Programming

Introduction

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

Topics

- Course introduction
- Problem solving and algorithm
- Program development steps
- .NET Framework

Course introduction

- This unit introduces students to the core concepts of programming with an introduction to algorithms and the characteristics of programming paradigms
- On successful completion of this unit students will be able to design and implement a simple computer program in a chosen language (C#) within a suitable IDE (Visual Studio .NET)

Learning outcomes

- LO1: Define basic algorithms to carry out an operation and outline the process of programming an application.
- LO2: Explain the characteristics of procedural, object-orientated and event-driven programming, conduct an analysis of a suitable Integrated Development Environment (IDE)
- LO3: Implement basic algorithms in code using an IDE.
- LO4: Determine the debugging process and explain the importance of a coding standard

Course preparation

- Drawing tools (choose one):
  - Visio
  - Draw.io or Lucichart (online)
  - Astah (recommendation, using student email to register full version)
- IDE
  - Visual Studio Community 2017

Problem solving

- It’s a creative process, it is an act of
  - Defining a problem
  - Determining the cause of the problem
  - Identifying, prioritizing, and selecting alternative for a solution
  - Implementing a solution

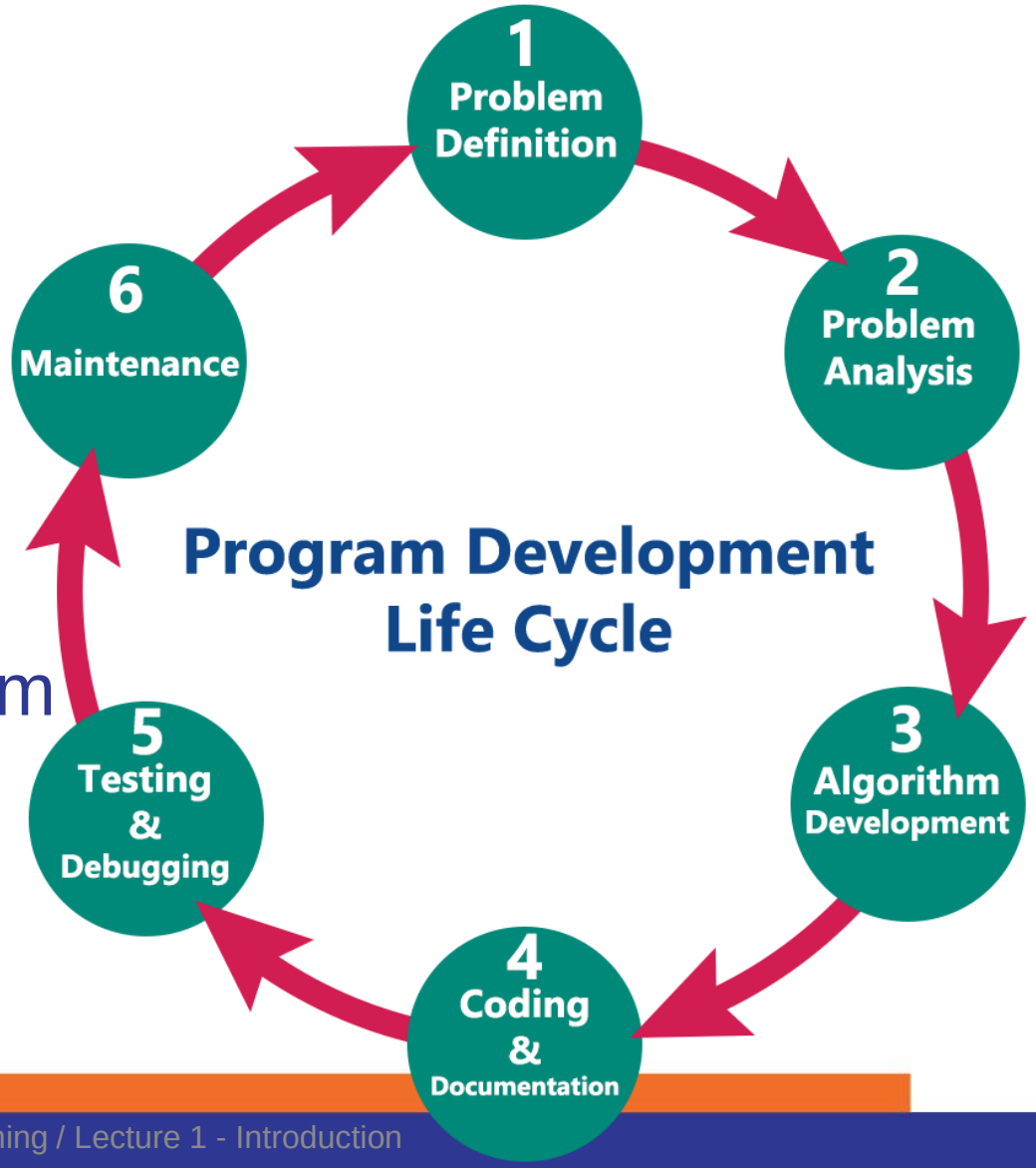



Algorithm

- An algorithm is a step-by-step description of the solution to a problem
- An algorithm must be
  - Definite
  - Finite
  - Precise and Effective
  - Implementation independent

Steps in Program Development

- The various steps involved are
  - Defining or Analyzing the problem
  - Design (Algorithm)
  - Coding
  - Documenting the program
  - Compiling and running the program
  - Testing and Debugging
  - Maintenance

