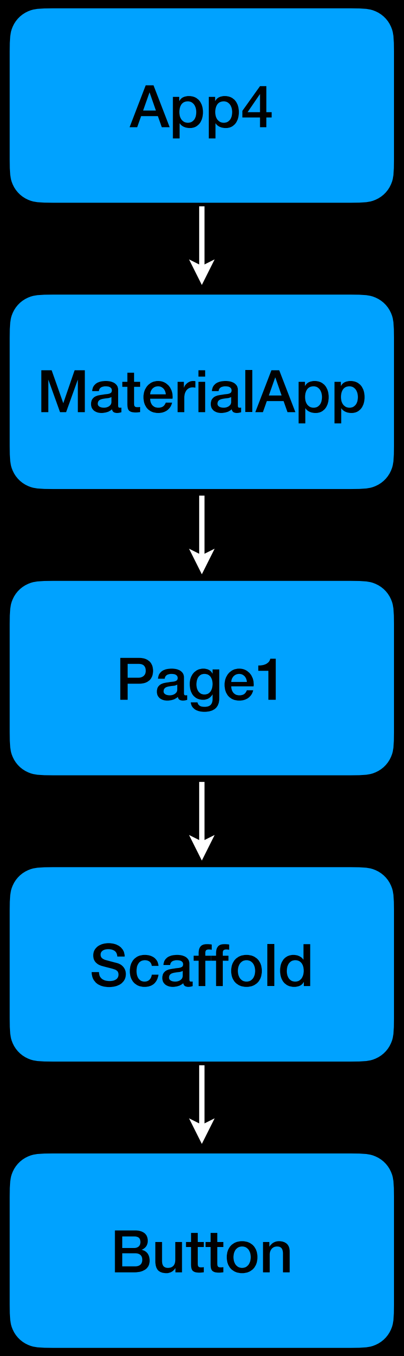
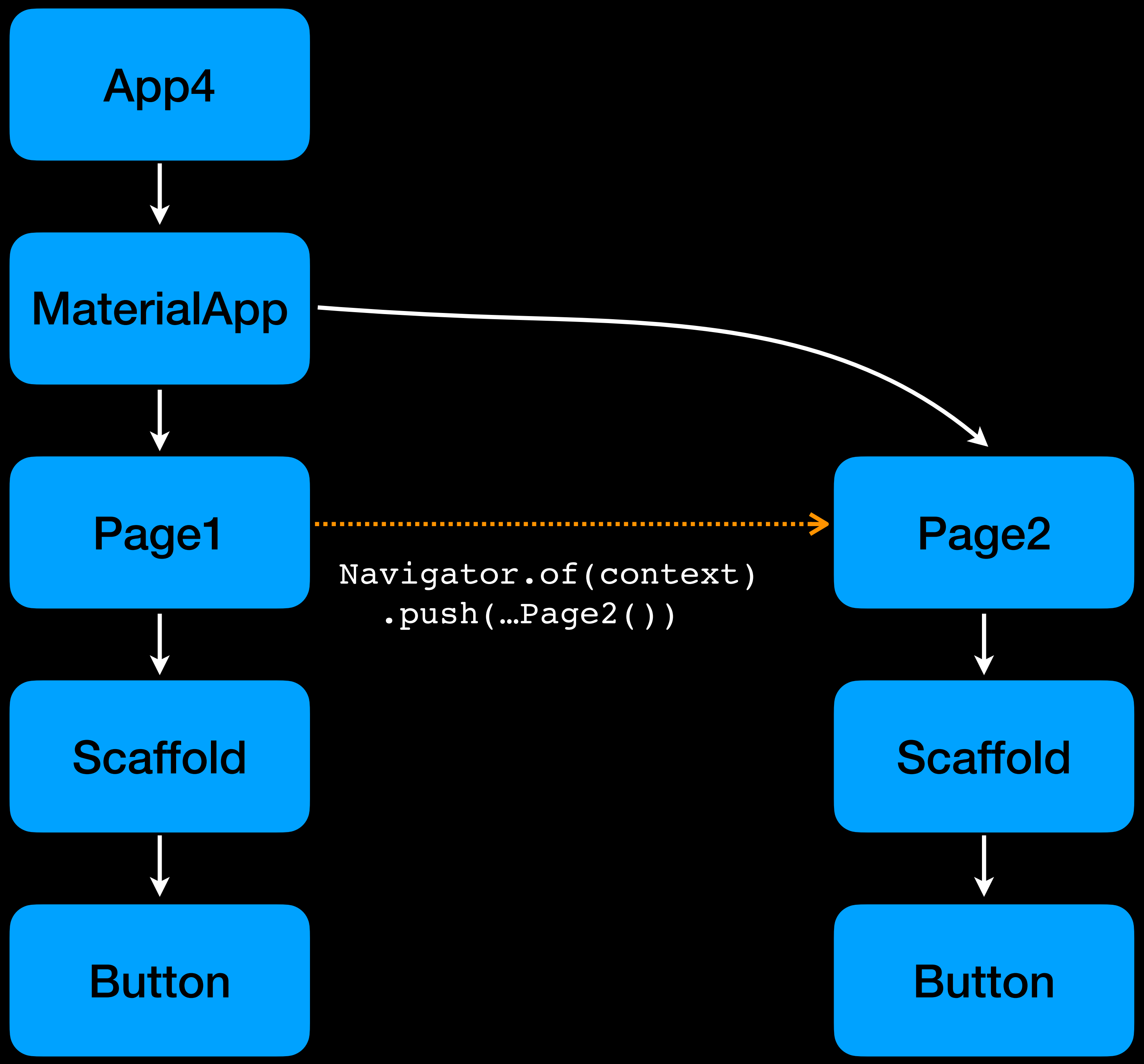
03_flutter_nav_route_eg1

