Stak Scheme: The tiny R7RS-small implementation

Scheme Workshop 2025

Yota Toyama

Background

- Ribbit Scheme, the tiny R4RS implementation
  - R4RS REPL in 7 KB
  - **Compact**, **portable**, fast, and simple
- Two separate components
  - Compiler written in Scheme
  - Virtual Machine (**Ribbit VM**, or just **RVM**) written in x86-64 assembly, C, Javascript, Bash, ...

Can we implement the entire R7RS-small standard on RVM? 🤔

Yes, we can! 😏

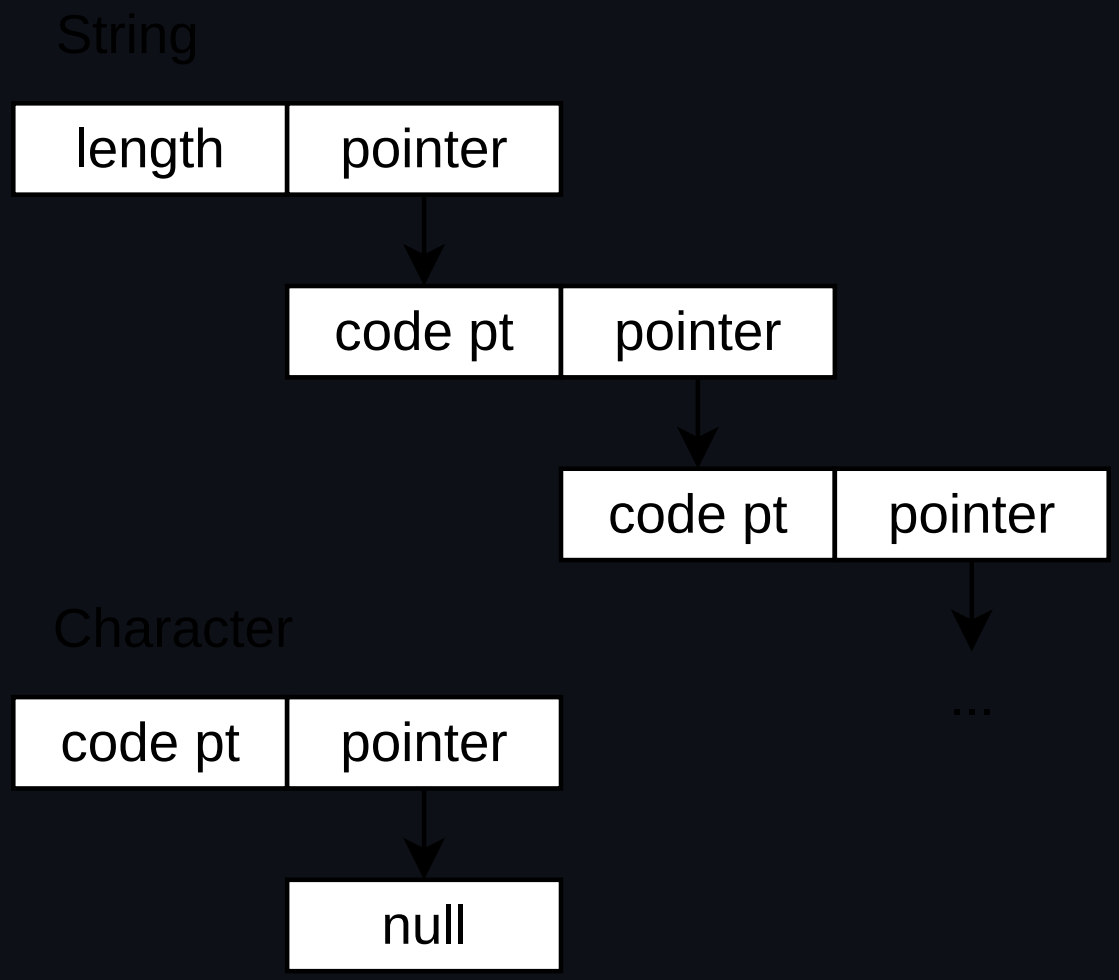
Stak Scheme

- Stak Scheme, the tiny **R7RS-small** implementation
  - The same language design as Ribbit Scheme
- Two use cases
  - Embedded scripting language
  - Standalone interpreter
- Open source on GitHub: `raviqqe/stak`

