



(51) International Patent Classification:

G06F 9/52 (2006.01)

G06F 12/02 (2006.01)

(21) International Application Number:

PCT/EP2019/058668

(22) International Filing Date:

05 April 2019 (05.04.2019)

(25) Filing Language:

English

(26) Publication Language:

English

(71) Applicant: HUAWEI TECHNOLOGIES CO., LTD.

[CN/CN]; Huawei Administration Building Bantian Longgang District, Shenzhen, Guangdong 518129 (CN).

(72) Inventor; and

(71) Applicant (for US only): ALVAREZ, Victor [DE/DE];

Huawei Technologies Duesseldorf GmbH Riesstr. 25, 80992 Munich (DE).

(74) Agent: KREUZ, Georg; Huawei Technologies Duesseldorf GmbH, Riesstr. 25, 80992 Munich (DE).

(81) Designated States (unless otherwise indicated, for every

kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP,

KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH,

GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— with international search report (Art. 21(3))

(54) Title: COMPUTER-IMPLEMENTED LOCK-FREE DATA STRUCTURE AND RELATED METHODS AND APPARATUSES

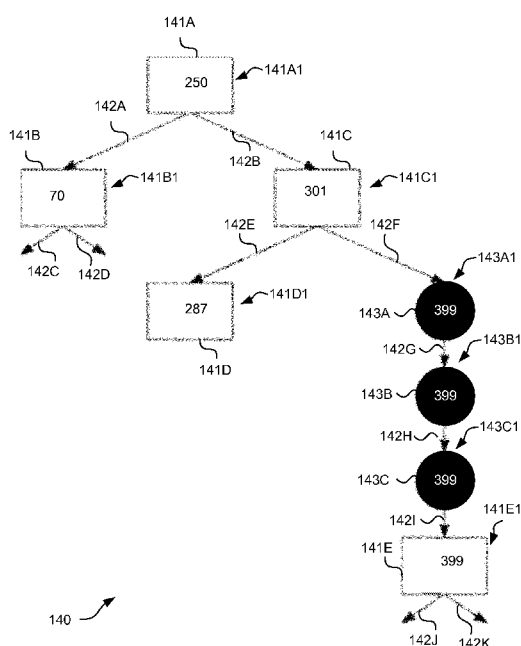


FIG. 1C

(57) Abstract: A computer-implemented lock-free data structure and related methods and apparatuses are disclosed. The data structure comprises at least one data type and a memory pointer per each data type, each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type. The presence of a dummy data type indicates that a native data type first following the dummy data type is protected against writes.



COMPUTER-IMPLEMENTED LOCK-FREE DATA STRUCTURE AND RELATED METHODS AND APPARATUSES

TECHNICAL FIELD

- 5 The present disclosure relates to the field of data processing, and more particularly to a computer-implemented lock-free data structure and related methods and apparatuses.

BACKGROUND

- 10 In computer science, a data structure is a format for organizing, managing and storing data. A data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways.

- At least for some data structures, strong consistency reads and safe memory reclamation may be desirable. In strong consistency reads, all reads see the latest
15 previously written values. Memory-constrained concurrent data structures are an example of a data structure that may require safe memory reclamation.

Strong consistency reads and safe memory reclamation have been traditionally provided e.g. via techniques including reference counting, latches or spinlocks, hazard pointers, and copy-read-update (RCU).

- 20 However, reference counting may produce a memory overhead on each data type or node found in a data structure for realizing a counter. E.g. on highly memory-constrained data structures this may not be desirable. Latches or spinlocks may be realized with a single bit of space per data type found in the data structure but they may strongly limit scalability of the data structure due to false sharing between threads.
25 Hazard pointers and RCUs may require a whole infrastructure to be realized, and thus they may be very intricate and therefore difficult to implement.

SUMMARY

- 30 This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the detailed description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

It is an object of the invention to provide an improved lock-free (i.e. non-blocking) data structure with strong consistency reads and safe memory reclamation, as well as related methods and apparatuses. The foregoing and other objects are achieved by the features of the independent claims. Further implementation forms are apparent
5 from the dependent claims, the description and the figures.

According to a first aspect, a computer-implemented lock-free data structure is provided. The computer-implemented lock-free data structure comprises at least one data type. The computer-implemented lock-free data structure further comprises a memory pointer per each of the at least one data type, each memory pointer pointing
10 to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following the dummy data type is protected against writes. The invention allows an improved lock-free data structure
15 with strong consistency reads and safe memory reclamation. Using tagged memory pointers allows using memory that has already been allocated in the data structure, and it allows avoiding introduction of a new data type to the structure. The space required by the dummy data type(s) is not persistent throughout the life of the data structure, and thus the invention can be used where reference counting is not possible.
20 Furthermore, the space required to realize the dummy data type(s) is a function of concurrency (number of thread/clients accessing the data structure) and not a function of the number of data elements/types of the data structure. Thus, the invention allows significant space savings in the data structure. Furthermore, the dummy data type(s) may be placed and removed in a lock-free manner with a low-precision (e.g. 8-byte
25 long) atomic operator compare-and-swap (CAS) currently supported by most processors. Also, (dis)placement of the dummy data type(s) does not affect scalability of the data structure. Furthermore, the dummy data type(s) are simple to program. For example, the dummy data type(s) require little intrusive changes to existing code to integrate them. If the data structure previously used e.g. reference counters, latches,
30 or spinlocks, the space required by them is now free for other purposes – or the overall memory footprint of the data structure can be made smaller.

In an implementation form of the first aspect, the computer-implemented lock-free data structure further comprises at least two dummy data types before a native data type,

each of the at least two dummy data types protecting the same native data type against writes and each of the at least two dummy data types inserted by a different party.

In an implementation form of the first aspect, the computer-implemented lock-free data structure further comprises a single dummy data type before a native data type
5 protecting the native data type against writes, the single dummy data type comprising a counter value indicating the number of different parties accessing the native data type.

In an implementation form of the first aspect, at least one native data type comprises a key and each dummy data type protecting it against writes comprises a copy of the
10 key or a pointer to the key.

In an implementation form of the first aspect, each party comprises a thread or a client.

In an implementation form of the first aspect, the computer-implemented lock-free data structure comprises one of a linked list and a binary search tree.

According to a second aspect, a method of performing a search operation on a
15 computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free
20 data structure indicates that a native data type first following said dummy data type is protected against writes is provided. The method comprises: starting, by a processor, a search for an object in the computer-implemented lock-free data structure. In response to finding the object with no dummy data type immediately preceding it, inserting, by the processor, a predetermined first positive number of linked dummy
25 data types immediately before the found data type; or in response to finding the object with at least one dummy data type immediately preceding it, inserting, by the processor, the first predetermined positive number of linked dummy data types immediately before the first of the at least one dummy data type preceding the found data type. In response to a successful insertion of the first positive number of linked
30 dummy data types, performing, by the processor, at least one read operation on the found object directly or via a callback function.

According to an implementation form of the third aspect, the method of performing a search operation further comprises deleting, by the processor, the successfully inserted first predetermined positive number of linked dummy data types upon exiting said at least one read operation.

- 5 According to an implementation form of the third aspect, each dummy data type comprises a counter value indicating the number of different parties accessing its protected native data type, and the method further comprises in response to finding the object with a dummy data type immediately preceding it, incrementing, by the processor, the counter value of the immediately preceding dummy data type;
10 decrementing, by the processor, the counter value upon exiting the at least one read operation; and deleting, by the processor, the successfully inserted dummy data type when its counter value falls below the predetermined first value. Preferably, before performing the search for an object, the counter value is initialized/set to a predetermined first value, greater or equal to one. Preferably, both incrementing and
15 decrementing of the counter value occurs in units of the initial predefined first value.

- According to a fourth aspect, a method of performing a delete operation on a computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag
20 indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following said dummy data type is protected against writes, the method is provided. The method comprises: performing, by a processor, a search for an object in the computer-implemented lock-free data
25 structure, and in response to finding the object with no dummy data type immediately preceding it, performing, by the processor, a lock-free delete; or in response to finding the object with at least one dummy data type immediately preceding it, ending, by the processor, the search.

- According to a fifth aspect, a method of performing an update operation on a
30 computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type,

wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following said dummy data type is protected against writes is provided. The method comprises: performing, by a processor, a search for an object in the computer-implemented lock-free data structure; and in response to finding the object with no dummy data type immediately preceding it, performing, by the processor, a lock-free update; or in response to finding the object with at least one dummy data type immediately preceding it, ending, by the processor, the search.

According to a sixth aspect, an apparatus is provided. The apparatus comprises at least one processor that is configured to implement a lock-free data structure, comprising at least one data type; and a memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following the dummy data type is protected against writes. The at least one processor is further configured to perform an access operation on the implemented lock-free data structure.

In an implementation form of the sixth aspect, the at least one processor is configured to perform the access operation by performing a search operation comprising: starting a search for an object in the implemented lock-free data structure; in response to finding the object with no dummy data type immediately preceding it, inserting a predetermined first positive number of linked dummy data types immediately before the found data type; or in response to finding the object with at least one dummy data type immediately preceding it, inserting the first positive number of linked dummy data types immediately before the first of the at least one dummy data type preceding the found data type; in response to a successful insertion of the first positive number of linked dummy data types, performing at least one read operation on the found object directly or via a callback function; and deleting the successfully inserted first positive number of linked dummy data types upon exiting the at least one read operation.

In an implementation form of the sixth aspect, each dummy data type comprises a counter value indicating the number of different parties accessing its protected native data type, and the at least one processor is configured to perform the access operation

by performing a search operation comprising: starting a search for an object in the implemented lock-free data structure; in response to finding the object with no dummy data type immediately preceding it, inserting a dummy data type immediately before the found data type and setting the counter value of the inserted dummy data type to a predetermined first value; or in response to finding the object with a dummy data type immediately preceding it, incrementing the counter value of the immediately preceding dummy data type by the predetermined first value; in response to a successful insertion of the dummy data type or incrementation of the counter value of the immediately preceding dummy data type, performing at least one read operation on the found object directly or via a callback function; decrementing the already set or incremented counter value by the predetermined first value upon exiting the at least one read operation; and deleting the successfully inserted dummy data type when its counter value falls below the predetermined first value.

In an implementation form of the sixth aspect, the at least one processor is further configured to perform the access operation by performing a delete operation comprising: performing a search for an object in the implemented lock-free data structure; and in response to finding the object with no dummy data type immediately preceding it, performing a lock-free delete; or in response to finding the object with at least one dummy data type immediately preceding it, ending the search.

