



# Node.js

**Reporter**  
**Andenaik Barseghyan**

<https://github.com/Rebiss/Presentation>



# Content



**Whot is Node.js**



**Libuv. Event Loop. Threads**



**Core Modules. C++ Bindings**



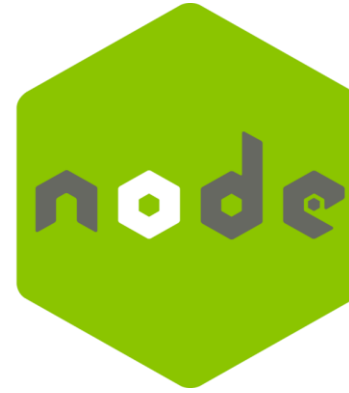
**C-ares. ZLib. HTTP Parser. OpenSSL**



**V8 Engin. WebAssembly**



**Microtask and Macrotask**



**libuv**



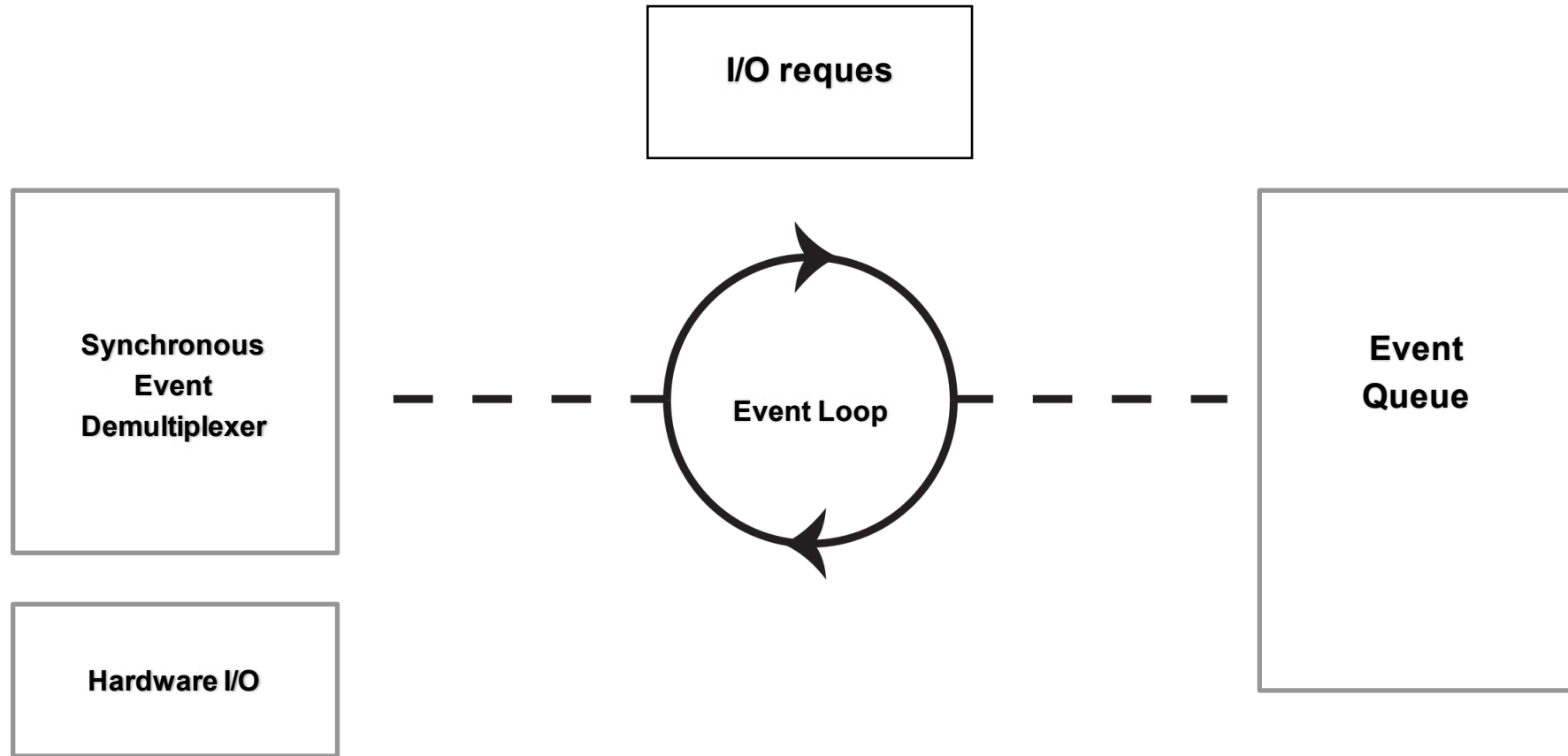
# What is Node.js

Node.js<sup>®</sup> is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Node.js uses an event-driven(Reactor Pattern), non-blocking I/O model that makes it lightweight and efficient.



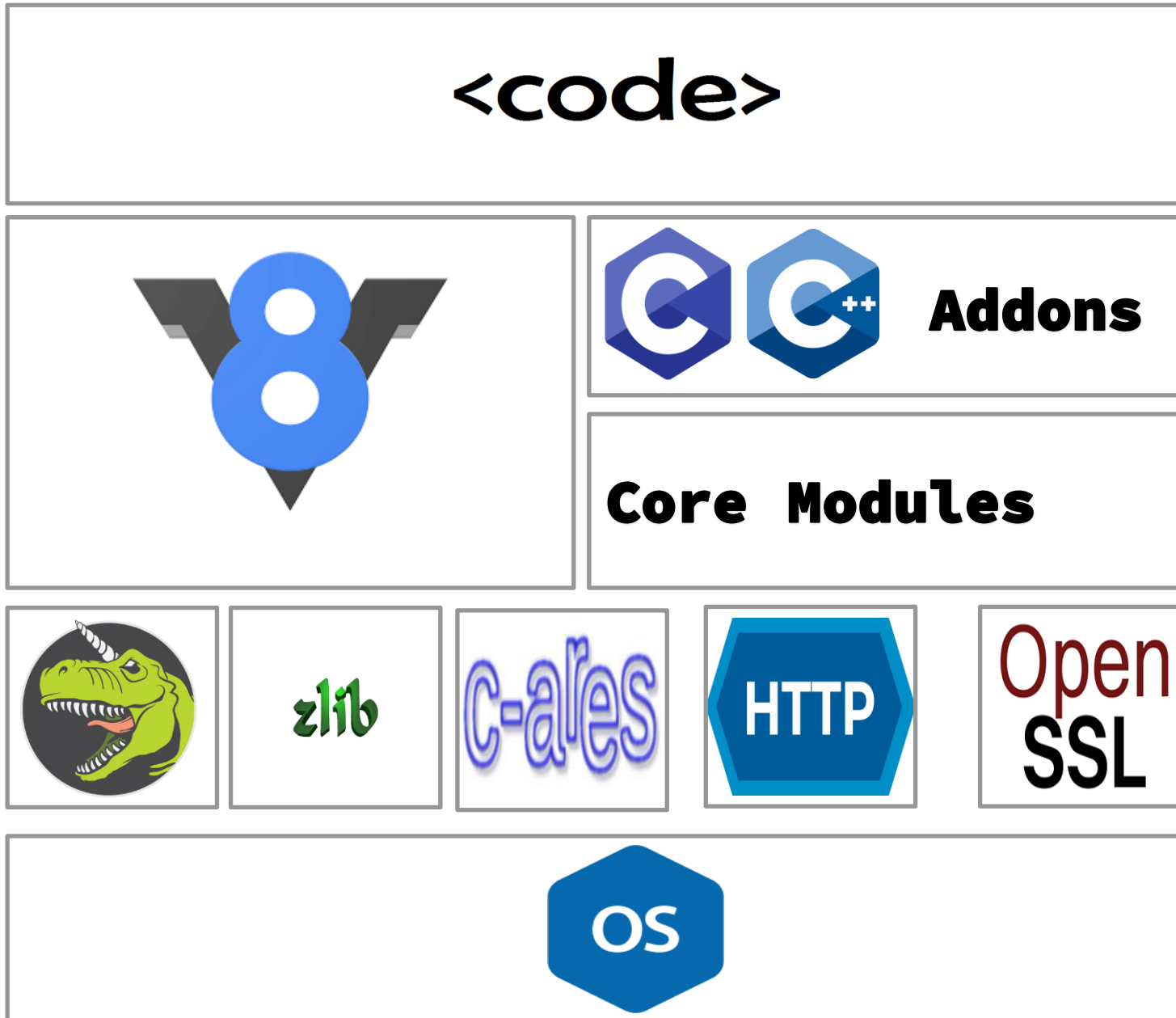
# Reactor Pattern



The **reactor design pattern** is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs. The service handler then demultiplexes the incoming requests and dispatches them synchronously to the associated request handlers

**Structure:** Synchronous Event Demultiplexer, Dispatcher, Request Handler

# Node.js Architecture



Written in

**C**

# Libuv



**Event Loop**

**Operating  
system**

**Cross-  
platform**

**Libuv (Unicorn Velociraptor Library)** is a multi-platform C library that provides support for asynchronous I/O based on event loops.

