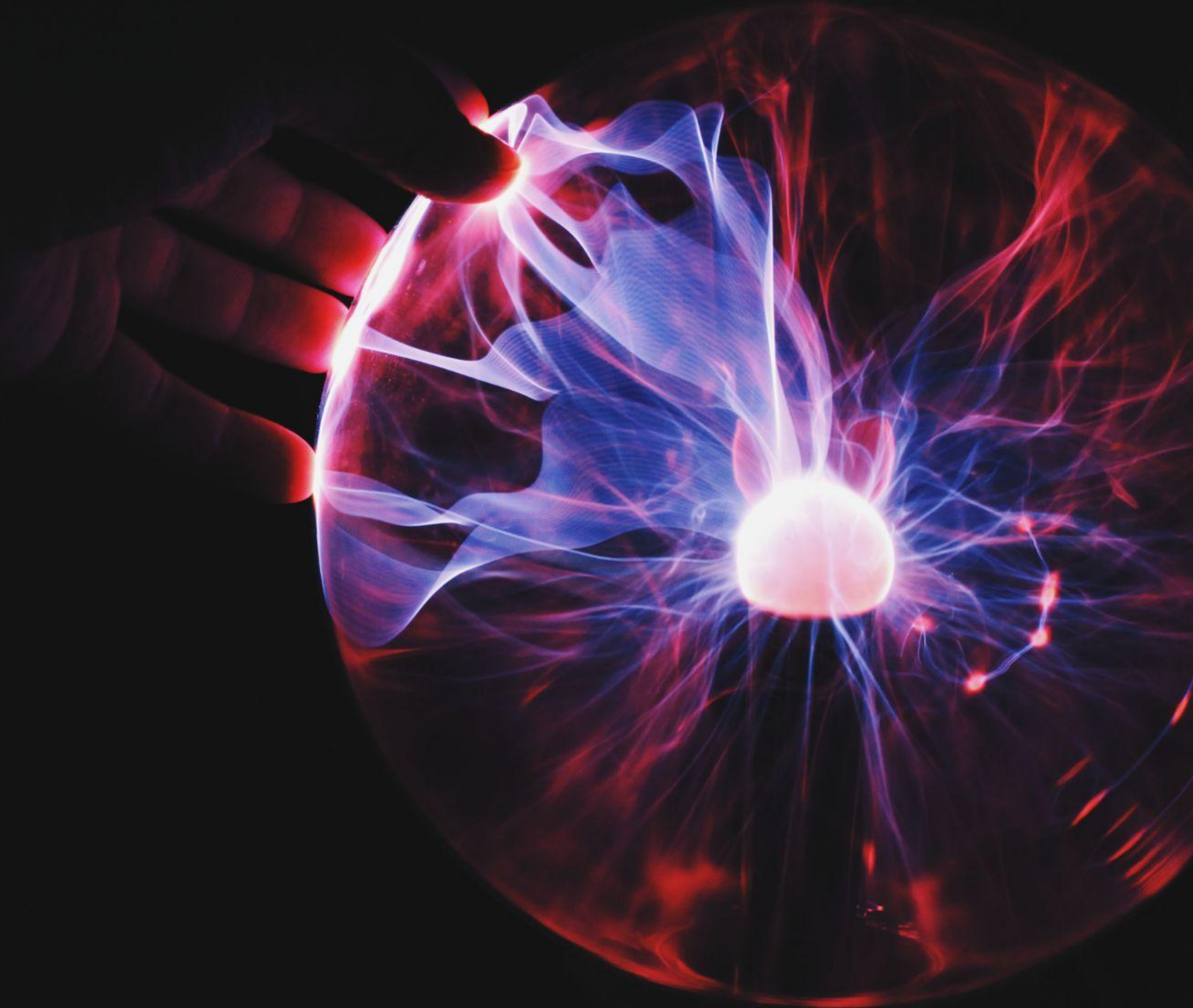
INPUT | OUTPUT

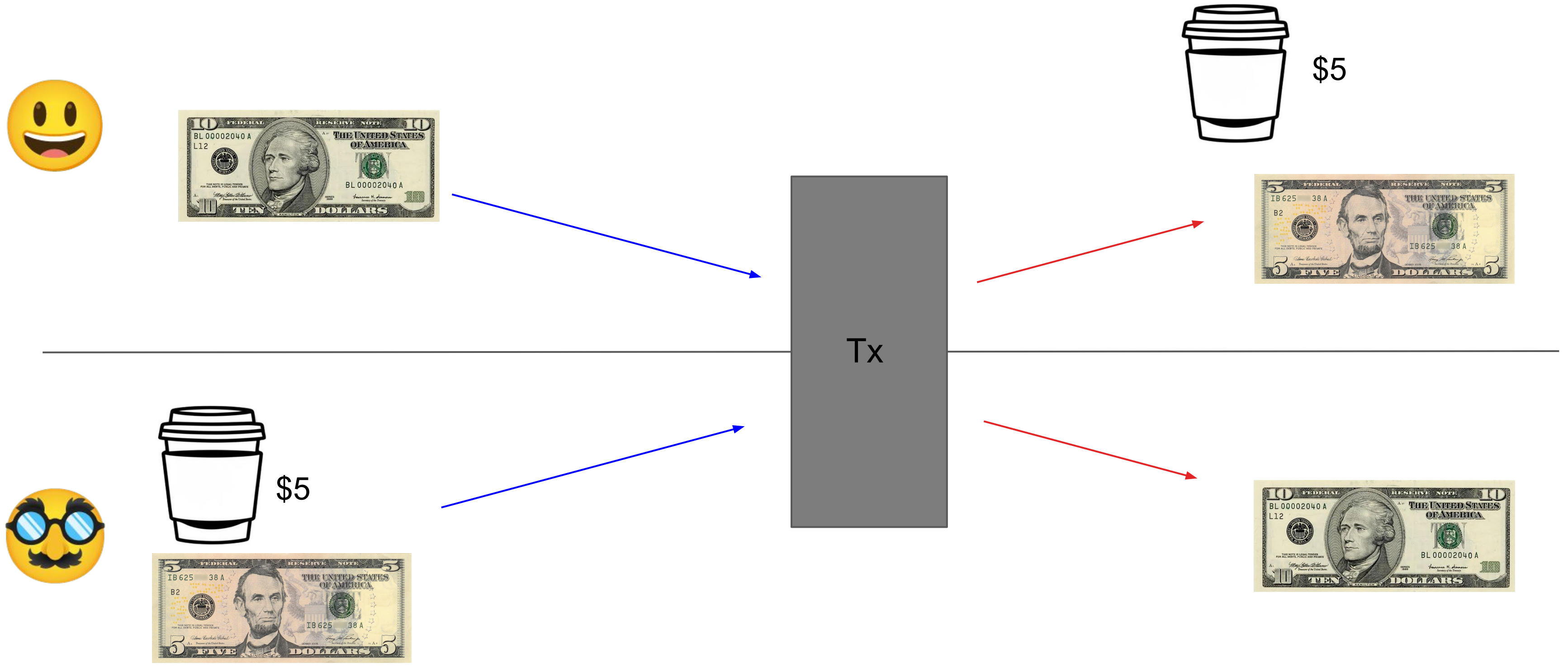


UTxO & (E)UTxO

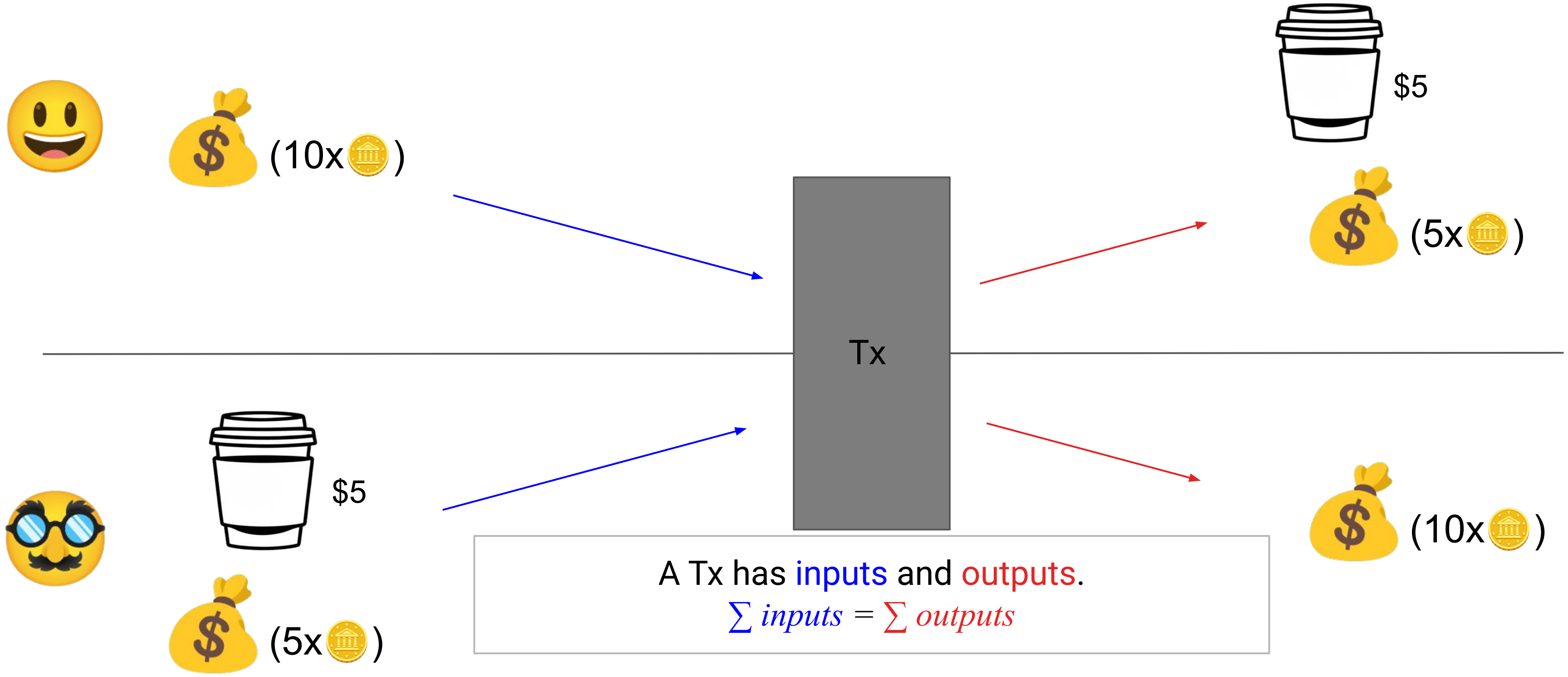
