



- (51) International Patent Classification:
G06F 12/02 (2006.01)
- (21) International Application Number:
PCT/US2013/052780
- (22) International Filing Date:
30 July 2013 (30.07.2013)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
61/677,390 30 July 2012 (30.07.2012) US
13/782,433 1 March 2013 (01.03.2013) US
- (71) Applicant: ORACLE INTERNATIONAL CORPORATION [US/US]; 500 Oracle Parkway, M/S 50p7, Redwood Shores, California 94065 (US).
- (72) Inventor: AHMAD, Saqib; 101 Esfahan Drive, San Jose, California 95111 (US).
- (74) Agents: MEYER, Sheldon, R. et al.; Fliesler Meyer LLP, 410 Pacific Avenue, San Francisco, California 94133 (US).
- (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, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, 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, 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, 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))
- before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))

(54) Title: SYSTEM AND METHOD FOR OBJECT DELETION IN PERSISTENT MEMORY USING BITMAP WINDOWS

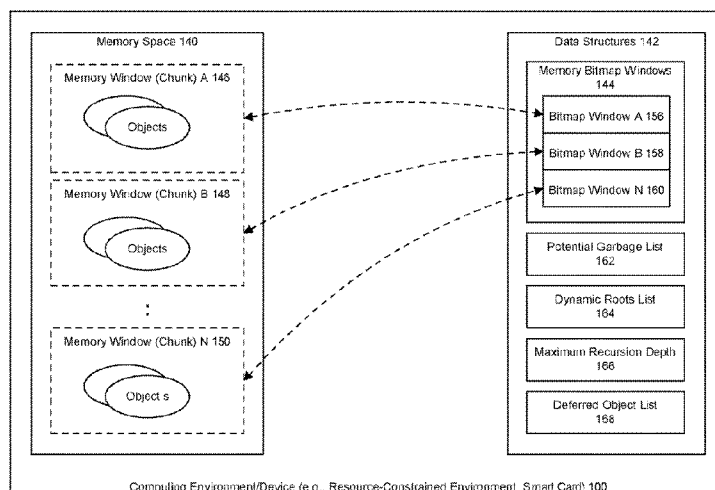


FIGURE 2

(57) Abstract: A system and method for object deletion in persistent memory using bitmap windows representing memory chunks. In accordance with an embodiment, the system can generally be used with computing environments (100) that use persistent memory, such as smart cards, Java Cards, and other resource-constrained environments. In accordance with an embodiment, the system comprises a processor or computational engine and a persistent memory for storage of software objects; and a data structure (142) which can include one or more memory bitmap windows (144), each of which represents a chunk (146,148,150) of addressable space in the persistent memory (140); wherein the system uses the one or more memory bitmap windows in deleting non-reachable objects from the persistent memory.



SYSTEM AND METHOD FOR OBJECT DELETION IN PERSISTENT MEMORY USING BITMAP WINDOWS

5

COPYRIGHT NOTICE

10

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Field of Invention:

15 **[0001]** Embodiments of the invention are generally related to persistent memory storage, and are particularly related to resource-constrained environments and other computing environments, and to systems and methods for deleting non-reachable objects from persistent memory, using bitmap windows representing memory chunks.

20 Background:

[0002] A typical resource-constrained environment, such as a smart card or similar execution environment or device, may have limited random access memory resources, and instead use persistent memory as a default location for creating and storing software objects.

25 **[0003]** Such environments have a need for reclaiming persistent memory which is in use by objects that have become garbage by virtue of being nonreachable from a defined set of roots. Garbage collection algorithms which may be useful in other environments, such as Java SE, often rely on techniques such as moving or marking the objects directly. However, these techniques are not as useful in a resource-constrained, smart card, or similar execution environment, since moving or marking objects in persistent memory is a time-consuming, and potentially damaging, operation
30 given that only a limited number of data writes can be made to a persistent memory device before it wears out.

Summary:

35 **[0004]** Described herein is a system and method for object deletion in persistent memory using bitmap windows representing memory chunks. In accordance with an embodiment, the system can generally be used with computing environments that use persistent memory, such as smart cards, Java Cards, and other resource-constrained environments. In accordance with an embodiment, the system comprises a processor or computational engine and a persistent memory for storage of software objects; and a data structure which can include one or more memory bitmap
40 windows, each of which represents a chunk of addressable space in the persistent memory; wherein the system uses the one or more memory bitmap windows in deleting non-reachable

objects from the persistent memory.

Brief Description of the Figures:

[0005] **Figure 1** illustrates an example of a computing environment that uses persistent memory, such as a resource-constrained environment or smart card, in accordance with an embodiment.

[0006] **Figure 2** illustrates the use of bitmap windows representing memory chunks, for object deletion in persistent memory, in accordance with an embodiment.

[0007] **Figure 3** further illustrates the use of bitmap windows for object deletion in persistent memory, in accordance with an embodiment.

[0008] **Figure 4** further illustrates the use of bitmap windows for object deletion in persistent memory, in accordance with an embodiment.

[0009] **Figure 5** is a flowchart illustrating an algorithm or method of using bitmap windows for object deletion in persistent memory, in accordance with an embodiment.

[0010] **Figure 6** is a flowchart further illustrating an algorithm or method of using bitmap windows for object deletion in persistent memory, in accordance with an embodiment.

[0011] **Figure 7** is a flowchart further illustrating an algorithm or method of using bitmap windows for object deletion in persistent memory, in accordance with an embodiment.

Detailed Description:

[0012] As described above, a typical resource-constrained environment, such as a smart card or similar execution environment or device, may have limited random access memory resources, and instead use persistent memory as a default location for creating and storing software objects. Such environments have a need for reclaiming persistent memory which is in use by objects that have become garbage by virtue of being nonreachable from a defined set of roots. However, garbage collection algorithms which may be useful in other environments, but rely on techniques such as moving or marking the objects directly, are not as useful in a resource-constrained, smart card, or similar execution environment, since moving or marking objects in persistent memory is a time-consuming, and potentially damaging, operation given that only a limited number of data writes can be made to a persistent memory device before it wears out.

[0013] Some examples of the types of approach that can be used to address this include: use of a brute force algorithm to check each object to determine if it is referenced from any other object in the system; use of a reference counting algorithm which updates objects with a reference count such that a reference count of 0 indicates that the object is not referenced by another; or use of a RAM bitmap representing objects in memory.

[0014] A problem with the first two of these approaches is that they do not account for cyclical references for even simple cycles. The first approach is also time-consuming since in a

worst-case scenario potentially every object in the system will have to be checked for every other object in the system, to determine references from one object to another. A problem with the third approach is that it relies on dereferencing objects, such that the system uses object references from an object table rather than actual address; however this slows the overall execution environment since many operations will require object addresses, and will have to go through the conversion process to get the address from a reference.

