STARKWARE

A Cambrian Explosion of Cryptographic Proofs

Eli Ben-Sasson

February 2023

Background

Background

Integrity* via Math

1992

Checking Computations in Polylogarithmic Time

ászló Babai[1]
Chicago[6] and
iv., Budapest

Lance Fortnow[2]
Dept. Comp. Sci.
Univ. of Chicago[6]

Leonid A. Levin[3]
Dept. Comp. Sci.
Boston University[4]

Mario Szegedy[5]
Dept. Comp. Sci.
Univ. of Chicago

Abstract. Motivated by Manuel Blum's concept of *instance checking*, we consider new, very fast and generic mechanisms of checking computations. Our results exploit recent advances in interactive proof protocols [LFKN92], [Sha92], and especially the $MIP = NEXP$ protocol from [BFL91].

We show that every nondeterministic computational task $S(x, y)$, defined as a polynomial time relation between the *instance* x, representing the input and output combined, and the *witness* y can be modified to a task S' such that: (i) the same instances remain accepted; (ii) each instance/witness pair becomes checkable in *polylogarithmic* Monte Carlo time; and (iii) a witness satisfying S' can be computed in polynomial time from a witness satisfying S.

Here the instance and the description of S have to be provided in error-correcting code (since the checker will not notice slight changes). A modification of the MIP proof was required to achieve polynomial time in (iii); the earlier technique yields $N^{O(\log\log N)}$ time only.

This result becomes significant if software and hardware *reliability* are regarded as a considerable cost factor. The polylogarithmic checker is the only part of the system that needs to be trusted; it can be *hard wired.* (We use just *one Checker* for all problems!) The checker is tiny and so presumably can be optimized and checked off-line at a modest cost.

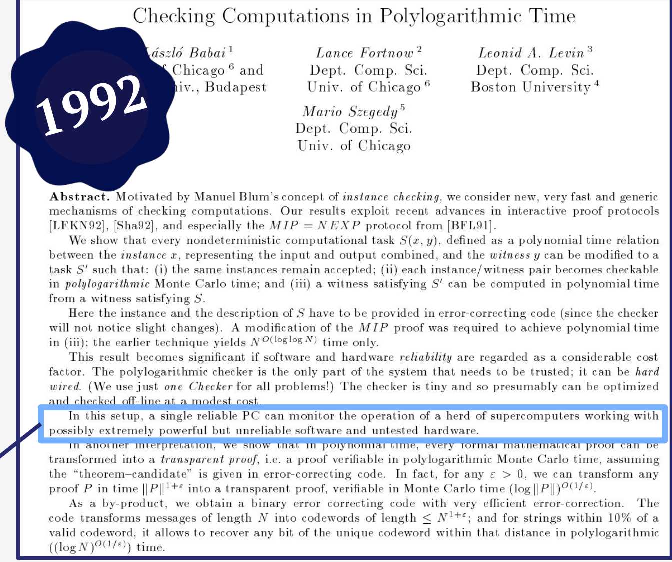
In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

In another interpretation, we show that in polynomial time, every formal mathematical proof can be transformed into a *transparent proof*, i.e. a proof verifiable in polylogarithmic Monte Carlo time, assuming the "theorem–candidate" is given in error-correcting code. In fact, for any $\varepsilon > 0$, we can transform any proof P in time $\|P\|^{1+\varepsilon}$ into a transparent proof, verifiable in Monte Carlo time $(\log \|P\|)^{O(1/\varepsilon)}$.

As a by-product, we obtain a binary error correcting code with very efficient error-correction. The code transforms messages of length N into codewords of length $\leq N^{1+\varepsilon}$; and for strings within 10% of a valid codeword, it allows to recover any bit of the unique codeword within that distance in polylogarithmic $((\log N)^{O(1/\varepsilon)})$ time.

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background


1992

Checking Computations in Polylogarithmic Time

ászló Babai[1]
Chicago[6] and
iv., Budapest

Lance Fortnow[2]
Dept. Comp. Sci.
Univ. of Chicago[6]

Leonid A. Levin[3]
Dept. Comp. Sci.
Boston University[4]

Mario Szegedy[5]
Dept. Comp. Sci.
Univ. of Chicago

Abstract. Motivated by Manuel Blum's concept of *instance checking*, we consider new, very fast and generic mechanisms of checking computations. Our results exploit recent advances in interactive proof protocols [LFKN92], [Sha92], and especially the $MIP = NEXP$ protocol from [BFL91].

We show that every nondeterministic computational task $S(x, y)$, defined as a polynomial time relation between the *instance* x, representing the input and output combined, and the *witness* y can be modified to a task S' such that: (i) the same instances remain accepted; (ii) each instance/witness pair becomes checkable in *polylogarithmic* Monte Carlo time; and (iii) a witness satisfying S' can be computed in polynomial time from a witness satisfying S.

Here the instance and the description of S have to be provided in error-correcting code (since the checker will not notice slight changes). A modification of the MIP proof was required to achieve polynomial time in (iii); the earlier technique yields $N^{O(\log\log N)}$ time only.

This result becomes significant if software and hardware *reliability* are regarded as a considerable cost factor. The polylogarithmic checker is the only part of the system that needs to be trusted; it can be *hard wired*. (We use just *one Checker* for all problems!) The checker is tiny and so presumably can be optimized and checked off-line at a modest cost.

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

In another interpretation, we show that in polynomial time, every formal mathematical proof can be transformed into a *transparent proof*, i.e. a proof verifiable in polylogarithmic Monte Carlo time, assuming the "theorem–candidate" is given in error-correcting code. In fact, for any $\varepsilon > 0$, we can transform any proof P in time $\|P\|^{1+\varepsilon}$ into a transparent proof, verifiable in Monte Carlo time $(\log\|P\|)^{O(1/\varepsilon)}$.

As a by-product, we obtain a binary error correcting code with very efficient error-correction. The code transforms messages of length N into codewords of length $\leq N^{1+\varepsilon}$; and for strings within 10% of a valid codeword, it allows to recover any bit of the unique codeword within that distance in polylogarithmic $((\log N)^{O(1/\varepsilon)})$ time.

Integrity* via Math

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

STARKWARE

Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P,** reached state hash **y***

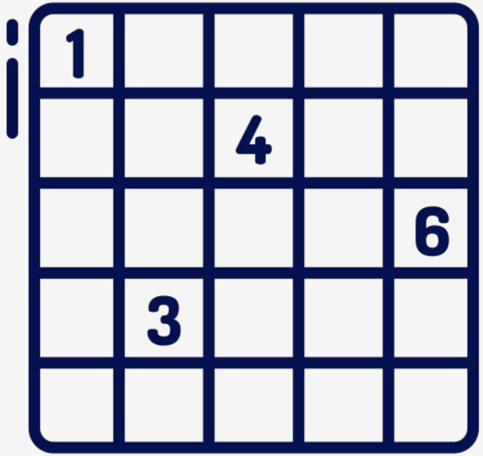
"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Background

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**



PCP

Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P**, reached state hash **y***

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

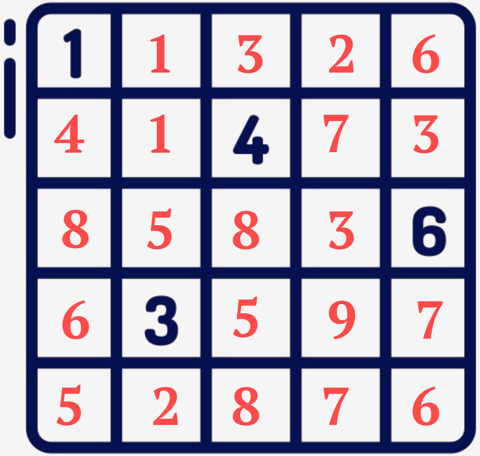
* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

STARKWARE

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**

Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P,*** *reached state hash* ***y***



PCP

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

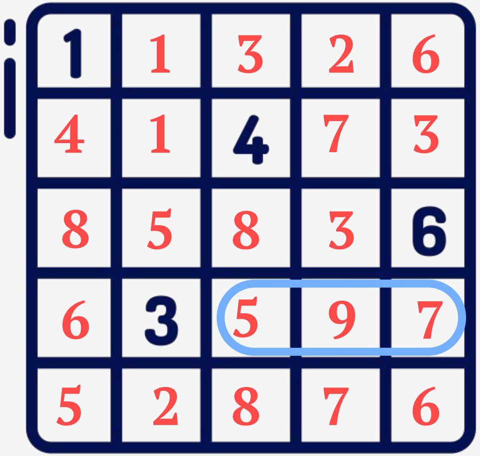
Prover submits solution

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P**, reached state hash **y***

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P***, and **#tx (=1,000,000)**



PCP

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

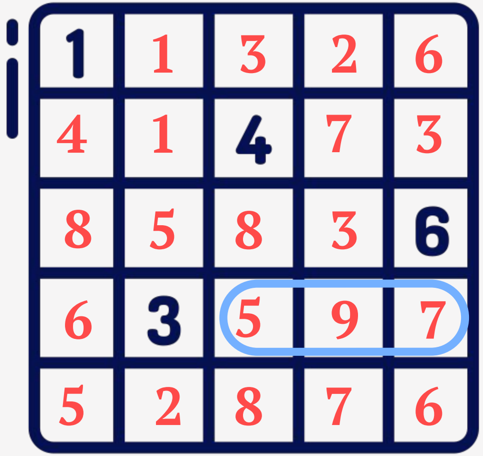
Prover submits solution

Verifier samples and checks a single constraint

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**



PCP

Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P,*** *reached state hash* ***y***

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A "proof" of a false claim satisfies < 1% of constraints!

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

Prover submits solution

Verifier samples and checks a single constraint

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

STARKWARE

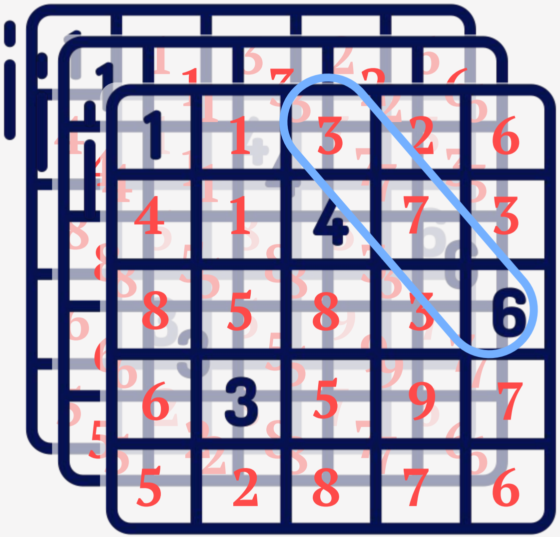
Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P****, reached state hash* ***y***

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A "proof" of a false claim satisfies < 1% of constraints!

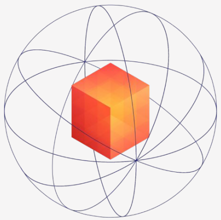
"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**



STARK



Prover submits solution

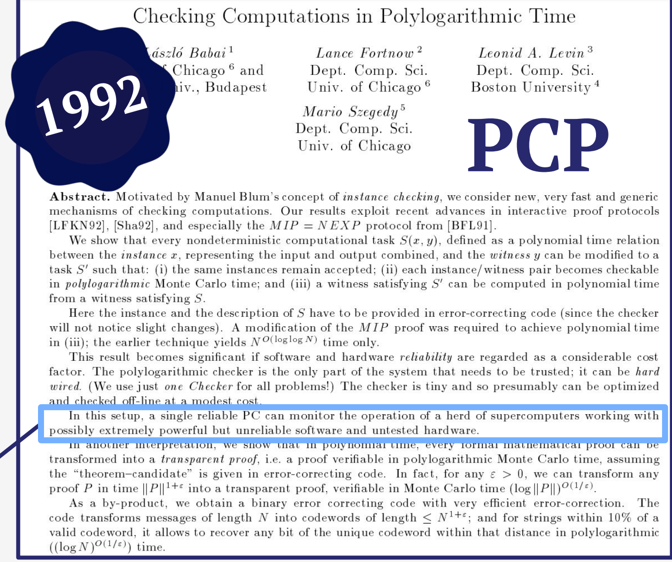
Verifier posts another (random) sudoku puzzle

Verifier samples and checks a single constraint

Background

STARK

Integrity* via Math (impractical)


Checking Computations in Polylogarithmic Time

ászló Babai [1]
f Chicago [6] and
niv., Budapest

Lance Fortnow [2]
Dept. Comp. Sci.
Univ. of Chicago [6]

Leonid A. Levin [3]
Dept. Comp. Sci.
Boston University [4]

Mario Szegedy [5]
Dept. Comp. Sci.
Univ. of Chicago

Abstract. Motivated by Manuel Blum's concept of *instance checking*, we consider new, very fast and generic mechanisms of checking computations. Our results exploit recent advances in interactive proof protocols [LFKN92], [Sha92], and especially the $MIP = NEXP$ protocol from [BFL91].

We show that every nondeterministic computational task $S(x, y)$, defined as a polynomial time relation between the *instance* x, representing the input and output combined, and the *witness* y can be modified to a task S' such that: (i) the same instances remain accepted; (ii) each instance/witness pair becomes checkable in *polylogarithmic* Monte Carlo time; and (iii) a witness satisfying S' can be computed in polynomial time from a witness satisfying S.

Here the instance and the description of S have to be provided in error-correcting code (since the checker will not notice slight changes). A modification of the MIP proof was required to achieve polynomial time in (iii); the earlier technique yields $N^{O(\log\log N)}$ time only.

This result becomes significant if software and hardware *reliability* are regarded as a considerable cost factor. The polylogarithmic checker is the only part of the system that needs to be trusted; it can be *hard wired.* (We use just *one Checker* for all problems!) The checker is tiny and so presumably can be optimized and checked off-line at a modest cost.

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

In another interpretation, we show that in polynomial time, every formal mathematical proof can be transformed into a *transparent proof*, i.e. a proof verifiable in polylogarithmic Monte Carlo time, assuming the "theorem–candidate" is given in error-correcting code. In fact, for any $\varepsilon > 0$, we can transform any proof P in time $\|P\|^{1+\varepsilon}$ into a transparent proof, verifiable in Monte Carlo time $(\log \|P\|)^{O(1/\varepsilon)}$.

As a by-product, we obtain a binary error correcting code with very efficient error-correction. The code transforms messages of length N into codewords of length $\leq N^{1+\varepsilon}$; and for strings within 10% of a valid codeword, it allows to recover any bit of the unique codeword within that distance in polylogarithmic $((\log N)^{O(1/\varepsilon)})$ time.

