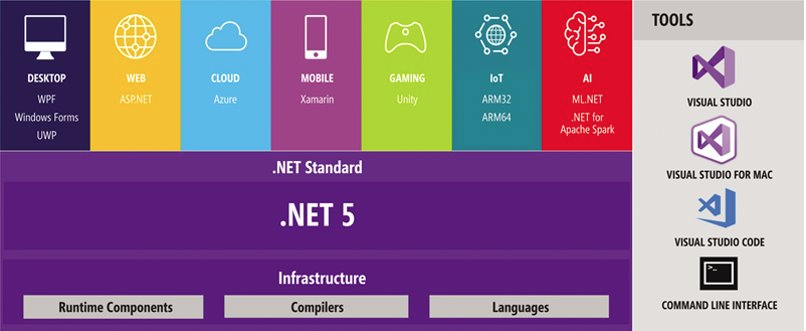
# .Net Architecture Components

* There are many frameworks that is already included in our .Net 5.0
  + A framework just has predefined classes and libraries to help us start an application.
  + One framework we will be using in the future is ASP .Net (used to develop web application)

## SDK vs. Runtime

* Software Development Kit (SDK) it includes everything we need to build and run a .Net application.
  + You can run them either using Command Line interface (CLI) or you can use an Integrated development environment (IDE).
* Runtime
  + It includes all the resource we need to run an existing .Net application
  + A lot less storage required to download and install
    - Mostly used for end-users.



## .Net standard

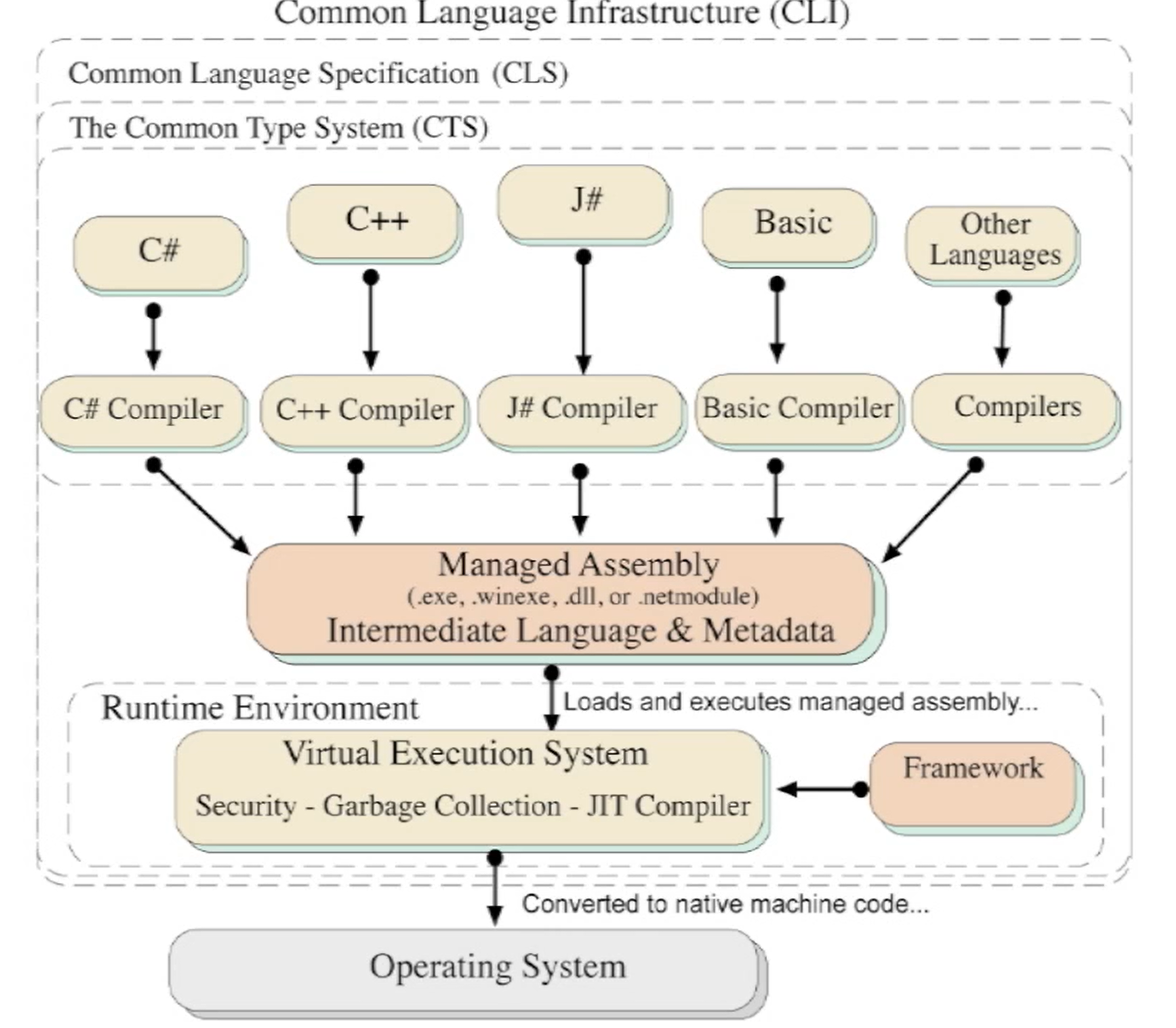
* It is a set of APIs that are implemented by the **Base Class Library** (BCL) which provides the classes and types that are helpful in performing day to day operations.
  + Ex: dealing with strings, primitives types (value types), database connection, etc.
* With .Net standard, you can use multiple language to create your application.
  + Your apps can run in F#, Visual Basics, and other languages
    - They are .Net compliant language
* Each version of .Net standard are additive
  + Any previous versions of .Net will be included to the current version of .Net

