API VERSIONING IS FOR EVERYONE!

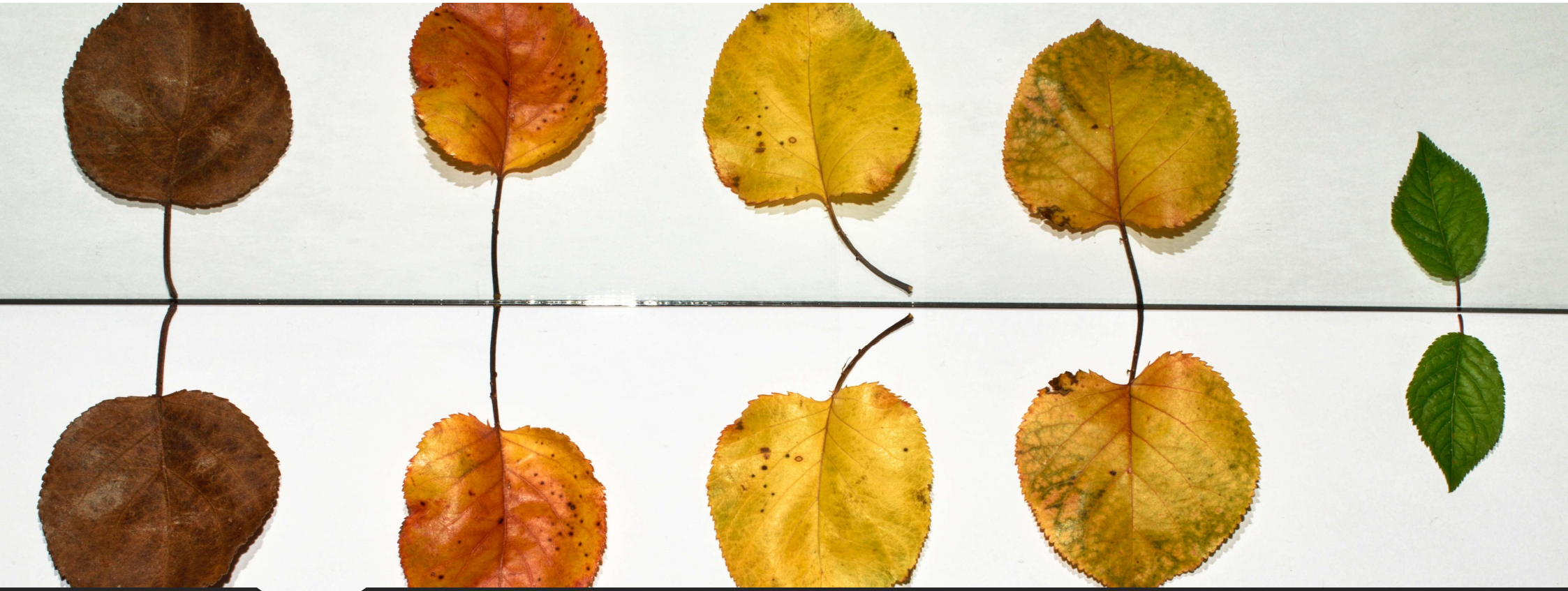
API Versioning

Versioning

- HTTP API
- gRPC Service
- GraphQL Server

The need for API versioning

- Maintain backwards compatibility
- Make changes to API without disrupting clients
- Allow API consumers to update at their own pace
- Phased introduction of breaking changes
- Reduce client-server coupling



When to version your API?

API versioning is not always the answer

Let clients embrace or ignore changes

BREAKING CHANGES

- Changes in contract - Request, Response, Endpoints names
- Changes in application logic*
- Changes in error code or fault contracts
- Removing part of API
- Anything that violates the Principle of Least Astonishment

Version Number Strategy

- SemVer – <*Major.Minor.Patch*>
- Major Versions only – <*Major*>
- Major.Minor - <*Major.Minor*>
- Date of release as the version – <*yyyy-mm-dd*>
- GroupVersion, internally interpreted as a version - <*yyyy-mm-dd*>

Keep it consistent!

