A STUDY AND TOOLKIT FOR ASYNCHRONOUS PROGRAMMING IN C#

Semih Okur

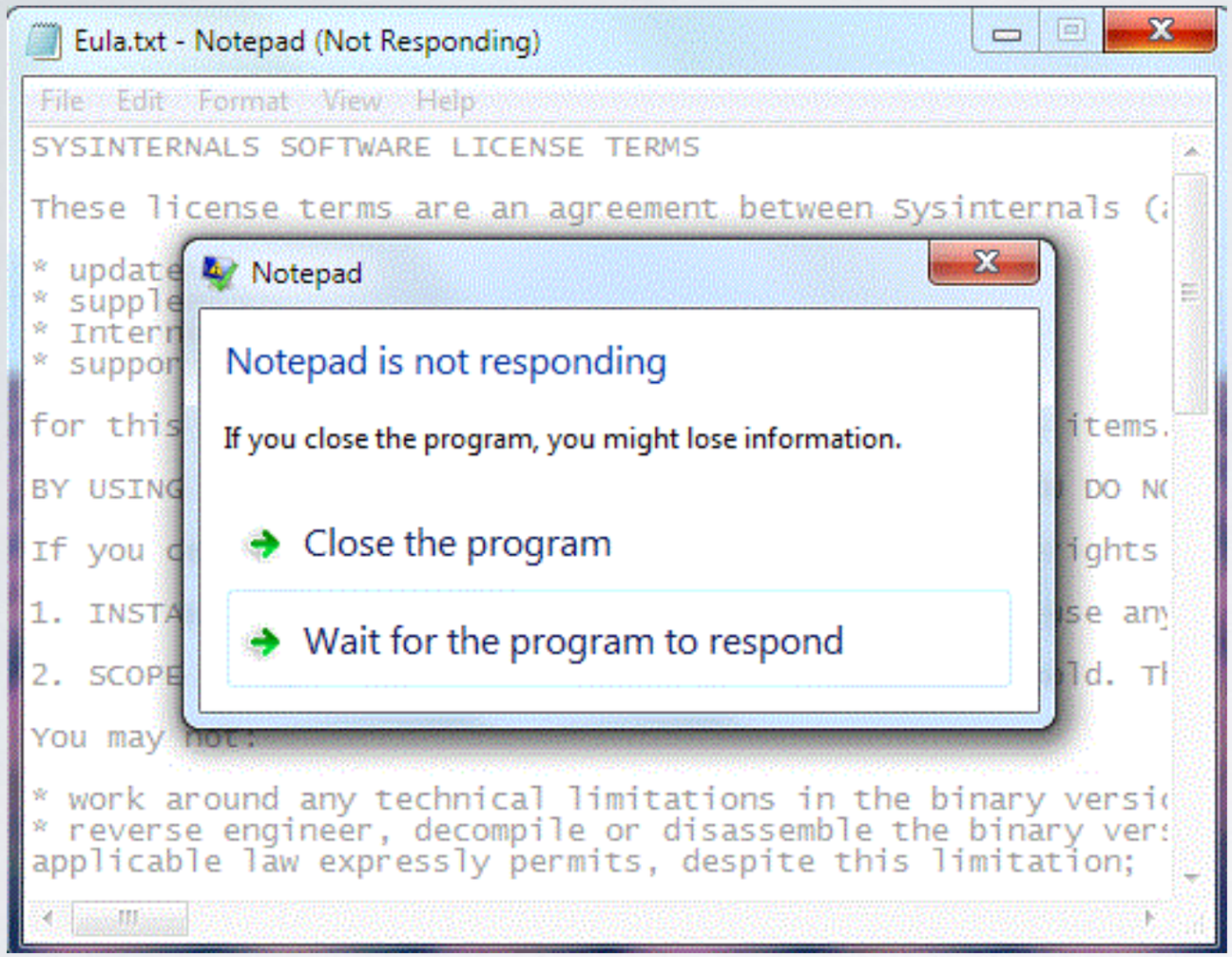
David Hartveld

Danny Dig

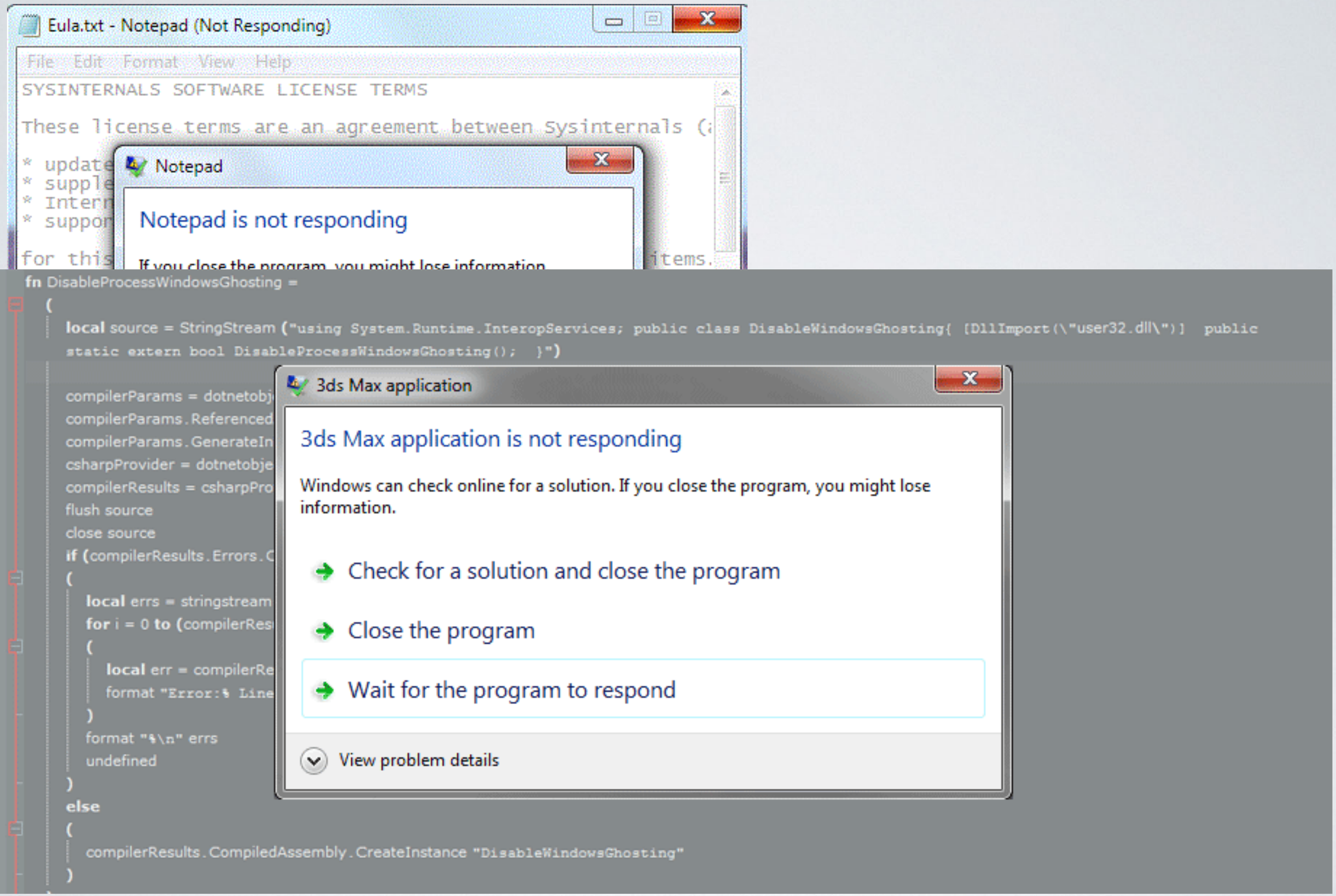
Arie van Deursen

Unresponsive UI Is Frustrating!

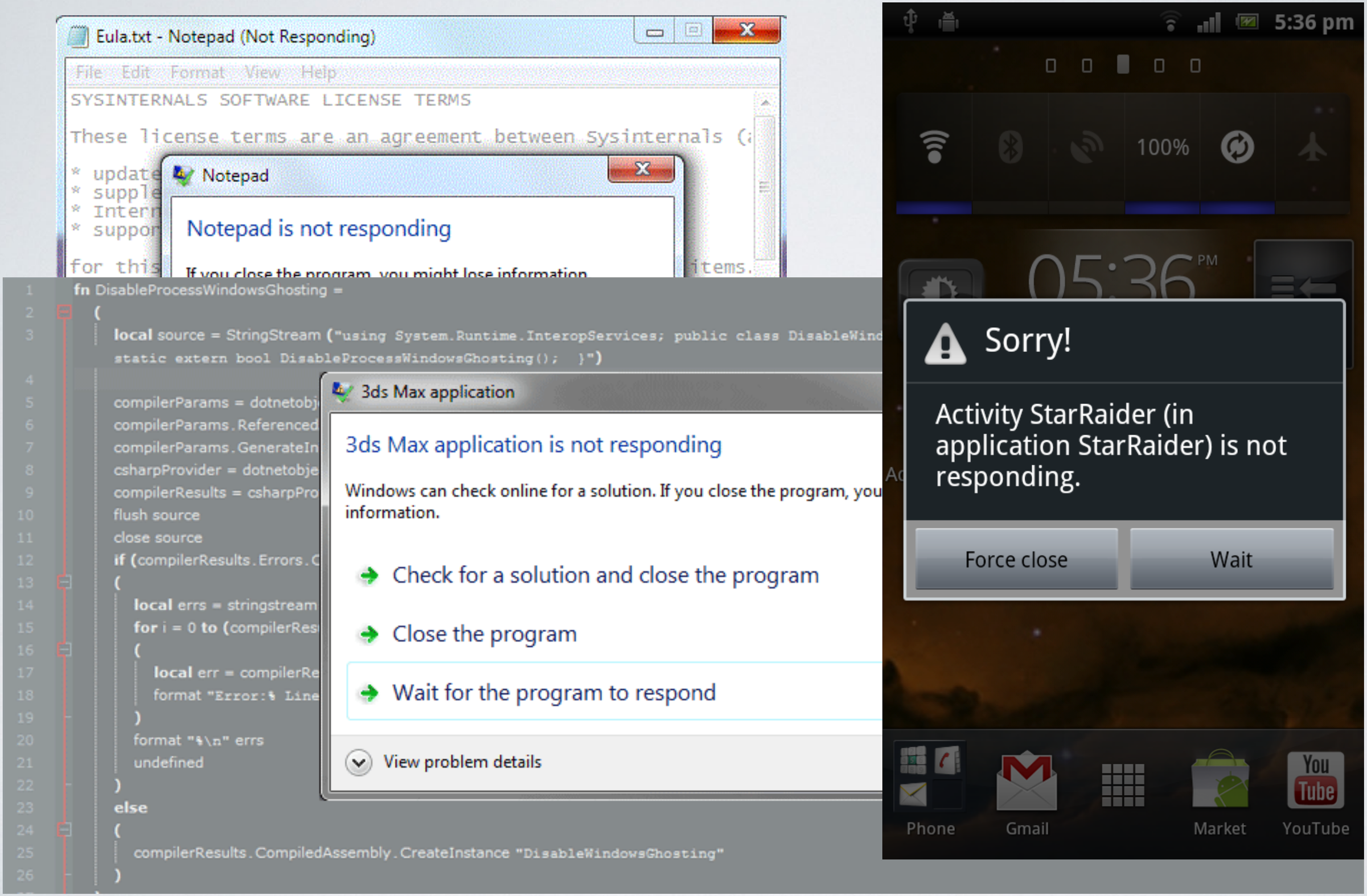
Unresponsive UI Is Frustrating!



Unresponsive UI Is Frustrating!



Unresponsive UI Is Frustrating!



Why Are Apps Unresponsive?

UI thread is running as an event loop

If processing one event takes too much time, the event loop does not proceed

Why Are Apps Unresponsive?

UI thread is running as an event loop

If processing one event takes too much time, the event loop does not proceed

Frozen UI

Why Are Apps Unresponsive?

