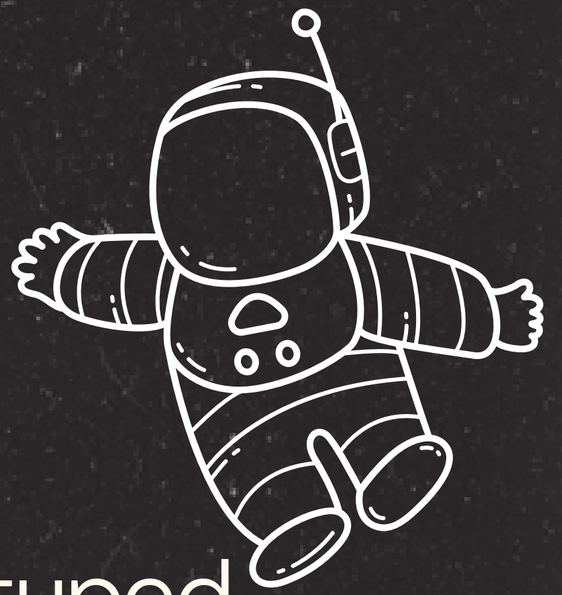
QAMAR

Programming Language

WHAT IS QAMAR?



Open-source multi-paradigm dynamically typed general-purpose programming language with automatic memory management and Bytecode Virtual Machine.

WHY QAMAR?

Qamar is a personal project that I started about 3 months ago, it is aimed to teach me more about programming languages, emulation, translators, and most importantly to have fun.

WHERE DID THE NAME COME FROM?

Qamar is highly inspired by Lua, which is a Portuguese word that means moon, thus I called it Qamar.

Advantages

Simplicity:

Only 16 Keywords exist in the language with simple syntax

Portability:

It runs on top of a bytecode virtual machine similar to JVM

Saftey:

