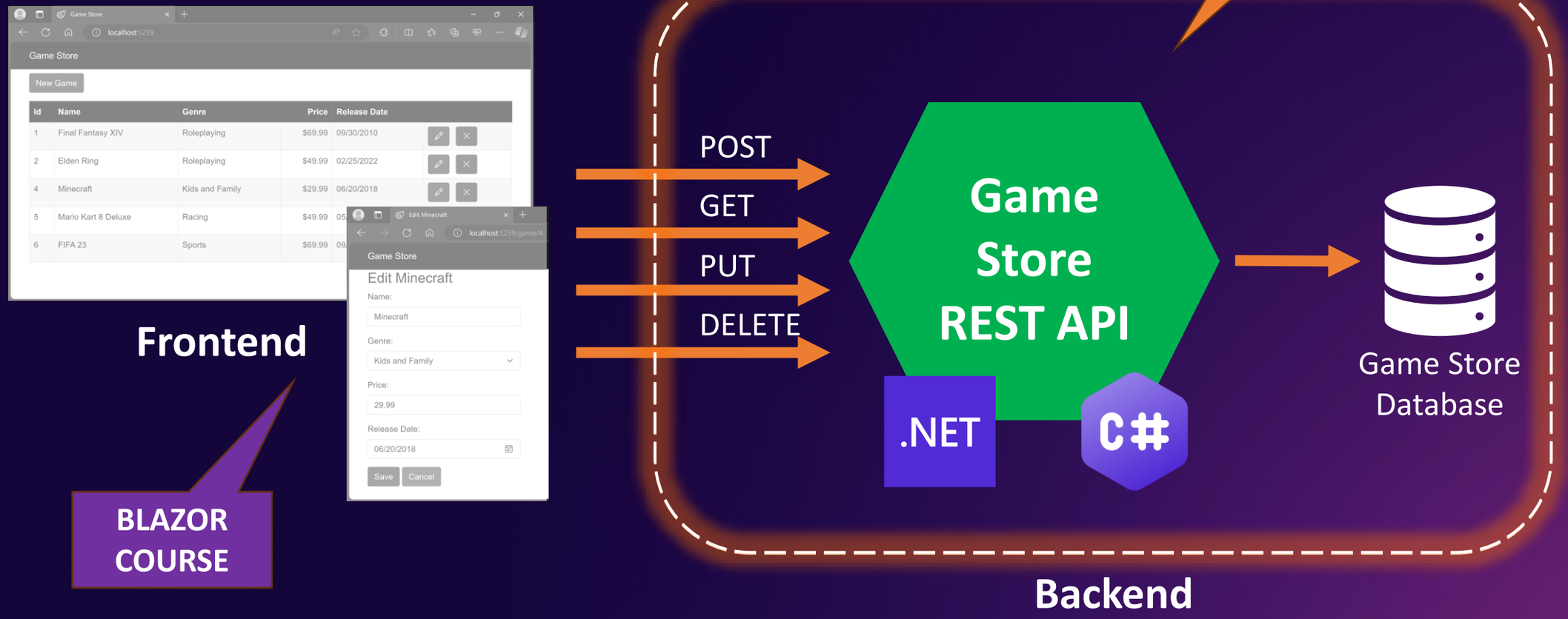
What you are going to build



What this course covers

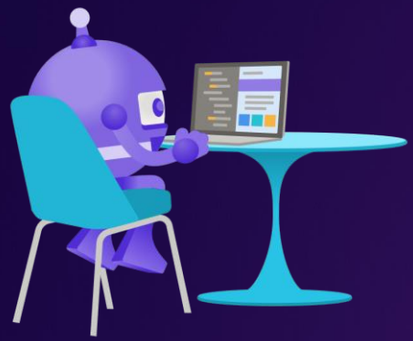
- Create ASP.NET Core Apps
- Understand REST APIs
- Implement CRUD Endpoints
- Data Transfer Objects (DTOs)
- Extension Methods
- Route Groups
- Handle Invalid Inputs
- Entity Framework Core
- Configuration System
- Dependency Injection
- Service Lifetime
- Mapping Entities to DTOs
- Asynchronous Programming
- Frontend Integration

Is this course for you?

Basic C# or Java Knowledge

Web Development Essentials

Some Database Experience



Beginner Level Course

dotnetacademy.io

Software prerequisites

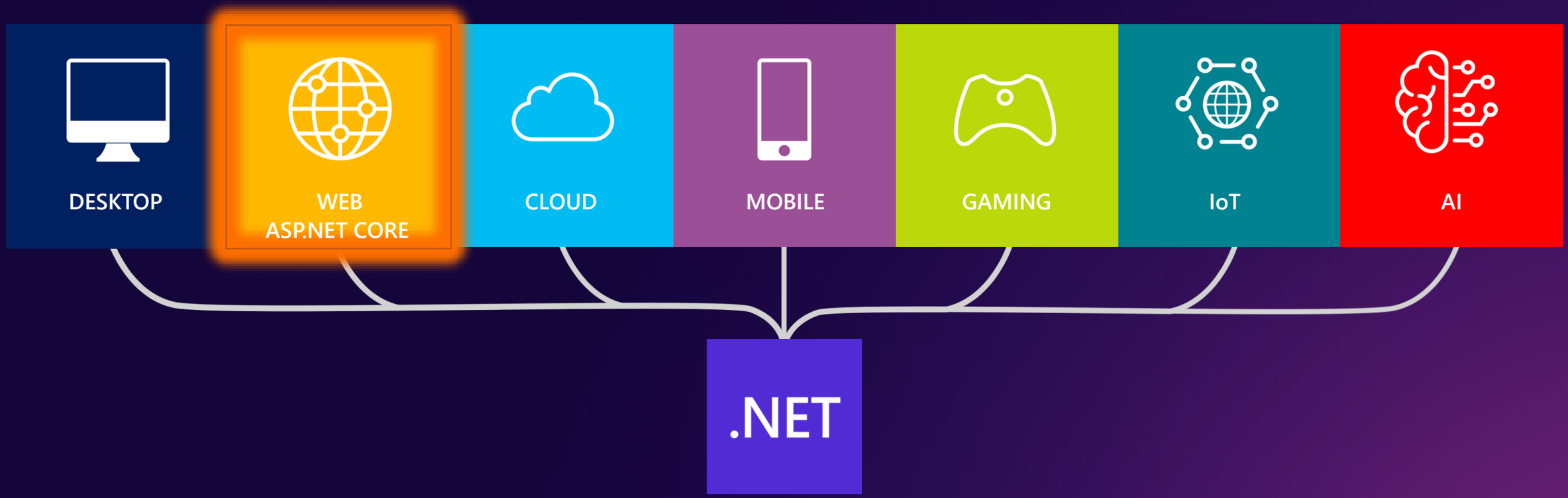
.NET SDK
https://dot.net/download

Visual Studio Code
https://code.visualstudio.com

What is ASP.NET Core?

ASP.NET Core is a popular web-development framework for building web apps on the .NET platform

What is.NET?



A free, cross-platform, open source developer platform
for building many different types of applications.

