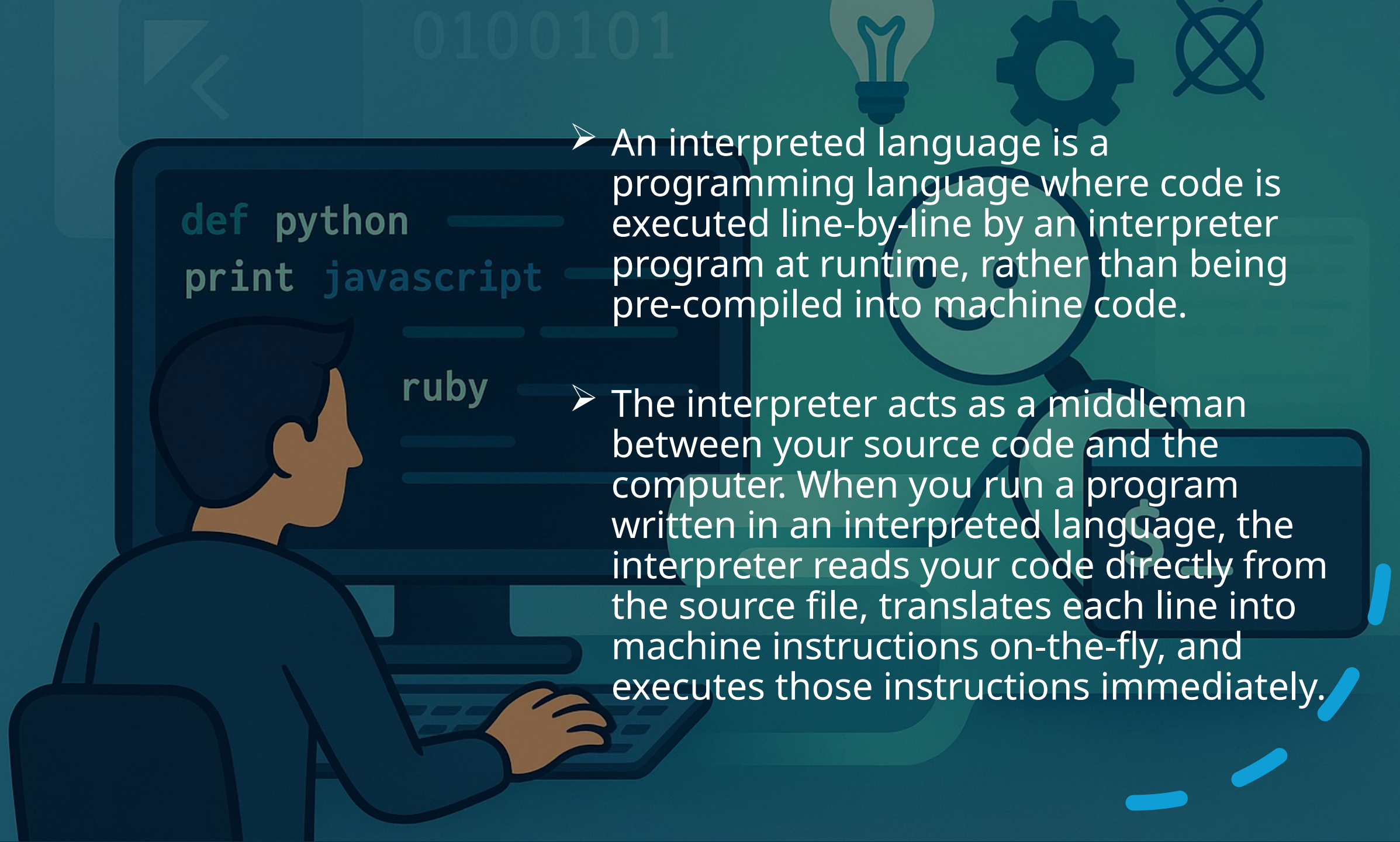
Mobile and Embedded Computing

Lecture 2. Languages & Flutter Intro

- A compiled language is a programming language that must be translated into machine code before it can run.
- Uses a compiler to convert the code into an executable file that the computer can directly run. This translation step happens once, before you run the program.



- An interpreted language is a programming language where code is executed line-by-line by an interpreter program at runtime, rather than being pre-compiled into machine code.

- The interpreter acts as a middleman between your source code and the computer. When you run a program written in an interpreted language, the interpreter reads your code directly from the source file, translates each line into machine instructions on-the-fly, and executes those instructions immediately.

Differences – Developer Experience – Compiled languages

- Slow to develop (edit, compile, link and run. The compile/link steps could take serious time)
- You need to "rebuild" the program every time you need to make a change
- Compiled languages require longer development cycles but catch errors earlier in the development process
- Early error detection during compilation prevents runtime failures
- Better for large, complex applications where catching bugs early saves time

Differences – Developer Experience – Interpreted languages

Interpreted languages accelerate development cycles through immediate code execution and dynamic testing capabilities. Developers can modify and test code without compilation delays

Since there is no compilation step, changes to the code can be immediately executed and tested, allowing for quick iterations and experimentation

Interpreted languages, on the other hand, offer greater flexibility and a shorter development feedback loop, which makes them well-suited for scripting, prototyping, and rapid application development

Better for rapid prototyping, web development, and scenarios requiring frequent updates

Differences – Runtime Execution Speed

Compiled languages tend to be about 100 times faster than interpreted languages

Compiled languages generally outperform interpreted languages in execution speed benchmarks. Machine code execution eliminates interpretation overhead, resulting in faster program execution

Performance differences matter most in computationally intensive applications like scientific computing, real-time systems, etc

Differences – Security

01

Compiled code is difficult to reverse engineer or modify

02

Since the source code is often distributed alongside the interpreter, it may be easier to access and reverse engineer proprietary algorithms

03

Compiled languages catch more errors at compile-time, preventing security vulnerabilities from reaching production

04

Interpreted languages offer more runtime flexibility but this can be exploited maliciously

Null safe languages

Null safe languages

Null safety refers to a programming language's ability to prevent null pointer exceptions (NPEs) at compile time rather than runtime. The type system distinguishes between types that can hold null values and those that cannot, making null-related errors largely impossible when the code compiles successfully.