In an implementation form of the sixth aspect, the at least one processor is further configured to perform the access operation by performing an update operation comprising: performing a search for an object in the implemented lock-free data structure; and in response to finding the object with no dummy data type immediately preceding it, performing a lock-free update; or in response to finding the object with at least one dummy data type immediately preceding it, ending the search.

Many of the attendant features will be more readily appreciated as they become better understood by reference to the following detailed description considered in connection with the accompanying drawings.

DESCRIPTION OF THE DRAWINGS

In the following, embodiments of the invention are described in more detail with reference to the attached figures and drawings, in which:

Fig. 1A is a diagram illustrating a computer-implemented lock-free data structure according to an embodiment;

Fig. 1B is a diagram illustrating a computer-implemented lock-free data structure according to another embodiment;

- 5 Fig. 1C is a diagram illustrating a computer-implemented lock-free data structure according to yet another embodiment;

Fig. 2 is a block diagram illustrating an apparatus according to an embodiment;

Fig. 3A is a flow diagram illustrating a method for performing a search operation on a computer-implemented lock-free data structure according to an embodiment;

- 10 Fig. 3B is a flow diagram illustrating a method for performing a search operation on a computer-implemented lock-free data structure according to another embodiment;

Fig. 3C is a flow diagram illustrating a method for performing a delete operation according to an embodiment;

- 15 Fig. 3D is a flow diagram illustrating a method of performing an update operation according to an embodiment;

Fig. 4A is a diagram illustrating inserting a dummy data type in a lock-free manner;

Fig. 4B is a diagram further illustrating inserting the dummy data type in the lock-free manner;

Fig. 4C is a diagram illustrating deleting a dummy data type in a lock-free manner; and

- 20 Fig. 4D is a diagram further illustrating deleting the dummy data type in the lock-free manner.

In the following, identical reference signs refer to identical or at least functionally equivalent features.

DETAILED DESCRIPTION

- 25 In the following description, reference is made to the accompanying drawings, which form part of the disclosure, and in which are shown, by way of illustration, specific

aspects in which the invention may be placed. It is understood that other aspects may be utilized and structural or logical changes may be made without departing from the scope of the invention. The following detailed description, therefore, is not to be taken in a limiting sense, as the scope of the invention is defined in the appended claims.

- 5 For instance, it is understood that a disclosure in connection with a described method may also hold true for a corresponding device or system configured to perform the method and vice versa. For example, if a specific method step is described, a corresponding device may include a unit to perform the described method step, even if such unit is not explicitly described or illustrated in the figures. On the other hand,
- 10 for example, if a specific apparatus is described based on functional units, a corresponding method may include a step performing the described functionality, even if such step is not explicitly described or illustrated in the figures. Further, it is understood that the features of the various example aspects described herein may be combined with each other, unless specifically noted otherwise.
- 15 The invention allows an improved lock-free data structure with strong consistency reads and safe memory reclamation. Using tagged memory pointers allows using memory that has already been allocated in the data structure, and it allows avoiding introduction of a new data type to the structure. The invention can be applied e.g. in data structures used in embedded software which may be constrained in resources
- 20 like memory. The invention may be implemented e.g. for lock-free pointer-based linked-lists and binary search trees.

In the following, example embodiments of a computer-implemented lock-free data structures 120, 130, 140 are described based on Figs. 1A to 1C. Some of the features of the described data structure are optional features which provide further advantages.

- 25 Fig. 1A is a diagram that illustrates an example of a computer-implemented lock-free data structure 120. Herein, "lock-free" is synonymous with "non-blocking". In the example of Fig. 1A, the computer-implemented lock-free data structure 120 comprises a linked list. A linked list is a linear collection of data elements/types, whose order is not given by their physical placement in memory. Instead, each element/type points to
- 30 the next list entry. Lock-free pointer-based linked-lists may be used to implement more complex lock-free data structures, such as hash tables via e.g. chained hashing and skip lists.

The computer-implemented lock-free data structure 120 comprises one or more data types. Herein, “data type” and “data element” are used interchangeably. Some of the data types may be native data types and some of them may be dummy data types. In the example of Fig. 1A, the computer-implemented lock-free data structure 120
5 comprises four native data types 121A-121D and three dummy data types 123A-123C (represented by black circles in Fig. 1A).

Herein, the term “dummy data type” (or “crumb”) is used to refer to a special kind of data type in the data structure the purpose of which is to signal that a thread/client is accessing a specific native data type of the data structure. In other words, a dummy
10 data type protects the other data types (i.e. “native” or “regular” data types) of the data structure against writes.

The computer-implemented lock-free data structure 120 further comprises a memory pointer 122A-122H per each of the at least one data type. Each memory pointer points to one of the at least one data type. Furthermore, each memory pointer comprises a
15 tag that indicates whether the pointed data type is a native data type 121A-121D or a dummy data type 123A-123C.

In the example of Fig. 1A, memory pointer 122A points to native data type 121A and comprises a tag that indicates that the pointed data type 121A is a native data type, memory pointer 122B points to native data type 121B and comprises a tag that
20 indicates that the pointed data type 121B is a native data type, memory pointer 122C points to dummy data type 123A and comprises a tag that indicates that the pointed data type 123A is a dummy data type, memory pointer 122D points to dummy data type 123B and comprises a tag that indicates that the pointed data type 123B is a dummy data type, memory pointer 122E points to dummy data type 123C and
25 comprises a tag that indicates that the pointed data type 123C is a dummy data type, memory pointer 122F points to native data type 121C and comprises a tag that indicates that the pointed data type 121C is a native data type, memory pointer 122G points to native data type 121D and comprises a tag that indicates that the pointed data type 121D is a native data type.

30 The presence of a dummy data type in the computer-implemented lock-free data structure 120 indicates that a native data type first following the dummy data type is protected against writes. In the example of Fig. 1A, the presence of the dummy data

type 123C in the computer-implemented lock-free data structure 120 indicates that the native data type 121C first following the dummy data type 123C is protected against writes.

In other words, the dummy data types are implemented by using tagged pointers, i.e. native computer (memory) pointers with a tag. This tag signals whether the data type pointed to by the tagged pointer is a data type containing data of the data structure, or whether it is a dummy data type. Advantages of using tagged pointers include e.g. the following: it allows using memory that has already been allocated in the data structure, and it allows avoiding introduction of a new data type to the structure.

- 5
- 10 In an embodiment, the computer-implemented lock-free data structure 120 may further comprise two or more dummy data types before a native data type. In such a case, each of the two or more dummy data types has been inserted by a different party and each of the two or more dummy data types protects the same native data type against writes. Herein, a party may comprise e.g. a thread or a client. In the example of Fig.
- 15 1A, the computer-implemented lock-free data structure 120 comprises three dummy data types 123A-123C before the native data type 121C, such that dummy data type 123A has been inserted by a first thread/client, dummy data type 123B has been inserted by a second thread/client, and dummy data type 123C has been inserted by a third thread/client. All three dummy data types 123A-123C protect the same native
- 20 data type 121C against writes.

- In other words, when accessing native data types (e.g. in a lock-free linked list or binary search tree), every accessing thread/client places a dummy data type before (in the native order of the data structure in question) the accessed native data type to protect it against writes, thus providing strong consistency reads and safe memory
- 25 reclamation. Thus, a chain of consecutive dummy data types indicates that more than one thread/client is accessing the protected data type of the data structure in question. When needed, this chain of dummy data types can be made longer by adding a new dummy data type before the first encountered dummy data type having a key matching the key of the protected native data type.

- 30 At least one native data type may comprise a key. In the example of Fig. 1A, native data type 121A comprises key 121A1 ("15"), native data type 121B comprises key 121B1 ("37"), native data type 121C comprises key 121C1 ("101"), and native data

type 121D comprises key 121D1 ("157"). In an embodiment, each dummy data type protecting a native data type against writes may comprise a copy of the key of the protected native data type or a pointer to the key of the protected native data type. In the example of Fig. 1A, native data type 121C comprising key 121C1 ("101") is the
5 protected one. Thus, dummy data type 123A comprises key copy 123A1 ("101"), dummy data type 123B comprises key copy 123B1 ("101"), and dummy data type 123C comprises key copy 123C1 ("101").

In other words, in the example of Fig. 1A, three threads/clients are currently reading from native data type 121C whose key is "101" and thus native data type 121C is
10 protected against writes. Dummy data types 123A-123C have the same key as the native data type 121C they protect (in this case "101"). Other native data types 121A, 121B, 121D (with keys "15", "37", and "157", respectively) are unprotected against writes.

In an embodiment, insertions of native data types in lock-free (concurrent) linked lists
15 and binary search trees may be performed without breaking chains of consecutive dummy data types and without splitting an object from its protecting chain of dummy data types (if any). That is, for insertions, a chain of dummy data types and its protected object may appear as a single entity.

Fig. 1B is a diagram that illustrates an example of a computer-implemented lock-free
20 data structure 130. In the example of Fig. 1B, the computer-implemented lock-free data structure 130 comprises a linked list.

The computer-implemented lock-free data structure 130 comprises one or more data types. Some of the data types may be native data types and some of them may be dummy data types. In the example of Fig. 1B, the computer-implemented lock-free
25 data structure 130 comprises four native data types 131A-131D and one dummy data type 133A (represented by a black circle in Fig. 1B).

The computer-implemented lock-free data structure 130 further comprises a memory pointer 132A-132F per each of the at least one data type. Each memory pointer points to one of the at least one data type. Furthermore, each memory pointer comprises a
30 tag that indicates whether the pointed data type is a native data type 131A-131D or a dummy data type 133A.

In the example of Fig. 1B, memory pointer 132A points to native data type 131A and comprises a tag that indicates that the pointed data type 131A is a native data type, memory pointer 132B points to native data type 131B and comprises a tag that indicates that the pointed data type 131B is a native data type, memory pointer 132C
5 points to dummy data type 133A and comprises a tag that indicates that the pointed data type 133A is a dummy data type, memory pointer 132D points to native data type 131C and comprises a tag that indicates that the pointed data type 131C is a native data type, and memory pointer 132E points to native data type 131D and comprises a tag that indicates that the pointed data type 131D is a native data type.

10 Again, the presence of a dummy data type in the computer-implemented lock-free data structure 130 indicates that a native data type first following the dummy data type is protected against writes. In the example of Fig. 1B, the presence of the dummy data type 133A in the computer-implemented lock-free data structure 130 indicates that the native data type 131C first following the dummy data type 133A is protected against
15 writes.

