

A concurrent DEX for Cardano

Bsc thesis

Peter Brühwiler

16.12.21

The Concurrency Issue



Eric Wall @ercwl · Sep 4

Someone help this man

...

binbal @binbal24 · Sep 4

I hate to be dramatic, but public testnet UX at @MinswapDEX has been horrible. The UTXO model is simply unusable in #defi systems due to the #concurrencyissue. Could someone plz provide some clarification as to how and when this will be resolved?

[Show this thread](#)

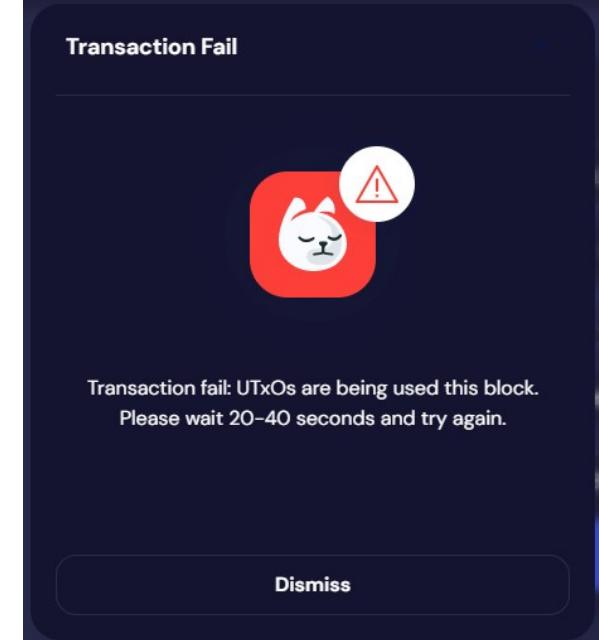
28

34

239



Tip



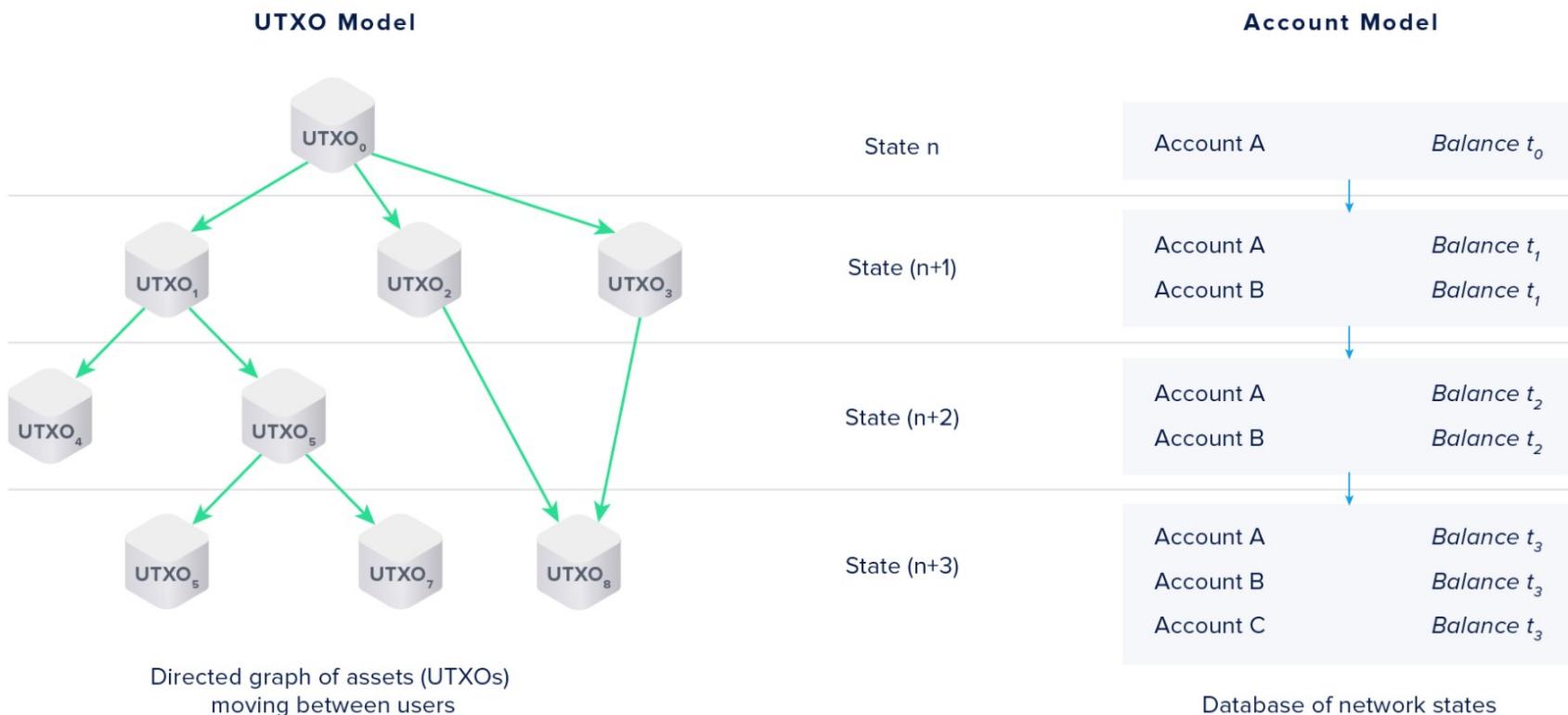
Minswap blog post:

“.. It’s not a fundamental flaw, but is simply a design challenge that must be addressed.”

