Ethereum Rollups

Saravanan Vijayakumaran

Department of Electrical Engineering
Indian Institute of Technology Bombay

April 8, 2024

Scaling Ethereum

- Each operation on Ethereum costs some gas
  - Minimum transaction gas cost = 21,000 gas
  - ETH transfer = 21,000 gas; ERC-20 transfer = 65,000 gas
- Ethereum block gas limit = 30 million gas
  - One Ethereum block can accommodate 1,428 ETH transfers
- Average inter-block time = 12 seconds
  - Ethereum can support 119 ETH transfers per second
  - Actual throughput is 15 txs/sec
- High demand for ETH block space = High transaction fees
- **Challenge:** How to increase Ethereum throughput?
- Previous attempt: **Plasma**
  - Move computation and state off-chain; only state roots stored on-chain
  - Needed trust on plasma operator to guarantee **data availability** of off-chain state
  - Limited market adoption
- **Rollup**
  - Move computation and state off-chain (like Plasma)
  - Data needed to reconstruct off-chain state is posted on Ethereum as calldata or blobs
  - Has mechanism to ensure correctness of state roots that are stored on-chain
    - Fault proofs or validity proofs
  - User experience is slightly degraded (more latency and/or extra on-boarding steps)

Data Locations in Ethereum Contracts

- Three types of data locations
  - **storage**: Contract state variables that persist for contract lifetime
  - **memory**: Variables that persist only during a contract method execution
  - **calldata**: Read-only locations containing function arguments

```
contract DataLocations {
    // storage array
    uint[] public arr;

    // memory array in function argument
    function g(uint[] memory _arr) public returns (uint[] memory) {
        // do something with memory array
    }

    // calldata array in function argument
    function h(uint[] calldata _arr) external {
        // do something with calldata array
    }
}
```

- Gas costs of different types of data
  - **storage**: Upto 690 gas per byte
  - **memory**: Scales quadratically with number of 32 byte words
  - **calldata**: 16 gas per non-zero byte and 4 gas per zero byte
- Calldata is much cheaper than storage for storing state on Ethereum
- Contract methods can only access the calldata of the current call, not past calls
- Rollups store only essential state in storage and the full data in calldata

EIP-4844 aka Proto-Danksharding

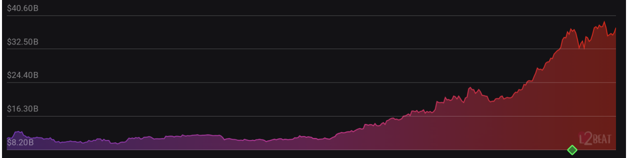
- In March 2024, EIP-4844 was activated on Ethereum
- Users can post a blob of data of size approx 125 KB along with a block
- Blob = 4096 field elements of 32 bytes each
- Upto 6 blobs per block allowed
- Data is only stored for 4096 epochs (18 days)
- Rollup operators can store their transaction data in blobs instead of calldata
- Assumes that blobs will be stored by the ecosystem if they are needed later

Total Value Locked in Ethereum Layer 2

Value Locked — **$37.41B**

Sum of all canonically bridged, externally bridged, and natively minted tokens, converted to USD — ▾ 3.50% / 7 days

2023 Apr 10 – 2024 Apr 08 — 7D 30D 90D 180D 1Y MAX



USD ETH — LOG LIN

Rollups only · Type · Stack · Stage · Purpose

Active projects 24 · Upcoming projects 26 · Archived projects 8 · Layer 3 projects 3

#	NAME	RISKS	TYPE	STAGE	PURPOSE	TOTAL	MKT SHARE
1	Arbitrum One		Optimistic Rollup	STAGE 1	Universal	$18.82B ▾ 4.57%	45.61%
2	OP Mainnet		Optimistic Rollup	STAGE 0	Universal	$7.72B ▾ 7.80%	18.71%
3	Base		Optimistic Rollup	STAGE 0	Universal	$4.15B ▴ 13.72%	10.07%
4	Blast		Optimistic Rollup	STAGE 0	Universal, DeFi	$2.77B ▾ 1.16%	6.73%
5	Starknet		ZK Rollup	STAGE 0	Universal	$1.39B ▾ 8.56%	3.39%
6	zkSync Era		ZK Rollup	STAGE 0	Universal	$752M ▾ 3.40%	1.82%

