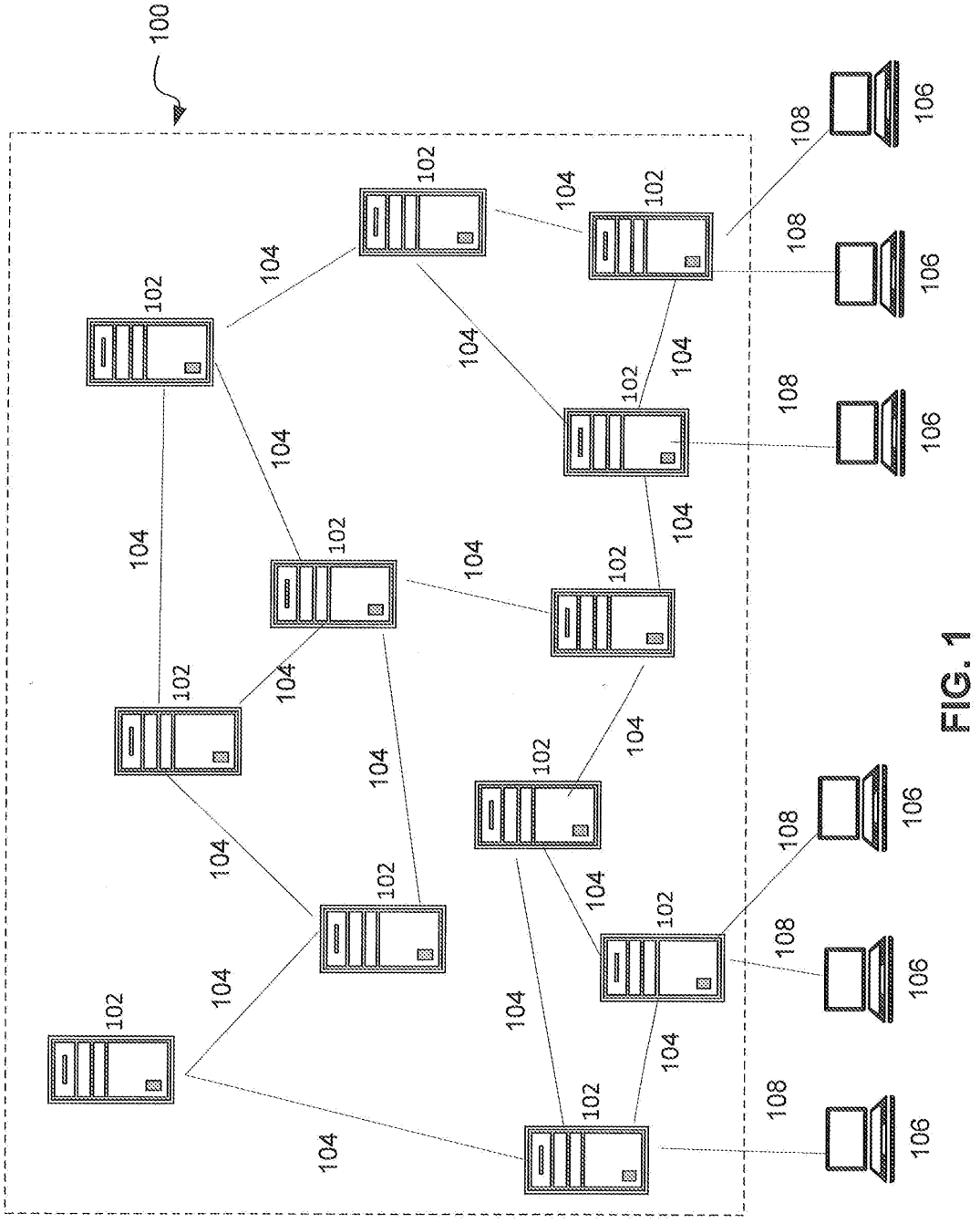

FIG. 1

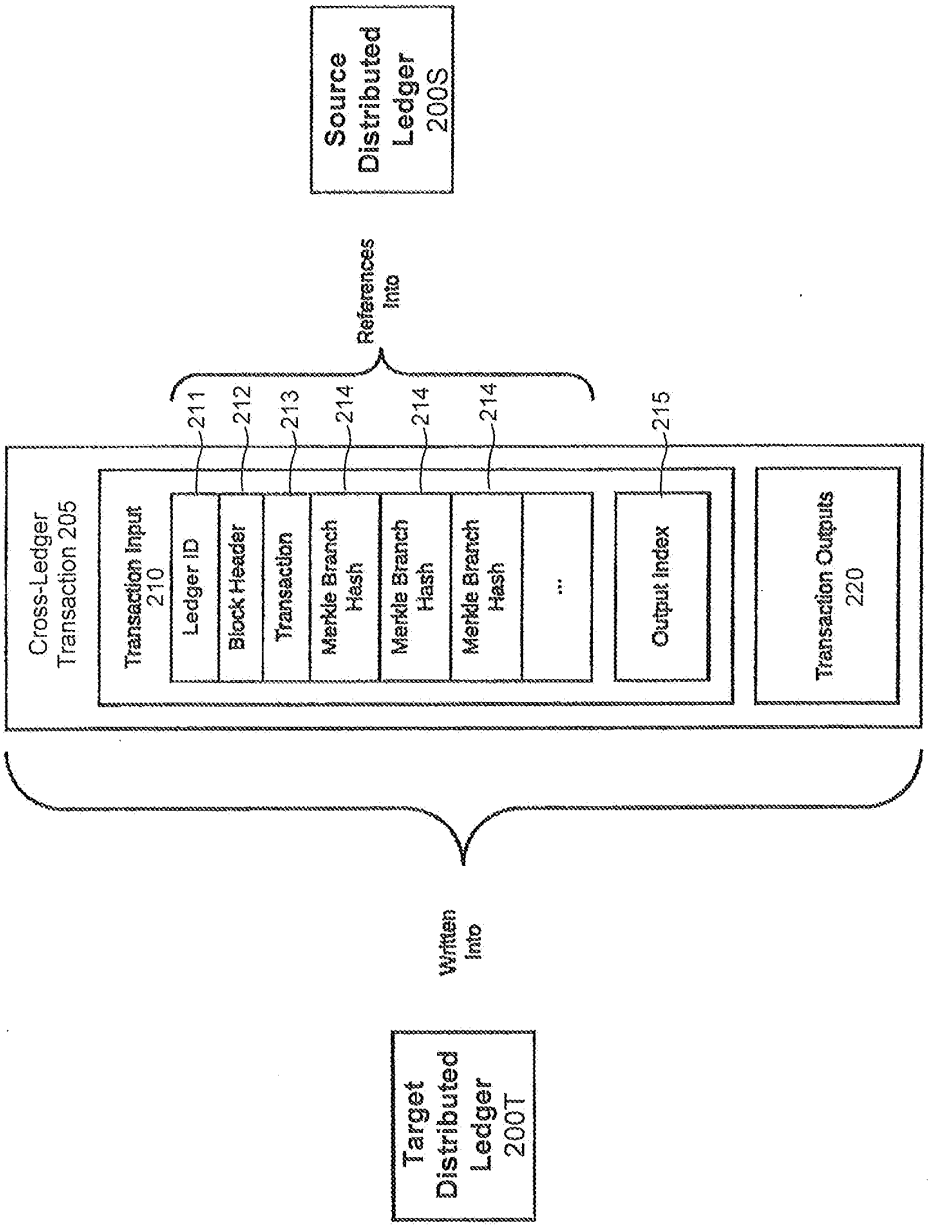
Source Distributed Ledger 200S
References into
211
212
213
214
214
214
215
Cross-Ledger Transaction 205
Transaction Input 210
Ledger ID
Block Header
Transaction
Merkle Branch Hash
Merkle Branch Hash
Merkle Branch Hash
...
Output Index
Transaction Outputs 220
Written into
Target Distributed Ledger 200T