Widget Tree



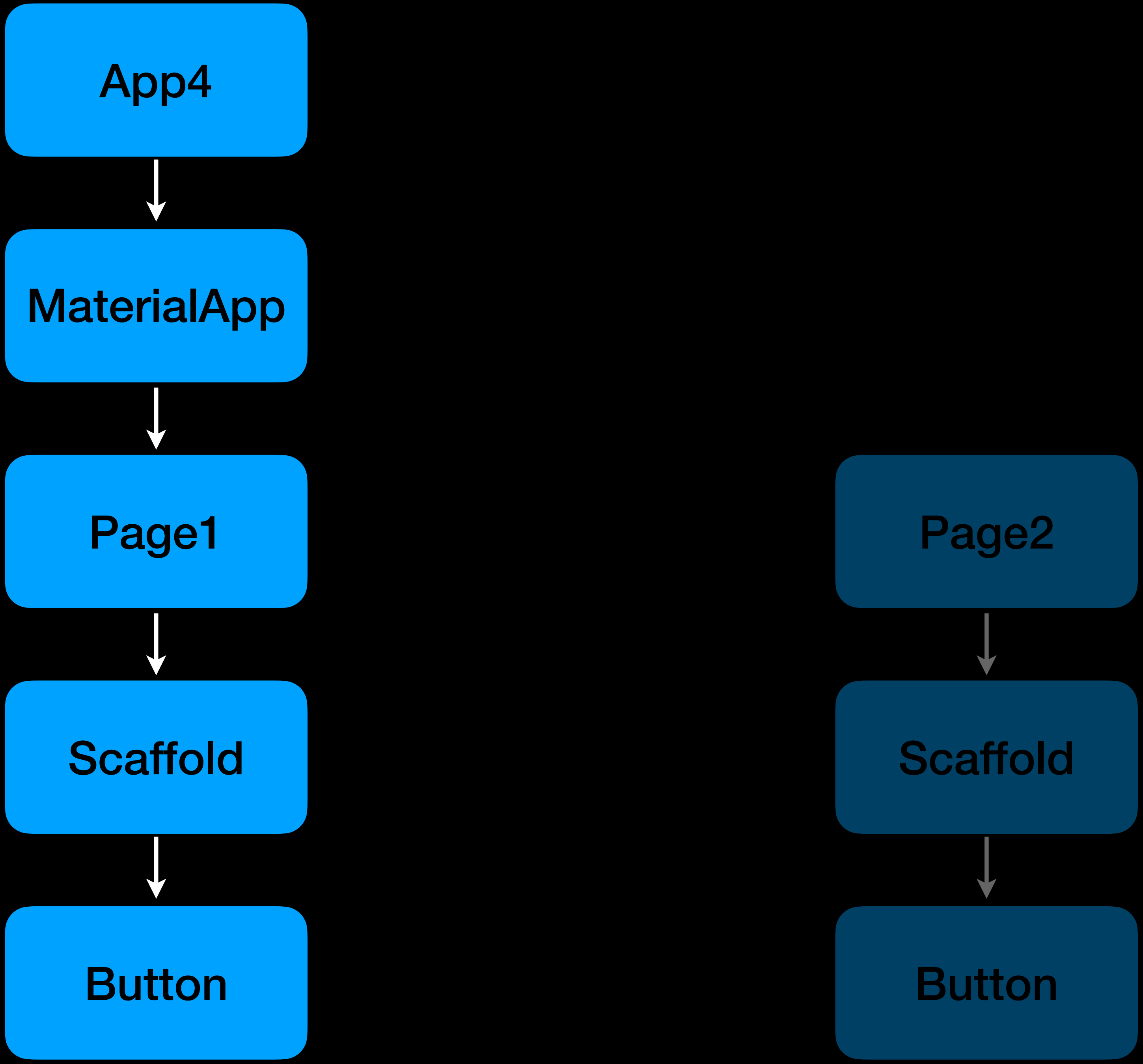
Widget Tree



```
Navigator.of(context)
  .push(…Page2())
```

This old page is still in stack with
all its values and variables

Widget Tree

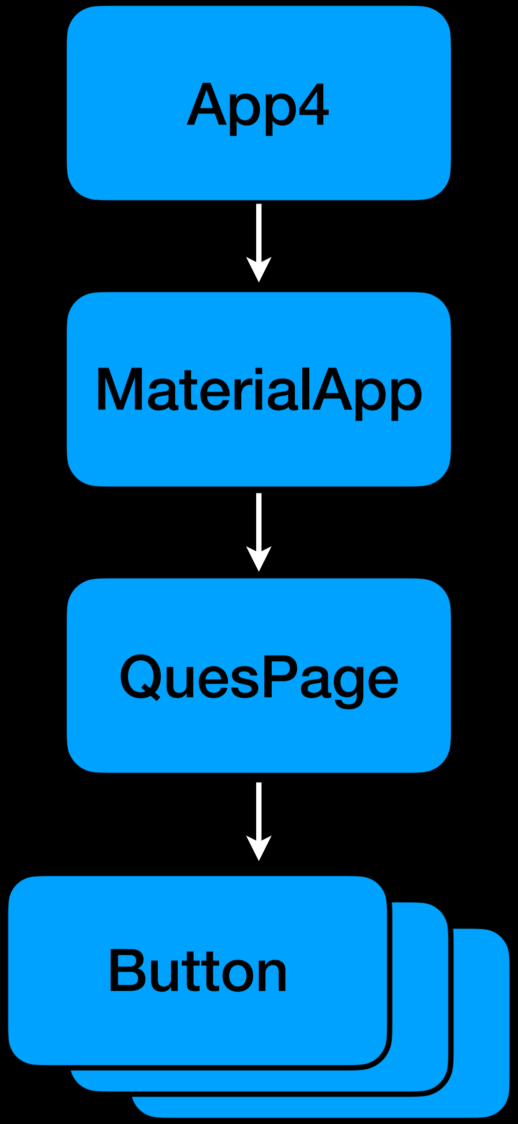


```
Navigator.of(context)
  .pop()
```

