Exploring Spans and Pipelines

Tim Iskhakov

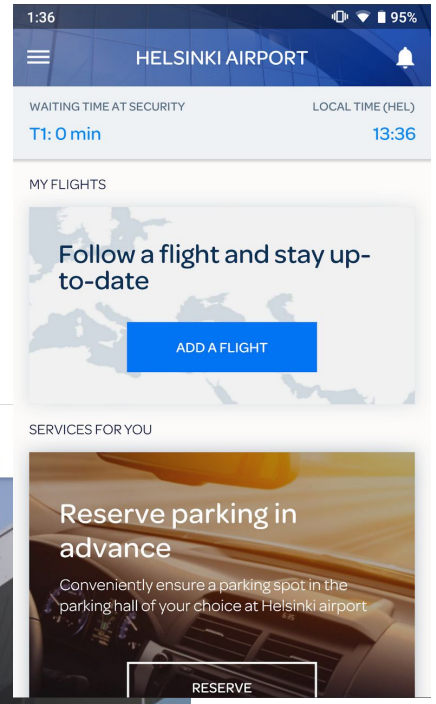
Disclaimer

This is a talk version of the post I wrote a while ago:

https://timiskhakov.github.io/posts/exploring-spans-and-pipelines

Finavia

- Backend for the website
- Mobile application
- Airport map
- SIS (Service Information Screens)



Fetching Parking Prices (for Helsinki Airport)

Choose area and additional services

You can also reserve and pay car cleaning services.

P3 PREM Available	50 m walk to T1 300 m walk to T2	ONLINE PRICE ⓘ ~~148€~~ 143€
P3 Available	50 m walk to T1 300 m walk to T2	ONLINE PRICE ⓘ ~~79€~~ 74€
P4A Available	750 m walk to T1 950 m walk to T2	ONLINE PRICE ⓘ ~~47€~~ 40€

Fetching Parking Prices (for Helsinki Airport)

- Check for a new file
- Download the file from a 3rd party system
- Parse the file (~10Mb)
- Apply application logic
- Save to database
- Repeat previous steps every 5 min

Choose area and additional services

You can also reserve and pay car cleaning services.

P3 PREM Available	50 m walk to T1 300 m walk to T2	ONLINE PRICE ⓘ ~~148€~~ 143€
P3 Available	50 m walk to T1 300 m walk to T2	ONLINE PRICE ⓘ ~~79€~~ 74€
P4A Available	750 m walk to T1 950 m walk to T2	ONLINE PRICE ⓘ ~~47€~~ 40€

Fetching Parking Prices (for Helsinki Airport)

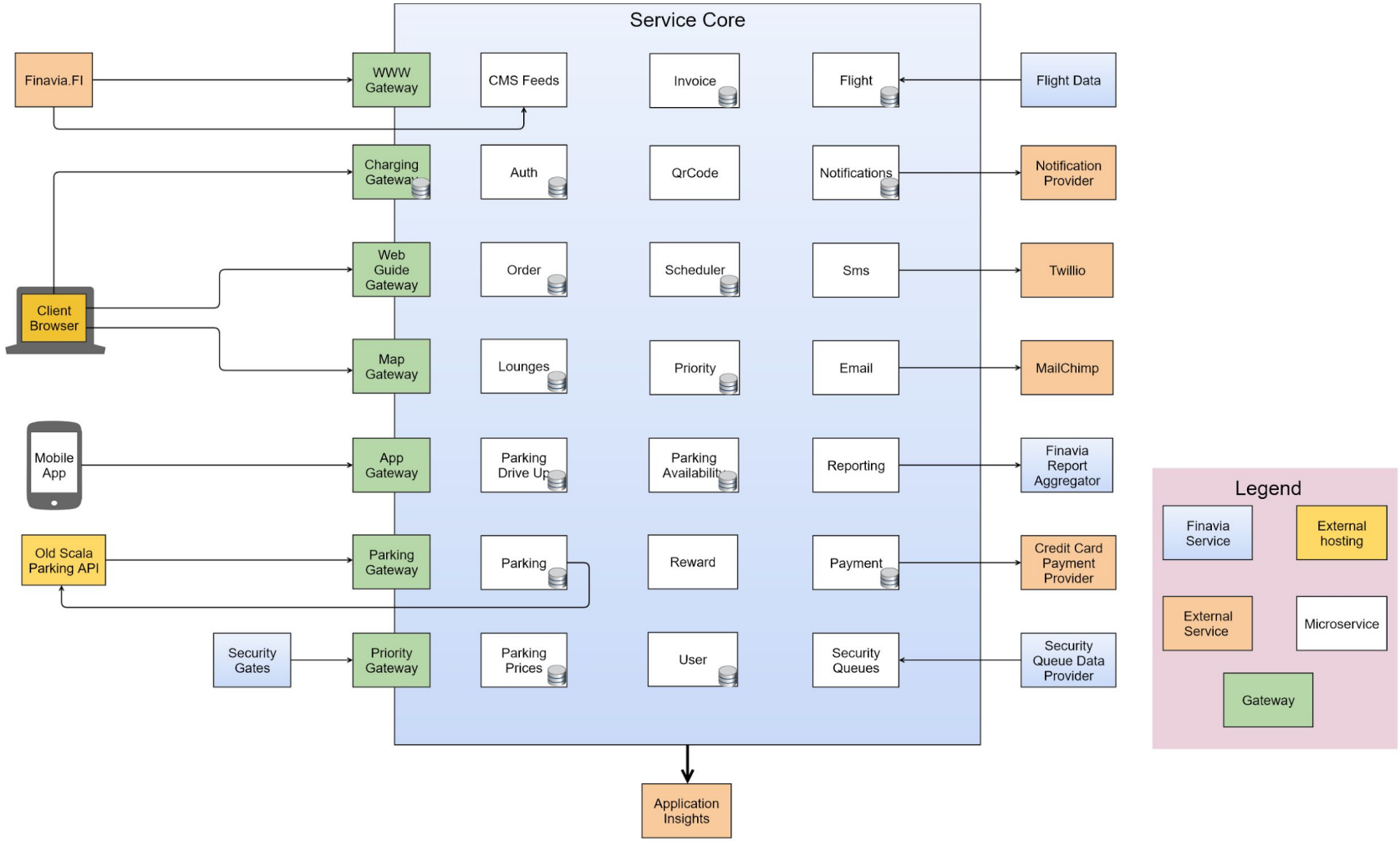
- Check for a new file
- Download the file from a 3rd party system
- **Parse the file (~10Mb)**
- Apply application logic
- Save to database
- Repeat previous steps every 5 min