+ UTxO MODEL _

Introduction to the UTxO Model

Cash Transactions



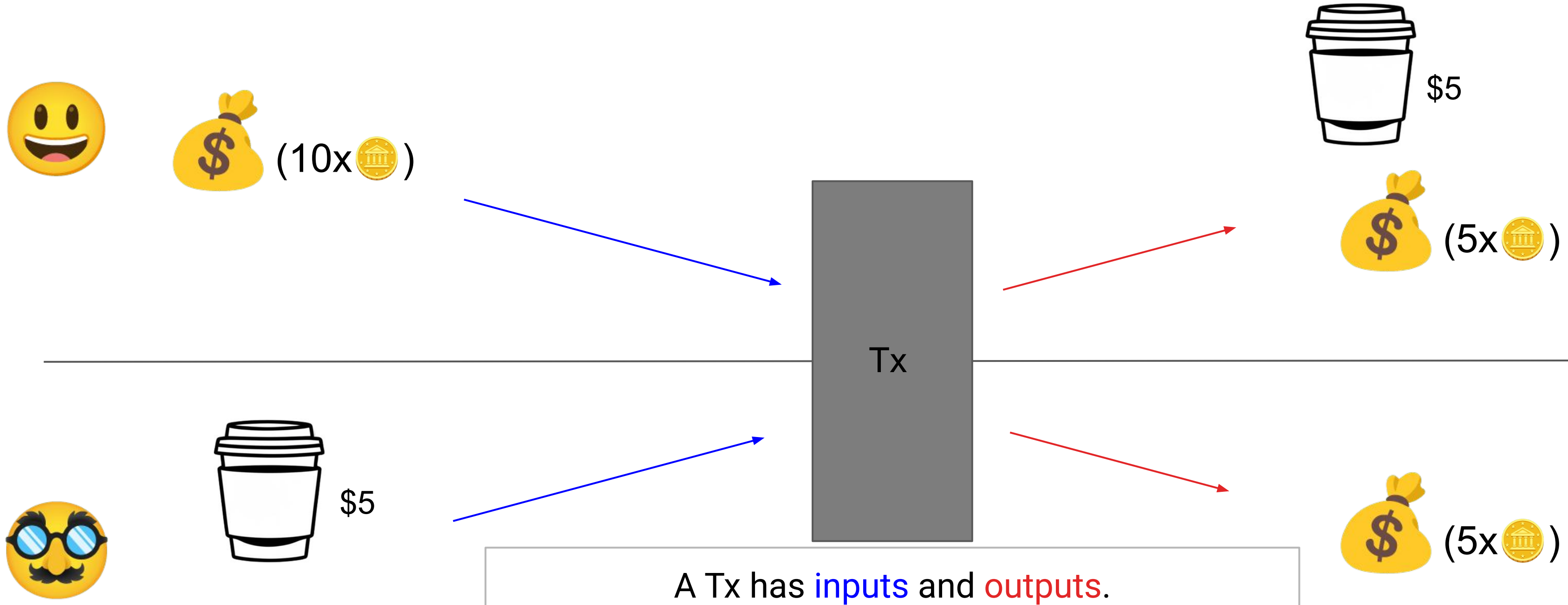
UTxO Transactions



A Tx has inputs and outputs.

