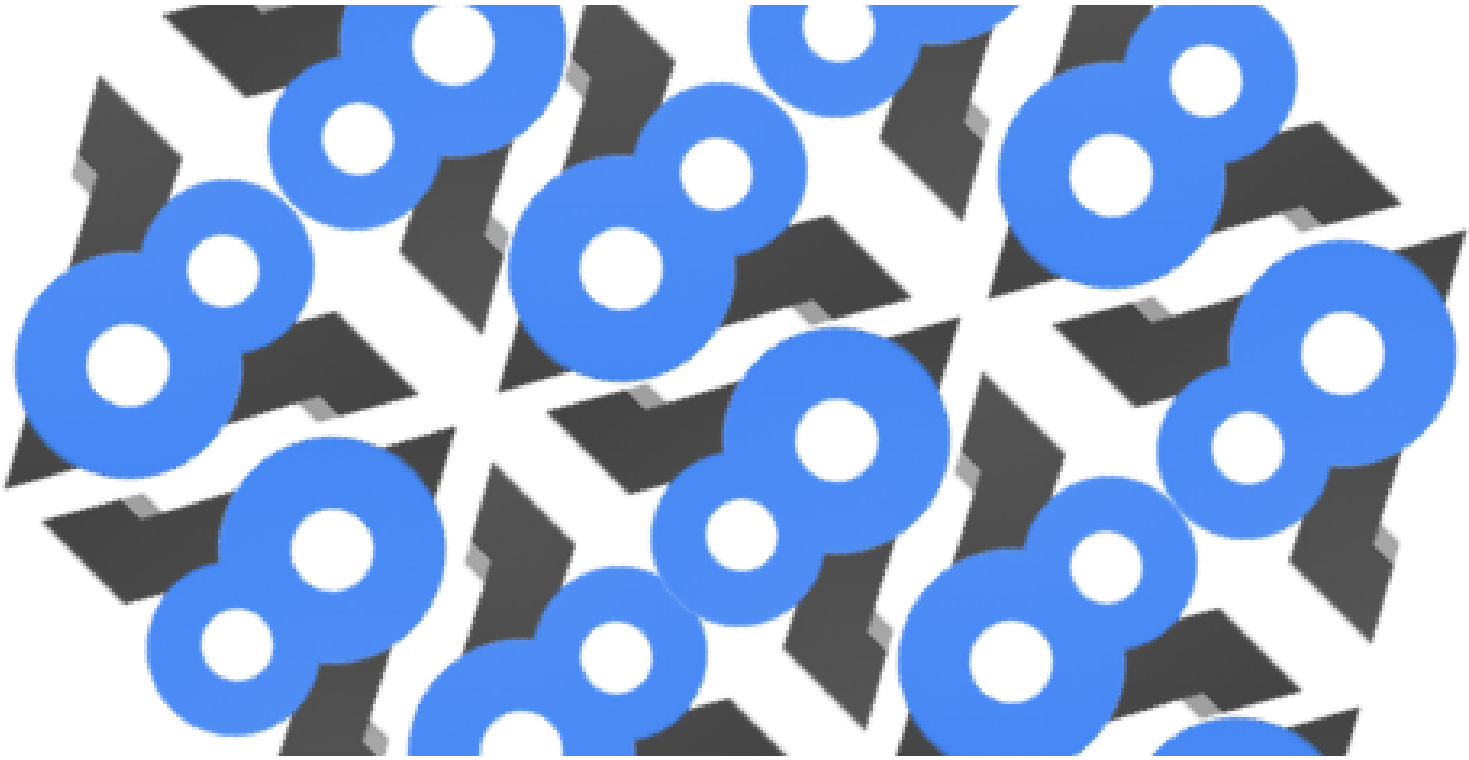
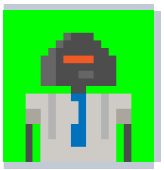
JavaScript V8 Engine Explained

