# What is Web Api



Web API is framework that help us **to build RESTFUL service or HTTP based service**.

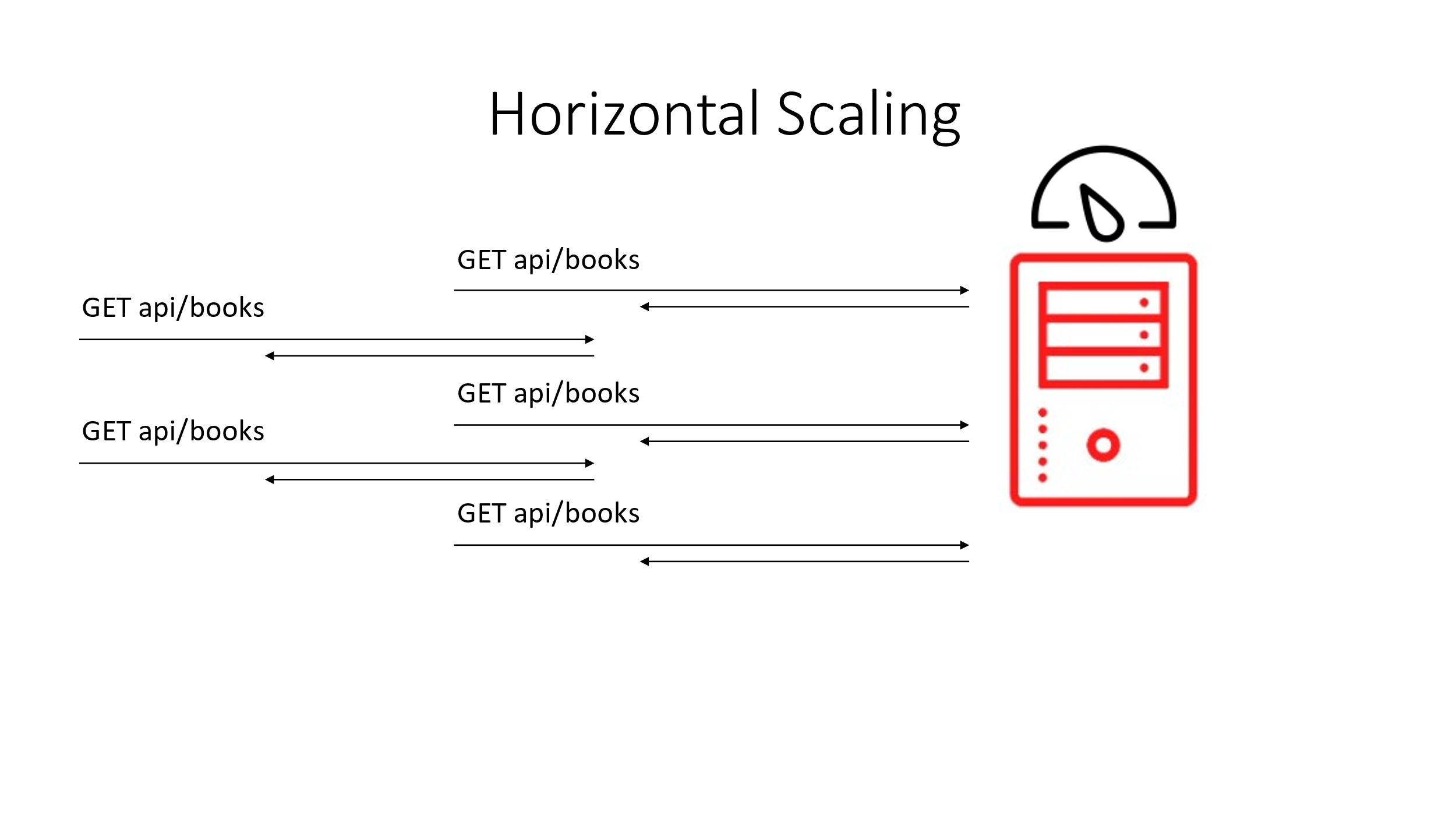
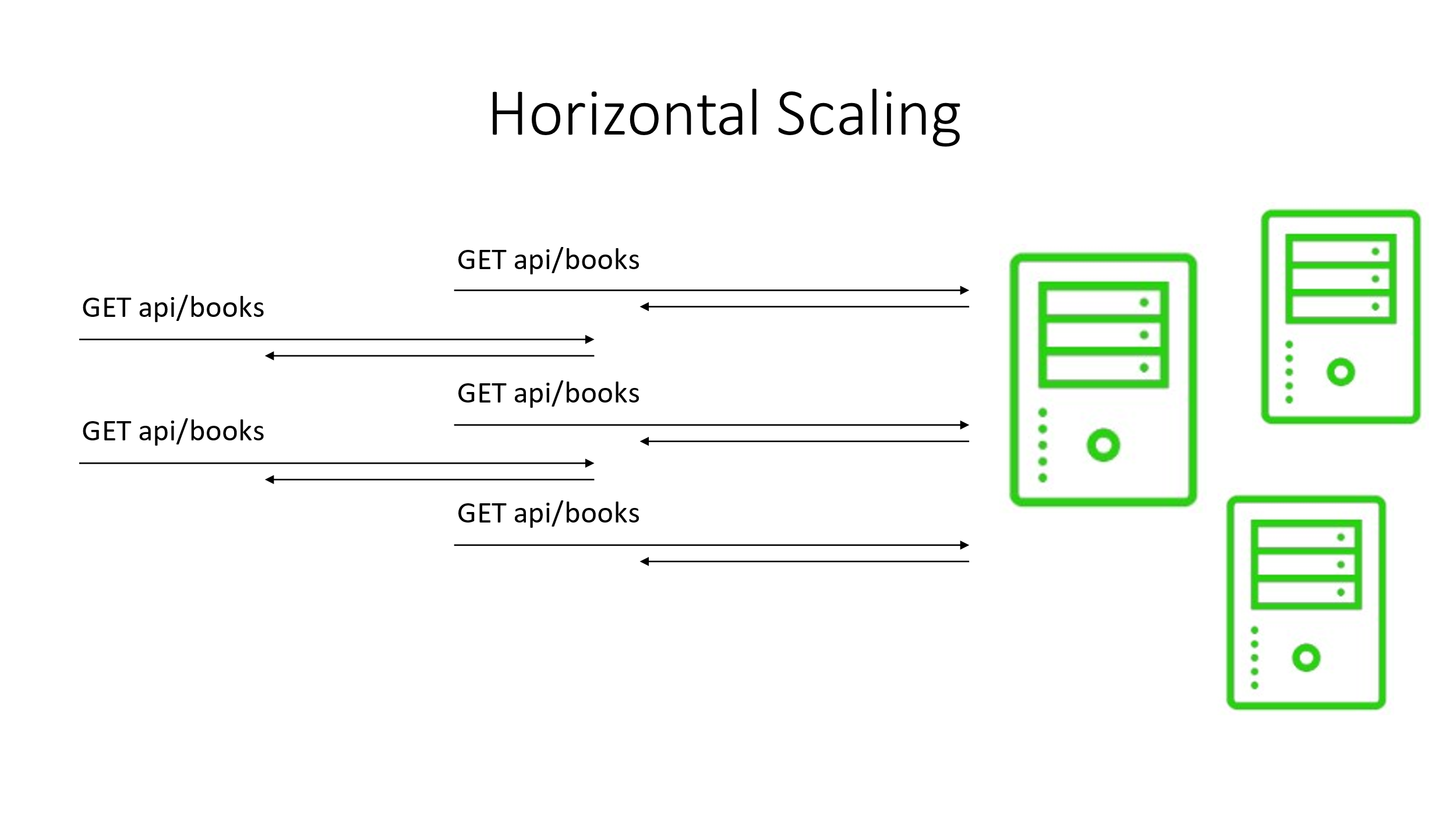
Web API can be consumed from **broad range of client**