HTTP API

Resources

Reuse HTTP

HTTP Verb	Url	Request Body	Response Body	Status Code
GET	/weatherforecast /weatherforecast/{id}	✗	✓	200OK/404 Not Found
POST	/weatherforecast	✓	✓	201 Created
PUT	/weatherforecast/{id}	✓	✗	200OK/204 No Content/ 202 Accepted
DELETE	/weatherforecast/{id}	✗	✗	204 No Content/404 Not Found

Reuse HTTP for Versioning

- Version by query string
- Version by header
- Version by media type
- Version by url

```
dotnet add package Asp.Versioning.Mvc --version 7.0.0
```

```
namespace ApiVersioningDemo.V1;

[ApiVersion( 1.0 )]
[ApiController]
[Route( "[controller]" )]
public class WeatherForecastController : ControllerBase
{
    [HttpGet]
    public string Get() {
        //code implementation
    }
}

```

- Controllers of same name with different [ApiVersion(v)] attributes
- Different namespaces

```
namespace ApiVersioningDemo.V2;

[ApiVersion( 2.0 )]
[ApiController]
[Route( "[controller]" )]
public class WeatherForecastController : ControllerBase
{
    [HttpGet]
    public string Get() {
        //code implementation
    }
}

```

- Controllers of different names with different [ApiVersion(v)] attributes - nameController , name#Controller
- Same namespace

```
namespace ApiVersioningDemo.V1;

[ApiVersion( 2.0 )]
[ApiController]
[Route( "weatherforecast" )]
public class WeatherForecast2Controller : ControllerBase
{
    [HttpGet]
    public string Get() {
        //code implementation
    }
}

```

```
namespace ApiVersioningDemo.V1;

[ApiVersion( 1.0 )]
[ApiController]
[Route( "[controller]" )]
public class WeatherForecastController : ControllerBase
{
    [HttpGet]
    public string Get(){
        //code implementation
    }
}
```

```
[ApiVersion( 1.0 )]
[ApiVersion( 2.0 )]
[ApiController]
[Route( "[controller]" )]
public class WeatherForecastController : ControllerBase
{
    [HttpGet]
    public string Get() {
        //code implementation for v1
    }

    [HttpGet, MapToApiVersion(2.0)]
    public string GetV2(){
        //code implementation for v2
    }
}
```

Version Interleaving

```
.AddApiVersioning(options =>
    {
        options.ApiVersionReader = new MediaTypeApiVersionReader("v");
        options.AssumeDefaultVersionWhenUnspecified = true;
        options.ApiVersionSelector = new CurrentImplementationApiVersionSelector( options );
    } );
```

```
GET weatherforecast HTTP/2
Host: localhost
Accept: application/json;v=1.0

```

```
GET weatherforecast HTTP/2
Host: localhost
Accept: application/json;v=2.0
```

```
1
2  .AddApiVersioning(options =>
3      {
4          options.ApiVersionReader = new UrlSegmentApiVersionReader();
5      } );
```

Not possible to have a default version for your API via url segment

```
1   namespace ApiVersioningDemo.V1;
2
3   [ApiVersion( 1.0 )]
4   [ApiVersion( 2.0 )]
5   [ApiController]
6   [Route( "v{version:apiVersion}/[controller]" )]
7   public class WeatherForecastController : ControllerBase
8   {
9       [HttpGet]
10      public string Get() {
11          //code implementation for v1
12      }
13
14      [HttpGet, MapToApiVersion(2.0)]
15      public string GetV2(){
16          //code implementation for v2
17      }
18  }
19
```

Deprecating Versions

API Versions can be marked deprecated

```
[ApiVersion(1.0, Deprecated = true)]
[ApiController]
[Route("[controller]")]

public class WeatherForecastController : ControllerBase
{
    //code
}
```

Define Sunset HTTP header in the response to indicate when the API will disappear

```
options =>
{
    options.ReportApiVersions = true;

    options.Policies.Sunset(1).Effective(2024, 3, 1)
    .Link("https://docs.api.com/policy.html?api-version=1.0")
    .Title("API Policy")
    .Type("text/html");
}
```

Remove the controller for the version

gRPC

Modern, high-performance, open-source Remote Procedure Call Framework

- Google’s implementation of RPC
- Officially supported from .NET Core 3.0
- Contract-based API development

```
syntax = "proto3";
```

```
//namespace for generated code
option csharp_namespace = "gRPC.Basic";
```

```
//namespace of the protocol buffer messages, helps prevent namespace clashes
package greet;
```

```
// The greeting service definition.
service Greeter {
  // rpc definitions
  rpc SayHello (HelloRequest) returns (HelloReply);
}
```

```
// The request message
message HelloRequest {
  string name = 1;
}
```

```
// The response message
message HelloReply {
  string message = 1;
}
```

```
// The greeting service definition.
service Greeter {
  // rpc definitions
  rpc SayHello (HelloRequest) returns (HelloReply);
}

// The request message
message HelloRequest {
  string name = 1;
}

// The response message
message HelloReply {
  string message = 1;
}
```

