

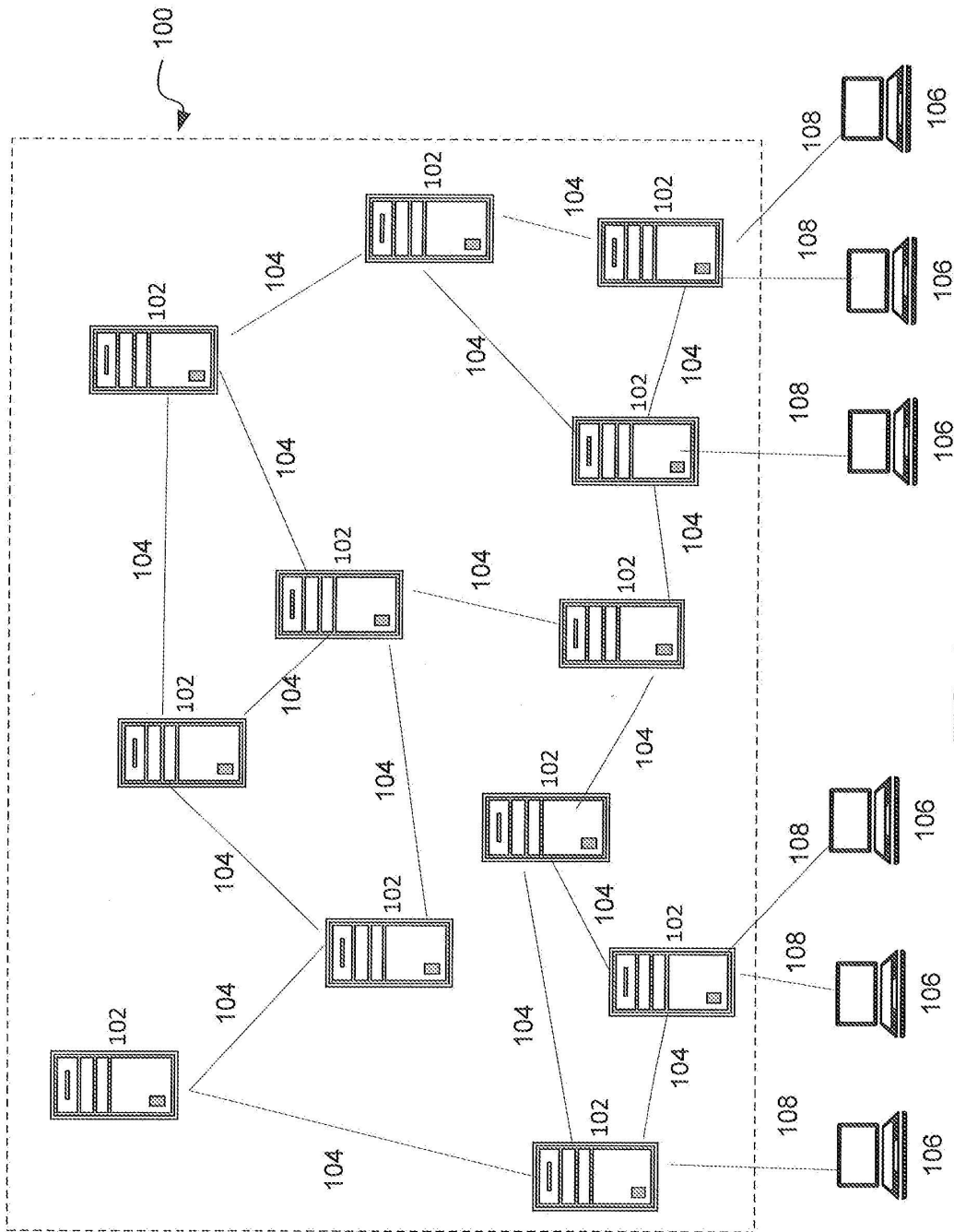


FIG. 1

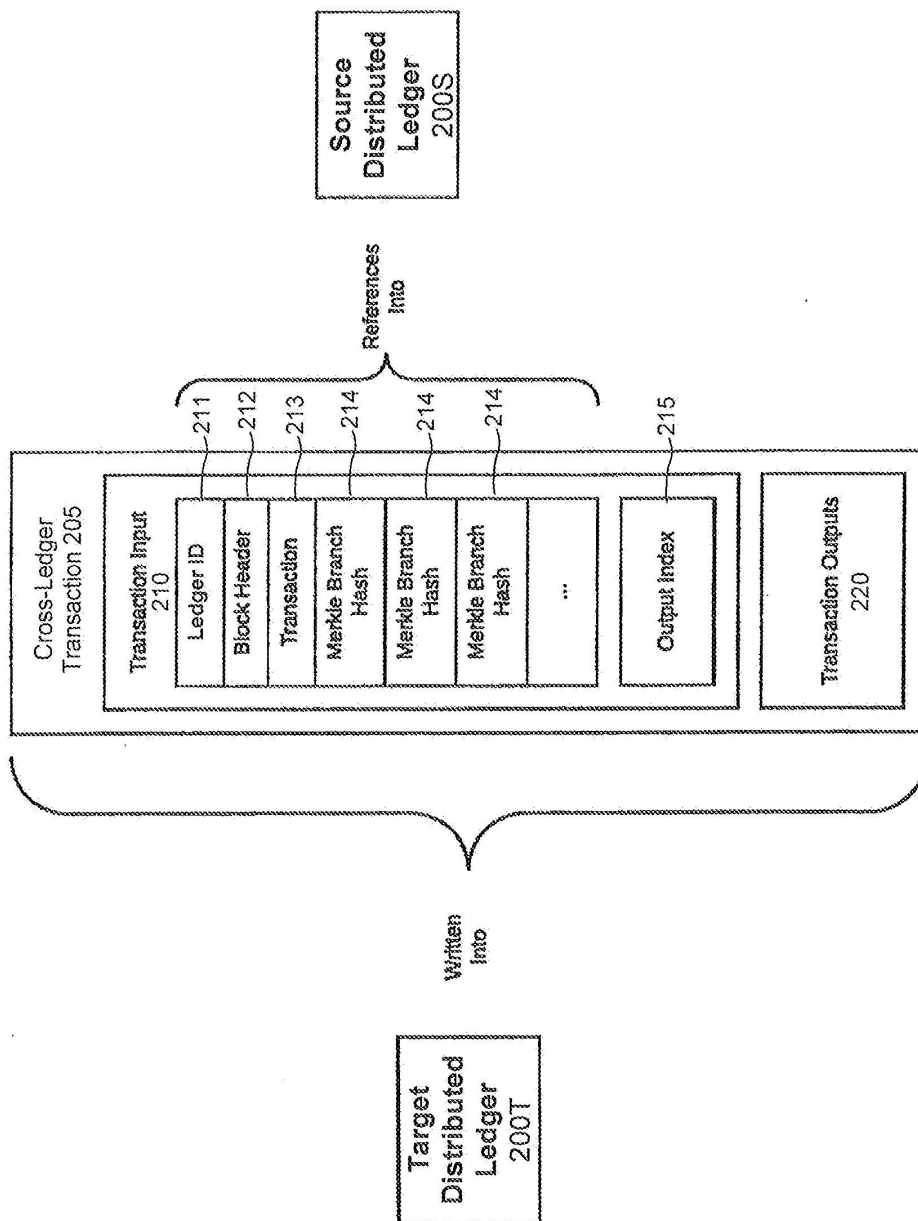
