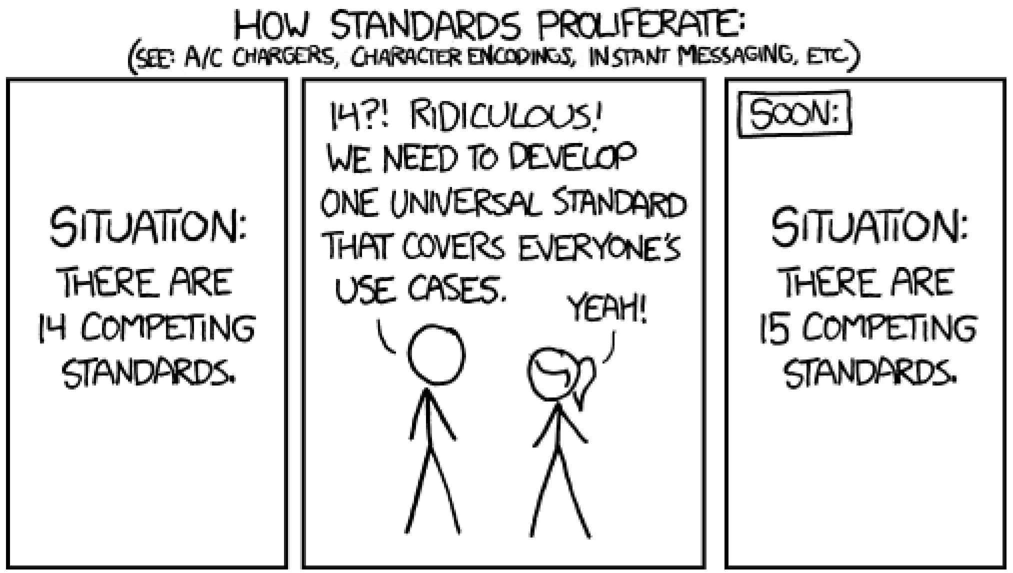
Async2

Andy Gocke

V1 – Green threads

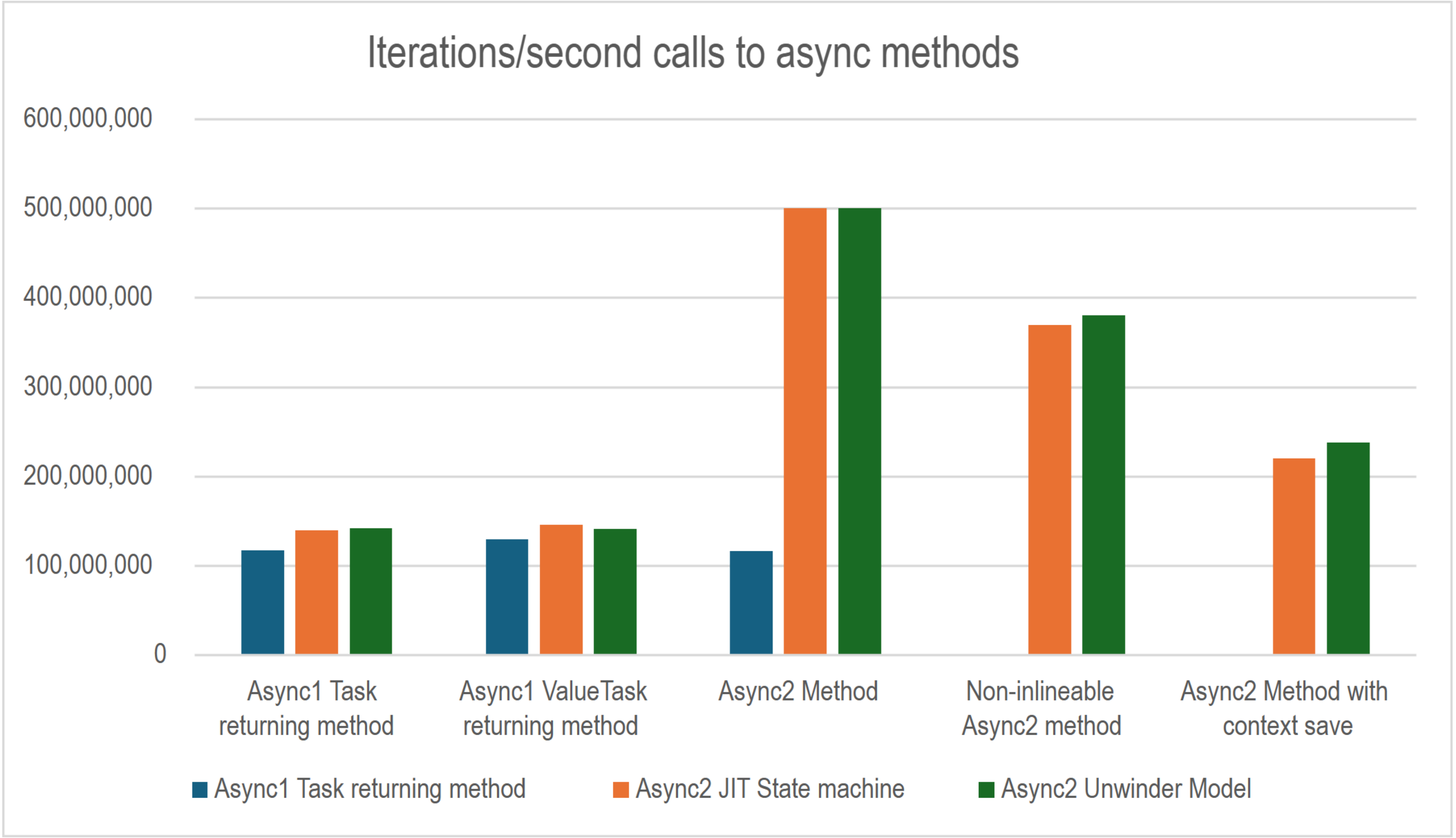
- Green Thread Experiment Results · Issue #2398 · dotnet/runtimelab (github.com)


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

V2 - The Pitch

- Async is basically a calling convention
  - You can either do a “suspend” call, or a “schedule” call
- Currently implemented by state machines in C#
- What if we move it to the runtime?
  - The runtime knows more, and it can cheat

