Asynchronously Reading Resources

Kevin Dockx
ARCHITECT

@KevinDockx https://www.kevindockx.com

Coming Up

Adding an Async Controller Action

Testing Async Code Improvements

Using an AsyncResultFilter

Demo

Getting Resources

Introducing WebSurge

Our goal is to test for scalability improvements

- Load testing
- Combined with thread pool throttling

WebSurge is a free tool specifically aimed at load testing

Demo

Using WebSurge to Test Async Code Improvements

The Outer Facing Model

Entity model

- Entity classes represent (partial) database rows as objects

Outer facing model

- DTO classes represent resources that are sent over the wire

The Outer Facing Model

Book entity	Book DTO
Guid Id	Guid Id
string Title	string Title
string Description	string Description
Guid AuthorId	
Author Author	

The Outer Facing Model

Book entity	Book DTO
`Guid Id`	`Guid Id`
`string Title`	`string Title`
`string Description`	`string Description`
`Guid AuthorId`	
`Author Author`	

The Outer Facing Model

Book entity	Book DTO
Guid Id	Guid Id
string Title	string Title
string Description	string Description
Guid AuthorId	
Author Author	

The Outer Facing Model

Book entity	Book DTO
`Guid Id`	`Guid Id`
`string Title`	`string Title`
`string Description`	`string Description`
`Guid AuthorId`	`string Author`
`Author Author`	

The Outer Facing Model

Book entity	Book DTO
Guid Id	Guid Id
string Title	string Title
string Description	string Description
Guid AuthorId	string Author
Author Author	IEnumerable<BookCover>

Mixing models & responsibilities between layers leads to evolvability issues

The Outer Facing Model

How do we represent the resource data type?

Model classes (DTOs)

Statically typed approach

Dynamics, anonymous objects, ExpandoObject

Dynamically typed approach

Mapping code in controller actions

The Outer Facing Model

How do we represent the resource data type?

Model classes (DTOs)

Statically typed approach

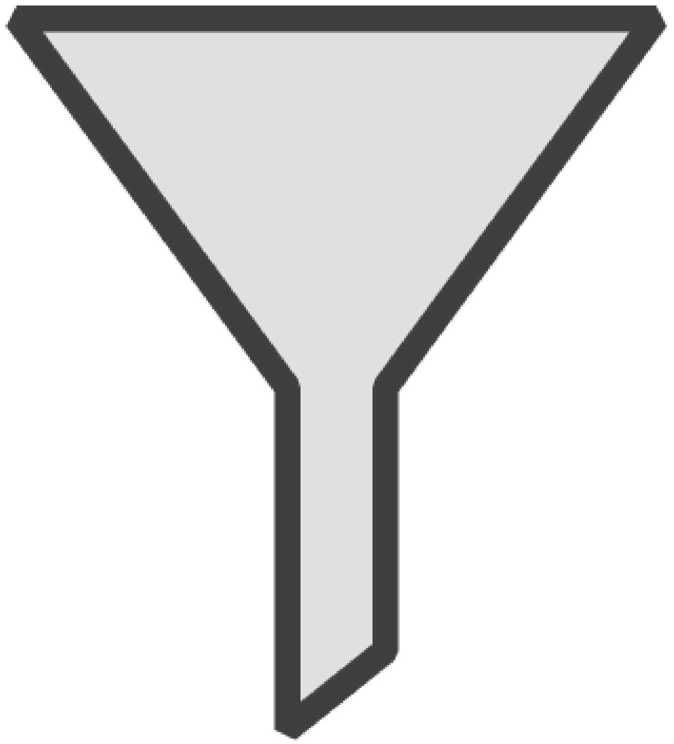
Dynamics, anonymous objects, ExpandoObject

