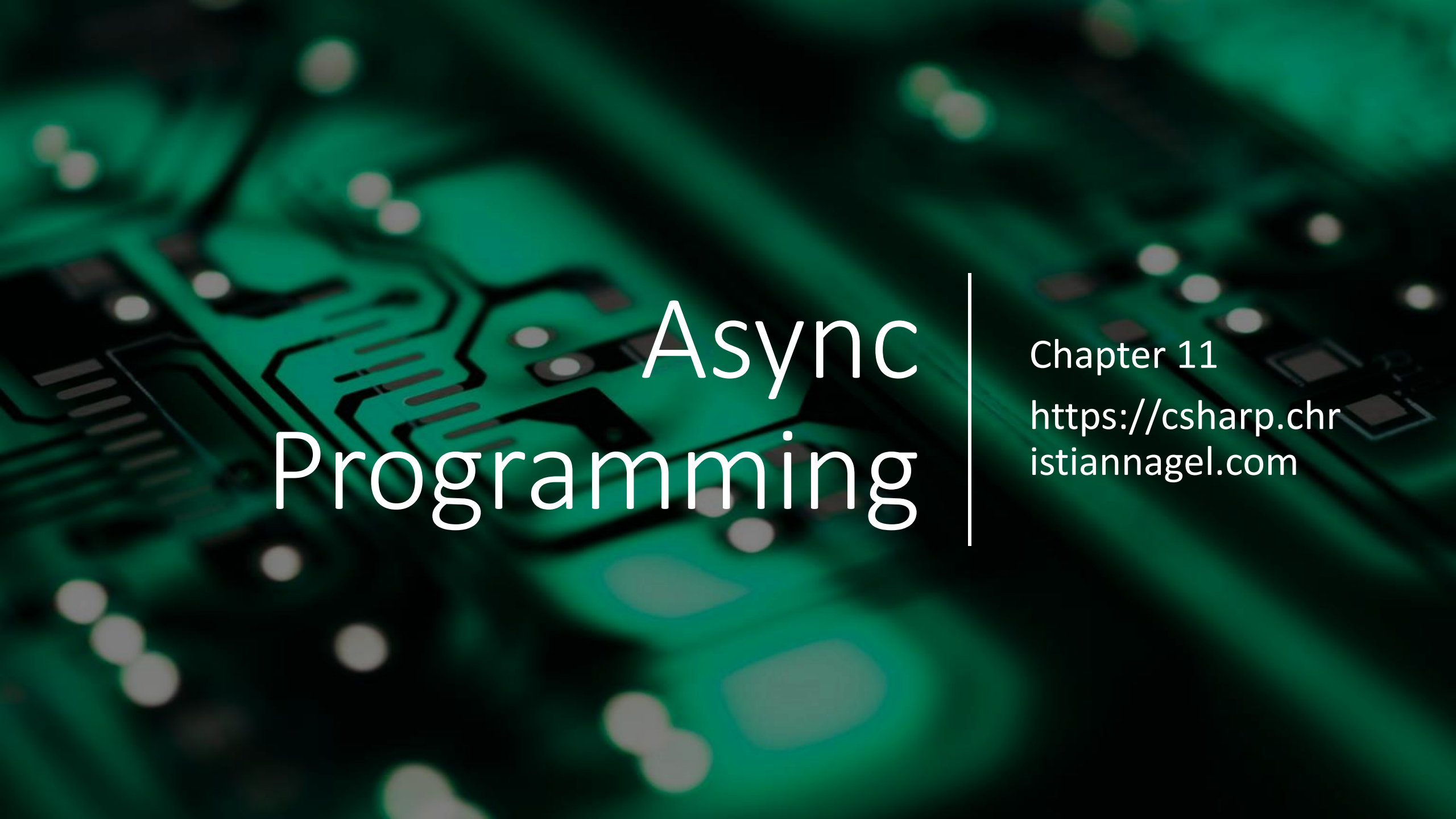


# Async Programming

Chapter 11

https://csharp.christiannagel.com

# Async Patterns

## Async Pattern

- IAsyncResult
- AsyncCallback
- BeginXX / End XX methods

## Event-based Async Pattern

- void method with Async postfix
- Completed event

## Task-based Async Pattern

- using async/await
- Method with Async Postfix returns a Task

# Create a Task

- Task.Run

```
static Task<string> GreetingAsync(string name) =>
  Task.Run(() =>
  {
    TraceThreadAndTask($"running {nameof(GreetingAsync)}");
    return Greeting(name);
  });
```