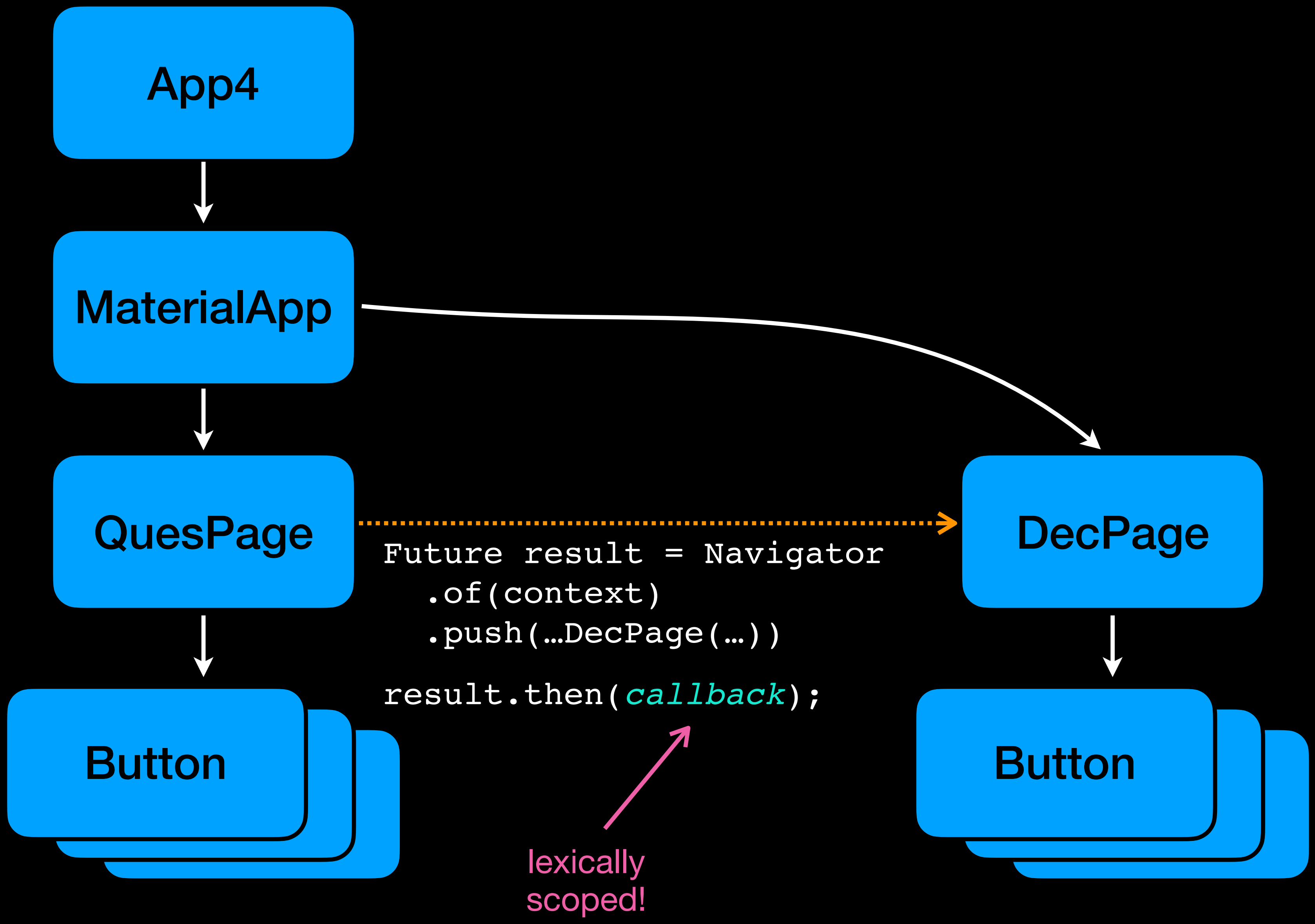
This is discarded when popped out.
Cannot bring back.

