Lecture #28: A Glimpse of the Cutting Edge

COMS 4995-001: The Science of Blockchains

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

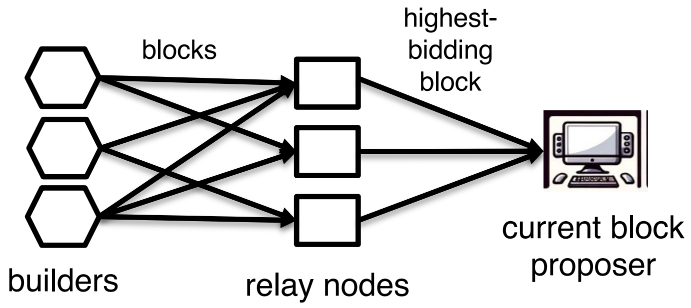
Tim Roughgarden

Goals for Lecture #28

1. Censorship-resistance.
   – experimental ideas to mitigate dangers with centralized builders
2. Protocols from principles.
   – recap of everything you now know about Bitcoin and Ethereum
3. "SNARK-ify everything."
   – scaling an L1 by outsourcing execution to builders via SNARKs
4. Some future directions.
   – e.g., "zk co-processors" to guarantee correctness of off-chain computation

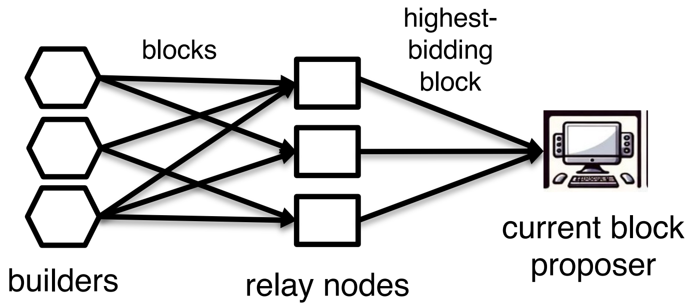
Proposer-Builder Separation + MEV-Boost

- specialized builders submit blocks (with bids) to trusted relay nodes

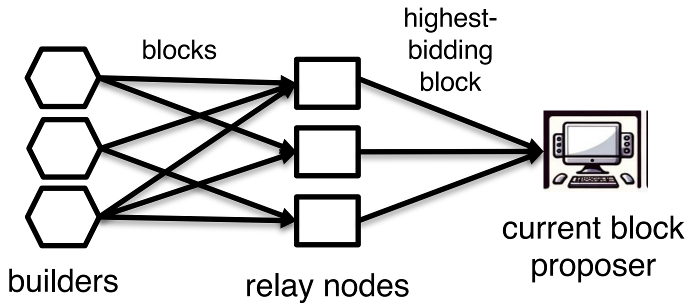


Proposer-Builder Separation + MEV-Boost

- specialized builders submit blocks (with bids) to trusted relay nodes
- relay nodes forward header of highest-bidding valid block to current block proposer

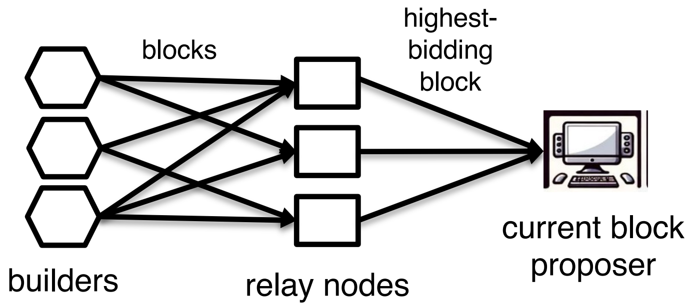


Proposer-Builder Separation + MEV-Boost



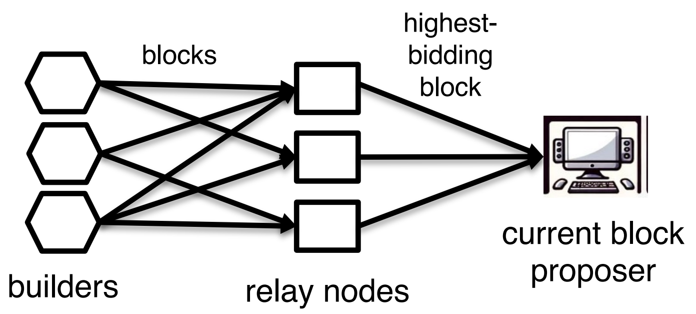
- specialized builders submit blocks (with bids) to trusted relay nodes
- relay nodes forward header of highest-bidding valid block to current block proposer
- proposer returns signer header to relay node, block is broadcast

