



ОНЛАЙН-ОБРАЗОВАНИЕ

# Онлайн-образование





# Меня хорошо видно && слышно?

Ставьте смайлик в чат, если все хорошо  
Напишите, если есть проблемы



# Правила вебинара



Активно участвуем и включаем камеры (+83%)



Задаем вопросы голосом или в чат



Off-topic обсуждаем в Slack



Вопросы в чате вижу, могу ответить не сразу

Проверить, идет ли запись!





The background of the slide is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer. A network of thin, light blue lines connects various points across the blue area, creating a digital or technological aesthetic. The title text is centered within this blue area.

# Асинхронные операции

---

Приходько Роман

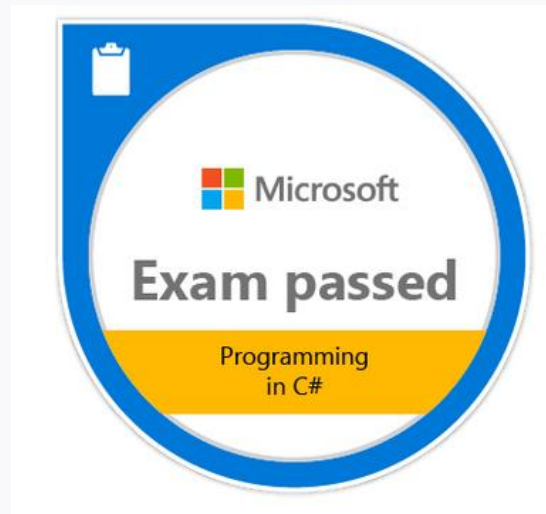


# Преподаватель



Приходько Роман

Старший .net разработчик SolarLab  
к.т.н., доцент СевГУ



# Цели вебинара

1

Понять, как пользоваться асинхронными методами

2

Писать асинхронный код

# План занятия

- Общие понятия
- Создание и выполнение тасок
- Цепочки асинхронных операций
- Обработка исключений
- Отмена асинхронных операций





# Общие понятия



# Интуитивное понятие асинхронности



Медленная внешняя система,  
которую невозможно ускорить

Синхронный подход:

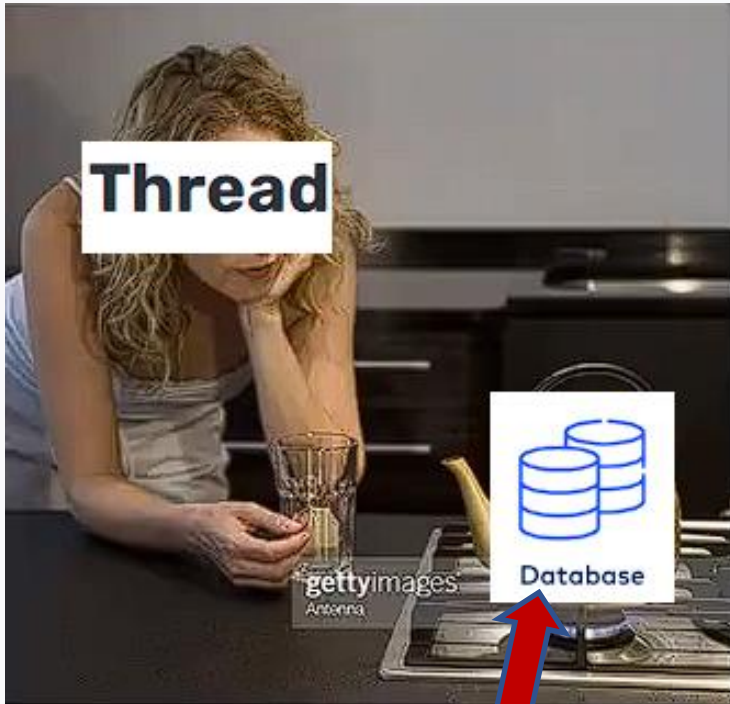
1. Поставить чайник
2. Дождаться закипания, выпить чай, начать уборку

Асинхронный подход:

1. Поставить чайник
2. Начать уборку
3. Когда закипит, выпить чай и продолжить уборку