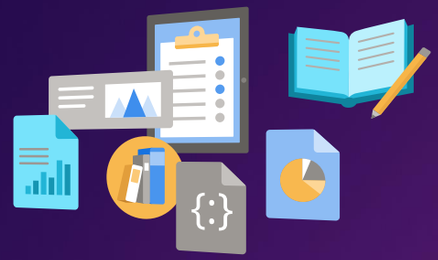
ASP.NET CORE

Multiple Languages

Open Source

Tools

Cross Platform



Libraries

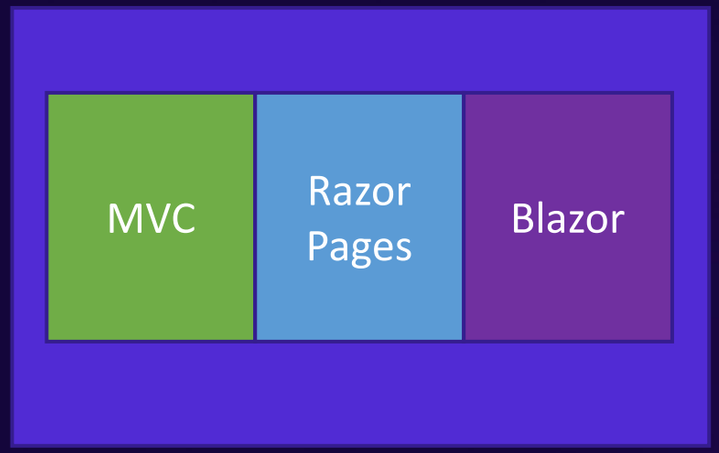
Flexible deployment

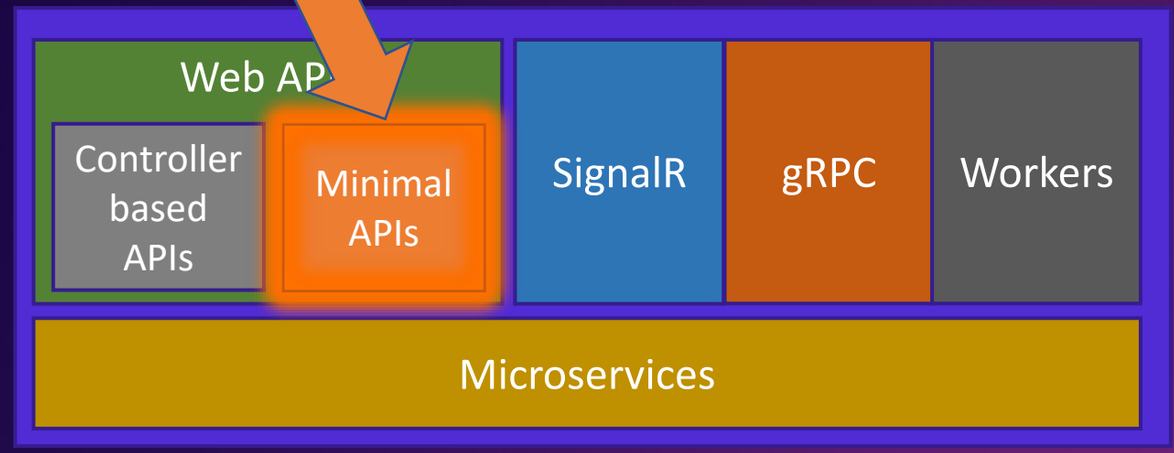
What can you create with ASP.NET Core?

Web UI

THIS COURSE COVERS

Backend Services

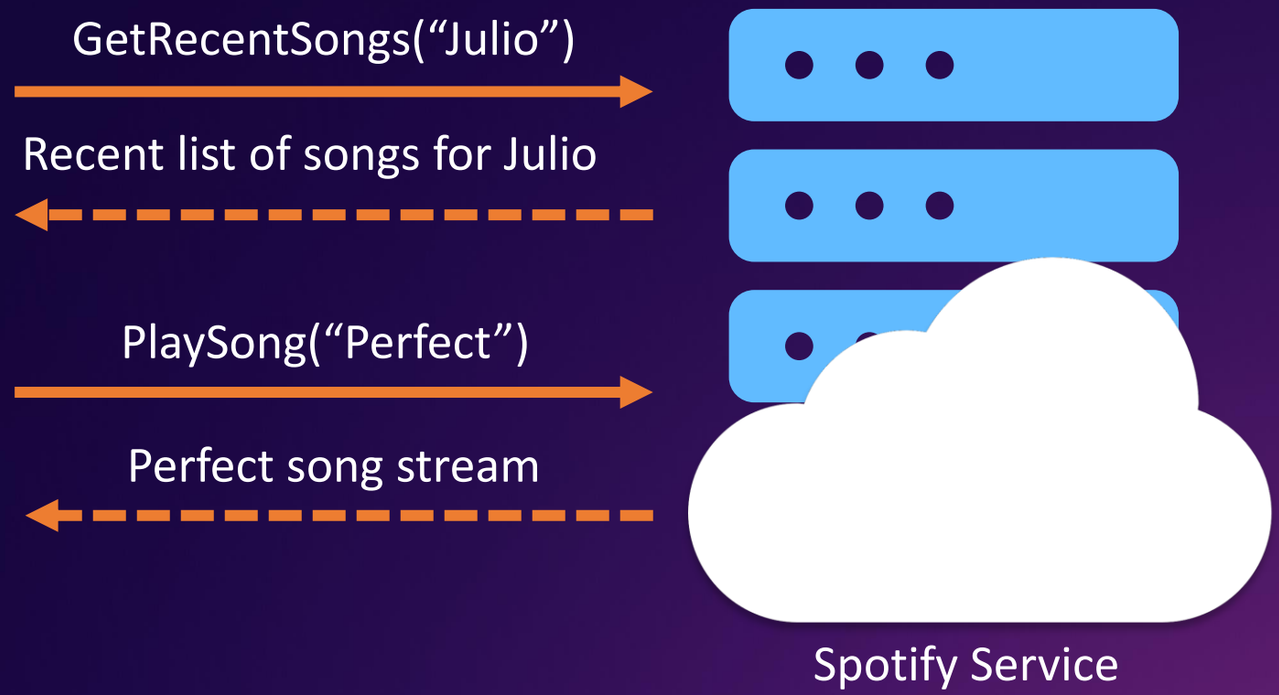




Clients and Servers

What is an API?

Application **P**rogramming **I**nterface



An API helps clients communicate what they want to the service so it can understand and fulfill the request.

What is REST?

REpresentational

State

Transfer

- Stateless
- Client-Server
- Uniform interface
- Layered system
- Cacheable
- Code on demand

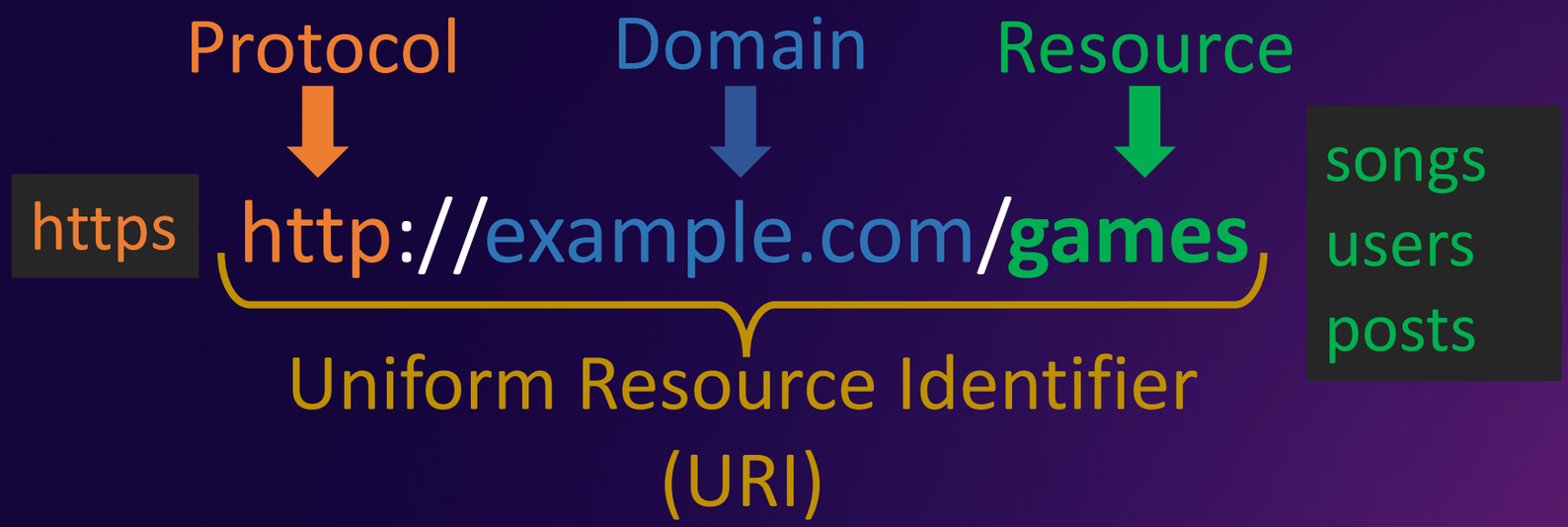
A set of guiding principles that impose conditions on how an API should work

What is a REST API?

A REST or RESTFUL API is one that conforms to the REST architectural style

How to identify resources in a REST API?