[0015] In accordance with an embodiment, described herein is a system and method for object deletion in persistent memory using bitmap windows representing memory chunks. In accordance with an embodiment, the system can generally be used with computing environments that use persistent memory, such as smart cards, Java Cards, and other resource-constrained environments. In accordance with an embodiment, the system comprises a processor or computational engine and a persistent memory for storage of software objects; and a data structure which can include one or more memory bitmap windows, each of which represents a chunk of addressable space in the persistent memory; wherein the system uses the one or more memory bitmap windows in deleting non-reachable objects from the persistent memory.

[0016] In accordance with various embodiments, some advantages of the systems and methods described herein include that it can provide a fast version of mark-sweep garbage collection technique which addresses the cyclical referencing problem; that there is no need for dereferencing objects; that there is also no need to write to persistent memory until actual reclamation of memory; and that the approach is scalable such that an increase in the overall target memory area does not increase resource requirements.

[0017] **Figure 1** illustrates an example of a computing environment that uses persistent memory, such as a resource-constrained environment or smart card, in accordance with an embodiment.

[0018] As shown in Figure 1, in accordance with an embodiment, a computing environment or device 100, such as a resource-constrained environment, smart card, or other environment, can be considered a miniature computing device, comprising a processor or computational engine 126, which can include circuitry for performing computational operations; and one or more memory, such as a random access memory (RAM) 120, electrically erasable programmable read-only memory (EEPROM) 122, and/or read-only memory (ROM) 124.

[0019] In accordance with an embodiment, the RAM can include a volatile random access memory, the EEPROM can include a writeable non-volatile memory, and the ROM can include a read-only memory. The RAM can be used to store data items and data structures, such as a system stack 121. The ROM can include a virtual machine 108, such as a Java virtual machine, which allows applications written in a platform-independent programming language, such as Java, to be executed by the smart card; and can also contain one or more applications 104, 105, which provide services for clients accessing the smart card. In accordance with an embodiment,

applications 106 can also be located in EEPROM, or in both ROM and/or EEPROM.

[0020] In accordance with an embodiment, the system can include a card manager 102, or a similar logic, which includes code for managing the execution of applications on the computing environment or device (including, for example, enabling communications between clients and applications); and deleting objects from memory. For example, as illustrated in Figure 1, during operation of the device the EEPROM can contain a plurality of objects 112, 113, 114, which are accessed by applications 104, 105 (more specifically, application 104 accesses object 112, application 105 accesses objects 113 and 114, and application 106 accesses object 114); while other objects 115, 116, that have become unlinked, are no longer referenced by any application. It is generally desirable to delete these unreferenced objects to free up memory space.

[0021] The environment illustrated in Figure 1 is provided by way of illustration of one type of environment that can utilize the systems and methods described herein. In accordance with various embodiments, the invention is not limited to resource-constrained environments or smart cards, or to the particular environment or configuration described above, and can generally be used with other types of computing environment, device, or system that can store object in a writeable persistent or non-volatile memory.

[0022] In accordance with an embodiment, a system and method is provided for use of bitmap windows which represent a chunk of memory. The technique uses an algorithm or method which can work with very limited resources, while mapping a large amount of addressable space. The technique is also scaleable such that, for increased persistent memory size, even if the overall RAM size remains relatively unchanged, the algorithm or method will still work. Additionally, the algorithm or method does not reduce overall system performance as might be the case with brute-force, reference counting, and dereferencing techniques.

[0023] **Figure 2** illustrates the use of bitmap windows representing memory chunks, for object deletion in persistent memory, in accordance with an embodiment.

[0024] As shown in Figure 2, in accordance with an embodiment, the system can include a data structure 142, which can include one or more memory bitmap windows 144 (156,158,160), which represent a chunk of addressable space 146, 148, 150 in persistent memory 140, for storage of objects. The data structure can further include a potential garbage list 162, which lists objects which are not referenced in windows that have already been processed; a dynamic roots list 164, which lists objects referenced from a previous window but not in the addressable space in the previous window, which become roots for a next window; a maximum recursion depth 166, which is a configurable value that limits the recursion depth for marking phase; and a deferred object list 168, which once the configured recursion depth is reached, includes the children objects which have not been processed yet.

[0025] **Figure 3** and **Figure 4** further illustrate the use of bitmap windows for object deletion in persistent memory, in accordance with an embodiment.

[0026] In accordance with an embodiment, in order to avoid unnecessary writes to EEPROM during the marking process, the system can maintain a bitmap in RAM. In some environments, for example a typical resource-constrained environment, RAM is a scarce resource; a typical amount might be 2k, which provides little space for bitmap storage. By way of example, a 512 byte bitmap contains 4k bits, and each bit can represent 4 bytes, so a bitmap of this size can be used to represent 16k of memory.

[0027] In accordance with an embodiment, during the object deletion process, the system can perform a marking process, which "marks" all objects that are reachable in memory. This can be accomplished by following chains of references from "roots", such as from applet tables and static variables, to find reachable objects. For example, an applet object can point to another object, which points to yet other objects, each of which also point to other objects. Similarly, static variables can point to objects, which point to yet other objects.

[0028] In accordance with an embodiment, during a marking phase, starting from a window's root objects, its object children are marked. If a child object resides outside a window, and in a next window, then that object can be put into the dynamic roots list as a root for the next window. If a child object is within this window, then it can be marked, and the system can recurse on the children of that object. If a child object is in a previous window, then the system checks the potential garbage list and recurses on that object, and then removes it from the potential garbage list since it is addressable from this window. After the marking phase for a window, the system looks for objects that were not marked, and puts those in a potential garbage list. These objects may not actually be considered garbage yet, since they may be addressable from a next window. The system then processes the next windows. Ultimately, after completing the windows, the potential garbage list will contain those objects which are not referenced and can be considered garbage.

[0029] As shown in Figure 3, in accordance with an embodiment, the memory space is divided into windows. A bitmap is initialized for the windows, wherein a bit represents an address in memory. For example, if memory management is performed at 4-byte alignment, then a bit can represent the first byte in the 4-byte memory cell. Then, the bitmap windows are processed. For each window, the root objects are marked, including marking applet objects 170 within the current window as indicated by an applet table 169; marking objects 174 that are referenced from static fields 176 which are in the window; and marking objects 178 from the dynamic roots table which are in the window.

