BEHIND MODERN CONCURRENCY PRIMITIVES

INTRODUCTION

Bartosz Sypytkowski

@Horusiath

b.sypytkowski@gmail.com

bartoszsypytkowski.com

AGENDA

- Concurrency in user vs. kernel space
- Thread pools
- Schedulers
- Coroutines
- State machines

WHY DO WE USE THIS?

ASYNC/AWAIT

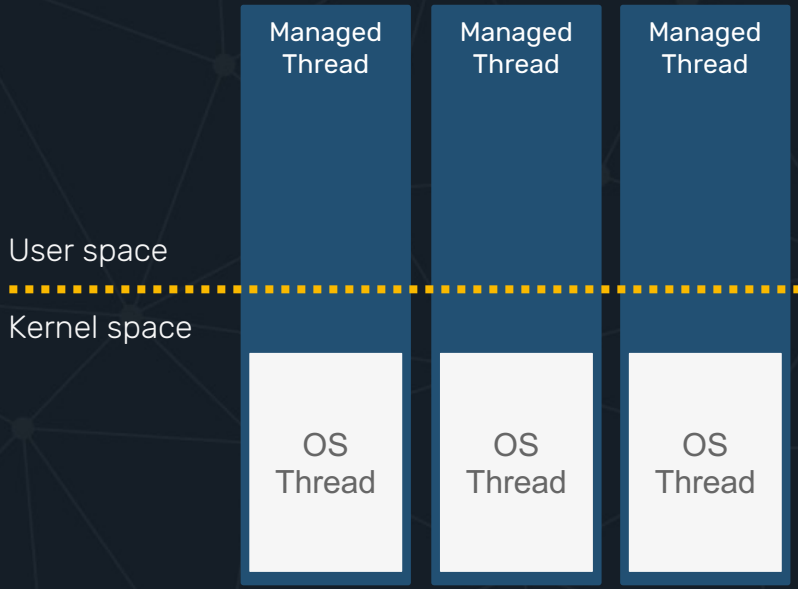
```
Task.Run(async () =>
{
    await using var conn = new SqlConnection(ConnectionString);
    await conn.OpenAsync();
    var user = await conn.QueryFirstAsync<User>(
        "SELECT * FROM Users WHERE Id = @id",
        new { id = 100 });

    Console.WriteLine($"Received user {user.FirstName} {user.LastName}");
});
```

INSTEAD OF THIS?

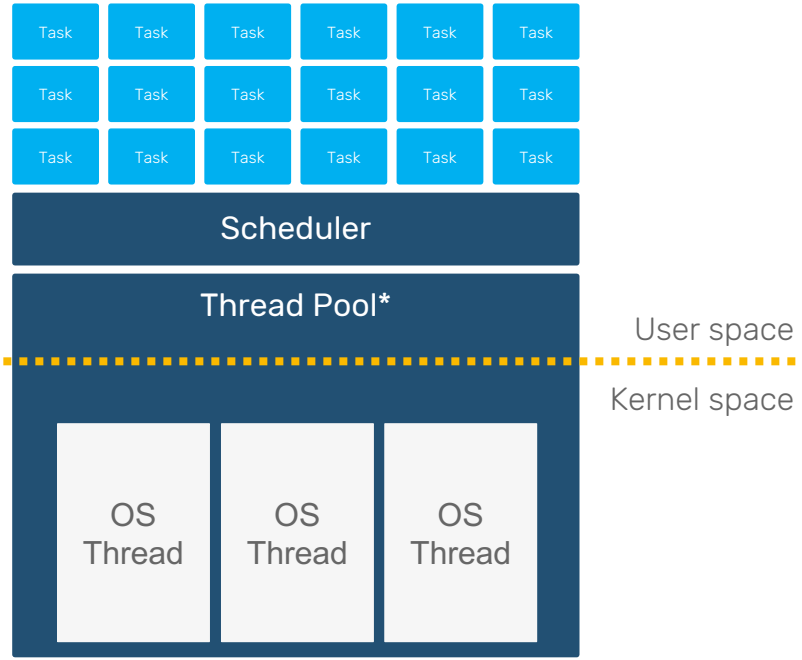
THREAD API

```
new Thread(() =>
{
    var conn = new SqlConnection(ConnectionString);
    conn.Open();
    var user = conn.QueryFirst<User>(
        "SELECT * FROM Users WHERE Id = @id",
        new { id = 100 });

    Console.WriteLine($"Received user {user.FirstName} {user.LastName}");
}).Start();
```

THREADS



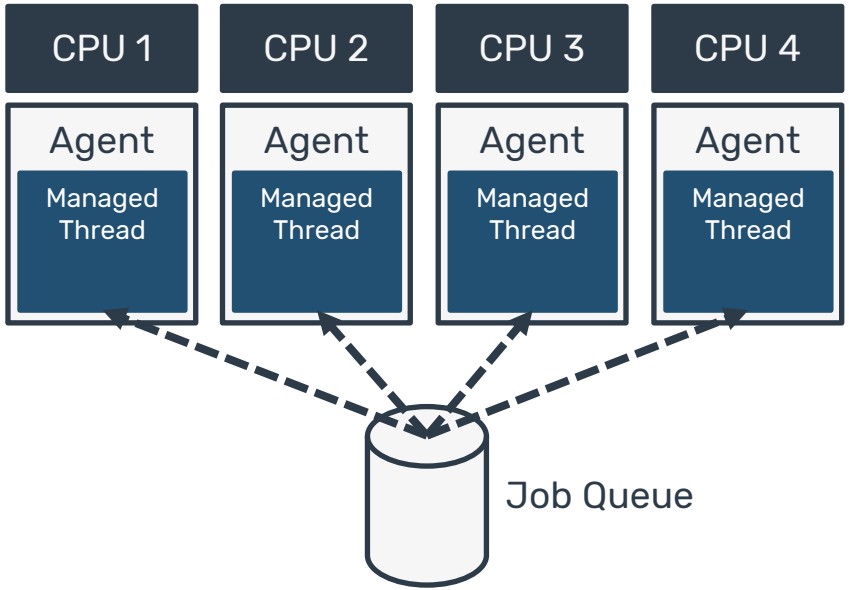
TASKS



THREAD POOLS

THREAD POOL

BASICS



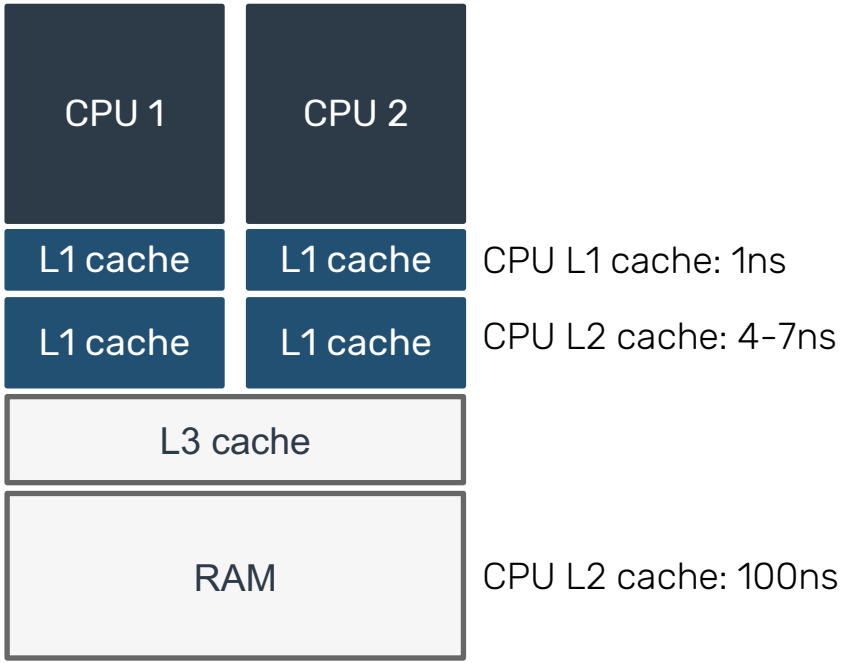
PINNING THREAD TO CPU

```
var threads = Process.GetCurrentProcess().Threads;
var cpuCount = Environment.ProcessorCount;
Console.WriteLine($"Using {threads.Count} threads on {cpuCount} cores.");
// => Using 10 threads on 4 cores.
for (int i = 0; i < threads.Count; i++)
{
    var pthread = threads[i];
    var cpuMask = (1L << (i % cpuCount));

    pthread.IdealProcessor = i % cpuCount;       // set preferred thread(s)
    pthread.ProcessorAffinity = (IntPtr)cpuMask; // set thread affinity mask
}
```

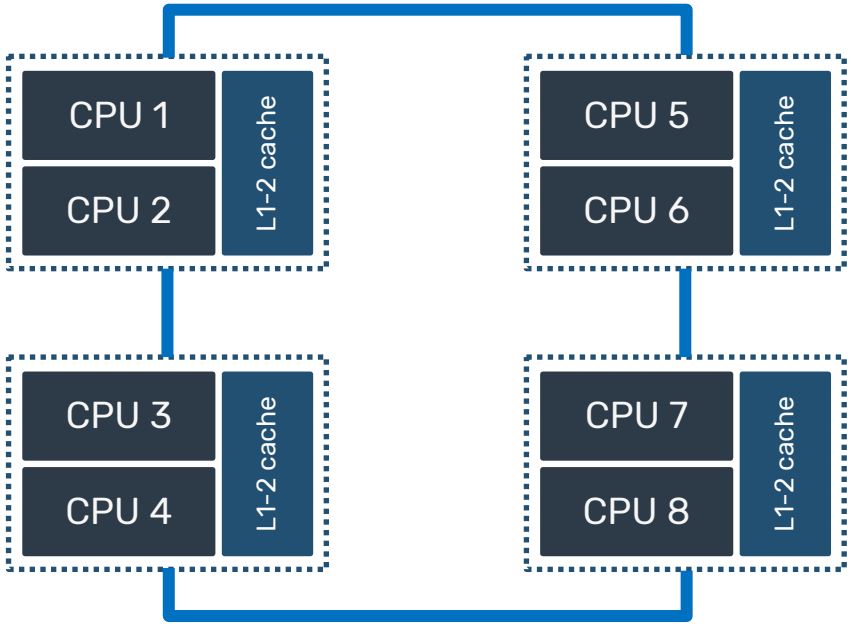
NOTE: it’s not always possible (iOS) or respected.

