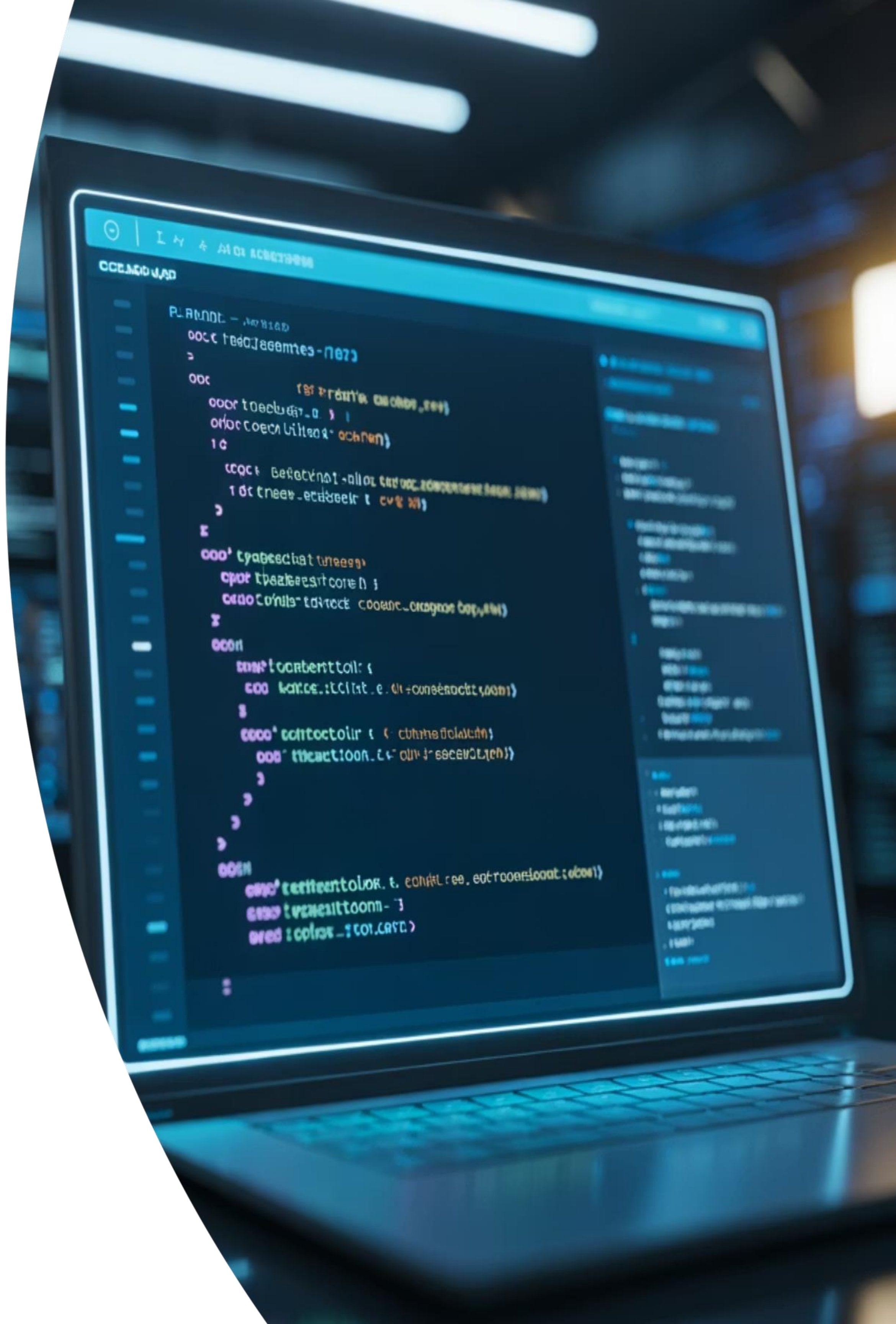
Application Development

Lecture 5

State Management in Flutter

Asst. Lect. Ali Al-khawaja

Class Room



Lecture Contents

01

Introduction to State in Flutter

02

Types of State (Ephemeral vs Shared)

03

Understanding Stateful and Stateless Widgets

04

setState() – Basic Local State Management

05

InheritedWidget – Flutter's Core Sharing Mechanism

06

Provider – Scalable and Modern State Management

General Objective

To provide students with a solid theoretical understanding of how Flutter manages and updates UI state using various mechanisms.

