Design Principles for a High-Performance Smalltalk

Dave Mason
Toronto Metropolitan Universityi

Toronto Metropolitan University

Design Principles

- Large memories
- 64-bit and IEEE-768
- Multi-core and threading
- Fast execution

Design Principles

- Large memories
- 64-bit and IEEE-768
- Multi-core and threading
- Fast execution

Design Principles

- Large memories
- 64-bit and IEEE-768
- Multi-core and threading
- Fast execution

Design Principles

- Large memories
- 64-bit and IEEE-768
- Multi-core and threading
- Fast execution

Zag Smalltalk

- from-scratch implementation
- low-level is implemented in Zig
- goal is to support existing OpenSmalltalk systems
- don't want to rewrite userland!

Zag Smalltalk

- from-scratch implementation
- low-level is implemented in Zig
- goal is to support existing OpenSmalltalk systems
- don’t want to rewrite userland!

Zag Smalltalk

- from-scratch implementation
- low-level is implemented in Zig
- goal is to support existing OpenSmalltalk systems
- don’t want to rewrite userland!

Zag Smalltalk

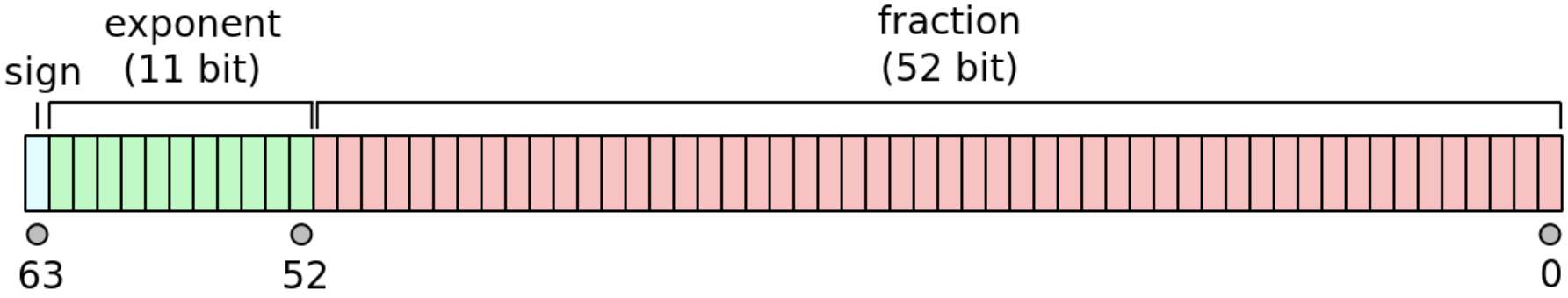
- from-scratch implementation
- low-level is implemented in Zig
- goal is to support existing OpenSmalltalk systems
- don’t want to rewrite userland!

Immediate Values

- 64 bit
- NaN-boxing
- double-floats, 51-bit SmallInteger, Booleans, nil, Unicode characters, Symbols
- room for instances of any type with single 32-bit value

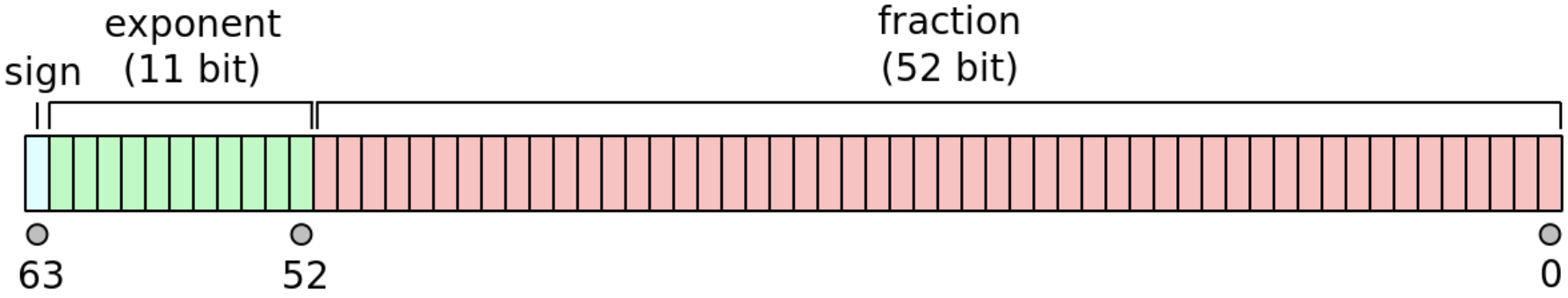
Immediate Values

- 64 bit
- NaN-boxing
- double-floats, 51-bit SmallInteger, Booleans, nil, Unicode characters, Symbols
- room for instances of any type with single 32-bit value



