Lecture #20: Permissionless Consensus and Proof-of-Work

COMS 4995-001: The Science of Blockchains

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

Tim Roughgarden

Goals for Lecture #20

1. Introduction to permissionless consensus protocols.
   - will relax the assumption of a fixed and known validator set
2. Sybil attacks.
   - permissionless ➔ one participant can masquerade as many
3. Proof-of-work (PoW).
   - sybil-resistant method of selecting a random validator as leader
   - winner = first validator to partially invert a cryptographic hash function
4. Combining PoW with longest-chain (👍) or Tendermint (👎).

Permissioned Consensus

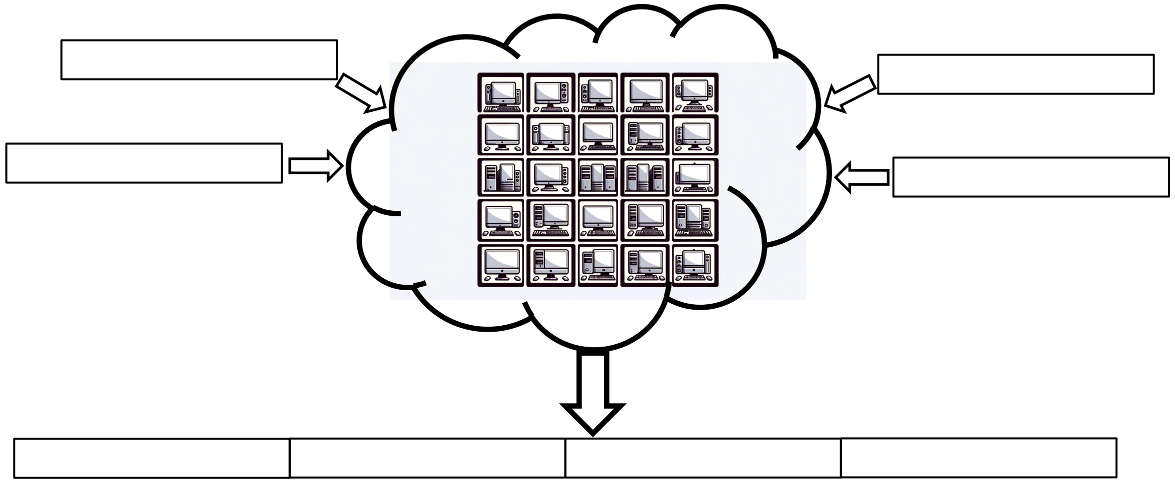
Consensus: keep validators in sync, despite failures and attacks.

Permissioned Consensus

Consensus: keep validators in sync, despite failures and attacks.

State machine replication (SMR): clients submit txs to validators.

- want protocol that guarantees consistency and liveness
  - no disagreements across validators or time, all txs eventually included



Permissioned Consensus

Consensus: keep validators in sync, despite failures and attacks.

State machine replication (SMR): clients submit txs to validators.

- want protocol that guarantees consistency and liveness
  - no disagreements across validators or time, all txs eventually included

Examples: Tendermint (quorums), longest-chain consensus.

Permissioned Consensus

Consensus: keep validators in sync, despite failures and attacks.

State machine replication (SMR): clients submit txs to validators.

- want protocol that guarantees consistency and liveness
  - no disagreements across validators or time, all txs eventually included

Examples: Tendermint (quorums), longest-chain consensus.

Standing assumption in Parts I + II: fixed and known set of n validators, each with known name, public key, and IP address.

- a.k.a. “permissioned” or “proof-of-authority” protocols

Permissionless Consensus

Permissionless setting: physical machines can enter/exit the validator set at any time.

Permissionless Consensus

Permissionless setting: physical machines can enter/exit the validator set at any time.

- communication often via gossip protocol (rather than point-to-point)
- maximally "decentralized" version of the "computer in the sky"

Permissionless Consensus

Permissionless setting: physical machines can enter/exit the validator set at any time.

- communication often via gossip protocol (rather than point-to-point)
- maximally "decentralized" version of the "computer in the sky"

Question: is consensus even possible in the permissionless setting? E.g., can we extend our permissioned protocols to it?

Permissionless Consensus

Permissionless setting: physical machines can enter/exit the validator set at any time.

- communication often via gossip protocol (rather than point-to-point)
- maximally “decentralized” version of the “computer in the sky”

Question: is consensus even possible in the permissionless setting? E.g., can we extend our permissioned protocols to it?

- issue for Tendermint: how many votes constitute a quorum? (n unknown)

Permissionless Consensus

Permissionless setting: physical machines can enter/exit the validator set at any time.

- communication often via gossip protocol (rather than point-to-point)
- maximally “decentralized” version of the “computer in the sky”

Question: is consensus even possible in the permissionless setting? E.g., can we extend our permissioned protocols to it?

- issue for Tendermint: how many votes constitute a quorum? (n unknown)
- issue for Tendermint and longest-chain: how to choose the leader of a view? (unknown validator set ➔ what does “round-robin order” mean?)

Sybil Attacks

Issue: how to choose the leader (i.e., block proposer) of a view?

Sybil Attacks

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: each view, choose a new leader at random.

Sybil Attacks

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: each view, choose a new leader at random.

Question: from which distribution? Uniformly at random?

Sybil Attacks

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: each view, choose a new leader at random.

Question: from which distribution? Uniformly at random?

Issue: “Sybil” attacks.

- single physical machine could masquerade as many via multiple pks

Sybil Attacks

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: each view, choose a new leader at random.

Question: from which distribution? Uniformly at random?

Issue: “Sybil” attacks.

- single physical machine could masquerade as many via multiple pks
- sybil attack ➔ can boost probability of being selected

Sybil Attacks

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: each view, choose a new leader at random.

Question: from which distribution? Uniformly at random?

Issue: “Sybil” attacks.