The Results

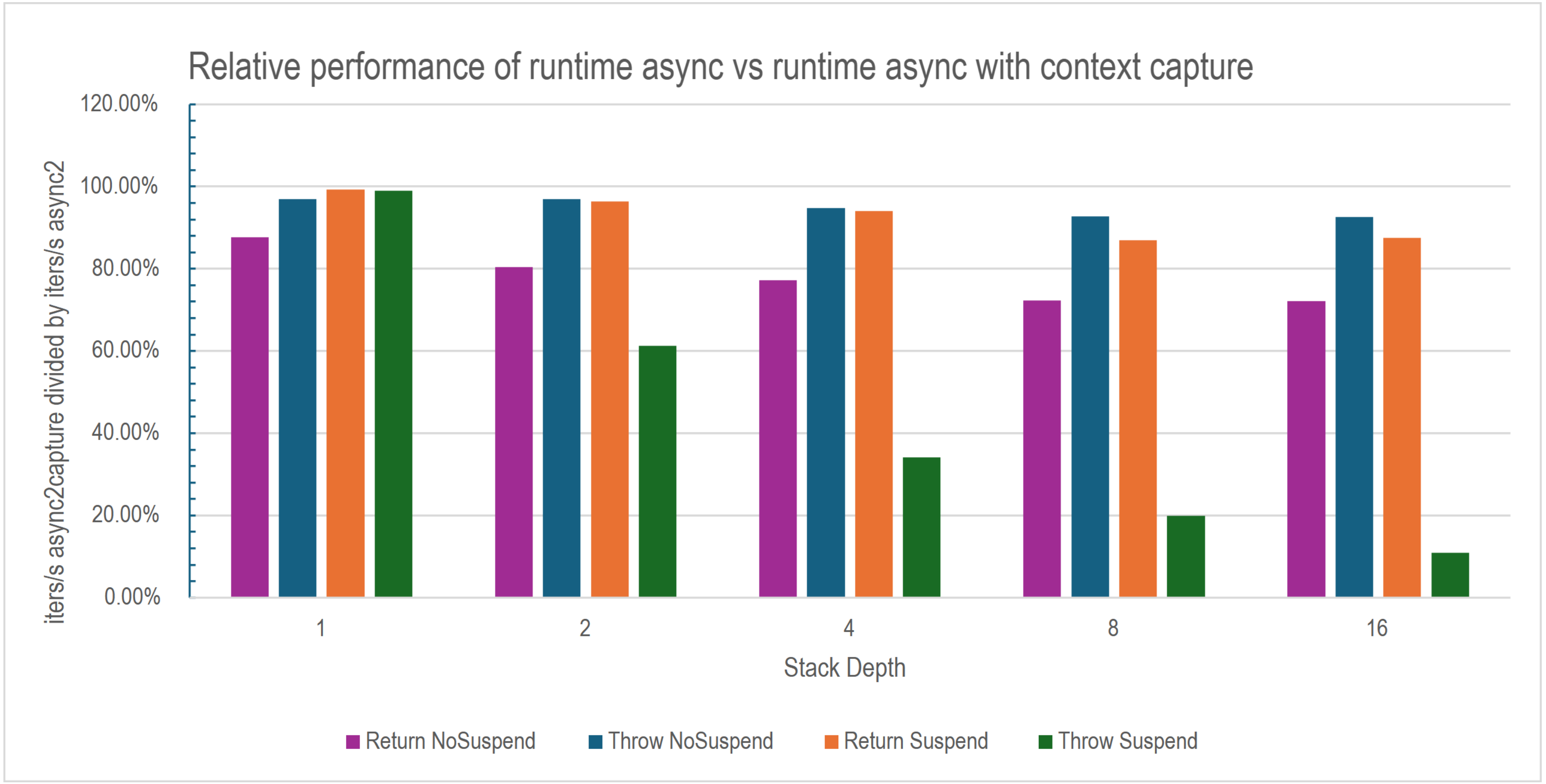


Iterations/second calls to async methods
600,000,000
500,000,000
400,000,000
300,000,000
200,000,000
100,000,000
0
Async1 Task returning method
Async1 ValueTask returning method
Async2 Method
Non-inlineable Async2 method
Async2 Method with context save
🟦 Async1 Task returning method
🟧 Async2 JIT State machine
🟩 Async2 Unwinder Model

Execution/Sync Context / AsyncLocals

- Current semantics: save restore on every async entry/exit
- Async local modifications don’t flow up
- Problem: save/restore limitations look like async1

Execution/Sync Context / AsyncLocals



Semantic questions

- Sync context change?
- ConfigureAwait
  - ConfigureAwaitAttribute
- Control async v. async2?
- Async2 delegates?
- Async void?
- Async iterator?

Opportunity: DispatchScope?

```
DispatchScope.RunOnMainThread(() =>
    …
});
DispatchScope.RunOnDefault(() => {
    …
});
```

Opportunity: CancelScope?

```
static void M() {
     CancelScope.Open((scope) => {
           M2(); M3(); M4();
     });
}
static async Task M2() {
      await Task.Delay(2, CancelScope.Current.Token);
}
static async Task M3() {
      CancelScope.Current.ThrowIfCanceled();
      await Task.Delay(2);
}
static async2 Task M4() {
     await Task.Delay(2); // Equivalent to M3();
}
```