

Verteilte Systeme/ Distributed Systems

Artur Andrzejak

11

Leader Election Algorithms

Leader Election - Choosing a Coordinator

- ▶ Important for example for ...
 - ▶ Dijkstra's token ring: who has process ID 0?
 - ▶ A centralized algorithm for mutual exclusion - who is the coordinator?
- ▶ Choosing a coordinator is one of the problems that *really* occur

PetersonLeader

- ▶ Uses only unidirectional communication
- ▶ Only needs $O(n \log n)$ messages, with a small constant - about 2
- ▶ Knowledge of n unnecessary
- ▶ Each process has unique ID (UID)
 - ▶ Can be a random number, or e.g. a MAC address
- ▶ Chooses any process as a leader, not necessarily the one with the largest / smallest UID
- ▶ Only comparisons of UIDs are needed

PetersonLeader /2

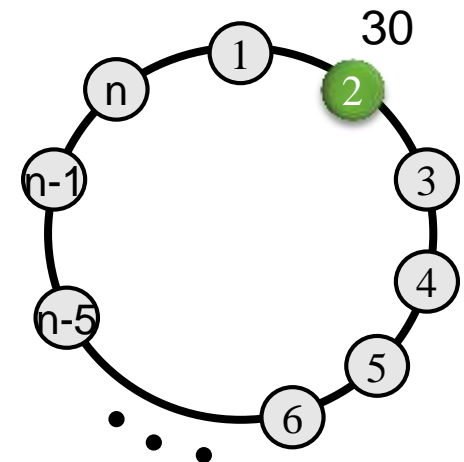
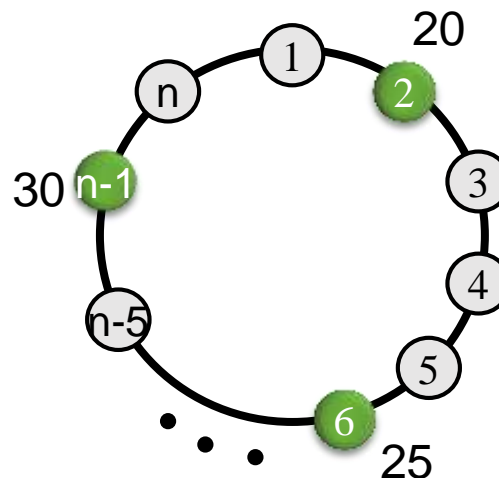
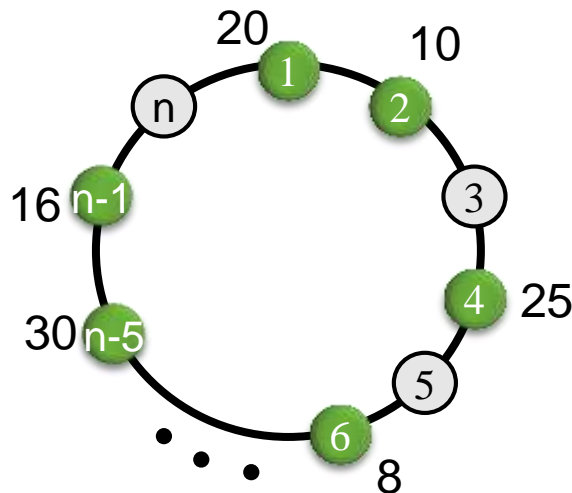
- ▶ Each process is in one of two modes
 - ▶ **active**, does the "real work" of the algorithm
 - ▶ **relay**, just forwarding the messages
- ▶ The execution takes place in *phases*
 - ▶ In each phase, the number of active processes is at least halved
 - ▶ So we are ready after $\sim \log n$ phases
- ▶ Each process has a variable **tempUID**, which is initially set equal to its own UID

PetersonLeader /3

- ▶ In each phase, the following takes place:
 - ▶ Each *relay* process simply forwards the messages
 - ▶ Each *active* process sends the contents of own tempUID clockwise twice
 - ▶ These will be intercepted by two *active* followers
 - ▶ Each *active* process i waits until it has received two messages from *active* predecessors
 - ▶ Let these messages be u_{i-2} , u_{i-1} and $u_i = \text{tempUID}_i$
 - ▶ IF $u_{i-2} < u_{i-1} > u_i$ then
 - i stays in *active* mode and updates own tempUID to $\text{tempUID}_i := u_{i-1}$
 - ▶ ELSE
 - i goes into relay mode

PetersonLeader /4

- ▶ If a process i receives a message from the immediate predecessor that equals its own tempUID, i.e. $u_{i-1} = u_i$, then i chooses itself the leader



PetersonLeader - Analysis

- ▶ Apparently, if more than one process is *active* in one phase, at least one processor will have a combination of UIDs that will leave it *active*
- ▶ At most half of the *active* processes can survive a round
 - ▶ As each surviving process must have an *active* predecessor, which then goes into *relay* mode
- ▶ The correctness is obvious

PetersonLeader - Analysis

- ▶ Offenbar: falls in einer Phase mehr als ein Prozess „active“ ist, wird mind. ein Prozessor eine Kombination von UID's haben, die ihm „active“ bleiben lässt
- ▶ Höchstens die Hälfte der „active“-Prozesse können eine Runde überleben, denn jeder überlebende Prozess muss einen „active“-Vorgänger haben, der dann in „relay“-Modus geht
- ▶ Die Korrektheit ist offensichtlich

PetersonLeader – Analysis /2

- ▶ **Message complexity:**

- ▶ Number of phases is at most $\lfloor \log n \rfloor + 1$
- ▶ At each stage, a process sends ≤ 2 messages
- ▶ So the number of messages is
$$2n(\lfloor \log n \rfloor + 1) \in O(n \log n)$$

- ▶ **Time complexity:**

- ▶ Naive: $O(n \log n (x + d))$
- ▶ More precise analysis: $O(n (x + d))$
- ▶ Here
 - ▶ x = upper bound on the time for a process step
 - ▶ d = upper bound on the time for a channel transmission

FloodMax Algorithm

- ▶ Arbitrary network, only some nodes connected unidirectionally
- ▶ All nodes all know the *diameter* of the network
- ▶ Each node can send to certain neighbors, these are called ***out-nbrs***
- ▶ Each node also has a unique ID (UID)
- ▶ **Algorithm:**
 - ▶ Each process stores the highest UID received so far (initially its own)
 - ▶ In each *round*, this maximum is sent to all *out-nbrs*
 - ▶ After *diameter* many rounds the process appoints itself to Leader, if the largest maximum was its own UID.
 - ▶ Otherwise it becomes a non-leader

FloodMax - Improvement

- ▶ Get we remove the rounds?
- ▶ We simulate rounds
 - ▶ Every process that sends a "round-r message" *appends the round number to the message*
 - ▶ On receipt, each process waits until it has received all "round-r messages" from its neighbors before executing the state transition of the original algorithm
 - ▶ After diameter many rounds one is finished

OptFloodMax – Improved FloodMax

- ▶ Original: After the first round, a process sends its maxUid value *only* if it has received a new maximum value in the last round
- ▶ Naive solution: add a round id to the message as well, and proceed as in synchronous case
- ▶ Almost correct solution: work completely asynchronously - if a process gets a new maxUid it will send it to its neighbors at some time
 - ▶ Works, but processes do not know when they are done

Bitcoin and Blockchain: Introduction

Slides and content based on the course

BerkeleyX: CS198.1x “Bitcoin and Cryptocurrencies”

@ edX: <https://courses.edx.org/courses/course-v1:BerkeleyX+CS198.1x+3T2018/course/>

Authors: NADIR AKHTAR, RUSTIE LIN, MENGYI WANG

Editors: MENGYI WANG

What is Bitcoin?

- ▶ 1. The first and most widely used **cryptocurrency**
 - ▶ Completely decentralized – no bank(s) needed
 - ▶ Motivation:
 - ▶ No need for central control entities (banks)
 - ▶ (Pseudo-) Anonymity
 - ▶ Based on sound cryptographic and economics principles
- ▶ 2. The **unit** of the bitcoin currency (1 BTC)



1 Bitcoin entspricht

3.511,98 Euro

7. Jan., 10:24 UTC · Haftungsausschluss

| | |
|---------|-----------|
| 1 | Bitcoin ▼ |
| 3511.98 | Euro ▼ |



Bitcoin vs. Banks

**Account and
Identity
Management**

Service

**Record
Management**

Trust



Links personal
information to
bank account
and verifies
ownership

Transfers
money and
redeems
money

Updates
and tracks
account
balance

Provides
services by
professionals
under
regulations of
government

Bitcoin vs. Banks

Account and
Identity
Management

Service

Record
Management

Trust



Gives users
autonomously
created and
managed
identities

Sends funds
between
peers directly
(P2P)

Updates every
peer, which
keeps its own
ledger
(blockchain)

Provides trusted
protocol which
incentivizes
actors to behave
honestly

Identity: Authentication + Integrity

- ▶ We need to ensure that all users can **authenticate** themselves through some identification method, and that all identification methods have **integrity**
- ▶ Authentication
 - ▶ ... ensures that no one else acts on your behalf
 - ▶ *Only you* should be able to claim, receive, and spend money
 - ▶ ... can also be used to enforce blaming
 - ▶ If someone tries to withdraw your funds from a bank, you would want a record of this, including the identity of this person
- ▶ Integrity
 - ▶ Means that our authentication methods cannot be replicated by anyone else, e.g. once you sign a check or transaction, no one should be able to manipulate it

Implementing Identity

- ▶ Identity has an “public” and “private” parts:
 - ▶ Houses have addresses and (mailbox) keys
 - ▶ Emails have aliases and passwords
- ▶ Bitcoin has:
 - ▶ Private key chosen at random, 256 bits (secret!)
 - ▶ (Public) address (160 bits) generated from private key
 - ▶ Note: in detail, the address is derived from a 256 bit “public key” (not really public), which is generated from the private key
- ▶ Address for receiving, private key for redeeming
- ▶ The identity in bitcoin is essentially: the private key + the public key + the public address
 - ▶ Anyone can generate many such identities!