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

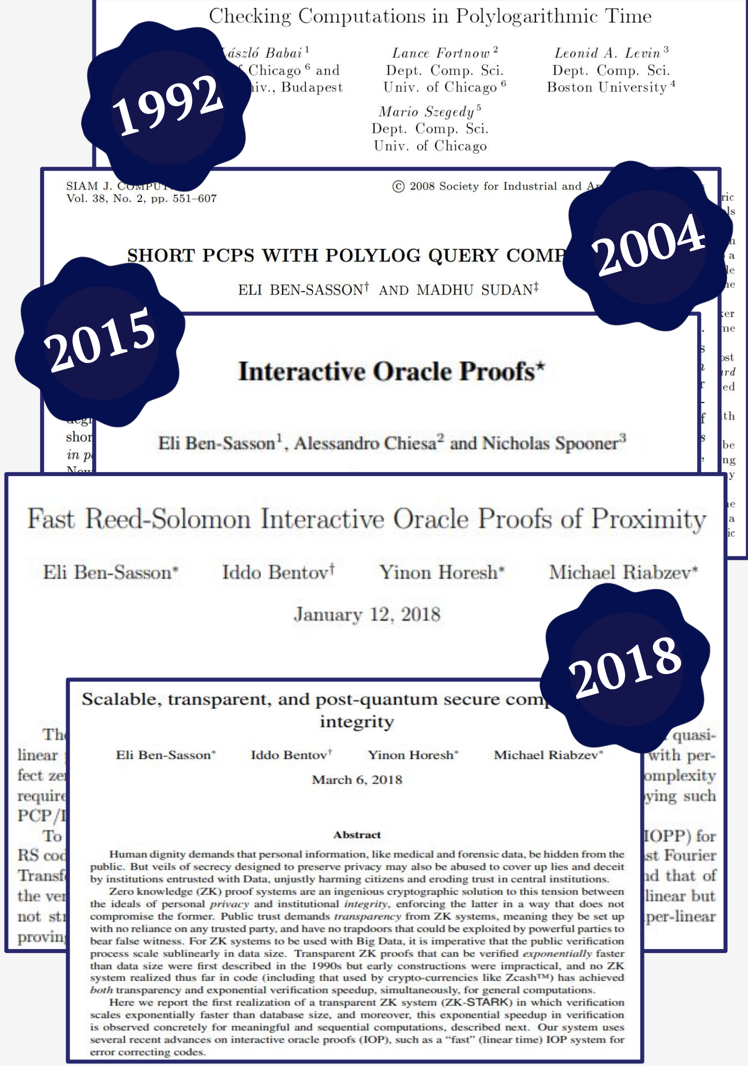
Background

STARK

Integrity* via Math
(~~im~~practical)

> *"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."*

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]



Background

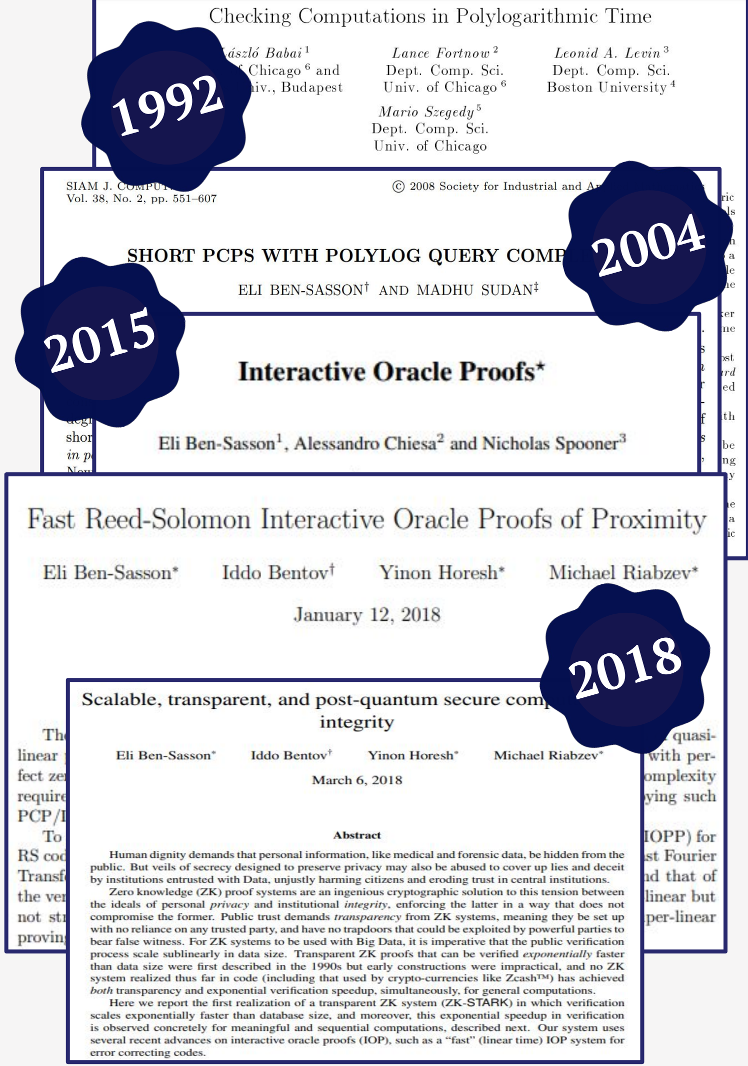
STARK

Integrity* via Math

(~~im~~practical)

> *"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."*

** Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]



STARKWARE

Background

Integrity* via Math

“…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …”

Bitcoin: A Peer-to-Peer Electronic Cash System

2008

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

Integrity* via Math

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

** Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Bitcoin: A Peer-to-Peer Electronic Cash System

2008

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.
By Vitalik Buterin (2014).

2014

When Satoshi Nakamoto first set the Bitcoin blockchain into motion in January was simultaneously introducing two radical and untested concepts. The first is the "bitcoin", a decentralized peer-to-peer online currency that maintains a value without any backing, intrinsic value or central issuer. So far, the "bitcoin" as a currency unit has taken up the bulk of the public attention, both in terms of the political aspects of a currency without a central bank and its extreme upward and downward volatility in price. However, there is also another, equally important, part to Satoshi's grand experiment: the concept of a proof of work-based blockchain to allow for public agreement on the order of transactions. Bitcoin as an application can be described as a first-to-file system: if one entity has 50 BTC, and simultaneously sends the same 50 BTC to A and to B, only the transaction that gets confirmed first will process. There is no intrinsic way of determining from two transactions which came earlier, and for decades this stymied the development of decentralized digital currency. Satoshi's blockchain was the first credible decentralized solution. And now, attention is rapidly starting to shift toward this second part of Bitcoin's technology, and how the blockchain concept can be used for more than just money.

Background

Integrity* via Math

> *"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"*

Blockchain is the "single reliable PC"

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

2013

Integrity* via Math

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

Blockchain is the "single reliable PC"

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Background

2013

Integrity* via Math

Blockchain is the "single reliable PC"

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

2015 - Zcash 1st general ZK for privacy

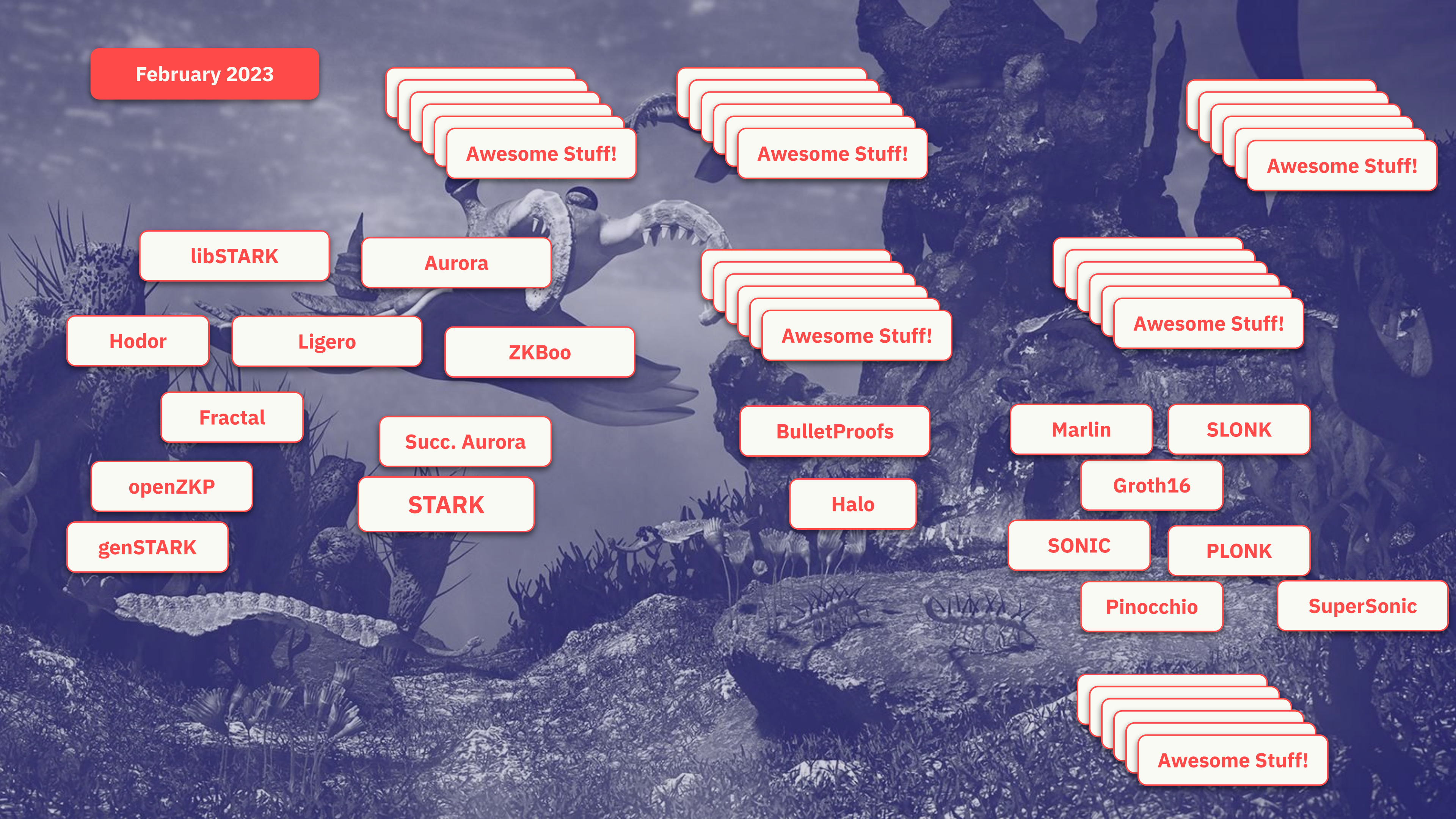
2018 - StarkWare 1st Proofs for scalability

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

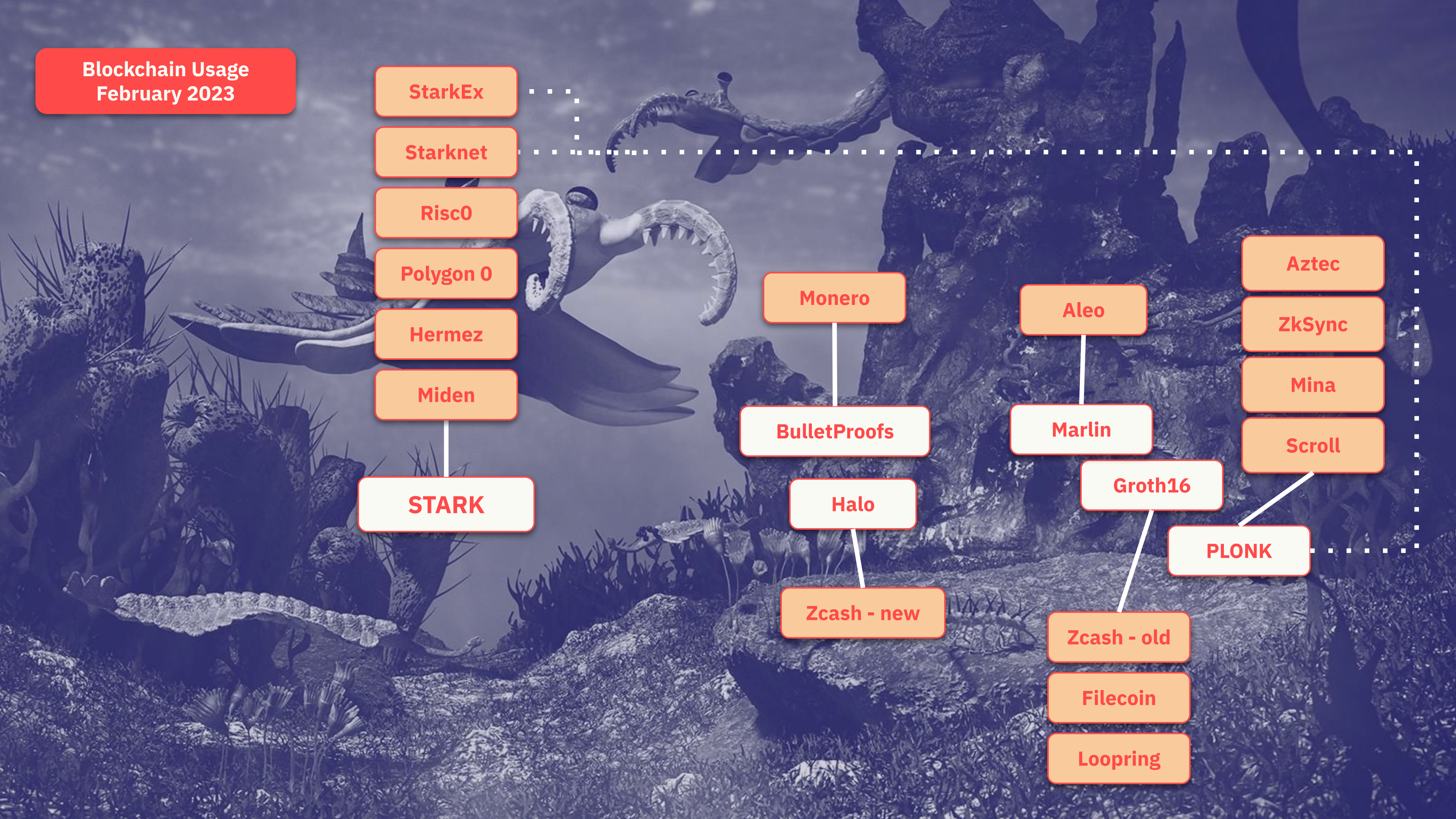
Cambrian Explosion of Cryptographic Proofs


September 2019
libSTARK
Aurora
Ligero
ZKBoo
BulletProofs
STARK
Halo
Groth16
genSTARK
SONIC
PLONK
Pinocchio

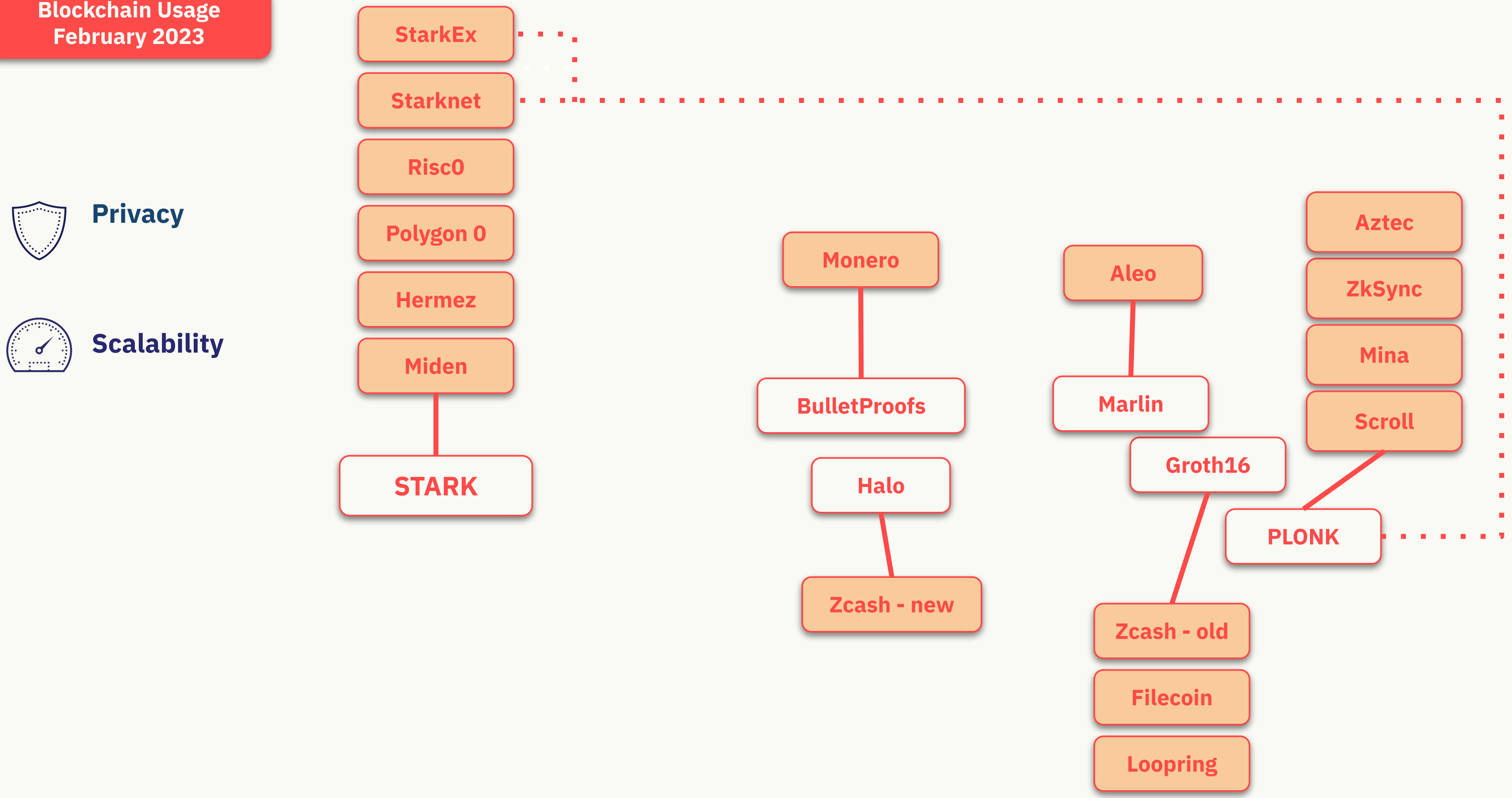

November 2019
libSTARK
Aurora
Hodor
Ligero
ZKBoo
Fractal
Succ. Aurora
BulletProofs
Marlin
SLONK
openZKP
STARK
Halo
Groth16
genSTARK
SONIC
PLONK
Pinocchio
SuperSonic


February 2023
Awesome Stuff!
Awesome Stuff!
Awesome Stuff!
libSTARK
Aurora
Hodor
Ligero
ZKBoo
Awesome Stuff!
Awesome Stuff!
Fractal
Succ. Aurora
openZKP
STARK
genSTARK
BulletProofs
Halo
Marlin
SLONK
Groth16
SONIC
PLONK
Pinocchio
SuperSonic
Awesome Stuff!

