TypeScript

Presenter: Ravi Gulve

Course Introduction

- Key concepts of Typescript
- Why to write Typescript
- Some of features Typescript offers
- Tools to be used
- Some other frameworks
- Alternative to typescript

Why use TypeScript ?

- JavaScript can feel messy !

Why use TypeScript ?

- We want maintainable code

JavaScript code encapsulation

Function Spaghetti Code

Ravioli Code
(JavaScript Pattern)

JavaScript Dynamic Types

- Javascript Provides dynamic type system
- **The Good:**
  - Variable can hold any object
  - Types determined on the fly
  - Implicit type coercion (eg. String to number)
- **The Bad:**
  - Difficult to ensure proper types are passed without tests
  - Not all developer use ===
  - Enterprise –scale apps can have 1000 of lines of code to maintain

Migrating from Server-side to Client-side

- Migrating from server-side apps to client side apps can be challenging.

What are alternatives?

- Several Typescript alternative exists:
  - Write pure javascript
  - Apply javascript patterns
  - Coffeescript
  - Dart

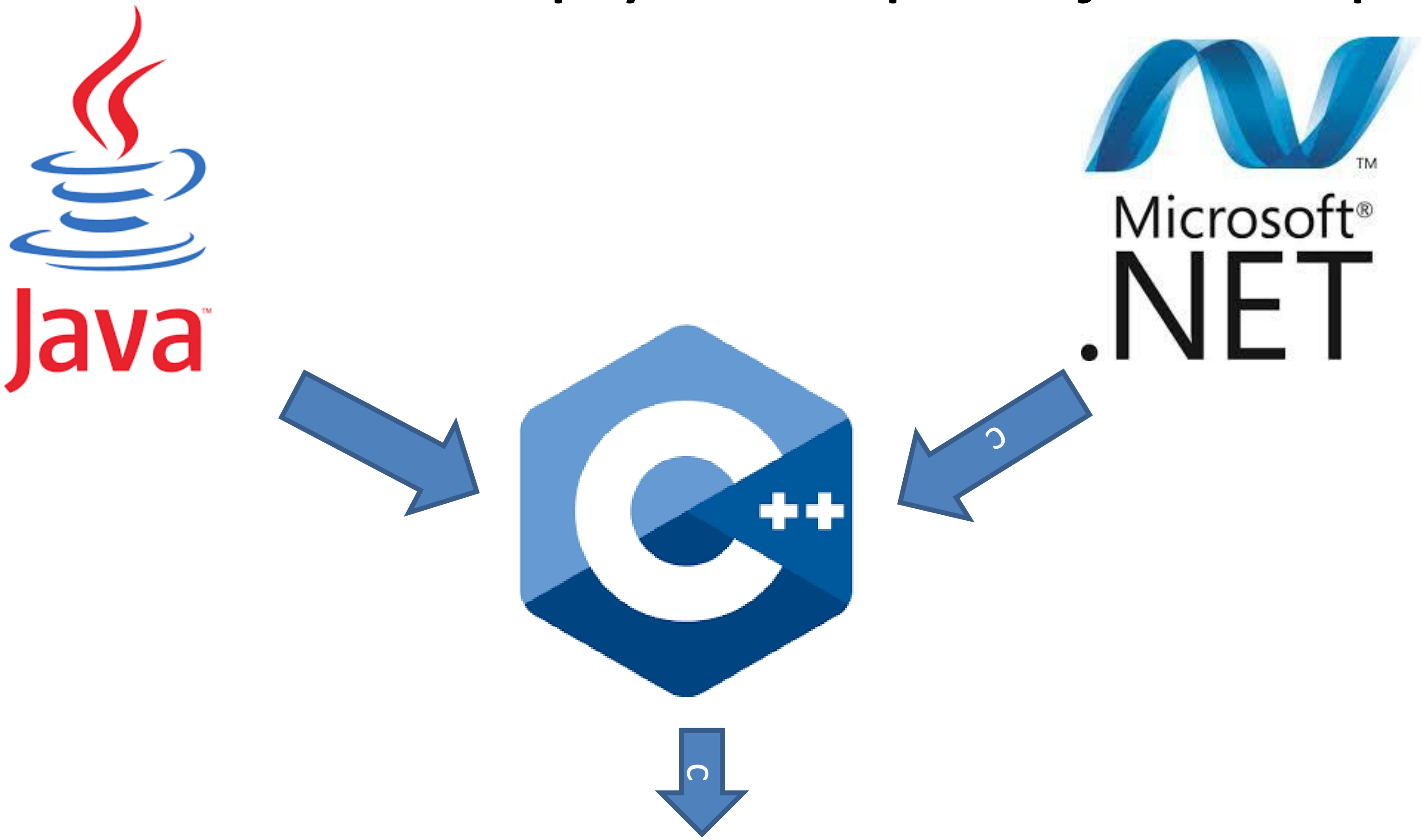
TypeScript

Dart

CoffeeScript

ecma INTERNATIONAL

Shouldn’t we simply write plain javascript?



```
0x52ac76:  movl   7306562(%ebx), %eax
0x52ac7c:  movl   %eax, -20(%ebp)
0x52ac7f:  movl   $0, (%edi,%eax)
0x52ac86:  testl  %esi, %esi
0x52ac88:  je     0x52ad21                      ; -
     [UINavigationController _updateScrollViewFromViewController:
     toViewController:] + 425
0x52ac8e:  movl   7306542(%ebx), %eax
0x52ac94:  movl   (%edi,%eax), %eax
0x52ac97:  movl   %eax, -24(%ebp)
0x52ac9a:  movl   7212558(%ebx), %eax
0x52aca0:  movl   %eax, 4(%esp)
0x52aca4:  movl   %esi, (%esp)
0x52aca7:  calll  0x9bff06                      ; symbol stub for:
     objc_msgSend
0x52acac:  movl   %eax, -28(%ebp)
0x52acaf:  movl   %edx, -32(%ebp)               Thread 1: instruction step over
0x52acb2:  movl   7211062(%ebx), %eax
0x52acb8:  movl   %eax, 4(%esp)
0x52acbc:  movl   %esi, (%esp)
```

Typescript Features

What is TypeScript?

“TypeScript is ty t of JavaScript that compile vaScript.”