UI thread is running as an event loop

If processing one event takes too much time, the event loop does not proceed

Frozen UI

(i) Long running CPU-bound (ii) Blocking I/O operations

Our Approach

Our Approach

(P1) No prior knowledge about (mis)use of asynchronous constructs => wrong assumptions

Thus, we conducted a formative study

Our Approach

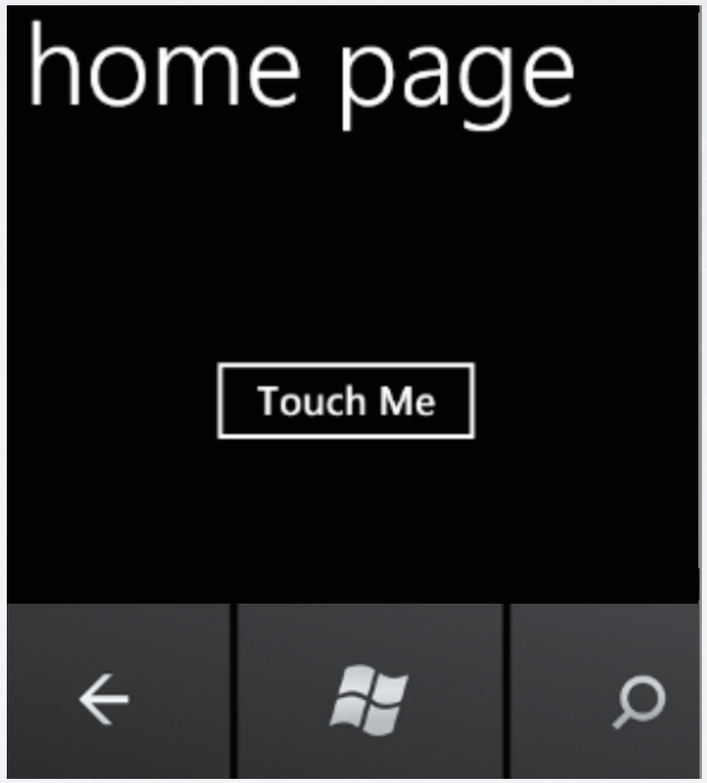
(P1) No prior knowledge about (mis)use of asynchronous constructs => wrong assumptions

Thus, we conducted a formative study

(P2) Asynchronous programming is hard to use

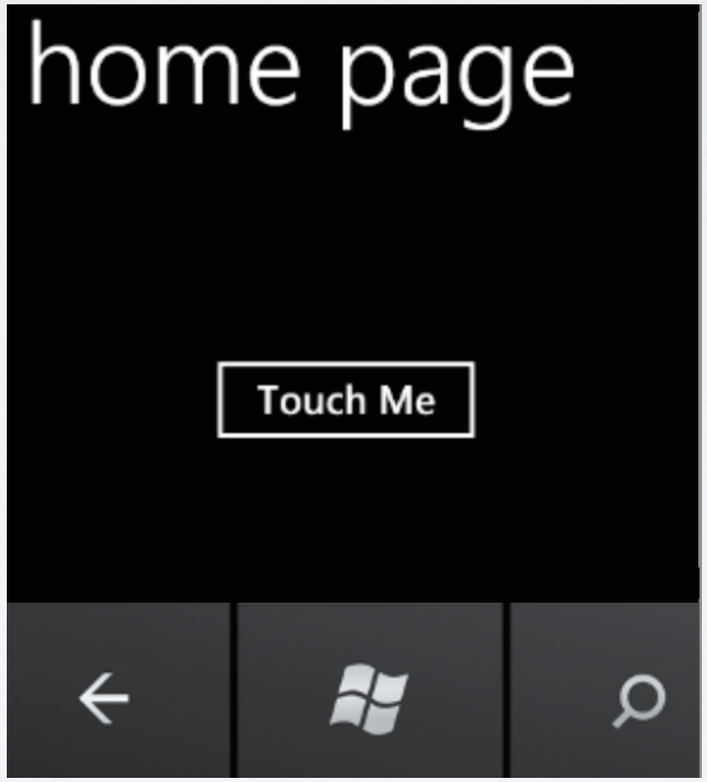
Thus, we provide (1) refactoring and (2) bug fixing tools

Example - Synchronous



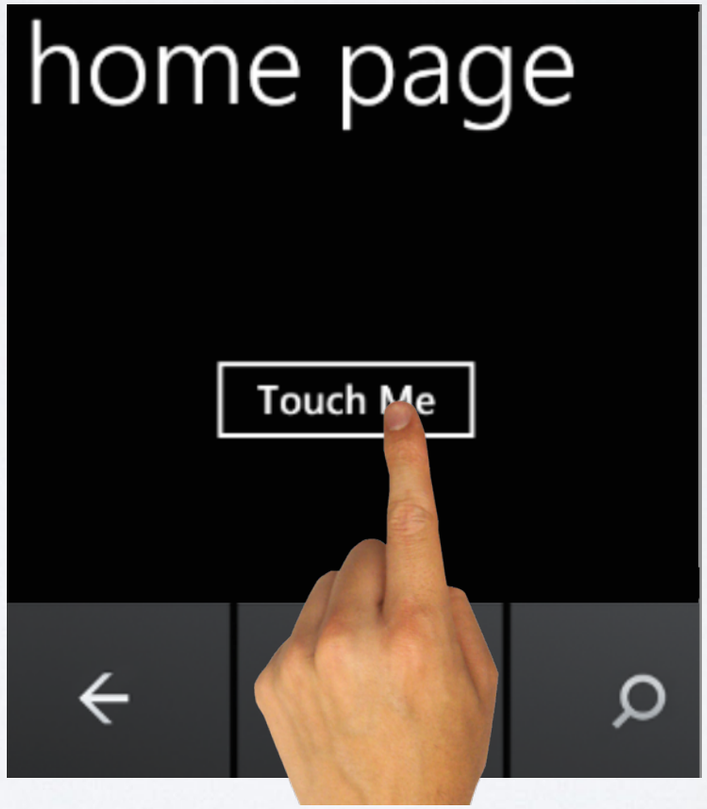
Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

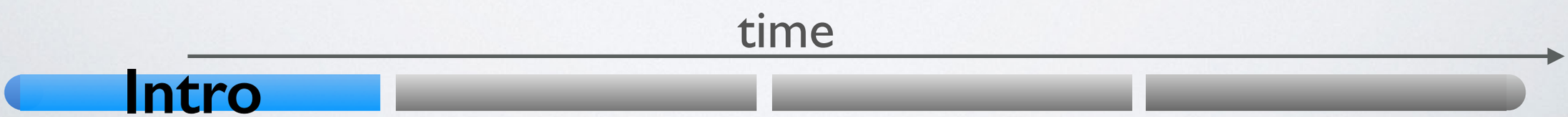
Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

UI thread

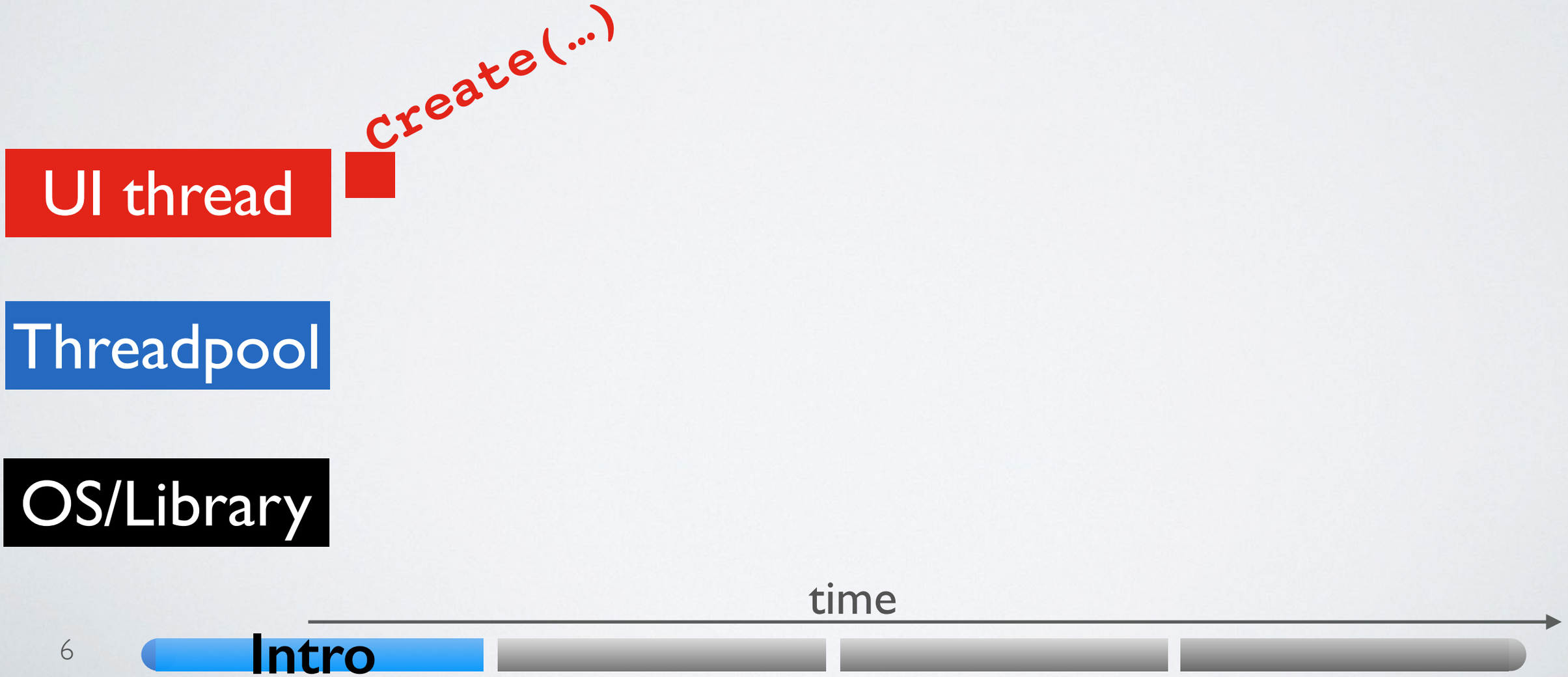
Threadpool

OS/Library



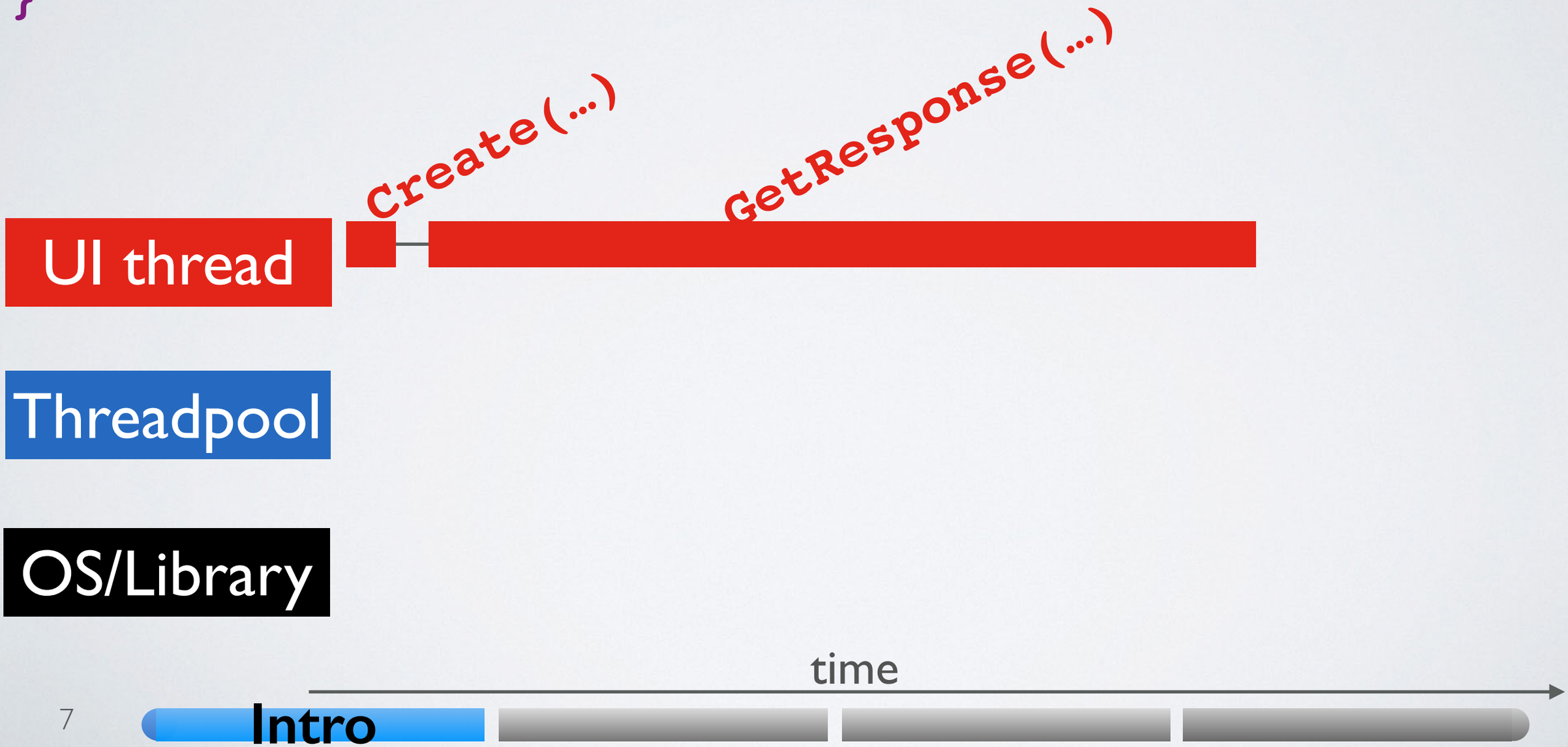
Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