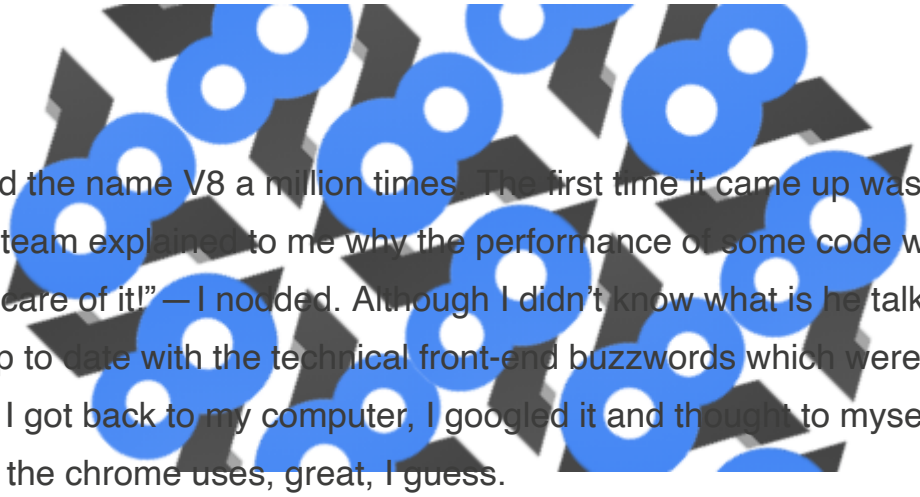
Originally published by Yotam Kadishay on January 12th 2019 28,252 reads





@kadishay

Yotam Kadishay



Well, I think I heard the name V8 a million times. The first time it came up was at 2008, when an engineer from my team explained to me why the performance of some code would be ok — he said: “V8 will take care of it!” — I nodded. Although I didn’t know what is he talking about, I still wanted to seem up to date with the technical front-end buzzwords which were flooding us these days. Then, when I got back to my computer, I googled it and thought to myself — cool, new JavaScript engine the chrome uses, great, I guess.

This first line on Wikipedia is what most of us know about V8, and about lots of other things. Here I’ll try to provide a simple explanation of what V8 actually does. As for the other things, next time, just read the entire first paragraph in Wikipedia, what the hack, you only live once, dive into the second one.

So yes, “V8 is Google’s open source high-performance JavaScript and WebAssembly engine, written in C++” (V8 documentation) but what does this actually mean? Well, actually it means, V8 is a C++ program, which receives JavaScript code, compiles, and executes it.

V8 Does:

1. Compiles and executes JS code
2. Handling call stack — running your JS functions in some order
3. Managing memory allocation for objects — the memory heap
4. Garbage collection — of objects which are no longer in use
5. Provide all the data types, operators, objects and functions

V8 Can:

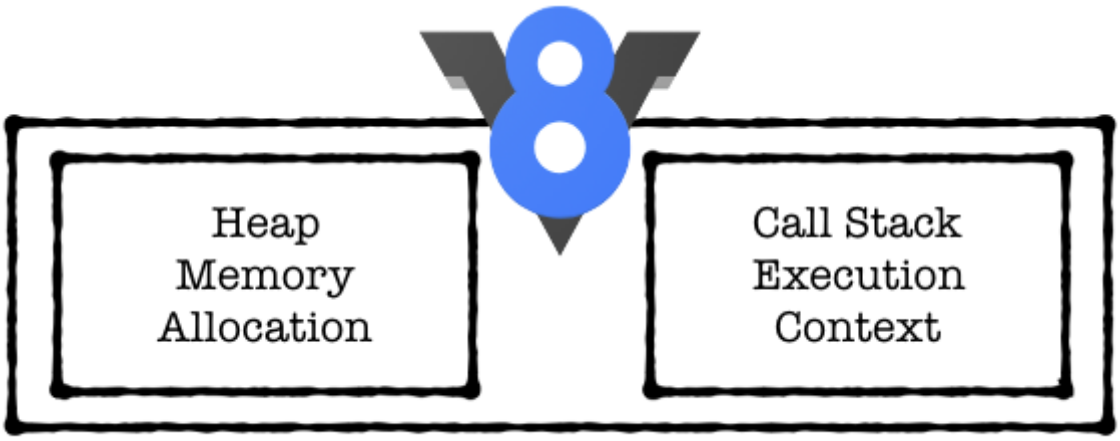
1. Provide the event loop, but this is sometimes implemented by the browser as well

V8 Doesn't:

1. Know anything about the Document Object Model (DOM) — which is provided by the browser, and obviously irrelevant to Node.js for example

V8 is a single threaded execution engine. It's built to run exactly one thread per JavaScript execution context. You can actually run two V8 engines in the same process — e.g. web-workers, but they won't share any variables or context like real threads. This doesn't mean V8 is running on a single thread, but it does mean it provides a JavaScript flow of a single thread.

