geometry

Scaling Ethereum with Lookup Arguments

JANI2023

Some Context

- The world of ZK proofs saw many revolutionary breakthroughs in 2022
- In particular, many new **lookup argument** techniques
- Today's agenda:
  - Motivation: from scaling blockchains to succinct proofs and the need for lookup arguments
  - An overview of the advances in lookup arguments

Part 1

Scaling blockchains with ZK proofs and the need for lookup arguments

ZK (and succinct) Proofs in Blockchains

- A *prover* can convince a *verifier* that they correctly ran a computation.
  - zero-knowledge: the verifier learns nothing about the computation
  - succinctness: the verifier does **much less** work than the original computation
- First used for blockchains by Zcash to introduce privacy
- Mostly used today for **scaling**: L2s such as Aztec, Scroll, StarkNet, Polygon Miden, zkSync

ZK Proofs in Practice

- Requires:
  - a way to hide numbers, but still apply operations on them
  - objects that don’t get too large
  - can’t break our group of objects into smaller groups

ZK Proofs in Practice

- Requires:
  - a way to hide numbers, but still apply operations on them
  - objects that don’t get too large
  - can’t break our group of objects into smaller groups

=> Introduce elliptic curves!

discrete logarithm problem hides the numbers, **finite field** ensures objects have a maximum size, **prime** number of elements so that it cannot be divided

Ready to Prove Ethereum?

- Classical computing is binary, working on 0s and 1s
- Ethereum was built without ZKPs in mind and follows the classical computing paradigm:
  - types are chosen to be stored as bits, e.g. uint256
  - use Keccak hash for the state tree

Ready to Prove Ethereum?

- Classical computing is binary, working on 0s and 1s
- Ethereum was built without ZKPs in mind and follows the classical computing paradigm:
  - types are chosen to be stored as bits, e.g. uint256
  - use Keccak hash for the state tree

How do we reconcile ZKP math with computer math?

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements

Bits	Field
00	0
01	1
10	2
11	3

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- (maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)

Bits	Field
00	0
01	1
10	2
11	3

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- (maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)

The XOR table in bits

A	B	A XOR B
00	00	00
00	01	01
00	10	10
00	11	11
01	00	01
01	01	00
01	10	11
01	11	10
10	00	10
10	01	11
10	10	00
10	11	01
11	00	11
11	01	10
11	10	01
11	11	00

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- (maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)

The XOR table in field elements

A	B	A XOR B
0	0	0
0	1	1
0	2	2
0	3	3
1	0	1
1	1	0
1	2	3
1	3	2
2	0	2
2	1	3
2	2	0
2	3	1
3	0	3
3	1	2
3	2	1
3	3	0

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- (maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)

The XOR table in field elements

A	B	A XOR B
0	0	0
0	1	1
0	2	2
0	3	3
1	0	1
1	1	0
1	2	3
1	3	2
2	0	2
2	1	3
2	2	0
2	3	1
3	0	3
3	1	2
3	2	1
3	3	0

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- (maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)

The XOR table in field elements

A	B	A XOR B
0	0	0
0	1	1
0	2	2
0	3	3
1	0	1
1	1	0
1	2	3
1	3	2
2	0	2
2	1	3
2	2	0
2	3	1
3	0	3
3	1	2
3	2	1
3	3	0

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- ~~(maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)~~
- Step 2: to prove that A XOR B = C, we **show that A, B, C is a row in the table**

The XOR table in field elements

A	B	A XOR B
0	0	0
0	1	1
0	2	2
0	3	3
1	0	1
1	1	0
1	2	3
1	3	2
2	0	2
2	1	3
2	2	0
2	3	1
3	0	3
3	1	2
3	2	1
3	3	0

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- ~~(maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)~~
- Step 2: to prove that A XOR B = C, we **show that A, B, C is a row in the table**

=> a lookup argument allows to prove that some element(s) exist as a row of the table

The XOR table in field elements

A	B	A XOR B
0	0	0
0	1	1
0	2	2
0	3	3
1	0	1
1	1	0
1	2	3
1	3	2
2	0	2
2	1	3
2	2	0
2	3	1
3	0	3
3	1	2
3	2	1
3	3	0

ZKP Math vs Computer Math

- Step 1: represent strings of bits as field elements
- ~~(maybe) Step 2: emulate logic operations (AND, OR, XOR, etc…) with our field operations (addition, multiplication, inversion)~~
- Step 2: to prove that A XOR B = C, we **show that A, B, C is a row in the table**

=> a lookup argument allows to prove that some element(s) exist as a row of the table

Faster lookups mean faster ZK proofs (and faster scaling!)

