Lecture #14:
Scaling Blockchain Protocols

COMS 4995-001:
The Science of Blockchains

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

Tim Roughgarden

Goals for Lecture #14

1. Bottlenecks to scaling.
   - consensus, execution, storage
2. Four approaches to scaling.
   - constrain validator set; better protocols and client implementations; outsourcing validator responsibilities; sharding/horizontal scaling
3. 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

Measuring Performance

Key security requirements: consistency and liveness.

- subject to this, want protocol to be as “high-performance” as possible

Key performance metrics:

Measuring Performance

Key security requirements: consistency and liveness.

- subject to this, want protocol to be as “high-performance” as possible

Key performance metrics:

- *throughput* (roughly, number of transactions per second)
  - tricky point: some txs require more work to process than others

Measuring Performance

Key security requirements: consistency and liveness.

- subject to this, want protocol to be as “high-performance” as possible

Key performance metrics:

- *throughput* (roughly, number of transactions per second)
  - tricky point: some txs require more work to process than others
- *latency* (time between tx submission and tx finalization)
  - tricky point: when to consider a tx confirmed/finalized?

Measuring Performance

Key security requirements: consistency and liveness.

- subject to this, want protocol to be as “high-performance” as possible

Key performance metrics:

- *throughput* (roughly, number of transactions per second)
  - tricky point: some txs require more work to process than others
- *latency* (time between tx submission and tx finalization)
  - tricky point: when to consider a tx confirmed/finalized?

Holy grail: millions of txs/sec, latency in 100s of milliseconds.

Measuring Performance

Key security requirements: consistency and liveness.

- subject to this, want protocol to be as “high-performance” as possible

Key performance metrics:

- *throughput* (roughly, number of transactions per second)
- *latency* (time between tx submission and tx finalization)

Holy grail: millions of txs/sec, latency in 100s of milliseconds.

Question: what’s stopping us?

Bottlenecks to Scaling

Answer: load on validators.

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:

Approaches to Scaling

Category #1: impose constraints on the validator set.

- often interpreted as “compromising on decentralization”

Approaches to Scaling

Category #1: impose constraints on the validator set.

- often interpreted as “compromising on decentralization”

- limit the number of validators (e.g., to maximum of 100)
  - limits work required at consensus layer
  - explicitly or implicitly e.g. through large staking requirement

Approaches to Scaling

Category #1: impose constraints on the validator set.

- often interpreted as “compromising on decentralization”

- limit the number of validators (e.g., to maximum of 100)
  - limits work required at consensus layer
  - explicitly or implicitly e.g. through large staking requirement
- require high-performance validators
  - e.g., requirements on number of cores, RAM, bandwidth, etc.
  - ideological split between Bitcoin/Ethereum and newer blockchains

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.
  - random sampling to manage large validator sets (“committees”)

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.
  - random sampling to manage large validator sets (“committees”)
  - faster/more sophisticated transaction dissemination

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.
  - random sampling to manage large validator sets (“committees”)
  - faster/more sophisticated transaction dissemination
  - “leaderless” consensus protocols

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.
  - random sampling to manage large validator sets (“committees”)
  - faster/more sophisticated block/transaction dissemination
  - “leaderless” consensus protocols
- examples at execution layer:

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.
  - random sampling to manage large validator sets (“committees”)
  - faster/more sophisticated block/transaction dissemination
  - “leaderless” consensus protocols
- examples at execution layer:
  - better state management (e.g., tailored databases to store accounts)

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.
  - random sampling to manage large validator sets (“committees”)
  - faster/more sophisticated block/transaction dissemination
  - “leaderless” consensus protocols
- examples at execution layer:
  - better state management (e.g., tailored databases to store accounts)
  - state expiry

Approaches to Scaling (con’d)

Category #2: better protocols and client implementations.

- examples at consensus layer:
  - pipelining to decrease latency (i.e., overlapping views)
  - optimistic block proposal, execution, etc.
  - random sampling to manage large validator sets (“committees”)
  - faster/more sophisticated block/transaction dissemination
  - “leaderless” consensus protocols
- examples at execution layer:
  - better state management (e.g., tailored databases to store accounts)
  - state expiry
  - parallel execution of non-conflicting transactions

Approaches to Scaling (con’d)

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

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

Approaches to Scaling (con’d)

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

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

- already standard: long-term storage outsourced to archival nodes and/or data availability committees

Approaches to Scaling (con’d)

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

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

- already standard: long-term storage outsourced to archival nodes and/or data availability committees
- recent development: block-building
  - currently (in Ethereum/Solana): to capture “MEV”
  - possible future: to outsource execution duties

Approaches to Scaling (con'd)

