1. What does the compiler do when it encounters the await keyword?

My answer: delegetas the “async” code to the thread pool, thread pool then yields one of the threads (might be a new one, might not be), stages the current state of the state machine (held in the Task object), returns the Task object to the thread calling context whose state is at that moment “captured” so the execution may return to it)

In essence the execution will “continue” (continuation context) right after the calling context was “captured and suspended” by means of a callback (through the Task object)

1. What is ConfigureAwait(false)

My answer: this one is relevant only in pre-netcore contexts, and for “UI thread” apps like webforms, where you caller (thread) is the one that the execution will return to after the async task has been completed, the default is ConfigureAwait(true) meaning - return to the exact calling thread, this can cause deadlocks/not responding because the returning context encounters a blocked calling thread and is unable to return, whilst the calling context is stuck waiting for the called context to return (deadlock)

1. How are errors handled in async await?

My answer: the compiler internally creates a try catch block and uses the Task object (which is a form of a promise) to hold the error (if it were to happen) so the calling context can evaluate it, if theres is no Task object, no errors will be caught (which is a classic junior level error)

This is the what they wanted to have a discussion about (the relevant compiler output)

[CompilerGenerated]

private sealed class <GetValueAsync3>d\_\_2 : IAsyncStateMachine

{

public int <>1\_\_state;

public AsyncTaskMethodBuilder<int> <>t\_\_builder;

public C <>4\_\_this;

private int <>s\_\_1;

private TaskAwaiter<int> <>u\_\_1;

private void MoveNext()

{

int num = <>1\_\_state;

int result;

try

{

TaskAwaiter<int> awaiter;

if (num != 0)

{

awaiter = Task.FromResult(42).GetAwaiter();

if (!awaiter.IsCompleted)

{

num = (<>1\_\_state = 0);

<>u\_\_1 = awaiter;

<GetValueAsync3>d\_\_2 stateMachine = this;

<>t\_\_builder.AwaitUnsafeOnCompleted(ref awaiter, ref stateMachine);

return;

}

}

else

{

awaiter = <>u\_\_1;

<>u\_\_1 = default(TaskAwaiter<int>);

num = (<>1\_\_state = -1);

}

<>s\_\_1 = awaiter.GetResult();

result = <>s\_\_1;

}

catch (Exception exception)

{

<>1\_\_state = -2;

<>t\_\_builder.SetException(exception);

return;

}

<>1\_\_state = -2;

<>t\_\_builder.SetResult(result);

}

void IAsyncStateMachine.MoveNext()

{

//ILSpy generated this explicit interface implementation from .override directive in MoveNext

this.MoveNext();

}

[DebuggerHidden]

private void SetStateMachine(IAsyncStateMachine stateMachine)

{

}

void IAsyncStateMachine.SetStateMachine(IAsyncStateMachine stateMachine)

{

//ILSpy generated this explicit interface implementation from .override directive in SetStateMachine

this.SetStateMachine(stateMachine);

}

}

1. How exactly does it return the captured context (compiler output inspection)?

My answer: did not know this one

All I knew was it had captured the “before async” context somehow, embedded the info somewhere (Task object?) and will use this info to return to the exact line where it needs to

Should look it up at some point I guess…

1. How to create a stack overflow condition using for loop?

My answer: concatenate strings, you will end up having a stack overflow

* 1. Why? My answer: reference type
  2. How to fix this?

My answer: use stringbuilder

1. Why is a string class/object made immutable?

My answer: design choice as strings are most often mutated

1. Can we iterate over a set of data without using window functions?

My answer: yes, via cursors

1. How do cursors work (implemented on db engine)?

My answer: didn't know, said I didn't care much how they work, I just use them and they work

need to look it up

1. What is the core difference between triggers on MSSQL vs Oracle

My answer: did not know this one, said I just use triggers, I don't care how they work internally

Looked it up later: Oracle (as most other vendors) has a different trigger concept as MS SQL Server. MS SQL (such as Sybase) has a set based approach. Rows that are affected by a data modification (insert, update, delete) are stored in the inserted and deleted tables. A regular DML trigger in MS SQL is always executed after the statement. The before image is stored in the deleted table, the after image in the inserted table. Both can be accessed from within the trigger. It is also possible to join the inserted and deleted table and use them to update the table on which the trigger was executed.

