

Speedrun Rollups: A Beginner's Guide to L2s, ZK, and WTF People are Talking About on Panels

It's a 1.5 hour speedrun maybe idk

Emily Lin

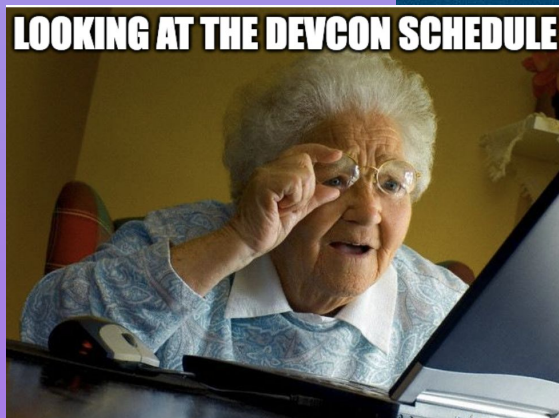
Devrel Lead, Scroll



Agenda

Aka me subjecting you to
the random stuff I get nerd
sniped by

1. L2 Fundamentals
2. Some Knowledge About Zero-Knowledge
3. Stuff You Might Hear About on a Panel

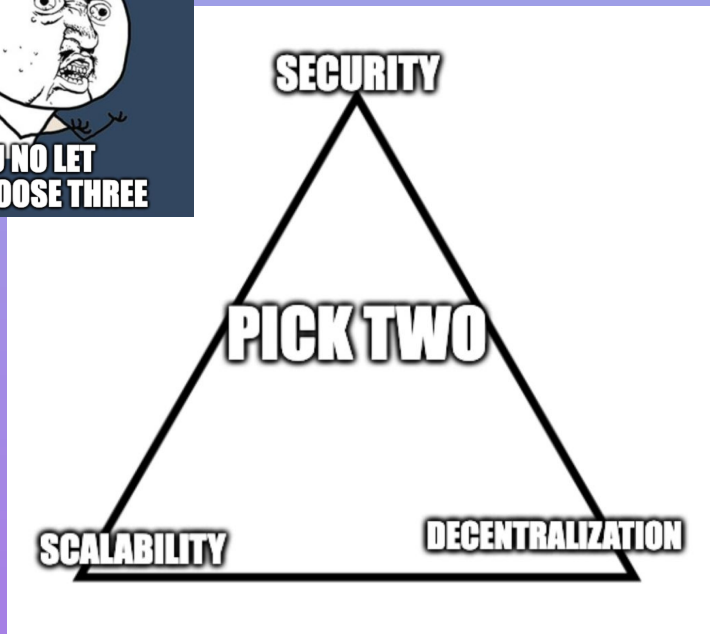




Section 1

L2 Fundamentals





Blockchain Trilemma

Secure and decentralized

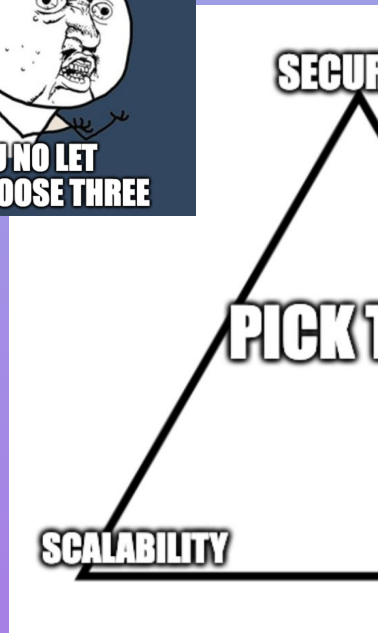
Requires every node to verify each transaction, which limits transaction throughput. Creates barriers to entry due to high gas costs

Decentralized and scalable

Requires sharing more transactions across nodes, but the more transactions, the longer the delay, the higher the probability of an attack during the delay

Secure and scalable

Requires increasing the size of Ethereum nodes and thus hardware requirements and thus accessibility and thus decentralization