Proposer-Builder Separation + MEV-Boost



- specialized builders submit blocks (with bids) to trusted relay nodes
- relay nodes forward header of highest-bidding valid block to current block proposer
- proposer returns signer header to relay node, block is broadcast
- relay node broadcasts signed block over gossip network

Proposer-Builder Separation + MEV-Boost



– specialized builders submit blocks (with bids) to trusted relay nodes

– relay nodes forward header of highest-bidding valid block to current block proposer

– proposer returns signer header to relay node, block is broadcast

– relay node broadcasts signed block over gossip network

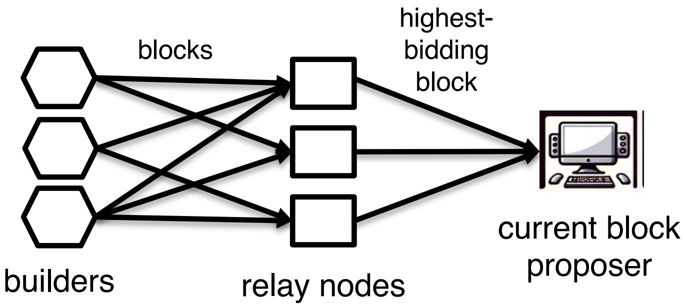
Properties: (i) validators can't steal MEV from builders

– txs in block unknown when proposer signs the block header

Proposer-Builder Separation + MEV-Boost

– specialized builders submit blocks (with bids) to trusted relay nodes

– relay nodes forward header of highest-bidding valid block to current block proposer



– proposer returns signer header to relay node, block is broadcast

– relay node broadcasts signed block over gossip network

Properties: (i) validators can't steal MEV from builders

– txs in block unknown when proposer signs the block header

- MEV rewards (per-unit-stake) equalized across validators

  – hopefully, no economic barriers to decentralized validator set

Centralization and Censorship

Question: big problem if a blockchain protocol has only a few validators?

Centralization and Censorship

Question: big problem if a blockchain protocol has only a few validators? [answer: yes, largely defeats the point of a blockchain protocol]

Centralization and Censorship

Question: big problem if a blockchain protocol has only a few validators? [answer: yes, largely defeats the point of a blockchain protocol]

Question: big problem if a blockchain protocol has only a few block-builders (but lots of validators)?

Block	Slot	Age	Txn	Fee Recipient	Gas Used	Gas Limit	Base Fee	Reward	Burnt Fees (ETH)
22330775	11548010	14 secs ago	184	Titan Builder	14,024,967 (38.96%)	35,999,965	1.166 Gwei	0.01742 ETH	0.016360 (48.42%)
22330774	11548009	26 secs ago	228	Titan Builder	17,704,576 (49.23%)	35,964,845	1.168 Gwei	0.02039 ETH	0.020692 (50.36%)
22330773	11548008	38 secs ago	204	beaverbuild	20,131,027 (55.92%)	36,000,000	1.151 Gwei	0.01934 ETH	0.023184 (54.52%)
22330772	11548007	50 secs ago	173	beaverbuild	16,695,651 (46.38%)	36,000,000	1.162 Gwei	0.01824 ETH	0.019404 (51.53%)
22330771	11548006	1 min ago	324	Titan Builder	35,644,314 (99.01%)	36,000,000	1.035 Gwei	0.0492 ETH	0.036904 (42.86%)
22330770	11548005	1 min ago	115	quasarbuilder	7,545,978 (20.96%)	36,000,000	1.116 Gwei	0.00557 ETH	0.008424 (60.18%)
22330769	11548004	1 min ago	186	beaverbuild	17,009,799 (47.25%)	35,999,965	1.124 Gwei	0.04548 ETH	0.019121 (29.59%)
22330768	11548003	1 min ago	214	beaverbuild	21,519,909 (59.84%)	35,964,845	1.097 Gwei	0.04709 ETH	0.023610 (33.39%)
22330767	11548002	1 min ago	244	Titan Builder	21,948,869 (60.97%)	36,000,000	1.067 Gwei	0.01889 ETH	0.023438 (55.37%)
22330766	11548001	2 mins ago	146	beaverbuild	10,282,523 (28.56%)	36,000,000	1.128 Gwei	0.00775 ETH	0.011602 (59.95%)
22330765	11548000	2 mins ago	211	Titan Builder	22,679,059 (63.00%)	35,999,931	1.092 Gwei	0.01958 ETH	0.024784 (55.85%)
22330764	11547999	2 mins ago	315	Titan Builder	30,120,386 (83.75%)	35,964,811	1.007 Gwei	0.0271 ETH	0.030355 (52.82%)
22330763	11547998	2 mins ago	43	Lido: Execution Layer Rew...	2,326,986 (6.48%)	35,929,725	1.13 Gwei	0.00809 ETH	0.002631 (24.52%)
22330762	11547997	2 mins ago	274	Titan Builder	25,964,257 (72.19%)	35,964,845	1.071 Gwei	0.03609 ETH	0.027818 (43.53%)
22330761	11547996	3 mins ago	216	beaverbuild	24,057,554 (66.83%)	36,000,000	1.028 Gwei	0.04062 ETH	0.024735 (37.84%)
22330760	11547995	3 mins ago	165	Lido: Execution Layer Rew...	11,446,914 (31.80%)	36,000,000	1.077 Gwei	0.00408 ETH	0.012330 (75.12%)

