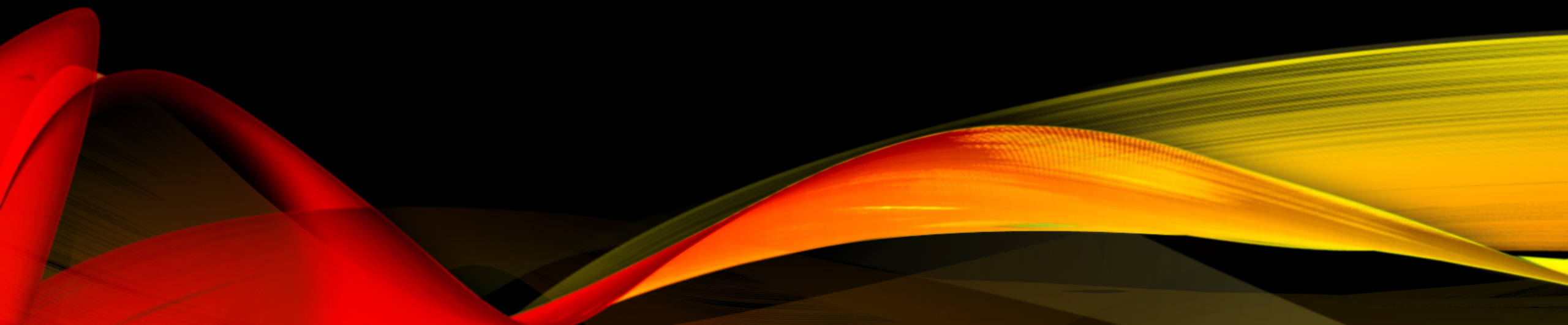
ЛЕКЦИЯ 18

- Модуль 3
- 15.03.2023
- Асинхронное программирование

ЦЕЛИ ЛЕКЦИИ

- Познакомиться с общими принципами параллельного программирования .NET
- Разобрать принципы асинхронного программирования .NET

АСИНХРОННОЕ ПРОГРАММИРОВАНИЕ



АСИНХРОННОСТЬ

- **Асинхронное программирование** - это параллельная модель программирования, в которой допускается выполнять большое количество задач единовременно в небольшом количестве потоков и при поддержке синтаксиса обычного синхронного кода

ПАТТЕРНЫ АСИНХРОННОГО ПРОГРАММИРОВАНИЯ

Основан на типах Task и Task<TResult>, необходимые для представления асинхронных операций

- Асинхронный шаблон на основе задач (TAP)
- Асинхронная модель на основе событий (EAP)
- Модель асинхронного программирования (APM)

Устаревшие модели

TPL

- Библиотека параллельных задач (Task Parallel Library, TPL) – набор открытых типов и API-интерфейсов из пространств имён System.Threading и System.Threading.Task
  - https://learn.microsoft.com/ru-ru/dotnet/standard/parallel-programming/task-parallel-library-tpl?source=recommendations

- Многопоточный код усложняет выполнение программы
- Не всякий под подходит для параллельных вычислений (легко можно добиться потери производительности)

РЕКОМЕНДАЦИИ К АСИНХРОННОЙ МОДЕЛИ

- **Ограничения производительности ввода-вывода**, рекомендовано использовать await для операции, которая возвращает Task или Task<T>, внутри метода async
  - Код ожидает чего-либо, например, данных из источника
- **Ограничение ресурсов процессора**, рекомендовано использоваться await для операции, которая запускается в фоновом потоке методом Task.Run
  - Код выполняет сложные вычисления, например, перерасчёт комплексного состояния

TASK И TASK<T>

Задача (Task) — это конструкция, реализующая модель асинхронной обработки на основе обещаний (Promise)

- Модель "обещает", что задача будет выполнена позже, позволяя взаимодействовать с помощью "обещаний" с чистым API
- Класс Task представляет одну задачу, которая не возвращает значение
- Класс Task<T> представляет одну задачу, которая возвращает значение типа T

Поддержка Task и Task<T> начиная с C# 5.0 (2012) реализована через async + await

TASK И TASK<T>

- Важно рассматривать задачи (Task), как абстракции асинхронных операций, а не как абстракции поверх потоков
- По умолчанию задачи (Task) выполняются в текущем потоке и при необходимости делегируют работу операционной системе
- Для задач может также явно запрашиваться запуск в отдельном потоке через метод Task.Run()