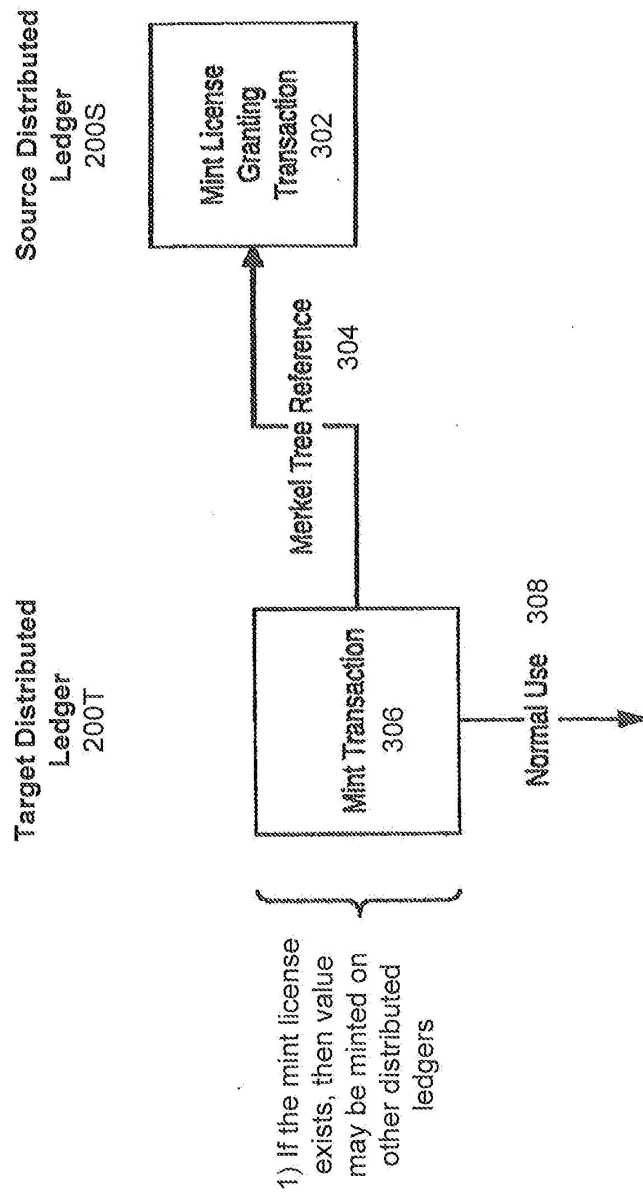
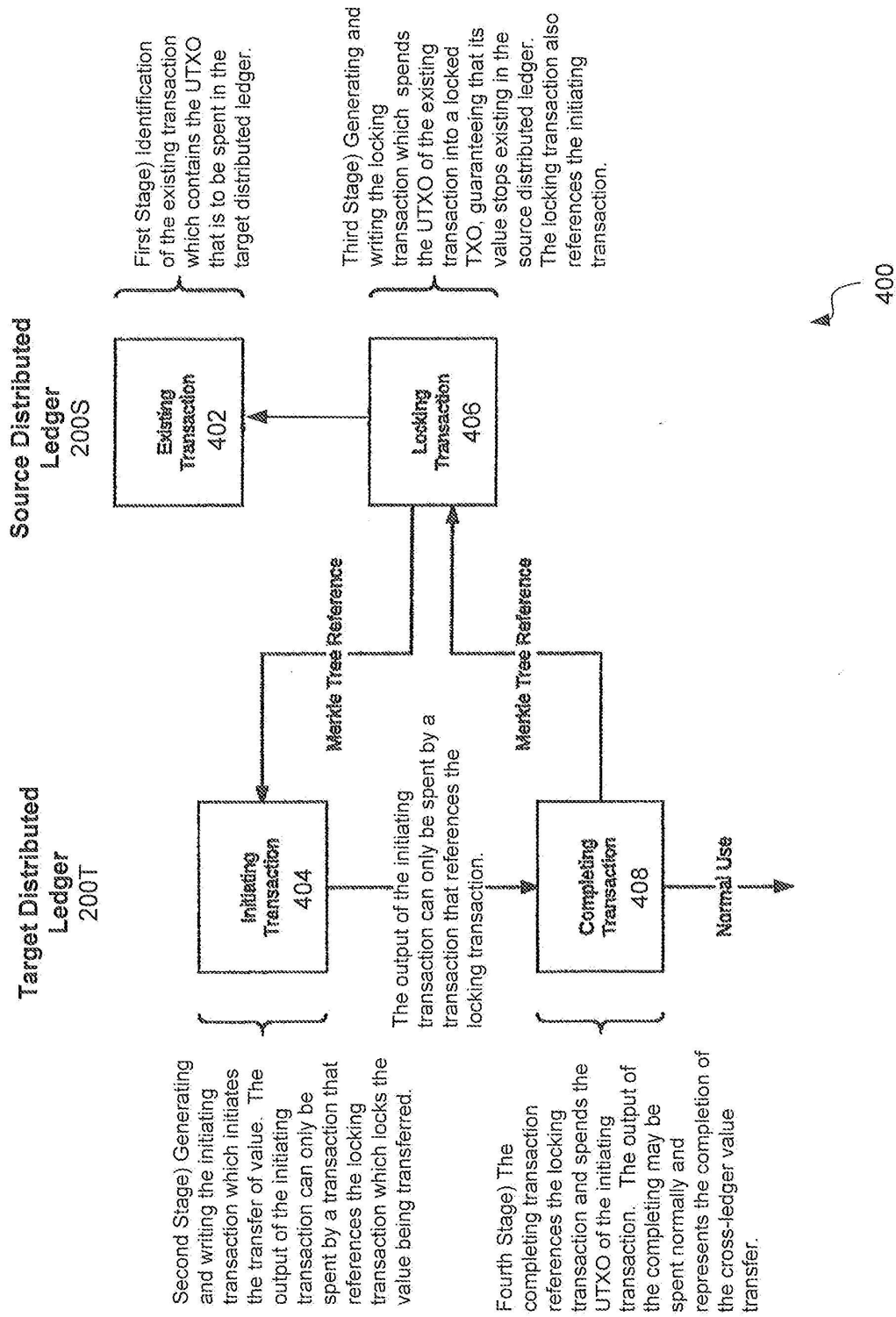
