Lecture #16: Optimistic Rollups

COMS 4995-001: The Science of Blockchains

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

Tim Roughgarden

Goals for Lecture #16

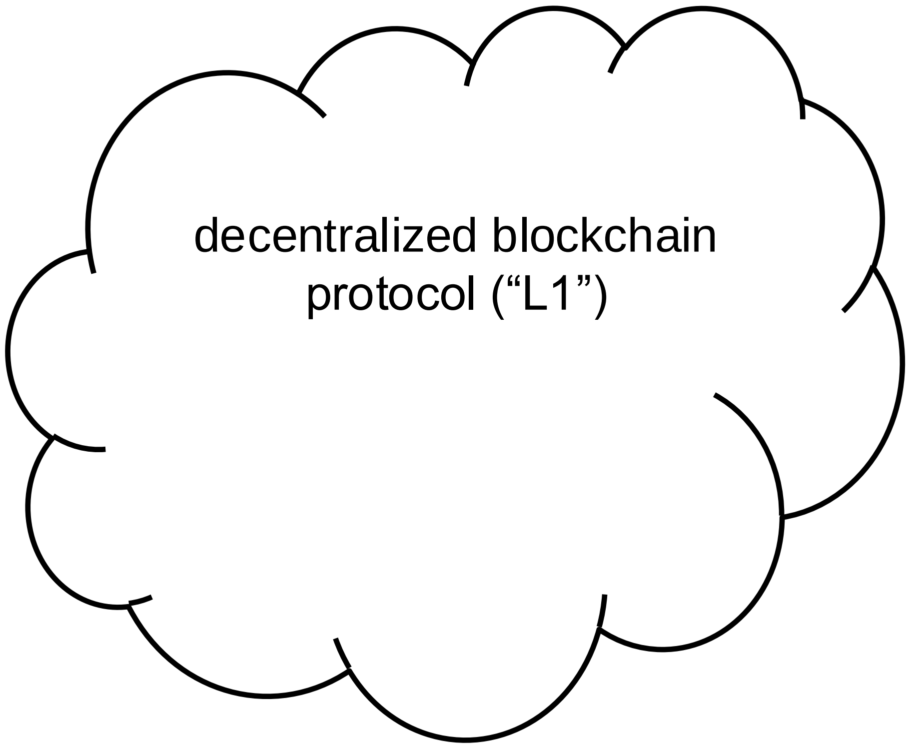
1. Rollups review.
   – an approach to sharding blockchain state and execution
   – piggyback on an “L1” for data availability, liveness, etc.
   – central to the Ethereum ecosystem

2. Optimistic rollups. (e.g., Arbitrum, Base, Optimism)
   – rollup state commitments verified via “bisection game”
   – “cryptoeconomic” security: derived from economic penalties (confiscation of staked collateral)

Recall: “Classic” Rollups

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

L1 ⬄ Rollup Architecture



Recall: “Classic” 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

decentralized blockchain
protocol (“L1”)

