Concurrency & Network calls in an android application

Applikationsudvikling

BEHU 2025

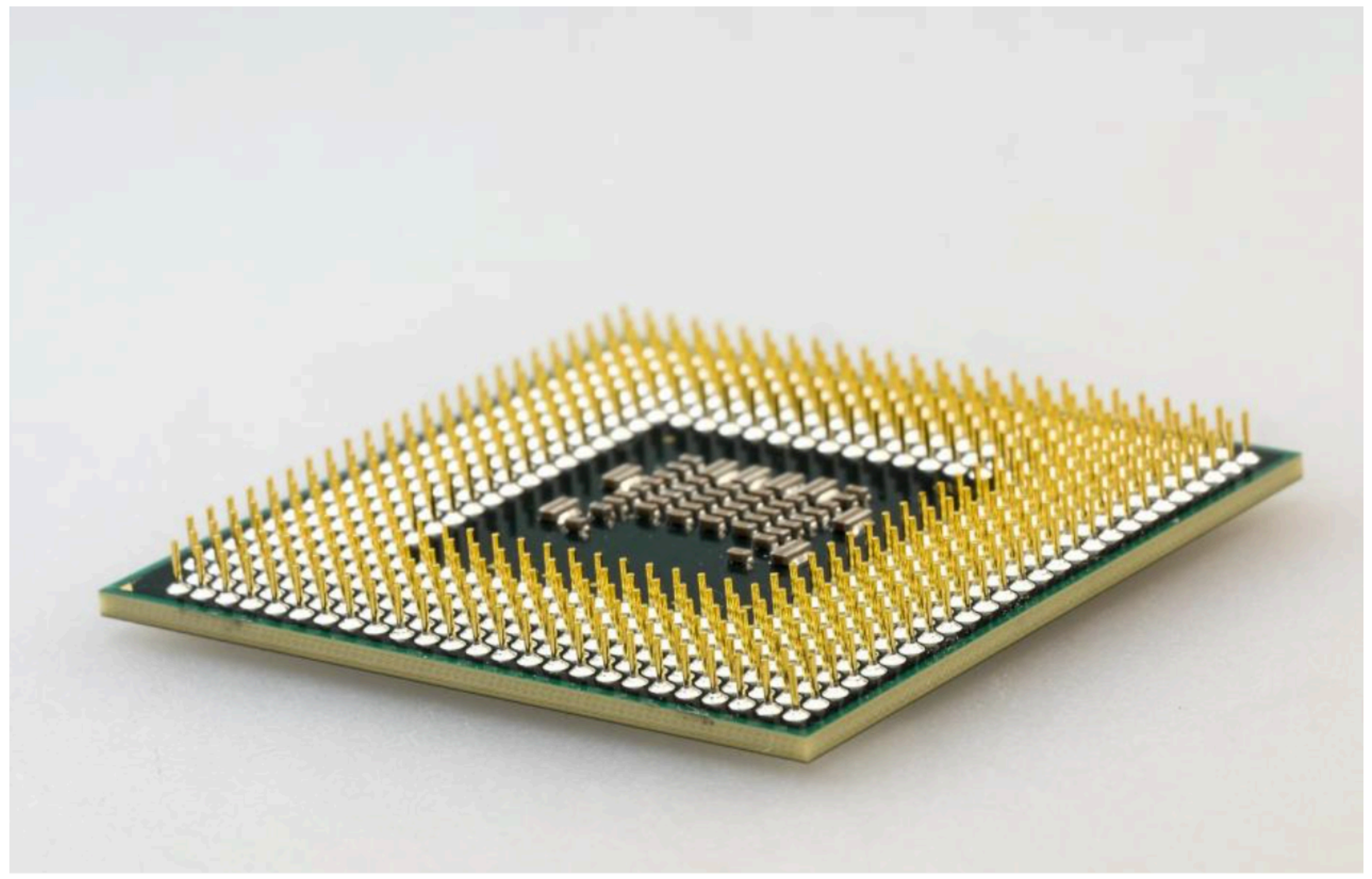
A blocking network call

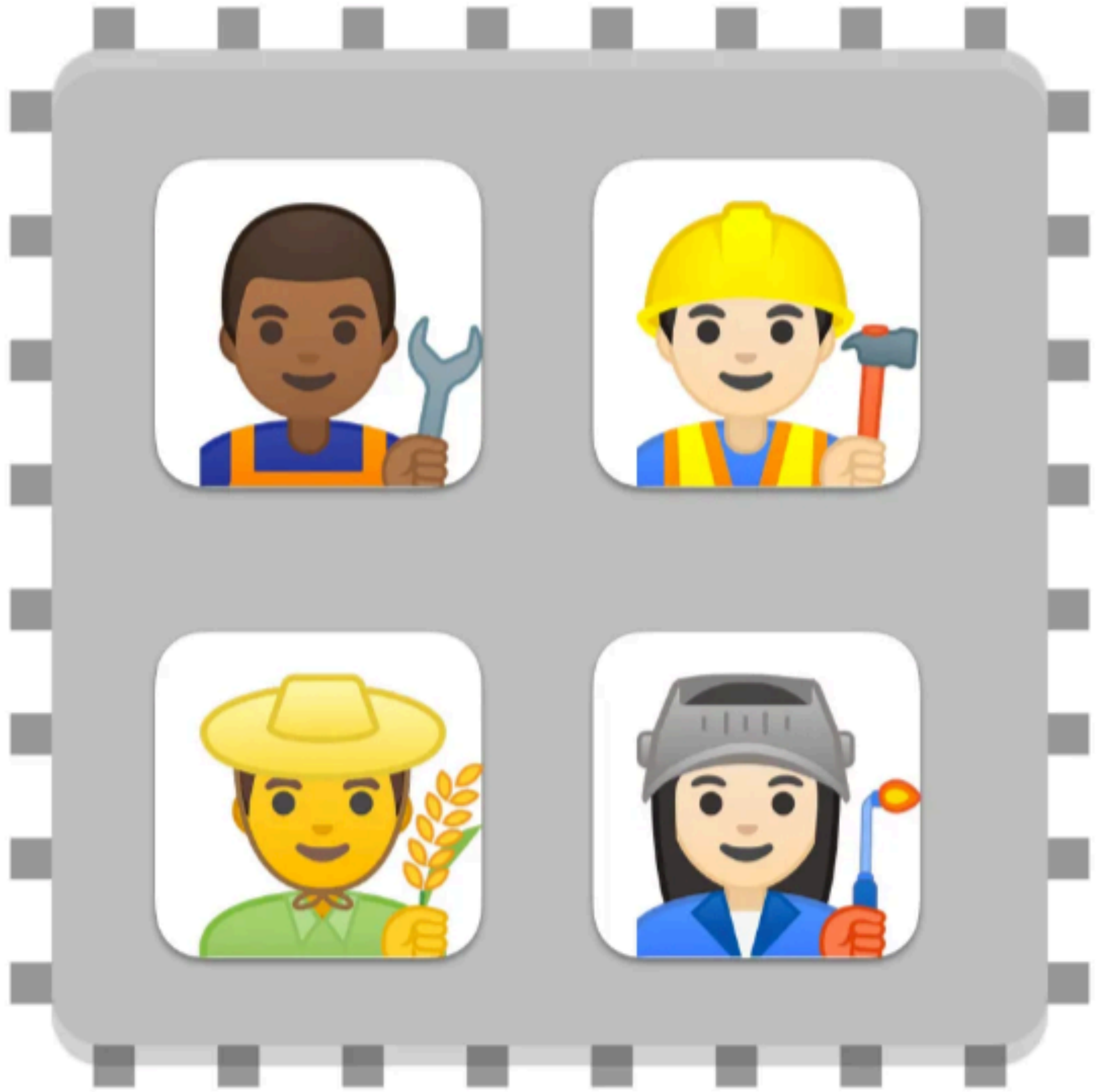
runBlocking: Example

```kotlin
fun main() {
    runBlocking {
        val instance = RetrofitInstance()
        val fact = instance.apiService.getFact()
        println(fact)
    }
}
```

