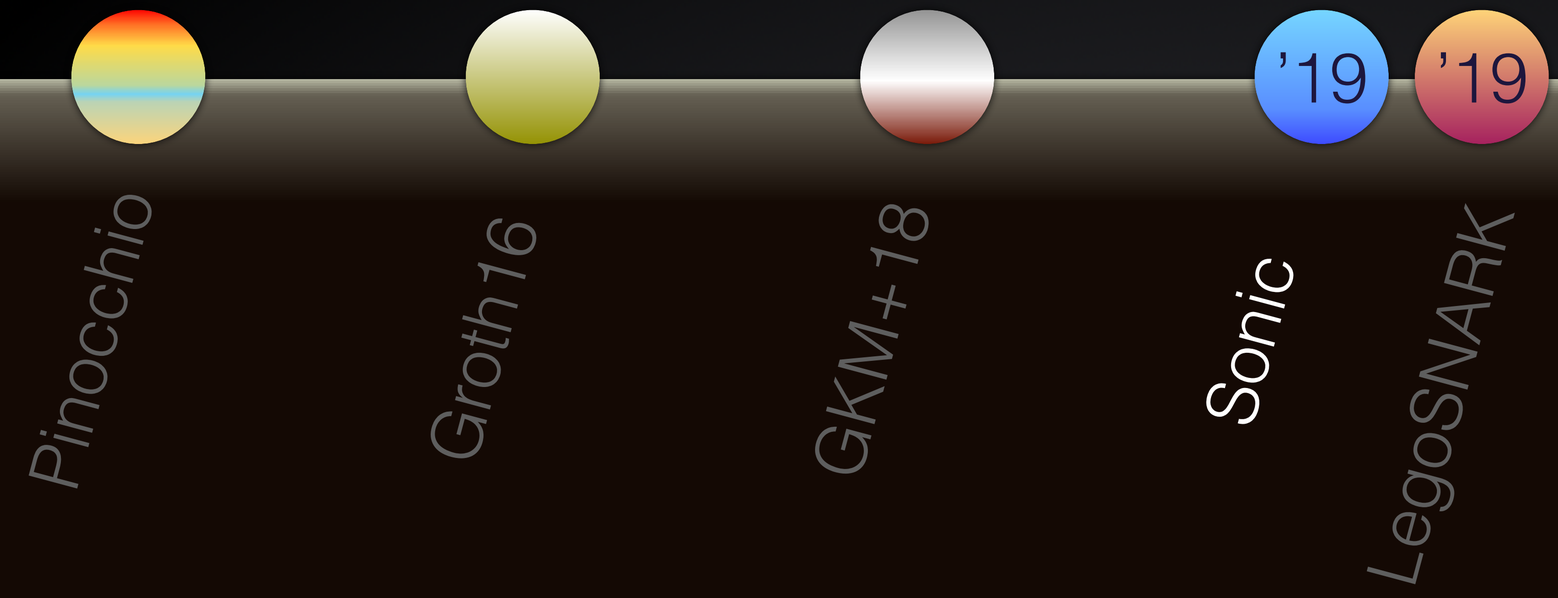
* universal updatable zkSNARKs
* no Fiat-Shamir
* 3G proof + linear prover
* universal setup
* quadratic updatable CRS
* constant verification

Pinocchio

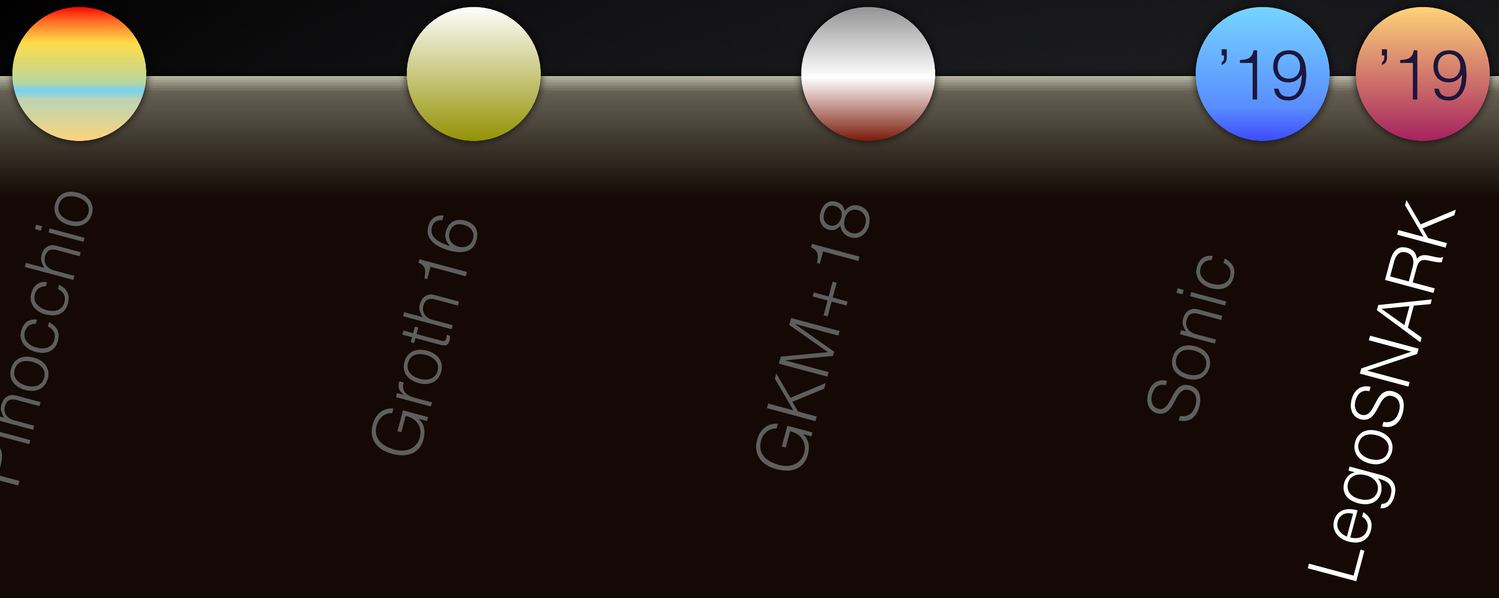
Groth16

GKM+18

* linear SRS universal updatable zkSNARKs
* Fiat-Shamir
* constant proof + quasilinear prover
* universal setup
* linear updatable SRS
* constant verification



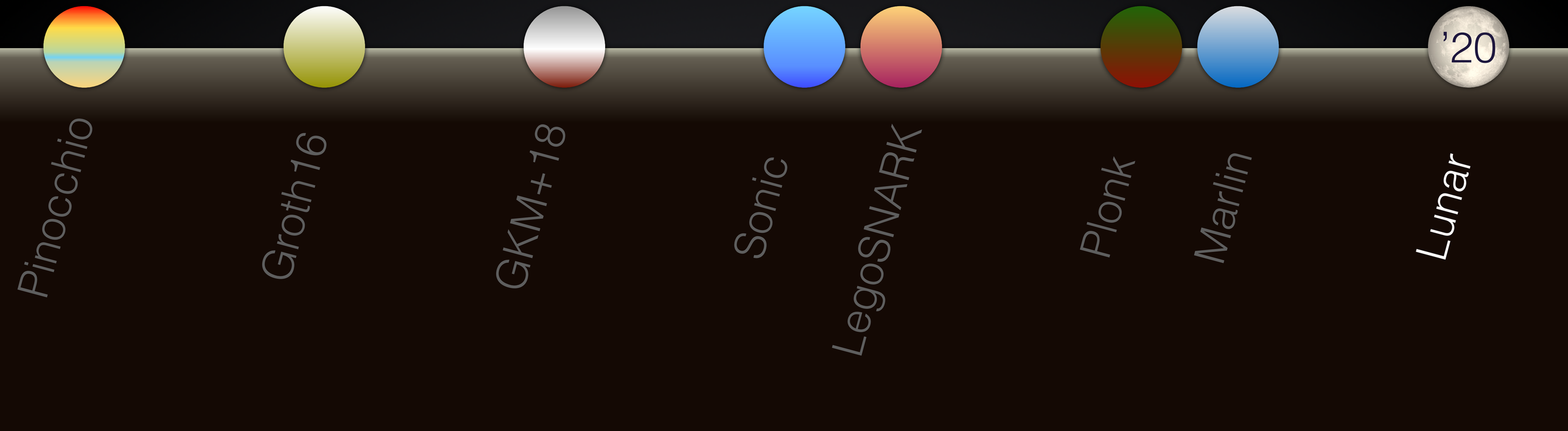
* linear SRS universal updatable zkSNARKs
* Fiat-Shamir
* polylog proof + linear prover
* universal setup
* linear updatable SRS
* constant verification



* linear SRS universal updatable zkSNARKs
* Fiat-Shamir
* (shorter) constant proof + (faster) quasilinear prover
* universal setup
* linear updatable SRS
* constant verification

Pinocchio *Groth16* *GKM+18* *Sonic* *LegoSNARK* *Plonk* *Marlin*

* linear SRS universal updatable CP-zkSNARKs
* Fiat-Shamir
* family of tradeoffs
* universal setup
* linear updatable SRS
* constant verification



Motivation

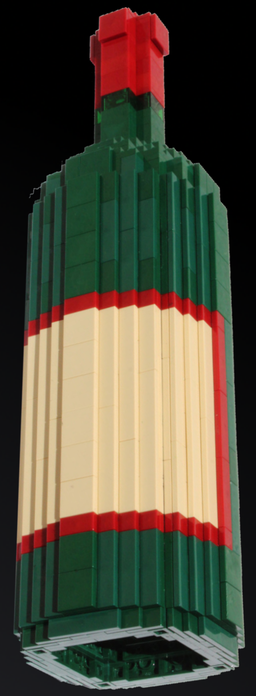
✳ fill in gaps of Marlin and Plonk

Motivation

* fill in gaps of Marlin and Plonk
* more comprehensive abstraction

Motivation

- fill in gaps of Marlin and Plonk
- more comprehensive abstraction
- compiler with modular security

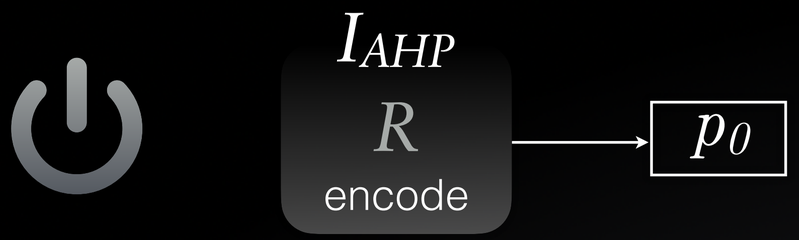


Motivation

- fill in gaps of Marlin and Plonk
- more comprehensive abstraction
- compiler with modular security
- efficiency through CP-SNARKs