[0030] In accordance with an embodiment, children objects are then marked, including if a child object 171, 172 is found within the current window, marking that object by marking 184 the corresponding bit in the bitmap. If a child object 180 is found outside the current window in the potential garbage list, then the object and its children are marked recursively in the current window and the potential garbage list. If a child object is found outside the current window and is not in the

potential garbage list then, if the object is in an already processed area of the memory, it has already been marked as reachable, and therefore it can be ignored. If, instead, the child object is in an area of memory that still needs to be processed, then it is put in the dynamic roots list.

[0031] In accordance with an embodiment, while marking child objects, if a maximum recursion depth is reached, then the next object in the hierarchy is put on the deferred object list, and will be processed at the end of processing the current hierarchy. If the deferred object list is full, and the maximum recursion depth has been reached, then the marking process is started again in the current window, without resetting the bitmap, therefore increasing the number of roots in the second and subsequent cycles. Once the marking phase is complete in a window, any unreferenced objects within the window are put on the potential garbage list.

[0032] As shown in Figure 4, the process continues with the next window and with its root and child objects 185, 186, 187, 188. When all of the windows have been processed, the potential garbage list will only contain objects 189 which are not referenced, and are therefore garbage. In accordance with an embodiment, these objects can be deleted to reclaim the memory used.

[0033] **Figure 5** is a flowchart illustrating an algorithm or method 190 of using bitmap windows for object deletion in persistent memory, in accordance with an embodiment.

[0034] As shown in Figure 5, at step 192, the memory space is divided into windows.

[0035] At step 194, a bitmap is initialized for the windows, wherein a bit represents an address in memory.

[0036] At steps 196, 198 and 202, the bitmap windows are processed. In accordance with an embodiment, for each window, root objects are marked, including marking applet objects within the current window, marking objects that are referenced from static fields which are in the window, and marking objects from the dynamic roots table which are in the window. Then, children objects are marked, subject to criteria as described in further detail below, and the bitmaps updated. Any unreferenced objects in the current window are put on the potential garbage list.

[0037] At step 204, when all of the windows have been processed, the potential garbage list will only contain objects which are not referenced, and are therefore garbage. In accordance with an embodiment, these objects can be deleted to reclaim the memory used.

[0038] **Figure 6** is a flowchart further illustrating an algorithm or method 210 of using bitmap windows for object deletion in persistent memory, in accordance with an embodiment. In particular Figure 6 further illustrates how the marking of root and children objects can be performed, in accordance with an embodiment.

[0039] As shown in Figure 6, at step 212, for each window, the root objects are marked, including marking applet objects within the current window, marking objects that are referenced from static fields which are in the window, and marking objects from the dynamic roots table which are in the window.

[0040] At step 214, the children objects are marked, including if a child object is found within

the current window, marking that object by marking the corresponding bit in the bitmap. If a child object is found outside the current window in the potential garbage list, then the object and its children are marked recursively in the current window and the potential garbage list. If a child object is found outside the current window and is not in the potential garbage list then, if the object is in an already processed area of the memory, it has already been marked as reachable, and therefore it can be ignored. If, instead, the child object is in an area of memory that still needs to be processed, then it is put in the dynamic roots list.

[0041] At step 216, while marking child objects, if a maximum recursion depth is reached, then the next object in the hierarchy is put on the deferred object list, and will be processed at the end of processing the current hierarchy.

[0042] At step 220, if the deferred object list is full, and the maximum recursion depth has been reached, then the marking process is started again in the current window, without resetting the bitmap, therefore increasing the number of roots in the second and subsequent cycles.

[0043] At step 222, once the marking phase is complete in a window, any unreferenced objects within the window are put on the potential garbage list. As described above, when all of the windows have been processed, the potential garbage list will only contain objects which are not referenced, and are therefore garbage. In accordance with an embodiment, these objects can be deleted to reclaim the memory used.

[0044] **Figure 7** is a flowchart further illustrating an algorithm or method 230 of using bitmap windows for object deletion in persistent memory, in accordance with an embodiment. In particular, Figure 7 illustrates the combination of each of the steps previously described at Figures 5 and 6, in accordance with an embodiment.

[0045] In general, the invention relates to a system for object deletion in persistent memory using bitmap windows representing memory chunks, operating on one or more microprocessors, comprising:

means for providing, at a computing environment or device, such as a resource-constrained environment, smart card, or other environment, including a processor or computational engine and a persistent memory for storage of software objects, a data structure which can include one or more memory bitmap windows, each of which represents a chunk of addressable space in the persistent memory; and

means for using the one or more memory bitmap windows in deleting non-reachable objects from the persistent memory.

[0046] Said system further comprising means for providing within the data structure one or more of:

- a potential garbage list;
- a dynamic roots list;
- a maximum recursion depth; and

a deferred object list.

[0047] Said system further comprising:

the potential garbage list includes objects which are not referenced in windows that have already been processed;

5 the dynamic roots list includes objects referenced from a previous window but not in the addressable space in the previous window, which become roots a next window;

the maximum recursion depth is a configurable value that limits the recursion depth for a marking phase; and

10 the deferred object list, once the configured recursion depth is reached, includes children objects which have not been processed yet.

[0048] Said system further comprising:

means for dividing the persistent memory space used for storage of object in windows;

15 means for initializing a bitmap for the windows where a bit represents an address in memory;

for each window, marking roots, and marking children recursively;

when the marking phase is complete in a window, means for putting any unreferenced objects within this window on the potential garbage list; and

20 means for processing each of the windows such that the potential garbage list ultimately contains those objects which are not referenced and can be considered garbage.

[0049] The present invention may be conveniently implemented using one or more conventional general purpose or specialized digital computer, computing device, machine, or microprocessor, including one or more processors, memory and/or computer readable storage media programmed according to the teachings of the present disclosure. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art.

[0050] In some embodiments, the present invention includes a computer program product which is a non-transitory storage medium or computer readable medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the present invention. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

35 **[0051]** The foregoing description of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the

practitioner skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the

5 following claims and their equivalence.