In Oracle there are before and after triggers and a trigger can be defined to be executed per row or per statement (there are also compound triggers that can have a section for per row and per statement). Before per statement triggers are executed before the statement (insert, update, delete) is executed and only have access to the table before the modification. After per statement triggers are executed after the statement has completed (which is why they are sometimes called deferred, which is a bit misleading as deferred in SQL usually means that an execution is deferred till commit in a transaction that may consist of more than one statement) and only have access to the table after the modification.

1. You have database (single instance), you want to scale it, the callers (rest api apps) want to get immediate results and you \*have\* to make it work.

The interviewers are well aware of eventual consistency problems and the CAP theorem, still you have to make it work, how would you do it?

**So both immediate consistency and scale out are a must.**

My answer: shard the data, create n clusters where each cluster has one master and n read slaves, and each shard holds a portion (shard) of data

1. How is the data organized in a clustered index as opposed to non clustered?

My answer: it's always a IOT (index organized table) hit in that case, without using a heap, this is considered fastest

1. Elaborate on that (IoT tables and heap tables)

My answer: couldn't, really have no idea what happens at that level (really low level)

need to look it up

1. When do we use heap tables?

My answer: tables with one index only are best implemented as clustered indexes or index-organized tables.. tables with more indexes can often benefit from heap tables

need to look it up

1. Why?

My answer: didn't know, couldn't explain

Expected answer: heap tables have the benefit of providing a stationary master copy that can be easily referenced

Need to look up further.

1. How do we turn a clustered index into a heap table

My answer: dropping a clustered index transforms the table into a heap table

1. What is ValueTask, why do we have it?

My answer: didn't know (read a bit on ValueTasks, but haven't remembered anything)

Answer: we can utilize ValueTask<T> to **avoid allocations when methods complete synchronously**

Explain - couldn't, need to look it up, has something to do with GC probably there are no allocations so there's less of an overhead

1. Why would we use Span<>?

My answer: didn't know (did read something on the topic at some point, couldn't remember)

Answer: need to look it up

1. Explain double lock pattern in regards to processor architecture?

My answer: did the best I could, haven't been using locks in a really long time, the lock is a keyword that enables, well locking a reference type (usually its a private readonly object) to “serialize” access to a mutable property to ensure it behaves nicely in a multithreaded scenario (where you have multiple threads mutating the same property), the access is anchored (blocked/released) via the object, sometimes we have to use 2 locks due to processor optimizations that may “go around” a single lock (couldn't explain further)

lock statement is syntactic sugar for a try/finally statement with Monitor.Enter and Monitor.Exit

Expected answer: there is no double lock but there is a double boolean check one implements manually to check if a lock has been already set/evaluated to prevent fringe case race conditions on certain processor architectures

1. Why can't we use value type as a lock object?

My answer: boxing would happen automatically and you would get a **different object each time**, preventing lock

1. Explain semaphore structure?

My answer: structure to manually control threading to avoid deadlocks, if implemented correctly

Expected answer:

A semaphore with a capacity of one is similar to a Mutex or lock, except that the semaphore has no “owner” — **it’s thread-agnostic**. Any thread can call Release on a Semaphore, whereas with Mutex and lock, only the thread that obtained the lock can release it. Semaphores can be useful in limiting concurrency — preventing too many threads from executing a particular piece of code at once.

1. How is semaphore implemented in .net (low level)? - didn't know, did know about the lock keyword thou

My answer: the lock has “under the cover” a Monitor.Enter and Monitor.Exit (inside a finally block)

the semaphore on the other hand .. TBD need to look it up

1. Why do we have .GetAwaiter()?

My answer: so we can get a handle on internal “state machine” to read/interact with the current state of the “promise” structure that is Task

Proper answer (what they wanted to hear): Task.GetAwaiter().GetResult() is preferred over Task.Wait and Task.Result because it **propagates exceptions** rather than wrapping them in an AggregateException)

1. How is the stack overflow exception handled by the .net runtime, and how does it arrive at userland? - My answer: no idea
2. What is a channel construct in TPL

My answer: didn't know

This is the relevant code sample, special attention to ValueTask and Channel

using BenchmarkDotNet.Attributes;

using BenchmarkDotNet.Running;

using System.Threading.Channels;

using System.Threading.Tasks;

using System.Threading.Tasks.Dataflow;

[MemoryDiagnoser]

public class Program

{

static void Main() => BenchmarkRunner.Run<Program>();

private readonly Channel<int> \_channel = Channel.CreateUnbounded<int>();

private readonly BufferBlock<int> \_bufferBlock = new BufferBlock<int>();

[Benchmark]

public async Task Channel\_ReadThenWrite()