Has Mark-and-Sweep garbage collector that manages memory

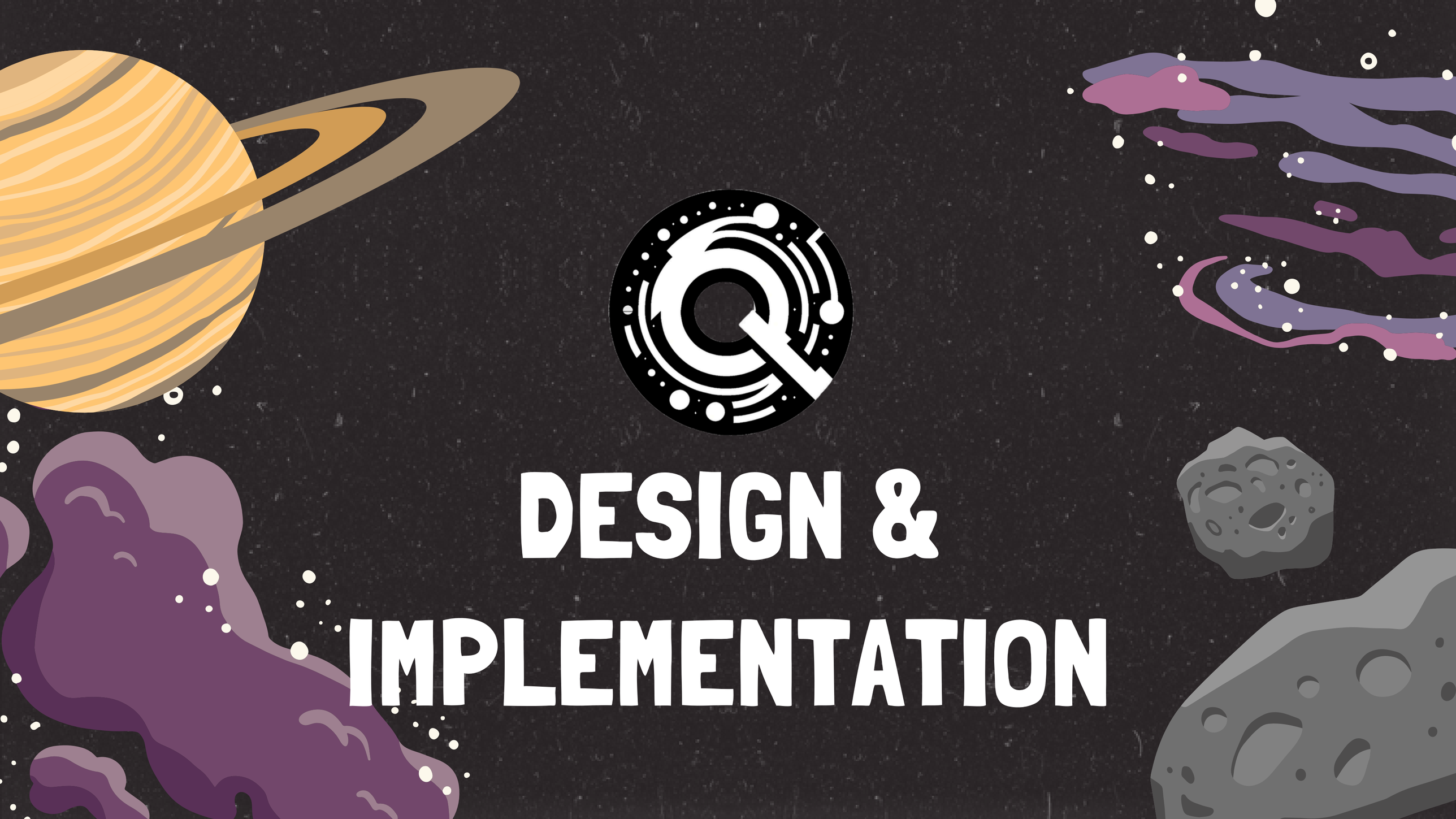
Disadvantages

VM

Overhead, because the language doesn't compile to native instructions

Support

It doesn't have a lot of libraries, IDEs, and compilers


DESIGN &
IMPLEMENTATION

MAIN COMPONENTS

SCANNER (LEXER)

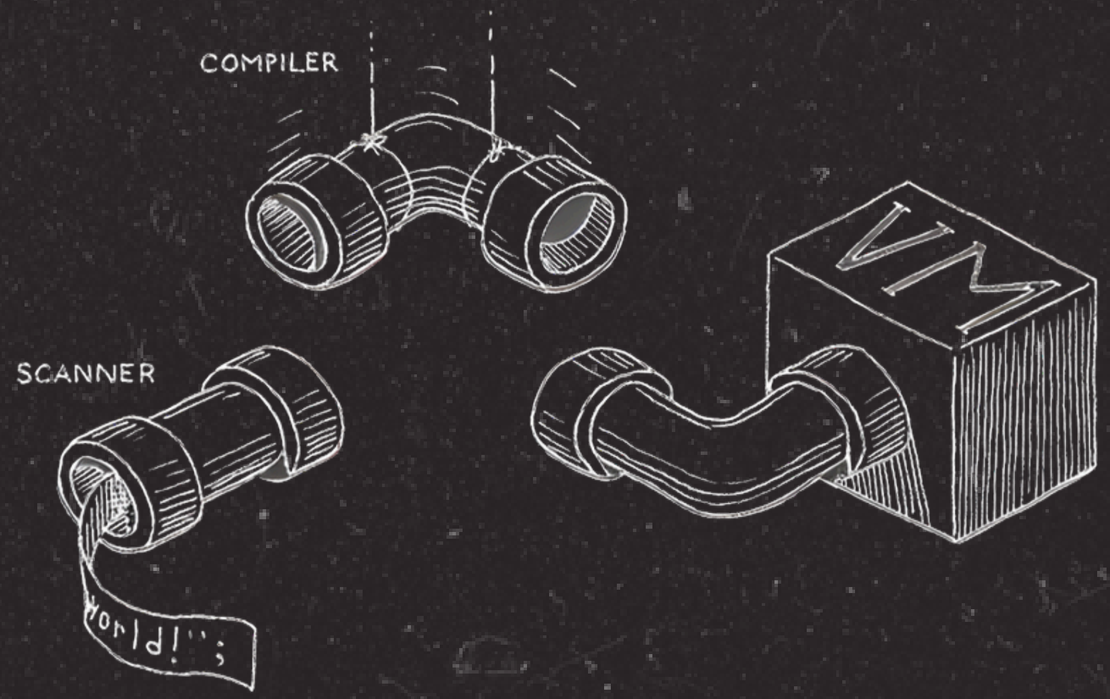
Responsible of tokenization of source code