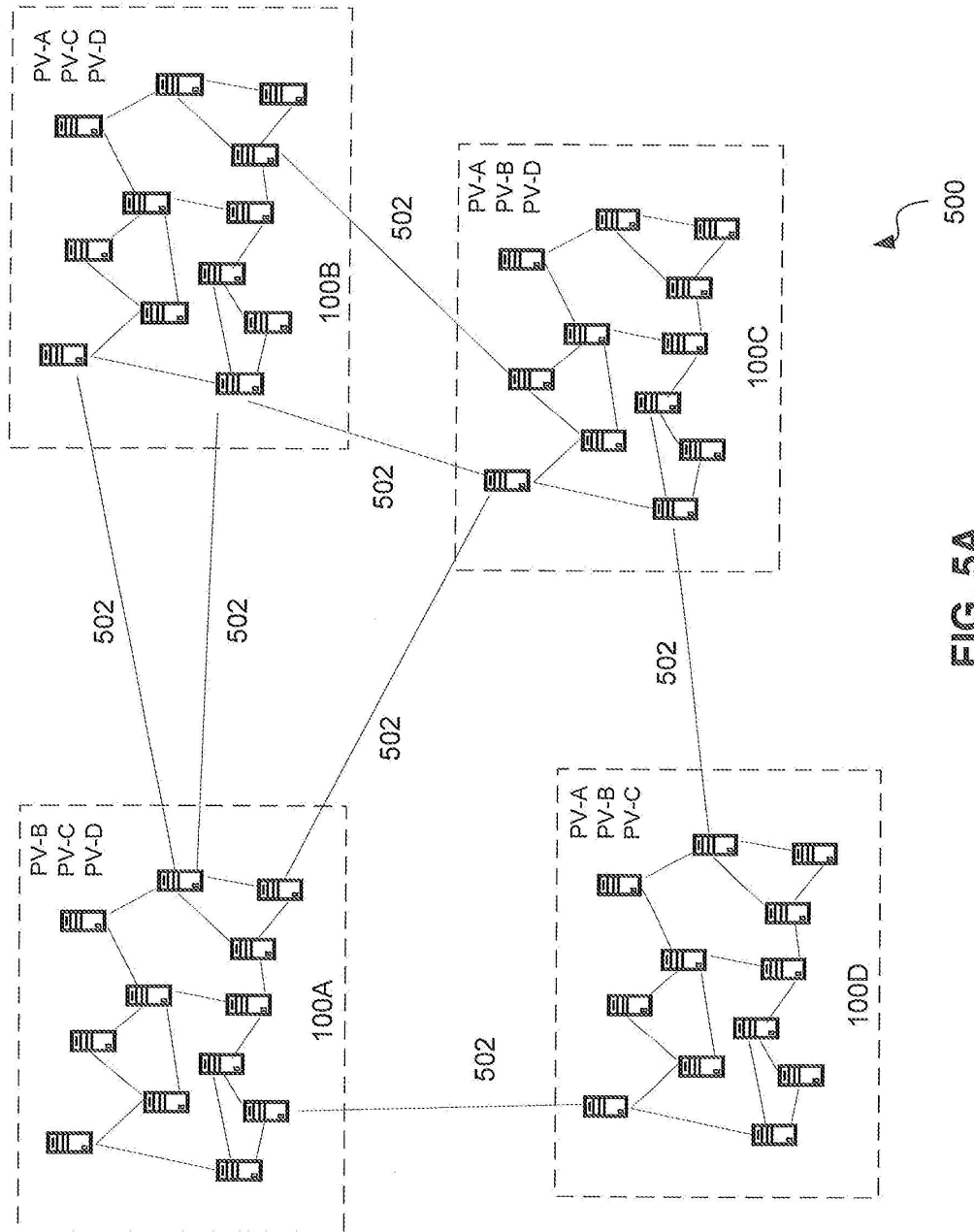
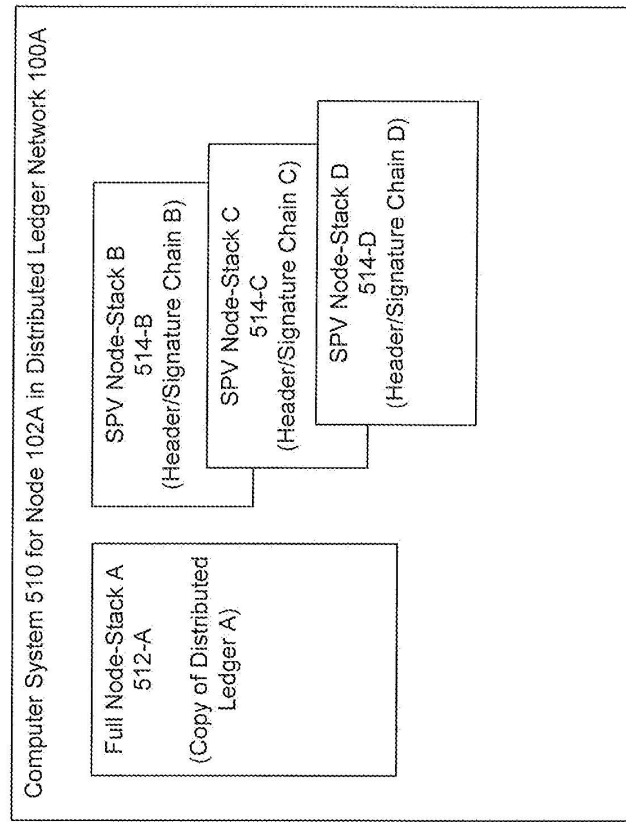