03_flutter_nav_route_eg2

Widget Tree



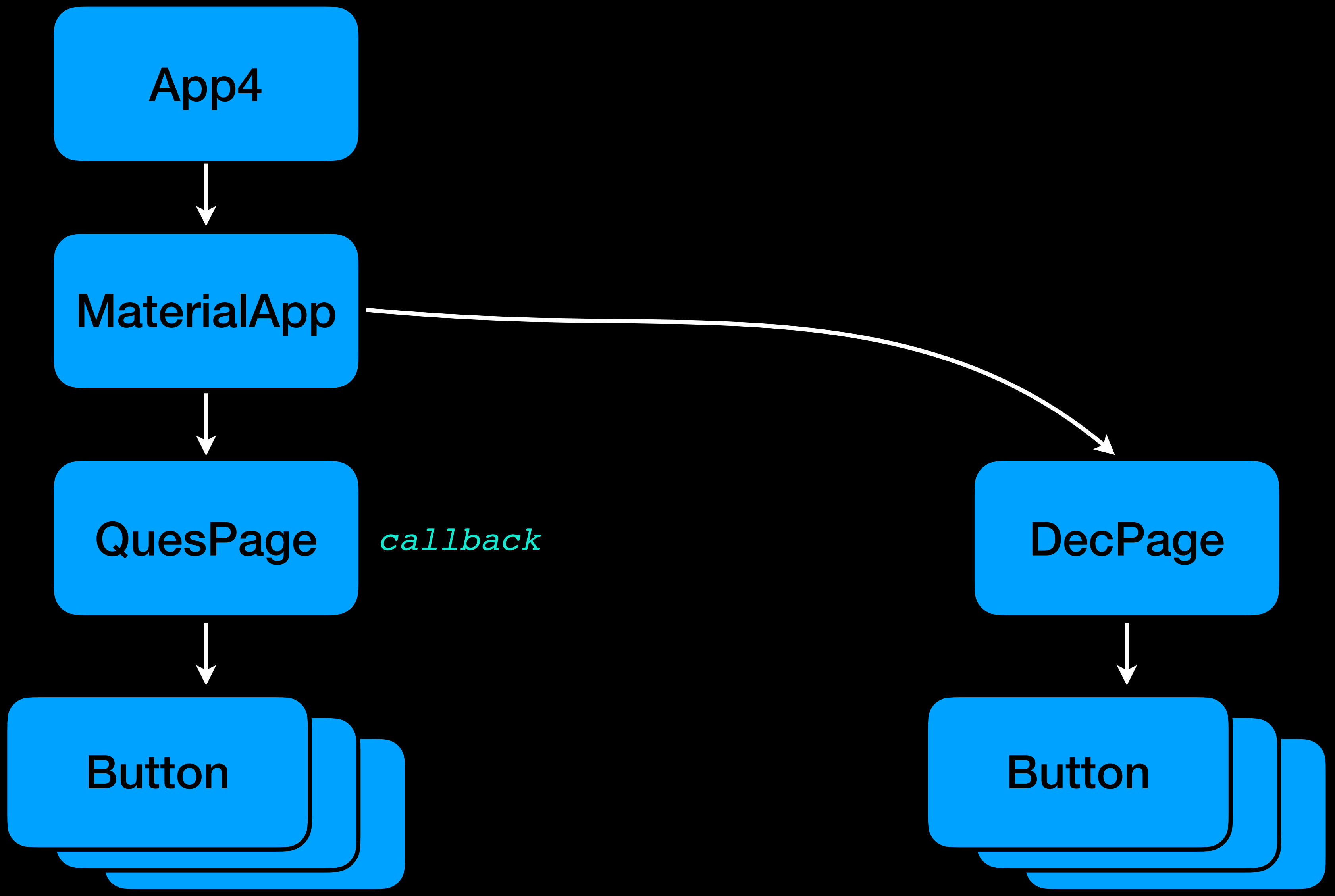
Widget Tree



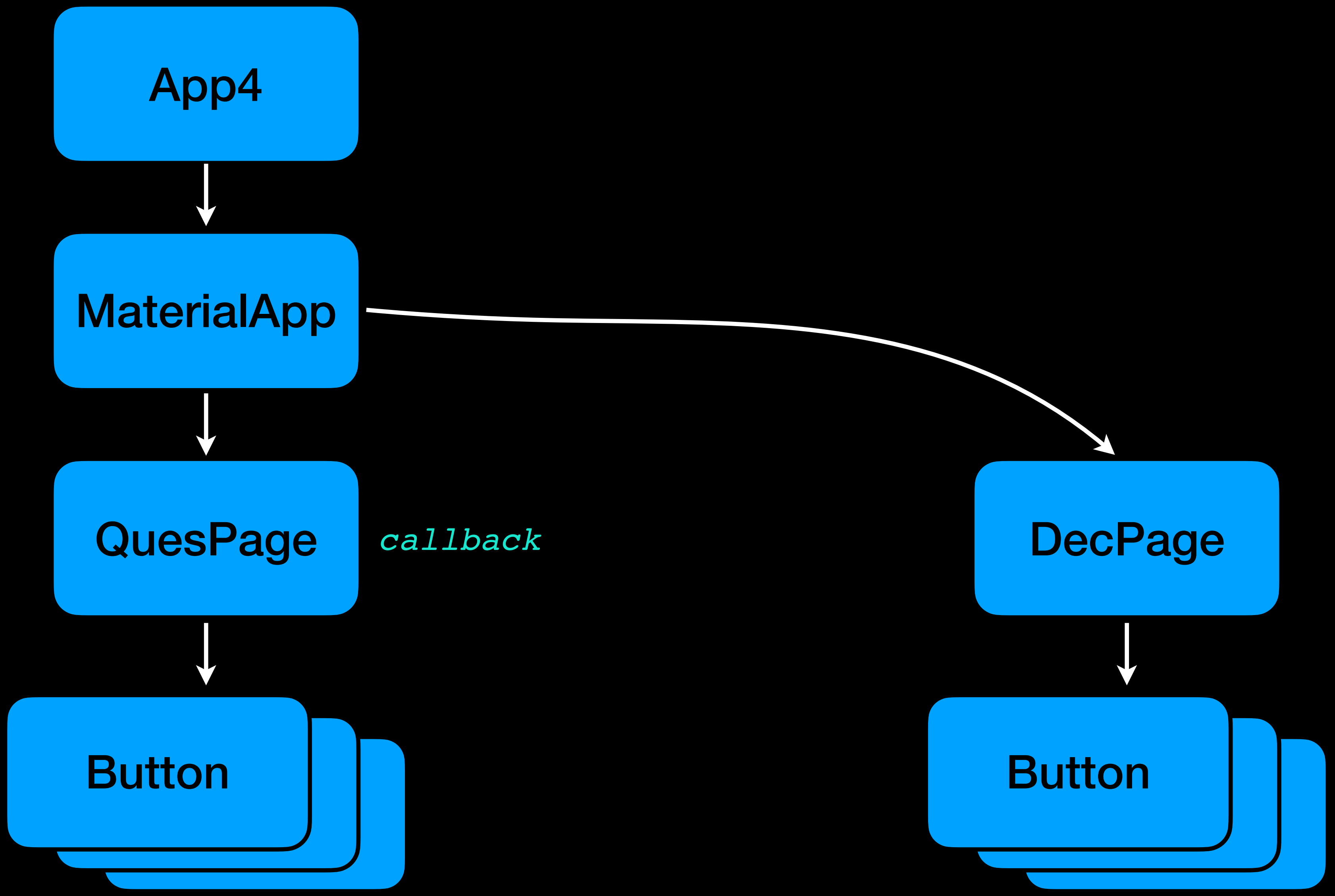
```
Future result = Navigator
   .of(context)
   .push(…DecPage(…))

result.then(callback);
```