1. Browsers
2. Mobile applications
3. Desktop applications
4. IOTs
5. <https://code-maze.com/asynchronous-programming-with-async-and-await-in-asp-net-core/>

# Async & await

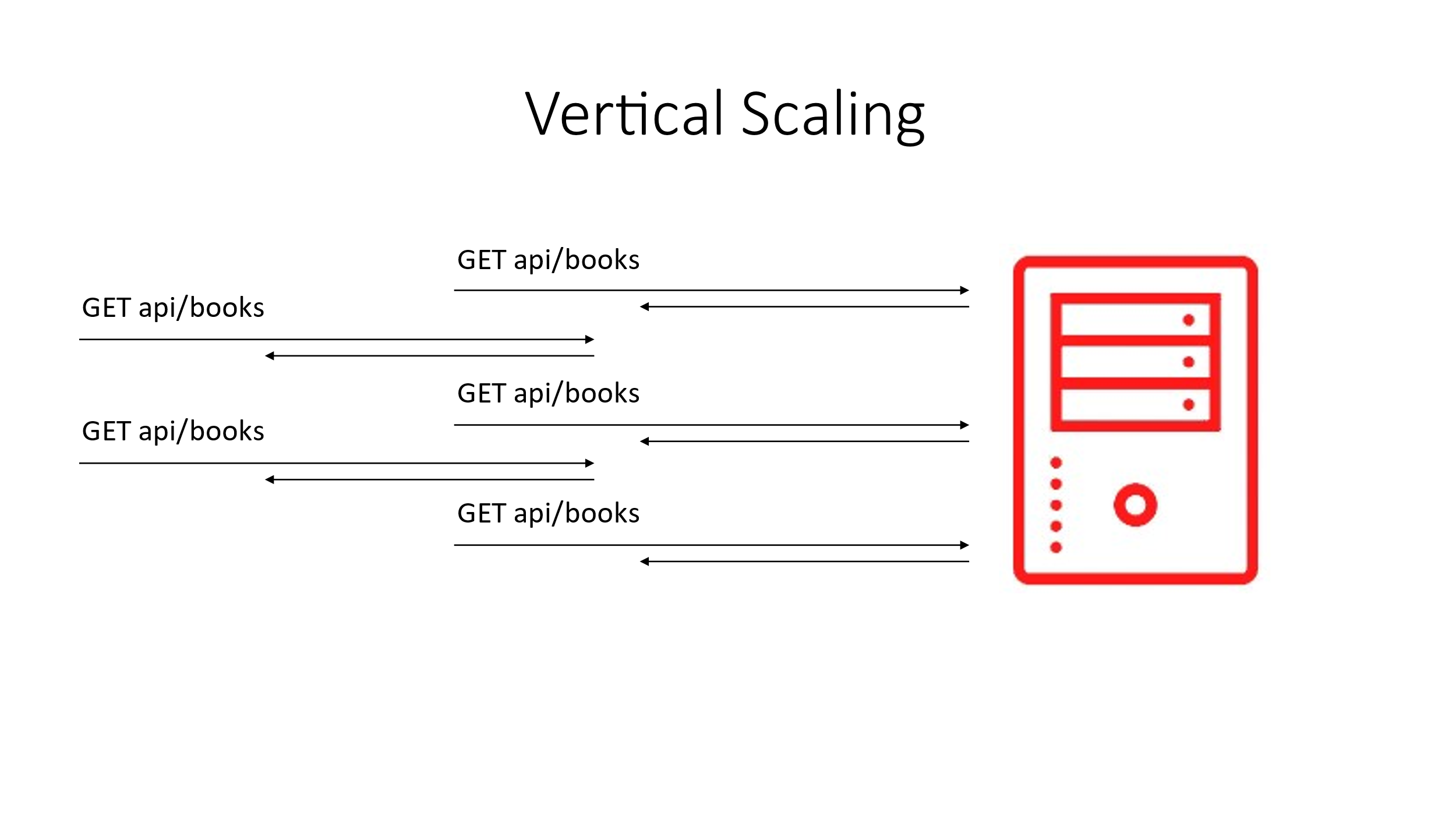
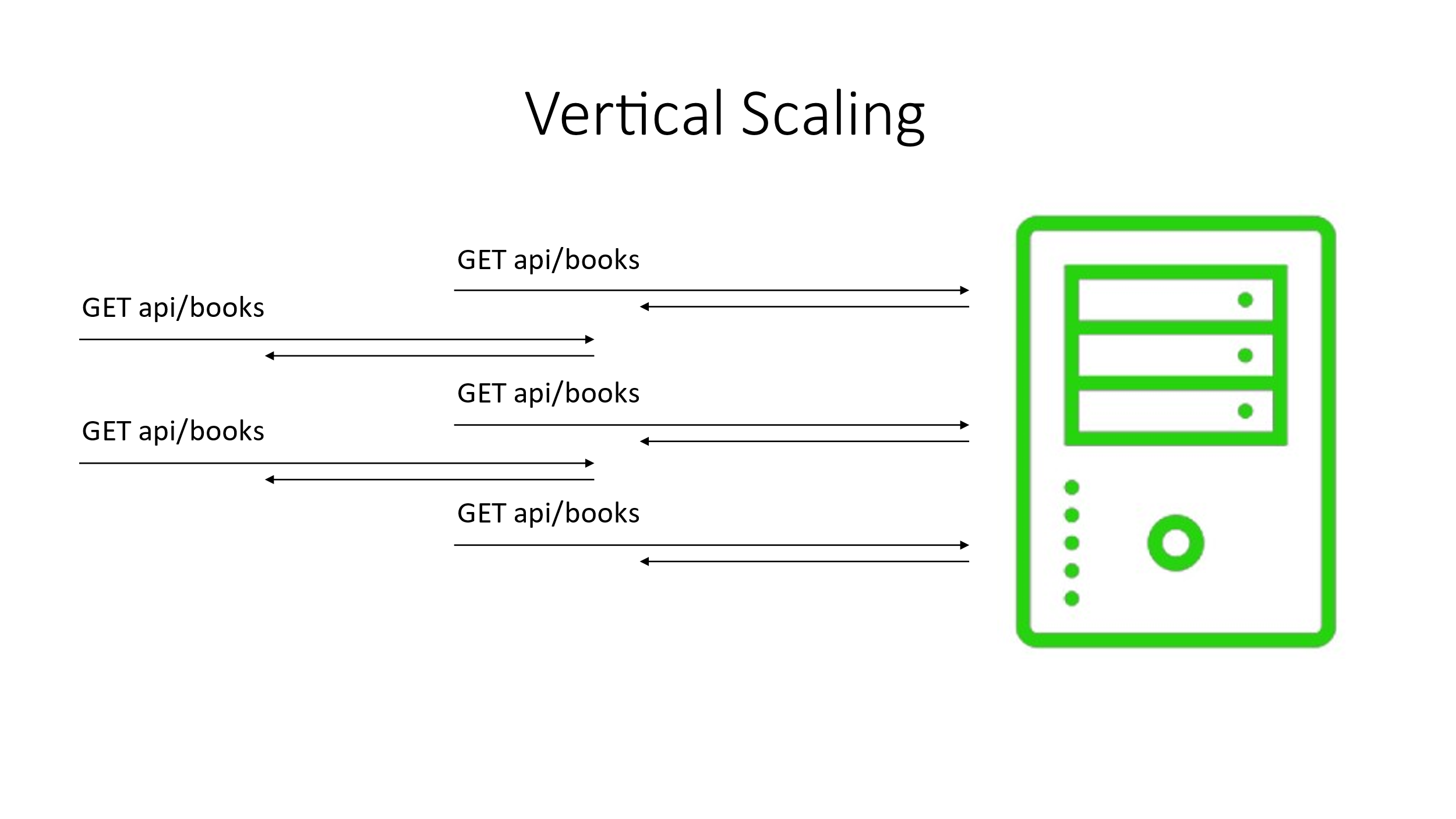