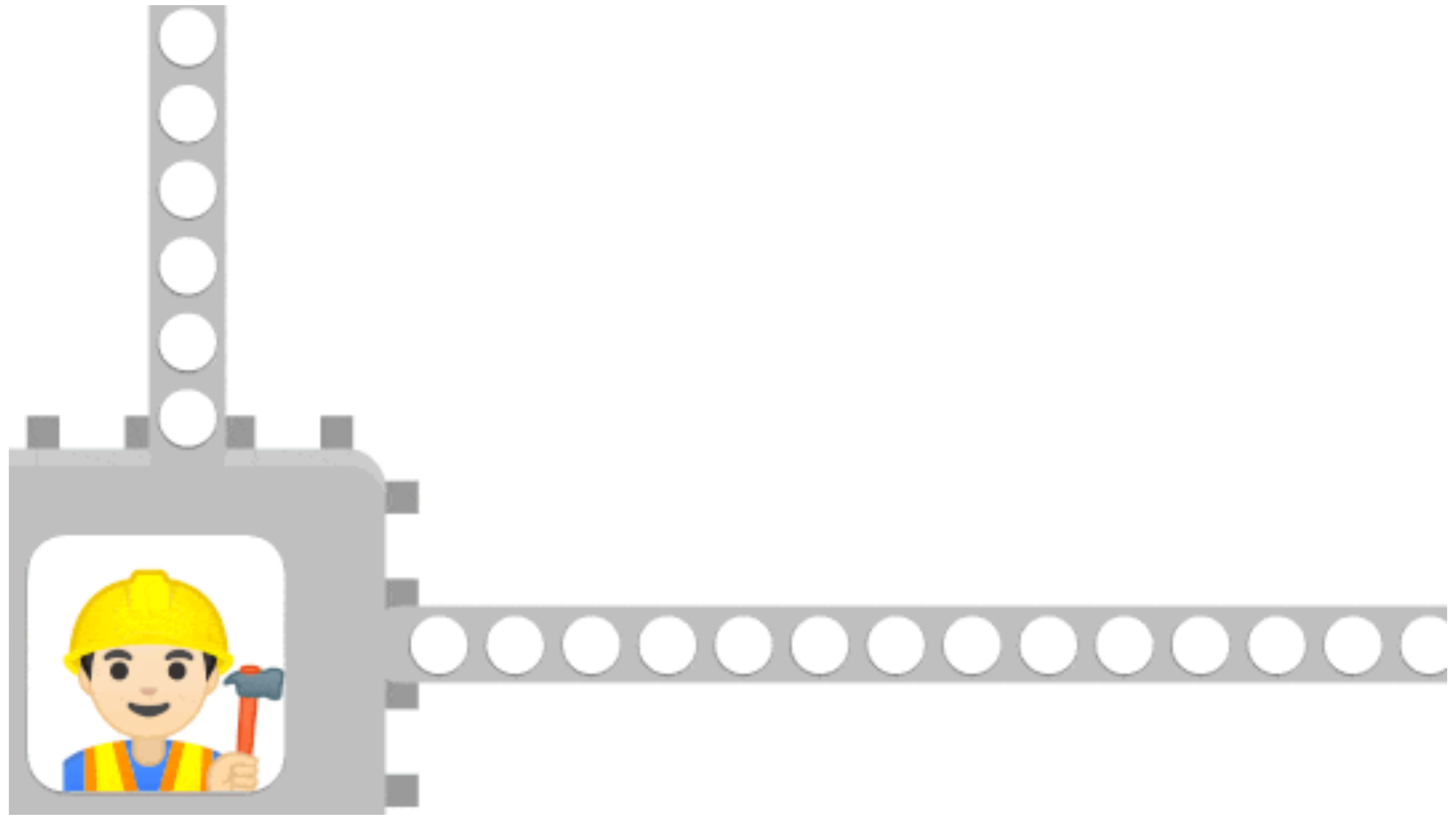
A Central Processing Unit (CPU) with 4 cores