Rather than the multiple consecutive dummy data type employed in the embodiment of Fig. 1A, in the embodiment of Fig. 1B the computer-implemented lock-free data structure 130 may comprise a single dummy data type before a native data type protecting the native data type against writes, such that the single dummy data type
20 comprises a counter value indicating the number of different parties accessing the native data type. In the example of Fig. 1B, the computer-implemented lock-free data structure 130 comprises the single dummy data type 133A before the native data type 131C protecting the native data type 131C against writes. The single dummy data type 133A comprises a counter value 133A2 indicating the number of different parties
25 accessing the native data type 131C. In the example of Fig. 1B, the counter value 133A2 is "3" indicating that three different parties are currently accessing the native data type 131C.

In other words, the embodiment of Fig. 1B implements "pluggable" reference counters. That is, a single dummy data type large enough to contain a counter may be placed
30 before (in the native order of the data structure 130) the data type to be protected. The dummy data type can then be used as a reference counter, but when not needed it may be removed from the data structure 130.

At least one native data type may comprise a key. In the example of Fig. 1B, native data type 131A comprises key 131A1 ("15"), native data type 131B comprises key 131B1 ("37"), native data type 131C comprises key 131C1 ("101"), and native data type 131D comprises key 131D1 ("157"). In an embodiment, each dummy data type
 5 protecting a native data type against writes may comprise a copy of the key of the protected native data type or a pointer to the key of the protected native data type. In the example of Fig. 1B, native data type 131C comprising key 131C1 ("101") is the protected one. Thus, dummy data type 133A comprises key copy 133A1 ("101").

Further features of the computer-implemented lock-free data structure 130 of Fig. 1B
 10 directly result from the functionalities and parameters of the computer-implemented lock-free data structure 120 of Fig. 1A and thus are not repeated here.

Fig. 1C is a diagram that illustrates an example of a computer-implemented lock-free data structure 140. In the example of Fig. 1C, the computer-implemented lock-free data structure 140 comprises a binary search tree (also known as an ordered binary
 15 tree or a sorted binary tree). A binary search tree keeps its keys in a sorted order so that lookup and other operations can use the principle of binary search: when looking for a key in a tree (or a place to insert a new key), they traverse the tree from root to leaf, making comparisons to keys stored in the nodes of the tree and deciding, on the basis of the comparison, to continue searching in the left or right subtrees.

20 The computer-implemented lock-free data structure 140 comprises one or more data types. Some of the data types may be native data types and some of them may be dummy data types. In the example of Fig. 1C, the computer-implemented lock-free data structure 140 comprises five native data types 141A-141E and three dummy data types 143A-143C (represented by black circles in Fig. 1C).

25 The computer-implemented lock-free data structure 140 further comprises a memory pointer 142A-142K per each of the at least one data type. Each memory pointer points to one of the at least one data type. Furthermore, each memory pointer comprises a tag that indicates whether the pointed data type is a native data type 141A-141E or a dummy data type 143A-143C.

30 In the example of Fig. 1C, memory pointer 142A points to native data type 141B and comprises a tag that indicates that the pointed data type 141B is a native data type, memory pointer 142B points to native data type 141C and comprises a tag that

indicates that the pointed data type 141C is a native data type, memory pointer 142E points to native data type 141D and comprises a tag that indicates that the pointed data type 141D is a native data type, memory pointer 142F points to dummy data type 143A and comprises a tag that indicates that the pointed data type 143A is a dummy data type, memory pointer 142G points to dummy data type 143B and comprises a tag that indicates that the pointed data type 143B is a dummy data type, memory pointer 142H points to dummy data type 143C and comprises a tag that indicates that the pointed data type 143C is a dummy data type, and memory pointer 142I points to native data type 141E and comprises a tag that indicates that the pointed data type 141E is a native data type.

The presence of a dummy data type in the computer-implemented lock-free data structure 140 indicates that a native data type first following the dummy data type is protected against writes. In the example of Fig. 1C, the presence of the dummy data type 143C in the computer-implemented lock-free data structure 140 indicates that the native data type 141E first following the dummy data type 143C is protected against writes.

In an embodiment, the computer-implemented lock-free data structure 140 may further comprise two or more dummy data types before a native data type. In such a case, each of the two or more dummy data types has been inserted by a different party and each of the two or more dummy data types protects the same native data type against writes. Herein, a party may comprise e.g. a thread or a client. In the example of Fig. 1C, the computer-implemented lock-free data structure 140 comprises three dummy data types 143A-143C before the native data type 141E, such that dummy data type 143A has been inserted by a first thread/client, dummy data type 143B has been inserted by a second thread/client, and dummy data type 143C has been inserted by a third thread/client. All three dummy data types 143A-143C protect the same native data type 141E against writes.

In other words, the native data type 141E with key "399" is being protected by three dummy data types 143A-143C which means that three threads/clients are currently accessing the native data type 141E and thus can neither be overwritten nor deleted. Rebalancing operations (rotations) of the binary search tree 140 are not affected by dummy data types, since the chains of dummy data types along with their protected native data type(s) are not to be broken and instead move as a whole.

At least one native data type may comprise a key. In the example of Fig. 1C, native data type 141A comprises key 141A1 ("250"), native data type 141B comprises key 141B1 ("70"), native data type 141C comprises key 141C1 ("301"), native data type 141D comprises key 141D1 ("287"), and native data type 141E comprises key 141E1 ("399"). In an embodiment, each dummy data type protecting a native data type against writes may comprise a copy of the key of the protected native data type or a pointer to the key of the protected native data type. In the example of Fig. 1C, native data type 141E comprising key 141E1 ("399") is the protected one. Thus, dummy data type 143A comprises key copy 143A1 ("399"), dummy data type 143B comprises key copy 143B1 ("399"), and dummy data type 143C comprises key copy 143C1 ("399").

In other words, in the example of Fig. 1C, three threads/clients are currently reading from native data type 141E whose key is "399" and thus native data type 141E is protected against writes. Dummy data types 143A-143C have the same key as the native data type 141E they protect (in this case "399"). Other native data types 141A, 141B, 141C, 141D (with keys "250", "70", "301", and "287", respectively) are unprotected against writes.

Further features of the computer-implemented lock-free data structure 140 of Fig. 1C directly result from the functionalities and parameters of the computer-implemented lock-free data structure 120 of Fig. 1A and thus are not repeated here.

In the following, example embodiments of an apparatus 200 are described based on Fig. 2. Some of the features of the described device are optional features which provide further advantages.

Fig. 2 is a block diagram that illustrates the apparatus 200. In an embodiment, the apparatus 200 may comprise a client device that may be any of various types of devices used directly by an end user entity and capable of communication in a wireless network, such as a user equipment (UE). Such devices include but are not limited to smartphones, tablet computers, smart watches, laptop computers, Internet-of-Things (IoT) devices, etc. In an embodiment, the apparatus 200 may comprise a device with embedded software.

The apparatus 200 comprises at least one processor or a processing unit 201 and optionally a memory 202 coupled to the at least one processor 201, which may be used to implement the functionalities described later in more detail.

The at least one processor 201 may include e.g. one or more of various processing devices, such as a co-processor, a microprocessor, a controller, a digital signal processor (DSP), a processing circuitry with or without an accompanying DSP, or various other processing devices including integrated circuits such as, for example, an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), a microcontroller unit (MCU), a hardware accelerator, a special-purpose computer chip, or the like.

The memory 202 may be configured to store e.g. computer programs and the like. The memory 202 may include one or more volatile memory devices, one or more non-volatile memory devices, and/or a combination of one or more volatile memory devices and non-volatile memory devices. For example, the memory 202 may be embodied as magnetic storage devices (such as hard disk drives, floppy disks, magnetic tapes, etc.), optical magnetic storage devices, and semiconductor memories (such as mask ROM, PROM (programmable ROM), EPROM (erasable PROM), flash ROM, RAM (random access memory), etc.).

As a person skilled in the art can appreciate, when the apparatus 200 is configured to implement a functionality, a component or components of the apparatus 200, such as the at least one processor 201 and/or the memory 202, may be configured to implement this functionality. Furthermore, when the at least one processor 201 is configured to implement a functionality, this functionality may be implemented using program code comprised, for example, in the memory 202.

The at least one processor 201 is configured to implement a lock-free data structure 120, 130, 140 that comprises at least one data type 121A-121D, 123A-123C, 131A-131D, 133A, 141A-141E, 143A-143C; and a memory pointer 122A-122H, 132A-132F, 142A-142K per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type 121A-121D, 131A-131D, 141A-141E or a dummy data type 123A-123C, 133A, 143A-143C, wherein the presence of a dummy data type 123A-123C, 133A, 143A-143C in the computer-implemented lock-free data structure 120, 130, 140 indicates that a native data type 121A-121D, 131A-131D, 141A-141E first following the dummy data type 123A-123C, 133A, 143A-143C is protected against writes.

Further features of the computer-implemented lock-free data structure 120, 130, 140 implemented by the at least one processor 201 of the apparatus 200 directly result from the functionalities and parameters of the computer-implemented lock-free data structure 120, 130, 140 of Figs. 1A to 1C and thus are not repeated here.

- 5 The at least one processor 201 is further configured to perform an access operation on the implemented lock-free data structure 120, 130, 140.

In an embodiment, the at least one processor 201 may be configured to perform the access operation by performing a search operation, such that the search operation comprises:

- 10 starting a search for an object in the implemented lock-free data structure;
- in response to finding the object with no dummy data type immediately preceding it, inserting a predetermined first positive number of linked dummy data types immediately before the found data type; or
- 15 in response to finding the object with at least one dummy data type immediately preceding it, inserting the first positive number of linked dummy data types immediately before the first of the at least one dummy data type preceding the found data type;
- in response to a successful insertion of the first positive number of linked dummy
- 20 data types, performing at least one read operation on the found object directly or via a callback function; and
- deleting the successfully inserted first positive number of linked dummy data types upon exiting said at least one read operation.

- 25 In another embodiment, each dummy data type may comprise a counter value indicating the number of different parties accessing its protected native data type, and the at least one processor 201 may be configured to perform the access operation by performing a search operation, such that the search operation comprises:

- starting a search for an object in the implemented lock-free data structure;
- in response to finding the object with no dummy data type immediately preceding
- 30 it, inserting a dummy data type immediately before the found data type and setting the counter value of the inserted dummy data type to a predetermined first value; or

in response to finding the object with a dummy data type immediately preceding it, incrementing the counter value of the immediately preceding dummy data type by the predetermined first value;

in response to a successful insertion of the dummy data type or incrementation
5 of the counter value of the immediately preceding dummy data type, performing at least one read operation on the found object directly or via a callback function;

decrementing the already set or incremented counter value by the predetermined first value upon exiting said at least one read operation; and

deleting the successfully inserted dummy data type when its counter value falls
10 below the predetermined first value.

