-

-

-

Asynchronous programming is a form of parallel programming that allows a unit of work to run separately from the primary application thread.

When the work is complete, it notifies the main thread (as well as whether the work was completed or failed). There are numerous benefits to using it, such as improved application performance and enhanced responsiveness.

-

-

-

-

-

-

But, like all things in programming, it’s not something that you should use in every instance; in fact, there are some situations in which you should avoid it. Asynchronous programming has actually been around for a long time, but in recent years, it’s become more widely used. According to Mike James at iProgrammer, “Often the programmer is fully aware that what they are doing is object oriented but only vaguely aware that they are writing asynchronous code.” So, we set out to identify the best use cases for async as well as situations in which you shouldn’t use it. We searched the web for insights and reached out to a panel of programmers and asked them to answer this question:

-

-

-

Asynchronous programming allows you to offload work. That way you can perform that work without blocking the main process/thread (for instance navigation and utilization of the app). It’s often related to parallelization, the art of performing independent tasks in parallel, that is achieved by using asynchronous programming.

With parallelization, you can **break what is normally processed sequentially**, meaning break it into smaller pieces that can run independently and simultaneously. Parallelization is not just related to processes and capabilities but also with the way systems and software are designed. The biggest advantage of applying parallelization principles is that you can achieve the outcomes much faster and it makes your system easier to evolve and more resilient to failure.

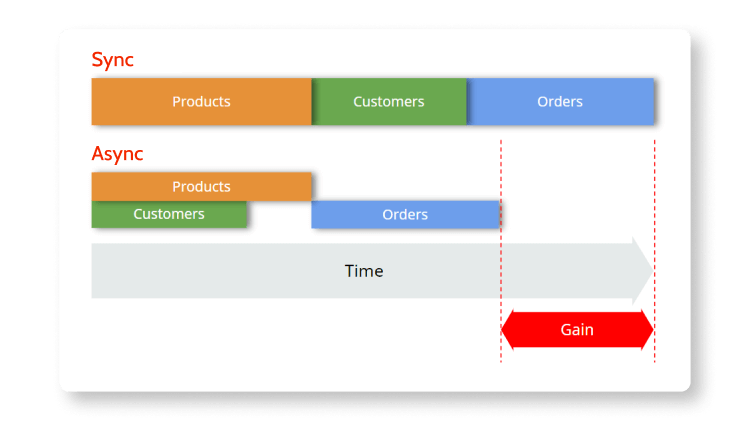
Pretty cool, right? However, not all processes should follow parallelization principles and execute asynchronously. In this blog post, I’ll explain when you should apply asynchronous programming and when sticking to synchronous execution is the best option.

## Asynchronous vs. Synchronous Programming

Before we jump into the juicy stuff, let’s start by clarifying the difference between asynchronous and synchronous programming.

In **synchronous** operations tasks are performed**one at a time**and only when one is completed, the following is unblocked. In other words, you need to wait for a task to finish to move to the next one.

In **asynchronous** operations, on the other hand, you can move to another task before the previous one finishes. This way, with asynchronous programming you’re able to **deal with multiple requests simultaneously**, thus completing more tasks in a much shorter period of time.



## When to Use Asynchronous Programming

As I said in the beginning, asynchronous execution is not the best scenario for all use cases. You should only use it if you’re dealing with **independent tasks**. So, when you’re designing a system, the first step you need to take is to identify the dependencies between processes and define which you can execute independently and which needs to be executed as a consequence of other processes.

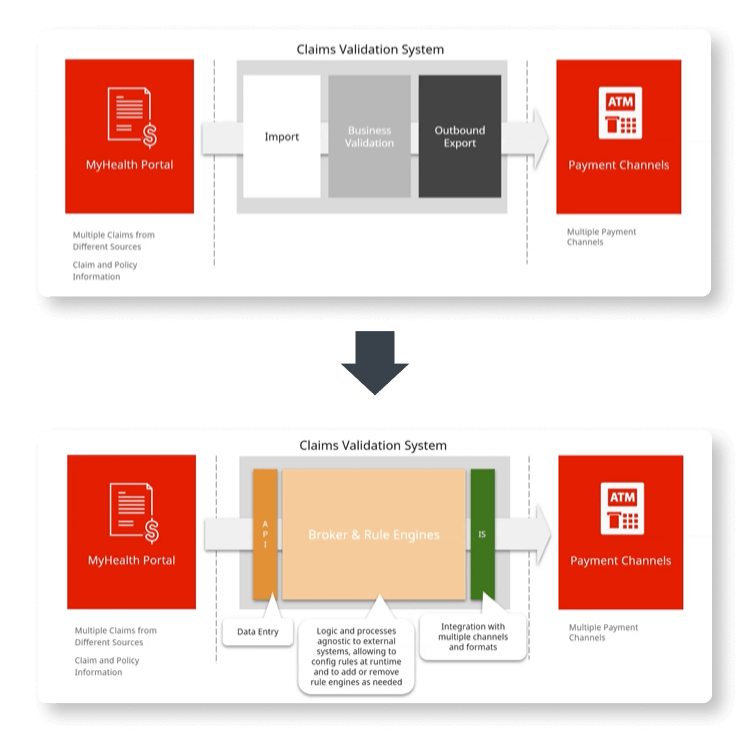
Take a look at the image above. On the top, you can see that in synchronous execution, the tasks are executed in a sequential way; the products are the first to be executed, then customers, and finally orders.