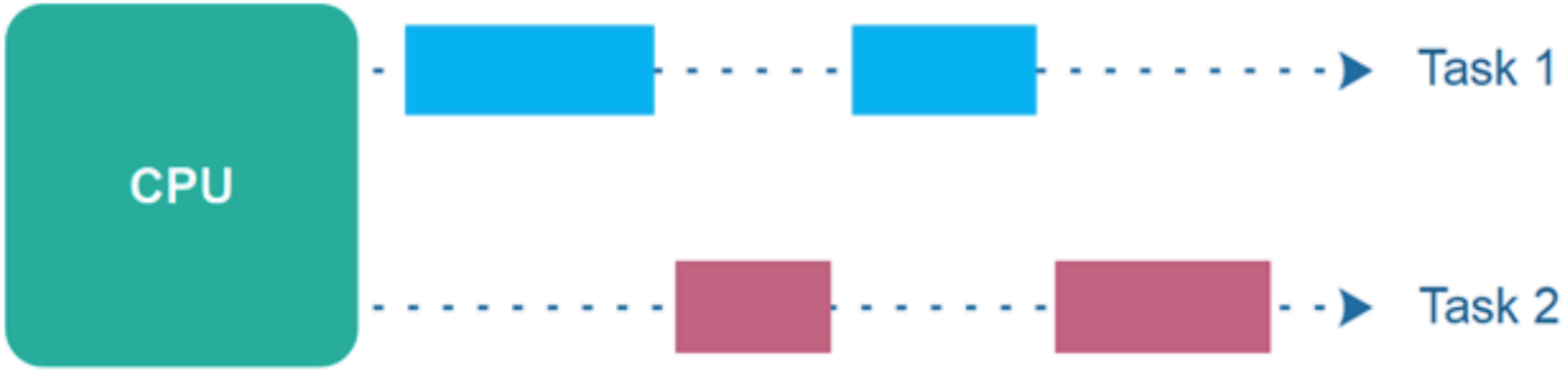
Concurrency in programming





https://medium.com/android-news/kotlin-coroutines-threads-concurrency-and-parallelism-101-78a56e09d373




CPU
Task 1
Task 2

A blocking network call

Not utilising concurrency

```
fun main() {
    runBlocking {
        val instance = RetrofitInstance()
        val fact = instance.apiService.getFact()
        println(fact)
    }
}
```

How to use concurrency in Kotlin?