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

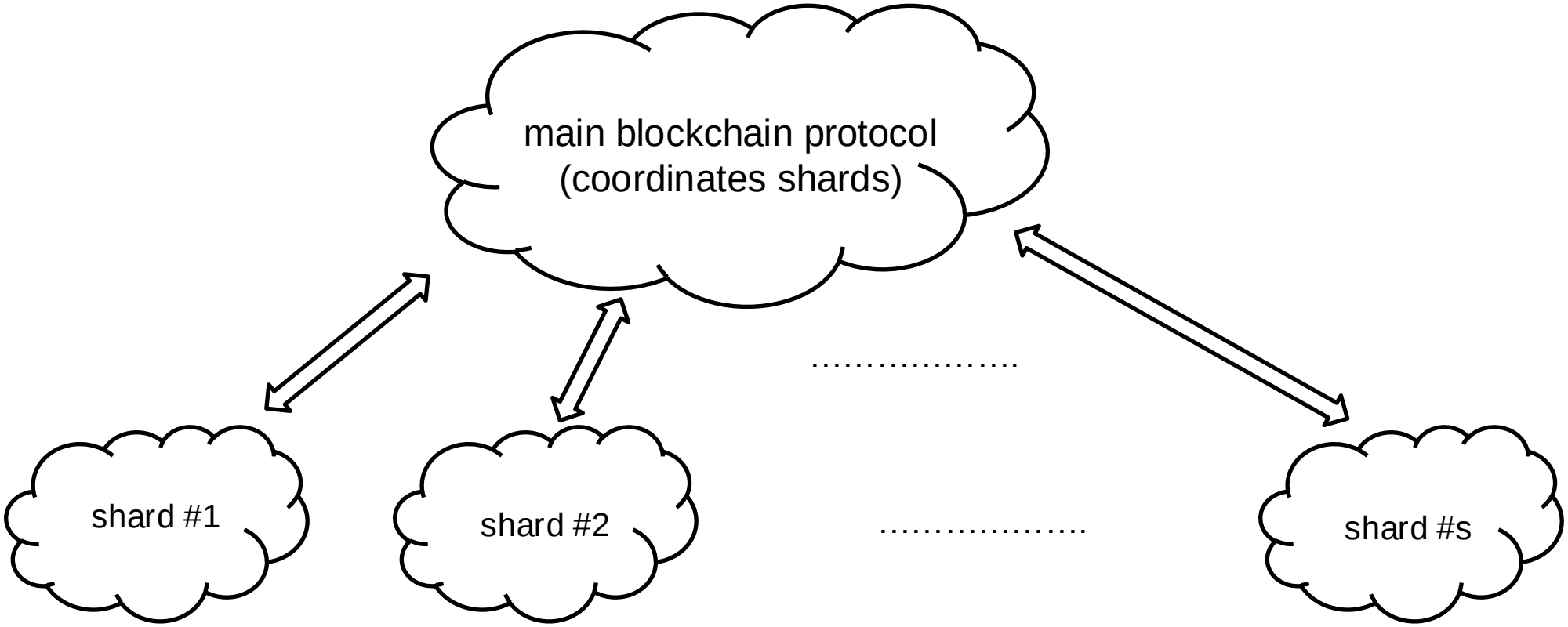
- 3rd parties may be specialized, centralized, and largely untrusted

- already standard: long-term storage outsourced to archival nodes and/or data availability committees
- recent development: block-building
  - currently (in Ethereum/Solana): to capture "MEV"
  - possible future: to outsource execution duties
- possible future: execution + maintenance of blockchain state
  - validators still expected to verify correctness of execution
  - cf., stateless clients (see last lecture)

Approaches to Scaling (con’d)

Category #4: “sharding”/horizontal scaling.

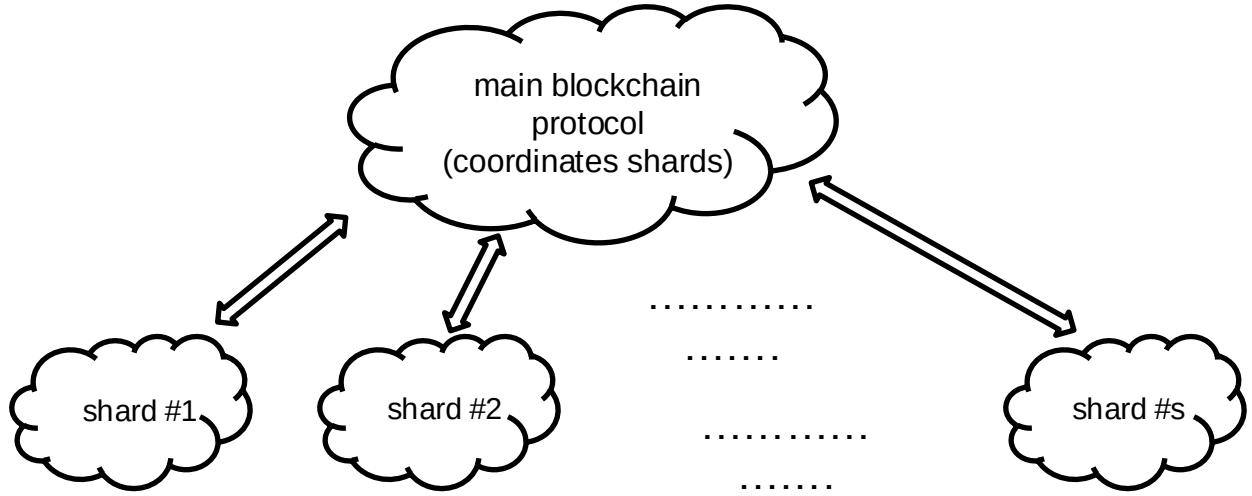
- unlike replication, want more validators ➔ more processing power



