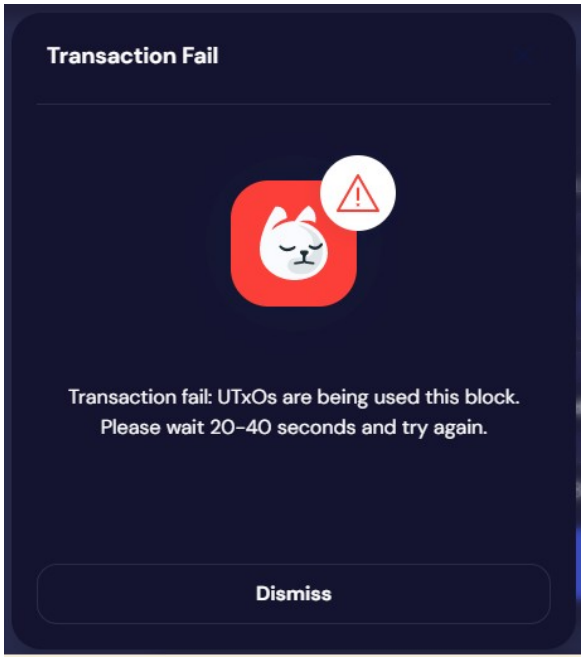
A concurrent DEX for Cardano

Bsc thesis

Peter Brühwiler

16.12.21

The Concurrency Issue



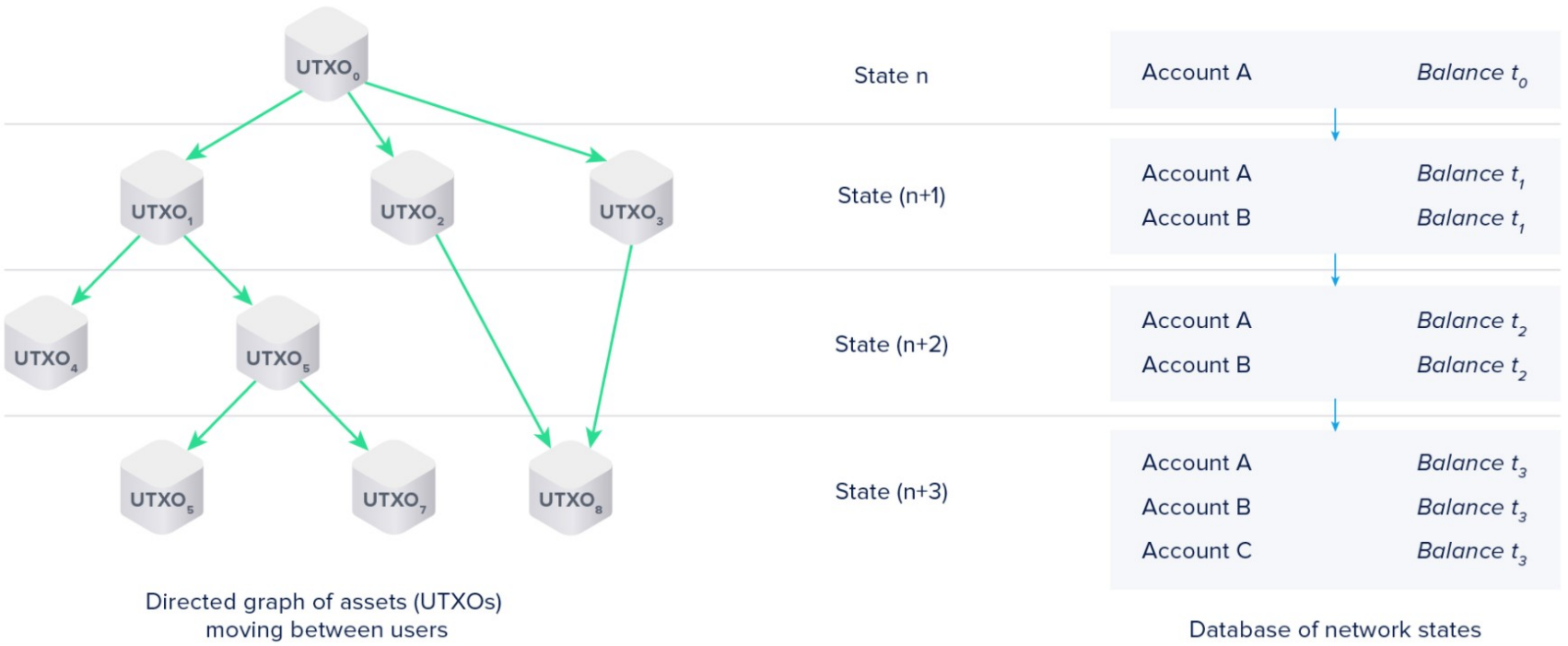
Minswap blog post:

