ACADEMIND

What Is Flutter?

What Is Flutter?

UI Framework

Code packages & utility functions for writing cross-platform app code

Collection of Tools

CLI & software that helps with developing, testing & building cross-platform apps

ACADEMIND

What Is Flutter?

+

UI Framework

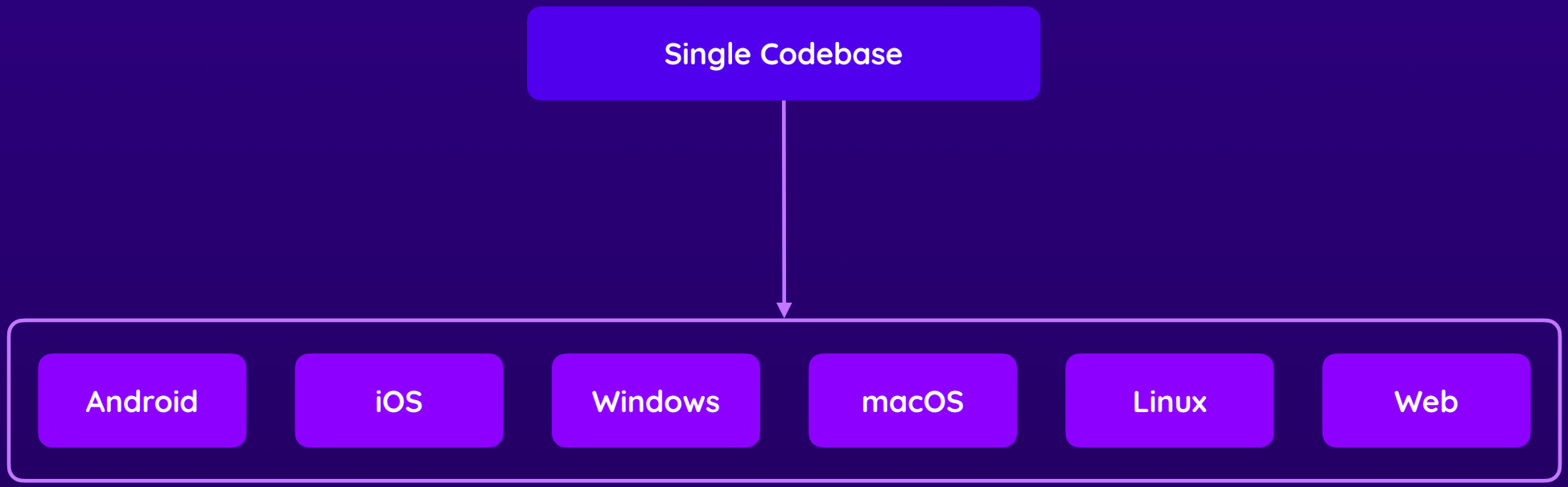
Code packages & utility functions for writing **cross-platform** app code

Collection of Tools

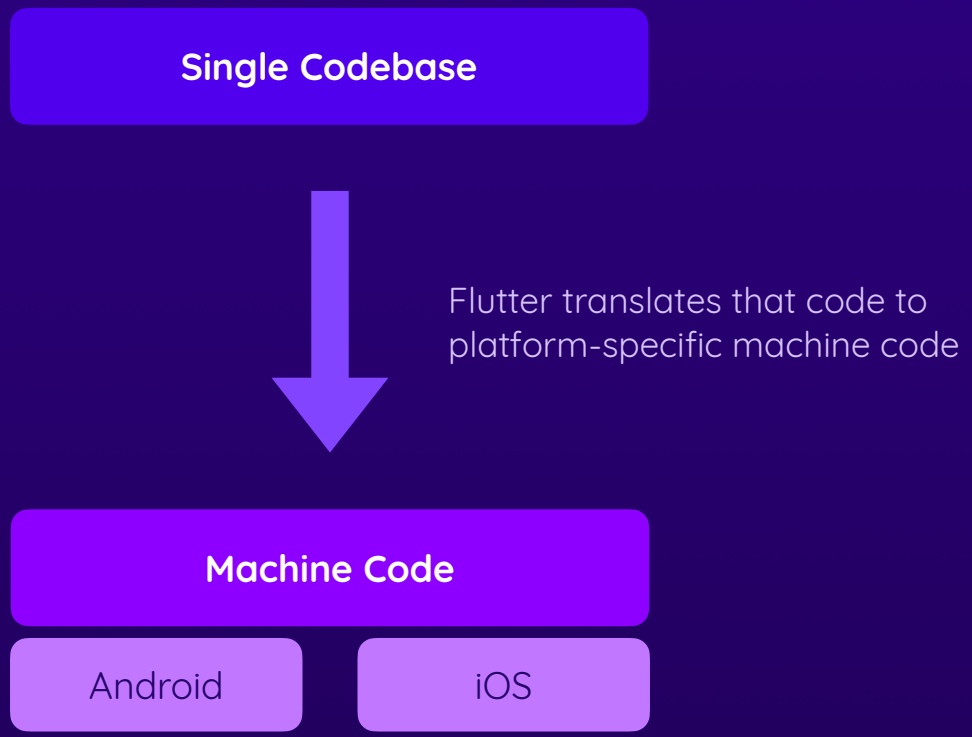
CLI & software that helps with developing, testing & building **cross-platform** apps

Flutter allows you to build **multi-platform apps** based on **one single codebase** and programming language

One Codebase, Multiple Apps



From Flutter Code To Platform Code



Flutter Is Not A Programming Language!

It's a framework for building user interfaces with **Dart**

Framework

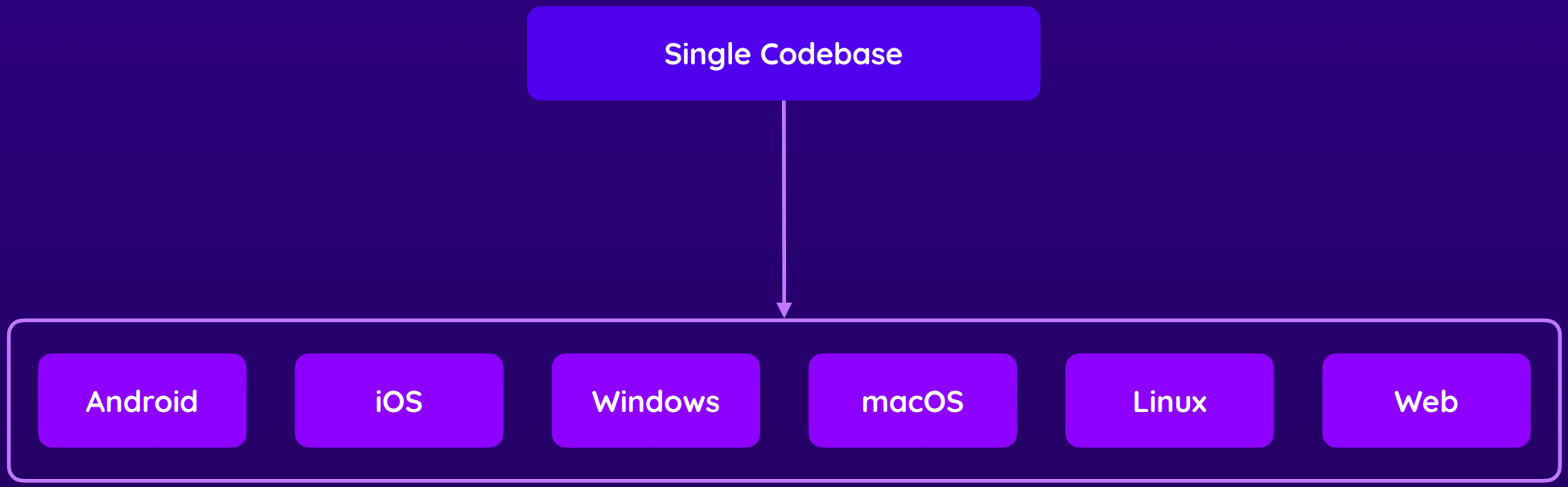
A collection of packages & utility functions you may use in your code

Dart

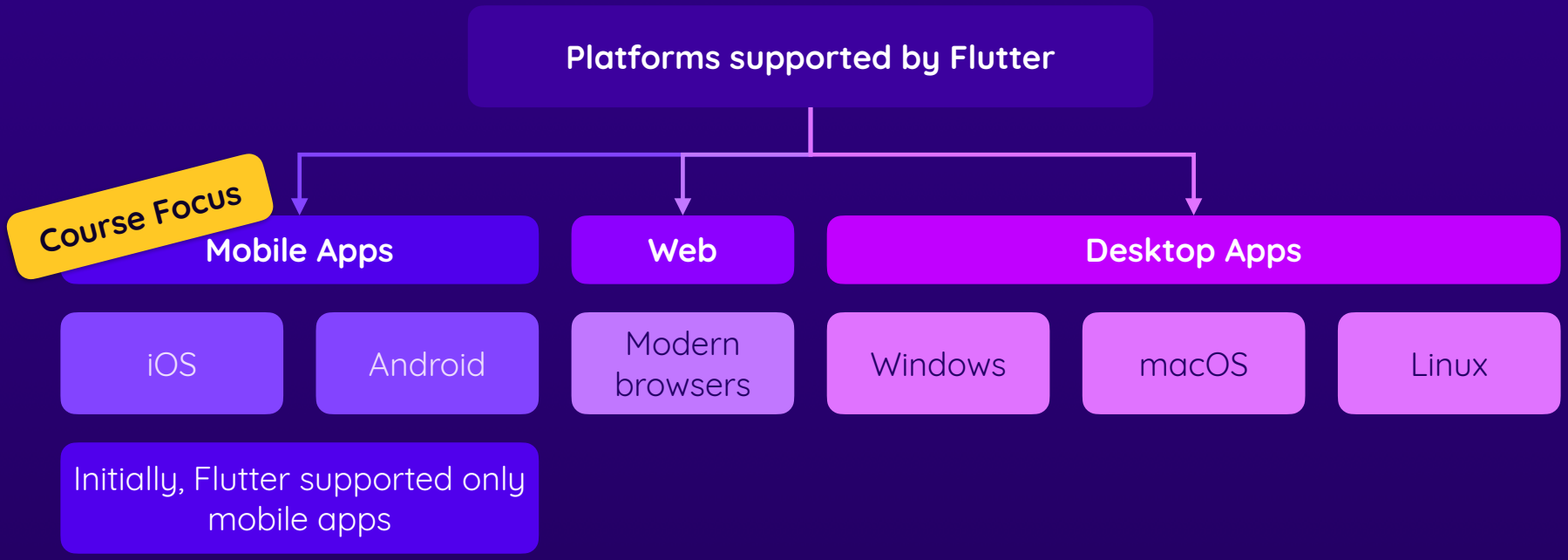
A programming language developed by Google

Main usage: Flutter app development

One Codebase, Multiple Apps



Target Platforms



Whilst you can write the code for all platforms on the same machine, you can **only test & run iOS & macOS apps on macOS machines, Windows apps on Windows machines and Linux apps on Linux machines!**
Android and web apps can be built on all operating systems.

ACADEMIND

Flutter Setup

1. **Flutter SDK**
   - Flutter SDK — For managing Flutter projects
   - Git — Version control software, used internally by Flutter SDK
2. **Platform Tools**
   - Android Studio — Used by Flutter SDK & needed for Android app deployment
   - XCode — Used by Flutter SDK & needed for iOS app deployment
3. **Virtual Devices**
   - Android — Preview Flutter apps on virtual Android devices
   - iOS — Preview Flutter apps on virtual iOS devices

Target Platform Tools & Devices Setup

	on Windows	on macOS	on Linux
build **iOS Apps**	Not possible	Download & install XCode Configure XCode command-line tools Create local iOS simulator	Not possible
build **Android Apps**	Download & install Android Studio Install SDK, command-line tools & build tools Create local Android emulator		

ACADEMIND

It’s one shared codebase!

You can write code for iOS apps on Windows machines

You just can’t build + deploy iOS apps from there

About Material Design

Google's flexible design system

A set of suggestions, rules & guidelines that help you build beautiful user interfaces

Highly customizable and extendable

About This Course

Beyond the Basics

Animating apps, connecting a backend, using native device features (e.g., camera) & more

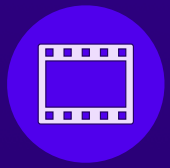
Advanced Features

Handling user interactions, customizing styles, building multi-screen apps

Dart & Flutter Fundamentals

Base syntax, core features & foundational concepts needed to build mobile app user interfaces

How To Get The Most Out Of This Course



Watch the Videos

- At your pace: Use the video player controls
- On-Demand: Repeat videos & sections as needed

Code Along & Practice

- Pause & try things on your own
- Practice what you learned (also in your own projects)
- Use attached slides & code snapshots

Help Each Other

- Ask & answer in the Q&A section
- Join our amazing Discord community!

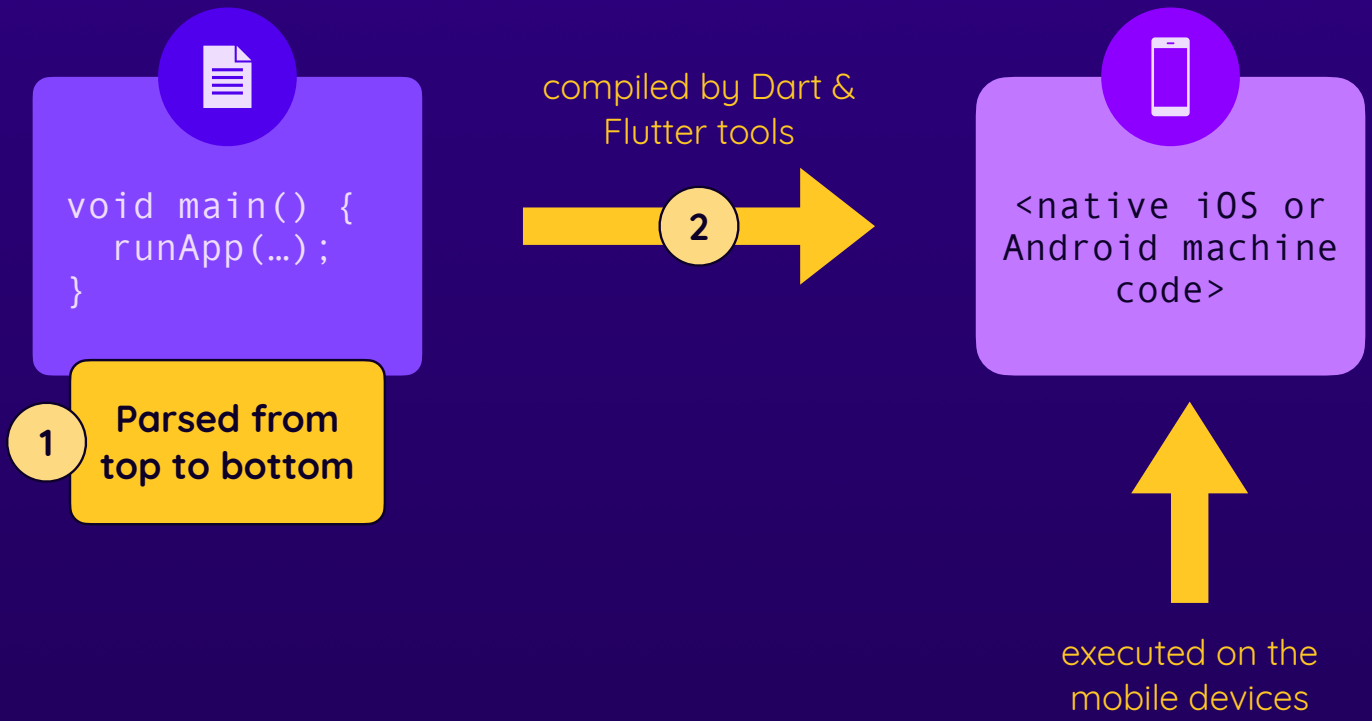
ACADEMIND

Flutter & Dart Fundamentals

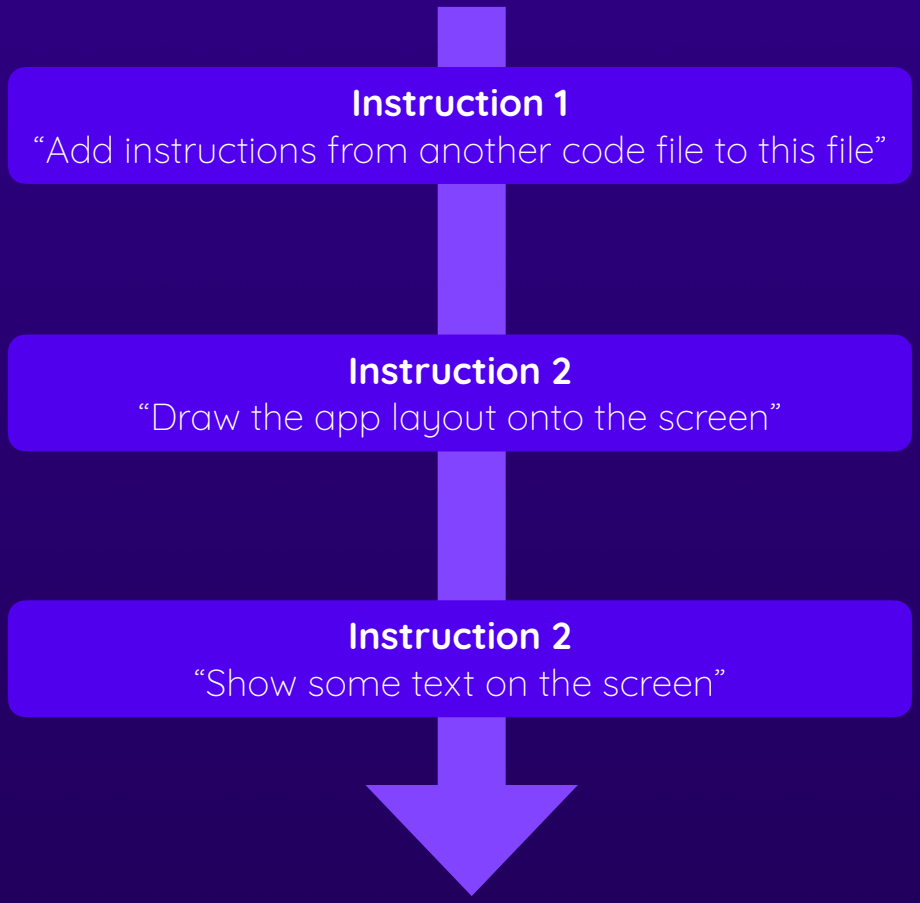
Basic Syntax & Features

- Explore Core Flutter & Dart Syntax
- Understanding & Writing Flutter and Dart Code
- Working with Flutter Widgets

Dart & Flutter Code Is Compiled



Programming?



Two Categories of “Words”

Keywords

Built into the programming language

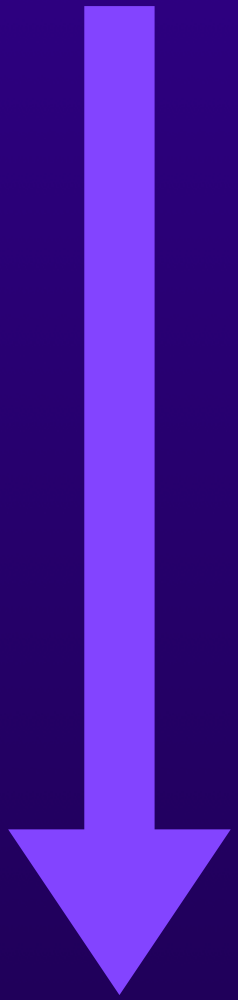
Have clear, specific meanings

Identifiers

Defined by developers

Used for identifying “things” in code

How Flutter Apps Become Active



1

`main()` function gets executed automatically

By Dart, when executing the compiled app on the target device

2

`runApp()` should be called inside of `main()`

`runApp()` "tells" Flutter what to display on the screen (i.e., which UI elements to display)

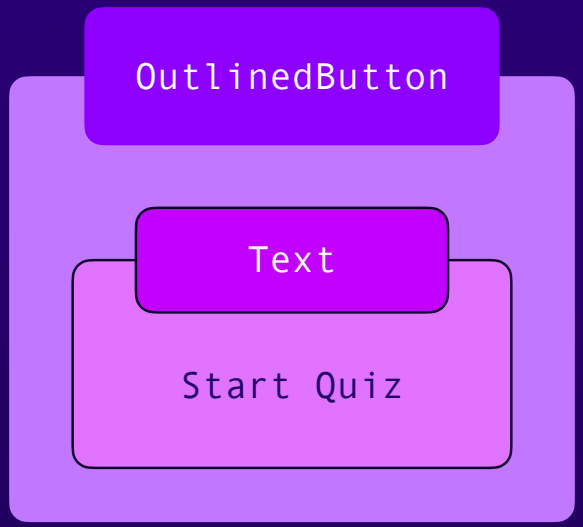
3

Pass the to-be-displayed "widget tree" to `runApp()`

A **"widget tree"** is a combination of (nested) Flutter widgets that build the overall user interface

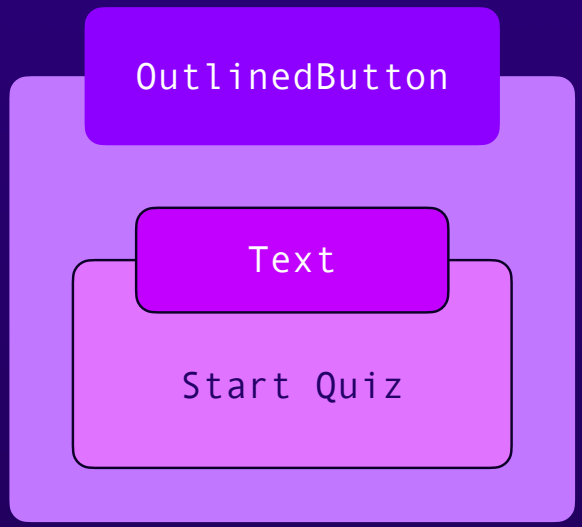
It's all about Widgets!

