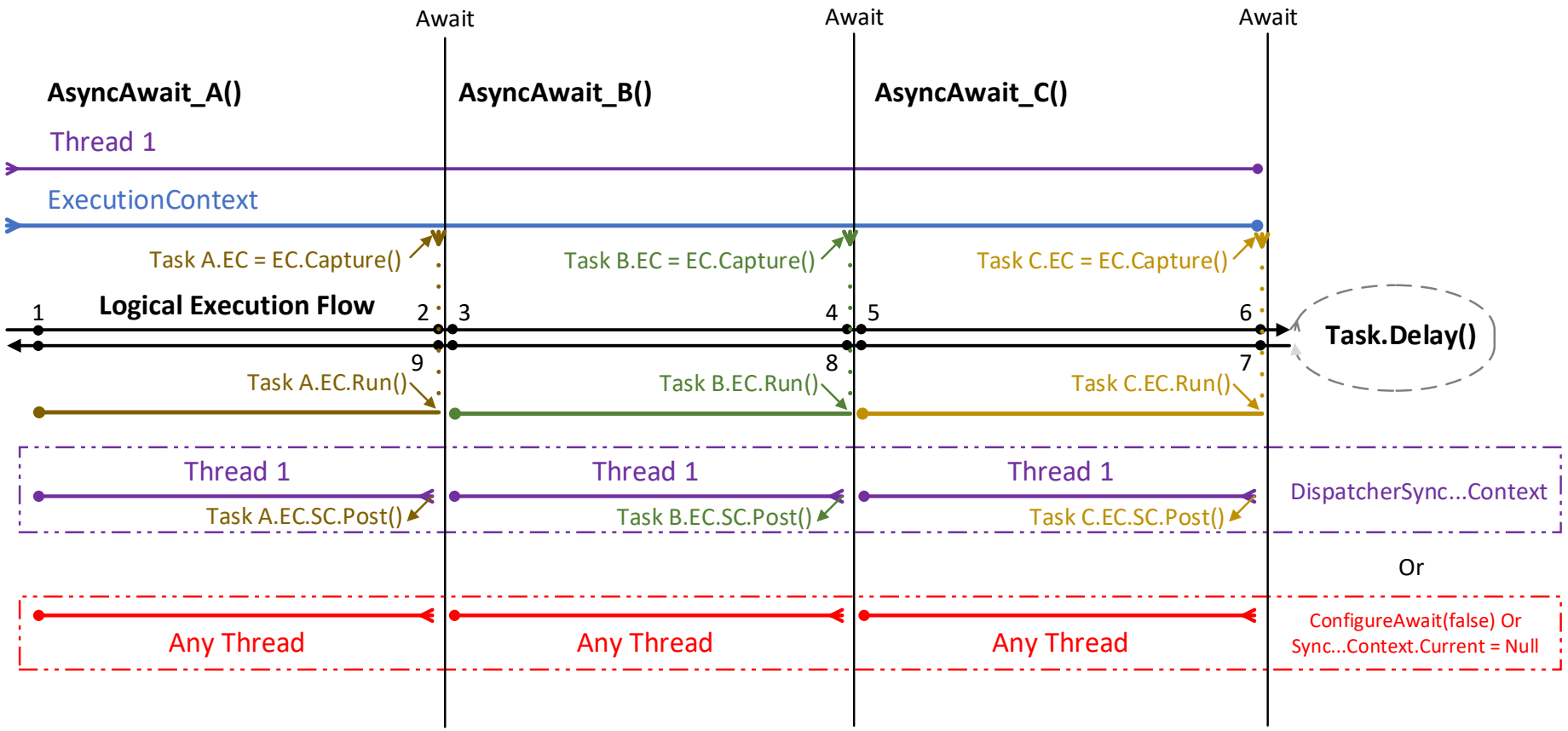
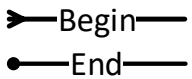
```
public static ThreadLocal<string> threadLocalA = new ThreadLocal<string>();
public static ThreadLocal<string> threadLocalB = new ThreadLocal<string>();
public static ThreadLocal<string> threadLocalC = new ThreadLocal<string>();

//// AsyncLocal is within a container held within ExecutionContext
public static AsyncLocal<string> asyncLocalA = new AsyncLocal<string>();
public static AsyncLocal<string> asyncLocalB = new AsyncLocal<string>();
public static AsyncLocal<string> asyncLocalC = new AsyncLocal<string>();

public static async Task AsyncAwait_A(Action<string> output
    , bool continueOnCapturedSynchronizationContext = true, int pause = 1000)
{
    // Logical Execution 1
    Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("fr-FR");
    threadLocalA.Value = "A"; asyncLocalA.Value = "A";
    LogicalExecution(1, output);

    // Logical Execution 2
    await AsyncAwait_B(output, continueOnCapturedSynchronizationContext, pause)
        .ConfigureAwait(continueOnCapturedContext: continueOnCapturedSynchronizationContext);

    // Logical Execution 9
    LogicalExecution(9, output);
}

private static async Task AsyncAwait_B(Action<string> output
    , bool continueOnCapturedSynchronizationContext, int pause = 1000)
{
    // Logical Execution 3
    Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("en-US");
    threadLocalB.Value = "B"; asyncLocalB.Value = "B";
    LogicalExecution(3, output);

    // Location Execution 4
    await AsyncAwait_C(output, continueOnCapturedSynchronizationContext, pause)
    .ConfigureAwait(continueOnCapturedContext: continueOnCapturedSynchronizationContext);

    // Logical Execution 8
    LogicalExecution(8, output);

}

private static async Task AsyncAwait_C(Action<string> output
    , bool continueOnCapturedSynchronizationContext, int pause = 1000)
{
    // Logical Execution 5
    Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("es-MX");
    threadLocalC.Value = "C"; asyncLocalC.Value = "C";
    LogicalExecution(5, output);

    // Location Execution 6
    await Task.Delay(millisecondsDelay: pause)
        .ConfigureAwait(continueOnCapturedContext: continueOnCapturedSynchronizationContext);

    // Logical Execution 7
    LogicalExecution(7, output);

}
```