Claims:

What is claimed is:

- 5 1. A system for object deletion in persistent memory using bitmap windows representing memory chunks, comprising:
- a computing environment or device, such as a resource-constrained environment, smart card, or other environment, including a processor or computational engine and a persistent memory for storage of software objects;
- 10 a data structure which can include one or more memory bitmap windows, each of which represents a chunk of addressable space in the persistent memory; and
- wherein the system uses the one or more memory bitmap windows in deleting non-reachable objects from the persistent memory.
- 15 2. The system of claim 1, wherein the data structure includes one or more of:
- a potential garbage list;
- a dynamic roots list;
- a maximum recursion depth; and
- a deferred object list.
- 20 3. The system of claim 2, wherein
- the potential garbage list includes objects which are not referenced in windows that have already been processed;
- the dynamic roots list includes objects referenced from a previous window but not in the
- 25 addressable space in the previous window, which become roots a next window;
- the maximum recursion depth is a configurable value that limits the recursion depth for a marking phase; and
- the deferred object list, once the configured recursion depth is reached, includes children objects which have not been processed yet.
- 30 4. The system of claim 3, wherein the system performs a method comprising
- dividing the persistent memory space used for storage of object in windows;
- initializing a bitmap for the windows where a bit represents an address in memory;
- for each window, marking roots, and marking children recursively;
- 35 when the marking phase is complete in a window, putting any unreferenced objects within this window on the potential garbage list; and
- processing each of the windows such that the potential garbage list ultimately contains

those objects which are not referenced and can be considered garbage.

5. A method of object deletion in persistent memory using bitmap windows representing memory chunks, comprising the steps of:

5 providing, at a computing environment or device, such as a resource-constrained environment, smart card, or other environment, including a processor or computational engine and a persistent memory for storage of software objects, a data structure which can include one or more memory bitmap windows, each of which represents a chunk of addressable space in the persistent memory; and

10 using the one or more memory bitmap windows in deleting non-reachable objects from the persistent memory.

6. The method of claim 5, further comprising providing within the data structure one or more of:

15 a potential garbage list;
a dynamic roots list;
a maximum recursion depth; and
a deferred object list.

7. The method of claim 6, wherein

20 the potential garbage list includes objects which are not referenced in windows that have already been processed;

the dynamic roots list includes objects referenced from a previous window but not in the addressable space in the previous window, which become roots a next window;

25 the maximum recursion depth is a configurable value that limits the recursion depth for a marking phase; and

the deferred object list, once the configured recursion depth is reached, includes children objects which have not been processed yet.

8. The method of claim 7, further comprising:

30 dividing the persistent memory space used for storage of object in windows;
initializing a bitmap for the windows where a bit represents an address in memory;
for each window, marking roots, and marking children recursively;

when the marking phase is complete in a window, putting any unreferenced objects within this window on the potential garbage list; and

35 processing each of the windows such that the potential garbage list ultimately contains those objects which are not referenced and can be considered garbage.

9. A computer program comprising program code instructions that when executed by a computer system cause the computer system to perform the method of any preceding claim.

10. A computer program product comprising the computer program of claim 9 stored on a machine readable medium.

11. A non-transitory computer readable medium including instructions stored thereon which when read and executed by a computing environment or device cause the computing environment or device to perform the steps comprising:

10 providing, at the computing environment or device, such as a resource-constrained environment, smart card, or other environment, including a processor or computational engine and a persistent memory for storage of software objects, a data structure which can include one or more memory bitmap windows, each of which represents a chunk of addressable space in the persistent memory; and

15 using the one or more memory bitmap windows in deleting non-reachable objects from the persistent memory.

12. The non-transitory computer readable medium of claim 11, further comprising providing within the data structure one or more of:

20 a potential garbage list;
a dynamic roots list;
a maximum recursion depth; and
a deferred object list.

25 13. The non-transitory computer readable medium of claim 12, wherein
the potential garbage list includes objects which are not referenced in windows that have already been processed;

the dynamic roots list includes objects referenced from a previous window but not in the addressable space in the previous window, which become roots a next window;

30 the maximum recursion depth is a configurable value that limits the recursion depth for a marking phase; and

the deferred object list, once the configured recursion depth is reached, includes children objects which have not been processed yet.

35 14. The non-transitory computer readable medium of claim 13, further comprising:
dividing the persistent memory space used for storage of object in windows;
initializing a bitmap for the windows where a bit represents an address in memory;

for each window, marking roots, and marking children recursively;

when the marking phase is complete in a window, putting any unreferenced objects within this window on the potential garbage list; and

processing each of the windows such that the potential garbage list ultimately contains

5 those objects which are not referenced and can be considered garbage.