$\sum inputs = \sum outputs$

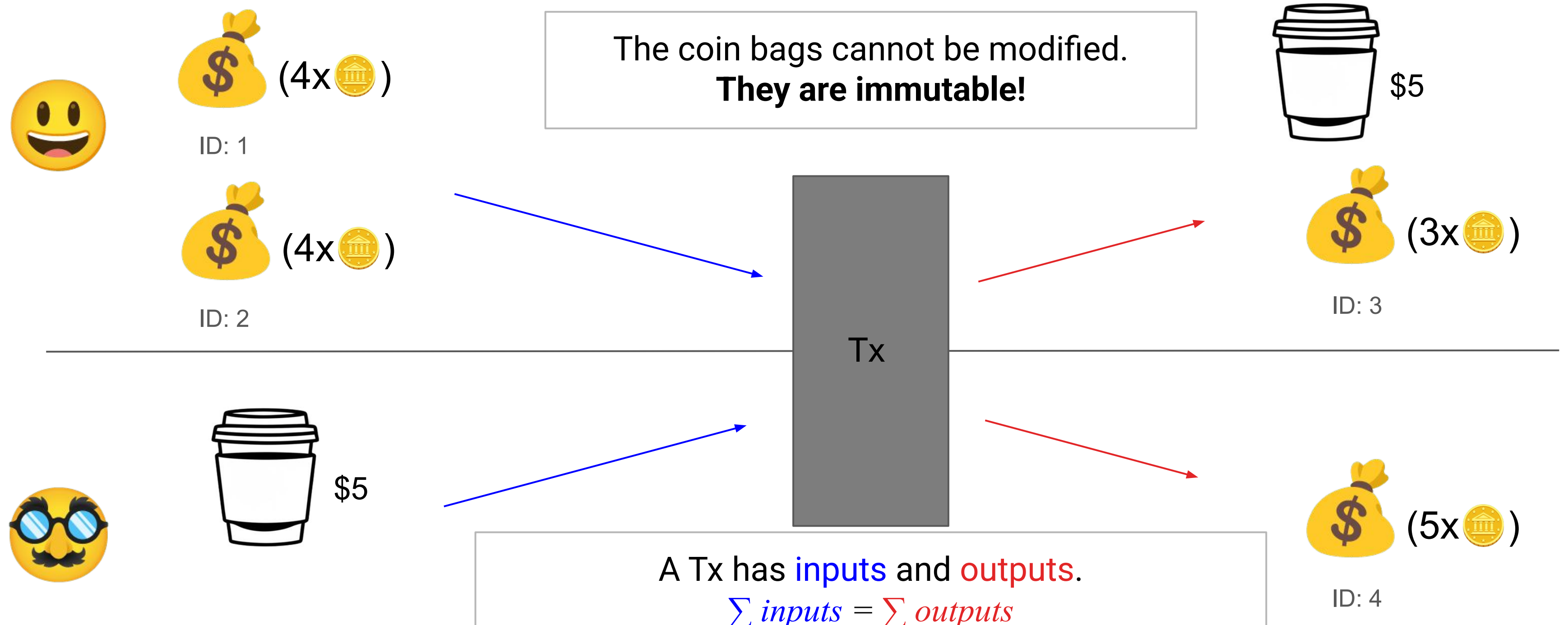
UTxO Transactions



A Tx has inputs and outputs.
$\sum inputs = \sum outputs$

Coin bags are destroyed and new ones are created.

UTxO Transactions

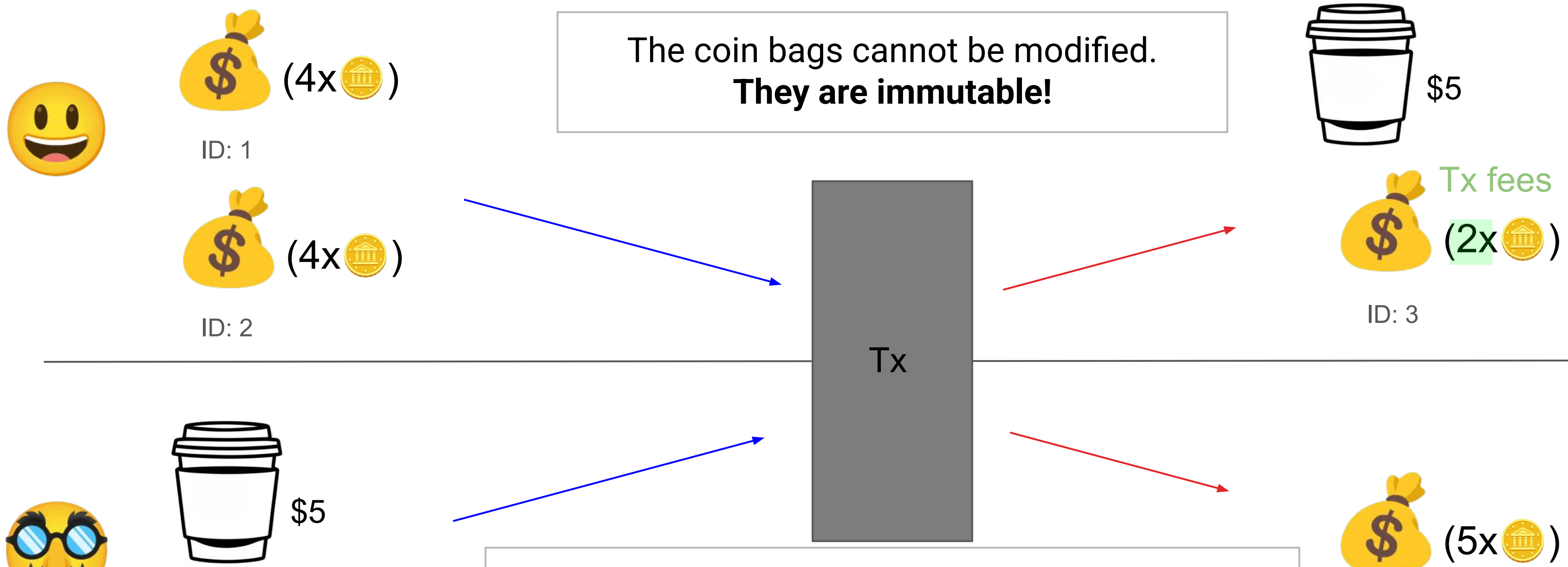


The coin bags cannot be modified.
They are immutable!

A Tx has inputs and outputs.
$\sum inputs = \sum outputs$

Coin bags are destroyed and new ones are created.

UTxO Transactions



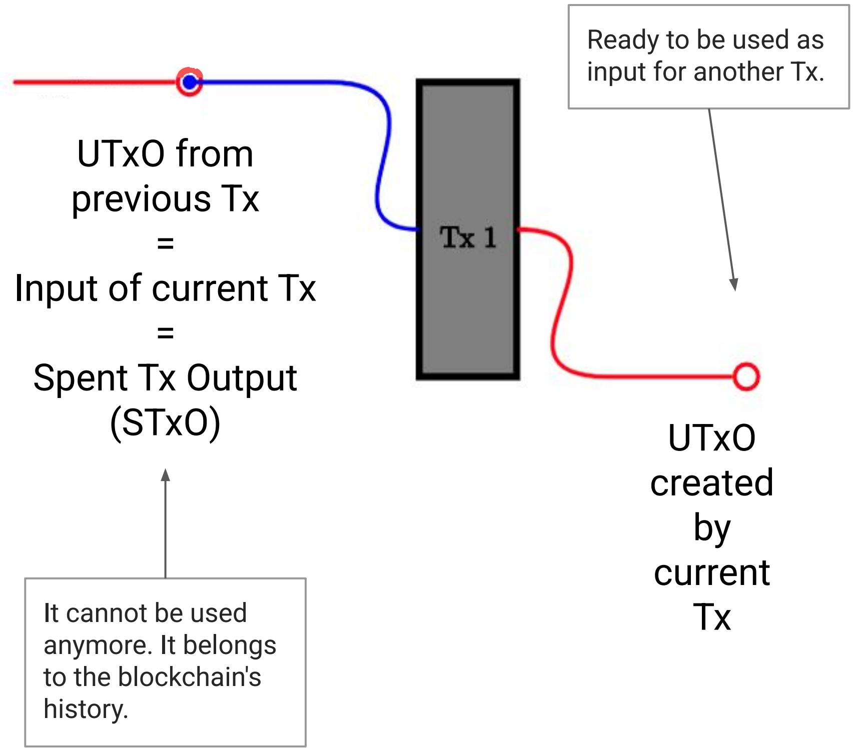
The coin bags cannot be modified.
They are immutable!

