// @jamescardona11

Decoding isolates

Content

intro

who has used isolates?

what is the goal of this talk?

who is this for?

- I’m: James Cardona
- Role: Mobile developer
- Company: phononx.com
- LinkedIn/Github/Medium: @jamescardona11
- Web: j11.io

wwh

what

- concurrent processing
- thread wrapper
- actor model

why

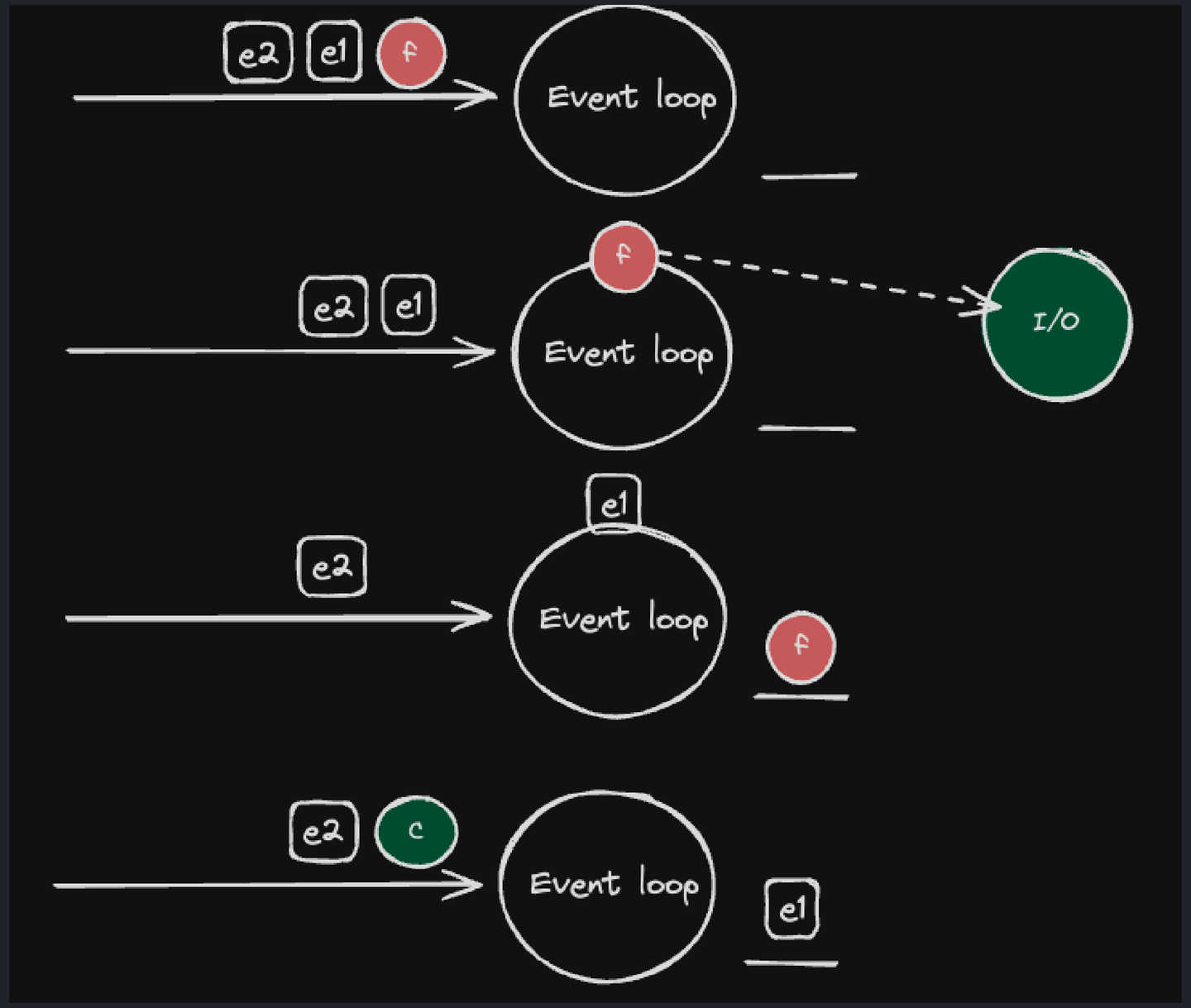
- event loop
- async - await
- event handling