http:// lang.org/

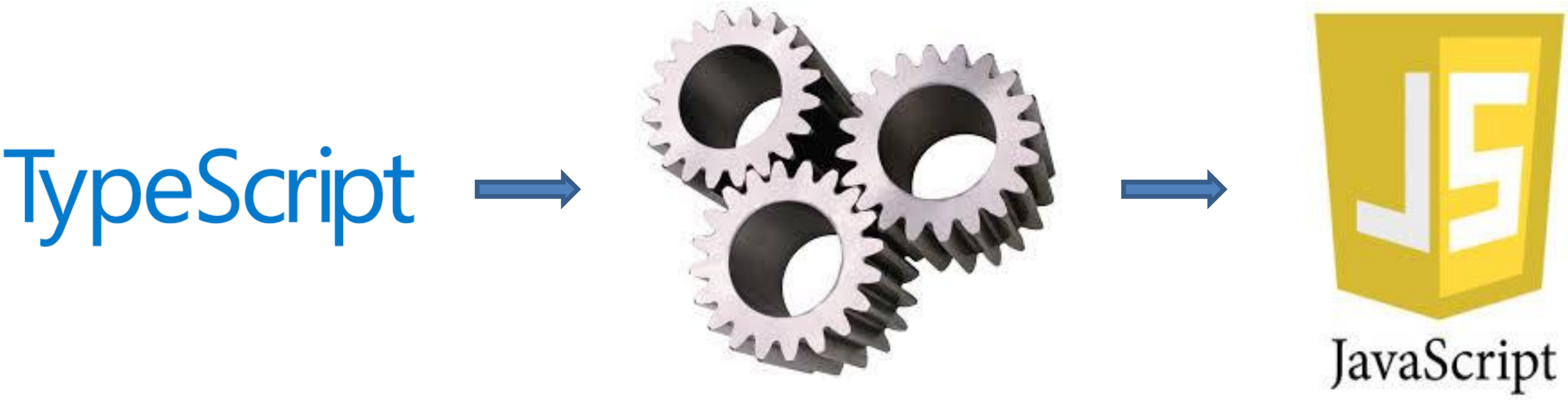
Flexible Options

- Any Browser
- Any Host
- Any OS
- OpenSource
- Tool Support

Key TypeScript Features

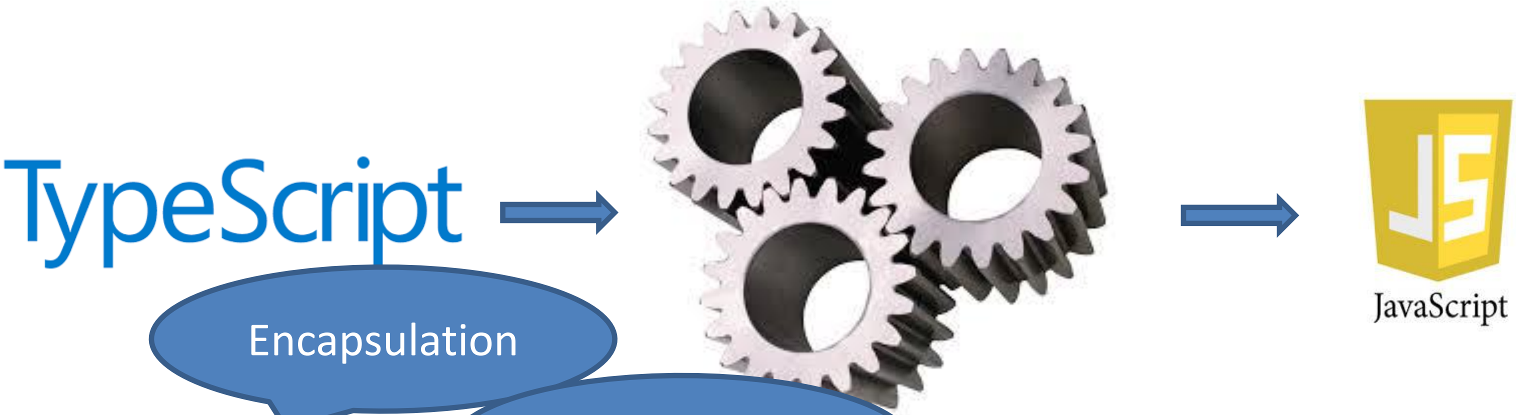
- Support standard Javascript code
- Provides static typing
- Encapsulation through classes and modules
- Support for constructors, properties functions
- Define interfaces
- => Function support (lambdas)
- Intellisense and syntax checking

TypeScript Compiler



tsc first.ts

TypeScript → JavaScript



Encapsulation

Static typing

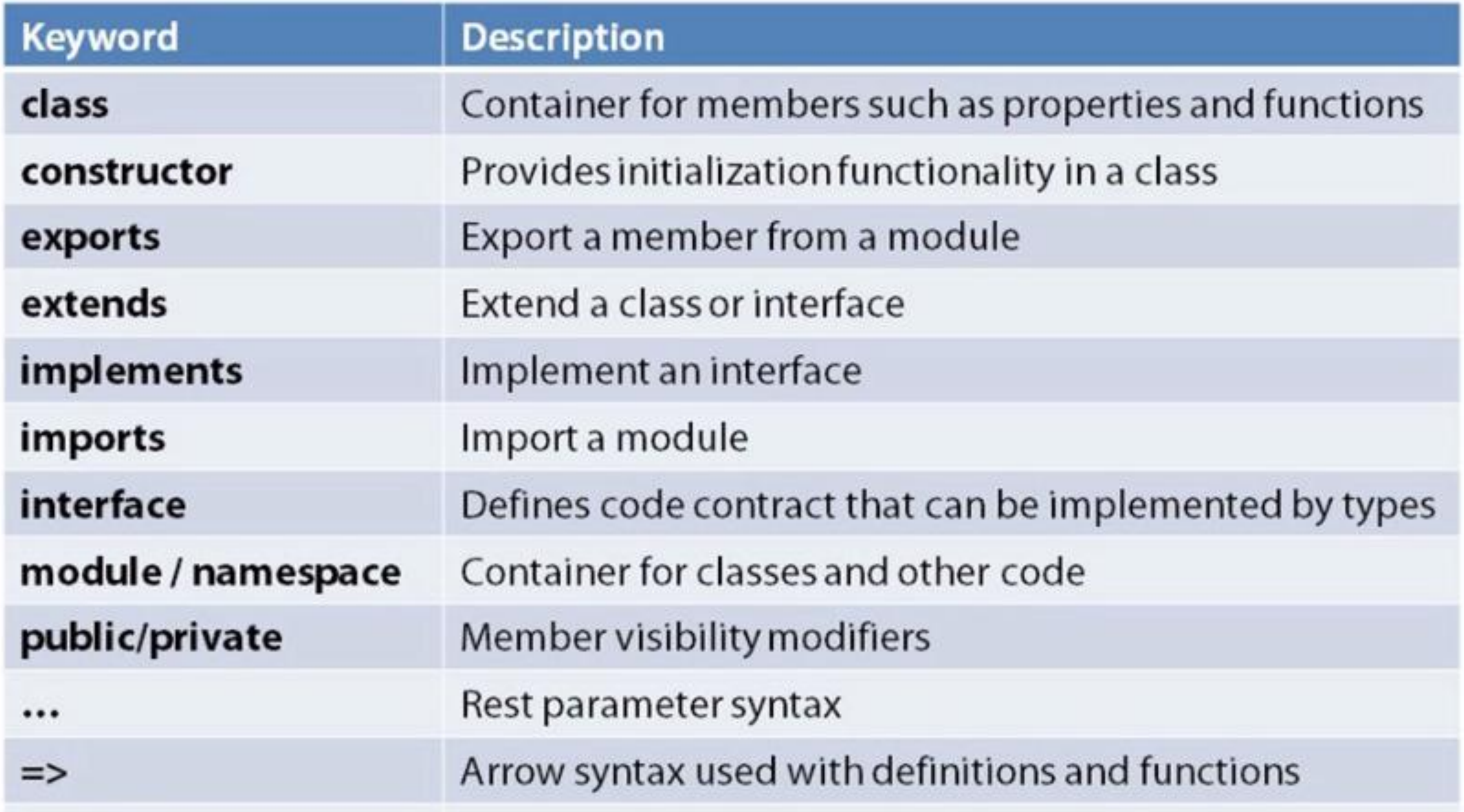
```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```

