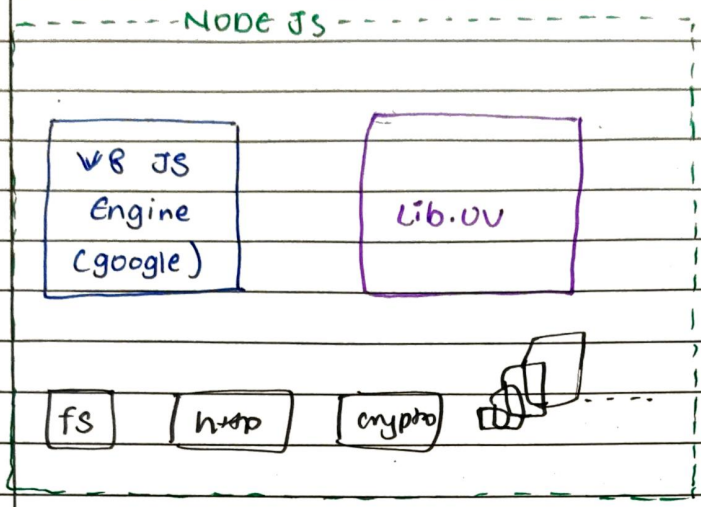
10/12/25

Deep dive into V8 JS Engine



⇒ CODE

V8

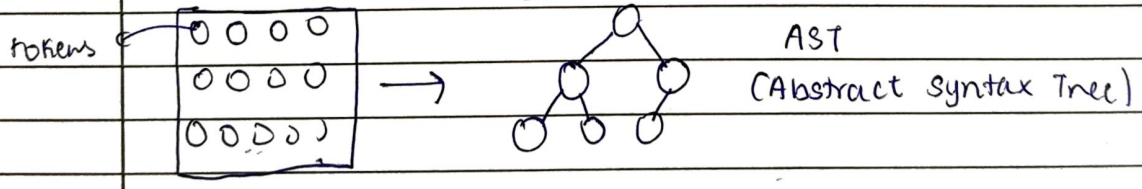
Ⓐ PARSING

① Lexical Analysis (Tokenization)

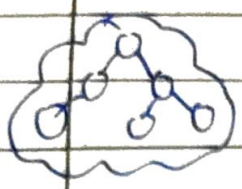
code → Tokens (Lexical tokens)

② Syntax Analysis : (Tokens → AST)

↳ (Parsing)



⇒ website ⇒ astexplorer.net



→ 2 Types of Language :

Interpretted	Complied
• Line by line	→ first compilation
• fast intial execution.	code → M. code
• Interpretter	→ initially heavy but executed fast.
	→ complier.

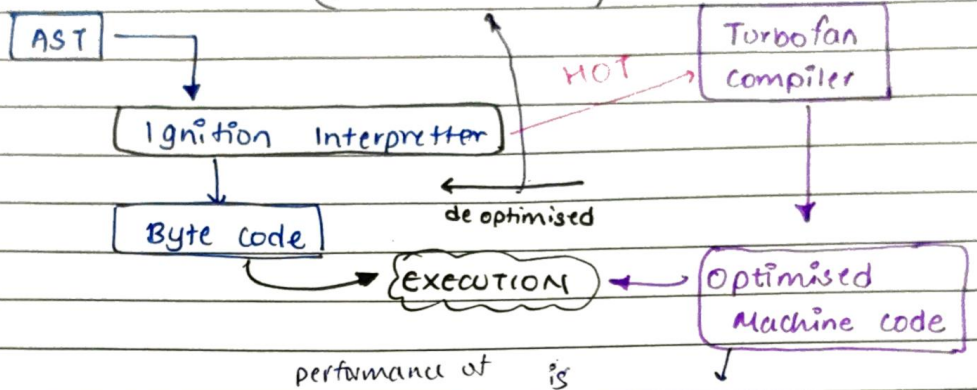
⇒ Javascript :

is both interpretted and JIT - complied
(Just-In-Time compilation)

↳ Not Ahead-Of-Time (AOT) complied

__/__/__

JIT complication (if assumption break) (optimization)



performance of is
This is why JS ~~can~~ ~~much~~ faster in modern browsers

⇒ When the AST gives the code to intrepretter it find out which part of the code which is repeated alot. That portion of code is given to turbofan compiler to optimize it

- makes assumption
  - → sum(10,5)
  - → sum(2,5)
  - → sum(7,6)

  } ye generally numbers hai aur compiler ye code ko optimized karega.

→ sum("a","b") ⟶ is mai deoptimisation ~~hota hai~~ hoga kyuki vo optimised code numbers ke liye tha aur ye strings hai toh ye optimised code deoptimise hoga. fir Ignition call karega sum function ko aur sum("a","b") hoga.

- V8 process JS (parsing → bytecode → optimisation → deoptimisation)

__/__/__

→ Why byte code ?
- Execute faster than raw JS.
- Uses less memory than older engines' method.

* Profiling while Running ?
As code runs, V8 watches it,
- which functions run many times ?
- ~~which~~ what data types are used ?
- Are variables stable (monomorphic) or changing type ?

