Lesson 8 - Cairo / Starknet

NEW Block Explorer - Starkscan

Starkscan

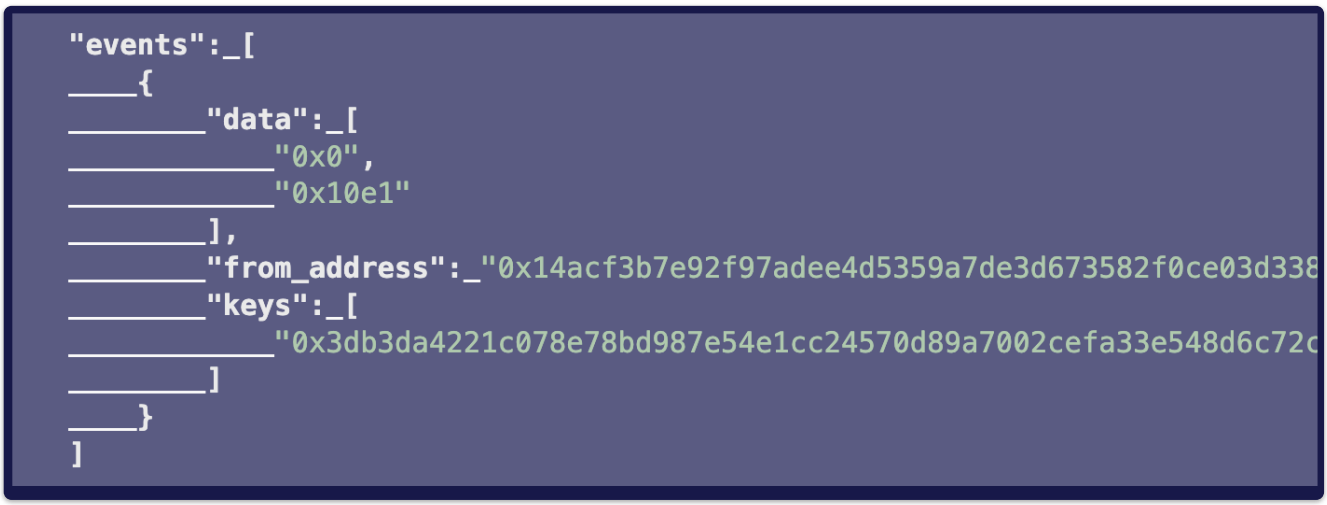
Latest Changes to Starknet

See docs

Events

Documentation

Events for transactions that have a state of at least PENDING will have event information in their transaction receipt.



