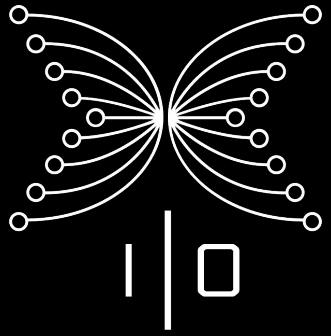
INPUT | OUTPUT

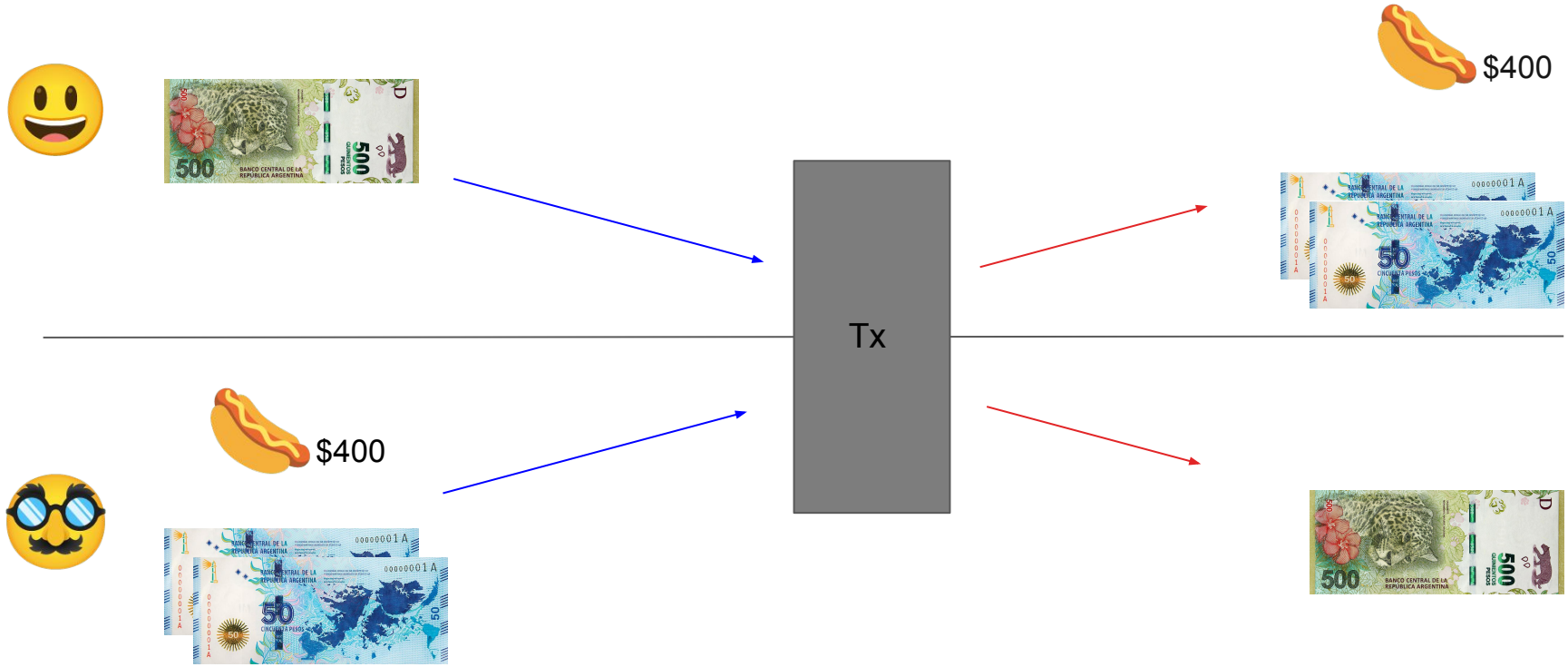
UTxO y (E)UTxO



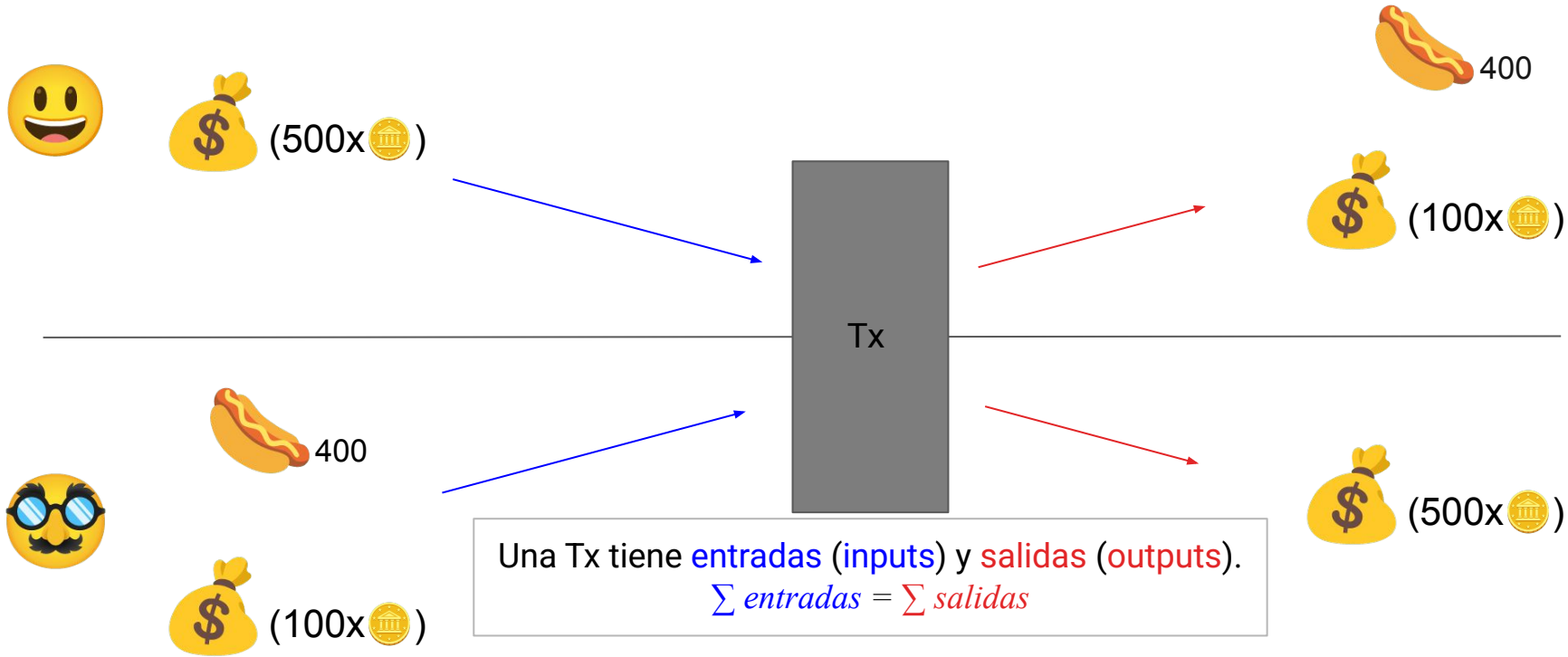
01

MODELO UTxO

Transacciones en efectivo



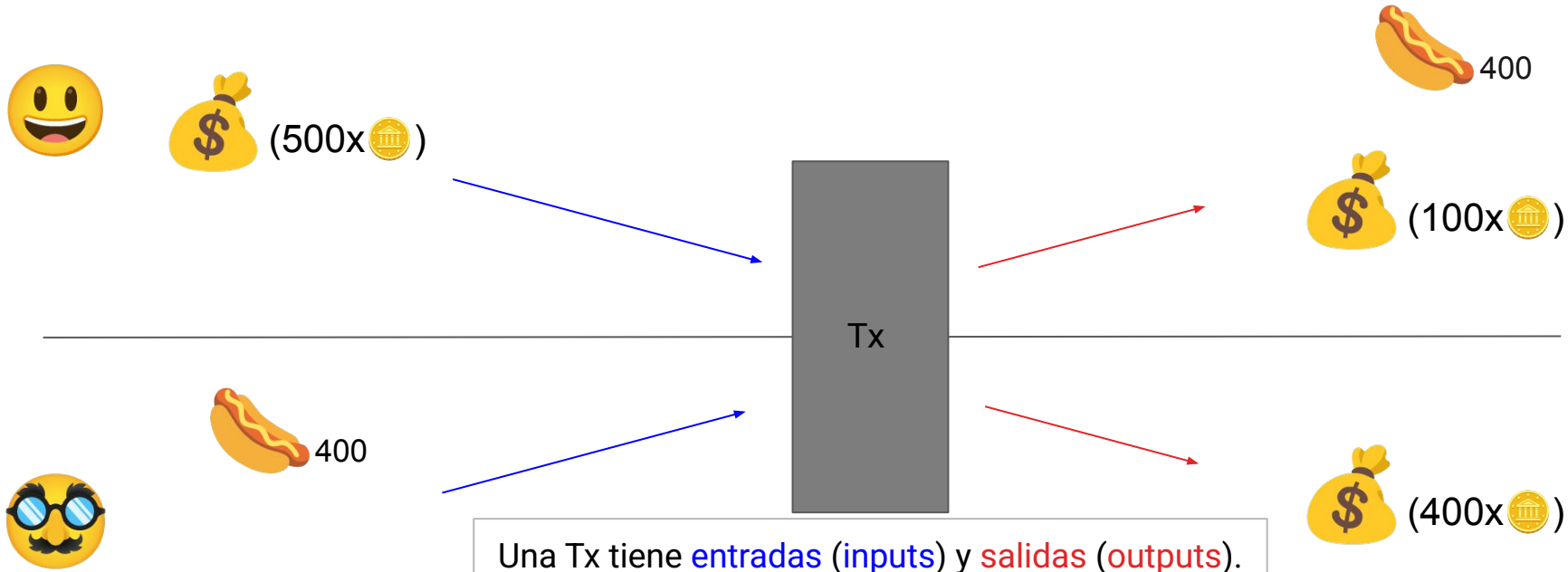
