Blockchain Economics

Joseph Abadi and Markus Brunnermeier

August 2, 2022

Motivation

- Problem in record-keeping: Create trusted *ledger* w/o trustworthy *record-keepers*
  - **Traditional model**: Ledger's owner has to be given incentives to behave
  - **Distributed ledgers**: "Trust problem" shifts to decentralized group of record-keepers

Motivation

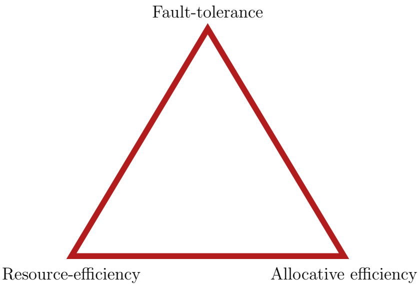
- Problem in record-keeping: Create trusted *ledger* w/o trustworthy *record-keepers*
  - **Traditional model**: Ledger's owner has to be given incentives to behave
  - **Distributed ledgers**: "Trust problem" shifts to decentralized group of record-keepers
- DLs often use costly schemes to provide incentives for honest record-keeping
  - **Proof-of-work**: Voting power allocated based on computational expenditures (BTC, ETH)
    - Expenditures large in practice!
  - **Proof-of-stake**: Voting power allocated based on token holdings (Solana, Cardano)
    - Typically record-keepers ("validators") restricted in their transactions. Also costly?

Motivation

- Problem in record-keeping: Create trusted *ledger* w/o trustworthy *record-keepers*
  - **Traditional model**: Ledger's owner has to be given incentives to behave
  - **Distributed ledgers**: "Trust problem" shifts to decentralized group of record-keepers
- DLs often use costly schemes to provide incentives for honest record-keeping
  - **Proof-of-work**: Voting power allocated based on computational expenditures (BTC, ETH)
    - Expenditures large in practice!
  - **Proof-of-stake**: Voting power allocated based on token holdings (Solana, Cardano)
    - Typically record-keepers ("validators") restricted in their transactions. Also costly?
- What are the fundamental **tradeoffs** and constraints in **distributed ledger design**?
  - Do distributed ledgers have to use costly schemes to incentivize honesty? (e.g. Bitcoin)
  - How should record-keeping be designed to most efficiently provide incentives?

The Blockchain Trilemma

- We study design of **record-keeping protocols** for distributed ledgers (**consensus algs.**)
  - Model general enough to capture PoW/PoS/centralized blockchains



1. **Fault-tolerance**: Ledger can be updated even when computers are offline/malfunction
2. **Resource-efficiency**: No waste of electricity to update ledger
3. **Allocative efficiency**: Record-keeping protocol implements Pareto-efficient allocations

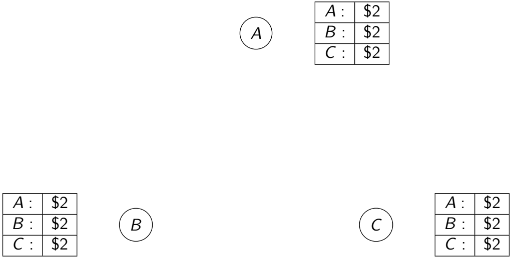
Related literature

- **Distributed consensus**: Ben-Or (1983); Bracha and Toueg (1985); Castro and Liskov (1998); Fisher, Lynch, and Paterson (1985); Lamport, Shostak, and Pease (1980, 1982)
- **Game-theoretic approaches**: Biais et al. (2021); Brown-Cohen et al. (2019); Eyal and Sirer (2014); Halaburda, He, and Li (2021); Nakamoto (2008)
- **(Un)mediated communication**: Aumann and Hart (2004); Ben-Porath (1998, 2003); Eliaz (2002); Forges (1986); Gerardi (2004); Maskin (1998); Myerson (1986)

Roadmap

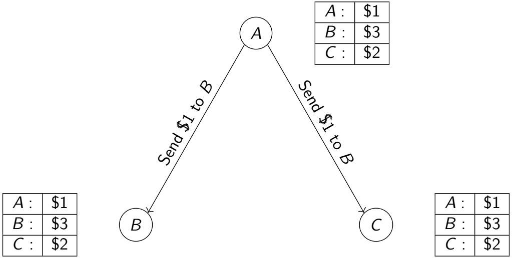
A simple example

- Alice (A), Bob (B), and Carol (C) exchange "tokens" on a digital ledger



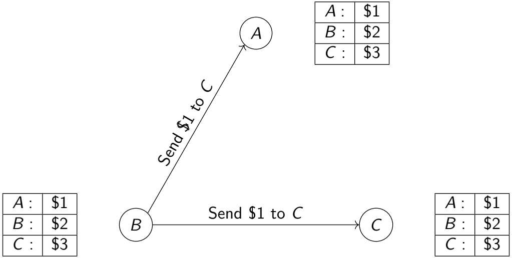
A simple example

- Alice (A), Bob (B), and Carol (C) exchange "tokens" on a digital ledger



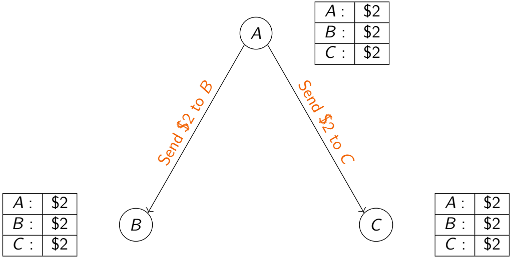
A simple example

- Alice (A), Bob (B), and Carol (C) exchange "tokens" on a digital ledger



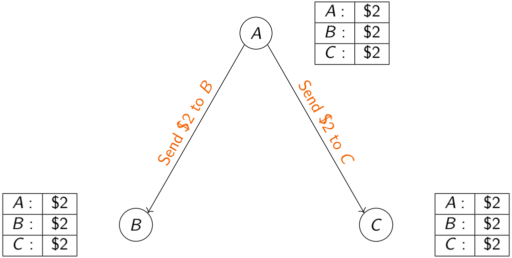
A simple example

- Alice (A), Bob (B), and Carol (C) exchange "tokens" on a digital ledger
  - **Problem**: What if Alice tries to send the same token twice? (**double-spending**)



A simple example

- Alice (A), Bob (B), and Carol (C) exchange "tokens" on a digital ledger
  - **Problem**: What if Alice tries to send the same token twice? (**double-spending**)



- **Naïve solution #1**: Everyone accepts whichever transaction Alice sent first

- **Naïve solution #2**: Accept new transaction only after **unanimous** vote

The classical approach

- N computers ("nodes") keep track of updates to a ledger
  - Ledger: Sequence of entries $\{b_1, b_2, \ldots, b_K\}$
  - E.g., blockchains are ledgers whose entries are transaction batches ("blocks")
- **Communication frictions**: Message delays + "faulty" nodes + asynchronicity
  - Messages are delivered with a random lag (Naïve solution #1)
  - Faulty nodes can't communicate or behave erratically (Naïve solution #2)
  - Nodes don't have synchronized clocks
- Want a communication protocol s.t. when all non-faulty nodes follow it,
  1. All **non-faulty** nodes' ledgers remain consistent
  2. As long as enough nodes are non-faulty, they can update their ledgers (**fault-tolerance**)

Model overview

- Setting with N agents who
  - Engage in a sequence of transactions ("ledger updates"), then
  - Decide how to split a fixed surplus (terminal "ledger state," represents future payoffs)
- Agents play a communication game to reach agreement on transactions + terminal state
  - Same communication frictions as in classical problem
  - Messages can be costly to send (e.g. Proof-of-Work)
- Want game form + communication protocol s.t.
  1. Agents reach agreement on a sequence of transactions + terminal state
  2. **Agents have incentives to follow communication protocol** (coaliton-proof eqm. concept)

Possible to achieve fault-tolerance, resource-efficiency, and allocative efficiency?

