Introduction to Javascript

Developer know html and CSS, why do we need javascript

1: We can't put C++ in the browser , it's too heavy, unsafe, and inaccessible. Our users are not kernel developers; they're web authors who just learned `<table>` and `<font>` . We need something lightweight, interpreted, forgiving, and safe.

```
#include<iostream> using namespace std; int main() { cout << "Hello World"; }
```

```
console.log("Hello World")
```

2: Massive Security Nightmare:

- **Unrestricted Access:** C++ gives you low-level control over memory and system calls. If a browser ran arbitrary C++ code from a website, that code could easily:
  - Read/write any file on your computer.
  - Install malware.
  - Access your webcam or microphone without permission.
  - Crash your entire operating system.

1. File system access

```
#include <fstream> std::ofstream file("C:\\Users\\rohit\\secrets.txt");
file << "stolen data";
```

- Without sandboxing, this code could read/write/delete any file on your machine.
- In a sandboxed environment, you'd have to **intercept all file I/O** calls and either block them or restrict them to a safe "virtual" file system.

---

2. System calls (executing programs)

```
#include <cstdlib> system("rm -rf /"); // Linux system("format C:");
// Windows 95 nightmare
```

- Raw C++ can call `system()` to run OS commands.
- Sandboxing would mean completely **disabling or trapping** such calls, otherwise a website could literally wipe your drive.

---

3. Direct memory access (pointers)

```
int* p = (int*)0xB8000; // Access video memory *p = 42;
```

- C++ allows arbitrary pointer arithmetic → could overwrite OS/kernel memory or peek into sensitive regions.
- In a sandbox, you'd have to **rewrite the runtime** so pointers never escape into raw machine addresses.

---

4. Networking

```
#include <sys/socket.h> connect(...); // Open a raw socket to exfiltrat
e data
```

- C++ can open arbitrary sockets, bypassing the browser's control.
- Sandboxing would require blocking direct socket creation and only allowing **browser-controlled HTTP requests**.

3: System Configurate was very less like 4-8mb ram, 200-400mb hard disk.

Typical Home PC Specs in 1995

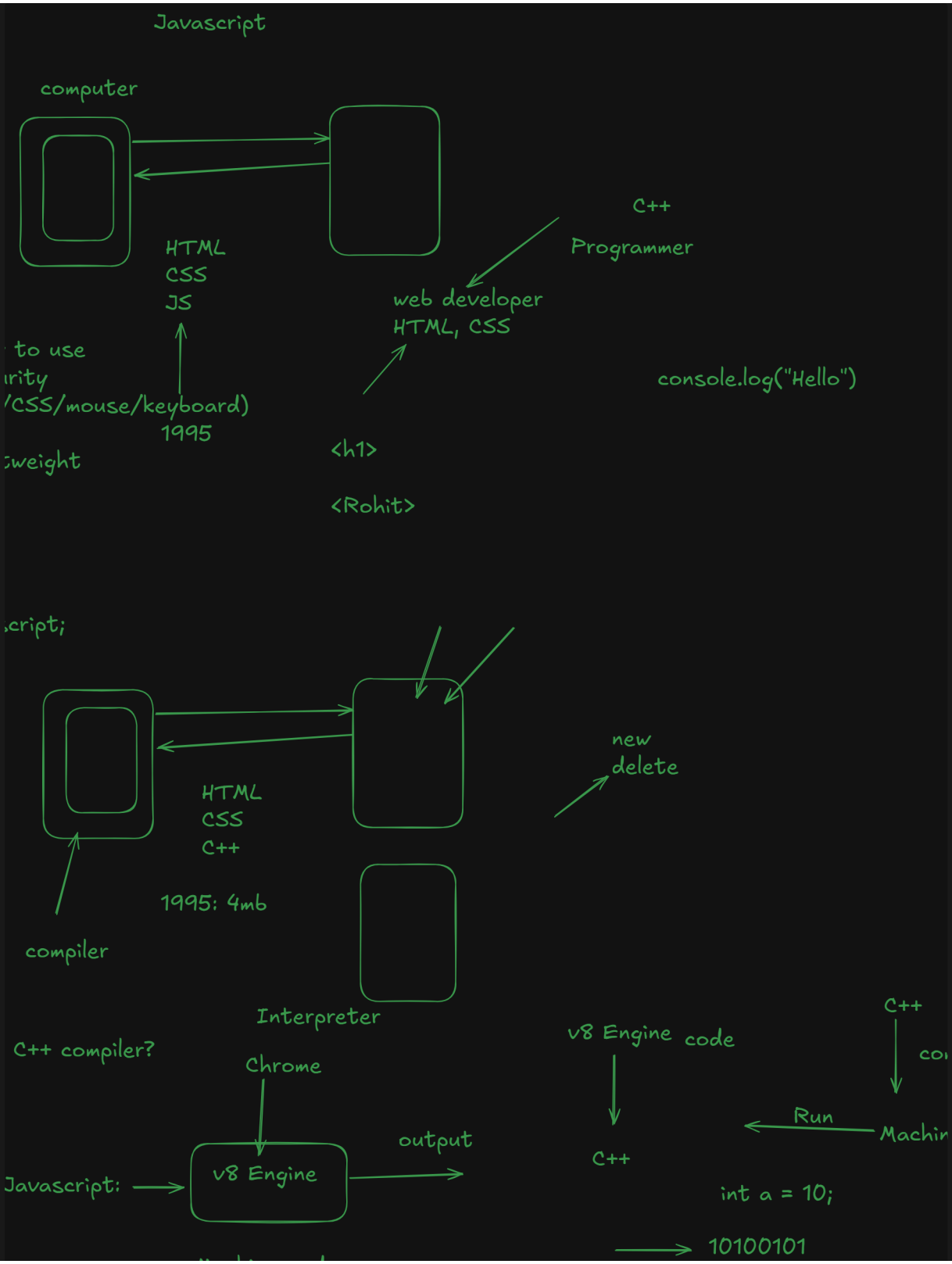
- **RAM:**
  - Average consumer PCs had **4 MB to 8 MB** of RAM.
  - Higher-end machines (for developers/enthusiasts) sometimes had **16 MB**.
  - Anything beyond that was rare and expensive.
- **Hard Disk:**
  - Common sizes: **200 MB – 500 MB**.
  - Higher-end PCs: **1 GB** drives were just starting to appear.
  - Compare that to today's **1 TB SSDs** 😅.
- **CPU:**
  - Intel **Pentium 75–133 MHz** was mainstream.
  - 486 processors were still common in cheaper systems.

🔹 Why this mattered for C++ vs JS in browsers

- Running a **sandboxed C++ runtime** would've eaten up tons of RAM and CPU → impossible when you only had 8 MB of RAM total, shared with Windows 95 and the browser itself.
- Hard disks were small and slow → no space for large runtime environments or heavy libraries.
- Browsers had to stay **lightweight** or else people simply wouldn't use them.

4: Automatic Memory Management (Garbage Collection):

- Developers don't have to manually allocate and free memory. The JavaScript engine handles it, reducing complexity and preventing common bugs like memory leaks that plague manual memory management in C++.


Javascript
computer
HTML
CSS
JS
C++
Programmer
web developer
HTML, CSS
to use
rity
/CSS/mouse/keyboard)
1995
weight
console.log("Hello")
<h1>
<Rohit>
cript;
HTML
CSS
C++
1995: 4mb
compiler
new
delete
Interpreter
C++ compiler?
Chrome
Javascript:
v8 Engine
output
v8 Engine code
C++
C++
Run
Machir
int a = 10;
10100101