COMPILER

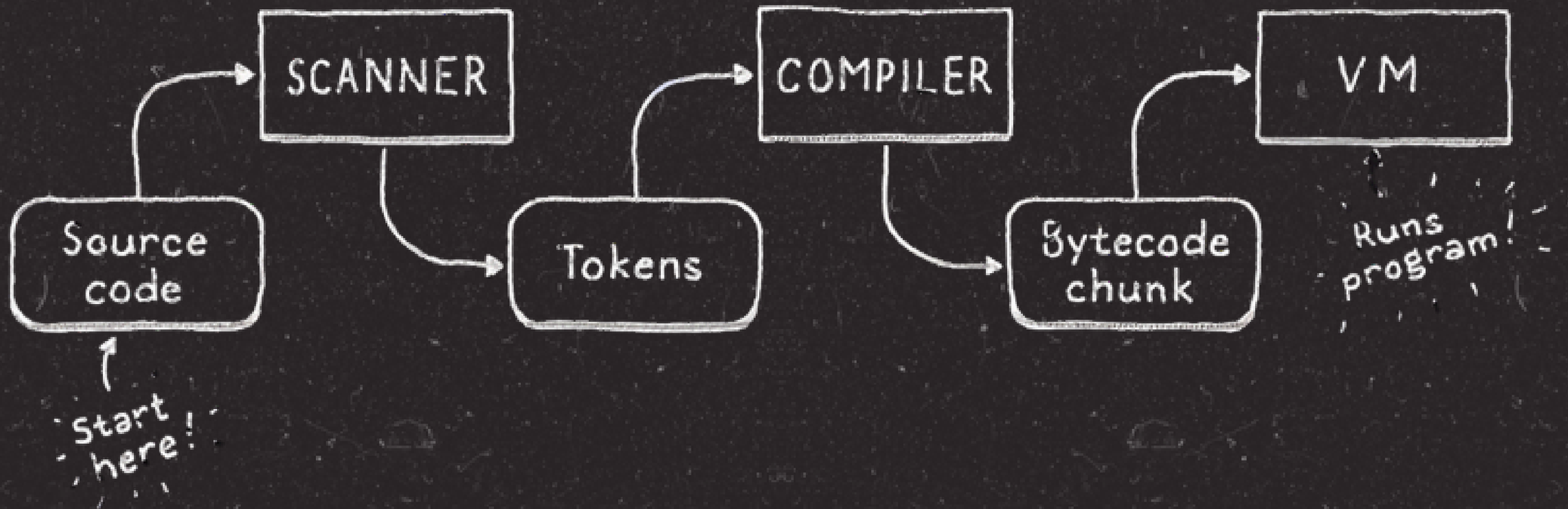
Takes the tokens outputted from the scanner and parses them to create an intermediate representation (Chunks of VM's Bytecode)

VIRTUAL MACHINE

Responsible for the run-time interpretation of the chunks of bytecode emitted by the compiler.



EXECUTION PIPELINE



SINGLE-PASS COMPILER

Qamar uses a single-pass compiler that processes the input exactly once, going directly from lexical analysis to code generator.

ADVANTAGES OF SINGLE-PASS

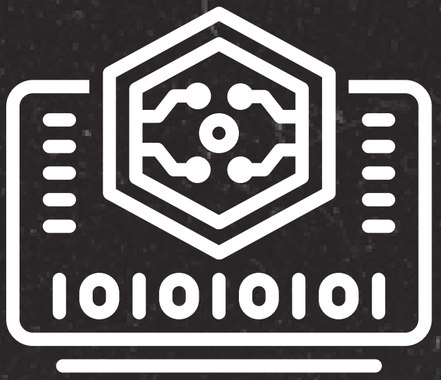
- Faster for small programs and embedded systems in case of small code size and critical time compilation.
- Requires less memory.
- Faster code generation.