```
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
```

TypeScript Syntax, Keywords and code Hierarchy

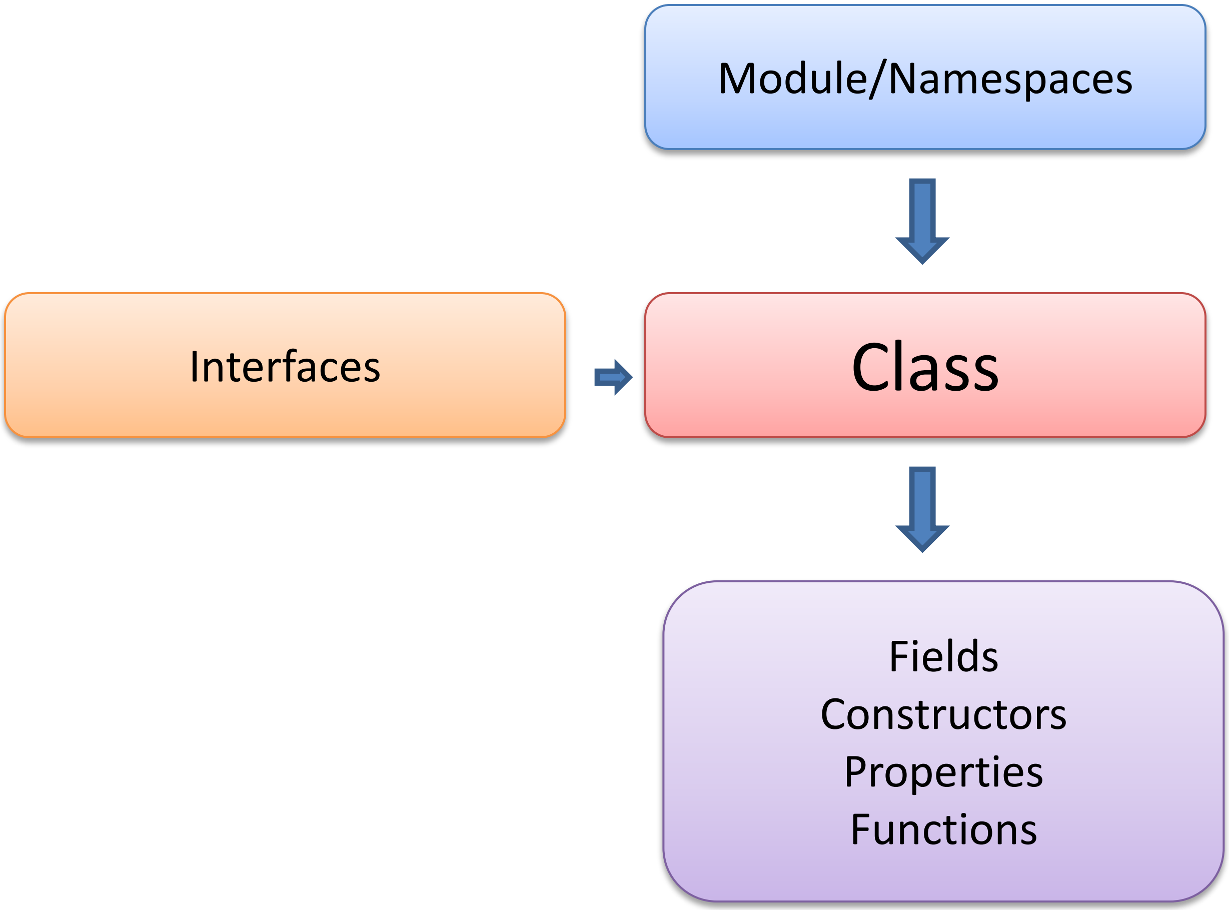
- **Typescript is superset of Javascript**
  - Follow the same syntax rules:
  - {} bracket defines code blocks
  - Semi-colons end code expressions
- **JavsScript keywords:**
  - For
  - If
  - More…

Important Keywords and Operators



Keyword	Description
class	Container for members such as properties and functions
constructor	Provides initialization functionality in a class
exports	Export a member from a module
extends	Extend a class or interface
implements	Implement an interface
imports	Import a module
interface	Defines code contract that can be implemented by types
module / namespace	Container for classes and other code
public/private	Member visibility modifiers
...	Rest parameter syntax
=>	Arrow syntax used with definitions and functions

Code Hierarchy



Module/Namespaces

Interfaces

Class

Fields
Constructors
Properties
Functions

Tool/Framework Support

Node.js

Sublime

Emacs

Vi

Visual Studio

TypeScript Plyground

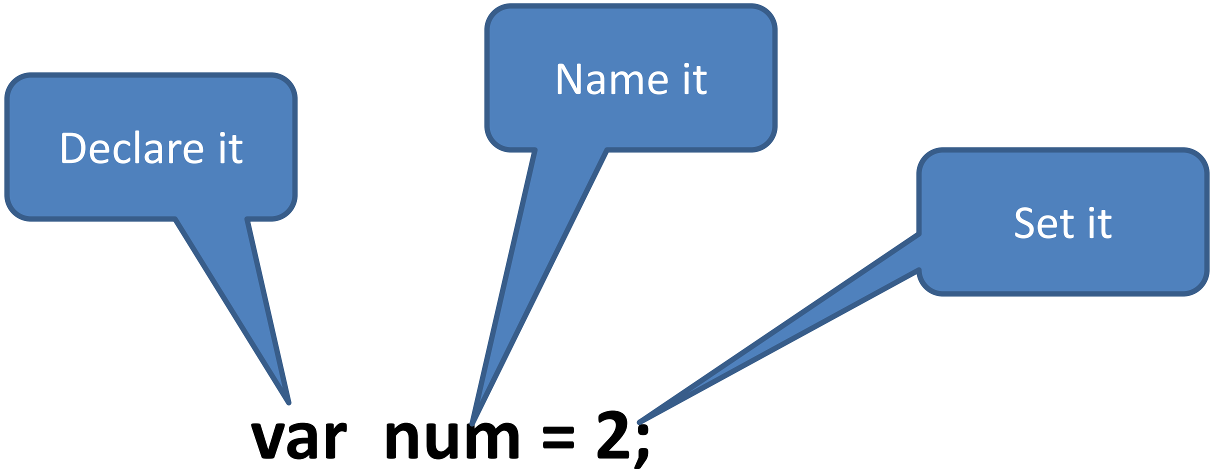
Visual Studio Code

- Free open source code editor
- Minimum memory footprints
- In build Git supports
- Numerous plug-ins

TypeScript types

- **Number**: the “number” is a primitive number type in TypeScript. There is no different type for float or double in TypeScript
- **Boolean**: The “boolean” type represents true or false condition
- **String**: The “string” represent sequence of characters similar to C#
- **Null**: The “null” is a special type which assigns null value to a variable
- **Undefined**: The “undefined” is also a special type and can be assigned to any variable

Type Inference



Declare it

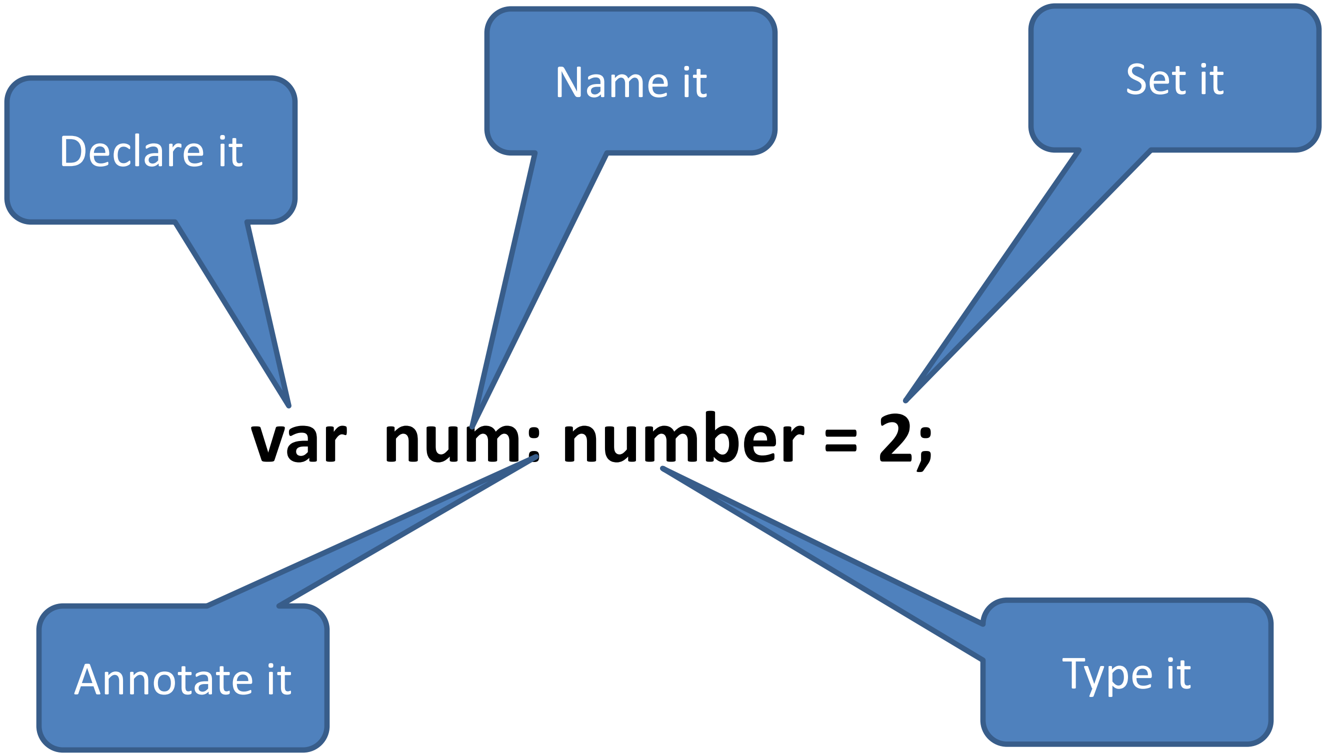
Name it

Set it