This info is called runtime feedback or typefeedback.
* HOT fn become candidate for optimisation.

* Inline caches
* Copy Elision

• Property accus :
→ ~~getting or setting a value stored inside a object~~

* Inline Cache (IC) :
→ is an optimisation used in V8 Javascript engine to speed up repeated property accus.

* Copy Elision : (Avoiding unnecessary object copies)
↳ purpose → Eliminate unnecessary object copies.
→ Avoid allocating and copying temporary objects.

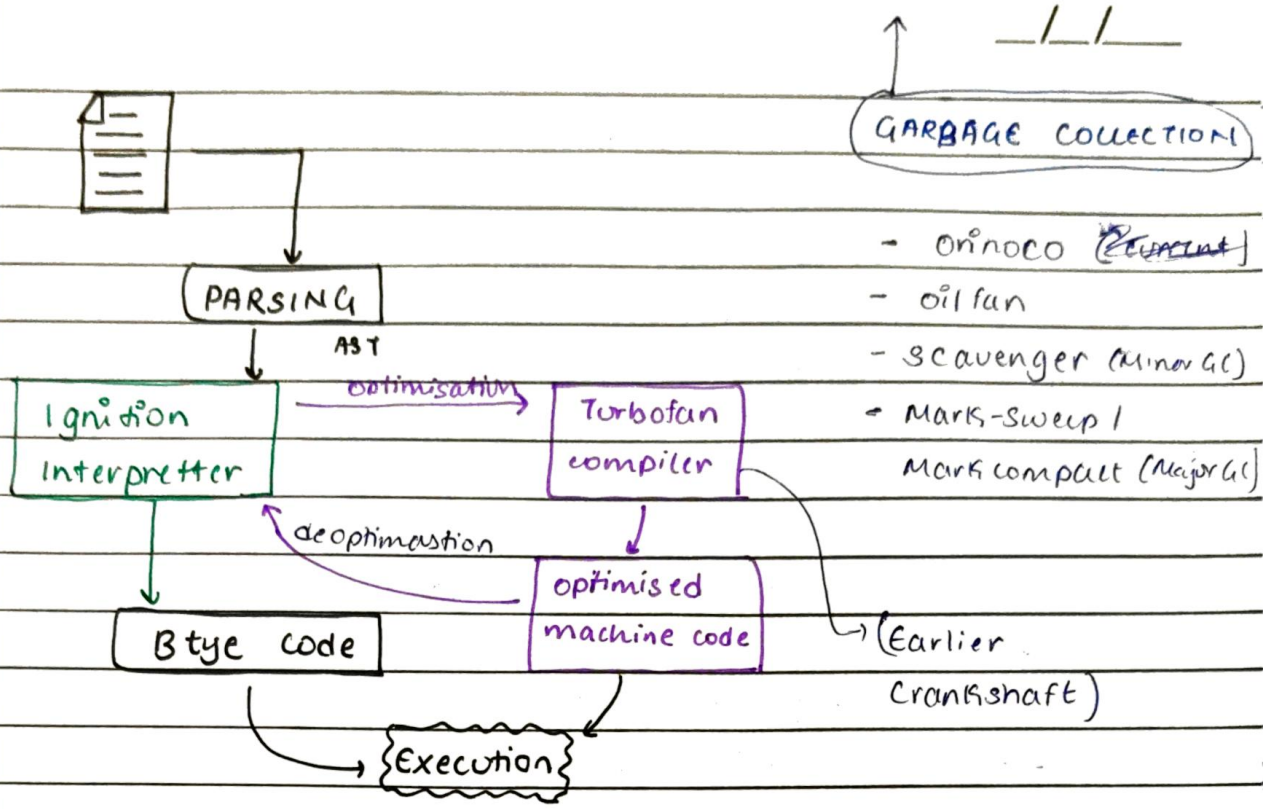
__/__/____

- **Without copy elision:**
  - Ek object ko phele ek temporrary storage ~~mai~~ banao fir usko uske final location mai dalo.
  - => Two time allocation ho rhai hai.
  - => Two memory space lagega
  - => Garbage collector ko extra kaam karna hoga.

* **With ~~eli~~ copy elision:**
  files ko unke final ~~di~~ location pe hi dall diya jata hai.

- **Ignition**
  - fast low-level register-based interpretter written using the backend of Turbofan.

overall GC architecture



GARBAGE COLLECTOR

1. Scavenger (Minor GC)
   - works on new generation (new objects)
   - uses semi-space copying algorithm
   - very fast
   - collects short-lived objects.

2. Mark-Sweep / Mark-compact (Major GC)
   - old generation (long-lived object)
   - slower but optimised
   - keeps memory compact

V8's modern GC system - Orinoco

⇒ overall V8's GC architecture.

- Introduced to improve
  - → concurrency
  - → parallelism
  - → pause time
  - → latency

- It includes improvements like:
  - → concurrent marking
  - → Parallel compaction.
  - → Incremental marking
  - → Idle-time GC.