- single physical machine could masquerade as many via multiple pks
- sybil attack ➔ can boost probability of being selected
- need sybil-proof random sampling method
  - probability of selection independent of number of pks used in protocol

Sybil-Proof Random Sampling

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: sybil-proof random sampling method.

– probability of selection independent of number of pks used in protocol

Sybil-Proof Random Sampling

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: sybil-proof random sampling method.

– probability of selection independent of number of pks used in protocol

Two dominant approaches:

Sybil-Proof Random Sampling

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: sybil-proof random sampling method.

– probability of selection independent of number of pks used in protocol

Two dominant approaches:

– proof-of-work (PoW) (this week): sample with probability proportional to the amount of computational power contributed to protocol

Sybil-Proof Random Sampling

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: sybil-proof random sampling method.

– probability of selection independent of number of pks used in protocol

Two dominant approaches:

– proof-of-work (PoW) (this week): sample with probability proportional to the amount of computational power contributed to protocol

– proof-of-stake (PoS) (next week): sample with probability proportional to the amount of collateral (i.e., locked-up stake) contributed to protocol

Sybil-Proof Random Sampling

Issue: how to choose the leader (i.e., block proposer) of a view?

Solution: sybil-proof random sampling method.

- probability of selection independent of number of pks used in protocol

Two dominant approaches:

- proof-of-work (PoW) (this week): sample with probability proportional to the amount of computational power contributed to protocol
- proof-of-stake (PoS) (next week): sample with probability proportional to the amount of collateral (i.e., locked-up stake) contributed to protocol
- NB: PoW/PoS are sybil-resistance mechanisms, not consensus protocols

Proof-of-Work

Idea: to propose next block, validator must solve a hard puzzle.

- publish solution along with a block proposal (substitutes for a signature)

Proof-of-Work

Idea: to propose next block, validator must solve a hard puzzle.

– publish solution along with a block proposal (substitutes for a signature)

Canonical hard puzzle: for a threshold t, find a valid x with $h(x) \le t$.

– h = cryptographic hash function (like SHA-256)

– t = difficulty parameter (auto-tuned by protocol to get desired block rate)

Proof-of-Work

Idea: to propose next block, validator must solve a hard puzzle.

- publish solution along with a block proposal (substitutes for a signature)

Canonical hard puzzle: for a threshold t, find a valid x with $h(x) \leq t$.

- h = cryptographic hash function (like SHA-256)
- t = difficulty parameter (auto-tuned by protocol to get desired block rate)

Proof-of-work: next leader = first validator to find puzzle solution x.

Proof-of-Work

Idea: to propose next block, validator must solve a hard puzzle.

– publish solution along with a block proposal (substitutes for a signature)

Canonical hard puzzle: for a threshold t, find a valid x with $h(x) \le t$.

– h = cryptographic hash function (like SHA-256)

– t = difficulty parameter (auto-tuned by protocol to get desired block rate)

Proof-of-work: next leader = first validator to find puzzle solution x.

– h as good as random ➔ only solution approach = repeated guessing

Proof-of-Work

Idea: to propose next block, validator must solve a hard puzzle.

- publish solution along with a block proposal (substitutes for a signature)

Canonical hard puzzle: for a threshold t, find a valid x with $h(x) \leq t$.

- h = cryptographic hash function (like SHA-256)
- t = difficulty parameter (auto-tuned by protocol to get desired block rate)

Proof-of-work: next leader = first validator to find puzzle solution x.

- h as good as random ➔ only solution approach = repeated guessing
- h as good as random ➔ each guess equally likely to be a solution

Proof-of-Work

Idea: to propose next block, validator must solve a hard puzzle.

– publish solution along with a block proposal (substitutes for a signature)

Canonical hard puzzle: for a threshold t, find a valid x with $h(x) \leq t$.

– h = cryptographic hash function (like SHA-256)

– t = difficulty parameter (auto-tuned by protocol to get desired block rate)

Proof-of-work: next leader = first validator to find puzzle solution x.

– h as good as random ➔ only solution approach = repeated guessing

– h as good as random ➔ each guess equally likely to be a solution

– probability of selection proportional to hashrate (sybil-resistant!)

Longest-Chain Consensus Revisited

Recall: (permissioned) longest-chain consensus.

- B_0 = "genesis block"

Longest-Chain Consensus Revisited

Recall: (permissioned) longest-chain consensus.

- B_0 = “genesis block”
- define view = Δ timesteps

Longest-Chain Consensus Revisited

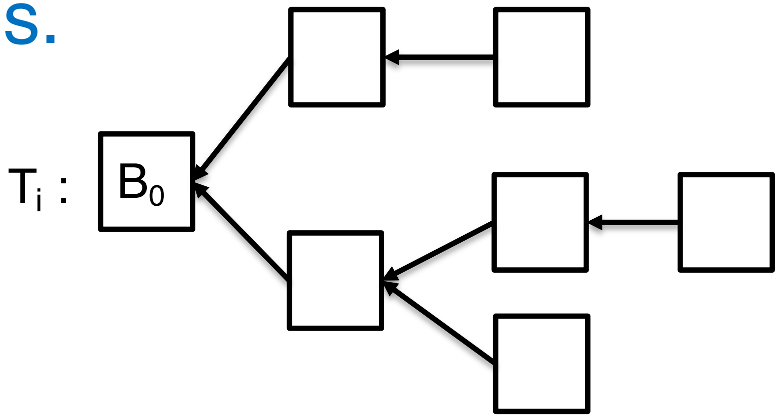
Recall: (permissioned) longest-chain consensus.

- B_0 = “genesis block”
- define view = Δ timesteps
- validators take turns as leader

Longest-Chain Consensus Revisited

Recall: (permissioned) longest-chain consensus.