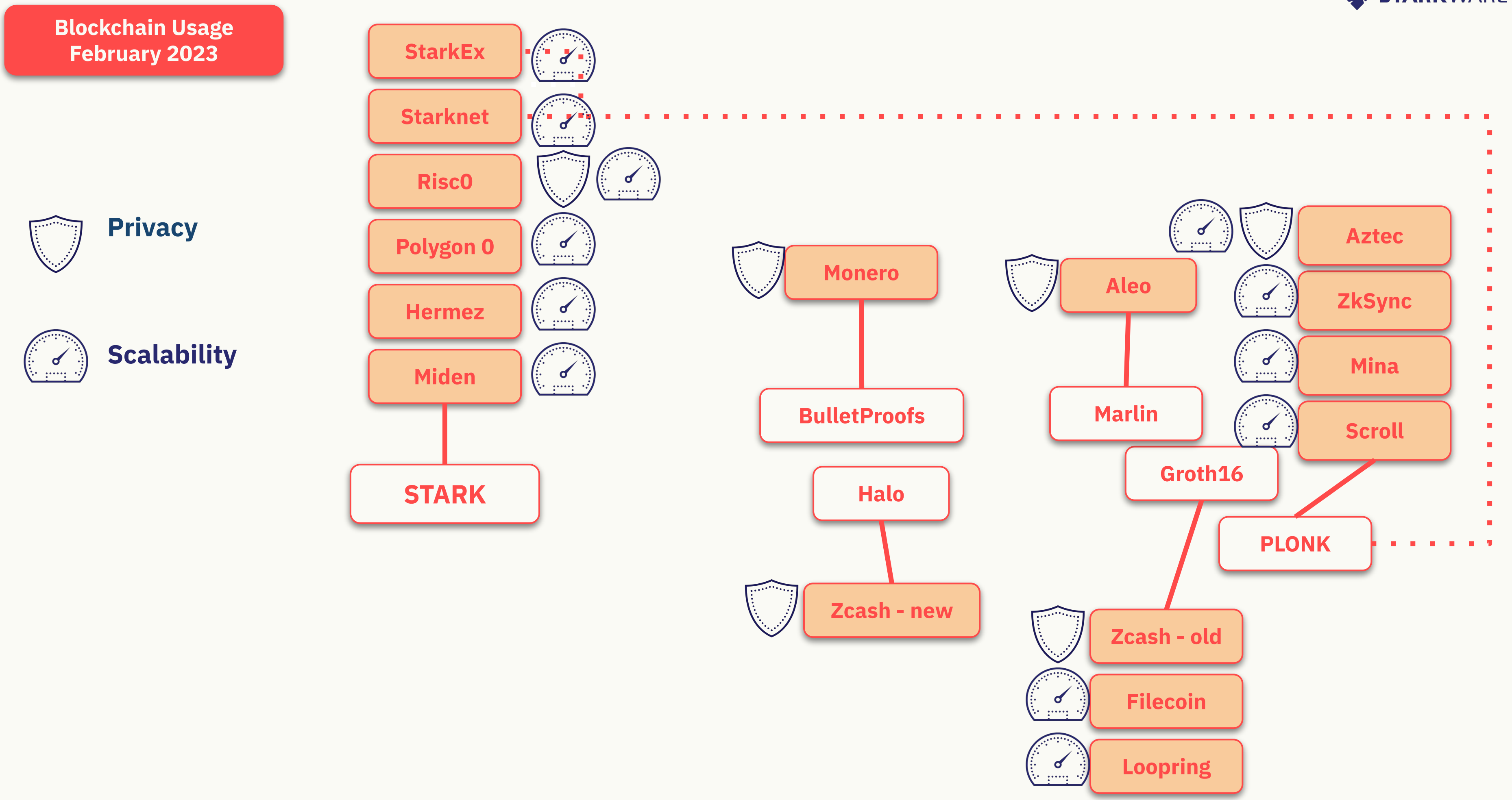
Blockchain Usage February 2023



Blockchain Usage February 2023



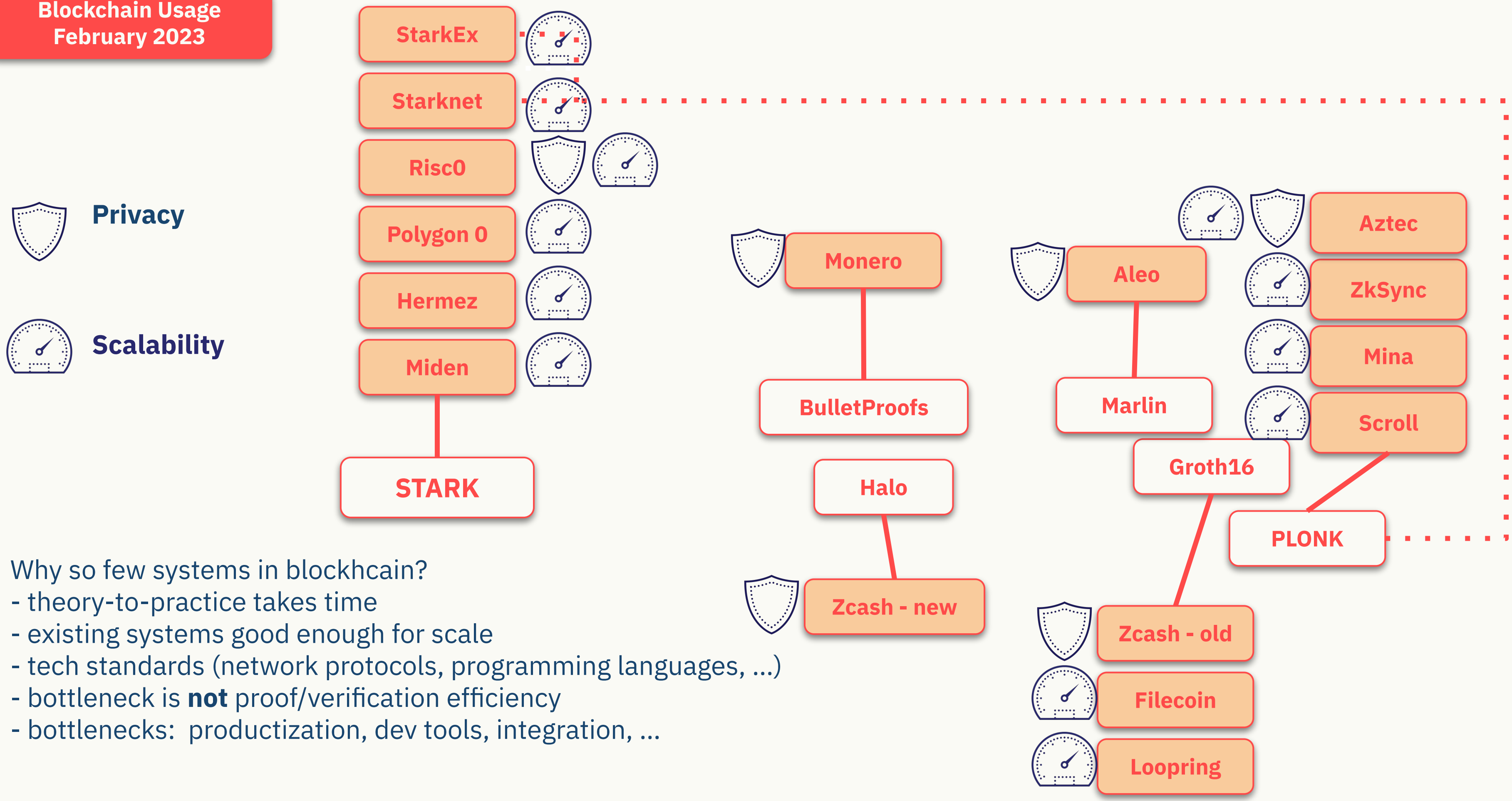
STARKWARE



Blockchain Usage
February 2023
Privacy
Scalability
StarkEx
Starknet
Risc0
Polygon 0
Hermez
Miden
STARK
Monero
BulletProofs
Halo
Zcash - new
Aleo
Marlin
Groth16
Zcash - old
Filecoin
Loopring
Aztec
ZkSync
Mina
Scroll
PLONK
STARKWARE

STARKWARE

Blockchain Usage February 2023



Why so few systems in blockhcain?
- theory-to-practice takes time
- existing systems good enough for scale
- tech standards (network protocols, programming languages, ...)
- bottleneck is **not** proof/verification efficiency
- bottlenecks: productization, dev tools, integration, ...

STARKWARE

Proofs of Computational Integrity (CI)

Privacy (Zero Knowledge, ZK)
Prover’s private inputs are shielded

Scalability
Exponentially small verifier running time*
Nearly linear prover running time*

Universality
Applicability to general computation

Transparency
No toxic waste (i.e. no trusted setup)

*With respect to size of computation

Common Ancestors

1. Arithmetization
2. Low degreeness

1) Arithmetization

Arithmetization Converts (“reduces”) Computational Integrity problems to problems about local relations between a bunch of polynomials

Example: For public 256-bit string z, Bob claims knows a SHA2-preimage of z

1) Arithmetization

Arithmetization Converts ("reduces") Computational Integrity problems to problems about local relations between a bunch of polynomials

Example: For public 256-bit string ***z***, Bob claims knows a SHA2-preimage of ***z***

Pre-arithmetization claim

"I know y such that SHA2(y)=z"

1) Arithmetization

Arithmetization Converts ("reduces") Computational Integrity problems to problems about local relations between a bunch of polynomials

Example: For public 256-bit string ***z***, Bob claims knows a SHA2-preimage of ***z***

Pre-arithmetization claim

"I know y such that SHA2(y)=z"

Reduction

produces 2 polynomials: ***Q(X,Y,T,W), R(X)*** *and degree bound* ***d***

1) Arithmetization

Arithmetization Converts ("reduces") Computational Integrity problems to problems about local relations between a bunch of polynomials

Example: For public 256-bit string ***z***, Bob claims knows a SHA2-preimage of ***z***

Pre-arithmetization claim

"I know y such that SHA2(y)=z"

Reduction

produces 2 polynomials: ***Q(X,Y,T,W), R(X)*** *and degree bound* ***d***

Post-arithmetization claim

I know 4 polynomials of degree ***d*** *- A(x), B(x), C(x), D(X) - such that:*

*Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)*

1) Arithmetization

Arithmetization Converts ("reduces") Computational Integrity problems to problems about local relations between a bunch of polynomials

Example: For public 256-bit string ***z***, Bob claims knows a SHA2-preimage of ***z***

Pre-arithmetization claim

"I know y such that SHA2(y)=z"

Reduction

produces 2 polynomials: ***Q(X,Y,T,W), R(X)*** *and degree bound* ***d***

Post-arithmetization claim

I know 4 polynomials of degree ***d*** *- A(x), B(x), C(x), D(X) - such that:*

*Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)*

Theorem

If A, B, C, D do not satisfy THIS,

then nearly all x expose Bob's lie

1) Arithmetization

Assuming Theorem, we get a scalable proof system for Bob's original claim:

1. Apply reduction, ask Bob to provide access to A,B,C,D of degree-d
2. Sample random x and accept Bob's claim iff equality holds for this x

Pre-arithmetization claim

"I know y such that SHA2(y)=z"

Reduction

produces 2 polynomials: ***Q(X,Y,T,W), R(X)*** *and degree bound* ***d***

Post-arithmetization claim

I know 4 polynomials of degree d - A(x), B(x), C(x), D(X) - such that:

*Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)*

Theorem

If A, B, C, D do not satisfy THIS,

then nearly all x expose Bob's lie

2) Low degreeness

Assuming Theorem, we get a scalable proof system for Bob's original claim:

1. Apply reduction, ask Bob to provide access to A,B,C,D of degree-d
2. Sample random x and accept Bob's claim iff equality holds for this x

New Computational Integrity problem: Force Bob to answer all queries according to some quadruple of degree-d polynomials

Post-arithmetization claim

I know 4 polynomials of degree d - A(x), B(x), C(x), D(X) - such that:

*Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)*

Theorem

If A, B, C, D do not satisfy THIS,

then nearly all x expose Bob's lie

Common Ancestors

1. Arithmetization
2. Low degreeness

Differentiating Factors

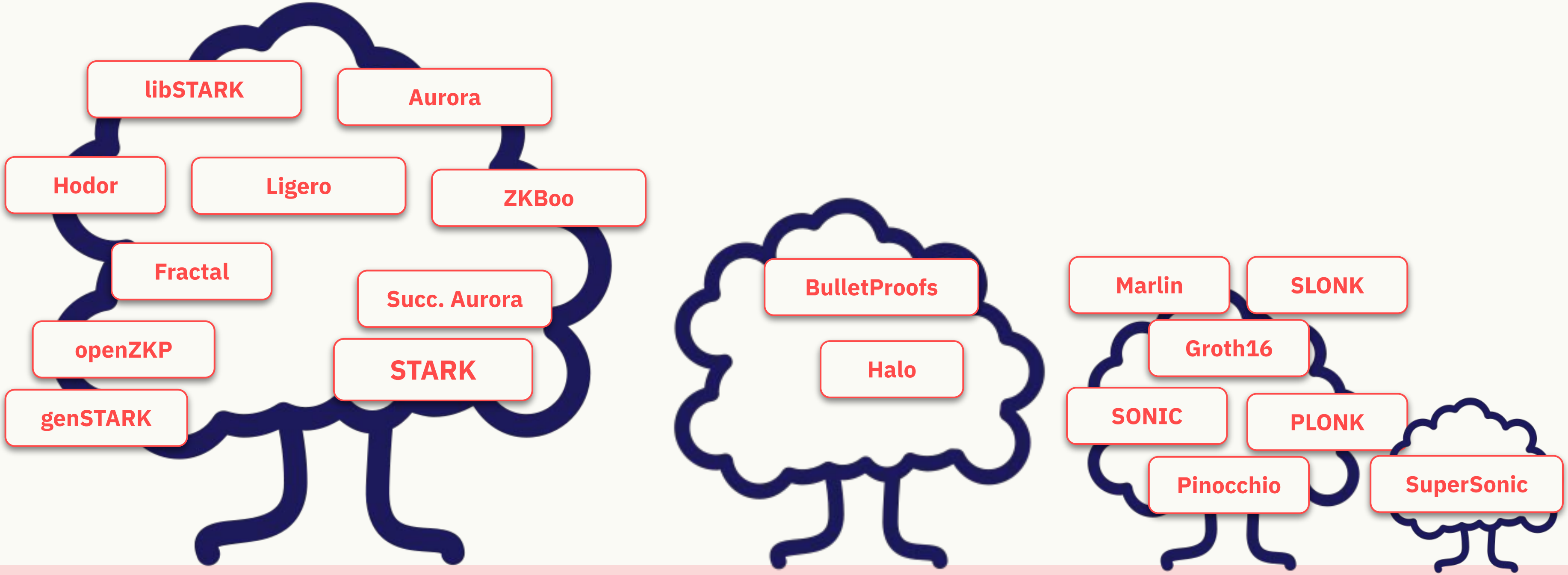
1. Arithmetization Method
2. Low degreeness enforcement
3. Cryptographic assumptions used to get 2

libSTARK
Aurora
Hodor
Ligero
ZKBoo
Fractal
Succ. Aurora
openZKP
STARK
genSTARK
BulletProofs
Halo
Marlin
SLONK
Groth16
SONIC
PLONK
Pinocchio
SuperSonic

