.NET Framework & C# 5.0

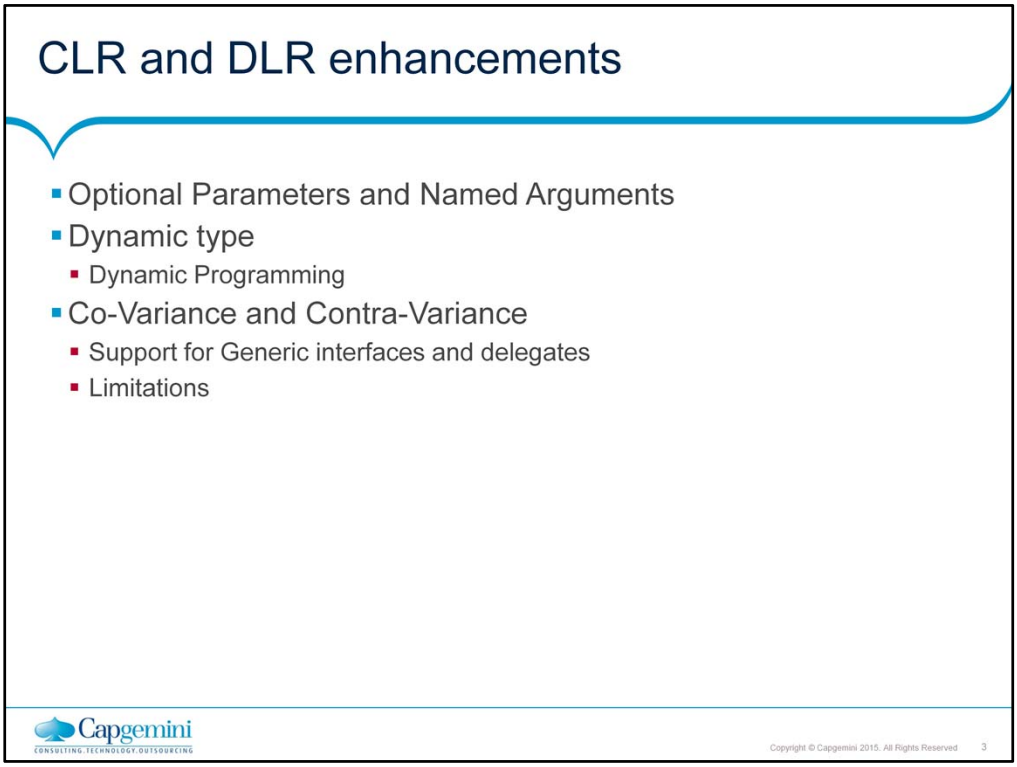
Lesson 11: Parallel and Async programming with C#

Lesson Objectives

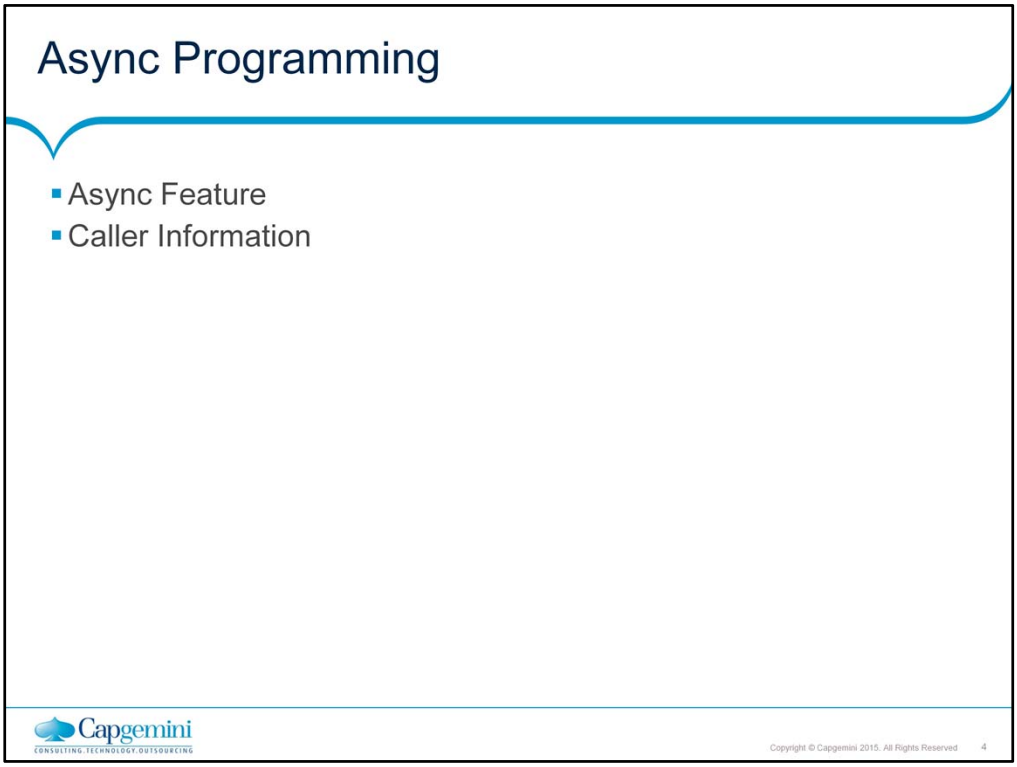
- Dynamic types
- Parallel Programming
- Async Keyword