In an embodiment, the at least one processor 201 may be configured to perform the access operation by performing a delete operation, such that the delete operation comprises:

15 performing a search for an object in the implemented lock-free data structure; and

in response to finding the object with no dummy data type immediately preceding it, performing a lock-free delete; or

in response to finding the object with at least one dummy data type immediately
20 preceding it, ending the search.

In an embodiment, the at least one processor 201 may be configured to perform the access operation by performing an update operation, such that the update operation comprises:

performing a search for an object in the implemented lock-free data structure;
25 and

in response to finding the object with no dummy data type immediately preceding it, performing a lock-free update; or

in response to finding the object with at least one dummy data type immediately preceding it, ending the search.
30

Fig. 3A is a flow diagram illustrating a method 310 of performing a search operation according to an embodiment.

At step 311, a processor starts a search for an object in a computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each

of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that

5 a native data type first following the dummy data type is protected against writes.

At step 312, the processor allocates a dummy data type.

At step 313, the processor performs the search for the object.

If the object is not found, step 314, the processor deletes the unused allocated dummy data type, step 315. Then, the search ends, step 320.

10 If the object is found, step 314, and no dummy data type immediately precedes it the processor inserts a predetermined first positive number of linked dummy data types immediately before the found data type/object, step 316.

If the object is found, step 314, and at least one dummy data type immediately precedes it the processor inserts the first positive number of linked dummy data types

15 immediately before the first of the at least one dummy data type preceding the found data type/object, step 316.

If the insertion of the first positive number of linked dummy data types is not successful, step 317, the method returns to step 313.

If the insertion of the first positive number of linked dummy data types is successful,

20 step 317, the processor performs at least one read operation on the found object directly or via a callback function, step 318.

At step 319, the processor deletes the successfully inserted first positive number of linked dummy data types upon exiting the at least one read operation. Then, the search ends, step 320.

25 The method 310 may be performed by the apparatus 200. The steps 311-320 can, for example, be performed by the at least one processor 201 and the memory 202. Further features of the method 310 directly result from the functionalities of the the apparatus 200 and the computer-implemented lock-free data structure 120, 130, 140. The method 310 can be performed by a computer program product.

Fig. 3B is a flow diagram illustrating a method 330 of performing a search operation according to another embodiment.

At step 331, a processor starts a search for an object in a computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each
5 of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following the dummy data type is protected against writes, each
10 dummy data type comprising a counter value indicating the number of different parties accessing its protected native data type.

At step 332, the processor allocates a dummy data type and initializes its' counter value to A units where $A \geq 1$.

At step 333, the processor performs the search for the object.

15 If the object is not found, step 334, the processor deletes the allocated dummy data type if unused, step 335. Then, the search ends, step 336.

If the object is found, step 334, and no dummy data type immediately precedes it, the processor inserts a dummy data type immediately before the found data type/object and sets the counter value of the inserted dummy data type to a predetermined first
20 value (A units), step 337.

If the object is found, step 334, and a dummy data type immediately precedes it, the processor increments the counter value of the immediately preceding dummy data type by the predetermined first value (A units), step 337.

If the insertion of the dummy data type or incrementation of the counter value of the immediately preceding dummy data type is not successful, step 338, the method
25 returns to step 333.

If the insertion of the dummy data type or incrementation of the counter value of the immediately preceding dummy data type is successful, step 338, the processor performs at least one read operation on the found object directly or via a callback
30 function, step 339.

At step 340, the processor decrements the already set or incremented counter value by the predetermined first value (A units) upon exiting the at least one read operation.

At step 341, the processor determines whether the counter value falls below the predetermined first value (A units). If not, the method proceeds to step 335 described above. If yes, the processor deletes the successfully inserted dummy data type, step 342. Then, the method proceeds to step 335 described above.

The method 330 may be performed by the apparatus 200. The steps 331-342 can, for example, be performed by the at least one processor 201 and the memory 202. Further features of the method 330 directly result from the functionalities of the the apparatus 200 and the computer-implemented lock-free data structure 120, 130, 140. The method 330 can be performed by a computer program product.

Fig. 3C is a flow diagram illustrating a method 350 of performing a delete operation according to an embodiment.

The method starts at step 351.

At step 352, a processor starts a search for an object in a computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following the dummy data type is protected against writes.

If the object is found and at least one dummy data type immediately precedes it, step 353, the method proceeds to step 356 and ends.

If the object is found and no dummy data type immediately precedes it, step 353, the processor performs a lock-free delete, step 354.

If the delete is not successful, step 355, the method returns to step 352.

If the delete is successful, step 355, the method proceeds to step 356 and ends.

The method 350 may be performed by the apparatus 200. The steps 351-356 can, for example, be performed by the at least one processor 201 and the memory 202. Further features of the method 350 directly result from the functionalities of the the apparatus 200 and the computer-implemented lock-free data structure 120, 130, 140.

5 The method 350 can be performed by a computer program product.

Fig. 3D is a flow diagram illustrating a method 370 of performing an update operation according to an embodiment.

The method starts at step 371.

At step 372, a processor starts a search for an object in a computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each
10 of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that
15 a native data type first following the dummy data type is protected against writes.

If the object is found and at least one dummy data type immediately precedes it, step 373, the method proceeds to step 376 and ends.

If the object is found and no dummy data type immediately precedes it, step 373, the processor performs a lock-free update, step 374.

20 If the update is not successful, step 375, the method returns to step 372.

If the update is successful, step 375, the method proceeds to step 376 and ends.

The method 370 may be performed by the apparatus 200. The steps 371-376 can, for example, be performed by the at least one processor 201 and the memory 202. Further features of the method 370 directly result from the functionalities of the the
25 apparatus 200 and the computer-implemented lock-free data structure 120, 130, 140. The method 370 can be performed by a computer program product.

Fig. 4A is a diagram illustrating inserting a dummy data type in a lock-free manner. As shown in Fig. 4A, after the search has successfully stopped at an entry (regular object), the dummy data type may be made to point to the object found in the search

(non-CAS (compare-and-swap) operation). Fig. 4B is a diagram further illustrating inserting the dummy data type in the lock-free manner. As shown in Fig. 4B, the dummy data type may be inserted by doing a CAS operation at the object the search stopped at (the one before the sought object or the first dummy data type protecting it).

Fig. 4C is a diagram illustrating deleting a dummy data type in a lock-free manner. For deleting one dummy data type, a search for the object may be performed and stopped before the first dummy data type protecting it (since dummy data types could have accumulated while a read from the object was being performed). As shown in Fig. 4C, the dummy data type may be unlinked via one CAS operation on the element the search stopped at. Fig. 4D is a diagram further illustrating deleting the dummy data type in the lock-free manner. As shown in Fig. 4D, the dummy data type is de-allocated.

The functionality described herein can be performed, at least in part, by one or more computer program product components such as software components. According to an embodiment, the network device 100 comprises at least one processor configured by the program code when executed to execute the embodiments of the operations and functionality described. Alternatively, or in addition, the functionality described herein can be performed, at least in part, by one or more hardware logic components. For example, and without limitation, illustrative types of hardware logic components that can be used include Field-programmable Gate Arrays (FPGAs), Program-specific Integrated Circuits (ASICs), Program-specific Standard Products (ASSPs), System-on-a-chip systems (SOCs), Complex Programmable Logic Devices (CPLDs), and Graphics Processing Units (GPUs).

Any range or device value given herein may be extended or altered without losing the effect sought. Also any embodiment may be combined with another embodiment unless explicitly disallowed.

Although the subject matter has been described in language specific to structural features and/or acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as

examples of implementing the claims and other equivalent features and acts are intended to be within the scope of the claims.

It will be understood that the benefits and advantages described above may relate to one embodiment or may relate to several embodiments. The embodiments are not
5 limited to those that solve any or all of the stated problems or those that have any or all of the stated benefits and advantages. It will further be understood that reference to 'an' item may refer to one or more of those items.

The steps of the methods described herein may be carried out in any suitable order, or simultaneously where appropriate. Additionally, individual blocks may be deleted
10 from any of the methods without departing from the spirit and scope of the subject matter described herein. Aspects of any of the embodiments described above may be combined with aspects of any of the other embodiments described to form further embodiments without losing the effect sought.

The term 'comprising' is used herein to mean including the method, blocks or elements
15 identified, but that such blocks or elements do not comprise an exclusive list and a method or apparatus may contain additional blocks or elements.

It will be understood that the above description is given by way of example only and that various modifications may be made by those skilled in the art. The above specification, examples and data provide a complete description of the structure and
20 use of exemplary embodiments. Although various embodiments have been described above with a certain degree of particularity, or with reference to one or more individual embodiments, those skilled in the art could make numerous alterations to the disclosed embodiments without departing from the spirit or scope of this specification.

CLAIMS:

1. A computer-implemented lock-free data structure (120, 130, 140), comprising:
 - at least one data type (121A-121D, 123A-123C, 131A-131D, 133A, 141A-141E, 143A-143C); and
 - a memory pointer (122A-122H, 132A-132F, 142A-142K) per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type (121A-121D, 131A-131D, 141A-141E) or a dummy data type (123A-123C, 133A, 143A-143C), wherein the presence of a dummy data type (123A-123C, 133A, 143A-143C) in the computer-implemented lock-free data structure (120, 130, 140) indicates that a native data type (121A-121D, 131A-131D, 141A-141E) first following said dummy data type (123A-123C, 133A, 143A-143C) is protected against writes.
2. The computer-implemented lock-free data structure (120, 130, 140) according to claim 1, further comprising at least two dummy data types (123A-123C, 143A-143C) before a native data type (121C, 141E), each of said at least two dummy data types (123A-123C, 143A-143C) protecting the same native data type (121C, 141E) against writes and each of said at least two dummy data types (123A-123C, 143A-143C) inserted by a different party.
3. The computer-implemented lock-free data structure (120, 130, 140) according to claim 1, further comprising a single dummy data type (133A) before a native data type (131C) protecting said native data type (131C) against writes, said single dummy data type (133A) comprising a counter value (133A2) indicating the number of different parties accessing said native data type (131C).
4. The computer-implemented lock-free data structure (120, 130, 140) according to any of claims 1 to 3, wherein at least one native data type (121C, 131C, 141E) comprises a key (121C1, 131C1, 141E1) and each dummy data type (123A-123C, 133A, 143A-143C) protecting it against writes comprises a copy (123A1-123C1, 133A1, 143A1-143C1) of said key (121C1, 131C1, 141E1) or a pointer to said key (121C1, 131C1, 141E1).