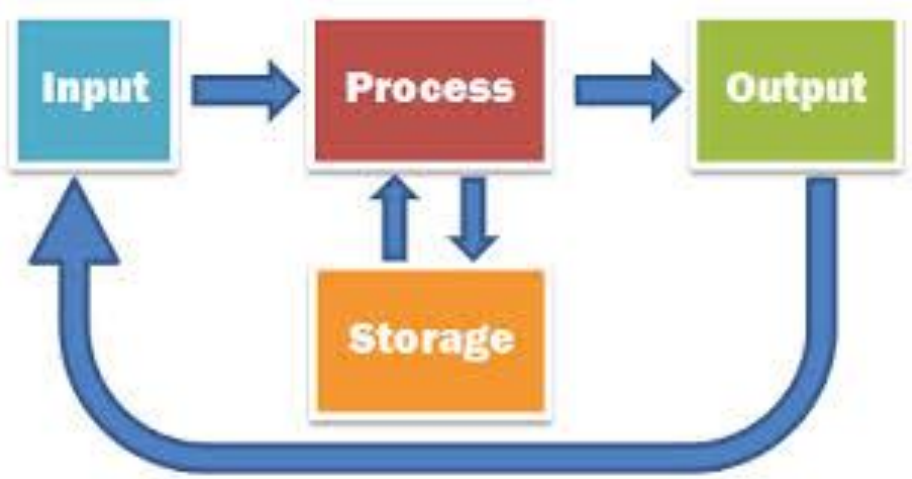


STEPS IN Program Development: Analyzing or Defining the Problem

- The problem is defined by doing a preliminary investigation
- Defining a problem helps us to understand problem clearly
- It is also known as Program Analysis

Tasks in defining a problem

- Followings are the tasks in order to define a problem
  - Specifying the input requirements
  - Specifying the output requirements
  - Specifying the processing requirem



Specifying the input requirements

- The input specification is obtained by answering following questions
  - What specific values will be provided as input to the program?
  - What format will the values be?
  - For each input item, what is the valid range of values that it may assume?
  - What restrictions are placed on the use of these values?

Specifying the output requirements

- The output specification is obtained by answering the following questions
  - What values will be produced?
  - What is the format of these values?
  - What specific annotation, headings, or titles are required in the report?
  - What is the amount of output that will be produced?

Specifying the processing requirements

- The processing requirement is obtained by answering following questions
  - What is the method (technique) required in producing the desired output?
  - What are the validation checks that need to be applied to the input data?
  - What calculations are needed?

Activity: Find Factorial Number

- Input?
- Output?
- Process?

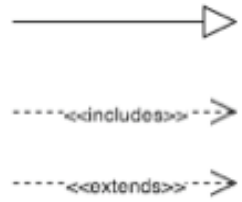
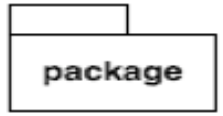
Activity Find Factorial Number

- Input: Positive integer number
- Output: Factorial of that number
- Process: Solution technique which transforms input to output. Factorial of a number can be calculated by the formula n!=1*2*3*…*n

Use-case diagram

- Use case diagrams are used to gather the requirements of a system
- So when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.
- The purposes of use case diagrams can be as follows:
  - Used to gather requirements of a system
  - Used to get an outside view of a system
  - Identify external and internal factors influencing the system
  - Show the interacting among the requirements are actors

Main elements of use-case diagram

Element	Description	Symbol
Actor	An actor is a person, organization, or external system that plays a role in one or more interactions with your system	 Actor
Use-cases	A use case describes a sequence of actions that provide something of measurable value to an actor	 Use Case
Associations	Associations between actors and use cases: An association exists whenever an actor is involved with an interaction described by a use case	
Some other relationships	The relationships among actors, or use-cases like inheritance, extends, includes, etc.	
System boundary boxes (optional)	A rectangle around the use cases, called the system boundary box, to indicates the scope of your system.	
Packages (optional)	UML constructs that enable you to organize model elements (such as use cases) into groups	

Activity: Draw use-case diagram

- You are hired to develop FAI's library system with following description
- Admin who could
  - Manage books, readers (staff, lecturers, and students), etc.
  - Manage borrow/return books
- Users (staff, lecturers, students) who could
  - View/search books
  - Reserve books
  - Borrow/return books
- Please draw Use-case diagram for this scenario

STEPS IN Program Development: Design

- A design is the path from the problem to a solution in code
- The well designed program is likely to be:
  - Easier to read and understand later
  - Less of bugs and errors
  - Easier to extend to add new features
  - Easier to program in the first place

Modular Design

- Once the problem is defined clearly, several design methodologies can be applied
- An important approach is Top-Down program design
- It is structured design technique
  - It breaks up the problem into a set of sub-problems called Modules
  - It creates a hierarchical structure of the modules

Flowchart

- Flowchart is a diagrammatic representation of an algorithm
- It uses different symbols to represent the sequence of operations, required to solve a problem
- It serves as a blueprint or a logical diagram of the solution to a problem

Flowchart symbols (1/2)