Common Ancestors

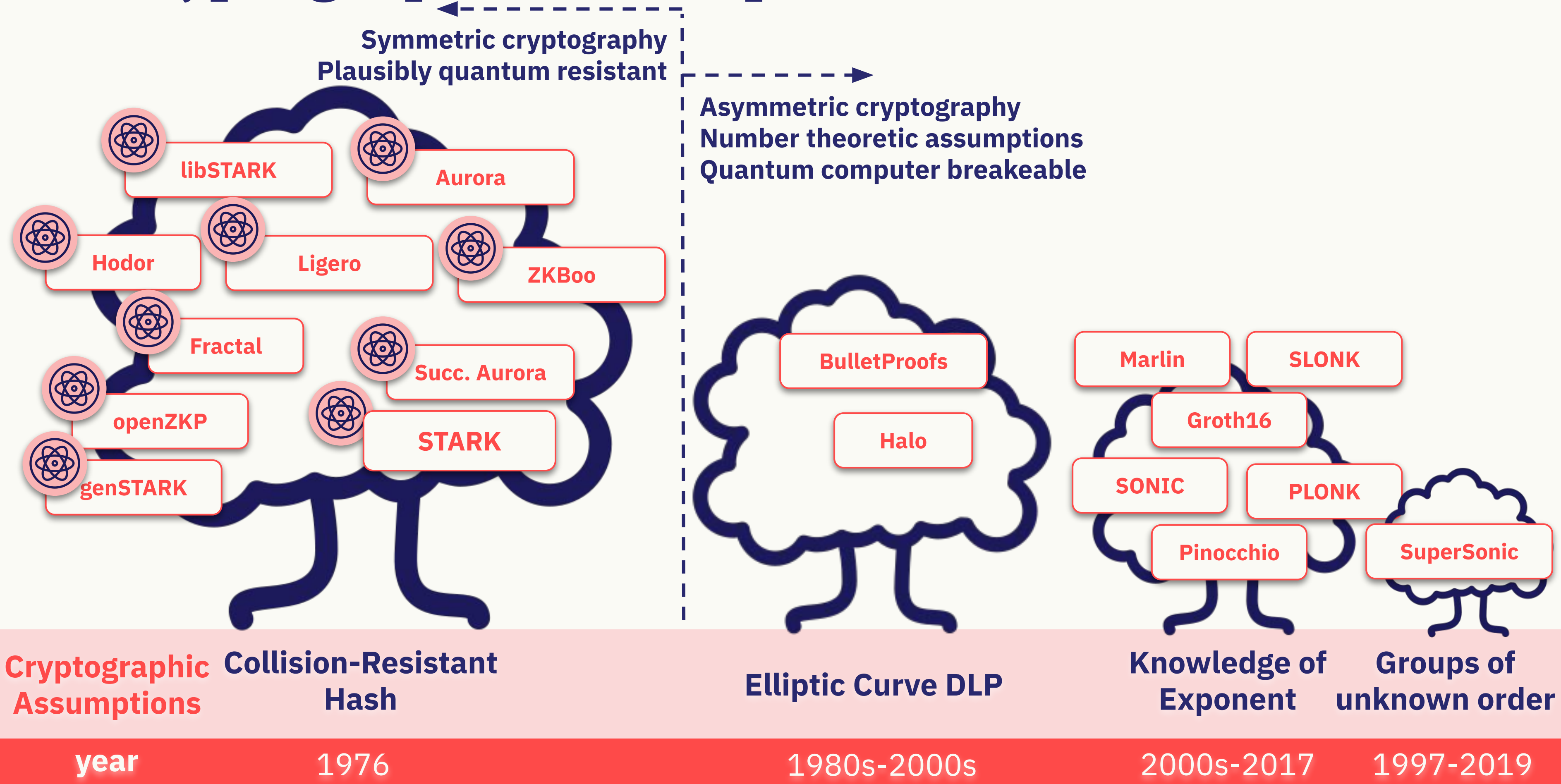
1. Arithmetization
2. Low degreeness

3. Cryptographic Assumptions

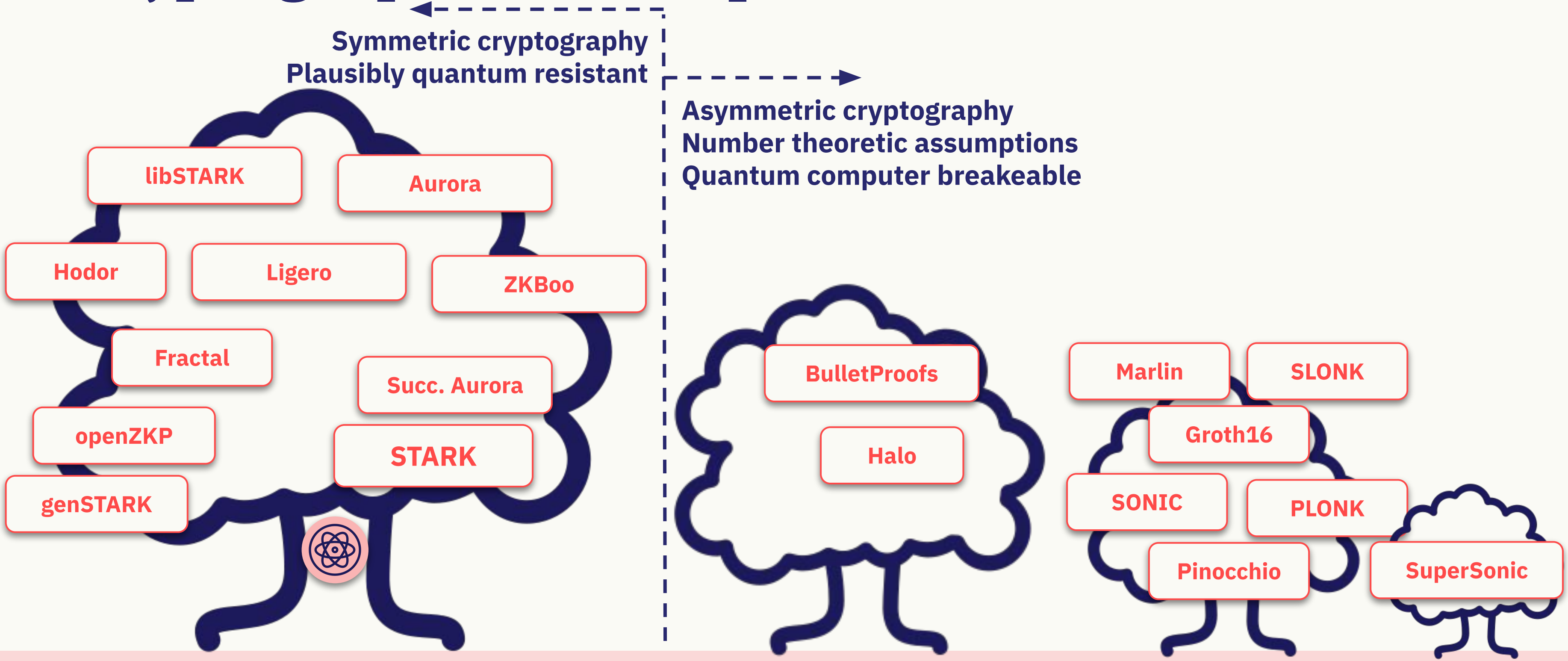


Cryptographic Assumptions	Collision-Resistant Hash	Elliptic Curve DLP	Knowledge of Exponent	Groups of unknown order
year	1976	1980s-2000s	2000s-2017	1997-2019

3. Cryptographic Assumptions

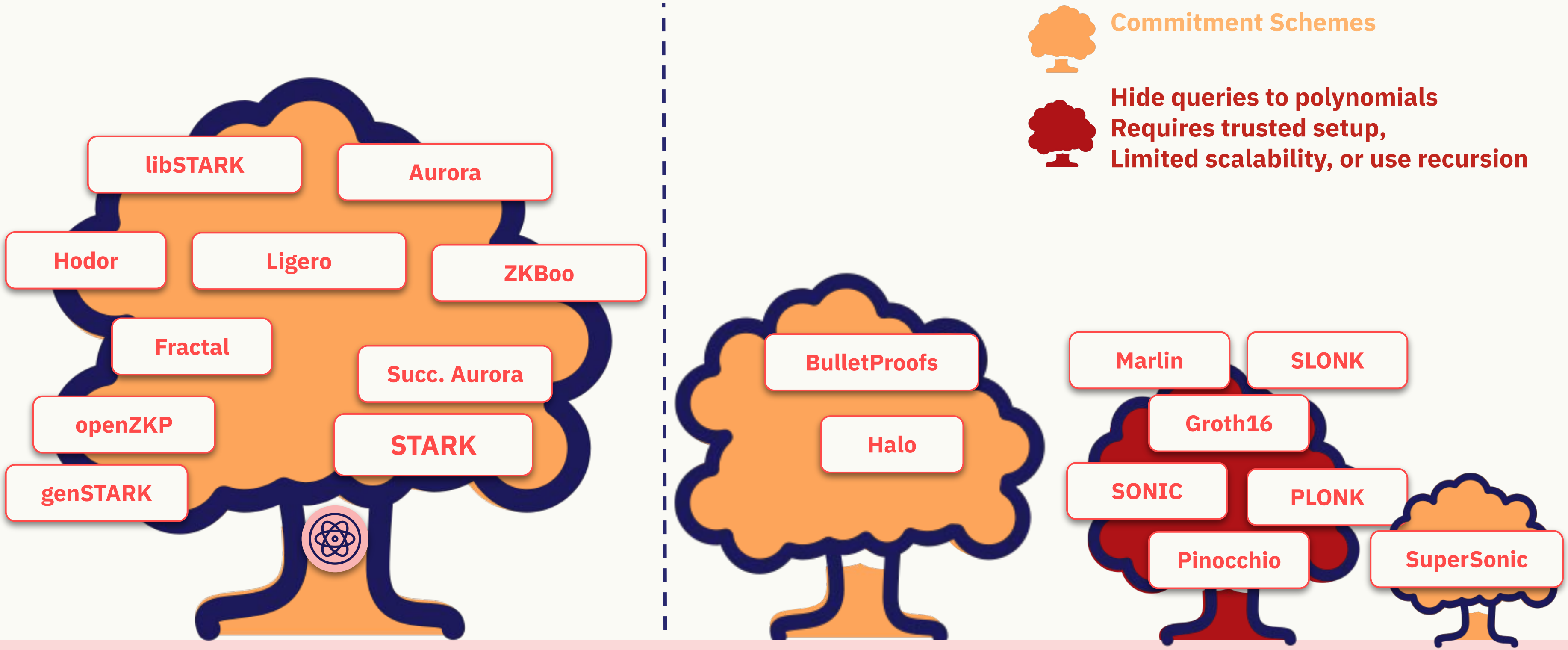


3. Cryptographic Assumptions



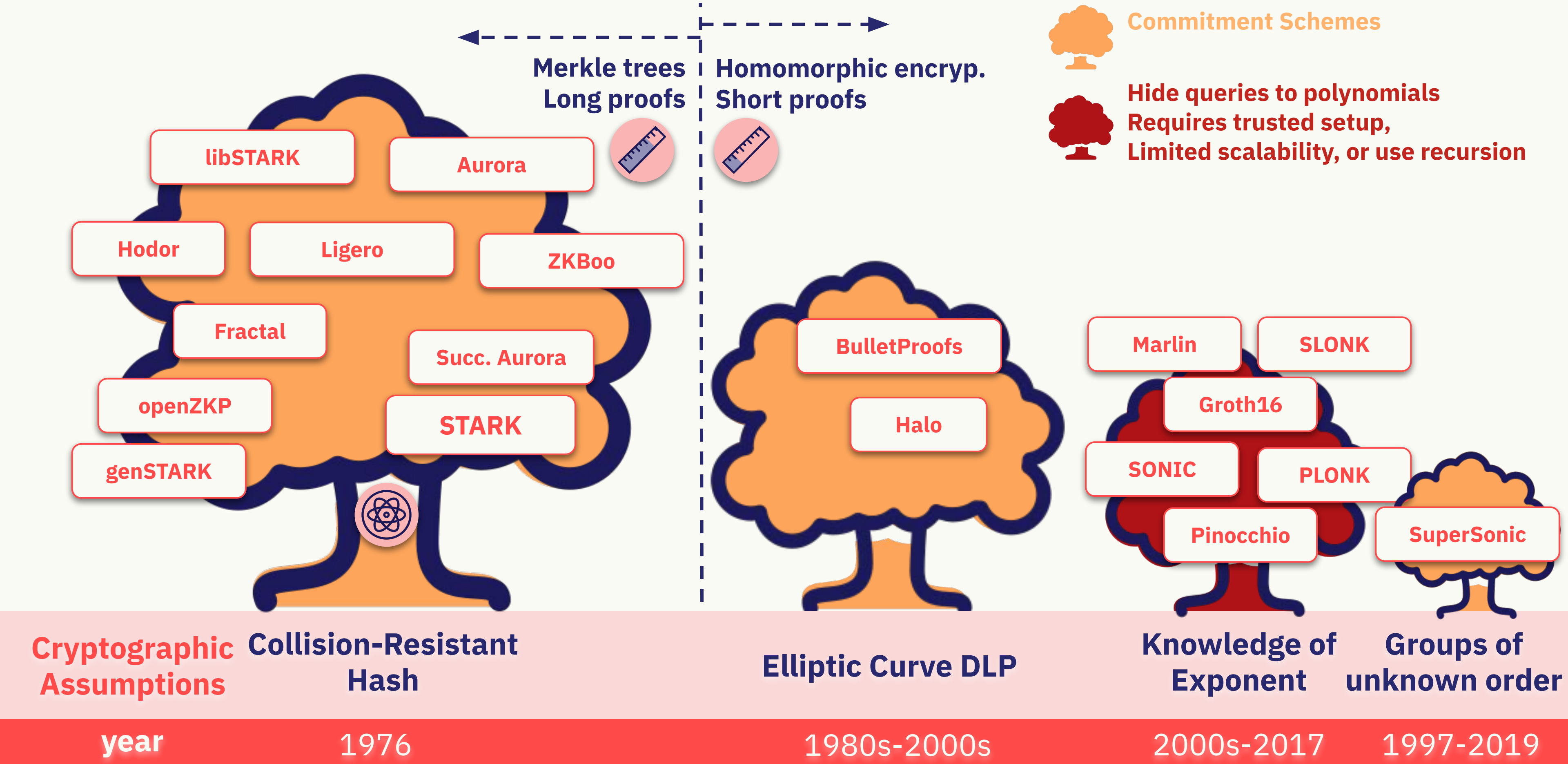
Cryptographic Assumptions	Collision-Resistant Hash	Elliptic Curve DLP	Knowledge of Exponent	Groups of unknown order
year	1976	1980s-2000s	2000s-2017	1997-2019