**Type**

**I/O  
abstraction  
library**

## Features

**Full-featured  
event loop  
backed by  
epoll,  
kqueue, IOCP,  
event ports.**

**Asynchronous  
TCP and UDP  
sockets.  
DNS resolution.**

**Asynchronous  
file and file system  
operations.  
File system  
events.**

**ANSI escape  
code  
controlled  
TTY.  
Child  
processes.**

**IPC with  
socket sharing,  
using Unix  
domain sockets  
or named  
pipes.**

**Thread pool.  
Signal  
handling.  
High  
resolution  
clock.**

# Event Loop or Semi-infinite Loop

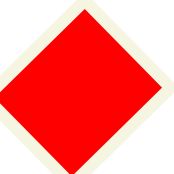
while loop.



**Timer.**  
`setTimeout, setInterval`



**I/O.**  
`Callback, network...`

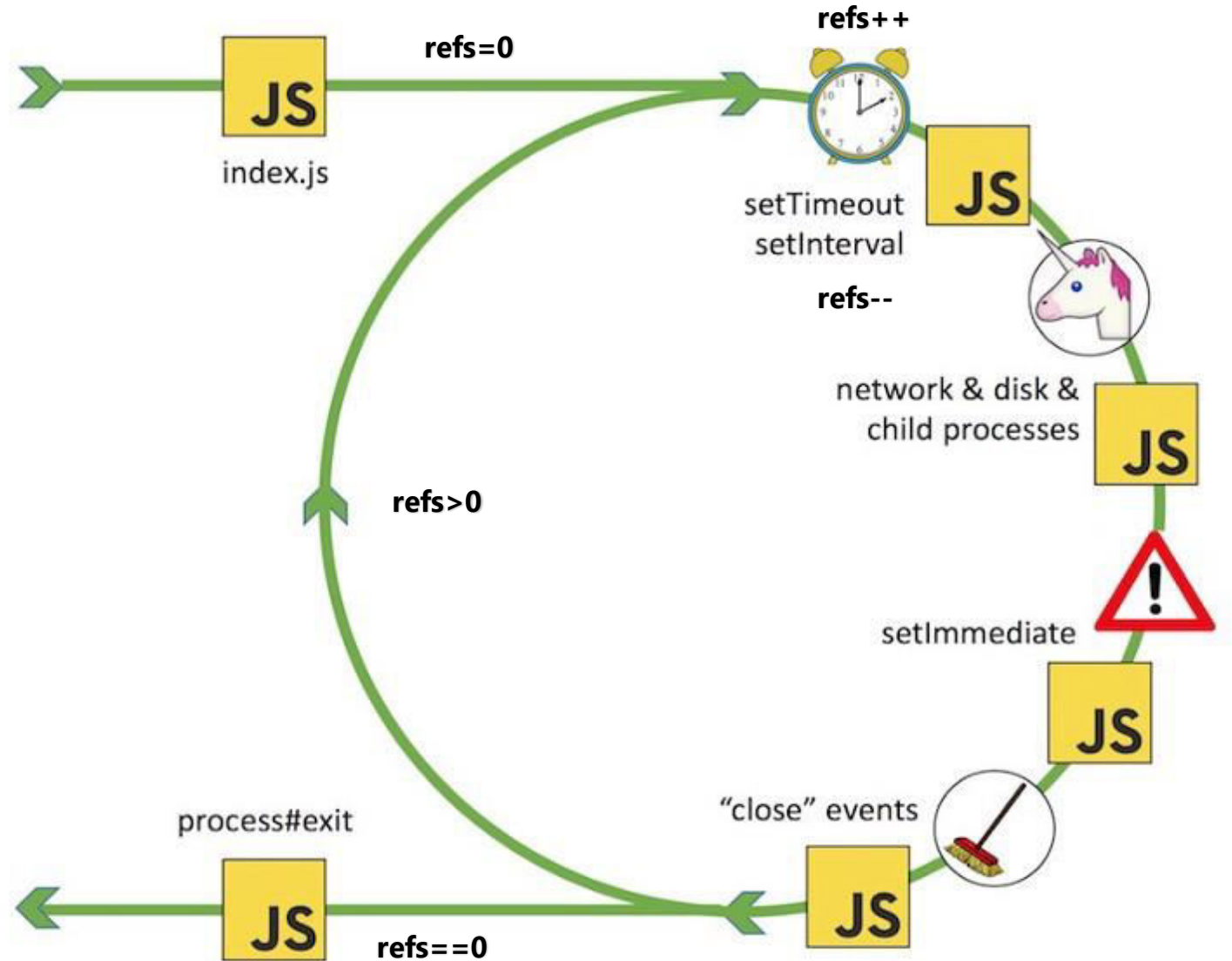


**Immediate.**  
`setImmediate`



**Close events.**  
`DB close connection`

Add your title





# Libuv

## Non-Blocking I/O

**Network I/O**

**IOCP**

**epoll**

**kqueue**

**Windows**

**Linux**

**BSD**

## Blocking I/O

**File I/O**

**DNS**

**etc...**

**Thread Pool**



# Threads

```
const fs = require('fs');

setInterval(() => {
  () => {console.log('Thread 1')}
}, 1000);

fs.readFile(__filename, 'utf8', () => {console.log('Thread 2')});
```

***Blocking I/O***

***CPU intensive***



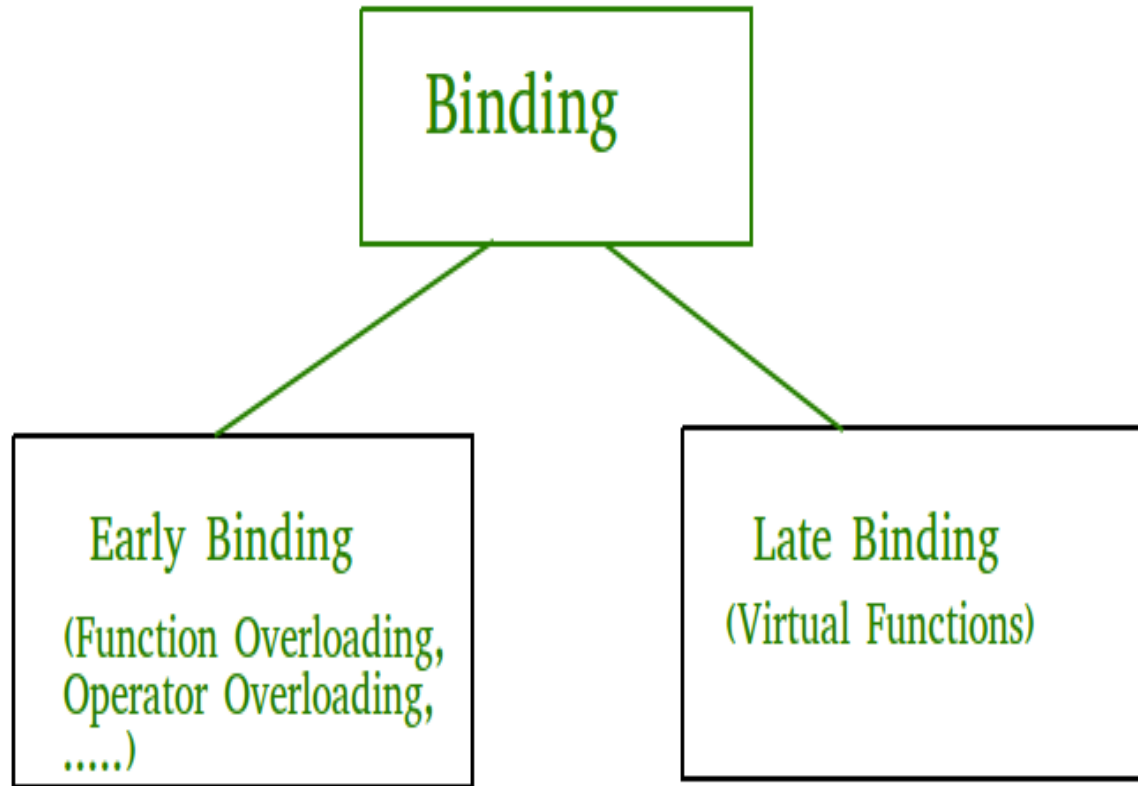
**UV\_THREADPOOL\_SIZE = 8    SIZE <= 128**



**Thread Pool**

*Web Workers*

# C++ Bindings



Binding refers to the process of converting identifiers (such as variable and performance names) into addresses. Binding is done for each variable and functions. For functions, it means that matching the call with the right function definition by the compiler. It takes place either at compile time or at runtime.

---

## **E**arly Binding (compile-time time polymorphism)

As the name indicates, compiler (or linker) directly associate an address to the function call. It replaces the call with a machine language instruction that tells the mainframe to leap to the address of the function.