Recall: “Classic” Rollups

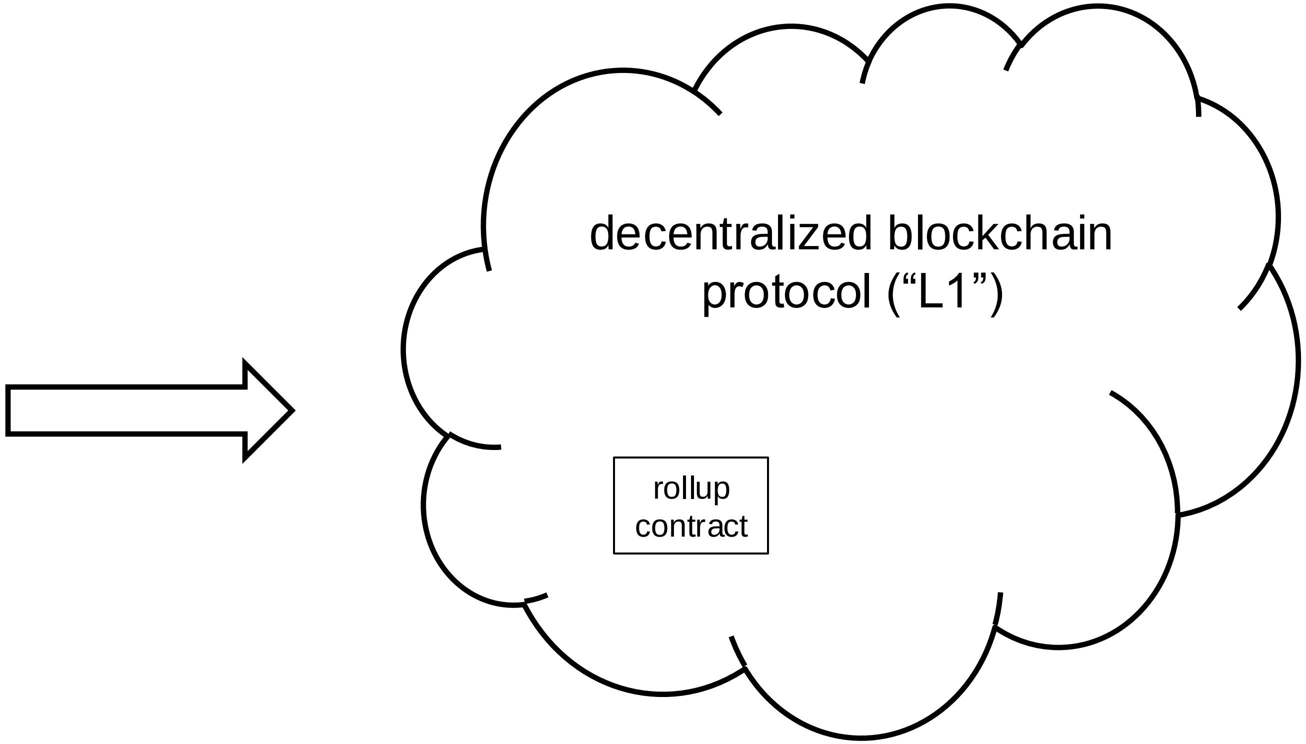
Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

“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



Recall: “Classic” 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)

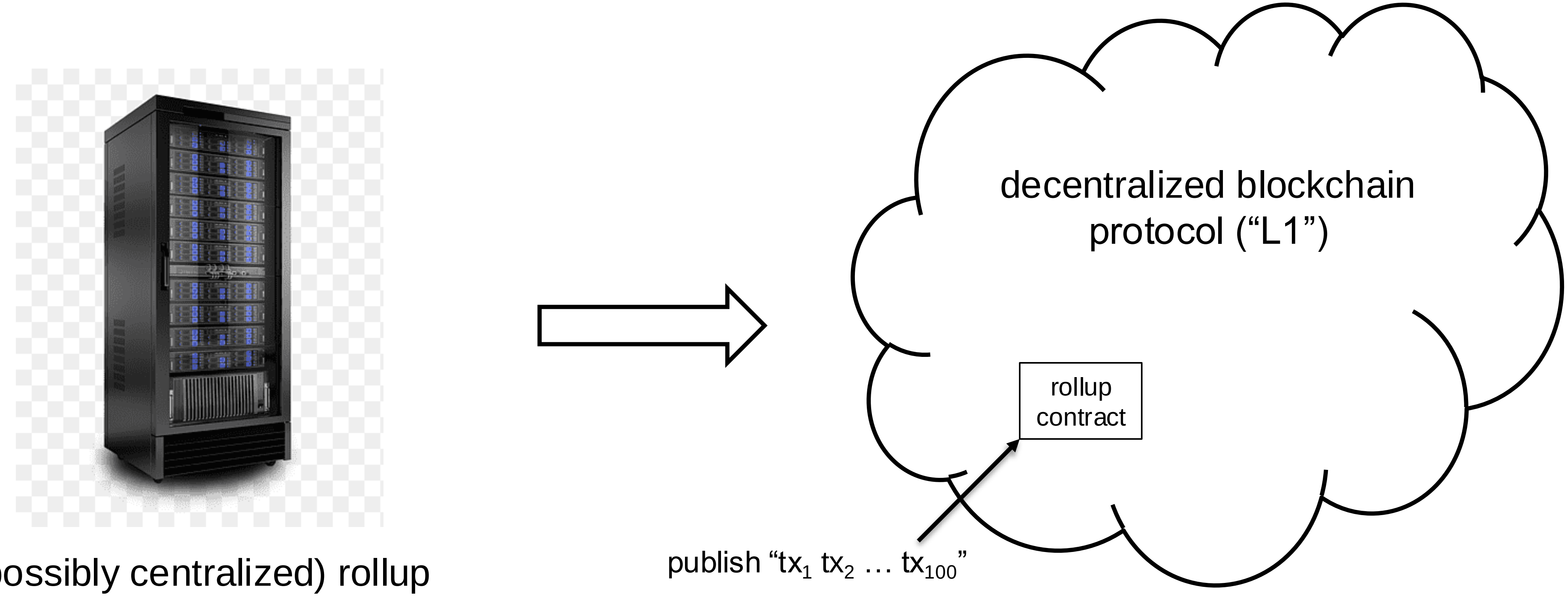
Recall: “Classic” Rollups

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)
  - pre-EIP-4844: via call data
  - post-EIP-4844: via blobs

