Filters and pipes

HOANG ANH TUAN

Definition

- Decompose complex problem into smaller reusable tasks

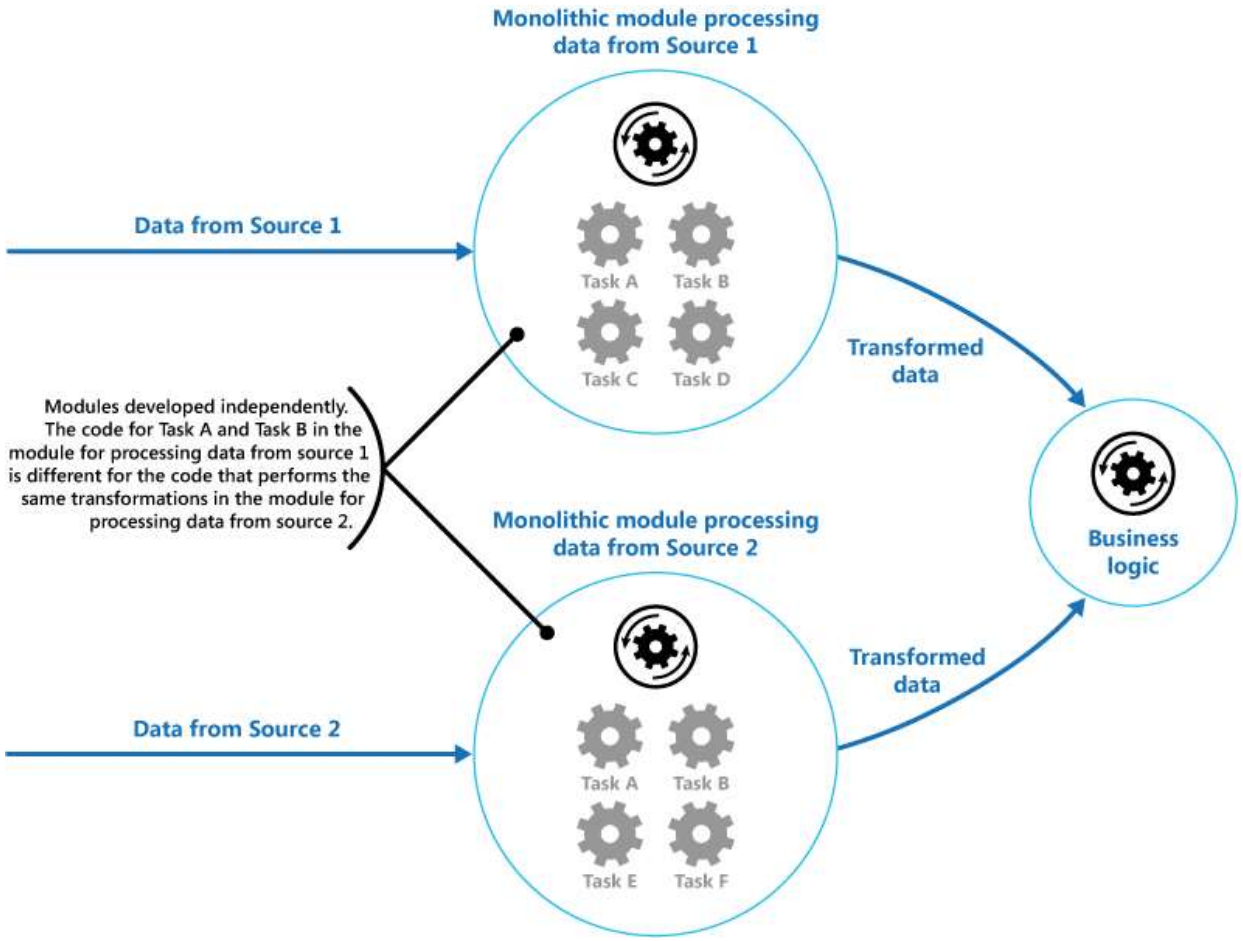
Filters

- Gets data
- Transforms data
- Returns data

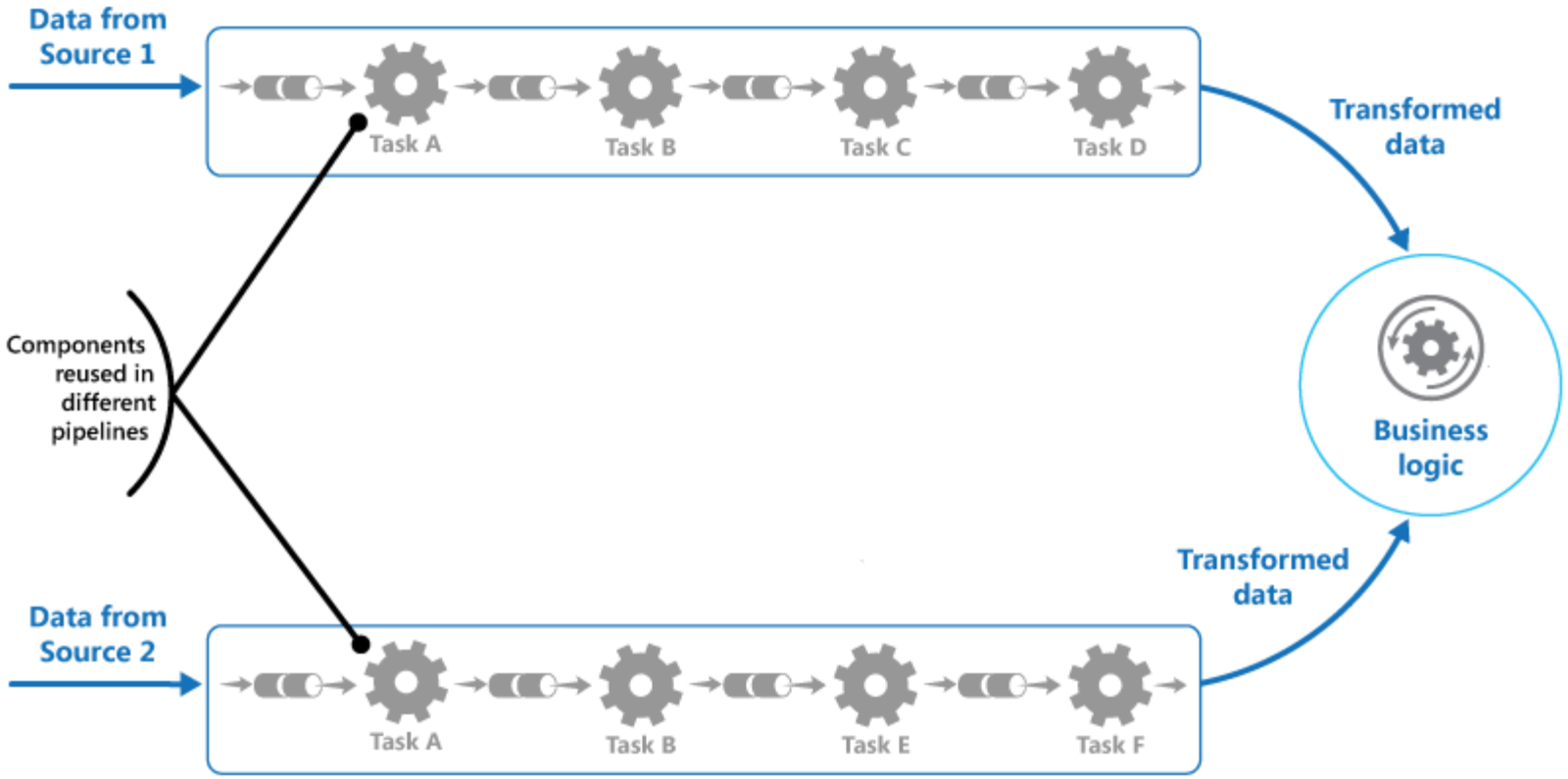
Pipes

- connectors for data flow

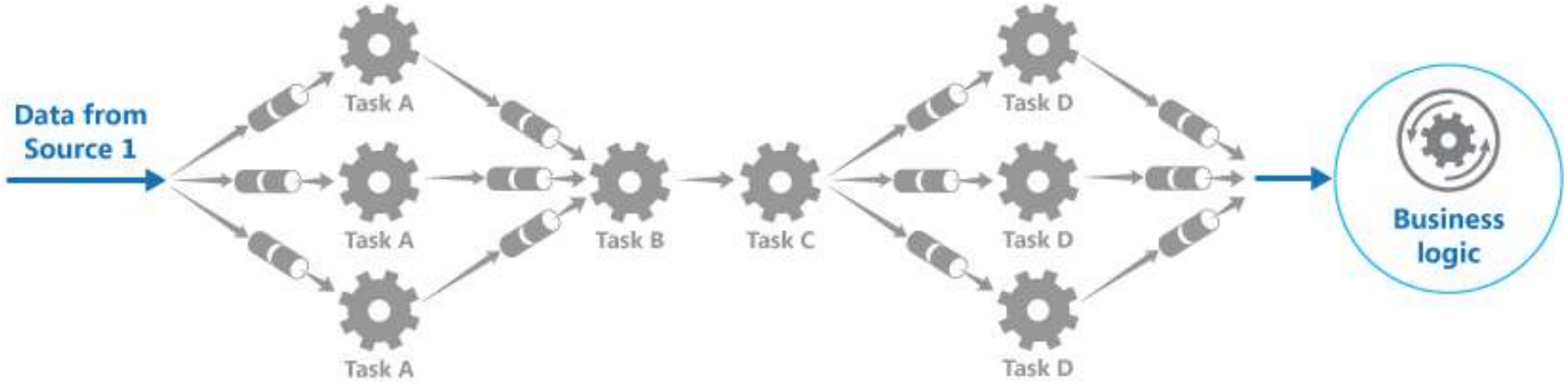
Motivation example #1



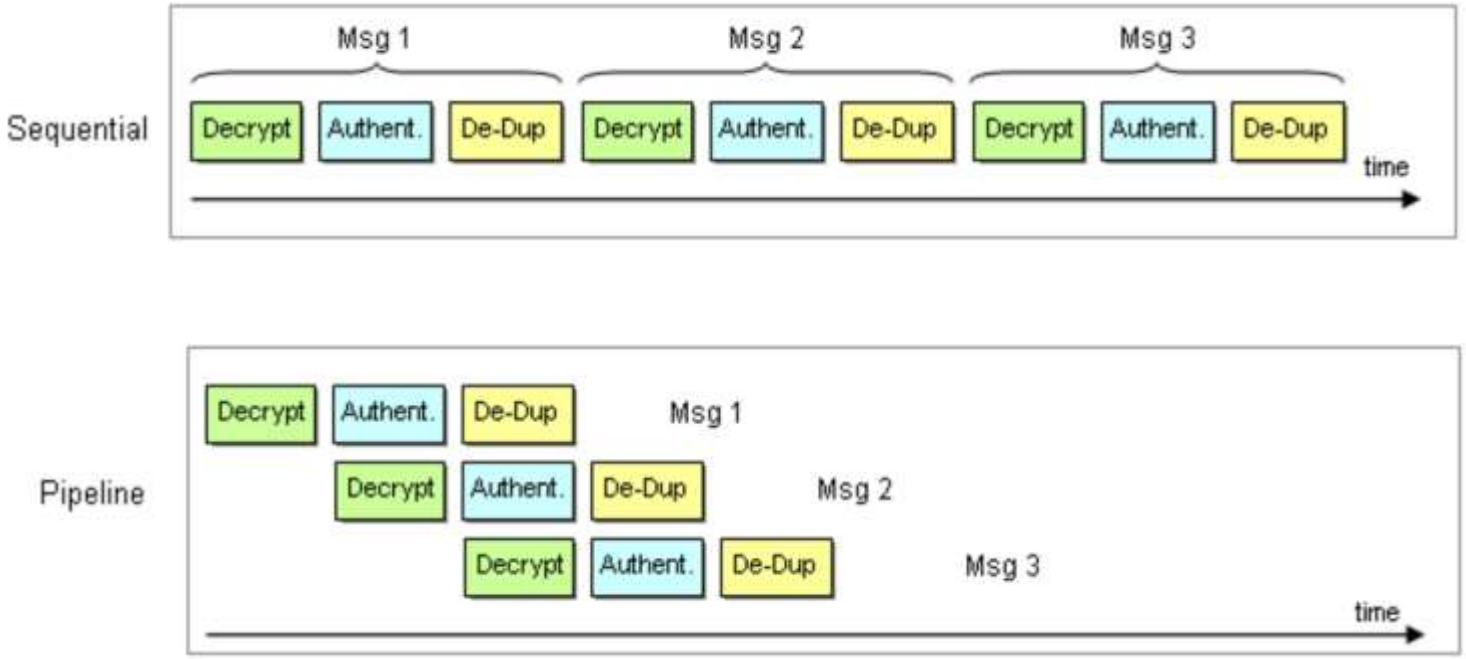
Motivation example #1



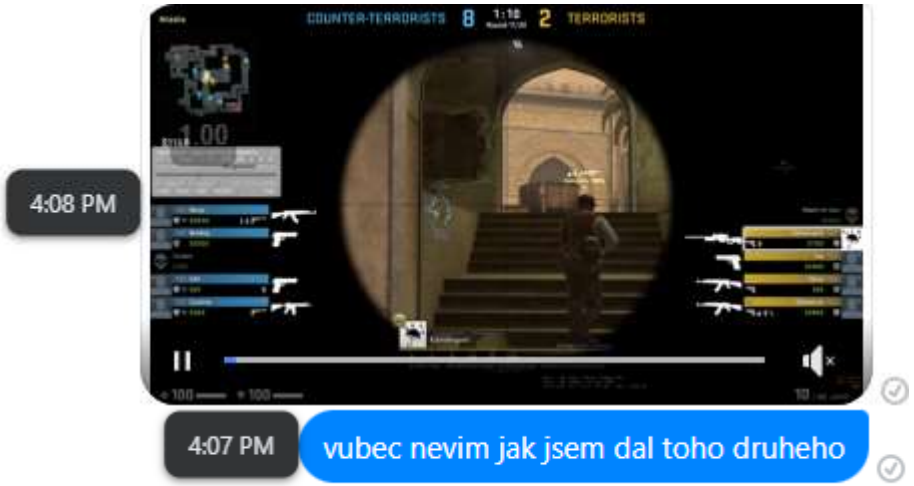
Motivational example #1

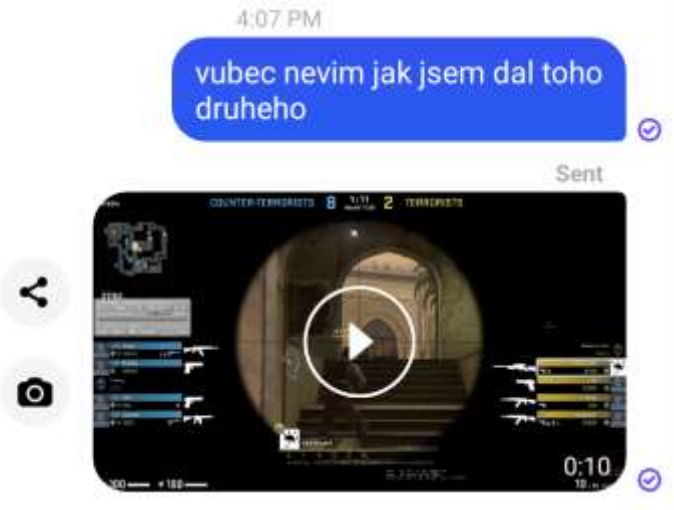


Motivational example #2



Motivational example #2