the starknet CLI gives you access to the transaction receipt

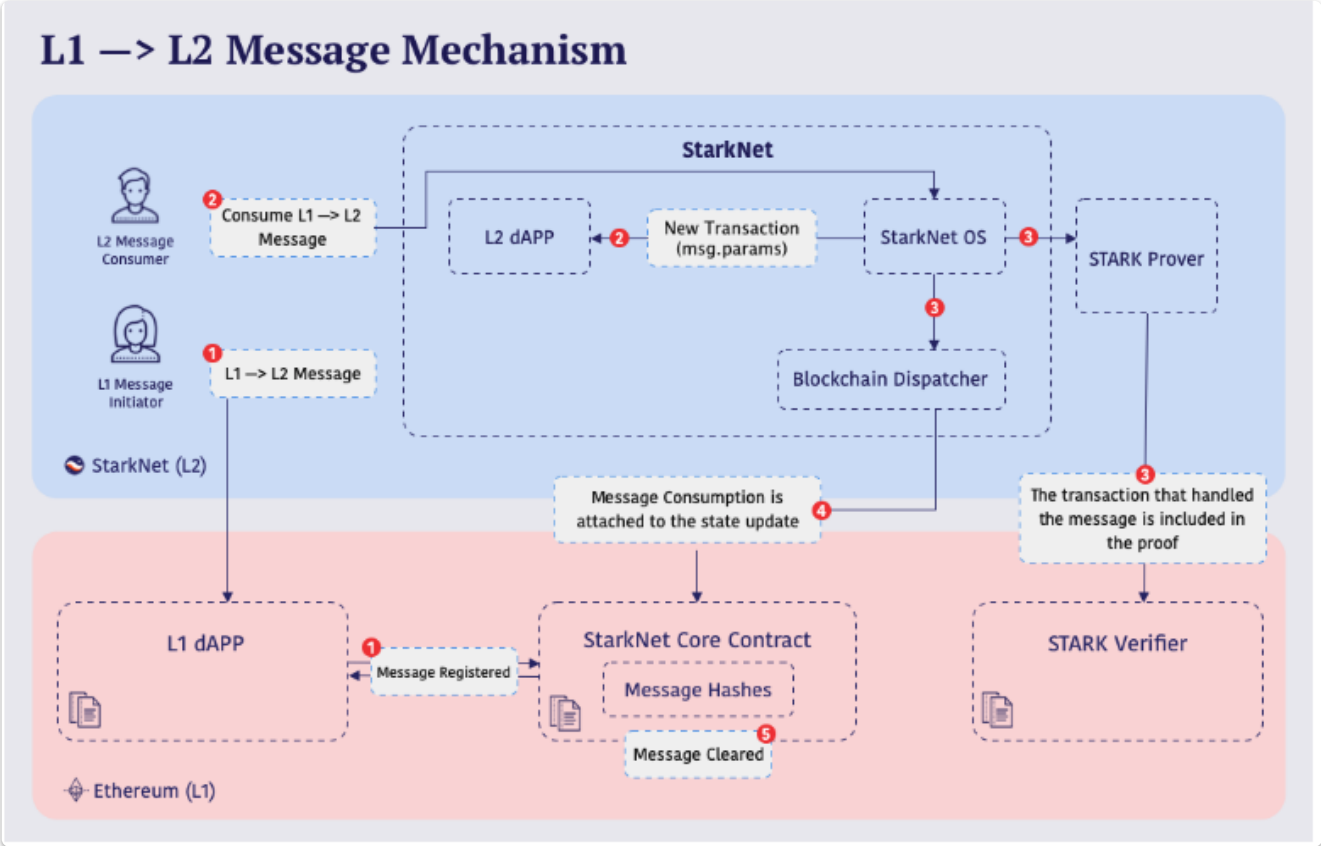
```
starknet get_transaction_receipt --hash ${TRANSACTION_HASH}
```

Support for querying events is limited.

Apibara

This project is similar to the Graph and can be used to index starknet events.
There is a tutorial here to show how to index events for an NFT