Now imagine that you did an analysis and concluded that customers are independent of products, and vice-versa, but that to execute orders you need the information from products first—there’s a dependency. In that case, the first two tasks can be executed asynchronously, but orders will only be executed when products are completed—so, synchronously.

As a result, by applying parallel computing and asynchronous programming when dealing with independent tasks, you’re able to perform these tasks way faster than with synchronous execution because they’re executed at the same time. This way, your system releases valuable resources earlier and it’s ready to execute other processes that queued.

## How to Design a System that Runs Asynchronously

To better illustrate how to design a system that follows asynchronous programming principles, let’s try to do it using a typical claims processing portal. Something like this:



In this challenge, we have the portal that policyholders or other entities use to insert and manage claim information. This portal communicates with a claims validation system through an API. The validation system imports data to a business validation engine that includes a broker and business rules that are processes and logic agnostic to external systems. Finally, this system integrates with payment channels to which it exports the outcomes of the business validation.

Now, I work with OutSystems and so I’m going to show you how to implement a system that can run asynchronous processing using the OutSystems platform. But if you’re not an OutSystems developer or architect, don’t run away just yet; you will  see how I was able to automate parallel asynchronous processes in this scenario and adapt it to your preferred technology.

Also, recently together with my colleague Davide Duarte, we did a demo showing all the steps we follow to automate these processes, so if you want to check it in more detail, take a look at our session about[how to use asynchronous techniques](https://www.outsystems.com/events/tech-talks/in-depth/asynchronous-programming/).

Before we get our hands dirty—I promise, we’re almost there—because I’m using OutSystems, I need to clarify some of the terms and capabilities I’ll be using. If you’re already familiarized with them, or you simply want to see our proposed architecture to solve this challenge, you can [skip the next section.](https://www.outsystems.com/blog/posts/asynchronous-vs-synchronous-programming/#section)

### OutSystems Asynchronous Capabilities

OutSystems is a modern, AI-powered application platform that offers several in-built asynchronous capabilities:

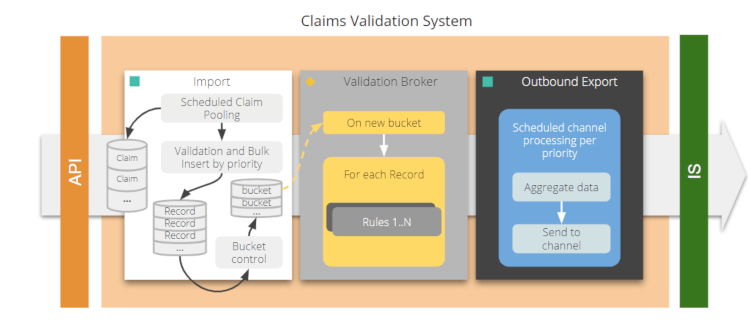
* **Timers:** Timers are meant to perform routine tasks like sending emails that are scheduled on a daily basis like a digest but also long and heavy processing tasks. If you want to prep data to use after publishing an app, you can also use timers to perform bootstrap logic and they are also recommended to process large chunks of information that take some time to execute (20-30min).
* **Business Process Technology (BPT):**This capability is  available to define and execute business processes over apps built with OutSystems. BPT performs system-to-systems and human-to-systems interactions. It  also supports business workflows and is able to perform multiple activities per process. Those activities can be automatic or human activities.
* **Light Processes:**Light processes are event-driven processes. You can think of them as triggers. Light processes are meant to handle several thousand events per day, like as event brokers that require scalable database queuing. Compared with BPT, they only support one activity inside each process and it must be an automatic activity.

In addition to that, there are a few other capabilities important to note:

* **Retry on error:** If an error happens, timers and processes will retry up to three times.
* **Timeout protection:** Timeout protection prevents a background process to run infinitely, allowing resources usage control.
* **Queue management:**Queues are managed automatically by the platform, removing the need to worry about queuing background processes.
* **Scalability vs parallelization:**Processes that are queued in the background are distributed by the front-end automatically, so you don’t need to worry about that.
* **Isolation:**Background processes can be isolated per front-end for both cloud and on-premises infrastructures or per zone if you’re dealing with infrastructure on-premise only. This provides you an additional layer of security.

### Automating Parallel Asynchronous Processes of a Claims Processing Portal: Suggested Architecture

Let’s look at the claims validation system in greater detail:



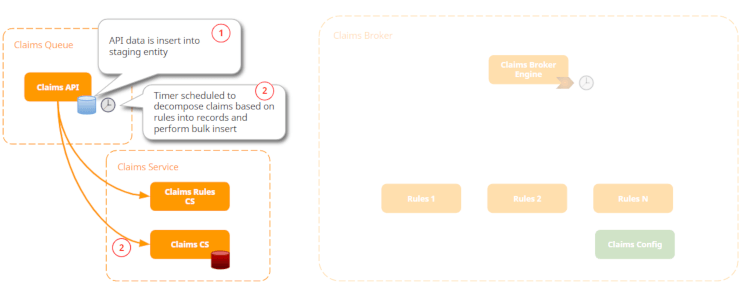
The API is going to insert each of the claims into a staging data. Then a timer, that is scheduled, is going to look into those claims. Note that each claim is a structure that can have multiple records inside. So, the timer is going to validate and decompose those claims and records and perform a Bulk Insert into the business entities for the claims.

Now, we want to be able to optimize our resources as much as we can. For that reason, we’re not going to process each one of the records as soon as they’re inserted into the table. We also want to remove the overhead of the starting of a validation process, so we’re going to use a bucket control which is basically a record where you’re going to specify which is the initial record and which is the end record. So, this bucket is basically an interval of records and claims that are going to be processed.

