Flutter Isolates

A guide to building responsive Flutter apps using parallel processing.

J by Jay Patil

What Are Isolates?

Definition

Isolates are Dart's version of threads. Each isolate has its own memory heap.

They run independently with no shared memory, preventing race conditions and deadlocks.

Buuuuut, you CAN share results using ports.

Compute Function

The **compute** function creates a temporary isolate for one-time tasks.

- Perfect for quick, isolated calculations
- Automatically handles isolate disposal
- Simpler API for beginners

Run Method

This method spawns an isolate, passes a callback to the spawned isolate to start some computation, returns a value from the computation, and then shuts the isolate down when the computation is complete.

- Better for long-running background tasks
- Supports custom error handling
- More flexible communication patterns

Most basic way to use an Isolate

Two of the most common ways to run an isolate is using .run or .compute method

```
datatype variableName = await Isolate.run(function);
```

```
datatype x = await compute(function);
```

A more controlled way..

Well if we want more control and need to update states,

`Isolate.spawn()` gives you **low-level control** over the isolate lifecycle:

- You manually pass and receive messages using `SendPort` and `ReceivePort`.
- Useful when:
  - You want to **send multiple messages**.
  - You want to **keep the isolate alive**.
  - You want to handle **errors, timeouts**, or communication back and forth.

```
await Isolate.spawn((SendPort sendPort) {
  final result = _computeFibonacci(n);
  sendPort.send(result);
}, receivePort.sendPort);
```

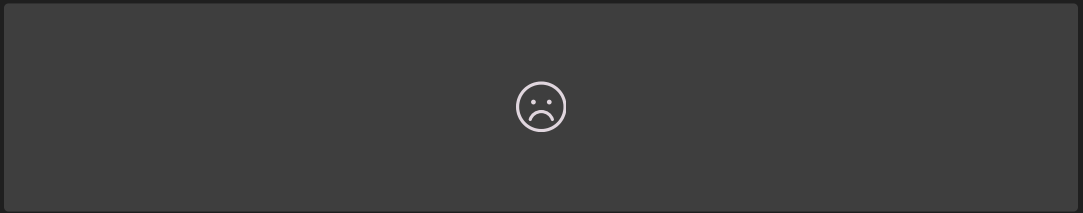
The Problem: Blocking the UI Thread

Heavy computation starts → Image processing, JSON parsing, Math Computations

CPU resources dedicated to calculation

UI thread blocked

No resources for rendering or input



App becomes unresponsive

Users experience lag and freezing

Fibonacci Seqence

```
int computeFibonacci(int n) {
  if (n <= 1) return n;
  return computeFibonacci(n-1) +
         computeFibonacci(n-2);
}
```

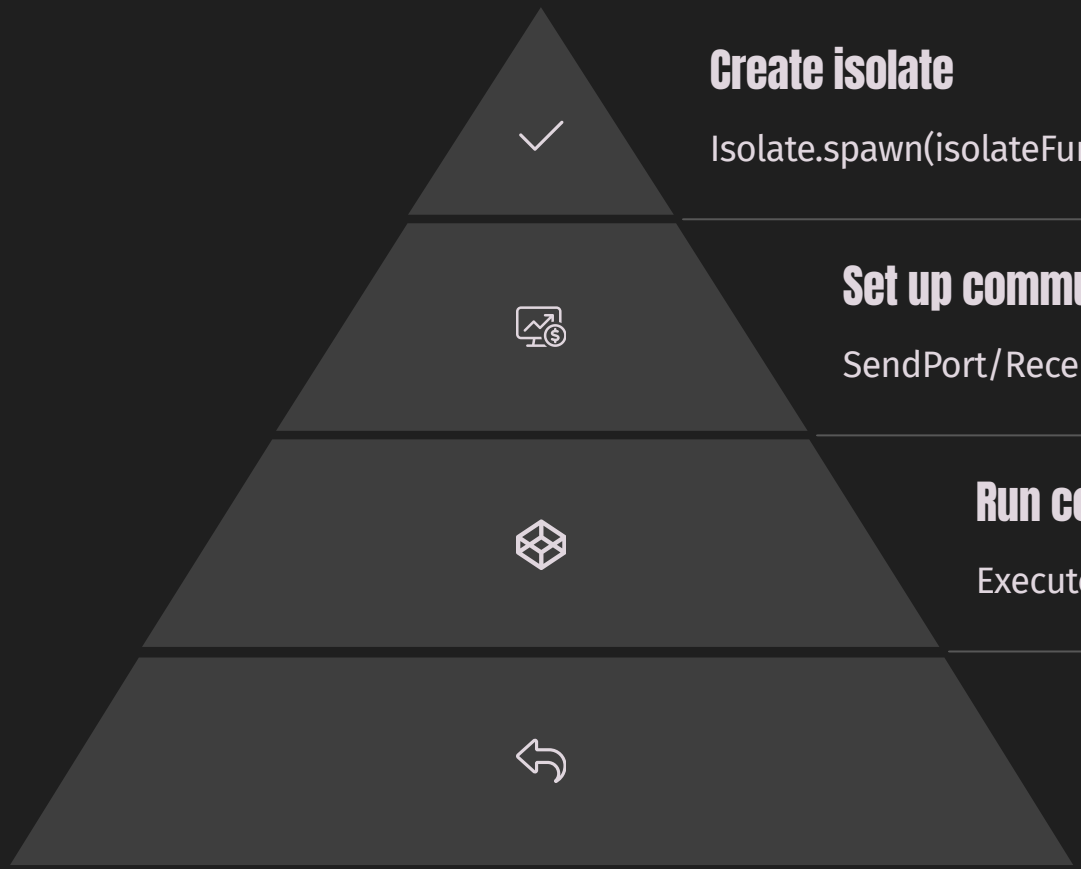
Exponential complexity

O(2^n) time complexity

Blocking computation

Freezes UI at n > 40

The Solution: Isolate Implementation



Create isolate

Isolate.spawn(isolateFunction, sendPort)

Set up communication

SendPort/ReceivePort for message passing

Run computation

Execute heavy code in separate memory space

Return result

Send result back to main thread

Implementing UI Integration

UI Components

- Slider to select Fibonacci number
- Two buttons: Main Thread vs Isolate
- Result card with computed value
- Loading indicator during calculation

Expected behavior

When running on the Main Thread (Main Isolate) the UI freezes for large input of fibonacci number

We can mitigate this by running it on an Isolate, so that the main UI keeps working properly. Then the isolate returns the value.

When to Use Isolates

Perfect For

- Parsing large JSON files
- Image processing and filtering
- Complex calculations and algorithms
- Network data processing

Not Needed For

- Simple UI updates
- Basic calculations
- Quick API calls
- Small data transformations

For small stuff (1+1?) its better to NOT use isolates because of the overhead it causes.

Best Practices

- Keep isolates alive for reuse
- Minimize message size between isolates
- Handle errors appropriately
- Consider compute() for simple cases