Suspending functions

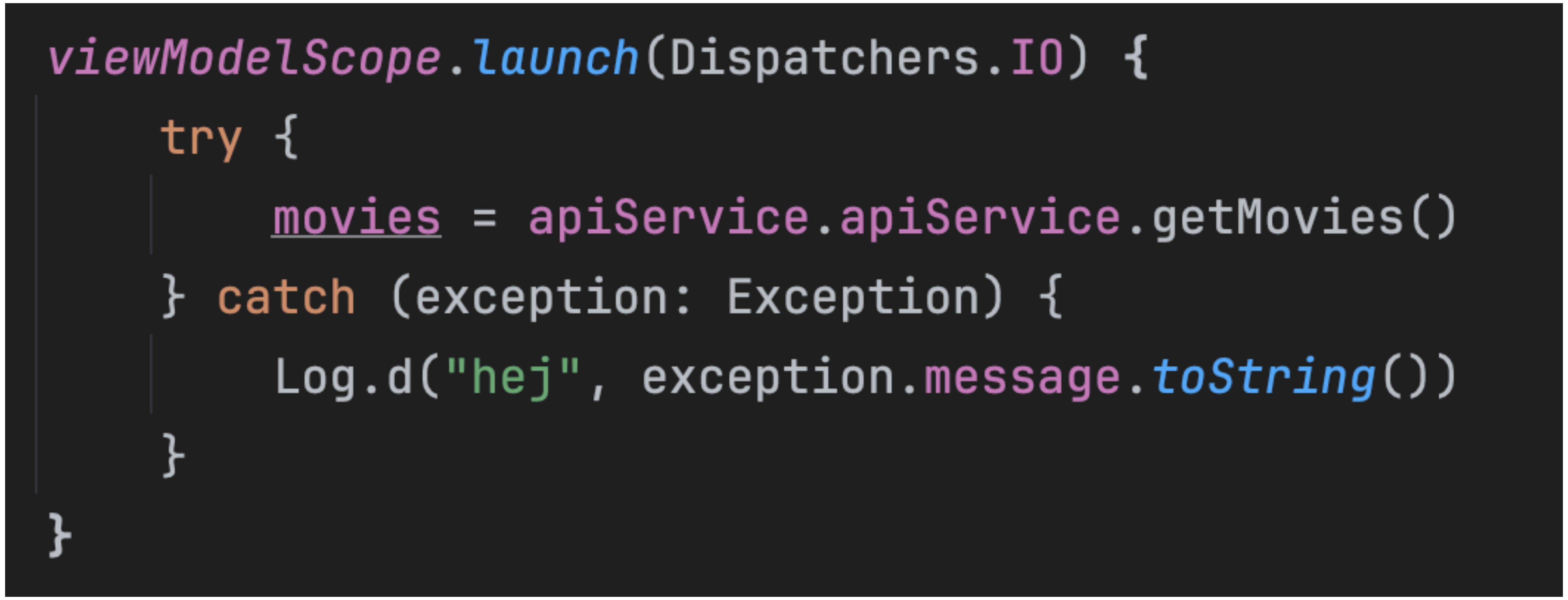
Kotlin concurrency

- A suspending function is a function that allows it to be paused and resumed at a later stage
- Suspending functions can only be invoked by another suspending function or within a coroutine
- In the following example, the function body is populated by the retrofit framework

```kotlin
interface CatFactsApi {
    @GET("/fact")
    suspend fun getFact(
    ): CatFact
}
```

Launching a suspending function

Kotlin concurrency

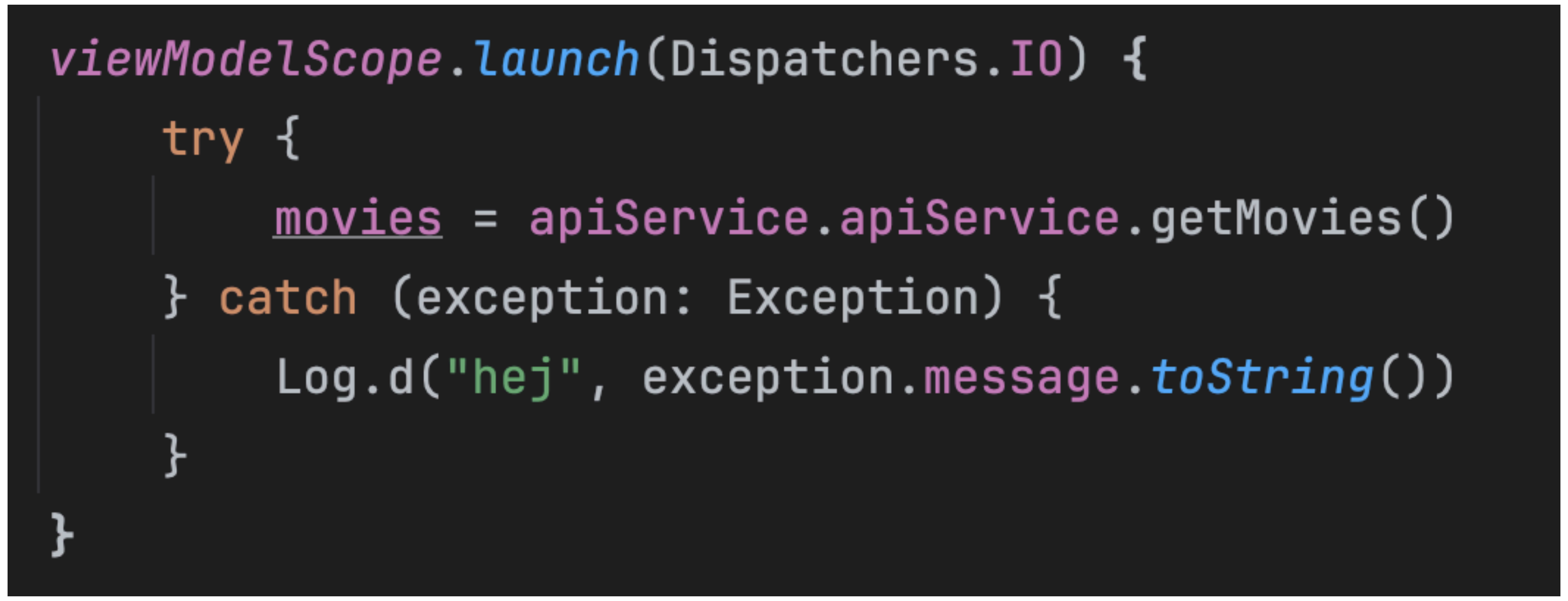


```kotlin
viewModelScope.launch(Dispatchers.IO) {
    try {
        movies = apiService.apiService.getMovies()
    } catch (exception: Exception) {
        Log.d("hej", exception.message.toString())
    }
}
```