ПАТТЕРН WAIT-UNTIL-DONE ЧЕРЕЗ TASK<T>

```
// объявление метода для асинхронного вызова
static long Sum(int x, int y) {
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}
```

```
public static void Main01Wait_Until_Done() {
    Task<long> task = new Task<long>(() => Sum(3, 5));
    Console.WriteLine("Before Start");
    task.Start();
    Console.WriteLine("After Start");
    Console.WriteLine("Doing stuff");
    long result = task.Result; // Wait for end and get result
    Console.WriteLine("After task.Result: {0}", result);
}
```

Результат:

Before Start
After Start
Doing stuff
		Inside Sum
After task.Result: 8

ПАТТЕРН POLLING ЧЕРЕЗ TASK<T>

```
static long Sum(int x, int y) {
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}
```

```
public static void Main02Polling() {
    Task<long> task = new Task<long>(() => Sum(3, 5));
    task.Start();
    Console.WriteLine("After Start");
    // Check whether the async method is done.
    while (!task.IsCompleted)  {
        Console.WriteLine("Not Done");
        // Continue processing, even though in this case it's just busywork.
        for (long i = 0; i < 10000000; i++) ; // Empty statement
    }
    Console.WriteLine("Done");
    long result = task.Result; // Call EndInvoke to get result and clean up.
    Console.WriteLine("Result: {0}", result);
}
```

Результаты выполнения программы:

```
After Start
Not Done
            Inside Sum
Not Done
Not Done
Not Done
Done
Result: 8
```

ПАТТЕРН CALLBACK ЧЕРЕЗ TASK<T>

```
static long Sum(int x, int y) {
    Console.WriteLine("\t\tInside Sum");
    Thread.Sleep(100);
    return x + y;
}

static void CallWhenDone(long result) {
    Console.WriteLine("\t\tInside CallWhenDone.");
    Console.WriteLine("\t\tThe result is: {0}.", result);
}

public static void Main03Callback() {
    Task<long> task = new Task<long>(() => Sum(3, 5));
    task.ContinueWith(t => CallWhenDone(t.Result));
    task.Start();
    Console.WriteLine("After Start");
    Console.WriteLine("task.Result: {0}", task.Result);
    // Console.ReadKey(true);  // НЕобязательно
}
```

TaskContinuationOptions.AttachedToParent

ТАЙМЕРЫ

System.Timers.Timer **System.Windows.Forms.Timer**

System.Threading.Timer
Конструктор (один из пяти):

```
public Timer (System.Threading.TimerCallback callback, object state, uint dueTime, uint period);
```

Прототип метода обратного вызова:

```
void TimerCallback( object state )
```

```
Timer myTimer=new Timer(MyCallback, someObj, 2000,1000);
```

имя метода обр. вызова

первый вызов через 2000 мс

объект для передачи в метод обр. вызова

вызов каждые 1000 мс

ПРИМЕР С ASYNC + AWAIT (1)

```
static long Factorial(int factor) {
    long res = 1;
    for (int k = 1; k <= factor; k++)
        res *= k;
    System.Threading.Thread.Sleep(1500);
    return res;
}
```

```
// запуск вычисления факториала в отдельном потоке
static async Task<long> FactorialAsync(int factor) {
    Console.WriteLine("2 - Запуск асинхронного потока!");
    var eee = await Task.Run(() => Factorial(factor));
    Console.WriteLine("4 - Завершен асинхронный поток!");
    return eee; // long
}
```