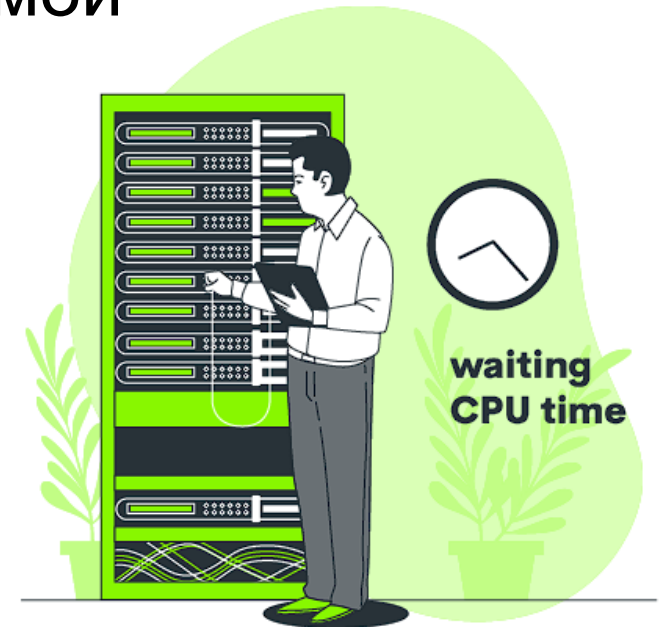
# Интуитивное понятие асинхронности



Медленная внешняя система,  
которую невозможно ускорить

## Действия с внешними системами

- Запросы в базу данных
- Работа с файловой системой
- Сетевые запросы



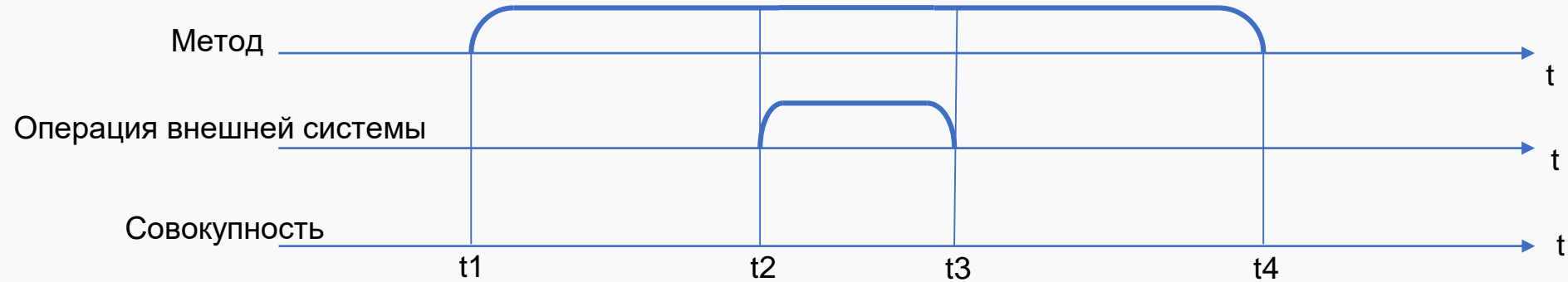
# Синхронность и асинхронность в виде временной диаграммы