	Stak	Ribbit
"Bytecode" encoding	Structured memory snapshot	Serialization + ad-hoc merging
`eval` procedure	The compiler itself	A separate library

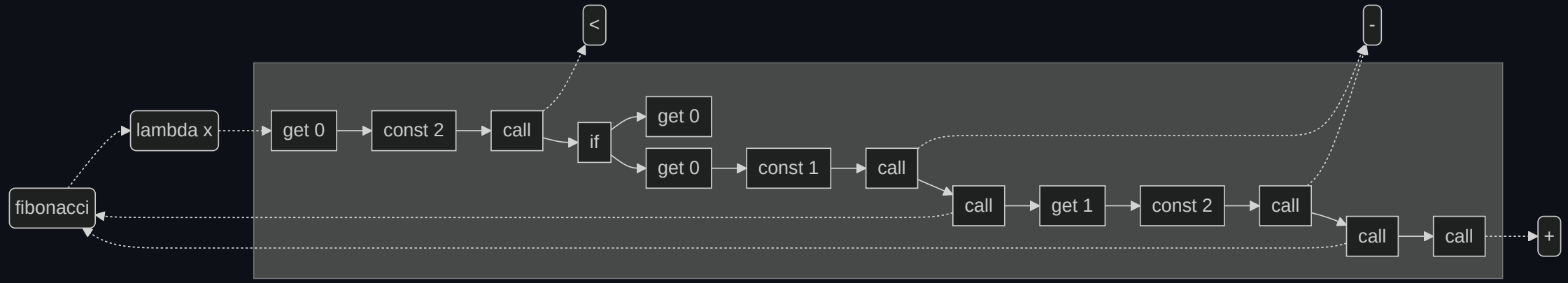
RVM in depth

- Ribbit VM (RVM)
  - A stack machine
- **Everything is a list**.
  - Code
  - Values
    - e.g. lists, characters, strings, ...
  - Call/value stacks
- "Von Neumann architecture"
  - Both code and data in heap



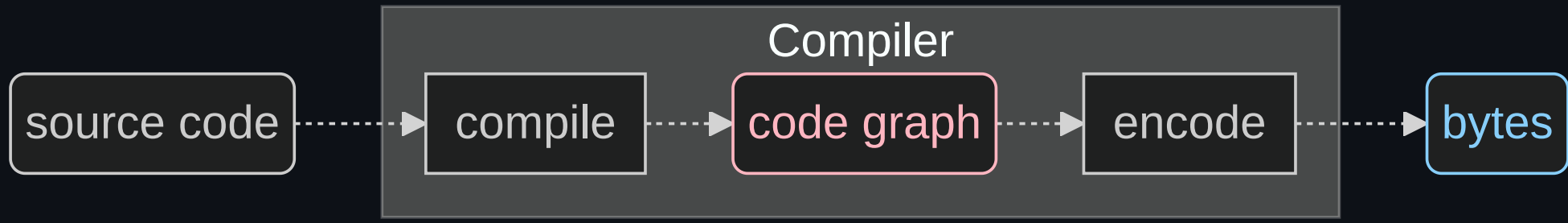
Code graph

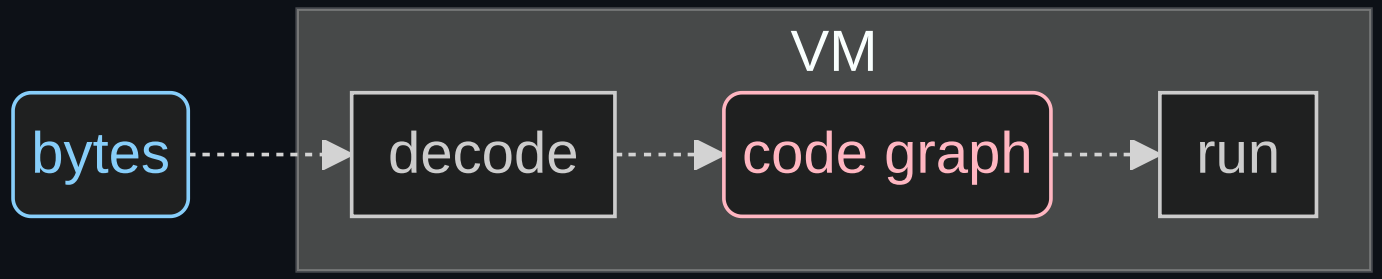
- A representation of a Scheme program on memory
  - Directed Acyclic Graph (DAG) of **pairs**
- Contains both **code** and **data**.
  - e.g. no special interpreter for `eval`



Compiling and running a program

- The compiler compiles source code into a **code graph**.
- The VM runs the code graph as a program.
- Code graphs are used at both **compile time** and **runtime**.





