Chapter 6:

Multi-screens apps

(Routes and Navigation)

Plan

1. **Introduction to route & Navigator class.**

2. **Imperative navigation (Navigator 1.0).**

   - Un-named routing : direct navigation.
   - Named routing : static & dynamic navigation.

3. **Declarative navigation (Navigator 2.0).**

   - Why and when to use it ?
   - GoRouter package.

Plan

1. **Introduction to route & Navigator class.**

2. **Imperative navigation (Navigator 1.0).**

   - Un-named routing : direct navigation.
   - Named routing : static & dynamic navigation.

3. **Declarative navigation (Navigator 2.0).**

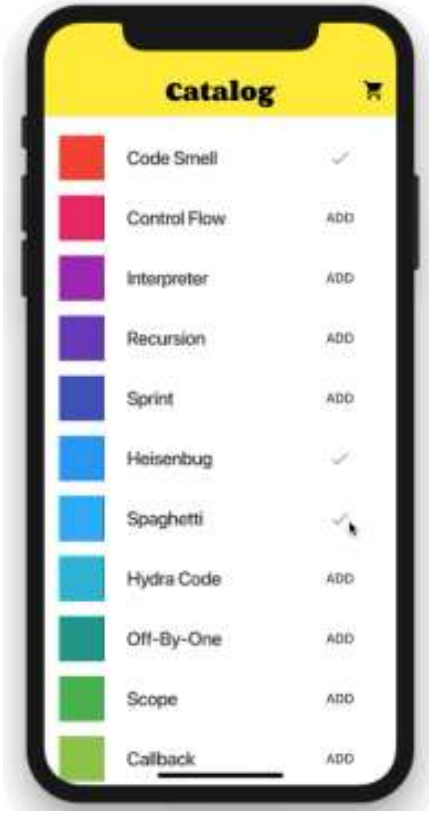
   - Why and when to use it ?
   - GoRouter package.

Introduction

- **Definitions :**

➢ **Route**: is a page or a screen in a Flutter app.

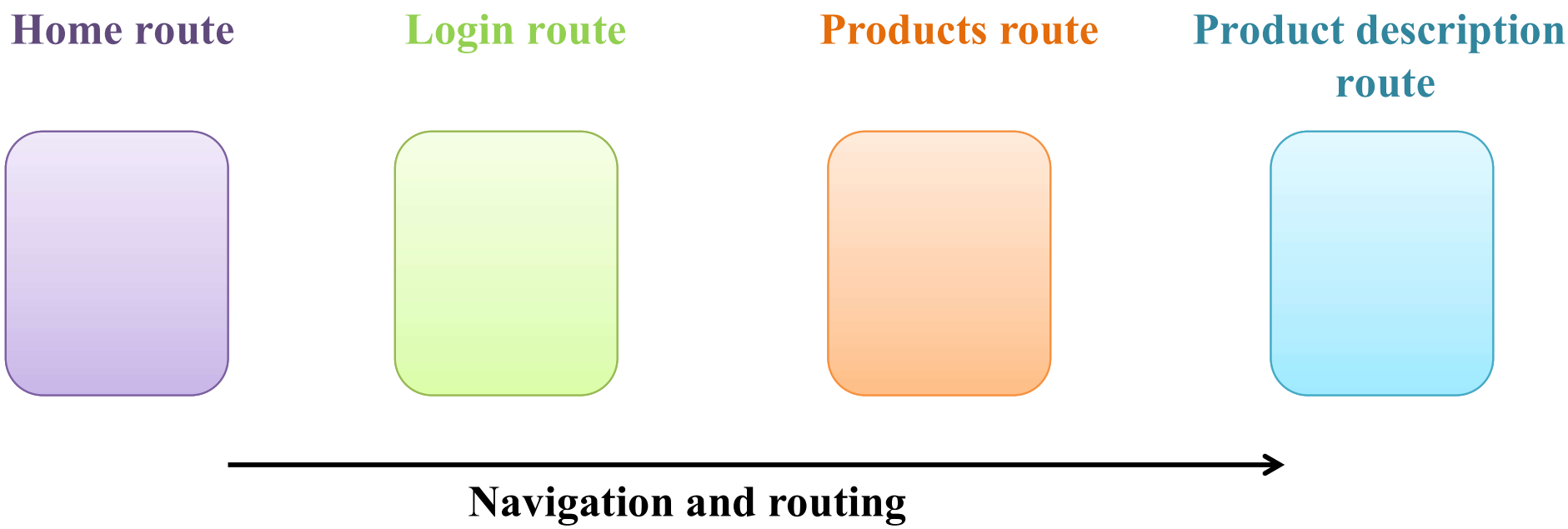
A route = A ***Widget***.



Introduction

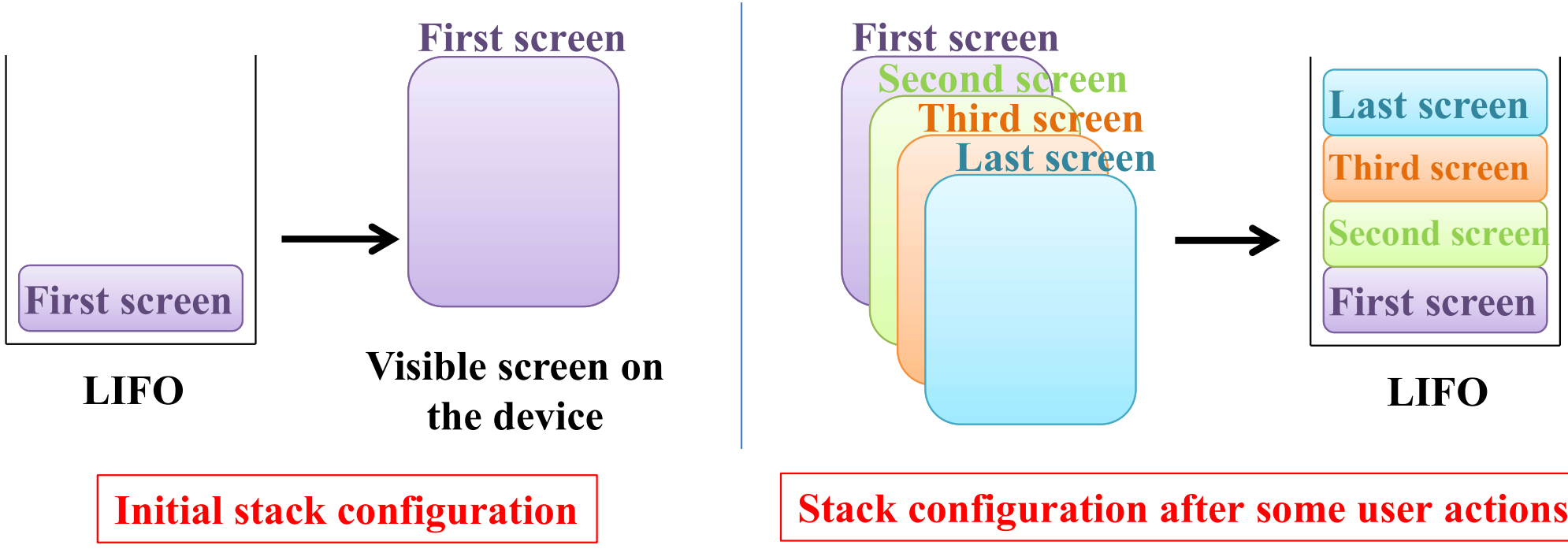
- **Definitions :**

  - **Navigator**: is a widget used to perform the process of navigating from one route to another.



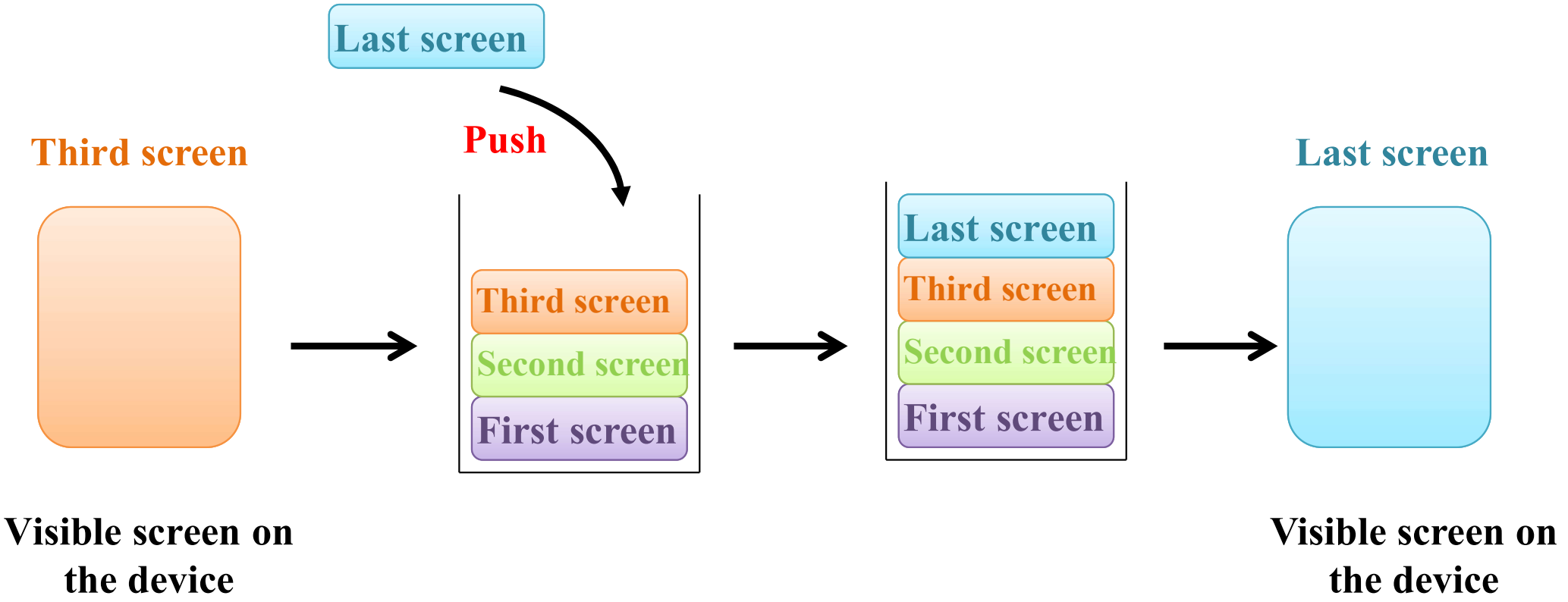
Navigator class

- The navigator is based on **stacks** discipline which positions all its child routes on top of the other.
- It arranges routes from bottom to top such as the **first route** is the most bottom screen, and the **last route** is the most top screen.



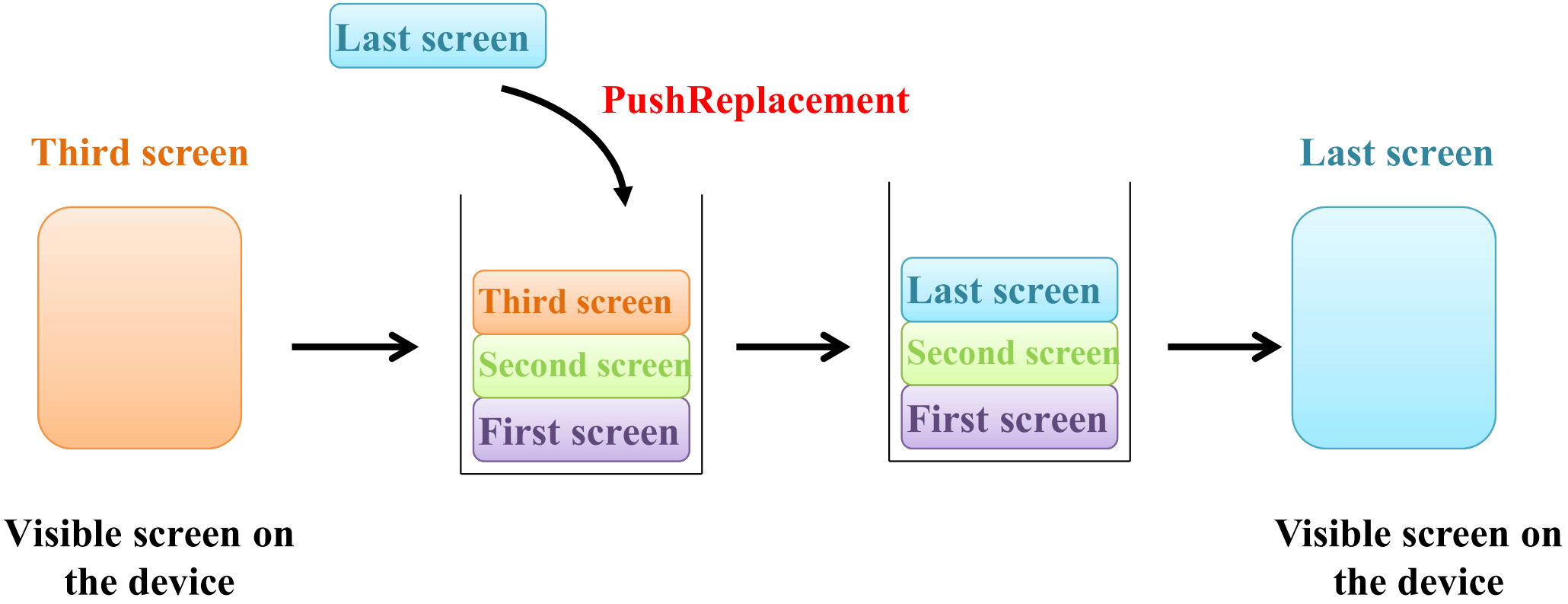
Navigator class

- **push** method adds a route to the stack of routes managed by the Navigator .



