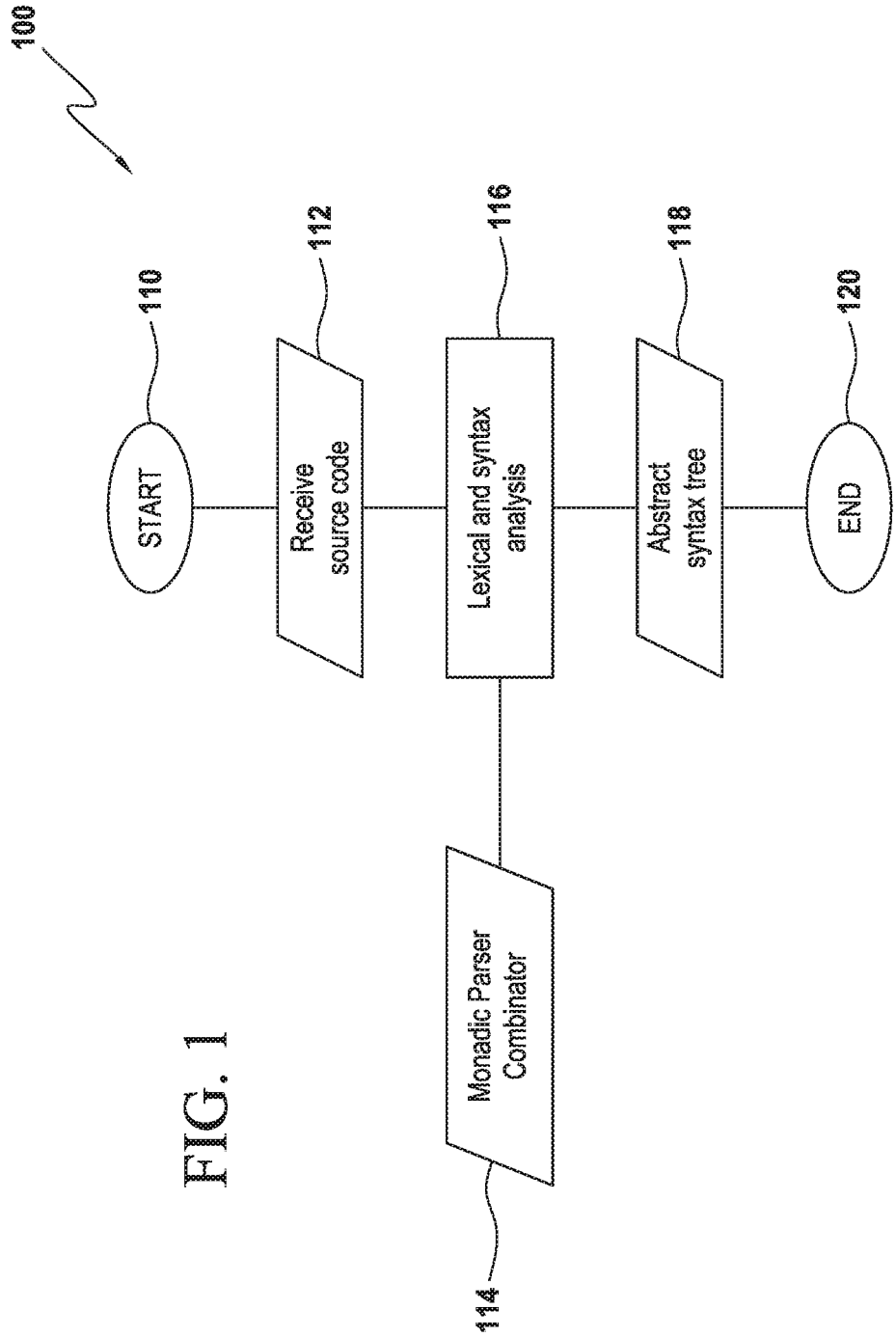

100
110
START
112
Receive source code
116
Lexical and syntax analysis
118
Abstract syntax tree
120
END
114
Monadic Parser Combinator

FIG. 1

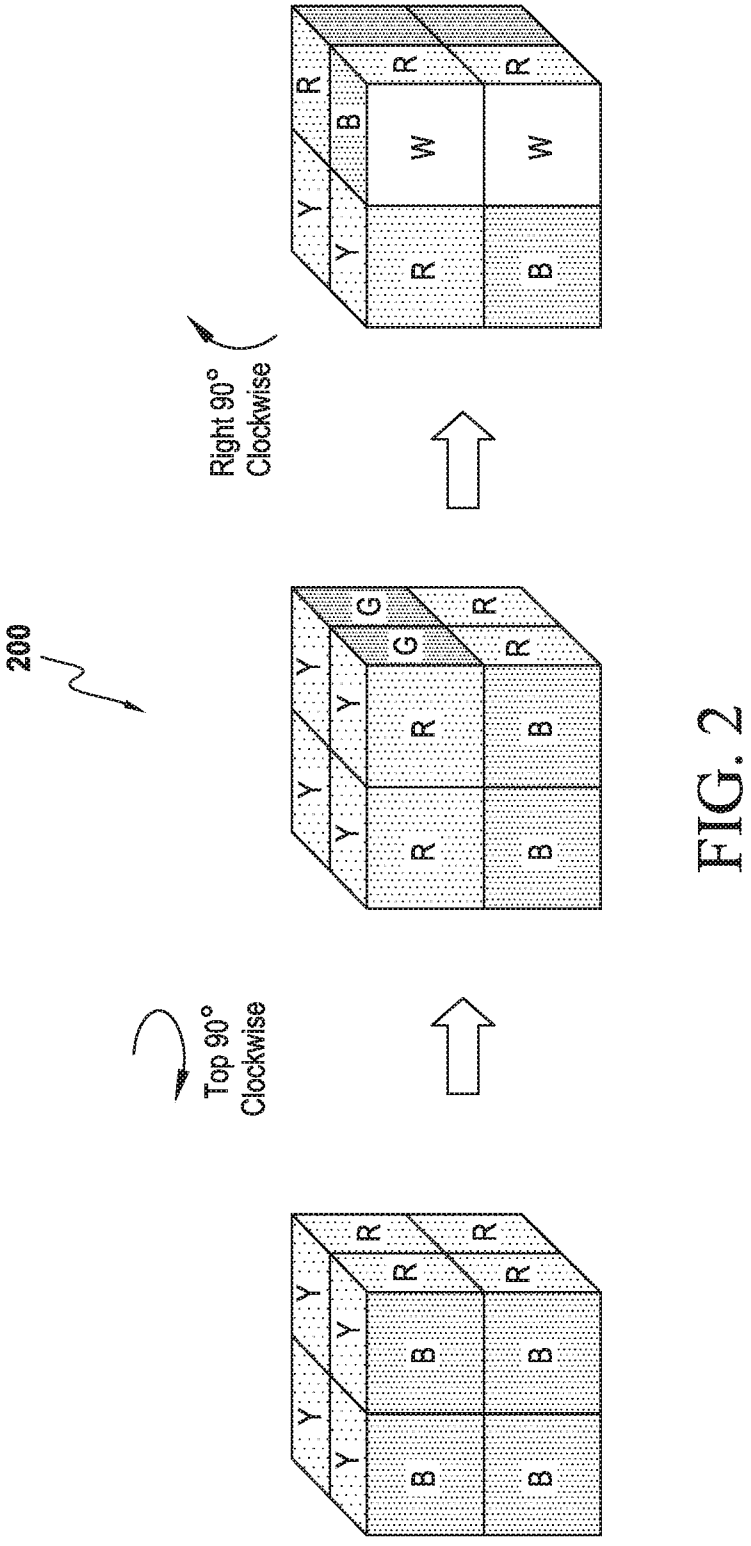


FIG. 2

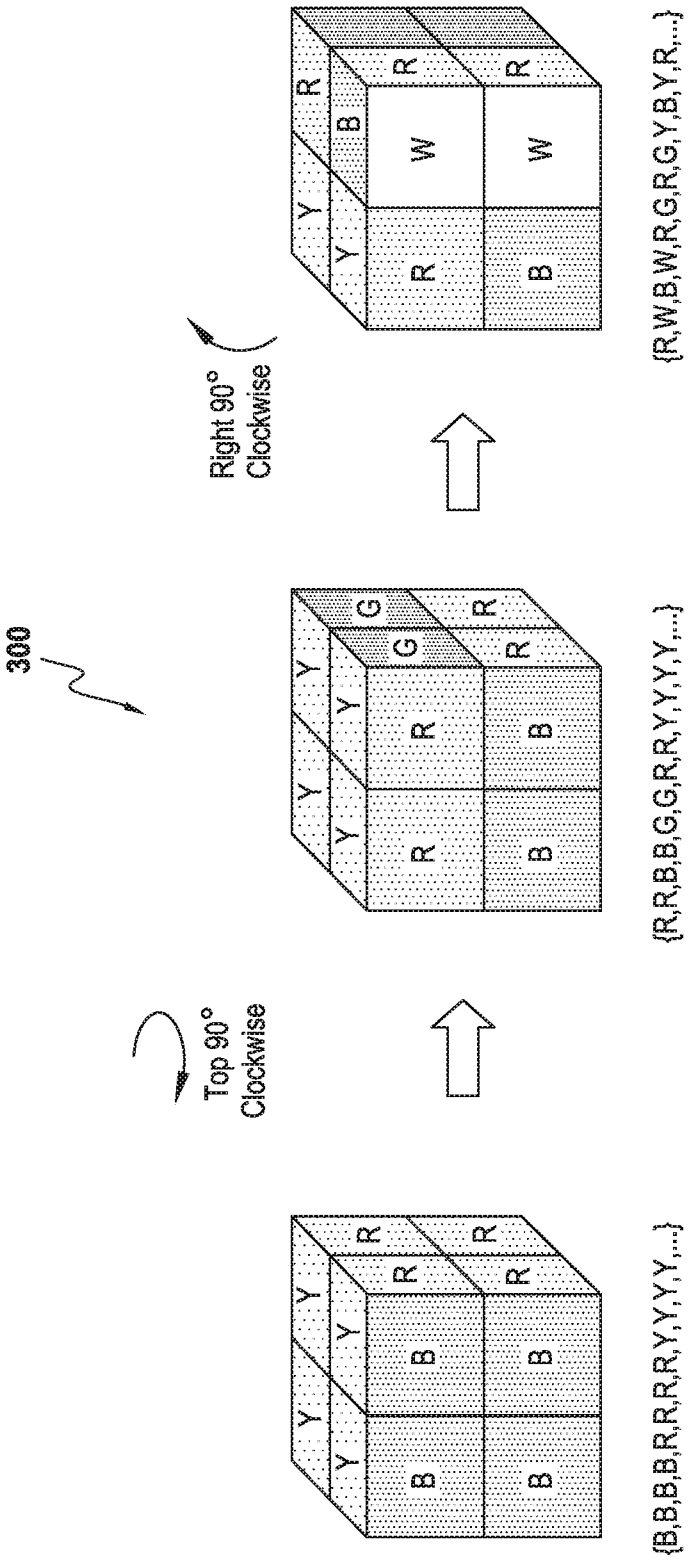

300
Top 90° Clockwise
Right 90° Clockwise
{B,B,B,R,R,R,Y,Y,Y,Y,...}
{R,R,B,B,G,G,R,R,Y,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}

FIG. 3

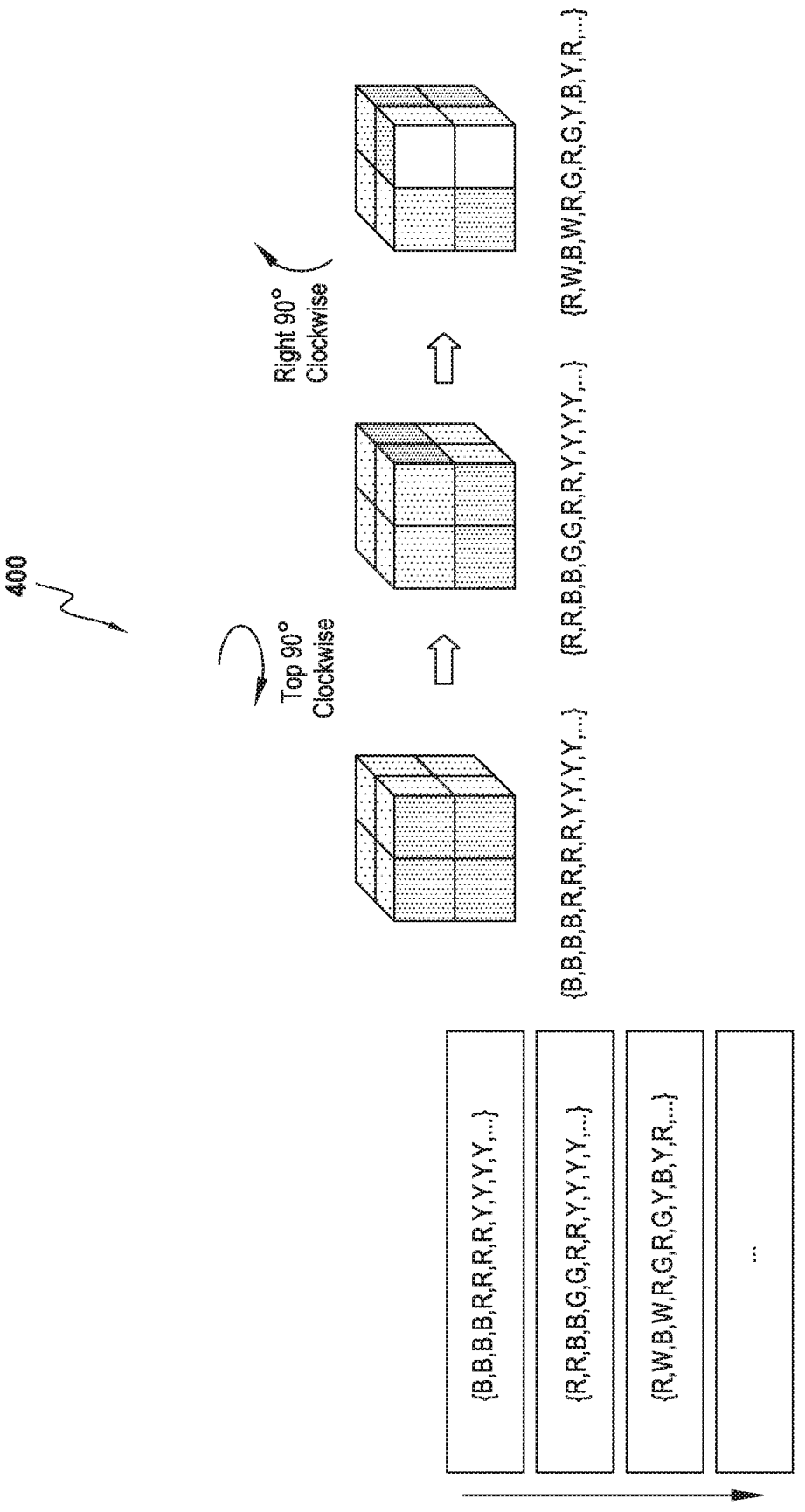

400
Top 90° Clockwise
Right 90° Clockwise
{B,B,B,B,R,R,R,R,Y,Y,Y,Y,...}
{R,R,B,B,G,G,R,R,Y,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
{B,B,B,B,R,R,R,R,Y,Y,Y,Y,...}
{R,R,B,B,G,G,R,R,Y,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
...

FIG. 4

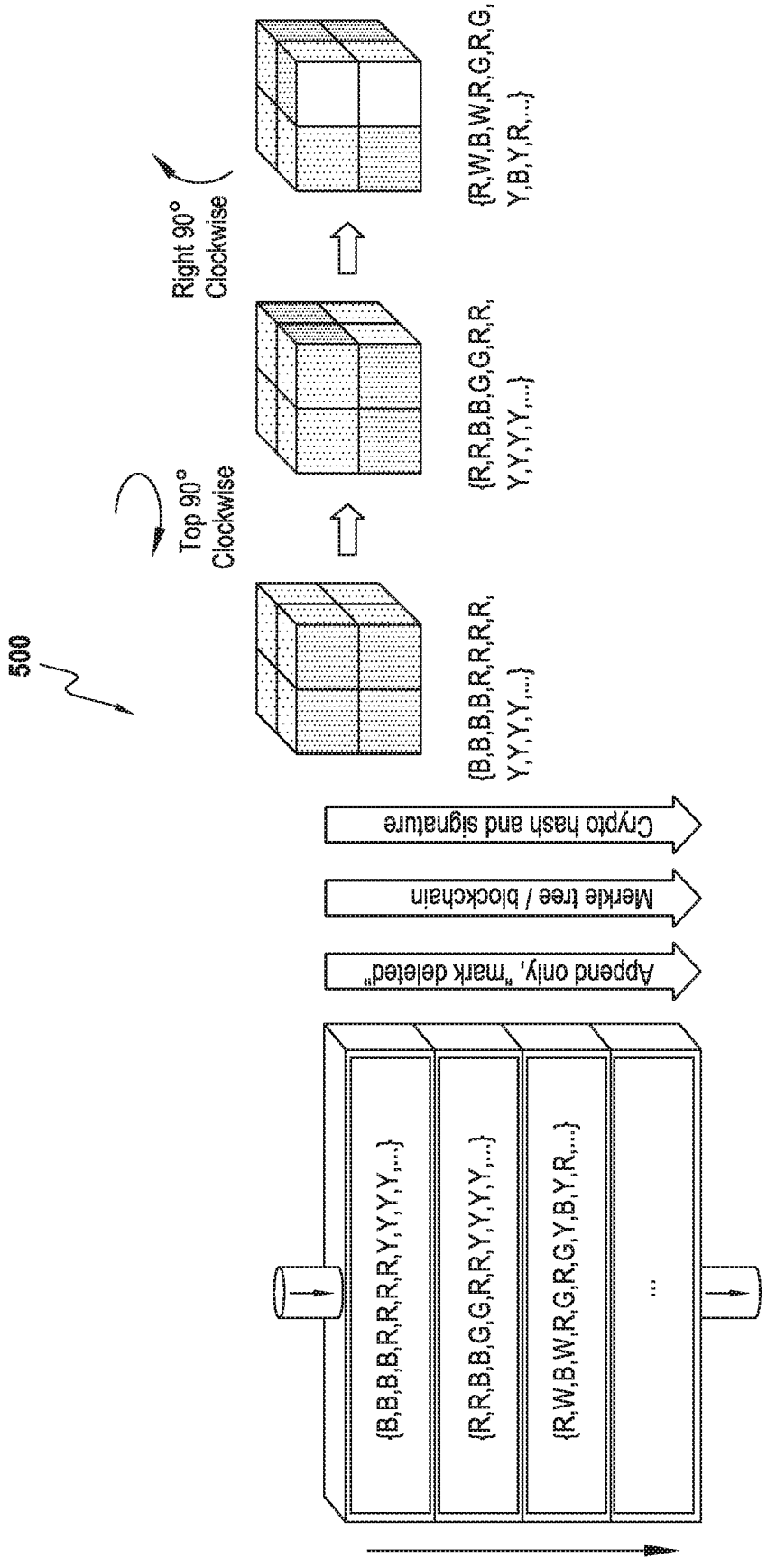

500
Top 90° Clockwise
Right 90° Clockwise
{B,B,B,R,R,R,R,R,
Y,Y,Y,Y,...}
{R,R,B,B,G,G,R,R,
Y,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,
Y,B,Y,R,...}
Crypto hash and signature
Merkle tree / blockchain
Append only, "mark deleted"
{B,B,B,R,R,R,R,Y,Y,Y,...}
{R,R,B,B,G,G,R,R,Y,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
...

FIG. 5

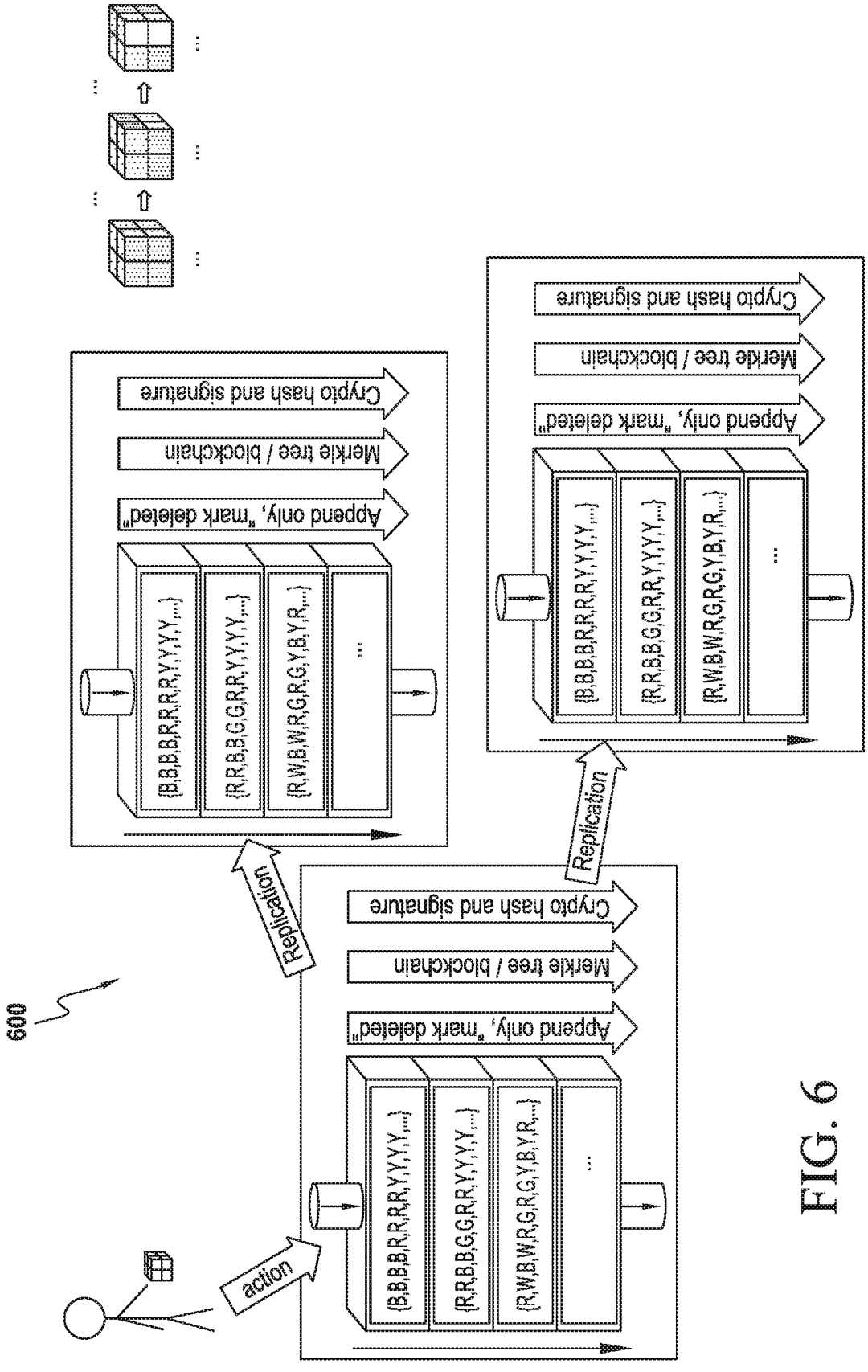

600
action
Append only, "mark deleted"
Merkle tree / blockchain
Crypto hash and signature
{B,B,B,R,R,R,Y,Y,Y,...}
{R,R,B,G,G,R,R,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
...
Replication
Replication

FIG. 6

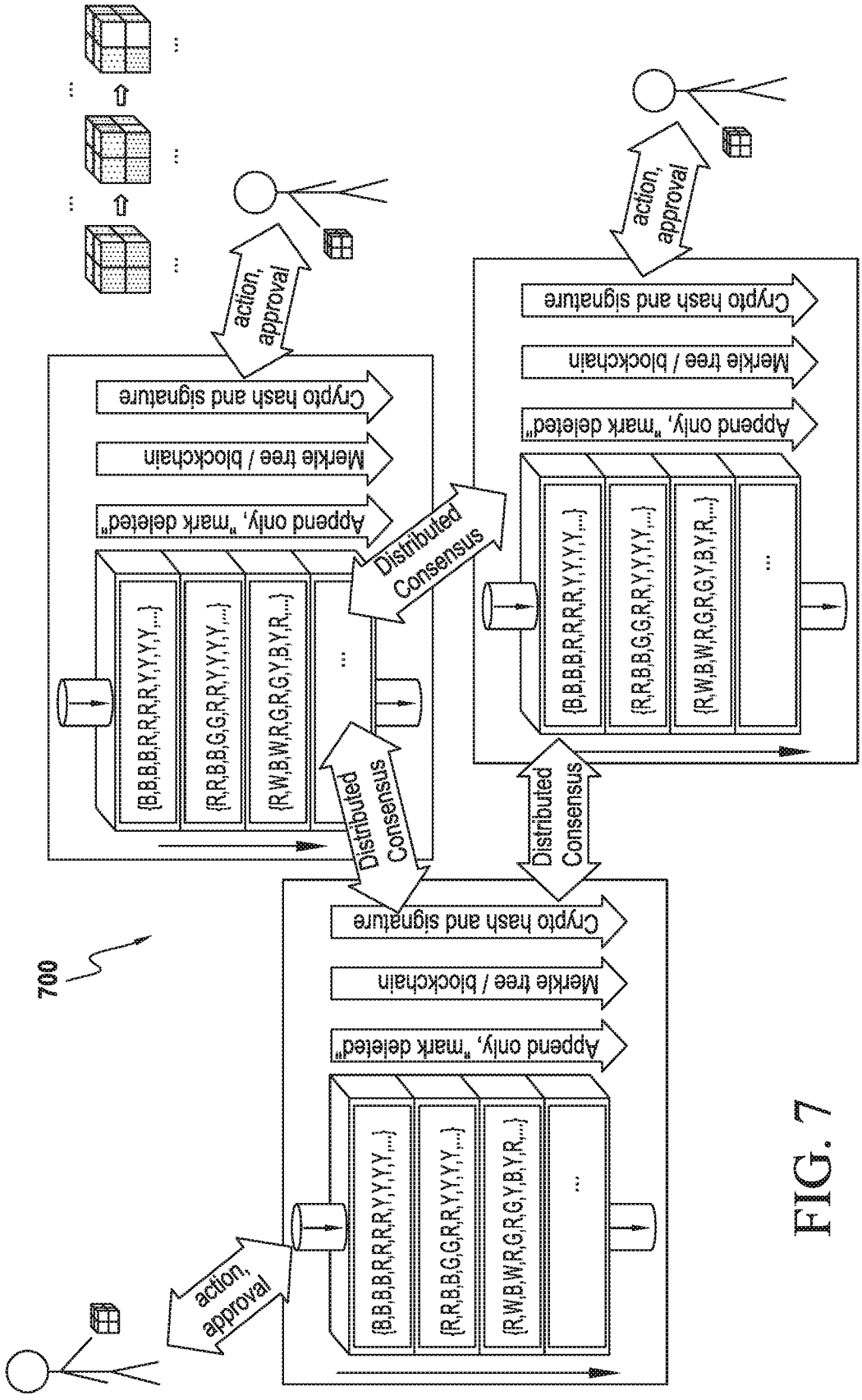

700
action, approval
Crypto hash and signature
Merkle tree / blockchain
Append only, "mark deleted"
{B,B,B,B,R,R,R,Y,Y,Y,...}
{R,R,B,B,G,G,R,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
...
Distributed Consensus

