CONCURRENCY PATTERNS

Pipelines

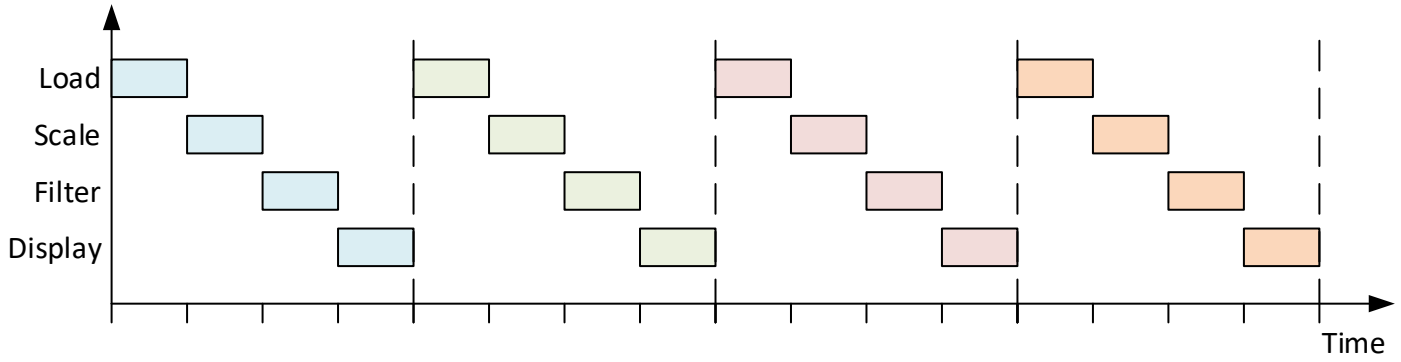
PIPELINES

- The Pipeline pattern parallelizes the processing of a sequence of input values.
- A pipeline consists of a series of producer/consumer stages (filters), connected by queues (pipes).
- Divide processing into parallelizable stages, where…
  - Output of stage `i` is input for stage `i+1`
  - Stages are otherwise independent

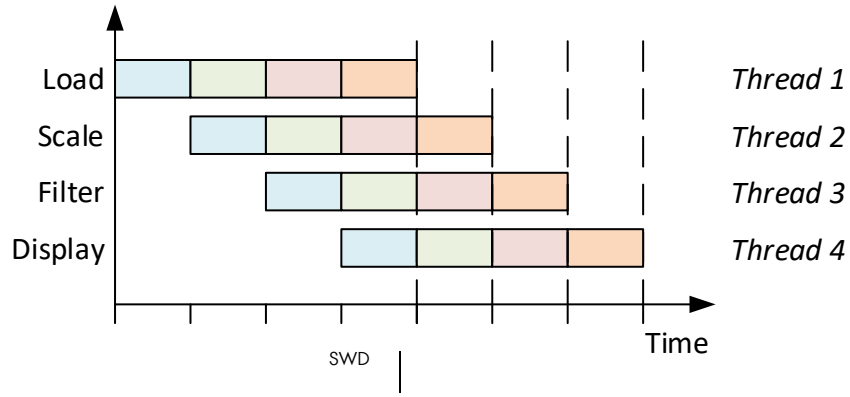
PIPELINING

- Producing thumbnails of images in stages:
  - Load image > scale image > filter image > display image

Sequential
(4 images)



