Data Access in C# 10 Fundamentals

Introduction to Data Access in C#

Filip Ekberg
Principal Consultant & CEO

@fekberg fekberg.com

Data Access in C#

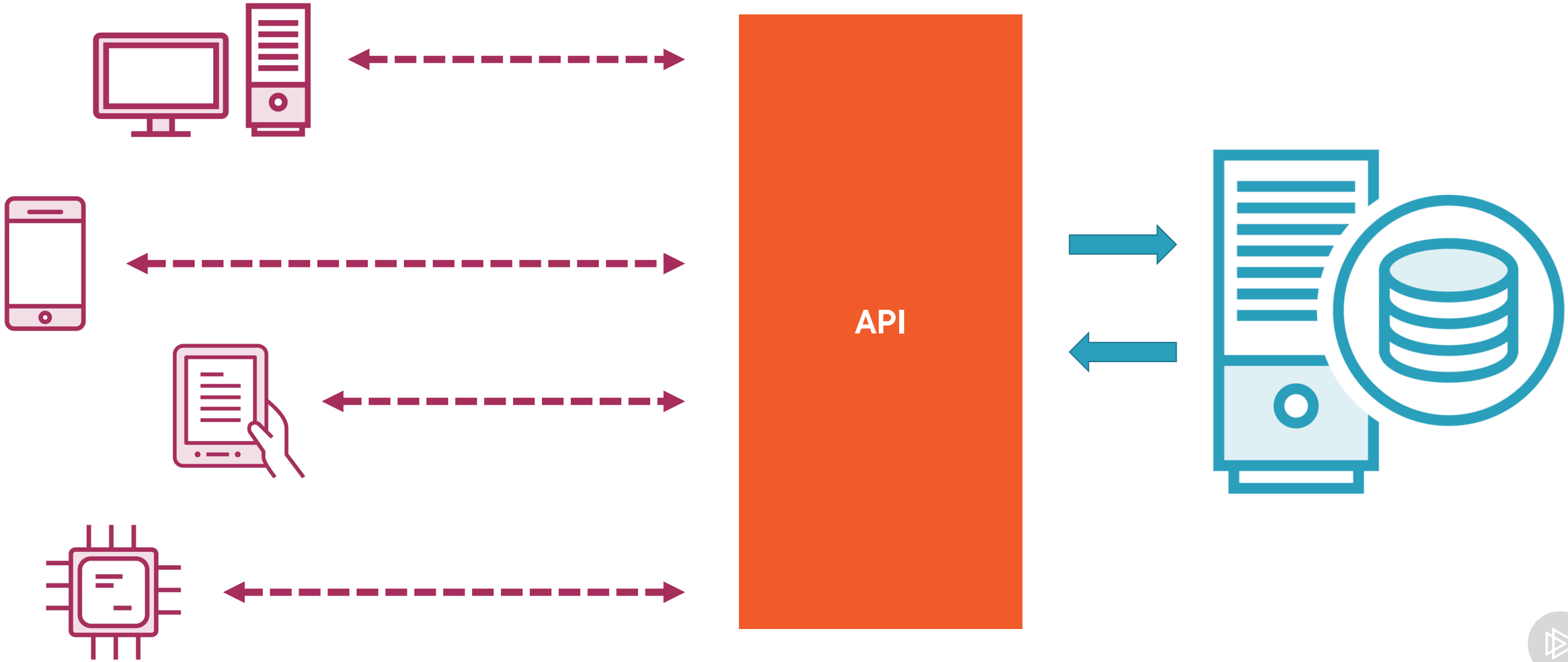
How do we access data?

What's suitable in my situation?

Applicable in any application!

Data access is a **fundamental** piece in **almost every application!**

Data Access from Desktop, Mobile & IoT



Data Access in C#

How do we access data?

What's suitable in my situation?

Applicable in any application!

Feel free to **ask questions!**

Overview

Entity Framework
Why is this so popular?

ADO.NET
When is this raw way of accessing data suitable?

Introducing a Data Access Layer
Repositories, Unit of Work & Lazy Loading

NoSQL
Document Database, Key-Value Store, Table Store

Accessing Data Everywhere
Different Applications, Best Practices, Variety of different data stores

Download the exercise

github.com/fekberg/csharp-data-access-fundamentals

Introduce a **database** to **allow data** to be **shared across applications** and **instances**

Which **Data Store** Should I Use?

Relational?

Non-relational?

