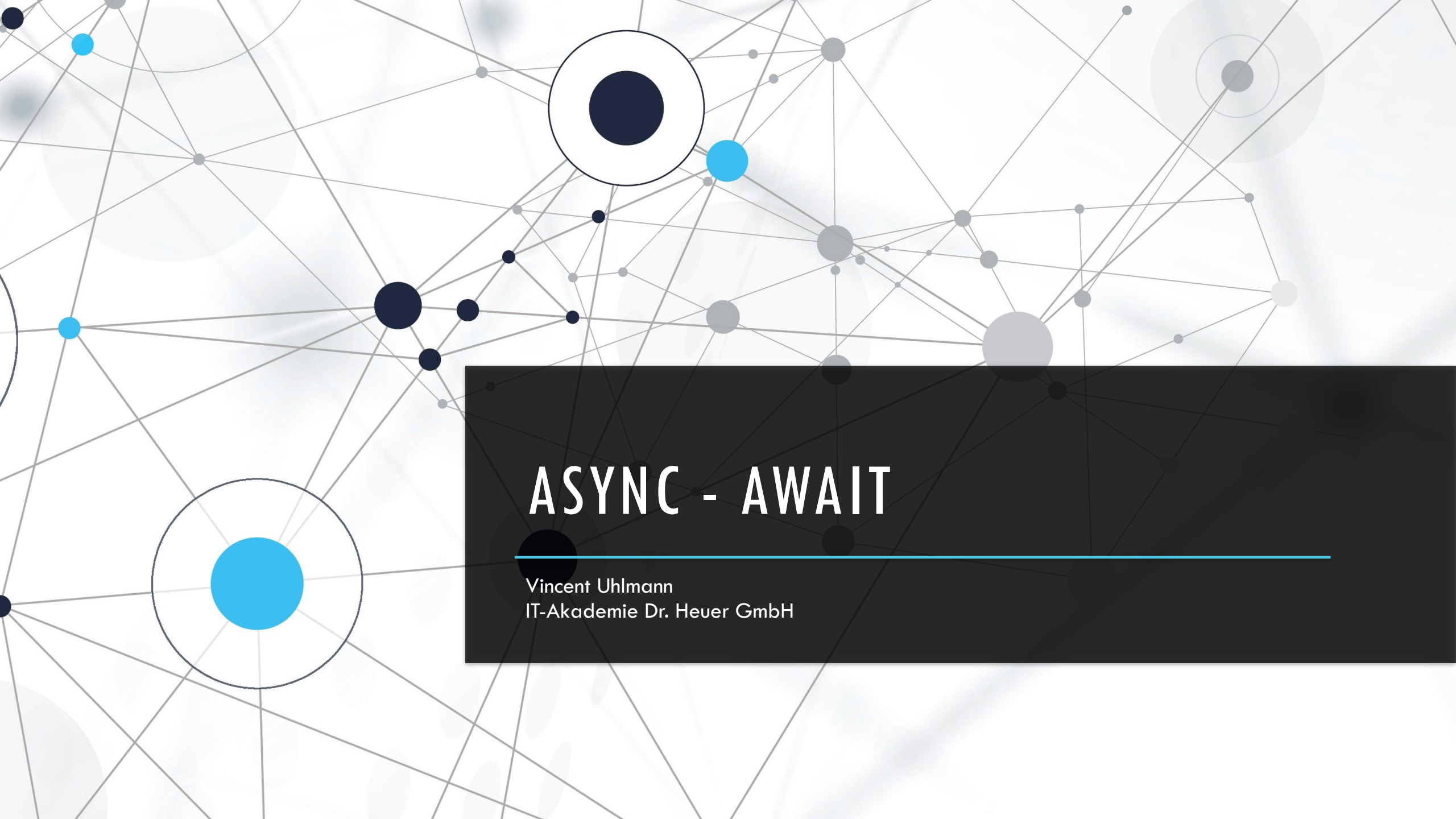

ASYNC - AWAIT

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

ASYNC AWAIT

- Schlüsselwörter async / await
- Ermöglicht uns „Linearen Code“ zu schreiben, der Compiler macht die Arbeit
- Langelaufende Operationen wie z.B. Dateizugriffe blockieren nicht den MainThread
- Konzept der „Future Variables“
- Das Ergebnis einer Methode wird zugewiesen, obwohl das Ergebnis erst später bekannt ist
- .NET Klassen bieten häufig asynchrone Methoden an