On the runtime, V8 is mainly managing the heap memory allocation and the single threaded call stack. The call stack is mainly a list of function to execute, by calling order. Every function which calls another function will be inserted one after the other directly, and callbacks will be sent to the end. This is actually why calling a function with setTimeout of zero milliseconds sends it to the end of the current line and doesn't call it straight away (0 milliseconds).



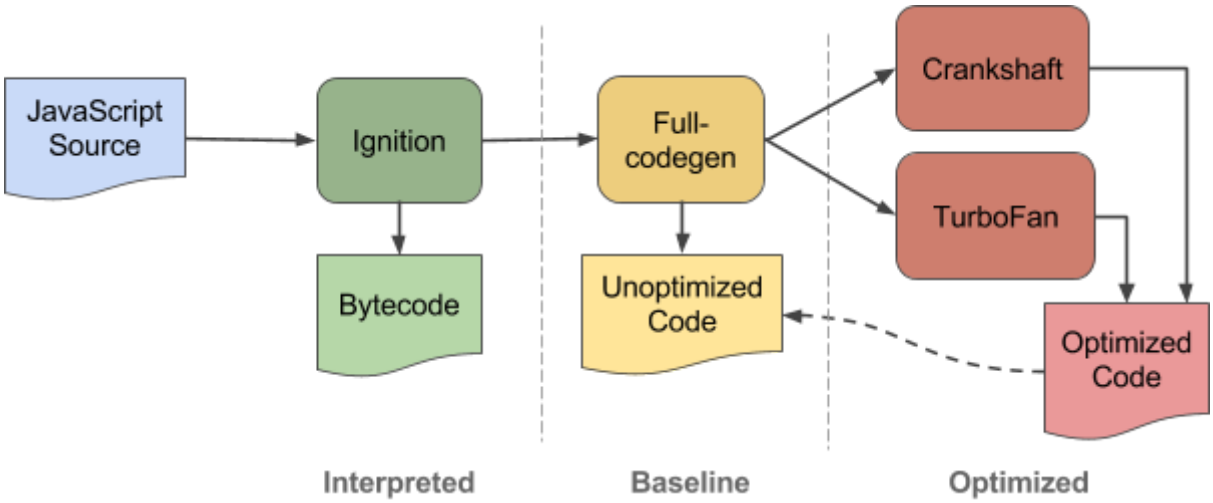
Other Key Components:

JS Interpreter — Ignition & Optimization Compiler — TurboFan & Crankshaft

V8 gets its speed from just-in-time (JIT) compilation of JavaScript to native machine code, just before executing it. First of all, the code is compiled by a baseline compiler, which quickly generates non-optimized machine code. On runtime, the compiled code is analyzed and can be re-compiled for optimal performance. Ignition provides the first while TruboFan & Crankshaft the second.

JIT compilation result machine code can take a large amount of memory, while it might be executed once. This is solved by the Ignition, which is executing code with less memory overhead.

The TurboFan project started in 2013 to improve the weakness of Crankshaft which isn't optimized for some part of the JavaScript functionality e.g. error handling. It was designed for optimizing both existing and future planned features at the time.



V8 Dev Blog — https://v8.dev/blog/ignition-interpreter

WebAssembly — Liftoff

Achieving great performance is also key in the browser, and this is the task Liftoff is used for — generating machine code. Not using the complex multi-tier compilation, Liftoff is a simpler code generator, which generates code for each opcode (a single portion of machine code, specifying an operation to be performed) at a time. Liftoff generates code much faster than TurboFan (~10x) which is obviously less performant (~50%). To read more, see the V8 Dev Blog.