A Tx has inputs and outputs.

$$\sum inputs = \sum outputs + Fees$$

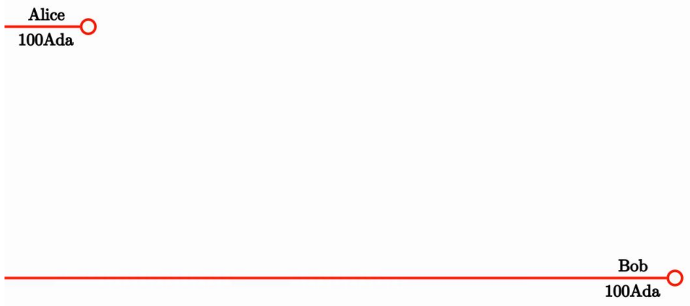
Coin bags are destroyed and new ones are created.

UTxO Model

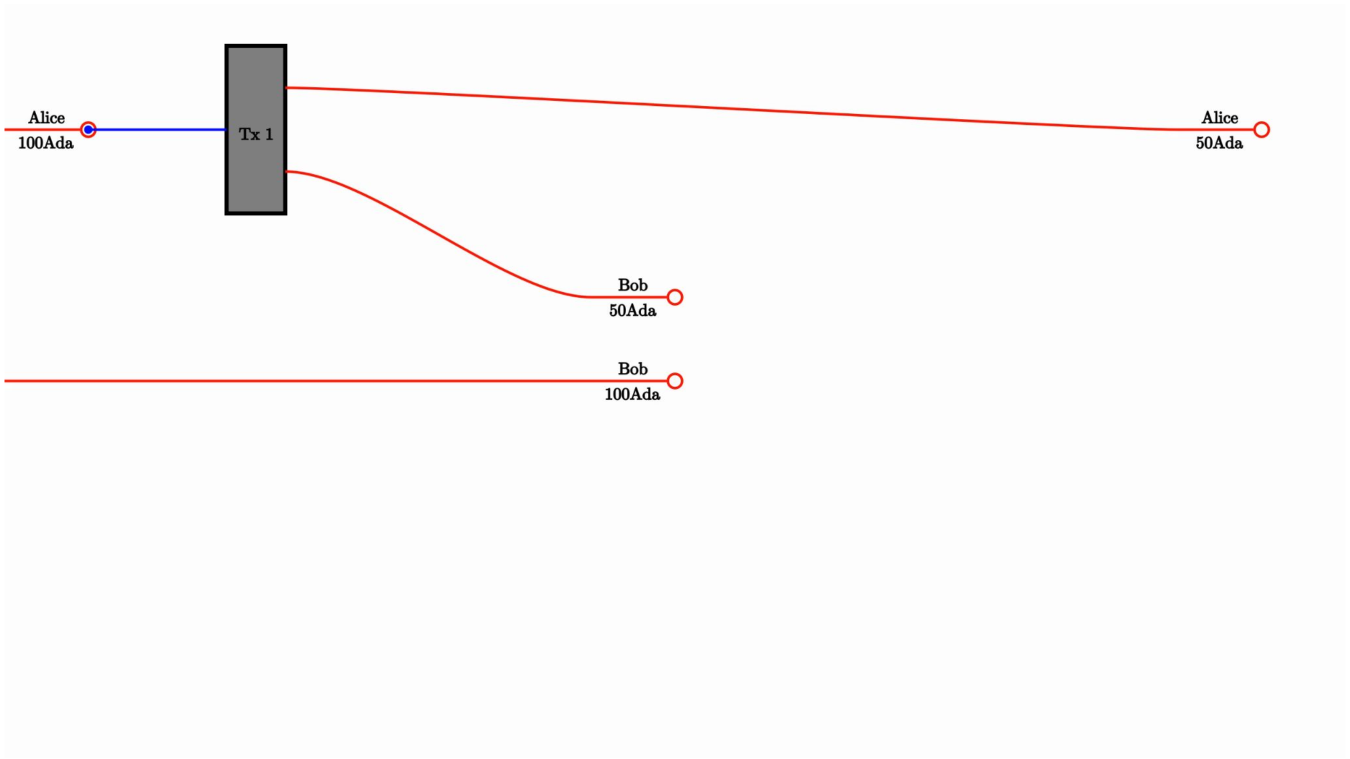

Ready to be used as input for another Tx.
Tx 1
UTxO from previous Tx = Input of current Tx = Spent Tx Output (STxO)
UTxO created by current Tx
It cannot be used anymore. It belongs to the blockchain's history.

- Transactions can have an **arbitrary number of inputs and outputs.** (Minimum 1 input and 1 output).
- An **Unspent Transaction Output (UTxO)** is the output of a transaction that has not yet been consumed by other transactions.
- If you want to use part of the value locked in a UTxO, it must be **consumed completely.**
- The only thing that happens on the blockchain are these transactions. UTxOs are consumed and created, but **THEY ARE NOT MODIFIED!**

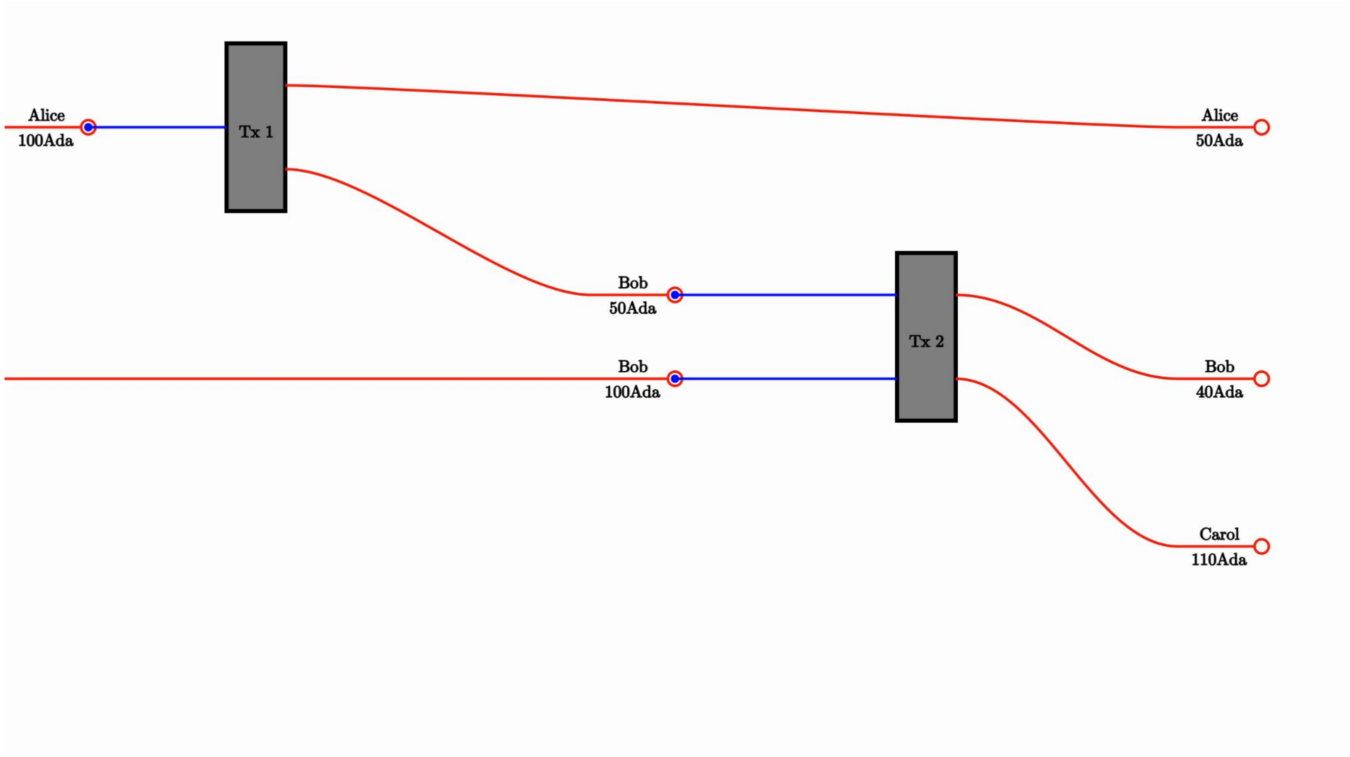
Transactions in UTxO: ✍️ Alice sends 50 ADA to Bob?



