WHAT'S COROUTINE?

A coroutine can be thought of as an instance of ***suspendable computation***

i.e. the one that can suspend at some points and later resume execution possibly on another thread.

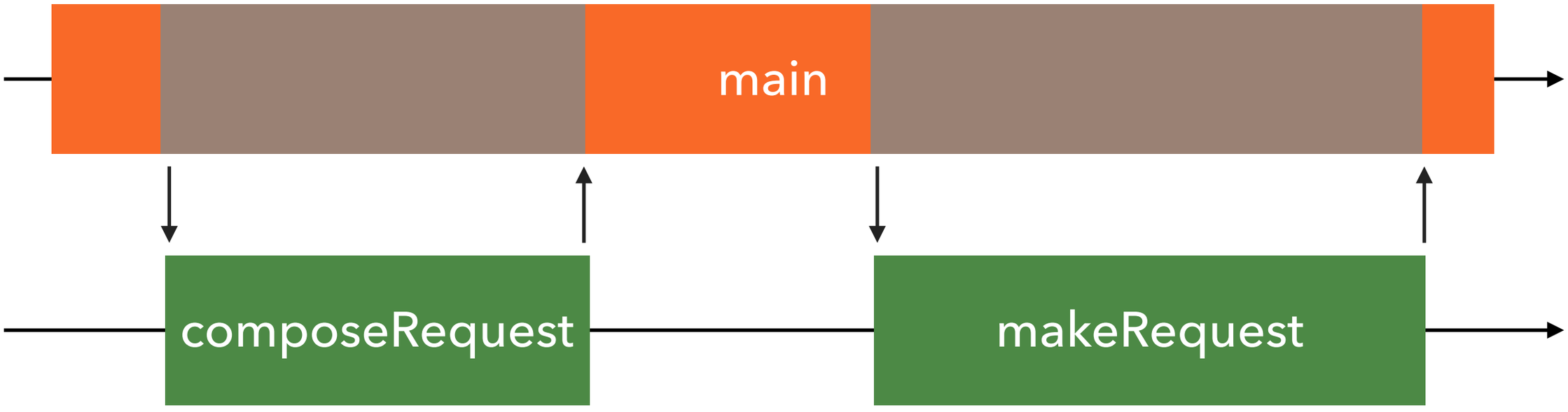
https://github.com/Kotlin/kotlin-coroutines/blob/master/kotlin-coroutines-informal.md#use-cases

THREAD-BLOCKING CODE SAMPLE

```
val request = composeRequest()
val post = makeRequest(request)
parsePost(post)

fun composeRequest(): Request = …

fun makeRequest(request: Request): Result = …
```

If it takes a long time?

If it takes a long time?

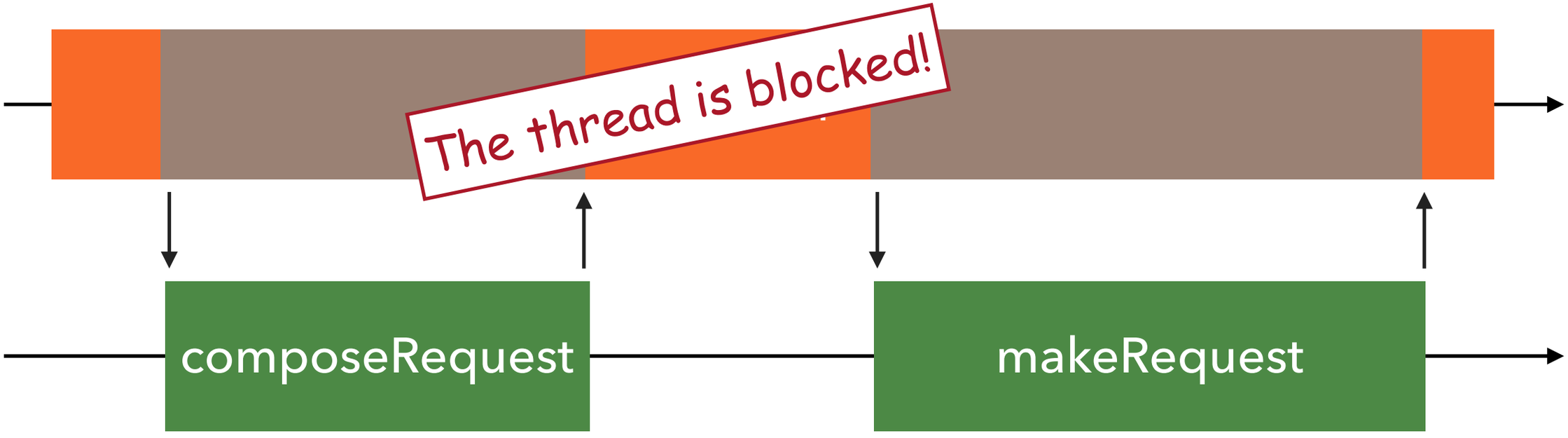


THREAD-BLOCKING CODE SAMPLE

```
val request = composeRequest()
val post = makeRequest(request)
parsePost(post)

fun composeRequest(): Request = …

fun makeRequest(request: Request): Result = …
```

If it takes a long time?