Source: `https://l2beat.com/scaling/summary`

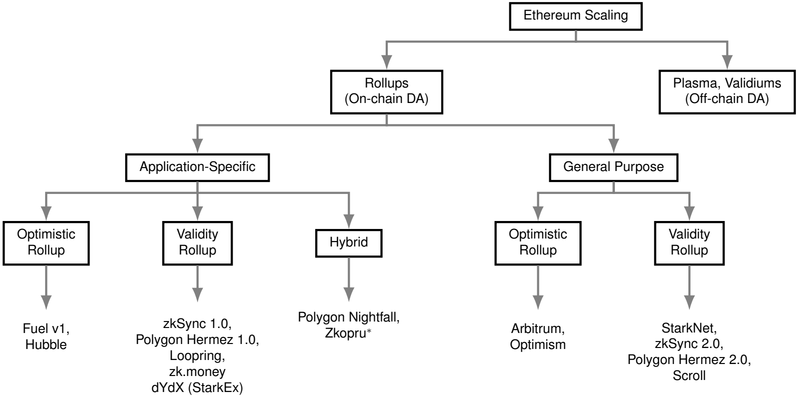
Fees in Ethereum Layer 2

Name	Send ETH	Swap tokens
Optimism	< $0.01	< $0.01
Arbitrum One	< $0.01	< $0.01
zkSync Era	< $0.01	-
Loopring	$0.03	$0.86
zkSync Lite	$0.03	$0.08
DeGate	$0.08	$0.20
Polygon zkEVM	$0.08	$0.29
Boba Network	$0.12	-
Ethereum	$2.12	$10.60

Source: `https://l2fees.info/`

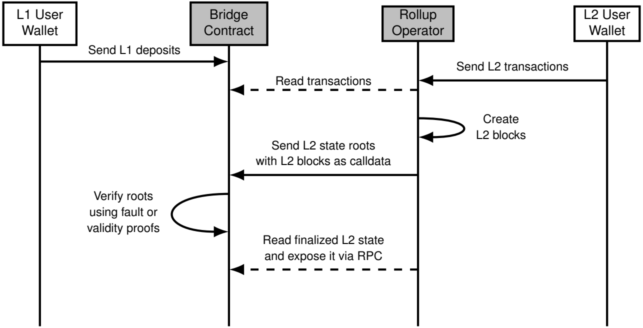
Ethereum Scaling Solutions

Rollups



- Application-specific rollups have limited functionality on L2, like ETH/ERC20 transfers
- General purpose rollups support arbitrary smart contracts on L2

Main Rollup Components



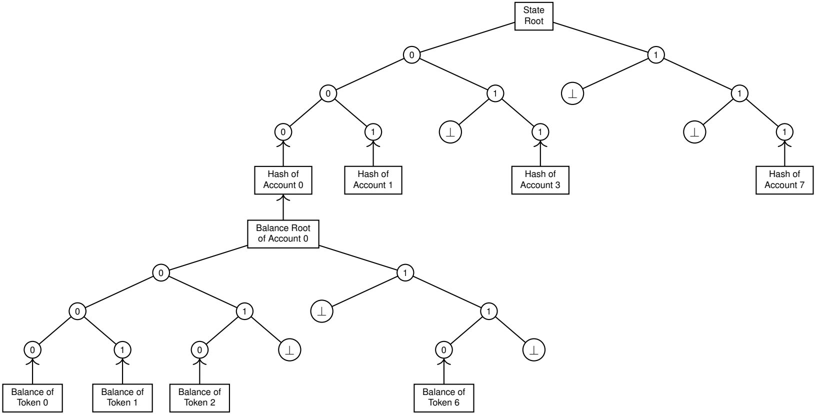
- **Rollup operator**
  - Also called **sequencer** or **validator**
  - Exposes RPC endpoint for accepting L2 transactions; no P2P network on L2
  - Reads or receives L2 transactions and produces L2 blocks
  - Monitors the bridge contract for L1 asset deposits and mints them on L2
- **Bridge contract**
  - L1 smart contract for coordinating asset movement between L1 and L2
  - Receives L2 transactions as calldata, stores sequence of L2 state roots
  - Facilitates verification of state roots using fault or validity proofs