ASYNC AWAIT

- Schlüsselwort async markiert eine Methode als asynchrone Methode

- Schlüsselwort await macht den Rest der Methode zu einem „Callback“ (Continuation)

- Task.Delay verzögert in diesem Beispiel die Ausführung ähnlich wie Thread.Sleep, ohne jedoch den aufrufenden Thread zu blockieren

```
public static async Task DoSomething()
{
    await Task.Delay(1000);
}
```

ASYNC AWAIT

- Eine mit async deklarierte Methode, wird im Laufe ihrer Ausführung in einem eigenen Thread fortgeführt
- Nach dem Start des Threads wird die Kontrolle zurück an den Aufrufer gegeben

```
static async Task Main(string[] args)
{
    await DoSomething();
}

public static async Task DoSomething()
{
    await Task.Delay(1000);
}
```

ASYNC AWAIT

- Asynchrone Methoden können folgende Rückgabetypen haben
- void
- Task
- Task<T>

- Der Rückgabetyp void entspricht dem Fire and Forget Prinzip, sollte wenn möglich vermieden werden (siehe Beispiel auf der nächsten Folie)
- Jede Asynchrone Methode muss ein await Statement enthalten

```
static async Task Main(string[] args)
{
    await DoSomething();
}

public static async Task DoSomething()
{
    await Task.Delay(1000);
}
```

ASYNC AWAIT

```
static void Main(string[] args)
{
    Console.WriteLine("1. Beginn Main");
    DoSomething();
    Console.WriteLine("5. Ende Main");
    Console.ReadLine();
}

public static async void DoSomething()
{
    Console.WriteLine("2. Methode DoSomething wurde aufgerufen");
    Console.WriteLine("3. Starte FooMethode");
    await Task.Run(() => FooMethode());
    Console.WriteLine("7. DoSomething ist fertig");
}

public static void FooMethode()
{
    Console.WriteLine("4. Aufgabe läuft");
    Thread.Sleep(5000);
    Console.WriteLine("6. Aufgabe ist fertig");
}
```

ASYNC AWAIT

- Asynchrone Methoden geben den Rückgabewert in einem Task verpackt zurück
- Rückgabe im Code mit return
- Das Schlüsselwort await entpackt den Rückgabewert einer asynchronen Methode

```
public static async Task AwaitSomething()
{
    int i = await DoSomethingAsync();
}

public static async Task<int> DoSomethingAsync()
{
    await Task.Delay(1000);
    return 5;
}
```

ASYNC AWAIT UND EXCEPTIONS

- Treten während der Ausführung Ausnahmen (Exceptions) auf, müssen diese beim Aufruf von await abgefangen werden

```
public static async Task AwaitSomething()
{
    try {
        int i = await DoSomethingAsync();
    } catch(Exception e) {
        // Fehler behandeln
    }
}

public static async Task<int> DoSomethingAsync()
{
    throw new Exception("Fehler");
    await Task.Delay(1000);
    return 5;
}
```

ASYNC AWAIT VORTEILE UND NACHTEILE

- Asynchroner Code sieht wie synchroner Code aus
- Callbacks müssen nicht manuell geschrieben werden
- Threads müssen nicht synchronisiert werden

- Das benutzen von async/await hat einen Overhead und ist damit nicht für kleine, kurze Operationen geeignet

ASYNC AWAIT NAMENSKONVENTION

- Asynchrone Methoden sollten immer mit dem Postfix Async enden

```
public static async Task<int> DoSomethingAsync()
{
    await Task.Delay(1000);
    return 5;
}
```