Lecture #7:
Longest-Chain Consensus

COMS 4995-001:
The Science of Blockchains

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

Tim Roughgarden

Two Categories of Blockchain Protocols

- "PBFT-type" protocols (e.g., Tendermint)
  - what we've been studying thus far
  - inspired by 20th-century consensus protocols (like PBFT [Castro/Liskov 99])

Two Categories of Blockchain Protocols

- "PBFT-type" protocols (e.g., Tendermint)
  - what we've been studying thus far
  - inspired by 20th-century consensus protocols (like PBFT [Castro/Liskov 99])
- "longest-chain" protocols (e.g., Bitcoin)
  - one of several innovations in Bitcoin
  - not considered pre-2008

Two Categories of Blockchain Protocols

- "PBFT-type" protocols (e.g., Tendermint)
  - what we've been studying thus far
  - inspired by 20th-century consensus protocols (like PBFT [Castro/Liskov 99])
- "longest-chain" protocols (e.g., Bitcoin)
  - one of several innovations in Bitcoin
  - not considered pre-2008
- perspective: design patterns with different consistency-liveness trade-offs; will fail in different ways (stalling vs. reorg/tx rollback)

Goals for Lecture #7

1. The essence of longest-chain consensus.
   - focus on permissioned implementation
   - will hint at proof-of-work permissionless version used in Bitcoin
2. Three drawbacks of longest-chain consensus.
   - loses consistency in asynchrony
   - even in synchrony, requires waiting to finalize transactions txs
   - Byzantine validators can control more than their fair share of blocks
3. Guarantees for longest-chain consensus.
   - consistent and live in synchrony with < 50% Byzantine validators

Longest-Chain Consensus

Longest-Chain Consensus [code run by every validator]

Longest-Chain Consensus

Longest-Chain Consensus

- B_0 = “genesis block”

[code run by every validator]

Longest-Chain Consensus

Longest-Chain Consensus

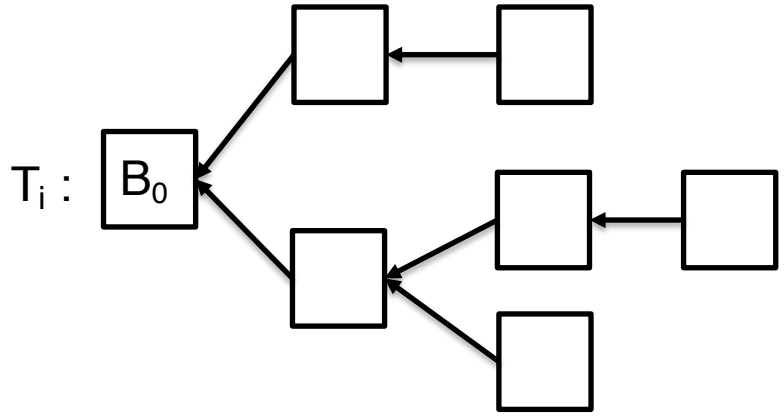
- B_0 = “genesis block”
- define view = Δ timesteps
- validators take turns as leader
- validators sign all messages

[code run by every validator]

Longest-Chain Consensus

Longest-Chain Consensus

[code run by every validator]

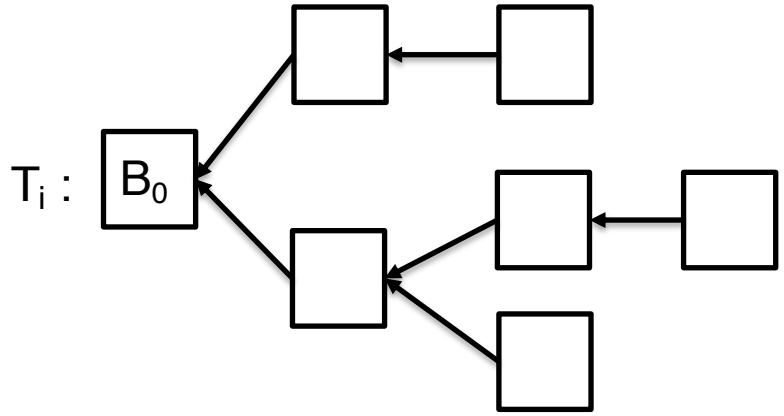


- B_0 = “genesis block”
- define view = Δ timesteps
- validators take turns as leader
- validators sign all messages
- validator i maintains in-tree T_i of valid blocks, rooted at B_0

Longest-Chain Consensus

Longest-Chain Consensus

[code run by every validator]



- B_0 = "genesis block"
- define view = Δ timesteps
- validators take turns as leader
- validators sign all messages
- 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' ≤ v
    - signed by leader of view v'
    - annotated with a predecessor block B'' from a view v'' < v'

Longest-Chain Consensus (con’d)

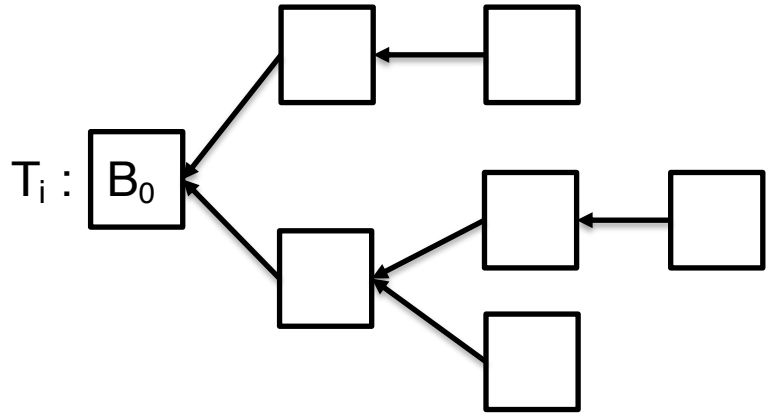
Longest-Chain Consensus [code run by every validator]

- at time $\Delta \cdot v$: [i.e., at beginning of view v]

Longest-Chain Consensus (con'd)

Longest-Chain Consensus [code run by every validator]

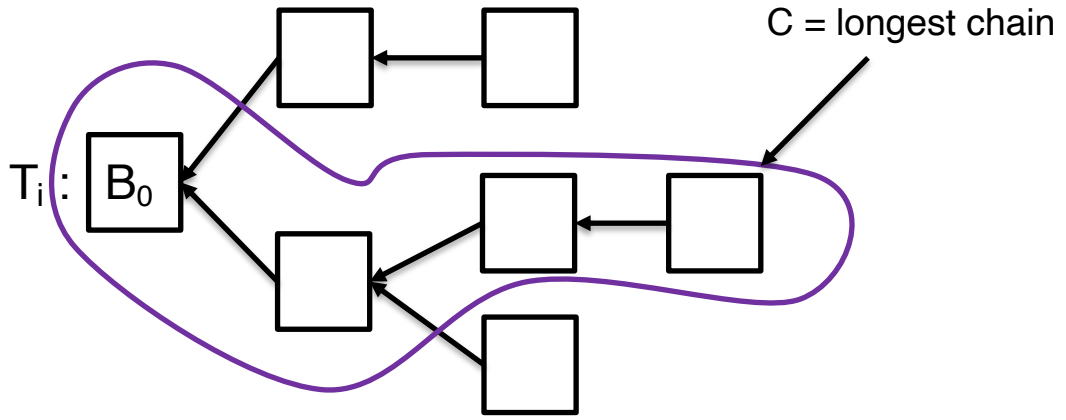
- at time $\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i updates T_i with any new blocks it's heard about, and forwards these blocks to all other validators



Longest-Chain Consensus (con'd)

Longest-Chain Consensus [code run by every validator]