L1 ⇔ Rollup Architecture



Recall: “Classic” Rollups

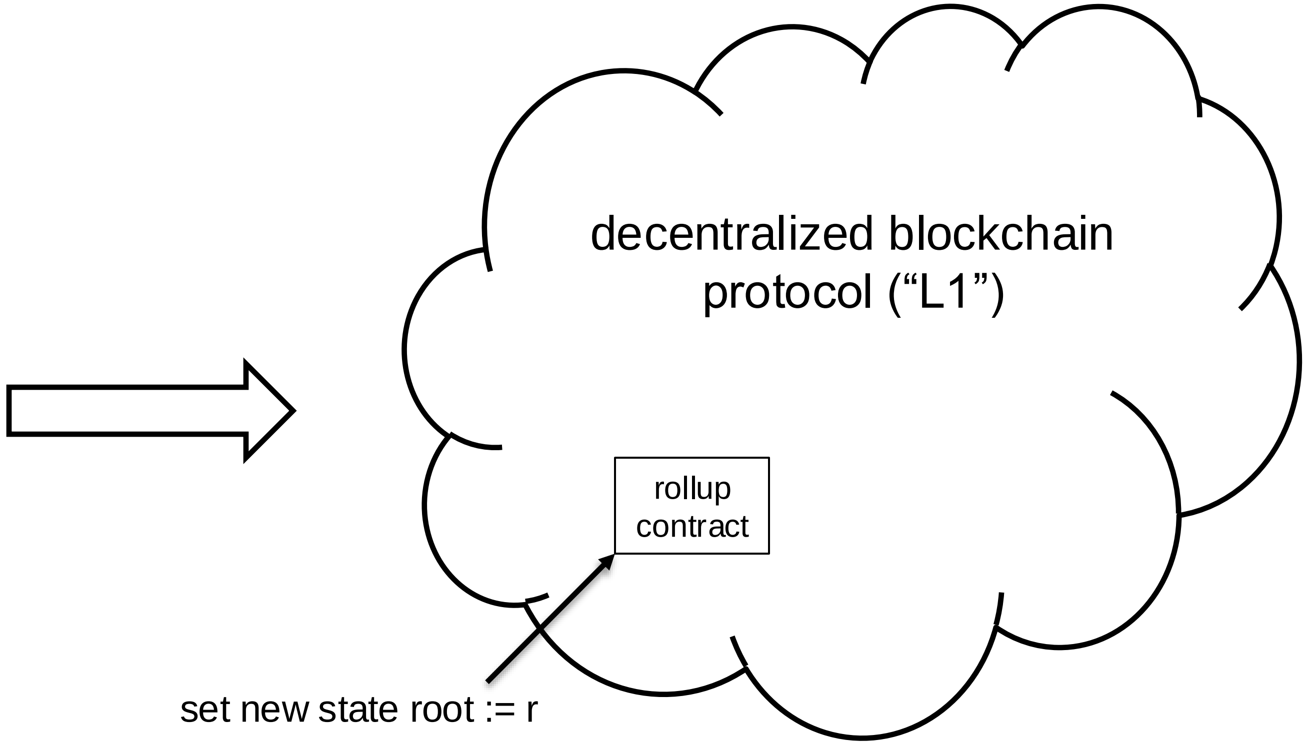
Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

“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)
- periodically publishes commitment to rollup state (e.g. state root) to L1
  - note: any full node can check correctness of commitment

L1 ⇔ Rollup Architecture

(possibly centralized) rollup



Recall: “Classic” Rollups

“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)
- periodically publishes commitment to rollup state (e.g. state root) to L1
    - note: any full node can check correctness of commitment

Recall: “Classic” Rollups

“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)
- periodically publishes commitment to rollup state (e.g. state root) to L1
  - note: any full node can check correctness of commitment
- (hard part) state commitment correctness verified by L1

Recall: “Classic” Rollups

“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)
- periodically publishes commitment to rollup state (e.g. state root) to L1
  - note: any full node can check correctness of commitment
- (hard part) state commitment correctness verified by L1
  - question: how can L1 do this without re-executing rollup txs itself?

