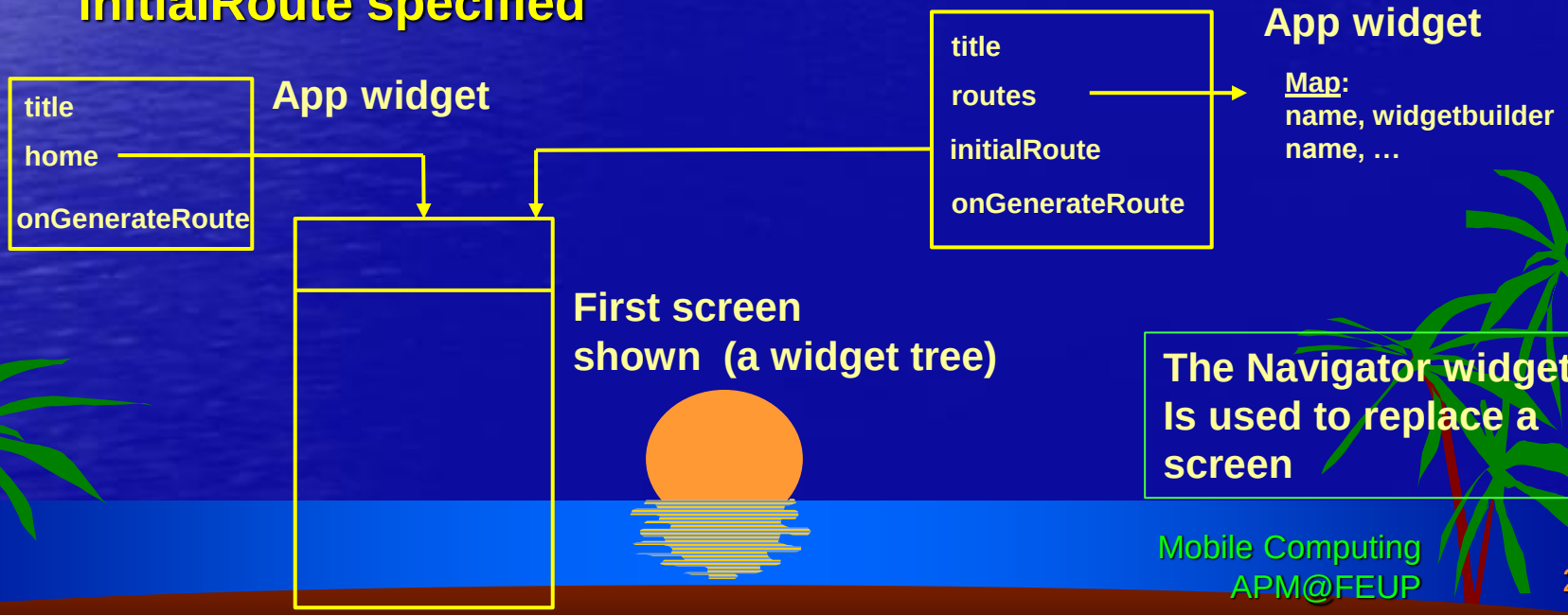
Mobile Computing

Flutter
Pages and Navigation
Drawing and External Calls

Flutter Pages

- Page or screen is a Widget tree aka a Route
  - The App widget defines the home route and others, from
    - Home, routes, and initialRoute properties (on the App constructor)
  - The home is the initial screen (widget tree) shown
    - Alternatively, a set of routes (pages) can be defined and an initialRoute specified