Flutter UIs are created by combining & nesting Widgets



It's all about Widgets!

Flutter UIs are created by combining & nesting Widgets



- **Flutter provides many built-in Widgets**
  e.g., Buttons, form inputs, layout widgets, ...
- **You can also build your own Widgets**
  Based on the built-in Widgets

Flutter UIs Are Built With Widgets

!

When using Flutter, you **build your user interface with code**

A combination of widgets

Widgets are nested into each other →

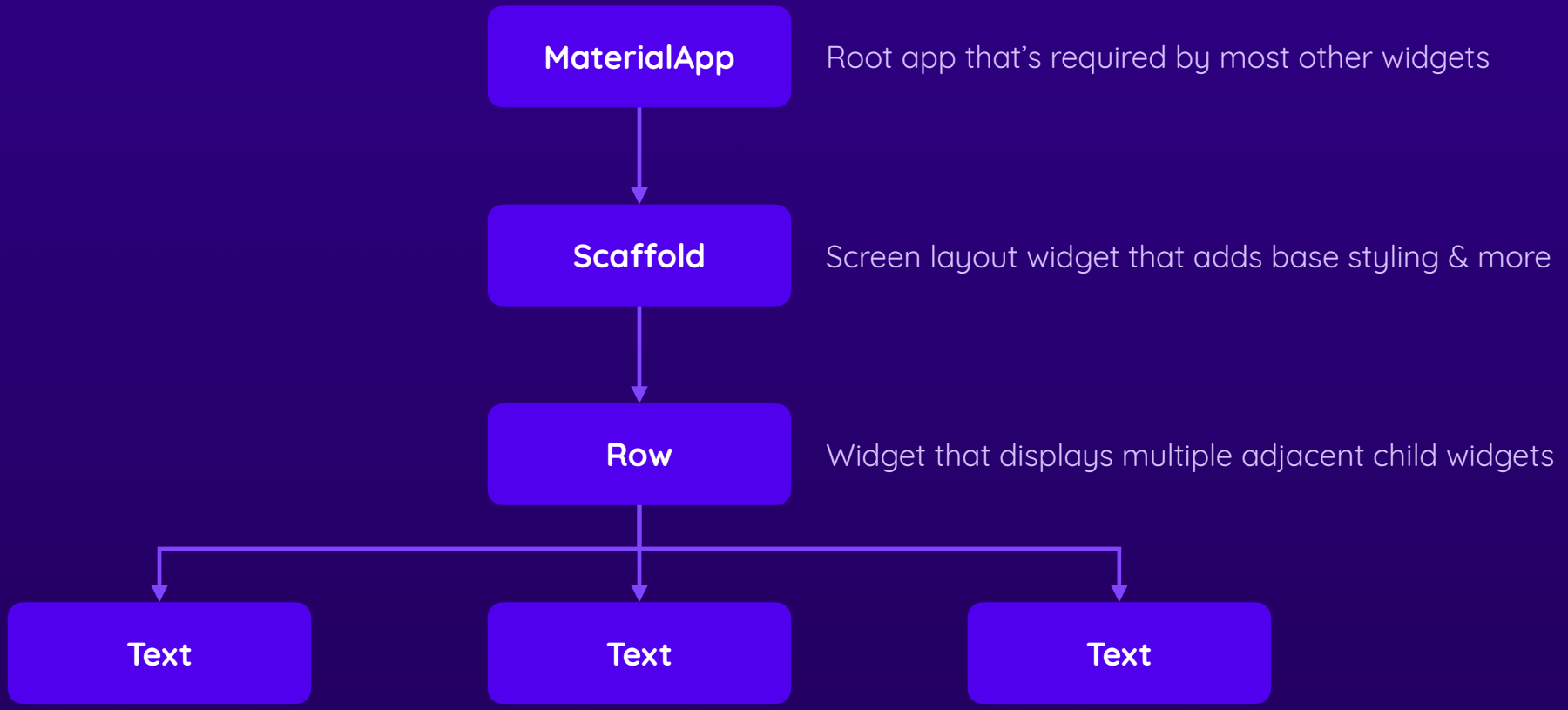
```
Center(
  child: Text('Hello World'),
);
```

→ **“Widget Tree”**

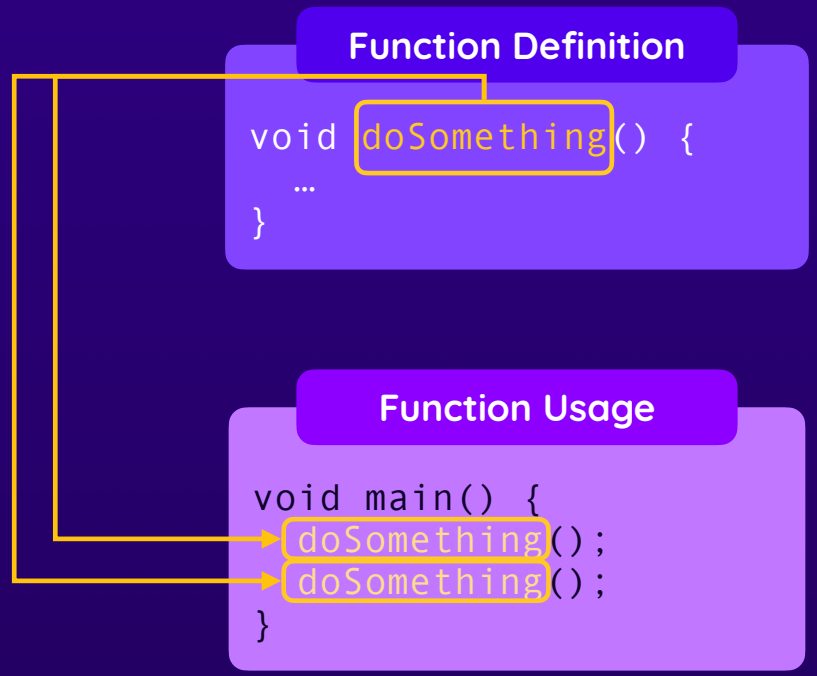
Built-in `Center` widget centers its content horizontally + vertically

Built-in `Text` widget displays some text on the screen

It's a Widget Tree!



Functions: “Code on Demand”



doSomething is the **function name** (i.e., it’s **identifier**)

Functions can be used (“called”) in your code **as often as needed**

Functions & Parameters

Functions may take **input values** — so-called "**Parameters**" or "**Arguments**"

No Parameters

```
void main() {…}
```

1 Parameter

```
void print(text) {…}
```

2 Parameters

```
void add(a, b) {…}
```

Multiple parameters are separated by commas

(these are just examples — functions can accept as many arguments / parameters as needed)

Named vs Positional Arguments

Functions may receive input values as "**named**" or "**positional**" **arguments**

Named Arguments

add(**num1**: 1, **num2**: 5)

Names of arguments must be specified to pass values to those arguments

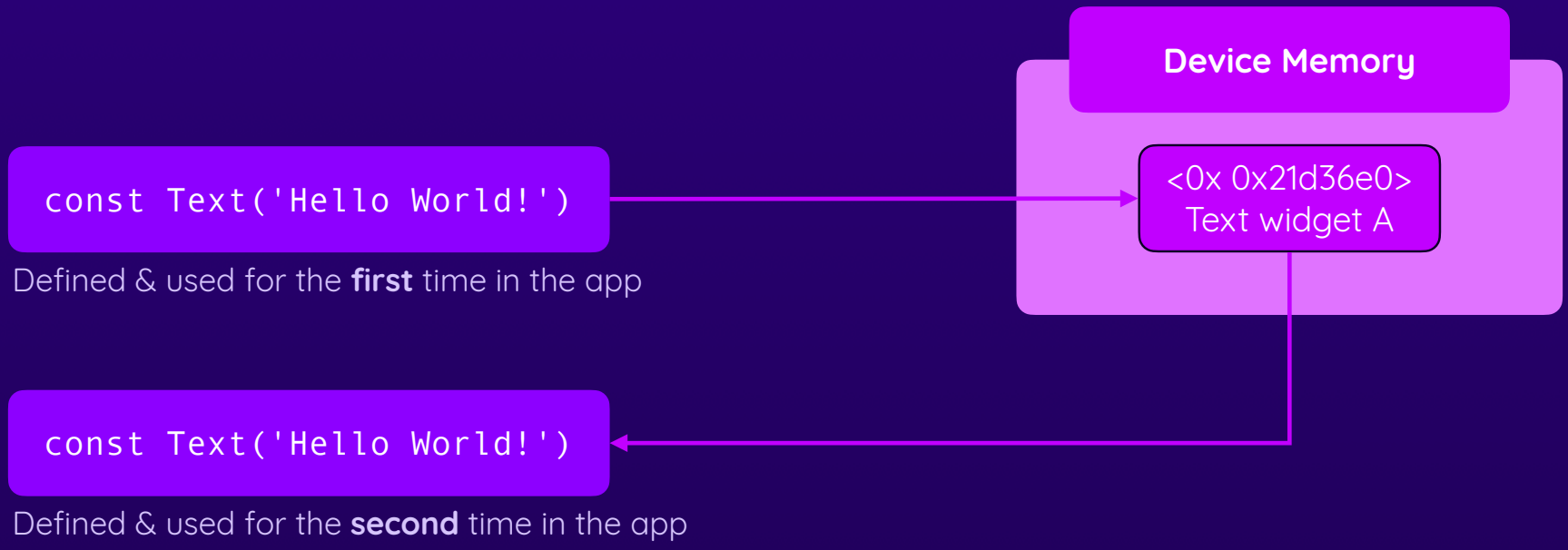
Positional Arguments

add(1, 5)

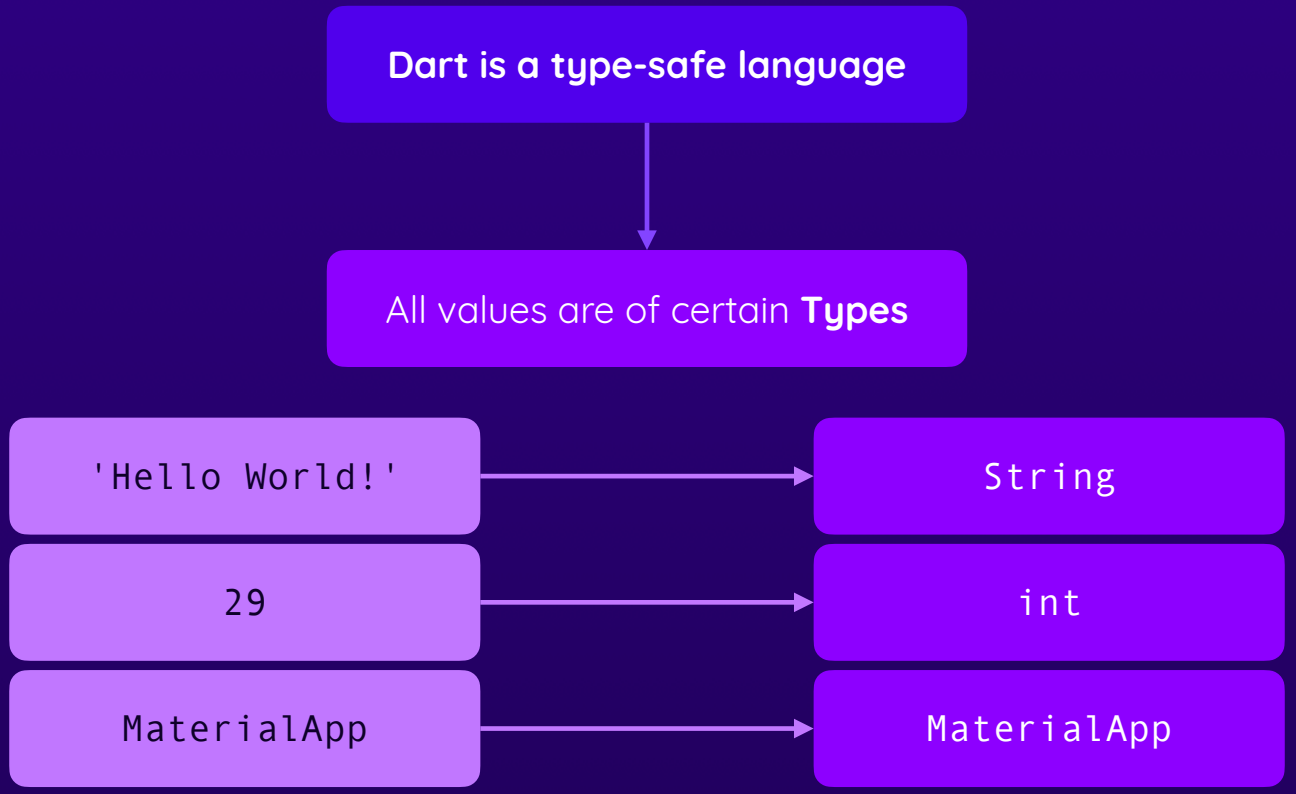
Argument values are mapped by position (first expected parameter receives first argument etc.)

Understanding “const”

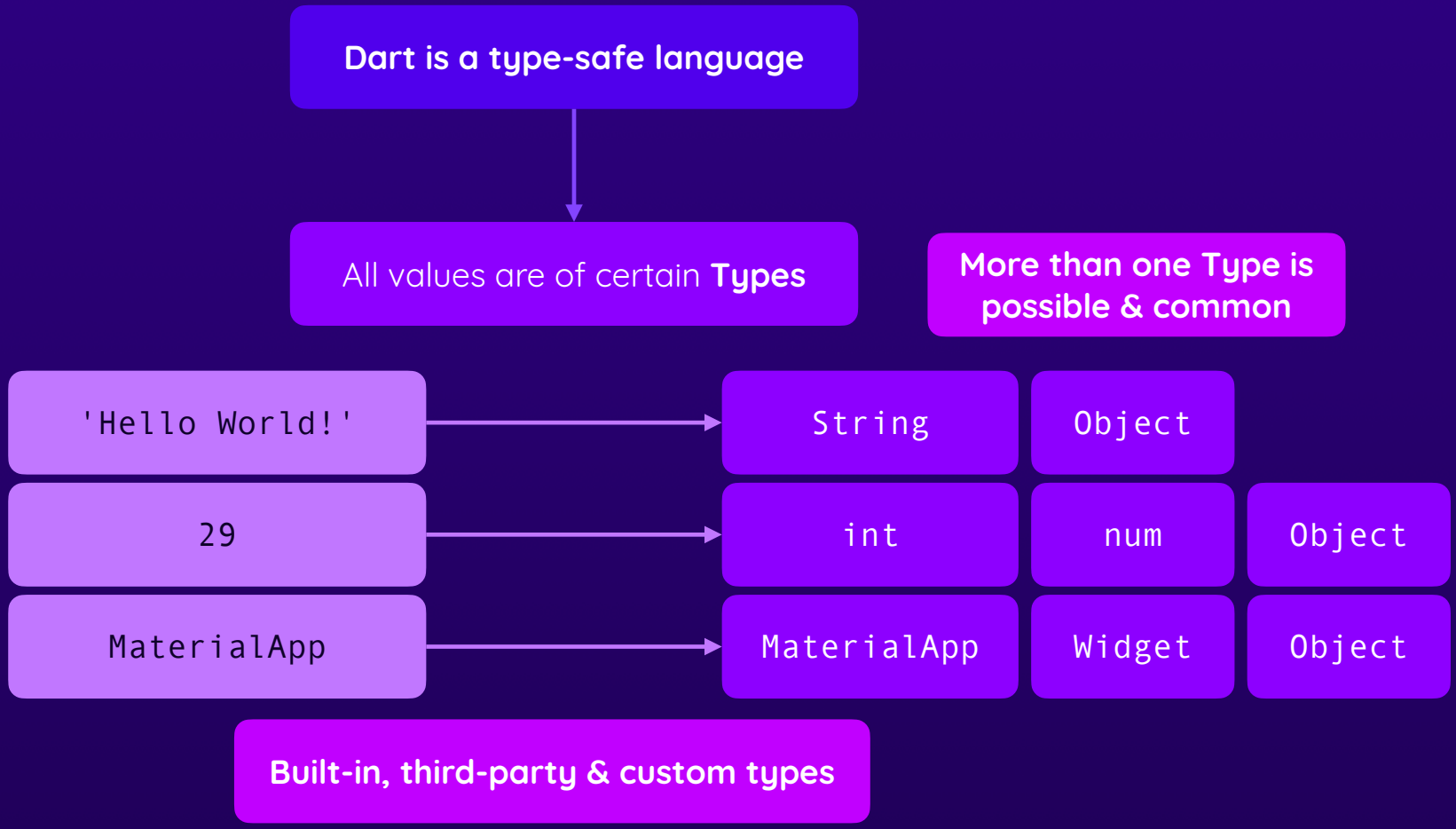
`const` helps Dart **optimize runtime performance**



Understanding Types



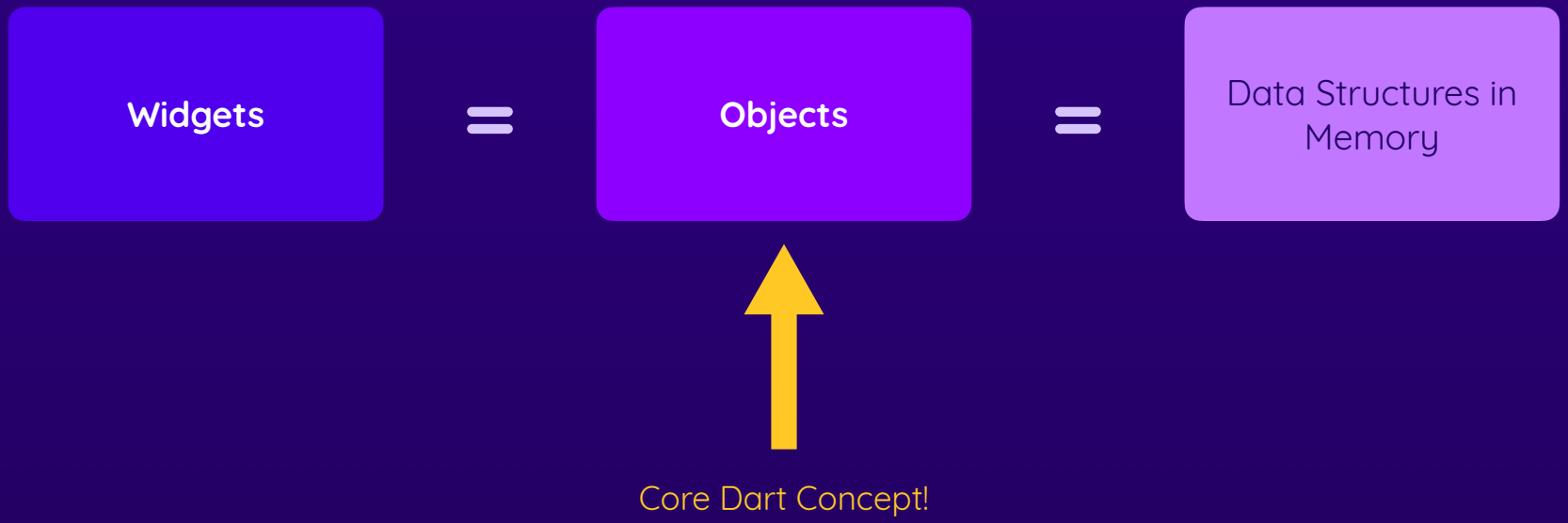
Understanding Types



Some Core Types

`int`	**Integer numbers**	Numbers **without** decimal places	`29, -15`
`double`	**Fractional numbers**	Numbers **with** decimal places	`3.91, -12.81`
`num`	**Integer or fractional numbers**	Numbers **with or without** decimal places	`15, 15.01, -2.91`
`String`	**Text**	Text, wrapped with single or double quotes	`'Hello World'`
`bool`	**Boolean values**	true or false	`true, false`
`Object`	**Any kind of object**	The base type of all values	`'Hi', 29, false`

Widgets Are Objects



Understanding Generic Types

Generic Types are “flexible types” that “work together” with other Types

E.g., list of hobbies

```
['Cooking', 'Sports', 'Reading']
```

List<string>

E.g., sensor data

```
[5.91, 3.87. 1.21]
```

List<double>

Understanding Classes

Dart is an object-oriented language

Every value is an **object**

Primitive Values

Text
`'Hello World'`

Numbers
`30, 12.31`

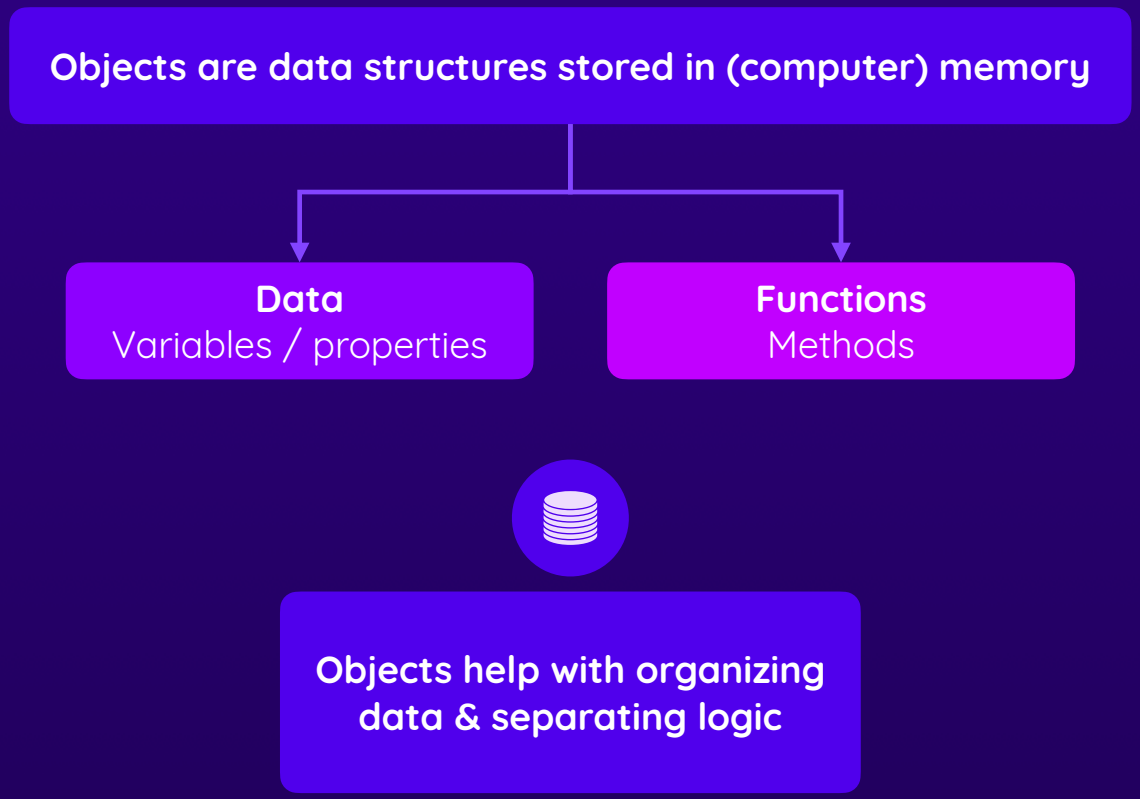
...

