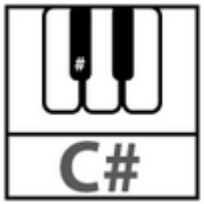
C# Events, Delegates and Lambdas

Dan Wahlin

Twitter: @DanWahlin

Blog: http://weblogs.asp.net/dwahlin

Pre-Req Classes



C# Fundamentals - Part 1

This course is designed to give you everything you need to become a productive C# developer on the .NET platform

Authored by: Scott Allen
Duration: 6h 17m
Level: Beginner

The Role of Events, Delegates and Event Handlers

Creating Delegates, Events and EventArgs

Handling Events

Lambdas, Action<T> and Func<T,TResult>

The Role of Events, Delegates and Event Handlers

Dan Wahlin

Twitter: @DanWahlin

Blog: http://weblogs.asp.net/dwahlin

The Role of Events

The Role of Delegates

The Role of Event Handlers


Event Handler
EventArgs
Event Raiser
Delegate

The Role of Events

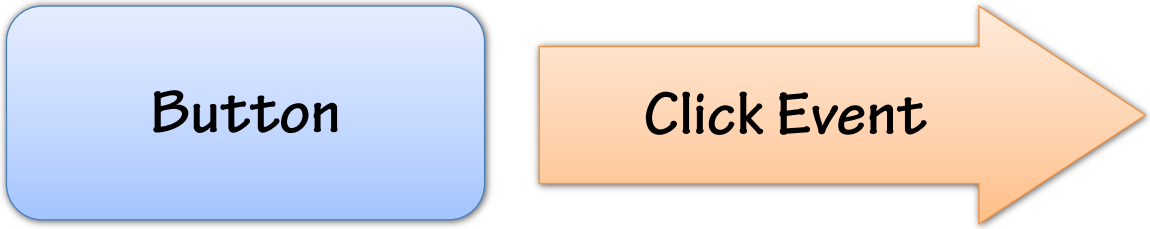

Event Raiser

What is an Event?

- Events are notifications
- Play a central role in the .NET framework
- Provide a way to trigger notifications from end users or from objects

The Role of Events

- Events signal the occurrence of an action/notification



- Objects that raise events don't need to explicitly know the object that will handle the event

- Events pass EventArgs (event data)

The Role of Delegates

pluralsight
hardcore developer training

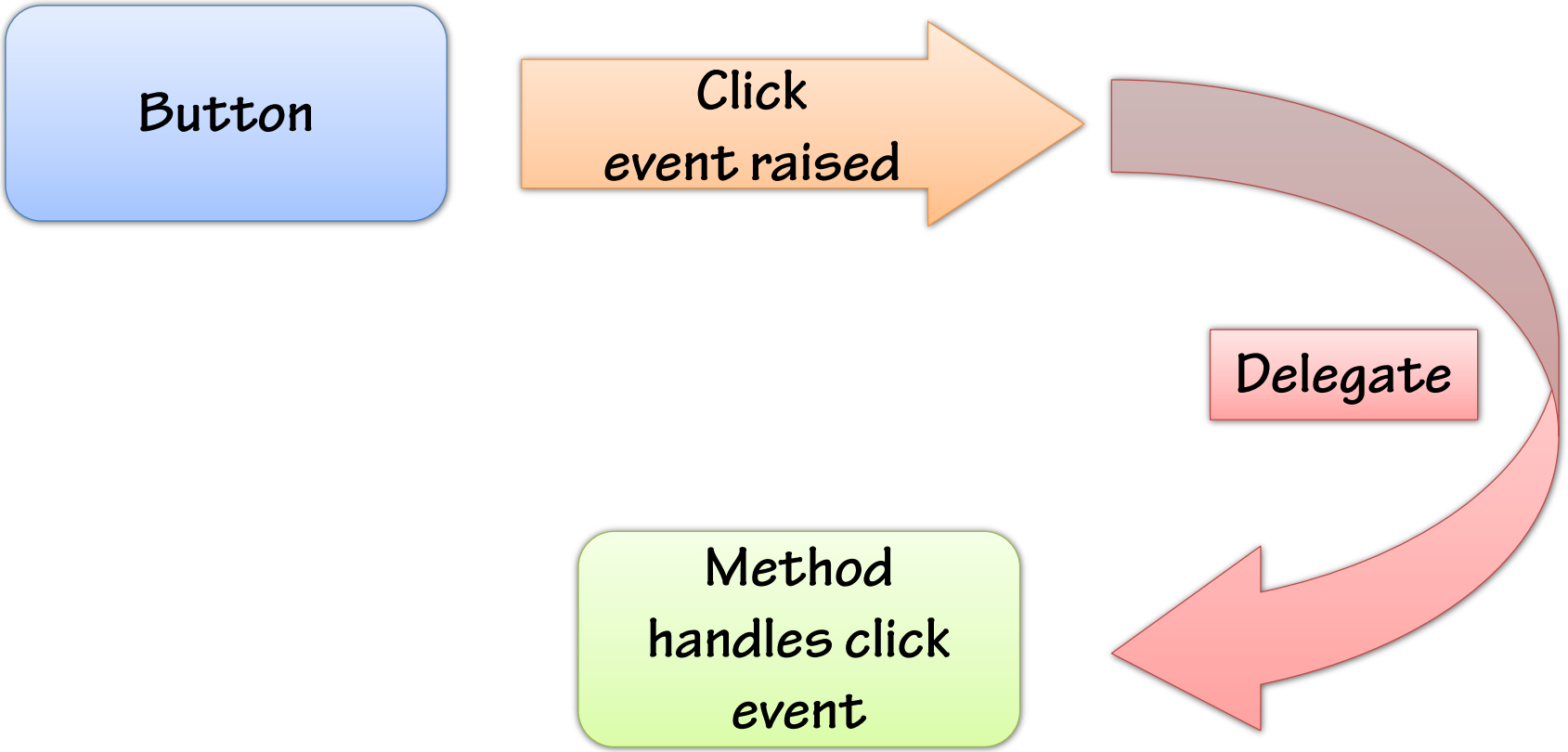

Delegate

What is a Delegate?

- A delegate is a specialized class often called a "Function Pointer"
- The glue/pipeline between an event and an event handler
- Based on a MulticastDelegate base class

Delegates are a Pipeline

The Role of Delegates



The Role of Event Handlers

pluralsight
hardcore developer training


EventArgs
Event Handler

What is an Event Handler?

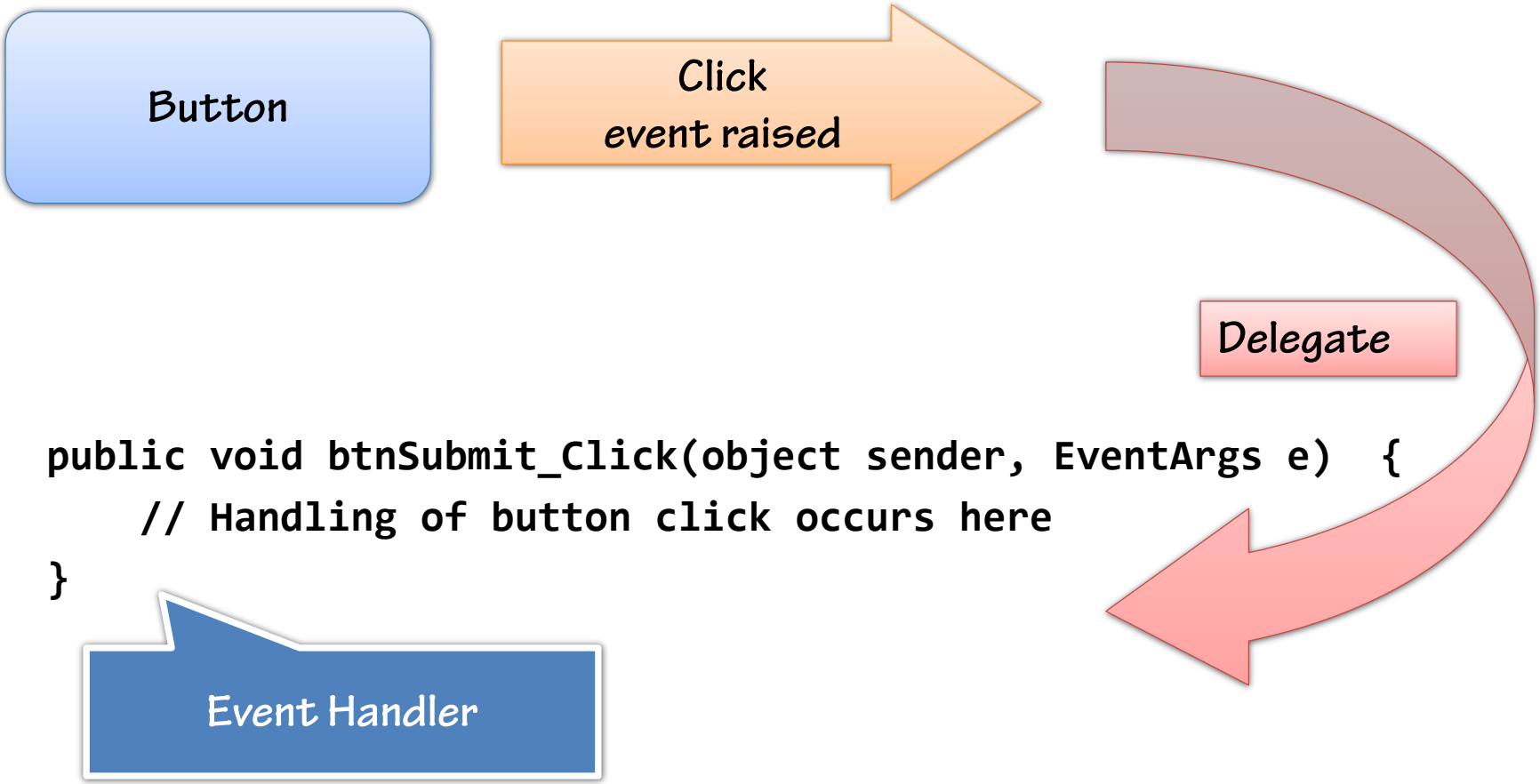
- **Event handler is responsible for receiving and processing data from a delegate**