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

The UTxO model

HORIZEN ACADEMY

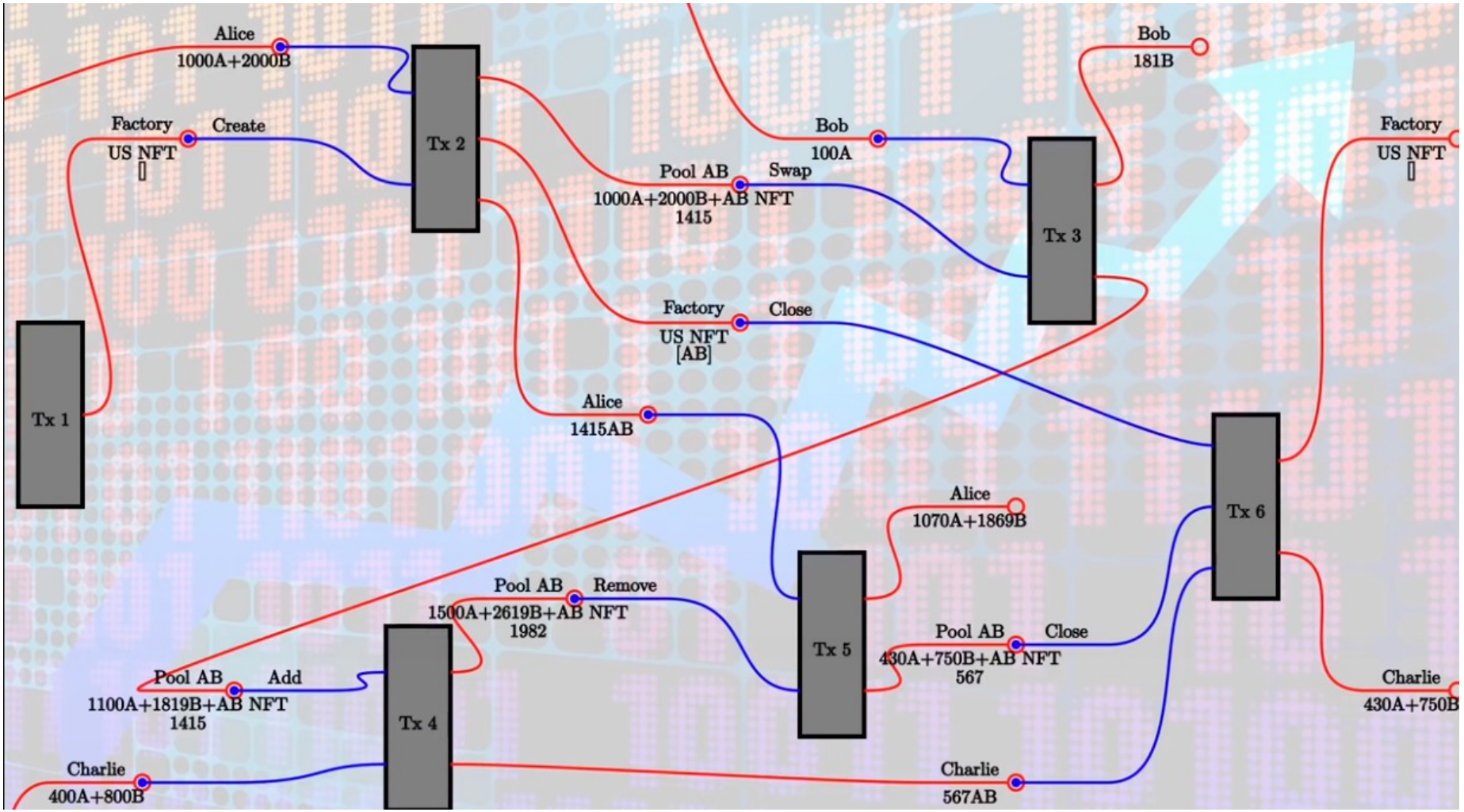
RECORDING THE STATE OF THE SYSTEM



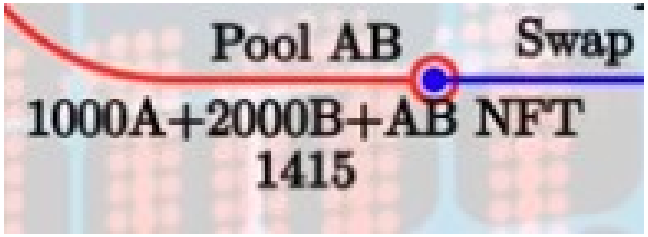
Directed graph of assets (UTXOs) moving between users