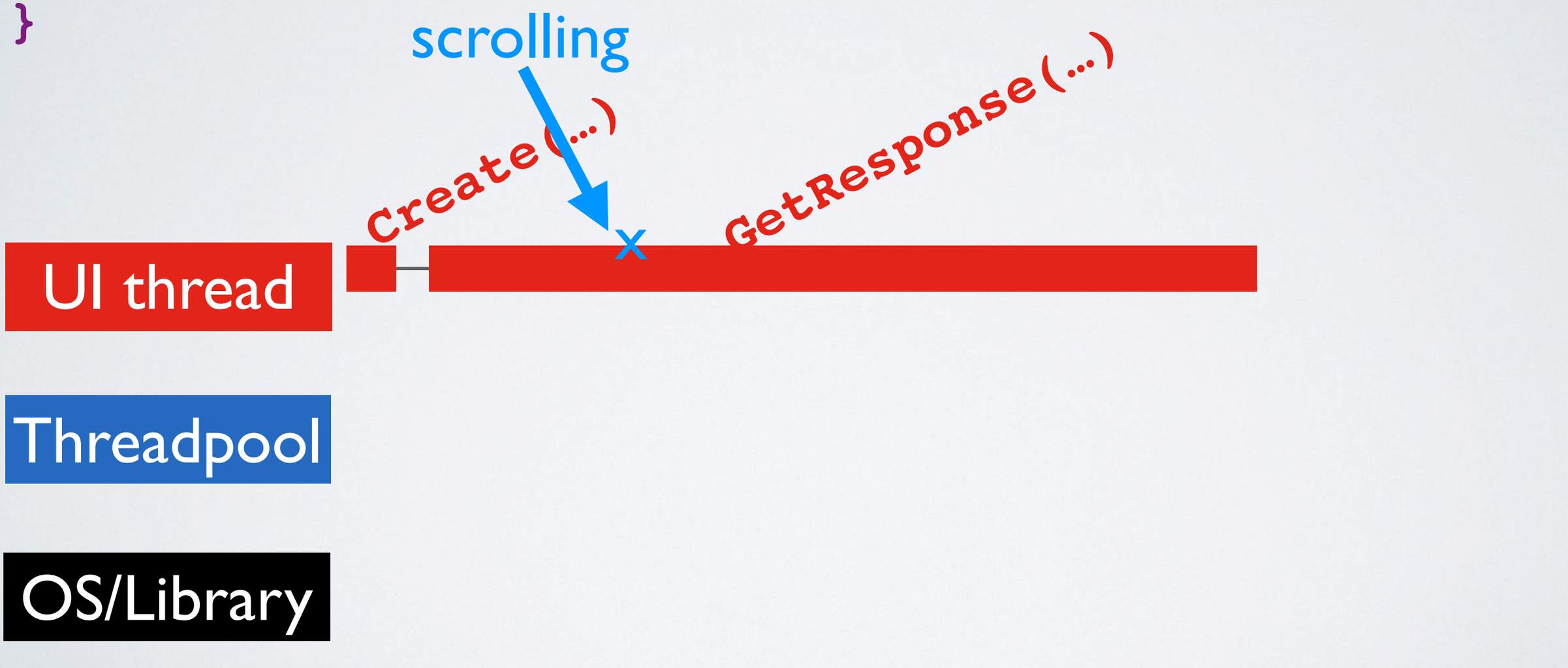
Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



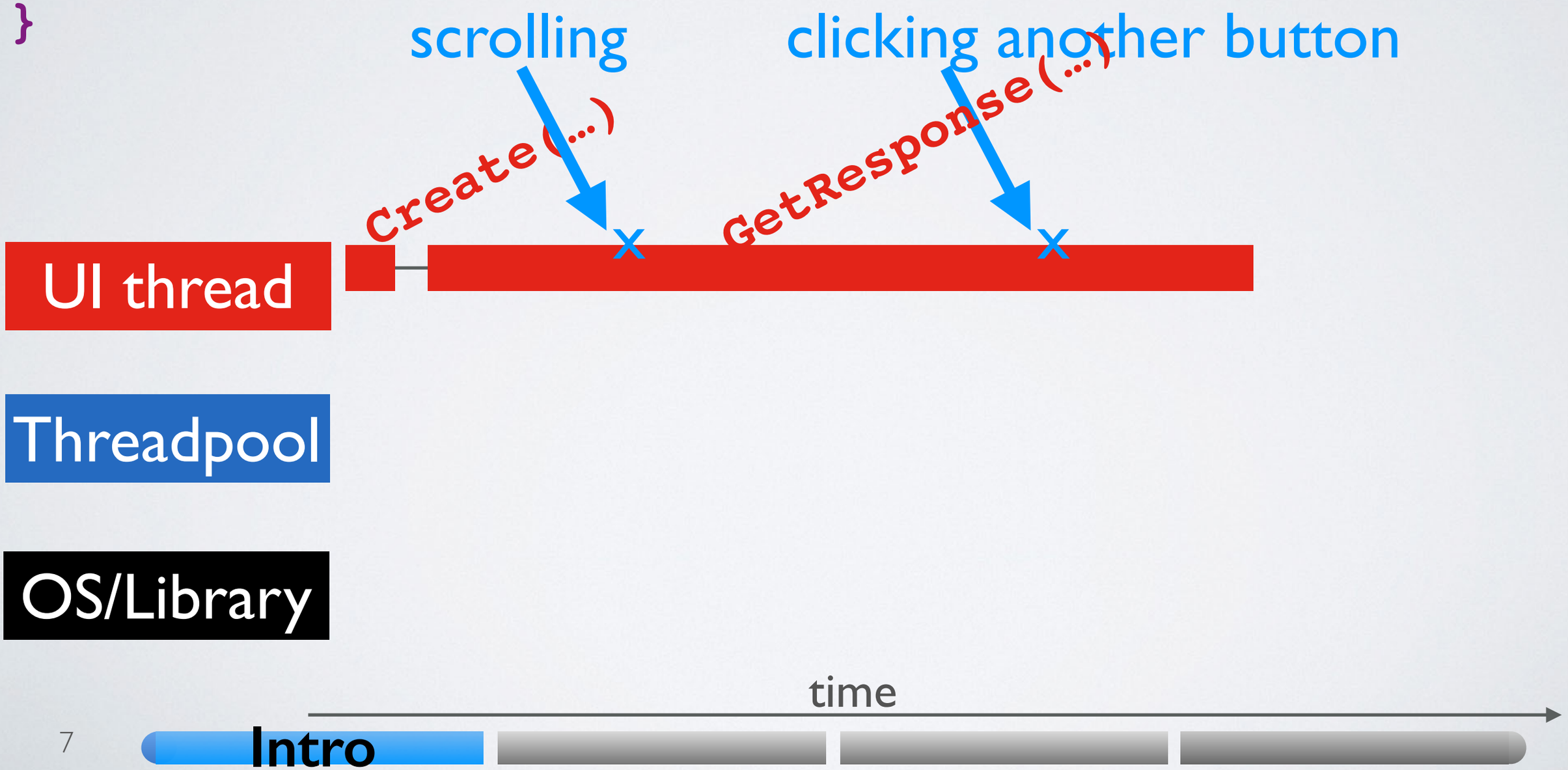
Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



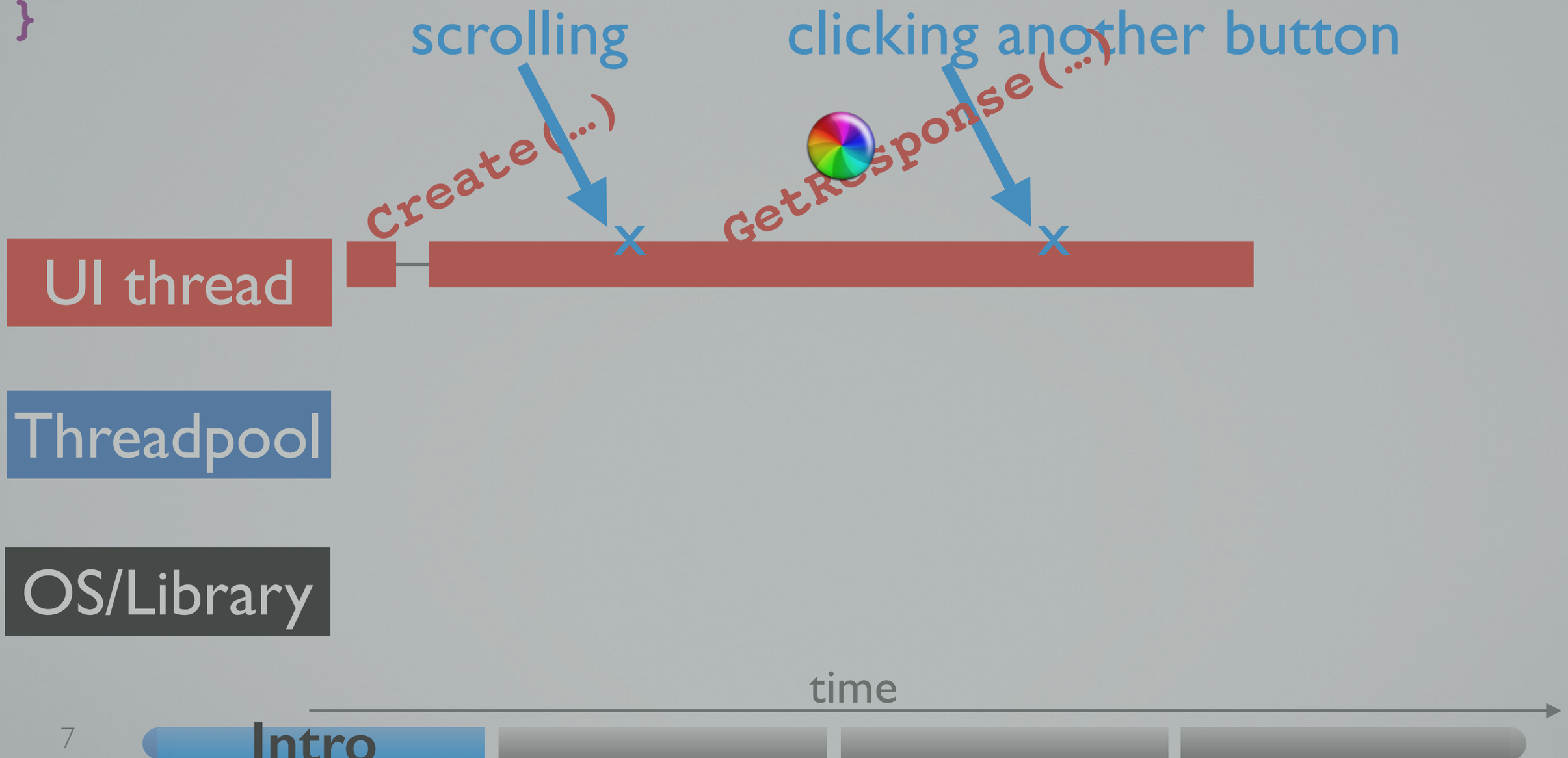
Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