Layer 2 State

- On L2, the rollup operator maintains a blockchain of L2 transactions
- **General-purpose rollups**
  - L2 state includes
    - the set of all L2 accounts and their token balances
    - the set of all contracts installed on L2, their code and storage
  - L2 state root is the hash of the entire L2 state
    - *Example*: In Optimism, the L2 state is maintained by a modified version of geth.
      L2 state root = Root hash of world state trie
- **Application-specific rollups**
  - L2 state needs to only express application state
  - Consider an application that supports token transfers
  - If application is account-based, a Merkle tree of account balances is sufficient
    - L2 state root = Root hash of Merkle tree
    - *Examples*: zkSync 1.0, Hermez 1.0
  - If application is UTXO-based, the application state is the set of all UTXOs
    - The set of all L2 blocks is needed to determine the state
    - L2 state root = Hash of latest L2 block header
    - *Example*: Fuel v1

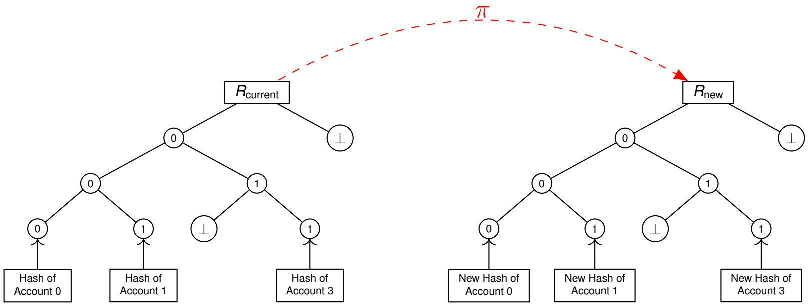
L2 State in zkSync 1.0

- A sparse Merkle tree (SMT) with account state hashes as leaves
- Account state
  - Root hash of an SMT holding token balances in its leaves (**balance root**)
  - 32-bit nonce
  - Ethereum address associated with account
  - Rescue hash of L2 public key (point on BN256 curve)



Verifying L2 State Updates in Validity Rollups

- Validity rollups use zero-knowledge proofs to prove correctness of L2 state updates
- Ethereum supports ZK proofs based on SNARKs or STARKs



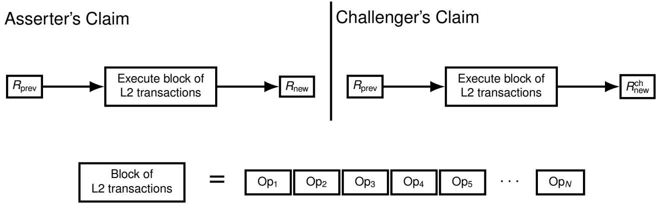
- Proof π proves correctness of L2 state root transition from R_{current} to R_{new}
- New root represents the state **after executing a block** of L2 txs
- Proofs are generated off-chain and sent to bridge contract for on-chain verification

Verifying L2 State Updates in Optimistic Rollups

- Each state root submitted to bridge contract is accompanied by an ETH deposit
  - For example, in Arbitrum the base deposit is 5 ETH
- If a state root is proved to be faulty, the submitter loses their deposit
- A successful fault prover gets half the deposit and the other half is burnt
- Once a state root is proved faulty, it and its successors are marked invalid
- If half the deposit was not burnt, malicious parties could delay L2 chain progress at no cost
- AS and GP optimistic rollups: **Different mechanisms** for proving faults
- **Application-specific optimistic rollups**
  - State root can be faulty in a small number of ways which can be exhaustively enumerated
  - State root faultiness can be proved using a small number of L1 transactions
  - **Example**: Fuel v1 is a payments-only optimistic rollup
    - Faulty state roots can be due to double spending, invalid input, malformed block, and a few other cases
    - Two L1 transactions required to prove faults
    - First L1 tx posts only hash of the fault proof to prevent frontrunning
    - Second L1 tx posts the actual fault proof

Verifying L2 State Updates in Optimistic Rollups

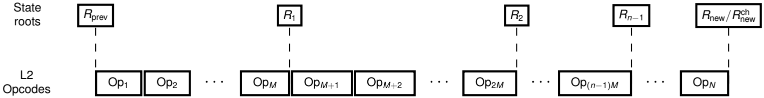
- **General-purpose optimistic rollups**
  - General-purpose rollups support arbitrary contracts
  - **Challenge**: Infeasible to enumerate all possible ways in which a state root can be faulty
- The setup
  - **Asserter** = Party posting a new state root R_{new} to the bridge contract
  - **Challenger** = Party challenging the correctness of R_{new}
  - Asserter and challenger agree on R_{prev}, the predecessor of R_{new}
  - Challenger claims $R^{\text{ch}}_{\text{new}}$ should be the next root
  - Both agree on the L2 block of transactions, which is a sequence of L2 opcodes