Collision of Private Keys

- ▶ Since anyone generates private key in separation, could two users obtain the same private key?
 - ▶ Consider: what is the (smaller) chance that two users have the same 160 bit address?

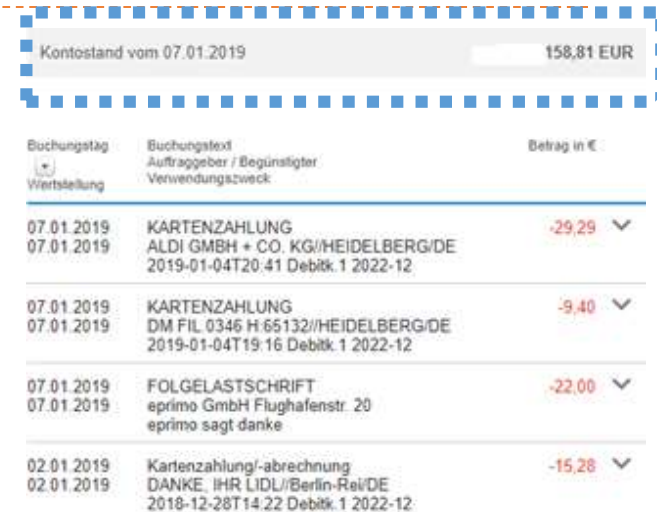
Analogy:



- ▶ Grains of sand on earth: 2^{63}
- ▶ With 2^{63} earths, each with 2^{63} grains of sand: 2^{126} total grains of sand
- ▶ 2^{126} is only 0.00000000058% of 2^{160}
- ▶ Or: every single person could have about 2^{127} addresses *all to themselves*
 - ▶ Population of the world: 7.7 billion in November 2018

Account Models

- ▶ In a traditional bank accounts, all funds are aggregated into one count/balance
- ▶ If we spend or deposit, your balance decreases/increases
- ▶ In Bitcoin, users spend directly from transactions made to them
- ▶ Your “account balance” is a list of “received”-transactions:
 - ▶ “<your-address> got <x> BTC from <address-A>”
 - ▶ “<your-address> got <y> BTC from <address-B>”
 - ▶ “ ... ”
- ▶ There is no aggregated balance/count

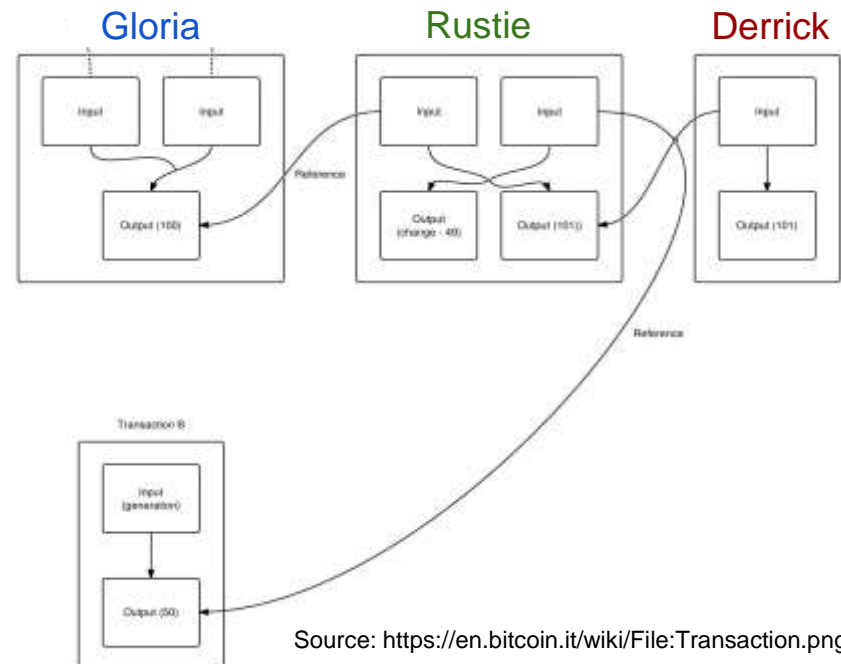


Kontostand vom 07.01.2019 158,81 EUR

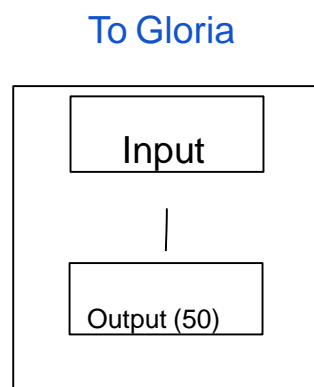
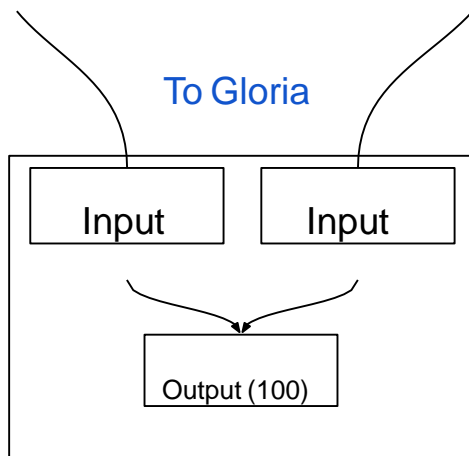
| Buchungstag Verteilung | Buchungstext Auftraggeber / Begünstigter Verwendungszweck | Betrag in € |
|---------------------------|---|-------------|
| 07.01.2019 07.01.2019 | KARTENZAHLUNG ALDI GMBH + CO. KG/HEIDELBERG/DE 2019-01-04T20:41 Debitk. 1 2022-12 | -29,29 |
| 07.01.2019 07.01.2019 | KARTENZAHLUNG DM FIL 0346 H 65132//HEIDELBERG/DE 2019-01-04T19:16 Debitk. 1 2022-12 | -9,40 |
| 07.01.2019 07.01.2019 | FOLGELASTSCHRIFT eprimo GmbH Flughafenstr. 20 eprimo sagt danke | -22,00 |
| 02.01.2019 02.01.2019 | Kartenzahlung/-abrechnung DANKE, IHR LIDL//Berlin-Rei/DE 2018-12-28T14:22 Debitk. 1 2022-12 | -15,28 |

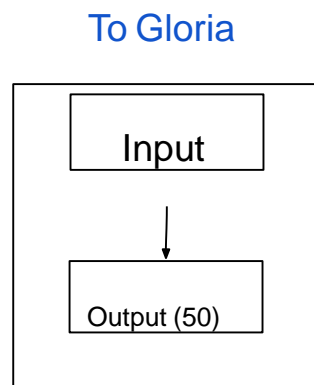
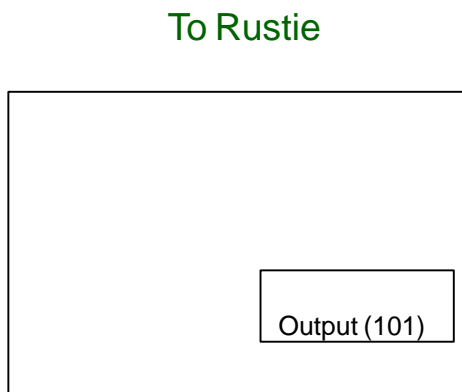
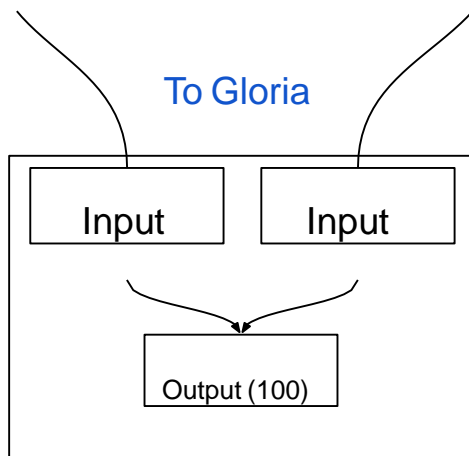
UTXO Model