time

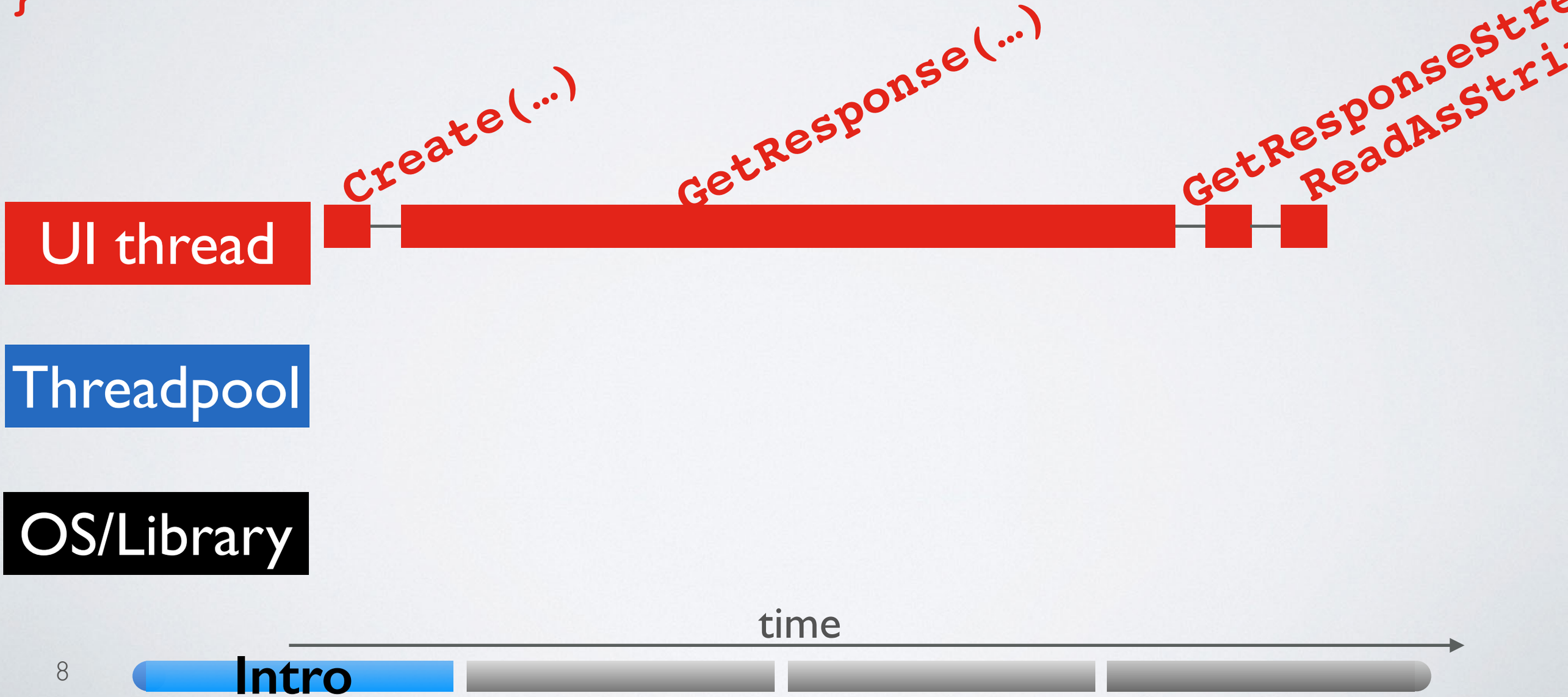
Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



Example - Synchronous

```
private void Button_Click(object sender, EventArgs e){
   var request=WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



Callback-Based Asynchronous

Asynchronous APIs for I/O operations rely on callbacks.

Callback-Based Asynchronous

Asynchronous APIs for I/O operations rely on callbacks.

For each asynchronous operation, there are two methods:

`BeginXyz, EndXyz`

Callback-Based Asynchronous

Asynchronous APIs for I/O operations rely on callbacks.

For each asynchronous operation, there are two methods:

`BeginXyz, EndXyz`

To access web:

`BeginGetResponse & EndGetResponse`

Callback-Based Asynchronous

Asynchronous APIs for I/O operations rely on callbacks.

For each asynchronous operation, there are two methods:

`BeginXyz, EndXyz`

To access web:

`BeginGetResponse & EndGetResponse`

`BeginXYZ(callback,…)`initiate the asynchronous operation

Callback-Based Asynchronous

Asynchronous APIs for I/O operations rely on callbacks.

For each asynchronous operation, there are two methods:

`BeginXyz, EndXyz`

To access web:

`BeginGetResponse & EndGetResponse`

`BeginXYZ(callback,…)` initiate the asynchronous operation

`EndXYZ(…)` get the result of the asynchronous operation

Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```

UI thread

Threadpool

OS/Library

Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```

UI thread

Threadpool

OS/Library

Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```

UI thread

Threadpool

OS/Library

Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```

Create(…)

UI thread

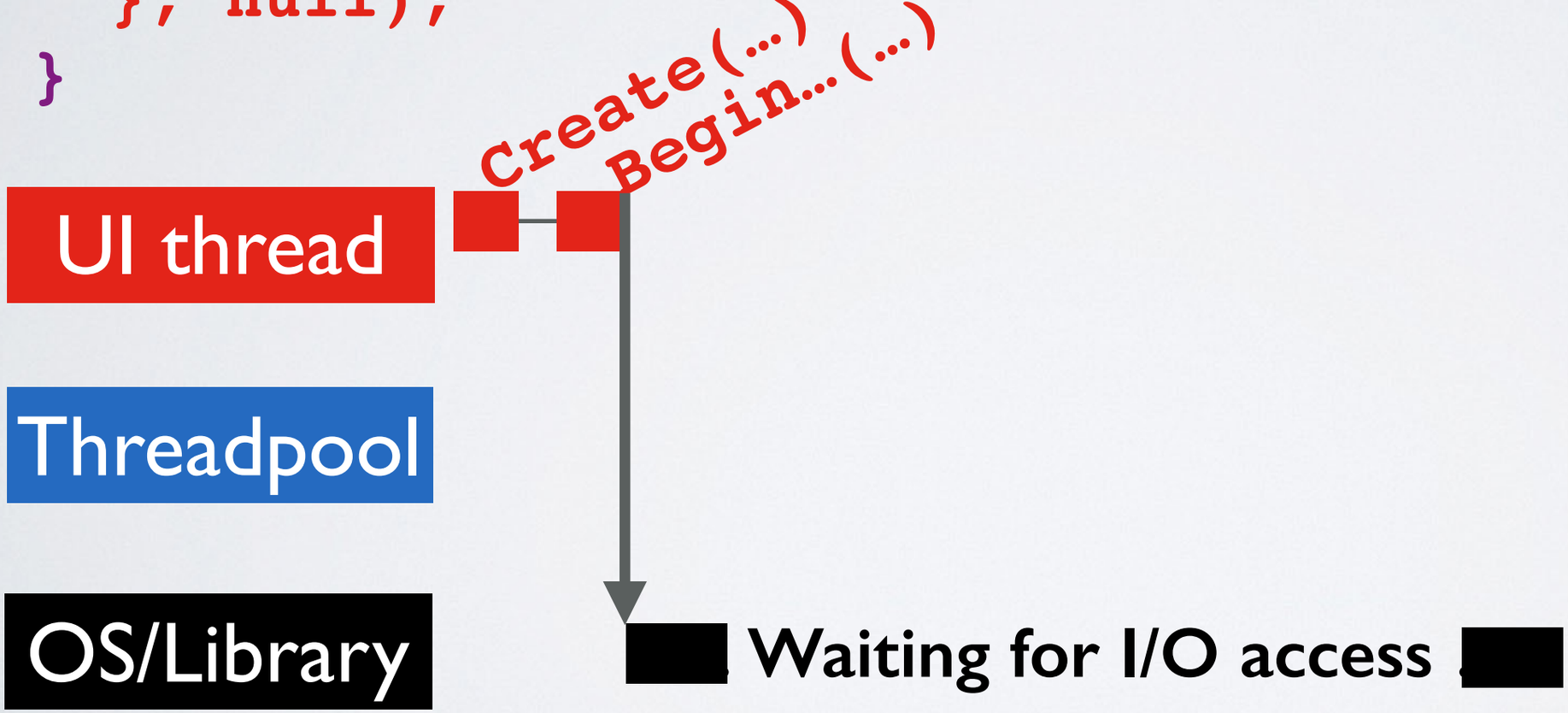
Threadpool

OS/Library

time

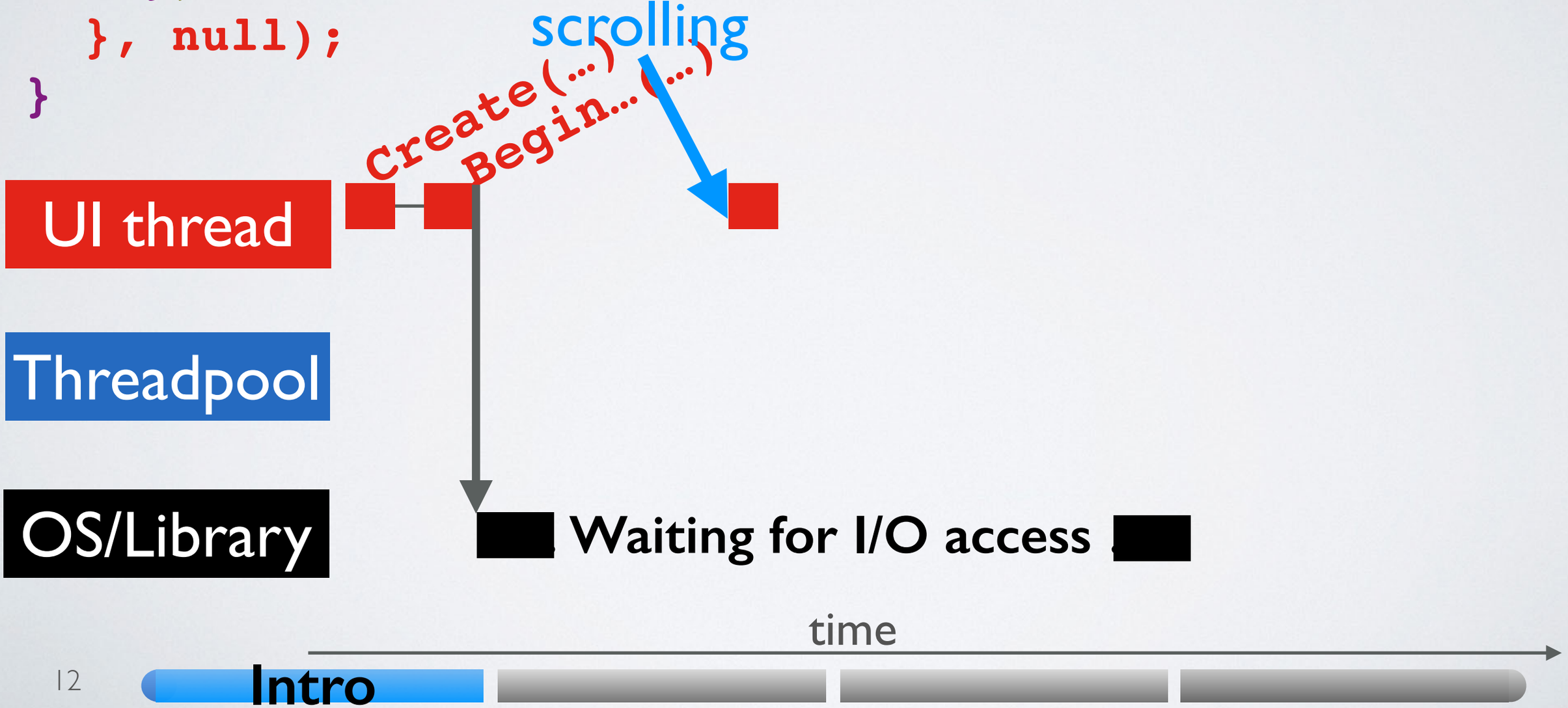
Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```



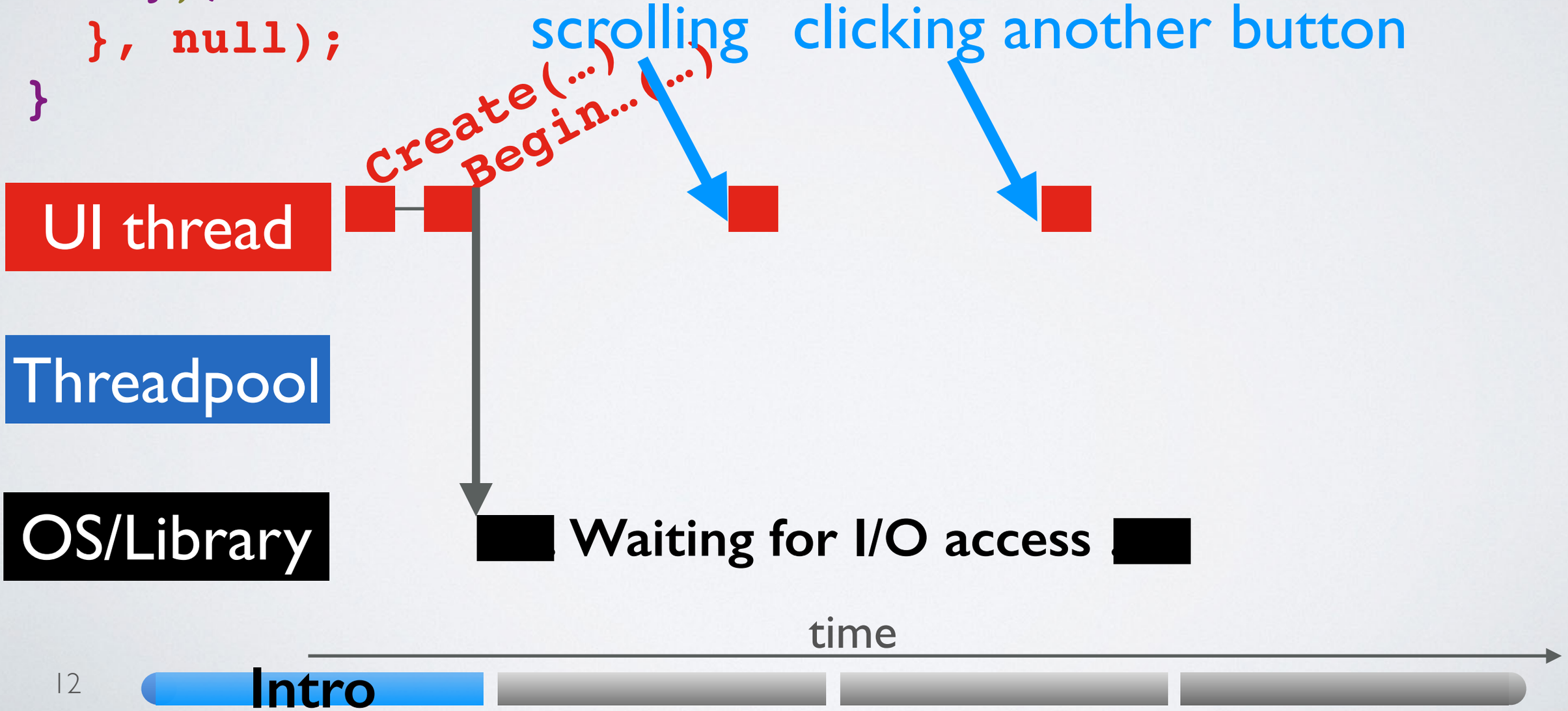
Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```



Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```

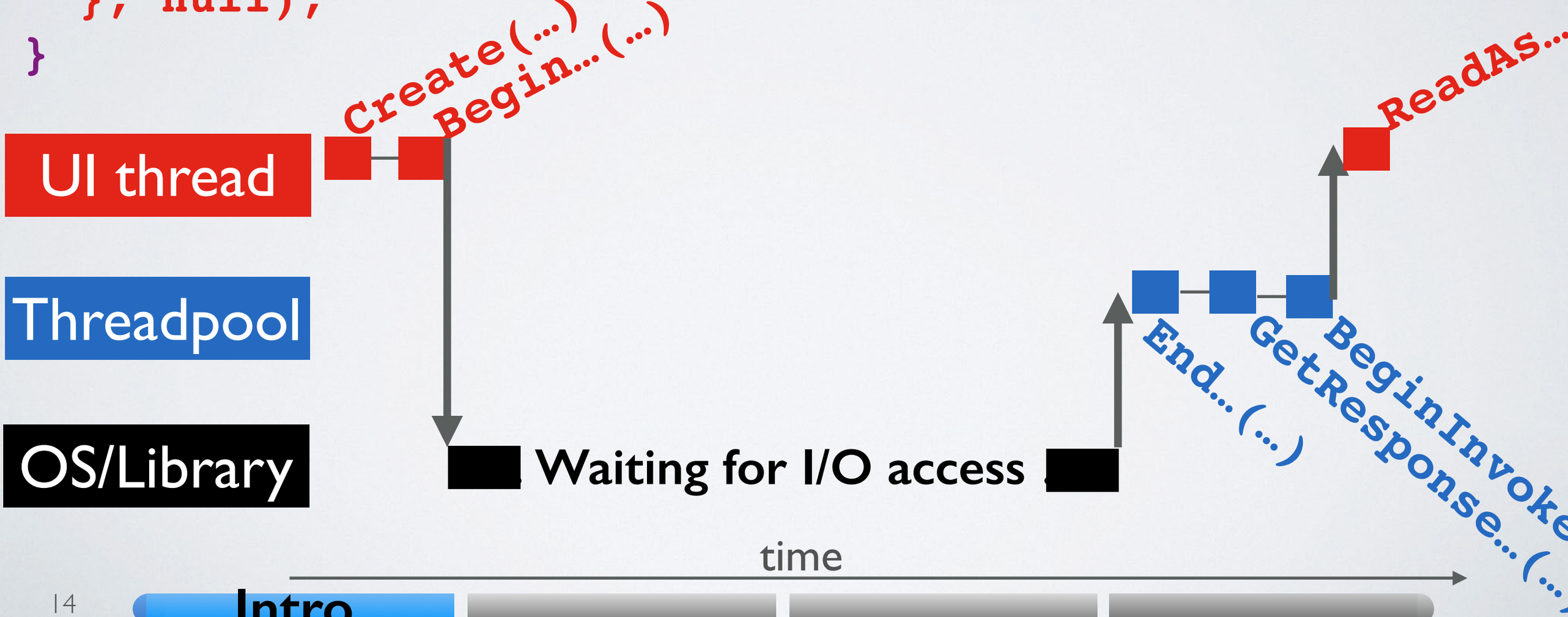


Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```

Example - Callback-Based Asynchronous