Transacciones en UTxO



Una Tx tiene entradas (inputs) y salidas (outputs).

$$\sum entradas = \sum salidas$$

Transacciones en UTxO

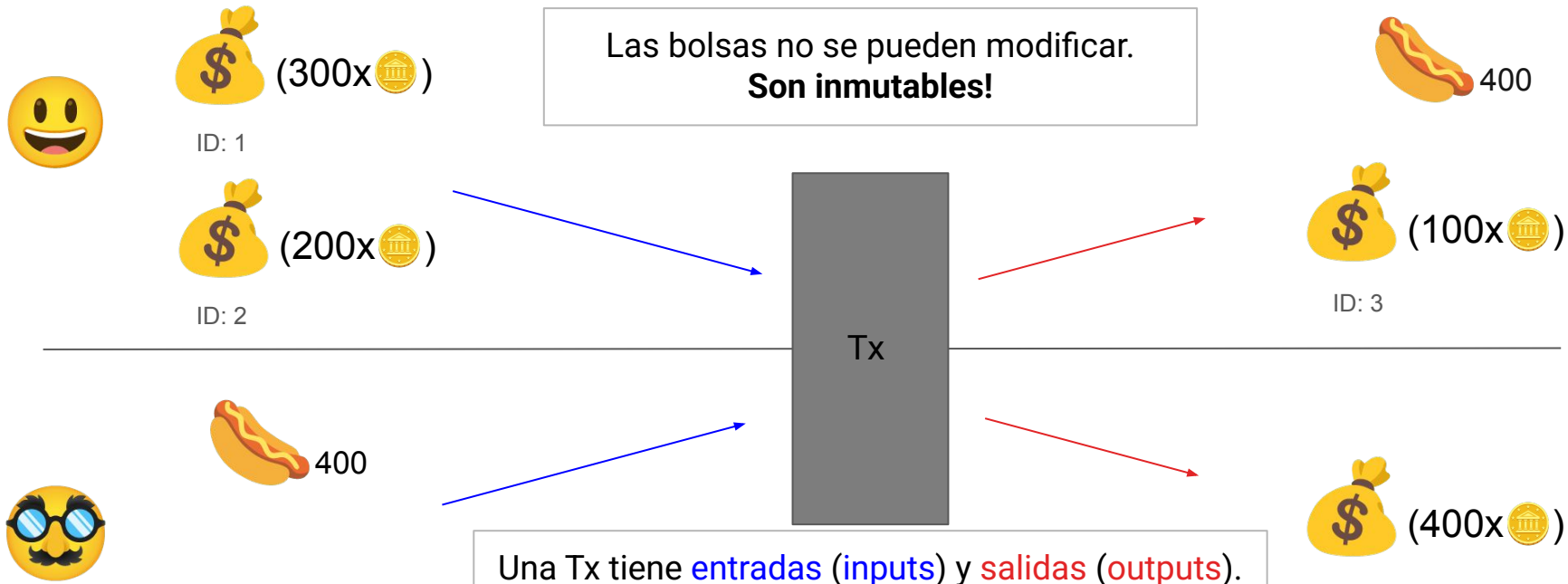


Una Tx tiene entradas (inputs) y salidas (outputs).