FIG. 2

3
G
L





**FIG. 5B**

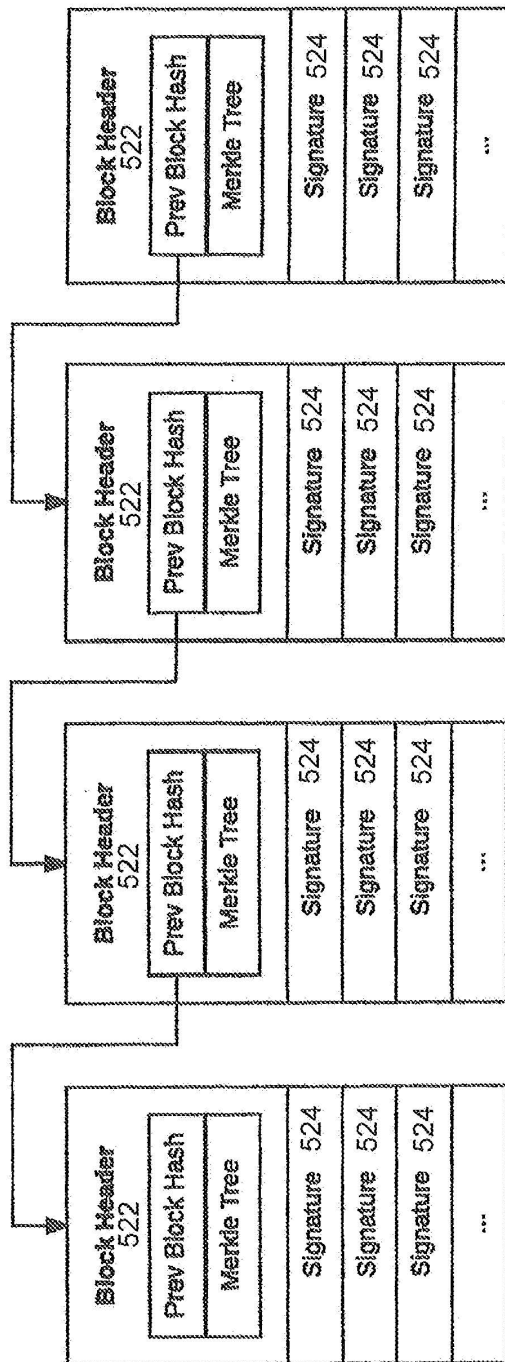


FIG. 5C Header/Signature Chain 520

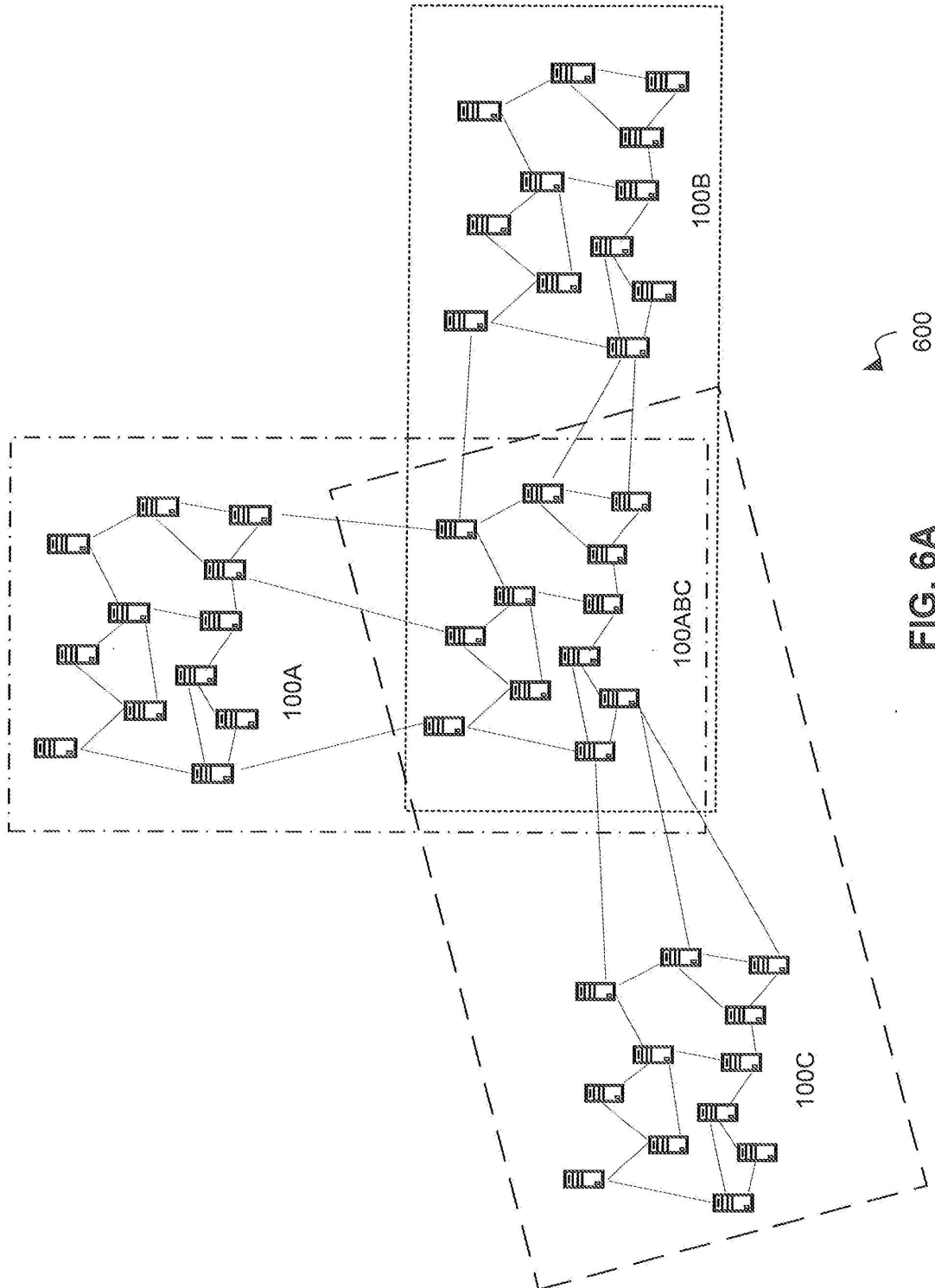