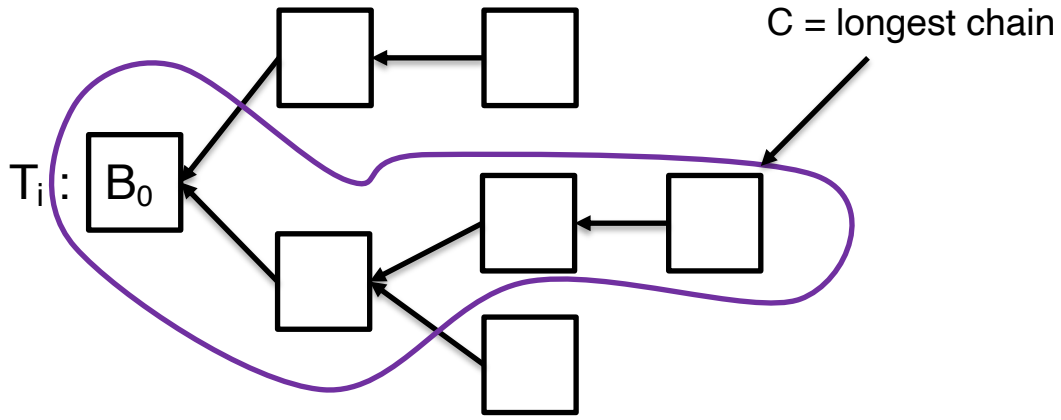
- at time $\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i updates T_i with any new blocks it's heard about, and forwards these blocks to all other validators
  - let C = longest chain in ℓ's in-tree
    - ℓ = leader of view v
    - break ties arbitrarily



Longest-Chain Consensus (con'd)

Longest-Chain Consensus [code run by every validator]

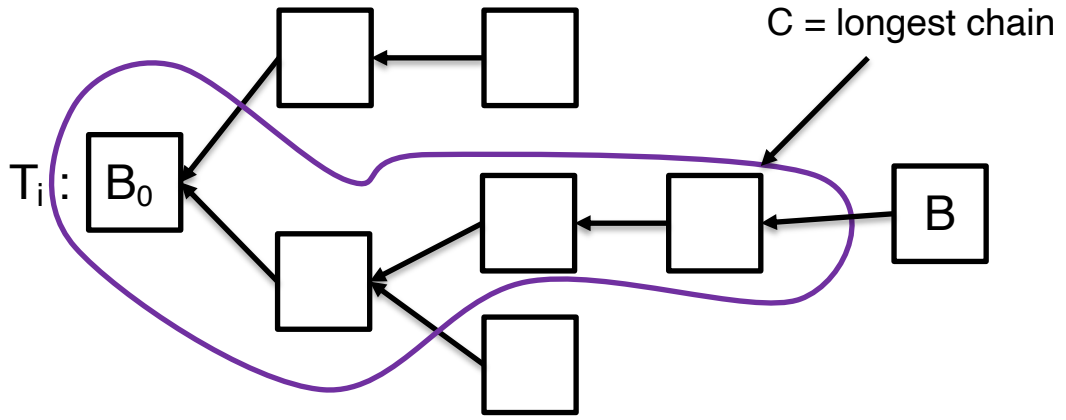
- at time $\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i updates T_i with any new blocks it's heard about, and forwards these blocks to all other validators
  - let C = longest chain in ℓ's in-tree
    - ℓ = leader of view v
    - break ties arbitrarily
  - let B := all not-yet-included (in C) txs ℓ knows about



Longest-Chain Consensus (con'd)

Longest-Chain Consensus [code run by every validator]

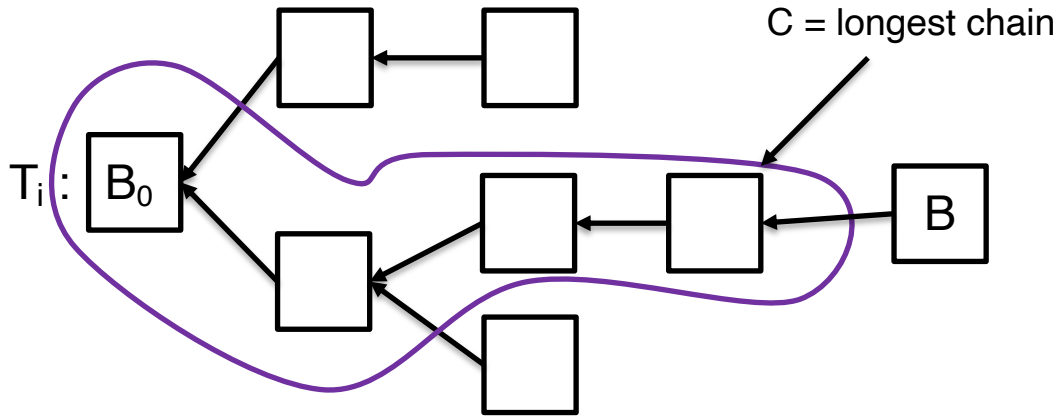
- at time $\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i updates T_i with any new blocks it's heard about, and forwards these blocks to all other validators
  - let C = longest chain in ℓ's in-tree
    - ℓ = leader of view v
    - break ties arbitrarily
  - let B := all not-yet-included (in C) txs ℓ knows about
  - ℓ adds B to its in-tree (extending C)



Longest-Chain Consensus (con'd)

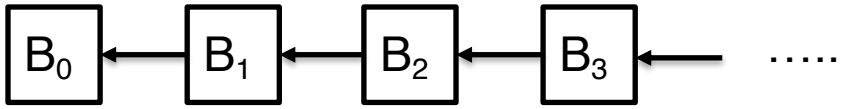
Longest-Chain Consensus [code run by every validator]

- at time $\Delta \cdot v$: [i.e., at beginning of view v]
  - each validator i updates T_i with any new blocks it's heard about, and forwards these blocks to all other validators
  - let C = longest chain in ℓ's in-tree
    - ℓ = leader of view v
    - break ties arbitrarily
  - 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



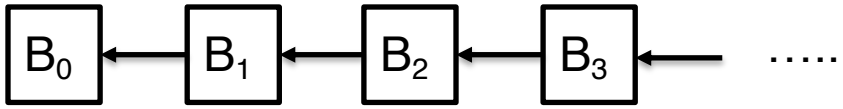
Forks in Longest-Chain Consensus

Sanity check: synchrony + no Byzantine validators ➔ no forks.

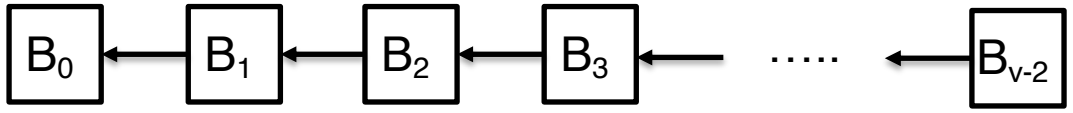


Forks in Longest-Chain Consensus

Sanity check: synchrony + no Byzantine validators ➔ no forks.

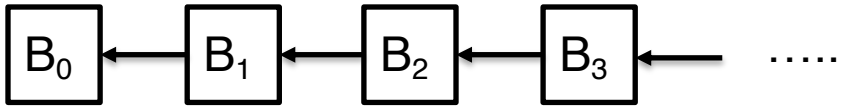


Scenario #1: leader of view v Byzantine

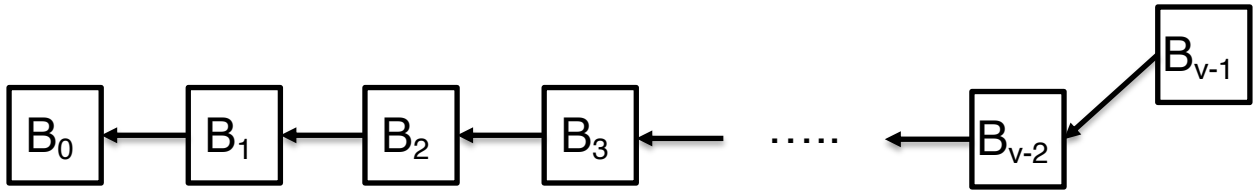


Forks in Longest-Chain Consensus

Sanity check: synchrony + no Byzantine validators ➔ no forks.

