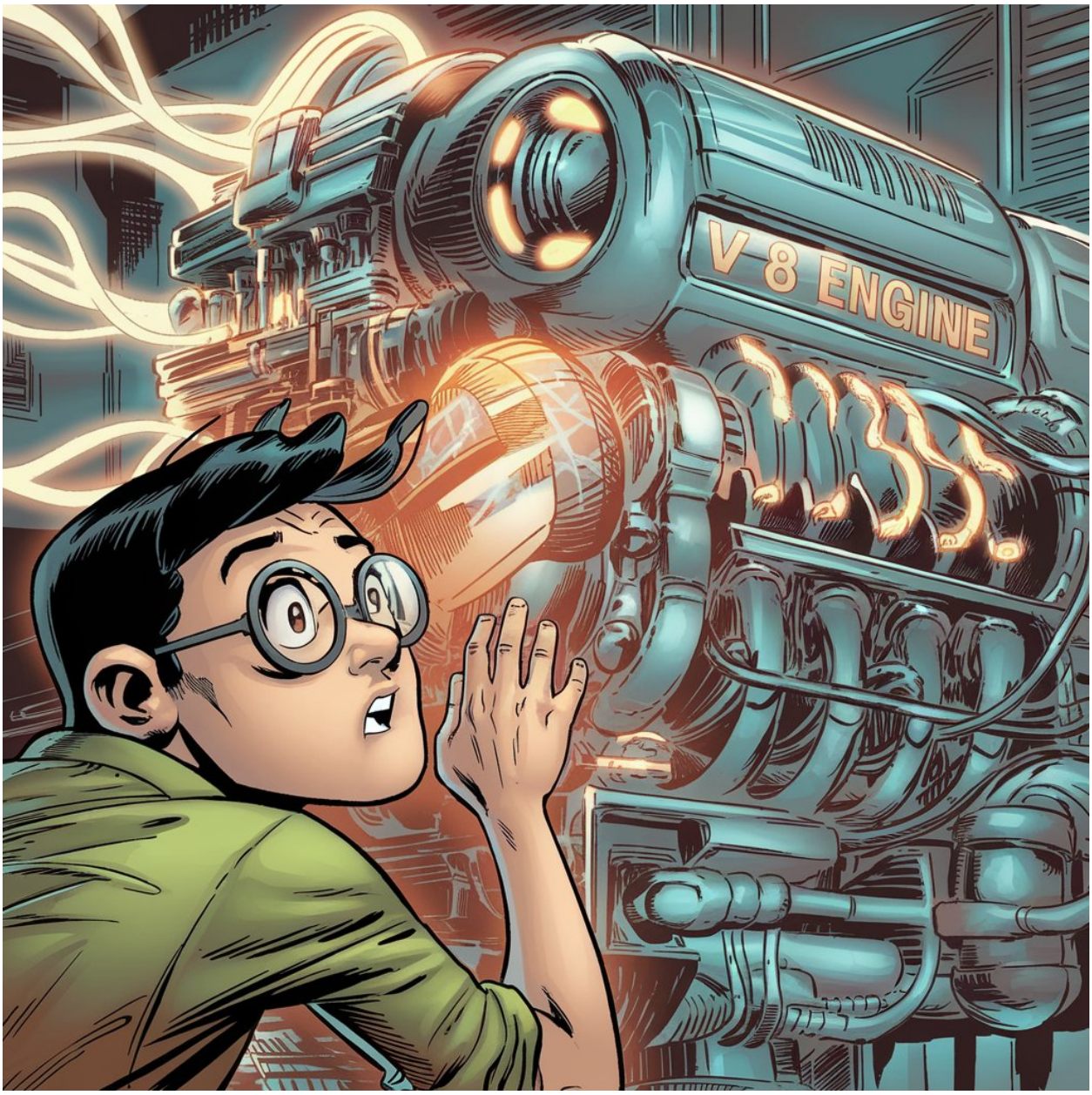
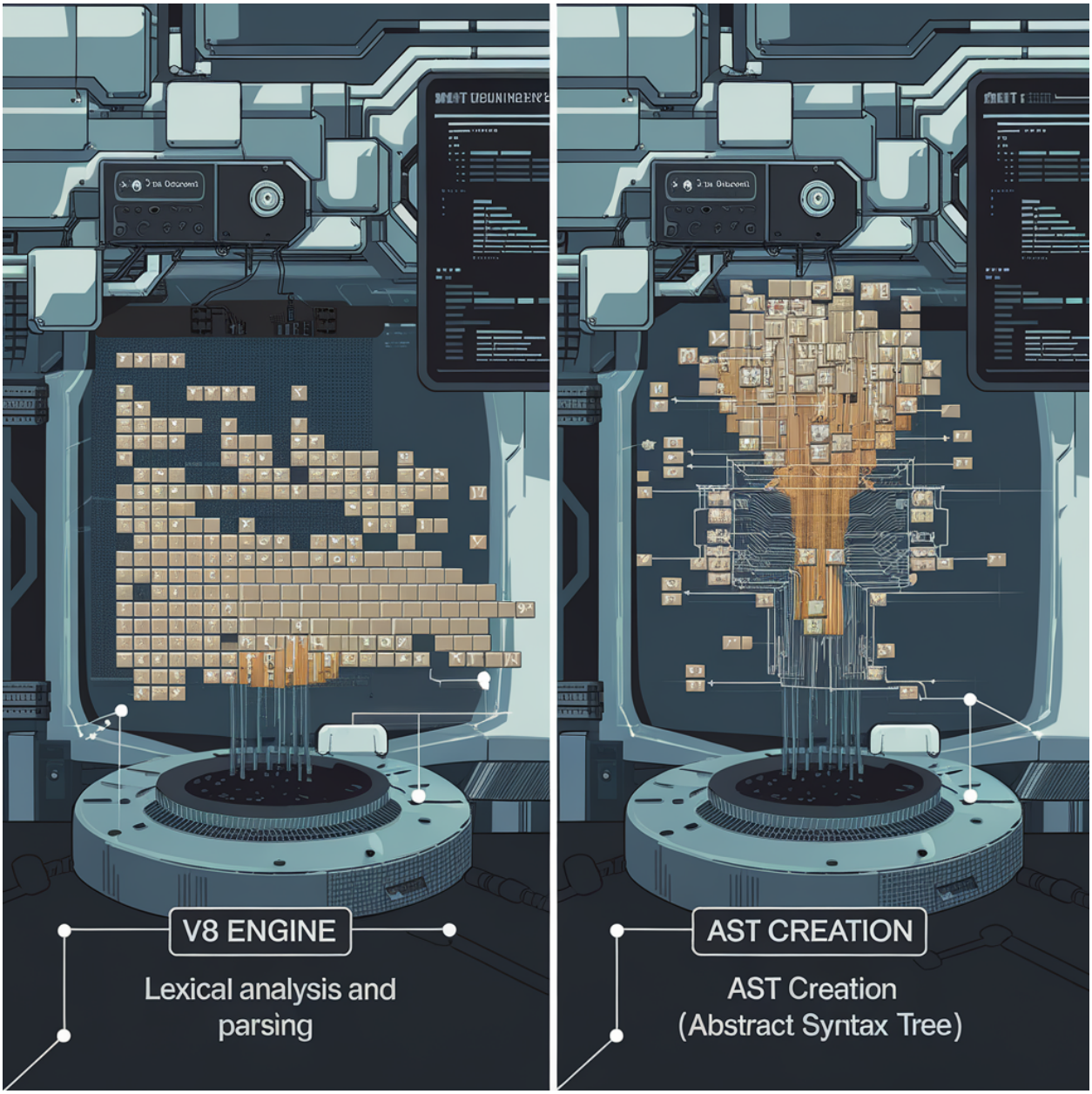
Episode-08 | Deep dive into v8 JS Engine



