Lecture #15:
Rollups

COMS 4995-001:
The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

Bottlenecks to Scaling

Answer: load on validators.

- consensus responsibilities:
  - assembling a block (can be hard to do well, more later)
  - communication/bandwidth
  - computation (e.g., signature verification)

Bottlenecks to Scaling

Answer: load on validators.

- consensus responsibilities:
  - assembling a block (can be hard to do well, more later)
  - communication/bandwidth
  - computation (e.g., signature verification)
- execution responsibilities:
  - storing the blockchain state
  - repeated reads/writes to state

Bottlenecks to Scaling

Answer: load on validators.

- consensus responsibilities:
  - assembling a block (can be hard to do well, more later)
  - communication/bandwidth
  - computation (e.g., signature verification)
- execution responsibilities:
  - storing the blockchain state
  - repeated reads/writes to state
- storage responsibilities:
  - storing sequence of all processed txs

Approaches to Scaling

Category #1: impose constraints on the validator set.

Approaches to Scaling

Category #1: impose constraints on the validator set.

Category #2: better protocols and client implementations.

Approaches to Scaling

Category #1: impose constraints on the validator set.

Category #2: better protocols and client implementations.

Category #3: outsourcing validator responsibilities to 3rd parties.

- 3rd parties may be specialized, centralized, and largely untrusted

Approaches to Scaling

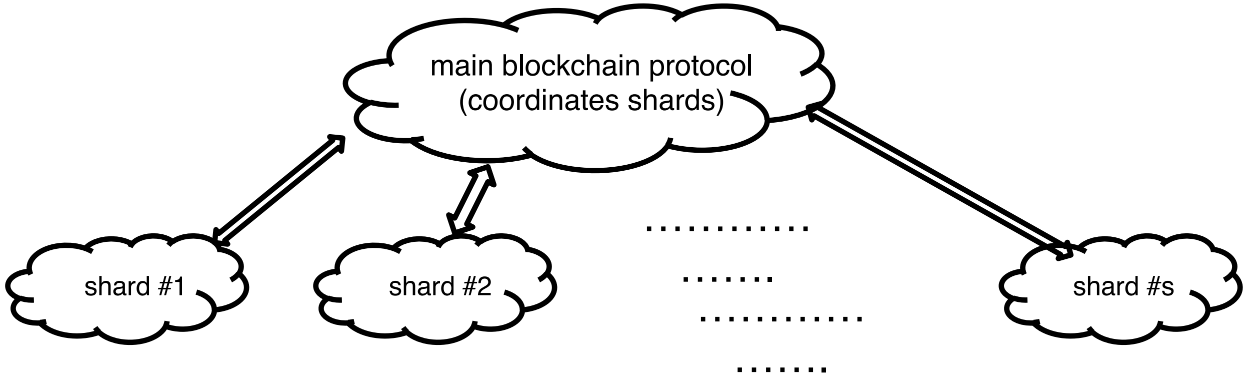
Category #1: impose constraints on the validator set.

Category #2: better protocols and client implementations.

Category #3: outsourcing validator responsibilities to 3rd parties.

- 3rd parties may be specialized, centralized, and largely untrusted

Category #4: "sharding"/horizontal scaling.



Goals for Lecture #15

1. Introduction to “rollups.”
   - an approach to sharding blockchain state and execution
   - piggyback on an “L1” for data availability, liveness, etc.
   - central to the Ethereum ecosystem

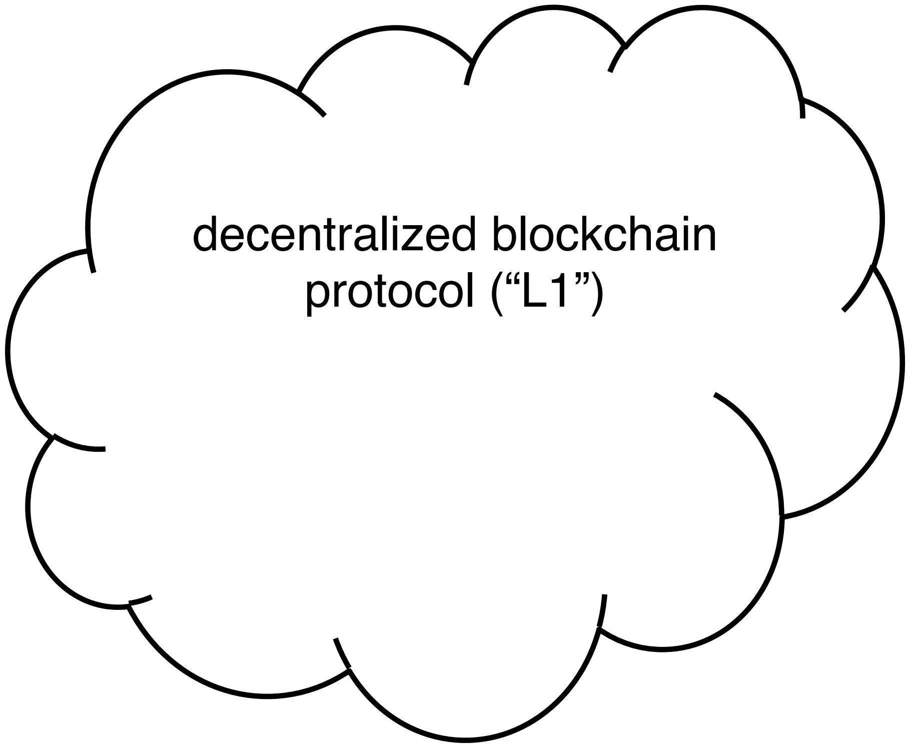
2. EIP-4844.
   - modern solution to DA required by rollups: “blob” storage