L1 to L2 Messaging

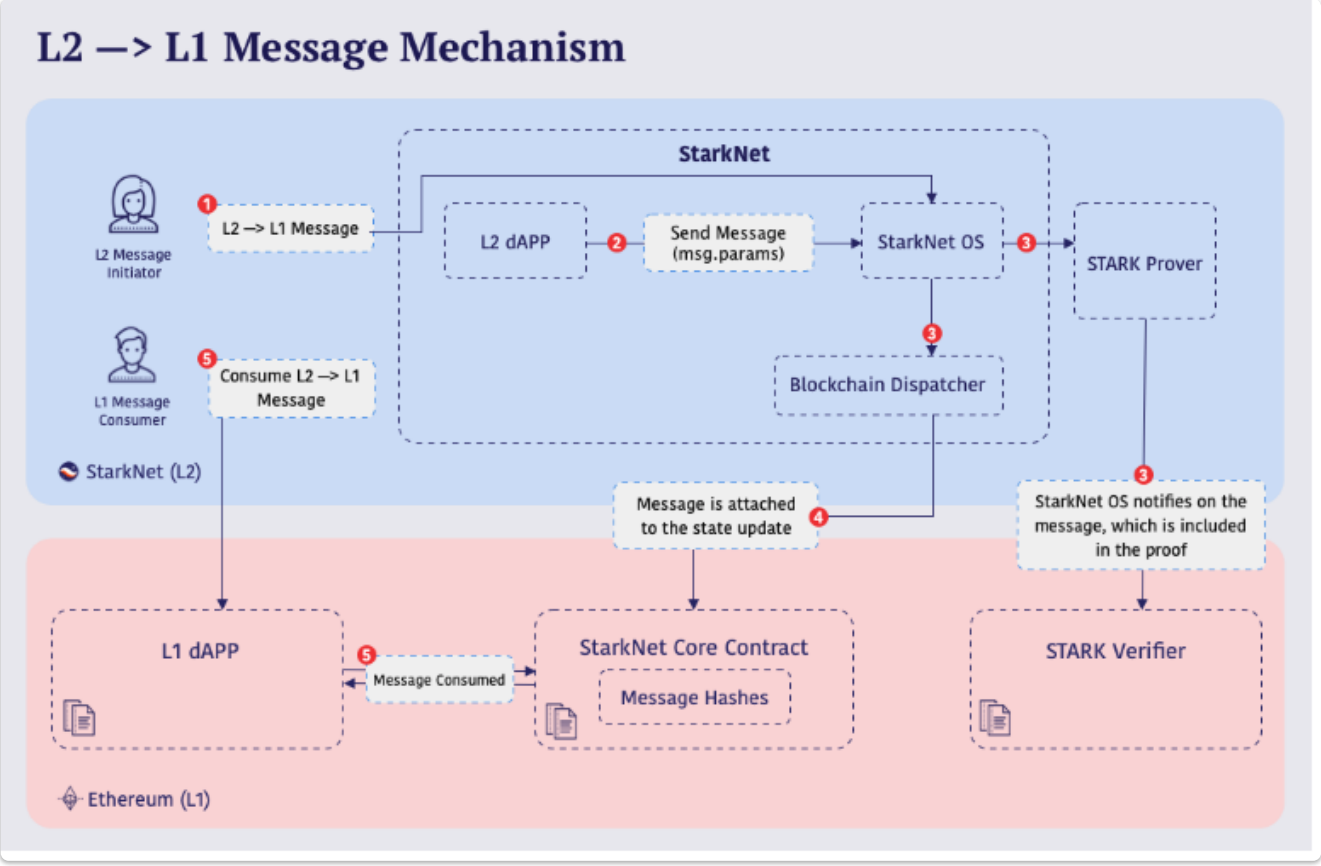


1. The L1 contract calls (on L1) the `send_message()` function of the StarkNet core contract, which stores the message. In this case the message includes an additional field - the "selector", which determines what function to call in the corresponding L2 contract.
2. The StarkNet Sequencer automatically consumes the message and invokes the requested L2 function of the contract designated by the "to" address.

This direction is useful, for example, for "deposit" transactions.

Note that while honest Sequencers automatically consume L1 -> L2 messages, it is not enforced by the protocol (so a Sequencer may choose to skip a message). This should be taken into account when designing the message protocol between the two contracts.

L2 to L1 Mesaging



Messages from L2 to L1 work as follows:

1. The StarkNet (L2) contract function calls the library function `send_message_to_l1()` in order to send the message. It specifies:

    1. The destination L1 contract ("to"),
    2. The data to send ("payload")

    The StarkNet OS adds the "from" address, which is the L2 address of the contract sending the message.

2. Once a state update containing the L2 transaction is accepted on-chain, the message is stored on the L1 StarkNet core contract, waiting to be consumed.

