Fig. 1

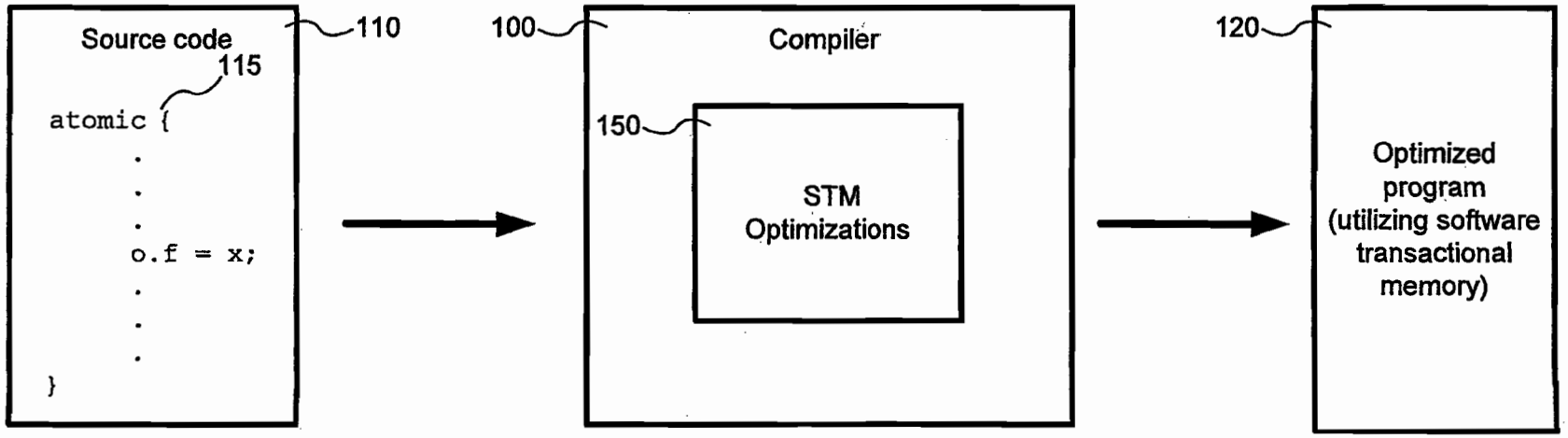

Source code
110
115
atomic {
o.f = x;
}
100
Compiler
150
STM
Optimizations
120
Optimized
program
(utilizing software
transactional
memory)

Fig. 2

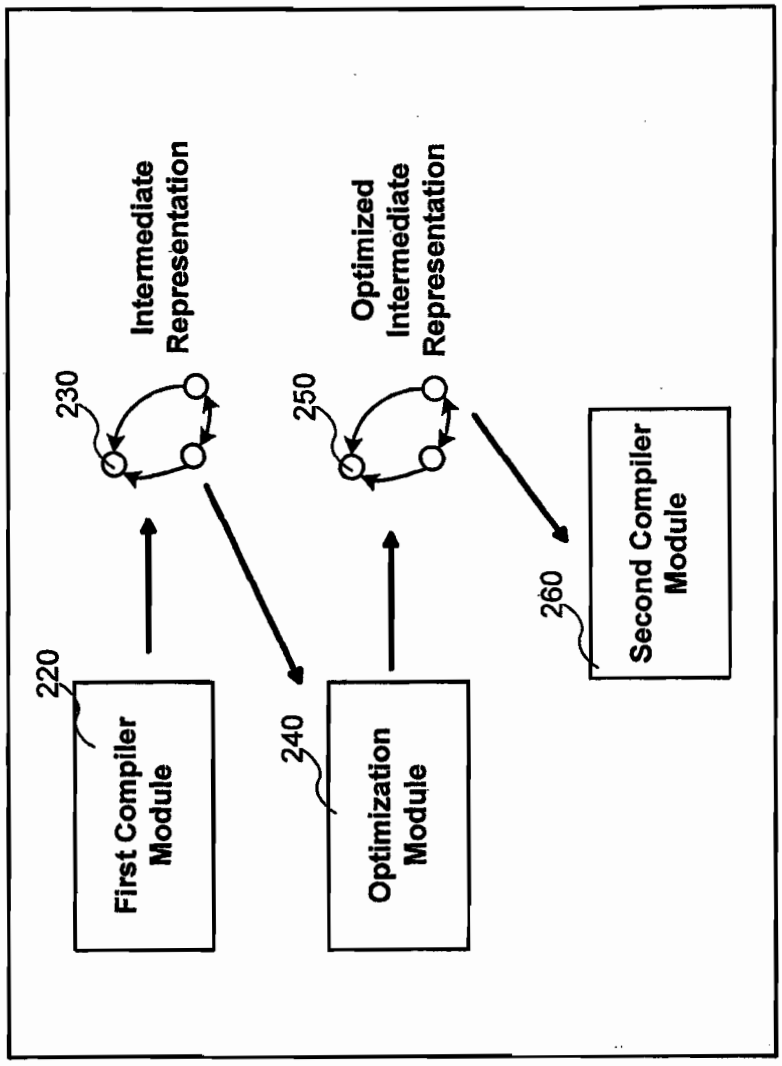

First Compiler Module
220
Intermediate Representation
230
Optimization Module
240
Optimized Intermediate Representation
250
Second Compiler Module
260
Compiler 100

Fig. 3

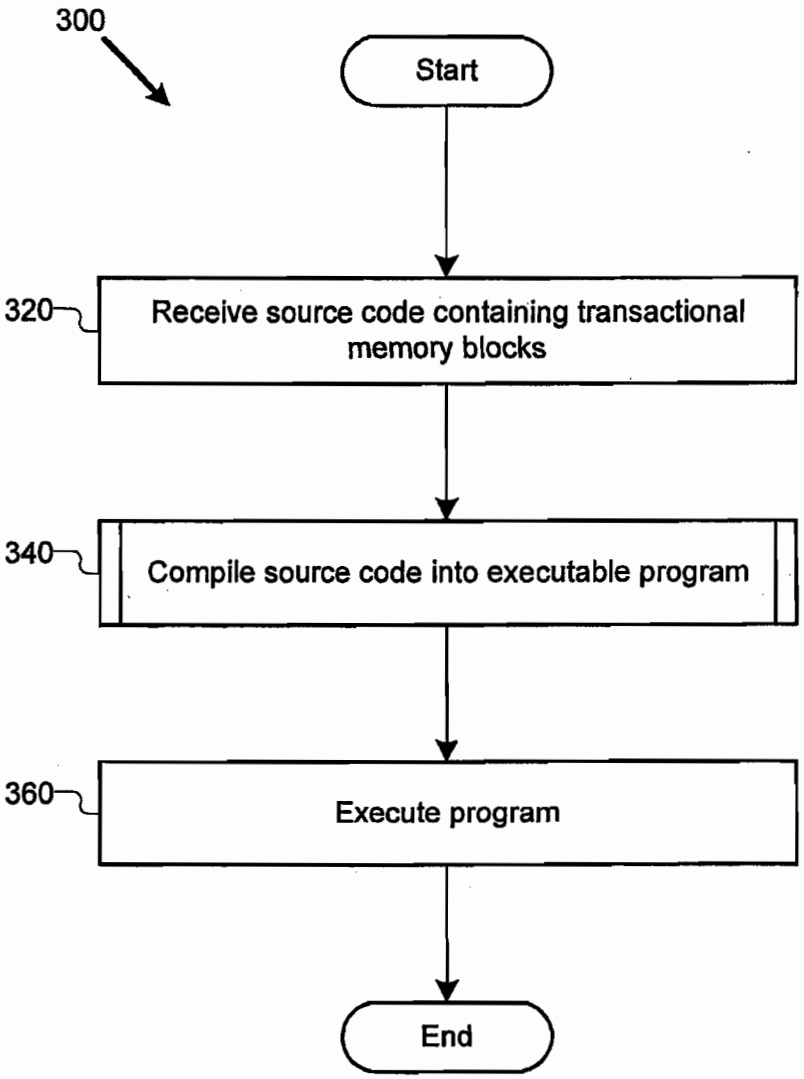

300
Start
320
Receive source code containing transactional memory blocks
340
Compile source code into executable program
360
Execute program
End

Fig. 4

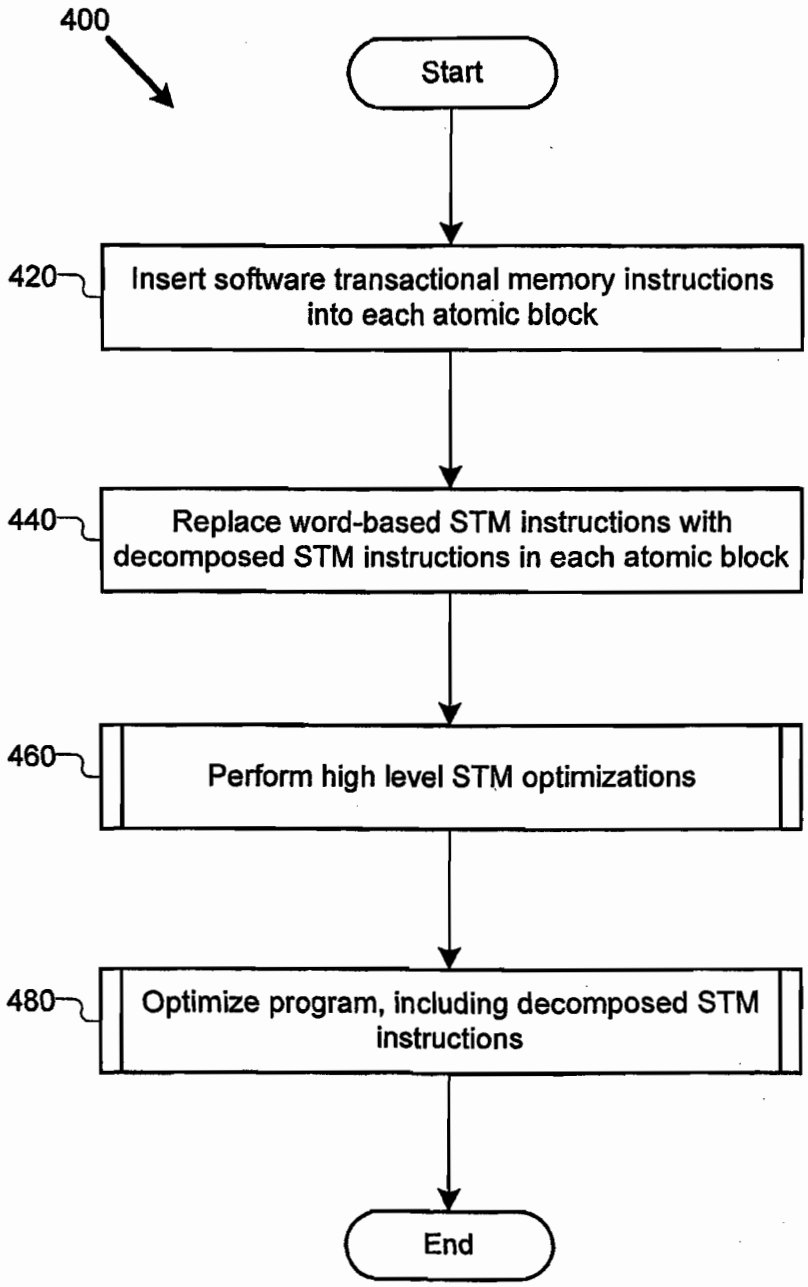

400
Start
420
Insert software transactional memory instructions into each atomic block
440
Replace word-based STM instructions with decomposed STM instructions in each atomic block
460
Perform high level STM optimizations
480
Optimize program, including decomposed STM instructions
End