2. Enforcing low-degreeness

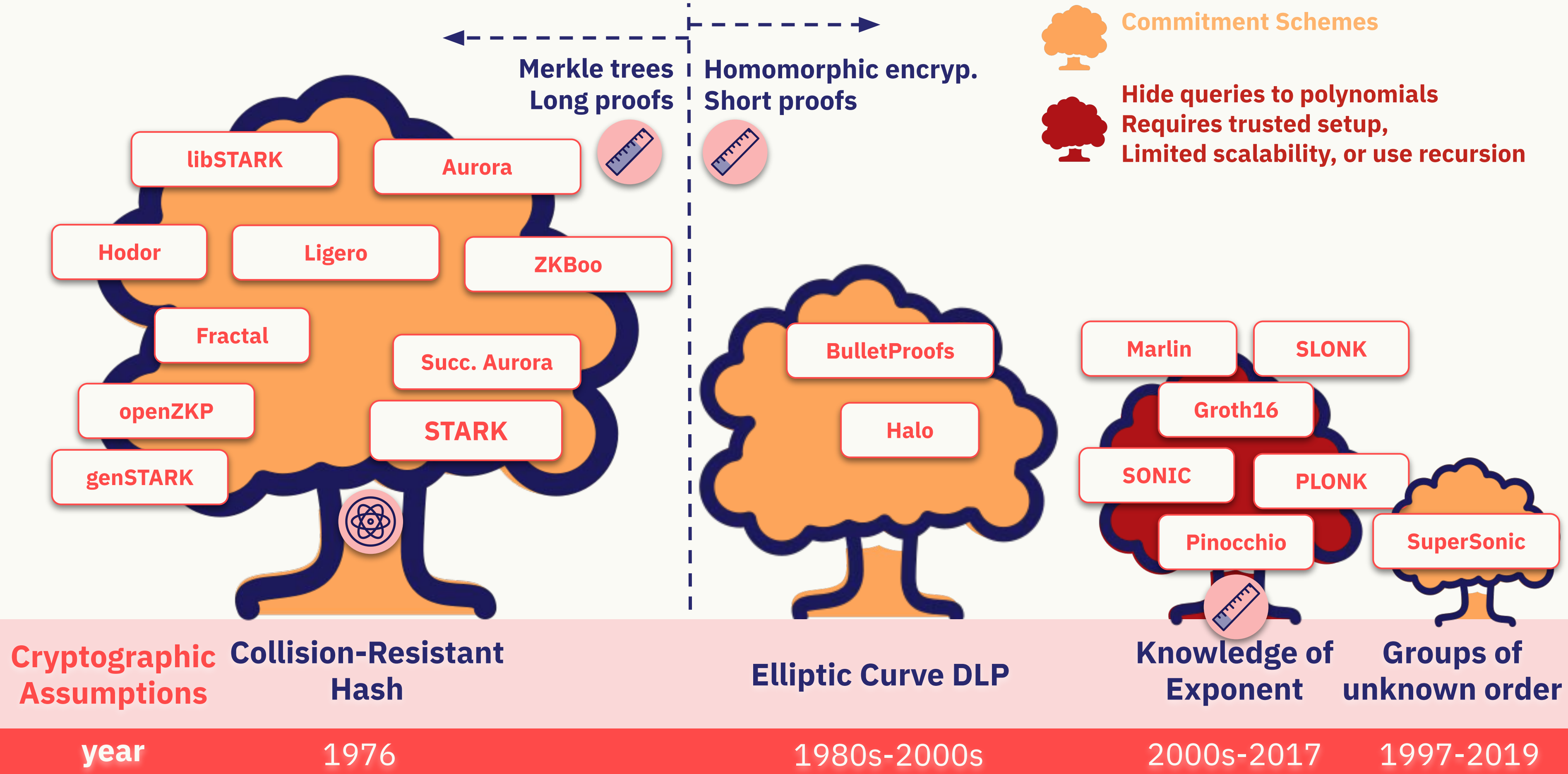


Cryptographic Assumptions	Collision-Resistant Hash	Elliptic Curve DLP	Knowledge of Exponent	Groups of unknown order
year	1976	1980s-2000s	2000s-2017	1997-2019

2. Enforcing low-degreeness



2. Enforcing low-degreeness



2. Enforcing low-degreeness

Polynomial Commitment Scheme (PCS) [Field $\mathbb{F}$, degree d]

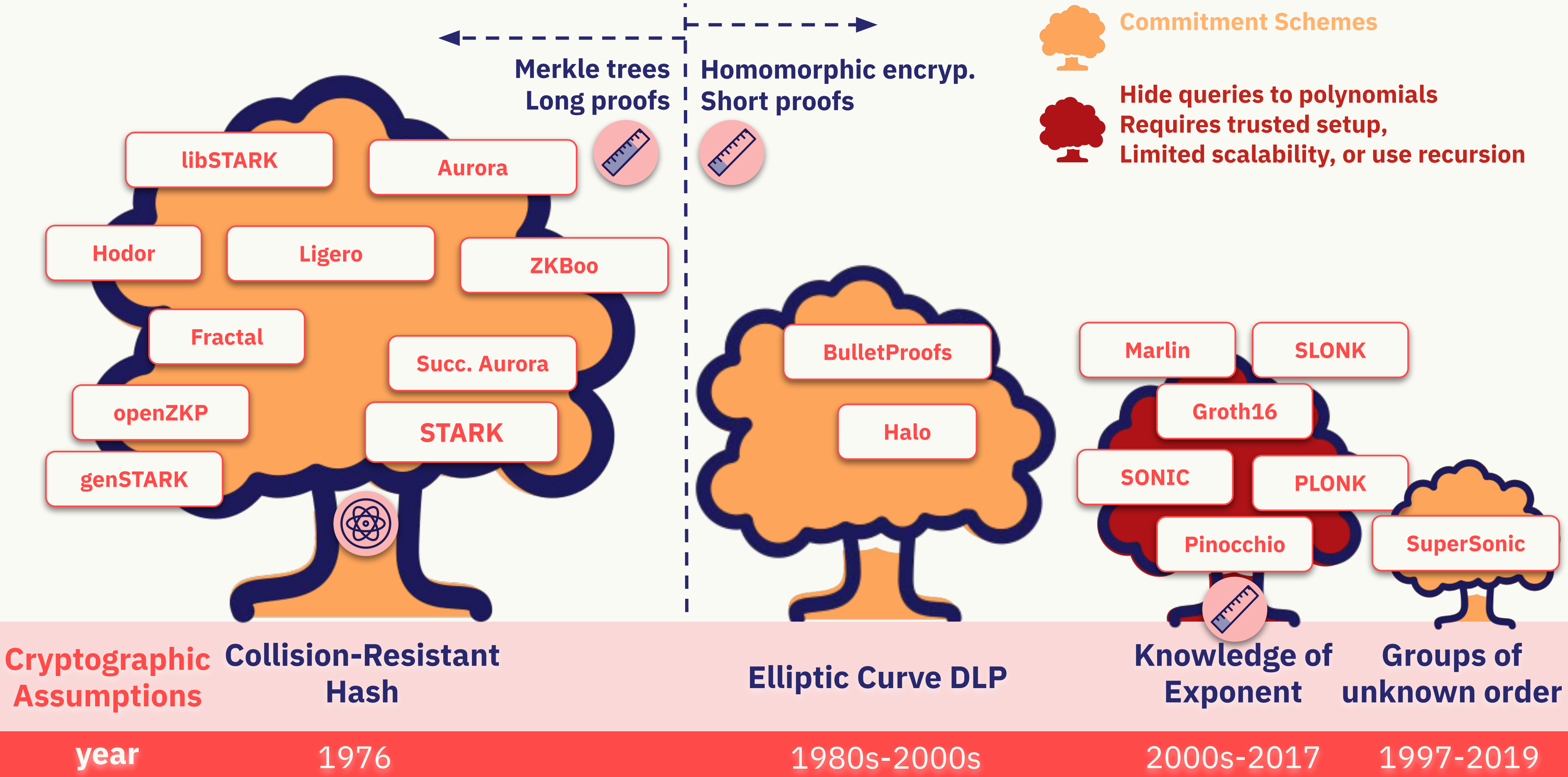
- Prover sends c = Commit ($P(x)$), deg(P)<d
- Verifier queries $z \in \mathbb{F}$
- Prover answers $a \in \mathbb{F}$, claiming "deg(P)<d and $P(z)=a$"
- Both parties interact; at end, verifier decides whether to accept/reject claim
- Want
    - **Completeness:** If $P(z)=a$ then Verifier accepts
    - **Soundness:** If $P(z) \neq a$ then whp Verifier rejects

2. Enforcing low-degreeness

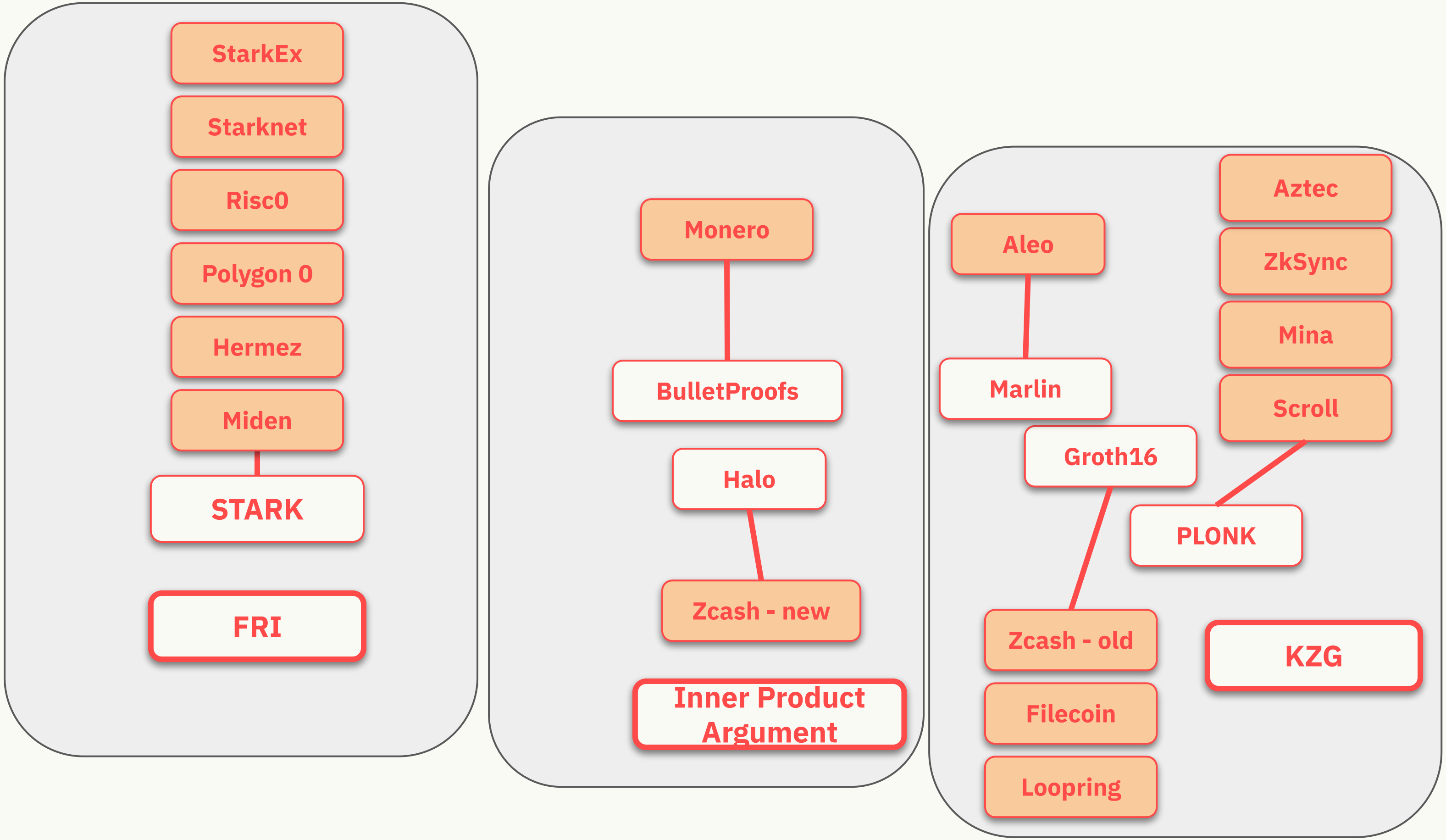
Polynomial Commitment Scheme (PCS) [Field $\mathbb{F}$, degree d]

- Prover sends c = Commit ($P(x)$), deg(P)<d
- Verifier queries $z \in \mathbb{F}$
- Prover answers $a \in \mathbb{F}$, claiming "deg(P)<d and $P(z)=a$"
- Both parties interact; at end, verifier decides whether to accept/reject claim
- Want
  - **Completeness:** If $P(z)=a$ then Verifier accepts
  - **Soundness:** If $P(z) \neq a$ then whp Verifier rejects
  - **Knowledge soundness:** efficient extractor can recover P(X) from good prover
  - **Efficiency:** low proving time, comm complexity, verification time, over all fields, ...
  - **Succinctness:** polylogarithmic verification time (and communication)
  - **Security:** minimal crypto assumptions