FIG. 2

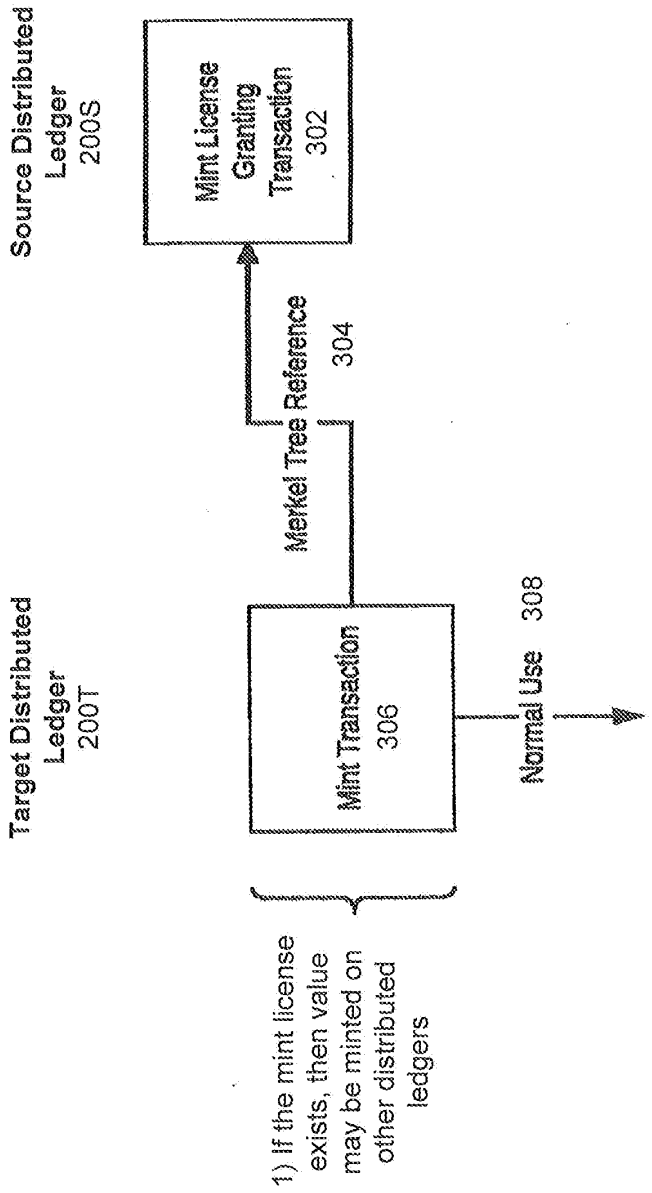
Source Distributed Ledger 200S
Mint License Granting Transaction 302
Merkel Tree Reference 304
Target Distributed Ledger 200T
Mint Transaction 306
Normal Use 308
1) If the mint license exists, then value may be minted on other distributed ledgers

FIG. 3

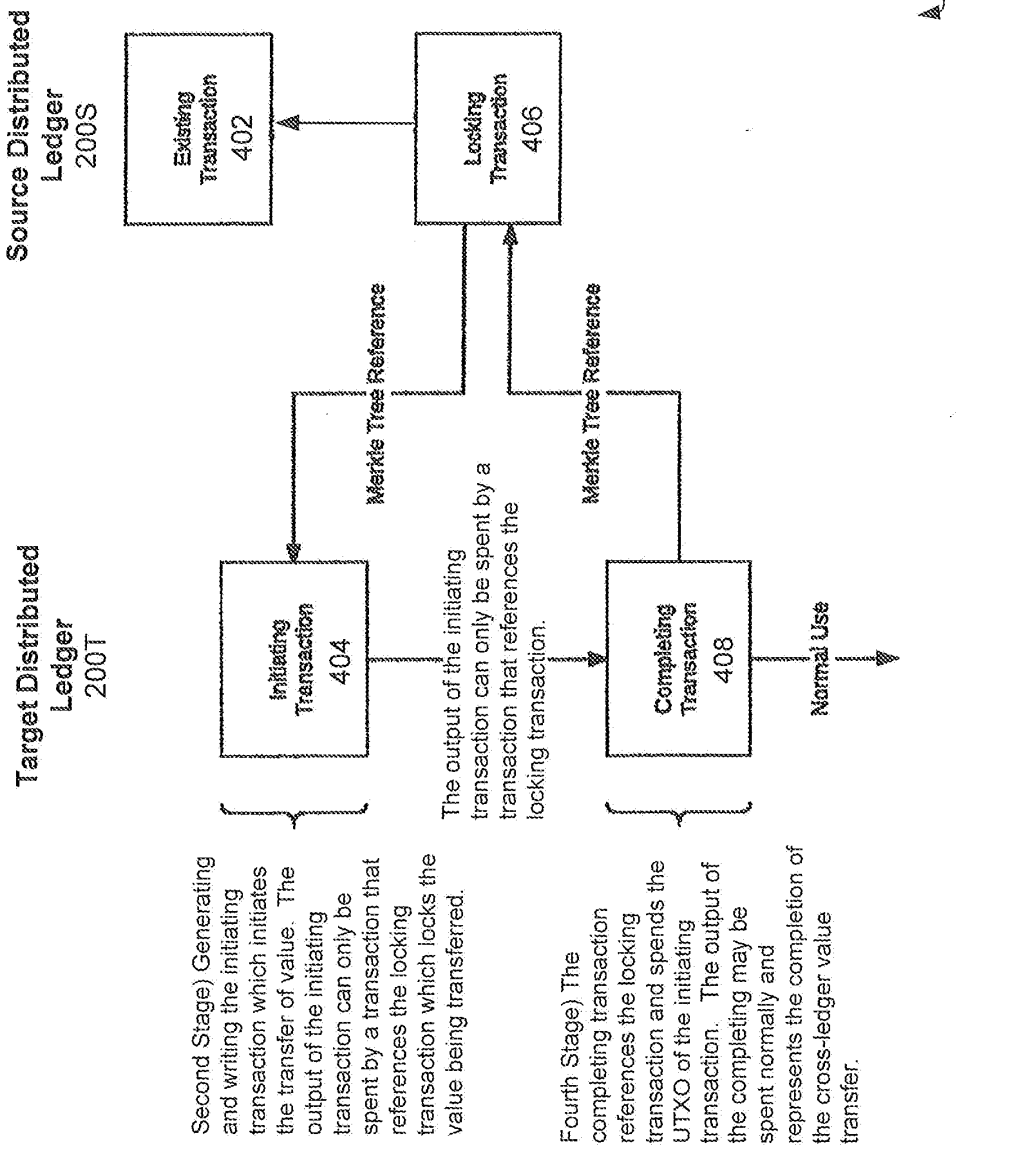
Source Distributed Ledger 200S
Existing Transaction 402
Locking Transaction 406
First Stage) Identification of the existing transaction which contains the UTXO that is to be spent in the target distributed ledger.
Third Stage) Generating and writing the locking transaction which spends the UTXO of the existing transaction into a locked TXO, guaranteeing that its value stops existing in the source distributed ledger. The locking transaction also references the initiating transaction.
Merkle Tree Reference
Merkle Tree Reference
Target Distributed Ledger 200T
Initiating Transaction 404
Completing Transaction 408
Normal Use
The output of the initiating transaction can only be spent by a transaction that references the locking transaction.
Second Stage) Generating and writing the initiating transaction which initiates the transfer of value. The output of the initiating transaction can only be spent by a transaction that references the locking transaction which locks the value being transferred.
Fourth Stage) The completing transaction references the locking transaction and spends the UTXO of the initiating transaction. The output of the completing may be spent normally and represents the completion of the cross-ledger value transfer.
400