More Complex Values

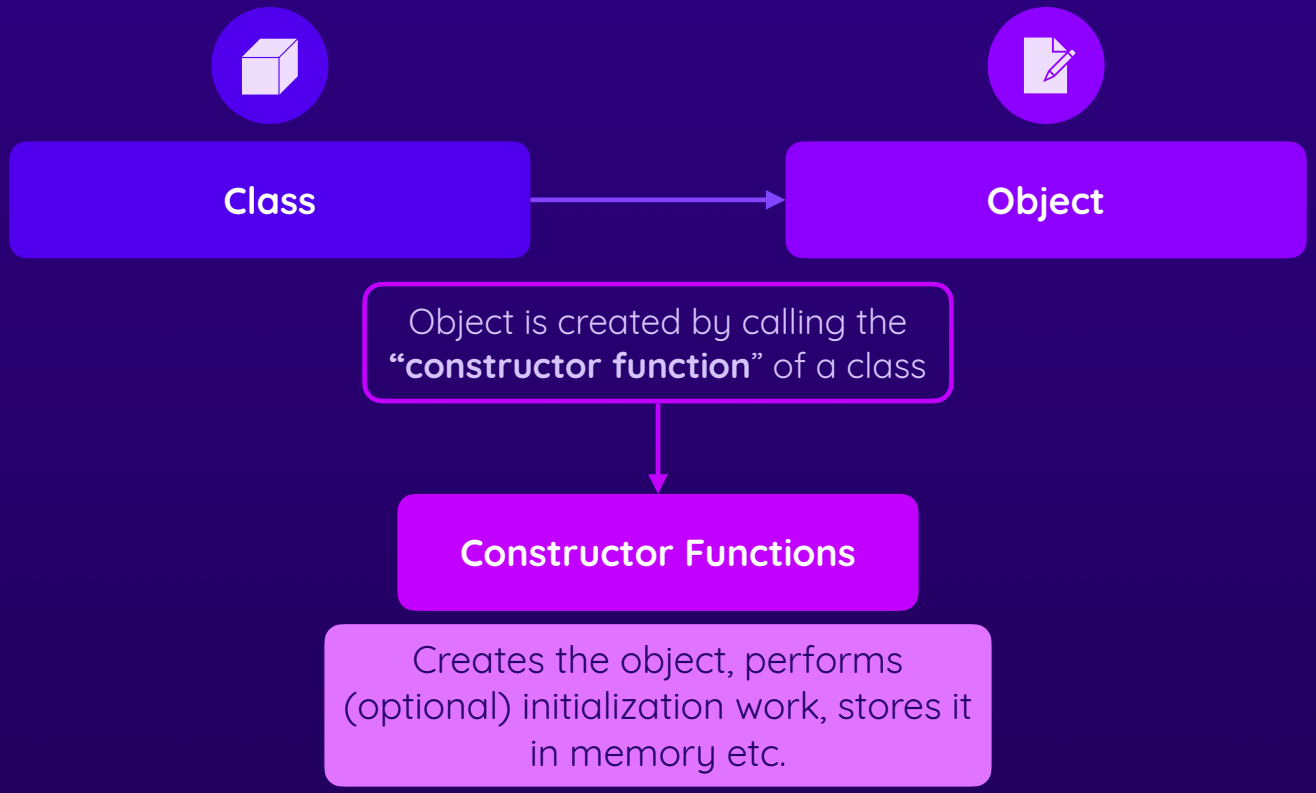
e.g., Widgets, Gradient Config Object

Created based on blueprints: **Classes**

Objects = Data Structures

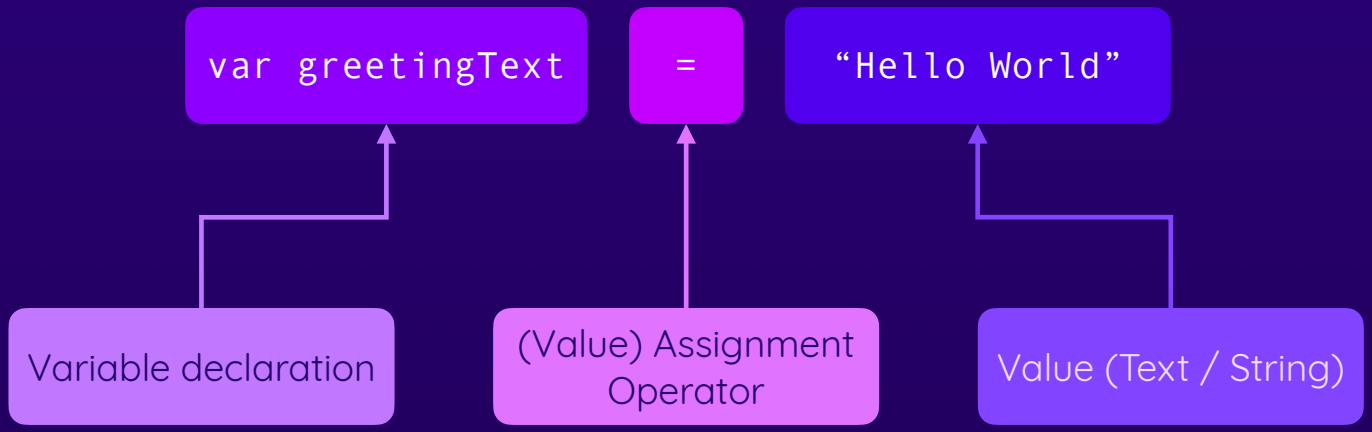


Objects Are Constructed From Classes



Understanding Variables

Variables are “Data Containers”



ACADEMIND

“final” vs “const” vs “var”

`var`

Creates a new variable that will be re-assigned at some point

Use the type (e.g., `String`) instead of var if the variable has no initial value

Otherwise, the type can be inferred by Dart

`final`

Create a new variable that will (and can) never be re-assigned

Prefer over `var` to avoid unintended re-assignments (e.g., by other developers)

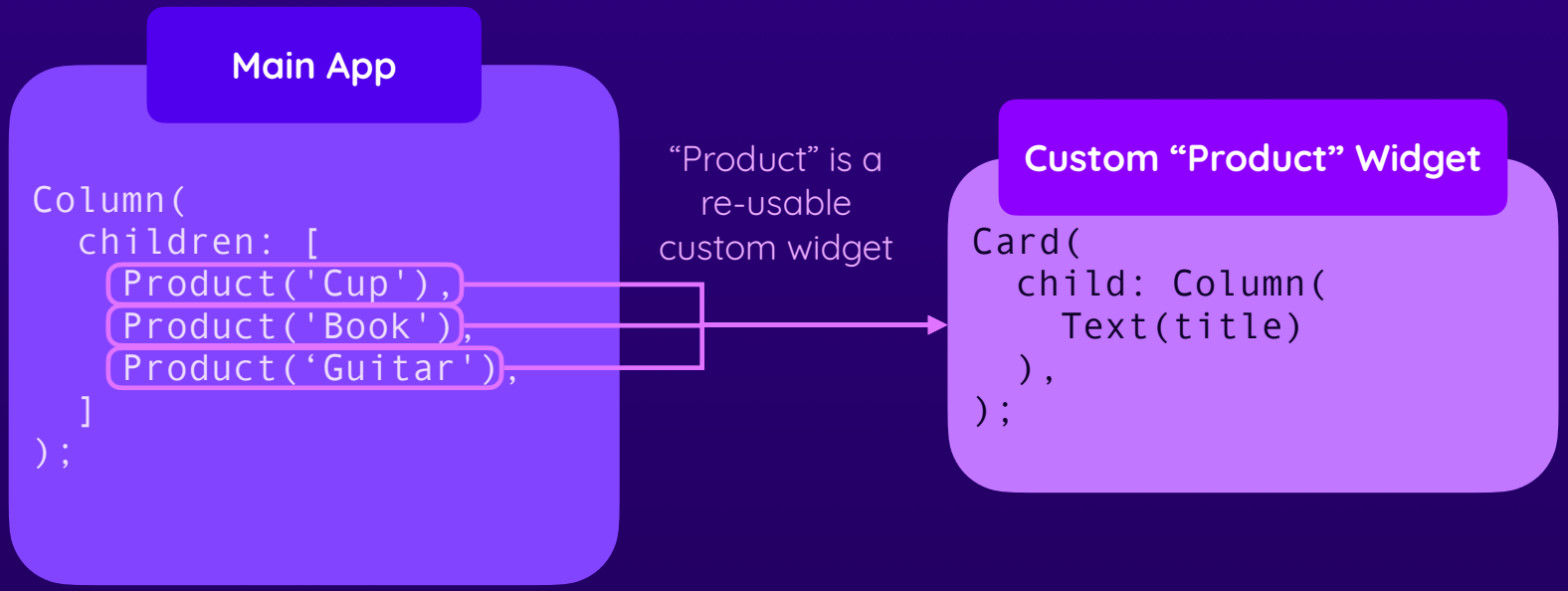
`const`

Create a new compile-time constant

Will (and can) never be re-assigned & value is “hardcoded” (fixed) at compile-time

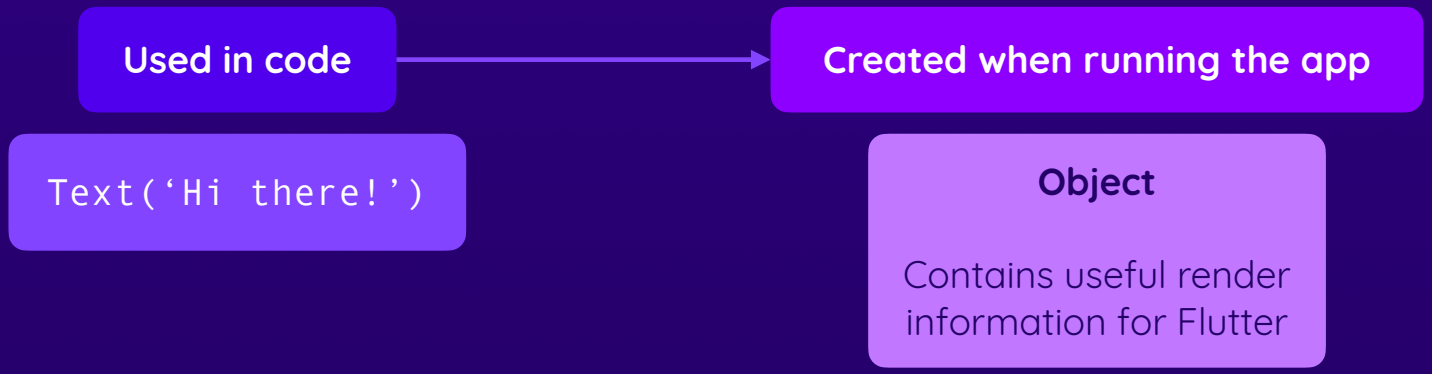
Can’t be used if some code must be executed in order to derive the value

Building Custom Widgets



(of course, Column, Card & many other built-in widgets are explained throughout the course)

Widgets Are Complex Objects



Column & Row

`Column()` & `Row()` can be used to place **multiple child widgets next to each other**

`Column()`

Main Axis: **Vertical** Axis

Cross Axis: Horizontal Axis

By default, occupies the **entire available height** but **only the width required** by its content (children)

`Row()`

Main Axis: **Horizontal** Axis

Cross Axis: Vertical Axis

By default, occupies the **entire available width** but **only the height required** by its content (children)

Functions As Values

In Dart, functions are also just values / objects!

Defining a Function

```
void start() { … }
```

Code inside the function **does not yet execute** — instead it's defined for **execution in the future**

Calling a Function

```
start();
```

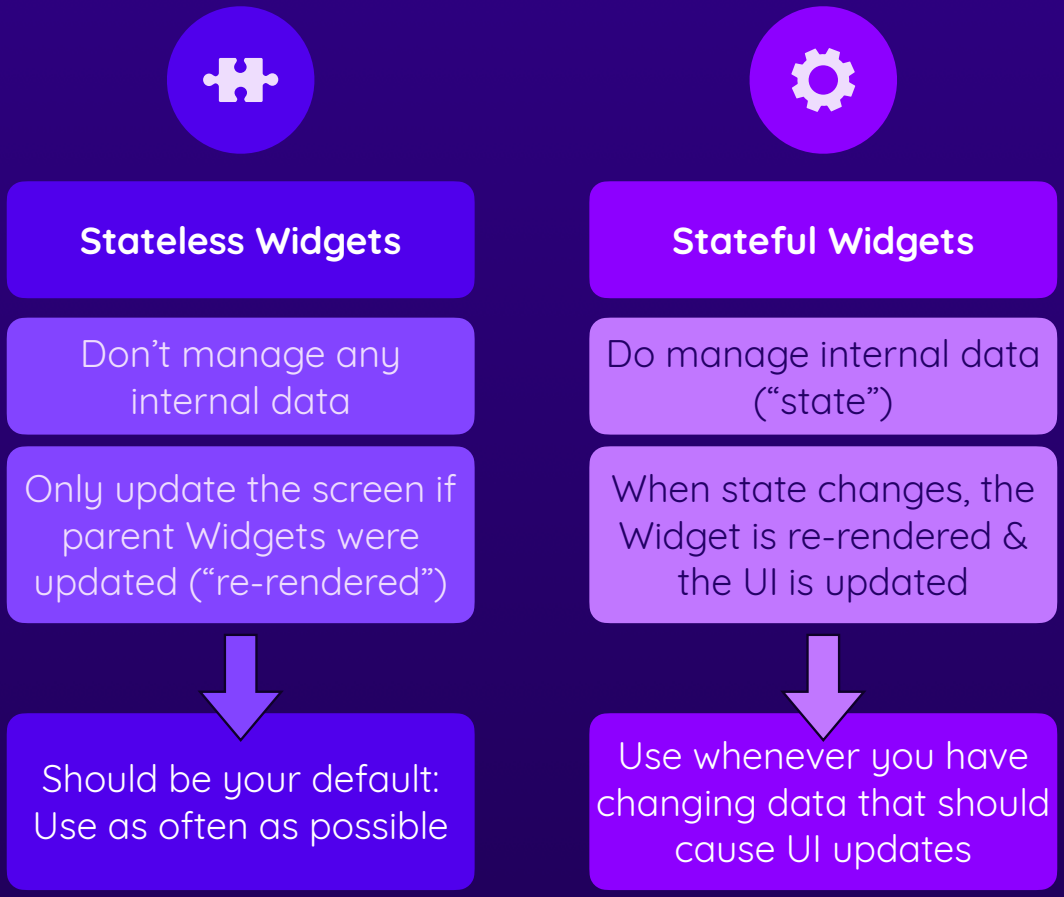
Code defined inside the function **is executed**

Using a Function as a Value

```
TextButton(onPressed: start)
```

Code inside the function is **not yet executed** — instead the function **may be called from inside the receiving object** (e.g., widget)

Stateless vs Stateful Widgets



Module Summary

Starting Flutter Apps

main.dart → main() → runApp()

Widgets, Widgets, Widgets

Flutter UIs are built by combining widgets

Widgets are nested into each other (“widget tree”)

Core Dart Features

Types, functions, variables, classes, objects, ...

Let the code editor (e.g., VS Code) help you!

Custom Widgets

StatelessWidget doesn’t change internally

StatefulWidget updates the UI upon state changes

Configuring Widgets

Many (built-in) widgets offer (named) configuration arguments

Typically, configuration objects are used

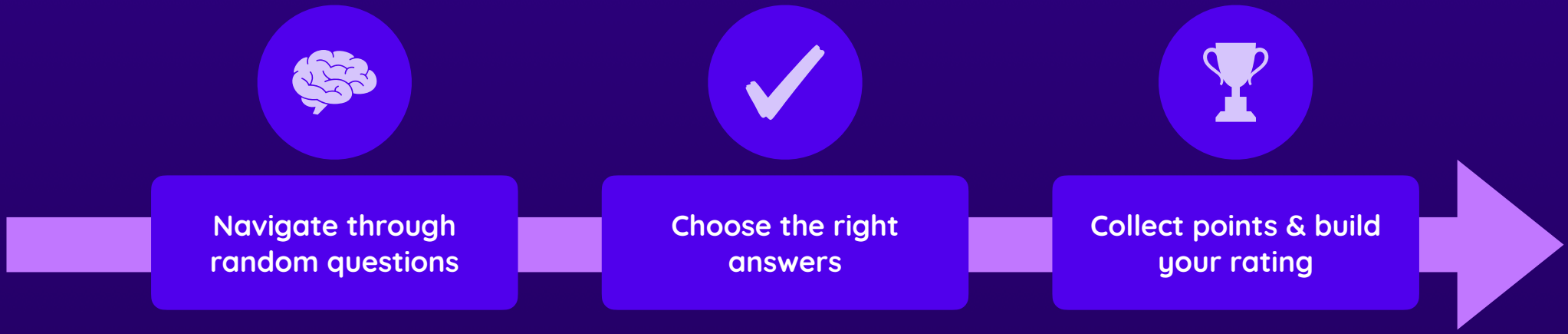
ACADEMIND

Advanced Fundamentals

Building Up on the Core Basics

- Explore & Use More Widgets
- Render Conditional & List Content
- Build More Complex User Interfaces

Quizzed — Our First App



ACADEMIND

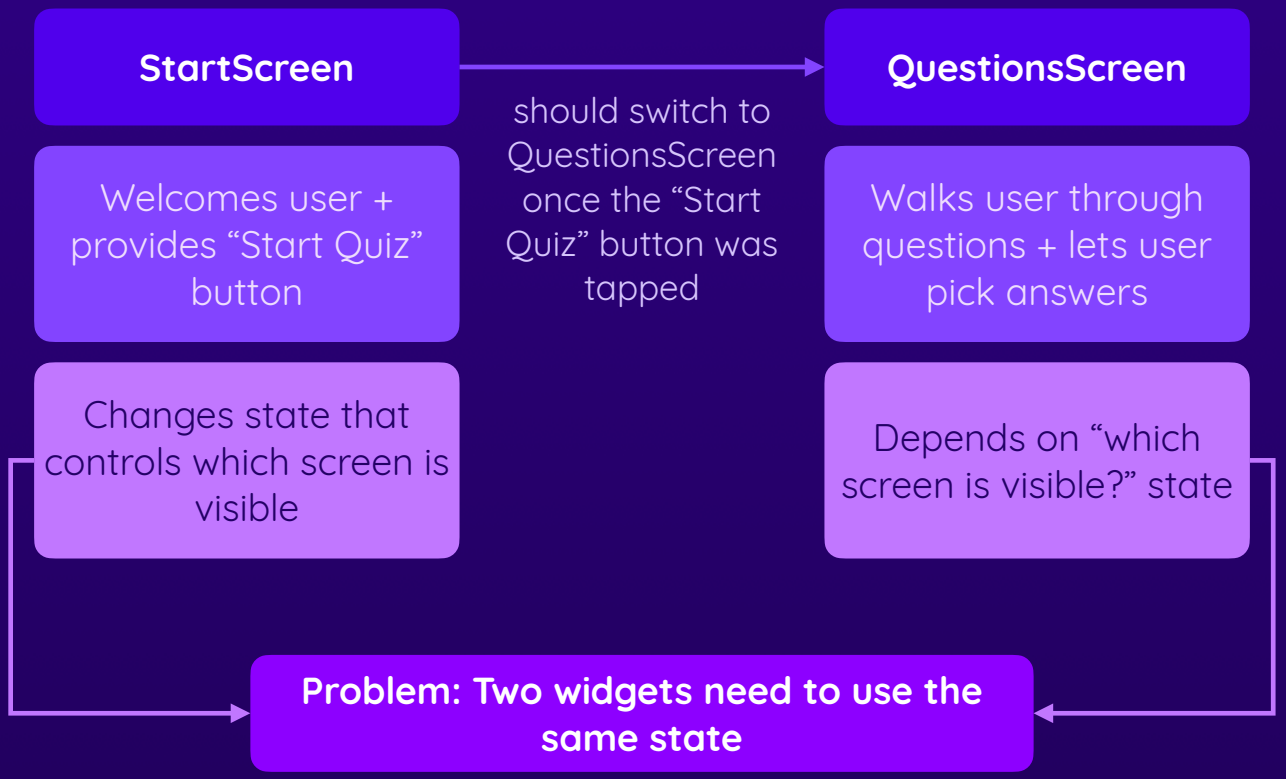
New Concepts!

