

# Unleashing the Power of Pipelines

---



**Mark Heath**

Software Architect

@mark\_heath    [www.markheath.net](http://www.markheath.net)



```
"10,5,0,8,10,1,4,0,10,1"  
    .Split(',')  
    .Select(int.Parse)  
    .OrderBy(n => n)  
    .Skip(3)  
    .Sum()
```

## Chaining LINQ Extension Methods

**Create a “pipeline”**



# Overview



**What are pipelines?**

**Solve some LINQ challenges**

**Real-world use of pipelines**

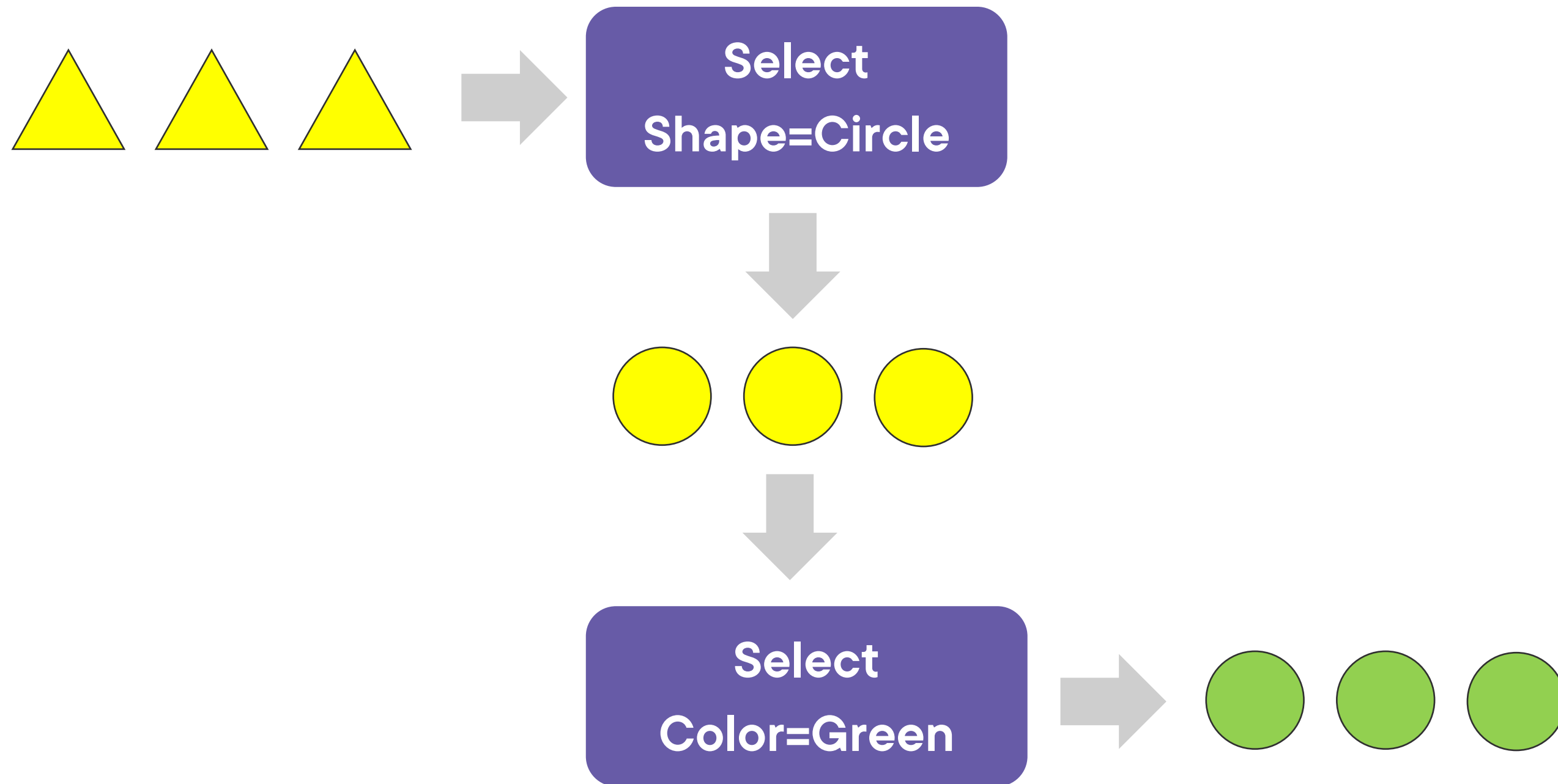