Fig. 5

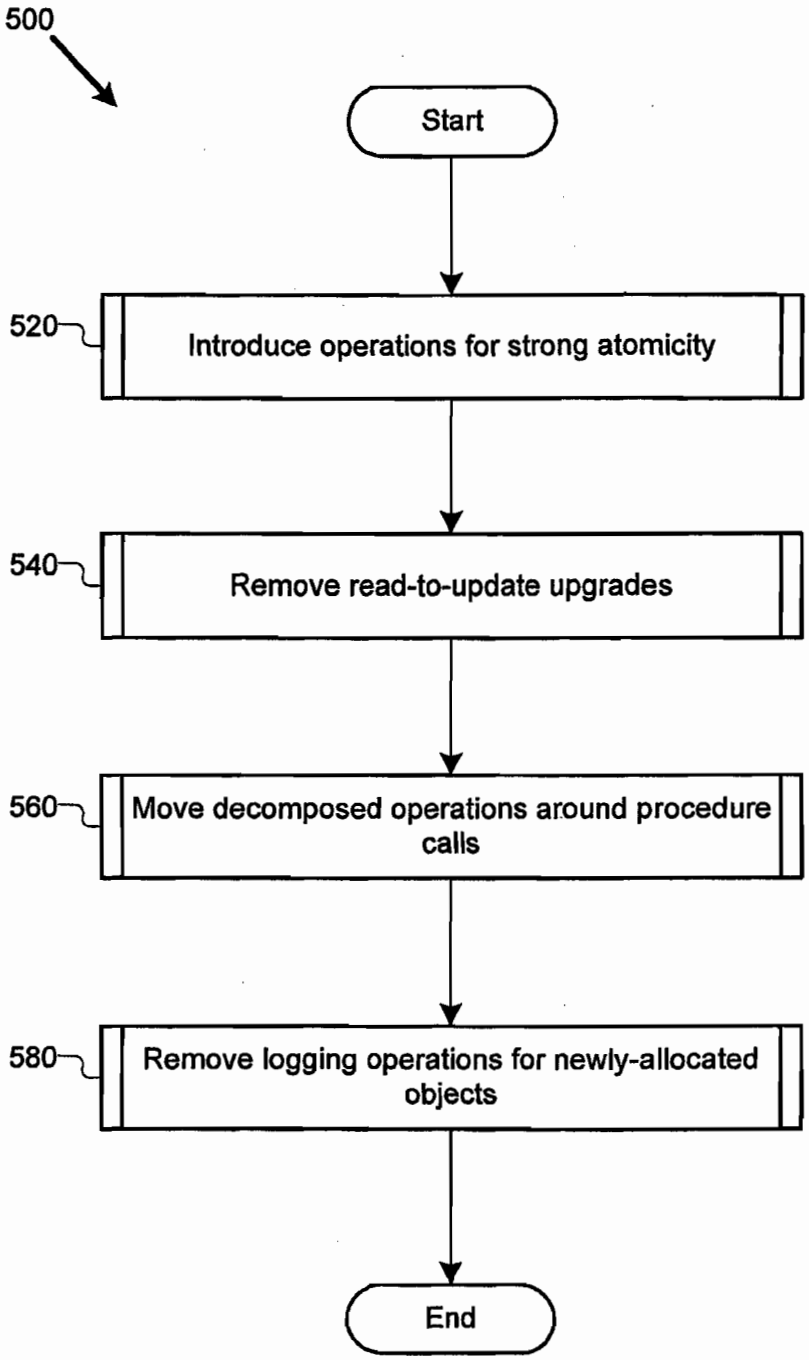

500
Start
520
Introduce operations for strong atomicity
540
Remove read-to-update upgrades
560
Move decomposed operations around procedure calls
580
Remove logging operations for newly-allocated objects
End

Fig. 6

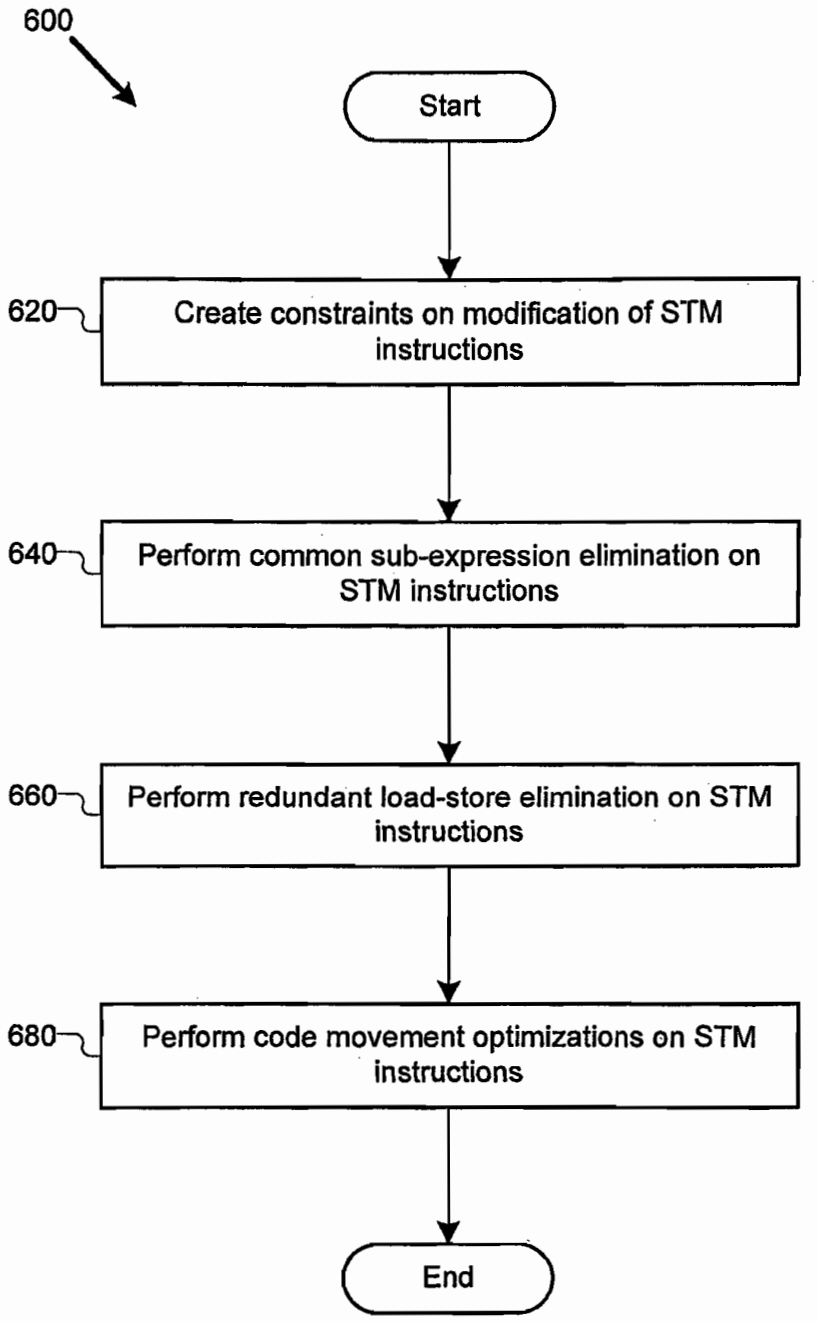

600
Start
620
Create constraints on modification of STM instructions
640
Perform common sub-expression elimination on STM instructions
660
Perform redundant load-store elimination on STM instructions
680
Perform code movement optimizations on STM instructions
End

Fig. 7

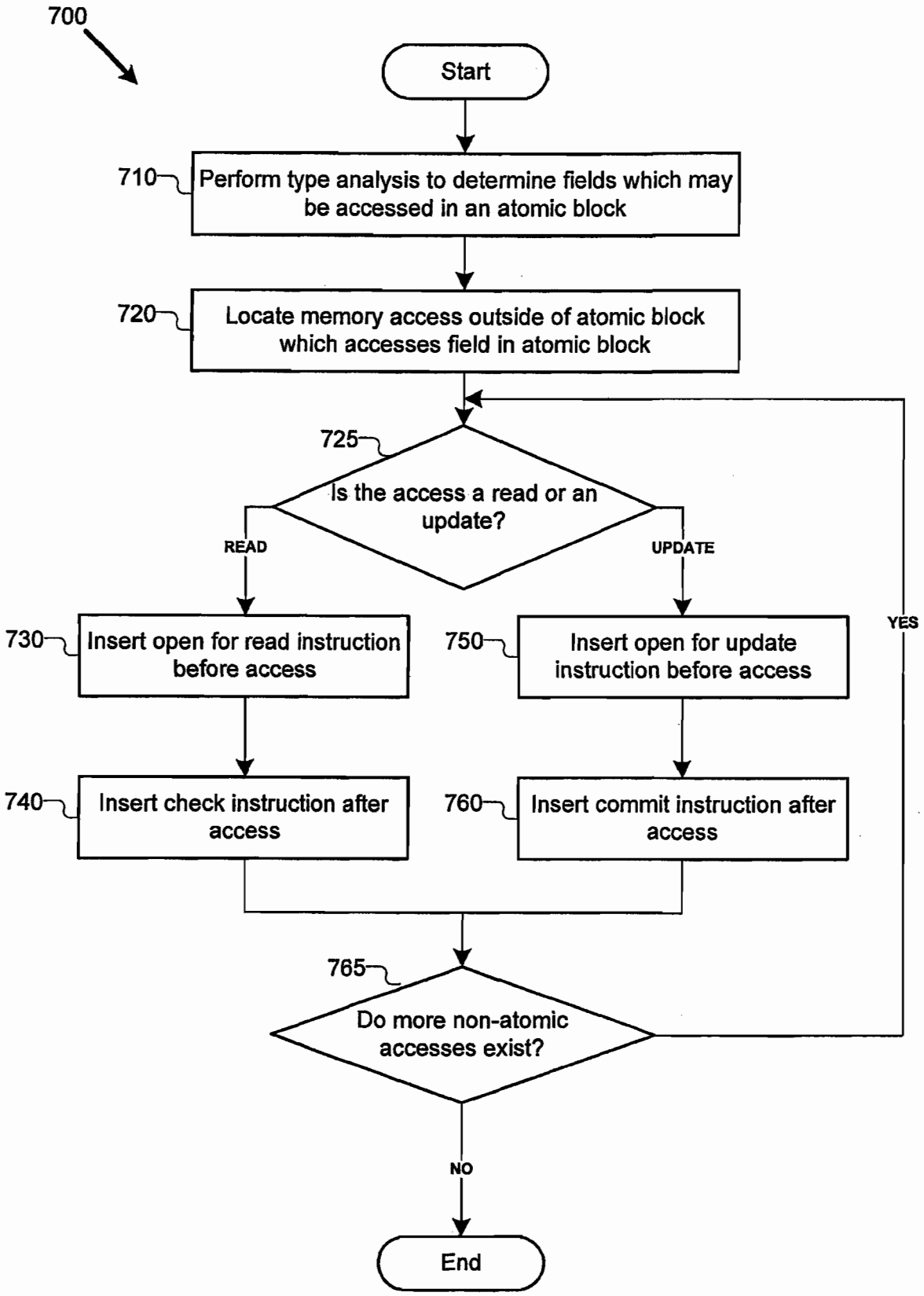

700
Start
710
Perform type analysis to determine fields which may be accessed in an atomic block
720
Locate memory access outside of atomic block which accesses field in atomic block
725
Is the access a read or an update?
READ
UPDATE
730
Insert open for read instruction before access
750
Insert open for update instruction before access
740
Insert check instruction after access
760
Insert commit instruction after access
765
Do more non-atomic accesses exist?
YES
NO
End