A resource is any object, document or thing that the API can receive from or send to clients

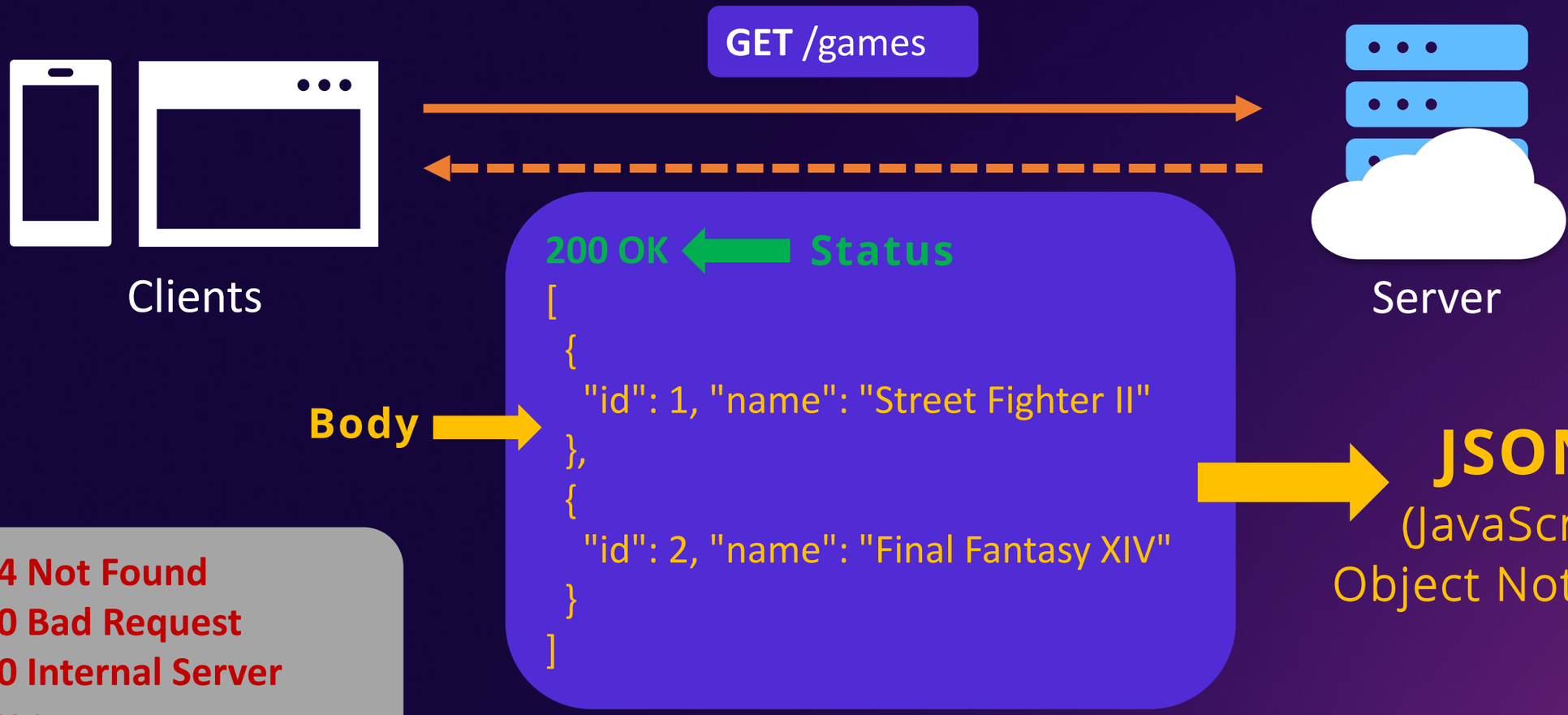


How to interact with a REST API?

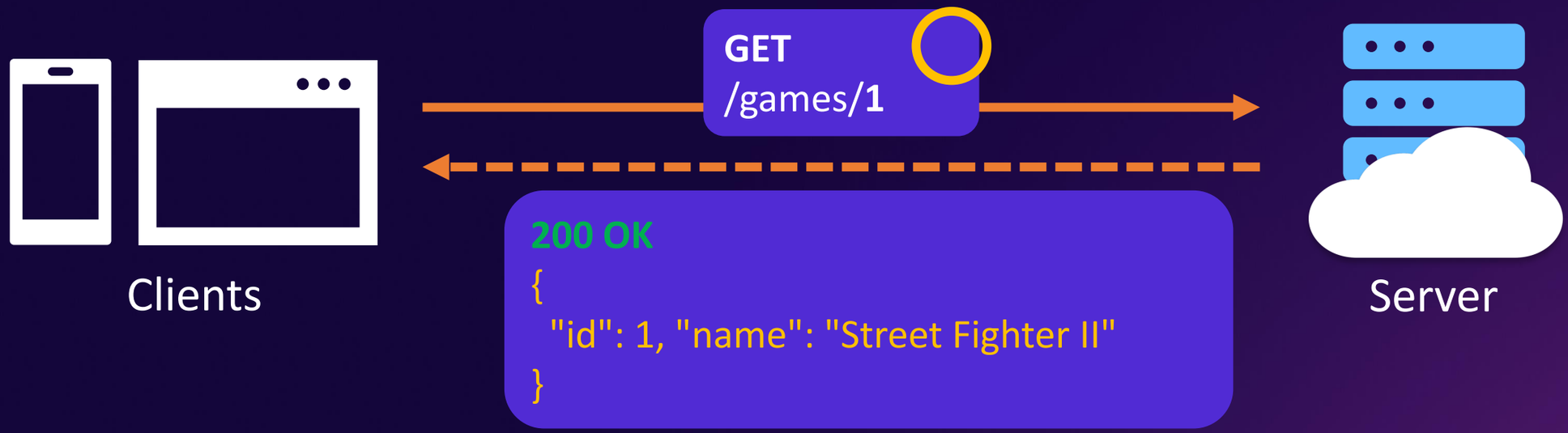
HTTP Methods

Create	POST	Creates a new resource
Read	GET	Retrieves the resource representation/state
Update	PUT	Updates an existing resource
Delete	DELETE	Deletes a resource