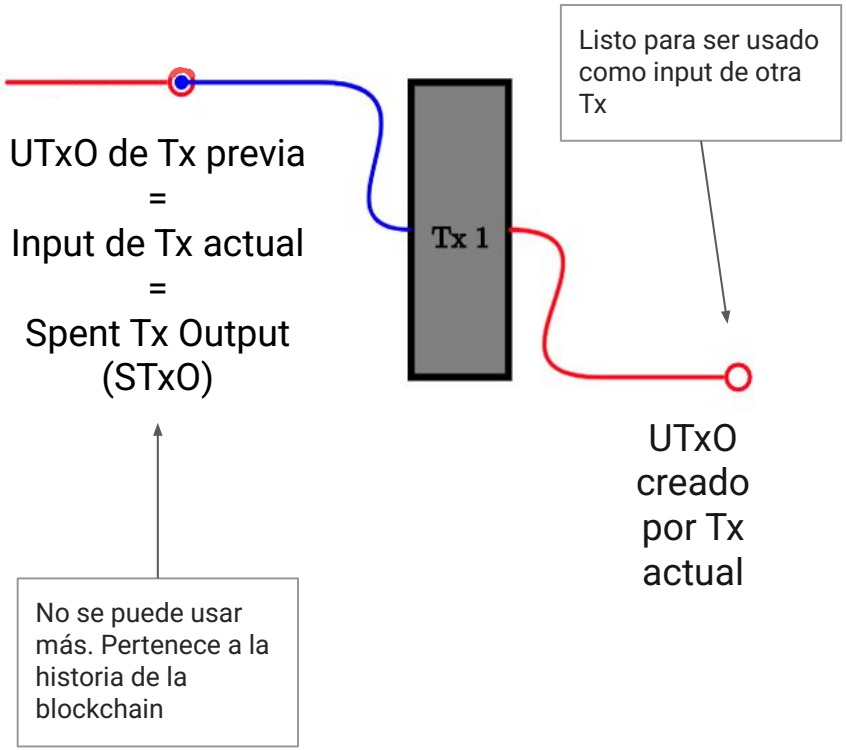
$$\sum entradas = \sum salidas$$

Durante la transacción, se destruyen y crean bolsas

Transacciones en UTxO

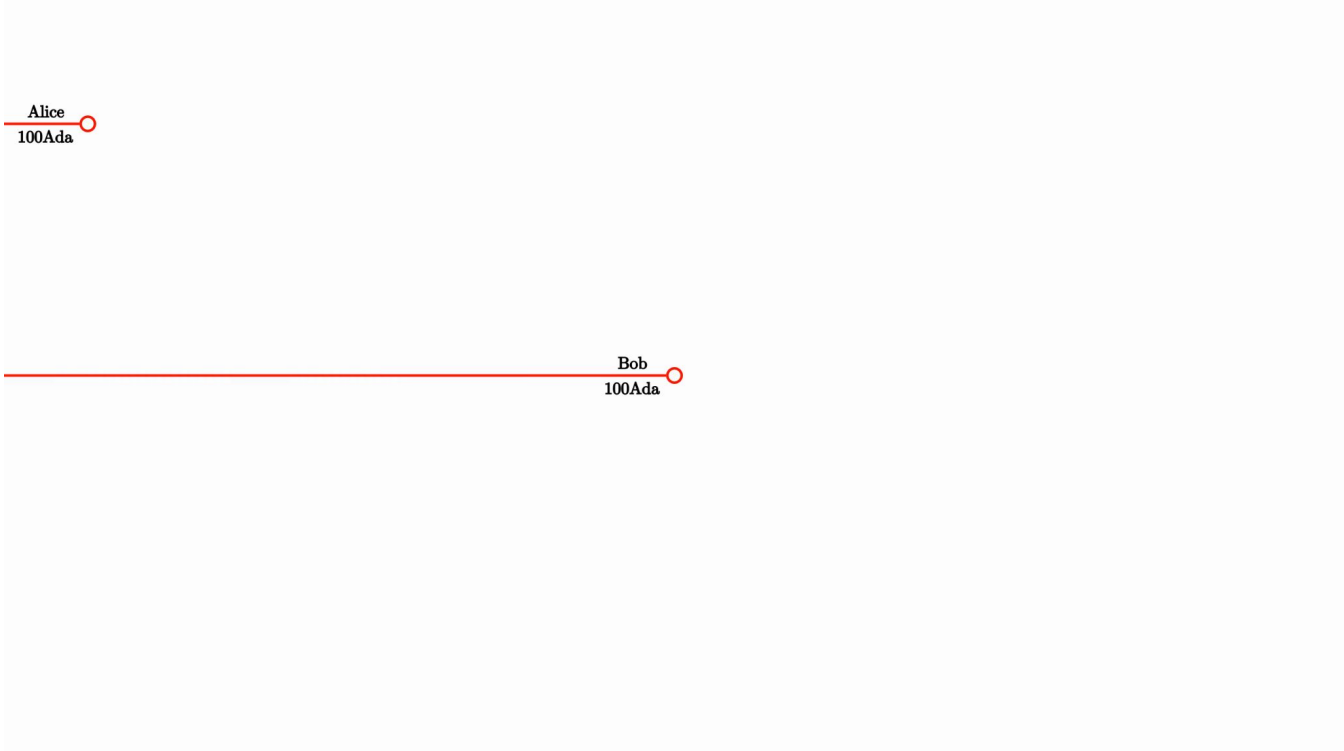


Modelo UTxO

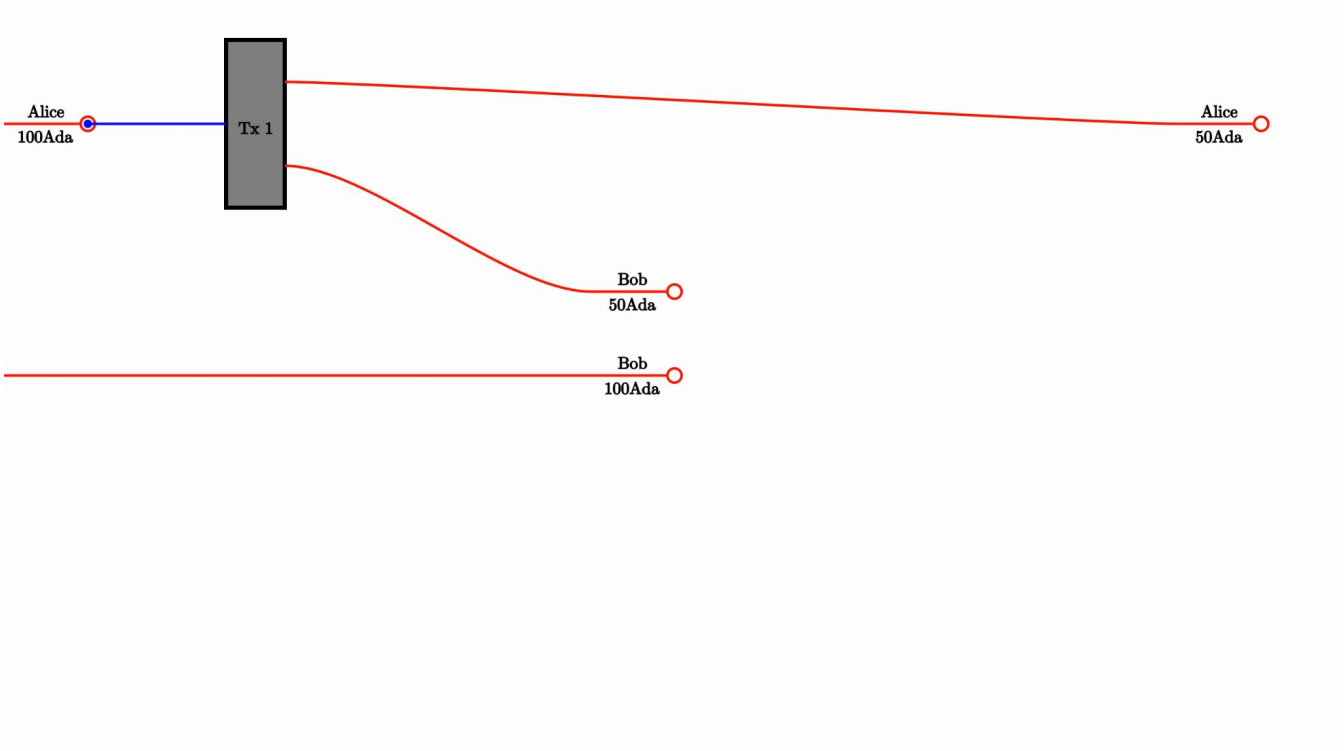


- Transacciones tienen un **número arbitrario de inputs** (entradas) y outputs (salidas). (Mínimo 1 y 1).
- **Unspent Transaction Output (UTxO)** son las salidas de transacciones que todavía no han sido consumidas por otras transacciones.