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

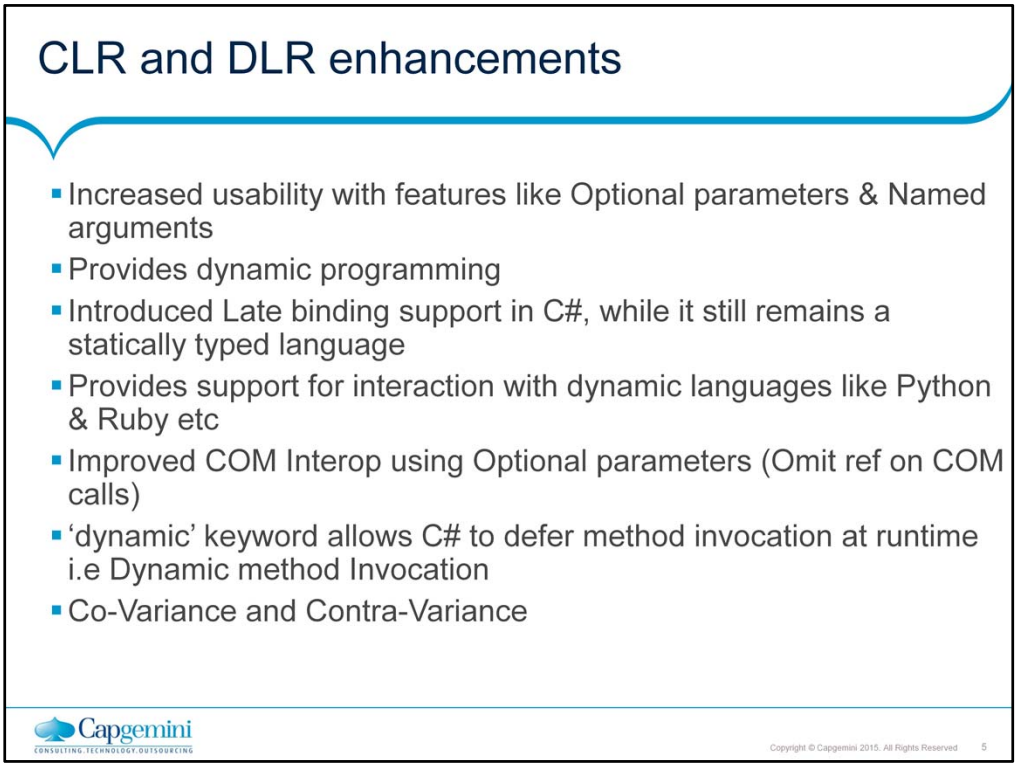
 2



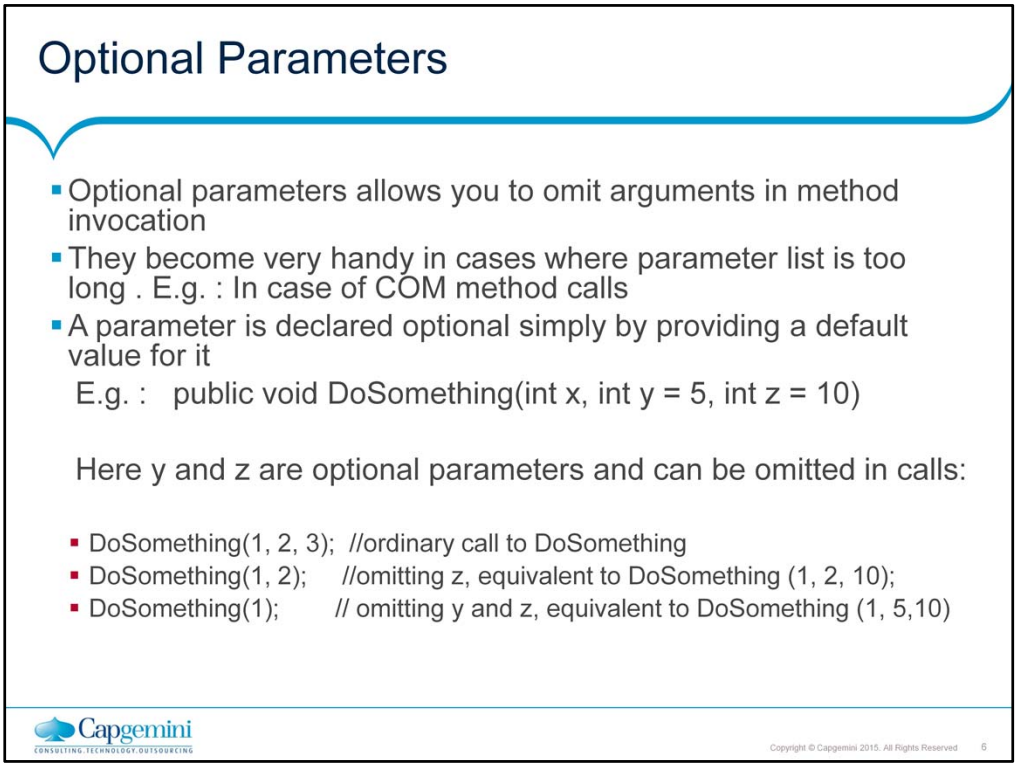
Add the notes here.



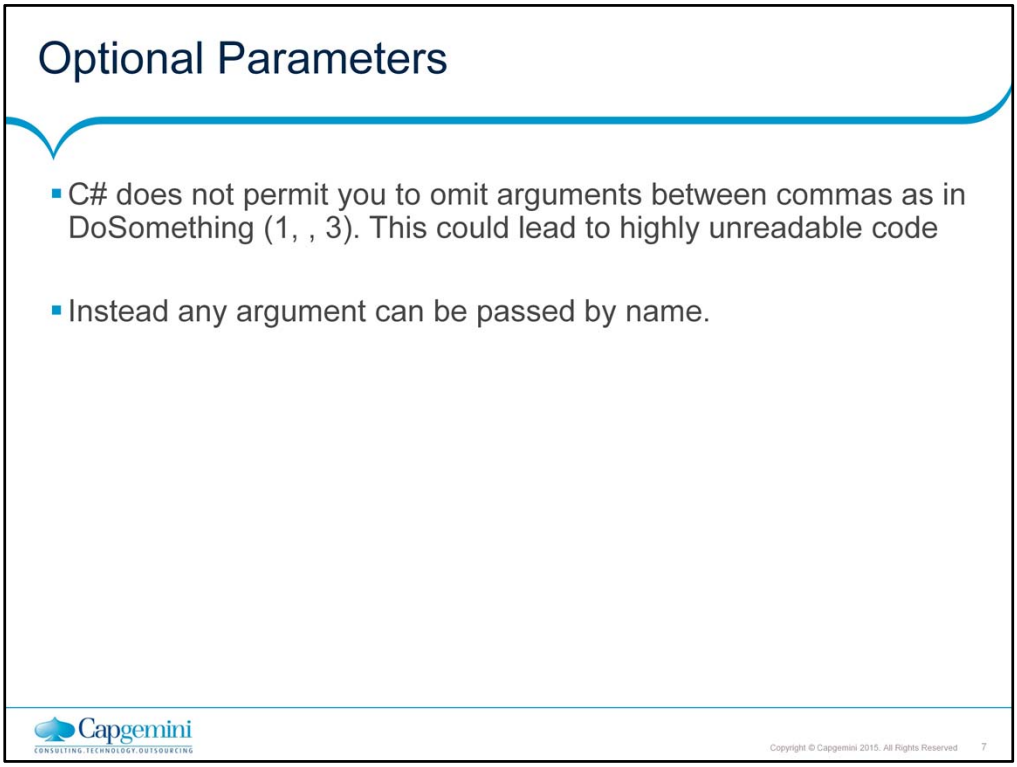
Add the notes here.



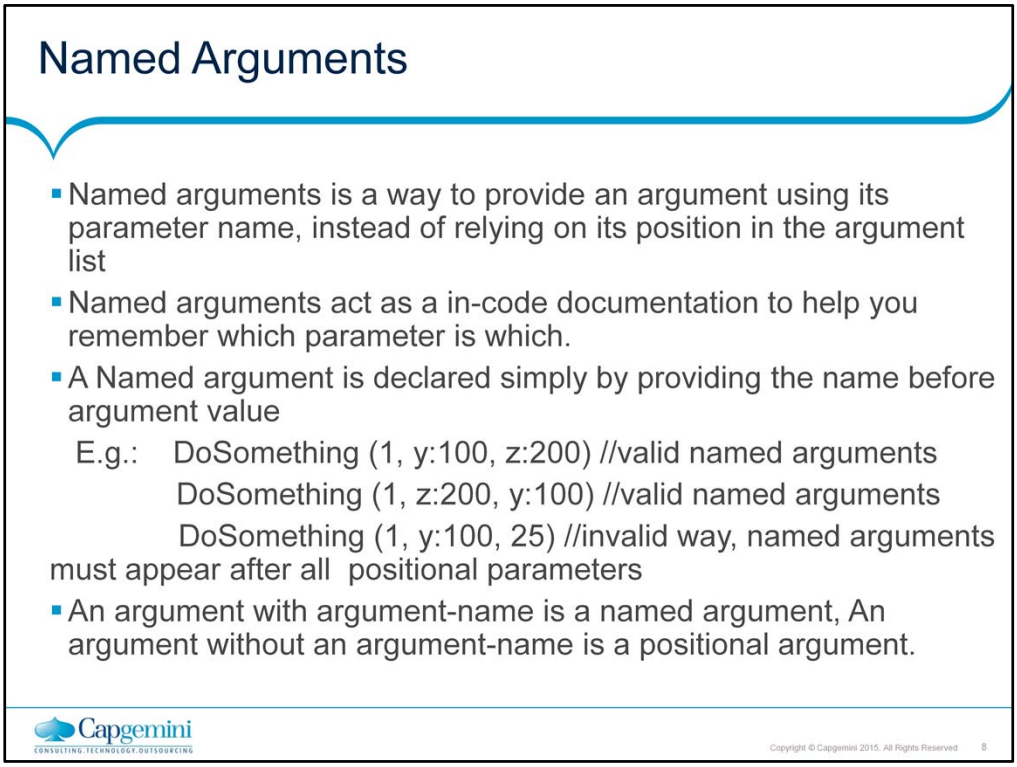
The major theme for C# 4.0 is dynamic programming. Increasingly, objects are "dynamic" in the sense that their structure and behavior is not captured by a static type, or at least not one that the compiler knows about when compiling your program. Some examples include :-
Objects from dynamic programming languages, such as Python or Ruby
COM objects accessed through IDispatch
Ordinary .NET types accessed through reflection
Objects with changing structure, such as HTML DOM objects.
While C# still remains a statically typed language, it aims to vastly improve the interaction with such objects.



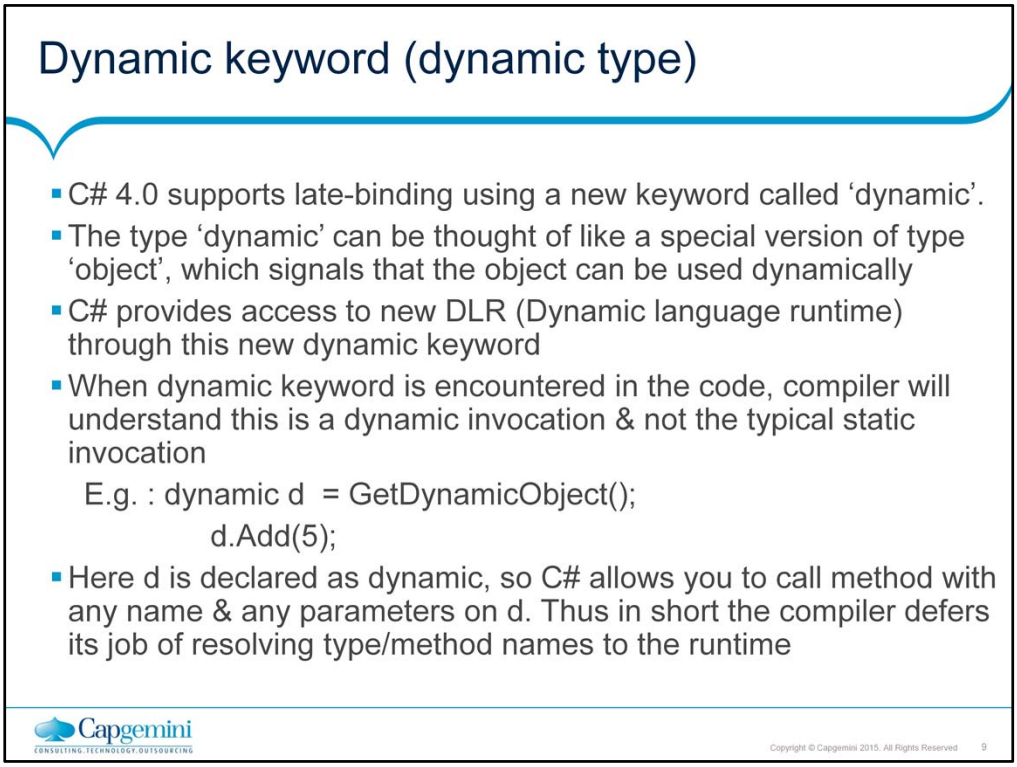
Add the notes here.