- **Normally receives two parameters:**
  - Sender
  - EventArgs

- **EventArgs responsible for encapsulating event data**

The Role of Event Handlers

An event handler is responsible for accessing data passed by a delegate:



Summary

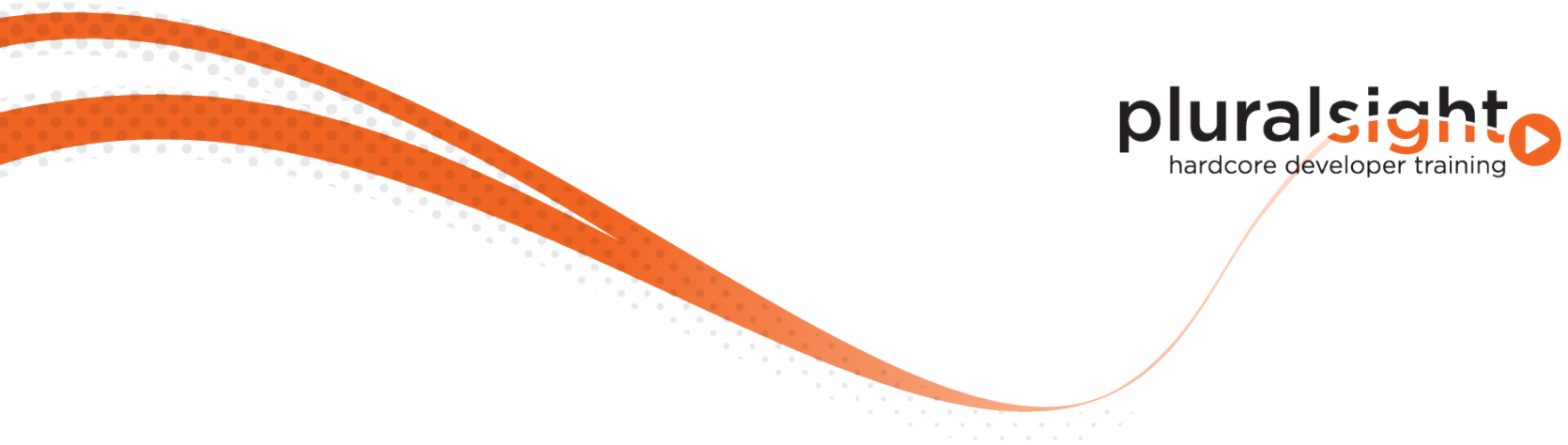
- The .NET framework relies heavily on events and delegates
- Events provide notifications and send data using EventArgs
- Delegates act as the glue/pipeline between events and event handlers
- Event Handlers receive and process EventArgs data