Trilemma

ized

verify each transaction, which
ghput

lable

transactions across nodes, but
the longer the delay, the higher
ack during the delay

the size of Ethereum nodes and
ments and thus accessibility and

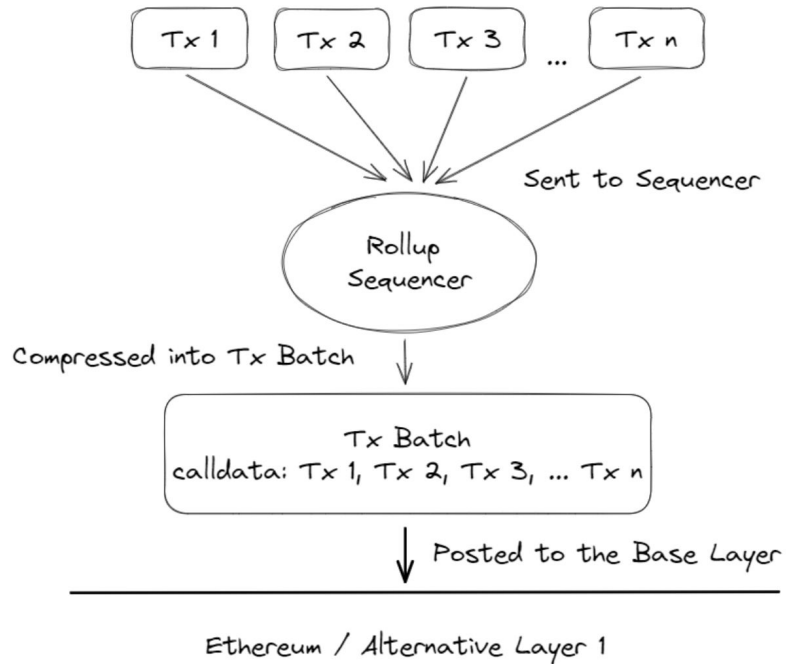
Introducing L2s.

A **layer 1** is the base blockchain like Ethereum or Bitcoin. It includes:

1. A network of node operators
2. A network of block producers
3. The blockchain itself and the history of data
4. The network's consensus mechanism (i.e., proof of stake)

A **layer 2** is a separate blockchain that extends Ethereum and **inherits the security guarantees of Ethereum (aka uses Ethereum for data availability)**.

Note side chains like Polygon PoS don't count



How Do Rollups Work

Rollups **take the execution burden off** of the L1 by executing transactions off-chain and then “rolling them up” into a single transaction to submit back to the L1.

Rollups are secure because **reverting a rollup transaction** after it has been uploaded to the L1 **requires reverting the L1 itself**.

Components of a Rollup.

L1 Contracts

1. Main rollup contract: for storing rollup blocks, tracking deposits, and monitoring state updates
2. Verifier contract (for ZK rollups): for verifying proofs
3. Bridge contract: for message passing between L1 and L2

Off-chain VM

1. Execution/computation of transactions happen in a separate virtual machine from the EVM

Operator

1. Responsible for combining and compressing transactions, and then submitting the block to Ethereum
2. Can be a node or a set of components, which often include things you might have heard of like sequencers, validators, provers, aggregators, and proposers

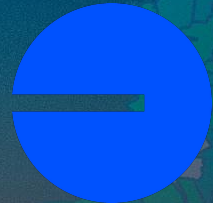


Let's dissect the rollup landscape

Optimistic Rollups.

Fraud/Fault Proofs: Innocent until proven guilty

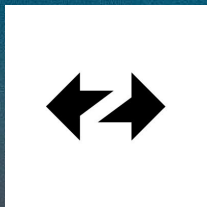
1. Optimistically assumes data is valid – fraud proofs
2. Relies on validators and crypto-economic incentives to dispute a state transition (challenge period of 7 days)
 - a. Note that third party bridges are required for the “feeling” of immediate L2 to L1 withdrawals - aka hackity hack hack vector
3. Requires at least one honest node
4. In case of fraud, L2 state is reverted
5. Needs to **post all tx data** back to L1



Zero Knowledge Rollups.

Validity Proofs: Verify, don't trust

1. Rollup state is posted alongside a validity proof, verified by the L1 verifier contracts
2. ZK proofs are used for validity proofs, as they require the least amount of information to be posted back to the L1 (data incurs gas costs) – *note that this does not mean a ZK rollup guarantees privacy*
3. Only need to post state transition data back to L1
4. Fast **finality** – no challenge period as data is posted with the proof of correctness



How do ZK rollups differentiate



**A zkEVM is an
EVM-compatible virtual
machine that is able to prove
the correctness of execution**



zkEVMs are complex

It does magic that makes code provable

EVM as is isn't built to be easily compatible with
ZK circuits

From its special opcodes (i.e. machine
instructions), storage schemes, and hardware,
the EVM makes it complex to generate and
prove efficient ZK proofs

Different Types of ZK EVMS.

