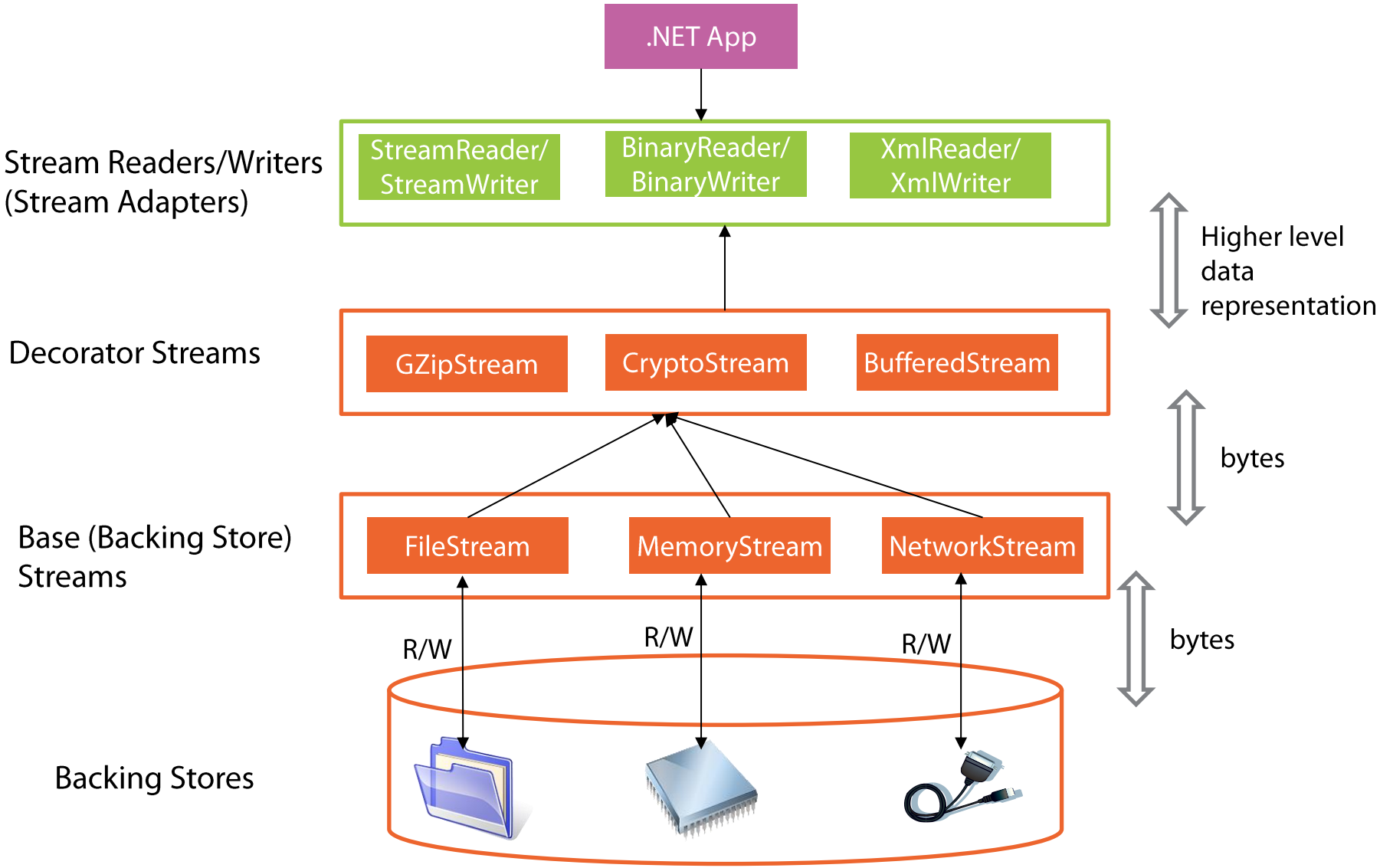
Decorator Streams

Mohamad Halabi
Microsoft Integration MVP
@mohamadhalabi

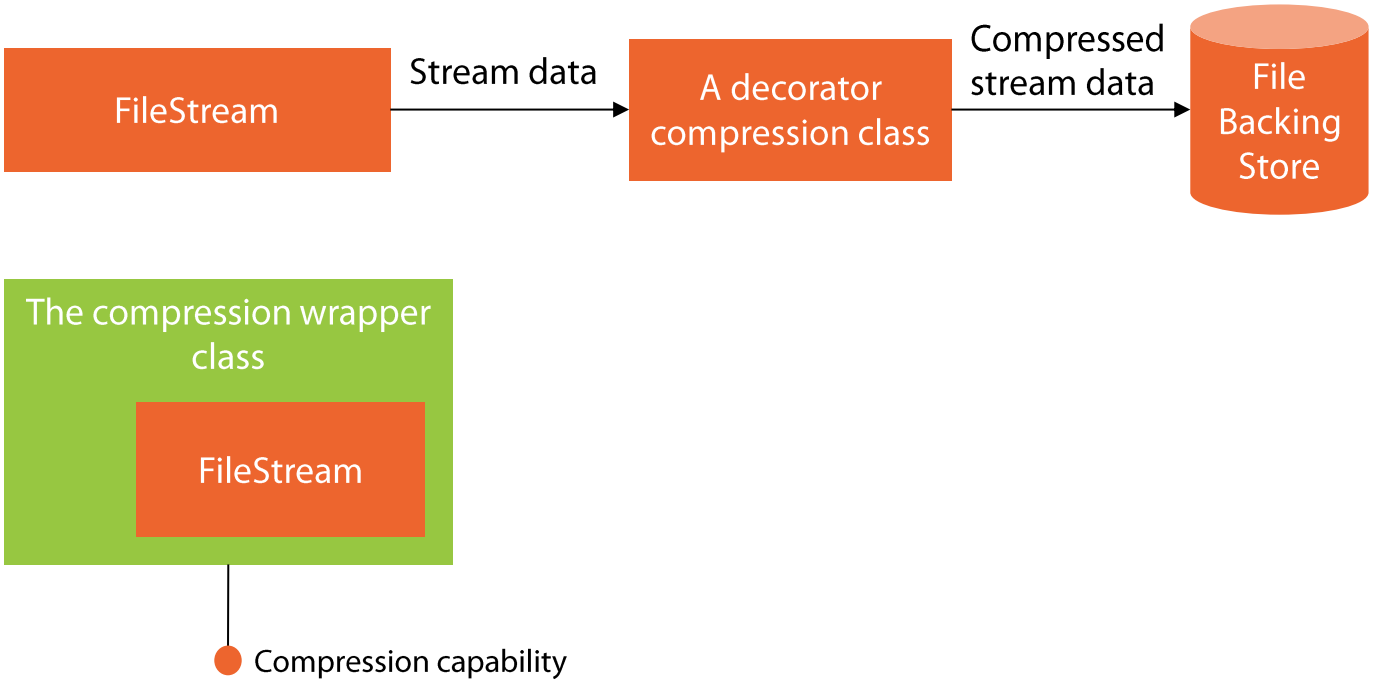