Add the notes here.



Add the notes here.



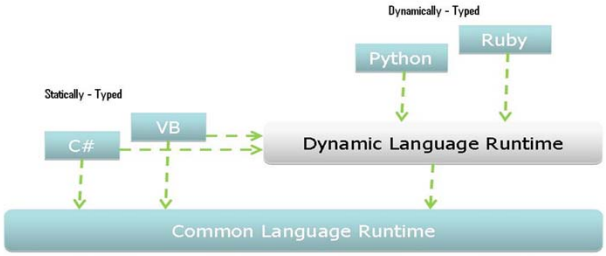
C# now supports late-binding. C# has always been strongly typed & continues to be in version 4.0 as well. But Microsoft thought providing the concept of dynamic binding in C#. It would make life easier for C# developers while they communicate with systems using COM objects or other non .NET based objects.

C# 4.0 introduces a new static type named dynamic. This resolves issues with type binding especially if the receiving object is a non .net based object. Here whatever you do with a dynamic type will take effect only at runtime i.e. it will be resolved at runtime.

```
E.g. : dynamic d  = GetDynamicObject();
                    d.Add(5);
```

C# allows you to call a method with any name and any arguments on d because it is of type dynamic. At runtime the actual object that d refers to will be examined to determine what it means to "call Add with an int" on it. This approach using dynamic type is much similar to using object type, but there are subtle differences between the two.

C# and DLR (dynamic language runtime)



- The infrastructure that supports 'dynamic' operations is called as Dynamic Language Runtime
- This new library introduced with .NET framework 4.0, for each dynamic operation acts as a broker between the language which initiated it and the object it occurs on

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

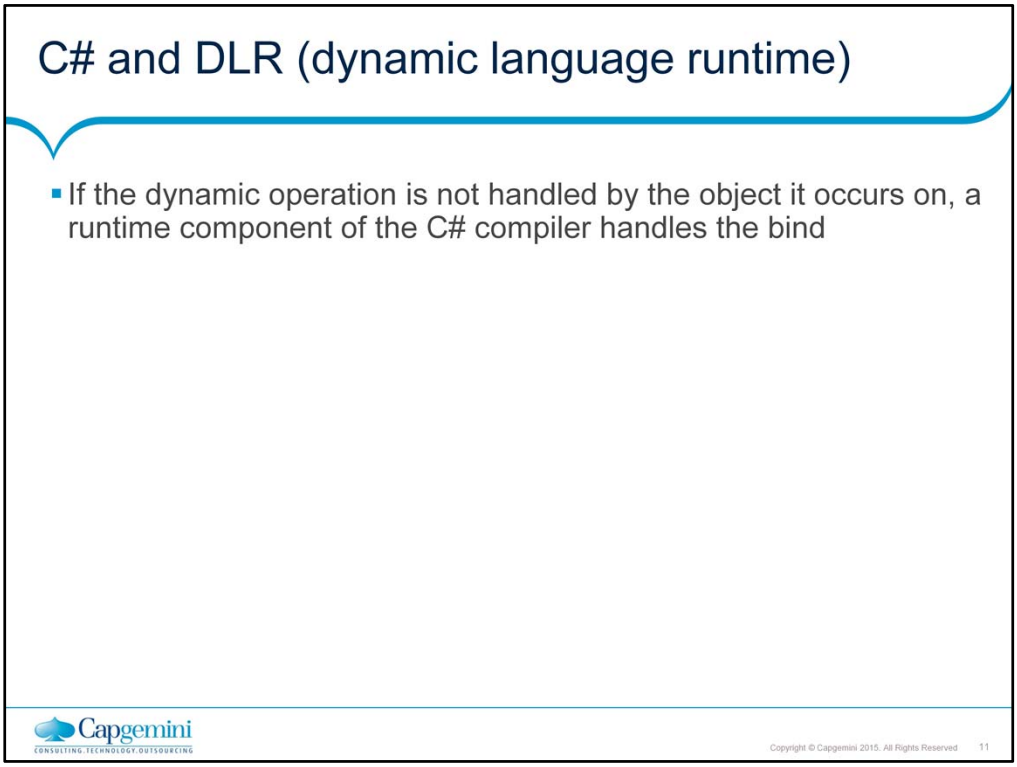
 10

Dynamic binding provides a unified approach to selecting operations dynamically. With dynamic binding developer does not need to worry about whether a given object comes from e.g. COM, IronPython, the HTML DOM or reflection; operations can uniformly be applied to it and the runtime will determine what those operations mean for that particular object.

This affords enormous flexibility, and can greatly simplify the source code, but it does come with a significant drawback: Static typing is not maintained for these operations. A dynamic object is assumed at compile time to support any operation, and only at runtime will an error occur if it was not so.

Dynamic Programming
E.g. - : dynamic d = new MyDynamicObject();
d.Add(5);

By declaring d to be of type dynamic, the code that consumes the MyDynamicObject instance effectively opts out of compile-time checking for the operations d participates in. Use of dynamic means "I don't know what type this is going to be, so I don't know what methods or properties there are right now. Compiler, please let them all through and then figure it out when you really have an object at run time." So the call to Add compiles even though the compiler doesn't know what it means. Then at run time, the object itself is asked what to do with this call to Add.



Dynamic binding provides a unified approach to selecting operations dynamically. With dynamic binding developer does not need to worry about whether a given object comes from e.g. COM, IronPython, the HTML DOM or reflection; operations can uniformly be applied to it and the runtime will determine what those operations mean for that particular object.

This affords enormous flexibility, and can greatly simplify the source code, but it does come with a significant drawback: Static typing is not maintained for these operations. A dynamic object is assumed at compile time to support any operation, and only at runtime will an error occur if it was not so.

Dynamic Programming
E.g. : dynamic d = new MyDynamicObject();
d.Add(5);

By declaring d to be of type dynamic, the code that consumes the MyDynamicObject instance effectively opts out of compile-time checking for the operations d participates in. Use of dynamic means “I don’t know what type this is going to be, so I don’t know what methods or properties there are right now. Compiler, please let them all through and then figure it out when you really have an object at run time.” So the call to Add compiles even though the compiler doesn’t know what it means. Then at run time, the object itself is asked what to do with this call to Add.

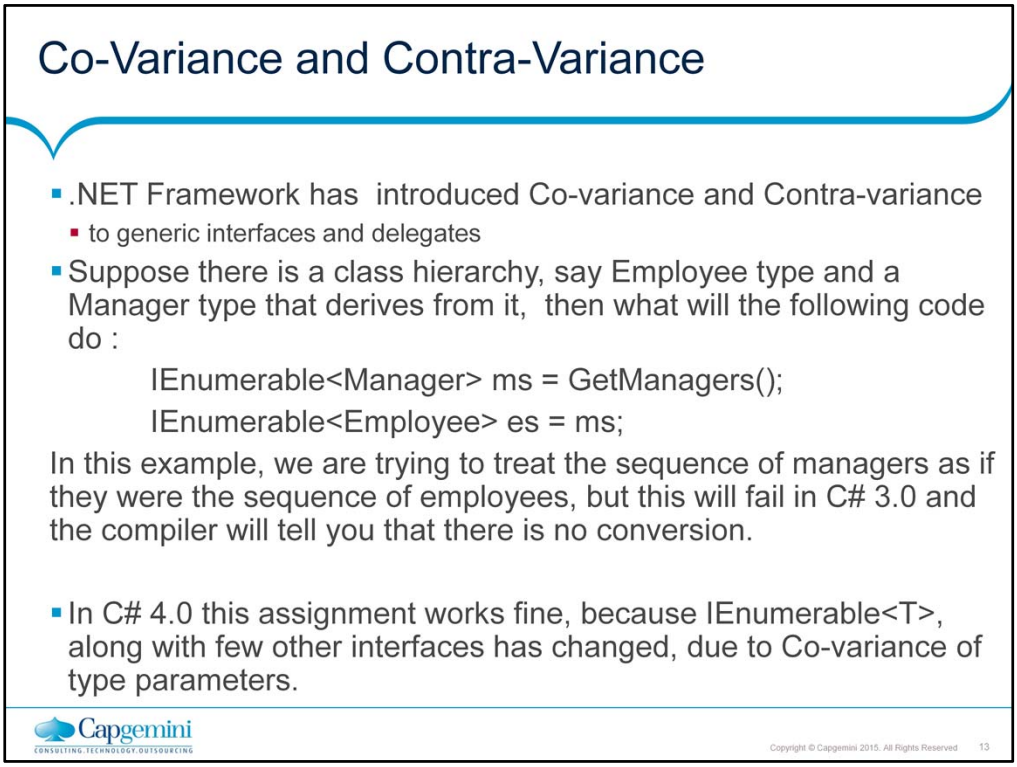
Demo

- Optional Parameters and Named arguments
- Dynamic Programming in C# 5.0

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

Add the notes here.