Dynamically typed approach

~~Mapping code in controller actions~~

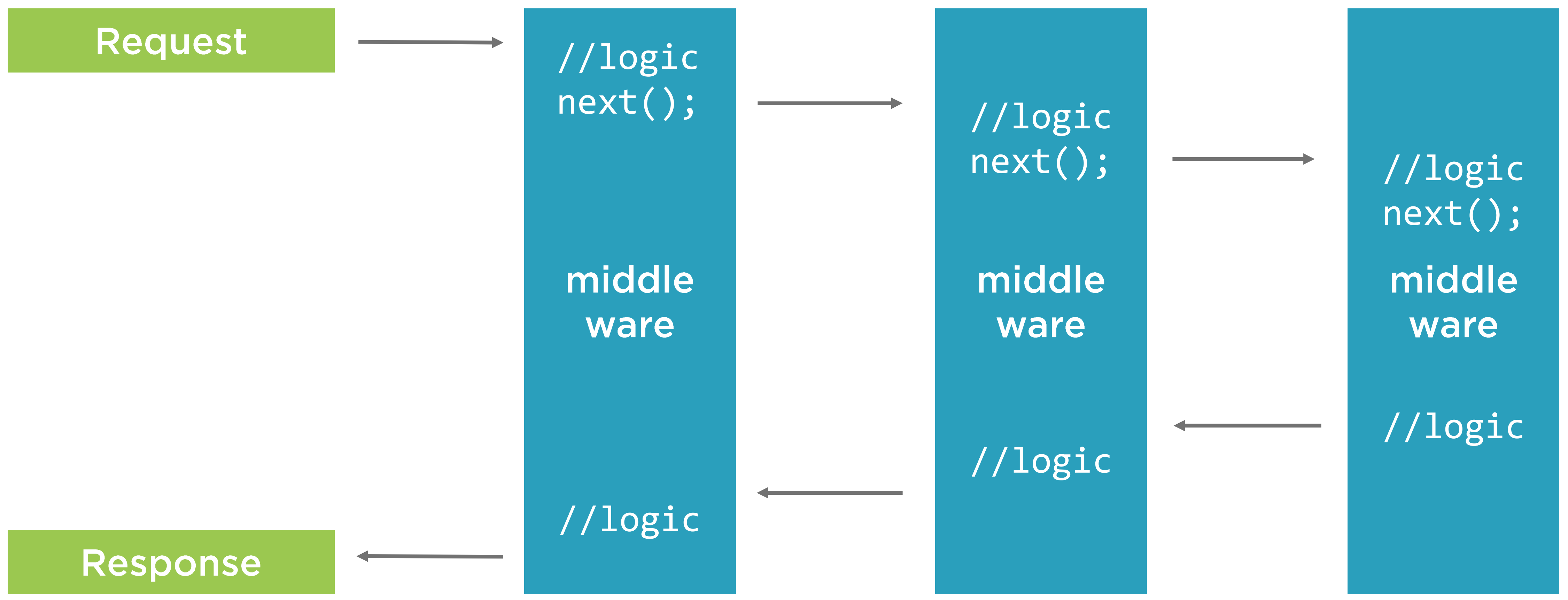
Reusable IAsyncResultFilter

Manipulating Output with an AsyncResultFilter

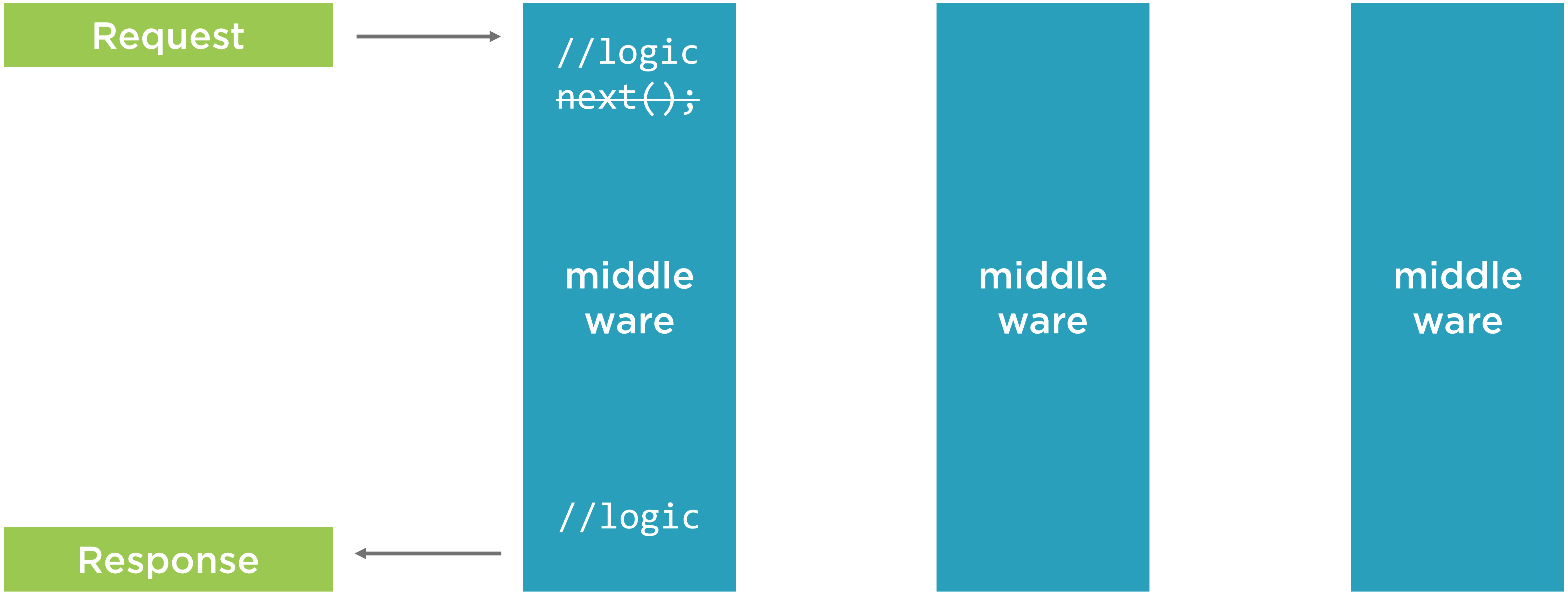


Filters in ASP.NET Core MVC allow us to run code