Greeter.GreeterBase – abstract class

SayHello() – virtual method

HelloRequest – C# class

HelloReply – C# class

```
// The greeting service definition.
service Greeter {
  // rpc definitions
  rpc SayHello (HelloRequest) returns (HelloReply);
}

// The request message
message HelloRequest {
  string name = 1;
}

// The response message
message HelloReply {
  string message = 1;
}
```

Greeter.GreeterClient

SayHello()

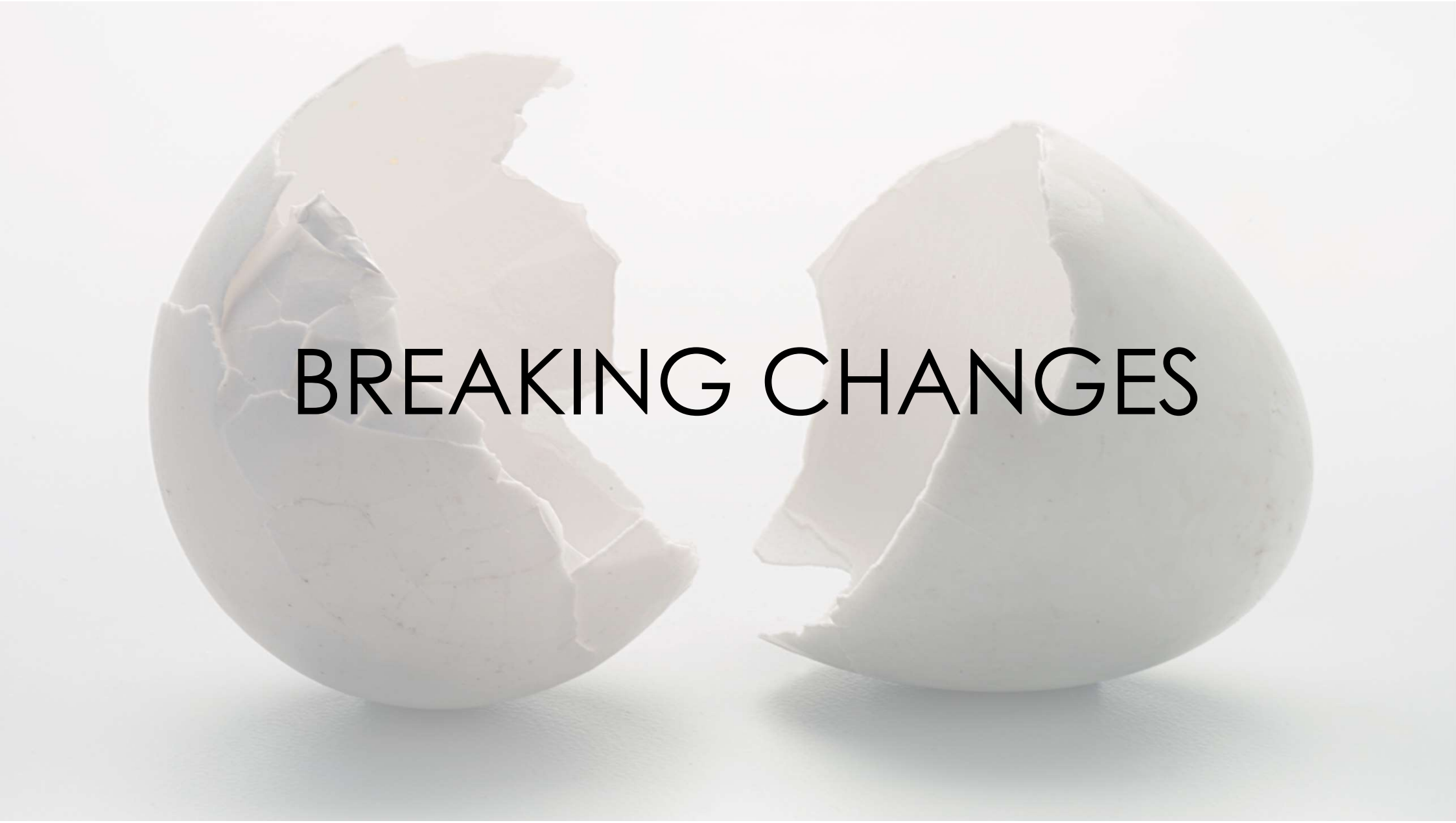
HelloRequest