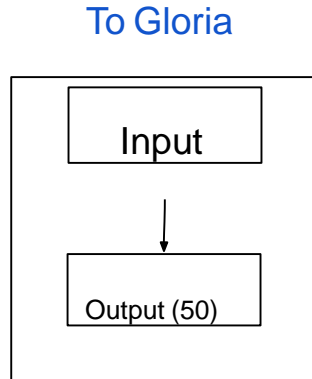
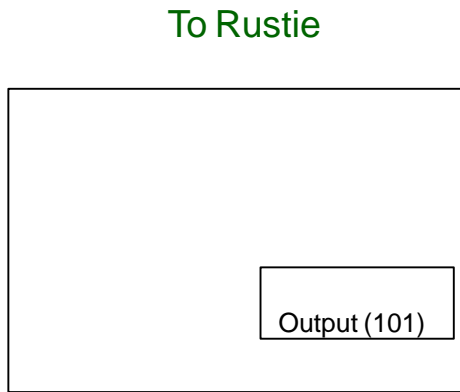
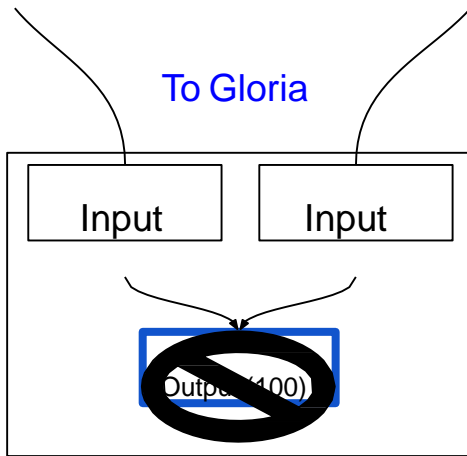
- ▶ Bitcoin's model for transactions is called **Unspent Transaction Output (UTXO)**
- ▶ Example:
 - ▶ Gloria has 150 BTC, received as two transactions over 100 BTC and 50 BTC
 - ▶ She transfers 101 BTC to Rustie
 - ▶ Rustie in turn transfers all to Derrick
- ▶ Is the amount 101 (vs. e.g. 100) problematic?

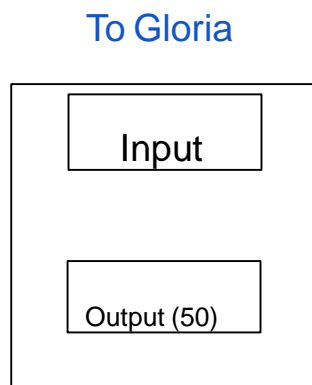
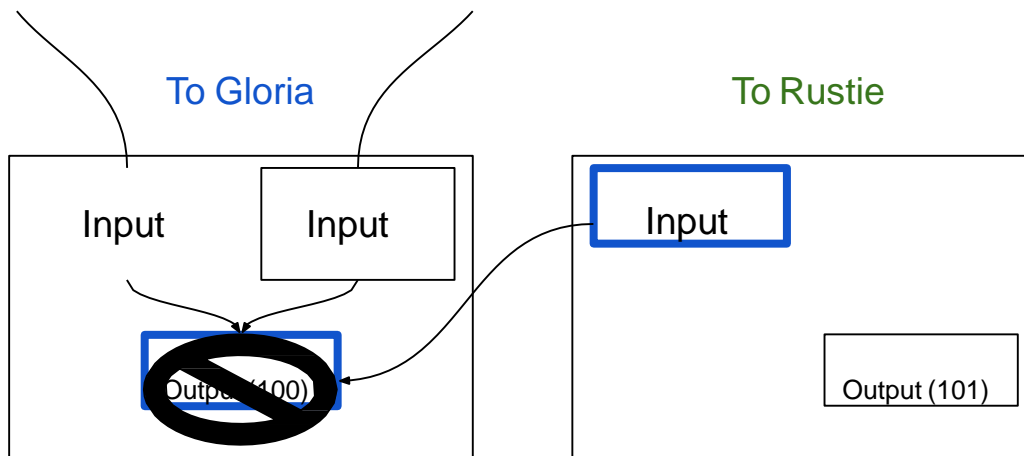


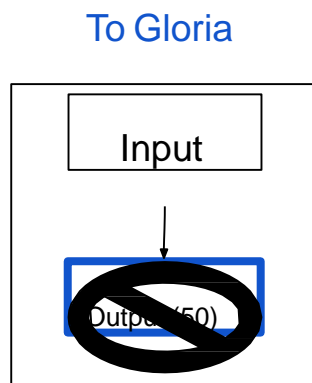
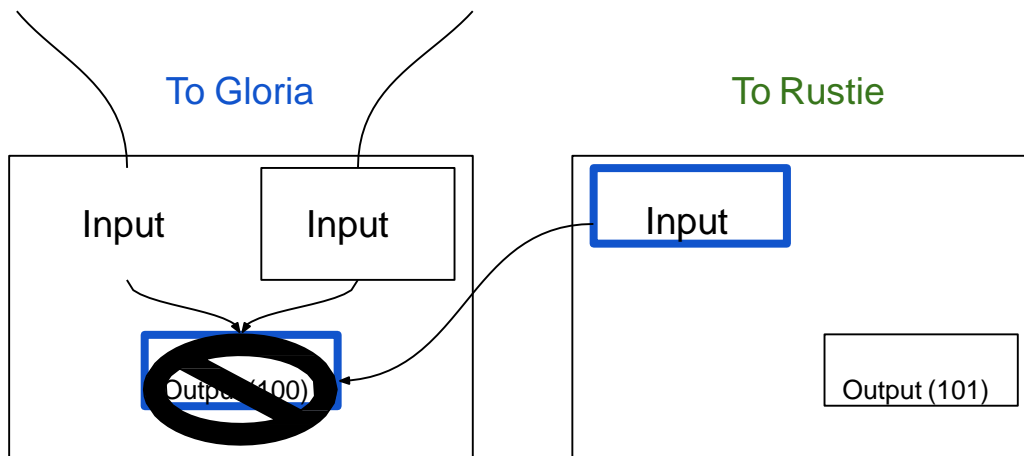
Source: <https://en.bitcoin.it/wiki/File:Transaction.png>