Behind the Scenes: Parsing Stage in V8 Engine

When JavaScript code is executed, it goes through several stages in the V8 engine. The first stage is
parsing, which includes **lexical analysis** and **tokenization**. Here's how it works:

A) Parsing Stage: Lexical Analysis and Tokenization



1. Lexical Analysis

- **Purpose**: The main goal of lexical analysis is to break down the raw JavaScript code into manageable pieces called **tokens**.
- **Process**:
  1. **Input Code**: `var a = 10;`
  2. **Tokenization**: The code is scanned to identify individual tokens.

Example:

For the code `var a = 10;`, the tokens might be:

- `var` (keyword)
- `a` (identifier)
- `=` (operator)
- `10` (literal)
- `;` (punctuation)

2. What is Tokenization?

- **Definition**: Tokenization is the process of converting code into a series of tokens. Each token represents a fundamental element of the language, such as keywords, operators, identifiers, and literals.
- **Why Tokenization?**: Tokenization helps the V8 engine to read and understand the code more effectively by breaking it down into smaller, more manageable pieces. This step is crucial for further analysis and compilation.

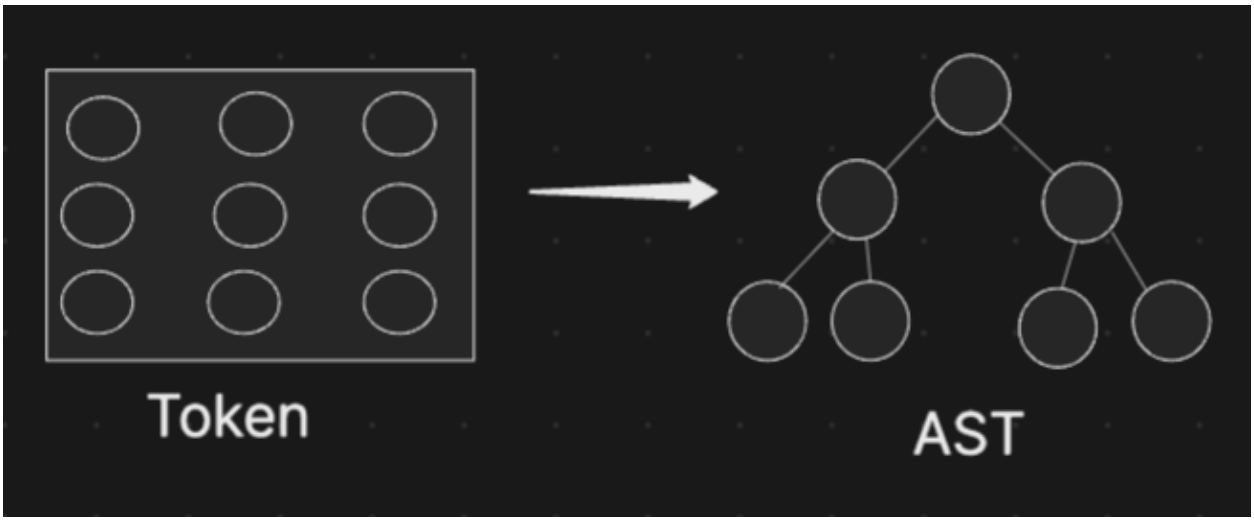
3. Output

- **Tokens**: The result of tokenization is a list of tokens that the V8 engine uses in subsequent stages of parsing.

In the parsing stage,
lexical analysis breaks down JavaScript code into **tokens**. This process helps the V8 engine understand and process the code by converting it into a format that can be easily analyzed and optimized in later stages.

2nd Stage: Syntax Analysis and Abstract Syntax Tree (AST)

After the
lexical analysis and **tokenization** stages, the next step in the parsing process is **syntax analysis**. In this stage, the tokens are converted into an **Abstract Syntax Tree (AST)**. This process is crucial for transforming the flat list of tokens into a structured representation of the code.