SHARING DATA IN MULTI-CPU ARCHITECTURE

HARDWARE ARCHITECTURE 101

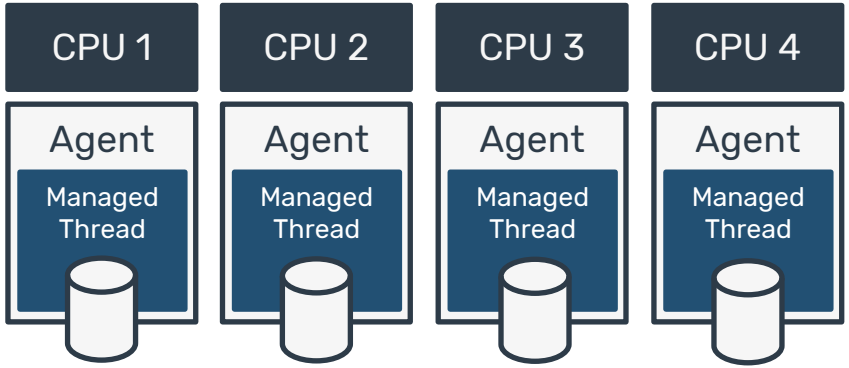


HARDWARE ARCHITECTURE
NUMA



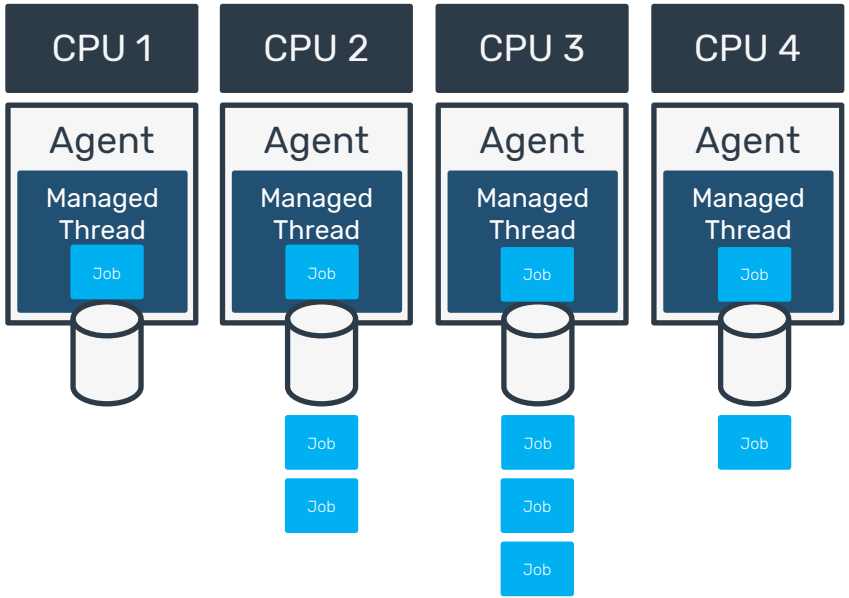
THREAD POOL

MULTIPLE QUEUES



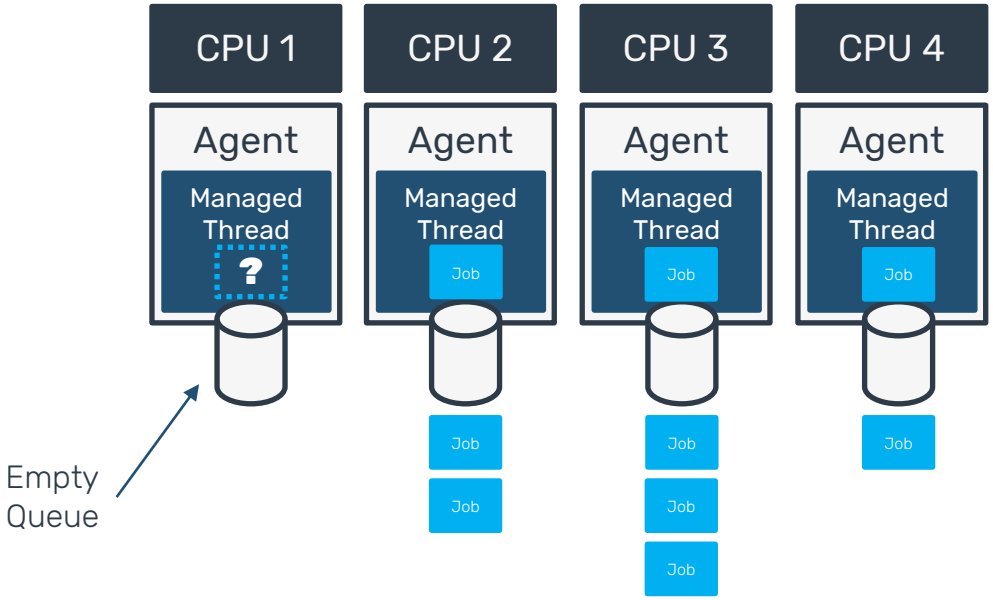
THREAD POOL

WORK STEALING



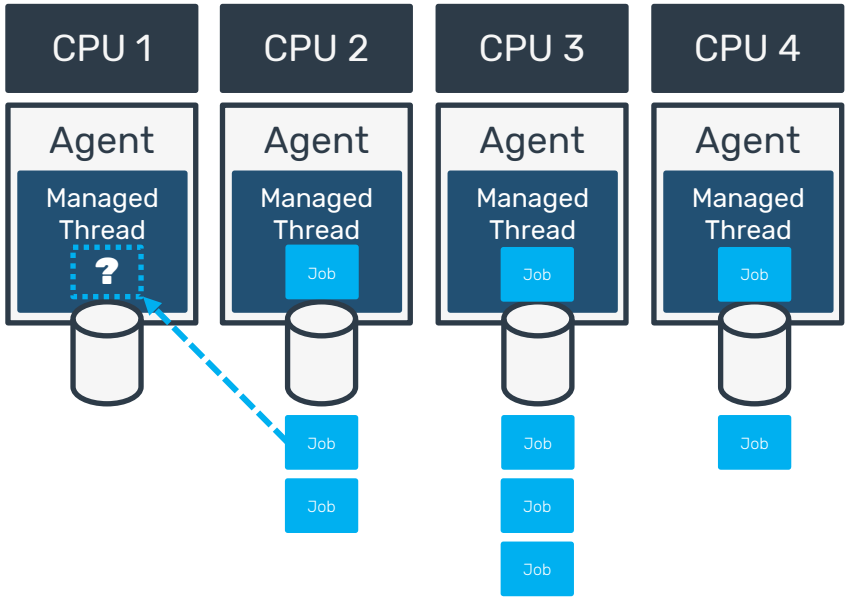
THREAD POOL

WORK STEALING



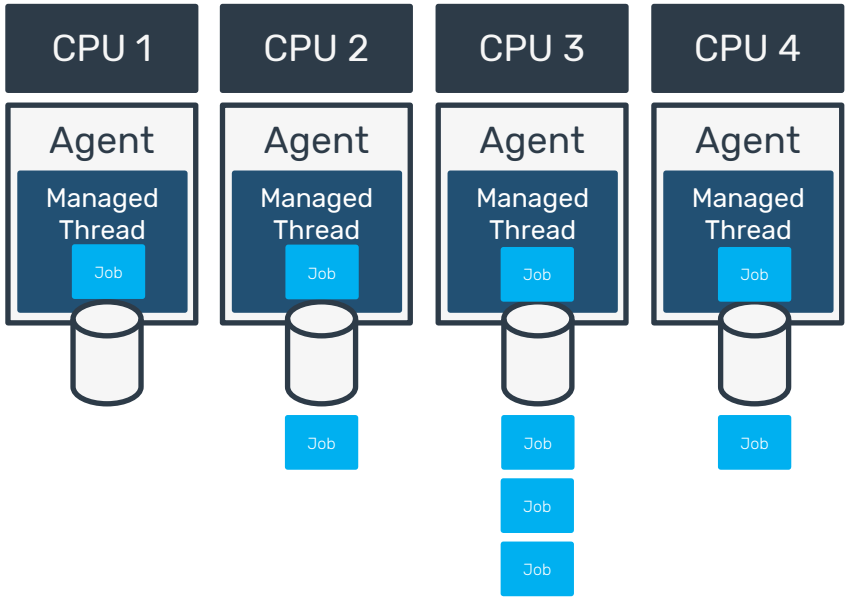
THREAD POOL

WORK STEALING



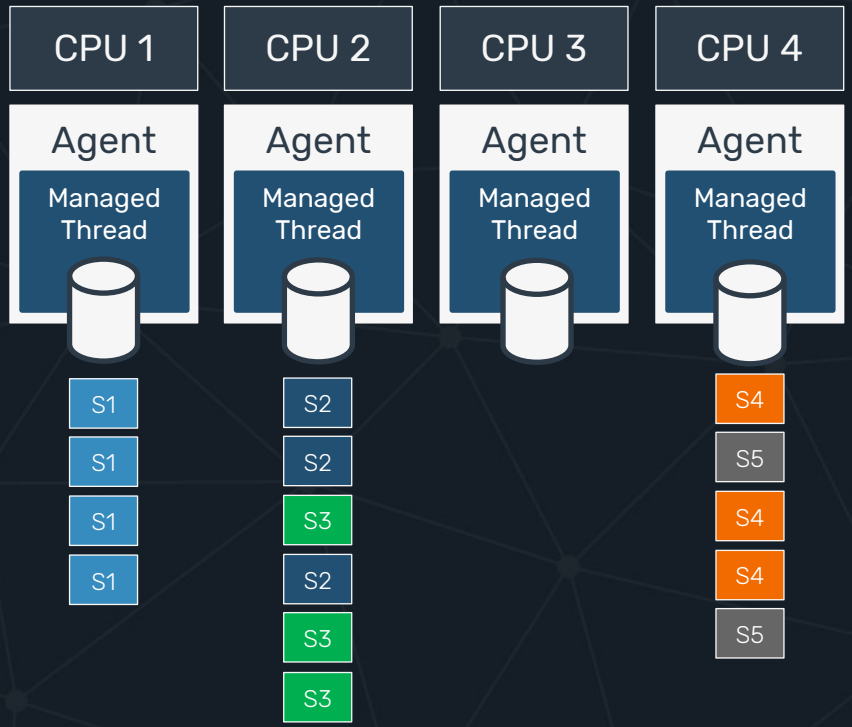
THREAD POOL

WORK STEALING

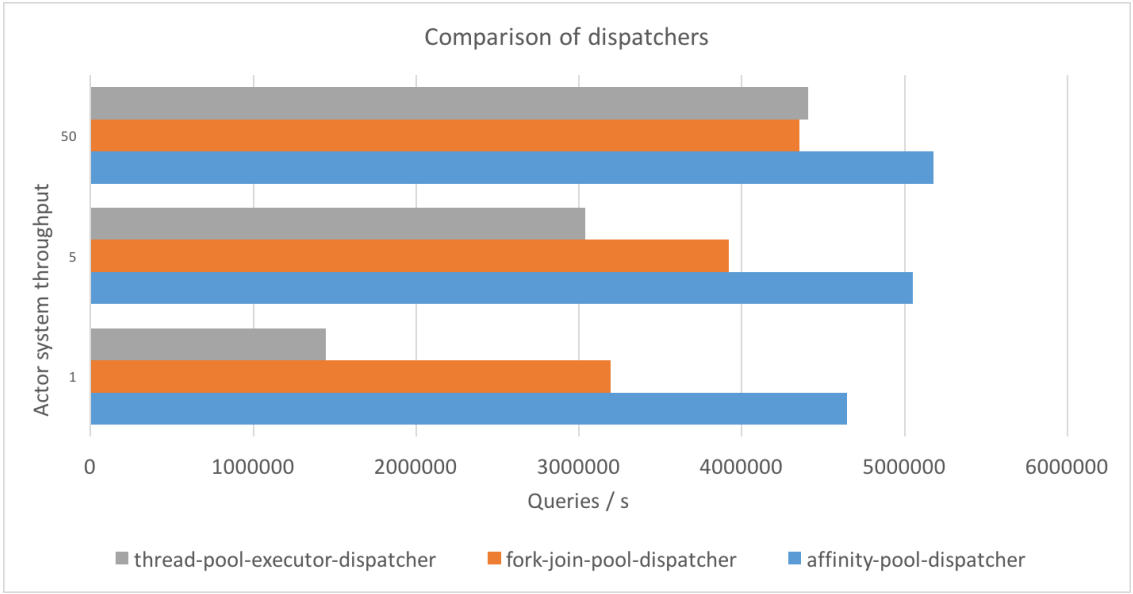


AFFINITY BASED THREAD POOL

*Always map **entity** to the same thread / core*



AFFINITY THREAD POOL: PERFORMANCE



Source: https://scalac.io/improving-akka-dispatchers/

THREAD-PER-CORE ARCHITECTURE

WORKING WITH I/O

SCENARIO #1

READING A FILE

WHY IS THIS CODE BAD?

```
static async Task ProcessFiles(FileInfo[] files)
{
    var tasks = new Task[files.Length];
    for (int i = 0; i < files.Length; i++)
        tasks[i] = ProcessFile(files[i]);

    await Task.WhenAll(tasks);
}

static async Task ProcessFile(FileInfo file)
{
    using var stream = file.OpenRead();
    using var reader = new StreamReader(stream);
    var text = reader.ReadToEnd();

    Process(text);
}
```

WHY IS THIS CODE BAD?

Blocking the thread belonging to a thread pool shared with other tasks.

```