Covariance :
In .NET 4.0 the IEnumerable<T> and IEnumerator<T>interfaces will be declared in the following way:

```
public interface IEnumerable<out T> : IEnumerable
{
        IEnumerator<T> GetEnumerator();
}
public interface IEnumerator<out T> : IEnumerator
{
        bool MoveNext();
        T Current { get; }
}
```

The “out” in these declarations signifies that the T can only occur in output position in the interface – the compiler will complain otherwise. In return for this restriction, the interface becomes “covariant” in T, which means that IEnumerable<A> is implicitly reference convertible to IEnumerable<B> if A has an implicit reference conversion to B. As a result, any sequence of strings is also a sequence of objects.

Co-Variance and Contra-Variance

- Covariance means think “out”

  E.g. :

```
public interface IEnumerable<out T> : IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
```

- Here the ‘out’ keyword, actually modifies the definition of type parameter, T. Compiler sees this and marks type T as covariant
- The out keyword signifies that the generic type parameter, T can appear only in output positions like, as method return value and read-only properties
- Here the interface becomes covariant in ‘T’.
- It means that IEnumerable<A> is implicitly reference convertible to IEnumerable<B>, if A has an implicit reference conversion to B,

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

Co-Variance and Contra-Variance

- In case of Employee and Manager classes, since there is a relationship between these classes, there is an implicit reference conversion from Manager to Employee :
  - Manager -> Employee

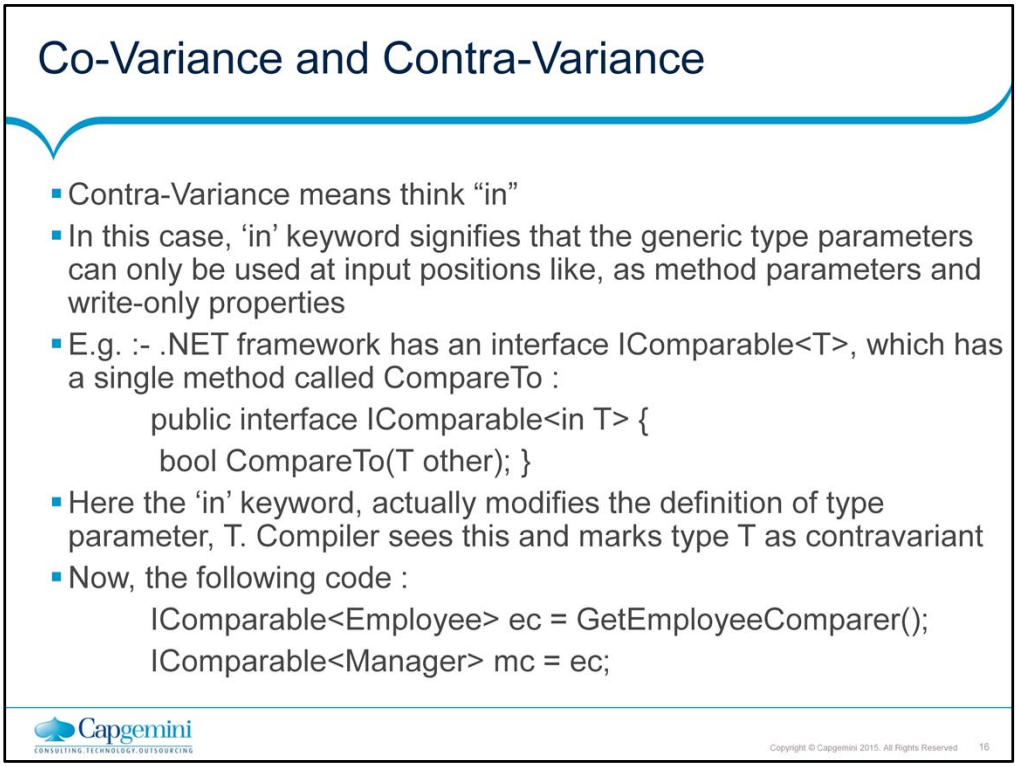
Also because now type T in IEnumerable is out i.e. IEnumerable<out T>, there is also an implicit reference conversion from IEnumerable<Manager> to IEnumerable<Employee> :

- IEnumerable<Manager> -> IEnumerable<Employee>

- Here these new types convert the same way as the old ones.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15



Contravariance :
Type parameters can also have an “in” modifier, restricting them to occur only in input positions. An example is IComparer<T>:

```
public interface IComparer<in T>
{
        public int Compare(T left, T right);
}
```

The result is that an IComparer<object> can in fact be considered an IComparer<string>. This may be surprising at first, but in fact makes perfect sense: If a comparer can compare any two objects, it can certainly also compare two strings. The interface is said to be “contravariant”.

Co-Variance and Contra-Variance

- Because a manager is an employee, putting manager in works due to contra-variance.
- In case of Employee and Manager classes, since there is a relationship between these classes, there is an implicit reference conversion from Manager to Employee :

  Manager -> Employee

Also because type T in IComparable is in i.e. IComparable<in T>, there is also an reference conversion from IComparable<Employee> to IComparable<Manager> :

- IComparable<Manager> <- IComparable<Employee>

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

Limitations

- Covariant and contravariant type parameters can be applied only on generic interfaces and delegates.
- Covariance and contravariance only applies when there is a reference (or identity) conversion between type arguments
- For instance, an IEnumerable<int> is not an IEnumerable<object> because the conversion from int to object is a boxing conversion, not a reference conversion