Roadmap

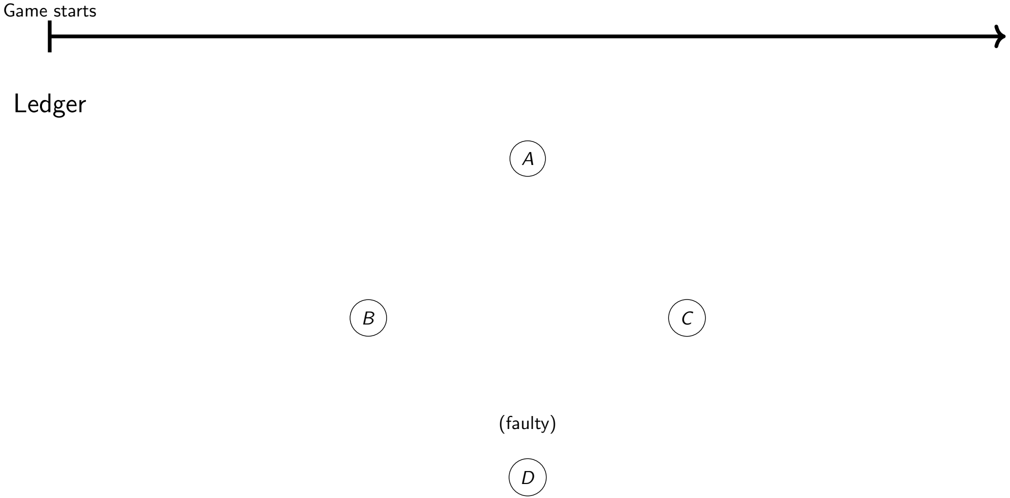
Environment

- Agents $\mathcal{N} = \{1, \ldots, N\}$, continuous time t, no discounting
- Time runs until agents reach agreement on a **terminal state** (split of fixed surplus V)
  - Terminal state is $\mathbf{v} = (v_1, \ldots, v_N)$ with $\sum_{n=1}^{N} v_n = V$
- Set of **transactions** $y \in \mathcal{Y}$ that can be realized before terminal state is reached
  - Each transaction associated with set of **participants** $S(y) \subset \mathcal{N}$ + payoffs $u_n(y)$ for $n \in S(y)$
- At $t = 0$, Nature draws a set of *feasible transactions* $Y^F \subset \mathcal{Y}$ and *faulty agents* $F \subset \mathcal{N}$
  - **Feasible allocation**: $\{y_1, y_2, \ldots, y_K, \mathbf{v}\}$ s.t. each y_k is feasible and $S(y_k)$ are non-faulty

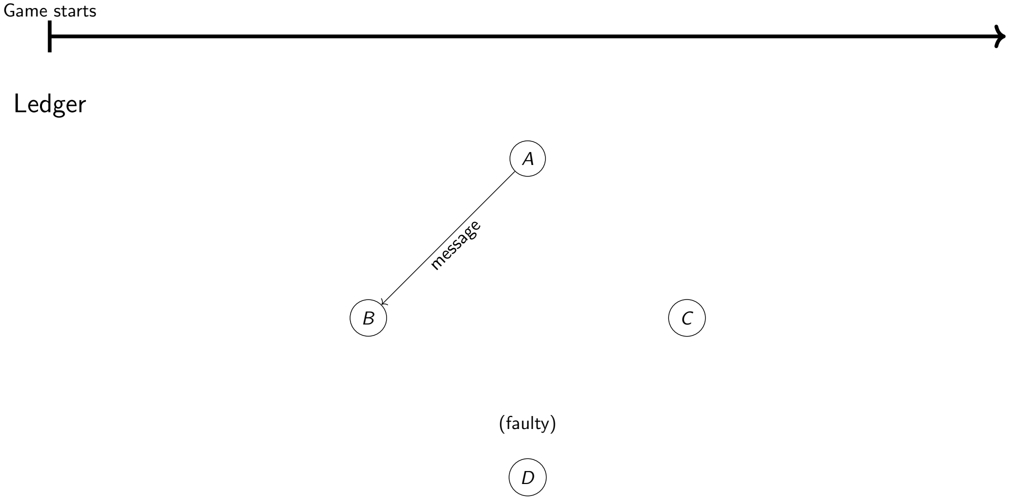
The communication game: Overview

- Agents can send **bilateral, private messages** + **agree** to transactions/terminal states
  - Each message m has a cost $\kappa(m) \geq 0$
  - All participants $n \in S(y)$ agree to transaction $y \Rightarrow$ Payoffs $u_n(y)$ realized
  - **All** agents agree to terminal state $\mathbf{v} \Rightarrow$ Payoffs v_n realized, game ends (**consensus**)
- **Assumption 1**: Two frictions in communication
  1. Messages are delivered with an iid **random lag** (of at most Δ)
  2. **Faulty agents** can't send messages or agree to transactions
     - Can only agree to terminal state at end of game
- **Assumption 2**: Agents don't have perfectly synchronized clocks (don't observe t)
  - Agent n also doesn't know which other agents n' are faulty
  - ... but n has perfect recall of own actions, messages received, transactions s.t. $n \in S(y)$