Behavioral Objectives

By the end of this lecture, students should be able to:

1. Define what "state" means in a Flutter applicatio

2. Distinguish between ephemeral (local) state and shared (global) state.

3. Explain how StatefulWidget and State classes function together.

4. Describe the purpose and limitations of setState().

5. Explain the mechanism of InheritedWidget as a state-sharing tool.

6. Understand the theoretical advantages of Provider for scalable apps.

What Is State?

State is **any piece of data that can change over time** and directly affects how the UI looks.

Examples of state:

- The currently selected tab
- Text in a form field
- Whether a user is logged in
- Items in a cart
- A counter increasing or decreasing

Two categories:

Ephemeral (local) → changes within a single widget

Shared (global) → needed across different screens

Activity 1 – Padlet

Scan the QR code to answer the question

Ephemeral vs Shared State

Ephemeral (Local) State

- Used within a single widget
- Temporary
- Best handled with **setState()**

Shared (global) State

- Needed in multiple parts of the app
- Examples: user session, theme, app settings
- Requires a scalable approach like **InheritedWidget** or **Provider**

Understanding which type of state you're dealing with determines the correct management method.

Activity 2 – Mentimeter

Scan the QR code to answer the question

Stateless vs Stateful Widgets

StatelessWidget

- Does NOT store state
- UI does not change after build
- Example: Icons, static text labels

StatefulWidget

- Contains a companion **State** object
- UI updates when internal data changes
- Used when the interface needs to be dynamic

Important: The state lives in the **State class**, not in the StatefulWidget itself.

Activity 3 – Hand Raising

Question: "Which widget type should be used for a button that changes color when pressed?"

setState(): Local State Management

setState() is Flutter's simplest mechanism to update UI.

How it works:

Update a variable.

Call setState() to notify Flutter that the UI needs rebuilding.

Flutter rebuilds only the affected widget subtree.

Strengths:

- Simple
- Ideal for small components
- Good for local state

Limitations:

- Not suitable for large applications
- Cannot handle shared state
- Leads to "state scattering" if overused

InheritedWidget: Foundation of Shared State

InheritedWidget is Flutter's **low-level mechanism** for passing data down the widget tree.

Purpose:

- Share state across many widgets without manually passing constructors
- Efficient rebuilds for dependent widgets only