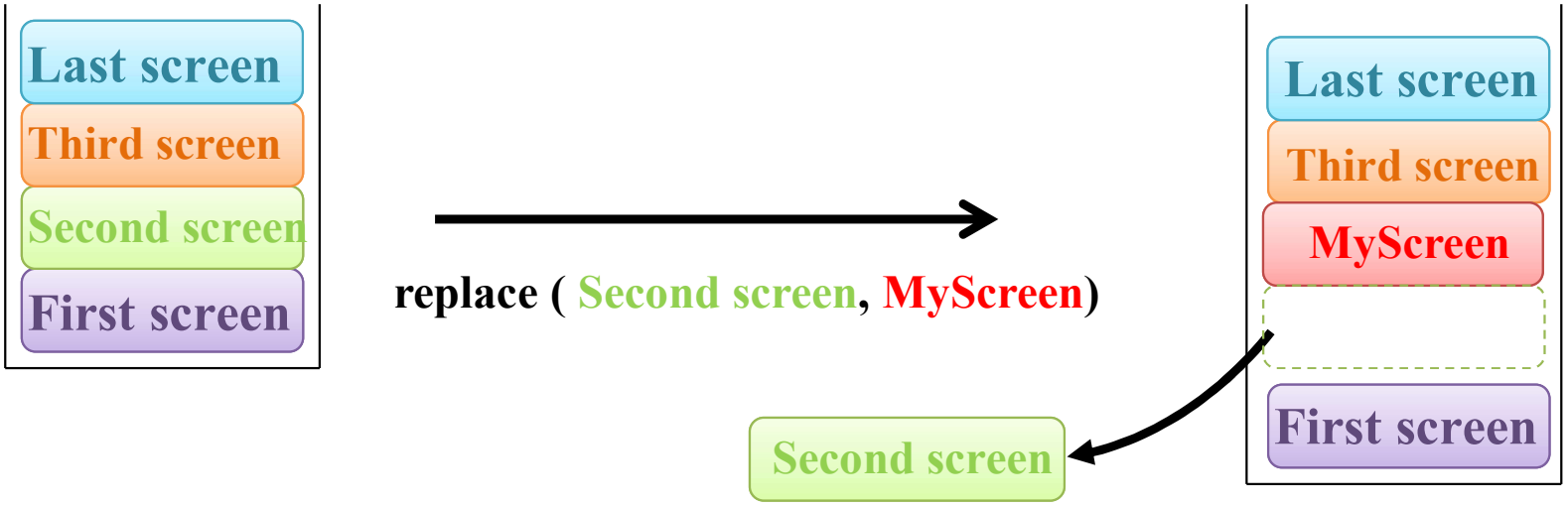
Navigator class

- **pushReplacement** method: replaces the current route of the navigator by pushing the given route.



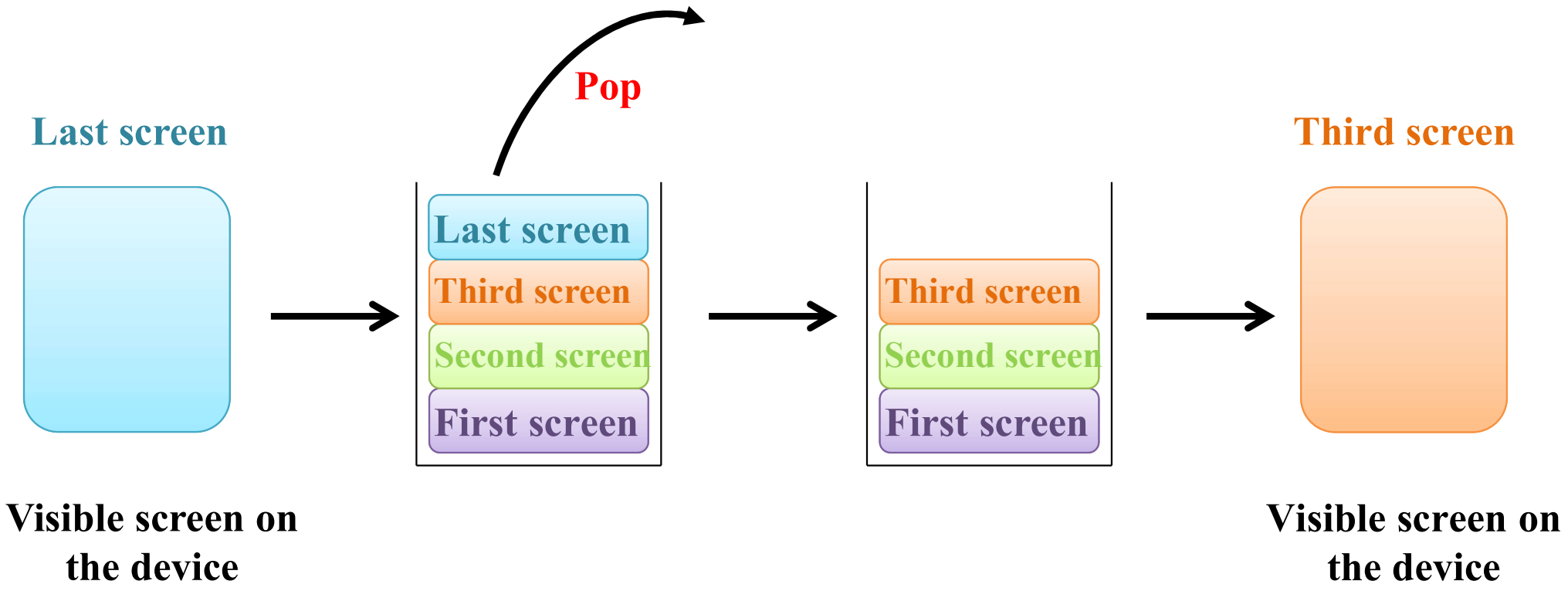
Navigator class

- **replace** method: replaces a route on the navigator with a new route.



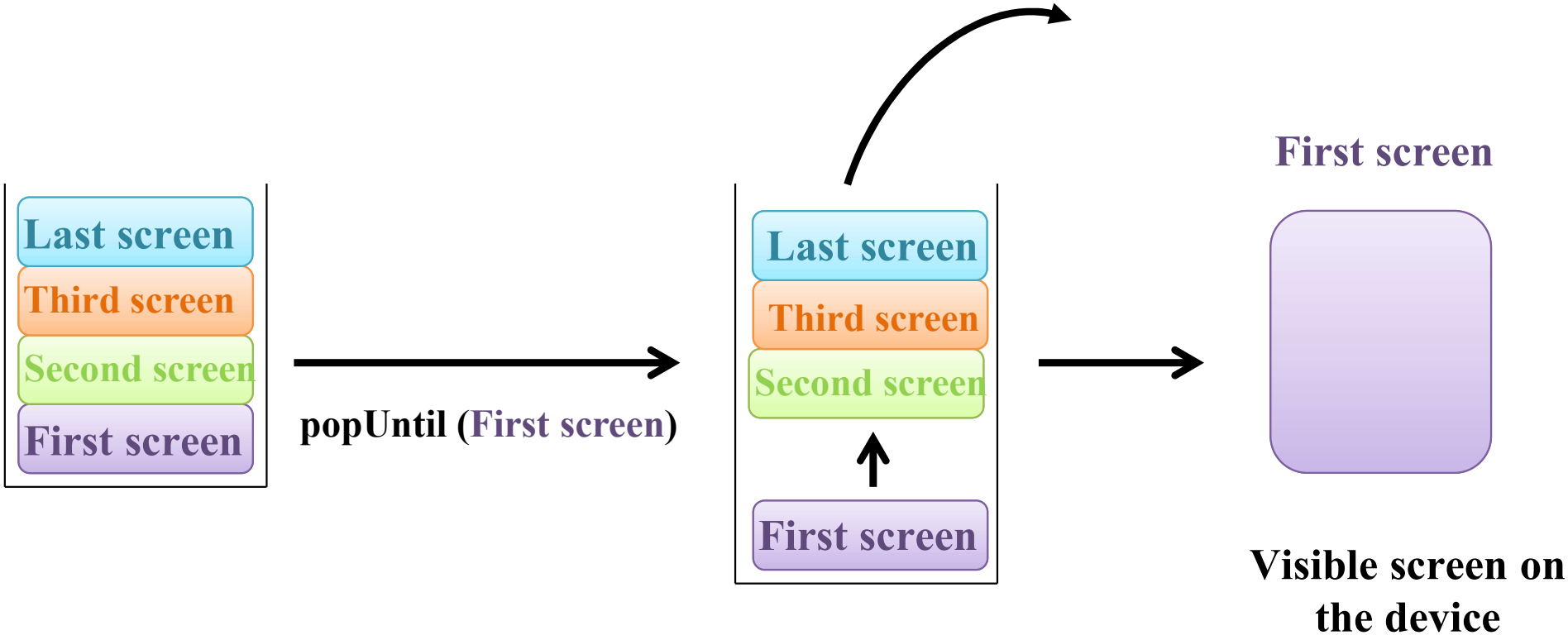
Navigator class

- **pop** method: removes the current route from the stack of routes managed by the Navigator .



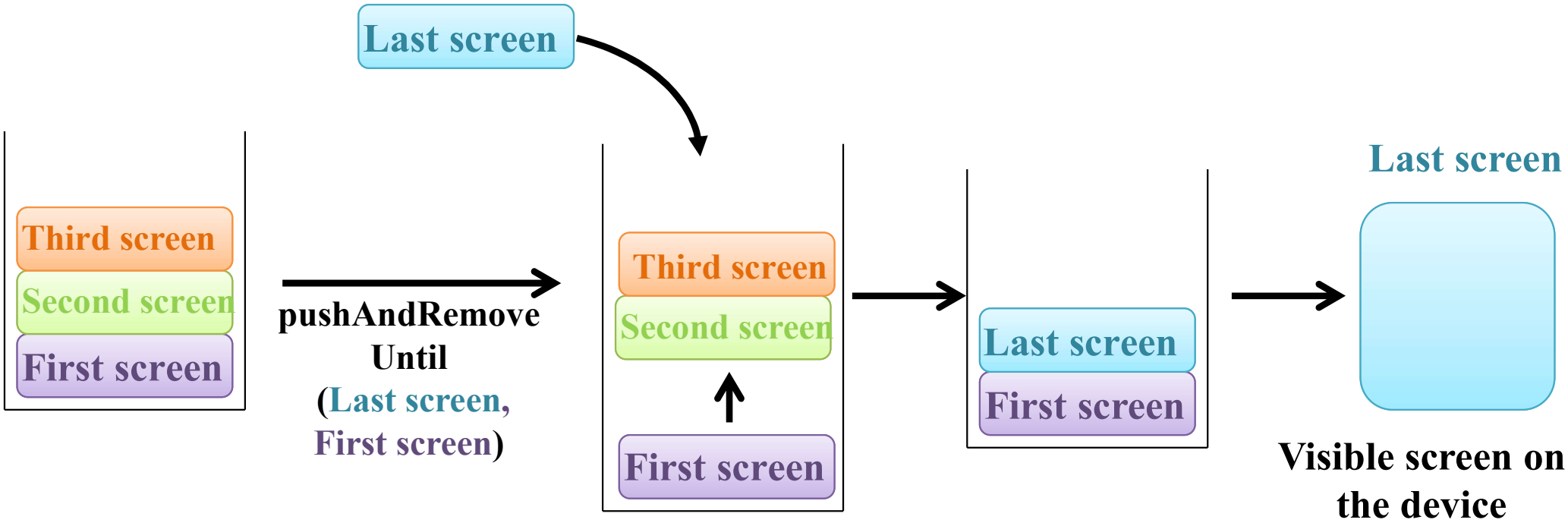
Navigator class

- **popUntil** method: calls pop method repeatedly until a certain condition (expressed by a predicate function) is met.



Navigator class

- **pushAndRemoveUntil** method: push the given route onto the navigator and then remove all the previous routes until the predicate returns true.



Types of navigation APIs

Imperative navigation
(Navigator 1.0)

- It's imperative => HOW

Taxi driver: where do you want to go ?
Passenger : go straight, turn left, then exit …

Declarative navigation
(Navigator 2.0)

- It's declarative => WHAT

Taxi driver: where do you want to go ?
Passenger : to the beach.
Taxi driver: so I need to go straight, turn left, then exit …

Navigator 2.0 does not replace Navigator 1.0 as they can used together.

Plan

1. Introduction to route & Navigator class.

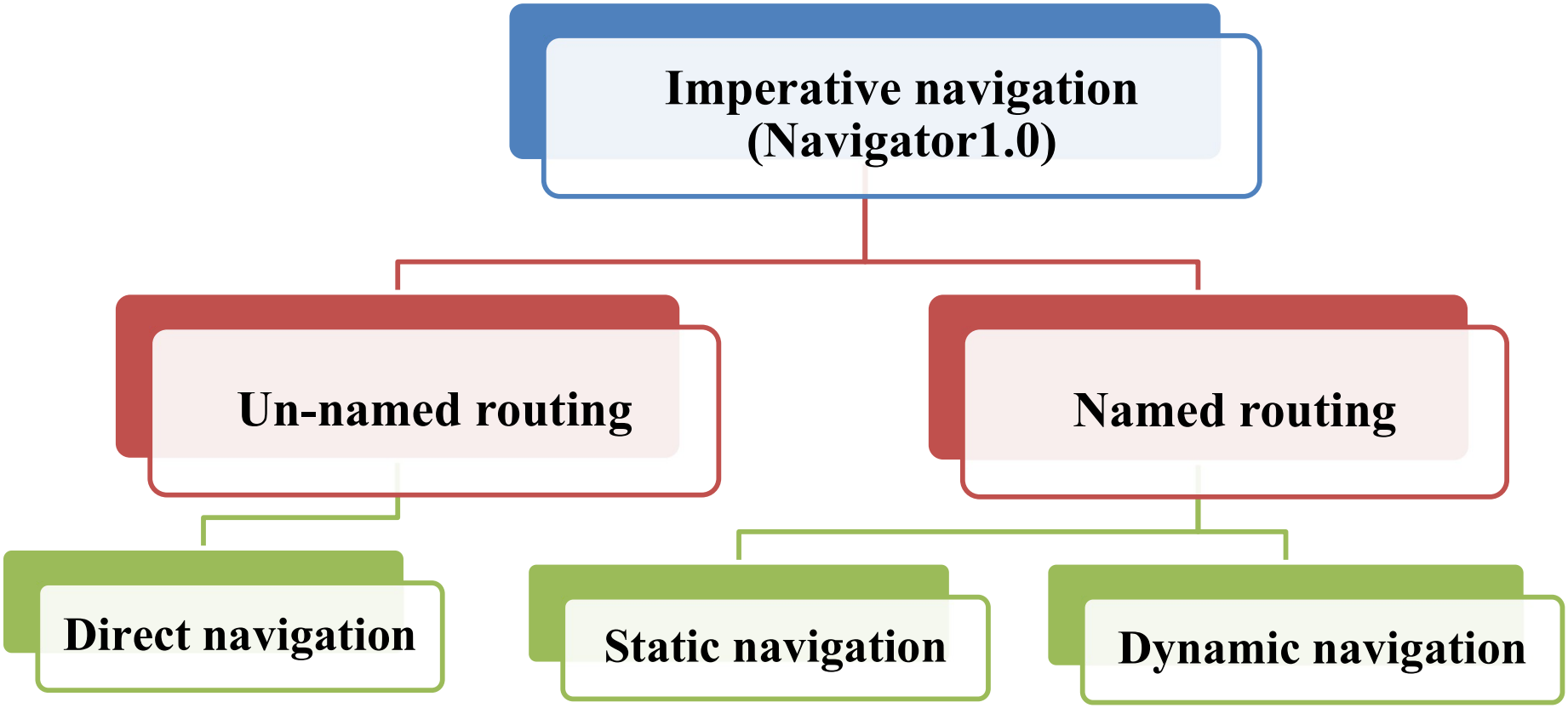
2. **Imperative navigation (Navigator 1.0).**

   - Un-named routing : direct navigation.
   - Named routing : static & dynamic navigation.