- Si se quiere usar parte del valor encerrado en un UTxO, se tiene que **consumir completamente**.
- Lo único que pasa en la blockchain son estas transacciones. Los UTxO se consumen y crean, pero **NO SE MODIFICAN**!

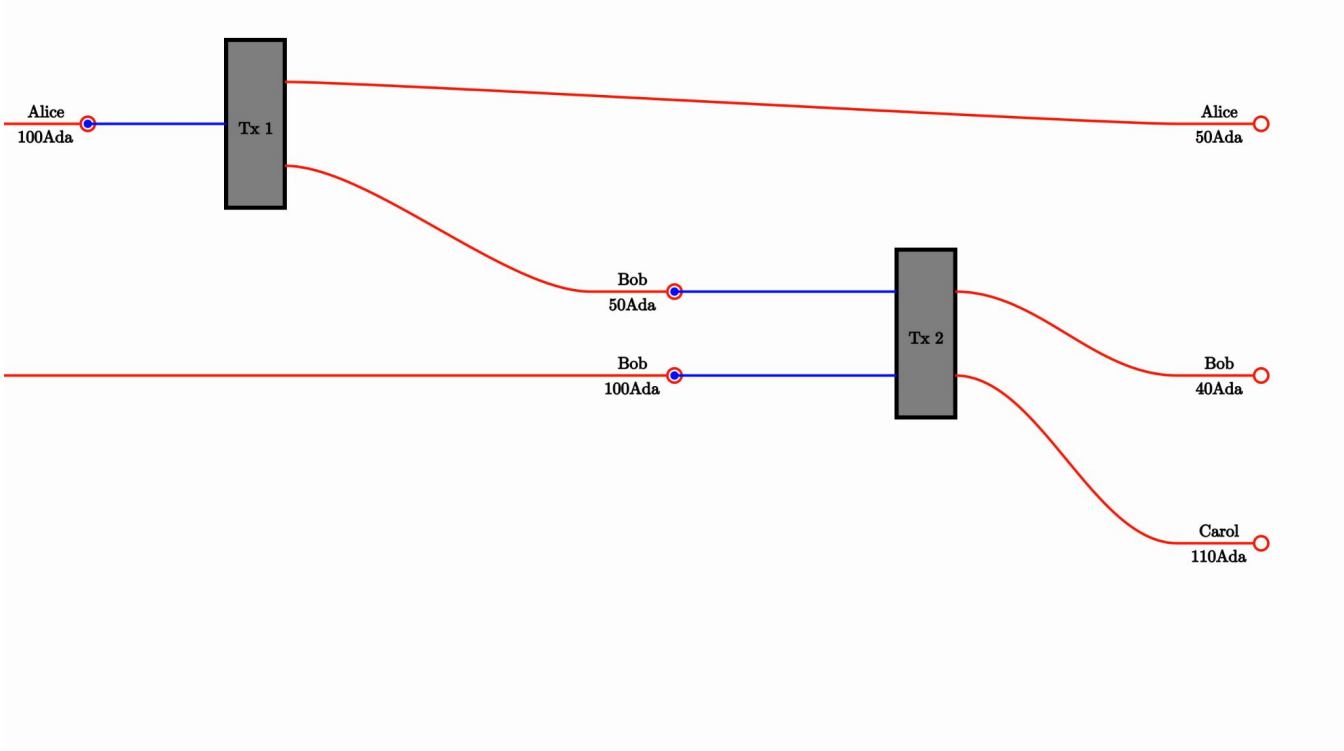
Transacciones en UTxO: ✍️ Alice le manda 50 ADA a Bob?



Transacciones en UTxO: ✍️ Bob le manda 110Ada a Carol?