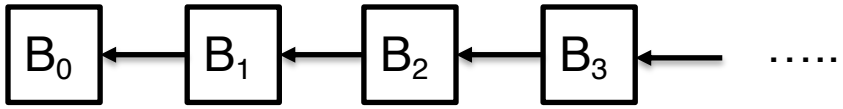


Scenario #1: leader of view v Byzantine

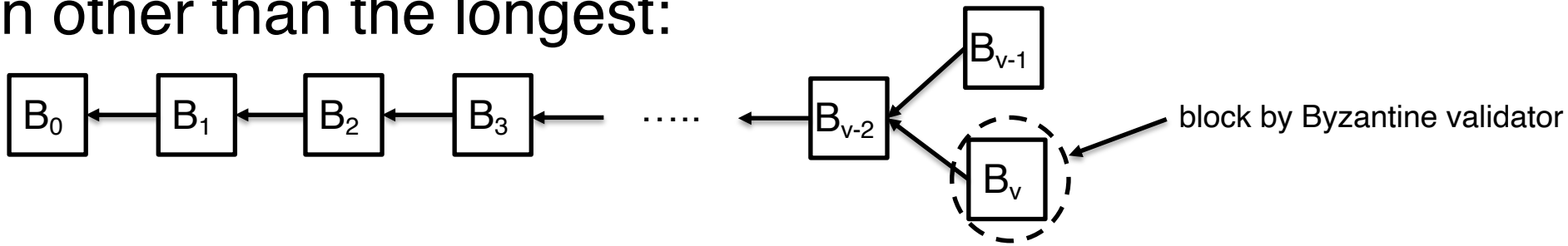


Forks in Longest-Chain Consensus

Sanity check: synchrony + no Byzantine validators ➔ no forks.

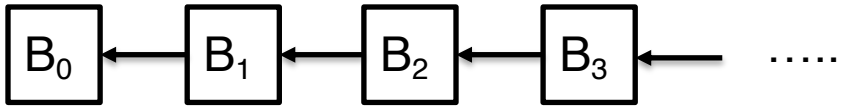


Scenario #1: leader of view v Byzantine ➔ deliberately extends chain other than the longest:

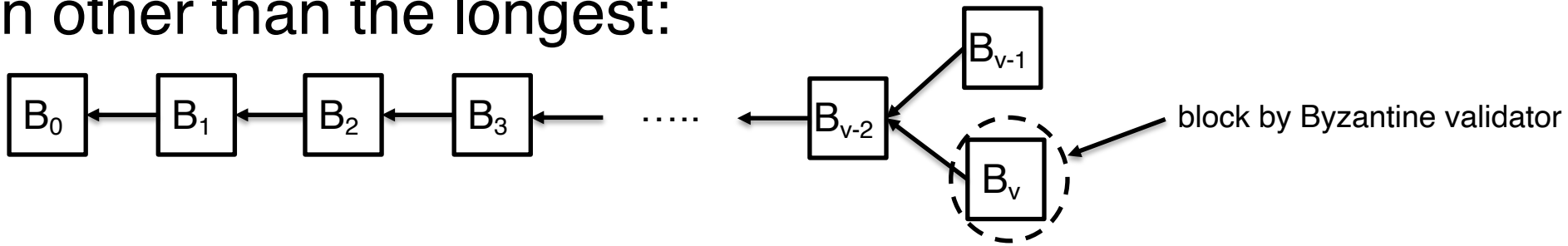


Forks in Longest-Chain Consensus

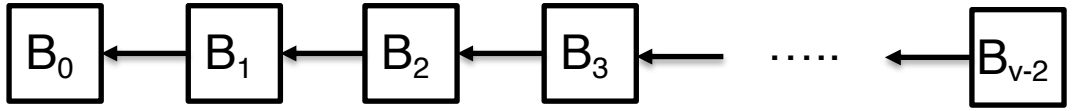
Sanity check: synchrony + no Byzantine validators ➔ no forks.



Scenario #1: leader of view v Byzantine ➔ deliberately extends chain other than the longest:

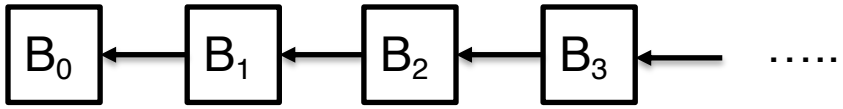


Scenario #2: leader of view v-1 Byzantine

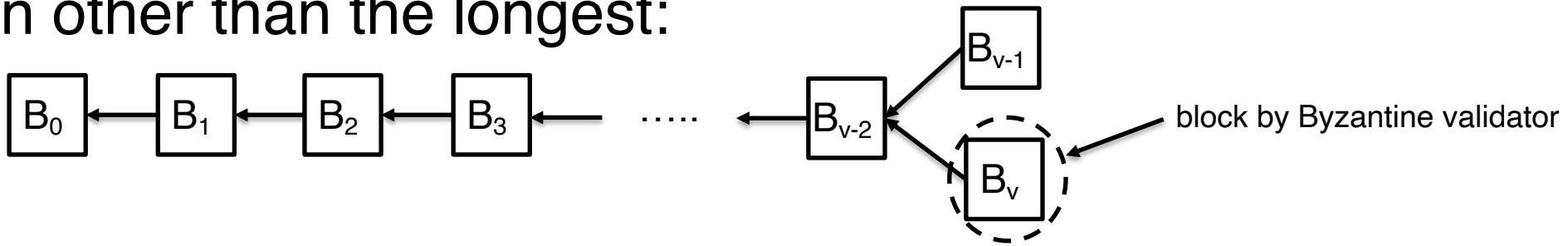


Forks in Longest-Chain Consensus

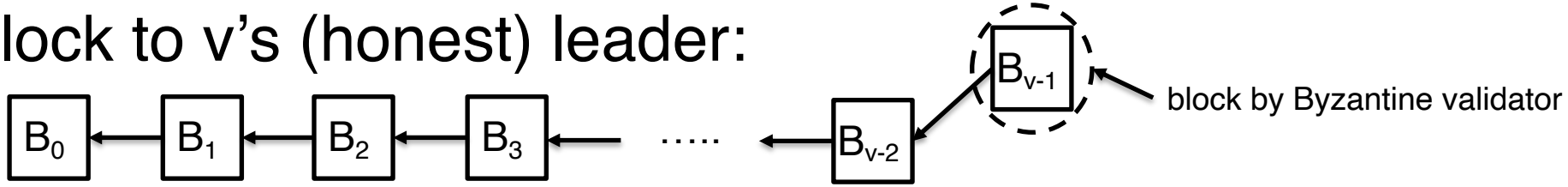
Sanity check: synchrony + no Byzantine validators ➔ no forks.



Scenario #1: leader of view v Byzantine ➔ deliberately extends chain other than the longest:

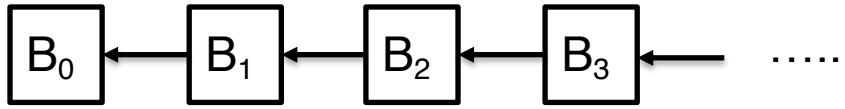


Scenario #2: leader of view v-1 Byzantine ➔ delays announcing its block to v's (honest) leader:

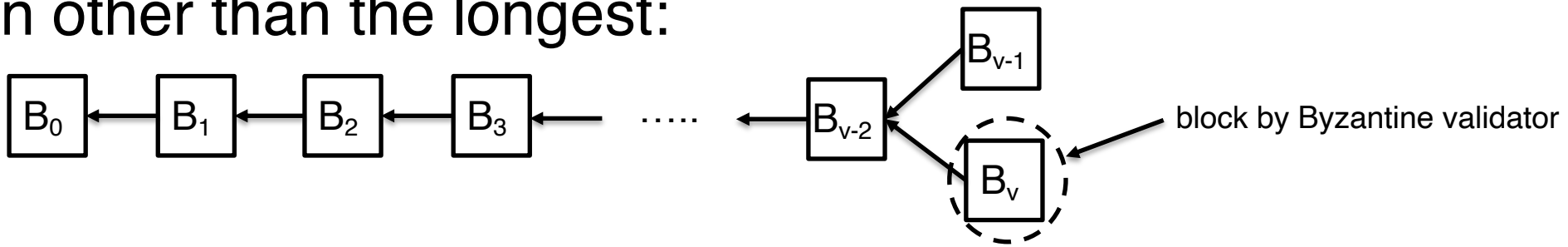


Forks in Longest-Chain Consensus

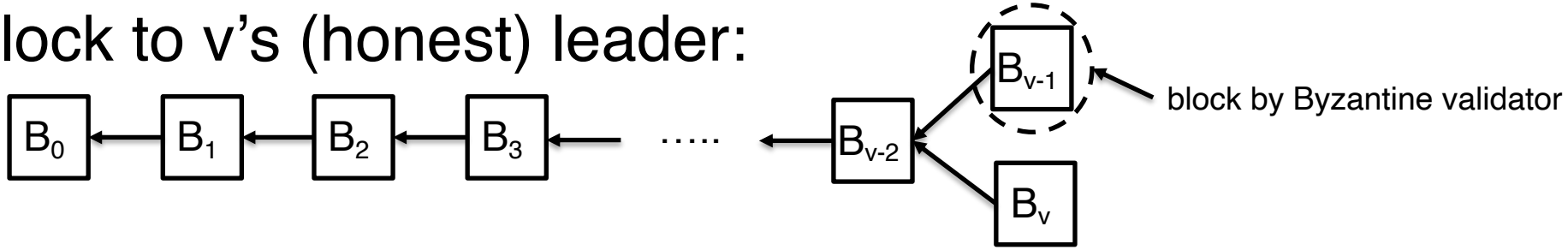
Sanity check: synchrony + no Byzantine validators ➔ no forks.