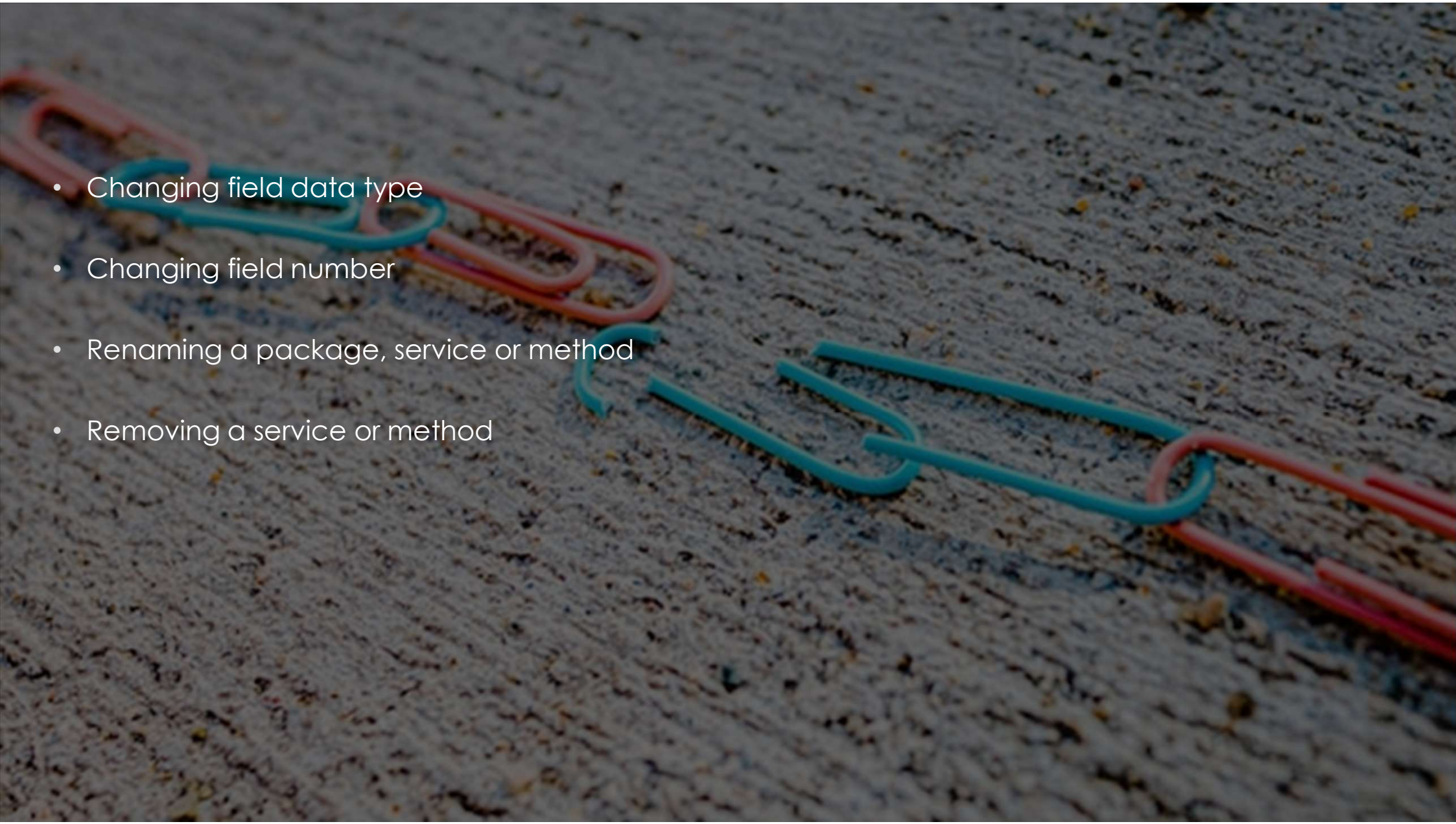
HelloReply

```
<ItemGroup>
    <Protobuf Include="Protos\greet.proto" GrpcServices="Server"/>
</ItemGroup>
```

- Both(default)
- Server
- Client
- None

- Smaller and faster than JSON
- Transmitted in binary format
- Value of each field serialised against the unique identifier for that field


BREAKING CHANGES



- Changing field data type
- Changing field number
- Renaming a package, service or method
- Removing a service or method