Follow along using
the exercise files

Version Check

This version was created by using:

- C# 10
- .NET 6
- Entity Framework Core 6
- Visual Studio 2022

Relevant Notes

A note on frameworks, libraries and tools:

- Free community version of Visual Studio 2022
- Many features, libraries and concepts work in older versions of C# & .NET
- Applicable to all types of .NET applications

Different Types of **Data Stores**

Relational Databases

SQL Server

MySQL

Oracle

PostgreSQL

SQLite

Many more..

Relational Databases

SQL Server

MySQL

Oracle

PostgreSQL

SQLite

Many more..

Hosting a Database

Azure

On-premises

Docker

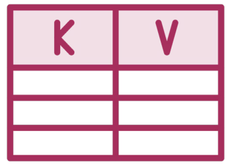
Azure SQL

Non-relational Database

Commonly referred to as NoSQL

A group of many different data stores

NoSQL Databases



Key-value store or Key-value cache

Document store or Document database

Object storage

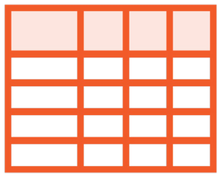
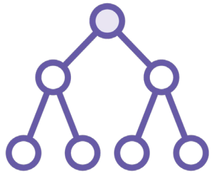


Table storage

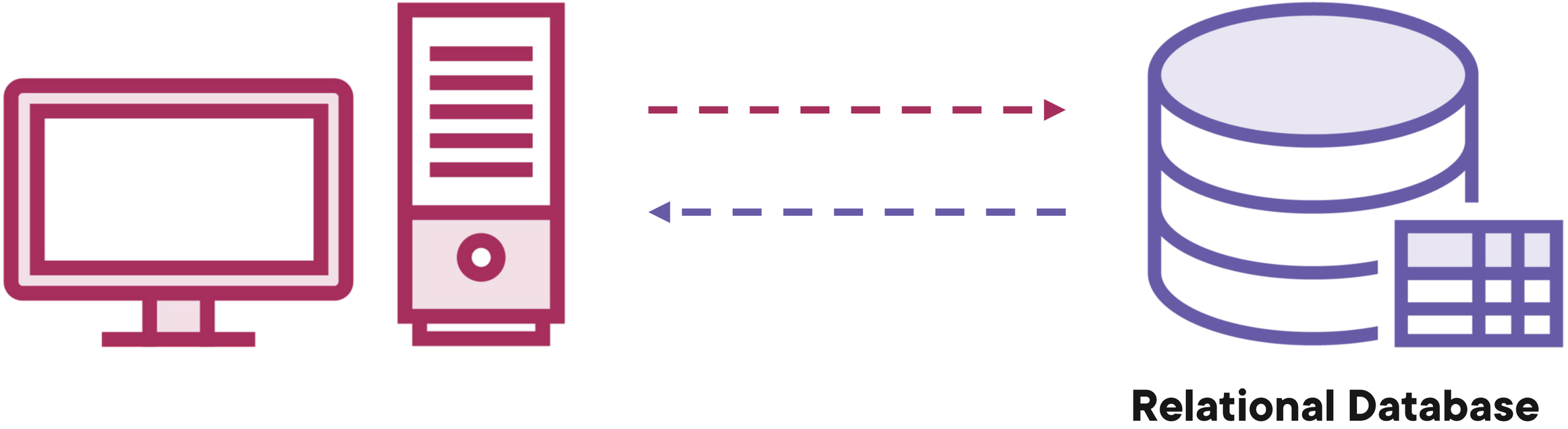


Graph database

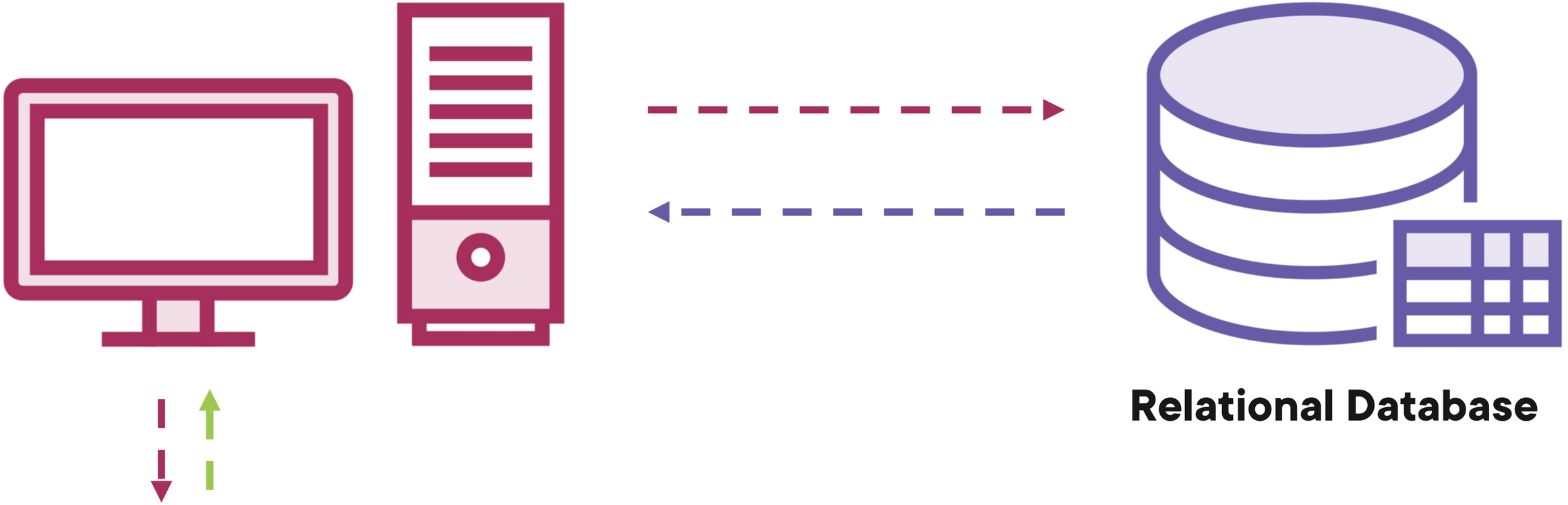
Not uncommon to use **more than one** type of **database**

Using More Than One Database

Using More Than One Database



Using More Than One Database



Use a **relational database** where you need to **enforce relationships**.

Good for both **simple** and **complex schemas**!