With that, for each bucket record, we’re going to trigger a light process that will process each of the records inside the bucket. By processing I mean we’re basically applying the rules. These rules are engines that can be plugged and played, so they can be added at runtime into the system.

Once a record is processed, if it’s considered valid according to the business rules, it will be set as such and a timer will aggregate the data and send it to the channel accordingly.

That said, here’s our proposed architecture:

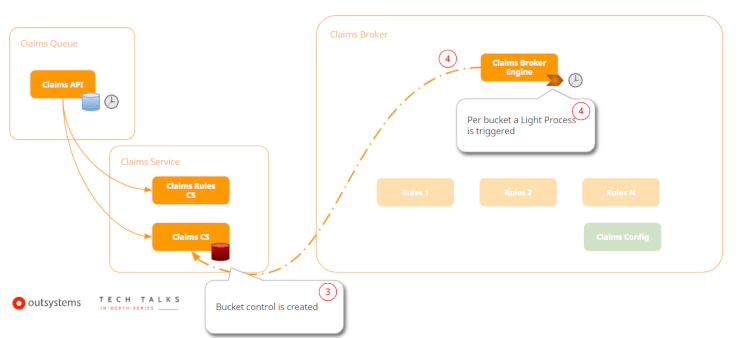


First, the Claims API will collect all the data that is imported through a JSON structure or XML. Each data is then inserted into the staging entity. The goal here is to speed up the data insertion and prevent  losing data.

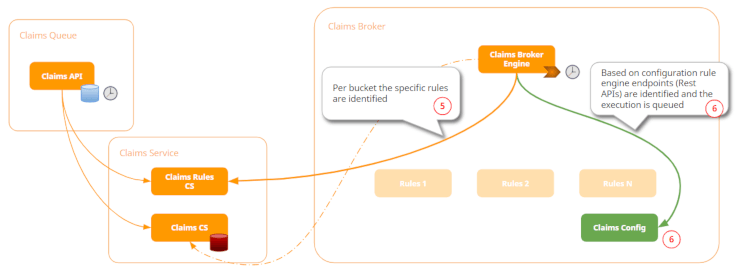
Once the data is inserted, a timer is launched on a scheduled basis. The purpose of this timer is to decompose claims based on Claims Rules previously pre-defined and perform the Bulk Insert. Based on those rules, the claims will be inserted in the Claims Service.

The number of records inside each bulk is the number of records you can process within a three-minute limit. At this point, you should avoid having parallel capacities still available and only run one light process at a time because you are specifying a bucket with a large number of claims inside.

Once the bucket control is created, then a light process is triggered.

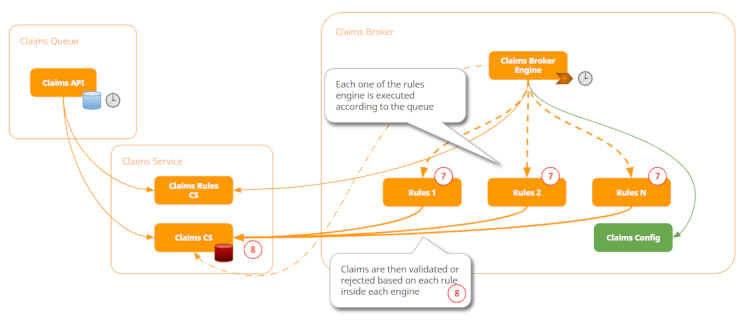


And the triggering is going to execute the process.



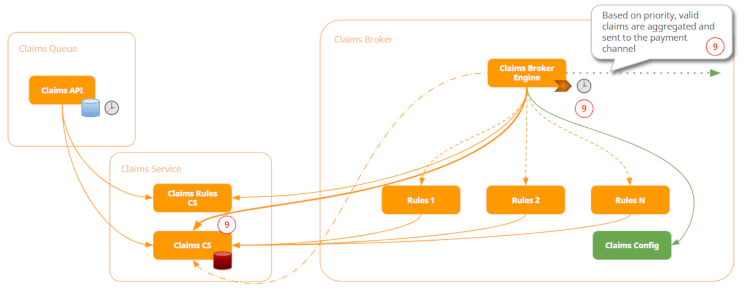
So, the Claims Broker Engine is going to get the rules that were predefined in the Claims Rules and, based on the configuration, the rule engine endpoints—which are basically REST APIs because we want to be able to plug and play rules engines in our system—are identified and the execution is queued.

Once we get all the rule engines that we need to execute that particular type, and have the endpoints for them, we’re now going to call them sequentially. So we’re going to apply those rules on top of claims data.



As a result of each rule, a claim will be considered “valid” and thus proceed to the next step, or “not valid” and, in that case, rejected by the system.

Finally, the system will have a timer that based on priority is going to get all the valid claims, aggregate them, and send them to payment channels.



You can also  make the prioritization rules more complex. For example, you may want to define that certain claims should be sent as soon as possible so they would need to be sent right after they’re validated by the system and are ready to be paid; or you may want to define certain claims as low-priority and, in that case, can be processed by the timer.

Another benefit of using this type of step-by-step rules in the engine is that the system can also recover from a timeout or even a crash. Imagine that after processing rule one but before executing rule two, there's a timeout or a catastrophic failure in the process and the system needs to recover from it. With this system, each of the claims is available to recover from the exact point where it was before the incident. So, in this case, the claim would recover and execute rule two instead of repeating rule one.