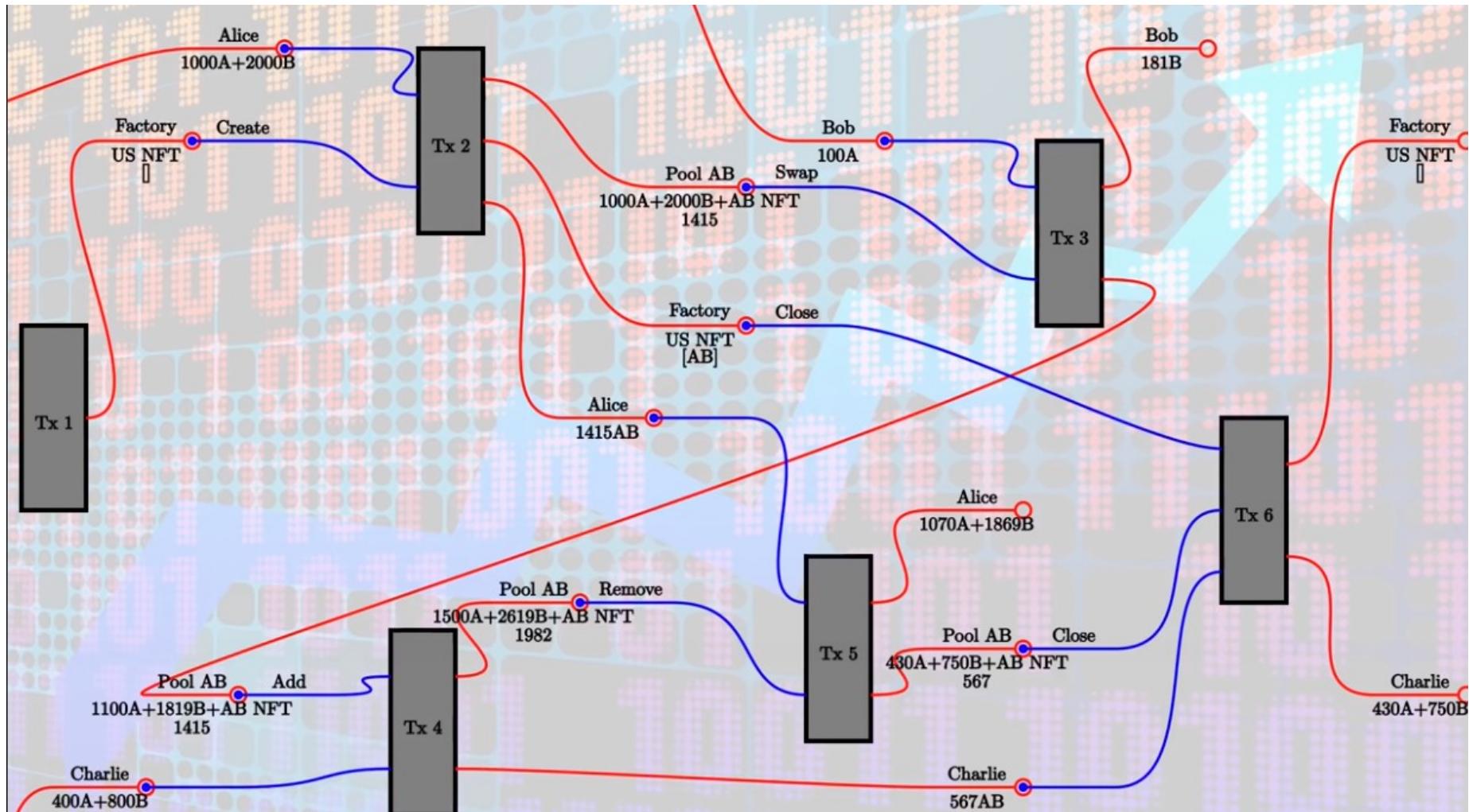
The UTxO model

HORIZEN ACADEMY

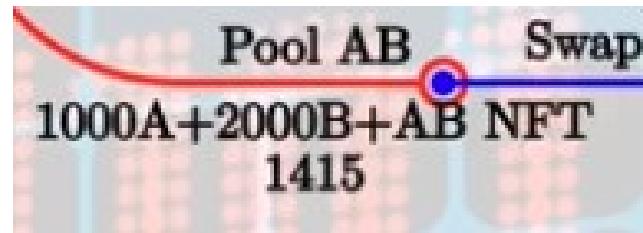
RECORDING THE STATE OF THE SYSTEM



Simple AMM style DEX



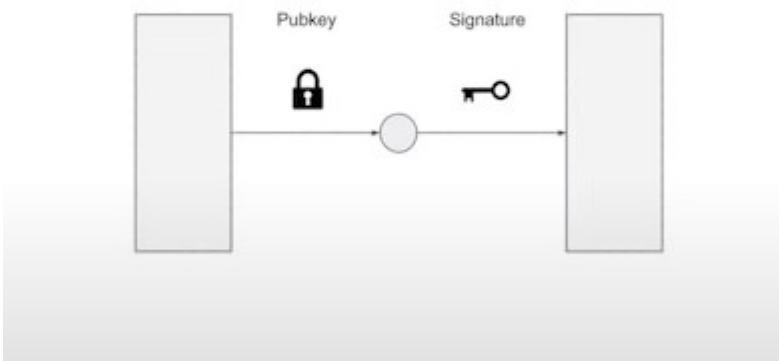
The extended UTxO



- Datum to describe a state
- Redeemer to describe an action

Generalized

UTXO



EUTXO



Simple validator

```
mkValidator :: () -> Integer -> ScriptContext -> Bool
mkValidator _ r _ = traceIfFalse "wrong redeemer" $ r == 42
```

```
data ScriptContext
```

Constructors

```
ScriptContext
```

```
    scriptContextTxInfo :: TxInfo
```

```
    scriptContextPurpose :: ScriptPurpose
```