Type 4

High level language equivalence
Not bytecode equivalent

Write in Solidity/Vyper/etc, tools change, major code changes

Type 3

High level language equivalence
Almost EVM equivalent

Write in Solidity/Vyper/etc, tools don't change, minor code changes

Type 2

High level language equivalence
EVM equivalent (bytecode equivalence)

Write in Solidity/Vyper/etc, tools don't change



Type 1

High level language equivalence
Ethereum equivalence

Write in Solidity, tools don't change, state storage and hashing exactly the same

Different rollup stages (by L2Beat).

Stage 0 — Full Training Wheels: At this stage, the rollup is **effectively run by the operators**. Still, there is an **source-available software that allows for the reconstruction of the state** from the data posted on L1, used to compare state roots with the proposed ones.

Stage 1 — Limited Training Wheels: In this stage, the rollup transitions to being **governed by smart contracts**. However, a **Security Council might remain in place** to address potential bugs. This stage is characterized by the **implementation of a fully functional proof system**, decentralization of fraud proof submission, and **provision for user exits** without operator coordination. The Security Council, comprised of a diverse set of participants, provides a safety net, but its power also poses a potential risk.

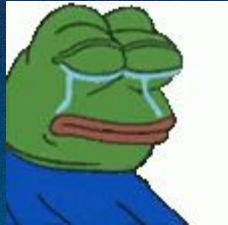
Stage 2 — No Training Wheels: This is the final stage where the rollup becomes **fully managed by smart contracts**. At this point, the fraud **proof system is permissionless**, and users are given ample time to exit in the event of unwanted upgrades. The **Security Council's role is strictly confined to addressing soundness errors** that can be adjudicated on-chain, and users are protected from governance attacks.

Trade-offs you might think about.

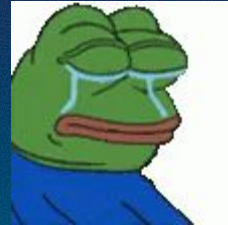
- Withdrawal times (finality)
- Security guarantees and trust assumptions
- Throughput
- Developer experience
- Decentralization
- Open-source software
- Community
- Ecosystem support
- TVL and users
- Network effects



But wait.
Rollups can still be expensive.



why.



Remember DA?

**Interacting with
Ethereum is still
expensive.**

Introducing EIP-4844

- BLOBSSSSSSSS
- New type of **ephemeral data storage** separate from `calldata`, which is stored forever
- Rollups only need DA for temporary validation, so storing on `calldata` is inefficient
- Blobs also have a **separate fee market**, making more resilient to fluctuations in Ethereum

Exploring Other Scaling Solutions.

A **validium** is a chain that performs off-chain computation and data availability, publishing ZK-proofs back to the main chain. As a result, they achieve much higher throughput than ZK-rollups, but lack protections provided by L1 **AKA ZK-rollups + Off-chain DA**

A **optimum** is an **optimistic rollup + off-chain DA**.

A **volition** is the marriage of a validium and a ZK-rollup, allowing users to switch between off-chain and on-chain data availability.

Quiz time + stretch sesh



Which type of zkEVM is bytecode-compatible

Types...