Database of network states

Simple AMM style DEX

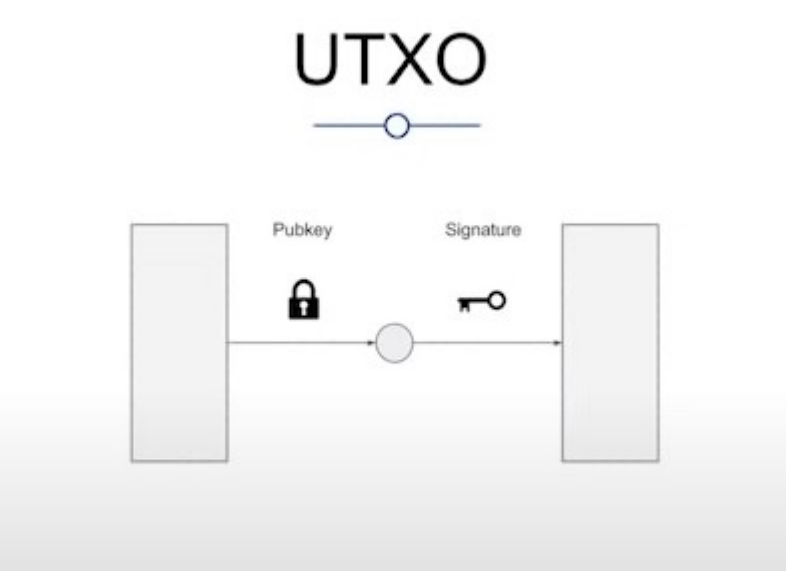


The extended UTxO



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

Generalized



Simple validator

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

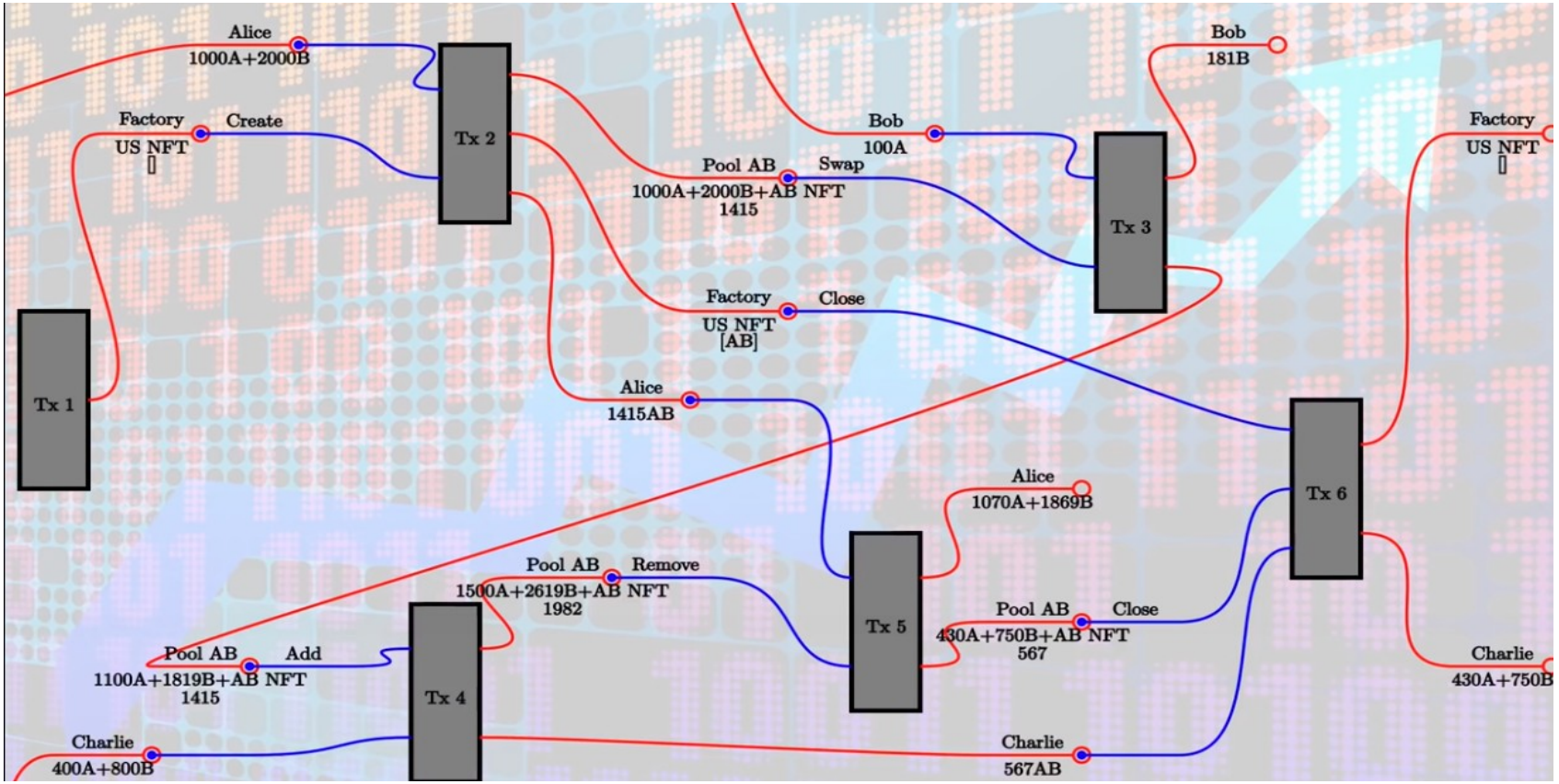
```
data ScriptContext
```

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

Only equally sized orders will be matched

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

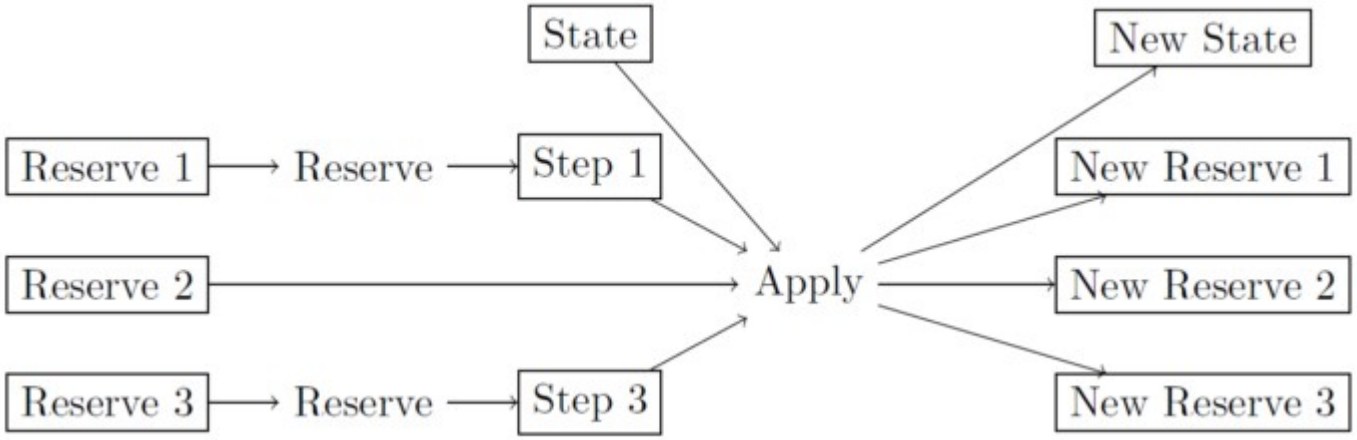
```
{-# INLINABLE mkOrderValidator #-}
mkOrderValidator :: OrderDatum -> OrderAction -> ScriptContext -> Bool
mkOrderValidator od redeemer ctx = case redeemer of
  CancelOrder ->
    traceIfFalse "signature does not match creator in datum" checkSig
  FullMatch ->
    traceIfFalse "expected creator to 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)
```