```
var  num = 2;
```

This typescript example is javascript

Type Annotations



Declare it

Name it

Set it

```
var  num: number = 2;
```

Annotate it

Type it

Initializing the variable is optional. The initialized variable must match the annotated type

Annotations and Inference

```
var  any1;
```

Type could be any type(any) any type is base type of object. It could be string , int etc.

```
var  num1: number;
```

Type Annotation

```
var  num2: number = 2;
```

Type annotation setting the value

```
var  num3 = 3;
```

Type Inference (number)

```
var  num4 = num3 + 100;
```

Type Inference (number)

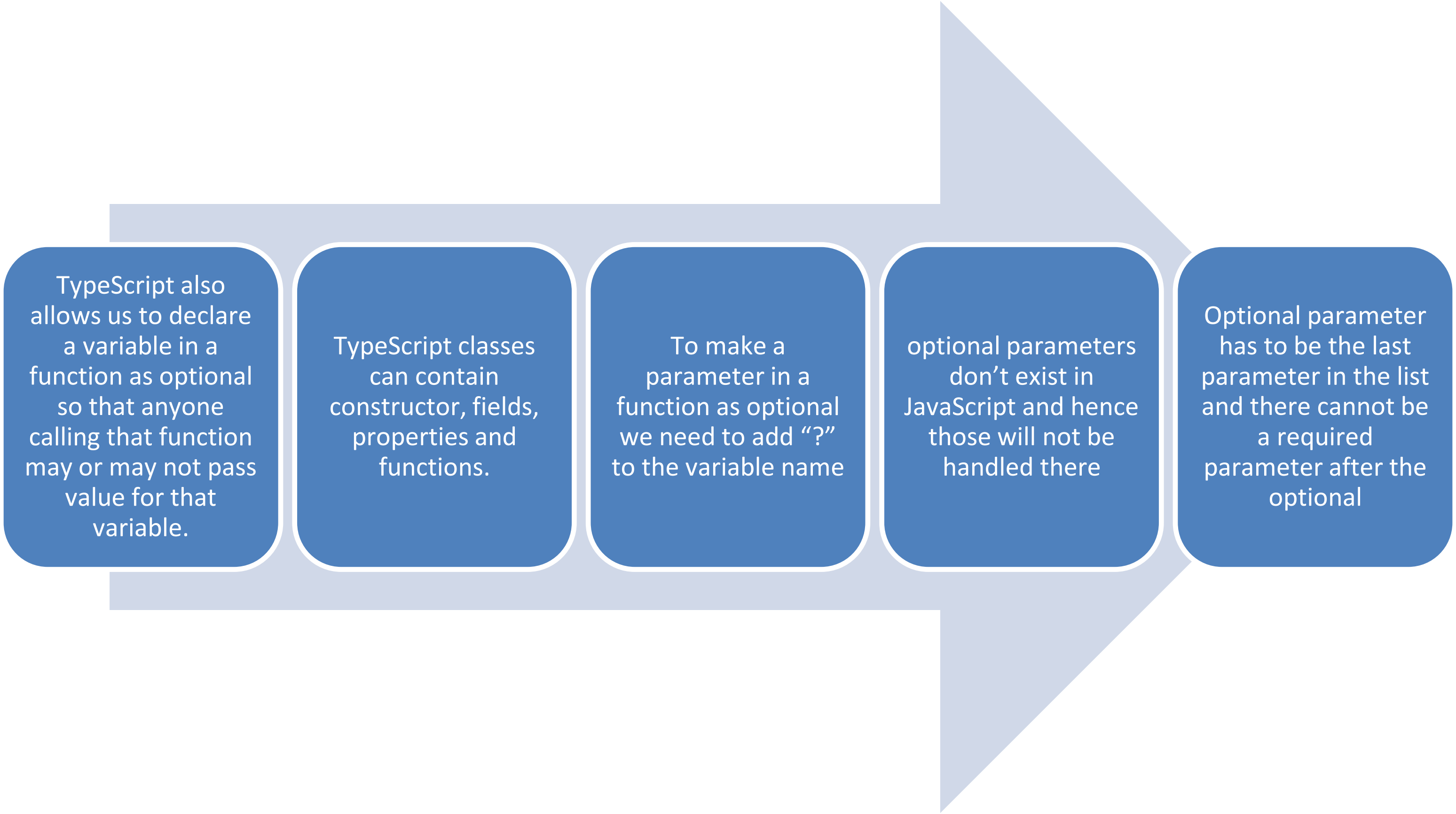
```
var  num4 = num3 + ‘abc’;
```

Type Inference (string)

```
var  nothappy : number = num1 + ‘abc’;
```

Error !

Optional Type



TypeScript also allows us to declare a variable in a function as optional so that anyone calling that function may or may not pass value for that variable.

TypeScript classes can contain constructor, fields, properties and functions.

To make a parameter in a function as optional we need to add “?” to the variable name

optional parameters don’t exist in JavaScript and hence those will not be handled there

Optional parameter has to be the last parameter in the list and there cannot be a required parameter after the optional

Typescript Arrays

This example defines an array that can only hold Customer objects using [] syntax

```
var  custs: Customr[];
```

You can also declare an array as an Array of some specific data type