Suspending functions

Kotlin concurrency

- A scope launches a suspending function. It handles the lifecycle of the coroutine - if the viewModel (in this instance) dies - the coroutine dies as well.
- Scopes can be GlobalScope, viewModel, lifecycleScope or CoroutineScope. What scope to use depends on where you are launching your coroutine from!
- In this course - the viewModelScope should be the primary (if not the only)

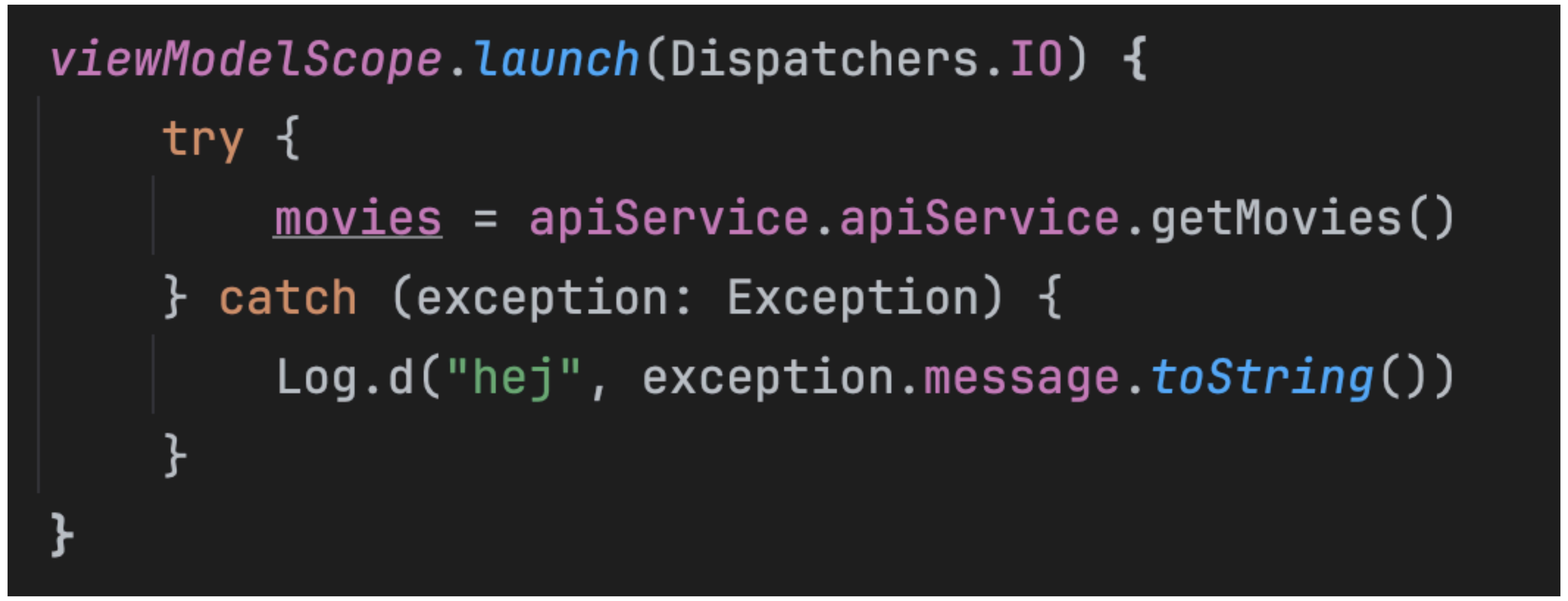


```
viewModelScope.launch(Dispatchers.IO) {
    try {
        movies = apiService.apiService.getMovies()
    } catch (exception: Exception) {
        Log.d("hej", exception.message.toString())
    }
}
```