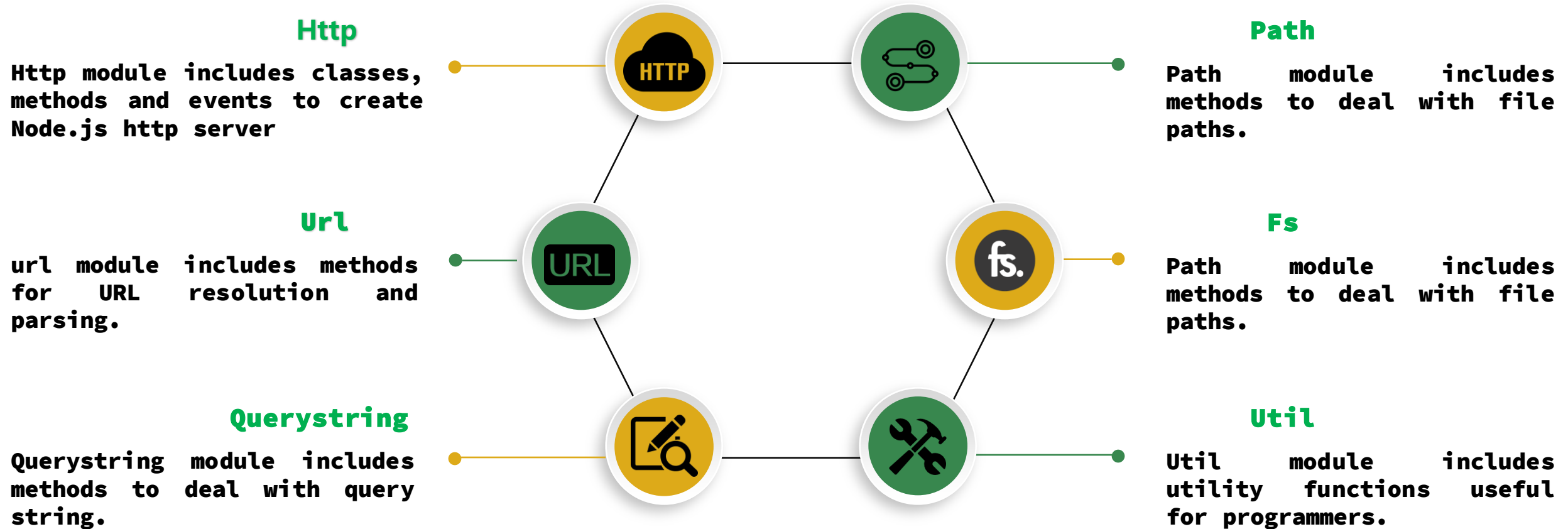
## **L**ate Binding

In this, the compiler adds code that identifies the kind of object at runtime then matches the call with the right function definition (Refer this for details). This can be achieved by declaring a virtual function.

# Core Modules

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope.



Also, each module can be placed in a separate .js file under a separate folder.

Node.js implements CommonJS modules standard. CommonJS is a group of volunteers who define JavaScript standards for web server, desktop, and console application.

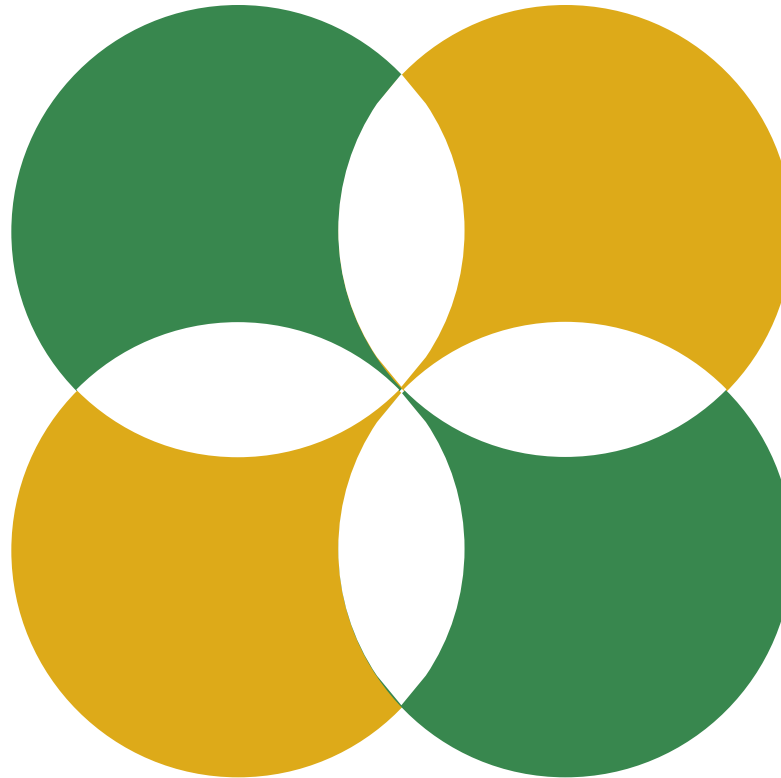
# C-ares ZLib

## C-ares

is a C library for asynchronous DNS requests (including name resolves). C89 compatibility, [MIT licensed](#), builds for and runs on POSIX, Windows, Netware, Android and many more operating systems

## ZLib

zlib is a software library used for data compression. zlib was written by [Jean-loup Gailly](#) and Mark Adler and is an abstraction of the DEFLATE compression algorithm used in their gzip file compression program. zlib is also a crucial component of many software platforms, including Linux, OSX, and iOS.



# HTTP Parser. OpenSSL

## HTTP Parser

This is a parser for HTTP messages written in C. It parses both requests and responses. The parser is designed to be used in performance HTTP applications. It does not make any syscalls nor allocations, it does not buffer data, it can be interrupted at anytime. Depending on your architecture, it only requires about 40 bytes of data per message stream (in a web server that is per connection).

## OpenSSL

OpenSSL is an open-source command line tool that is commonly used to generate private keys, create CSRs, install your SSL/TLS certificate, and identify certificate information. We designed this quick reference guide to help you understand the most common OpenSSL commands and how to use them.

# Javascript V8 engine

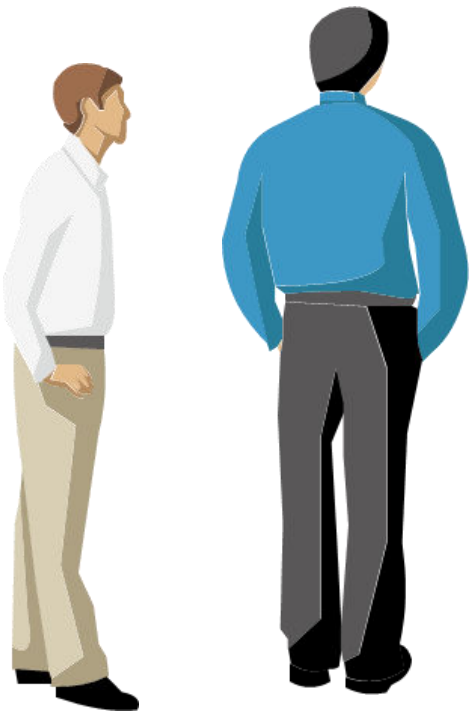
## What is javascript engine ?

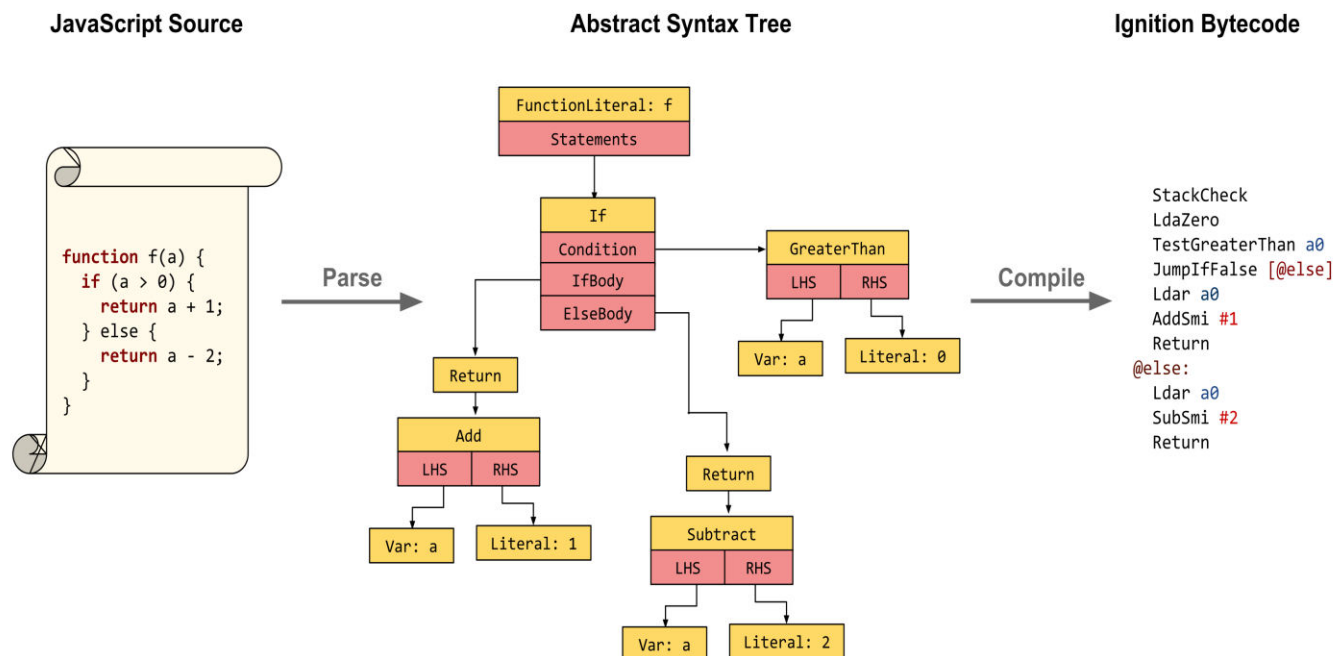
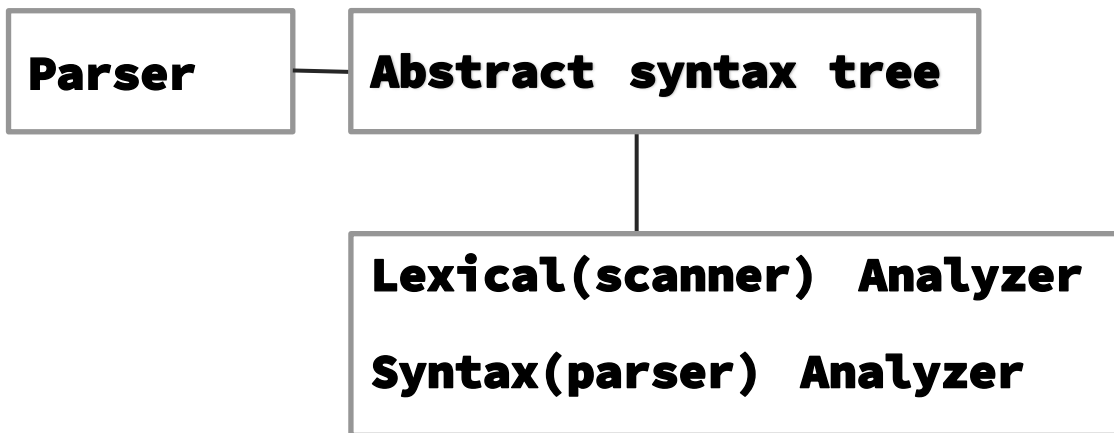