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

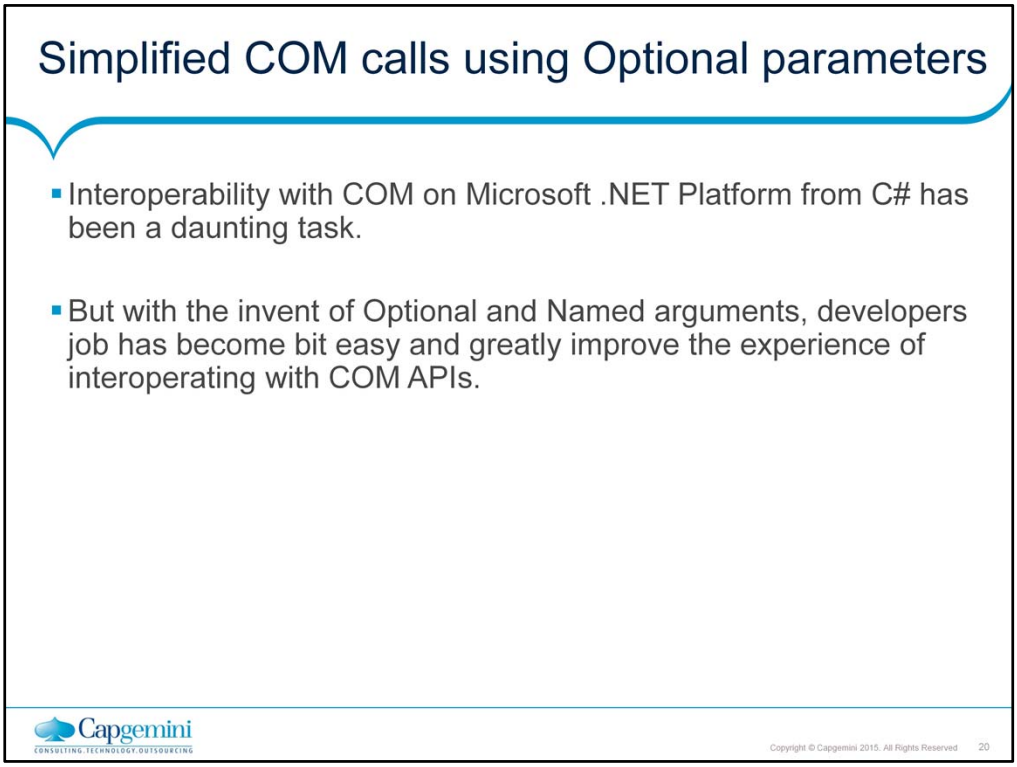
Demo

- Covariance and Contravariance

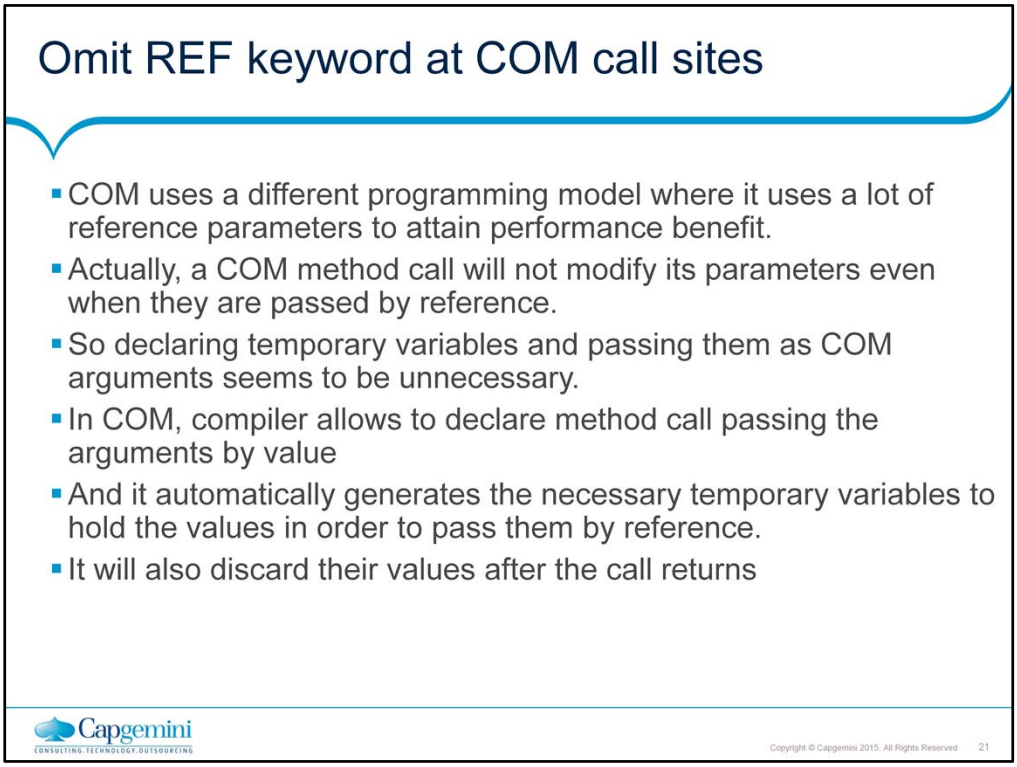
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

Add the notes here.



Simplified COM Calls with Optional Parameters and Named Arguments
Interoperability with COM on Microsoft .Net Platform from C# has been a daunting task. But with the invent of Optional and Named arguments, developers job has become bit easy and greatly improve the experience of interoperating with COM APIs.

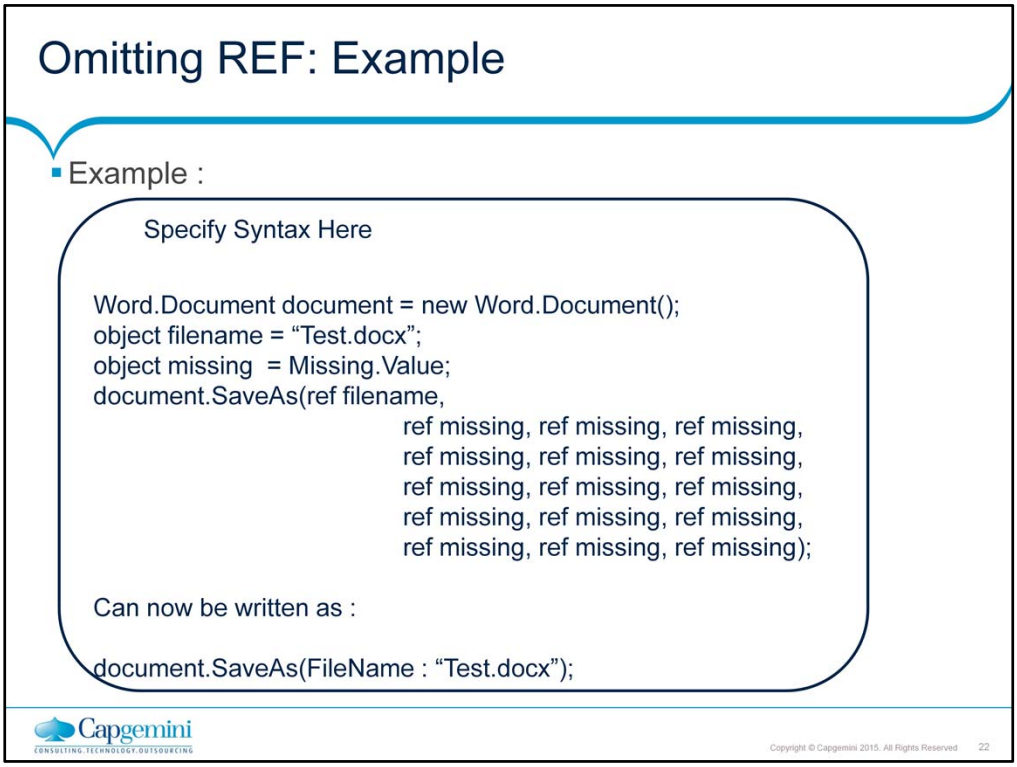


Omitting ref
Since COM uses a different programming model, it usually uses a lot of reference parameters because of a perceived performance benefit. Actually in common case, a COM method call will not modify its parameters even when passed by reference. In this case, it seems unnecessary for a caller to declare temporary variables for all these arguments and pass them by reference. Specifically for COM methods, the compiler allows to declare the method call passing the arguments by value and will automatically generate the necessary temporary variables to hold the values in order to pass them by reference and will discard their values after the call returns.
For e.g.:

```
object filename = “Test.docx”;
object missing  = Missing.Value;
document.SaveAs(ref.filename,
 ref missing, ref missing, ref missing,
                ref missing, ref missing, ref missing,
                                ref missing, ref missing, ref missing,
                                                ref missing, ref
missing, ref missing,
ref missing, ref missing, ref missing);
```

Can now be written like this :

```
                document.SaveAs(“Test.docx”,

        Missing.Value, Missing.Value, Missing.Value,

        Missing.Value, Missing.Value, Missing.Value,

        Missing.Value, Missing.Value, Missing.Value,

        Missing.Value, Missing.Value, Missing.Value,

        Missing.Value, Missing.Value, Missing.Value);
```



Add the notes here.

Demo

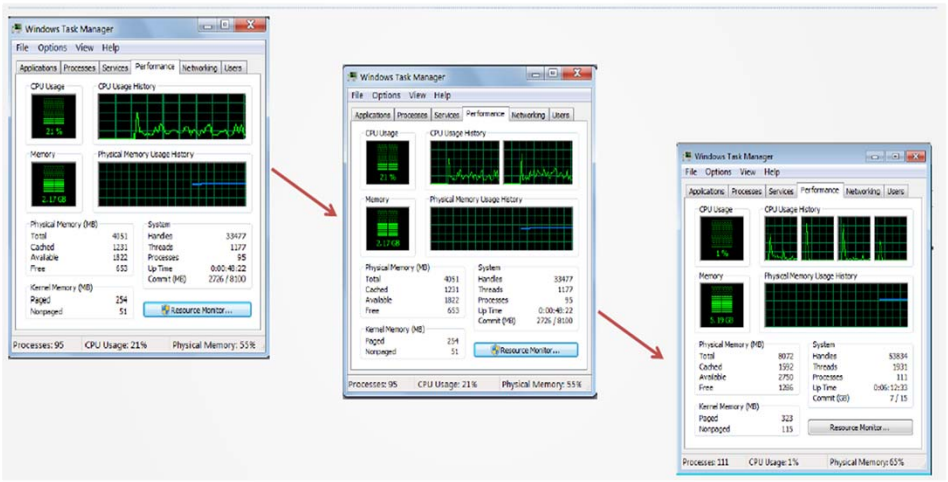
- Simplified COM Calls using Optional Parameters
- Omitting REF with COM calls

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23

Add the notes here.