## Key Takeaways

Hope this article helps you clear up any questions you may have about when you should use asynchronous or synchronous programming. To wrap it up, here are the main key points:

* Use the asynchronous techniques that are more suitable for the outcome.
* Scale front-end servers and configurations to fit your needs. Keep in mind that when you go into millions of records, you need more front-end servers to accomplish your needs.
* Design with flexibility in mind and avoid hard coded values or site properties. Imagine you use hard coded values for bucket control; if your claims validation process becomes slower and for some reason you’re not aware, you start having timeouts. Now you’re in an even worse situation, because you need to publish the changes and not go into a backoffice to change it.
* Don’t over engineer. Try to keep your architecture and system as simple as possible.

If you want to see this scenario in action, take a look at my recent TechTalk, [How to Use Asynchronous Techniques in OutSystems](https://www.outsystems.com/events/tech-talks/in-depth/asynchronous-programming/). My colleague Davide and I will show you the solution here proposed while leveraging the OutSystems asynchronous capabilities, fostering scalability and resilience to failure and ready to handle large data volumes.

-

-

-

“What are some examples of situations where programmers use asynchronous programming but shouldn’t (or areas where they don’t but probably should)?”

Meet Our Panel of Expert Programmers:

|  |  |  |
| --- | --- | --- |
| * [James McCarthy](https://stackify.com/when-to-use-asynchronous-programming/#McCarthy) * [Calvin Brown](https://stackify.com/when-to-use-asynchronous-programming/#Brown) * [Kevin Ng](https://stackify.com/when-to-use-asynchronous-programming/#Ng) * [Lindsey Havens](https://stackify.com/when-to-use-asynchronous-programming/#Havens) * [Manu Singh](https://stackify.com/when-to-use-asynchronous-programming/#Singh) * [Steve Silberberg](https://stackify.com/when-to-use-asynchronous-programming/#Silberberg) * [Max Galka](https://stackify.com/when-to-use-asynchronous-programming/#Galka) | * [Peter Tasker](https://stackify.com/when-to-use-asynchronous-programming/#Tasker) * [Ashish Datta](https://stackify.com/when-to-use-asynchronous-programming/#Datta) * [Johannes Kadak](https://stackify.com/when-to-use-asynchronous-programming/#Kadak) * [Microsoft Developer Network](https://stackify.com/when-to-use-asynchronous-programming/#MSDN) * [Mark Patterson](https://stackify.com/when-to-use-asynchronous-programming/#Patterson) * [Brown University Computer Science Program](https://stackify.com/when-to-use-asynchronous-programming/#BrownU) | * [Maria Yakimova](https://stackify.com/when-to-use-asynchronous-programming/#Yakimova) * [Brij Bhushan Mishra](https://stackify.com/when-to-use-asynchronous-programming/#Mishra) * [Eric Vogel](https://stackify.com/when-to-use-asynchronous-programming/#Vogel) * [Peter Olson](https://stackify.com/when-to-use-asynchronous-programming/#Olson) * [Joy George Kunjikkuru](https://stackify.com/when-to-use-asynchronous-programming/#Kunjikkuru) * [Sathyaish Chakravarthy](https://stackify.com/when-to-use-asynchronous-programming/#Chakravarthy) * [Richard Rodgers](https://stackify.com/when-to-use-asynchronous-programming/#Rodgers) |

Learn more about the ideal use cases for asynchronous programming by reading what our experts had to say below.

James McCarthy

[James](https://boldtask.com/) is a serial web entrepreneur who builds websites with the aim of helping millions of people. In the last 2 years he has created MRI Directory, Create Brief, and Headphone Charts.

**“Every situation is different and there’s no real grab-bag solution…”**

As a rule of thumb, asynchronously updating records that are dependent, or depended upon, is generally a bad idea. It’s a quick way to have your data become out of sync. Simple processing of independent data is a great place for asynchronous programming and provides a better UX and, most of the time, better performance in your application.

Calvin Brown

[@iamcalvinbrown](https://twitter.com/iamcalvinbrown)

Calvin Brown is the Founder of [Kairu Consulting](http://www.kairuconsulting.com/).

**“Programmers often use async too much…”**

In mobile applications to prevent the freezing of the screen, and give the customer the impression that the process is taking actions visually, but it’s usually not. Literally speaking, when an application asks for data (from a database, webservice, etc.), it has to wait until that service replies. All of the fancy gestures that are usually present in applications happen because of async calls to the services at the same time as entertaining the user with some logo or messaging. The amount of resources used at this time is much higher than normal app usage and ultimately, with lots of refreshes, users’ memory suffers.

There are work-arounds, and async is necessary in instances such as preloading data when the app first begins. In other instances, though, it’s cliché and often to the detriment of the users’ device to utilize async so heavily.

Kevin Ng

[@wildebeest\_dev](https://twitter.com/wildebeest_dev)

Kevin Ng is the Tech Lead at [Wildebeest](https://wildebee.st/).

**“Asynchronous programming and parallel programming have great benefits…”**

But can be overly used at times. One case I feel programmers may overly use async is for simple and basic computations. There’s no real performance benefits to using async for these calculations.

In addition, another situation when async may not be useful is when you have a single database server not utilizing connection pooling. If all requests hit the same database using a long running connection, it won’t make a difference if the calls are asynchronous or synchronous. Your bottleneck will be the database server.

Lindsey Havens

[@PhishLabs](https://twitter.com/phishlabs)

Lindsey Havens is the Senior Marketing Manager for [PhishLabs](https://www.phishlabs.com/).

**“Basically you can use Asynchronous programming except when the following conditions are true…”**

* You are aiming for simplicity rather than efficiency.
* You are looking to run simple or short running operations.