how

- compute/ run
- spawn

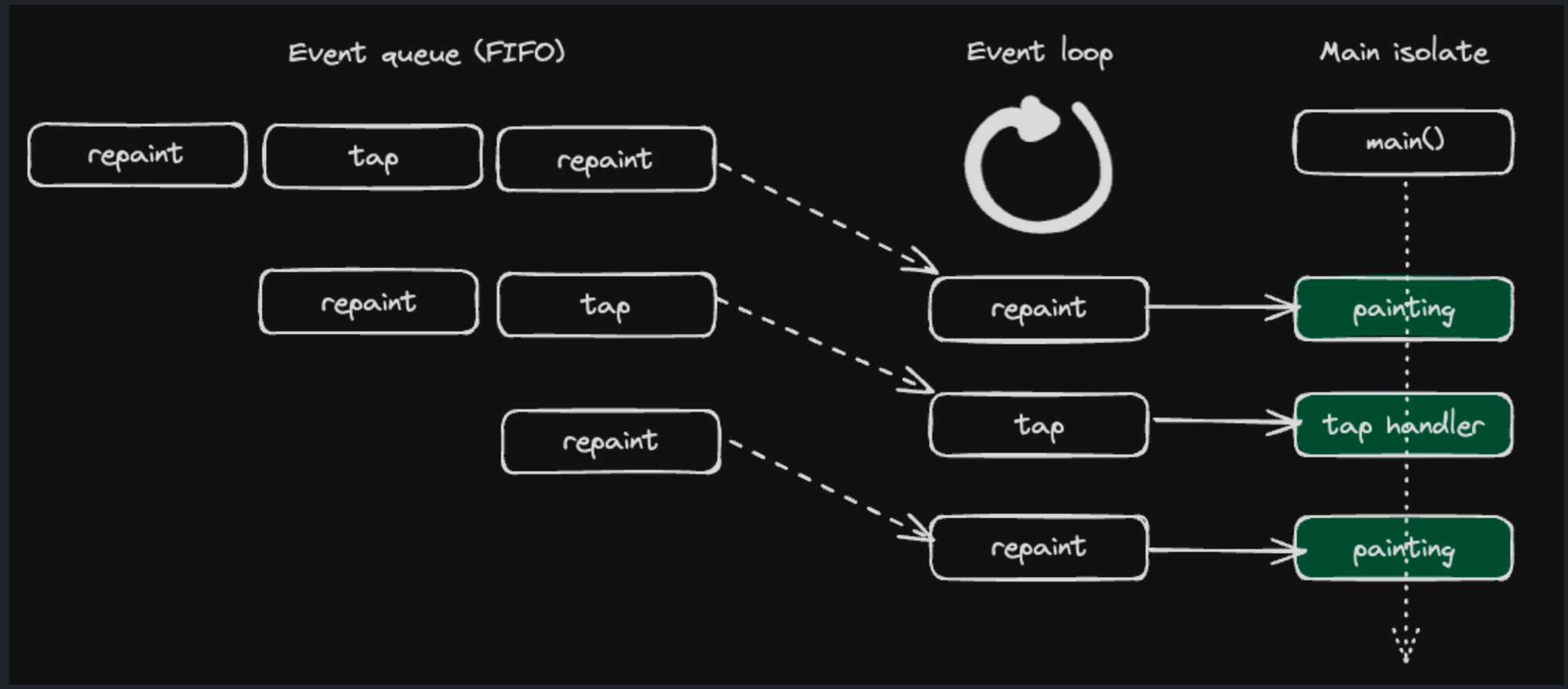
why

- event loop
- async - await
- event handling



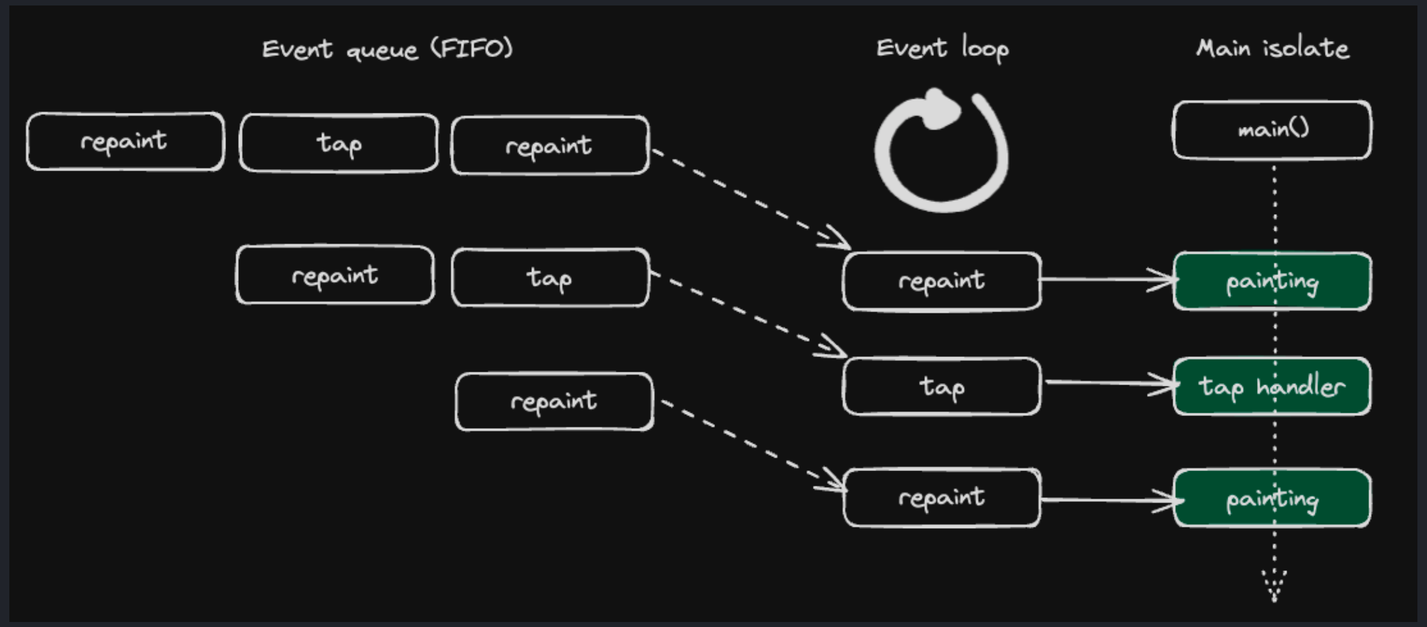
why

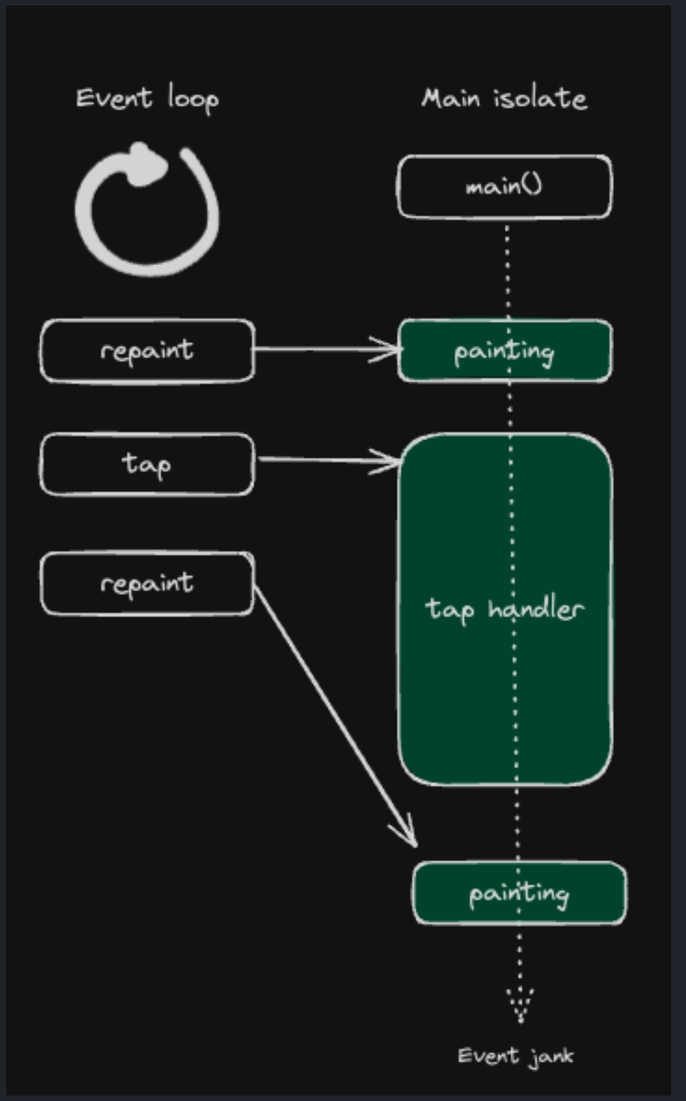
- event loop
- async - await
- event handling



why

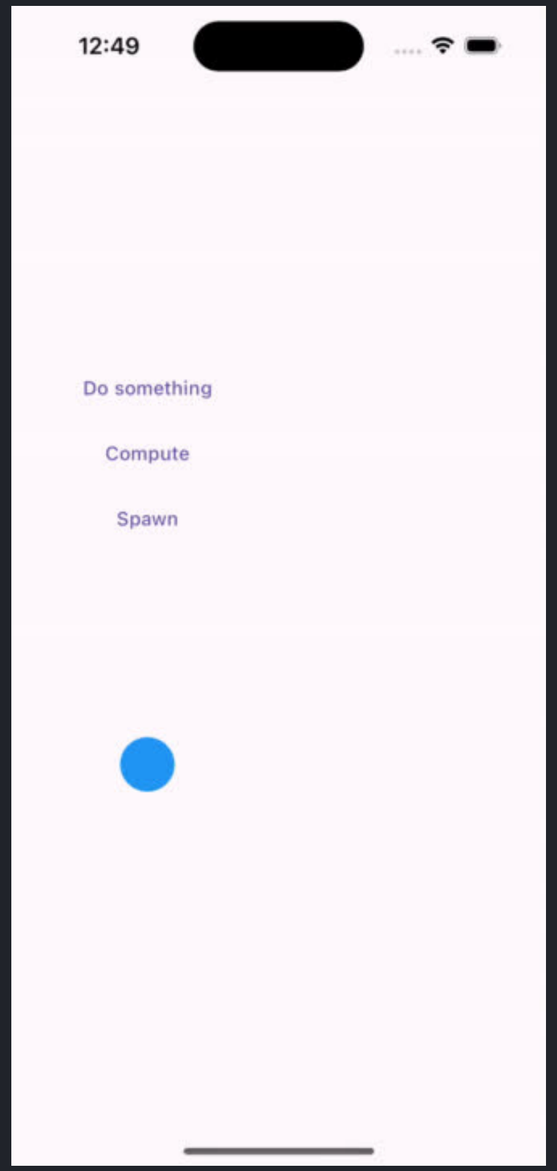
- event loop
- async - await
- event handling





why

- event loop
- async - await
- event handling



wwh

what

- concurrent processing
- thread wrapper
- actor model

why

- event loop
- async - await
- event handling

- Concurrency
- Isolation/memory
- Parallelism
- Communication
- Heavy task

how

- compute/ run
- spawn

how

- compute/ run
- spawn

compute/run

```dart
void computeIsolate() {
  print('Compute');
  compute(doSomething, 1000000000);
}

void runIsolate() {
  print('Run');
  Isolate.run(() => doSomething(1000000000));
}
```

how

- compute/ run
- spawn

Spawn

```
void spawn() async {
    print('Spawn');
    final rcvPort = ReceivePort();

    final isolate = await Isolate.spawn(_doSomethingForSpawn, rcvPort.sendPort);

    final completer = Completer<SendPort>();
    rcvPort.listen((message) {
      if (message is SendPort) completer.complete(message);

      print(message);

      if (message is! SendPort) {
        rcvPort.close();
        isolate.kill();
      }
    });

    final send2Isolate = await completer.future;
    send2Isolate.send(1000000000);
  }
```

