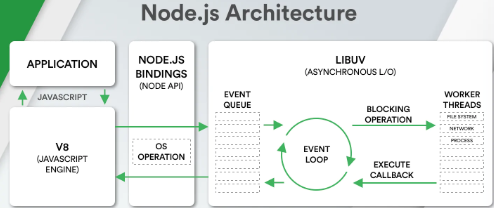
* **Node – JS Introduction: -**
* **Write an essay on the history and evolution of Node.js, discussing its architecture and key features.**
* **The History & Evolution of Node-JS: -**
* Before Node.js JavaScript was primarily used for client-side development, running with web browsers.
* Server-side programming was dominated by languages such as PHP, Python and Java.
* However, in 2009, Ryan Dahl created Node.js to address the limitation of traditional web servers.
* Dahl is introduced event-driven, non-blocking I/O as the core philosophy of Node.js leveraging Google Chrome’s V8 JavaScript Engine to execute JavaScript on the server side.
* This approach provided a lightweight, high performance solution for handling concurrent requests, making Node.js particularly suitable for real-time application.
* **Node.JS Architecture: -**



* Node.js is built on an event-driven, non-blocking I/O model that makes it efficient for handling multiple concurrent requests.
* **Application: -** The Application layer consist of JavaScript code written by the developer.
* It interacts with the debugging system using Node.js APIs to platform various operations. [e.g. file handling, networking and database interaction.]
* **V8 JS Engine: -** Node.js uses Google’s V8 engine to execute JavaScript code.0
* The V8 engine compiles JavaScript intro machine code ensuring high performance.
* **Node.js Bindings: -** Node.js provides **bindings** that allow JavaScript code to interact with **low-level system operations**.
* **Event Queue: -** Incoming requests from clients are placed into an **Event Queue**.
* The event queue is responsible for **managing multiple concurrent requests** efficiently.
* **Libuv: -** **Libuv** is a key part of Node.js that provides an **event loop** and handles asynchronous operations.
* It manages **I/O operations, file system interactions, networking, and worker threads.**
* **Event Loop: -** The **event loop** is the heart of Node.js, handling requests asynchronously.
* It continuously checks the **event queue** and processes callbacks.
* If a request is **non-blocking**, it executes immediately; if **blocking**, it is sent to **worker threads**.
* **Blocking Operations & Worker Threads: -** **CPU-intensive tasks** (like file system operations, networking, and processing) are sent to **worker threads.**
* Once completed, the **callback function** is executed in the event loop.
* **Callback Execution: -** When an asynchronous operation is completed, its associated **callback function** is added back to the **event queue**.
* The event loop processes it and sends the response back to the application.
* **Compare Node.js with traditional server-side technologies like PHP and Java.**

|  |  |  |
| --- | --- | --- |
| **Node.JS** | **PHP** | **Java** |
| |  | | --- | | JavaScript (JS) | | PHP | |  | | --- | | Java | |
| Non-blocking, event-driven | Blocking | Multi-threaded |
| Very fast (V8 engine, async I/O) | |  | | --- | | Slower (interpreted, blocking I/O) |  |  | | --- | |  | | Fast (JVM optimizations, multi-threading) |
| High (handles thousands of requests with a single thread) | Moderate (relies on multi-threading) | High (multi-threaded, used in enterprise apps) |
| Real-time apps, APIs, microservices | Web apps, CMS (WordPress, Laravel) | Large-scale enterprise apps, banking systems |
| Easy for JavaScript developers | Very easy | Moderate (strong typing, OOP concepts) |