Java
Not null saf

```java
record User(String name, String email) {  11 usages
}

class UserService {
    public void processUser(User user) {  3 usages
        // This can throw NullPointerException at runtime
        System.out.println("Processing: " + user.name().toUpperCase());

        // Even checking for null doesn't guarantee safety
        if (user.email() != null) {
            System.out.println("Email: " + user.email().toLowerCase());
        }
    }

    Change signature
    static void main() {
        User user1 = new User( name: "John",  email: "john@example.com");
        User user2 = new User( name: null,  email: null); // Compiles fine!
        User user3 = null; // Also compiles fine!

        UserService service = new UserService();
        service.processUser(user1); // Works fine
        service.processUser(user2); // Runtime NPE on getName().toUpperCase()
        service.processUser(user3); // Runtime NPE on user.getName()  Passing 'null'
    }
}
```

Dart – Null safe

```dart
class User {
  final String name;    // Non-nullable - cannot be null
  final String? email;  // Nullable - can be null (note the ?)

  User({required this.name, this.email});
}

class UserService {
  void processUser(User user) {
    // This is guaranteed safe - name cannot be null
    print('Processing: ${user.name.toUpperCase()}');

    // Compiler forces null checking for nullable types
    if (user.email != null) {
      // Smart cast: email is promoted to non-null String inside this block
      print('Email: ${user.email!.toLowerCase()}');
    }

    // Alternative null-aware operators
    print('Email length: ${user.email?.length ?? 0}');
  }
}
```

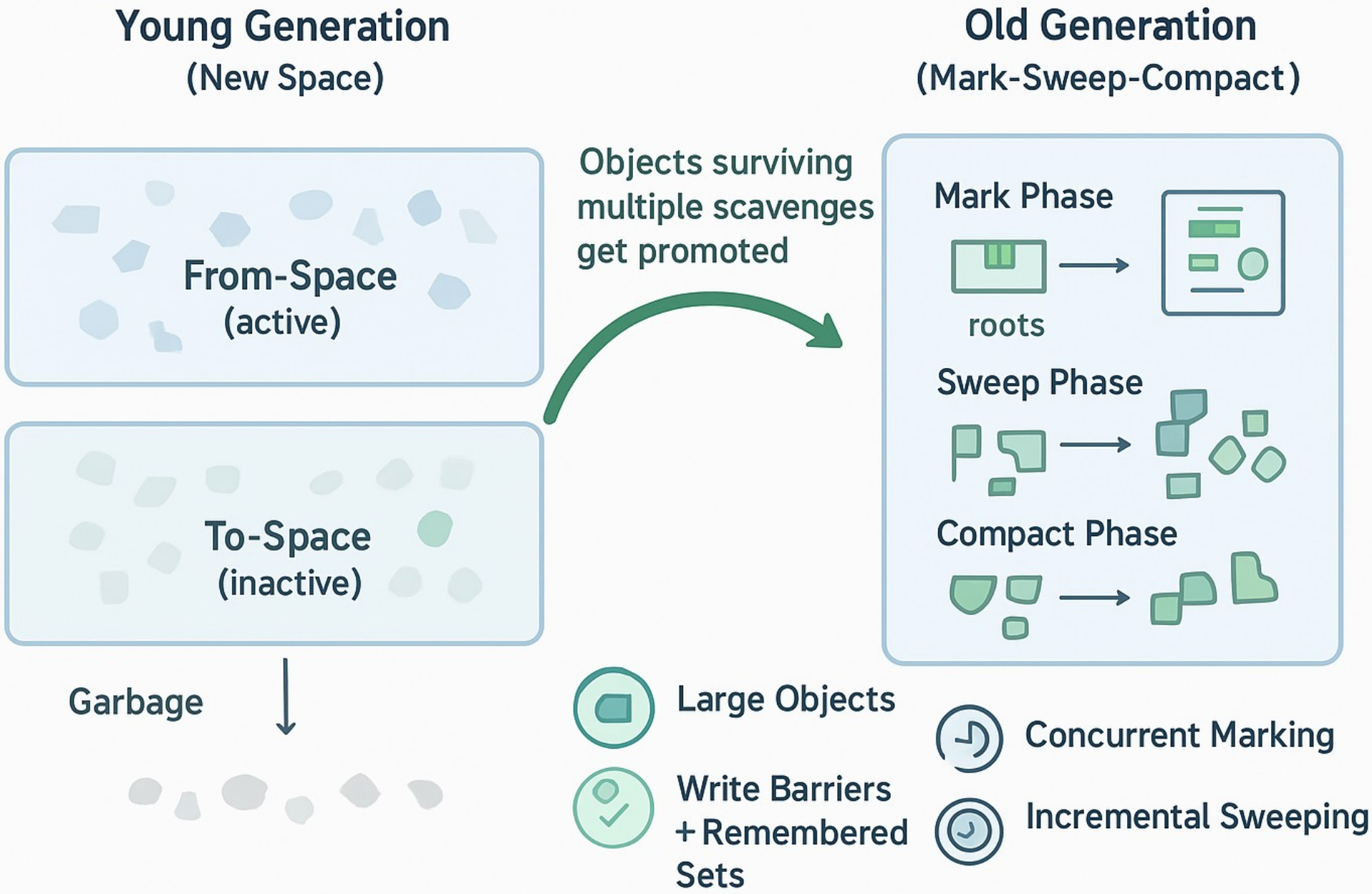
```dart
void main() {
  var user1 = User(name: 'John', email: 'john@example.com');
  var user2 = User(name: 'Jane'); // email is null, but explicitly nullable


  // This won't compile - name is required and non-nullable
  // var user3 = User(name: null); // Compilation error!

  // This won't compile - User itself is non-nullable
  // User user4 = null; // Compilation error!

  var service = UserService();
  service.processUser(user1); // Safe
  service.processUser(user2); // Safe - null handling is explicit

  // print(user2.email!.toLowerCase()); // Unsafe - will throw if email is null
  // print(user2.email?.toLowerCase() ?? 'No email provided'); // Safe alternative
}
```

