

# The Future of Parallel Programming in the .NET Framework

Igor Ostrovsky  
Software Engineer  
Microsoft Corporation

# DISCLAIMER

- This is a talk about the {near} future...
  - All content is subject to change.
  - The technology being discussed...
    - ...is mostly available in CTP form now.
    - ...may never actually ship (but we're doing the best we can to make it).

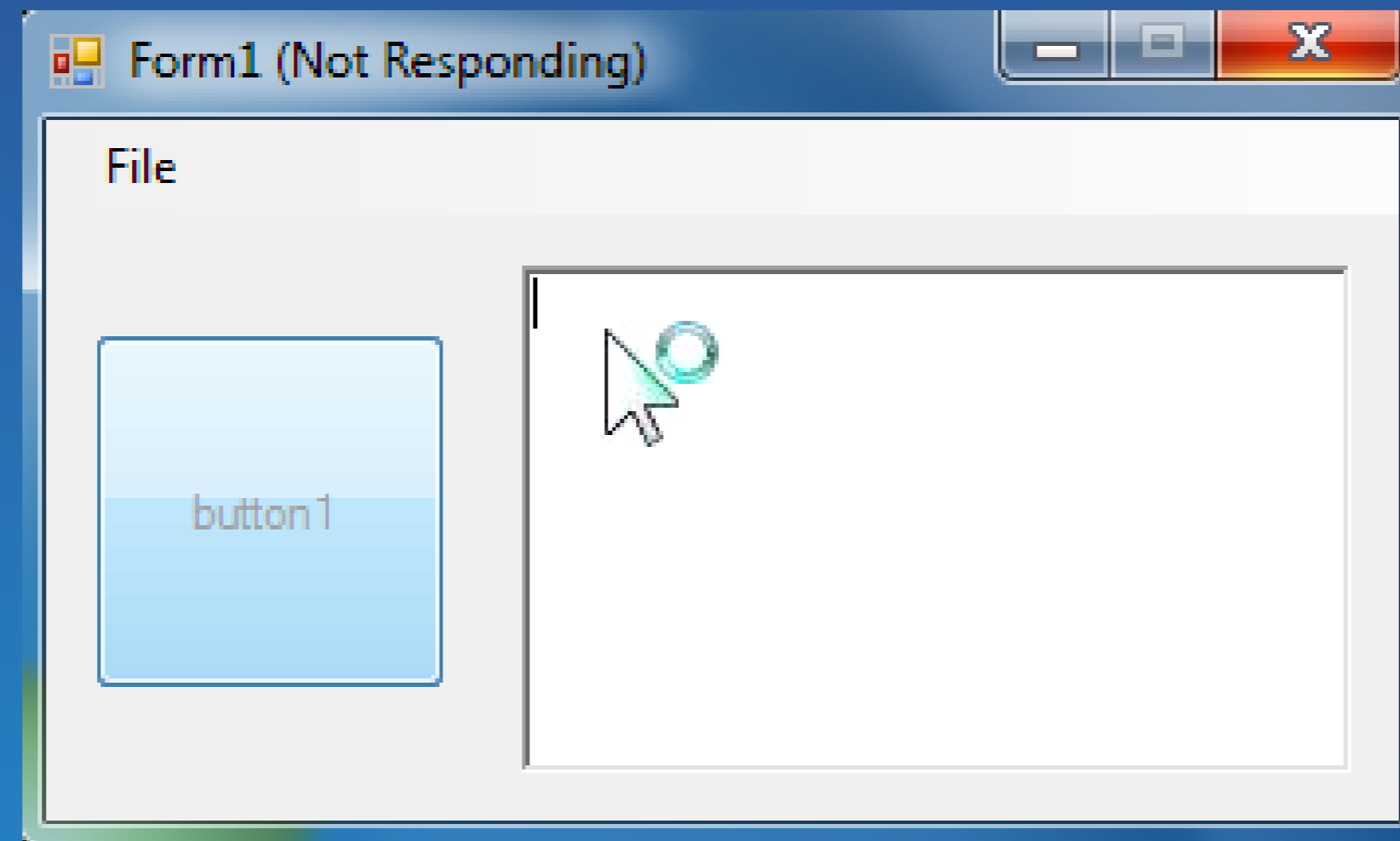
# Agenda

- Future
    - Visual Studio Async
    - TPL Dataflow
-

# Visual Studio Async Trends

Increasingly connected applications

- More latency (e.g. everything as a service)
- More UI responsiveness problems
- User → =(