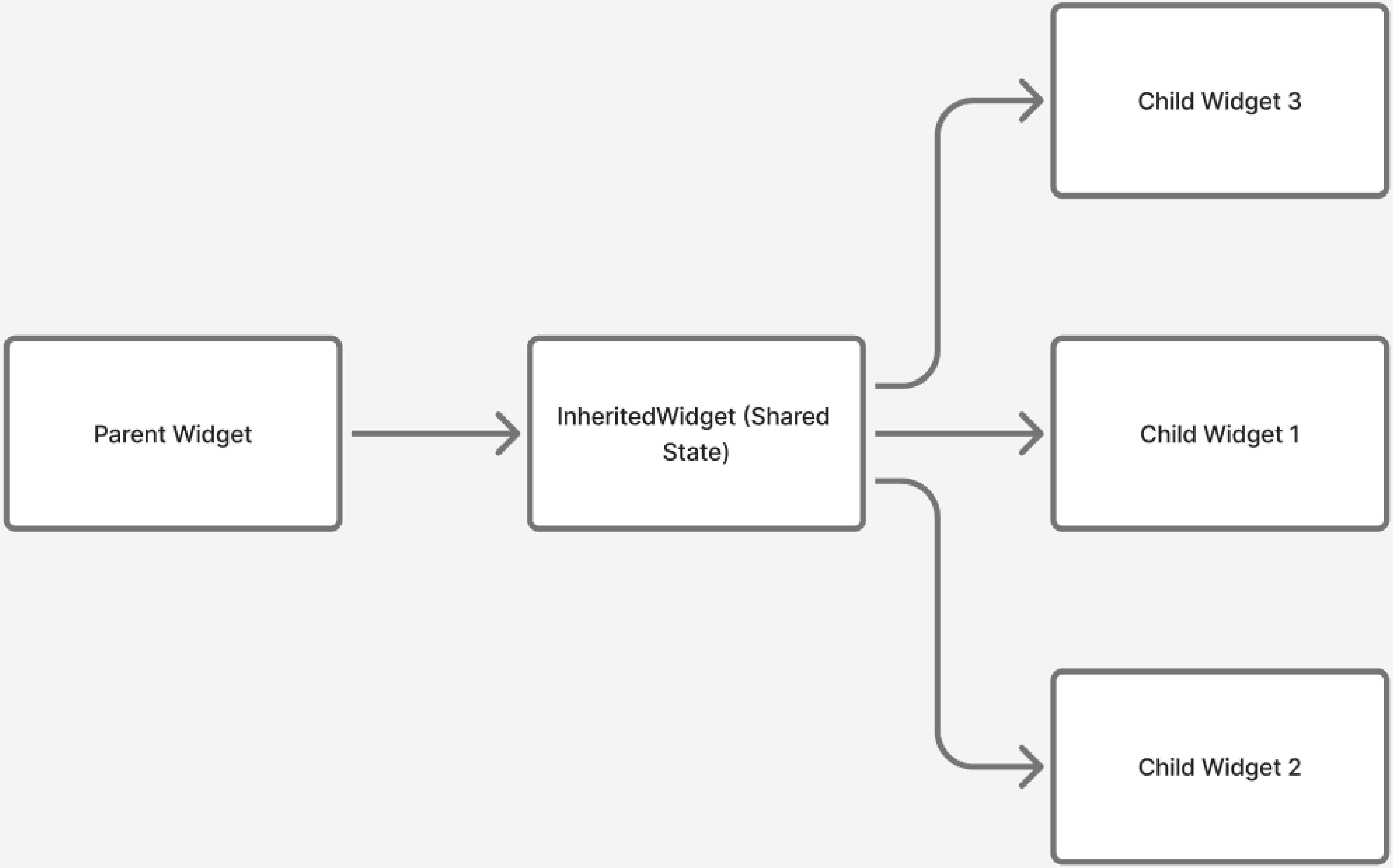
Key Concepts:

Parents provide data

Children "inherit" the data

Rebuilds occur only where needed

This is the base mechanism behind **Provider** and other state libraries.


Parent Widget
InheritedWidget (Shared State)
Child Widget 3
Child Widget 1
Child Widget 2

