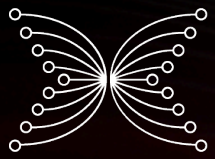
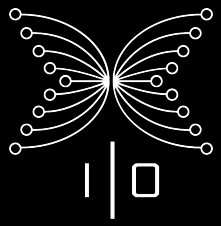


INPUT | OUTPUT

Plutus Scripts y sus propósitos

Robertino Martinez



1

Validadores

Validadores : Estructura

Validadores : Estructura

Para crear un validador, usamos la keyword `validator` en un bloque con nombre:

```
validator mi_validador {
  // ...
}
```

Validadores : Estructura

Para crear un validador, usamos la keyword `validator` en un bloque con nombre:

```
validator mi_validador {
  // ...
}
```

> ⚠ **Advertencia**
>
> Dentro de este bloque solo podemos usar predicados con nombres y argumentos específicos que llamamos **handles**.

Validadores : Estructura

Para crear un validador, usamos la keyword `validator` en un bloque con nombre:

```
validator mi_validador {
  // ...
}
```

> ⚠ **Advertencia**
>
> Dentro de este bloque solo podemos usar predicados con nombres y argumentos específicos que llamamos **handles**.

Por ejemplo:

Validadores : Estructura

Para crear un validador, usamos la keyword `validator` en un bloque con nombre:

```
validator mi_validador {
  // ...
}
```

> ⚠ **Advertencia**
>
> Dentro de este bloque solo podemos usar predicados con nombres y argumentos específicos que llamamos **handles**.

Por ejemplo:

```
validator my_script {
  mint(redeemer: MyRedeemer, policy_id: PolicyId, self: Transaction) {
    todo @"lógica del validor"
  }
}
```

Validadores : Handles

Los handles son predicados que indican:

Validadores : Handles

Los handles son predicados que indican:

- **Handle devuelve `True` :** El handle permite la transacción (puede fallar por otra razón).

Validadores : Handles

Los handles son predicados que indican:

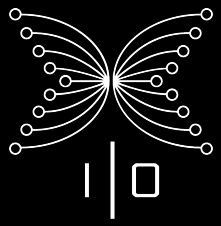
- **Handle devuelve `True` :** El handle permite la transacción (puede fallar por otra razón).
- **Handle devuelve `False` o *falla*:** La transacción completa falla (es rechazada por el ledger), independientemente de si todo el resto está bien.

Validadores : Handles

Los handles son predicados que indican:

- **Handle devuelve `True` :** El handle permite la transacción (puede fallar por otra razón).
- **Handle devuelve `False` o *falla*:** La transacción completa falla (es rechazada por el ledger), independientemente de si todo el resto está bien.

> **Info**
>
> Estos handles representan los **posibles tipos de transacción o «propósitos»**.
> Cualquier otra cosa que no sea estos handles, no compila.



2

Propósitos

Propósitos : `spend`

Scripts que validan si la transacción puede **consumir** un UTxO.

Propósitos : `spend`

Scripts que validan si la transacción puede **consumir** un UTxO.

```
validator placeholder {
    spend(datum: Option<Data>, redeemer: Data, utxo: OutputReference, self: Transaction) {
      todo @"spend logic goes here"
    }
    // Otros handles...
  }
```

- `datum: Option<Data>` : Datum del UTxO que se se está evaluando.

Propósitos : `spend`

Scripts que validan si la transacción puede **consumir** un UTxO.

```
validator placeholder {
    spend(datum: Option<Data>, redeemer: Data, utxo: OutputReference, self: Transaction) {
      todo @"spend logic goes here"
    }
    // Otros handles...
  }
```

- `datum: Option<Data>` : Datum del UTxO que se se está evaluando.
- `redeemer: Data` : Redeemer proveído por la transacción para éste UTxO en particular.

Propósitos : `spend`

Scripts que validan si la transacción puede **consumir** un UTxO.

```
validator placeholder {
    spend(datum: Option<Data>, redeemer: Data, utxo: OutputReference, self: Transaction) {
      todo @"spend logic goes here"
    }
    // Otros handles...
  }
```

- `datum: Option<Data>` : Datum del UTxO que se se está evaluando.
- `redeemer: Data` : Redeemer proveído por la transacción para éste UTxO en particular.
- `utxo: OutputReference` : Identificador único del UTxO que se está evaluando.

