Highly-scalable, data-intensive and real-time apps

ûdom

Non-blocking

ASYNCHRONOUS

Node is non - blocking asynchronous in nature.



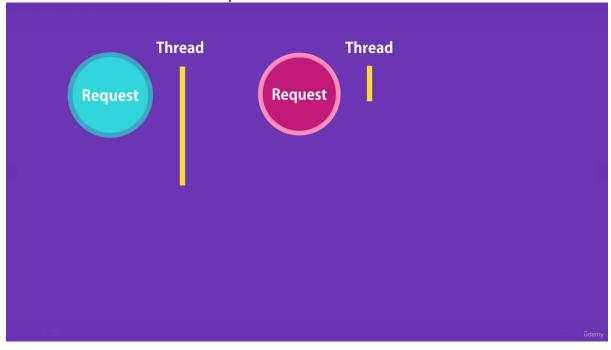
Imagine you go to a restaurant, a server comes to your table, takes the order and gives it in the kitchen. Then the server moves on to another table to serve while the order is getting ready in the kitchen. So, the same server can serve many different tables. Server does not wait for the chef to cook one meal before he serves the another table.

This is called non-blocking asynchronous architecture.

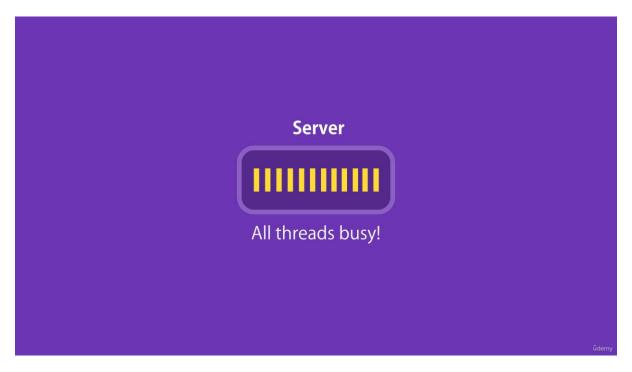
Contrary to this we have blocking synchronous nature where server waits for the cook while the order is getting ready and then serves the order. After that, server serves the another table.



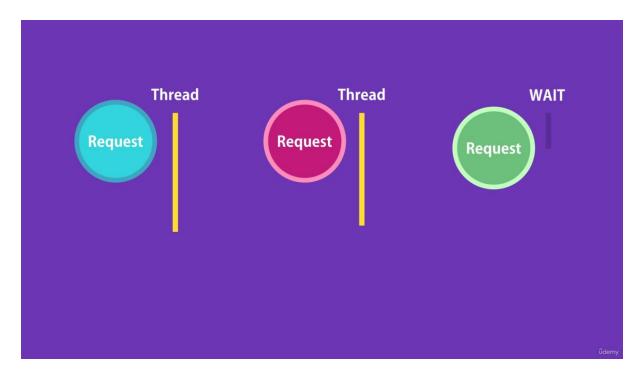
In the node environment, server is the thread which is allocated to handle the request. When we receive a request on the server a thread is allocated to handle that request.



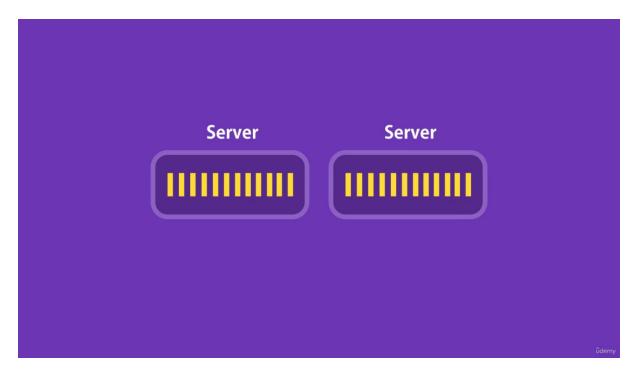
In synchronous nature a thread is allocated each time server receives the request.



Imagine we have large number of concurrent clients. At some point, we will run out of threads to serve these clients.



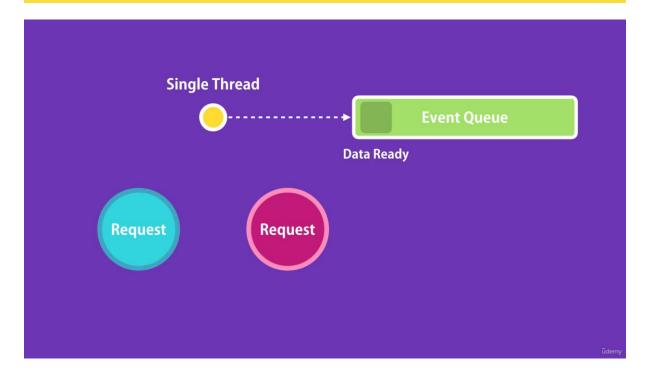
New clients have to wait until free threads are available.



If we don't want these clients to wait, we need more hardware.

So, with synchronous behaviour, we are not utilizing the resources efficiently.

Node applications are **asynchronous** by default



In node we have single thread to handle all requests. When a request arrives, that single thread is used to handle that request. If it needs to query a database, our thread does not need to wait for the database to return the data. While the database is executing the query, that thread will be used to serve another request. When the database is ready with the results, it puts the message in event queue. Node is continuously monitoring this queue in the background. When node finds an event in the event queue, it takes it out and process it.

Node is ideal for I/O-intensive apps

ûdem