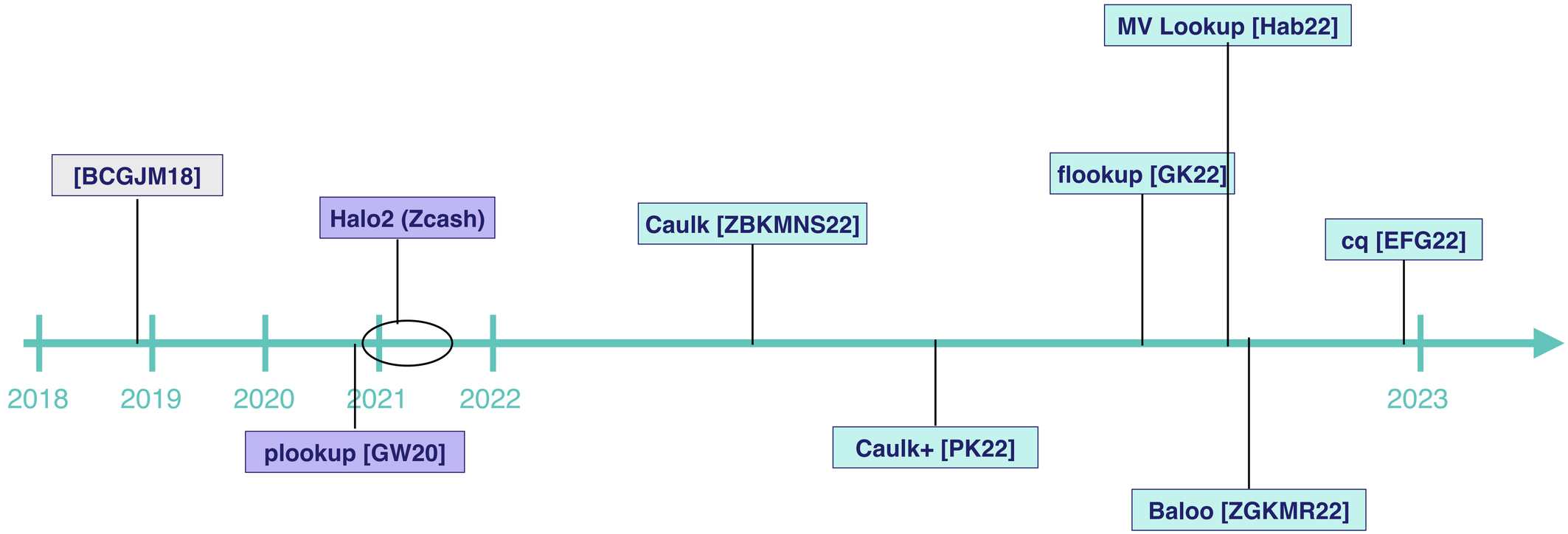
The XOR table in field elements

A	B	A XOR B
0	0	0
0	1	1
0	2	2
0	3	3
1	0	1
1	1	0
1	2	3
1	3	2
2	0	2
2	1	3
2	2	0