**V8 is Google's open source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Chrome and in Node.js, among others.**

**This documentation is aimed at C++ developers who want to use V8 in their applications, as well as anyone interested in V8's design and performance. This document introduces you to V8, while the remaining documentation shows you how to use V8 in your code and describes some of its design details, as well as providing a set of JavaScript benchmarks for measuring V8's performance. V8 implements ECMAScript and WebAssembly, and runs on Windows 7 or later, macOS 10.12+, and Linux systems that use x64, IA-32, ARM, or MIPS processors. V8 can run standalone, or can be embedded into any C++ application. V8 compiles and executes JavaScript source code, handles memory allocation for objects, and garbage collects objects it no longer needs.**

# Javascript engine



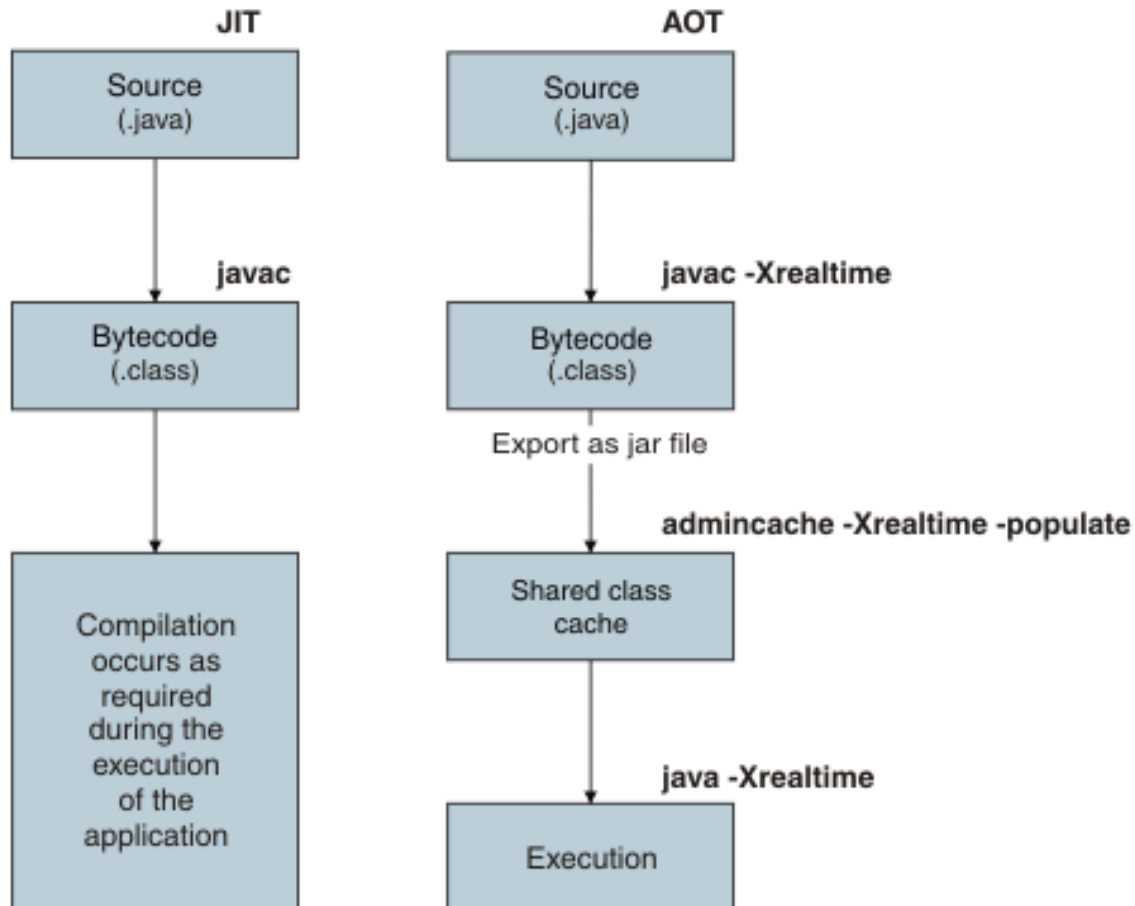




# Compilation

## Ahead of Time

### Static translation



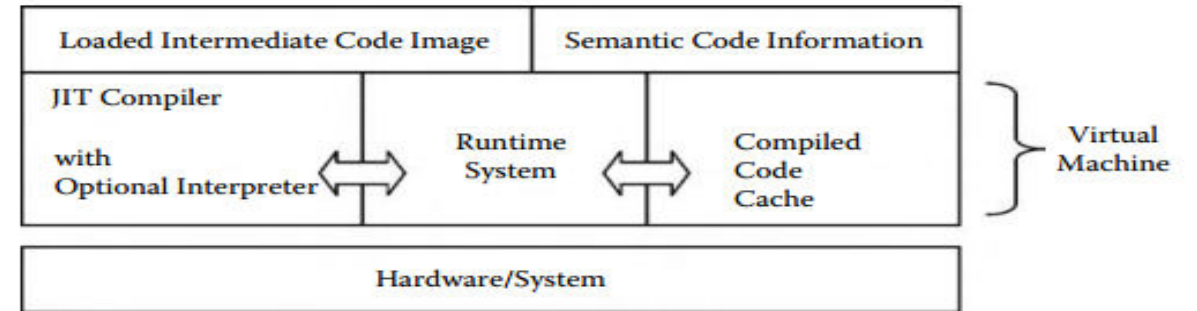
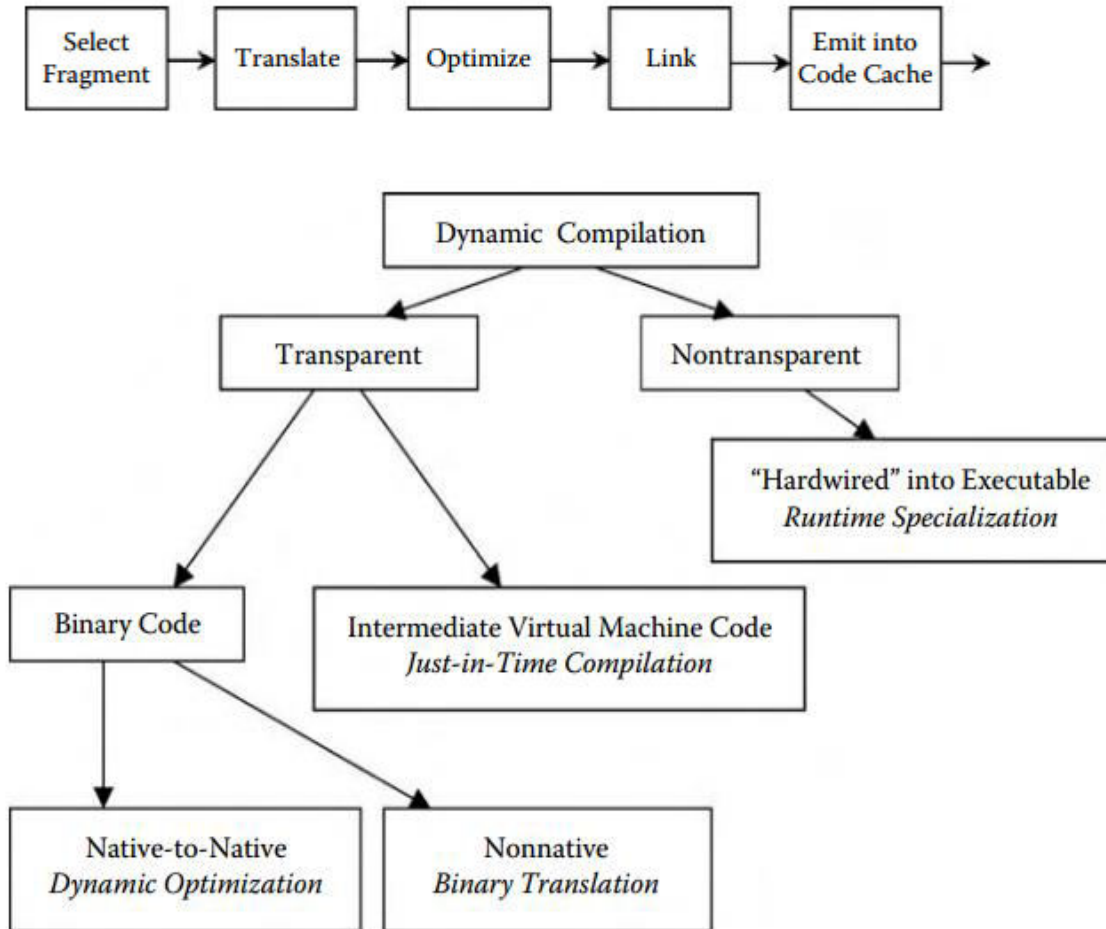
**AOT is the act of compiling a higher-level programming language such as C or C++, or an intermediate representation such as Java bytecode or .NET Framework Common Intermediate Language (CIL) code, into a native (system-dependent) machine code so that the resulting binary file can execute natively.**