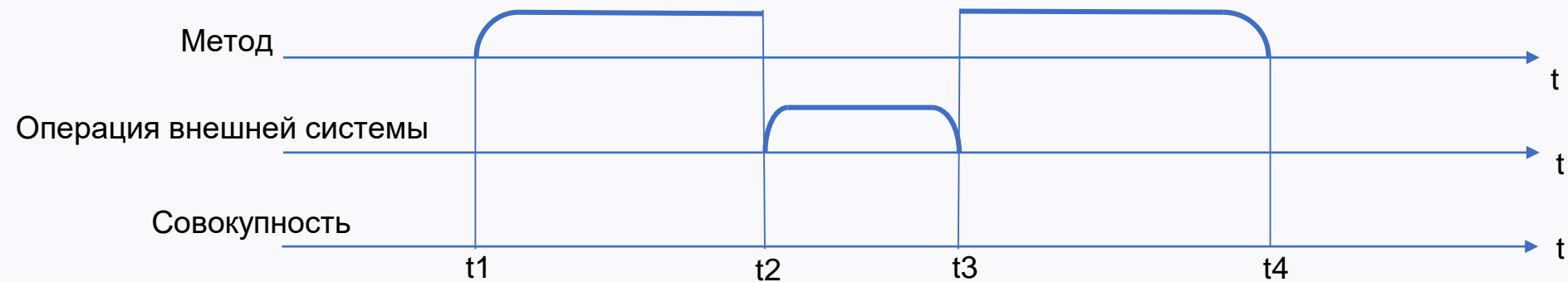


# Синхронность и асинхронность в виде временной диаграммы

**Синхронное выполнение**



**Асинхронное выполнение**



# Терминология



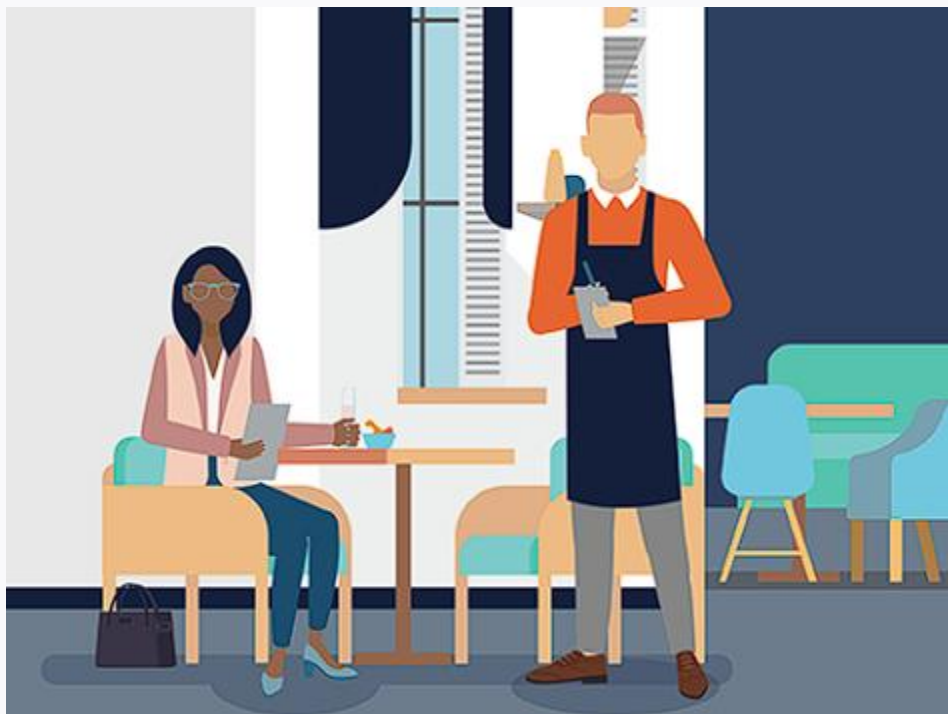
Терминология: “поток простаивает”, “поток заблокирован”, “пул потоков истощается”



# Асинхронность

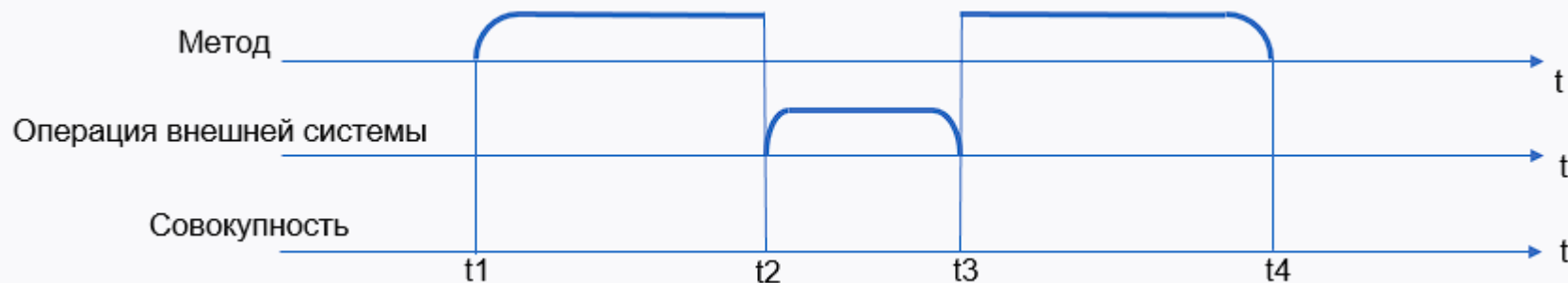
Асинхронность — выполнение программного кода, не блокирующее потоки во время ожидания