FIG. 7

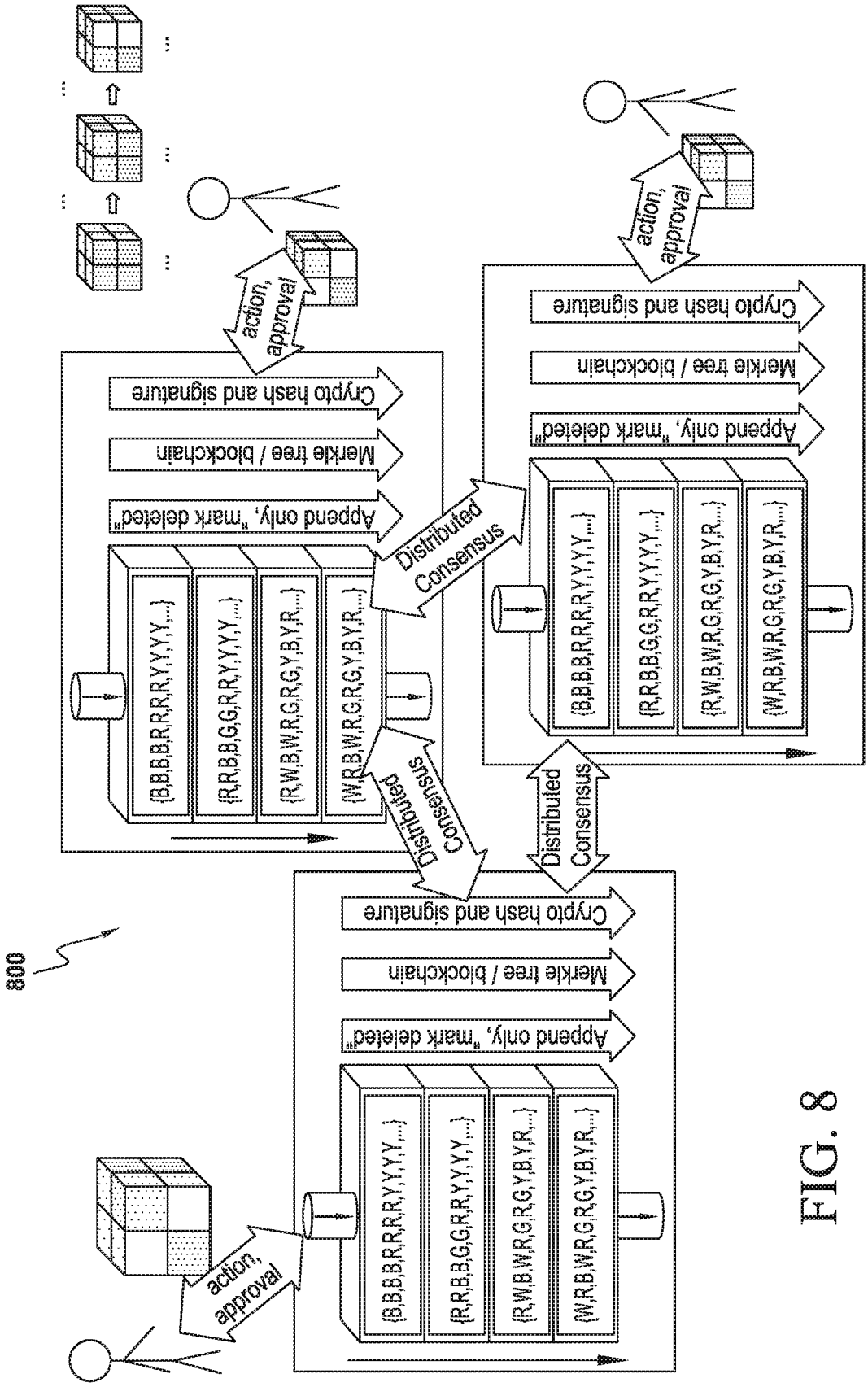

800
action, approval
Crypto hash and signature
Merkle tree / blockchain
Append only, "mark deleted"
{B,B,B,R,R,R,Y,Y,Y,...}
{R,R,B,B,G,R,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
{W,R,B,W,R,G,R,G,Y,B,Y,R,...}
Distributed Consensus
Distributed Consensus
Distributed Consensus

FIG. 8

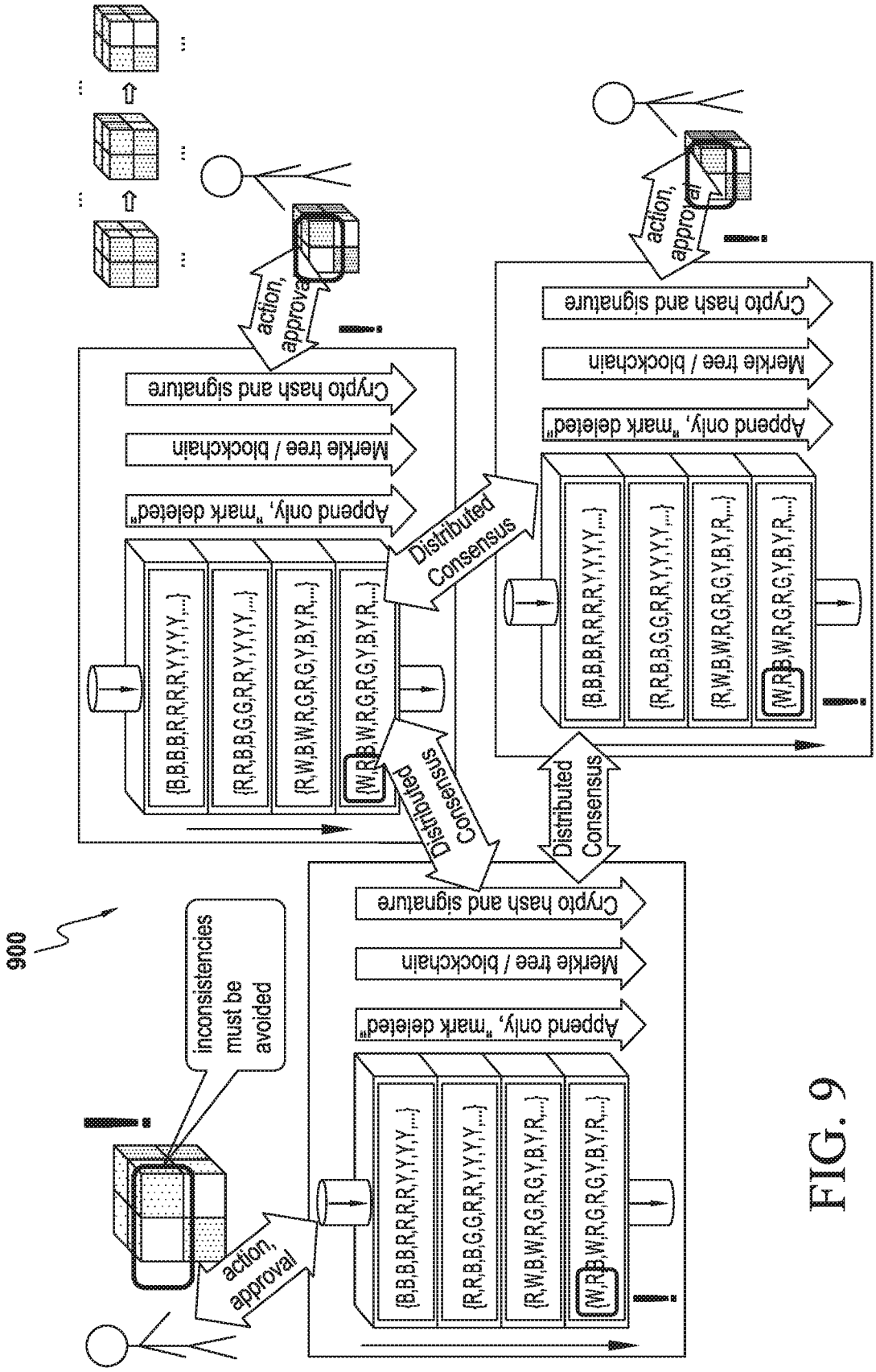

900
inconsistencies must be avoided
action, approval
Crypto hash and signature
Merkle tree / blockchain
Append only, "mark deleted"
{B,B,B,R,R,R,Y,Y,Y,...}
{R,R,B,B,G,G,R,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
{W,R,B,W,R,G,R,G,Y,B,Y,R,...}
Distributed Consensus
Distributed Consensus
Distributed Consensus
action, approval
action, approval
Crypto hash and signature
Merkle tree / blockchain
Append only, "mark deleted"
{B,B,B,R,R,R,Y,Y,Y,...}
{R,R,B,B,G,G,R,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
{W,R,B,W,R,G,R,G,Y,B,Y,R,...}
Crypto hash and signature
Merkle tree / blockchain
Append only, "mark deleted"
{B,B,B,R,R,R,Y,Y,Y,...}
{R,R,B,B,G,G,R,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
{W,R,B,W,R,G,R,G,Y,B,Y,R,...}

FIG. 9

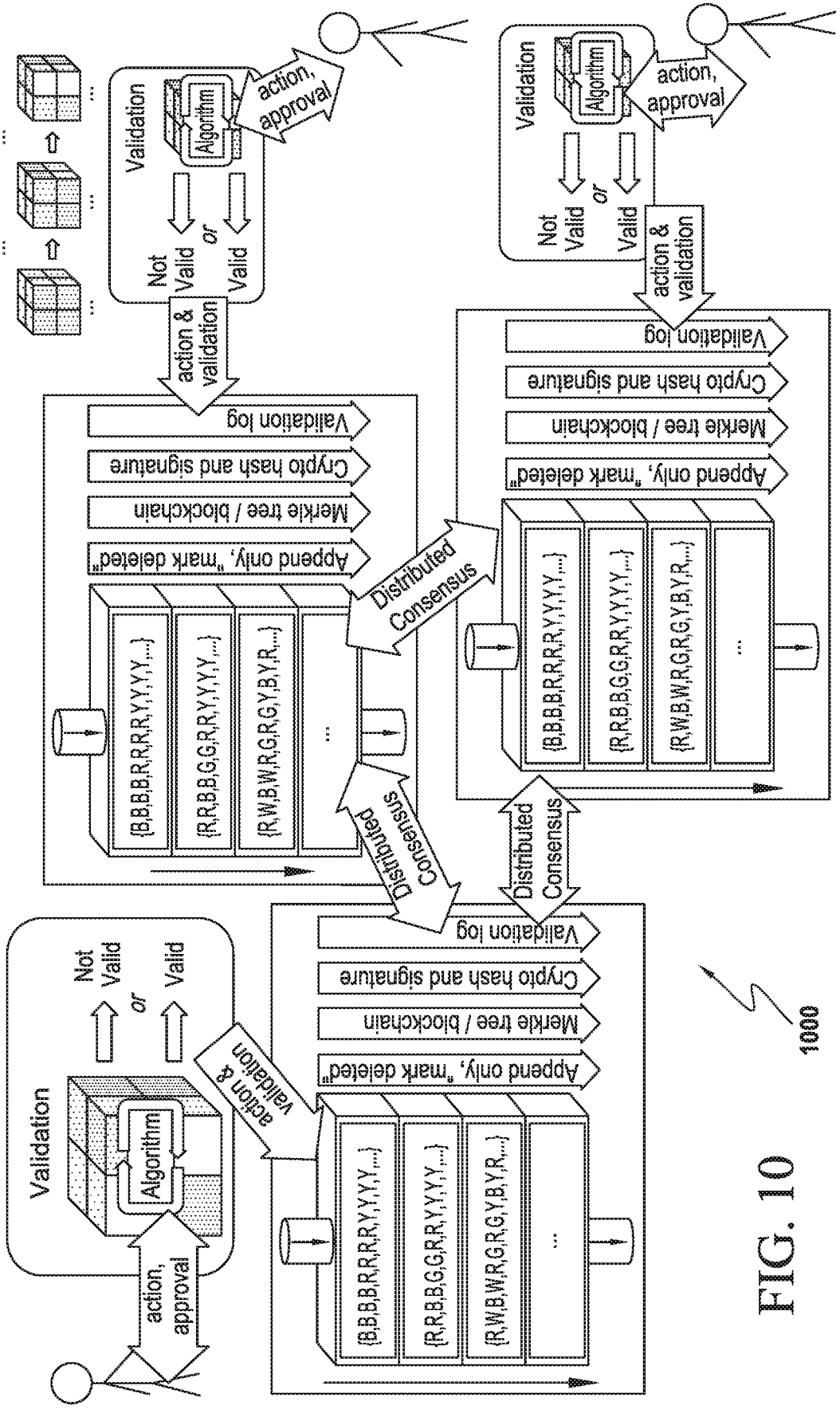

Validation
Algorithm
action, approval
Not Valid
or
Valid
action & validation
Validation log
Crypto hash and signature
Merkle tree / blockchain
Append only, "mark deleted"
{B,B,B,R,R,R,Y,Y,Y,...}
{R,R,B,B,G,G,R,Y,Y,Y,...}
{R,W,B,W,R,G,R,G,Y,B,Y,R,...}
...
Distributed Consensus
1000

FIG. 10

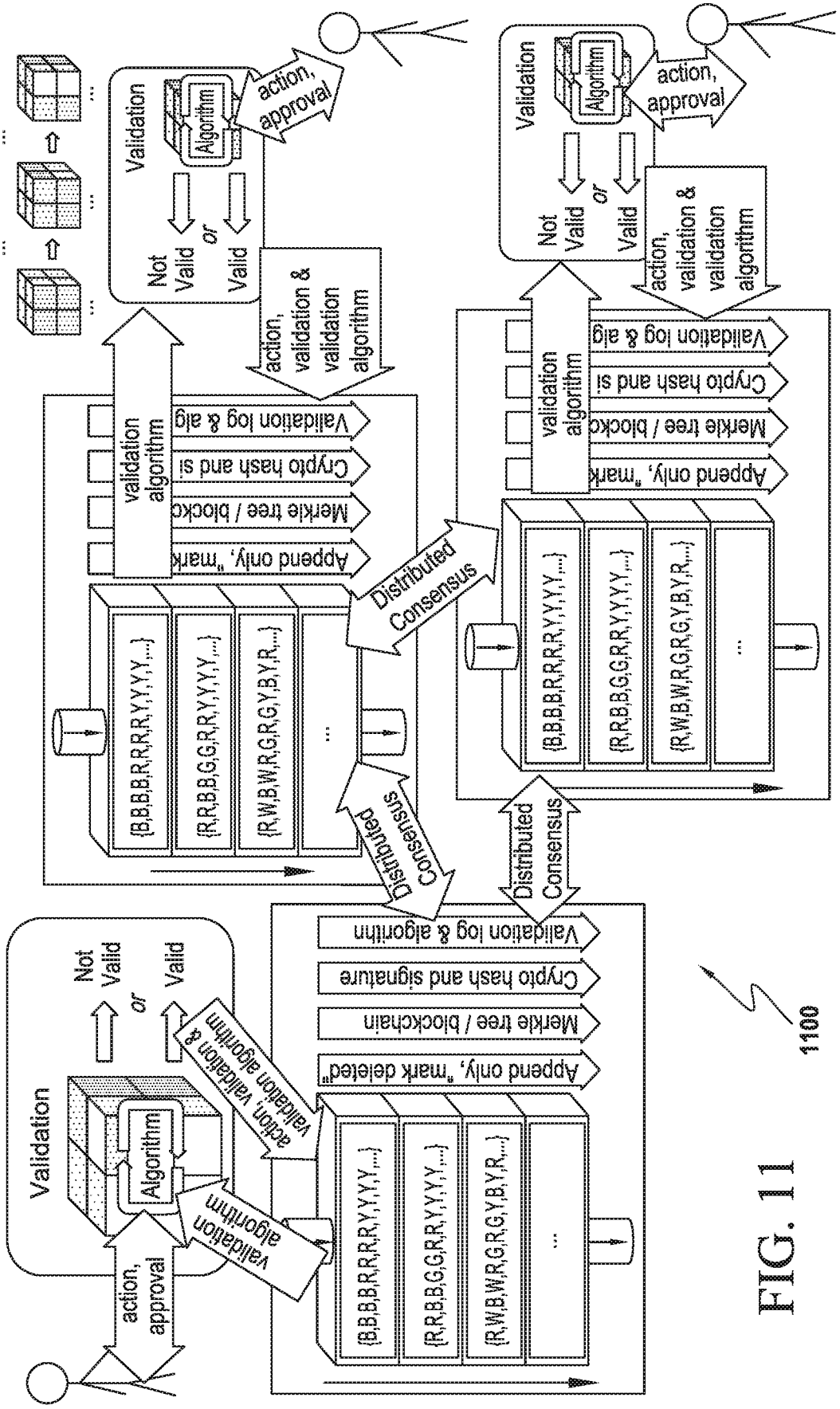

Validation
Algorithm
action, approval
Not Valid
or
Valid
validation algorithm
action, validation & validation algorithm
Append only, "mark deleted"
Merkle tree / blockchain
Crypto hash and signature
Validation log & algorithm
Distributed Consensus
1100

FIG. 11

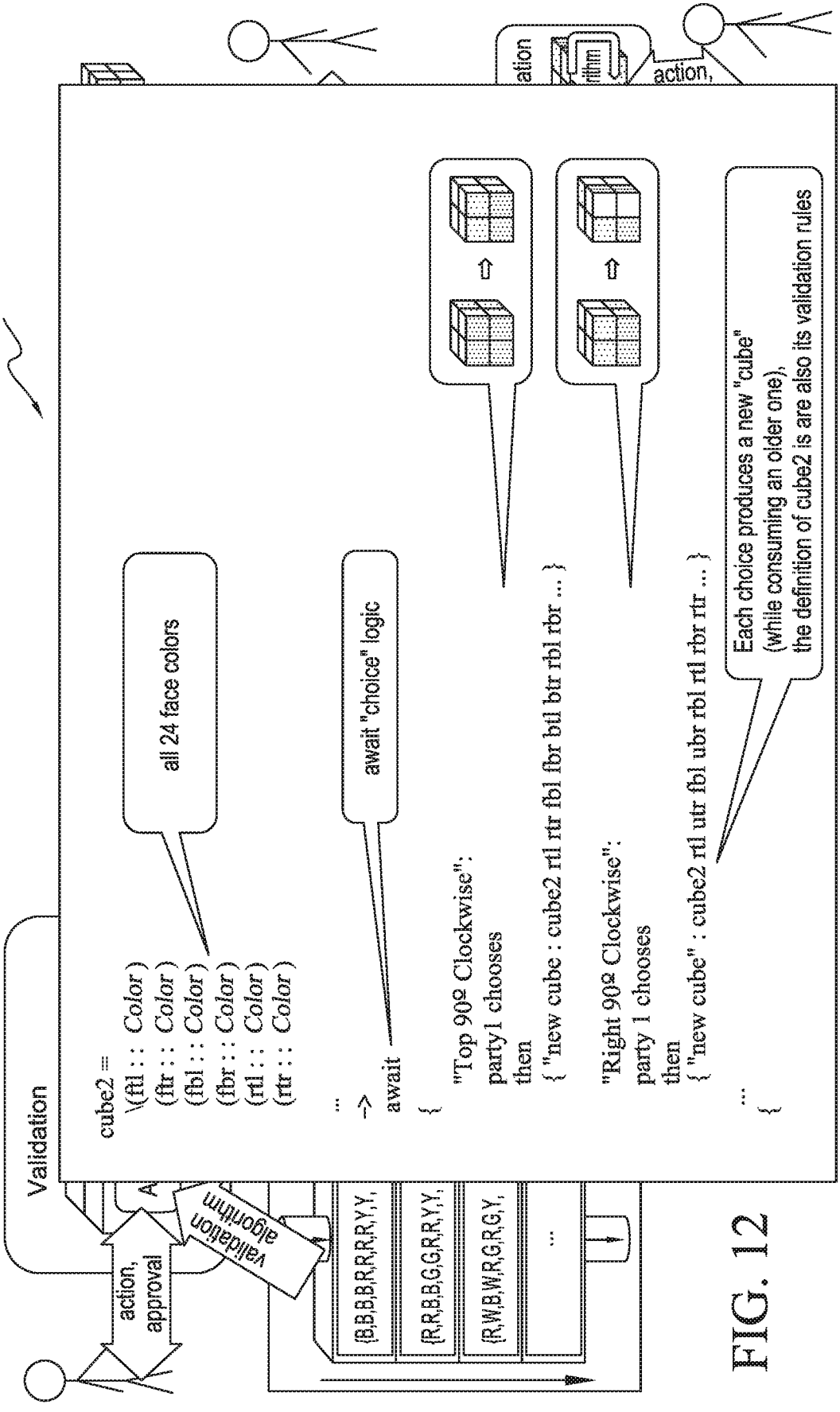

1200
Validation
action, approval
validation algorithm
{B,B,B,R,R,R,Y,Y,
{R,R,B,G,G,R,Y,Y,
{R,W,B,W,R,G,G,Y,
...
cube2 =
\(ftl : : Color)
(ftr : : Color)
(fbl : : Color)
(fbr : : Color)
(rtl : : Color)
(rtr : : Color)
all 24 face colors
...
-> await
await "choice" logic
{
"Top 90º Clockwise":
party1 chooses
then
{ "new cube : cube2 rtl rtr fbl fbr btl btr rbl rbr ... }
"Right 90º Clockwise":
party 1 chooses
then
{ "new cube" : cube2 rtl utr fbl ubr rbl rtl rbr rtr ... }
...
{
Each choice produces a new "cube"
(while consuming an older one),
the definition of cube2 is are also its validation rules
ation
rithm
action,

FIG. 12

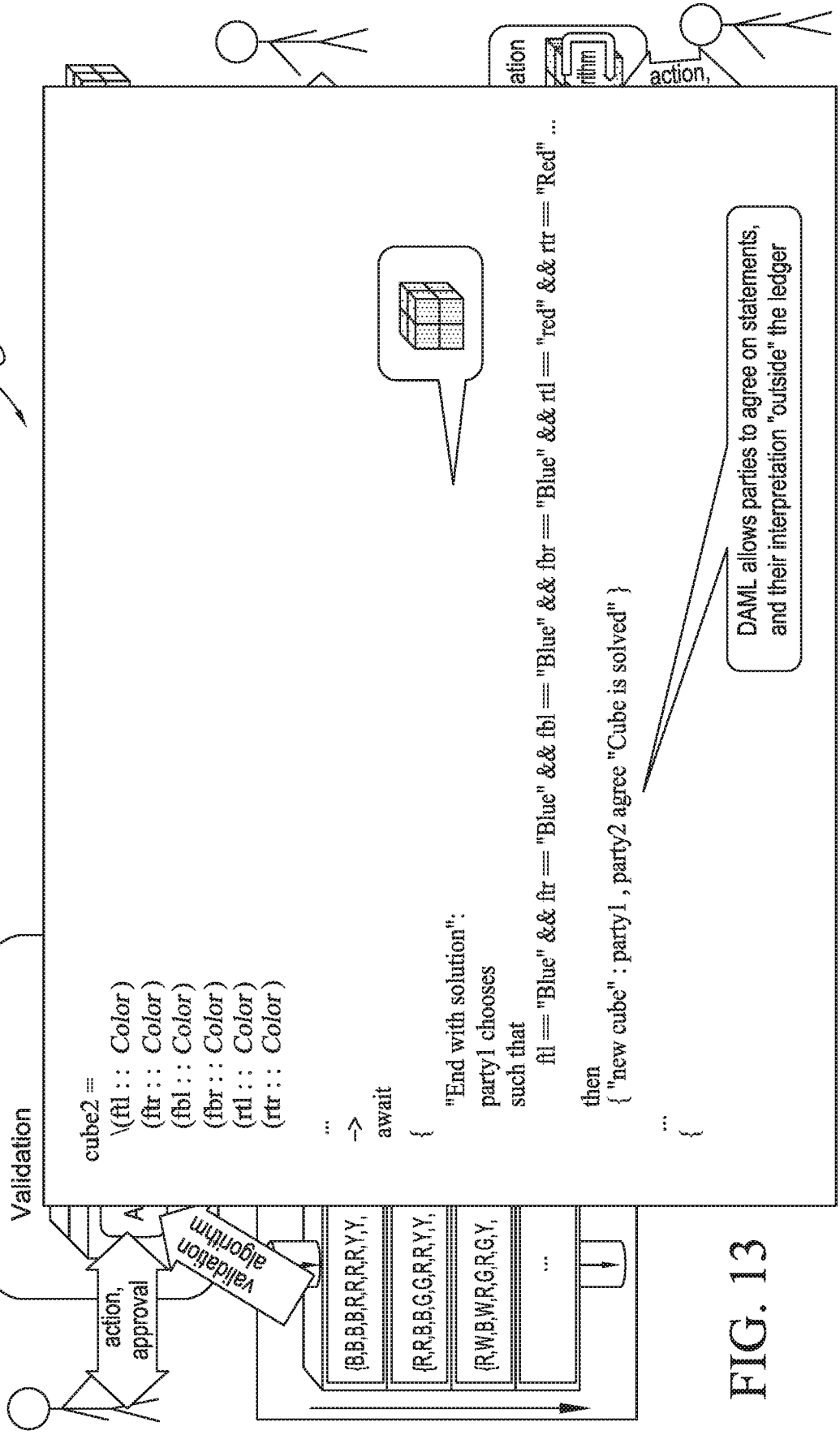

1300
Validation
action,
approval
validation
algorithm
{B,B,B,R,R,R,Y,Y,
{R,R,B,G,G,R,Y,Y,
{R,W,B,W,R,G,G,Y,
...
cube2 =
\(ftl :: Color)
(ftr :: Color)
(fbl :: Color)
(fbr :: Color)
(rtl :: Color)
(rtr :: Color)
...
->
await
{
"End with solution":
party1 chooses
such that
ftl == "Blue" && ftr == "Blue" && fbl == "Blue" && fbr == "Blue" && rtl == "red" && rtr == "Red" ...
then
{ "new cube" : party1 , party2 agree "Cube is solved" }
...
{
DAML allows parties to agree on statements,
and their interpretation "outside" the ledger