**Pipeline building blocks**



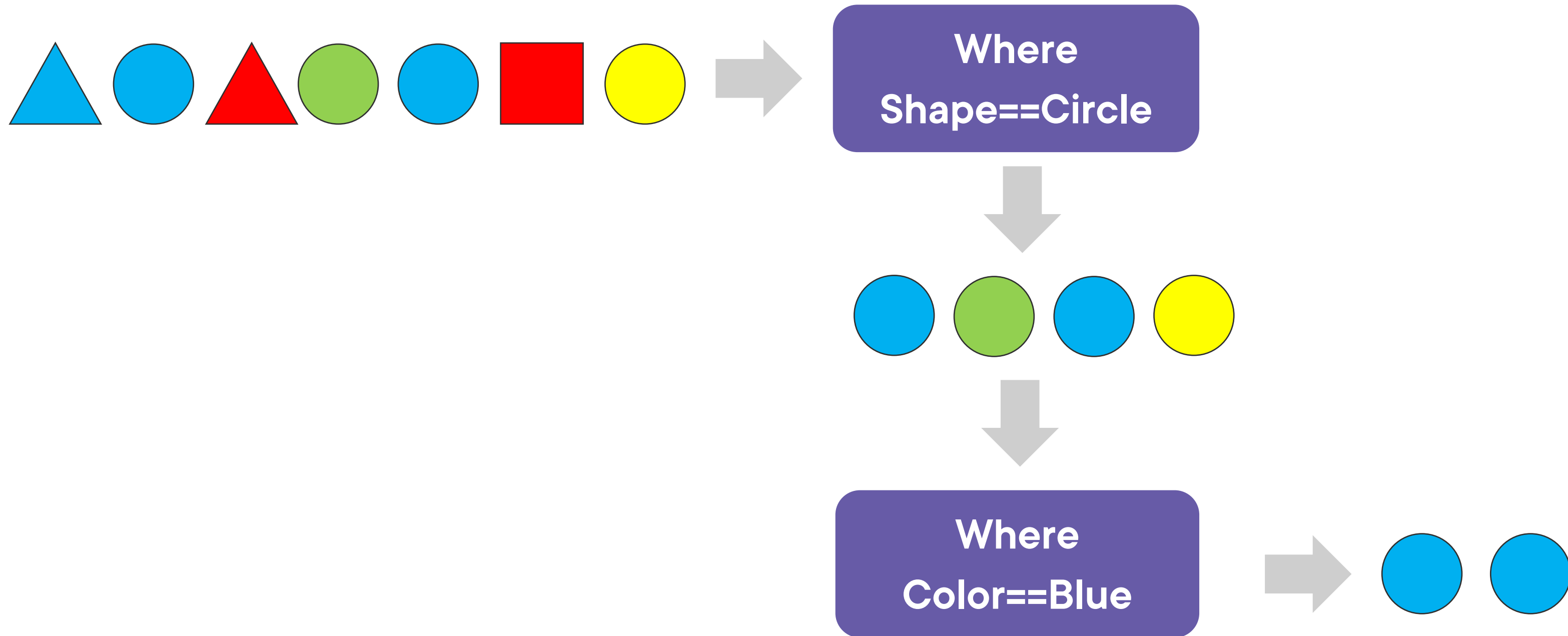
# Pipelines



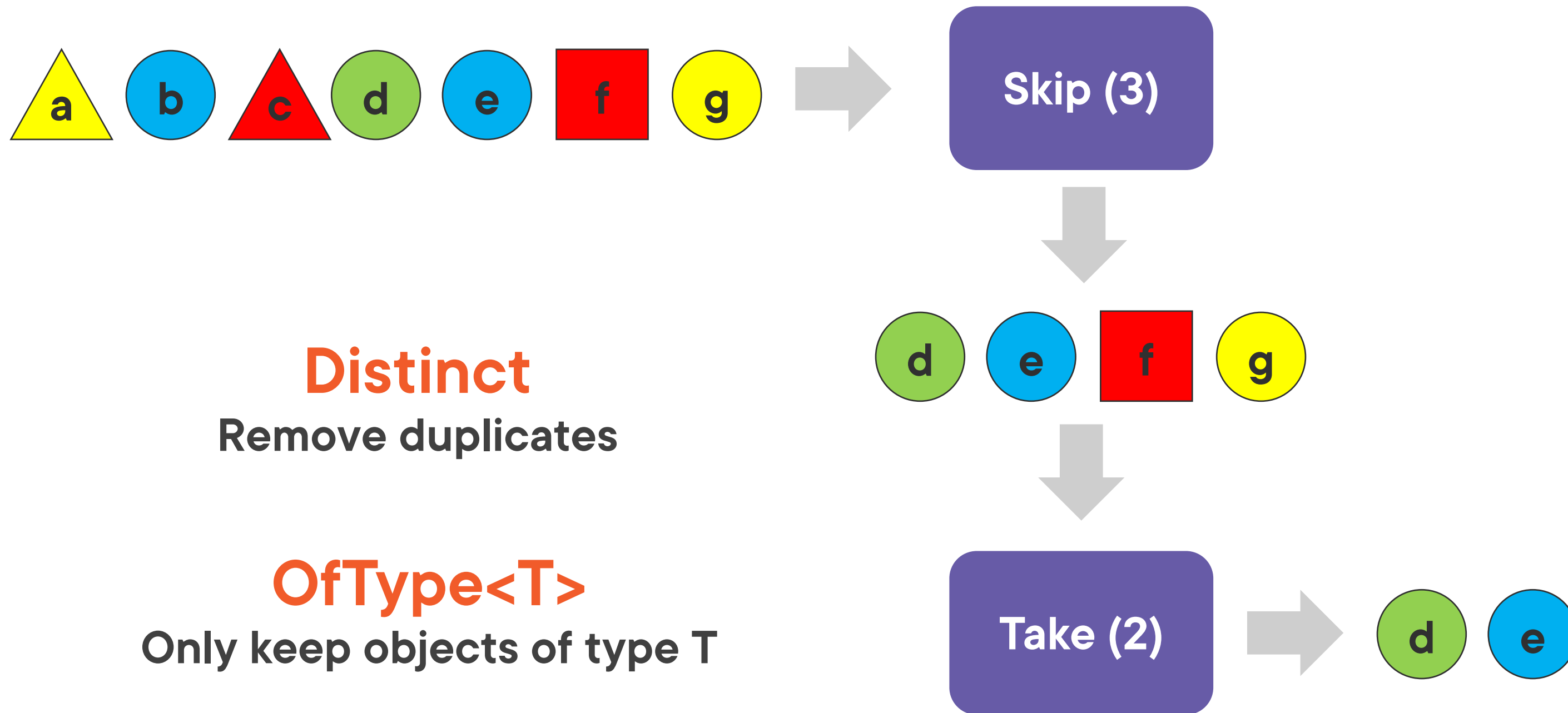
# Transforming Elements



# Filtering Elements



# Filtering Elements

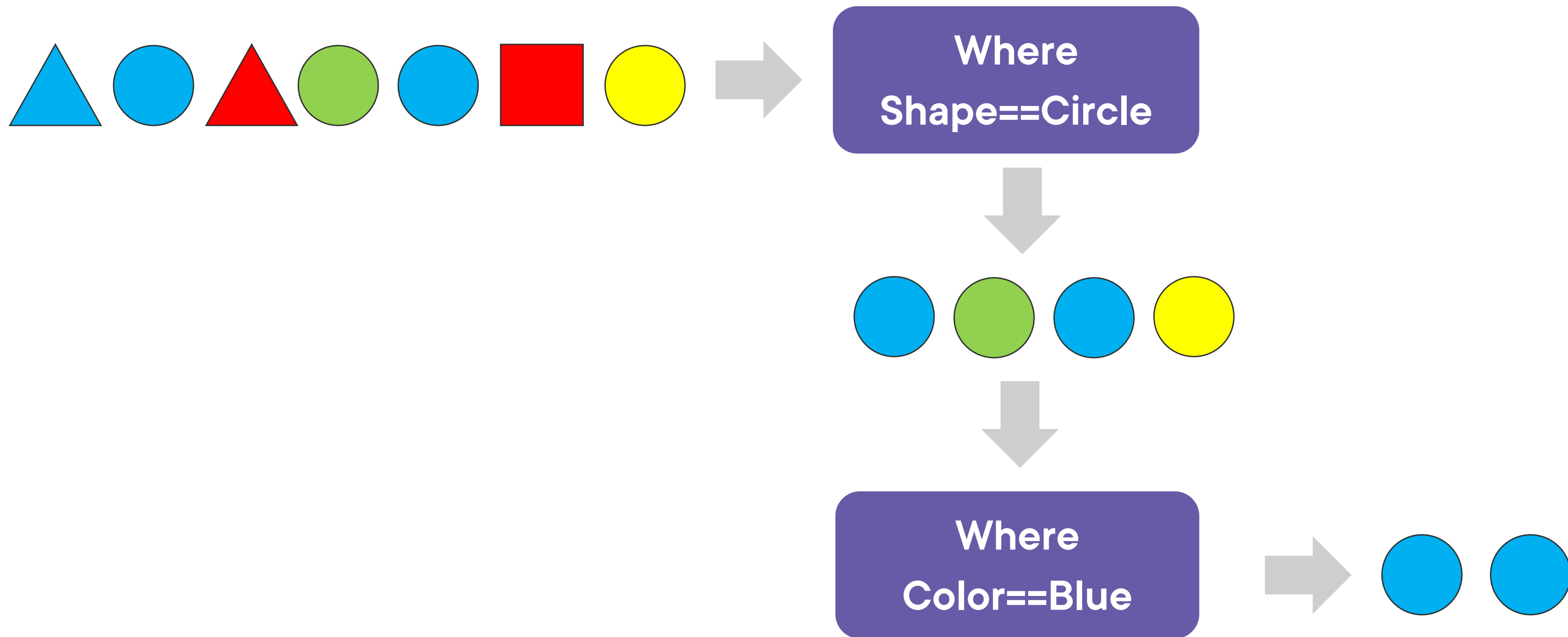


Order is important!