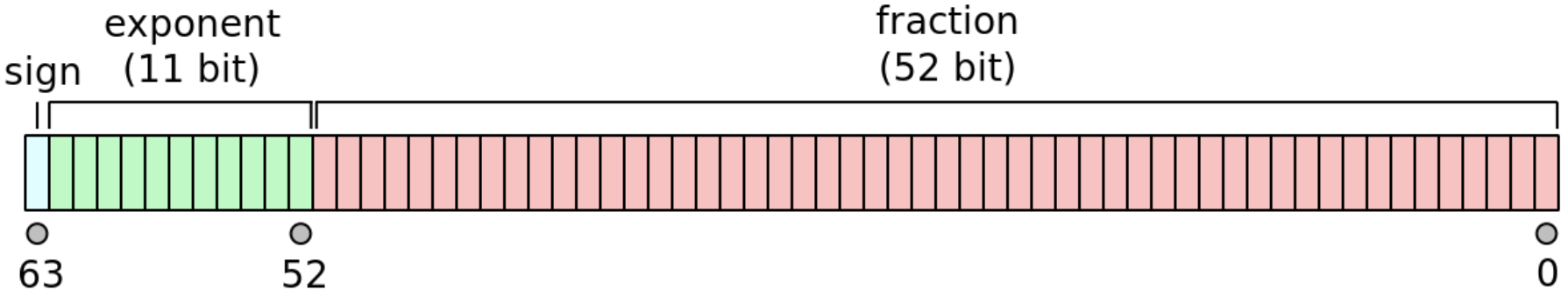
Immediate Values

- 64 bit
- NaN-boxing
- double-floats, 51-bit SmallInteger, Booleans, nil, Unicode characters, Symbols
- room for instances of any type with single 32-bit value



Immediate Values

- 64 bit
- NaN-boxing
- double-floats, 51-bit SmallInteger, Booleans, nil, Unicode characters, Symbols
- room for instances of any type with single 32-bit value



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
FFF0-5	xxxx	xxxx	xxxx	NaN (currently unused)
FFF6	xxxx	xxxx	xxxx	heap object
FFF7	0001	xxxx	xxxx	reserved (tag = Object)
FFF7	0002	xxxx	xxxx	reserved (tag = SmallInteger)
FFF7	0003	xxxx	xxxx	reserved (tag = Double)
FFF7	0004	0001	0000	False
FFF7	0005	0010	0001	True
FFF7	0006	0100	0002	UndefinedObject
FFF7	0007	aaxx	xxxx	Symbol
FFF7	0008	00xx	xxxx	Character
FFF8-F	xxxx	xxxx	xxxx	SmallInteger
FFF8	0000	0000	0000	SmallInteger minVal
FFFC	0000	0000	0000	SmallInteger 0
FFFF	FFFF	FFFF	FFFF	SmallInteger maxVal

Multi-core support

- only way to speed up applications
- minimal blocking
- computational/mutator threads - typically 1 per core
- I/O threads - one per open “file”
- global collector thread

Multi-core support

- only way to speed up applications
- minimal blocking
- computational/mutator threads - typically 1 per core
- I/O threads - one per open “file”
- global collector thread

Multi-core support

- only way to speed up applications
- minimal blocking
- computational/mutator threads - typically 1 per core
- I/O threads - one per open “file”
- global collector thread

Multi-core support

- only way to speed up applications
- minimal blocking
- computational/mutator threads - typically 1 per core
- I/O threads - one per open “file”
- global collector thread

Multi-core support

- only way to speed up applications
- minimal blocking
- computational/mutator threads - typically 1 per core
- I/O threads - one per open “file”
- global collector thread

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
- global collector thread
- ...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
- global collector thread
- ...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
- global collector thread
- ...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
- global collector thread
- ...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
- ...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, interned tables)
  - ...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

...

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

Memory management

- mutator threads
  - copying collector
  - private nursery (includes stack)
  - 2 teen arenas - n copies before promotion
  - when prompted, finds next 100 refs to global stack, then blocks, repeat
  - then can proceed
- I/O threads
  - maintains list of current shared buffers while I/O blocked
- global collector thread
  - non-moving mark/sweep arena
  - periodically does mark
  - marks known shared structures (class table, symbol table, dispatch tables)
  - asks mutators for global roots
  - processes them until all roots have been found
  - then can proceed to sweep

...

... Memory management

- global collector for non-moving mark-&-sweep
- uses Fibonacci heap (similar to Mist)
- large objects (e.g. 16Kib) have separately mapped pages (allows mmap of large files) to minimize memory creep

... Memory management

- global collector for non-moving mark-&-sweep
- uses Fibonacci heap (similar to Mist)
- large objects (e.g. 16Kib) have separately mapped pages (allows mmap of large files) to minimize memory creep

... Memory management

- global collector for non-moving mark-&-sweep
- uses Fibonacci heap (similar to Mist)
- large objects (e.g. 16Kib) have separately mapped pages (allows mmap of large files) to minimize memory creep

Heap objects

- header

Bits	What	Characteristics
12	length	number of long-words beyond the header
4	age	0 - nursery, 1-7 teen, 8+ global
8	format	
24	identityHash	
16	classIndex	LSB