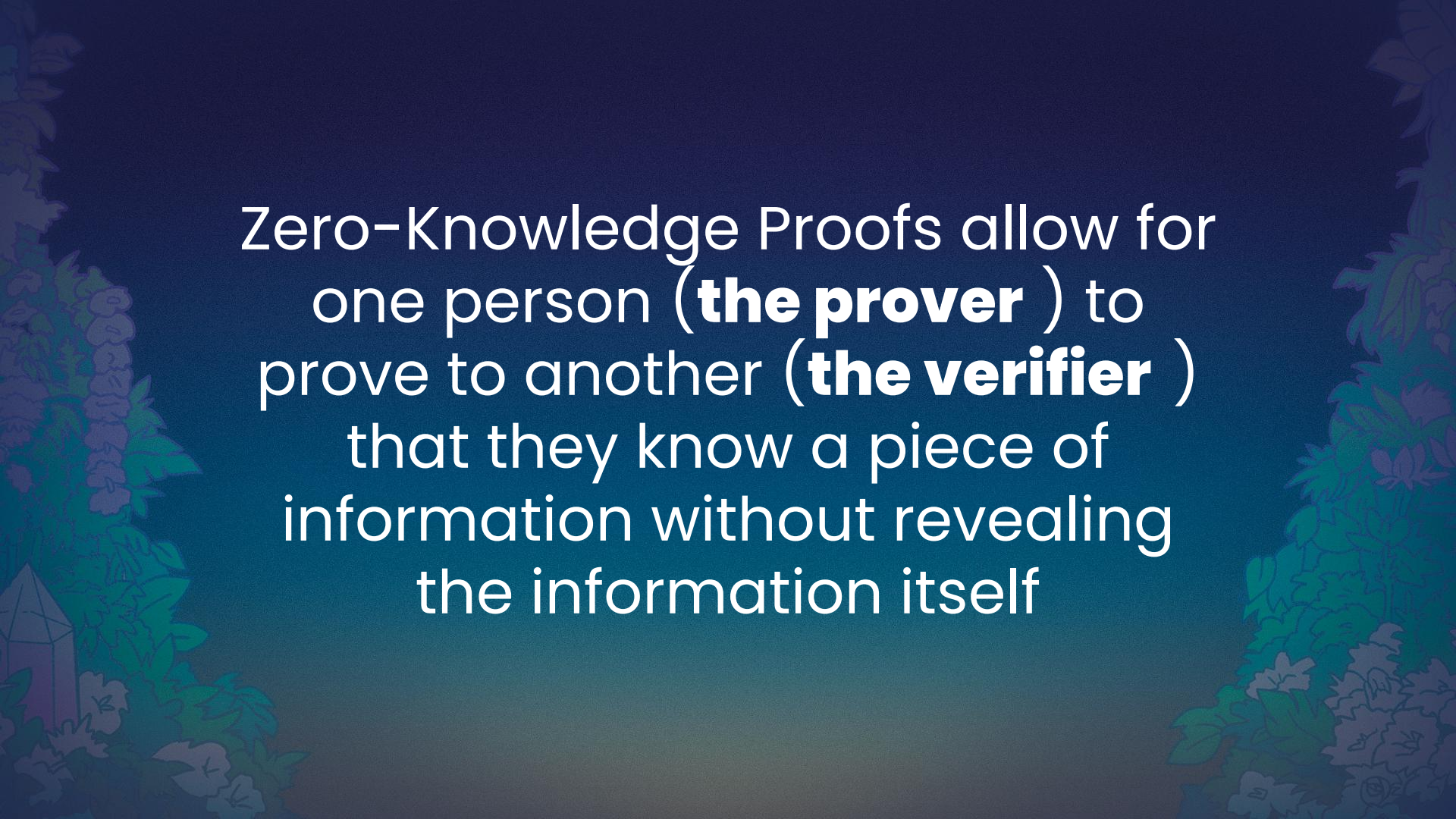
- A. Type 4
- B. Type 3
- C. Type 2
- D. Type 1
- E. Type 0
- F. Type 3 and 4
- G. Type 0 and 1
- H. Type 1 and 2
- I. All of the above



Section 2

Some Knowledge About Zero-Knowledge.





Zero-Knowledge Proofs allow for
one person (**the prover**) to
prove to another (**the verifier**)
that they know a piece of
information without revealing
the information itself

There are 3 main properties of a ZKP

Completeness: If the prover knows the information, they can convince the verifier with their proof.

Soundness: If the prover doesn't know the information, they can't successfully trick the verifier.

Zero-Knowledge: The proof doesn't reveal any part of the actual information itself—only that the prover knows it.