**AOT produces machine optimized code, just like a standard native compiler. The difference is that AOT transforms the bytecode of an extant virtual machine (VM) into machine code.**

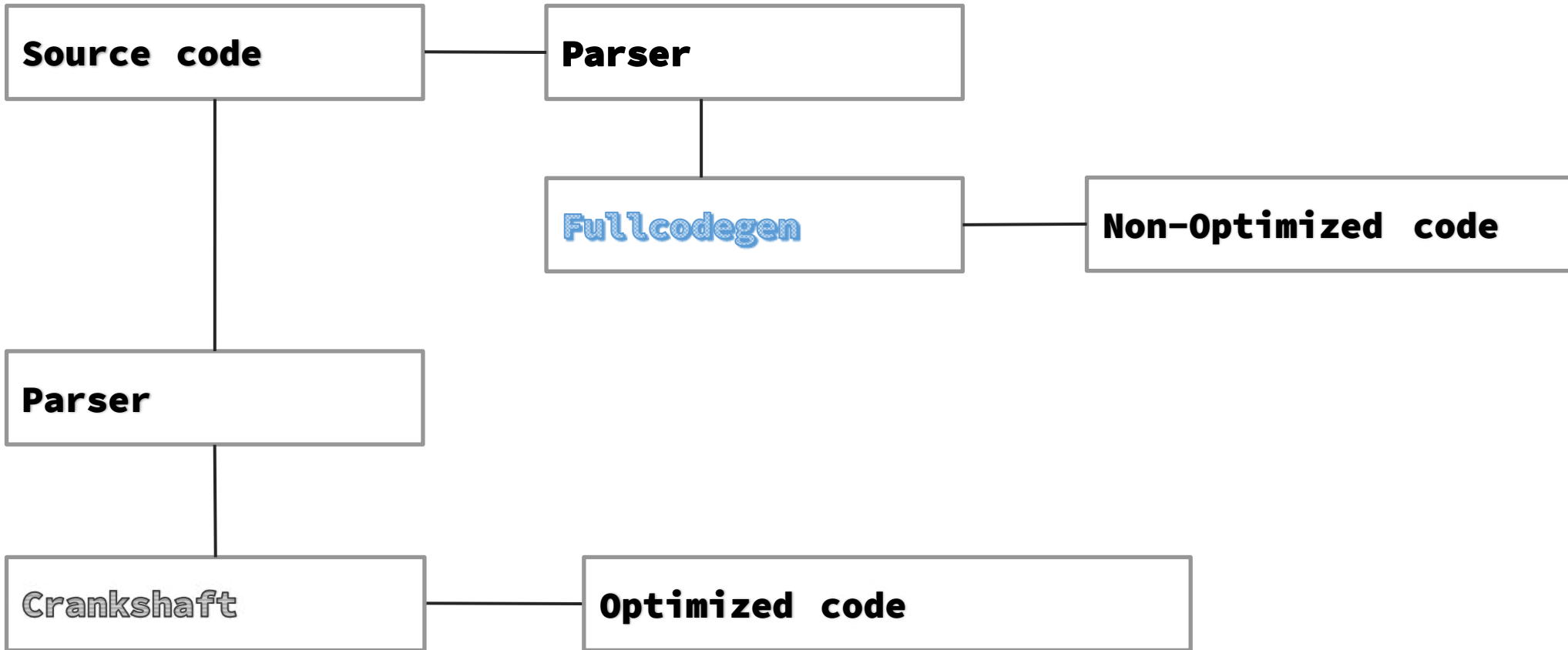
# Compilation

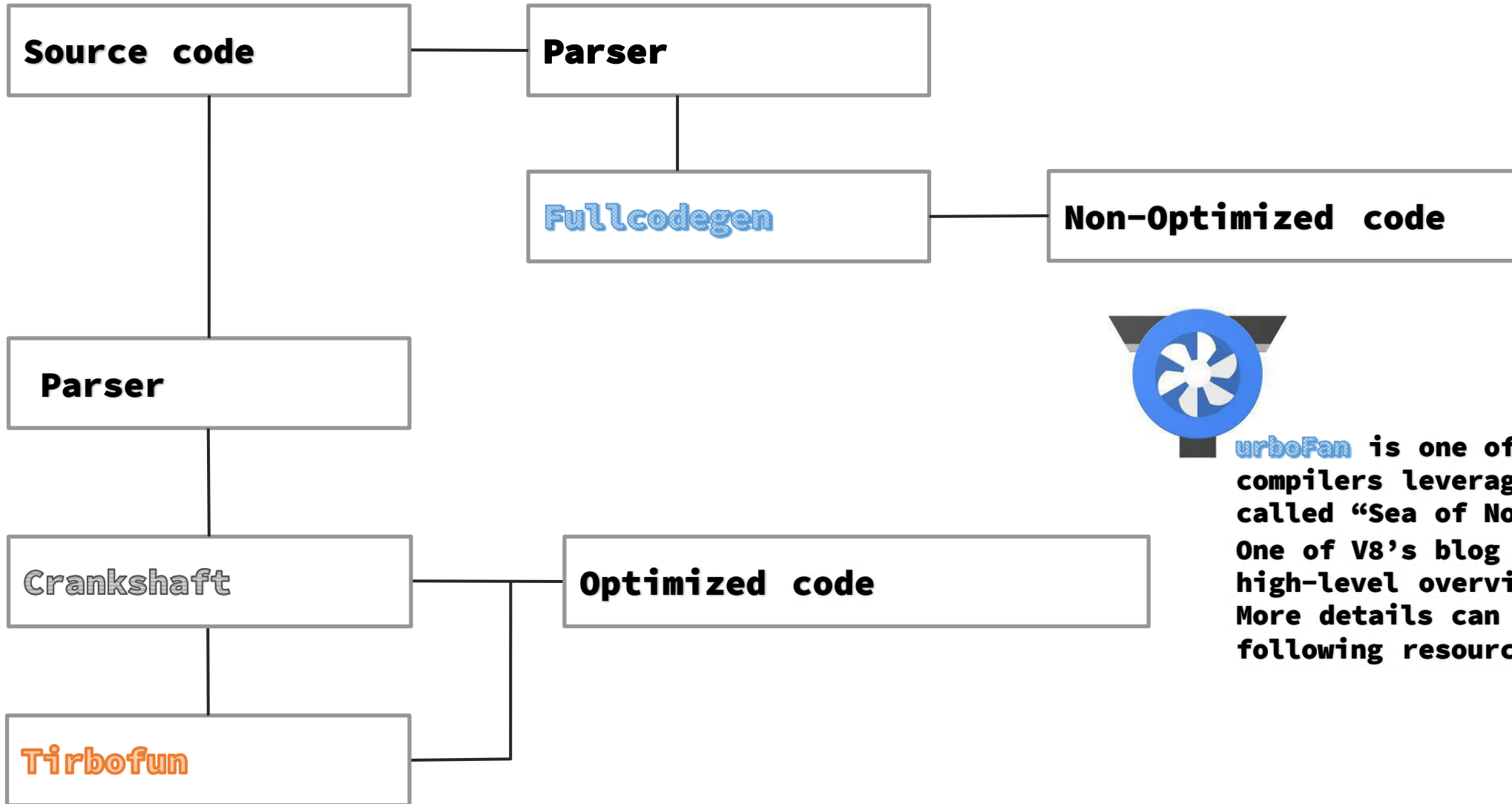
## Just in Time

Dynamic translation or Run-time compilations



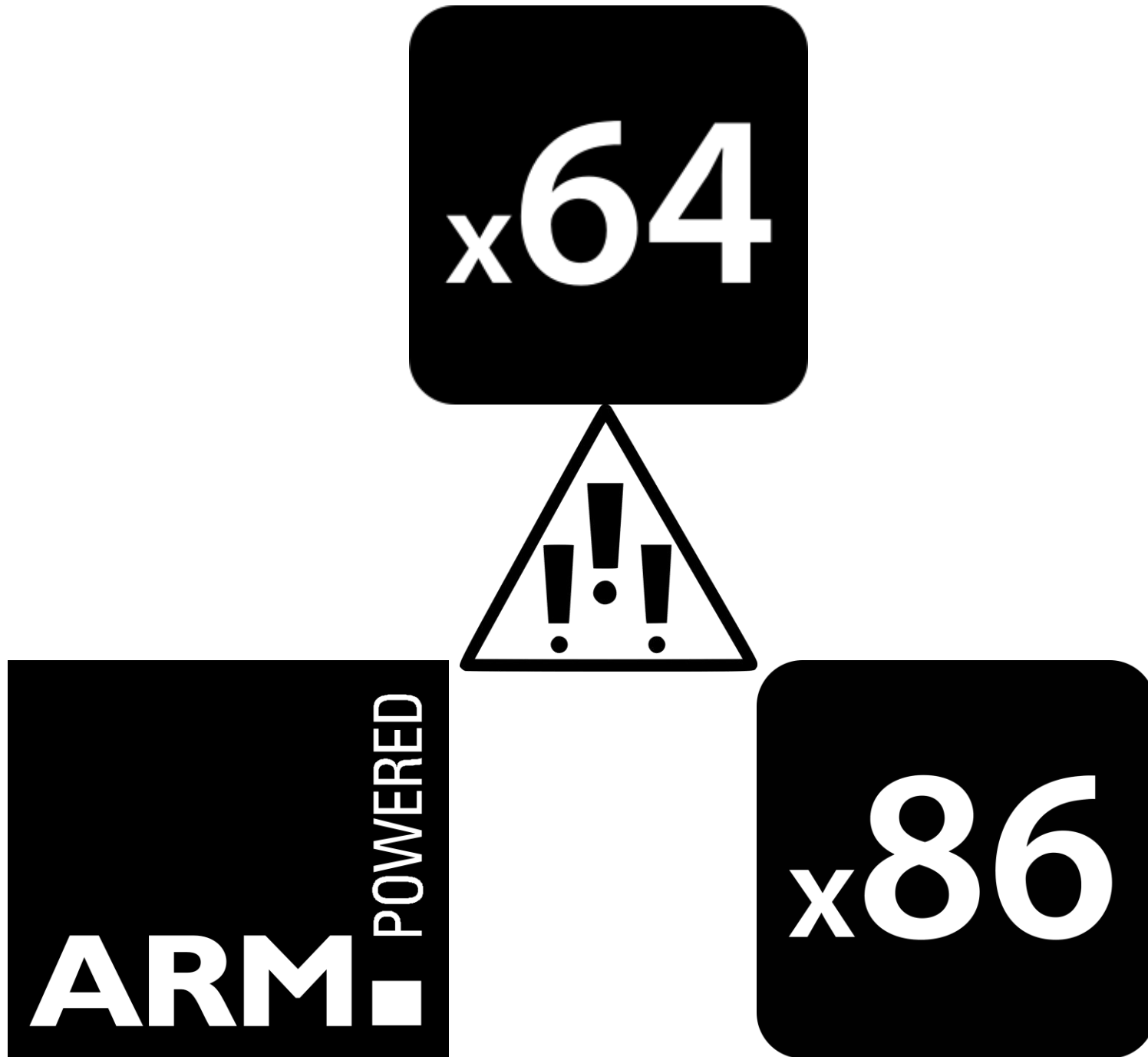
**JIT** is a way of executing computer code that involves compilation during execution of a program - at run time - rather than prior to execution. Most often, this consists of source code or more commonly bytecode translation to machine code, which is then executed directly. A system implementing a JIT compiler typically continuously