


Pipes and Filters

Samuel Serafín



Definiti on

- An architectural pattern which decomposes a complex problem into small generic tasks, which can then be reused in different problems

Structure

Filter

Gets input data

Performs a function (enriching, refining or transformation) on its input data

Supplies output data

Data source(s)

Delivers input to processing pipeline

Pipe (Pipeline)

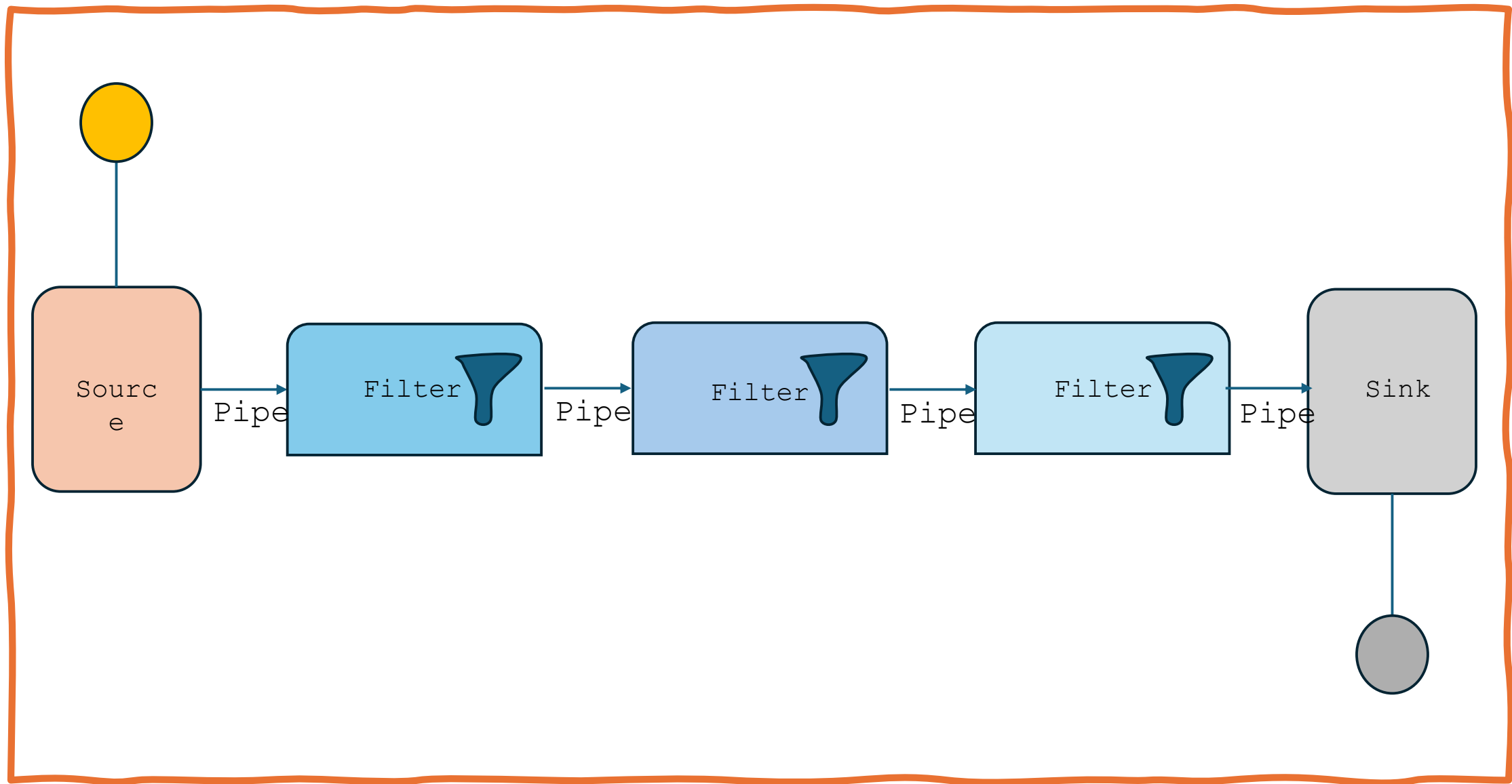
Transfers data

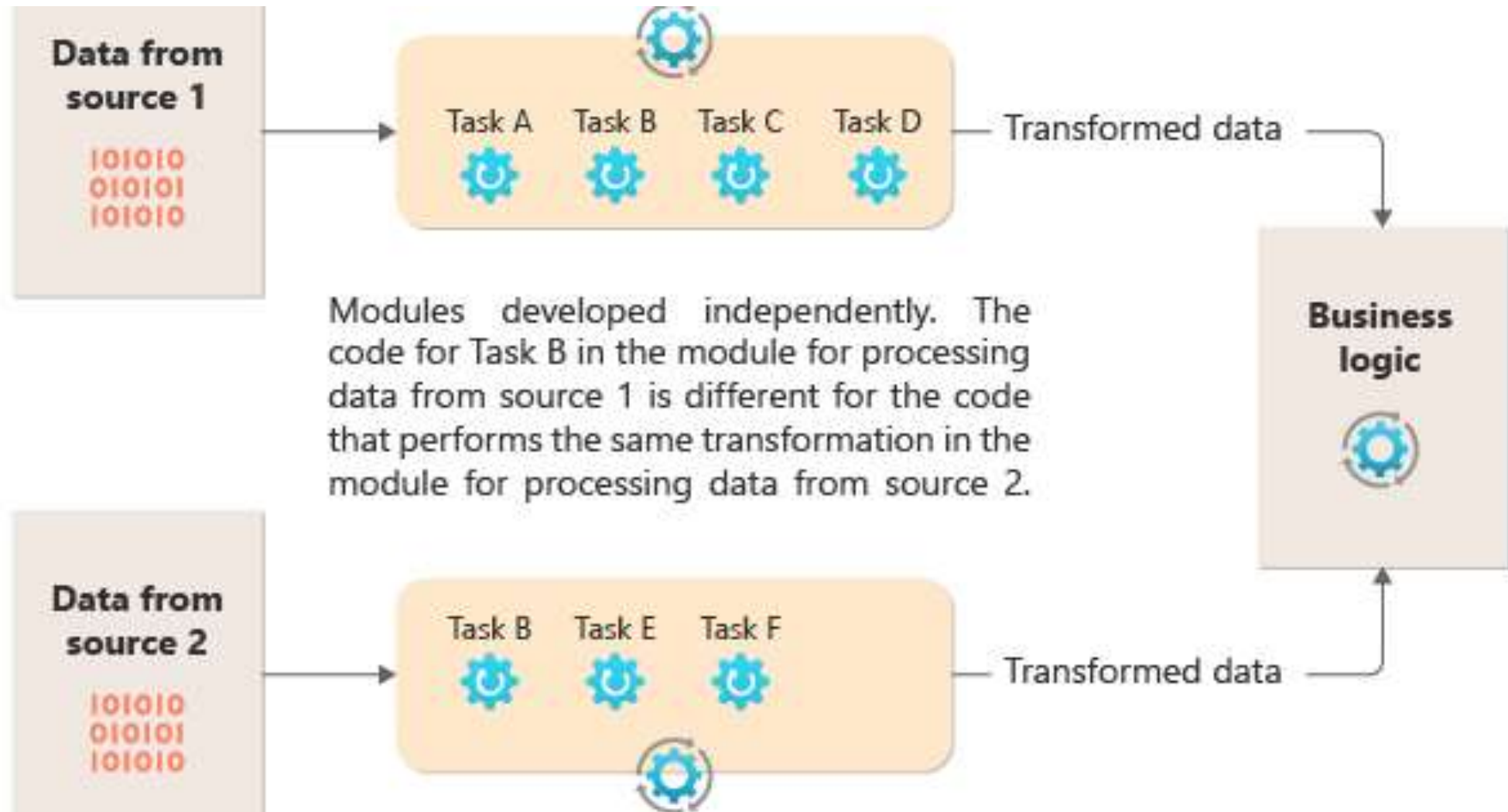
Buffers data

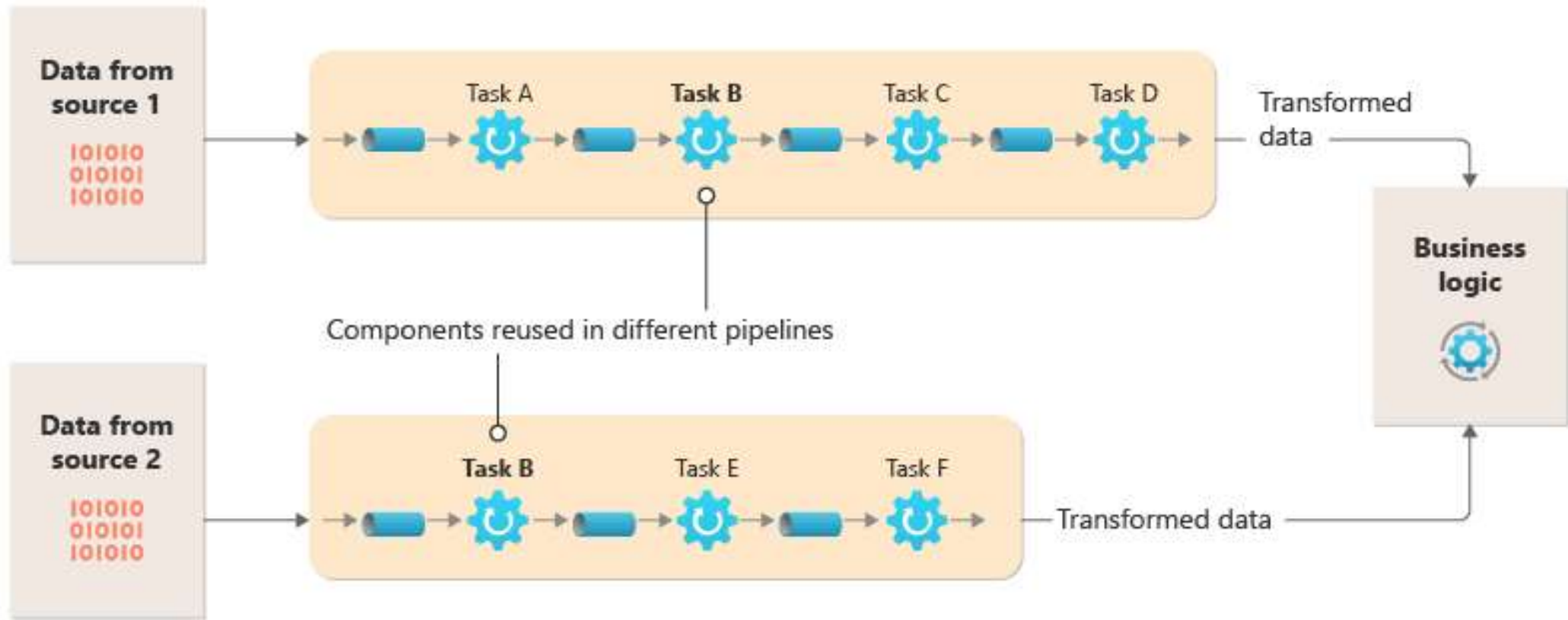
Synchronizes active neighbors

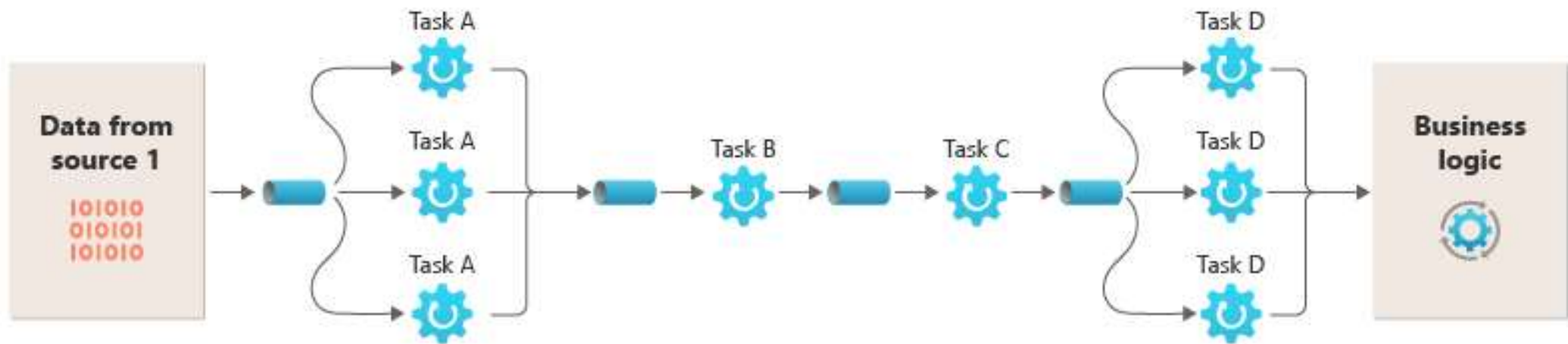
Data sink

Consumes output

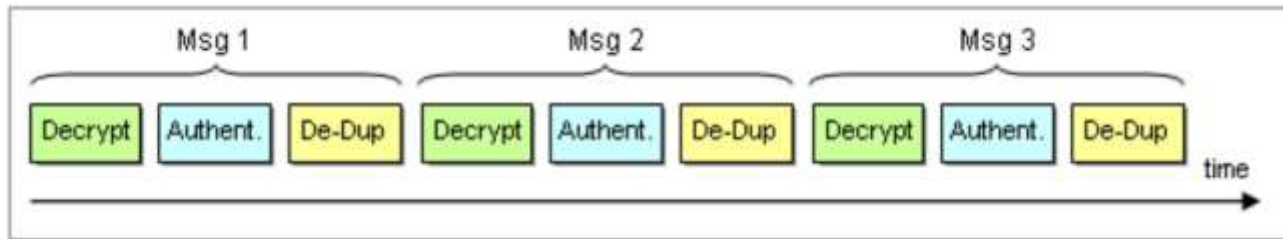




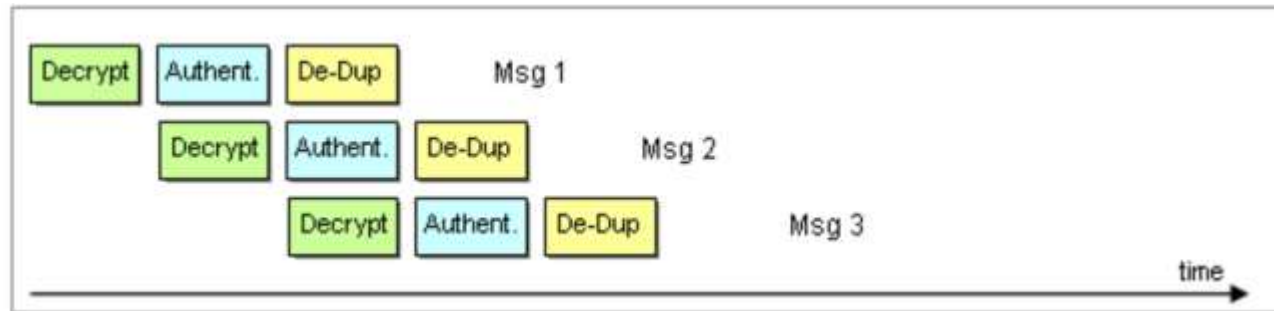




Sequential



Pipeline



Code example

```
public interface IFilter<T> { T Execute(T input); }