Asynchronous programming will not provide benefit and actually will result in more overhead on operations that are primarily CPU operations instead of those that involve network or disk overhead.

Manu Singh

[@ClrMobile](https://twitter.com/ClrMobile)

Manu Singh is a Mobile Architect at [Clearbridge Mobile](https://clearbridgemobile.com/) with experience in Android Design, Development, and Testing. At Clearbridge, Manu manages project resources for a team of developers building world-class apps for enterprise clients.

**“If there is a resource that is used by different elements or a single thread in an application that is responsible for a resource or functionality…”**

It needs to be synchronized. For example, on Android, there is a main UI thread, which is responsible for displaying the UI for the app. Having too many asynchronous calls affecting the UI can cause issues and can slow things down. In this case of the UI main thread, you would want some synchronicity so that the UI is displayed properly, there are no conflicts with what is displayed, and the UI population is done in proper sequence.

Steve Silberberg

[@fitpacking](https://twitter.com/fitpacking)

Steve is founder of [Fitpacking](http://www.fitpacking.com/), a small business that takes people on backpacking adventure vacations to get fit and lose fat.

**“If you program Android apps using MIT App Inventor 2…”**

You may be forced into asynchronous programming even though you don’t want it. Example: I programmed an app that queries Google Fusion Tables on program load. However, App Inventor 2 continues executing other blocks even though results of the Fusion Table queries have not returned. Because of this, you had better not make these subsequent code blocks dependent upon query results or you will get errors. This is one case where a programmer uses asynchronous programming when they really shouldn’t.

Max Galka

[@galka\_max](https://twitter.com/galka_max)

Max Galka is an entrepreneur and web developer based in New York City. He is founder of the mapping platform [Blueshift](https://blueshift.io/) and is an adjunct lecturer at the University of Pennsylvania.

**“When transitioning from front-end JavaScript to back-end development with Node…”**

The first lesson you learn is to make your loops asynchronous to avoid blocking the event loop. But I often see people take this idea too far. Asynchronous loops are necessary when there is a large number of iterations involved or when the operations within the loop are complex. But for simple tasks like iterating through a small array, there is no reason to overcomplicate things by using a complex recursive function. A simple synchronous for/while loop works just fine, and will also be faster and more readable.

Peter Tasker

[@petetasker](https://twitter.com/petetasker/)

Peter is a PHP and JavaScript developer from Ottawa, Ontario, Canada. He currently works on the WP Migrate DB Pro team at [Delicious Brains](https://deliciousbrains.com/). In a previous life he worked for marketing and public relations agencies. He loves WordPress and dislikes FTP.

**“Asynchronous code is often used when…”**

Writing JavaScript, especially with AJAX handlers or callbacks. It can cause problems when used incorrectly, however. We work with WordPress developers who often run into this when nesting anonymous callback functions. For example, you want to make an AJAX request that depends on an earlier AJAX request. Developers can end up creating nested anonymous callback functions which can lead to messy code and difficulty when debugging errors. With modern JavaScript, this can be avoided with Promises or abstraction to named functions.

Ashish Datta

[@adatta02](https://twitter.com/adatta02)

North Jersey born and bred, Ashish graduated from Tufts University with a BS in computer science. Following Tufts, Ashish held engineering roles at ArmchairGM and Wikia before founding [Setfive Consulting](https://setfive.com/). Outside of the office, you can find Ashish in the gym, out for a run, or sipping cocktails with his pinky out.

**“A distinction that many programmers miss is…”**

The difference between asynchronous and concurrent programming. At a high level, asynchronous is the ability to do something while waiting for something else to complete, and concurrent is the ability to compute multiple things at the same time, the big trade off vs. sequential execution being that both asynchronous and concurrent programming add additional complexity.

With the explosion in popularity of nodejs, which features an asynchronous environment, programmers are increasingly using asynchronous programming in situations where it offers no benefits. Generally, CPU bound tasks fall into this bucket and examples would be most machine learning algorithms, the transform step in an ETL pipeline, and any sort of cryptocurrency mining.

Johannes Kadak

[@Qminder](https://twitter.com/Qminder)

Johannes is a Lead Developer at [Qminder](https://www.qminder.com/).

**“Synchronous programming is a better fit for…”**

Sequential tasks, where stopping the entire program to wait for a network request or disk IO makes sense.

Asynchronous programming is a better fit for code that must respond to events – for example, any kind of graphical UI.

An example of a situation where programmers use async but shouldn’t is any code that can focus entirely on data processing and can accept a “stop-the-world” block while waiting for data to download.

Microsoft Developer Network

[@Microsoft](https://twitter.com/Microsoft)

The [Microsoft Developer Network](https://msdn.microsoft.com/en-us/dn308572.aspx) is your resource for development tips, tricks, research, case studies…Everything you need to develop apps that users love.

***NOTE: The following information is excerpted from***[***Asynchronous Programming with Async and Await (C# and Visual Basic)***](https://msdn.microsoft.com/library/hh191443(vs.110).aspx)***via MSDN.***

**“Asynchrony is essential for activities that are potentially blocking, such as when your application accesses the web…”**

Access to a web resource sometimes is slow or delayed. If such an activity is blocked within a synchronous process, the entire application must wait. In an asynchronous process, the application can continue with other work that doesn’t depend on the web resource until the potentially blocking task finishes.

Mark Patterson

[Mark Patterson](https://www.linkedin.com/in/mark-patterson-68a63316/) is a .NET Developer and Founder of HealthSpan Foods.