FIG. 4

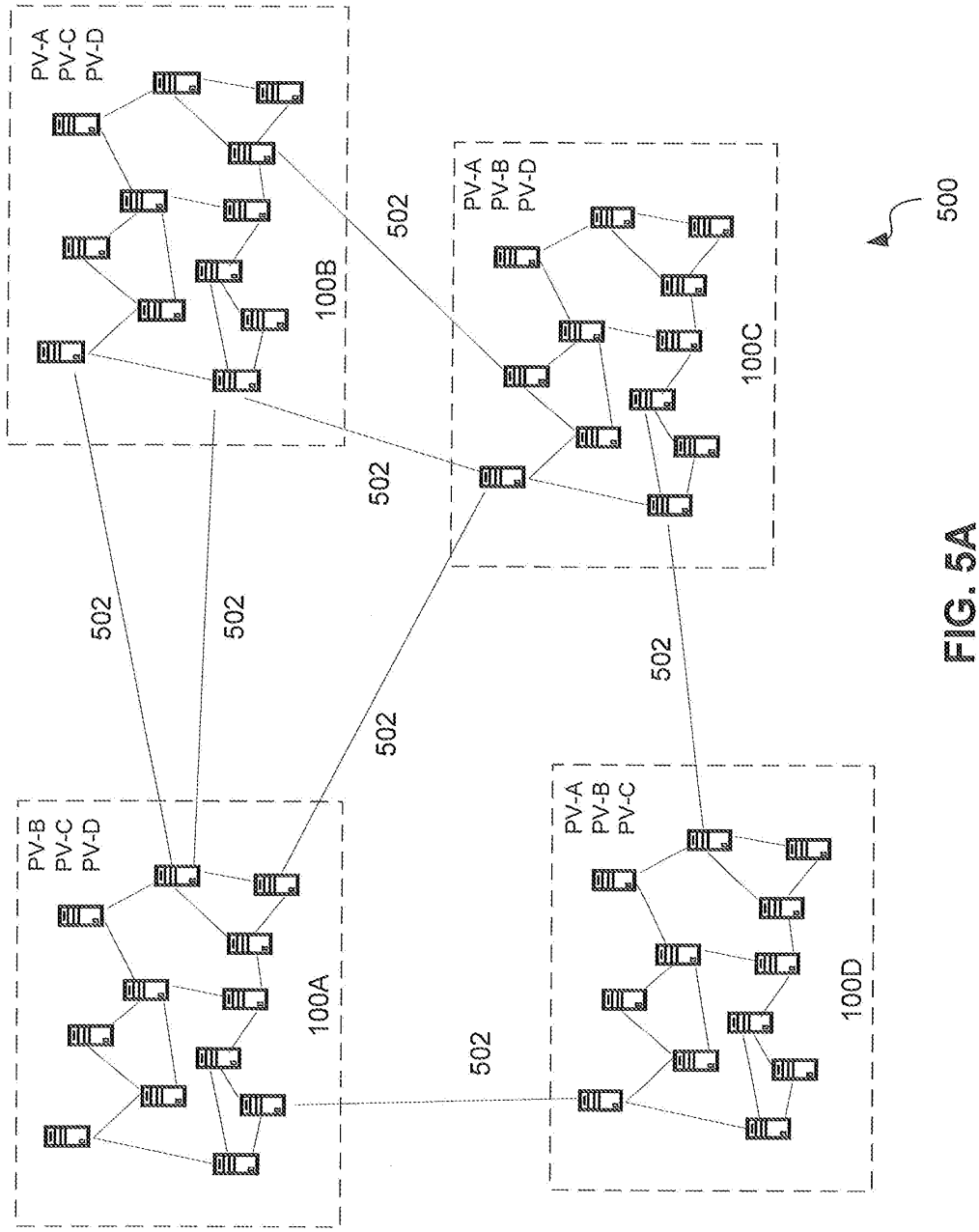
PV-A
PV-C
PV-D
100B
502
PV-A
PV-B
PV-D
100C
500
502
502
502
502
PV-B
PV-C
PV-D
100A
502
PV-A
PV-B
PV-C
100D

FIG. 5A

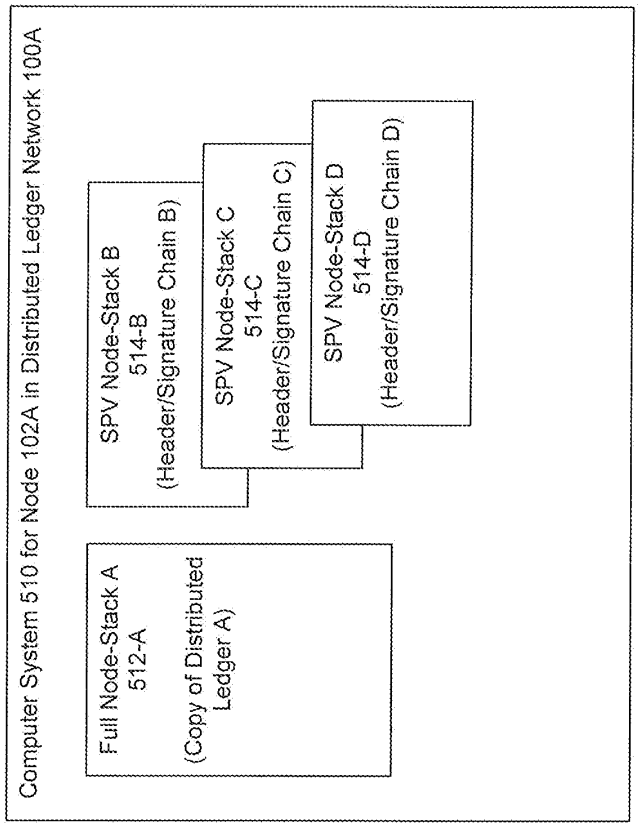
Computer System 510 for Node 102A in Distributed Ledger Network 100A
Full Node-Stack A
512-A
(Copy of Distributed Ledger A)
SPV Node-Stack B
514-B
(Header/Signature Chain B)
SPV Node-Stack C
514-C
(Header/Signature Chain C)
SPV Node-Stack D
514-D
(Header/Signature Chain D)

FIG. 5B

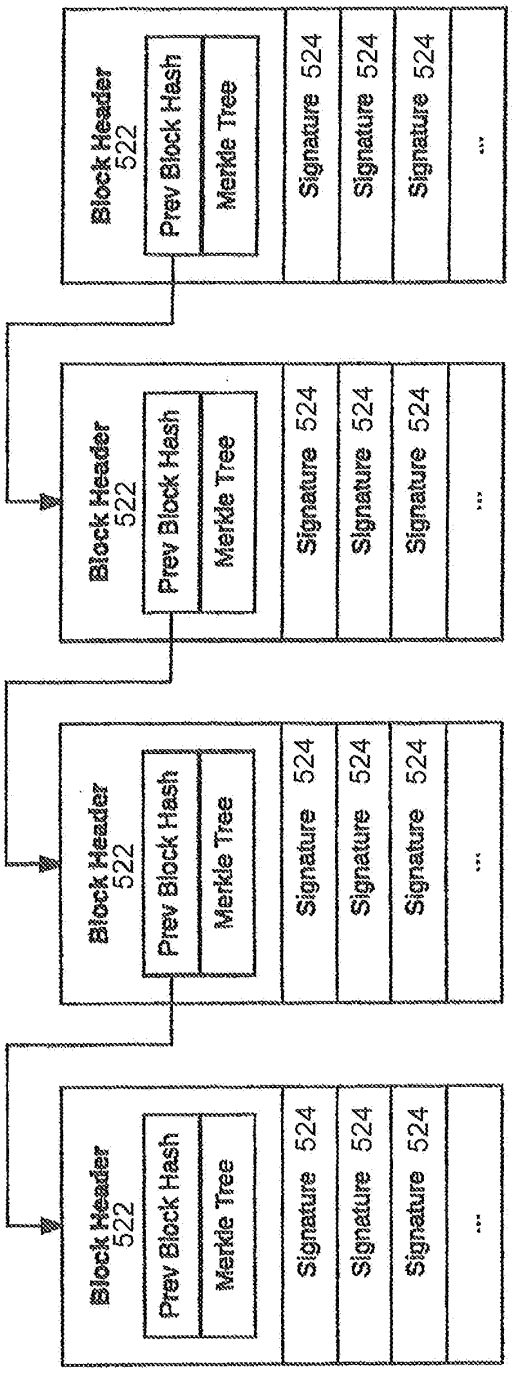
Block Header
522
Prev Block Hash
Merkle Tree
Signature 524
Signature 524
Signature 524
...
Block Header
522
Prev Block Hash
Merkle Tree
Signature 524
Signature 524
Signature 524
...
Block Header
522
Prev Block Hash
Merkle Tree
Signature 524
Signature 524
Signature 524
...
Block Header
522
Prev Block Hash
Merkle Tree
Signature 524
Signature 524
Signature 524
...
Header/Signature Chain
520