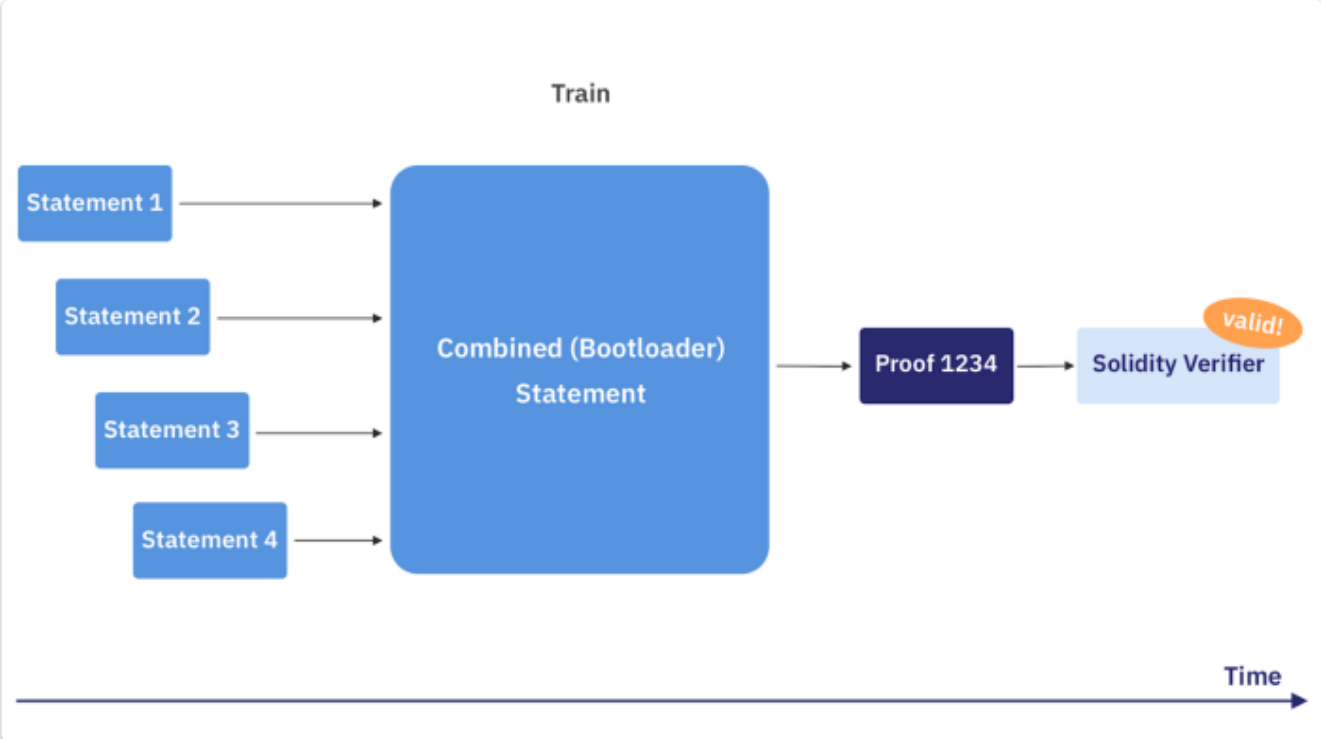
3. The L1 contract specified by the "to" address invokes the `consumeMessageFromL2()` of the StarkNet core contract.

Note: Since any L2 contract can send messages to any L1 contract it is recommended that the L1 contract check the "from" address before processing the transaction.

Recursive STARKS

See post

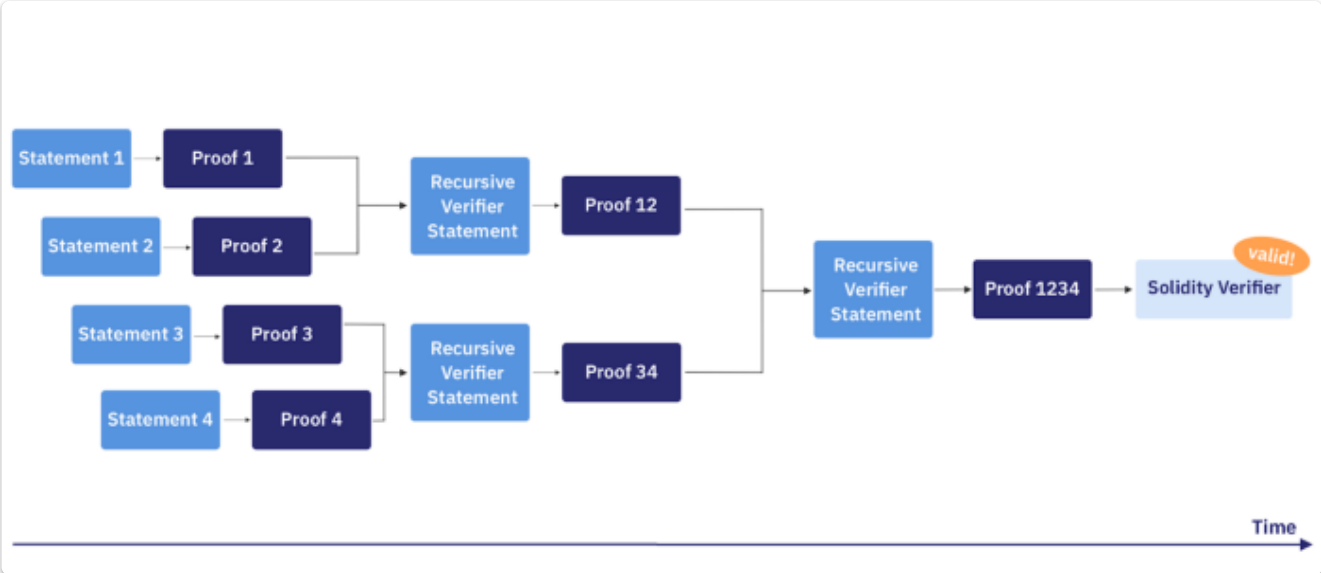
Initially SHARP (The shared prover) would process proofs from applications sequentially, oncce a threshold number of tranactions had arrived, a proof would be generated for all of them.



