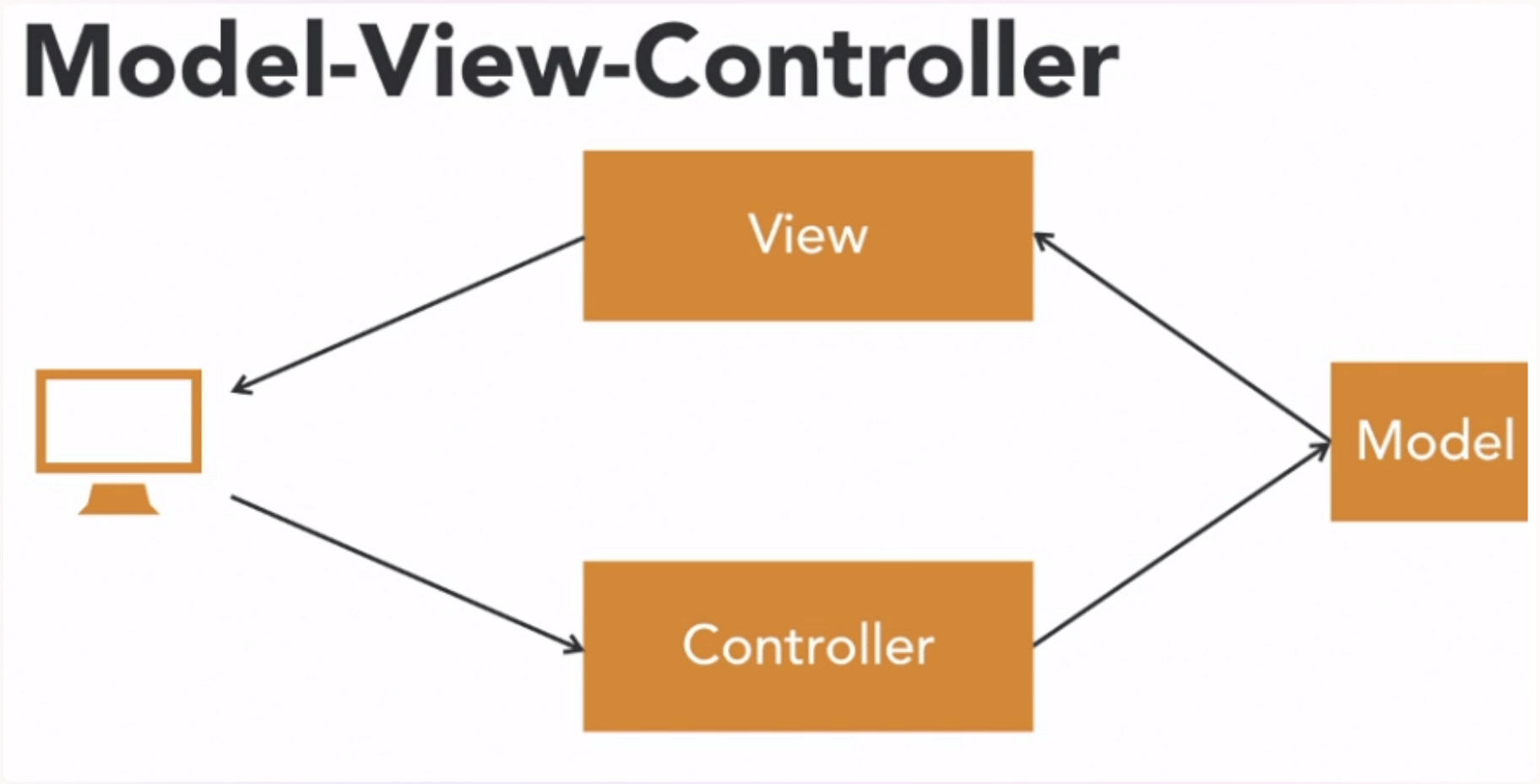
Exploring UI Software Architecture Patterns: MVC, MVP, and MVVM

Software architecture patterns are essential when it comes to developing UI/UX for software products. Out of numerous architecture patterns available, MVC, MVP, and MVVM are the most widely accepted. In this article, we will delve into each of these three patterns and discuss their advantages, disadvantages, differences, and use cases.