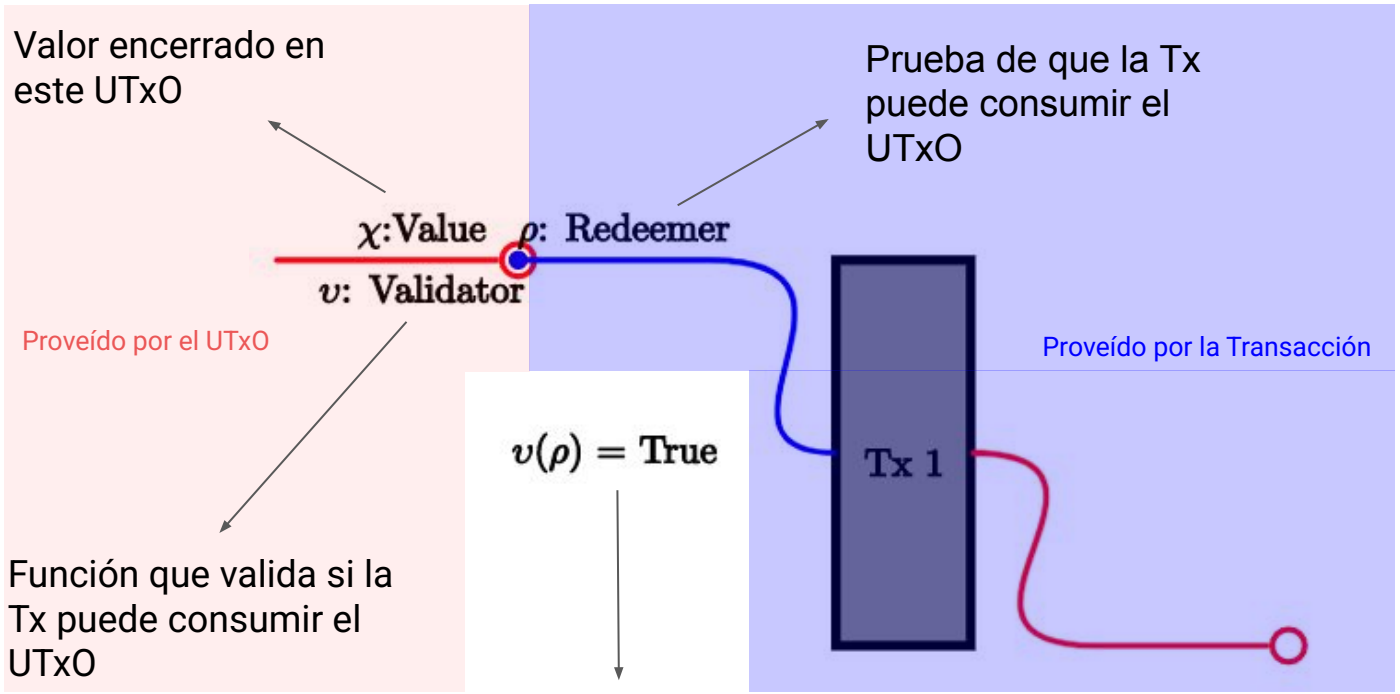
Transacciones en UTxO: ✍️ Alice le manda 50 ADA a Bob



Transacciones en UTxO: ✍️ Realizar Tx con Lace wallet

1. Instalar Extensión lace.io.
2. Crear una nueva wallet.
3. Configurar para usar Preview/Preprod Testnets.
4. Pedir fondos a la Testnet faucet.
5. Explorar UTxO de Tx en blockchain explorer.
6. Crear una nueva Tx enviando X ADA a otro estudiante.
7. Explorar UTxO de Tx en blockchain explorer.

Transacciones en UTxO: Cómo me aseguro que no consuman mis UTxO?



Se aplica el Validador al Redeemer y devuelve `True`/`False` si la Tx puede/no puede consumir el UTxO

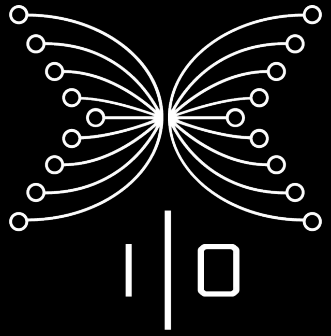
Transacciones en UTxO: Limitaciones del modelo UTxO

El Modelo UTxO sirve perfectamente para **transferencias simples**. Pero **no tiene suficiente información** como para realizar transferencias más complejas. Por ejemplo:

- Permitir consumir el UTxO después de una fecha específica
- Permitir consumir el UTxO sólo si se envía a una persona específica
- Permitir consumir el UTxO sólo si también se consume otro UTxO específico
- Permitir consumir el UTxO sólo si existe otro UTxO (que no es consumido) con un valor específico
- …

Pero esas son cosas **necesarias** para poder crear **programas descentralizados** que permitan hacer **lo mismo** que hoy en día se utiliza **centralizadamente**:

- Prestamos
- Financiaciones
- Inversiones
- Calendario de adquisición de acciones
- …

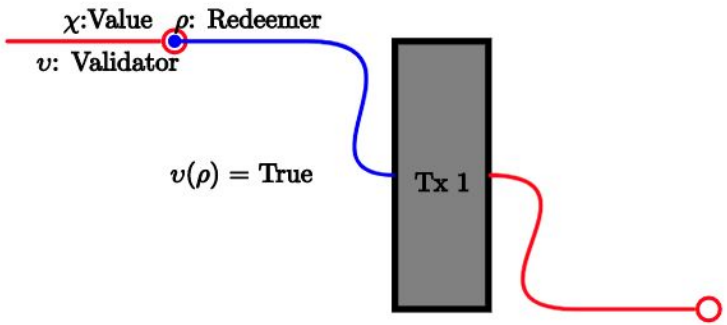


02

MODELO (E)UTxO

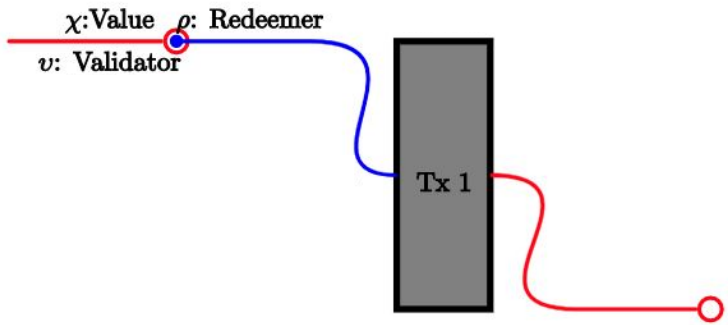
Modelo (E)UTxO: Comparar modelo UTxO con (E)UTxO

UTxO



Expresividad extremadamente limitada

Extended UTxO



Expresividad suficiente como para reemplazar gran parte de los servicios financieros/bancarios centralizados

Modelo (E)UTxO: Ejemplo Vesting - Idea

- Muchas compañías tipo Facebook, Google, Apple, y Netflix, proveen “**Restricted Stocks Units**” (RSU) a los empleados como parte de la compensación.

- Las “Unidades de acciones restringidas” tienen usualmente un calendario de por ejemplo **25% de las acciones por año durante 4 años**.

- Entre otras razones, esto se hace para que el empleado:
  - No venda el 100% de las acciones a la vez (potencialmente impactando el precio del mercado)
  - No las venda y se vaya al otro día.

Modelo (E)UTxO: Ejemplo Vesting - Diseño

1. En el momento de que el empleado acepta la posición, el empleador va a generar **4 UTxO con 25% de las acciones en cada uno**.

