**Blocking & Non-Blocking**

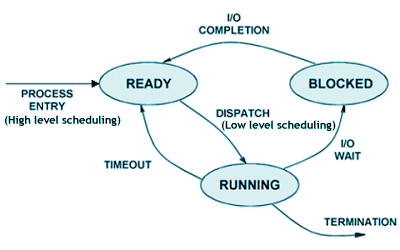
**Accessing external resources**

Blocking or non-blocking operating model is relevant when dealing with input/output operations. Writing to a resource or reading from a resource are considered I/O operations. The targets for such operations can be anything from files to network sockets to screen to keyboard. Any resource that is not accessible instantly at the machine instruction level but involves waiting instead requires I/O communication.

Computer memory, on the other hand, can be addressed instantly and does not require waiting. Accessing memory is therefore not considered an I/O operation. In fact, the random-access memory (RAM) name originates from its ability to access any location in the same amount of time irrespective of its physical location.

When an application is running, it spawns a process. Then, when it has to execute its JavaScript, it makes a request, or a call, to the server it is hosted on, be it locally on your computer or externally on a provider. Then, the server has to process that request and return the necessary output for the executed script. That is how many programs work: a request is made, then an output is returned. However, where blocking and non-blocking come into play is what happens when the application is waiting for the output by the server. It is relatively simple.

* **Blocking -** This type of call occurs when the Node.js process will not execute any additional JavaScript until the non-JavaScript portion of the call receives the server’s output. Only then will the JavaScript execute, and the application will proceed. This type of execution is synchronously, meaning line by line. Computer systems often read code from top to bottom, one line at a time, without skipping lines unless specifically instructed. While such a method of execution may have its advantages, it is not optimal, and that is why the majority of Node.js’ scripts and modules are non-blocking



* **Non-blocking -** This type of call is the exact opposite of the above one and is what Node.js uses most of the time. These calls will not halt the JavaScript from executing until the non-script processes finish. Instead, the program will execute all its operations asynchronously, meaning the line-by-line method described above will not apply. Often, the application will call for the function to simply keep going after starting an operation without waiting for the output. As you can imagine, this is a much smoother way of handling the process of interacting with the hosting server.

**Conclusion**

Blocking call waits for the I/O operation to complete before returning. Its results are returned synchronously. Nothing else in that process takes place during the waiting. In contrast, non-blocking call returns immediately without results and uses alternate means to check for completion. Other processing can be done while waiting and the results are returned asynchronously. Node.js libraries and core api provide non-blocking calls that can be used to build performant applications. Such applications make use of I/O waiting time to serve other requests.