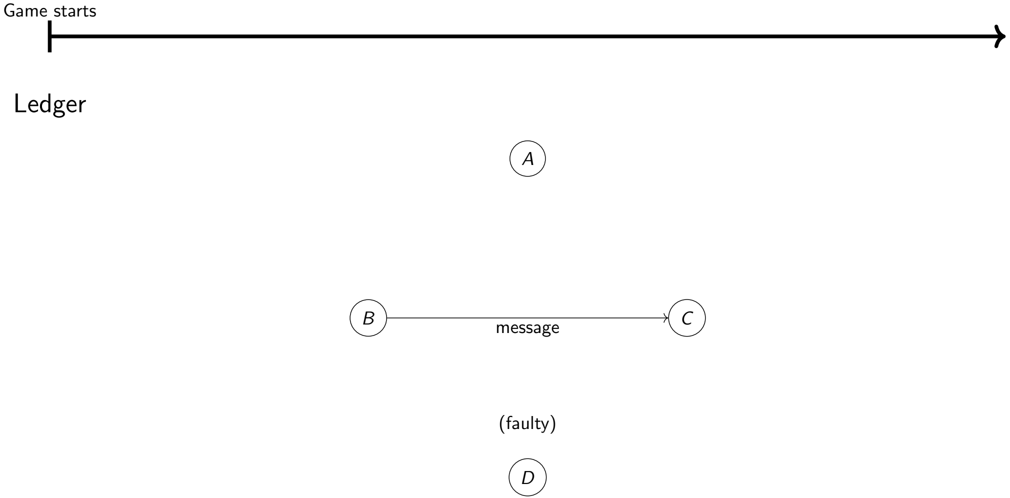
An example



An example

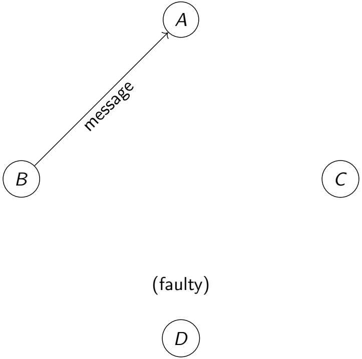


An example

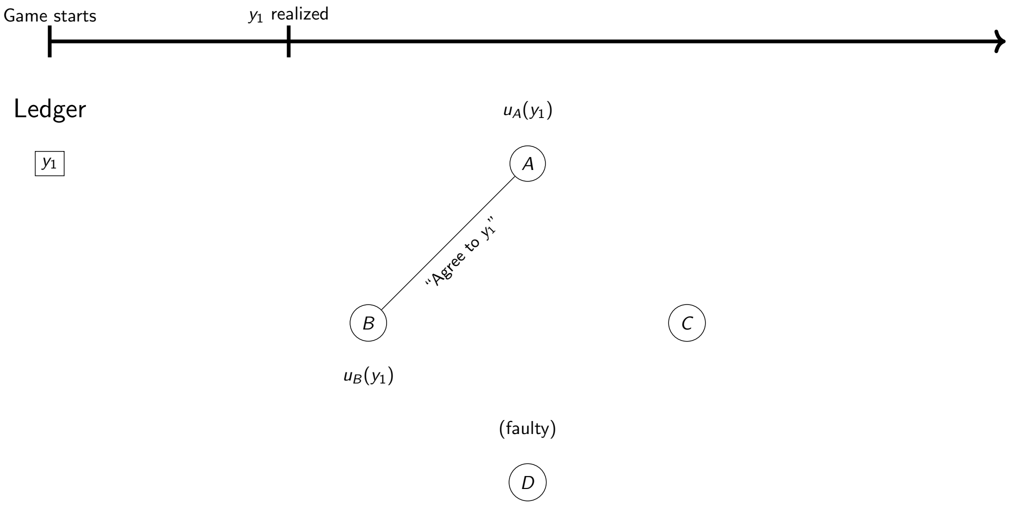


An example

Ledger

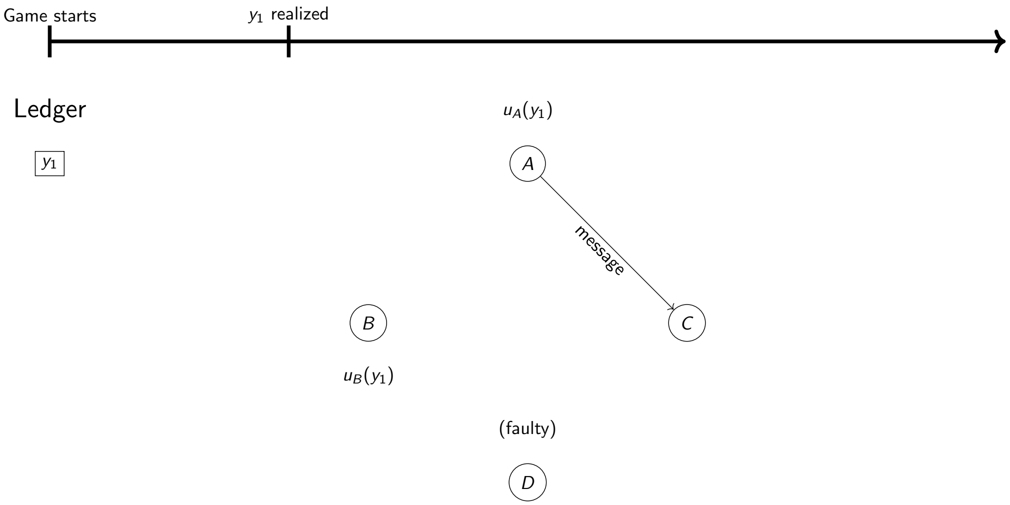


An example



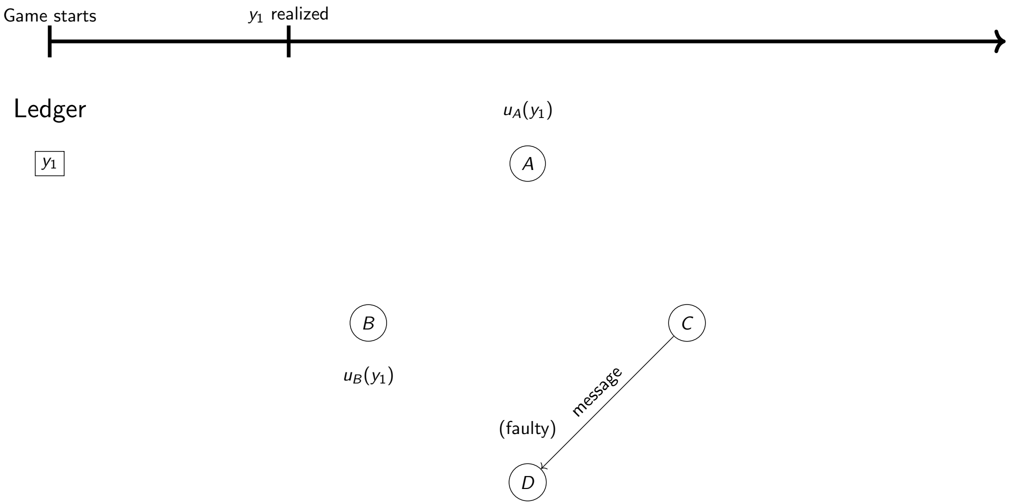
Game starts
y_1 realized
Ledger
y_1
$u_A(y_1)$
A
"Agree to y_1"
B
C
$u_B(y_1)$
(faulty)
D

An example



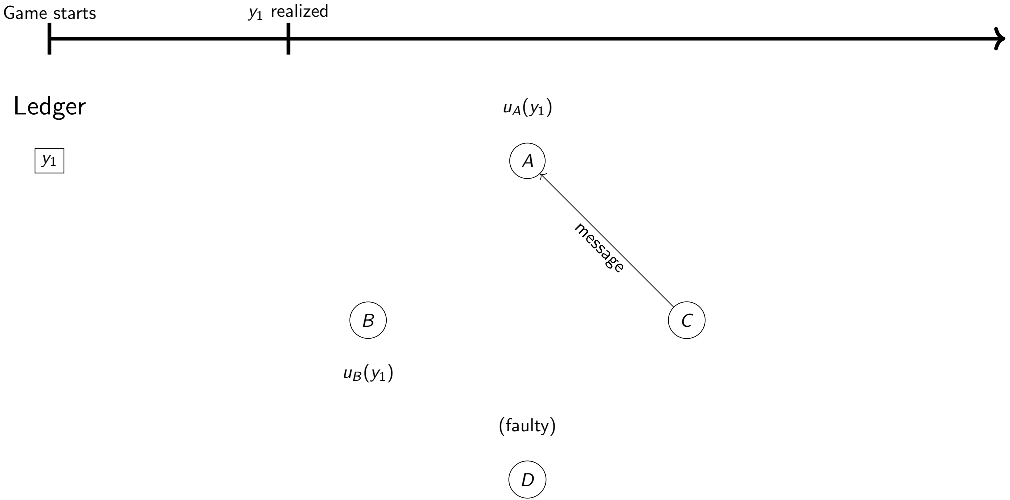
Game starts
y_1 realized
Ledger
y_1
$u_A(y_1)$
A
message
B
C
$u_B(y_1)$
(faulty)
D

An example

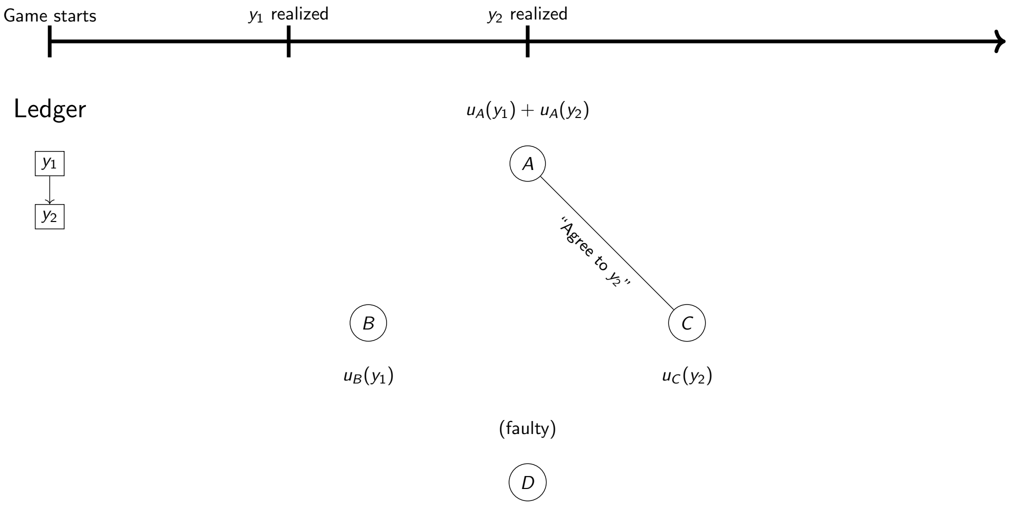


Game starts
y_1 realized
Ledger
y_1
$u_A(y_1)$
A
B
$u_B(y_1)$
C
message
(faulty)
D

An example

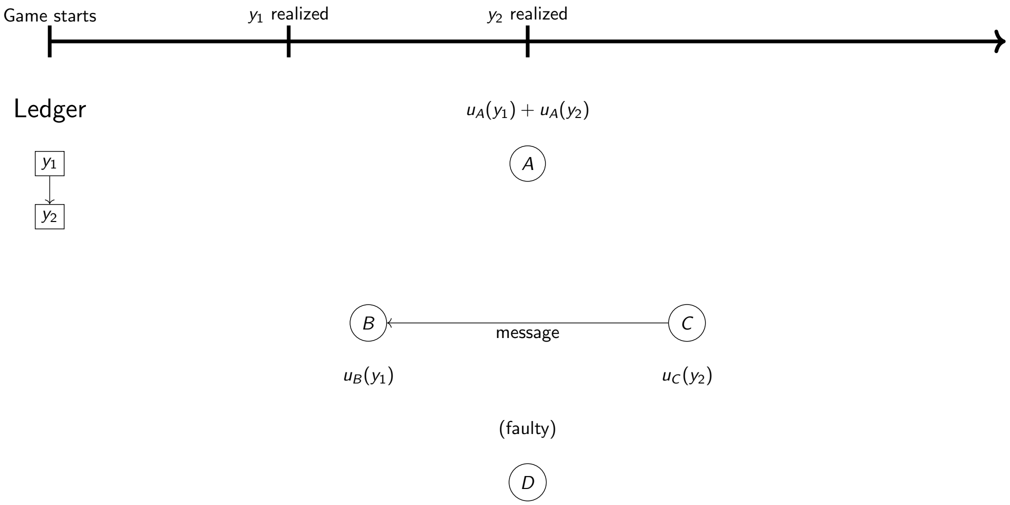

Game starts
y_1 realized
Ledger
y_1
$u_A(y_1)$
A
message
B
C
$u_B(y_1)$
(faulty)
D

An example

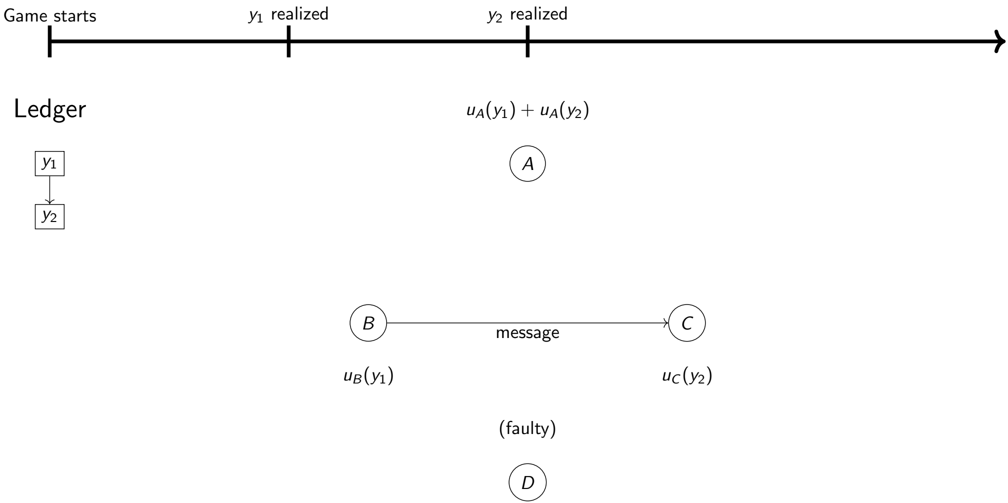


Game starts
y_1 realized
y_2 realized
Ledger
y_1
y_2
$u_A(y_1) + u_A(y_2)$
A
"Agree to y_2"
B
C
$u_B(y_1)$
$u_C(y_2)$
(faulty)
D