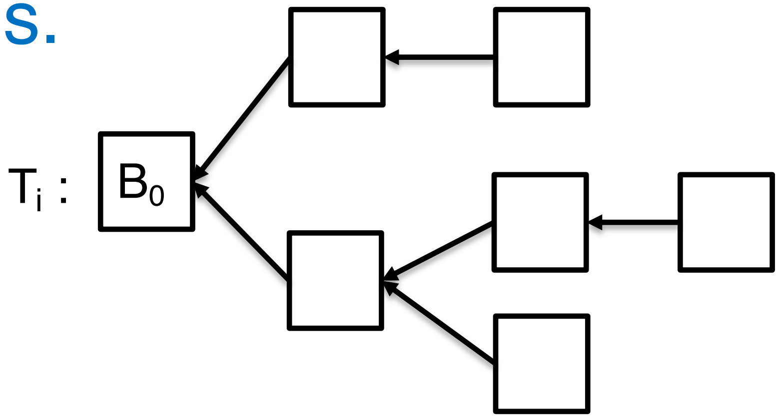
- B_0 = "genesis block"
- define view = Δ timesteps
- validators take turns as leader
- validator i maintains in-tree T_i of valid blocks, rooted at B_0



Longest-Chain Consensus Revisited

Recall: (permissioned) longest-chain consensus.

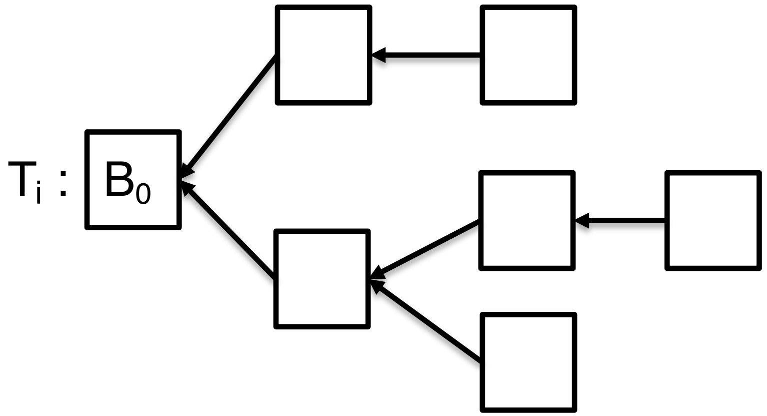
- B_0 = "genesis block"
- define view = Δ timesteps
- validators take turns as leader
- validator i maintains in-tree T_i of valid blocks, rooted at B_0
  - block B is valid in view v if:
    - annotated with a view v' $\leq$ v
    - signed by leader of view v'
    - annotated with (hash of header of) predecessor block B'' from a view v'' < v'
    - contains only valid txs



Longest-Chain Consensus (con'd)

Recall: (permissioned) longest-chain consensus.

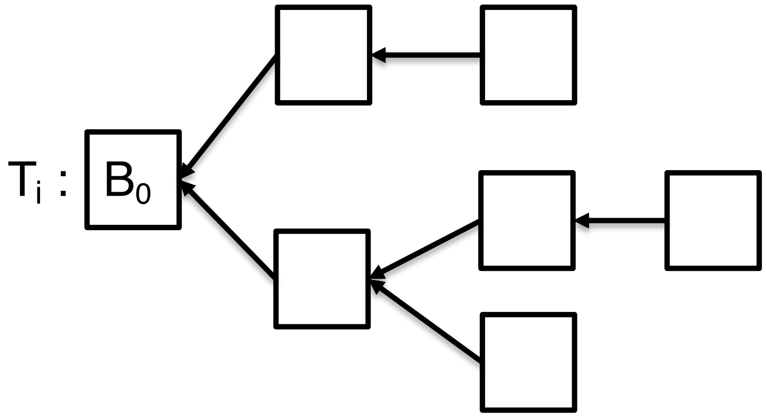
- in view v:



Longest-Chain Consensus (con'd)

Recall: (permissioned) longest-chain consensus.

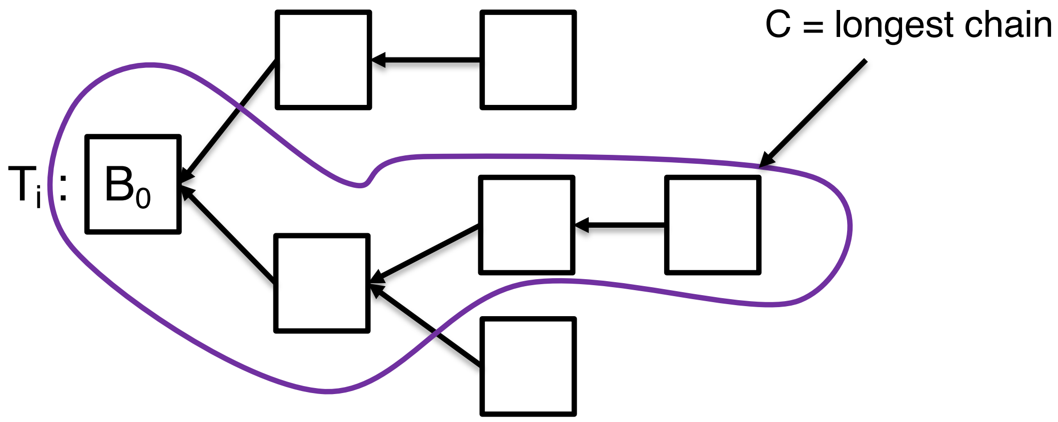
- in view v:
  - let ℓ = leader of view



Longest-Chain Consensus (con'd)

Recall: (permissioned) longest-chain consensus.