***NOTE: The following information is excerpted from***[***Await Overhead Approximately 50 + (n \* 15) ms***](https://social.msdn.microsoft.com/Forums/en-US/801ea455-a296-4b01-b3ac-0d38c097b599/await-overhead-approximately-50-n-15-ms?forum=async)***via Microsoft Developer Network forum.***

**“Considering the state gymnastics being performed, the Await operator is very efficient. The performance penalty on a fast machine appears to be approximately…”**

50 + (n \* 15) milliseconds

Where 50 ms is a one time setup delay deferred until runtime the first time an asynchronous method encounters an Await operator. The penalty for each Await operation (including the first one) is approximately 15 ms.

I arrived at these rough estimates by comparing the times required to load images synchronously vs. asynchronously. Obviously your results will vary depending on the speed of your machine, the size of the state being saved and reloaded, and the number of competing threads that might delay scheduling the continuance.

However, this little exploration suggests to me that:

1) It doesn’t make sense to Await tasks which RELIABLY consume <= 20 ms. Obviously an unreliable task, even a very short one – say 1 or 2 ms – could still be a very good candidate for Awaiting. For example, if an IO task fails 1 out of 100 times it should be considered to be very unreliable, and Awaiting the completion of this task would be one way to manage this uncertainty.

2) There is zero penalty for coding an Async method if the Await operator is never encountered. This makes it painless to write methods which can run synchronously or asynchronously depending on an IsAsync parameter passed to the method. In pseudo code something like this:

Private Async Sub DoSomething(ByVal IsAsync as Boolean)

Dim Result as T

If IsAsync Then

Result = Await SomethingTask

Else

Result = Something

End if

UseResult(Result)

End Sub

Private Function Something() as T

Dim Result as T

...

Return Result

End Function

Private Function SomethingTask() as Task(of T)

Dim f = New Func(of T)(AddressOf Something)

Dim s = TaskScheduler.FromCurrentSynchronizationContext

Dim tf = New TaskFactory(of T)(s)

Return tf.StartNew(f)

End Function

(Note the CurrentSynchronizationContext used above is only required if your Async task needs to run on the same thread as the Async method was invoked on. In Silverlight, ANYTHING that touches the UI must be  
on the UI thread, for example.)

Brown University Computer Science

[@BrownCSDept](https://twitter.com/BrownCSDept)

Since 1979, [Brown’s Department of Computer Science](http://cs.brown.edu/) has forged a path of innovative information technology research and teaching.

***NOTE: This information is excerpted from***[***Introduction to Asynchronous Programming***](http://cs.brown.edu/courses/cs168/s12/handouts/async.pdf)***via the Brown University Computer Science Program.***

**“Compared to the synchronous model, the asynchronous model performs best when…”**

* There are a large number of tasks so there is likely always at least one task that can make progress.
* The tasks perform lots of I/O, causing a synchronous program to waste lots of time blocking when  
  other tasks could be running.
* The tasks are largely independent from one another so there is little need for inter-task communication  
  (and thus for one task to wait upon another).

These conditions almost perfectly characterize a typical busy network server (like a web server) in a  
client-server environment. Each task represents one client request with I/O in the form of receiving the  
request and sending the reply. A network server implementation is a prime candidate for the asynchronous  
model, which is why Twisted and Node.js, among other asynchronous server libraries, have grown so much  
in popularity in recent years.

Maria Yakimova

[@djangostars](https://twitter.com/djangostars/)

[Maria Yakimova](https://www.linkedin.com/in/yakimova/?ppe=1) is a Python/Django developer at [Django Stars](http://djangostars.com/).

***NOTE: The following information is excerpted from***[***Asynchronous Programming in Python | Asyncio (Guide)***](http://djangostars.com/blog/asynchronous-programming-in-python-asyncio/)***via Django Stars.***

**“In a classic sequential programming, all the instructions you send to the interpreter will be executed one by one. It is easy to visualize and predict the output of such a code. But…”**

Say you have a script that requests data from 3 different servers. Sometimes, depending on who knows what, the request to one of those servers may unexpectedly take too much time to execute. Imagine that it takes 10 seconds to get data from the second server. While you are waiting, the whole script is actually doing nothing. What if you could write a script that could, instead of waiting for the second request, simply skip it and start executing the third request, then go back to the second one, and proceed from where it left? That’s it. You minimize idle time by switching tasks.

Still, you don’t want to use an asynchronous code when you need a simple script, with little to no I/O.

One more important thing to mention is that all the code is running in a single thread. So if you expect that one part of the program will be executed in the background while your program will be doing something else, this won’t happen.

Brij Bhushan Mishra

[@code\_wala](https://twitter.com/code_wala)

Brij Bhushan Mishra has more than 8 years’ experience and is currently working as an Architect/Consultant. He has had a passion for learning about computers since childhood. He blogs at [Code Wala](https://codewala.net/).

***NOTE: The following information is excerpted from***[***Concurrency vs Multi-threading vs Asynchronous Programming : Explained***](https://codewala.net/2015/07/29/concurrency-vs-multi-threading-vs-asynchronous-programming-explained/)***via Code Wala.***

**“There are two things which are very important for any application – Usability and Performance…”**

Usability because, say, a user clicks on a button to save some data. This requires multiple smaller tasks like reading and populating data in internal object, establishing a connection with SQL and saving it there, etc. As SQL runs on another machine in the network and runs under a different process, it could be time consuming and may take a bit longer. So, if the application runs on a single thread then the screen will be in a hanged state until all the tasks complete, which is a very bad user experience. That’s why nowadays many applications and new frameworks completely rely on the asynchronous model.