5. The computer-implemented lock-free data structure (120, 130, 140) according to any of claims 1 to 4, wherein each party comprises a thread or a client.

6. The computer-implemented lock-free data structure (120, 130, 140) according to any of claims 1 to 5, wherein the computer-implemented lock-free data structure (120, 130, 140) comprises one of a linked list (120, 130) and a binary search tree (140).

7. A method (310, 330) of performing a search operation on a computer-implemented lock-free data structure comprising at least one data type, and a memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following said dummy data type is protected against writes,

15 the method comprising the steps:

starting (311, 331), by a processor, a search (313, 333) for an object in the computer-implemented lock-free data structure

in response to finding (314, 334) the object with no dummy data type immediately preceding it, inserting (316, 337), by the processor, the predetermined first positive number of linked dummy data types immediately before the found data type; or

in response to finding (314, 334) the object with at least one dummy data type immediately preceding it, inserting (316, 337), by the processor, the first predetermined positive number of linked dummy data types immediately before the first of the at least one dummy data type preceding the found data type;

in response to a successful (317, 338) insertion (316, 337) of the first positive number of linked dummy data types, performing (318, 339), by the processor, at least one read operation on the found object directly or via a callback function.

8. The method (310, 330) according to claim 7, further comprising:

deleting (319), by the processor, the successfully inserted first predetermined positive number of linked dummy data types upon exiting said at least one read operation.

5 9. The method (310) according to claim 7 or 8, wherein each dummy data type comprises a counter value indicating the number of different parties accessing its protected native data type, and the method further comprises:

in response to finding (334) the object with no dummy data type immediately preceding it, setting (337) the counter values of the inserted dummy
10 data types to a predetermined first value; or

in response to finding (334) the object with a dummy data type immediately preceding it, incrementing (337), by the processor, the counter value of the immediately preceding dummy data type by the predetermined first value;

15 decrementing (340), by the processor, the counter value by the predetermined first value upon exiting said at least one read operation; and

deleting (342), by the processor, the successfully inserted dummy data type when its counter value falls (341) below the predetermined first value.

10. A method (350) of performing a delete operation on a computer-implemented lock-free data structure comprising at least one data type, and a
20 memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following said dummy data type is
25 protected against writes, the method comprising:

performing (353), by a processor, a search for an object in the computer-implemented lock-free data structure; and

in response to finding (353) the object with no dummy data type immediately preceding it, performing (354), by the processor, a lock-free delete; or

in response to finding (353) the object with at least one dummy data type immediately preceding it, ending (356), by the processor, the search.

11. A method (370) of performing an update operation on a computer-implemented lock-free data structure comprising at least one data type, and a
 5 memory pointer per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type or a dummy data type, wherein the presence of a dummy data type in the computer-implemented lock-free data structure indicates that a native data type first following said dummy data type is
 10 protected against writes, comprising:

performing (373), by a processor, a search for an object in the computer-implemented lock-free data structure; and

in response to finding (373) the object with no dummy data type immediately preceding it, performing (374), by the processor, a lock-free update; or

15 in response to finding (373) the object with at least one dummy data type immediately preceding it, ending (376), by the processor, the search.

12. An apparatus (200), comprising at least one processor (201) configured to:

implement a lock-free data structure (120, 130, 140), comprising at
 20 least one data type (121A-121D, 123A-123C, 131A-131D, 133A, 141A-141E, 143A-143C); and a memory pointer (122A-122H, 132A-132F, 142A-142K) per each of the at least one data type, each memory pointer pointing to one of the at least one data type and each memory pointer comprising a tag indicating whether the pointed data type is a native data type (121A-121D, 131A-131D, 141A-141E) or a dummy data
 25 type (123A-123C, 133A, 143A-143C), wherein the presence of a dummy data type (123A-123C, 133A, 143A-143C) in the computer-implemented lock-free data structure (120, 130, 140) indicates that a native data type (121A-121D, 131A-131D, 141A-141E) first following said dummy data type (123A-123C, 133A, 143A-143C) is protected against writes; and

perform an access operation on the implemented lock-free data structure (120, 130, 140).

13. The apparatus (200) according to claim 12, wherein the at least one processor (201) is configured to perform the access operation by performing a search operation comprising:

starting a search for an object in the implemented lock-free data structure;

- in response to finding the object with no dummy data type immediately preceding it, inserting a predetermined first positive number of linked dummy data types immediately before the found data type; or

in response to finding the object with at least one dummy data type immediately preceding it, inserting the first positive number of linked dummy data types immediately before the first of the at least one dummy data type preceding the found data type;

- in response to a successful insertion of the first positive number of linked dummy data types, performing at least one read operation on the found object directly or via a callback function; and

deleting the successfully inserted first positive number of linked dummy data types upon exiting said at least one read operation.

14. The apparatus (200) according to claim 12 or 13, wherein each dummy data type comprises a counter value indicating the number of different parties accessing its protected native data type, and the at least one processor (201) is configured to perform the access operation by performing a search operation comprising:

- starting a search for an object in the implemented lock-free data structure;

in response to finding the object with no dummy data type immediately preceding it, inserting a dummy data type immediately before the found data type

and setting the counter value of the inserted dummy data type to a predetermined first value; or

in response to finding the object with a dummy data type immediately preceding it, incrementing the counter value of the immediately preceding dummy
5 data type by the predetermined first value;

in response to a successful insertion of the dummy data type or incrementation of the counter value of the immediately preceding dummy data type, performing at least one read operation on the found object directly or via a callback function;

10 decrementing the already set or incremented counter value by the predetermined first value upon exiting said at least one read operation; and

deleting the successfully inserted dummy data type when its counter value falls below the predetermined first value.

15 15. The apparatus (200) according to any of claims 12 to 14, wherein the at least one processor (201) is further configured to perform the access operation by performing a delete operation comprising:

performing a search for an object in the implemented lock-free data structure; and

20 in response to finding the object with no dummy data type immediately preceding it, performing a lock-free delete; or

in response to finding the object with at least one dummy data type immediately preceding it, ending the search.

16. The apparatus (200) according to any of claims 12 to 15, wherein the at least one processor (201) is further configured to perform the access operation by
25 performing an update operation comprising:

performing a search for an object in the implemented lock-free data structure; and

in response to finding the object with no dummy data type immediately preceding it, performing a lock-free update; or

in response to finding the object with at least one dummy data type immediately preceding it, ending the search.