```
private void Button_Click(object sender, EventArgs e){
  var request=WebRequest.Create("icse-conferences.org");
  request.BeginGetResponse((asyncResult)=> {
    var response = request.EndGetResponse(asyncResult);
    var stream = response.GetResponseStream();
    Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
    });
  }, null);
}
```



Callback Hell!

```
private void SnapAndPost ()
{
    Busy = true;
    UpdateUIStatus ("Taking a picture");
    var picker = new Xamarin.Media.MediaPicker ();
    var picTask = picker.TakePhotoAsync (new Xamarin.Media.StoreCameraMediaOptions ());
    picTask.ContinueWith ((picRetTask) => {
        InvokeOnMainThread (() => {
            if (picRetTask.IsCanceled) {
                Busy = false;
                UpdateUIStatus ("Canceled");
            } else {
                var tagsCtrl = new GetTagsUIViewController (picRetTask.Result.GetStream ());
                PresentViewController (tagsCtrl, true, () => {
                    UpdateUIStatus ("Submitting picture to server");
                    var uploadTask = new Task (() => {
                        return PostPicToService (picRetTask.Result.GetStream (), tagsCtrl.Tags);
                    });
                    uploadTask.ContinueWith ((uploadRetTask) => {
                        InvokeOnMainThread (() => {
                            Busy = false;
                            UpdateUIStatus (uploadRetTask.Result.Failed ? "Canceled" : "Success");
                        });
                    });
                    uploadTask.Start ();
                });
            }
        });
    });
}
```

Callback Hell!