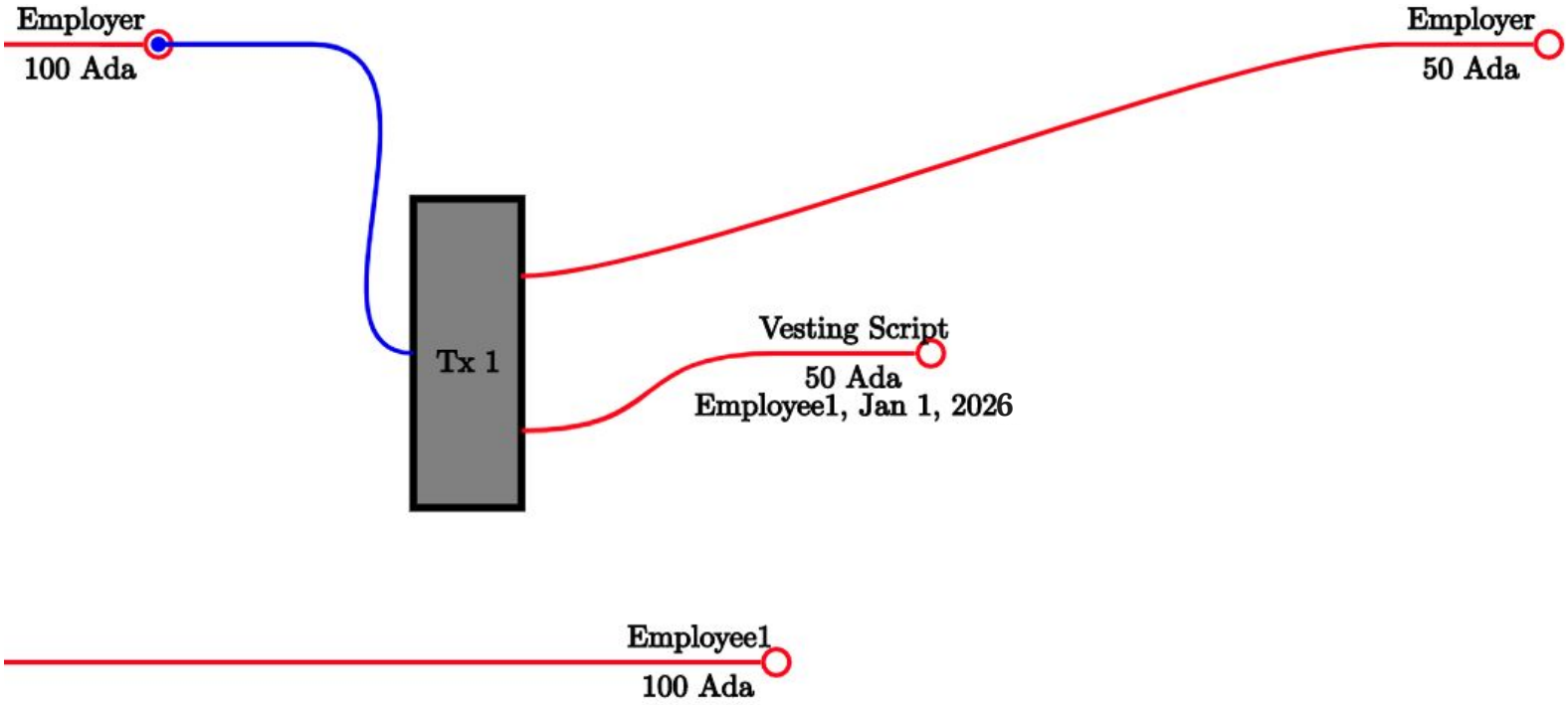
2. Los 4 UTxO **chequean** que el que quiere consumir el UTxO **sea el empleado** y que una **fecha límite haya pasado**.

3. Cada UTxO tiene una **fecha límite distinta** (Ej. 1/1/2026, 1/1/2027, 1/1/2028, 1/1/2029)

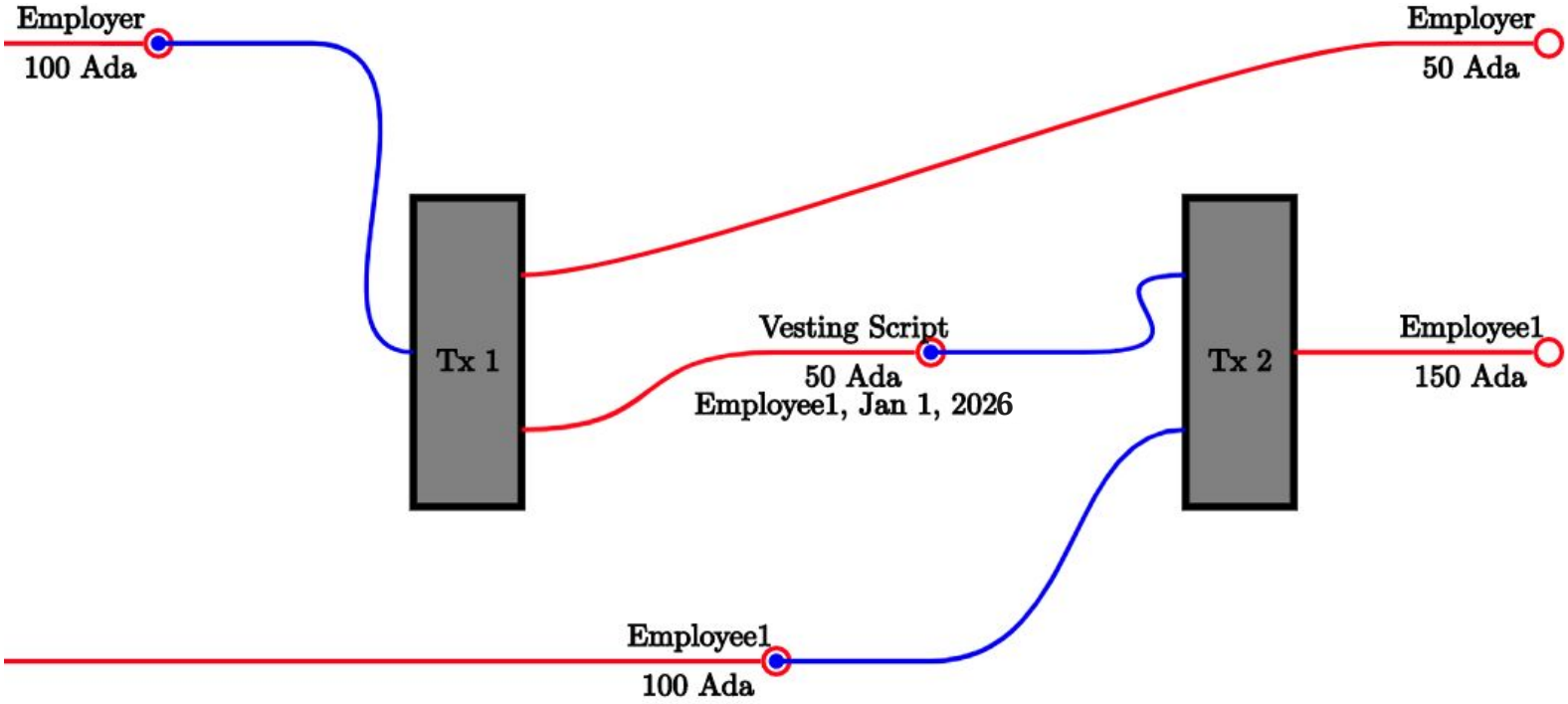
4. El beneficiario sólo tiene que **esperar** que pase la fecha límite de cada uno **para poder consumir el UTxO** y obtener sus acciones.