Centralization and Censorship

Question: big problem if a blockchain protocol has only a few validators? [answer: yes, largely defeats the point of a blockchain protocol]

Question: big problem if a blockchain protocol has only a few block-builders (but lots of validators)?

- block-building might be an intrinsically specialized skill

Centralization and Censorship

Question: big problem if a blockchain protocol has only a few validators? [answer: yes, largely defeats the point of a blockchain protocol]

Question: big problem if a blockchain protocol has only a few block-builders (but lots of validators)?

- block-building might be an intrinsically specialized skill
- at least builders don't control consensus, right?
  - block proposer could always propose their own block if they prefer

Centralization and Censorship

Question: big problem if a blockchain protocol has only a few validators? [answer: yes, largely defeats the point of a blockchain protocol]

Question: big problem if a blockchain protocol has only a few block-builders (but lots of validators)?

- block-building might be an intrinsically specialized skill
- at least builders don't control consensus, right?
  - block proposer could always propose their own block if they prefer

One issue: censorship --- i.e., systematic exclusion of certain txs.

- e.g., for financial or legal/regulatory reasons

Censorship-Resistance

Question: big problem if a blockchain protocol has only a few block-builders (but lots of validators)?

One issue: censorship --- i.e., systematic exclusion of certain txs.

Open question: how to mitigate censorship risks.

Censorship-Resistance

Question: big problem if a blockchain protocol has only a few block-builders (but lots of validators)?

One issue: censorship --- i.e., systematic exclusion of certain txs.

Open question: how to mitigate censorship risks.

- idea #1: inclusion lists (IL) --- let validators designate txs whose inclusion is part of block validity (cf., forced inclusion in rollups)

Censorship-Resistance

Question: big problem if a blockchain protocol has only a few block-builders (but lots of validators)?

One issue: censorship --- i.e., systematic exclusion of certain txs.

Open question: how to mitigate censorship risks.

- idea #1: inclusion lists (IL) --- let validators designate txs whose inclusion is part of block validity (cf., forced inclusion in rollups)
- idea #2: multiple concurrent proposers (MCP) --- take union of multiple validator block proposals ➔ censoring requires large bribes to multiple validators [Fox/Pai/Resnick 23]

Bitcoin in a Nutshell

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
    - pros: simple, scales well, consistent + live in synchrony w/majority honest hashrate

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
    - pros: simple, scales well, consistent + live in synchrony w/majority honest hashrate
    - cons: lose safety in partial synchrony, guarantees only probabilistic, large latency

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
    - pros: simple, scales well, consistent + live in synchrony w/majority honest hashrate
    - cons: lose safety in partial synchrony, guarantees only probabilistic, large latency
- execution layer: UTXOs (unspent tx outputs)
  - tx = inputs (previously created UTXOs) and outputs (new UTXO)

