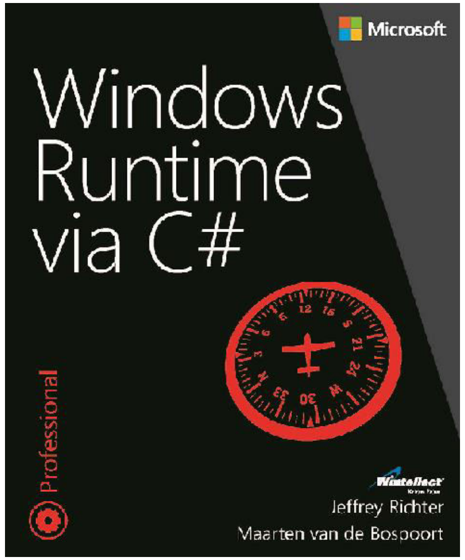
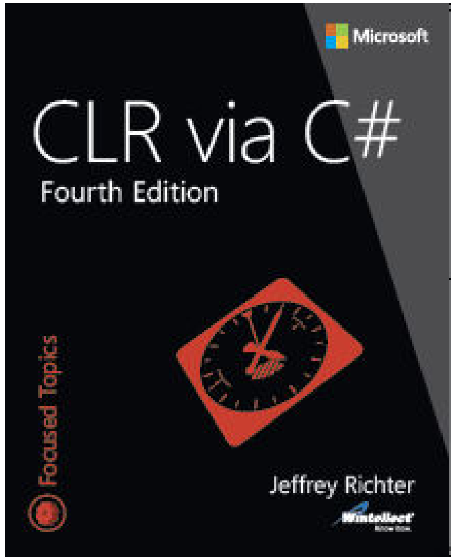
Building Responsive & Scalable Applications

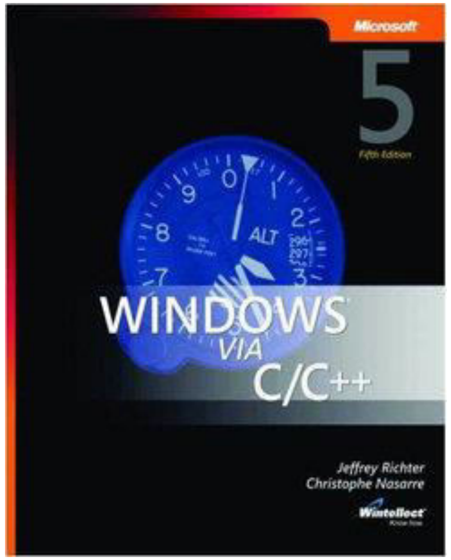
Jeffrey Richter

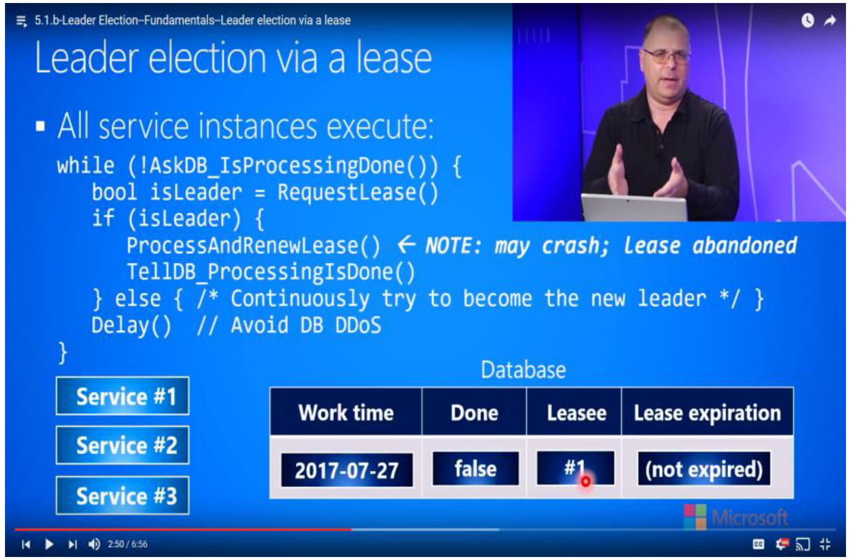
Consulting/Training

Jeffrey Richter: Microsoft Azure Software Architect, Author, & Wintellect Co-Founder