pluralsight
hardcore dev and IT training

Overall Architecture

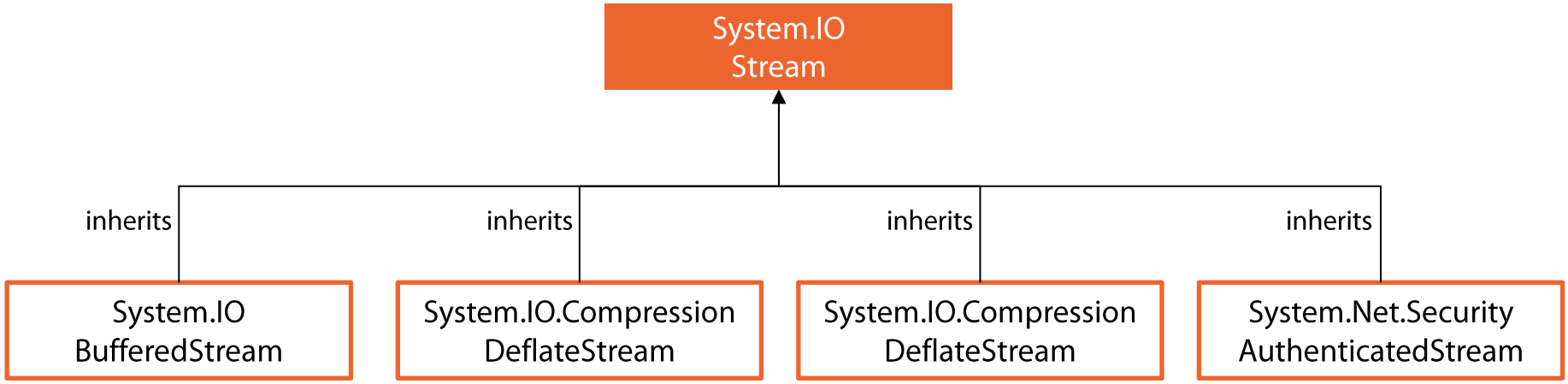


So What is a Decorator?