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
    - pros: simple, scales well, consistent + live in synchrony w/majority honest hashrate
    - cons: lose safety in partial synchrony, guarantees only probabilistic, large latency
- execution layer: UTXOs (unspent tx outputs)
  - tx = inputs (previously created UTXOs) and outputs (new UTXO)
  - user signatures = ECDSA or (post-Taproot) Schnorr

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
    - pros: simple, scales well, consistent + live in synchrony w/majority honest hashrate
    - cons: lose safety in partial synchrony, guarantees only probabilistic, large latency
- execution layer: UTXOs (unspent tx outputs)
  - tx = inputs (previously created UTXOs) and outputs (new UTXO)
  - user signatures = ECDSA or (post-Taproot) Schnorr
  - tx fee = first-price auction (“bid” = diff between value of inputs, outputs)

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
    - pros: simple, scales well, consistent + live in synchrony w/majority honest hashrate
    - cons: lose safety in partial synchrony, guarantees only probabilistic, large latency
- execution layer: UTXOs (unspent tx outputs)
  - tx = inputs (previously created UTXOs) and outputs (new UTXO)
  - user signatures = ECDSA or (post-Taproot) Schnorr
  - tx fee = first-price auction (“bid” = diff between value of inputs, outputs)
  - txs disseminated via public mempool (implemented via gossip protocol)

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
- execution layer: UTXO-based (UTXO = “unspent tx output”)
  - tx = inputs (previously created UTXOs) and outputs (new UTXO)
  - user signatures = ECDSA or (post-Taproot) Schnorr
  - tx fee = first-price auction (“bid” = diff between value of inputs, outputs)
  - txs disseminated via public mempool (implemented via gossip protocol)
- light clients (commit to txs in block header via Merkle root)

Bitcoin in a Nutshell

- consensus layer: Nakamoto consensus
  - longest-chain consensus (detail: “longest” = “most amount of work”)
  - proof-of-work sybil-resistance
- execution layer: UTXO-based (UTXO = “unspent tx output”)
  - tx = inputs (previously created UTXOs) and outputs (new UTXO)
  - user signatures = ECDSA or (post-Taproot) Schnorr
  - tx fee = first-price auction (“bid” = diff between value of inputs, outputs)
  - txs disseminated via public mempool (implemented via gossip protocol)
- light clients (commit to txs in block header via Merkle root)
- misc. trivia/lore (blocksize wars, SegWit, etc.)

Ethereum in a Nutshell (Consensus)

- consensus layer:

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)
  - backbone = longest-chain-type consensus (view length = 12 seconds)

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)
  - backbone = longest-chain-type consensus (view length = 12 seconds)
    - leader sequence chosen using a VRF (derived from BLS signatures)

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)
  - backbone = longest-chain-type consensus (view length = 12 seconds)
    - leader sequence chosen using a VRF (derived from BLS signatures)
    - “LMD-GHOST” = idiosyncratic fork choice rule (not longest-chain)

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)
  - backbone = longest-chain-type consensus (view length = 12 seconds)
    - leader sequence chosen using a VRF (derived from BLS signatures)
    - "LMD-GHOST" = idiosyncratic fork choice rule (not longest-chain)
  - "finality gadget" = "Casper" ≈ Tendermint (but pipelined)

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)
  - backbone = longest-chain-type consensus (view length = 12 seconds)
    - leader sequence chosen using a VRF (derived from BLS signatures)
    - "LMD-GHOST" = idiosyncratic fork choice rule (not longest-chain)
  - "finality gadget" = "Casper" ≈ Tendermint (but pipelined)
    - effective view length = 32 slots = 6.4 minutes
      - need time for e.g. signature aggregation/verification for > 1M validators!

