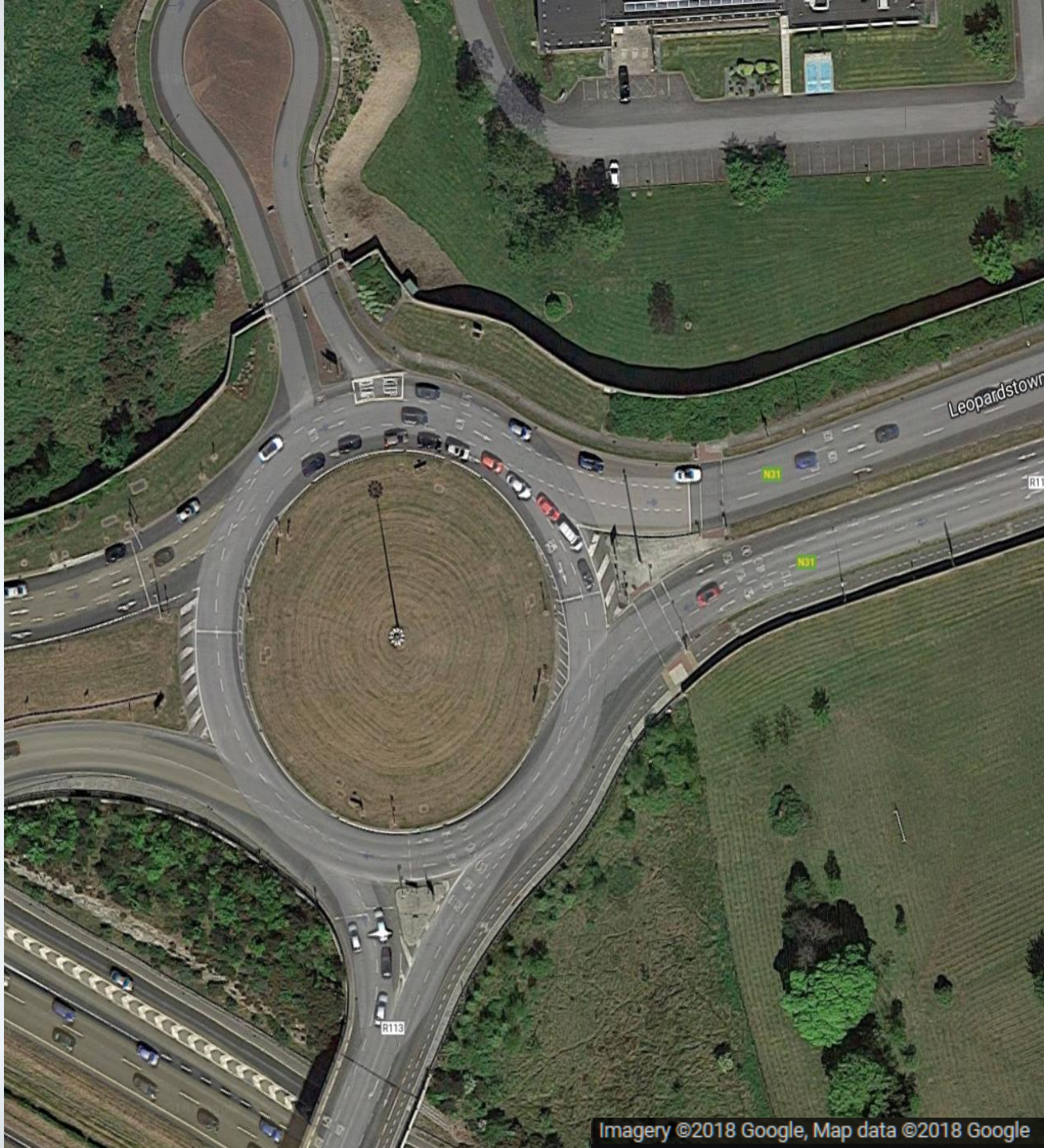
Scalable Coding

Igore@microsoft.com
@_igore_

The Roundabout

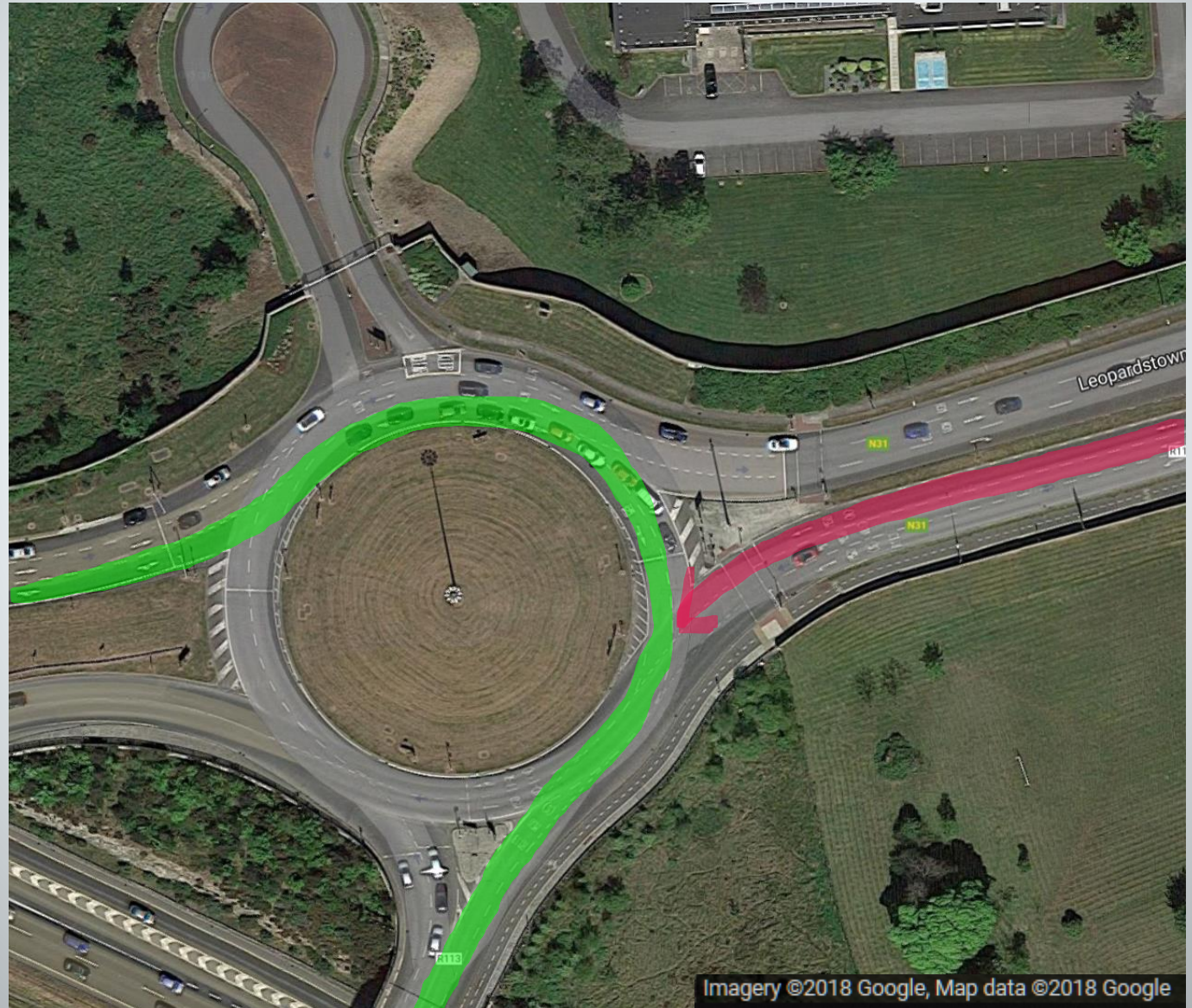
- Designed for “fair” traffic control
- Reduce driver confusion due to simplicity of rules
- Reduced risk of likelihood and severity of collisions




R413
M50
Image capture: Apr 2017 © 2018 Google


R413
150
Image capture: Apr 2017 © 2018 Google

Roundabouts



What does the roundabout story tell us?