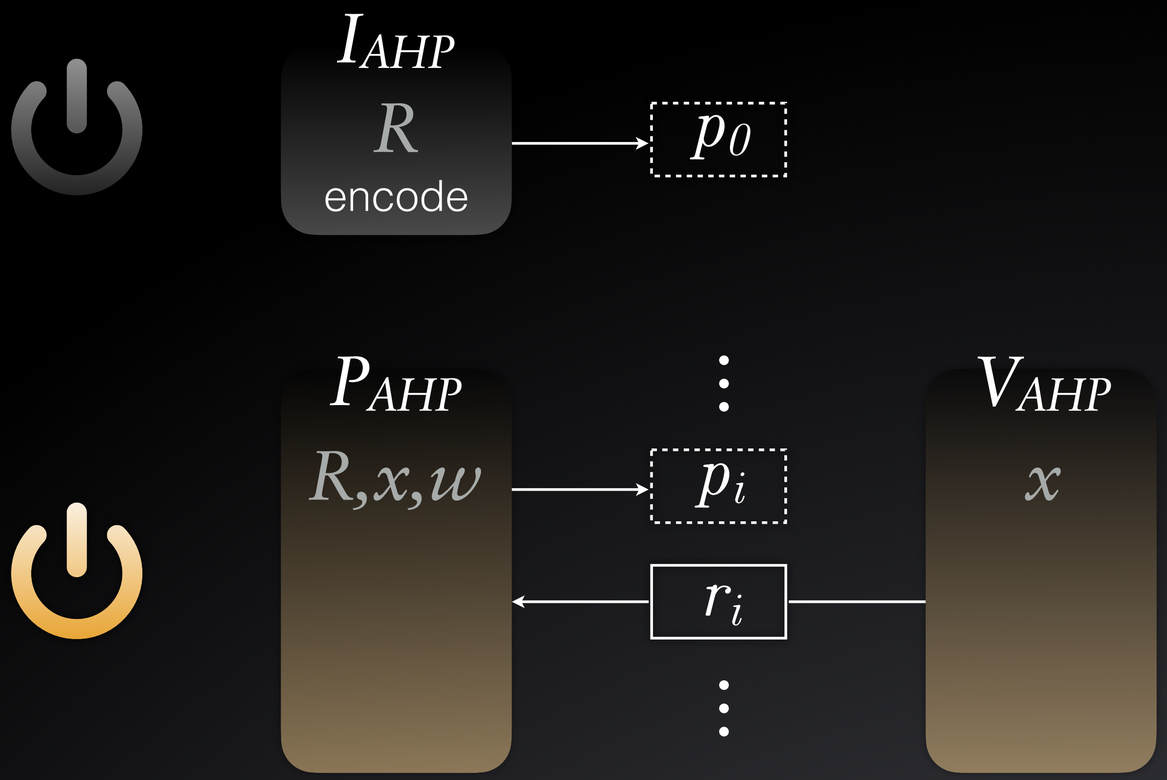
Marlin's AHP

algebraic holographic proof



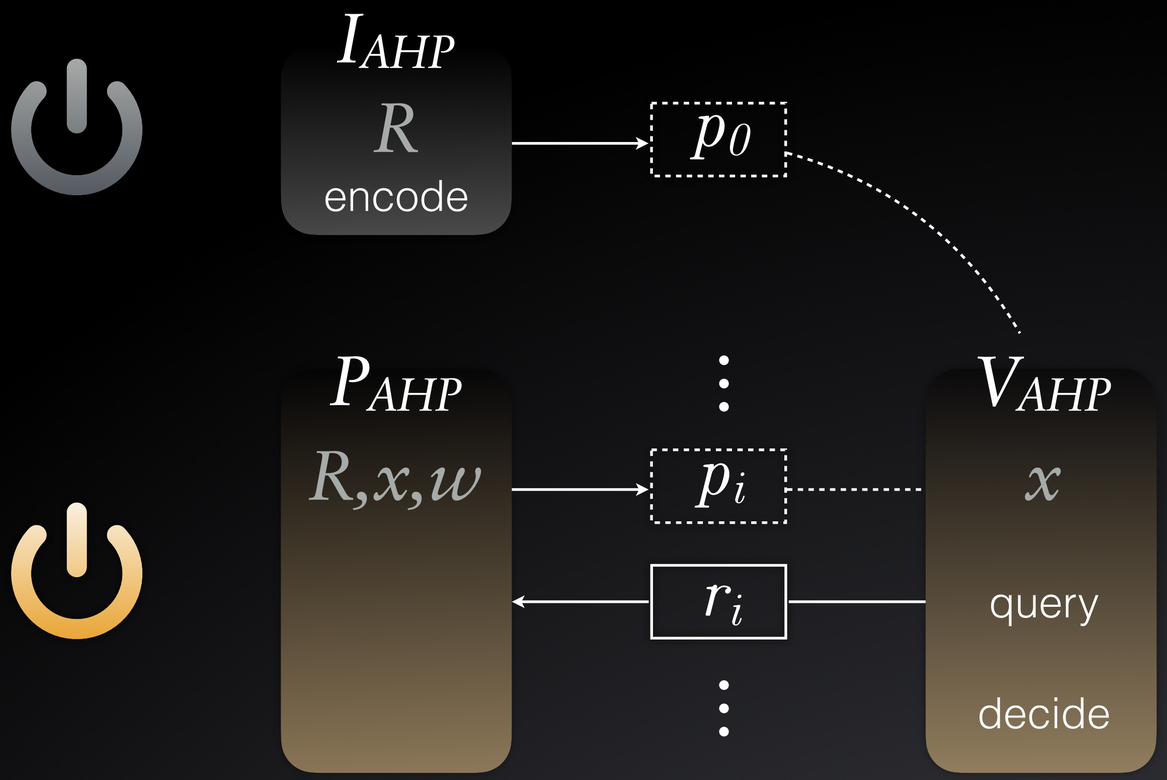
Marlin's AHP

algebraic holographic proof



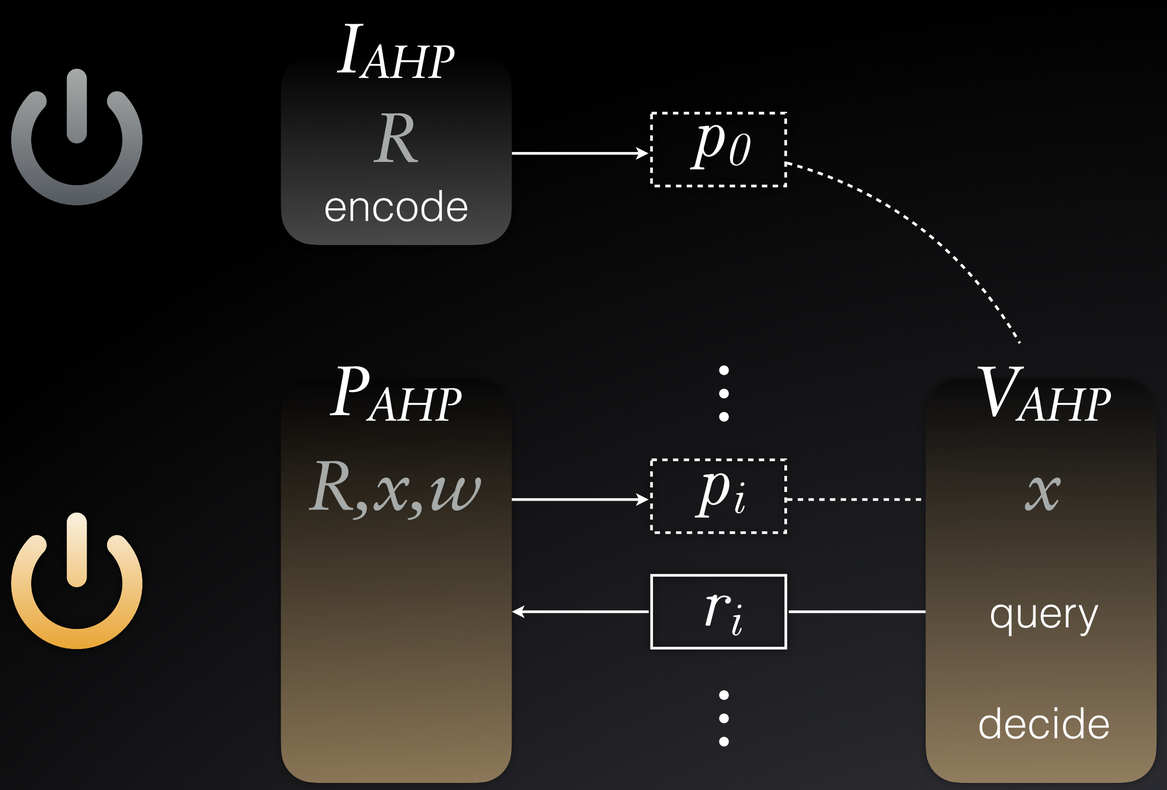
Marlin's AHP

algebraic holographic proof



Marlin's AHP

algebraic holographic proof

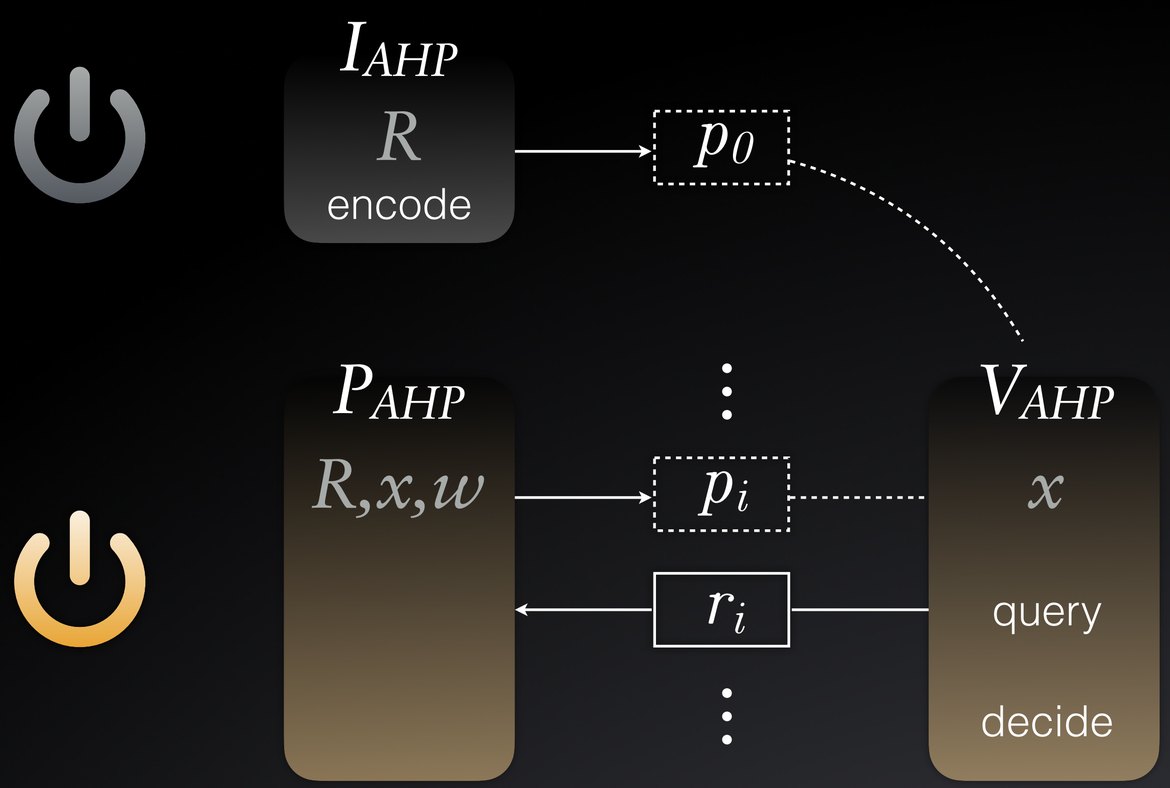


Plonk's ILDP

Idealised low degree protocol

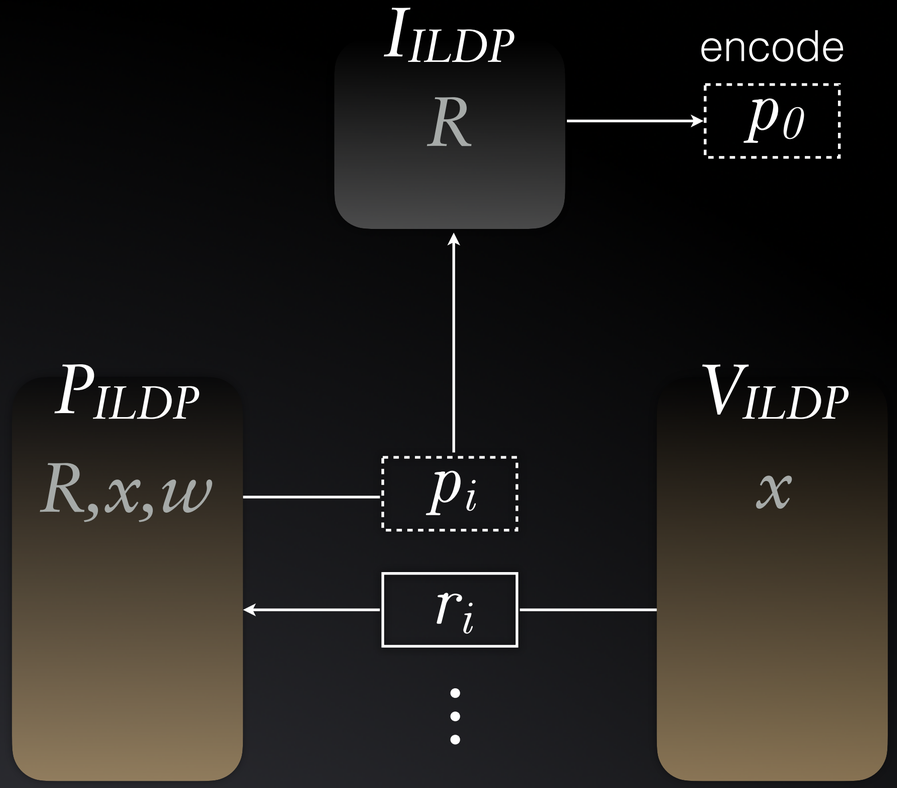
Marlin's AHP

algebraic holographic proof



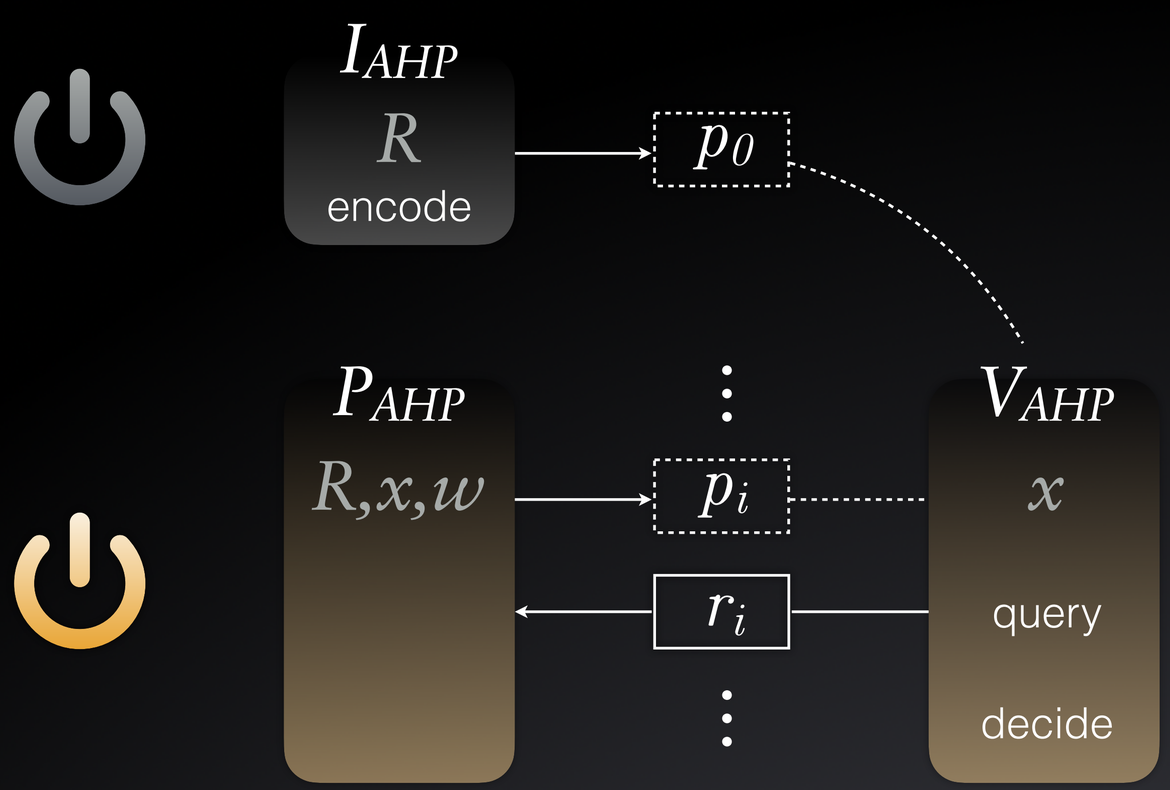
Plonk's ILDP

Idealised low degree protocol



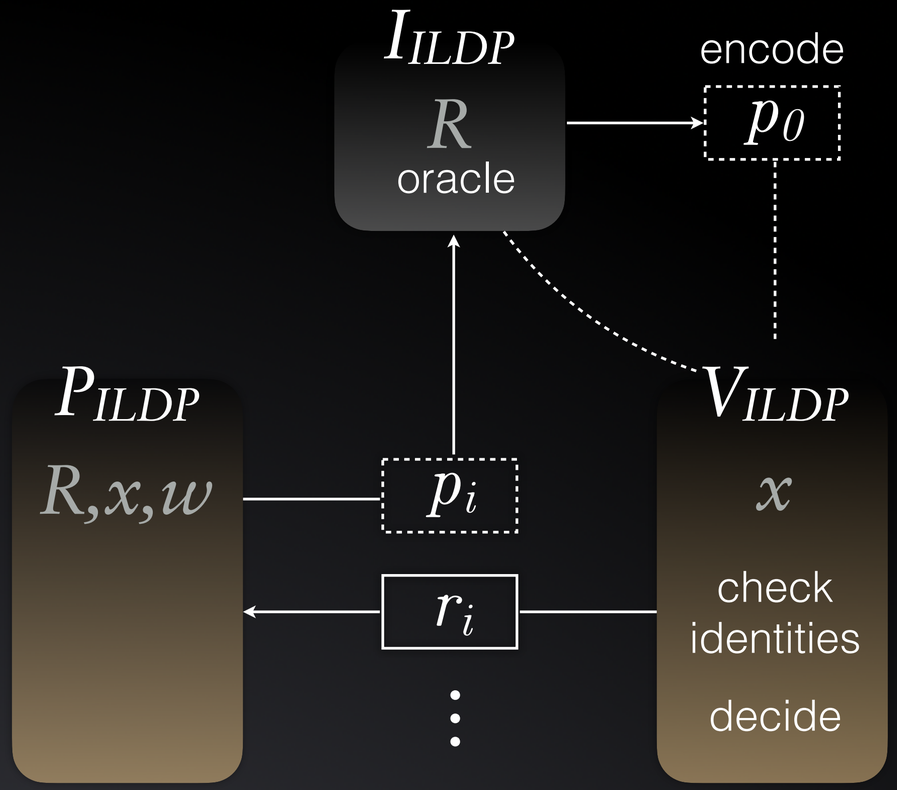
Marlin's AHP

algebraic holographic proof

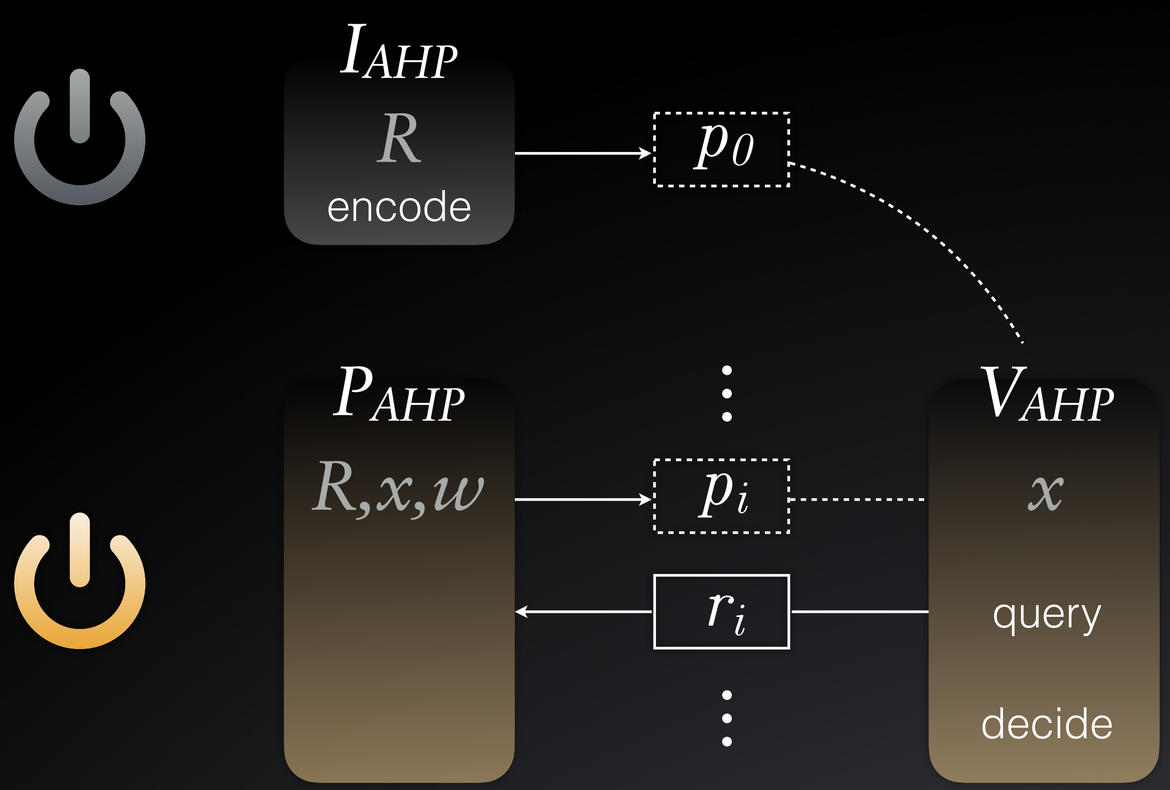


Plonk's ILDP