FIG. 13

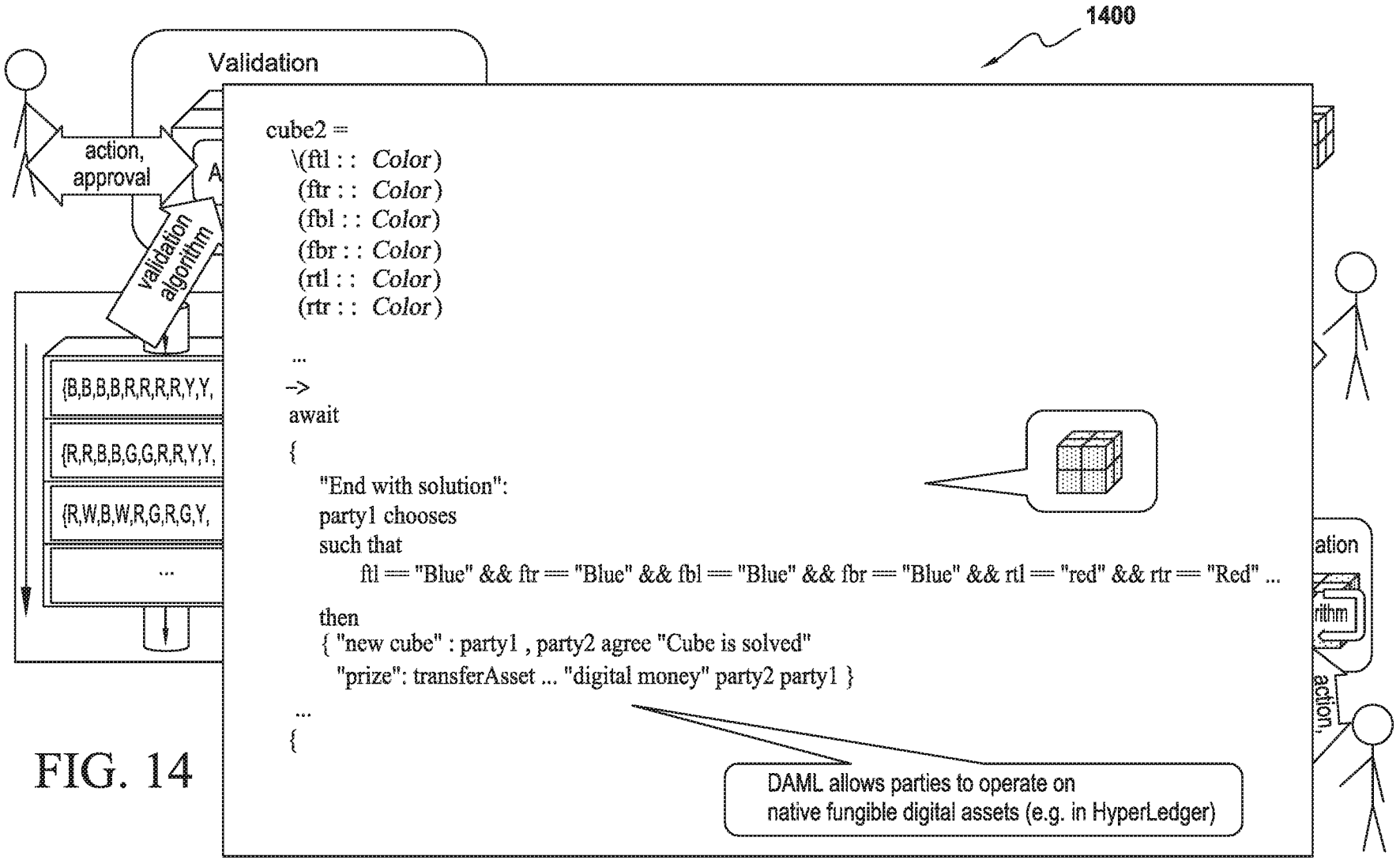

1400
Validation
action,
approval
validation
algorithm
{B,B,B,B,R,R,R,R,Y,Y,
{R,R,B,B,G,G,R,R,Y,Y,
{R,W,B,W,R,G,R,G,Y,
...
cube2 =
\(ftl : : Color)
(ftr : : Color)
(fbl : : Color)
(fbr : : Color)
(rtl : : Color)
(rtr : : Color)
...
->
await
{
"End with solution":
party1 chooses
such that
ftl == "Blue" && ftr == "Blue" && fbl == "Blue" && fbr == "Blue" && rtl == "red" && rtr == "Red" ...
then
{ "new cube" : party1 , party2 agree "Cube is solved"
"prize": transferAsset ... "digital money" party2 party1 }
...
{
DAML allows parties to operate on
native fungible digital assets (e.g. in HyperLedger)
ation
ithm
action,

FIG. 14

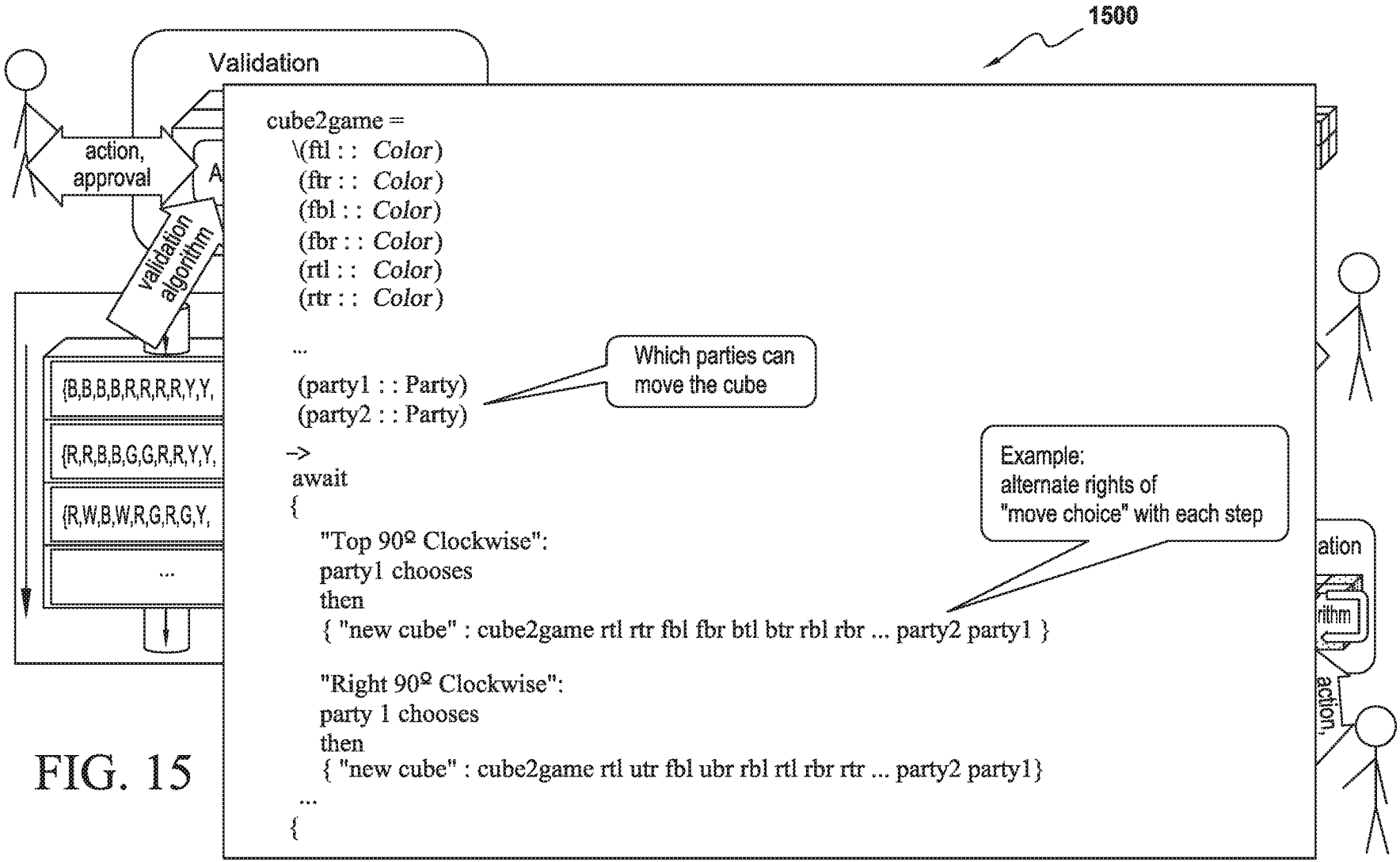

1500
Validation
action,
approval
validation
algorithm
{B,B,B,B,R,R,R,R,Y,Y,
{R,R,B,B,G,G,R,R,Y,Y,
{R,W,B,W,R,G,R,G,Y,
...
cube2game =
\(ftl : : Color)
(ftr : : Color)
(fbl : : Color)
(fbr : : Color)
(rtl : : Color)
(rtr : : Color)
...
(party1 : : Party)
(party2 : : Party)
Which parties can move the cube
->
await
{
"Top 90º Clockwise":
party1 chooses
then
{ "new cube" : cube2game rtl rtr fbl fbr btl btr rbl rbr ... party2 party1 }
Example:
alternate rights of
"move choice" with each step
"Right 90º Clockwise":
party 1 chooses
then
{ "new cube" : cube2game rtl utr fbl ubr rbl rtl rbr rtr ... party2 party1}
...
{
ation
ithm
action,

FIG. 15

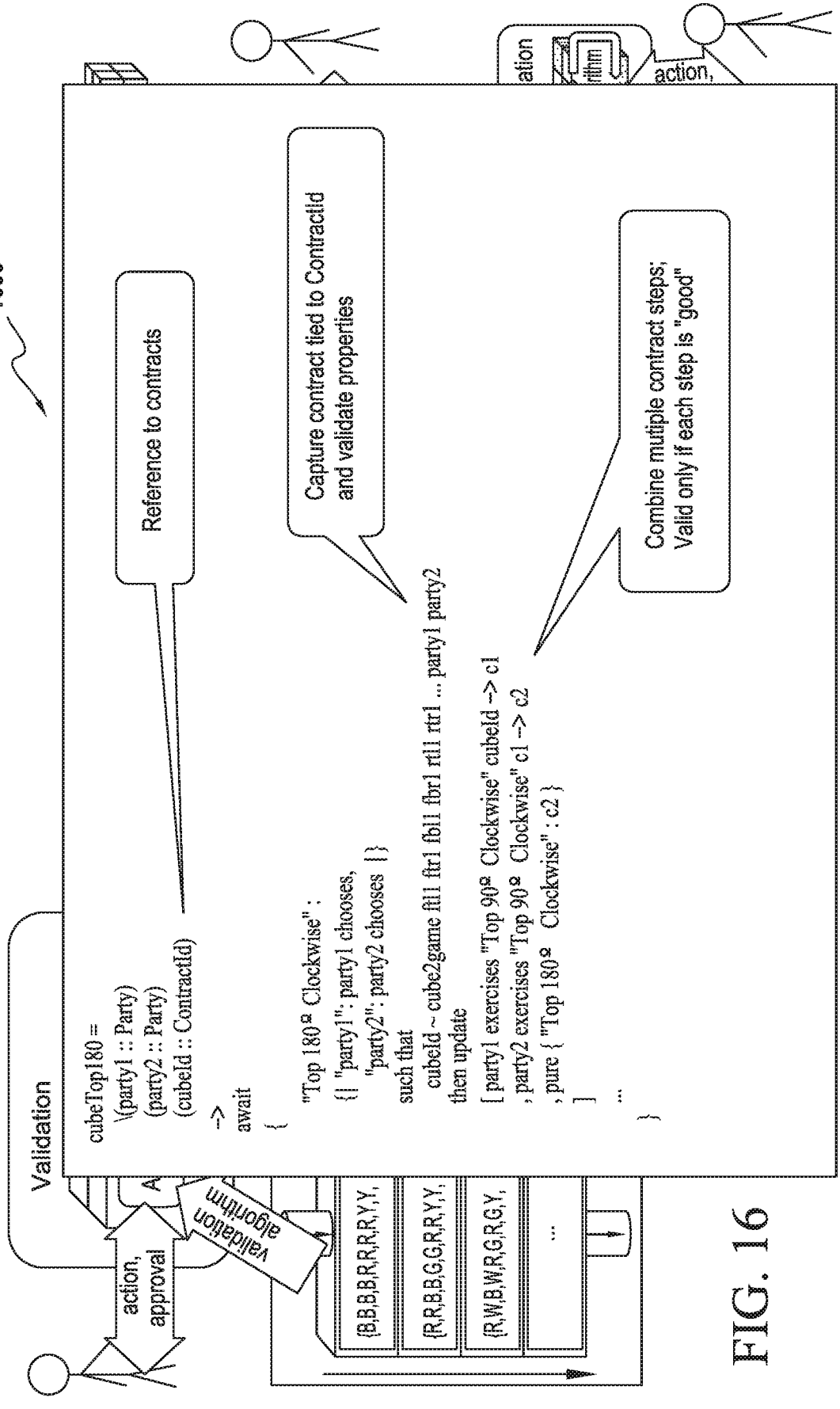

1600
Validation
action, approval
Validation algorithm
{B,B,B,R,R,R,Y,Y,
{R,R,B,G,G,R,Y,Y,
{R,W,B,W,R,G,G,Y,
...
cubeTop180 =
\(party1 :: Party)
(party2 :: Party)
(cubeId :: ContractId)
->
await
{
"Top 180º Clockwise" :
{| "party1": party1 chooses,
"party2": party2 chooses |}
such that
cubeId ~ cube2game fll1 ftr1 fbl1 fbr1 rtl1 rtr1 ... party1 party2
then update
[party1 exercises "Top 90º Clockwise" cubeId -> c1
, party2 exercises "Top 90º Clockwise" c1 -> c2
, pure { "Top 180º Clockwise" : c2 }
]
...
}
Reference to contracts
Capture contract tied to ContractId and validate properties
Combine mutiple contract steps; Valid only if each step is "good"
ation
rithm
action,

FIG. 16

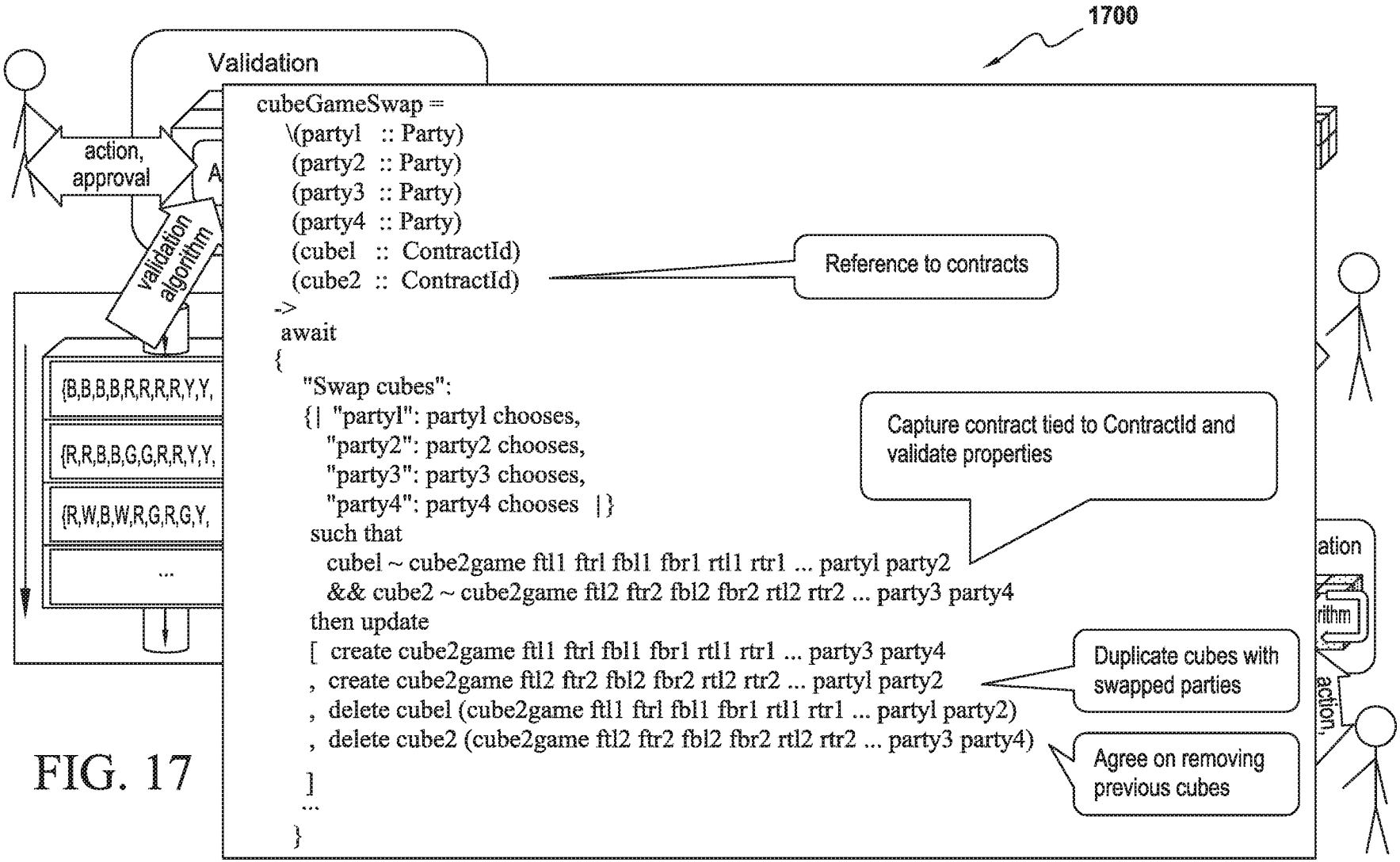

1700
Validation
action, approval
validation algorithm
{B,B,B,B,R,R,R,R,Y,Y,
{R,R,B,B,G,G,R,R,Y,Y,
{R,W,B,W,R,G,R,G,Y,
...
cubeGameSwap =
\(party1 :: Party)
(party2 :: Party)
(party3 :: Party)
(party4 :: Party)
(cube1 :: ContractId)
(cube2 :: ContractId)
Reference to contracts
-\>
await
{
"Swap cubes":
{| "party1": party1 chooses,
"party2": party2 chooses,
"party3": party3 chooses,
"party4": party4 chooses |}
such that
cube1 ~ cube2game ftl1 ftr1 fbl1 fbr1 rtl1 rtr1 ... party1 party2
&& cube2 ~ cube2game ftl2 ftr2 fbl2 fbr2 rtl2 rtr2 ... party3 party4
Capture contract tied to ContractId and validate properties
then update
[create cube2game ftl1 ftr1 fbl1 fbr1 rtl1 rtr1 ... party3 party4
, create cube2game ftl2 ftr2 fbl2 fbr2 rtl2 rtr2 ... party1 party2
Duplicate cubes with swapped parties
, delete cube1 (cube2game ftl1 ftr1 fbl1 fbr1 rtl1 rtr1 ... party1 party2)
, delete cube2 (cube2game ftl2 ftr2 fbl2 fbr2 rtl2 rtr2 ... party3 party4)
Agree on removing previous cubes
]
...
}
ation
ithm
action,

FIG. 17

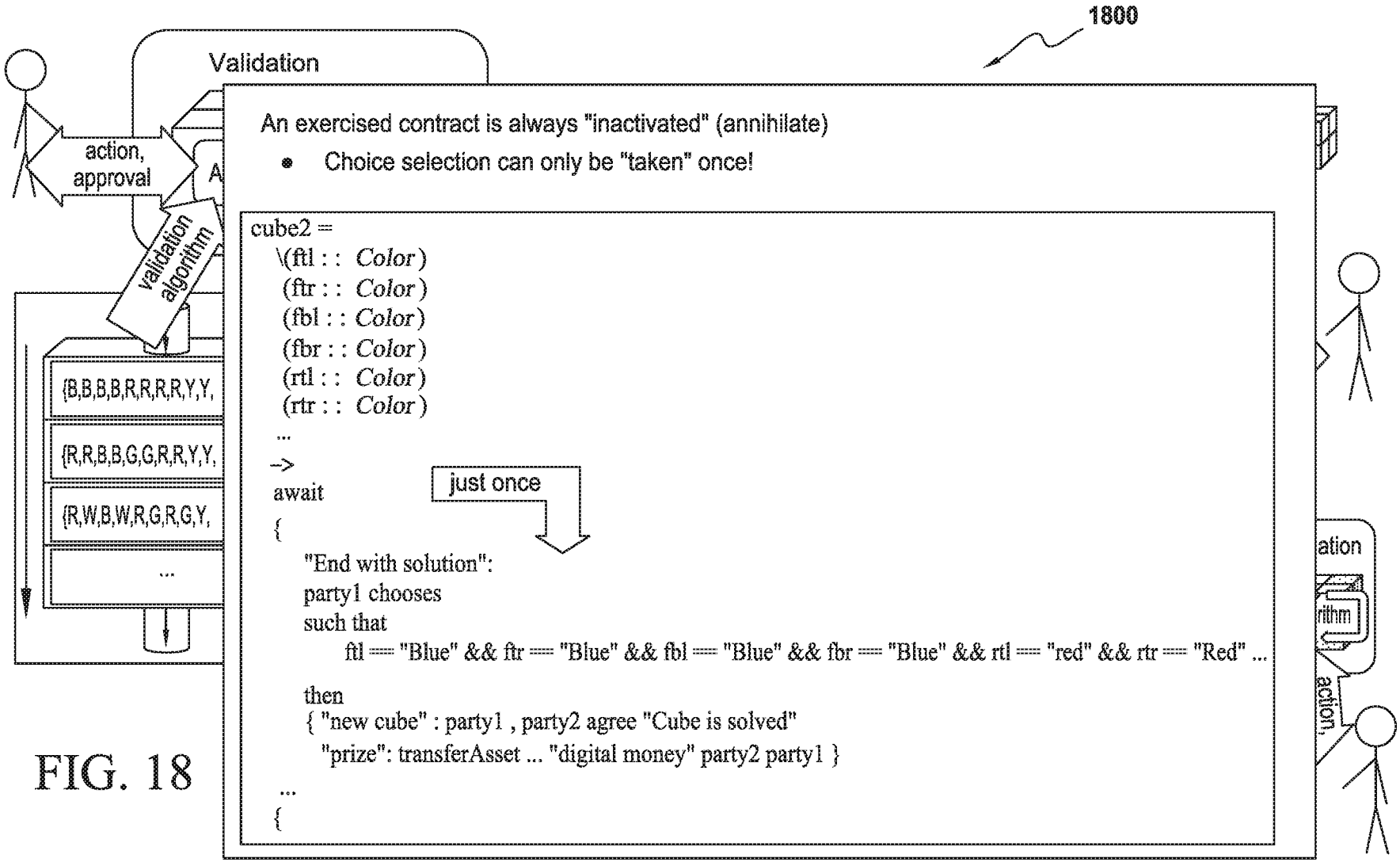

1800
Validation
action,
approval
validation
algorithm
{B,B,B,B,R,R,R,R,Y,Y,
{R,R,B,B,G,G,R,R,Y,Y,
{R,W,B,W,R,G,R,G,Y,
...
An exercised contract is always "inactivated" (annihilate)
Choice selection can only be "taken" once!
cube2 =
\(ftl : : Color)
(ftr : : Color)
(fbl : : Color)
(fbr : : Color)
(rtl : : Color)
(rtr : : Color)
...
->
await
just once
{
"End with solution":
party1 chooses
such that
ftl == "Blue" && ftr == "Blue" && fbl == "Blue" && fbr == "Blue" && rtl == "red" && rtr == "Red" ...
then
{ "new cube" : party1 , party2 agree "Cube is solved"
"prize": transferAsset ... "digital money" party2 party1 }
...
{
ation
ithm
action,