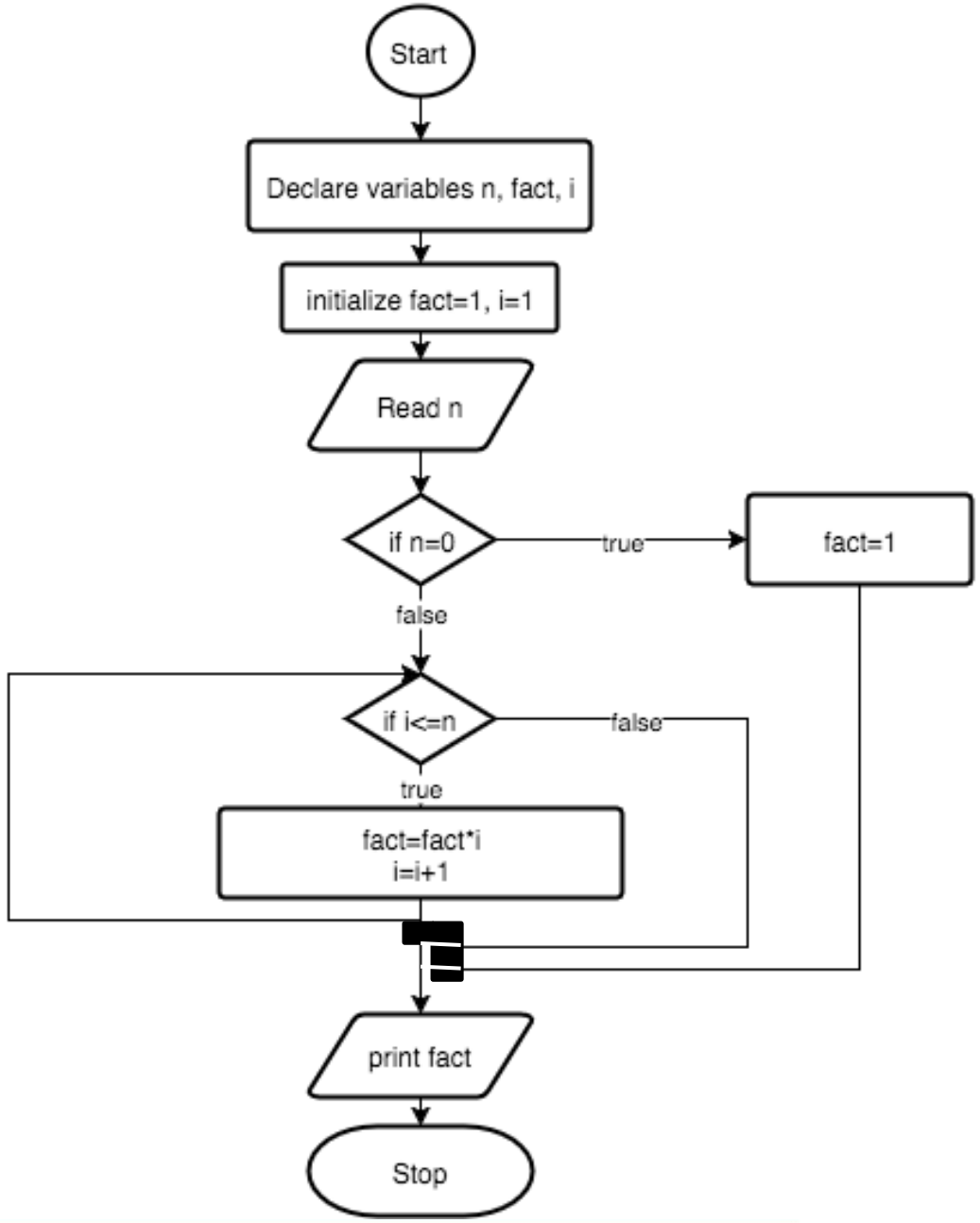
	Represents Start, End
	Represents Input, Output data
	Represents Process (actions, calculations)
	Represents Decision Making
	Represents Pre-defined Process / module

Flowchart symbols (2/2)

	Represents off page connector which are used to indicate that the flow chart continues on another page. Page numbers are usually placed inside for easy reference.
	**Connector Symbol** represents the exit to, or entry from, another part of the same flow chart. It is usually used to break a flow line that will be continued elsewhere.
	The **Document Symbol** is used to represent any type of hard copy input or output (i.e. reports).
	Represents control flow

Activity: Drawing flov

- Draw a flowchart for the algorithr in previous activity



STEPS IN Program Development: Coding

- An algorithm expressed in programming languages is called Program
- Writing a program is called Coding
- The logic that has been developed in the algorithm is used to write program

STEPS IN Program Development: Documenting the Program

- Document explains
  - How the program works and how to use the program (user manual)
  - How to maintain the program (developer manual)
- Details of particular programs, or particular pieces of programs, are easily forgotten or confused without suitable documentation

Forms of documentation

- Documentation comes in two forms
  - External documentation, which includes things such as reference manuals, algorithm descriptions, flowcharts, and project workbooks
  - Internal documentation, which is part of the source code itself (essentially, the declarations, statements, and comments)

Compiling and Executing the Program

- Compilation is a process of translating a source program into machine understandable form
- The compiler is system software
  - It examines each instruction for its correctness
  - It does the translation
- During the execution
  - Program is loaded into the computer's memory
  - The program instructions are executed

STEPS IN Program Development: Testing

- Testing is the process of executing a program with the deliberate intent of finding errors
- Testing is needed to check whether the expected output matches the actual output
- Testing is done during every phase of program development
- Initially, requirements can be tested for its correctness
- Then, the design (algorithm, flow charts) can be tested for its exactness and efficiency

Test criteria

- Programs are tested with several test criteria and the important ones are given below
  - Test whether each and every statement in the program is executed at least one (Basic path testing)
  - Test whether every branch in the program is traversed at least once (control flow)
  - Test whether the input data flows through the program and is converted to an output (data flow)

STEPS IN Program Development: Debugging

- Debugging is a process of correcting the errors
  - Programs may have logical errors which cannot be caught during compilation
  - Debugging is the process of identifying their root causes
  - One of the ways is to print out the intermediate results at strategic points of computation
  - Another way is to use support from the IDE
- Testing vs Debugging
  - Testing means detecting errors
  - Debugging means diagnosing and correcting the root causes

STEPS IN Program Development: Maintenance

- Program maintenance
  - Continuing process of maintenance and modification
  - To keep pace with changing requirements and technologies
- Maintainability of the program is achieved by
  - Modularizing it
  - Providing proper documentation for it
  - Following standards and conventions (naming conventions, using symbolic constants, etc.)