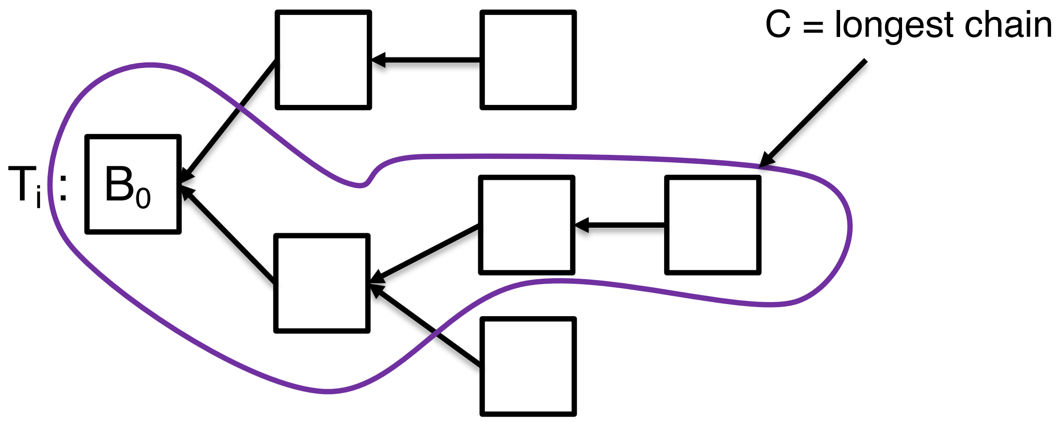
- in view v:
  - let ℓ = leader of view
  - let C = longest chain in ℓ's in-tree



Longest-Chain Consensus (con'd)

Recall: (permissioned) longest-chain consensus.

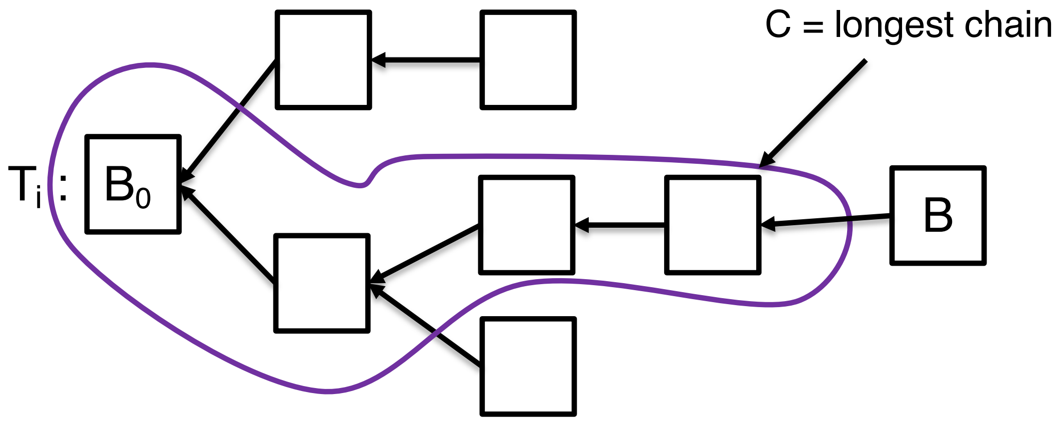
- in view v:
  - let ℓ = leader of view
  - let C = longest chain in ℓ's in-tree
  - let B := all not-yet-included (in C) txs ℓ knows about



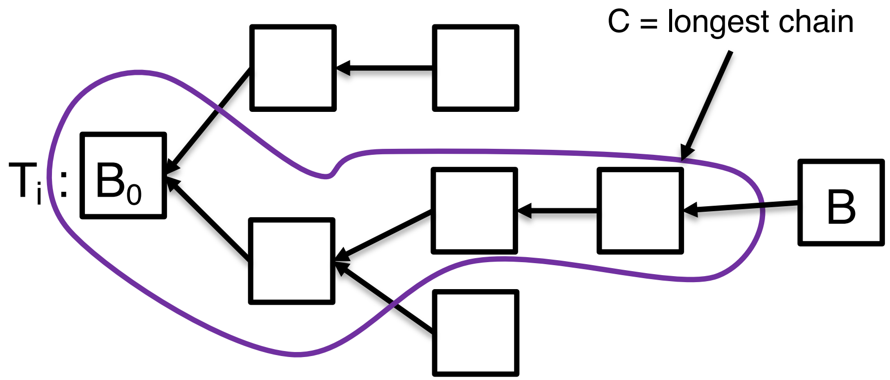
Longest-Chain Consensus (con'd)

Recall: (permissioned) longest-chain consensus.

- in view v:
  - let ℓ = leader of view
  - let C = longest chain in ℓ's in-tree
  - let B := all not-yet-included (in C) txs ℓ knows about
  - ℓ adds B to its in-tree (extending C)
  - ℓ sends B to all other validators



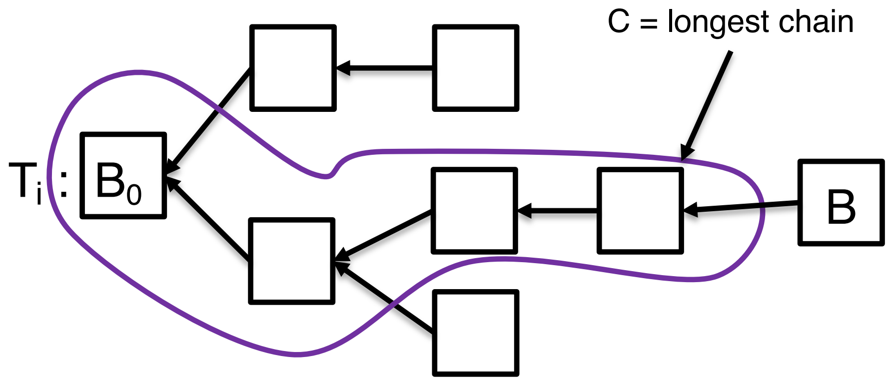
Nakamoto Consensus



Nakamoto consensus: longest-chain consensus + PoW leader selection.

- selection probability proportional to hashrate
- random length of a view = time for someone to solve puzzle

Nakamoto Consensus



Nakamoto consensus: longest-chain consensus + PoW leader selection.