FIG. 18

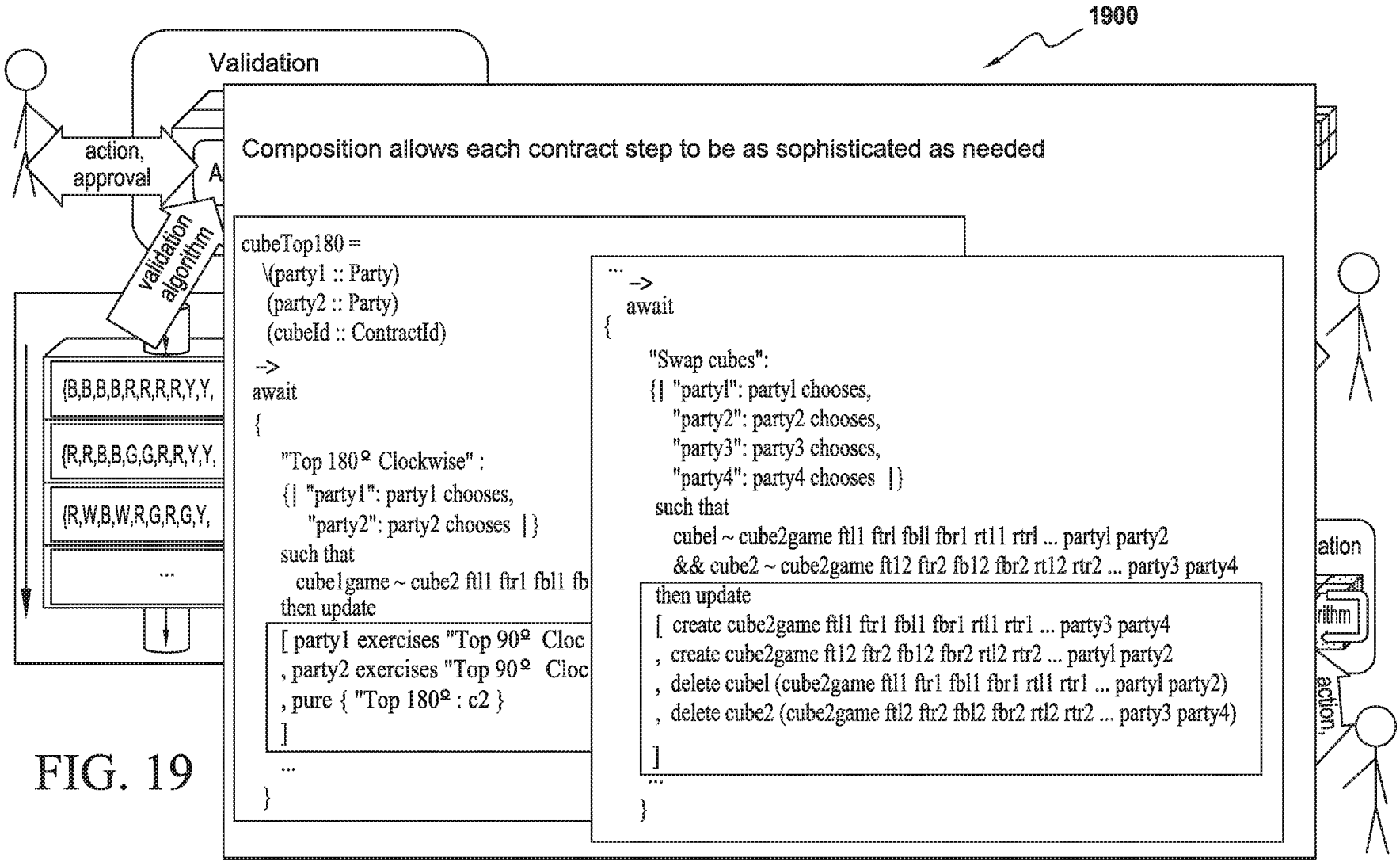

1900
Validation
action,
approval
validation
algorithm
Composition allows each contract step to be as sophisticated as needed
{B,B,B,B,R,R,R,R,Y,Y,
{R,R,B,B,G,G,R,R,Y,Y,
{R,W,B,W,R,G,R,G,Y,
...
cubeTop180 =
\(party1 :: Party)
(party2 :: Party)
(cubeId :: ContractId)
->
await
{
"Top 180º Clockwise" :
{| "party1": party1 chooses,
"party2": party2 chooses |}
such that
cube1game ~ cube2 ftl1 ftr1 fbl1 fb
then update
[party1 exercises "Top 90º Cloc
, party2 exercises "Top 90º Cloc
, pure { "Top 180º : c2 }
]
...
}
...
->
await
{
"Swap cubes":
{| "party1": party1 chooses,
"party2": party2 chooses,
"party3": party3 chooses,
"party4": party4 chooses |}
such that
cube1 ~ cube2game ftl1 ftr1 fbl1 fbr1 rtl1 rtr1 ... party1 party2
&& cube2 ~ cube2game ft12 ftr2 fb12 fbr2 rt12 rtr2 ... party3 party4
then update
[create cube2game ftl1 ftr1 fbl1 fbr1 rtl1 rtr1 ... party3 party4
, create cube2game ft12 ftr2 fb12 fbr2 rtl2 rtr2 ... party1 party2
, delete cube1 (cube2game ftl1 ftr1 fbl1 fbr1 rtl1 rtr1 ... party1 party2)
, delete cube2 (cube2game ftl2 ftr2 fbl2 fbr2 rtl2 rtr2 ... party3 party4)
]
...
}
ation
ithm
action,

FIG. 19

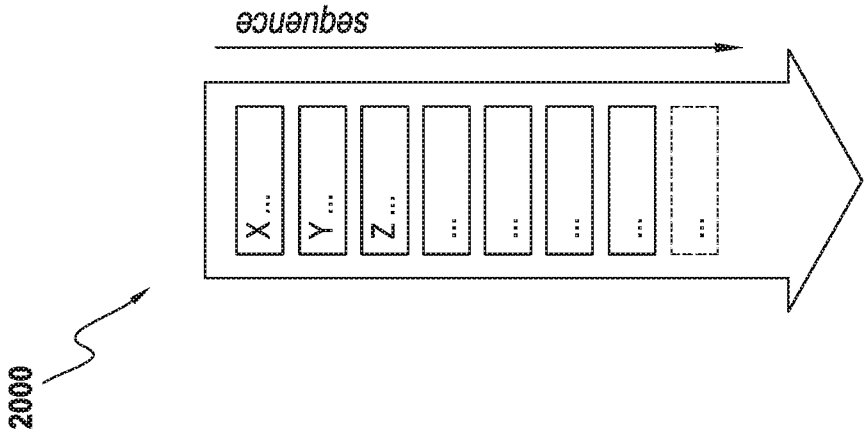


FIG. 20

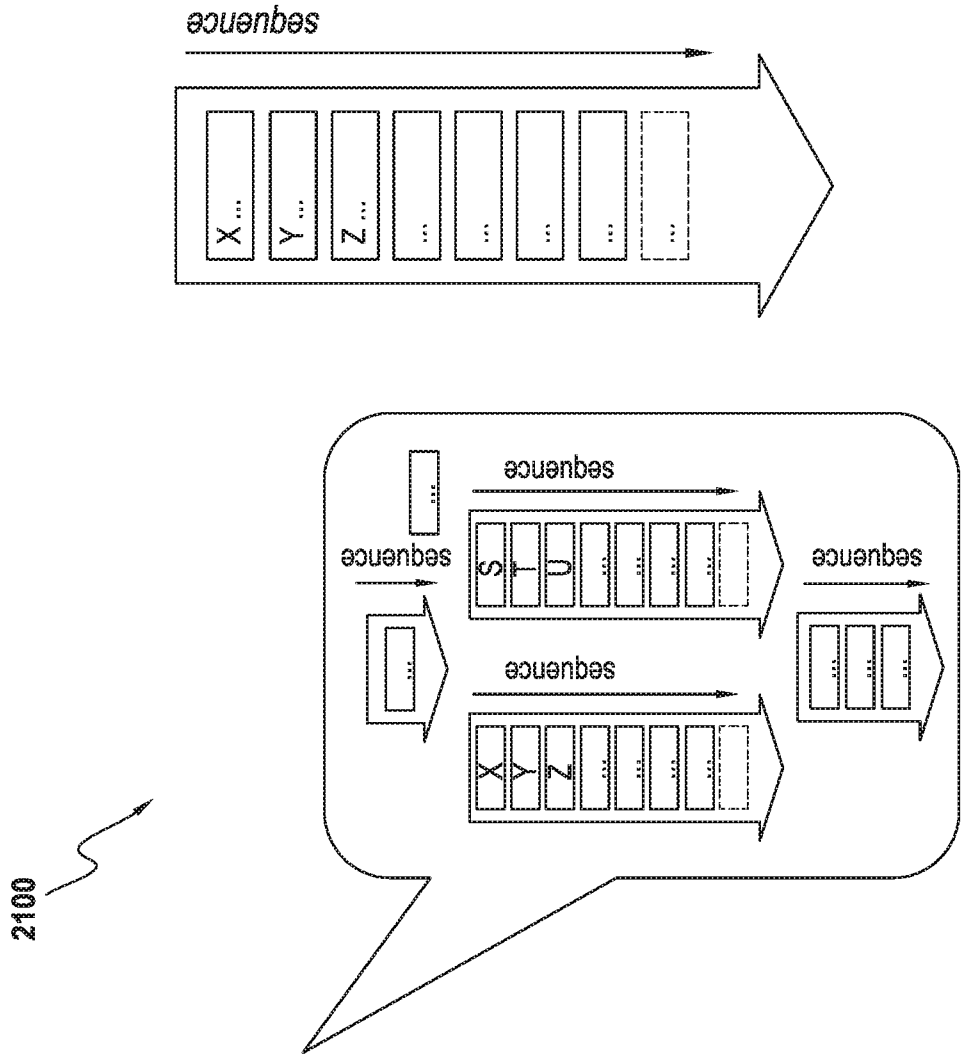

sequence
X...
Y...
Z...
2100
sequence
S
T
U
sequence
X
Y
Z
sequence
sequence

FIG. 21

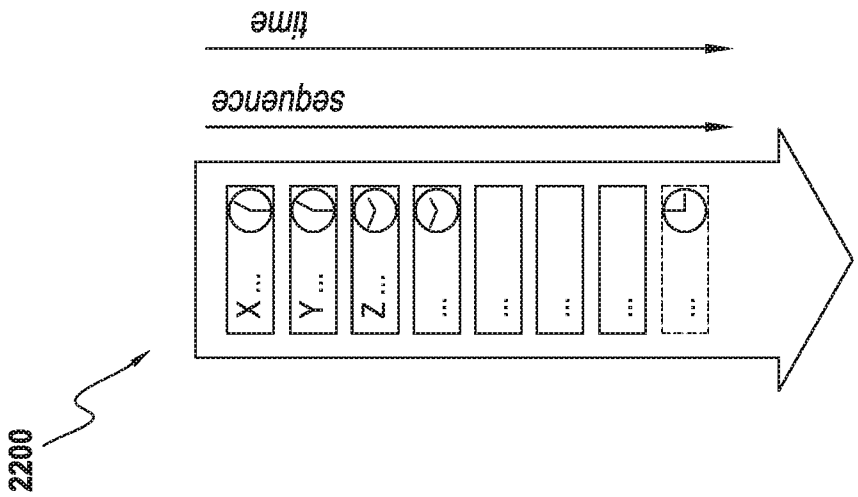


FIG. 22

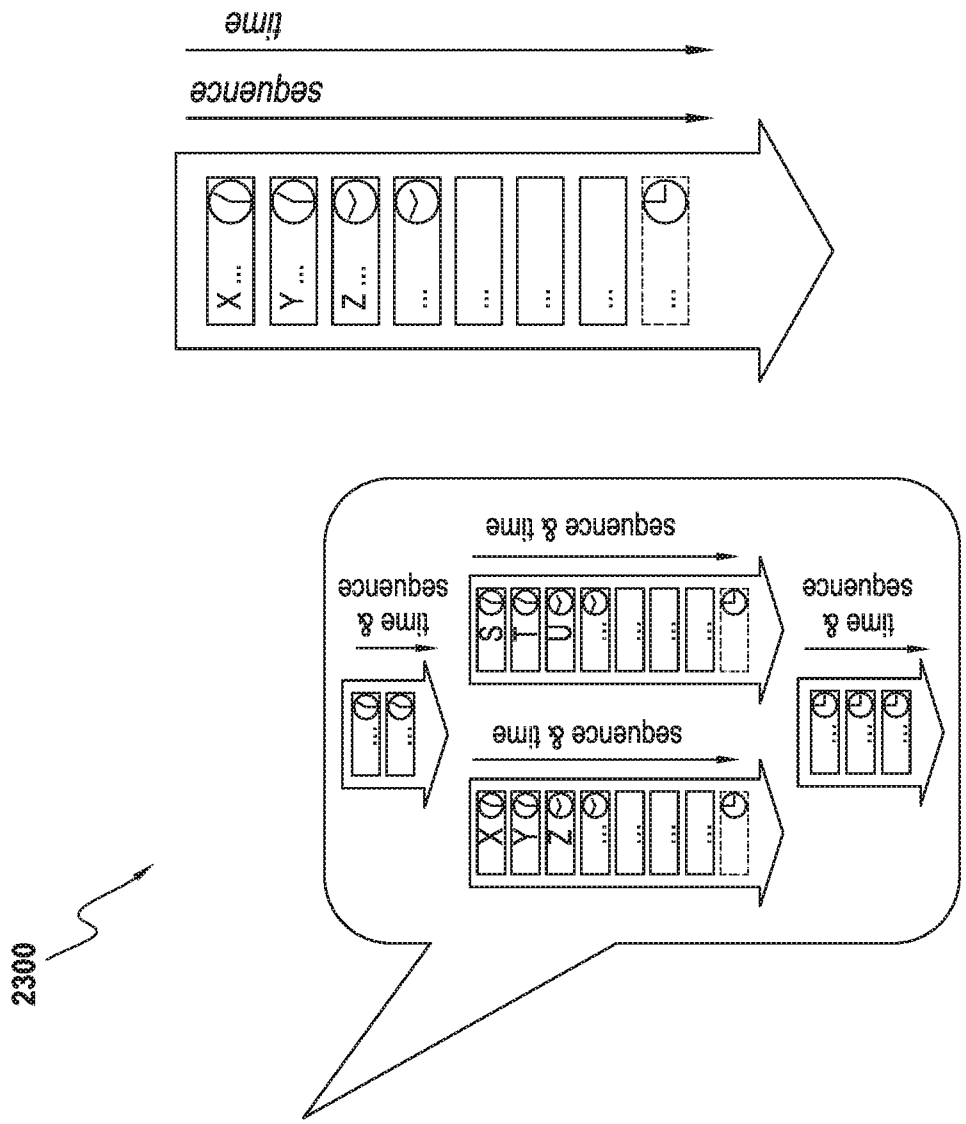

time
sequence
X...
Y...
Z...
2300
sequence & time
time & sequence
sequence & time
time & sequence

FIG. 23

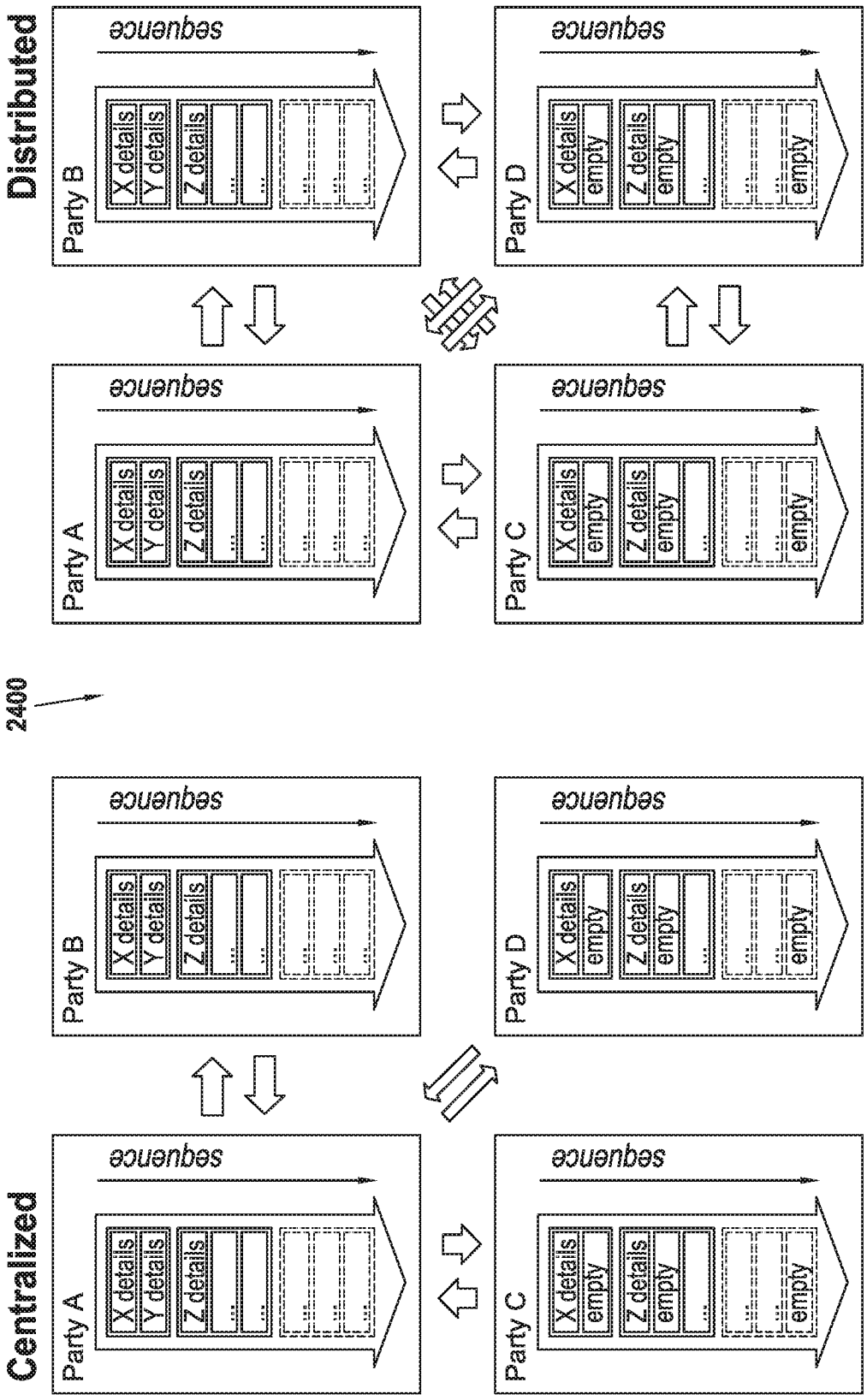

Distributed
Party B
Party D
Party A
Party C
Centralized
Party B
Party D
Party A
Party C
sequence
X details
Y details
Z details
empty
2400

FIG. 24

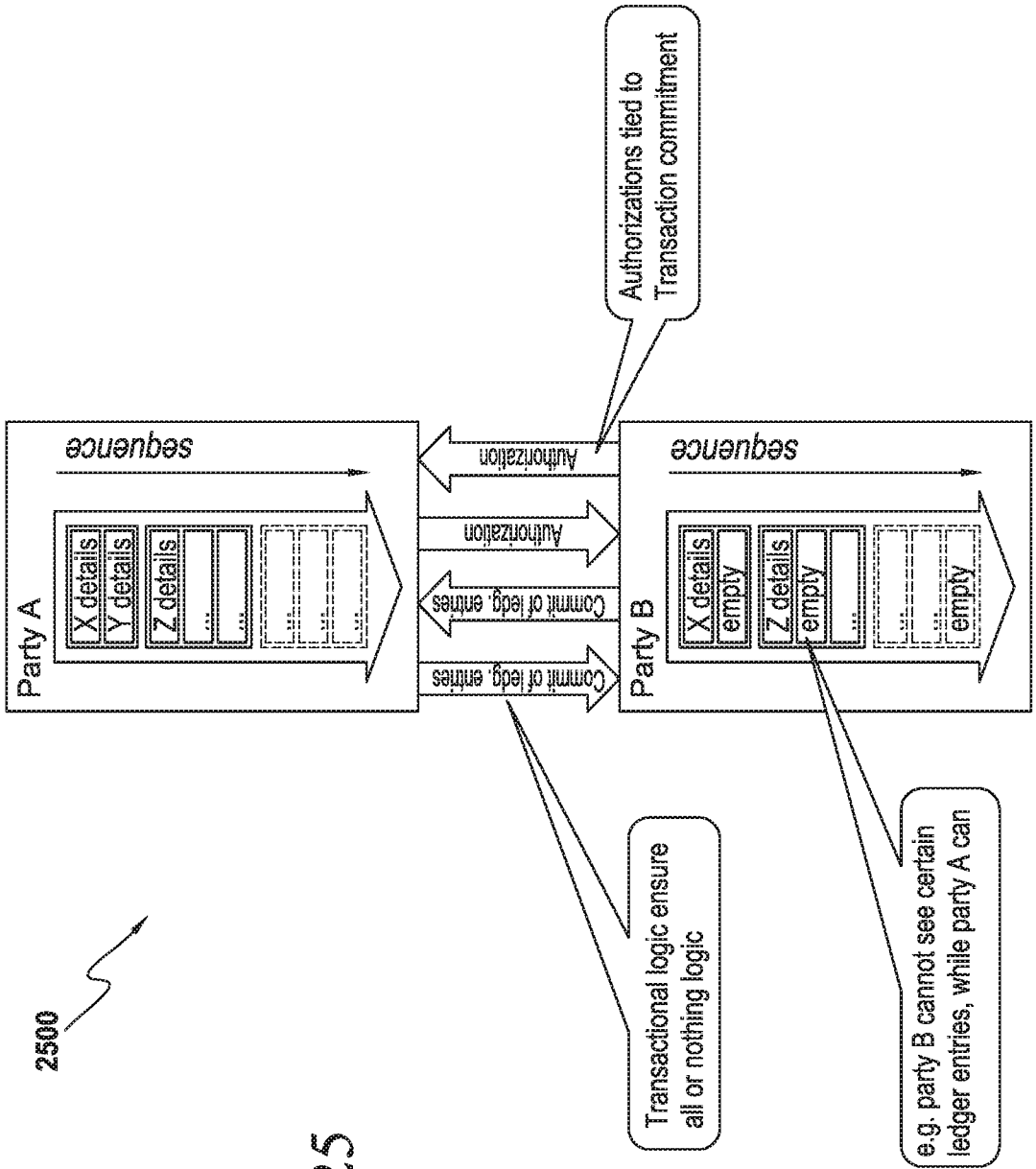

2500
Party A
sequence
X details
Y details
Z details
Party B
sequence
X details
empty
Z details
empty
empty
Authorization
Authorization
Commit of ledg. entries
Commit of ledg. entries
Authorizations tied to Transaction commitment
Transactional logic ensure all or nothing logic
e.g. party B cannot see certain ledger entries, while party A can

FIG. 25

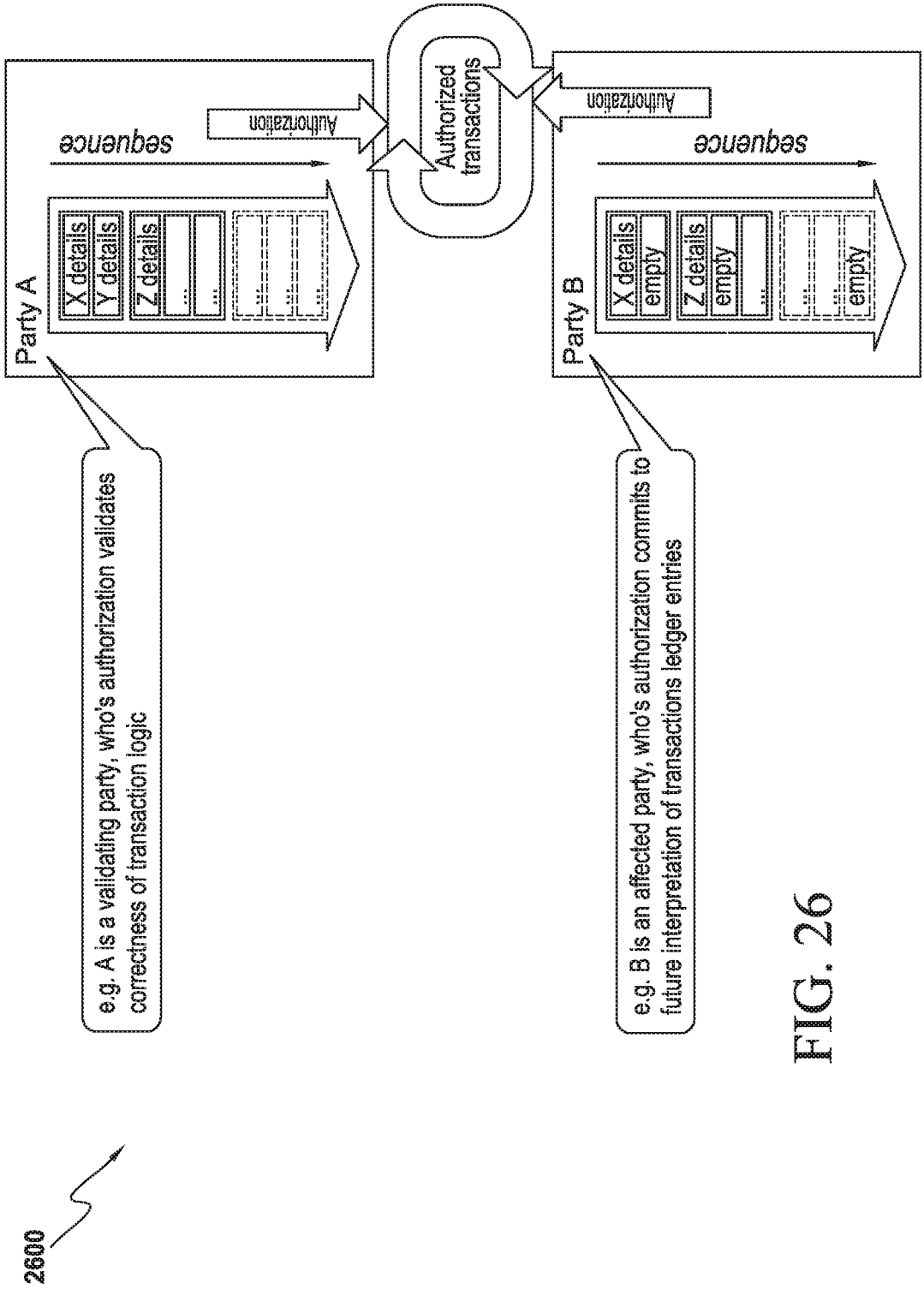

2600
Party A
sequence
Authorization
X details
Y details
Z details
Authorized transactions
Party B
sequence
Authorization
X details
empty
Z details
empty
empty
e.g. A is a validating party, who's authorization validates correctness of transaction logic
e.g. B is an affected party, who's authorization commits to future interpretation of transactions ledger entries