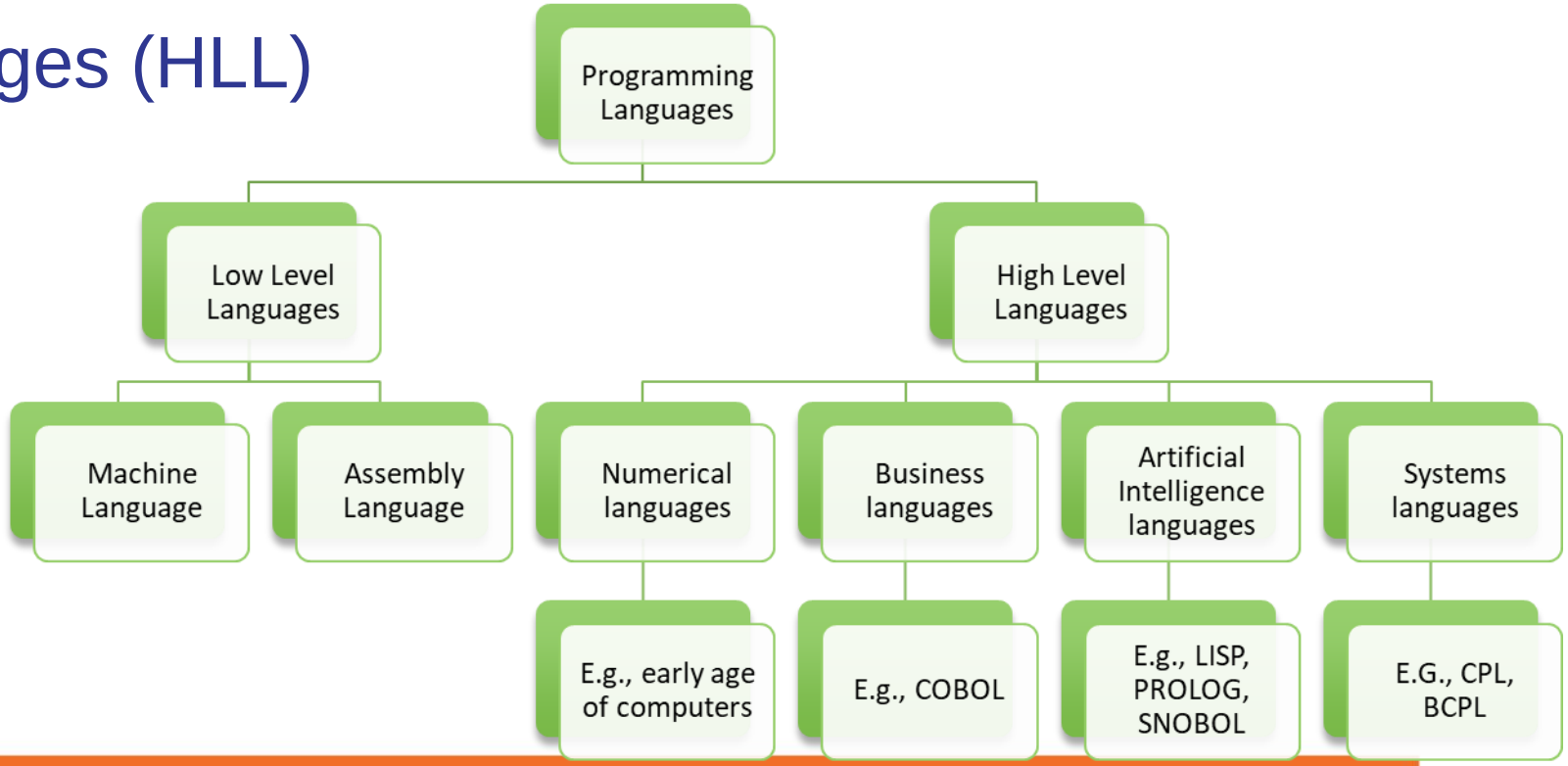
INTRODUCTION TO PROGRAMMING LANGUAGE

What is a Programming Language?

- It is an art of making a computer to do the required operations
  - By means of issuing sequence of commands to it
- It can be defined as
  - A vocabulary (unique set of characters/keywords)
  - A set of grammatical rules (syntax)
- The term programming languages usually refers to high-level languages
  - E.g., BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal

Types of programming languages

- There are two major types of programming languages
  - Low level languages (LLL)
  - High level languages (HLL)



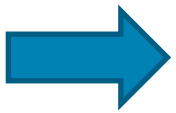
What makes a good language?

- Every language has its strengths and weaknesses
- FORTRAN is good for numeric data but not good to organize large program
- PASCAL is good for structured and readable programs, but it is not as flexible as C
- C++ has powerful object-oriented features, but it is complex and difficult to learn
- The choice of PL depends on type of the computer used, type of program, and the expertise of the programmer

Development Environments

- Programming on Host Environment:
  - The environment under which a program is designed, coded, tested & debugged
- Operating on Target Environment
  - The external environment which supports the execution of a program Target

Hello.c
Hello.exe



Hello.ex
e

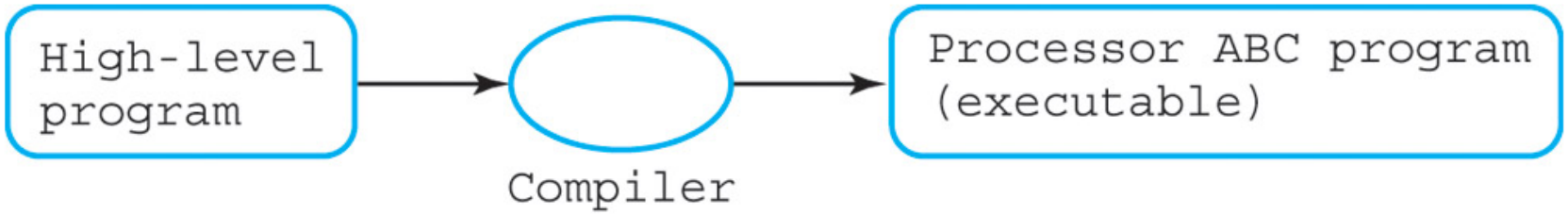
.NET FRAMEWORK

.NET Framework

- Common Intermediate Language (CIL)
- Common Language Runtime (CLR)
- Just-In-Time (JIT) Compiler
- Common Language Specification

Compiler

- Software that translate high-level language (C++, Java, C#, etc) to machine language.



- Compiler X can covert high-level Y to machine language Z.