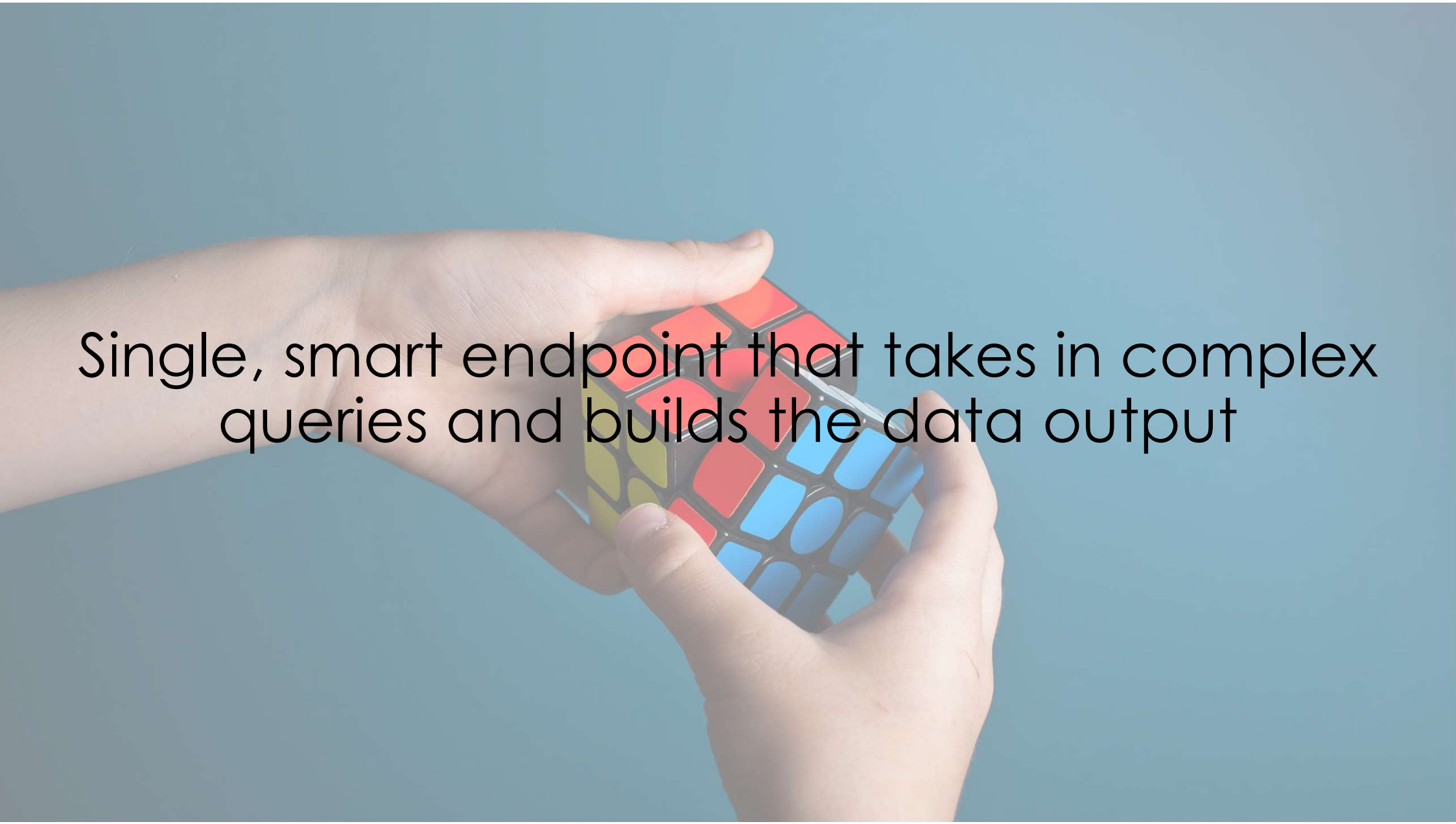
Deprecating Versions

```
// The request message containing the user's name.
message HelloRequest {
    // name to say hello to
    string name = 1 [deprecated = true];
  }
```