Warehouse Management System

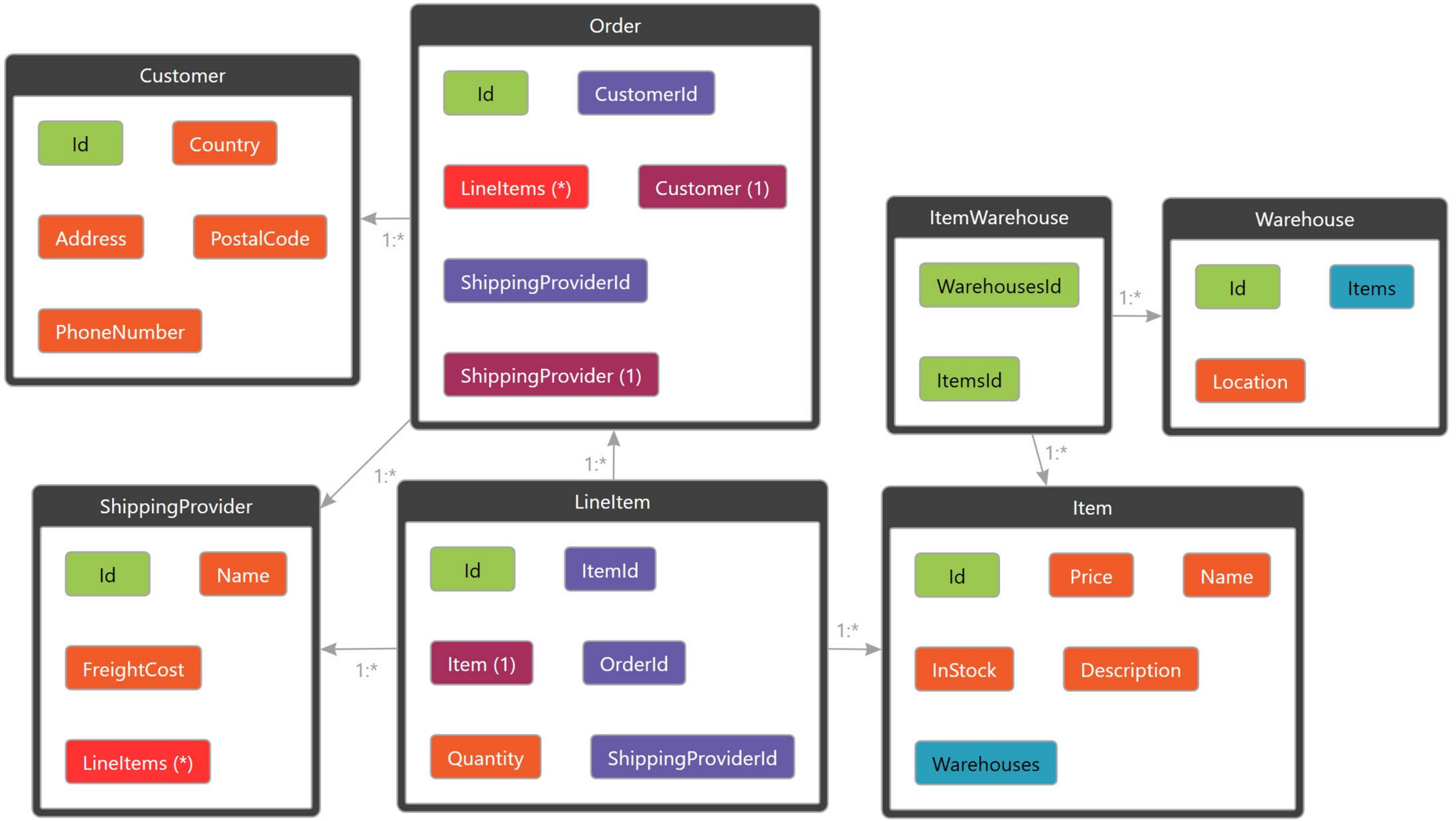


Diagram generated using EF Core Power Tools: marketplace.visualstudio.com/items?itemName=ErikEJ.EFCorePowerTools

NoSQL databases **provide** a **different way** of **persisting** data.

There are **many alternatives**, and they are all different!

Example: NoSQL Databases

Key-value store

Persistent dictionary

Document database

JSON serialized data

Example: Document Database

Receipt

Invoice

Purchase Order

Example: Document in a Document Database

```
{
  "InvoiceId": "IN123",
  "CustomerId": "1",
  "Due": "2025-01-01",
  "Items": [],
  "Total": 1999.50
}
```

Connection String

Where is the database?

IP Address / Resource

Filename

Credentials?

Username / Password

Key

Example: SQL Server Connection String

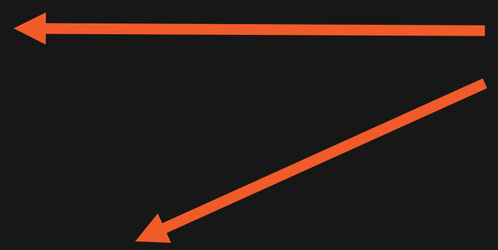
```
Server=server;Database=Warehouse;User Id=fekberg;Password=d&0Xtt4^9JKD;
```

Example: SQL Server Connection String

```
Server=server;Database=Warehouse;User Id=fekberg;Password=d&0Xtt4^9JKD;
```

```
Server=server;
Database=Warehouse;
Integrated Security=True;
```

```
Data Source=(LocalDB)\\MSSQLLocalDB;
Initial Catalog=Warehouse;
Integrated Security=True;
```



Address?

Database Name?