3. Declarative navigation (Navigator 2.0).

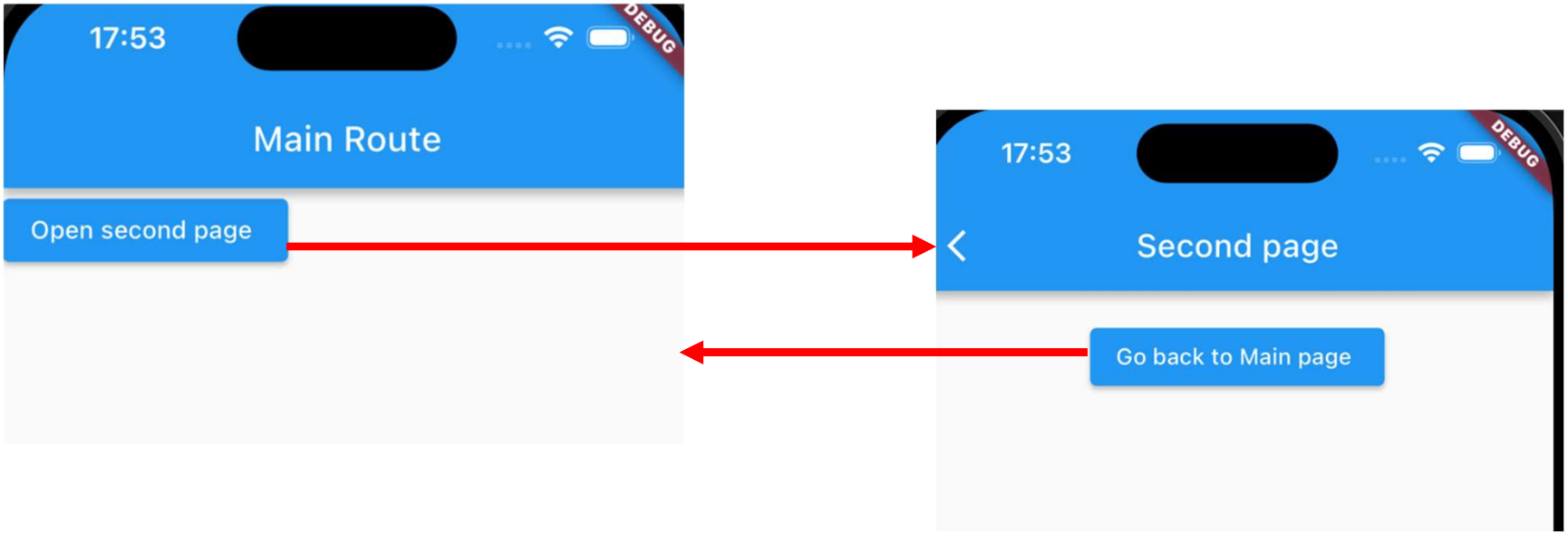
   - Why and when to use it ?
   - GoRouter package.

Imperative approach (Navigator 1.0)



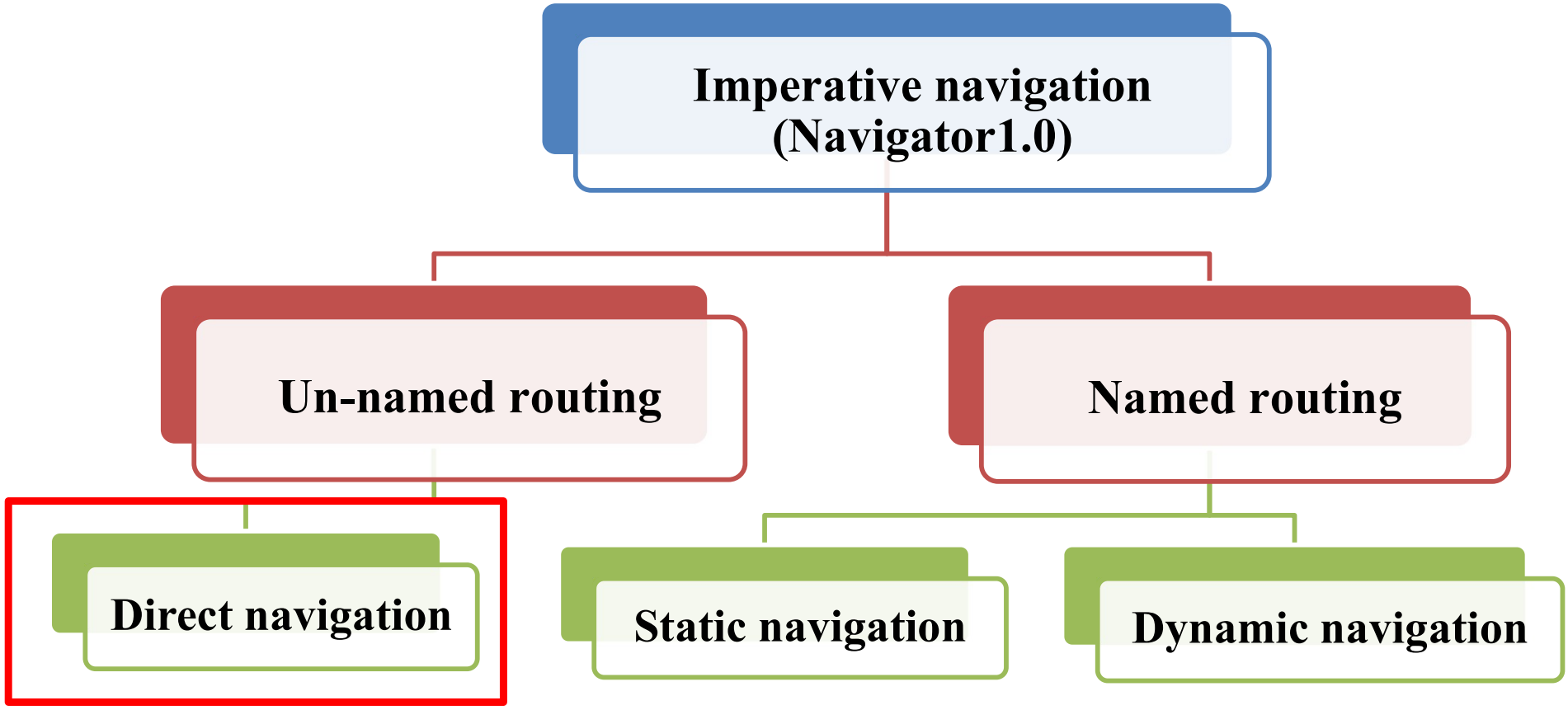
Imperative approach (Navigator 1.0)

- **Illustrative example:**



➢ Next, the example will be implemented using: **Direct, Static & Dynamic navigation.**

Imperative approach (Navigator 1.0)



Imperative approach (Navigator 1.0)

- **Implementation of the example with direct navigation :**
  - **push** method has two arguments:
    - ✓ A ***BuildContext***.
    - ✓ A ***PageBuilder*** (e.g. *MaterialPageRoute*).

```
// Pushing SecondRoute
Navigator.push( context,MaterialPageRoute(builder: (context) => SecondRoute()),);},
```

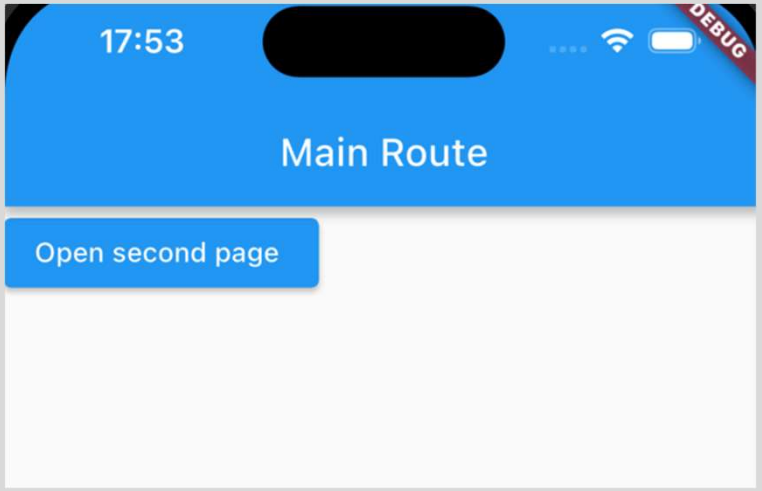
  - **pop** method has a ***BuildContext*** as argument and changes the current route to the most recently visited route.

```
Navigator.pop( context);
```

Imperative approach (Navigator 1.0)

- **Implementation of the example with direct navigation :**

Main route




```
import 'package:flutter/material.dart';
import 'SecondRoute.dart';

void main() { runApp ( MaterialApp (
            home: MainPage () ,)); }

class MainPage extends StatelessWidget {
const MainPage ({super.key});
  @override
  Widget build (BuildContext context) {
    return Scaffold(
              appBar: AppBar( title: const Text('Main Route'),), // AppBar
              body: Center( child:ElevatedButton(
                                  child: const Text('Open second route'),
                                  onPressed: () {
//  Pushing SecondRoute
Navigator.push( context,MaterialPageRoute(builder: (context) => SecondRoute()),);},
), // ElevatedButton), // Center); // Scaffold } }
```

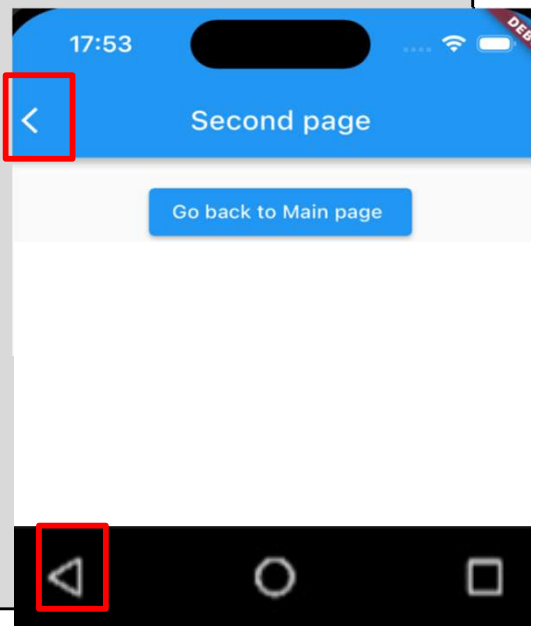
Imperative approach (Navigator 1.0)

- **Implementation of the example with direct navigation :**

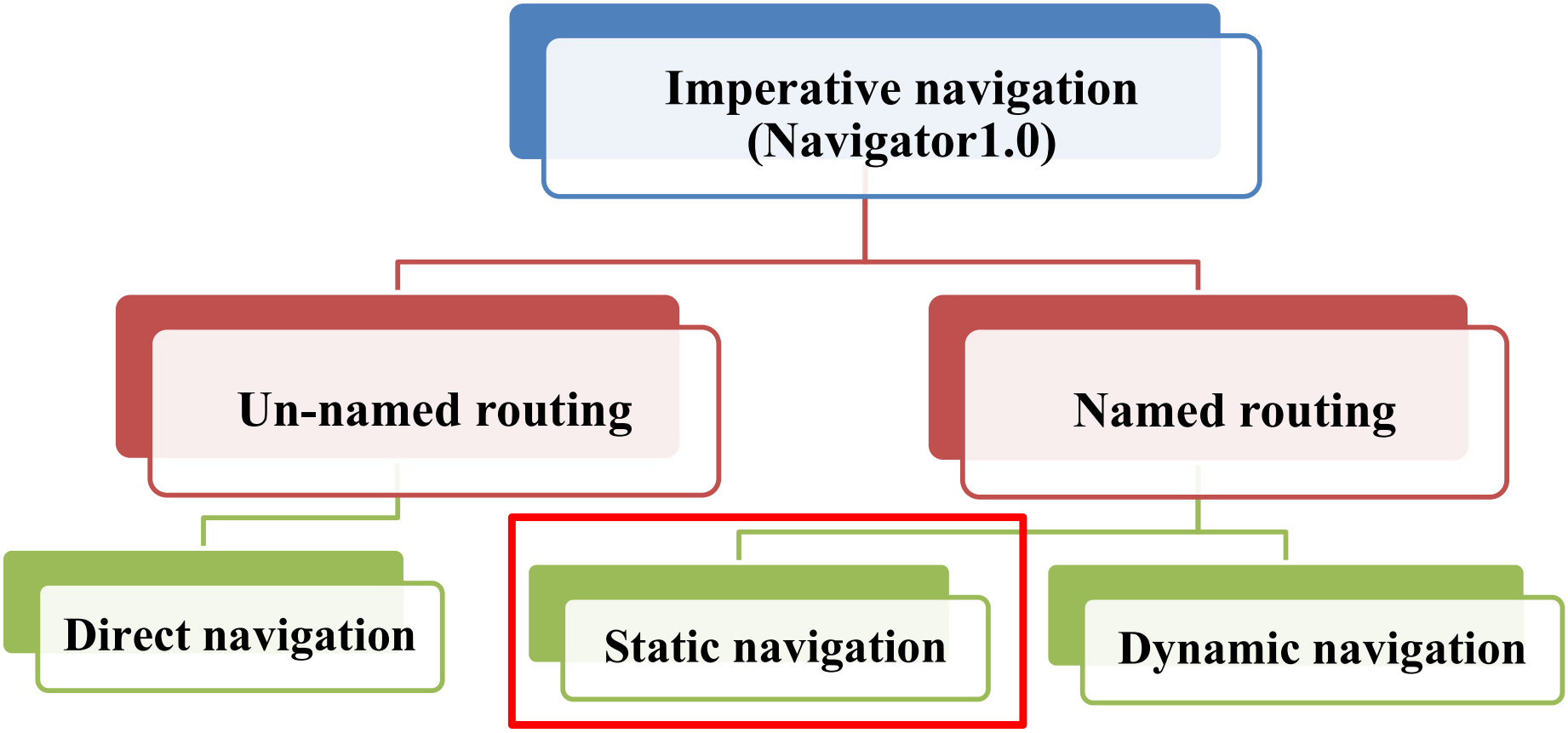
Second route

```
import 'package:flutter/material.dart';

class SecondRoute extends StatelessWidget {
  @override
  Widget build (BuildContext context) {
   return Scaffold(
            appBar: AppBar( title: const Text('Second Page'),), // AppBar
            body: Center( child:ElevatedButton(
                        child: const Text('Go back  to Main page'),
                        onPressed: () {
                        //  Removing (pop) SecondRoute
                                Navigator.pop( context);
                                    },
                                ), // ElevatedButton
                        ), // Center
            ); // Scaffold
} }
```



Imperative approach (Navigator 1.0)



Imperative approach (Navigator 1.0)

- **Implementation of the example with static navigation :**

1) Assign the route map (*Map<String, WidgetBuilder>*) to the "*routes*" property of MaterialApp class:

```
MaterialApp (
        home: MainPage () ,
        routes: { "second": (context) => SecondRoute()}, // Defining routes map
)
```

2) Use Navigator.pushNamed method :

```
Navigator.pushNamed( context, "second");},
```

Imperative approach (Navigator 1.0)