FIG. 5C

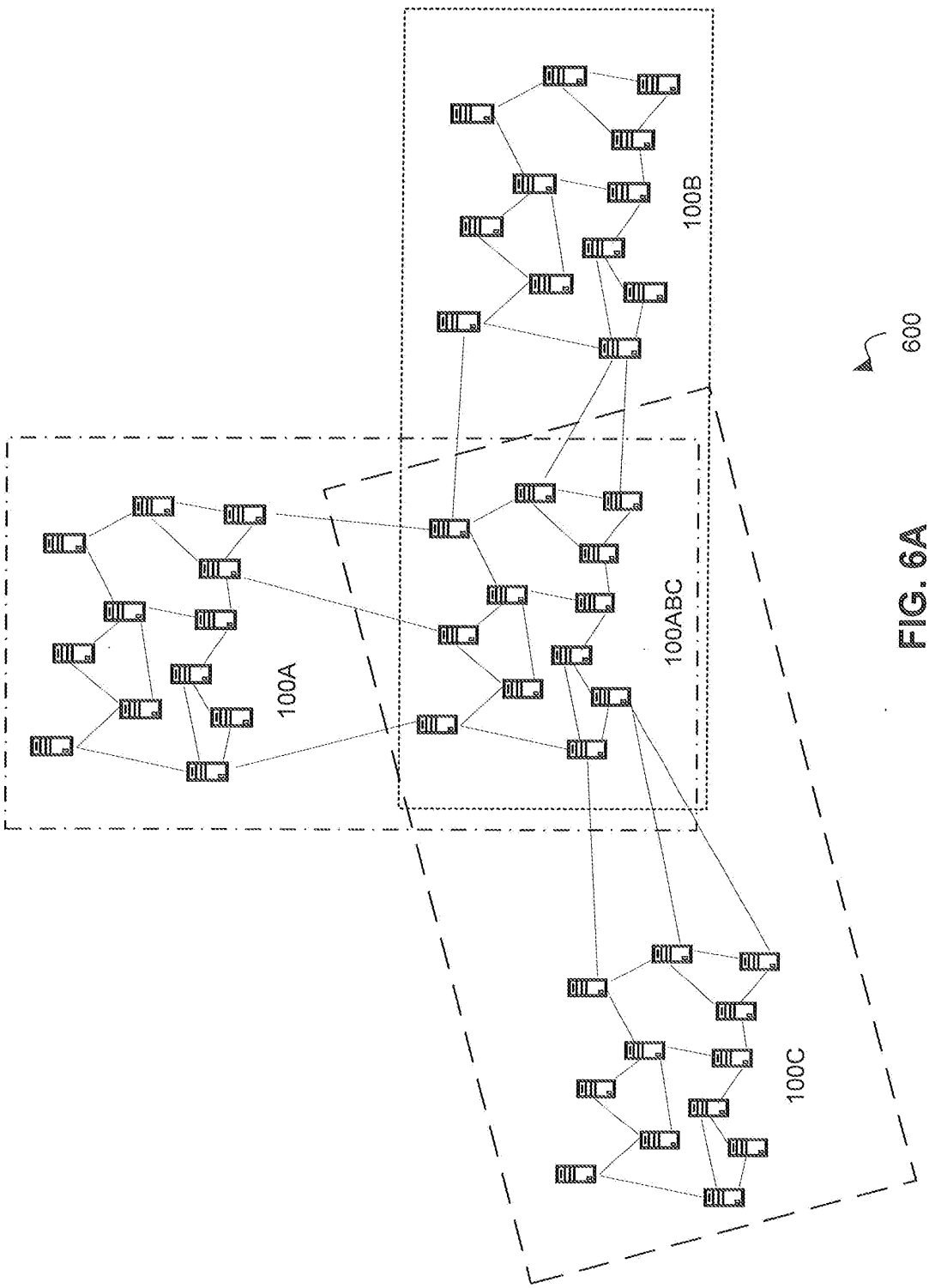

FIG. 6A

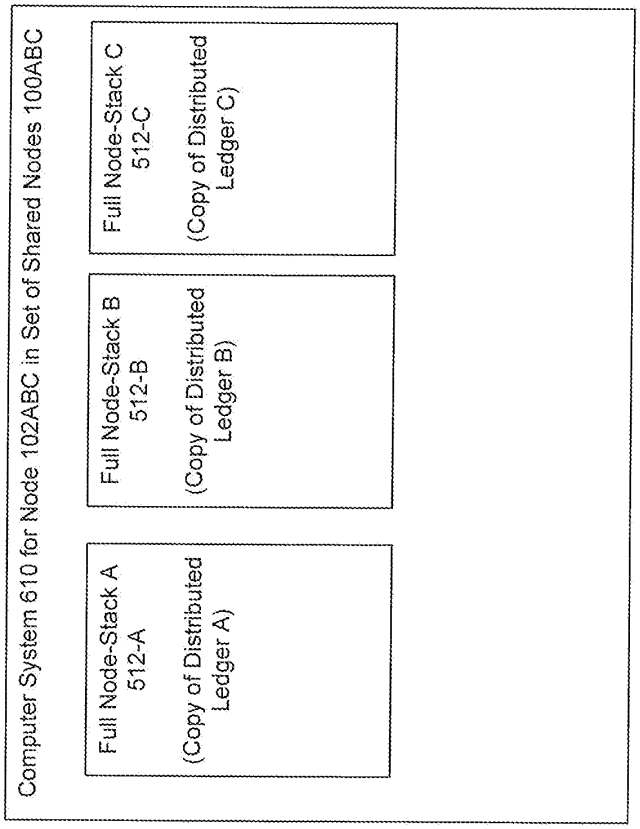
Computer System 610 for Node 102ABC in Set of Shared Nodes 100ABC
Full Node-Stack A 512-A (Copy of Distributed Ledger A)
Full Node-Stack B 512-B (Copy of Distributed Ledger B)
Full Node-Stack C 512-C (Copy of Distributed Ledger C)

