An object memory management solution for small devices with heterogeneous memories

Kevin Marquet, Gilles Grimaud

IRCICA/LIFL, CNRS UMR 8022, INRIA Futurs, POPS Research Group

Madrid, June 2007

Outline

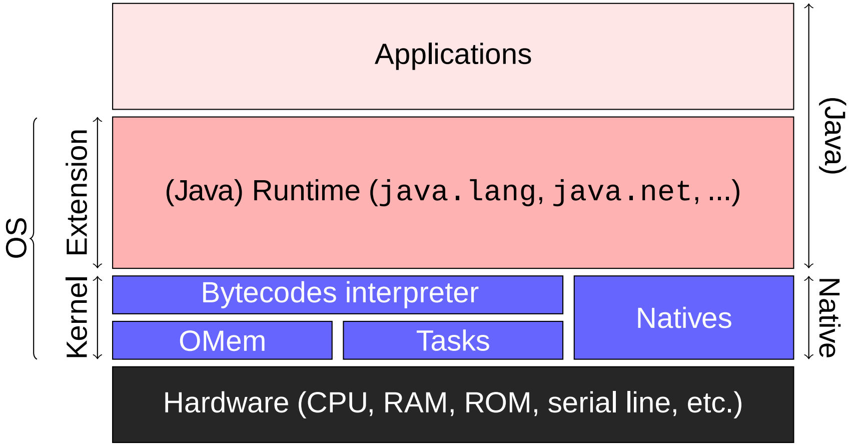
Small devices

Tiny devices (sensors, smart cards):

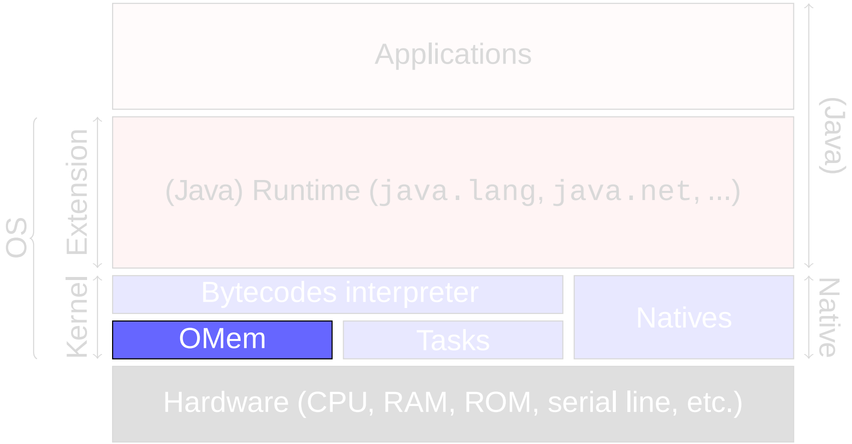
- Small amount of memory,
- Different kinds of memory: ROM, RAM, EEPROM, Flash;
- Different properties (security, efficiency, cost);
- Different proportions;
- Depends on the device.

Typically:
Few KBytes of RAM (Internal and External RAM)
Dozens of KBytes of EEPROM,
Hundreds of KBytes of ROM.

System: (Java) virtual machine


Applications
(Java) Runtime (java.lang, java.net, ...)
Bytecodes interpreter
OMem
Tasks
Natives
Hardware (CPU, RAM, ROM, serial line, etc.)
OS
Extension
Kernel
(Java)
Native

System: (Java) virtual machine


Applications
(Java) Runtime (java.lang, java.net, ...)
Bytecodes interpreter
OMem
Tasks
Natives
Hardware (CPU, RAM, ROM, serial line, etc.)
OS
Extension
Kernel
(Java)
Native

Memory management

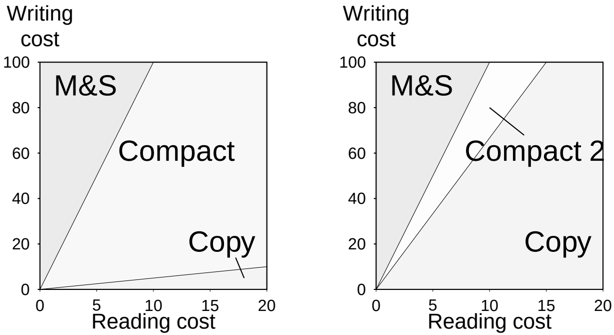
Goal: Automatic data reclamation (garbage collection).

Which garbage collection algorithm ?

Garbage collectors have different properties:

- execution speed;
- memory consumption;
- latency;
- ...

Example: performances



Objectives

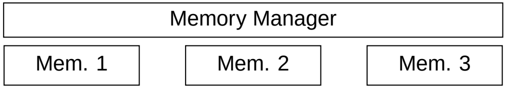
Objectives:

- design a memory management architecture...
- ...flexible (regarding to applications and hardware configuration)...
- ...allowing automatic data reclamation...
- of several memories with different properties

Existing works

Actually: memory manager dedicated to the uderlying hardware (Java Card...)

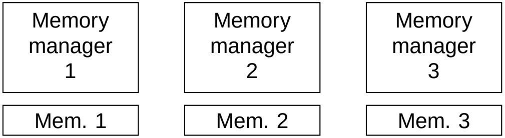
- Not portable;
- Not flexible;
- Suboptimal.



Outline

Our approach

Our approach: Assign one specific manager to each memory space



Problems:

- Marking live objects;
- Moving management;
- Reference management (update);
- **Interactions between managers**.

$\longrightarrow$ Make managers to cooperate.