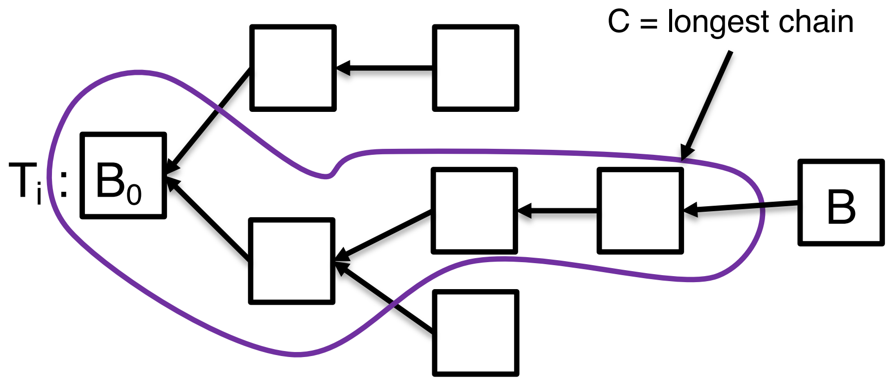
– selection probability proportional to hashrate

– random length of a view = time for someone to solve puzzle

Puzzle format: find x with h(x) ≤ t where:

– h = cryptographic hash fn, t = difficulty parameter (both protocol-defined)

Nakamoto Consensus



Nakamoto consensus: longest-chain consensus + PoW leader selection.

- selection probability proportional to hashrate
- random length of a view = time for someone to solve puzzle

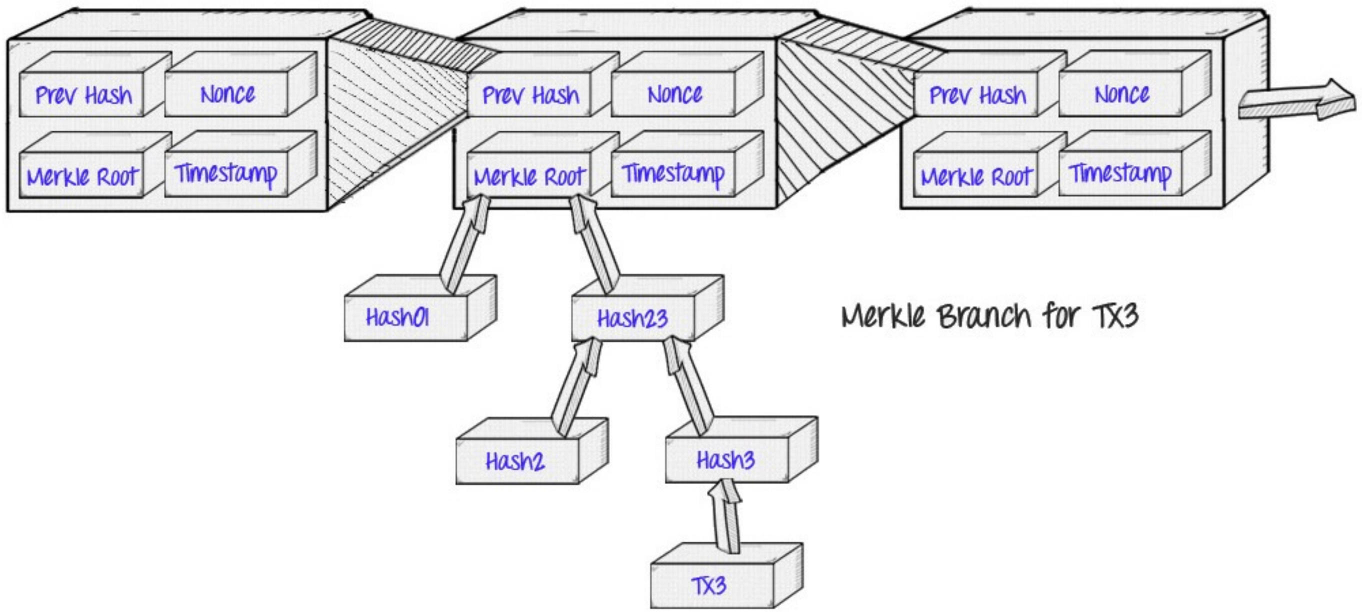
Puzzle format: find x with $h(x) \leq t$ where:

- h = cryptographic hash fn, t = difficulty parameter (both protocol-defined)
- x = block header of the form…

Recall: Bitcoin Block Headers

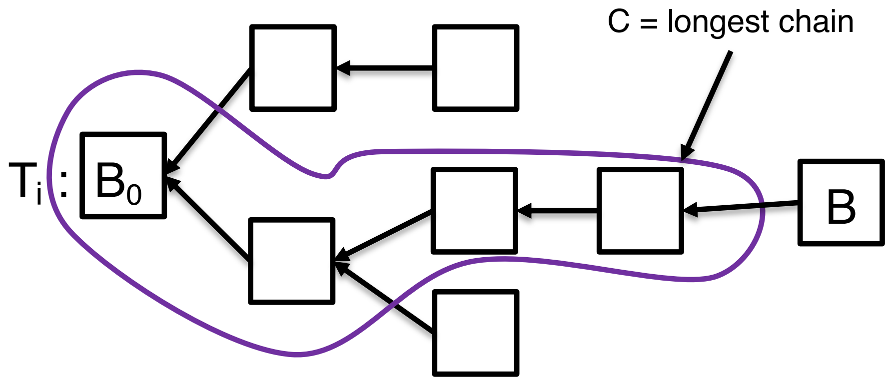
In Bitcoin: each block includes Merkle root of its txs (as metadata).

- block name = hash of its metadata ("block header"), not of entire block
- block name depends on each of its txs via Merkle root in block header



From "Merkling in Ethereum" by Vitalik Buterin https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/

Nakamoto Consensus



Nakamoto consensus: longest-chain consensus + PoW leader selection.