?

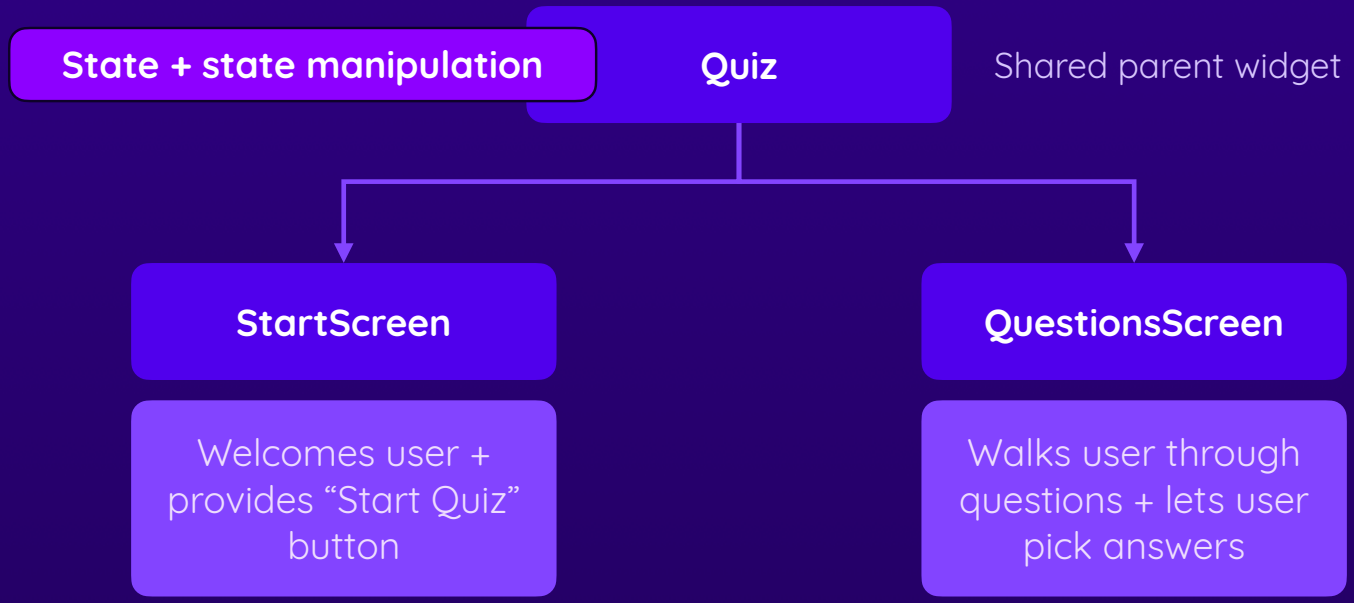
Render Content Conditionally

Lifting State Up

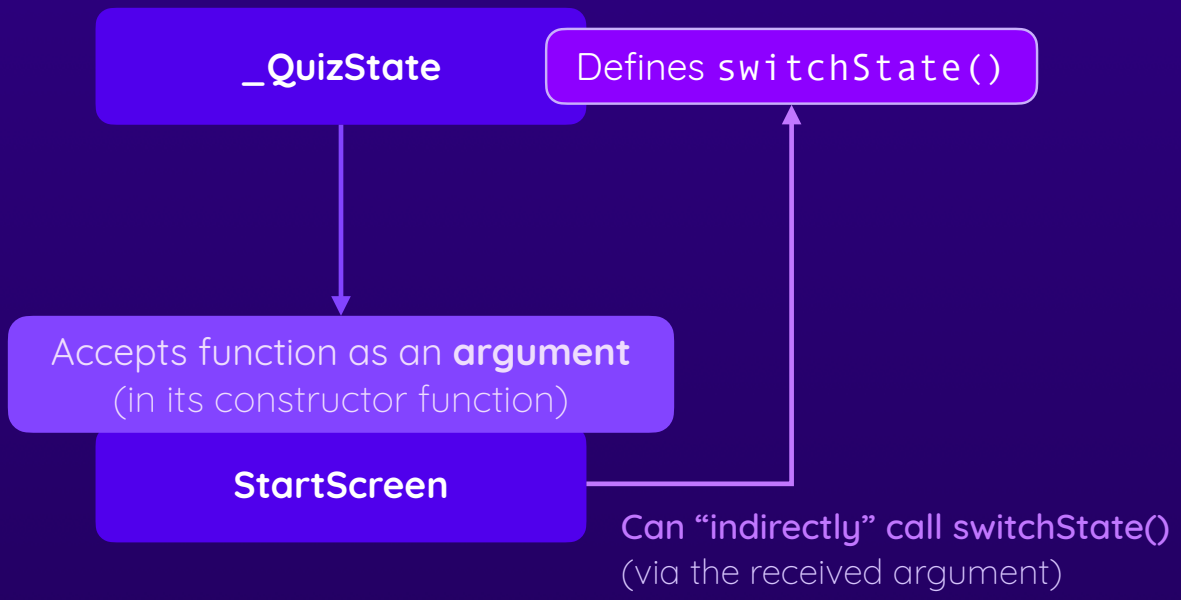
Lifting Up State



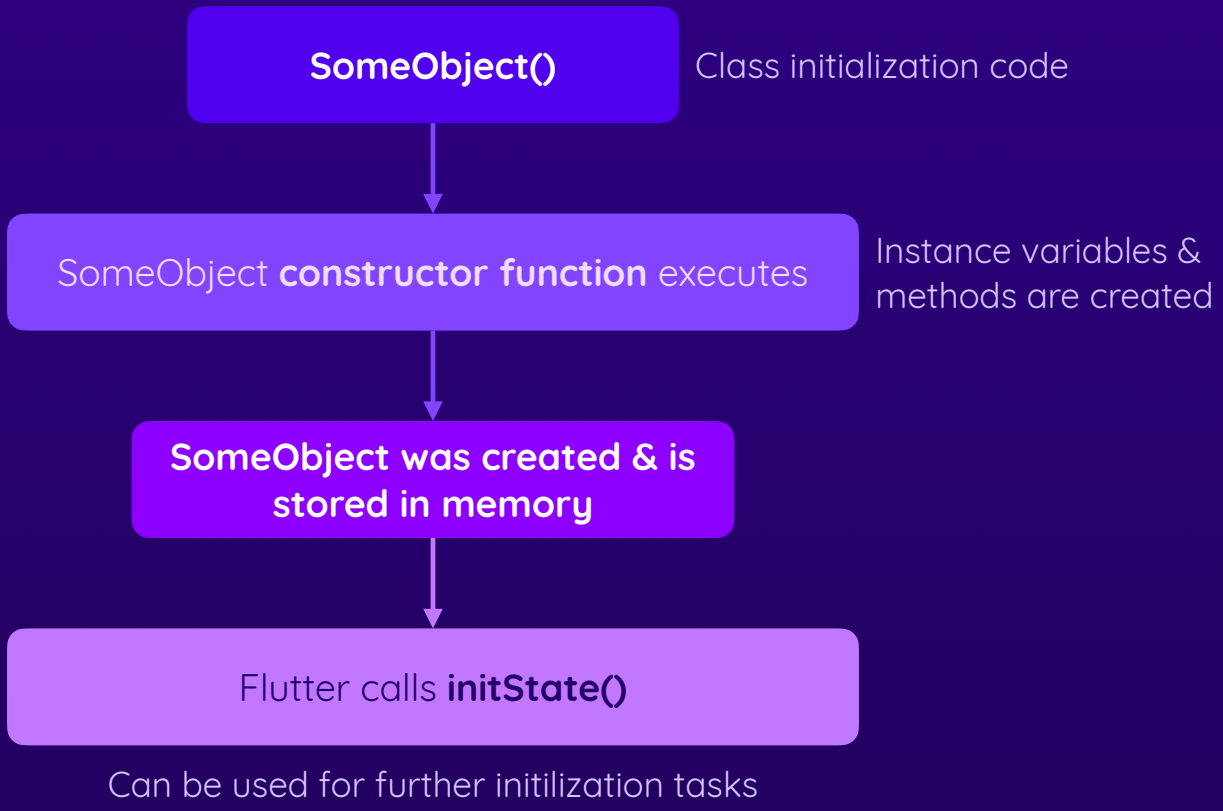
Lifting Up State



Using Functions As Argument Values



Using initState()



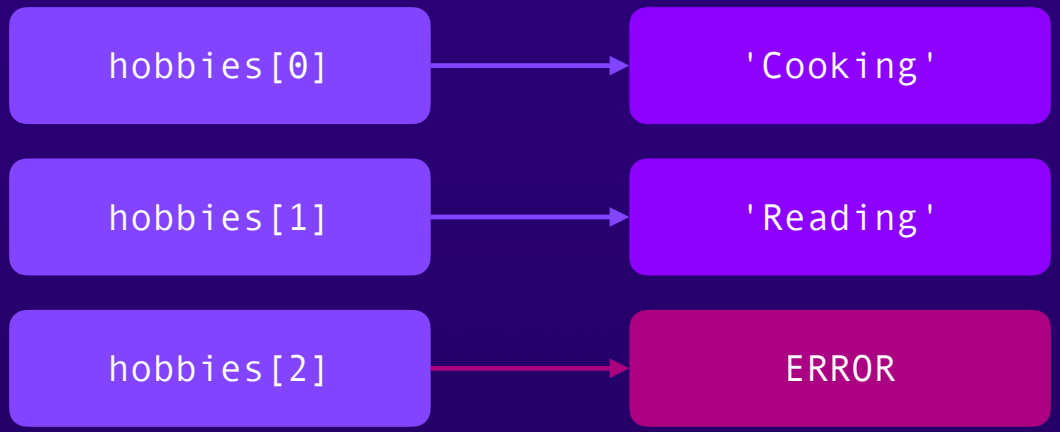
Comparison Operators

Yield `true` or `false`

`==`	Checks for equality	`'Hi' == 'Hi' // true`
`!=`	Checks for inequality	`'Hi' != 'Yay' // true`
`>`	Checks for "greater than"	`5 > 6 // false`
`<`	Checks for "smaller than"	`5 < 6 // true`
`>=`	Checks for "greater than or equal"	`5 >= 5 // true`
`<=`	Checks for "smaller than or equal"	`5 <= 4 // false`

Accessing List Values

```
const hobbies = ['Cooking', 'Reading'];
```



List Methods: map()

```
const numbers = [1, 2, 3];
```

```
const doubled = numbers.map((num) {
  return num * 2; // * operator multiplies values
});
```

```
print(doubled);
```

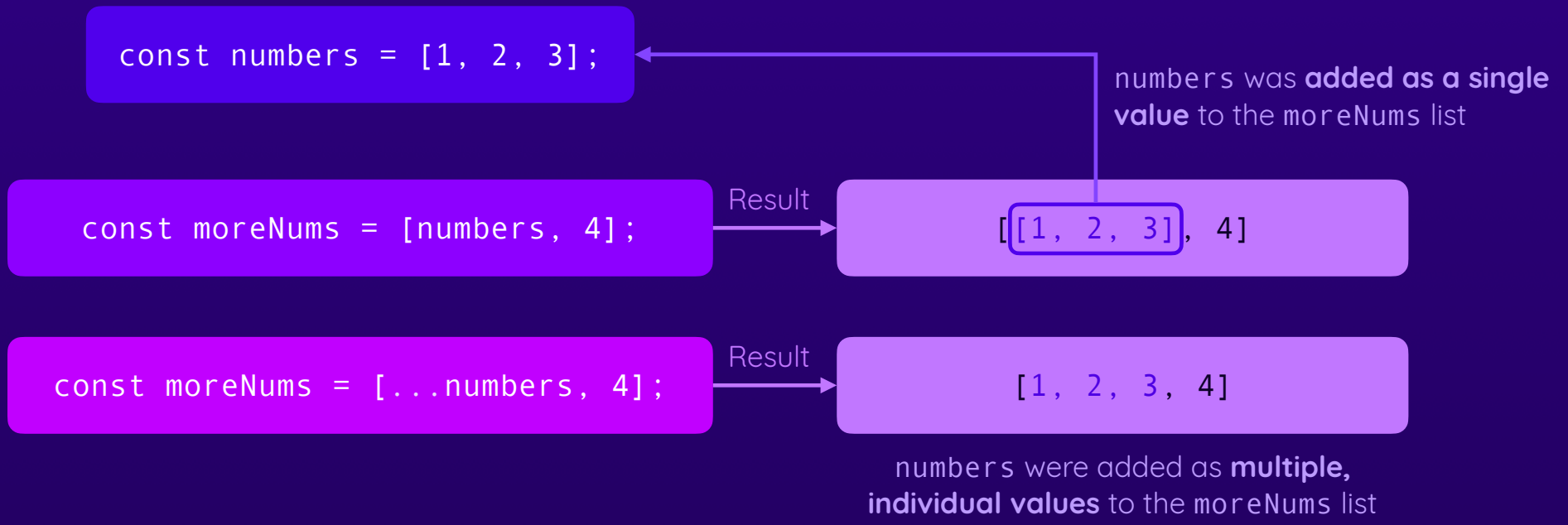
yields `[2, 4, 6]`

```
print(numbers);
```

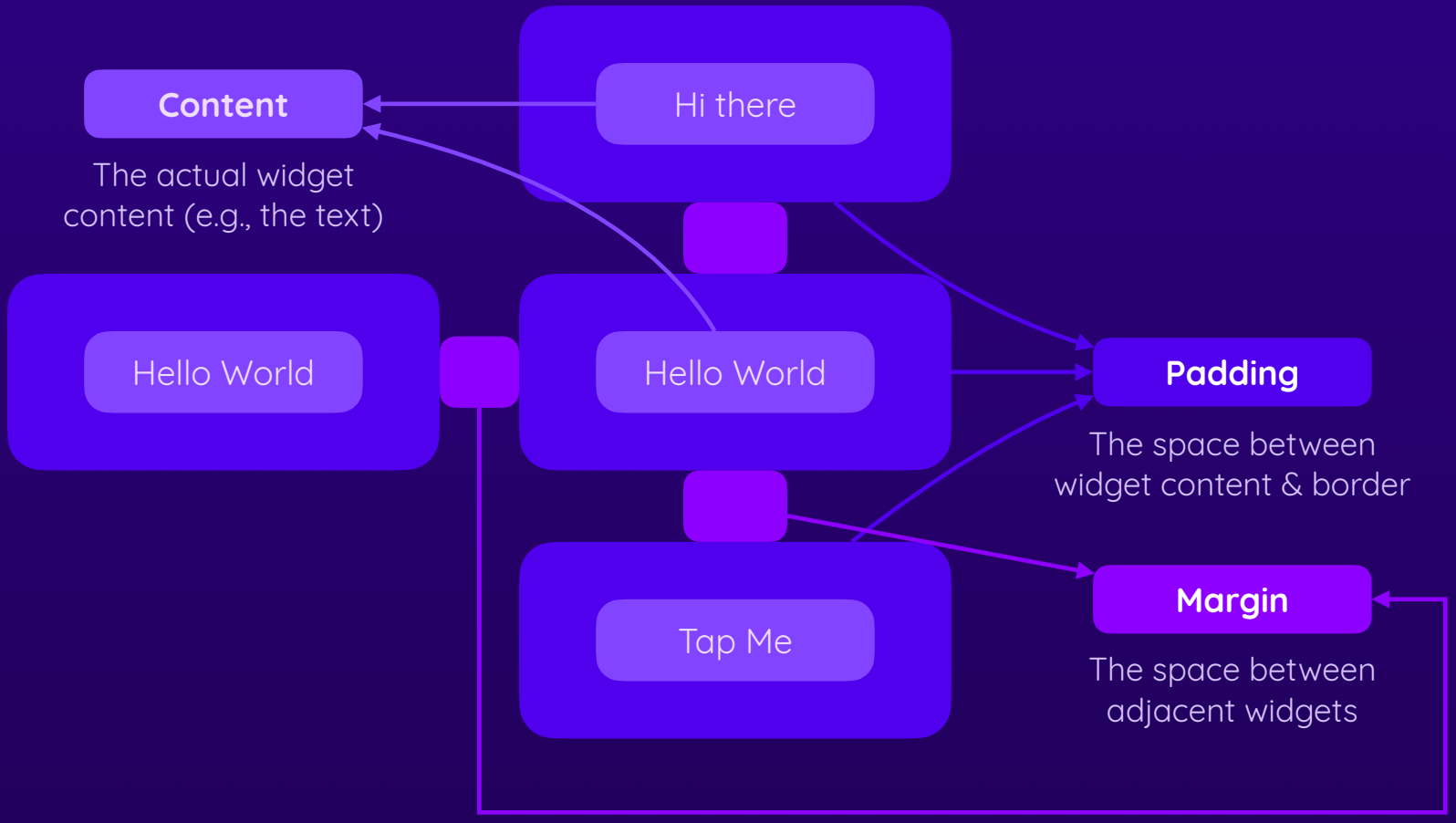
yields `[1, 2, 3]`

i.e., values & list
did not change!

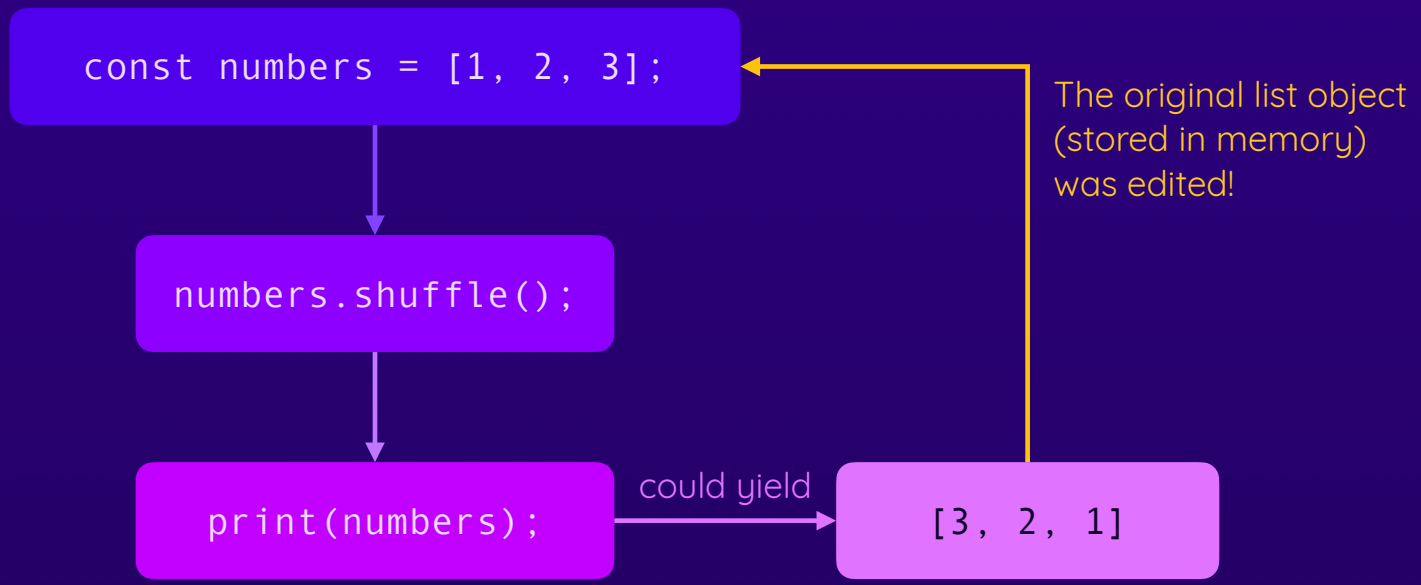
Spreading Values (...)