ExectionContext, SynchronizationContext, Threading and Logical Exectuion Flow



Interconnects:

- Task.ExecutionContext
- Thread.ExecutionContext
- ExecutionContext.SynchronizationContext



Using Async / Await in C# as designed

- Understand and follow the Task-Based Asynchronous Programming (TAP) design pattern and use it in *all* layers of your application.
- Always deal with the returned Task from an awaitable method
    - Await it, return it, Task.Wait() or Task.Result.
- Allow Async/Await to spread in your application.
    - Start at the top and connect the Async / Await flow deeper
        - Top: Use 'async Task' for ASP.NET Controller and Unit Test methods and 'Async Void' for WPF/Forms events.
    - It's better to have async available and not need it then to need it and not have it. Use Task.FromResult<> or Task.CompletedTask to end an async flow without an 'await'.
- Never use Async Void anywhere outside of WPF / Forms control events (ButtonClick, OnLoaded, etc.)
    - This is unsafe in all platforms where the SyncrhonizationContext.Current is null. Exceptions will be lost.
- End with a .ConfigureAwait(false) on every await method call within your libraries
    - This allows async methods to be used with .Result or .Wait() when necessary without the risk of blocking.
    - This is annoying, yes, but this is why ConfigureAwait was provided
- Use the System.Threading.Tasks Namespace Toolbox. No longer use System.Thread.
    - Task.Delay, Tasks.WhenAll, Tasks.WaitAll, Tasks.CompletedTask
    - TaskCompletionSource instead of AutoResetEvent
    - AsyncLocal instead of ThreadLocal
    - Task.CompletedTask and Task.FromResult<T>()
- Use Visual Studio 2012 or newer
    - Critical – Introduced 'async Task' TestMethod for MsTest
- .Net Framework 4.6.1 or newer
    - Critical pieces like Task.CompletedTask and AsyncLocal are introduced and other important async APIs.