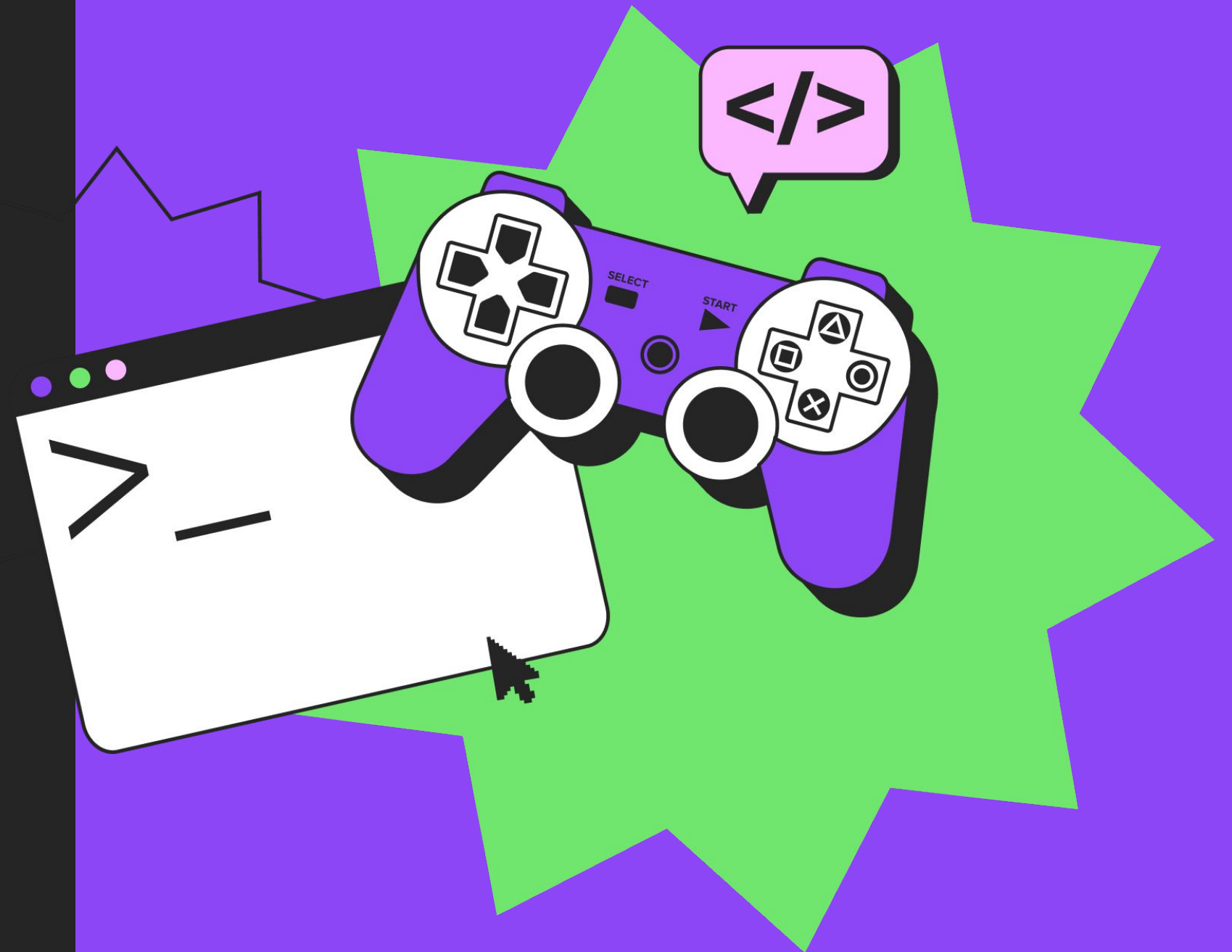


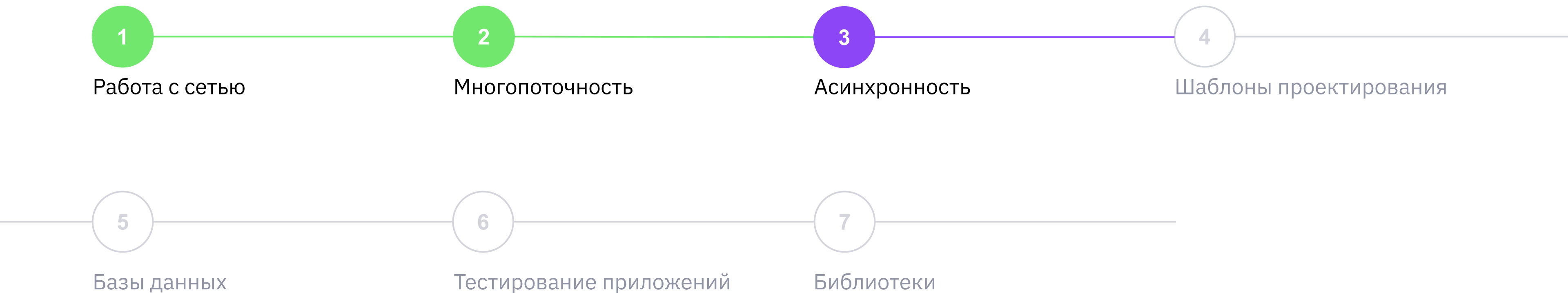
# Разработка сетевого приложения на C#

Урок 3  
Асинхронность





# План курса



# Содержание урока



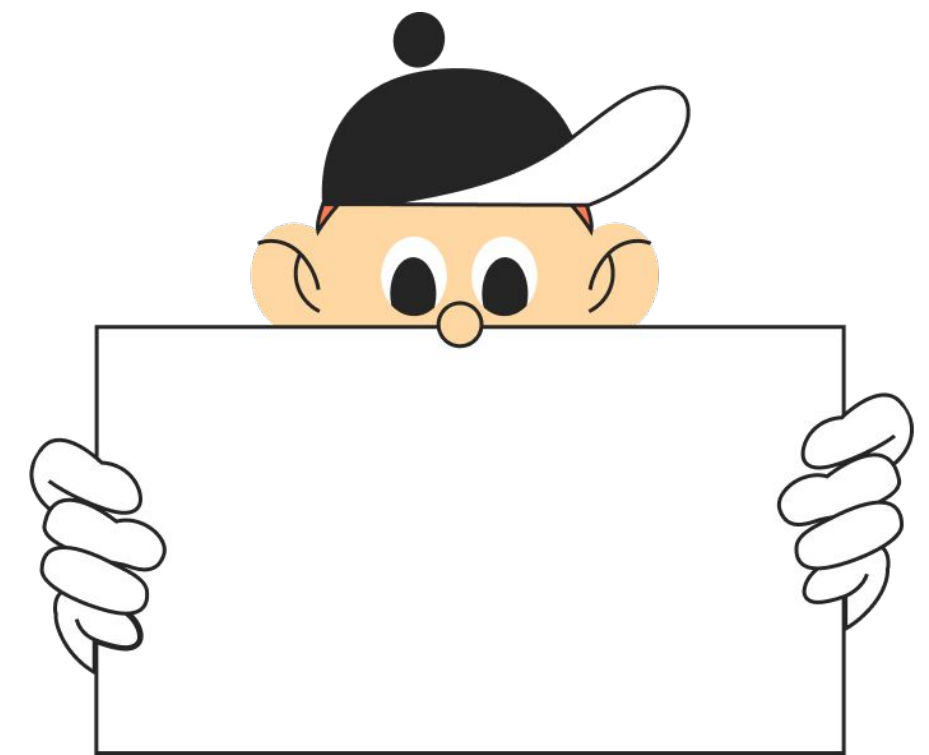
- TPL
- Task
- async/await
- ValueTask
- IAsyncDisposable
- ConfigureAwait
- Parallel
- ParallelLoopResult
- ParallelLoopState
- PLINQ
- IAsyncEnumerable



# TPL

Или же Task Parallel Library – библиотека параллельных **задач**, предназначенная для работы с асинхронными операциями.