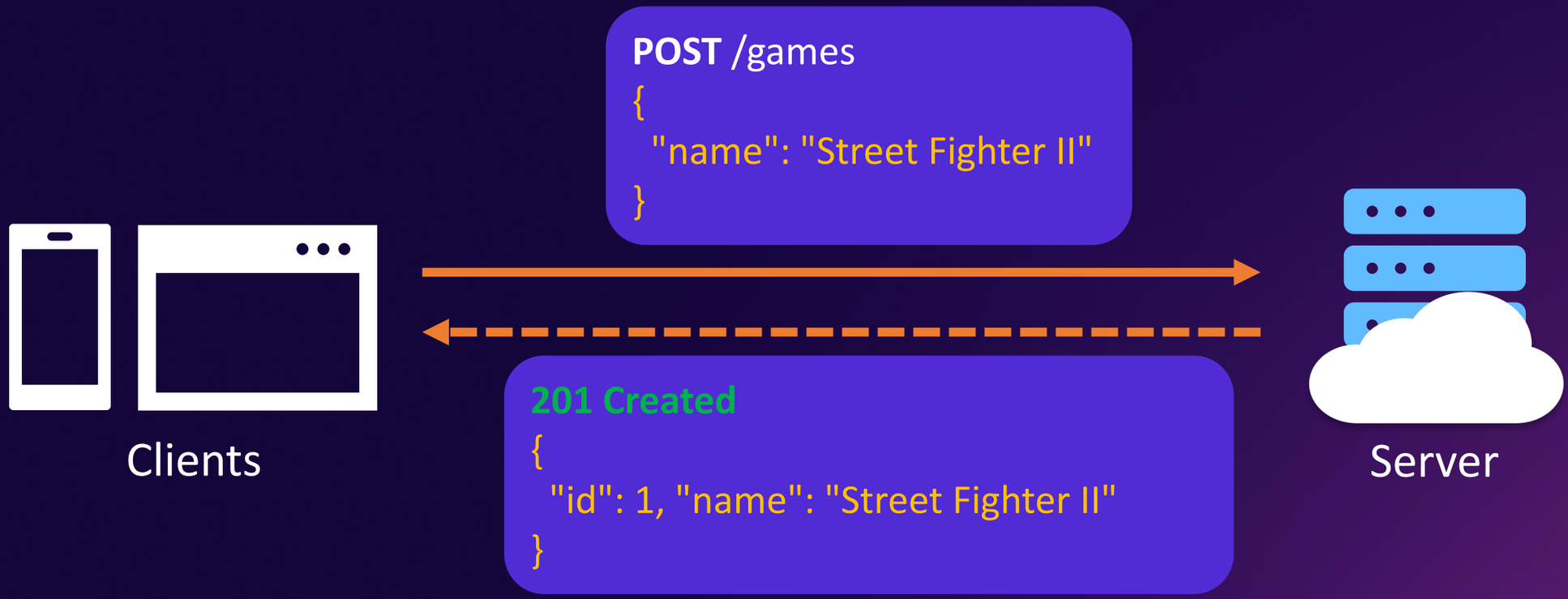
Get All Games - HTTP GET



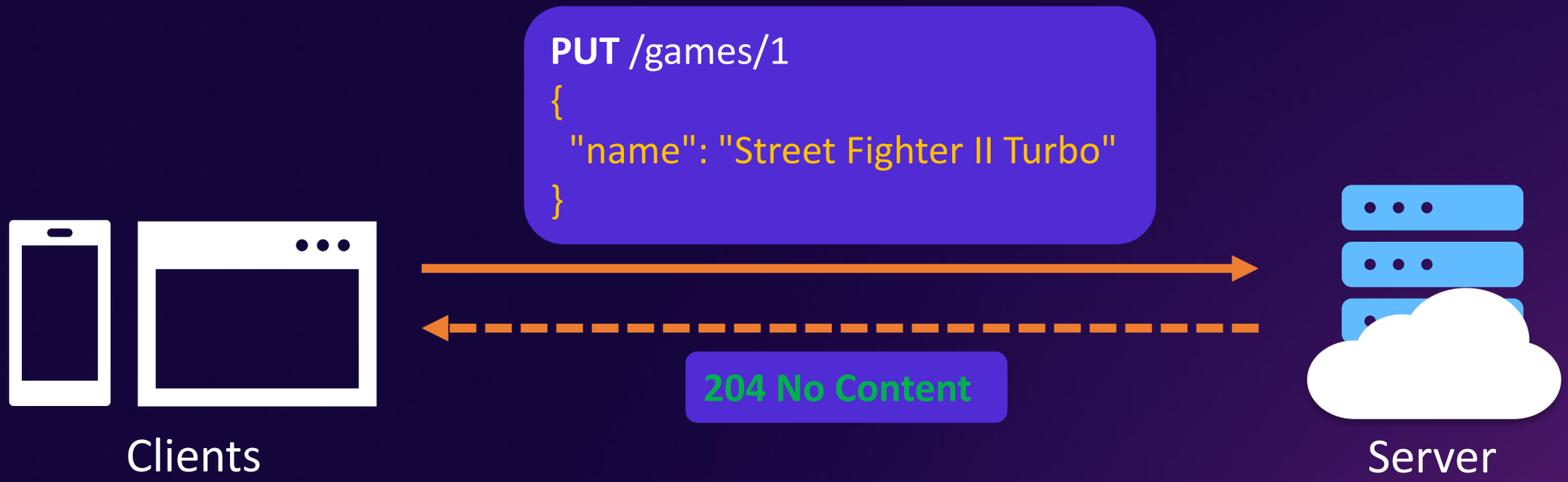
Get A Specific Game - HTTP GET



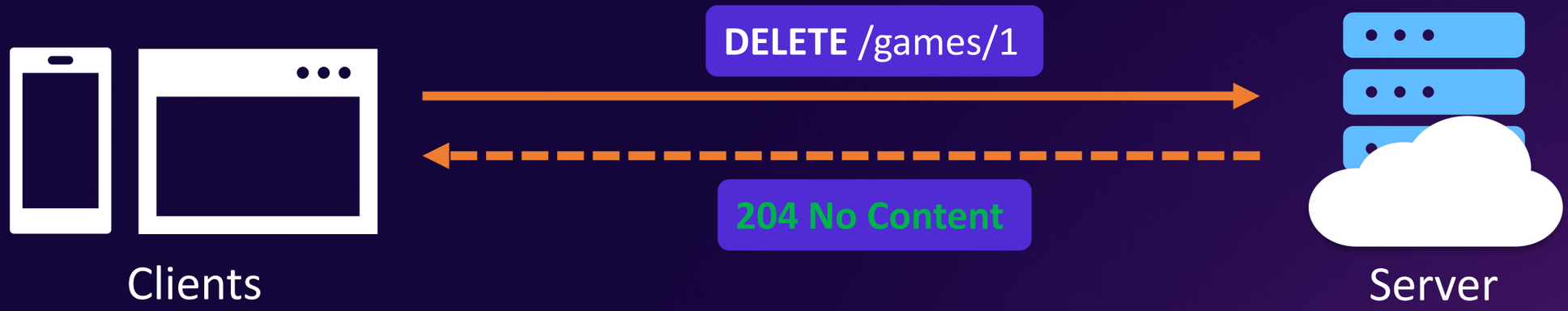
Create A Game - HTTP POST



Update A Game - HTTP PUT



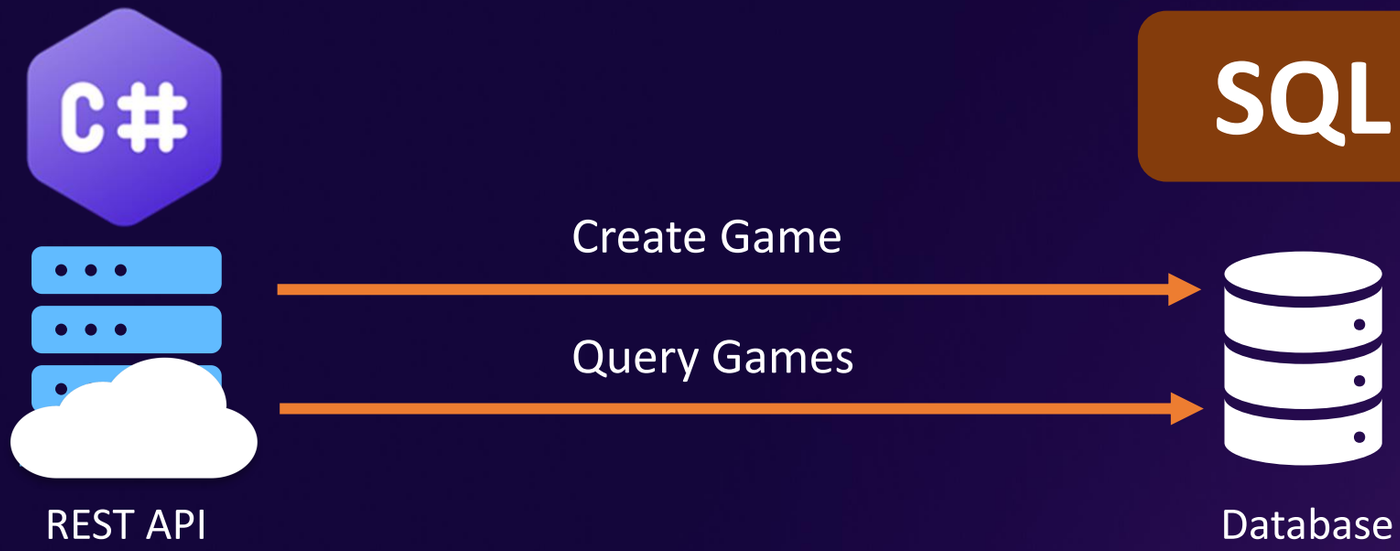
Delete A Game - HTTP DELETE



Games REST API

GET	/games
GET	/games/1
POST	/games
PUT	/games/1
DELETE	/games/1

The Need For Object-Relational Mapping (O/RM)



Translate Web API request to SQL query

Send SQL query to database server

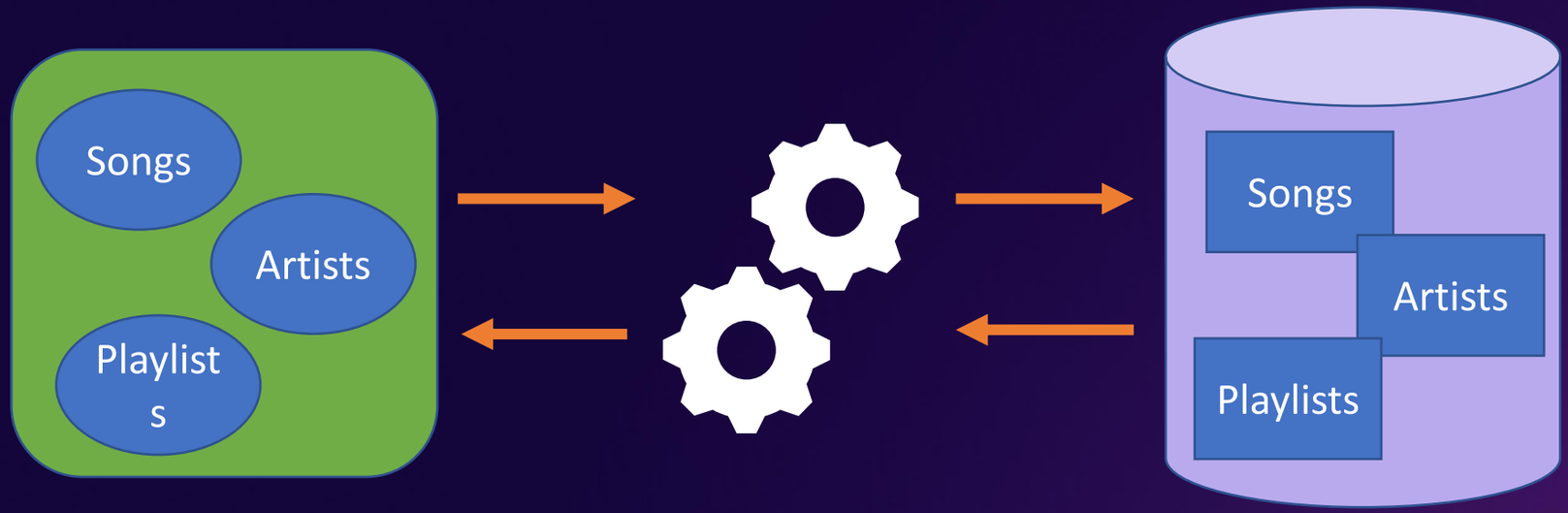
Translate database rows to Web API response