5.1.b-Leader Election–Fundamentals–Leader election via a lease
2:50 / 6:56
Microsoft

Leader election via a lease

- All service instances execute:

```
while (!AskDB_IsProcessingDone()) {
   bool isLeader = RequestLease()
   if (isLeader) {
      ProcessAndRenewLease() ← NOTE: may crash; lease abandoned
      TellDB_ProcessingIsDone()
   } else { /* Continuously try to become the new leader */ }
   Delay()  // Avoid DB DDoS
}
```

Service #1
Service #2
Service #3

Database

Work time	Done	Leasee	Lease expiration
2017-07-27	false	#1	(not expired)

Architecting Distributed Cloud Apps
6.5hr technology-agnostic course
http://aka.ms/RichterCloudApps

JeffreyR@Microsoft.com

www.linkedin.com/in/JeffRichter

@JeffRichter

Consulting/Training

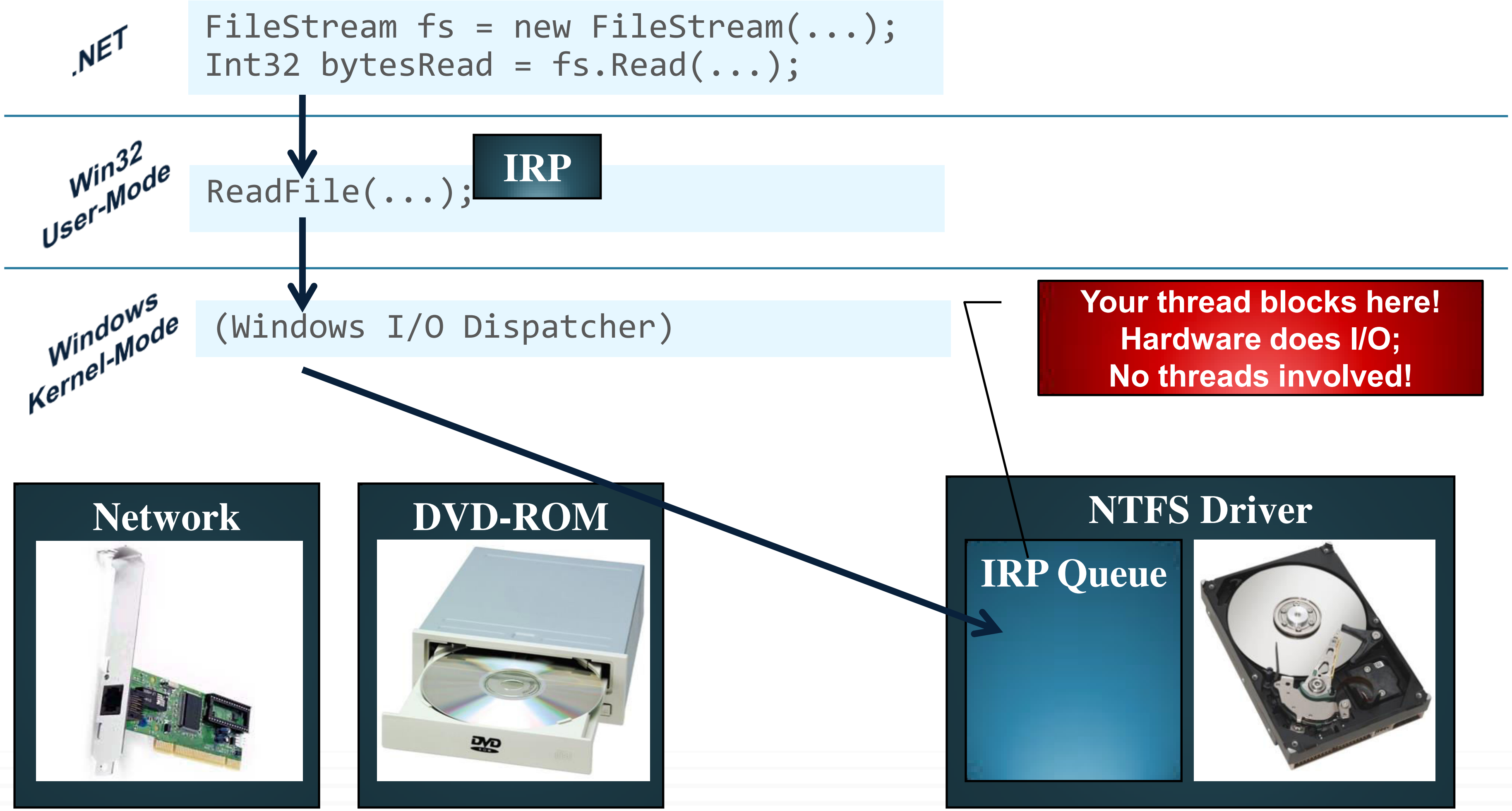
Motivation

- Early OSes didn't support threads (there was just 1 thread)
  - Problem: Long-running tasks affected all apps and the OS
  - Solution: Windows supports 1+ threads/process for robustness
- Threads have space & time overhead
  - Kernel object (contains thread's properties & register set context)
    - Context size in bytes: x86 = ~700, x64 = ~1240, ARM = ~350
  - User-mode data (Thread Environment Block)
    - 4KB, exception-handling chain, TLS, GDI/OpenGL stuff
  - Stacks: user-mode (1MB committed) & kernel-mode (12KB/24KB)
  - DLL thread attach/detach notifications
- 1 CPU can only run 1 thread at a time
  - After quantum, Windows context switches to another thread

Motivation

- Every context switch requires that Windows
  - Save registers from CPU to running thread's kernel object
  - Determine which thread to schedule next
    - If thread owned by other process, switch address space
  - Load registers from selected thread's kernel object into CPU
  - After the switch, CPU suffers cache misses repopulating its cache
- All of this is pure overhead and hurts performance
  - But required for a robust OS
- Conclusion
  - Avoid threads: incur time & memory overhead
  - Use threads: responsiveness & scalability (on multi-CPU system)
  - This talk is about wrestling with this tension

Synchronous I/O



.NET

```
FileStream fs = new FileStream(...);
Int32 bytesRead = fs.Read(...);
```

Win32 User-Mode

```
ReadFile(...);
```

IRP

Windows Kernel-Mode

```
(Windows I/O Dispatcher)
```

Your thread blocks here!
Hardware does I/O;
No threads involved!