## Horizontal Scaling

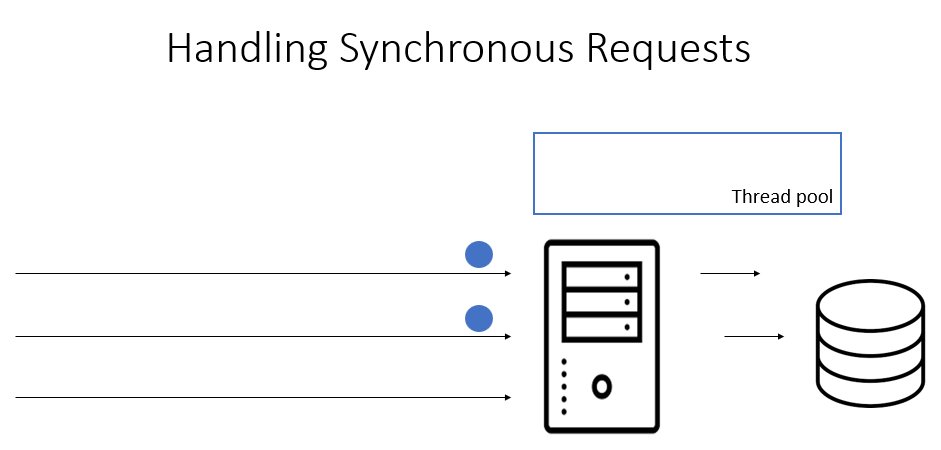
* One way of increasing scalability is by writing an API in a way that can accommodate horizontal scaling - RESTful systems are a good start
* Other components can still hurt scalability - non-distributed databases or caches