If it takes a long time?



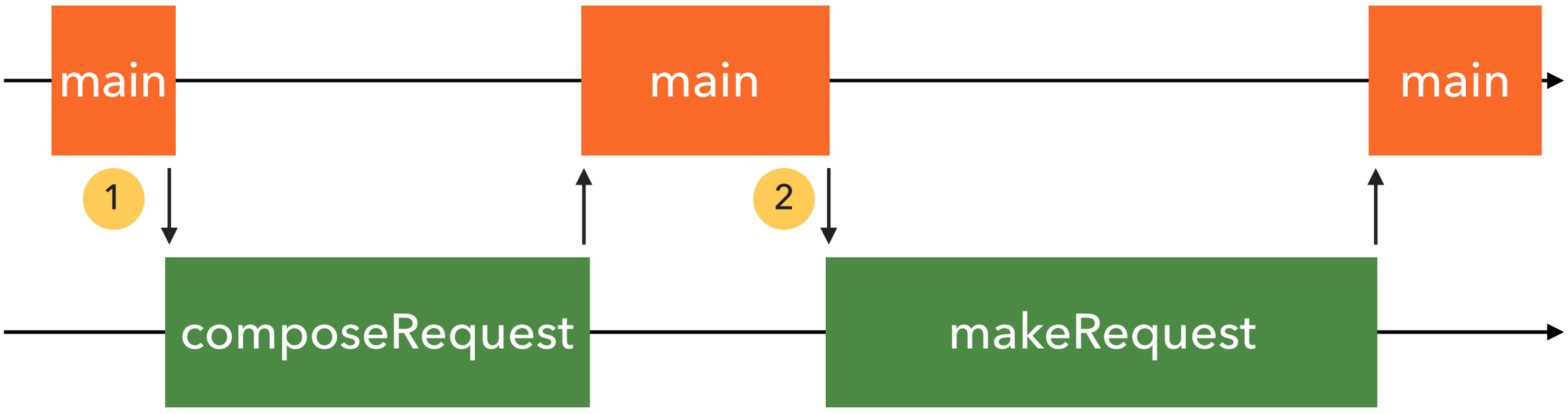
SUSPEND TO THE RESCUE

```
val request = composeRequest()   1
val post = makeRequest(request)  2
parsePost(post)

suspend fun composeRequest(): Request = …

suspend fun makeRequest(request: Request): Result = …
```

- The code suspends at the suspension points
- The code resumes when the return value is ready

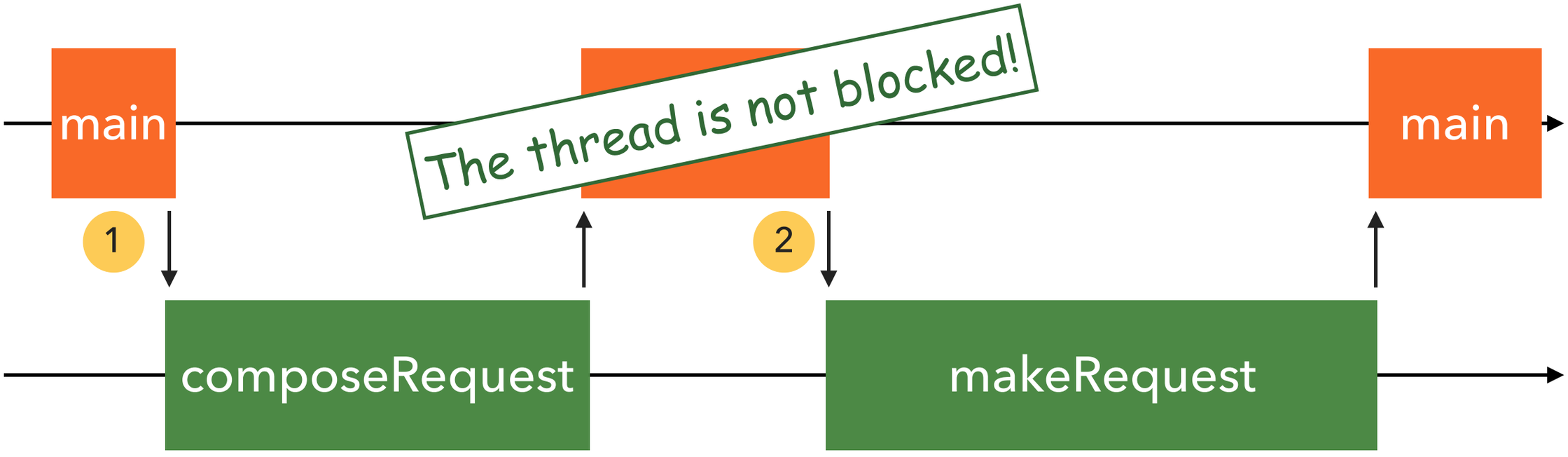
SUSPEND TO THE RESCUE

```
val request = composeRequest()   1
val post = makeRequest(request)  2
parsePost(post)

suspend fun composeRequest(): Request = …

suspend fun makeRequest(request: Request): Result = …
```