# Асинхронность vs многопоточность



Многопоточность – о увеличении ресурсов

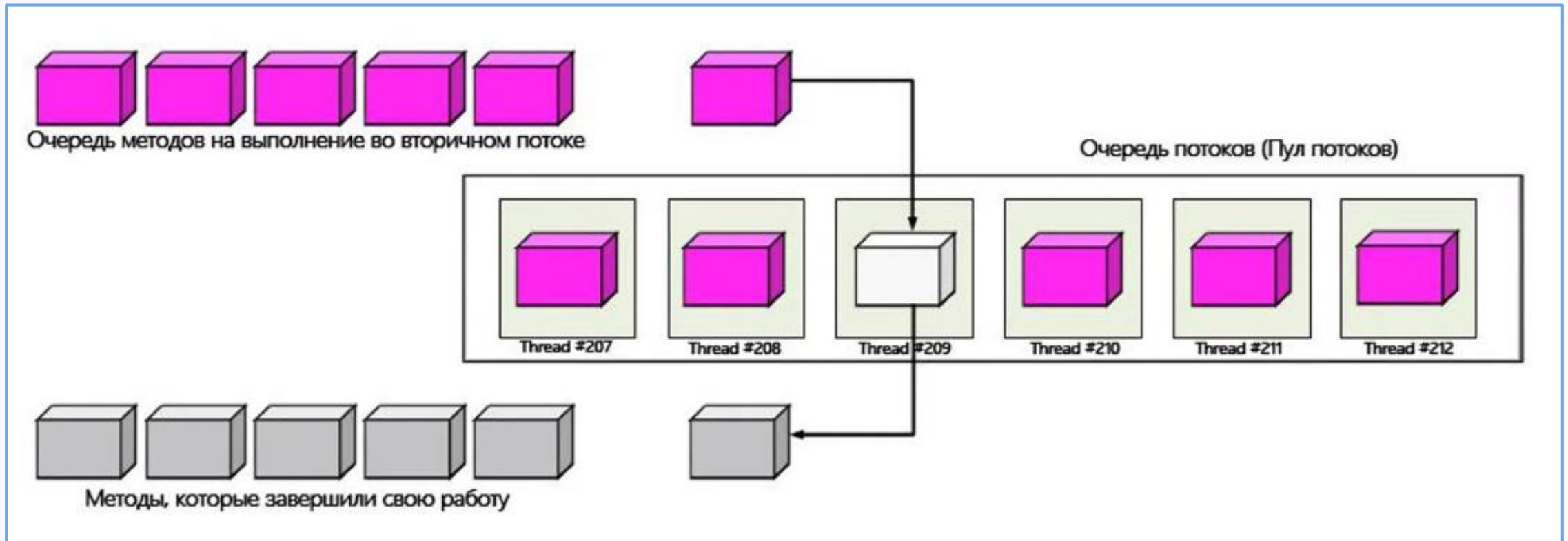
Асинхронность - о более рациональном использовании ресурсов





# Пул потоков

**Пул потоков (Thread Pool)** – набор уже созданных потоков, готовых к выполнению задач



# История развития реализации асинхронности

Реализация одного и того же метода с помощью APM, EAP и TAP

APM (.Net 1.1)

```
C# Copy
public class MyClass
{
    public IAsyncResult BeginRead(
        byte [] buffer, int offset, int count,
        AsyncCallback callback, object state);
    public int EndRead(IAsyncResult asyncResult);
}
```

EAP (.Net 2)

```
C# Copy
public class MyClass
{
    public void ReadAsync(byte [] buffer, int offset, int count);
    public event ReadCompletedEventHandler ReadCompleted;
}
```

TAP (.Net 4.5)

async-await (C# 5.0)

```
C# Copy
public class MyClass
{
    public Task<int> ReadAsync(byte [] buffer, int offset, int count);
}
```