FIG. 6B

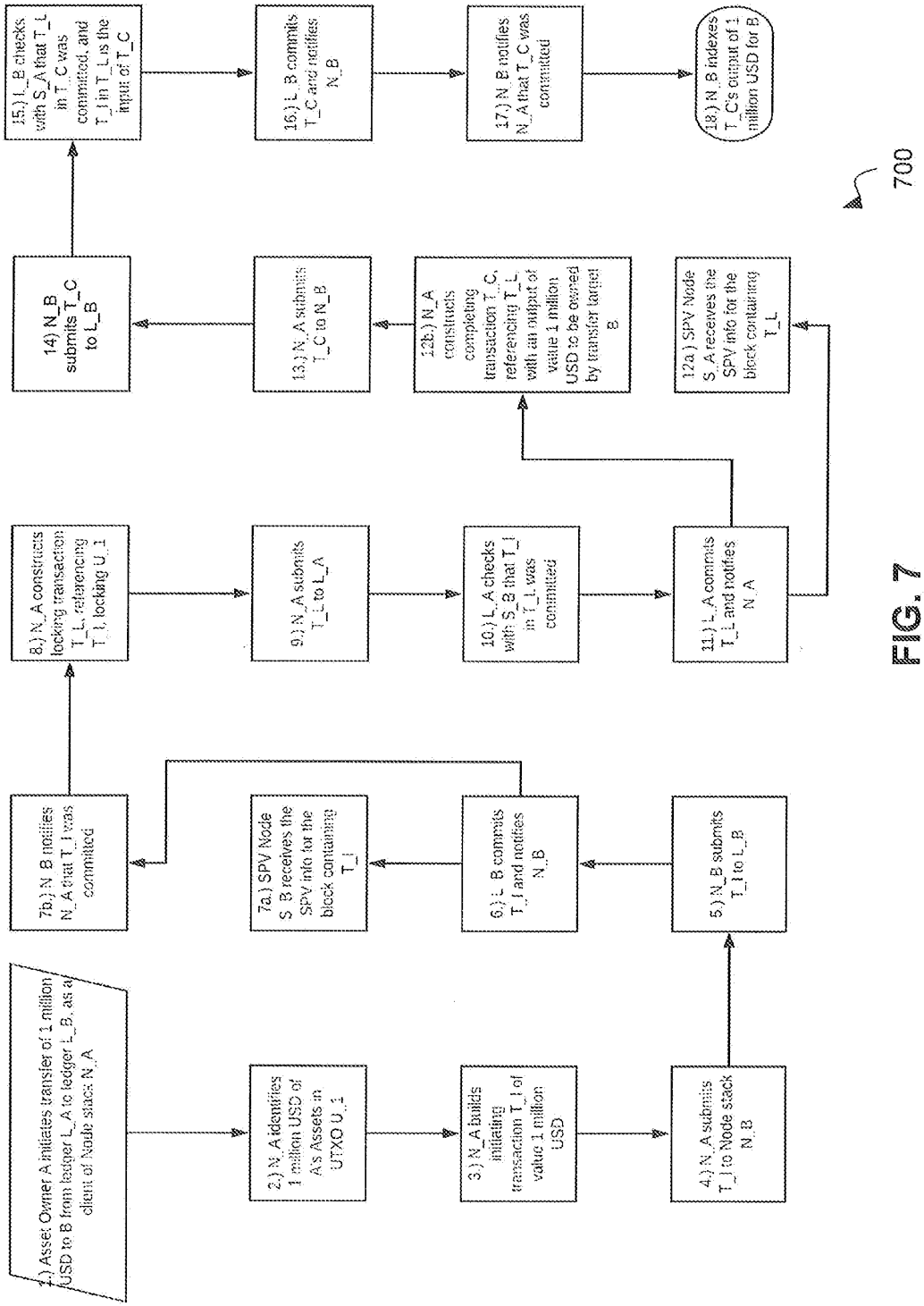
1.) Asset Owner A initiates transfer of 1 million USD to B from ledger L_A to ledger L_B, as a client of Node stack N_A
2.) N_A identifies 1 million USD of A's Assets in UTXO U_1
3.) N_A builds initiating transaction T_I of value 1 million USD
4.) N_A submits T_I to Node stack N_B
5.) N_B submits T_I to L_B
6.) L_B commits T_I and notifies N_B
7a.) SPV Node S_B receives the SPV info for the block containing T_I
7b.) N_B notifies N_A that T_I was committed
8.) N_A constructs locking transaction T_L referencing T_I, locking U_1
9.) N_A submits T_L to L_A
10.) L_A checks with S_B that T_I in T_L was committed
11.) L_A commits T_L and notifies N_A
12a.) SPV Node S_A receives the SPV info for the block containing T_L
12b.) N_A constructs completing transaction T_C, referencing T_L with an output of value 1 million USD to be owned by transfer target B
13.) N_A submits T_C to N_B
14) N_B submits T_C to L_B
15.) L_B checks with S_A that T_L in T_C was committed, and T_I in T_L is the input of T_C
16.) L_B commits T_C and notifies N_B
17.) N_B notifies N_A that T_C was committed
18.) N_B indexes T_C's output of 1 million USD for B
700

FIG. 7

801) The reference data in the transaction input is passed to the foreign node(s) inside a query

802) The foreign node(s) use the ledger identifier in the transaction input to find the correct block hash to block index to query.

803) The foreign node(s) use the block hash in the transaction input to find the correct block to examine.

804) The foreign node(s) extract the block header from the identified block.

805) The foreign node(s) extract the Merkle root hash from the identified block.

806) The foreign node(s) compute a transaction hash for the transaction in the transaction input.

807) The foreign node(s) compute a Merkle root hash from the transaction hash and the provided Merkle branch hashes.

808) The foreign node(s) compare the computed Merkle root hash with the Merkle root hash in the identified block.

809) The foreign node(s) return that the transaction is valid if the hashes match, invalid if they don't match.

800

FIG. 8

(steps 10 and 15 in FIG. 7)

```
1  // When a source wants to send an asset to the target, which is in another distributed ledger (step 1 in FIG. 7)
2  NodeStack.InitiateCrossLedgerTransaction(Source, Asset, Target)
3      // Find the UTXO containing the asset to transfer (step 2 in FIG. 7)
4      UTXOToSpend = FindUTXOForAsset(Source, Asset)
5      // Construct the transaction (step 3 in FIG. 7)
6      InitiatingTransaction = MakeInitiatingTransactionForUTXO(UTXOToSpend)
7      // Index the transaction with metadata for later use
8      InitiatingTransactionIndex.Index(InitiatingTransaction, {Source, UTXOToSpend, Target})
9      // Sign the transaction with the private key of the source
10     Source.SignTransaction(InitiatingTransaction)
11     // Find the target node stack from the target
12     TargetNodeStack = GetNodeStack(Target)
13     // Send the transaction to the foreign node stack (step 4 in FIG. 7)
14     TargetNodeStack.CommitForeignTransaction(Source, InitiatingTransaction)
15     // Done
16     Exit
```

FIG. 9

```
1  // Called when a transaction is committed. Add UTXOs to pool.
2  NodeStack.NotifyCommitedTransaction(Transaction, SPVInfo (If foreign))
3      // Received an initiating transaction from a foreign node (step 7b in FIG. 7)
4      If IsInitiating(Transaction) and SPVInfo exists
5          // Find the UTXO that was selected for transfer from the index
6          UTXOToLock = InitiatingTransactionIndex.GetUTXOFor(Transaction)
7          // Build the locking transaction (step 8 in FIG. 7)
8          LockingTransaction = MakeLockingTransaction(UTXOToLock, Transaction, SPVInfo)
9          // Find the source of the initiating transaction
10         Source = InitiatingTransactionIndex.GetSourceFor(Transaction)
11         // Sign the locking transaction with the private key of the source
12         Source.SignTransaction(LockingTransaction)
13         // Submit the locking transaction to the local ledger (step 9 in FIG. 7)
14         DistributedLedger.Submit(Transaction)
15         // Done
16         Exit
17     // Received a locking transaction
18     If IsLocking(Transaction)
19         // Find the target from the indexed target for the initiating transaction within the locking transaction
20         Target = InitatingTransactionIndex.GetTargetFor(GetInitiatingTransaction(Transaction))
21         // Build the completing transaction (step 12b in FIG. 7)
22         CompletingTransaction=MakeCompletingTransaction(Transaction, DistributedLedger.GetSPVInfo(Transaction), Target)
23         // Find the source of the initiating transaction in the locking transaction from the index
24         Source = InitiatingTransactionIndex.GetSourceFor(GetInitiatingTransaction(Transaction))
25         // Sign the completing transaction with the private key of the source
26         Source.SignTransaction(CompletingTransaction)
27         // Find the target node stack from the target
28         TargetNodeStack = GetNodeStack(Target)
29         // Send the transaction to the foreign node stack (step 13 in FIG. 7)
30         TargetNodeStack.CommitForeignTransaction(Source, CompletingTransaction)
31         // Done
32         Exit
33     // Remove the TXOs spent and the TXOs created from this transaction from the set of UTXOs (step 18 in FIG. 7)
34     UpdateUTXOs(Transaction)
35     // Done
36     Exit
```

FIG. 10

```
1  // Called to commit a transaction on a foreign distributed ledger (steps 4 and 13 in FIG. 7)
2  NodeStack.CommitForeignTransaction(Source, Transaction)
3      // Adds the foreign transaction to an index so we can look up the foreign nodestack that submitted it
4      ForeignIndex.IndexForeignTransaction(Transaction,GetNodeStack(Source) )
5      // Submits the transaction (steps 5 and 14 in FIG. 7)
6      DistributedLedger.Submit(Transaction)
7      // Done
8      Exit
```