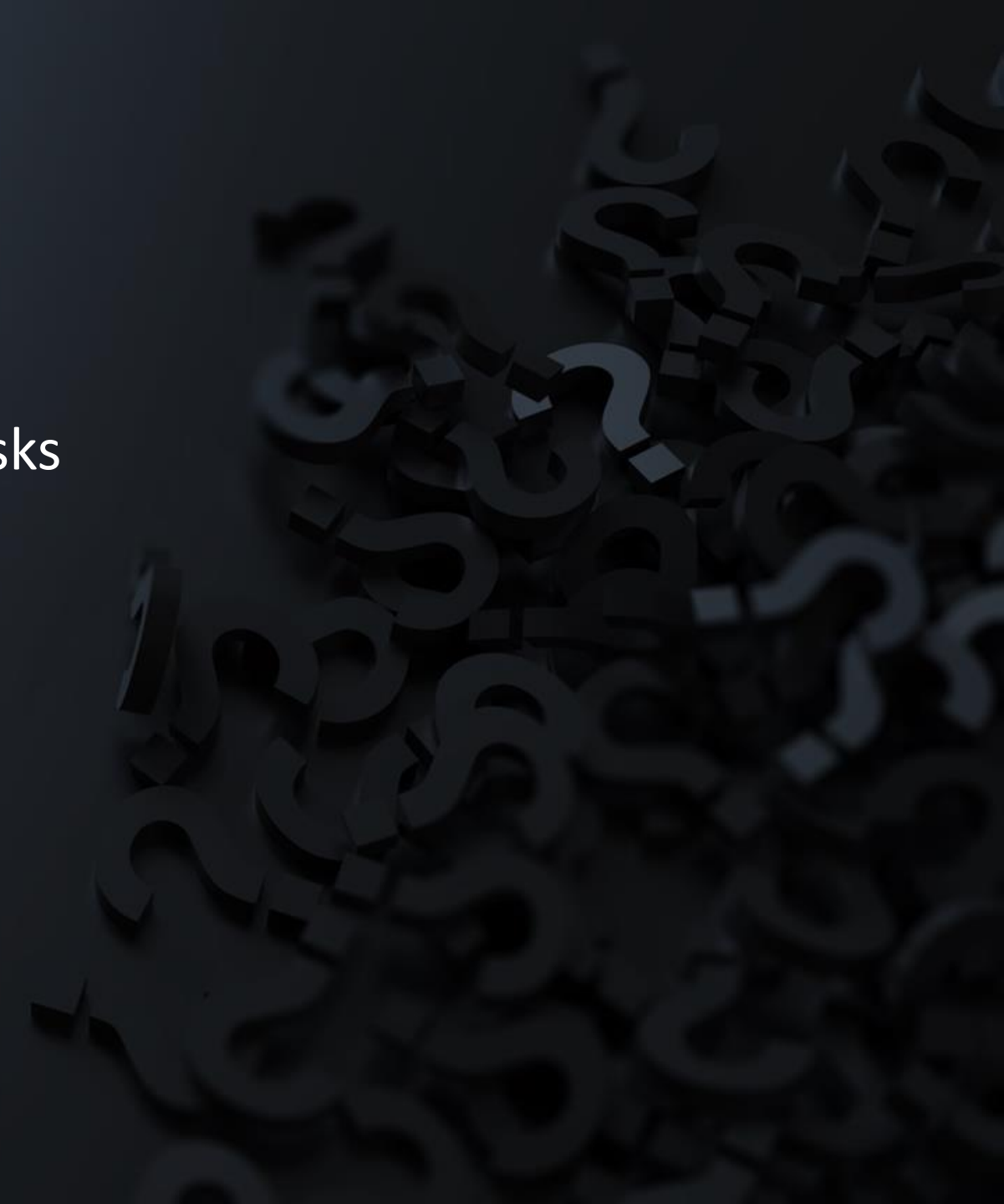
# Calling an Async Method

- async/await

```
private async static void CallerWithAsync()
{
  TraceThreadAndTask($"started {nameof(CallerWithAsync)}");
  string result = await GreetingAsync("Stephanie");
  Console.WriteLine(result);
  TraceThreadAndTask($"ended {nameof(CallerWithAsync)}");
}
```