```
var  custs: Array<Customer>;
```

```
custs = [];
```

This code provides a very basic initialization

```
custs = [new Customer("A123"),
new Customer("B456")];
```

This example initializes my array with two Customer objects using an array literal

Spread Operator

The main objective of the spread operator is to *spread* the elements of an array or object

A common use case is to spread an array into the function arguments. Previously you would need to use Function.prototype.apply:

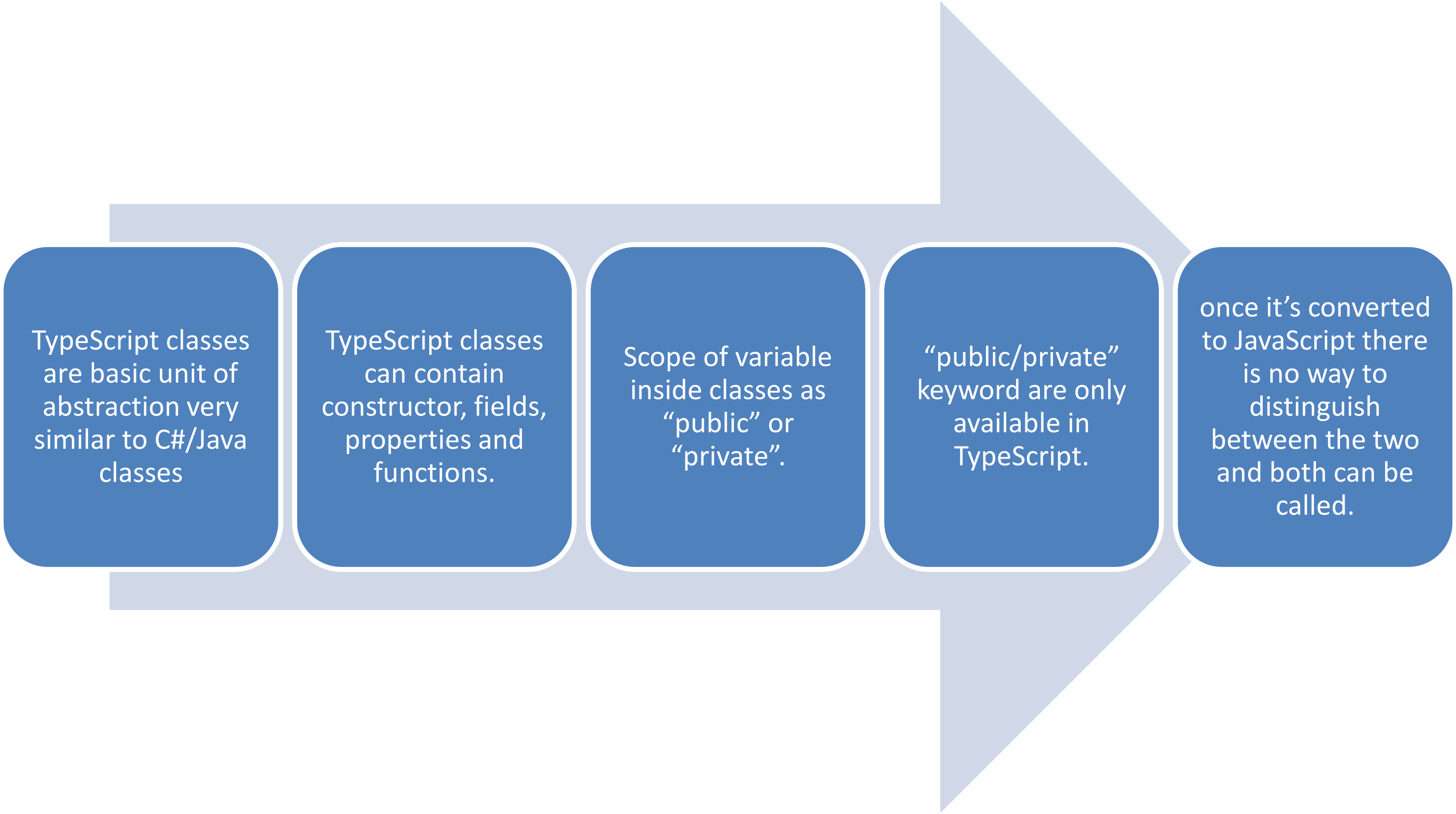
```
function foo(x, y, z) { }
var args = [0, 1, 2];
foo.apply(null, args);
```

Now you can do this simply by prefixing the arguments with ... as shown below:

```
function foo(x, y, z) { }
var args = [0, 1, 2];
foo(...args);
```

Here we are *spreading* the args array into positional arguments.

Classes



TypeScript classes are basic unit of abstraction very similar to C#/Java classes

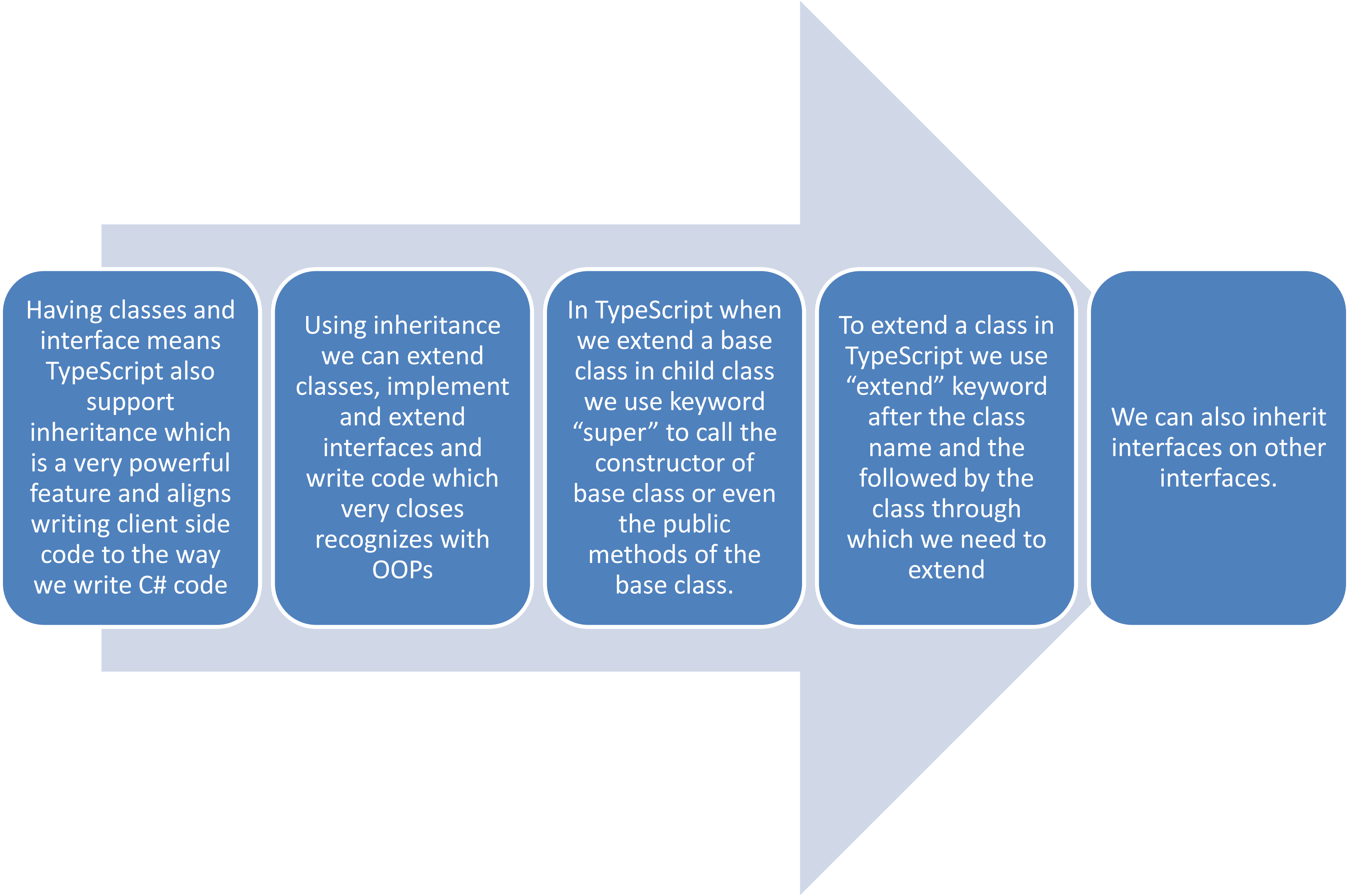
TypeScript classes can contain constructor, fields, properties and functions.

Scope of variable inside classes as “public” or “private”.

“public/private” keyword are only available in TypeScript.

once it’s converted to JavaScript there is no way to distinguish between the two and both can be called.

Inheritance



Having classes and interface means TypeScript also support inheritance which is a very powerful feature and aligns writing client side code to the way we write C# code

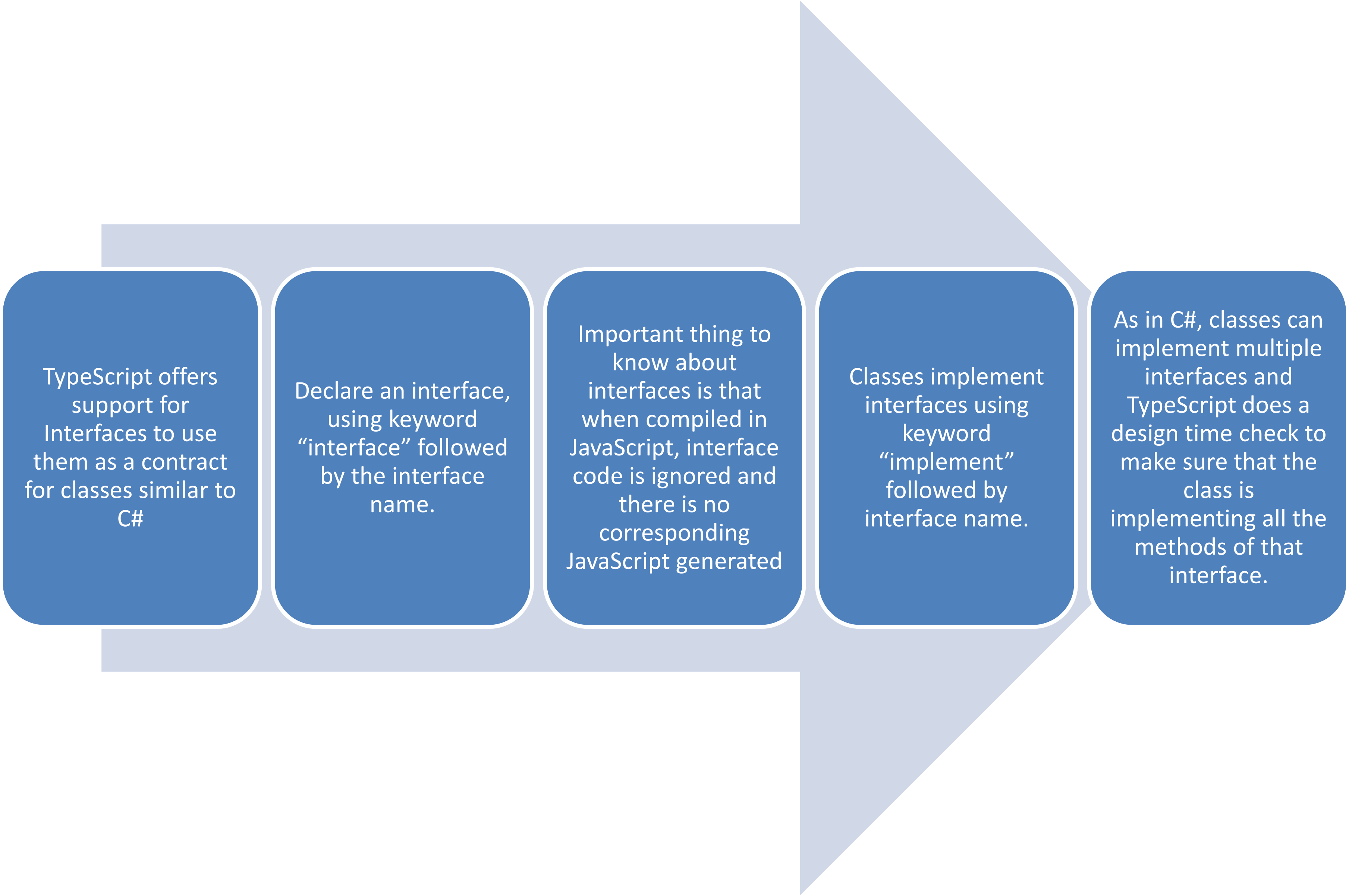
Using inheritance we can extend classes, implement and extend interfaces and write code which very closes recognizes with OOPs

In TypeScript when we extend a base class in child class we use keyword “super” to call the constructor of base class or even the public methods of the base class.

To extend a class in TypeScript we use “extend” keyword after the class name and the followed by the class through which we need to extend

We can also inherit interfaces on other interfaces.

Interfaces



TypeScript offers support for Interfaces to use them as a contract for classes similar to C#

Declare an interface, using keyword “interface” followed by the interface name.

Important thing to know about interfaces is that when compiled in JavaScript, interface code is ignored and there is no corresponding JavaScript generated

Classes implement interfaces using keyword “implement” followed by interface name.

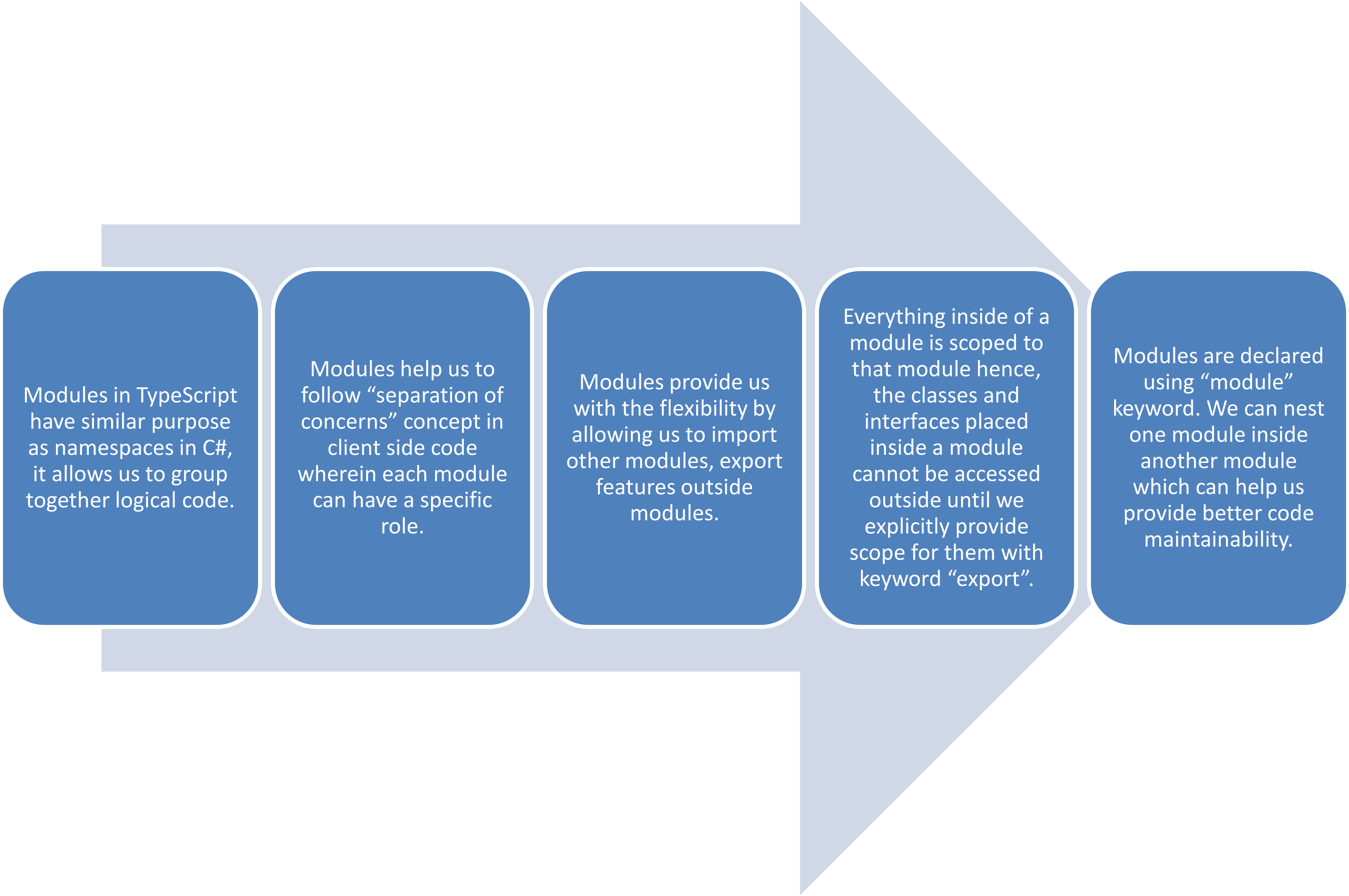