3. Optimistic rollups. (e.g., Arbitrum, Base, Optimism)
   - rollup state commitments verified via “bisection game”
   - security derived from economic penalties (confiscated collateral)

Introduction to Rollups

Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

L1 ⇔ Rollup Architecture



Introduction to Rollups

Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

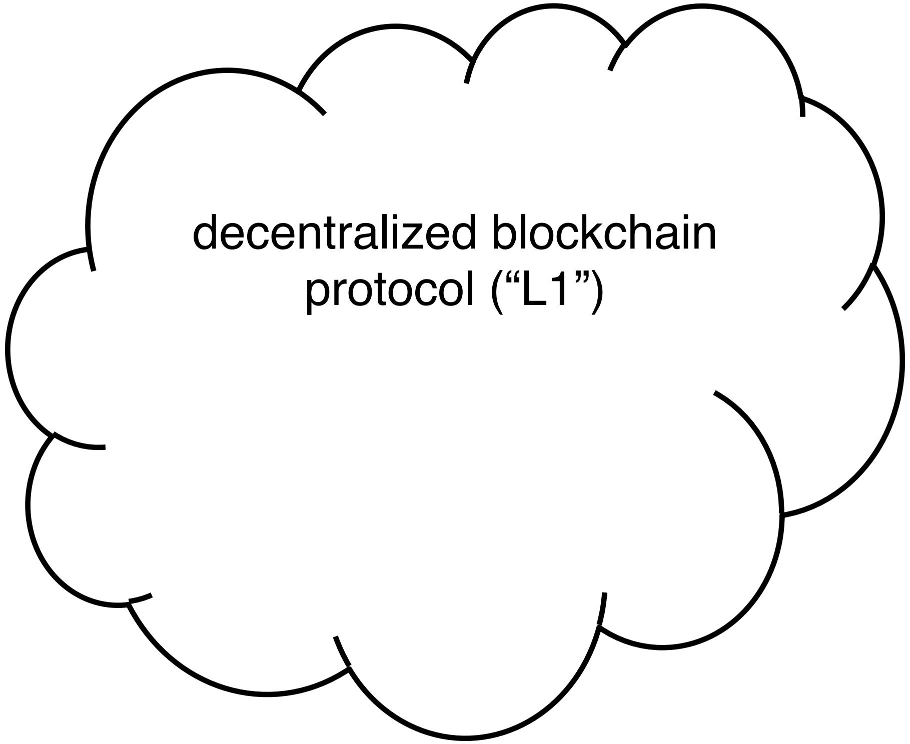
“Classic” rollup: a blockchain/virtual machine with its own state

– not necessarily decentralized, subject to crash or Byzantine failure

– performs its own consensus (i.e., tx sequencing) and execution

L1 ⇔ Rollup Architecture

(possibly centralized) rollup



Introduction to Rollups

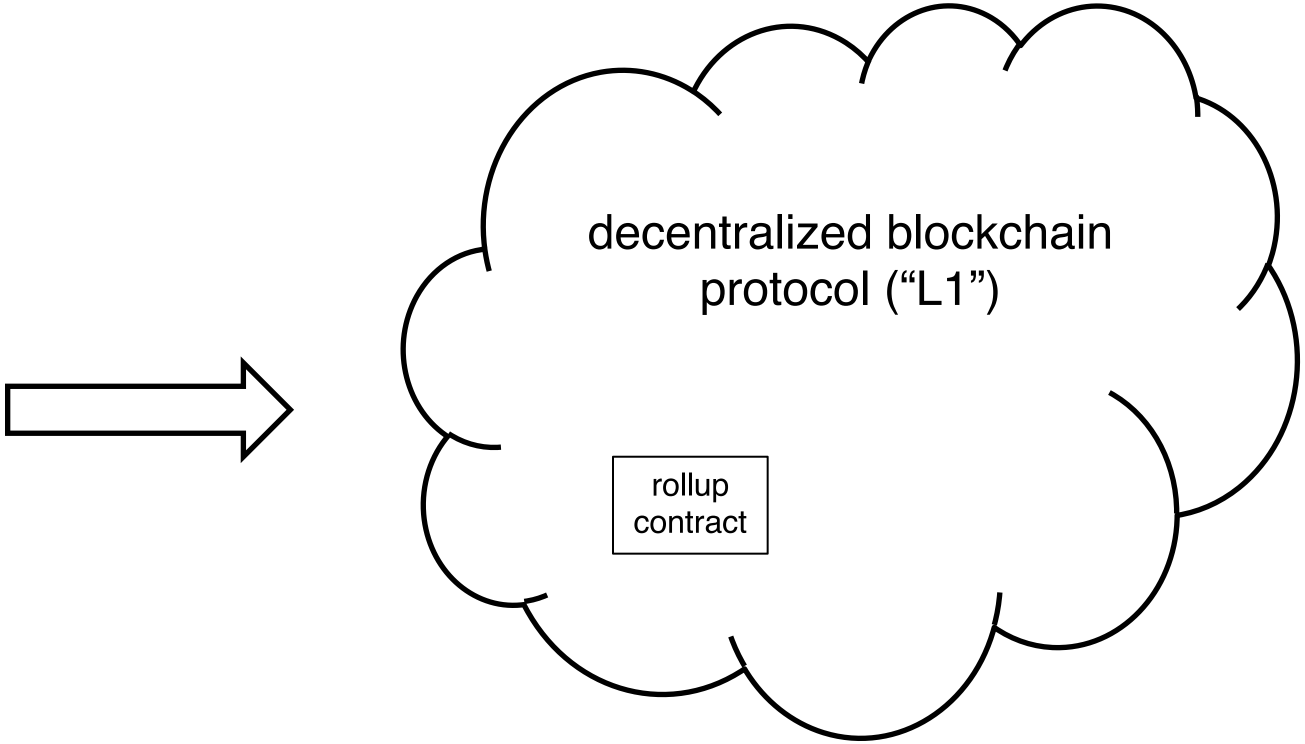
Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