Fig. 8

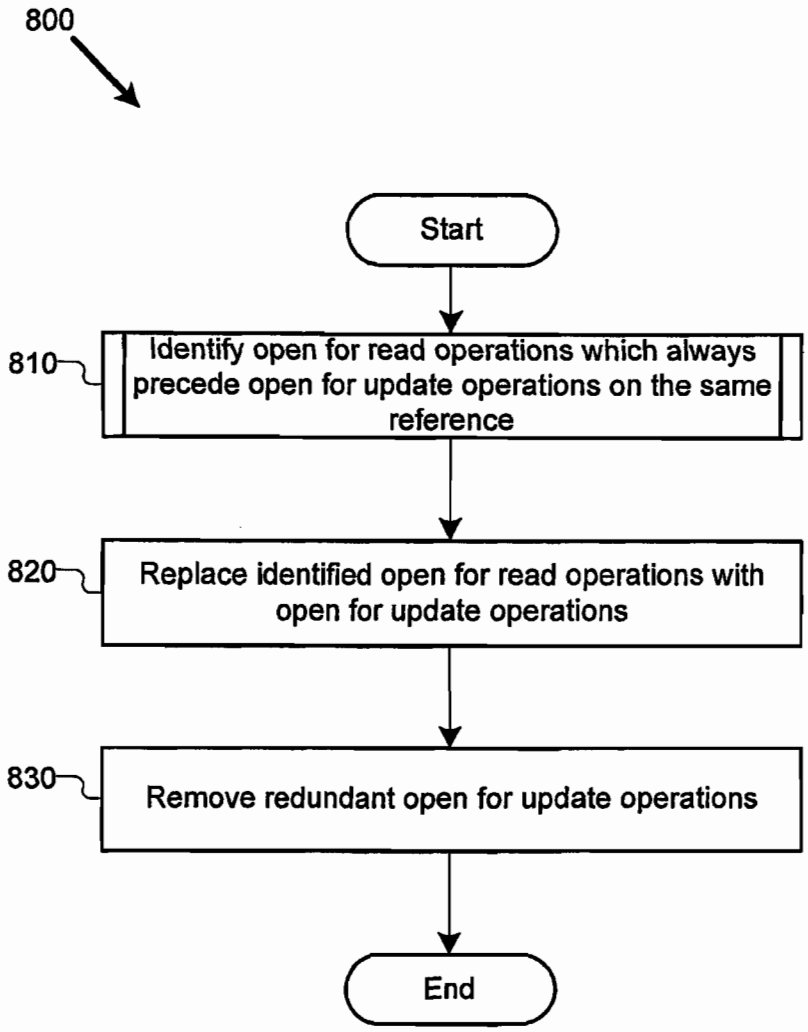

800
Start
810
Identify open for read operations which always precede open for update operations on the same reference
820
Replace identified open for read operations with open for update operations
830
Remove redundant open for update operations
End

Fig. 9

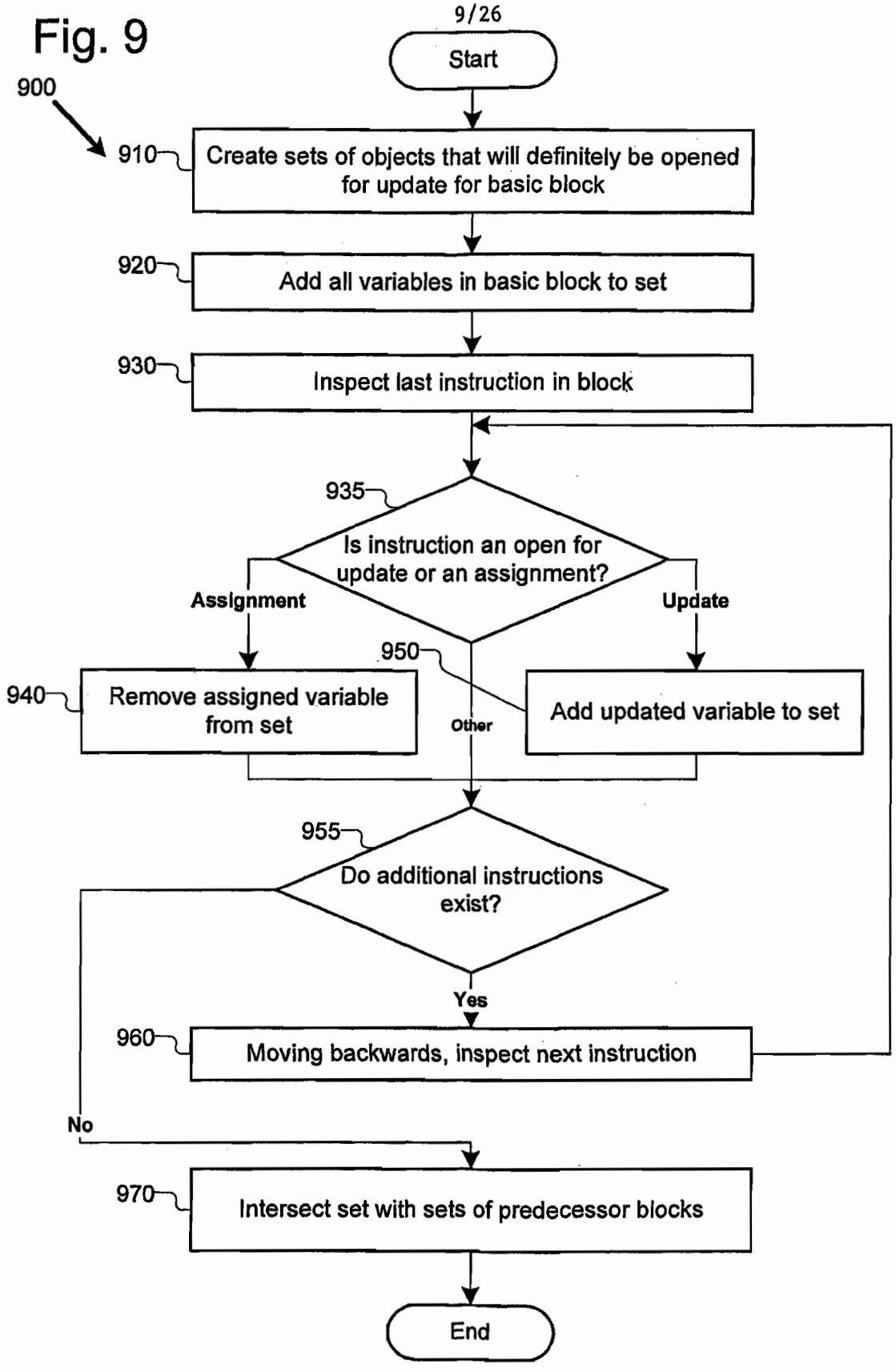

900
Start
910
Create sets of objects that will definitely be opened for update for basic block
920
Add all variables in basic block to set
930
Inspect last instruction in block
935
Is instruction an open for update or an assignment?
Assignment
Update
940
Remove assigned variable from set
950
Other
Add updated variable to set
955
Do additional instructions exist?
Yes
960
Moving backwards, inspect next instruction
No
970
Intersect set with sets of predecessor blocks
End

Fig. 10

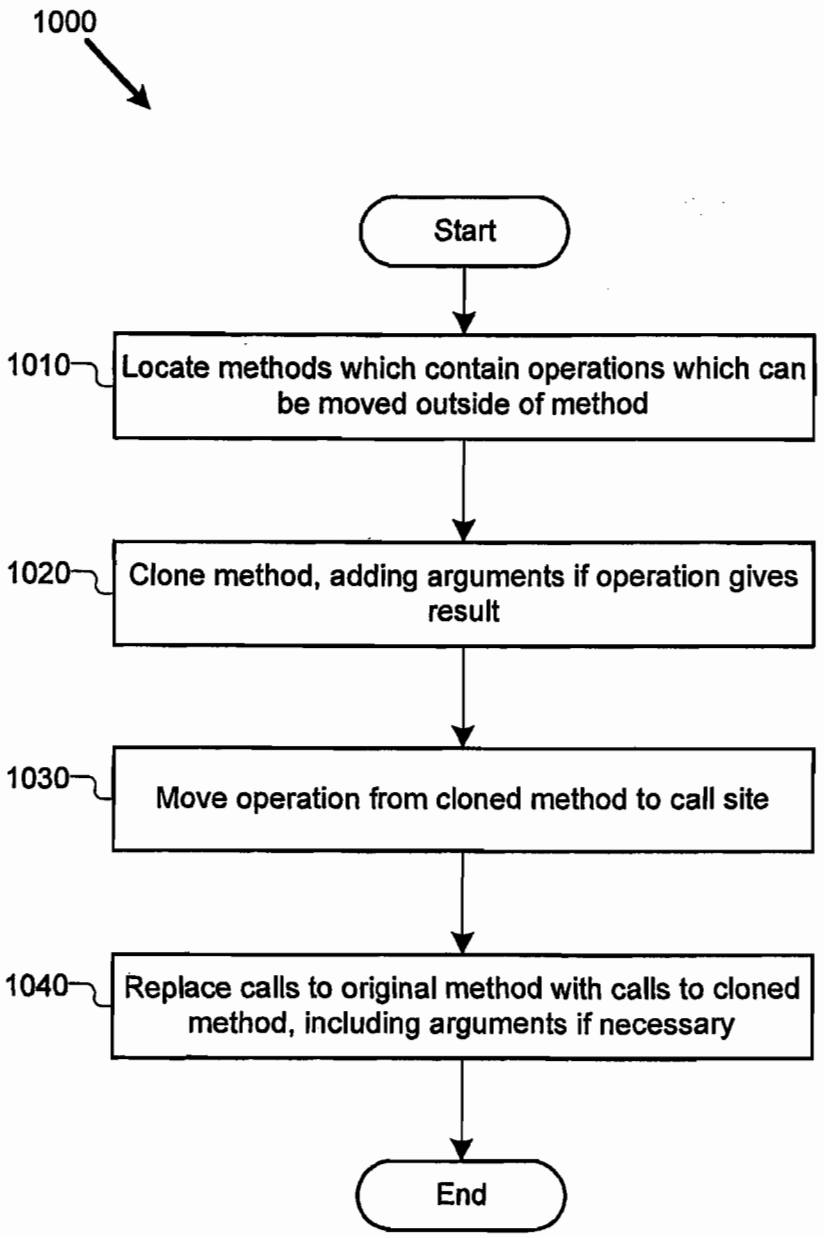

1000
Start
1010
Locate methods which contain operations which can be moved outside of method
1020
Clone method, adding arguments if operation gives result
1030
Move operation from cloned method to call site
1040
Replace calls to original method with calls to cloned method, including arguments if necessary
End