As in C#, classes can implement multiple interfaces and TypeScript does a design time check to make sure that the class is implementing all the methods of that interface.

TypeScript

Continue…

Presenter: Ravi Gulve

Modules



Modules in TypeScript have similar purpose as namespaces in C#, it allows us to group together logical code.

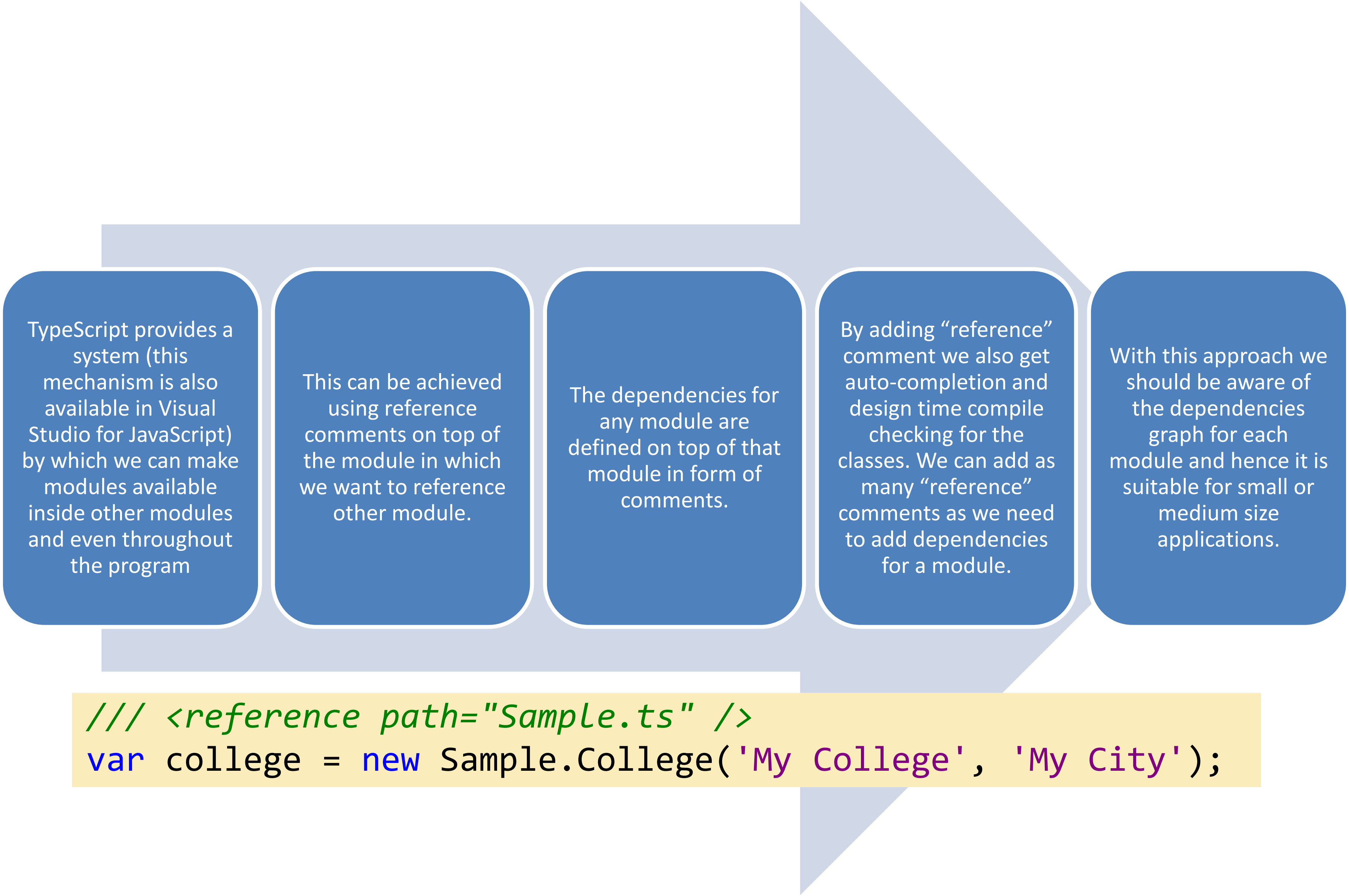
Modules help us to follow “separation of concerns” concept in client side code wherein each module can have a specific role.

Modules provide us with the flexibility by allowing us to import other modules, export features outside modules.

Everything inside of a module is scoped to that module hence, the classes and interfaces placed inside a module cannot be accessed outside until we explicitly provide scope for them with keyword “export”.

Modules are declared using “module” keyword. We can nest one module inside another module which can help us provide better code maintainability.

Modules – referencing internal modules



TypeScript provides a system (this mechanism is also available in Visual Studio for JavaScript) by which we can make modules available inside other modules and even throughout the program

This can be achieved using reference comments on top of the module in which we want to reference other module.

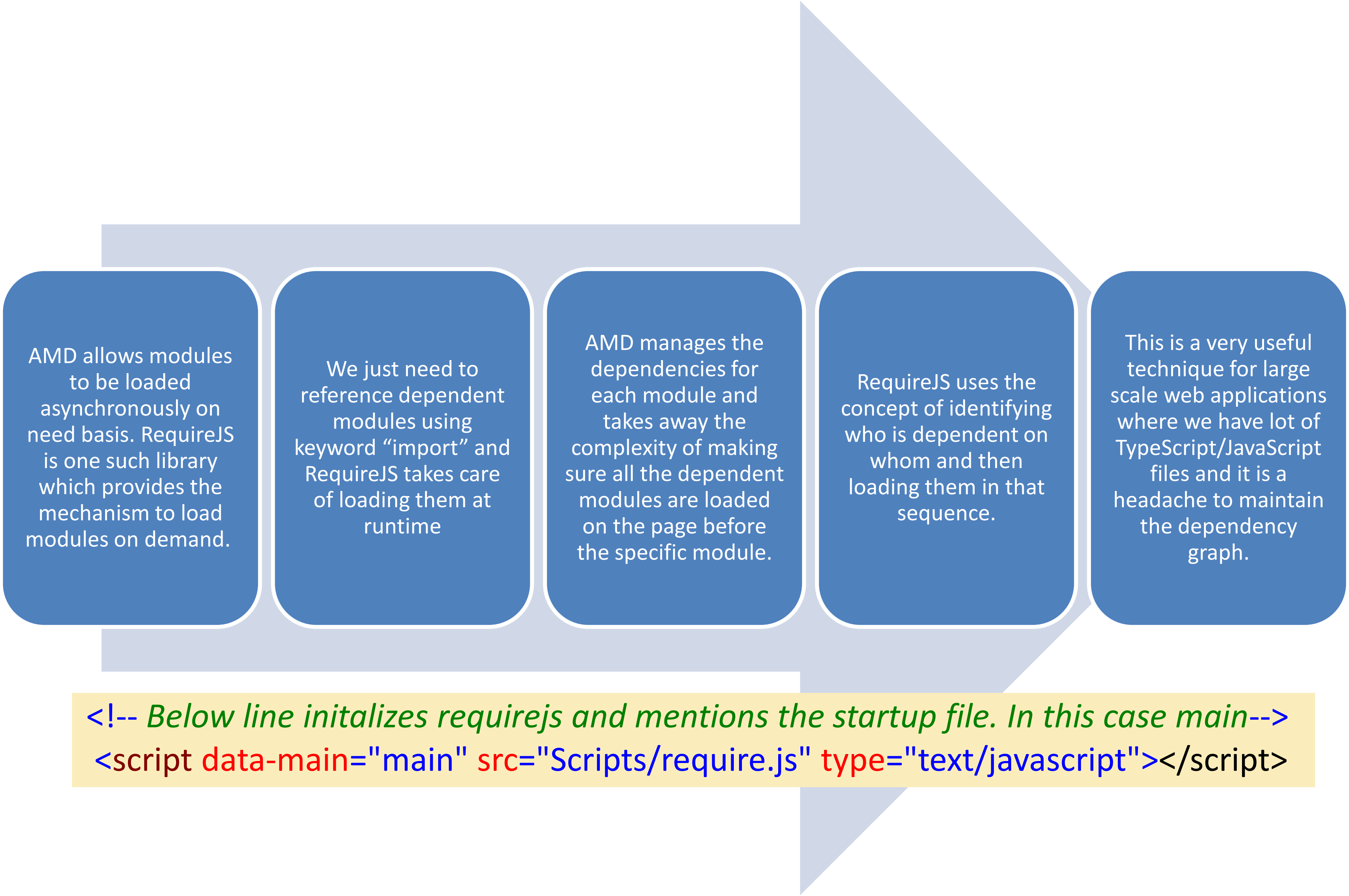
The dependencies for any module are defined on top of that module in form of comments.

By adding “reference” comment we also get auto-completion and design time compile checking for the classes. We can add as many “reference” comments as we need to add dependencies for a module.

With this approach we should be aware of the dependencies graph for each module and hence it is suitable for small or medium size applications.