static async Task ProcessFiles(FileInfo[] files)
{
    var tasks = new Task[files.Length];
    for (int i = 0; i < files.Length; i++)
        tasks[i] = ProcessFile(files[i]);

    await Task.WhenAll(tasks);
}

static async Task ProcessFile(FileInfo file)
{
    using var stream = file.OpenRead();
    using var reader = new StreamReader(stream);
    var text = reader.ReadToEnd();

    Process(text);
}
```

I/O

WHAT CAN WE DO ABOUT IT?

1. Just use async I/O API…

I/O

WHAT CAN WE DO ABOUT IT?

1. Just use async I/O API…
2. … but what when it’s not possible?

I/O

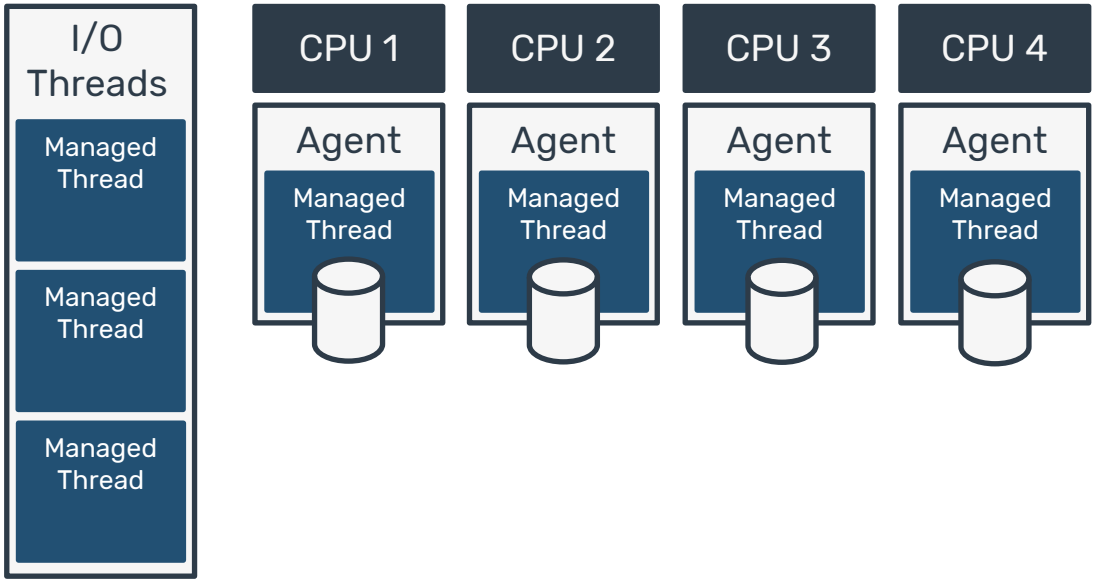
WHAT CAN WE DO ABOUT IT?

1. Just use async I/O API…
2. … but what when it's not possible?

```
internal static class LmdbMethods
{
    [DllImport("lmdb", CallingConvention = CallingConvention.Cdecl)]
    public static extern int mdb_dbi_open(IntPtr txn, string name, DatabaseOpenFlags flags, out uint db);

    [DllImport("lmdb", CallingConvention = CallingConvention.Cdecl)]
    public static extern int mdb_cursor_get(IntPtr cursor, ref ValueStructure key, ref ValueStructure
data, CursorOperation op);

    [DllImport("lmdb", CallingConvention = CallingConvention.Cdecl)]
    public static extern int mdb_cursor_put(IntPtr cursor, ref ValueStructure key, ref ValueStructure
value, CursorPutOptions flags);

    // other methods
}
```