- selection probability proportional to hashrate
- random length of a view = time for someone to solve puzzle

Puzzle format: find x with $h(x) \leq t$ where:

- h = cryptographic hash fn, t = difficulty parameter (both protocol-defined)
- x = block header of the form < tx Merkle root || pred || pk || nonce >
  - point of the nonce: “grind” through possibilities until find a solution

Some Recent Bitcoin Blocks

Block 891328

00000000000000000001f5831d1c2c6d9ed1536b6d66ef1164c462250b75d36a

Block 891329

000000000000000000024eec6c0038843111d81decbf074e916401aee9eaef52

Block 891330

000000000000000000003c52639327ede237bf41b17ff73dcde093fde88ce579

Block 891331

0000000000000000000013029bd213f663a5b4472242efe0e74e2c3636153af78

Block 891332

0000000000000000000017296d2ac7ffde35a988c17c6d8728a3ed0036853d796

Some Recent Bitcoin Blocks

Block 891332

0000000000000000000017296d2ac7ffde35a988c17c6d8728a3ed0036853d796

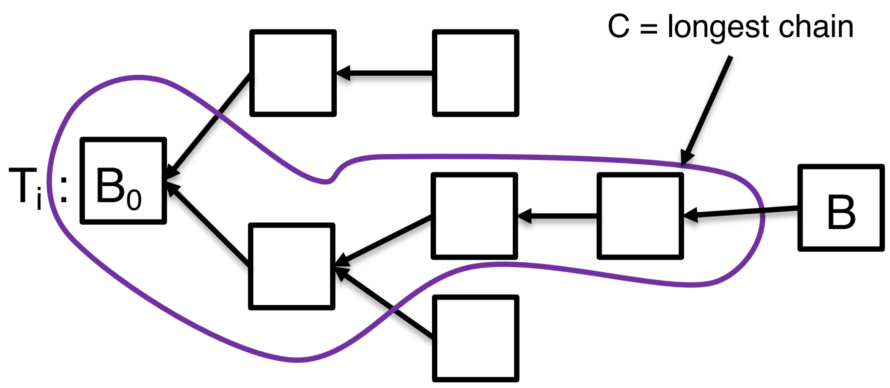
Block was mined on 2025-04-07 07:01:06 GMT -4. It has 1 confirmation on the Bitcoin blockchain. There are 2800 transactions in block 891332.

PREVIOUS

DETAILS

HEIGHT	891332
STATUS	In best chain (1 confirmation)
TIMESTAMP	2025-04-07 07:01:06 GMT -4
SIZE	1617.481 KB
VIRTUAL SIZE	999 vKB
WEIGHT UNITS	3993.379 KWU
VERSION	0x23bdc000
MERKLE ROOT	1601e7d398911c9a6fa501e372fe84231acfbc5db9bf89aab5d43f8ad0b061ea
BITS	0x17025105
DIFFICULTY	121507793131898.06
NONCE	0x7951a124

Nakamoto Consensus



Nakamoto consensus: longest-chain consensus + PoW leader selection.

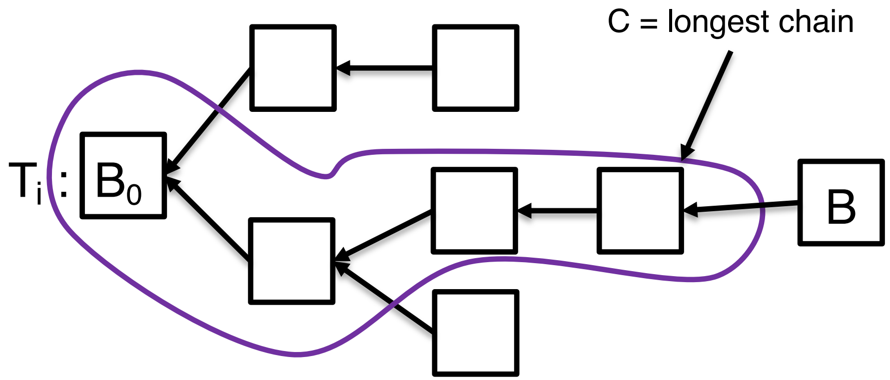
– selection probability proportional to hashrate

Puzzle format: find x with $h(x) \leq t$ where:

– h = cryptographic hash fn, t = difficulty parameter (both protocol-defined)

– x = block header of the form < tx Merkle root II pred II pk II nonce >

Nakamoto Consensus



Nakamoto consensus: longest-chain consensus + PoW leader selection.

– selection probability proportional to hashrate

Puzzle format: find x with $h(x) \leq t$ where:

– h = cryptographic hash fn, t = difficulty parameter (both protocol-defined)

– x = block header of the form < tx Merkle root ll pred ll pk ll nonce >

Note: unlike permissioned version, impossible for validator to:

– specify multiple predecessors for a block (i.e., equivocate)

– specify a predecessor from a later view

Guarantees for Longest-Chain Consensus

Confirmation rule: for a security parameter $k \geq 1$, finalized txs = all txs in the longest chain, except for those in the last k blocks.

Guarantees for Longest-Chain Consensus

Confirmation rule: for a security parameter $k \geq 1$, finalized txs = all txs in the longest chain, except for those in the last k blocks.

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay $\leq \Delta$).

Guarantees for Longest-Chain Consensus

Confirmation rule: for a security parameter $k \geq 1$, finalized txs = all txs in the longest chain, except for those in the last k blocks.

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay $\leq \Delta$).
2. < 50% Byzantine validators.

Guarantees for Longest-Chain Consensus

Confirmation rule: for a security parameter $k \geq 1$, finalized txs = all txs in the longest chain, except for those in the last k blocks.

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay $\leq \Delta$).
2. < 50% Byzantine validators.
3. k large enough that, in every interval of $\geq 2k+2$ views, a strict majority of the leaders are honest. [e.g., (n/2)-1 suffices]

Guarantees for Longest-Chain Consensus

Confirmation rule: for a security parameter $k \geq 1$, finalized txs = all txs in the longest chain, except for those in the last k blocks.