```
/// <reference path="Sample.ts" />
var college = new Sample.College('My College', 'My City');
```

Modules – Asynchronous module definition(AMD)



AMD allows modules to be loaded asynchronously on need basis. RequireJS is one such library which provides the mechanism to load modules on demand.

We just need to reference dependent modules using keyword “import” and RequireJS takes care of loading them at runtime

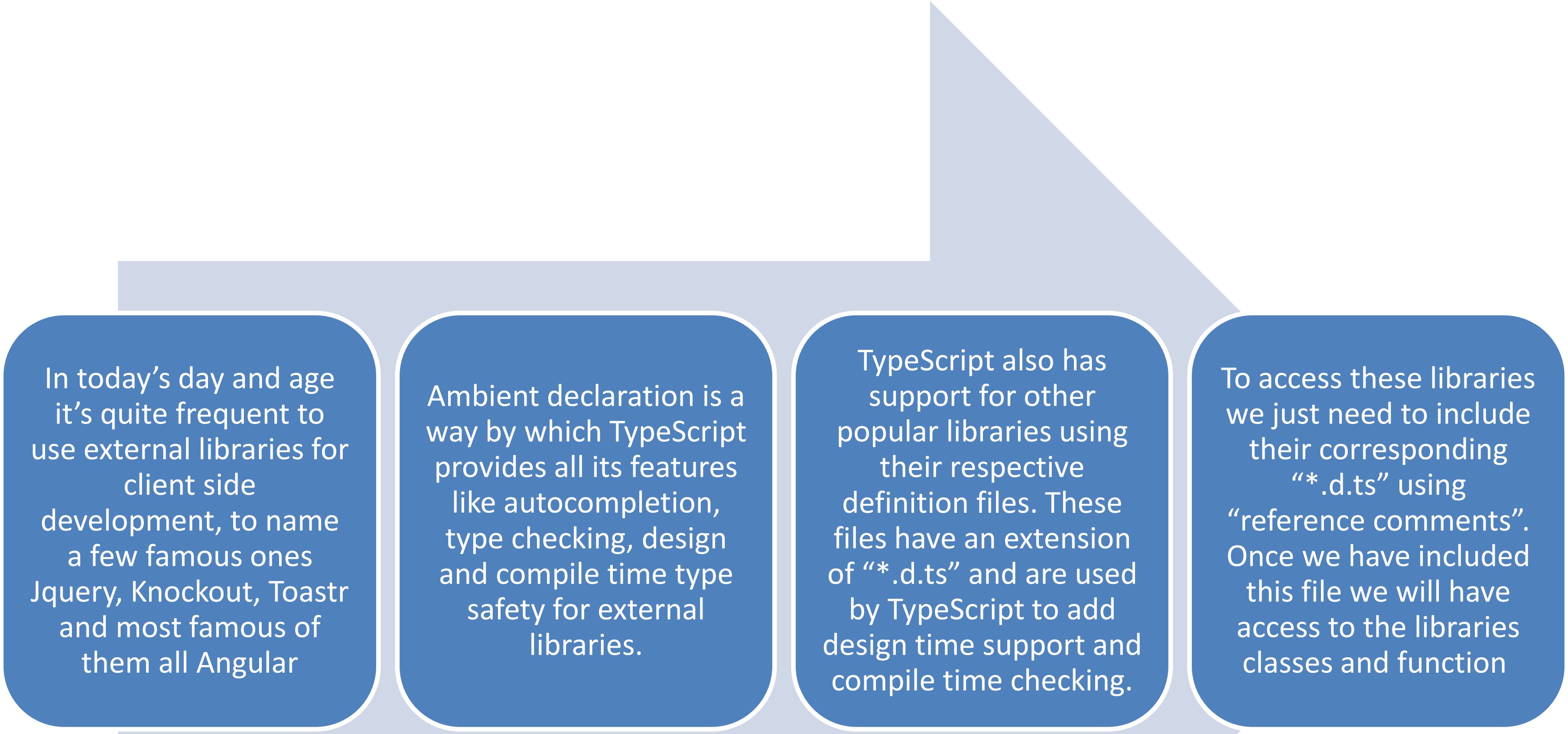
AMD manages the dependencies for each module and takes away the complexity of making sure all the dependent modules are loaded on the page before the specific module.

RequireJS uses the concept of identifying who is dependent on whom and then loading them in that sequence.

This is a very useful technique for large scale web applications where we have lot of TypeScript/JavaScript files and it is a headache to maintain the dependency graph.

```
<!-- Below line initalizes requirejs and mentions the startup file. In this case main-->
<script data-main="main" src="Scripts/require.js" type="text/javascript"></script>
```

Typescript with external libraries : Ambient declarations



In today’s day and age it’s quite frequent to use external libraries for client side development, to name a few famous ones Jquery, Knockout, Toastr and most famous of them all Angular

Ambient declaration is a way by which TypeScript provides all its features like autocompletion, type checking, design and compile time type safety for external libraries.

TypeScript also has support for other popular libraries using their respective definition files. These files have an extension of “*.d.ts” and are used by TypeScript to add design time support and compile time checking.

To access these libraries we just need to include their corresponding “*.d.ts” using “reference comments”. Once we have included this file we will have access to the libraries classes and function

```
/// <reference path="typings/jquery.d.ts" />
document.title = 'Hello TypeScript';
$(document).ready(function() {
var v;
});
```

Type Guards

- Type Guards allow us to narrow down the type of an object within a conditional block.
- Typescript is aware of the usages of *instanceof* and *typeof* operators.
- If we use these in a conditional block, TypeScript will understand the type of the variable to be different within that conditional block.

Thank You !!!