An example

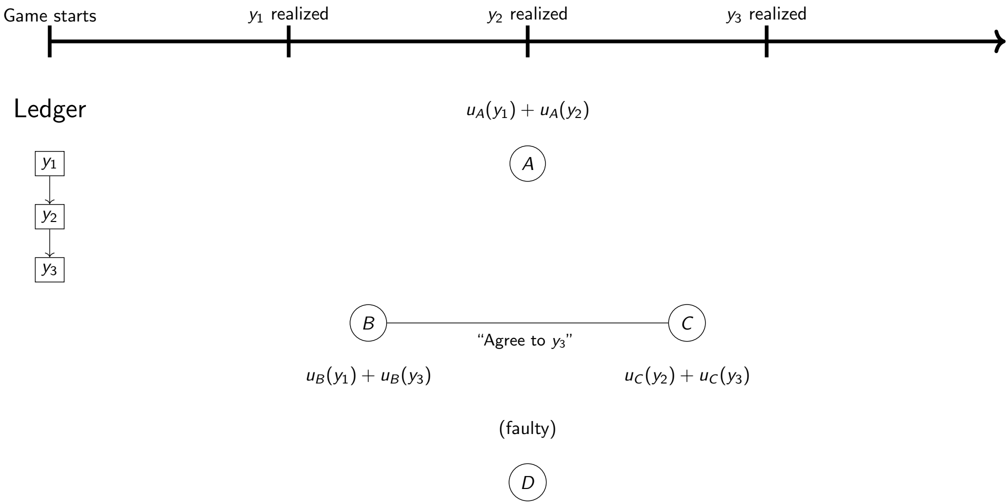

Game starts
y_1 realized
y_2 realized
Ledger
y_1
y_2
$u_A(y_1) + u_A(y_2)$
A
B
message
C
$u_B(y_1)$
$u_C(y_2)$
(faulty)
D

An example

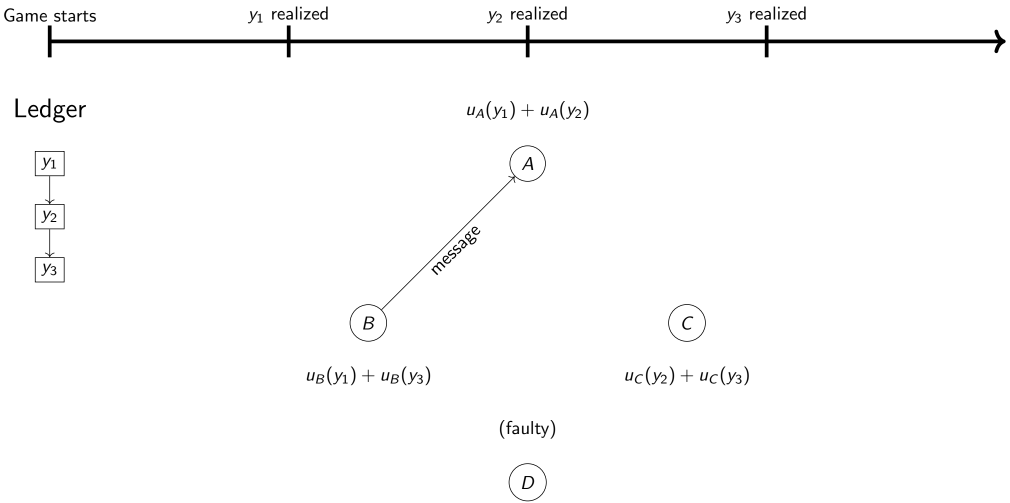


Game starts
y_1 realized
y_2 realized
Ledger
y_1
y_2
$u_A(y_1) + u_A(y_2)$
A
B
message
C
$u_B(y_1)$
$u_C(y_2)$
(faulty)
D

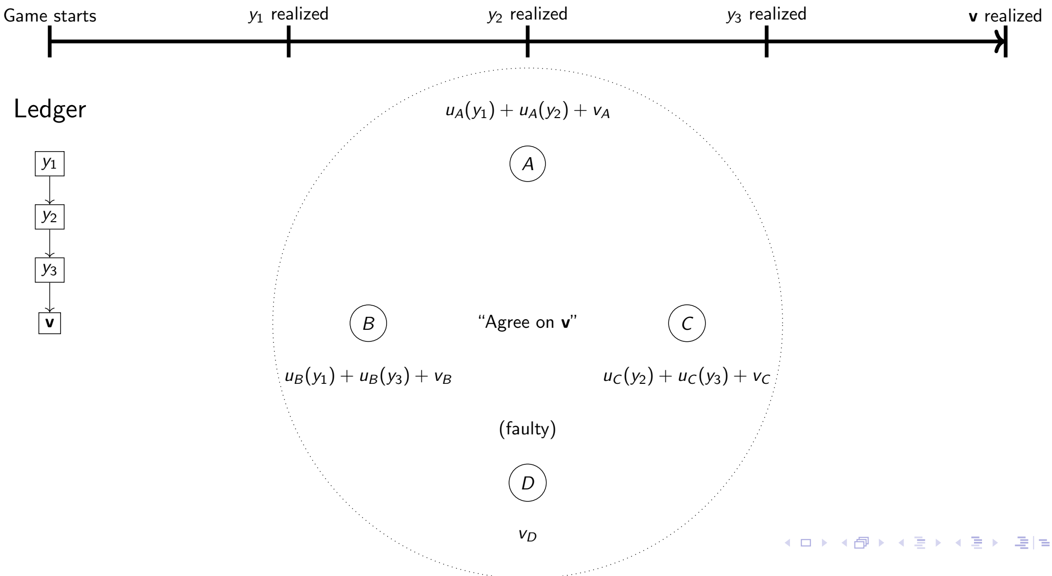
An example


Game starts
y_1 realized
y_2 realized
y_3 realized
Ledger
y_1
y_2
y_3
$u_A(y_1) + u_A(y_2)$
A
B
C
"Agree to y_3"
$u_B(y_1) + u_B(y_3)$
$u_C(y_2) + u_C(y_3)$
(faulty)
D

An example


Game starts
y_1 realized
y_2 realized
y_3 realized
Ledger
y_1
y_2
y_3
$u_A(y_1) + u_A(y_2)$
A
message
B
C
$u_B(y_1) + u_B(y_3)$
$u_C(y_2) + u_C(y_3)$
(faulty)
D

An example


Game starts
y_1 realized
y_2 realized
y_3 realized
$\mathbf{v}$ realized
Ledger
y_1
y_2
y_3
$\mathbf{v}$
$u_A(y_1) + u_A(y_2) + v_A$
A
B
"Agree on $\mathbf{v}$"
C
$u_B(y_1) + u_B(y_3) + v_B$
$u_C(y_2) + u_C(y_3) + v_C$
(faulty)
D
v_D

The communication game: Formal description

- **Actions**: Messages $M_n(h_t)$ + agreements $A_n(h_t)$ for each agent n at each history h_t
- **Payoffs**: Transactions + terminal state - communication costs

$$U_n = \sum_{n \in S(y_k)} u_n(y_k) + v_n - \sum_{m \in \hat{M}_n} \kappa(m)$$

- **Equilibrium**: Profile of strategies $\boldsymbol{\sigma}$ s.t. no **coalition** $S \subset \mathcal{N}$ has incentives to deviate

$$\begin{aligned} \nexists \;\; \tilde{\boldsymbol{\sigma}}_S \;\; \text{s.t. in instance } (Y^F, F): \quad \mathbb{E}[U_n | \tilde{\boldsymbol{\sigma}}_S, \boldsymbol{\sigma}_{-S}] &\geq \mathbb{E}[U_n | \boldsymbol{\sigma}] \; \forall \; n \in S \\ &> \mathbb{E}[U_n | \boldsymbol{\sigma}] \text{ for some } n \in S. \end{aligned}$$

- **Assumption 3**: Technical restriction of class of games
  - Certain types of proofs allowed (e.g. signatures), but “lies of omission” always possible
  - Still general enough to capture all distributed record-keeping systems in reality

Roadmap

Record-keeping

- Study **record-keeping protocols** $\boldsymbol{\sigma}$ of communication game $\mathcal{G}$
  - After each history h_t, promised payoffs $v_n(h_t)$ for each agent
  - As if agents update a "ledger": Each transaction associated w/a transfer $\mathbf{t} = \mathbf{v}' - \mathbf{v}$
    - **A4**: Any restriction on transfers of value (terminal $\mathbf{v}$) $\Rightarrow$ Inefficient allocation Details