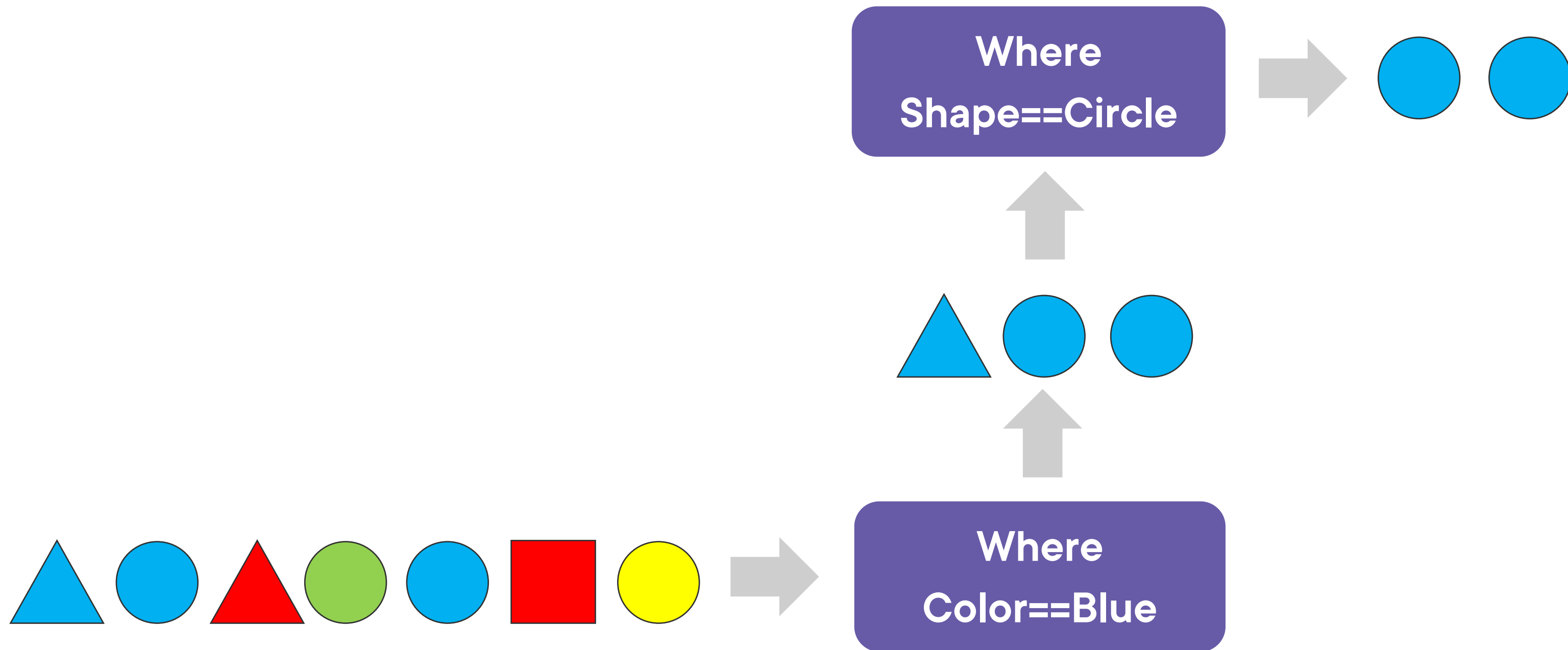




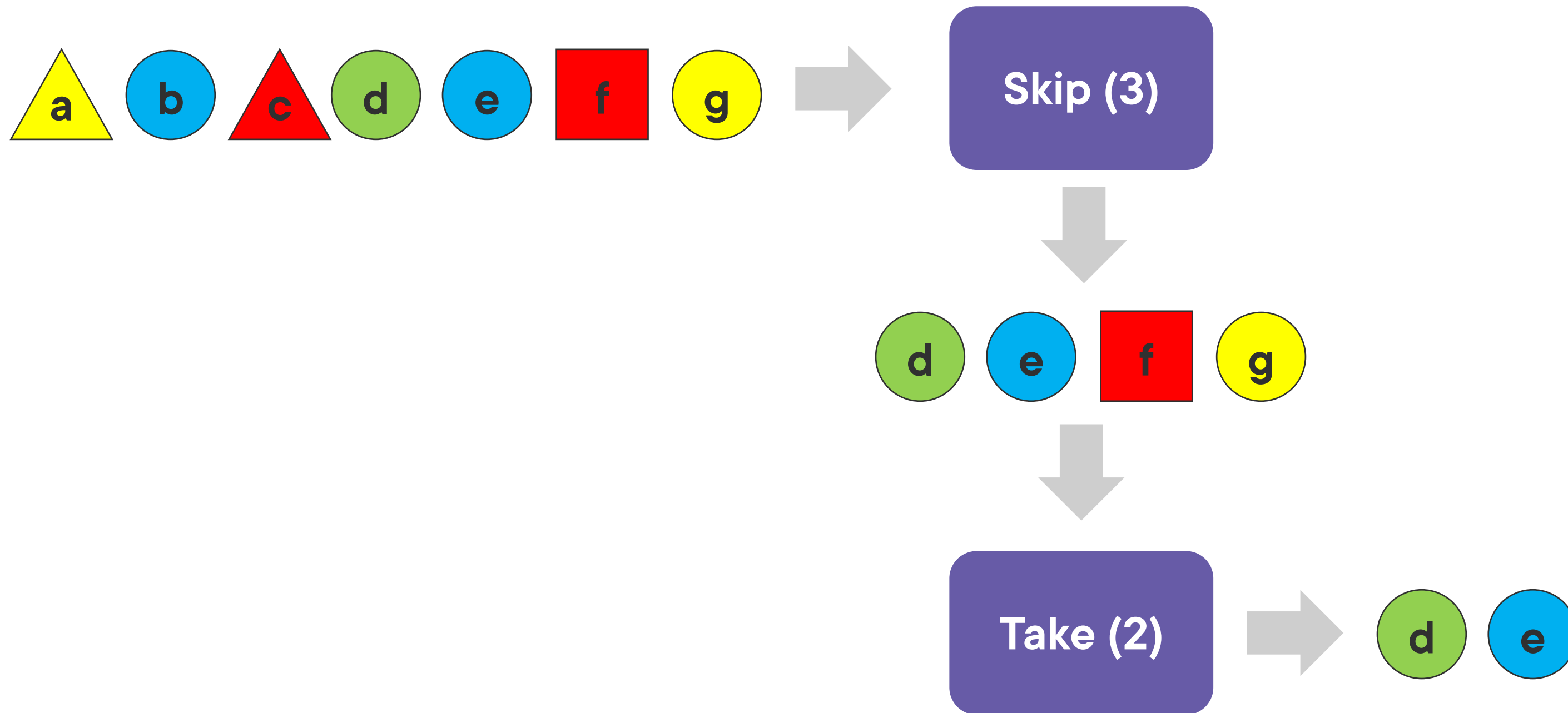
# Reordering Where Nodes



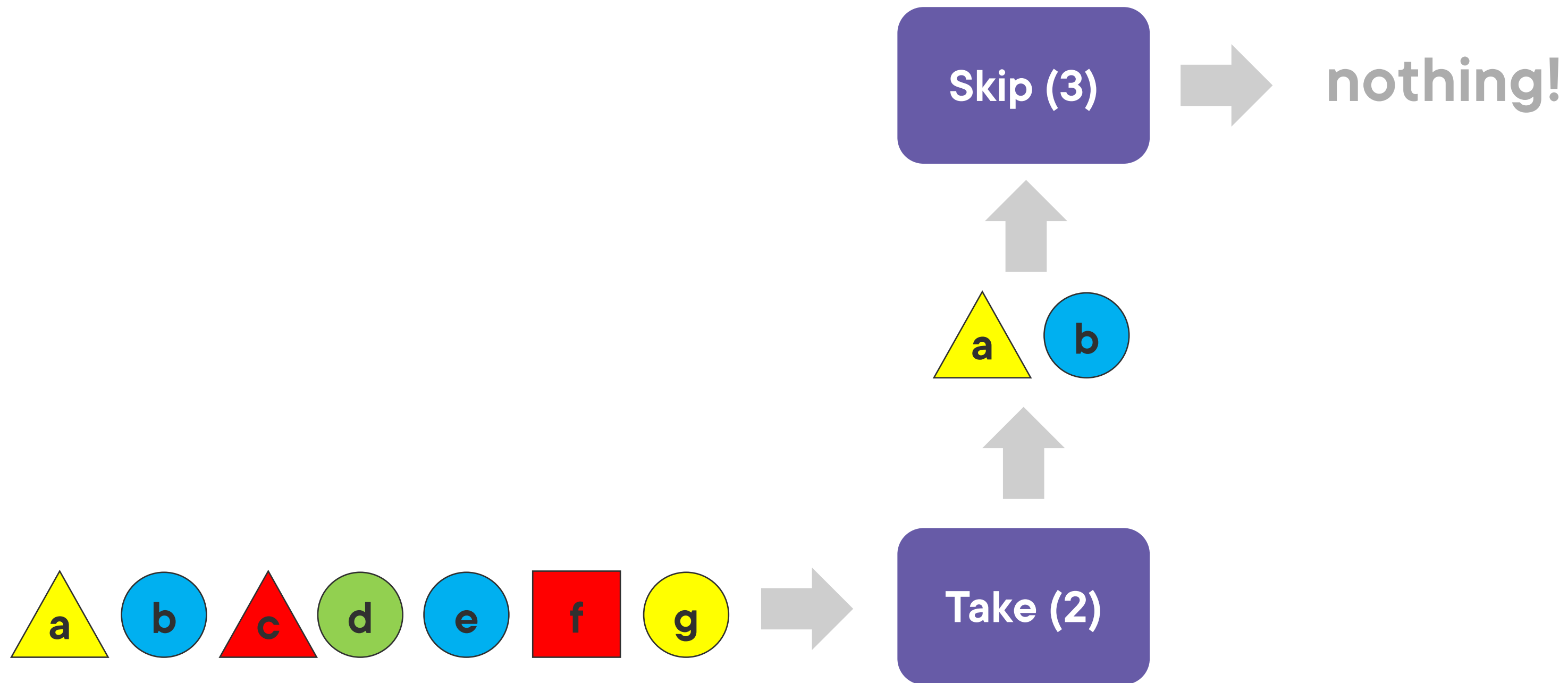
# Reordering Where Nodes



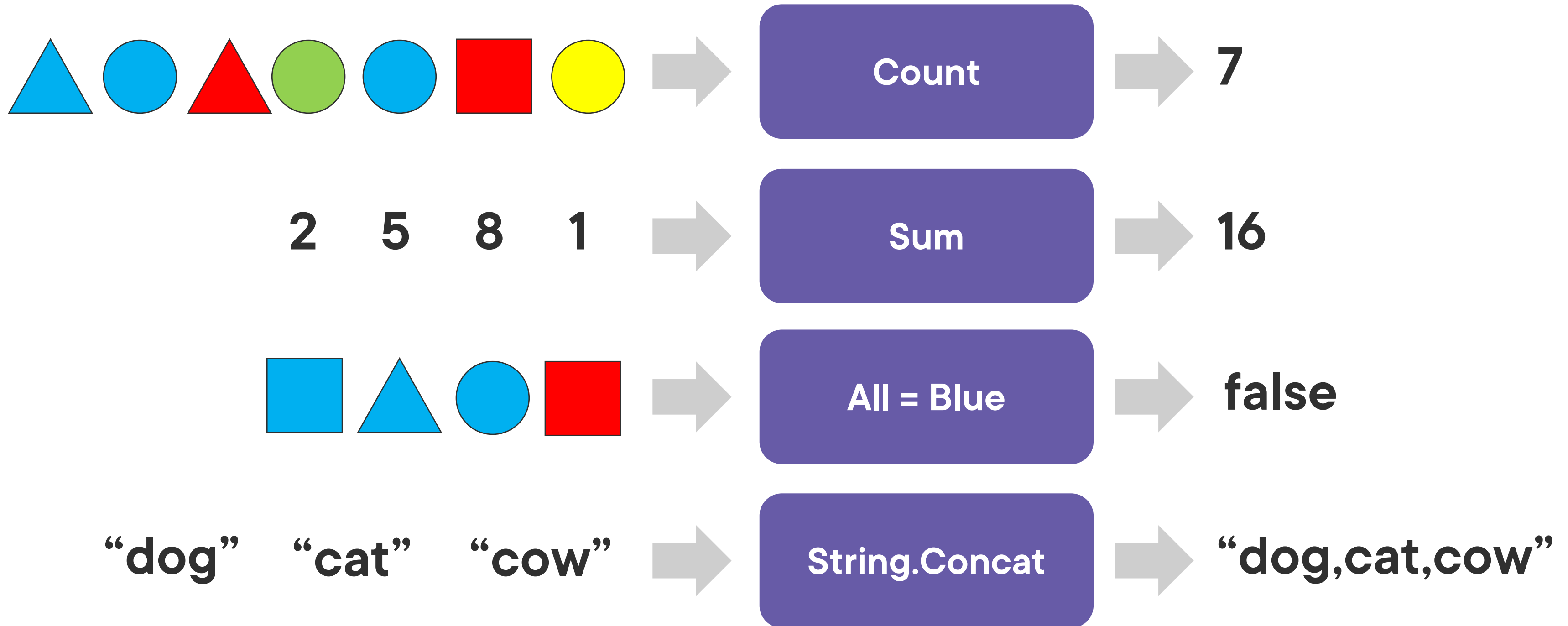
# Reordering Skip and Take



# Reordering Skip and Take



# Reducing Sequences



# Many-to-One Methods

**First**

**Count**

**Min**

**FirstOrDefault**

**Sum**

**Max**

**Last**

**All**

**Any**

**LastOrDefault**

**String.Concat**

**Aggregate**

`IEnumerable<T>` in, single value out



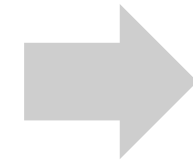
# Aggregate

General purpose LINQ extension method to apply a function to each element in an `IEnumerable<T>` to calculate a single value