Choose area and additional services

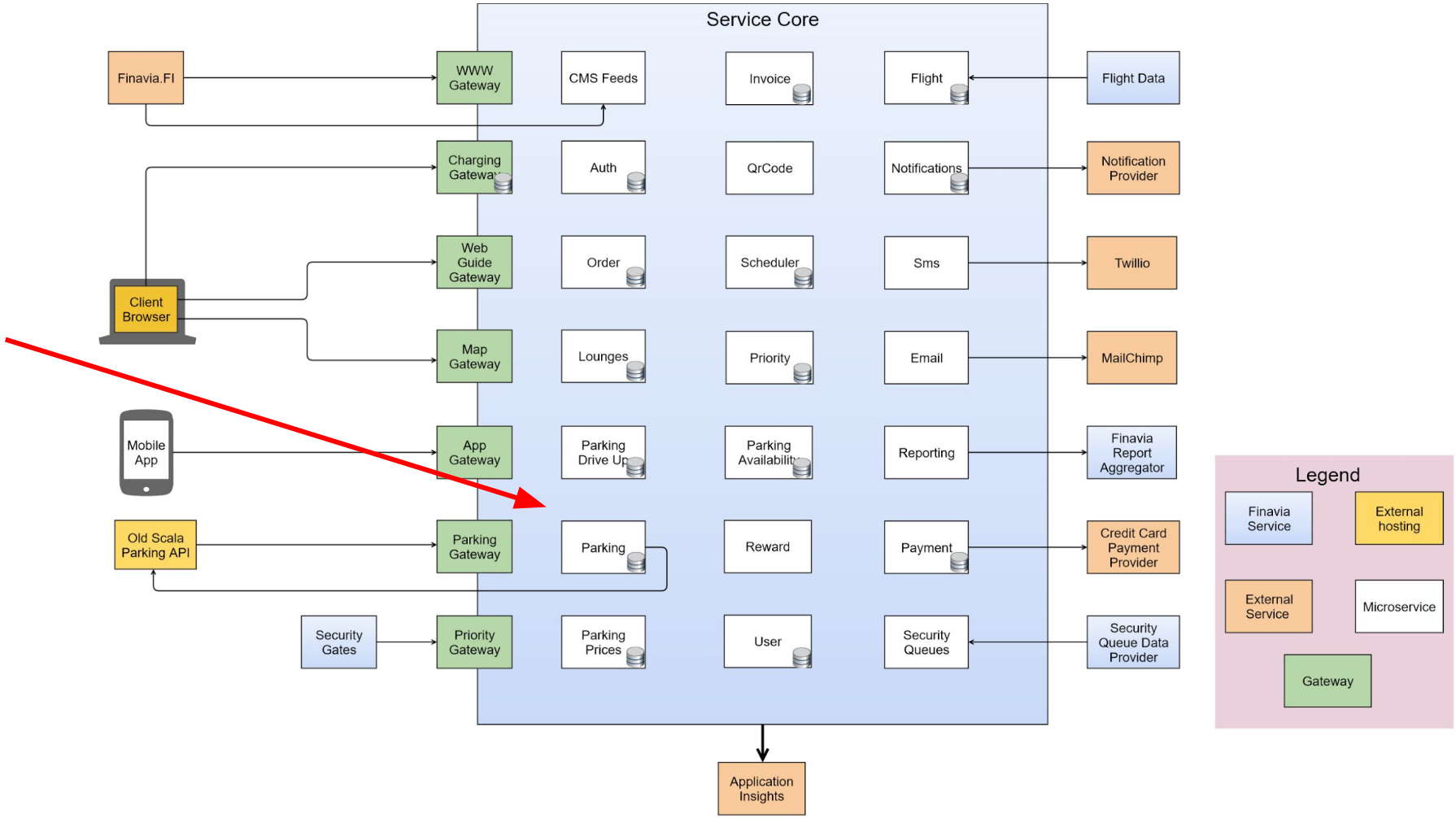
You can also reserve and pay car cleaning services.