Creating Delegates, Events and EventArgs

Dan Wahlin

Twitter: @DanWahlin

Blog: http://weblogs.asp.net/dwahlin

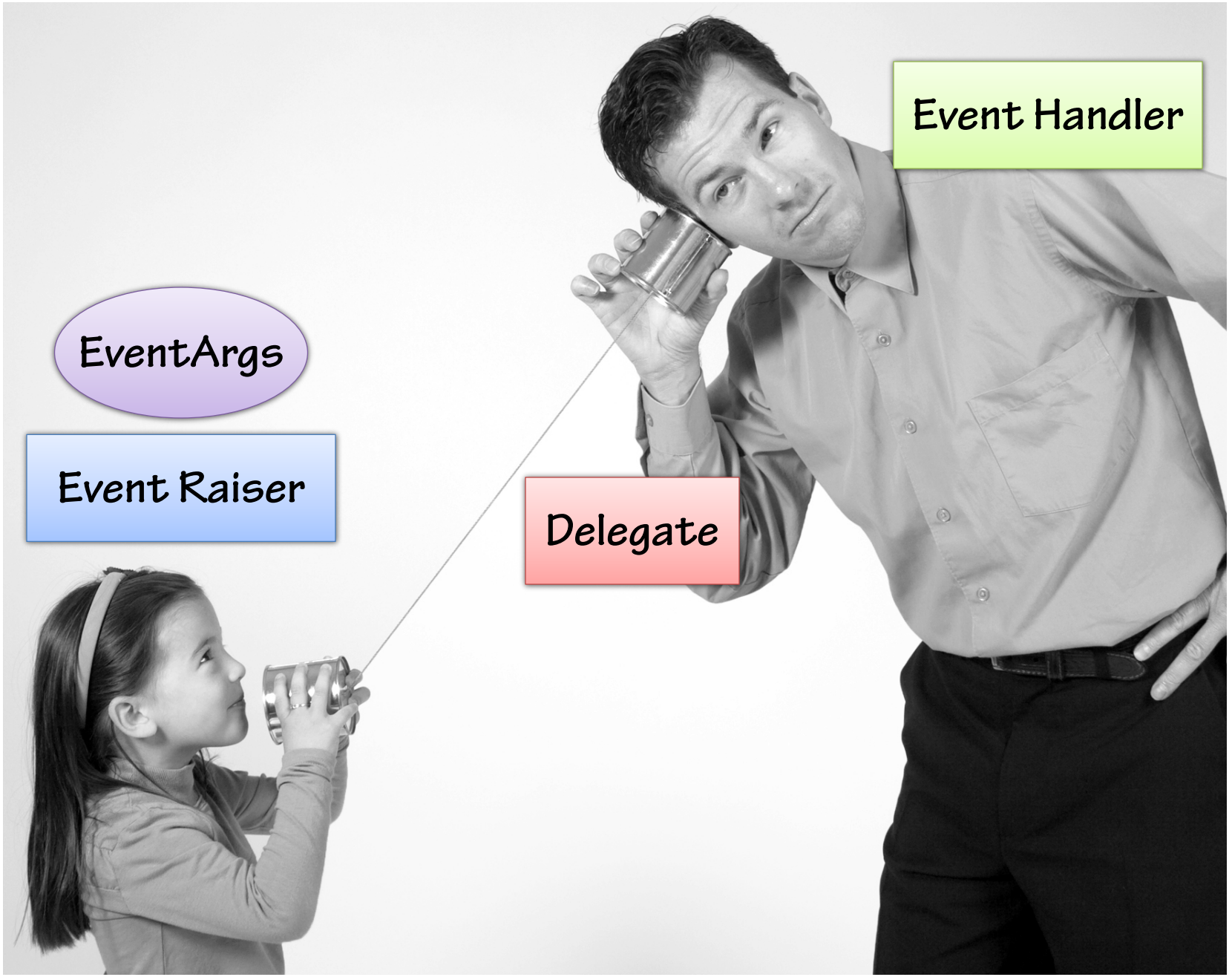


Creating a Delegate

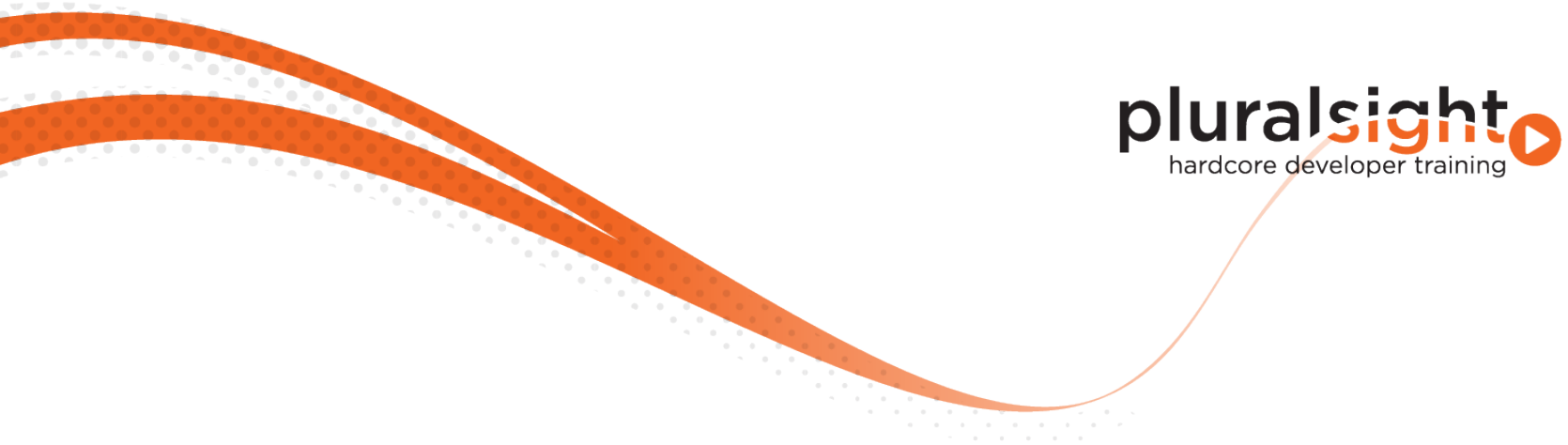
Defining an Event

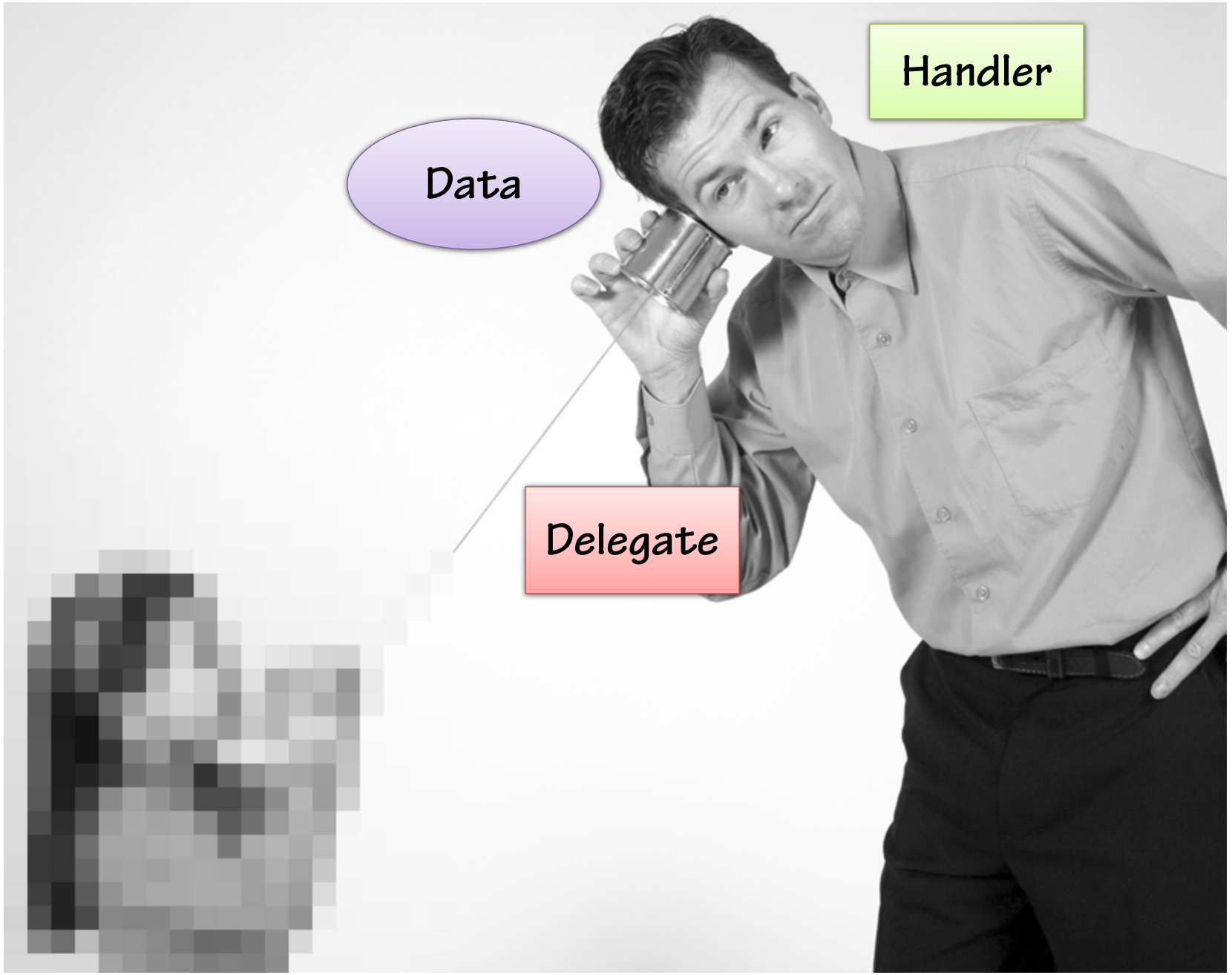
Raising Events

Creating an EventArgs Class


Event Handler
EventArgs
Event Raiser
Delegate

Creating a Delegate




Handler
Data
Delegate

Creating Delegates

- Custom delegates are defined using the **delegate** keyword:

```
public delegate void WorkPerformedHandler(int hours,
    WorkType workType);
```

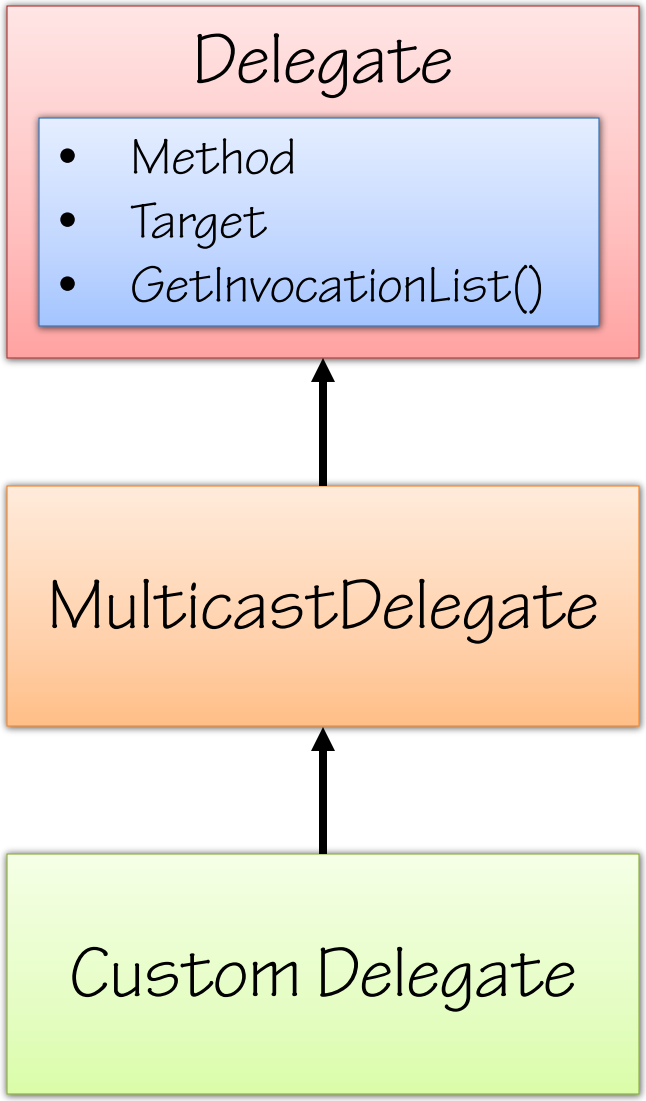
Delegate and Handler Method Parameters

- The delegate signature must be mimicked by a handler method:

```
public delegate void WorkPerformedHandler(int hours,
    WorkType workType);
```

```
public void Manager_WorkPerformed(int workHours,
   WorkType wType) {
   ...
}
```

Delegate Base Classes



What is a Multicast Delegate?

- Can reference more than one delegate function
- Tracks delegate references using an invocation list
- Delegates in the list are invoked sequentially

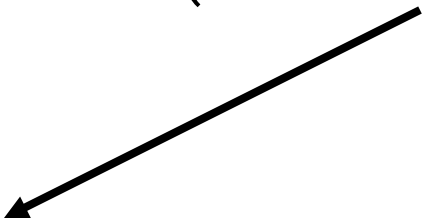
Creating a Delegate Instance

Delegate

```
public delegate void WorkPerformedHandler(int hours,
    WorkType workType);
```

Delegate Instance

```
WorkPerformedHandler del1 =
  new WorkPerformedHandler(WorkPerformed1);
```

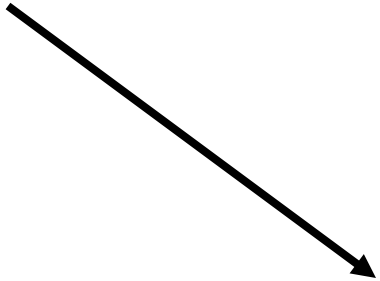


Handler

```
static void WorkPerformed1(int hours, WorkType workType)
{
    Console.WriteLine("WorkPerformed1 called");
}
```

Invoking a Delegate Instance

```
WorkPerformedHandler del1 =
  new WorkPerformedHandler(WorkPerformed1);

del1(5, WorkType.Golf);
```

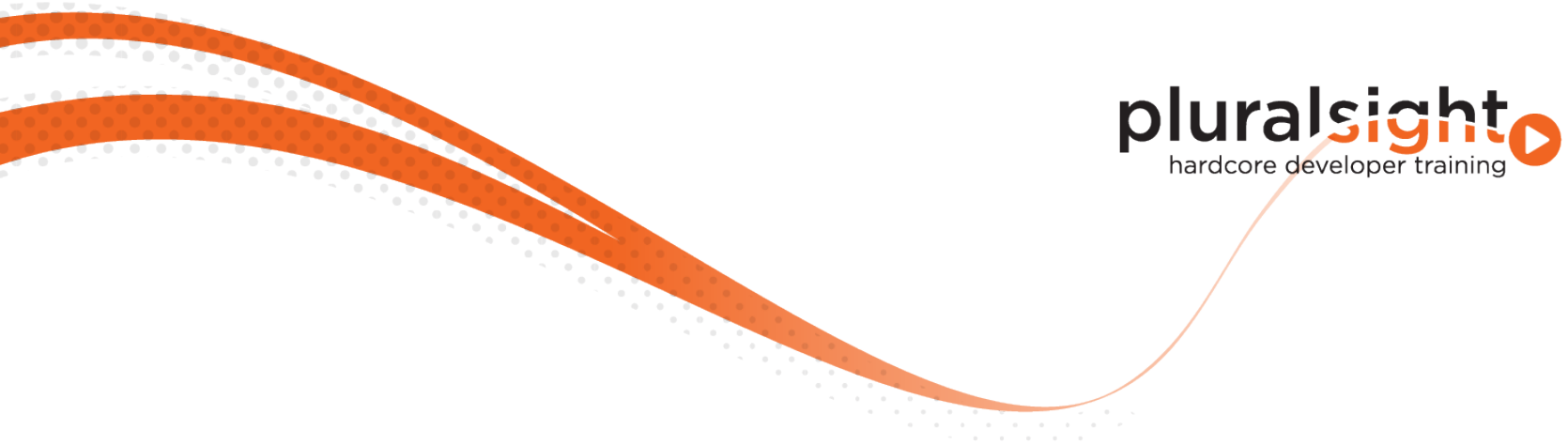


```
static void WorkPerformed1(int hours, WorkType workType)
{
    Console.WriteLine("WorkPerformed1 called");
}
```