Fig. 11

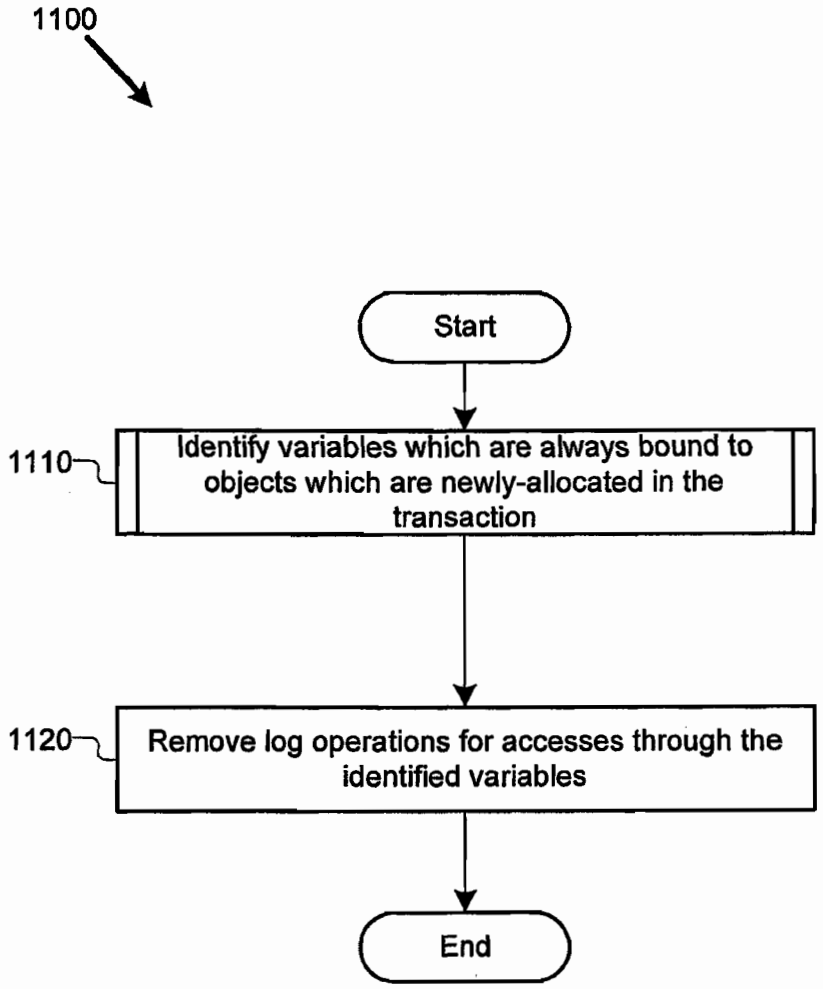

1100
Start
1110
Identify variables which are always bound to objects which are newly-allocated in the transaction
1120
Remove log operations for accesses through the identified variables
End

Fig. 12

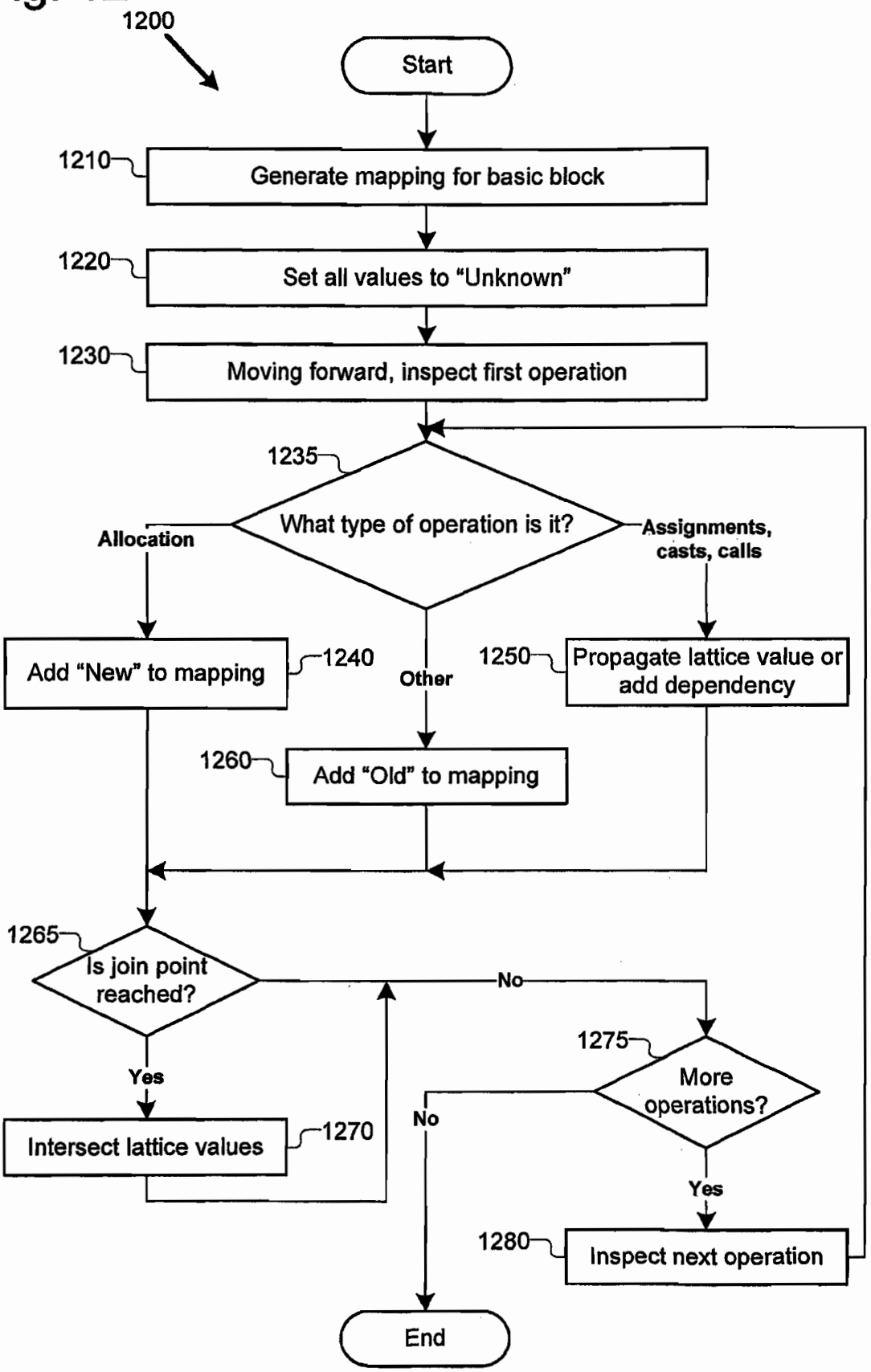

1200
Start
1210
Generate mapping for basic block
1220
Set all values to "Unknown"
1230
Moving forward, inspect first operation
1235
What type of operation is it?
Allocation
Assignments, casts, calls
Other
Add "New" to mapping
1240
1250
Propagate lattice value or add dependency
1260
Add "Old" to mapping
1265
Is join point reached?
No
Yes
1275
More operations?
No
Yes
Intersect lattice values
1270
1280
Inspect next operation
End

Fig. 13

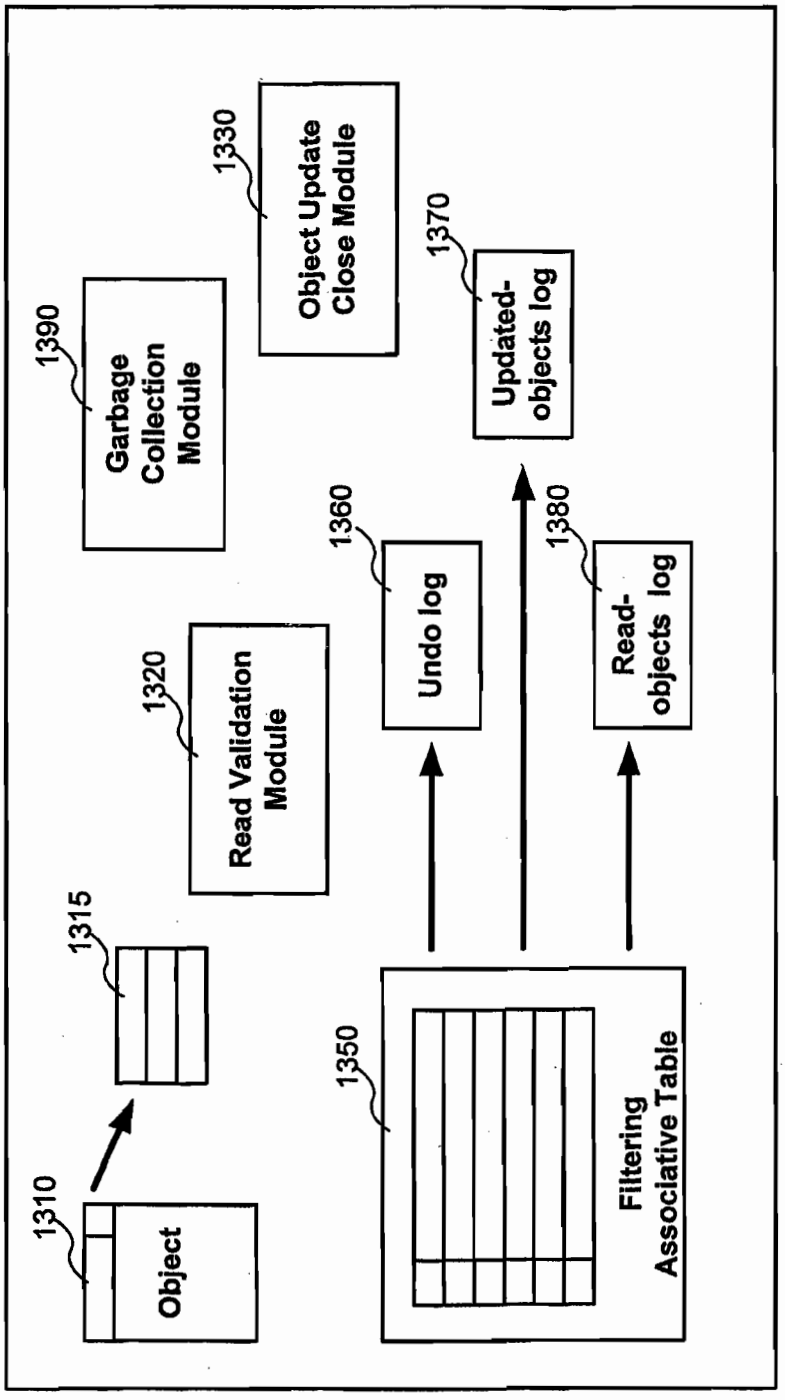

Runtime environment
1300
1310
Object
1315
1320
Read Validation Module
1330
Object Update Close Module
1350
Filtering Associative Table
1360
Undo log
1370
Updated-objects log
1380
Read-objects log
1390
Garbage Collection Module