- **Implementation of the example with static navigation :**

```
import 'package:flutter/material.dart';
import 'SecondRoute.dart';

void main() { runApp ( MaterialApp (
            home: MainPage () ,
            routes: { "second": (context) => SecondRoute()}, // Defining routes map
)); }
class MainPage extends StatelessWidget {
const MainPage ({super.key});
  @override
  Widget build (BuildContext context) {
    return Scaffold(
            appBar: AppBar( title: const Text('Main Route'),), // AppBar
            body: Center( child:ElevatedButton(
                                child: const Text('Open second route'),
                                onPressed: () {//  Pushing SecondRoute
                                Navigator.pushNamed( context, "second");},
), // ElevatedButton), // Center); // Scaffold } }
```

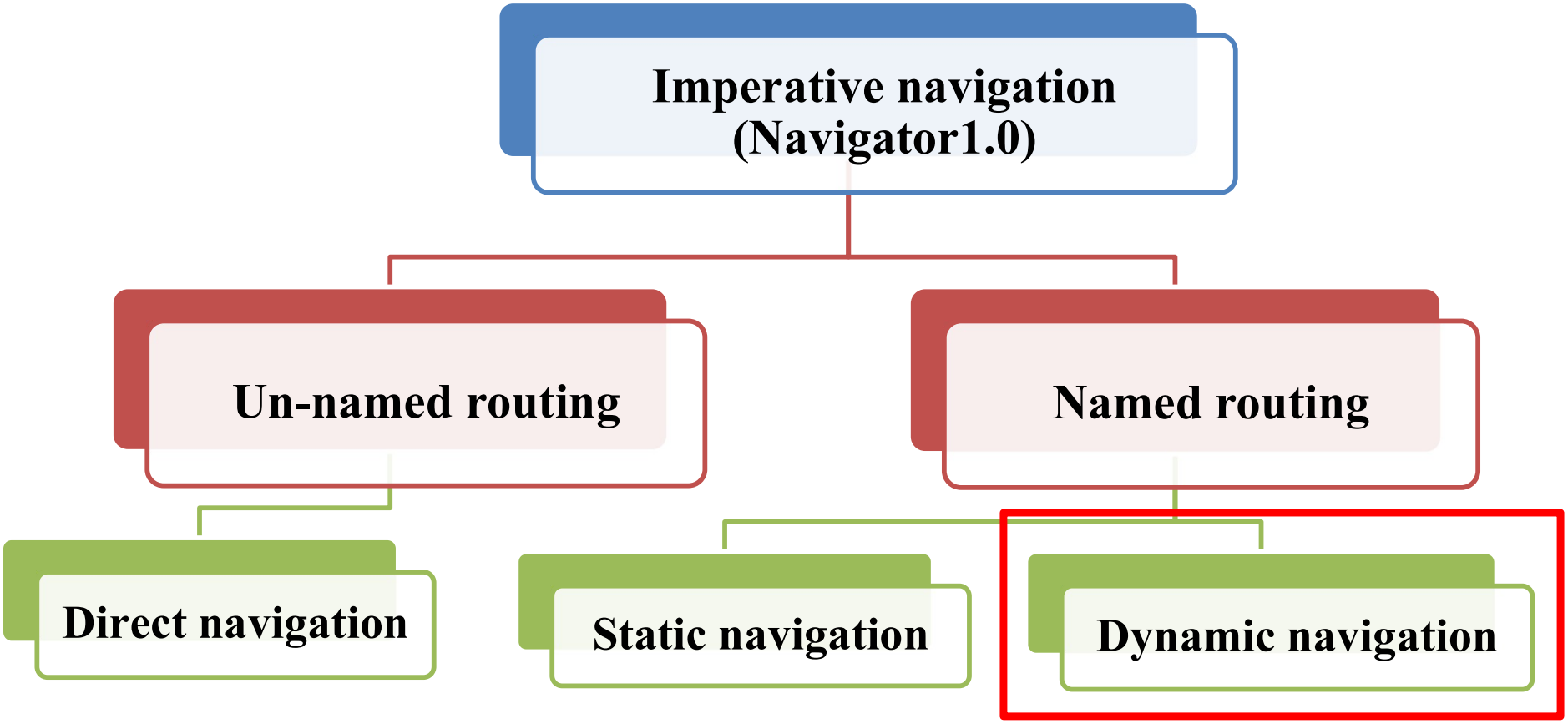
Imperative approach (Navigator 1.0)

- **Static navigation :**

  - Allows to change the path by using strings instead of providing component classes (*MaterialPageRoute)*.

  - They are usable from any point of the application.

Imperative approach (Navigator 1.0)



Imperative approach (Navigator 1.0)

- **Implementation of the example with dynamic navigation :**

  1) Use ***onGenerateRoute*** property to generate routes :

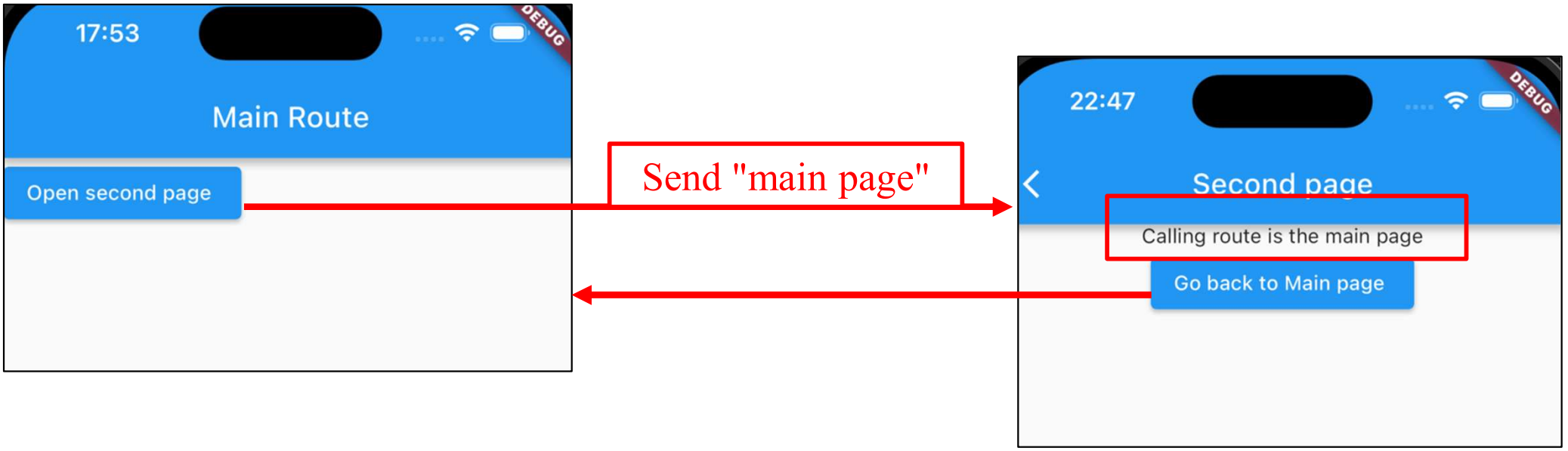
```
void main() => runApp( MaterialApp( home: MainPage(),
                                    onGenerateRoute: (RouteSettings setting) {
                                                       switch (setting.name) {
                                                                  case 'MainPage':
                          return MaterialPageRoute( builder: (context) => const MainPage());
                                                                  case 'second':
          //  Set settings parameter in case of passing data (using Argument approach)
              to the second page
                          return MaterialPageRoute(builder: (context) => SecondRoute(),
                                                                 settings: setting);}
                          return null;}),);
```

  2) ***Navigator.pushNamed*** method is the same as in static navigation.

Imperative approach (Navigator 1.0)

- **Passing data between screens: illustrative example.**

  sending data from one screen to another consists of passing an object along to the target route.



Imperative approach (Navigator 1.0)

- **Passing data between screens: direct navigation.**

1) Modify *Navigator.push* method :

```
//  Pushing SecondRoute and sending a value of the name argument
        Navigator.push( context,MaterialPageRoute(builder: (context) =>
            SecondRoute(callingPageName: " main page")),);},
```

Imperative approach (Navigator 1.0)

- **Passing data between screens: direct navigation.**

2) Modify target page class :

```
import 'package:flutter/material.dart';

class SecondRoute extends StatelessWidget {
final String callingPageName ; // Add an attribut to the SecondRoute class
SecondRoute ({required this.callingPageName }); // Add a constructor
  @override
  Widget build (BuildContext context) {
    return Scaffold(
              appBar: AppBar( title: const Text('Second Route'),), // AppBar
              body: Column( children: [
              // Display the received data in a Text widget
 Center(child: Text ("Calling route is the ${ this.callingPageName" },), // Text),//Center
              ElevatedButton( child : Text("Return to calling page "),
                                  onPressed: () {Navigator.pop(context);}
) // ElevatedButton], // children), // Column); // Scaffold  } }
```

Imperative approach (Navigator 1.0)

- **Passing data between screens: static and dynamic navigation.**
  - **Passing data** by using ***arguments*** parameter of the ***pushNamed*** method:

1) Modify *Navigator.pushNamed* method :

```
// Pushing SecondRoute using static navigation and passing data to it using
   arguments parameter of pushNamed method
Navigator.pushNamed(context, "second", arguments: " main page");
```

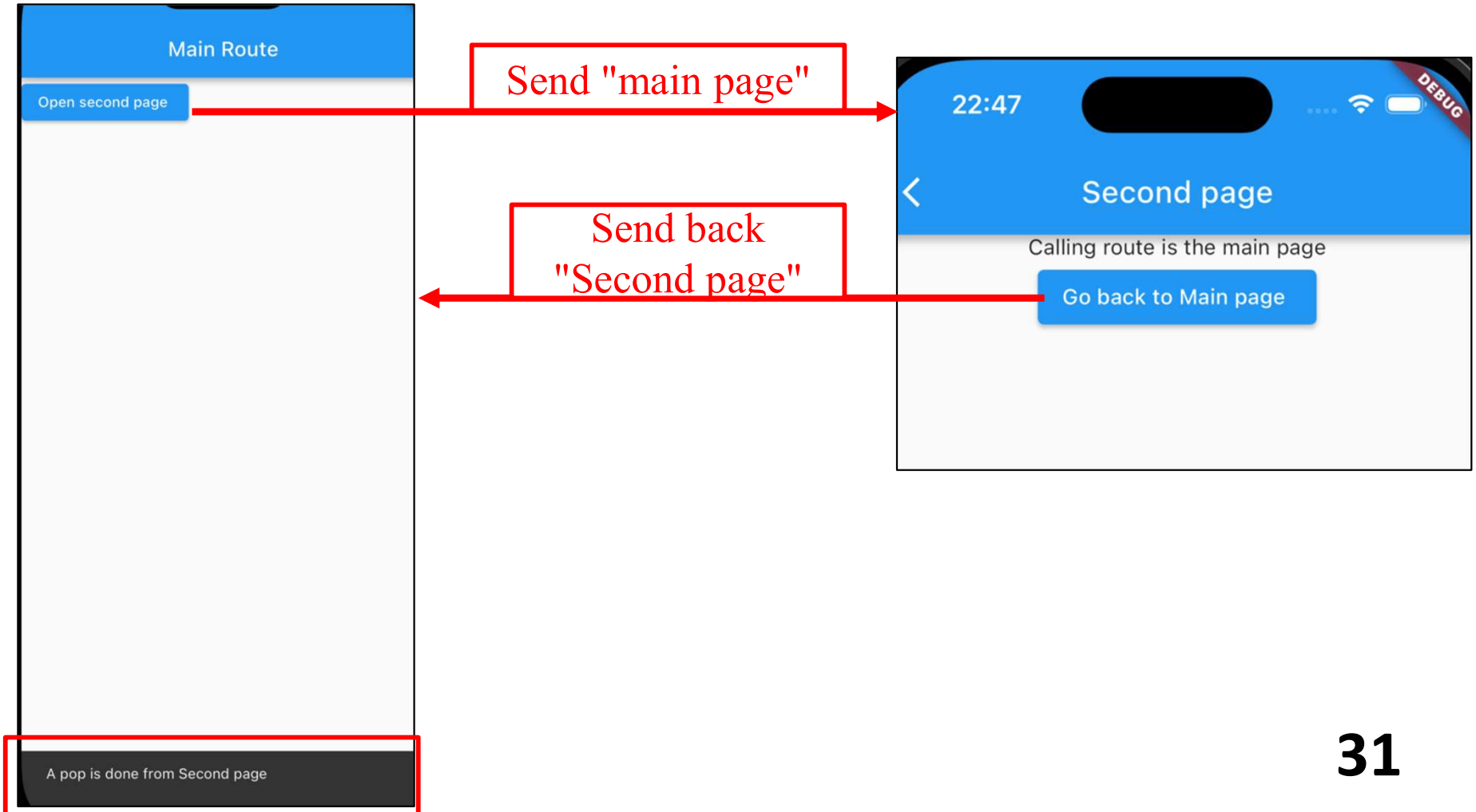
2) Get received data using ***ModalRoute*** class :

```
// ModalRoute is a class covers the entire Navigator
   final receivedArg = ModalRoute.of(context)?.settings.arguments ;
```

Imperative approach (Navigator 1.0)

- **Sending data back: illustrative example.**

  Returning information from one screen to the previous one.



