

100

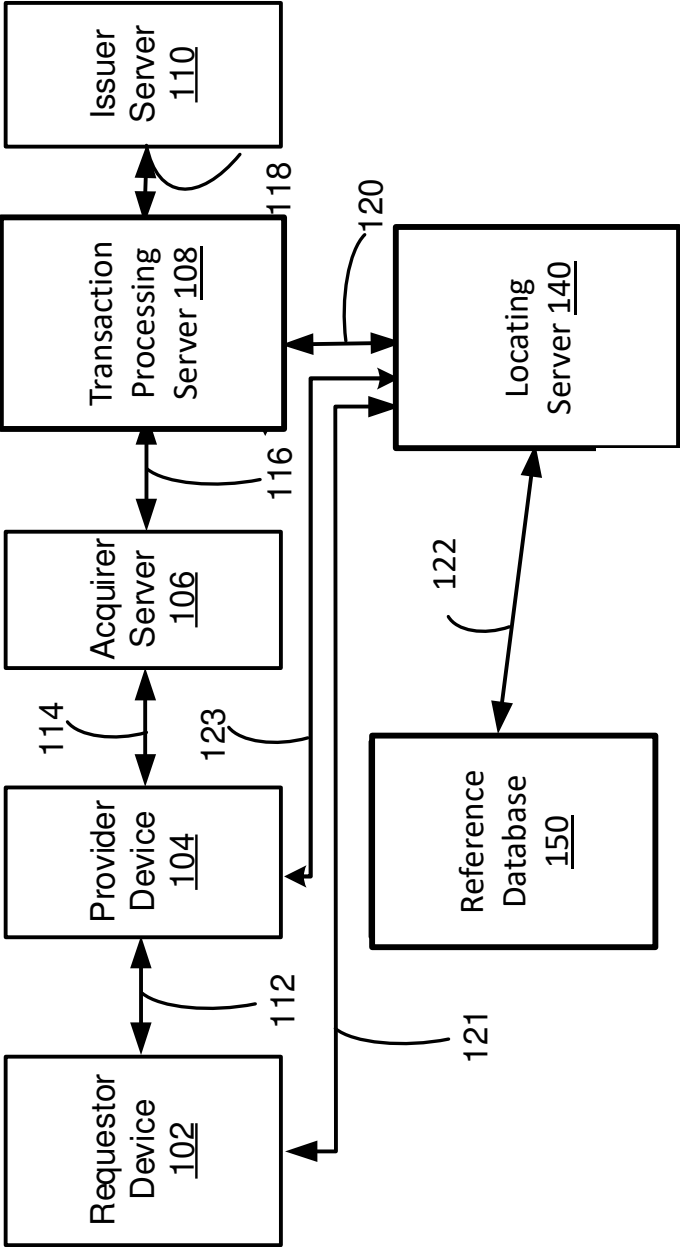


Fig. 1

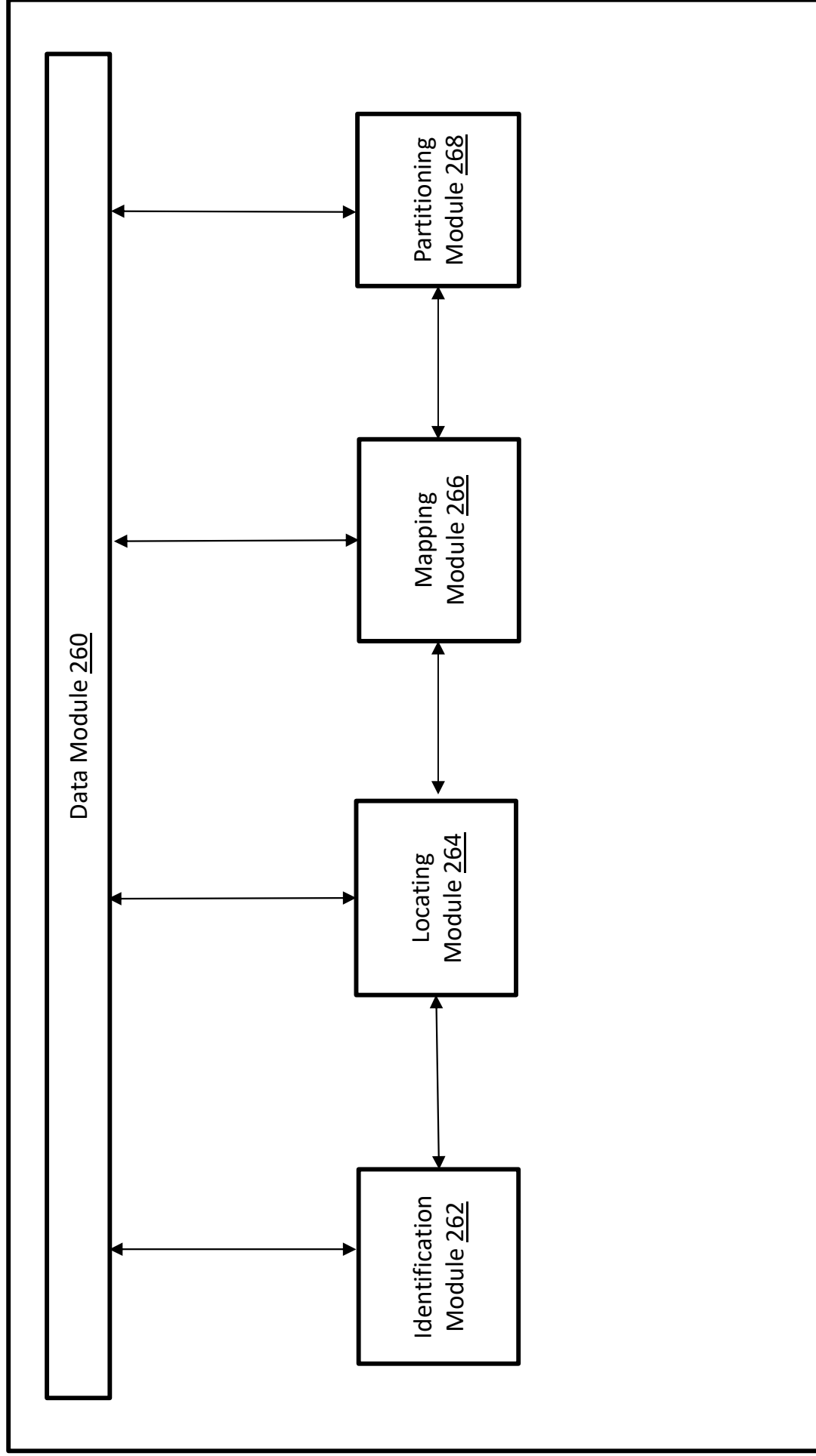


FIG. 2

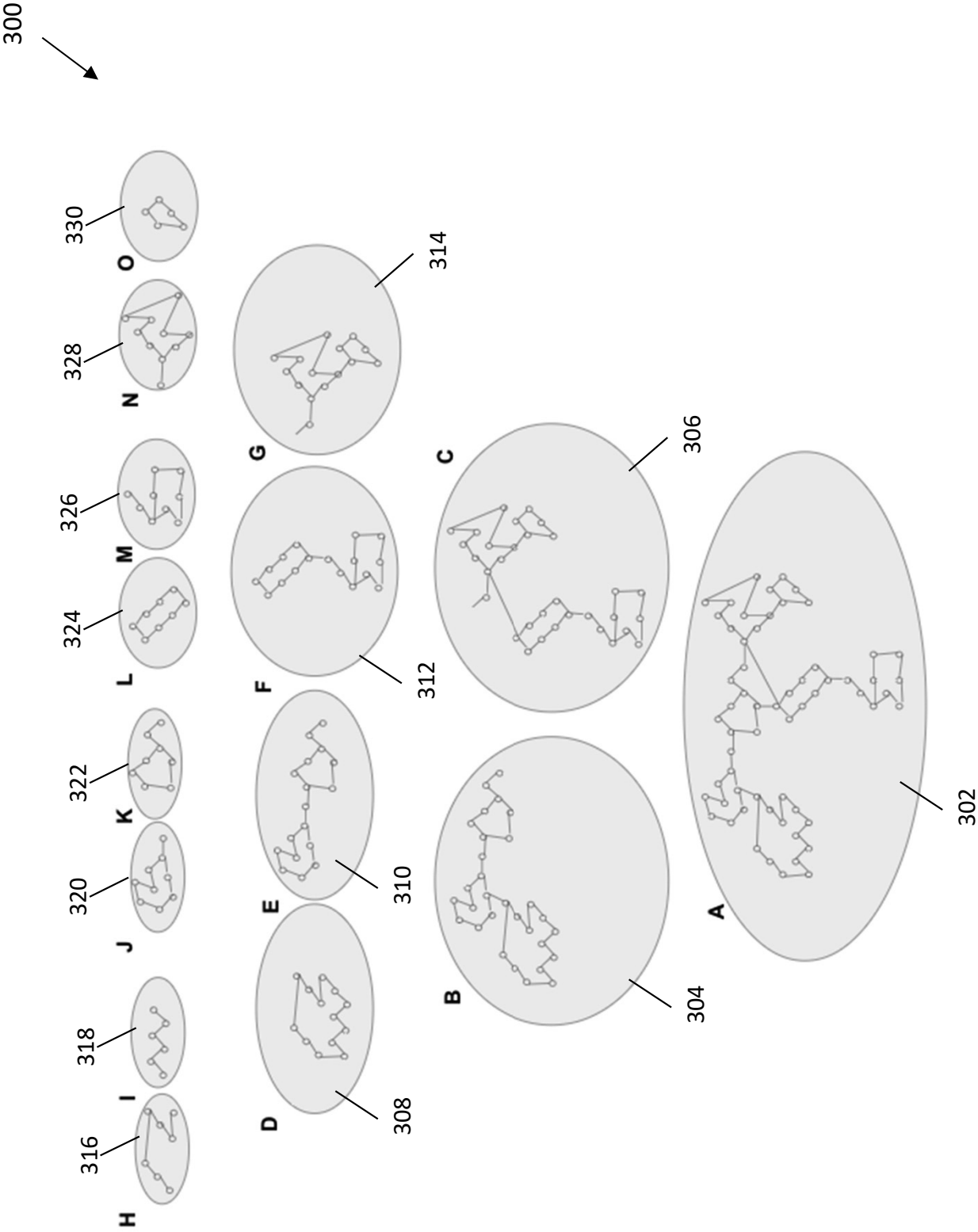


Figure 3

400

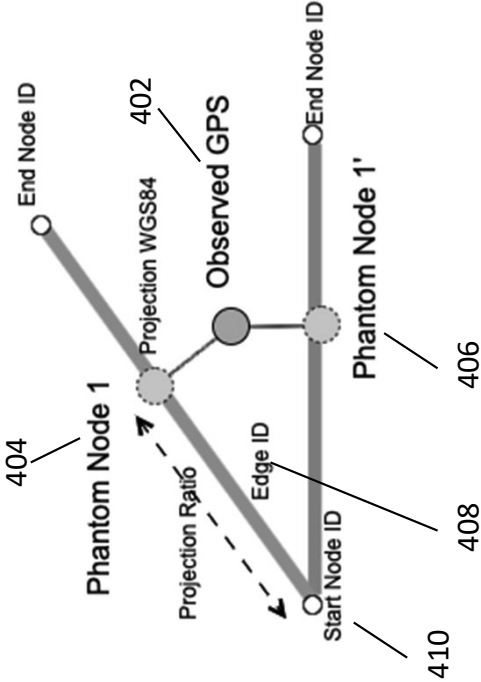


Figure 4A

412



Driver storage

```
{
    driver1:[Node1ID, Partition1ID, Partition3ID],
    driver2:[Node1ID, Partition1ID, Partition3ID],
    driver3:[Node2ID, Partition2ID, Partition3ID],
    driver4:[Node2ID, Partition2ID, Partition3ID]
}
```