GraphQL

GraphQL is a query language


Supported by a runtime for fulfilling your queries



Single, smart endpoint that takes in complex queries and builds the data output

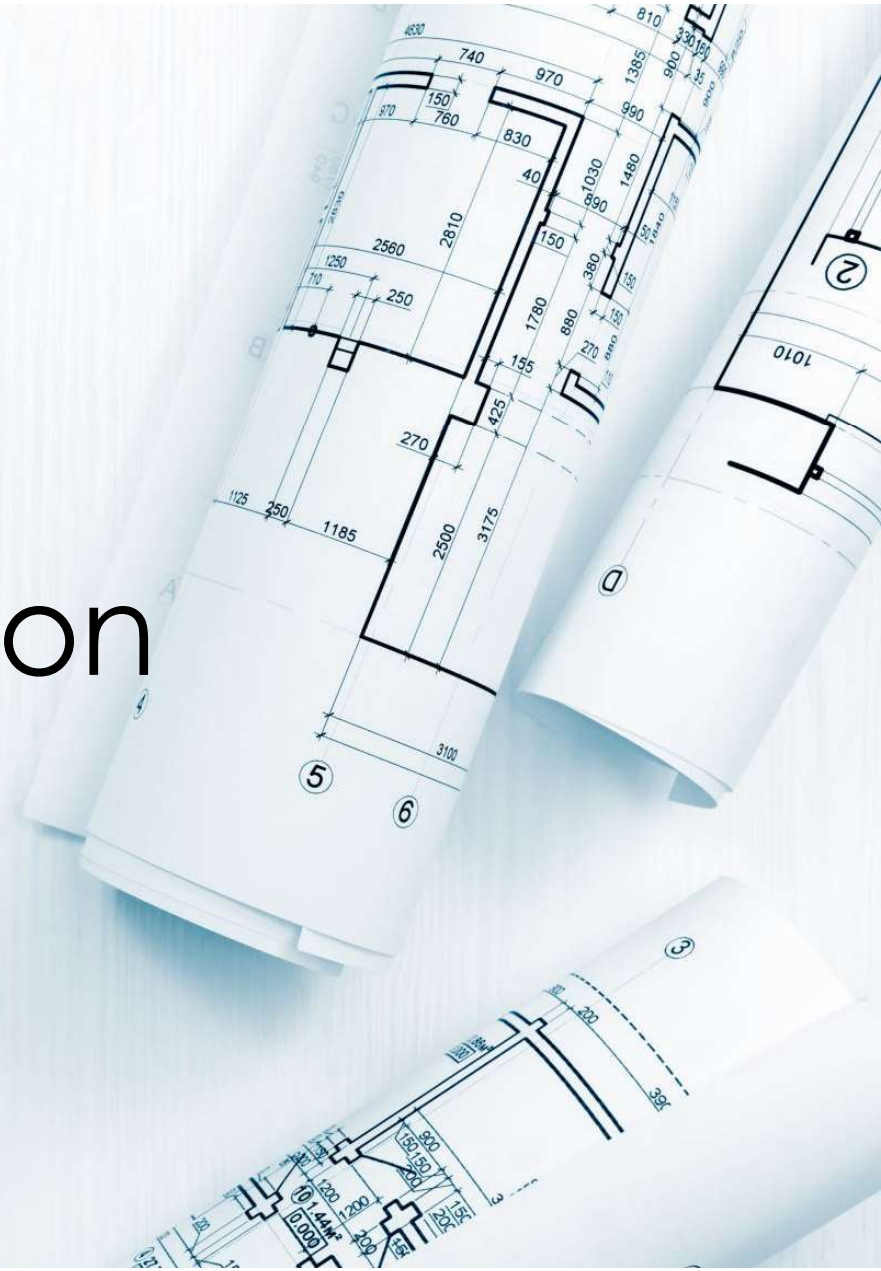
```
{
  hero {
    name
    friends {
      name
    }
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "friends": [
        {
          "name": "Luke Skywalker"
        },
        {
          "name": "Han Solo"
        },
        {
          "name": "Leia Organa"
        }
      ]
    }
  }
}
```

- Clients specify the shape of the data they need
- Server responds with the exact same data graph as the query
- Query related entities