TxInfo

```
txInfoInputs :: [TxInInfo]
```

```
txInfoOutputs :: [TxOut]
```

```
txInfoFee :: Value
```

```
txInfoMint :: Value
```

```
txInfoDCert :: [DCert]
```

```
txInfoWdrl :: [(StakingCredential, Integer)]
```

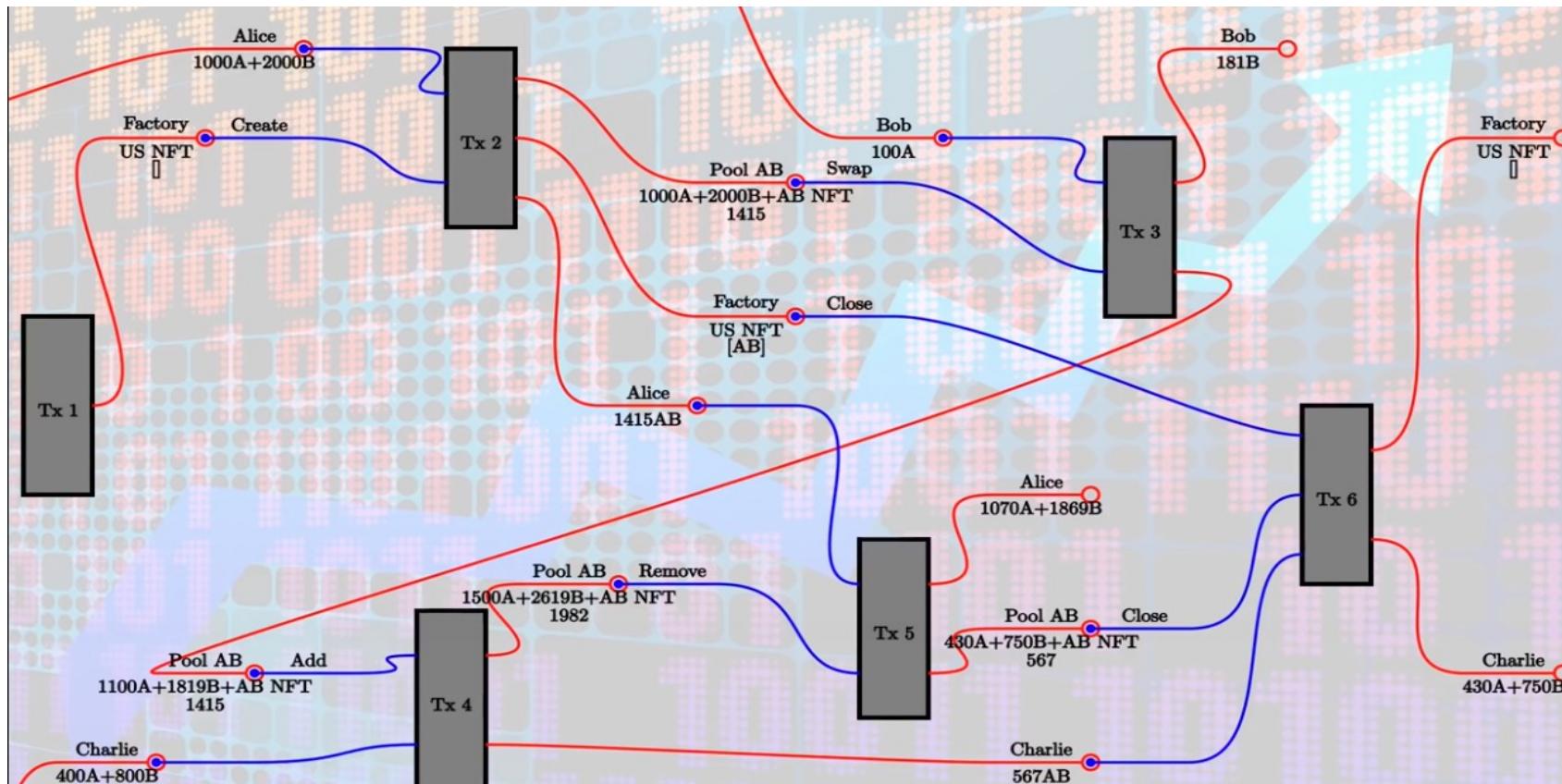
```
txInfoValidRange :: POSIXTimeRange
```

```
txInfoSignatories :: [PubKeyHash]
```

```
txInfoData :: [(DatumHash, Datum)]
```

```
txInfoId :: TxId
```

So, Bob can Swap 100A for 181B..



But only once per block ..