Imperative approach (Navigator 1.0)

- **Sending data back: direct, static and dynamic navigation.**

1) Modify *Navigator.pop* method:

```
// pop to return to the main page while sending "Second page" back to it.
Navigator.pop( context, "Second page");},
```

2) Modify *onPressed* argument of the pushing button :

Main route

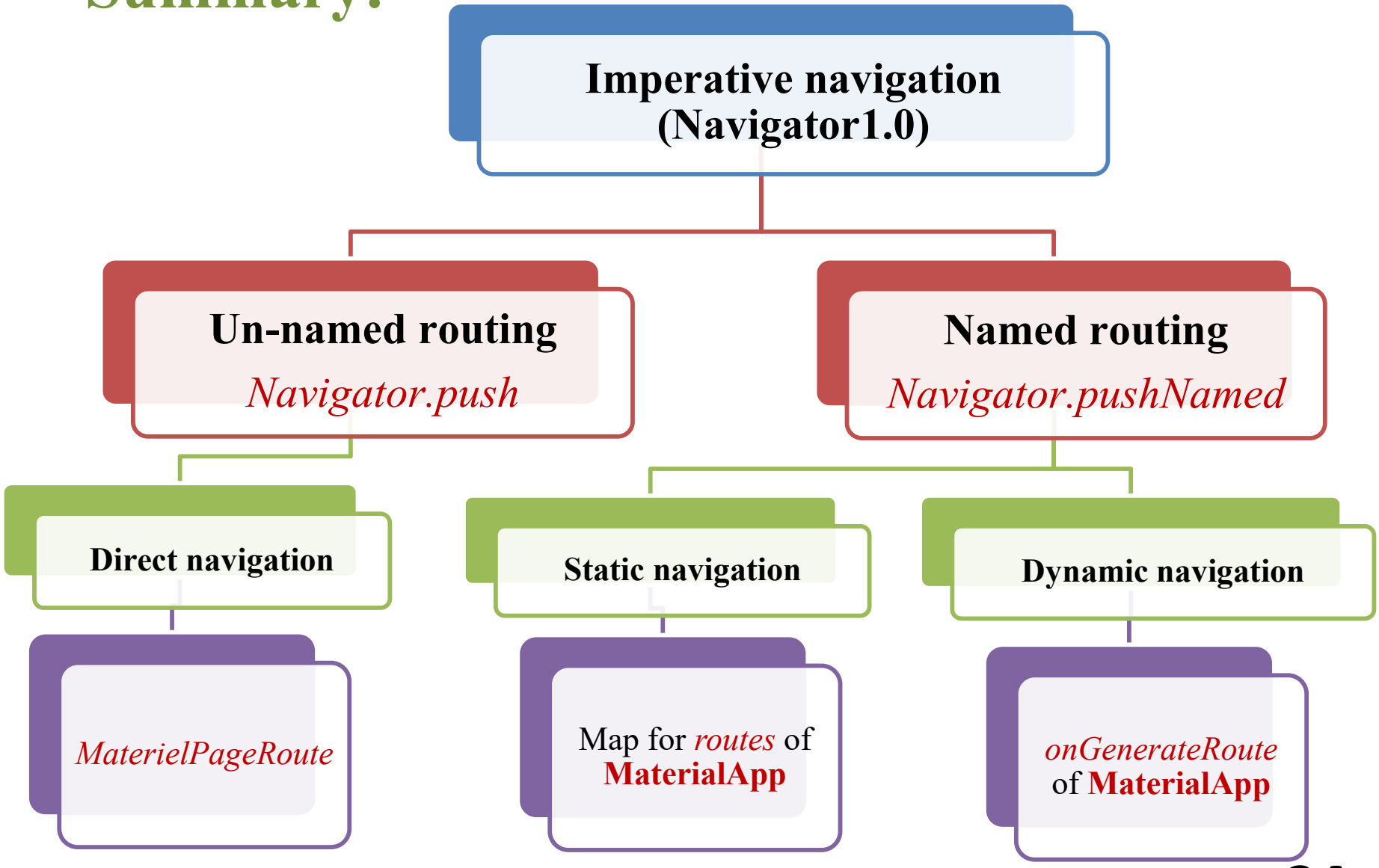
```
// Pushing SecondRoute and sending a value of the callingPageName argument
ElevatedButton( onPressed: () async {final receivedData = await Navigator.push(
context,MaterialPageRoute(builder(builder: (context) =>
SecondRoute(callingPageName : "main page"),),);
// Use a SnackBar widget to desplay received data from the Second page
ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text("A pop is done
from $receivedData")));},
child: const Text("Open second page ")),
```

Imperative approach (Navigator 1.0)

- **Summary:**
  - ***Un-named routing (Direct navigation):*** is implemented with *MaterielPageRoute*.
  - ***Named routing,*** implemented in two ways where the route name is pushed using *Navigator.pushNamed(...)* method:
    - ✓ ***Static navigation:*** is implemented by assigning a map of routes to the property *routes* of **MaterialApp** class.
    - ✓ ***Dynamic navigation:*** is implemented by generating routes using the callback function *onGenerateRoute* of **MaterialApp** class.

Imperative approach (Navigator 1.0)

- **Summary:**



Imperative approach (Navigator 1.0)

- **Summary:**

  - **Direct navigation:** ok for small project but adds code duplication for complex apps.
  - **Static navigation :** a good option if no logic around the routes (eg. checking authentication before showing a page).
  - **Dynamic navigation:** benefits of using named routes and adding logic.

Plan

1. **Introduction to route & Navigator class.**

2. **Imperative navigation (Navigator 1.0).**

   - Un-named routing : direct navigation.
   - Named routing : static & dynamic navigation.

3. **Declarative navigation (Navigator 2.0).**

   - Why and when to use it ?
   - GoRouter package.

Plan

1. **Introduction to route & Navigator class.**

2. **Imperative navigation (Navigator 1.0).**

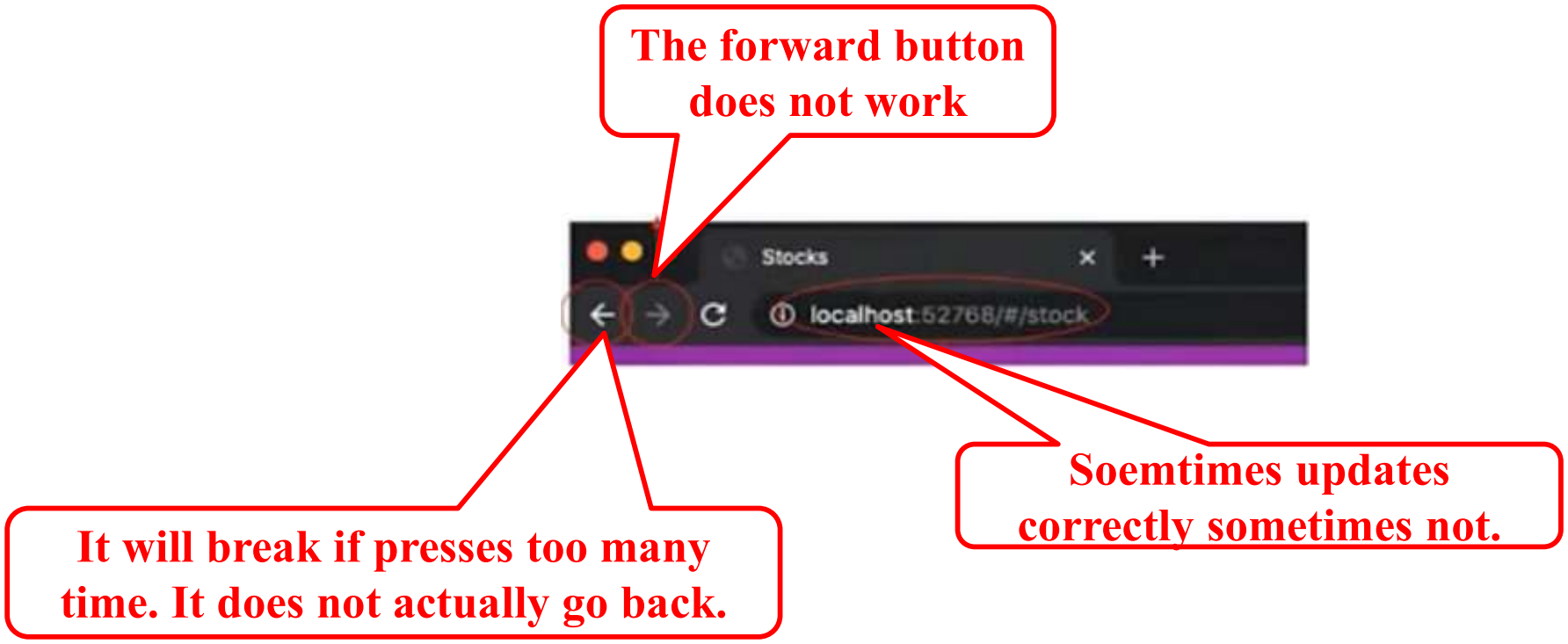
   - Un-named routing : direct navigation.
   - Named routing : static & dynamic navigation.

3. **Declarative navigation (Navigator 2.0).**

   - Why and when to use it ?
   - GoRouter package.

Declarative approach (Navigator 2.0)

- **Why and when to use it ? Gaps in imperative approach:**
  - ***Hard to reason about and maintain:*** the navigation logic is spread across different parts of the application code.
  - ***Web URL synchronization:***



Declarative approach (Navigator 2.0)

- **Why and when to use it ? Gaps in imperative approach:**

  - ***Challenging for complex scenarios:***

    - ✓ ***Conditional navigation:***

      If the user is logged in → show home screen.

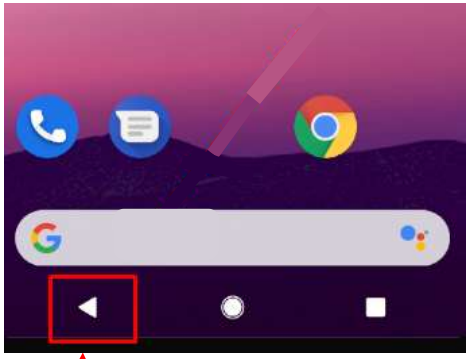
      If not → show login screen.

    - ✓ ***Nested navigation:*** multiple navigation stacks.

      Tab 1 (Home): Home → Details.

      Tab 2 (Search): Search → Results → ItemDetails.

      Tab 3 (Profile): Profile → EditProfile.

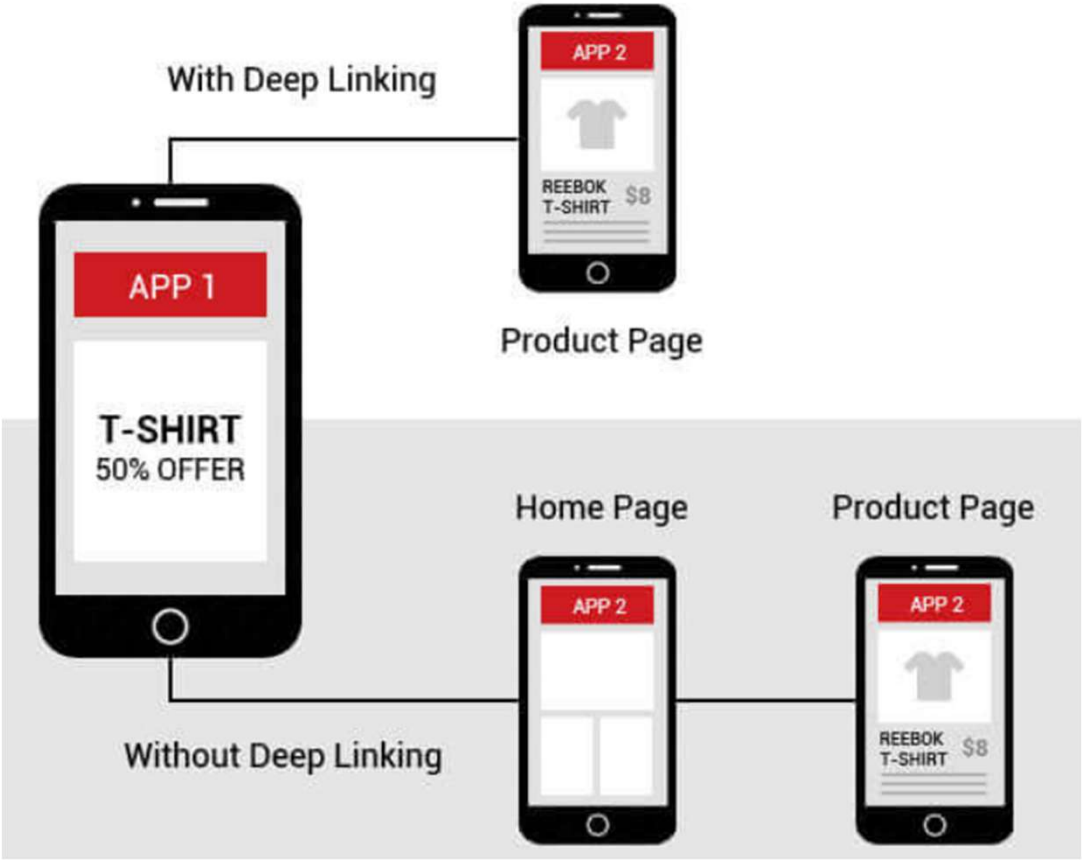


No good and easy way to navigate through multiple routes when user presses the **Android hardware Back** button

  - ***Limited support for Deep Linking:***

Declarative approach (Navigator 2.0)

- **Deep links :** are mobile links that operate much like hyperlinks, but instead of directing users to a web page, they send them to a specific screen within a mobile application.



Declarative approach (Navigator 2.0)

- **Deep linking :** allows the mobile app to open a **specific screen** directly from:

  - a web link (https://example.com/product/23).
  - a custom URL scheme (myapp://profile/45).
  - an email, SMS, or WhatsApp message.
  - a push notification.
  - a QR code.
  - ….

  Instead of launching the home screen, the app jumps **straight to the right page with the right data**.

Declarative approach (Navigator 2.0)

- **Why deep linking is important in mobile app ?**

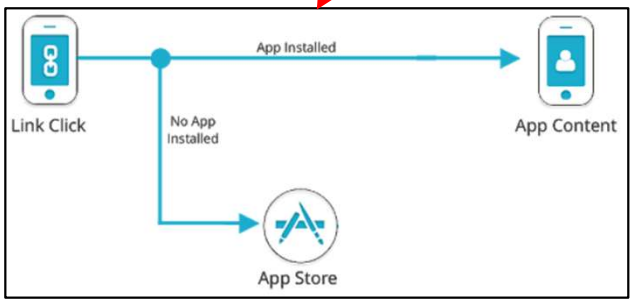
  It allows to:

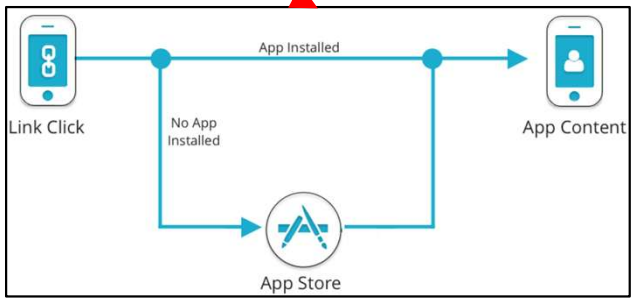
  - Navigate directly to a screen with parameters → e.g., /product/15.
  - Improve user experience (notifications, external links).
  - Support marketing campaigns.
  - Support content sharing.