P3 PREM	50 m walk to T1	ONLINE PRICE ⓘ
Available	300 m walk to T2	~~148€~~ 143€
P3	50 m walk to T1	ONLINE PRICE ⓘ
Available	300 m walk to T2	~~79€~~ 74€
P4A	750 m walk to T1	ONLINE PRICE ⓘ
Available	950 m walk to T2	~~47€~~ 40€

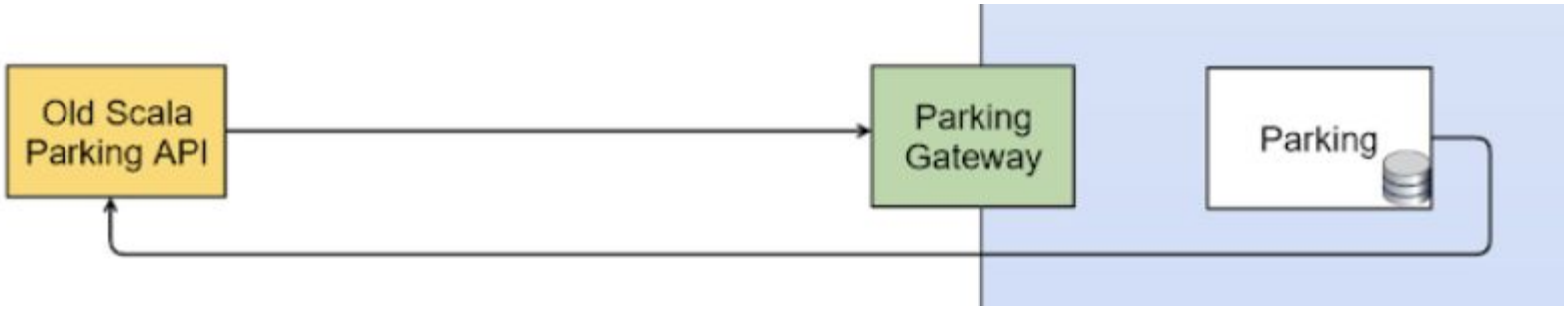
Backend (Service Core)