Propósitos : `spend`

Scripts que validan si la transacción puede **consumir** un UTxO.

```
validator placeholder {
    spend(datum: Option<Data>, redeemer: Data, utxo: OutputReference, self: Transaction) {
      todo @"spend logic goes here"
    }
    // Otros handles...
  }
```

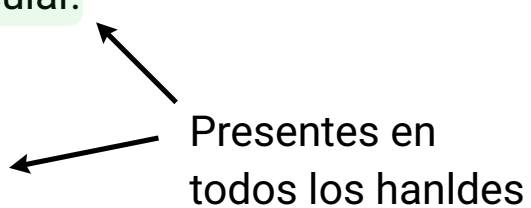
- `datum: Option<Data>` : Datum del UTxO que se se está evaluando.
- `redeemer: Data` : Redeemer proveído por la transacción para éste UTxO en particular.
- `utxo: OutputReference` : Identificador único del UTxO que se está evaluando.
- `self: Transaction` : Contexto de la transacción que está ejecutando este script.

Propósitos : `spend`

Scripts que validan si la transacción puede **consumir** un UTxO.

```
validator placeholder {
    spend(datum: Option<Data>, redeemer: Data, utxo: OutputReference, self: Transaction) {
      todo @"spend logic goes here"
    }
    // Otros handles...
  }
```

- `datum: Option<Data>` : Datum del UTxO que se se está evaluando.
- `redeemer: Data` : Redeemer proveído por la transacción para éste UTxO en particular.
- `utxo: OutputReference` : Identificador único del UTxO que se está evaluando.
- `self: Transaction` : Contexto de la transacción que está ejecutando este script.



Propósitos : `spend`

Scripts que validan si la transacción puede **consumir** un UTxO.

```
validator placeholder {
    spend(datum: Option<Data>, redeemer: Data, utxo: OutputReference, self: Transaction) {
      todo @"spend logic goes here"
    }
    // Otros handles...
  }
```

> **Info**
>
> - Para que un validador custodie un UTxO, hay que crearlo en la dirección del validador.
> - La dirección del validador depende del hash del script. Por lo tanto, los scripts son inmutables.

Propósitos : `mint`

Scripts que validan si los tokens que **la transacción quiere acuñar o quemar** cumplen con una política previamente establecida.

```
validator placeholder {
    mint(redeemer: Data, policy_id: PolicyId, self: Transaction) {
      todo @"mint logic goes here"
  }
  // Otros handles...
}
```

- `policy_id: PolicyId` : identificador único de la política (hash del script)

Propósitos : `mint`

Scripts que validan si los tokens que **la transacción quiere acuñar o quemar** cumplen con una política previamente establecida.

```
validator placeholder {
    mint(redeemer: Data, policy_id: PolicyId, self: Transaction) {
      todo @"mint logic goes here"
  }
  // Otros handles...
}
```

Info

- Cada token tiene su propia política, la cual es única e inmutable.
- Una política monetaria sólo custodia el acuñado o quemado de sus propios tokens.

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

PolicyId	AssetName	Int

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

PolicyId	AssetName	Int
Identificador único de la política		

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

PolicyId	AssetName	Int
Identificador único de la política	Nombre del token	

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

PolicyId	AssetName	Int
Identificador único de la política	Nombre del token	Cantidad del token

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

PolicyId	AssetName	Int
Identificador único de la política	Nombre del token	Cantidad del token
Hash del script con Blake2b_224 (array de exactamente 28 bytes)		

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

PolicyId	AssetName	Int
Identificador único de la política	Nombre del token	Cantidad del token
Hash del script con Blake2b_224 (array de exactamente 28 bytes)	Array de entre 0 y 32 bytes	

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

PolicyId	AssetName	Int
Identificador único de la política	Nombre del token	Cantidad del token
Hash del script con Blake2b_224 (array de exactamente 28 bytes)	Array de entre 0 y 32 bytes	Número entero