Read resulting database rows

Problems

- Need to learn new language
- Need a lot of additional data-access code
- Error prone
- Need to manually keep C# models in sync with DB tables

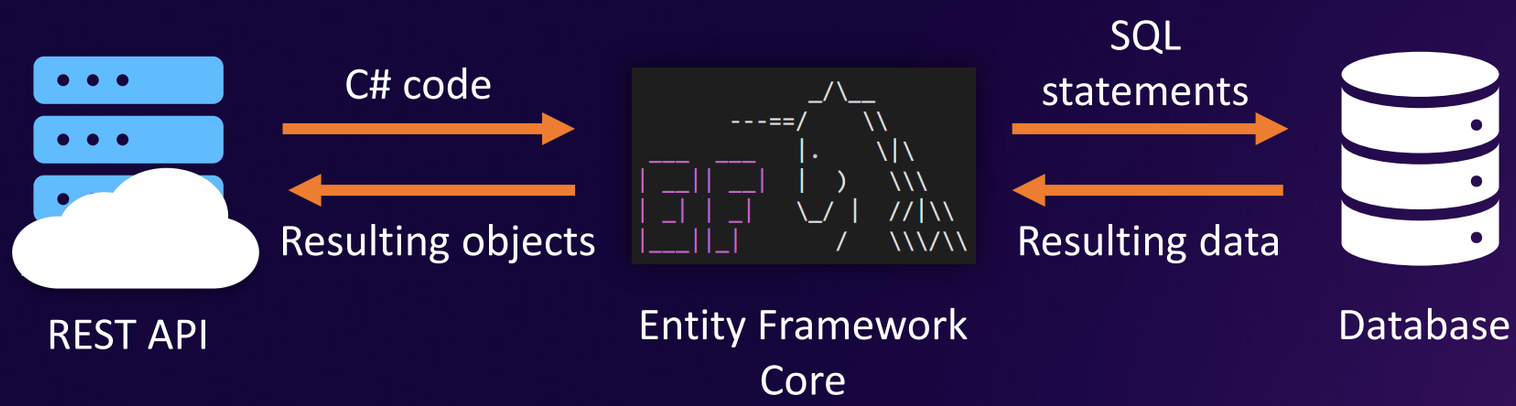
What is Object-Relational Mapping (O/RM)?



A technique for converting data between a relational database and an object-oriented program

What is Entity Framework Core?

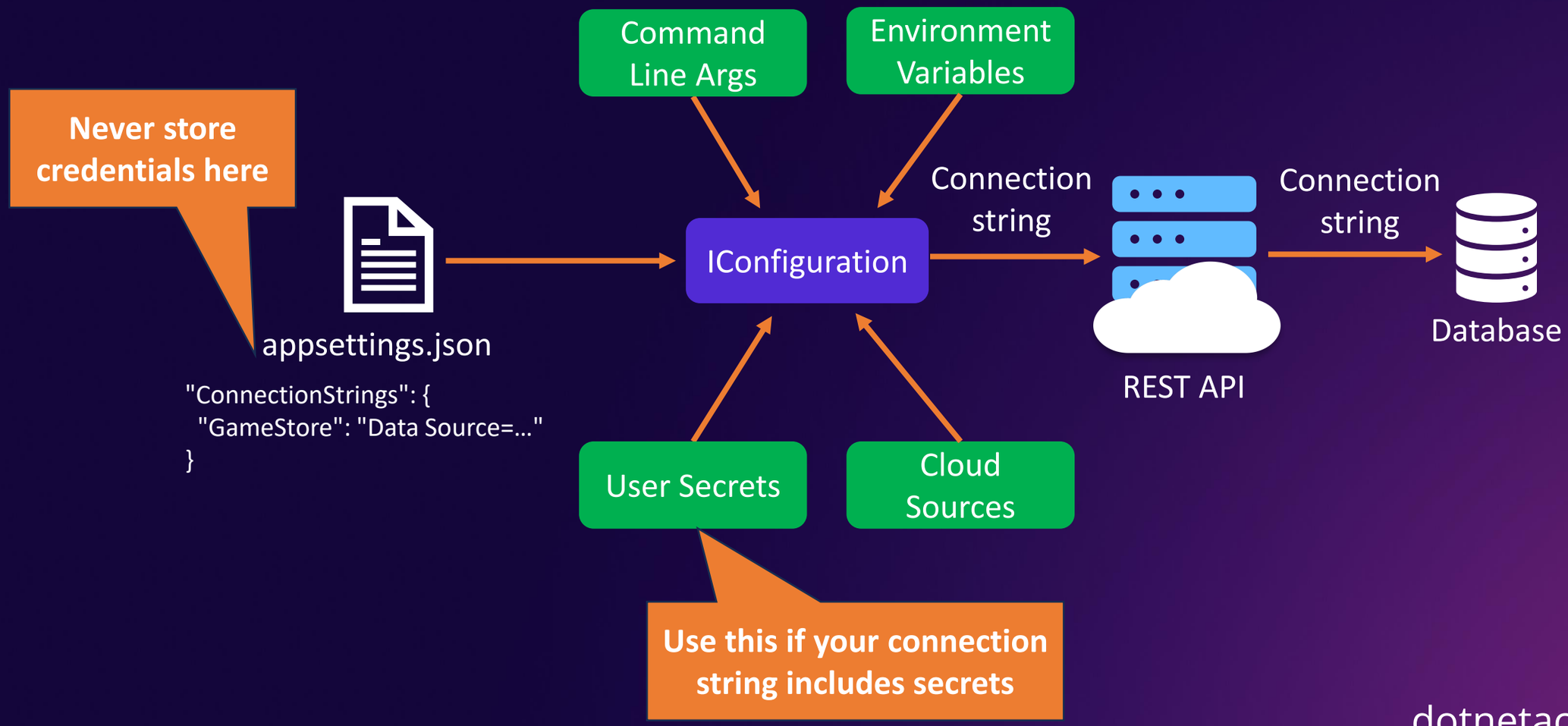
A lightweight, extensible, open source and cross-platform object-relational mapper for .NET



Benefits

- No need to learn a new language
- Minimal data-access code (LINQ)
- Tooling to keep C# models in sync with DB tables
- Change tracking
- Multiple database providers

The ASP.NET Core Configuration System



What is a Dependency?


```
public MyService()
{
   var logger = new MyLogger();
   logger.LogThis("I'm Ready!");
}
```

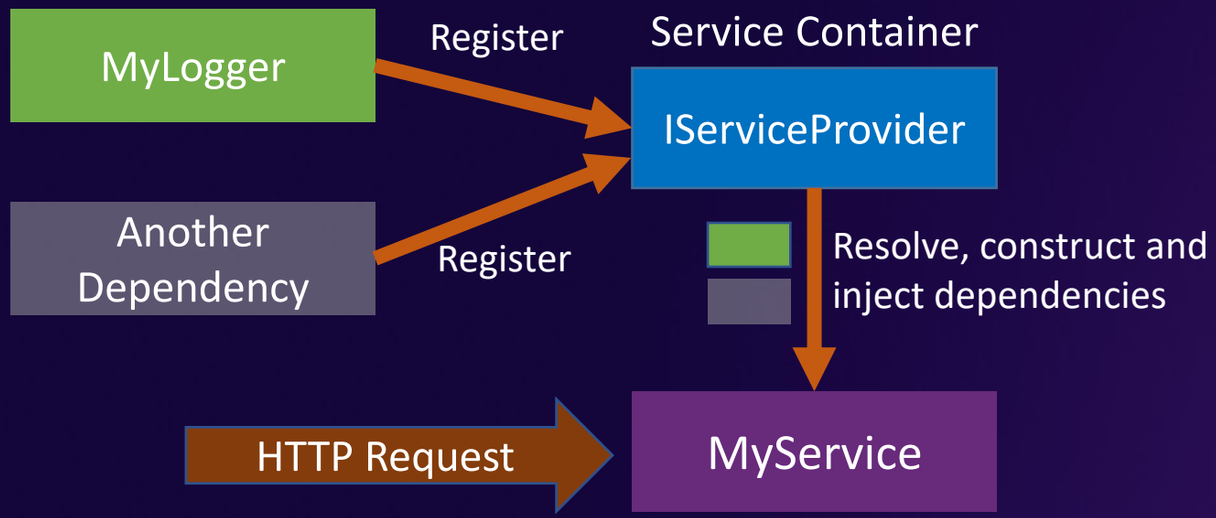
```
public MyService()
{
   var writter = new MyFileWritter("output.log");
   var logger = new MyLogger(writter);
   logger.LogThis("I'm Ready!");
}
```