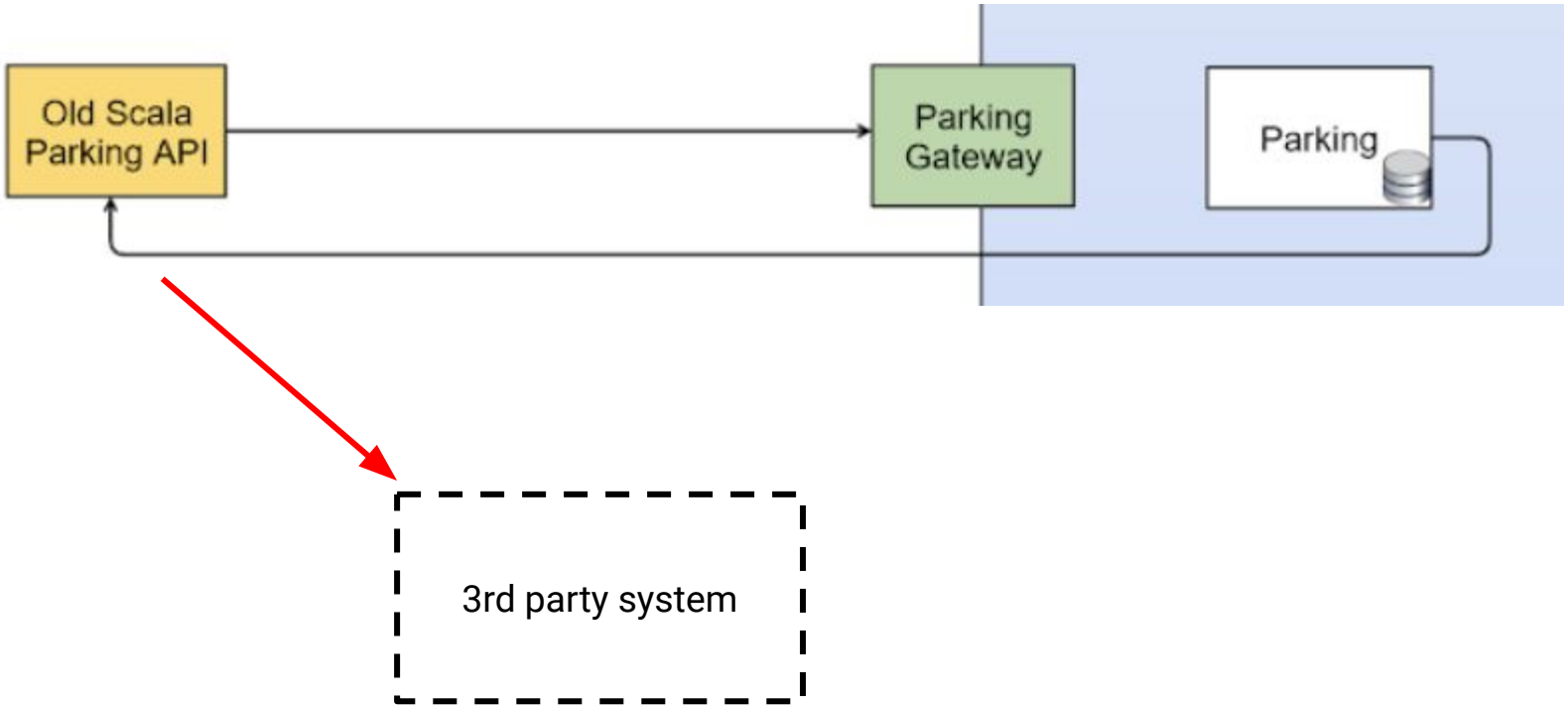
Backend (Service Core)



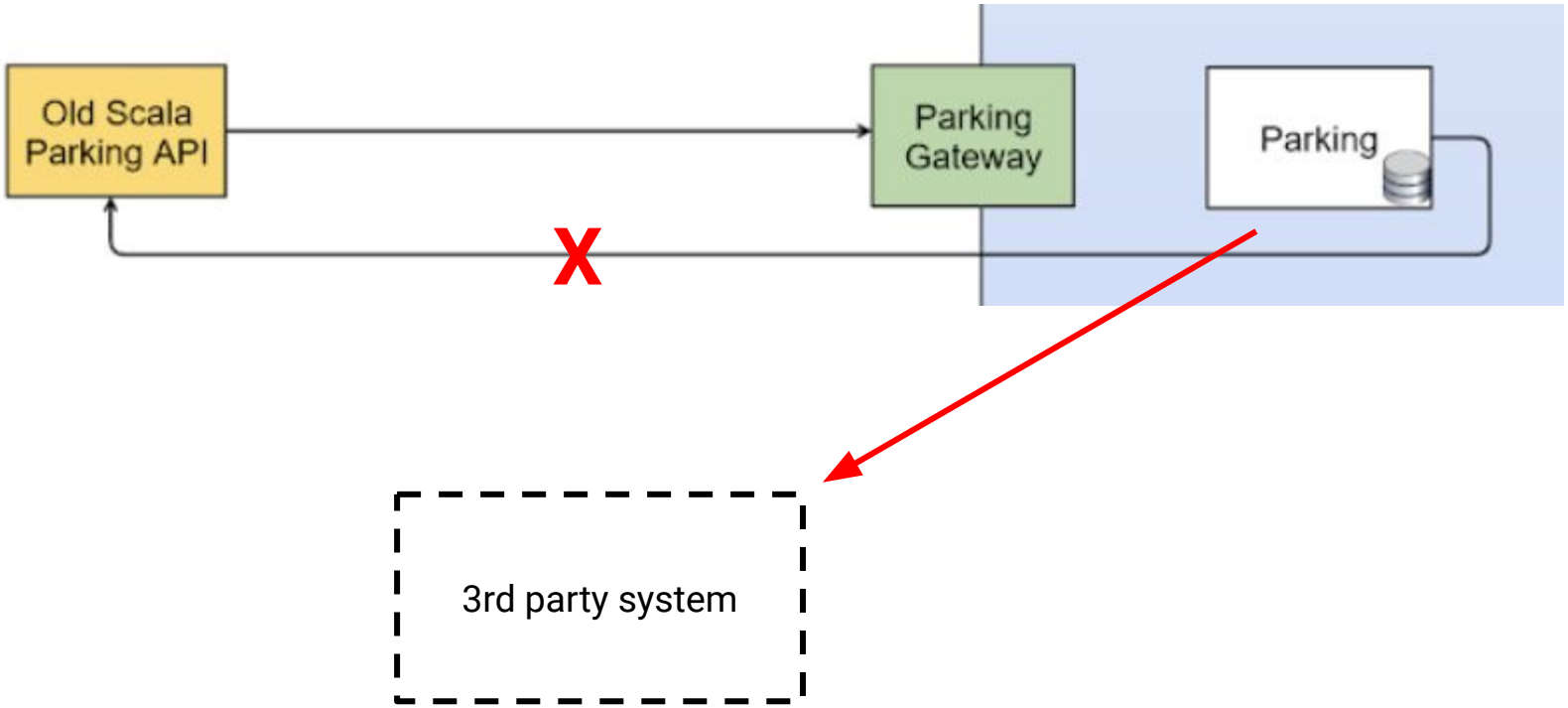
Backend (Service Core)