Approaches to Scaling (con’d)

Category #4: “sharding”/horizontal scaling.

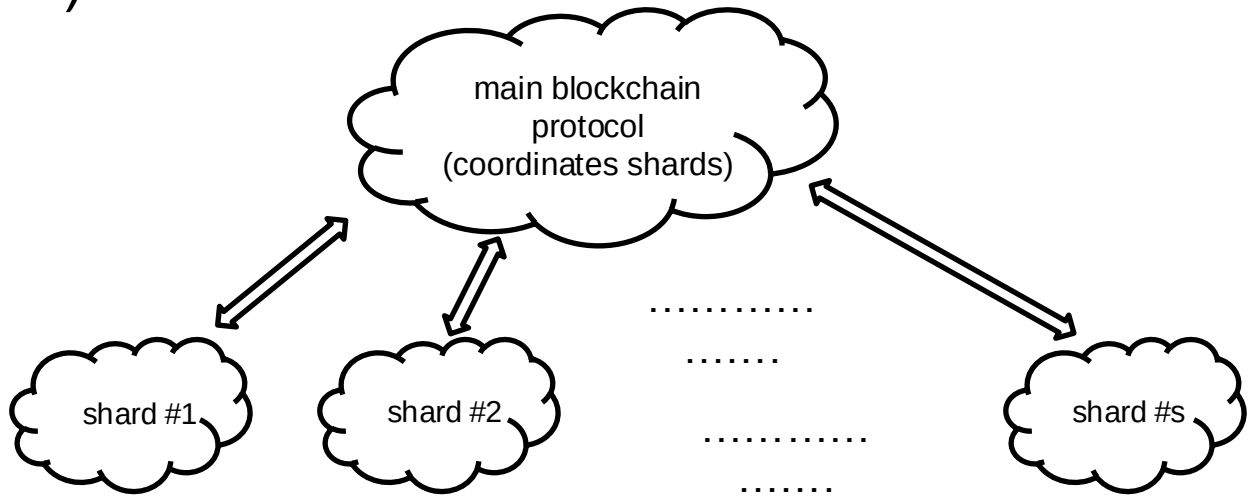
- sharding storage: conceptually straightforward



Approaches to Scaling (con’d)

Category #4: “sharding”/horizontal scaling.

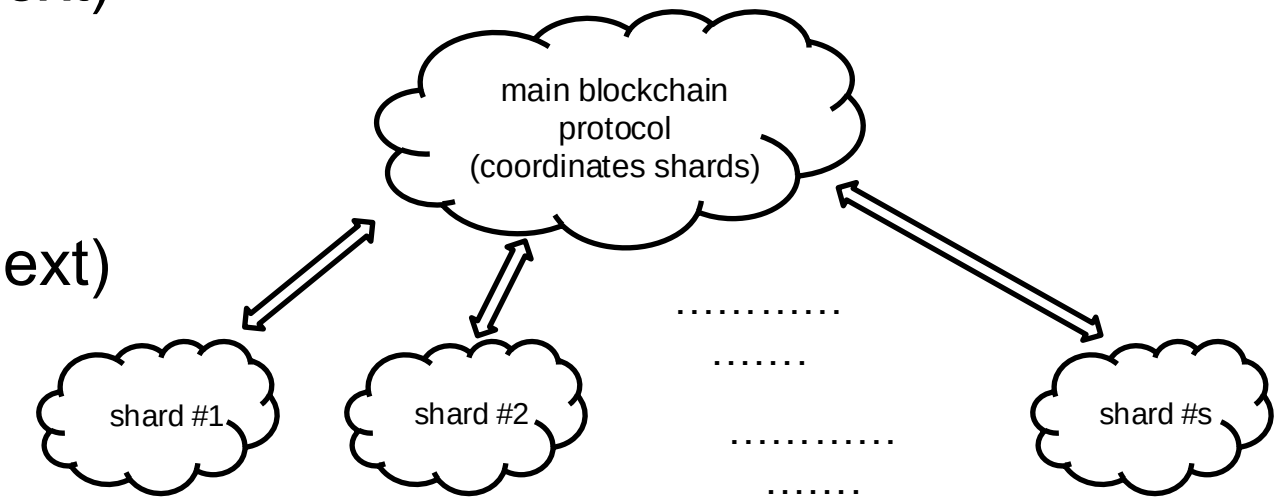
- sharding storage: conceptually straightforward
- sharding consensus (i.e., tx sequencing):
  - can be achieved via “rollups” (next)