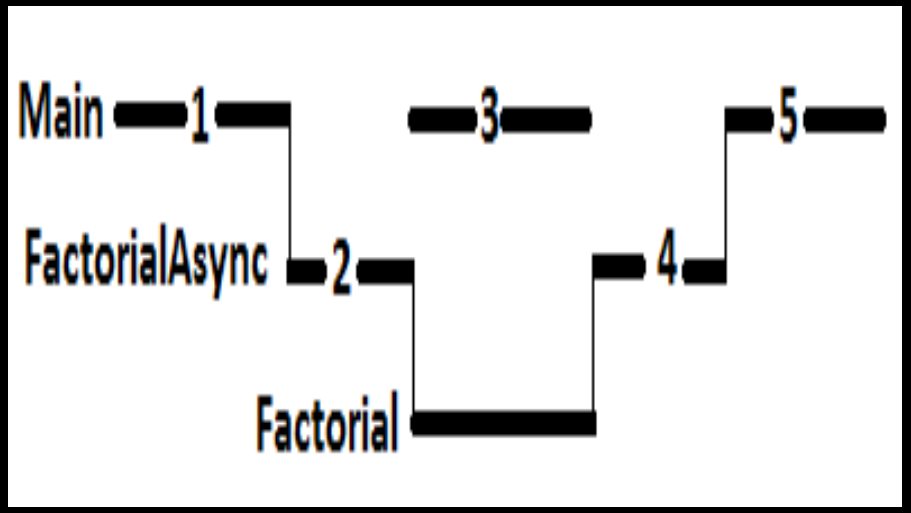
ПРИМЕР С ASYNC + AWAIT (2)

```
public static void Main()
{
    Console.WriteLine("1 - Выполнение основного потока!");
    var result = FactorialAsync(5);
    Console.WriteLine("3 - Продолжение основного потока!");
    Console.WriteLine("5 - В основном потоке: 5! = " +
        result.Result.ToString());
}
```

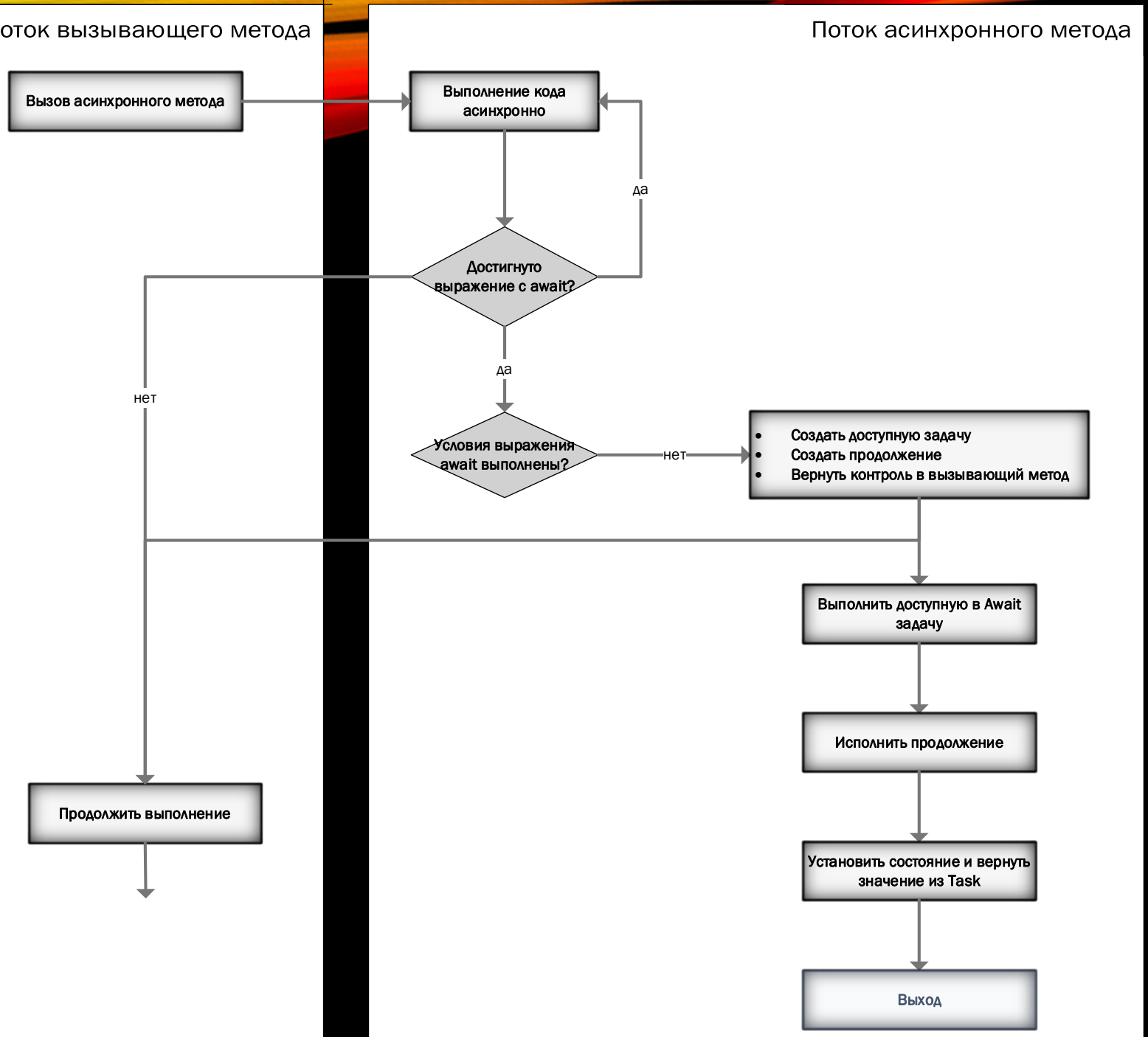
Результаты выполнения программы:

1 - Выполнение основного потока!
2 - Запуск асинхронного потока!
3 - Продолжение основного потока!
4 - Завершен асинхронный поток!
5 - В основном потоке: 5! = 120

Схема двух потоков:



ПРИНЦИП РАБОТЫ ASYNC + AWAIT



ПРИМЕР С ASYNC + AWAIT (3)

```
// Синхронный метод выполняет суммирование части массива:
static long SumAr(int[] ar, int beg, int end) {
    long res = 0;
    for (int k = beg; k < end; k++)
    res += ar[k];
    System.Threading.Thread.Sleep(1000);
    Console.Write("Beg = {0}  ", beg);
    Thread at = Thread.CurrentThread;
    Console.WriteLine("HashCode = " + at.GetHashCode());
    return res;
}
```

```
// Асинхронный метод
static async Task<long> SumAsync(int[] ar, int beg, int end) {
    var eee = await Task.Run(() => SumAr(ar, beg, end));
    return eee;  // long
}
```

ASYNC + AWAIT (4)

```
static void Main02() {
    int N = 8000;   // размер вектора
    int[] vec = new int[N];    // вектор
    for (int j = 0; j < vec.Length; j++)
        vec[j] = 1;

    long sum = 0;
    int P = 4;  // Число потоков
    object[] part = new object[P];  // Результаты потоков
    for (int j = 0; j < P; j++)
        part[j] = SumAsync(vec, j * N / P, (j + 1) * N / P);

    for (int j = 0; j < P; j++)
        sum += ((Task<long>)part[j]).Result;
    Console.WriteLine("Итоговая сумма = " + sum);
}
```

PARALLEL LOOPS

Пространство имен: **System.Threading.Tasks**
Parallel.For loop **Parallel.ForEach loop**

Прототип метода:
```
void Parallel.For(int fromInclusive, int toExclusive, Action body );
```

Параметры Parallel.For:
int *fromInclusive* – начальное значение параметра (индекса)

int *toExclusive* – верхняя граница параметра (индекса) серии итераций

Action *body* – **библиотечный** делегат, представляющий исполняемый на каждой итерации код: **public delegate void** Action()

ПРИМЕР С МЕТОДОМ PARALLEL.FOR

```
public static void MainFor()
{
        Parallel.For(0, 15, i =>
           Console.WriteLine("The square of {0} is {1}", i, i * i)
        );
}
```

Результаты выполнения:

```
The square of 0 is 0
 The square of 7 is 49
 The square of 8 is 64
 The square of 9 is 81
……….
```


https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.parallel.for?view=net-6.0

PARALLEL.FOREACH

```
static  ParallelLoopResult  ForEach<TSource>(IEnumerable<TSource> source,
       Action<TSource>  body)
```

Параметры Parallel.ForEach:
`TSource` – тип элементов - объектов в коллекции

`source` – коллекция объектов типа TSource

`body` – лямбда-выражение
Библиотечный обобщенный делегат: `public delegate void Action<T>(T obj)`

https://docs.microsoft.com/ru-ru/dotnet/api/system.threading.tasks.parallel.foreach?view=net-6.0