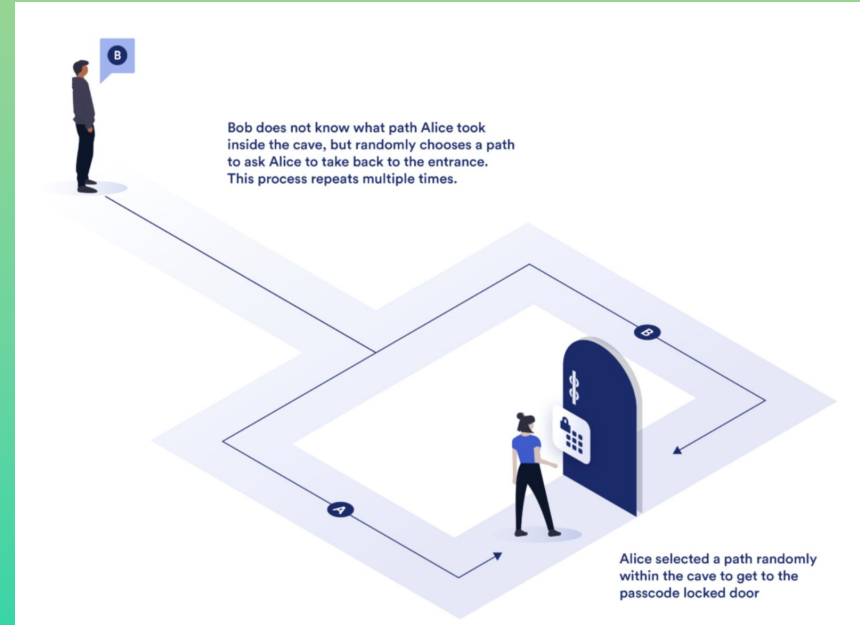


No cheating bro.

Interactive proofs

- Requires back and forth between prover and verifier
- Verifier asks the prover questions or challenges them multiple times to confirm they truly know the secret

Knowing the code to a door



Non-interactive proofs

- Prover provides a one-time proof that anyone can verify later without direct interaction
- Ex. Proving you know the solution to a Sudoku puzzle
 - Create a proof that allows anyone to verify that each row, column, and box meets the Sudoku rules without seeing the actual numbers you filled in

	7		3	1			5	
	1	5		8				
9			5				8	1
5		3	6			2		
2			1		9			4
		7			8	9		6
9	6				5			7
				6				
	2			9				

A deeper dive into the Sudoku example.

We utilize **hashing** – a one-way transformation of data

1. Hash each cell's number in the Sudoku puzzle.
2. Combine hashes into “**commitments**” for rows, columns, and grids, ensuring each satisfies Sudoku's rules and create proofs that the conditions are satisfied
 - a. I.e, all numbers 1 to 9 are present without duplicates
3. Verifier checks commitments to confirm each row, column, and grid has unique values from 1 to 9 without repetition, without seeing the actual numbers.

zkSNARKs.