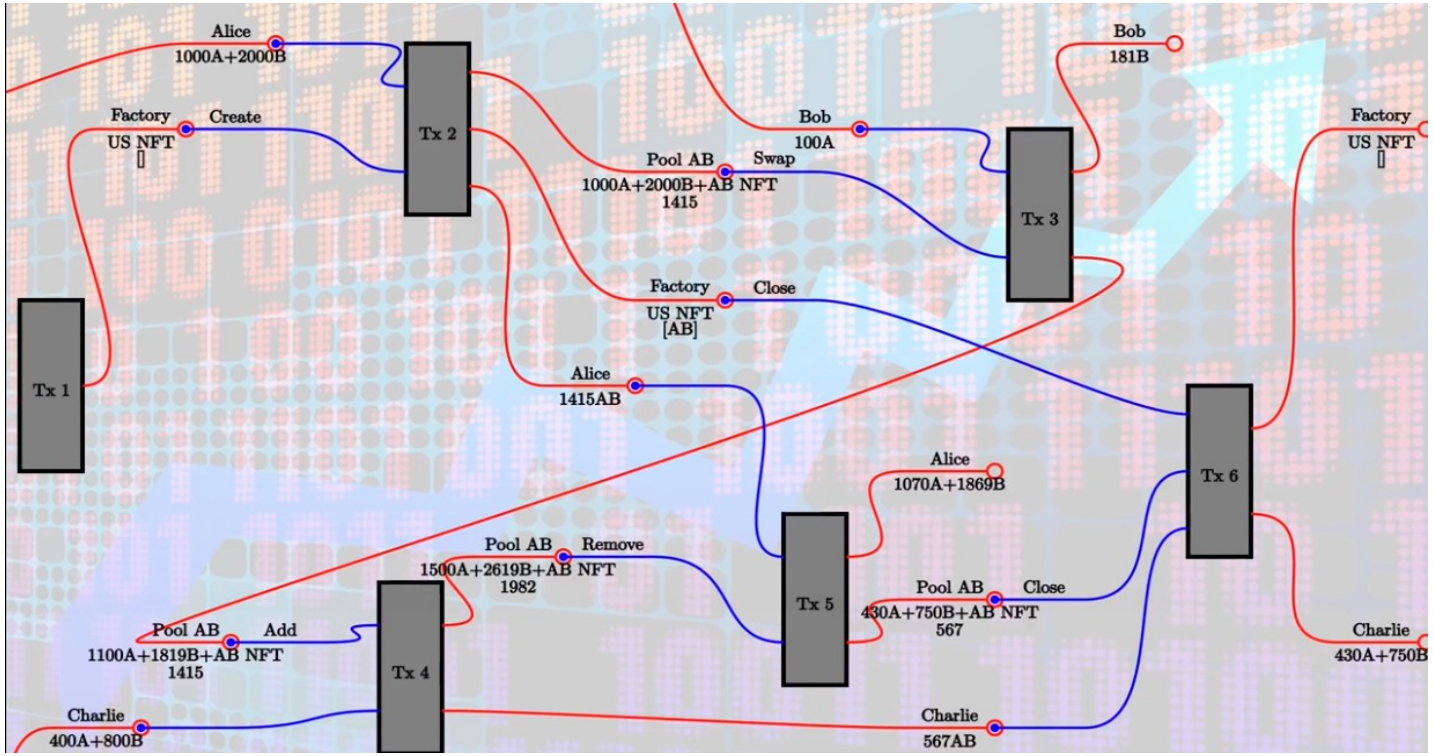
Back to the AMM model, but with ..