Figure 4B

414



416

Node Storage

NodeID will be mapped to a list of drivers on that *NodeID*

```
{  
    Node1ID: [driver1, driver2],  
    Node2ID: [driver3, driver4],  
}
```

418

Partition Storage

PartitionID will be mapped to the number of driver on that *PartitionID*

```
{  
    Partition1ID: 2,  
    Partition2ID: 2,  
    Partition3ID: 4  
}
```

Figure 4C

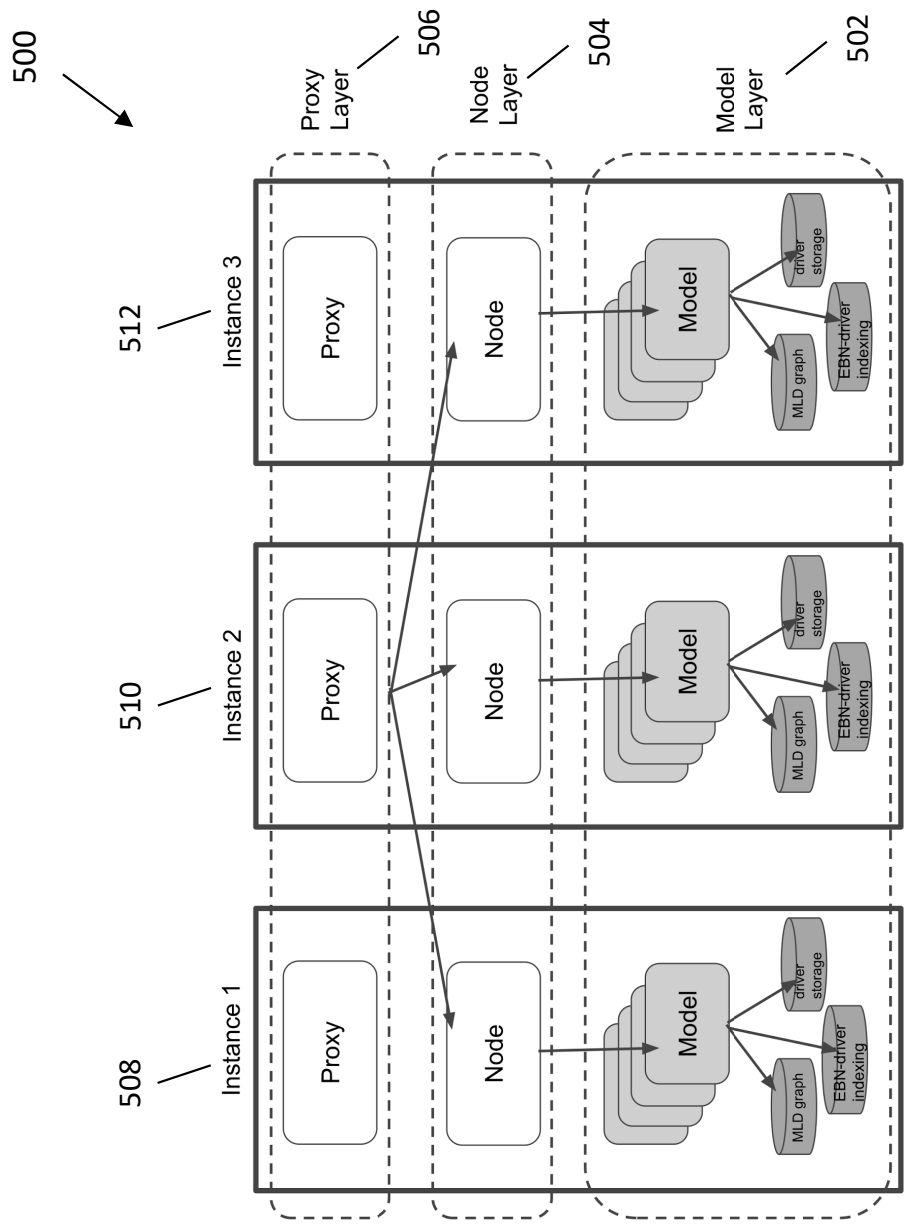


Figure 5

600

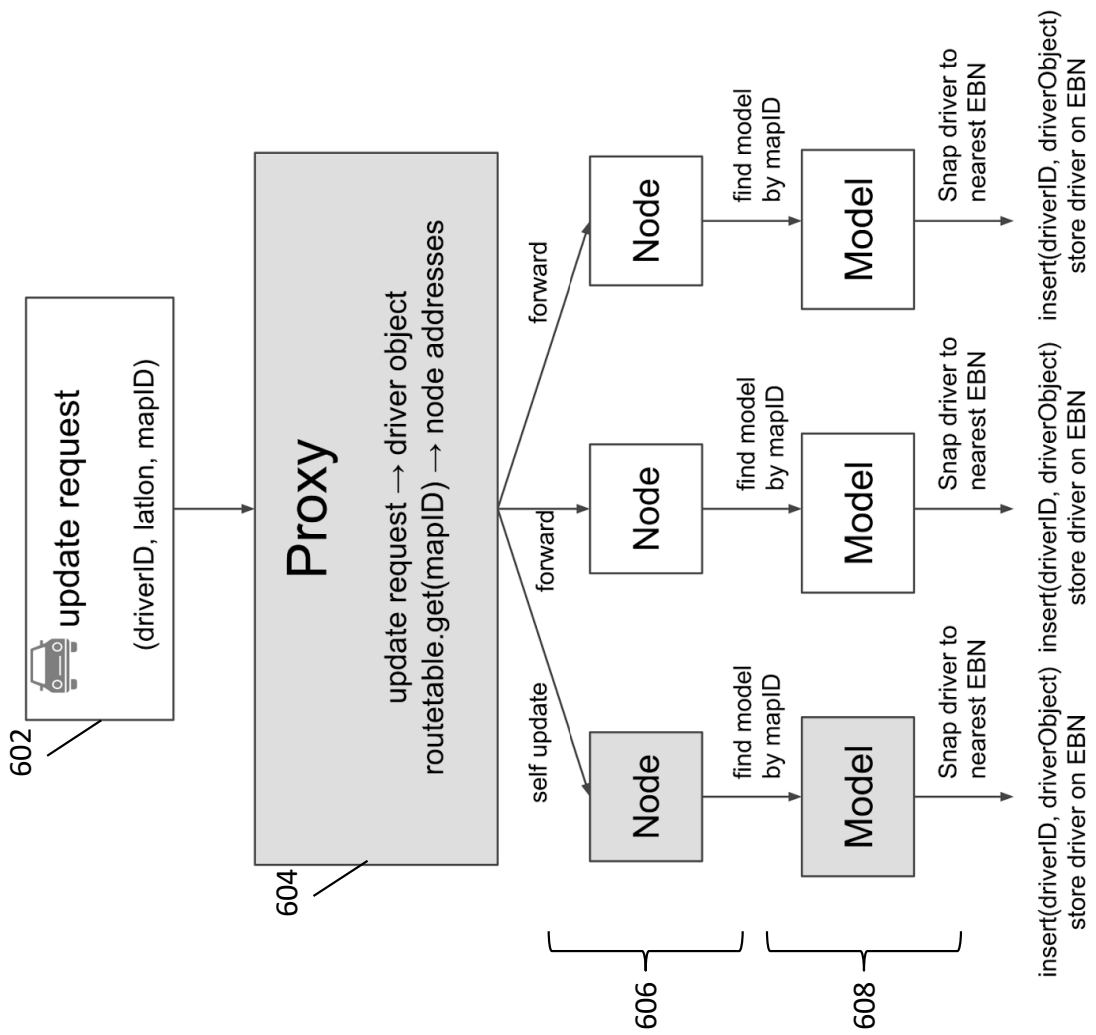


Figure 6A



Algorithm 1: Driver Update

Input: *driverObjects*, *driverStorage*, *associationDirectory*

Output: nil

```
1 for each driverObject in driverObjects do
2   // set partitionID from each level to driverObject
3   for each level in [1..maxQueryLevel] do