Evolution of Processors: The Multi Core Shift



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

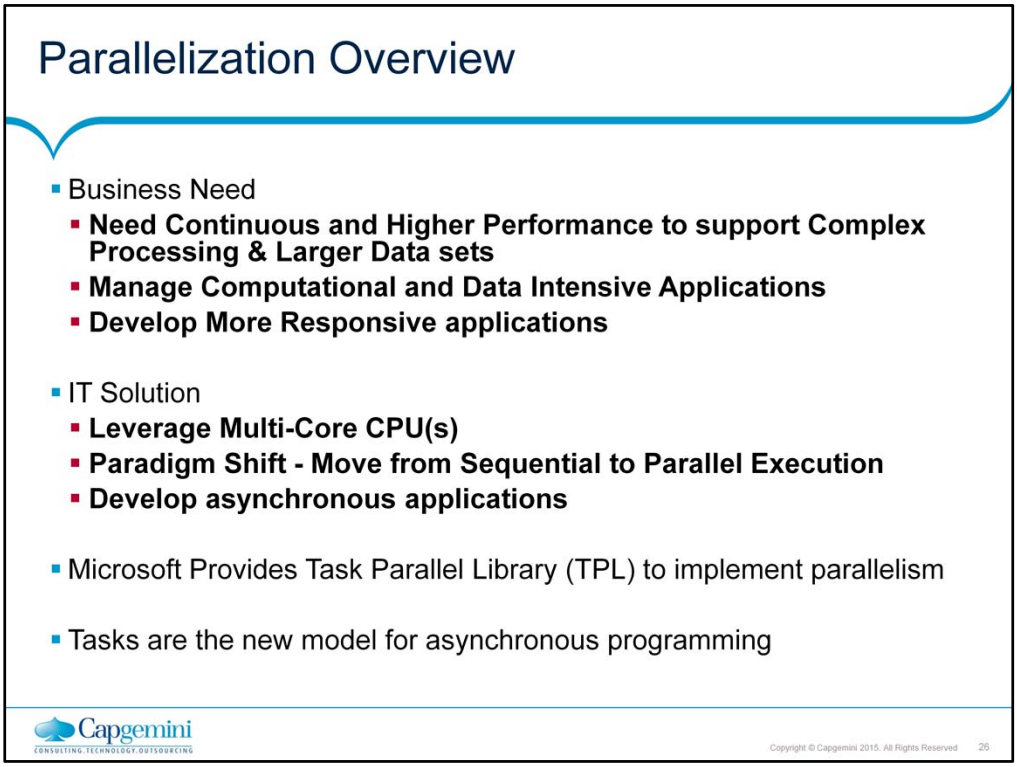
 24

The Multi Core Shift

- New model for increasing hardware processing power
- Focus shifted from increasing PC CPU Clock speed
- Overall processing power increased due to additional CPUs, or “cores” on a single chip
- Most desktop and laptop systems now ship with multi-core microprocessors
- Developers should leverage these multi cores in the programs

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25



In the last 20-30 years we have seen a tremendous growth in processing power where the average speed of the processor has increased year by year. But over the last couple of years we are observing a shift in this trend. Instead of increase in processor speed the number of processors on a single chip are increasing. We are very fast moving from dual core to quad core to eight core processors. So we are gradually shifting towards a multicore processor architecture. But most of our present applications are still of sequential nature and not capable of leveraging the performance benefits offered by multicore processors. This is because parallel/concurrent programming involves threads,locks,mutexes etc. which are difficult to program and also hard to maintain because of lack of tools for profiling and debugging those applications.

Asynchronous Defined

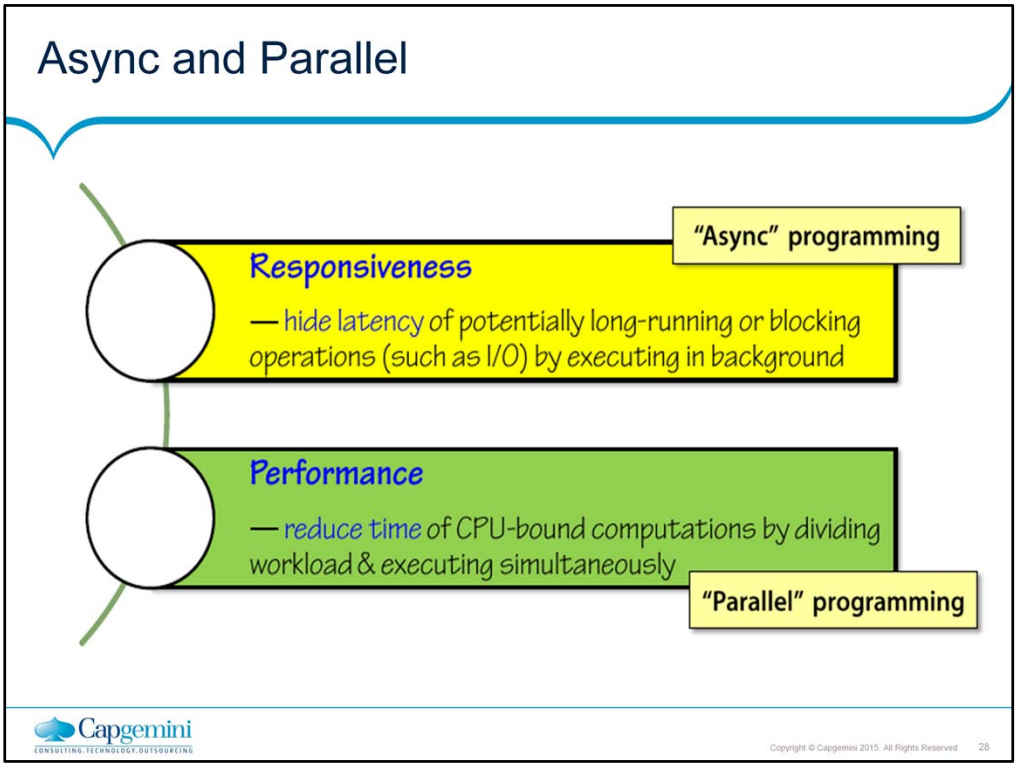
- Concurrent
  - **Several things happening at once**
- Multithreaded
  - **Multiple execution contexts**
- Parallel
  - **Multiple simultaneous Computations**
- Asynchronous
  - **Not having to wait**

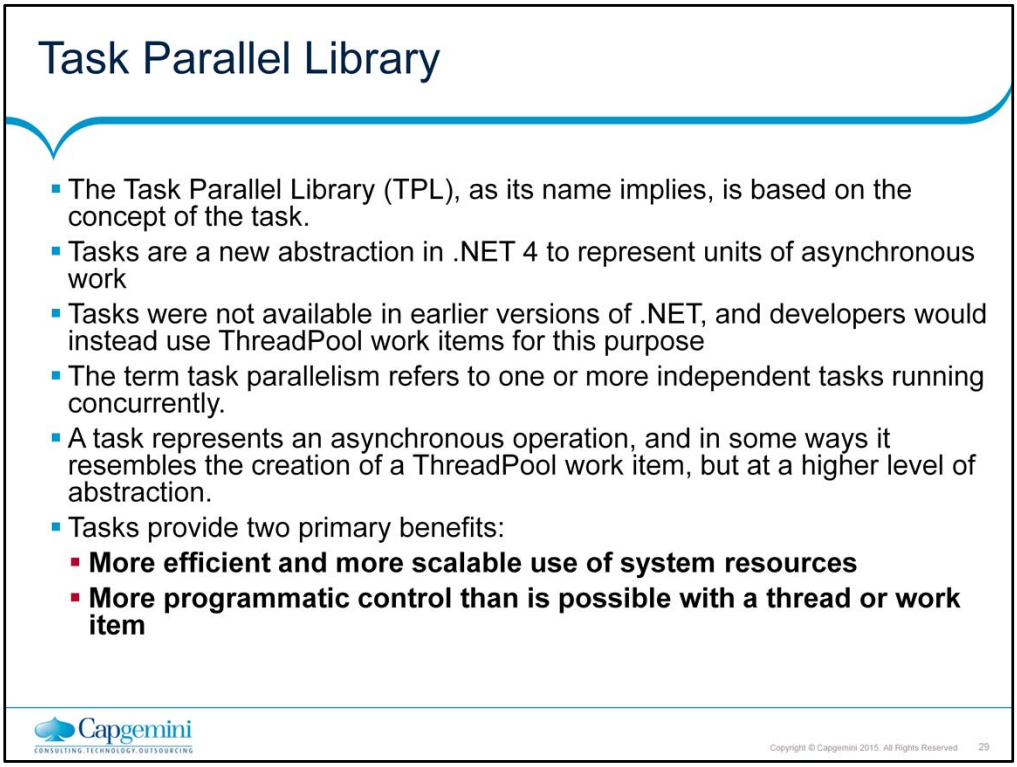
- Parallel => divide workload so that you can finish sooner (Performance)

- Async: => start an operation and then return to UI (responsiveness)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 27


Async and Parallel
"Async" programming
Responsiveness
— hide latency of potentially long-running or blocking operations (such as I/O) by executing in background
Performance
— reduce time of CPU-bound computations by dividing workload & executing simultaneously
"Parallel" programming
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
28



More efficient and more scalable use of system resources.
Behind the scenes, tasks are queued to the ThreadPool, which has been enhanced with algorithms (like hill-climbing) that determine and adjust to the number of threads that maximizes throughput. This makes tasks relatively lightweight, and you can create many of them to enable fine-grained parallelism. To complement this, widely-known work-stealing algorithms are employed to provide load-balancing.
More programmatic control than is possible with a thread or work item.
Tasks and the framework built around them provide a rich set of APIs that support waiting, cancellation, continuations, robust exception handling, detailed status, custom scheduling, and more.