.. multiple state UtxOs

.. a batching mechanism



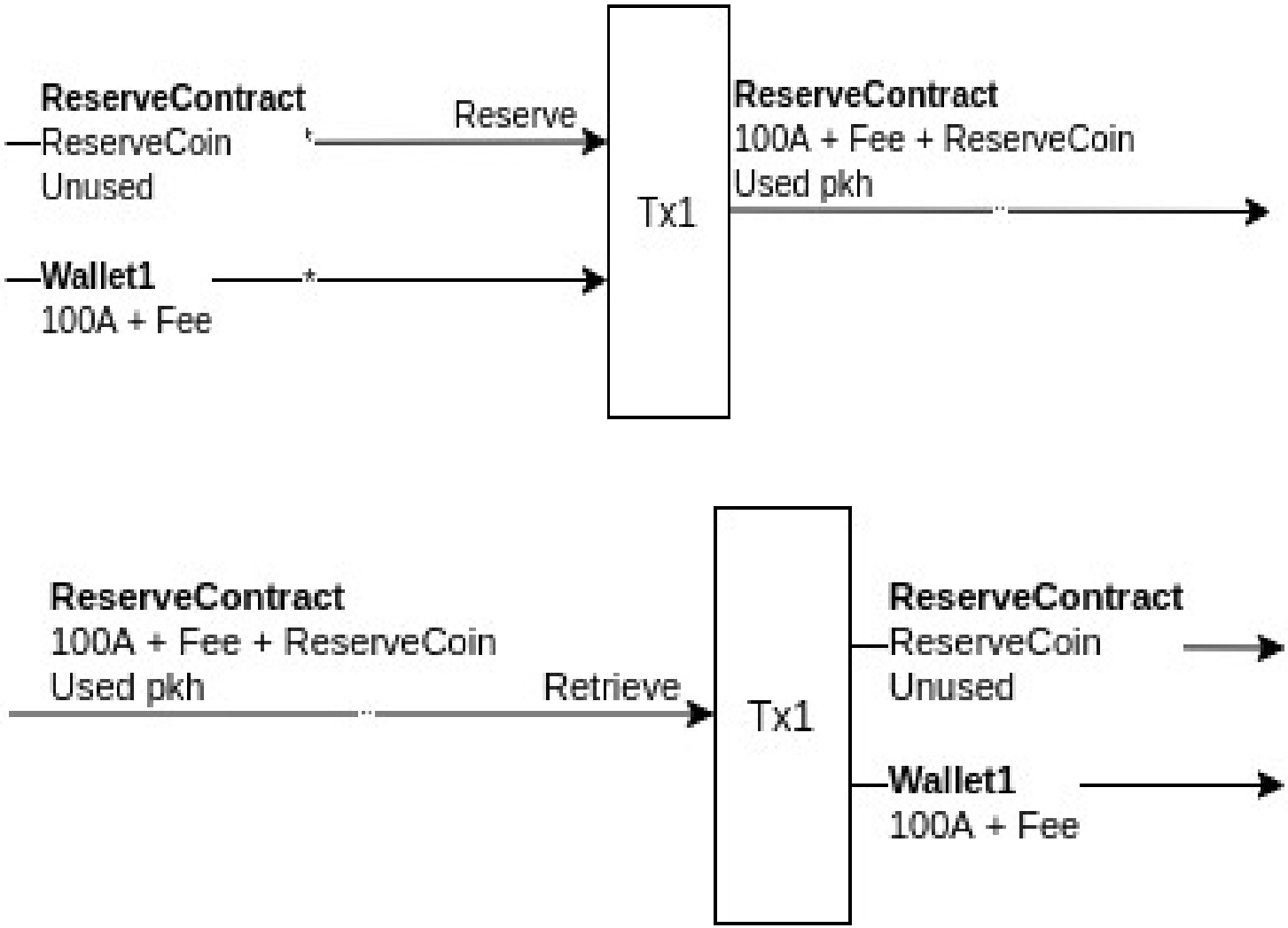
So one more time..



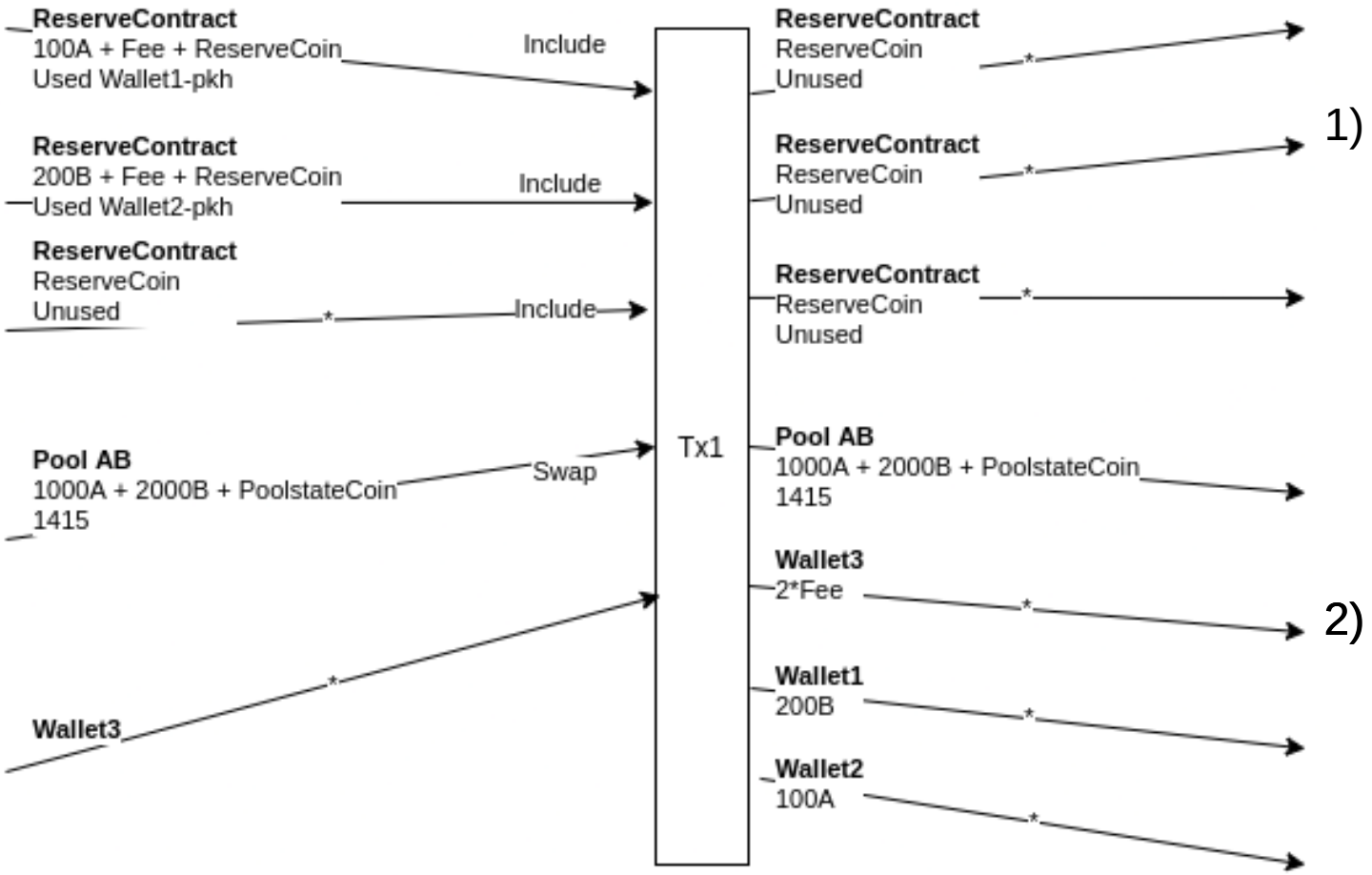
With the following changes:

- 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 → 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

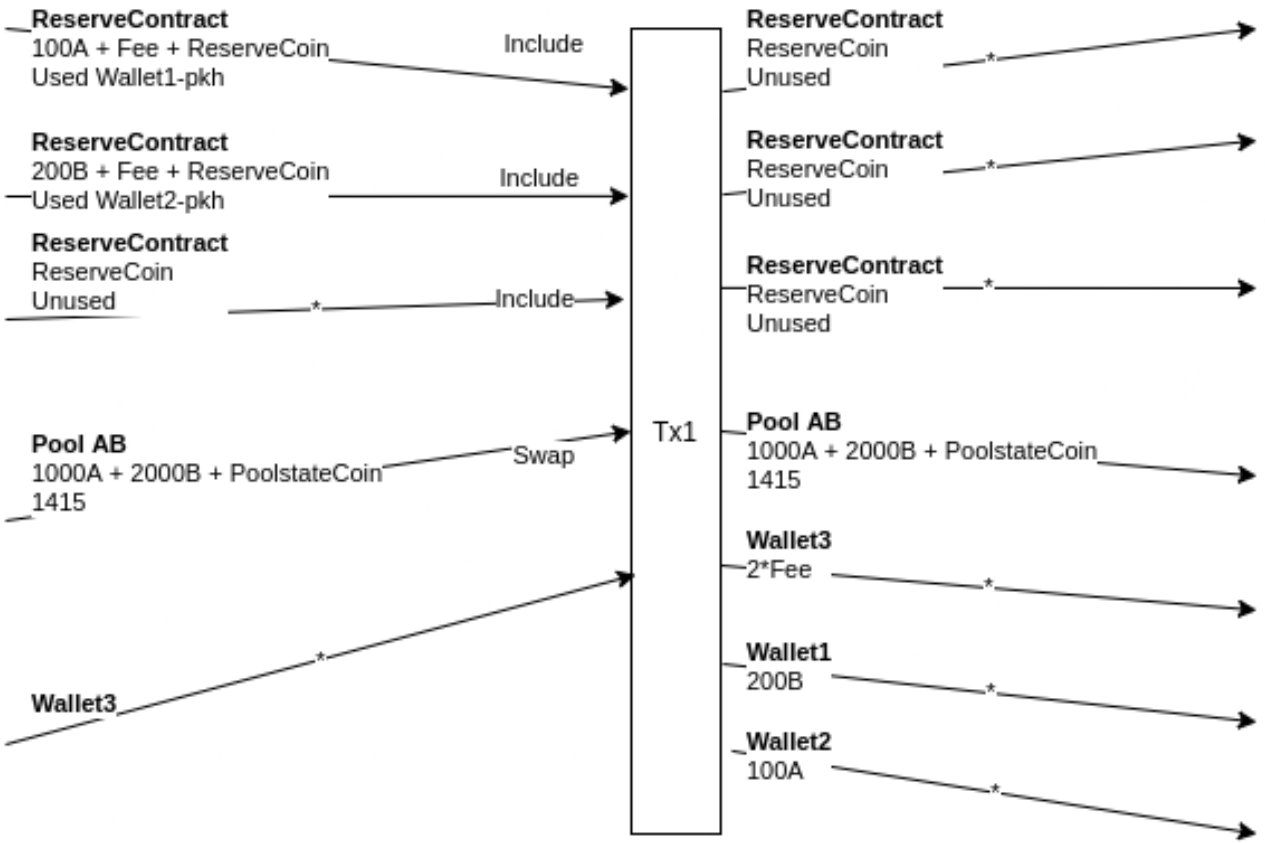
```
totalValA' :: Amount A
totalValA'
    -- if B was put into swap:
    | newB - oldB > 0 = (oldA - newA) + totalValA
    | otherwise       = totalValB `divide` ratio

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



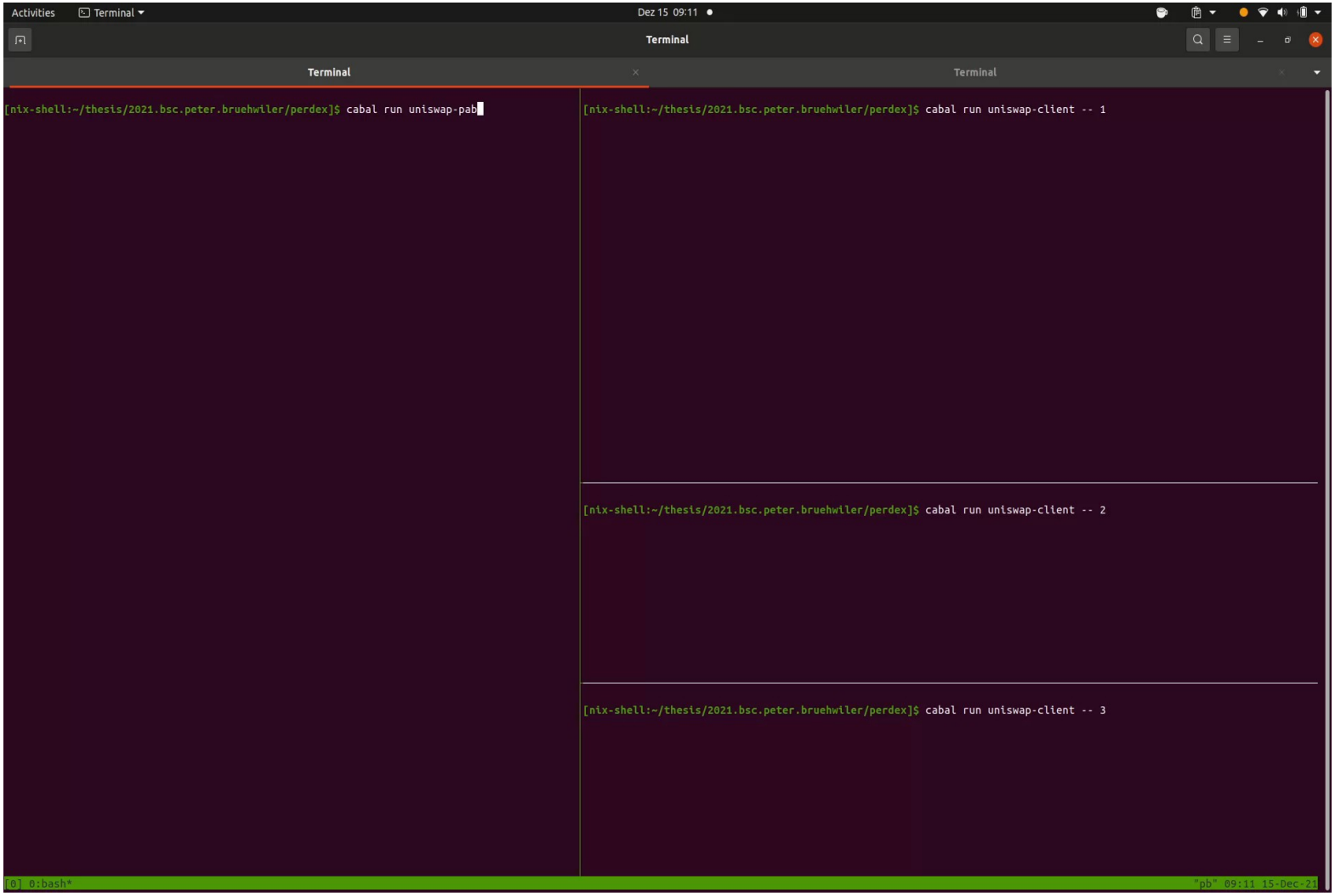
OffChain code

.. 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)
```