Which scope to choose

Kotlin concurrency

- viewModelScope - Best for Running coroutines in a `ViewModel`. The coroutine gets automatically cancelled when the viewModel is cleared
- lifecycleScope - Running coroutines in `Activity` or `Fragment`. Automatically cancels coroutines when the activity and fragment is destroyed
- coroutineScope - Best for custom coroutine scopes that you manage yourself. Use when you need a custom scope for non-UI related tasks
- GlobalScope - This scope should generally be avoided. But is for long-running background work. Coroutine does not get cancelled

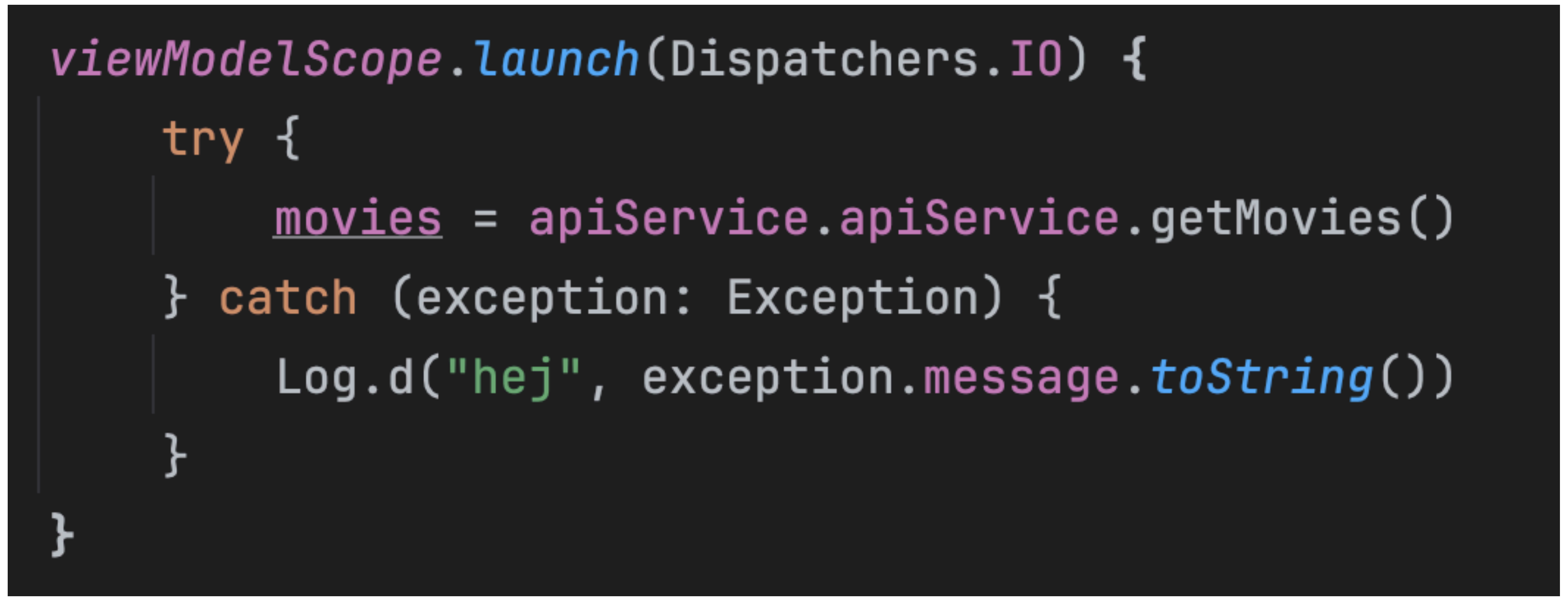


```kotlin
viewModelScope.launch(Dispatchers.IO) {
    try {
        movies = apiService.apiService.getMovies()
    } catch (exception: Exception) {
        Log.d("hej", exception.message.toString())
    }
}
```

Dispatchers

Dispatchers control which thread a coroutine runs on.