Client-focused, experience driven

Specification

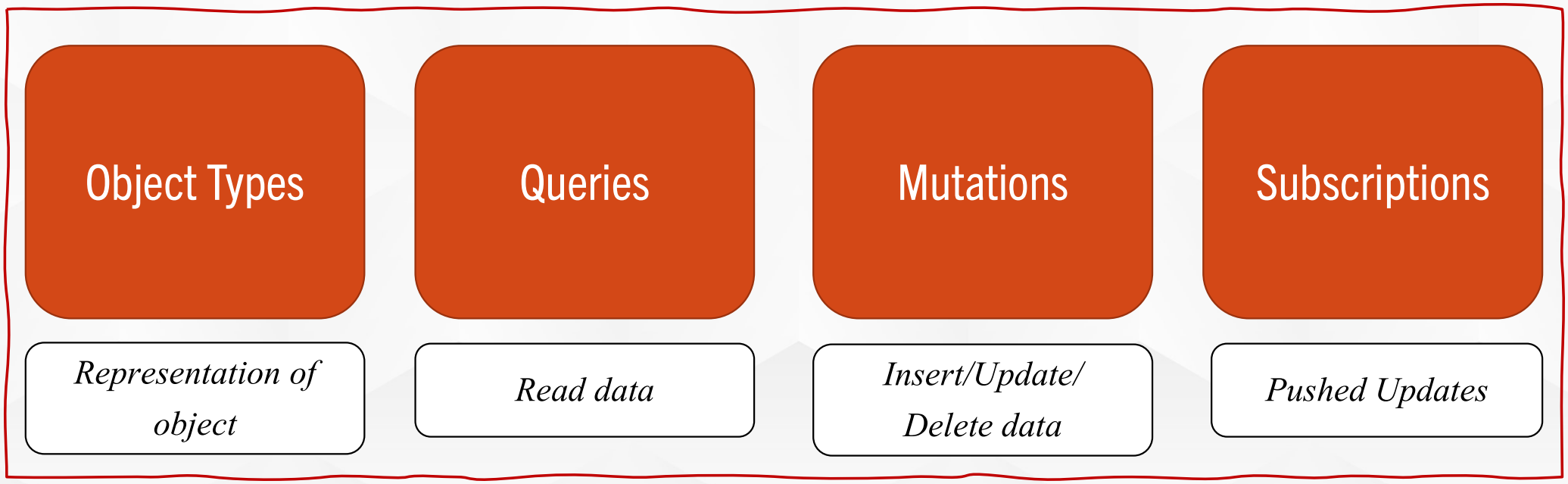


Various implementations

Hot Chocolate

GraphQL.NET

SCHEMA



Types

FIELDS

Name	Type	Resolvers
Name of the field	*Type of the field*	*Methods that help fields resolve to data*

Creating Types Using Hot Chocolate

Object Type

Public properties = Fields

```
1  public class Place
2  {
3      public string Country { get; set; }
4
5      public string City { get; set; }
6  }
```

“get” = Resolver

Query

“Places” field

```
1  public class Query
2  {
3      public List<Place> GetPlaces([Service] PlaceRepository placeRepository)
4          => placeRepository.Places.ToList();
5
6  }
7
```

Resolver

Startup

Add a graphql server

```
builder.Services
    .AddSingleton<PlaceRepository>()
    .AddGraphQLServer()
    .AddQueryType<Query>();


var app = builder.Build();

//middleware for Routing, CORS...

//default endpoint /graphql
app.MapGraphQL();

app.Run();
```

Map endpoint “/graphql”

GraphQL is version free

Multiple schemas on same server

Extend Query

“RandomPlace” field

```
public class NewQuery : Query
{
    public Place GetRandomPlace([Service] PlaceRepository placeRepository, string place)
    {
        var random = placeRepository.Random.Next(0, placeRepository.Places.Count);
        return placeRepository.Places[random];
    }
}
```

Startup

Add a new server "newgraphql"

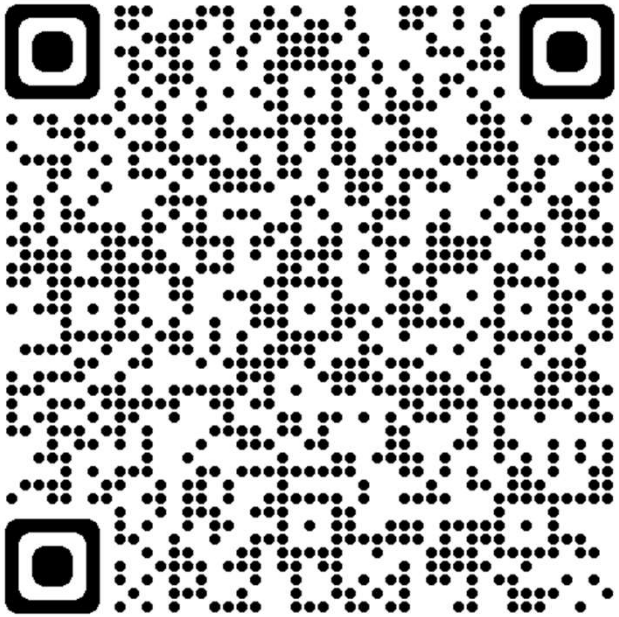
Map a new endpoint "/newgraphql"

```
builder.Services
    .AddSingleton<PlaceRepository>()
    .AddGraphQLServer()
    .AddQueryType<Query>()
    .AddGraphQLServer("newgraphql")
    .AddQueryType<NewQuery>();


var app = builder.Build();

//middleware for Routing, CORS...

//default endpoint /graphql
app.MapGraphQL();

//new endpoint /newgraphql
app.MapGraphQL("/newgraphql", "newgraphql");

app.Run();
```