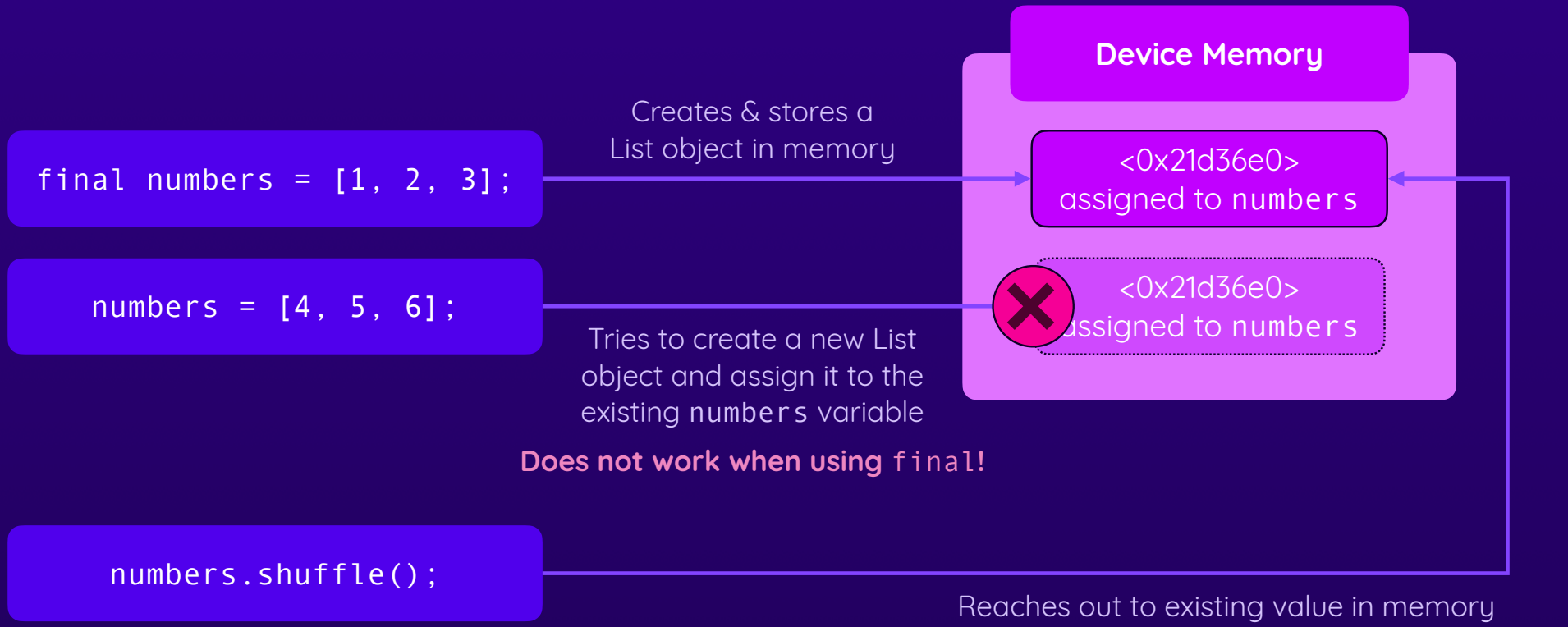
Margin & Padding



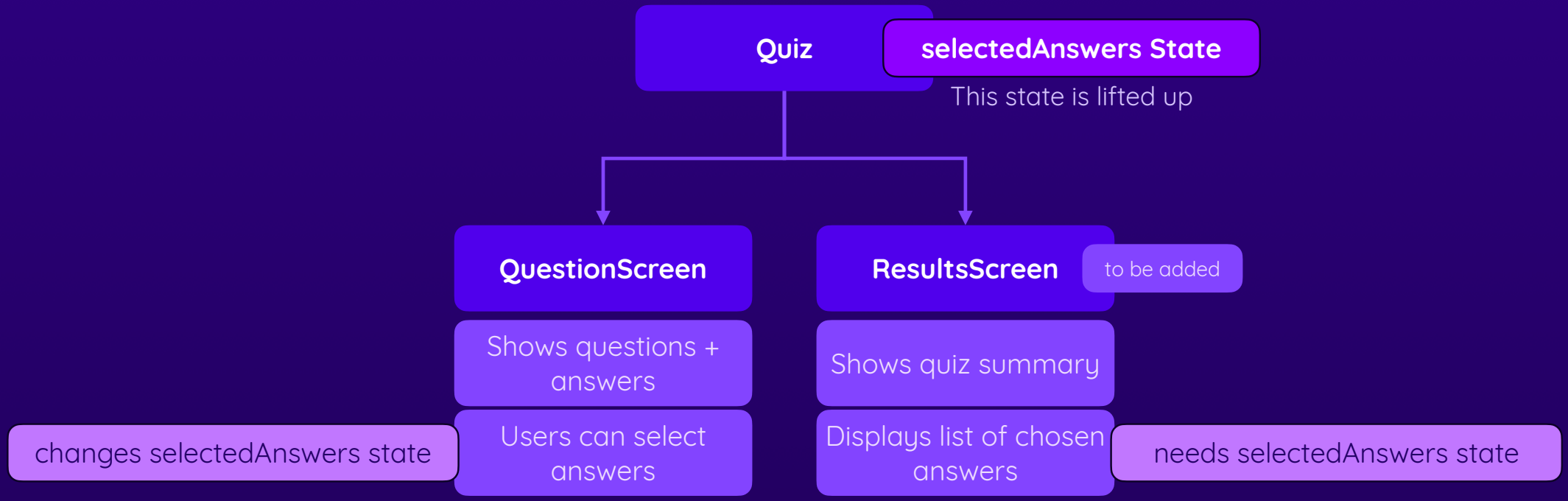
Shuffling Lists



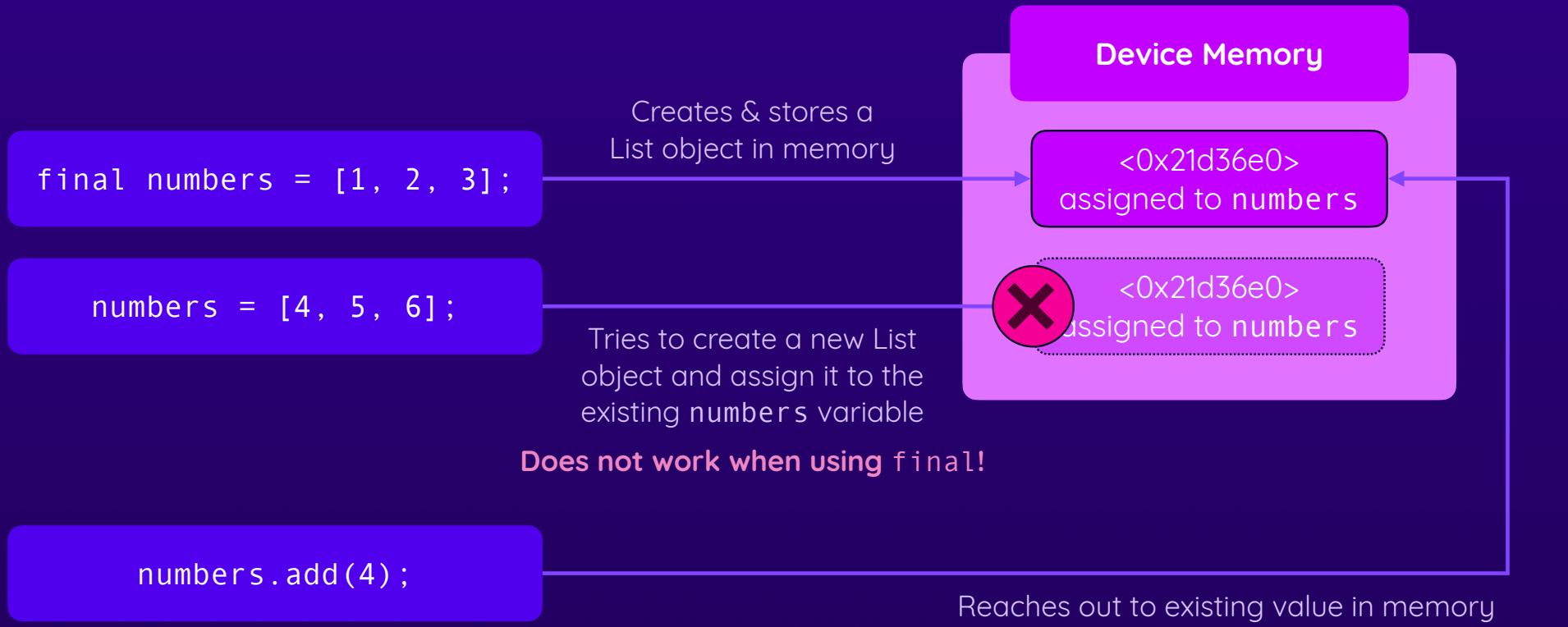
Mutating Values In Memory



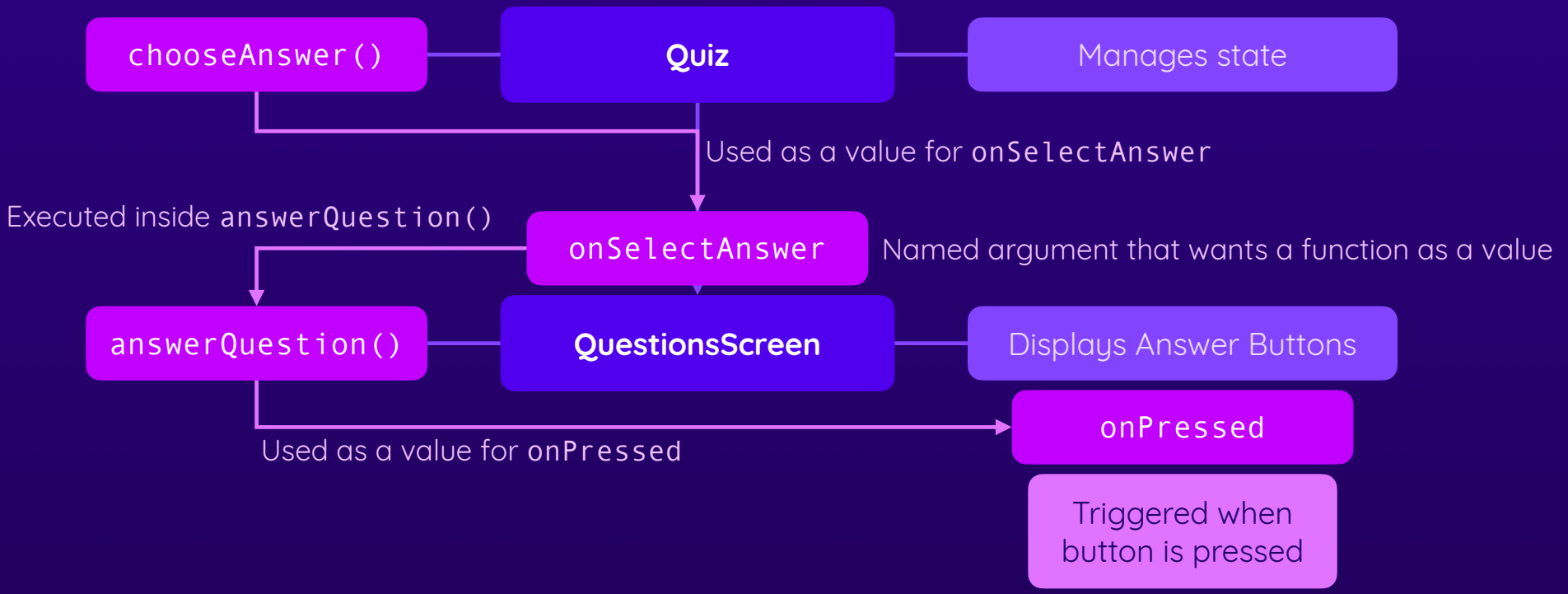
Lifting State Up - Again!



Mutating Values In Memory with add()



Passing Values Across Multiple Widgets



Understanding Maps

Maps are collections of key/value pairs

```
var user = {
  'user_name': 'Maximilian',
  'password': 'supersecret',
  'age': 33
};
```

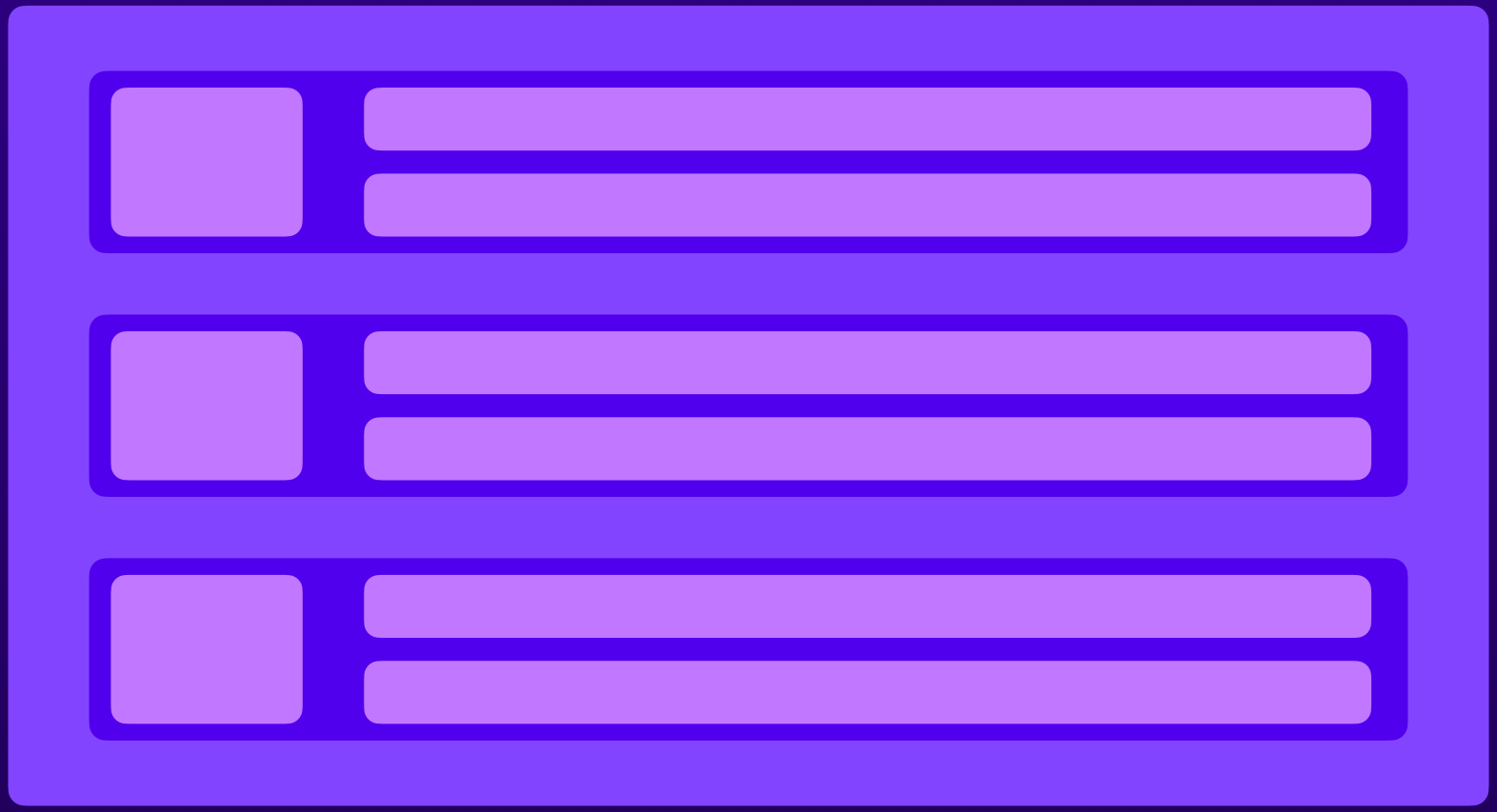
The keys and values can be any type of value
(the keys often are Strings but don't have to be)

Values can be accessed (read or modified) via []

```
user['age'] = 34;
```

Maps, like Lists, offer many built-in methods & properties
e.g., user.containsKey('age')

Combining Rows & Columns



ACADEMIND

Debugging Flutter Apps

Because Things Can Go Wrong

- Understanding Error Messages
- Using “Debug Mode”
- Use the Flutter DevTools

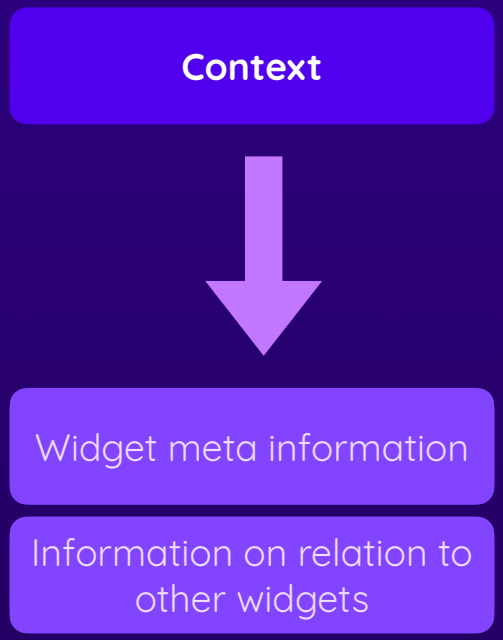
ACADEMIND

Interactivity & Theming

Making Apps More Interactive & Easier To Style

- Using Modals, Dialogs & More
- Basic User Input Handling
- Configuring & Using App Themes

Understanding Context



Understanding Theming

ACADEMIND

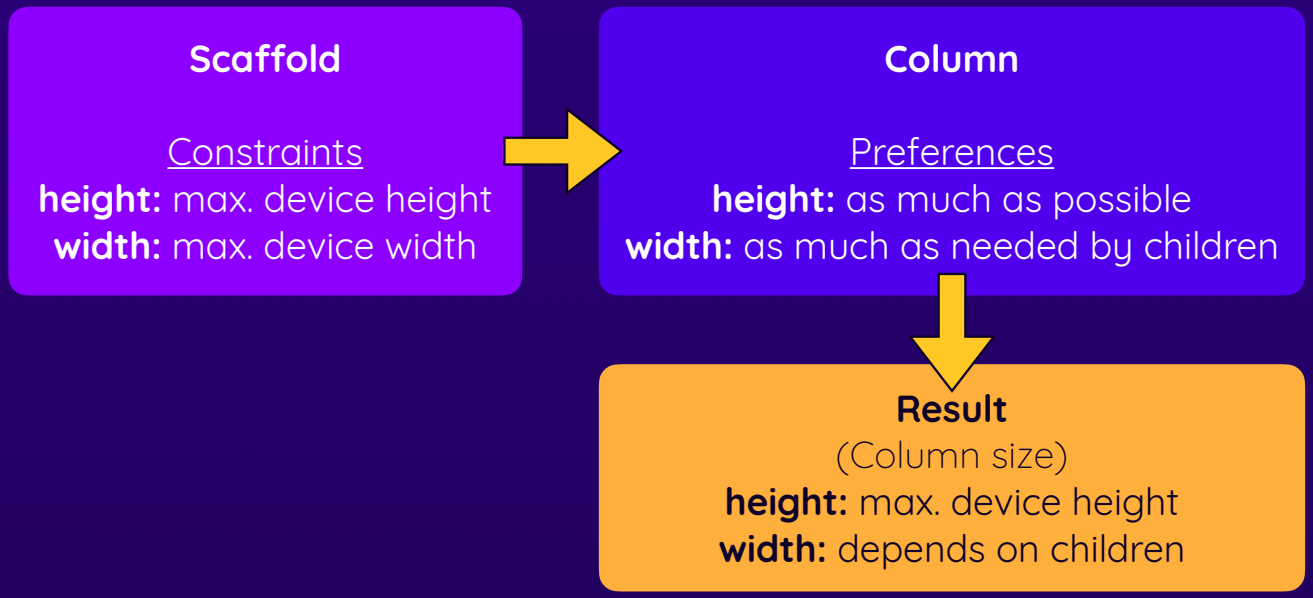
Responsive & Adaptive Apps

Adjusting Apps For Different Screen Sizes & Platforms

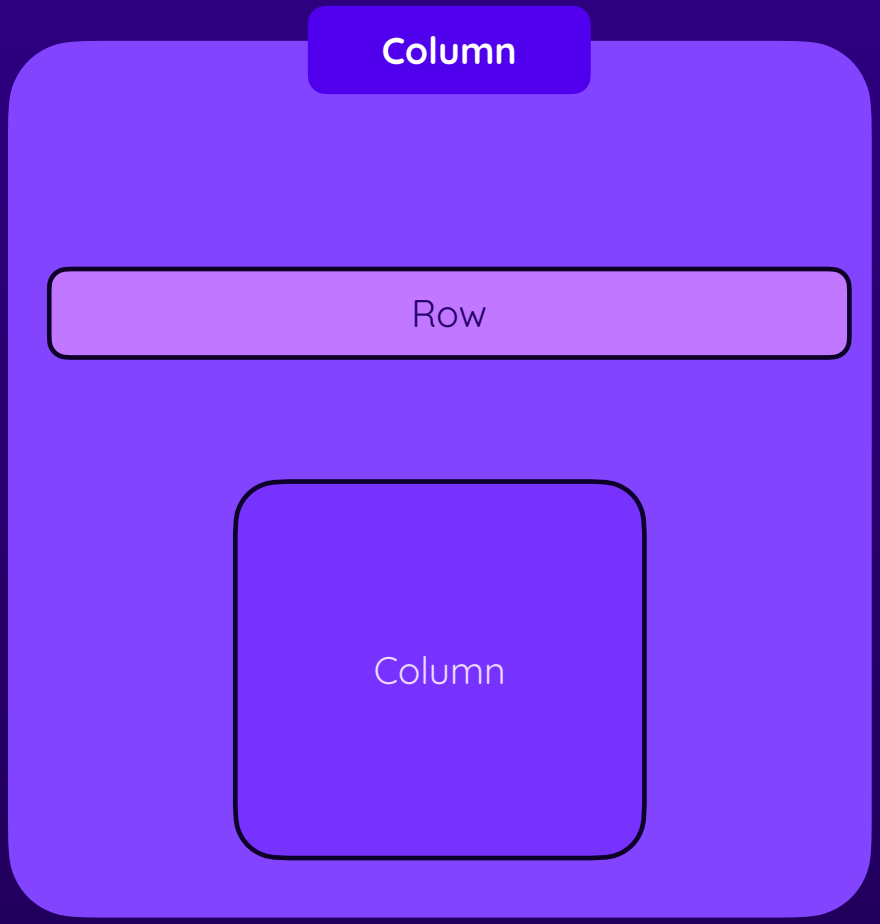
- Changing Layouts Based On Screen Sizes
- Detecting & Using Screen and Platform Information
- Building Adaptive Widgets

Understanding Widget Size Constraints

Widgets get sized based on their **size preferences** & parent widget **size constraints**



Understanding Widget Size Constraints



Column constraints

Width: 0 → depends on children
Height: 0 → INFINITY

Row constraints

Width: 0 → INFINITY
Height: 0 → depends on children

Column constraints

Width: 0 → depends on children
Height: 0 → INFINITY

⚠️ Problem: No height constraint from parent

ACADEMIND

Building Adaptive, Cross-Platform Apps

You can use the same widgets & styling on Android & iOS!