2	3	1
3	0	3
3	1	2
3	2	1
3	3	0

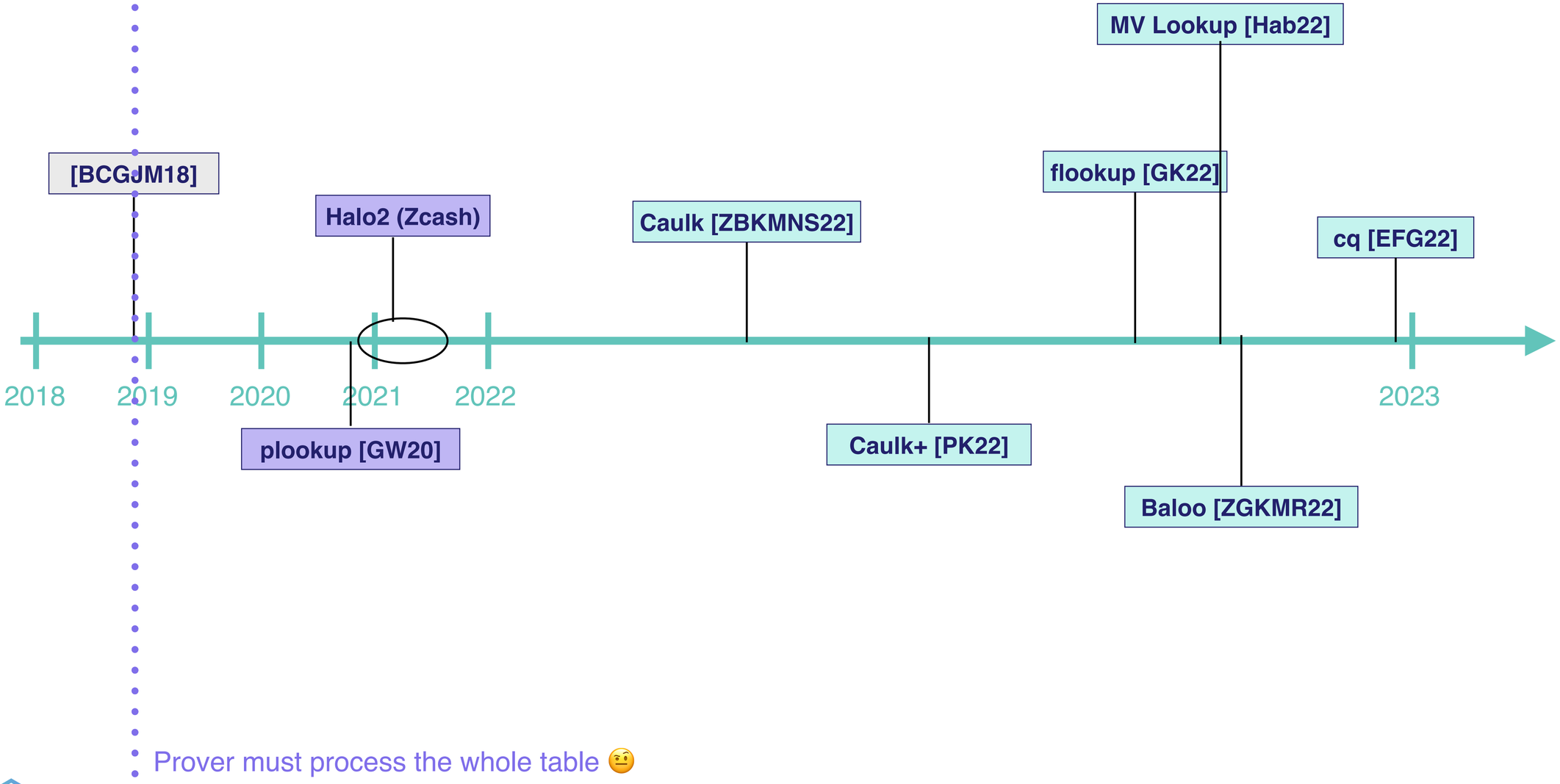
Part 2

2022, Year of the Lookup

A Timeline of Lookups