Order book

Open orders			
Buy			Sell
Amount	Price per Token	Total	
1,000	0.57 ₣	570 ₣	SELL
1,000	0.551 ₣	551 ₣	SELL
100	0.55 ₣	55 ₣	SELL
1,000	0.55 ₣	550 ₣	SELL
100	0.54 ₣	54 ₣	SELL
50	0.53 ₣	26.5 ₣	SELL
100	0.53 ₣	53 ₣	SELL
500	0.521 ₣	260.5 ₣	SELL
100	0.52 ₣	52 ₣	SELL
100	0.5 ₣	50 ₣	SELL
500	0.5 ₣	250 ₣	SELL
10	0.45 ₣	4.5 ₣	SELL
50	0.45 ₣	22.5 ₣	SELL
50	0.45 ₣	22.5 ₣	SELL
			BUY
100	0.65 ₣	65 ₣	BUY
1,000	0.67 ₣	670 ₣	BUY
100	0.679 ₣	67.9 ₣	BUY
100	0.68 ₣	68 ₣	BUY
1,000	0.69 ₣	690 ₣	BUY
500	0.7 ₣	350 ₣	BUY
1,000	0.7 ₣	700 ₣	BUY
10	0.75 ₣	7.5 ₣	BUY
10	0.75 ₣	7.5 ₣	BUY
10	0.75 ₣	7.5 ₣	BUY
250	0.75 ₣	187.5 ₣	BUY
1,000	0.75 ₣	750 ₣	BUY
1,000	0.75 ₣	750 ₣	BUY
1,000	0.75 ₣	750 ₣	BUY

```
{-# INLINABLE mkOrderValidator #-}

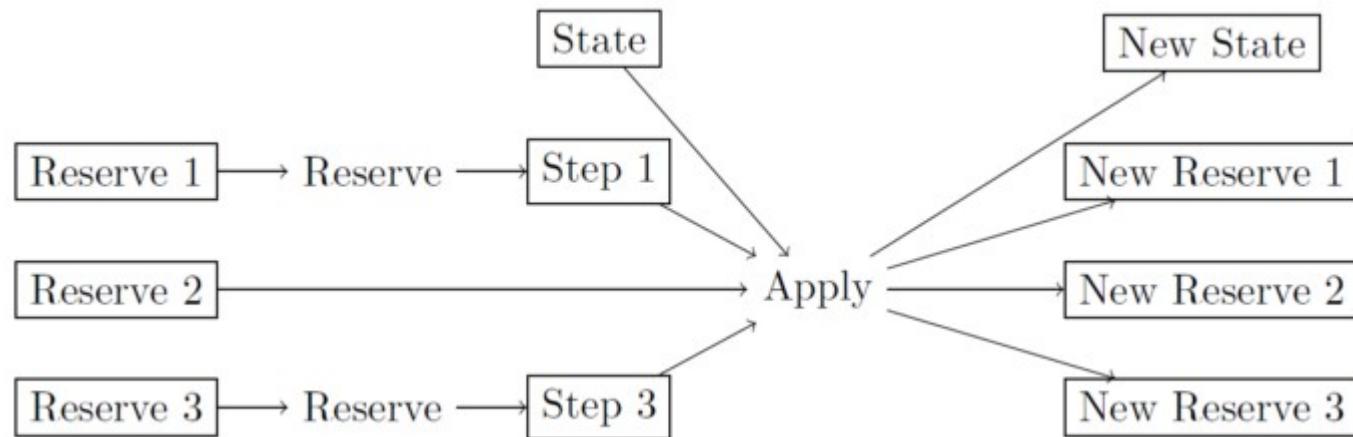
mkOrderValidator :: OrderDatum -> OrderAction -> ScriptContext -> Bool
mkOrderValidator od redeemer ctx = case redeemer of
  CancelOrder ->
    traceIfFalse "signature does not match creator in datum" checkSig
  FullMatch ->
    traceIfFalse "expected creator to get all of what she ordered" correctFull
    && traceIfFalse "only matches of pairs of orders allowed" twoParties
```

```
data Order = Order
  { oCreator      :: !PubKeyHash
  , oBuyCurrency  :: !CurrencySymbol
  , oBuyToken     :: !TokenName
  , oBuyAmount    :: !Integer
  }
deriving (Generic, ToJSON, FromJSON)
```

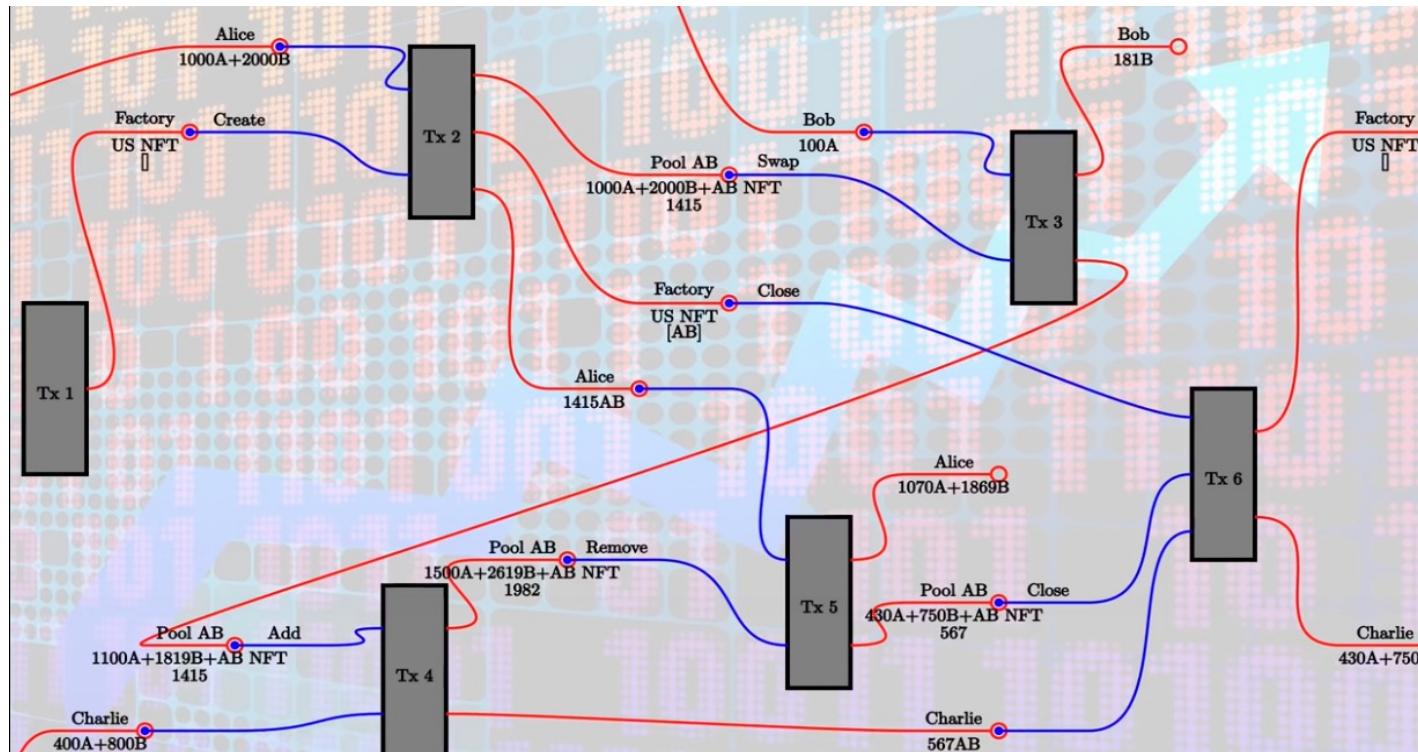
Back to the AMM model, but with ..