Record-keeping

- Study **record-keeping protocols $\boldsymbol{\sigma}$** of communication game $\mathcal{G}$
  - After each history h_t, promised payoffs $v_n(h_t)$ for each agent
  - As if agents update a "ledger": Each transaction associated w/a transfer $\mathbf{t} = \mathbf{v}' - \mathbf{v}$
    - **A4**: Any restriction on transfers of value (terminal $\mathbf{v}$) $\Rightarrow$ Inefficient allocation Details

- Three desired properties of record-keeping protocol $\boldsymbol{\sigma}$:
  1. **Fault-tolerance**: $\boldsymbol{\sigma}$ is a record-keeping eqm. of $\mathcal{G}$ whenever a majority are non-faulty
  2. **Resource-efficiency**: $\boldsymbol{\sigma}$ doesn't use costly messages
  3. **Allocative efficiency**: Whenever $\boldsymbol{\sigma}$ is a record-keeping eqm., then a Pareto-efficient allocation is realized with positive probability

The Blockchain Trilemma

Theorem

Under Assumptions 1-4, the following hold:

1. *(**Impossibility**) There does not exist a record-keeping protocol $\boldsymbol{\sigma}$ of a game $\mathcal{G}$ achieving fault-tolerance, resource-efficiency, and allocative efficiency.*
2. *(**Existence**) For any two of the desired properties, there exists a record-keeping protocol $\boldsymbol{\sigma}$ of some game $\mathcal{G}$ achieving both.*

The Blockchain Trilemma

Theorem

Under Assumptions 1-4, the following hold:

1. (**Impossibility**) *There does not exist a record-keeping protocol $\boldsymbol{\sigma}$ of a game $\mathcal{G}$ achieving fault-tolerance, resource-efficiency, and allocative efficiency.*
2. (**Existence**) *For any two of the desired properties, there exists a record-keeping protocol $\boldsymbol{\sigma}$ of some game $\mathcal{G}$ achieving both.*

- Characterizes costs of a lack of **trust**
  - With a trusted mediator, possible to achieve all three properties
- Tradeoff: Fault-tolerance vs. efficiency
  - **Any** amount of fault-tolerance implies some inefficiency

The main idea

- Fault-tolerant communication protocol $\Rightarrow$ Possible for some coalition to "double-spend"
  - Deviating coalition agrees to transfer value to two different groups of agents

$$U_n^D - U_n^H \leq \underbrace{\kappa_n^D}_{\text{Ex-ante cost}} + \underbrace{v_n^H - v_n^D}_{\text{Ex-post punishment}}$$

- **Ex-ante cost**: Communication costs $\Rightarrow$ Expensive to double-spend
  - Need to give up **resource-efficiency** ($\kappa_n^D > 0$)
- **Ex-post punishment**: Take value away from agents who double-spend
  - Need to prevent agents from spending entire balance ($v_n^H = 0$) $\Rightarrow$ **Allocative inefficiency**

Proof sketch: Impossibility

- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)

$A:$	v_A
$B:$	v_B
$C:$	v_C

A

- **A4**: $\exists\ y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul

$A:$	v_A
$B:$	v_B
$C:$	v_C

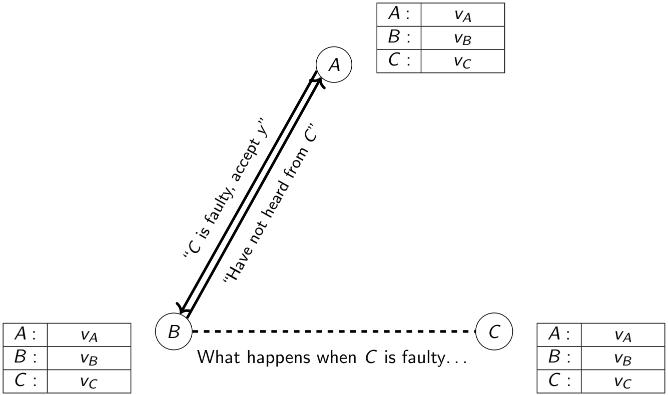
B

C

$A:$	v_A
$B:$	v_B
$C:$	v_C

Proof sketch: Impossibility

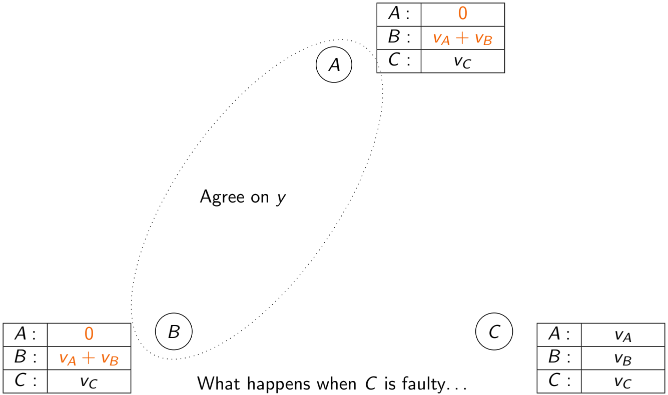
- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)



- **A4**: $\exists\, y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty

Proof sketch: Impossibility

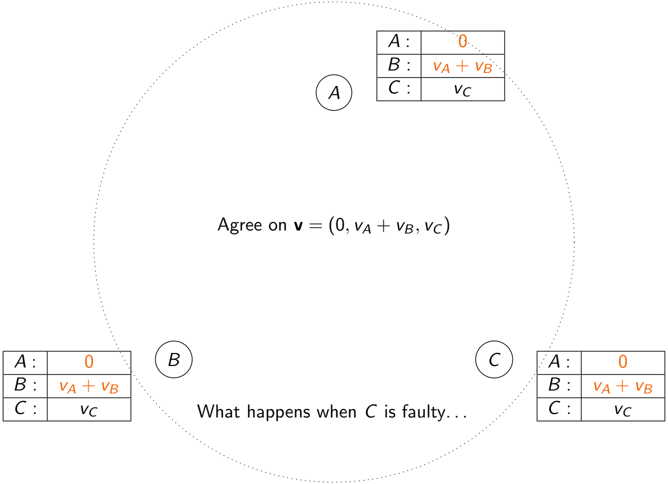
- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)



- **A4**: $\exists\ y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty

Proof sketch: Impossibility

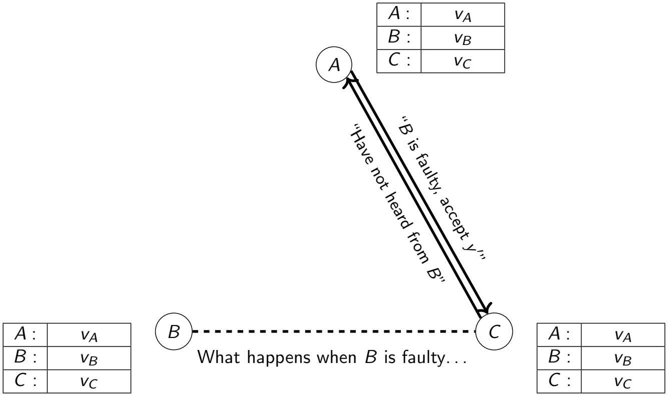
- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)



- **A4**: $\exists\ y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty

Proof sketch: Impossibility

- Suppose σ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)

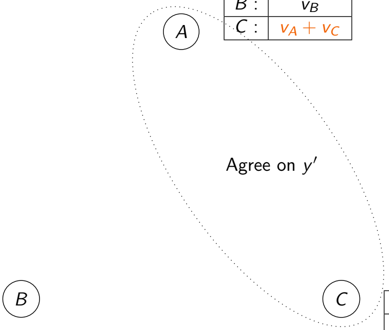


- **A4**: $\exists\, y$ s.t. A transfers v_A to B
- **FT**: σ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty

Proof sketch: Impossibility

- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)

$A:$	0
$B:$	v_B
$C:$	$v_A + v_C$



$A:$	v_A
$B:$	v_B
$C:$	v_C

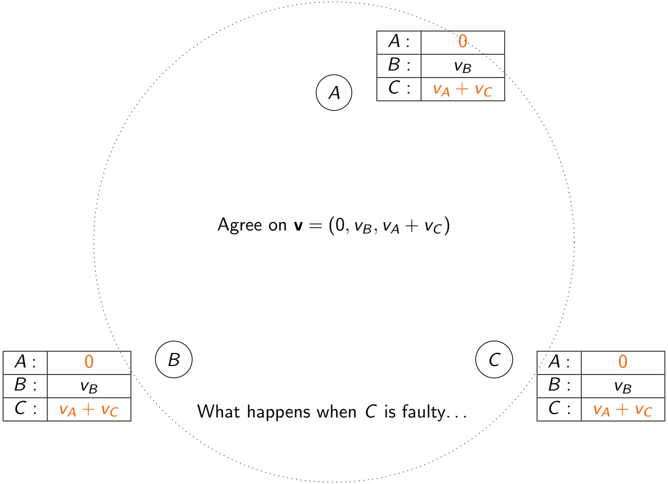
What happens when B is faulty...