Fig. 14a

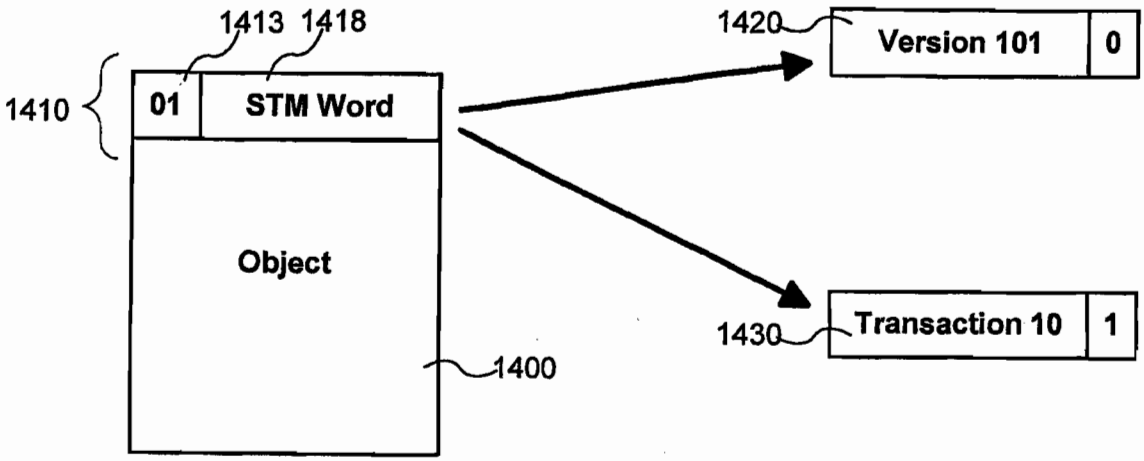


Fig. 14b

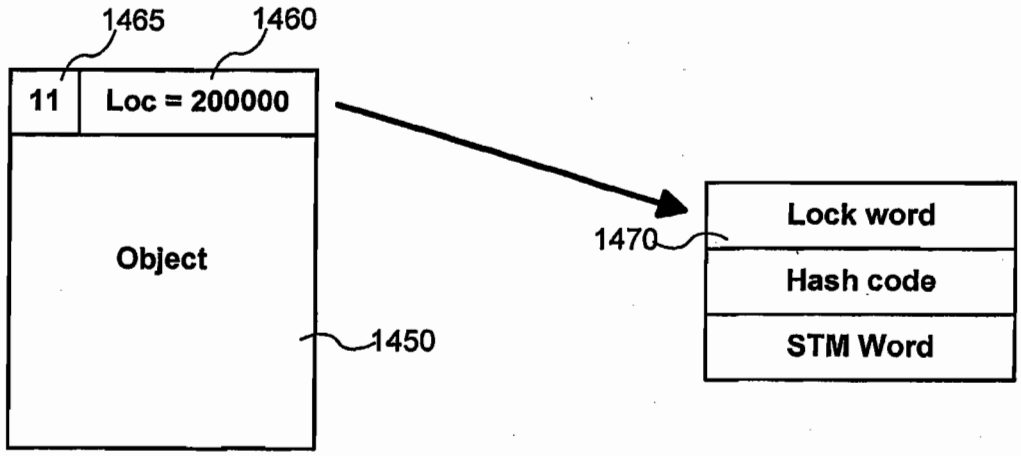


Fig. 15a

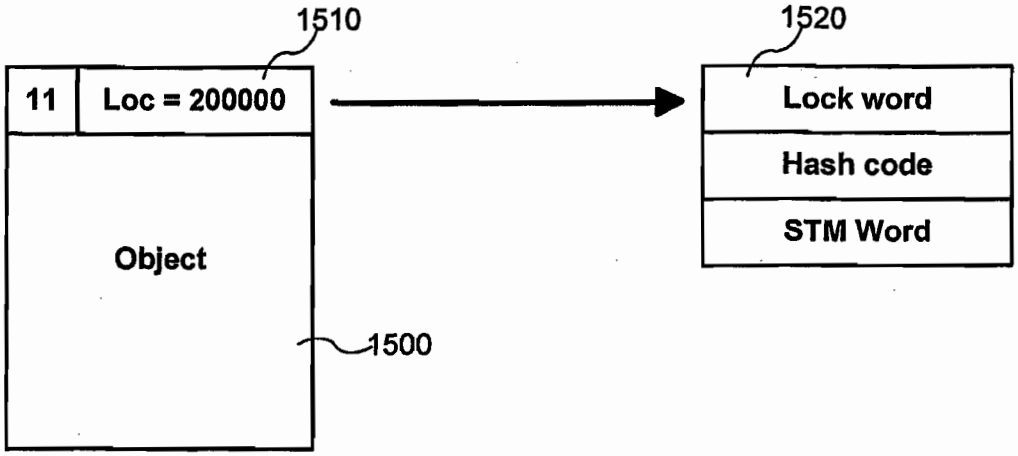


Fig. 15b

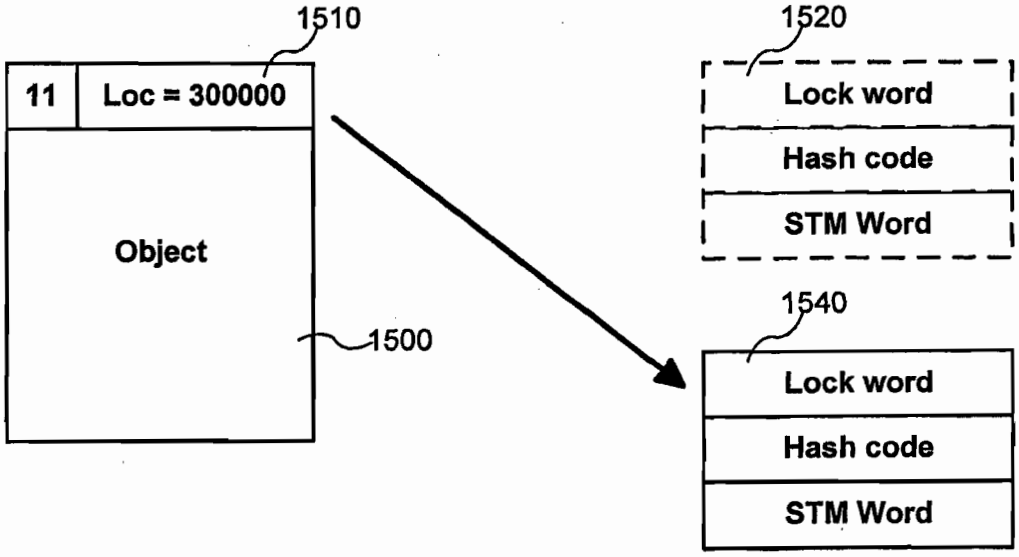


Fig. 16

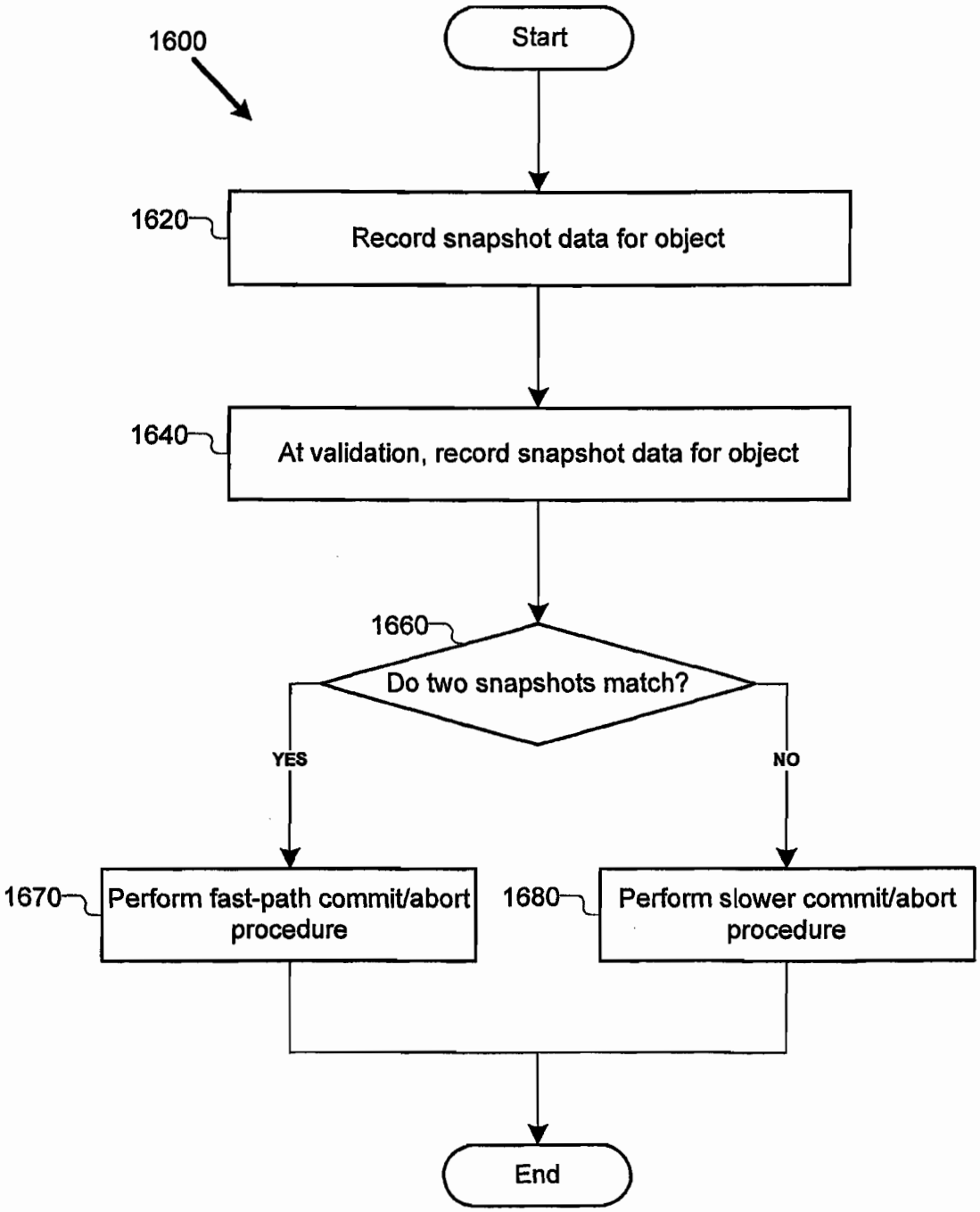

1600
Start
1620
Record snapshot data for object
1640
At validation, record snapshot data for object
1660
Do two snapshots match?
YES
NO
1670
Perform fast-path commit/abort procedure
1680
Perform slower commit/abort procedure
End