FIG. 26

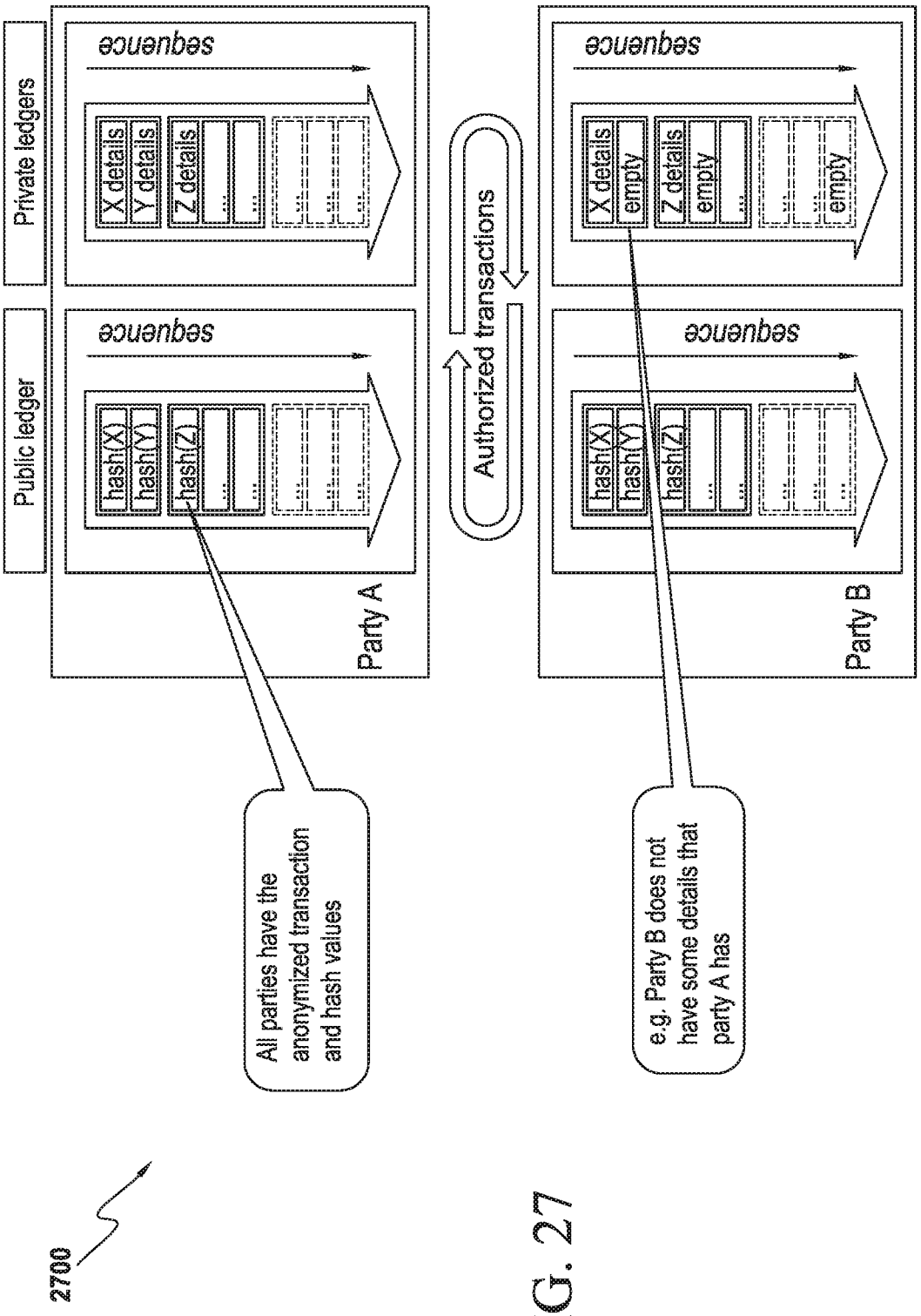

2700
Private ledgers
Public ledger
sequence
X details
Y details
Z details
hash(X)
hash(Y)
hash(Z)
Party A
Party B
Authorized transactions
empty
All parties have the anonymized transaction and hash values
e.g. Party B does not have some details that party A has

FIG. 27

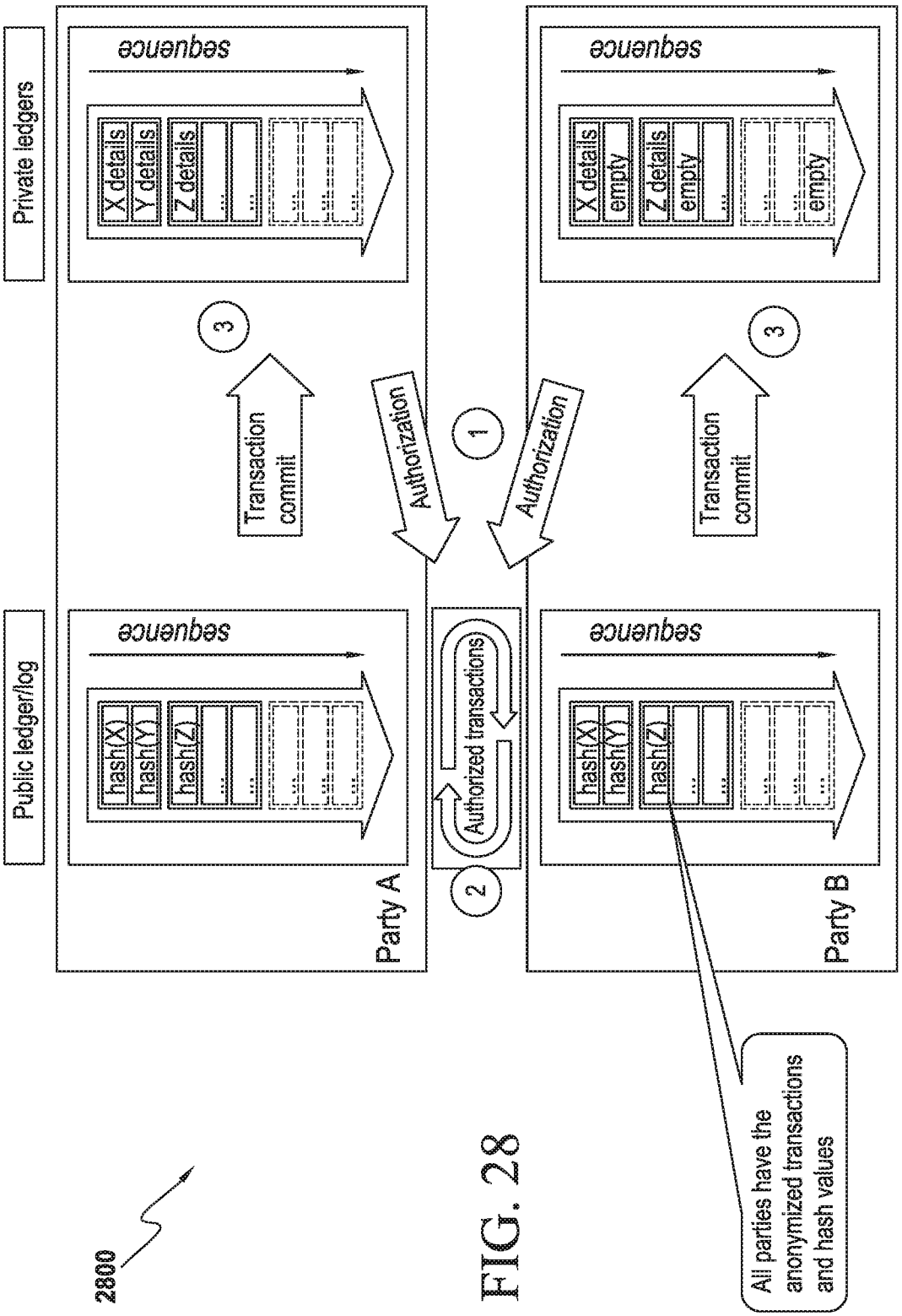

Private ledgers
Public ledger/log
sequence
X details
Y details
Z details
hash(X)
hash(Y)
hash(Z)
empty
Party A
Party B
Authorization
Authorization
Authorized transactions
Transaction commit
Transaction commit
1
2
3
3
All parties have the anonymized transactions and hash values
2800

FIG. 28

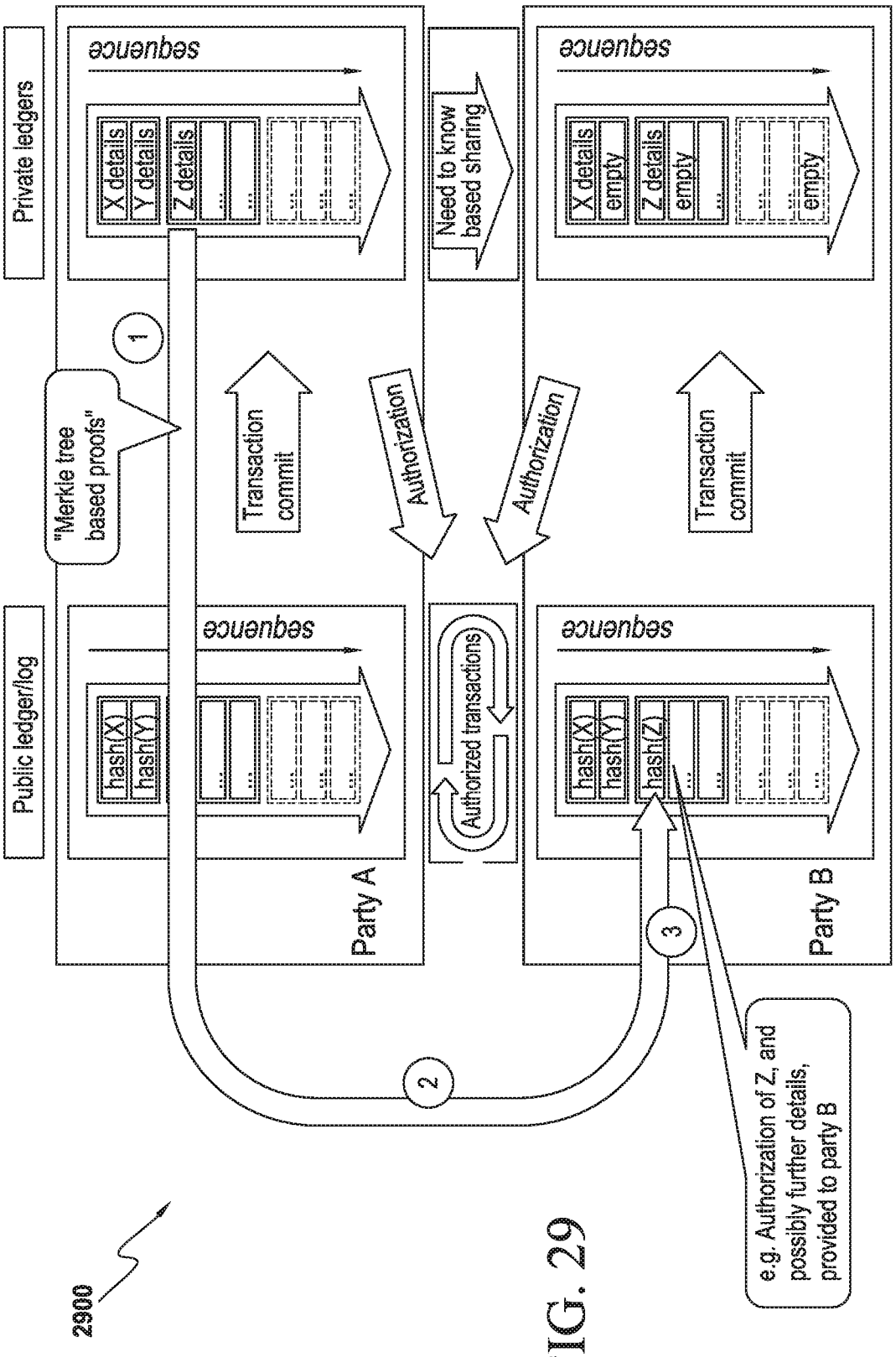

Private ledgers
sequence
X details
Y details
Z details
Need to know based sharing
X details
empty
Z details
empty
empty
"Merkle tree based proofs"
1
Transaction commit
Authorization
Authorization
Transaction commit
Public ledger/log
sequence
hash(X)
hash(Y)
Authorized transactions
hash(X)
hash(Y)
hash(Z)
Party A
Party B
2
3
e.g. Authorization of Z, and possibly further details, provided to party B
2900

FIG. 29

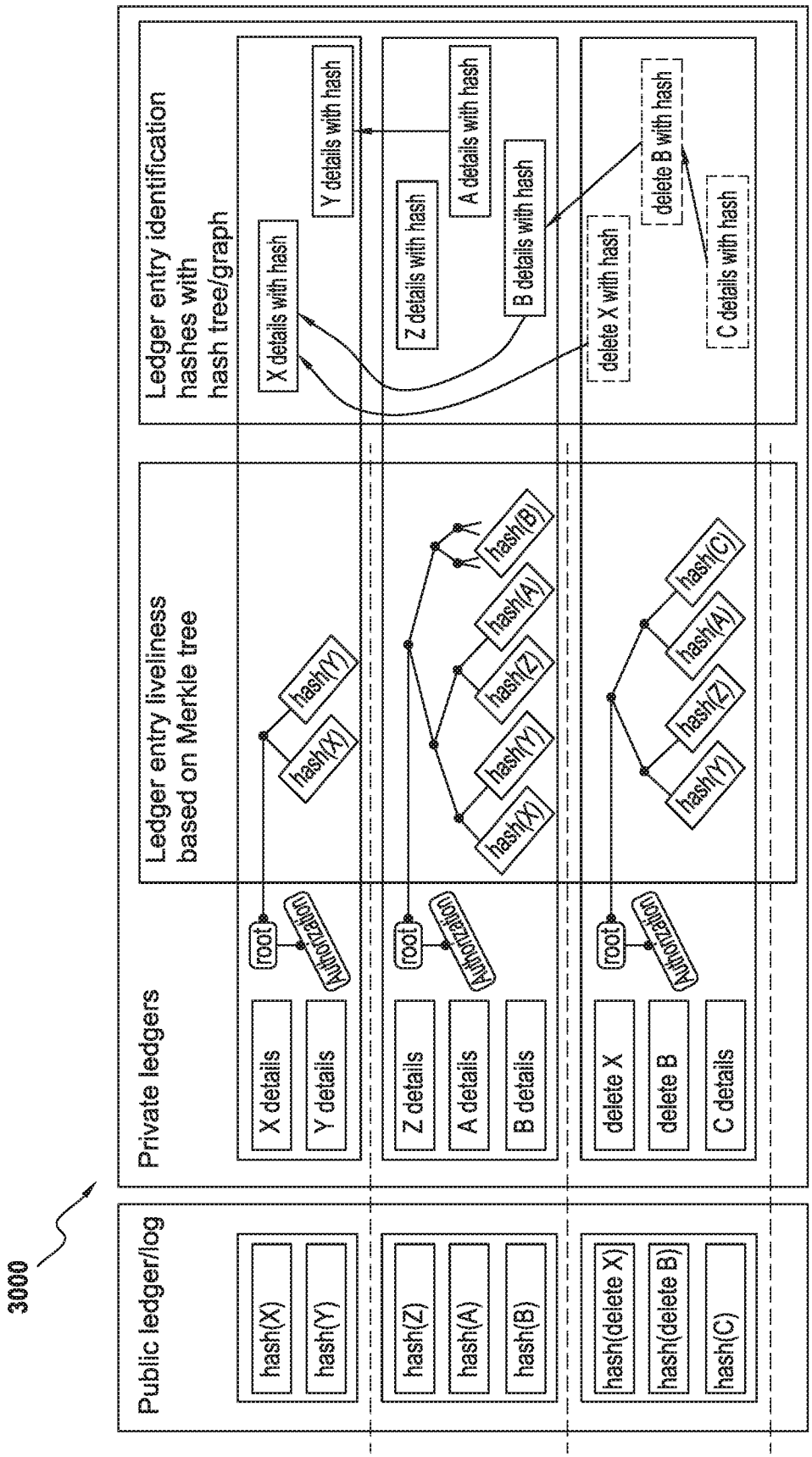

3000
Public ledger/log
Private ledgers
Ledger entry liveliness based on Merkle tree
Ledger entry identification hashes with hash tree/graph
hash(X)
hash(Y)
hash(Z)
hash(A)
hash(B)
hash(delete X)
hash(delete B)
hash(C)
X details
Y details
Z details
A details
B details
delete X
delete B
C details
root
Authorization
hash(X)
hash(Y)
hash(Z)
hash(A)
hash(B)
hash(C)
X details with hash
Y details with hash
Z details with hash
A details with hash
B details with hash
delete X with hash
delete B with hash
C details with hash

FIG. 30

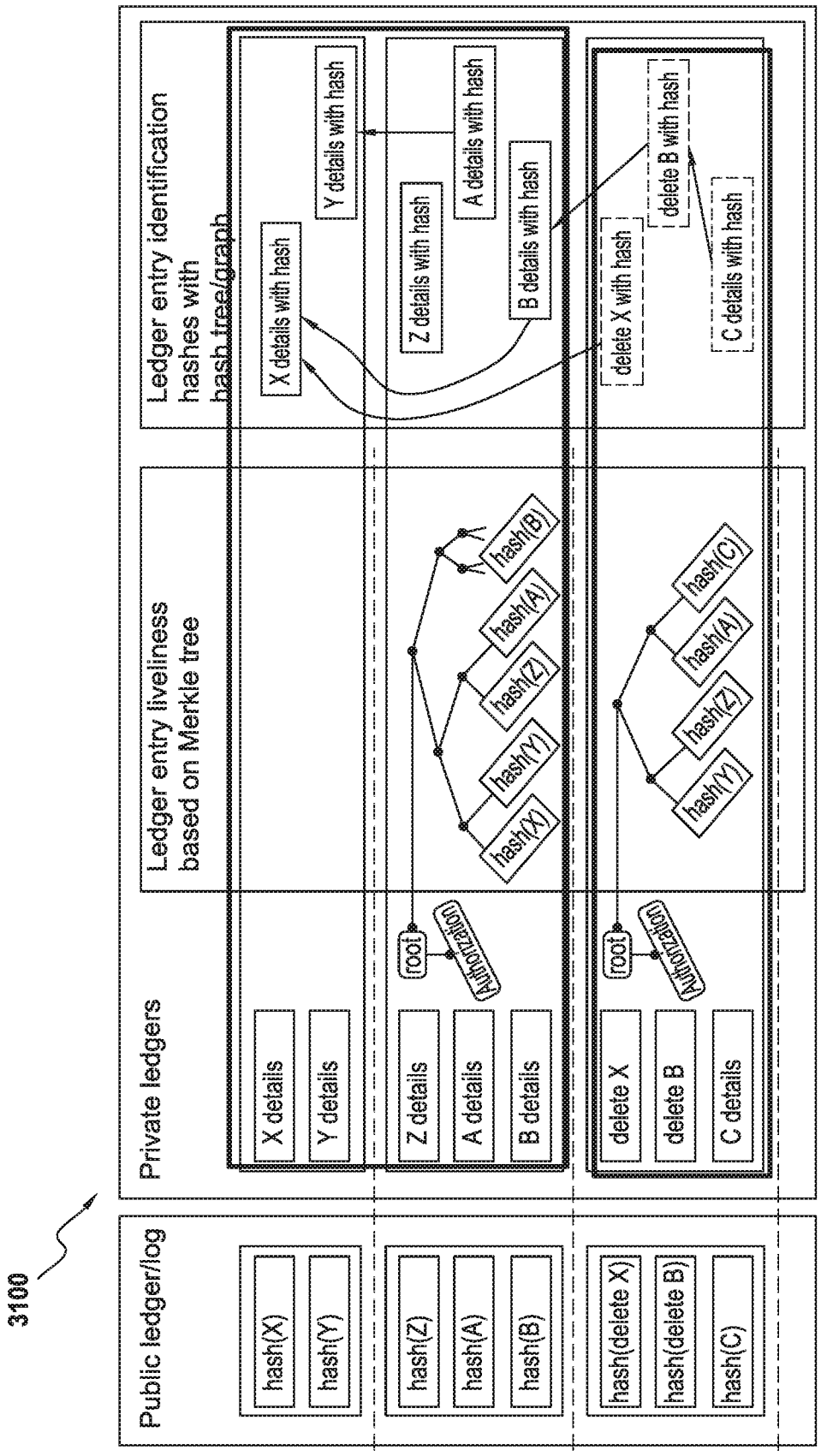

3100
Public ledger/log
Private ledgers
Ledger entry liveliness based on Merkle tree
Ledger entry identification hashes with hash tree/graph
hash(X)
hash(Y)
hash(Z)
hash(A)
hash(B)
hash(delete X)
hash(delete B)
hash(C)
X details
Y details
Z details
A details
B details
delete X
delete B
C details
root
Authorization
root
Authorization
hash(X)
hash(Y)
hash(Z)
hash(A)
hash(B)
hash(Y)
hash(Z)
hash(A)
hash(C)
X details with hash
Y details with hash
Z details with hash
A details with hash
B details with hash
delete X with hash
delete B with hash
C details with hash

FIG. 31

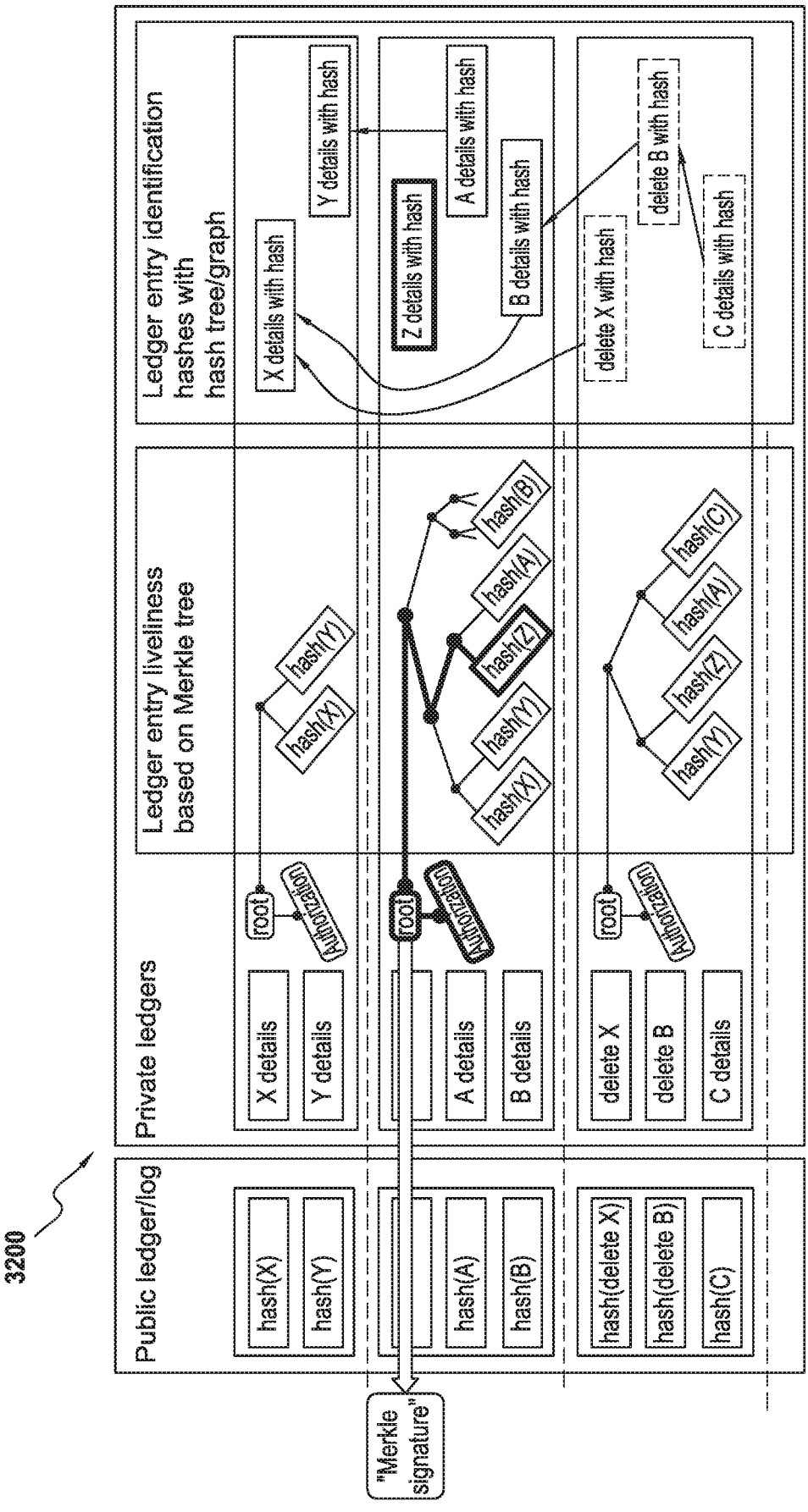

3200
Public ledger/log
Private ledgers
Ledger entry liveliness based on Merkle tree
Ledger entry identification hashes with hash tree/graph
hash(X)
hash(Y)
hash(A)
hash(B)
hash(delete X)
hash(delete B)
hash(C)
X details
Y details
A details
B details
delete X
delete B
C details
root
Authorization
hash(Y)
hash(X)
hash(B)
hash(A)
hash(Z)
hash(C)
X details with hash
Y details with hash
Z details with hash
A details with hash
B details with hash
delete X with hash
delete B with hash
C details with hash
"Merkle signature"