Scenario #1: leader of view v Byzantine ➔ deliberately extends chain other than the longest:



Scenario #2: leader of view v-1 Byzantine ➔ delays announcing its block to v's (honest) leader:



Which Transactions Are Finalized?

Which Transactions Are Finalized?

First attempt: finalized txs = all txs in the longest chain.

Which Transactions Are Finalized?

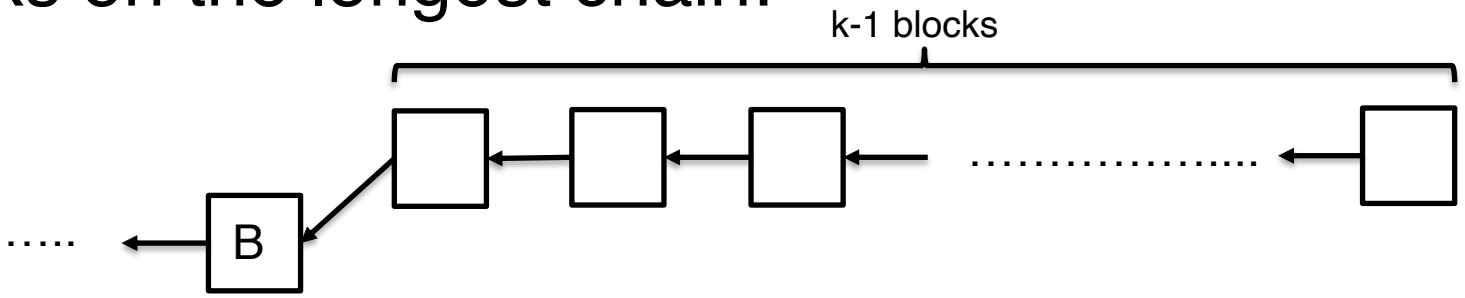
First attempt: finalized txs = all txs in the longest chain.

Note: k Byzantine leaders in a row ➔ can “cancel” the last k-1 blocks on the longest chain:

Which Transactions Are Finalized?

First attempt: finalized txs = all txs in the longest chain.

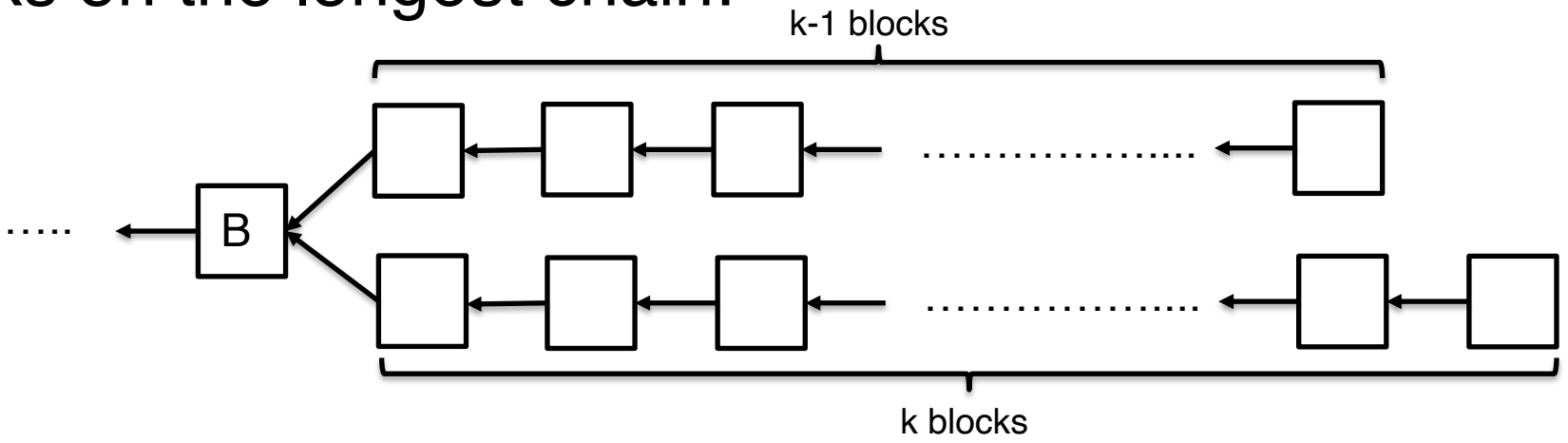
Note: k Byzantine leaders in a row ➔ can “cancel” the last k-1 blocks on the longest chain:



Which Transactions Are Finalized?

First attempt: finalized txs = all txs in the longest chain.

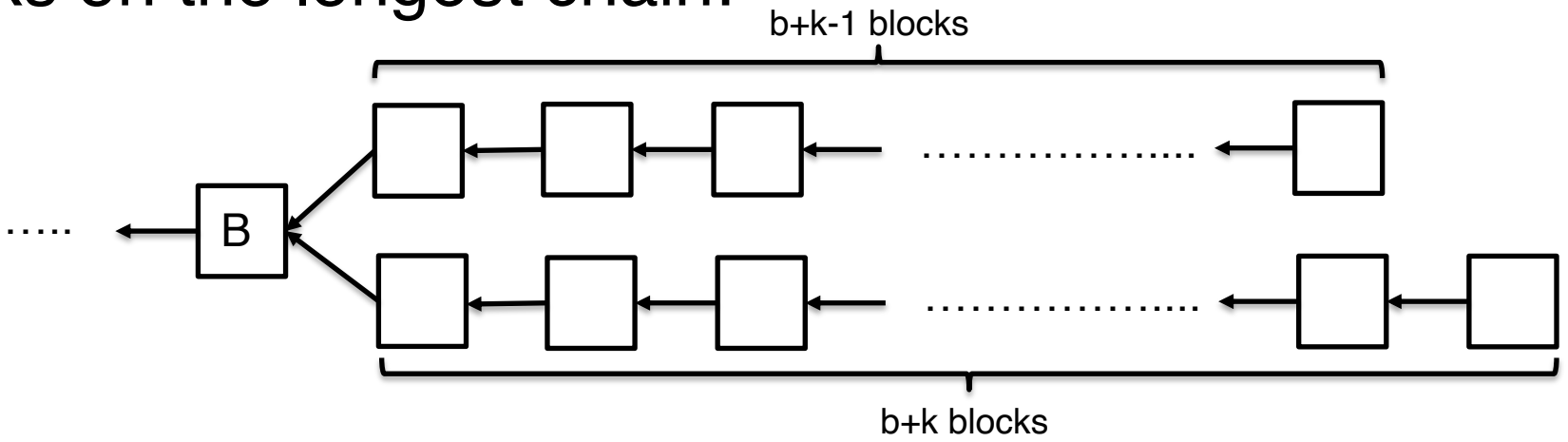
Note: k Byzantine leaders in a row ➔ can "cancel" the last k-1 blocks on the longest chain:



Which Transactions Are Finalized?

First attempt: finalized txs = all txs in the longest chain.

