**ASYNC RUST ARTICLE SUMMARY**

**INTRODUCTION:**

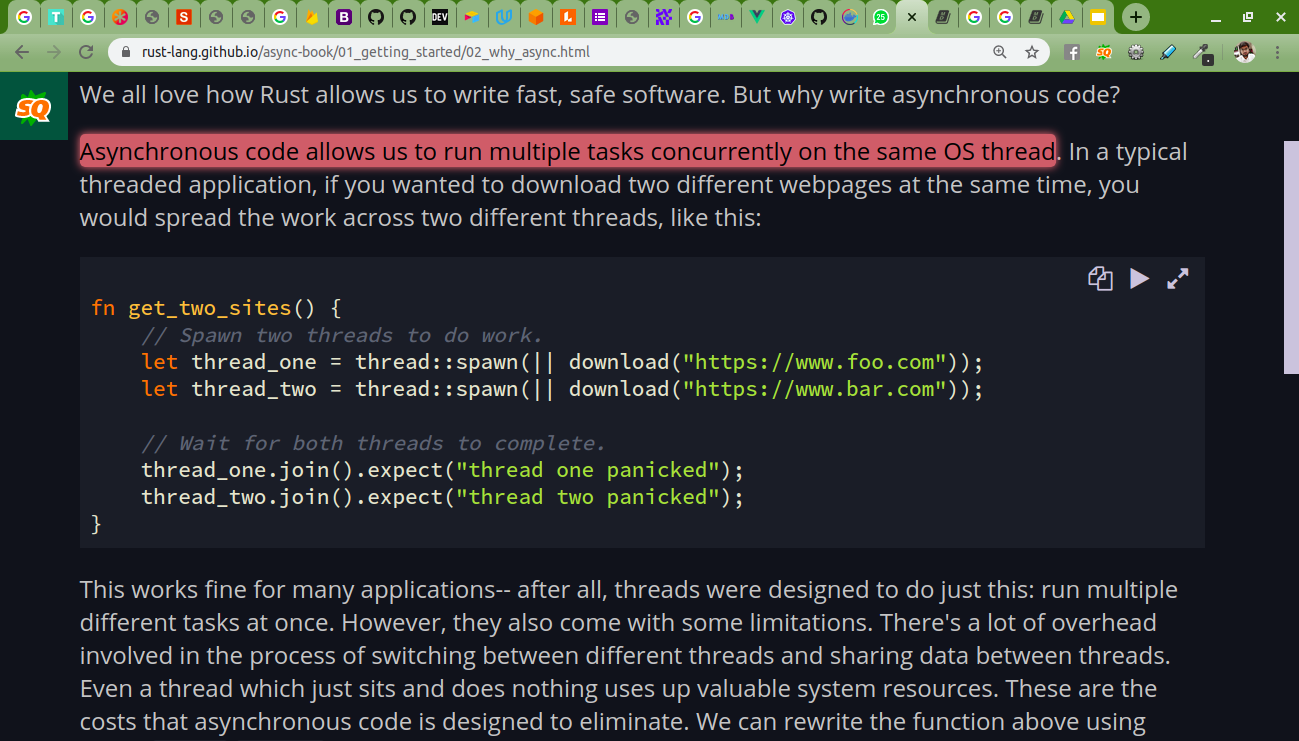
Async Programming allows you to run multiple operations simultaneously on a single application. It allows you to run multiple task concurrently on a single thread.

In async programming we use the concept of multitasking or concurrent programming. Concurrent programming or multitasking is better suited for when the task spends a lot of time waiting, such as for a response from a server. These tasks are called IO-bound or I/O intensive means when there is too much I/O processing in our system. So, multiple tasks are running at the same time because whenever they wait for a response, they remain idle, so we allow the computer to keep running the multiple tasks at the same time without waiting.

When we reach a point where we need the result of an asynchronous computation, we must (.await) it. In Rust, values that are ‘awaitable’ are known as ‘futures’.