Approaches to Scaling (con'd)

Category #4: "sharding"/horizontal scaling.

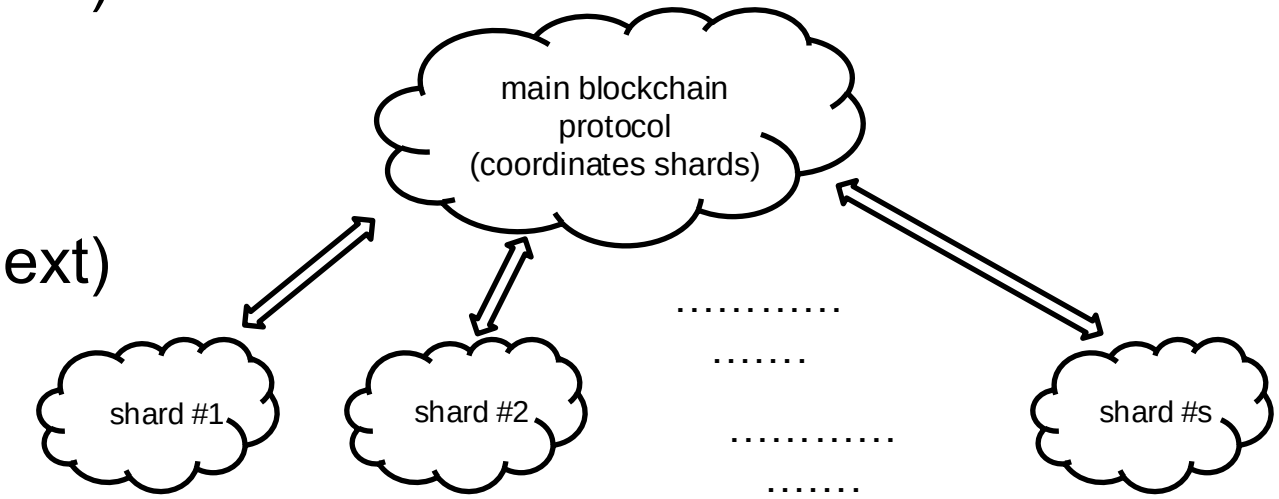
- sharding storage: conceptually straightforward
- sharding consensus (i.e., tx sequencing):
  - can be achieved via "rollups" (next)
- sharding execution:
  - with separate state per shard:
    can be achieved via "rollups" (next)



Approaches to Scaling (con'd)

Category #4: "sharding"/horizontal scaling.

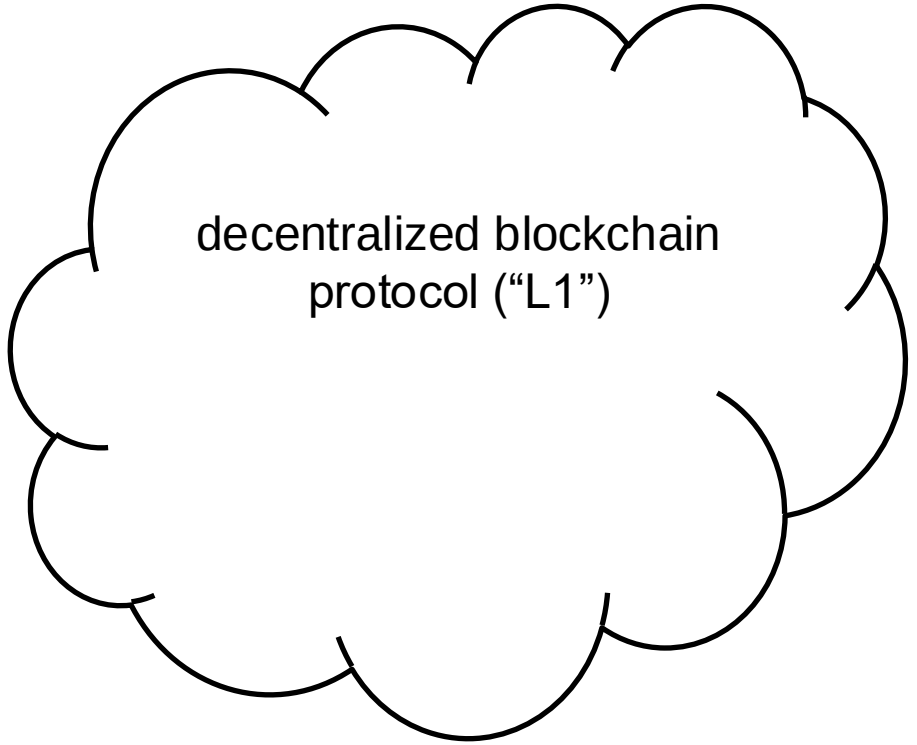
- sharding storage: conceptually straightforward
- sharding consensus (i.e., tx sequencing):
  - can be achieved via "rollups" (next)
- sharding execution:
  - with separate state per shard: can be achieved via "rollups" (next)
  - with shared state across shards: largely unsolved