Optimistic Rollups

High-level idea: innocent until proven guilty.

- examples: Arbitrum, Base, Optimism

Optimistic Rollups

High-level idea: innocent until proven guilty.

- examples: Arbitrum, Base, Optimism

- L1 assumes by default that each state commitment is correct
  - should be the common case; no work needed

Optimistic Rollups

High-level idea: innocent until proven guilty.

- examples: Arbitrum, Base, Optimism

- L1 assumes by default that each state commitment is correct
  - should be the common case; no work needed
- rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (“fault proof”)
  - intuitively, a specific line of code that was executed incorrectly

Optimistic Rollups

High-level idea: innocent until proven guilty.

- examples: Arbitrum, Base, Optimism

- L1 assumes by default that each state commitment is correct
  - should be the common case; no work needed
- rely on watchdogs to catch incorrect state commitments, submit short proof of incorrectness (“fault proof”)
  - intuitively, a specific line of code that was executed incorrectly
- L1 verifies proof of incorrectness directly
  - L1 performs minimal re-execution necessary to resolve dispute
  - details quite complex, hard to get right

Sequencers vs. Challengers

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

Sequencers vs. Challengers

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

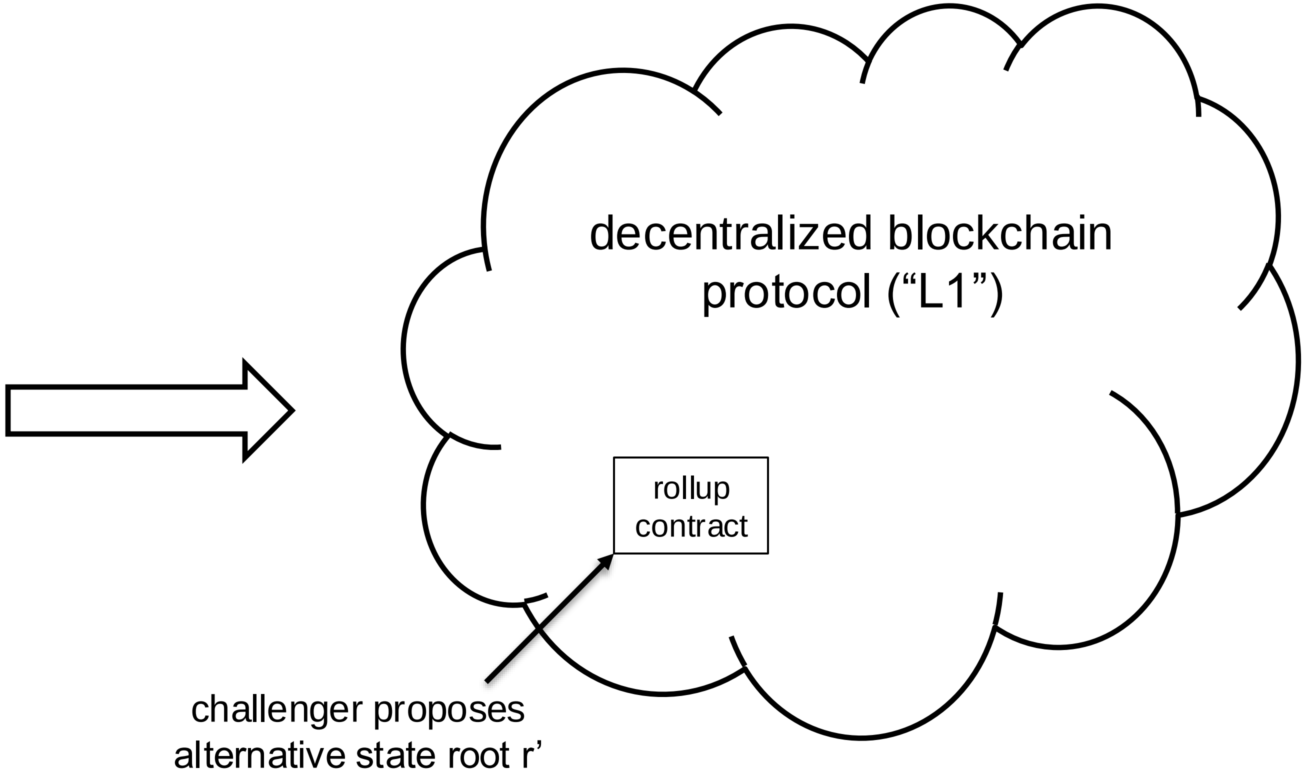
- includes new state commitment with each batch
- deposits bounty (i.e., lots of money) for catching bogus commitments

Challengers: anyone can propose (to the L1 contract) an alternative state commitment for any published batch of rollup txs.

