# Lecture #21: Nakamoto Consensus: Difficulty Adjustment, Block Rewards, and Selfish Mining

COMS 4995-001:
The Science of Blockchains
URL: https://timroughgarden.org/s25/

Tim Roughgarden

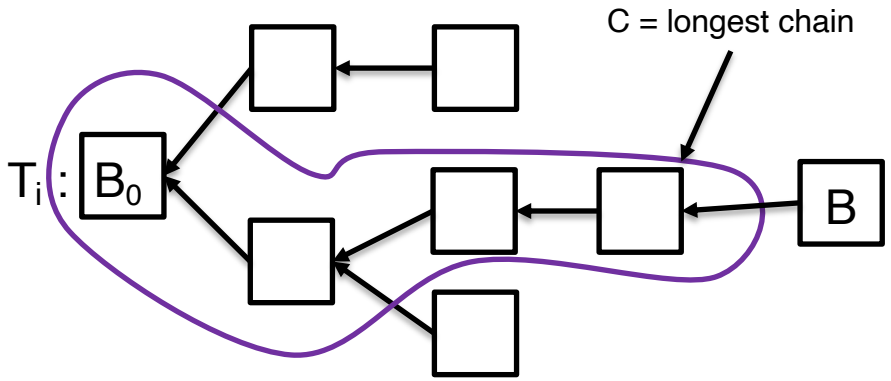
# Goals for Lecture #21

1. Difficulty adjustment in Nakamoto consensus.
   - tuning the difficulty threshold to achieve a target block rate
2. Limitations of proof-of-work.
   - drawbacks of Nakamoto consensus hold for PoW protocols generally
3. Why cryptocurrencies?
   - among other reasons, bootstrap a PoW blockchain protocol
4. Block rewards and selfish mining.
   - incentivizes block production; does it incentive other behavior even more?

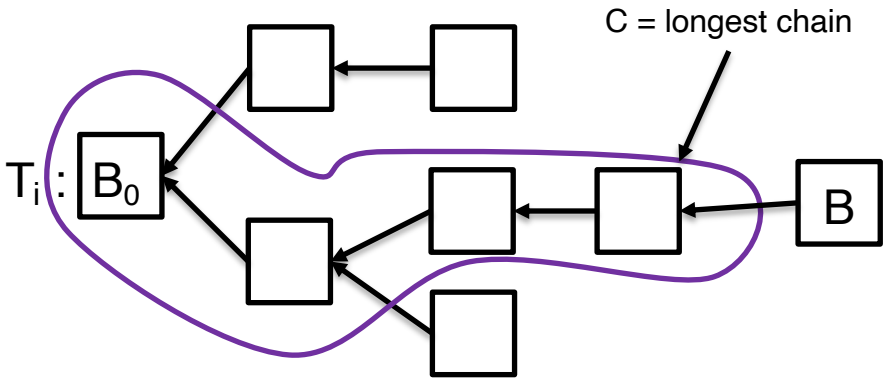
# Recall: Nakamoto Consensus

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

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



# Recall: 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)
- x = block header of the form < tx Merkle root II pred II pk II nonce >
  - point of the nonce: "grind" through possibilities until find a solution

# Difficulty Adjustment

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

Question: how to set t? [controls puzzle difficulty]

# Difficulty Adjustment

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

Question: how to set t? [controls puzzle difficulty]

Answer: to achieve a target rate of block production.

# Difficulty Adjustment

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

Question: how to set t? [controls puzzle difficulty]

Answer: to achieve a target rate of block production.

- to balance the pros of faster block production (lower latency)

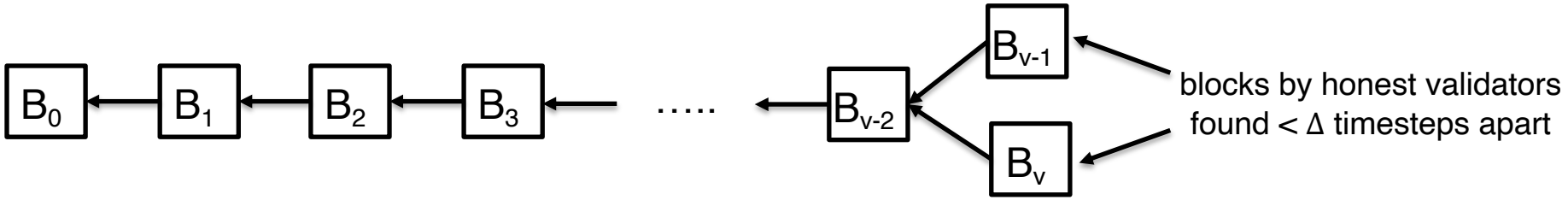
# Difficulty Adjustment

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

Question: how to set t? [controls puzzle difficulty]

Answer: to achieve a target rate of block production.

- to balance the pros of faster block production (lower latency) with the cons (more wasted work due to inadvertent forks)



# Difficulty Adjustment

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

Question: how to set t? [controls puzzle difficulty]

Answer: to achieve a target rate of block production.

- to balance the pros of faster block production (lower latency) with the cons (more wasted work due to inadvertent forks)

Consequence: threshold most decrease (resp., increase) as total amount of hashrate increases (resp., decreases).