# Visual Studio Async

## Asynchronous Programming in .NET Today

```
// Synchronous  
TResult Foo(...);
```

```
// Asynchronous Programming Model (APM)  
IAsyncResult BeginFoo(..., AsyncCallback callback, object state);  
TResult EndFoo(IAsyncResult asyncResult);
```

```
// Event-based Asynchronous Pattern (EAP)  
public void FooAsync(...);  
public event EventHandler<FooCompletedEventArgs> FooCompleted;
```

# Visual Studio Async

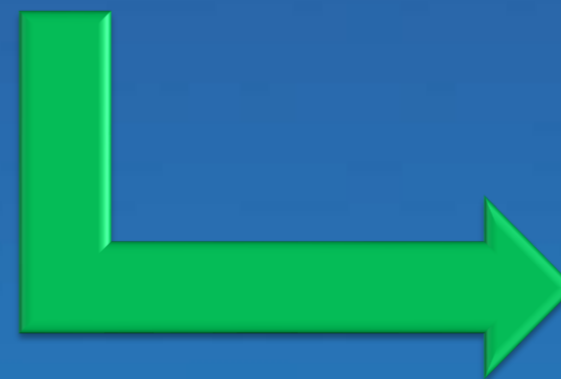
## Your synchronous code with .NET 4...

```
public void CopyStreamToStream(Stream source, Stream destination)
{
    byte[] buffer = new byte[0x1000];
    int numRead;
    while ((numRead = source.Read(buffer, 0, buffer.Length)) != 0)
    {
        destination.Write(buffer, 0, numRead);
    }
}
```

# Visual Studio Async

## An expert's asynchronous code with .NET 4...

```
public void CopyStreamToStream(Stream source, Stream destination)
{
    byte[] buffer = new byte[0x1000];
    int numRead;
    while ((numRead = source.Read(buffer, 0, buffer.Length)) != 0)
    {
        destination.Write(buffer, 0, numRead);
    }
}
```



```
public IAsyncResult BeginCopyStreamToStream(
    Stream source, Stream destination)
{
    var tcs = new TaskCompletionSource<object>();
    byte[] buffer = new byte[0x1000];

    Action<IAsyncResult> readWriteLoop = null;
    readWriteLoop = iar =>
    {
        try
        {
            for (bool isRead = iar == null; ; isRead = !isRead)
            {
                switch (isRead)
                {
                    case true:
                        iar = source.BeginRead(buffer, 0, buffer.Length,
                            readResult =>
                            {
                                if (readResult.CompletedSynchronously) return;
                                // ... (readResult);
                                // ... (readResult);
                            });
                        break;
                    case false:
                        iar = destination.BeginWrite(buffer, 0, buffer.Length,
                            writeResult =>
                            {
                                if (writeResult.CompletedSynchronously) return;
                                // ... (writeResult);
                                // ... (writeResult);
                            });
                        break;
                }
            }
        }
        catch (Exception e) { tcs.TrySetException(e); }
    };
    readWriteLoop(null);

    return tcs.Task;
}

public void EndCopyStreamToStream(IAsyncResult asyncResult)
{
    ((Task)asyncResult).Wait();
}
```

# Visual Studio Async

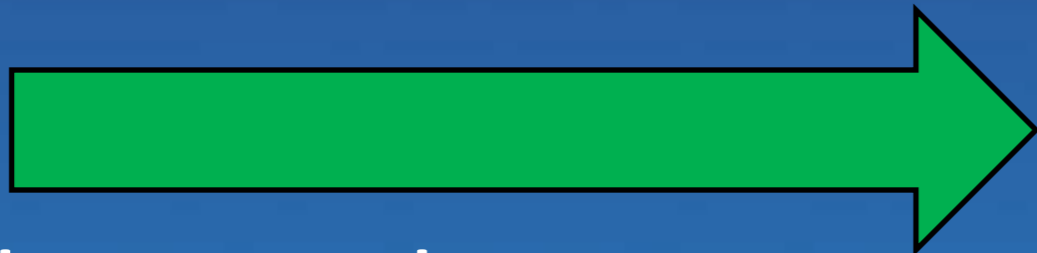
## Your asynchronous code with the Visual Studio Async CTP...