## Vertical Scaling

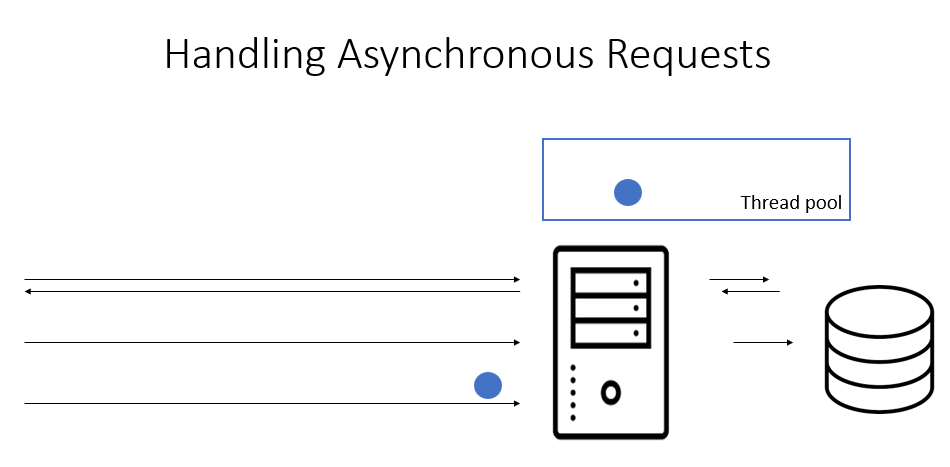
 

* Another way of increasing scalability is by writing an API in such a way that resource utilization is improved
* **Writing async code helps with improving the vertical scalability** at server level

## Handling Synchronous Requests



## Handling Asynchronous Requests



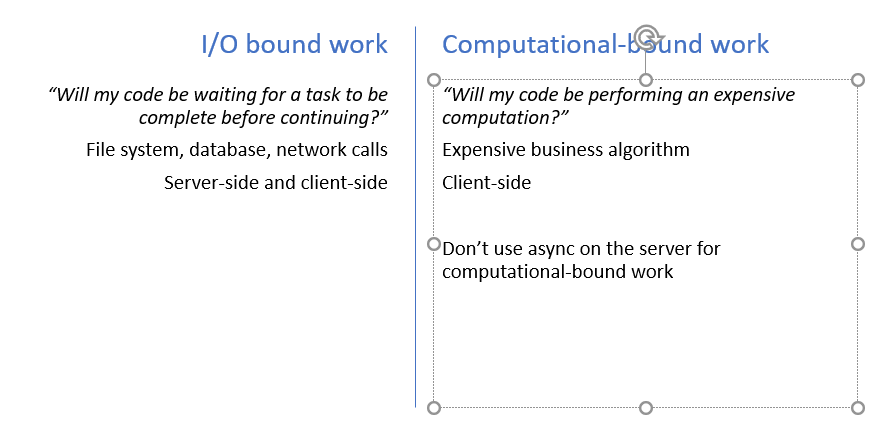
## The async/await Keywords

* Marking a method with the async Ensures that the await keyword can be used inside that method
* Transforms the method into a state machine (generated by the compiler)
* Using the await operator Tells the compiler that the async method can't continue until the awaited asynchronous process is complete
* Returns control to the caller of the async method (potentially right back up to the thread being freed)
* A method that is not marked with the async modifier cannot be awaited
* When an async method doesn't contain an await operator, the method simply executes as a synchronous method does