analyses the code being executed and identifies parts of the code where the speedup gained from compilation or recompilation would outweigh the overhead of compiling that code.

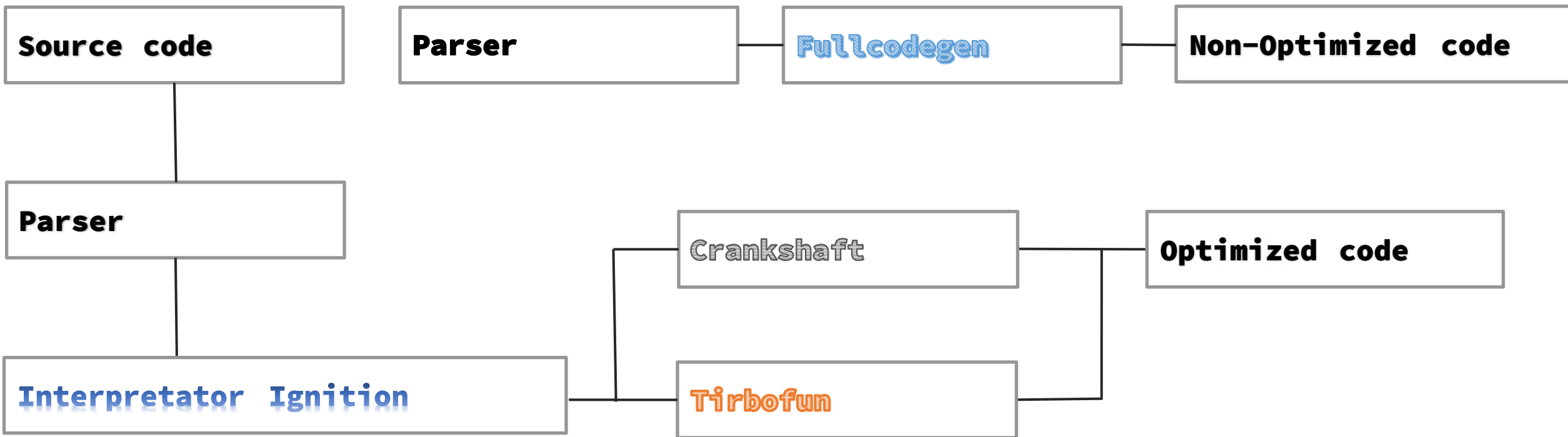




**urboFan** is one of V8's optimizing compilers leveraging a concept called "Sea of Nodes".

One of V8's blog posts offers a high-level overview of TurboFan. More details can be found in the following resources.





V8 features an interpreter called Ignition. Ignition is a fast low-level register-based interpreter written using the backend of TurboFan. The V8 blog post presents a high-level overview of the Ignition interpreter. More details can be found in the following resources