Pipelined –
throughput * 4

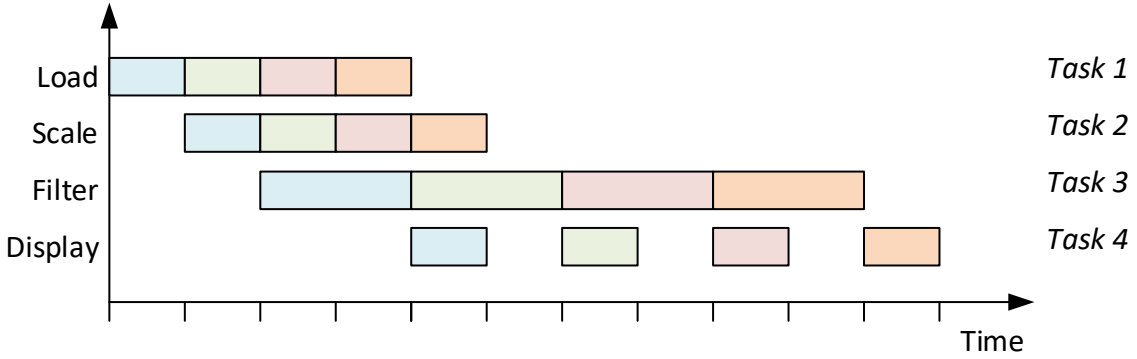


AARHUS
UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER
ENGINEERING

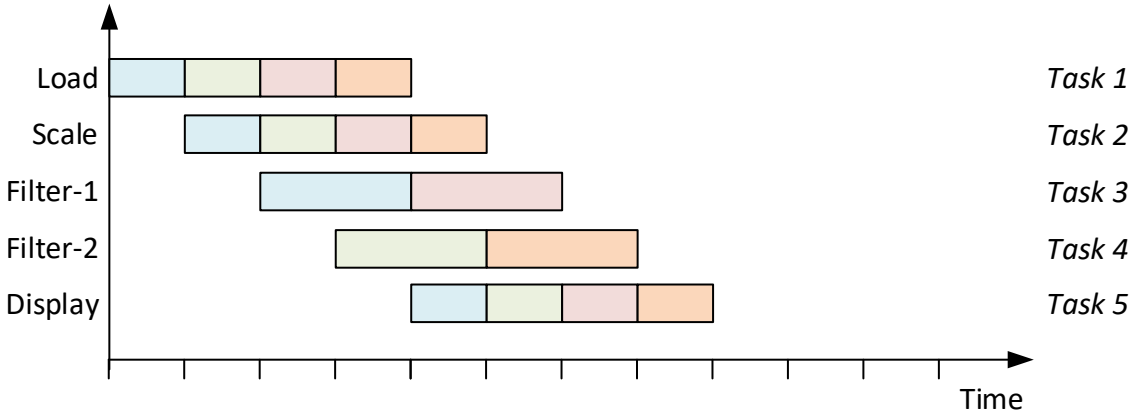
PIPELINING – UNEVEN STAGE DURATION

- When steps are of unequal duration, the pipeline may be duplicated (parallel) for the bottleneck

Problem: Filter stage becomes bottleneck



Solution: Several Filter stages to feed Display



EXAMPLES

- Audio and video processing
  - gstreamer framework
- Visual processing
  - autonomous robots
- "Real-time"/live processing
  - Stock market data

PIPELINE PATTERN IN C#

In C#, pipelines are created using tasks and concurrent queues (`BlockingCollection<T>`)

```
void DoStage(BlockingCollection<T> input,
             BlockingCollection<T> output)
  try
  {
     foreach (var item in input.GetConsumingEnumerable())
     {
        var result = ...
        output.Add(result);
     }
  }
  finally
  {
     output.CompleteAdding();
  }
}
```

Get an iterator for the blocking collection – very important!