FIG. 6A

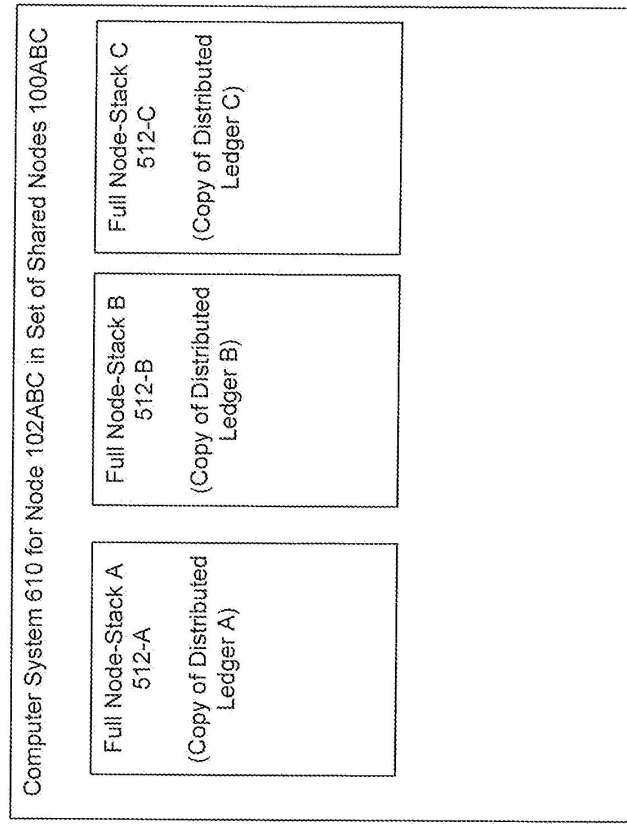
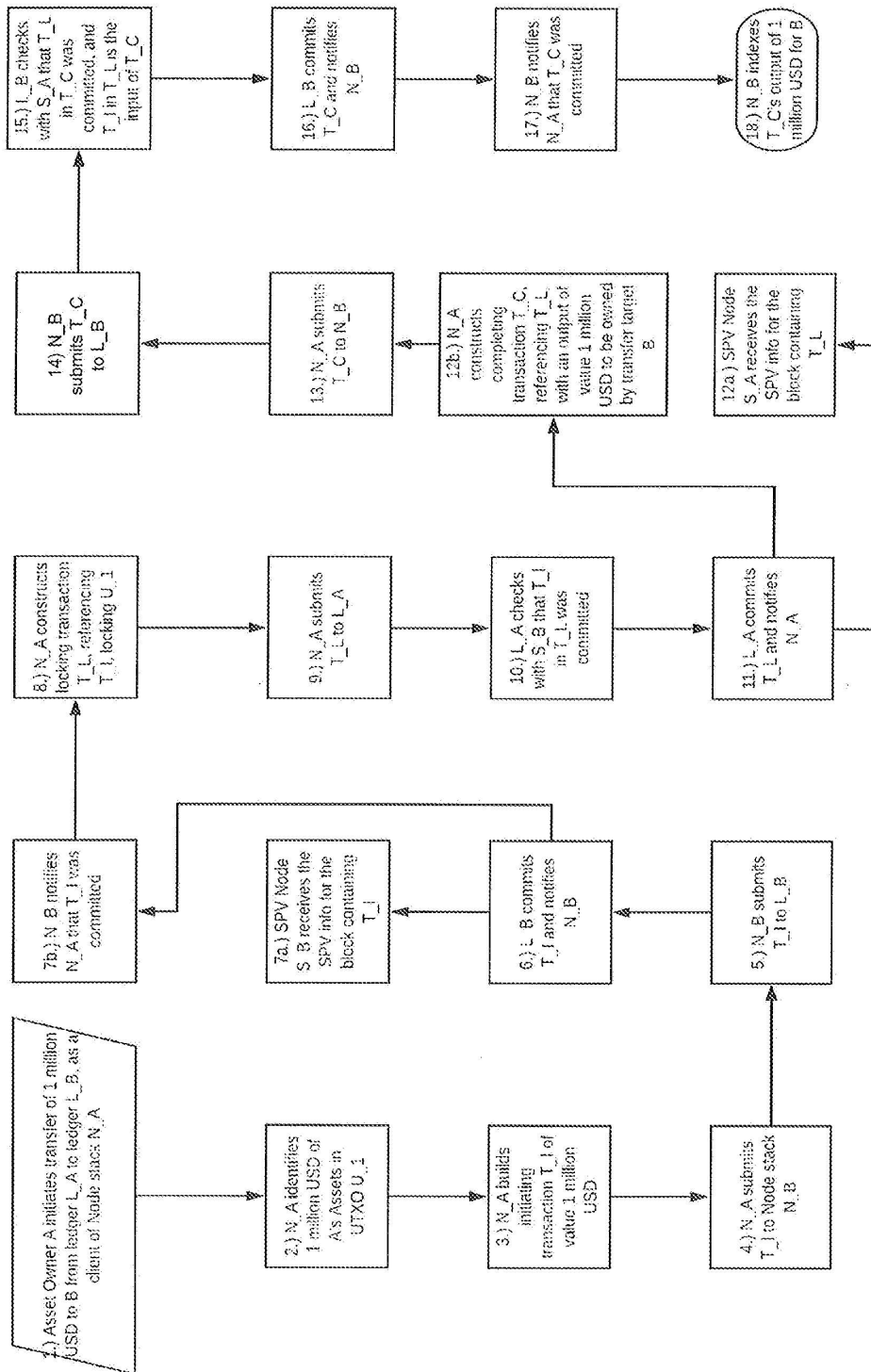


FIG. 6B



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.