Transactions in UTxO: ✍️ Bob sends 110 ADA to Carol?



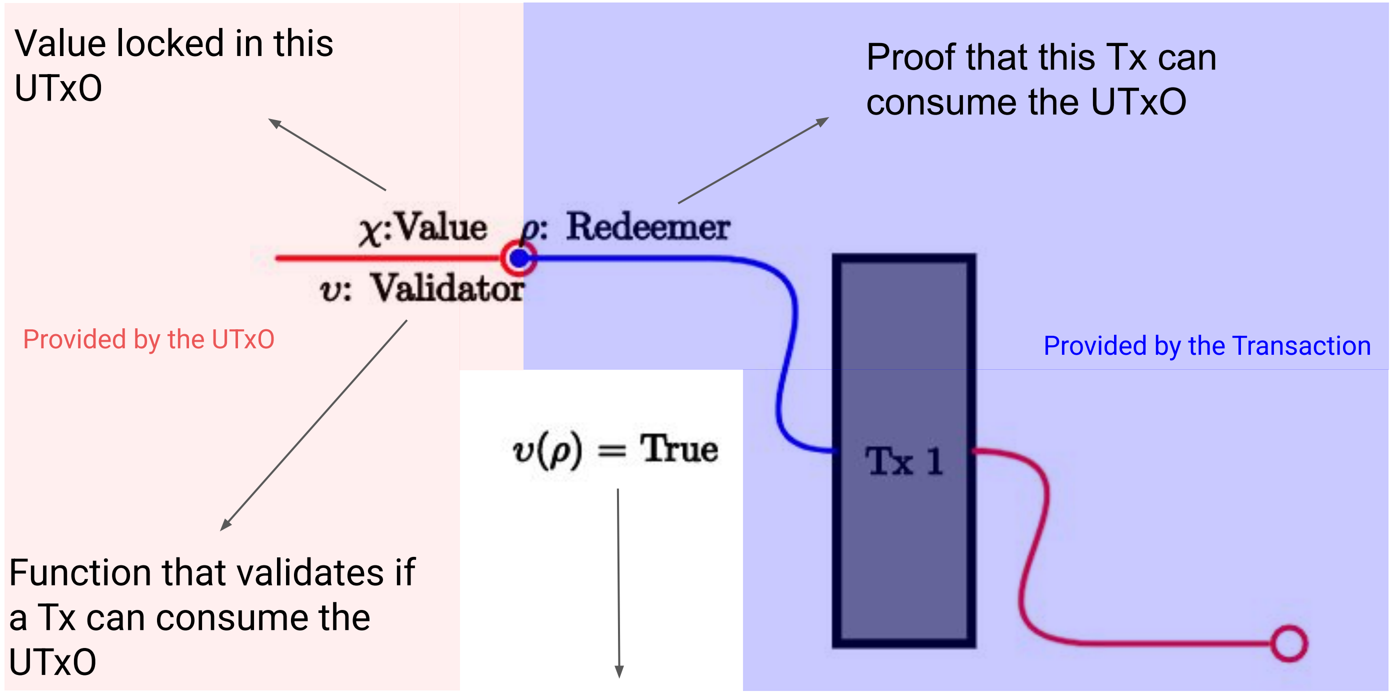
Transactions in UTxO



Transactions in UTxO: ✍️ Make a Tx with Lace Wallet

1. Install the lace.io Extension.
2. Create a new wallet.
3. Configure to use Preview/Preprod Testnets.
4. Request funds from the Testnet faucet.
5. Explore the UTxO of the Tx in the blockchain explorer.
6. Create a new Tx sending X ADA to another student.
7. Explore the UTxO of the Tx in the blockchain explorer.

Transactions in UTxO: How do I ensure that my UTxOs are not stolen?


Value locked in this UTxO
Proof that this Tx can consume the UTxO
χ:Value
ρ: Redeemer
υ: Validator
Provided by the UTxO
Provided by the Transaction
$\upsilon(\rho) = \text{True}$
Tx 1
Function that validates if a Tx can consume the UTxO

The Validator is applied to the Redeemer and returns `True`/`False` if the Tx can/cannot consume the UTxO.

Transactions in UTxO: Limitations of the UTxO Model

The UTxO Model works perfectly for **simple transfers**. However, it **does not have enough information** to perform more complex transfers. For example:

- Allow consuming the UTxO only if part of its value is sent to a specific person.
- Allow consuming the UTxO only if another specific UTxO is also consumed.
- Allow consuming the UTxO only if there is another UTxO with a specific value/information.
- …

But these are the things **required** to create **decentralized programs** that enable performing the **same actions** that are currently done in a **centralized way**:

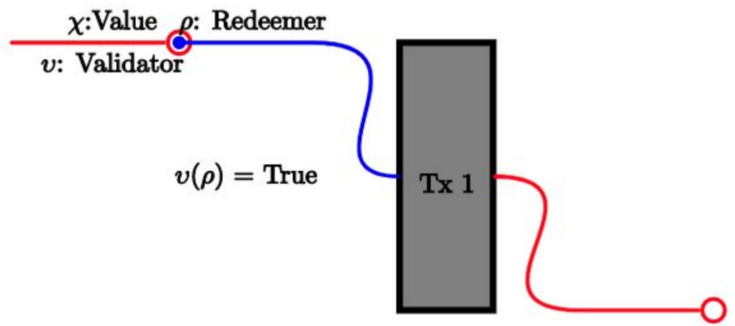
- Loans
- Financing
- Investments
- Vesting schedules
- …

+ (E)UTxO MODEL _

Introduction to the (E)UTxO Model

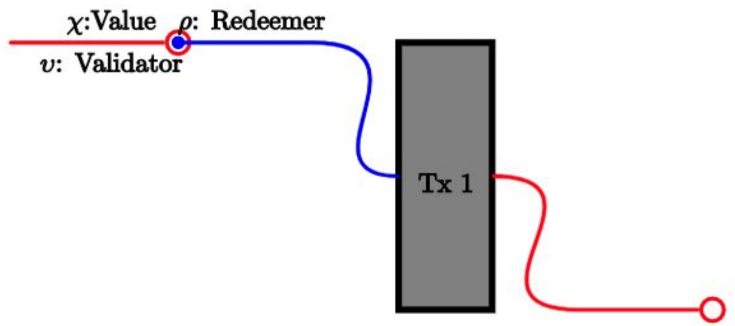
(E)UTxO Model: Comparison of the UTxO model with (E)UTxO

UTxO



Extremely limited expressiveness

Extended UTxO



Sufficient expressiveness to replace much of the centralized financial/banking services

(E)UTxO Model: Vesting Example - Idea

- Many companies like Facebook, Google, Apple, and Netflix provide “**Restricted Stock Units**” (RSUs) to employees as part of compensation.