1. Syntax Analysis

- To analyze the syntactic structure of the tokens and build the Abstract Syntax Tree (AST).
- The tokens are analyzed according to the grammar rules of JavaScript to create a hierarchical tree structure that represents the code.

```
Tokens
   ↓
Syntax Analysis
   ↓
Abstract Syntax Tree (AST)
```

2. Abstract Syntax Tree (AST)

- The AST is a tree-like data structure that represents the syntactic structure of the source code. Each node in the tree corresponds to a construct in the code, such as variables, expressions, or statements.
- **Example**:

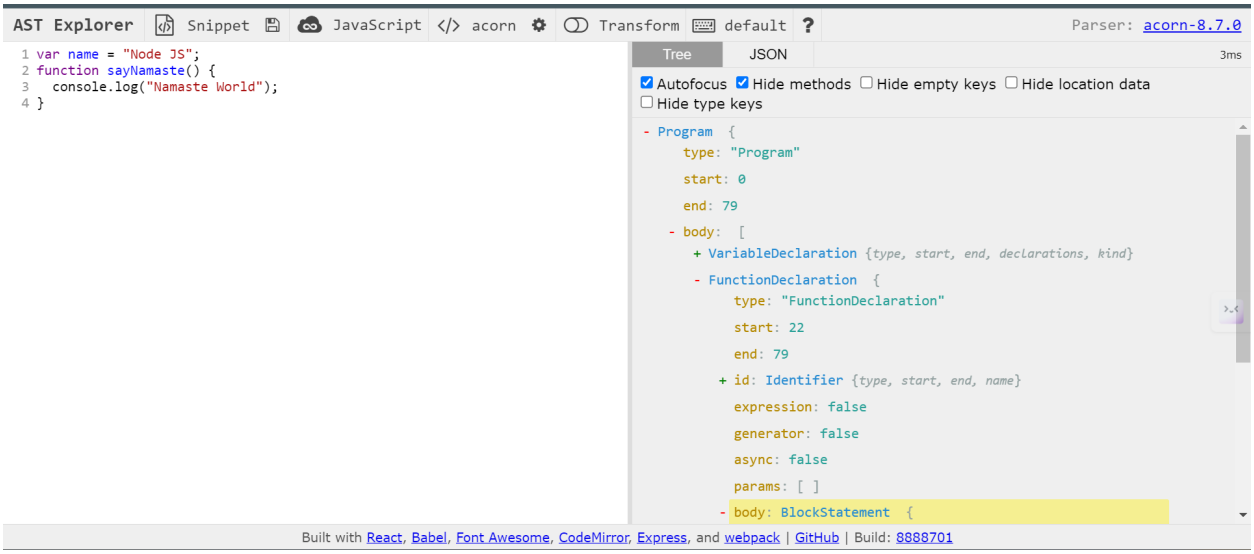
```
For the code:
var a = 10;
```

The AST might look something like this:

```
VariableDeclaration
  ├─ Identifier (a)
  └─ Literal (10)
```

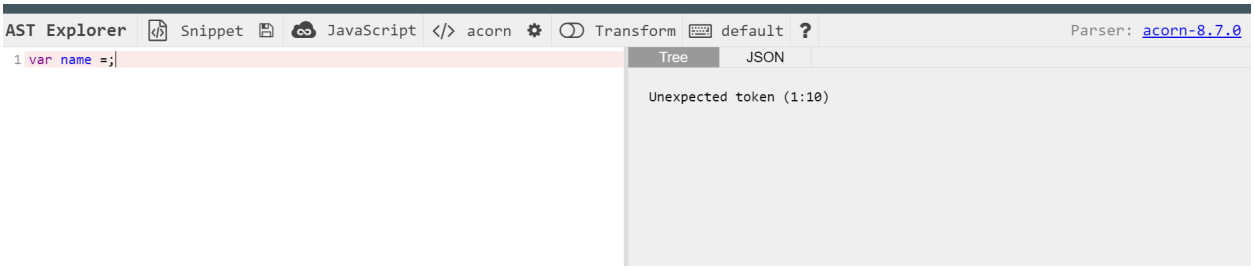
- **VariableDeclaration**: Represents the variable declaration statement.
- **Identifier**: Represents the variable name `a`.
- **Literal**: Represents the value `10`.

Website: You can explore AST structures using tools like AST Explorer, which provides a visual representation of the AST for various pieces of code.



3. Interesting Fact

- **Syntax Errors**: When the V8 engine reads code, it processes tokens one by one. If an unexpected token is encountered that does not fit the grammar rules, a **syntax error** occurs. This is because the AST cannot be generated if the code does not adhere to the expected syntax, indicating that something is wrong with the structure of the code.