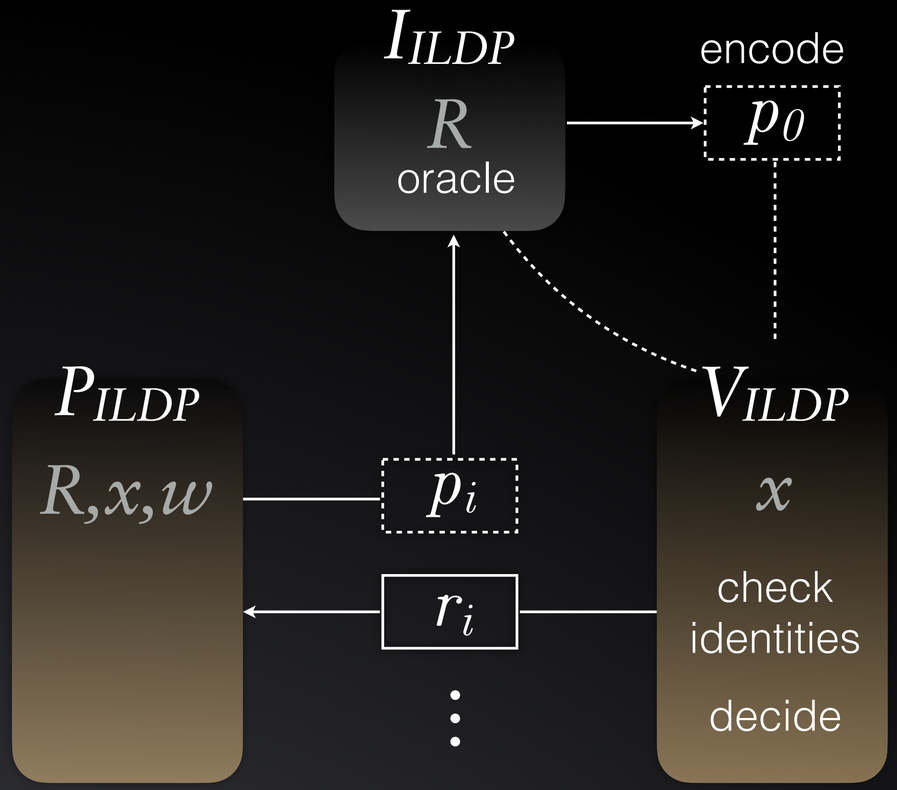
idealised low degree protocol



Marlin's AHP
algebraic holographic proof



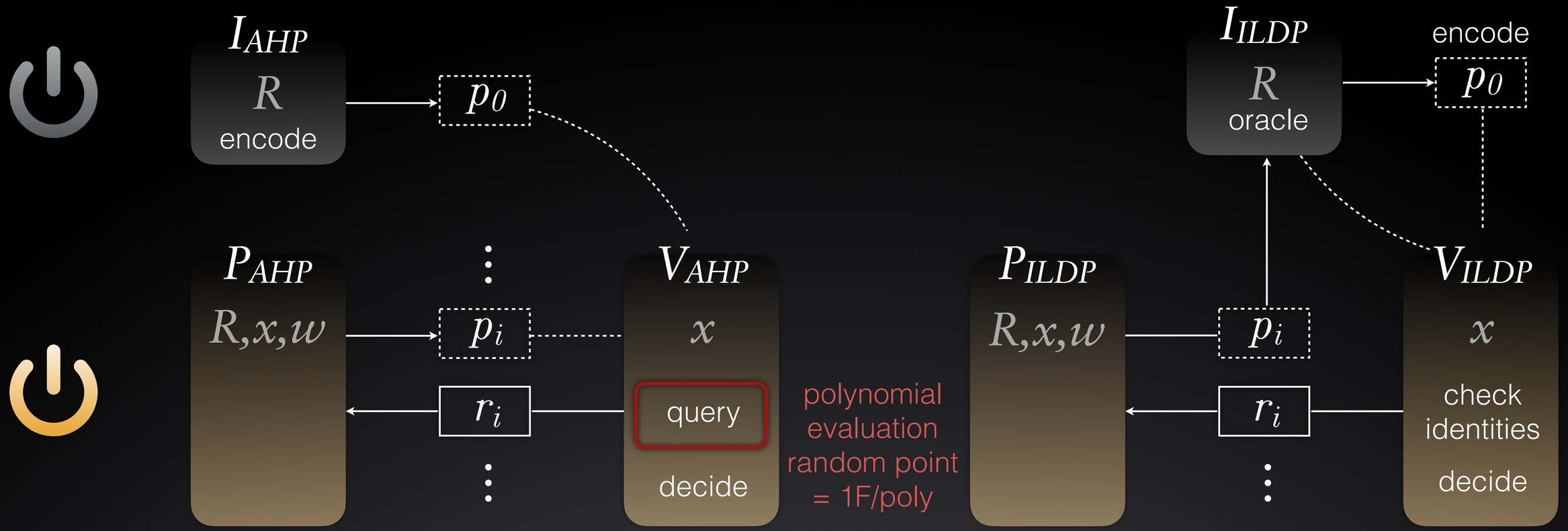
Plonk's ILDP
Idealised low degree protocol



algebraic IOP + opening polynomial commitments → SNARK

Marlin's AHP
algebraic holographic proof

Plonk's ILDP
Idealised low degree protocol

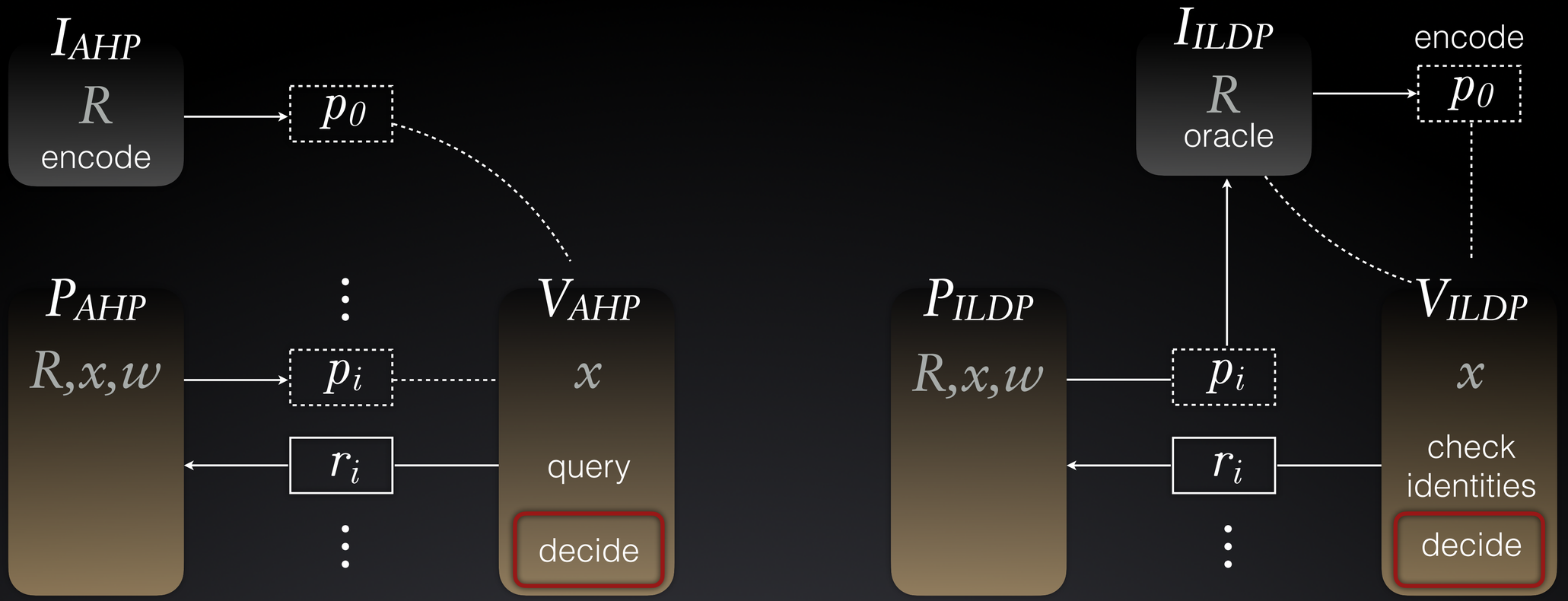


algebraic IOP + opening polynomial commitments → SNARK

Marlin's AHP
algebraic holographic proof

optimizations deviate from abstraction

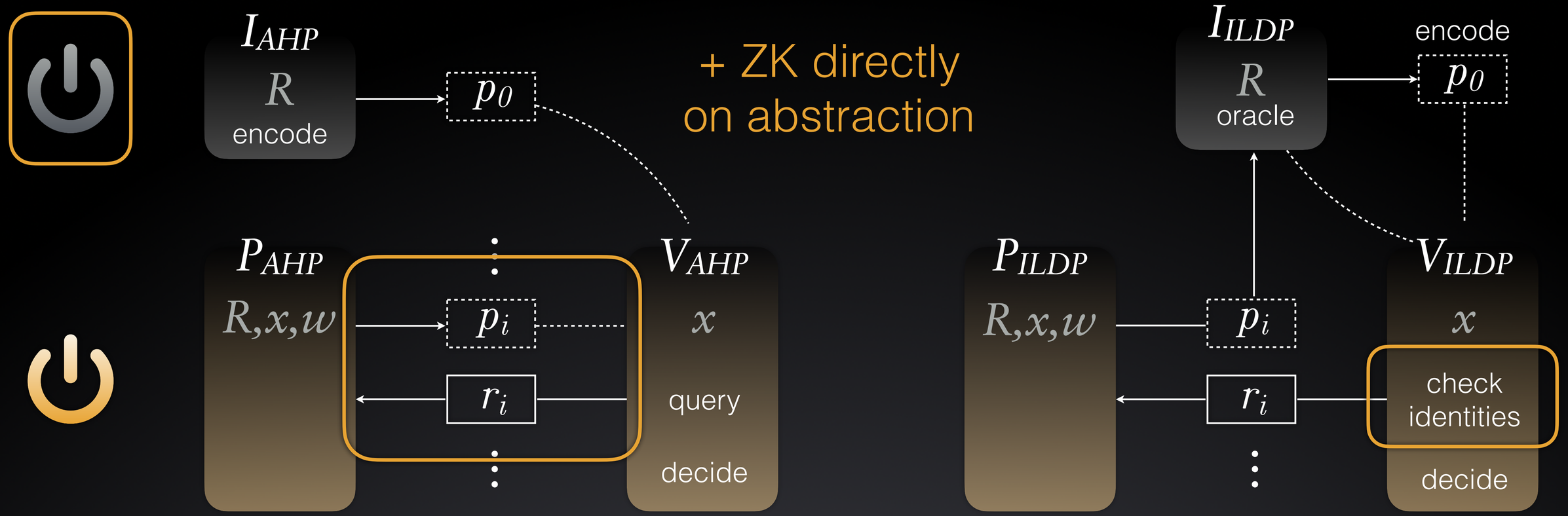
Plonk's ILDP
Idealised low degree protocol



algebraic IOP + opening polynomial commitments → SNARK

Lunar's PHP

polynomial holographic proof



general algebraic IOP + CP-SNARKs → Lunar SNARKs

Lunar's PHP

polynomial holographic proof

	AHP	ILDP	PHP
Preprocessing	yes	yes	yes
Proof	batch	short	direct
Decision	polynomial evaluations	identity checks	identity checks
ZK	not native	not native	within abstraction
Compilation	polycom	polycom	modular gadgets
CP	no	no	yes

naive: hiding commitments + perfect zero knowledge → zkSNARKs

Lunar's PHP

polynomial holographic proof

	AHP	ILDP	PHP
Preprocessing	yes	yes	yes
Proof	batch	short	direct
Decision	polynomial evaluations	identity checks	identity checks
ZK	not native	not native	within abstraction
Compilation	polycom	polycom	modular gadgets
CP	no	no	yes

thm: *somewhat* *hiding* commitments + *bounded* ~~perfect~~ zero knowledge → zkSNARKs