## Managed Code

* It is a code whose execution is managed by runtime.
* The CLR uses the managed code and compiles into machine code and then executes it.

## Unmanaged Code

* Code developed outside of the .Net environment.
* Not manage by the CLR.
* Can be executed with the help of wrapper classes.



# CLI

* Stands for Common Language Infrastructure.
* This enables an application program to be written in numerous programming language to be run on any operating system.
* It is comprised of:
  + CLS
    - CTS
    - VES
      * CLR

## CLS

* Stands for Common Language Specification.
* They will define the rules and restriction that every language must follow for it to be able to run the .Net framework.
* **CTS**
  + Common Type System
  + Provides a library of the basic primitive data types.
  + Ensures that every data type in each programming language is talking about the same thing.
    - i.e. int data type in C#,J#, VB has to be 32-bit
    - This lets us create **Language Interoperability**.

## VES

* Run-time system of CLI
* It provides an environment for executing managed code
* **CLR**
  + Stands for Common Language Runtime
  + It is essentially .Net framework’s implementation of VES
  + This lets us able to run our application across multiple systems (Windows, Mac, Linux, etc.)
    - Automatic memory management (older language you have to manually release unused resources).
    - JIT compilation – Just-in-time compilation involves compiling the code while executing for optimization.
    - Exception handling support

## Garbage Collection

* CLR provides automatic memory management of your heap memory.
* It checks for objects in the managed heap that are no longer being used by the app.
* However, resources that are unmanaged code needs to be clean up manually.
  + We can use IDisposable interface to clean up those external resources.
* You can also use *using* statement for clean up.
* Pros:
  + No memory leaks.
  + Efficient memory allocation.
  + Garbage collection ensures that one object cannot use the memory of another object.
* Cons:
  + Since this process is automated, it can negatively impact the performance of your code especially when garbage collection must regularly run through the program and checking if the object is being referenced and needs cleaning up.

# 

# Data types

* We can use data types to structure the data and tell the compiler to how we intend to use that data.
* Fun fact! Unlike java, all types inherit from the Object base class.
* They are two major types:
  + Value types
  + Reference types

## Value types

* They are same as Java primitive types.
* They derived from the System.ValueType which inherites System.Object.
* Stores in the **stack** and not the heap.
* It means you get the value directly and not a reference to where the value is stored.
* There is no garbage collection for value-type variables.
  + Memory is automatically reclaimed when a stack frame is removed.

## Reference types

* They do not have the value directly, but they have a variable that reference that value.
  + Ex: classes, delegates, arrays, or interfaces, etc.
* When you declare a variable of a reference type and not have it point to anything in the beginning, it will have a null value.
* They are stored in the **heap** and the reference variable is stored in the **stack.**
* Like in Java, string are also classes, but main difference is we can use “==” to compare two strings since .Net is smart enough to not compare the reference value and compare the actual value.

# Collections

* It is a data structure that can hold many values.
* All collections have methods add, remove, or find items since they all inherit from Innumerable Interface.
* In C#, there are two major types of collection: Generic and non-generic.

## Generic

* They allow you to type safe collection.
  + Meaning you can add a data type in the collection and only that data type is allowed to be stored.
* They come from System.Collections.Generic namespace.
* The “T” you see in documentation is where you can put that data type that collection will hold.
* List of Generic Collections:
  + List<T> - It grows automatically as you add element, zero-based index
  + Stack<T> - LIFO
  + Queue<T> - FIFO
  + Dictionary<TKey, TValue> - A collection that represents a key/value pair
  + SortedList<TKey, TValue> - same as dictionary buuut it automatically order the key in (you decide)
  + HashSet<T> - contains non-duplicated elements

## Non-generic

* They allow you to store multiple data types.
* They come from System.Collection namespace.
* List of non-generic collections
  + ArrayList – non-generic version of list
  + SortedList – non-generic version of sortedlist
  + Queue – FIFO
  + Stack
  + Hashtable