In the syntax analysis stage, the tokens are analyzed to create an Abstract Syntax Tree (AST), which provides a structured and hierarchical representation of the code. This tree structure helps the V8 engine understand and process the code more effectively. Syntax errors occur when the engine encounters unexpected tokens that prevent the generation of a valid AST.

B) Interpreter and Compilation

1. Interpreted vs. Compiled Languages

Interpreted Languages:

- **Definition**: These languages are executed line by line. The interpreter reads and executes the code directly, which can lead to slower execution times compared to compiled languages.
- **Pros**: Faster to start executing code, easier to debug.
- **Cons**: Slower execution compared to compiled languages because of the line-by-line interpretation.

Example: Python

Compiled Languages:

- **Definition**: These languages are first translated into machine code (binary code) through a process called compilation. The machine code is then executed by the computer's hardware, leading to faster execution times.
- **Example**: C, C++.
- **Pros**: Faster execution because the code is pre-compiled into machine code.
- **Cons**: Longer initial compilation time, more complex debugging process.

Q: Is JavaScript interpreted or compiled language?😐

A:

JavaScript is neither purely interpreted nor purely compiled. It utilizes a combination of both techniques

- **Interpreter**:
  - **Initial Execution**: JavaScript uses an interpreter to execute code quickly and start running the script. This allows for rapid execution of scripts and immediate feedback.
- **Compiler**:
  - **Just-In-Time (JIT) Compilation**: JavaScript engines like V8 use JIT compilation to improve performance. JIT compilation involves compiling code into machine code at runtime, just before execution. This process optimizes performance by compiling frequently executed code paths into optimized machine code.

```
JavaScript Code
      ↓
  Initial Interpretation (line-by-line)
      ↓
  Execution
      ↓
 JIT Compilation (optimization)
      ↓
  Machine Code Execution
```

1. Abstract Syntax Tree (AST) to Bytecode

- **AST to Bytecode**: After parsing the code and generating the AST, the code is passed to the interpreter. In the V8 engine, this interpreter is called **Ignition**.

