CPS506 - Comparative Programming Languages
Implementation

Dr. Dave Mason
Department of Computer Science
Ryerson University

Why Managed Languages?

- productivity - focus on the problem
- expressive languages - functional, OO, declarative
- safety - hard to get low-level details right

What Managed Languages?

- memory management - usually garbage collected
- higher-level abstractions
- often interpreted/JIT
- often VM - JavaVM and CLR are most well known

Which Managed Languages?

- OO - Smalltalk, Java, Python, Ruby, C#, Scala, Javascript
- functional - Elixir/Erlang, Haskell, SML, Ocaml, Racket/Scheme/LISP/Clojure
- array - APL/J, R, MATLAB, Maple
- logic/declarative - Prolog
- procedural/systems - Go, Nim, Lua
- easier question - what's not? - C, C++, Rust, Zig, Odin, Jai

Modern Execution Structure?

- most machine architecture: PC, SP, other registers, memory

Dynamically Typed Languages

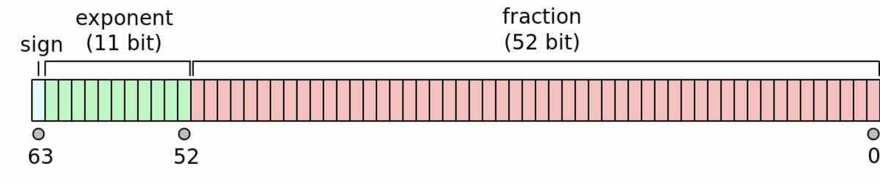
- values are typed
- some form of polymorphism - parametric or OO
- means everything must be same size - ideally a register
- could heap-allocate everything - really bad for integers
- otherwise, need to tag values with type
- some hardware has had tags - SPARC, B7700

Conventional tag

- modern architectures are byte-addressable
- heap objects will always be aligned - say 8-byte boundaries
- can put tag in low bits, have integers shifted
- keep floating point values boxed

IEEE-FP tag

- modern processors have 64-bit integers, 64-bit pointers, and 64-bit IEEE floats
- IEEE floats have many Nan values - exp all 1s - 2^{53} Nan values



- several ways to do NaN tagging/encoding
  - you can choose integers, pointers, or doubles to be naturally encoded
  - all the others be encoded with some shifting/adding
  - while integers and pointers are probably more common in most Smalltalk images
  - leaving doubles as naturally encoded means that FPU, vector instructions and/or GPUs can act directly on memory

IEEE-FP tag...

AST Smalltalk uses the following encoding based on the **S**ign+**E**xponent and **F**raction bits:

S+E	F	F	F	Type
0000	0000	0000	0000	double +0
0000-7FEF	xxxx	xxxx	xxxx	double (positive)
7FF0	0000	0000	0000	+inf
7FF0-F	xxxx	xxxx	xxxx	NaN (unused)
8000	0000	0000	0000	double -0
8000-FFEF	xxxx	xxxx	xxxx	double (negative)
FFF0	0000	0000	0000	-inf
FFF2-F	xxxx	xxxx	xxxt	tagged literals
FFF2/3	xxxx	xxxx	xxxx	heap object
FFF4	0000	0001	0000	False
FFF6	0000	0010	0001	True
FFF8	1000	0000	0000	UndefinedObject
FFFA/B	xxxx	xxxx	xxxx	Symbol
FFFC/D	xxxx	xxxx	xxxx	Character
FFFE/F	xxxx	xxxx	xxxx	SmallInteger

Heaps

- sequential allocation - very cheap but run out of space eventually
  - can work if we can compact out the freed memory eventually
- block allocation - data structure to remember freed memory
  - many algorithms
  - external fragmentation - free space that can't be allocated
  - internal fragmentation - allocations larger than the object
- page allocation - "pages" of uniform types
  - external fragmentation - just at end of "page"
  - internal fragmentation - should be none apart from alignment

Heap allocation

- manual allocation - explicit malloc/free - very error prone
- reference counting - problem with cyclic structures, cascading-free
- non-moving collection
- compacting collector - enables sequential allocation
- copying collector - enables sequential allocation
- generational collection

Mark+Sweep Garbage Collection

- 2 phases
- mark phase - go from roots to find all accessible data
- go through all object putting inaccessible into “free list”
- can be written to be mostly parallel
- can be conservative
- does not support sequential allocation
- significant fragmentation can exist
- allocation can be slow - finding appropriate free space

Compacting Garbage Collection

- similar to mark+sweep with extra overhead to manage compacting
- sequential collector
- consolidate free space to prevent fragmentation and support sequential allocation

Copying Garbage Collection

- consolidate free space to prevent fragmentation and support sequential allocation
- sequential collector
- from roots collect all live objects into new area
- leaving "forwarding pointers" behind
- make the new space the current space
- only touches live data

Generational Collection

- can be best of all worlds
- per-thread copying collector - nursery + intermediate
- shared mark+sweep collector - can be parallel