Parallel & Asynchronous Programming in Dart

By: Laams Innovation Lab

Revision

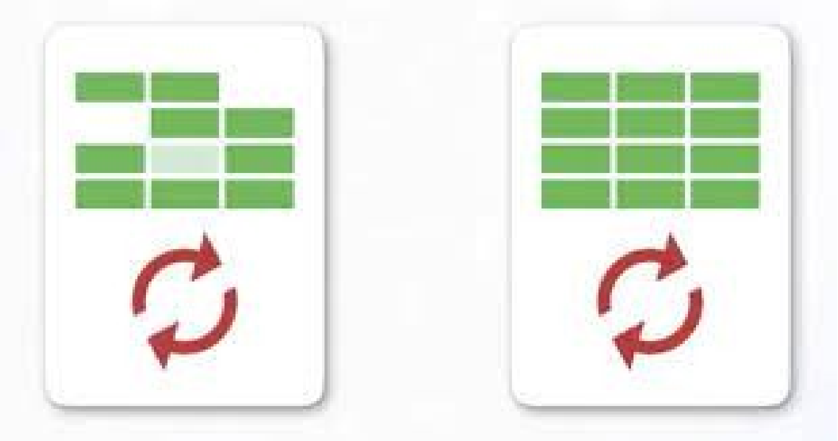
- Anonymous, recursive, closure, and callable classes.
- Grammar: { }, [ ], sync*
- Create a package called demographics
  - Create a random name generator function
  - Create a random number generator function,
  - Create a formatted person info printer which should include id, name, age, height, income,
  - Create one million list of people function with the aforementioned info.
  - Create a stats for the list of people. Total income, highest income, lowest income, number of children, number of working class, number of old people
  - Creating a formatted printer for the info

Understanding Dart Isolates

A Dart program runs in an **Isolate**, which is an isolated space, with its own **private chunk of memory**, and a **single-threaded** event loop.

You can spawn as many isolate as you like, for concurrent/parallel programming.

Communication between isolate happens through the **send/receive ports** only.