Problems

- MyService is tightly coupled to the Logger dependency. Any changes to MyLogger require changes to MyService.
- MyService needs to know how to construct and configure the MyLogger dependency.
- It's hard to test MyService since the MyLogger dependency cannot be mocked or stubbed.

What is Dependency Injection?

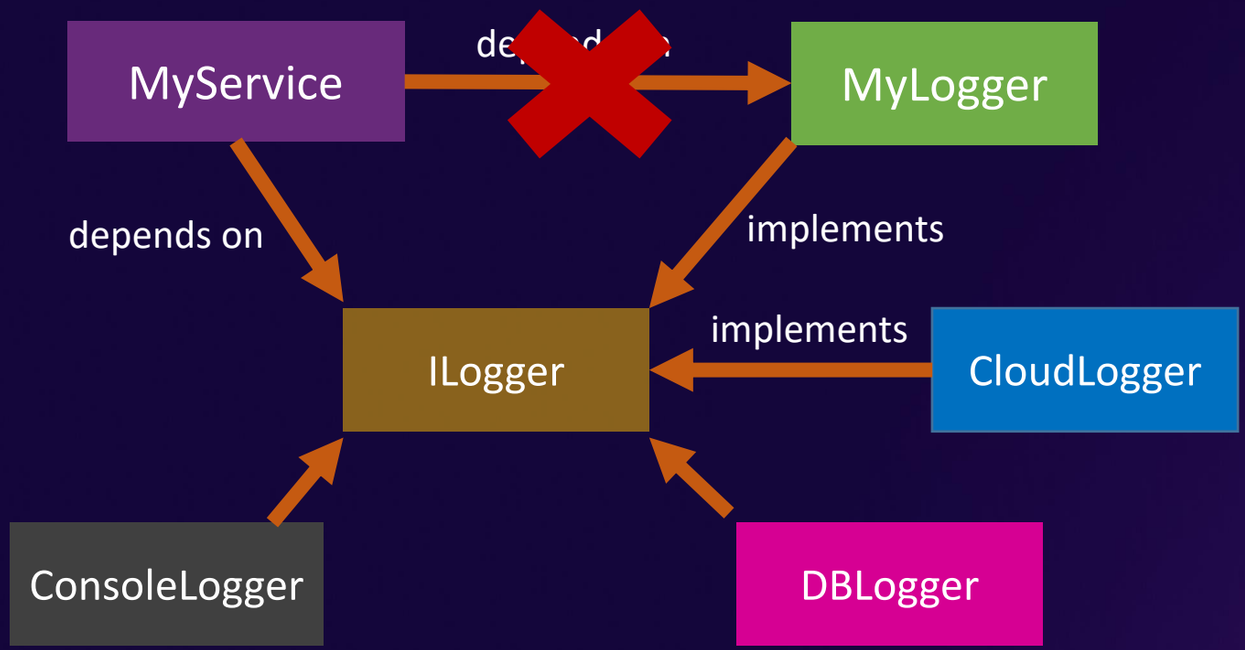

```
public MyService(MyLogger logger)
{
    logger.LogThis("I'm Ready!");
}
```



Benefits

- MyService won't be affected by changes to its dependencies.
- MyService doesn't need to know how to construct or configure its dependencies.
- Dependencies can also be injected as parameters to minimal API endpoints
- Opens the door to using Dependency Inversion

Using Dependency Inversion



The Dependency Inversion Principle

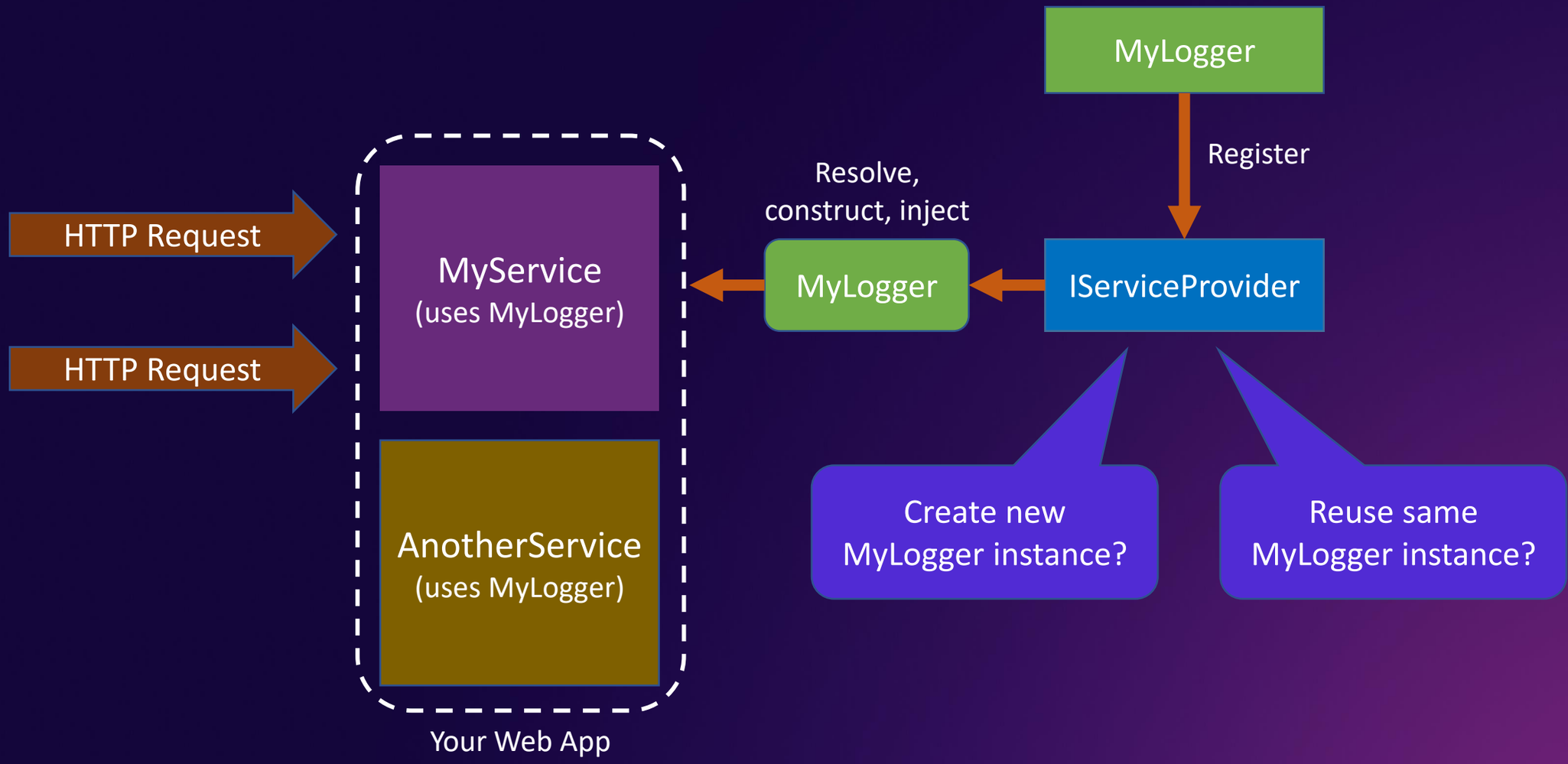
"Code should depend on abstractions as opposed to concrete implementations."