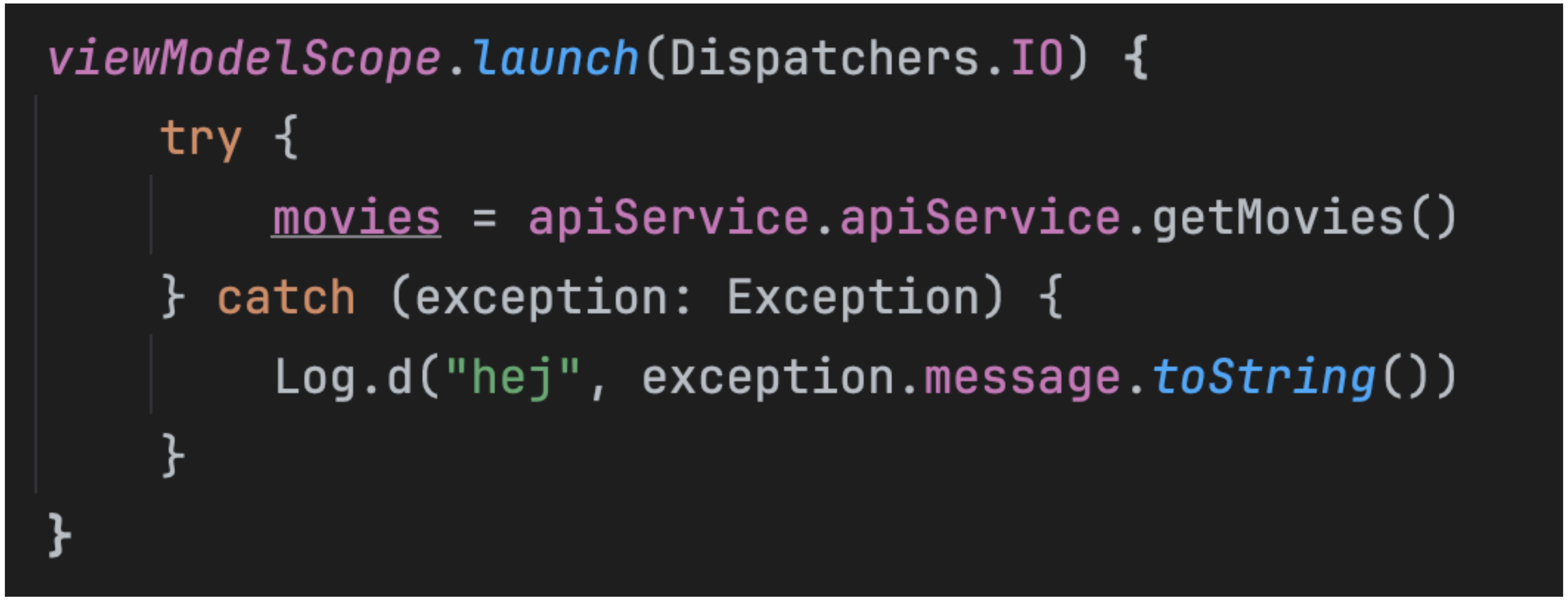
- **Default**: For CPU-bound tasks.
- **IO**: For I/O-bound tasks.
- **Main**: For UI-related tasks in Android or UI applications.
- **Unconfined**: For inheriting the context of the enclosing coroutine.
- **Custom**: Tailored to specific use cases defined by developers.



```kotlin
viewModelScope.launch(Dispatchers.IO) {
    try {
        movies = apiService.apiService.getMovies()
    } catch (exception: Exception) {
        Log.d("hej", exception.message.toString())
    }
}
```

Try/Catch block

Error handling



```
viewModelScope.launch(Dispatchers.IO) {
    try {
        movies = apiService.apiService.getMovies()
    } catch (exception: Exception) {
        Log.d("hej", exception.message.toString())
    }
}
```

Example

Introduction to todays project