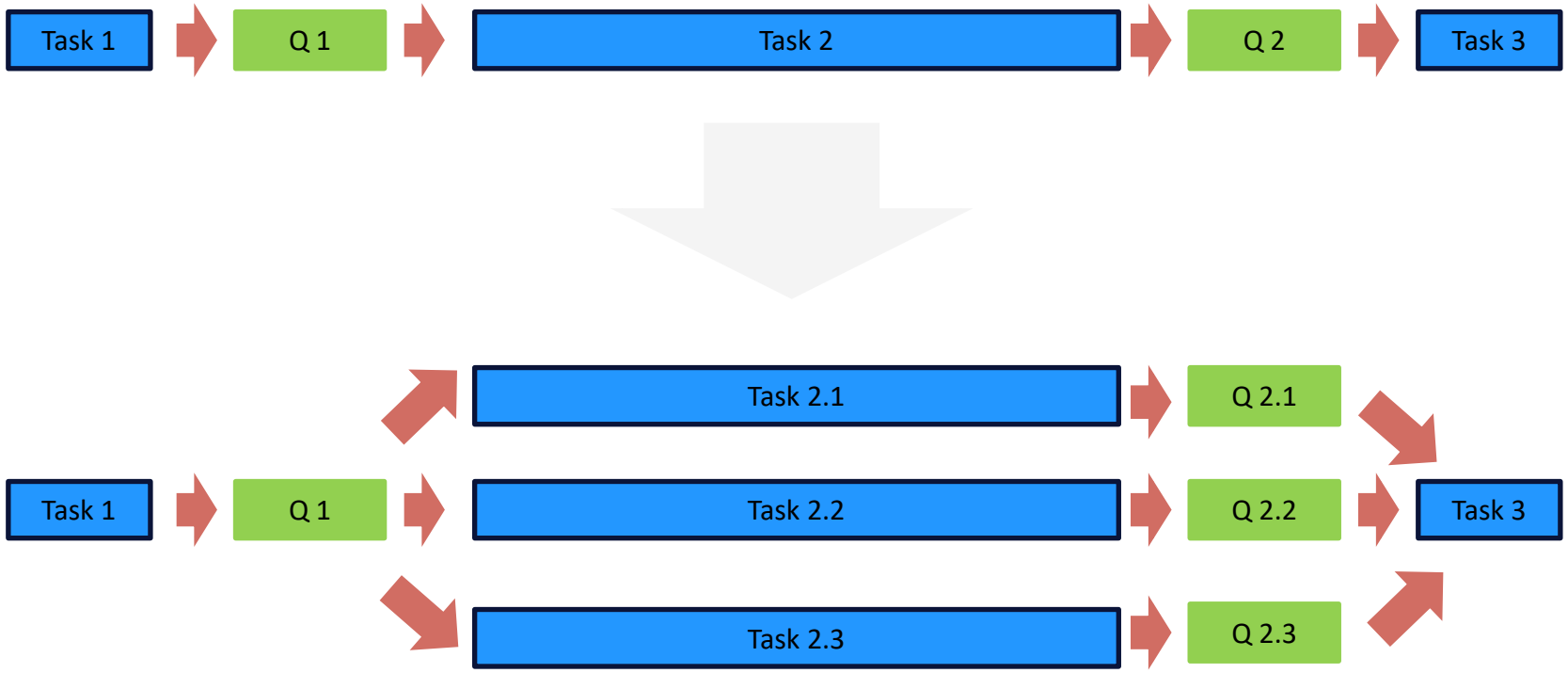
Generate a result, add it to the output

When all output addition is done: Signal the completion of adding

What is the purpose of `CompleteAdding()`?

PIPELINE IMPLEMENTATION – UNEVEN STAGE DURATION

When stages are of uneven length, use several stages (tasks) in parallel:



PIPELINE IMPLEMENTAION – UNEVEN STAGE DURATION

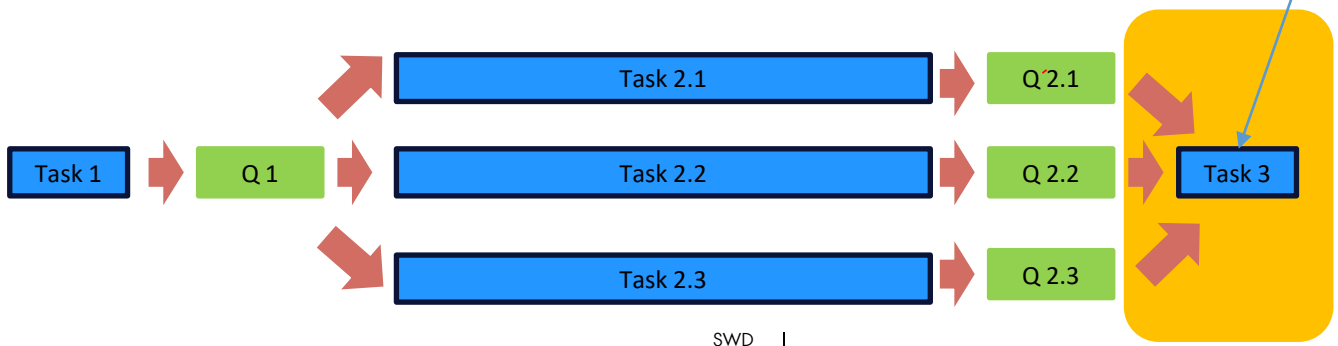
- Note that the consumer stage (not the producer stage) has to change