4     driver.Position.PartitionIDMap[level] ←
5     getPartitionID(level, driverObject.Position.StartNodeID)
6   // set driver storage
7   driverStorage.Set(driverObject)
8   associationDirectory.SetNodeIDToDriverMapping(driverObject)
9   // set partitionID mapping in Association Directory
10  for each level in [1..maxQueryLevel] do
11    associationDirectory.SetPartitionIDToDriverMapping(driver.Position.PartitionIDMap[level])
```

Figure 6B

612

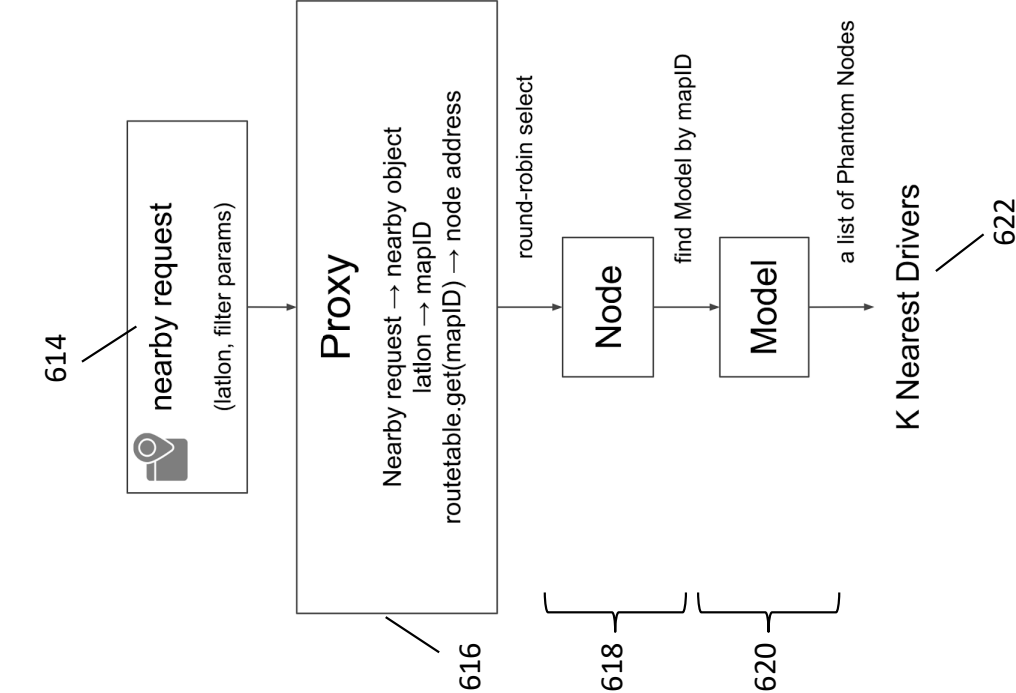


Figure 6C

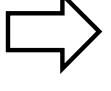
Algorithm 1: K Nearest Driver Search With Association Directory**Input:** *phantomNodes*, *filterParams*, *driverLimit*, *driverStorageSnapshot* — 626**Output:** nearest K drivers by routing distance