```
private void SnapAndPost ()
{
    Busy = true;
    UpdateUIStatus ("Taking a picture");
    var picker = new Xamarin.Media.MediaPicker ();
    var picTask = picker.TakePhotoAsync (new Xamarin.Media.StoreCameraMediaOptions ());
    picTask.ContinueWith ((picRetTask) => {
        InvokeOnMainThread (() => {
            if (picRetTask.IsCanceled) {
                Busy = false;
                UpdateUIStatus ("Canceled");
            } else {
                var tagsCtrl = new GetTagsUIViewController (picRetTask.Result.GetStream ());
                PresentViewController (tagsCtrl, true, () => {
                    UpdateUIStatus ("Submitting picture to server");
                    var uploadTask = new Task (() => {
                        return PostPicToService (picRetTask.Result.GetStream (), tagsCtrl.Tags);
                    });
                    uploadTask.ContinueWith ((uploadRetTask) => {
                        InvokeOnMainThread (() => {
                            Busy = false;
                            UpdateUIStatus (uploadRetTask.Result.Failed ? "Canceled" : "Success");
                        });
                    });
                    uploadTask.Start ();
                });
            }
        });
    });
}
```

- Invert the control flow
- Obfuscate the intent of the original synchronous code
- Don’t scale to large programs

Async/Await Keywords

Special language constructs introduced in C# 5 (2012)

Make asynchronous programming a first-class citizen!

Async/Await Keywords

Special language constructs introduced in C# 5 (2012)

Make asynchronous programming a first-class citizen!

Possible to express efficient asynchronous code in a familiar direct style

`async`: method identifier

`await`: used to define pausing points

To access web: `<Task> GetResponseAsync(…)`

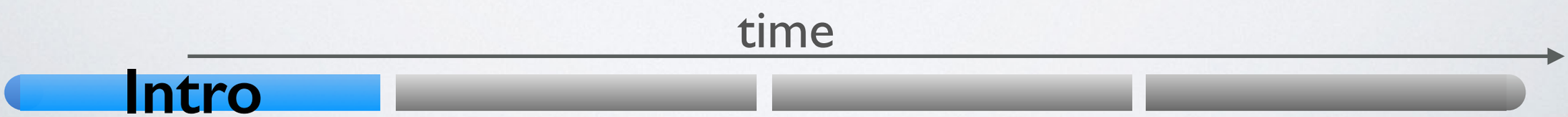
Example - Pause'n'play With Async/Await

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

UI thread

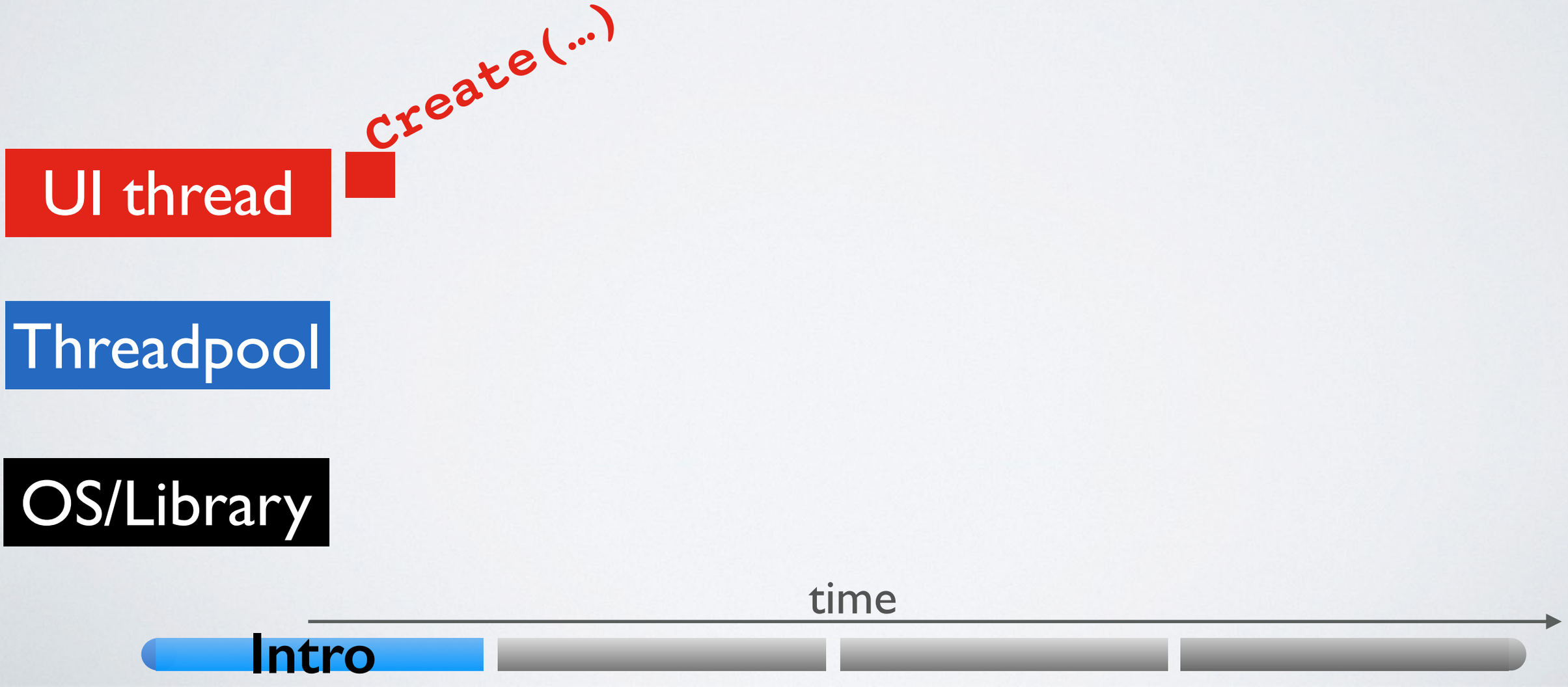
Threadpool

OS/Library



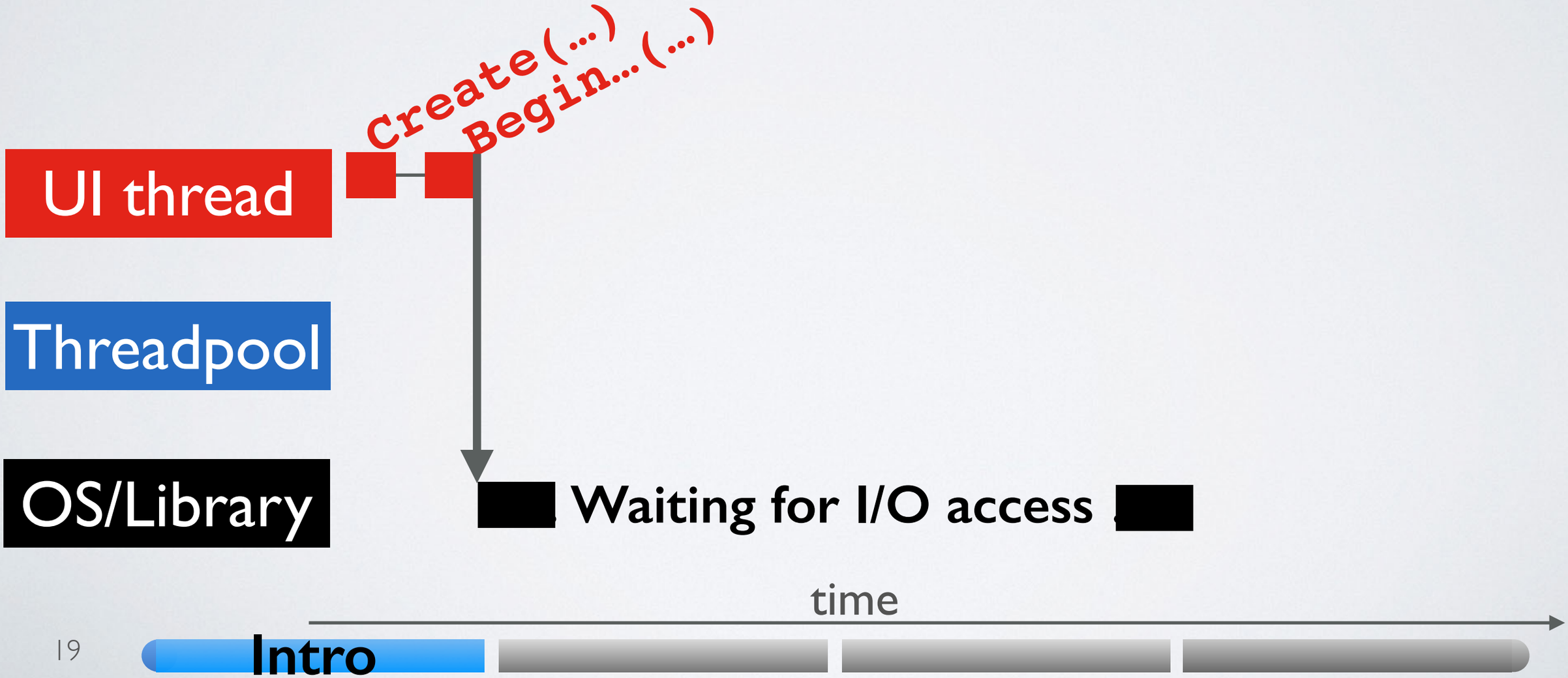
Example - Pause'n'play With Async/Await

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



Example - Pause'n'play With Async/Await

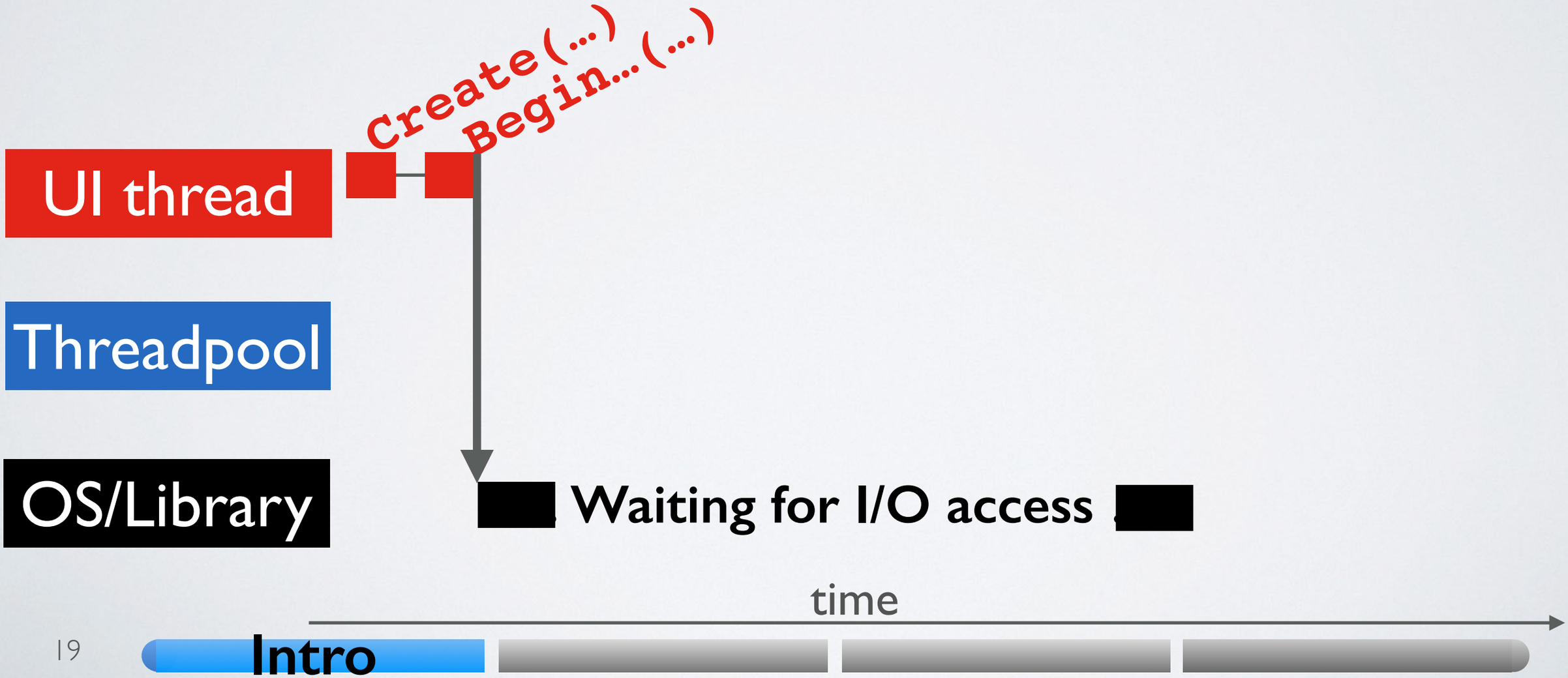
```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```



Example - Pause'n'play With Async/Await

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

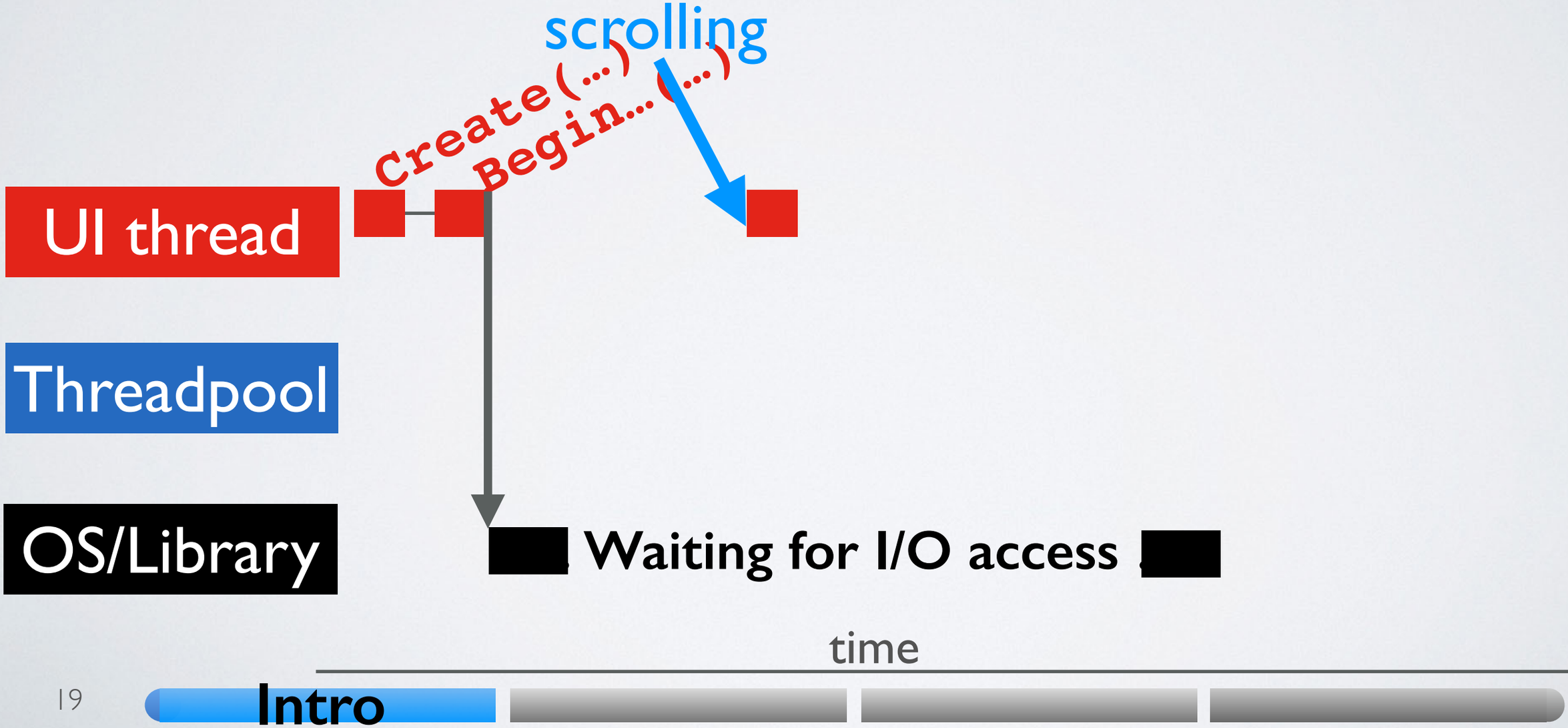
Method is paused!



Example - Pause'n'play With Async/Await

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

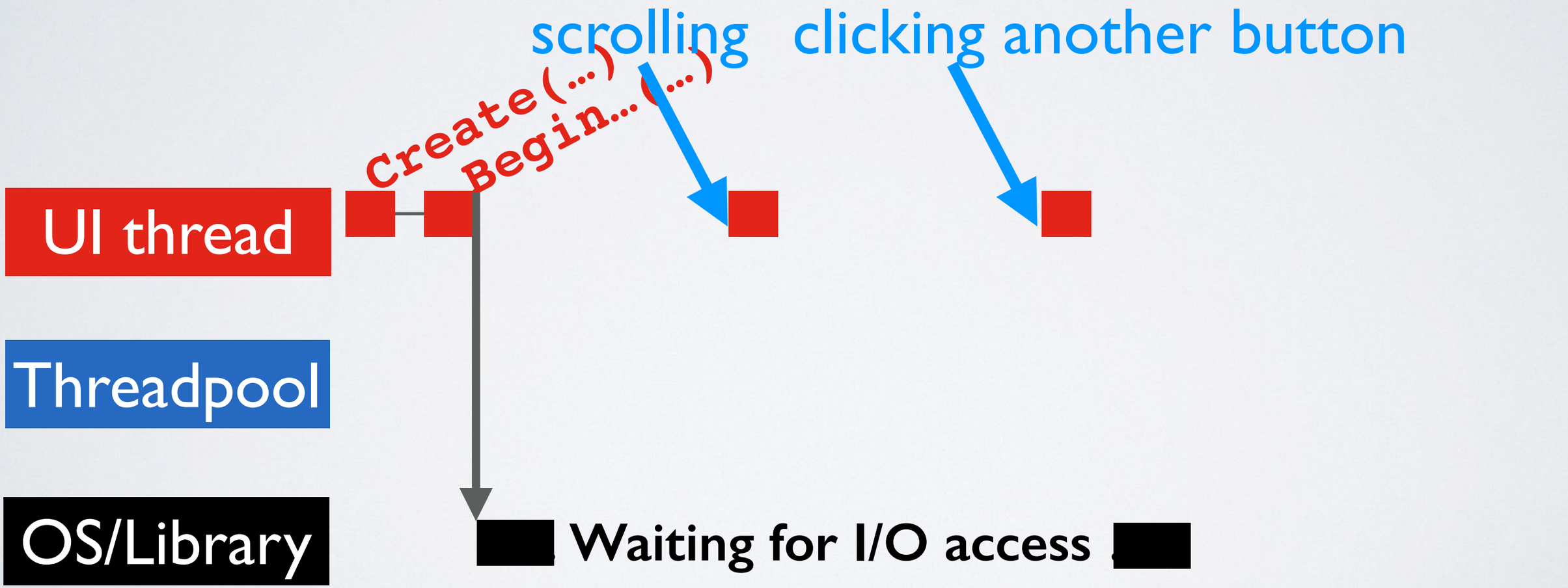
Method is paused!