Backend (Service Core)



Backend (Service Core)



A File Structure (Demo Example)

```
38e27dea-1d7d-4279-be97-e29d53a8af89|Into the Breach|6|2018-02-27|90|False
6f3e9012-5d8c-43c4-b0d0-894fbff5a521|Football Manager 2020|5|2019-11-19|80|True
d79bbb41-f66a-46e9-b4d3-72295cca8324|The Witcher 3: Wild Hunt|3|2015-05-19|95|False
// ...
```

⇓

```csharp
public class Videogame
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public Genres Genre { get; set; }
    public DateTime ReleaseDate { get; set; }
    public int Rating { get; set; }
    public bool HasMultiplayer { get; set; }
}
```

```csharp
public enum Genres
{
    Action = 1,
    Adventure = 2,
    Rpg = 3,
    Shooter = 4,
    Sports = 5,
    Strategy = 6
}
```

First Approach: LineParser

```
public class LineParser
{
  public Videogame Parse(string line)
  {
    var parts = line.Split('|');

    return new Videogame
    {
      Id = Guid.Parse(parts[0]),
      Name = parts[1],
      Genre = Enum.Parse<Genres>(parts[2]),
      ReleaseDate = DateTime.ParseExact(
        parts[3], "yyyy-MM-dd",
        DateTimeFormatInfo.InvariantInfo, DateTimeStyles.None),
      Rating = int.Parse(parts[4]),
      HasMultiplayer = bool.Parse(parts[5])
    };
  }
}
```

First Approach: FileParser

```
public class FileParser
{
  // Initializing _lineParser from the constructor

  public async Task<List<Videogame>> Parse(string file)
  {
    var videogames = new List<Videogame>();

    using (var stream = File.OpenRead(file))
    using (var reader = new StreamReader(stream))
    {
      while (!reader.EndOfStream)
      {
        var line = await reader.ReadLineAsync();
        var videogame = _lineParser.Parse(line);
        videogames.Add(videogame);
      }
    }

    return videogames;
  }
}
```

First Approach: Possible Performance Issue

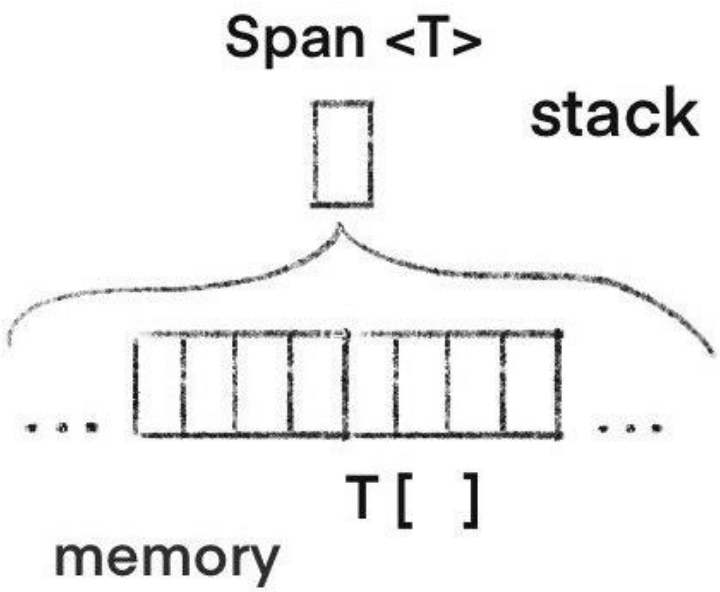
A lot of string allocations

```
public class LineParser
{
  public Videogame Parse(string line)
  {
    var parts = line.Split('|');

    return new Videogame
    {
      Id = Guid.Parse(parts[0]),
      Name = parts[1],
      Genre = Enum.Parse<Genres>(parts[2]),
      ReleaseDate = DateTime.ParseExact(
        parts[3], "yyyy-MM-dd",
        DateTimeFormatInfo.InvariantInfo, DateTimeStyles.None),
      Rating = int.Parse(parts[4]),
      HasMultiplayer = bool.Parse(parts[5])
    };
  }
}
```

```
public class FileParser
{
  // Initializing _lineParser from the constructor

  public async Task<List<Videogame>> Parse(string file)
  {
    var videogames = new List<Videogame>();

    using (var stream = File.OpenRead(file))
    using (var reader = new StreamReader(stream))
    {
      while (!reader.EndOfStream)
      {
        var line = await reader.ReadLineAsync();
        var videogame = _lineParser.Parse(line);
        videogames.Add(videogame);
      }
    }

    return videogames;
  }
}
```