Content

01 ~~intro~~

02 ~~what, why, how?~~ 5 -8 min

03 wrapper/controller

- Actor Model
- Communicate
- Bidirectional (demo)
- Controller

04 demo time

05 questions?

Actor model

Concurrency models

- Processes
- Threads
- Futures
- Coroutines
- Actor
- etc

what is?

encapsulates state and behavior

??

??

??

??

??

??

??

??

??

??

Actor model

- Actors are persistent.
- Encapsulate internal state (Private).
- Actors are asynchronous.
- Communication through messages.
- Independence between actors.
- Supervision.

what is?

encapsulates state and behavior

key concepts

features

??

??

??

??

??

??

??

??

Actor model

- Create new actors.
- Send messages to other actors.
- Receive messages and in-responses.
- Process exactly one message at a time.

"Do not communicate by sharing memory; instead, share memory by communicating" | Effective Go

what is?

encapsulates state and behavior

key concepts

features

what can actors do?

functions

??

??

??

??

??

??

Actor model

- NO channels or intermediaries.
- “best effort” delivery.
- Messages can take an arbitrarily long time to deliver.
- No message ordering guarantees.

what is?

encapsulates state and behavior

key concepts

features

what can actors do?

functions

properties of communication

safe

??

??

??

??

Actor model

- Each actor has an address.
- Actors can communicate with other actors using their addresses (Use SendPort to send messages).
- The actor receives addresses from other actors in messages (Listen ReceivePort).
- One actor can have more than one address.
- Address != identify; this means two actors with the same identity can have different addresses.

what is?

encapsulates state and behavior

key concepts

features

what can actors do?

functions

properties of communication

safe

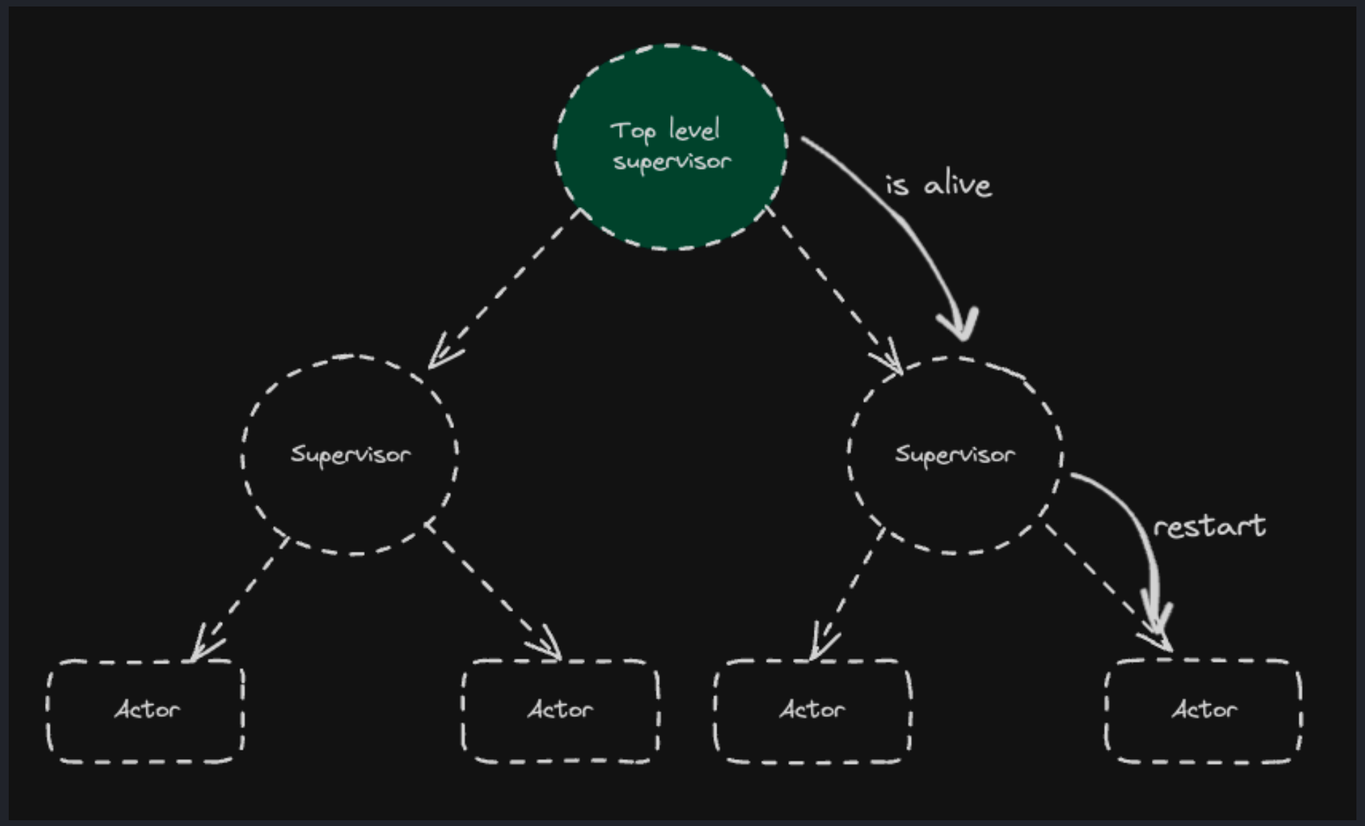
address

mailbox

??