But you can also adjust some widgets or styles

ACADEMIND

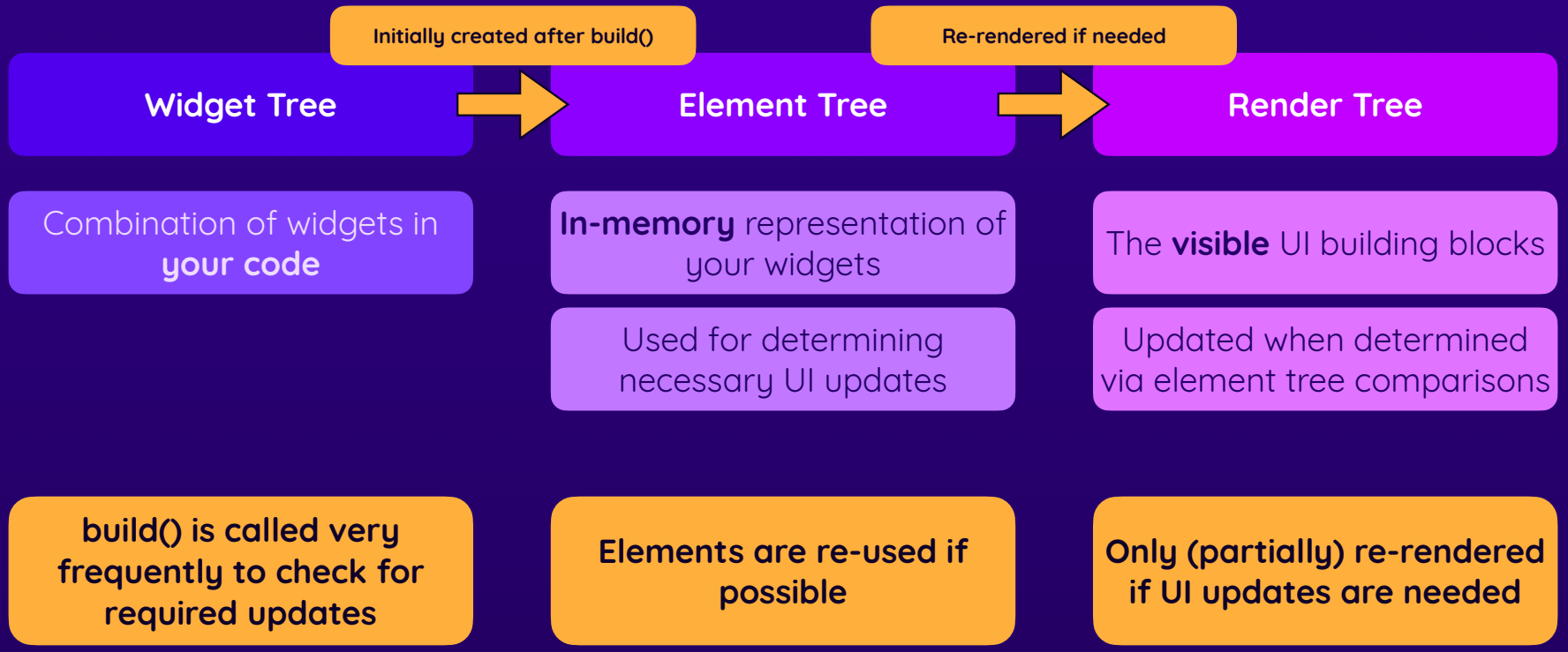
Flutter Internals

A Look Behind The Scenes

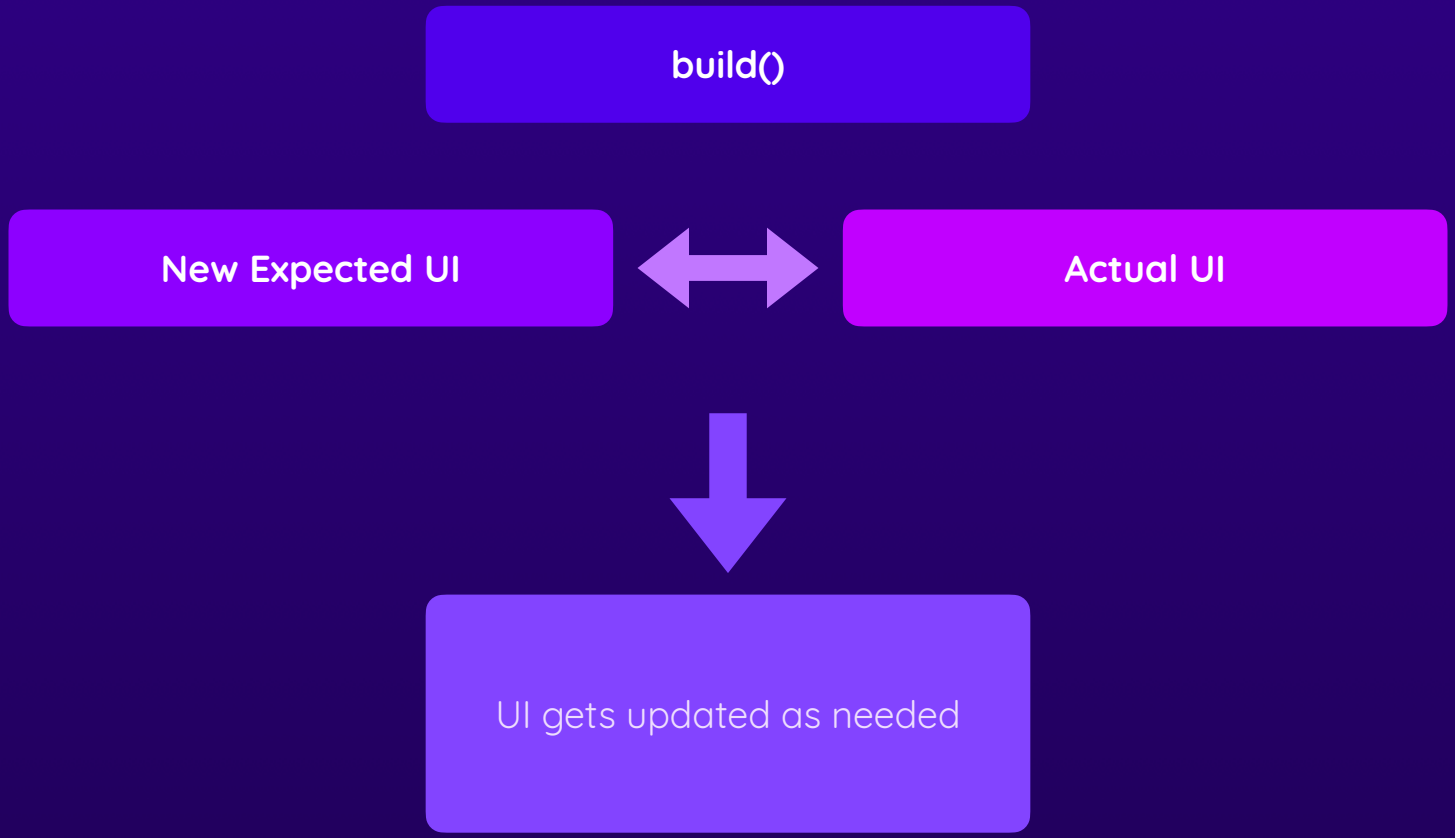
- Widget, Element & Render Trees
- How Flutter Updates UIs
- Understanding Keys

ACADEMIND

Three Trees



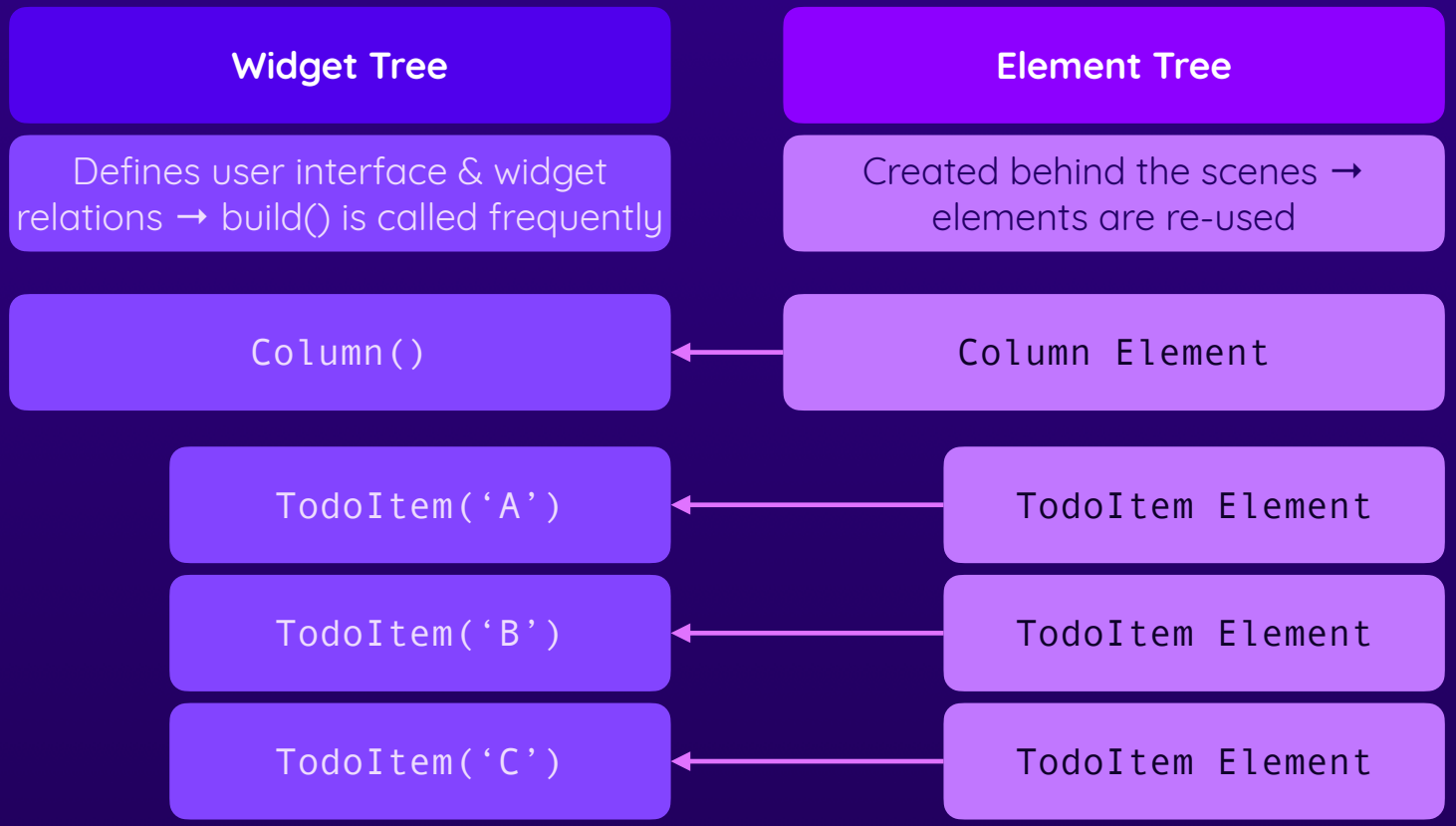
How The UI Gets Updated



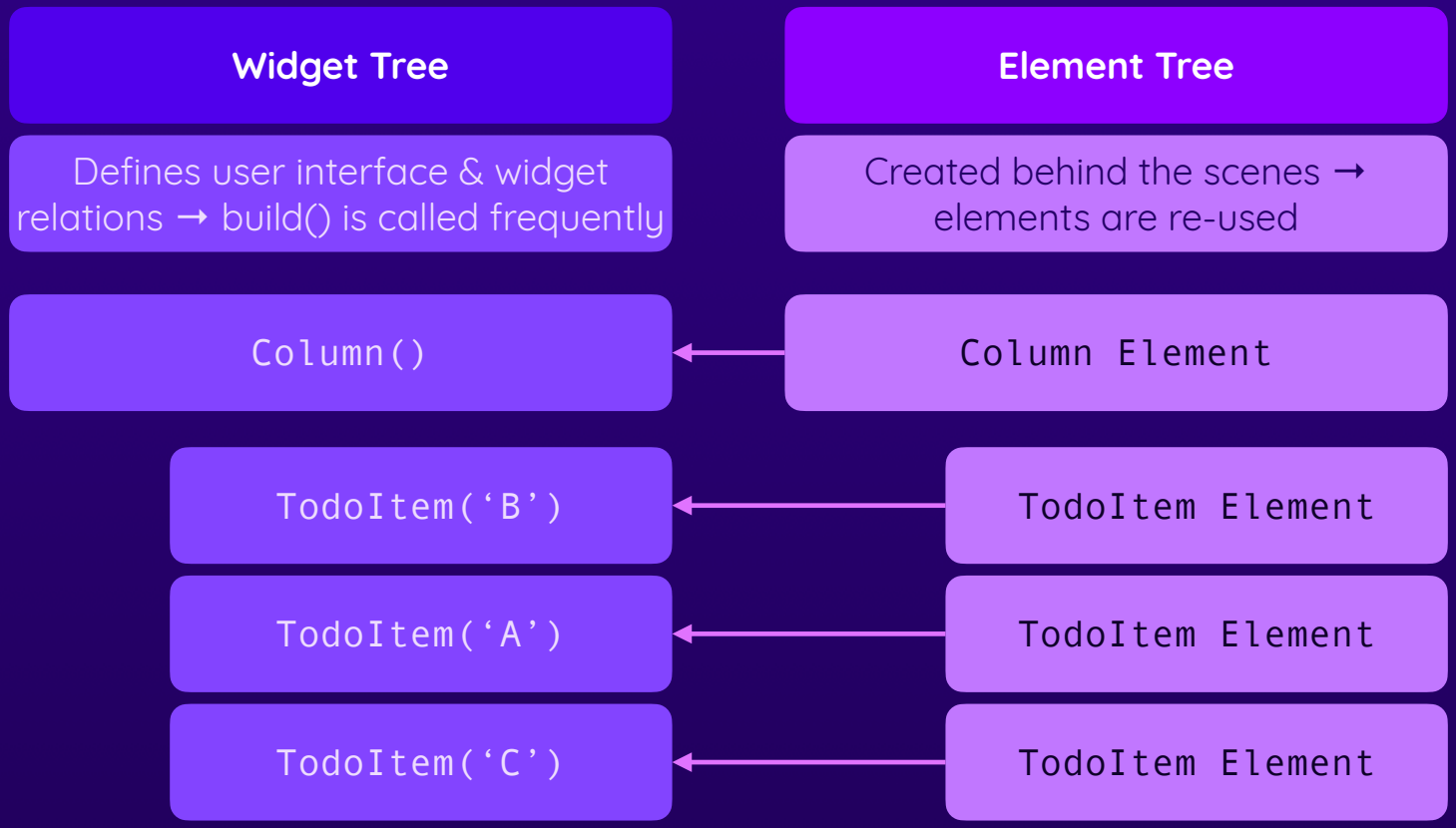
ACADEMIND

Keys?

Widget & Element Trees



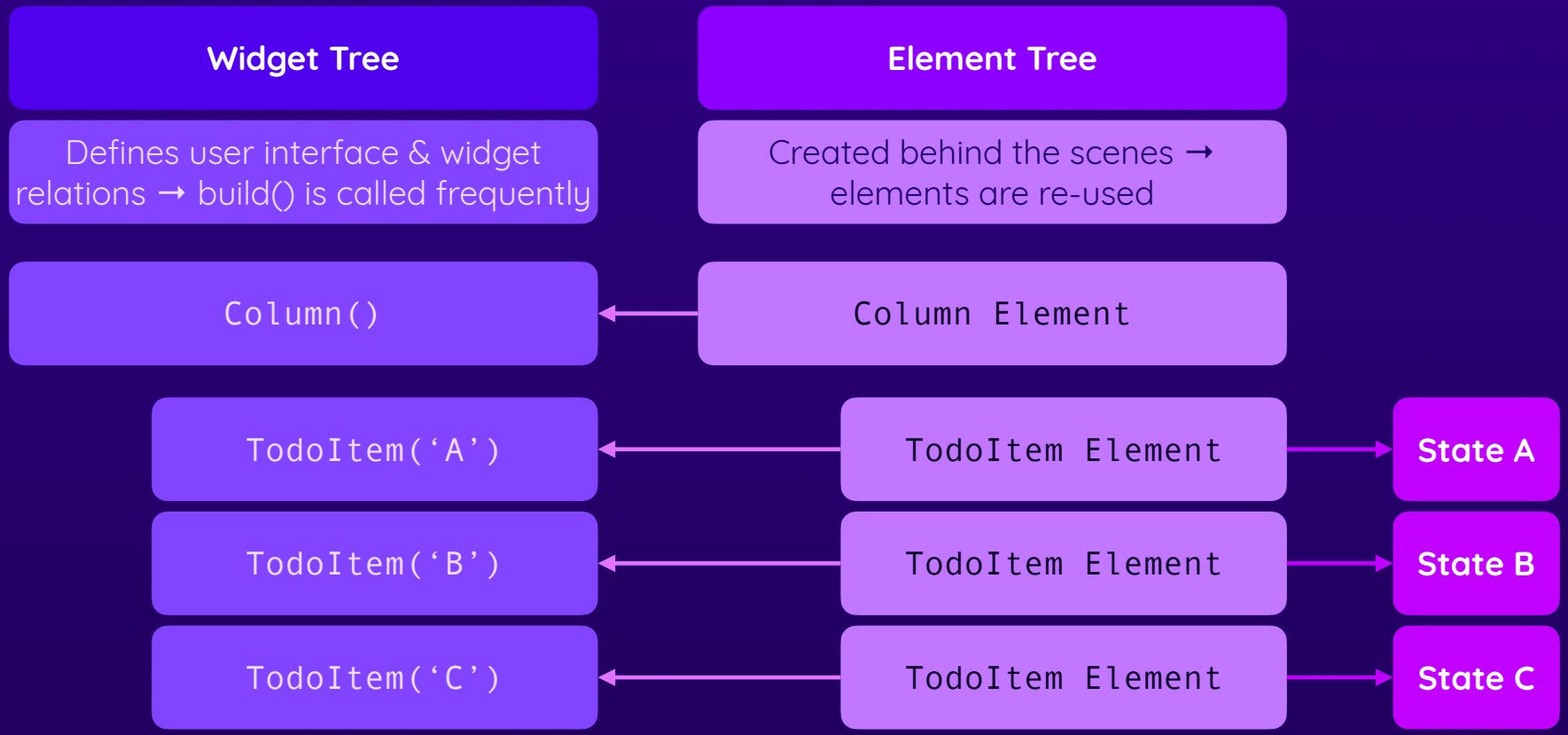
Widget & Element Trees



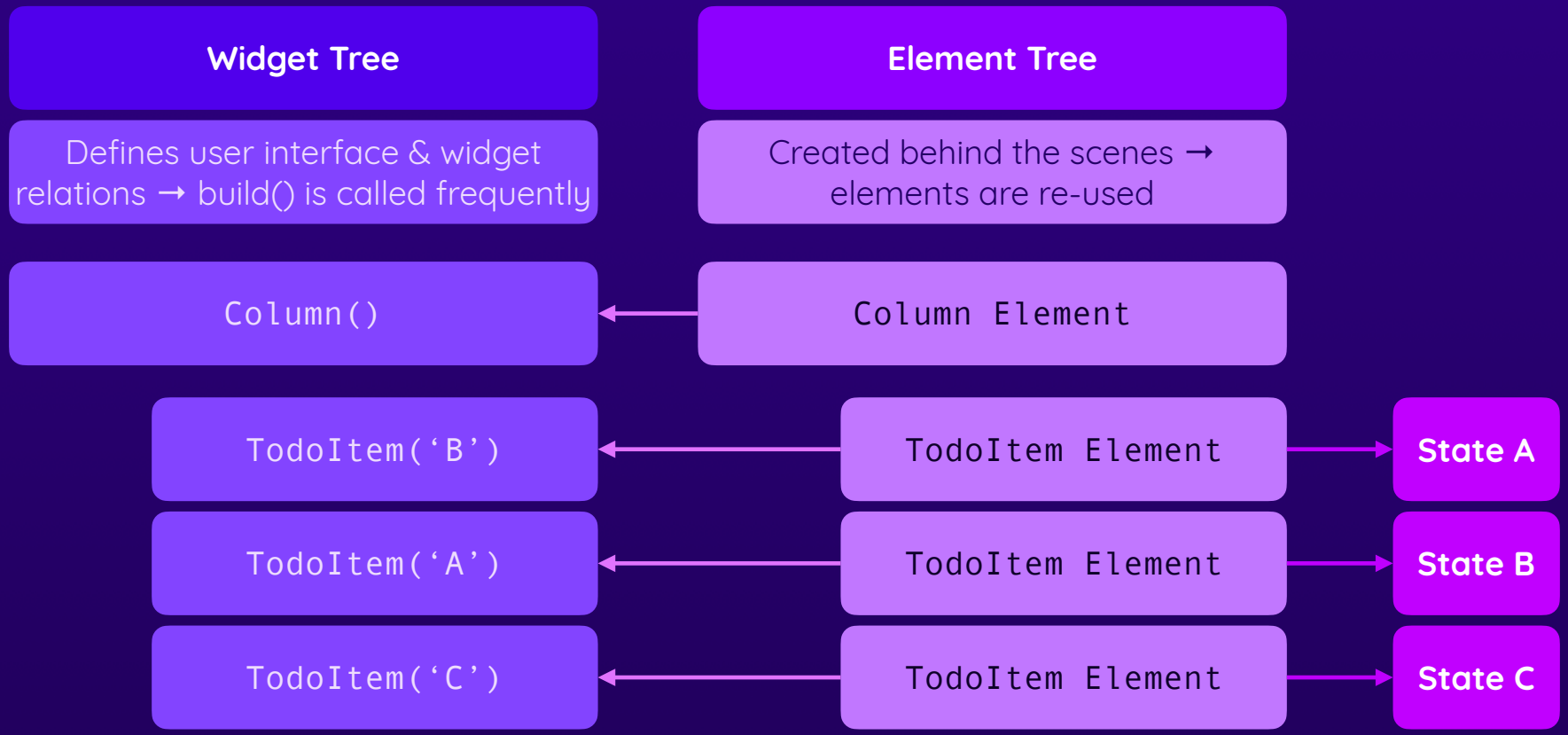
Element skeleton stays the same (i.e., same element structure)