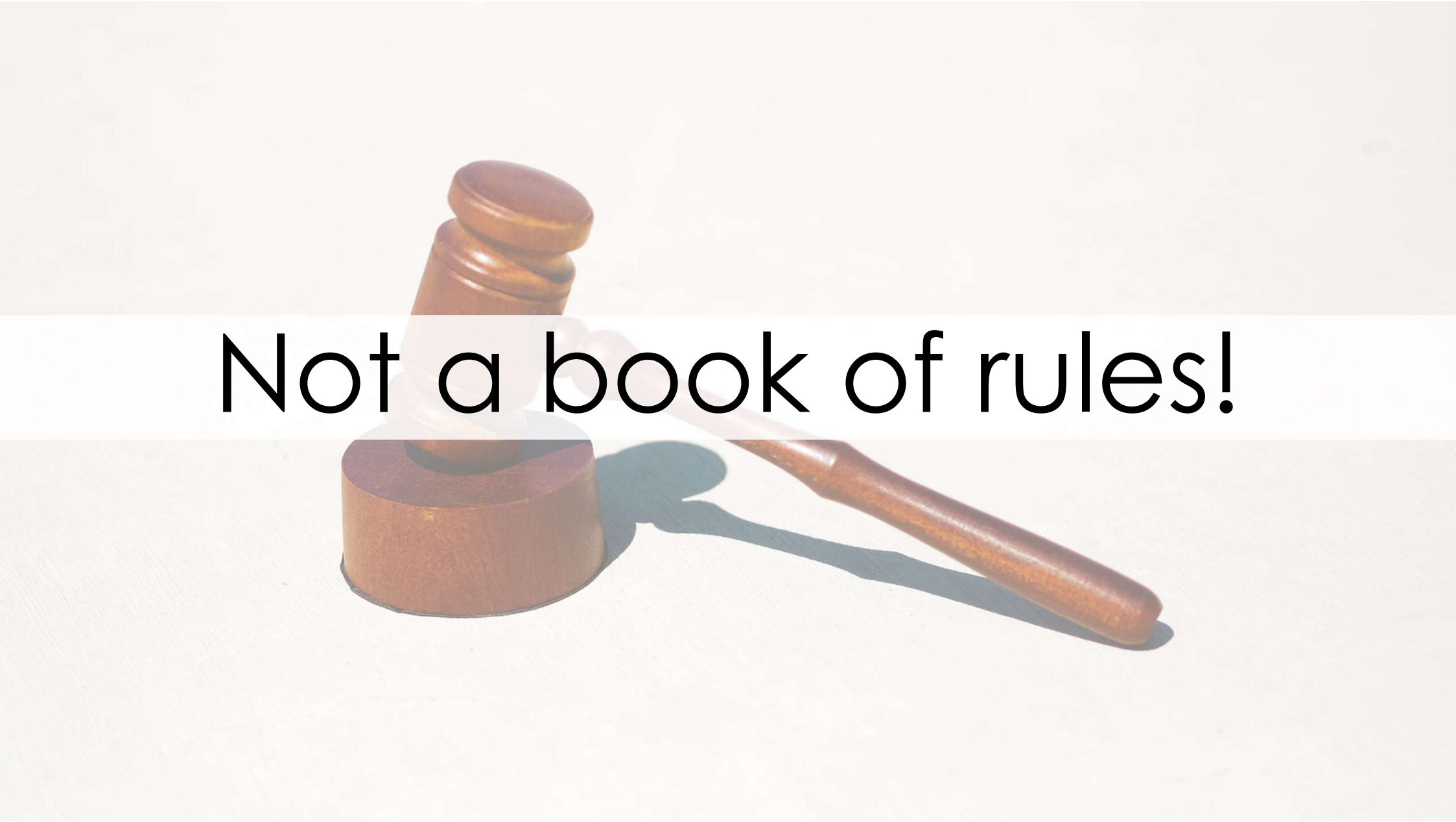
```
1
2  [GraphQLDeprecated("Use the `capital` field instead")]
3  public string City { get; set; }
```

Documentation

- Elements, Redoc, Postman, Swaggerhub
- GenDocu for gRPC Service
- SpectaQL, MagiDoc, DociQL, GraphDoc for GraphQL



Key Takeaways



Not a book of rules!

Strategy & Best Practice

- Always consider a path of no/least breaking changes
- Adopt API Versioning for breaking changes
- Keep your versioning strategy consistent
- Share your API Contract
- Document! Document! Document!
- Communicate! Communicate ! Communicate!
- Sunset old API versions regularly
- Try and adopt an api versioning strategy early on in your project

API CLIENTS ARE USERS!

https://bit.ly/version-apis

@poornimanayar