Provider: Modern Shared State Management

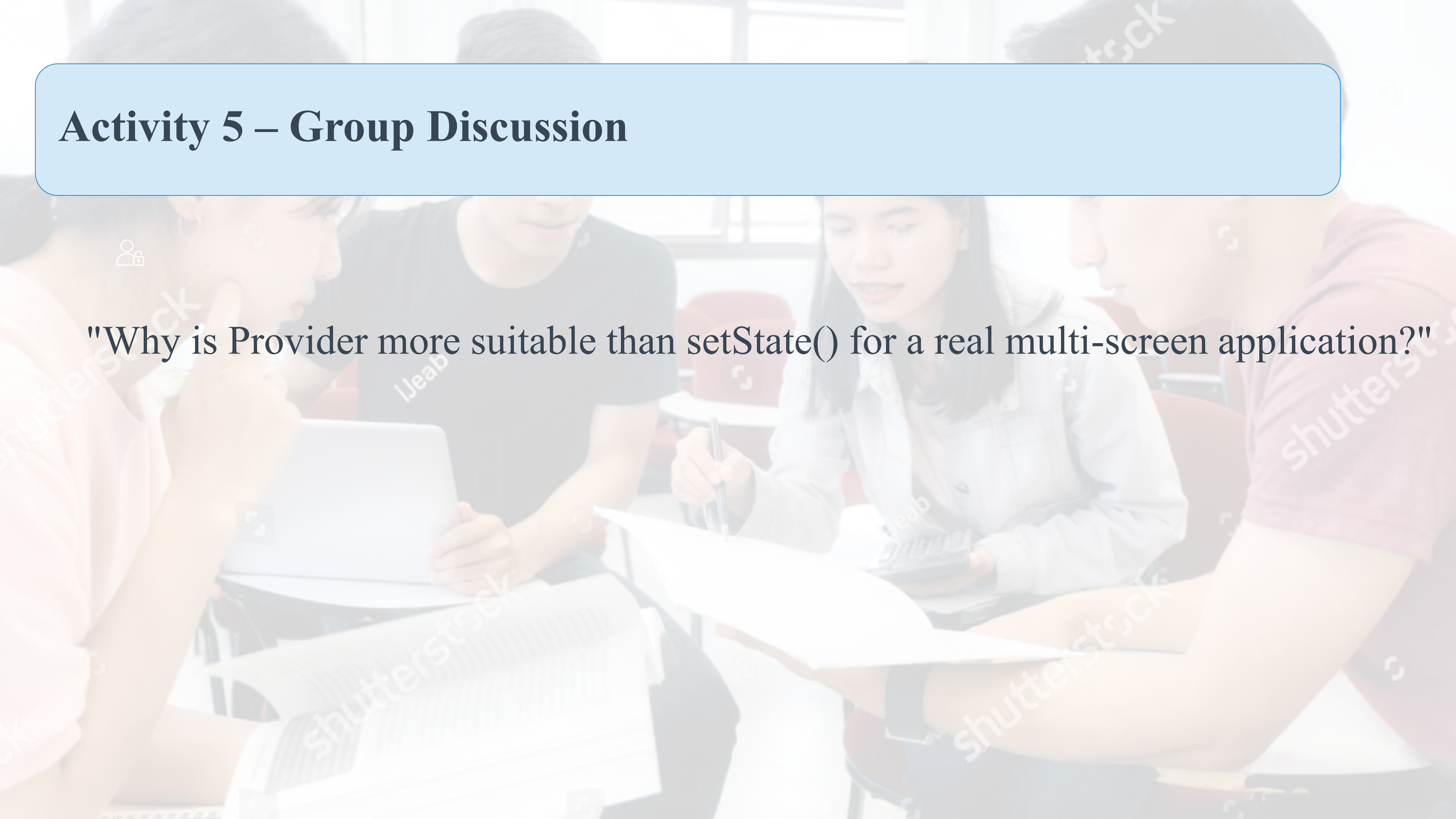
Provider builds on InheritedWidget but is easier and cleaner.

Why Provider is widely used:

- Simple architecture
- Minimal boilerplate
- Clean separation between UI and logic
- Scalable for large applications
- Officially recommended by Flutter team

Theoretical Workflow:

1. Create a ChangeNotifier class
2. Wrap the app with a Provider
3. Widgets read or listen to updates
4. Only listening widgets rebuild


shutterstock

Activity 5 – Group Discussion

"Why is Provider more suitable than setState() for a real multi-screen application?"

Choosing the Right State Management Approach

Method	Best For	Scope	Complexity
setState	Small/local updates	Single widget	Low
InheritedWidget	Custom shared state	Multi-widget	Medium
Provider	Modern & scalable shared state	Multi-screen	Low–Medium

Note: Choosing the right method prevents future refactoring and improves app design.

Thank you…

Any questions??

My google site

يرجى مسح رمز الاستجابة السريعة QR Code لتعبئة نموذج التغذية الراجعة حول المحاضرة. ملاحظاتكم مهمة لتحسين المحاضرات القادمة.