FIG. 32

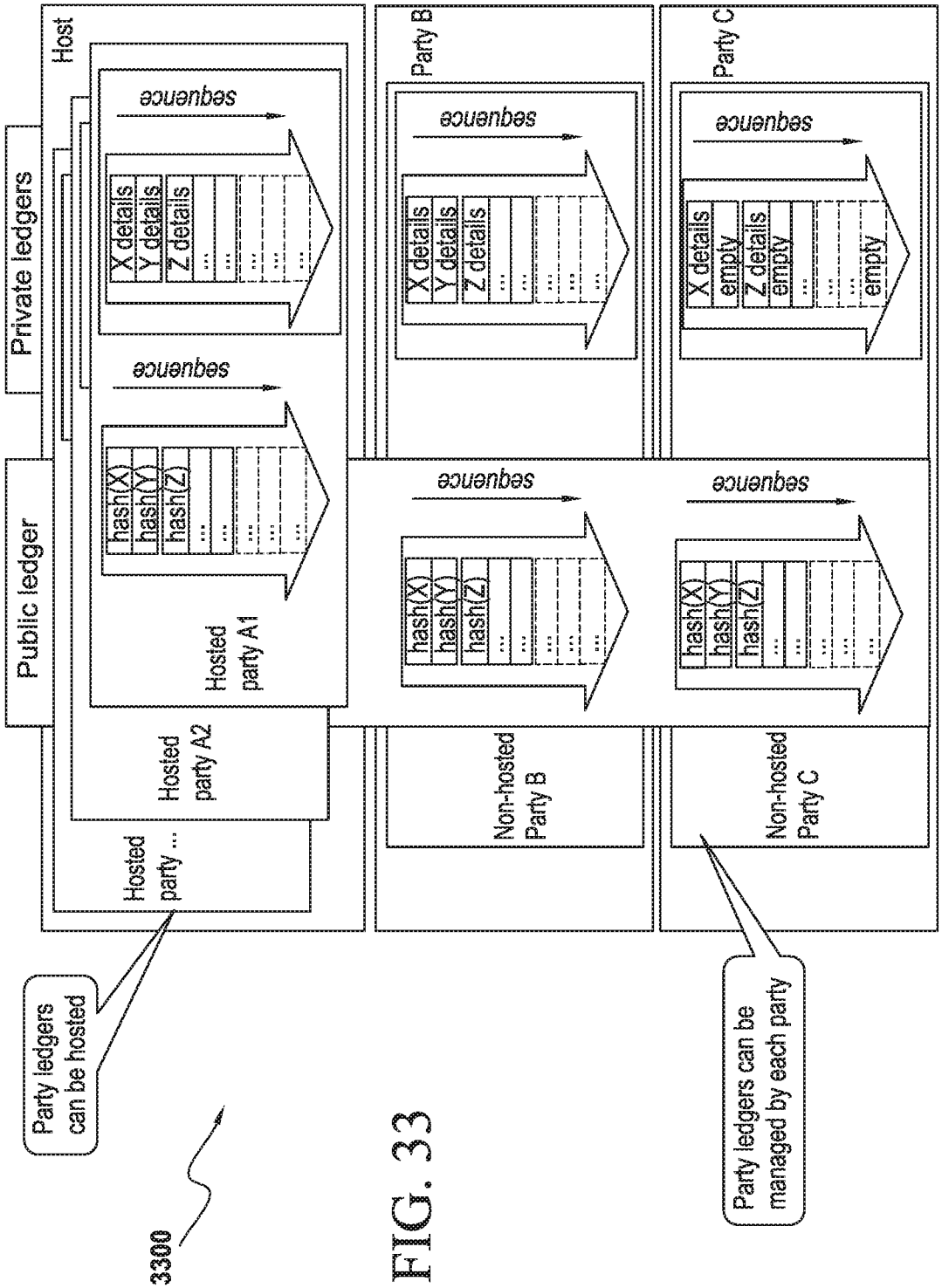

Host
Party B
Party C
Private ledgers
Public ledger
sequence
X details
Y details
Z details
hash(X)
hash(Y)
hash(Z)
empty
Hosted party A1
Hosted party A2
Hosted party ...
Non-hosted Party B
Non-hosted Party C
Party ledgers can be hosted
Party ledgers can be managed by each party
3300

FIG. 33

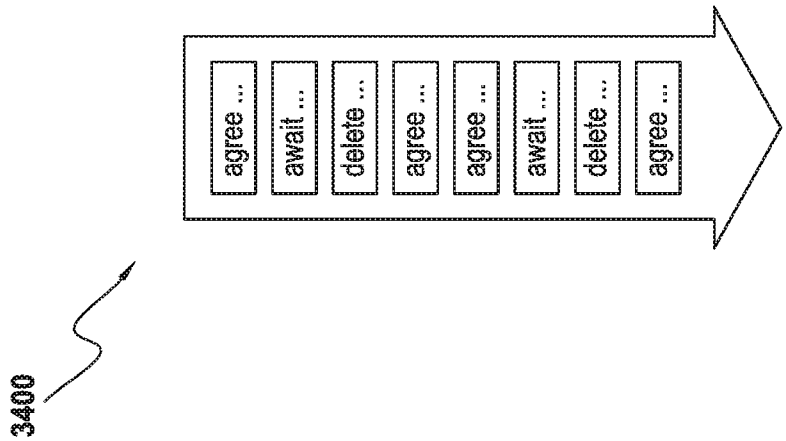
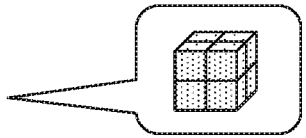


FIG. 34

3500

```
cube2 =
   \(ftl : :  Color)
    (ftr : :  Color)
    (fbl : :  Color)
    (fbr : :  Color)
    (rtl : :  Color)
    (rtr : :  Color)
    ...
   ->
    await
    {
       "End with solution":
       party1 chooses
       such that
            ftl == "Blue" && ftr == "Blue" && fbl == "Blue" && fbr == "Blue" && rtl == "red" && rtr == "Red" ...
       then
       { "new cube" : party1 , party2 agree "Cube is solved"
         "prize" : ...}
    ...
    {
```



Syntax and semantics is fully up to the parties to agree "off ledger".
- All parties to an agreement need to authorize it.

FIG. 35

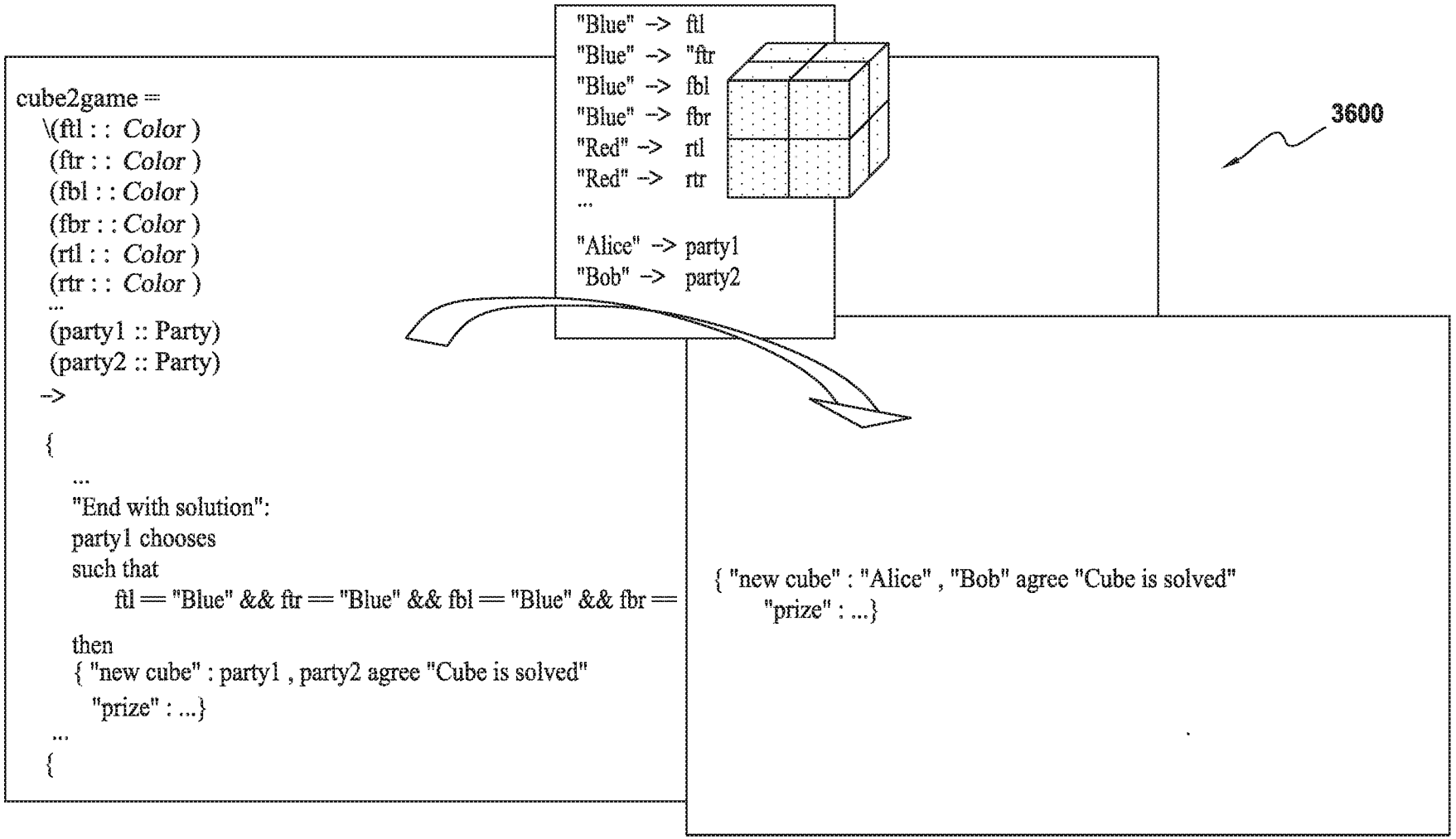

"Blue" -> ftl
"Blue" -> "ftr
"Blue" -> fbl
"Blue" -> fbr
"Red" -> rtl
"Red" -> rtr
...
"Alice" -> party1
"Bob" -> party2
3600
cube2game =
\(ftl : : Color)
(ftr : : Color)
(fbl : : Color)
(fbr : : Color)
(rtl : : Color)
(rtr : : Color)
...
(party1 :: Party)
(party2 :: Party)
->
{
...
"End with solution":
party1 chooses
such that
ftl == "Blue" && ftr == "Blue" && fbl == "Blue" && fbr ==
then
{ "new cube" : party1 , party2 agree "Cube is solved"
"prize" : ...}
...
{
{ "new cube" : "Alice" , "Bob" agree "Cube is solved"
"prize" : ...}

FIG. 36

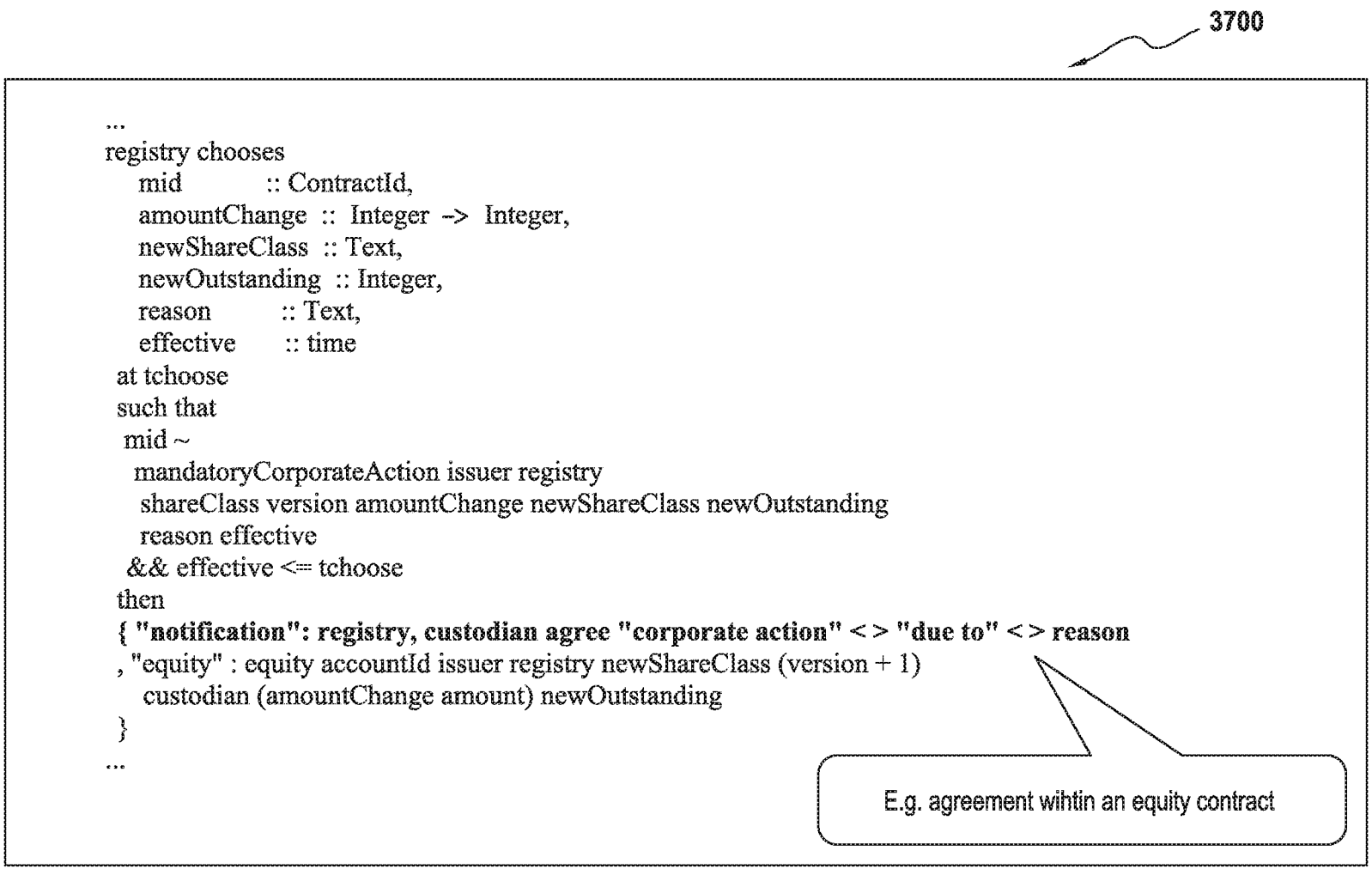

3700
...
registry chooses
mid :: ContractId,
amountChange :: Integer -> Integer,
newShareClass :: Text,
newOutstanding :: Integer,
reason :: Text,
effective :: time
at tchoose
such that
mid ~
mandatoryCorporateAction issuer registry
shareClass version amountChange newShareClass newOutstanding
reason effective
&& effective <= tchoose
then
{ "notification": registry, custodian agree "corporate action" <> "due to" <> reason
, "equity" : equity accountId issuer registry newShareClass (version + 1)
custodian (amountChange amount) newOutstanding
}
...
E.g. agreement wihtin an equity contract

FIG. 37

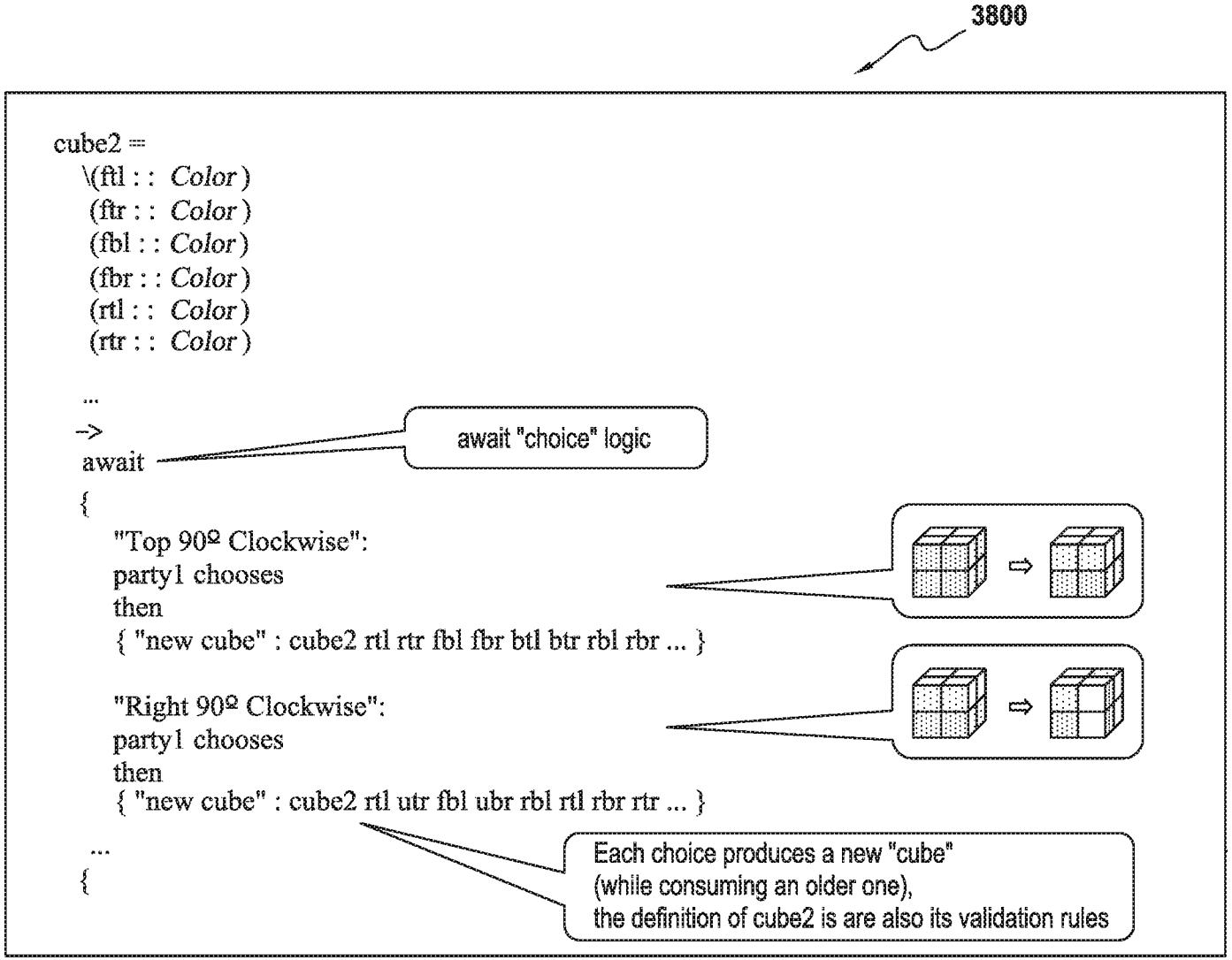

3800
cube2 =
\(ftl : : Color)
(ftr : : Color)
(fbl : : Color)
(fbr : : Color)
(rtl : : Color)
(rtr : : Color)
...
->
await
await "choice" logic
{
"Top 90º Clockwise":
party1 chooses
then
{ "new cube" : cube2 rtl rtr fbl fbr btl btr rbl rbr ... }
"Right 90º Clockwise":
party1 chooses
then
{ "new cube" : cube2 rtl utr fbl ubr rbl rtl rbr rtr ... }
...
{
Each choice produces a new "cube"
(while consuming an older one),
the definition of cube2 is are also its validation rules

FIG. 38

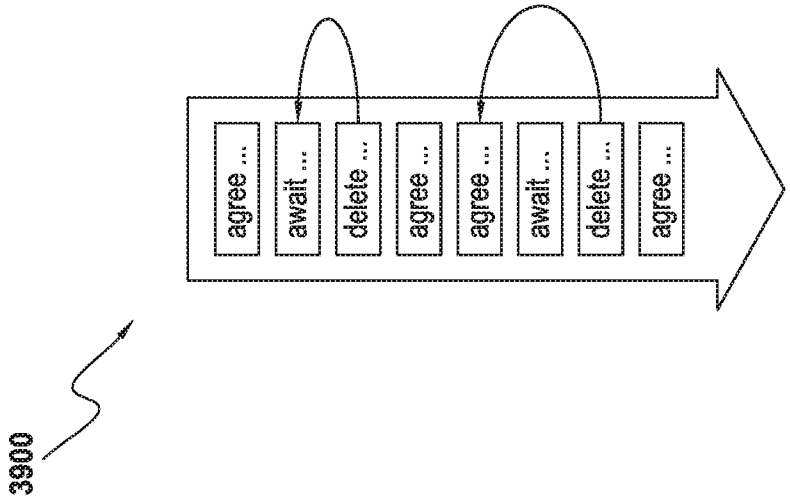


FIG. 39

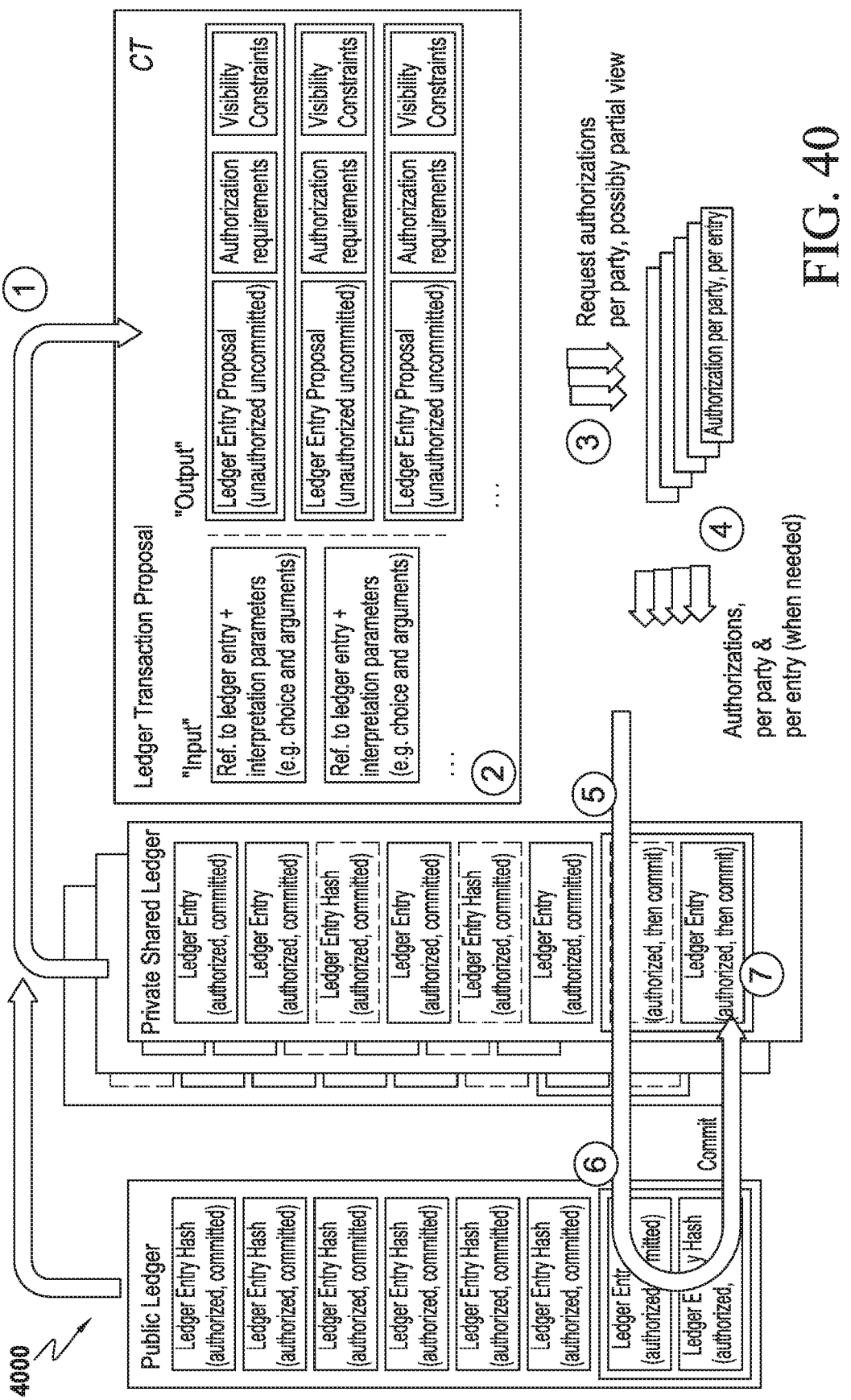

4000
Public Ledger
Ledger Entry Hash (authorized, committed)
Private Shared Ledger
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
Ledger Entry (authorized, committed)
Ledger Transaction Proposal
CT
"Input"
"Output"
Ref. to ledger entry + interpretation parameters (e.g. choice and arguments)
Ref. to ledger entry + interpretation parameters (e.g. choice and arguments)
Ledger Entry Proposal (unauthorized uncommitted)
Authorization requirements
Visibility Constraints
Ledger Entry Proposal (unauthorized uncommitted)
Authorization requirements
Visibility Constraints
Ledger Entry Proposal (unauthorized uncommitted)
Authorization requirements
Visibility Constraints
1
2
3
Request authorizations per party, possibly partial view
Authorization per party, per entry
4
Authorizations, per party & per entry (when needed)
5
6
Commit
7