  - difficulty adjustment algorithm programmatically makes these updates

# Case Study: The Bitcoin Protocol

- target rate = 1 block/10 minutes (➔ 144 blocks/day)

# Case Study: The Bitcoin Protocol

- target rate = 1 block/10 minutes (➔ 144 blocks/day)
- update t every epoch := 2016 blocks

# Case Study: The Bitcoin Protocol

- target rate = 1 block/10 minutes (➔ 144 blocks/day)
- update t every epoch := 2016 blocks
- if elapsed time in epoch = $\beta \cdot$ (14 days), reset t := $\beta \cdot$ t
  - if next epoch like previous one, expect to see 1 block/10 minutes

# Case Study: The Bitcoin Protocol

- target rate = 1 block/10 minutes (➔ 144 blocks/day)
- update t every epoch := 2016 blocks
- if elapsed time in epoch = $\beta \cdot$ (14 days), reset t := $\beta \cdot$ t
  - if next epoch like previous one, expect to see 1 block/10 minutes

Question: how is time measured?

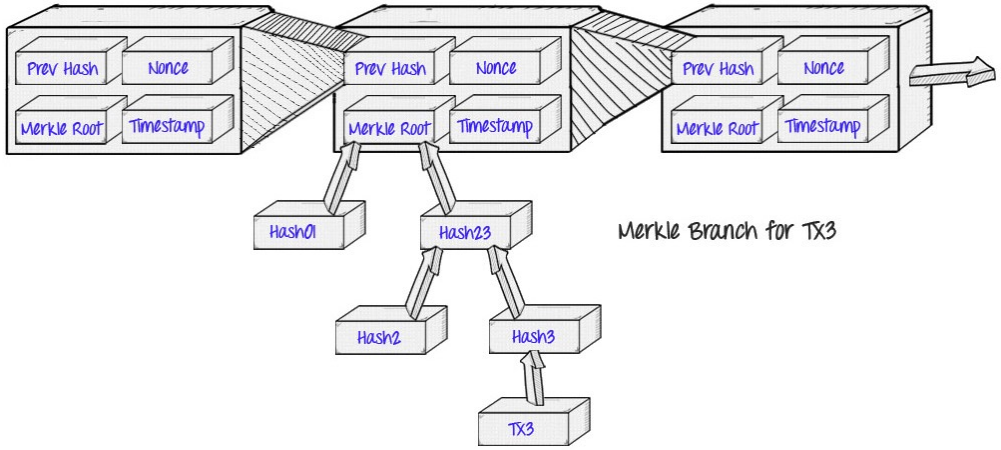
# Case Study: The Bitcoin Protocol

- target rate = 1 block/10 minutes (➔ 144 blocks/day)
- update t every epoch := 2016 blocks
- if elapsed time in epoch = $\beta \cdot$ (14 days), reset t := $\beta \cdot$ t
  - if next epoch like previous one, expect to see 1 block/10 minutes

Question: how is time measured?

Answer: timestamps.

- recorded in block headers



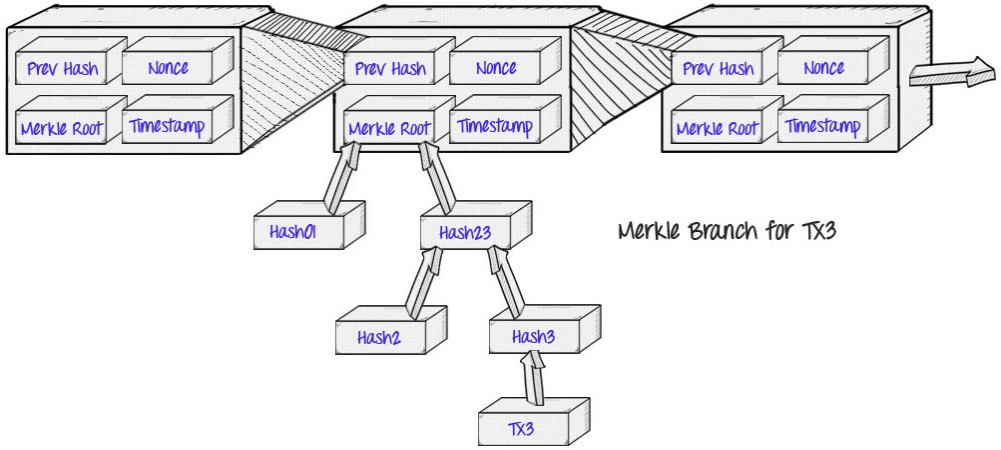
# Case Study: The Bitcoin Protocol

- target rate = 1 block/10 minutes (➔ 144 blocks/day)
- update t every epoch := 2016 blocks
- if elapsed time in epoch = $\beta \cdot$ (14 days), reset t := $\beta \cdot$ t
  - if next epoch like previous one, expect to see 1 block/10 minutes

Question: how is time measured?

Answer: timestamps.

- recorded in block headers
  - rules to limit timestamp manipulation by Byzantine validators (see HW7)



# Case Study: The Bitcoin Protocol

- target rate = 1 block/10 minutes (➔ 144 blocks/day)
- update t every epoch := 2016 blocks
- if elapsed time in epoch = $\beta \cdot$ (14 days), reset t := $\beta \cdot$ t
  - if next epoch like previous one, expect to see 1 block/10 minutes
- time measured via timestamps in block headers

Also: need to redefine Nakamoto consensus so that (honest) leaders extend the *heaviest* chain (rather than the longest chain).