Task Parallel Library

- The Task Parallel Library (TPL) is a set of APIs in the System.Threading and System.Threading.Tasks namespaces in the .NET Framework version 4.
- The TPL scales the degree of concurrency dynamically to most efficiently use all the processors that are available.
- In addition, the TPL handles the partitioning of the work, the scheduling of threads on the ThreadPool, cancellation support, state management, and other low-level details.
- By using TPL, you can maximize the performance of your code while focusing on the work that your program is designed to accomplish

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

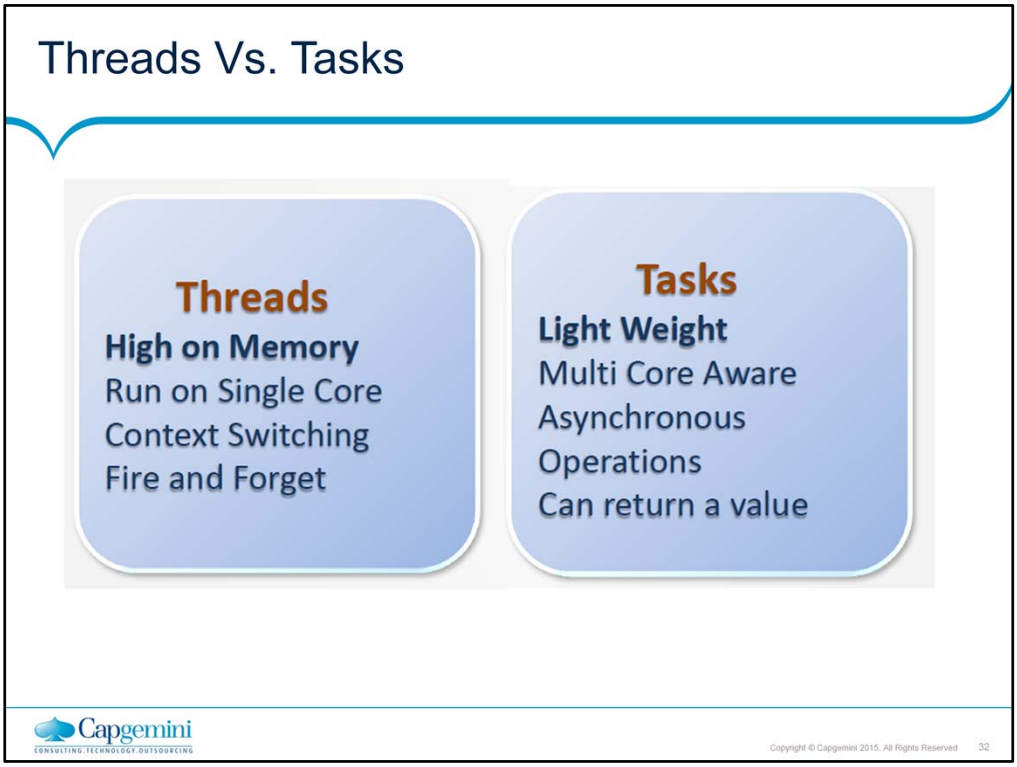
 30

Task Parallel Library

- Parallel Extensions in .NET 4.0 provides a set of libraries and tools which
  - **Makes concurrent programming easier where developers need not care much details of threads and locks**
  - **Makes profiling and debugging concurrent applications easier**
  - **Enables the applications to scale up without any code as number of cores increases**
  - **Allows existing applications to be easily parallelized**

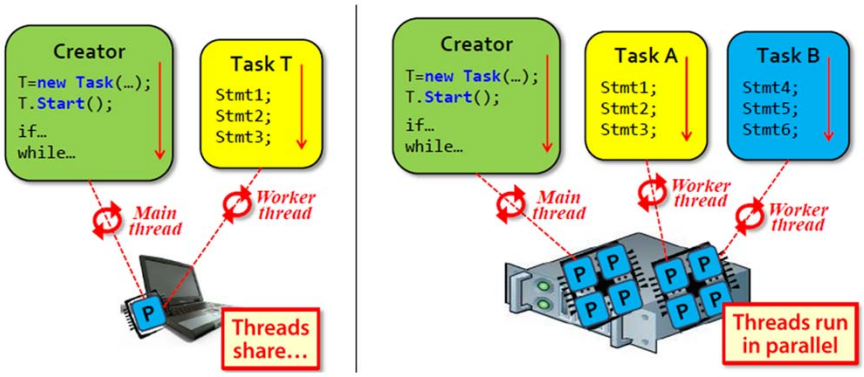
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31


Threads Vs. Tasks
Threads
High on Memory
Run on Single Core
Context Switching
Fire and Forget
Tasks
Light Weight
Multi Core Aware
Asynchronous
Operations
Can return a value
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
32

Threads Vs. Tasks

- Code based tasks are executed by a thread on some processor
- Thread is dedicated to task until task completes



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33

Parallel Extensions in .NET

- Data Parallelism
  - This refers to dividing the data across multiple processors for parallel execution. e.g. we are processing an array of 1000 elements we can distribute the data between two processors say 500 each.
  - Using Parallel Foreach, For, Invoke, PLINQ
- Task Parallelism
  - This breaks down the program into multiple tasks which can be parallelized and are executed on different processors.
  - This is supported by Task Parallel Library (TPL) in .NET and the Task class
  - Implement the “Task Asynchronous Pattern” TAP

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34

Data Parallelism

```
// Sequential version
foreach (var item in sourceCollection)
{
    Process(item);
}

// Parallel equivalent
Parallel.ForEach(sourceCollection, item => Process(item));
```

- When a parallel loop runs, the TPL partitions the data source so that the loop can operate on multiple parts concurrently.
- Behind the scenes, the Task Scheduler partitions the task based on system resources and workload.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 35

Task Parallelism

- The term task parallelism refers to one or more independent tasks running concurrently.

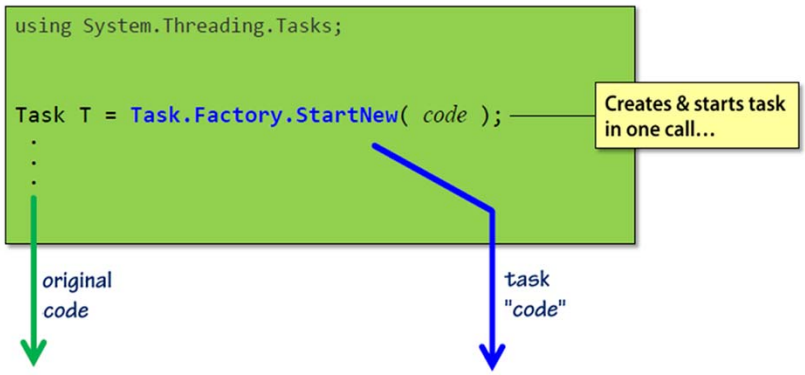
```
// Create a task and supply a user delegate by using a lambda expression.
var taskA = new Task(() => Console.WriteLine("Hello from taskA."));

// Start the task.
taskA.Start();

// Output a message from the joining thread.
Console.WriteLine("Hello from the calling thread.");


/* Output:
 * Hello from the joining thread.
 * Hello from taskA.
 */
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 36

Creating code Tasks – preferred approach

- More efficient way..



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 37

Lambda Expressions

- TPL accepts lambda expressions as parameters
  - Making it easy to create tasks..