$A:$	0
$B:$	v_B
$C:$	$v_A + v_C$

- **A4**: $\exists\ y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty

Proof sketch: Impossibility

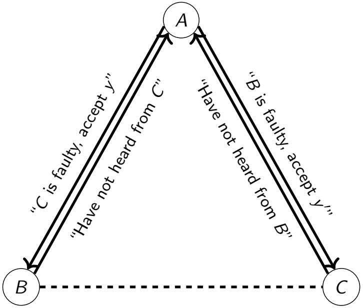
- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)



- **A4**: $\exists\ y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty

Proof sketch: Impossibility

- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)



A :	v_A
B :	v_B
C :	v_C

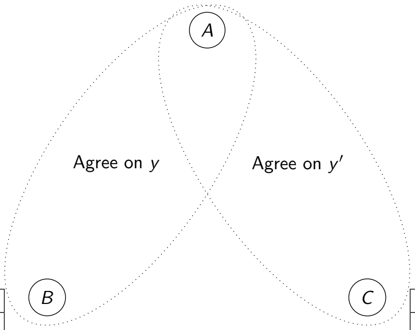
A :	v_A
B :	v_B
C :	v_C

- **A4**: $\exists\ y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty
- **A2**: B and C don't know how long to wait!

Proof sketch: Impossibility

- Suppose $\boldsymbol{\sigma}$ achieves fault-tolerance (FT), resource-efficiency (RE), and allocative efficiency (AE)

A :	0
B :	$v_A + v_B$
C :	v_C



A :	0
B :	v_B
C :	$v_A + v_C$

- **A4**: $\exists\ y$ s.t. A transfers v_A to B
- **FT**: $\boldsymbol{\sigma}$ is an eqm. when $A \cup B$ are non-faul
- **AE**: $A \cup B$ agree on y when C is faulty
- **A2**: B and C don't know how long to wait!
- **RE**: Costless for A to engage in this deviati

Intuition behind the existence result

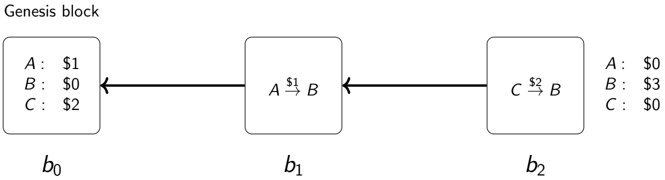
- Give up **fault-tolerance** $\Rightarrow$ Easy to prevent double-spending!
  - Simple communication protocol: Require unanimous vote before approving any transaction
- Impossibility result: There exist **mutually incompatible** efficient allocations
  - Permitting A to transfer its entire balance to anyone allows double-spending
  - One option: Forbid mutually incompatible allocations, give up **allocative efficiency**
- Can design communication costs so that **any** set of allocations is compatible
  - Restore allocative efficiency, give up **resource efficiency**

Key assumptions Conclusion

Roadmap

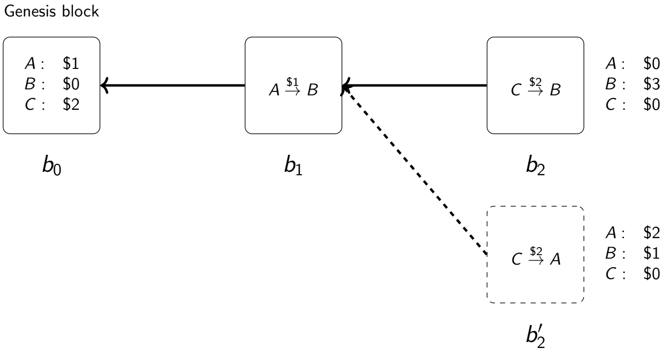
What's a blockchain?

- Blockchain: Type of data structure (ledger) consisting of a sequence of blocks
  - Block: Consists of data + pointer to the previous block
  - Each block is usually a batch of transactions



What's a blockchain?

- Blockchain: Type of data structure (ledger) consisting of a sequence of blocks
  - Block: Consists of data + pointer to the previous block
  - Each block is usually a batch of transactions



- **Challenge**: What if conflicting blocks are added at the end of the chain (**fork**)?
  - **Consensus algorithm**: Communication protocol to **finalize** blocks

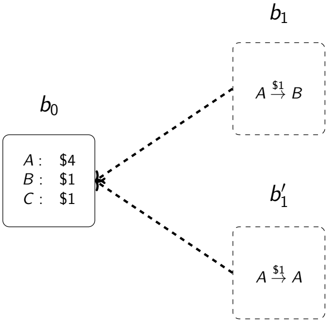
Mapping blockchains to the Trilemma

- This section: For proof-of-work/proof-of-stake systems, specify
  1. How does the consensus algorithm work?
  2. Under the consensus algorithm, which coalitions can collude to double-spend?
  3. What incentives prevent those coalitions from doing so?
- Useful to think of consensus algorithms as consisting of two components:
  - **Write protocol**: Who gets to add the next block? Where should it be added?
  - **Read protocol**: At what point is a block on one branch considered to be final?
- **Note**: A different type of double-spend is more common in practice
  - Attackers wait until one transaction is confirmed and goods are delivered...
  - ...then attackers send tokens back **to themselves**, create consensus on that transaction

The Proof-of-Work algorithm

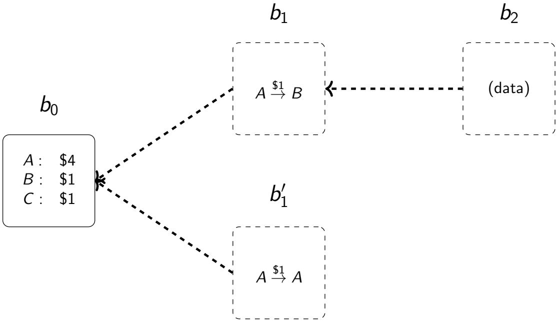
- In PoW blockchains, when are blocks finalized? How are forks resolved?
  - PoW is by far the most popular consensus algorithm despite high mining costs
  - E.g. Bitcoin, Ethereum (for now), Litecoin
- **Write protocol**: "Longest chain rule"
  - Miners should attempt to add a block at the end of the longest chain they currently see
  - Logic: Miners tacitly vote in favor of every block in a chain when extending it
- **Read protocol**: "k confirmations"
  - A block b is final if there are (at least) k blocks following it ("confirmations")
    - For example, $k = 6$ in Bitcoin
  - Effectively, a block is confirmed once it gets six votes

An example of the PoW consensus algorithm



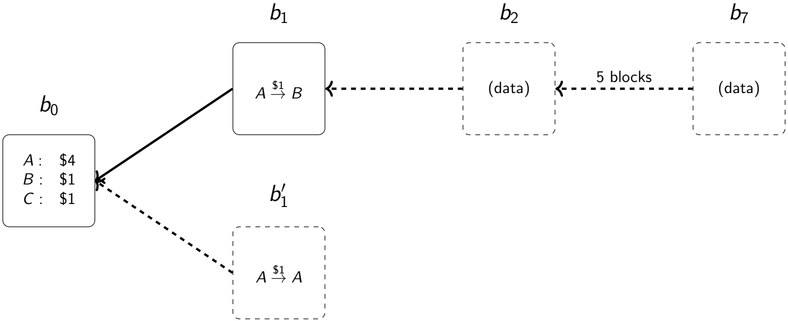
- Initially there's a fork...

An example of the PoW consensus algorithm



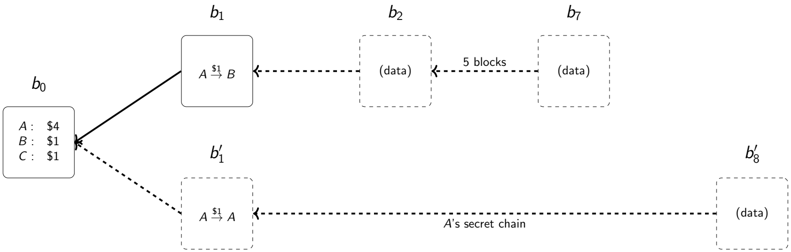
- Initially there's a fork...
- Then a miner adds b_2 after b_1...

An example of the PoW consensus algorithm



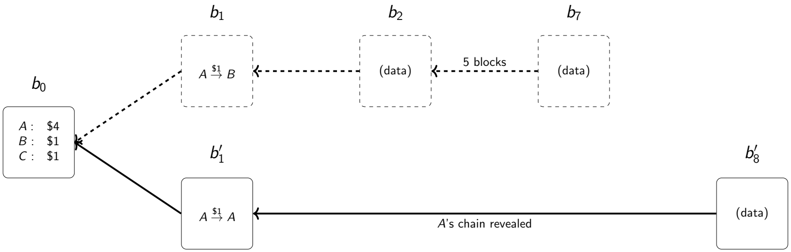
- Initially there's a fork...
- Then a miner adds b_2 after b_1...
- Then miners add blocks to the longest chain in sequence until b_1 is final.

A double-spend attempt



- Alice first mines a chain secretly. . .

A double-spend attempt

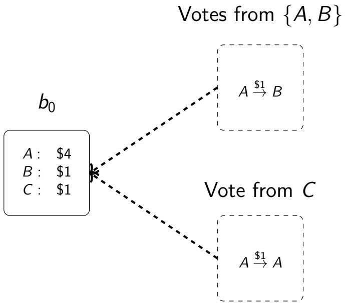


- Alice first mines a chain secretly. . .
- . . . and then reveals it. If Alice controls majority of hash power, she can double-spend.