Garbage Collection — Orinoco

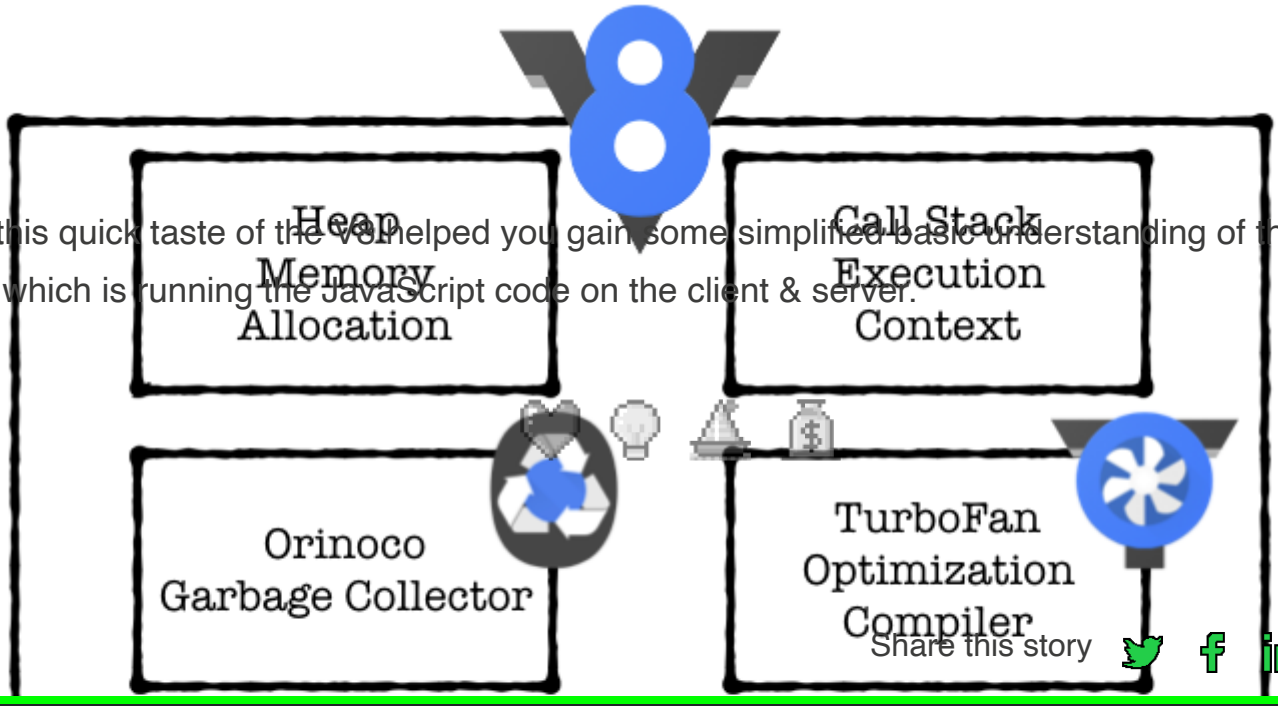
Running over the memory heap, looking for disconnected memory allocations is the Orinoco. Implementing a generational garbage collector, moving objects within the young generation, from the young to the old generation, and within the old generation. These moves leave holes, and Orinoco performs both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

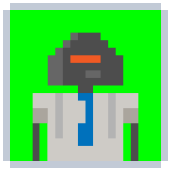
Conclusion

JavaScript is not aiming to be the most optimized server-side language for high scale and throughput. Nevertheless, since the introduction of V8 and the above architectural improvements, The set of tools a web developer can use transformed completely, enabling huge improvements and new features.



I hope this quick taste of the V8 helped you gain some simplified basic understanding of the V8 engine which is running the JavaScript code on the client & server.

Share this story



@kadishay

Yotam Kadishay

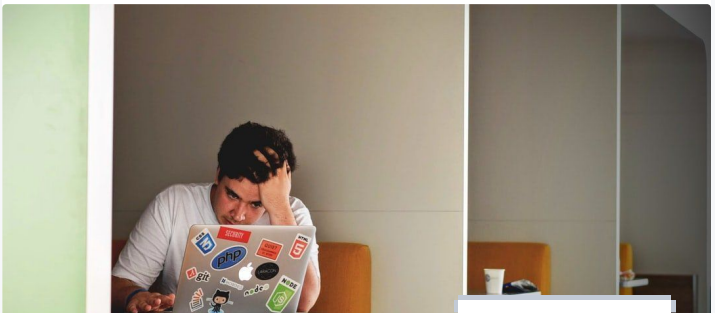
Read my stories

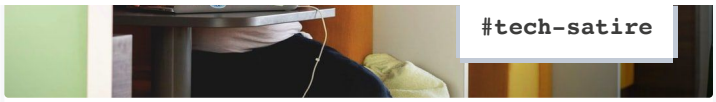
RELATED

Create Polished React Apps Much Faster - Hire a UI Library!

I Wish I Never Learned to Code

1 reaction





#tech-satire

@thawkin3
Tyler Hawkins
10/23/20

Introducing FBSQL: Frontend Backend SQL Server

1 reaction

#frontend-backend

@fbsql
FBSQL
10/18/20

TAGS

#javascript #v8 #compilers #javascript-v8-engine #v8-engine