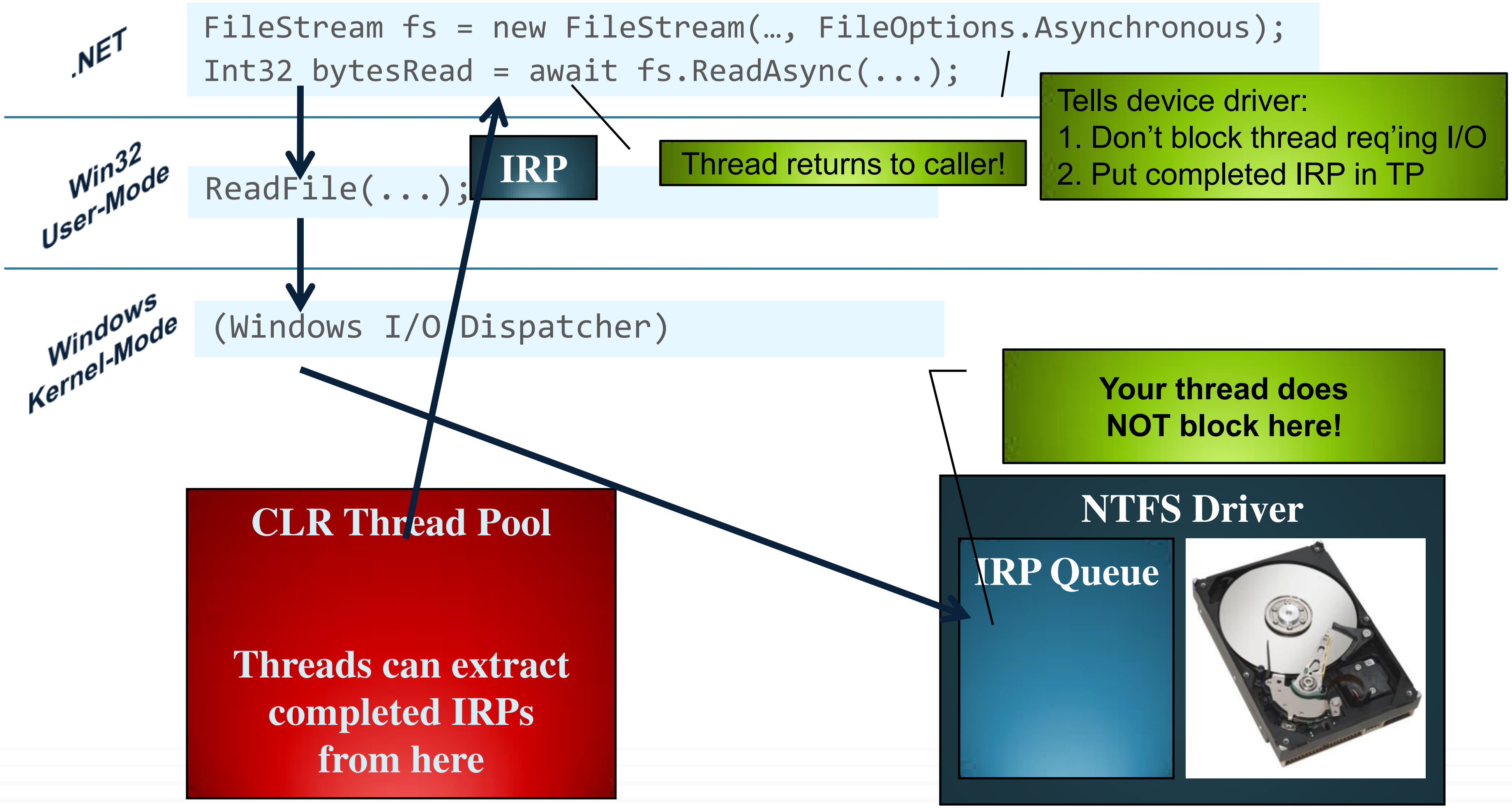
Network

DVD-ROM

NTFS Driver

IRP Queue

Asynchronous I/O with ***Xxx***Async



```
FileStream fs = new FileStream(…, FileOptions.Asynchronous);
Int32 bytesRead = await fs.ReadAsync(...);
```

```
ReadFile(...);
```

```
(Windows I/O Dispatcher)
```

Thread returns to caller!

Tells device driver:
1. Don’t block thread req’ing I/O
2. Put completed IRP in TP

Your thread does NOT block here!

CLR Thread Pool

Threads can extract completed IRPs from here

NTFS Driver

IRP Queue

Async Functions are State Machine Objects

```
// 'async' turns method into state machine, requires Task return type
// (identifying operation completing in future) & allows use of await
async Task<Int32> HttpLengthAsync(String uri) {
   String html = await new HttpClient().GetStringAsync(uri);
   return html.Length;
}
```

```
Task<Int32> HttpLengthAsync() {  // uri → m_uri
   try {
      switch (m_state) { // Defaults to 0
      case 0:
         m_taskHLA = new Task<Int32>();   // HttpLengthAsync’s task

         // XxxAsync queues IRP to device driver & returns Task<String>
         m_taskGSA = new HttpClient().GetStringAsync(m_uri);
         if (m_taskGSA.IsCompleted) goto case 1;  // Perf optimization
         m_state = 1; m_taskGSA.ContinueWith(HttpLengthAsync); break;  // From await

       case 1:
         String html = m_taskGSA.Result;   // Throws if I/O failed
         m_taskHLA.SetResult(html.Length);
         break;
      }
   }
   catch (Exception e) { m_taskHLA.SetException(e); }
   return m_taskHLA;
}  // Thread returns to caller or thread pool
```