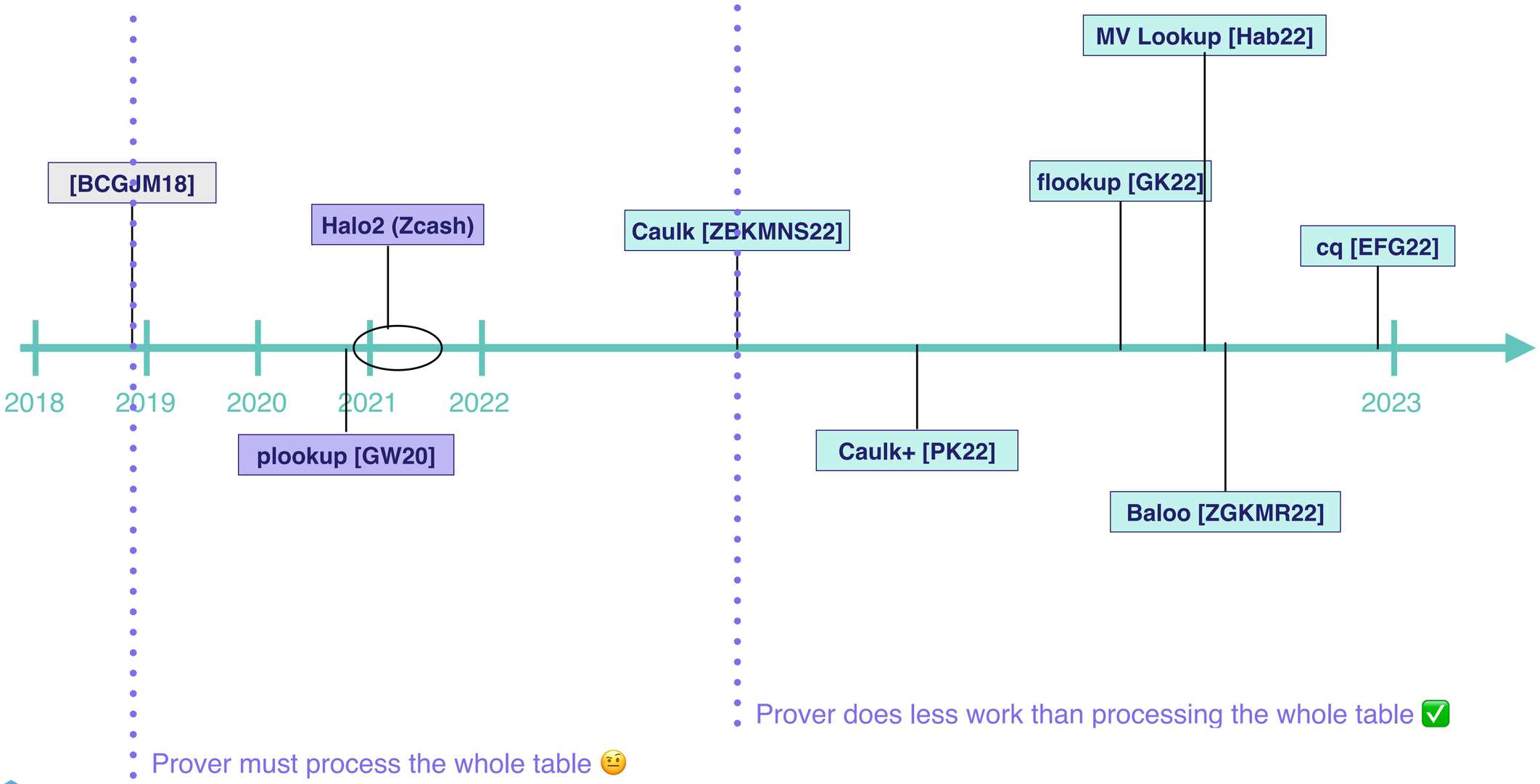


A Timeline of Lookups

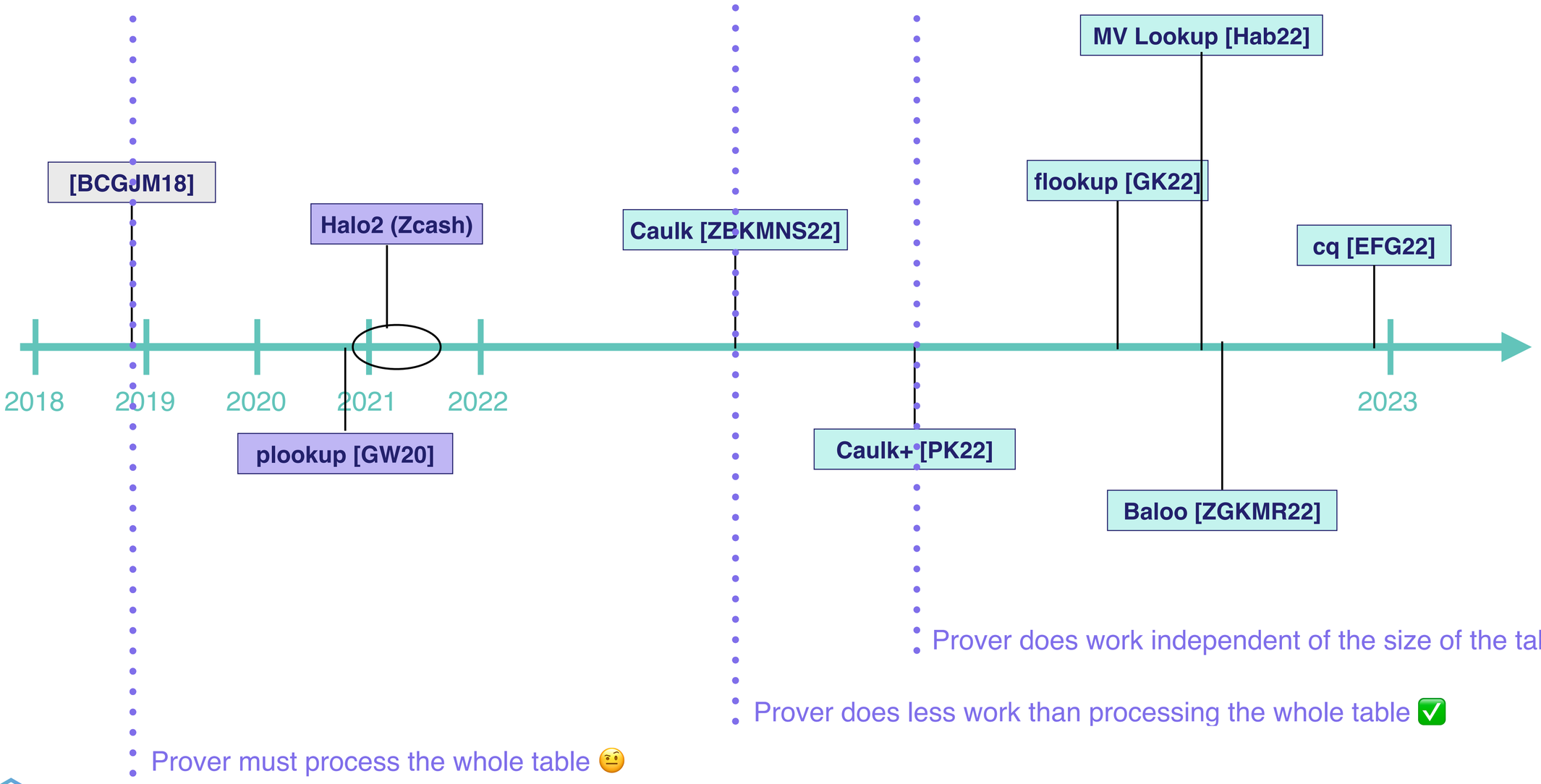


Prover must process the whole table 😬

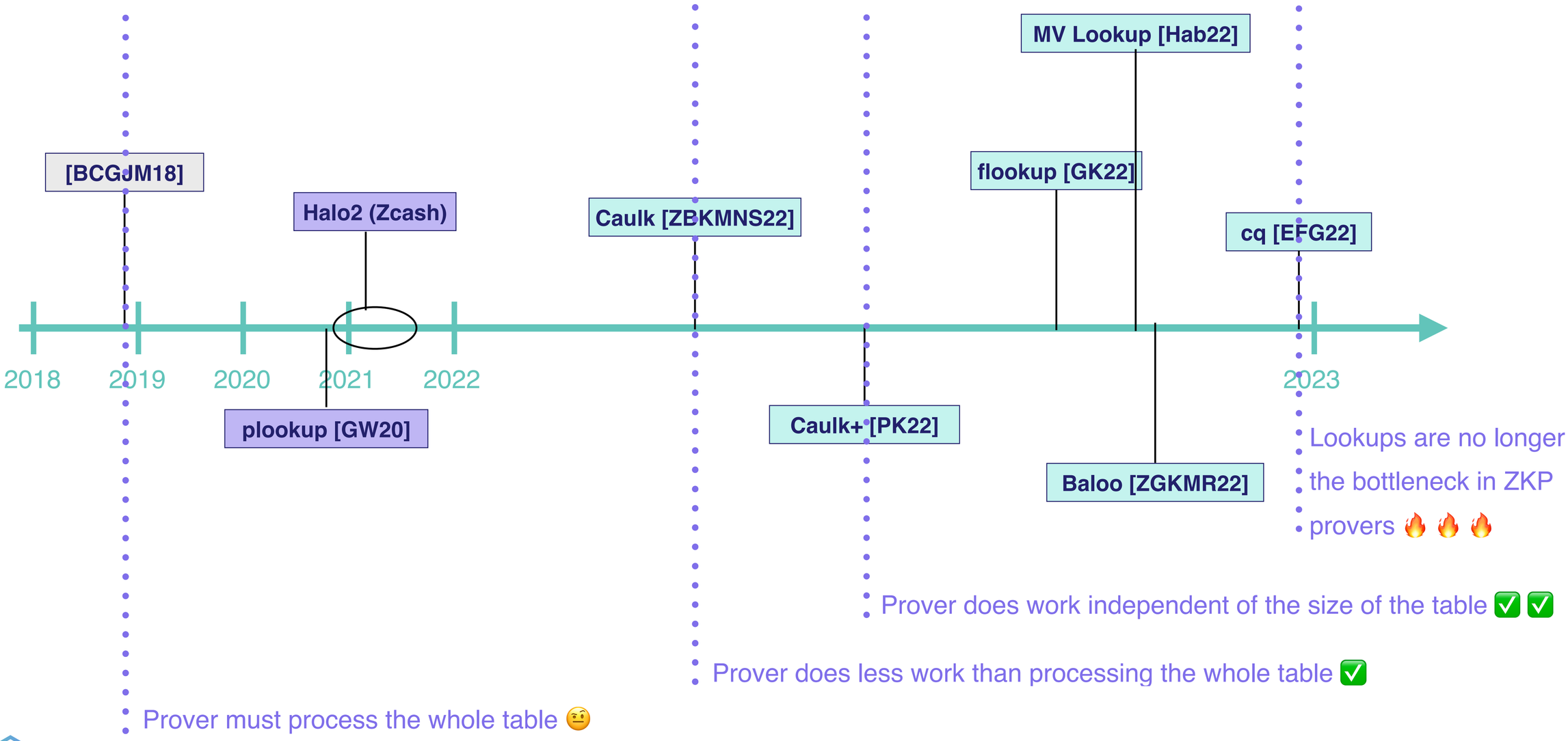