R1CS lite

PHP lite

Compiler

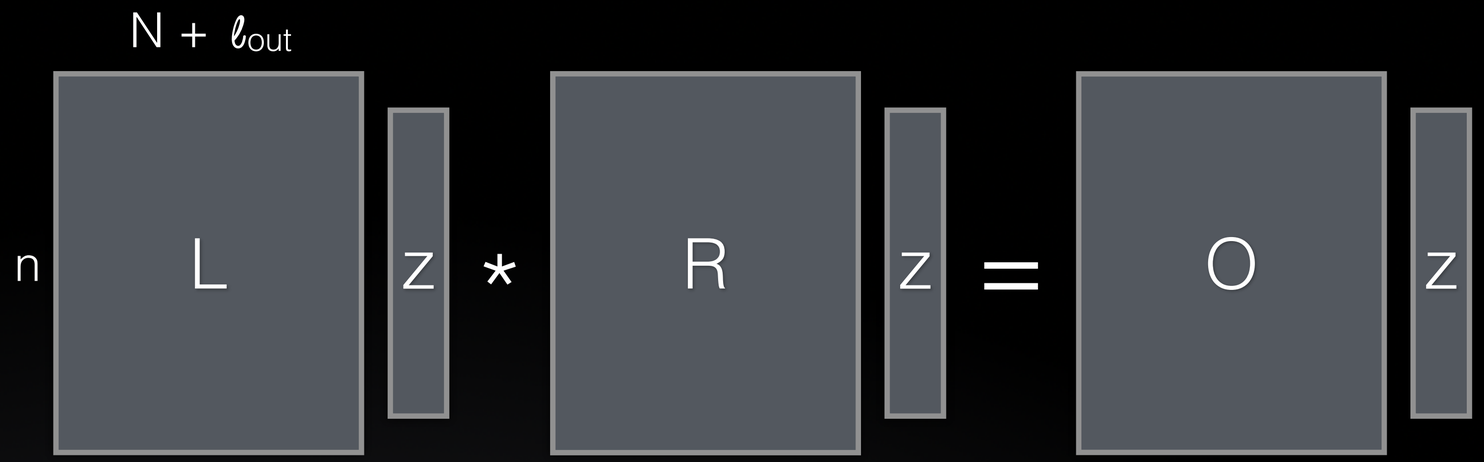
R1CS

$z =$ | 1 | x | w |

$$n = 1 + \ell_{in} + \ell_{out} + N$$

$$\| $$

$$\otimes$$



R1CS

$z = [\ 1 \mid x \mid w\ ]$

$n = 1 + \ell_{in} + \ell_{out} + N$

$N + \ell_{out}$

n | L | z $*$ R | z $=$ O | z

$\otimes$

R1CS-lite

$c = [\ 1 \mid x \mid w\ ]$

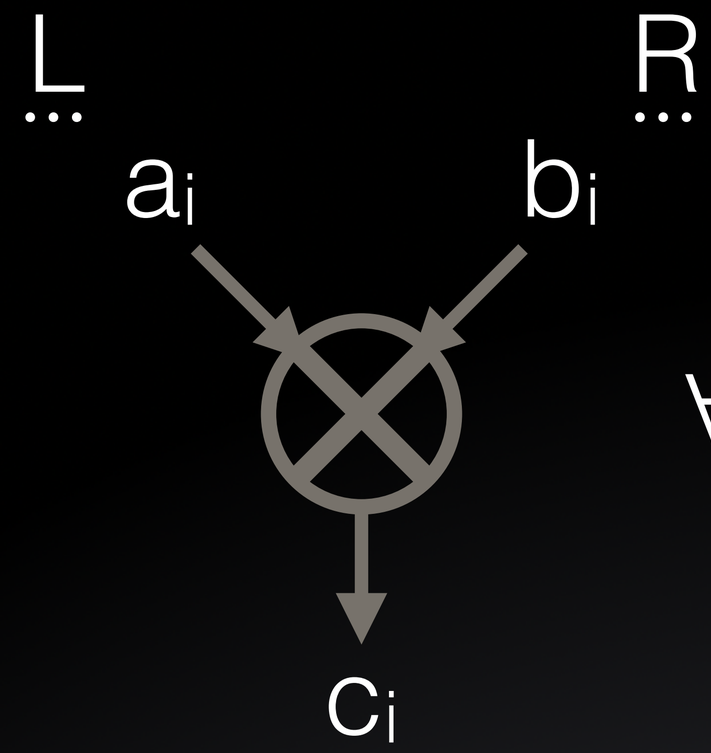
n

n | L' | c $*$ R' | c $=$ c

R1CS lite

PHP lite

Compiler



$$\forall i \in I_{mid} \begin{cases} a_i - L_i \cdot c = 0 \\ b_i - R_i \cdot c = 0 \\ c_i - a_i \cdot b_i = 0 \end{cases}$$

R1CS-lite

c = 

R1CS lite

PHP lite

Compiler

$$\forall i \in I_{out} \begin{cases} a_i - L_i \cdot c = 0 \\ b_i - c_1 = 0 \\ c_i - a_i \cdot b_i = 0 \end{cases}$$

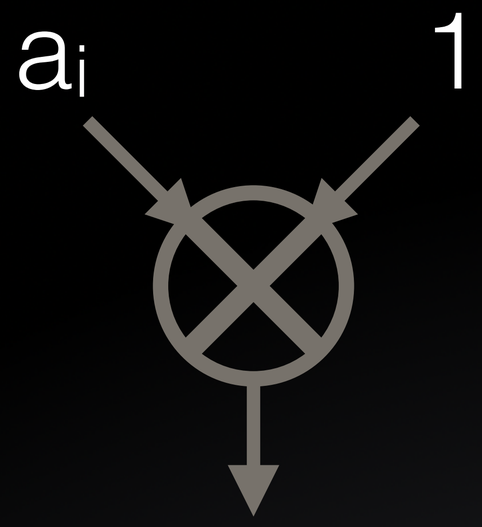
R1CS-lite

$c =$ 

R1CS lite

PHP lite

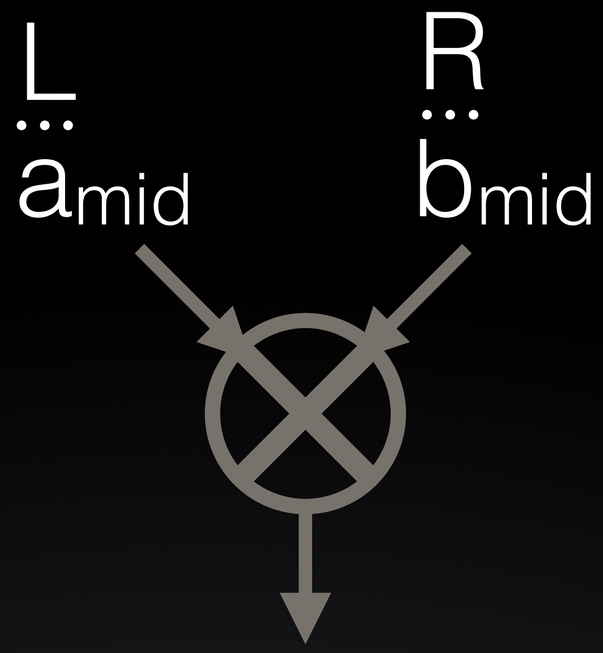
Compiler



$$\forall i \in I_{in} \quad \begin{cases} a_i - c_i = 0 \\ b_i - c_1 = 0 \\ c_i - a_i \cdot b_i = 0 \end{cases}$$

