Debugging of Memory Safety Errors in Memory Managers

Karl Cronburg
@Tufts University
karl@cs.tufts.edu

The Problem

- I set my GC header bit to zero.

- Other runtime system code runs.

- My GC header bit is now set to one.

The Problem: Errors

- I set my GC header bit to zero.
- Other runtime system code runs.
- My GC header bit is now set to one.
- Memory safety **errors**
    - Dynamic semantics

Not The Problem: Bugs

- Statically possible **bugs**
- Infeasible without intervention
- Memory safety **bugs:**
    - Static semantics

Printf Debugging and Code Editors

```
72 public class JavaHeader implements JavaHeaderConstants {
73
74   protected static final int SCALAR_HEADER_SIZE = JAVA_HEADER_BYTES + OTHER_HEADER_BYTES;
75   protected static final int ARRAY_HEADER_SIZE = SCALAR_HEADER_SIZE + ARRAY_LENGTH_BYTES;
76
77   /** offset of object reference from the lowest memory word */
78   public static final int OBJECT_REF_OFFSET = ARRAY_HEADER_SIZE;  // from start to ref
79   protected static final Offset TIB_OFFSET = JAVA_HEADER_OFFSET;
80   protected static final Offset STATUS_OFFSET = TIB_OFFSET.plus(STATUS_BYTES);
81   public static final Offset AVAILABLE_BITS_OFFSET =
82       VM.LittleEndian ? (STATUS_OFFSET) : (STATUS_OFFSET.plus(STATUS_BYTES - 1));
```

- Mental math
- Static control flow graph
- Source code modification
- Mailing lists

Documentation to the rescue!

```
61  * |<- lo memory                                                hi memory ->|
62  *
63  *   SCALAR LAYOUT:
64  * |<---------- scalar header --------->|
65  * +----------+------------+------------+------+------+------+--------+
66  * | GCHeader | MiscHeader | JavaHeader | fld0 | fld1 | fldx | fldN-1 |
67  * +----------+------------+------------+------+------+------+--------+
68  *                         ^ JHOFF              ^objref
69  *                                              .
```

The Current Solution

- Weeks or more of manual debugging efforts
- False assumption: all program errors are the result of **bugs**
    - Unspecified behavior need not be a program bug
    - Documentation is a PL concern
- A program error is a developer ascribing correctness to behavior

PL Approach: What information do we need?

- What's the current configuration of memory?

- What code is running?

- What memory is that code allowed to access?

PL Approach: Every problem is a language

- What's the current configuration of memory?
- What code is running?
- What memory is that code allowed to access?
- M(VM) = { MemoryConfig }
    - Is (0x0000 … 0xFFFF) ϵ M(VM)?
- C(VM) = { (Code, MemoryType) }
    - Is (Code(ip), MemoryType(esi)) ϵ C(VM)?

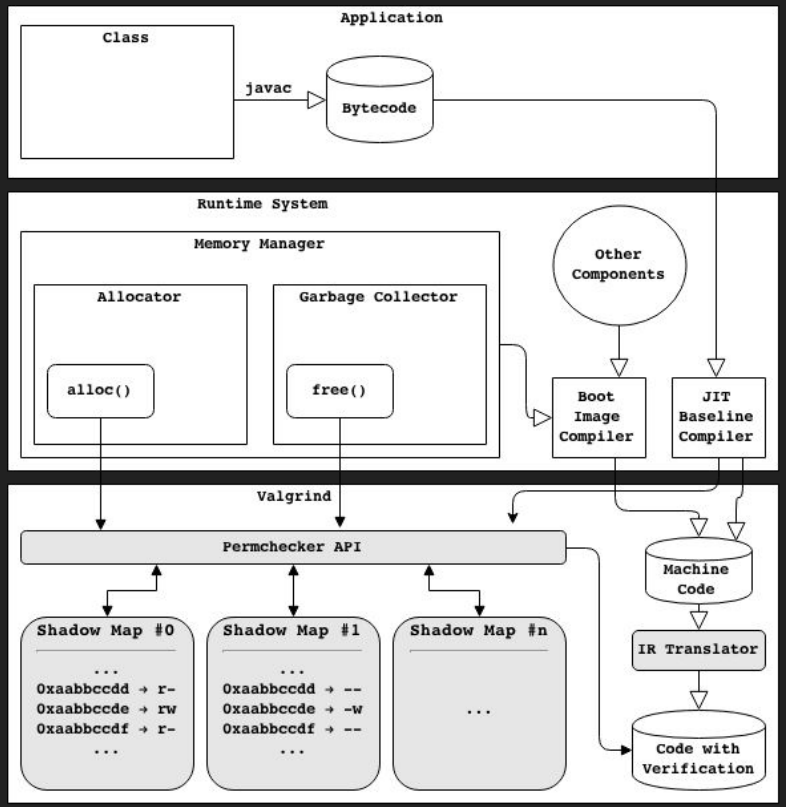
Prior Work: Dynamic Binary Instrumentation