FIG. 8

(steps 10 and 15 in FIG. 7)


800

```

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 InitiatingTransaction.Index.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.NotifyCommittedTransaction(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 = InitiatingTransactionIndex.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
4 // Adds the foreign transaction to an index so we can look up the foreign nodestack that submitted it
5 ForeignIndex.IndexForeignTransaction(Transaction, GetNodeStack(Source) )
6 // Submits the transaction (steps 5 and 14 in FIG. 7)
7 DistributedLedger.Submit(Transaction)
8
9 // Done
10 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        NotifyCommittedTransaction(Block.Transaction)
12    // Done
13    Exit

```

FIG. 14

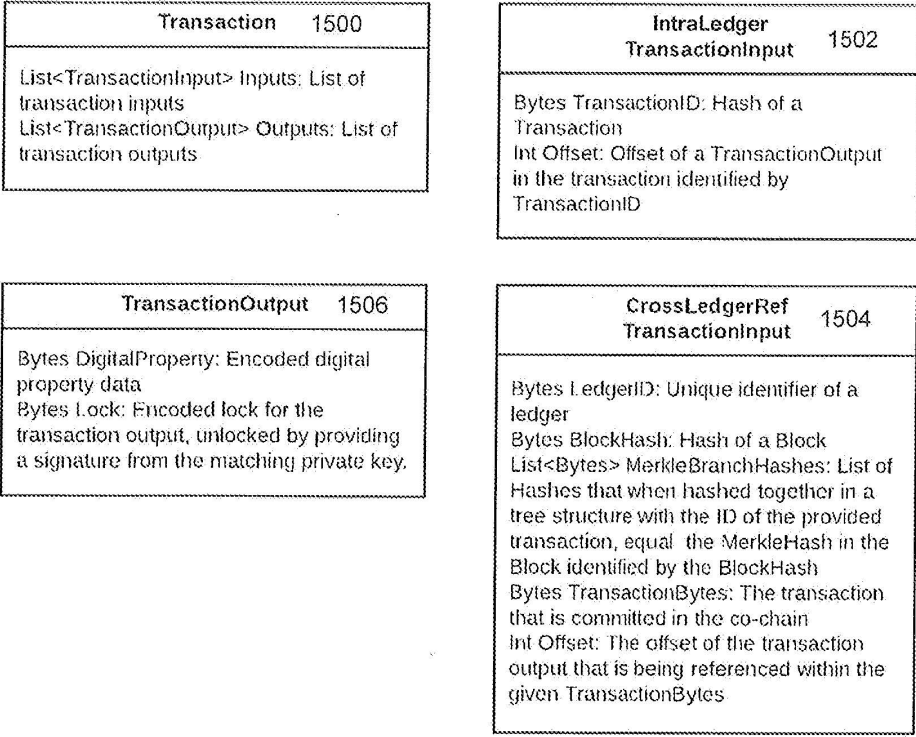


FIG. 15

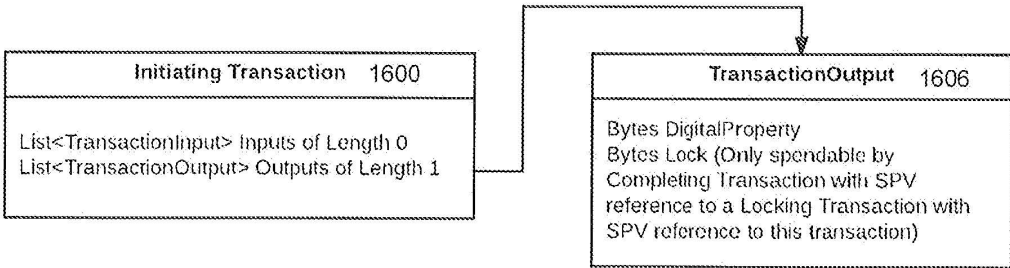


FIG. 16

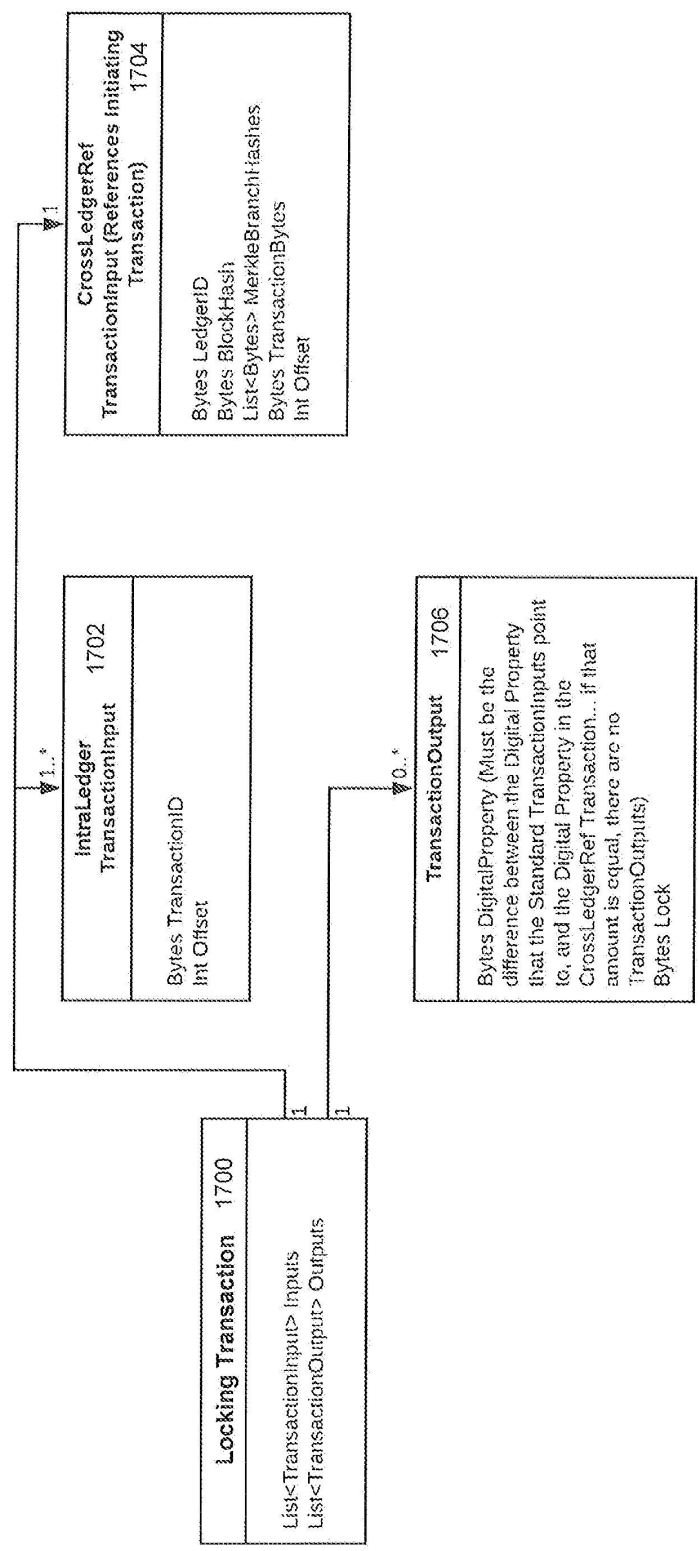


FIG. 17

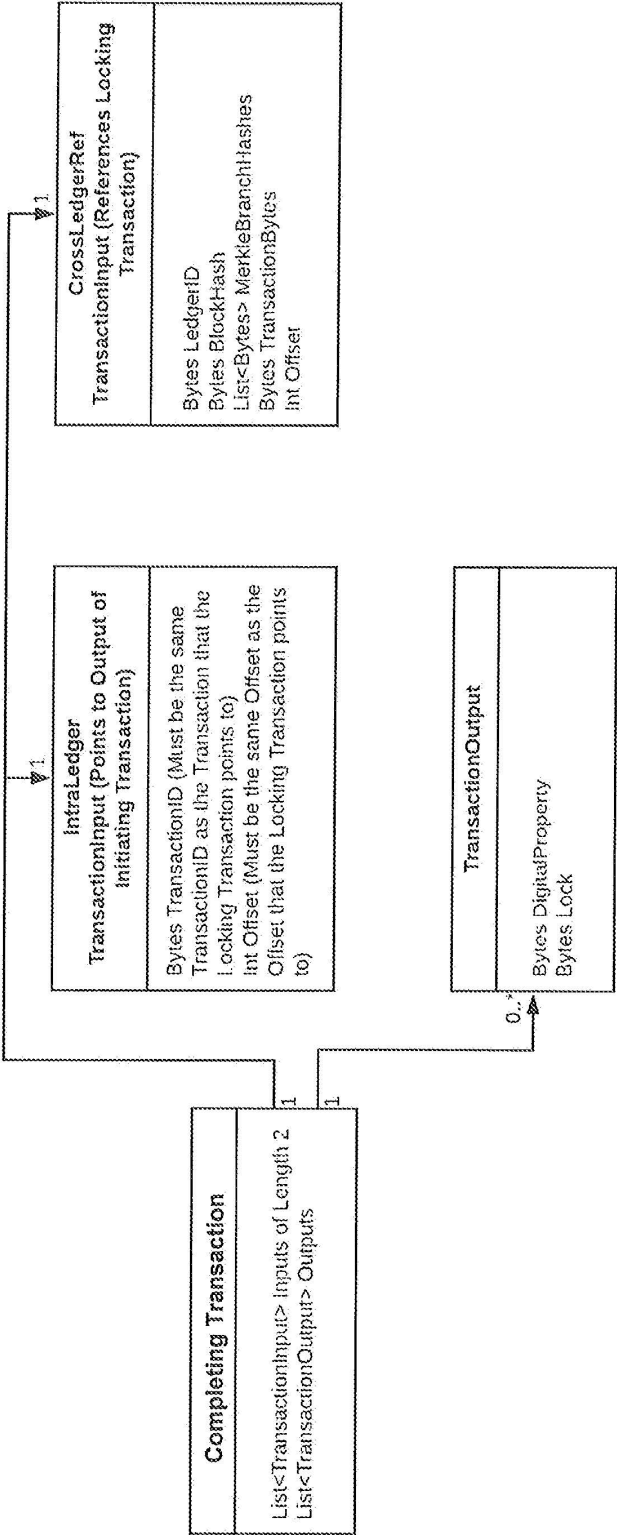


FIG. 18

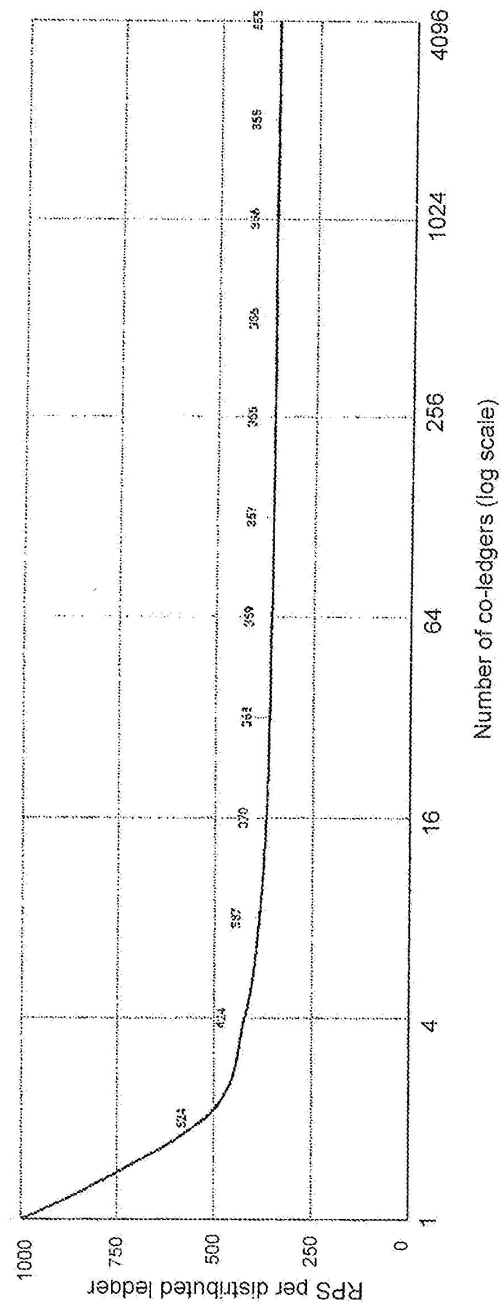


FIG. 19

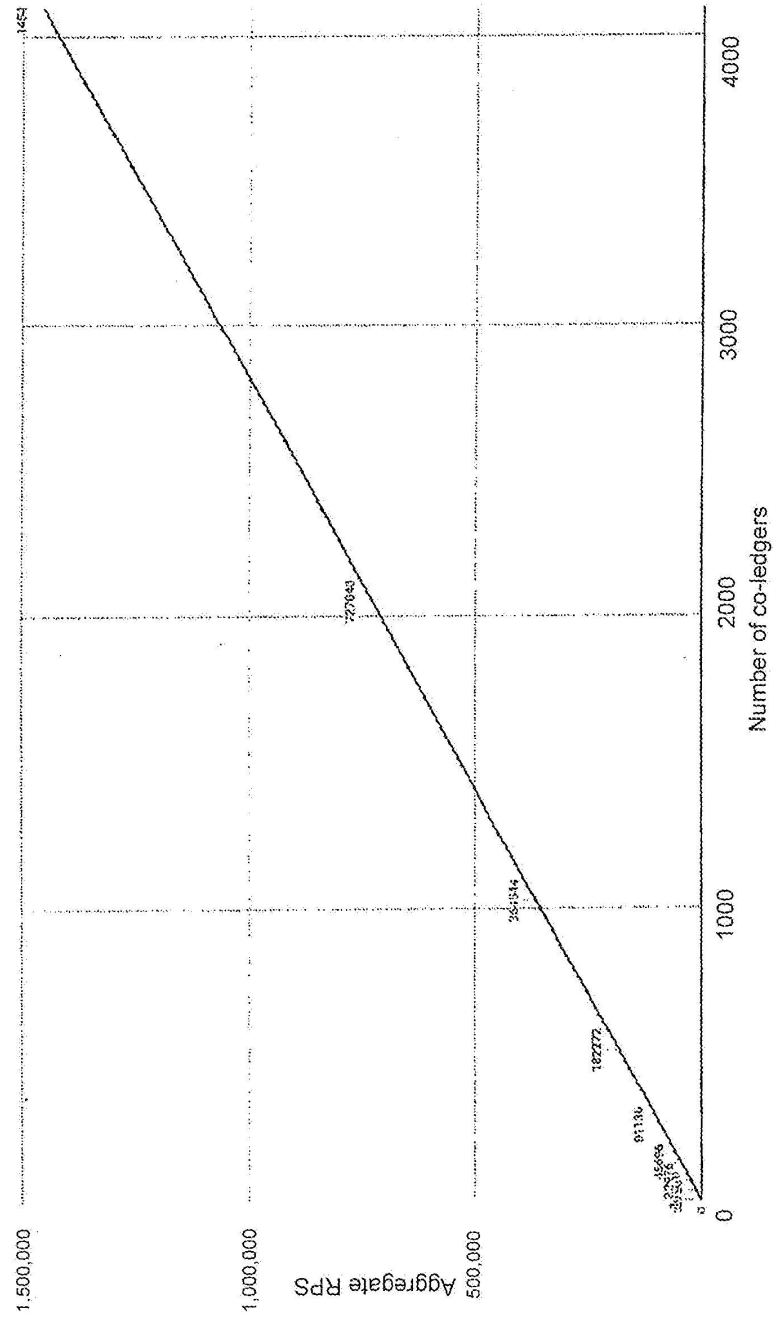


FIG. 20

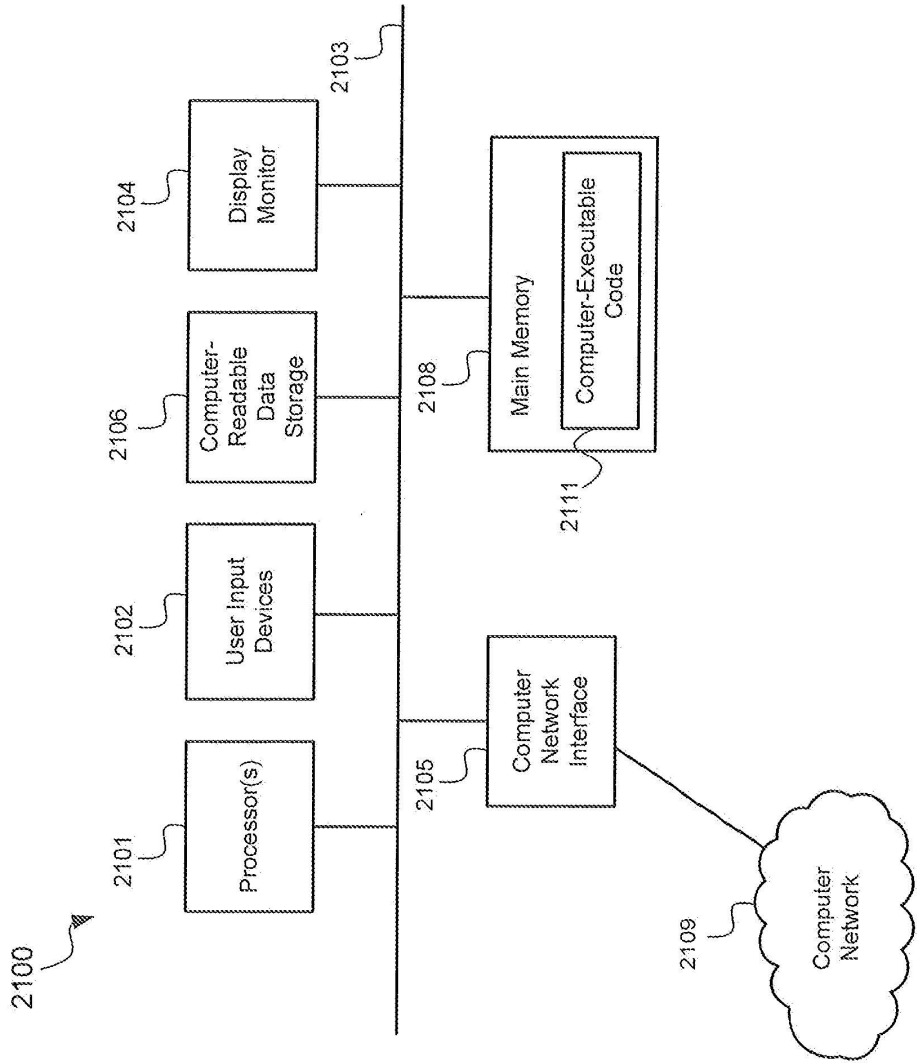


FIG. 21