Example

Scheme

```
; Define a `(foo)` library.
(define-library (foo)
  (export foo)

  (begin
    (define foo 123)))
```

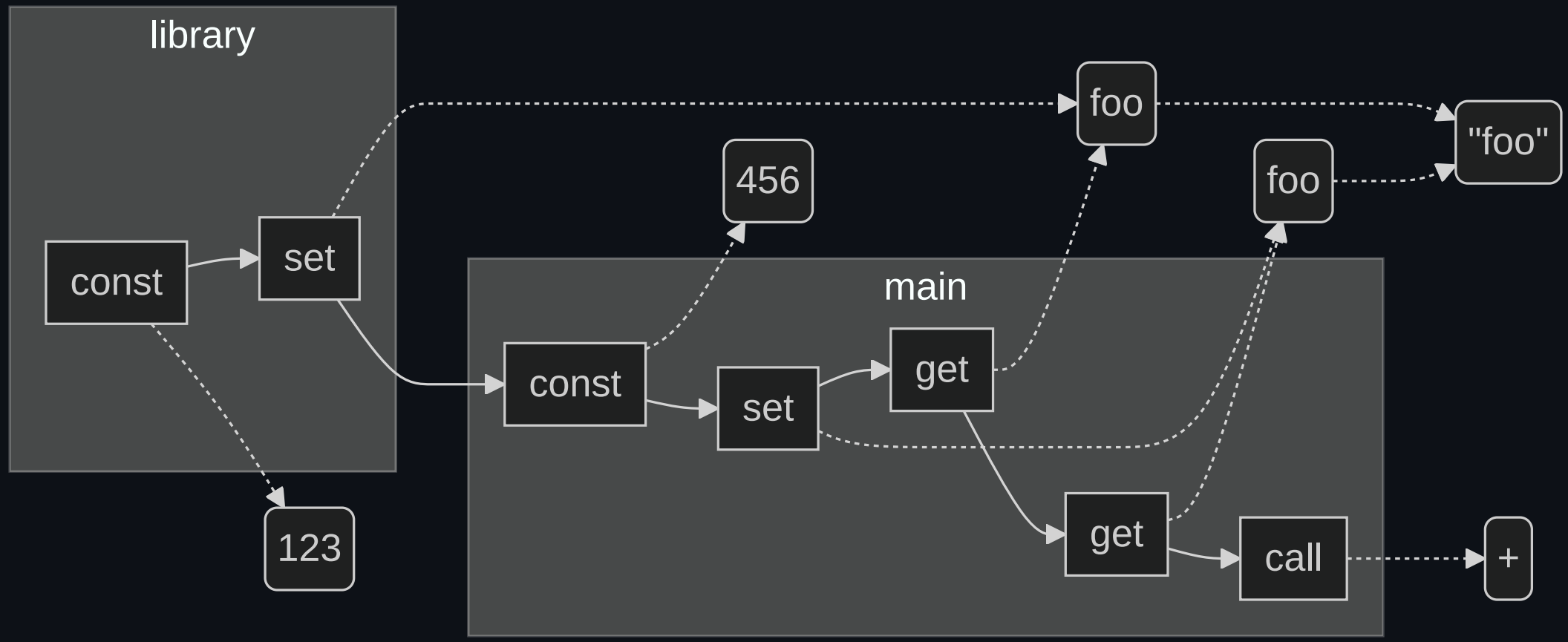
```
; Import the `foo` variable with a prefix.
(import (prefix (foo) bar-))

; Define another `foo` variable.
(define foo 456)

(+ bar-foo foo)
```