“Classic” rollup: a blockchain/virtual machine with its own state

– not necessarily decentralized, subject to crash or Byzantine failure
– performs its own consensus (i.e., tx sequencing) and execution
– associated with smart contract(s) running on the L1

L1 ⇔ Rollup Architecture

(possibly centralized) rollup



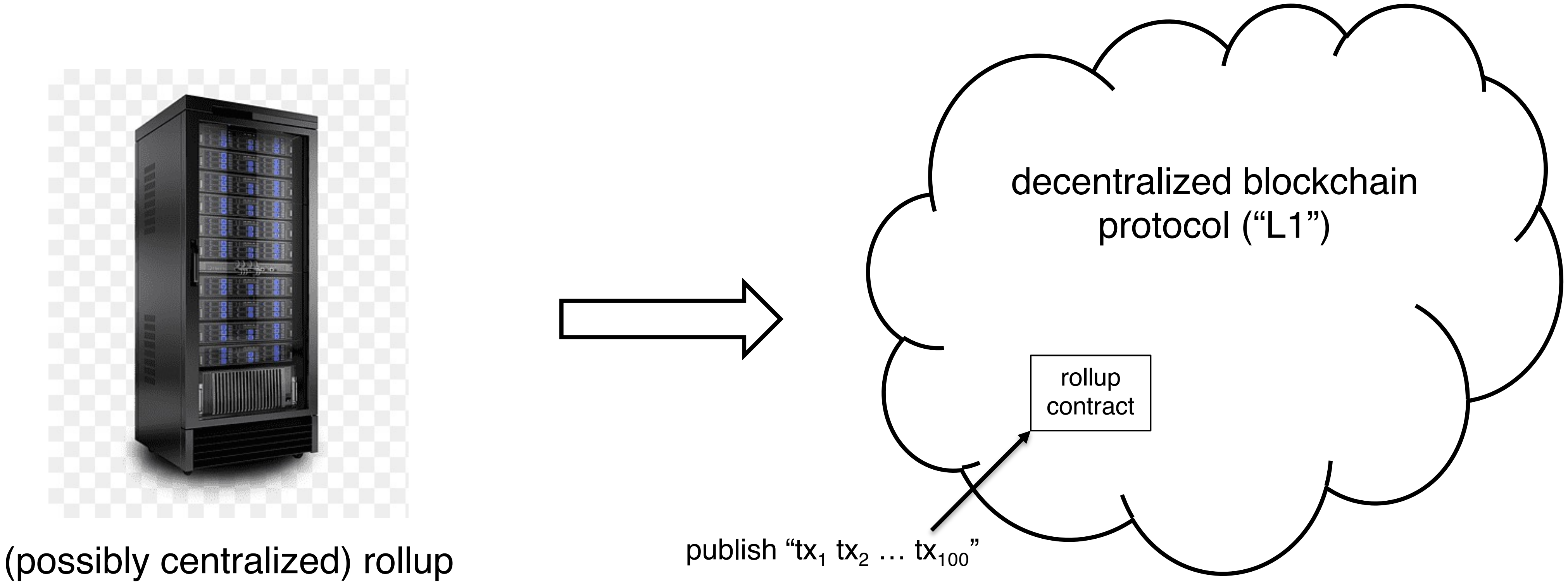
Introduction to Rollups

Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

“Classic” rollup: a blockchain/virtual machine with its own state

- not necessarily decentralized, subject to crash or Byzantine failure
- performs its own consensus (i.e., tx sequencing) and execution
- associated with smart contract(s) running on the L1
- publishes rollup txs via L1 contract (i.e., uses L1 for data availability)
    - note: anyone can run a rollup full node (i.e., maintain full rollup state)