A Timeline of Lookups



A Timeline of Lookups



A Timeline of Lookups

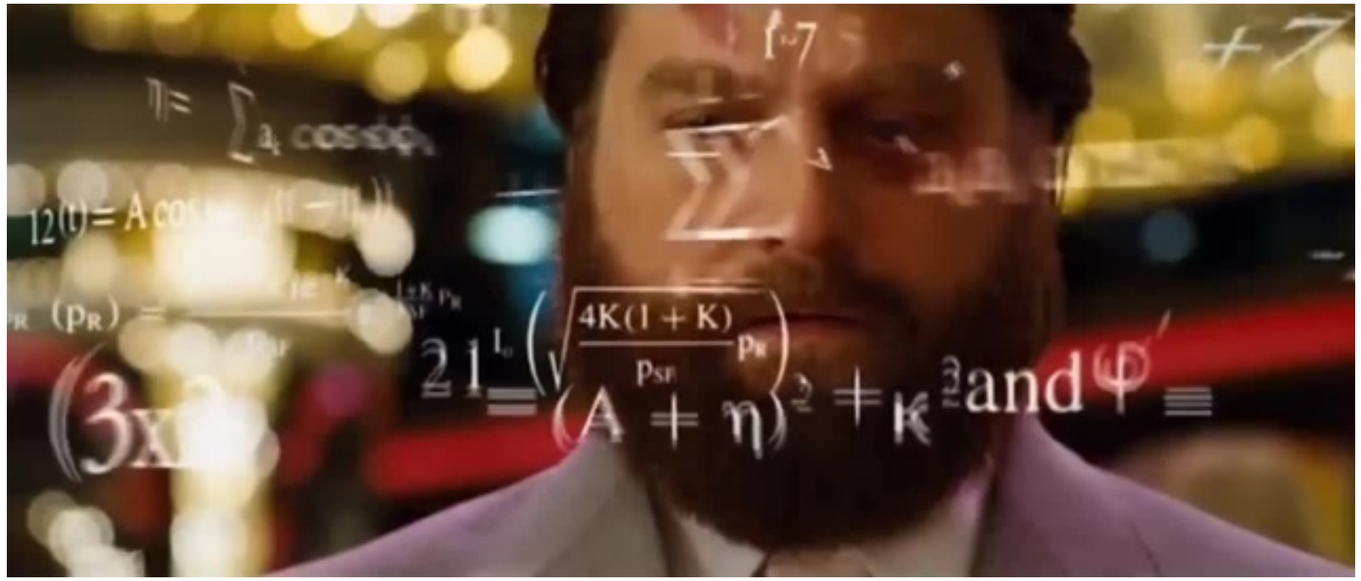


The Caulk Breakthrough (High Level)

Rather than encoding the whole table, Caulk derives a polynomial $t(X)$ which *only encodes the values that we are interested in*.