**Source code**

**Parser**

**Ignition**

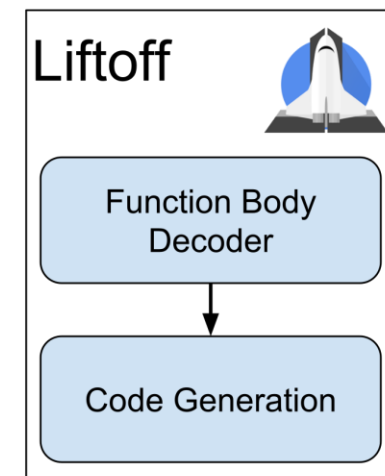
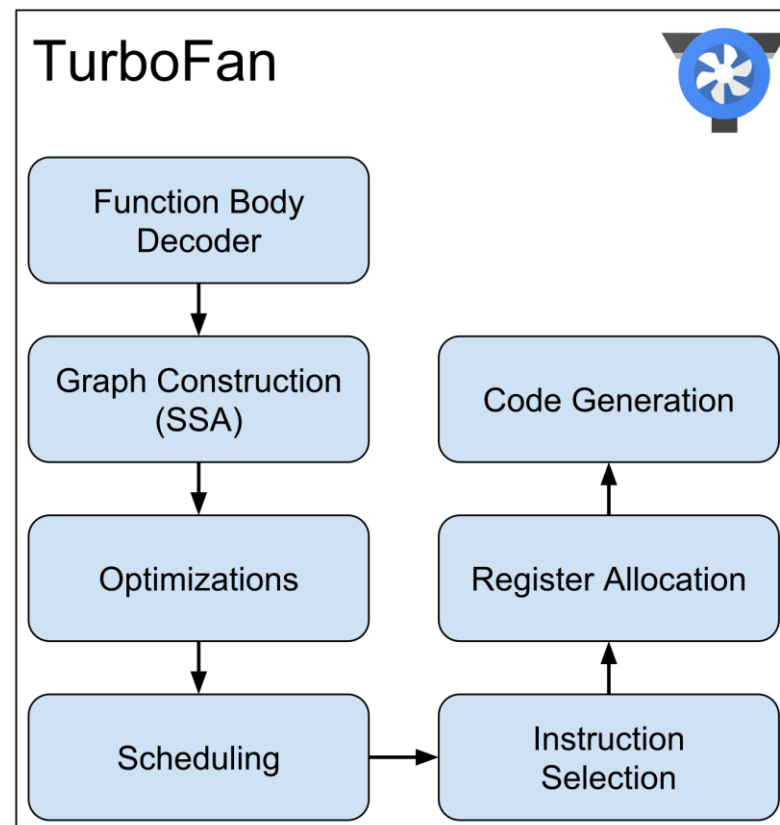
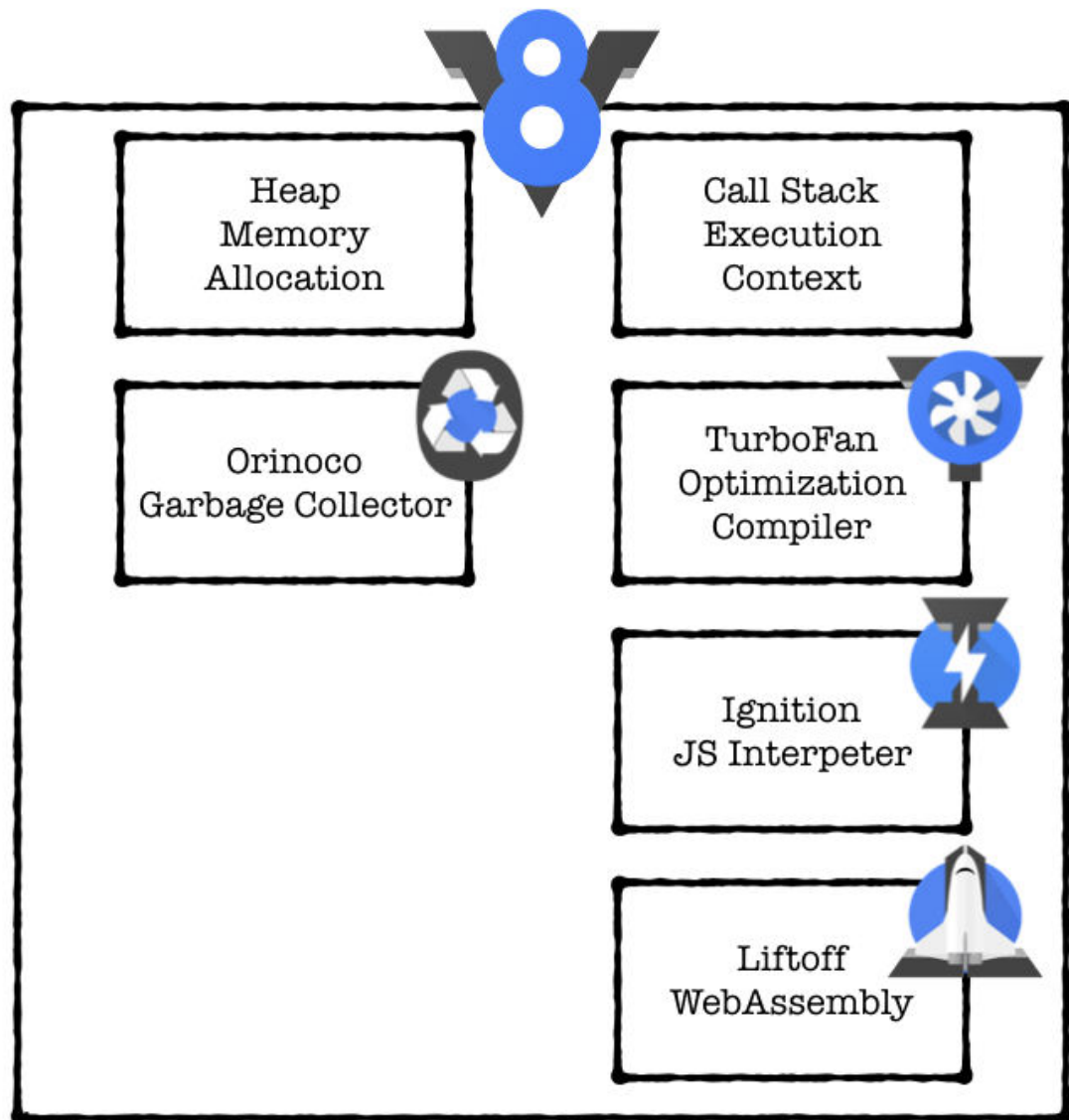


**Tirbofun**



**Optimized Code**





## Liftoff



V8 v6.9 includes Liftoff, a new baseline compiler for WebAssembly. Liftoff is now enabled by default on desktop systems. This article details the motivation to add another compilation tier and describes the implementation and performance of Liftoff. But why does it take this long to start up a WebAssembly app, if similar JS apps start up much faster? The reason is that WebAssembly promises to deliver predictable performance, so once the app is running, you can be sure to consistently meet your performance goals (e.g. rendering 60 frames per second, no audio lag or artifacts...). In order to achieve this, WebAssembly code is compiled ahead of time in V8, to avoid any compilation pause introduced by a just-in-time compiler that could result in visible jank in the app.

## Orinoco



Running over the memory heap, looking for disconnected memory allocations is the Orinoco. Implementing a generational garbage collector, moving objects within the young generation, from the young to the old generation, and within the old generation. These moves leave holes, and Orinoco performs both evacuation and compaction to free space for more objects.

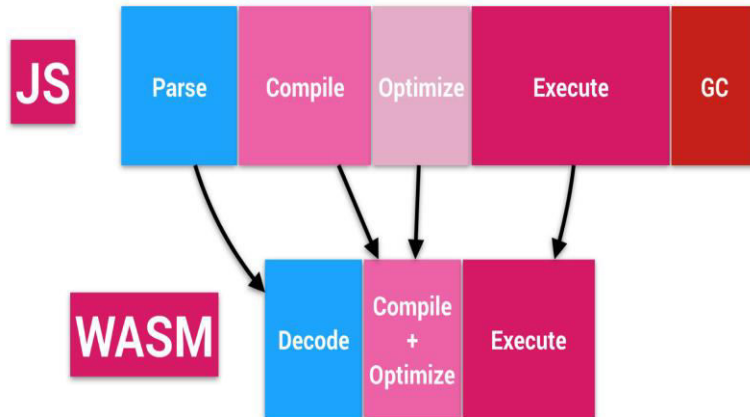
Another optimization performed by Orinoco is in the way it searches through the heap to find all pointers that contain the old location of the objects moved and update them with the new location. This is made using a data structure called remembered set. On top of these, black allocation is added, which basically means the garbage collection process automatically marks living objects in black in order to speed up the iterative marking process.