# Generating Sequences

**Start = 2**  
**Length = 4**

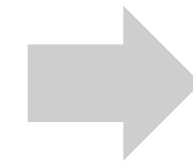


**Enumerable.Range**

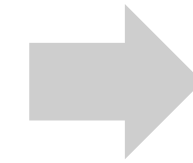


**2 3 4 5**

**“dog,cat,cow”**



**String.Split**



**“dog” “cat” “cow”**

**Enumerable.Repeat**

Repeats an item a specified  
number of times

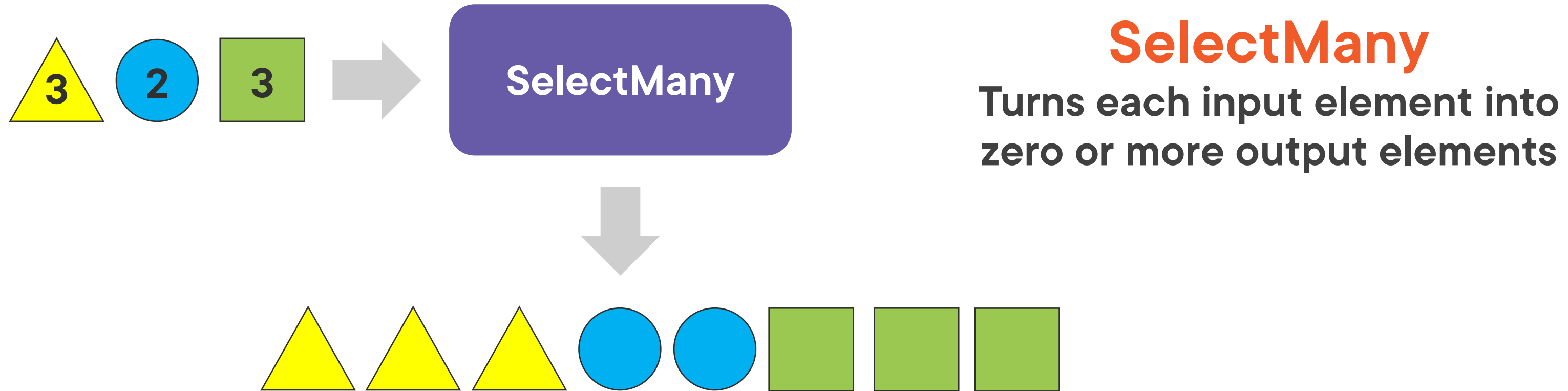
**Regex.Matches**

Returns all the matches on a  
Regular Expression

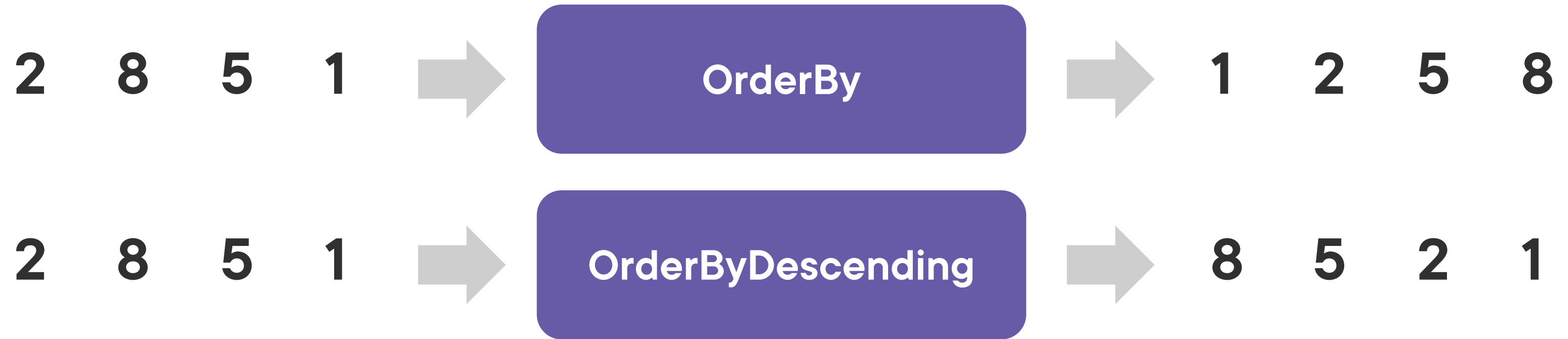




# Expanding Sequences



# Reordering Sequences



## ThenBy

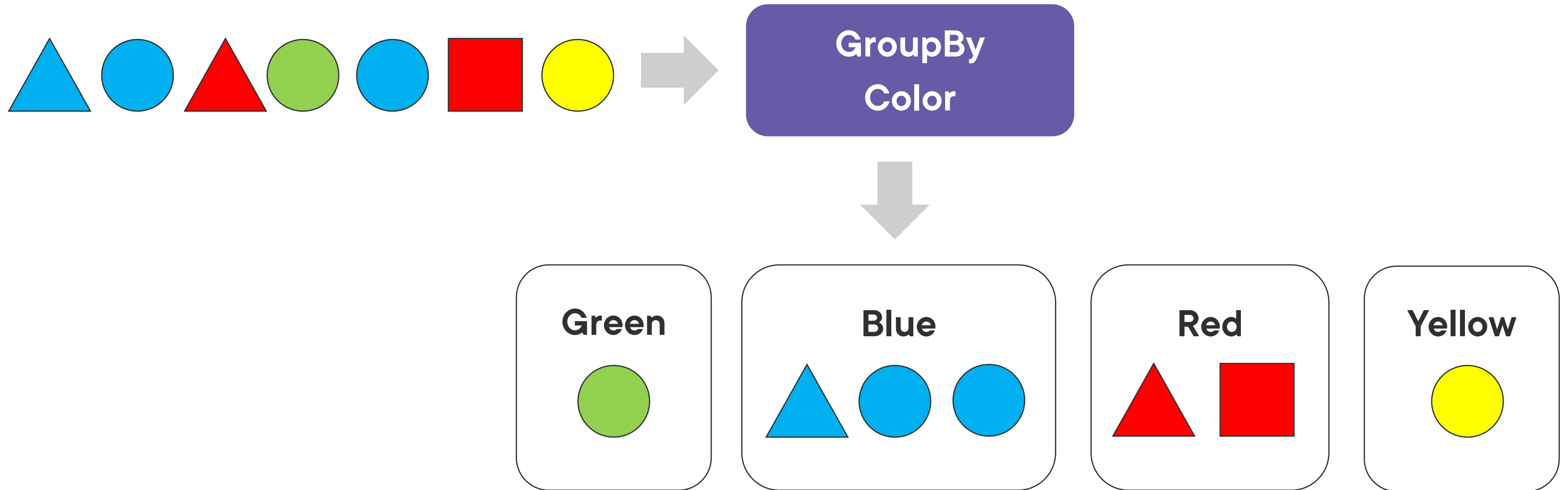
Follow OrderBy with a secondary sort order