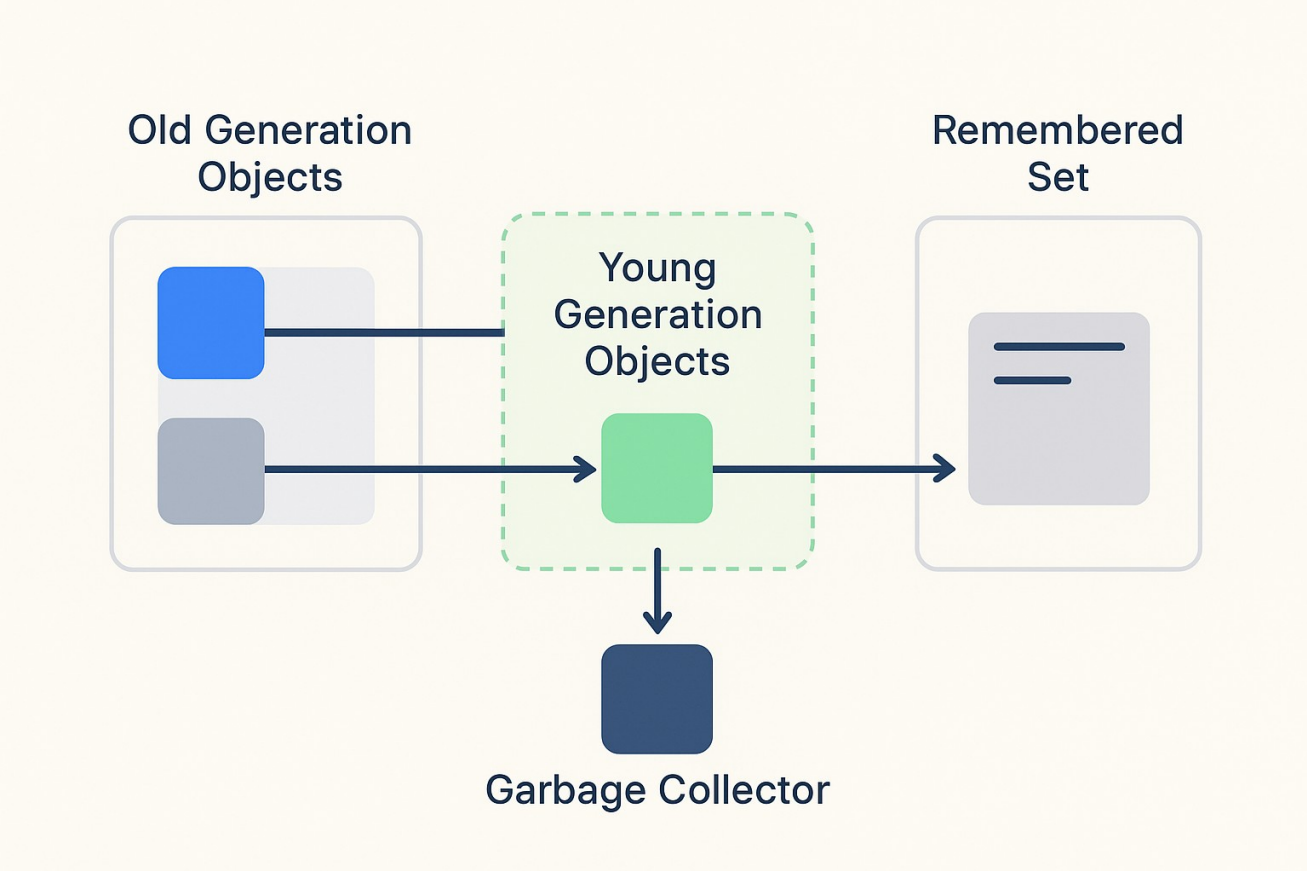
Dart

- client-optimized programming language developed by Google
- released in 2011
- Strong typed
- Null safe
- Object Oriented
- Built in support for Async programming
- Garbage collected
- Can hot reload

Dart Garbage Collector

- Two generation approach (Young generation and Old generation)
- Objects that survive multiple young generation collections are **promoted** to the old generation.
- The Dart VM implements **concurrent marking** for old generation collections. The marking phase runs concurrently with the application (mutator), reducing pause times. The collector uses **write barriers** to track modifications made by the application during concurrent marking, ensuring collection correctness.
- **Incremental Sweeping** The sweep phase can be performed incrementally, spreading the work across multiple small time slices rather than completing it all at once. This further reduces pause times by avoiding long stop-the-world phases.