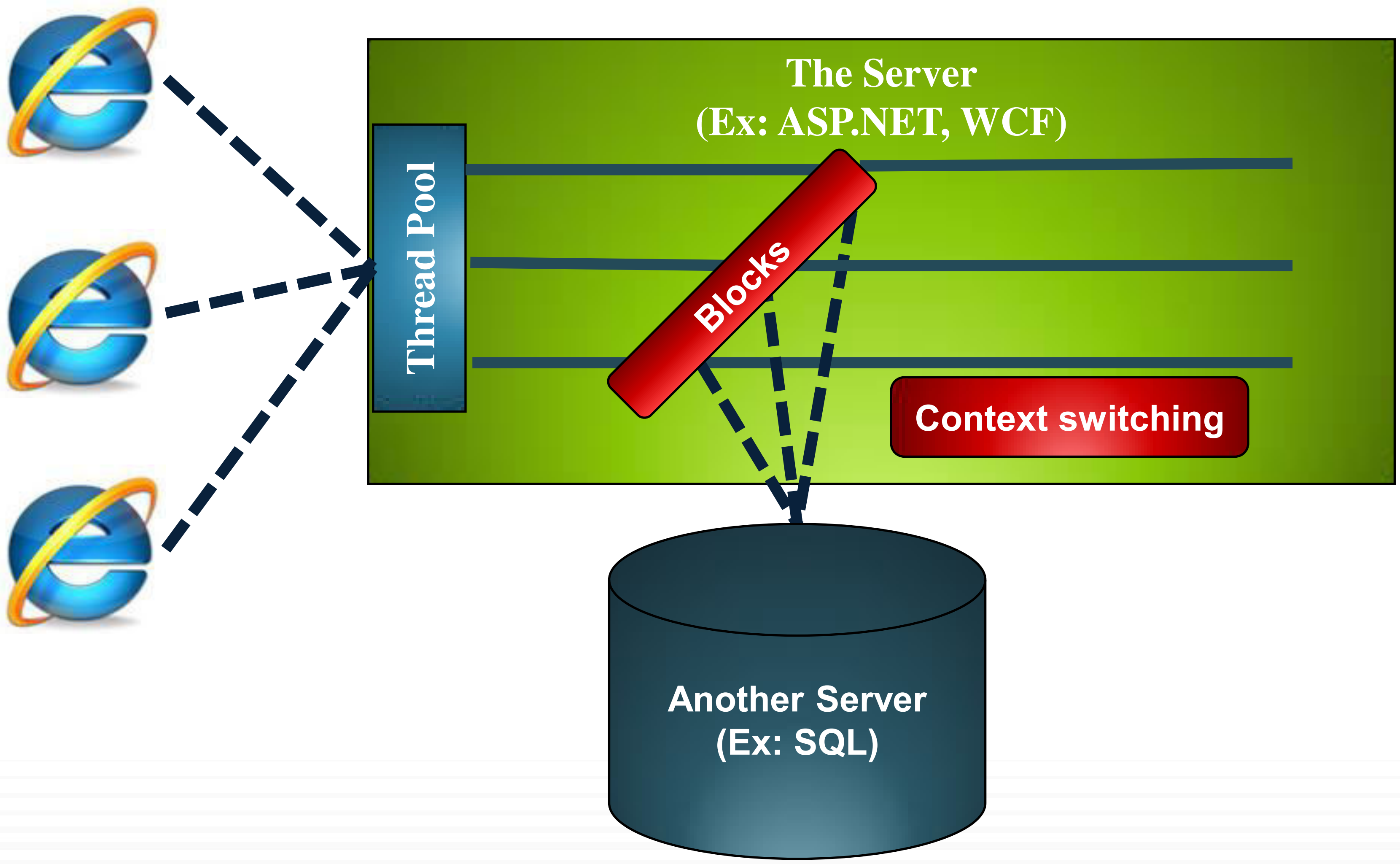
Named Pipe Client

```
async Task<String> IssueClientRequestAsync(String serverName, String msg) {

   using (var pipe = new NamedPipeClientStream(serverName, "PipeName",
      PipeDirection.InOut, PipeOptions.Asynchronous)) {

      pipe.Connect(); // Must Connect before setting ReadMode
      pipe.ReadMode = PipeTransmissionMode.Message;

      // Asynchronously send data to the server
      Byte[] request = Encoding.UTF8.GetBytes(msg);
      await pipe.WriteAsync(request, 0, request.Length);

      // Asynchronously read the server's response
      Byte[] response = new Byte[1000];
      Int32 bytesRead = await pipe.ReadAsync(response, 0, response.Length);
      return Encoding.UTF8.GetString(response, 0, bytesRead);
   }  // Close the pipe
}
```