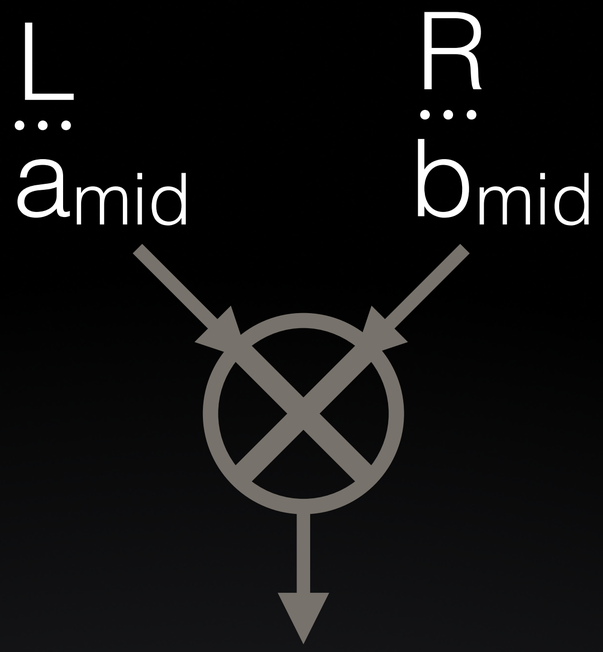
R1CS-lite

c = | 1 | x y | w |



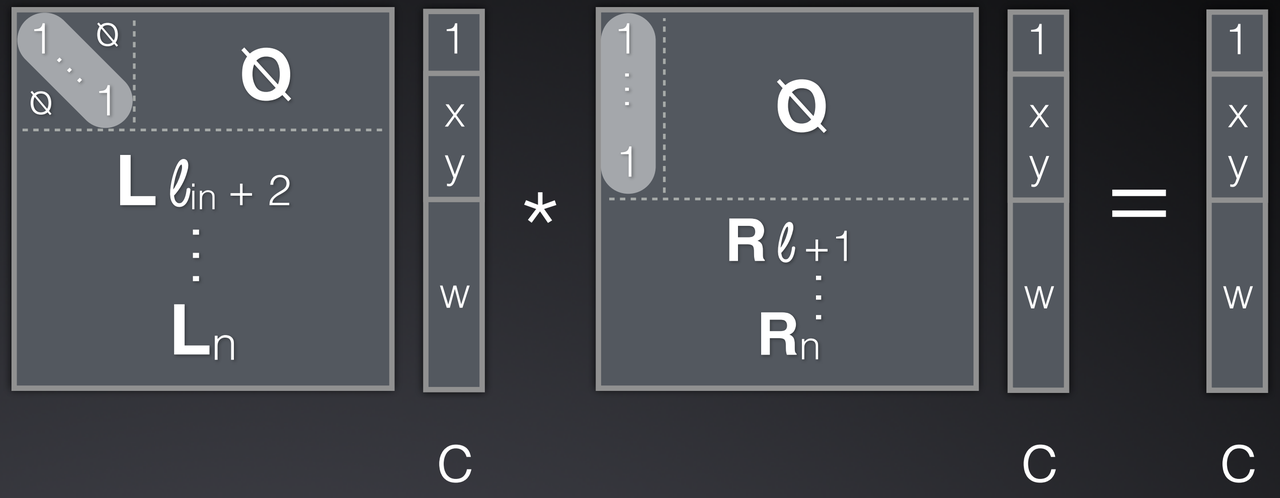
R1CS-lite

c = | 1 | x y | w |



R1CS-lite

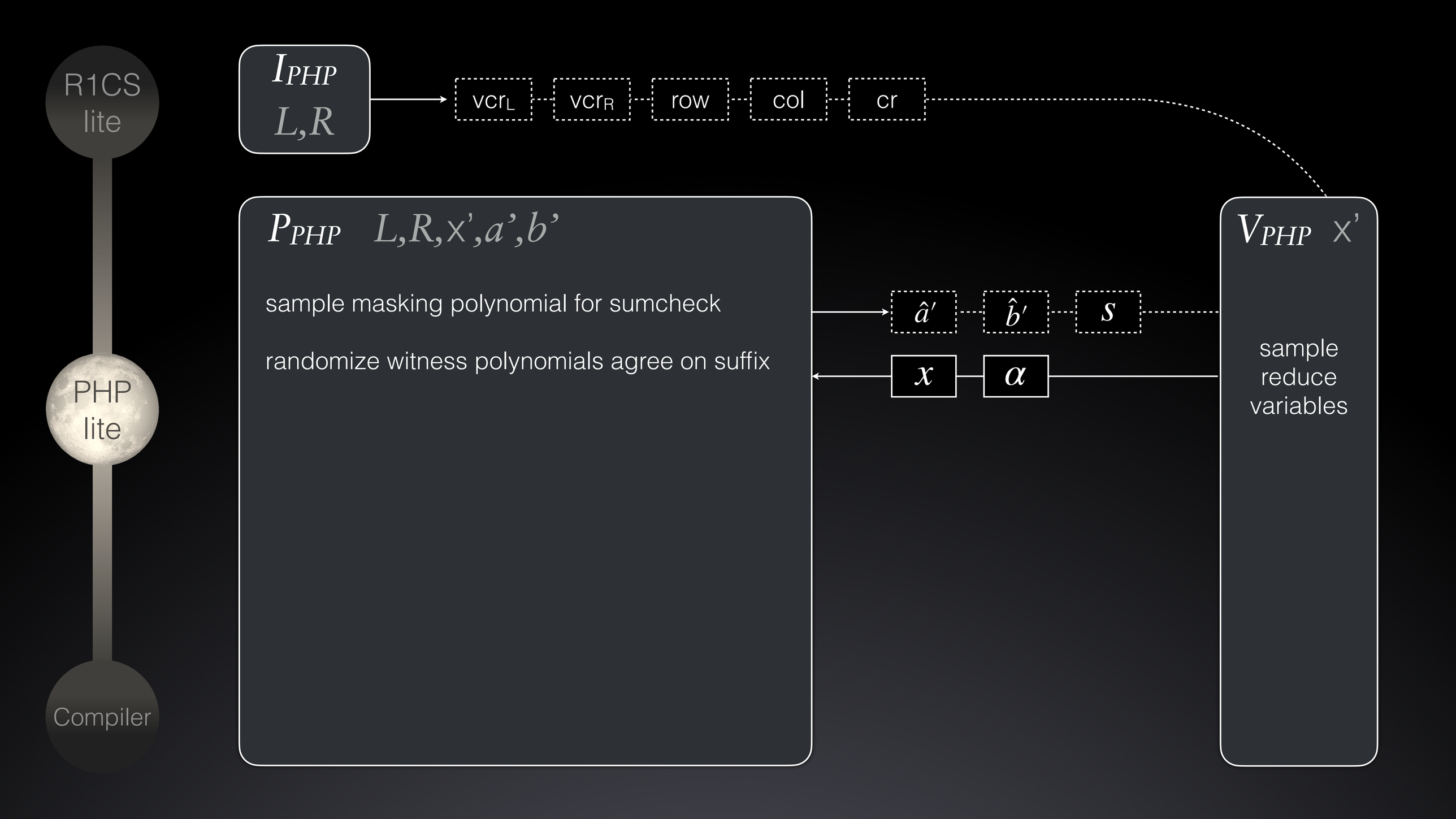
$\ell_{in} + 1$ more columns

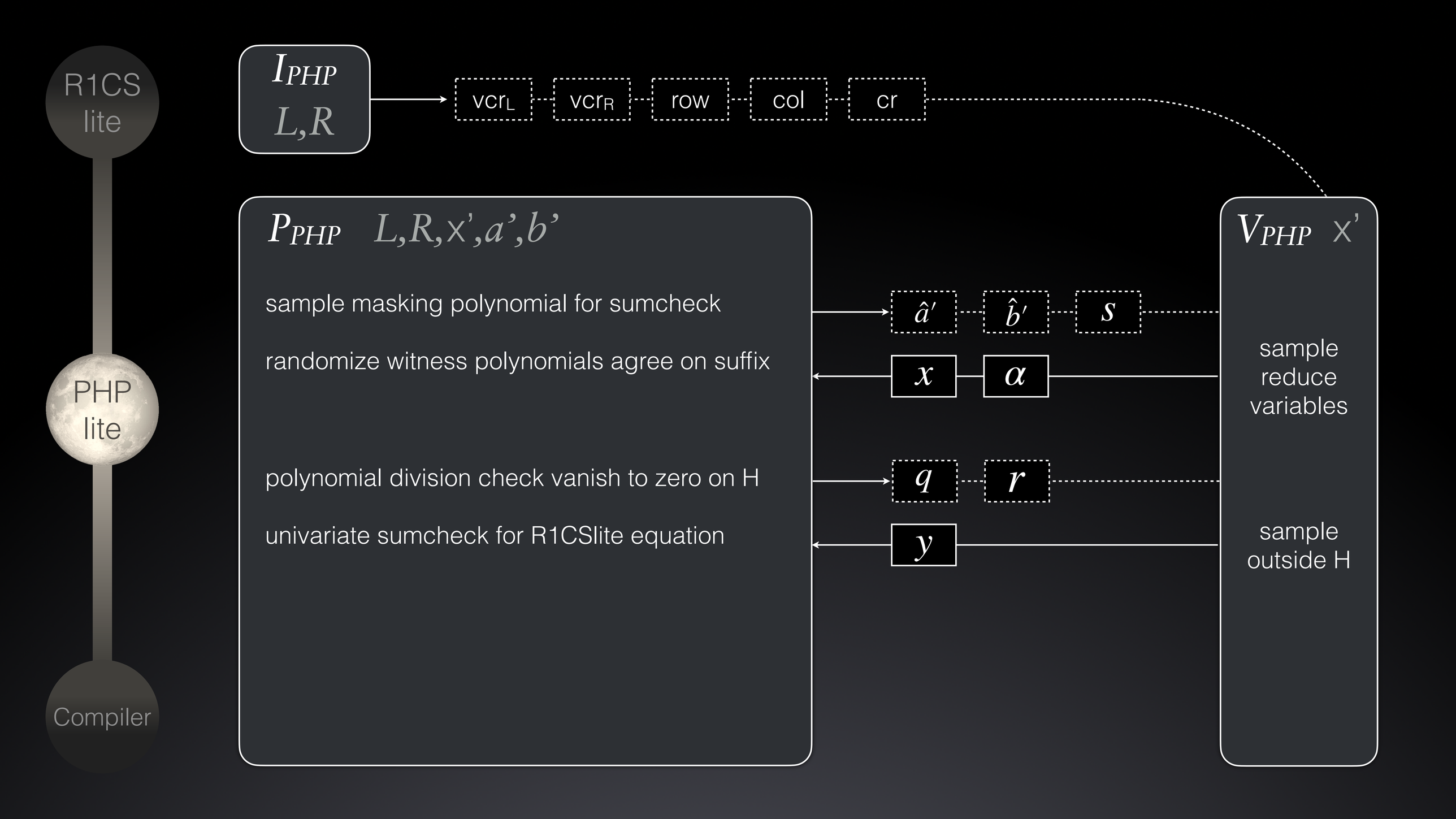


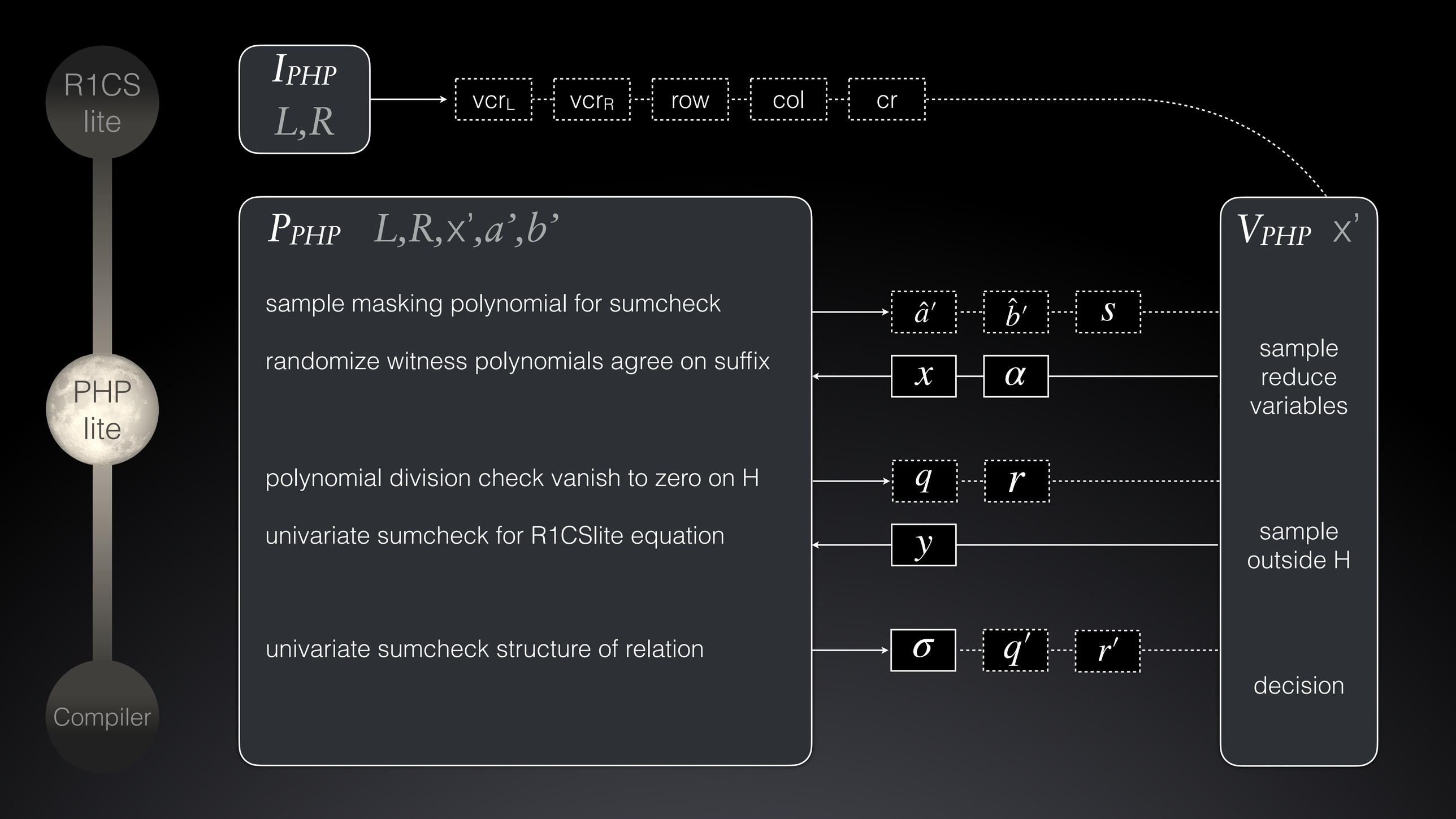
I_{PHP}
L,R

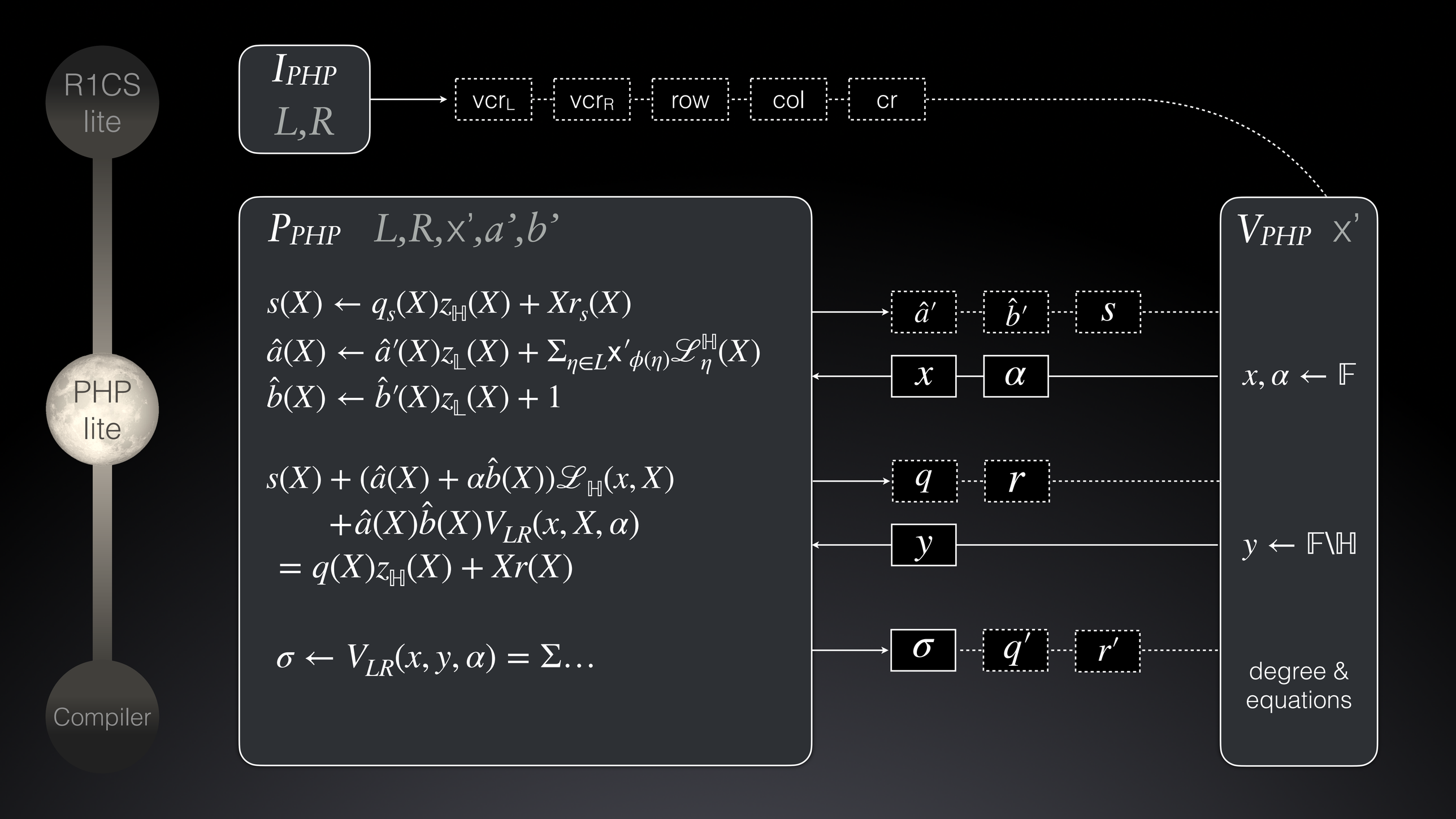
vcr_L vcr_R row col cr

low degree encodings of frequent compuations with R1CS-lite matrices


R1CS lite
PHP lite
Compiler
I_{PHP} L,R
vcr_L
vcr_R
row
col
cr
P_{PHP} L,R,x',a',b'
sample masking polynomial for sumcheck
randomize witness polynomials agree on suffix
$\hat{a}'$
$\hat{b}'$
s
x
α
V_{PHP} x'
sample reduce variables


R1CS lite
PHP lite
Compiler
I_{PHP} L,R
vcr_L
vcr_R
row
col
cr
P_{PHP} L,R,x',a',b'
sample masking polynomial for sumcheck
randomize witness polynomials agree on suffix
polynomial division check vanish to zero on H
univariate sumcheck for R1CSlite equation
$\hat{a}'$
$\hat{b}'$
s
x
α
q
r
y
V_{PHP} x'
sample reduce variables
sample outside H


R1CS lite
PHP lite
Compiler
I_{PHP} L,R
vcr_L
vcr_R
row
col
cr
P_{PHP} L,R,x',a',b'
sample masking polynomial for sumcheck
randomize witness polynomials agree on suffix
polynomial division check vanish to zero on H
univariate sumcheck for R1CSlite equation
univariate sumcheck structure of relation
$\hat{a}'$
$\hat{b}'$
s
x
α
q
r
y
σ
q'
r'
V_{PHP} x'
sample reduce variables
sample outside H
decision


R1CS lite
PHP lite
Compiler
I_{PHP} L,R
vcr_L
vcr_R
row
col
cr
P_{PHP} L,R,x',a',b'
$s(X) \leftarrow q_s(X)z_{\mathbb{H}}(X) + Xr_s(X)$
$\hat{a}(X) \leftarrow \hat{a}'(X)z_{\mathbb{L}}(X) + \Sigma_{\eta \in L}\mathrm{x}'_{\phi(\eta)}\mathscr{L}^{\mathbb{H}}_{\eta}(X)$
$\hat{b}(X) \leftarrow \hat{b}'(X)z_{\mathbb{L}}(X) + 1$
$s(X) + (\hat{a}(X) + \alpha\hat{b}(X))\mathscr{L}_{\mathbb{H}}(x, X)$
$+\hat{a}(X)\hat{b}(X)V_{LR}(x, X, \alpha)$
$= q(X)z_{\mathbb{H}}(X) + Xr(X)$
$\sigma \leftarrow V_{LR}(x, y, \alpha) = \Sigma\ldots$
$\hat{a}'$
$\hat{b}'$
s
x
α
q
r
y
σ
q'
r'
V_{PHP} x'
$x, \alpha \leftarrow \mathbb{F}$
$y \leftarrow \mathbb{F}\backslash\mathbb{H}$
degree & equations

Decision phase

Degree checks

- maximum degrees (soundness)
- remainder degree (completeness)

R1CS lite

PHP lite

Compiler

Decision phase

Degree checks

$$\deg(\hat{a}', \hat{b}', s, q, q') \leq \mathsf{D}_{\mathsf{snd}}$$

$$\deg(r) \leq n - 2 \wedge \deg(r') \leq |\mathbb{K}| - 2$$

R1CS lite

PHP lite

Compiler

Decision phase

Degree checks

$\mathsf{deg}(\hat{a}', \hat{b}', s, q, q') \leq \mathsf{D}_{\mathsf{snd}}$

$\mathsf{deg}(r) \leq n - 2 \wedge \mathsf{deg}(r') \leq |\mathbb{K}| - 2$

Polynomial checks

random point evaluations (≈ AHP)

equation identities (≈ ILDP)

Decision phase

Degree checks

$$\deg(\hat{a}', \hat{b}', s, q, q') \leq \mathsf{D}_{\mathsf{snd}}$$

$$\deg(r) \leq n - 2 \wedge \deg(r') \leq |\mathbb{K}| - 2$$

Polynomial checks