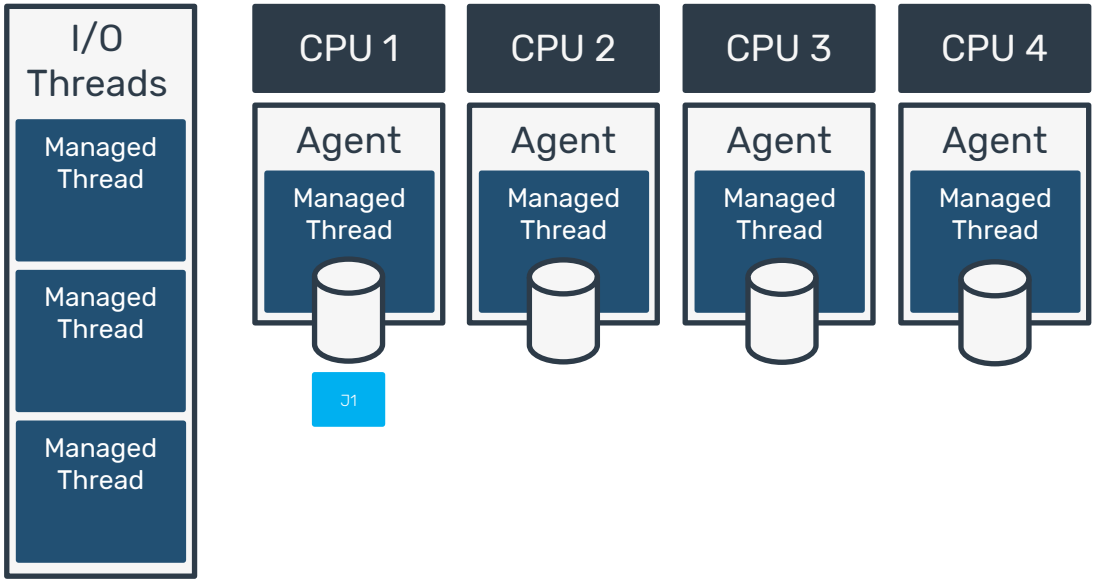
THREAD POOL

I/O THREADS



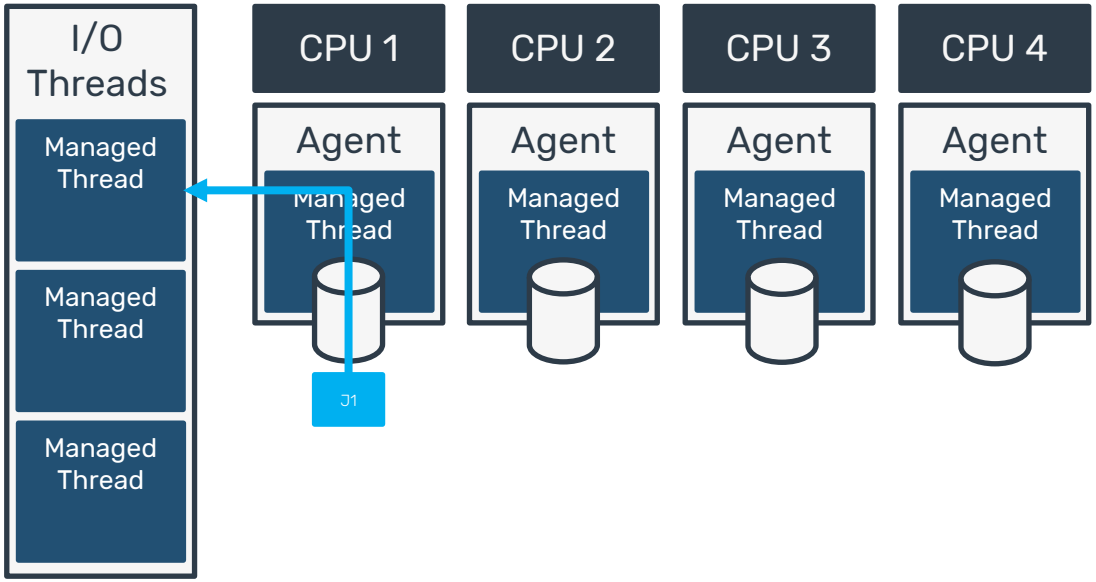
THREAD POOL

I/O THREADS



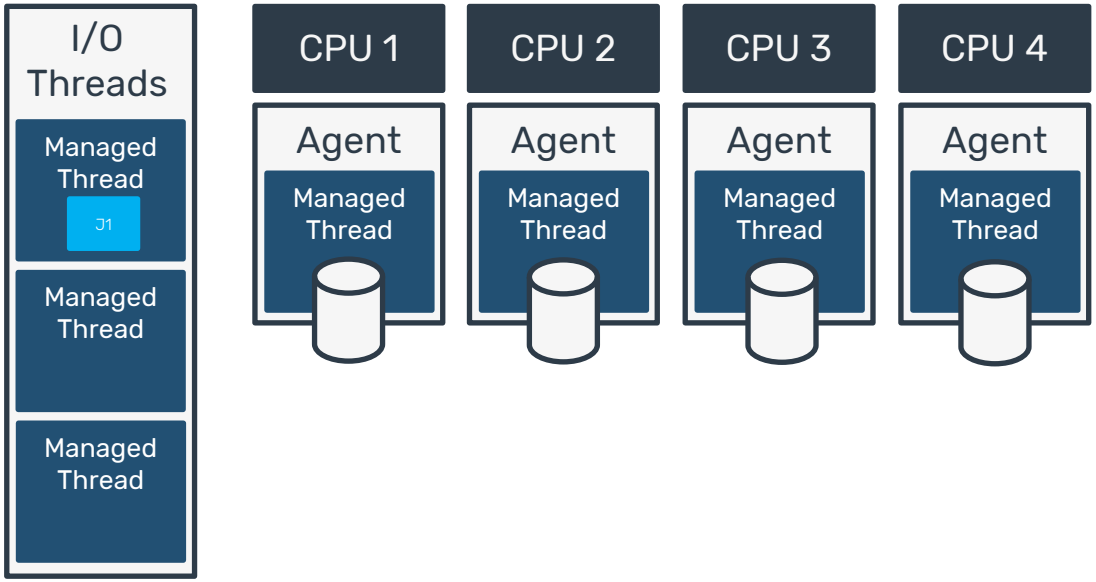
THREAD POOL

I/O THREADS



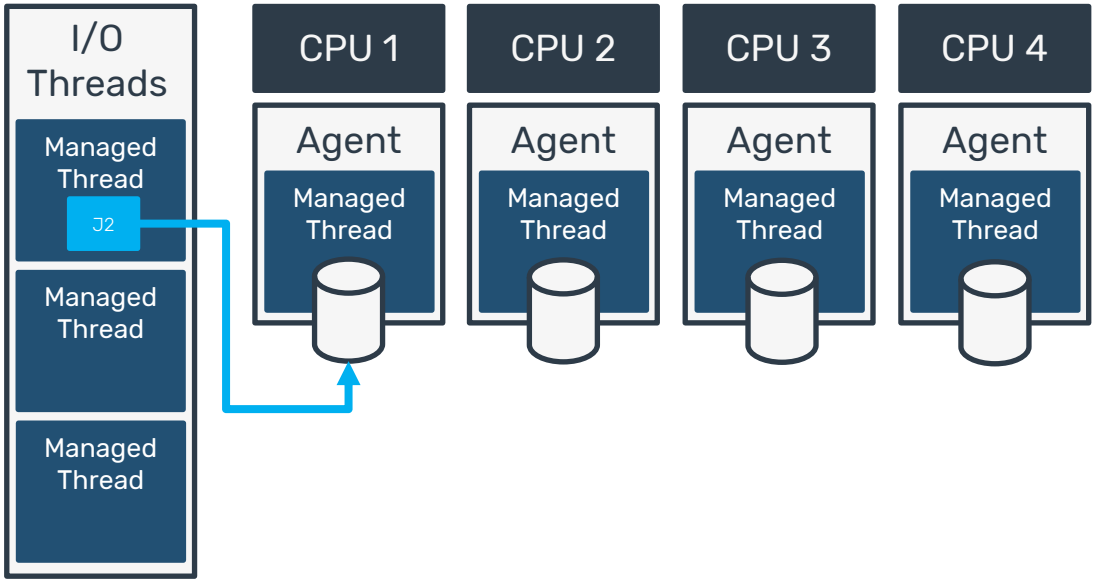
THREAD POOL

I/O THREADS



THREAD POOL

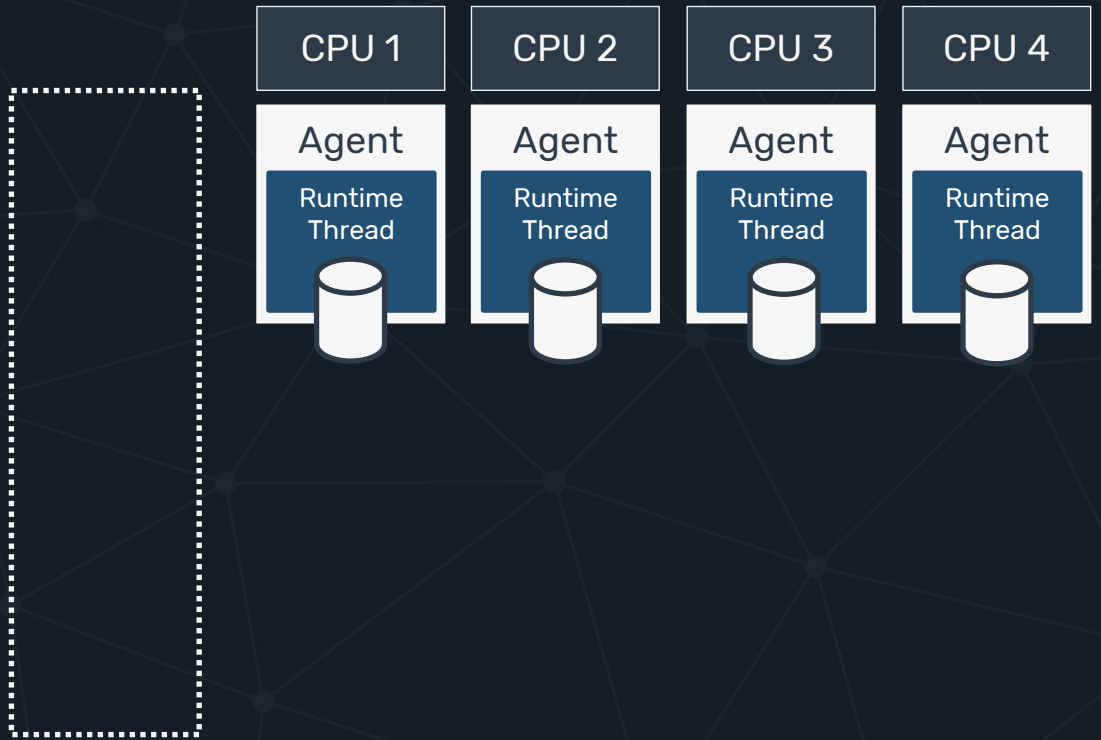
I/O THREADS



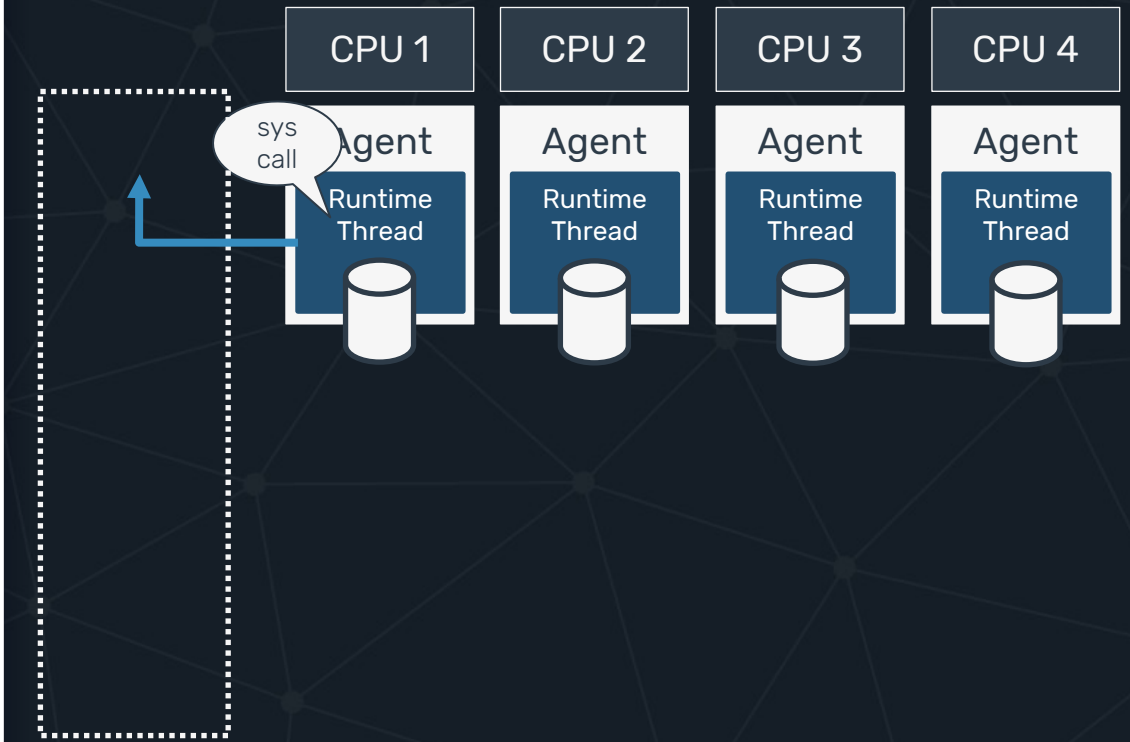
GOROUTINES & I/O

DEALING WITH KERNEL CALLS

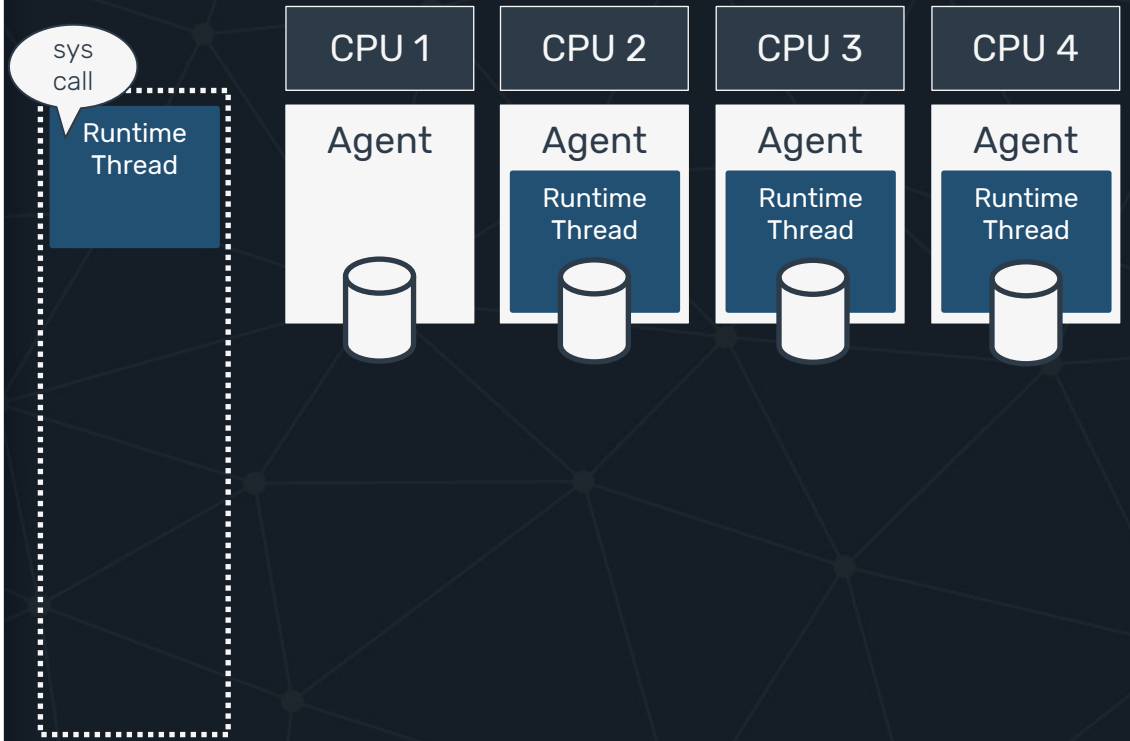
GOROUTINES I/O CALLS



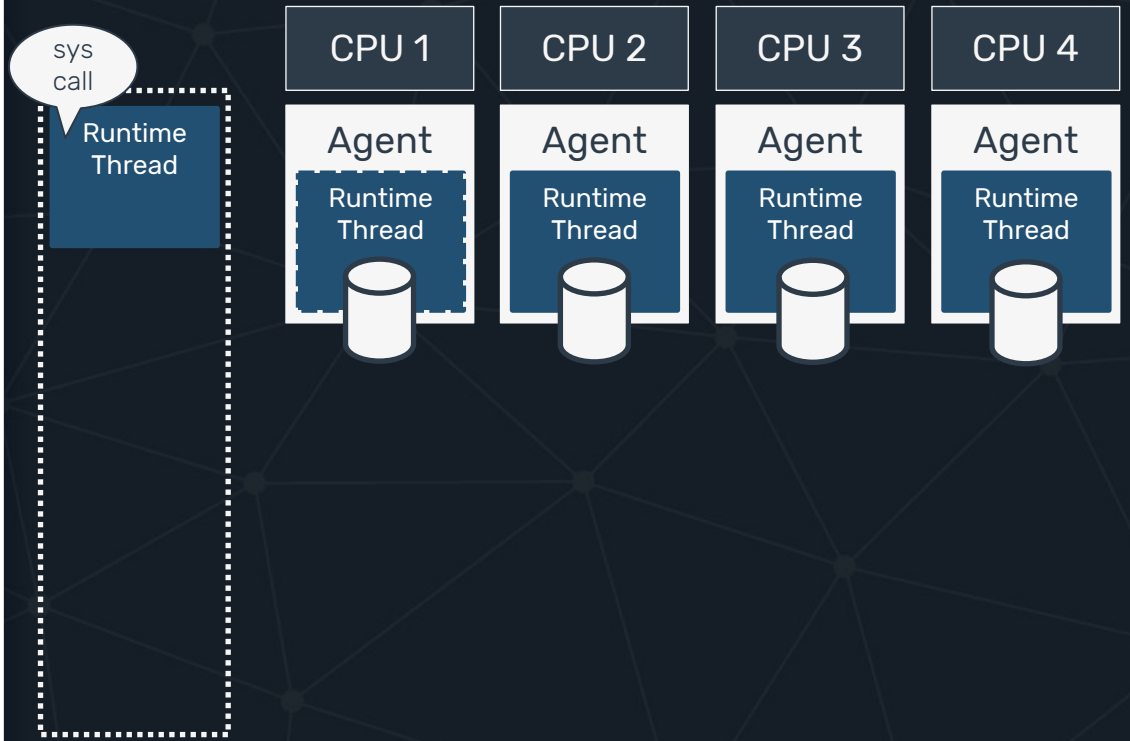
GOROUTINES I/O CALLS



GOROUTINES I/O CALLS



GOROUTINES I/O CALLS



SCHEDULERS

Preemptive vs. Cooperative

PREEMPTIVE

Scheduler is in charge of execution. Coroutines can be preempted.

COOPERATIVE

Scheduler depends on coroutines to return control to scheduler.

SCENARIO #2

CREATE AN EXCEL FILE

BUILDING AN EXCEL FILE

```
app.get('/valuations.xlsx', async (req, res) => {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    var valuations = await getValuations(req.params.id);
    for (let valuation in valuations) {
        worksheet.addRow(valuation);
    }
    await workbook.xlsx.write(res);