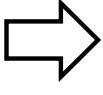
```

// initialization
1 driverPQ.reset(); — 628 — 630
2 EBNPQ.reset()
3 result ← empty driver array
4 for each pnode in phantomNodes do
    // insert drivers on starting EBNs to driverPQ
    5 for each driver on pnode.EBN do — 632
    6     driverPQ ← driver
    // insert adjacent EBNs to EBNPQ
    7 adjacentEBNs ← getAdjacentEBNs(pnode.EBN) — 634
    8 for each node in adjacentEBNs do
    9     EBNPQ ← node
10 while EBNPQ is not empty do
11     minEBN ← EBNPQ.delMin()
12     minEBN.visited ← true
13     for each driver on minEBN do
14         driverPQ ← driver
15     while driverPQ is not empty do
16         driver ← driverPQ.delMin()
17         passAllFilters ← true
18         for each filter in filterParams do
19             if driver does not pass filter then
20                 passAllFilters ← false
21                 break

```

Continued to Fig. 6E

**Figure 6D**



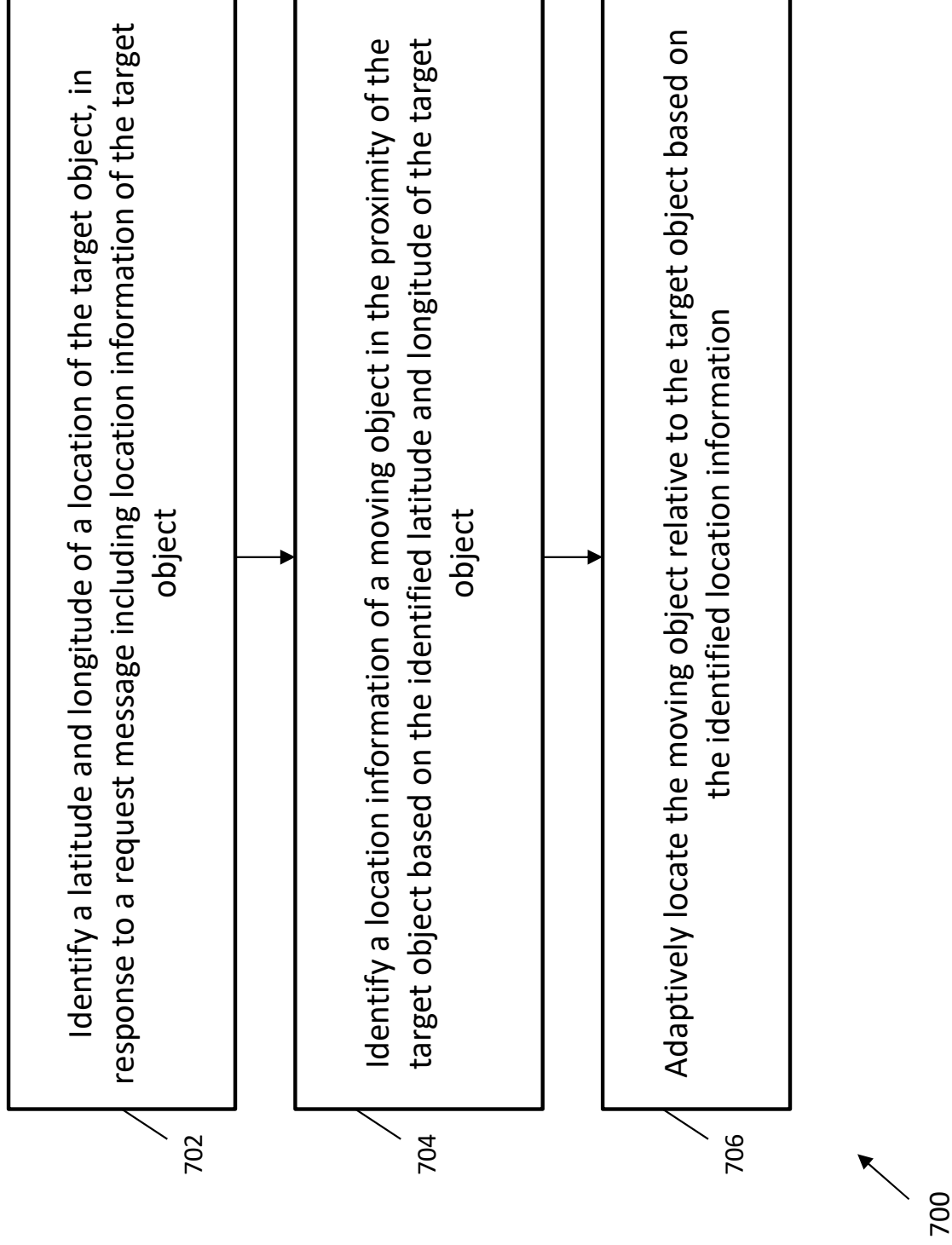
Continued from Fig. 6D

```

22 // only append driver to result if it passes all filters
23 if passAllFilters then
24 | result  $\leftarrow$  append(result, driver)
25 else
26 | continue
27 | return directly when driver size reaches limit } 636
28 | if len(result)  $\geq$  driverLimit then
29 | | return result
30
31 partitionLevel  $\leftarrow$  getTheHighestPartitionLevelWithDriver(minEBN, driverStorageSnapshot)
32 if partitionLevel  $\geq$  0 then
33 | // expand into border nodes at the partitionLevel
34 | adjacentBorderNodes  $\leftarrow$  getAdjacentBorderNodes(minEBN, partitionLevel)
35 | for each node in adjacentBorderNodes do
36 | | if node.visited || outOfRange (node) then
37 | | | continue
38 | | EBNPQ  $\leftarrow$  node
39
40 | // expand into adjacent EBNS
41 | adjacentEBNs  $\leftarrow$  getAdjacentEBNs(minEBN, partitionLevel)
42 | for each node in adjacentEBNs do
43 | | if node.visited || outOfRange (node) then
44 | | | continue
45 | | EBNPQ  $\leftarrow$  node
46
47 return result