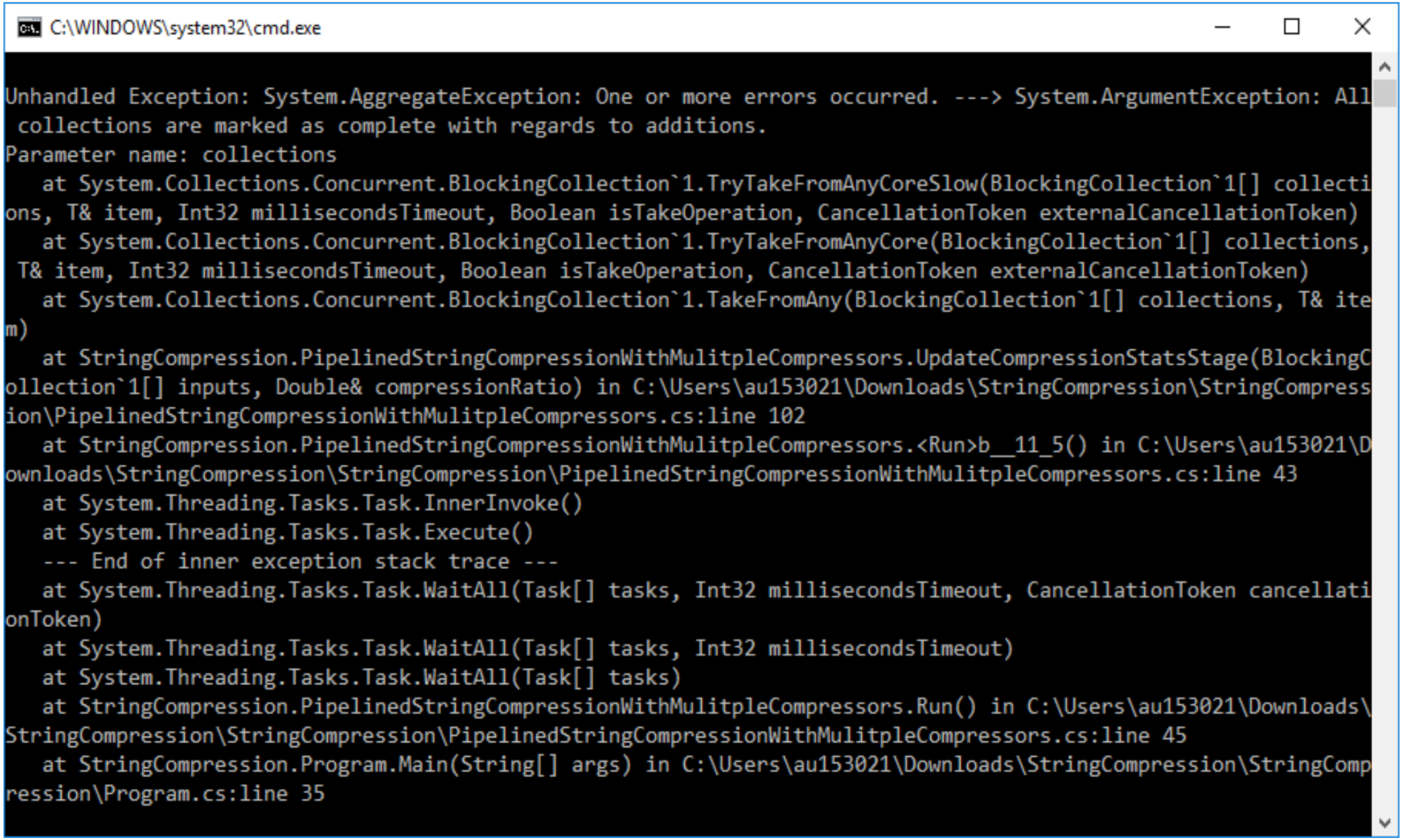
```
private void ToLowerCase(BlockingCollection<string>[] inputs, BlockingCollection<string> output)
{
    var str = "";

    while(!inputs.All(bc=> bc.IsCompleted))
    {
        BlockingCollection<string>.TakeFromAny(inputs, out str);
        str = str.ToLower();
        output.Add(str);
    }
    output.CompleteAdding();
}
```

TakeFromAny() finds a ”non-completed”, data-bearing collection and retrieves data from it

Will crash with an exception if all collections are Complete. How can that happen?