- **Fault proof strategy**: Identify the first L2 opcode where fault occurs and prove the fault on-chain
- **Main Insight**: Number of L2 opcodes is limited; Can be exhaustively enumerated and simulated in an L1 contract

Interactive Game for Proving Faults

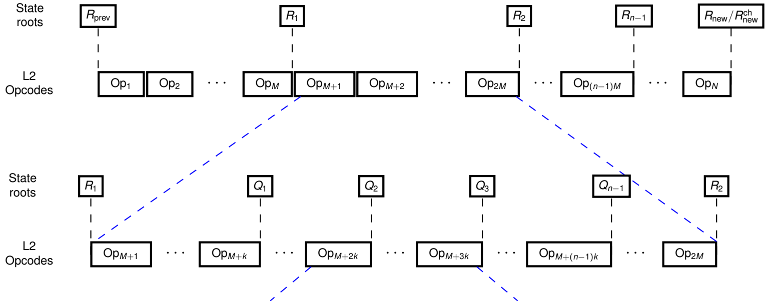
- The interactive game between asserter and challenger has two stages
  - *n*-ary search
  - one-step proof
- In the *n*-ary search stage, the **challenger** first publishes intermediate state roots $R_1, R_2, \ldots, R_{n-1}$ between R_{prev} and $R^{\text{ch}}_{\text{new}}$
- Root locations are chosen such that the amount of computation between consecutive intermediate roots is approximately the same



- Since $R_{\text{new}} \neq R^{\text{ch}}_{\text{new}}$, the asserter disagrees with the challenger in at least one root in the sequence $R_1, R_2, \ldots, R_{n-1}, R_n = R^{\text{ch}}_{\text{new}}$
- The **asserter** chooses first root R_i where it disagrees and publishes $n-1$ state roots between R_{i-1} and R_i

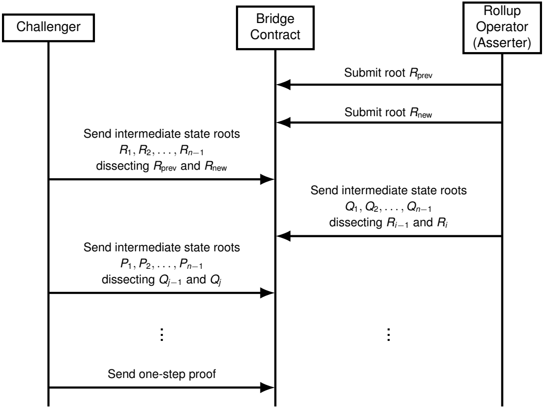
Interactive Game for Proving Faults

- The **asserter** chooses first root R_i where it disagrees and publishes $n-1$ state roots between R_{i-1} and R_i
  - Suppose R_2 is the first root where the asserter disagrees with the challenger
  - Asserter publishes roots $Q_1, Q_2, \ldots, Q_{n-1}$ between R_1 and R_2



- This dissection process **alternates** between the asserter and challenger
- At the end, a single L2 opcode is identified
  - Both parties agree on the input but disagree on the output of this opcode
- **One-step proof**: The opcode is re-executed in an L1 contract and the winner identified
- Losing party's deposit is confiscated and the winner is rewarded

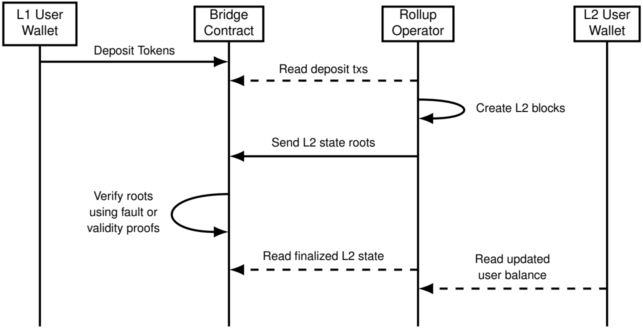
Interactive Game for Proving Faults



- The fault proof protocol involves multiple L1 transactions which could be censored by malicious miners
- Asserter and challenger are each given one week of time in a chess-style clock
- So the entire fault proof protocol can take upto two weeks
- A state root is considered finalized if it is not challenged for one week