```

Figure 6E

**Figure 7**

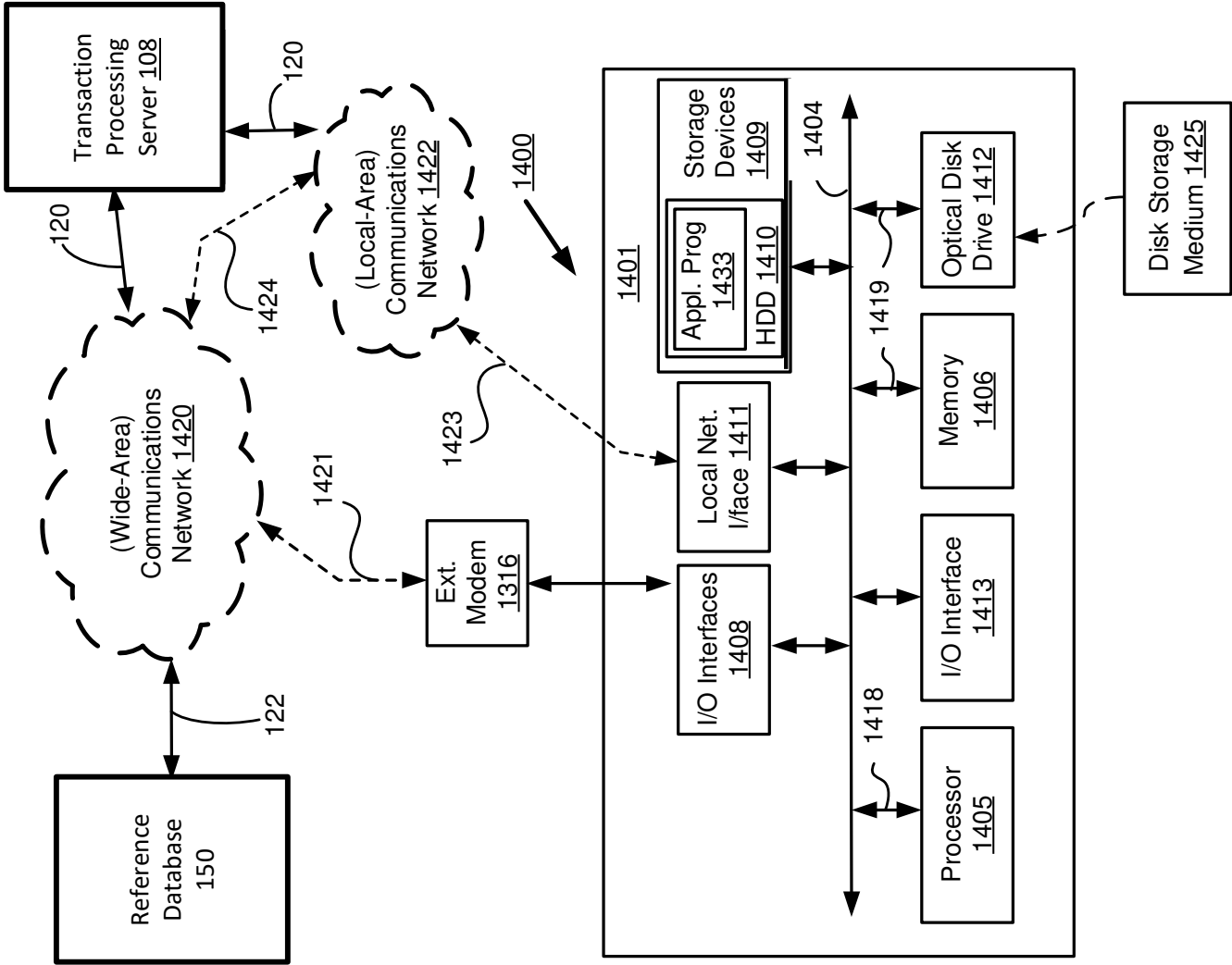


Figure 8A