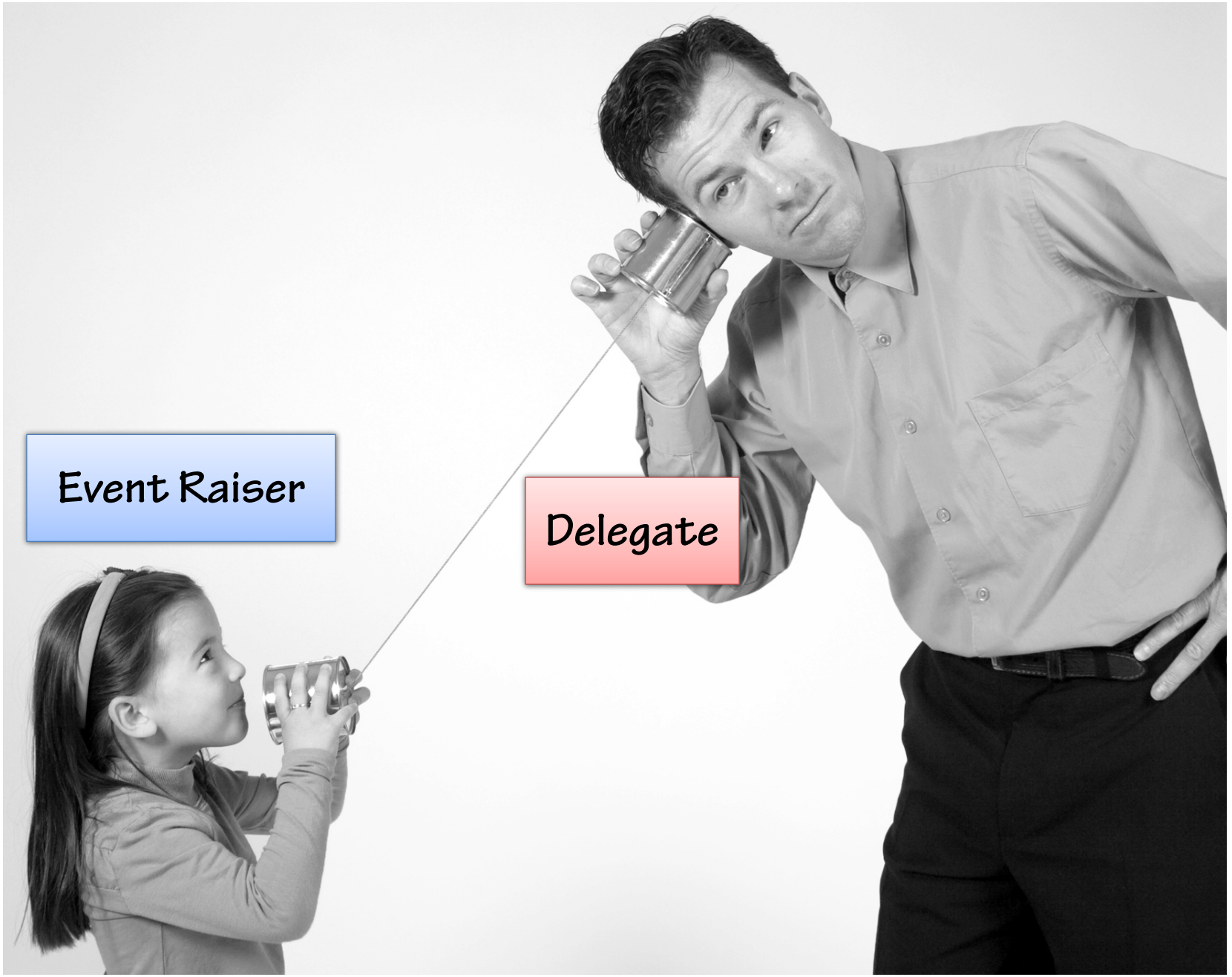
Adding to the Invocation List

```
WorkPerformedHandler del1 =
  new WorkPerformedHandler(WorkPerformed1);

WorkPerformedHandler del2 =
  new WorkPerformedHandler(WorkPerformed2);

del1 += del2;

del1(5, WorkType.GoToMeetings);
```

Defining an Event



pluralsight
hardcore developer training


Event Raiser
Delegate

Defining an Event

- Events can be defined in a class using the **event** keyword

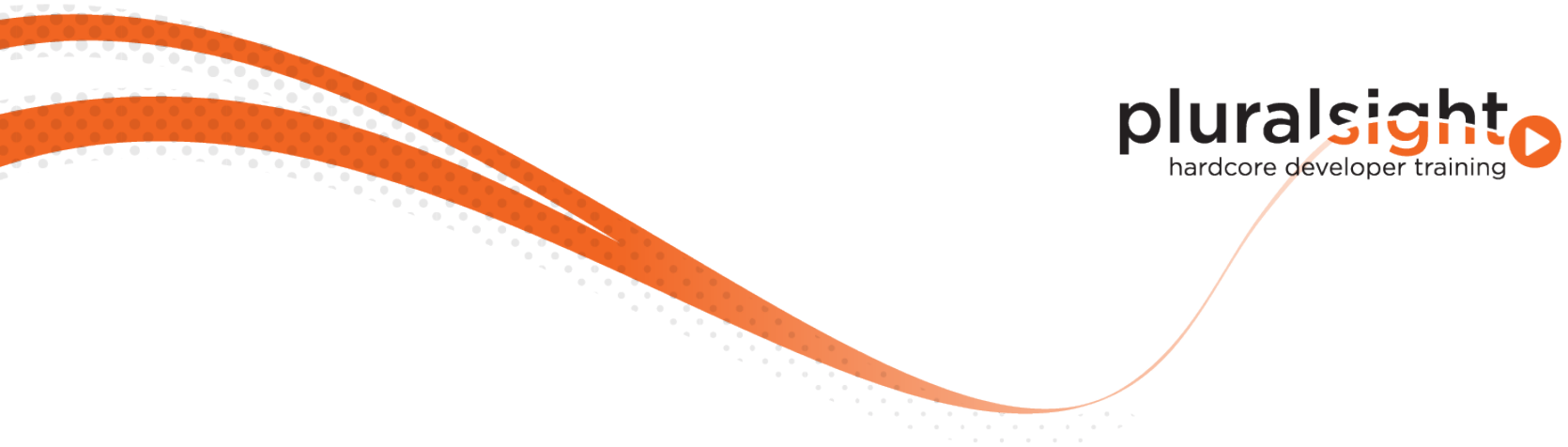
```
public event WorkPerformedHandler WorkPerformed;
```

Defining an Event with add/remove

- Events can be defined using add/remove accessors:

```
private WorkPerformedHandler _WorkPerformedHandler;
public event WorkPerformedHandler WorkPerformed
{
    [MethodImpl(MethodImplOptions.Synchronized)]
    add
    {
        _WorkPerformedHandler = (WorkPerformedHandler)Delegate.Combine(
          _WorkPerformedHandler, value);
    }
    [MethodImpl(MethodImplOptions.Synchronized)]
    remove
    {
        _WorkPerformedHandler = (WorkPerformedHandler)Delegate.Remove(
           _WorkPerformedHandler, value);
    }
}
```

Raising Events



pluralsight
hardcore developer training

Raising Events

- Events are raised by calling the event like a method:

```
if (WorkPerformed != null) {
    WorkPerformed(8, WorkType.GenerateReports);
}
```

- Another option is to access the event's delegate and invoke it directly:

```
WorkPerformedHandler del = WorkPerformed as WorkPerformedHandler;
if (del != null) {
    del(8, WorkType.GenerateReports);
}
```