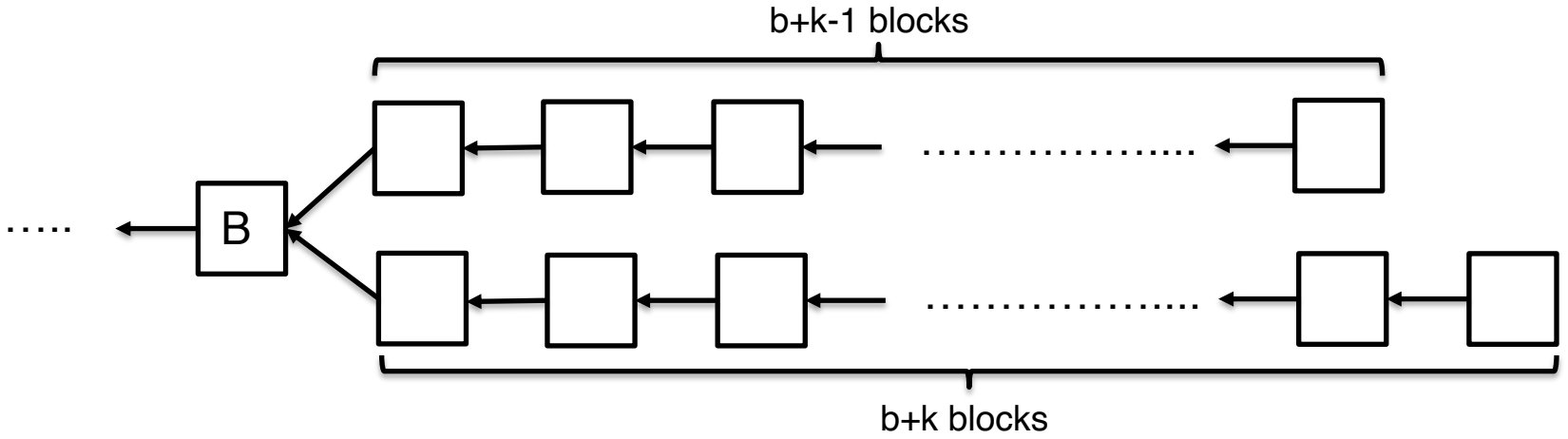
Note: k Byzantine leaders in a row ➔ can "cancel" the last k-1 blocks on the longest chain:



More generally: true for any interval in which Byzantine leaders outnumber honest leaders by ≥ k.

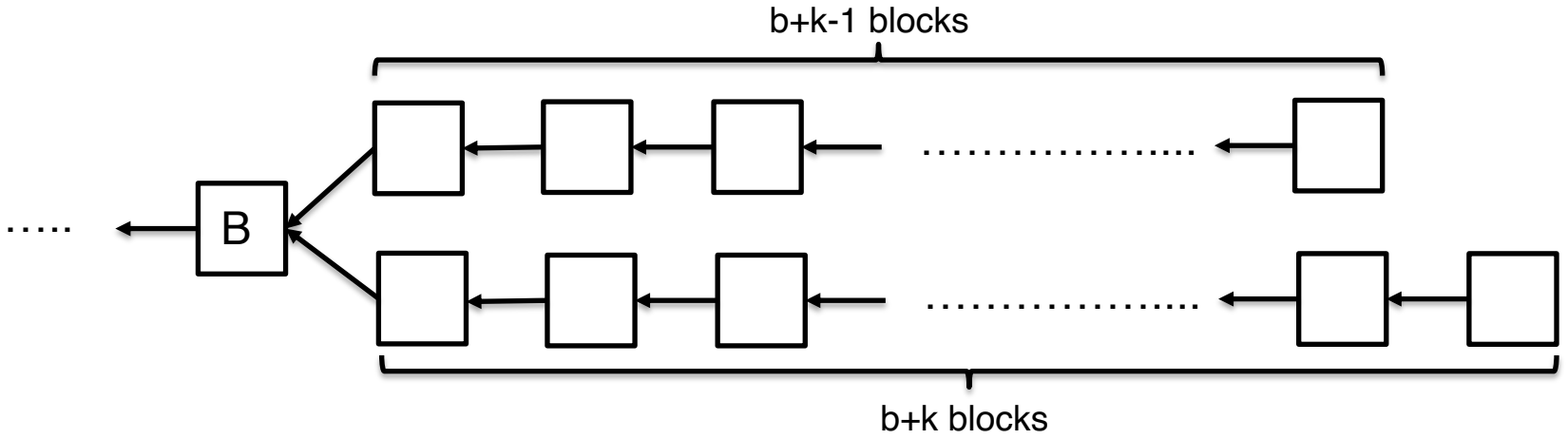
Which Transactions Are Finalized?

Note: if Byzantine leaders outnumber honest leaders by ≥ k :



Which Transactions Are Finalized?

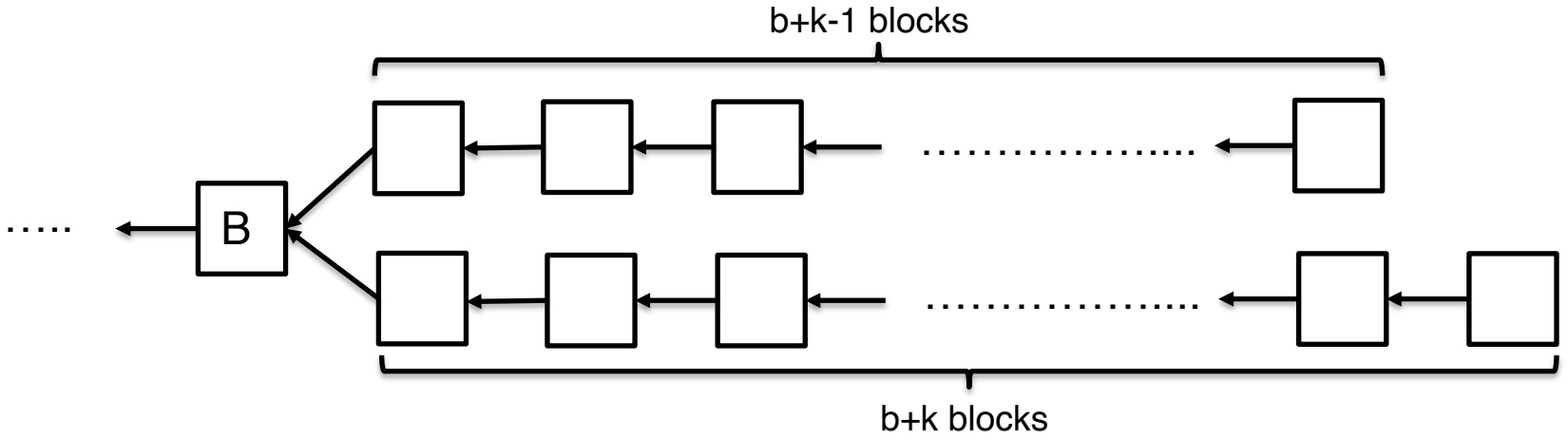
Note: if Byzantine leaders outnumber honest leaders by ≥ k :



Thus: > 50% Byzantine validators ➔ never safe to finalize a tx.

Which Transactions Are Finalized?

Note: if Byzantine leaders outnumber honest leaders by ≥ k :

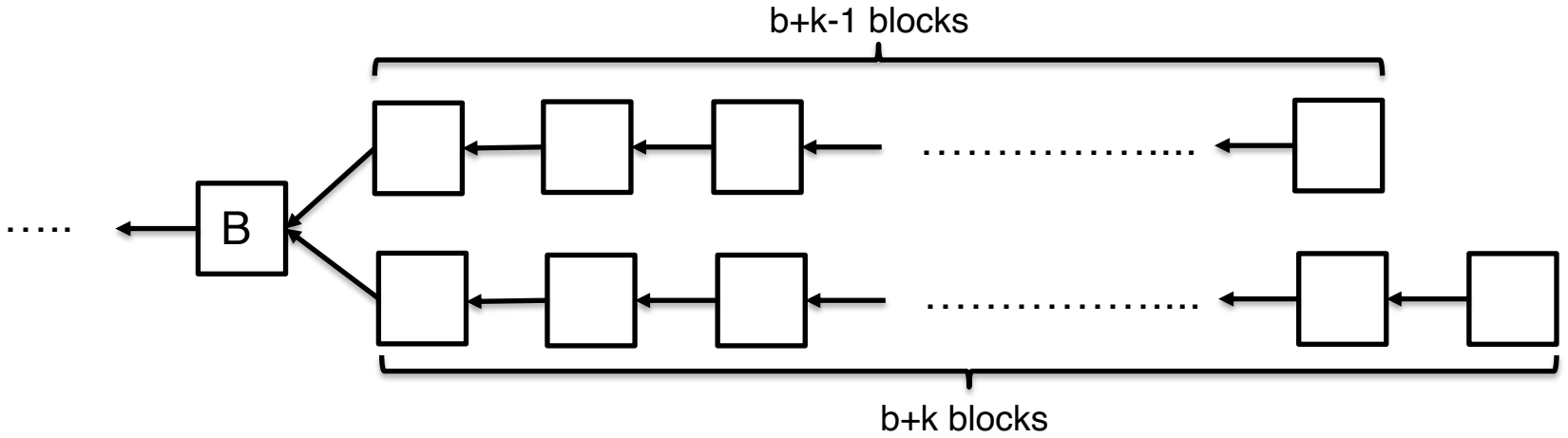


Thus: > 50% Byzantine validators ➔ never safe to finalize a tx.