L1 ⇔ Rollup Architecture



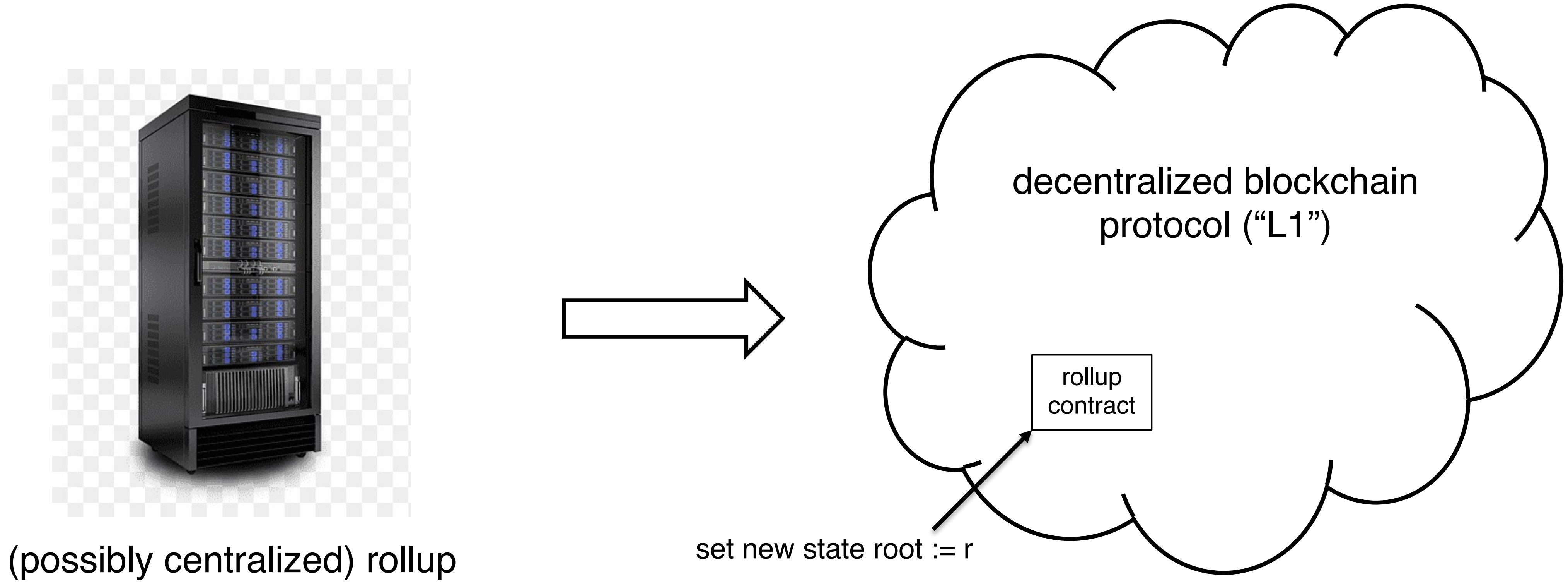
Introduction to Rollups

Assume: a decentralized "layer-one" blockchain ("L1") with strong consistency and liveness guarantees. (e.g., Ethereum)

"Classic" rollup: a blockchain/virtual machine with its own state

- not necessarily decentralized, subject to crash or Byzantine failure
- performs its own consensus (i.e., tx sequencing) and execution
- associated with smart contract(s) running on the L1
- publishes tx sequence via L1 contract (i.e., uses L1 for data availability)
    - note: anyone can run a rollup full node (i.e., maintain full rollup state)
- periodically publishes commitment to rollup state (e.g. state root) to L1
    - note: any full node can check correctness of commitment

L1 ⇔ Rollup Architecture



Dealing with Rollup Failures

“Classic” rollup: a blockchain/virtual machine with its own state

- performs its own consensus (i.e., tx sequencing) and execution
- publishes tx sequence via L1 contract (i.e., uses L1 for data availability)
- periodically publishes commitment to rollup state (e.g. state root) to L1

Dealing with Rollup Failures

"Classic" rollup: a blockchain/virtual machine with its own state

- performs its own consensus (i.e., tx sequencing) and execution
- publishes tx sequence via L1 contract (i.e., uses L1 for data availability)
- periodically publishes commitment to rollup state (e.g. state root) to L1

Protection against rollup liveness failure: can "reboot" or "fork" rollup to resume execution from most recent state commitment.

- tx data available on L1 ➔ blockchain state (not just state root) is known

Dealing with Rollup Failures

“Classic” rollup: a blockchain/virtual machine with its own state

- performs its own consensus (i.e., tx sequencing) and execution
- publishes tx sequence via L1 contract (i.e., uses L1 for data availability)
- periodically publishes commitment to rollup state (e.g. state root) to L1

Protection against rollup liveness failure: can “reboot” or “fork” rollup to resume execution from most recent state commitment.

- tx data available on L1 ➔ blockchain state (not just state root) is known

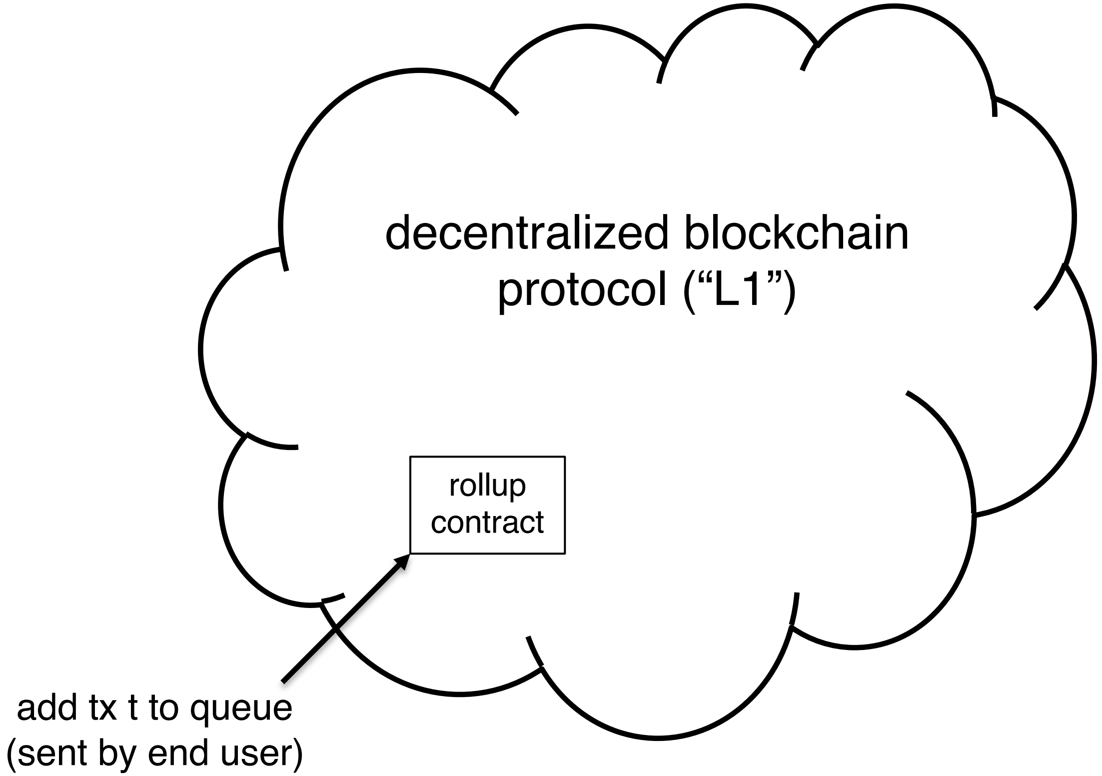
Protection against rollup safety failure: any full node can detect an incorrect state commitment and raise an alarm.