Flutter only updates the widget references + UI output if necessary

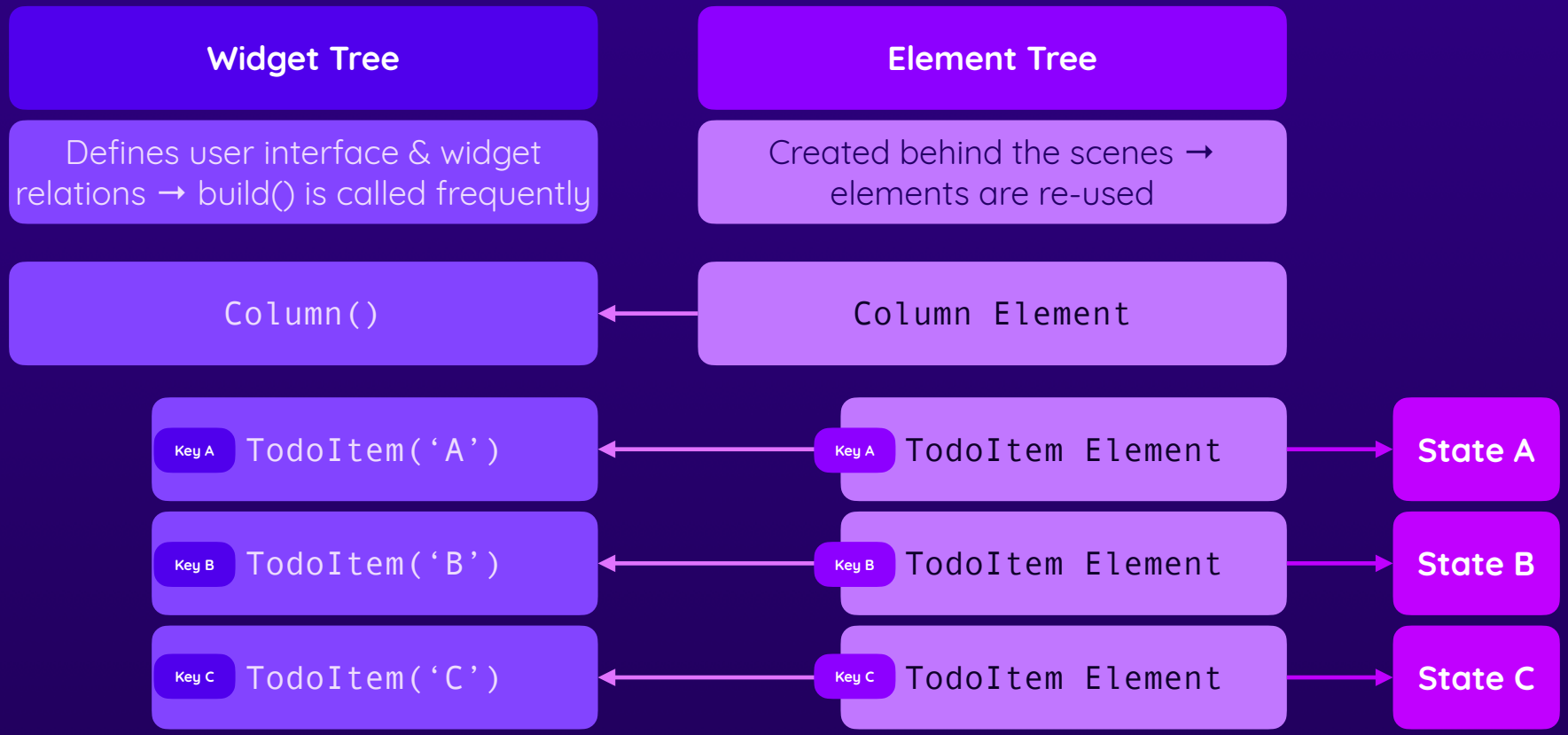
Widget & Element Trees



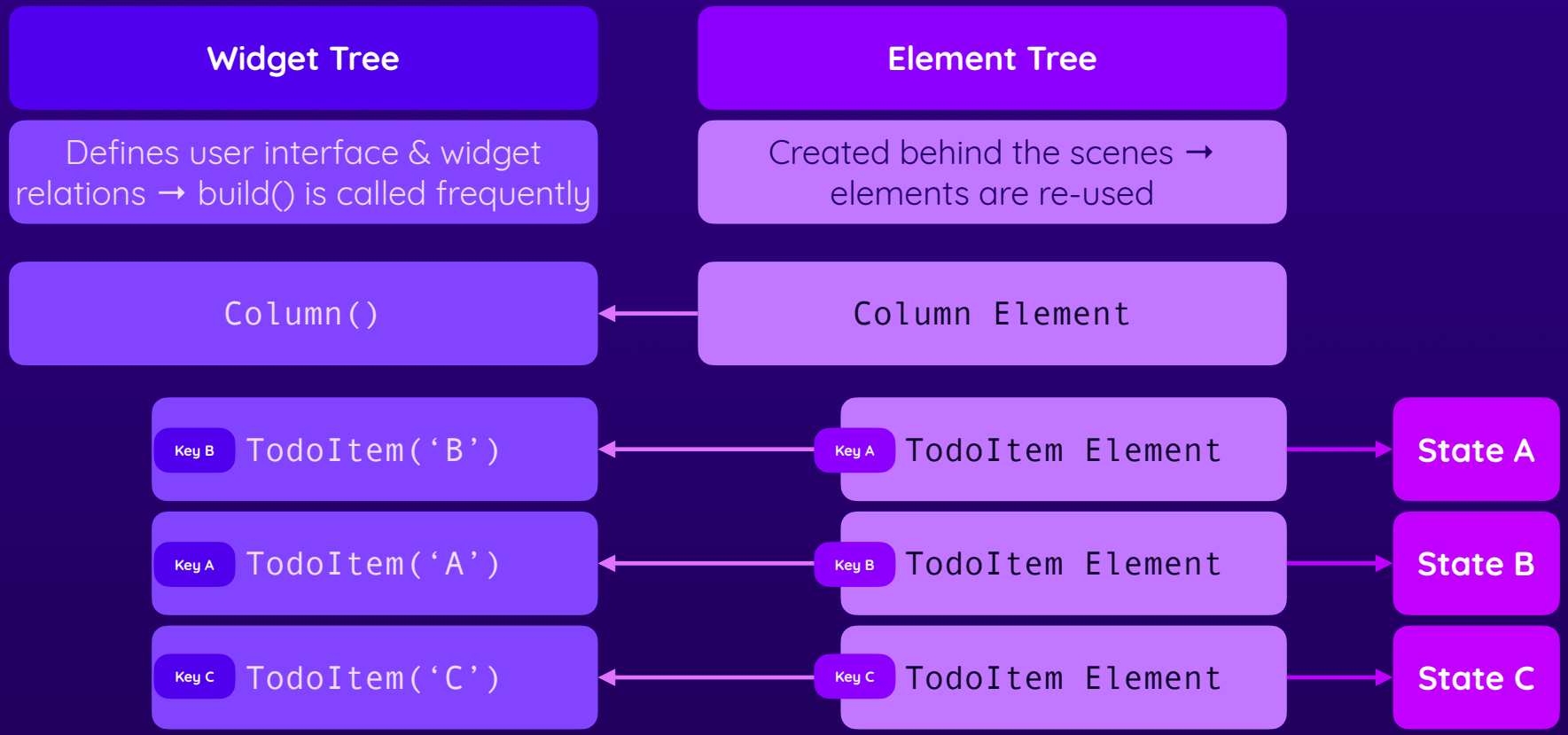
Widget & Element Trees



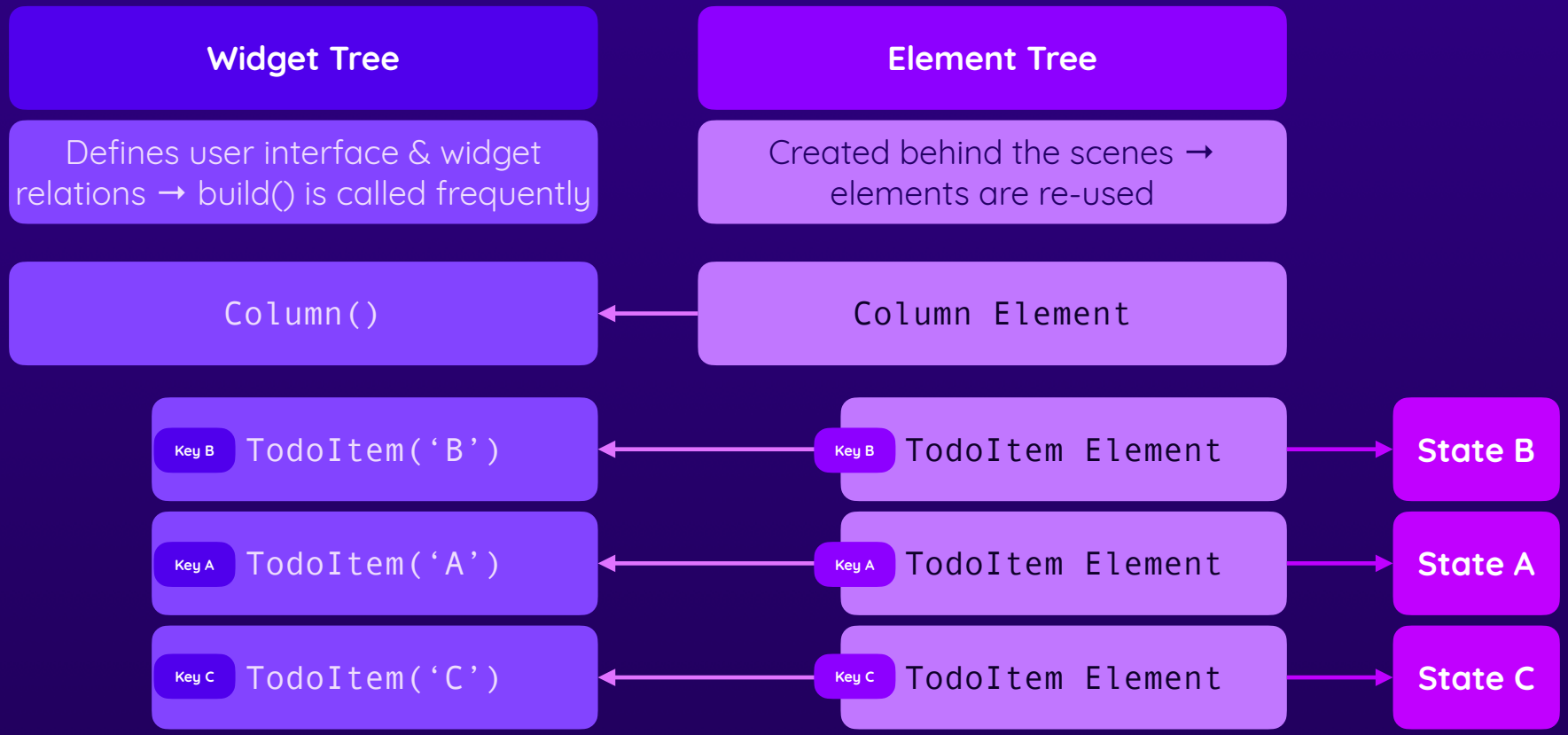
Widget & Element Trees



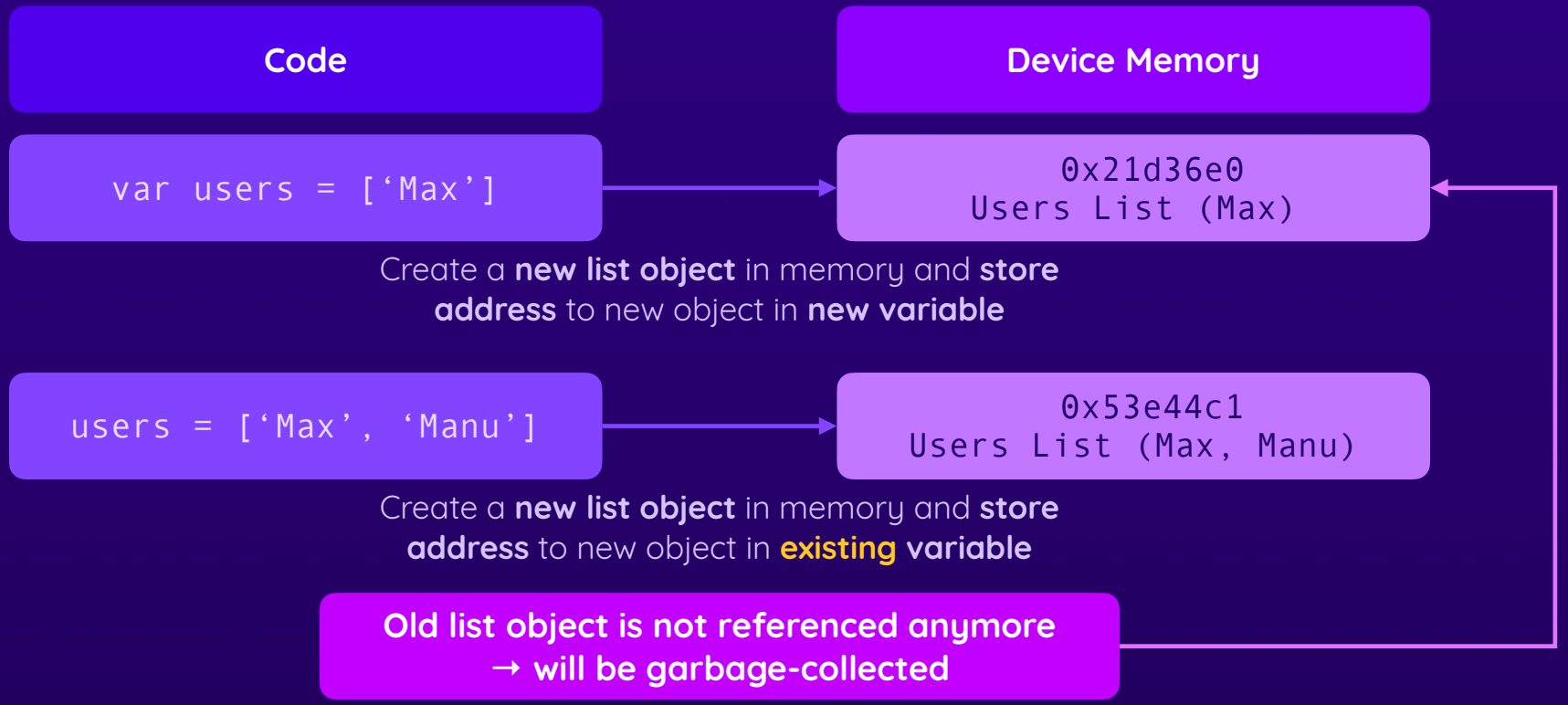
Widget & Element Trees



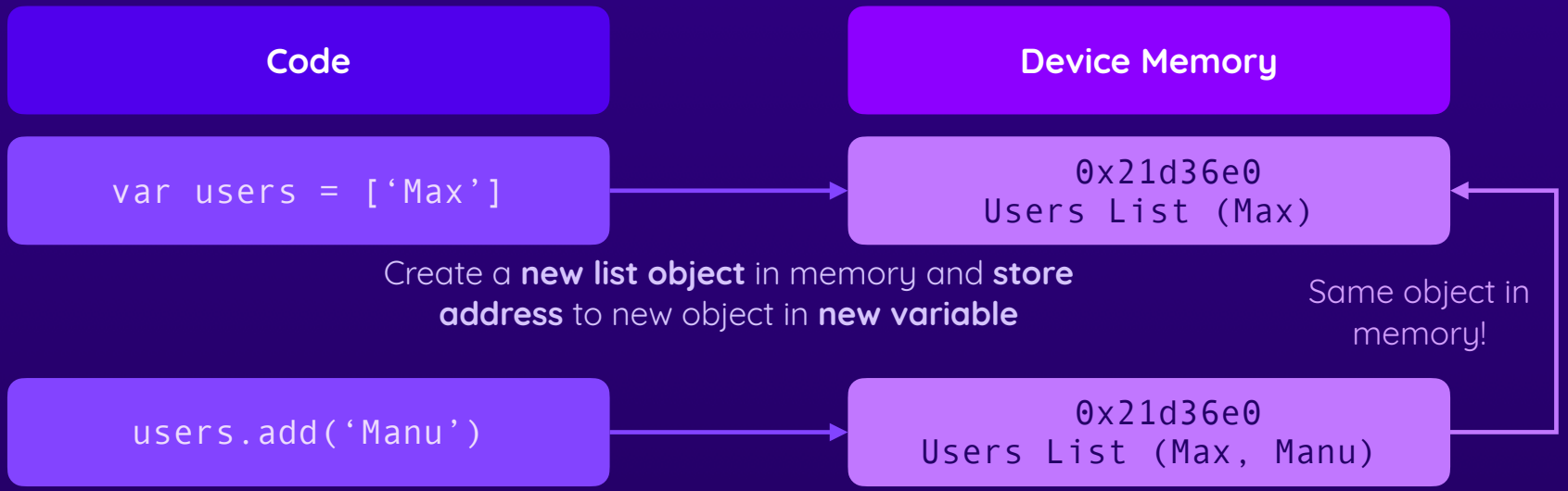
Widget & Element Trees



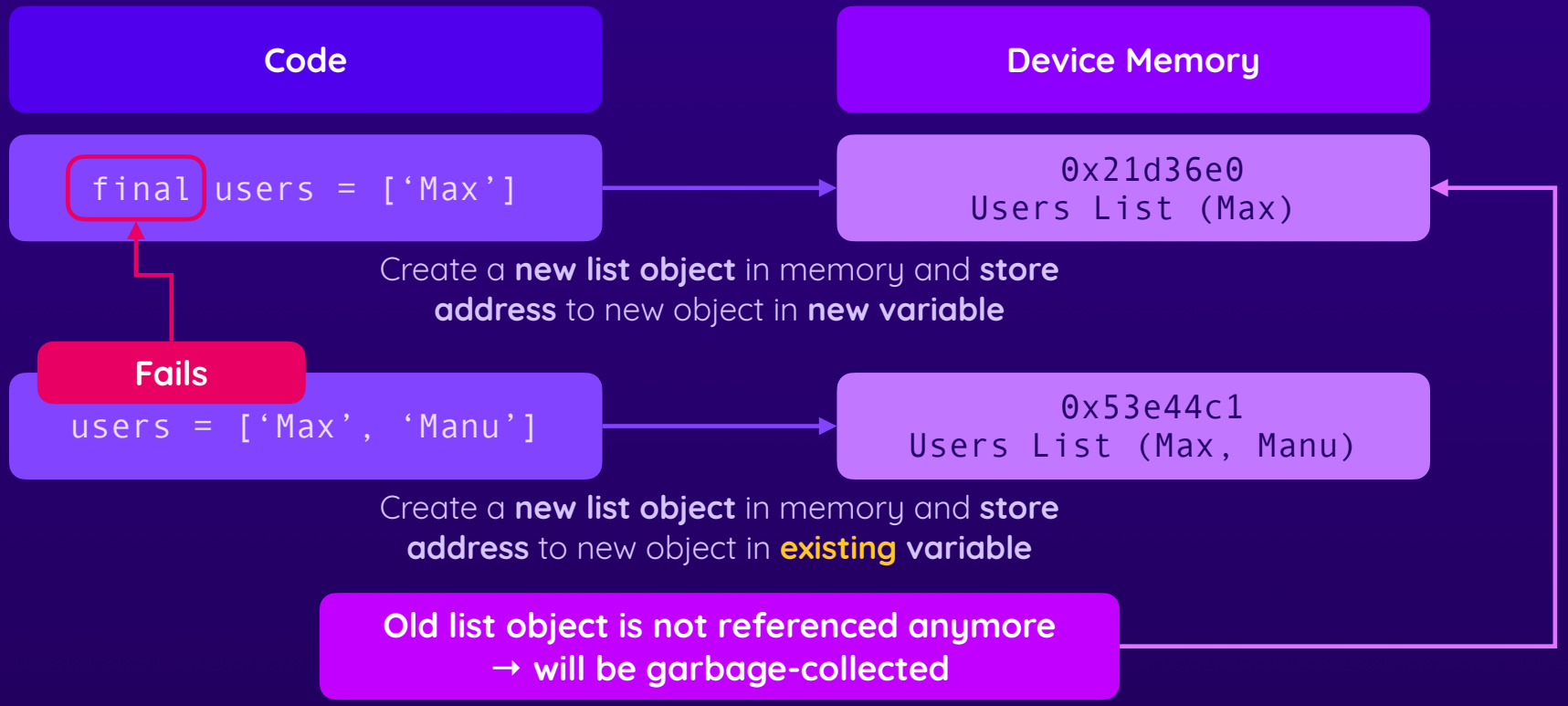
Variables Store Addresses To Values



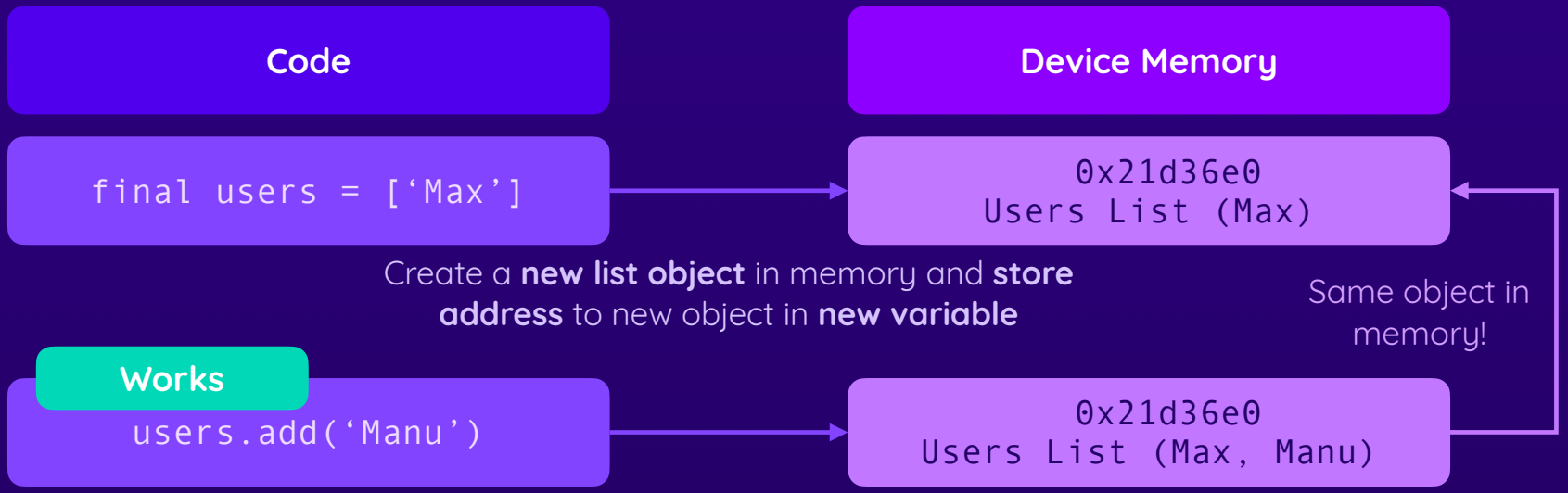
Mutating Objects In Memory



Variables Store Addresses To Values



Mutating Objects In Memory



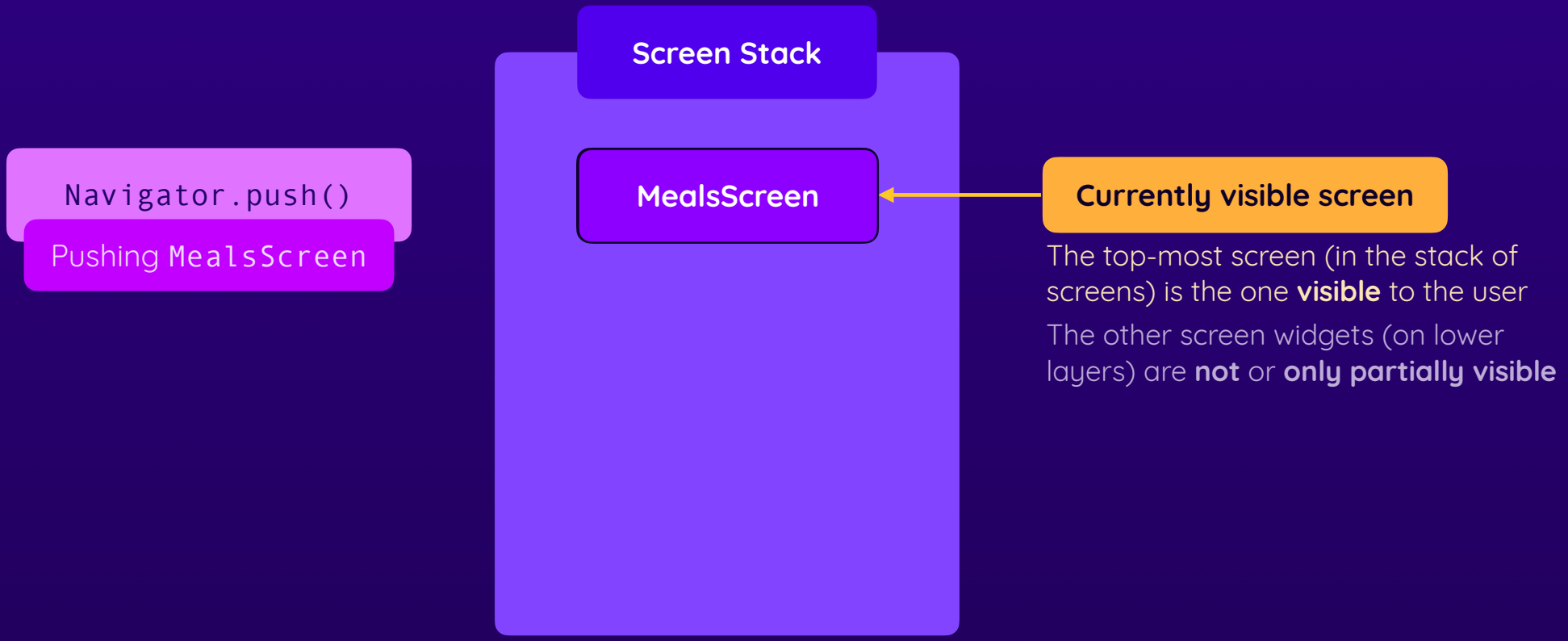
ACADEMIND

Navigation & Multi-Screen Apps

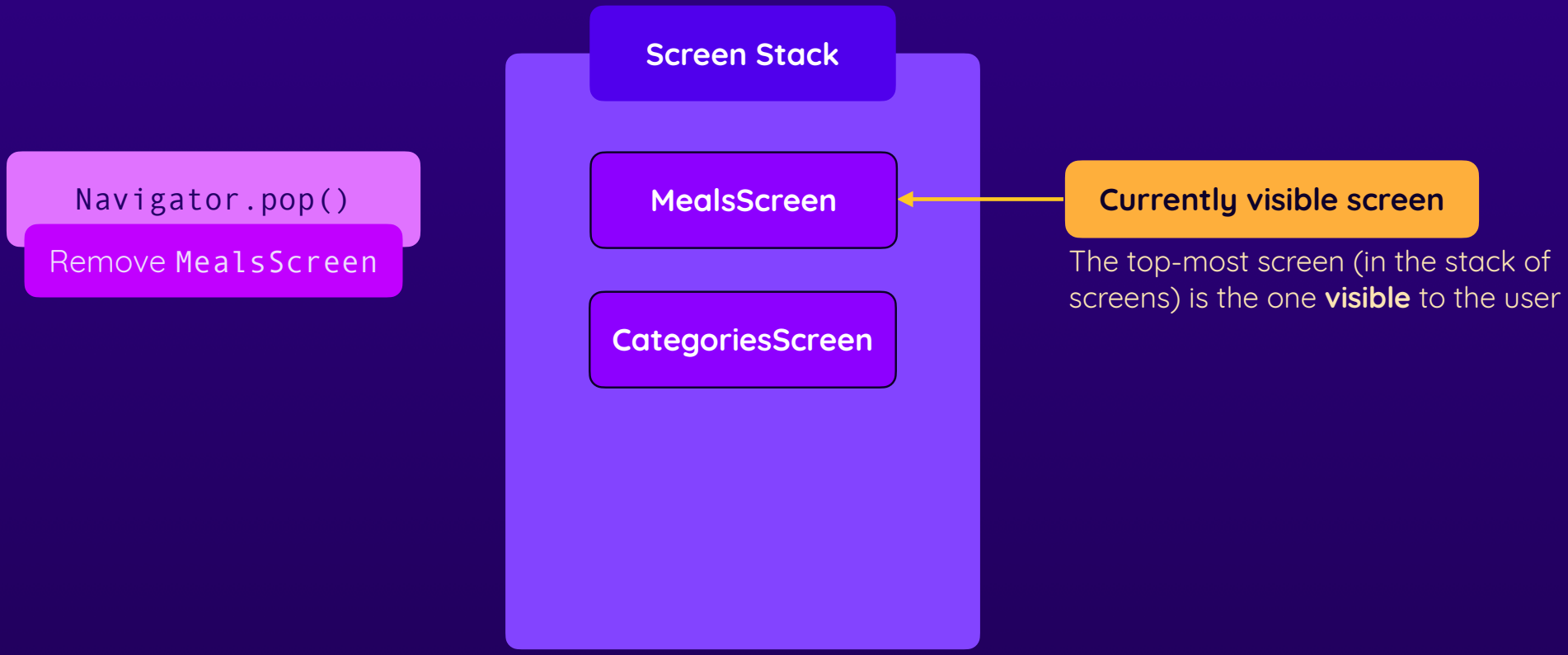
Allowing Users To Navigate Between Screens

- Managing Screen Stacks
- Working with Tab Bars
- Using Side Drawers

A Stack of Screens



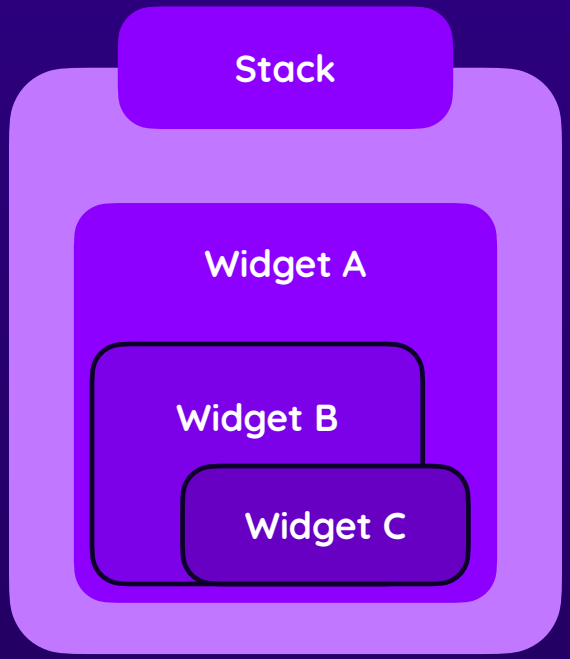
A Stack of Screens



The Stack Widget

Multiple widgets are positioned **next to each** other along the **Y-Axis**

e.g., a Text() above a TextField()



Multiple widgets are positioned **on top of each** other along the **Z-Axis**

e.g., a Text() on top of an Image()

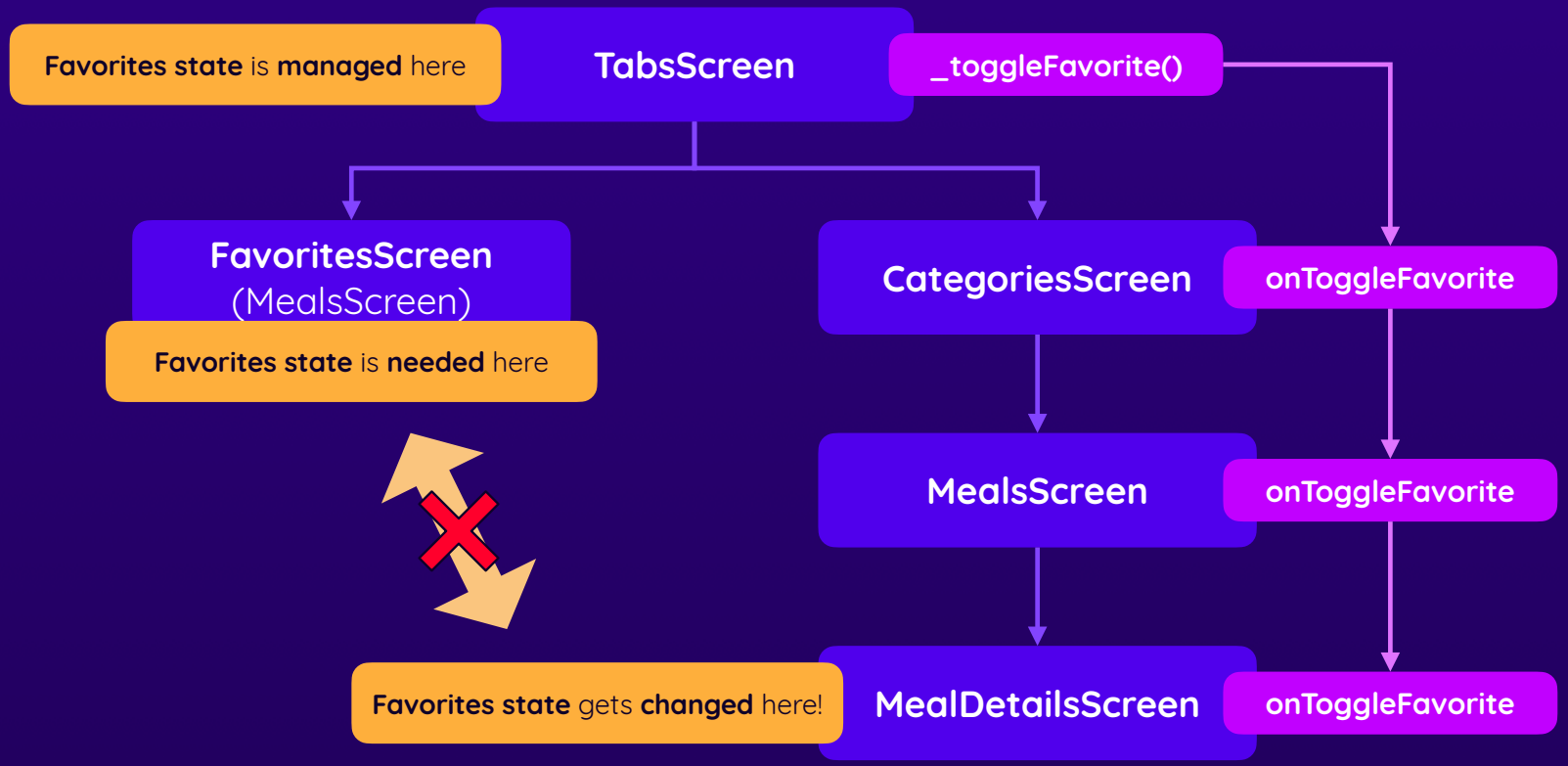
ACADEMIND

Cross-Widget State Management

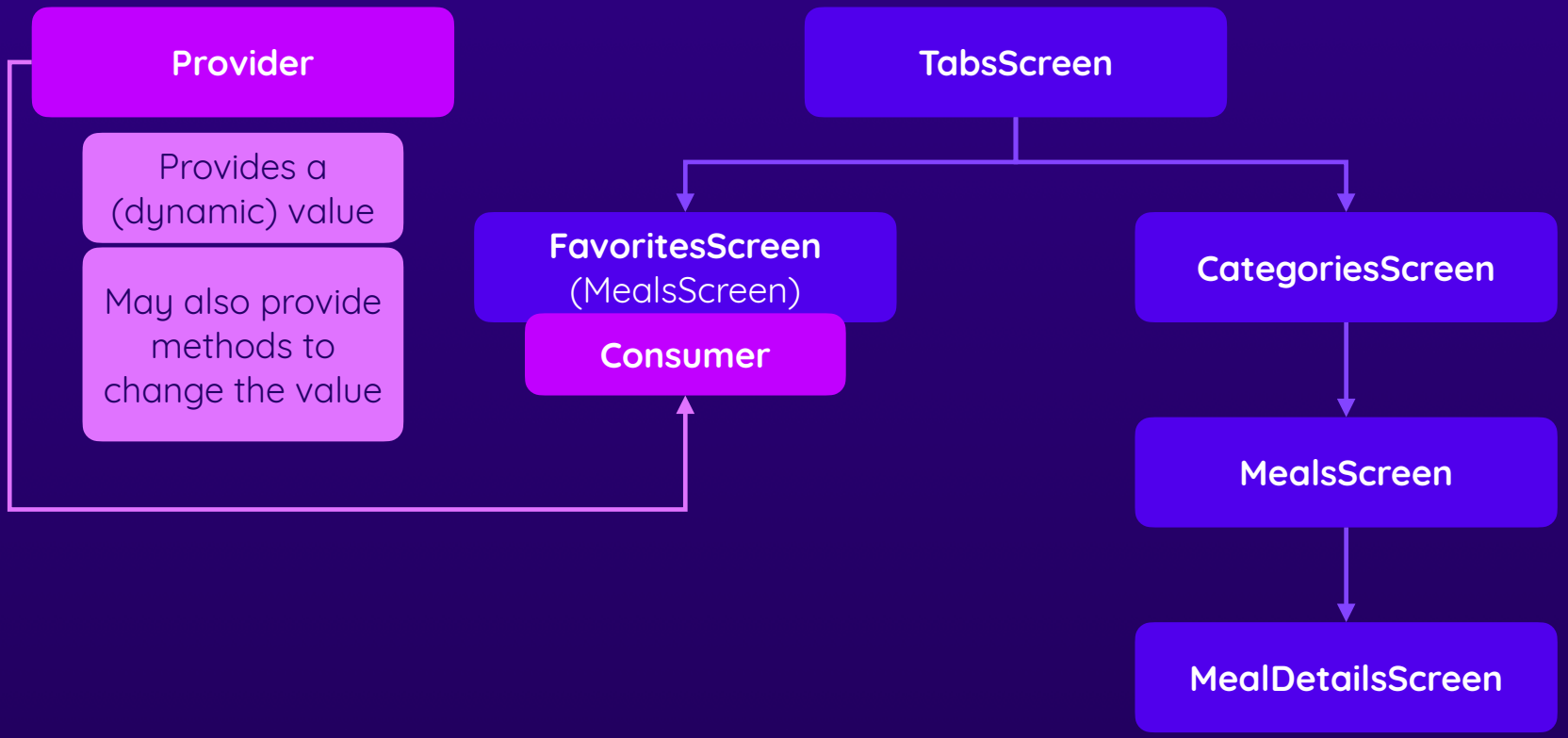
Making Things Less Complex

- What's The Problem?
- A Solution: The "riverpod" Package
- Using "riverpod" Providers

The Problem


Favorites state is managed here
TabsScreen
_toggleFavorite()
FavoritesScreen
(MealsScreen)
Favorites state is needed here
CategoriesScreen
onToggleFavorite
MealsScreen