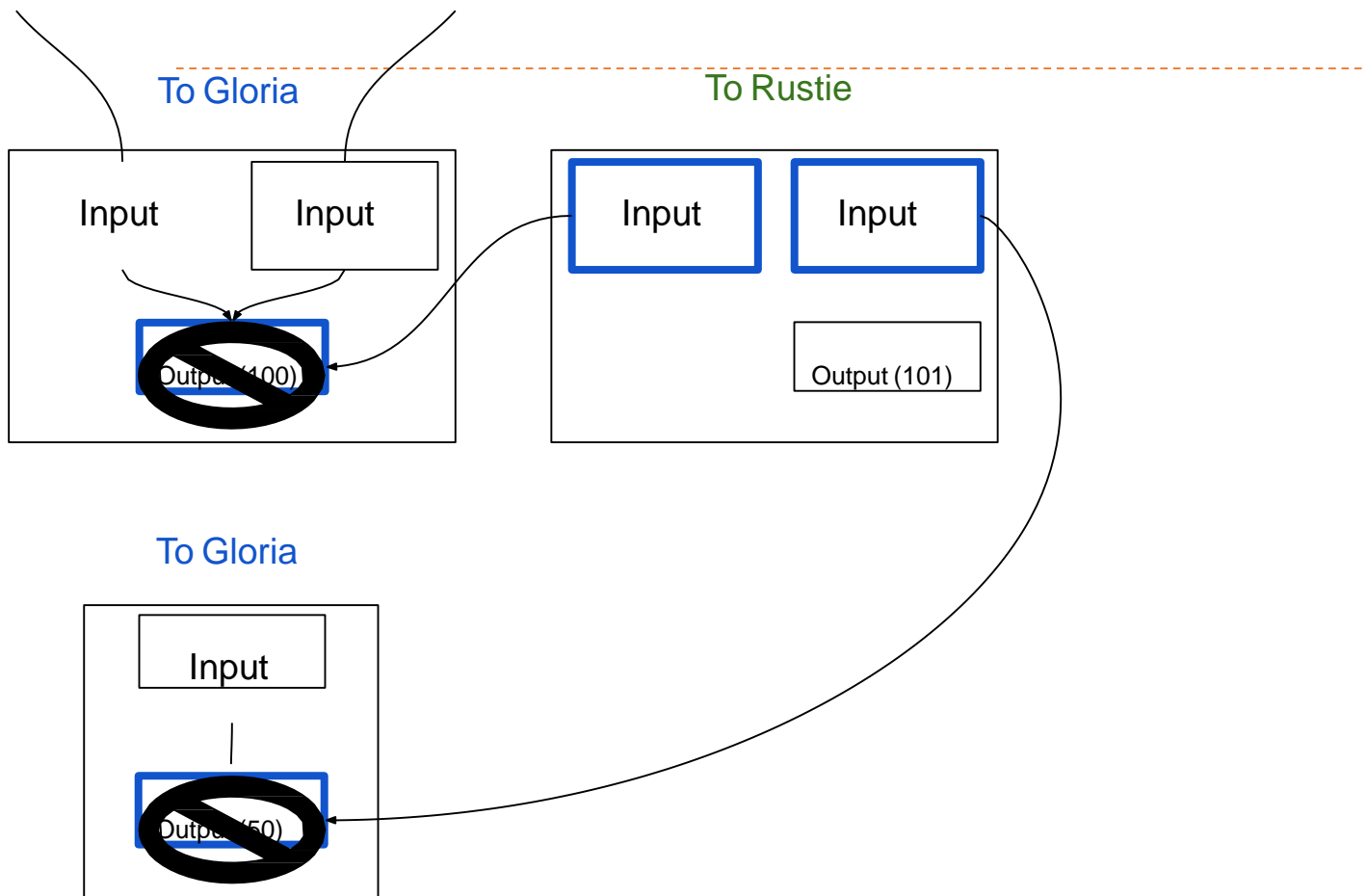


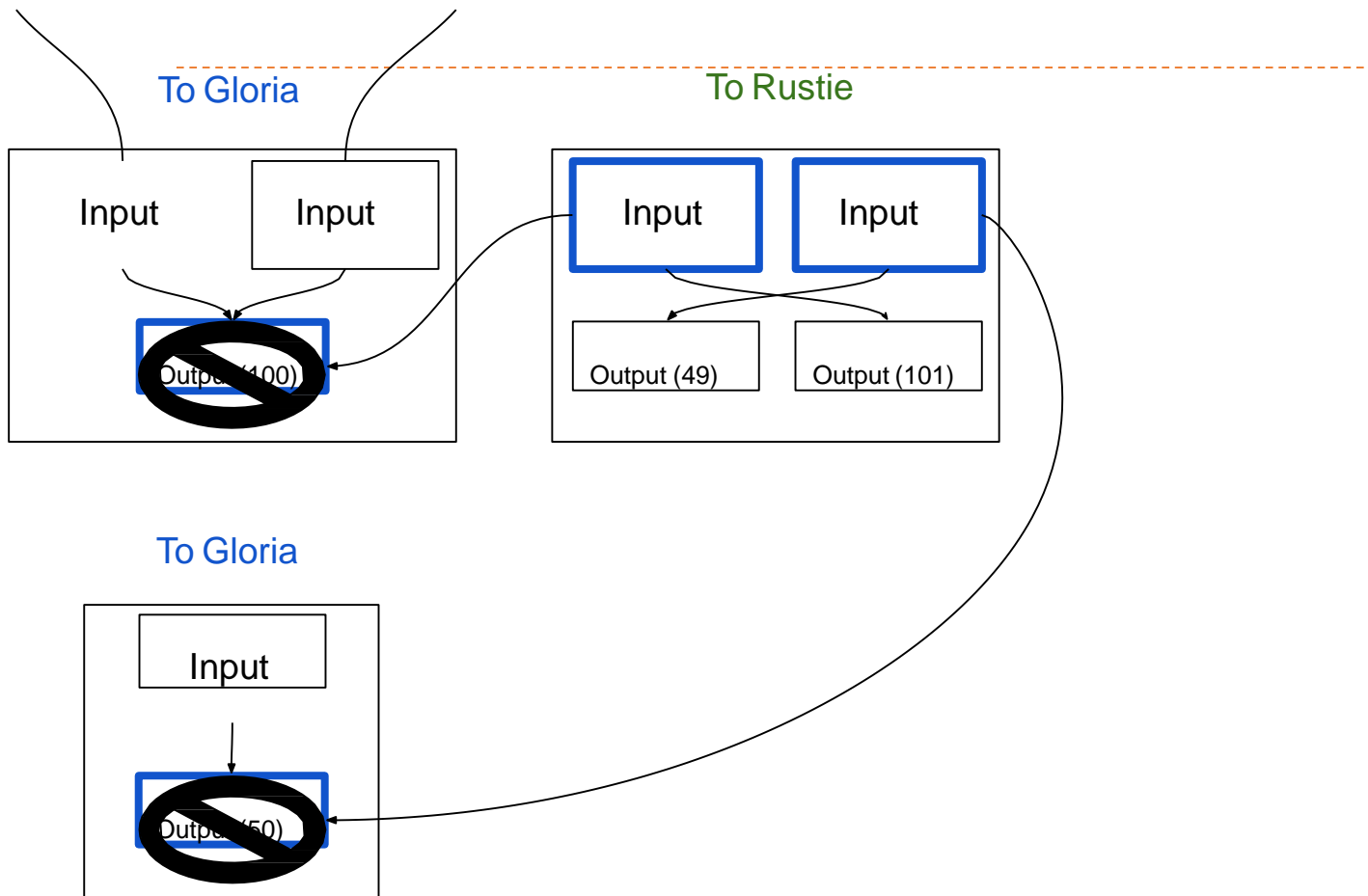


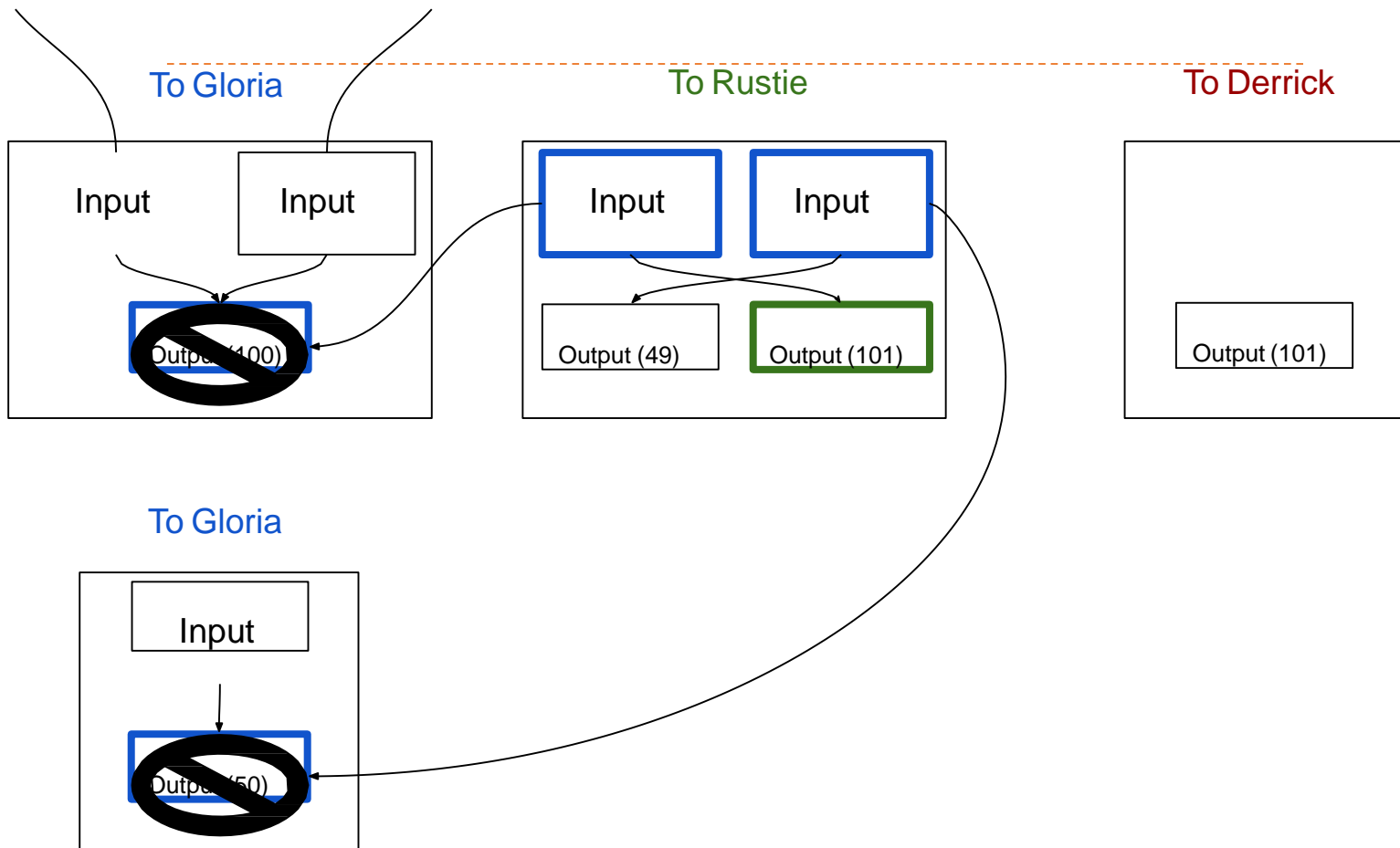


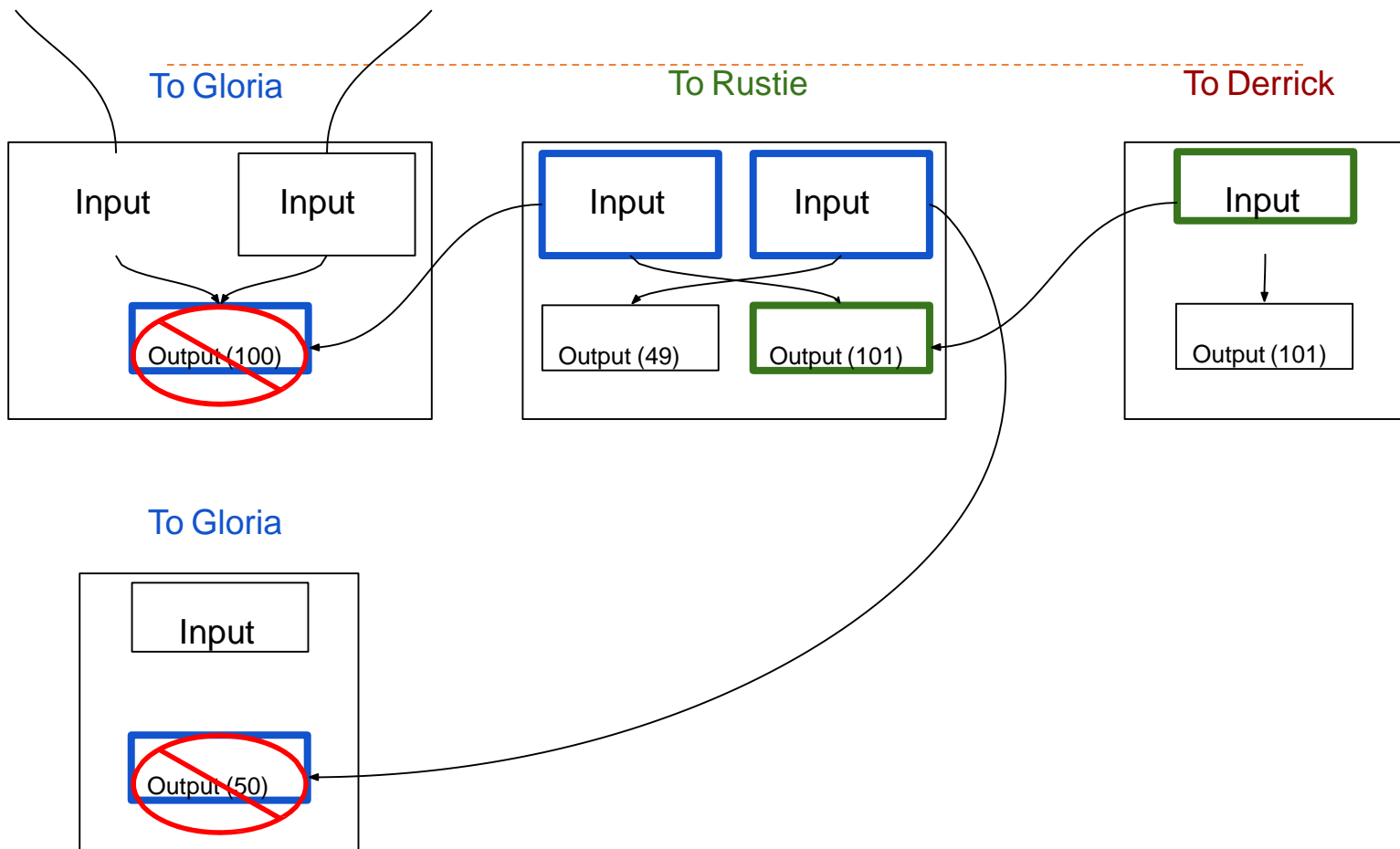










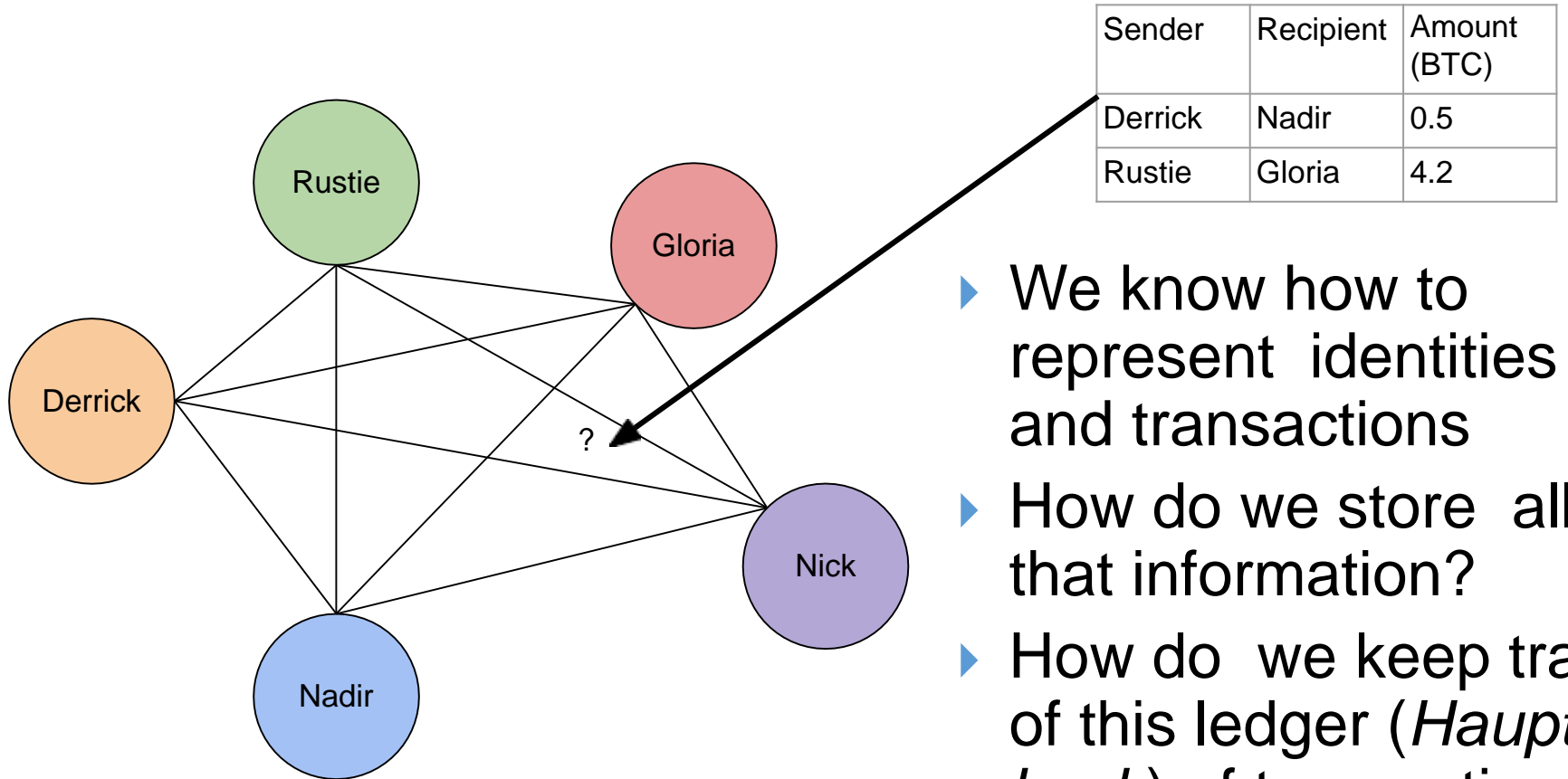


Validity of Transactions