by Michael Hanna

The MVC Pattern: Breaking Down the Model View Controller



1 Definition

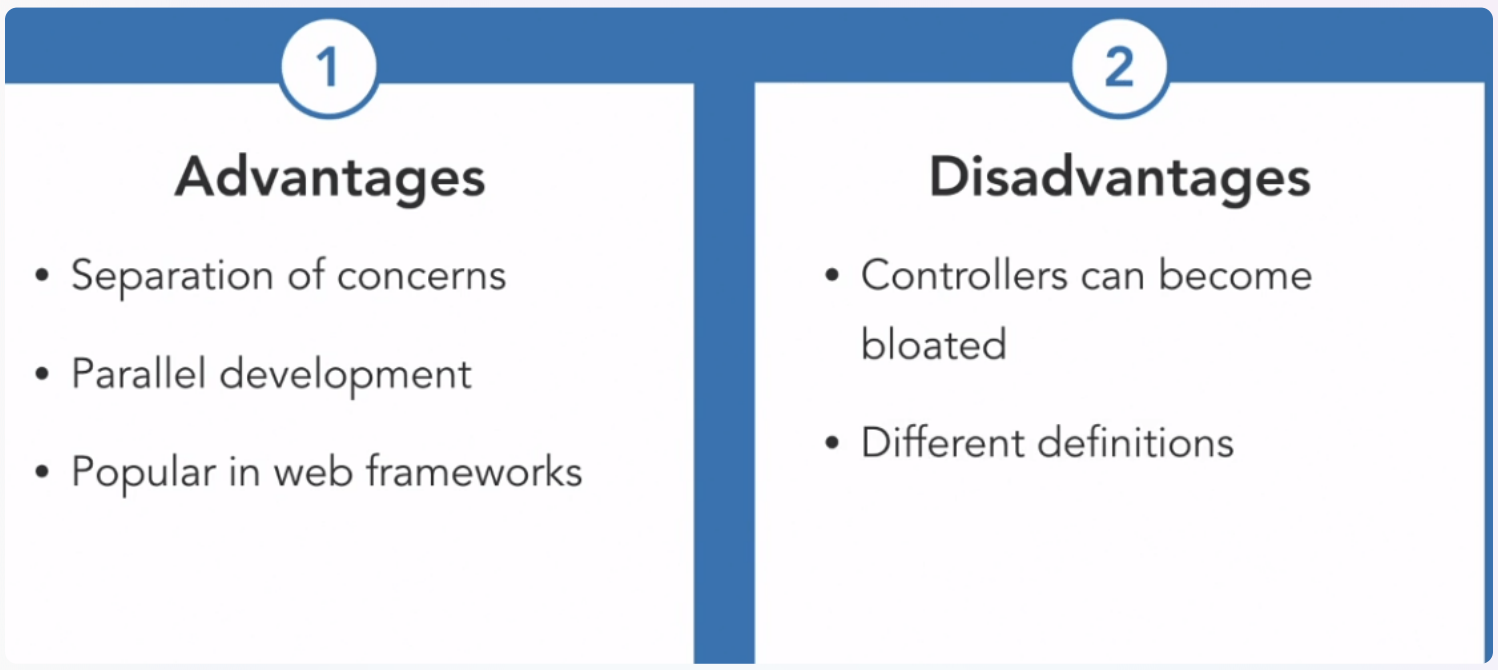
The MVC (Model View Controller) Pattern separates the application logic into three interconnected parts.

2 Advantages

The separation of the application logic makes it easy to modify or update the code.