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)
  - backbone = longest-chain-type consensus (view length = 12 seconds)
    - leader sequence chosen using a VRF (derived from BLS signatures)
    - "LMD-GHOST" = idiosyncratic fork choice rule (not longest-chain)
  - "finality gadget" = "Casper" ≈ Tendermint (but pipelined)
    - effective view length = 32 slots = 6.4 minutes
      - need time for e.g. signature aggregation/verification for > 1M validators!
    - consistent + live in partial synchrony with 67% honest stake

Ethereum in a Nutshell (Consensus)

- consensus layer:
  - proof-of-stake sybil-resistance (1 validator = 32 ETH)
  - backbone = longest-chain-type consensus (view length = 12 seconds)
    - leader sequence chosen using a VRF (derived from BLS signatures)
    - “LMD-GHOST” = idiosyncratic fork choice rule (not longest-chain)
  - “finality gadget” = “Casper” ≈ Tendermint (but pipelined)
    - effective view length = 32 slots = 6.4 minutes
      - need time for e.g. signature aggregation/verification for > 1M validators!
    - consistent + live in partial synchrony with 67% honest stake
    - slashing for consistency, liveness violations

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts
  - account = balance + (byte)code + data (+ nonce, to avoid replay attacks)

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts
  - account = balance + (byte)code + data (+ nonce, to avoid replay attacks)
  - accounts organized in Merkle-Patricia tree (“state root” = root of MPT)

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts
  - account = balance + (byte)code + data (+ nonce, to avoid replay attacks)
  - accounts organized in Merkle-Patricia tree (“state root” = root of MPT)
  - tx = ETH transfer or contract function call

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts
  - account = balance + (byte)code + data (+ nonce, to avoid replay attacks)
  - accounts organized in Merkle-Patricia tree (“state root” = root of MPT)
  - tx = ETH transfer or contract function call
  - EVM used to execute (byte)code, modify blockchain state as needed

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts
  - account = balance + (byte)code + data (+ nonce, to avoid replay attacks)
  - accounts organized in Merkle-Patricia tree ("state root" = root of MPT)
  - tx = ETH transfer or contract function call
  - EVM used to execute (byte)code, modify blockchain state as needed
  - computation measured (line-by-line of bytecode) in "gas"

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts
  - account = balance + (byte)code + data (+ nonce, to avoid replay attacks)
  - accounts organized in Merkle-Patricia tree (“state root” = root of MPT)
  - tx = ETH transfer or contract function call
  - EVM used to execute (byte)code, modify blockchain state as needed
  - computation measured (line-by-line of bytecode) in “gas”
  - tx fees = EIP-1559 (protocol-computed base fee (burned) + priority fees)

Ethereum in a Nutshell (Execution)

- execution layer: account-based state + EVM virtual machine
  - state = set of accounts
  - account = balance + (byte)code + data (+ nonce, to avoid replay attacks)
  - accounts organized in Merkle-Patricia tree (“state root” = root of MPT)
  - tx = ETH transfer or contract function call
  - EVM used to execute (byte)code, modify blockchain state as needed
  - computation measured (line-by-line of bytecode) in “gas”
  - tx fees = EIP-1559 (protocol-computed base fee (burned) + priority fees)
  - tx dissemination via public mempool (gossipsub) or sent directly (as private order flow) to searchers/builders

Ethereum in a Nutshell (Rollups)

- scaling: “rollup-centric roadmap”

Ethereum in a Nutshell (Rollups)

- scaling: “rollup-centric roadmap”
  - use L1 (i.e., Ethereum) for DA (i.e., data availability of rollup txs)
    - anyone can run full node for rollup, e.g. detect bogus state roots
    - post EIP-4844: rollup tx descriptions in “blobs” (expire after 18 days), KZG commitments to blobs archived

Ethereum in a Nutshell (Rollups)

- scaling: “rollup-centric roadmap”
  - use L1 (i.e., Ethereum) for DA (i.e., data availability of rollup txs)
    - anyone can run full node for rollup, e.g. detect bogus state roots
    - post EIP-4844: rollup tx descriptions in “blobs” (expire after 18 days), KZG commitments to blobs archived
  - key challenge: state root verification (SRV) problem

Ethereum in a Nutshell (Rollups)

- scaling: “rollup-centric roadmap”
  - use L1 (i.e., Ethereum) for DA (i.e., data availability of rollup txs)
    - anyone can run full node for rollup, e.g. detect bogus state roots
    - post EIP-4844: rollup tx descriptions in “blobs” (expire after 18 days), KZG commitments to blobs archived
  - key challenge: state root verification (SRV) problem
  - optimistic rollups: innocent until proven guilty by challenger who wins a dispute-resolution game (“fault proofs”) (➔ sequencer’s stake is slashed)
    - L1 re-executes a single line of bytecode to determine the winner