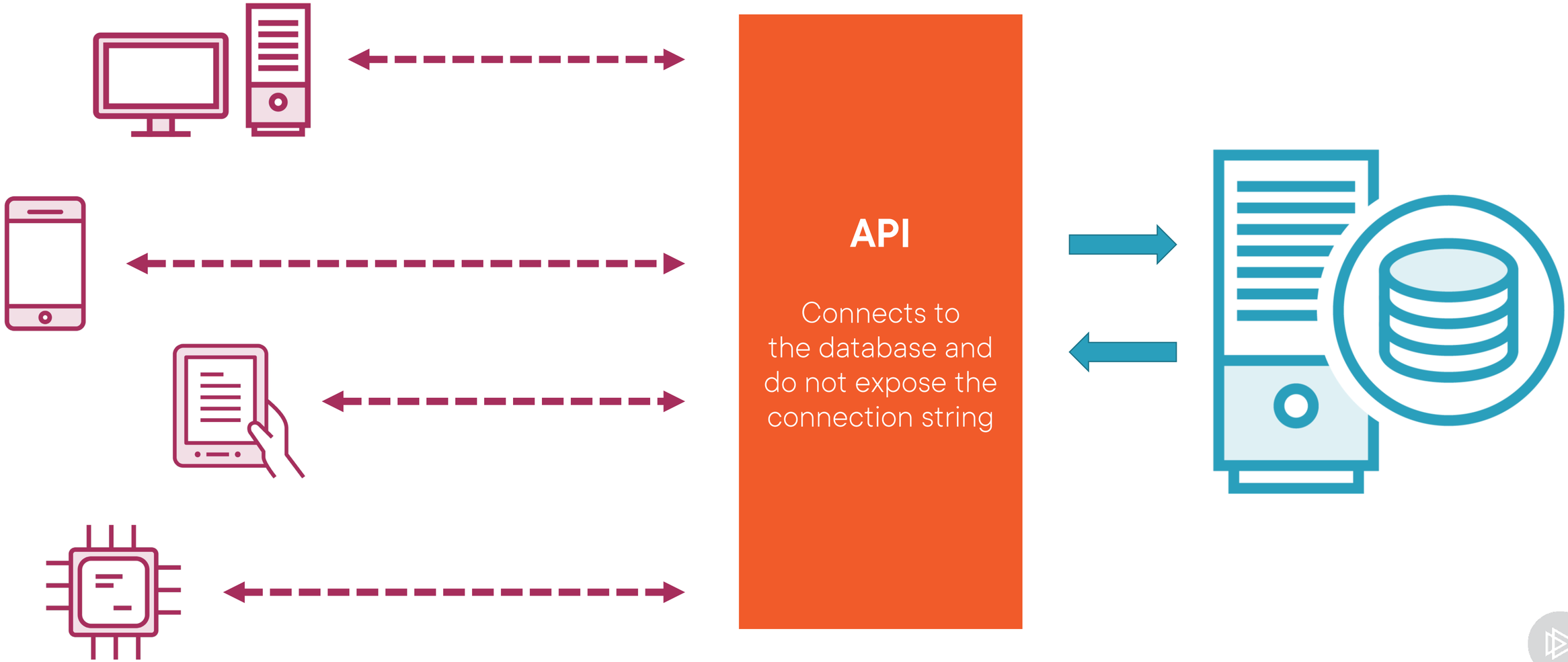
Windows Authentication?

Many different configurations!

Provider + **Connection string**

Enables you work with different databases

Connection String and Data Access in the API



LocalDB

Not meant for production

Quickly get access to a database engine

No configuration necessary!

SQLite

File based database

Cross-platform

Runs everywhere

Self-contained

Lightweight

SQLite can be used as a **local database** for the application to **cache data**

Connecting to **SQL Server** and **SQLite**

```
using SqlConnection connection
    = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;
                          AttachDbFilename=WarehouseManagement.mdf;
                          Integrated Security=True;");
```

Connecting to **SQL Server** and **SQLite**

```
using SqlConnection connection
    = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;
                          AttachDbFilename=WarehouseManagement.mdf;
                          Integrated Security=True;");

using SqliteConnection connection
    = new SqliteConnection(@"Data Source=warehouse.db");
```

Connecting to **SQL Server** and **SQLite**

```
using SqlConnection connection
    = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;
                          AttachDbFilename=WarehouseManagement.mdf;
                          Integrated Security=True;");

using SqliteConnection connection
    = new SqliteConnection(@"Data Source=warehouse.db");
```

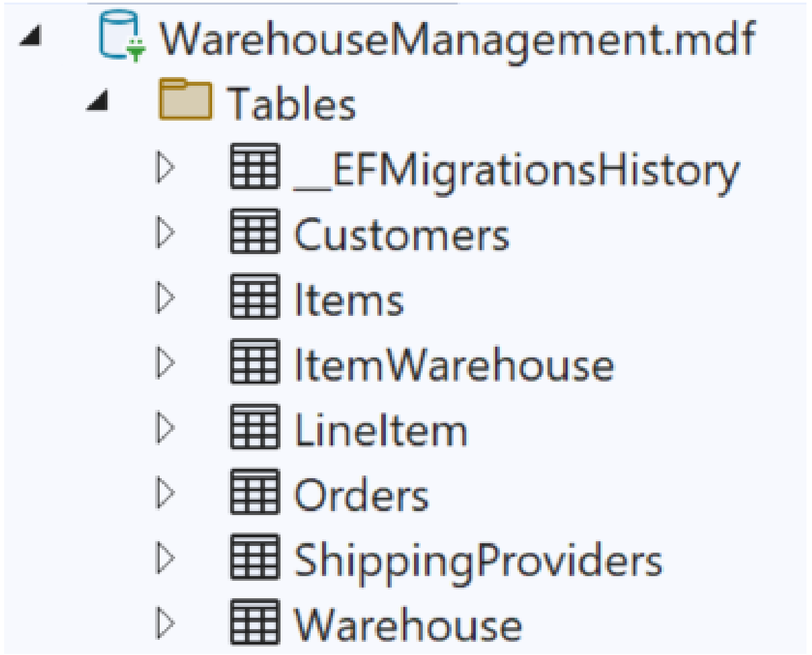
ADO.NET provides a
consistent API for working
with **different databases**

No need to write all the SQL!