Propósitos : Qué son los valores?

```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

`PolicyId`	`AssetName`	`Int`
Identificador único de la política	Nombre del token	Cantidad del token
Hash del script con Blake2b_224 (array de exactamente 28 bytes)	Array de entre 0 y 32 bytes	Número entero

`#"010203…"`	`#"040506"`
	`#"070809"`
	`...`
		`42`
		`18`
		`...`

Propósitos : Qué son los valores?

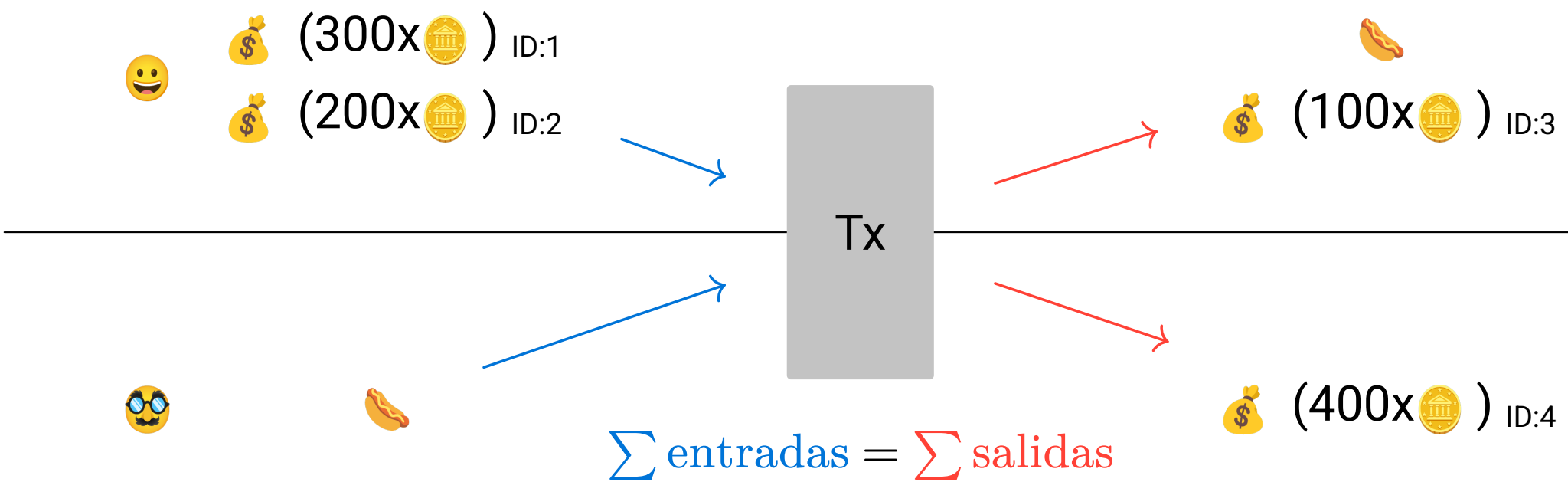
```
Value = Dict<PolicyId, Dict<AssetName, Int>>
```