- recognize pointer-free instance variables and arrays separately
- length of 4095 - forwarding pointer - copying, `become:`, promoted
- format is somewhat similar to SPUR encoding - various-sized non-object arrays
- strings stored in UTF-8

Heap objects

- header

Bits	What	Characteristics
12	length	number of long-words beyond the header
4	age	0 - nursery, 1-7 teen, 8+ global
8	format	
24	identityHash	
16	classIndex	LSB

- recognize pointer-free instance variables and arrays separately
- length of 4095 - forwarding pointer - copying, `become:`, promoted
- format is somewhat similar to SPUR encoding - various-sized non-object arrays
- strings stored in UTF-8

Heap objects

- header

Bits	What	Characteristics
12	length	number of long-words beyond the header
4	age	0 - nursery, 1-7 teen, 8+ global
8	format	
24	identityHash	
16	classIndex	LSB

- recognize pointer-free instance variables and arrays separately
- length of 4095 - forwarding pointer - copying, `become:`, promoted
- format is somewhat similar to SPUR encoding - various-sized non-object arrays
- strings stored in UTF-8

Heap objects

- header

Bits	What	Characteristics
12	length	number of long-words beyond the header
4	age	0 - nursery, 1-7 teen, 8+ global
8	format	
24	identityHash	
16	classIndex	LSB

- recognize pointer-free instance variables and arrays separately
- length of 4095 - forwarding pointer - copying, `become:`, promoted
- format is somewhat similar to SPUR encoding - various-sized non-object arrays
- strings stored in UTF-8

Heap objects

- header

Bits	What	Characteristics
12	length	number of long-words beyond the header
4	age	0 - nursery, 1-7 teen, 8+ global
8	format	
24	identityHash	
16	classIndex	LSB

- recognize pointer-free instance variables and arrays separately
- length of 4095 - forwarding pointer - copying, `become:`, promoted
- format is somewhat similar to SPUR encoding - various-sized non-object arrays
- strings stored in UTF-8

Unified dispatch

- single level of hashing for method dispatch
- each class dispatch table has entry for every method it has been sent - regardless of place in hierarchy
- near-perfect hash using Φ hashing
- standard SPUR/OpenVM optimizations don't work well in multi-core environments

Unified dispatch

- single level of hashing for method dispatch
- each class dispatch table has entry for every method it has been sent - regardless of place in hierarchy
- near-perfect hash using Φ hashing
- standard SPUR/OpenVM optimizations don't work well in multi-core environments

Unified dispatch

- single level of hashing for method dispatch
- each class dispatch table has entry for every method it has been sent - regardless of place in hierarchy
- near-perfect hash using Φ hashing
- standard SPUR/OpenVM optimizations don't work well in multi-core environments

Unified dispatch

- single level of hashing for method dispatch
- each class dispatch table has entry for every method it has been sent - regardless of place in hierarchy
- near-perfect hash using Φ hashing
- standard SPUR/OpenVM optimizations don't work well in multi-core environments

High performance Inlining

- references to `self` / `super` code are inlined
- methods with small number of implementations are inlined - rather than heuristic
- prevents creation of many blocks
- provides large compilation units for optimization

High performance Inlining

- references to `self` / `super` code are inlined
- methods with small number of implementations are inlined - rather than heuristic
- prevents creation of many blocks
- provides large compilation units for optimization

High performance Inlining

- references to `self` / `super` code are inlined
- methods with small number of implementations are inlined - rather than heuristic
- prevents creation of many blocks
- provides large compilation units for optimization

High performance Inlining

- references to `self` / `super` code are inlined
- methods with small number of implementations are inlined - rather than heuristic
- prevents creation of many blocks
- provides large compilation units for optimization

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
- JIT

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
- JIT

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
- JIT

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
- JIT

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation
- JIT

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation
- JIT

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation
- JIT
  - future

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation
- JIT
  - future
  - LLVM jitter

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation
- JIT
  - future
  - LLVM jitter

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation
- JIT
  - future
  - LLVM jitter

Code Generation

- no interpreter, 3 code generation models
- threaded-execution
  - method is sequence of Zig function addresses
  - uses Zig tail-call-elimination - passes pc, sp, hp, thread, context
  - primitives and control implementations
- stand-alone generator
  - generates Zig code for methods
  - depends on Zig inlining and excellent code generation
- JIT
  - future
  - LLVM jitter

Conclusions

- while the intent of this paper is to provide design principles, also described some implementation
- this is preliminary work, so some open questions
- many experiments to run to validate my intuitions

Conclusions

- while the intent of this paper is to provide design principles, also described some implementation
- this is preliminary work, so some open questions
- many experiments to run to validate my intuitions

Conclusions

- while the intent of this paper is to provide design principles, also described some implementation
- this is preliminary work, so some open questions
- many experiments to run to validate my intuitions

Questions?

@DrDaveMason dmason@ryerson.ca

https://github.com/dvmason/Zag-Smalltalk