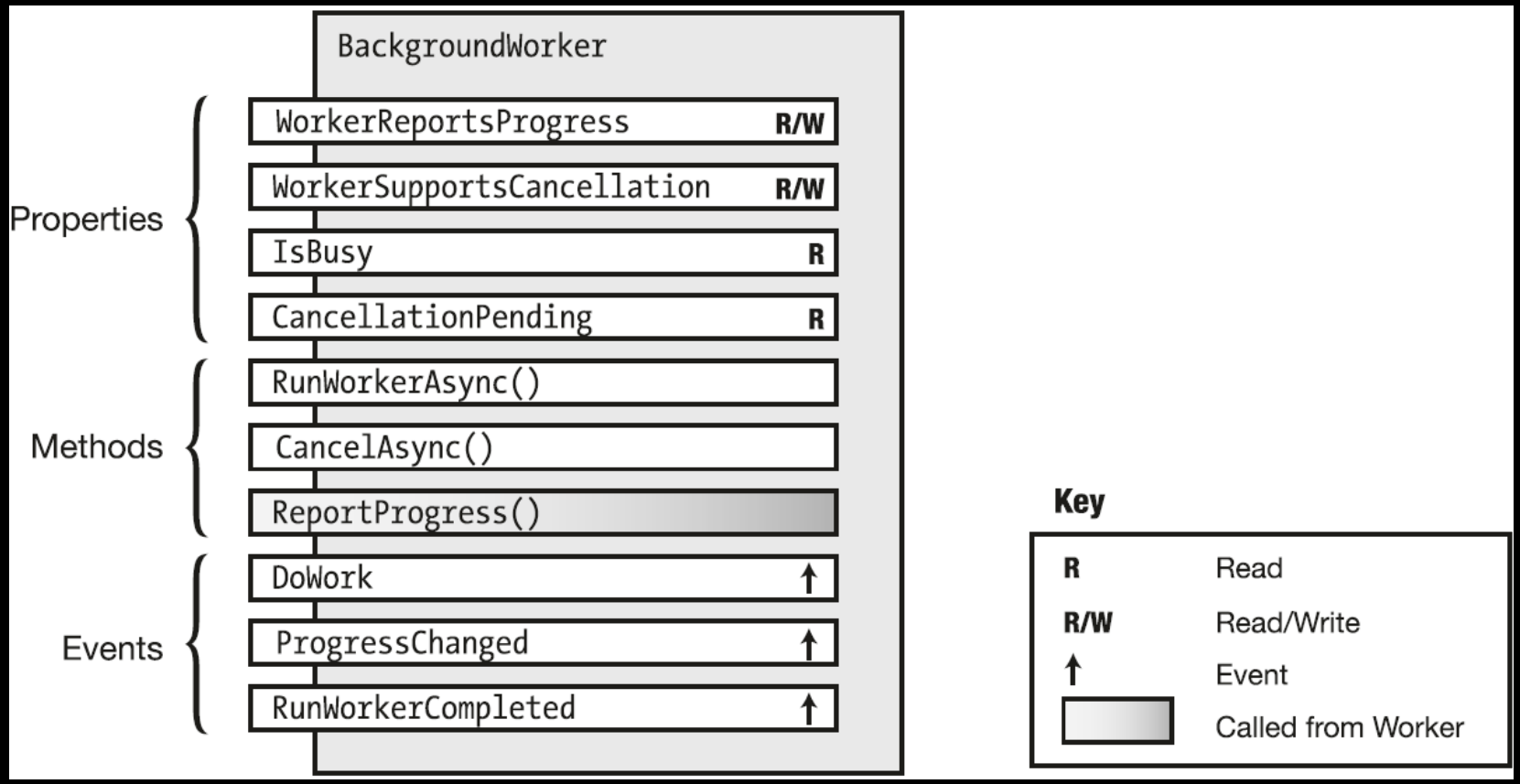
PARALLEL.FOREACH

```
public static void Main02ForEach()  {
    string[] squares = new string[]
                { "We", "hold", "these", "truths", "to", "be",
                "self-evident", "that", "all", "men", "are", "created", "equal" };
    Parallel.ForEach(squares,
        i => Console.WriteLine(string.Format("{0} has {1} letters", i, i.Length)));
}
```

Результат: "We" has 2 letters
"truths" has 6 letters
"be" has 2 letters……
"equal" has 5 letters
"to" has 2 letters