Ethereum in a Nutshell (Rollups)

- scaling: “rollup-centric roadmap”
  - use L1 (i.e., Ethereum) for DA (i.e., data availability of rollup txs)
    - anyone can run full node for rollup, e.g. detect bogus state roots
    - post EIP-4844: rollup tx descriptions in “blobs” (expire after 18 days), KZG commitments to blobs archived
  - key challenge: state root verification (SRV) problem
  - optimistic rollups: innocent until proven guilty by challenger who wins a dispute-resolution game (“fault proofs”) (➔ sequencer’s stake is slashed)
    - L1 re-executes a single line of bytecode to determine the winner
  - validity rollups: guilty until proven innocent with a SNARK for SRV
    - L1 verifies SNARK directly

Ethereum: Looking Ahead

Challenge: scale better (higher throughput, lower latency).

- hardest part = scaling execution

Ethereum: Looking Ahead

Challenge: scale better (higher throughput, lower latency).

- hardest part = scaling execution

Approach #1: increase capacity to support rollups.

- e.g., allow more blobs per block for additional rollup DA

Ethereum: Looking Ahead

Challenge: scale better (higher throughput, lower latency).

- hardest part = scaling execution

Approach #1: increase capacity to support rollups.

- e.g., allow more blobs per block for additional rollup DA

Approach #2: directly scale the core Ethereum protocol.

- e.g., 100x throughput (1.8 billion gas/block), lower latency
- "preconfirmations" for sub-second latency (even with > 1M validators)
- see e.g. Justin Drake's CBER seminar on May 8th

Scaling the Core Ethereum Protocol

One approach: "SNARK-ify everything."

- $\approx$ turns Ethereum into a validity rollup of itself

Scaling the Core Ethereum Protocol

One approach: "SNARK-ify everything."

- ≈ turns Ethereum into a validity rollup of itself

- validators outsource execution to builders (≈ rollup sequencers)

Scaling the Core Ethereum Protocol

One approach: "SNARK-ify everything."

- ≈ turns Ethereum into a validity rollup of itself

- validators outsource execution to builders (≈ rollup sequencers)
- builders include (L1) tx data (in blobs) + SNARK proof in block
  - perhaps collaborating with 3^{rd}-party provers to do this quickly

Scaling the Core Ethereum Protocol

One approach: "SNARK-ify everything."

- ≈ turns Ethereum into a validity rollup of itself

- validators outsource execution to builders (≈ rollup sequencers)
- builders include (L1) tx data (in blobs) + SNARK proof in block
  - perhaps collaborating with 3rd-party provers to do this quickly
  - Ethereum validators now effectively become stateless
    - don't need blockchain state to verify block validity (just check SNARK)
    - like stateless clients, but still with consensus voting power

Scaling the Core Ethereum Protocol (con'd)

One approach: "SNARK-ify everything."

- validators outsource execution to builders (≈ rollup sequencers)
- builders include (L1) tx data (in blobs) + SNARK proof in block

Open issues:

Scaling the Core Ethereum Protocol (con’d)

One approach: “SNARK-ify everything.”

- validators outsource execution to builders (≈ rollup sequencers)
- builders include (L1) tx data (in blobs) + SNARK proof in block

Open issues:

1. Which type(s) of SNARKs should be accepted?
   - SNARK verification now enshrined in core protocol, not a smart contract

Scaling the Core Ethereum Protocol (con'd)

One approach: "SNARK-ify everything."

- validators outsource execution to builders (≈ rollup sequencers)
- builders include (L1) tx data (in blobs) + SNARK proof in block

Open issues:

1. Which type(s) of SNARKs should be accepted?
   - SNARK verification now enshrined in core protocol, not a smart contract
2. Redesign the EVM to make SNARK proving/verification easier?
   - e.g., recent discussion around moving from EVM to RISC V

Scaling the Core Ethereum Protocol (con’d)

One approach: “SNARK-ify everything.”

- validators outsource execution to builders ($\approx$ rollup sequencers)
- builders include (L1) tx data (in blobs) + SNARK proof in block

Open issues:

1. Which type(s) of SNARKs should be accepted?
   – SNARK verification now enshrined in core protocol, not a smart contract
2. Redesign the EVM to make SNARK proving/verification easier?
   – e.g., recent discussion around moving from EVM to RISC V
3. Will SNARK generation ever be fast enough for this vision?