{

ChannelWriter<int> writer = \_channel.Writer;

ChannelReader<int> reader = \_channel.Reader;

for (int i = 0; i < 10\_000\_000; i++)

{

ValueTask<int> vt = reader.ReadAsync();

writer.TryWrite(i);

await vt;

}

}

[Benchmark]

public async Task BufferBlock\_ReadThenWrite()

{

for (int i = 0; i < 10\_000\_000; i++)

{

Task<int> t = \_bufferBlock.ReceiveAsync();

\_bufferBlock.Post(i);

await t;

}

}

}

1. How would you write a combinator?

My answer: didn't know

Answer (googled afterwards):

public static async ValueTask<ChannelReader<T>> WhenAny<T>(

ChannelReader<T> reader1, ChannelReader<T> reader2)

{

var cts = new CancellationTokenSource();

Task<bool> t1 = reader1.WaitToReadAsync(cts.Token).AsTask();

Task<bool> t2 = reader2.WaitToReadAsync(cts.Token).AsTask();

Task<bool> completed = await Task.WhenAny(t1, t2);

cts.Cancel();

return completed == t1 ? reader1 : reader2;

}

1. What is the latest C# you know and what are your thoughts on what if offers?

My answer: managed to get around the question by babbling about the new Record type (the immutable one)

1. What happens in runtime if GC is getting choked, how would you troubleshoot that?

My answer: someone is probably creating ton load of classes in rapid succession, probably in a for loop, that are not being collected

1. How does the GC decide if an object can be cleaned up?

My answer: root graphing then propagating gen0 -> gen2, gen0 get popped off the stack immediately, long lived ones survive all the way to gen2

1. Any fringe cases?

My answer: yes, the finalizer queue and finalizer thread where we can have objects “living a little bit longer” so that the finalizer can complete, does not work all that well combined with GC.Collect

1. What does the yield keyword signify?

My answer: Ienumerable implementation

1. What are statistics?

My answer: The query optimizer uses these to decide on the strategy

1. Explain?

My answer: didn't know

1. Are statistics row based or column based?

My answer: didn't know

Answer: they are **column** based need to look it up

1. Are null values calculated into statistics

My answer: didn't know

Answer: need to look it up

1. How are function based indexes shown in statistics

My answer: didn't know, the answer was: they are kept on TABLE level as VIRTUAL COLUMNS don't really know, nor care, why this is so, need to look it up

1. Are they ever renewed automatically by db?

My answer: didn't know (answer: it depends)

1. Do statistics auto-update when a new index is created

My answer: yes (but -> no)

1. On all databases?

My answer: yes (wrong, Oracle does **not** recalculate)

1. What is an index?

My answer: explained root, leaf nodes, page, b-tree, doubly linked list and why those as used, what problems to they solve (all boils down to how data is laid out on the disk level, need to be able to explain why leaf nodes are a linked list (for fast inserts)

1. What is a covering index?

My answer: an index that includes all the attributes from where part and the select

1. Does a covering index read data from the disk?

My answer: nope, that's their advantage, the data they need is already contained at the leaf level so no actual disk access is actually needed

1. Is there a limit?

My answer: Obviously, the size

* 1. what is the max allowed size?

My answer: Depends

* 1. for MSSQL?

My answer: didn't know, said 8000 bytes but that's the size of a page not the limit for a covering index - should I be memorizing these things? For Oracle, for MSSQL, for Postgresql and every other database etc.? Why?

1. What is a secondary index and why do we use it?

My answer: didn't know need to look it up

1. Does it (the secondary index) contain ROWID?

My answer: didn't know

Expected answer: contains only the primary key of the primary index, this might be Oracle specific though, need to read up on this

1. When would you use constructor injection (IoC/DI) and when property injection and why?

My answer: constructor injection is better because by the constructor signature you can dictate all required dependencies and the container will immediately tell you if any are not resolvable, makes coding easier, also convention, I read code faster if all dependencies are listed inside constructor signature

Expected answer: we usually advise people to use constructor injection for all mandatory collaborators and setter injection for all other properties. Again, constructor injection ensures all mandatory properties have been satisfied, and it is simply not possible to instantiate an object in an invalid state (not having passed its collaborators). In other words, when using constructor injection you do not have to use a dedicated mechanism to ensure required properties are set (other than normal Java mechanisms).

One of the arguments for not using constructor injection is the lack of argument names in constructors and the fact that these do not appear in the XML. I would argue that in **most** applications, this does not matter that much.