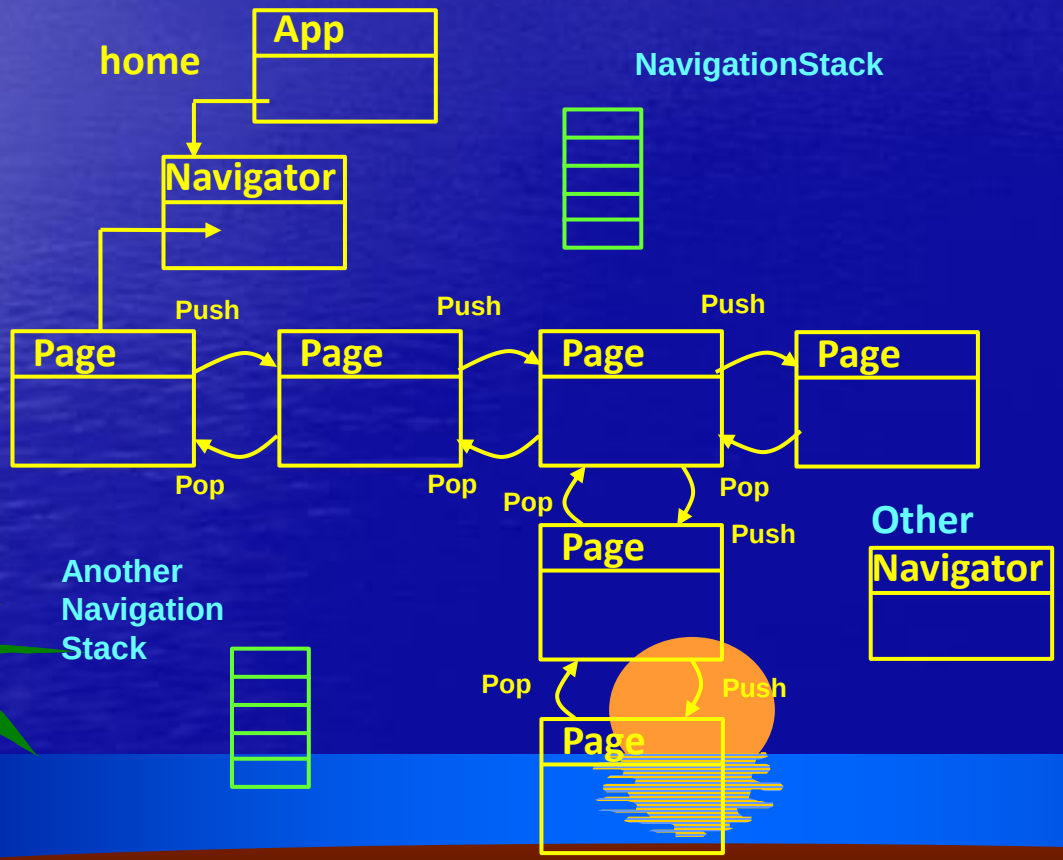
Navigation

The Navigator widget allows the replacement of a page by another using a stack discipline. It is possible to create a set of navigation routes previously in the app, or build each one when we want to navigate to it. Initially the App has already a Navigator. We can use it or create a new independent one with new routes.

The Navigator defines a set of static methods that manipulate the stack of pages (routes):

push(context, Route) and pop(context)

pushNamed(), removeRoute(), replace(), popAndPushNamed(), …



Routes and new screens

New screens or pages (widgets trees) can be built from a PageRoute (derived from) e.g., the MaterialPageRoute, requires a WidgetBuilder (a function) that builds the tree from context

```
// within any widget built from context        The builder function must return a widget (tree)
…
Navigator.push(context,
      MaterialPageRoute(
                builder: (context) => MyPage1(title: ‘page A’) )
);
```

or

Prebuild a route table in the App widget and put it in the routes property

```
void main() {
 runApp(MaterialApp(
   home: MyAppHomePage(),                    // becomes the route named '/'
   routes: <String, WidgetBuilder> {
     '/a': (BuildContext context) => MyPage1(title: 'page A'),
     '/b': (BuildContext context) => MyPage2(title: 'page B'),
     '/c': (BuildContext context) => MyPage3(title: 'page C'),
   },
 ));
 }
```

Navigate forward

```
// within a widget
…
Navigator.pushNamed(context, '/b');
```

Navigate backward

```
// within the navigated page
…
Navigator.pop(context);
```

Passing data on a route

A Named Route can transport arguments. They can be extracted on the destination page.