});
```

BUILDING AN EXCEL FILE

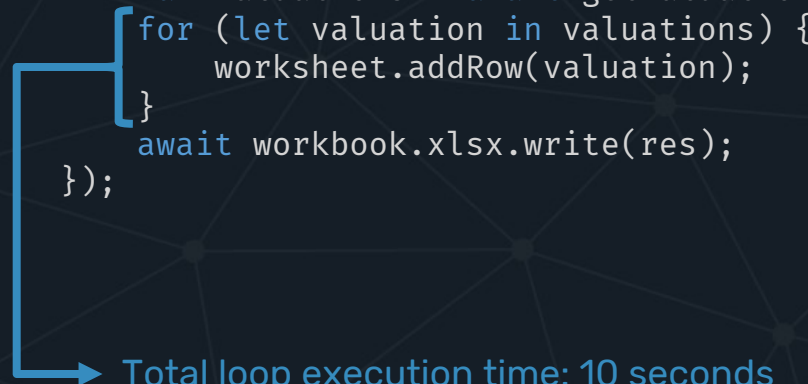
```
app.get('/valuations.xlsx', async (req, res) => {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    var valuations = await getValuations(req.params.id);
    for (let valuation in valuations) {
        worksheet.addRow(valuation);    // 500µs
    }
    await workbook.xlsx.write(res);
});
```

BUILDING AN EXCEL FILE

```
app.get('/valuations.xlsx', async (req, res) => {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    var valuations = await getValuations(req.params.id);
    for (let valuation in valuations) { // 20,000 items
        worksheet.addRow(valuation);    // 500µs
    }
    await workbook.xlsx.write(res);
});
```

BUILDING AN EXCEL FILE

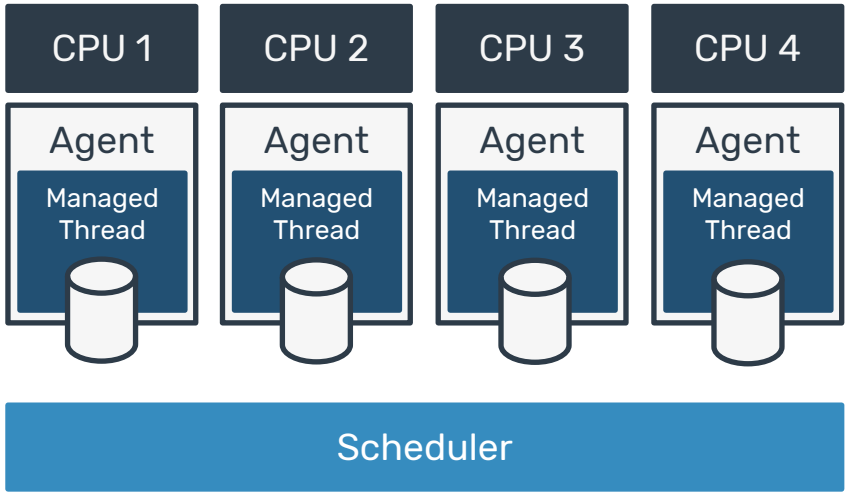
```
app.get('/valuations.xlsx', async (req, res) => {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    var valuations = await getValuations(req.params.id);
    for (let valuation in valuations) { // 20,000 items
        worksheet.addRow(valuation);    // 500µs
    }
    await workbook.xlsx.write(res);
});
```



HOW TO DEAL WITH LONG RUNNING TASKS?