- weight of block with threshold t := $2^{256}$/t [expected # of attempts to obtain]
- weight of chain = sum of weights of blocks in the chain

# Recall: 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 *with high probability*.

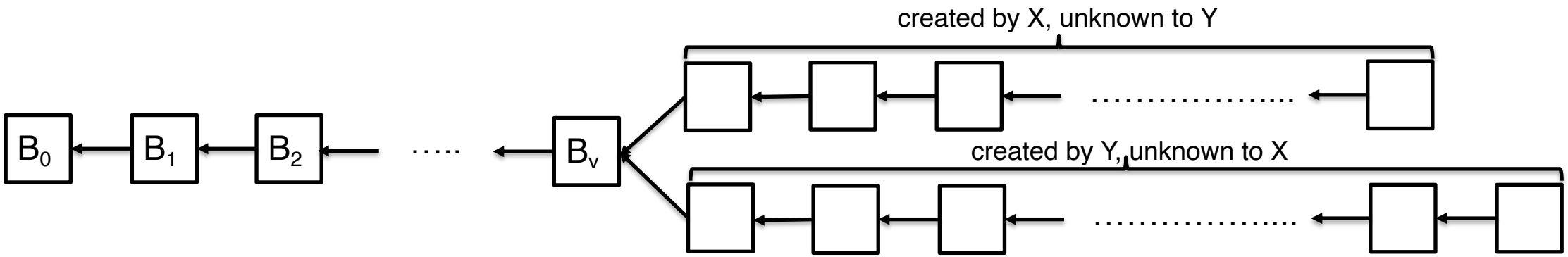
# Limitations of Nakamoto Consensus

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

# Incentivizing Validators

Question: why run a validator (e.g., for a PoW protocol)?

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol).

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol). Uses (incomplete list):

1. interesting in its own right (e.g., Bitcoin)

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol). Uses (incomplete list):

1. interesting in its own right (e.g., Bitcoin)
2. charge for usage (i.e., transaction fees)

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol). Uses (incomplete list):

1. interesting in its own right (e.g., Bitcoin)
2. charge for usage (i.e., transaction fees)
3. rewards contributions to protocol (e.g., PoW validators)

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol). Uses (incomplete list):

1. interesting in its own right (e.g., Bitcoin)
2. charge for usage (i.e., transaction fees)
3. rewards contributions to protocol (e.g., PoW validators)
4. proof-of-stake sybil-resistance

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol). Uses (incomplete list):

1. interesting in its own right (e.g., Bitcoin)
2. charge for usage (i.e., transaction fees)
3. rewards contributions to protocol (e.g., PoW validators)
4. proof-of-stake sybil-resistance
5. punish protocol deviators (a.k.a. “slashing”)

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol). Uses (incomplete list):

1. interesting in its own right (e.g., Bitcoin)
2. charge for usage (i.e., transaction fees)
3. rewards contributions to protocol (e.g., PoW validators)
4. proof-of-stake sybil-resistance
5. punish protocol deviators (a.k.a. “slashing”)

Questions: (i) how does cryptocurrency get distributed initially?

# Why Cryptocurrencies?

Cryptocurrecy: currency native to a blockchain protocol (i.e., minted/burned/tracked by the protocol). Uses (incomplete list):

1. interesting in its own right (e.g., Bitcoin)
2. charge for usage (i.e., transaction fees)
3. rewards contributions to protocol (e.g., PoW validators)
4. proof-of-stake sybil-resistance
5. punish protocol deviators (a.k.a. “slashing”)

Questions: (i) how does cryptocurrency get distributed initially?

(ii) why a validator be “honest” (vs. profit-maximizing)?

# Block Rewards

Question: why run a validator (e.g., for a PoW protocol)?

# Block Rewards

Question: why run a validator (e.g., for a PoW protocol)?

Answer in Nakamoto consensus: block rewards in native currency.

- newly minted coins paid to the “miner” of each block on longest chain
- in Bitcoin: currently 3.125 BTC per block

# Block Rewards

Question: why run a validator (e.g., for a PoW protocol)?

Answer in Nakamoto consensus: block rewards in native currency.

- newly minted coins paid to the “miner” of each block on longest chain
- in Bitcoin: currently 3.125 BTC per block

Worry: does incentivizing block production incentivize any deviations from Nakamoto consensus?

- hope: validators maximize rewards by following the protocol

# Block Rewards

Question: why run a validator (e.g., for a PoW protocol)?

Answer in Nakamoto consensus: block rewards in native currency.

- newly minted coins paid to the “miner” of each block on longest chain
- in Bitcoin: currently 3.125 BTC per block

Worry: does incentivizing block production incentivize any deviations from Nakamoto consensus?

- hope: validators maximize rewards by following the protocol
- paid for getting blocks on the longest chain, not for being honest per se
- e.g., could validators have their blocks orphaned at different rates?

# Selfish Mining in Nakamoto Consensus

Recall: difficulty adjustment ➔ fixed average rate of growth of longest chain (e.g., 2016 new blocks on longest chain per 14 days).

- says nothing about the number of orphaned blocks during this time

# Selfish Mining in Nakamoto Consensus

Recall: difficulty adjustment ➔ fixed average rate of growth of longest chain (e.g., 2016 new blocks on longest chain per 14 days).

- says nothing about the number of orphaned blocks during this time

- fixed rate of block rewards (e.g., 6300 BTC per 14 days)
  - ➔ to maximize rewards, maximize share of blocks on longest chain

# Selfish Mining in Nakamoto Consensus

Recall: difficulty adjustment ➔ fixed average rate of growth of longest chain (e.g., 2016 new blocks on longest chain per 14 days).