Rollup User Experience (1/9)

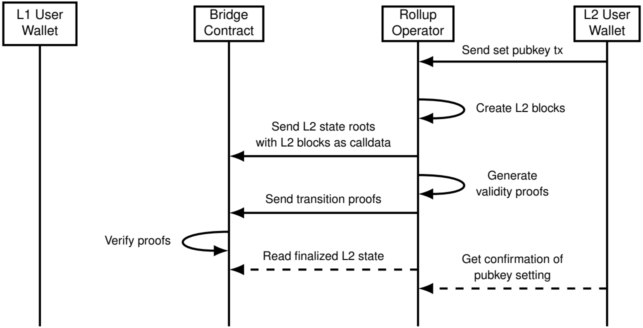
Depositing L1 tokens to L2



- Users send their assets to the bridge contract on L1
- Rollup operator monitors bridge contract for deposits
- For each L1 deposit, an L2 transaction will mint the asset for the user on L2
- Once a state root is verified in the bridge contract, assets appear on user's L2 wallet (typically in less than an hour)
  - While fault proofs have a 7-day waiting period, this does not apply to L1 deposits

Rollup User Experience (2/9)

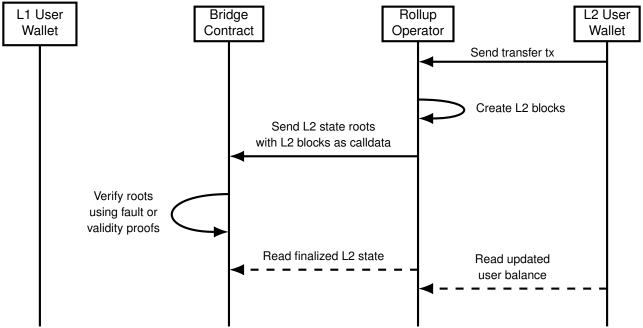
Setting the L2 Public Key in Validity Rollups



- This step applies only to validity rollups
- Validity rollups use zero-knowledge proofs to prove correctness of L2 state roots
- For efficiency reasons, some validity rollups require an L2 public key
    - Required in zkSync 1.0, Polygon Hermez 1.0
    - Not needed in zkEVMs (zkSync 2.0, Polygon Hermez 2.0)
- User sends an L2 tx to register the L2 pubkey (costs L2 fees)
- Once the L2 block containing the tx is verified on-chain, the public key change is confirmed

Rollup User Experience (3/9)

Transacting on L2



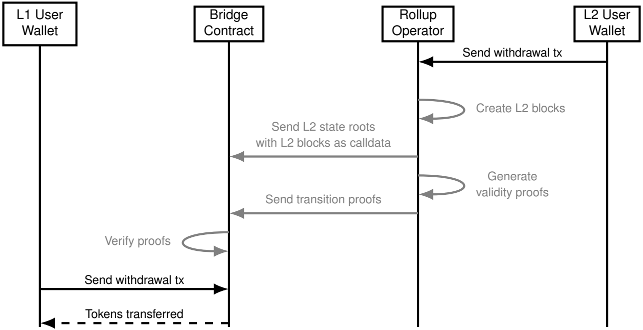
- Users send their L2 transactions to the operator's RPC endpoint
  - There is no P2P network on L2
- L2 users experience latency between transaction submission and finalization
  - The fundamental tradeoff of rollups: **lower cost for higher latency**
  - Validity and optimistic rollups have different L2 transaction finalization trust models and latencies

L2 Transaction Finalization Latency

- **Validity Rollups**
  - L2 transactions are finalized once proofs are verified on-chain
  - To amortize on-chain verification fees, several L2 state roots are verified together
  - zkSync 1.0 latency = 1 hour, StarkEx latency = 7 to 10 hours, Hermez 1.0 latency = 6 hours
- **Optimistic Rollups**
  - **1-of-*N* trust model**: At least one honest party exists that can submit a fault proof
    - L2 transactions are finalized if no challenges in the 7 days after submission
    - **Latency = 7 days**
  - **1-of-1 trust model:** User trusts the sequencer **or** user trusts a party that reads the sequence of L2 blocks submitted to bridge contract and calculates L2 state
    - The sequence of L2 blocks is frozen once the submitting transactions have enough confirmations
    - If the trusted party confirms correctness of submitted state roots, the user will accept them as final
    - *Example of trusted party*: L2 wallet provider
    - **Latency = few minutes** (L1 confirmation time of transactions submitting L2 state roots)
  - **Sequencer mode**: Only the sequencer is allowed to add L2 blocks and the user trusts the sequencer to submit only correct state roots
    - **Latency = a few seconds** (sequencer response time); No need to wait for L1 block containing L2 state root