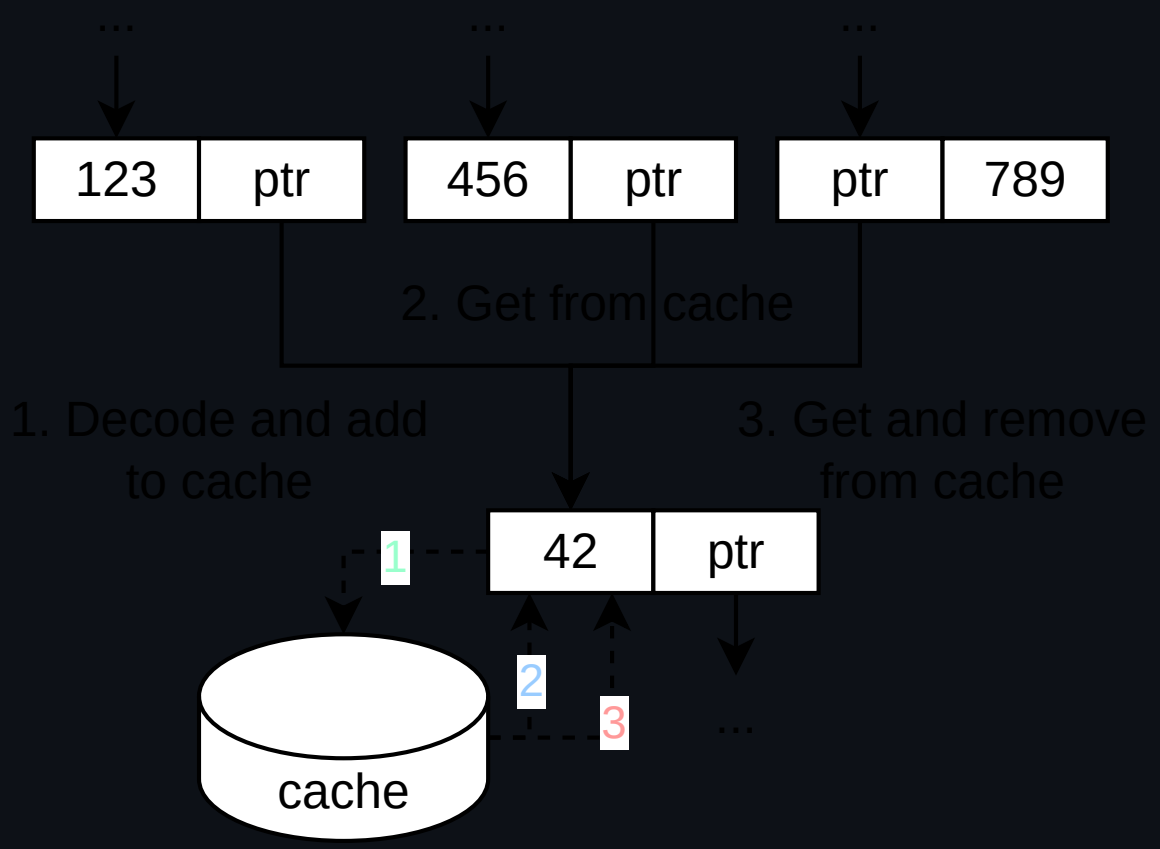
Example

Code graph



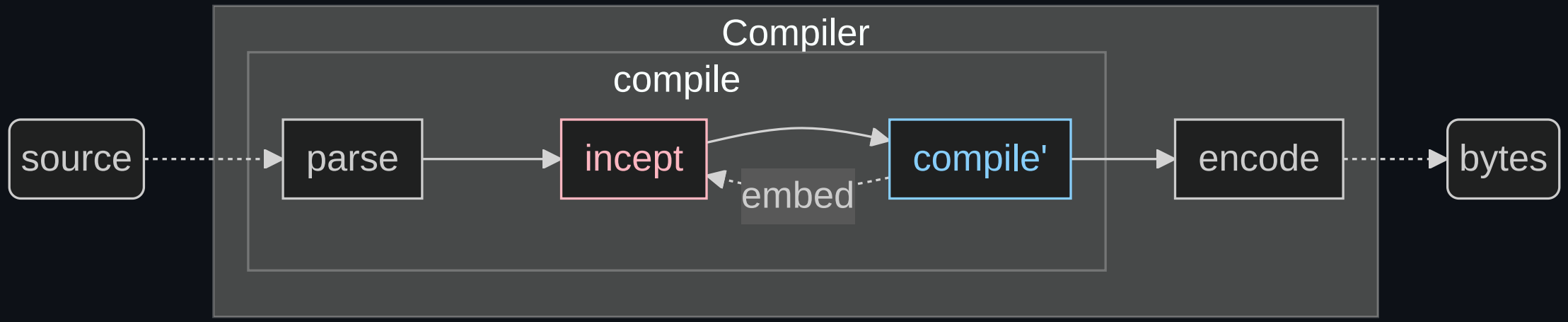
Encoding for structured memory snapshot

- A code graph is encoded by a topological sort.
  - Caches shared nodes
  - **A cache table as a list**
- Simple and portable
- It naturally encodes:
  - `if` instructions' continuations
  - Symbols from different libraries



`eval` and the compiler

- `eval = compiler + VM`
- The compiler from S-expression to code graph is defined as **data**.
- `(incept compiler source)` **embeds the compiler** into source code.
- `((eval compiler) source)` compiles the source code into a code graph.