- deposits money (to L1 contract) along with its challenge
- only need “1 out of N honest” assumption

L1 ⇔ Rollup Architecture

(possibly centralized) rollup



Sequencers vs. Challengers

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 (to the L1 contract) an alternative state commitment for any published batch of rollup txs.

- deposits money (to L1 contract) along with its challenge
- only need “1 out of N honest” assumption

Question: how can L1 know which state commitment is correct?

Dispute Resolution

Idea: L1 performs minimal amount of re-execution necessary to determine winner (sequencer vs. challenger).

- easily the most complex + tricky component of an optimistic rollup

Dispute Resolution

Idea: L1 performs minimal amount of re-execution necessary to determine winner (sequencer vs. challenger).

- easily the most complex + tricky component of an optimistic rollup

Canonical scenario: initial state commitment σ_0, assumed correct.

- ordered batch L = $t_1, t_2, \ldots, t_k$ of rollup txs
- sequencer alleges that σ_1 is correct state commitment after executing L
- challenger disagrees, posts alternative commitment $\sigma'_1 \neq \sigma_1$
- for simplicity, suppose both L1 + rollup execution layers are EVM-based

Dispute Resolution

Canonical scenario: initial state commitment σ_0, assumed correct.

- ordered batch L = $t_1, t_2, \ldots, t_k$ of rollup txs
- sequencer posts σ_1, challenger posts $\sigma'_1 \neq \sigma_1$

Resolving σ'_1 $vs.$ σ_1:

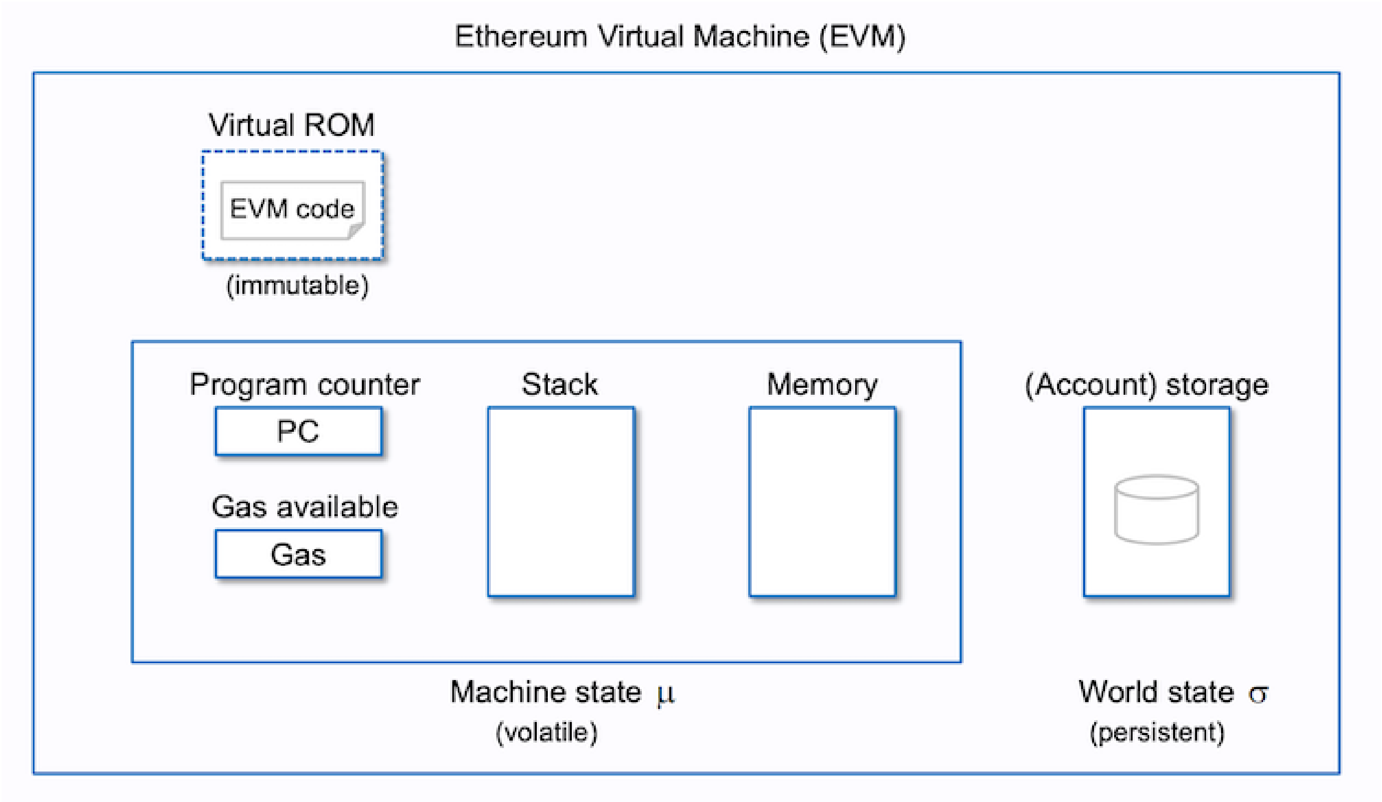
Dispute Resolution

Canonical scenario: initial state commitment σ_0, assumed correct.

- ordered batch L = $t_1,t_2,\ldots,t_k$ of rollup txs
- sequencer posts σ_1, challenger posts $\sigma'_1 \neq \sigma_1$

Resolving σ'_1 $vs.\sigma_1$: *execution trace* = view execution of $t_1,t_2,\ldots,t_k$ as sequence $\mu_1, \mu_2, \ldots, \mu_N$ of EVM states (≈ one per line of EVM bytecode executed, in the rollup's execution layer)

Visualizing an EVM State



[source: https://www.quicknode.com/guides/ethereum-development/smart-contracts/a-dive-into-evm-architecture-and-opcodes]

Dispute Resolution

Canonical scenario: initial state commitment σ_0, assumed correct.

- ordered batch L = $t_1, t_2, \ldots, t_k$ of rollup txs
- sequencer posts σ_1, challenger posts $\sigma'_1 \neq \sigma_1$

Resolving σ'_1 $vs.\sigma_1$: *execution trace* = view execution of $t_1, t_2, \ldots, t_k$ as sequence $\mu_1, \mu_2, \ldots, \mu_N$ of EVM states (≈ one per line of EVM bytecode executed, in the rollup's execution layer)

- 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 rollup txs
- sequencer posts σ_1, challenger posts $\sigma'_1 \neq \sigma_1$

Resolving σ'_1 *vs.* σ_1: *execution trace* = view execution of $t_1, t_2, \ldots, t_k$ as sequence $\mu_1, \mu_2, \ldots, \mu_N$ of EVM states (≈ one per line of EVM bytecode executed, in the rollup's execution layer)

- 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]
- challenger 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 Merkle tree root r committing to its computation $\mu_1, \mu_2, \dots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

Bisection game:

Bisection Games

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

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing 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 σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing 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
- challenger reveals midpoint $\mu'_{N/2}$ of its computation (with Merkle proof)

Bisection Games

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

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \dots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

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
- challenger reveals midpoint $\mu'_{N/2}$ of its computation (with Merkle proof)

Throughout: if one party fails to submit expected L1 tx in a reasonable (TBA) amount of time ➔ lose dispute and its stake.

Bisection Games

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

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing 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
- challenger reveals midpoint $\mu'_{N/2}$ of its computation (with Merkle proof)

Bisection Games

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

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing 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
- challenger 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 Merkle tree root r committing to its computation $\mu_1, \mu_2, \dots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

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
- challenger 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 Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer reveals midpoint $\mu_{N/2}$ of its computation (with Merkle proof)
- challenger 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 $\sigma'_1\ vs.\ \sigma_1$: view processing of txs in as a sequence of EVM states

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer, challenger 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 $vs.\sigma_1$: view processing of txs in as a sequence of EVM states

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \dots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \dots, \mu'_N$

Bisection game:

- sequencer, challenger 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

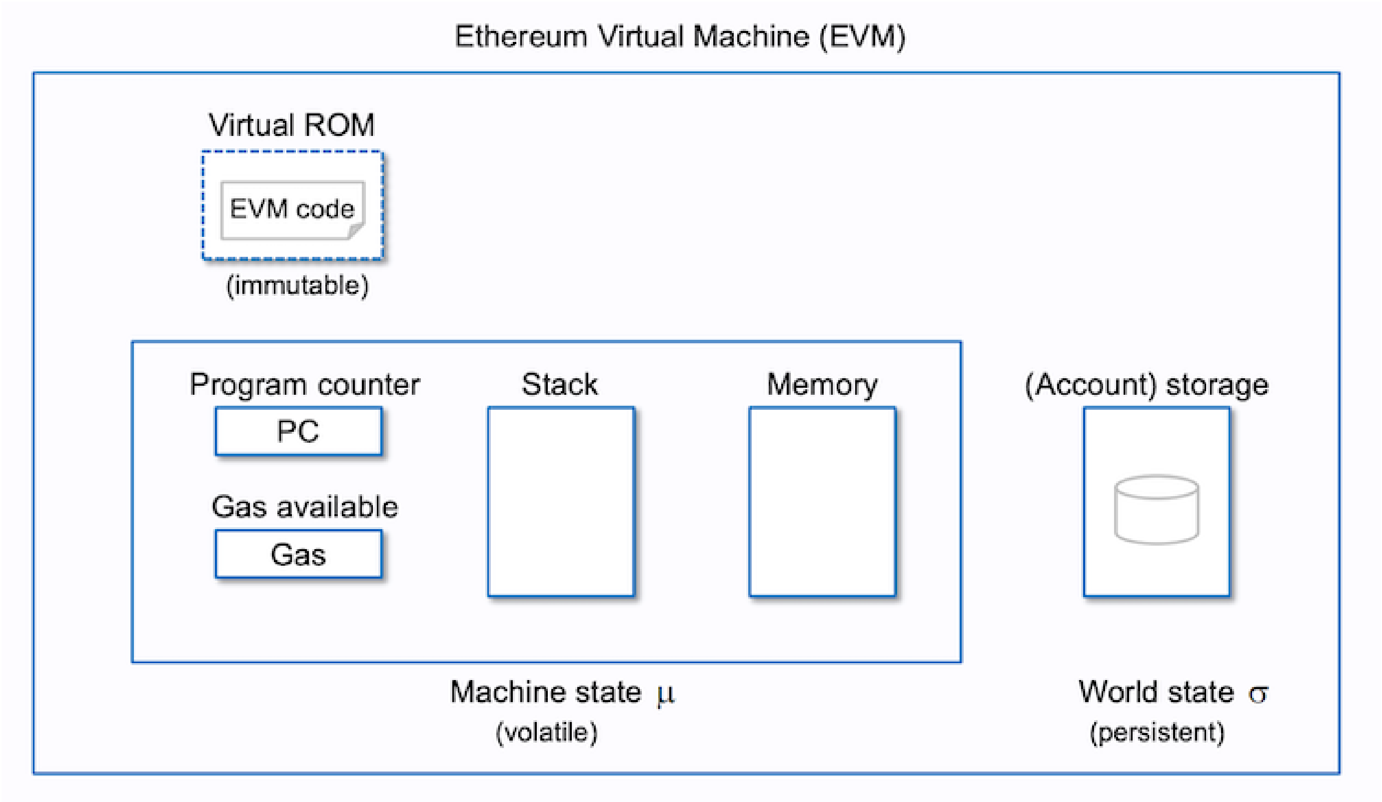
Resolving σ'_1 $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer, challenger 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)

Visualizing an EVM State



[source: https://www.quicknode.com/guides/ethereum-development/smart-contracts/a-dive-into-evm-architecture-and-opcodes]

Bisection Games

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

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game:

- sequencer, challenger 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

Key Property of Optimistic Rollups

Resolving ${\sigma'}_1$ $vs.$ σ_1: view processing of txs in as a sequence of EVM states

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \dots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation ${\mu'}_1, {\mu'}_2, \dots, {\mu'}_N$

Bisection game: sequencer, challenger 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

Key Property of Optimistic Rollups

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

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game: sequencer, challenger 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

Key property: if sequencer posts incorrect state commitment, any honest challenger can win dispute resolution

Key Property of Optimistic Rollups

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

- sequencer posts Merkle tree root r committing to its computation $\mu_1, \mu_2, \ldots, \mu_N$
- challenger posts Merkle tree root r' committing to its computation $\mu'_1, \mu'_2, \ldots, \mu'_N$

Bisection game: sequencer, challenger 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

Key property: if sequencer posts incorrect state commitment, any honest challenger can win dispute resolution ➔ no safety violation, and big economic penalty to sequencer.

- would expect sequencer to only publish correct state commitments

Danger: Censorship by L1 Validators

Issue: what if honest challengers can't get the necessary L1 txs executed (by the L1) in time?

Danger: Censorship by L1 Validators

Issue: what if honest challengers can’t get the necessary L1 txs executed (by the L1) in time?

– note: attacker would need to block *all* honest challengers (recall “1 in N”)

Danger: Censorship by L1 Validators

Issue: what if honest challengers can’t get the necessary L1 txs executed (by the L1) in time?