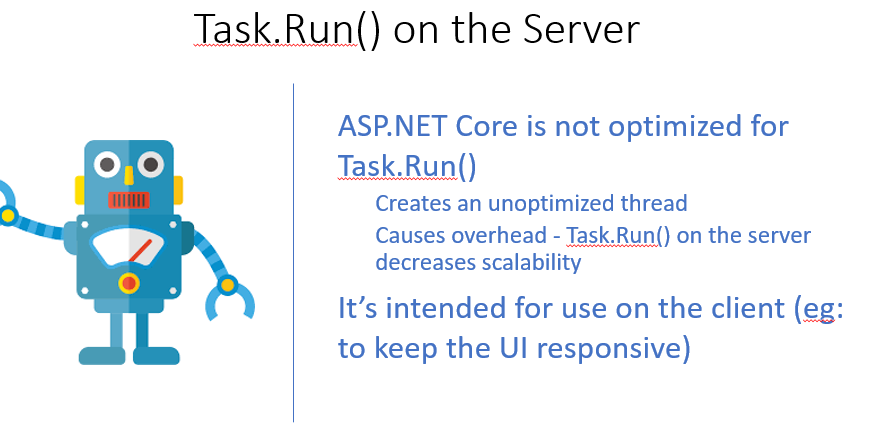
## Async Return Types

* Task<TResult>, for an async method that returns a value
* Task, to use it for an async method that does not return a value
* void, which we can use for an event handler
* Task and Task<T>
  + Represents a single operation that returns nothing (Task) or a value of type T (Task<T>) and usually executes asynchronously.
  + Status, IsCanceled, IsCompleted, and IsFaulted properties allow determining the state of a Task
  + Gets status complete when the method completes (and optionally returns the method value as the Task’s result)
* Void
  + Only advised for event handlers
  + Hard to handle exceptions
  + No easy way to notify the calling code of their status

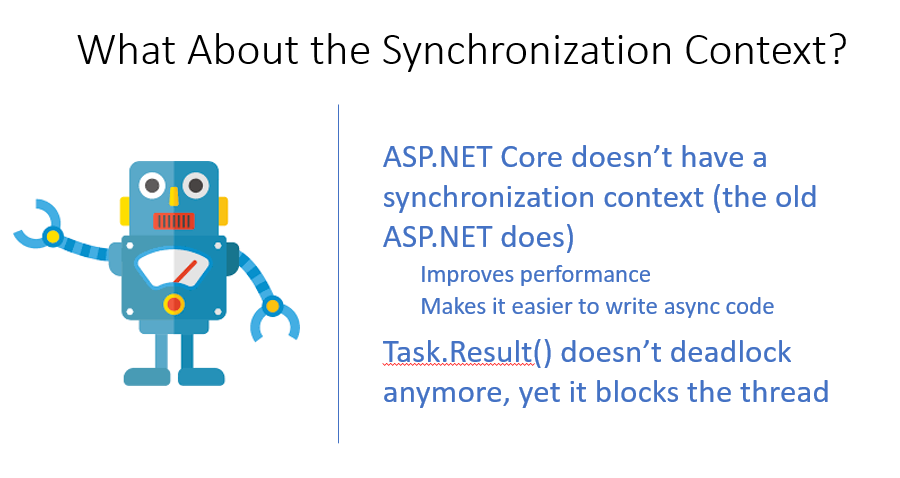
## I/O Bound Versus Computational-bound Work