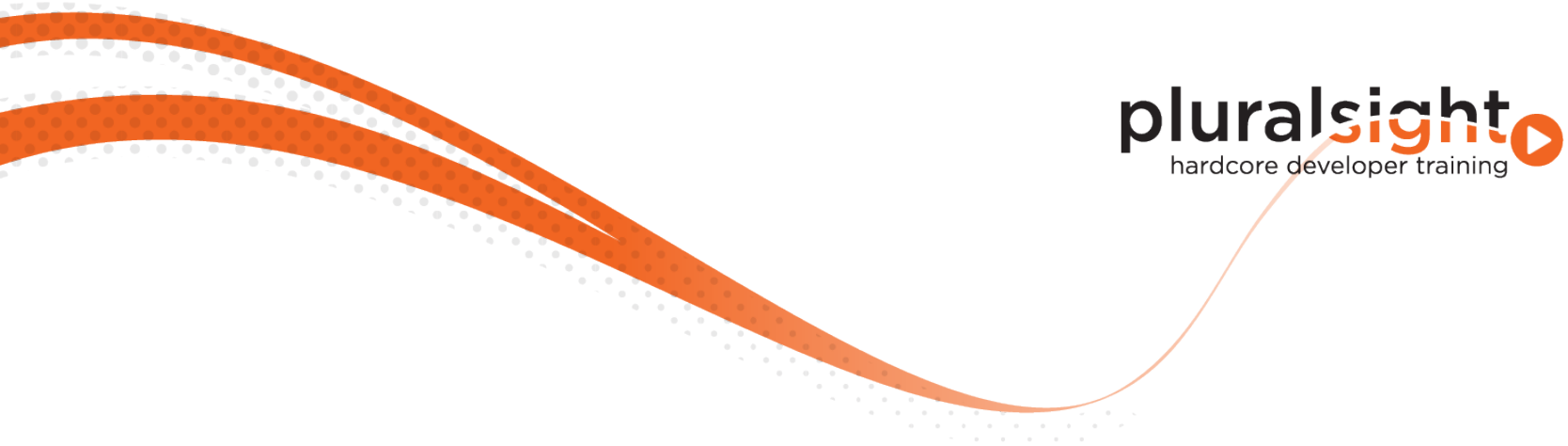
Exposing and Raising Events

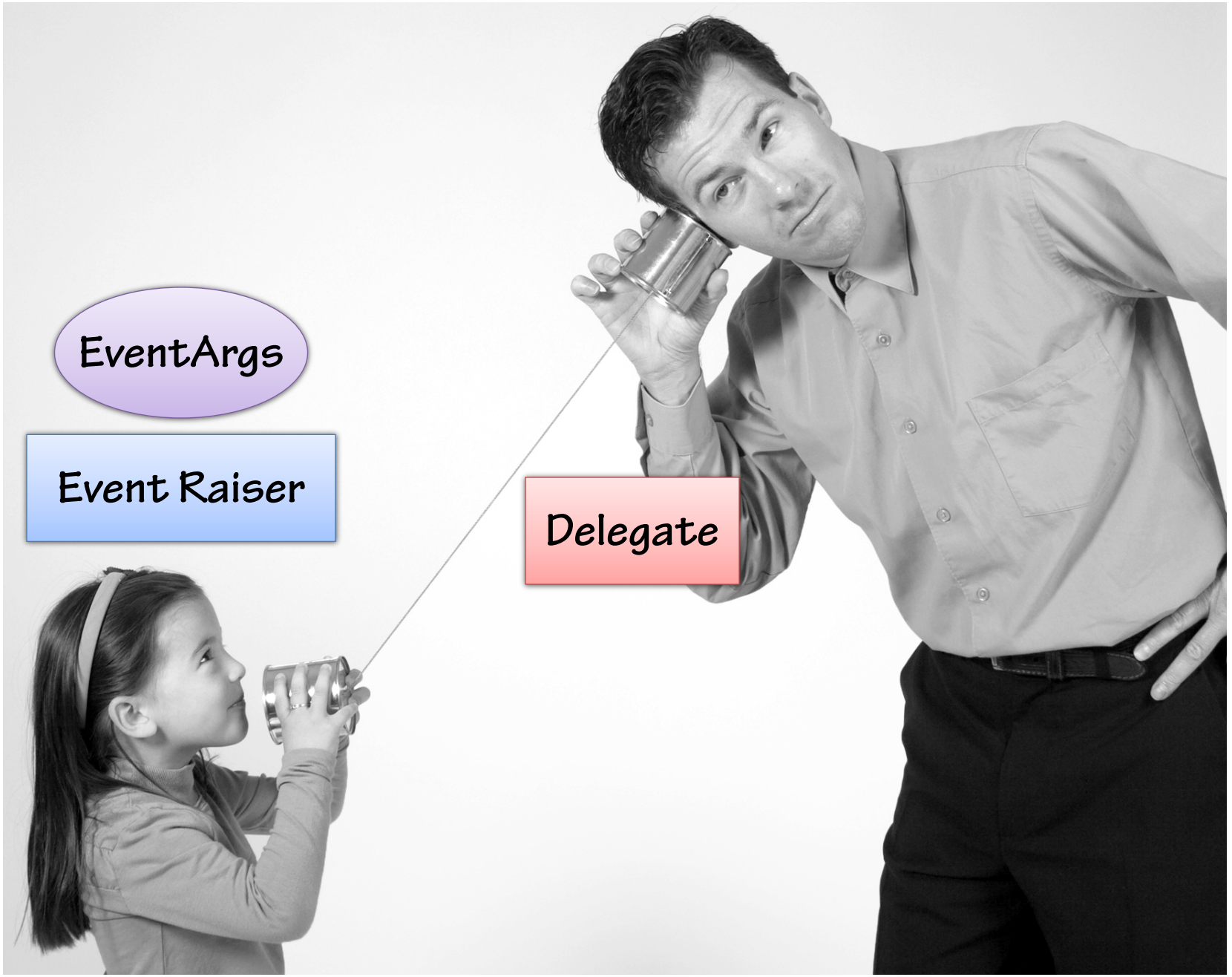
```
public delegate void WorkPerformedHandler(int hours, WorkType workType);
public class Worker
{
    public event WorkPerformedHandler WorkPerformed;
    public virtual void DoWork(int hours, WorkType workType)
    {
        // Do work here and notify consumer that work has been performed
        OnWorkPerformed(hours, workType);
    }

    protected virtual void OnWorkPerformed(int hours, WorkType workType)
    {
         WorkPerformedHandler del = WorkPerformed as WorkPerformedHandler;
         if (del != null) //Listeners are attached
         {
             del(hours, workType);
         }
    }
}
```

Event Definition

Raise Event

Creating an EventArgs Class




EventArgs
Event Raiser
Delegate

Creating a Custom EventArgs Class

- The **EventArgs** class is used in the signature of many delegates and event handlers:

```
public void button_Click(object sender, EventArgs e)
{
    // Handle button click
}
```

- When custom data needs to be passed the **EventArgs** class can be extended.

Deriving from the System.EventArgs Class

```
public class WorkPerformedEventArgs : System.EventArgs
{
    public int Hours { get; set; }
    public WorkType WorkType { get; set; }
}
```

Using a Derived System.EventArgs Class

- To use a custom EventArgs class, the delegate must reference the class in its signature:

```
public delegate void WorkPerformedHandler(object sender,
   WorkPerformedEventArgs e);
```

Holds event data

Sender of event

Using EventHandler<T>

.NET includes a generic **EventHandler<T>** class that can be used instead of a custom delegate:

```
public delegate                                    (object sender,
  WorkPerformedEv
```

```
public event EventHandler<WorkPerformedEventArgs> WorkPerformed;
```

Built-in Delegate

Summary

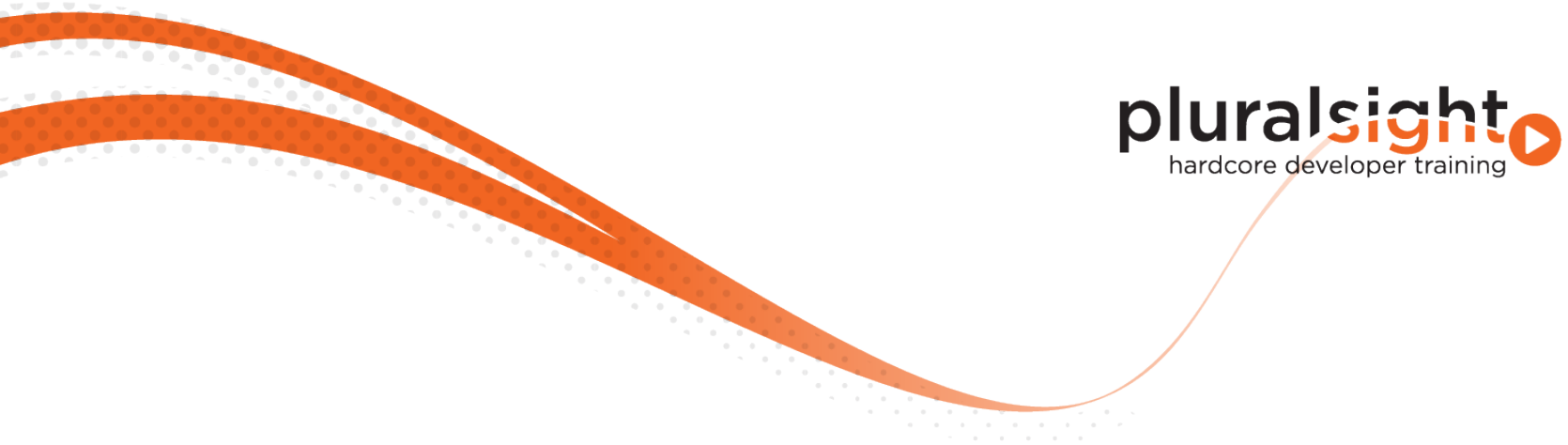
- Events are associated with Delegates
- A standard .NET event signature accepts:
  - Sender
  - Custom EventArgs
- The parameter signature for an event handler matches the delegate signature
- EventHandler<T> provides a simple way to create a custom delegate for an event