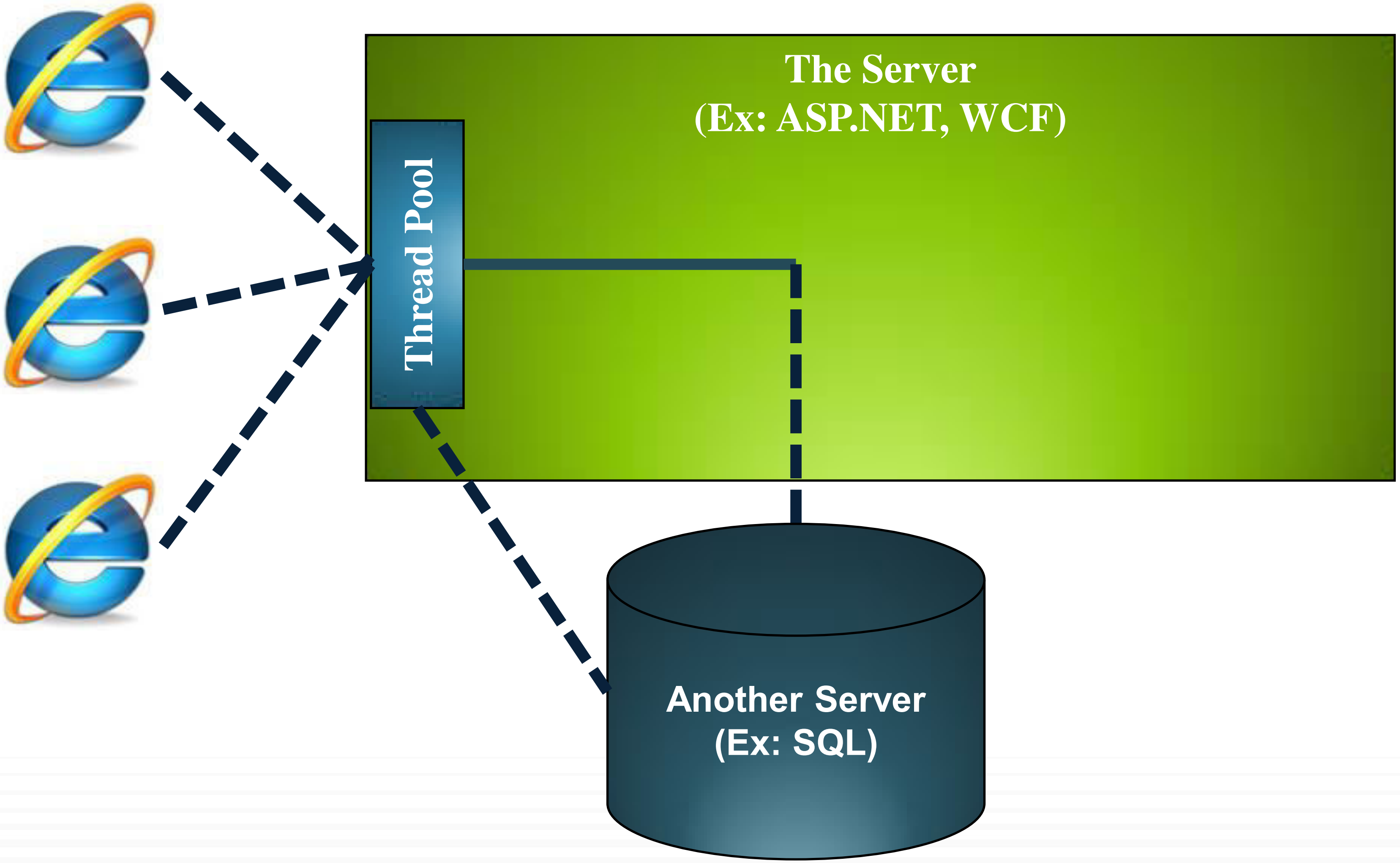
Some Async Functions in the FCL

- Stream-derived types
  - ReadAsync, WriteAsync, FlushAsync, CopyToAsync
- TextReader-derived types
  - ReadAsync, ReadLineAsync, ReadToEndAsync, ReadBlockAsync
- TextWriter-derived types
  - WriteAsync, WriteLineAsync, FlushAsync
- HttpClient
  - GetAsync, PostAsync, PutAsync, DeleteAsync, …
- SqlCommand
  - ExecuteDbDataReaderAsync, ExecuteNonQueryAsync, ExecuteReaderAsync, ExecuteScalarAsync, …
- Tools (like SvcUtil.exe) that produce web service proxy classes

Non-Scalable Servers



Scalable Servers



Application Models & their Threading Models

Applications & their Threading Models

- Applications impose their own threading model
  - CUI/Services: no model; any thread can do anything
  - GUI: window must be modified by thread that creates it
  - ASP.NET (Forms/Services): impersonates client's culture/identity
    - http://msdn.microsoft.com/en-us/library/bz9tc508.aspx
- SynchronizationContext-derived objects connect an application model to its threading model
- The **await** operator captures the calling thread's SC and calls through it when resuming the state machine
  - For application code, this is usually good
  - For class library code, this is usually bad

GUI App Deadlocks

```
private sealed class MyWpfWindow : Window {
   protected override void OnActivated(EventArgs e) {
      // Calling GetResult makes GUI thread block waiting for the result
      var uri = "http://Wintellect.com/";
      Int32 length = HttpLengthAsync(uri).GetAwaiter().GetResult();
      // Do something with ‘length’ ...
      base.OnActivated(e);
   }