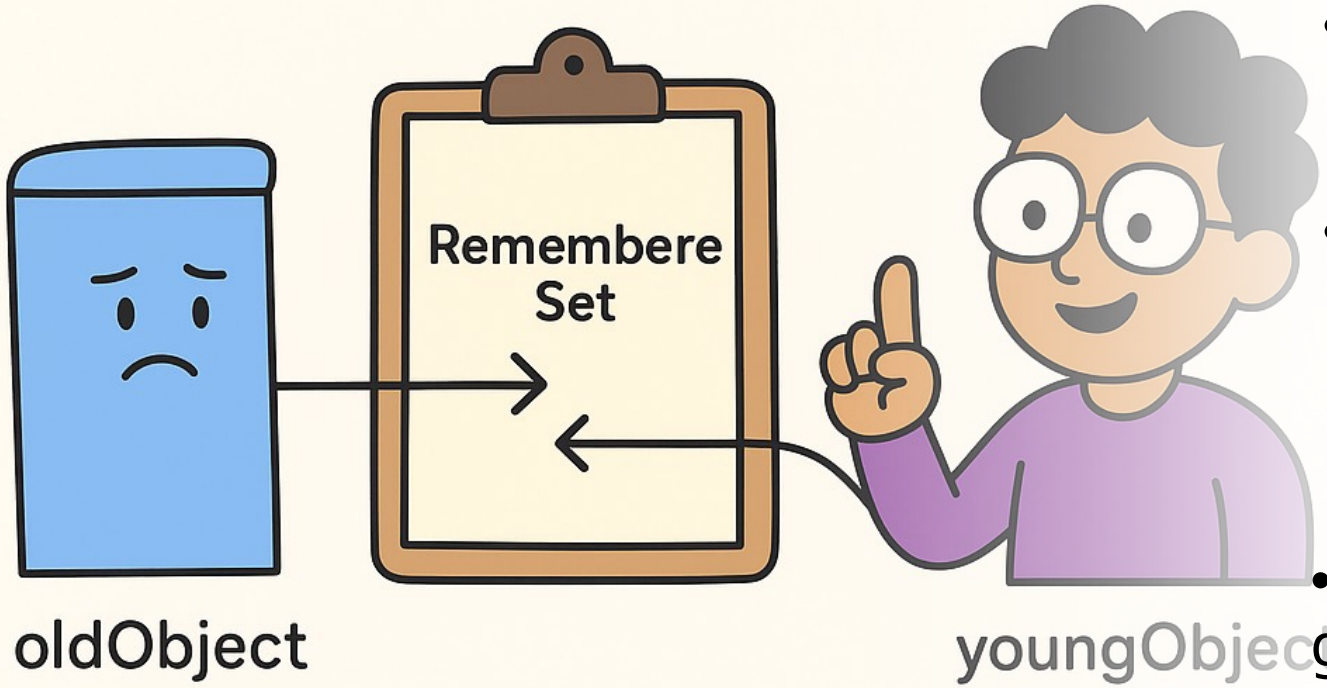

Young Generation
(New Space)
From-Space
(active)
To-Space
(inactive)
Garbage
Objects surviving
multiple scavenges
get promoted
Old Generation
(Mark-Sweep-Compact)
Mark Phase
roots
Sweep Phase
Compact Phase
Large Objects
Write Barriers
+ Remembered
Sets
Concurrent Marking
Incremental Sweeping



•In a generational garbage collector, the young generation is collected independently and much more frequently than the old generation. However, this creates a fundamental problem: **what if an old generation object holds the only reference to a young generation object?**

•A **remembered set** is a data structure that tracks references from the old generation to the young generation. Think of it as a "reminder list" that tells the young generation collector: "Don't forget to check these old objects when determining what's reachable in the young generation."

Remembered Set



- ✓ Contains pointers from old → young
- ✓ Used as extra roots during GC
- ✓ Ensures young objects are preserved

The remembered set typically contains:

- Pointers to old generation objects that reference young generation objects
- The specific memory locations (slots) within old objects that contain young generation references

•When performing a young generation collection, the garbage collector treats the remembered set as additional "roots" for reachability analysis. It scans all objects referenced in the remembered set to find young generation objects that should be preserved.

Write Barrier

A write barrier is a small piece of code that executes whenever the application modifies an object reference. It acts as a "hook" that allows the garbage collector to monitor and respond to changes in the object graph.

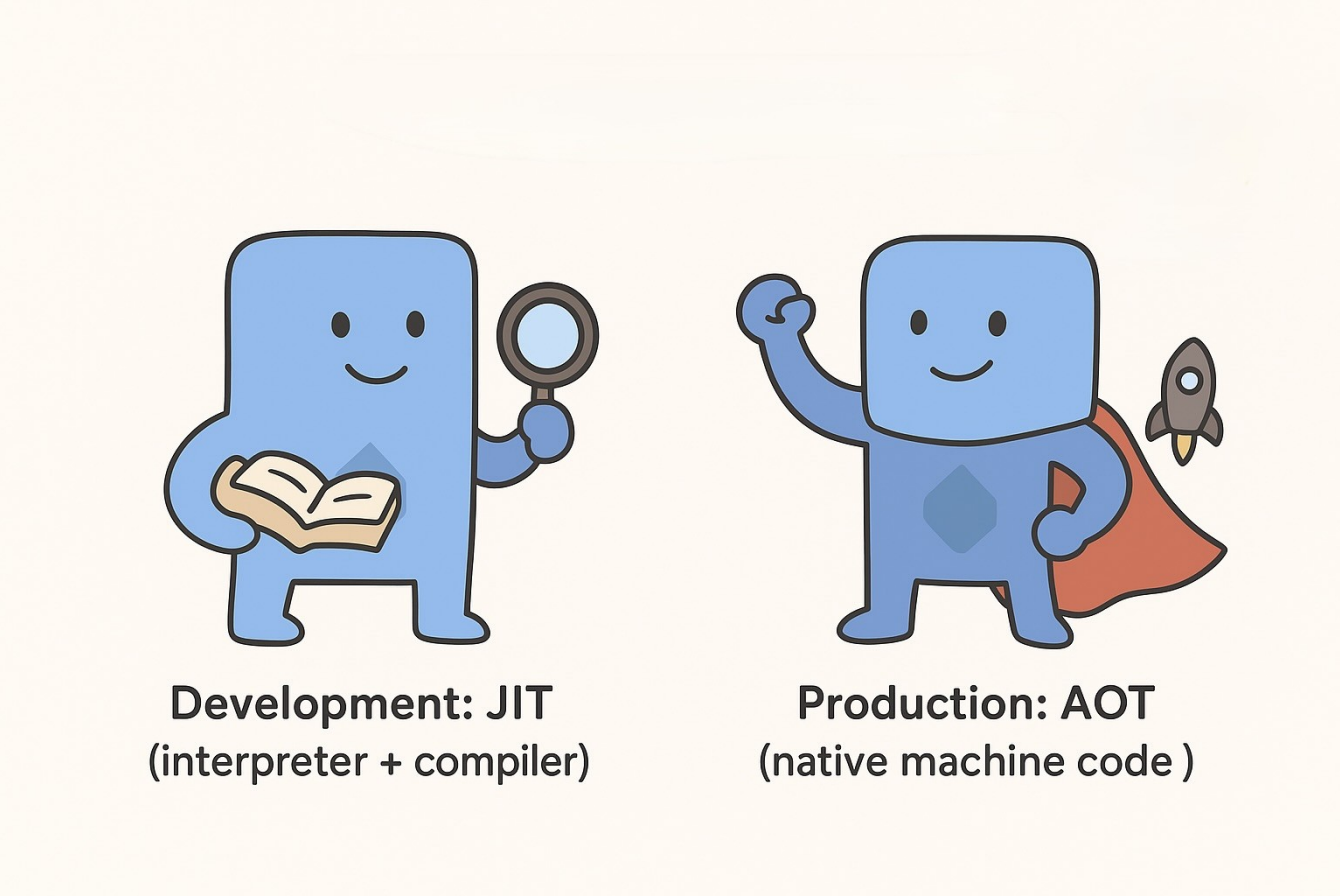
When the application executes code like oldObject.field = youngObject, the write barrier triggers and performs these steps:

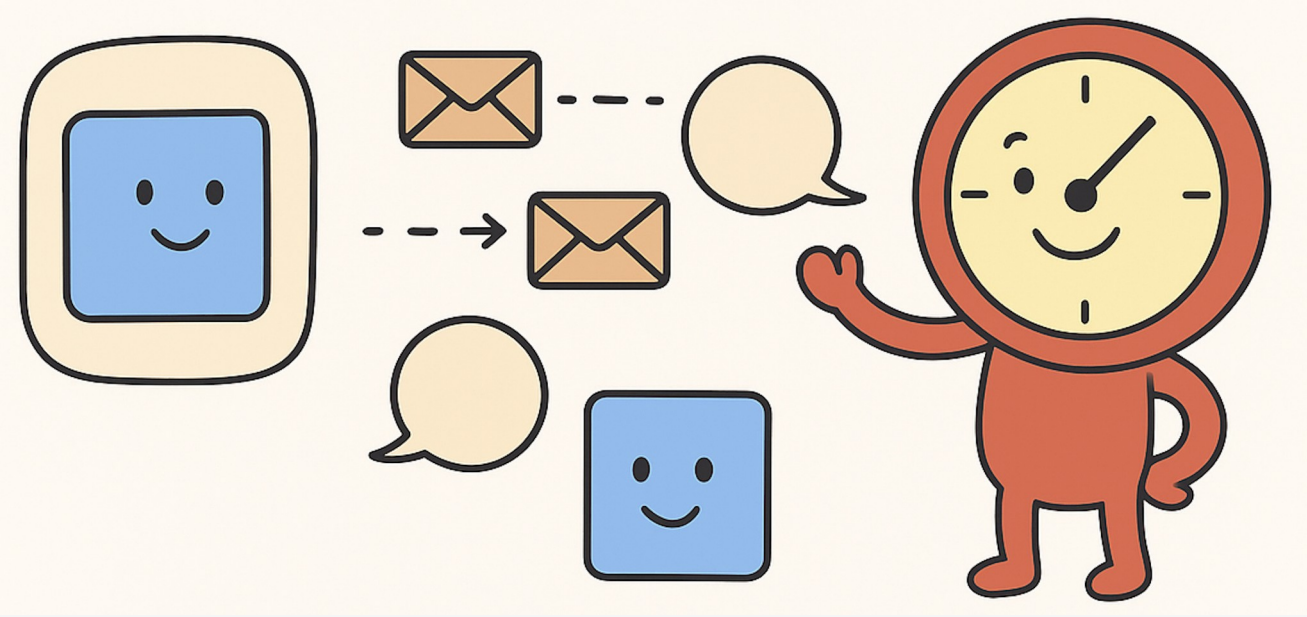
1. Is the target object (oldObject) in the old generation?
2. Is the new reference (youngObject) in the young generation?

=> If both conditions are true, add information about this reference to the remembered set

Dart VM

- During development operates in JIT mode (Dart source code is initially parsed and executed through an interpreter)
- In production runs in AOT mode (compiled ahead-of-time to native machine code, eliminating the VM overhead entirely. AOT compilation produces standalone executables or libraries that start faster and consume less memory than JIT-compiled applications)





Dart concurrency

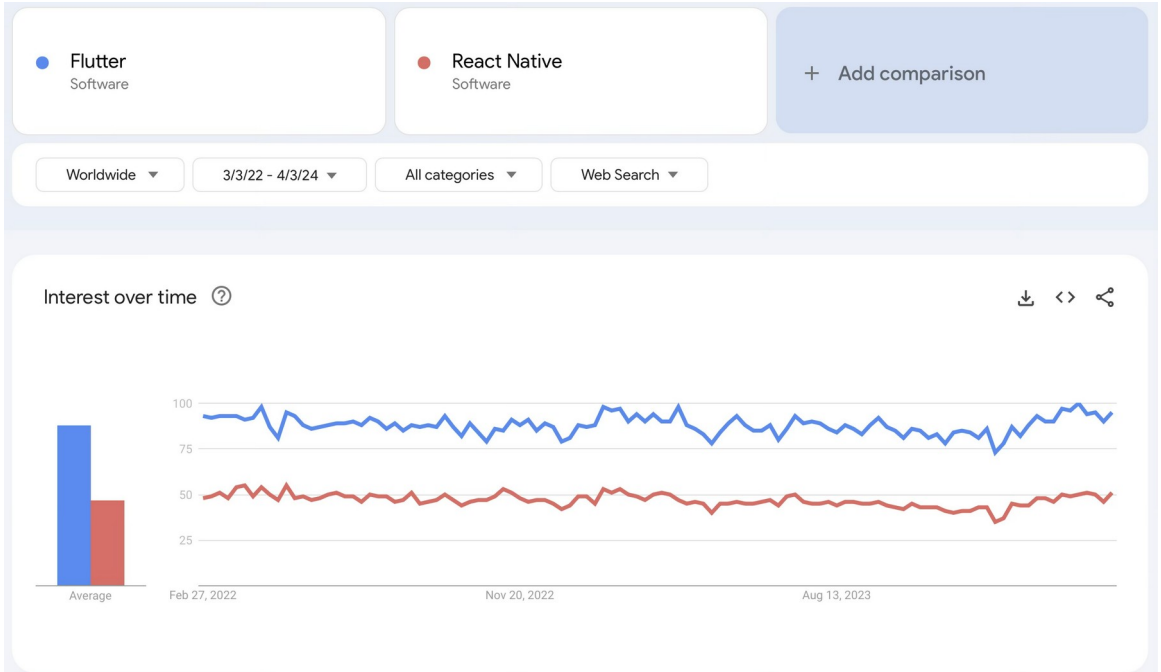
- Isolates (independent workers that communicate through message passing, each Isolate has its own memory heap and event loop)
- Async