Handling Events

Dan Wahlin

Twitter: @DanWahlin

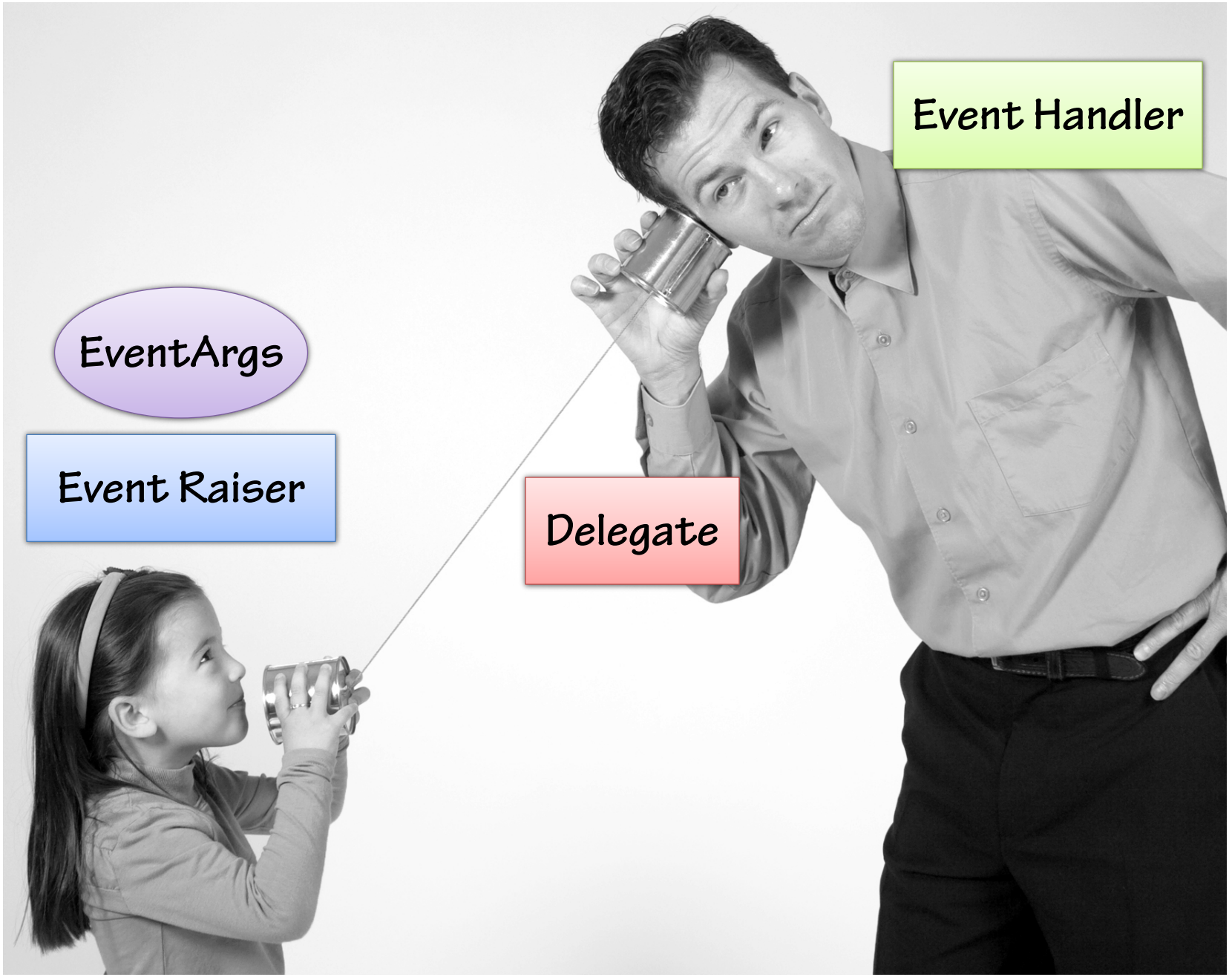
Blog: http://weblogs.asp.net/dwahlin



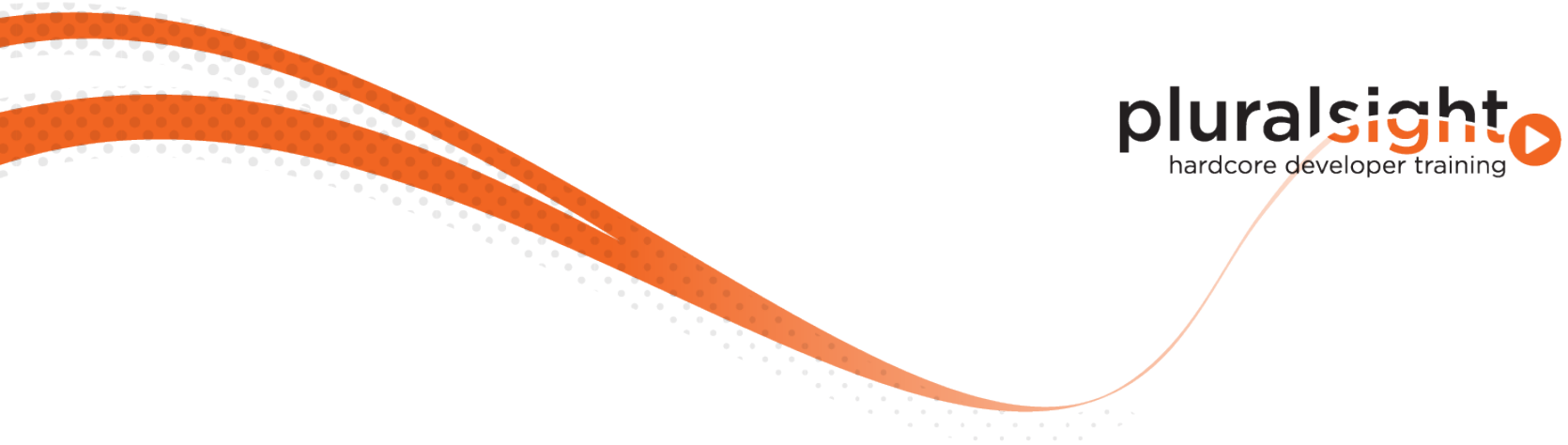
Instantiating Delegates and Handling Events

Delegate Inference

Using Anonymous Methods


Event Handler
EventArgs
Event Raiser
Delegate

Instantiating Delegates and Handling Events



pluralsight
hardcore developer training


Event Handler
Delegate

Delegate and Handler Method Parameters

- The delegate signature must be mimicked by a handler method:

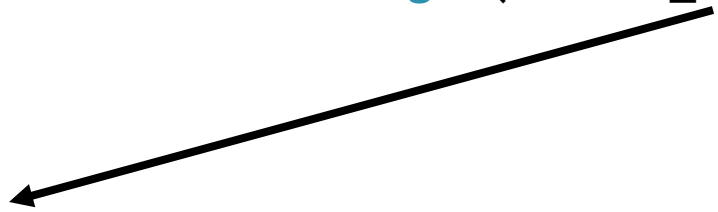
```
public delegate void WorkPerformedHandler(object sender,
    WorkPerformedEventArgs e);
```

```
public void Manager_WorkPerformed(object sender,
    WorkPerformedEventArgs e) {

    ...

}
```

Defining and Attaching Event Handlers

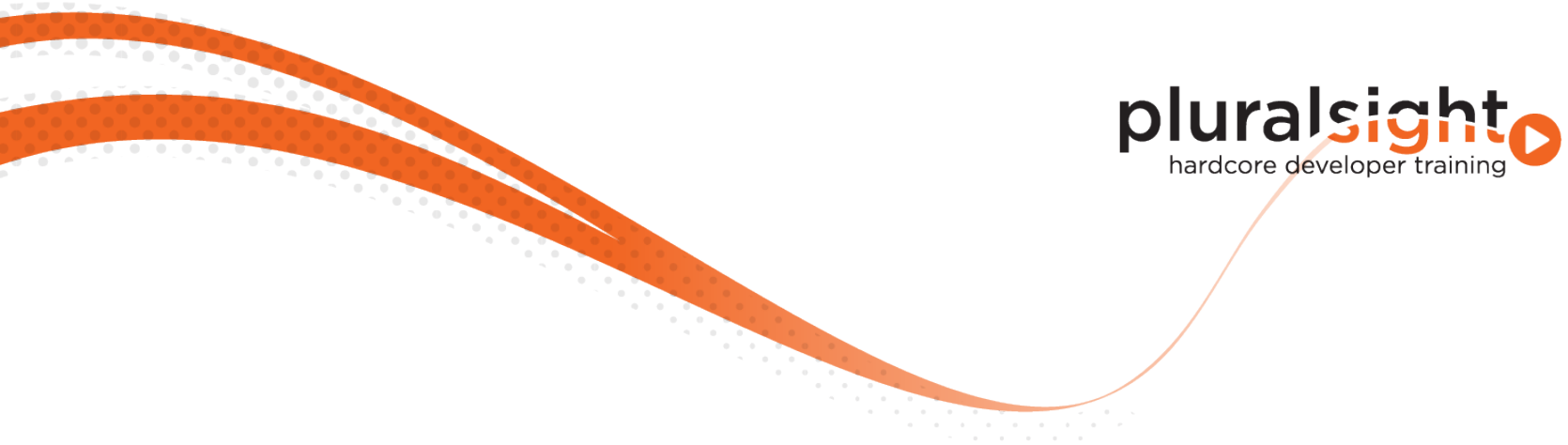
The += operator is used to attach an event to an event handler:

```
var worker = new Worker();
worker.WorkPerformed +=
    new EventHandler<WorkPerformedEventArgs>(worker_WorkPerformed);
```



```
void worker_WorkPerformed(object sender, WorkPerformedEventArgs e)
{
    Console.WriteLine(e.Hours.ToString());
}
```

Delegate Inference



pluralsight
hardcore developer training

Delegate Inference

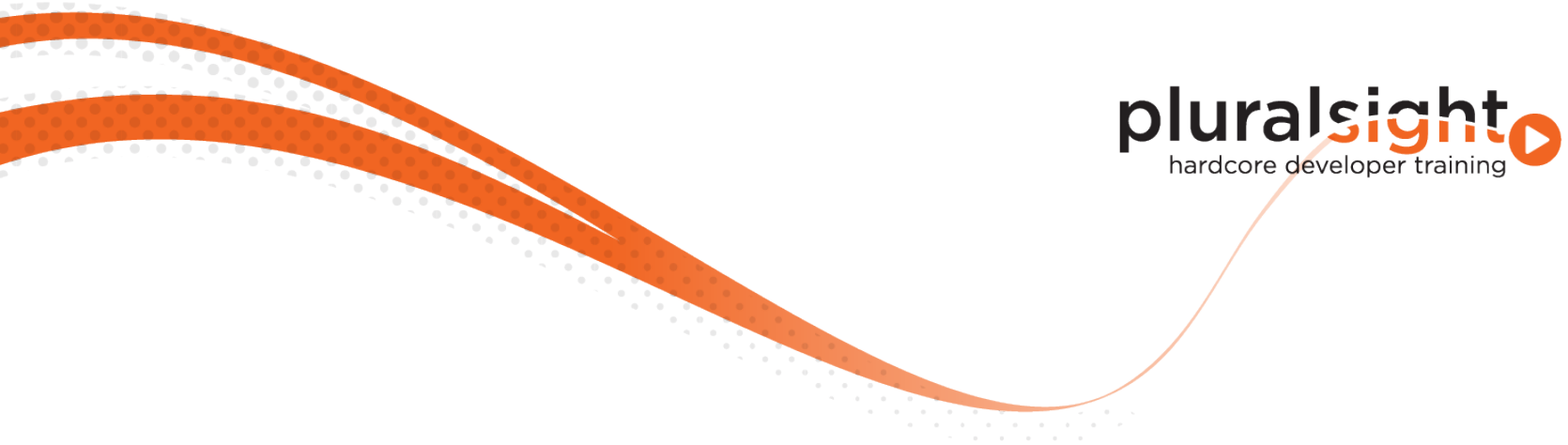
The += operator is used to attach an event to an event handler:

```
var worker = new Worker();
worker.WorkPerformed += worker_WorkPerformed;
```