2. Enforcing low-degreeness



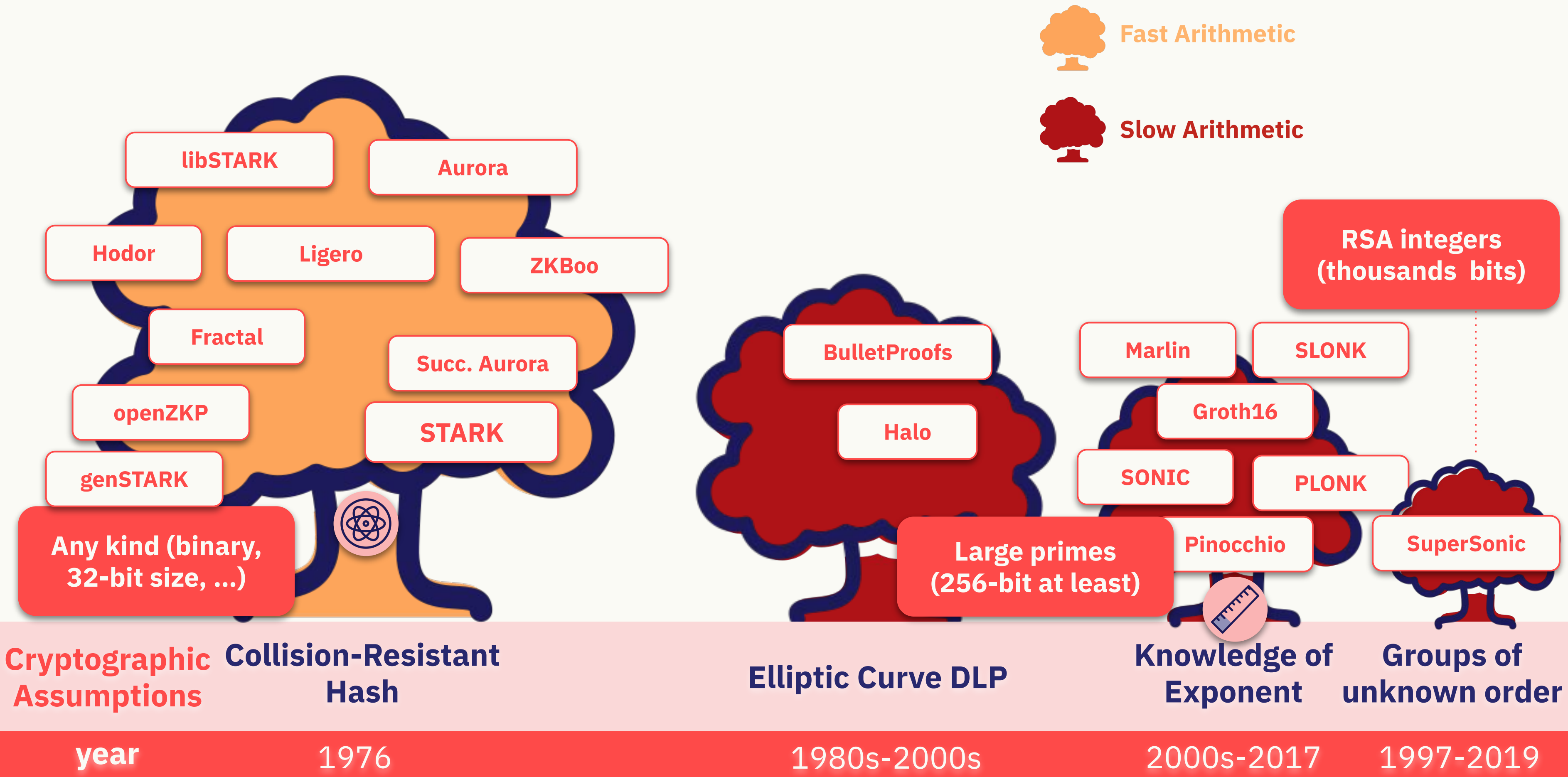
2. Enforcing low-degreeness



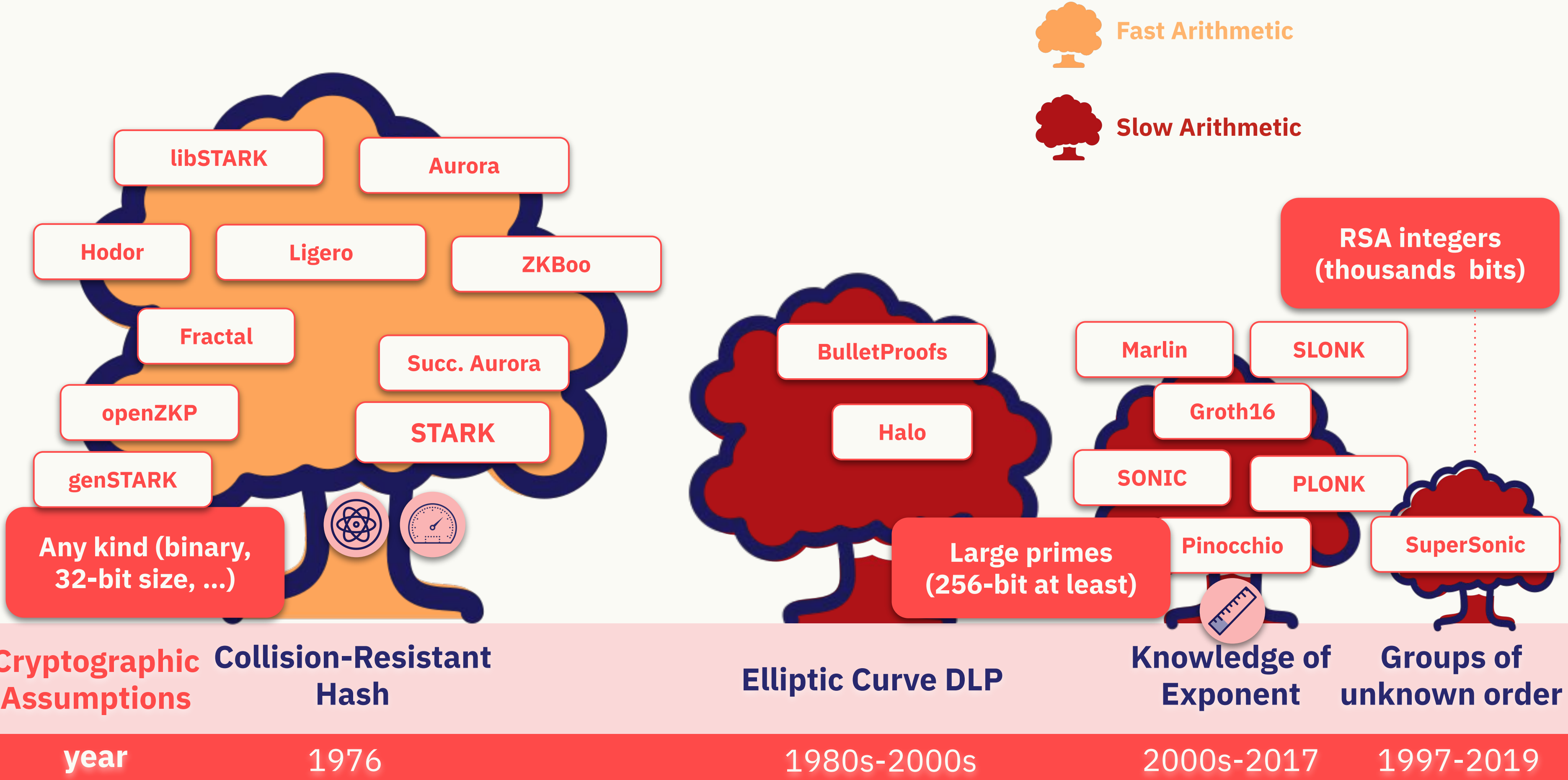
2. Enforcing low-degreeness

	FRI	Inner Product Arguments	KZG
Pros	- Succinct verification - Succinct setup - Transparent - Post quantum secure - Works over all fields	- Transparent - Short proofs (KBs) - Additivity	- Very short pf (<1KB) - Additivity
Cons	- Long proofs (dozens KB)	- Linear time verifier - Quantum breakable	- Trusted setup - Linear size/time setup - Quantum breakable
Assumptions	- Collision resistant hash	- Discrete log hardness	- Knowledge exponent

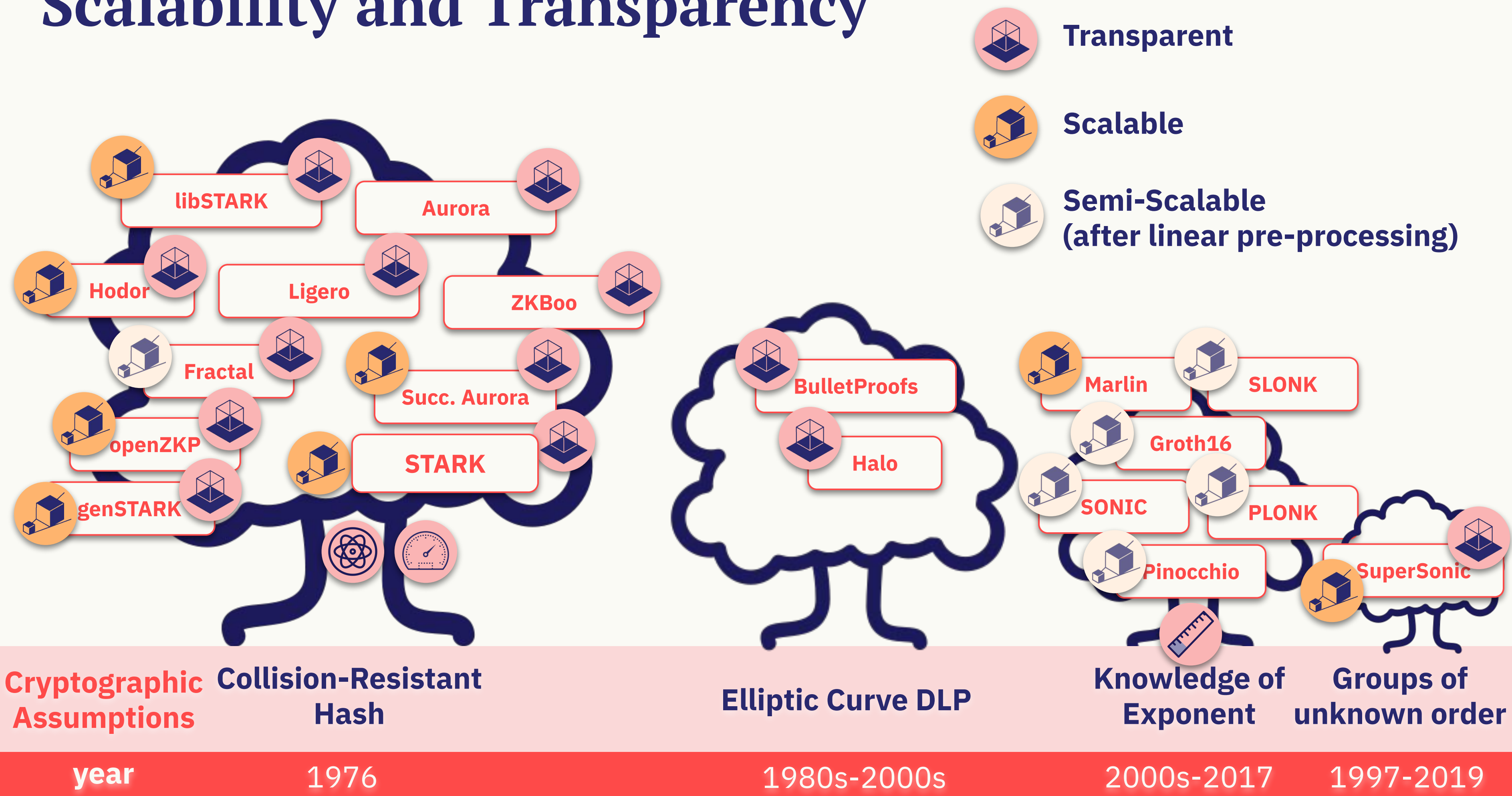
1. Arithmetization - finite field type



1. Arithmetization - finite field type

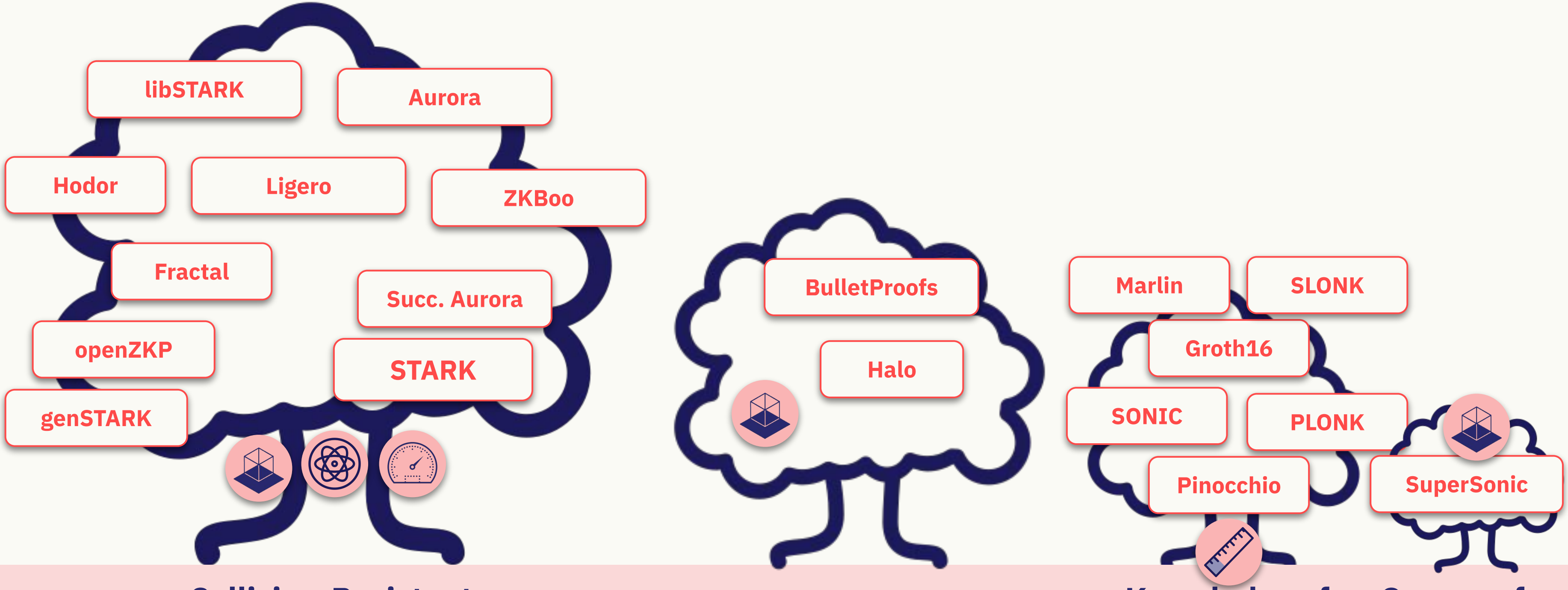


Scalability and Transparency

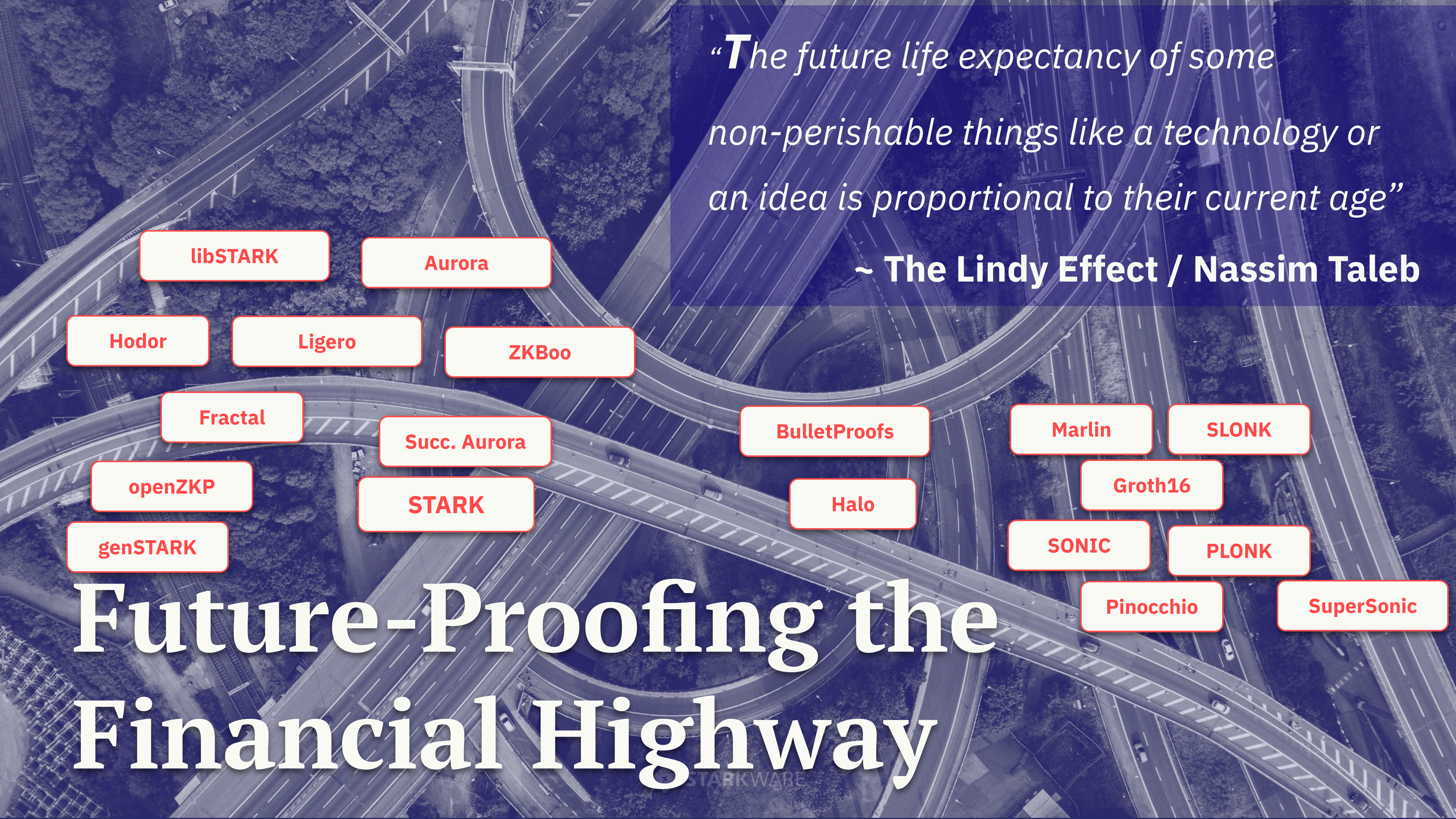


Scalability and Transparency

Transparent



Cryptographic Assumptions	Collision-Resistant Hash	Elliptic Curve DLP	Knowledge of Exponent	Groups of unknown order
year	1976	1980s-2000s	2000s-2017	1997-2019



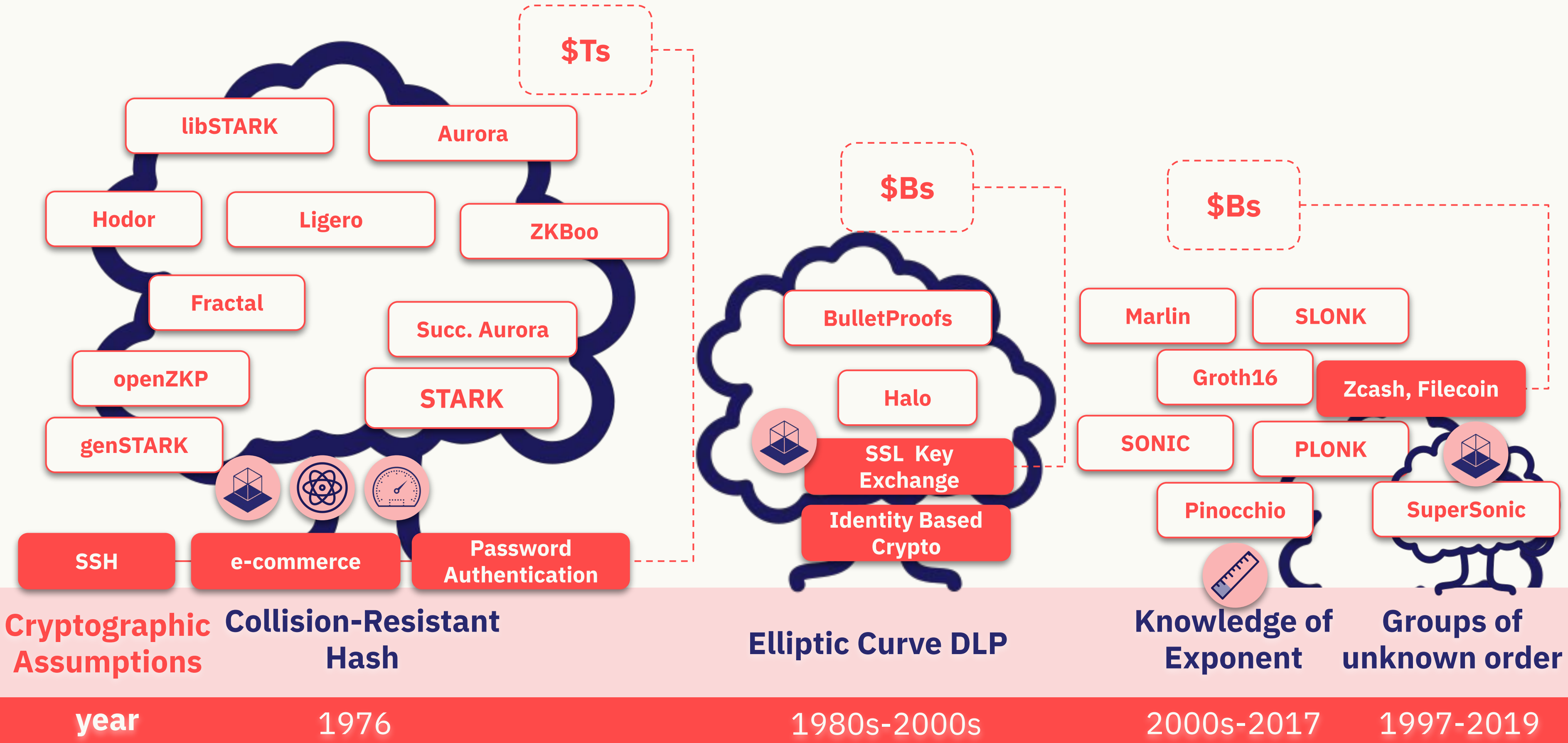
"The future life expectancy of some non-perishable things like a technology or an idea is proportional to their current age"