FIG. 11

```
1   // Submits a transaction to the ledger (steps 5, 9 and 14 in FIG. 7)
2   DistributedLedger.Submit(Transaction)
3       // Make sure the transaction has a valid format
4       CheckFormat(Transaction)
5       ...
6       // Verify all inputs
7       For Transaction.Inputs
8           // Checks SPV references in committing and locking transactions
9           If ContainsSPVReference(Transaction.Input)
10              // If the input reference isn't in the SPV node (steps 10 and 15 in FIG. 7)
11              If SPVNode.CheckSPV(Transaction.Input) is false
12                  // Deny the transaction
13                  Reject Transaction
14                  // Done
15                  Exit
16              // Checks if locking transaction references TXO spent by this Transaction
17              If ContainsSPVReference(Transaction.Input.Transaction) (step 15 in FIG. 7)
18                  // Get the standard input from the transaction
19                  StandardInput = GetStandardInput(Transaction.Inputs)
20                  // If the initiating transaction inside the locking transaction is different
21                  // from the input of this transaction
22                  If StandardInput.Transaction.ID != Transaction.Input.Transaction.ID
23                      // Deny the transaction
24                      Reject Transaction
25                      // Done
26                      Exit
27      // Put the transaction in the mempool so it will be selected to commit in a following block
28      PrepareForCommit(Transaction)
29      // Done
30      Exit
```

FIG. 12

```
1   // Called after a consensus on a block is reached (steps 6, 11 and 16 in FIG. 7)
2   DistributedLedger.CommitBlock(Block)
3       // Persist the block
4       StoreBlock(Block)
5       // Send SPV info to SPV nodes (steps 7a and 12a in FIG. 7)
6       SPVNode.SendSPVInfo(Block.GetSPVInfo())
7       // Notify the node stack that a block was committed (steps 7b and 12b in FIG. 7)
8       NodeStack.NotifyBlockCommit(Block)
9       // Done
10      Exit
```

FIG. 13

```
1   // Indexes UTXOs in blocks that have been committed, and
2   //notifies foreign stacks when their transactions are committed (steps 7b, 12b and 17 in FIG. 7)
3   NodeStack.NotifyBlockCommit(Block)
4       // For every transaction in a block
5       For Block.Transactions
6           // If it's a foreign transaction
7           If ForeignIndex.HasTransaction(Block.Transaction)
8               // Find the foreign node stack and notify it
9               ForeignIndex.Get(Block.Transaction).NodeStack.NotifyCommittedTransaction(Block.Transaction, SPVInfo)
10          // Always notify local stack of a transaction commit
11          NotifyCommitedTransaction(Block.Transaction)
12      // Done
13      Exit
```

FIG. 14

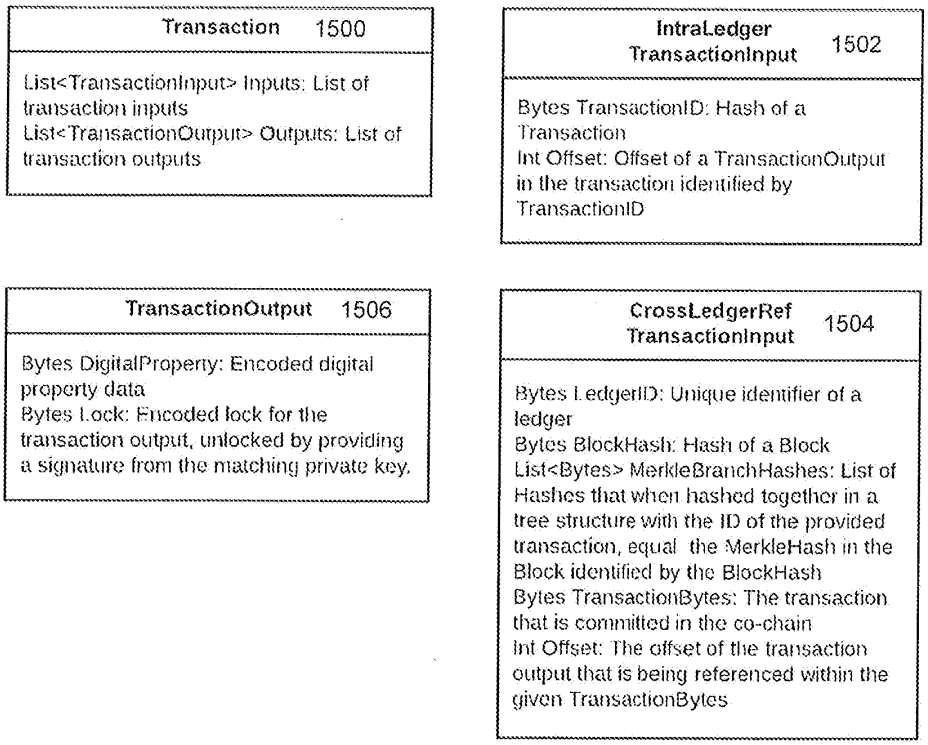
Transaction 1500
List<TransactionInput> Inputs: List of transaction inputs
List<TransactionOutput> Outputs: List of transaction outputs
IntraLedger TransactionInput 1502
Bytes TransactionID: Hash of a Transaction
Int Offset: Offset of a TransactionOutput in the transaction identified by TransactionID
TransactionOutput 1506
Bytes DigitalProperty: Encoded digital property data
Bytes Lock: Encoded lock for the transaction output, unlocked by providing a signature from the matching private key.
CrossLedgerRef TransactionInput 1504
Bytes LedgerID: Unique identifier of a ledger
Bytes BlockHash: Hash of a Block
List<Bytes> MerkleBranchHashes: List of Hashes that when hashed together in a tree structure with the ID of the provided transaction, equal the MerkleHash in the Block identified by the BlockHash
Bytes TransactionBytes: The transaction that is committed in the co-chain
Int Offset: The offset of the transaction output that is being referenced within the given TransactionBytes

FIG. 15

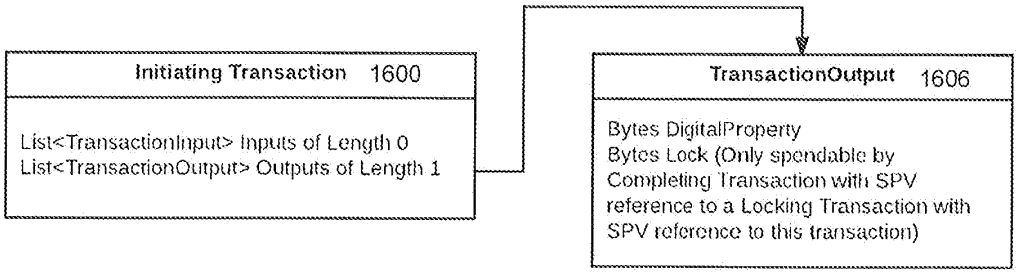
Initiating Transaction 1600
List<TransactionInput> Inputs of Length 0
List<TransactionOutput> Outputs of Length 1
TransactionOutput 1606
Bytes DigitalProperty
Bytes Lock (Only spendable by Completing Transaction with SPV reference to a Locking Transaction with SPV reference to this transaction)