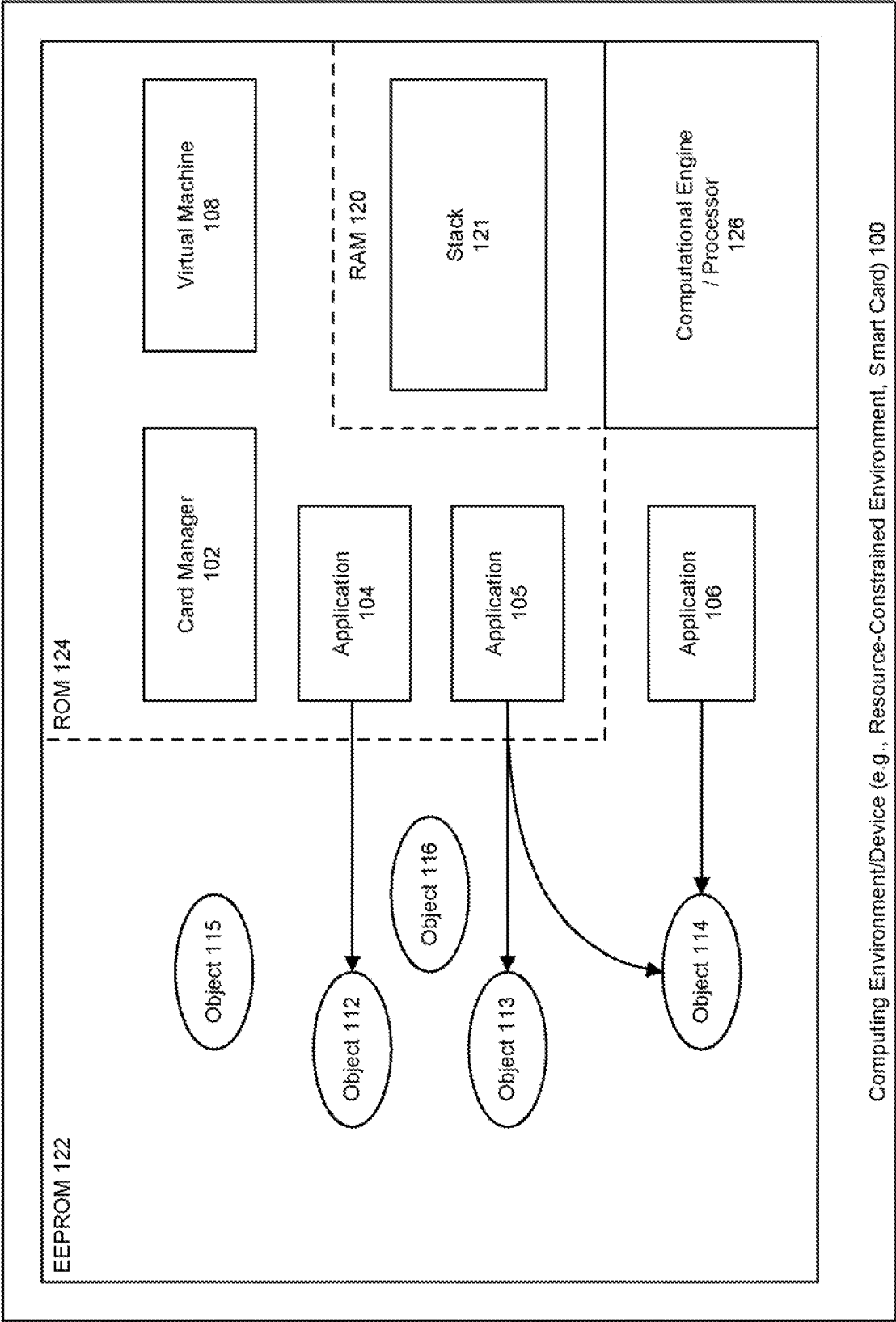


FIGURE 1

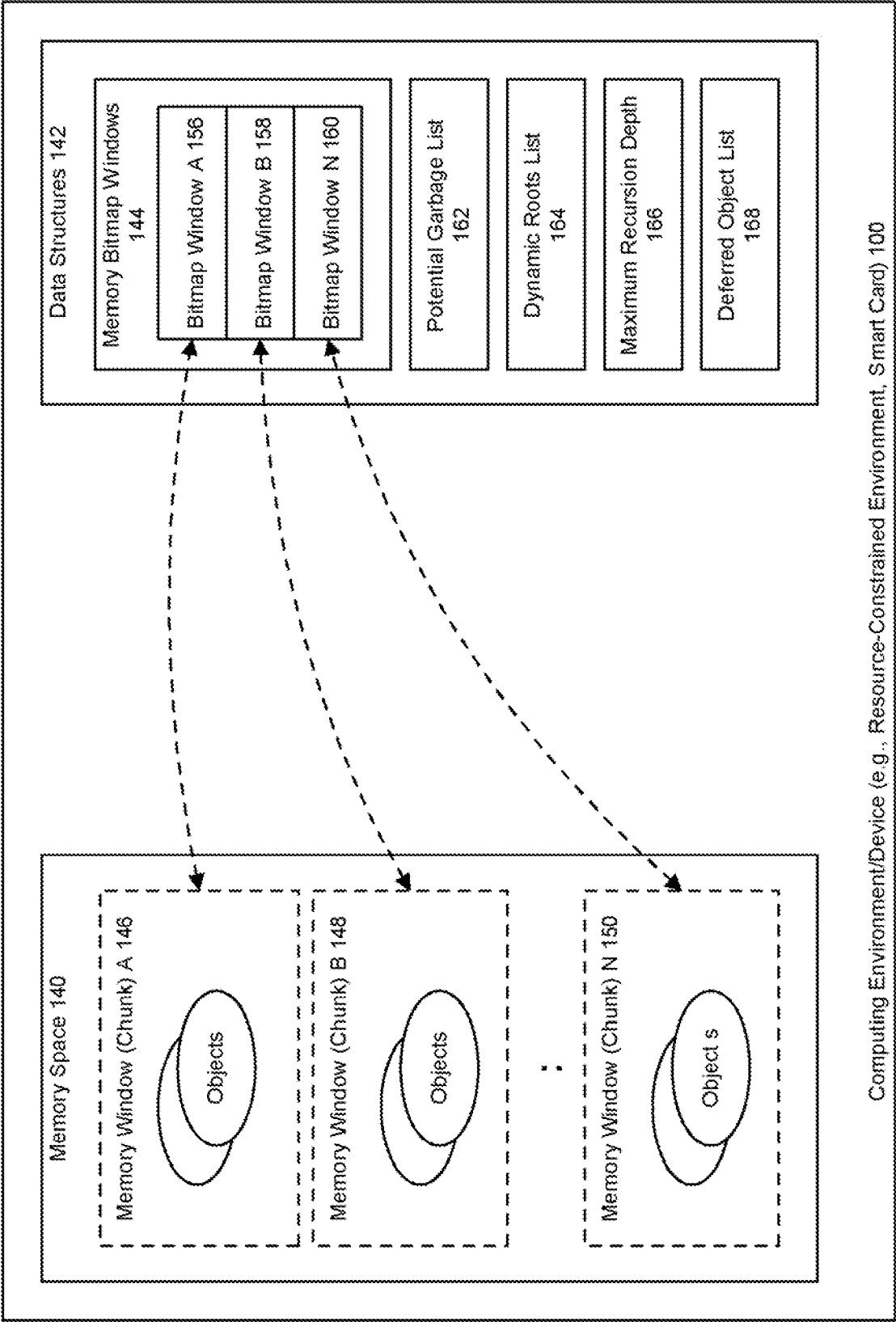
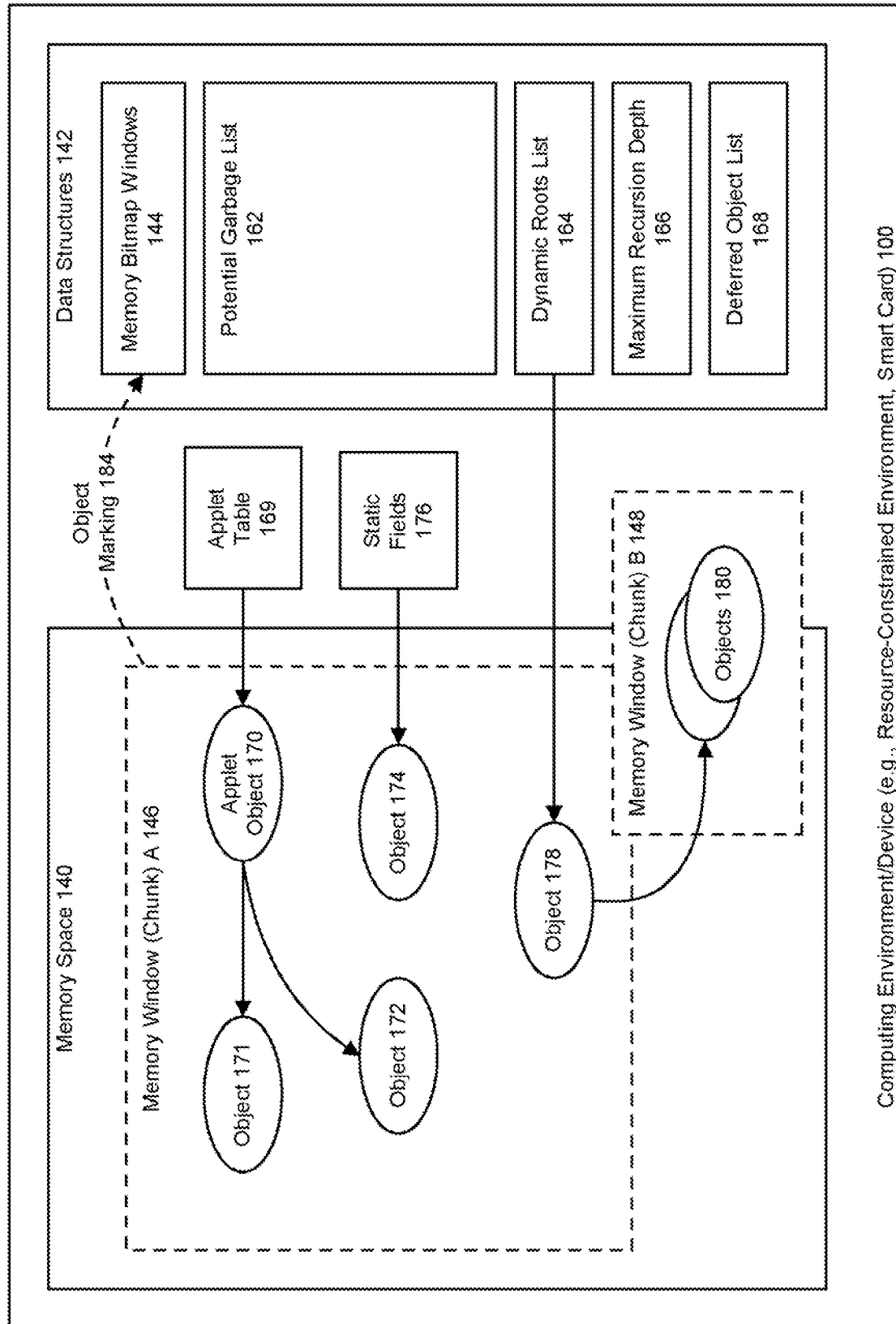