The Proof-of-Stake algorithm

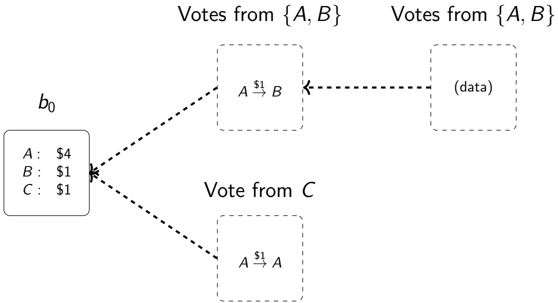
- In PoS, record-keepers perform two functions: "Forging" and "validating"
  - Forging: Adding new blocks to the chain
  - Validating: Attesting that blocks forged by others are valid
- **Write protocol**: Longest chain rule
  - Token drawn at random $\Rightarrow$ Token's owner gets to mine a block b (add to longest chain)
  - Other tokenholders should attest to the validity of block b if it is on the longest chain
- **Read protocol**: Supermajority rule $+$ k confirmations
  - A block b is considered final if:
    1. Two-thirds of validators (weighted by token holdings) have attested b is valid
    2. Block b is followed by at least k blocks

An example of PoS consensus



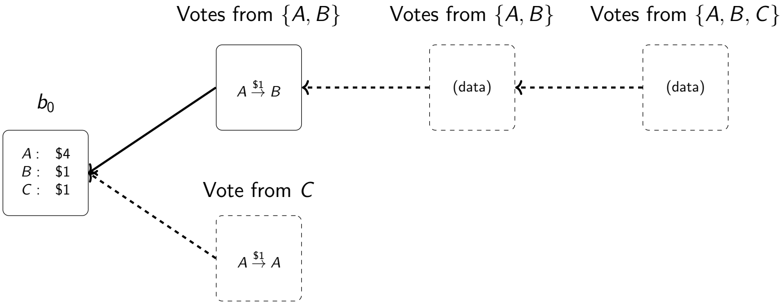
- Alice and Bob first vote for (b_1, b_2), while Carol votes for $b'_1 \ldots$

An example of PoS consensus



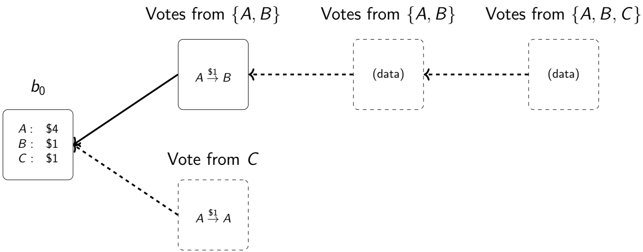
- Alice and Bob first vote for (b_1, b_2), while Carol votes for b_1'...
- ...but then Carol sees b_2 and votes along with Alice and Bob.

An example of PoS consensus



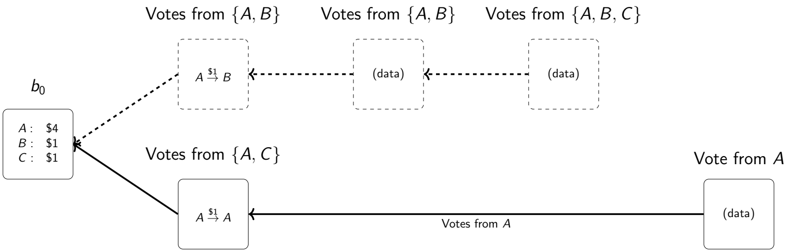
- Alice and Bob first vote for (b_1, b_2), while Carol votes for b_1'...
- ...but then Carol sees b_2 and votes along with Alice and Bob.

Double-spending in PoS



- Alice has $\frac{2}{3}$ of tokens $\Rightarrow$ Can validate any block on her own
- Alice can actually attempt a double-spend whenever she has $\geq \frac{1}{3}$ of tokens

Double-spending in PoS



- Alice has $\frac{2}{3}$ of tokens $\Rightarrow$ Can validate any block on her own
- Alice can actually attempt a double-spend whenever she has $\geq \frac{1}{3}$ of tokens

Punishments in PoW and PoS systems

- **PoW**: Need resource costs to be large enough to dissuade double-spends
  - Easy to measure resource costs in practice: Total hash power is observable
  - Bitcoin $>$ Argentina, Ethereum $\approx$ Netherlands
- **PoS**: Two types of schemes
  1. Force validators to stake collateral (Avalanche, Solana)
     - Cost $\approx$ Liquidity premium $\times$ Collateral quantity
  2. Validators earn rents that can be taken away (Cardano)

Roadmap

The fault-tolerance requirement

- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;

$A \qquad \begin{array}{ll} A: & x \\ B: & ? \\ C: & ? \end{array}$

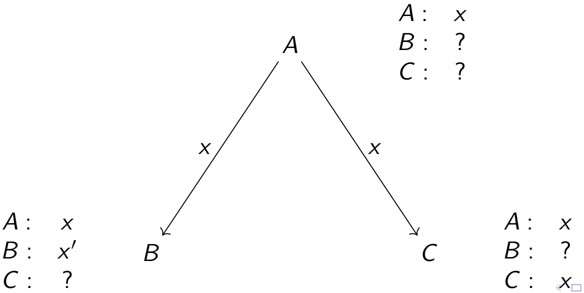
$\begin{array}{ll} A: & ? \\ B: & x' \\ C: & ? \end{array} \qquad B$

$C \qquad \begin{array}{ll} A: & ? \\ B: & ? \\ C: & x \end{array}$

The fault-tolerance requirement

- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

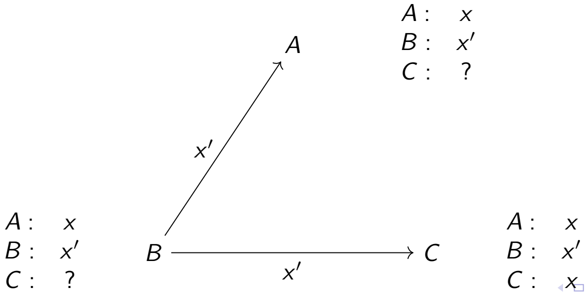
- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;



The fault-tolerance requirement

- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

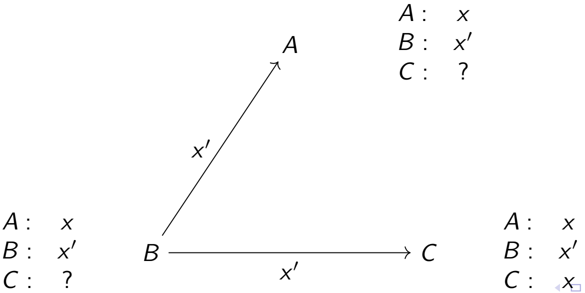
- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;



The fault-tolerance requirement

- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

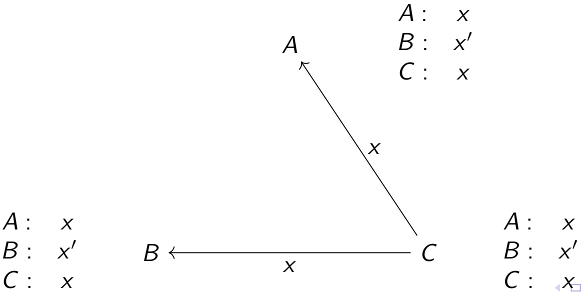
- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;



The fault-tolerance requirement

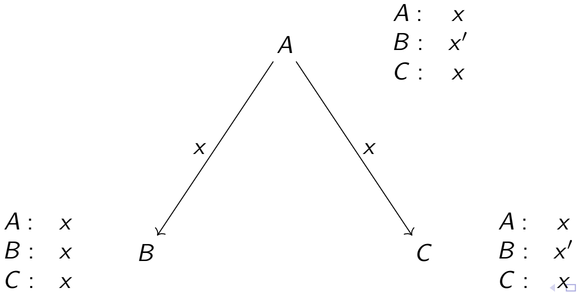
- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;



The fault-tolerance requirement

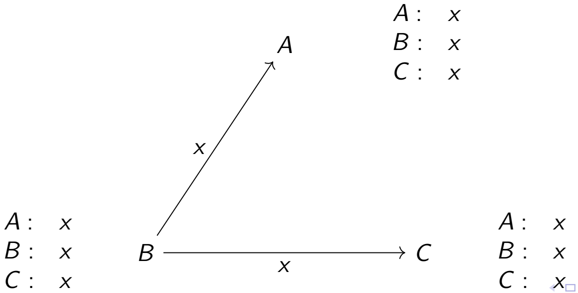
- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?
- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;



The fault-tolerance requirement

- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;



The fault-tolerance requirement

- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;
  2. Once they have received confirmation from **all** other players that a particular allocation x^* should be finalized, agree to x^*.

