Babel

An untyped, stack-based HLL

Clayton Bauman, 2013

The background

The preliminaries

- General-purpose high-level language
- Stack-based (postfix order) :

```
var 1 2 + set
```

versus:

```
var = 1 + 2
```

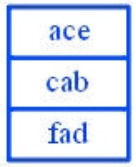
- Not pure – many operators can have side-effects
- Untyped

The data-structure: bstruct

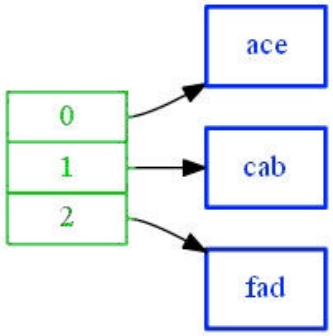
- Underlying 'container' for all data in a Babel program
- Strings, integers, unsigned, floats are all stored as `values` in `leaf-arrays`. No `pointers` can be stored in a `leaf-array`.
- Ordinary `pointers` are stored in `interior-arrays` – every `pointer` in an `interior-array` must be initialized and valid. A valid pointer is a pointer that points at the first `entry` of a `leaf-array`, `interior-array` or `tagged-pointer`. No `values` can be stored in an `interior-array`.
- A `tagged-pointer` is a pointer stored with an associated 128-bit `hash`-value called a `tag`
- A `bstruct` has a **single** root pointer

The data-structure: cont'd

```
(val 0xace 0xcab 0xfad)
```



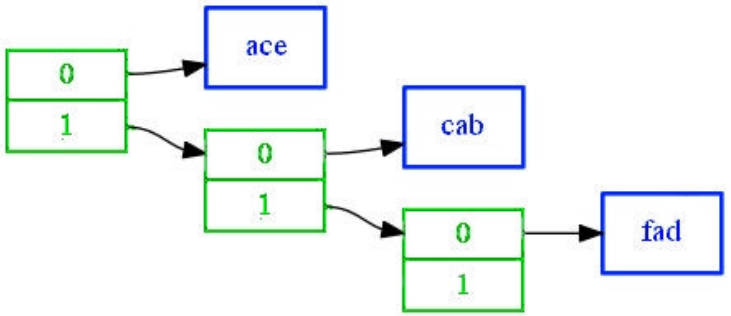

```
(ptr 0xace 0xcab 0xfad)
```



The data-structure: cont'd

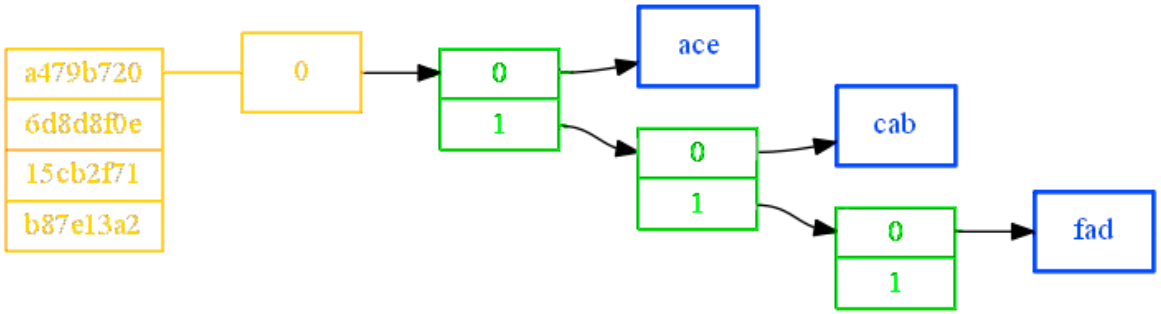
```
(ptr 0xace (ptr 0xcab (ptr 0xcabe nil)))
```

```
(list 0xace 0xcab 0xfad)
```



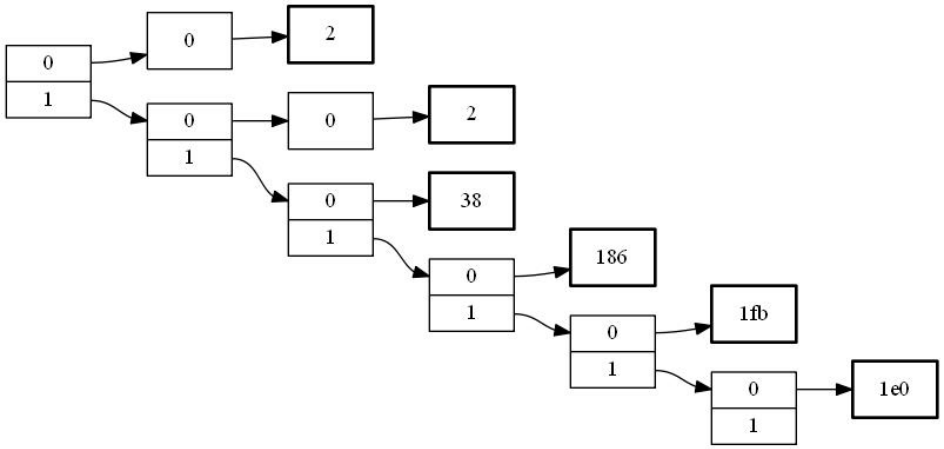
The data-structure: cont'd

```
(tag 'foo' (list 0xace 0xcab 0xfad))
```



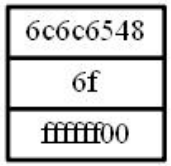
The data-structure: cont'd

```
(code 2 2 + %d cr <<)
```