public abstract class Pipeline<T>
{
    protected List<IFilter<T>> Filters { get; set; } =
new();
    public Pipeline<T> Register(IFilter<T> filter)
    {
        Filters.Add(filter);
        return this;
    }
    public abstract T Process(T input);
}
public sealed class StringPipeline : Pipeline<string>
{
    public override string Process(string input) =>
        Filters.Aggregate(input, (current, filter)
=> filter.Execute(current));
}
```

```
public sealed class CapitalizeAllLettersFilter :
IFilter<string>
{
    public string Execute(string input) => input.ToUpper();
}
public sealed class RemoveSpacesFilter : IFilter<string>
{
    public string Execute(string input) => input.Replace(" ",
""");
}

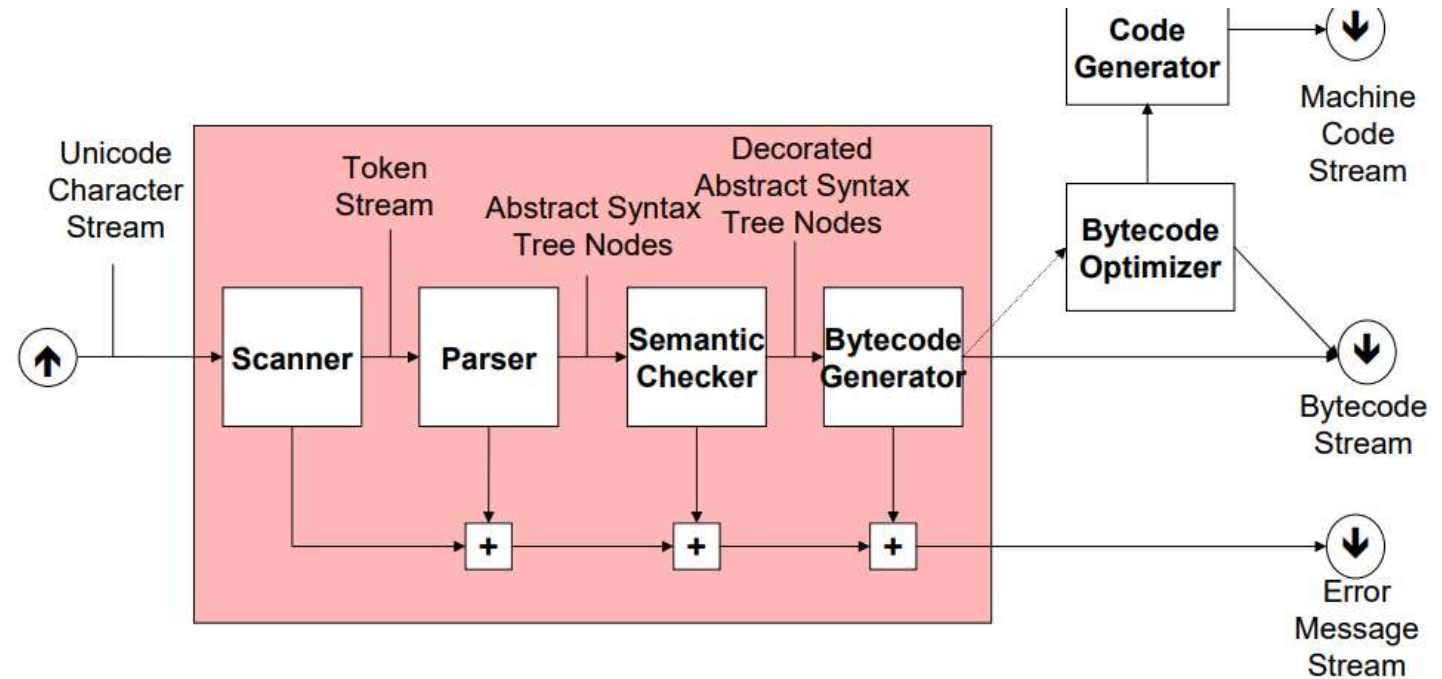
var pipeline = new StringPipeline();
pipeline.Register(new CapitalizeAllLettersFilter());