$$s(y) + (\hat{a}(y) + \alpha\hat{b}(y))\mathscr{L}_{\mathbb{H}}(x, y) + \hat{a}(y)\hat{b}(y)\sigma - q(y)z_{\mathbb{H}}(y) - yr(y) = 0$$

$$n^2\Big(Xr'(X) + \sigma/|\mathbb{K}|\Big)\Big(xy + \mathsf{cr}(X) - x\mathsf{col}(X) - y\mathsf{row}(X)\Big)$$
$$-\Big(\mathsf{vcr}_{\mathsf{L}}(X) + \alpha\mathsf{vcr}_{\mathsf{R}}(X)\Big)z_{\mathbb{H}}(x)v_{\mathbb{H}}(y) - q'(X)z_{\mathbb{K}}(X) = 0$$

R1CS lite

PHP lite

Compiler

Leaky ZK

can simulate interaction with a pool of at most b leaks

how many evaluations are really needed?

$$s(y) + (\hat{a}(y) + \alpha\hat{b}(y))\mathscr{L}_{\mathbb{H}}(x,y) + \hat{a}(y)\hat{b}(y)\sigma - q(y)z_{\mathbb{H}}(y) - yr(y) = 0$$

$$n^2\left(Xr'(X) + \sigma/|\mathbb{K}|\right)\left(xy + \mathsf{cr}(X) - x\mathsf{col}(X) - y\mathsf{row}(X)\right)$$
$$-\left(\mathsf{vcr}_{\mathsf{L}}(X) + \alpha\mathsf{vcr}_{\mathsf{R}}(X)\right)z_{\mathbb{H}}(x)v_{\mathbb{H}}(y) - q'(X)z_{\mathbb{K}}(X) = 0$$

R1CS lite

PHP lite

Compiler

Leaky ZK

can simulate interaction with a pool of at most b leaks

how many evaluations are really needed?

this PHP is (0,0,1,0,0,∞,∞)-bounded ZK

$$s(y) + (\hat{a}(y) + \alpha\hat{b}(y))\mathscr{L}_{\mathbb{H}}(x,y) + \hat{a}(y)\hat{b}(y)\sigma - q(y)z_{\mathbb{H}}(y) - yr(y) = 0$$

Pairings!

(witness-independent check, no privacy needed)

CS: type-based polynomial commitment scheme *in the exponent* • rel • swh

$\prod$.KeyGen(1^λ, N) → srs

CS.Setup(d) → ck monomials in exp

CP_{php}.KeyGen(ck) → $ek_{\text{php}}, vk_{\text{php}}$

CP_{opn}.KeyGen(ck) → $ek_{\text{opn}}, vk_{\text{opn}}$

R1CS lite

PHP lite

Compiler

CS: type-based polynomial commitment scheme *in the exponent* • rel • swh

$\prod$.KeyGen(1^λ, N) → srs

CS.Setup(d) → ck monomials in exp

CP_{php}.KeyGen(ck) → $ek_{\text{php}}, vk_{\text{php}}$

CP_{opn}.KeyGen(ck) → $ek_{\text{opn}}, vk_{\text{opn}}$

$\prod$.Derive(ck, srs, R) → srs_R

CS.Commit(ck, I_{PHP} R)

$ek_R := ek \cup p_0,\ o_0$

$vk_R := vk \cup p_0 = [p_0(\$)]_{1 \vee 2}$

CS1 or CS2

R1CS lite

PHP lite

Compiler

CS: type-based polynomial commitment scheme *in the exponent*

•rel •swh

$\prod$.KeyGen(1^λ, N) → srs

CS.Setup(d) → ck monomials in exp

CP$_{\text{php}}$.KeyGen(ck) → $ek_{\text{php}}, vk_{\text{php}}$

CP$_{\text{opn}}$.KeyGen(ck) → $ek_{\text{opn}}, vk_{\text{opn}}$

$\prod$.Prove(ek_R,x,w) → π

i CS.Commit(ck, $P_{PHP}^{i,\rho}$) FS

CP$_{\text{opn}}$.Prove(ek_{opn}, p_i , o_i)

π = ({ p_i , m_i, π_{opn_i} }, π_{php})

proof that a V_{PHP} would accept

$\prod$.Derive(ck, srs, R) → srs_R

CS.Commit(ck, I_{PHP}^{R})

$ek_R := ek \cup p_0,\ o_0$

$vk_R := vk \cup p_0 = [p_0(\$)]_{1 \vee 2}$

CS$_1$ or CS$_2$

R1CS lite

PHP lite

Compiler

CS: type-based polynomial commitment scheme *in the exponent*

•rel •swh

$\prod$.KeyGen(1^λ, N) → srs

CS.Setup(d) → ck monomials in exp

CP$_{\text{php}}$.KeyGen(ck) → $ek_{\text{php}}, vk_{\text{php}}$

CP$_{\text{opn}}$.KeyGen(ck) → $ek_{\text{opn}}, vk_{\text{opn}}$

$\prod$.Derive(ck, srs, R) → srs_R

CS.Commit(ck, I_{PHP} / R)