# TAP (Task-based Asynchronous Pattern)

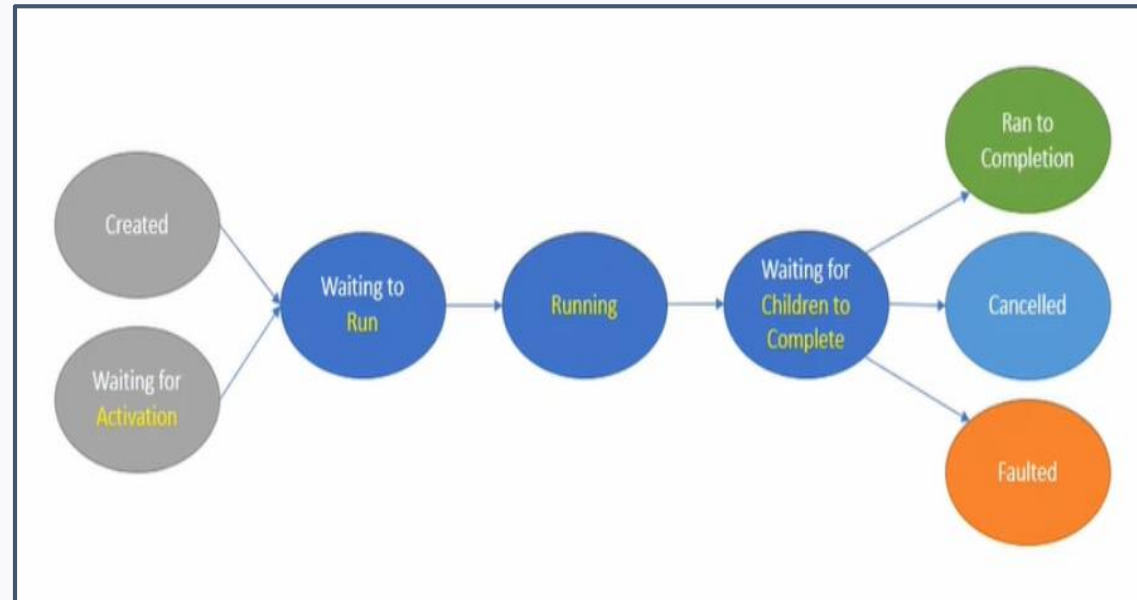
```
public async Task<int> ExecuteAsync()  
{  
    Console.WriteLine("hello");  
}
```

Асинхронные операции в большинстве случаев возвращают Task<T>

## System.Threading.Tasks.Task

Экземпляр задачи хранит в себе:

- Метод, который нужно выполнить
- Статус
- Исключения
- ..