`PolicyId`	`AssetName`	`Int`
Identificador único de la política	Nombre del token	Cantidad del token
Hash del script con Blake2b_224 (array de exactamente 28 bytes)	Array de entre 0 y 32 bytes	Número entero

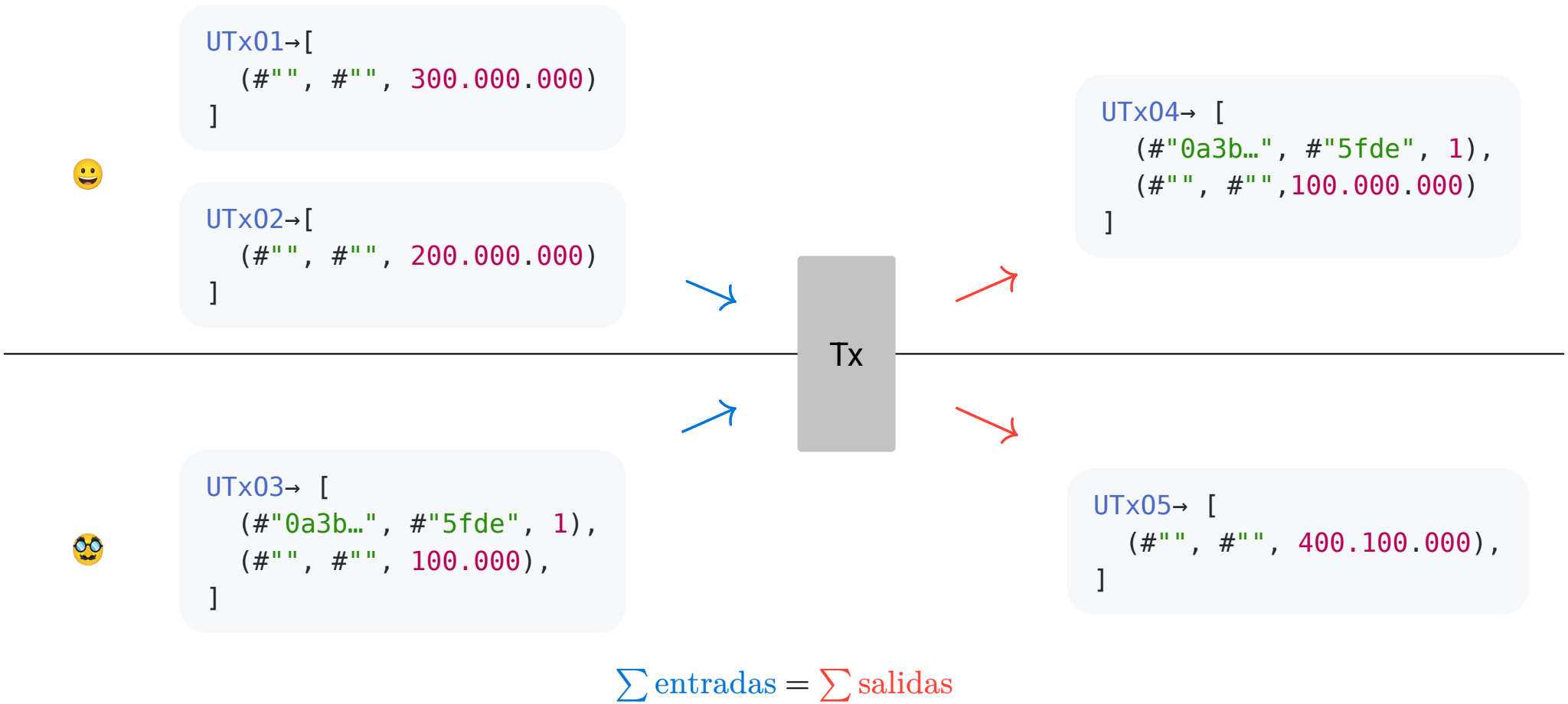
`#"010203…"`	`#"040506"`
	`#"070809"`
	`...`
		`42`
		`18`
		`...`

`#""`	`#""`	`10_000_000`

Propósitos : Transacción con valores imaginarios

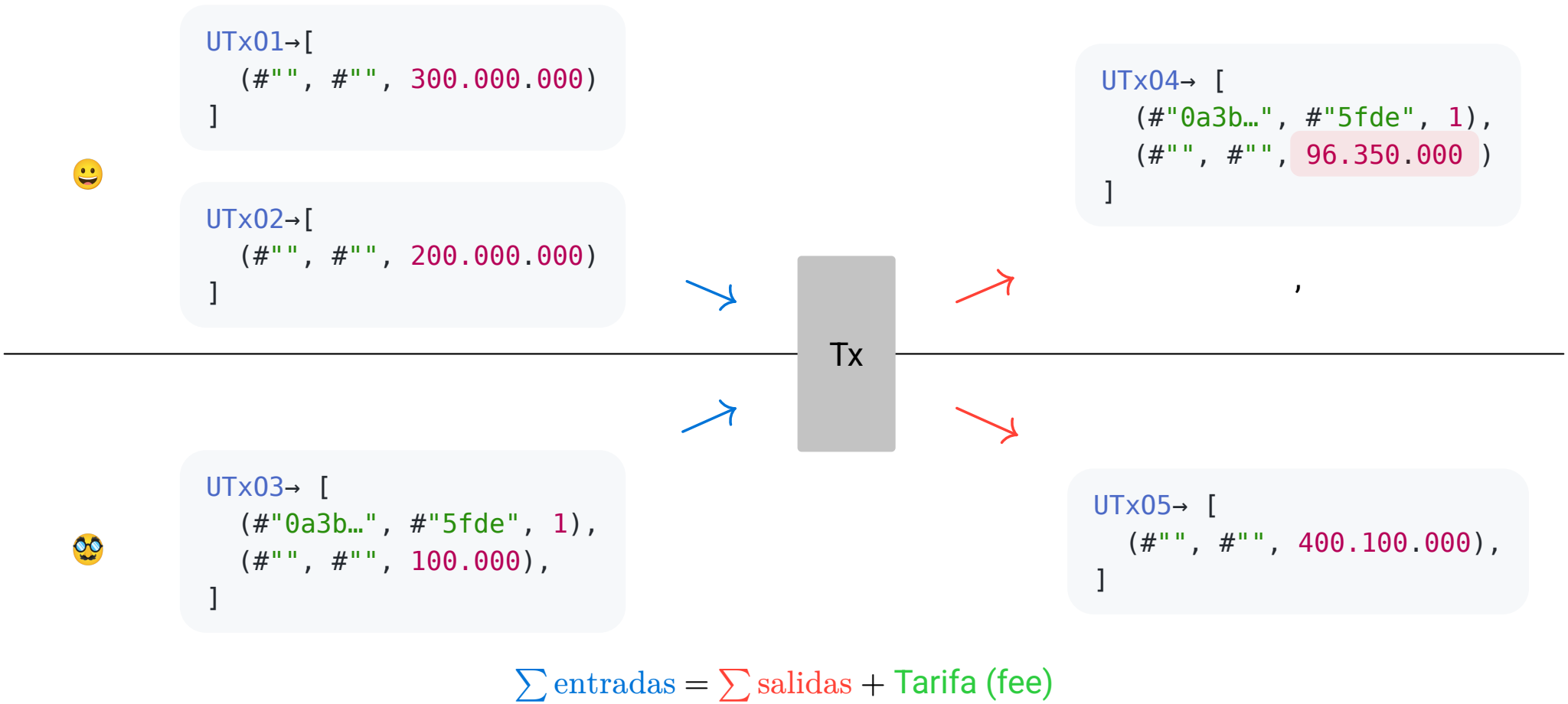


Propósitos : Transacción con Valores reales sin tarifa (fee)