Fig. 17

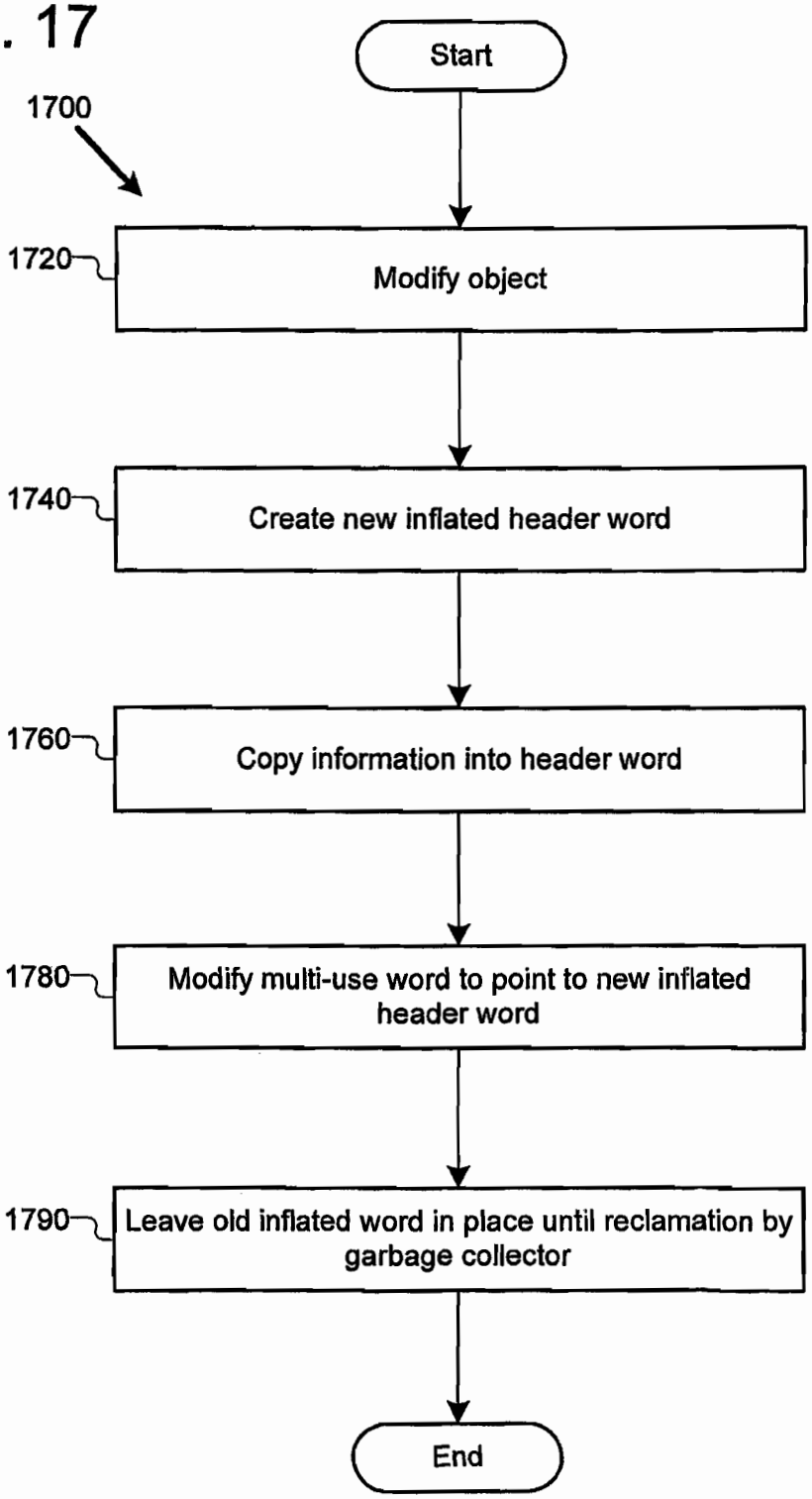

Start
1700
1720
Modify object
1740
Create new inflated header word
1760
Copy information into header word
1780
Modify multi-use word to point to new inflated header word
1790
Leave old inflated word in place until reclamation by garbage collector
End

Fig. 18a

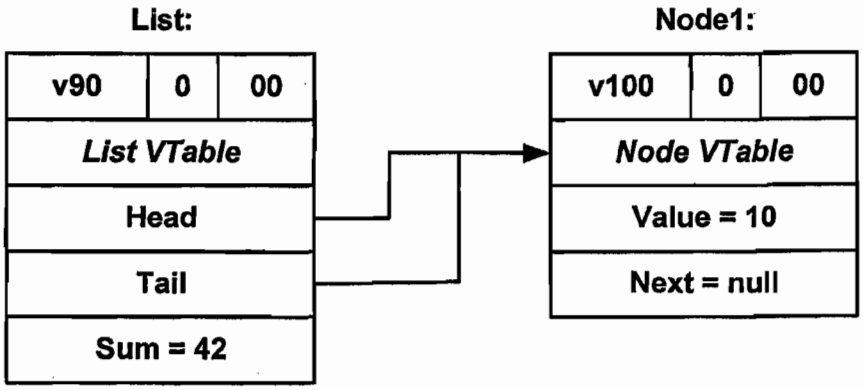


Fig. 18b

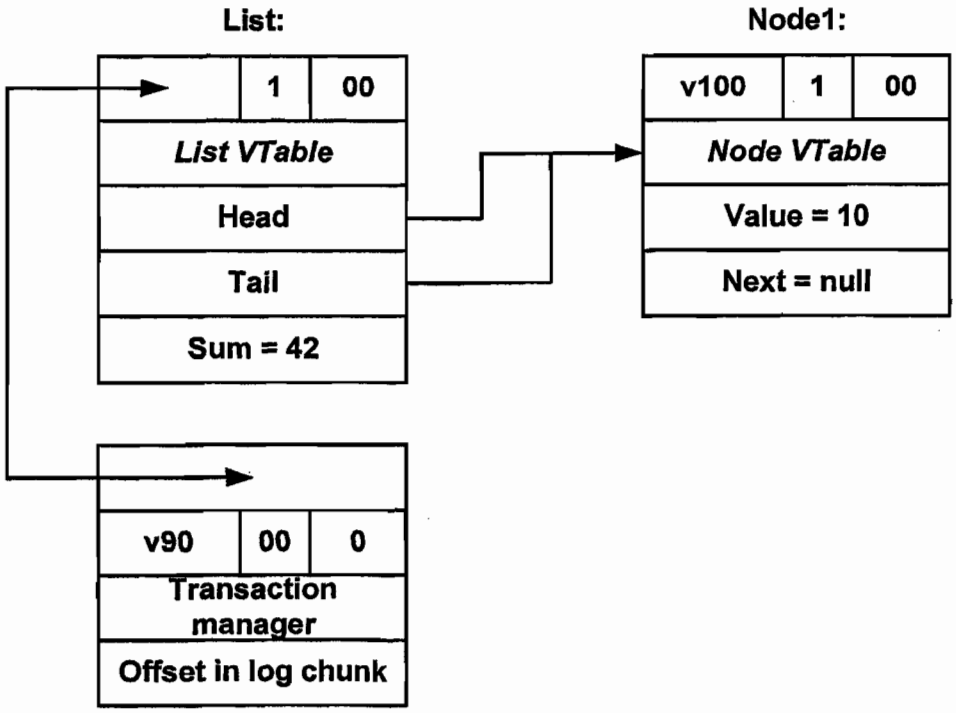


Fig. 19a

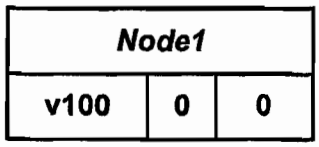


Fig. 19b

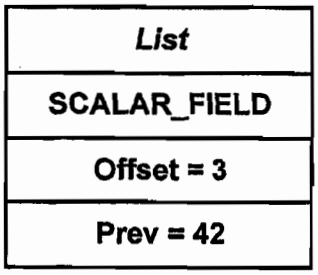
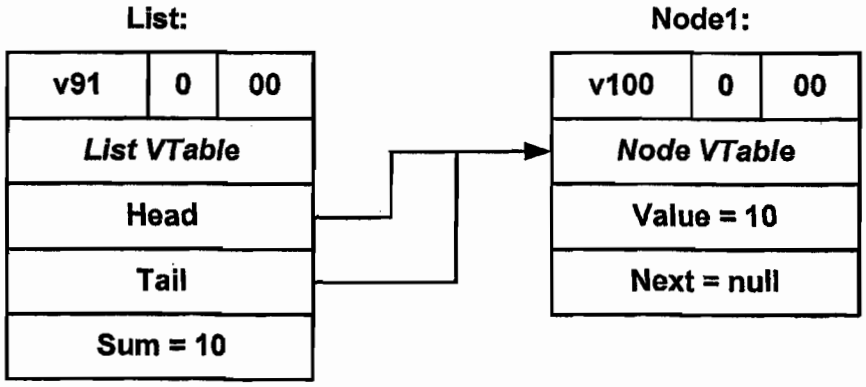


Fig. 19c



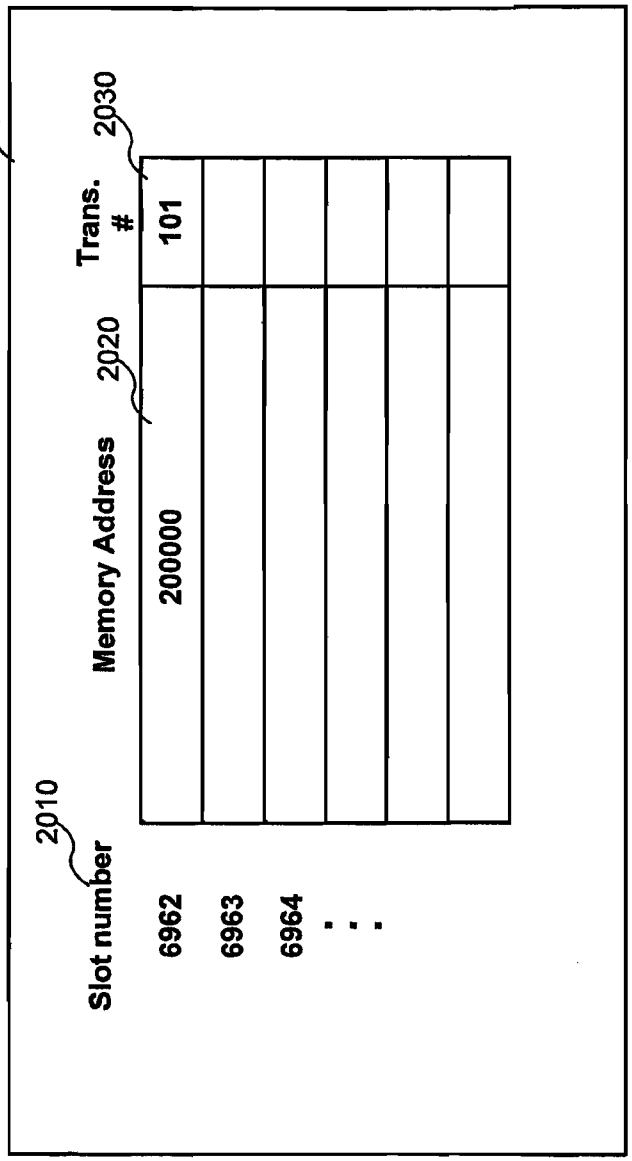

2000
2010
2020
2030
Slot number
Memory Address
Trans. #
6962
6963
6964
200000
101
Associative Table