- ▶ What makes a transaction valid?
 1. Proof of ownership (a signature)
 2. Sufficient available funds
 3. No other transactions using the same funds (no double spending)

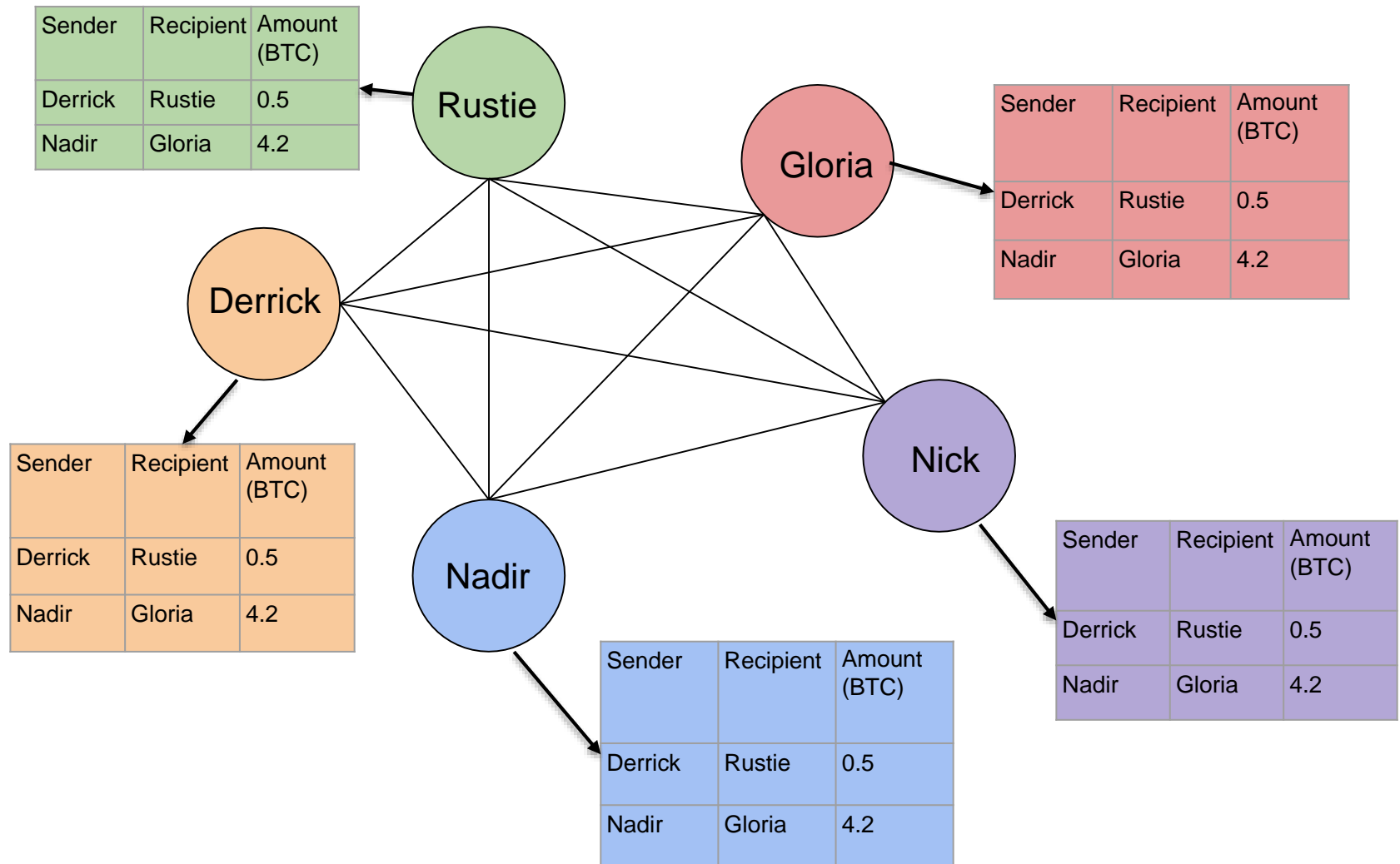
- ▶ For #2 and #3, we need a tamper-proof record of all transactions