Performance of an application is also very important. It has been seen that while executing a request, around 70-80% of the time gets wasted while waiting for the dependent tasks. So, it can be utilized to the max by asynchronous programming, where once the task is passed to another process (say SQL), the current thread saves the state and is available to take another task. When the SQL task completes, any thread which is free can take it up further.

Eric Vogel

[@VSMdev](https://twitter.com/VSMdev)

[Eric Vogel](https://www.linkedin.com/in/eric-vogel-1b6b62/) is a columnist at [Visual Studio Magazine](https://visualstudiomagazine.com/Home.aspx). He’s a Senior Software Engineer at Red Cedar Solutions Group and President at Great Lansing User Group for .NET.

***NOTE: The following information is excerpted from***[***Asynchronous Programming in .NET: I’ll Call You Back***](https://visualstudiomagazine.com/articles/2011/03/24/wccsp_asynchronous-programming.aspx)***via Visual Studio Magazine.***

**“Asynchronous programming is a means of parallel programming in which a unit of work runs separately from the main application thread and notifies the calling thread of its completion, failure or progress…”**

You may be wondering when you should use asynchronous programming and what are its benefits and problem points.

The main benefits one can gain from using asynchronous programming are improved application performance and responsiveness. One particularly well suited application for the asynchronous pattern is providing a responsive UI in a client application while running a computationally or resource expensive operation.

The .NET Framework provides a few avenues to get on the ramp to asynchronous programming. Some of your implementation choices from the most basic to complex include using background workers, invoking a method asynchronously from a delegate, or implementing the IAsynchResult interface. All of these options allow you to multi-thread your application without ever having to manage your own threads. The .NET Framework asynchronous APIs handle this drudgery for you.

[](https://cta-redirect.hubspot.com/cta/redirect/207384/2560dee6-b9ad-4455-a6da-be9e74991e71)

Peter Olson

[@pluralsight](https://twitter.com/pluralsight)

Peter Olson contributes to hack.guides() on Pluralsight. You can also find him on [GitHub](https://github.com/peterolson).

***NOTE: The following information is excerpted from***[***Introduction to asynchronous JavaScript***](https://www.pluralsight.com/guides/front-end-javascript/introduction-to-asynchronous-javascript)***via Pluralsight.***

**“In asynchronous programs, you can have two lines of code (L1 followed by L2), where L1 schedules some task to be run in the future, but L2 runs before that task completes…”**

You can imagine as if you are eating at a sit-down restaurant. Other people order their food. You can also order your food. You don’t have to wait for them to receive their food and finish eating before you order. Similarly, other people don’t have to wait for you to get your food and finish eating before they can order. Everybody will get their food as soon as it is finished cooking.

The sequence in which people receive their food is often correlated with the sequence in which they ordered food, but these sequences do not always have to be identical. For example, if you order a steak, and then I order a glass of water, I will likely receive my order first, since it typically doesn’t take as much time to serve a glass of water as it does to prepare and serve a steak.

Note that asynchronous does not mean the same thing as concurrent or multi-threaded. JavaScript can have asynchronous code, but it is generally single-threaded. This is like a restaurant with a single worker who does all of the waiting and cooking. But if this worker works quickly enough and can switch between tasks efficiently enough, then the restaurant seemingly has multiple workers.

The setTimeout function is probably the simplest way to asynchronously schedule code to run in the future:

// Say "Hello."

console.log("Hello.");

// Say "Goodbye" two seconds from now.

setTimeout(function() {

console.log("Goodbye!");

}, 2000);

// Say "Hello again!"

console.log("Hello again!");

If you are only familiar with synchronous code, you might expect the code above to behave in the following way:

* Say “Hello”.
* Do nothing for two seconds.
* Say “Goodbye!”
* Say “Hello again!”

But setTimeout does not pause the execution of the code. It only schedules something to happen in the future, and then immediately continues to the next line.

* Say “Hello.”
* Say “Hello again!”
* Do nothing for two seconds.
* Say “Goodbye!”

Joy George Kunjikkuru

[@OrionSIOfficial](https://twitter.com/orionsiofficial)

[Joy George Kunjikkuru](https://www.linkedin.com/in/joymon) is a passionate software engineer holding deep expertise in the Microsoft technologies stack (.Net Windows Form, ASP.NET, Silverlight, SQL Server) as well as in open web technologies including JavaScript, AngularJS, TypeScript, HTML5, NodeJS, Azure, C#, ASP.Net MVC, WPF, WCF, Linq, and many more. Joy holds a MBA degree from Bharatiar University, Coimbatore and a B.Tech degree in Computer Science from Anna University, Chennai.

***NOTE: The following information is excerpted from C# async and await programming model from scratch via Orion.***

**“Asynchronous programming is essential when we develop any application because it avoids waiting in the main thread on long running operations such as disk I/O, network operations, database access, etc…”**

In a normal case, if our program needs something to be done from the results of these long operations, our code is stuck until the operation is done and we proceed from that point.

Using the async mechanism, we can just trigger long running operations and continue with other tasks. These long running operations do the job in a different thread and when they complete it, they notify our main code, and our code can do the post actions from here. When we refer to our code, we are talking about our main thread which deals with the user interface or the thread that primarily processes a web request.

Sathyaish Chakravarthy

Sathyaish Chakravarthy has been programming since early 1997. His area of expertise is the Microsoft .NET framework. He can also work in other languages such as Java, Kotlin, Python, Ruby, JavaScript, and C/C++. He is a freelance full-stack developer. Find him on [GitHub](https://github.com/Sathyaish) and [StackOverflow](https://stackoverflow.com/users/303685/water-cooler-v2).