- Restricted Stock Units usually have a vesting schedule, such as **25% of the shares per year over 4 years.**

- For various reasons, this is done to ensure that the employee:
    - Does not sell 100% of the shares at once (potentially impacting the market price)
    - Does not sell them and quit the next day.

(E)UTxO Model: Vesting Example - Design

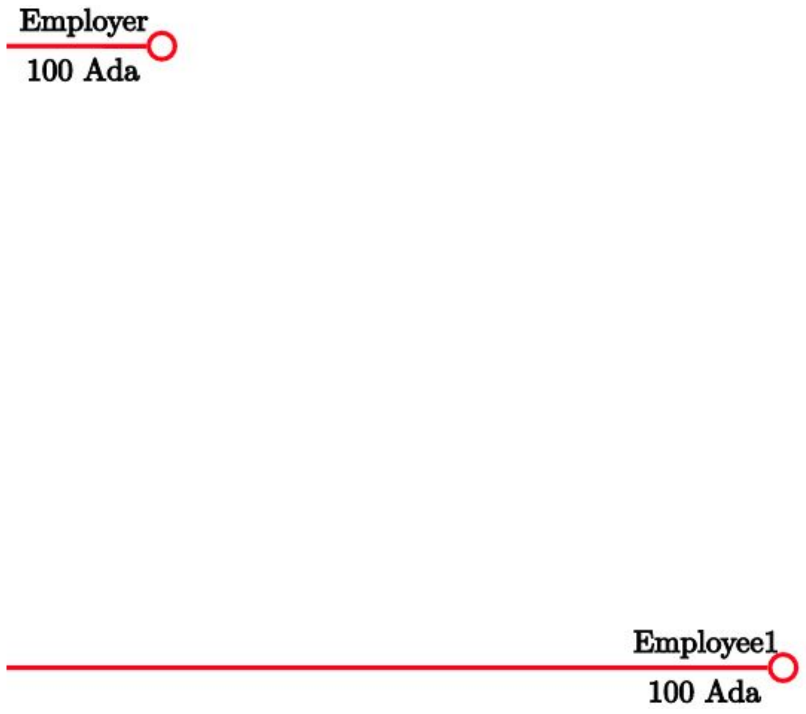
1. When the employee accepts the job offer, the employer will create **4 UTxOs, each containing 25% of the shares.**

2. Each of the 4 UTxOs **check** that the one attempting to consume the UTxO **is the employee** and that a **deadline has passed**.

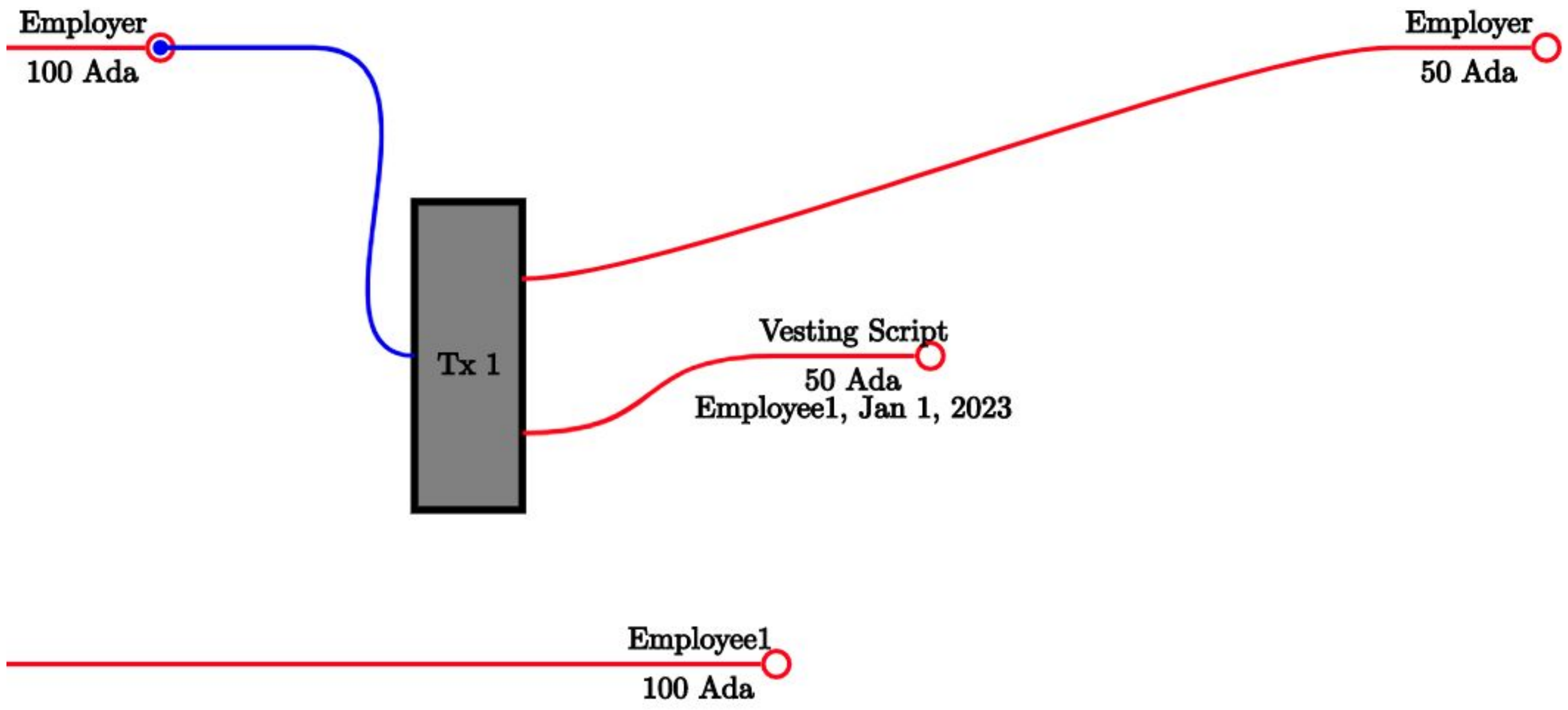
3. Each UTxO has a **different deadline** (e.g., 1/12/2025, 1/12/2026, 1/12/2027, 1/12/2028).

4. The beneficiary just needs to **wait** for each deadline to pass before they **can consume the UTxO** and claim their shares.