A Distributed Database

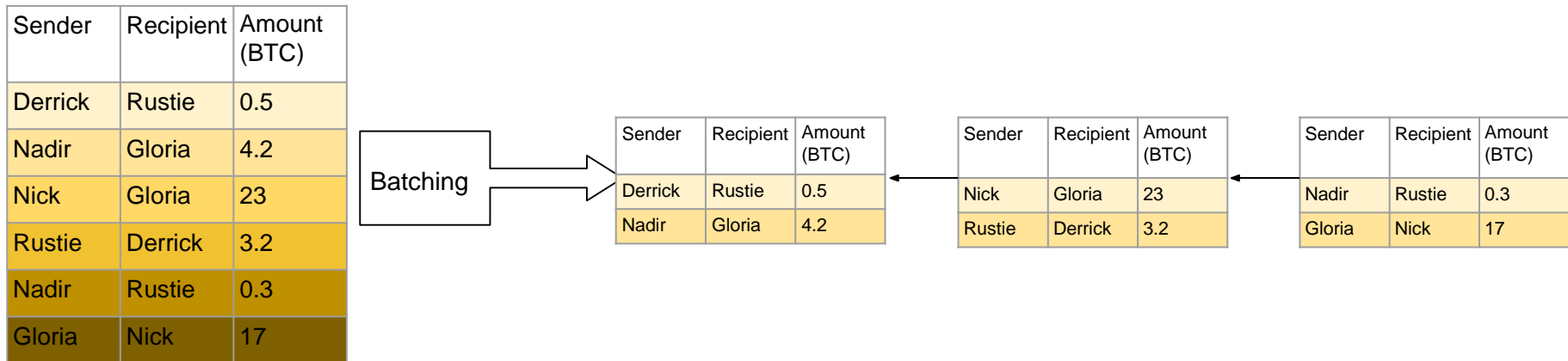


- ▶ We know how to represent identities and transactions
- ▶ How do we store all that information?
- ▶ How do we keep track of this ledger (*Hauptbuch*) of transactions?
- ▶ ⇒ With a distributed (replicated) database

Everyone Stores the Ledger

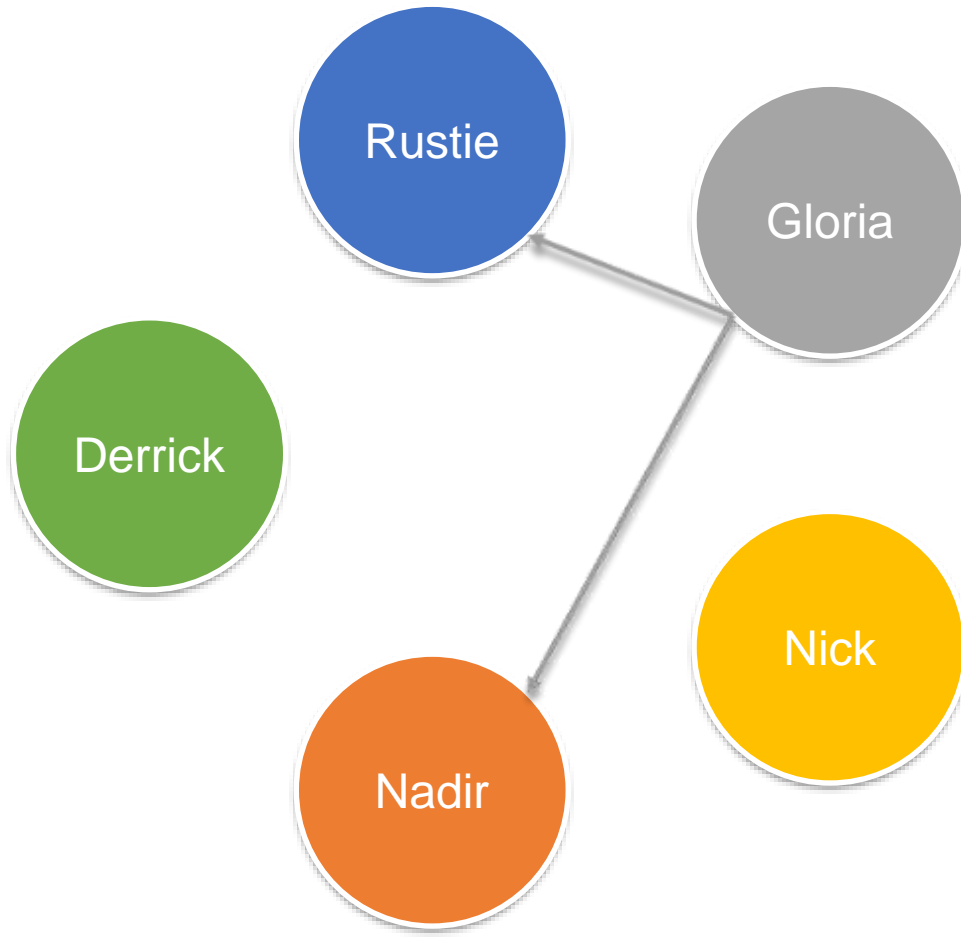


Transactions are Grouped in Blocks



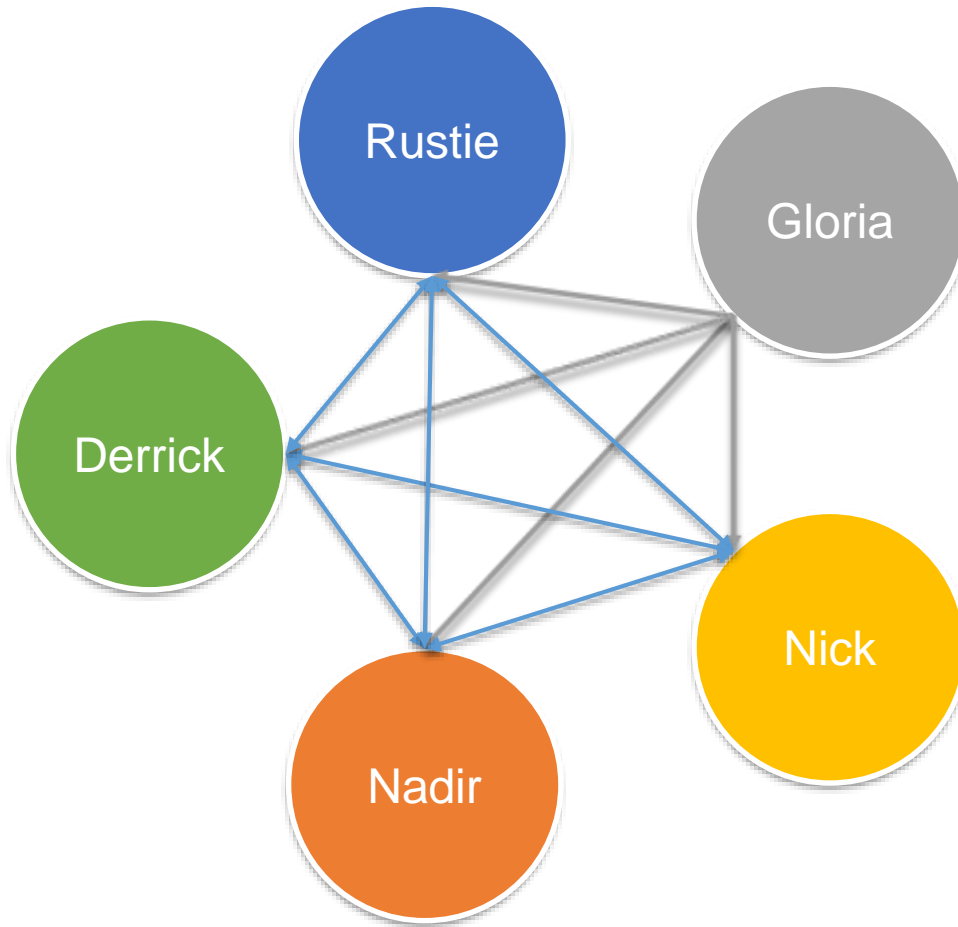
- ▶ Transactions are grouped into a list of blocks, each describing up to few 100's transactions
- ▶ Blocks are “linked” together => **blockchain**
 - ▶ “Link”: changes of a tiny block details would dramatically change fingerprint of this and of all subsequent blocks