# Виды задач



Task

Delegate Task

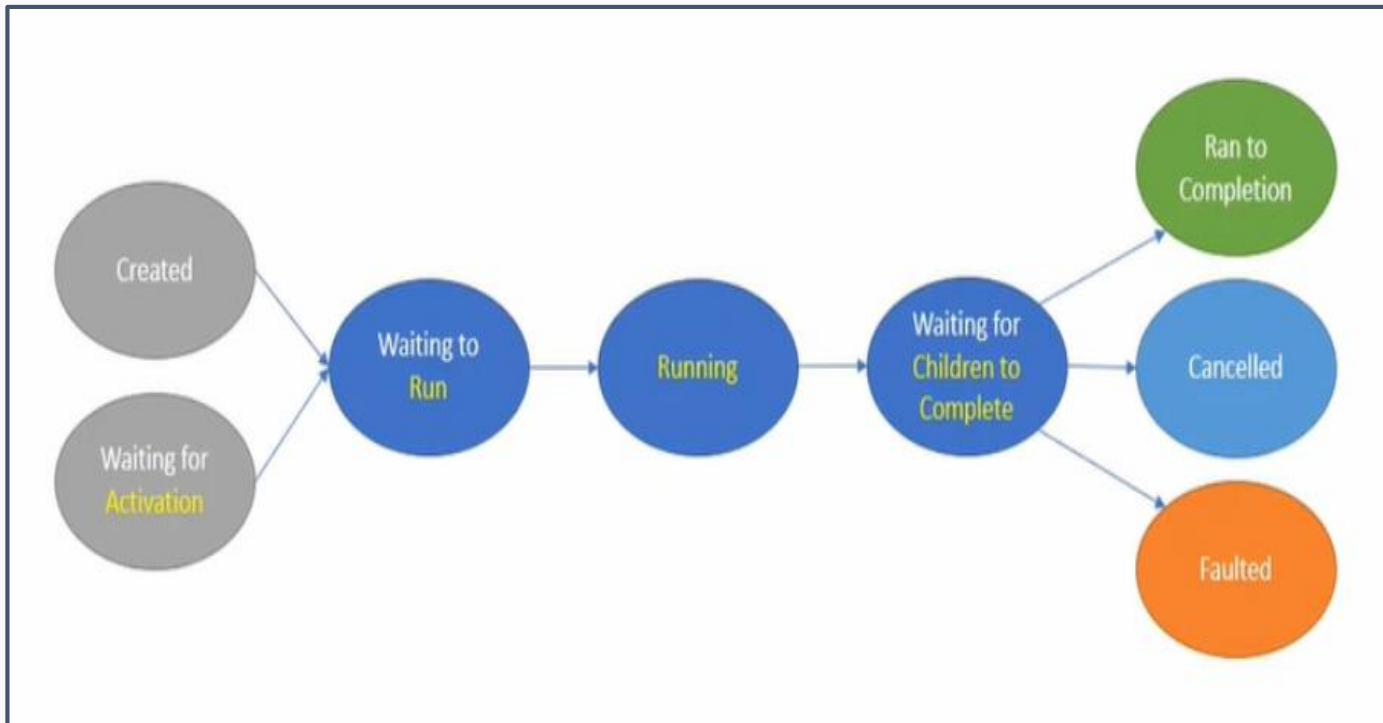
Promise Task



# Delegate Task

Задача, содержащая код, который нужно выполнить

```
Task t = new Task( () => { foreach(var path in Directory.GetFiles(dirName))  
                             list.Add(path); } );
```



# Promise Task

Задача, не содержащая в себе никакого кода, который нужно выполнить

`Task.FromResult(true)`

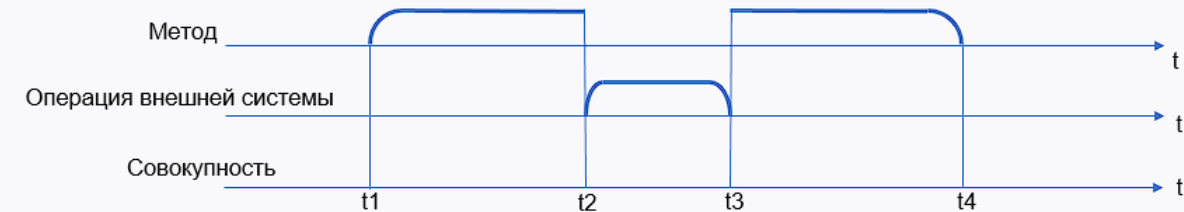
`Task.CompletedTask`.





# async - await

```
public async Task ExecuteAsync()  
{  
    Console.WriteLine("Start");  
    await InternalAsync();  
    Console.WriteLine("Finish");  
}
```