Macros and libraries in `eval`

- R7RS-small adds some good programming constructs...
  - Hygienic macros
    - i.e. `define-syntax` and `syntax-rules`
  - Library system
- Macros and libraries are expanded at compile time.
- `eval` needs their information at runtime.
  - Macro definitions
  - Initialization code for libraries
- The encoding/decoding transfers macro and library data naturally.

Compactness

- TR7, the tiniest R7RS-small implementation before Stak Scheme
- Stak Scheme implements all the procedures and syntaxes from R7RS-small with some limitations.
  - Only integers and floating-point numbers
  - Partial handling of Unicode

	Lines of code	Binary size (bytes)
mstak	9,127	108,648
tr7i	16,891	301,536

Benchmarks

- Relative computation time
- Fast startup time for small programs

Benchmark	mstak	stak	mstak (embed)	stak (embed)	tr7i	gsi	chibi	gosh
empty	1.00	1.04	0.14	0.38	0.77	0.51	3.63	1.27
hello	1.00	1.04	0.13	0.36	0.73	0.53	9.84	3.62
fibonacci	1.00	1.12	0.96	1.05	1.35	1.66	0.93	0.45
sum	1.00	1.13	1.01	1.06	1.19	1.64	0.98	0.24
tak	1.00	1.09	0.89	0.98	0.96	1.23	1.21	0.54

Future work

- Type checking
  - RVM is flexible but not as secure as other modern ones.
- Porting to another host language
  - e.g. Go, TypeScript, assembly...
- Unicode in the `(scheme char)` library
- Full numeric tower

Acknowledgements

Huge thanks 🙏 to:

- Developers of Ribbit Scheme
  - Especially, the dynamic programming language team at the University of Montréal
- Léonard Oest O’Leary and William E. Byrd for early comments on the draft
- @sisshiki1969 and @yhara for discussions on the language processor design
- And, of course, all the reviewers!