Simple code example – array

```python
class Filter:
    def __init__(self, method):
        self.method = method
        self.next = None
```

```python
def to_lower(text):
    return text.lower()


def remove_overhead(text):
    return text[3:]
```

```python
class Pipeline:
    def __init__(self):
        self.method_array = []

    def add(self, next_filter):
        self.method_array.append(next_filter)

    def call(self, args):
        tmp = args
        for current_filter in self.method_array:
            tmp = current_filter.method(tmp)
        return tmp
```

```python
if __name__ == '__main__':
    input_text = 'CCCVelIcE PODstATny teXT'

    pipeline = Pipeline()
    pipeline.add(Filter(to_lower))
    pipeline.add(Filter(remove_overhead))

    output_text = pipeline.call(input_text)
    print(output_text)
```

```
velice podstatny text

Process finished with exit code 0
```

Even simpler code example

```python
def to_lower(text):
    return text.lower()


def remove_overhead(text):
    return text[3:]
```

```python
if __name__ == '__main__':
    input_text = 'CCCVelIcE PODstATny teXT'
    output_text = remove_overhead(to_lower(input_text))
    print(output_text)
```

```
velice podstatny text

Process finished with exit code 0
```

Even even simpler code example

```python
if __name__ == '__main__':
    input_text = 'CCCVelIcE PODstATny teXT'
    output_text = input_text.lower()[3:]
    print(output_text)
```

```
velice podstatny text

Process finished with exit code 0
```

Pros and cons

Pros

- Flexibility
- Parallelism

Cons

- Complex pipeline + overhead
- Less efficient if state is needed
- Needs proper error handling
- Possible different filter data types