??

Actor model



what is?

encapsulates state and behavior

key concepts

features

what can actors do?

functions

properties of communication

safe

address

mailbox

supervision

state

Actor Model

Isolates similarities

- ReceiverPort is a similar concept to Actor Mailbox.
- The Mailbox is a message queue
- SendPort is a similar address concept in the actor.

Isolates vs Threads

- Isolation of memory
- Lightweight
- Safe concurrency

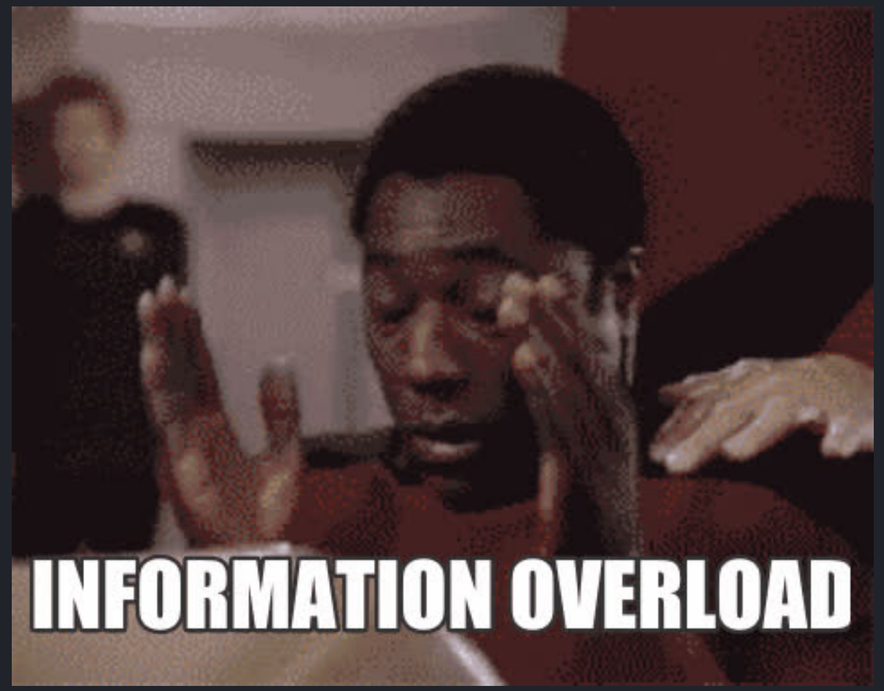
Actor Model

Isolates similarities

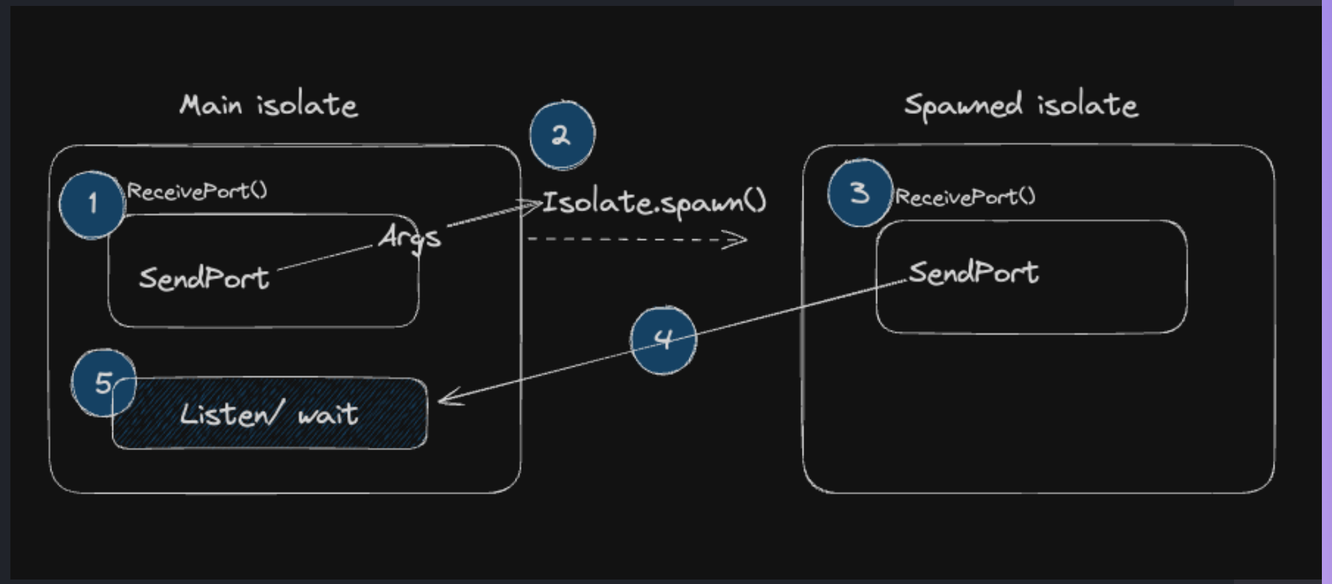
- ReceiverPort is a similar concept to Actor Mailbox.
- The Mailbox is a message queue
- SendPort is a similar address concept in the actor.