PROBLEMS WITH SINGLE-PASS

- Lack of optimization.
- As we can’t back up and process code again the grammar should be limited or simplified.
- Faster code generation.

CHUNKS OF BYTECODE

Qamar’s compiler generates chunks of Bytecode instructions for a custom virtual machine. then the machine interprets the Bytecode at runtime.

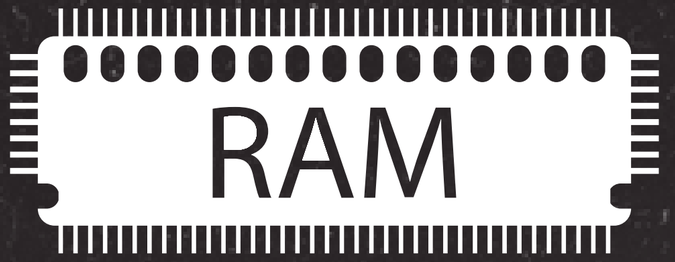


WHY NOT WALK THE AST

Even tho AST's are easier to deal with, there are a couple of reasons why I chose not to go with a Tree-Walk interpreter, and write my own Bytecode virtual machine instead.

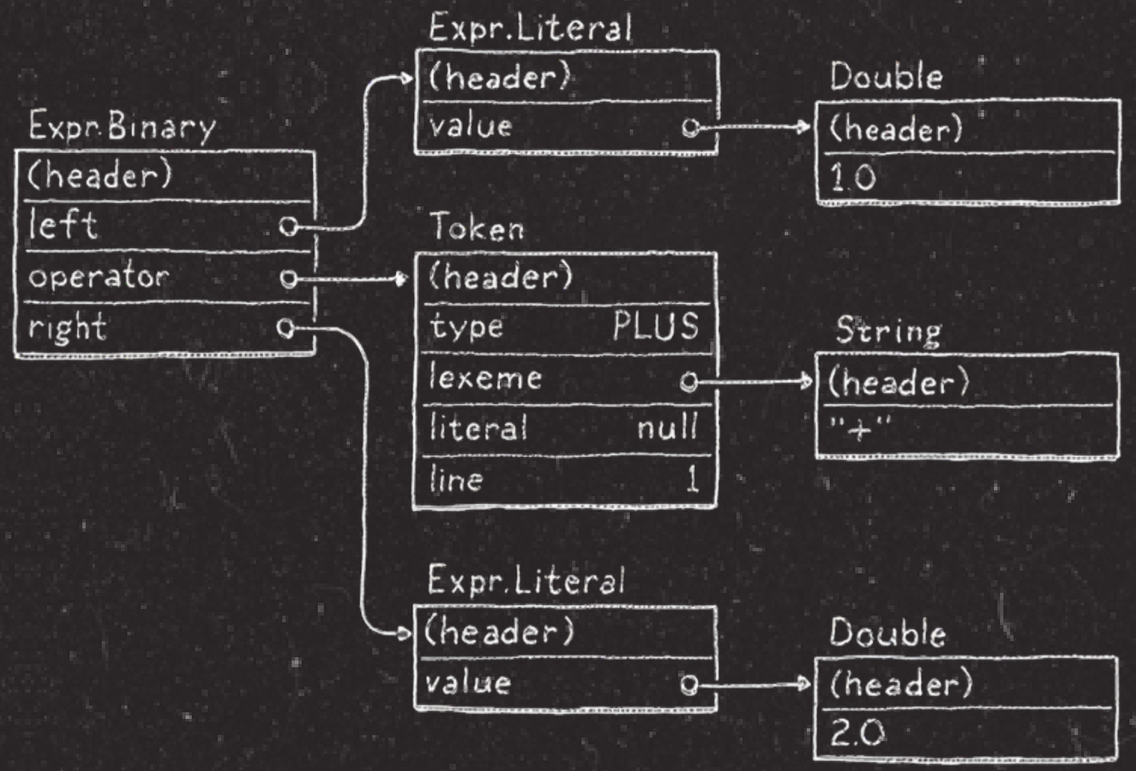
1. AST ISN'T MEMORY-FRIENDLY

AST isn't memory-efficient. Each piece of syntax becomes an AST node. A tiny expression like **1 + 2** turns into a bunch of objects with lots of pointers between them.