Succinct Non-Interactive Argument of Knowledge

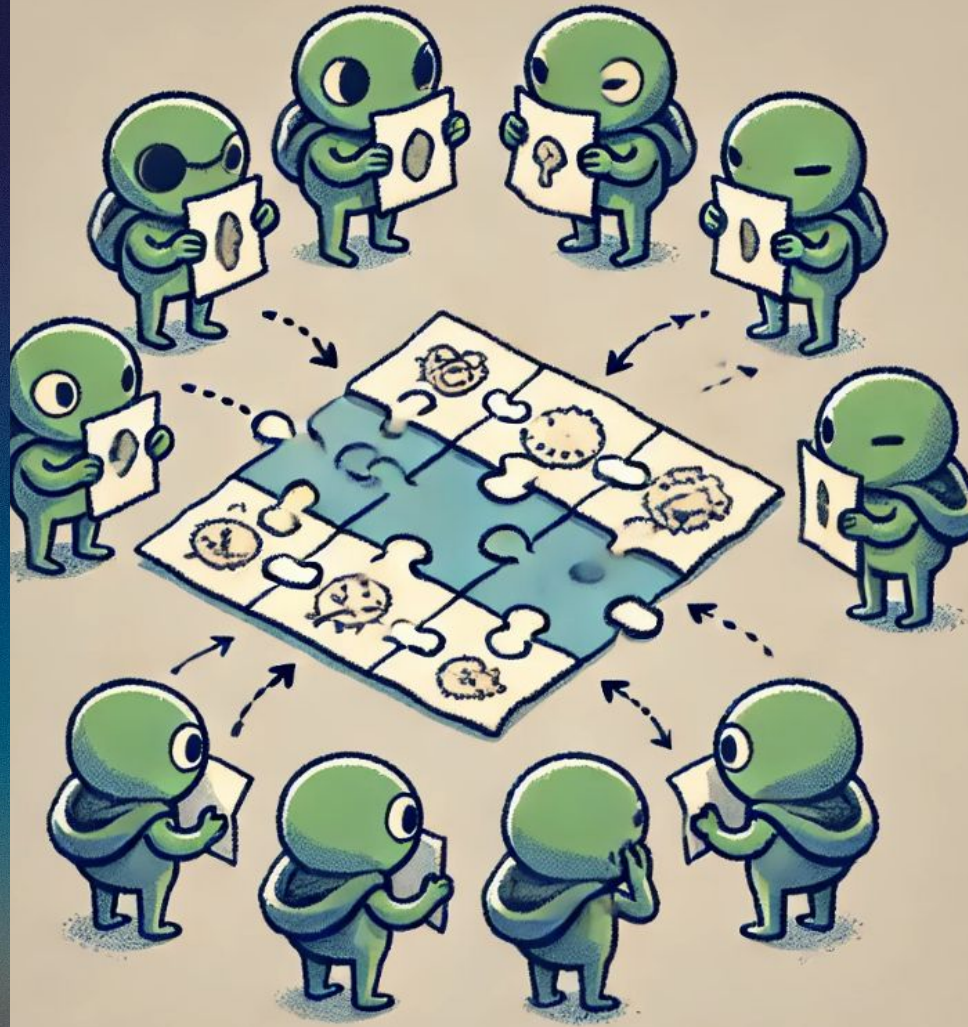
- Succinct (small in size)
- Non-interactive (does not require back-and-forth between the prover and verifier)
- Compact proofs and allows for fast verification
- Requires a trusted setup
 - A preprocessing phase which uses randomness to generate proving/verifying keys
 - The party running this preprocessing shouldn't save the private keys. Otherwise, it'd be able to create fake proofs

Multi-Party Computation (MPC).

- Cryptographic method that allows multiple parties to jointly compute a result based on their private inputs, without revealing those inputs to each other
- Splits data into “shares” and distribute them among participants
- Each participant works on their own data share, and the results are combined at the end to get the final outcome—without ever exposing the original, private data
- Good for use cases like trusted setups and private keys
- Vulnerable to collusion, where some participants might work together to deduce information.
- Computational and communication overhead increases as the number of participants grows.

Let's build a treasure map!

1. A group of frogs each has a part of a treasure map that leads to the treasure, but they don't want to share their full piece with others.
2. Each frog breaks their map piece into smaller "shares" and gives one share to every other frog.
3. No individual share reveals the treasure's location, so no frog has enough information alone.
4. When it's time to find the treasure, the frogs combine all their shares together.
5. By combining the shares, they recreate the full map without ever needing to share their individual pieces directly.



zkSTARKs.

Scalable Transparent Argument of Knowledge

- Transparent setup (no trusted setup)
- Also non-interactive
- Scalable proof generation (can handle larger, more complex computations)
- However, larger proof sizes as compared to SNARKs
- Very fast verification for large data, less efficient for smaller data sets
- Innovations like **wrapping STARKs inside of SNARKs to leverage best of both worlds**

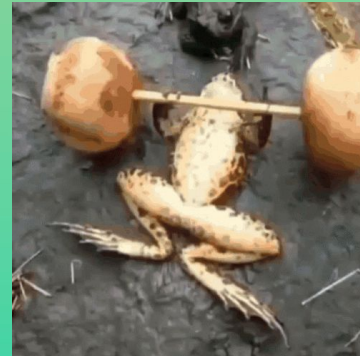
At a glance.

SNARKs	STARKs
Requires trusted set up	Doesn't require trusted set up
Not post quantum	Post quantum
Higher adoption = more dev tooling	Lower adoption
Smaller proof sizes, but slower verification than STARKs	Larger proof sizes, but faster verification than SNARKs
Good for both large and smaller data sets	Better for large data sets, less efficient for smaller data sets

What does SNARK stand for

- A. Succinct Non-interactive Argument of Knowledge
- B. Short No-touch Association with Knowledge
- C. Some Nouns Are Kool

Quiz time + stretch sesh



Section 3

**Jargon that Makes You Go WTF.
Current Events. Twitter Definition
Arguments. You Get It.**



R

DIS ME RN.