Complier Example

C++ Code

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!\n";
}
```

C++ Complier for Intel

```
.MODEL Small
.STACK 100h
.DATA
   db msg 'Hello, world!$'
.CODE
start:
   mov ah, 09h
   lea dx, msg ; or mov dx, offset msg
   int 21h
   mov ax,4C00h
   int 21h
end start
```

Intel x86 machine code

C++ Complier for Motorola

```
;print
     move.l    #Hello,-(A7)
     move.w    #9,-(A7)
     trap      #1
     addq.l    #6,A7

;wait for key
     move.w    #1,-(A7)
     trap      #1
     addq.l    #2,A7

;exit
     clr.w     -(A7)
     trap      #1

Hello
     dc.b      'Hello, world!',0
```

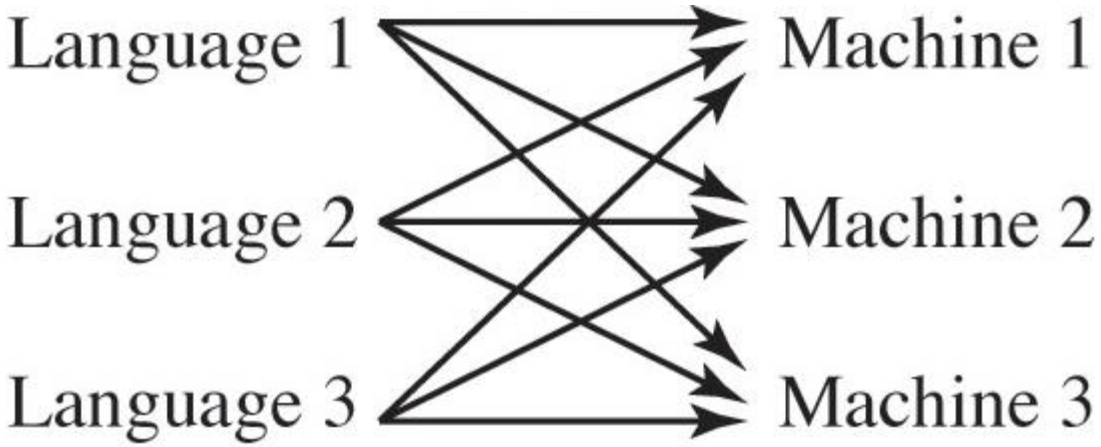
Motorola 68000 machine code

Problem

- A big problem facing developers is the many different types of processors that run code.
- Windows, Macintosh, and Unix machines use a wide variety of hardware, as do personal digital assistants, cell phones, large computers, and other platforms.
- One way to make a program work on each of these devices is to translate the program to the native instruction

Problem

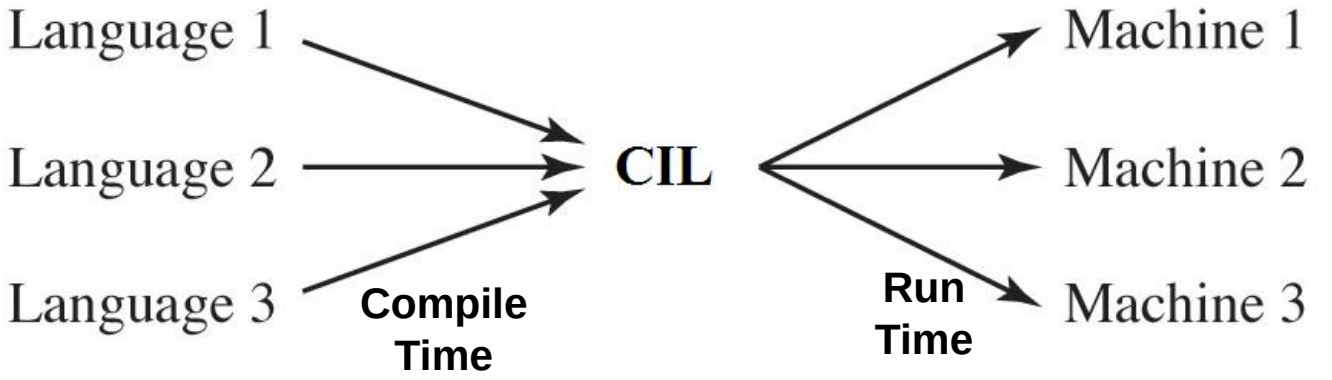
- So if we have 3 programming languages and 3 devices, how many compilers do we need?



- So, how they solved this?!

Two Steps Compilation Process

- Compilation is done in two steps:
  - At compile time: compile each language (C#, C++, etc) to Common Intermediate Language (CIL)
  - At runtime: Common Language Runtime (CLR) uses a Just In Time (JIT) compiler to compile the CIL code to the native code for the device used