A

$A: \; x$
$B: \; x$
$C: \; x$

$A: \; x$
$B: \; x$
$C: \; x$

B

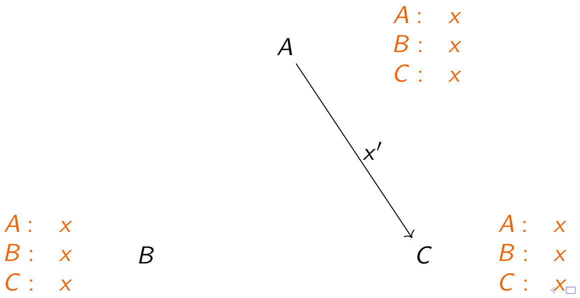
C

$A: \; x$
$B: \; x$
$C: \; x$

The fault-tolerance requirement

- **Fault-tolerance** is a key requirement in the Blockchain Trilemma
  - Why is this important? What happens if we give up fault-tolerance?

- Simple algorithm to achieve consensus in the absence of faults: Each player n should
  1. Communicate to others to determine which allocation x should be finalized;
  2. Once they have received confirmation from **all** other players that a particular allocation x^* should be finalized, agree to x^*.
  3. After agreeing to x^*, never agree to anything else.



A general result

- Double-spending impossible when fault-tolerance isn't required!
  - An agent receives input from **all** others before deciding
  - Construct consensus alg. so that no two honest agents ever agree to different allocations
- Why does fault-tolerance allow double-spends? Can't require input from everyone
  - ...so two honest agents can decide without ever hearing from each other (e.g. B and C)
- **Result**: Blockchain Trilemma holds even if faulty players can behave in arbitrary ways
  - Generalizes beyond simple model where faulty players are offline (e.g. glitches, hacks, ...)
  - **Key feature**: All that's needed is *possibility* of non-responsiveness

The asynchronicity assumption

- Is it hopeless to design an ideal fault-tolerant consensus alg.? No!
  - **Asynchronicity** is also a critical assumption (designer does not know message lag Δ)

- Suppose players have synchronized clocks, and consider the following protocol:
  1. "If more than Δ seconds have passed without receiving a message from n, label n as faulty and ignore thereafter."

A

$A:\ x$
$B:\ ?$
$C:\ ?$

$A:\ ?$
$B:\ x'$
$C:\ ?$

B

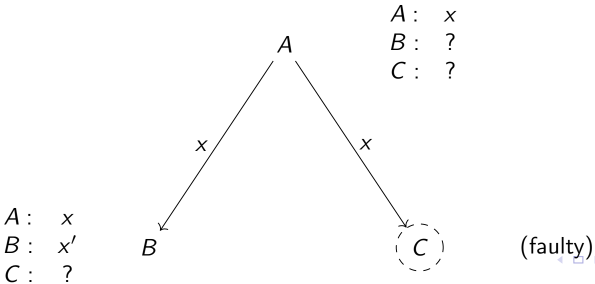
C

(faulty)

The asynchronicity assumption

- Is it hopeless to design an ideal fault-tolerant consensus alg.? No!
  - **Asynchronicity** is also a critical assumption (designer does not know message lag Δ)

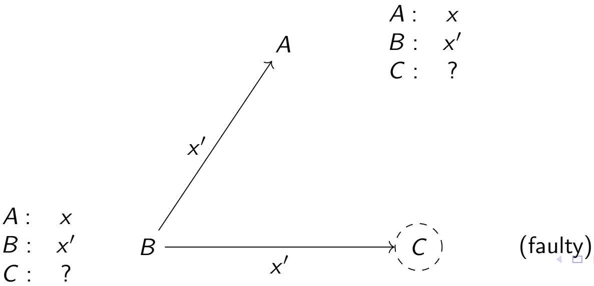
- Suppose players have synchronized clocks, and consider the following protocol:
  1. "If more than Δ seconds have passed without receiving a message from n, label n as faulty and ignore thereafter."



The asynchronicity assumption

- Is it hopeless to design an ideal fault-tolerant consensus alg.? No!
  - **Asynchronicity** is also a critical assumption (designer does not know message lag Δ)

- Suppose players have synchronized clocks, and consider the following protocol:
  1. "If more than Δ seconds have passed without receiving a message from n, label n as faulty and ignore thereafter."



The asynchronicity assumption

- Is it hopeless to design an ideal fault-tolerant consensus alg.? No!
  - **Asynchronicity** is also a critical assumption (designer does not know message lag Δ)
- Suppose players have synchronized clocks, and consider the following protocol:
  1. "If more than Δ seconds have passed without receiving a message from n, label n as faulty and ignore thereafter."
  2. "After receiving confirmation from all **non-faulty** players that a particular allocation x^* should be finalized, agree to x^*."

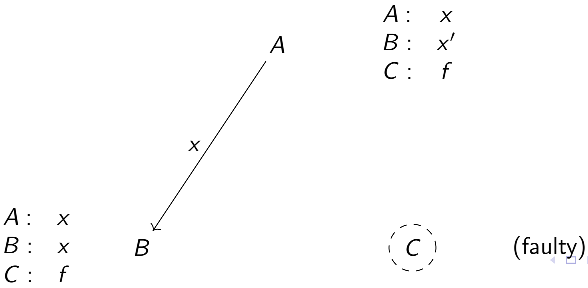
A $\quad$ $\begin{array}{ll} A: & x \\ B: & x' \\ C: & f \end{array}$

$\begin{array}{ll} A: & x \\ B: & x' \\ C: & f \end{array}$ $\quad$ B $\qquad$ C $\quad$ (faulty)

The asynchronicity assumption

- Is it hopeless to design an ideal fault-tolerant consensus alg.? No!
  - **Asynchronicity** is also a critical assumption (designer does not know message lag Δ)

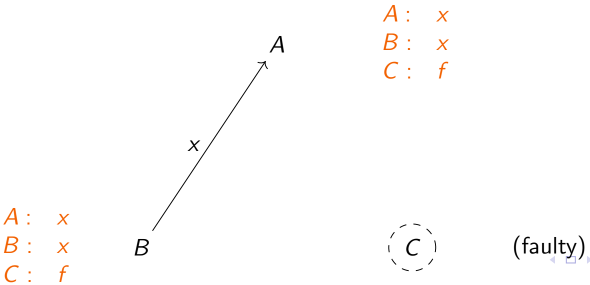
- Suppose players have synchronized clocks, and consider the following protocol:
  1. "If more than Δ seconds have passed without receiving a message from n, label n as faulty and ignore thereafter."
  2. "After receiving confirmation from all **non-faulty** players that a particular allocation x^* should be finalized, agree to x^*."



The asynchronicity assumption

- Is it hopeless to design an ideal fault-tolerant consensus alg.? No!
  - **Asynchronicity** is also a critical assumption (designer does not know message lag Δ)

- Suppose players have synchronized clocks, and consider the following protocol:
  1. "If more than Δ seconds have passed without receiving a message from n, label n as faulty and ignore thereafter."
  2. "After receiving confirmation from all **non-faulty** players that a particular allocation x^* should be finalized, agree to x^*."
  3. "After agreeing to x^*, do not agree to anything else."



Scalability in synchronous settings

- In practical settings, asynchronicity is usually the most appropriate assumption
  - Protocol requires strong form of common knowledge $\Rightarrow$ Need **perfectly** synchronized clocks
  - Any error implies some users would be left out of ledger **forever**
- What if we relax the asynchronicity assumption?
  - Possible to resolve Trilemma, but comes at the cost of **scalability** (key challenge)
  - Intuition: In order to prevent double-spends, amount of cross-checking scales with N

Theorem

Under synchronous communication, $\exists$ a game $\mathcal{G}$ and a protocol $\boldsymbol{\sigma}$. achieving fault-tolerance, resource efficiency, and full transferability. However, any such algorithm takes at least $\Delta \cdot \frac{N}{3}$ rounds of communication.

Roadmap

Conclusion

- What are the inherent constraints and tradeoffs in the design of digital record-keeping?
  - Blockchain Trilemma $\Rightarrow$ Either give up fault-tolerance. . .
  - . . . or provide incentives at the cost of inefficiency (resource costs/transferability restrictions)
- PoW gives up resource-efficiency, while PoS/permissioned give up allocative efficiency
- Trilemma applies generally to **all** fault-tolerant distributed record-keeping systems
  - Fundamental result in **consensus algorithm design** adapted to econ from comp sci

Technical details

- **Definition** A record-keeping protocol is $\boldsymbol{\sigma}$ specifying $v_n(h_t)$ s.t. $\sum_{n=1}^{N} v_n(h_t) = V$ and

$$\sum_{t'>t} u_n(y_{t'}) + v_{nT} \geq v_n(h_t) \ \forall \ n \in \mathcal{N} \text{ w/prob. } 1$$

in any (Y^F, F) s.t. $\boldsymbol{\sigma}$ is an eqm.

- **Assumption 4**: Restrictions on transfers of value $\Rightarrow$ Loss of efficiency
  - For each transfer of value $\mathbf{t}$, there is an individually rational transaction y s.t.

$$u_n(y) = -t_n \ \forall \ n \text{ s.t. } t_n > 0$$

  - **Participation constraint binds** for all n who incur a cost in transaction y

Back