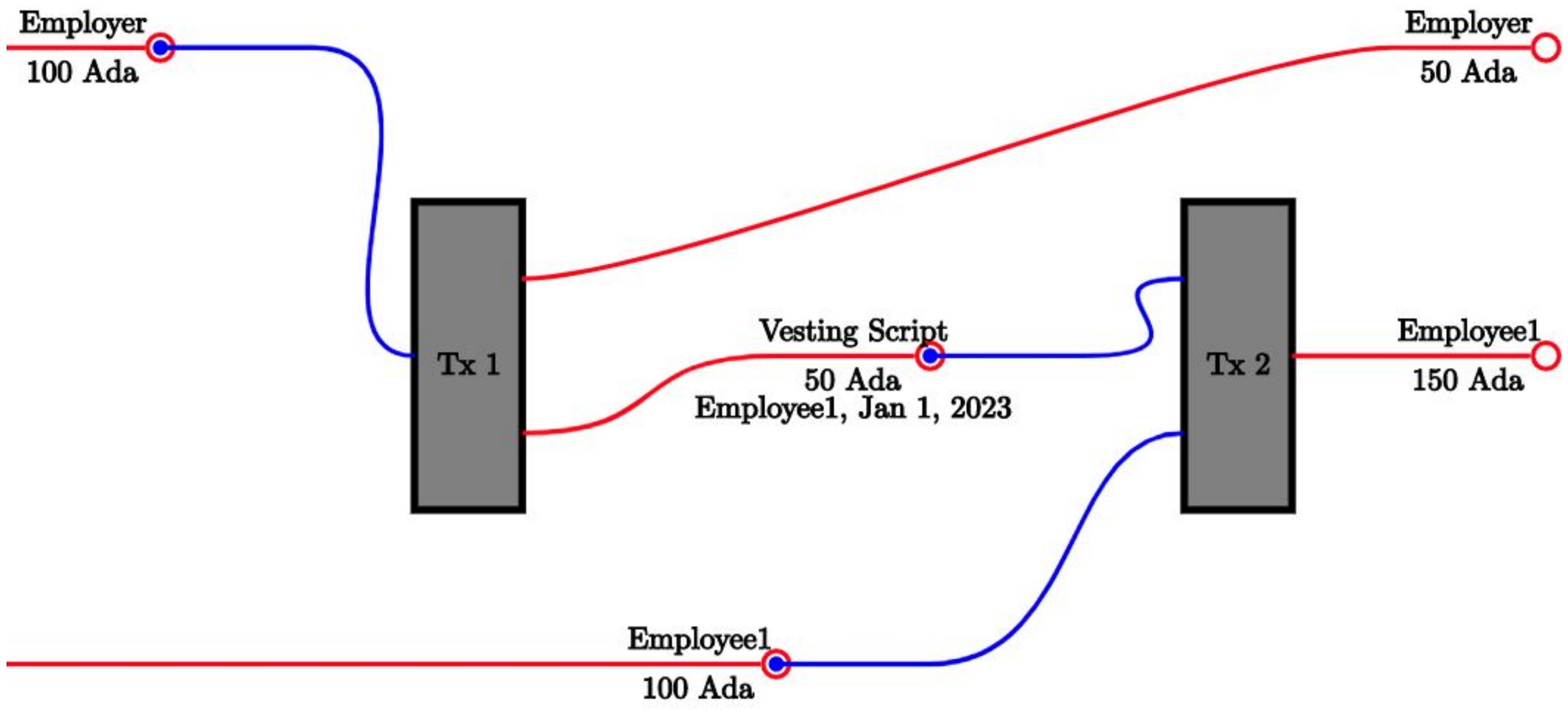
(E)UTxO Model: Vesting Example - Visualization



(E)UTxO Model: Vesting Example - Visualization



(E)UTxO Model: Vesting Example - Visualization



(E)UTxO Model: Vesting Example - Validator

```
type PKH = Hash<Blake2b_224, VerificationKey>
type VestingDatum { VestingDatum { beneficiary: PKH, deadline: Int } }

validator vesting {
  spend(datum: Option<VestingDatum>, _r, _utxo, tx: Transaction) {
    expect Some(vd) = datum
    let Transaction { extra_signatories, validity_range, .. } = tx
    and {
      list.has(extra_signatories, vd.beneficiary)?,
      when validity_range.lower_bound.bound_type is {
        Finite(tx_earliest_time) -> vd.deadline <= tx_earliest_time
        _ -> False
      }?,
    }
  }
}
```

(E)UTxO Model: Item Details - Value

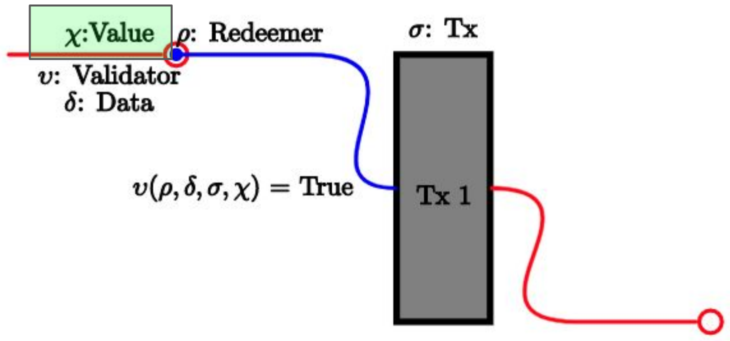
What can it be?:

- Lovelace (1 ADA = 1.000.000 Lovelace) ✅
- Lovelace + Tokens (FT and/or NFT) ✅
- Tokens without Lovelace 🚫

Things to keep in mind:

- minUTxO (the minimum amount of Lovelace required for the UTxO) depends on the size of the UTxO.

- It's possible to hold many tokens at the same time, and you can hold millions of a single token without it affecting the cost or size.



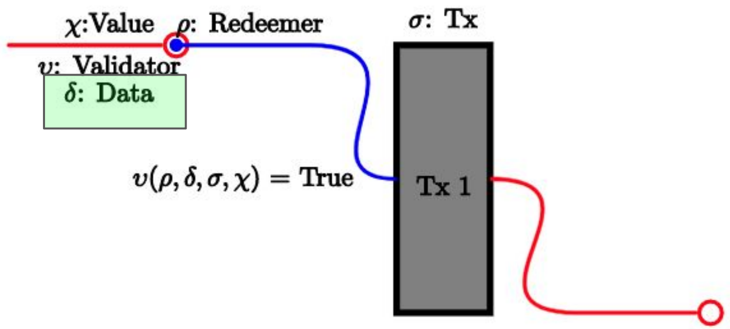
(E)UTxO Model: Item Details - Datum

What can it be?:

- Anything (a number, text, a list of PKH, a custom structure, etc.)

Things to keep in mind:

- The **creator of the UTxO** must choose a Datum when creating the UTxO, but they are not *required* to make it public.



Creator Provides		Consumer Provides
In UTxO	**In Transaction**	
Hash of the Datum	-	Datum
Hash of the Datum	Datum	Datum
Datum (inline)	-	-

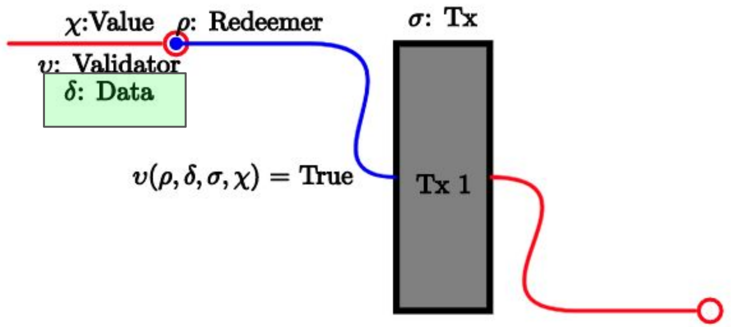
(E)UTxO Model: Item Details - Datum

What can it be?:

- Anything (a number, text, a list of PKH, a custom structure, etc.)

Things to keep in mind:

- The **creator of the UTxO** must choose a Datum when creating the UTxO, but they are not *required* to make it public.