Isolates vs Threads

- Isolation of memory
- Lightweight
- Safe concurrency



Communicate between two isolates

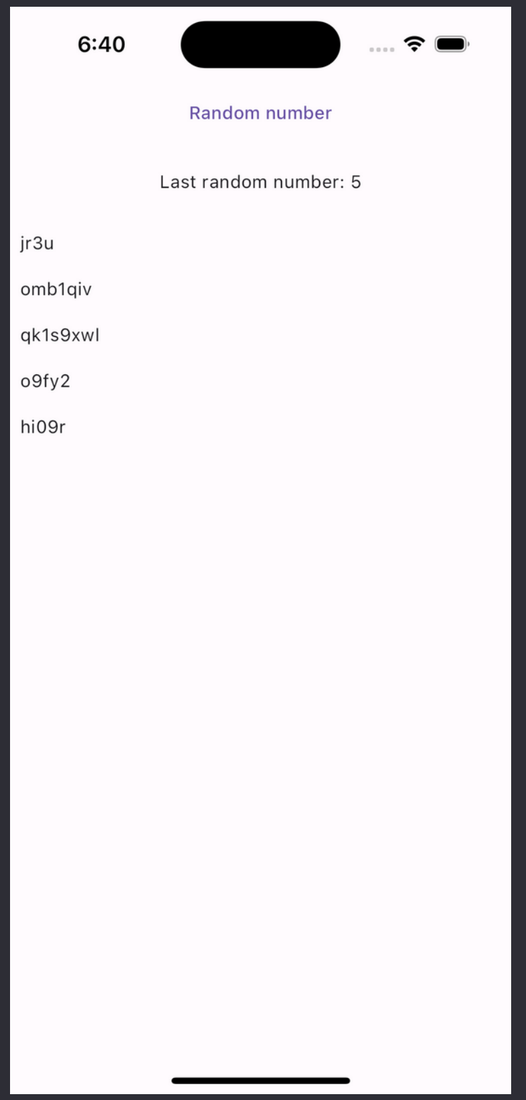



```dart
void spawn() async {
  print('Spawn');
  final rcvPort = ReceivePort(); // --> Step 1

  // --> Step 2
  final isolate = await Isolate.spawn(_doSomethingForSpawn, rcvPort.sendPort);

  final completer = Completer<SendPort>();
  rcvPort.listen((message) {
    // --> Step 5
    if (message is SendPort) completer.complete(message);

    if (message is! SendPort) {
      rcvPort.close();
      isolate.kill();
    }
  });

  final send2Isolate = await completer.future;
  send2Isolate.send(1000000000);
}

void _doSomethingForSpawn(SendPort sendPort) {
  // --> Step 3
  final rcvPort = ReceivePort();

  // --> Step 4
  sendPort.send(rcvPort.sendPort);

  rcvPort.listen((bigNumber) {
    var sum = 0;
    for (var i = 0; i <= bigNumber; i++) {
      sum += i;
    }

    sendPort.send(sum);
  });
}
```