Example - Pause'n'play With Async/Await

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

Method is paused!



Example - Pause'n'play With Async/Await

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

Synchronous Code

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   var response = request.GetResponse();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

Asynchronous Async/Await Code

```
private async void Button_Click(object sender, EventArgs e)
{
   var request = WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   textBox.Text = stream.ReadAsString();
}
```

Our Study

(i) We do not know how developers use asynchronous programming in practice.

(ii) Asynchronous programming is hard to use and developers lack tool support for it.

Our Study

(i) We do not know how developers use asynchronous programming in practice.

(ii) Asynchronous programming is hard to use and developers lack tool support for it.

Conducted a first large-scale formative study about asynchronous programming to answer two RQs:

Our Study

(i) We do not know how developers use asynchronous programming in practice.

(ii) Asynchronous programming is hard to use and developers lack tool support for it.

Conducted a first large-scale formative study about asynchronous programming to answer two RQs:

Q1) How do developers use asynchronous programming?

Built `Asyncifier`, a refactoring tool.

Our Study

(i) We do not know how developers use asynchronous programming in practice.

(ii) Asynchronous programming is hard to use and developers lack tool support for it.

Conducted a first large-scale formative study about asynchronous programming to answer two RQs:

Q1) How do developers use asynchronous programming?

Built `Asyncifier`, a refactoring tool.

Q2) Do developers misuse async/await?

Built `Corrector`, a bug fixing tool.

Code Corpus

Analyzed 1378 open source Windows Phone apps, comprising 12M SLOC, produced by 3376 developers.

Code Corpus

Analyzed 1378 open source Windows Phone apps, comprising 12M SLOC, produced by 3376 developers.

Built a static analysis tool with Microsoft's Roslyn (C# compiler apis: syntax, symbol, binding)

Code Corpus

Analyzed 1378 open source Windows Phone apps, comprising 12M SLOC, produced by 3376 developers.

Built a static analysis tool with Microsoft's Roslyn (C# compiler apis: syntax, symbol, binding)

A detailed information about creating code corpus is available in the paper

RQ1) How Do Developers Use Asynchronous Programming?

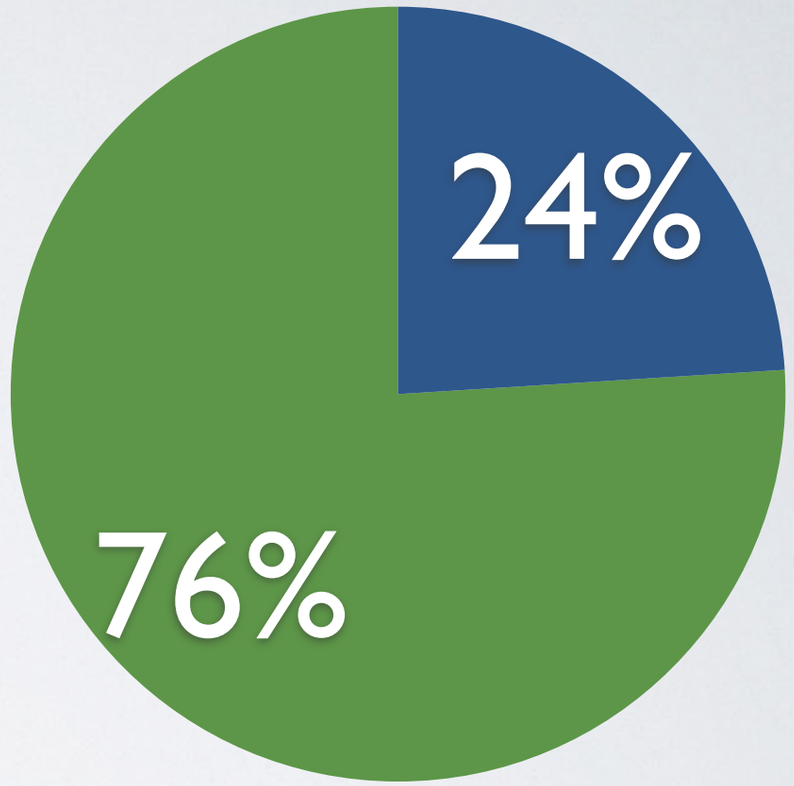
Analyzed I/O bound operations:

- Callback-based async idioms
- Task-based new async idioms (can be used with async/await)

RQ1) How Do Developers Use Asynchronous Programming?

Analyzed I/O bound operations:

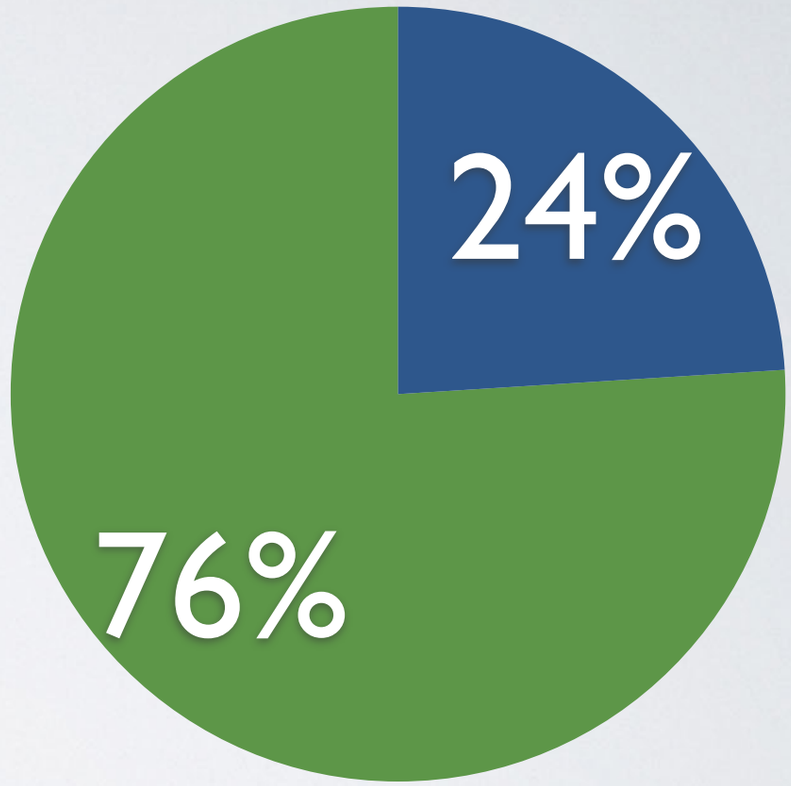
- Callback-based async idioms
- Task-based new async idioms (can be used with async/await)



RQ1) How Do Developers Use Asynchronous Programming?

Analyzed I/O bound operations:

- Callback-based async idioms
- Task-based new async idioms (can be used with async/await)



Much more about CPU-bound and I/O-bound operations in the paper

Developers Need a Refactoring Tool: ASYNCIFIER

Developers heavily use callback-based operations

Developers Need a Refactoring Tool: ASYNCIFIER

Developers heavily use callback-based operations

Microsoft officially no longer recommends these operations

Developers Need a Refactoring Tool: ASYNCIFIER

Developers heavily use callback-based operations

Microsoft officially no longer recommends these operations

Developers Need a Refactoring Tool: ASYNCIFIER

Developers heavily use callback-based operations

Microsoft officially no longer recommends these operations

Refactoring from callbacks to async/await is non-trivial

Developers Need a Refactoring Tool: ASYNCIFIER

Developers heavily use callback-based operations

Microsoft officially no longer recommends these operations

Refactoring from callbacks to async/await is non-trivial

We built ASYNCIFIER:
a refactoring tool to upgrade callback code to async/await

It Is Not One-To-One Mapping Between Old And New Operations

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```

It Is Not One-To-One Mapping Between Old And New Operations

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```

`BeginXYZ(…)` initiate the asynchronous operation

It Is Not One-To-One Mapping Between Old And New Operations

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```

`BeginXYZ(…)` initiate the asynchronous operation

`EndXYZ(…)` get the result of the asynchronous operation

It Is Not One-To-One Mapping Between Old And New Operations

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```

`BeginXYZ(…)` initiate the asynchronous operation

`EndXYZ(…)` get the result of the asynchronous operation

Asyncifier

It Is Not One-To-One Mapping Between Old And New Operations

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```

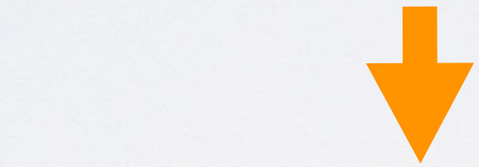
`BeginXYZ(…)` initiate the asynchronous operation

`EndXYZ(…)` get the result of the asynchronous operation

`XYZAsync(…)` initiate & get the result of the operation

(1) Inlining Lambda Statements

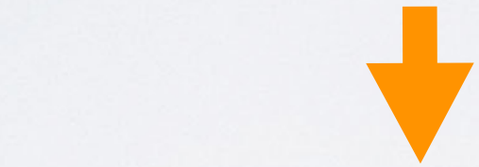
```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```



Asyncifier

(1) Inlining Lambda Statements

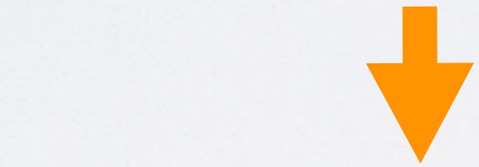
```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```



Asyncifier

(1) Inlining Lambda Statements

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      var response = request.EndGetResponse(asyncResult);
      var stream = response.GetResponseStream();
      Dispatcher.BeginInvoke(() => {
         textBox.Text = stream.ReadAsString();
      });
   }, null);
}
```



```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse();
   var response = request.EndGetResponse(asyncResult);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```


Asyncifier

(2) Introduce Async/Await Method

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse();
   var response = request.EndGetResponse(asyncResult);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

Asyncifier

(2) Introduce Async/Await Method

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse();
   var response = request.EndGetResponse(asyncResult);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```


Asyncifier

(2) Introduce Async/Await Method

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse();
   var response = request.EndGetResponse(asyncResult);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

```
private async void Button_Click(…){
   var request=WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```


Asyncifier

(3) Preserve Original Behavior

```
private async void Button_Click(…){
   var request = WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

Asyncifier

(3) Preserve Original Behavior

```
private async void Button_Click(…){
   var request = WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

```
private async void Button_Click(…){
   var request = WebRequest.Create("icse-conferences.org");
   var response = await
request.GetResponseAsync().ConfigureAwait(false);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

(3) Preserve Original Behavior

```
private async void Button_Click(…){
   var request = WebRequest.Create("icse-conferences.org");
   var response = await request.GetResponseAsync();
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

```
private async void Button_Click(…){
   var request = WebRequest.Create("icse-conferences.org");
   var response = await
request.GetResponseAsync().ConfigureAwait(false);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```


Asyncifier

Challenges

(1) There are many different ways to introduce callbacks:

Challenges

(1) There are many different ways to introduce callbacks:

(2) `EndXyz` is 'hidden' deeper down the call chain

Challenges

(1) There are many different ways to introduce callbacks:

(2) `EndXyz` is 'hidden' deeper down the call chain

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      IntermediateMethod(request, asyncResult);
   }, null);
}
public void IntermediateMethod(...){
   var response = request.EndGetResponse(asyncResult);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

Challenges

(1) There are many different ways to introduce callbacks:

(2) `EndXyz` is 'hidden' deeper down the call chain

```
private void Button_Click(object sender, EventArgs e){
   var request = WebRequest.Create("icse-conferences.org");
   request.BeginGetResponse((asyncResult)=> {
      IntermediateMethod(request, asyncResult);
   }, null);
}
public void IntermediateMethod(...){
   var response = request.EndGetResponse(asyncResult);
   var stream = response.GetResponseStream();
   Dispatcher.BeginInvoke(() => {
      textBox.Text = stream.ReadAsString();
   });
}
```

(3) The exception behavior is different

RQ2) Do Developers Misuse Async?

Misuse: anti-patterns which hurt performance and might cause serious problems like deadlocks.

RQ2) Do Developers Misuse Async?

Misuse: anti-patterns which hurt performance and might cause serious problems like deadlocks.

167 apps that use async/await: 2383 async methods

RQ2) Do Developers Misuse Async?

Misuse: anti-patterns which hurt performance and might cause serious problems like deadlocks.

167 apps that use async/await: 2383 async methods

i. 14% of methods that use async/await keywords do this unnecessarily.