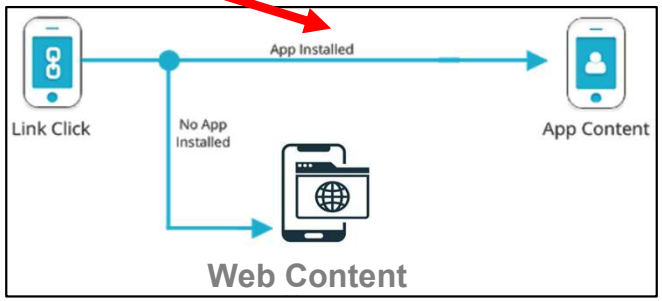
Declarative approach (Navigator 2.0)

- **Types of deep linking :**

Classic (Standard)	Deferred	Fallback (redirect)	Contextual
Directs users to a specific part of the app using the app URL. Work **only** if the app is installed, otherwise, it leads to an error page.	Takes to the relevant content inside the app, if the app is installed. If not, it takes to the right app store and asks to download the app to view the content.	Redirects to an alternative location (e.g. mobile website) instead of to the app store when the app is not installed.	Like deferred deep linking, but with **data (context) transferred** into the application. E.g: https://myapp.com/welcome?**ref=ABC123**

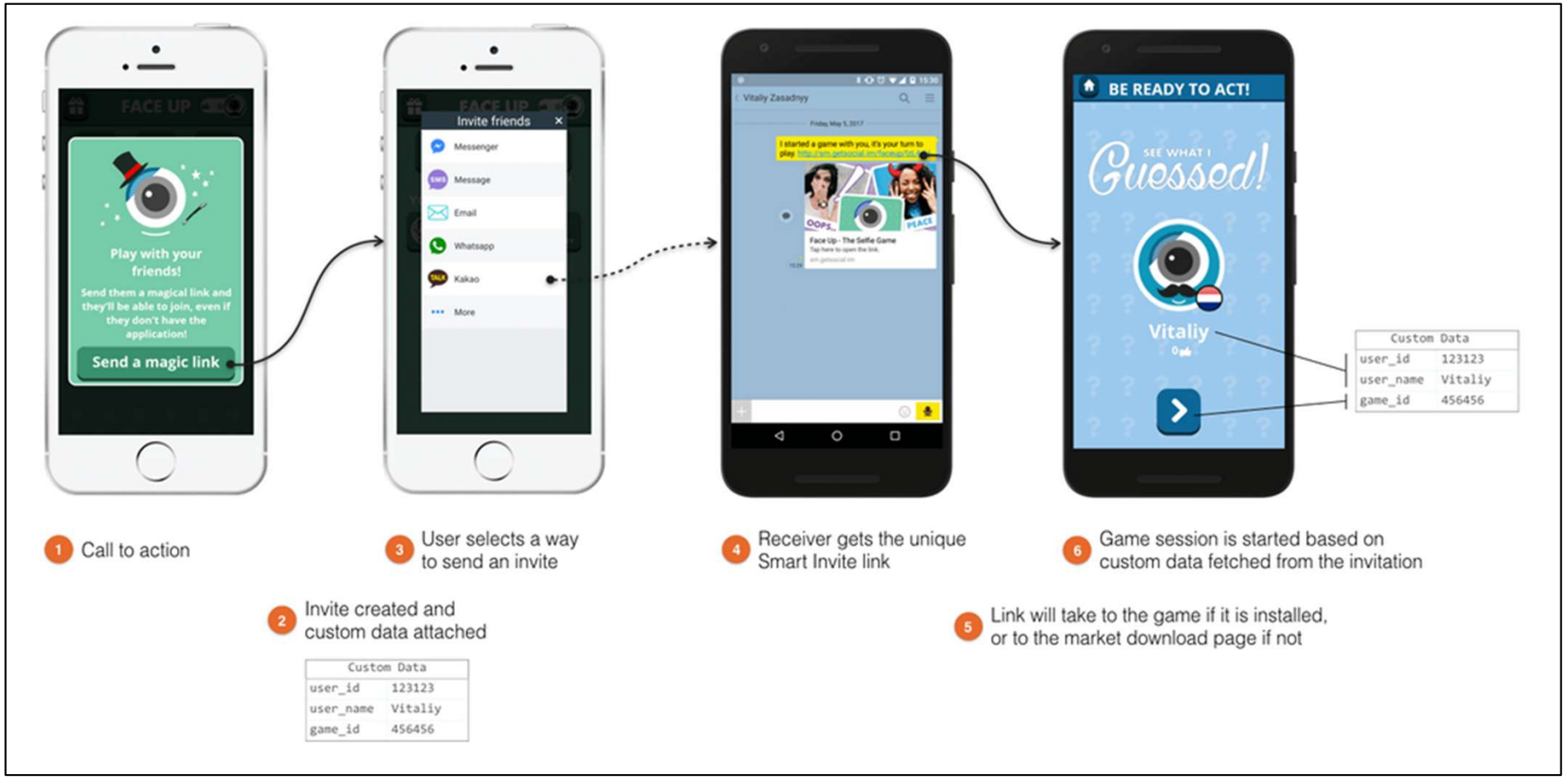






Declarative approach (Navigator 2.0)

- **Types of deep linking : contextual.**



Plan

1. **Introduction to route & Navigator class.**

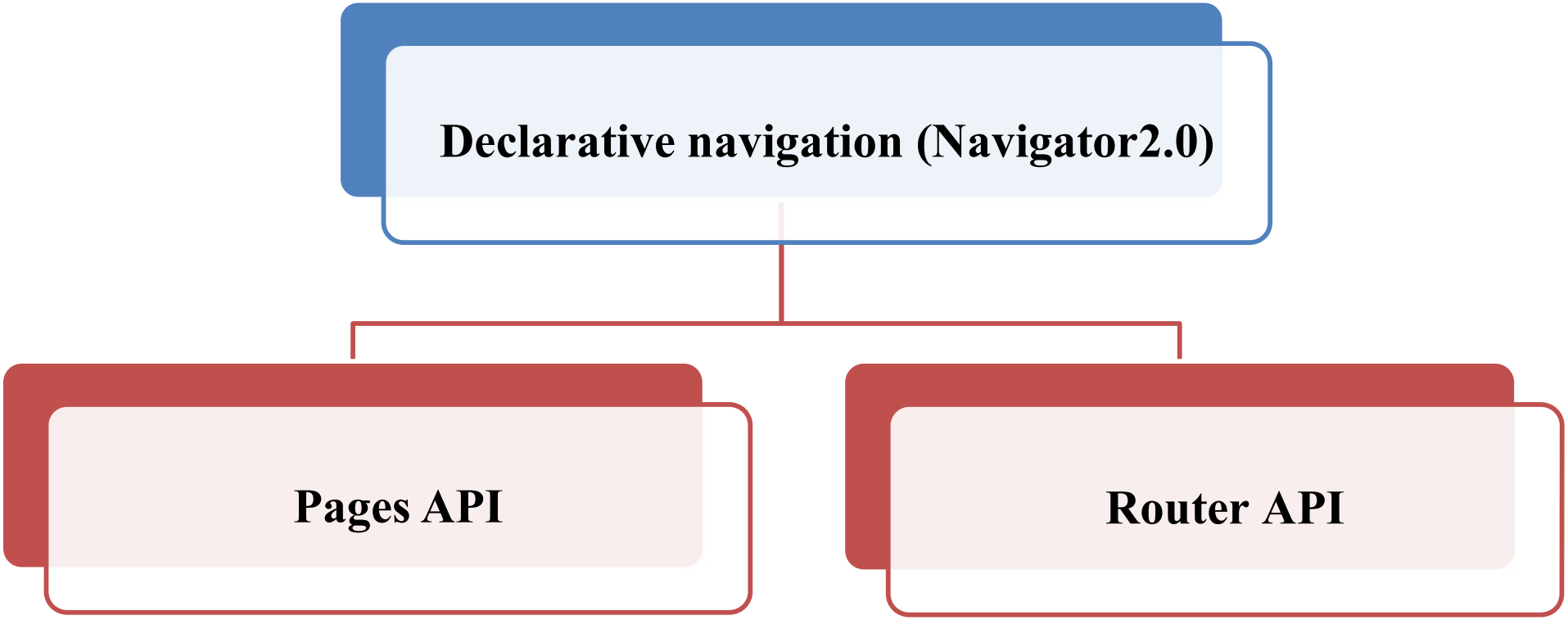
2. **Imperative navigation (Navigator 1.0).**

   - Un-named routing : direct navigation.
   - Named routing : static & dynamic navigation.

3. **Declarative navigation (Navigator 2.0).**

   - Why and when to use it ?
   - GoRouter package.

Declarative approach (Navigator 2.0)



Declarative approach (Navigator 2.0)

Declarative Router API so complexe ?

go_router package

Declarative approach (Navigator 2.0)

- **Go Router:**

GoRouter is currently the best and recommended navigation system for deep linking in Flutter apps.

- **go_router :** a declarative routing package reducing the complexity of the **Router API**.

•On les utilise quand l'information fait partie de l'URL, comme un identifiant ou un nom d'u

Declarative approach (Navigator 2.0)

- **go_router features :**
  - Parsing path and query parameters.
  - Displaying multiple screens for a destination (sub-routes).
  - Redirection support: redirect the user to a different URL based on app state.
  - Support nested tab navigation.

Declarative approach (Navigator 2.0)

- **go_router :** installation instruction

  - Run the command : `$ flutter pub add go_router`

    This will add the following line to the pubspec.yaml.

```
dependencies:
  go_router: ^12.1.1
```

  - Import the package from the dart code:

```
import 'package:go_router/go_router.dart';
```

Declarative approach (Navigator 2.0)

- **go_router :** configuration step (GoRouter definition).

Two main classes are used: **GoRouter** & **GoRoute.**

```
import 'package:go_router/go_router.dart';
// Defining a GoRouter object
final MyRouter = GoRouter (
initialLocation: '/',
routes:[ // Defining routes list
        GoRoute (// Defining a GoRoute object for each route
                name: "home", // Optional, used to navigate instead of the path parameter
                path : '/firstpage',
                builder: (context, State) => Page1() ), // GoRoute
        GoRoute (
                name: 'second page',
                path: '/second',
                builder: (context, State) => Page2() ), // GoRoute

.   .   .   .
] //routes ); // GoRouter
```

Declarative approach (Navigator 2.0)

- **go_router :** configuration step (setting the app router using **MaterialApp.router**).

```
@override
Widget build(BuildContext context) {
        return MaterialApp.router(
                        routerConfig:  MyRouter,
                        ); // MaterialApp.router
    } // build
```

Declarative approach (Navigator 2.0)

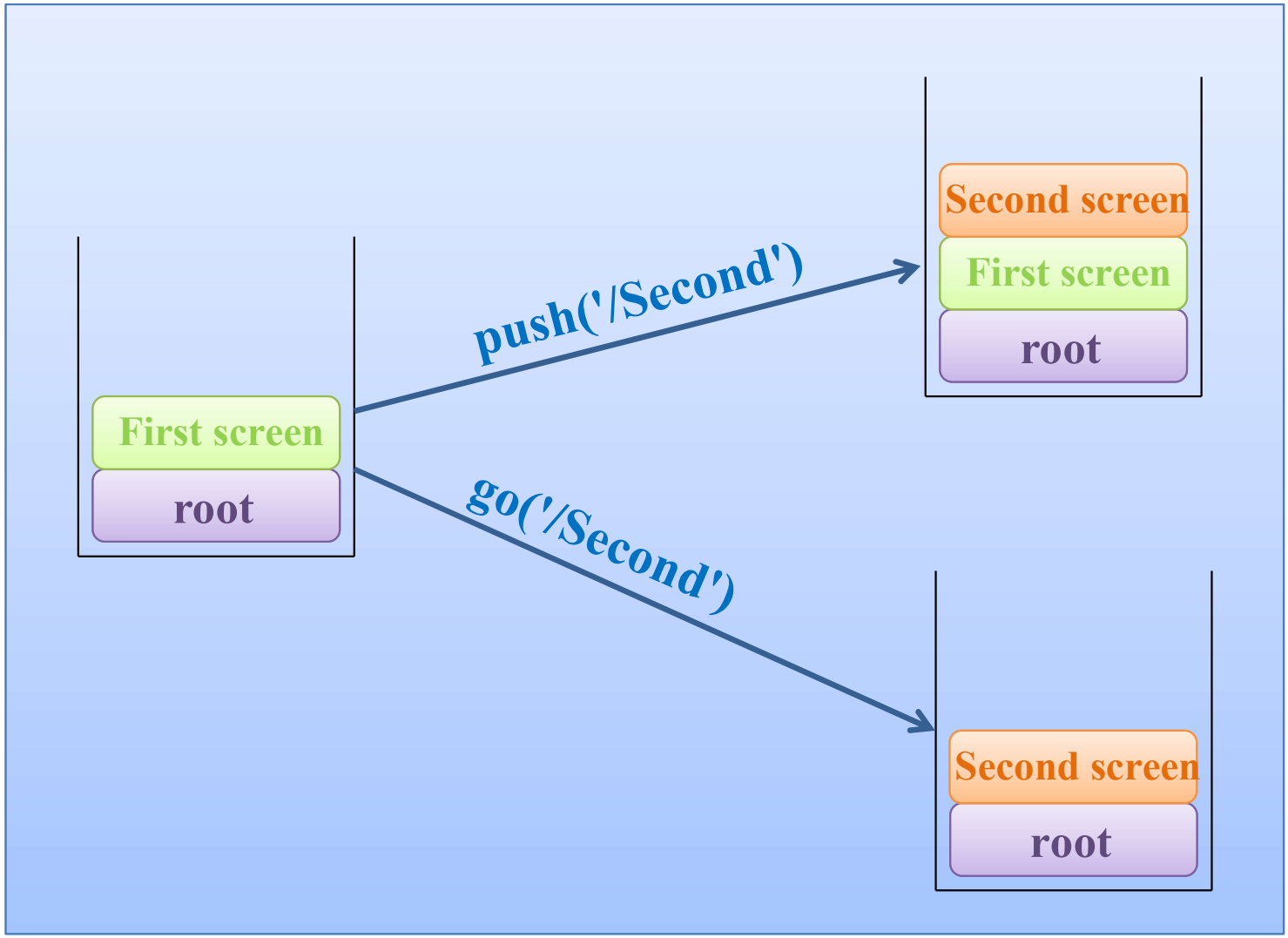
- **go_router :** navigation between screens

```
build(BuildContext context) {
return TextButton(
                        onPressed: () => context.go('/second'),
// OR
                        onPressed: () => context.goNamed('second page'),
// OR
                         onPressed: () => context.push('/second'),
// OR
                        onPressed: () => context.pushNamed('second page'),
                    );
}
```

```
build(BuildContext context) {
return TextButton(
                            onPressed: () => context.pop( ),
                        );
}
```