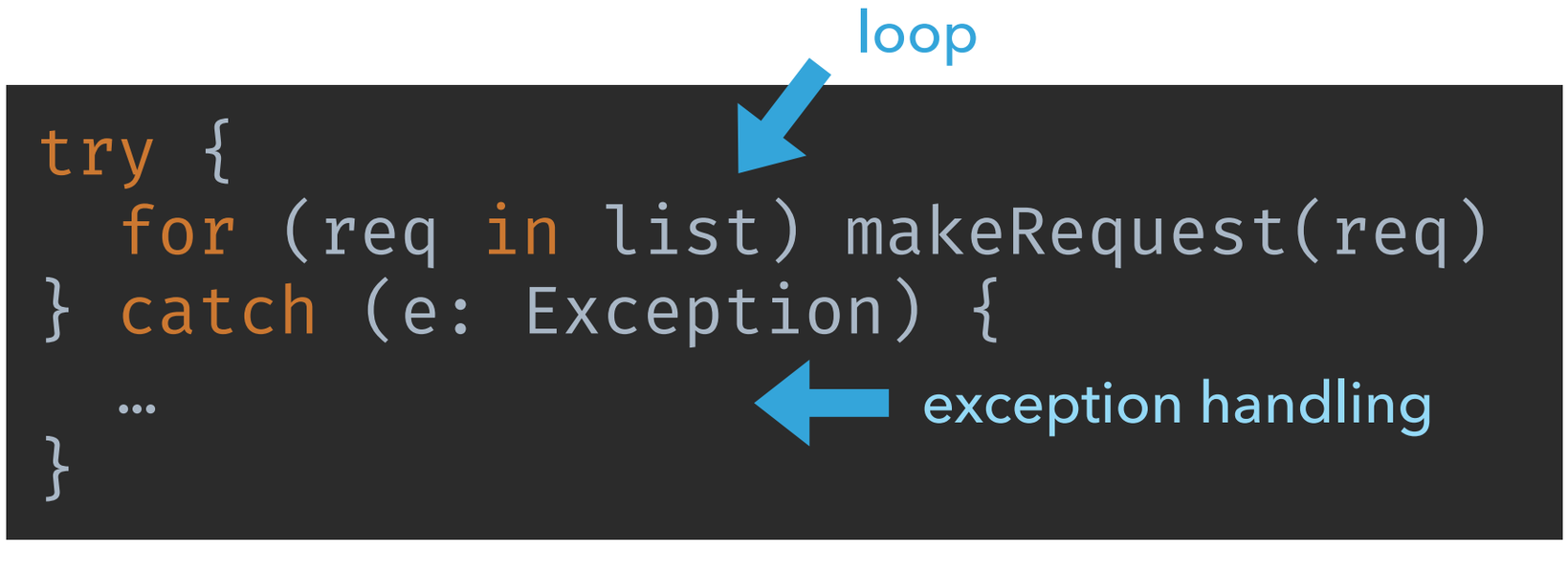


SUSPEND TO THE RESCUE

```
val request = composeRequest()   1
val post = makeRequest(request)  2
parsePost(post)

suspend fun composeRequest(): Request = …

suspend fun makeRequest(request: Request): Result = …
```