Introducing Span<T>

- A struct that is allocated on the stack, but can point to data that's stored either on the stack or on the heap
- Is an abstraction over a contiguous region of arbitrary memory
- Has some limitations (can't be boxed or can't be used in async/await or yield methods)



https://docs.microsoft.com/en-us/dotnet/api/system.span-1

Second Approach: LineParserSpans (1)

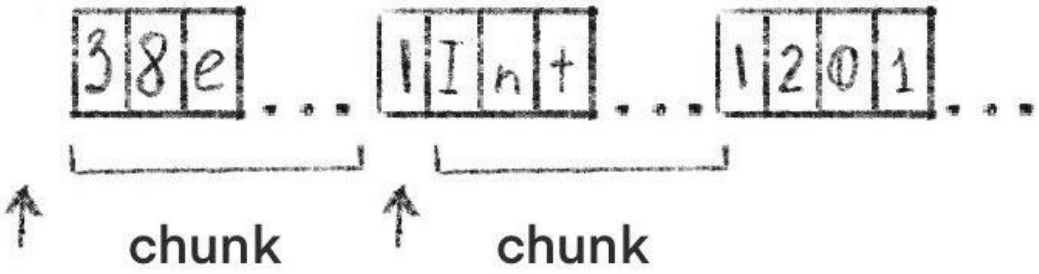
```
public class LineParserSpans
{
  public Videogame Parse(string line)
  {
    var span = line.AsSpan();

    return Parse(span);
  }

  private static Videogame Parse(ReadOnlySpan<char> span)
  {
    // ...
  }

  private static ReadOnlySpan<char> ParseChunk(
    ref ReadOnlySpan<char> span,
    ref int scanned,
    ref int position)
  {
    ...
  }
}
```

Second Approach: LineParserSpans (2)

```
private Videogame Parse(ReadOnlySpan<char> span)
{
  // Don't worry, we will increment this value in ParseChunk
  var scanned = -1;
  var position = 0;

  var id = ParseChunk(ref span, ref scanned, ref position);
  var name = ParseChunk(ref span, ref scanned, ref position);
  var genre = ParseChunk(ref span, ref scanned, ref position);
  var releaseDate = ParseChunk(ref span, ref scanned, ref position);
  var rating = ParseChunk(ref span, ref scanned, ref position);
  var hasMultiplayer = ParseChunk(ref span, ref scanned, ref position);

  return new Videogame
  {
    Id = Guid.Parse(id),
    Name = name.ToString(),
    Genre = (Genres)int.Parse(genre),
    ReleaseDate = DateTime.ParseExact(releaseDate,"yyyy-MM-dd",
      DateTimeFormatInfo.InvariantInfo),
    Rating = int.Parse(rating),
    HasMultiplayer = bool.Parse(hasMultiplayer)
  };
}
```

Second Approach: LineParserSpans (3)

```
private ReadOnlySpan<char> ParseChunk(
  ref ReadOnlySpan<char> span,
  ref int scanned,
  ref int position)
{
  scanned += position + 1;

  position = span.Slice(scanned, span.Length - scanned).IndexOf('|');
  if (position < 0)
  {
    position = span.Slice(scanned, span.Length - scanned).Length;
  }

  return span.Slice(scanned, position);
}
```



Second Approach: Same FileParser

```
public class FileParser
{
  // Initialize _lineParser as LineParserSpans

  public async Task<List<Videogame>> Parse(string file)
  {
    var videogames = new List<Videogame>();

    using (var stream = File.OpenRead(file))
    using (var reader = new StreamReader(stream))
    {
      while (!reader.EndOfStream)
      {
        var line = await reader.ReadLineAsync();
        var videogame = _lineParser.Parse(line);
        videogames.Add(videogame);
      }
    }

    return videogames;
  }
}
```