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

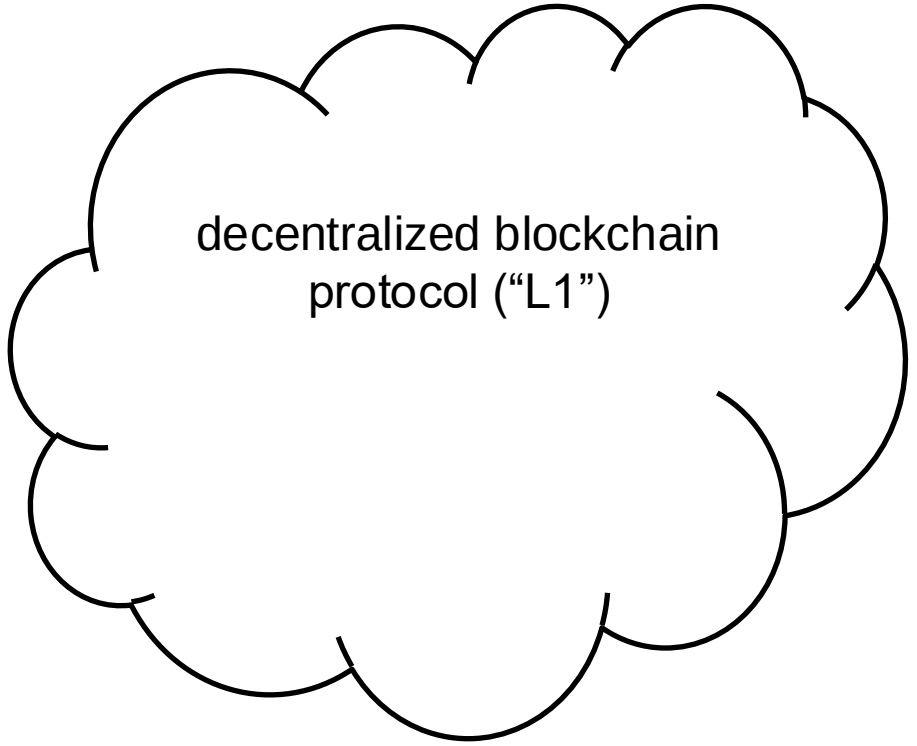
Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

“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

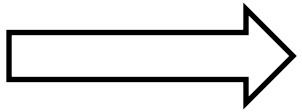
Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

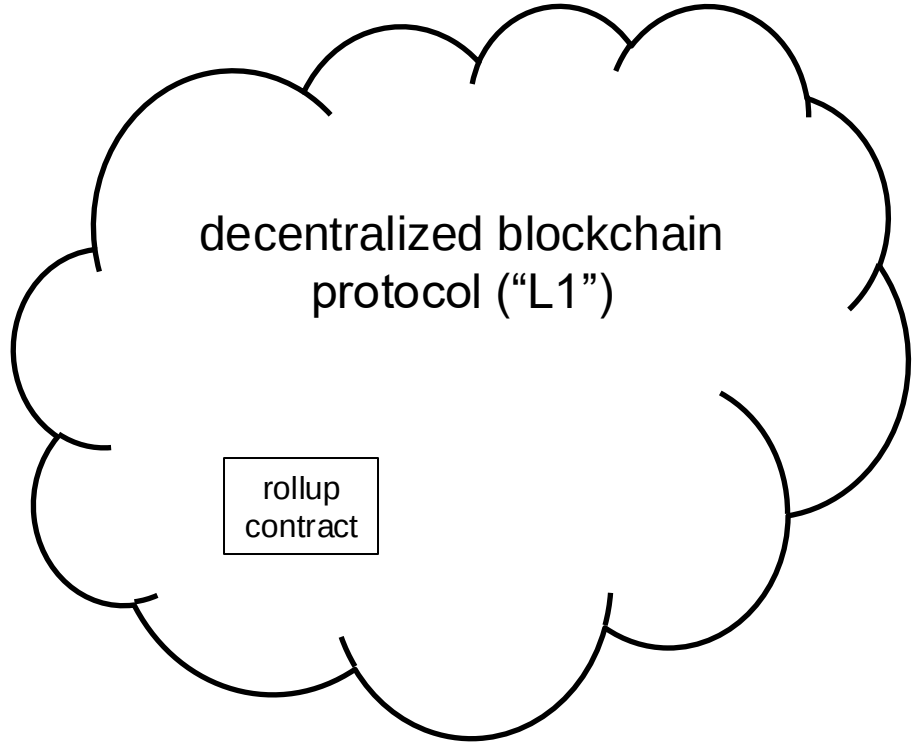
“Classic” rollup: a blockchain/virtual machine with its own state