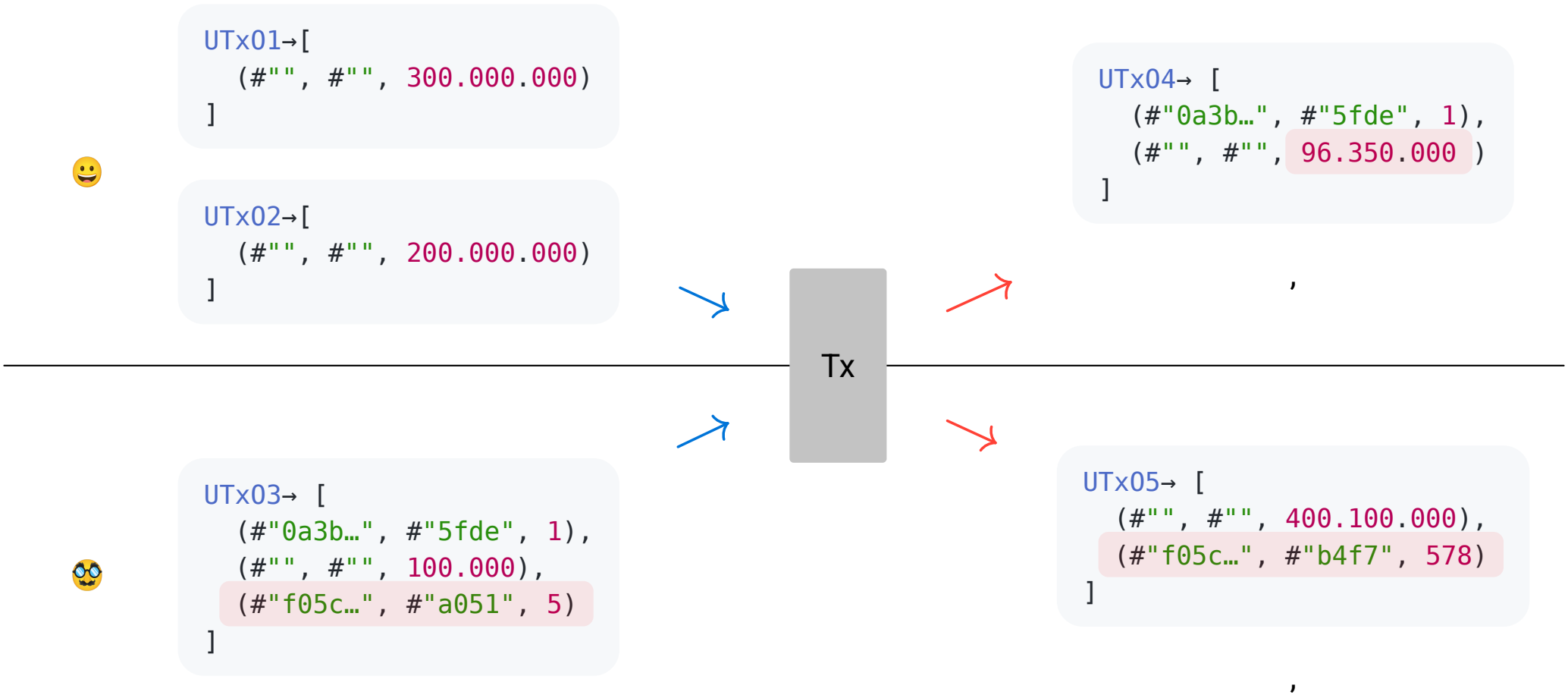
$$\sum \text{entradas} = \sum \text{salidas}$$

Propósitos : Transacción con Valores reales con tarifa (fee)



$$\sum \text{entradas} = \sum \text{salidas} + \text{Tarifa (fee)}$$

Propósitos : Transacción con Valores reales con tarifa (fee) y minting/burning



$$\sum \text{entradas} = \sum \text{salidas} + \text{Tarifa (fee)} + \text{T. Acuñados} + \text{T. Quemados}$$

Propósitos : `withdraw`

Scripts que validan **retiros de recompensas** acumulados por hacer **staking**.