- Roundabouts will fail for targeted high traffic scenarios.
- Brains can fall back to a degraded state
- We only perceive an abstracted version of reality
- To be able to identify patterns we must either have experienced the patterns or we need to know what to look for

Code Reviews

- Find problems early
- Ensure consistency
- Teaching/Learning
- Find and Fix Pain Generators

Regular Expressions

```
public static class Splitter
    {
        static readonly Regex splitRegex = new Regex(@"\s", RegexOptions.Compiled);
        // Splitting sentences at spaces
        public static string[] SplitOnSpaces(string text)
        {
            return splitRegex.Split(text);
        }
    }
```

Regular Expressions

```
public static class Splitter
    {
        static readonly Regex splitRegex = new Regex(@" ", RegexOptions.Compiled);
        // Splitting sentences at spaces
        public static string[] SplitOnSpaces(string text)
        {
            return splitRegex.Split(text);
        }
    }
```

Regular Expressions

```
public static class WordSplitter
    {
        static readonly Regex splitWordsRegex = new Regex(@" ", RegexOptions.Compiled);

        public static string[] SplitUsingRegex(string text)
        {
            return splitWordsRegex.Split(text);
        }

        public static string[] SplitUsingSplit(string text)
        {
            return text.Split();
        }
    }
```

Regular Expressions

- Measure performance of Regex vs reference implementation.
- Opt for readability if performance is similar or if performance doesn't matter
- Keep Regexes as precise as possible.
- Avoid quantifiers (*,+,?), conditional matchings and either/or constructs.

Exceptions vs Error Codes

```
public void ProcessStuff(bool IsSuccess)
       {
           if (IsSuccess)
           {
               int b = 0;
               for (int i = 0; i < 100; i++)
               {
                   b = b - i;
               }
               return;
           }
           throw new Exception("I failed you!");
       }
```

>70 seconds

```
public int ProcessStuff(bool IsSuccess)
       {
           if (IsSuccess)
           {
               int b = 0;
               for (int i = 0; i < 100; i++)
               {
                   b = b - i;
               }
               return 0;
           }
           return 1;
       }
```

2ms

Exceptions vs Error Codes

- Use exceptions are a source of information for unexpected problems.
- Limit stack depth of exceptions
- Load test your error scenarios

Exceptions vs Error Codes

- Use exceptions are a source of information for unexpected problems.
- Limit stack depth of exceptions
- Load test your error scenarios

Atomicity

```
public void ProcessData(Object Data)
{
    var A = FirstStep(Data);
    SecondStep(A);
}
```

Atomicity

```
public static void ProcessData(Object Data)
{
    var A = FirstStep(Data, out int errorCode);
    if (errorCode == 0)
    {
        SecondStep(A);
    }
    else
    {
        // Rollback
    }
}
```

Atomicity

- Should we use locks?
- Can we make the underlying calls idempotent?
- Can we use some principles of asynchronous processing?

Hardcoded Strings, Magic Numbers and other Evil Powers

```
public void DoSomeOperation()
{
    logMetric("Success","Storage Operation");
}
public void DoSAnotherOperation()
{
   logMetric(“Error", "Storage operation");
}
```

Hardcoded Strings, Magic Numbers and other Evil Powers

```
public void DoSomeOperation()
{
    logMetric(Consts.OperationResult.Success, Consts.ERROR_STORAGE);
}
public void DoSAnotherOperation()
{
   logMetric(Consts.OperationResult.Error, Consts.ERROR_STORAGE);
}
```

Hardcoded Strings, Magic Numbers and other Evil Powers

```
{
  "serviceName": "MyService",
  "severity": 1,
  "alertName": "Storage error count too high",
  "scope": "Instance",
  "type": "Counter",
  "comparator": ">",
  "value": 500,
  "frequency": "00:05:00"
}
```

Other stuff we haven't covered

- Consistency models
- Transient Faults
- Degrading vs Failing
- Time assumptions and calculations
- Tight Loops
- Locks
- Configuration drift
- …

cat /etc/motd

- Pay attention to internal implementations
- Understand the costs of frameworks and libraries
- Assume every method will break at any point of execution
- Load test code paths with high-execution rate
- Load test error scenarios
- Reviews are your friends

Questions?