***NOTE: The following information is excerpted from***[***What is the benefit of using asynchronous programming on a single core machine?***](https://www.quora.com/What-is-the-benefit-of-using-asynchronous-programming-on-a-single-core-machine)***via Quora.***

**“All operations a computer program performs can be categorized into two broad categories…”**

* Compute-bound operations; and
* I/O-bound operations

Compute-bound operations use the CPU to do some work. Examples are crunching some numbers, doing mathematical calculations such as finding the square root of a number or the value of the constant Pi to a fixed, but rather large number of decimal points, interpolating the missing bits in a bitmap, performing an operation on two bitmaps or two layers of a bitmap, or performing data compression.

I/O-bound operations, on the other hand, do not use the CPU at all. They send a request to the I/O-device-driver and wait for an interrupt from the driver to inform them of the readiness of the result of their requested operation. Examples of I/O-bound operations are reading from a disk or writing to the disk, i.e., reading or writing to files. This can take various forms such as executing a query on a database, reading from or writing to a database, reading from or writing to a network socket, sending a request to a CD-ROM drive, reading or writing to memory, receiving input from the keyboard or mouse, or displaying input on the standard output device, i.e., the monitor, etc.

When a program performs an I/O-bound operation, it sends a request to the underlying operating system API to further direct its request to the appropriate device driver.

Every OS includes both a synchronous version and an asynchronous version of API for I/O-bound operations.

If your computer program makes a call to the synchronous version of your operating system’s I/O-bound API, then while it’s waiting on the results from the device driver, it is essentially blocked from doing anything else. The thread that was allocated to perform the operation simply has to wait for the results. All this time while the thread waits for results from the device driver, it will never be scheduled by the operating system to be on the ready-queue. Consequently, it will get no CPU time. Thus, if your program needs to perform another concurrent operation at this time, it needs an additional thread.

Creating an additional thread involves a significant performance overhead. Also, it reduces scalability since more threads are now doing what needs no doing by any thread, i.e., to simply wait on a device driver.

On the other hand, if your computer program makes a call to the asynchronous version of your operating system’s I/O API, the thread allocated to perform that operation simply issues a request to the device driver and immediately returns. It is now free to do any other work that your program wishes to perform concurrently. If there is nothing required to be done concurrently, it can continue to listen to messages from the operating system’s message queue / dispatcher.

That means if your program is running in an event-driven environment such as a graphical user interface environment, it will provide a more responsive user interface to the user, allowing him to click willy-nilly while the I/O-bound operation is still in progress.

Whether or not your machine has a single core, calling an I/O bound API’s asynchronous version provides better scalability to your application since it obviates the need to create threads when they are not required.

Richard Rodgers

[@KeyNCar](https://twitter.com/KeyNCar)

[Richard Rodgers](https://www.linkedin.com/in/richard-rodgers-107571/) is an entrepreneur, currently Software Architect and Developer at [Key and Car Tracking LLC](http://www.keyandcartracking.com/). Previously, he founded TechMate Inc. and has served as the company President since 1993.

***NOTE: The following information is excerpted from***[***What situations in programming may we need to use an asynchronous operation?***](https://www.quora.com/What-situations-in-programming-may-we-need-to-use-an-asynchronous-operation)***via Quora.***

**“When you need to wait for another process to complete, it can be very useful to go asynchronous…”**

This permits you to continue other operations instead of simply waiting. The operation could be simply a long running process, such as a significant graphic transformation or mathematical calculation. Or, more commonly, it is some type of input/output (IO) operation. Often, you want to run many of these same types of processes simultaneously and asynchronous coding allows you to do that.

Event-driven communications is a common asynchronous paradigm. Basically, you set up communications and provide code, called a callback, that should run when the communication occurs.

For example, I am currently using WebSockets. So, I open the socket and provide callbacks for it including those for receiving a message, recognizing an error, or closing the socket. Whenever a message arrives, my message callback runs. If an error occurs, I handle the error in that callback. If my communication partner terminates, my close callback cleans up my environment. Using these callbacks, I can run off and do something else knowing that, if something happens that I need to handle, the appropriate callback will trigger.

Asynchronous coding often means that you need to multi-thread your code. This means that you have to start another thread that can run independently of your main task. This is often necessary because, as an example, waiting on communication to complete completely stops the thread that is waiting from running. The term for this is “blocking.” If you issue a communication command that is blocking, it will stop and wait for arrival of a message or, if no message arrives, that command will timeout. Running this in a separate thread prevents your main program from having to wait.

Most high-level languages provide libraries that support event-driven and/or asynchronous programming.

Try Stackify’s free code profiler, [Prefix](https://stackify.com/prefix), to write better code on your workstation. Prefix works with .NET, Java, PHP, Node.js, Ruby, and Python.

# # #

Asynchronous programming is actually [easier than you may think](https://stackify.com/asynchronous-programming-easier-think/). Check out [this post](https://stackify.com/return-ajax-response-asynchronous-javascript-call/) to learn how to return AJAX response from an [asynchronous JavaScript call](https://stackify.com/async-javascript-approaches/). Or, read [this post](https://stackify.com/csharp-async-await-task-performance/) for more information on how Microsoft has made asynchronous programming simple with the implementation of async await in C# and how the latest versions of ASP.NET are utilizing it to boost performance.

-

-

-

-

-

-

-

-

-

-

-

-