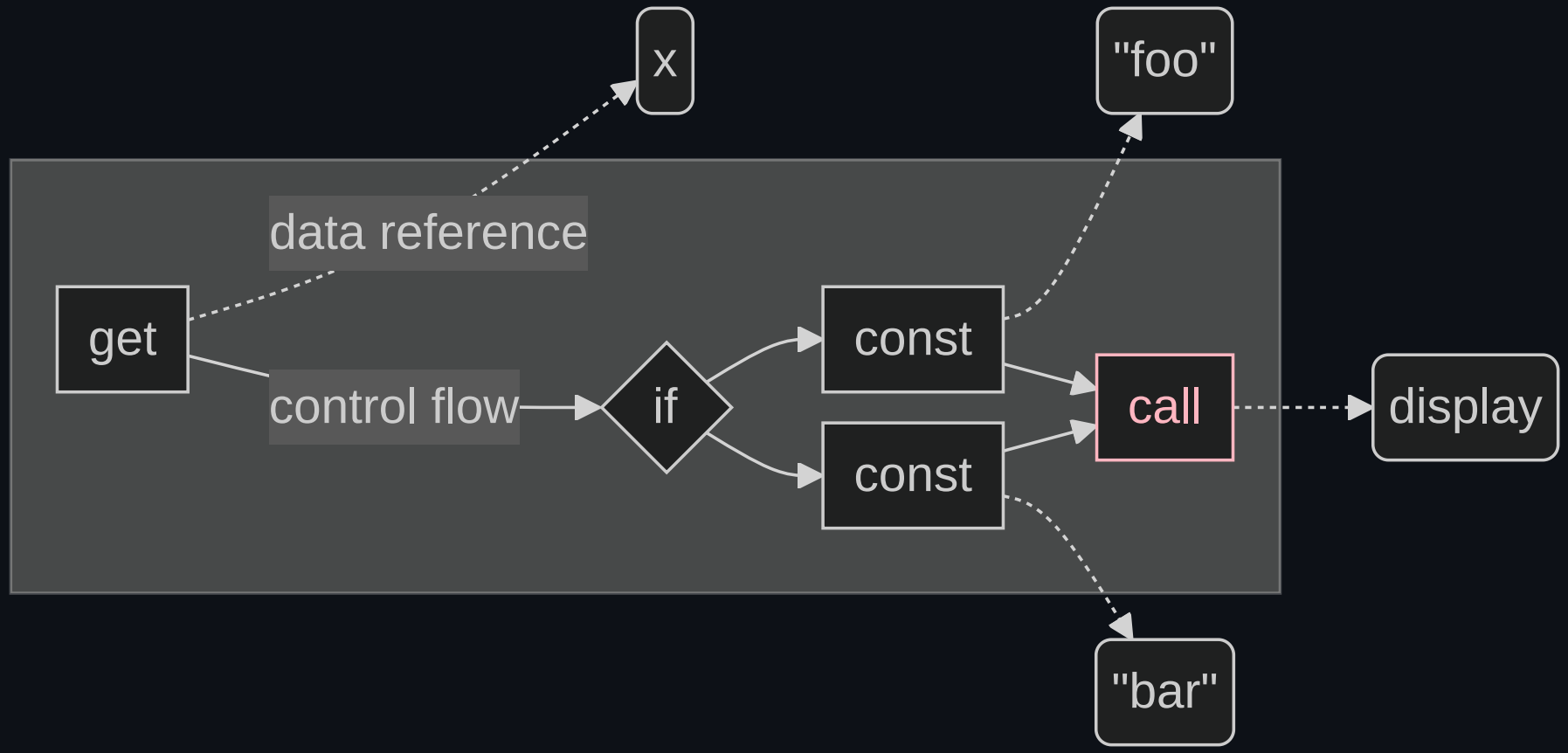
Interpreter demo

- Interpreter: https://raviqqe.com/stak/demo/standalone/interpreter
- Compiler: https://raviqqe.com/stak/demo/standalone/compiler

Appendix

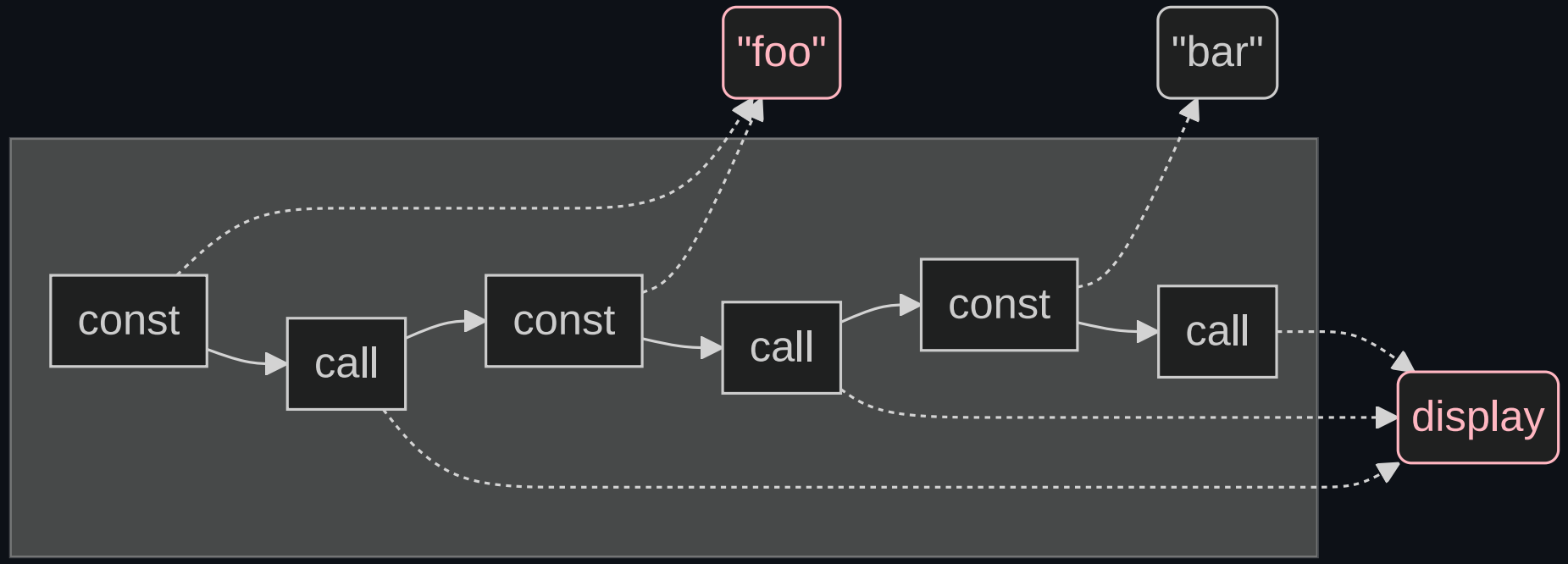
If instruction

```
(display (if x "foo" "bar"))
```