Two Requirements for “Classic” Rollups

1. Escape hatch/forced tx inclusion via the L1.
   - any user can send a rollup tx direct to the rollup’s L1 contract to force its inclusion in the next batch of rollup txs

Forcing the Inclusion of a Rollup Tx

(possibly centralized) rollup

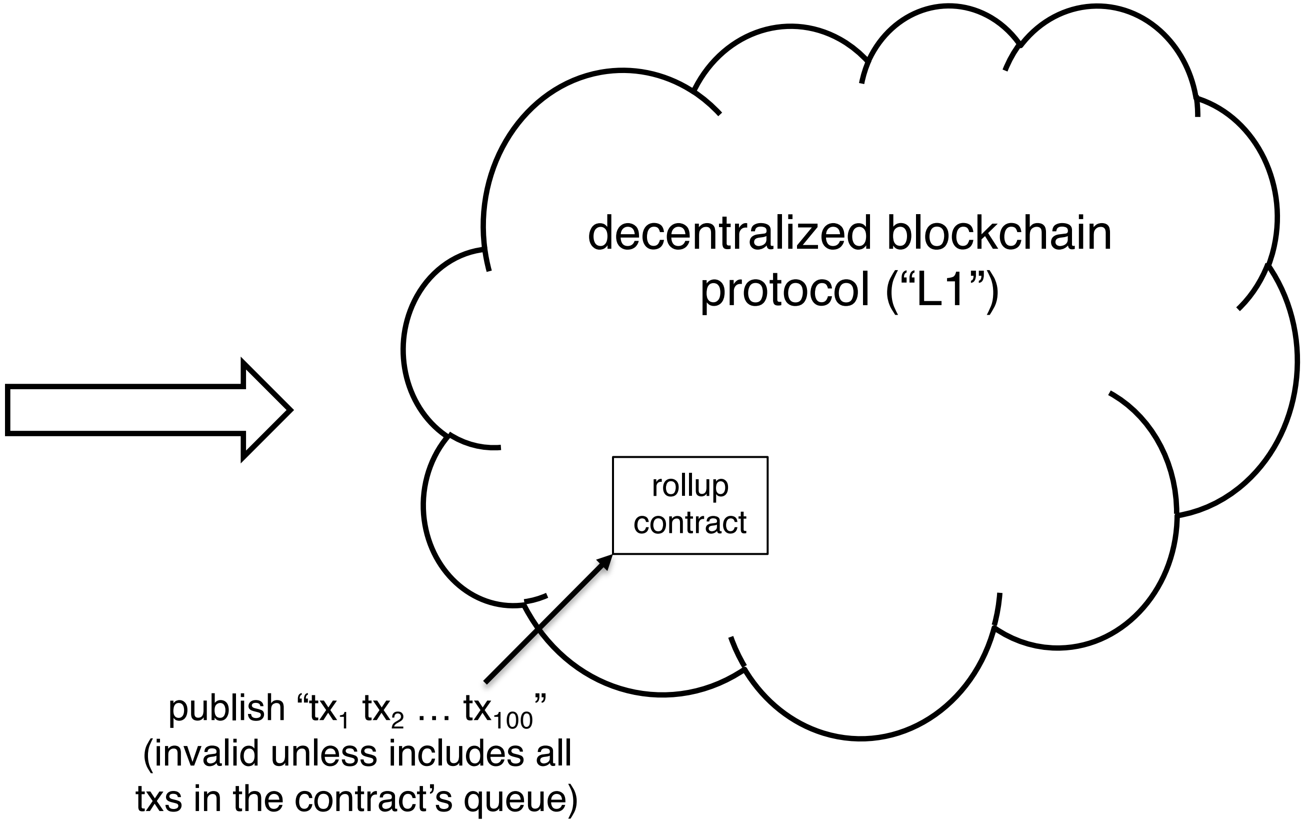


Two Requirements for "Classic" Rollups

1. Escape hatch/forced tx inclusion via the L1.
   - any user can send a rollup tx direct to the rollup's L1 contract to force its inclusion in the next batch of rollup txs
     - L1 tx records the specified rollup tx in queue in rollup's L1 contract
     - next publication of rollup txs must "clear the queue" to be valid