Simple code example – array

```python
class Filter:
    def __init__(self, method):
        self.method = method
        self.next = None
```

```python
def to_lower(text):
    return text.lower()


def remove_overhead(text):
    return text[3:]
```

```python
class Pipeline:
    def __init__(self):
        self.method_array = []

    def add(self, next_filter):
        self.method_array.append(next_filter)

    def call(self, args):
        tmp = args
        for current_filter in self.method_array:
            tmp = current_filter.method(tmp)
        return tmp
```

```python
if __name__ == '__main__':
    input_text = 'CCCVelIcE PODstATny teXT'

    pipeline = Pipeline()
    pipeline.add(Filter(to_lower))
    pipeline.add(Filter(remove_overhead))

    output_text = pipeline.call(input_text)
    print(output_text)
```

```
velice podstatny text

Process finished with exit code 0
```

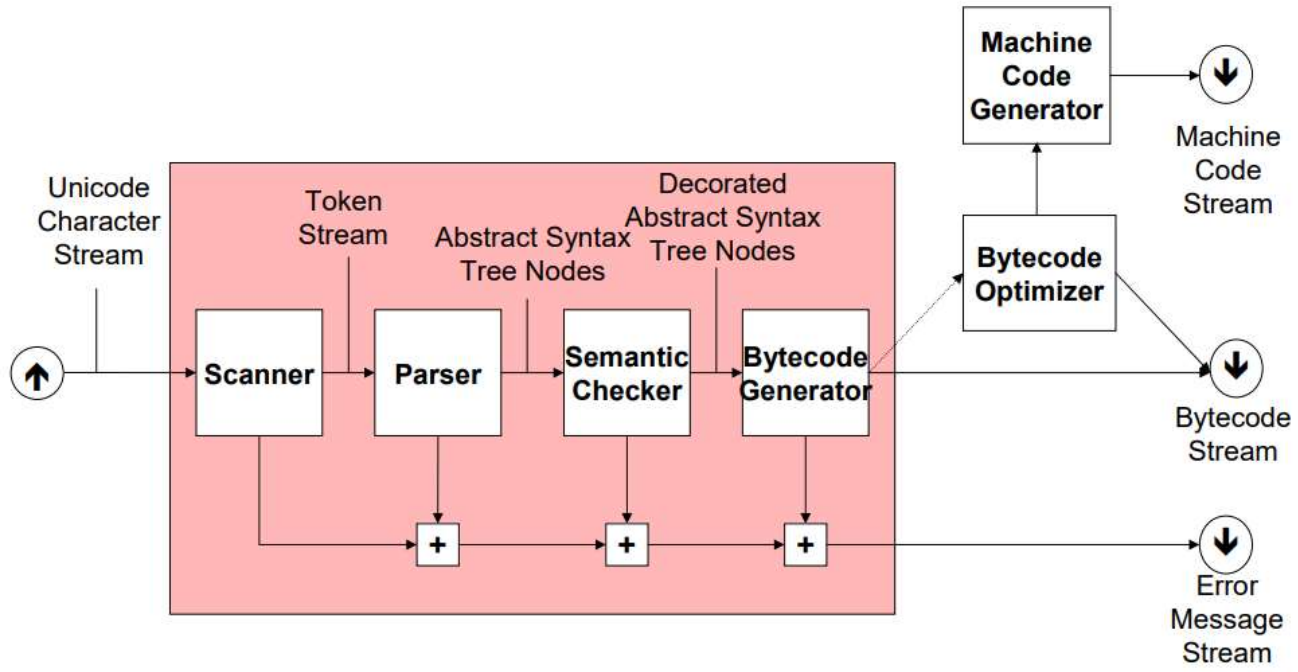
When to use this pattern?

- Problem can be divided into smaller tasks => Data processing
- Multiple problems can be solve with common filters
- Flexibility is required for reordering

Real word usage - unix

```
$ls -l | grep "Aug" | sort +4n
-rw-rw-r--  1 carol doc      1605 Aug 23 07:35 macros
-rw-rw-r--  1 john  doc      2488 Aug 15 10:51 intro
-rw-rw-rw-  1 john  doc      8515 Aug  6 15:30 ch07
-rw-rw-rw-  1 john  doc     11008 Aug  6 14:10 ch02
```

Real word usage - compilers



Pipes and filters vs. Layers

- Both decompose a problems into smaller tasks
- Both parts can be easily reedited or replaced

BUT !

- Pipes and filters to tackle multiple problems thanks to flexibility
- Layers tackle one specific problem
- Sometimes hard to differentiate between layers