Widget Tree

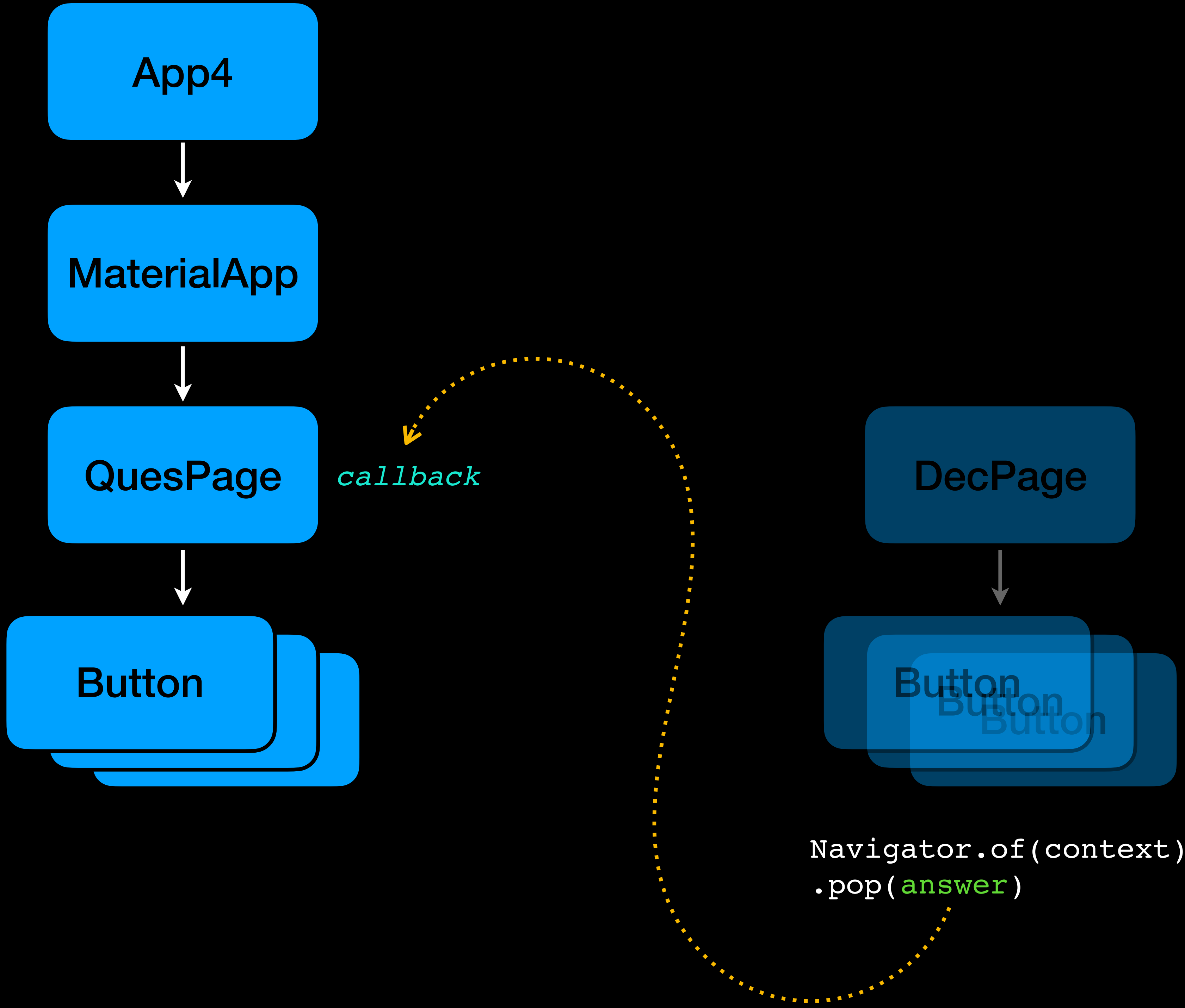

App4
MaterialApp
QuesPage
callback
Button
DecPage
Button

Widget Tree

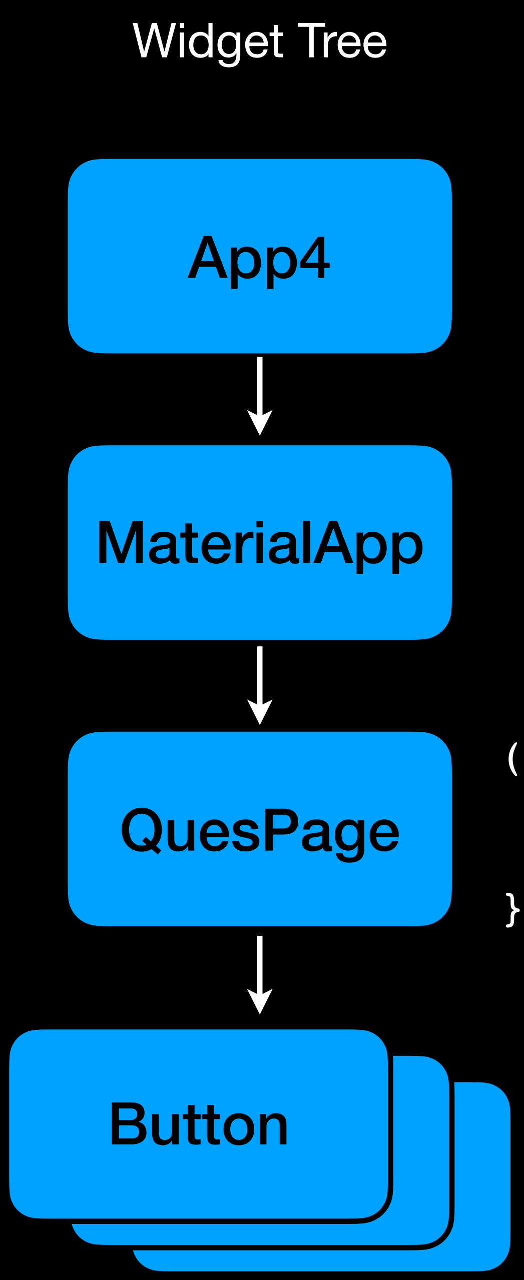


```
Navigator.of(context)
.pop(answer)
```