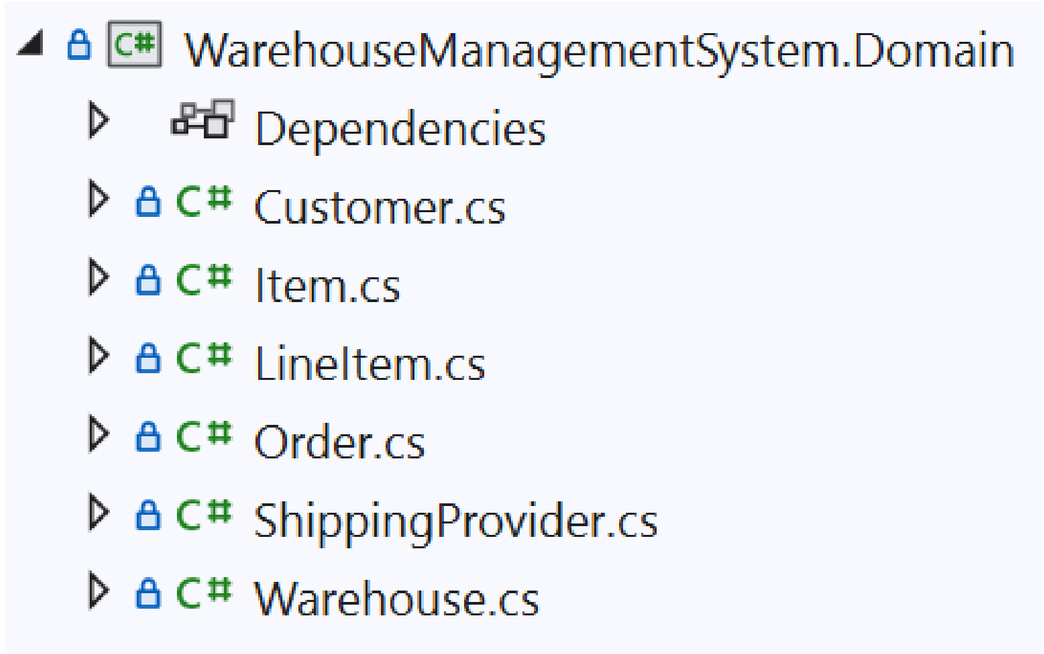
We can **introduce** an **abstraction** to help us

Object-Relational Mapping

Database schema

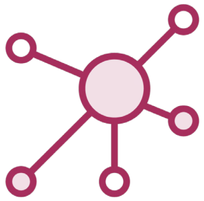


Models represented in C#



Example **ORMs**

NHibernate



Entity Framework Core

Commonly referred to as just "Entity Framework"

Dapper

LLBLGen Pro

Work with objects instead of having to write SQL

Entity Framework Core

“Entity Framework (EF) Core is a lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology.”

Built on-top of the data access services ADO.NET

docs.microsoft.com/en-us/ef/core

ADO.NET provides total control when needed in high performance situations

Consuming Data Using **LINQ**

```
var orders = context.
    Orders.Where(order => order.LineItems.Count > 3);
```

Consuming Data Using **LINQ**

```
var orders = context.
    Orders.Where(order => order.LineItems.Count > 3);
```

ADO.NET: Select All Orders

```
using var connection = new SqlConnection(@"Data Source=...");

using var command = new SqlCommand(
        "SELECT * FROM [Orders]",
        connection);

connection.Open();

using var reader = command.ExecuteReader();

while(reader.Read())
{
    Console.WriteLine(reader["Id"]);
}
```

Entity Framework Core: Select All Orders

```
using var context = new WarehouseContext();

foreach(var order in context.Orders)
{
    Console.WriteLine(order.Id);
}
```

Avoid Slow Queries!

```
context.Customers.Where(customer => customer.Name.Contains("Filip"));
```

Avoid Slow Queries!

```
context.Customers.Where(customer => customer.Name.Contains("Filip"));
```

WARNING!

Must search through all rows in the database!

Perform lookups on an index!

Benefits of Using an ORM

Do not necessarily need to know about the specific database

No need to learn the SQL specific syntax for the database