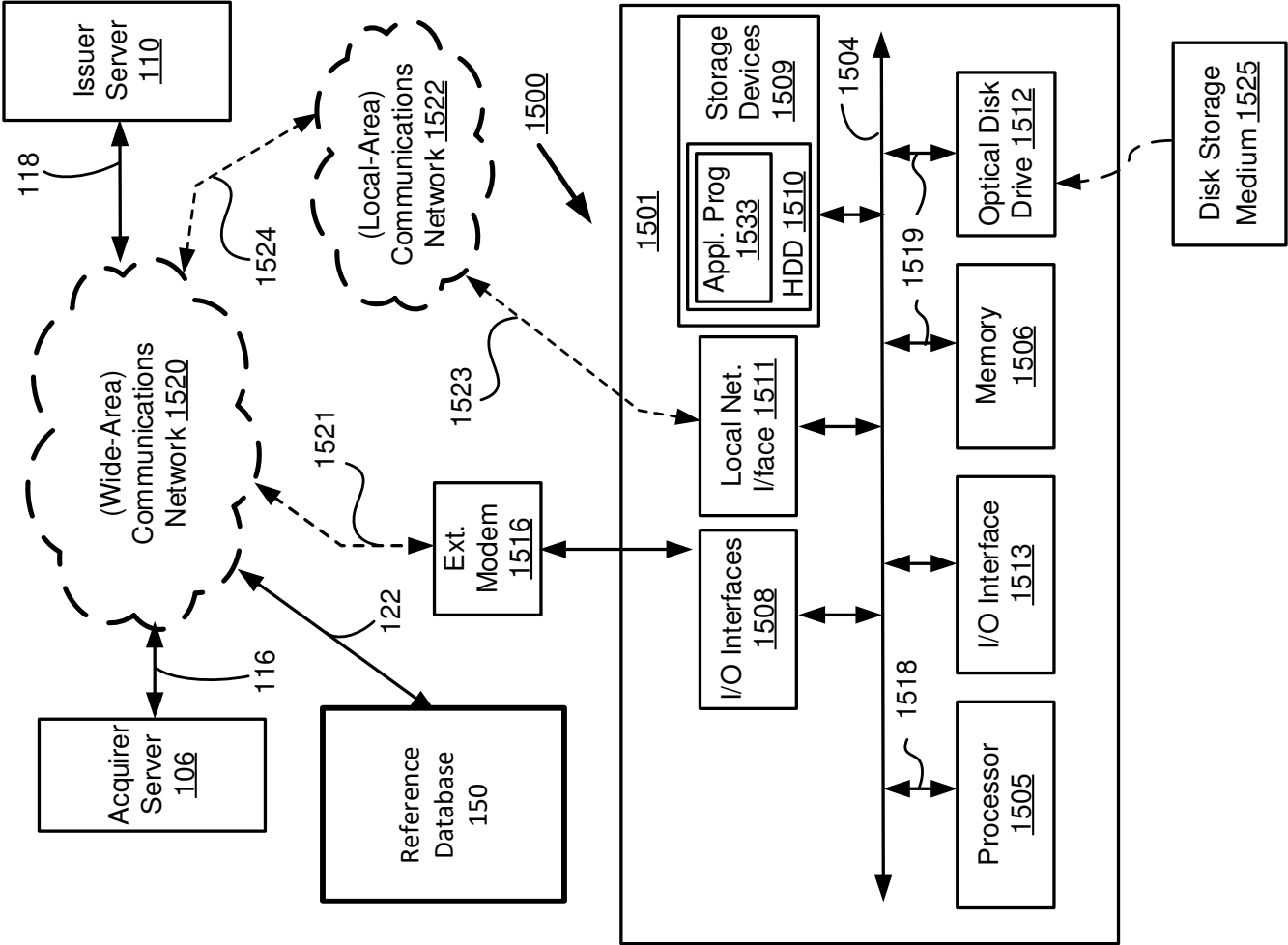


Figure 8B

108

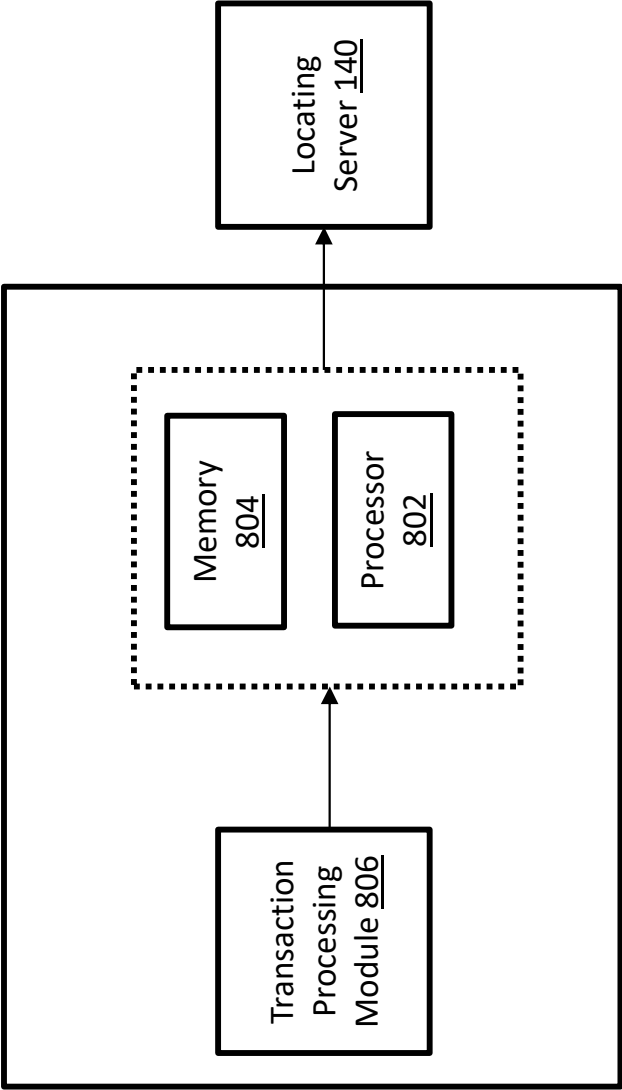


Figure. 9

