JÖNKÖPING UNIVERSITY

School of Engineering

KOTLIN ASYNC. OPERATIONS

Peter Larsson-Green
Jönköping University
Spring 2020

THE MAIN THREAD

Executes by default all your code.

```
println("I'm executed by the main thread!")
```

A BACKGROUND THREAD

Can be created and started using the `thread()` function.

```
println("I'm executed by the main thread!")
thread {
  println("I'm executed by a background thread!")
}
println("I'm executed by the main thread!")
```

Usually you want to use coroutines instead.

COROUTINES

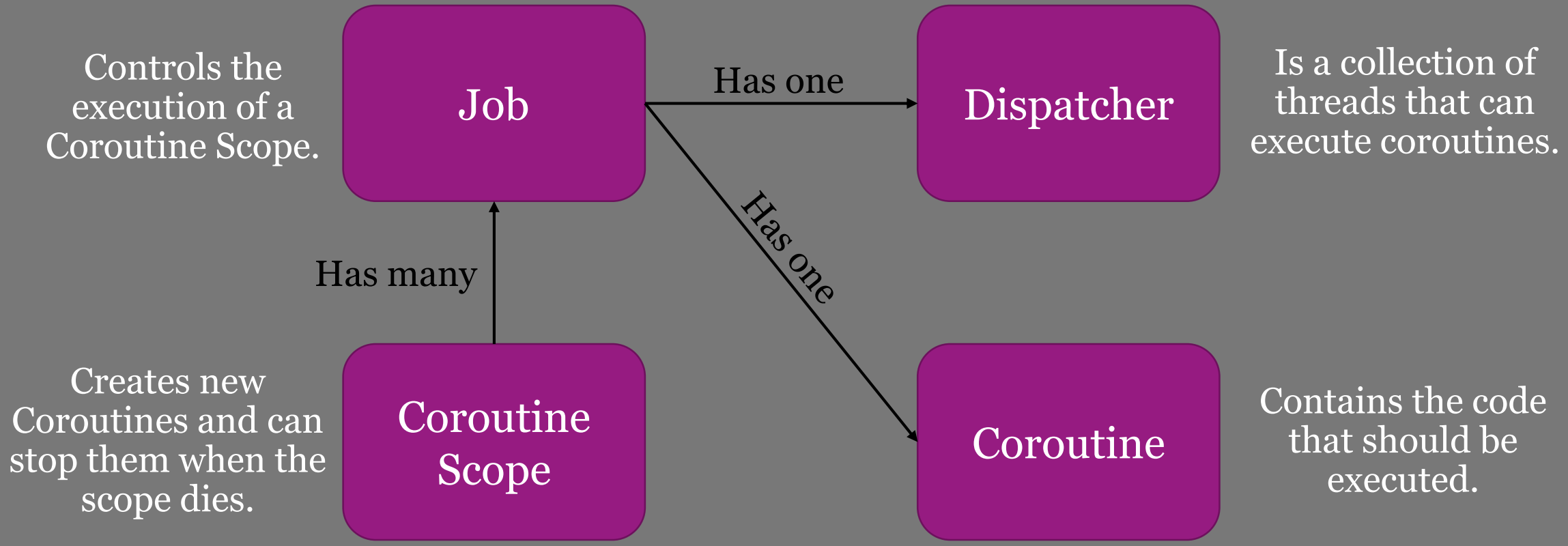
A more abstract view of asynchronous programming.

- Threads are used under the hood, but we don't worry about them.
- Kotlin contains extra syntax to more conveniently work with coroutines.

What is a coroutine?

- A piece of code that should be executed, potentially concurrently with other code.
  - Think of it as a function.

COROUTINES


Controls the execution of a Coroutine Scope.
Job
Has one
Dispatcher
Is a collection of threads that can execute coroutines.
Has many
Has one
Creates new Coroutines and can stop them when the scope dies.
Coroutine Scope
Coroutine
Contains the code that should be executed.

DISPATCHERS

- A collection of threads that can execute coroutines.
- There are pre-defined coroutine scopes we can use, e.g.:
  - `GlobalScope`
    Use this one if no more specific dispatcher is suitable.
  - `Dispatchers.IO`
    Use this one for Input/Output operations (files, network, etc.).
  - `Dispatchers.Main`
    Use this one for code you want to be executed by the main thread.
- You can also create your own dispatchers.

COROUTINE SCOPES

- Starts/stops Jobs automatically.
- There are pre-defined Coroutine Scopes we can use, e.g.:
  - `GlobalScope`
    Stops when the application stops.
- Frameworks usually provide their own Coroutine Scopes, e.g.:
  - `ViewModelScope` in Android
    Stops when the activity/fragment is destroyed (not re-created).
- You can also create your own Coroutine Scopes.

CREATING COROUTINES

Coroutines can be created using the `launch()` method on a Coroutine Scope.

```
println("I'm executed by the main thread!")
val job = GlobalScope.launch(Dispatchers.Default){
  println("I'm executed by a background thread!")
  // Put long running operations here!
}
println("I'm executed by the main thread!")
```

CHANGING DISPATCHER

Use the `withContext()` function to change dispatcher.

```
println("I'm executed by the main thread!")
val job = GlobalScope.launch(Dispatchers.Default){
  val result = longRunningOperations()
  withContext(Dispatchers.Main){
    println("I'm executed by the main thread!")
    // Update the GUI.
  }
}
println("I'm executed by the main thread!")
```

SUSPENDING VS WAITING

The thread executing a coroutine can swap between coroutines it should execute when calling a *suspend function*.

```
GlobalScope.launch(Dispatchers.Main).launch{
  delay(1000L) // Built in suspend function.
  println("2")
}
GlobalScope.launch(Dispatchers
  println("1")
}
```

```
GlobalScope.launch(Dispatchers.Main).launch{
  Thread.sleep(1000L)
  println("1")
}
GlobalScope.launch(Dispatchers.Main).launch{
  println("2")
}
```

SUSPENDING FUNCTIONS

Suspend functions can only be used in coroutines and other suspend functions.

```
suspend fun waitAndPrint(time: Long, message: String): Unit {
  delay(time)
  println(message)
}
GlobalScope.launch(Dispatchers.Main).launch{
  waitAndPrint(1000L, "2")
}
GlobalScope.launch(Dispatchers.Main).launch{
  println("1")
}
```