## Reverse

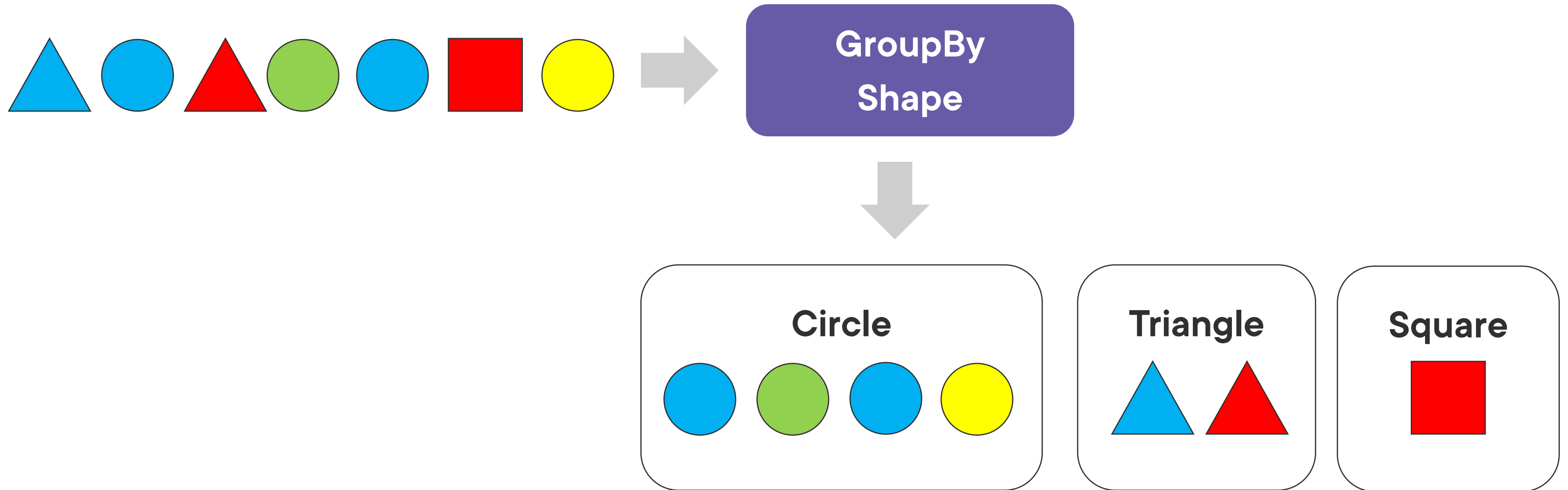
Reverses the order of elements



# Grouping Sequences



# Grouping Sequences



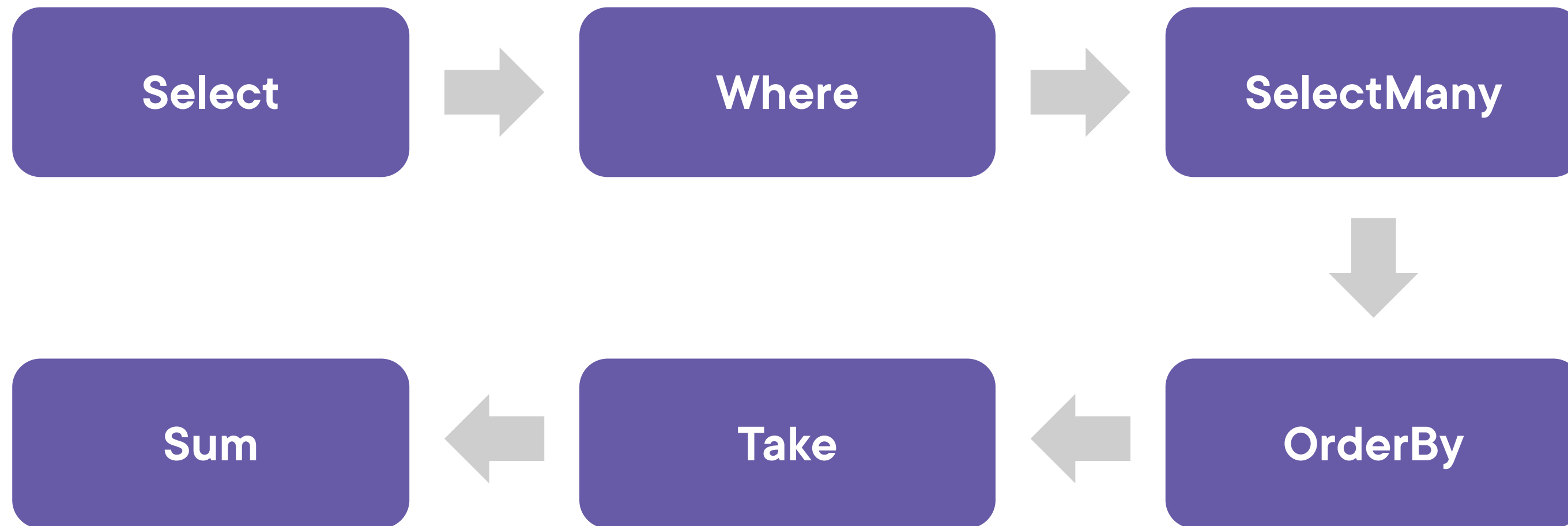
Reordering methods may  
require caching the whole  
sequence in memory