Flutter

BMW

Scaling customer-centric product development at BMW Group with Flutter

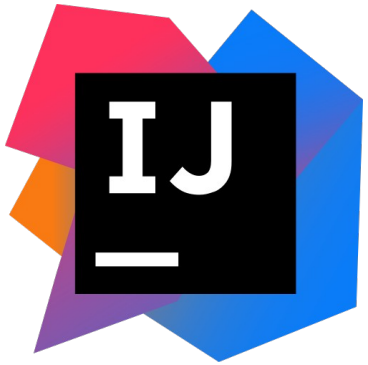
Read the story →



250k+ installs
Flutter + Spring Boot, Firebase
https://scanneralimente.ro

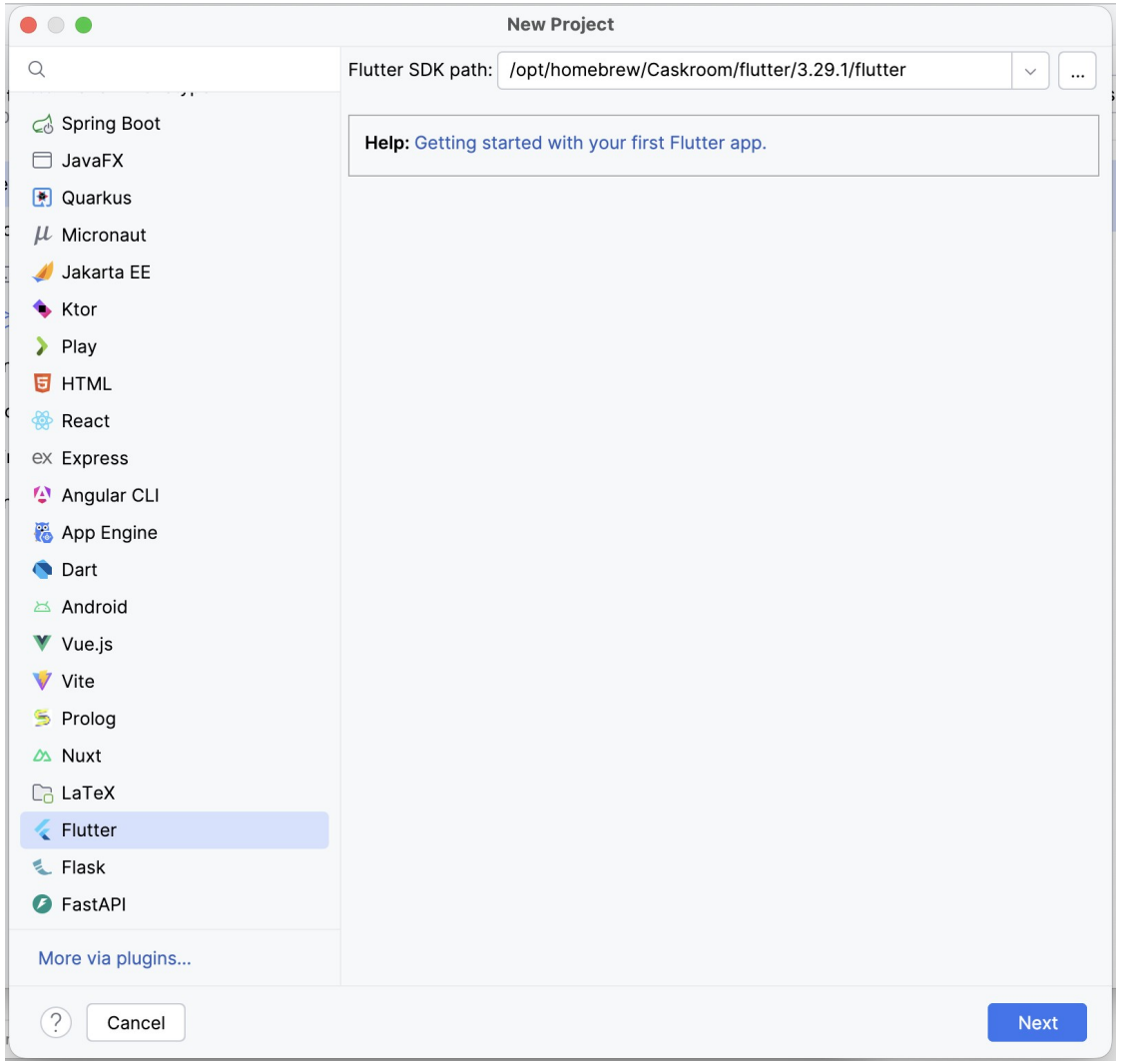
+45 other apps

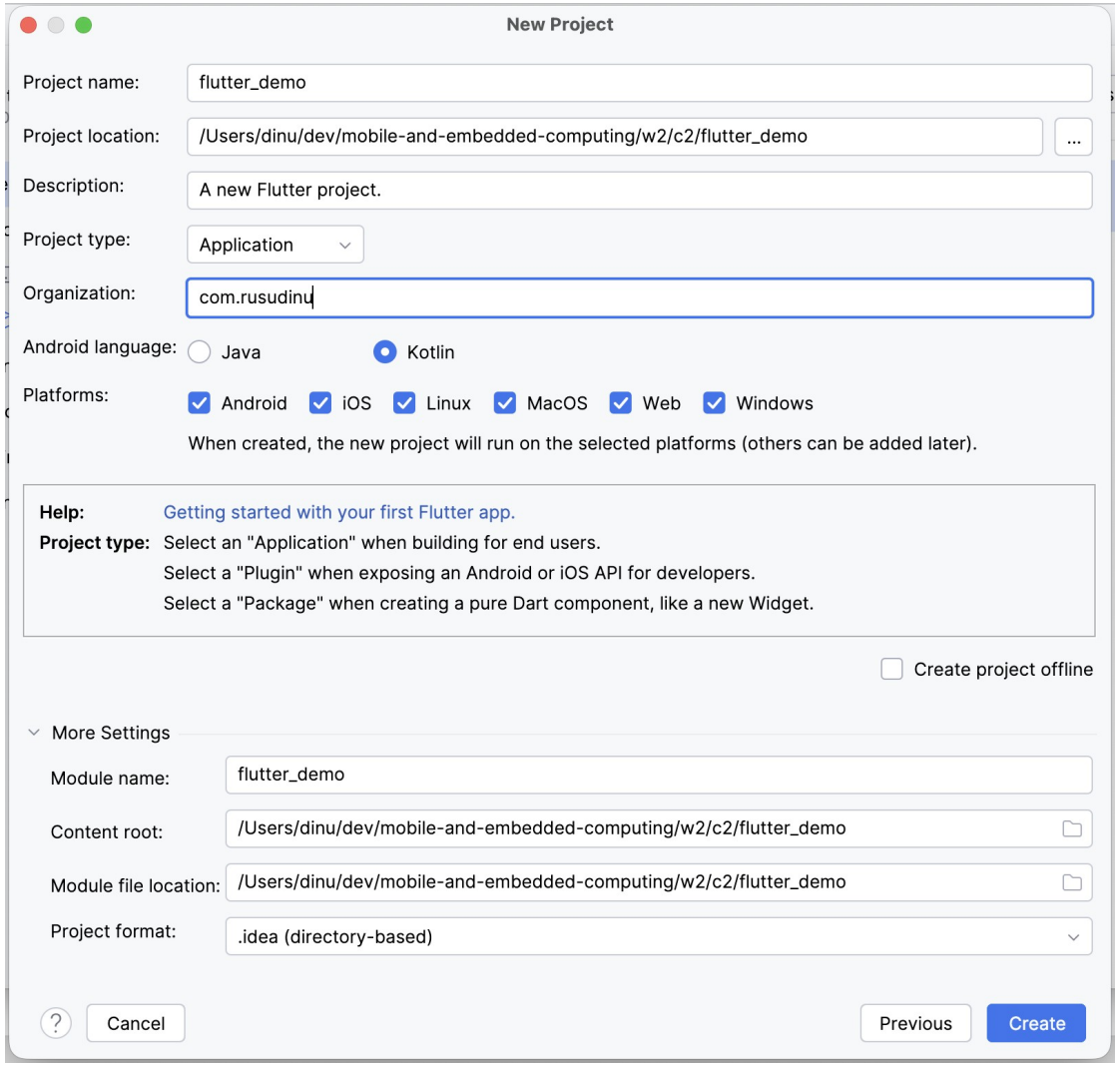
- Google's open-source UI toolkit for building natively compiled applications across multiple platforms from a single codebase
- Uses the Dart programming language and provides a rich set of pre-designed widgets that follow platform-specific design guidelines
- Excels at creating cross-platform mobile applications (iOS and Android) with almost native performance
- Well-suited for apps requiring custom UI designs, complex animations, and rapid prototyping
- Supports web and desktop development
- Hot reload
- Widget-based architecture
- Not be ideal for applications requiring extensive platform-specific integrations, heavy reliance on native device APIs, or projects where app size is critical (Flutter apps tend to be larger than native