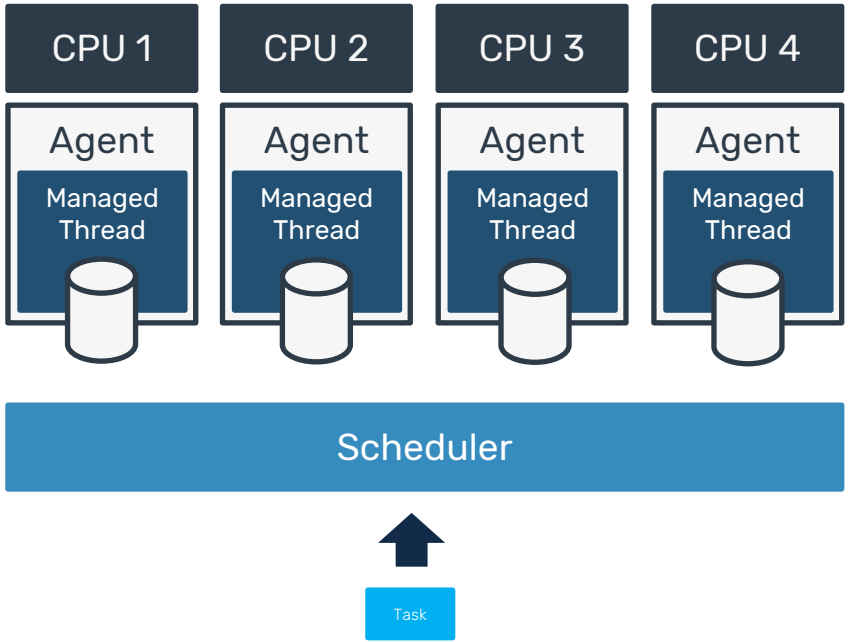
LONG RUNNING TASK

SEPARATE THREAD



LONG RUNNING TASK

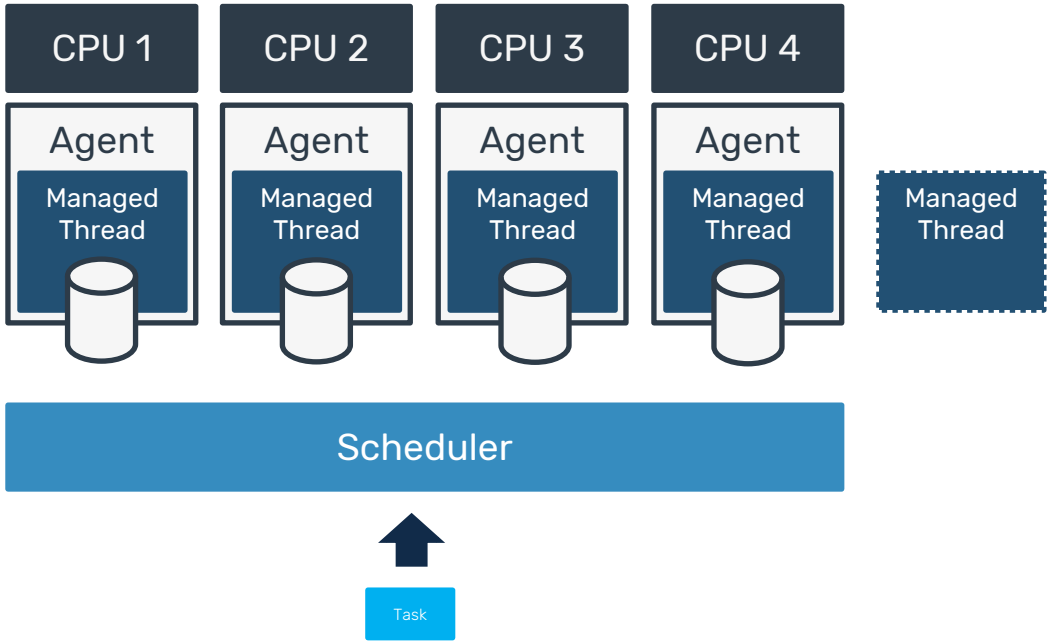
SEPARATE THREAD




```
Task.Factory.StartNew(DownloadExcel, TaskCreationOptions.LongRunning)
```

LONG RUNNING TASK

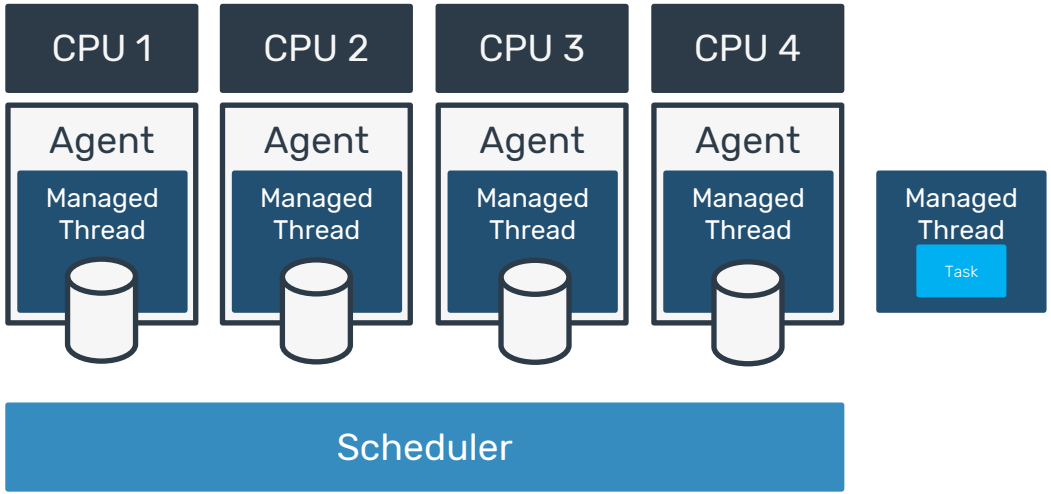
SEPARATE THREAD




```
Task.Factory.StartNew(DownloadExcel, TaskCreationOptions.LongRunning)
```

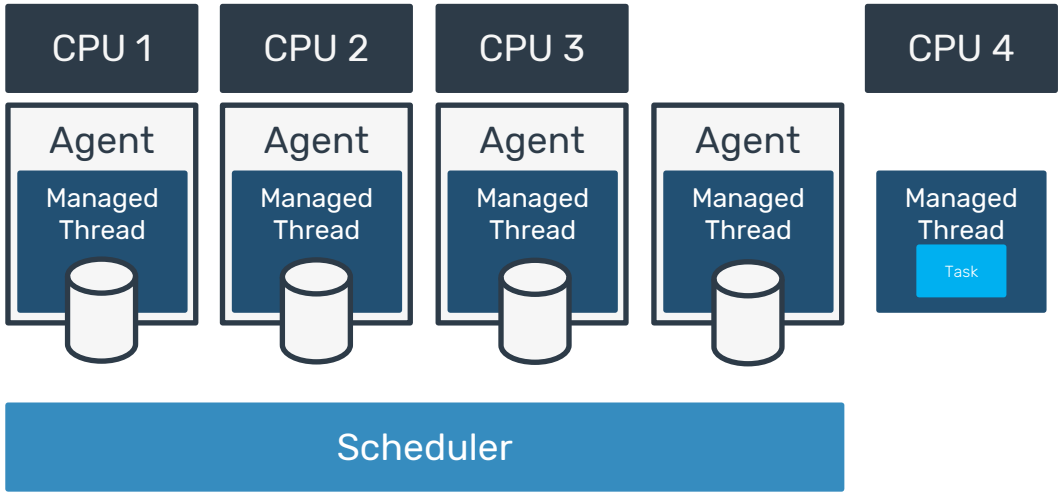
LONG RUNNING TASK

SEPARATE THREAD



LONG RUNNING TASK

SEPARATE THREAD



INTRODUCING INTERRUPTION POINTS

```
function park() {
    return new Promise((resolve) => setTimeout(resolve, 0));
}

app.get('/valuations.xlsx', async (req, res) => {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    var valuations = await getValuations(req.params.id);
    var i = 0;
    for (let valuation in valuations) { // 20,000 items
        if ((i++) % 100 === 0) {
            await park();
        }
        worksheet.addRow(valuation);    // 500µs
    }
    await workbook.xlsx.write(res);
});
```

PREEMPTIVE SCHEDULER

step-based / time-based

STEP-BASED PREEMPTION

```
% hello world program
-module(helloworld).
-export([start/0, write/1]).

write([]) -> ok;
write([File|T]) ->
    io:fwrite("Writing a file ~p~n", File),
    write(T).

start() ->
    Files = ["A", "B", "C"],
    write(Files).
```

STEP-BASED PREEMPTION

Reduction counter under the hood

```
% hello world program
-module(helloworld).
-export([start/0, write/1]).

write([], Reductions) -> ok;
write([File|T], Reductions) ->
    % if Reductions == 0 then yield()
    io:fwrite("Writing a file ~p~n", File),
    write(T, Reductions-1).

start() ->
    Files = ["A", "B", "C"],
    write(Files, 2000).
```

PREEMPTIVE

- *OS threads*
- *Go goroutines*
- *Erlang processes*

COOPERATIVE

- *JavaScript promises*
- *.NET Task Parallel Library*
- *Java/Scala Futures*

COROUTINES

eager vs. lazy

```
static async Task Run()
{
    var sw = new Stopwatch();
    sw.Start();

    var t1 = Task.Delay(TimeSpan.FromSeconds(5));
    var t2 = Task.Delay(TimeSpan.FromSeconds(5));

    await t1;
    await t2;

    Console.WriteLine(
        $"Time passed: {sw.ElapsedMilliseconds}ms");
}
```

```
let run () = async {
    let sw = Stopwatch()
    sw.Start()

    let t1 = Async.Sleep(5000)
    let t2 = Async.Sleep(5000)

    do! t1
    do! t2

    printfn "Time passed: %ims" sw.ElapsedMilliseconds
}
```

```
static async Task Run()
{
    var sw = new Stopwatch();
    sw.Start();

    var t1 = Task.Delay(TimeSpan.FromSeconds(5));
    var t2 = Task.Delay(TimeSpan.FromSeconds(5));

    await t1;
    await t2;

    Console.WriteLine(
        $"Time passed: {sw.ElapsedMilliseconds}ms");
}
```