$ek_R := ek \cup p_0,\ o_0$ CS$_1$ or CS$_2$

$vk_R := vk \cup$ p_0 $= [p_0(\$)]_{1 \vee 2}$

$\prod$.Prove(ek_R,x,w) → π

i CS.Commit(ck, P_{PHP} / i,ρ) FS

CP$_{\text{opn}}$.Prove(ek_{opn}, p_i , o_i)

$\pi = (\ \{\ p_i\ , m_i, \pi_{\text{opn}_i}\ \}, \pi_{\text{php}}\ )$

proof that a V_{PHP} would accept

$\prod$.Verify(vk_R,x,π) → ok/ko

CP$_{\text{php}}$.Verify(vk_{php},*deg*,*eqs*, checks p_0 p_1 … p_r , π_{php})

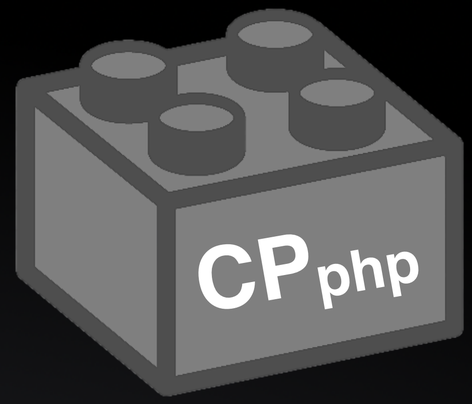
CP$_{\text{opn}}$.Verify(vk_{opn} , p_i , π_{opn_i})

R1CS lite

PHP lite

Compiler

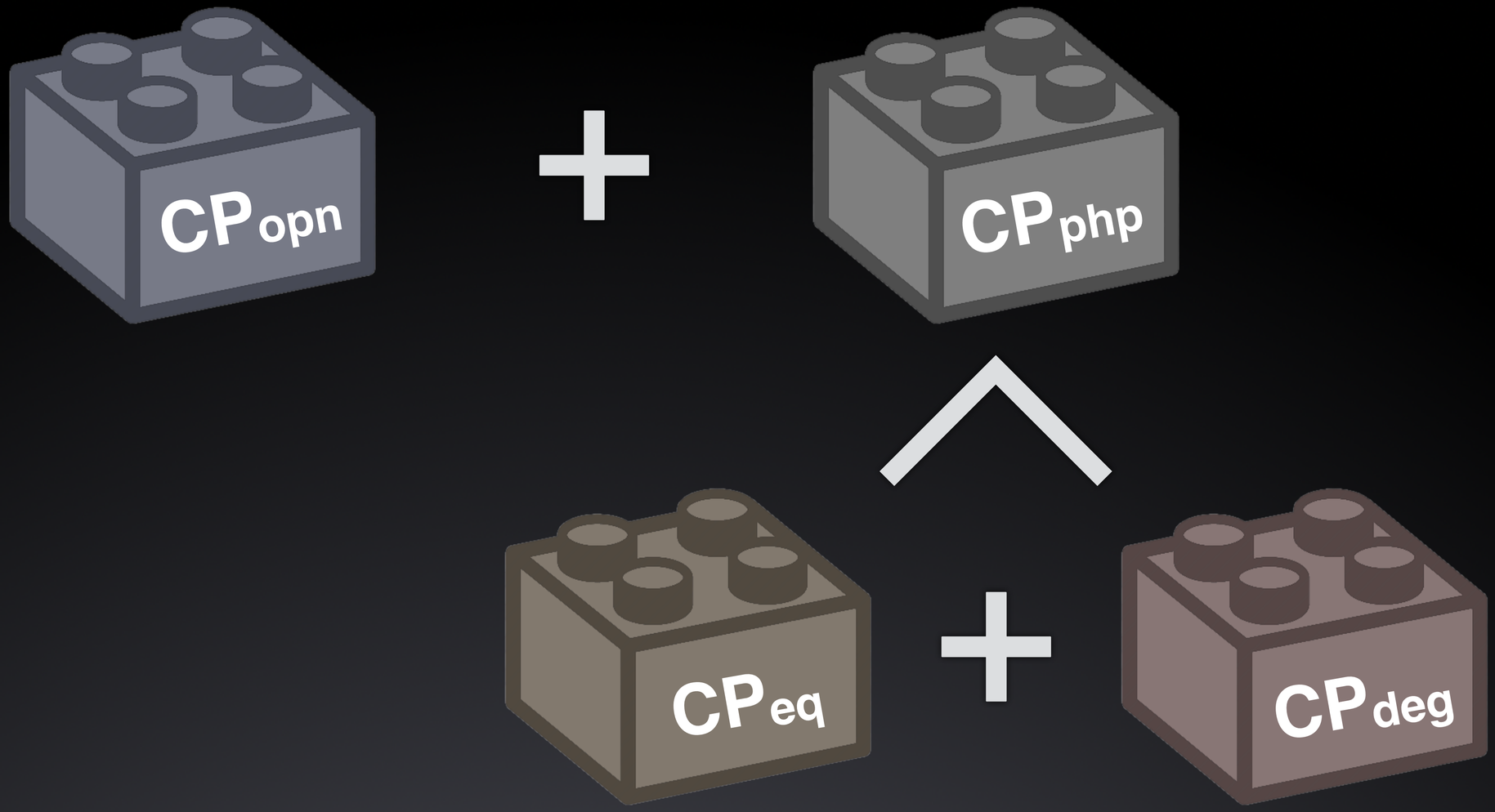
+



R1CS lite

PHP lite

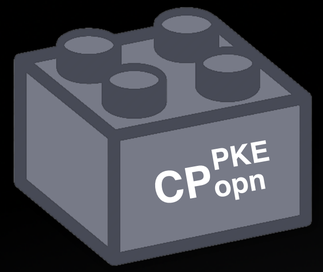
Compiler



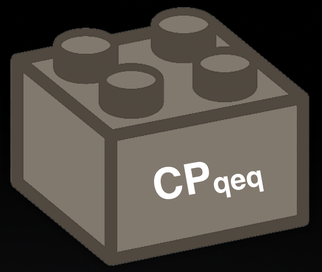
R1CS lite

PHP lite

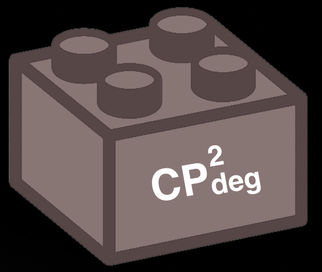
Compiler



novel batch ℓ com only 1 G

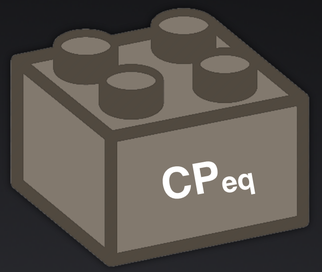


novel empty proof

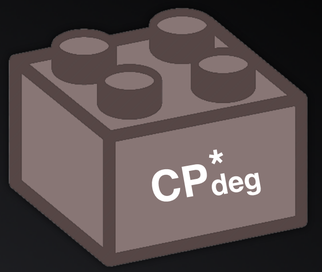


commit to shifted polynomial, batch

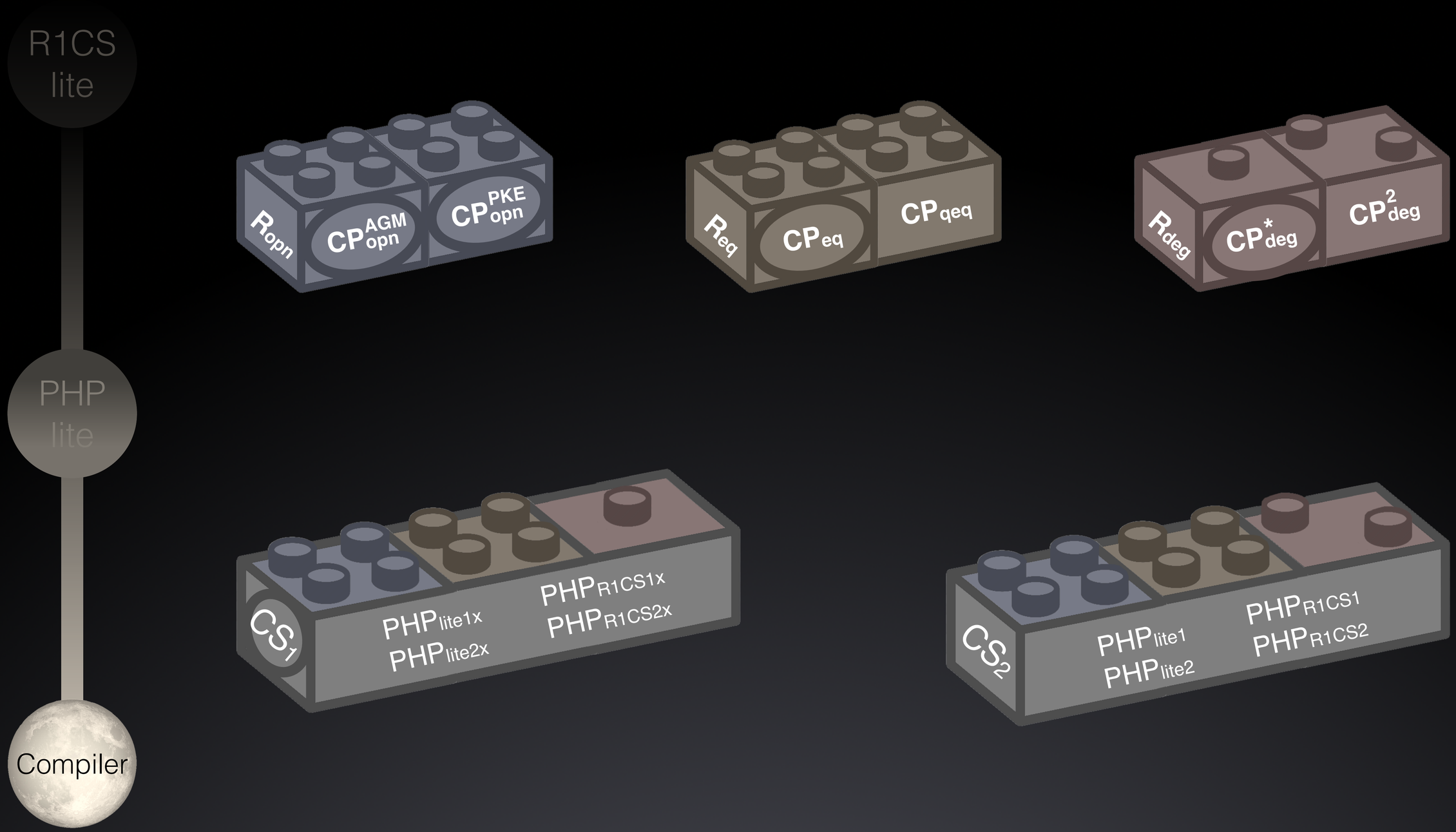
trivial empty proof Marlin, Plonk



eval random point + Plonk lin tricks



commit to shifted polynomial, batch


R1CS lite
PHP lite
Compiler
R_{opn}
CP^{AGM}_{opn}
CP^{PKE}_{opn}
R_{eq}
CP_{eq}
CP_{qeq}
R_{deg}
CP^{*}_{deg}
CP^{2}_{deg}
CS_1
PHP_{lite1x}
PHP_{lite2x}
PHP_{R1CS1x}
PHP_{R1CS2x}
CS_2
PHP_{lite1}
PHP_{lite2}
PHP_{R1CS1}
PHP_{R1CS2}