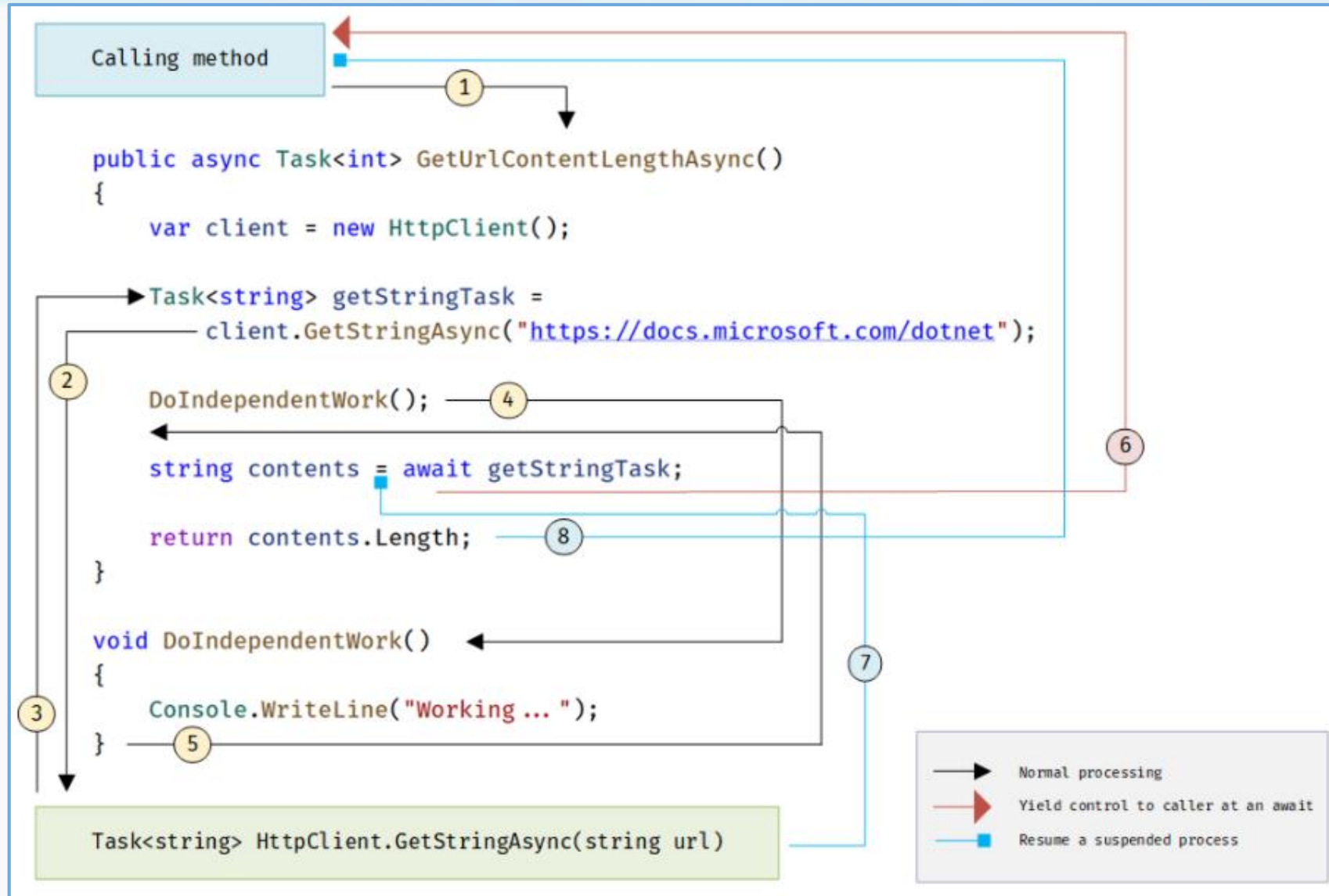
## Async

1. Превращает метод в асинхронную операцию
2. Накладывает ограничения на сигнатуру и возвращаемое значение
3. Позволяет применить await

## Await

1. приостанавливает выполнение метода и возвращает управление вызывающему коду до завершения асинхронной операции, идущей после него
2. Извлекает результат метода асинхронной операции и исключения если они есть

# Что происходит в методе Async





# Асинхронная мантра

```
private static async Task<Toast> ToastBreadAsync(int slices)
{
    for (var slice = 0; slice < slices; slice++)
    {
        Console.WriteLine("Помещаем хлеб в тостер");
    }
    Console.WriteLine("Включаем тостер...");
    await Task.Delay(5000);
    Console.WriteLine("Вынимаем тосты из тостера");

    return new Toast();
}
```

Асинхронная мантра: «**async Task await Async**»

# Асинхронные методы

**Асинхронные методы** — методы использующие ключевые слова **async/await** и имеют специальный **тип** возвращаемого значения. При наименовании метода в конец добавляется суффикс **Async**.


```
public async Task PrintMeAsync()
{
    await Task.Run(() => Console.WriteLine("Printing"));
}
```

```
public async Task<int> MultiplyMeAsync(int a, int b)
{
    return await Task.Run(function: () => a * b);
}
```

```
public async void KillMeAsync()
{
    await Task.Run(() => Console.WriteLine("Nooooo"));
}
```

Асинхронный метод, как и обычный, может использовать любое количество параметров или не использовать их вообще. Однако асинхронный метод **не может** определять параметры с модификаторами **out** и **ref**.





# Цепочки асинхронных операций



# Цепочки вызовов асинхронных методов

Асинхронный метод чаще всего вызывается из асинхронного метода.

Вызывающие друг друга асинхронные методы формируют цепочки.

Важно понимать, где цепочка вызова начинается и где ее конечный вызов

# Синхронное ожидание асинхронной операции

`Task.Wait()` (`Task.Result`) (исключения передаются в составе `AggregateException`)

`GetAwaiter.GetResult()` (исключения передаются в исходном виде)

Недостатки:

1. Истощение ресурсов пула потоков
2. Возможность дедлока

# Что если делать await только в конце асинхронной операции

1. Async-await распространяются по коду
2. Можно делать цепочки без async – await, но не рекомендуется

<https://blog.stephencleary.com/2016/12/eliding-async-await.html>







# Запуск и выполнение тасок



# Типы возвращаемых значений

- **Task** — для асинхронного метода, не возвращающего значение
- **Task<TResult>** — для асинхронного метода, возвращающего значение
- **void** — для обработчика событий (event handler)
- **IAsyncEnumerable<T>\*** — для асинхронного метода, который возвращает асинхронный поток.