FIG. 40

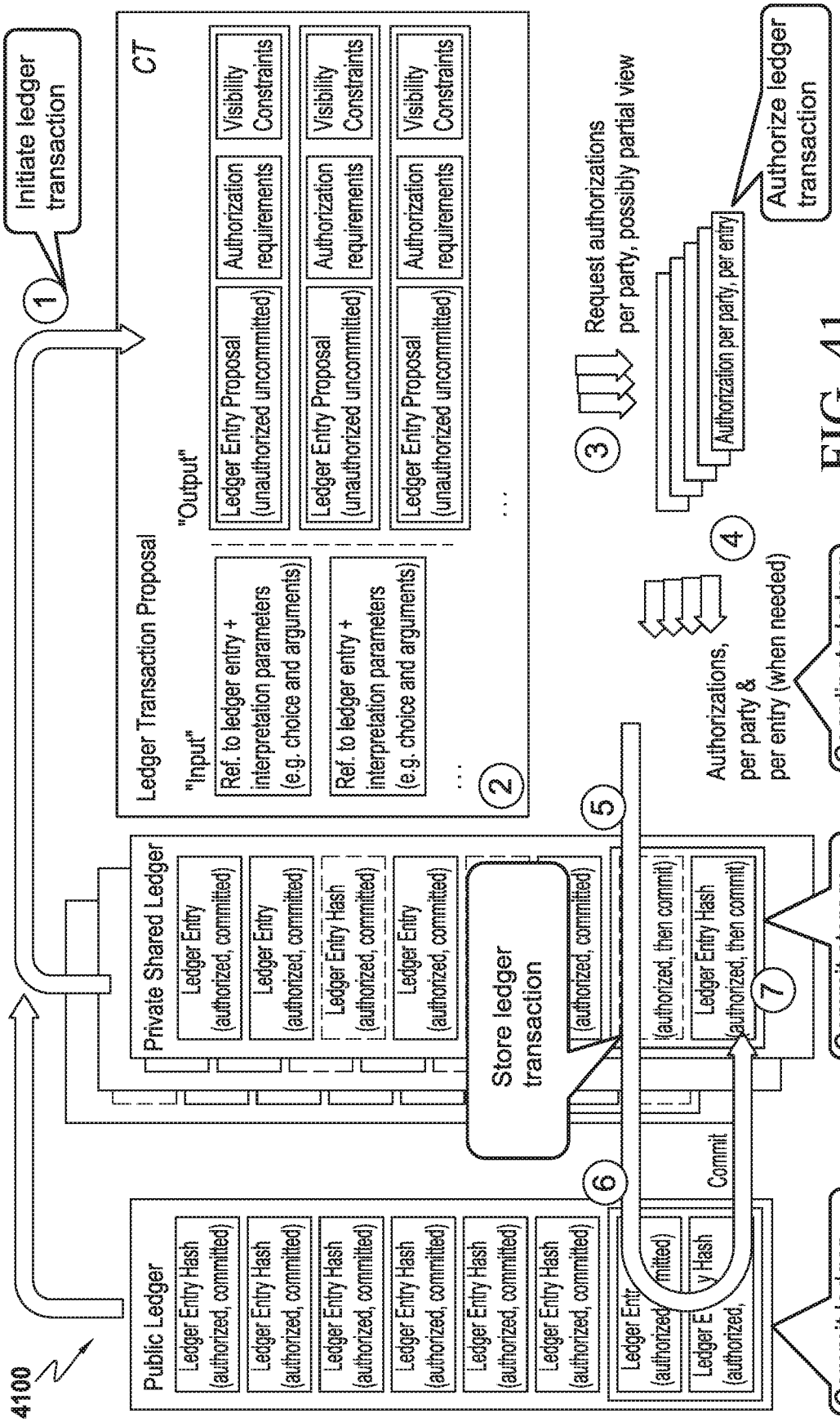

4100
Initiate ledger transaction
1
Ledger Transaction Proposal
CT
"Input"
"Output"
Ref. to ledger entry + interpretation parameters (e.g. choice and arguments)
Ledger Entry Proposal (unauthorized uncommitted)
Authorization requirements
Visibility Constraints
2
3
Request authorizations per party, possibly partial view
Authorization per party, per entry
Authorizations, per party & per entry (when needed)
4
Coordinate ledger transaction
Authorize ledger transaction
Private Shared Ledger
Ledger Entry (authorized, committed)
Ledger Entry Hash (authorized, committed)
5
Store ledger transaction
Public Ledger
6
Commit
7
Commit storage state
Commit ledger transaction

FIG. 41

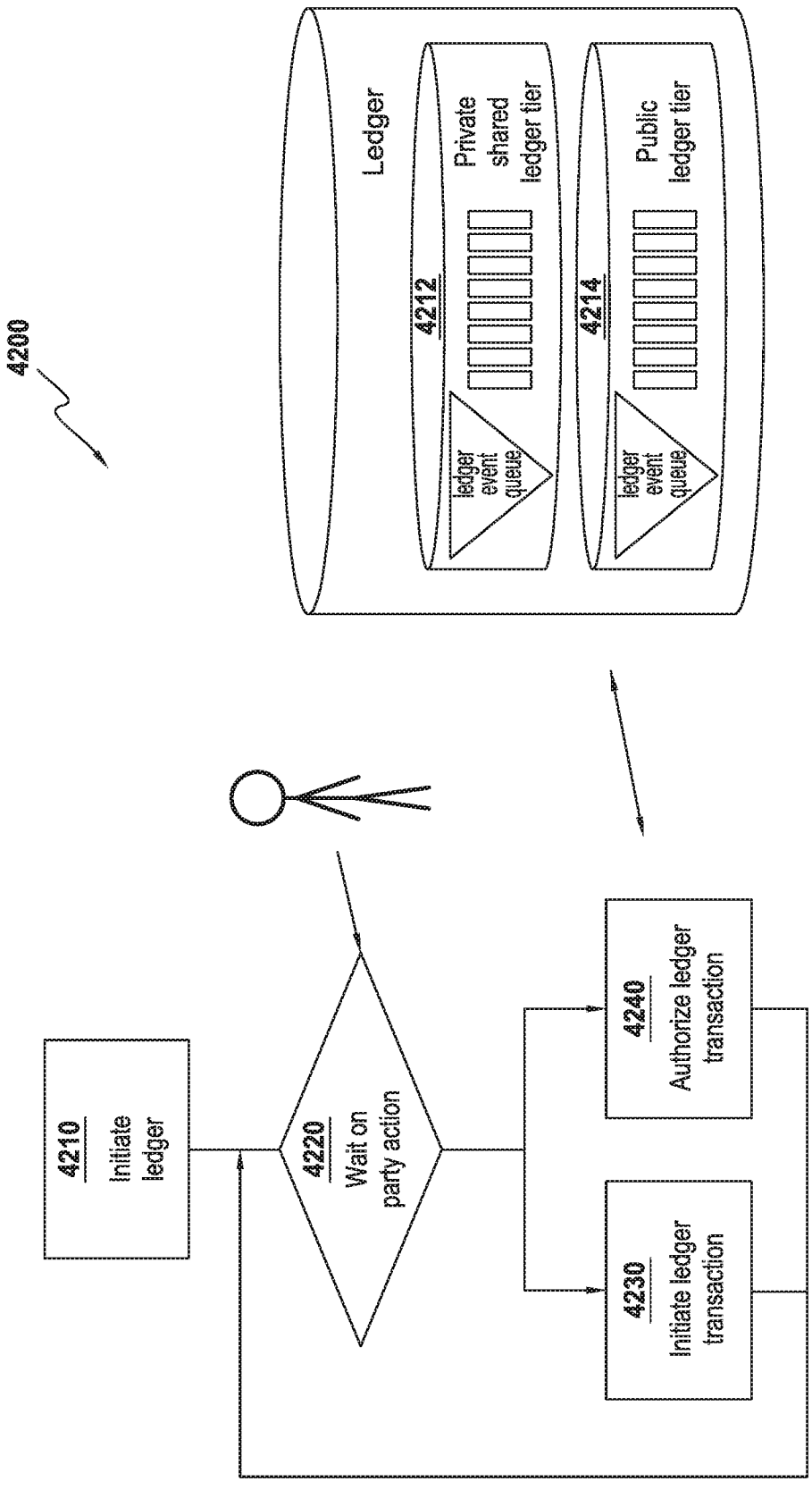

4200
Ledger
4212
Private shared ledger tier
ledger event queue
4214
Public ledger tier
ledger event queue
4210
Initiate ledger
4220
Wait on party action
4240
Authorize ledger transaction
4230
Initiate ledger transaction

FIG. 42

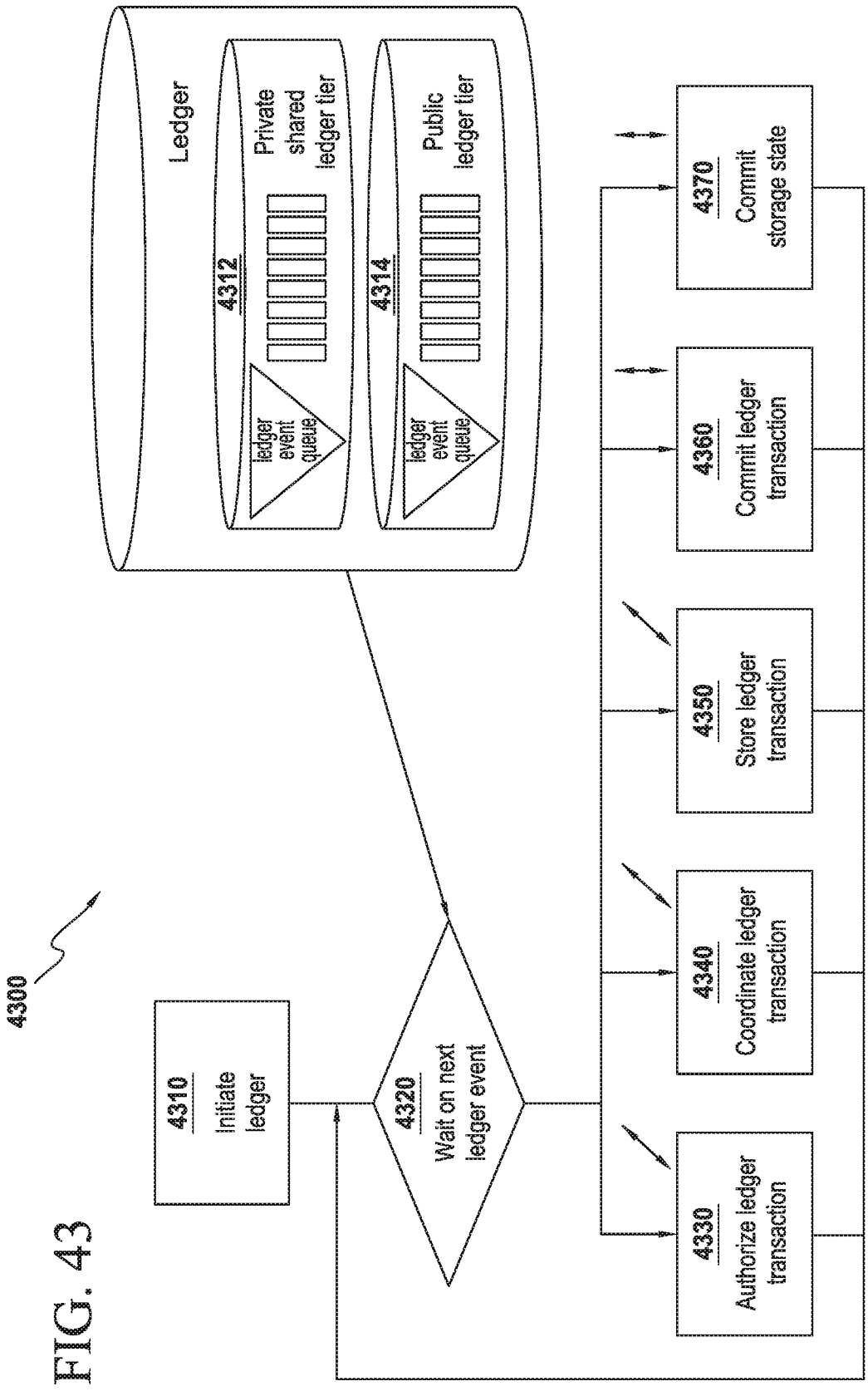

4300
Ledger
4312
Private shared ledger tier
ledger event queue
4314
Public ledger tier
ledger event queue
4310
Initiate ledger
4320
Wait on next ledger event
4330
Authorize ledger transaction
4340
Coordinate ledger transaction
4350
Store ledger transaction
4360
Commit ledger transaction
4370
Commit storage state

FIG. 43

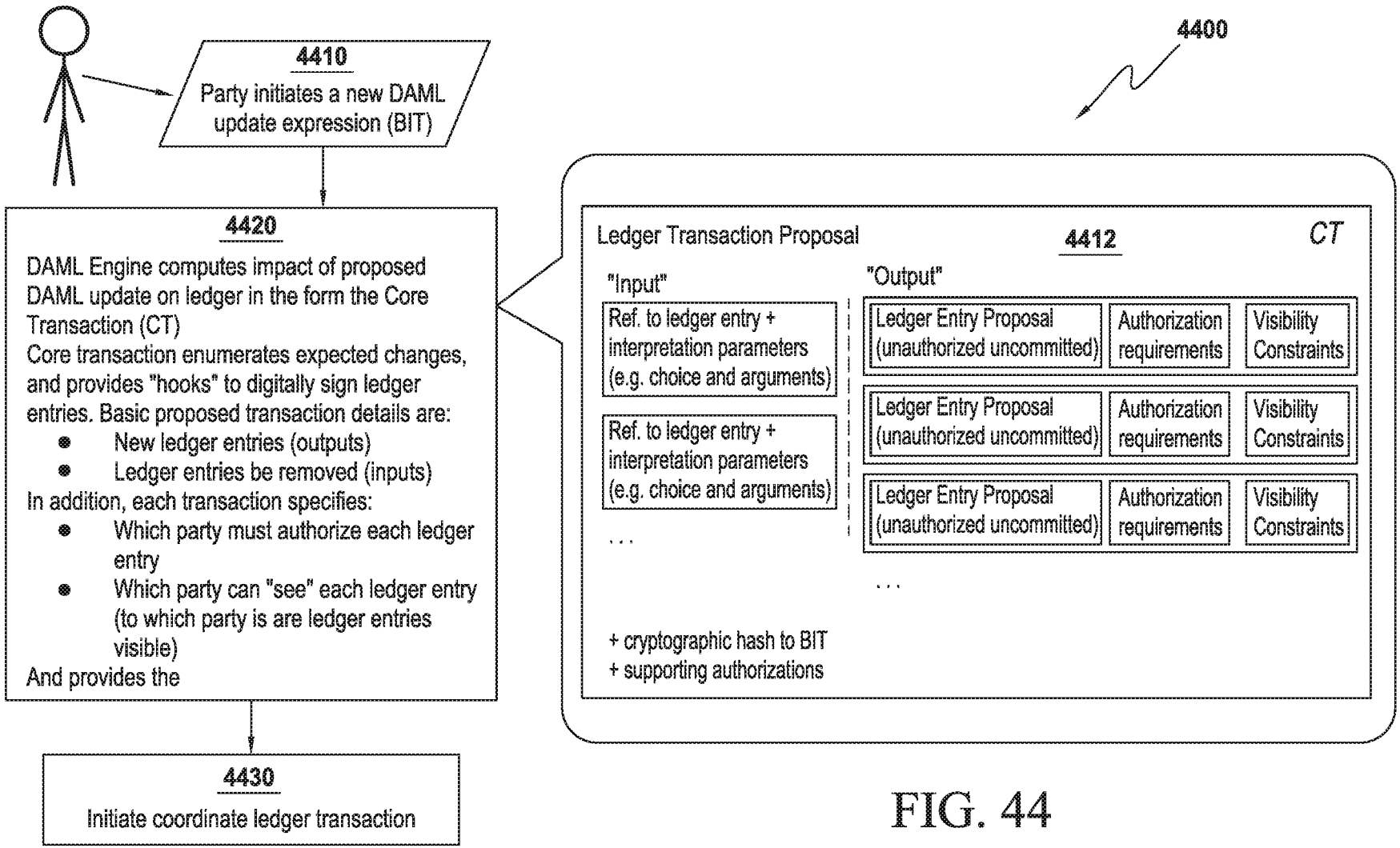

4400
4410
Party initiates a new DAML update expression (BIT)
4420
DAML Engine computes impact of proposed DAML update on ledger in the form the Core Transaction (CT)
Core transaction enumerates expected changes, and provides "hooks" to digitally sign ledger entries. Basic proposed transaction details are:
New ledger entries (outputs)
Ledger entries be removed (inputs)
In addition, each transaction specifies:
Which party must authorize each ledger entry
Which party can "see" each ledger entry (to which party is are ledger entries visible)
And provides the
4430
Initiate coordinate ledger transaction
Ledger Transaction Proposal
4412
CT
"Input"
"Output"
Ref. to ledger entry + interpretation parameters (e.g. choice and arguments)
Ref. to ledger entry + interpretation parameters (e.g. choice and arguments)
. . .
Ledger Entry Proposal (unauthorized uncommitted)
Authorization requirements
Visibility Constraints
Ledger Entry Proposal (unauthorized uncommitted)
Authorization requirements
Visibility Constraints
Ledger Entry Proposal (unauthorized uncommitted)
Authorization requirements
Visibility Constraints
. . .
+ cryptographic hash to BIT
+ supporting authorizations

FIG. 44

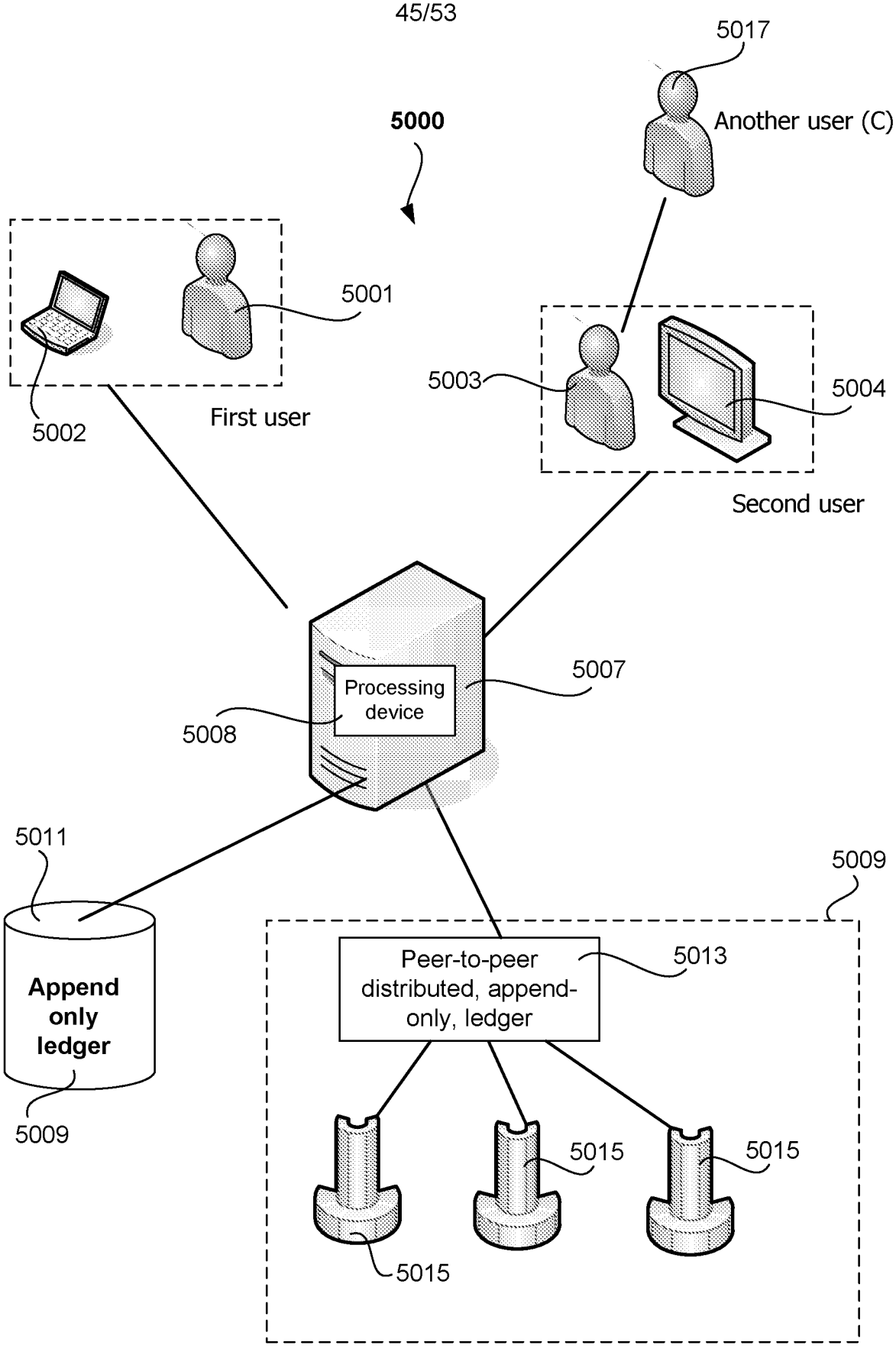

5017
Another user (C)
5000
5001
First user
5002
5003
5004
Second user
Processing device
5007
5008
5011
Append only ledger
5009
5009
Peer-to-peer distributed, append-only, ledger
5013
5015
5015
5015

FIG. 45

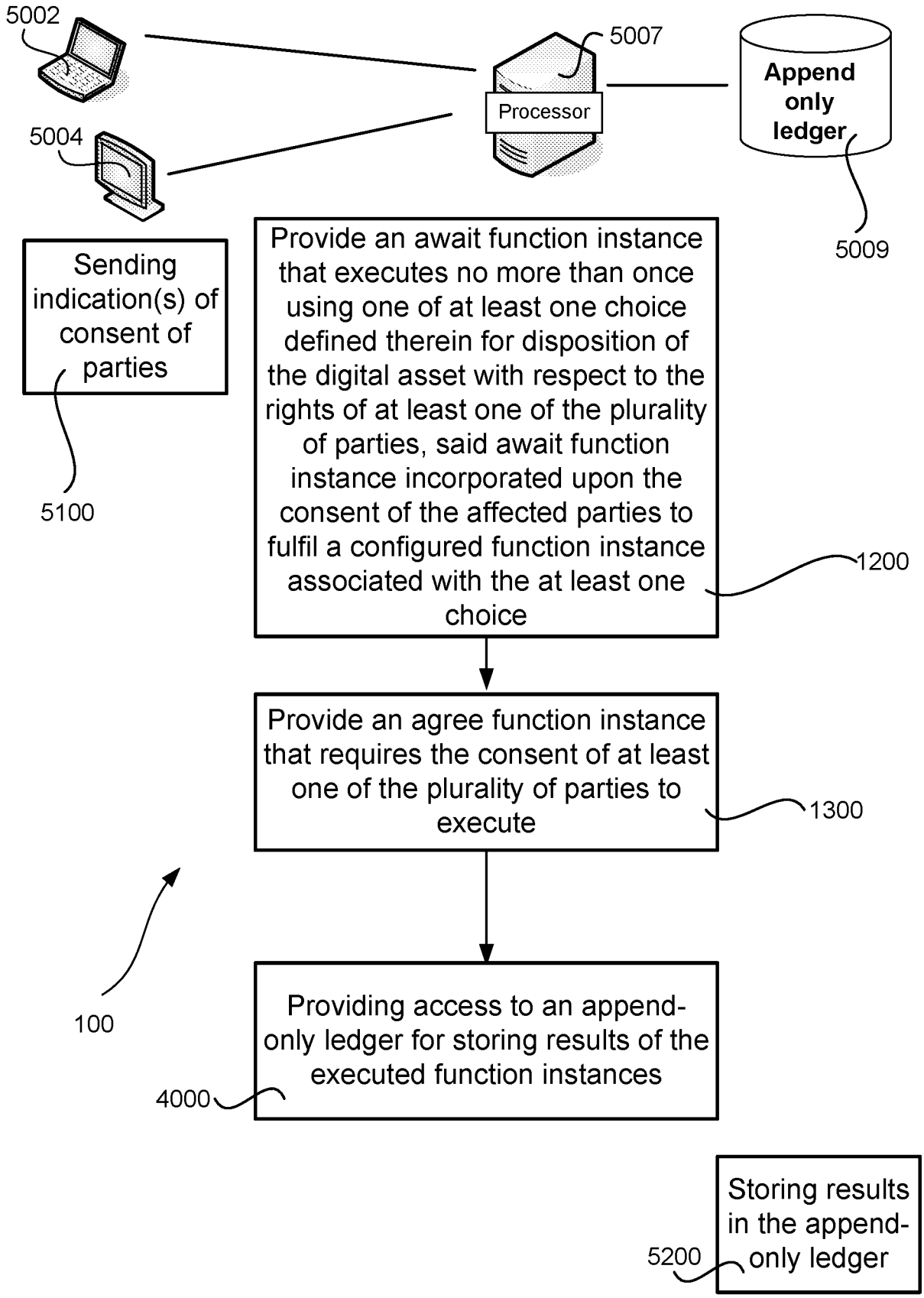
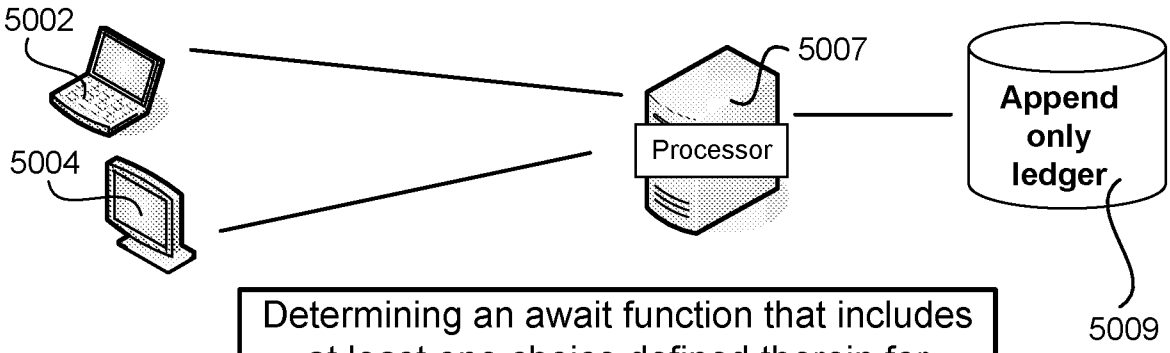

5002
5007
Append only ledger
Processor
5004
5009
Sending indication(s) of consent of parties
Provide an await function instance that executes no more than once using one of at least one choice defined therein for disposition of the digital asset with respect to the rights of at least one of the plurality of parties, said await function instance incorporated upon the consent of the affected parties to fulfil a configured function instance associated with the at least one choice
5100
1200
Provide an agree function instance that requires the consent of at least one of the plurality of parties to execute
1300
100
Providing access to an append-only ledger for storing results of the executed function instances
4000
Storing results in the append-only ledger
5200

FIG. 46



Determining an await function that includes at least one choice defined therein for disposition of the digital asset, wherein the at least one choice has an associated configured function and a respective choice-step condition

6100

6250

Sending indication(s) of consent of a party or parties required by the at least one choice

Executing the await function instance no more than once, wherein upon receiving an indication of consent of parties required by the at least one choice and receiving a valid choice selection from the at least one choice, the method further comprises executing the associated configured function instance of the at least one choice, wherein the await function terminates with determination that the choice-step condition of the valid choice selection is satisfied

6200

6000

6350

Sending indication(s) of consent of a party or parties required for the agreement

Determining an agree function to create an agree record associated with the digital asset, wherein the method further comprises executing an agree function instance based on receiving consent of parties required for the agreement

6300

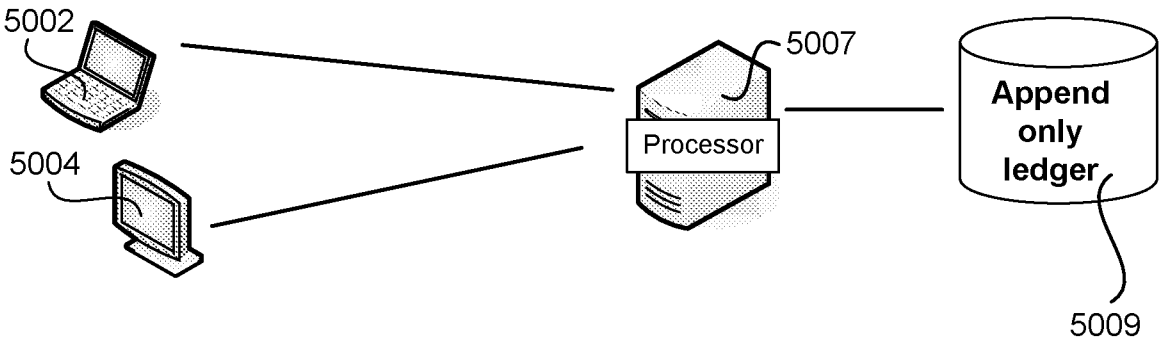
6450

Storing results in append-only ledger

Sending, to an append-only ledger for storage, results of the executed function instances.