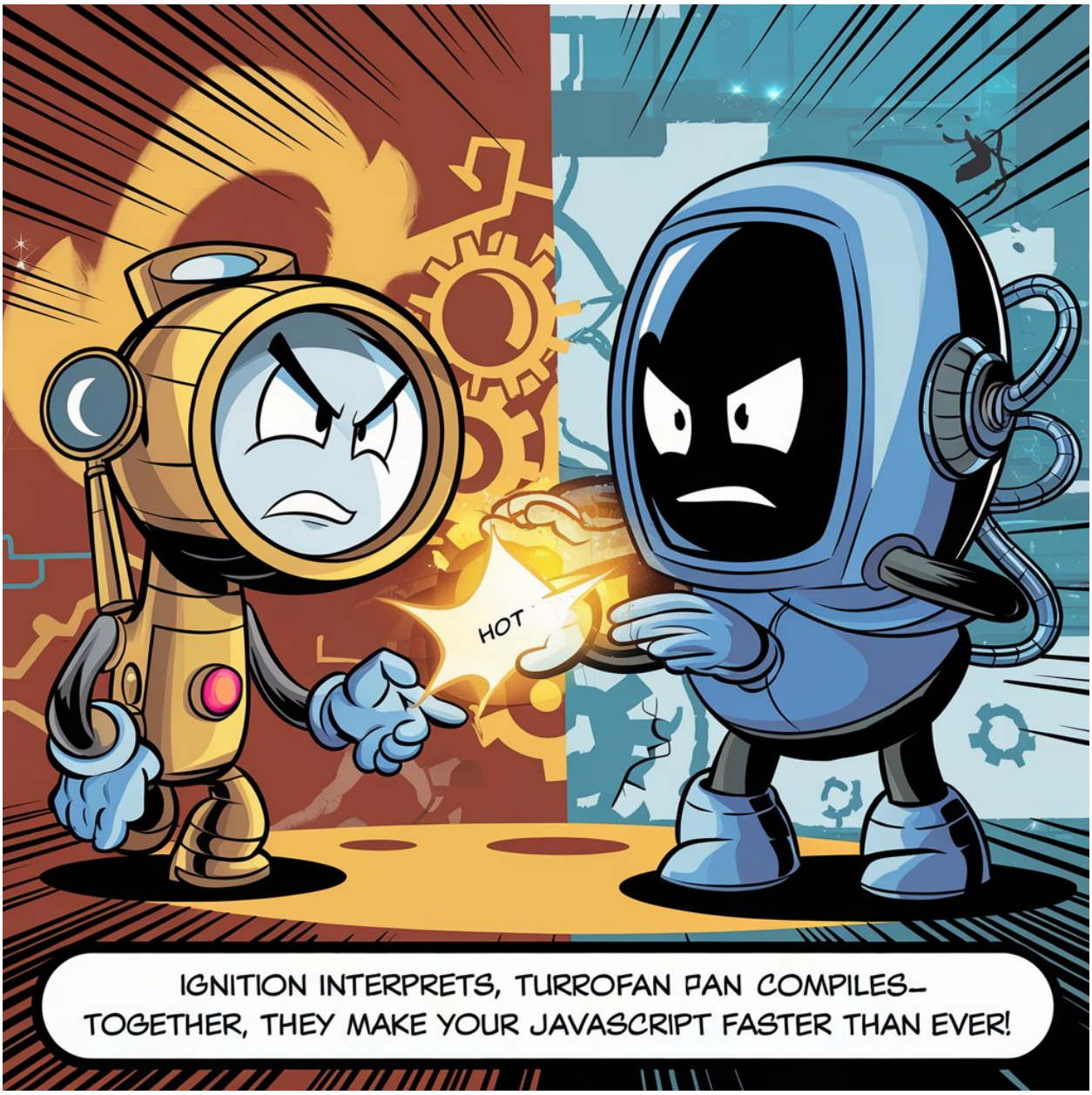
Ignition:

- Converts the AST into bytecode. Bytecode is a lower-level, intermediate representation of the code that the JavaScript engine can execute more efficiently than raw source code.
- **Execution**: Ignition reads and executes the bytecode line by line.

2. Just-In-Time (JIT) Compilation

TurboFan:

- A compiler within the V8 engine that optimizes frequently executed (hot) code paths. When Ignition identifies a portion of the code that runs frequently (hot code), it sends this code to TurboFan for optimization.
- **Optimization**: TurboFan converts the bytecode into optimized machine code, which improves performance for repeated executions.



3. Hot Code Optimization and Deoptimization

- **Hot Code**: Refers to code that is executed frequently. TurboFan focuses on optimizing hot code to improve performance.
- **Optimization Assumptions**:
  - TurboFan makes certain assumptions during optimization based on the types and values it encounters. For example, if a function is optimized with

the assumption that it only processes numbers, it will run very efficiently for such cases.