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

- buzzword: “zk co-processors”

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

- buzzword: “zk co-processors”

Example: ML inference.

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

- buzzword: “zk co-processors”

Example: ML inference.

- commitment z to a (possibly large) model posted on-chain

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

- buzzword: "zk co-processors"

Example: ML inference.

- commitment z to a (possibly large) model posted on-chain
- for query x and alleged output y, SNARK asserts existence of model M with h(M) = z and M(x) = y

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

- buzzword: “zk co-processors”

Example: ML inference.

- commitment z to a (possibly large) model posted on-chain
- for query x and alleged output y, SNARK asserts existence of model M with h(M) = z and M(x) = y
- upshot: results of complex inference queries can be verifiably posted on-chain (no trust required)

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

Example: ML inference.

- commitment z to a (possibly large) model posted on-chain
- for query x and alleged output y, SNARK asserts existence of model M with h(M) = z and M(x) = y
- or even ML training:

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

Example: ML inference.

- commitment z to a (possibly large) model posted on-chain
- for query x and alleged output y, SNARK asserts existence of model M with h(M) = z and M(x) = y
- or even ML training: w.r.t. on-chain commitment w to some data set and commitment z to alleged training output, SNARK asserts existence of D and M s.t. (i) h(D) = w; (ii) h(M) = z;

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

Example: ML inference.

- commitment z to a (possibly large) model posted on-chain
- for query x and alleged output y, SNARK asserts existence of model M with h(M) = z and M(x) = y
- or even ML training: w.r.t. on-chain commitment w to some data set and commitment z to alleged training output, SNARK asserts existence of D and M s.t. (i) h(D) = w; (ii) h(M) = z; and (iii) training a model (with an agreed-upon algorithm) with training data D results in the model M

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

Example: ML inference (and maybe even ML training).

Example: “zkTLS.”

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

Example: ML inference (and maybe even ML training).

Example: "zkTLS."

- TLS: how two parties can communicate securely over Internet
  - e.g. client logging into a Web site, performs operations on its account
  - alas, servers do not generally sign their messages

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

Example: ML inference (and maybe even ML training).

Example: "zkTLS."

- TLS: how two parties can communicate securely over Internet
  - e.g. client logging into a Web site, performs operations on its account
  - alas, servers do not generally sign their messages
- zkTLS: proving results of such a secure communication to a third party (without e.g. leaking your login password)

Further Directions (Speculative)

Note: SNARKs can be used to prove correctness of *any* (NP) computation, not just state root verification.

Example: ML inference (and maybe even ML training).

Example: "zkTLS."

- TLS: how two parties can communicate securely over Internet
- zkTLS: proving results of such a secure communication to a third party (without e.g. leaking your login password)
  - one application: getting Web2 data on-chain in verifiable way

Epilogue

Benefits of learning a mature field:

- agreed-upon models and definitions
- agreement on the most important open problems
- agreement on what constitutes a “solution” or “progress”
- shared language and knowledge base
- comprehensive textbooks, MOOCs, etc.

Epilogue

~~Benefits~~ Challenges of learning ~~a mature field~~ about blockchains:

- ~~agreed-upon models and definitions~~
- ~~agreement on the most important open problems~~
- ~~agreement on what constitutes a “solution” or “progress”~~
- ~~shared language and knowledge base~~
- ~~comprehensive textbooks, MOOCs, etc.~~

Epilogue

~~Benefits~~ Challenges of learning ~~a mature field~~ about blockchains:

- ~~agreed-upon models and definitions~~
- ~~agreement on the most important open problems~~
- ~~agreement on what constitutes a "solution" or "progress"~~
- ~~shared language and knowledge base~~
- ~~comprehensive textbooks, MOOCs, etc.~~

Opportunity: get in on the ground floor, shape the technology!

Upcoming Conferences in NYC

- May 12-13 @ Columbia: TLDR (The Latest in DeFi Research)
- May 19-23 in NYC: Accelerate (Solana)
- June 24-26 in Brooklyn: Permissionless IV
- August 4-6 @ Berkeley: Science of Blockchains Conference (SBC)
- December @ Columbia: Columbia Cryptoeconomics Workshop
- etc.

THANKS!