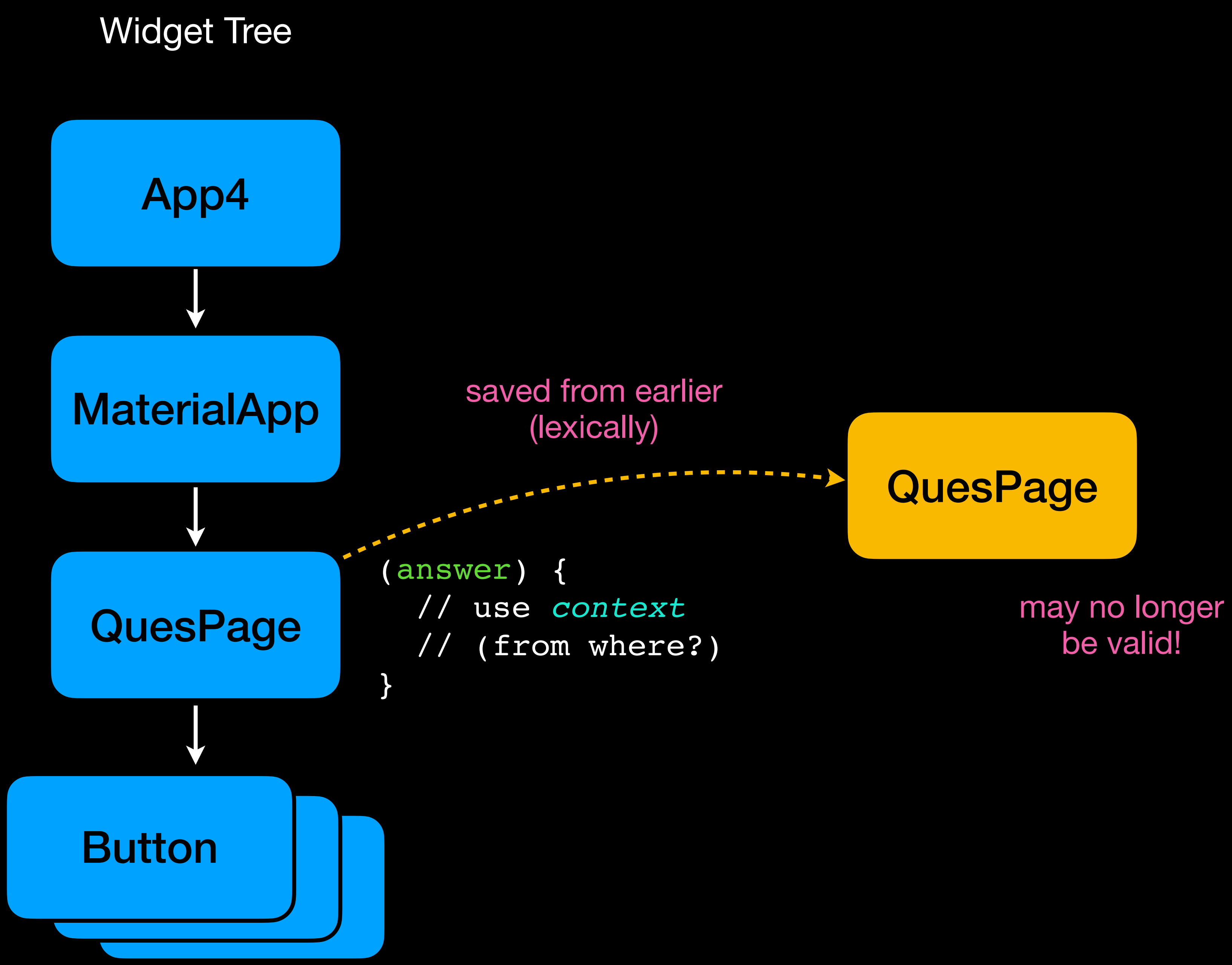
Widget Tree



```
Navigator.of(context)
.pop(answer)
```



```
(answer) {
  // use context
  // (from where?)
}
```



```
(answer) {
  // use context
  // (from where?)
}
```

“Async Gap”

futures (aka promises) & async/await

type of returned variable

```
abstract class Future<T> {
  Future<R> then<R>(R Function (T));
  Future<T> catchError(Function onError);
}
```

do this second thing when 1st future result has come

on your process, do this if there is an error

Example - Person 1 assigns a task to person 2 and asks to send mail when done, once mail sent, do the second task as mentioned...this keeps progressing. Person1 never sits idle until any of these are done. So, the main function keeps going

```
abstract class FullOfPromises {
  Future<String> longOperation(String input);
}

void consumer(FullOfPromises fop) {
  Future<String> future = fop.longOperation('input');
}
```

here in this consumer fn , there is no action/return given -->we wont know when work is done, which is why " then" is used

```dart
abstract class FullOfPromises {
  Future<String> longOperation(String input);
}

void consumer(FullOfPromises fop) {
  Future<String> future = fop.longOperation('input');
  future.then((result) {
    print('Got result "$result"');
  });
}
```

done when the previous line has a result

```
abstract class FullOfPromises {
  Future<String> longOperation(String input);
}

void consumer(FullOfPromises fop) {
  Future<String> future = fop.longOperation('input');
  future.then((result) {
    print('Got result "$result"');
  });
}

void main() {
  consumer(...);
  print('After consumer call');
}
```

Here, Consumer is called--> then the long operation is triggerred, but the then part is not done until result comes, however--> the control goes to main function and prints out "after consumer call" before the consumer fn completion.

later when long operation completes, result is received and the "Got result" is printed-

Main function is flow is not dependent on the consumer fn completion

```
abstract class FullOfPromises {
  Future<String> longOperation(String input);
}

void consumer(FullOfPromises fop) {
  Future<String> future = fop.longOperation('input');
  future.then((result) {
    print('Got result "$result"');
  });
}

void main() {
  consumer(...);
  print('After consumer call');
}
```

```
abstract class FullOfPromises {
  Future<String> longOperation(String input);
}

void consumer(FullOfPromises fop) {
  Future<String> future = fop.longOperation('input');
  future.then((result) {
    print('Got result "$result"'); <········ called later!
  });
}

void main() {
  consumer(...);
  print('After consumer call');
}
```

This implementation is same as above-->but neater way

```
abstract class FullOfPromises {
  Future<String> longOperation(String input);
}

void consumer(FullOfPromises fop) async {
  var result = await fop.longOperation(‘input');
```

Consumer fn runs until it sees a await keyword.--->when it sees the keyword, it stops and goes back to its previous work--->maybe main fn-->completes it and comeback and does the long operation mentioned after the await keyword

```
  print('Got result "$result"');
}


void main() {
  consumer(...);
  print('After consumer call');
}
```

```
abstract class FullOfPromises {
  Future<String> longOperation(String input);
}

void consumer(FullOfPromises fop) async {
  var result = await fop.longOperation('input');

  print('Got result "$result"');
}


void main() {
  consumer(...);
  print('After consumer call');
}
```

called later!

```dart
void consumer(FullOfPromises fop) {
  fop.longOperation('input')
  .then((result) {
      fop.nextLongOperation(result)
      .then((result2) {
          print('Got result2 "$result2"');
      })
      .catchError((err) {
          print('Got error: "$err"');
      });
  }).catchError((err) {
      print('Got error: "$err"');
  });
}
```

```dart
void consumer(FullOfPromises fop) async {
  try {
    var result = await fop.longOperation('input');
    print('Got result "$result"');

    var result2 = await fop.nextLongOperation(result);
    print('Got result2 "$result2"');
  } catch (err) {
    print('Got error: "$err"');
  };
}
```

creating futures

```dart
void longComputation(void Function(String) callback) {
  Timer(const Duration(seconds: 1), () {
    callback('result');
  });
}
```

```dart
Future<String> longComputation2() {
  final completer = Completer<String>();
  Timer(const Duration(seconds: 1), () {
    completer.complete('result');
  });
  return completer.future;
}
```

```dart
abstract class Future<T> {
  factory Future.delayed(Duration duration,
                         T Function ());
  factory Future.value(T value);
}
```