```
validator placeholder {
  withdraw(redeemer: Data, account: Credential, self: Transaction) {
      todo @"withdraw logic goes here"
  }
  // Otros handles...
}
```

Propósitos : `withdraw`

Scripts que validan **retiros de recompensas** acumulados por hacer **staking**.

```
validator placeholder {
  withdraw(redeemer: Data, account: Credential, self: Transaction) {
      todo @"withdraw logic goes here"
  }
  // Otros handles...
}
```

- `account: Credential` : identificador único de la cuenta de recompensa (hash del script)

Propósitos : `withdraw`

Scripts que validan **retiros de recompensas** acumulados por hacer **staking**.

```
validator placeholder {
  withdraw(redeemer: Data, account: Credential, self: Transaction) {
      todo @"withdraw logic goes here"
  }
  // Otros handles...
}
```

- `account: Credential` : identificador único de la cuenta de recompensa (hash del script)

> ⚠️ **Advertencia**
>
> No se puede retirar **parte** de las recompensas, tiene que ser **todo**. Pero podemos tomar decisiones sobre cómo distribuir ese todo.

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

- `certificate: Certificate` : Certificado que se está utilizando

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

- `certificate: Certificate` : Certificado que se está utilizando

Los cambios pueden ser varios:

Registrar una credencial		

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

- `certificate: Certificate` : Certificado que se está utilizando

Los cambios pueden ser varios:

Registrar una credencial	Dar de baja una credencial	

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

- `certificate: Certificate` : Certificado que se está utilizando

Los cambios pueden ser varios:

Registrar una credencial	Dar de baja una credencial	Delegar una credencial a una stake pool o DRep

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

- `certificate: Certificate` : Certificado que se está utilizando

Los cambios pueden ser varios:

Registrar una credencial	Dar de baja una credencial	Delegar una credencial a una stake pool o DRep
Registrar una stake pool		

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

- `certificate: Certificate` : Certificado que se está utilizando

Los cambios pueden ser varios:

Registrar una credencial	Dar de baja una credencial	Delegar una credencial a una stake pool o DRep
Registrar una stake pool	Retirar una stake pool	

Propósitos : `publish`

Scripts que validan **cambios en certificados**:

```
validator placeholder {
    publish(redeemer: Data, certificate: Certificate, self: Transaction) {
      todo @"publish logic goes here"
    }
  // Otros handles...
}
```

- `certificate: Certificate` : Certificado que se está utilizando

Los cambios pueden ser varios:

Registrar una credencial	Dar de baja una credencial	Delegar una credencial a una stake pool o DRep
Registrar una stake pool	Retirar una stake pool	...

Propósitos : `vote`

Scripts que validan **votar** en propuestas de governanza:

```
validator placeholder {
  vote(redeemer: Data, voter: Voter, self: Transaction) {
    todo @"vote logic goes here"
  }
  // Otros handles...
}
```

Propósitos : `vote`

Scripts que validan **votar** en propuestas de governanza:

```
validator placeholder {
  vote(redeemer: Data, voter: Voter, self: Transaction) {
    todo @"vote logic goes here"
  }
  // Otros handles...
}
```

- `voter: Voter` : Tipo e identificador único del votante (hash del script)

Propósitos : `vote`

Scripts que validan **votar** en propuestas de governanza:

```
validator placeholder {
  vote(redeemer: Data, voter: Voter, self: Transaction) {
    todo @"vote logic goes here"
  }
  // Otros handles...
}
```

- `voter: Voter` : Tipo e identificador único del votante (hash del script)

Un script puede votar como:

Miembro del comité constitucional		

Propósitos : `vote`

Scripts que validan **votar** en propuestas de governanza:

```
validator placeholder {
  vote(redeemer: Data, voter: Voter, self: Transaction) {
    todo @"vote logic goes here"
  }
  // Otros handles...
}
```

- `voter: Voter` : Tipo e identificador único del votante (hash del script)

Un script puede votar como:

Miembro del comité constitucional	DRep	

Propósitos : `vote`

Scripts que validan **votar** en propuestas de governanza:

```
validator placeholder {
  vote(redeemer: Data, voter: Voter, self: Transaction) {
    todo @"vote logic goes here"
  }
  // Otros handles...
}
```

- `voter: Voter` : Tipo e identificador único del votante (hash del script)

Un script puede votar como:

Miembro del comité constitucional	DRep	Stake pool

Propósitos : `proposal`

Scripts que validan **crear** propuestas de governanza

```
validator placeholder {
    propose(redeemer: Data, proposal: ProposalProcedure, self: Transaction) {
      todo @"propose logic goes here"
    }
  // Otros handles...
}
```

Propósitos : `proposal`

Scripts que validan **crear** propuestas de governanza

```
validator placeholder {
    propose(redeemer: Data, proposal: ProposalProcedure, self: Transaction) {
      todo @"propose logic goes here"
    }
  // Otros handles...
}
```

- `proposal: ProposalProcedure` : Información sobre la propuesta (depósito, acción, etc.)

Propósitos : `proposal`

Scripts que validan **crear** propuestas de governanza