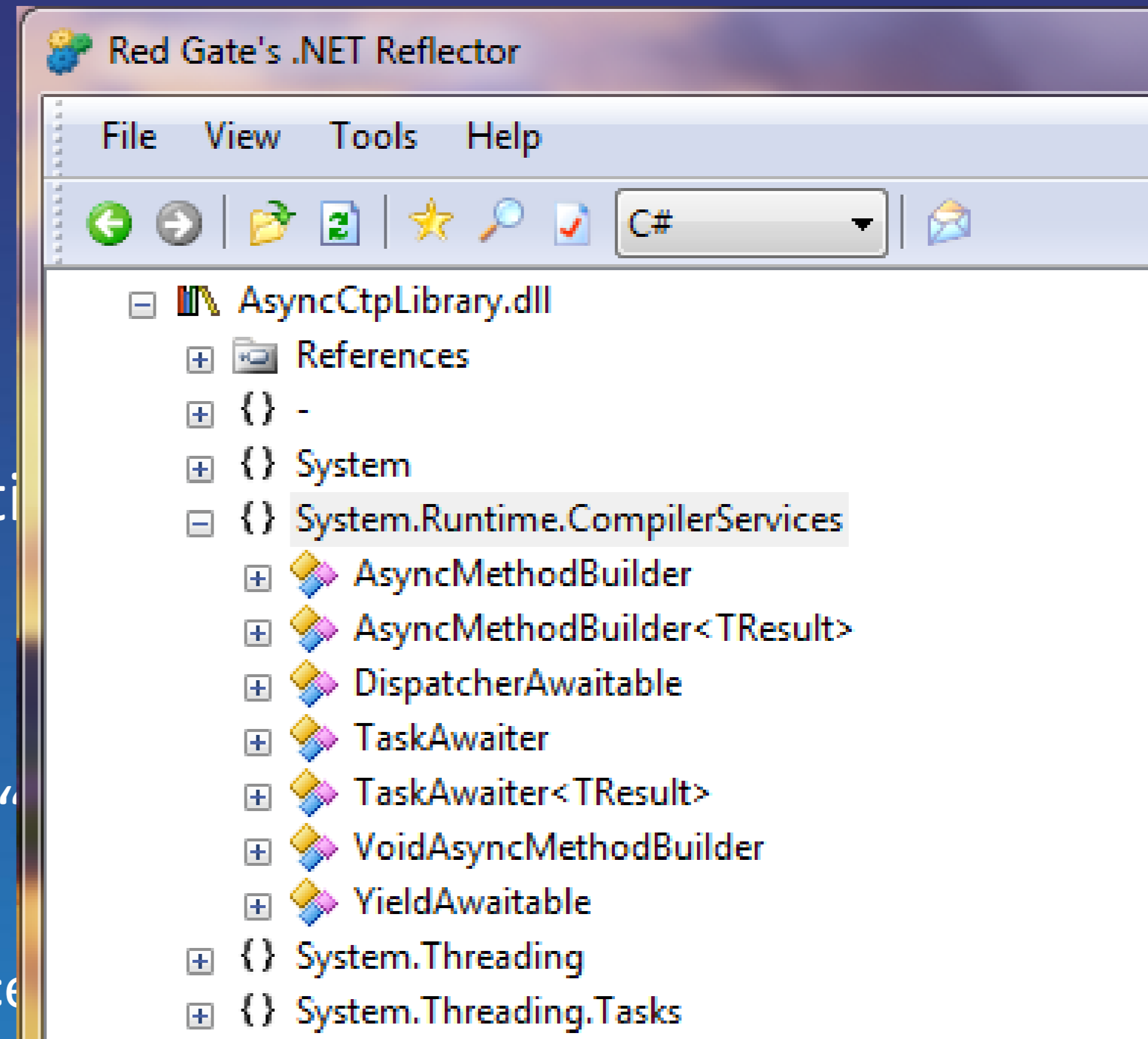
```
public void CopyStreamToStream(Stream source, Stream destination)
{
    byte[] buffer = new byte[0x1000];
    int numRead;
    while ((numRead = source.Read(buffer, 0, buffer.Length)) != 0)
    {
        destination.Write(buffer, 0, numRead);
    }
}
```

```
public async Task CopyStreamToStreamAsync(Stream source, Stream destination)
{
    byte[] buffer = new byte[0x1000];
    int numRead;
    while ((numRead = await source.ReadAsync(buffer, 0, buffer.Length)) != 0)
    {
        await destination.WriteAsync(buffer, 0, numRead);
    }
}
```