Rollup User Experience (4/9)

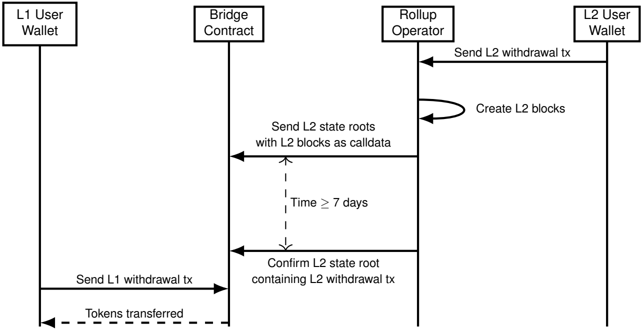
Withdrawals from L2 to L1 in Validity Rollups



- When a user wants to withdraw their assets back to L1, they send an L2 transaction to the operator requesting the withdrawal
- User waits for the L2 transaction to be finalized on L1
- User sends an L1 transaction to withdraw their assets from the bridge contract

Rollup User Experience (5/9)

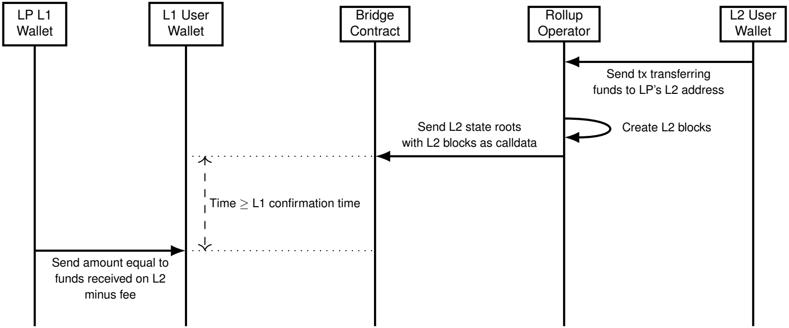
Withdrawals from L2 to L1 in Optimistic Rollups (without LPs)



- Two possible workflows: One without liquidity providers (LPs) and other with them
- Consider the no LP case first
- User sends an L2 transaction to the operator requesting the withdrawal to L1
- Operator posts a state root including the effects of the L2 withdrawal tx to the bridge contract
- If 7 days pass without a challenge, the state root is confirmed on L1
- User sends an L1 transaction to withdraw their assets from the bridge contract

Rollup User Experience (6/9)

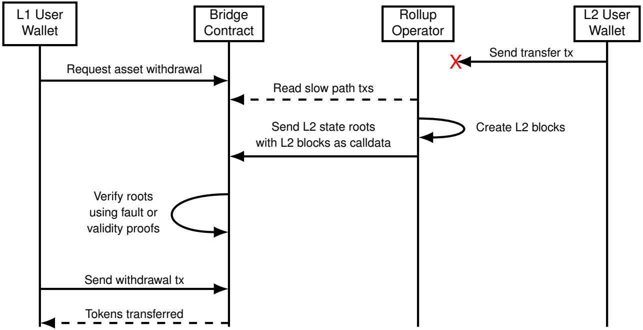
Withdrawals from L2 to L1 in Optimistic Rollups (with LPs)



- Liquidity providers speed up users' L2-to-L1 withdrawals for a fee
- Waiting times for withdrawals reduce from 7 days to a few minutes
- User sends their L2 assets to the LP's address on L2
- Operator posts a state root including the effects of the L2 transfer tx to the bridge contract
- LP waits (a few minutes) for the L1 transaction that submitted the state root to confirm
- LP sends the amount it received on L2 minus a fee to the user's L1 address

Rollup User Experience (7/9)