- Wrapper class that implements the Decorator design pattern
- Adds runtime capabilities to the wrapped class



Decorator Streams



Buffering Reviewed

- **Enhances the performance of reading/writing by reducing number of I/O operations**
- **Implemented via an in-memory buffer**

Write	Read
Data is written to buffer instead of backing store	More data is read than what is actually requested
When buffer is full, data is flushed to the backing store	Extra data is stored in buffer and used for future reads
Single I/O operation instead of multiple ones	Number of I/O operations reduced

- **FileStream implements an internal buffer**
- **Q: How do streams without internal buffers benefit from buffering?**
- **A: BufferedStream decorator stream**

BufferedStream

- **Two constructor overloads:**
  1. *BufferedStream(Stream)* : default buffer size is 4096 bytes
  2. *BufferedStream(Stream, Int32)* : buffer size specified

```
NetworkStream networkStream = tcpClient.GetStream();
BufferedStream bs = new BufferedStream(networkStream);
bs.WriteByte(200);
bs.ReadByte();
```

Compression Streams

- **DeflateStream and GZipStream**
  - Use the same compression algorithm
  - GZipStream uses DeflateStream
  - GZipStream adds Cyclic Redundancy Check (CRC) to compressed data
  - GZipStream is more reliable than DeflateStream
  - GZipStream slower than DeflateStream and produces larger size

Compression and Decompression

- [GZipStream/DeflateStream].Write: compresses data
- [GZipStream/DeflateStream].Read: decompresses data

```
using (FileStream fs1 = new FileStream(@"c:\files\data.txt", FileMode.CreateNew))


using (Stream gs = new GZipStream(fs1, CompressionMode.Compress))


 gs.Write(dataToRead, 0, dataToRead.Length);
```

CryptoStream

- **Provides cryptographic operations over stream data**
- **Requires understanding of cryptographic concepts**
  - Search ‘cryptography’ in Pluralsight library

Encryption Example

```
//encrypt
byte[] key = { 11, 22, 33, 44, 55, 66, 77, 88, 99, 100, 200, 123, 156, 34, 89, 93 };
byte[] iv = { 34, 24, 32, 44, 55, 60, 13, 9, 22, 55, 77, 90, 23, 12, 13, 11 };
byte[] data = Encoding.UTF8.GetBytes("this is a text to encrypt");

using (SymmetricAlgorithm algorithm = Aes.Create())
using (ICryptoTransform encryptor = algorithm.CreateEncryptor(key, iv))
using (FileStream stream = new FileStream(@"c:\files\text.txt", FileMode.CreateNew))
using (CryptoStream crypto = new CryptoStream(stream, encryptor, CryptoStreamMode.Write))
    crypto.Write(data, 0, data.Length);
```

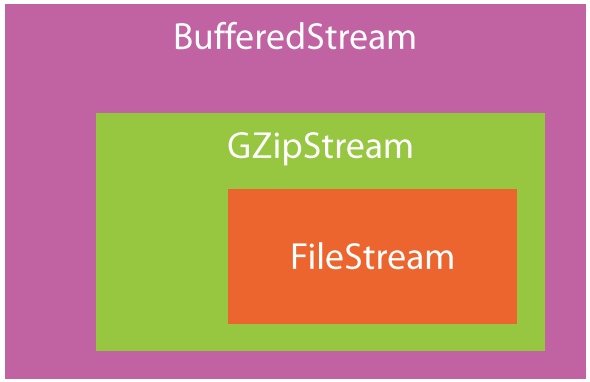
Decryption Example

```
//decrypt
StreamReader sr = null;
using (SymmetricAlgorithm algorithm = Aes.Create())
using (ICryptoTransform decryptor = algorithm.CreateDecryptor(key, iv))
using (FileStream stream = new FileStream(@"c:\files\text.txt", FileMode.Open))
using (CryptoStream crypto = new CryptoStream(stream, decryptor, CryptoStreamMode.Read))
{
    sr = new StreamReader(crypto);
    Console.WriteLine(sr.ReadToEnd());
}

Console.ReadLine();
```

Chaining Decorator Streams

- **Decorator streams can be chained**
  - Ex: Compression stream followed by encryption stream




```
FileStream fs = new FileStream(@"c:\files\text.txt", FileMode.Open);
BufferedStream bs = new BufferedStream(new GZipStream(fs, CompressionMode.Compress));
```

Disposing Decorator Streams

- Closing a decorator stream also closes the decorated backing store stream
- In a chain, closing the stream at the start of the chain, also closes the entire chain

Summary

- **Implement decorator design pattern**
- **Perform runtime functions on byte data of backing store stream**
- **Three commonly used backing store streams:**
  - BufferedStream
  - DeflateStream and GZipStream
  - CryptoStream
- **A chain of decorator streams can be created**
  - Closing the start of the chain, closes the entire chain