~ The Lindy Effect / Nassim Taleb

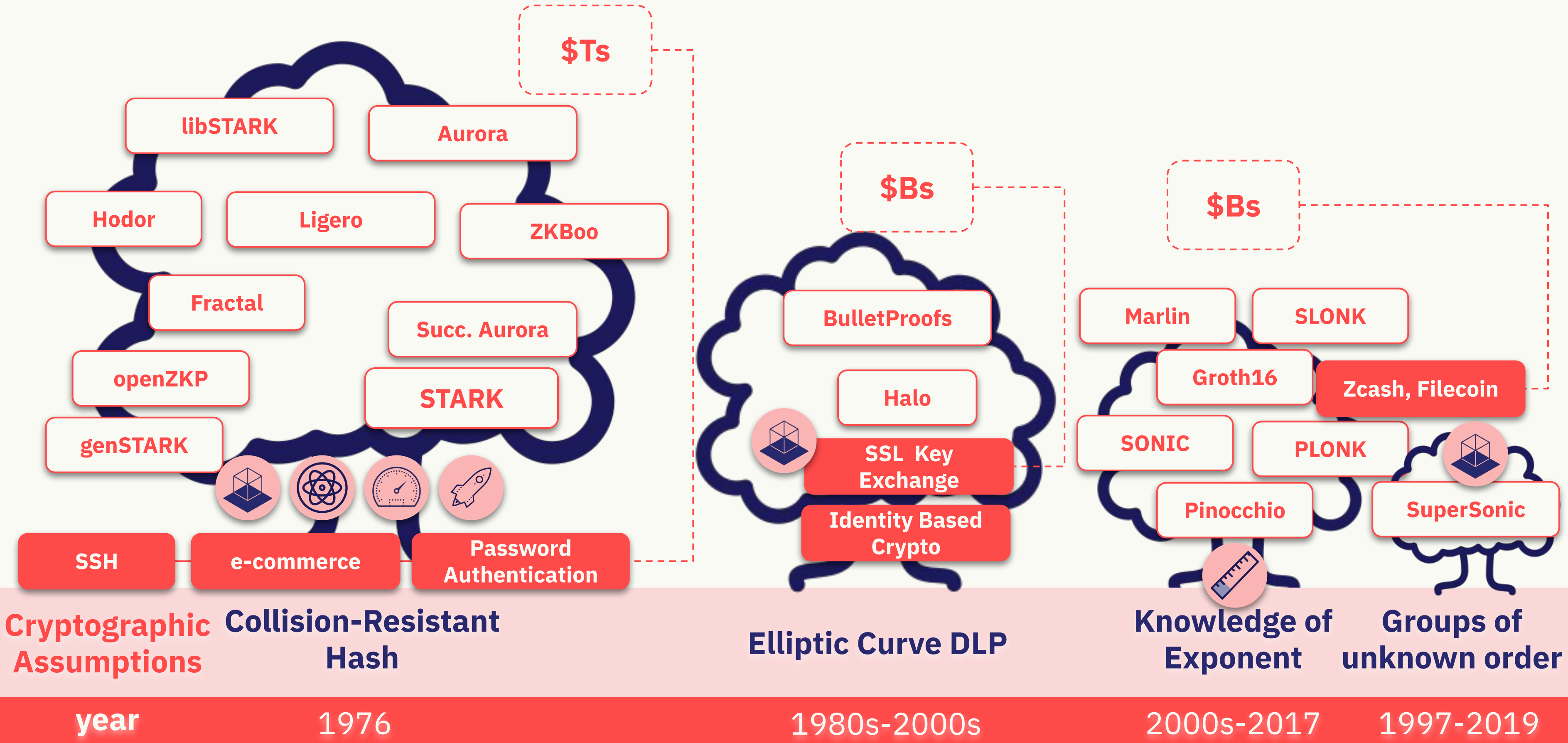
- libSTARK
- Aurora
- Hodor
- Ligero
- ZKBoo
- Fractal
- Succ. Aurora
- openZKP
- STARK
- genSTARK
- BulletProofs
- Halo
- Marlin
- SLONK
- Groth16
- SONIC
- PLONK
- Pinocchio
- SuperSonic

Future-Proofing the Financial Highway

ZKP Family Trees



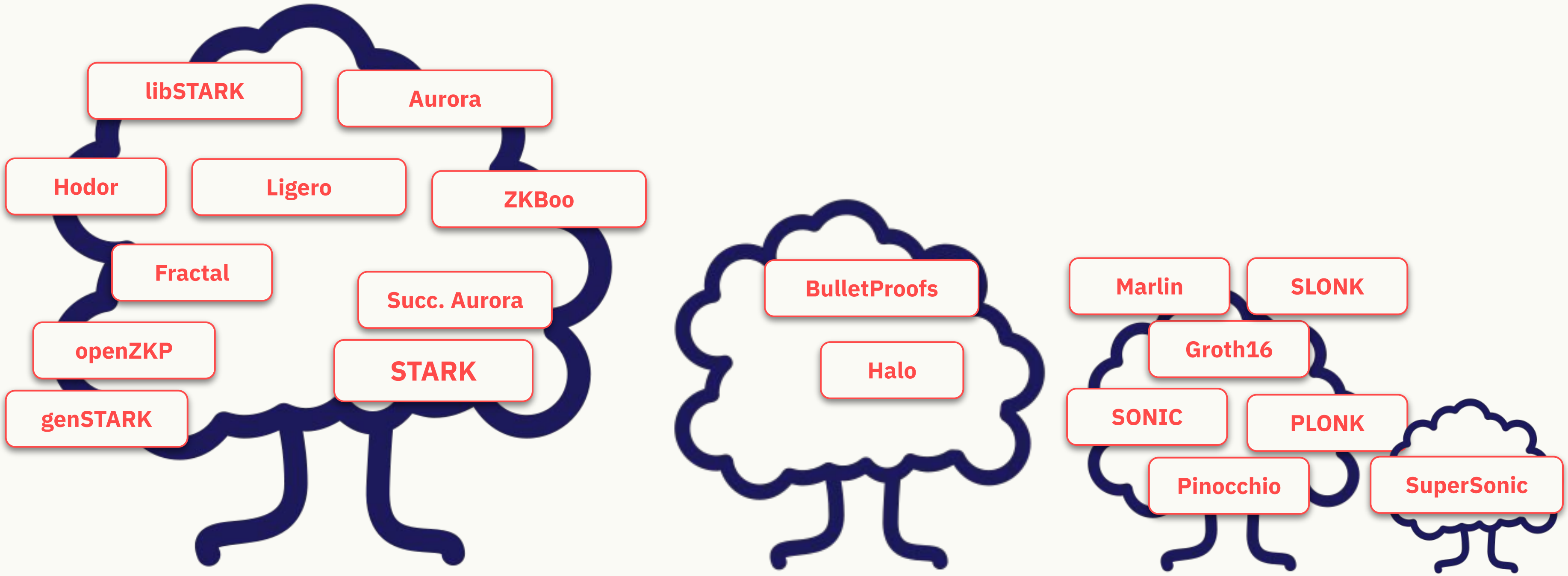
ZKP Family Trees



Summary

ZKP Cambrian explosion ongoing, expect more science!

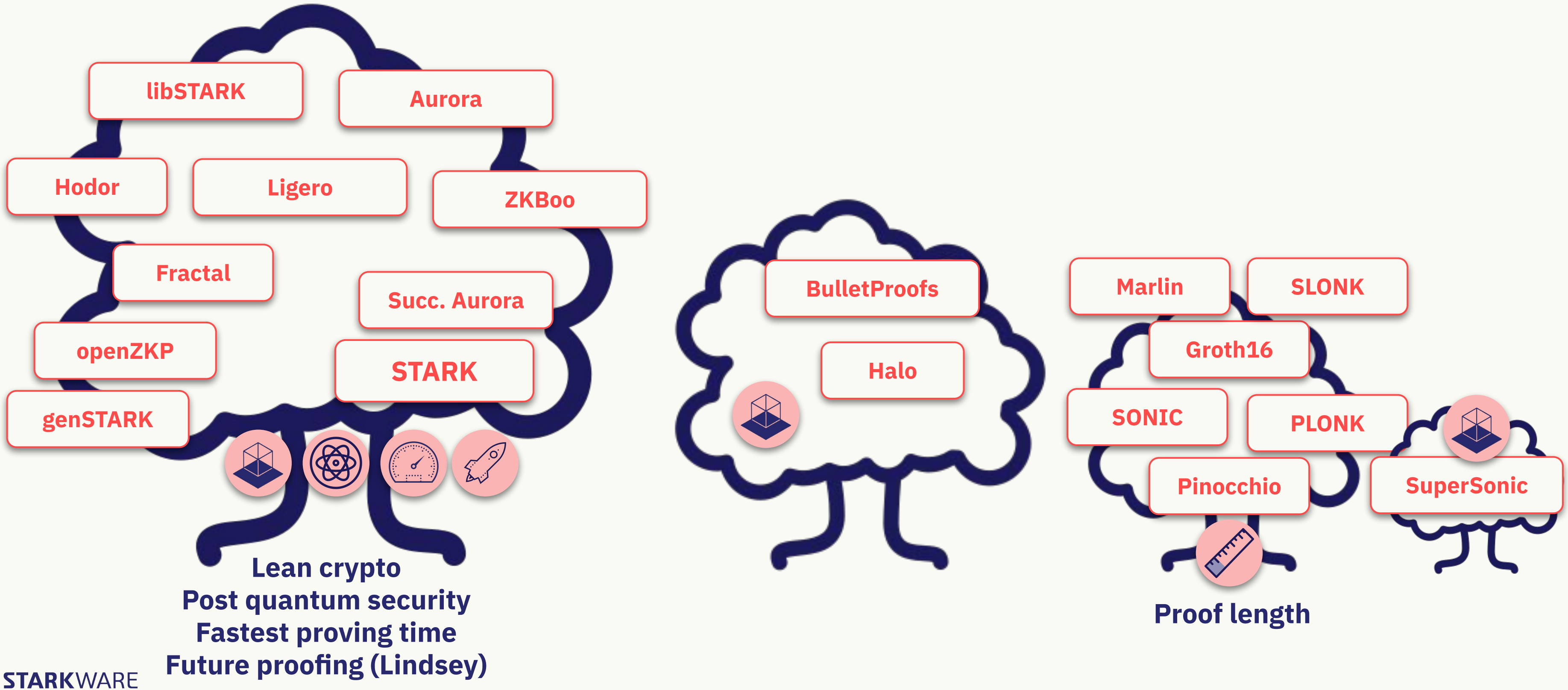
ZKP members differ by (**i**) arithmetization, (**ii**) low-degreeness, and (**iii**) crypto assumptions



Summary

ZKP Cambrian explosion ongoing, expect more science!

ZKP members differ by (**i**) arithmetization, (**ii**) low-degreeness, and (**iii**) crypto assumptions



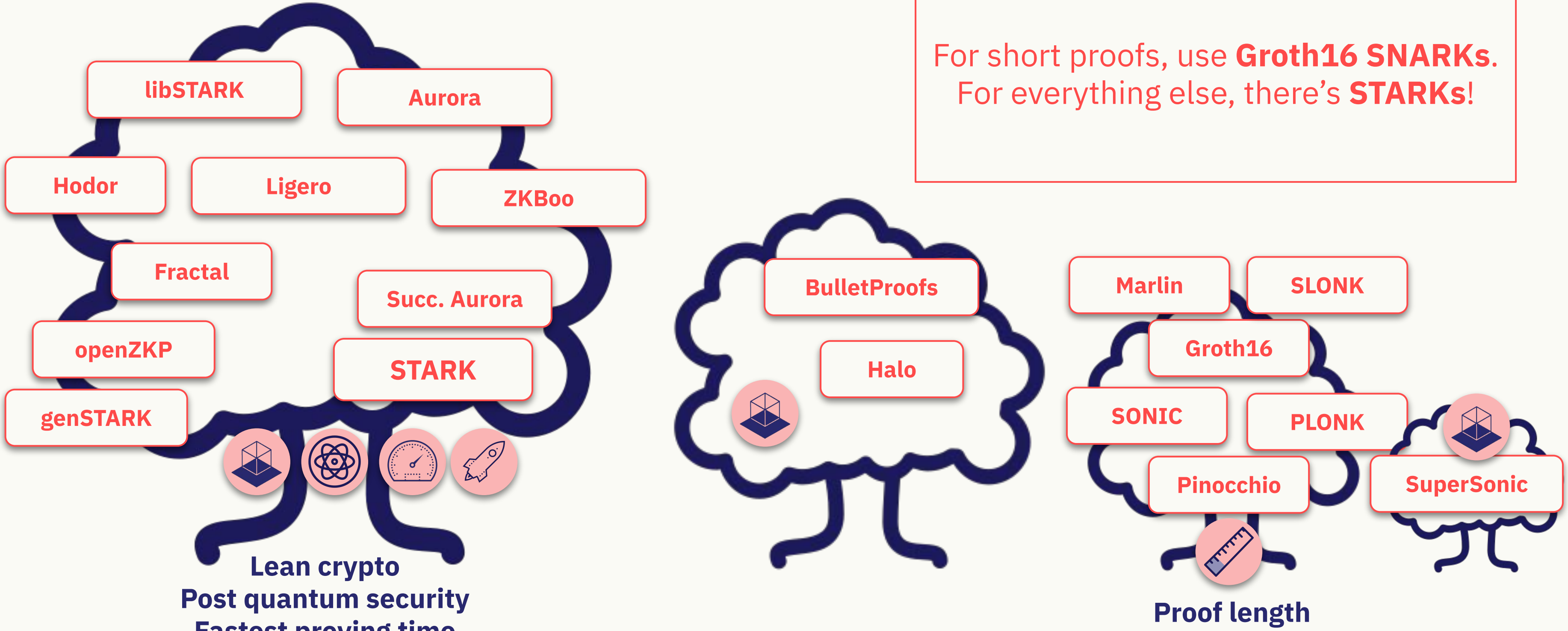
STARKWARE

Summary

ZKP Cambrian explosion ongoing, expect more science!