- Specify M(VM) and C(VM) imperatively "at the same time"

- Half portable

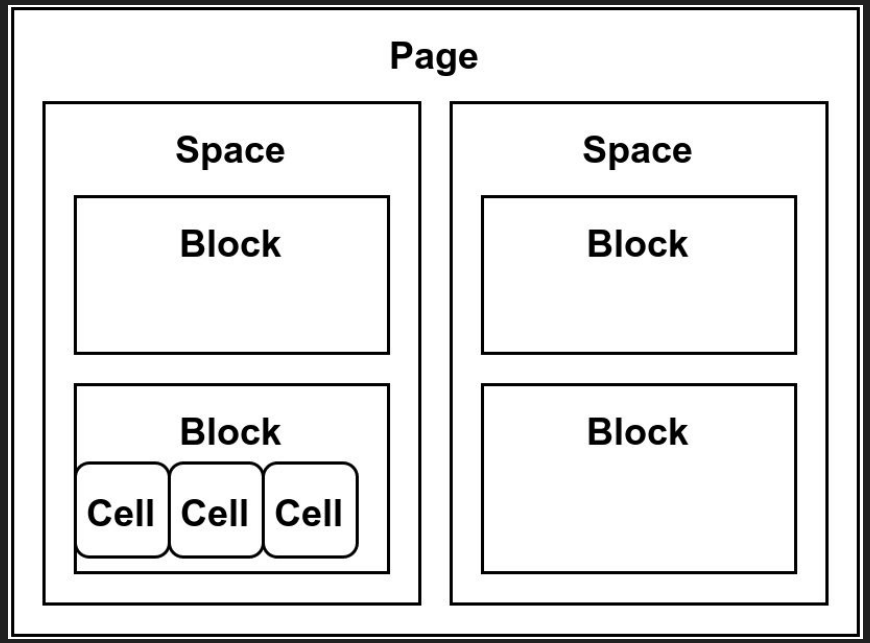
    - VM-specific alloc() and free()

    - Shared Permchecker API



Ongoing Work: Specify M(VM) Separately

- MM memory is layered
    - Page → Space → Block → Cell
- MM memory is bifurcated
    - (1) alloc, and (2) free
- MM memory is "owned"
    - Metadata at each layer
    - Free chunks at each layer



Ongoing Work: Specification Language (FLP)

- MM memory is layered
    - Page → Space → Block → Cell
- MM memory is bifurcated
    - (1) alloc, and (2) free
- MM memory is "owned"
    - Metadata at each layer
    - Free chunks at each layer

```
Heap |heapSz pages| @(1 page) heapSz ->
  ( mapped :
    ( # Region |2^22 bytes| @(2^22 bytes) ->
      ( Meta  |sz pages| ->
        ( next : Ptr Block
        , bmd  : 1 byte
        , csc  : 1 byte
        , iu   : 2 bytes
        , fl_meta : Ptr Block
        , pad1 bytes)
      , # Block |sz pages| @(1 page) ->
        ( Free ->
          ( prevBlock : Ptr Free
          // More fields go here
          , pad2 bytes)
        | Alloc ->
          ( Bump      ->
            ( objs : # Object
            , free : # bytes)
          | LargeObj -> # bytes))))
  , unmapped : # bytes)
```