# Error Handling

- Without awaiting, no errors
- AggregateException with multiple tasks

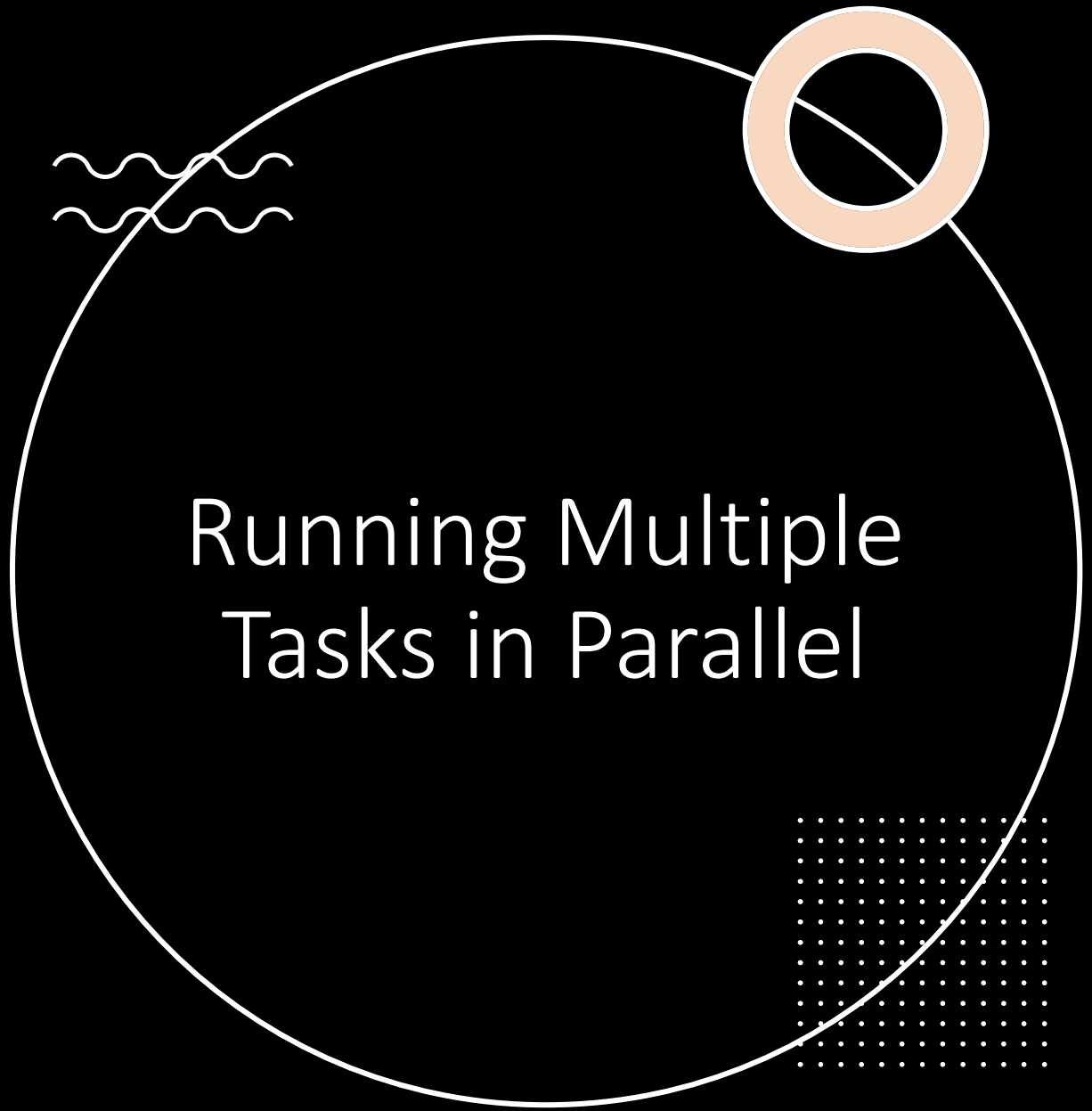


# Guidelines Async Methods

- Use the Async postfix if possible
- Don’t return void unless necessary
- If void is necessary, catch exceptions
- Don’t mix async/await with blocking calls

# async/await

- Doesn’t block the calling thread
- The method continues (and gets a result) as the task completes
- Uses the synchronization context (if available)



# Running Multiple Tasks in Parallel

- Use Task.WhenAll
- wait for all tasks to complete

# Synchronization Context

- Configured with WPF/UWP/Windows Forms applications
- Console applications don’t have a synchronization context (yet)

# Summary

- Task
- async/await
- Error Handling
- Task-based Async Pattern