With LINQ to Entities,  
reordering and grouping is  
performed by the database



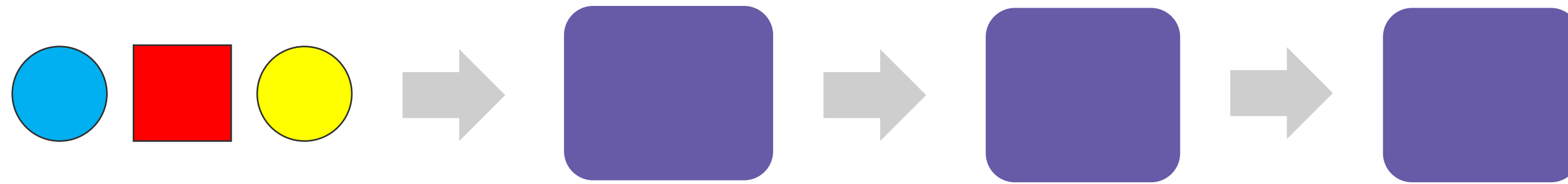
# The Power of Pipelines



Solve **complex** problems with many  
**simple** steps



# Pulling Data Through the Pipeline



Nothing comes through the pipeline until we **ask** for it

**foreach**

**Max**

“deferred execution”





Challenge



LINQ Challenge



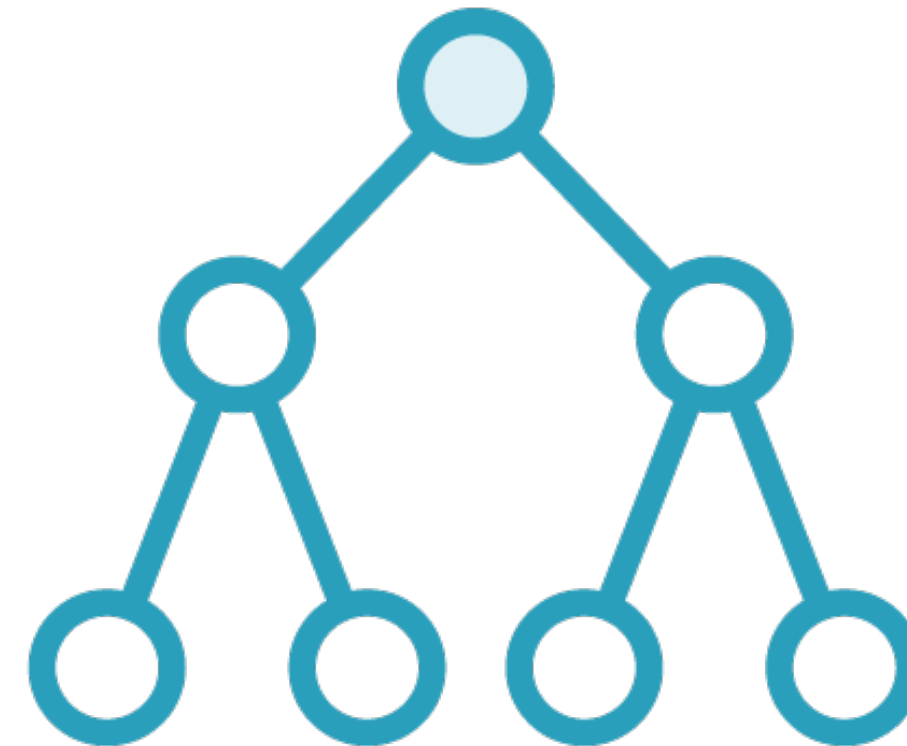
Album Duration



# Challenge



# LINQ Challenge



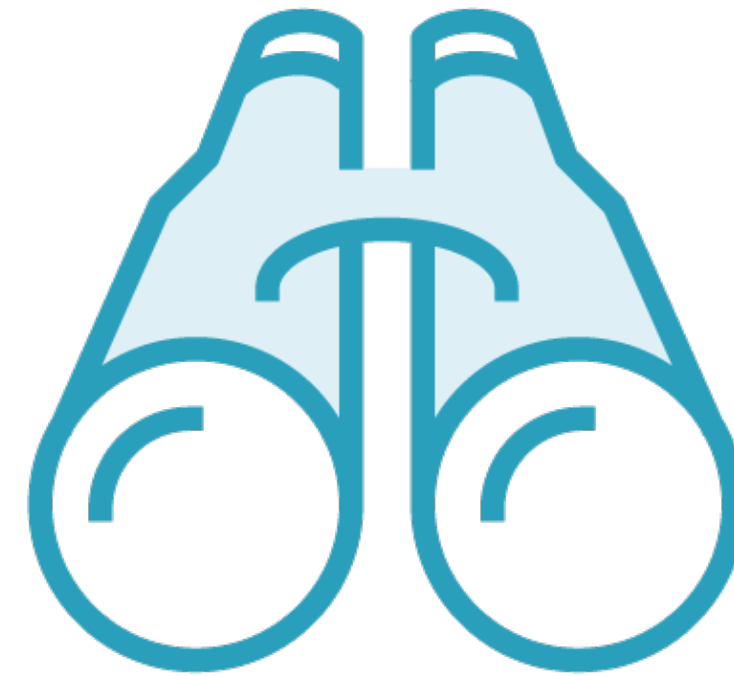
# Range Expansion



Demo



Real World LINQ



Find in Files



Demo



Real World LINQ



Parsing Log Files



Demo



Real World LINQ



Validating Enums



# Summary



## Pipeline building blocks:

- From one to many
- Transform elements
- Filter elements
- From many to one
- Combine sequences

## LINQ challenges

## Real world LINQ



# Up Next: Clean and Readable Code

---