# Visual Studio Async Tasks and Language

- Language
  - “async” modifier marks method or
  - “await” operator yields control until
- Framework 
  - Task and Task<TResult> represent “
  - E.g. Async I/O, background work, etc.
  - Single object for status, result, and exce
  - New APIs round out the experience



# Visual Studio Async

## Related Additions

- Combinators
    - Task.WhenAll, Task.WhenAny
  - Timer integration
    - Task.Delay(TimeSpan),  
CancellationTokenSource.CancelAfter(TimeSpan)
  - Task scheduling
    - ConcurrentExclusiveSchedulerPair
  - Fine-grained control
    - TaskCreationOptions.DenyChildAttach
    - EnumerablePartitionerOptions
  - ThreadLocal.Values
-

# Visual Studio Async

## Async in .NET Tomorrow

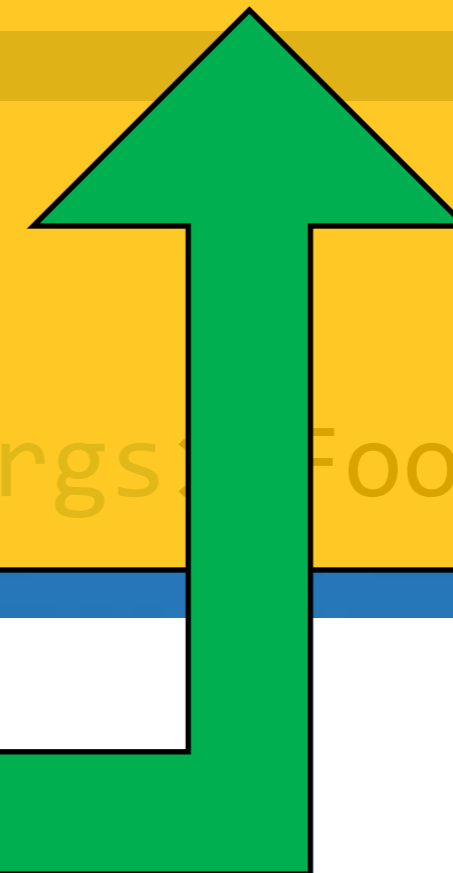
```
// Synchronous  
TResult Foo(...);
```

```
// Asynchronous Programming Mod  
IAsyncResult BeginFoo(..., Async  
TResult EndFoo(IAsyncResult asyncResult);
```

```
// Event-based Asynchronous Pattern (EAP)  
public void FooAsync(...);  
public event EventHandler<FooCompletedEventArgs> FooCompleted;
```

```
// Task-based Asynchronous Pattern (TAP)  
Task<TResult> FooAsync(...);
```

```
System.IO.Stream.ReadAsync(...);  
                .WriteAsync(...);  
                .FlushAsync();  
                .CopyToAsync(...);
```



# Visual Studio Async Demo

- DEMO – Sleeping on the UI
  - Async CTP: <http://msdn.microsoft.com/en-us/vstudio/async.aspx>
-

# Agenda Checkpoint

- Future
  - Tasks and Language
  - **TPL Dataflow**

# TPL Dataflow

## Complementing Parallel Programming in .NET 4

- Proactive in nature
    - “Here’s the data. Now set up the computation.”
    - Primitives for task and data parallelism
  - Missing the reactive piece
    - “Set up the computation. Now here’s the data.”
    - Primitives for dataflow parallelism
-

