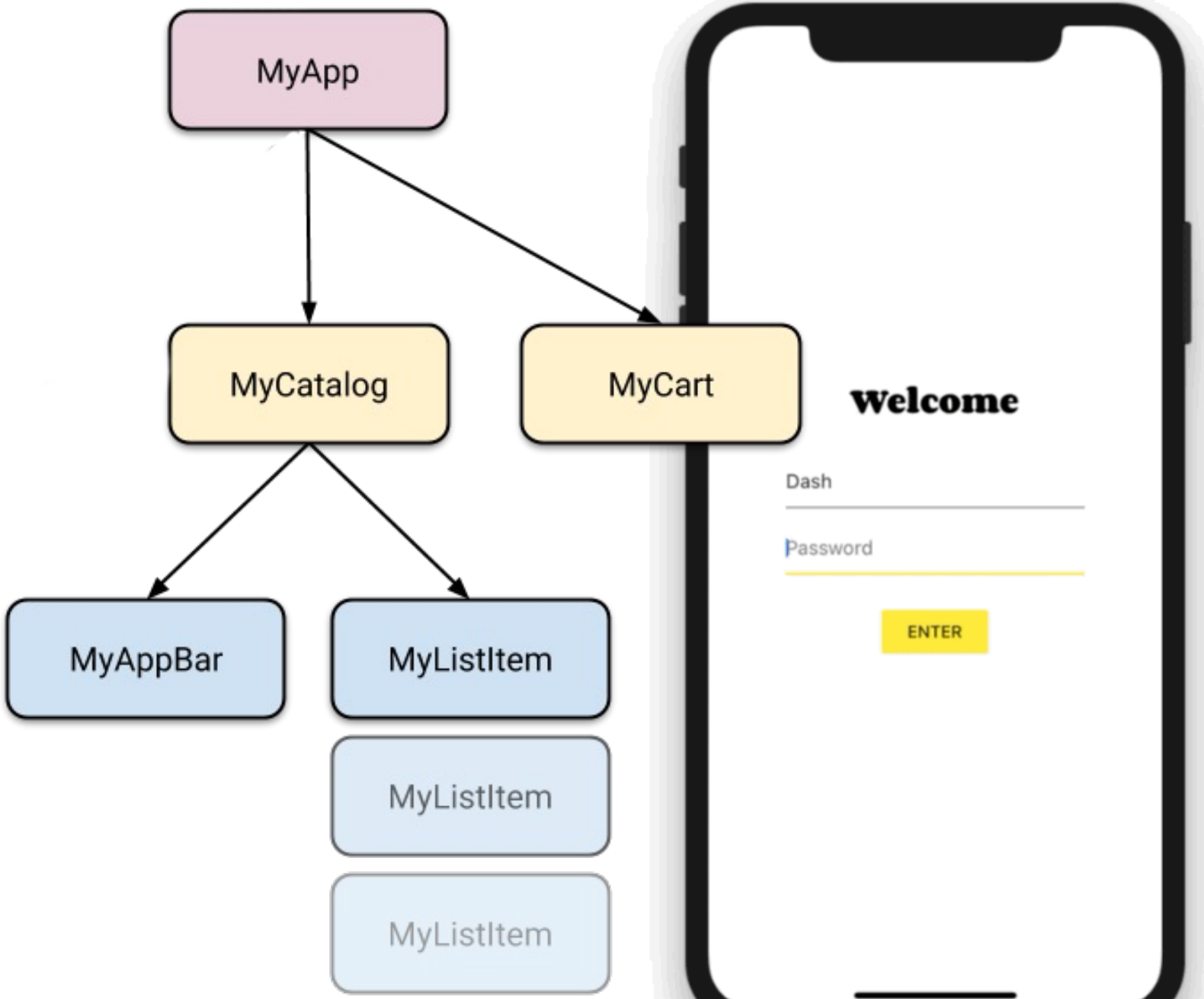
State Management with Provider

Written by Thapanapong Rukkanchanunt

Widget Tree

- App has a combination of stateless and stateful widget that can be expressed as widget tree.
- Where should we put the current state of the cart?