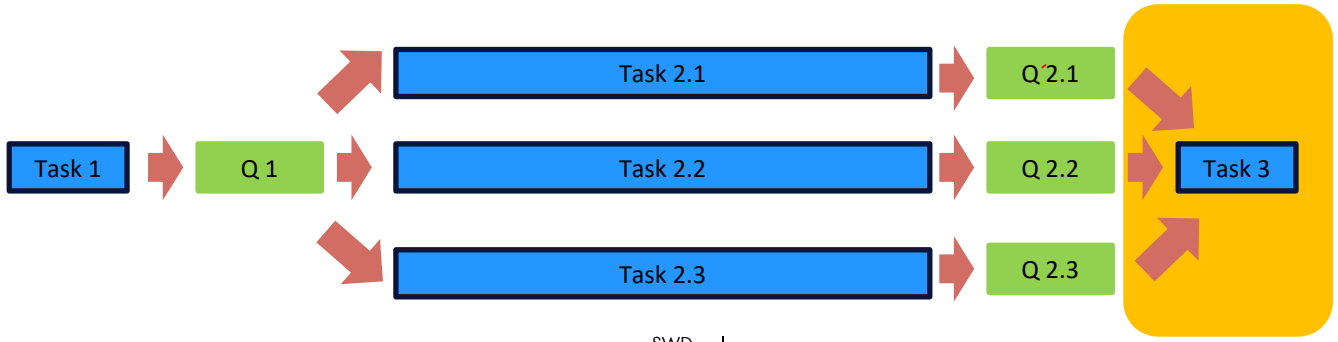

C:\WINDOWS\system32\cmd.exe
Unhandled Exception: System.AggregateException: One or more errors occurred. ---> System.ArgumentException: All collections are marked as complete with regards to additions.
Parameter name: collections
at System.Collections.Concurrent.BlockingCollection`1.TryTakeFromAnyCoreSlow(BlockingCollection`1[] collections, T& item, Int32 millisecondsTimeout, Boolean isTakeOperation, CancellationToken externalCancellationToken)
at System.Collections.Concurrent.BlockingCollection`1.TryTakeFromAnyCore(BlockingCollection`1[] collections, T& item, Int32 millisecondsTimeout, Boolean isTakeOperation, CancellationToken externalCancellationToken)
at System.Collections.Concurrent.BlockingCollection`1.TakeFromAny(BlockingCollection`1[] collections, T& item)
at StringCompression.PipelinedStringCompressionWithMulitpleCompressors.UpdateCompressionStatsStage(BlockingCollection`1[] inputs, Double& compressionRatio) in C:\Users\au153021\Downloads\StringCompression\StringCompression\PipelinedStringCompressionWithMulitpleCompressors.cs:line 102
at StringCompression.PipelinedStringCompressionWithMulitpleCompressors.<Run>b__11_5() in C:\Users\au153021\Downloads\StringCompression\StringCompression\PipelinedStringCompressionWithMulitpleCompressors.cs:line 43
at System.Threading.Tasks.Task.InnerInvoke()
at System.Threading.Tasks.Task.Execute()
--- End of inner exception stack trace ---
at System.Threading.Tasks.Task.WaitAll(Task[] tasks, Int32 millisecondsTimeout, CancellationToken cancellationToken)
at System.Threading.Tasks.Task.WaitAll(Task[] tasks, Int32 millisecondsTimeout)
at System.Threading.Tasks.Task.WaitAll(Task[] tasks)
at StringCompression.PipelinedStringCompressionWithMulitpleCompressors.Run() in C:\Users\au153021\Downloads\StringCompression\StringCompression\PipelinedStringCompressionWithMulitpleCompressors.cs:line 45
at StringCompression.Program.Main(String[] args) in C:\Users\au153021\Downloads\StringCompression\StringCompression\Program.cs:line 35

PIPELINE IMPLEMENTATION – UNEVEN STAGE DURATION

- Note that the consumer stage (not the producer stage) has to change

```
private void ToLowerCase(BlockingCollection<string>[] inputs, BlockingCollection<string> output)
{
    var str = "";

    while(!inputs.All(bc=> bc.IsCompleted))
    {
        if (BlockingCollection<string>.TryTakeFromAny(inputs, out str) != -1)
        {
          str = str.ToLower();
          output.Add(str);
        }
    }
    output.CompleteAdding();
}
```



DETECTING UNEVEN STAGE DURATION

- Test and time the filters individually – the design is very easy to test
- Check the queue lengths during runtime from main to find the bottleneck(s)
- Replace the other filters with dummies only forwarding data, and time the whole thing