Benchmark (Single Line): LineParser vs LineParserSpans

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
LineParser	756.0 ns	14.61 ns	17.39 ns	0.1945	-	-	408 B
LineParserSpans	581.2 ns	12.61 ns	17.68 ns	0.0496	-	-	104 B

Spec:

- macOS Mojave 10.14.6
- Intel Core i7-7567U CPU 3.50GHz (Kaby Lake)
- .NET Core SDK=3.0.100

Benchmark (500k Line File): FileParser vs FileParserSpans

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
FileParser	1,156.2 ms	8.31 ms	7.77 ms	114000.0000	40000.0000	5000.0000	375.13 MB
FileParserSpans	862.6 ms	16.51 ms	16.22 ms	50000.0000	15000.0000	4000.0000	230.51 MB

Spec:

- macOS Mojave 10.14.6
- Intel Core i7-7567U CPU 3.50GHz (Kaby Lake)
- .NET Core SDK=3.0.100

Second Approach: Possible Performance Issue

```
public Videogame Parse(string line)
{
  var span = line.AsSpan();

  return Parse(span);
}
```

```
private ReadOnlySpan<char> ParseChunk(
  ref ReadOnlySpan<char> span,
  ref int scanned,
  ref int position)
{
  scanned += position + 1;

  position = span.Slice(scanned, span.Length - scanned).IndexOf('|');
  if (position < 0)
  {
    position = span.Slice(scanned, span.Length - scanned).Length;
  }

  return span.Slice(scanned, position);
}
```

```
private Videogame Parse(ReadOnlySpan<char> span)
{
  // Don't worry, we will increment this value in ParseChunk
  var scanned = -1;
  var position = 0;

  var id = ParseChunk(ref span, ref scanned, ref position);
  var name = ParseChunk(ref span, ref scanned, ref position);
  var genre = ParseChunk(ref span, ref scanned, ref position);
  var releaseDate = ParseChunk(ref span, ref scanned, ref position);
  var rating = ParseChunk(ref span, ref scanned, ref position);
  var hasMultiplayer = ParseChunk(ref span, ref scanned, ref position);

  return new Videogame
  {
    Id = Guid.Parse(id),
    Name = name.ToString(),
    Genre = (Genres)int.Parse(genre),
    ReleaseDate = DateTime.ParseExact(releaseDate,"yyyy-MM-dd",
      DateTimeFormatInfo.InvariantInfo),
    Rating = int.Parse(rating),
    HasMultiplayer = bool.Parse(hasMultiplayer)
  };
}
```

Second Approach: Possible Performance Issue

We still work with strings

```
public Videogame Parse(string line)
{
    var span = line.AsSpan();

    return Parse(span);
}
```

```
private ReadOnlySpan<char> ParseChunk(
    ref ReadOnlySpan<char> span,
    ref int scanned,
    ref int position)
{
    scanned += position + 1;

    position = span.Slice(scanned, span.Length - scanned).IndexOf('|');
    if (position < 0)
    {
        position = span.Slice(scanned, span.Length - scanned).Length;
    }

    return span.Slice(scanned, position);
}
```

```
private Videogame Parse(ReadOnlySpan<char> span)
{
    // Don't worry, we will increment this value in ParseChunk
    var scanned = -1;
    var position = 0;

    var id = ParseChunk(ref span, ref scanned, ref position);
    var name = ParseChunk(ref span, ref scanned, ref position);
    var genre = ParseChunk(ref span, ref scanned, ref position);
    var releaseDate = ParseChunk(ref span, ref scanned, ref position);
    var rating = ParseChunk(ref span, ref scanned, ref position);
    var hasMultiplayer = ParseChunk(ref span, ref scanned, ref position);

    return new Videogame
    {
        Id = Guid.Parse(id),
        Name = name.ToString(),
        Genre = (Genres)int.Parse(genre),
        ReleaseDate = DateTime.ParseExact(releaseDate,"yyyy-MM-dd",
            DateTimeFormatInfo.InvariantInfo),
        Rating = int.Parse(rating),
        HasMultiplayer = bool.Parse(hasMultiplayer)
    };
}
```

Introducing System.IO.Pipelines

- Designed to make high performance IO operations in .NET Core
- Makes it easier to work with buffers
- Works like streams, but has a more clean API

```
var pipe = new Pipe();

// Producer
var message = Encoding.UTF8.GetBytes("Moi Tampere");
await pipe.Writer.WriteAsync(message);
pipe.Writer.Advance(message.Length);

// Consumer
var read = await pipe.Reader.ReadAsync();
await pipe.Reader.CompleteAsync();

// Print result
var result = Encoding.UTF8.GetString(read.Buffer.FirstSpan);
Console.WriteLine(result); // => Moi Tampere
```