  – says nothing about the number of orphaned blocks during this time

- fixed rate of block rewards (e.g., 6300 BTC per 14 days)
  – ➔ to maximize rewards, maximize share of blocks on longest chain
- all validators honest ➔ share rewards proportional to hashrate

# Selfish Mining in Nakamoto Consensus

Recall: difficulty adjustment ➔ fixed average rate of growth of longest chain (e.g., 2016 new blocks on longest chain per 14 days).

  – says nothing about the number of orphaned blocks during this time

- fixed rate of block rewards (e.g., 6300 BTC per 14 days)
  – ➔ to maximize rewards, maximize share of blocks on longest chain
- all validators honest ➔ share rewards proportional to hashrate

Question: can a validator with an $\alpha$ fraction of the hash rate get $> \alpha$ fraction of overall rewards by deviation from the protocol?

# Selfish Mining in Nakamoto Consensus

Recall: difficulty adjustment ➔ fixed average rate of growth of longest chain (e.g., 2016 new blocks on longest chain per 14 days).

  – says nothing about the number of orphaned blocks during this time

- fixed rate of block rewards (e.g., 6300 BTC per 14 days)
  – ➔ to maximize rewards, maximize share of blocks on longest chain
- all validators honest ➔ share rewards proportional to hashrate

Question: can a validator with an $\alpha$ fraction of the hash rate get $> \alpha$ fraction of overall rewards by deviation from the protocol?

- in general, yes! [Eyal/Sirer 14]

# A Profitable Deviation

Setup: adversary controls $\alpha < ½$ of the overall hashrate.

- other $1 - \alpha$ fraction obediently follows the protocol
- synchronous model with $\Delta$=0 (all msgs delivered instantly)
- assume ties broken in favor of adversary (can relax w/more work)

Profitable deviation from Nakamoto consensus:

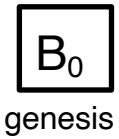
- goal (i): get as many "A-blocks" on longest chain as possible (ideally, all of them)
- goal (ii): orphan as many "H-blocks" as possible, to maximize share of A-blocks on the longest chain

# A Profitable Deviation (con’d)

Profitable deviation from Nakamoto consensus:

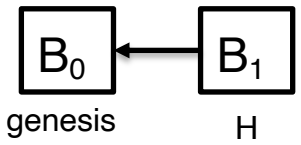
# A Profitable Deviation (con’d)

Profitable deviation from Nakamoto consensus:



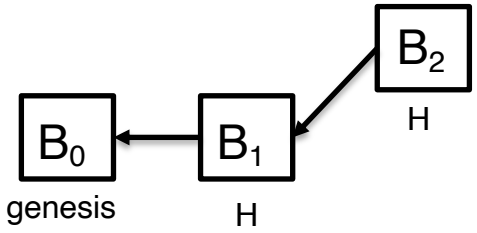
# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:



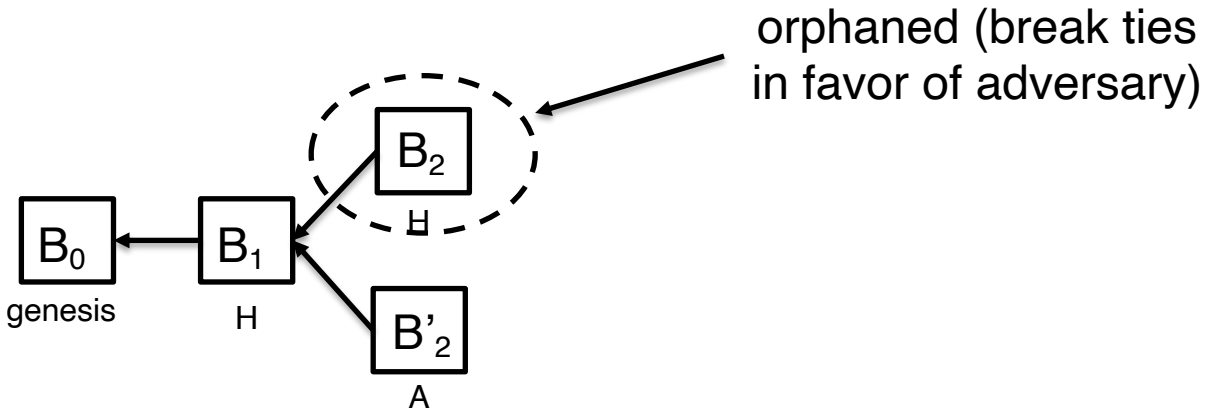
# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:



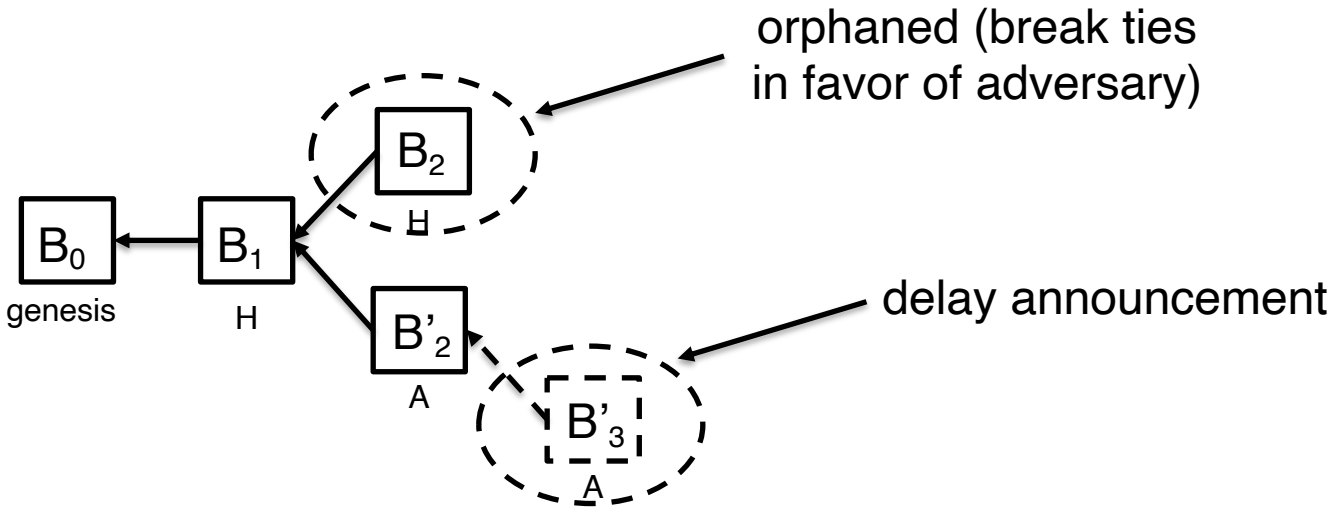
# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:



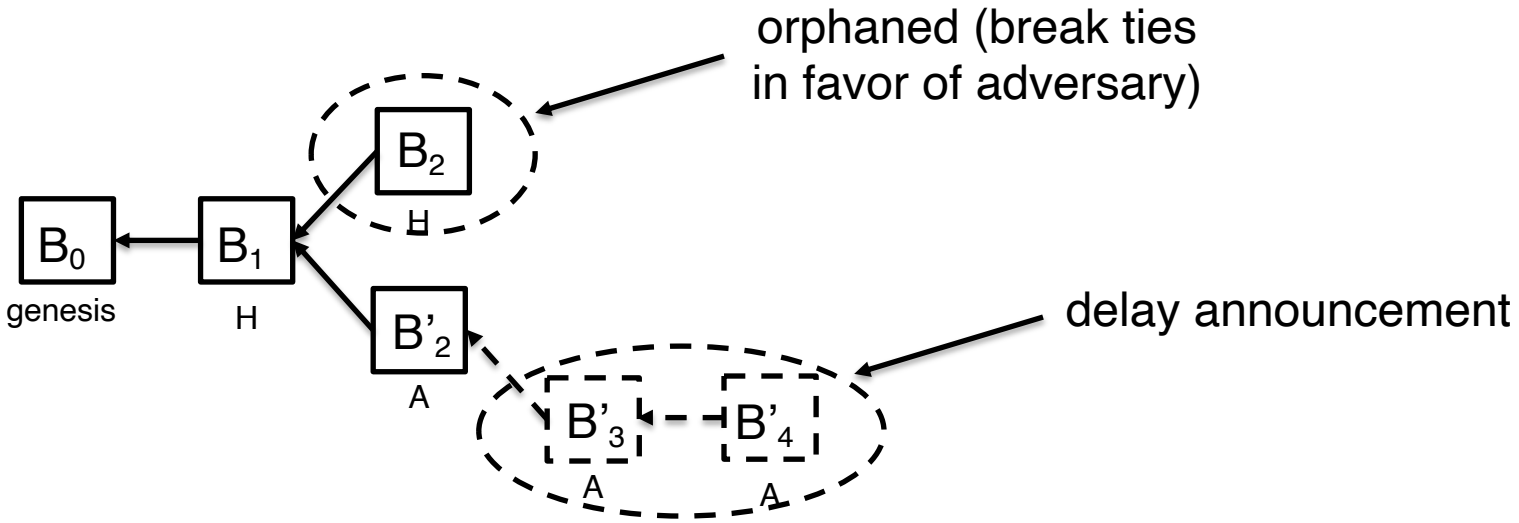
# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:



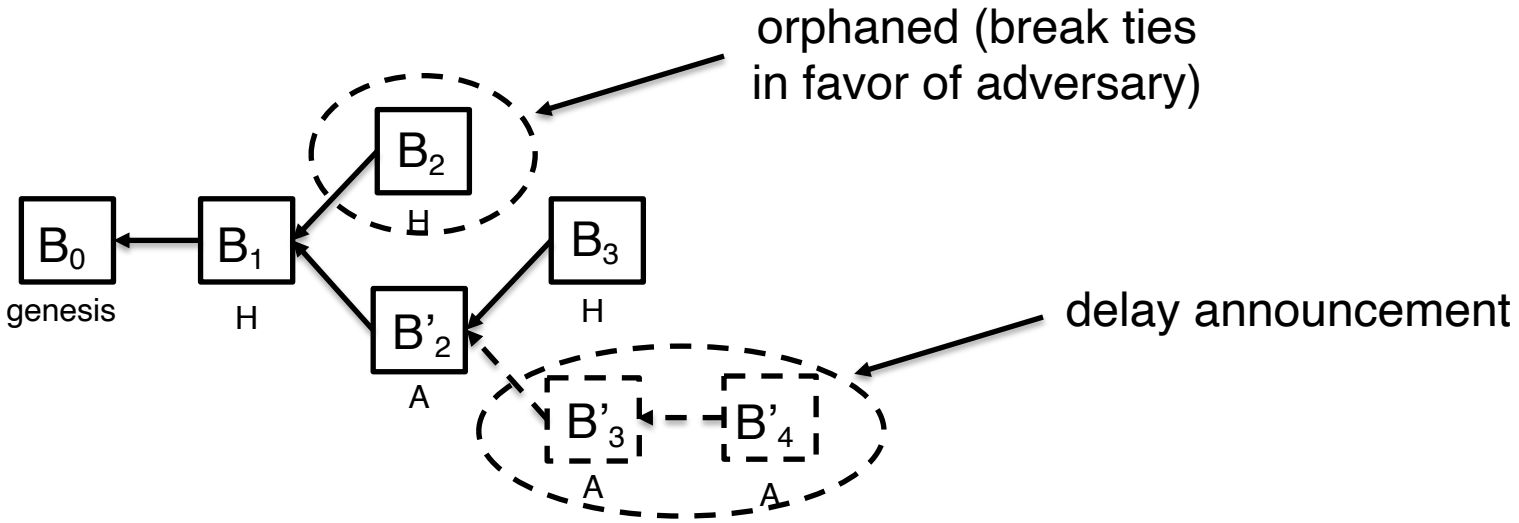
# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:



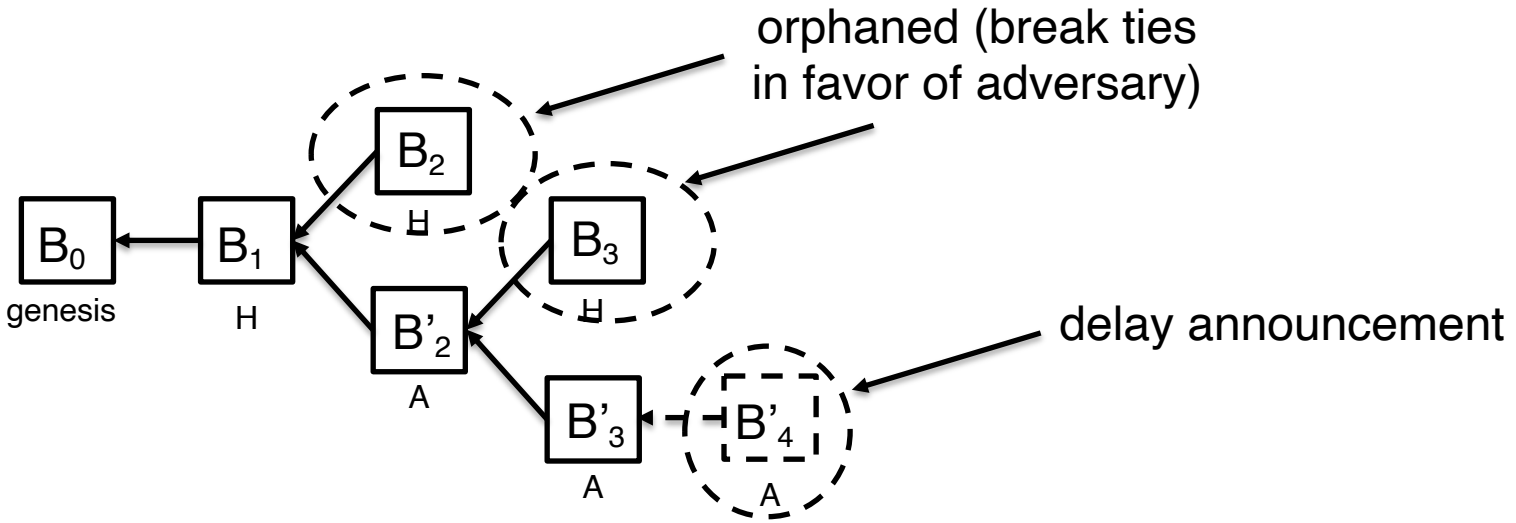
# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:



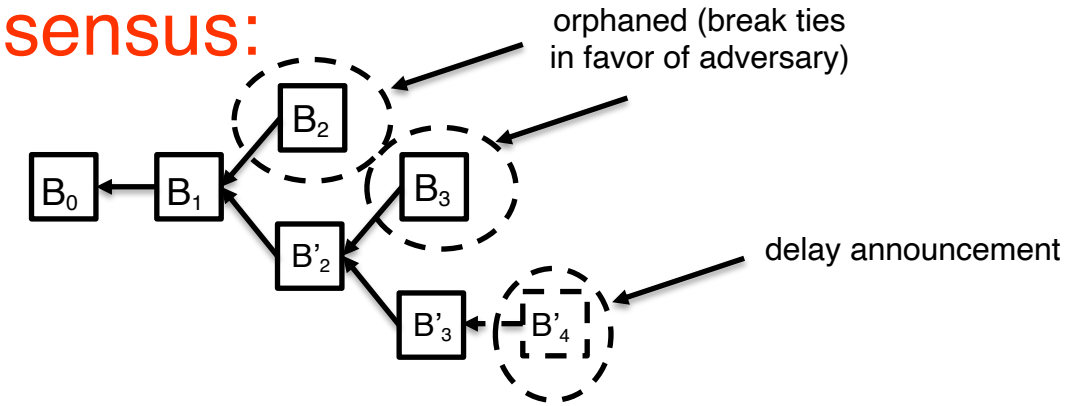
# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:



# A Profitable Deviation (con'd)

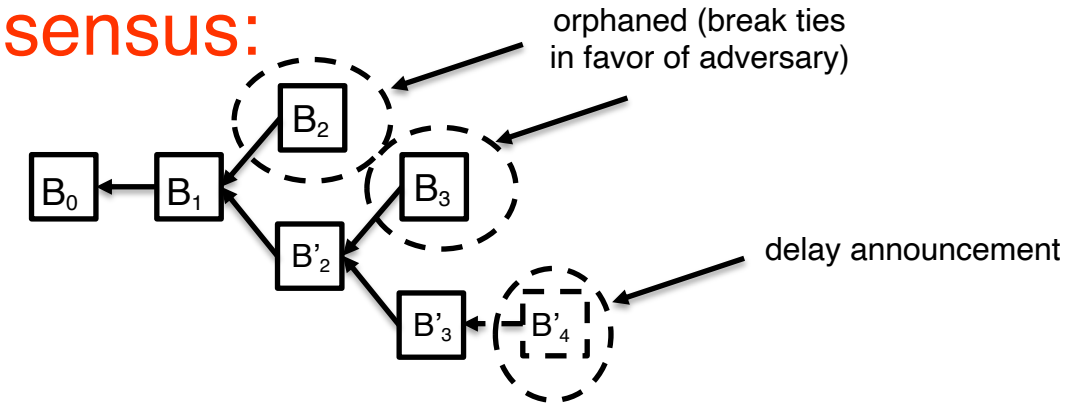
Profitable deviation from Nakamoto consensus:



# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:

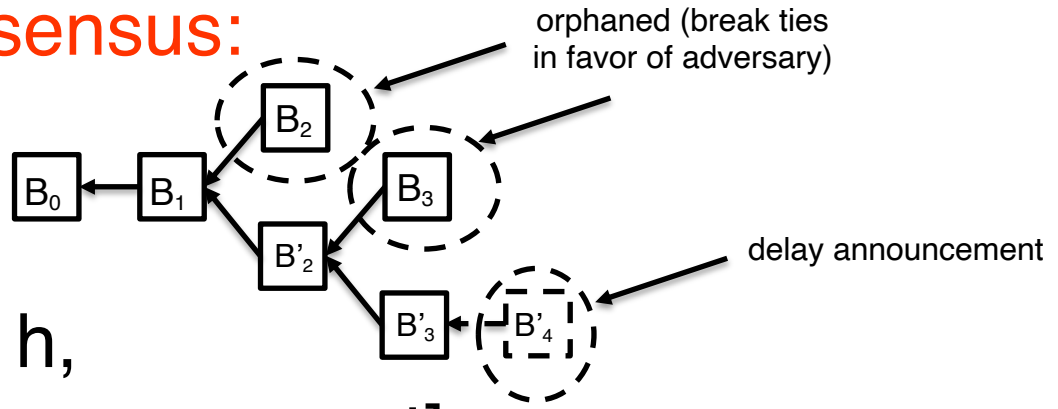
- let h = max height of any block produced thus far



# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:

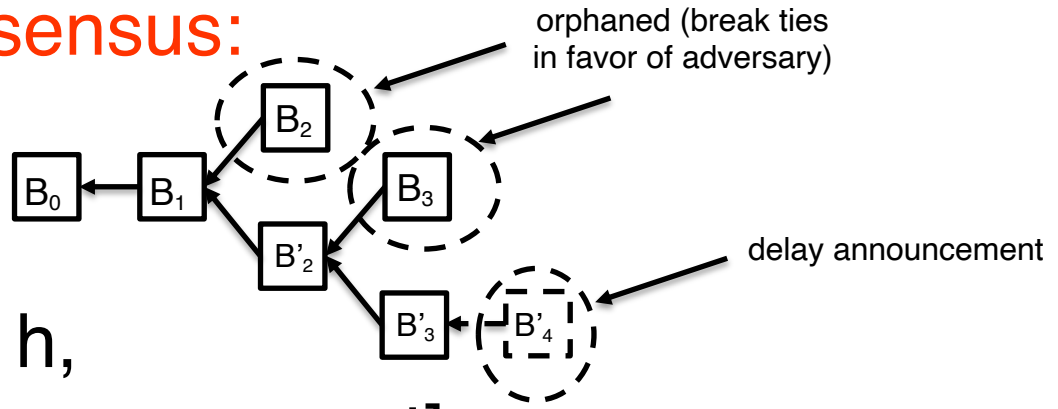
- let h = max height of any block produced thus far
- case 1: if there is an A-block at height h, try to extend it [successful ➔ delay announcement]



# A Profitable Deviation (con'd)

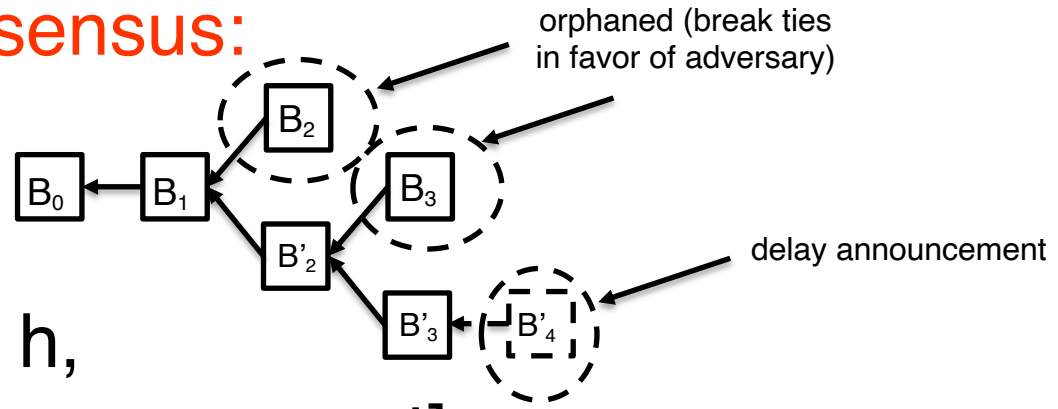
Profitable deviation from Nakamoto consensus:

- let h = max height of any block produced thus far
- case 1: if there is an A-block at height h, try to extend it [successful ➔ delay announcement]
- case 2: if only an H-block at height h, try to orphan it [successful ➔ announce immediately]



# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:

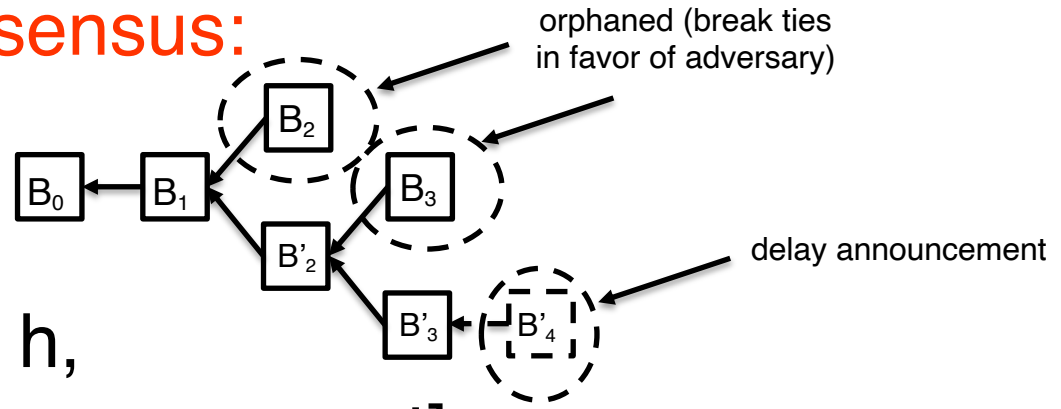



- let h = max height of any block produced thus far
- case 1: if there is an A-block at height h, try to extend it [successful ➔ delay announcement]
- case 2: if only an H-block at height h, try to orphan it [successful ➔ announce immediately]
- throughout: announce an A-block only once (+ immediately after) there in an H-block at the same height

# A Profitable Deviation (con'd)

Profitable deviation from Nakamoto consensus:

- let h = max height of any block produced thus far
- case 1: if there is an A-block at height h, try to extend it [successful ➔ delay announcement]
- case 2: if only an H-block at height h, try to orphan it [successful ➔ announce immediately]
- throughout: announce an A-block only once (+ immediately after) there in an H-block at the same height
  - ➔ every A-block deployed to knock an H-block off of the longest chain



# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced
2. every A-block gets on the longest chain

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced
2. every A-block gets on the longest chain
3. every A-block orphans a distinct H-block

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced
2. every A-block gets on the longest chain
3. every A-block orphans a distinct H-block

- ➔ # of H-blocks on longest chain = ≈$(1-\alpha)$N - $\alpha$N = $(1-2\alpha)$N

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced
2. every A-block gets on the longest chain
3. every A-block orphans a distinct H-block

- ➔ # of H-blocks on longest chain = ≈$(1-\alpha)$N - $\alpha$N = $(1-2\alpha)$N
- # of A-blocks on longest chain ≈ $\alpha$N

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced
2. every A-block gets on the longest chain
3. every A-block orphans a distinct H-block

- ➔ # of H-blocks on longest chain = ≈$(1-\alpha)$N - $\alpha$N = $(1-2\alpha)$N
- # of A-blocks on longest chain ≈ $\alpha$N
- ➔ A's share of blocks n longest chain ≈ $\alpha$N/[$\alpha$N + $(1-2\alpha)$N]

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced
2. every A-block gets on the longest chain
3. every A-block orphans a distinct H-block

- ➔ # of H-blocks on longest chain = ≈$(1-\alpha)$N - $\alpha$N = $(1-2\alpha)$N
- # of A-blocks on longest chain ≈ $\alpha$N
- ➔ A's share of blocks n longest chain ≈ $\alpha$N/[$\alpha$N + $(1-2\alpha)$N]
  = $\alpha/(1-\alpha) > \alpha$

# Analysis

Consider sequence of N rounds (e.g., N in the 1000s)

1. expect ≈ $\alpha$N A-blocks, ≈$(1-\alpha)$N H-blocks to be produced
2. every A-block gets on the longest chain
3. every A-block orphans a distinct H-block

- ➔ # of H-blocks on longest chain = ≈$(1-\alpha)$N - $\alpha$N = $(1-2\alpha)$N
- # of A-blocks on longest chain ≈ $\alpha$N
- ➔ A's share of blocks n longest chain ≈ $\alpha$N/[$\alpha$N + $(1-2\alpha)$N]
  = $\alpha/(1-\alpha) > \alpha$

Upshot: can boost rewards by deviating from intended behavior!