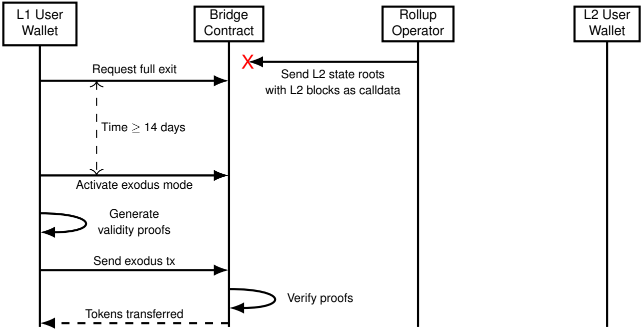
Censorship by Operator on L2



- Suppose the operator censors a user's L2 transactions at the RPC endpoint
- Rollups offer users a **slow path** to force include their L2 transactions
  - Arbitrum allows any arbitrary L2 tx on the slow path
  - zkSync allows only asset withdrawals
- Operator is forced to include slow path transactions within a deadline
- Once state root of withdrawal tx is finalized, users can withdraw via an L1 tx
- **Slow Path Cons**: Delays, L1 tx fees

Rollup User Experience (8/9)

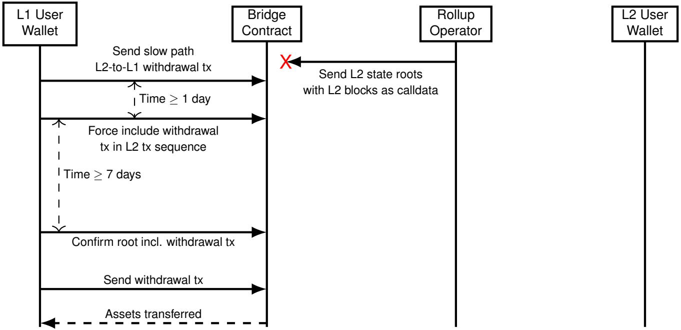
Dealing with an Offline Operator: Validity Rollups



- Suppose the operator goes offline and stops generating L2 blocks
- **Example**: zkSync 1.0
  - Users can request a full exit of their L2 assets
  - If exit tx has not been processed in 14 days, anyone can activate **exodus mode**
  - In exodus mode, normal rollup operations are not allowed (no deposits or withdrawals, no new L2 state roots added)
  - Users can withdraw their funds (perform exodus) by generating individual validity proofs of asset ownership and submitting them to the bridge contract

Rollup User Experience (9/9)

Dealing with an Offline Operator: Optimistic Rollups



- Suppose the operator goes offline and stops generating L2 blocks
- **Example**: Arbitrum
  - Users can request a full withdrawal of their L2 assets via the slow path
  - After a 1 day delay, withdrawal tx can be forced into the L2 tx sequence
  - After another 7 days, the L2 root containing the withdrawal tx can be confirmed (by user or anyone else)
  - User can then withdraw assets from bridge contract via an L1 tx

Effect of Miner Censorship or Congestion

- Ethereum miners could potentially censor transactions submitted to the bridge contract
- Congestion on Ethereum could also cause censorship
- If new L2 state roots are not submitted, L2 chain progress stalls
- **Validity Rollups**
  - Only **liveness** of L2 chain is affected; user funds on L2 are secure
  - Censorship by L1 miners is indistinguishable from an offline operator
  - Users can use fallback mechanisms to withdraw their L2 funds
- **Optimistic Rollups**
  - Miners could selectively censor transactions involving fault proofs
  - Both **liveness and security** are affected
  - If fault proof transactions are censored for 1 week, an invalid state root can be confirmed
    - Unlikely but possible
  - Operator would need to clone the bridge contract and restart operation from a correct state

References

- Rollups for Scaling Application-Specific Blockchains https://www.ee.iitb.ac.in/~sarva/reports/rollups_for_scaling_asbs.pdf
- L2 Beat https://l2beat.com/scaling/tvl/
- L2 Fees https://l2fees.info/
- Proto-danksharding https://ethereum.org/en/roadmap/danksharding/
- EIP 4844 https://eips.ethereum.org/EIPS/eip-4844
- Arbitrum https://arbitrum.io/
- Optimism https://www.optimism.io/
- zkSync L2 Block Explorer https://zkscan.io/
- Hermez 1.0 Block Explorer https://explorer.hermez.io/
- Starkware https://starkware.co/
- Starkware Verifier Transactions https://etherscan.io/address/0x47312450B3Ac8b5b8e247a6bB6d523e7605bDb60
- Polygon zkEVM https://docs.hermez.io/zkEVM/Overview/Overview/
- Scroll zkEVM https://scroll.io/
- Zkopru https://zkopru.network/