5 seconds

```
let run () = async {
    let sw = Stopwatch()
    sw.Start()

    let t1 = Async.Sleep(5000)
    let t2 = Async.Sleep(5000)

    do! t1
    do! t2

    printfn "Time passed: %ims" sw.ElapsedMilliseconds
}
```

10 seconds

EAGER

```
static async Task Run()
{
    var sw = new Stopwatch();
    sw.Start();

    var t1 = Task.Delay(TimeSpan.FromSeconds(5));
    var t2 = Task.Delay(TimeSpan.FromSeconds(5));

    await t1;
    await t2;

    Console.WriteLine(
        $"Time passed: {sw.ElapsedMilliseconds}ms");
}
```

5 seconds

LAZY

```
let run () = async {
    let sw = Stopwatch()
    sw.Start()

    let t1 = Async.Sleep(5000)
    let t2 = Async.Sleep(5000)

    do! t1
    do! t2

    printfn "Time passed: %ims" sw.ElapsedMilliseconds
}
```

10 seconds

COROUTINES

stackless vs. stackful

OLD CALLBACK API

```
app.get('/valuations.xlsx', function (req, res, next) {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    getValuations(req.params.id, function (err, valuations) {
        for (let valuation in valuations) {
            worksheet.addRow(valuation);
        }
        workbook.xlsx.write(res, function (err) {
            next();
        });
    });
});
```

STACKLESS COROUTINES

OLD PROMISE API

```
app.get('/valuations.xlsx', (req, res) => {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    return getValuations(req.params.id)
        .then((valuations) => {
            for (let valuation in valuations) {
                worksheet.addRow(valuation);
            }
            return workbook.xlsx.write(res);
        });
});
```

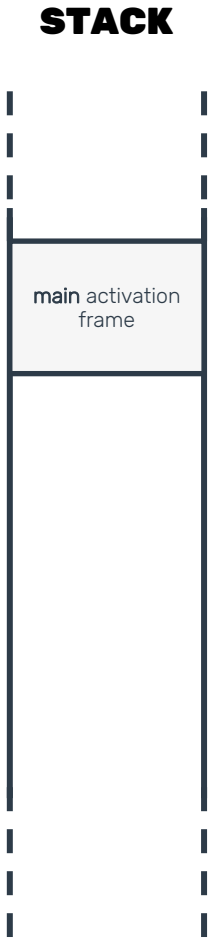
STACKLESS COROUTINES

STACKLESS COROUTINES

```
app.get('/valuations.xlsx', async (req, res) => {
    var workbook = new Excel.Workbook();
    var worksheet = workbook.addWorksheet("Valuations");
    var valuations = await getValuations(req.params.id);
    for (let valuation in valuations) {
        worksheet.addRow(valuation);
    }
    await workbook.xlsx.write(res);
});
```

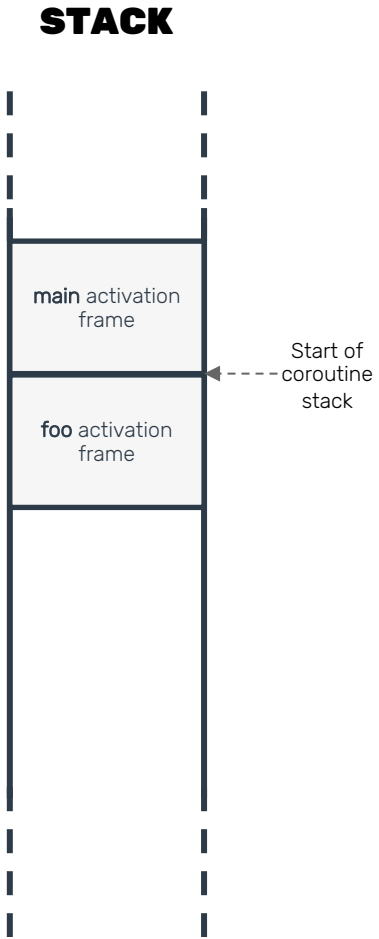
PROBLEM OF FUNCTION COLOURING

STACKFUL COROUTINES



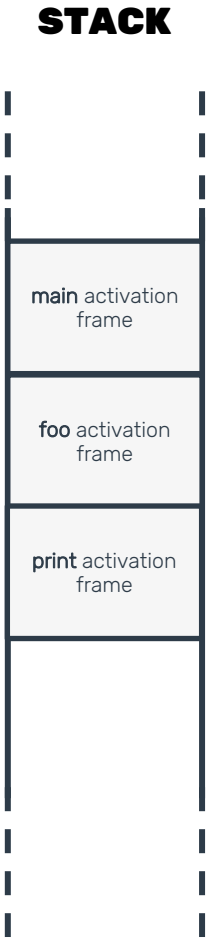

```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES



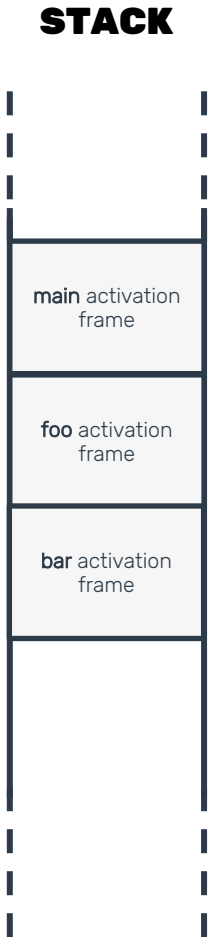

```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES



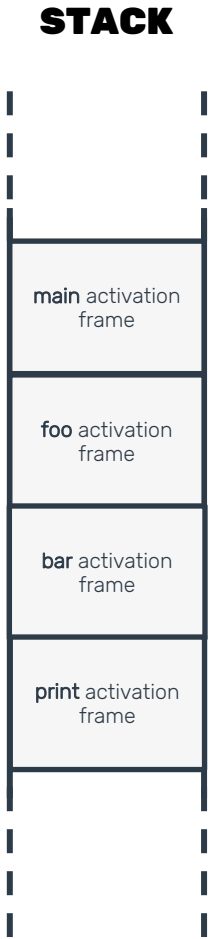

```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES



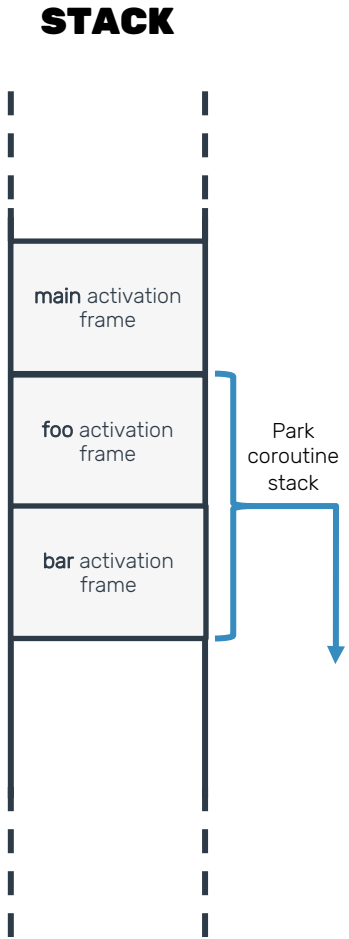

```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES



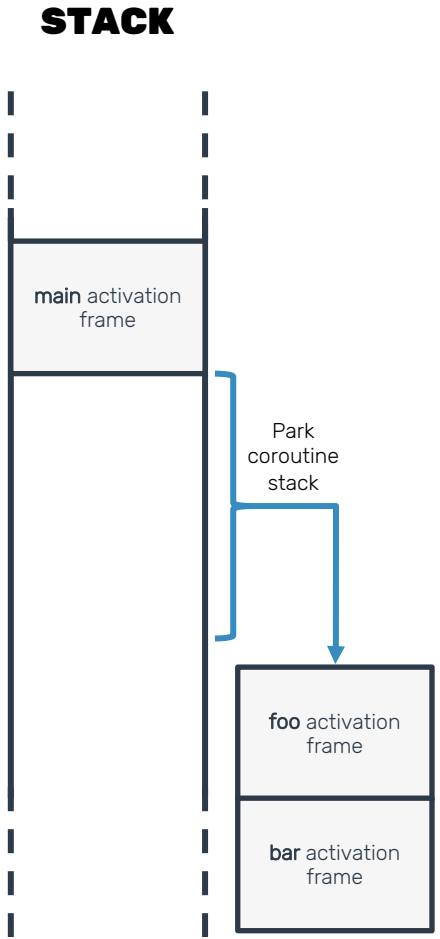

```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES



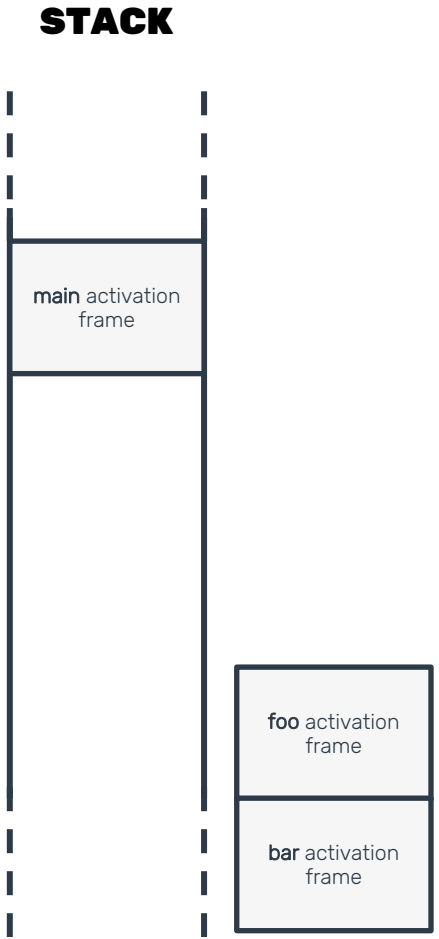