Modelo (E)UTxO: Ejemplo Vesting - Visualización

Modelo (E)UTxO: Ejemplo Vesting - Visualización



Modelo (E)UTxO: Ejemplo Vesting - Visualización



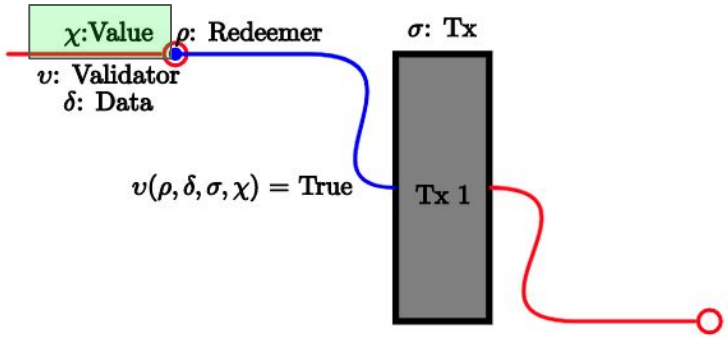
Modelo (E)UTxO: Ejemplo Vesting - Validador

```
pub type VestingDatum {
  beneficiary: VerificationKeyHash,
  deadline: Int,
}

validator placeholder {
  spend(datum: Option<VestingDatum>, _r: Data, _utxo, self: Transaction) {
    expect Some(datum) = datum
    let Transaction { extra_signatories, validity_range, .. } = self
    let is_signed = list.has(extra_signatories, datum.beneficiary)
    let is_after_deadline = interval.is_entirely_after(validity_range, datum.deadline)
    is_signed? && is_after_deadline?
  }

  else(_) { fail }
}
```

Modelo (E)UTxO: Detalles de cada ítem - Valor

Que puede ser?:

- Lovelace (1 ADA = 1.000.000 Lovelace) ✅
- Lovelace + Tokens (FT y/o NFT) ✅
- Tokens sin Lovelace 🚫



A tener en cuenta:

- minUTxO (Lovelace mínimos que tiene que tener el UTxO) depende del tamaño del UTxO.

- Se puede tener muchos Tokens al mismo tiempo, y se puede tener millones de un Token sin que haga la diferencia en costo o tamaño.

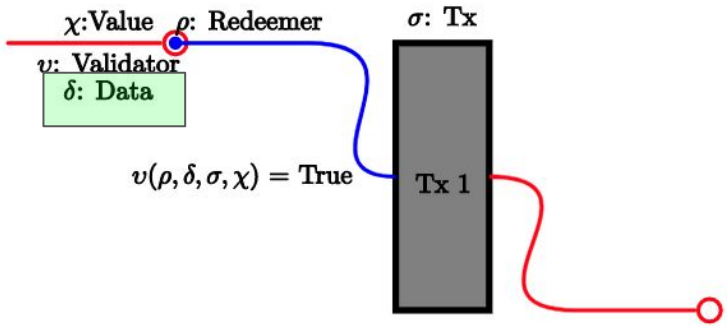
Modelo (E)UTxO: Detalles de cada ítem - Datum

Que puede ser?:

- Cualquier cosa (un número, texto, una lista de PKH, una estructura inventada, …)

A tener en cuenta:

- El **creador del UTxO** tiene que comprometerse a un Datum al crear el UTxO. Pero no necesariamente proveerlo públicamente.



Creador Provee		Consumidor Provee
En UTxO	**En Transacción**	
Hash del Datum	-	Datum
Hash del Datum	Datum	Datum
Datum (inline)	-	-

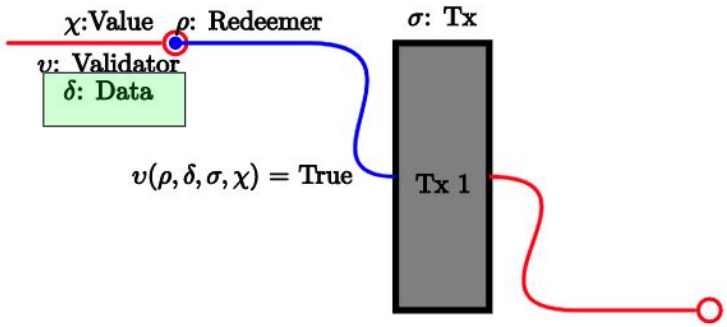
Modelo (E)UTxO: Detalles de cada ítem - Datum

Que puede ser?:

- Cualquier cosa (un número, texto, una lista de PKH, una estructura inventada, …)

A tener en cuenta:

- El **creador del UTxO** tiene que comprometerse a un Datum al crear el UTxO. Pero no necesariamente proveerlo públicamente.



Creador Provee		Consumidor Provee
En UTxO	**En Transacción**	
Hash del Datum	-	Datum
Hash del Datum	Datum	Datum
Datum (inline)	-	-

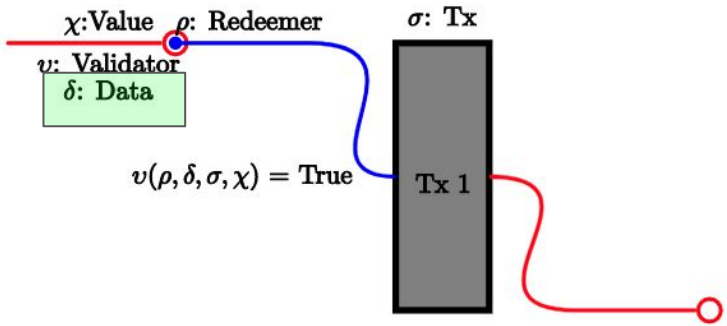
Modelo (E)UTxO: Detalles de cada ítem - Datum

Que puede ser?:

- Cualquier cosa (un número, texto, una lista de PKH, una estructura inventada, …)

A tener en cuenta:

- El **creador del UTxO** tiene que comprometerse a un Datum al crear el UTxO. Pero no necesariamente proveerlo públicamente.