Common Intermediate Language (CIL)

- Much like the native languages of devices.
- CIL was originally known as Microsoft Intermediate Language (MSIL).
- CIL is a CPU- and platform-independent instruction set.
- It can be executed in any environment supporting the .NET framework
- Hello World Example in CIL

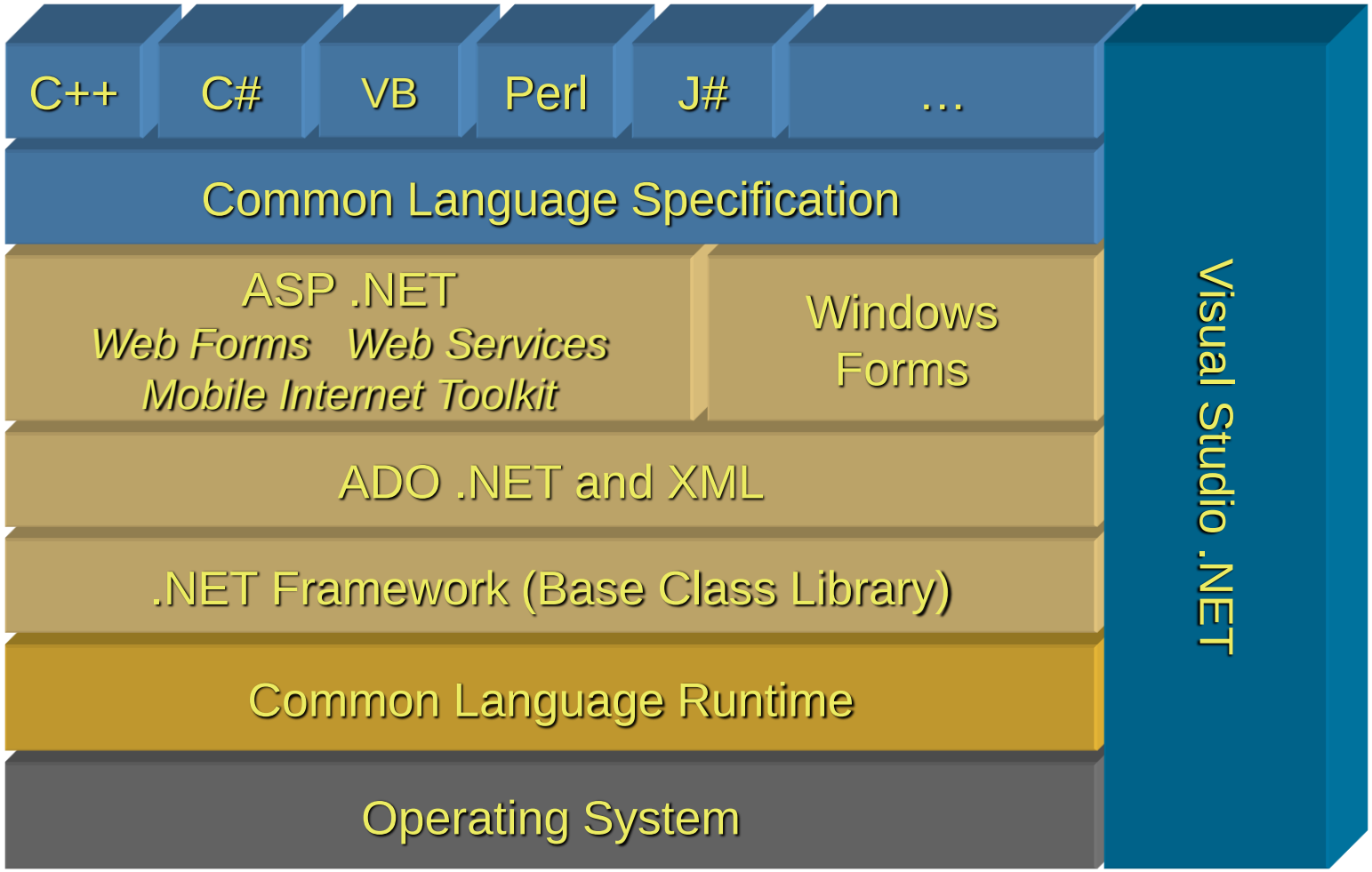
```
.assembly Hello {}
.method public static void Main() cil managed
{
      .entrypoint
      .maxstack 1
      ldstr "Hello, world!"
      call void [mscorlib]System.Console::WriteLine(string)
      ret
}
```

UNIVERSITY of GREENWICH
Alliance with FPT Education

Common Language Runtime (CLR)

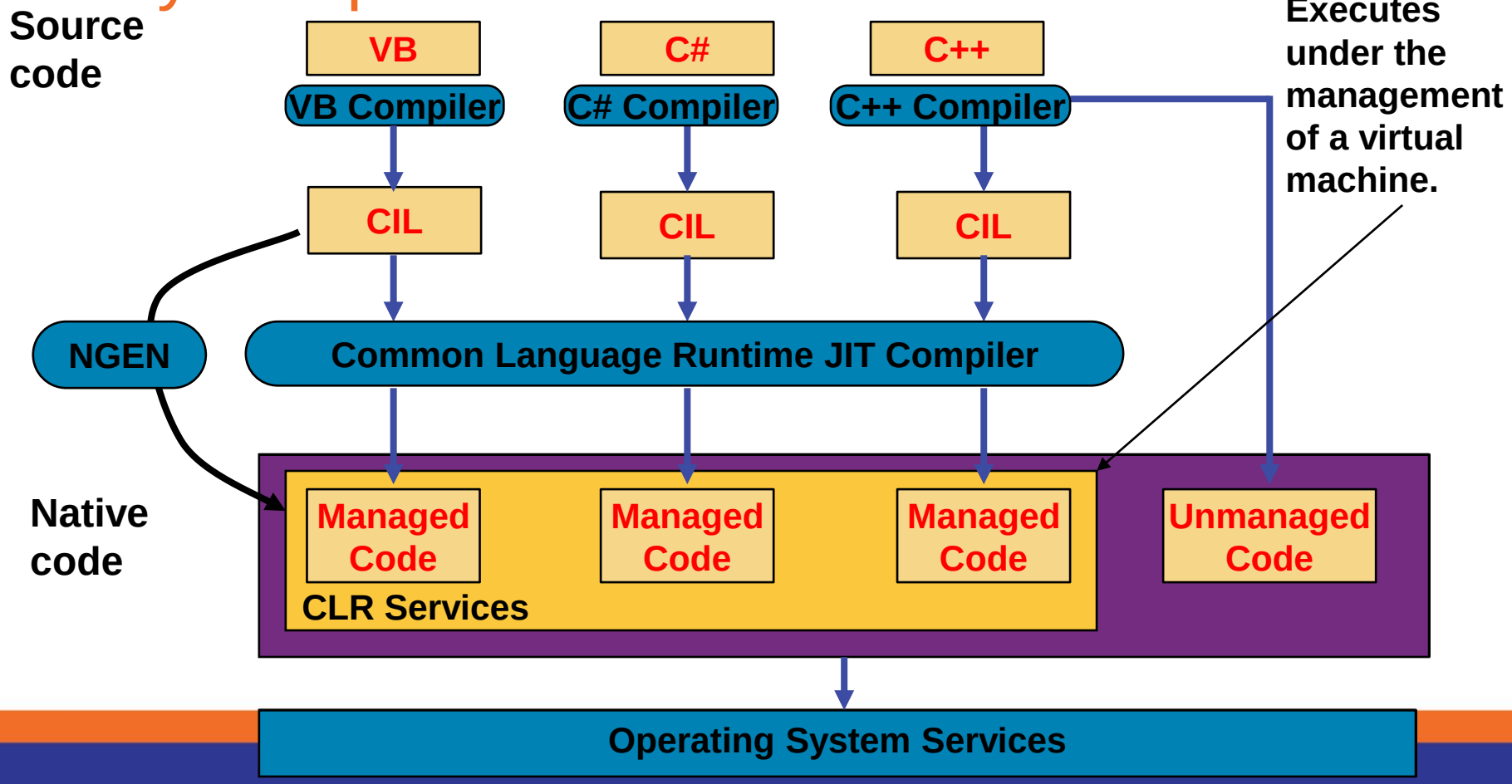
- The Common Language Runtime (CLR) manages the execution of code.
- CLR uses Just-In-Time (JIT) compiler to compile the CIL code to the native code for device used.
- Through the runtime compilation process CIL code is verified for safety during runtime, providing better security and reliability than natively compiled binaries.
- Native image generator compilation (NGEN) can be used to produces a native binary image for the a specific environment. What is the point?

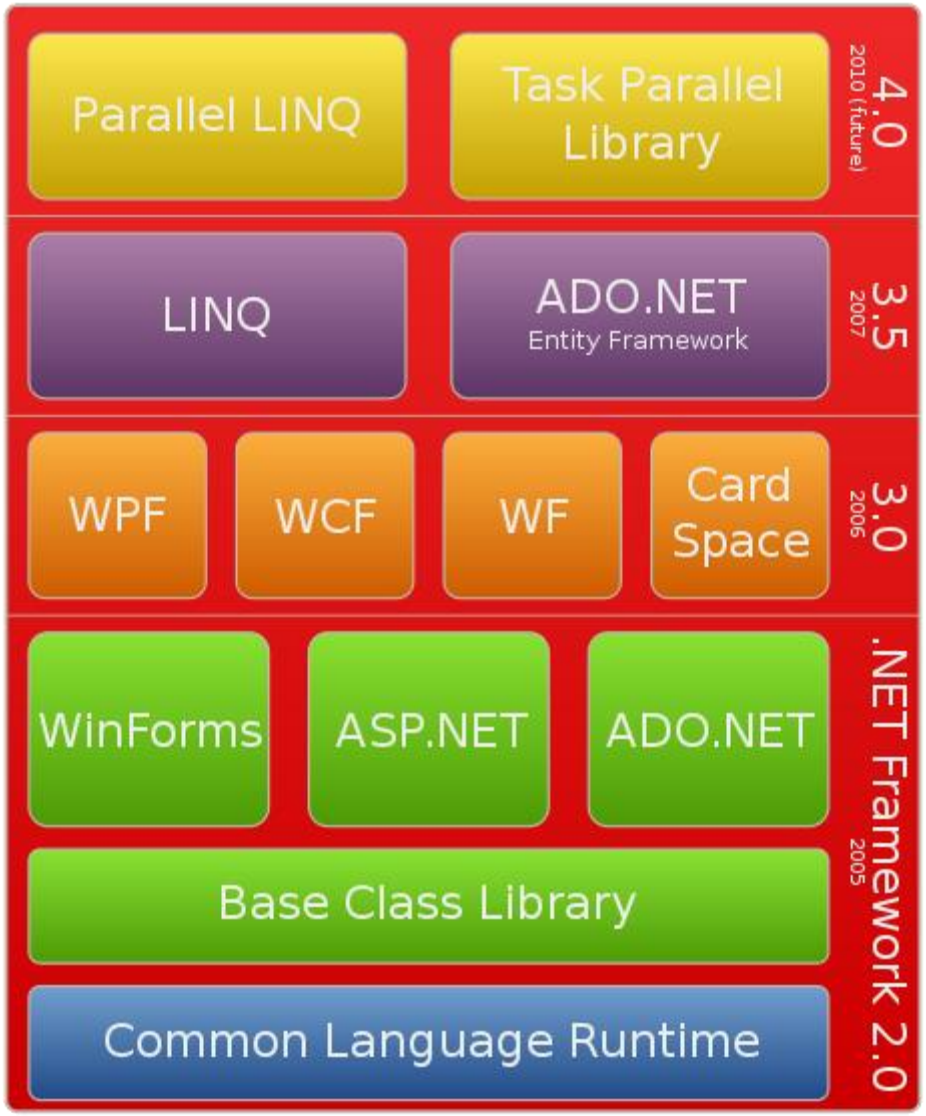
.NET Framework
Visual Studio .NET



Compilation Process

- So if we have 3 programming languages and 3 devices, how many compilers do we need?





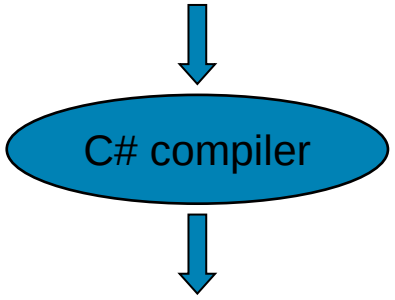
The .NET Framework Stack

CIL

- C# compiler translates C# source code into CIL

C# source