.. multiple state UtxOs

.. a batching mechanism



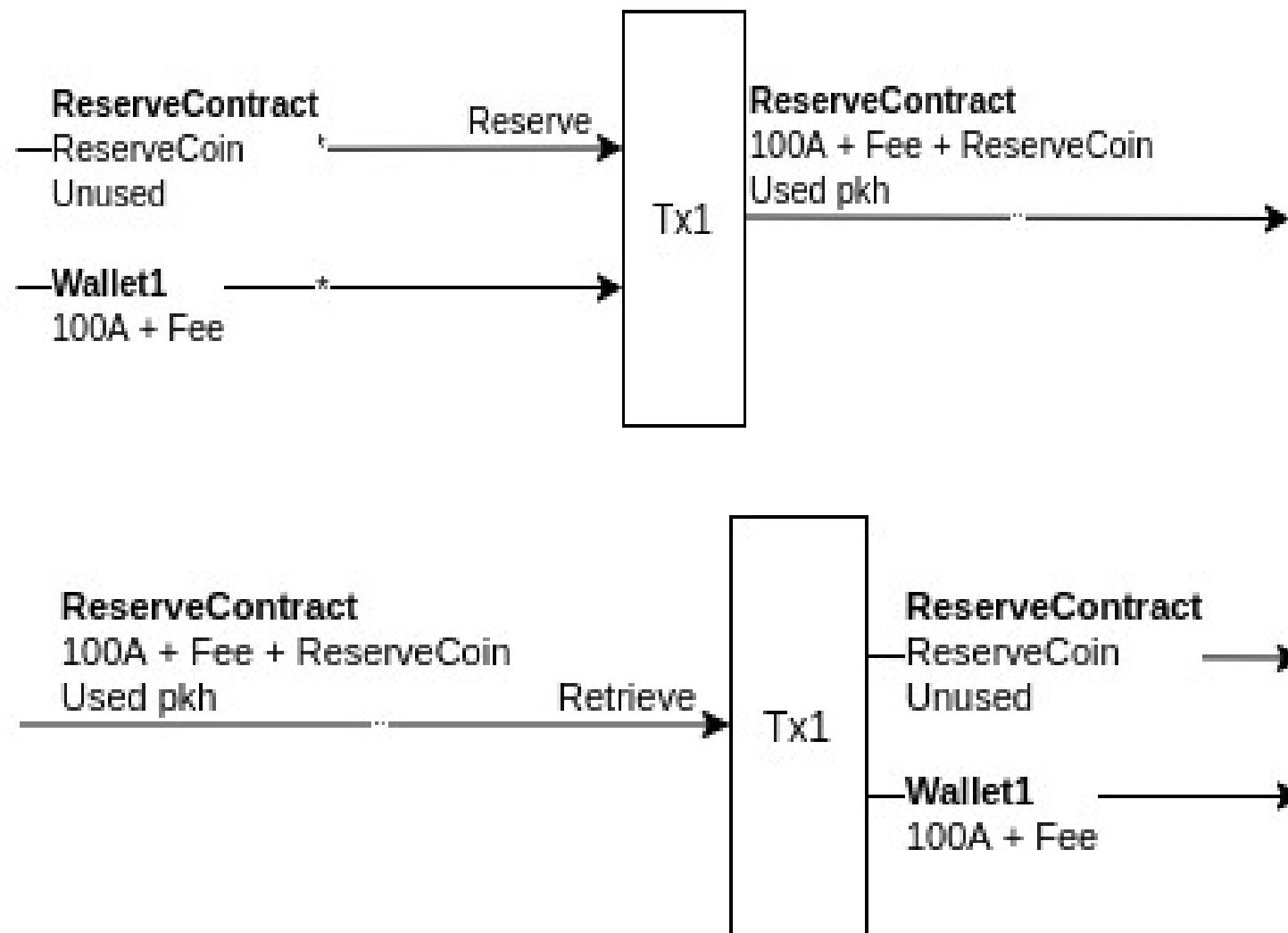
So one more time..