ZKP members differ by (**i**) arithmetization, (**ii**) low-degreeness, and (**iii**) crypto assumptions

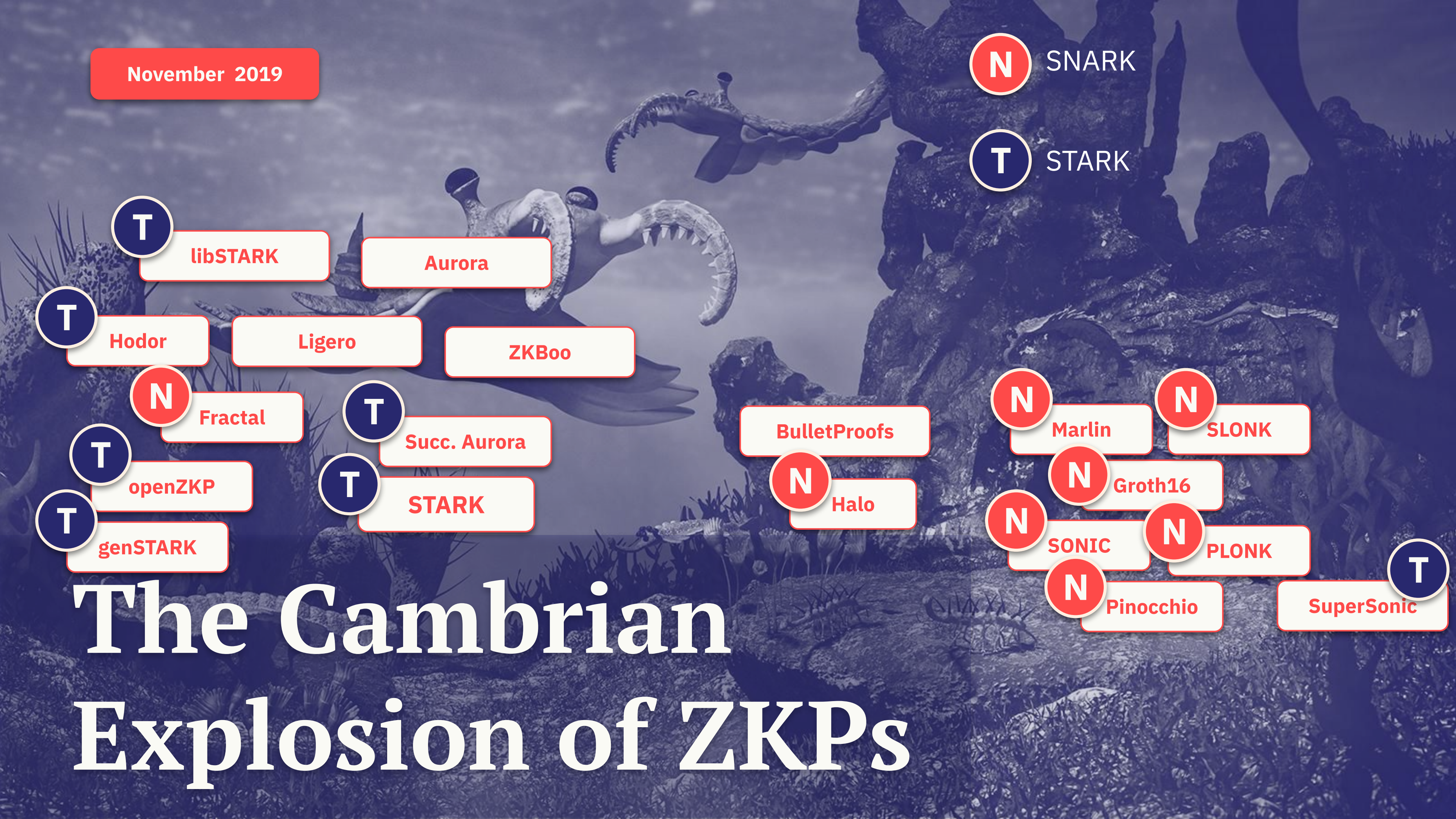
For short proofs, use **Groth16 SNARKs**.
For everything else, there's **STARKs**!

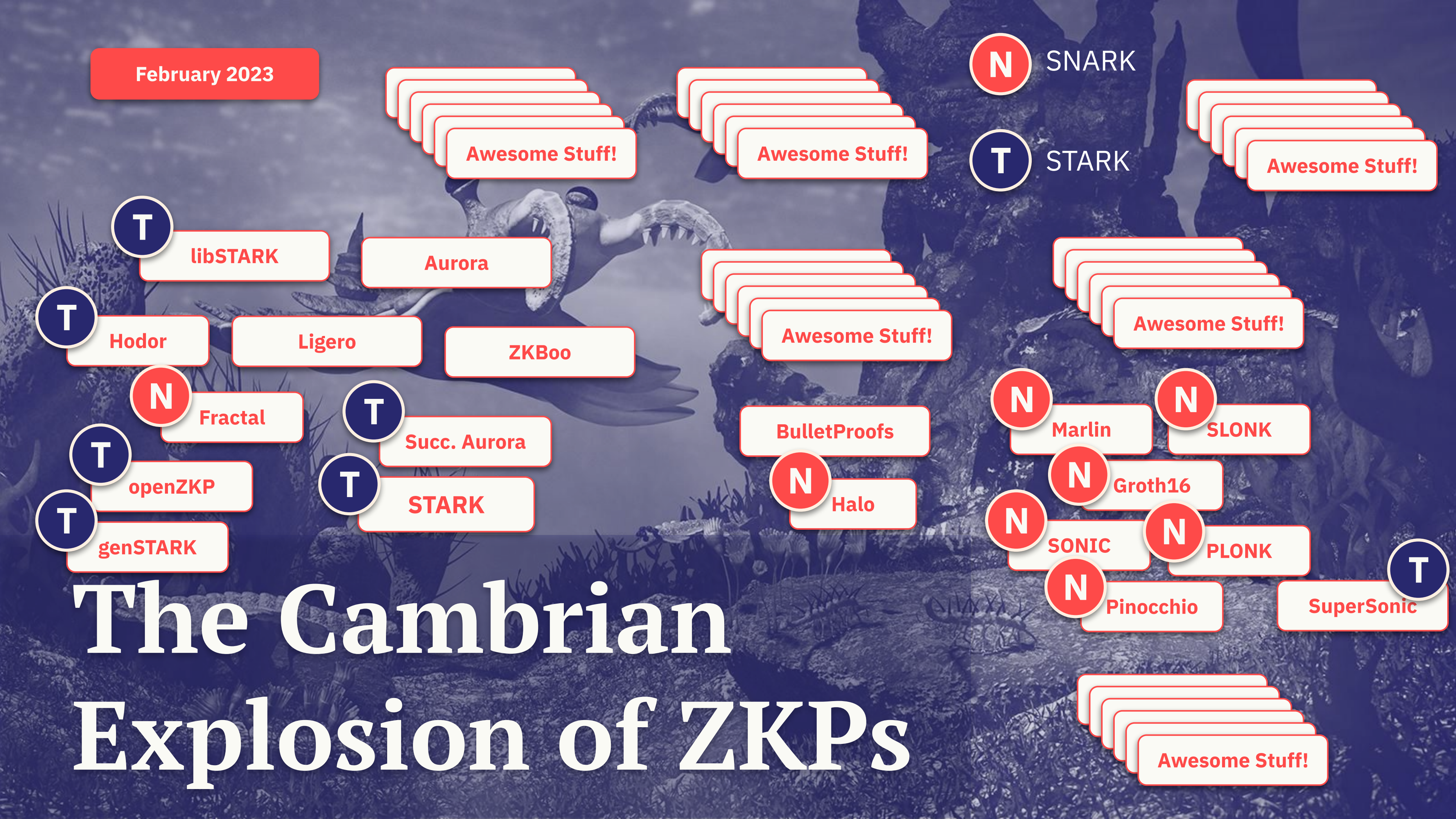


STARKWARE

The End

The Cambrian Explosion of ZKPs




February 2023
Awesome Stuff!
Awesome Stuff!
N SNARK
T STARK
Awesome Stuff!
T libSTARK
Aurora
T Hodor
Ligero
ZKBoo
Awesome Stuff!
Awesome Stuff!
N Fractal
T Succ. Aurora
T openZKP
T STARK
T genSTARK
BulletProofs
N Halo
N Marlin
N SLONK
N Groth16
N SONIC
N PLONK
N Pinocchio
T SuperSonic
The Cambrian
Explosion of ZKPs
Awesome Stuff!

Proofs of Computational Integrity (CI)

Privacy (Zero Knowledge, ZK)
Prover's private inputs are shielded

Scalability
Exponentially small verifier running time*
Nearly linear prover running time*

Universality
Applicability to general computation

Transparency
No toxic waste (i.e. no trusted setup)

Lean & Battle-Hardened Cryptography
e.g. post-quantum secure

*With respect to size of computation

STARK vs. SNARK - emphasizing different aspects

T

STARKs *must be*

Transparent no trusted setup

***Scalable**:* logarithmic verifying time **and** nearly-linear proving time

Succinct setup, at most logarithmic time

N

SNARKs *must be*

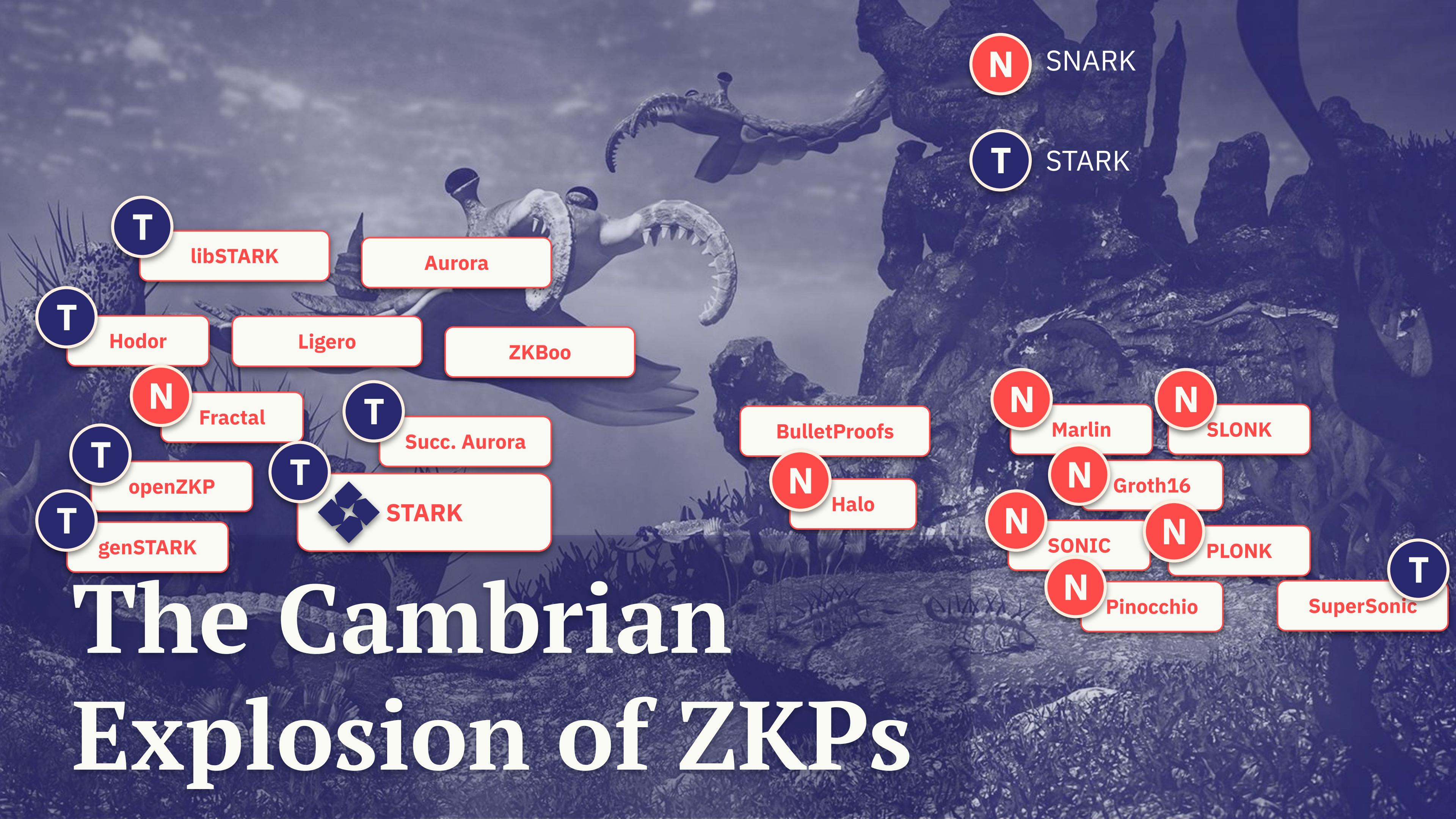
Noninteractive: pf is single message (after preprocessing)

Succinct: logarithmic verifying time

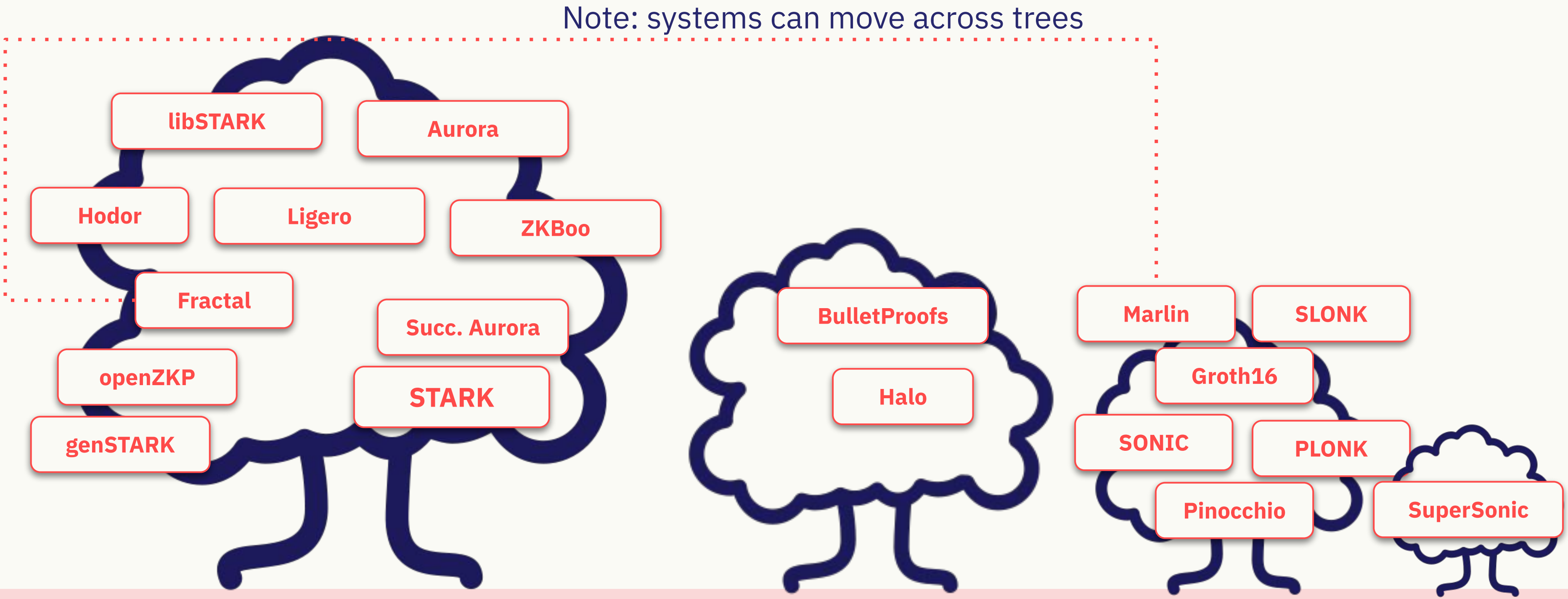
Setup can take linear time (and more)

Non-interactive STARKs are SNARKs (transparent ones)

Transparent SNARKs w/ succinct setup are STARKs


SNARK
STARK
libSTARK
Aurora
Hodor
Ligero
ZKBoo
Fractal
Succ. Aurora
openZKP
STARK
genSTARK
BulletProofs
Halo
Marlin
SLONK
Groth16
SONIC
PLONK
Pinocchio
SuperSonic
The Cambrian Explosion of ZKPs

3. Cryptographic Assumptions



Cryptographic Assumptions	Collision-Resistant Hash	Elliptic Curve DLP	Knowledge of Exponent	Groups of unknown order
year	1976	1980s-2000s	2000s-2017	1997-2019

STARK efficiency

- Arithmetization over any field
  - Initially over any “FFT-friendly” field, including small binary fields, small primes
  - Recently: over any field, using Elliptic curves [BCKL 2021-2]
- New Computational Model - IOP [RRR 2016; BCS 2016]
- Fast Reed-Solomon IOP of Proximity (FRI) [BBHR 2018]
  - Proving time is O(n), small constants (6 or less)
  - Verification time is O(log n), small constants (20 or less)
  - Nearly no soundness loss till Johnson bound [BCIKS 2020]
    - Formally: if f is delta-far from RS code, then single query-phase (log n queries to f and IOP) rejects f w.p. at least min (delta, 1-sqrt{rate})
    - Proof: relies on the Guruswami-Sudan list decoding algorithm