Double-Spend Attack



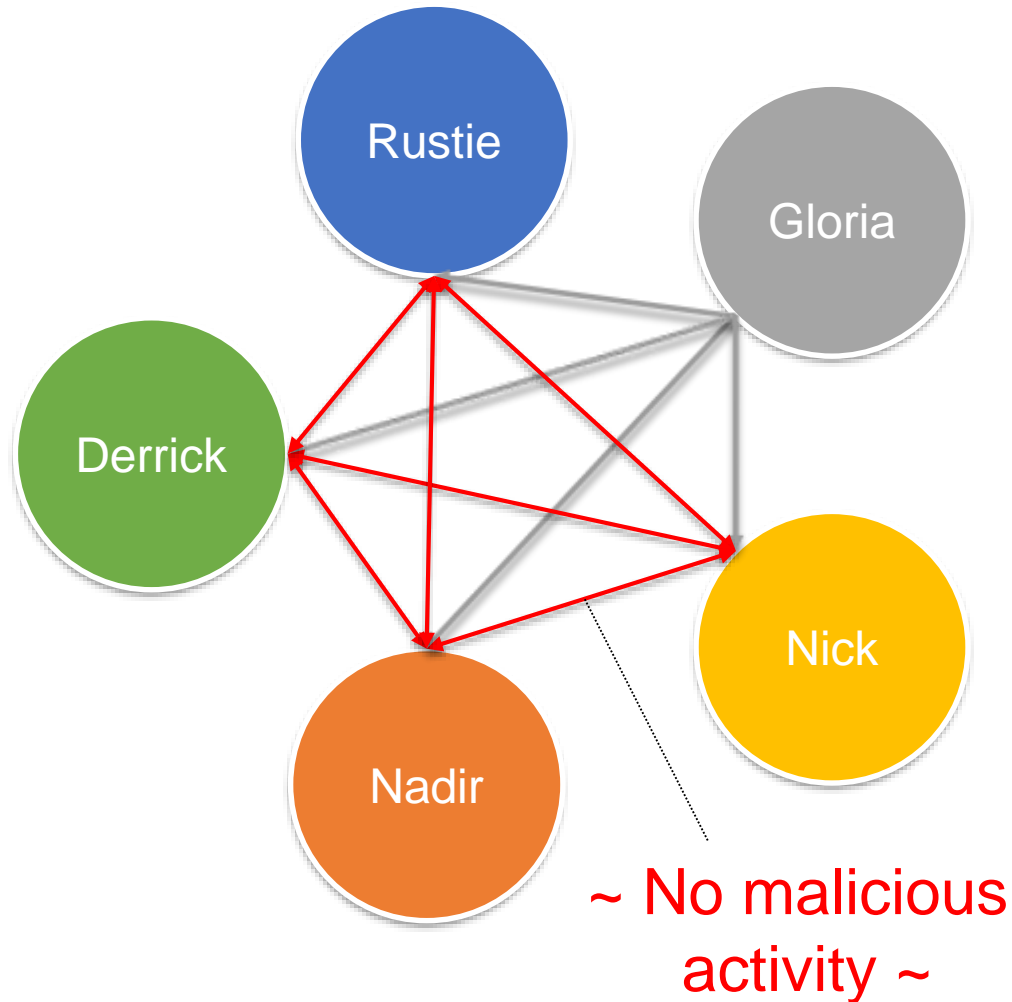
- ▶ Gloria promises 10 BTC to Rustie in one transaction, and she promises 10 BTC to Nadir in another -- but she only has 10 BTC total!
- ▶ Gloria is performing a **double spend** attack

Peer Validation



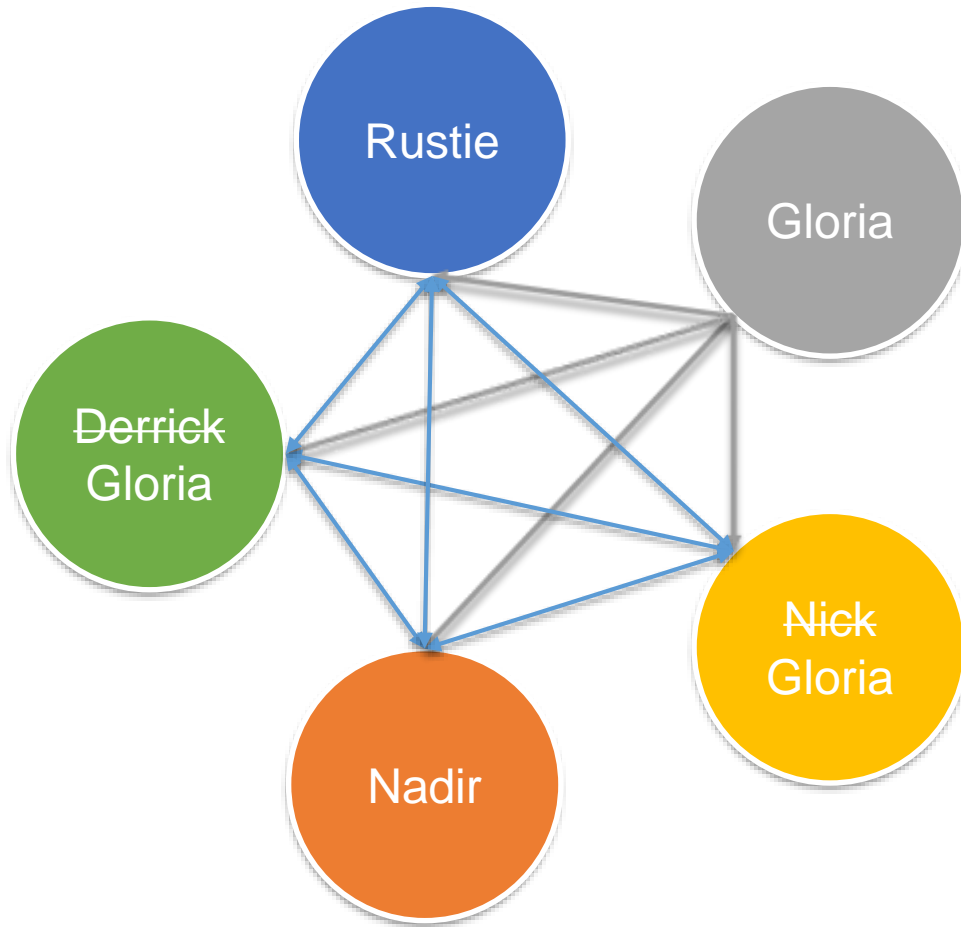
- ▶ Instead of siloed decisions, let's have **proposers** and **voters**
 - ▶ The **proposer** submits a transaction to everyone else
 - ▶ **Peers** cast votes
 - ▶ Transaction is accepted only after receiving a certain number of votes

Rejecting the Double Spend



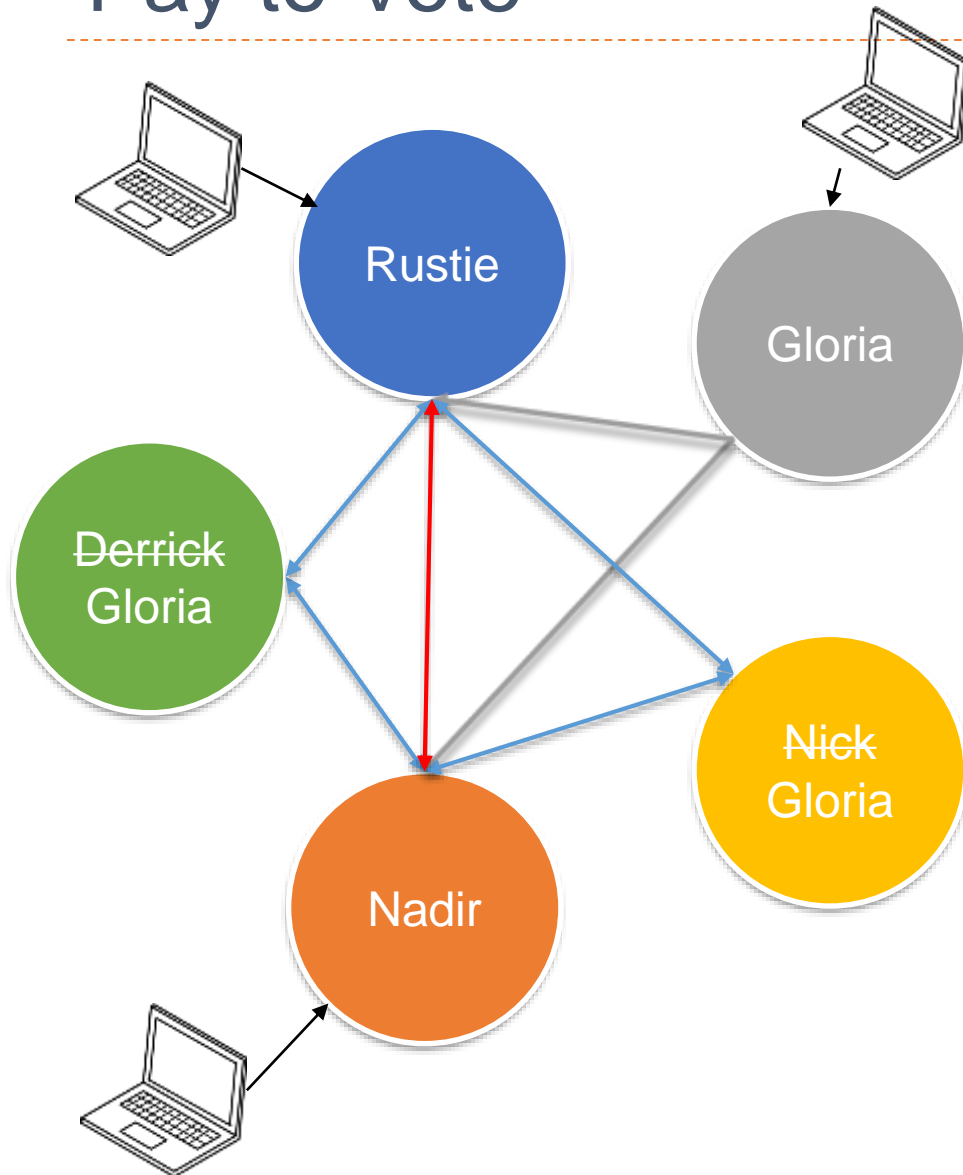
- ▶ Now, when Gloria attempts to double spend, she will be rejected by observing peers
- ▶ Peers vote “no” on Gloria’s proposal, as they notice multiple transactions trying to spend the same funds
- ▶ But there is still a problem ...