CONT...

Each of those pointers adds an extra 32 or 64 bits of overhead to the object.



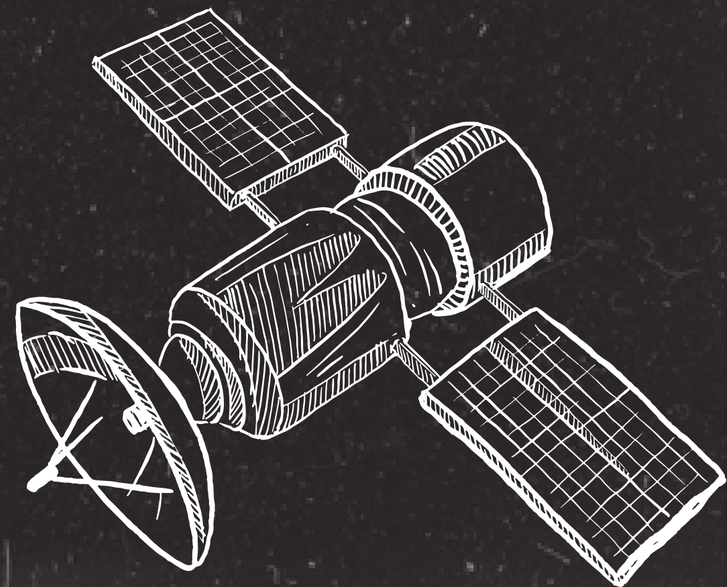
2. AST ISN'T CACHE FRIENDLY

Modern CPUs use caches to significantly speed up data processing, often by 100 times if the data is preloaded. However, abstract syntax trees (ASTs) are not cache-friendly because their scattered structure frequently forces CPUs to fetch new data from RAM, leading to processing delays.

DATATYPES & REPRESENTATION

Qamar has only 3 datatypes:

- Number: Represented as double in C
- String: Represented as a C String
- NIL: Represented as **NULL** in C
- Boolean: Represented as bool in C



GARPAGE COLLECTION

Qamar utilizes a memory management technique known as **Mark-and-Sweep**, which was originally developed by John McCarthy for **Lisp**.

CONT...

Mark-and-Sweep works in two phases:

- Marking
- Sweeping

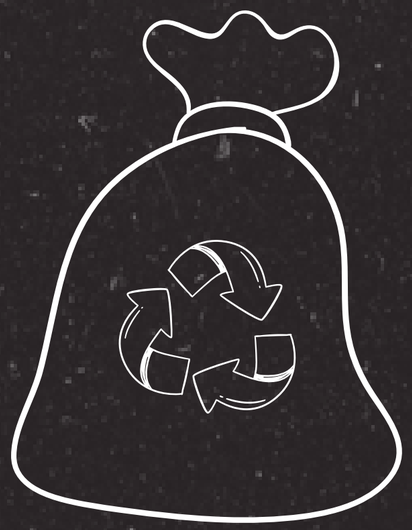
MARKING

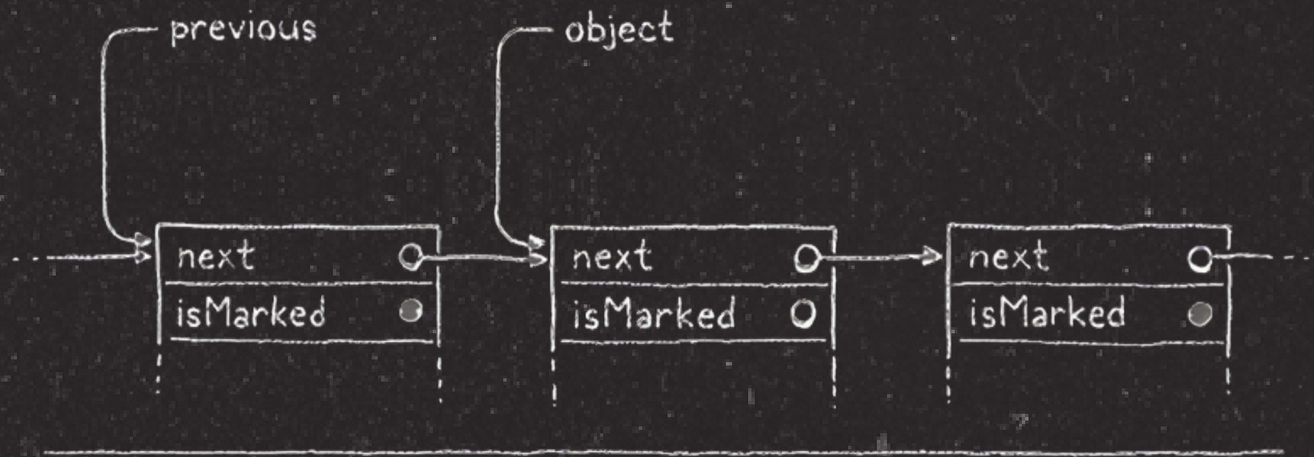
Qamar represents all of the heap-allocated objects as a linked list where each allocated object points to a previously allocated one, each object has a field that determines if it is used (then marked) or not used (not marked).

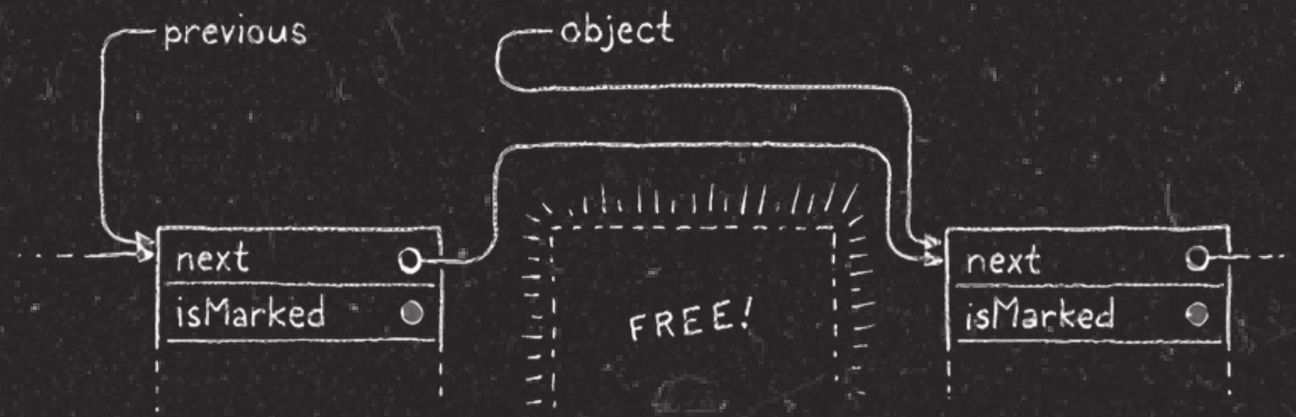
SWEEPING

Once the mark phase completes, all used objects in the heap will be marked. That means any unmarked object is unused thus we can reclaim it. We go through all the unmarked objects and free each one.

VISUALIZATION







CLOSURES & UPVALUES

Closures in programming are a concept where a function captures its surrounding state or environment. Qamar makes use of closures so a function would capture the local variables of the outer function its nested inside.

CLOSURES & UPVALUES

Upvalues refer to the variables from an outer function's scope that are accessed by an inner function, creating a closure. Using this In my language was inspired by Lua’s implementation.

CONT...

This is an example code snippet from Qamar where closures are used. After running this code it should print “x” as the string “value”.

```
15 fun outer() {
14     var x = "value";
13     fun middle() {
12         fun inner() {
11             print x;
10         }
 9
 8         print "create inner closure";
 7         return inner;
 6     }
 5
 4     print "return from outer";
 3     return middle;
 2 }
 1
.6 var mid = outer();
 1 var in = mid();
 2 in();
```


VISION & GOALS

The current objective is to enhance the Qamar project on GitHub by adding more features and native functions, as well as attracting additional contributors.


Thank You :)