- note: attacker would need to block *all* honest challengers (recall “1 in N”)
- note: any honest challenger can pick up dispute resolution where another one left off (due to uniqueness of the correct computation)

Danger: Censorship by L1 Validators

Issue: what if honest challengers can't get the necessary L1 txs executed (by the L1) in time?

– note: attacker would need to block *all* honest challengers (recall "1 in N")

– note: any honest challenger can pick up dispute resolution where another one left off (due to uniqueness of the correct computation)

Question: how could things go wrong?

Danger: Censorship by L1 Validators

Issue: what if honest challengers can't get the necessary L1 txs executed (by the L1) in time?

- note: attacker would need to block *all* honest challengers (recall "1 in N")
- note: any honest challenger can pick up dispute resolution where another one left off (due to uniqueness of the correct computation)

Question: how could things go wrong?

- DoS attacks (e.g., if only whitelisted challengers)

Danger: Censorship by L1 Validators

Issue: what if honest challengers can't get the necessary L1 txs executed (by the L1) in time?

- note: attacker would need to block *all* honest challengers (recall "1 in N")
- note: any honest challenger can pick up dispute resolution where another one left off (due to uniqueness of the correct computation)

Question: how could things go wrong?

- DoS attacks (e.g., if only whitelisted challengers)
- failure of L1 (i.e., not consistent/live/censorship-resistant)

Danger: Censorship by L1 Validators

Issue: what if honest challengers can't get the necessary L1 txs executed (by the L1) in time?

- note: attacker would need to block *all* honest challengers (recall "1 in N")
- note: any honest challenger can pick up dispute resolution where another one left off (due to uniqueness of the correct computation)

Question: how could things go wrong?

- DoS attacks (e.g., if only whitelisted challengers)
- failure of L1 (i.e., not consistent/live/censorship-resistant)
- bribery (i.e., dishonest rollup sequencer pays L1 validators to exclude challengers' L1 txs)

Choosing the Dispute Period

Issue: DoS attacks/L1 failure/censorship via bribery.

Choosing the Dispute Period

Issue: DoS attacks/L1 failure/censorship via bribery.

Solution: rollup state commitment regarded as tentative until it’s been undisputed for sufficiently long number T of L1 blocks.

- e.g., T = 7 days worth of blocks (hopefully makes an attack infeasible)

Choosing the Dispute Period

Issue: DoS attacks/L1 failure/censorship via bribery.

Solution: rollup state commitment regarded as tentative until it’s been undisputed for sufficiently long number T of L1 blocks.

- e.g., T = 7 days worth of blocks (hopefully makes an attack infeasible)

Fact: long delay before final rollup tx confirmation the biggest drawback of optimistic rollups (cf., validity rollups).

Choosing the Dispute Period

Issue: DoS attacks/L1 failure/censorship via bribery.

Solution: rollup state commitment regarded as tentative until it's been undisputed for sufficiently long number T of L1 blocks.

- e.g., T = 7 days worth of blocks (hopefully makes an attack infeasible)

Fact: long delay before final rollup tx confirmation the biggest drawback of optimistic rollups (cf., validity rollups).

- rollup users have option to treat rollups txs as finalized earlier, if desired
  - cf., security parameter k in longest-chain consensus

Example: The Cost of Censorship

Simplified model:

Example: The Cost of Censorship

Simplified model:

- to foil a rogue sequencer, honest challenger must successfully submit N L1 txs over the course of T blocks

Example: The Cost of Censorship

Simplified model:

- to foil a rogue sequencer, honest challenger must successfully submit N L1 txs over the course of T blocks
- each block proposer includes challenger's tx if and only if challenger pays proposer at least as much as sequencer

Example: The Cost of Censorship

Simplified model:

- to foil a rogue sequencer, honest challenger must successfully submit N L1 txs over the course of T blocks
- each block proposer includes challenger's tx if and only if challenger pays proposer at least as much as sequencer

HW6: as long as challenger's budget for paying proposers is at least ≈ N/T times that of the sequencer ➔ guaranteed to win.

Example: The Cost of Censorship

Simplified model:

- to foil a rogue sequencer, honest challenger must successfully submit N L1 txs over the course of T blocks
- each block proposer includes challenger's tx if and only if challenger pays proposer at least as much as sequencer

HW6: as long as challenger's budget for paying proposers is at least ≈ N/T times that of the sequencer ➔ guaranteed to win.

  – in practice, N ≈ 60 and T ≈ 50K (7 days), so N/T ≈ 0.12%