RQ2) Do Developers Misuse Async?

Misuse: anti-patterns which hurt performance and might cause serious problems like deadlocks.

167 apps that use async/await: 2383 async methods

i. 14% of methods that use async/await keywords do this unnecessarily.

ii. 19% of methods are fire&forget calls, cannot be awaited.

RQ2) Do Developers Misuse Async?

Misuse: anti-patterns which hurt performance and might cause serious problems like deadlocks.

167 apps that use async/await: 2383 async methods

i. 14% of methods that use async/await keywords do this unnecessarily.

ii. 19% of methods are fire&forget calls, cannot be awaited.

iii. 1 out of 5 apps misses opportunities in async methods to increase asynchronicity

RQ2) Do Developers Misuse Async?

Misuse: anti-patterns which hurt performance and might cause serious problems like deadlocks.

167 apps that use async/await: 2383 async methods

i. 14% of methods that use async/await keywords do this unnecessarily.

ii. 19% of methods are fire&forget calls, cannot be awaited.

iii. 1 out of 5 apps misses opportunities in async methods to increase asynchronicity

iv. In 99% of cases, developers unnecessarily capture context => performance problems, deadlocks

Developers Need A Bug Fixing Tool: Corrector

Developers frequently misuse async/await keywords

in the paper

Developers Need A Bug Fixing Tool: Corrector

Developers frequently misuse async/await keywords

We built CORRECTOR:
a transformation tool to find and correct the misused async/await

in the paper

Developers Need A Bug Fixing Tool: Corrector

Developers frequently misuse async/await keywords

We built CORRECTOR:
a transformation tool to find and correct the misused async/await

Batch mode to fix the existing mistakes

in the paper

Developers Need A Bug Fixing Tool: Corrector

Developers frequently misuse async/await keywords

We built CORRECTOR:
a transformation tool to find and correct the misused async/await

Batch mode to fix the existing mistakes

Quick fix (VS) mode to prevents user from introducing mistake

in the paper

Developers Need A Bug Fixing Tool: Corrector

Developers frequently misuse async/await keywords

We built CORRECTOR:
a transformation tool to find and correct the misused async/await

Batch mode to fix the existing mistakes

Quick fix (VS) mode to prevents user from introducing mistake

Will mention just one of them, the other three anti-patterns

in the paper

Unnecessary Async/Await Methods

```
public async Task<…> ChangeTemperatureAsync(...) {
   ...
   return await SendPutRequestAsync(url, requestString);
}
```

Unnecessary Async/Await Methods

```
public async Task<…> ChangeTemperatureAsync(...) {
   ...
   return await SendPutRequestAsync(url, requestString);
}
```

Unnecessary Async/Await Methods

```
public async Task<…> ChangeTemperatureAsync(...) {
   ...
   return await SendPutRequestAsync(url, requestString);
}
```

Corrector

Unnecessary Async/Await Methods

```
public async Task<…> ChangeTemperatureAsync(...) {
   ...
   return await SendPutRequestAsync(url, requestString);
}
```

```
public Task<…> ChangeTemperatureAsync(...) {
   ...
   return SendPutRequestAsync(url, requestString);
}
```

Unnecessary Async/Await Methods

```
public async Task<…> ChangeTemperatureAsync(...) {
   ...
   return await SendPutRequestAsync(url, requestString);
}
```

```
public Task<…> ChangeTemperatureAsync(...) {
   ...
   return SendPutRequestAsync(url, requestString);
}
```

14% of methods that use (the expensive) async/await keywords do this unnecessarily

Unnecessary Async/Await Methods

```
public async Task<…> ChangeTemperatureAsync(...) {
   ...
   return await SendPutRequestAsync(url, requestString);
}
```

```
public Task<…> ChangeTemperatureAsync(...) {
   ...
   return SendPutRequestAsync(url, requestString);
}
```

14% of methods that use (the expensive) async/ await keywords do this unnecessarily

26% of the apps have at least one such an async method.

Corrector Eliminates Unnecessary Async/ Await

```
public async Task<CacheResponse> RequestAsync(CacheRequest request)
{
    return await downloaderPluginBase.RequestAsync(request);
}
```

Unnecessary await in async method

Corrector Eliminates Unnecessary Async/ Await

```
public async Task<CacheResponse> RequestAsync(CacheRequest request)
{
    return await downloaderPluginBase.RequestAsync(request);
}
```

Unnecessary await in async method

Corrector Eliminates Unnecessary Async/ Await

```
public async Task<CacheResponse> RequestAsync(CacheRequest request)
{
    return await downloaderPluginBase.RequestAsync(request);
}
```

Unnecessary await in async method

Remove unnecessary async/await

```
...

public Task<CacheResponse> RequestAsync(CacheRequest request)
{
    return  downloaderPluginBase.RequestAsync(request);
}
```

Empirical Evaluation

Used the same code corpus from the Formative Empirical Study: 1378 WP apps, 12M SLOC

Empirical Evaluation

Used the same code corpus from the Formative Empirical Study: 1378 WP apps, 12M SLOC

(1) Are they applicable?

Empirical Evaluation

Used the same code corpus from the Formative Empirical Study: 1378 WP apps, 12M SLOC

(1) Are they applicable?

(2) What is the impact of refactoring on code?

Empirical Evaluation

Used the same code corpus from the Formative Empirical Study: 1378 WP apps, 12M SLOC

(1) Are they applicable?

(2) What is the impact of refactoring on code?

(3) Are they efficient?

Empirical Evaluation

Used the same code corpus from the Formative Empirical Study: 1378 WP apps, 12M SLOC

(1) Are they applicable?

(2) What is the impact of refactoring on code?

(3) Are they efficient?

(4) Are the transformations useful?

(1) Our Tools Are Highly Applicable

54% of 1245 callbacks passed the preconditions.

ASYNCIFIER refactored 54% of callbacks.

We manually verified that 10% of the instances are correct.

The reasons for un-refactored 46% of 1245 callbacks:

(i) Algorithm's preconditions

(ii) Tool limitations

(1) Our Tools Are Highly Applicable

54% of 1245 callbacks passed the preconditions.

ASYNCIFIER refactored 54% of callbacks.

We manually verified that 10% of the instances are correct.

The reasons for un-refactored 46% of 1245 callbacks:

(i) Algorithm's preconditions

(ii) Tool limitations

CORRECTOR fixed 4 main types of anti-patterns (2209)

(1) Our Tools Are Highly Applicable

54% of 1245 callbacks passed the preconditions.

ASYNCIFIER refactored 54% of callbacks.

We manually verified that 10% of the instances are correct.

The reasons for un-refactored 46% of 1245 callbacks:

(i) Algorithm's preconditions

(ii) Tool limitations

CORRECTOR fixed 4 main types of anti-patterns (2209)

Our tools can be applied to any platform written in C# (not only Windows Phone)

(2) Automation is needed

(2) Automation is needed

ASYNCIFIER changes 28.9 lines on average per refactoring!

(2) Automation is needed

ASYNCIFIER changes 28.9 lines on average per refactoring!

(3) Our tools are efficient

(2) Automation is needed

ASYNCIFIER changes 28.9 lines on average per refactoring!

(3) Our tools are efficient

ASYNCIFIER takes 508ms on average per refactoring

CORRECTOR takes 47ms on average per refactoring

(2) Automation is needed

ASYNCIFIER changes 28.9 lines on average per refactoring!

(3) Our tools are efficient

ASYNCIFIER takes 508ms on average per refactoring

CORRECTOR takes 47ms on average per refactoring

suitable for an interactive usage in an IDE

(4) Developers Find Asyncifier Useful

Choose 10 most recently updated apps that have callbacks.

Applied Asyncifier ourselves and offered patches.

9 out of 10 apps responded and accepted 28 refactorings.

(4) Developers Find Asyncifier Useful

Choose 10 most recently updated apps that have callbacks.

Applied Asyncifier ourselves and offered patches.

9 out of 10 apps responded and accepted 28 refactorings.

"That was pretty good. look forward to the release of refactoring tool, it seems to be really useful." *Ocell app*

(4) Developers Find Asyncifier Useful

Choose 10 most recently updated apps that have callbacks.

Applied Asyncifier ourselves and offered patches.

9 out of 10 apps responded and accepted 28 refactorings.

"That was pretty good. look forward to the release of refactoring tool, it seems to be really useful." *Ocell app*

"You are doing very interesting stuff. That's fortunate, because I've been thinking about replacing all asynchronous calls to new async/await style code" *phoneguitartab app*

(4) Developers Find Asyncifier Useful

Choose 10 most recently updated apps that have callbacks.

Applied Asyncifier ourselves and offered patches.

9 out of 10 apps responded and accepted 28 refactorings.

"That was pretty good. look forward to the release of refactoring tool, it seems to be really useful." *Ocell app*

"You are doing very interesting stuff. That's fortunate, because I've been thinking about replacing all asynchronous calls to new async/await style code" *phoneguitartab app*

"That's very interesting and useful." *32feet app*

(4) Developers Find Corrector Useful

Choose 20 most recently updated apps (have anti patterns)

All apps accepted all our 288 instances of CORRECTOR transformations

(4) Developers Find Corrector Useful

Choose 20 most recently updated apps (have anti patterns)

All apps accepted all our 288 instances of CORRECTOR transformations

"This is awesome insight!Great work!" *indulged-flickr app*

(4) Developers Find Corrector Useful

Choose 20 most recently updated apps (have anti patterns)

All apps accepted all our 288 instances of CORRECTOR transformations

"This is awesome insight! Great work!" *indulged-flickr app*

"Thanks for your great advice. You are right, the async/await keywords could be removed" *kanboxwp app*

(4) Developers Find Corrector Useful

Choose 20 most recently updated apps (have anti patterns)

All apps accepted all our 288 instances of CORRECTOR transformations

"This is awesome insight! Great work!" *indulged-flickr app*

"Thanks for your great advice. You are right, the async/await keywords could be removed" *kanboxwp app*

"Totally agree, I normally try to take the same minimizing approach, though it seems I missed these. Thanks!" *Cimbalino app*

(4) Developers Find Corrector Useful

Choose 20 most recently updated apps (have anti patterns)

All apps accepted all our 288 instances of CORRECTOR transformations

"This is awesome insight! Great work!" *indulged-flickr app*

"Thanks for your great advice. You are right, the async/await keywords could be removed" *kanboxwp app*

"Totally agree, I normally try to take the same minimizing approach, though it seems I missed these. Thanks!" *Cimbalino app*

"That you have pointed out is correct. This time, performance has been improved to 28 milliseconds from 49 milliseconds." *Softbuild.Data app*

Implications

Developers

LearnAsync.NET: hundreds of real-world positive and negative examples of asynchronous idioms

Implications

Developers

LearnAsync.NET: hundreds of real-world positive and negative examples of asynchronous idioms

Usage Examples of Asynchronous Idioms

Developers learn a new programming construct through both positive and negative examples. We provide hundreds of real-world examples of all asynchronous idioms. Because developers might need to inspect the whole source file or project to understand the example, we link to highlighted source files on GitHub.

Long-running CPU-Bound Operations	
New Thread	Link to Usage Examples
Background Worker	Link to Usage Examples
Threadpool	Link to Usage Examples
New Task	Link to Usage Examples
Blocking I/O-Bound Operations	
Callback-Based APM Idioms	Link to Usage Examples
Task-Based TAP Idioms with async/await	Link to Usage Examples
Misuse of Async/Await	
Fire-Forget async Methods	Link to Usage Examples
Unnecessary async Methods	Link to Usage Examples
Long-running Operations Under async Methods	Link to Usage Examples
Unnecessarily Capturing Context Under async Methods	Link to Usage Examples

Implications

Language and Library Designers

async/await will be available in other languages too
Provided a detailed insight about why async/await is misused

New concepts require not only documentation but also IDE support.

Conclusions

First large-scale formative study on the usage of asynchronous programming

Conclusions

First large-scale formative study on the usage of asynchronous programming

A toolkit inspired by our empirical study: (1) refactoring, (2) bug fixing tools

Conclusions

First large-scale formative study on the usage of asynchronous programming

A toolkit inspired by our empirical study: (1) refactoring, (2) bug fixing tools

Evaluation: highly applicable, efficient, useful

Conclusions

First large-scale formative study on the usage of asynchronous programming

A toolkit inspired by our empirical study: (1) refactoring, (2) bug fixing tools

Evaluation: highly applicable, efficient, useful

Architects of C# and F# confirmed that our findings will influence the language design

Conclusions

First large-scale formative study on the usage of asynchronous programming

A toolkit inspired by our empirical study: (1) refactoring, (2) bug fixing tools

Evaluation: highly applicable, efficient, useful

Architects of C# and F# confirmed that our findings will influence the language design

Some part of CORRECTOR will ship with official Visual Studio 2014