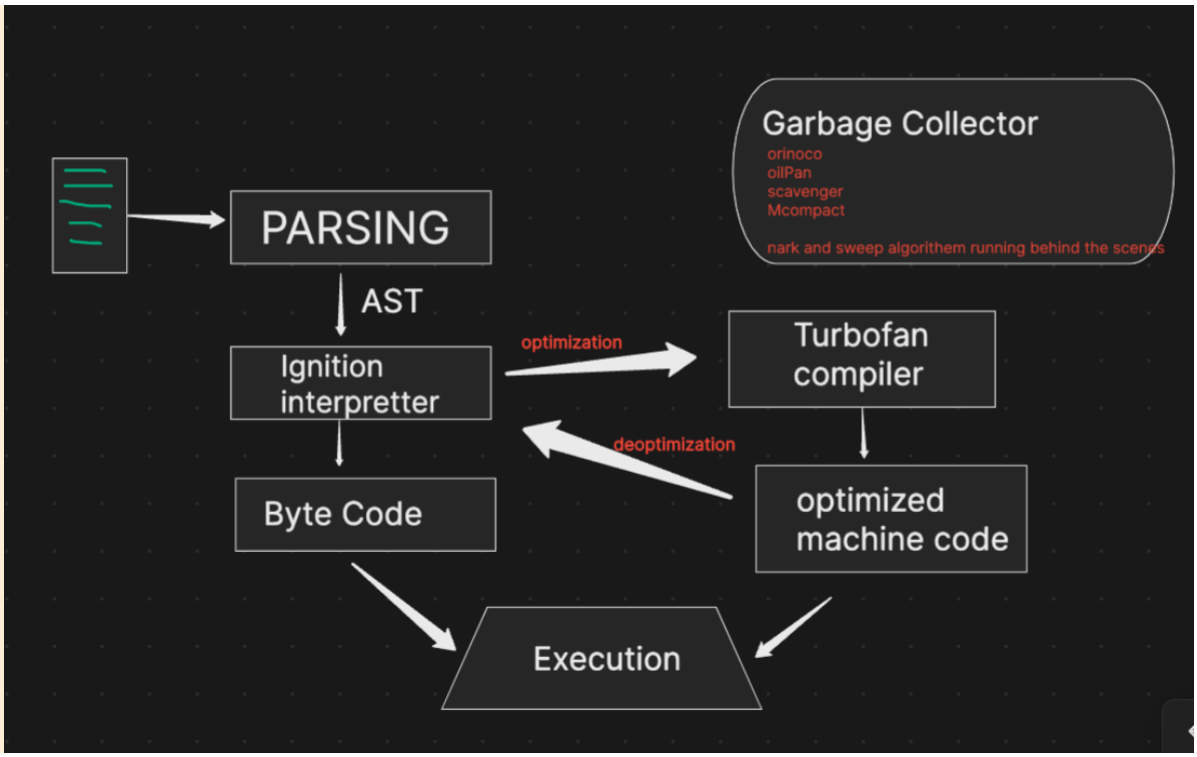
- **Deoptimization**:
  - **Scenario**: If TurboFan's assumptions are incorrect (e.g., a function that was optimized for numbers receives strings), the optimization may fail.
  - **Process**: In such cases, TurboFan will deoptimize the code and revert it to a less optimized state. The code is then sent back to Ignition for further interpretation and possible re-optimization.

Key Terms:

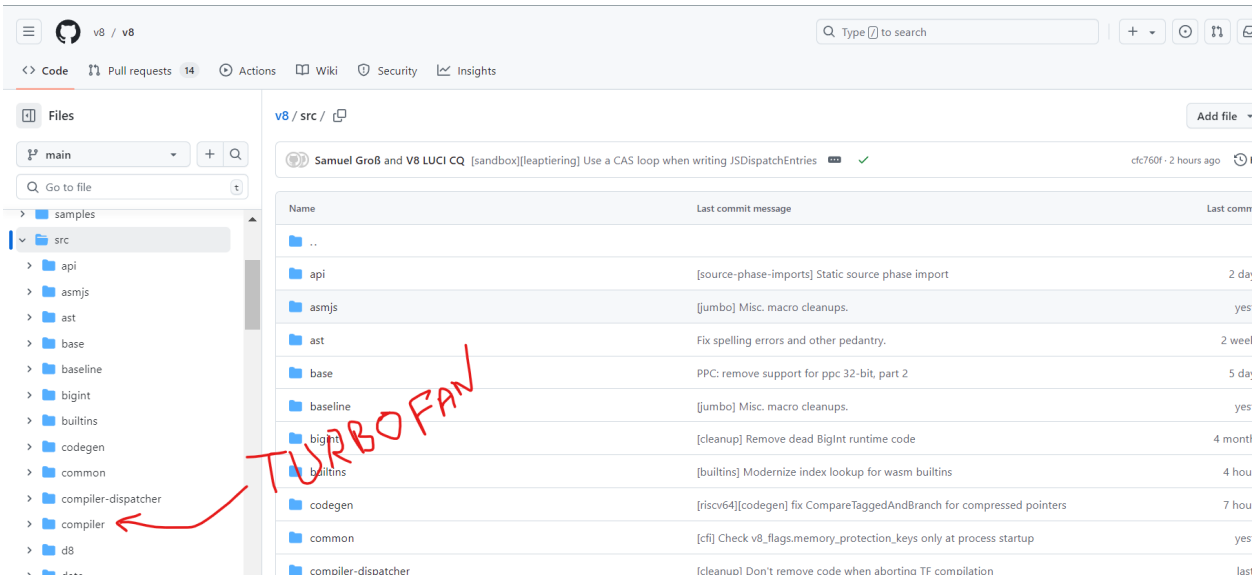
- **Inline Caching**: A technique used to speed up property access by caching the results of lookups.
- **Copy Elision**: An optimization technique that eliminates unnecessary copying of objects.