Compiler will “infer” the delegate

```
void worker_WorkPerformed(object sender, WorkPerformedHandler e)
{
    Console.WriteLine(e.);
}
```

Using Anonymous Methods

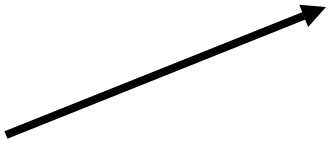


pluralsight
hardcore developer training

Attaching Event Handlers Directly to Events

What if you want to attach an event handler directly to an event?

```
var worker = new Worker();
worker.WorkPerformed += //Attach event handler here
```



```
void worker_WorkPerformed(object sender, WorkPerformedEventArgs e)
{
    Console.WriteLine(e.Hours.ToString());
}
```

Anonymous Methods

- Anonymous methods allow event handler code to be hooked directly to an event
- Anonymous methods are defined using the delegate keyword:

```
var worker = new Worker();
worker.WorkPerformed += delegate(object sender,
                                 WorkPerformedEventArgs e)
    {
        Console.WriteLine(e.Hours.ToString());
    }; //End of anonymous method
```

Summary

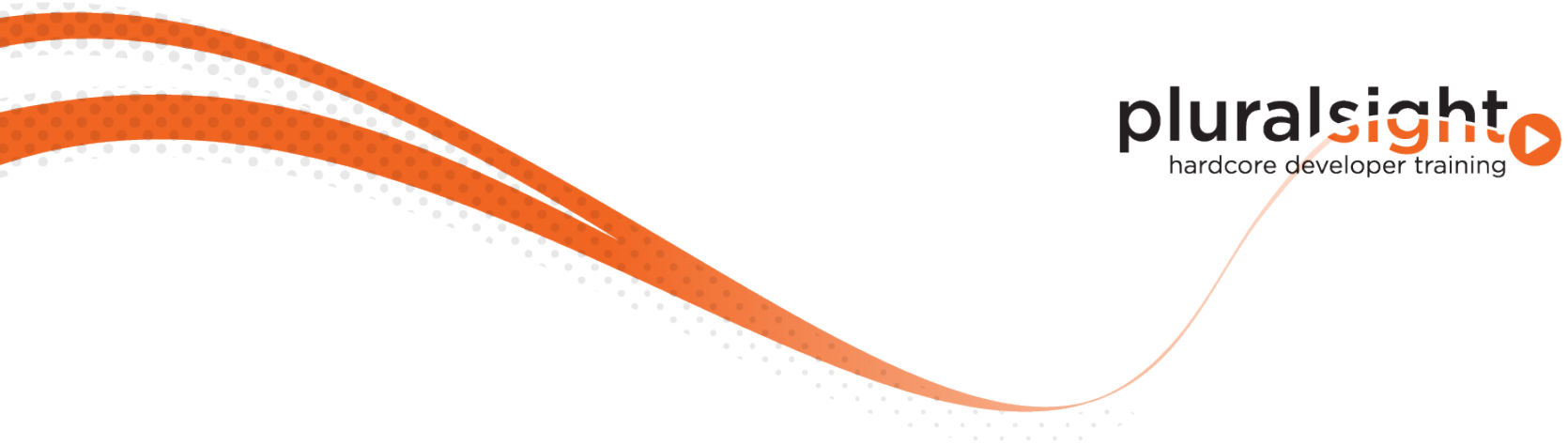
- To listen to an event you must construct a delegate and add it to the invocation list
- The C# compiler provides “delegate inference” functionality
- Event handler methods can be attached directly to events

Lambdas, Action<T> and Func<T,TResult>

Dan Wahlin

Twitter: @DanWahlin

Blog: http://weblogs.asp.net/dwahlin



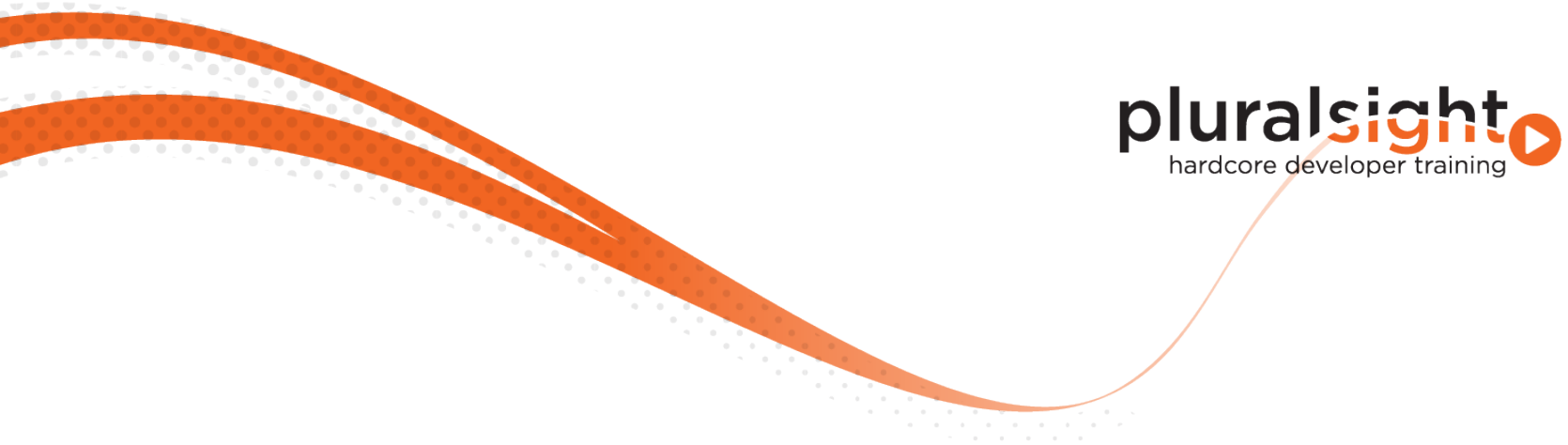
pluralsight
hardcore developer training

Lambdas and Delegates

Using Action<T>

Using Func<T,TResult>

Lambdas and Delegates

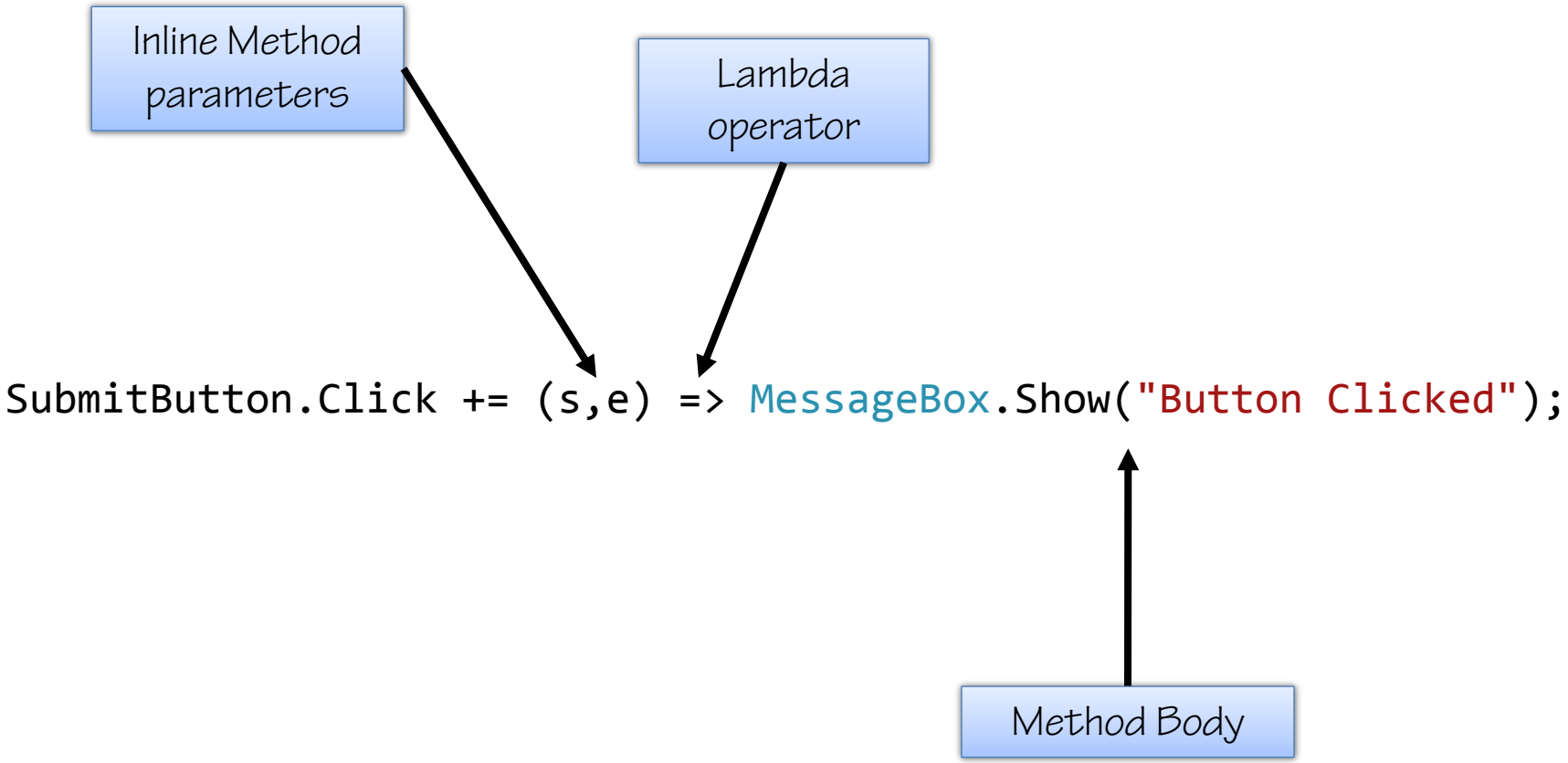


pluralsight
hardcore developer training

Anonymous Methods in Action

```
SubmitButton.Click += delegate(object sender, EventArgs e)
 {
     MessageBox.Show("Button Clicked");
 };
```

