

.NET / ASYNC UNDER THE SURFACE

Await the work. Not the thread.

Follow the execution.

Watch the state machine keep its place.

C# async / await

Tasks

Continuations

AN ILLUSTRATIVE .NET MODEL



Arrow keys to explore · Synchronized lab on slide 3

The same code. Two possible paths.

The caller supplies a task representing a read. The method returns a different task representing the count.

THE C# METHOD

```
static async Task<int> CountAsync(  
    Task<string> readTask)  
{  
    string text = await readTask;  
    return text.Length;  
}
```

`readTask`

`Task<string>` · the read result

`CountAsync(...)`

`Task<int>` · the method result

AWAITER ALREADY COMPLETED

Keep going.

Get the result immediately.
No suspension at this await.

Run ————— GetResult ————— Finish

AWAITER NOT YET COMPLETED

Save. Yield. Resume.

Save the resume point and awaiter.
Continue after completion.

Run ————— Suspend ————— Resume

Execution on the left. State on the right.

COUNTASYNC / INPUT "hello"

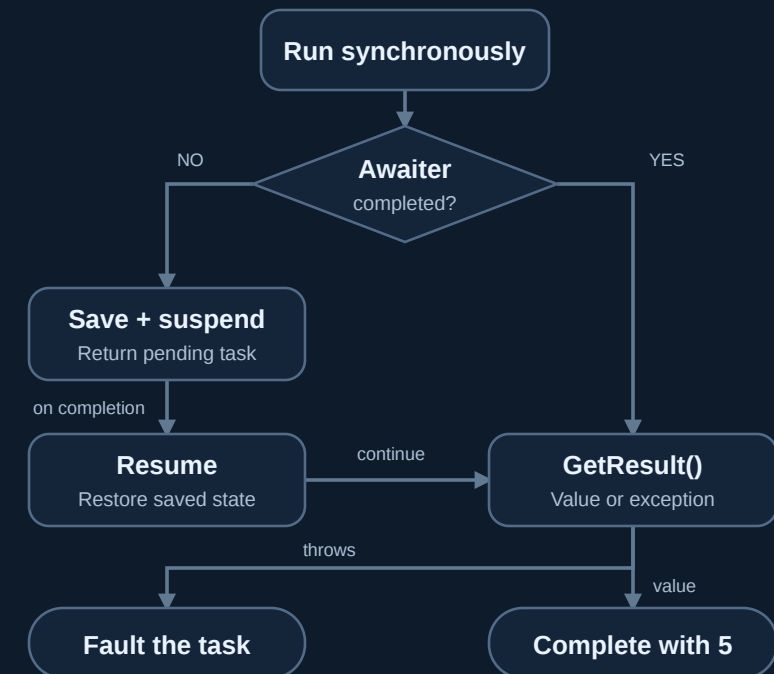
STATIC MAP / ALL PATHS

EXECUTION + TASK LIFETIMES

```
01 static async Task<int> CountAsync(Task<string> readTask)
02 {
03     string text = await readTask;
04     return text.Length;
05 }
```

THIS METHOD / CPU Runs → yields → resumes**READ TASK / STRING** Pending → value or error**COUNT TASK / INT** Pending → 5 or fault**SAVED STATE** Resume point + awaiter when suspended

CONCEPTUAL COMPILER STATE MACHINE



Read the branches: incomplete → save / resume; already complete → direct to GetResult. No dedicated thread waits for the pictured I/O.

The method remembers where to continue.

WHILE THE METHOD IS SUSPENDED

RESUME POINT

Which await should continue?

AWAITER

How to observe completion and get the result.

NEEDED VALUES

Only state that must survive suspension.

METHOD BUILDER

Completes or faults the returned task.

A conceptual view. Generated fields and optimizations vary.

01 / NO MAGIC THREAD

async does not mean Task.Run.

The body starts synchronously.

Naturally asynchronous I/O need not occupy a waiting thread.

02 / NO GUARANTEED YIELD

await can keep going.

An already-completed awaiter takes the fast path. A continuation need not change threads.

03 / NO CONFUSION ABOUT TASKS

Completion belongs to an operation.

The read task delivers text.

The count task delivers a number, or a fault.

THE MENTAL MODEL

Split execution at an incomplete await. Preserve state. Continue when ready.

Sources: C# await operator · Task-based async model · C# async functions / state machines