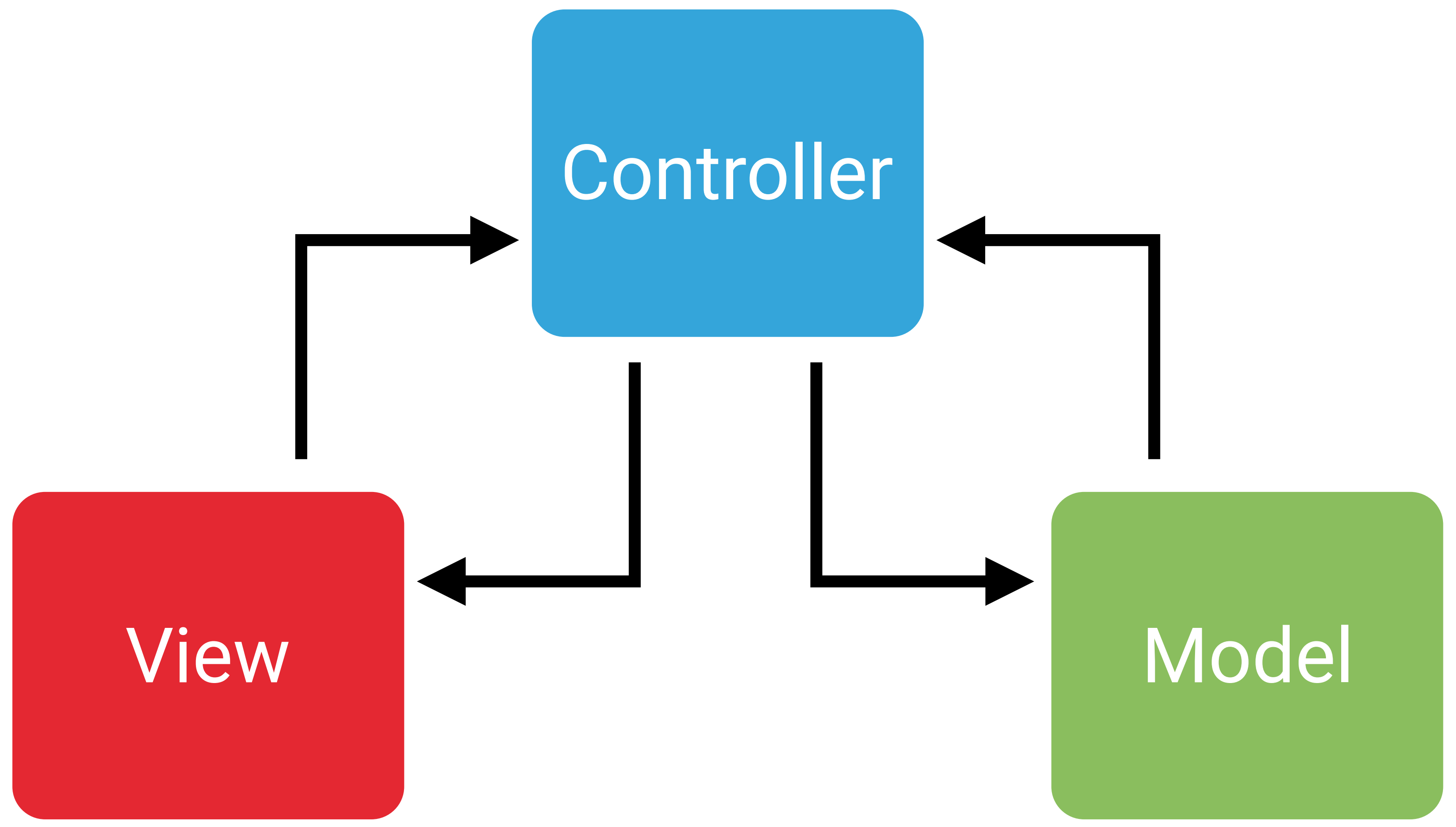

MOBILE APPLICATION DEVELOPMENT

ANDROID (2017)

LECTURE 12: MODEL-VIEW-CONTROLLER

SOFTWARE ARCHITECTURE

- The class has focused largely on building blocks of applications so far.
- While learning the fundamentals of a platform is important, implementing sensible architecture regardless of platform is often more important.
  - Making code predictable and reusable to other programmers is valuable.
  - Being 'correct' is often less important than being consistent when designing how large software systems are architected.
- Paradigms such as Object Oriented Programming describe how to structure code, but not (necessarily) how to structure entire programs.

MODEL-VIEW-CONTROLLER (AND VARIANTS)

- Model-View-Controller is a design paradigm that emphasizes separation of concerns between pieces of an application that perform different kinds of tasks. Code is organized into Model code, View code, and Controller code according to the sorts of functionality provided by that code.
  - Model code defines how data acts and is structured.
  - View code is concerned with presenting content to the user.
  - Controller code allows the Model representation to change, and Views to be updated to reflect those changes.
- Note: This is somewhat of an oversimplification, and a lot of other paradigms closely overlap with Model-View-Controller.

MODEL CODE

- Model code is where the data in the application is represented and given structure. This includes the state of the application itself, along with the various kinds of data that the application manipulates.
- Model code should ideally exist in a form that does not depend on the OS or graphical system in which it exists and which can be moved from system to system with minimal modifications.
- Model code is generally manipulated with Controller code, and transmits updates to Controller code using listener relationships or other forms of messaging.

VIEW CODE

- View code defines how information is presented to the user. It may include associated non-visual pieces of state that help control how to present content.
- View code generally receives updates from the Controller code with accessor methods, and may directly receive updates from the Model code (but this should mostly be avoided).
- View code generally transmits updates to the Controller code using listeners or other forms of messaging, and may update the Model code directly (again, this should mostly be avoided).

CONTROLLER CODE

- Controller code serves as the intermediary between Model code and View code, and defines how interactions with one affect the other.
- A general goal of Controller code is to keep the Model updated in response to View events, and to keep the View code updated in response to Model events.
- Controller code is often the least portable part of a program, and should be thought of as a place where all of the situationally-specific concerns of a program can be addressed (such as hardware interaction, OS-level dependencies, and so on).

ADVANTAGES OF MODEL-VIEW-CONTROLLER

- Separates concerns to a point where software components relating to either the Model, View, or Controller can be understood in isolation.
- Allows multiple people to work on different parts of an application without needing to coordinate on anything except interfaces between components.
  - If interfaces are designed carefully, nearly an entire application can be 'filled in' without the need to share implementation details between code on either side of an interface.
  - Makes it easy to swap one part of the system out for another without changing the rest of the system.
- Multiple Views can represent a Model, multiple Models can populate one View, etc.

MODEL-VIEW-ADAPTER

- Model-View-Adapter is an architectural paradigm which is closely related to Model-View-Controller, with a few important differences.
- Model-View-Adapter imposes a strict separation between Model and View code.
  - The View should have no knowledge of the Model behind it.
  - The Model should have no knowledge of the View that will display it.
- Android's AdapterViews and their associated Adapters can be a good example of this paradigm in action (assuming the programmer does not violate the separation of concerns while using them).