6400

FIG. 47


5002
5004
5007
Processor
Append only ledger
5009

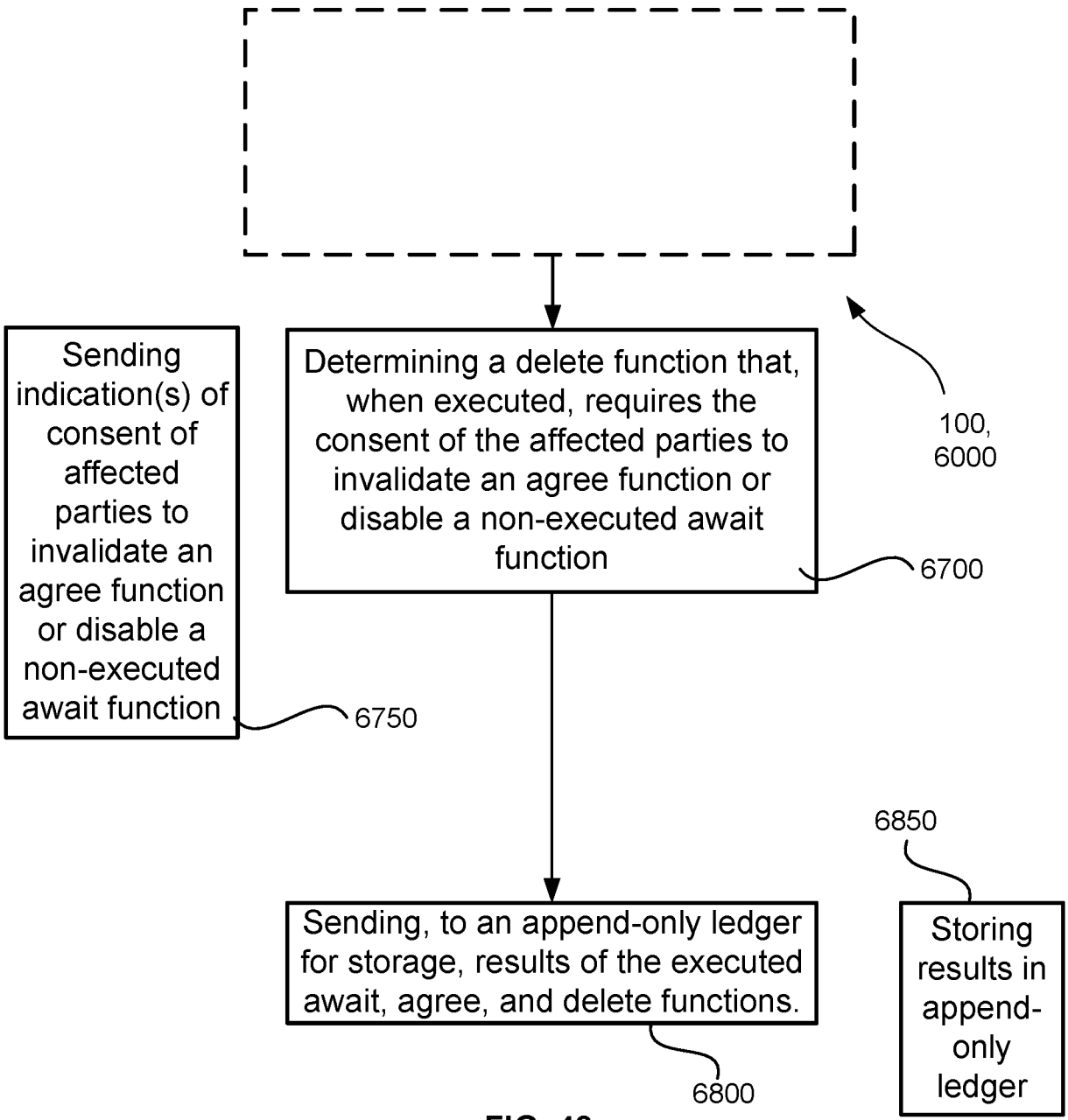

Sending indication(s) of consent of affected parties to invalidate an agree function or disable a non-executed await function
6750
Determining a delete function that, when executed, requires the consent of the affected parties to invalidate an agree function or disable a non-executed await function
6700
100, 6000
Sending, to an append-only ledger for storage, results of the executed await, agree, and delete functions.
6800
6850
Storing results in append-only ledger

FIG. 48

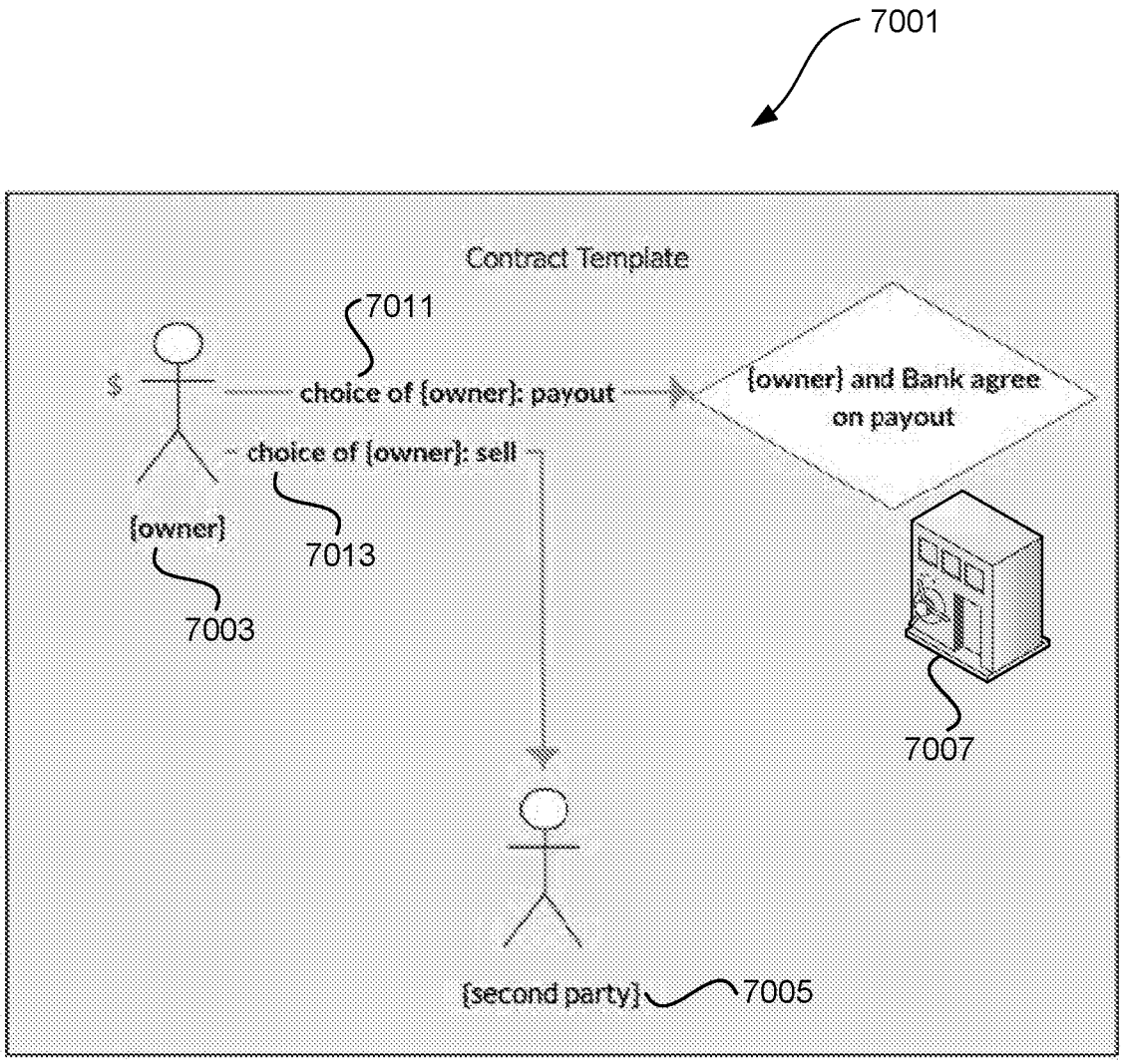

7001
Contract Template
7011
$
choice of {owner}: payout
{owner} and Bank agree on payout
choice of {owner}: sell
{owner}
7013
7003
7007
{second party}
7005

FIG. 49

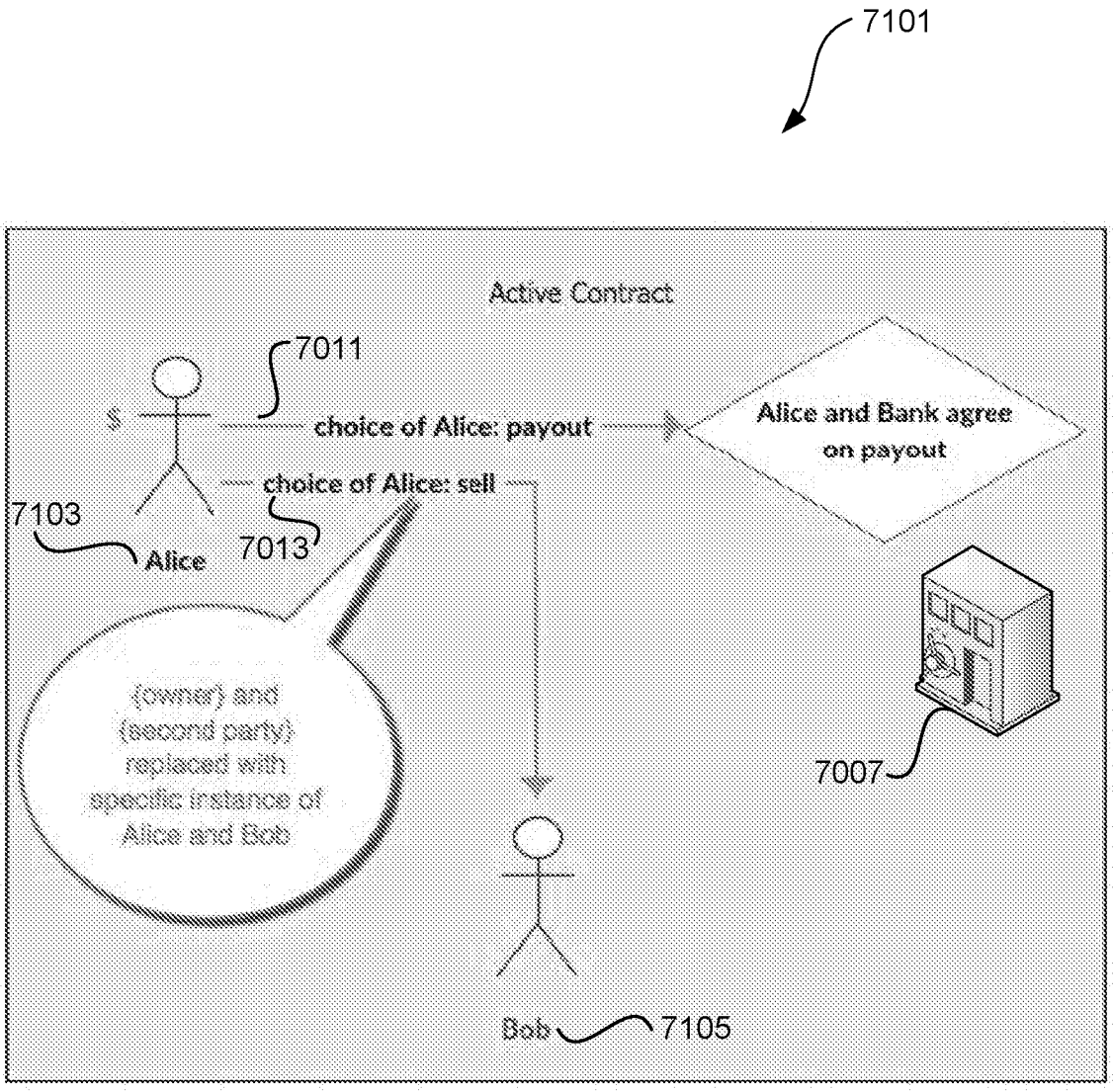

7101
Active Contract
7011
$
choice of Alice: payout
Alice and Bank agree on payout
choice of Alice: sell
7103
Alice
7013
(owner) and (second party) replaced with specific instance of Alice and Bob
7007
Bob
7105

FIG. 50

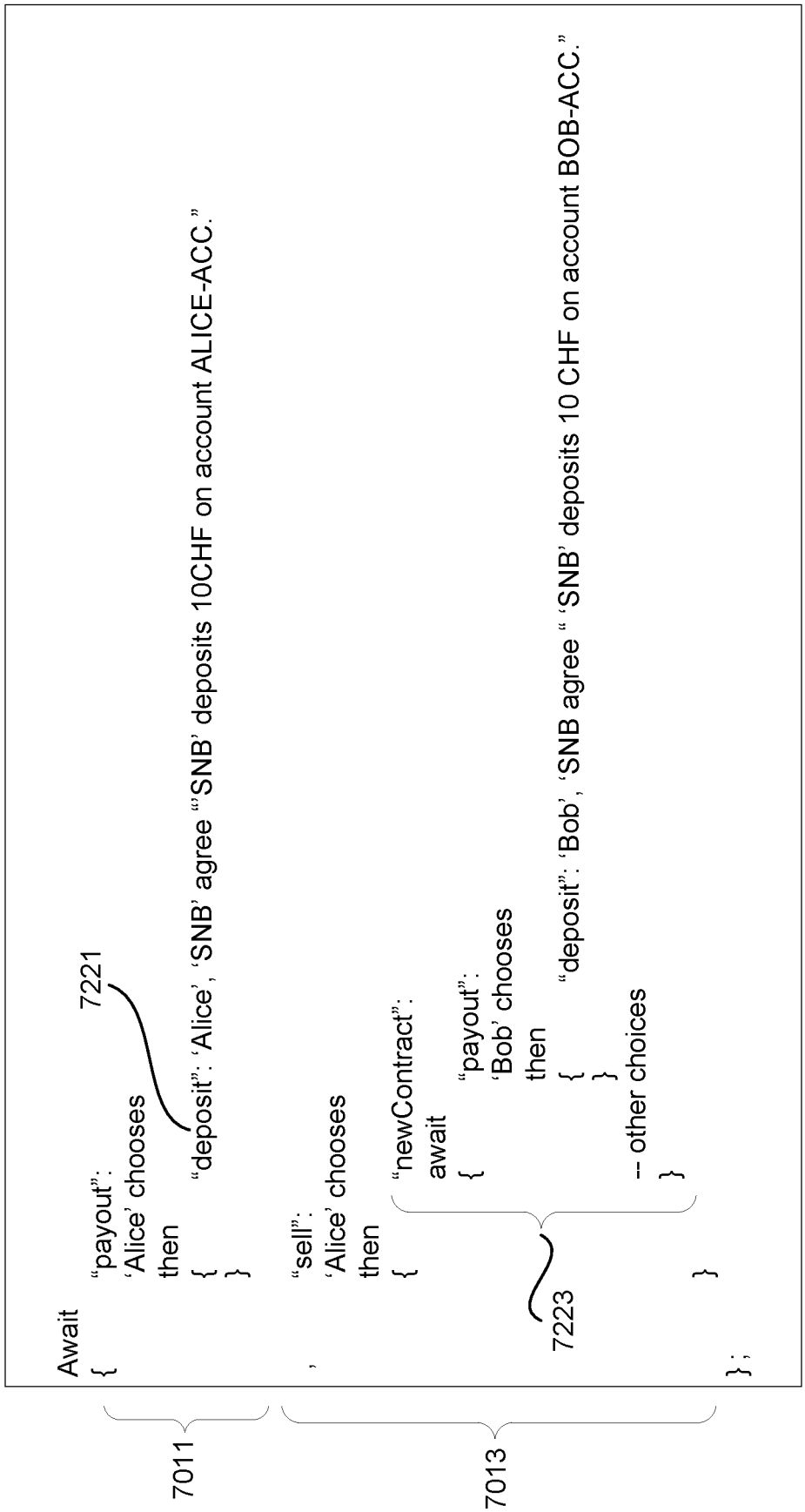

7201
Await
{
"payout":
'Alice' chooses
then
{
"deposit": 'Alice', 'SNB' agree "'SNB' deposits 10CHF on account ALICE-ACC."
}
7221
,
"sell":
'Alice' chooses
then
{
"newContract":
await
{
"payout":
'Bob' chooses
then
{
"deposit": 'Bob', 'SNB agree " 'SNB' deposits 10 CHF on account BOB-ACC."
}
-- other choices
}
}
7223
}
};
7011
7013

FIG. 51

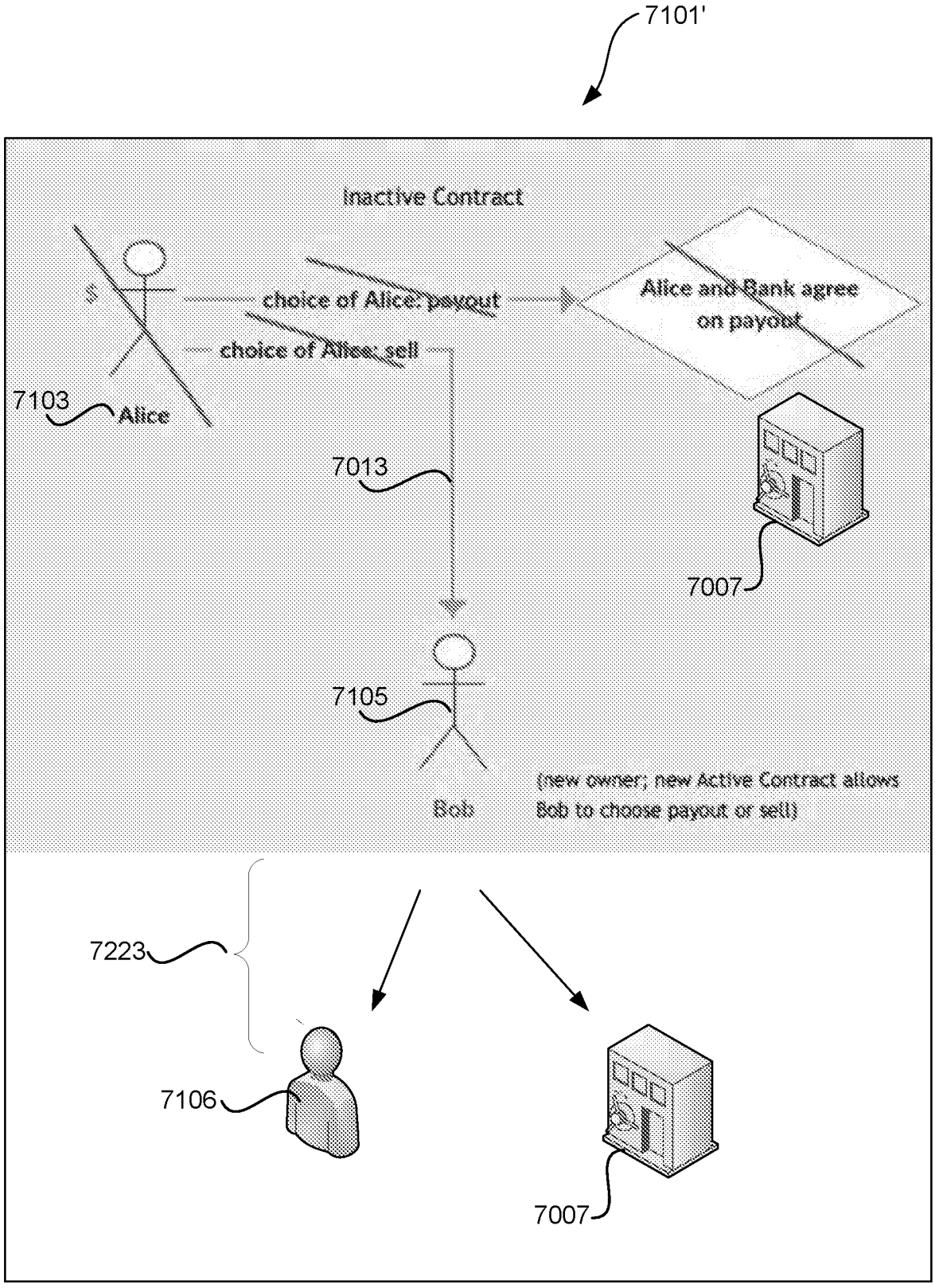

7101'
Inactive Contract
$
choice of Alice: payout
choice of Alice: sell
Alice and Bank agree on payout
7103
Alice
7013
7007
7105
Bob
(new owner; new Active Contract allows Bob to choose payout or sell)
7223
7106
7007

FIG. 52

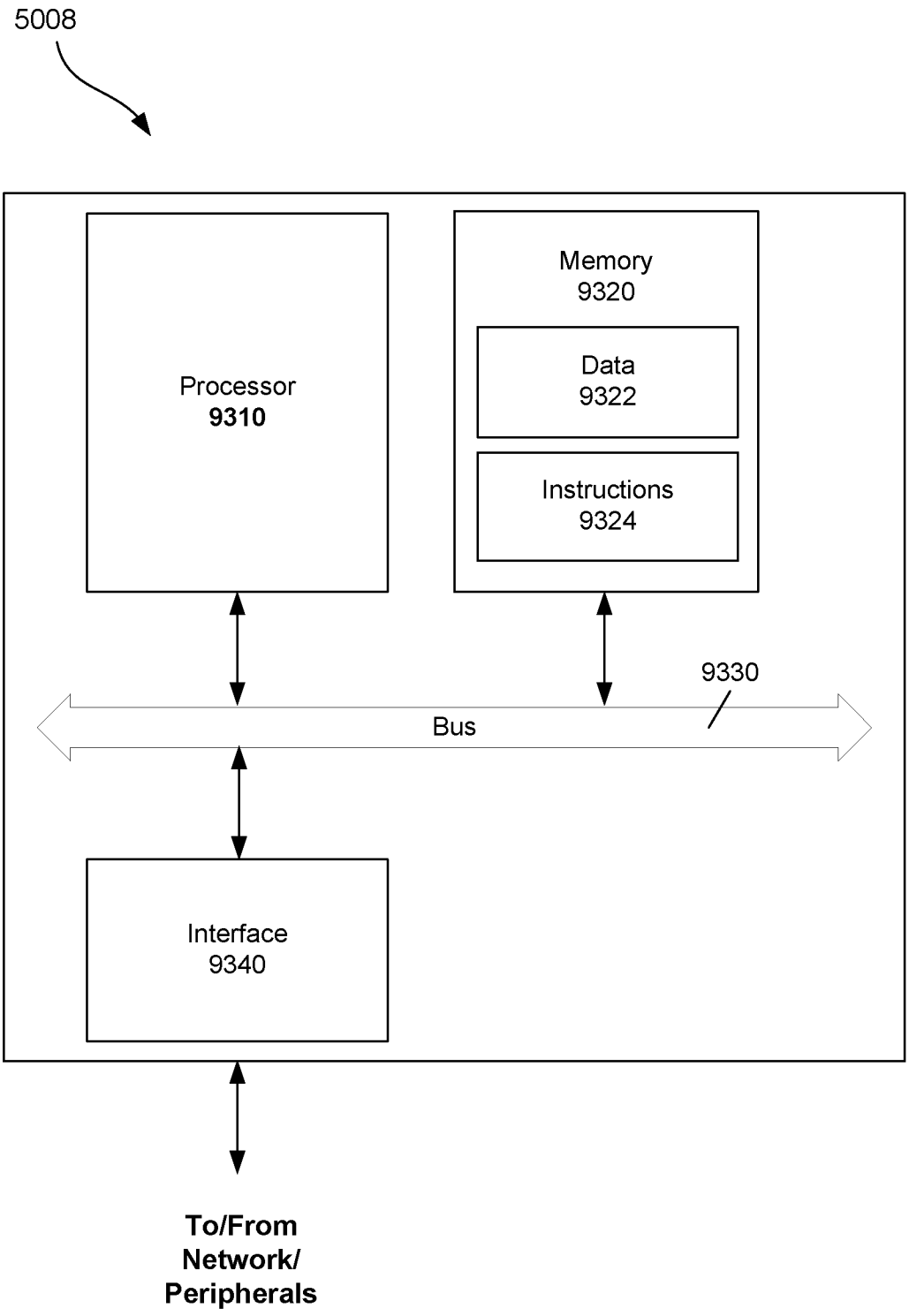

5008
Processor
9310
Memory
9320
Data
9322
Instructions
9324
9330
Bus
Interface
9340
To/From
Network/
Peripherals

FIG. 53