- 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

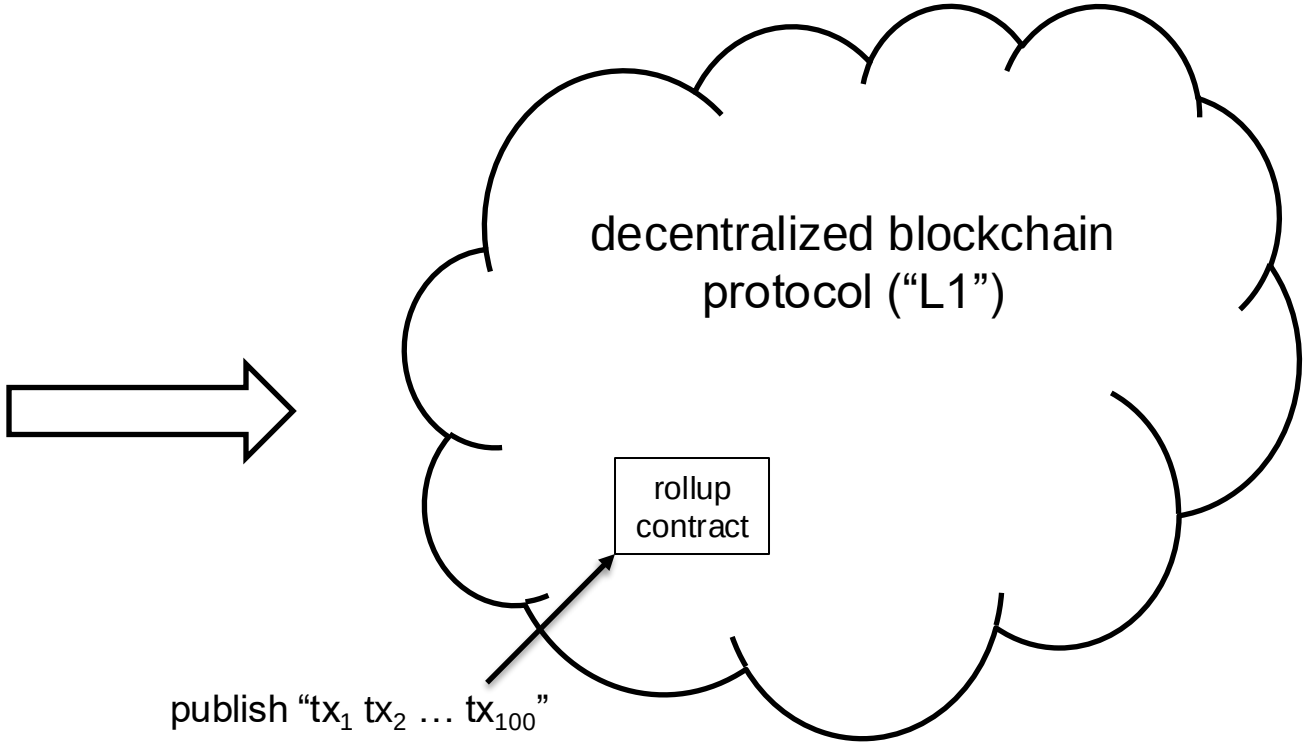
Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

“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 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)