Benefits

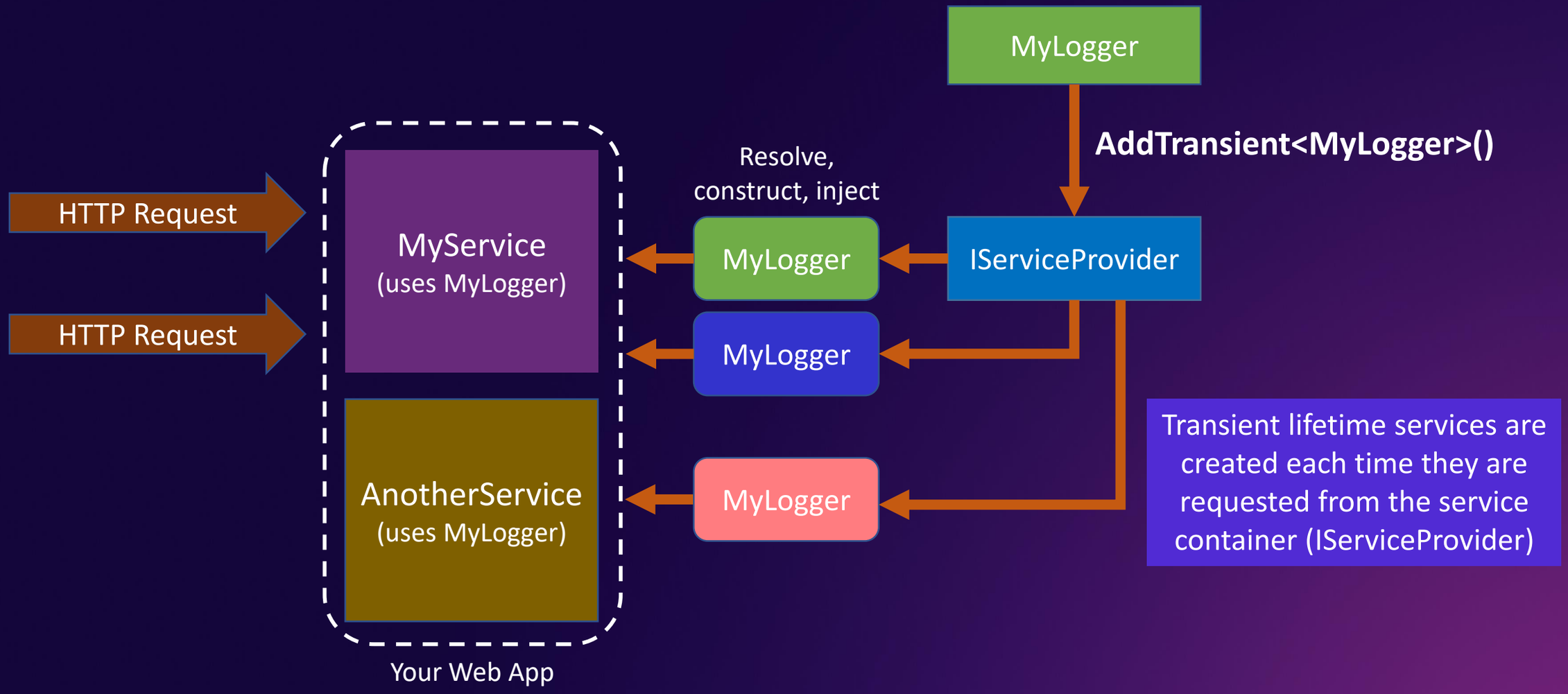
- The logger dependency can be swapped out for a different implementation without modifying MyService
- It's easier to test MyService since the logger dependency can be mocked or stubbed
- Code is cleaner, easier to modify and easier to reuse

```
public MyService(ILogger logger)
{
   logger.LogThis("I'm Ready!");
}
```

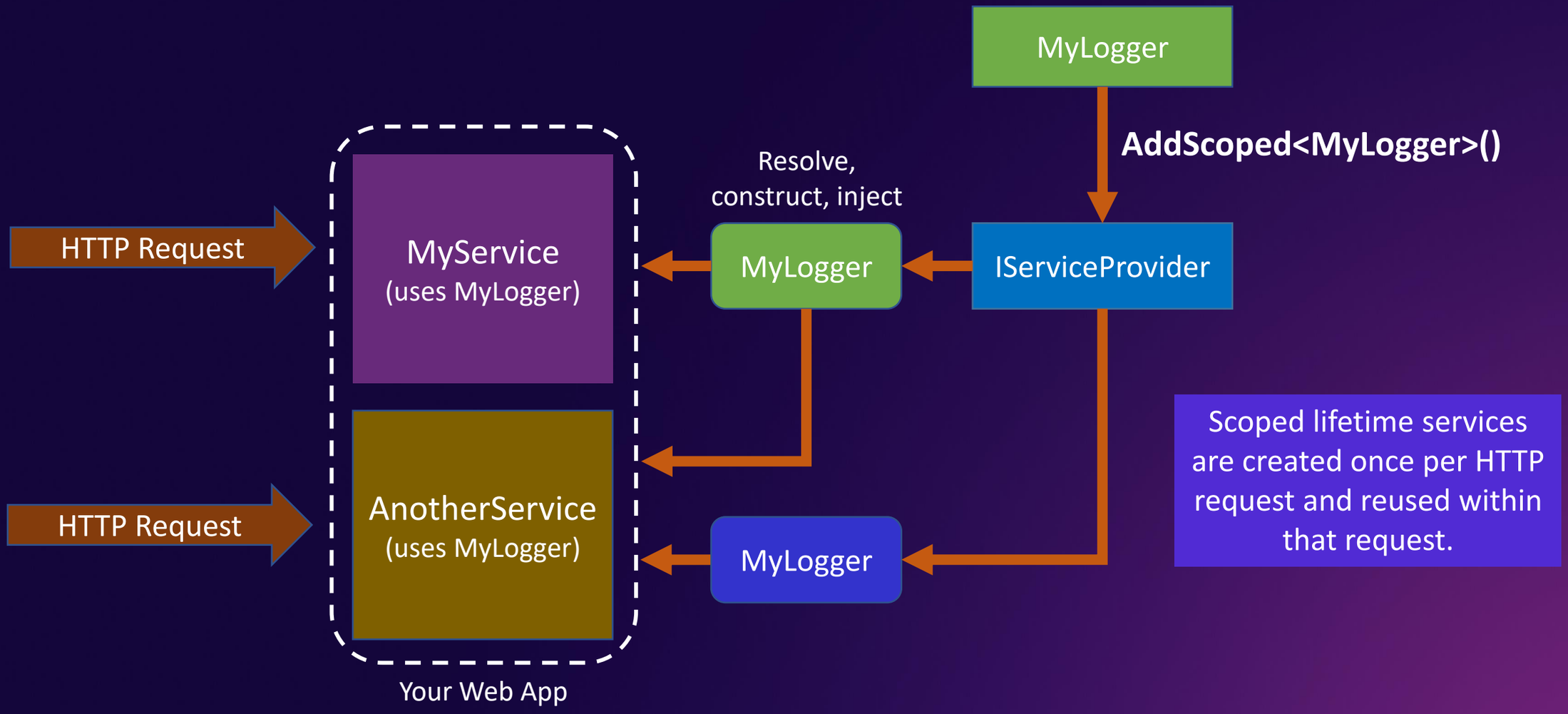
When should instances be created?