Creator Provides		Consumer Provides
In UTxO	**In Transaction**	
Hash of the Datum	-	Datum
Hash of the Datum	Datum	Datum
Datum (inline)	-	-

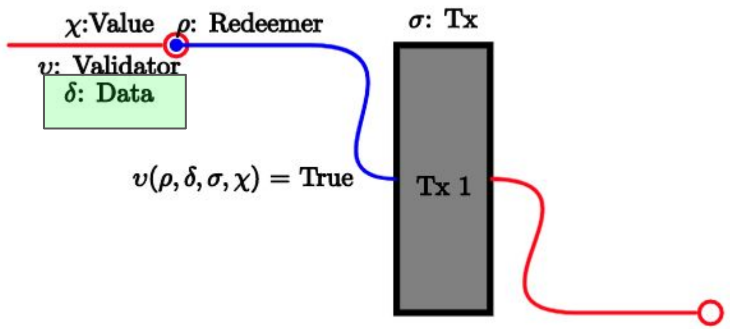
(E)UTxO Model: Item Details - Datum

What can it be?:

- Anything (a number, text, a list of PKH, a custom structure, etc.)

Things to keep in mind:

- The **creator of the UTxO** must choose a Datum when creating the UTxO, but they are not *required* to make it public.



Creator Provides		Consumer Provides
In UTxO	**In Transaction**	
Hash of the Datum	-	Datum
Hash of the Datum	Datum	Datum
Datum (inline)	-	-

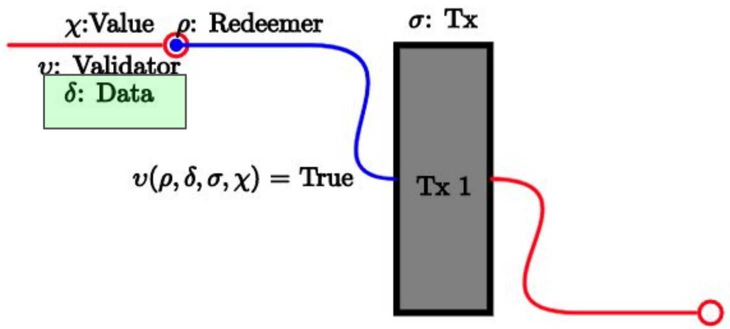
(E)UTxO Model: Item Details - Datum

What can it be?:

- Anything (a number, text, a list of PKH, a custom structure, etc.)

Things to keep in mind:

- The **creator of the UTxO** must choose a Datum when creating the UTxO, but they are not *required* to make it public.



Creator Provides		Consumer Provides
In UTxO	**In Transaction**	
Hash of the Datum	-	Datum
Hash of the Datum	Datum	Datum
Datum (inline)	-	-

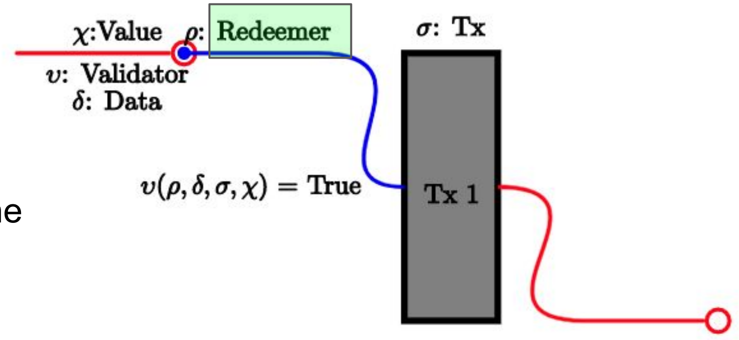
(E)UTxO Model: Item Details - Redeemer

What can it be?:

- Anything (a number, text, a list of PKH, a custom structure, etc.)

What's the difference with the Datum?

- The Redeemer is chosen by the one **consuming** the UTxO.



Common Datums and Redeemers (in any combination):

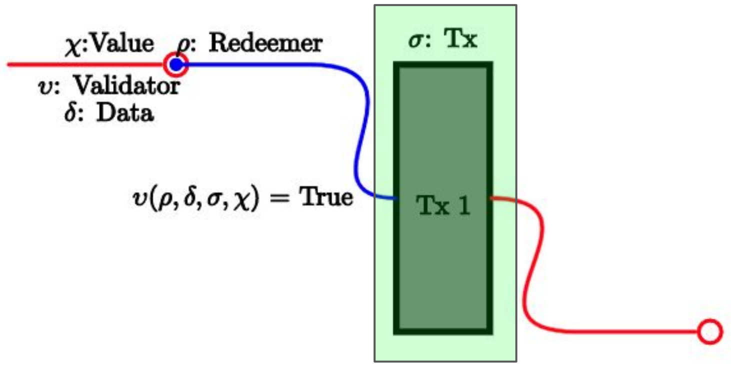
Datum
Who/When/With what can I consume a UTxO?
Current state of the UTxO
Metadata/Configurations

Redeemer
Reason for use: request a loan, pay a fee, cancel a loan.
Information known only by X person
Value to replace the Datum

(E)UTxO Model: Item Details - Transaction Context

Things to keep in mind:

- It can only be a value of a specific shape (ScriptContext)



What does the ScriptContext value contain?:

- Type of Tx
- Validity time range
- Inputs (with their respective values, Datums, and Redeemers)
- Outputs (with their respective values and Datums)
- References to UTxOs and relevant validators that are not part of the Tx
- Tx cost
- Those who signed the Tx
- Identifier and amount of Tokens being minted/burned in the Tx
- …

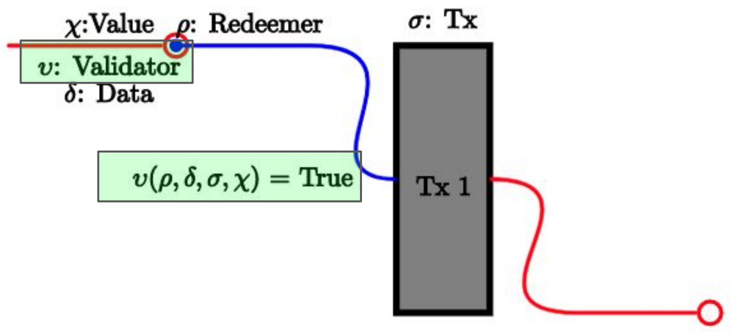
(E)UTxO Model: Item Details - Validator/Script

- It takes into account all the elements we've just discussed to decide whether the UTxO can be consumed by the Tx.

- If the Tx consumes multiple UTxOs, each UTxO has its own validator, which decides only for that particular UTxO.

- For the transaction to be successful, the validators of all UTxOs must return True.

- There are several types of validators depending on what they validate. The 2 most common ones are:
    - **Spend:** The validator that runs to allow or deny the consumption of a UTxO.
    - **Mint:** The validator that runs to mint or burn tokens.



(E)UTxO Model: (E)UTxO VS Account

The **Account Model** is used by Ethereum and other EVM blockchains. It works like bank accounts, where the total of each account is recorded and updated with each transaction.

(E)UTxO Model	Account Model
More difficult to understand and use than the Account Model	Easy to understand and use
Allows sequential and parallel Tx (Using global state is poor design)	Only sequential (Using global state is considered good design)
Fully deterministic and reproducible Tx	Indeterministic Tx
Typically more complex architectures	Typically simpler architectures

The End

Questions?

INPUT | OUTPUT