Solution

All types of collectors can be inserted in a special framework:

- Prepare;
- Modify references; **[All memory spaces]**
- Collect;
- Update references. **[All memory spaces]**

Marking: special distributed algorithm (not detailed here)

Solution

All types of collectors can be inserted in a special framework:

- Prepare;
- Modify references; [**All memory spaces**]
- Collect;
- Update references. [**All memory spaces**]

Marking: special distributed algorithm (not detailed here)

Solution

All types of collectors can be inserted in a special framework:

- Prepare;
- Modify references; [**All memory spaces**]
- Collect;
- Update references. [**All memory spaces**]

Marking: special distributed algorithm (not detailed here)

Different algorithms

Semi-space:

Mark:	Traces all live objects. Each live object encountered is moved to the unused semi-space and its new address is stored at its old location. Only live objects are copied, thus performing the collection implicitly.
Update:	Scans the other semi-space and update each reference with the new location of the pointed object.

Different algorithms

Mark and compact:

Mark:	traverses all live objects and mark them as such.
Prepare:	scans memory, compute the new location of objects so that they are sequential from the bottom of the heap. The new address is stored into an extra word of the object.
Modify:	scans the memory and replace all references towards objects with the new location of this object.
Collect:	moves each live object to its new location. As for semi-space collectors, only live objects are moved, thereby collecting others implicitly.

Different algorithms

Mark and compact with dereferencing table:

Mark:	Traverses all live objects and mark them as such.
Prepare:	Scan memory and store each address of live objects into a references table (address increasingly ordered). *Version 1:* each entry of the table contains one word. *Version 2:* each entry of the table contains two words. The address is stored in the first word.
Modify:	(*Version 1*), dereferences all references in memory towards the table (search by dichotomy).
Collect:	move each live object to its new location. *Version 1:* this new location erases the old location in the table. *Version 2:* it is stored in the second word of the entry of the table.
Update:	updates each reference to point to the new location. *version 1:* this new location is the one pointed by the reference in the table. *version 2*, it is found in the second word of the entry.

Different algorithms

Mark and sweep:

Mark:	traverses all live objects and marks them as such.
Collect:	scans the memory to reclaim all unused objects.

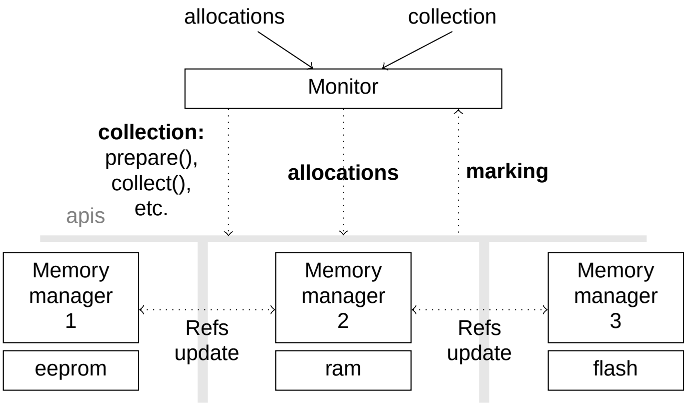
Different algorithms

2-generational copying (Sun's HotSpot):

Mark:	traces all live objects of the nursery and marks them as such;
Collect:	copies all live objects of the nursery in the bottom of the heap;
Update:	updates all references to match the new location of objects previously located in the nursery.

Outline

Architecture

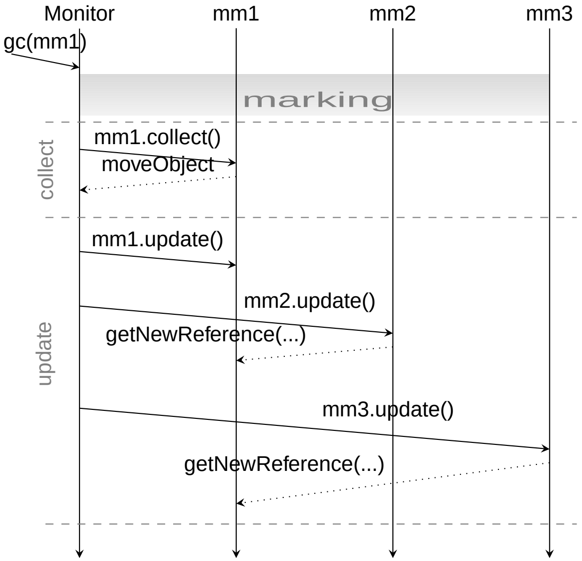


Implementation

- Configuration files;
- Generation of C structures, compiled with the source.

```
struct heap_variables_s {
        const unsigned int id;
        unsigned char* heapBase;
        const unsigned int memsize;
        void (*prepare) ( void );
        void (*modify) (struct heap_variables_s *);
        void (*collect) ( void );
        void (*update) (struct heap_variables_s *);
        struct java_lang_Object*
            (*getUpdatedAddress) (struct java_lang_Object *aref);
        struct java_lang_Object*
            (*getModifiedAddress) (struct java_lang_Object *aref);
        struct modifiable_heap_variables_s *modifiable_variables;
};
struct modifiable_heap_variables_s {
        unsigned char* toh;
        unsigned int nbObjects;
};
```

Example (semi-space)



Few results

Code size:

- tools: 8.5 kilobytes of compiled code;
- each manager: from 1 to 2.5 kilobytes of compiled code;

Data:

- Few bytes per manager (C structure)

Outline

Conclusion

- Architecture able to manage several memories;
- One memory manager per partition;
- Acceptable performances;
- Flexibility: Guillaume Salagnac (VERIMAG institute) inserted easily its own dedicated manager in the framework.

Perspectives

Work in progress:

- Optimizations (reference management);
- Implementations of new collectors.

Future work:

- Set up a gc algorithm for low access times memories;
- Dynamic modification of the memory management

Thank you for your attention...

Questions?