FIG. 16

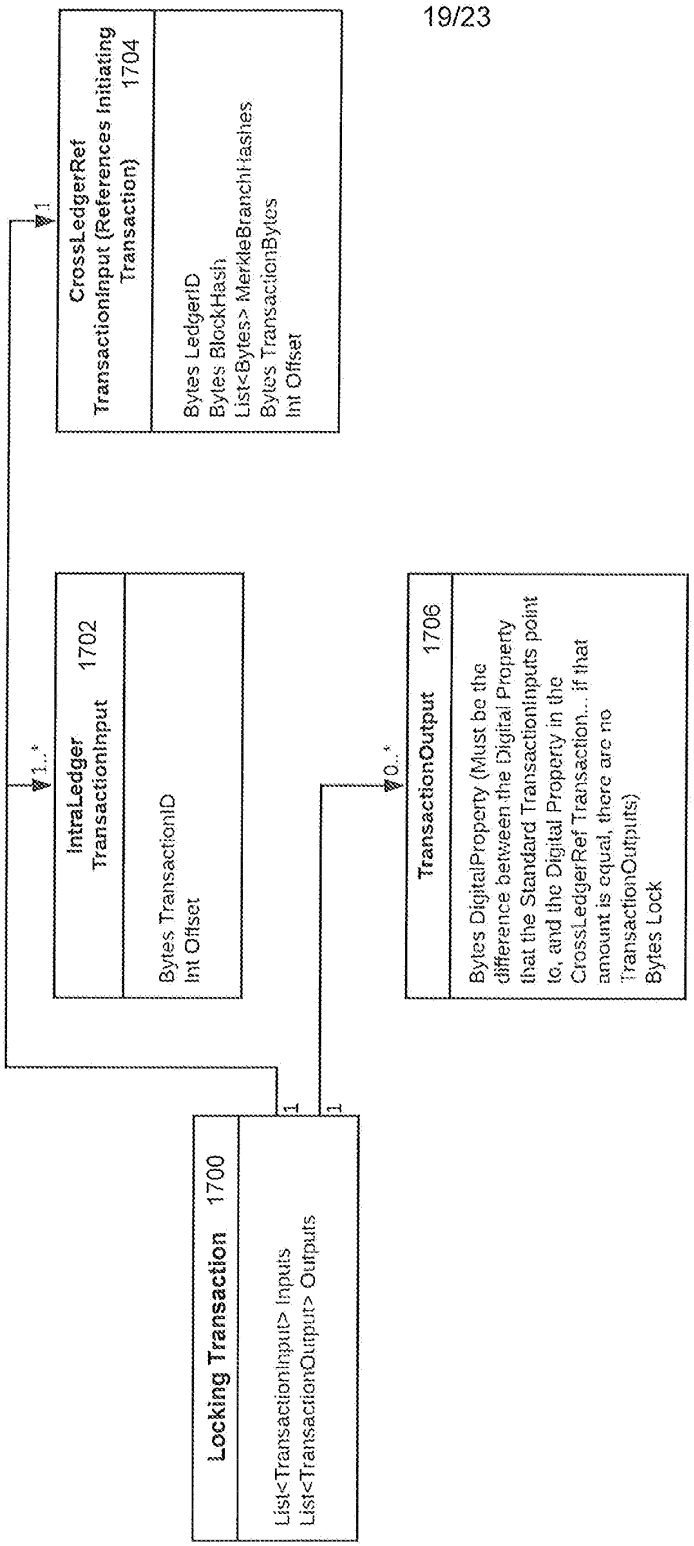
CrossLedgerRef
TransactionInput (References Initiating Transaction) 1704
Bytes LedgerID
Bytes BlockHash
List<Bytes> MerkleBranchHashes
Bytes TransactionBytes
Int Offset
1
IntraLedger
TransactionInput 1702
Bytes TransactionID
Int Offset
1..*
0..*
TransactionOutput 1706
Bytes DigitalProperty (Must be the difference between the Digital Property that the Standard TransactionInputs point to, and the Digital Property in the CrossLedgerRef Transaction... if that amount is equal, there are no TransactionOutputs)
Bytes Lock
Locking Transaction 1700
List<TransactionInput> Inputs
List<TransactionOutput> Outputs
1
1

FIG. 17

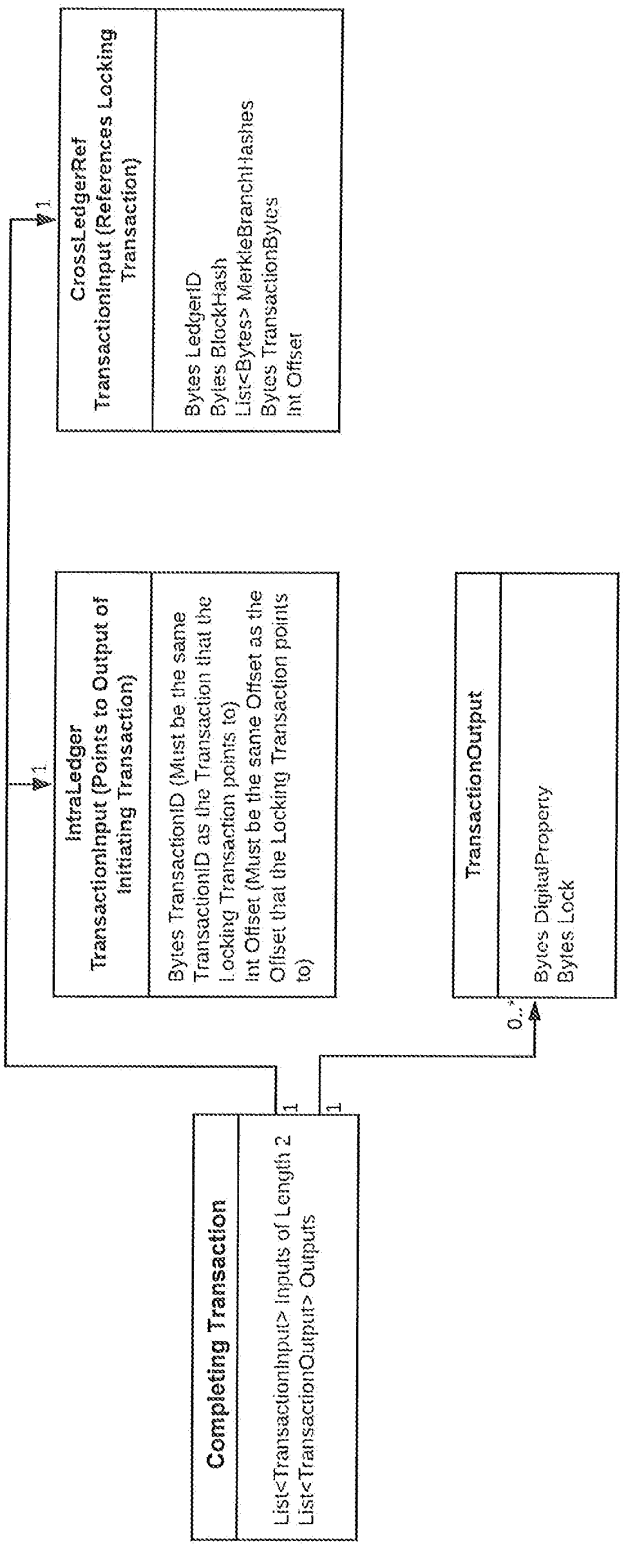
CrossLedgerRef
TransactionInput (References Locking Transaction)
Bytes LedgerID
Bytes BlockHash
List<Bytes> MerkleBranchHashes
Bytes TransactionBytes
int Offset
IntraLedger
TransactionInput (Points to Output of Initiating Transaction)
Bytes TransactionID (Must be the same TransactionID as the Transaction that the Locking Transaction points to)
int Offset (Must be the same Offset as the Offset that the Locking Transaction points to)
TransactionOutput
Bytes DigitalProperty
Bytes Lock
Completing Transaction
List<TransactionInput> Inputs of Length 2
List<TransactionOutput> Outputs
1
1
1
1
0..*