onToggleFavorite
Favorites state gets changed here!
MealDetailsScreen
onToggleFavorite

How Riverpod Works



ACADEMIND

Animations

Making Things Move

- Explicit vs Implicit Animations
- Building a Custom Animation
- Using Built-in Animation Widgets

Explicit vs Implicit Animations

Explicit	Implicit
You control the entire animation	Flutter controls the animation
More control but also more complexity	Less control and therefore less complexity
Can often be avoided (by using pre-built Widgets)	Use pre-built animation widgets as often as possible!

ACADEMIND

Handling User Input With Forms

A Closer Look At Handling User Input

- Building & Using Forms
- Showing On-Screen Validation Errors
- Form Submission & Resetting

ACADEMIND

Connecting a Backend

Sending HTTP Requests From The App To A Backend

- Why Would You Add A Backend?
- Sending HTTP Requests From Flutter Apps To Backends

Why A Backend?

Flutter App

Runs on the user's device / mobile phone

Data is only stored locally (e.g., lost if the device is replaced)

Other users have no access to it (→ bad if data should be shared)

Communication via HTTP requests

Backend Server

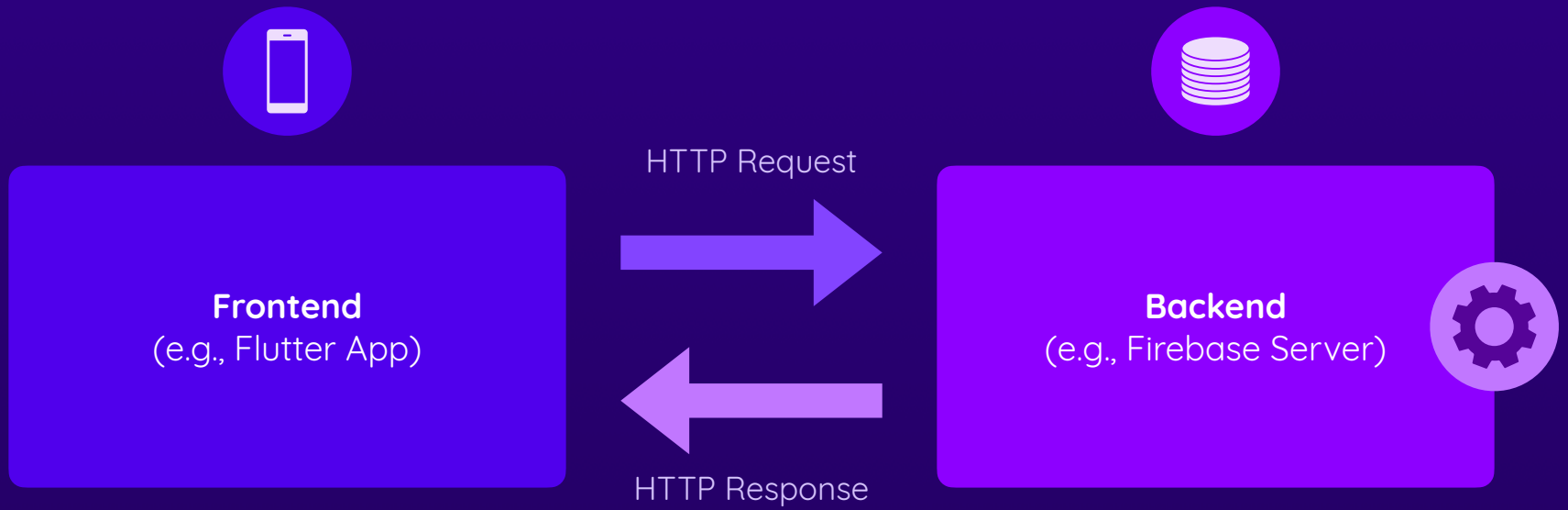
Runs on some server, somewhere "in the internet"

Data is stored in a central, remote place (e.g., SQL database)

App users from all over the world can interact with same data

ACADEMIND

HTTP?



HTTP Requests

- URL (address)
- Method (type)
- Headers (metadata)
- Body (data)

HTTP Requests & Responses are standardized "**information packages**"

HTTP Responses

- Headers (metadata)
- Body (data)

Different HTTP Methods