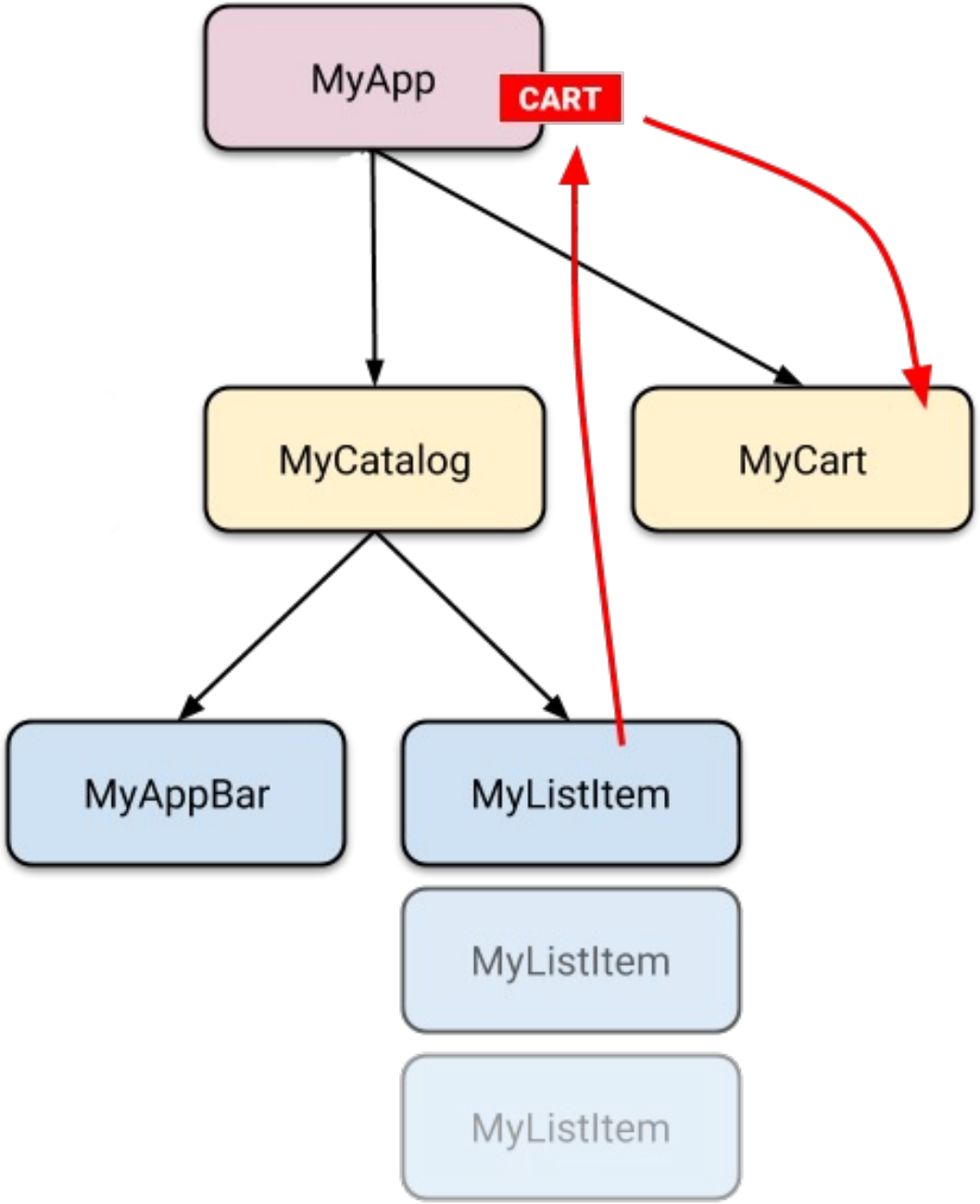
Lifting State Up

- In Flutter, it makes sense to keep the state above the widgets that use it.
- In declarative frameworks like Flutter, if you want to change the UI, you have to rebuild it. There is no easy way to have MyCart.updateWith(somethingNew).

```
// BAD: DO NOT DO THIS
void myTapHandler() {
  var cartWidget = somehowGetMyCartWidget();
  cartWidget.updateWith(item);
}
```

Living with Parents

- In Flutter, you construct a new widget every time its contents change. Instead of MyCart.updateWith(somethingNew) (a method call) you use MyCart(contents) (a constructor).
- Because you can only construct new widgets in the build methods of their parents, if you want to change contents, it needs to live in MyCart's parent or above.



Better Coding

- MyCart has only one code path for building any version of the UI.

```
// GOOD
void myTapHandler(BuildContext context) {
  var cartModel = somehowGetMyCartModel(context);
  cartModel.add(item);
}
```

```
// GOOD
Widget build(BuildContext context) {
  var cartModel = somehowGetMyCartModel(context);
  return SomeWidget(
    // Just construct the UI once, using the current state of the cart.
    // ...
  );
}
```

Accessing the State

- Implementing somehowGetMyCartModel function can be tricky.
- Luckily, Flutter has mechanisms for widgets to provide data and services to their descendants (in other words, not just their children, but any widgets below them). This is called provider.
- Before working with provider, don't forget to add the dependency on it to your pubspec.yaml.
  - $ flutter pub add provider
- With provider, you need to understand 3 concepts:
  - ChangeNotifier
  - ChangeNotifierProvider
  - Consumer