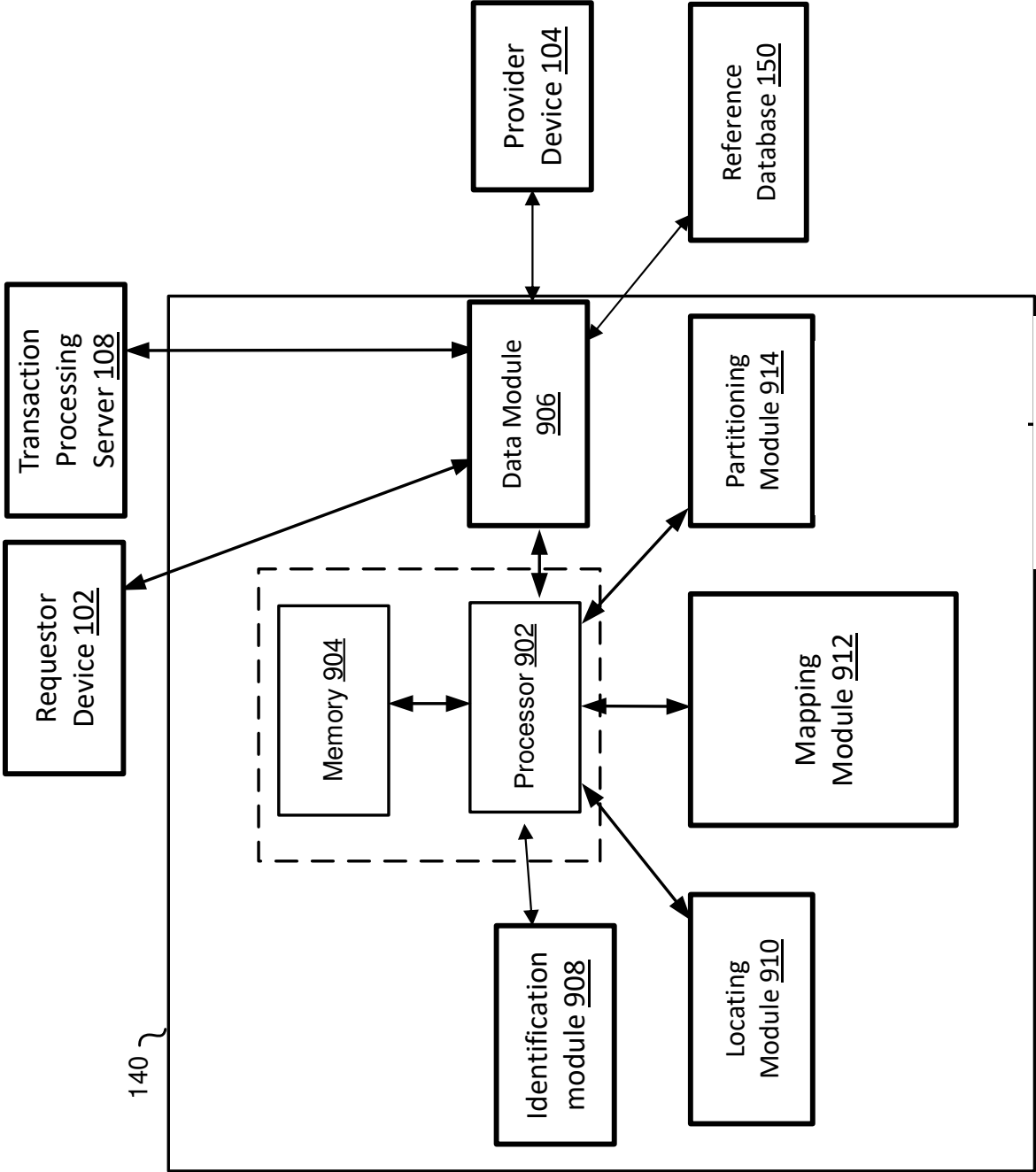


Figure 10

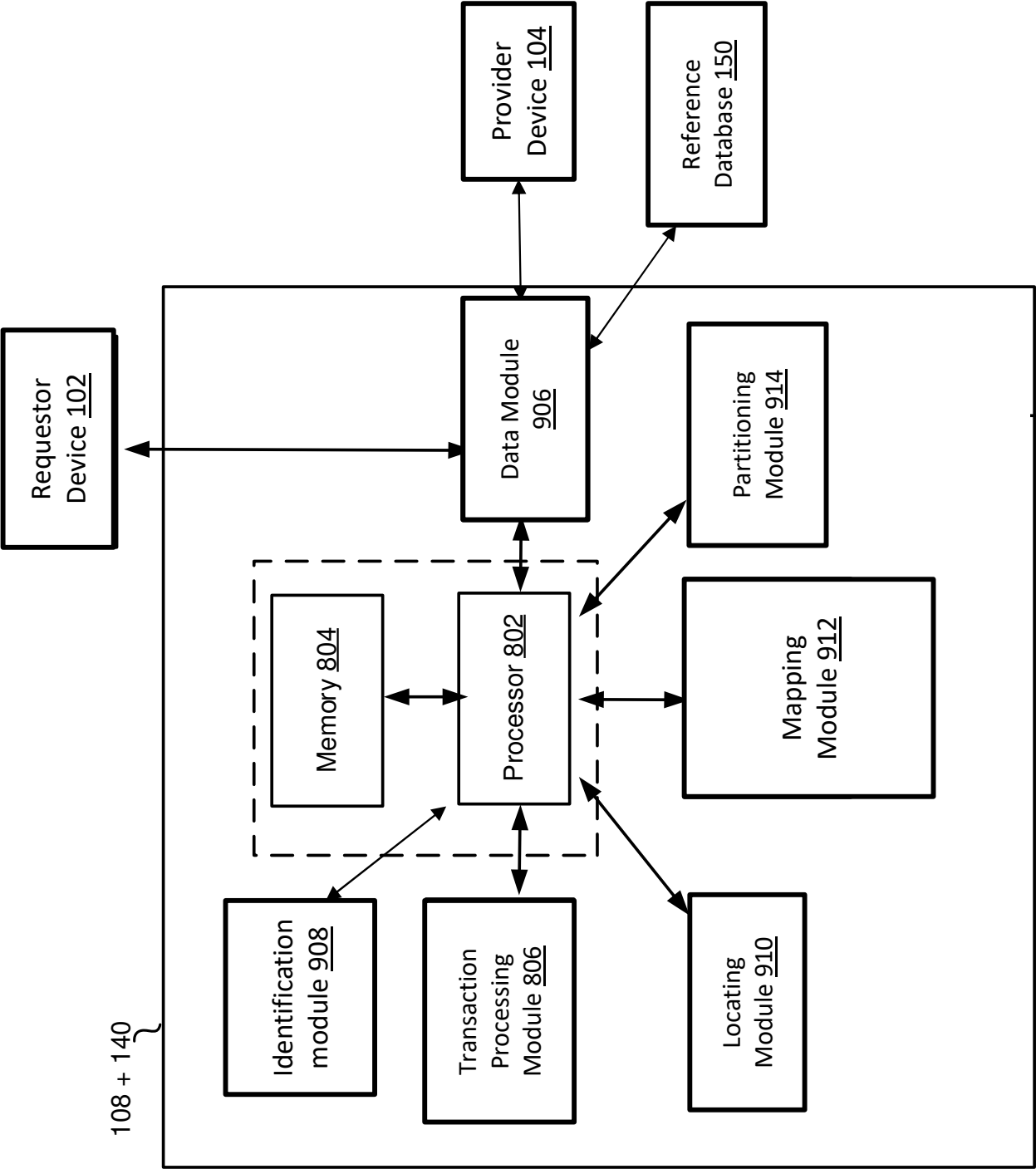


Figure 11