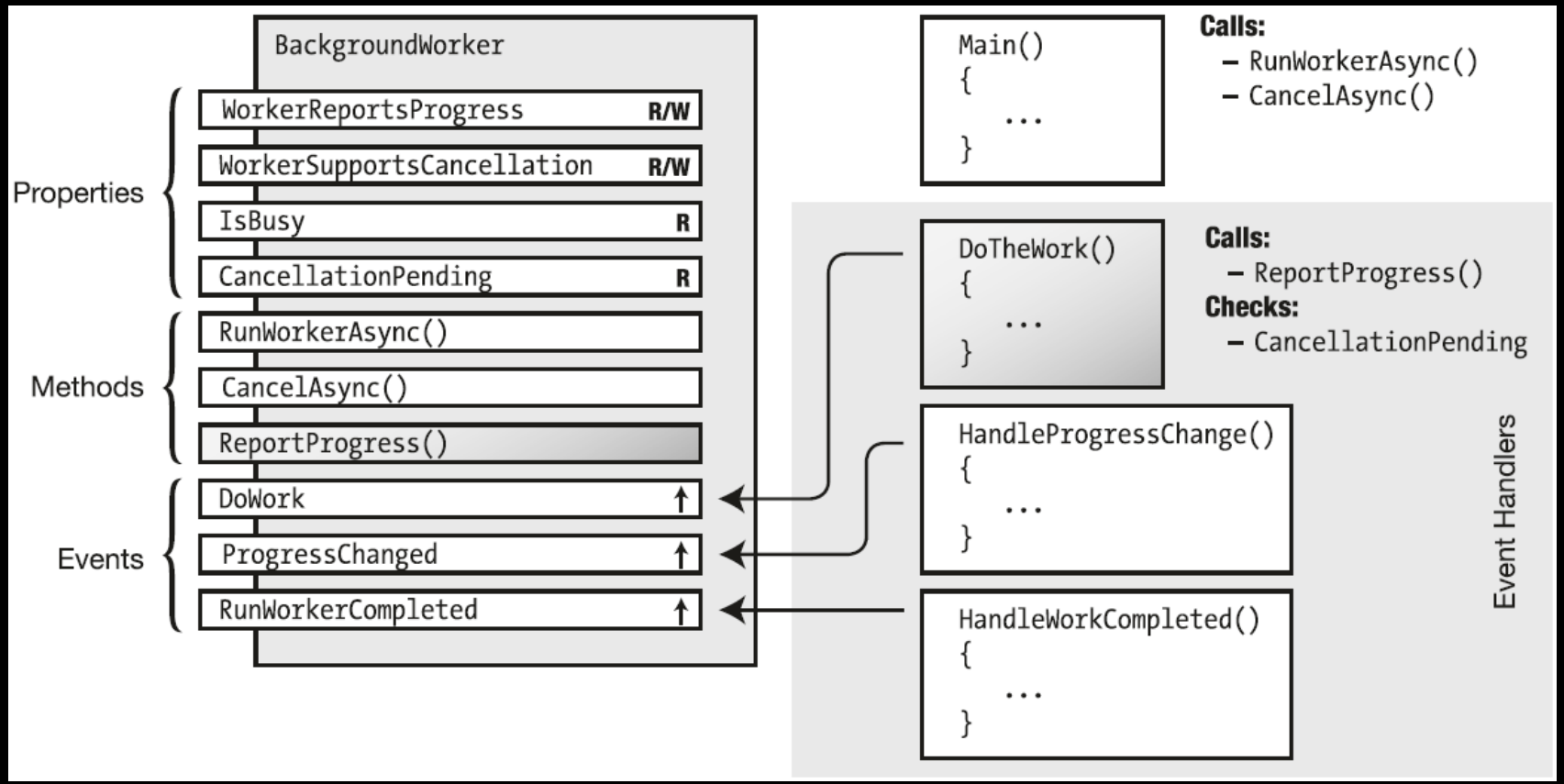
КЛАСС BACKGROUNDWORKER

Содержится в пространстве имен System.ComponentModel



https://learn.microsoft.com/ru-ru/dotnet/api/system.componentmodel.backgroundworker?view=net-7.0