Understanding Lambda Expressions



Assigning a Lambda to a Delegate

- Lambda expressions can be assigned to any delegate:

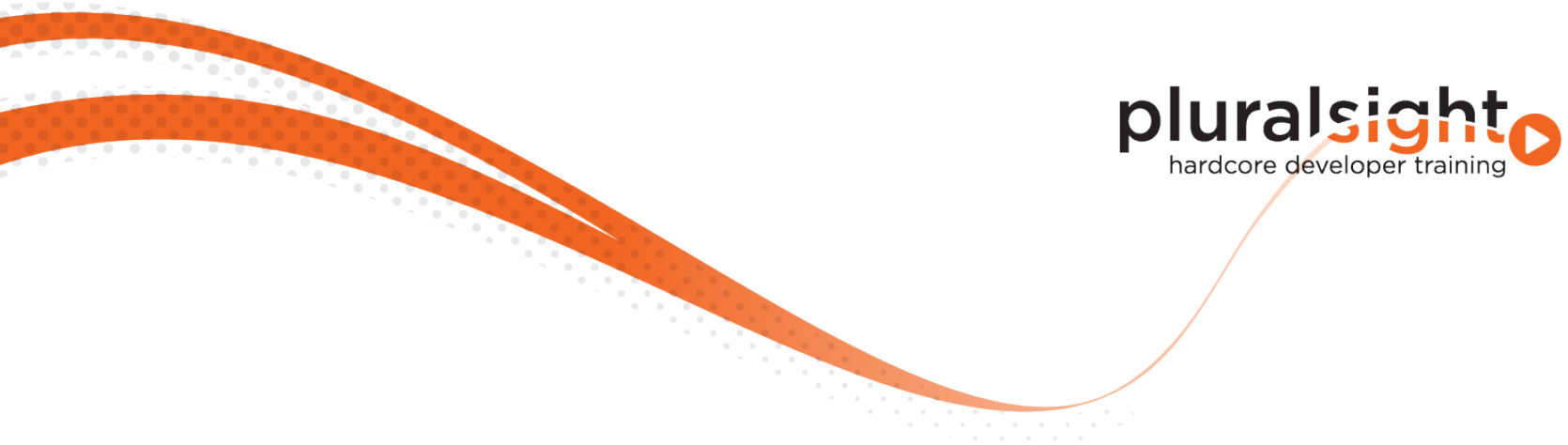
```
delegate int AddDelegate(int a, int b);

static void Main(string[] args){
    AddDelegate ad = (a,b) => a + b;
    int result = ad(1,1); //result = 2
}
```

Handling Empty Parameters

- Delegates that don't accept any parameters can be handled using lambdas:

```
delegate bool LogDelegate();

static void Main(string[] args)
{
    LogDelegate ld = () =>
    {
        UpdateDatabase();
        WriteToEventLog();
        return true;
    };
    bool status = ld();
}
```

Using Action<T>



pluralsight
hardcore developer training

Delegates in .NET

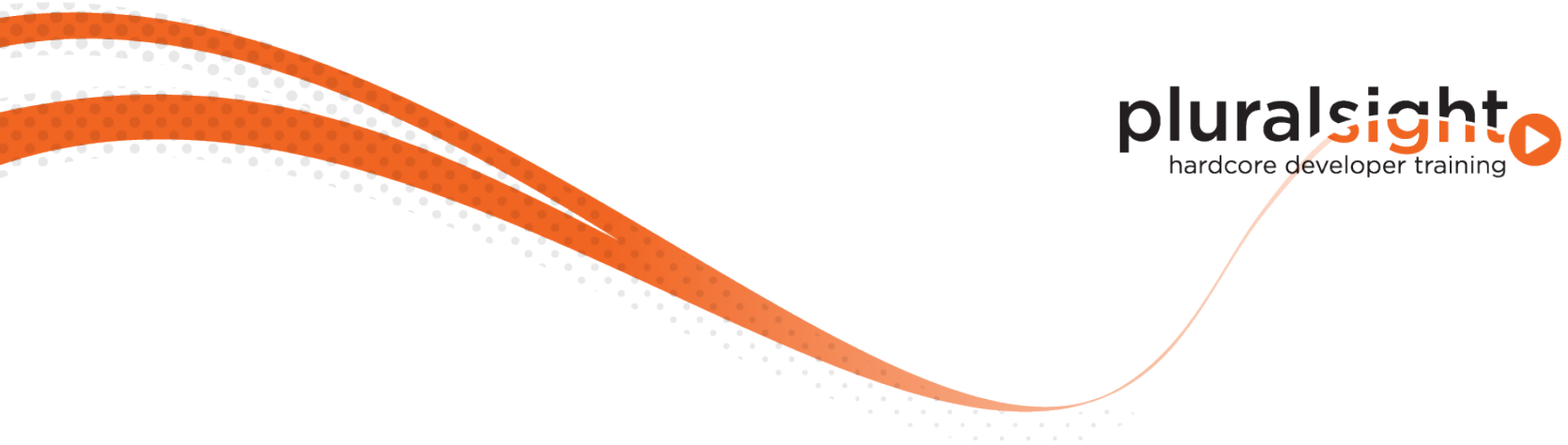
- The .NET framework provides several different delegates that provide flexible options:
  - **Action<T>** - Accepts a single parameter and returns no value
  - **Func<T,TResult>** - Accepts a single parameter and return a value of type TResult

Using Action<T>

- Action<T> can be used to call a method that accepts a single parameter of type T:

```
public static void Main(string[] args)
{
    Action<string> messageTarget;
    if (args.Length > 1) messageTarget = ShowWindowsMessage;
    else messageTarget = Console.WriteLine;
    messageTarget("Invoking Action!");
}

private static void ShowWindowsMessage(string message)
{
    MessageBox.Show(message);
}
```

Using Func<T,TResult>



pluralsight
hardcore developer training

Using Func<T,TResult>

- **Func<T,TResult>** supports a single parameter (T) and returns a value (TResult):

```
public static void Main(string[] args)
{
    Func<string, bool> logFunc;
    if (args[0] == "EventLog") logFunc = LogToEventLog;
    else logFunc = LogToFile;
    bool status = logFunc("Log Message");
}

private static bool LogToEventLog(string message) { /* log */ }
private static bool LogToFile(string message) { /*log */ }
```

Summary

- Lambdas provide a way to define inline methods using a concise syntax

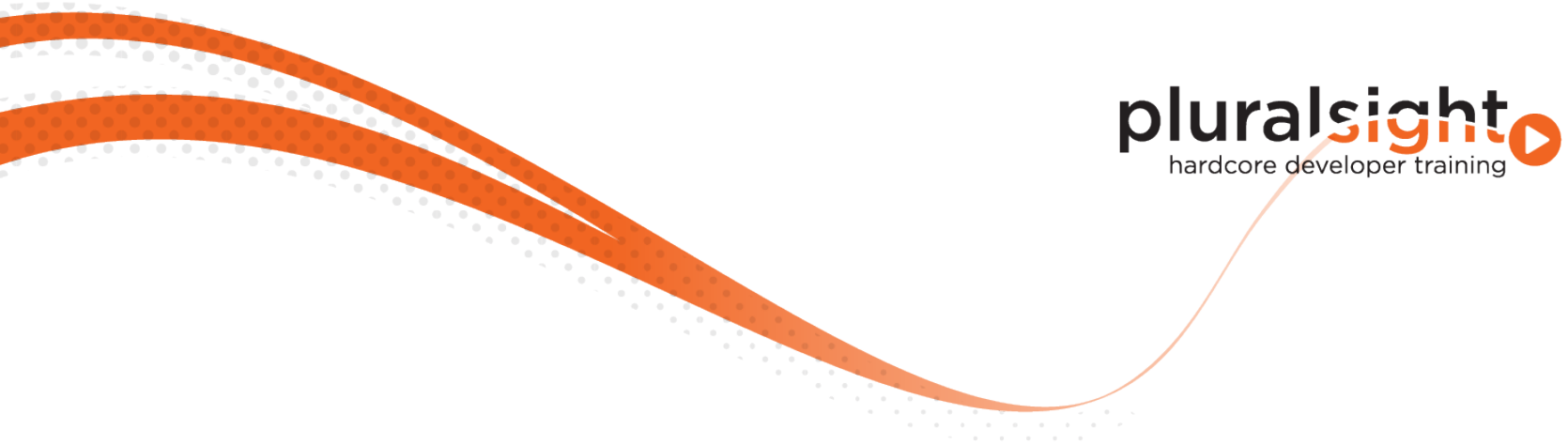
- The .NET Framework provides several built-in delegate types such as:
  - Action<T>
  - Func<T, TResult>

Events and Delegates in Action

Dan Wahlin

Twitter: @DanWahlin

Blog: http://weblogs.asp.net/dwahlin



pluralsight
hardcore developer training

Communicating Between Components

Asynchronous Delegates

Using BackgroundWorker Events

The Role of Delegates with Threads

Putting it All Together

Summary

- Events and Delegates can be used in many scenarios:
  - Communication between objects using events
  - To start asynchronous processes
  - To handle BackgroundWorker functionality
  - To start threads
  - Many more!