before or after specific stages in the request processing pipeline

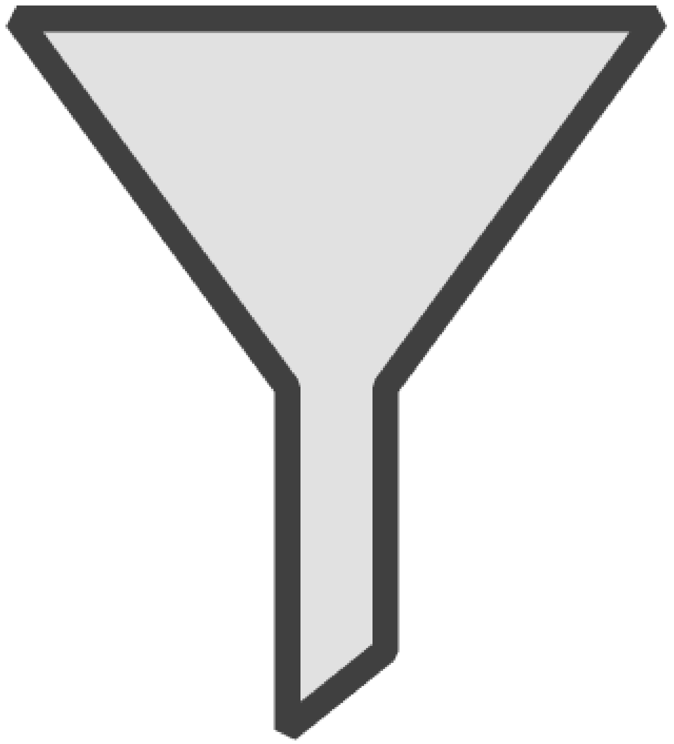
The ASP.NET Core Request Pipeline



The ASP.NET Core Request Pipeline



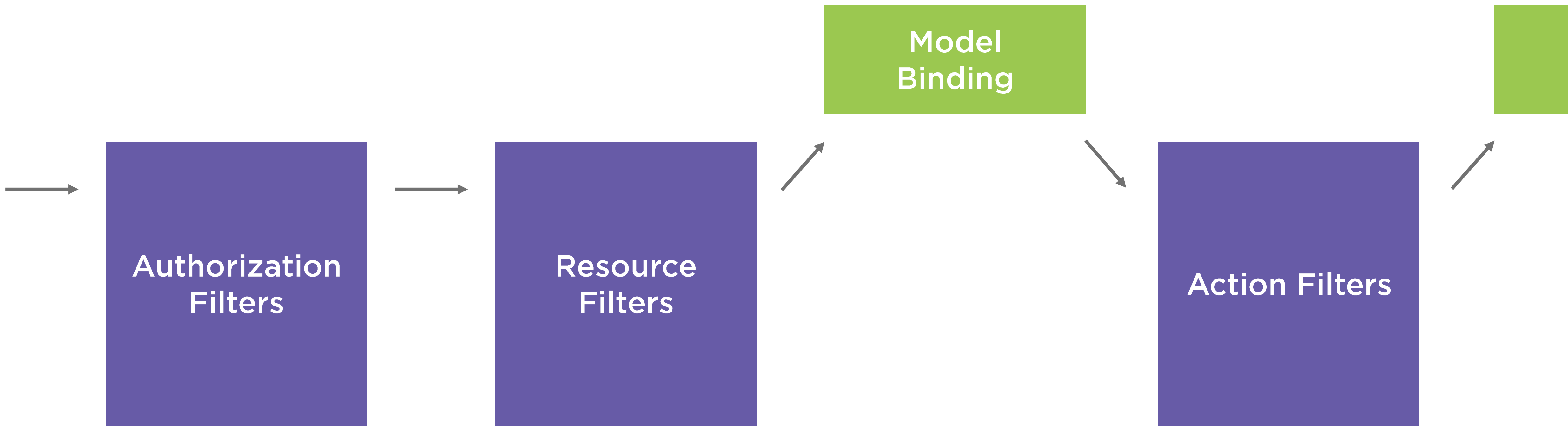
Manipulating Output with an AsyncResultFilter