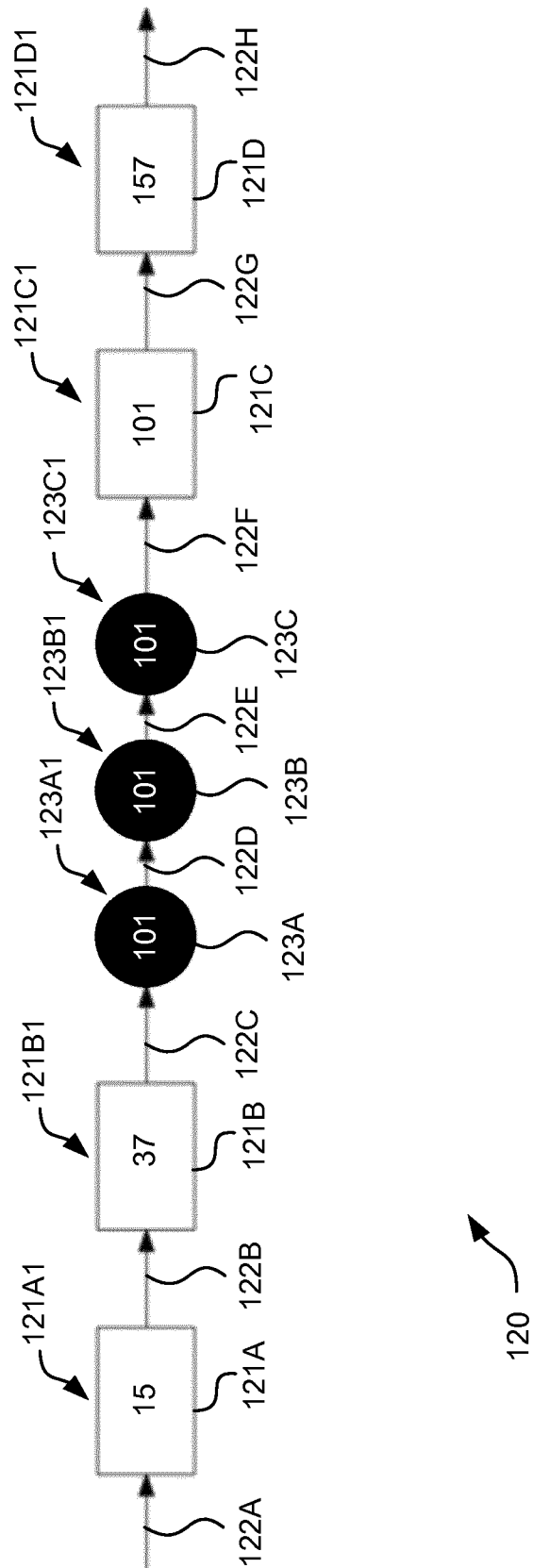


FIG. 1A

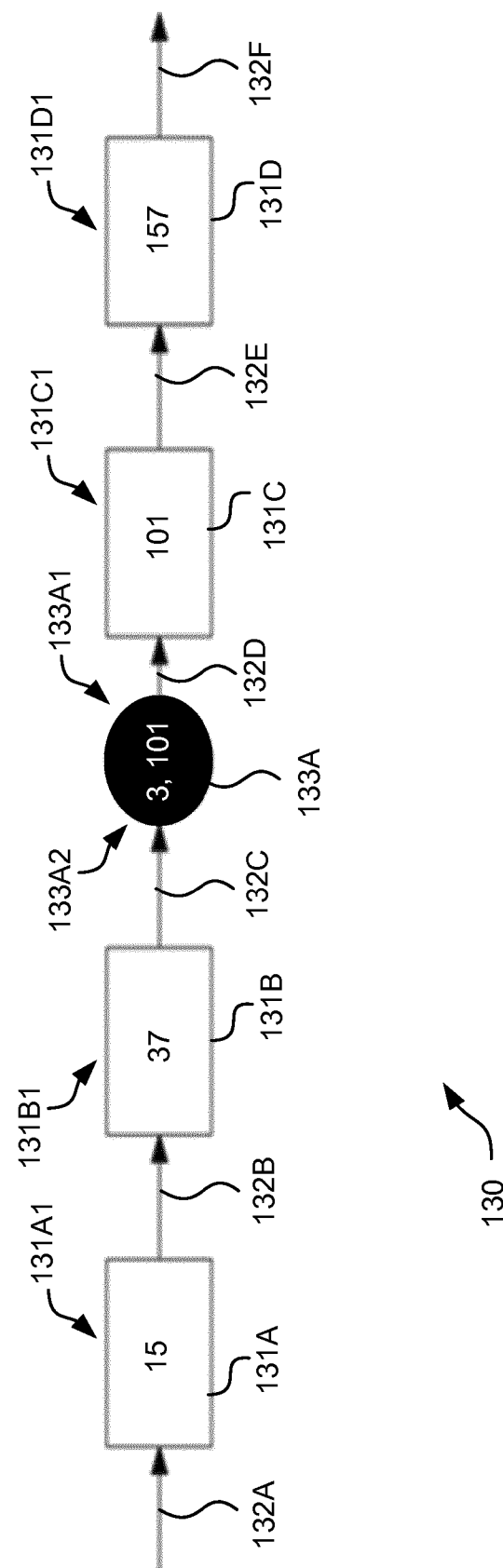


FIG. 1B

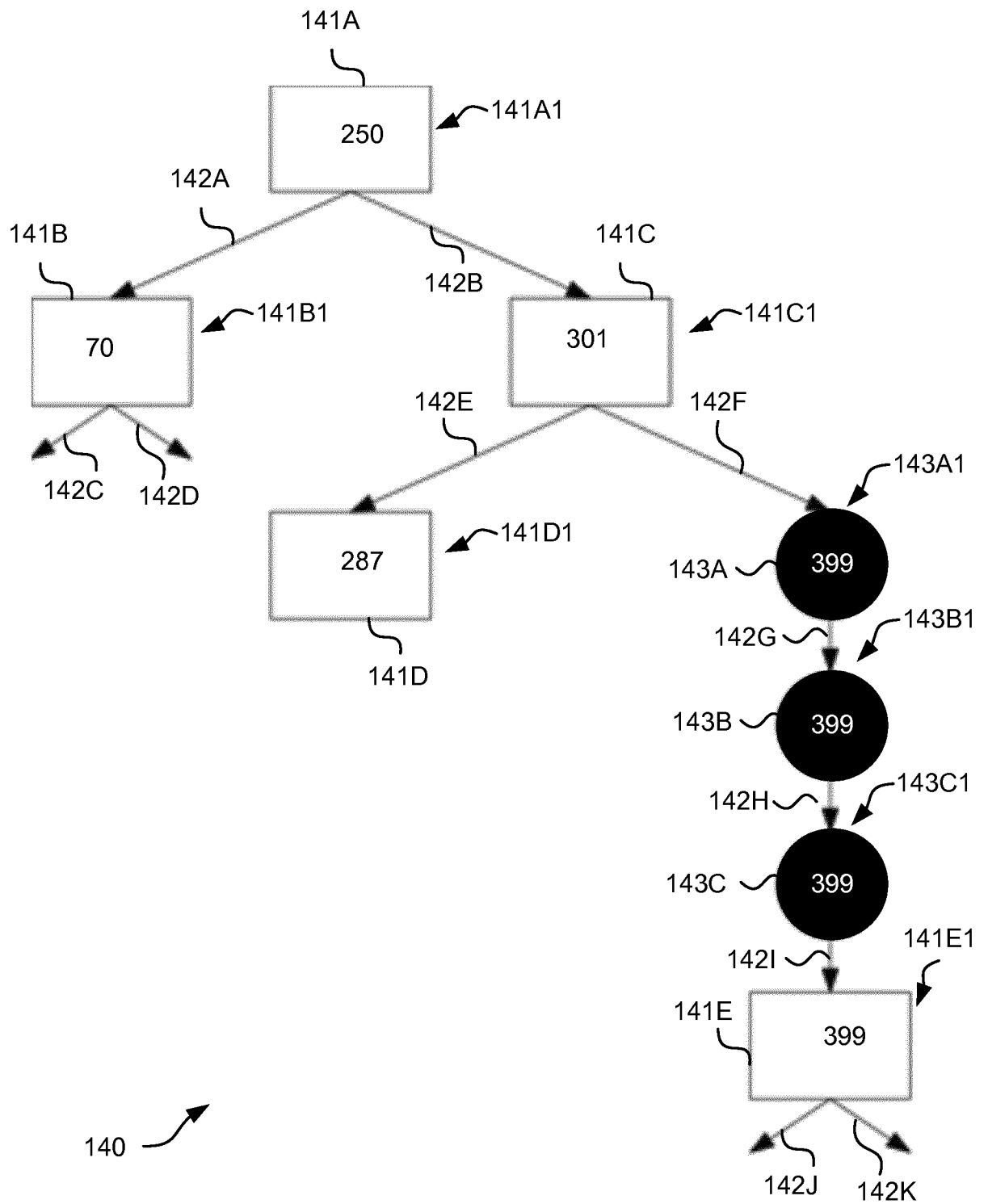


FIG. 1C

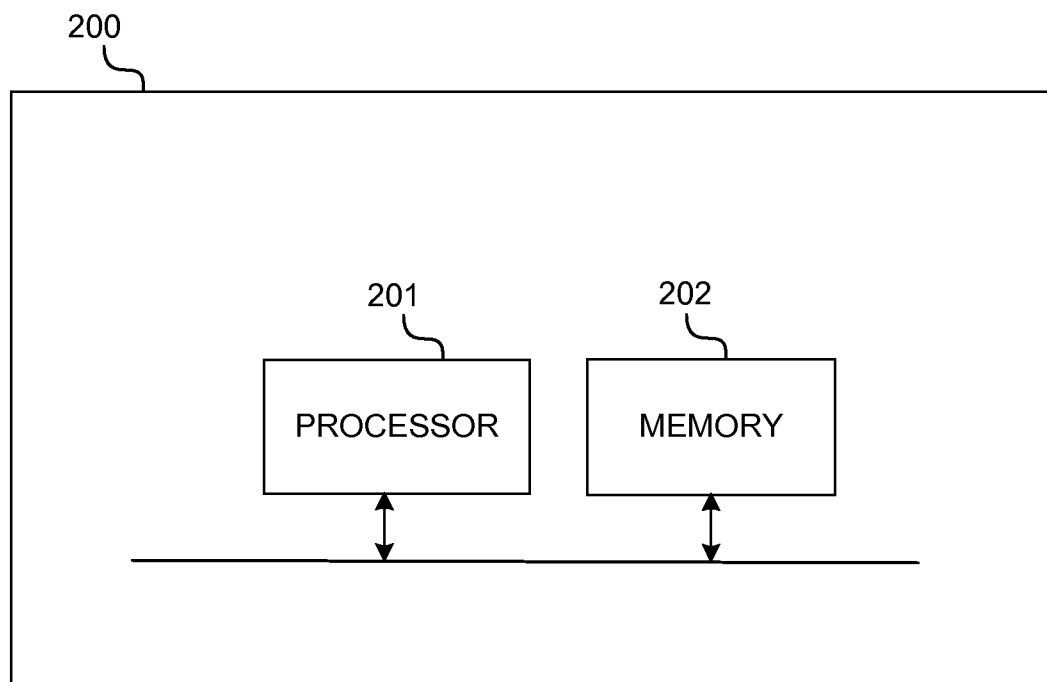


FIG. 2

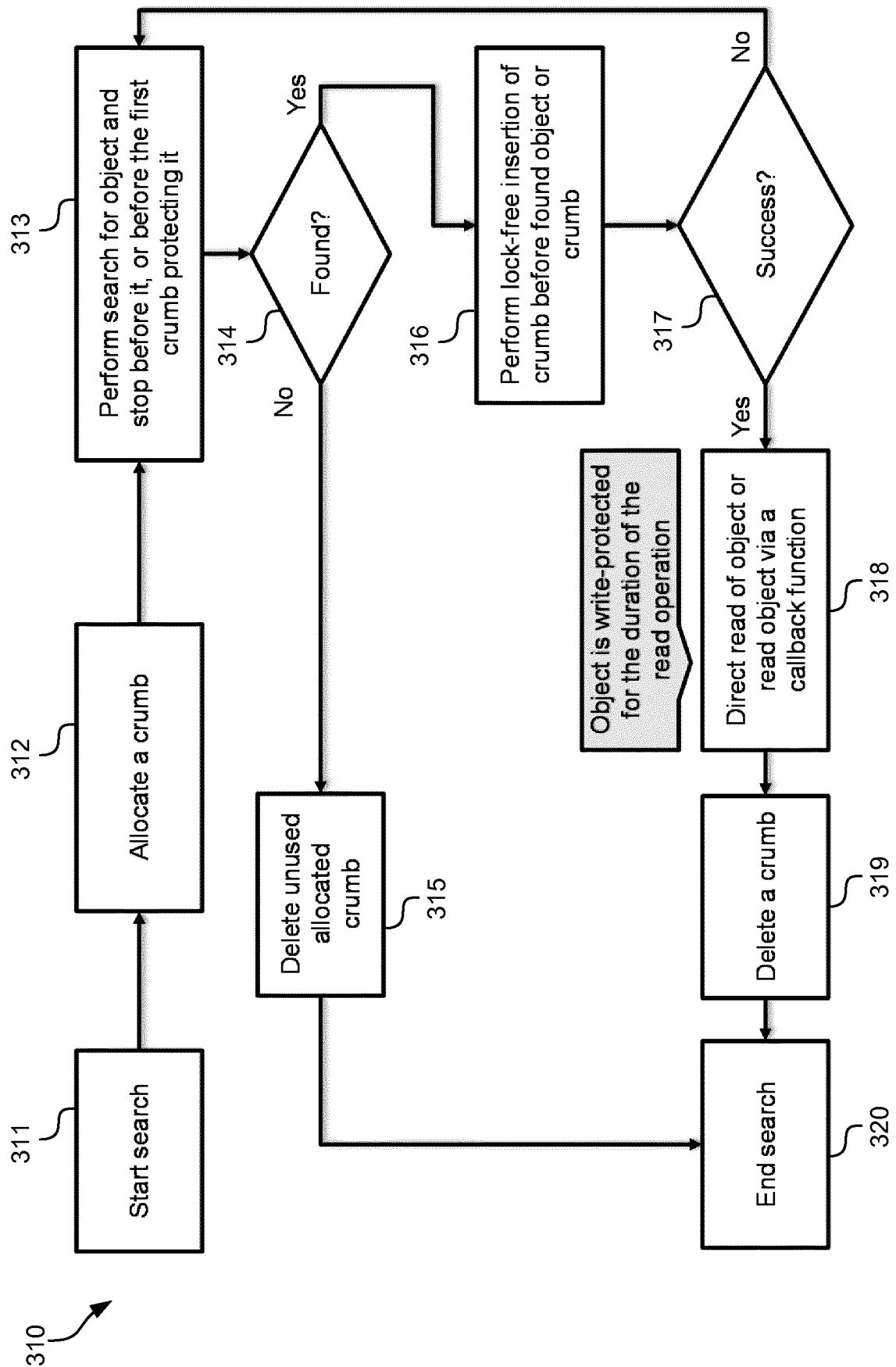


FIG. 3A

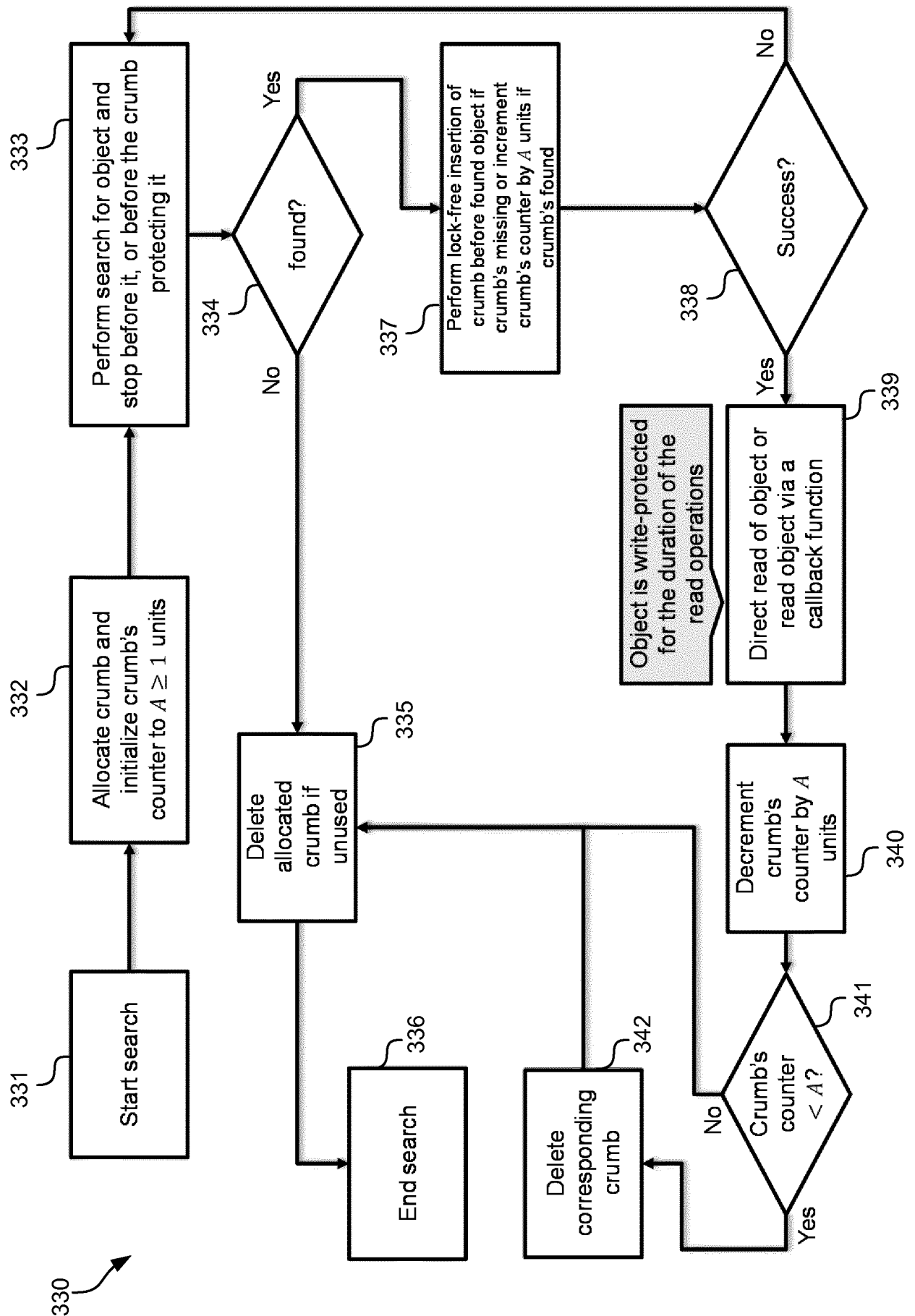


FIG. 3B

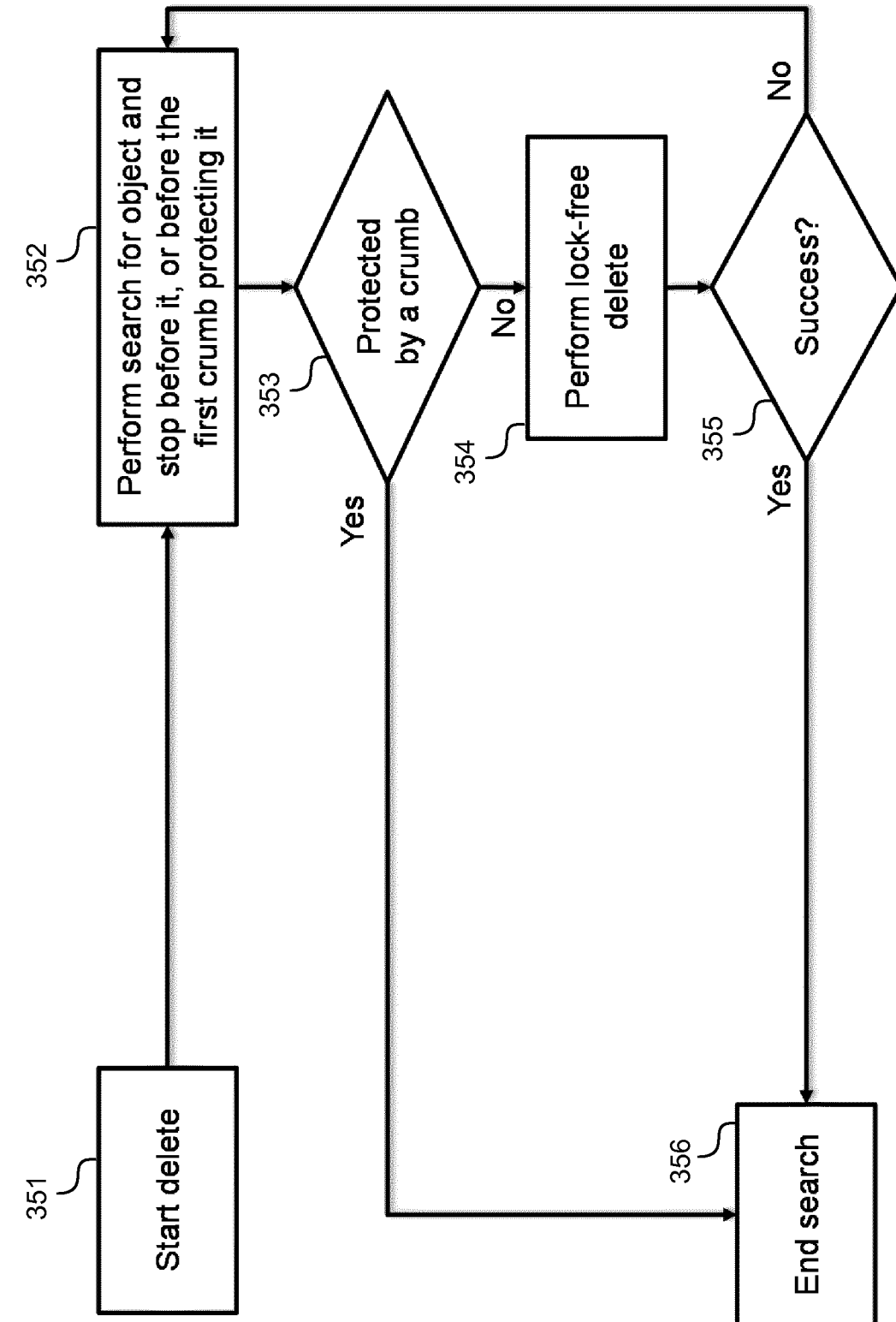


FIG. 3C

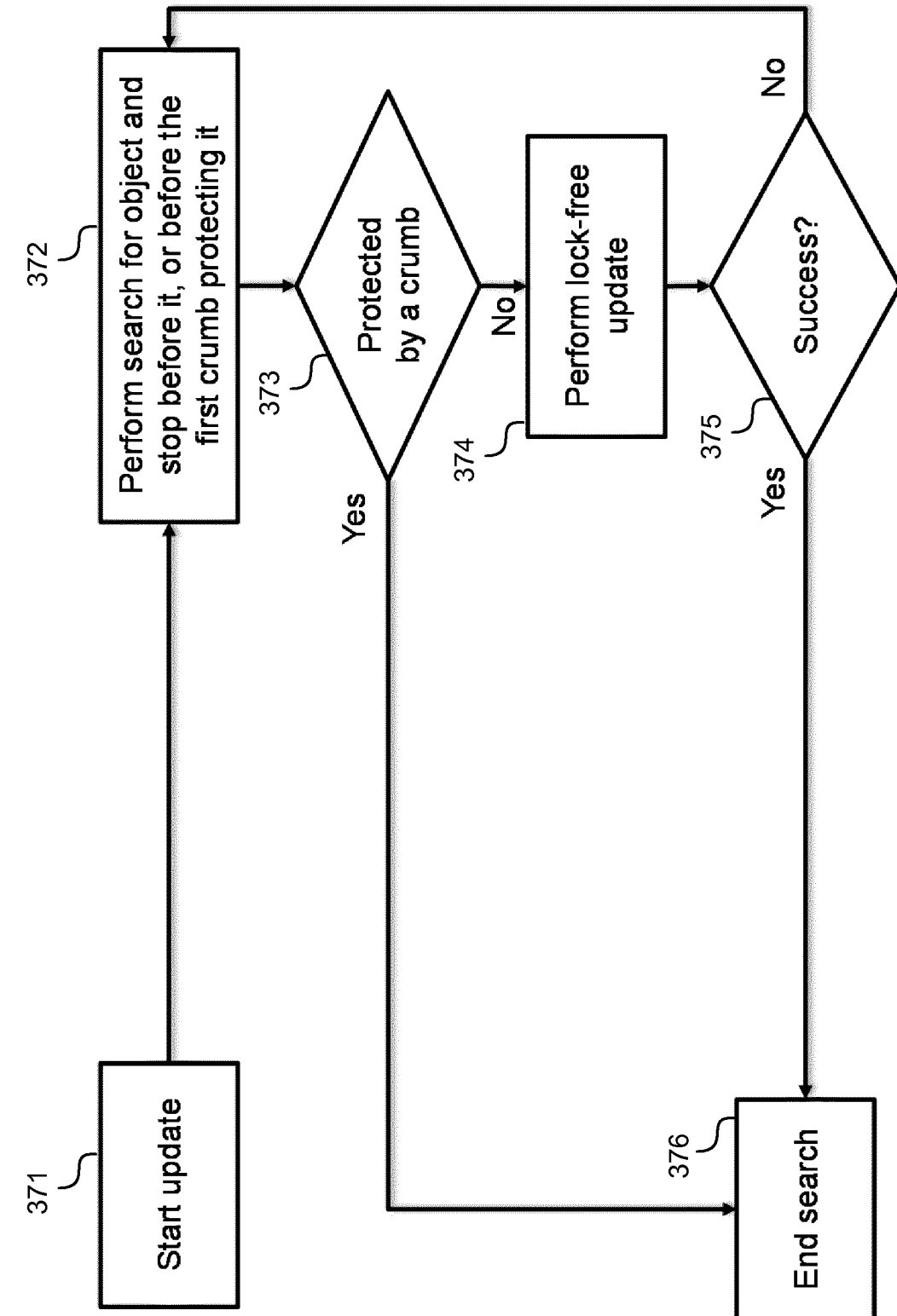


FIG. 3D

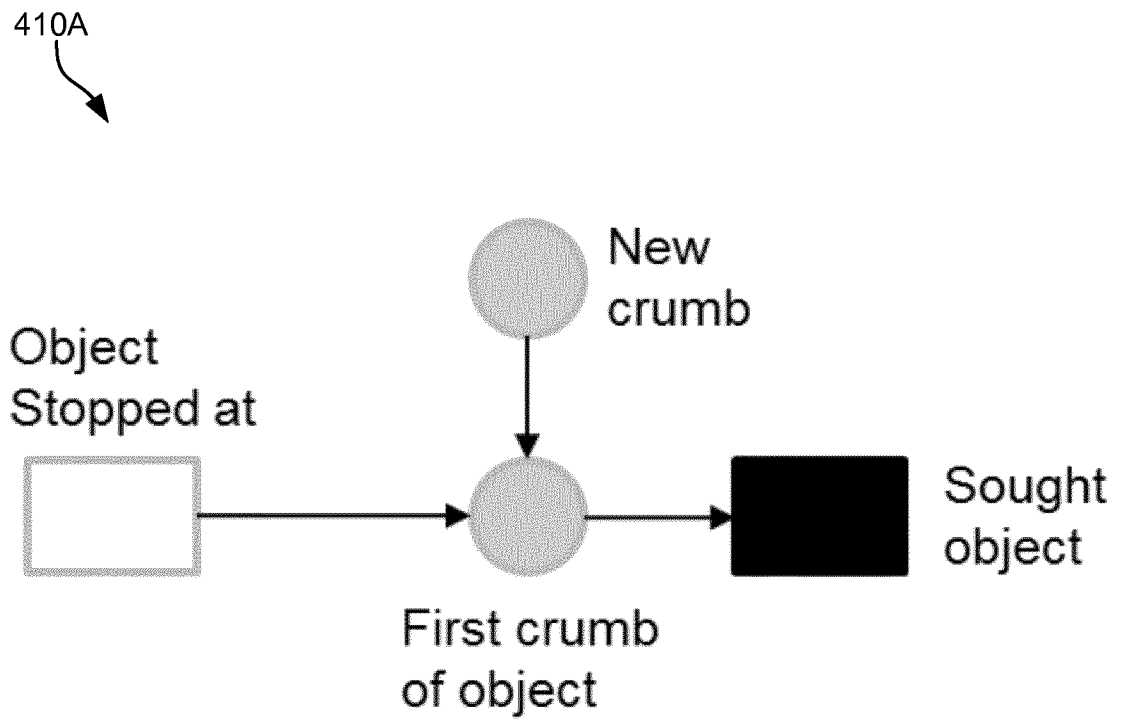


FIG. 4A

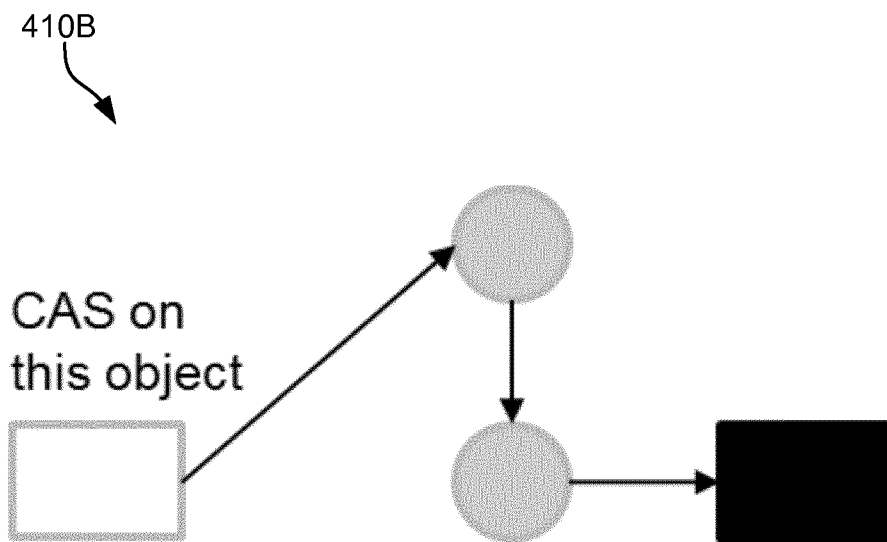


FIG. 4B

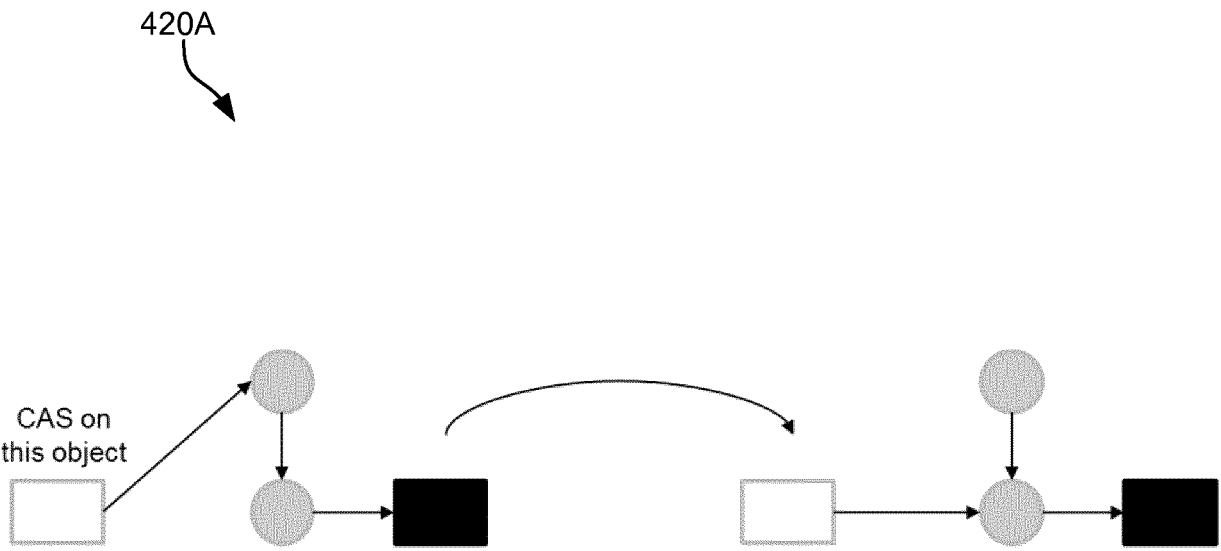


FIG. 4C

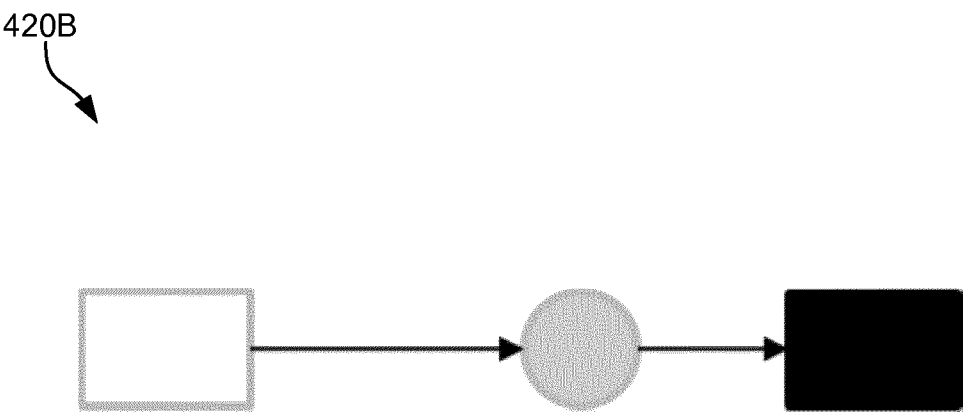


FIG. 4D

INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2019/058668

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/52
ADD. G06F12/02

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2005/071335 A1 (KADATCH ANDREW V [US]) 31 March 2005 (2005-03-31) paragraph [0019] paragraph [0021] - paragraph [0022] -----	1-16
X	JOY ARULRAJ ET AL: "Bztree", PROCEEDINGS OF THE VLDB ENDOWMENT; [ACM DIGITAL LIBRARY], ASSOC. OF COMPUTING MACHINERY, NEW YORK, NY, vol. 11, no. 5, 1 January 2018 (2018-01-01), pages 553-565, XP058384830, ISSN: 2150-8097, DOI: 10.1145/3164135.3164147 the whole document ----- -/-	1-16



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

25 October 2019

Date of mailing of the international search report

04/11/2019

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Alecu, Mihail

INTERNATIONAL SEARCH REPORT

International application No

PCT/EP2019/058668

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2004/107227 A1 (MICHAEL MAGED M [US]) 3 June 2004 (2004-06-03) paragraph [0004] - paragraph [0006] paragraph [0029] - paragraph [0032] -----	1-16

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/EP2019/058668

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2005071335	A1	31-03-2005	NONE

US 2004107227	A1	03-06-2004	NONE