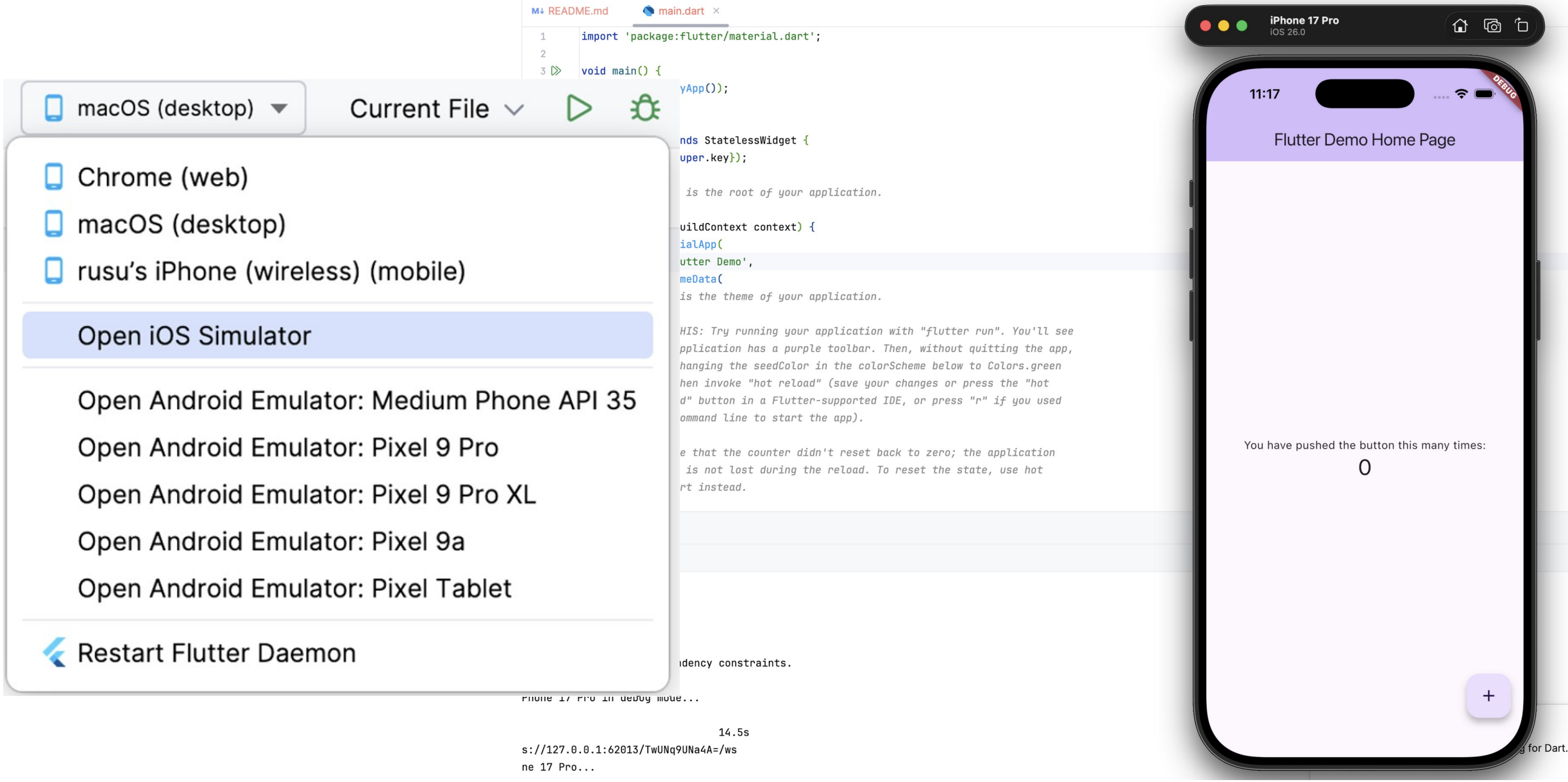


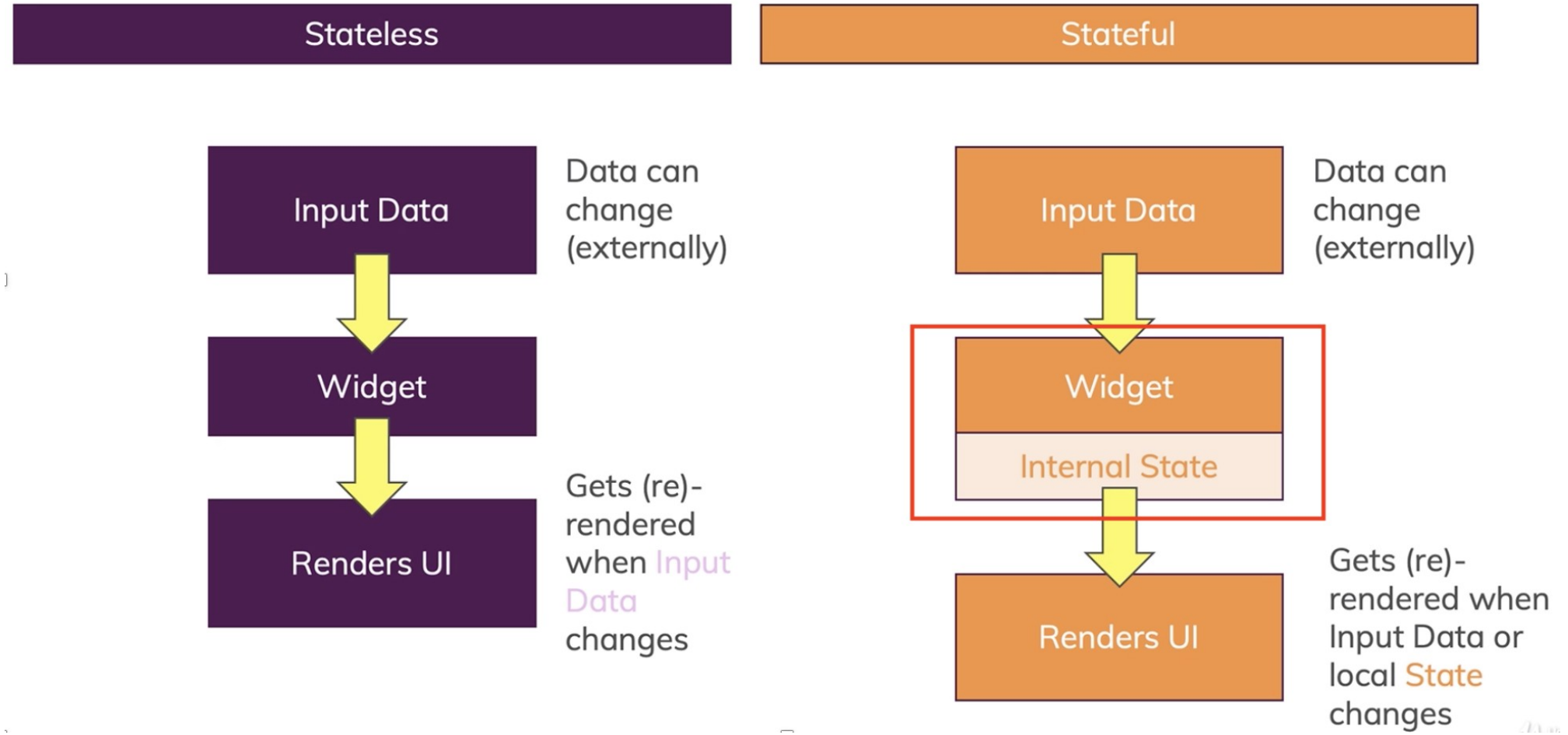
Flutter and your development environment:

- Flutter SDK (https://docs.flutter.dev/install) (click on Install manually)
- Android Studio / Xcode
- IntelliJ or VS Code to write the code (with the Flutter and Dart plugins)


New Project
Flutter SDK path: /opt/homebrew/Caskroom/flutter/3.29.1/flutter
Help: Getting started with your first Flutter app.
Spring Boot
JavaFX
Quarkus
Micronaut
Jakarta EE
Ktor
Play
HTML
React
Express
Angular CLI
App Engine
Dart
Android
Vue.js
Vite
Prolog
Nuxt
LaTeX
Flutter
Flask
FastAPI
More via plugins...
Cancel
Next


New Project
Project name: flutter_demo
Project location: /Users/dinu/dev/mobile-and-embedded-computing/w2/c2/flutter_demo
Description: A new Flutter project.
Project type: Application
Organization: com.rusudinu
Android language: Java Kotlin
Platforms: Android iOS Linux MacOS Web Windows
When created, the new project will run on the selected platforms (others can be added later).
Help: Getting started with your first Flutter app.
Project type: Select an "Application" when building for end users.
Select a "Plugin" when exposing an Android or iOS API for developers.
Select a "Package" when creating a pure Dart component, like a new Widget.
Create project offline
More Settings
Module name: flutter_demo
Content root: /Users/dinu/dev/mobile-and-embedded-computing/w2/c2/flutter_demo
Module file location: /Users/dinu/dev/mobile-and-embedded-computing/w2/c2/flutter_demo
Project format: .idea (directory-based)
Cancel
Previous
Create


README.md
main.dart
import 'package:flutter/material.dart';
void main() {
macOS (desktop)
Current File
Chrome (web)
macOS (desktop)
rusu's iPhone (wireless) (mobile)
Open iOS Simulator
Open Android Emulator: Medium Phone API 35
Open Android Emulator: Pixel 9 Pro
Open Android Emulator: Pixel 9 Pro XL
Open Android Emulator: Pixel 9a
Open Android Emulator: Pixel Tablet
Restart Flutter Daemon
iPhone 17 Pro
iOS 26.0
11:17
DEBUG
Flutter Demo Home Page
You have pushed the button this many times:
0
+



Stateless vs Stateful

Flutter Widget Lifecycle