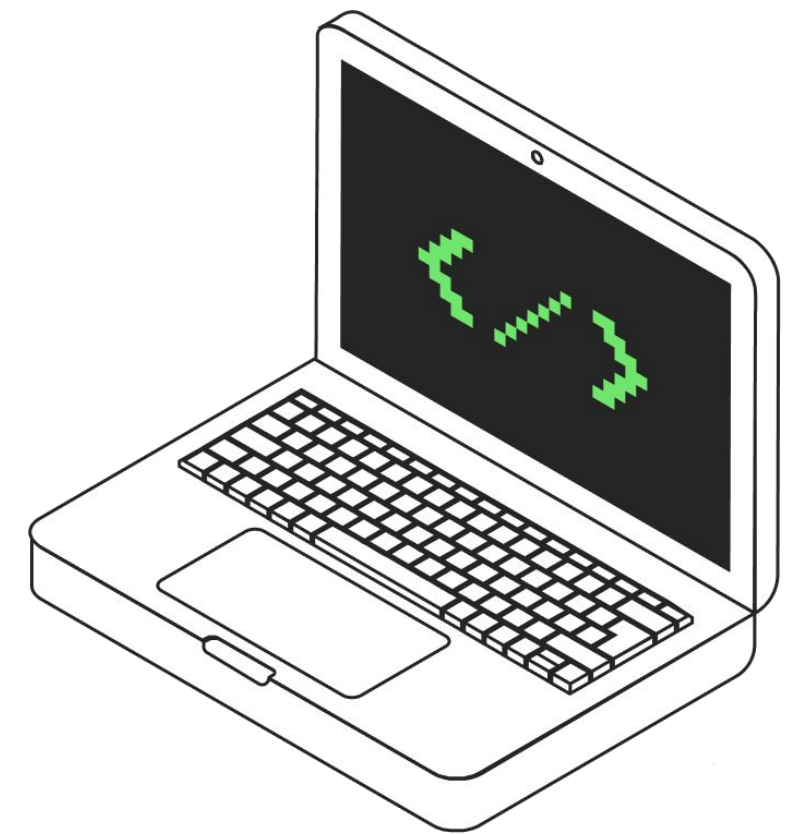
Чтобы точно все понять, давайте для начала вспомним, что такое параллелизм – это одновременное выполнение нескольких задач.





# Task

Библиотека TPL основана на `Task` — специальном классе, отвечающим за параллельное выполнение.





## async / await

Ранее в лекции мы уже применяли эти два ключевых слова при написании наших примеров. Теперь, когда мы разобрались с тем, что же такое Task, пришло время рассказать о них подробнее.

Начнем с ключевого слова `await`: оператор `await` ожидает выполнения задачи, блокируя поток, и возвращает результат ее выполнения.

В общем виде работа оператора выглядит следующим образом:

```
await someTask;
```

где `someTask` – это экземпляр выполняющейся задачи `Task`, или же:

```
TResult x = await someTask;
```

где `someTask` это экземпляр `Task<TResult>`.