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay $\leq \Delta$).
2. < 50% Byzantine validators.
3. k large enough that, in every interval of $\geq 2k+2$ views, a strict majority of the leaders are honest. [e.g., (n/2)-1 suffices]

Recall guarantee: under these assumptions, (permissioned) longest-chain consensus is consistent and live.

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay ≤ Δ).

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay $\leq \Delta$).
2. $<$ 50% Byzantine *hashrate* …

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay $\leq \Delta$).
2. $<$ 50% Byzantine *hashrate at all times*.

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay ≤ Δ).
2. < 50% Byzantine *hashrate at all times*.
3. k large enough that, in every interval of ≥ 2k+2 views, a strict majority of the leaders are honest *with high probability*.

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay ≤ Δ).
2. < 50% Byzantine *hashrate at all times*.
3. k large enough that, in every interval of ≥ 2k+2 views, a strict majority of the leaders are honest *with high probability*.
4. *difficulty threshold t small enough that avg view length >> Δ*

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay ≤ Δ).
2. < 50% Byzantine *hashrate at all times*.
3. k large enough that, in every interval of ≥ 2k+2 views, a strict majority of the leaders are honest *with high probability*.
4. *difficulty threshold t small enough that avg view length >> Δ*

Guarantee: under these assumptions, Nakamoto consensus is consistent and live ….

Guarantees for Nakamoto Consensus

Assumptions: (all necessary)

1. Synchronous network (i.e., max message delay $\leq \Delta$).
2. < 50% Byzantine *hashrate at all times*.
3. k large enough that, in every interval of $\geq$ 2k+2 views, a strict majority of the leaders are honest *with high probability*.
4. *difficulty threshold t small enough that avg view length >> Δ*

Guarantee: under these assumptions, Nakamoto consensus is consistent and live *with high probability*.

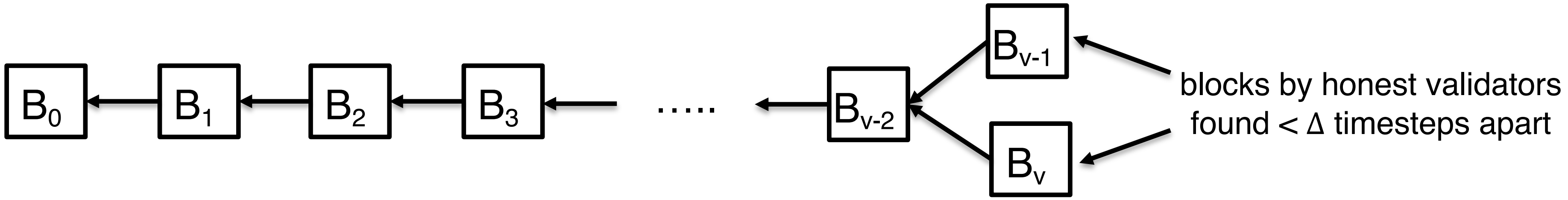
New Challenge: Honestly Produced Forks

New assumption: t small enough that avg view length >> Δ.

New Challenge: Honestly Produced Forks

New assumption: t small enough that avg view length >> Δ.

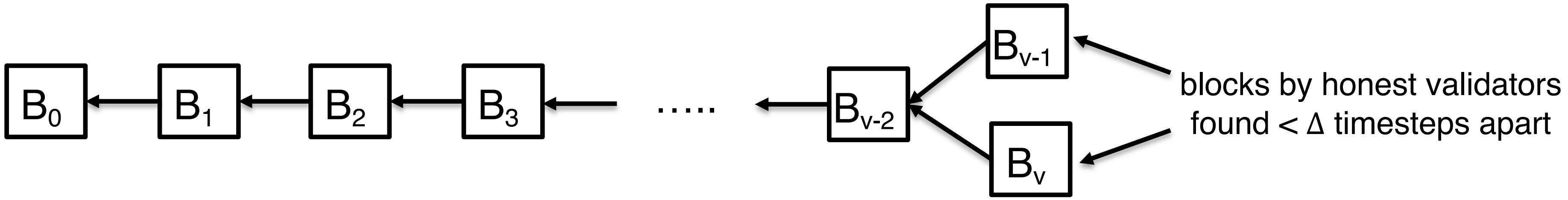
Reason: new source of forks in Nakamoto consensus: two honest validators solve puzzle at almost the same time.



New Challenge: Honestly Produced Forks

New assumption: t small enough that avg view length >> Δ.

Reason: new source of forks in Nakamoto consensus: two honest validators solve puzzle at almost the same time.



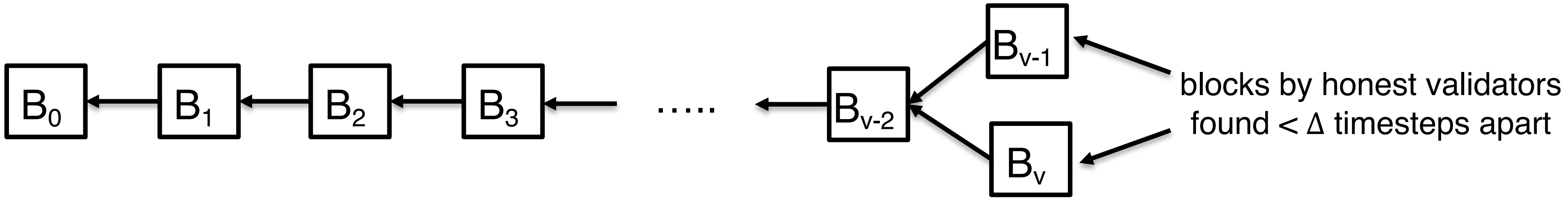
Consequence: some honestly produced blocks are "wasted."

- ➔ threshold for probabilistic consistency + liveness therefore < 50%

New Challenge: Honestly Produced Forks

New assumption: t small enough that avg view length >> Δ.

Reason: new source of forks in Nakamoto consensus: two honest validators solve puzzle at almost the same time.