```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES



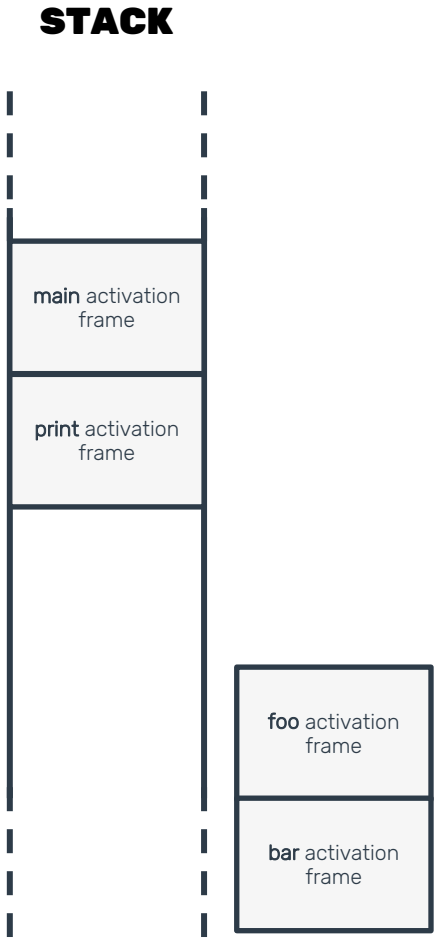

```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES



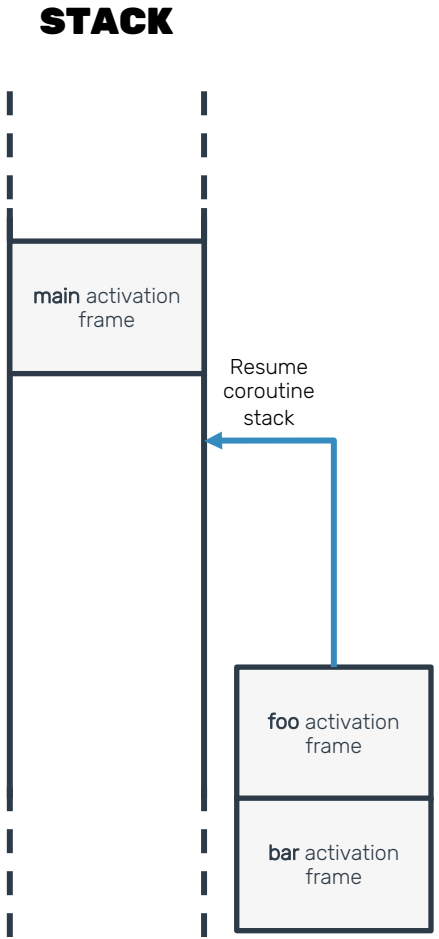

```lua
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES




```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES

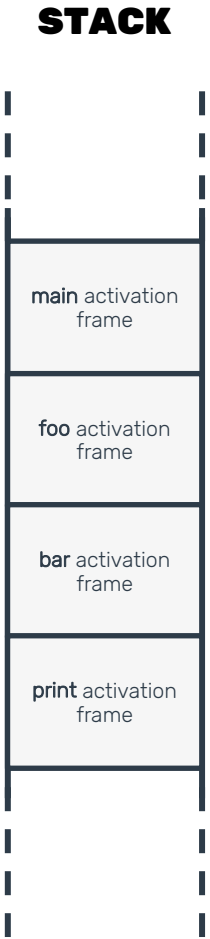



```lua
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES


```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

STACKFUL COROUTINES




```
function foo(a, b)
   print("foo", a)
   bar("foo", b)
end

function bar(caller, x)
   print(“bar: ", caller)
   coroutine.yield()
   print(“bar", x)
end

co = coroutine.create(foo)
coroutine.resume(co, 1, 2)
print(“main”)
coroutine.resume(co)
```

WHAT IS THE CORRECT STACK SIZE FOR EACH COROUTINE?

STACKFUL

- *Go goroutines*
- *LUA coroutines*
- *(soon) Java Loom project*

STACKLESS

- *JavaScript promises*
- *.NET Task Parallel Library*
- *Java/Scala Futures*

STACKLESS COROUTINES AND YIELD GENERATORS

ASYNC/AWAIT

```
async Task WaitAndPrint()
{
    Console.WriteLine("Starting at " + DateTime.Now);
    await Task.Delay(100);
    Console.WriteLine("Ending at " + DateTime.Now);
}
```

UNITY COROUTINES

```
IEnumerable WaitAndPrint()
{
    print("Starting at " + Time.time);
    yield return new WaitForSeconds(0.1f);
    print("Ending at " + Time.time);
}
```

INTEGRATING TPL AND LINQ

```
public static class TaskExtensions
{
    public static Task<T3> SelectMany<T1, T2, T3>(
        this Task<T1> task,
        Func<T1, Task<T2>> binding,
        Func<T1, T2, T3> combine)
    {
        var tcs = new TaskCompletionSource<T3>();
        task.ContinueWith(t1 =>
        {
            if (t1.IsFaulted) tcs.SetException(t1.Exception);
            else if (t1.IsCanceled) tcs.SetCanceled();
            else binding(t1.Result).ContinueWith(t2 =>
                {
                    if (t2.IsFaulted) tcs.SetException(t2.Exception);
                    else if (t2.IsCanceled) tcs.SetCanceled();
                    else tcs.SetResult(combine(t1.Result, t2.Result));
                });

        });
        return tcs.Task;
    }
}
```

```
public static Task Rename(int userId, string name)
{
    return from user in GetUser(userId)
           from _ in SaveUser(user with { Name = name })
           select 0;
}
```

BEHIND ASYNC/AWAIT

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

WHAT YOU SEE

```
static class Program {
    static async Task Foo(int a, int b) {
        Console.WriteLine("Foo", a);
        await Task.Yield();
        Console.WriteLine("Foo", b);
    }
}
```

WHAT YOU DON'T SEE

```
struct Foo__0 : IStateMachine {
    AsyncTaskMethodBuilder builder;
    int state;
    int a;
    int b;

    public void MoveNext() {
        switch (this.state) {
            case -1:
                Console.WriteLine("Foo", this.a);
                var awaiter = Taks.Yield().GetAwaiter();
                this.state = 0;
                this.builder.AwaitUnsafeOnCompleted(ref awaiter, ref this);
                // ^ awaiter.OnComplete(() =>
                //       ThreadPool.QueueUserWorkItem(this.MoveNext));
                return;
            case 0:
                Console.WriteLine("Foo", this.b);
                builder.SetResult();
                return;
        }
    }
}
static class Program {
    static Task Foo(int a, int b)        {
        var builder = AsyncTaskMethodBuilder.Create();
        var stateMachine = new Foo__0();
        // initialize state machine fields
        stateMachine.a = a;
        stateMachine.b = b;
        stateMachine.state = -1;
        stateMachine.builder = builder;
        builder.Start(ref stateMachine);
        // ^ similar to: ThreadPool.QueueUserWorkItem(this.MoveNext);
        return builder.Task;
    }
}
```

AWAITER PATTERN

Custom awaitable objects

```
public readonly struct PromiseAwaiter<T> : INotifyCompletion
{
    private readonly Promise<T> promise;

    public PromiseAwaiter(Promise<T> promise)
    {
        this.promise = promise;
    }

    #region mandatory awaiter methods

    public bool IsCompleted => promise.IsCompleted;

    public T GetResult() => promise.Result;

    public void OnCompleted(Action continuation) => promise.RegisterContinuation(continuation);

    #endregion
}
```

ASYNC METHOD BUILDER

Custom async methods

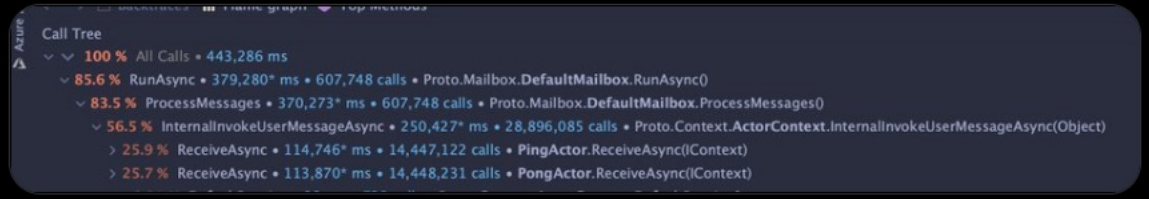
```
public struct PromiseAsyncMethodBuilder<T>
{
    private Promise<T>? promise;

    #region mandatory methods for async state machine builder

    public static PromiseAsyncMethodBuilder<T> Create() => default;

    public Promise<T> Task => promise ??= new Promise<T>();

    public void SetException(Exception e) => Task.TrySetException(e);

    public void SetResult(T result) =>  Task.TrySetResult(result);

    public void AwaitOnCompleted<TAwaiter, TStateMachine>(ref TAwaiter awaiter, ref TStateMachine stateMachine)
        where TAwaiter : INotifyCompletion
        where TStateMachine : IAsyncStateMachine { }

    public void AwaitUnsafeOnCompleted<TAwaiter, TStateMachine>(ref TAwaiter awaiter, ref TStateMachine stateMachine)
        where TAwaiter : ICriticalNotifyCompletion
        where TStateMachine : IAsyncStateMachine { }

    public void Start<TStateMachine>(ref TStateMachine stateMachine) where TStateMachine : IAsyncStateMachine { }

    public void SetStateMachine(IAsyncStateMachine stateMachine) { }

    #endregion
}
```

Roger Johansson
@RogerAlsing

There, I did it. I got rid of all async state machines for the entire actor receive pipeline.
Only activating state-machines incase there are non-completed tasks.

Throughput went from 47 mln msg/sec to 73 mln msg/sec on my small laptop.

Przetłumacz Tweeta



6:19 PM · 3 kwi 2021 · Twitter Web App

SUMMARY

REFERENCES

- Custom AsyncMethodBuilder “docs”: https://github.com/dotnet/roslyn/blob/master/docs/features/task-types.md
- Building custom (affine) thread pool: https://bartoszsypytkowski.com/thread-safety-with-affine-thread-pools/
- What color is your function?: https://journal.stuffwithstuff.com/2015/02/01/what-color-is-your-function/
- How Go scheduler works: https://www.youtube.com/watch?v=-K11rY57K7k
- Thread per core architecture: https://www.datadoghq.com/blog/engineering/introducing-glommio/

THANK YOU