ПРОГРАММА С ОБЪЕКТОМ КЛАССА…



ДЕЛЕГАТЫ СОБЫТИЙ ДЛЯ КЛАССА BACKGROUNDWORKER

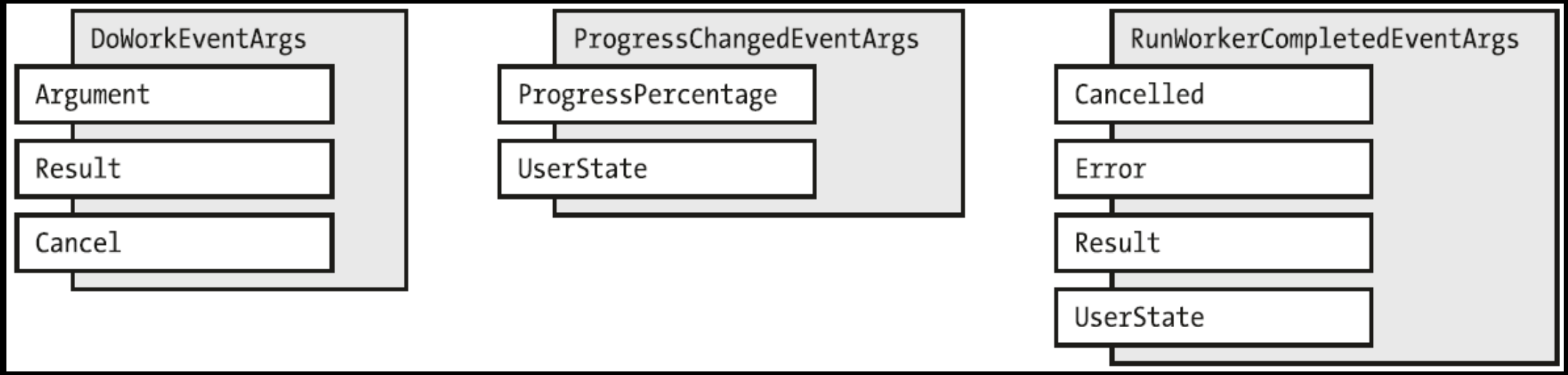
```
void DoWorkEventHandler (object sender, DoWorkEventArgs e)

void ProgressChangedEventHandler (object sender, ProgressChangedEventArgs e)

void RunWorkerCompletedEventHandler (object sender,RunWorkerCompletedEventArgs e)
```

КЛАССЫ, ПРОИЗВОДНЫЕ ОТ EVENTARGS

- DoWorkEventArgs
- ProgressChangedEventArgs
- RunWorkerCompletedEventArgs



КОНСОЛЬНЫЙ ПРИМЕР OUNDWORKER

```
using System;
using System.ComponentModel; // BackgroundWorker
using System.Threading;

namespace DemoLect_MultiThreading {
    public class BackgroundWorkerDemo {
        public static void Main() { // Основной поток
            BackgroundWorker bgw = new BackgroundWorker();
            bgw.WorkerReportsProgress = true;
            bgw.DoWork += (sender, args) => {
                for (int n = 0; n++ < 50;) {
                    Thread.Sleep(50);  // Задержка потока
                    Console.Write("w");
                    if (n % 5 == 0)
                        bgw.ReportProgress(2 * n);
                }
            };
            bgw.ProgressChanged += (sender, args) => {
                Console.Write(args.ProgressPercentage + "%");
            };
```

КОНСОЛЬНЫЙ ПРИМЕР BACKGROUNDWORKER (ПРОДОЛЖЕНИЕ)

```
            bgw.RunWorkerCompleted += (sender, args) => {
                Console.WriteLine("\r\nCancelled!");
            };

            Console.WriteLine("Основной поток выводит точки!");
            bgw.RunWorkerAsync();   // Фоновый поток запущен

            while (true) {   // Цикл основного потока, Ctrl + C для прерывания
                Console.Write(".");
                // Искусственная задержка основного потока:
                for (int k = 0; k < int.MaxValue / 300; k++) ;
            }
        }
    }
}
```

ССЫЛКИ

- Параллельное программирование в .NET. Руководство по документации (https://learn.microsoft.com/ru-ru/dotnet/standard/parallel-programming/)
- Асинхронная модель на основе задач (TAP) в .NET: введение и обзор (https://learn.microsoft.com/ru-ru/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap)
- Асинхронное программирование (https://learn.microsoft.com/ru-ru/dotnet/csharp/asynchronous-programming/async-scenarios)
- https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/async/
- https://docs.microsoft.com/ru-ru/dotnet/standard/async-in-depth