A class for the arguments can be defined and built.

```
class ScreenArguments {
  final String title;
  final String message;

  ScreenArguments(this.title, this.message);
}
```

```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ….
        ….
          onPressed:  () {
            Navigator.pushNamed(context, '/page2',
              arguments: ScreenArguments('Title', 'Message'));
            }
      ….
    }
}
```

```
// App root widget
…
return MaterialApp(
  routes: { '/page2': (context) => SecondPage() },
  home: HomePage()      // route name: '/'
)
…

class SecondPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final ScreenArguments args = ModalRoute.of(context).
                    settings.arguments;
    return ….
          … args.title …
          … args.message …
  }
}
```

Return a value using pop

When we do a Navigator.pop(context), pop() accepts a second parameter that is a result. It can be of any type.

This result is returned by the Navigator.push(…) that created the popped page. The result comes in a Future, resolved only when the page is popped.

```
…
final result = await Navigator.push(context, MaterialPageRoute(builder: (context) => SomePage()));
…

ScaffoldMessenger.of(context)                    // Show a SnackBar with the result
  ..removeCurrentSnackBar()
  ..showSnackBar(SnackBar(content: Text('$result'));
…
```

```
class SomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
      …
      onPressed: () {
          Navigator.pop(context, 'Some message');
      }
      …
  }
}
```

Mobile Computing

Flutter
Drawing in a Canvas

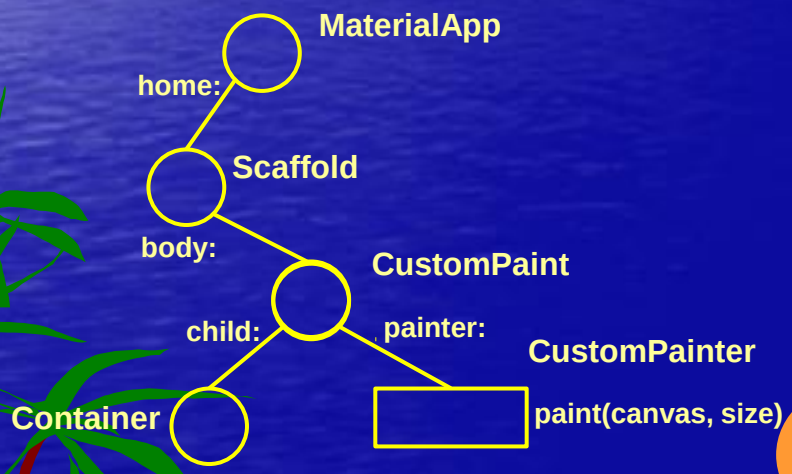
Using a CustomPaint widget

```
Import 'dart:ui';
Import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Custom Painter',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: MyPainter()
    )
  }
}
```

```
class MyPainter extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Lines')),
      body: CustomPaint(
        painter: ShapePainter(),
        child: Container()
      )
    )
  }
}
```




```
class ShapePainter extends CustomPainter {
  @override
  void paint(Canvas canvas, Size size) {
    ....
  }

  @override
  bool shouldRepaint(CustomPainter oldDelegate) {
    return true;
  }
```

Mobile Computing

External REST calls

Calling an external service

Flutter does not have a direct API for web requests.
But there are several packages in the package repository (pub.dev).
One of the simplest is the http package (https://pub.dev/packages/http)

For using external packages, it is necessary to import them in code, and to declare them in the project pubspec.yaml file. Finally, it must be downloaded.

The declaration and installation can be done from the command line, or from the used IDE:

```
$ flutter pub add http
```

The .yaml file will be added with the line:

```
dependencies:
  http: ^0.13.4                    // Last version available
```

For example, to define a function to perform a REST GET request:

```
import 'package:http/http.dart';
…
Future<String> getResponse() async {
  final response = await http.get(Uri.http('data.fixer.io', '/api/latest',
                                  { 'access_key': '<your API key>',
                                    'base': 'EUR',
                                    'symbols': 'USD,GBP' }
  ));
  if (response.statusCode == 200)
    return response.body;
  else
    throw Exception('HTTP failed');
}
```

Note: If the target platform requires some sort of 'permissions' for the these types of requests, they must be included in the correspondent files.