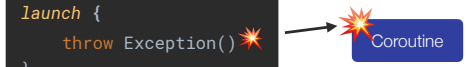
1) Exceptions can be handled directly in a Coroutine with a try-catch block. This way, the Coroutine does not complete exceptionally.

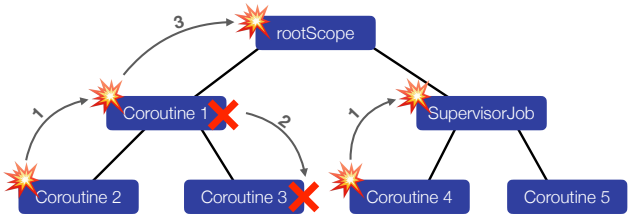
```
launch {
        try {
            throw Exception()
        } catch (exception: Exception) {
            println("Handled $exception")
        }
    }
}
```

2) If an exception is thrown in a Coroutine and not handled directly within the Coroutine with a try-catch block, the Coroutine completes exceptionally.

```
launch {
    throw Exception()
}
```

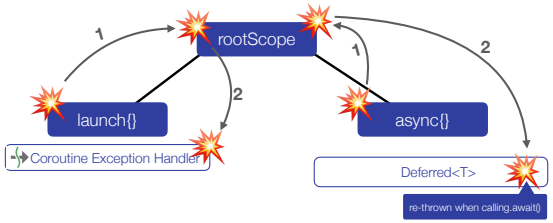


3) Then, the exception is propagated up the job hierarchy until it reaches either the RootScope or a SupervisorJob. While the exceptions travels upwards, parent coroutines fail too and sibling coroutines get cancelled.



4) When the root coroutine was started with launch{}, the exception will be passed to an installed CoroutineExceptionHandler.

When the root coroutine was started with async{}, the exception is encapsulated in the Deferred object.



5) coroutineScope{} re-throws uncaught exceptions of its child Coroutines and so we can handle them with a try/catch.

```
launch {
    try {
        coroutineScope {
            launch {
                throw Exception()
            }
        }
    } catch (exception: Exception) {
        println("Handled $exception")
    }
}
```

6) Keep in mind:

- When your Coroutine is not completing exceptionally, parent and sibling coroutines won't be canceled.
- Suspend functions can throw a CancellationException at any point and by catching them, the Coroutine will keep on running.