- "51% attack": Byzantine validators can grow their own alternative chain, overwrite all of history

Which Transactions Are Finalized?

Note: if Byzantine leaders outnumber honest leaders by ≥ k :



Thus: > 50% Byzantine validators ➔ never safe to finalize a tx.

- "51% attack": Byzantine validators can grow their own alternative chain, overwrite all of history
  - ➔ always assume < 50% Byzantine validators in longest-chain consensus

Which Transactions Are Finalized?

Which Transactions Are Finalized?

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

Which Transactions Are Finalized?

Second attempt: for a security parameter k ≥ 1, finalized txs = all txs in the longest chain, except for those in the last k blocks.

- last k blocks of the longest chain tentative, under negotiation
  - hope: <50% Byzantine validators ➔ can only roll back bounded number of blocks (so blocks that are deep enough should be safe)

Which Transactions Are Finalized?

Second attempt: for a security parameter k ≥ 1, finalized txs = all txs in the longest chain, except for those in the last k blocks.

- last k blocks of the longest chain tentative, under negotiation
  - hope: <50% Byzantine validators ➔ can only roll back bounded number of blocks (so blocks that are deep enough should be safe)

Question: how to set k? [note: no reference to k in protocol code]

Which Transactions Are Finalized?

Second attempt: for a security parameter k ≥ 1, finalized txs = all txs in the longest chain, except for those in the last k blocks.

- last k blocks of the longest chain tentative, under negotiation
  - hope: <50% Byzantine validators ➔ can only roll back bounded number of blocks (so blocks that are deep enough should be safe)

Question: how to set k? [note: no reference to k in protocol code]

Answer: user-specified, trades off between security and latency.

- bigger k ➔ longer wait to finalize tx, less likely to ever be rolled back

Which Transactions Are Finalized?

Second attempt: for a security parameter k ≥ 1, finalized txs = all txs in the longest chain, except for those in the last k blocks.

- last k blocks of the longest chain tentative, under negotiation
  - hope: <50% Byzantine validators ➔ can only roll back bounded number of blocks (so blocks that are deep enough should be safe)

Question: how to set k? [note: no reference to k in protocol code]

Answer: user-specified, trades off between security and latency.

- bigger k ➔ longer wait to finalize tx, less likely to ever be rolled back
- folklore: for Bitcoin, k=6 (though Coinbase uses k=1)

Longest-Chain Consensus in Partial Synchrony

Recall: the partially synchronous model:

- shared global clock (timesteps=0,1,2,…)
- unknown transition time GST from asynchrony to synchrony
- known upper bound Δ on message delays post-GST

Longest-Chain Consensus in Partial Synchrony

Recall: the partially synchronous model:

- shared global clock (timesteps=0,1,2,…)
- unknown transition time GST from asynchrony to synchrony
- known upper bound Δ on message delays post-GST

Claim: longest-chain consensus is not consistent in partial synchrony, even with only honest validators (!).

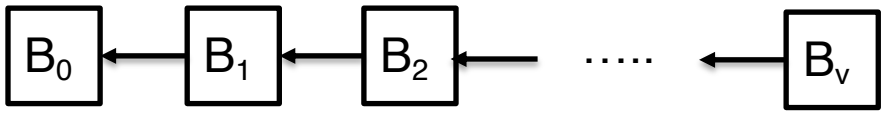
Longest-Chain Consensus in Partial Synchrony

Claim: longest-chain consensus is not consistent in partial synchrony, even with only honest validators (!).

Longest-Chain Consensus in Partial Synchrony

Claim: longest-chain consensus is not consistent in partial synchrony, even with only honest validators (!).

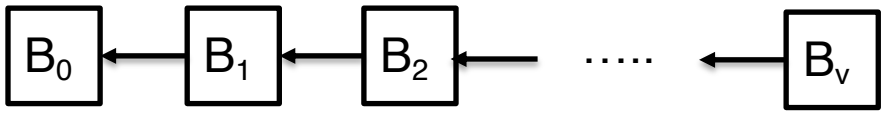
Example: Suppose current chain is (still pre-GST):



Longest-Chain Consensus in Partial Synchrony

Claim: longest-chain consensus is not consistent in partial synchrony, even with only honest validators (!).

Example: Suppose current chain is (still pre-GST):

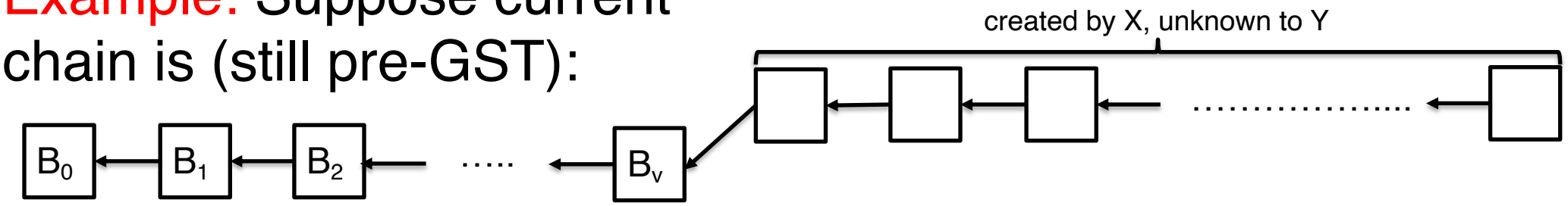


– let X,Y = partition of validator set, suppose all X⇔Y messages delayed for a long time (a.k.a. “network partition” – possible since pre-GST)

Longest-Chain Consensus in Partial Synchrony

Claim: longest-chain consensus is not consistent in partial synchrony, even with only honest validators (!).

Example: Suppose current chain is (still pre-GST):

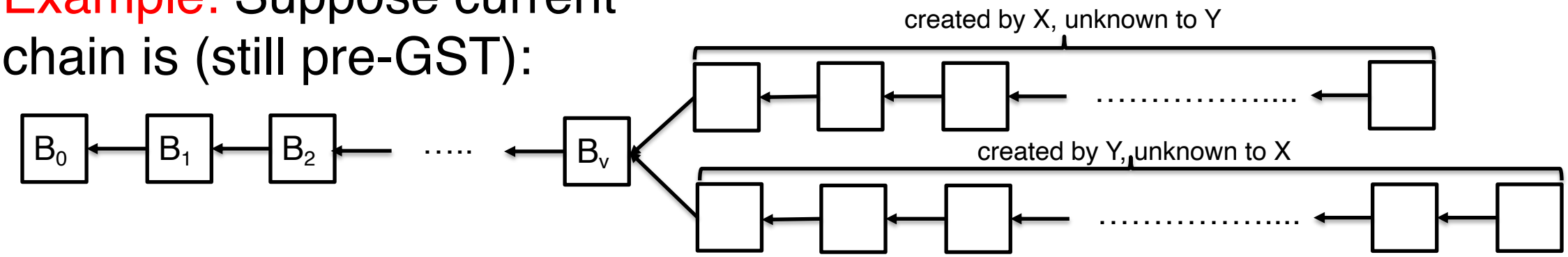


- let X,Y = partition of validator set, suppose all X⇔Y messages delayed for a long time (a.k.a. “network partition” – possible since pre-GST)
- each of X,Y continue to finalize their own (incompatible) blocks

Longest-Chain Consensus in Partial Synchrony

Claim: longest-chain consensus is not consistent in partial synchrony, even with only honest validators (!).