L1 ⇔ Rollup Architecture

(possibly centralized) rollup



Introduction to Rollups

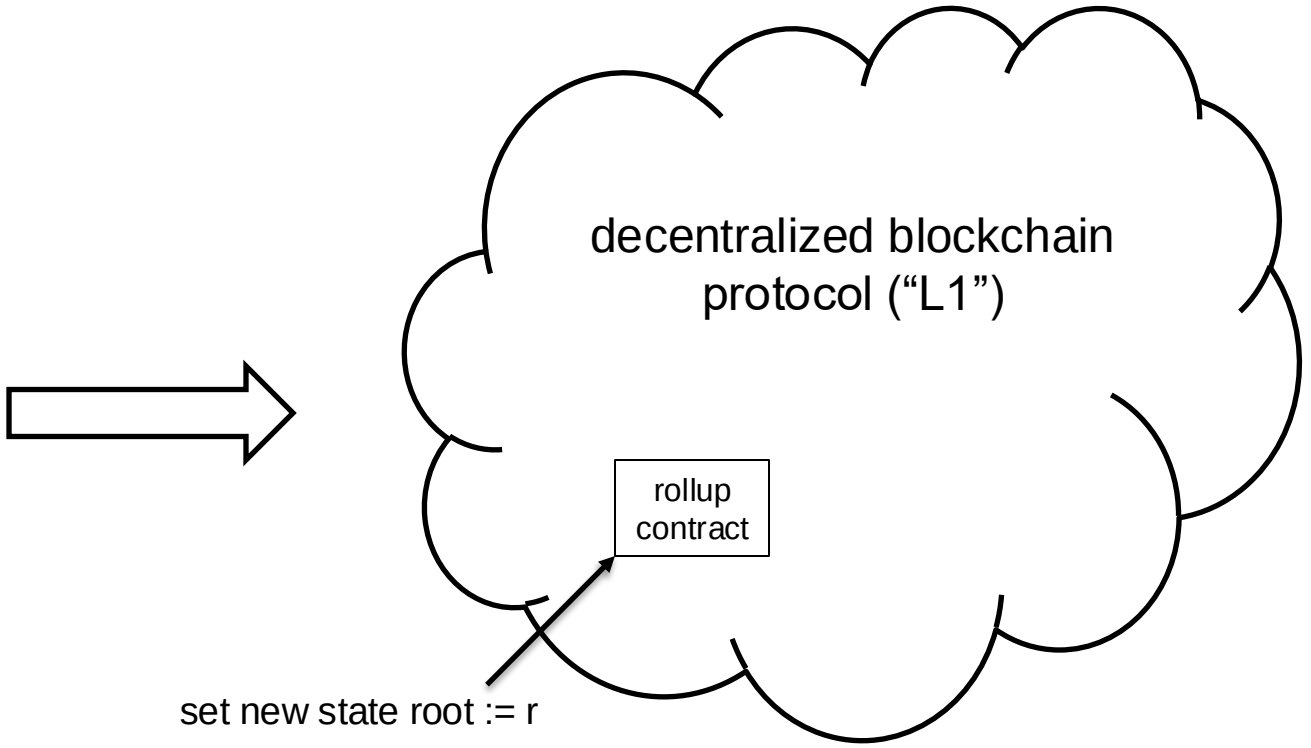
Assume: a decentralized “layer-one” blockchain (“L1”) with strong consistency and liveness guarantees. (e.g., Ethereum)

“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 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

(possibly centralized) rollup



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

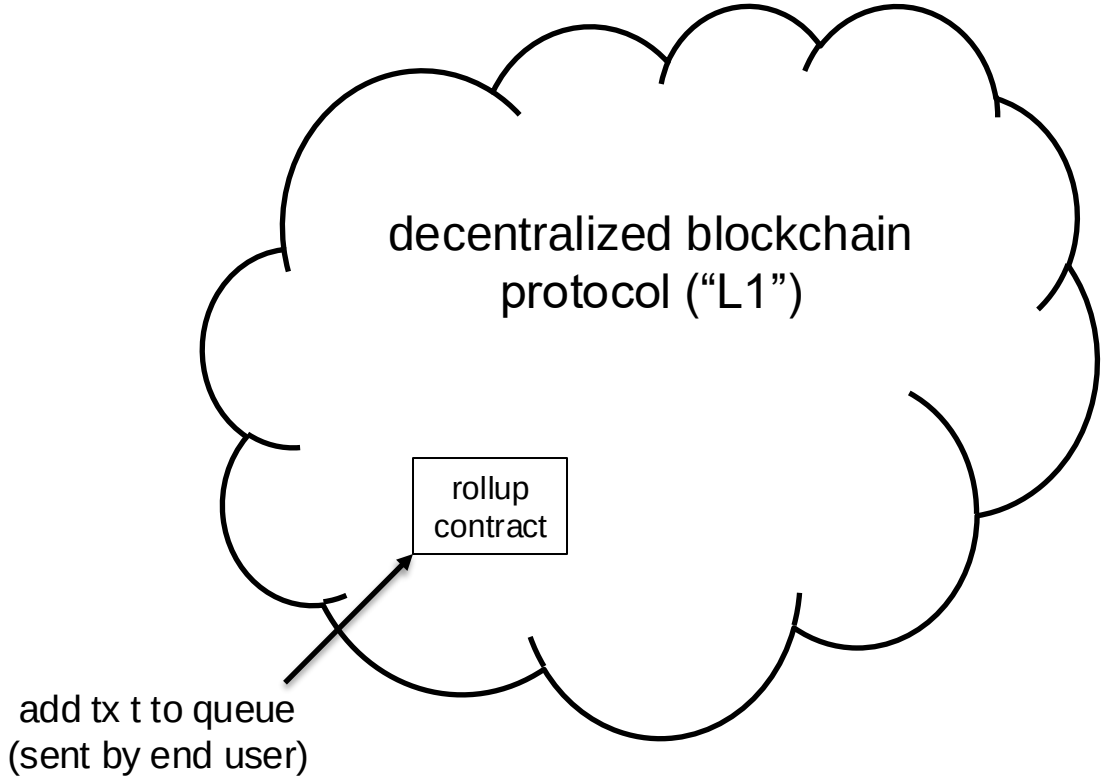
Protection against rollup safety failure: any full node can detect an incorrect state root and raise an alarm.