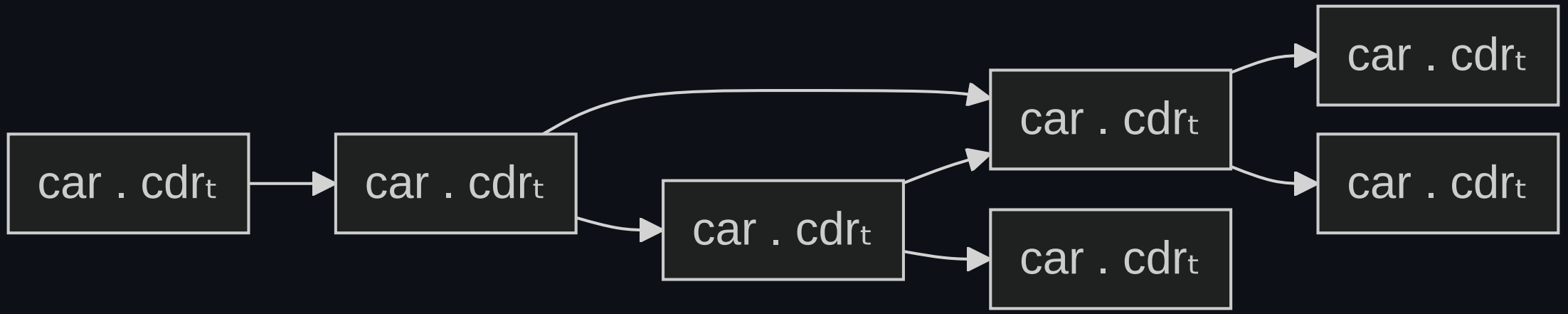
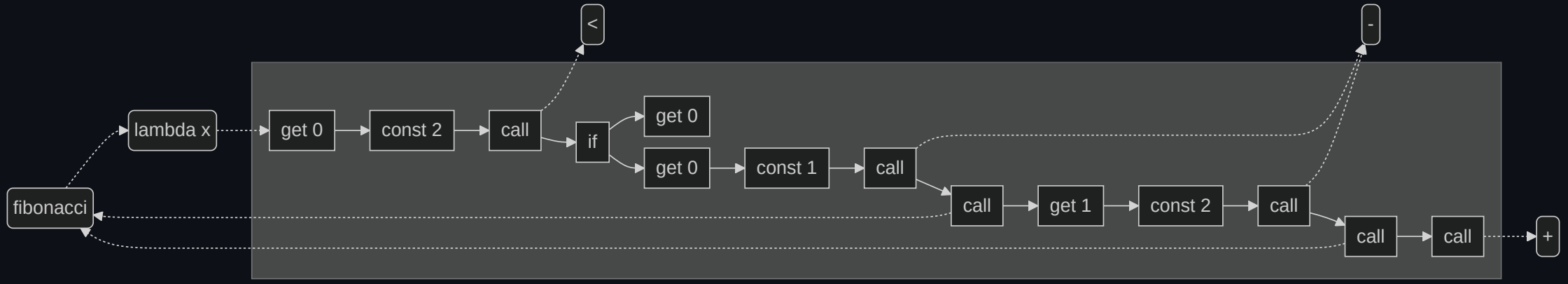
Duplicate strings

```
(display "foo")
(display "foo")
(display "bar")
```



Code graph in depth

- A pair consists of `car`, `cdr`, and a tag on the side of `cdr`.
  - Tags represent either instructions or data types.
- Universal representation for both in-memory bytecode and Scheme values



Fibonacci function