**TYPICAL THREADED APPLICATION VS CONCURRENT PROGRAMMING**

**TYPICAL THREADED APPLICATION:**



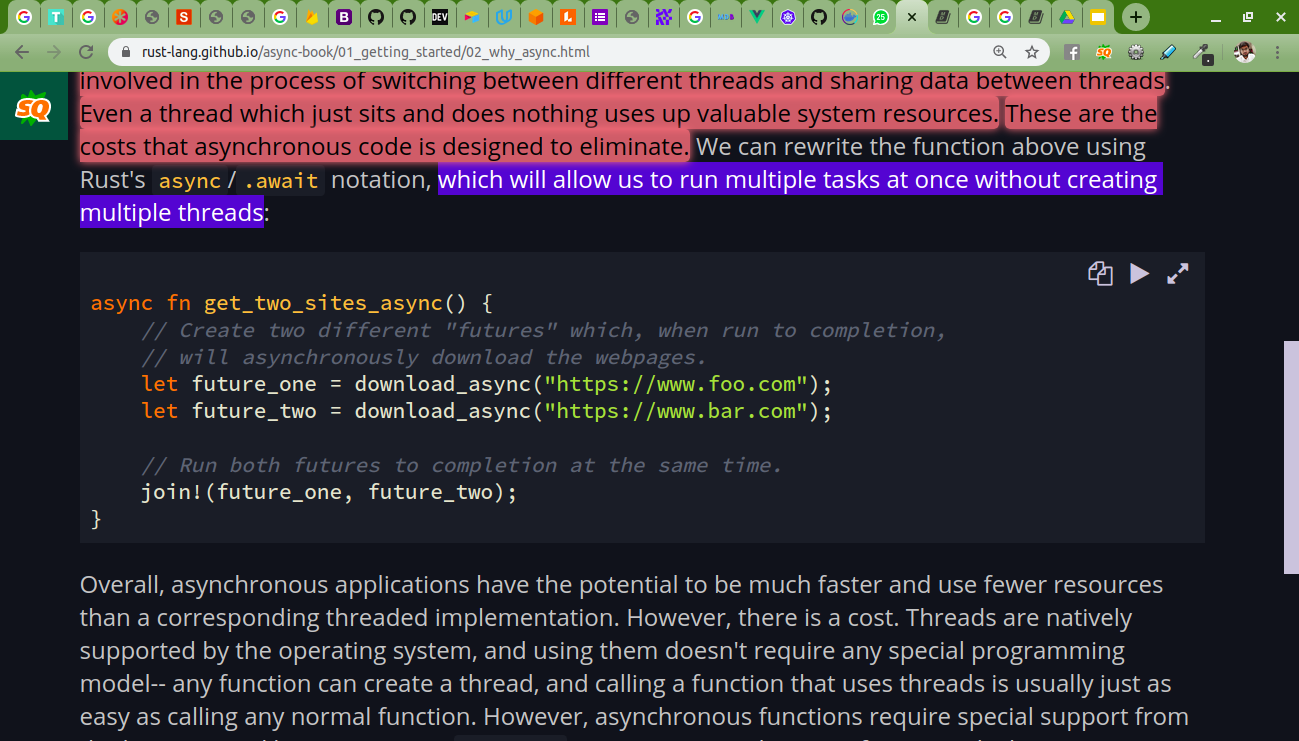
Here we can see in typical threaded application if we want to run two tasks at the same time then we need to use two different threads these types of code create many problems, or it makes the code complex. In above example join() is used. Whenever we use .join() then it means there are some conditions and they are interlinked with each other and all conditions are must be fulfilled.

**WHY ASYNC RUST**

To Avoid the limitations of multi-threaded application we use **async/await.**

Async/await allows you to run multiple tasks concurrently on a single thread.

**For Example:**



Async rust has fast performance and fewer resources as compare to multi-threading however it has a cost. asynchronous functions require special support from the language or libraries.

In Rust, async fn creates an asynchronous function which returns a Future. To execute the body of the function, the returned Future must be run to completion.