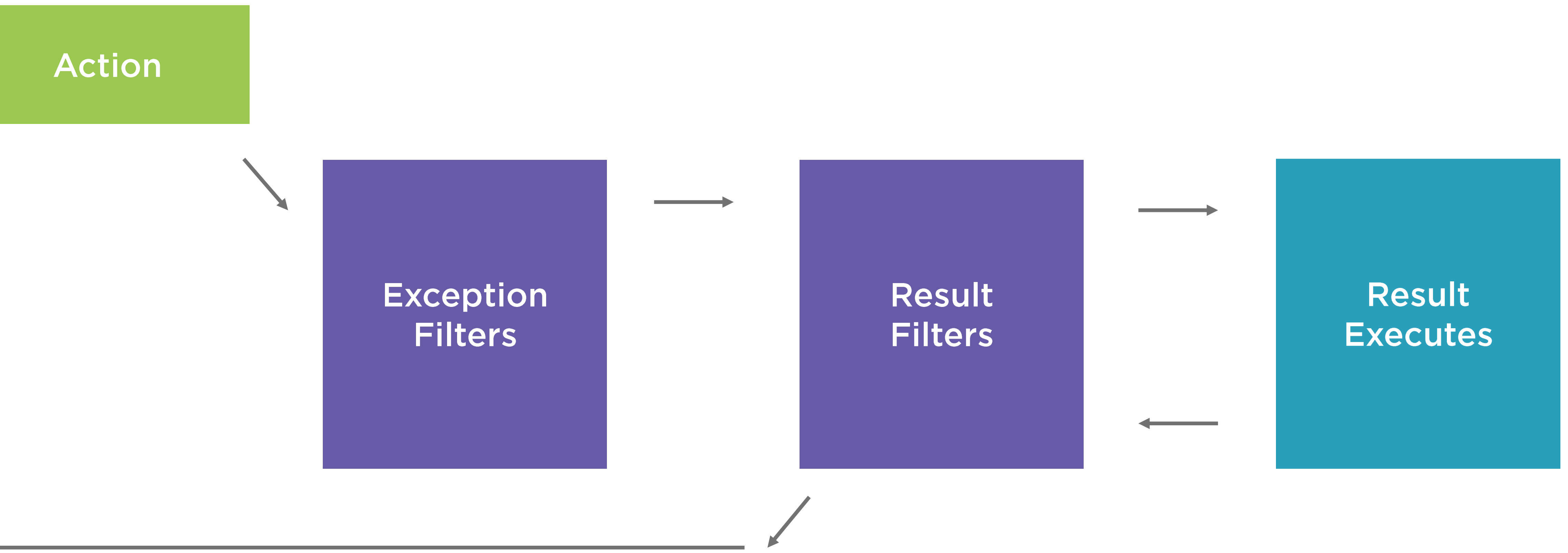
ASP.NET Core MVC has its own pipeline it sends requests through

Filters run within the MVC action invocation pipeline (aka filter pipeline)

