Concurrency & Network calls in an android application

Applikationsudvikling

NIFR 2024

Agenda

Applikationsudvikling

- Concurrency
  - Blocking vs. Threads
  - Co-routines: suspend & launch
- Error handling (Try/Catch)
- Refactoring from experiment to application

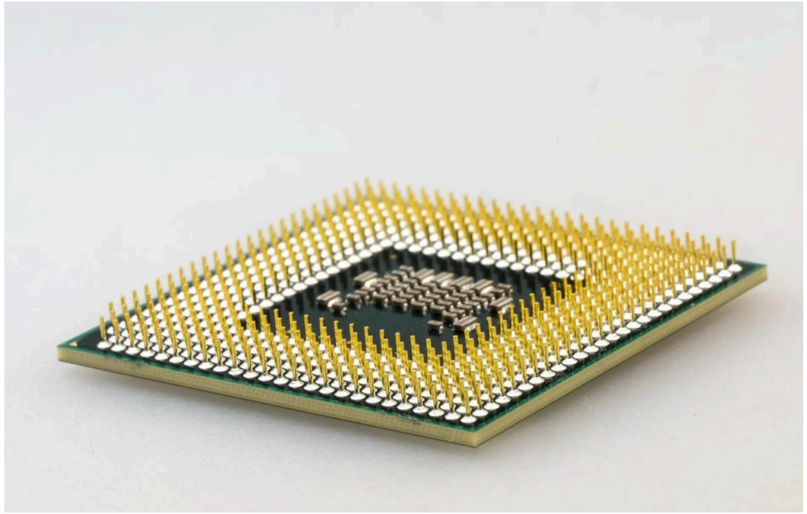
A blocking network call

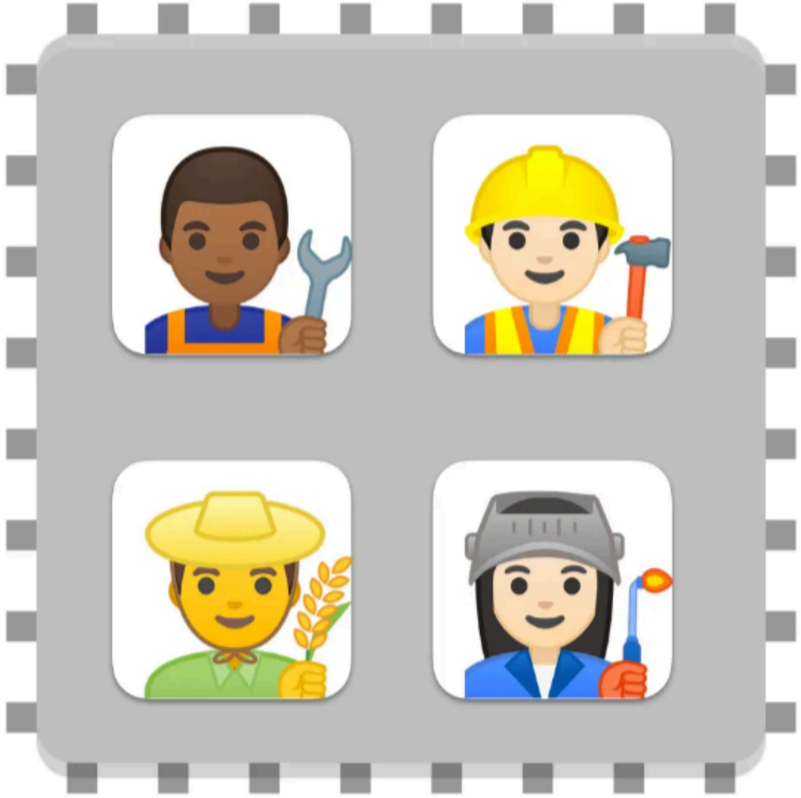
runBlocking: Example

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

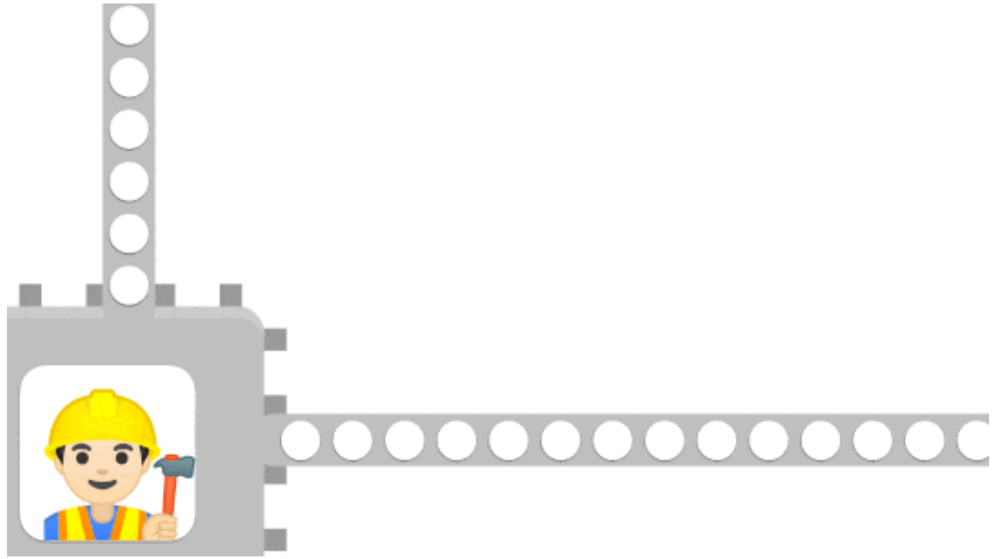
A Central Processing Unit (CPU) with 4 cores