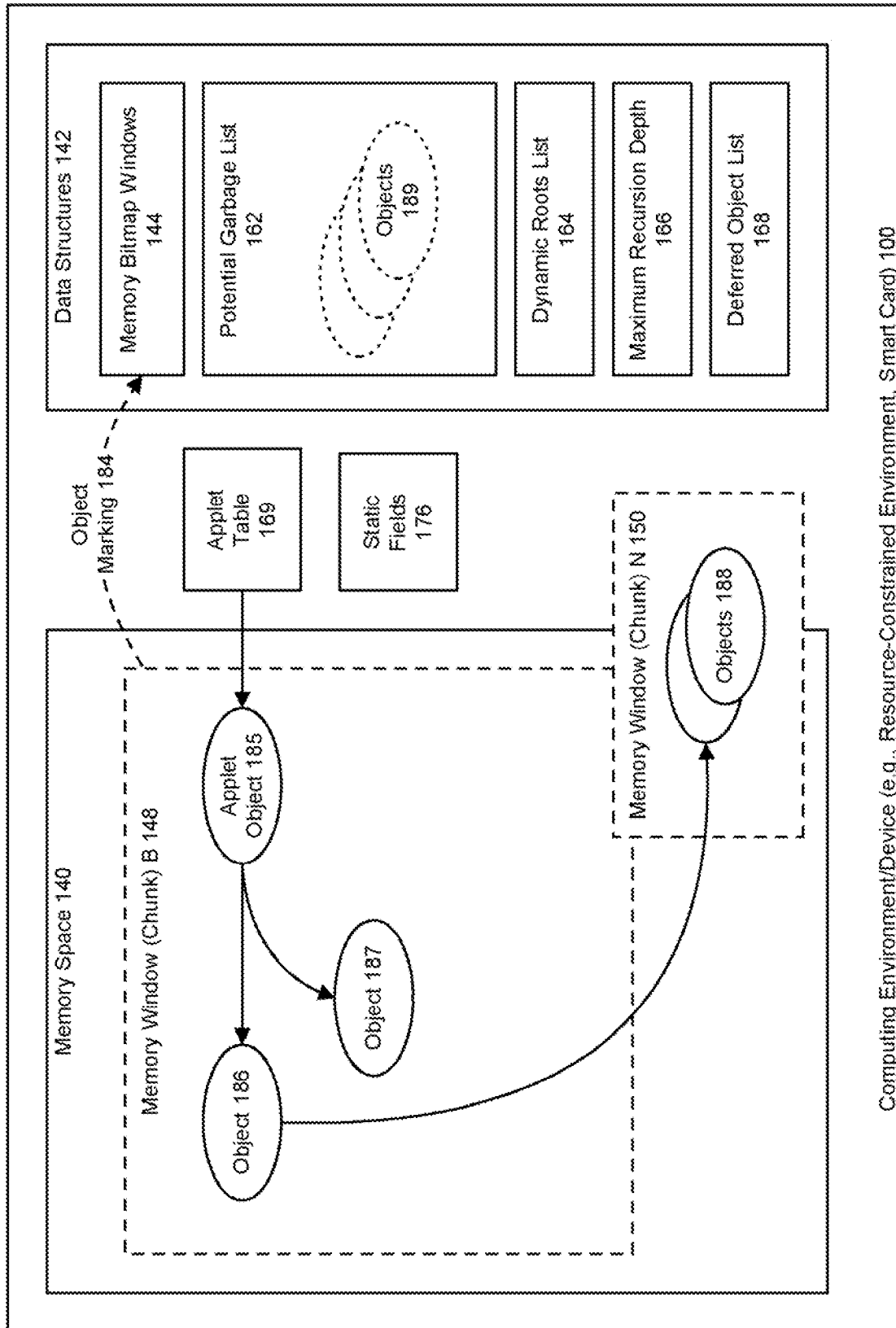


FIGURE 2



Computing Environment/Device (e.g., Resource-Constrained Environment, Smart Card) 100