Example: Suppose current chain is (still pre-GST):



- let X,Y = partition of validator set, suppose all X⇔Y messages delayed for a long time (a.k.a. "network partition" – possible since pre-GST)
- each of X,Y continue to finalize their own (incompatible) blocks, eventually with > k blocks on each branch (➔ *consistency violation*)

Consistency-Liveness Trade-Offs

Longest-chain consensus in partial synchrony:

- bad news: lose consistency

Consistency-Liveness Trade-Offs

Longest-chain consensus in partial synchrony:

- bad news: lose consistency
- good news: continues to make progress during network partition
  - txs on the longer of the two branches remain finalized after partition ends

Consistency-Liveness Trade-Offs

Longest-chain consensus in partial synchrony:

- bad news: lose consistency
- good news: continues to make progress during network partition
  - txs on the longer of the two branches remain finalized after partition ends

Implication of FLP Theorem: in partial synchrony, can't guarantee both consistency and liveness pre-GST.

Consistency-Liveness Trade-Offs

Longest-chain consensus in partial synchrony:

- bad news: lose consistency
- good news: continues to make progress during network partition
  - txs on the longer of the two branches remain finalized after partition ends

Implication of FLP Theorem: in partial synchrony, can't guarantee both consistency and liveness pre-GST.

- Tendermint: favors consistency over liveness pre-GST
  - drawback: may stall during periods of asynchrony

Consistency-Liveness Trade-Offs

Longest-chain consensus in partial synchrony:

- bad news: lose consistency
- good news: continues to make progress during network partition
  - txs on the longer of the two branches remain finalized after partition ends

Implication of FLP Theorem: in partial synchrony, can't guarantee both consistency and liveness pre-GST.

- Tendermint: favors consistency over liveness pre-GST
  - drawback: may stall during periods of asynchrony
- longest-chain consensus: favors liveness over consistency pre-GST
  - drawback: asynchrony ➔ may reorg/roll back thought-to-be-finalized txs

Consistency-Liveness Trade-Offs (con'd)

Implication of FLP Theorem: in partial synchrony, can't guarantee both consistency and liveness pre-GST.

- Tendermint: favors consistency over liveness pre-GST
  - drawback: may stall during periods of asynchrony
- longest-chain consensus: favors liveness over consistency pre-GST
  - drawback: asynchrony ➔ may reorg/roll back thought-to-be-finalized txs

Consistency-Liveness Trade-Offs (con'd)

Implication of FLP Theorem: in partial synchrony, can't guarantee both consistency and liveness pre-GST.

- Tendermint: favors consistency over liveness pre-GST
  - drawback: may stall during periods of asynchrony
- longest-chain consensus: favors liveness over consistency pre-GST
  - drawback: asynchrony ➔ may reorg/roll back thought-to-be-finalized txs

Analog: the *CAP Principle* from distributed systems.

- can only pick two of {consistency, availability, partition-tolerance}
- which to give up on is application-specific (e.g., a bank vs. amazon.com)

Consistency-Liveness Trade-Offs (con'd)

Implication of FLP Theorem: in partial synchrony, can't guarantee both consistency and liveness pre-GST.

- Tendermint: favors consistency over liveness pre-GST
  - drawback: may stall during periods of asynchrony
- longest-chain consensus: favors liveness over consistency pre-GST
  - drawback: asynchrony ➔ may reorg/roll back thought-to-be-finalized txs

Analog: the *CAP Principle* from distributed systems.

- can only pick two of {consistency, availability, partition-tolerance}
- which to give up on is application-specific (e.g., a bank vs. amazon.com)
  - Bitcoin: favors liveness despite hosting a valuable cryptocurrency (mismatch?)

Chain Quality

Setup: synchronous model, < 50% Byzantine validators.

Chain Quality

Setup: synchronous model, < 50% Byzantine validators.

Definition: *quality* of a chain = fraction of blocks that were created by honest validators. [as a function of the fraction α of Byzantine validators]

- import: blocks by Byzantine validators may censor certain txs, or be empty

Chain Quality

Setup: synchronous model, < 50% Byzantine validators.

Definition: *quality* of a chain = fraction of blocks that were created by honest validators. [as a function of the fraction α of Byzantine validators]

- import: blocks by Byzantine validators may censor certain txs, or be empty
  - example: in Tendermint, for $\alpha < {}^{1}/_{3}$, post-GST chain quality is $\geq 1 - \alpha$

Chain Quality

Setup: synchronous model, < 50% Byzantine validators.

Definition: *quality* of a chain = fraction of blocks that were created by honest validators. [as a function of the fraction α of Byzantine validators]

- import: blocks by Byzantine validators may censor certain txs, or be empty
  - example: in Tendermint, for $\alpha < {}^{1}/_{3}$, post-GST chain quality is $\geq 1 - \alpha$

Bad news: longest-chain ➔ chain quality can be as bad as $\frac{1-2\alpha}{1-\alpha}$.

Chain Quality

Bad news: longest-chain ➔ chain quality can be as bad as $\frac{1-2\alpha}{1-\alpha}$.

Chain Quality

Bad news: longest-chain ➔ chain quality can be as bad as $\frac{1-2\alpha}{1-\alpha}$.

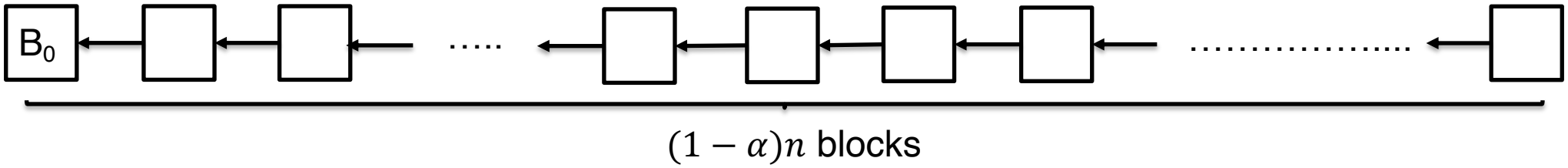
Example: suppose αn Byzantine validators appear consecutively in the round-robin ordering.

Chain Quality

Bad news: longest-chain ➔ chain quality can be as bad as $\frac{1-2\alpha}{1-\alpha}$.

Example: suppose αn Byzantine validators appear consecutively in the round-robin ordering.

- honest validators add $(1-\alpha)n$ new blocks to longest chain:

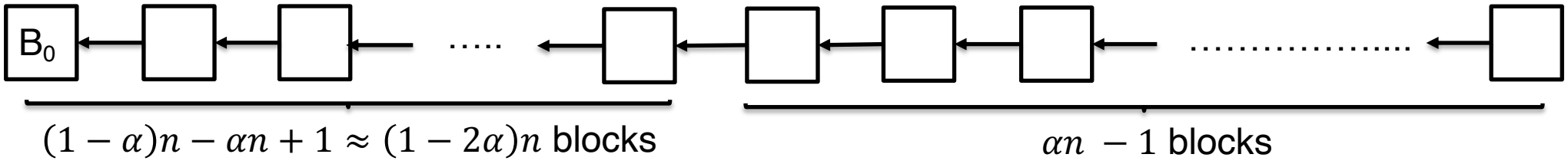


Chain Quality

Bad news: longest-chain ➔ chain quality can be as bad as $\frac{1-2\alpha}{1-\alpha}$.

Example: suppose αn Byzantine validators appear consecutively in the round-robin ordering.

- honest validators add $(1-\alpha)n$ new blocks to longest chain:

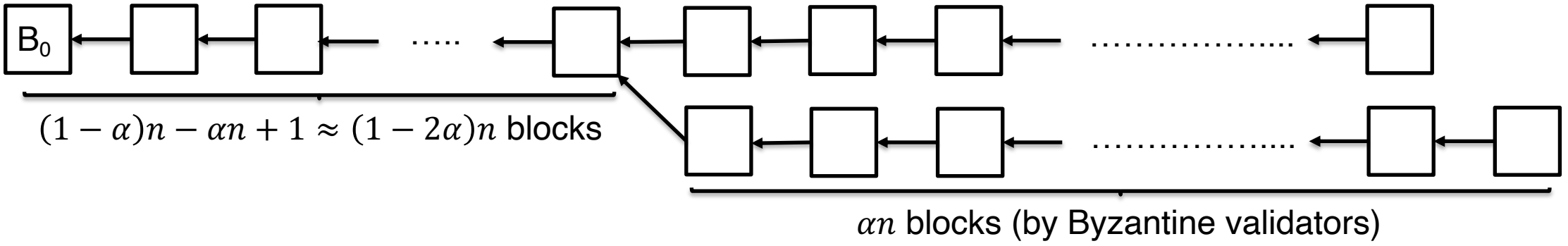


Chain Quality

Bad news: longest-chain ➔ chain quality can be as bad as $\frac{1-2\alpha}{1-\alpha}$.