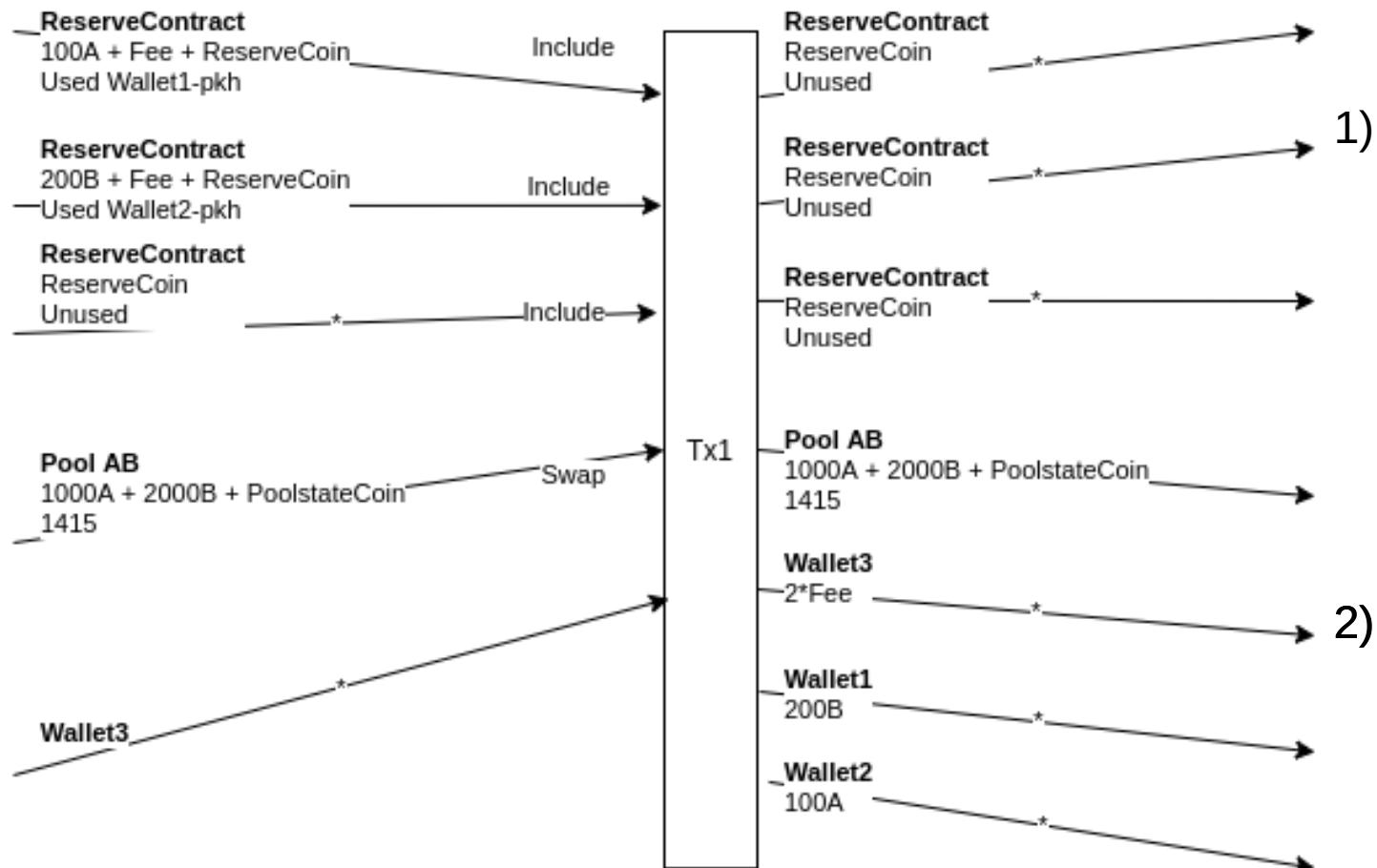
With the following changes:

- Tx2: Create n Reserve UTxOs
- Let users reserve and retrieve Reserve UtxOs
- Tx3: Include n Reserve UTxOs
- Tx6: Burn n Reserve UTxOs

Reserve and Retrieve



The Swap transaction



1) Nr of Reserve UTxOs variable

2) Wallet3 gets all the 'leftovers'

Reserve validation logic

```
mkReserveValidator :: Coin U -> Coin ReserveState -> Coin PoolState -> ReserveDatum -> ReserveRedeemer -> ScriptContext -> Bool
mkReserveValidator u rs ps reserveDatum reserveRedeemer ctx =
  case reserveDatum of
    (Unused lp n) -> case reserveRedeemer of
      Reserve ->
        outputContainsFee      &&
        outputContainsReserveToken &&
        correctLiquidityPool      &&
        datumIsUnchanged
      Include ->
        poolStateCoinIncluded
      Destroy ->
        uniswapCoinIncluded
    (Used lp pkh n) -> case reserveRedeemer of
      Retrieve ->
        (Validation.txSignedBy info pkh) &&
        outputContainsReserveToken      &&
        outputStateUnused outputDatum      &&
        correctLiquidityPool      &&
        datumIsUnchanged
      Include ->
        poolStateCoinIncluded
```

- ‘Include’ validation is delegated to the Swap validator
- Coins needed to identify the correct UTxO

Swap validation

$x\%$ of total value of A as input \rightarrow $x\%$ of total value of B as output

```
amountForSwapInput :: Integer -> (Amount A, Amount B) -> (Amount A, Amount B)
amountForSwapInput r (a,b)
| a * r > b = (a - b / r, 0)
| otherwise = (0, b - a * r)
```

Only the ‘excess’ part of the currency with the higher value is put into the swap