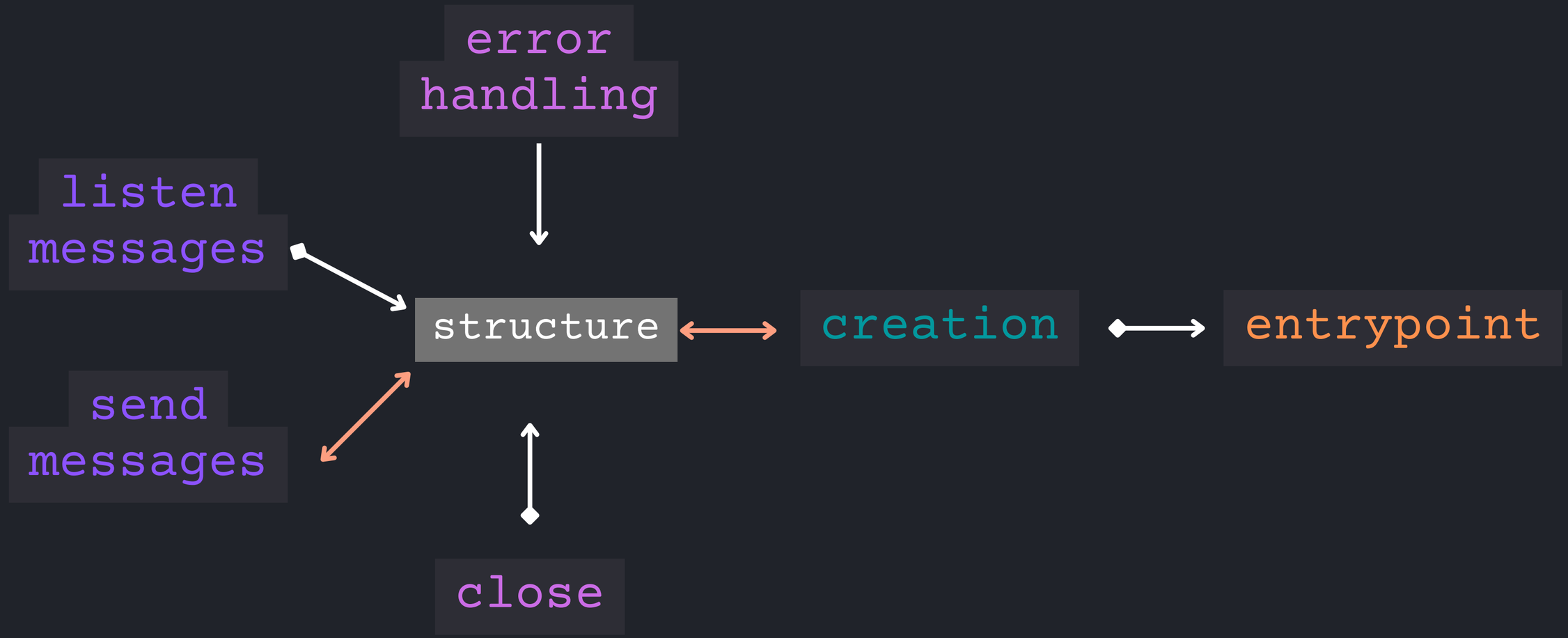
bidirectional

best practices with isolates

- Error handling: Always handle errors in isolates to prevent crashes
- Kill isolates: Always kill isolates when they’re no longer needed.
- Limit communication: Avoid sending large objects between isolates.



wrapper/controller



controller v1

```dart
class IsolateController<T> {
  final Isolate _isolate;
  final ReceivePort _receivePort;
  final Stream<dynamic> _broadcastRp;
  final SendPort _sendPort;

  static Future<IsolateController<T>?> create<T>();

  Stream<dynamic> get broadcastRp;

  void send(T message);

  void close();
}
```

controller v2

```dart
class IsolateController<I, O> {
  final SendPort _commands;
  final ReceivePort _responses;

  final StreamController<O> _controller =
      StreamController<O>.broadcast();
  late final StreamSubscription<O> _subscription;

  static Future<IsolateController<I, O>?> create<I, O>();

  Stream<O> get broadcastRp;

  void send(I message);

  void dispose();
}
```