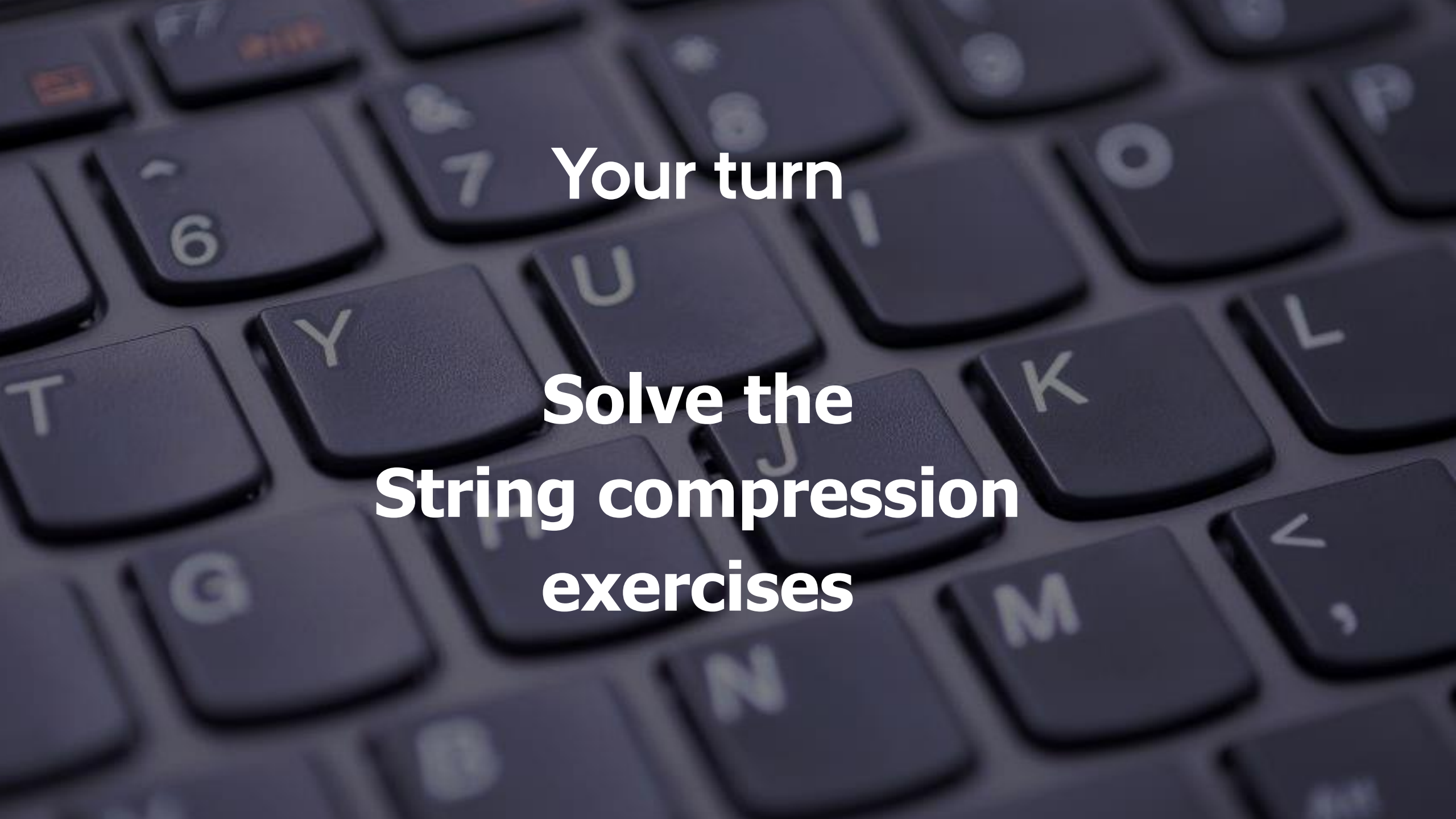
PIPELINE CANCELATION

- Canceling a Pipeline with Cancellation Token

```
private void ToLowerCase(BlockingCollection<string> input, BlockingCollection<string> output,
                                                    CancellationToken token)
{
    try {
        foreach(var item in input.GetConsumingEnumerable())
        {
            if (token.IsCancellationRequested) break;
            var result = ...
            output.Add(result, token);
        }
    } catch (OperationCanceledException) {}
    } finally { output.CompleteAdding() }
}
```

- `output.Add(result)` can block – with a `Cancellationtoken` it will throw an exception, when cancellation is set.

- if not cancelled, the program can deadlock



Your turn

Solve the String compression exercises

AARHUS
UNIVERSITY