   private async Task<Int32> HttpLengthAsync(String uri) {
      // Issue HTTP request & let thread return to caller
      String text = await new HttpClient().GetStringAsync(uri);

      // We never get here: GUI thread waits for this method to finish but it
      // can't because the GUI thread is waiting for it to finish → DEADLOCK!
      return text.Length;
   }
}
```

App-Model Agnostic Code
should use ConfigureAwait(false)

```
private async Task<Int32> HttpLengthAsync(String uri) {
   // Issue HTTP request & let thread return to caller
   String text = await new HttpClient().GetStringAsync(uri)
      .ConfigureAwait(false); // Do NOT use calling SynchronizationContext

   // We DO get here now because a thread pool thread can execute
   // this code as opposed to forcing the GUI thread to execute it.
   // Of course, don’t try to update the UI here!

   return text.Length;
}
```

You ***must*** apply .ConfigureAwait(false) to every Task you await !
(because some tasks may complete synchronously)

Also, ignoring SynchronizationContext improves performance

Task.Run Forces use of Thread Pool Threads

```
private /* async */ Task<Int32> HttpLengthAsync(String uri) {
   // Task.Run is called on the GUI thread & returns immediately
   return Task.Run(async () => {
      // The lambda body executes via a thread pool thread which
      // doesn’t have a SynchronizationContext associated with it
      String text = await new HttpClient().GetStringAsync(uri);

      // We DO get here because a thread pool thread can execute this code

      return text.Length;
   });
}
```

Note: .ConfigureAwait(false) not needed anywhere now !

Questions