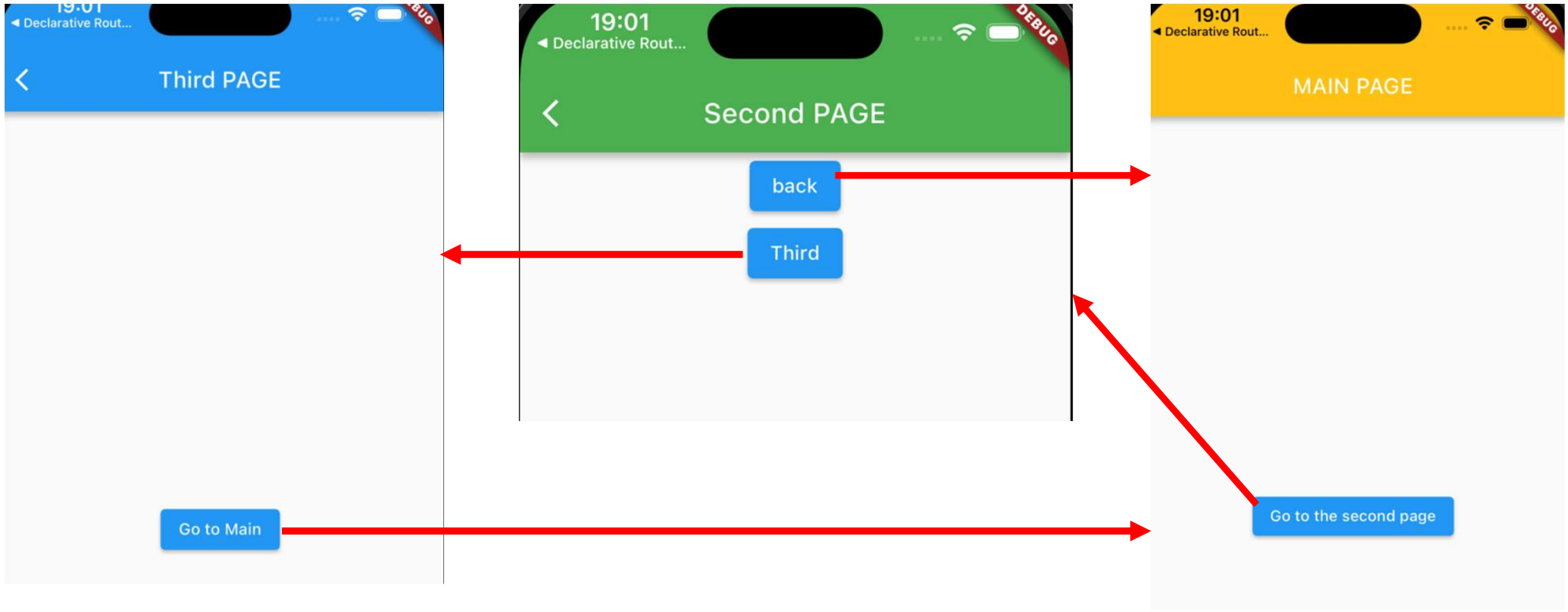
Declarative approach (Navigator 2.0)

- go_router : **Go vs Push**



Declarative approach (Navigator 2.0)

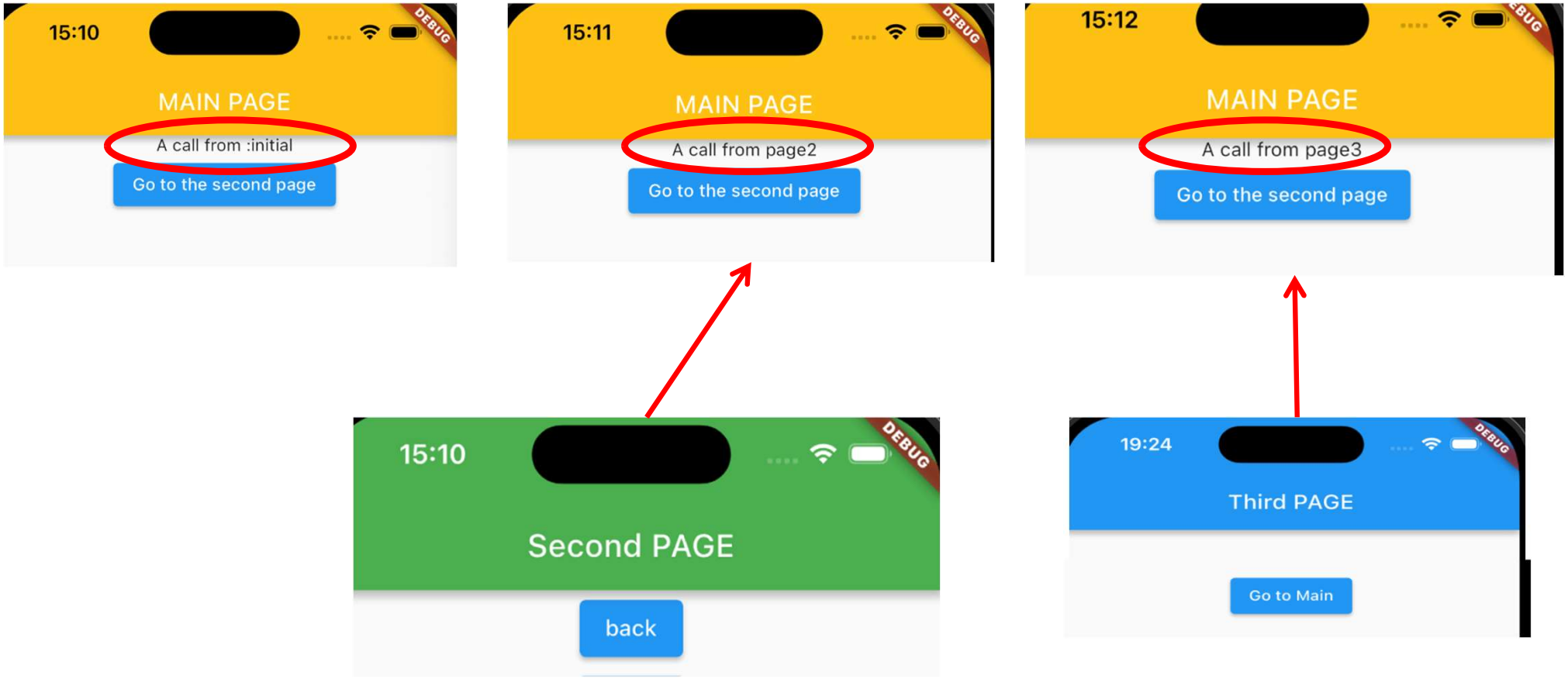
- **go_router :** passing data to Route.



Illustrative example

Declarative approach (Navigator 2.0)

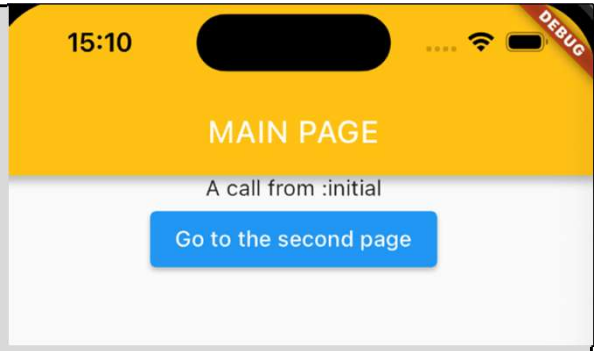
- **go_router :** passing data to Route.



Illustrative example

Declarative approach (Navigator 2.0)

- **go_router :** passing data to Route.

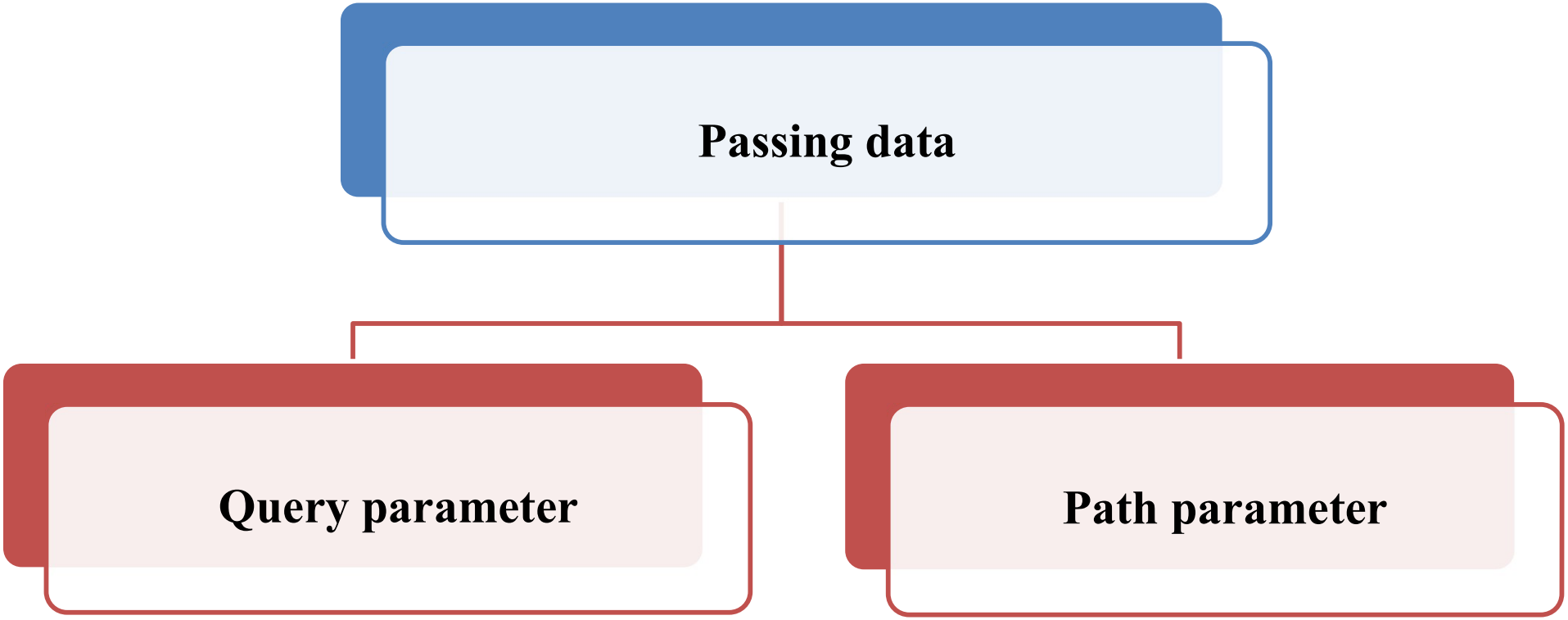



```
import 'package:flutter/material.dart';
import 'package:go_router/go_router.dart';
class Page1 extends StatelessWidget {
final String  nameParam;

Page1({super.key, required this. nameParam});
  @override
Widget build (BuildContext context){
return Scaffold( appBar: AppBar(title:  Text("MAIN PAGE"),backgroundColor:
                                                          Colors.amber,),
body: Center(  child: Column ( children: [
                              Text("A call from $ nameParam "),
                               ElevatedButton(
                                        child: Text("Go to the second page"),
                                        onPressed: (){
                                                  context.go('/secondpage');
} ), // ElevatedButton ] // Children ), //Column ), //Center); //Scaffold } //build} //Page1
```

Declarative approach (Navigator 2.0)

- **go_router :** passing data to Route.



Declarative approach (Navigator 2.0)

- **go_router** : passing data to Route using **Path parameters.**

```
import 'package:go_router/go_router.dart';

class Page3 extends StatelessWidget {
Page3({super.key, });
  @override
Widget build (BuildContext context){
return Scaffold( appBar: AppBar(title:  Text("Third PAGE"),backgroundColor:
                                                                    Colors.blue,),
                    body: Center(  child: ElevatedButton(
                                                child: Text("Go to Main"),
                                                onPressed: ( ){
                                  // Go to the first page using its path combined with the
                                  value "page3" passed to its parameter nameParam
                                                String name = "page3";
                                                context.go('/firstpage/$name');
                                               }
                          ), // ElevatedButton ), //Center); //Scaffold } //build} //Page3
```

Declarative approach (Navigator 2.0)

- **go_router** : passing data to Route using **Path parameters.**
  - Prefix the path segment with ":" followed by a unique name.
  - Use the **GoRouterState State** to access to the value of the path parameter.

```
final _MyRouter  = GoRouter (initialLocation: '/firstpage/initial',
routes: [ GoRoute (
            name: 'firstpage',
            path: '/firstpage/:name', // Specify the parameter called "name" for
                                      the path of the first page.
            builder: (context, state)
            //  Get the value of the path parameter "name" and assign it to the
nameParam  argument of Page1.
                        => Page1(nameParam: state.pathParameters['name']! ,)
),
```

Declarative approach (Navigator 2.0)

- **go_router** : passing data to Route using **Query parameters** (?name=page3).

```
class Page3 extends StatelessWidget {
Page3({super.key, });
  @override
Widget build (BuildContext context){
return Scaffold( appBar: AppBar(title:  Text("Third PAGE"),backgroundColor:
                                                                    Colors.blue,),
                body: Center(  child: ElevatedButton(
                                                child: Text("Go to Main"),
                                                onPressed: ( ){
                                // Go to the first page using its path combined with the
                                value "page3" passed to its parameter nameParam
                                                String name = "page3";
                                                context.go('/firstpage?name=$name');
                                                }
                        ), // ElevatedButton ), //Center); //Scaffold } //build} //Page3
```

Declarative approach (Navigator 2.0)

- **go_router :** passing parameter to Route using **Query parameters** (?name=page3).

```
final _MyRouter  = GoRouter (initialLocation: '/firstpage?name=initial',
routes: [
GoRoute (
            name: 'firstpage',
            path: '/firstpage', // Specify the path of the first page without the
                                parameter "name".
            builder: (context, state)
            //  Get the value of the query parameter "name" and assign it to the
nameParam  argument of Page1.
                        => Page1(nameParam: state.uri.queryParameters['name']! ,)
),
```

Declarative approach (Navigator 2.0)

- go_router : **Path** vs **Query** **parameters**.

Path parameters	Query parameters
•Part of the URL path: **/user/:userId** •Typically used to **identify a specific resource.** •**Example**: **/user/42 → userId = 42**	•Key/value pairs after ? in the URL: **?key1=value1&key2=value2** •Used for **optional options, filters, search, sorting, …** , not for identifying a resource •**Example**: **/search?q=flutter&page=2**

Declarative approach (Navigator 2.0)

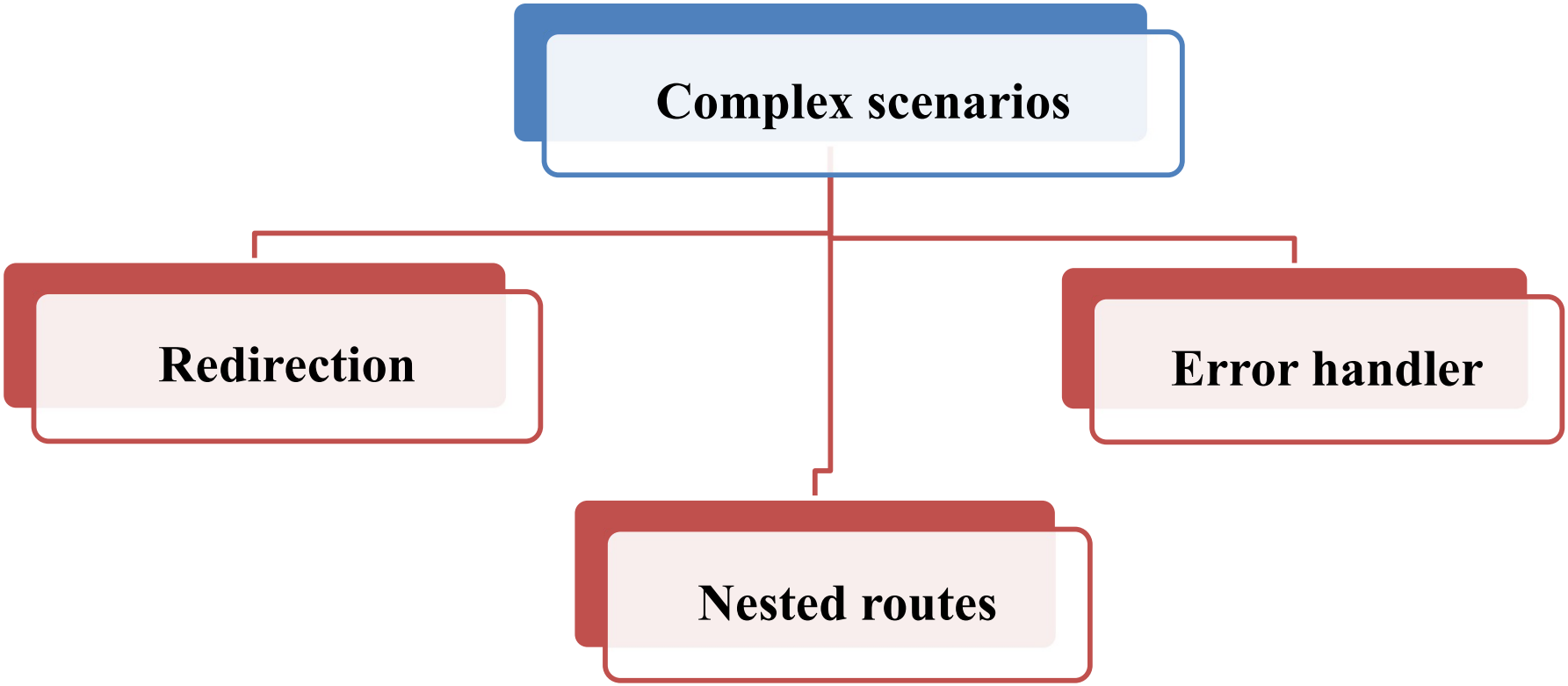
- go_router : **Path** vs **Query** parameters example.