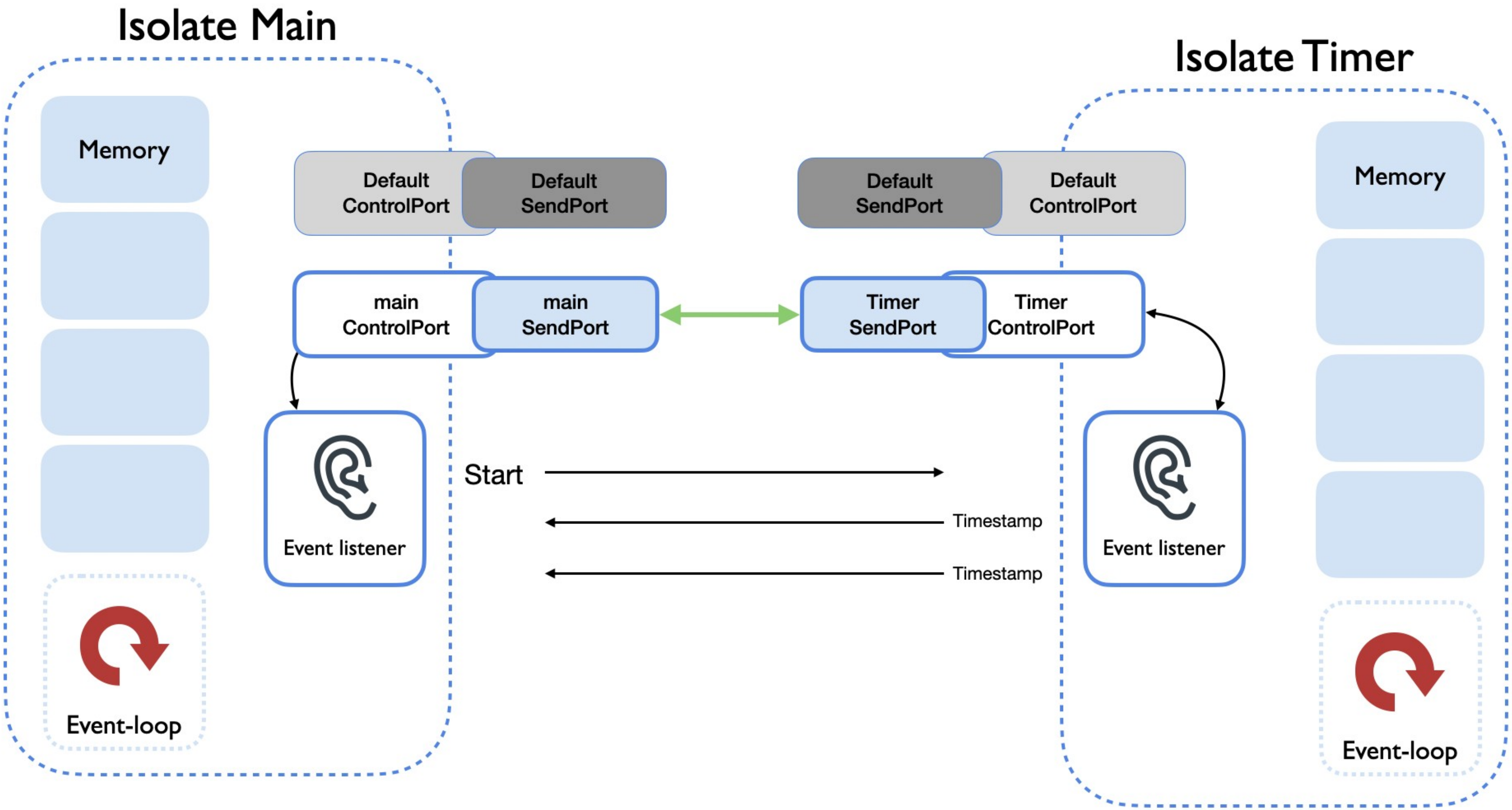
Parallel Programming

Helps **a unit of code** to run **in its own Isolate (application)** independent from **the main isolate,** and notify the main thread once **a message (data) is** is sent through the port.

This form of programming leads to better **performance** and **user experience**, by running multiple tasks in parallel and independent from each other, none blocking the other.

It is usually used when **computing large amount of data**.