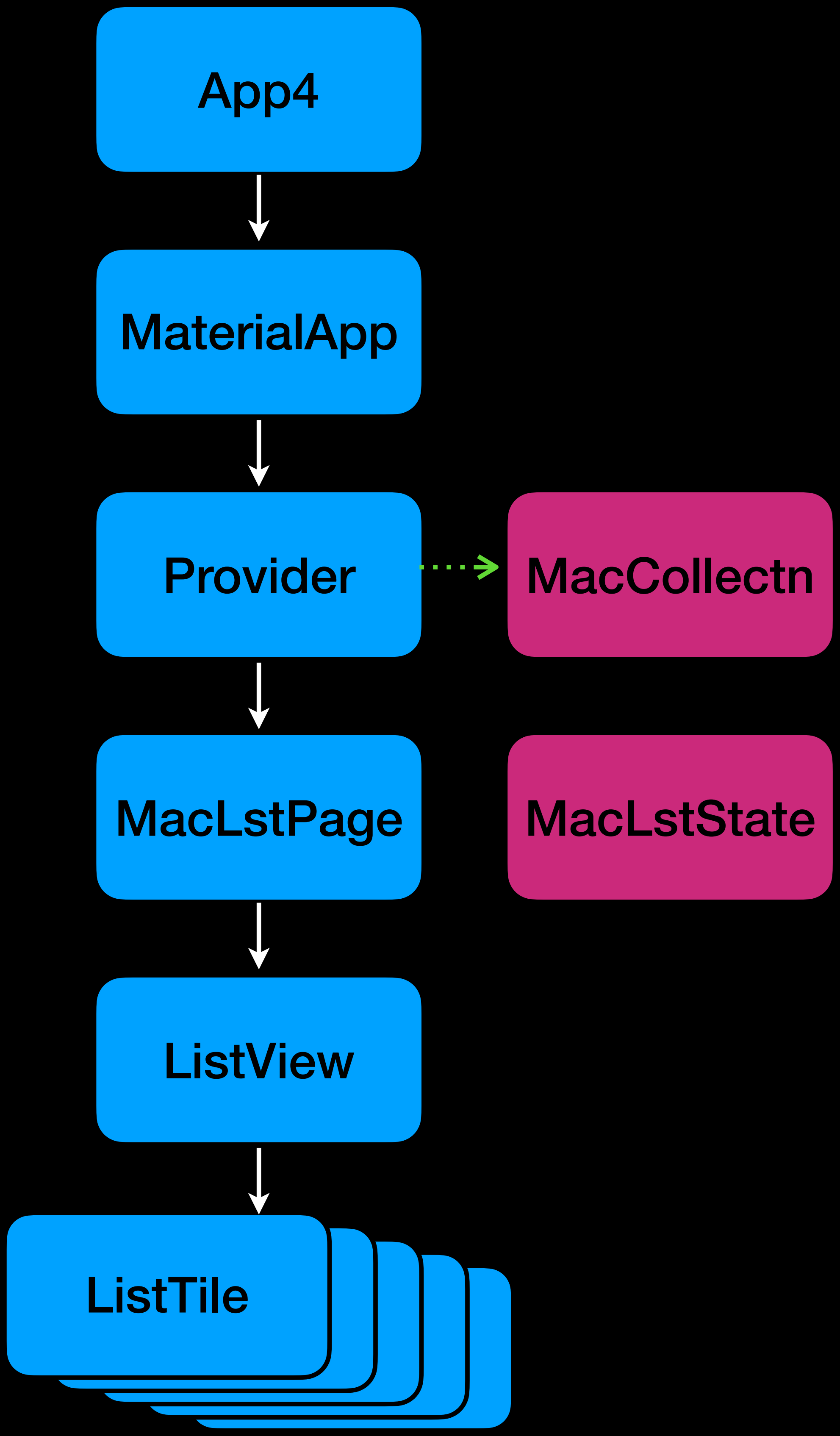
```dart
Future<String> longComputation3() {
  return Future.delayed(const Duration(seconds: 1),
                        () => 'result');
}
```

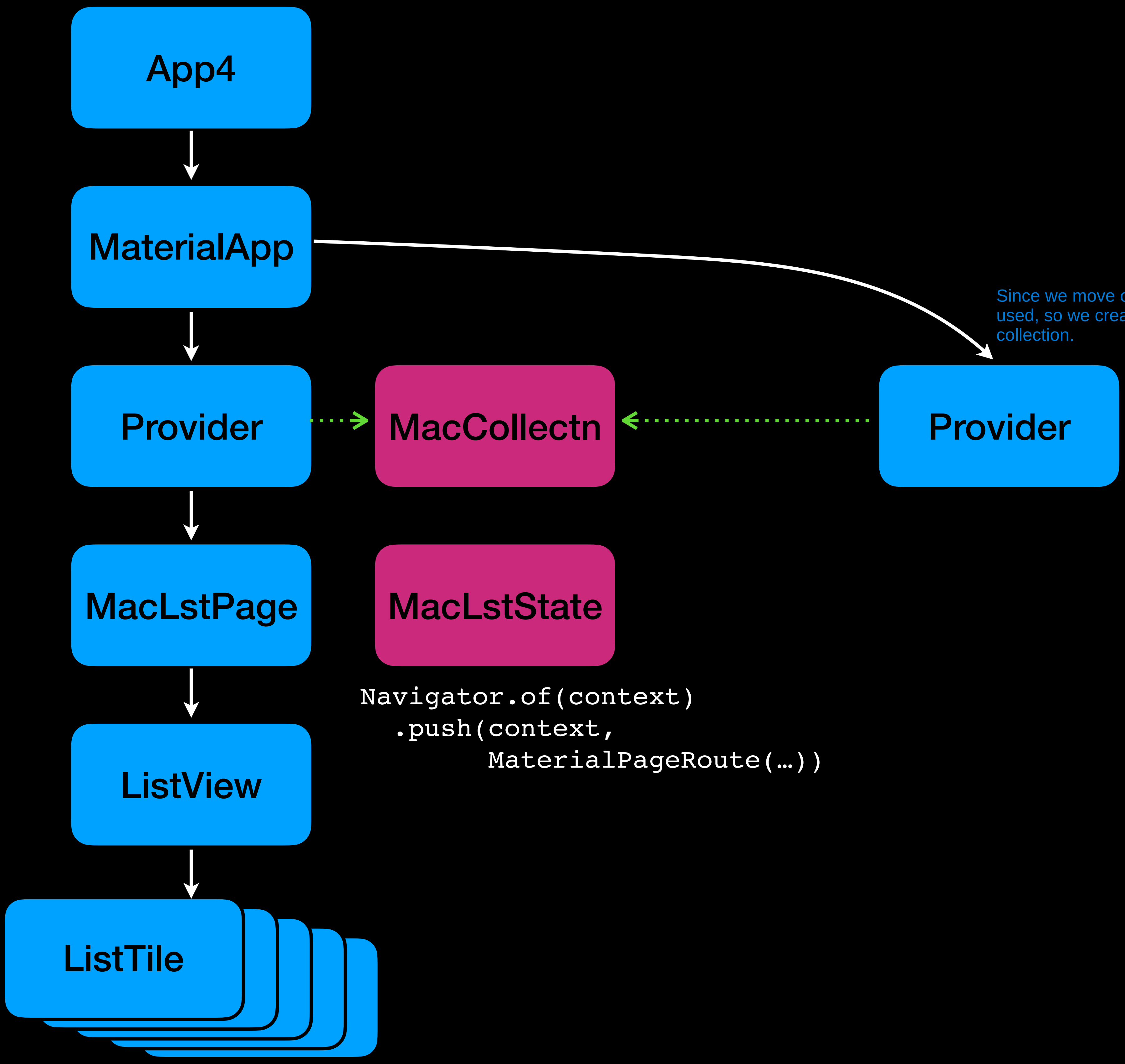
```dart
Future<String> longComputation4() async {
  await Future.delayed(const Duration(seconds: 1));
  return 'result';
}
```

```dart
Future<String> shortComputation() {
  return Future.value('Hello');
}
```