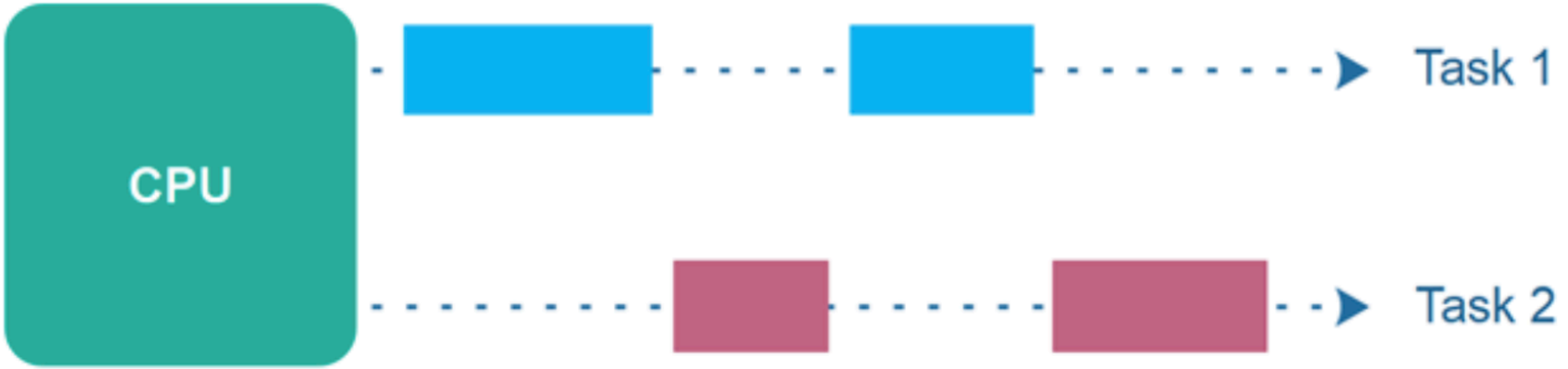
Concurrency in programming

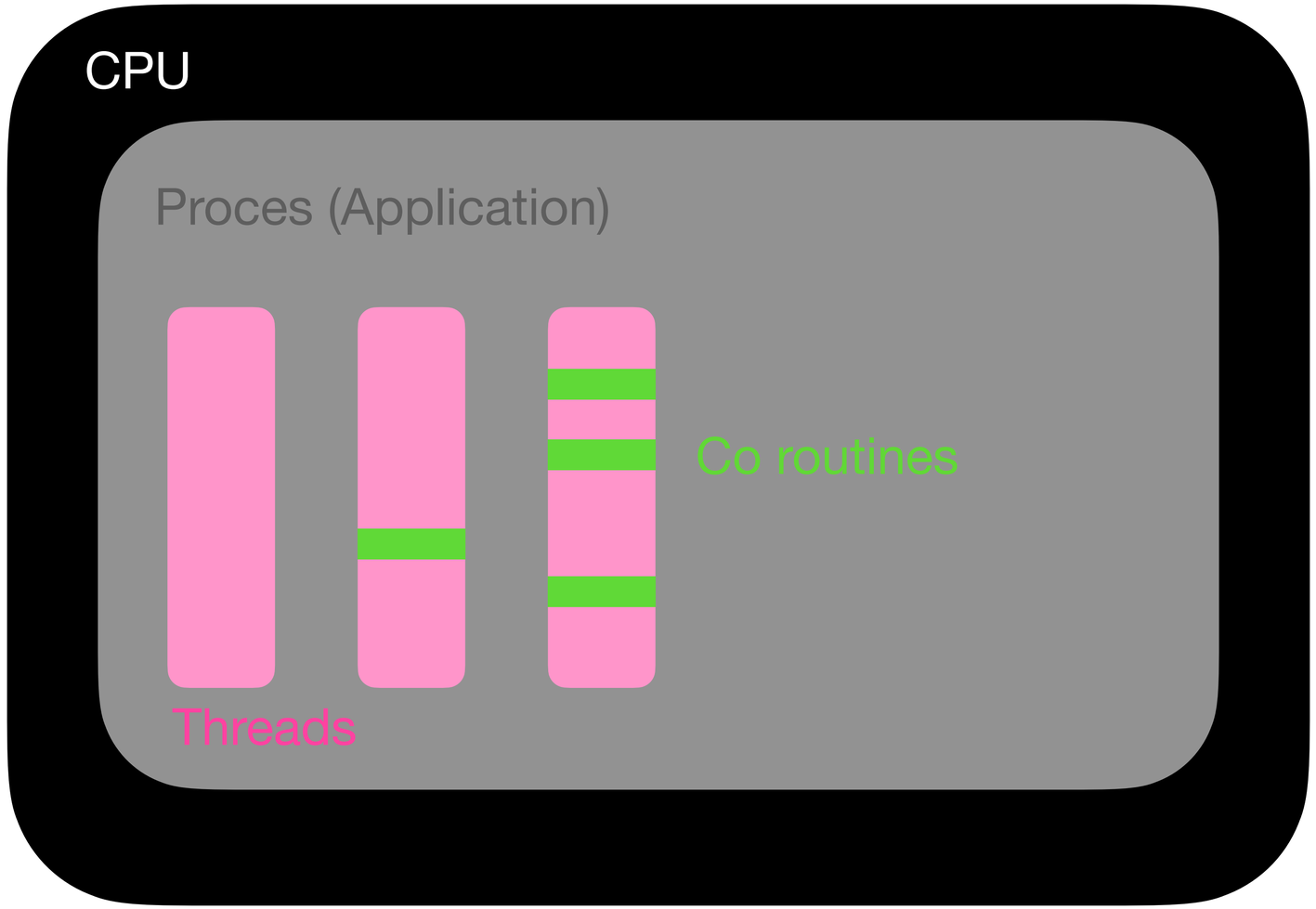




https://medium.com/android-news/kotlin-coroutines-threads-concurrency-and-parallelism-101-78a56e09d373




CPU
Task 1
Task 2


CPU
Proces (Application)
Co routines
Threads

A **coroutine** live in a thread,
a **thread** lives in process,
a **process** lives in core,
a **core** lives in a CPU.

https://dnshariprasad.medium.com/process-thread-coroutine-1f34d100fb0b

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

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

Launching a suspending function

Kotlin concurrency

```
viewModelScope.launch {
    try {
        val data = withContext(Dispatchers.IO) {
            currentCatFactString = catFactsRepository.getCatFact().fact
        }
    } catch (e: Exception) {
        currentCatFactString = e.message.toString()
    }
}
```

viewModelScope.launch

Kotlin concurrency

```
viewModelScope.launch {
    try {
        val data = withContext(Dispatchers.IO) {
            currentCatFactString = catFactsRepository.getCatFact().fact
        }
    } catch (e: Exception) {
        currentCatFactString = e.message.toString()
    }
}
```

Suspending functions

Kotlin concurrency

- A scope launches a suspending function.
- Scopes can be global, viewModel, lifecycleScope or custom.
- In this course - the viewModelScope should be the primary (if not the only)
- The scope defines the lifecycle of the coroutine - if the viewModel (in this instance) dies - the coroutine dies as well.

```
viewModelScope.launch {
    try {
        val data = withContext(Dispatchers.IO) {
            currentCatFactString = catFactsRepository.getCatFact().fact
        }
    } catch (e: Exception) {
        currentCatFactString = e.message.toString()
    }
}
```

Executes the retrofit function

```
viewModelScope.launch {
    try {
        val data = withContext(Dispatchers.IO) {
            currentCatFactString = catFactsRepository.getCatFact().fact
        }
    } catch (e: Exception) {
        currentCatFactString = e.message.toString()
    }
}
```

- **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.

Try/Catch block

Error handling

```
viewModelScope.launch {
    try {
        val data = withContext(Dispatchers.IO) {
            currentCatFactString = catFactsRepository.getCatFact().fact
        }
    } catch (e: Exception) {
        currentCatFactString = e.message.toString()
    }
}
```

Example

Introduction to todays project