```
Calc c = new Calc();
int sum = c.Add(2, 4);
```



CIL

```
.locals init ([0] class Calc c, [1] int32 sum)
newobj instance void Calc::.ctor()
stloc.0  // c = ptr to new object
ldloc.0
ldc.i4.2 // pass second arg
ldc.i4.4 // pass first arg
callvirt instance int32 Calc::Add(int32,int32)
stloc.1  // sum = retval
```

UNIVERSITY of GREENWICH
Alliance with FPT Education

Platform and Language Independent

- What we have described so far will lead us to Platform independent environment. How?
- Can we use compiled classes written in X language in a program written in Y language?
- VB.NET + C#.NET code

Language interoperability

- All .NET languages can interoperate

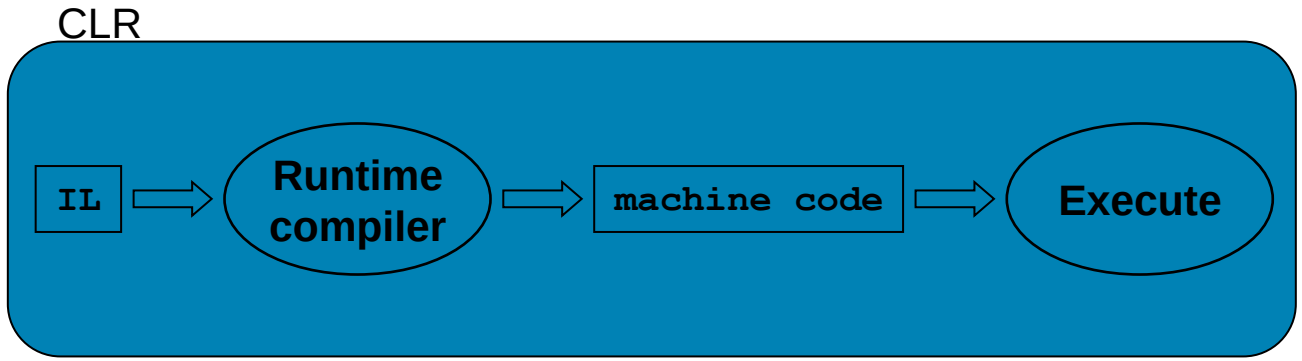
C# calling VB.NET →