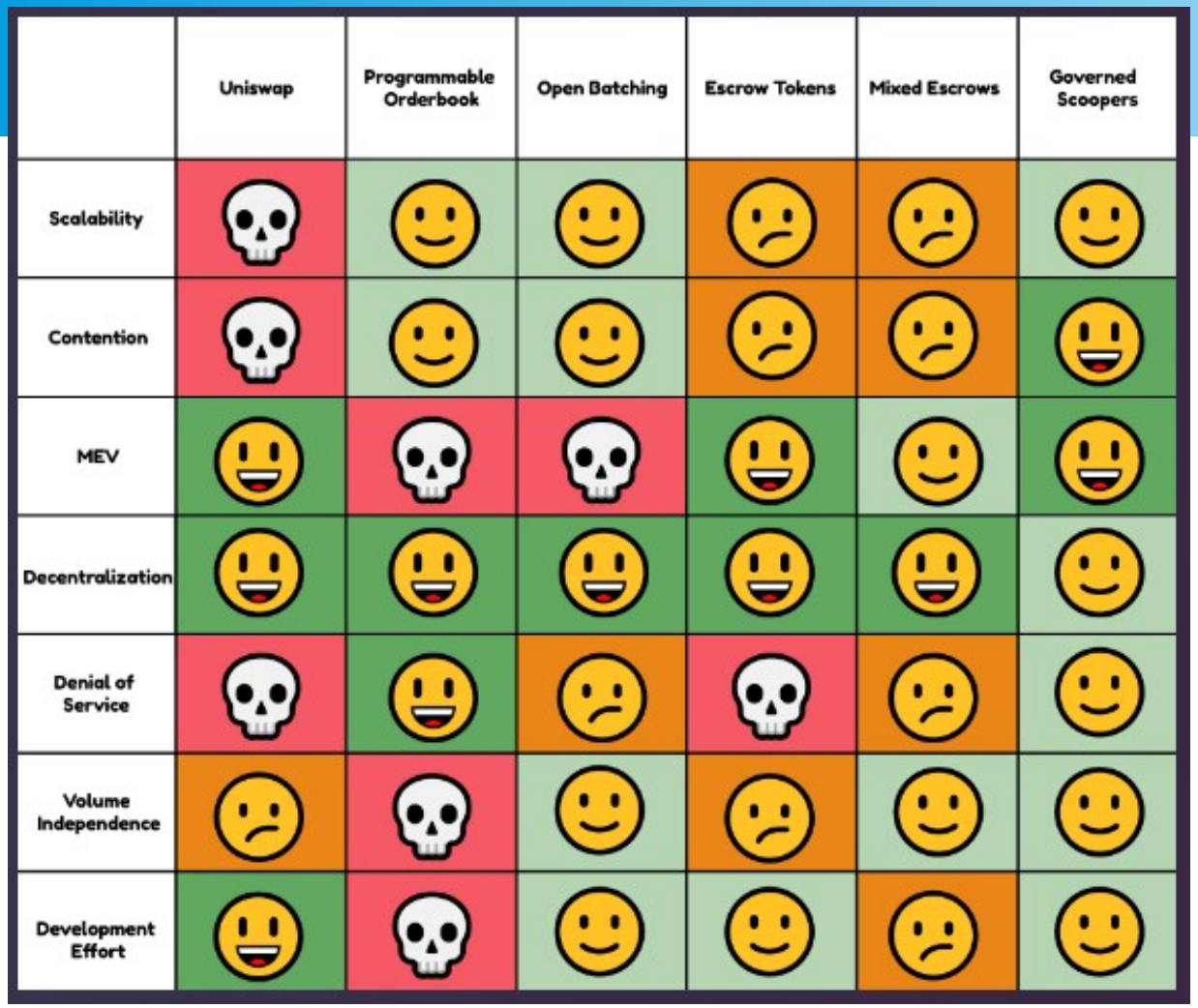
Demonstration



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 → → →

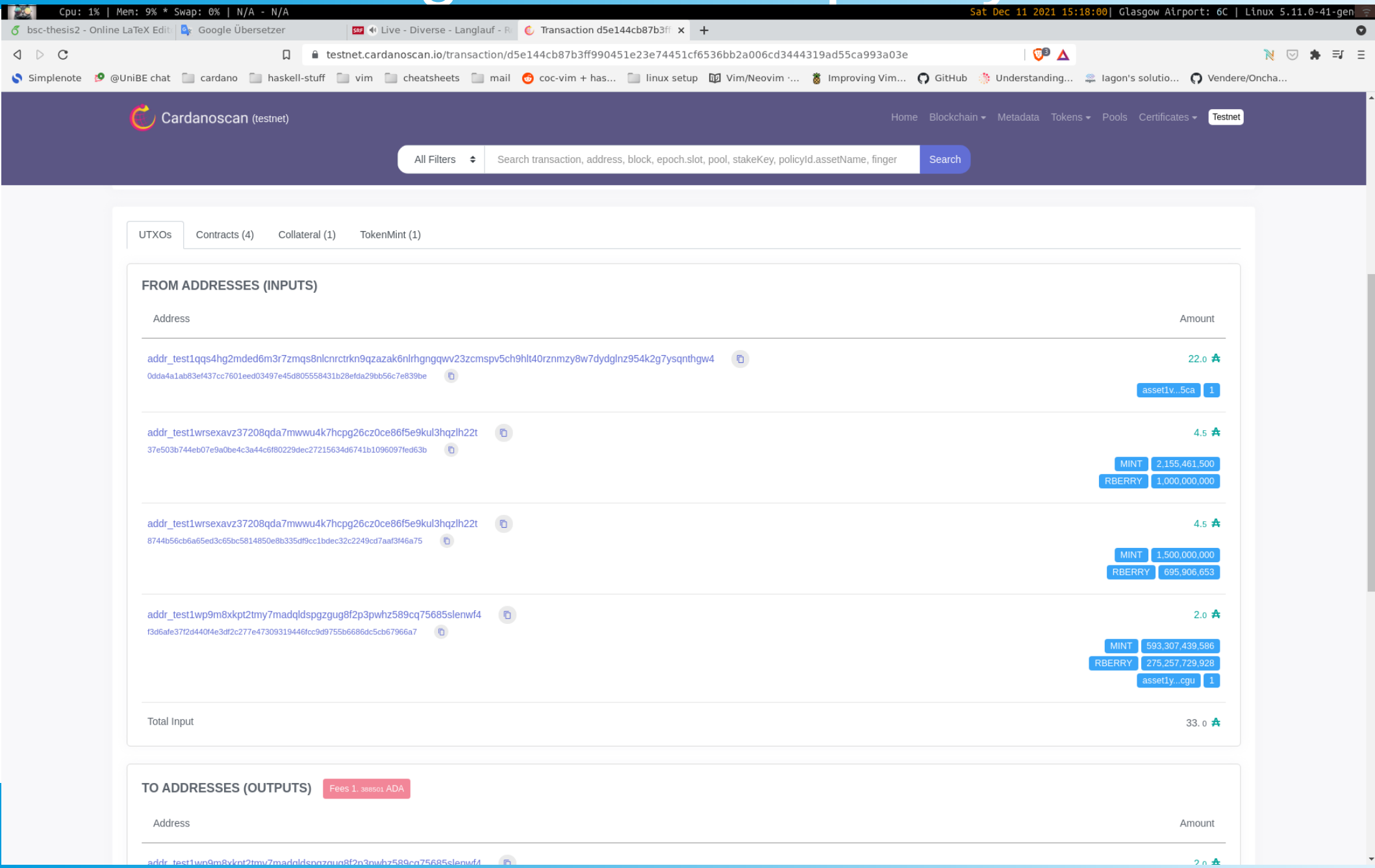
Sundaeswap



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’



Total Input 33.0 ₳

TO ADDRESSES (OUTPUTS)

Fees 1.388501 ADA

Address	Amount
addr_test1wp9m8xkpt2tmy7madqldspgzgug8f2p3pwhz589cq75685slenwf4	2.0 ₳ MINT 596,962,901,084 RBERRY 276,953,636,581 asset1y...cgu 1
addr_test1wpfzvzpa046hkfy65mp4ez6vgjunmytzg0ye0ds7mm26v0g77pj9h	3.611499 ₳
addr_test1qq7825dp3s9qtzgale2zyrepj4uvqtyekdv23g52n3szluu3uan9jrune5vtfeemqpk93a0uljpp8fdd59t0u606p6rqnk38vx	2.0 ₳ asset12...f4h 1,466,547,114
addr_test1qqd47q5qzkv7avnlvef20v9t7m90uwkz0d2fd8cnyaa6carwr700yca9wxn4vk5sv02px0q6ke6uchd6qy8q3zj788wqs0j2ex	2.0 ₳ MINT 2 asset12...f4h 1,020,579,893
addr_test1qqs4hg2mded6m3r7zmqs8nlcnrctrkn9qzazak6nlrhgngqwv23zcmspv5ch9hlt40rznmzy8w7dydglnz954k2g7ysqnthgw4	22.0 ₳ asset1v...5ca 1
Total Output	31.611499 ₳

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