The amount of memory needed to to generate the proof was a limiting factor.

STARKs have roughly linear proving time and log validation time.

Recursive proofs



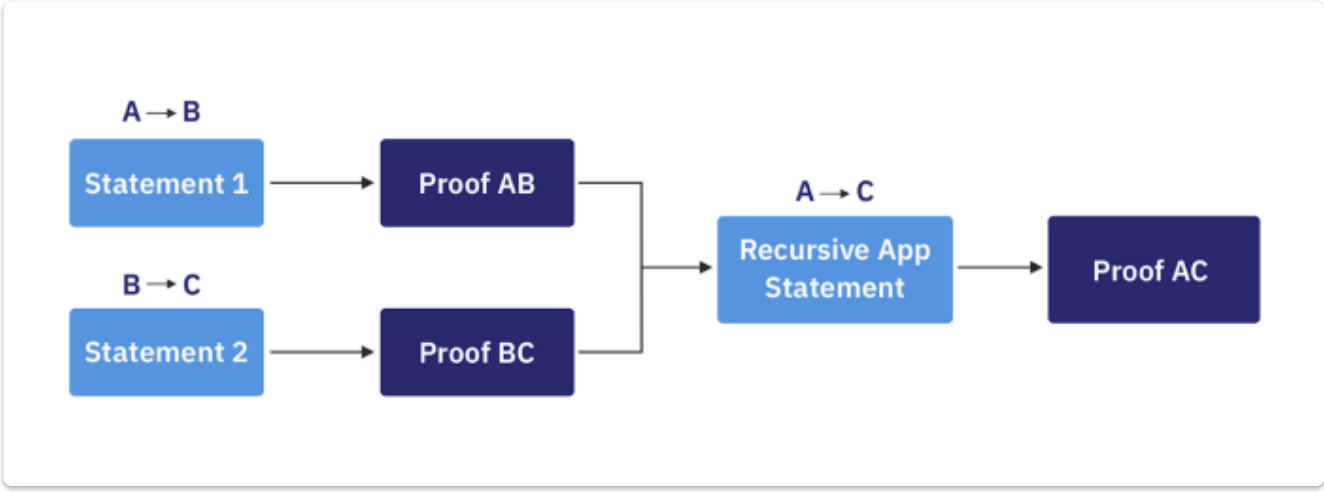
Here the proofs are calculated in parallel, then combined in pairs and a proof created and so on.

This results in

1. Reduced on chain cost, and memory requirements
2. Reduced latency since the proofs can be computed in parallel and we don't need to wait for the final proof to arrive.

Application recursion

Each STARK proof attests to the validity of a statement applied to some input
STARK recursion compresses two proofs with *two* inputs into *one* proof with two inputs. In other words, while the number of proofs is reduced, the number of inputs is kept constant. If the recursive statement is allowed to be *application-aware*, i.e. recognizes the semantics of the application itself, it can both compress two proofs into one *as well as* combine the two inputs into one.