Sybil Attack



- ▶ Keep in mind, Bitcoin is an anonymous service with no central registry
- ▶ Inexpensive to create multiple identities
- ▶ Multiple identities \Rightarrow multiple opportunities to cast votes!
- ▶ Gloria can perform a **Sybil attack**, which will allow her to double spend

Pay to Vote

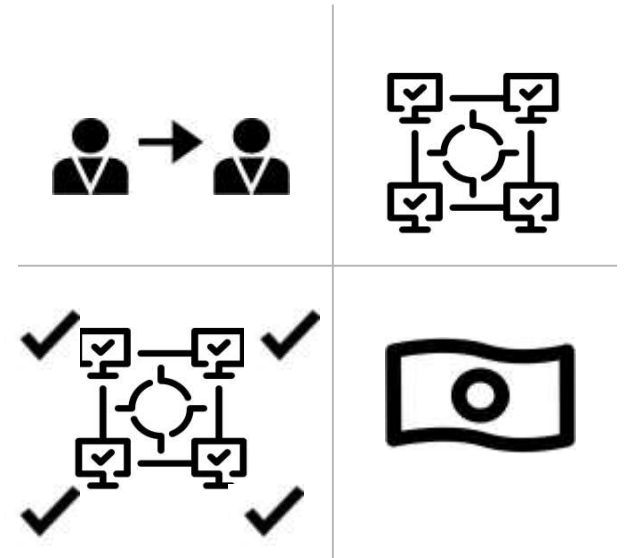


- ▶ Instead of casting votes with identities, we cast votes with resources
- ▶ To vote, one needs to solve a problem via brute forcing
 - ▶ Like guessing a pw via trial and error
- ▶ “Proof-of-work”
 - ▶ More on this later

Summary Overview Bitcoin

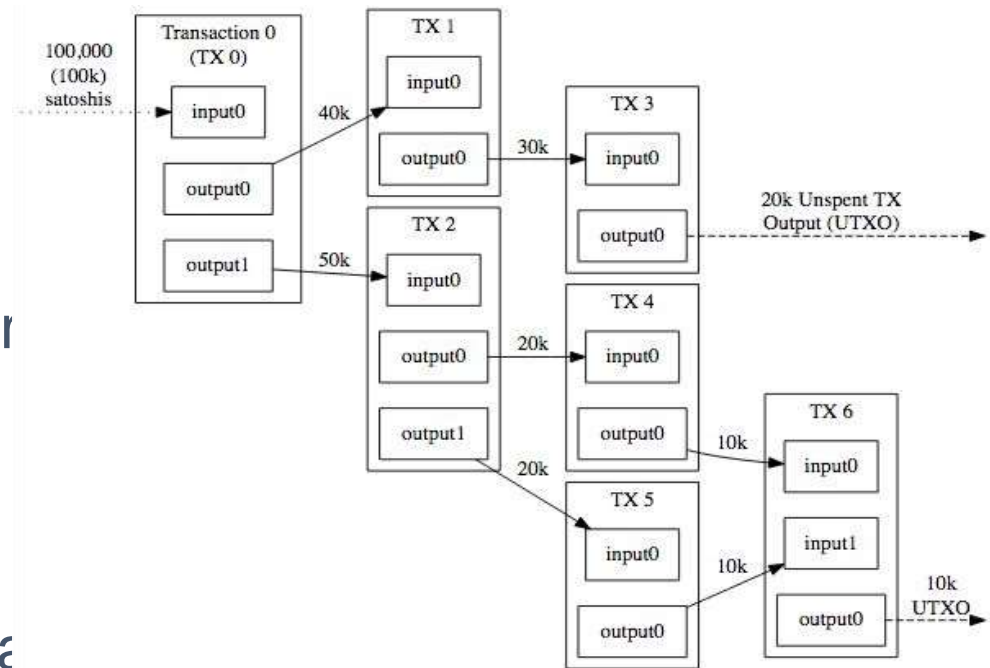
Four components of bitcoin

- ▶ A decentralized P2P network
- ▶ A public, distributed, and immutable ledger
- ▶ A mechanism for reaching global consensus on the history of the ledger
- ▶ A set of rules for currency issuance and software update



Takeaway - Transactions

- ▶ Transactions: nodes transfer ownership of UTXOs (Unspent Transaction Outputs)
 - ▶ Unique identifiers help keep record of the entire lineage of the tx's
 - ▶ One can send change back to a new address
 - ▶ A user can make several payments in parallel

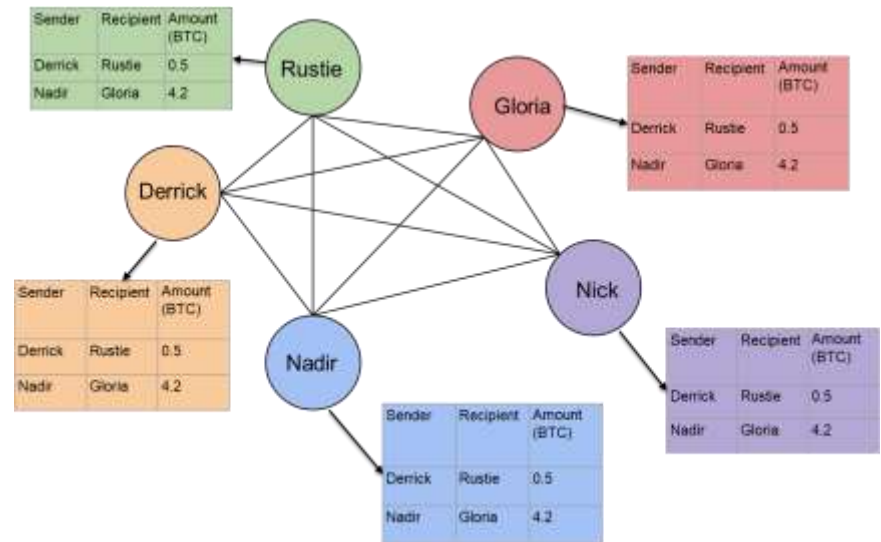


Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

Takeaway – Record-Keeping

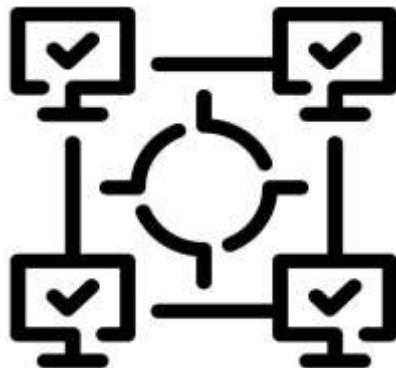
Record-keeping: mining nodes keep record of transactions by updating their own blockchain (ledger) and broadcast their version of ledger to the network

- ▶ A block is an ordered group of transactions; each block references a previous block
- ▶ Everyone maintains their own copy of the ledger based on what they hear from the network
- ▶ Updates on blockchain are irreversible



Takeaway - Consensus

- ▶ Consensus: the network agrees on a single version of history (or transaction record/blockchain) through **proof-of-work**
 - ▶ Proof-of-work makes participation in the consensus process expensive, preventing malicious entities from tampering with transaction history
 - ▶ Allows the network to reach consensus without relying on a central authority



Unique Properties of Bitcoin: Buzzwords

- ▶ **Pseudonymous**: users use pseudonyms, not real world identities to make transactions
- ▶ **Decentralized**: every user possesses the same copy of transaction history
- ▶ **Immutable**: it is close to impossible for any users to change the network transaction history in their favor
- ▶ **Trustless**: users don't need to trust anyone they are transacting with to be sure that their transactions will be accurately recorded by the rest of the network

Thank you.