```
// original code:
statement1;
statement2;
statement3;
```

```
// parallel code:
Task.Factory.StartNew( () =>
  {
    statement1;
    statement2;
    statement3;
  }
);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Parallel LINQ - PLINQ

- PLINQ provides a parallel implementation of LINQ and supports all the standard query operators.
- This is achieved by the System.Linq.ParallelEnumerable class which provides a set of extension methods on IEnumerable interface.
- Developers can opt-in for parallelism by invoking the AsParallel method of ParallelEnumerable class on the data source.
- Using PLINQ we can combine sequential and parallel queries

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 39

Task Based Asynchronous Model in .NET 4.5

- We can avoid performance bottlenecks and enhance the overall responsiveness of your application by using asynchronous programming.
- However, traditional techniques for writing asynchronous applications can be complicated, making them difficult to write, debug, and maintain.
- C# 5.0 introduces a simplified approach, async programming, that leverages asynchronous support in the .NET Framework 4.5 and the Windows Runtime

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 40

Async and Await - .NET 4.5

- Async programming is recommended in the following scenarios:
  - To keep your application responsive while waiting for something that is likely to take a long time to complete
  - To make your application as scalable as possible (particularly important for services and ASP.NET that have limited threads/resources to service requests)
- However, not all problems are suited to be run asynchronously

- Consider using async functionality in the following situations
  - You are waiting for a response from a remote service
  - The task you want to perform is computationally expensive and doesn't complete very quickly

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 41

Async

- async: This modifier indicates the method is now asynchronous.
- The benefit of using this keyword is that it provides a simpler way to perform potentially long-running operations without blocking the callers thread.
- The caller of this async method can resume its work without waiting for this asynchronous method to finish its job.
- This reduces developer's efforts for writing an additional code.
- The method with async modifier has at least one await.
- This method now runs synchronously until the first await expression written in it

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 42

Await

- await: Typically, when a developer writes some code to perform an asynchronous operations using threads, then he/she has to explicitly write necessary code to perform wait operations to complete a task.
- In asynchronous methods, every operation which is getting performed is called a Task.
- The ‘await’ keyword is applied on the task in an asynchronous method and suspends the execution of the method, until the awaited task is not completed.
- During this time, the control is returned back to the caller of this asynchronous method

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 43

Await

- Most important behavior of the await keyword is it does not block the thread on which it is executing.
- What it does it that it signals to the compiler to continue with other async methods, till the task is in an await state.
- Once the task is completed, it resumes back where it left off from.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 44

.NET 4.5 async and await

- Async
  - Enables use of asynchronous features
  - Applied to methods
- Await

```
void Work()
{
   Task<string> ts = Get();
   ts.ContinueWith(t =>
   {
      string result = t.Result;
      Console.WriteLine(result);
   });
}
```

```
async void Work()
{
   Task<string> ts = Get();
   string result = await ts;
   Console.WriteLine(result);
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 45

Async Rules

- The compiler will give you a warning for methods that contain async modifier but no await operator
- A method that isn't using the await operator will be run synchronously
- Microsoft suggests a convention of post-fixing an Async method with the words Async e.g. DoSomethingAsync
- Async methods can return any of the following:
  - **Void**
  - **Task**
  - **Task<T>**
- A number of BCL methods now have an Async version for you to use with the await keyword

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

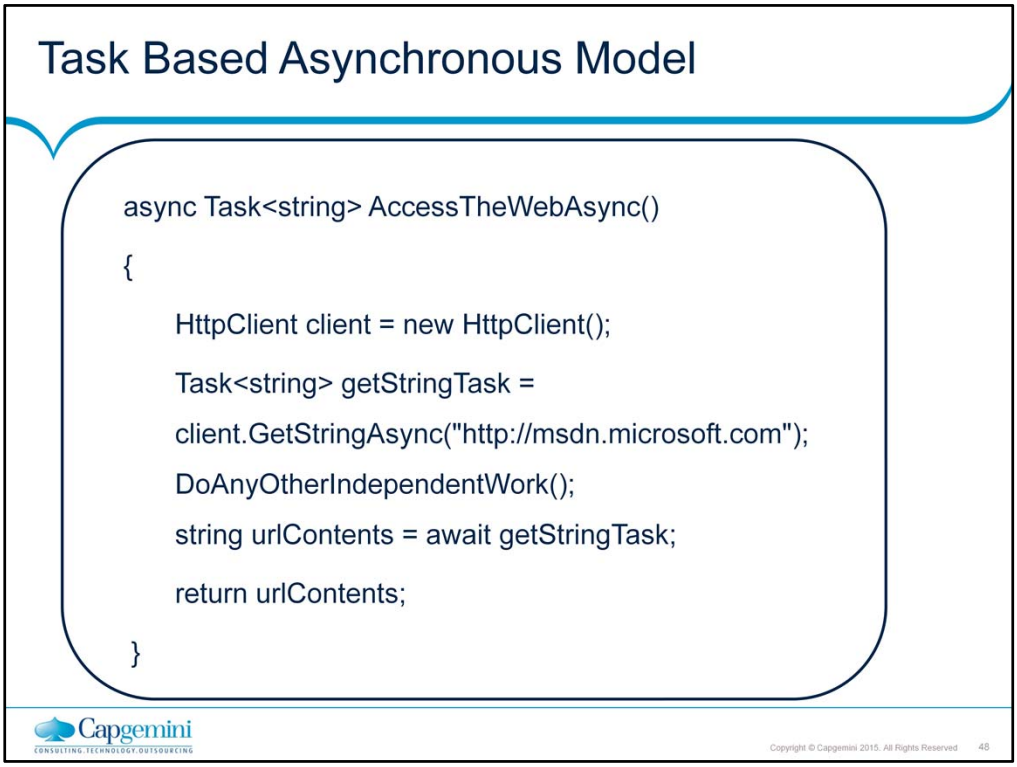
 46

The BCL Support for Async in .NET 4.5

```
public class Stream
{
  ...
  public Task WriteAsync(byte[] buffer, int offset, int count)
  public void Write(byte[] buffer, int offset, int count)
  ...
  public int Read(byte[] buffer, int offset, int count)
  public Task<int> ReadAsync(byte[] buffer, int offset, int count)
  ...

}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

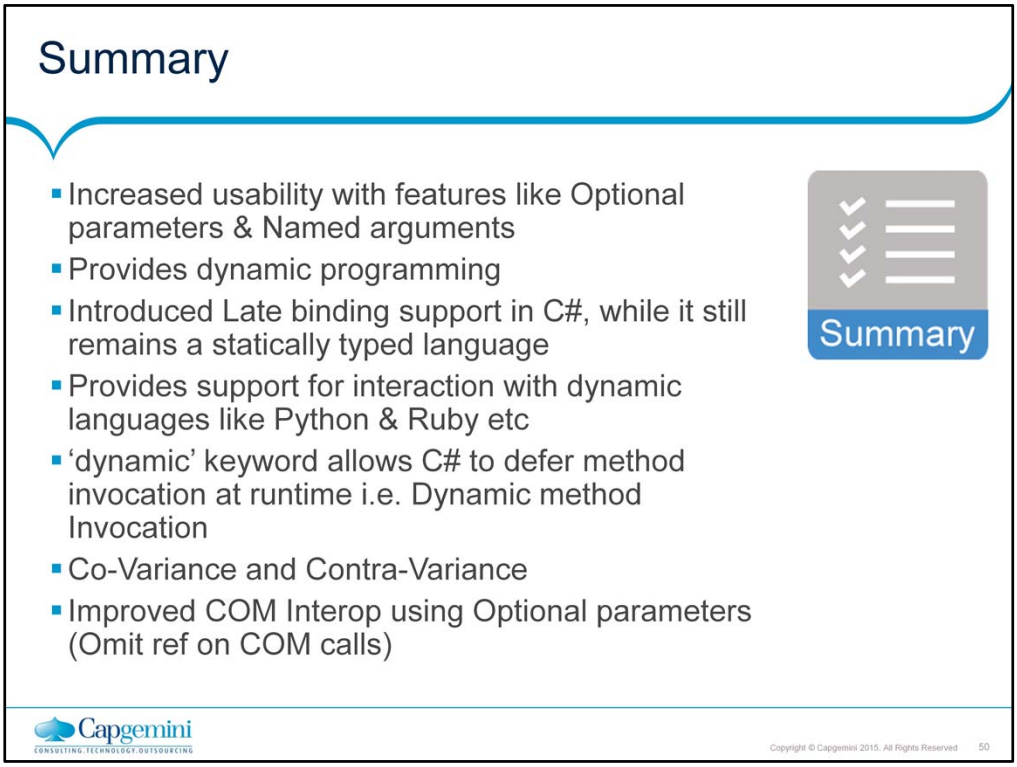
 47


Task Based Asynchronous Model
async Task<string> AccessTheWebAsync()
{
HttpClient client = new HttpClient();
Task<string> getStringTask =
client.GetStringAsync("http://msdn.microsoft.com");
DoAnyOtherIndependentWork();
string urlContents = await getStringTask;
return urlContents;
}
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved. 48

Demo

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 49



Add the notes here.

Review Question

- Question 1: Which feature of C# automatically generates temporary variables to pass them as reference to COM methods?
- Linking of PIA
- Omit REF
- Indexed Properties
- Question 2: C# 4.0 Compiler offers to embed definitions inside PIA in your application assembly
- True/False
- Question 3: _____ offers No-PIA option so that entire PIA need not be deployed along with your application setup.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 51

Add the notes here.