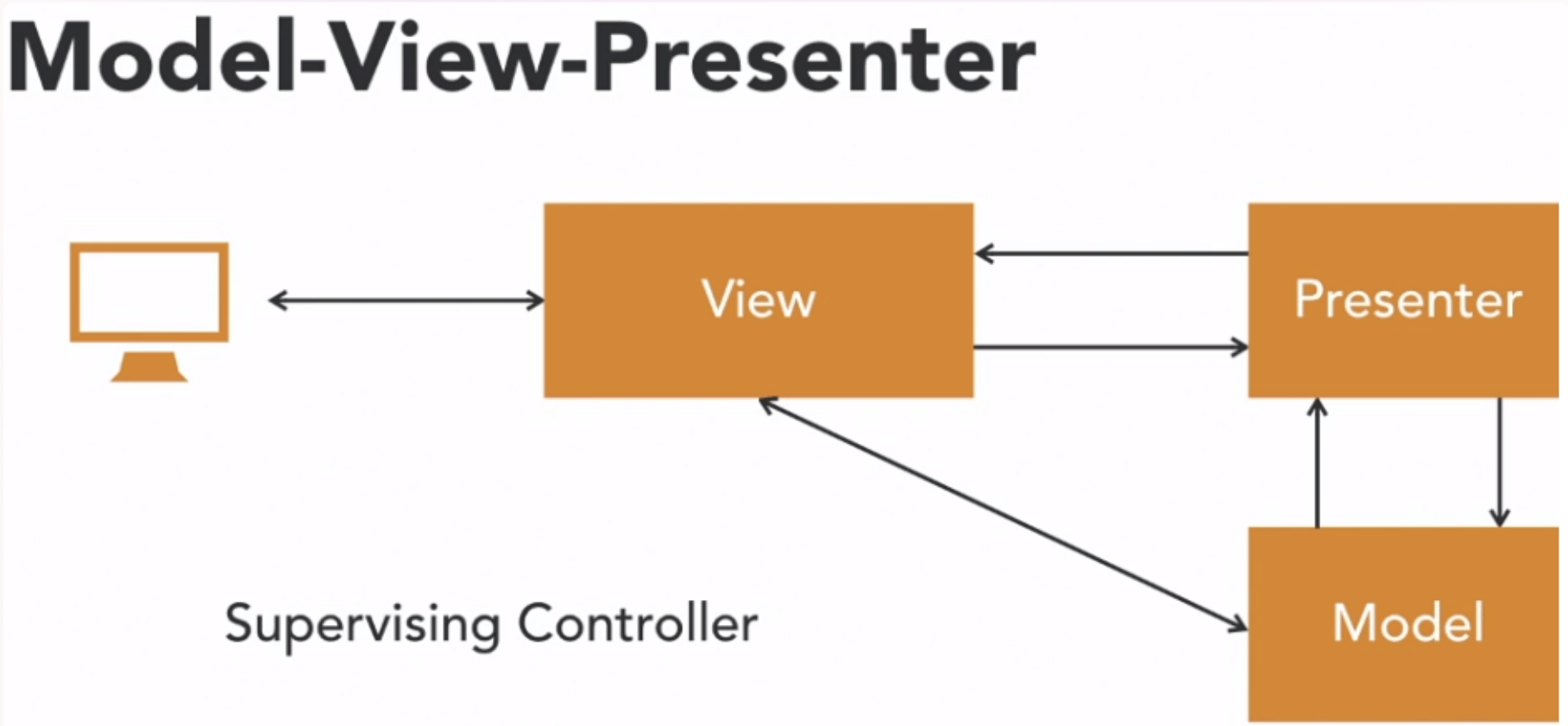
3 Disadvantages

It can be complicated and challenging to understand for beginners.



The **controller** in an MVC pattern is responsible for receiving user input.

The MVP Pattern: Decoupling the View and Model with a Presenter



Definition:

The MVP (Model View Presenter) Pattern separates the application into three parts and decouples the view from the model using a presenter.

Advantages:

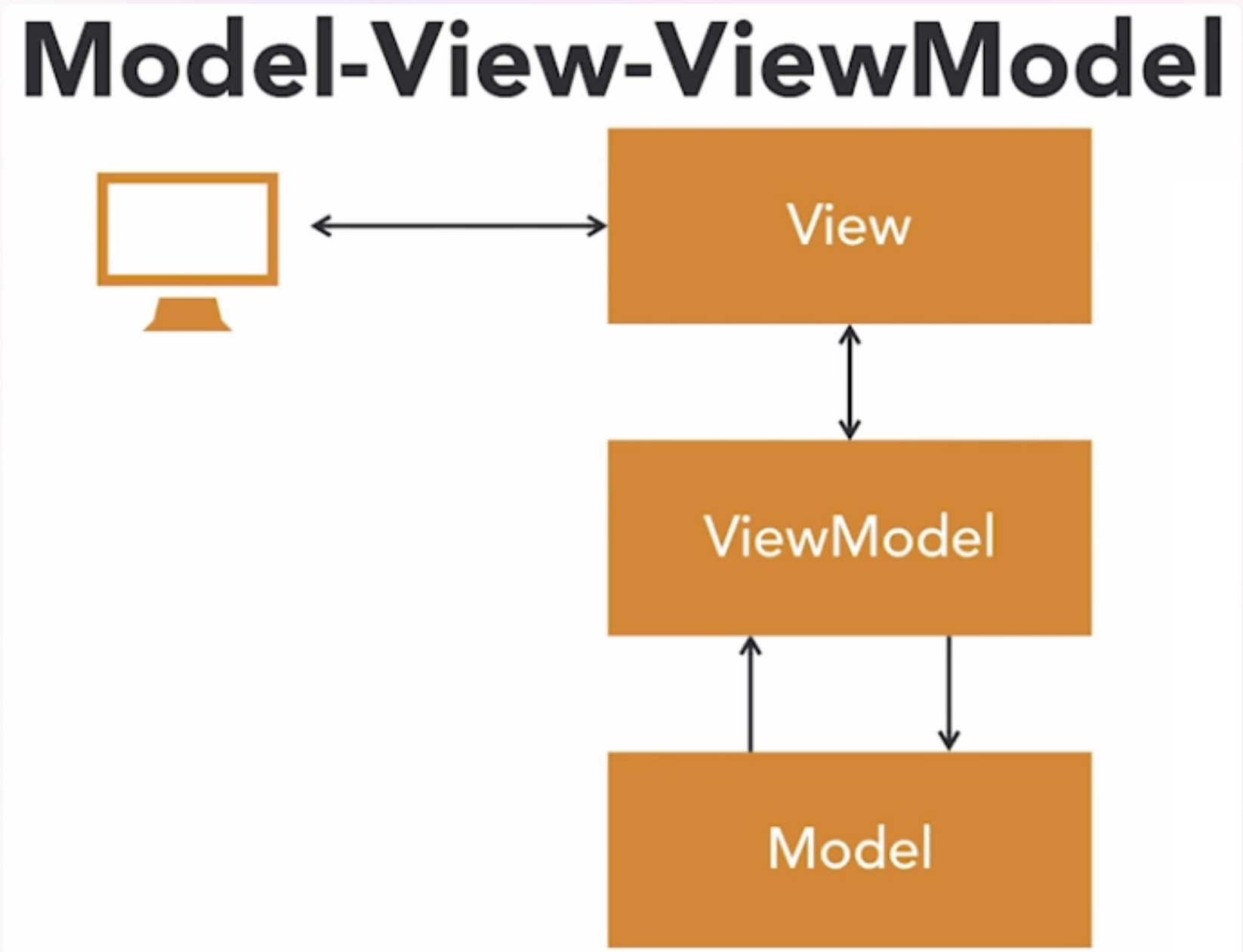
- The view is as dumb as possible and only responsible for displaying the data.
- The presenter is responsible for handling user input and updating the model and view.
- Testability.
- Great for desktop applications.

Disadvantages:

- It can be harder to test as compared to other patterns.
- Developing complex and larger applications may result in a growing number of presenters that can get out of control.
- Data binding can be difficult to debug.

The MVVM Pattern: Introducing the ViewModel

> The MVVM (Model View View-Model) Pattern is the third and most common type used in modern UI/UX design. The MVVM design separates the UI from the business logic and leverages the data-binding mechanism of the language/frameworks.



Definition:

The MVVM Pattern adds a ViewModel layer that sits between the view and model, consisting of data and commands that the view can bind to directly.

Advantages:

The separation of the application logic makes the code more modular and testable, which results in cleaner code.

Disadvantages:

Adding an extra layer can increase the complexity of code, which requires more effort.

Key Differences Between MVC, MVP, and MVVM

Pattern	View-ViewModel Binding	Usage	Advantages	Disadvantages
MVC	Loosely Coupled	Desktop Applications	Code Reusability and Scalability	Learning Curve and Code Complexity
MVP	Explicitly Bound	Mobile Applications and Legacy Systems	Clear Separation of Concerns	Complexity and Testability
MVVM	Tightly Bound	Modern UI Design	Modularity and Testability	Increased Code Complexity and Time-to-Market

MVC, MVP, and MVVM: Differences

	MVC	MVP	MVVM
User Interaction Handled By	Controller	View	View
Code in UI (Code Behind)	Minimal	Yes	Minimal
View Aware of Model	Yes	Yes (Supervising Controller) No (Passive View)	No
Data Binding	Basic	Basic (Supervising Controller) No (Passive View)	Advanced

What is the overall shared goal of the MVC, MVP, and MVVM patterns?

Decoupling the view and model which is better for testability

When To Use MVC Pattern

The MVC Pattern works best for simple desktop applications where the UI doesn't need to be updated frequently, especially for smaller software systems with a less complicated user interface, where the code can be optimized for code reuse and scalability.

Use in WEB: The MVC pattern is best because each one of the three components has clearly defined responsibilities.

When To Use MVP Pattern

The MVP Pattern is ideal for developing mobile applications and for legacy systems when the separation of concerns and the detachment of the view from the business logic without affecting testability is vital.

Use in basic data binding

When To Use MVVM Pattern

The MVVM Pattern works best for modern UI/UX design where the framework uses data binding and commands (e.g., WPF, Silverlight). It supports modularity and testability and is particularly useful when the software is developed in isolation of other software components.

Use in Advanced data binding: Android, IOS, WBF