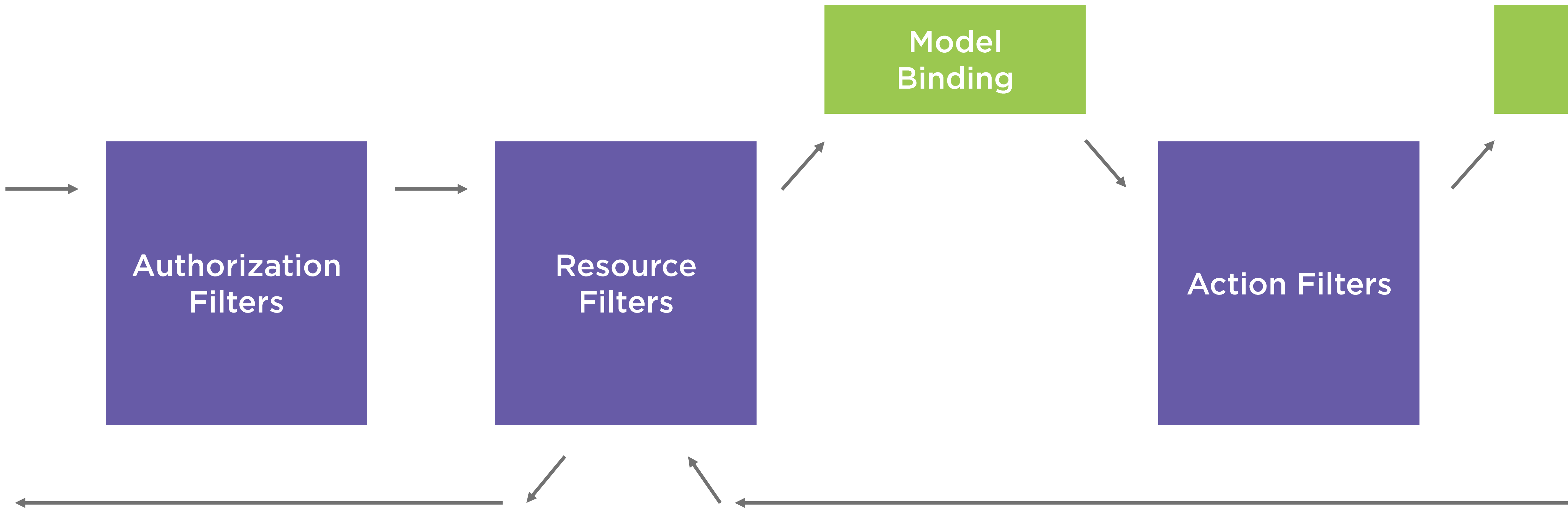
The ASP.NET Core MVC Filter Pipeline



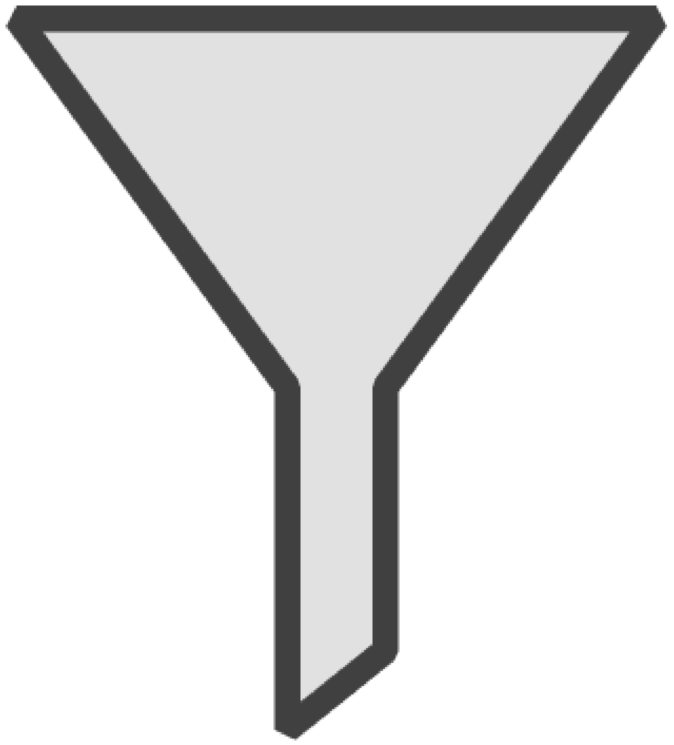
The ASP.NET Core MVC Filter Pipeline



The ASP.NET Core MVC Filter Pipeline



Manipulating Output with an AsyncResultFilter



By using result filters, we ...

- ... can keep our actions cleaner
- ... promote reuse

IResultFilter, IAsyncResultFilter interfaces

- ResultFilterAttribute (abstract)

Demo

Creating a Custom AsyncResultFilter (Part 1)

Demo

Adding and Configuring AutoMapper

Demo

Creating a Custom AsyncResultFilter (Part 2)

Summary

Async code has a tendency to bubble up application layers due to compiler errors and warnings

- Can’t await if the calling method isn’t marked with the async modifier

Summary

Keep models separate

- Mixing models & responsibilities between layers leads to evolvability issues

Summary

Result filters run right before and after the result is executed

- Makes them a good location for mapping code
- Makes the mapping code reusable