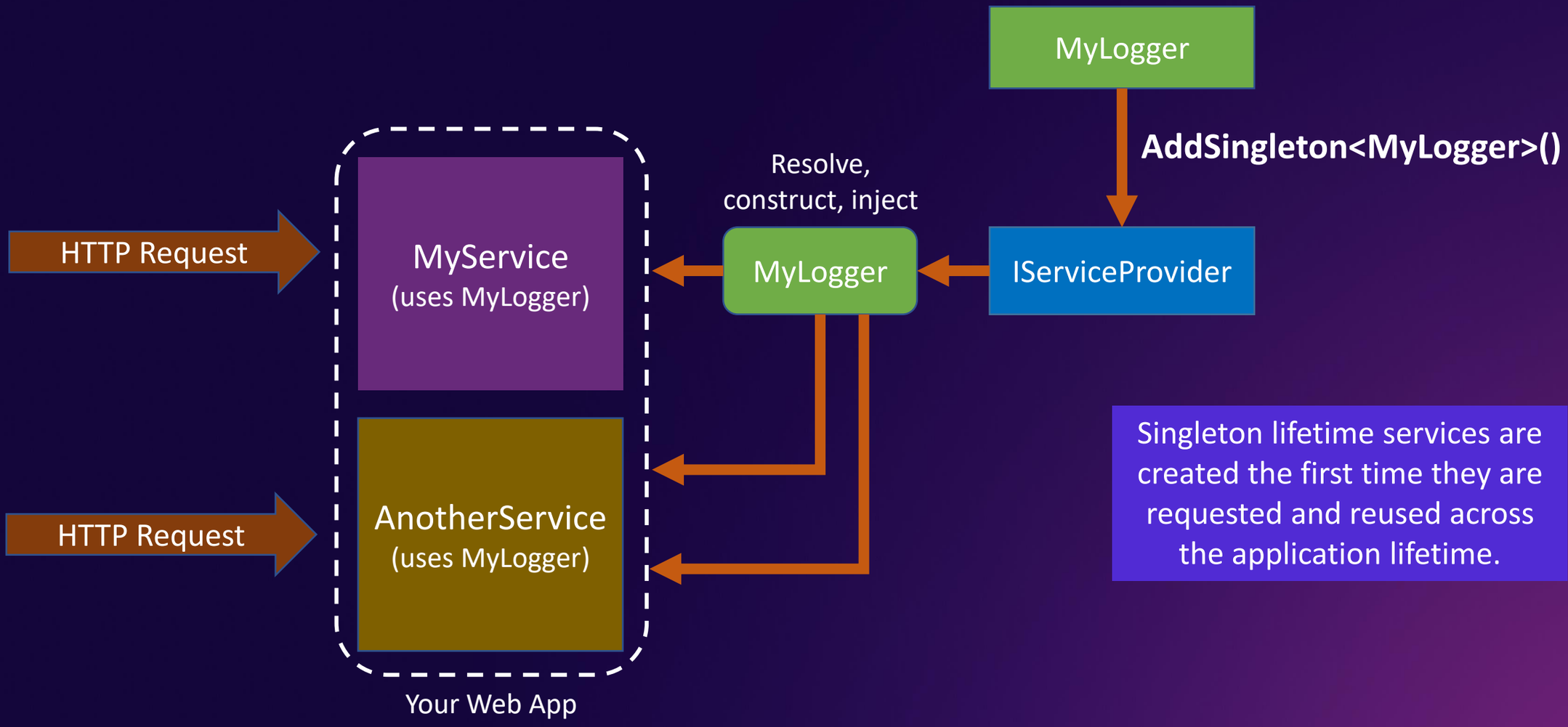
The Transient Service Lifetime



The Scoped Service Lifetime

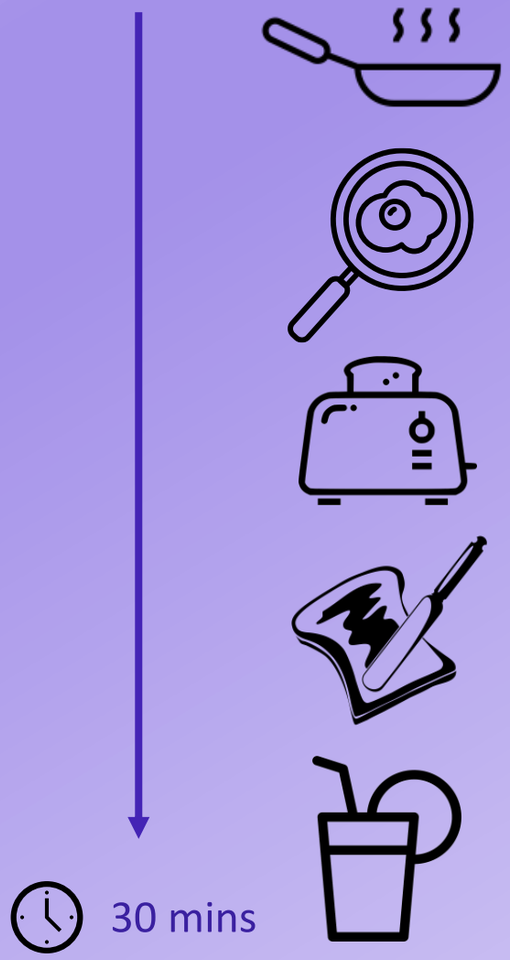


The Singleton Service Lifetime



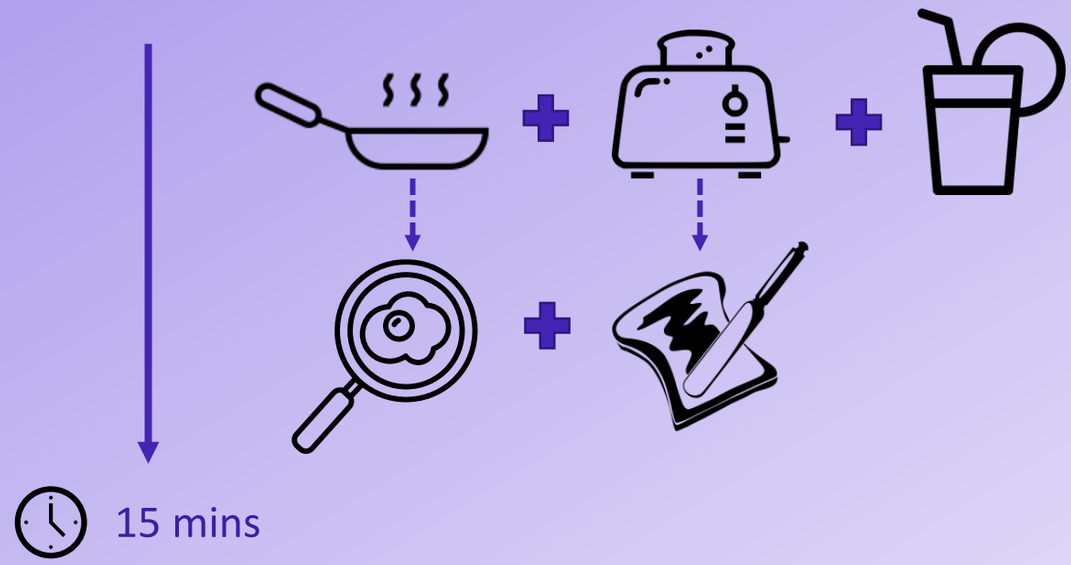
Singleton lifetime services are created the first time they are requested and reused across the application lifetime.

Performing asynchronous work



30 mins

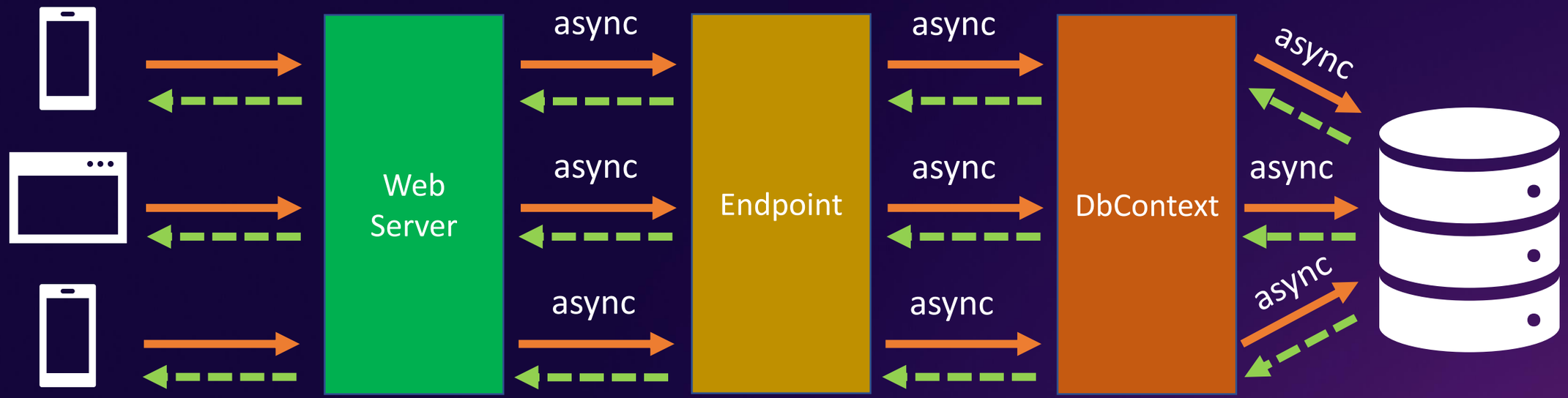
Synchronous



15 mins

Asynchronous

Asynchronous programming in ASP.NET Core



Benefits of the Asynchronous Programming Model

- **Improved Performance:** Avoids blocking callers, freeing them up for other tasks
- **Improved Scalability:** Allows your application to handle more requests and users simultaneously
- **Simplified code:** Asynchronous code is simple to write via task objects and the async and await keywords