Fig. 20

Fig. 21

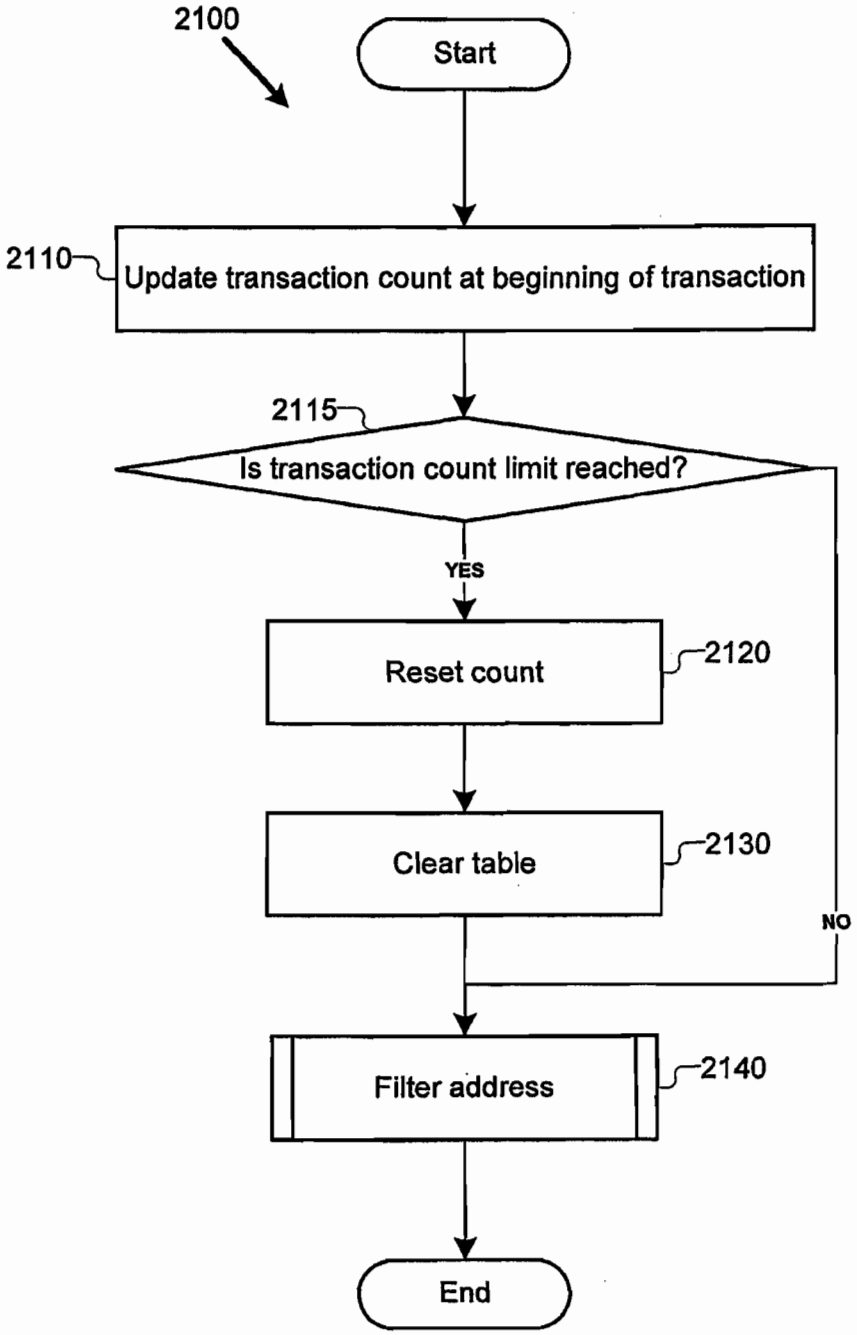

2100
Start
2110
Update transaction count at beginning of transaction
2115
Is transaction count limit reached?
YES
NO
Reset count
2120
Clear table
2130
Filter address
2140
End

Fig. 22

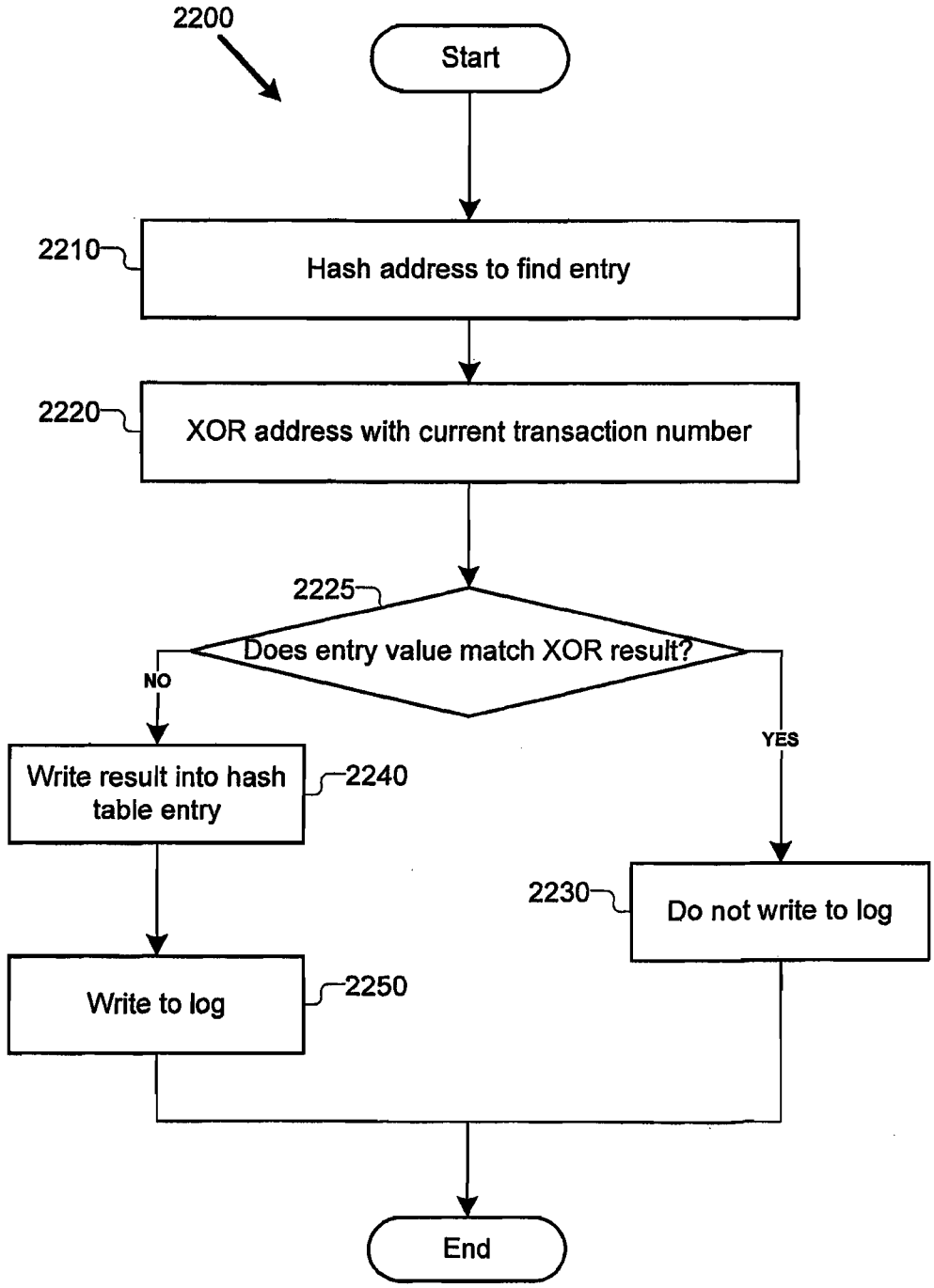

2200
Start
2210
Hash address to find entry
2220
XOR address with current transaction number
2225
Does entry value match XOR result?
NO
YES
Write result into hash table entry
2240
2230
Do not write to log
Write to log
2250
End

Fig. 23

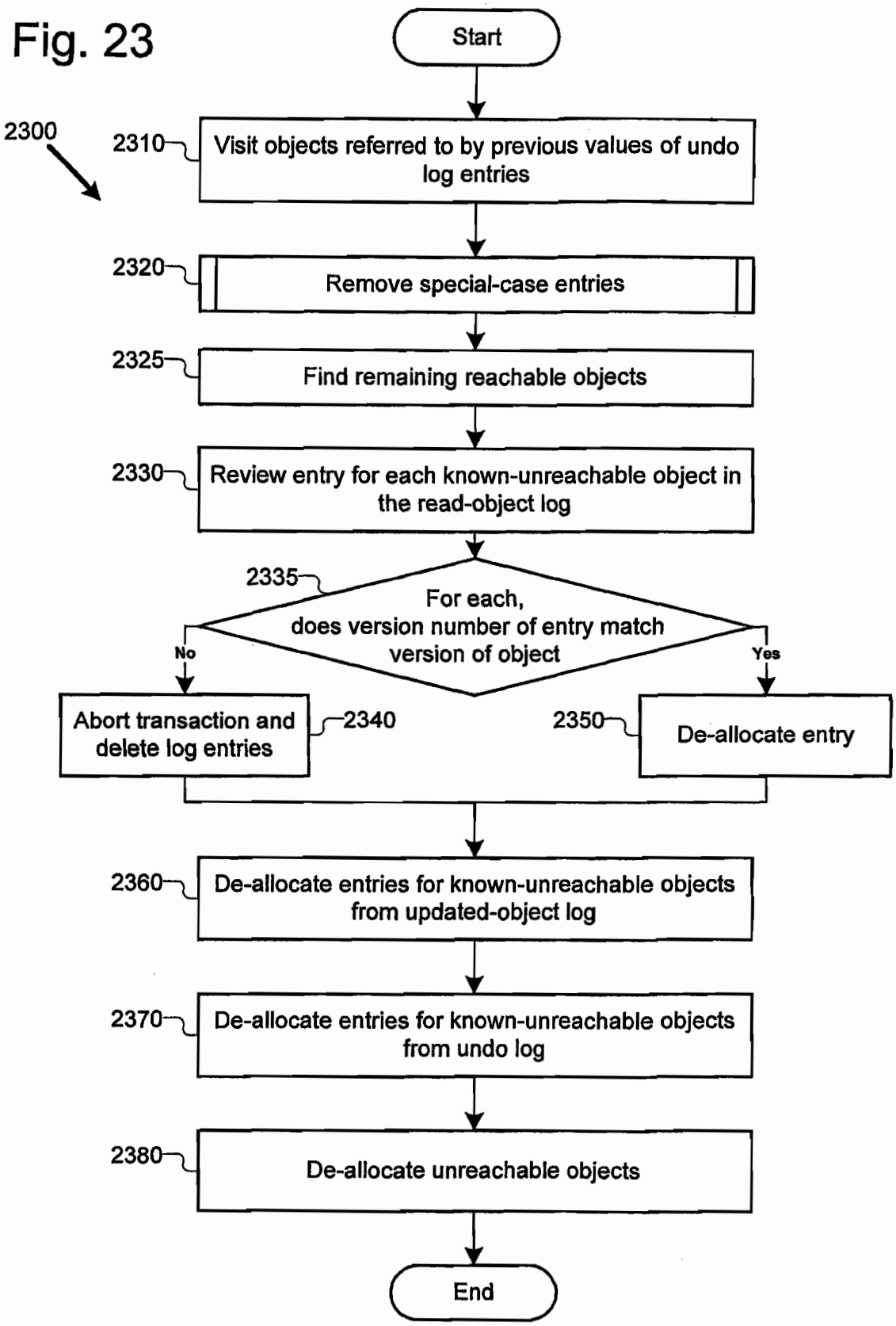

Start
2300
2310
Visit objects referred to by previous values of undo log entries
2320
Remove special-case entries
2325
Find remaining reachable objects
2330
Review entry for each known-unreachable object in the read-object log
2335
For each, does version number of entry match version of object
No
Yes
Abort transaction and delete log entries
2340
2350
De-allocate entry
2360
De-allocate entries for known-unreachable objects from updated-object log
2370
De-allocate entries for known-unreachable objects from undo log
2380
De-allocate unreachable objects
End