## WebAssembly (WASM)

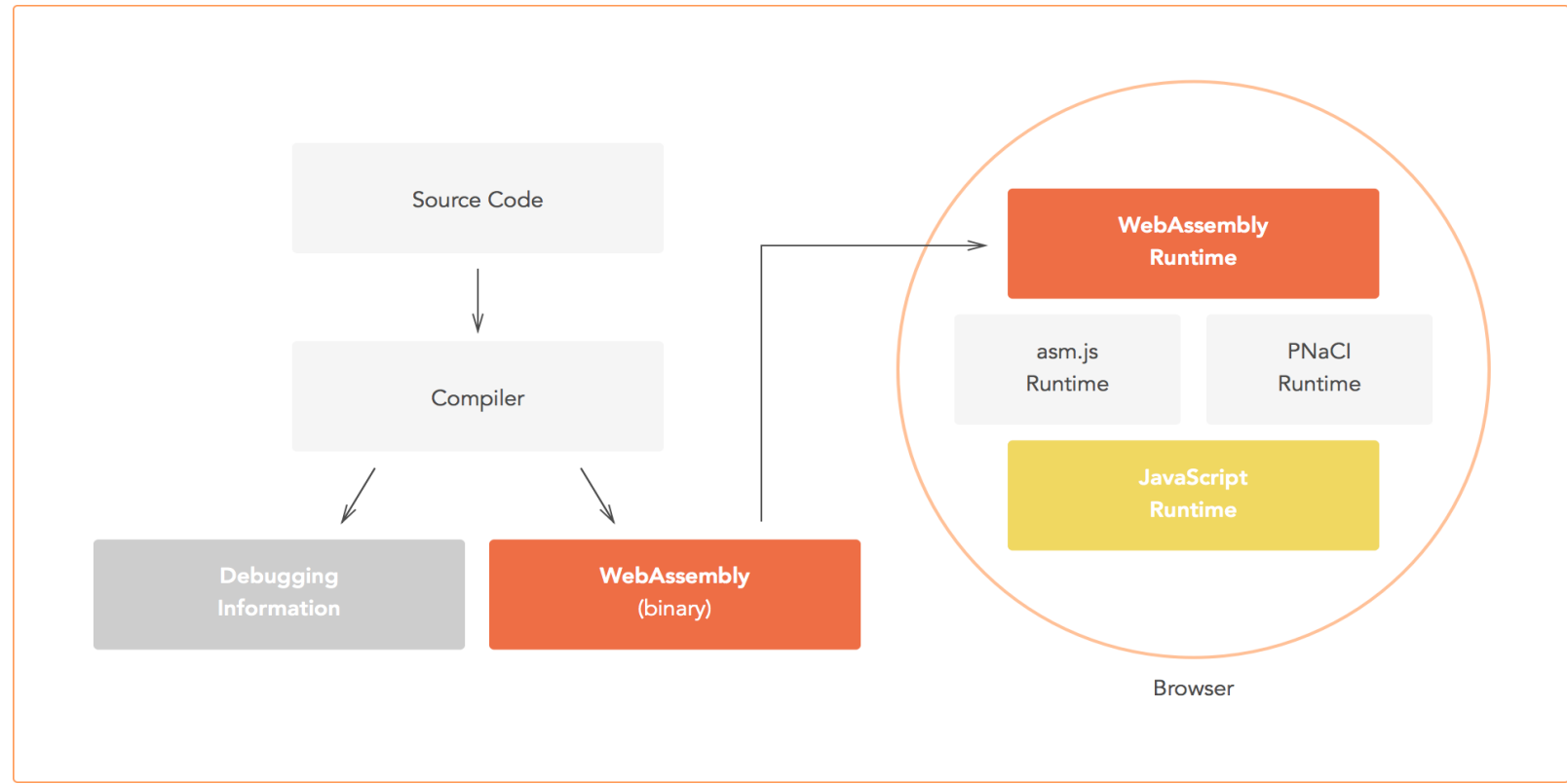


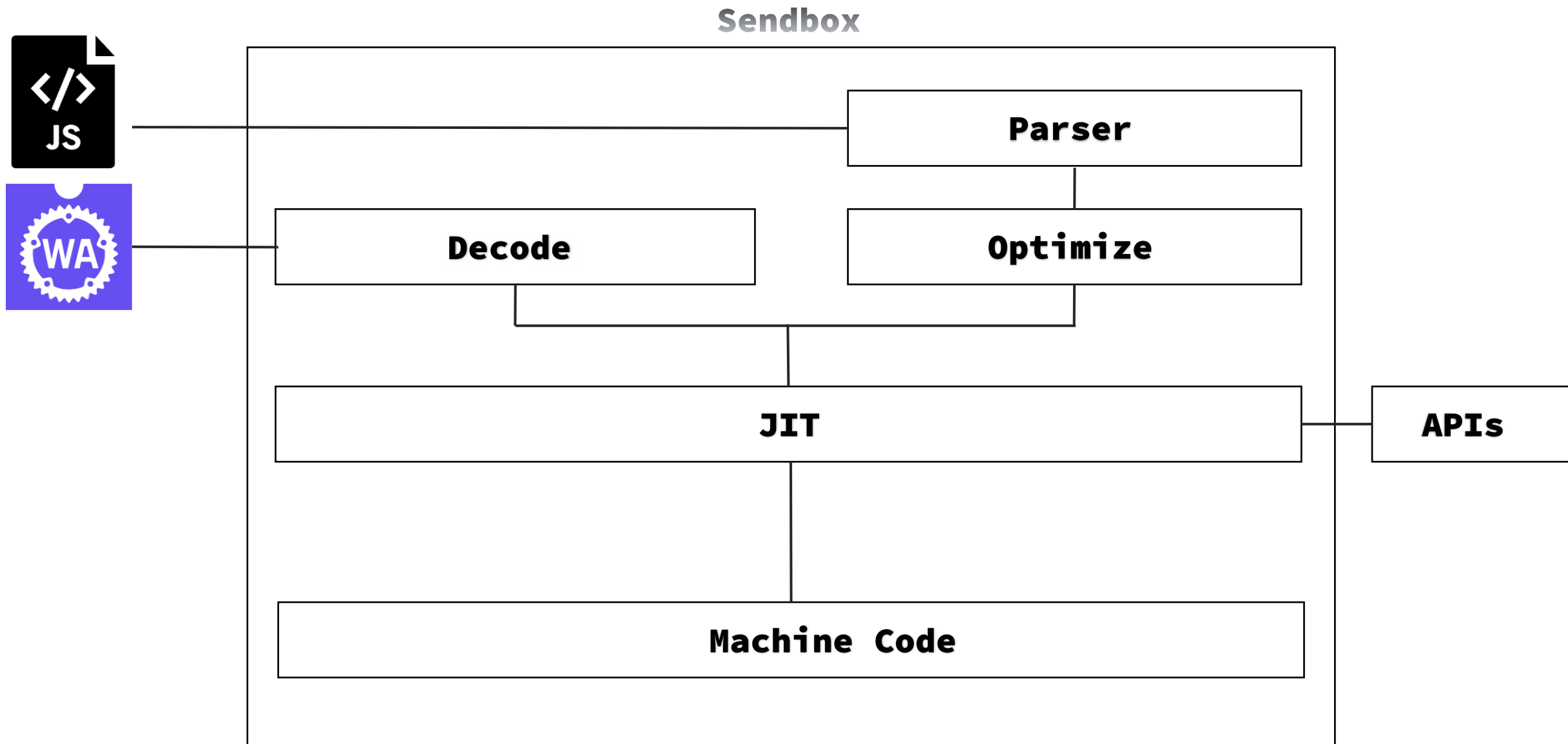
# WEBASSEMBLY

<https://webassembly.org>



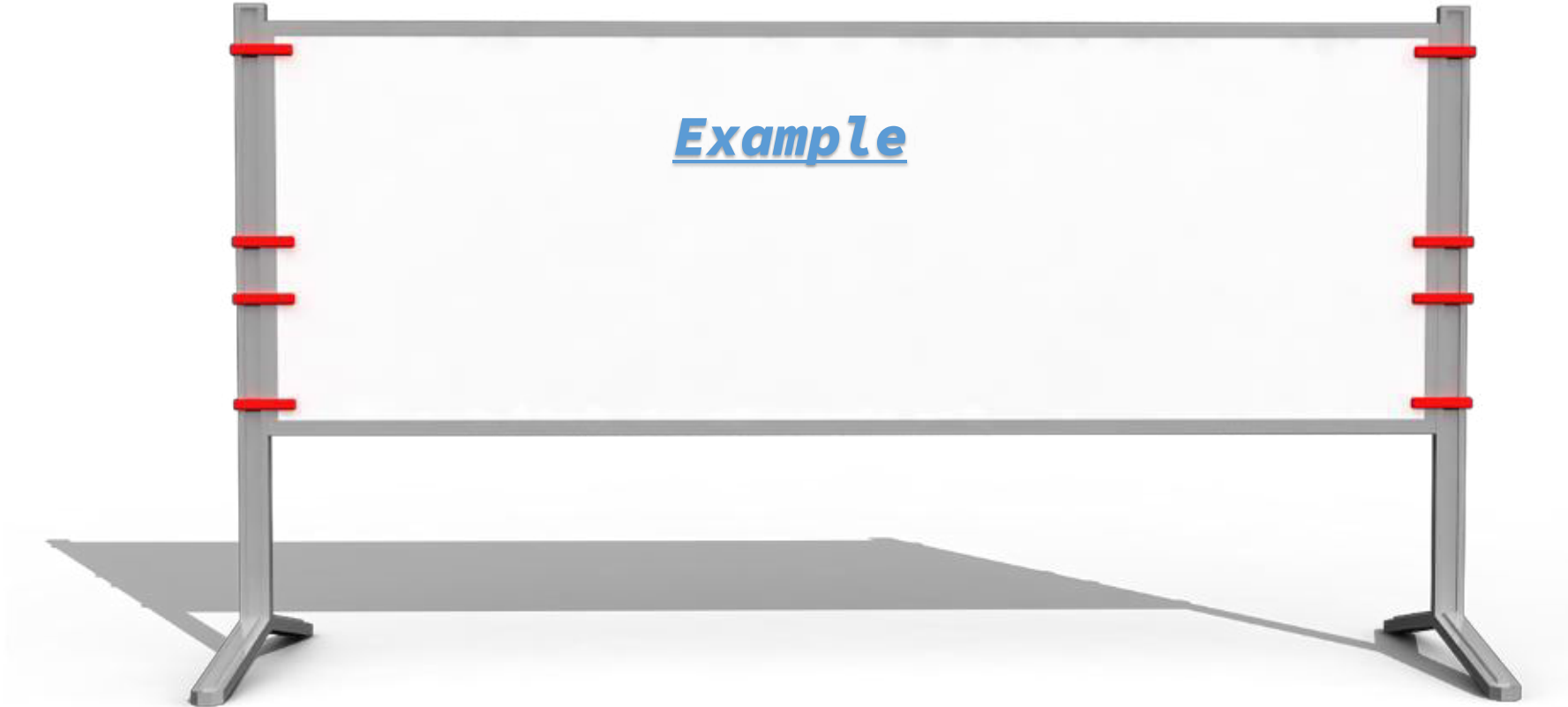
**WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.**





# Microtask and Macrotask

Example





**Thank you!**