Ongoing Work: **Shared** Language (FLP)

```
Heap m k ->
  # MegaBlock |2^m bytes| @(2^m bytes) ->
    // -------Megablock "Header"------------------
    ( Descrs |2^k bytes| ->
      ( padMB bytes
      // ++++++++++++++++++++++++++++++++++++++++
      , bds : n BlockDescr |2^d bytes| @(2^k bytes) ->
        ( start : Ptr Stg.Word
        , free  : Ptr Stg.Word
        , link  : Ptr BlockDescr
        , ( back   : Ptr BlockDescr
          | bitmap : Ptr Stg.Word
          | scan   : Ptr Stg.Word)
        , gen     : Ptr Generation
        , gen_no  : Stg.Word16
        , dest_no : Stg.Word16
        , node    : Stg.Word16
        , Flags |Stg.Word16| ->
          ( LARGE  | EVACUATED  | FREE
          | PINNED | MARKED     | KNOWN
          | EXEC   | FRAGMENTED | SWEPT
          | COMPACT )
        , n_blocks : Stg.Word32
        , padD bytes)))
      // ++++++++++++++++++++++++++++++++++++++++
    // -------Megablock payload------------------
    , blocks : n Block |2^k bytes| @(2^k bytes) ->
      ( closures : # Stg.Closures
      , free     : # bytes)
```

```
Heap |heapSz pages| @(1 page) heapSz ->
  ( mapped :
    ( # Region |2^22 bytes| @(2^22 bytes) ->
      ( Meta  |sz pages| ->
        ( next : Ptr Block
        , bmd  : 1 byte
        , csc  : 1 byte
        , iu   : 2 bytes
        , fl_meta : Ptr Block
        , pad1 bytes)
      , # Block |sz pages| @(1 page) ->
        ( Free ->
          ( prevBlock : Ptr Free
          // More fields go here
          , pad2 bytes)
        | Alloc ->
          ( Bump     ->
            ( objs : # Object
            , free : # bytes)
          | LargeObj -> # bytes))))
  , unmapped : # bytes)
```

Ongoing Work: Satisfying constraints

```
Heap m k ->
  # MegaBlock |2^m bytes| @(2^m bytes) ->
    // -------Megablock "Header"------------------
    ( Descrs |2^k bytes| ->
      ( padMB bytes
      // ++++++++++++++++++++++++++++++++++++++++
      , bds : n BlockDescr |2^d bytes| @(2^k bytes) ->
        ( start : Ptr Stg.Word
        , free  : Ptr Stg.Word
        , link  : Ptr BlockDescr
        , ( back   : Ptr BlockDescr
          | bitmap : Ptr Stg.Word
          | scan   : Ptr Stg.Word)
        , gen     : Ptr Generation
        , gen_no  : Stg.Word16
        , dest_no : Stg.Word16
        , node    : Stg.Word16
        , Flags |Stg.Word16| ->
          ( LARGE  | EVACUATED  | FREE
          | PINNED | MARKED     | KNOWN
          | EXEC   | FRAGMENTED | SWEPT
          | COMPACT )
        , n_blocks : Stg.Word32
        , padD bytes)))
      // ++++++++++++++++++++++++++++++++++++++++
    // -------Megablock payload-------------------
    , blocks : n Block |2^k bytes| @(2^k bytes) ->
      ( closures : # Stg.Closures
      , free     : # bytes)
```

$$
\begin{aligned}
2^d &= \mathrm{sz(BlockDescr)} \\
&= 5 \times \mathrm{sz(Ptr)} + 4 \times \mathrm{sz(Word16)} + \\
&= \mathrm{sz(Word32)} + \mathrm{padD\ bytes} \\
&= 5\ \mathrm{words} + (12 + \mathrm{padD})\ \mathrm{bytes} \\
d_{32} &= \log_2(32 + \mathrm{padD}) = 5\ [\ \mathrm{padD} = 0 \quad \mathrm{bytes}] \\
d_{64} &= \log_2(52 + \mathrm{padD}) = 6\ [\ \mathrm{padD} = 12\ \mathrm{bytes}]
\end{aligned}
$$

- General constraint solving

Ongoing Work: FLP Toolchain

- Parser implemented in Haskell

- Currently implementing transpiler

    - Tile FLP "layers" onto C structs or C++ classes

    - Examples: Hotspot, Jikes RVM, GHC

- Pin and Valgrind DBI backend

- **Take-away: a language for specifying memory layout capable of describing a wide variety of VM implementations** (for Java, Haskell, Javascript, …)

Future Work: Specifying C(VM)

- Integrate with host language or a framework, options include:
    - Target C++ class methods (portability: lowest common denominator)
    - Target Rust (safety: stronger guarantees)
    - Integrate with LLVM (automation)
    - Other suggestions?