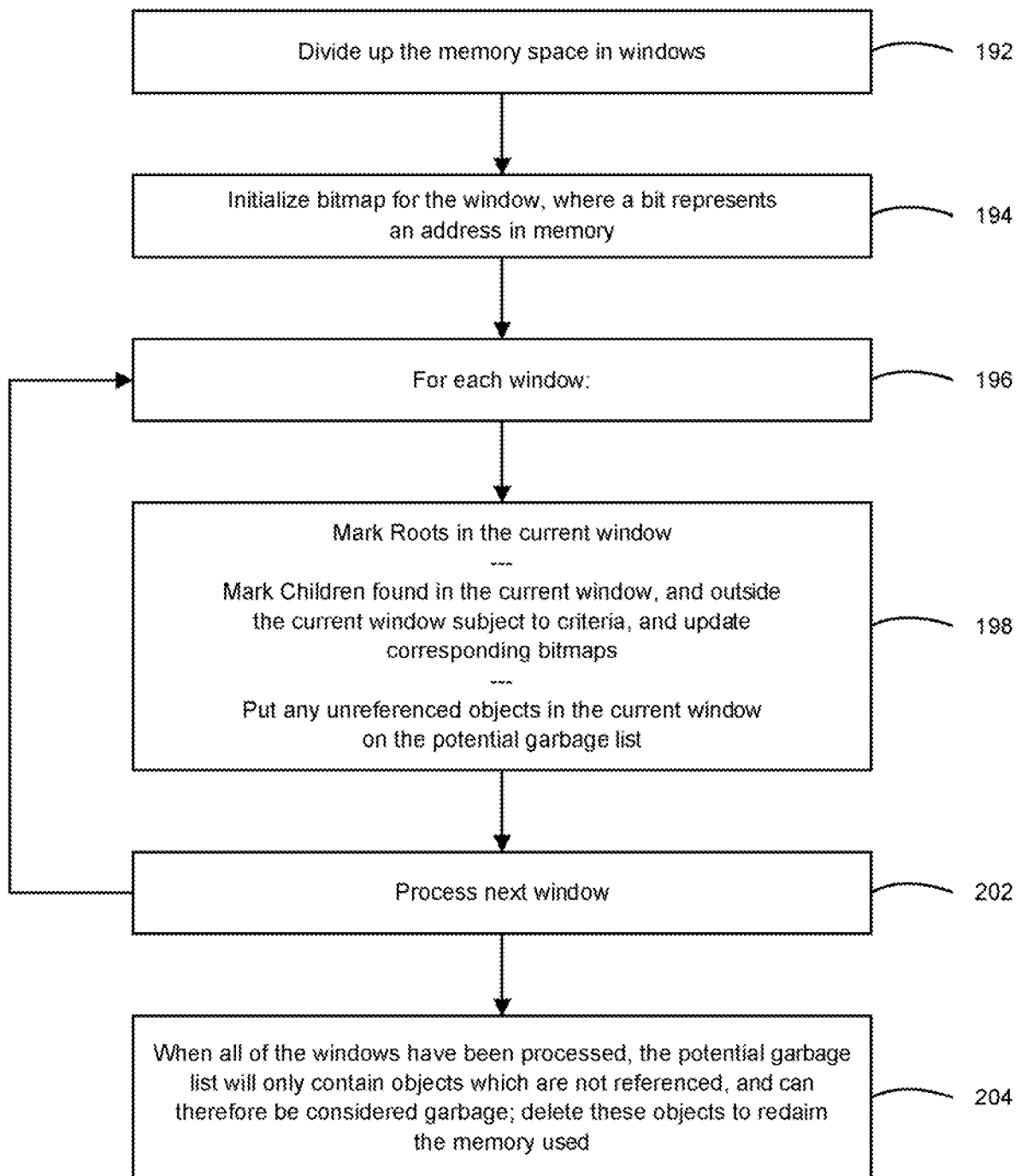
FIGURE 3



Computing Environment/Device (e.g., Resource-Constrained Environment, Smart Card) 100