Developer Note:

- **Best Practice**: For optimal performance, try to pass consistent types and values to functions. For example, if a function is optimized for numeric calculations, avoid passing strings to prevent deoptimization.


Garbage Collector
orinoco
oilPan
scavenger
Mcompact
nark and sweep algorithem running behind the scenes
PARSING
AST
Ignition interpretter
optimization
Turbofan compiler
deoptimization
Byte Code
optimized machine code
Execution

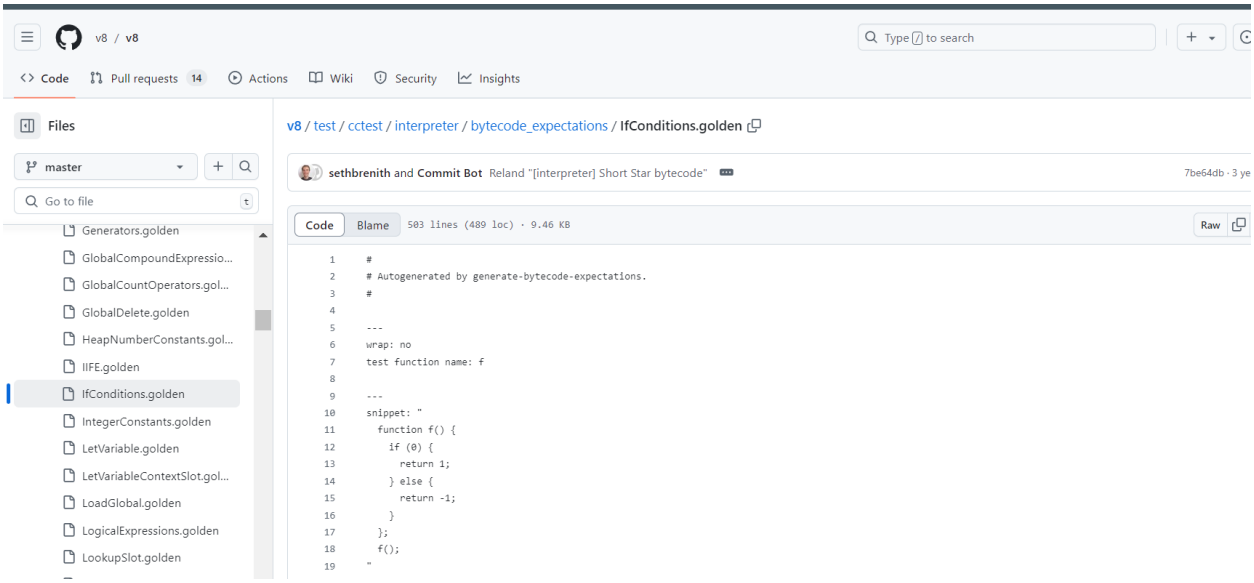

LET ME SHOW YOU
SOME COOL THINGS!



All of these processes work differently in each JavaScript engine, such as SpiderMonkey or others, but the V8 engine is considered the best on the market. Understanding the structure of the V8 engine is very beneficial for you.

Now, I'll show you what bytecode looks like. Your task is to explore this code using ChatGPT.

> https://github.com/v8/v8/blob/master/test/cctest/interpreter/bytecode_expectations/IfConditions.golden



Check:

https://v8.dev/


You've just been
with broadoar advanced,
in-depth in-depth Knowledge!
Thank for joining uss journey.
Stay cun to let tlaarnd
keep learning!
V8