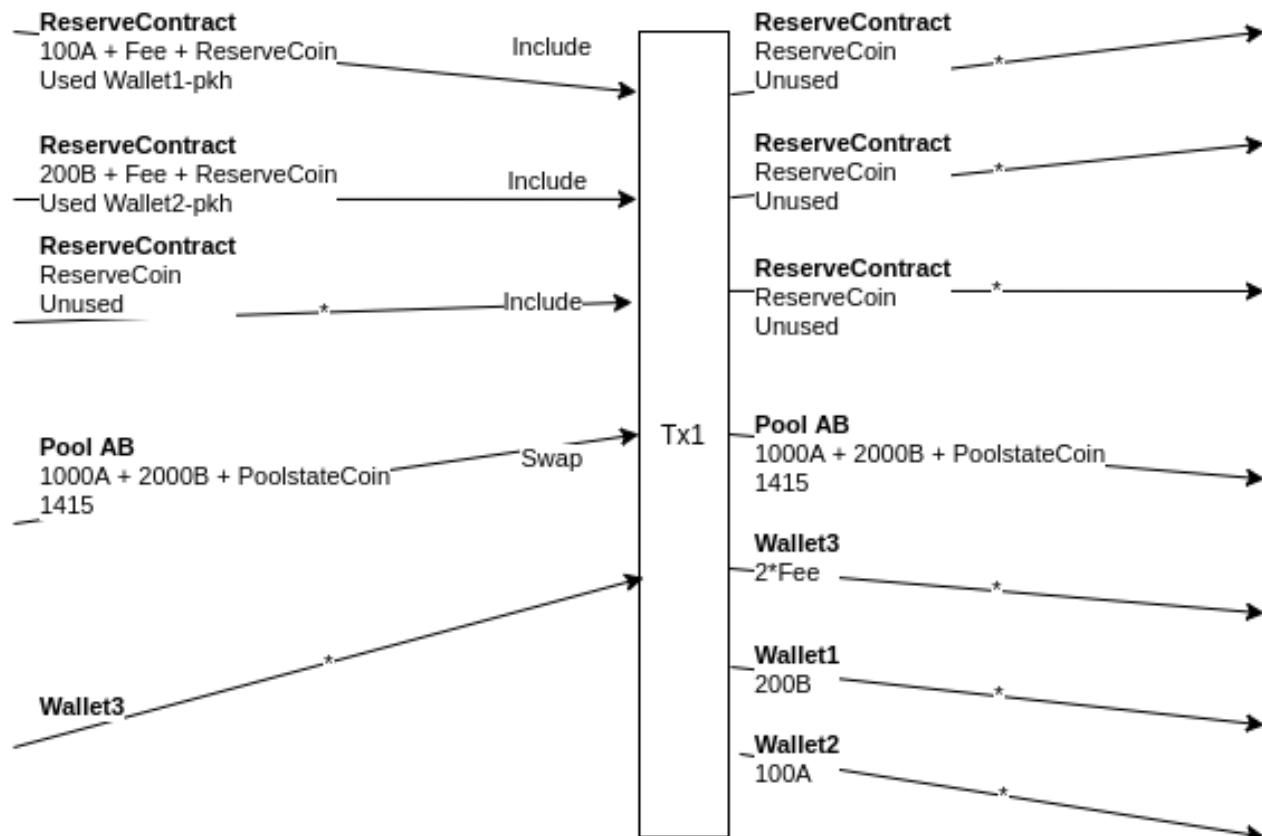
```
totalValA' :: Amount A
totalValA'
-- if B was put into swap:
| newB - oldB > 0 = (oldA - newA) + totalValA
| otherwise         = totalValB `divide` ratio

totalValB' :: Amount B
totalValB'
| newA - oldA > 0 = (oldB - newB) + totalValB
| otherwise         = totalValA * ratio
```

oldA, newA, oldB, newB are the values in the liquidity pool before and after the swap

Swap is deterministic, no arbitrary order of execution

So, given these inputs, the liquidity pool stays unchanged



OffChain code

.. basically spends and creates UtxOs without breaking the rules of the validator

Building the swap transaction

```
tx = Constraints.mustSpendScriptOutput oref (Redeemer $ PlutusTx.toBuiltinData Swap) <>
    mconcat (scriptSpending $ map fst orefsAndOsUsed) <>
    mconcat (scriptSpending $ map fst orefsAndOsUnused) <>
    Constraints.mustPayToOtherScript (validatorHash usScript) (Datum $ PlutusTx.toBuiltinData (Pool lp liquidity)) val <>
    Constraints.mustPayToOtherScript (validatorHash $ reserveValidator us) (Datum $ PlutusTx.toBuiltinData (Unused lp)) (unitValue rsc) <>
    Constraints.mustPayToOtherScript (validatorHash $ reserveValidator us) (Datum $ PlutusTx.toBuiltinData (Unused lp)) (unitValue rsc) <>
    Constraints.mustPayToOtherScript (validatorHash $ reserveValidator us) (Datum $ PlutusTx.toBuiltinData (Unused lp)) (unitValue rsc) <>
    mconcat (pubKeyConstraints clientAmounts)
```

Demonstration

The screenshot shows a Linux desktop environment with a dark theme. Two terminal windows are open side-by-side. The left terminal window has a red header bar and displays the command `cabal run uniswap-pab`. The right terminal window has a grey header bar and displays three commands: `cabal run uniswap-client -- 1`, `cabal run uniswap-client -- 2`, and `cabal run uniswap-client -- 3`. The bottom status bar shows the session identifier [0] 0:bash* and the date/time "pb" 09:11 15-Dec-21.

```
[nix-shell:~/thesis/2021.bsc.peter.bruehwiler/perdex]$ cabal run uniswap-pab
[nix-shell:~/thesis/2021.bsc.peter.bruehwiler/perdex]$ cabal run uniswap-client -- 1
[nix-shell:~/thesis/2021.bsc.peter.bruehwiler/perdex]$ cabal run uniswap-client -- 2
[nix-shell:~/thesis/2021.bsc.peter.bruehwiler/perdex]$ cabal run uniswap-client -- 3
[0] 0:bash* "pb" 09:11 15-Dec-21
```

How decentralized is the DEX?

- Execution of the Swap happens OffChain
- Large OffChain part is in the nature of Cardano, a validator can not jump into action on it's own.
- But: The Validator guarantees the outcome
- So, six smileys for decentralization → → →

Sundaeswap

	Uniswap	Programmable Orderbook	Open Batching	Escrow Tokens	Mixed Escrows	Governed Scoopers
Scalability	💀	😊	😊	😐	😐	😊
Contention	💀	😊	😊	😐	😐	😁
MEV	😁	💀	💀	😊	😊	😊
Decentralization	😁	😁	😁	😁	😁	😊
Denial of Service	💀	😁	😐	💀	😐	😊
Volume Independence	😐	💀	😊	😐	😊	😊
Development Effort	😊	💀	😊	😊	😐	😊

The ‘Escrow Tokens’ approach “crashes head first into the sizing limits of the Cardano blockchain.” getting minimal contention “depends on hundreds of escrow tokens, and in practice, our first implementation of this was hitting the Cardano protocol limits with 5 or 6 escrow tokens, before even adding in features like governance.”