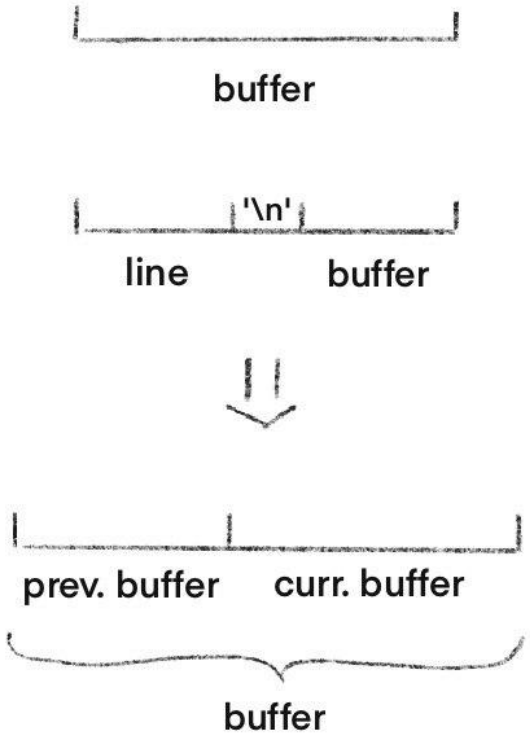
https://devblogs.microsoft.com/dotnet/system-io-pipelines-high-performance-io-in-net

Third Approach: FileParserSpansAndPipes (1)

```
public async Task<List<Videogame>> Parse(string file)
{
  var result = new List<Videogame>();
  using (var stream = File.OpenRead(file))
  {
    var reader = PipeReader.Create(stream);
    while (true)
    {
      var read = await reader.ReadAsync();
      var buffer = read.Buffer;
      while (TryReadLine(ref buffer, out ReadOnlySequence<byte> sequence))
      {
        var videogame = ProcessSequence(sequence);
        result.Add(videogame);
      }

      reader.AdvanceTo(buffer.Start, buffer.End);
      if (read.IsCompleted)
      {
          break;
      }
    }
  }
  return result;
}
```

Third Approach: FileParserSpansAndPipes (2)

```
private static bool TryReadLine(
    ref ReadOnlySequence<byte> buffer,
    out ReadOnlySequence<byte> line)
{
    var position = buffer.PositionOf((byte)'\n');
    if (position == null)
    {
        line = default;
        return false;
    }

    line = buffer.Slice(0, position.Value);
    buffer = buffer.Slice(buffer.GetPosition(1, position.Value));

    return true;
}
```



Third Approach: FileParserSpansAndPipes (3)

```
private static Videogame ProcessSequence(ReadOnlySequence<byte> sequence)
{
  if (sequence.IsSingleSegment)
  {
    return Parse(sequence.FirstSpan);
  }

  var length = (int) sequence.Length;
  Span<byte> span = stackalloc byte[(int)sequence.Length];
  sequence.CopyTo(span);

  return Parse(span);
}

private static Videogame Parse(ReadOnlySpan<byte> bytes)
{
  Span<char> chars = stackalloc char[bytes.Length];
  Encoding.UTF8.GetChars(bytes, chars);

  return LineParserSpans.Parse(chars);
}
```

Third Approach: FileParserSpansAndPipes (4)

```
private Videogame Parse(ReadOnlySpan<char> span)
{
  // Don't worry, we will increment this value in ParseChunk
  var scanned = -1;
  var position = 0;

  var id = ParseChunk(ref span, ref scanned, ref position);
  var name = ParseChunk(ref span, ref scanned, ref position);
  var genre = ParseChunk(ref span, ref scanned, ref position);
  var releaseDate = ParseChunk(ref span, ref scanned, ref position);
  var rating = ParseChunk(ref span, ref scanned, ref position);
  var hasMultiplayer = ParseChunk(ref span, ref scanned, ref position);

  return new Videogame
  {
    Id = Guid.Parse(id),
    Name = name.ToString(),
    Genre = (Genres)int.Parse(genre),
    ReleaseDate = DateTime.ParseExact(releaseDate,"yyyy-MM-dd",
      DateTimeFormatInfo.InvariantInfo),
    Rating = int.Parse(rating),
    HasMultiplayer = bool.Parse(hasMultiplayer)
  };
}
```

Benchmark (500k Line File)

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
FileParser	1,378.5 ms	15.78 ms	14.76 ms	133000.0000	60000.0000	25000.0000	375.34 MB
FileParserSpans	905.5 ms	17.71 ms	25.96 ms	52000.0000	18000.0000	8000.0000	230.56 MB
FileParserSpansAndPipes	640.5 ms	7.69 ms	6.82 ms	15000.0000	7000.0000	3000.0000	78.77 MB

Thank you!

Questions?

Contact Information

Tim Iskhakov

- Email: tim.iskhakov@futurice.com
- Blog: https://timiskhakov.github.io