# TPL Dataflow

## Overview

- Primitives for in-process message passing
    - Blocks that can buffer and process data
    - Can be linked together to create networks
  - Inspired by
    - Decades of computer science research/history
    - Related Microsoft technologies
      - Asynchronous Agents library in Visual C++ 2010
      - CCR from Microsoft Robotics
      - Axum incubation project
-

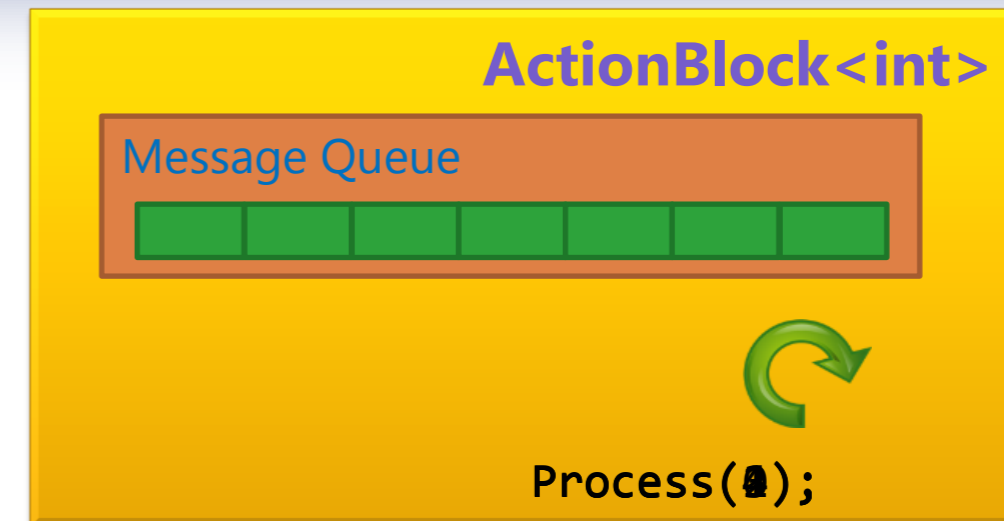
# TPL Dataflow

## Async Posting Example

```
var c = new ActionBlock<int>(i =>
{
    Process(i);
});

for(int i = 0; i < 5; i++)
{
    c.Post(i);
}
```

4





# TPL Dataflow

## Blocks for Buffering and Propagation

- `BufferBlock<T>`
    - Buffers an unlimited number of elements
    - Delivers each element to at most 1 target
  - `WriteOnceBlock<T>`
    - Accepts and buffers only 1 element, ever
    - Delivers the 1 element to all linked targets
  - `BroadcastBlock<T>`
    - Overwrites each element with the next (buffers until this happens)
    - Delivers each element to all linked targets
-

# TPL Dataflow

## Blocks for Executing

- `ActionBlock<TInput>`
  - Executes an `Action<TInput>` for each element
  - Buffers input until processed
- `TransformBlock<TInput, TOutput>`
  - Executes a `Func<TInput, TOutput>` for each element
  - Buffers input until processed and output until consumed
- `TransformManyBlock<TInput, TOutput>`
  - Executes a `Func<TInput, IEnumerable<TOutput>>` for each element
  - Buffers input until processed and output until consumed

# TPL Dataflow

## Blocks for Joining

- `BatchBlock<T>`
  - Groups multiple Ts into one `T[]`
  - Supports greedy and non-greedy
- `JoinBlock<T1, T2>`
  - Groups on T1 and one T2 to form a `Tuple<T1, T2>`
  - Supports greedy and non-greedy
- `BatchedJoinBlock<T1, T2>`
  - Groups T1s and T2s into one `Tuple<IList<T1>, IList<T2>>`

# Related Content

- Parallel Programming Dev Center:
  - <http://msdn.microsoft.com/en-us/concurrency>
- Downloads
  - Async CTP: <http://msdn.microsoft.com/en-us/vstudio/async.aspx>
  - TPL Dataflow: <http://msdn.microsoft.com/en-us/devlabs/tclabs>
- Forums
  - Async: <http://social.msdn.microsoft.com/Forums/en-US/async/threads>
  - Parallel Extensions: <http://social.msdn.microsoft.com/Forums/en-US/parallelextensions/threads>

# *Microsoft*<sup>®</sup>

© 2008 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries.

The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.