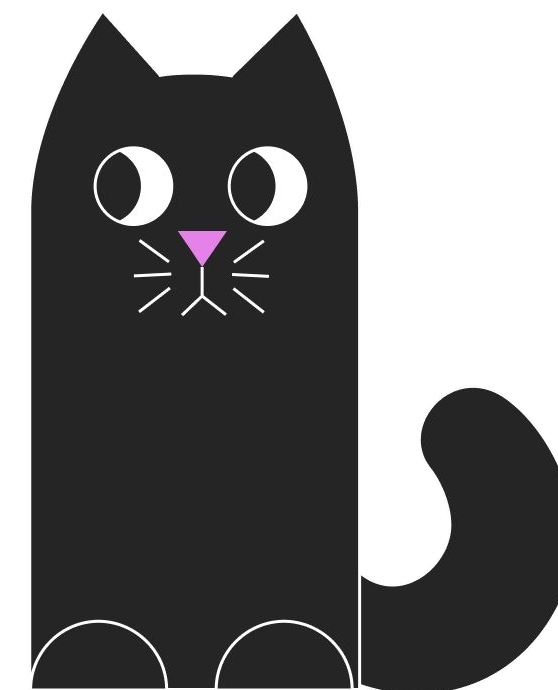


## ValueTask

Начиная с Net Core 2.0, альтернативой классу `Task` появился новый тип задач – `ValueTask`. По сути это всего лишь обертка над `Task`, у которой есть возможность ссылаться либо на незавершенную задачу `Task`, либо же на результат выполнения метода.



Как несложно догадаться по названию, это `value`-тип, и как следствие, он располагается в стеке, что дает прирост производительности при работе с ним. Если метод с `Task` был закончен синхронно, то есть результат был получен мгновенно, `ValueTask` копирует его в свое свойство `Result`.



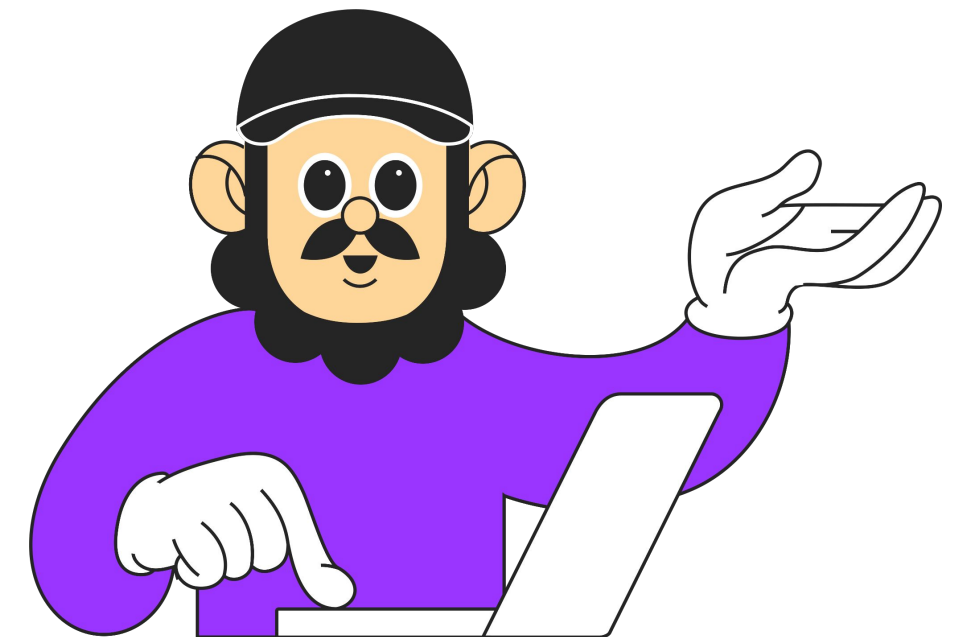


# IAsyncDisposable

Данный интерфейс предоставляет механизм для асинхронного освобождения ресурсов.



Можно рассматривать интерфейс как асинхронный аналог `IDisposable`.