The data-structure: cont'd

```
(val "Hello")
```




```
(val 0x6c6c6548 0x6f 0xffffff00)
```

The virtual machine: BVM

- Stored as a `bstruct`
- Three stacks: down-stack (`dstack`) up-stack (`ustack`) and the managed-stack (`rstack`, h/t to Chuck Moore)
- Code-list is a linked list containing data operands, code (operators) or both – pointed to by `code_ptr`
- Every data-operand is pushed on the `dstack`
- A kind of stack-nesting uses the `ustack` (`up`/`down` operators)
- Conditional, looping and other operators that manipulate `code_ptr` utilize the `rstack`
- Symbol table (`sym_table`) contains all symbols

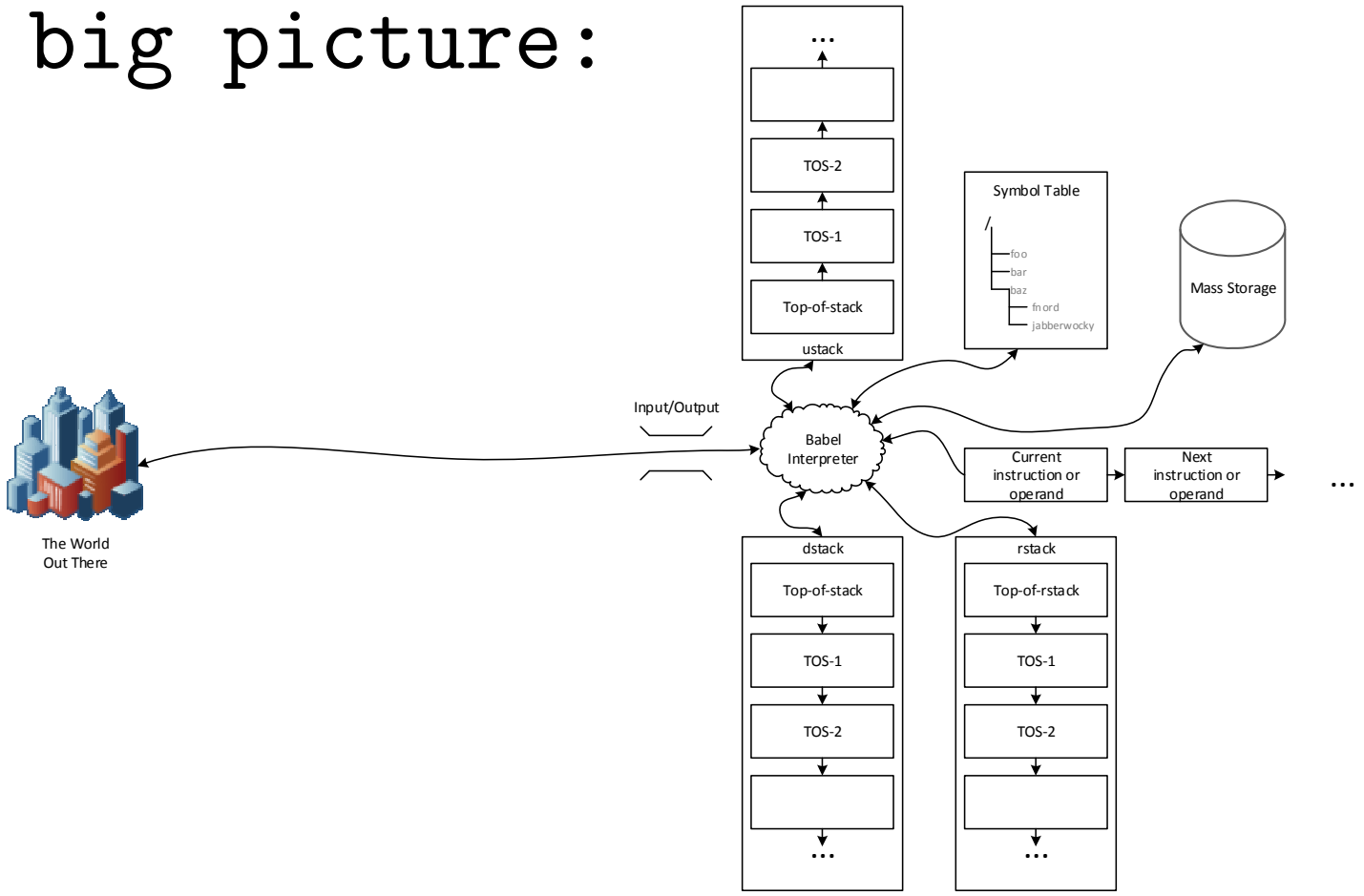
The runtime: Babel interpreter

- Actually has a small boostrap - written in Babel - that loads your Babel program for you (see `src/construct.sp`, `src/root.sp` and `src/construct.sp.c`)
- Maintains a cached copy of the most important pointers to avoid needless traversal of the BVM
- Jump-table permits O(1) vectoring to built-in operator functions
- Command-line capture (`argv`), STDIN, STDOUT, and environment-variable capture

The reference implementation: babel

- Implemented in C
- Perl wrapper front-end generates native Babel binary (.bbl) from Sparse syntax (sparse.pl). Uses s-expressions and a minimalist syntax for immediate values
- Optimization will begin after Babel 1.0

The big picture:



The future: Babel 1.0

- Built-in parser for Bipedal syntax
- Wide selection of crypto primitives (libtomcrypt)
- Linked with:
  - libcurl
  - bzip2
  - Graphviz
- Compile on gcc, mingw
- Cross-platform compatible on Windows and *nix.
- Additional built-in operators

The demo…