```
validator placeholder {
    propose(redeemer: Data, proposal: ProposalProcedure, self: Transaction) {
      todo @"propose logic goes here"
    }
  // Otros handles...
}
```

- `proposal: ProposalProcedure` : Información sobre la propuesta (depósito, acción, etc.)

> **Info**
>
> En el caso de actualizar los parámetros o retirar de la tesorería, se ejecutan los **Constitution Guardrails**. Estos, verifican que la propuesta no viole la constitución.

Propósitos : Validador completamente definido

Como tenemos todos esto handles, para definir un validador completamente y asegurarnos de que se comporte correctamente en todos los casos, deberíamos definirlos a todos:

Propósitos : Validador completamente definido

Como tenemos todos esto handles, para definir un validador completamente y asegurarnos de que se comporte correctamente en todos los casos, deberíamos definirlos a todos:

```
validator placeholder {
  mint(redeemer: Data, policyid: PolicyId, self: Transaction) {
    todo @"mint logic goes here"
  }

  spend( datum: Option<Data>, redeemer: Data, utxo: OutputReference, self: Transaction,) {
    todo @"spend logic goes here"
  }

  withdraw(redeemer: Data, account: Credential, self: Transaction) {
    todo @"withdraw logic goes here"
  }

  publish(redeemer: Data, certificate: Certificate, self: Transaction) {
    todo @"publish logic goes here"
  }

  vote(redeemer: Data, voter: Voter, self: Transaction) {
    todo @"vote logic goes here"
  }

  propose(redeemer: Data, proposal: ProposalProcedure, self: Transaction) {
    todo @"propose logic goes here"
  }
}
```

Propósitos : `else`

- En la mayoría de casos, nuestro portocolo va a tener en cuenta **2 o 3 de estos casos**, por lo que definiríamos el resto como `False` para rechazar la transacción.

Propósitos : `else`

- En la mayoría de casos, nuestro portocolo va a tener en cuenta **2 o 3 de estos casos**, por lo que definiríamos el resto como `False` para rechazar la transacción.
- También existen escenarios en los que podría no ser conveniente la granularidad y protecciones que ofrece Aiken (p. ej., Hydra con un contexto de script distinto)

Propósitos : `else`

- En la mayoría de casos, nuestro portocolo va a tener en cuenta **2 o 3 de estos casos**, por lo que definiríamos el resto como `False` para rechazar la transacción.
- También existen escenarios en los que podría no ser conveniente la granularidad y protecciones que ofrece Aiken (p. ej., Hydra con un contexto de script distinto)

Para estos casos tenemos `else` :

Propósitos : `else`

- En la mayoría de casos, nuestro portocolo va a tener en cuenta **2 o 3 de estos casos**, por lo que definiríamos el resto como `False` para rechazar la transacción.
- También existen escenarios en los que podría no ser conveniente la granularidad y protecciones que ofrece Aiken (p. ej., Hydra con un contexto de script distinto)

Para estos casos tenemos `else` :

```
validator placeholder {
  mint(redeemer: Data, policyid: PolicyId, self: Transaction) {
    todo @"mint logic goes here"
  }

  else(_ctx: ScriptContext) {
    fail
  }
}
```

Propósitos : `else` y `ScriptContext`

`else` nos da acceso a `ScriptContext` que contiene toda la Información de la transacción:

```
ScriptContext { transaction: Transaction, redeemer: Redeemer, info: ScriptInfo }
```

Propósitos : `else` y `ScriptContext`

`else` nos da acceso a `ScriptContext` que contiene toda la Información de la transacción:

```
ScriptContext { transaction: Transaction, redeemer: Redeemer, info: ScriptInfo }
```

- `info: ScriptInfo` : Contiene Información relacionada con los propósitos.

Propósitos : `else` y `ScriptContext`

`else` nos da acceso a `ScriptContext` que contiene toda la Información de la transacción:

```
ScriptContext { transaction: Transaction, redeemer: Redeemer, info: ScriptInfo }
```

- `info: ScriptInfo` : Contiene Información relacionada con los propósitos.

> **Info**
>
> `ScriptContext` otorga mayor flexibilidad a cambio de que el desarrollador se responsablize de verificar los propósitos del script y obtener el Redeemer y/o Datum.

INPUT | OUTPUT

Preguntas?