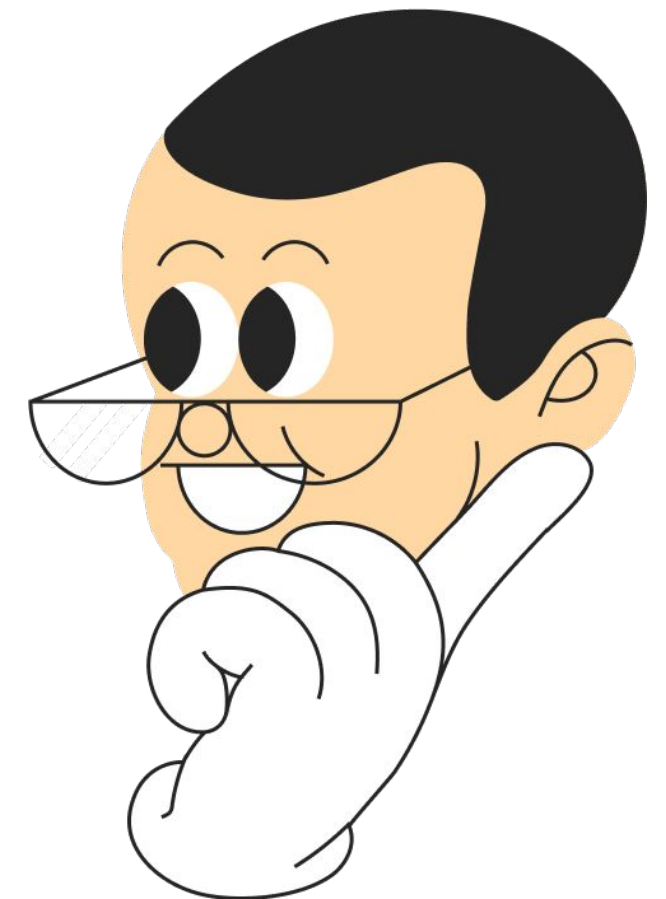






## Task.ConfigureAwait(bool)

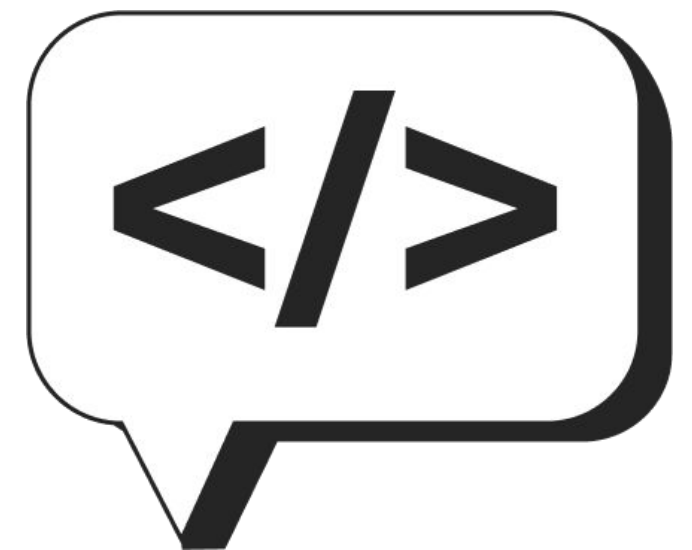
Это метод-расширение, позволяет создать задачу на основе текущей с переопределенным планировщиком задач таким образом, что продолжение выполнения метода будет работать в контексте ожидаемого потока или же в контексте метода, где был вызван `await`.





# Parallel

`Parallel` – статический класс, который предоставляет поддержку для параллельных циклов.





# ParallelLoopResult

Класс предназначен для хранения результатов работы цикла.





## ParallelLoopState

Объект, представляющий состояние цикла и позволяющий им управлять.





# PLINQ

Является частью `System.Linq.Parallel`. Основой PLINQ является класс `ParallelQuery`, предоставляющий инструменты для параллельного выполнения доступа к источникам данных.





# **IAsyncEnumerable**

Асинхронная версия эnumератора, позволяющая выполнять цикл `foreach` в асинхронном режиме.






# Подведение итогов

На этой лекции вы:

- Разобрались с тем, как работает асинхронность
- Узнали про работу Task
- Узнали про ключевые слова `async` и `await`
- Научились работать с асинхронными интерфейсами
- Узнали про PLINQ

**Спасибо**   
**за внимание**