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

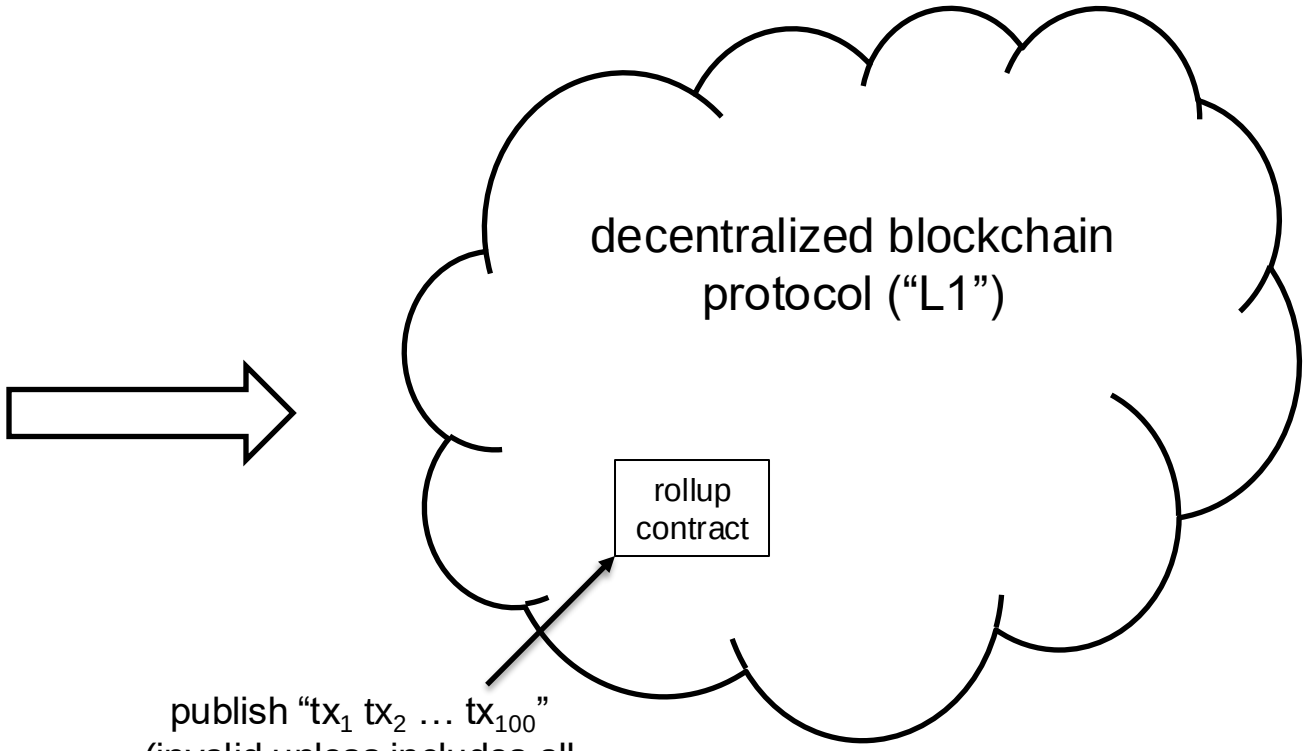


Two Requirements for “Classic” Rollups

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 “censor-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 “censor-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 “censor-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 “censor-resistance” of the L1
2. State commitment correctness verified by L1.
   - naïve/infeasible approach: L1 re-computes rollup’s state commitment
   - optimistic approach: L1 assumes state commitment correct unless dispute raised, re-execute rollup txs as needed to resolve dispute

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 “censor-resistance” of the L1

2. State commitment correctness verified by L1.
   - naïve/infeasible approach: L1 re-computes rollup’s state commitment
   - optimistic approach: L1 assumes state commitment correct unless dispute raised, re-execute rollup txs as needed to resolve dispute
   - “zk”/validity approach: L1 verifies a “SNARK” which proves the correctness of the state commitment