Consequence: some honestly produced blocks are "wasted."

- ➔ threshold for probabilistic consistency + liveness therefore < 50%
- but if honest forking is rare, threshold remains close to 50%
- primary reason for slow block rate in Bitcoin (one block/10 minutes)

Limitations of Nakamoto Consensus

Drawbacks of Nakamoto consensus:

Limitations of Nakamoto Consensus

Drawbacks of Nakamoto consensus:

- loses consistency in the partially synchronous setting
  - true already for permissioned version

Limitations of Nakamoto Consensus

Drawbacks of Nakamoto consensus:

- loses consistency in the partially synchronous setting
  - true already for permissioned version
- even in synchrony, consistency + liveness only probabilistic
  - wasn't a problem is permissioned case (deterministic round-robin)

Limitations of Nakamoto Consensus

Drawbacks of Nakamoto consensus:

- loses consistency in the partially synchronous setting
  - true already for permissioned version
- even in synchrony, consistency + liveness only probabilistic
  - wasn’t a problem is permissioned case (deterministic round-robin)

Questions:

- tweak Nakamoto consensus so that one/both problems fixed?
- combine PoW with Tendermint rather than longest-chain?

Limitations of Proof-of-Work

Facts: [Lewis-Pye/Roughgarden, 2020-3]

Limitations of Proof-of-Work

Facts: [Lewis-Pye/Roughgarden, 2020-3]

1. no PoW protocol is consistent in partial synchrony

Limitations of Proof-of-Work

Facts: [Lewis-Pye/Roughgarden, 2020-3]

1. no PoW protocol is consistent in partial synchrony
   - assuming protocol is live in synchrony with no Byzantine validators
   - rules out combining PoW with Tendermint in any straightforward way

Limitations of Proof-of-Work

Facts: [Lewis-Pye/Roughgarden, 2020-3]

1. no PoW protocol is consistent in partial synchrony
   – assuming protocol is live in synchrony with no Byzantine validators
   – rules out combining PoW with Tendermint in any straightforward way
2. even in synchrony, no POW protocol always guarantees (deterministic) consistency + liveness

Limitations of Proof-of-Work

Facts: [Lewis-Pye/Roughgarden, 2020-3]

1. no PoW protocol is consistent in partial synchrony
   – assuming protocol is live in synchrony with no Byzantine validators
   – rules out combining PoW with Tendermint in any straightforward way
2. even in synchrony, no POW protocol always guarantees (deterministic) consistency + liveness

Upshot: drawbacks of Nakamoto consensus fundamental to all PoW protocols.

– can be overcome (under extra assumptions) with proof-of-stake protocols

Limitations of Proof-of-Work (con’d)

1. no PoW protocol is consistent in partial synchrony
2. even in synchrony, no POW protocol guarantees (deterministic) consistency + liveness

Intuition for (1): catch-22 a la “CAP principle” argument.

Limitations of Proof-of-Work (con’d)

1. no PoW protocol is consistent in partial synchrony
2. even in synchrony, no POW protocol guarantees (deterministic) consistency + liveness

Intuition for (1): catch-22 a la “CAP principle” argument. If validator hears no messages for a long time, can’t distinguish between:

- (i) in synchrony, other validators turned off their machines

Limitations of Proof-of-Work (con’d)

1. no PoW protocol is consistent in partial synchrony
2. even in synchrony, no POW protocol guarantees (deterministic) consistency + liveness

Intuition for (1): catch-22 a la “CAP principle” argument. If validator hears no messages for a long time, can’t distinguish between:

- (i) in synchrony, other validators turned off their machines
- (ii) in partial synchrony + pre-GST, all messages delayed

Limitations of Proof-of-Work (con’d)

1. no PoW protocol is consistent in partial synchrony
2. even in synchrony, no POW protocol guarantees (deterministic) consistency + liveness

Intuition for (1): catch-22 a la “CAP principle” argument. If validator hears no messages for a long time, can’t distinguish between:

- (i) in synchrony, other validators turned off their machines
- (ii) in partial synchrony + pre-GST, all messages delayed

Should the validator ever finalize any additional txs?

Limitations of Proof-of-Work (con'd)

1. no PoW protocol is consistent in partial synchrony
2. even in synchrony, no POW protocol guarantees (deterministic) consistency + liveness

Intuition for (1): catch-22 a la "CAP principle" argument. If validator hears no messages for a long time, can't distinguish between:

- (i) in synchrony, other validators turned off their machines
- (ii) in partial synchrony + pre-GST, all messages delayed

Should the validator ever finalize any additional txs?

- yes ➔ might be in scenario (ii), cause a consistency violation

Limitations of Proof-of-Work (con’d)

1. no PoW protocol is consistent in partial synchrony
2. even in synchrony, no POW protocol guarantees (deterministic) consistency + liveness

Intuition for (1): catch-22 a la “CAP principle” argument. If validator hears no messages for a long time, can’t distinguish between:

- (i) in synchrony, other validators turned off their machines
- (ii) in partial synchrony + pre-GST, all messages delayed

Should the validator ever finalize any additional txs?

- yes ➔ might be in scenario (ii), cause a consistency violation
- no ➔ might be in scenario (i), liveness violation (in synchrony)

Limitations of Proof-of-Work (con'd)

1. no PoW protocol is consistent in partial synchrony
2. even in synchrony, no POW protocol guarantees (deterministic) consistency + liveness

Intuition for (1): catch-22 a la "CAP principle" argument. If validator hears no messages for a long time, can't distinguish between:

- (i) in synchrony, other validators turned off their machines
- (ii) in partial synchrony + pre-GST, all messages delayed

Should the validator ever finalize any additional txs?

- yes ➔ might be in scenario (ii), cause a consistency violation
- no ➔ might be in scenario (i), liveness violation (in synchrony)

Proof of (2): similar to proof of FLP Impossibility Theorem.

– churning validators can substitute for unbounded message delays