## Task.Run() on the Server

* 

## Synchronization Context?

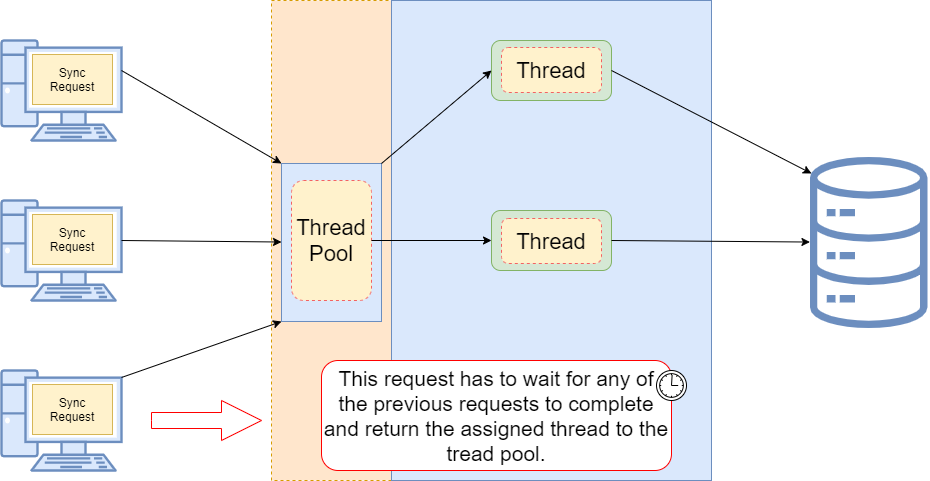


## What’s Next?

* **Testing for scalability Websurge:** [**https://websurge.west-wind.com/**](https://websurge.west-wind.com/)
* **API to API communication -** HttpClient, Cancellation, Handling faults,

## 2.11 Async & await details for interview

* [https://code-maze.com/asynchronous-programming-with-async-and-await-in-asp-net-core/#](https://code-maze.com/asynchronous-programming-with-async-and-await-in-asp-net-core/)
* <https://docs.microsoft.com/en-us/archive/msdn-magazine/2014/october/async-programming-introduction-to-async-await-on-asp-net>
* Async & await help to write asynchronous programming
* implementing asynchronous code help to improve performance and responsiveness of our apps
* Async & Await use thread pool effectively – When request come a thread from thread pool handle request – explain further in youtube and blog
* Marking a method with the async Ensures that the await keyword can be used inside that method
* Using the await operator Tells the compiler that the async method can't continue until the awaited asynchronous process is complete
* Async & await use thread pool effectively - <https://www.youtube.com/watch?v=_T3kvAxAPpQ>
* Don’t use async on the server for computational-bound work
* Use Task.run at client side - like Xamrian for keep UI responsive but do not use at server side
* Async Leakage - Repository does not have aysn await but controller api has – make sure that every layer has async keyword
* **Task.Result() blocks the** thread so do not use in async await code
* primary benefit of asynchronous code on the server side is scalability. we increase the scalability of our application by using the async and await keywords.
* allows us to execute our flows **without blocking our application or causing the thread pool starvation.**
* By its proper usage of async and await keywords, we can increase the vertical scalability at the server level of our API.
* When a client sends a request to our API to fetch the list of companies from the database, the ASP.NET Core assigns the thread from a thread pool to handle that request.
* The important difference is that the request thread has been returned to the thread pool while the asynchronous call is in progress
* Note that with synchronous handlers, the same thread is used for the lifetime of the request; with asynchronous handlers, in contrast, different threads may be assigned to the same request (at different times).
* Sync Request Issue
* As a result of a request waiting for an available thread, our client experiences a slow down for sure. Additionally, if the client has to wait too long, they will receive an error response, usually, the service is unavailable (503).
* But this is not the only problem. Since the client expects the list of companies from the database, we know that it is an I/O operation. So, if we have a lot of companies in the database and it takes three seconds for the database to return a result to the API, our thread is doing nothing except waiting for the task to complete. So basically, we are blocking that thread and making it three seconds unavailable for any additional requests that arrive at our API.



## How Async method can be used synchronsly

* If the async method does not contain an await operator, the method executes synchronously.

## 2.13 Async Method in C#

* An async method runs synchronously until it reaches its first await operator, at which point the method is suspended while the awaited task is completed. In the meantime, the control returns to the caller of the method.
* If the async method does not contain an await operator, the method executes synchronously.
* In C#, a Task represents an asynchronous operation
  + It is analogous to a promise or observables, used in other languages.
* Async method contains at least one await keyword
* Consequently, an async method with no await keyword is executed synchronously. A warning is emitted in this situation.
* In the async method MethodAAsync(), once the keyword await is meet for the first time the remaining of the task is actually executed by some random threads obtained from the runtime thread pool.- Asked by interviewer
* <https://blog.ndepend.com/c-async-await-explained/> - programe

# WEB API Best Practice

* 
* <https://www.youtube.com/watch?v=_T3kvAxAPpQ>