Dart Isolates in a Picture



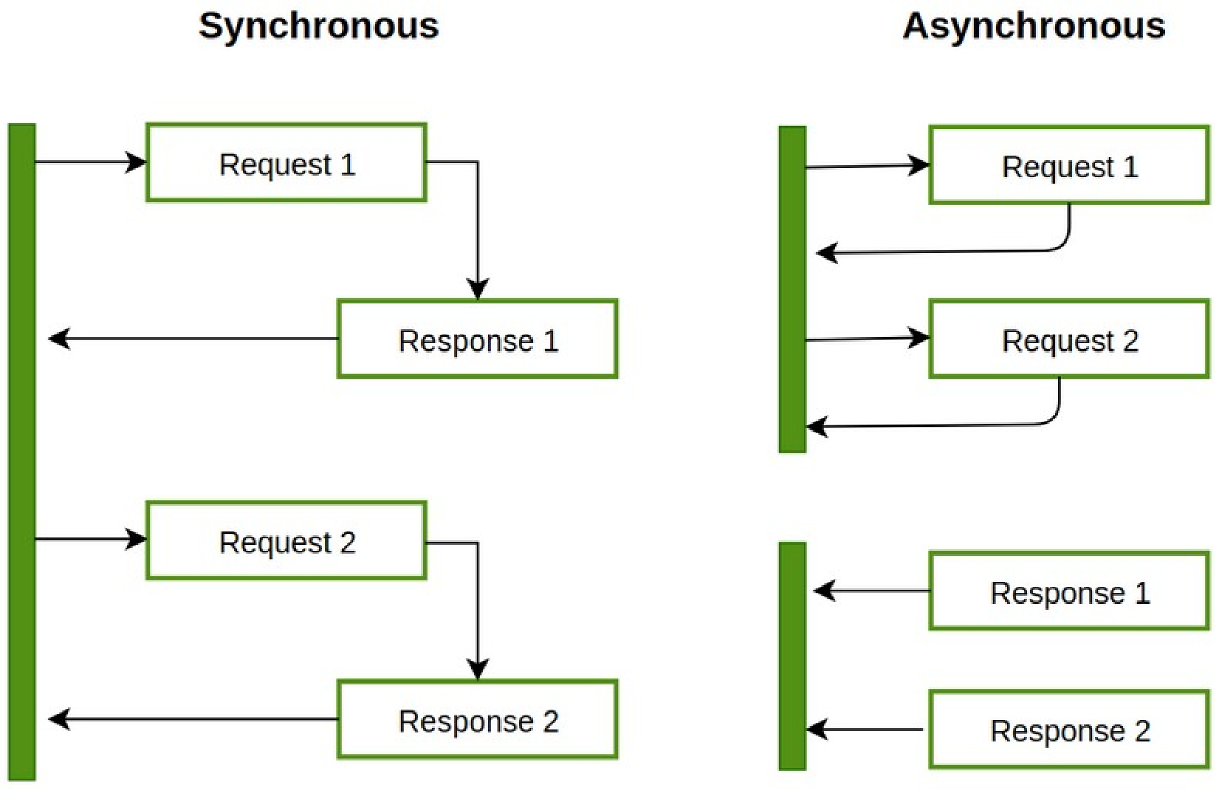
Asynchronous Programming in Dart

Helps **a unit of code** to run **independently** from **the main thread,** and notify the main thread once **completed** or when some **data/error** is available at a **later** time.

This form of programming leads to better **performance** and **user experience**, due to avoidance of **blocking code**.

It is usually used when **fetching data** from **i/o devices**.

Synchronous vs Asynchronous



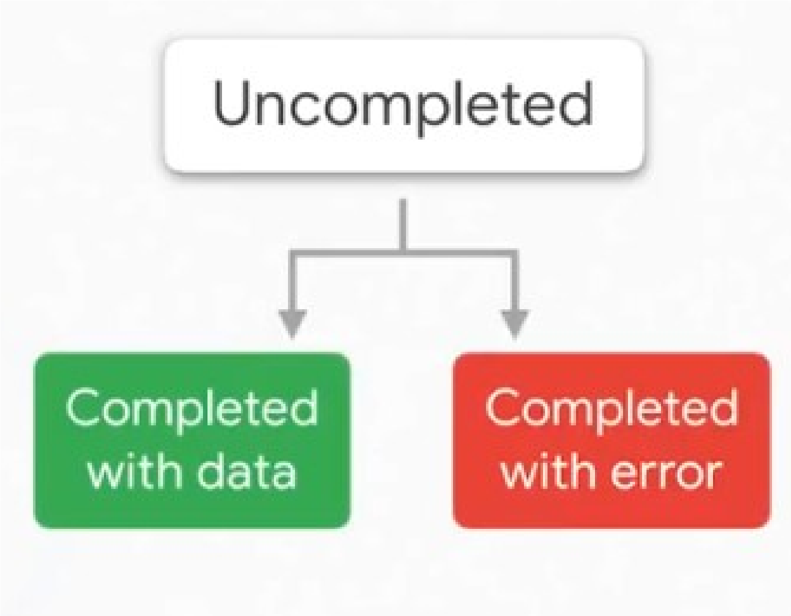
Future

A **Future<T>** represents a **value** or an **error** that will be available in the **future**, making **asynchronous programming** possible in Dart.

Future States

Futures hold three states:

- **Uncompleted:** the function has been called, but the **data** is NOT retrieved
- **Completed with Data:** the called data has been retrieved.
- **Completed with Error:** instead of the requested data an **error** has been retrieved.



Using Futures

Instantiating **Future<T>**:

- You can instantiate a Future<T> using its constructors.
- Calling **APIs** with a return data type of **Future<T>**.

Using **Future<T>**:

- Using its properties and methods
- Using **async**/**await** keywords.

Understanding async/await Keywords

It is used to call asynchronous code synchronously.

Alternative **syntax** for asynchronous programming, leading to **cleaner** and **readable** code.

Creating Your Own **Future<T>** functions

You can simply create a function with a return type of **Future<T>**.