FIG. 18

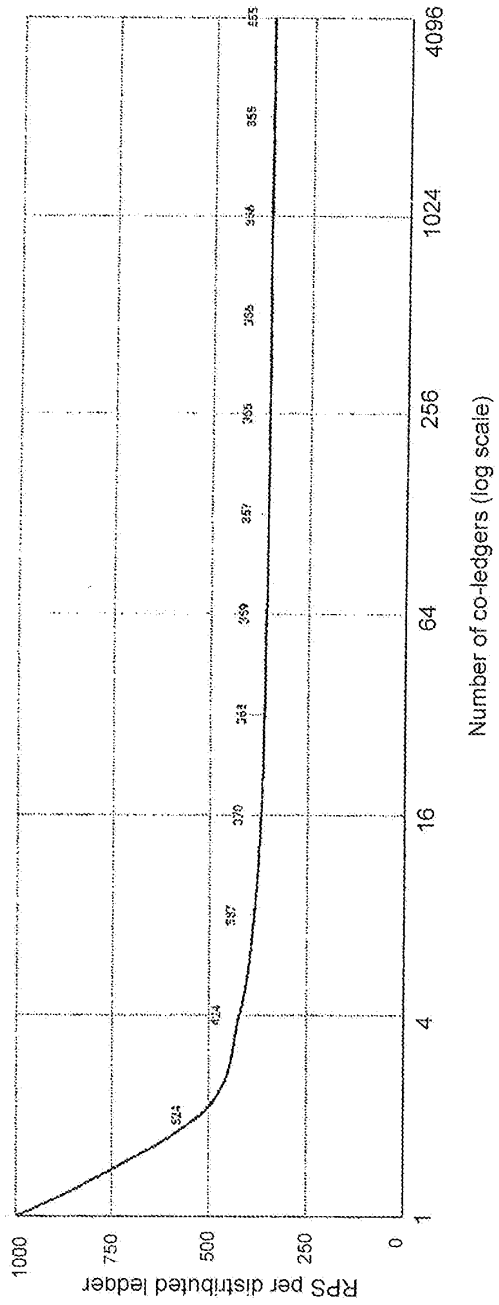
RPS per distributed ledger
1000
750
500
250
0
1
4
16
64
256
1024
4096
Number of co-ledgers (log scale)

FIG. 19

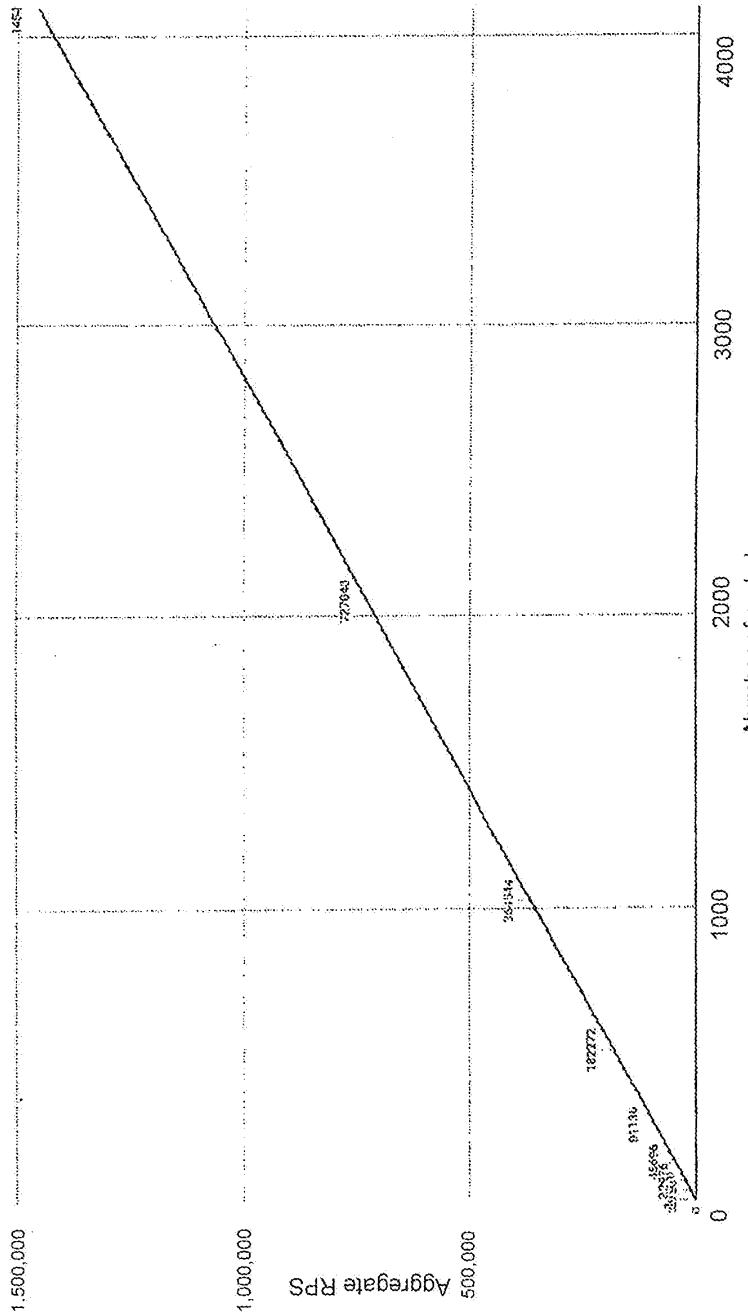
1,500,000
1,000,000
500,000
Aggregate RPS
0
1000
2000
3000
4000
Number of co-ledgers

FIG. 20

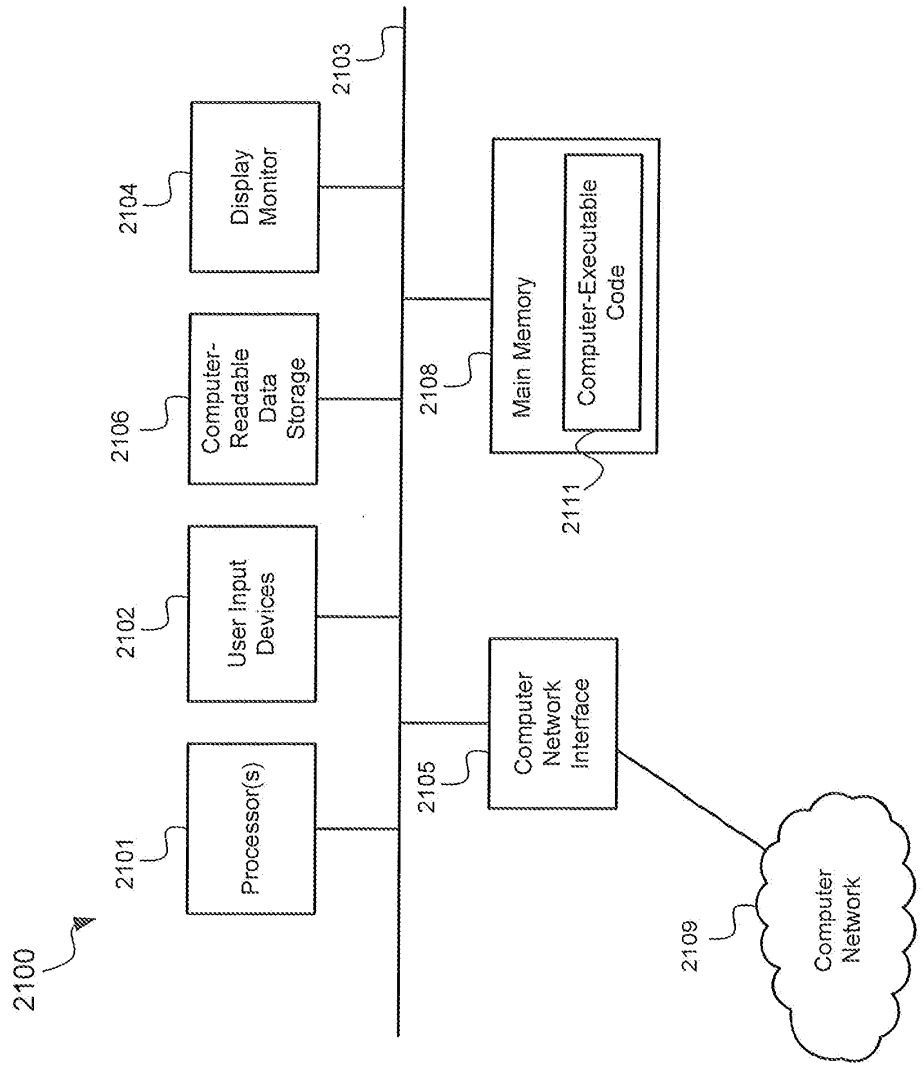
2100
2101
Processor(s)
2102
User Input Devices
2103
2104
Display Monitor
2105
Computer Network Interface
2106
Computer-Readable Data Storage
2108
Main Memory
Computer-Executable Code
2109
Computer Network
2111

FIG. 21