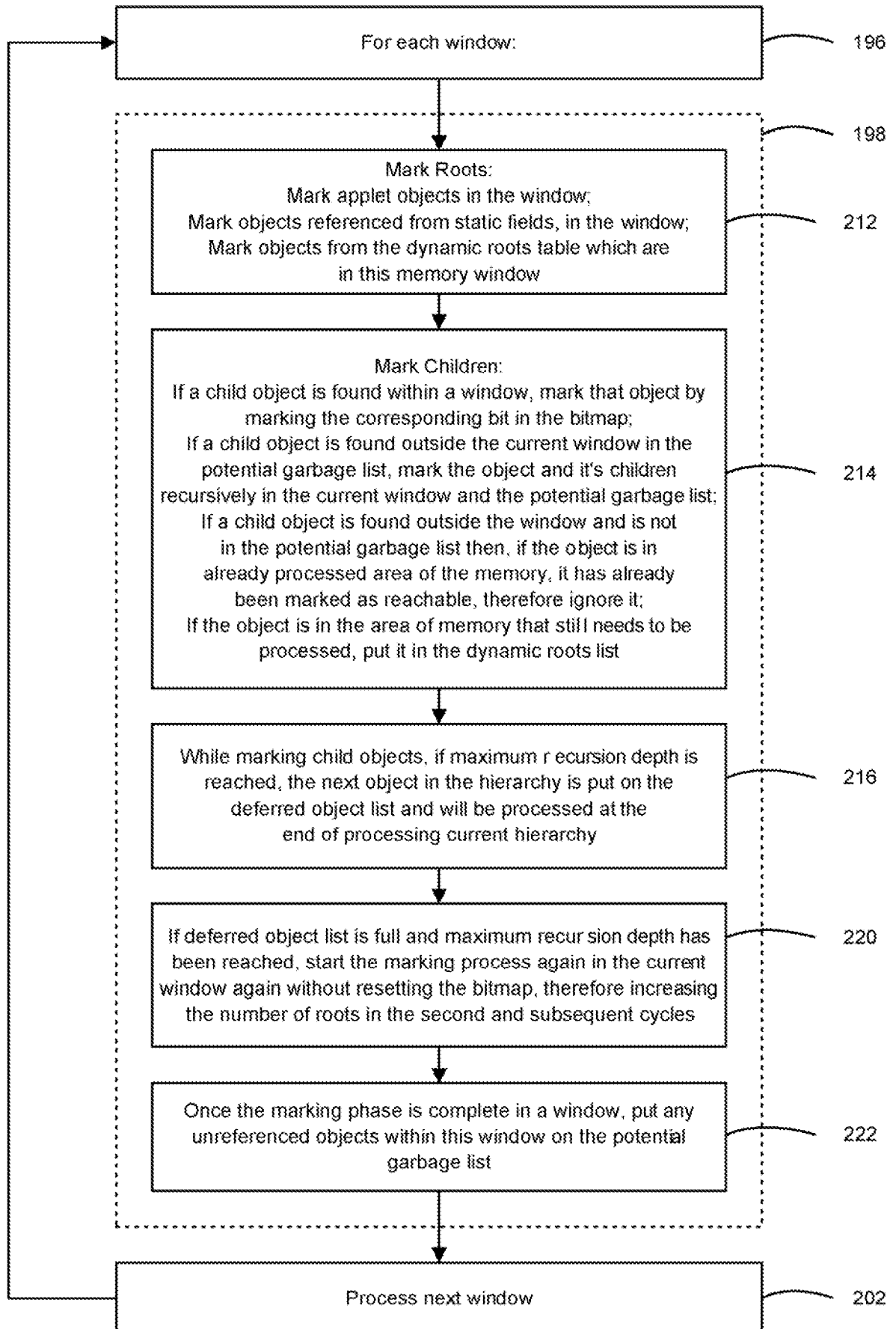
FIGURE 4

5/7



190

FIGURE 5



210

FIGURE 6

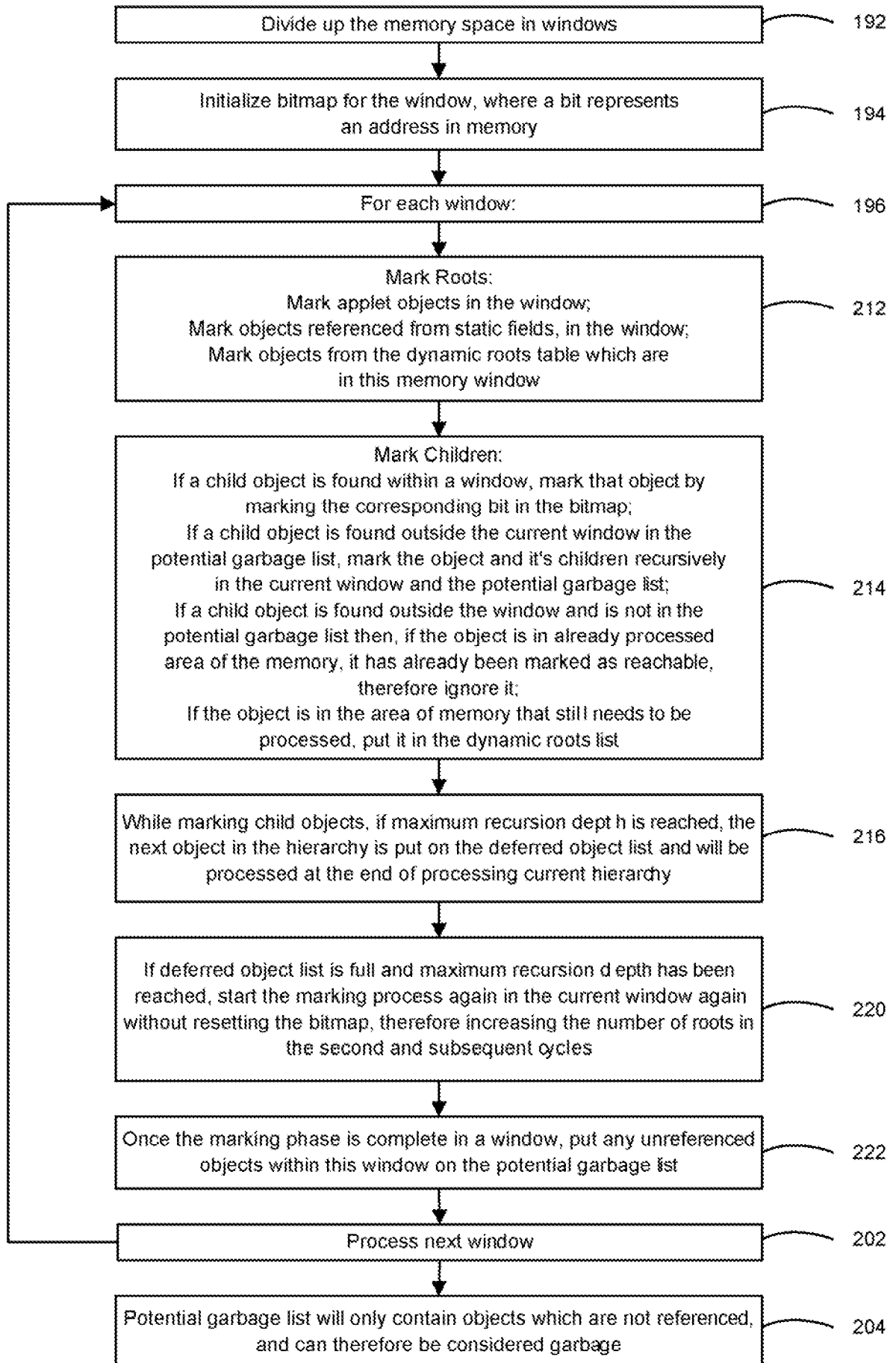


FIGURE 7

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2013/052780

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F12/02

ADD.

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 2003/221047 A1 (AHMED SAQIB J [US] AHMAD SAQIB J [US]) 27 November 2003 (2003-11-27)	1-3,5-7, 9-13
A	paragraphs [0028], [0040] - [0059]; figures 1,2,3	4,8,14
X	US 2007/174369 A1 (DETLEFS DAVID L [US]) 26 July 2007 (2007-07-26)	1,5,9-11
	paragraphs [0009] - [0012], [0022] - [0032]; figures 1-4	
X	US 2009/248766 A1 (GARTHWAITE ALEXANDER T [US] ET AL) 1 October 2009 (2009-10-01)	1-3,5-7, 9-13
	paragraphs [0013] - [0017], [0024] - [0027]	
	----- -/--	

☒ 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

29 November 2013

Date of mailing of the international search report

09/12/2013

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

Nielsen, Ole

INTERNATIONAL SEARCH REPORT

International application No

PCT/US2013/052780

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CHANG J M ET AL: "Active memory processor: a hardware garbage collector for real-time java embedded devices", IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE SERVICE CENTER, LOS ALAMITOS, CA, US, vol. 2, no. 2, 1 April 2003 (2003-04-01), pages 89-101, XP011099365, ISSN: 1536-1233, DOI: 10.1109/TMC.2003.1217230 page 89, right-hand column, line 20 - page 90, left-hand column, line 12 page 91, left-hand column, line 33 - page 93, left-hand column; figures 1,2,3,5 -----	1-14
A	US 6 453 403 B1 (CZAJKOWSKI GRZEGORZ [US]) 17 September 2002 (2002-09-17) column 3, line 14 - column 4, line 2 -----	1-14

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2013/052780

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2003221047	A1	27-11-2003	NONE
US 2007174369	A1	26-07-2007	NONE
US 2009248766	A1	01-10-2009	NONE
US 6453403	B1	17-09-2002	NONE