\* - для c# версии 8.0 и выше

# Создание и запуск тасок

**Task.Run**

**Task.Factory.StartNew**

**(new Task()).Start()**

**TaskCreationOptions**

**TaskScheduler**



<https://blog.stephencleary.com/2013/08/startnew-is-dangerous.html>



Стивен Туб - MSFT

<https://devblogs.microsoft.com/pfxteam/task-factory-startnew-vs-new-task-start>



# Дочерние задачи

## Виды:

- Прикрепленные
- Открепленные

Category	Detached child tasks	Attached child tasks
Parent waits for child tasks to complete.	No	Yes
Parent propagates exceptions thrown by child tasks.	No	Yes
Status of parent depends on status of child.	No	Yes

Стивен Клири:

“TAP обычно не использует AttachedToParent. AttachedToParent была частью TPL. И TPL, и TAP имеют один и тот же тип Task, но есть много членов TPL, которых следует избегать в коде TAP. В TAP лучше рассматривать понятия **родительских и дочерних асинхронных методов.**”

# Запуск нескольких Task одновременно

Task.WhenAll  
Task.WhenAny





# Обработка исключений



# Обработка ошибок в асинхронных методах

Исключения хранятся в таске, в виде `AggregateException`

# Обработка ошибок в асинхронных методах

Для обработки ошибок выражение **await** помещается в блок **try**


```
try
{
    await DoSomethingAsync();
}
catch (Exception e)
{
    Console.WriteLine(e);
}
```

```
Task task = null;
try
{
    task = DoSomethingAsync();
    await task;
}
catch (Exception e)
{
    Console.WriteLine("Exception: " + e.Message);
    Console.WriteLine("IsFaulted: " + task.IsFaulted);
}
```

```
Task allTasks = null;
try
{
    var task1 = DoSomethingAsync();
    var task2 = DoSomethingAsync();
    var task3 = DoSomethingAsync();

    allTasks = Task.WhenAll(task1, task2, task3);
    await allTasks;
}
catch (Exception e)
{
    Console.WriteLine("Exception: " + e.Message);
    Console.WriteLine("IsFaulted: " + allTasks.IsFaulted);
    foreach (var inx:Exception in allTasks.Exception.InnerExceptions)
    {
        Console.WriteLine("internal exception: " + inx.Message);
    }
}
```





# Отмена асинхронных операций



# Отмена асинхронных операций

Отмена по согласию: нужно не только запросить отмену в вызывающем коде, но и реализовать ее в вызываемом

Виды отмены:

- По событию
- По шедулеру

CancellationToken  
CancellationTokenSource.

# Отмена асинхронных операций

TaskCancelledException  
OperationCancelledException

Для CancellationTokenSource нужно вызывать Dispose

Если метод не поддерживает отмену

```
public static async Task Timeout(this Task operation, TimeSpan timeout)
{
    using (var cts = new CancellationTokenSource())
    {
        var timeoutTask = Task.Delay(timeout, cts.Token);
        var any = await Task.WhenAny(new[] { timeoutTask, operation }).ConfigureAwait(false);
        if (any == timeoutTask)
            throw new TimeoutException();
        else
            cts.Cancel();
    }
}
```

WaitAsync

# Best Practices

## Не делай

- Никогда **не** используй **void**, если это не обработчик событий(event handler)
- Никогда **не** блокируй **асинхронные операции** в **асинхронном** коде вызовом методов **GetResult()** или **Wait()**

## Делай

- Всегда используй **async** и **await** вместе
- Всегда возвращай **Task** из асинхронных методов
- Всегда используй **await** для асинхронных методов

**async Task await Async**

(Асинхронный таск ожидает суффикс Async)




# Вопросы для самопроверки

- Что такое асинхронность?
- Как создать асинхронный метод?
- Какие типы может возвращать асинхронный метод?
- Где применяется асинхронность?

# Список материалов для изучения

- <https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/>
- <https://github.com/davidfowl/AspNetCoreDiagnosticScenarios/blob/master/AsyncGuidance.md>
- <https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/async/task-asynchronous-programming-model>
- <https://devblogs.microsoft.com/pfxteam/executioncontext-vs-synchronizationcontext/>
- <https://habr.com/ru/post/416751/>
- <https://docs.microsoft.com/en-us/dotnet/api/system.threading.asynclocal-1?view=netcore-3.1>



The background of the image is an aerial photograph of a city with many skyscrapers, overlaid with a semi-transparent blue layer. A network of white lines connects various points across the blue area, creating a digital or technological aesthetic. The text is centered in this blue area.

Заполните, пожалуйста,  
опрос о занятии по ссылке в чате



The background of the slide is an aerial view of a city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue layer that features a white network pattern of dots and lines. The title text is centered within this blue band.

# ОТВЕТЫ НА ВОПРОСЫ