03_flutter_nav_route_eg4

Widget Tree

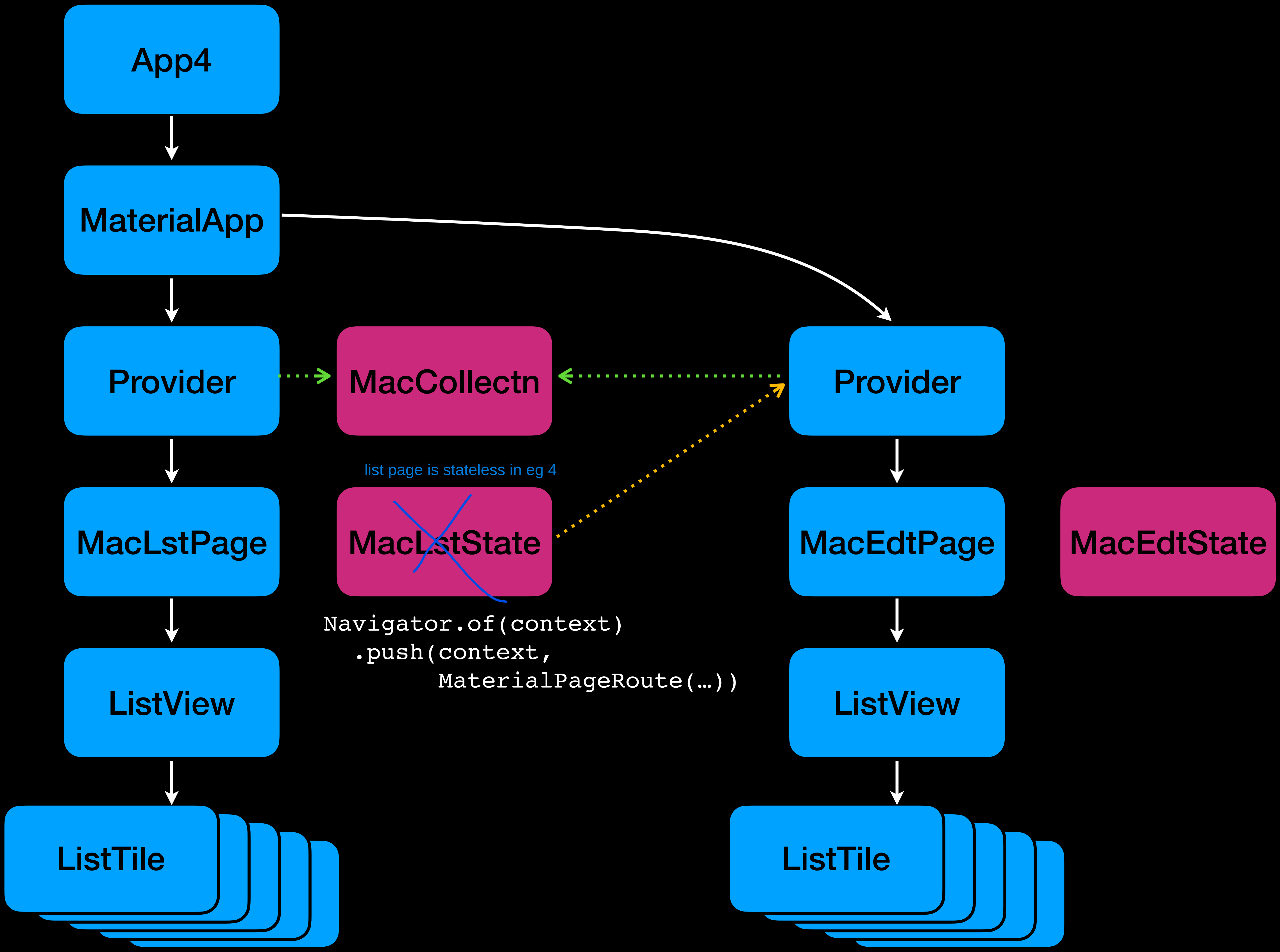

App4
MaterialApp
Provider
MacCollectn
MacLstPage
MacLstState
ListView
ListTile



Since we move on to next page, the same provider cannot be used, so we create a new one which refers to the same collection.

```
Navigator.of(context)
   .push(context,
         MaterialPageRoute(…))
```

Widget Tree



```
Navigator.of(context)
   .push(context,
          MaterialPageRoute(…))
```