Creador Provee		Consumidor Provee
En UTxO	**En Transacción**	
Hash del Datum	-	Datum
Hash del Datum	Datum	Datum
Datum (inline)	-	-

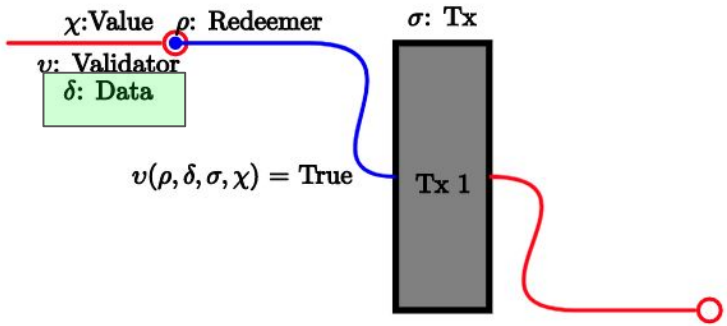
Modelo (E)UTxO: Detalles de cada ítem - Datum

Que puede ser?:

- Cualquier cosa (un número, texto, una lista de PKH, una estructura inventada, …)

A tener en cuenta:

- El **creador del UTxO** tiene que comprometerse a un Datum al crear el UTxO. Pero no necesariamente proveerlo públicamente.



Creador Provee		Consumidor Provee
En UTxO	**En Transacción**	
Hash del Datum	-	Datum
Hash del Datum	Datum	Datum
Datum (inline)	-	-

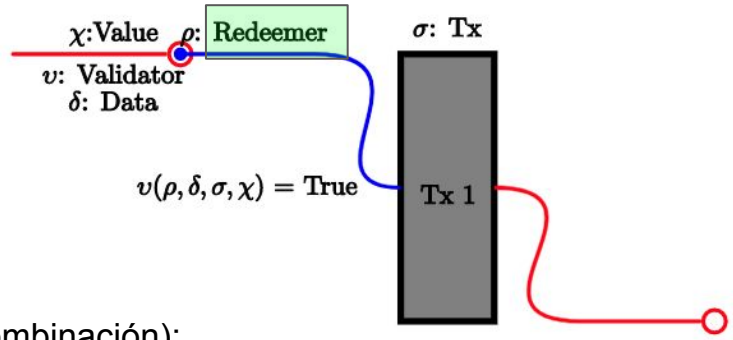
Modelo (E)UTxO: Detalles de cada ítem - Redeemer

Que puede ser?:

- Cualquier cosa (un número, texto, una lista de PKH, una estructura inventada, …)

Qué diferencia hay con el Datum?

- El Redeemer lo provee el que **consume** el UTxO



Usos **comunes** de Datums y Redeemers (en cualquier combinación):

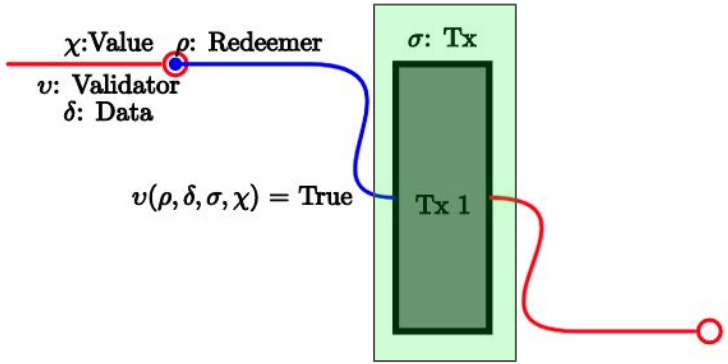
Datum
Quién/Cuándo/Con qué puedo consumir UTxO?
Estado actual del UTxO
Metadatos/”Configuraciones”

Redeemer
Razón de uso: Pedir prestamo, Pagar una cuota, Cancelar prestamo
Información que sólo X persona sabe
Valor por el cual reemplazar Datum

Modelo (E)UTxO: Detalles de cada ítem - Contexto de Transacción

A tener en cuenta:

- En Cardano mainet, solo puede ser un valor de tipo ScriptContext.

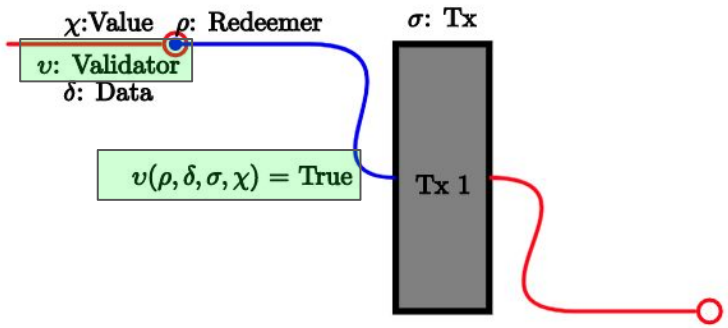


Qué contiene el valor ScriptContext?:

- Tipo de Tx
- Rango temporal de validez
- Inputs (con sus respectivos Valores, Datums y Redeemers)
- Outputs (Con sus respectivos Valores y Datums)
- Referencias a UTxO y validadores relevantes pero que no son parte de la Tx
- Costo de la Tx
- Los que firmaron la Tx
- Identificador y cantidad de Tokens siendo acuñados/quemados en la Tx
- ...

Modelo (E)UTxO: Detalles de cada ítem - Validador/Script

- Tiene en cuenta todos los elementos que acabamos de ver para tomar la decisión si el UTxO puede ser consumido por la Tx.
- Si la Tx consume múltiples UTxO, cada UTxO tiene su própio validador que decide sólo por ese UTxO.
- Para que la Tx sea exitosa, los validadores de todos los UTxO tienen que devolver True.
- Hay varios tipos de validadores. En este curso vamos a aprender a usar los 2 más comunes:
  - **Spend:** El que se ejecuta para permitir/denegar consumir un UTxO.
  - **Mint:** El que se ejecuta para acuñar o quemar un token.



Modelo (E)UTxO: (E)UTxO VS Account

El Modelo de cuentas (Account Model) es el utilizado por Ethereum y otras EVM blockchains. Funciona como las cuentas de banco donde se anota el total de cada uno en una cuenta y se modifica ese valor en cada transacción.

(E)UTxO Model	Account Model
Más difícil de entender y usar qué Account Model	Fácil de entender y usar
Permite Tx en secuencia y en paralelo (Usar estado global es mal diseño)	Sólo secuencial (Usar estado global es buen diseño)
Transacciones completamente deterministas y reproducibles	Transacciones indeterministas
Arquitecturas usualmente más complejas	Arquitecturas usualmente más simples

INPUT | OUTPUT

Preguntas?