Games on starknet

"I define the on-chain gaming thesis as this: **on-chain games will be the frontier for technological advancement of blockchain for the next five years.** Web3 game developers like MatchboxDAO are building the toolkit for future consumer-facing Dapps."

See Matchbox DAO blog
Matchbox DAO repo

Solve2Mint

Solve 2 Mint is a framework for NFT emission where each NFT maps to a unique solution to an equation or puzzle.

Topology

Topology are also very involved in this area.
Their philosophy is that usually the game developers are 'gods' but with blockchain and zkps, we can prevent this and allow a common exploration of problems.
They have maxims such as "Compute more, store less" which makes economic sense on a layer 2.
They also believe that humans are at a disadvantage to bots, since humans need to go through devices to interact with a game.
They think that games should be designed around NP-hard problems so that an algorithm cannot be written to solve it.
They also advise to add randomness from the user, because if the game is deterministic, then someone could simulate theh future off chain.

Game interaction has been helped by account abstraction and the use of session keys for example with the Argentx wallet.

Fountain

A 2-dimensional physics engine written in Cairo
repo
See tweet

Roll Your Own

Roll Your Own - A Dope Wars open universe project.

A modular game engine architecture for the StarkNet L2 roll-up, where computation is cheap and new game styles are being explored. Roll your own module and join the ecosystem.

Influence

Influence is an immersive, persistent space strategy game where players compete through multiple avenues, including: mining, building, trading, researching, and fighting, and interact in a 3D universe through third-person control of their ships and the installations they build.

Advantages using starknet

1. High Scalability & Low Fees
   Moving to Starknet, they expect a reduction of 100X in fees
2. Instant Actions
   Actions are regarded as final once they are submitted to the sequencer.
3. More complex transactions at the same price

Loot Realms

Advantages

1. Cheap Fees
2. Heavy computation possible
3. Composability of game items and liquidity

Dope War

Design

Physics Engine on Starknet

Video

Also Physics puzzle

Dark Forest (Not on Starknet)

Dark Forest is an massively multiplayer online real-time strategy (MMORTS) space conquest game build on top of the Ethereum blockchain.

From a medium article describing the game

Dark Forest game V0.3 uses zero-knowledge proof technology to prove 2 operations regarding the planet's location:
1/ planet initialization (init)
2/ planet movement (move).
The Circuit logic is implemented in darkforest-v0.3/circuits/.
They are implemented with circom, and the circuit proof uses Groth16.

Circuits used in Dark Forest

init Circuit

init Circuit ensures the coordinate falls in certain range during the creation of planet. Both x and y coordinates cannot exceed 2^{32}.

mimc(x,y) hash calculates on the coordinates.
x/y is private input, while hash value is public input.

move Circuit

During the planet's movement, the move circuit checks the moving range does not exceed a circular area of radius distMax:

We need to consolidate the hash value of both original coordinates and post-moving coordinates.

Dark Forest uses Circom to create its proofs.
Circom is a novel domain-specific language for defining arithmetic circuits that can be used to generate zero-knowledge proofs. `Circom compiler` is a circom language compiler written in Rust that can be used to generate a R1CS file with a set of associated constraints and a program (written either in C++ or WebAssembly) to efficiently compute a valid assignment to all wires of the circuit.

See circuits

- `/circuits/init`: Proof for initializing a player into the universe
- `/circuits/move`: Proof for initiating a move between two planets
- `/circuits/reveal`: Proof for broadcasting/revealing the coordinates of a planet. Note that nothing in the broadcast action needs to happen in "zero-knowledge"; we just

found it easier to implement verification of MiMC hash preimage via a ZK verifier than via a Solidity verifier.
- `/circuits/biomebase`: Proof that a planet has a given `biomebase`, which in combination with the planet's `spacetype` will specify the planet's biome.

There are two additional subdirectories for auxiliary utility circuits:

- `/circuits/perlin`: Perlin Noise ZK Circuit.
- `/circuits/range_proof`: Proof that an input, or list of inputs, has an absolute value that is at most a user-provided upper bound.

There are some play to earn features, you can earn DAI for revealing your planets locations.