pipeline.Register(new RemoveSpacesFilter());
var output = pipeline.Process("a b c d e");
Console.WriteLine(output);
```

=> ABCDE

Usage (Further examples)

- UNIX pipes
- Compilers

```
stack@undercloud-0:~/test
File Edit View Search Terminal Help
[stack@undercloud-0 test]$ cat contents.txt | grep file
-rw-rw-r--. 1 stack stack 0 Aug  8 13:51 file1
-rw-rw-r--. 1 stack stack 0 Aug  8 13:51 file2
-rw-rw-r--. 1 stack stack 0 Aug  8 13:51 file3
-rw-rw-r--. 1 stack stack 0 Aug  8 13:51 file4
-rw-rw-r--. 1 stack stack 0 Aug  8 13:51 file5
[stack@undercloud-0 test]$ cat contents.txt | grep "file" | awk '{print $9}'
file1
file2
file3
file4
file5
[stack@undercloud-0 test]$ cat contents.txt | wc -l
9
```



Pros and Cons

+

Flexibility

Testability

Reusability

Scalability

Loose coupling of filters
(if using same type)

-

Error handling -> requires
a common strategy for all
the steps

Data transformation
overhead


State information might
need to be stored (if in
database then +load,
persist operations)

When to use

- The processing steps of the application have different scalability requirements
- The processing can be easily broken down into series of independent steps
- The system can benefit from distributing the processing for steps across different servers
- There is a requirement for flexibility, such as reordering, adding or removing steps in the processing
- There are procedures, which have common steps



When not to use

- 
- The application follows a request-response pattern
 - The processing steps are not independent or they have to be performed together
 - If the processing stages need to share a large amount of global data -> it becomes inflexible/expensive



Related patterns

- Layers
- Competing Consumers
- Chain of Responsibility



Relation to:

Layers

Both specialize in
decomposition into easier tasks

Lacks easy recombination and
reuse of elements

Easier error handling

Competing Consumers

Can be used together with Pipes
and Filters pattern

The Competing Consumers pattern
can distribute incoming tasks
or data chunks across multiple
instances of a processing
pipeline.

Sources

<https://learn.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters>

Pattern Oriented Software Architecture (Volume 1)

<https://www.enterpriseintegrationpatterns.com/patterns/messaging/PipesAndFilters.html>

[https://en.wikipedia.org/wiki/Pipeline_\(software\)](https://en.wikipedia.org/wiki/Pipeline_(software))