ChangeNotifier

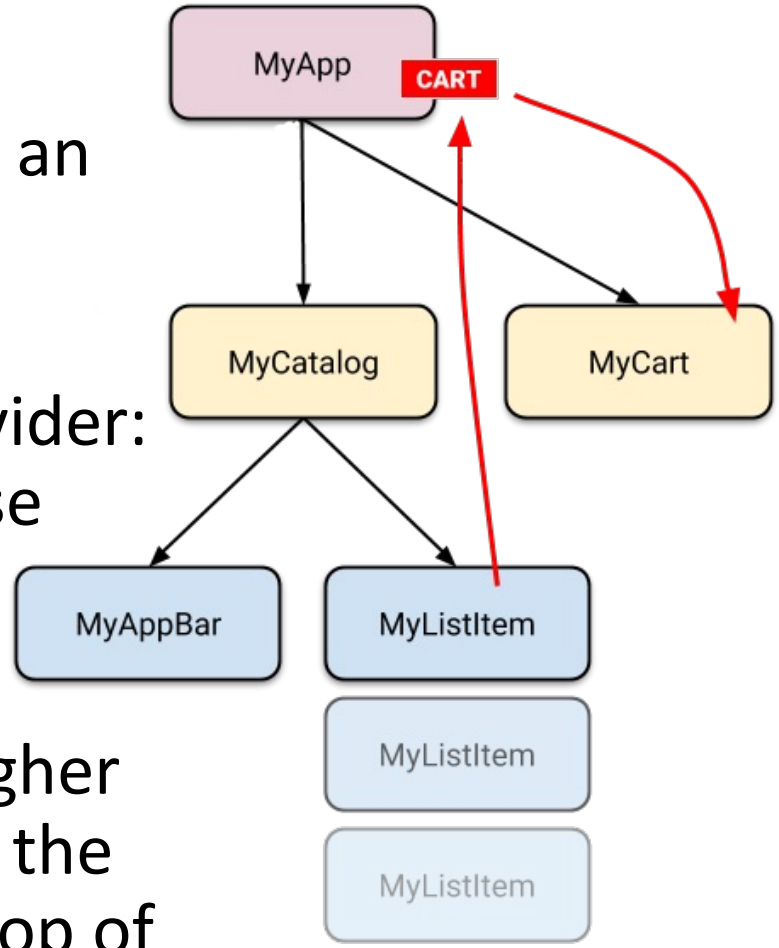
- ChangeNotifier is a simple class included in the Flutter SDK which provides change notification to its listeners.
- In other words, if something is a ChangeNotifier, you can subscribe to its changes. (It is a form of Observable)
- In provider, ChangeNotifier is one way to encapsulate your application state. For very simple apps, you get by with a single ChangeNotifier. In complex ones, you'll have several models, and therefore severalChangeNotifiers.

Cart Model

```
class CartModel extends ChangeNotifier {
  final List<Item> _items = [];   /// Internal, private state of the cart.
  UnmodifiableListView<Item> get items => UnmodifiableListView(_items);
  int get totalPrice => _items.length * 42;
  void add(Item item) {
    _items.add(item);
    notifyListeners(); // This call tells the widgets that are listening to this model to rebuild.
  }
  void removeAll() {
    _items.clear();
    notifyListeners(); // This call tells the widgets that are listening to this model to rebuild.
  }
}
```

ChangeNotifierProvider

- ChangeNotifierProvider is the widget that provides an instance of a ChangeNotifier to its descendants. It comes from the provider package.
- We already know where to put ChangeNotifierProvider: above the widgets that need to access it. In the case of CartModel, that means somewhere above both MyCart and MyCatalog.
- You don’t want to place ChangeNotifierProvider higher than necessary (because you don’t want to pollute the scope). But in our case, the only widget that is on top of both MyCart and MyCatalog is MyApp.



ChangeNotifierProvider in Action

```
void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => CartModel(),
      child: const MyApp(),
    ),
  );
}
```

Consumer

- Now that CartModel is provided to widgets in our app through the ChangeNotifierProvider declaration at the top, we can start using it.
- This is done through the Consumer widget.

```
return Consumer<CartModel>(
  builder: (context, cart, child) {
    return Text('Total price: ${cart.totalPrice}’);
  },
);
```

- We must specify the type of the model that we want to access.
- The second argument of the builder function is the instance of the ChangeNotifier.

Provider.of

- Sometimes, you don’t really need the data in the model to change the UI but you still need to access it.
- For example, a ClearCart button wants to allow the user to remove everything from the cart.
- It doesn’t need to display the contents of the cart; it just needs to call the clear() method.
- For this use case, we can use Provider.of, with the listen parameter set to false.

```
Provider.of<CartModel>(context, listen: false).removeAll();
```