CODE CAN BE WRITTEN IN A REGULAR STYLE



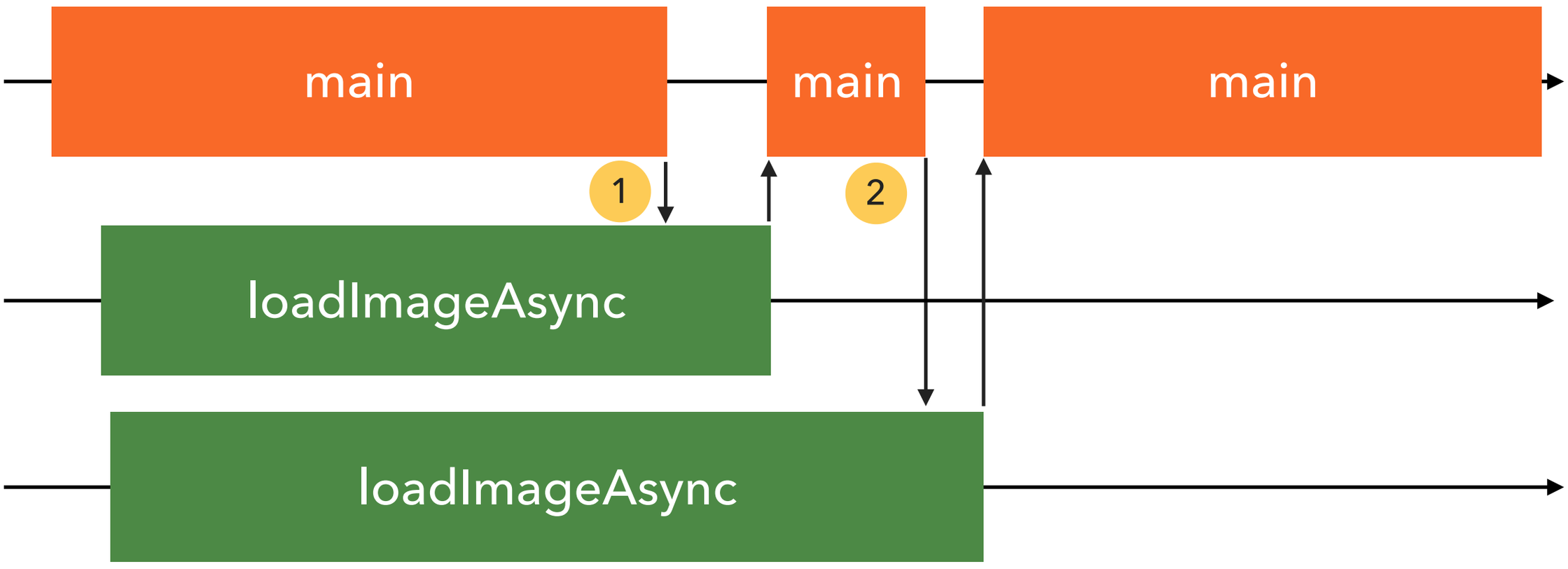
Applications 💼

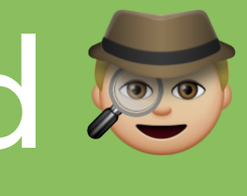
ASYNC/AWAIT

```
val first = loadImageAsync("green")
val second = loadImageAsync("red")
overlay(first.await(), second.await()) 1 2

fun loadImageAsync(name: String) = async { … }
```

ASYNC/AWAIT

```
val first = loadImageAsync("green")
val second = loadImageAsync("red")
overlay(first.await(), second.await()) 1 2

fun loadImageAsync(name: String) = async { … }
```



Coroutine under the hood 

SUSPEND COROUTINE

```
suspend fun <T> suspendCoroutine(
  block: (Continuation<T>) → Unit
): T
```

- When the method is called, it suspends the current coroutine
- The coroutine is resumed when resume() is called on the passed continuation

Scheme call/cc style!

SUSPEND COROUTINE

```
suspend fun <T> suspendCoroutine(
  block: (Continuation<T>) → Unit
): T
```

```
fun <T> doLongTask(cb: (T, Throwable) → Unit) {
}
```

```
suspendCoroutine { cont: Continuation<T> →
  doLongTask { result, exception →
    if (exception == null)
      cont.resume(result)
    else
      cont.resumeWithException(exception)
  }
}
```

SUSPEND COROUTINE

```
result = doLongTaskSuspend()
…
// do something with the result
```

```
suspend fun <T> doLongTaskSuspend() =
    suspendCoroutine { cont: Continuation<T> →
      doLongTask { result, exception →
        if (exception == null)
          cont.resume(result)
        else
          cont.resumeWithException(exception)
      }
    }
```

What is continuation? 🤔

CONTINUATION

```
val request = composeRequest()   1
val post = makeRequest(request)  2
parsePost(post)
```

CONTINUATION

```
val request = composeRequest()    1
val post = makeRequest(request)   2
parsePost(post)
```

Continuation passed to (1)

CONTINUATION

```
val request = composeRequest()    1
val post = makeRequest(request)   2
parsePost(post)
```

Continuation passed to (2)

- Kotlin builds a state machine for each coroutine
- A suspension point just adds a state to the state machine

No deep stacks and light-weight 🤦🏻‍♀️

COROUTINE BUILDER

- Suspending function only can be called from a suspending function or a coroutine
- Use coroutine builders to call suspending functions from a regular function
  - launch
  - runBlocking
  - async

COROUTINE BUILDER

```
fun postItem(item: Item) {
  launch(CommonPool) {
    val token = preparePost()
    val post = submitPost(token, item)
    processPost(post)
  }
}
```

WRAP UP

- Coroutine is like a light-weight thread
  - Can be suspended/resumed
- async/await, generator, channel is just library built on that