To implement the URL: /user/42/orders?sort=desc&page=2

```
final _MyRouter  = GoRouter (
routes: [
        GoRoute(
                path : '/user/:userId/orders',
                builder : (context, state) {
                        final userId = state.params['userId'];        // "42"
                        final sort = state.queryParams['sort'];       // "desc"
                        final page = state.queryParams['page'];       // "2"
                    return OrdersPage(userId: userId, sort: sort, page: page);
            },
        )
    ]
);
```

Declarative approach (Navigator 2.0)

- **go_router :** handling complex navigation scenarios.

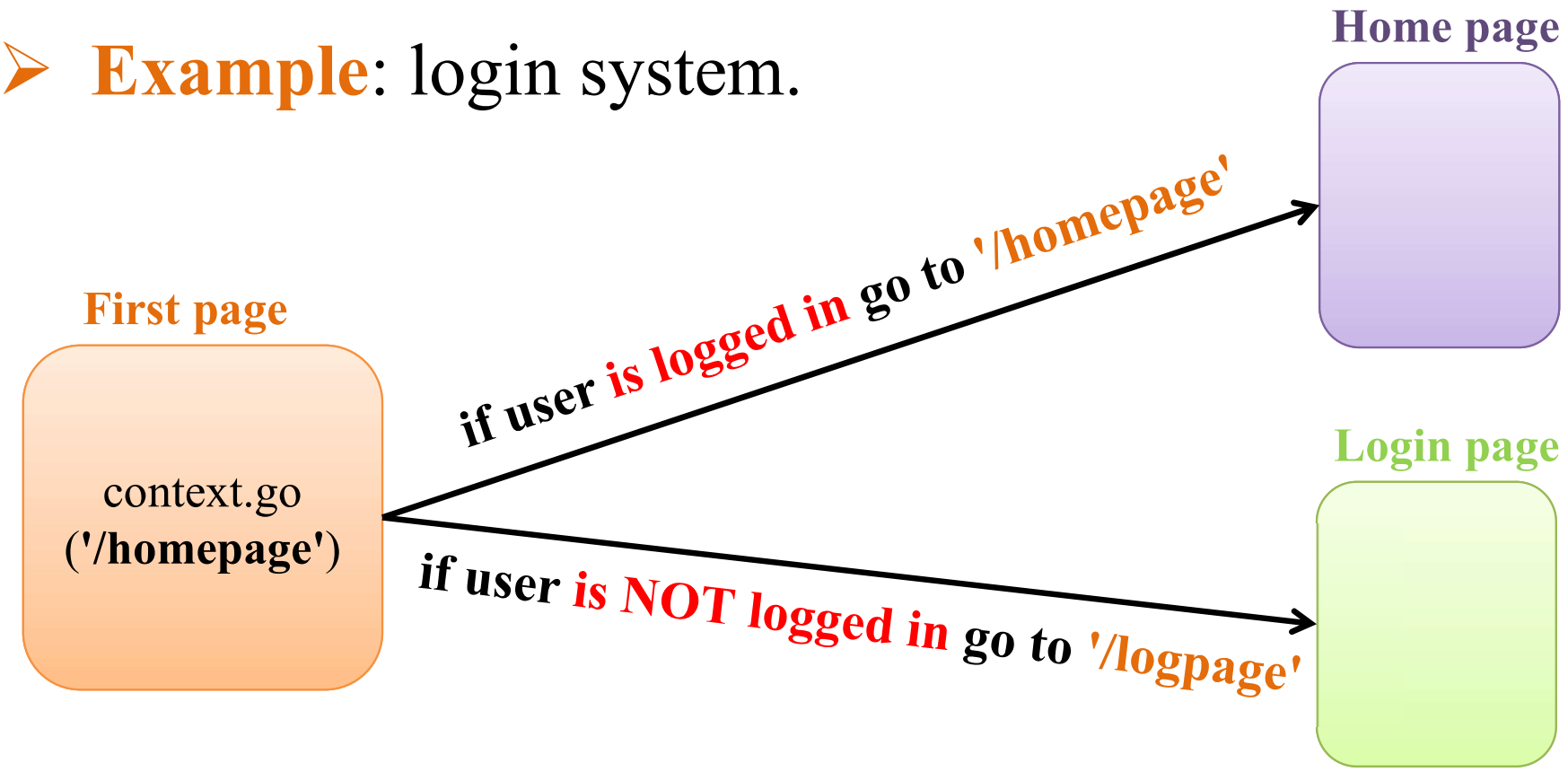


Declarative approach (Navigator 2.0)

- **go_router :** redirecting routes.

➢ It is used to change the location to a new one based on the app **state**.

➢ **Example**: login system.



Declarative approach (Navigator 2.0)

- **go_router :** redirecting routes.

```
bool  isLogged = false;
final  MyRouter = GoRouter (
        redirect: (context, state) { //  redirect homepage route depending on the
                                     // variable isLogged (app state).
                if( state.matchedLocation = = '/homepage'){
                        if (isLogged ) {
                                return "/ homepage";
                        } else {
                                return "/logpage";
                        }
                        return null;
                }
        },
        initialLocation: "/firstpage",
        routes: [
                .
                .
```

Declarative approach (Navigator 2.0)

- **go_router :** Nested routes (adding child routes).

```
GoRoute (
        name: "home",
        path : '/first',
        builder: (context, State) => Page1(),
        routes: <RouteBase> [ // Defining a child route
                        GoRoute (
                                name: 'second page',
                                path: 'second',
                                builder: (context, State) => Page2()
                                ), // GoRoute
                        ], // routes
        ); // GoRoute
```

To go to the second page, use :
context.go('/first/second');

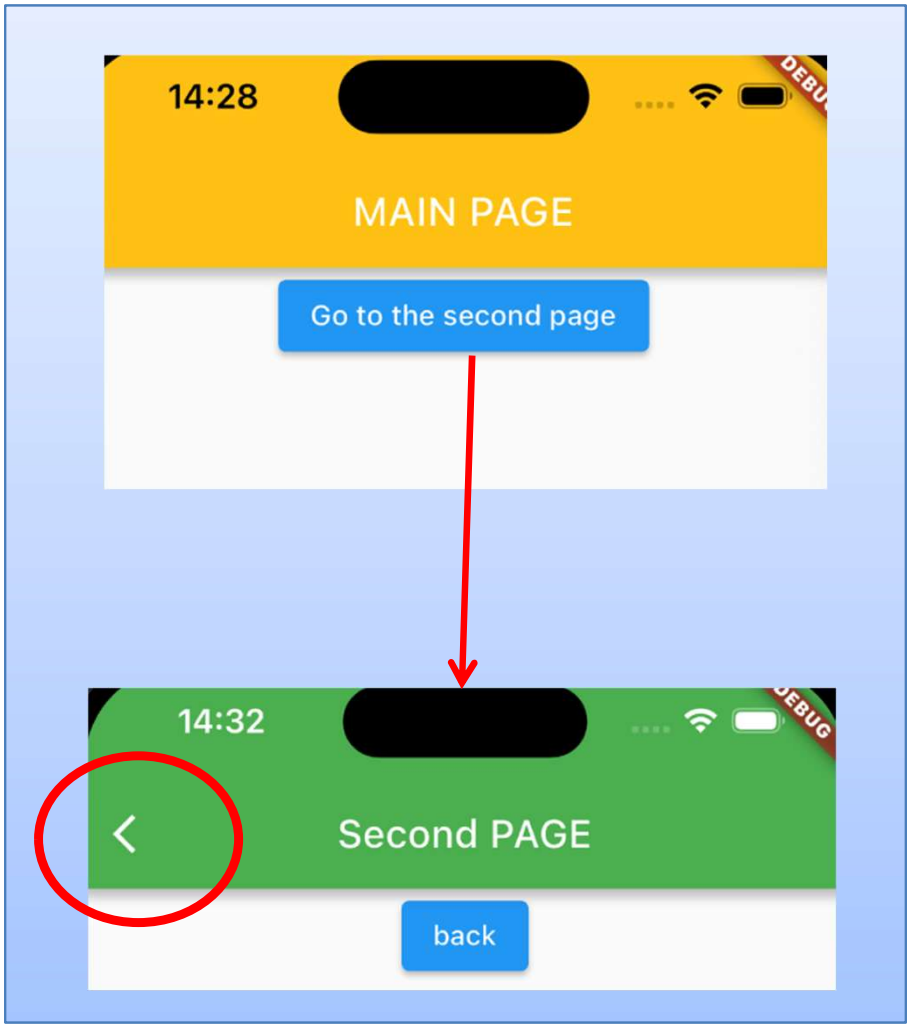
the '/' not needed

- **routes** parameter allows adding multiple screens to the same Navigator (stack), which is equivalent to PUSH.

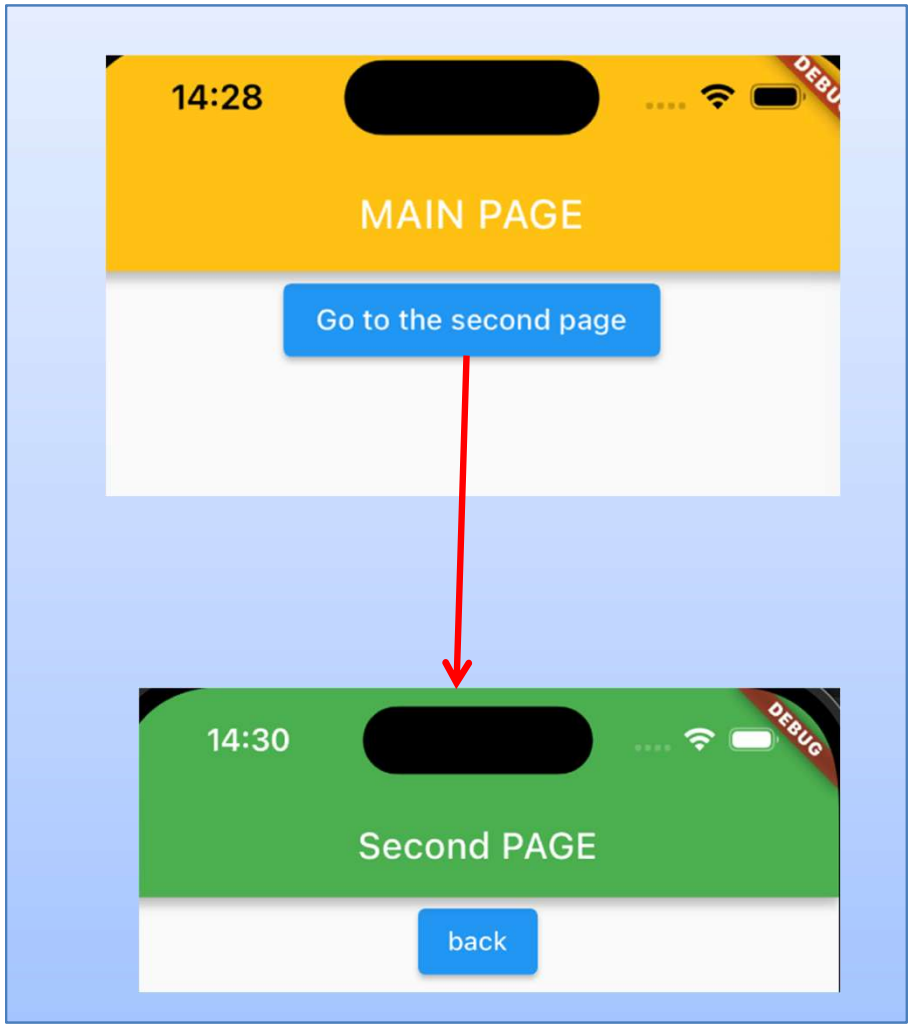
Declarative approach (Navigator 2.0)

- **go_router :** Nested routes (adding child routes).

Page 2 is a child route of Page 1



Page 2 is not a child route



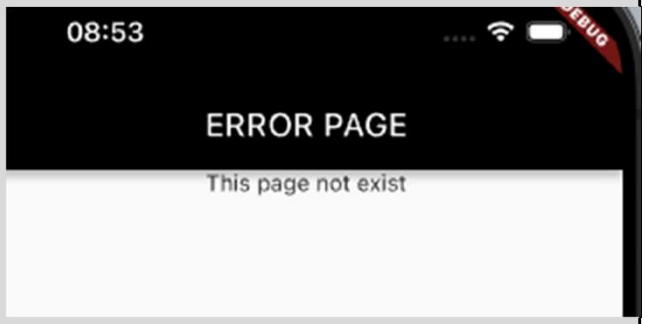
Declarative approach (Navigator 2.0)

- **go_router :** error handler (customise an invalid route)

  It is called when the GoRouter doesn't find a match.

```
final MyRouter = GoRouter (
        errorBuilder: (context, state) => Error(),  // Error handling : calling the
                                                    customised page Error() if  a path doesn't match
                .
                .
                .
                .
        ); // GoRouter

// For example, Error()  is displayed  when trying to go to an unknown page (/wrongpage)
        context.go('/wrongpage');
```



Conclusion

- **go_router :**
  - Provides a **modern declarative** approach to navigation (i.e. reduces the declarative implementation complexity).
  - Handles **path** and **query parameters** easily.
  - Supports **deep linking** and **conditional redirection**.
  - Makes navigation **cleaner**, **state-driven**, and **maintainable**.
  - **Recommended solution** for scalable Flutter apps.