Example: suppose αn Byzantine validators appear consecutively in the round-robin ordering.

- honest validators add $(1-\alpha)n$ new blocks to longest chain:

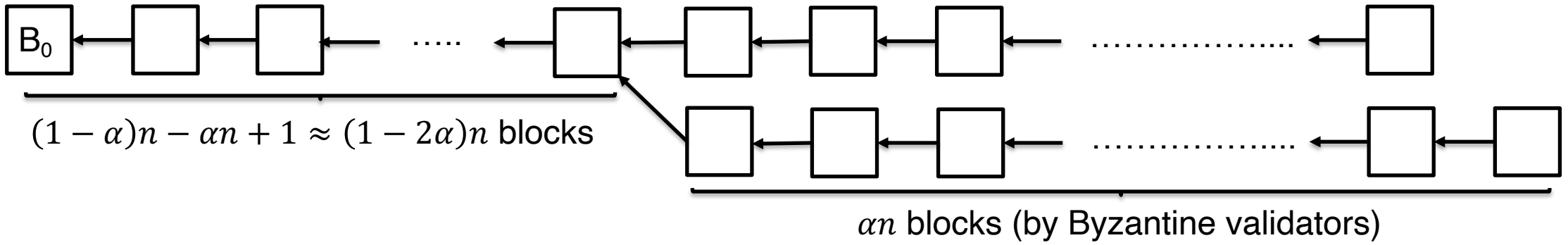


- Byzantine validators "cancel" the last $\approx \alpha n$ such blocks and replace them with αn blocks of their own

Chain Quality

Bad news: longest-chain ➔ chain quality can be as bad as $\frac{1-2\alpha}{1-\alpha}$.

Example: honest validators add $(1-\alpha)n$ blocks to longest chain:



- Byzantine validators "cancel" the last $\approx \alpha n$ such blocks and replace them with αn blocks of their own
  - longest chain grows by $\approx (1-\alpha)n$ blocks, of which $\approx (1-2\alpha)n$ were created by honest validators ➔ chain quality $\approx {}^{1-2\alpha}/_{1-\alpha}$

Guarantees for Longest-Chain Consensus

Recall: 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

Recall: 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, as we've seen)

Guarantees for Longest-Chain Consensus

Recall: 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, as we've seen)

1. Synchronous network.

Guarantees for Longest-Chain Consensus

Recall: 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, as we've seen)

1. Synchronous network.
2. $< 50\%$ Byzantine validators.

Guarantees for Longest-Chain Consensus

Recall: 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, as we've seen)

1. Synchronous network.
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

Recall: 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, as we've seen)

1. Synchronous network.
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]

Conclusion: longest-chain consensus is consistent,

Guarantees for Longest-Chain Consensus

Recall: 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, as we've seen)

1. Synchronous network.
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]

Conclusion: longest-chain consensus is consistent, live,

Guarantees for Longest-Chain Consensus

Recall: for a security parameter k ≥ 1, finalized txs = all txs in the longest chain, except for those in the last k blocks.

Assumptions: (all necessary, as we've seen)

1. Synchronous network.
2. < 50% Byzantine validators.
3. k large enough that, in every interval of ≥ 2k+2 views, a strict majority of the leaders are honest. [e.g., (n/2)-1 suffices]

Conclusion: longest-chain consensus is consistent, live, and guarantees chain quality $\geq \frac{1-2\alpha}{1-\alpha}$.

Notes on the Proof

Longest-chain consensus: synchrony, <50% Byzantine validators, k sufficiently large ➔ consistent, live, chain quality $\geq \frac{1-2\alpha}{1-\alpha}$.

Notes on the Proof

Longest-chain consensus: synchrony, <50% Byzantine validators, k sufficiently large ➔ consistent, live, chain quality $\geq \frac{1-2\alpha}{1-\alpha}$.

Key property: the common prefix property.

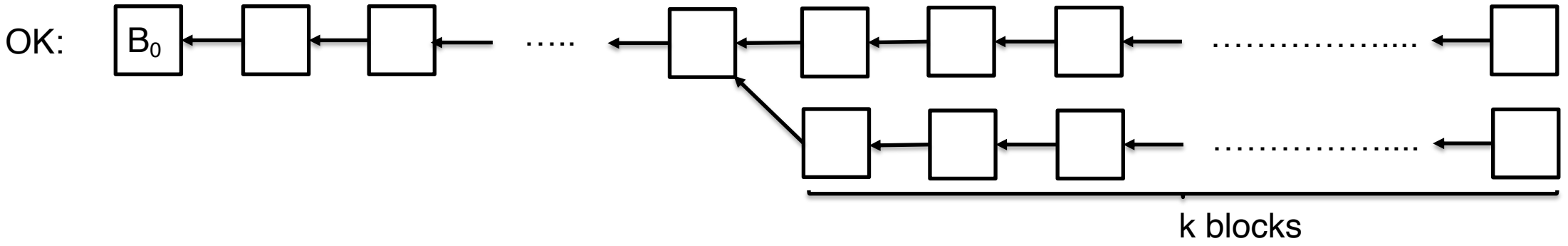
- any two longest chains disagree only on their last ≤ k blocks

Notes on the Proof

Longest-chain consensus: synchrony, <50% Byzantine validators, k sufficiently large ➔ consistent, live, chain quality $\geq \frac{1-2\alpha}{1-\alpha}$.

Key property: the common prefix property.

- any two longest chains disagree only on their last ≤ k blocks

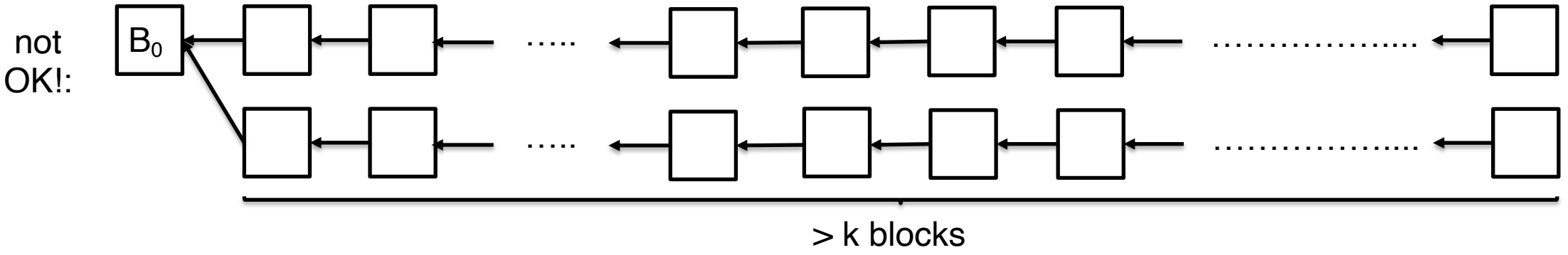


Notes on the Proof

Longest-chain consensus: synchrony, <50% Byzantine validators, k sufficiently large ➔ consistent, live, chain quality $\geq \frac{1-2\alpha}{1-\alpha}$.

Key property: the common prefix property.

- any two longest chains disagree only on their last ≤ k blocks



Notes on the Proof

Longest-chain consensus: synchrony, <50% Byzantine validators, k sufficiently large ➔ consistent, live, chain quality $\geq \frac{1-2\alpha}{1-\alpha}$.

Key property: the common prefix property.

- any two longest chains disagree only on their last ≤ k blocks

Easier version: proof-of-work implementation.

- PoW cryptographically prevents leader equivocation (cf., signatures)

Notes on the Proof

Longest-chain consensus: synchrony, <50% Byzantine validators, k sufficiently large ➔ consistent, live, chain quality $\geq \frac{1-2\alpha}{1-\alpha}$.

Key property: the common prefix property.

- any two longest chains disagree only on their last ≤ k blocks

Easier version: proof-of-work implementation.

- PoW cryptographically prevents leader equivocation (cf., signatures)

Harder version: permissioned/proof-of-stake implementations.

- Byzantine leaders can equivocate, but guarantees still hold (harder proofs)