Proving that $t(X)$ is correctly formed can be done with less work than processing the whole table *if we are given the right precomputed values*.

The Caulk Breakthrough (Math Time)



The Caulk Breakthrough (Math Time)

Aside: the KZG polynomial commitment

To prove that $T(z_1) = y_1$ we need to show the existence of $q_1(X)$ such that:

$$q_1(X) = \frac{T(X) - y_1}{X - z_1}$$

The Caulk Breakthrough (Math Time)

Aside: the KZG polynomial commitment

To prove that $T(z_1) = y_1$ we need to show the existence of $q_1(X)$ such that:

$$q_1(X) = \frac{T(X) - y_1}{X - z_1}$$

This extends to multiple values! To show that $T(z_1) = y_1, T(z_2) = y_2, \ldots, T(z_n) = y_n$ we need

$$q(X) = \frac{T(X) - \sum_{i=1}^{n} y_i L_i(X)}{\prod_{i=1}^{n} (X - z_i)}$$

$L_i(X) = 1$ for $X = z_i$,
$= 0$ otherwise

The Caulk Breakthrough (Math Time)

Aside: the KZG polynomial commitment

To prove that $T(z_1) = y_1$ we need to show the existence of $q_1(X)$ such that:

$$q_1(X) = \frac{T(X) - y_1}{X - z_1}$$

This extends to multiple values! To show that $T(z_1) = y_1, T(z_2) = y_2, \ldots, T(z_n) = y_n$ we need

$$q(X) = \frac{T(X) - \sum_{i=1}^{n} y_i L_i(X)}{\prod_{i=1}^{n} (X - z_i)}$$

This can be all the values of our smaller polynomial $t(X)$

The Caulk Breakthrough (Math Time)

Aside: the KZG polynomial commitment

To prove that $T(z_1) = y_1$ we need to show the existence of $q_1(X)$ such that:

$$q_1(X) = \frac{T(X) - y_1}{X - z_1}$$

This extends to multiple values! To show that $T(z_1) = y_1, T(z_2) = y_2, \ldots, T(z_n) = y_n$ we need

$$q(X) = \frac{T(X) - \sum_{i=1}^{n} y_i L_i(X)}{\prod_{i=1}^{n} (X - z_i)}$$

This can be all the values of our smaller polynomial $t(X)$

Problem: working with $T(X)$ is still the size of the big table…

The Caulk Breakthrough (Math Time)

This extends to multiple values! To show that $T(z_1) = y_1, T(z_2) = y_2, \ldots, T(z_n) = y_n$ we need

$$q(X) = \frac{T(X) - \sum_{i=1}^{n} y_i L_i(X)}{\prod_{i=1}^{n} (X - z_i)}$$

Problem: working with $T(X)$ is still the size of the big table…

The Caulk Breakthrough (Math Time)

This extends to multiple values! To show that $T(z_1) = y_1, T(z_2) = y_2, \ldots, T(z_n) = y_n$ we need

$$q(X) = \frac{T(X) - \sum_{i=1}^{n} y_i L_i(X)}{\prod_{i=1}^{n} (X - z_i)}$$

Problem: working with $T(X)$ is still the size of the big table…

Solution: pre-computation!

If we have commitments $Q_1, Q_2, \ldots, Q_n$ that respectively show that $T(z_1) = y_1, T(z_2) = y_2, \ldots, T(z_n) = y_n$, we can compute the commitment Q to $q(X)$ as above via linear combinations. No dependence on the size of the big table only on n

Recap

- We use ZKPs for scaling blockchains, but some computations are hard to represent inside a ZKP
- Lookup arguments allow to perform the computation outside of the ZKP yet prove that the output is correctly derived from the input
- 2022 saw rapid successive breakthroughs is lookup techniques; lookups are no longer the bottleneck of a SNARK prover!
- 2023, year of the “fast lookup”-enabled zkEVM?

hello@geometry.xyz

@__geometry__

LONDON ST. HELIER SINGAPORE BELGRADE TEL AVIV BOSTON