Forcing the Inclusion of a Rollup Tx

(possibly centralized) rollup



Two Requirements for “Classic” Rollups

1. Escape hatch/forced tx inclusion via the L1.
   - any user can send a rollup tx direct to the rollup’s L1 contract to force its inclusion in the next batch of rollup txs
     - L1 tx records the specified rollup tx in queue in rollup’s L1 contract
     - next publication of rollup txs must “clear the queue” to be valid
   - rollup liveness failure ➔ can use L1 for liveness until reboot completes

Two Requirements for "Classic" Rollups

1. Escape hatch/forced tx inclusion via the L1.
   - any user can send a rollup tx direct to the rollup's L1 contract to force its inclusion in the next batch of rollup txs
     - L1 tx records the specified rollup tx in queue in rollup's L1 contract
     - next publication of rollup txs must "clear the queue" to be valid
   - rollup liveness failure ➔ can use L1 for liveness until reboot completes
   - rollup inherits the "censorship-resistance" of the L1

Two Requirements for “Classic” Rollups

1. Escape hatch/forced tx inclusion via the L1.
   - any user can send a rollup tx direct to the rollup’s L1 contract to force its inclusion in the next batch of rollup txs
   - rollup liveness failure ➔ can use L1 for liveness until reboot completes
   - rollup inherits the “censorship-resistance” of the L1

2. State commitment correctness verified by L1.
   - as opposed to relying on full nodes

Two Requirements for "Classic" Rollups

1. Escape hatch/forced tx inclusion via the L1.
   - any user can send a rollup tx direct to the rollup's L1 contract to force its inclusion in the next batch of rollup txs
   - rollup liveness failure ➔ can use L1 for liveness until reboot completes
   - rollup inherits the "censorship-resistance" of the L1

2. State commitment correctness verified by L1.
   - as opposed to relying on full nodes
   - naïve/infeasible approach: L1 re-computes rollup's state commitment
     - defeats purpose of rollup (to offload execution from L1 validators)

Two Requirements for “Classic” Rollups

1. Escape hatch/forced tx inclusion via the L1.
   - any user can send a rollup tx direct to the rollup’s L1 contract to force its inclusion in the next batch of rollup txs
   - rollup liveness failure ➔ can use L1 for liveness until reboot completes
   - rollup inherits the “censorship-resistance” of the L1

2. State commitment correctness verified by L1.
   - as opposed to relying on full nodes
   - naïve/infeasible approach: L1 re-computes rollup’s state commitment
     - defeats purpose of rollup (to offload execution from L1 validators)

Question: how can L1 verify correctness without tx re-execution?

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   - L1 assumes state commitment correct unless dispute raised, re-execution only as needed to resolve dispute
   - rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (which L1 can verify directly)

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   – L1 assumes state commitment correct unless dispute raised, re-execution only as needed to resolve dispute
   – rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (which L1 can verify directly)

2. Validity (a.k.a. “zk”/“proof-based”) rollups. (ex: StarkWare, zkSync)

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   - L1 assumes state commitment correct unless dispute raised, re-execution only as needed to resolve dispute
   - rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (which L1 can verify directly)

2. Validity (a.k.a. “zk”/“proof-based”) rollups. (ex: StarkWare, zkSync)
   - rollup sequencer publishes easy-to-verify proof of correctness (“SNARK”) along with each new tx batch + state commitment, L1 can verify SNARK directly

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   - L1 assumes state commitment correct unless dispute raised, re-execution only as needed to resolve dispute
   - rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (which L1 can verify directly)

2. Validity (a.k.a. "zk"/"proof-based") rollups. (ex: StarkWare, zkSync)
   - rollup sequencer publishes easy-to-verify proof of correctness ("SNARK") along with each new tx batch + state commitment, L1 can verify SNARK directly
   - SNARKs known since mid-1990s, becoming practical in mid-2020s

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   - L1 assumes state commitment correct, re-execution only as needed to resolve dispute
   - rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness
2. Validity (a.k.a. "zk"/"proof-based") rollups. (ex: StarkWare, zkSync)
   - rollup sequencer publishes easy-to-verify proof of correctness ("SNARK") along with each new tx batch + state commitment, L1 can verify SNARK directly

Question: which is better? Summary of trade-offs:

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   – L1 assumes state commitment correct, re-execution only as needed to resolve dispute
   – rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness

2. Validity (a.k.a. "zk"/"proof-based") rollups. (ex: StarkWare, zkSync)
   – rollup sequencer publishes easy-to-verify proof of correctness ("SNARK") along with each new tx batch + state commitment, L1 can verify SNARK directly

Question: which is better? Summary of trade-offs:

– dispute resolution logic complex; SNARKs *really* complex

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   - L1 assumes state commitment correct, re-execution only as needed to resolve dispute
   - rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness

2. Validity (a.k.a. “zk”/“proof-based”) rollups. (ex: StarkWare, zkSync)
   - rollup sequencer publishes easy-to-verify proof of correctness (“SNARK”) along with each new tx batch + state commitment, L1 can verify SNARK directly

Question: which is better? Summary of trade-offs:

– dispute resolution logic complex; SNARKs *really* complex

– economic guarantees (optimistic) vs. cryptographic guarantees (validity)

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   – L1 assumes state commitment correct, re-execution only as needed to resolve dispute
   – rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness
2. Validity (a.k.a. "zk"/"proof-based") rollups. (ex: StarkWare, zkSync)
   – rollup sequencer publishes easy-to-verify proof of correctness ("SNARK") along with each new tx batch + state commitment, L1 can verify SNARK directly

Question: which is better? Summary of trade-offs:

– dispute resolution logic complex; SNARKs *really* complex

– economic guarantees (optimistic) vs. cryptographic guarantees (validity)

– common case requires little work (optimistic) vs. lots of work (validity)

Two Approaches to Verification

1. Optimistic rollups. (examples: Arbitrum, Base, Optimism)
   - L1 assumes state commitment correct, re-execution only as needed to resolve dispute
   - rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness
2. Validity (a.k.a. "zk"/"proof-based") rollups. (ex: StarkWare, zkSync)
   - rollup sequencer publishes easy-to-verify proof of correctness ("SNARK") along with each new tx batch + state commitment, L1 can verify SNARK directly

Question: which is better? Summary of trade-offs:

- dispute resolution logic complex; SNARKs *really* complex
- economic guarantees (optimistic) vs. cryptographic guarantees (validity)
- common case requires little work (optimistic) vs. lots of work (validity)
- rollup txs might get reversed (optimistic) vs. final (validity)

EIP-4844

Rollup: uses L1 for availability of tx data.

EIP-4844

Rollup: uses L1 for availability of tx data.

Original approach: stuff compressed tx descriptions into calldata.

- not stored in blockchain state, only in historical tx data

EIP-4844

Rollup: uses L1 for availability of tx data.

Original approach: stuff compressed tx descriptions into calldata.

– not stored in blockchain state, only in historical tx data

EIP-4844: specifically reserve portion of block for data availability.

EIP-4844

Rollup: uses L1 for availability of tx data.

Original approach: stuff compressed tx descriptions into calldata.

– not stored in blockchain state, only in historical tx data

EIP-4844: specifically reserve portion of block for data availability.

– introduces "blob" txs, max 6 blobs/block, ≈ 125kB/block
– blob data only at consensus layer, validators can delete after 2-3 weeks
– KZG commitments to blobs included in tx data (verified by validators)

EIP-4844

Rollup: uses L1 for availability of tx data.

Original approach: stuff compressed tx descriptions into calldata.

– not stored in blockchain state, only in historical tx data

EIP-4844: specifically reserve portion of block for data availability.

– introduces "blob" txs, max 6 blobs/block, ≈ 125kB/block

– blob data only at consensus layer, validators can delete after 2-3 weeks

– KZG commitments to blobs included in tx data (verified by validators)

Upshot: rollup txs became much cheaper (by 10-100x).

– blobs priced separately from regular txs

Optimistic Rollups: The High-Level Idea

Idea: watchdogs correct inaccurate state commitments.

Optimistic Rollups: The High-Level Idea

Idea: watchdogs correct inaccurate state commitments.

Sequencer: party authorized to publish rollup txs to L1 contract.

- includes new state commitment with each batch
- deposits bounty (i.e., lots of money) for catching bogus commitments

Optimistic Rollups: The High-Level Idea

Idea: watchdogs correct inaccurate state commitments.

Sequencer: party authorized to publish rollup txs to L1 contract.

- includes new state commitment with each batch
- deposits bounty (i.e., lots of money) for catching bogus commitments

Challengers: anyone can propose an alternative state commitment any published batch of rollup txs.

- deposits money (to L1 contract) along with its challenge

Optimistic Rollups: The High-Level Idea

Idea: watchdogs correct inaccurate state commitments.

Sequencer: party authorized to publish rollup txs to L1 contract.

- includes new state commitment with each batch
- deposits bounty (i.e., lots of money) for catching bogus commitments

Challengers: anyone can propose an alternative state commitment any published batch of rollup txs.

- deposits money (to L1 contract) along with its challenge

Dispute resolution: L1 contract determines correct commitment.

- idea: re-execute minimal amount to determine winner

Dispute Resolution

Canonical scenario: initial state commitment σ_0, assumed correct.

- ordered batch L = $t_1, t_2, \ldots, t_k$ of txs
- sequencer alleges that σ_1 is correct state commitment after executing L
- defender disagrees, posts alternative commitment $\sigma'_1 \neq \sigma_1$

»

Dispute Resolution

Canonical scenario: initial state commitment σ_0, assumed correct.

- ordered batch L = $t_1, t_2, \ldots, t_k$ of txs
- sequencer alleges that σ_1 is correct state commitment after executing L
- defender disagrees, posts alternative commitment $\sigma'_1 \neq \sigma_1$

»

Resolving σ'_1 *vs.* σ_1: view processing of txs in L as a sequence $\mu_1, \mu_2, \ldots, \mu_N$ of EVM states (≈ one per line of EVM bytecode executed)

Dispute Resolution

Canonical scenario: initial state commitment σ_0, assumed correct.

- ordered batch L = $t_1, t_2, \ldots, t_k$ of txs
- sequencer alleges that σ_1 is correct state commitment after executing L
- defender disagrees, posts alternative commitment $\sigma'_1 \neq \sigma_1$

Resolving σ'_1 $vs.$ σ_1: view processing of txs in L as a sequence $\mu_1, \mu_2, \ldots, \mu_N$ of EVM states (≈ one per line of EVM bytecode executed)

- sequencer posts Merkle tree root r committing to its EVM computation
  - leaves = μ_i's [μ_1 = consistent with σ_0, μ_N = consistent with σ_1]

Dispute Resolution

Canonical scenario: initial state commitment σ_0, assumed correct.