Modular blockchains.

Monolithic blockchains do everything

- **Consensus:** Agreement process for validating and ordering transactions. (i.e., PoW, PoS)
- **Settlement:** Finalization of transactions, making them irreversible and secure.
- **Data Availability:** Ensures transaction data is accessible for verification.
- **Execution:** Running the actual code or transaction logic to determine the blockchain's new state.

Modular blockchains do a subset

- Rollups like Scroll handle execution
- Blockchains like Avail, Celestia, and EigenDA handle data availability

Smart contract rollups.

These are the **rollups you might know today**



Remember the L1 contracts? They:

1. Store rollup state
2. Track deposits/withdrawals
3. Monitor state updates
4. Handle verification

Key takeaway - smart contract rollups are **only the execution layers** and are completely separate from the consensus layer on Ethereum.

Enshrined rollups.



Enshrined rollups **integrate with the L1 at the consensus level** (i.e., they are not controlled by smart contracts, but rather the rollup logic lives in the L1 itself)

Pros

1. No need for the security councils/multi-sigs/bridges requisite for L2 upgrades
2. For a zkEVM enshrined rollup, validators no longer have to re-execute all tx to validate a block (just verify proof)

Cons

1. VM inflexibility
2. Slow upgrades, complexity

Example of an enshrined zkRollup: SNARK verifier is embedded in core protocol and parallel zk-rollups use that as its verifier

Based rollups.



Based rollups **use the L1 for sequencing**, unlike rollups that use a separate (likely) centralized sequencer.

Pros:

- Decentralization
- Inherits the liveness of the L1
- Simpler design

Cons

- Limited flexibility in performance improvement techniques involving sequencing
- Lose out on MEV to the L1

Sovereign rollups.

Nodes of sovereign rollups **choose their own execution and settlement layers** (as opposed to a sequencer in L2s reliant on a smart contract on Ethereum for settlement)

As an example:

1. SC rollup upgrades are forced as they rely on a SC upgrade on the L1
2. Sovereign rollup upgrades **fork like a traditional L1** - where nodes choose to opt in/out of upgrades

Think of it like its own L1 that posts its data elsewhere (Celestia, Ethereum, Bitcoin)

Section 4

Some other niche hot topics.

Trusted Execution Environments (TEE).

TEEs are a secure area within a computer's processor where you can run code without it being accessible by other parts of the system - hardware

1. **Isolation:** TEEs create a secure "enclave" within the main processor. Only specific, pre-authorized code can run within this enclave, keeping it safe from outside access.
2. **Attestation:** TEEs can produce an "attestation" that proves to others that the code running within the TEE is genuine and unmodified. This attestation confirms that computations were securely executed.
3. **Confidentiality:** Data and computations inside the TEE remain encrypted and inaccessible to other parts of the system, including the operating system

Trusted Execution Environments (TEE).

Widespread use

- Intel SGX – most popular
- AWS nitro

Supports use-cases like:

- Privacy
- MEV protection
- Multi-provers

Trusted Execution Environments (TEE).