```
class Hello
{
  static void Main()
  {
    System.Console.WriteLine(Greeting.Message());
  }
}
```

```
Class Greeting
  Shared Function Message() As String
    Return "hello"
  End Function
End Class
```

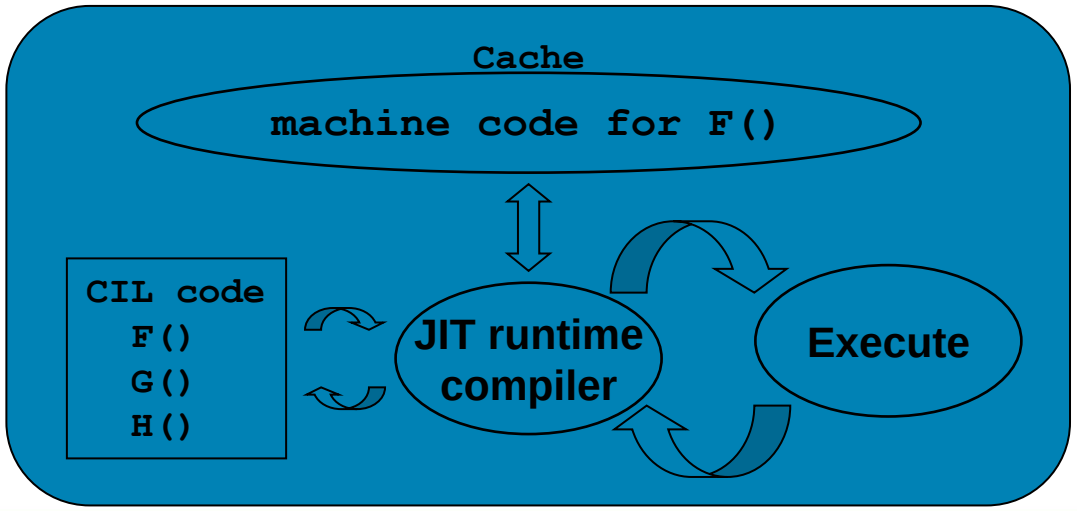
Execution engine

- *Common Language Runtime* (CLR) is the execution engine
  - loads IL
  - compiles IL
  - executes resulting machine code



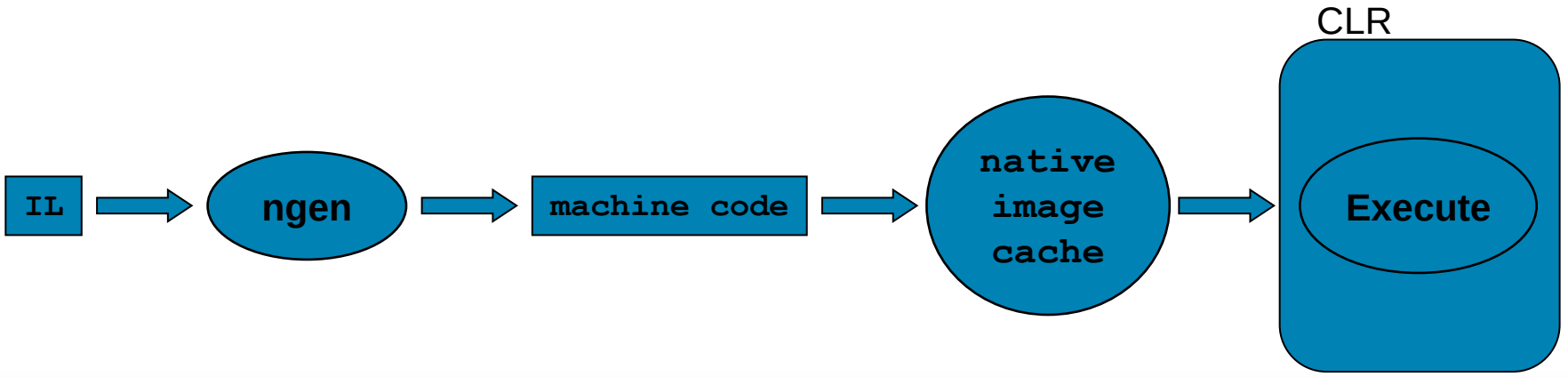
JIT runtime compile

- CIL is compiled into machine code at runtime by the CLR
  - compiles methods as needed
  - called *just in time* (JIT) compile
- JIT compilation model:
  - first time method is called the IL is compiled and optimized
  - compiled machine code is cached in transient memory
  - cached copy used for subsequent calls



NGEN install time compile

- Can compile CIL into machine code when app installed
  - use native image generator **`ngen.exe`**
  - can speed startup time since code pre-compiled
  - but cannot do as many optimizations
  - original IL must still be available for type information



Language variability

- Not all .NET languages have exactly the same capabilities
  - differ in small but important ways

C#

```
class Hello
{
  static void Main()
  {
    int  i;
    uint u;
  }
}
```

signed integer

unsigned integer

VB.NET

```
Class Greeting
  Shared Sub Main()
    Dim i as Integer
  End Sub
End Class
```

signed integer only

Common Language Specification

- Common Language Specification (CLS) defines type subset
  - required to be supported by all .NET languages
  - limiting code to CLS maximizes language interoperability
  - code limited to CLS called *CLS compliant*

not CLS compliant
to use **`uint`** in public
interface of public class

```
public class Calculator
{
  public uint Add(uint a, uint b)
  {
    return a + b;
  }
}
```