Finally, they went for the ‘Governed Scoopers’ model: “by aligning incentives and creating systems of self-governance, you can scale a system by building trust into the protocol.”

Batching ‘Add to Liquitidy Pool’

Cpu: 1% | Mem: 9% * Swap: 0% | N/A - N/A Sat Dec 11 2021 15:18:00 | Glasgow Airport: 6C | Linux 5.11.0-41-gen

bsc-thesis2 - Online LaTeX Edit | Google Übersetzer | Live - Diverse - Langlauf - R | Transaction d5e144cb87b3ff990451e23e74451cf6536bb2a006cd3444319ad55ca993a03e | +

Simplenote | @UniBE chat | cardano | haskell-stuff | vim | cheatsheets | mail | coc-vim + has... | linux setup | Vim/Neovim | Improving Vim... | GitHub | Understanding... | Iagon's solutio... | Vendere/Oncha...

 Cardanoscan (testnet)

All Filters | Search transaction, address, block, epoch.slot, pool, stakeKey, policyId.assetName, finger | Search

Home | Blockchain | Metadata | Tokens | Pools | Certificates | Testnet

UTXOs | Contracts (4) | Collateral (1) | TokenMint (1)

FROM ADDRESSES (INPUTS)

Address	Amount
addr_test1qqqs4hg2mded6m3r7zmq8nlcnrtrkn9qzazak6nlrhgngqww23zcmfspv5ch9hlt40rznmy8w7dydglnz954k2g7ysqnthgw4 0dda4a1ab83ef437cc7601eed03497e45d805558431b28efda29bb56c7e839be	22.0 
	 assetiv..5ca 1
addr_test1wrsexavz37208qda7mwwu4k7hcpg26cz0ce86f5e9kul3hqzlh22t 37e503b744eb07e9a0be4c3a44c6f80229dec27215634d6741b1096097fed63b	4.5 
	 MINT 2,155,461.500  RBERRY 1,000,000.000
addr_test1wrsexavz37208qda7mwwu4k7hcpg26cz0ce86f5e9kul3hqzlh22t 8744b56cb6a65ed3c65bc5814850e8335dfcc1bdec32c2249cd7aaaf3f46a75	4.5 
	 MINT 1,500,000.000  RBERRY 695,906.653
addr_test1wp9m8xkpt2tmy7madqlldspgzugug8f2p3pwhz589cq75685slenwf4 f3d6afe37f2d40f4e3df2c277e47309319446fcc9d9755b6686dc5cb67966a7	2.0 
	 MINT 593,307,439.586  RBERRY 275,257,729.928  assetly..cgw 1
Total Input	33.0 

TO ADDRESSES (OUTPUTS)

Fees 1. 388501 ADA

Address	Amount
addr_test1wp9m8xkpt2tmy7madqlldspgzugug8f2p3pwhz589cq75685slenwf4	2.0 

bsc-thesis - Online LaTeX Edit | Google Übersetzer | Live - Diverse - Langlauf - R | Transaction d5e144cb87b3ff990451e23e74451cf6536bb2a006cd3444319ad55ca993a03e | +

Simplenote @UniBE chat cardano haskell-stuff vim cheatsheets mail coc-vim + has... linux setup Vim/Neovim ... Improving Vim... GitHub Understanding... Iagon's solution Vendere/Oncha...

Cardanoscan (testnet)

All Filters Search transaction, address, block, epoch.slot, pool, stakeKey, policyId.assetName, finger Search

Total Input 33.0 A

TO ADDRESSES (OUTPUTS) Fees 1.388501 ADA

Address	Amount
addr_test1wp9m8xkpt2tmy7madqlspgzug8f2p3pwhz589cq75685slenwf4	2.0 A MINT 596,962,901.084 RBERRY 276,953,636.581 asset1y...cgw 1
addr_test1wpfvzpa046hkfy65mp4ez6vgjunmytzg0ye0ds7mm26v0g77pj9h	3.611499 A
addr_test1qq7825dp3s9qtzgale2zyrepj4uvqtyekdv23g52n3sllu3uan9jrune5vtfeemqp93a0uljpp8fdd59t0u606p6rqnk38vx	2.0 A asset12...f4h 1,466,547.114
addr_test1qqd47q5qzkv7avnlfef20v9t7m90uwkz0d2fd8cnyaa6carwr700yca9wxn4vk5sv02px0q6ke6uchd6qy8q3zj788wqs0j2ex	2.0 A MINT 2 asset12...f4h 1,020,579.893
addr_test1qqss4hg2mded6m3r7zmq8nlcnrcrkn9qzazak6nlrhgngqww23zcmfspv5ch9ht40rznmzy8w7dydglnz954k2g7ysqnthgw4	22.0 A asset1v...5ca 1
Total Output	31.611499 A

Further improvements

Use order tokens instead of order UtxOs,
because the reserve validator gets executed n times

Steps, described by IOG:

Order submission:

- User submits ‘order’ token to own public key address.
- User sends PubKeyHash to the request script for order notification.

Order processing:

- Batcher inspects the UTXOs at the request script address
- Batcher collects ‘order’ tokens and builds the aggregated transaction.
- users to sign the transaction to spend their funds.

Is the UTxO model worth all the trouble?

- Transactions and fees are deterministic
- Cannot fail, if the inputs are present
- Every token is a native token

```
Map CurrencySymbol (Map TokenName Integer)
```

- Layer 2 solutions easier to implement