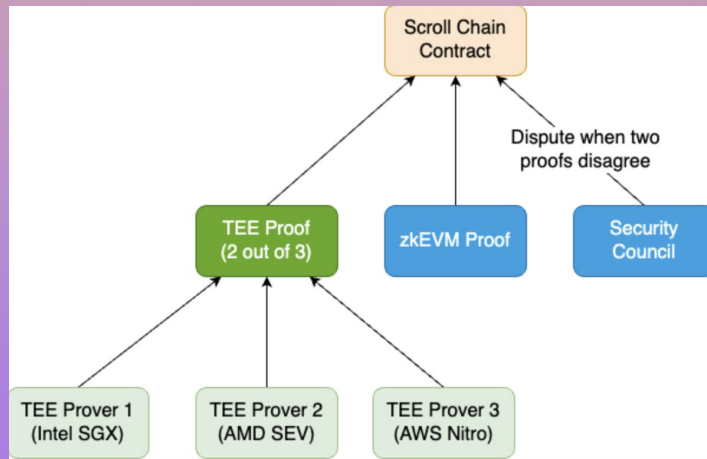
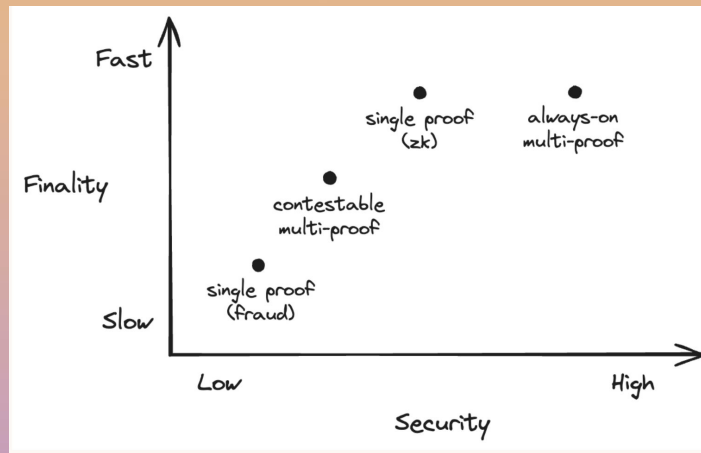
Aspect	Pros	Cons
Security	Hardware-secure environment, isolating sensitive data and code.	Vulnerable to hardware-specific attacks; a breach in the TEE compromises security.
Performance	Allows fast computation due to hardware acceleration, reducing computational load.	Dependent on specialized hardware, limiting accessibility and scalability in decentralized settings.
Integrity Guarantees	Offers secure attestation to prove the code was executed as intended in the secure enclave.	Requires trust in hardware manufacturers (e.g., Intel, AMD) and their TEE implementation, potentially limiting decentralization.

Multi-provers.

Singular proof systems are a vulnerability – hence, we need multi-prover systems to protect against dishonest chain operators

What are options for multi-provers

- Fraud proofs – well researched but adds to finality time
- 2nd zkEVM prover – resource expensive endeavor
- TEE – runs tx in a secure enclave and generates a proof





But wait.
Are zkEVMs end game?

Development of zkVMs.

A zkVM is a more **general-purpose virtual machine** that **enables computations using zero-knowledge proofs**, without being limited to Ethereum's specific design

A zkVM is **designed to support a broader range of applications, programming languages, and platforms** - i.e., Rust

The logo for RISC ZERO, featuring the words "RISC" and "ZERO" stacked vertically in a white, sans-serif font on a black rectangular background.

RISC
ZERO

The logo for Succinct, featuring a stylized icon of three interlocking shapes to the left of the word "Succinct" in a black, sans-serif font, all contained within a white rectangular background.

 Succinct

What this allows for

- Pathway for optimistic rollups to start using ZK proofs
- Accelerate ZK development (write in mature languages like Rust)
- Auditing surface is easier vs. custom circuits

With developments like Zeth and SP1, zkVMs are an interesting design space to follow



Dive Deeper.

- **Rollup stuff**

- Enshrined rollups
 - <https://www.youtube.com/watch?v=tOWHr8lqmjg>
 - <https://www.reddit.com/r/ethereum/comments/vrx9xe/comment/if7auu7/>
- S-tier summary of everything
 - <https://members.delphidigital.io/reports/the-complete-guide-to-rollups#rollup-fees-1581>
- Sovereign rollups
 - https://www.youtube.com/watch?v=cx4CsmOKK_g
 - <https://celestia.org/learn/sovereign-rollups/an-introduction/>
- Based rollups
 - <https://unchainedcrypto.com/based-rollups/>
- Stages
 - <https://medium.com/l2beat/introducing-stages-a-framework-to-evaluate-rollups-maturity-d290bb22befe>

Dive Deeper.

- ZK stuff
 - <https://chain.link/education-hub/zero-knowledge-proof-use-cases>
 - <https://blog.thirdweb.com/zkevm/>
- TEEs and Multi-prover
 - <https://www.youtube.com/watch?v=Xq7oWtiwWII>
 - <https://scroll.io/blog/scaling-security?ref=web3builder.news#what-is-the-design-space-of-a-multi-proof-system->
- zkVMs
 - <https://risczero.com/blog/designing-high-performance-zkVMs>

Thank you!



Emily

Devrel Lead, Scroll

emily@scroll.io

[@_emjlin](#)