Fig. 24

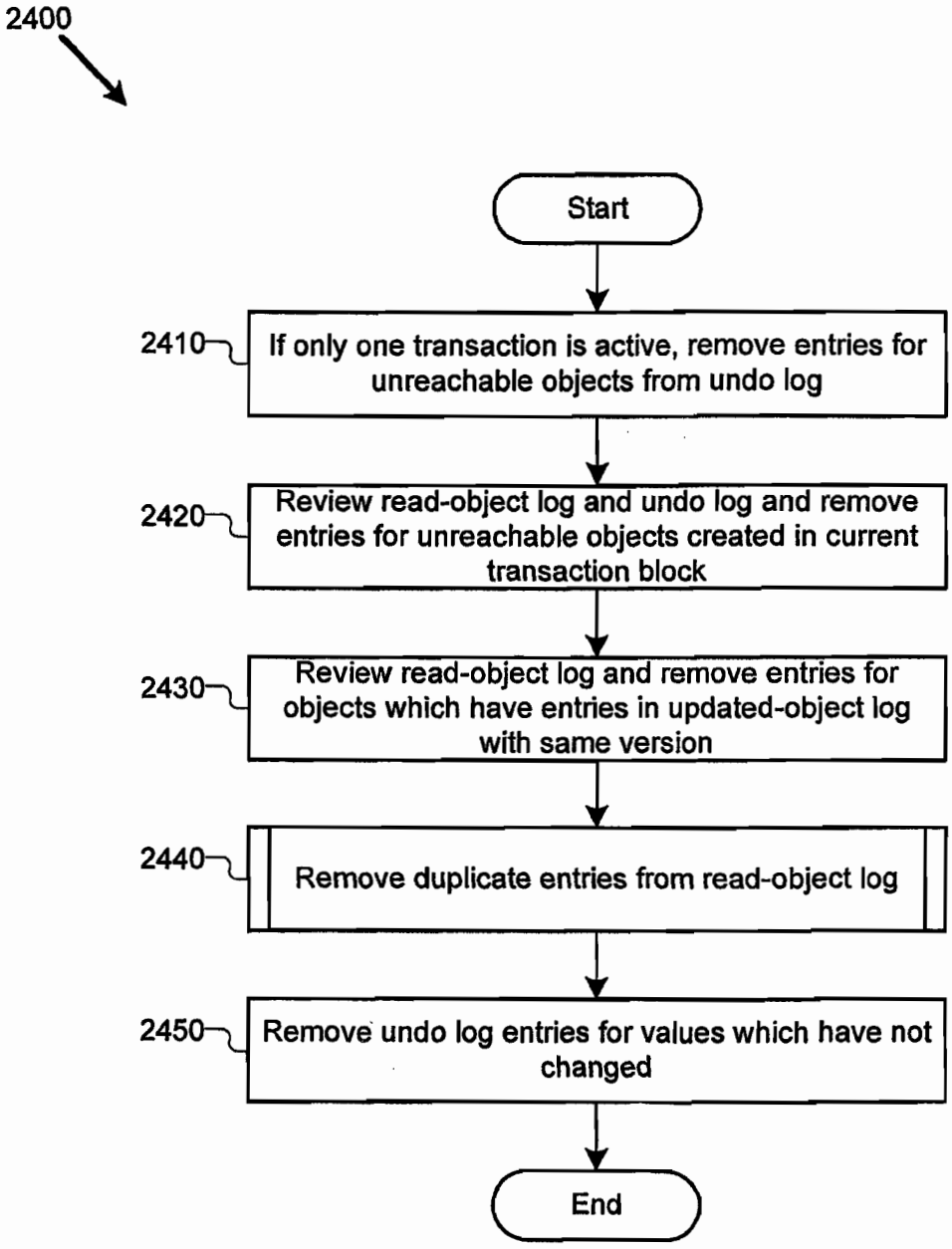

2400
Start
2410
If only one transaction is active, remove entries for unreachable objects from undo log
2420
Review read-object log and undo log and remove entries for unreachable objects created in current transaction block
2430
Review read-object log and remove entries for objects which have entries in updated-object log with same version
2440
Remove duplicate entries from read-object log
2450
Remove undo log entries for values which have not changed
End

Fig. 25

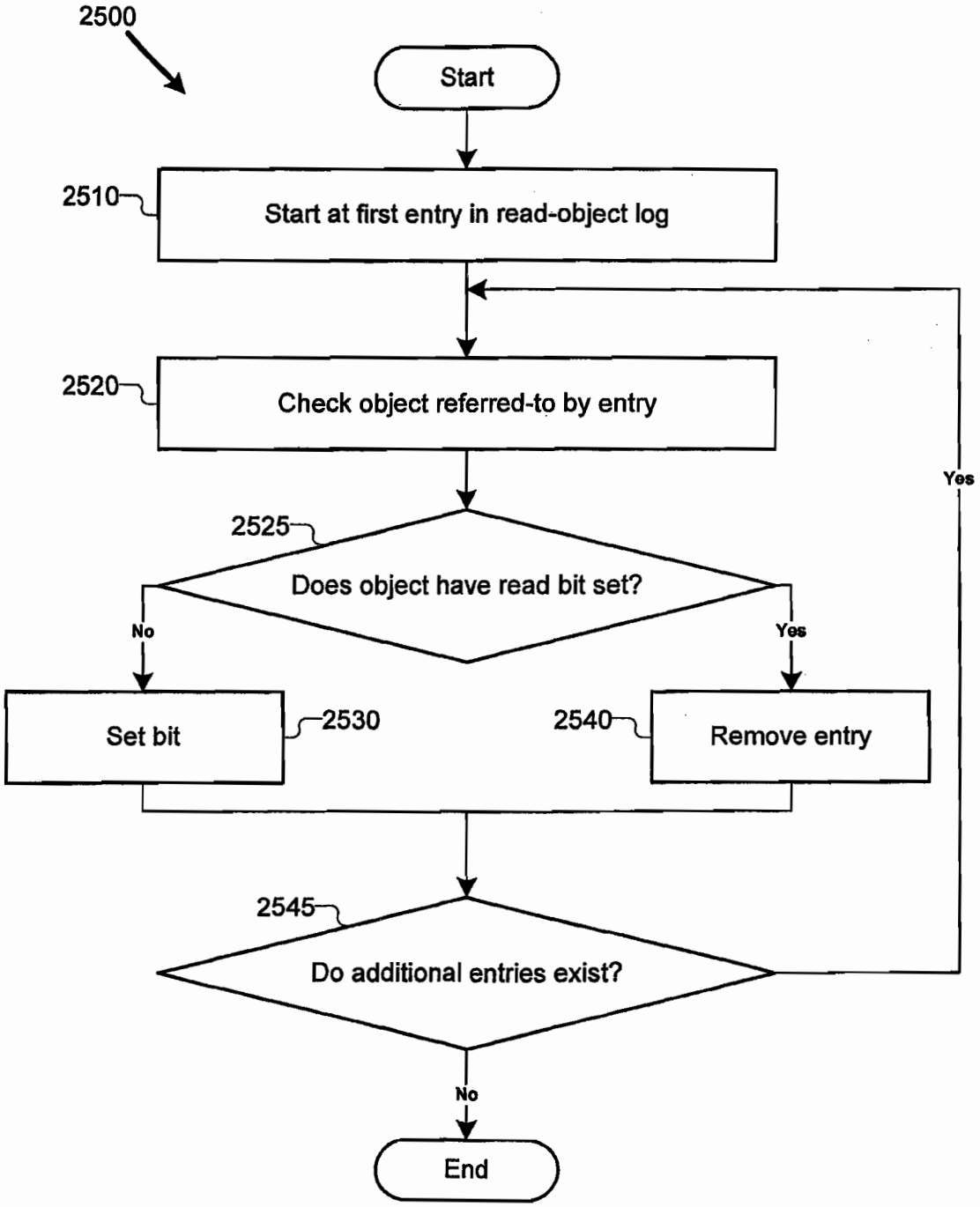
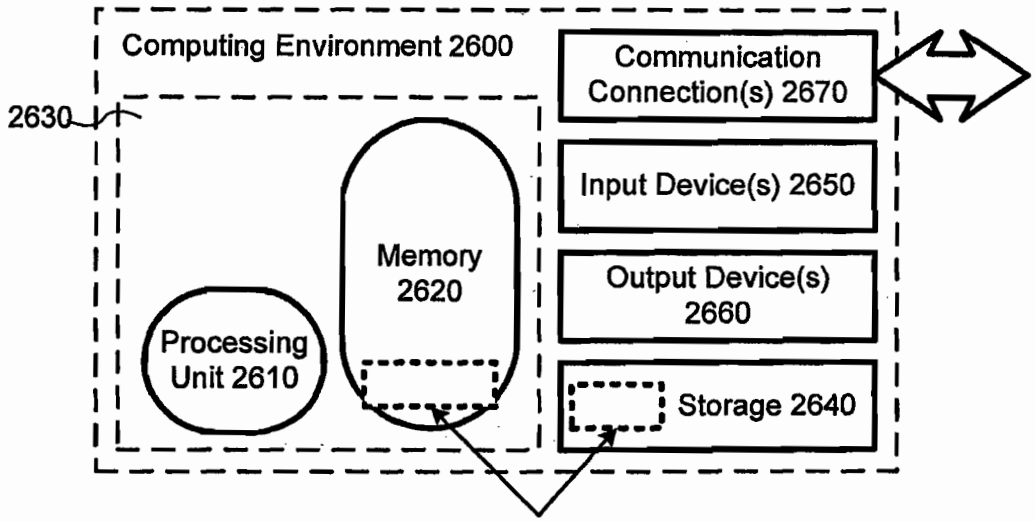

2500
Start
2510
Start at first entry in read-object log
2520
Check object referred-to by entry
2525
Does object have read bit set?
No
Yes
Set bit
2530
2540
Remove entry
2545
Do additional entries exist?
Yes
No
End

Fig. 26


Computing Environment 2600
2630
Communication Connection(s) 2670
Input Device(s) 2650
Output Device(s) 2660
Storage 2640
Memory 2620
Processing Unit 2610
Software 2680 Implementing Transactional Memory Techniques