– ordered batch L = $t_1, t_2, \ldots, t_k$ of txs

– sequencer alleges that σ_1 is correct state commitment after executing L

– defender disagrees, posts alternative commitment $\sigma'_1 \neq \sigma_1$

Resolving σ'_1 $vs.$ σ_1: view processing of txs in L as a sequence $\mu_1, \mu_2, \ldots, \mu_N$ of EVM states (≈ one per line of EVM bytecode executed)

– sequencer posts Merkle tree root r committing to its EVM computation

- leaves = μ_i's [μ_1 = consistent with σ_0, μ_N = consistent with σ_1]

– defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection Games

Resolving $\sigma'_1\ vs.\ \sigma_1$: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

Bisection Games

Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer reveals midpoint $\mu_{N/2}$ of its computation (with Merkle proof)
  - i.e., submits to rollup's L1 contract, which verifies the proof

Bisection Games

Resolving $\sigma'_1\ vs.\ \sigma_1$: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer reveals midpoint $\mu_{N/2}$ of its computation (with Merkle proof)
  - i.e., submits to rollup's L1 contract, which verifies the proof
- defender reveals midpoint $\mu'_{N/2}$ of its computation (with Merkle proof)

Bisection Games

Resolving $\sigma'_1 \; vs. \, \sigma_1$: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer reveals midpoint $\mu_{N/2}$ of its computation (with Merkle proof)
  - i.e., submits to rollup's L1 contract, which verifies the proof
- defender reveals midpoint $\mu'_{N/2}$ of its computation (with Merkle proof)
- if $\mu_{N/2} = \mu'_{N/2}$ ➔ recurse on second half of computation trace

Bisection Games

Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer reveals midpoint $\mu_{N/2}$ of its computation (with Merkle proof)
  - i.e., submits to rollup's L1 contract, which verifies the proof
- defender reveals midpoint $\mu'_{N/2}$ of its computation (with Merkle proof)
- if $\mu_{N/2} = \mu'_{N/2}$ ➔ recurse on second half of computation trace
- if $\mu_{N/2} \neq \mu'_{N/2}$ ➔ recurse on first half of computation trace

Bisection Games

Resolving $\sigma'_1\ vs.\ \sigma_1$: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \dots, \mu_N$
- defender posts commitment r’ to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

Bisection game:

- sequencer reveals midpoint $\mu_{N/2}$ of its computation (with Merkle proof)
- defender reveals midpoint $\mu'_{N/2}$ of its computation (with Merkle proof)
- if $\mu_{N/2} = \mu'_{N/2}$ ➔ recurse on second half of computation trace
- if $\mu_{N/2} \neq \mu'_{N/2}$ ➔ recurse on first half of computation trace
- repeat until locate position i of computation s.t. $\mu_i = \mu'_i$ and $\mu_{i+1} \neq \mu'_{i+1}$

Bisection Games

Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer, defender reveal midpoints $\mu_{N/2}$,$\mu'_{N/2}$ of computations
- repeatedly recurse on first or second half of computation trace until locate position i of computation s.t. $\mu_i = \mu'_i$ and $\mu_{i+1} \neq \mu'_{i+1}$

Bisection Games

Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer, defender reveal midpoints $\mu_{N/2}$,$\mu'_{N/2}$ of computations
- repeatedly recurse on first or second half of computation trace until locate position i of computation s.t. $\mu_i = \mu'_i$ and $\mu_{i+1} \neq \mu'_{i+1}$
- L1 contract directly verifies if transition $\mu_i \rightarrow \mu_{i+1}$ correctly computed

Bisection Games

Resolving $\sigma'_1 \; vs. \; \sigma_1$: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \dots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

Bisection game:

- sequencer, defender reveal midpoints $\mu_{N/2}$,$\mu'_{N/2}$ of computations
- repeatedly recurse on first or second half of computation trace until locate position i of computation s.t. $\mu_i = \mu'_i$ and $\mu_{i+1} \neq \mu'_{i+1}$
- L1 contract directly verifies if transition $\mu_i \rightarrow \mu_{i+1}$ correctly computed
  - ≈ simulating one step of the EVM (inside a smart contract)
  - if not, contract rejects σ_1 as invalid, confiscates sequencer's stake
  - if so, contract confiscates challenger's stake

Properties of Optimistic Rollups

Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \dots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

Bisection game: sequencer, defender reveal midpoints of computations

- repeatedly recurse on first or second half of computation trace until locate position i of computation s.t. $\mu_i = \mu'_i$ and $\mu_{i+1} \neq \mu'_{i+1}$
- L1 contract directly verifies if transition $\mu_i \rightarrow \mu_{i+1}$ correctly computed

Properties of Optimistic Rollups

Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game: sequencer, defender reveal midpoints of computations

- repeatedly recurse on first or second half of computation trace until locate position i of computation s.t. $\mu_i = \mu'_i$ and $\mu_{i+1} \neq \mu'_{i+1}$
- L1 contract directly verifies if transition $\mu_i \rightarrow \mu_{i+1}$ correctly computed

Good news: incorrect state commitment ➔ big economic penalty.

Properties of Optimistic Rollups

Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts commitment r to its computation $\mu_1, \mu_2, \dots, \mu_N$
- defender posts commitment r' to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

Bisection game: sequencer, defender reveal midpoints of computations

- repeatedly recurse on first or second half of computation trace until locate position i of computation s.t. $\mu_i = \mu'_i$ and $\mu_{i+1} \neq \mu'_{i+1}$
- L1 contract directly verifies if transition $\mu_i \to \mu_{i+1}$ correctly computed

Good news: incorrect state commitment ➔ big economic penalty.

Bad news: requires time (days) for dispute resolution to play out.