Content

01 ~~intro~~

02 ~~what, why, how?~~ 5-8 min

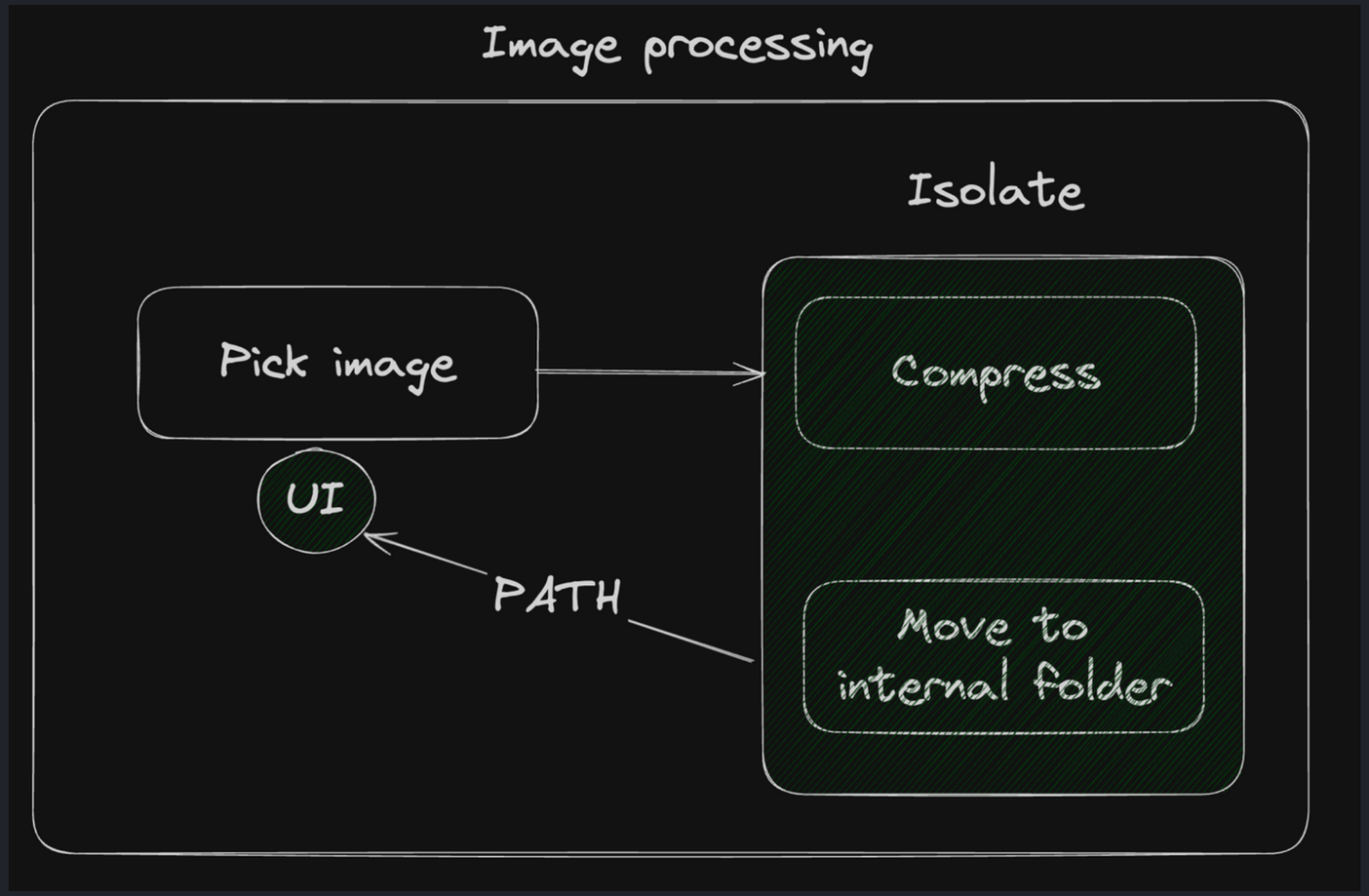
03 ~~wrapper/controller~~

- ~~Actor Model~~
- ~~Communicate~~
- ~~Bidirectional (demo)~~
- ~~Controller~~

04 demo time 25 min

05 questions?

demo



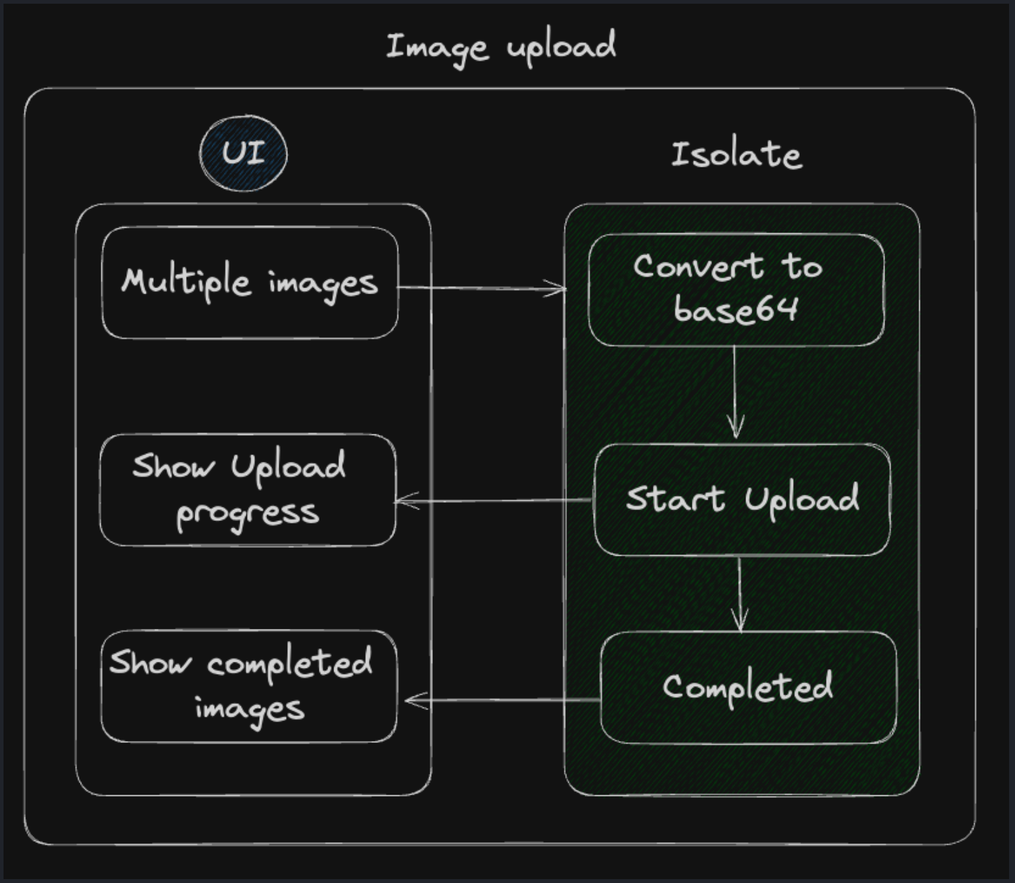
problem

Create a solution to compress an image; move it to the internal folder of the device and show the compressed image.

solution

- Use a compute/Isolate.run because is a single operation
- Pick an image from the UI (MainIsolate) and send the image path to the Isolate
- Use a compress algorithm
- Move the isolate to the internal folder
- Return the new Path to the Main Isolate

demo



problem

Upload the images to the server as a String in Base64; you can pick the images and go to another screen; the upload will be kept in the background and all screens need to listen to the updates.

solution

- Use a Spawn Isolate to send images to Isolates
- Starts uploading images as you select images
- Inform when the upload Start
- Inform when the Image completes the Upload
- Change the Image state—Uploading/ Completed
- Show progress in the UI (How many images are left?)
- Use a provider to listen and send messages to the isolate.

// @jamescardona11

Gracias {}

- code/presentation:https://github.com/jamescardona11/isolates
- posts: https://medium.com/@jamescardona11
- linkedIn: @jamescardona11
- web: j11.io