Till last class

1) Mobile HW and the Android software stack

2) Apps are made; Android Studio (IDE)
⇓
Android Software Development Kit (SDK)
⇓
Compiler + Emulator

3) How to create and handle user interfaces
⇒ Column, Add images; Buttons; RadioButtons, Text, Scrollable TextField } Lazy fields, etc.

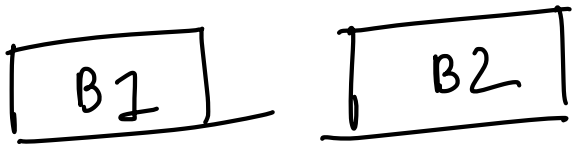
Concept of Composition & Recomposition

Why is UI creation & management challenging?

⇒ 1) Users have different expectations depending on the type of display available.

* Larger phone ⇒ Expect higher resolution
⇒ Scrollable Text ⇒ No scrolling needed if the content is already visible.

* App's running logic can get mixed up with the UI's logic ⇒ makes the program logic complex.



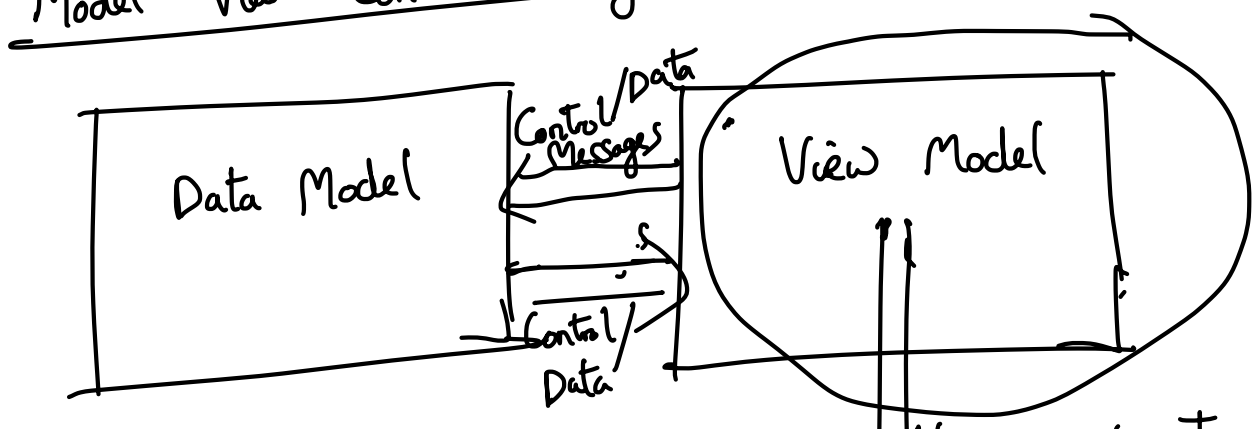
} UI-State ? not the underlying data layer.

if B1 is pressed once:
B2 should be inactivated.
if B1 is again pressed:
B2 should be activated

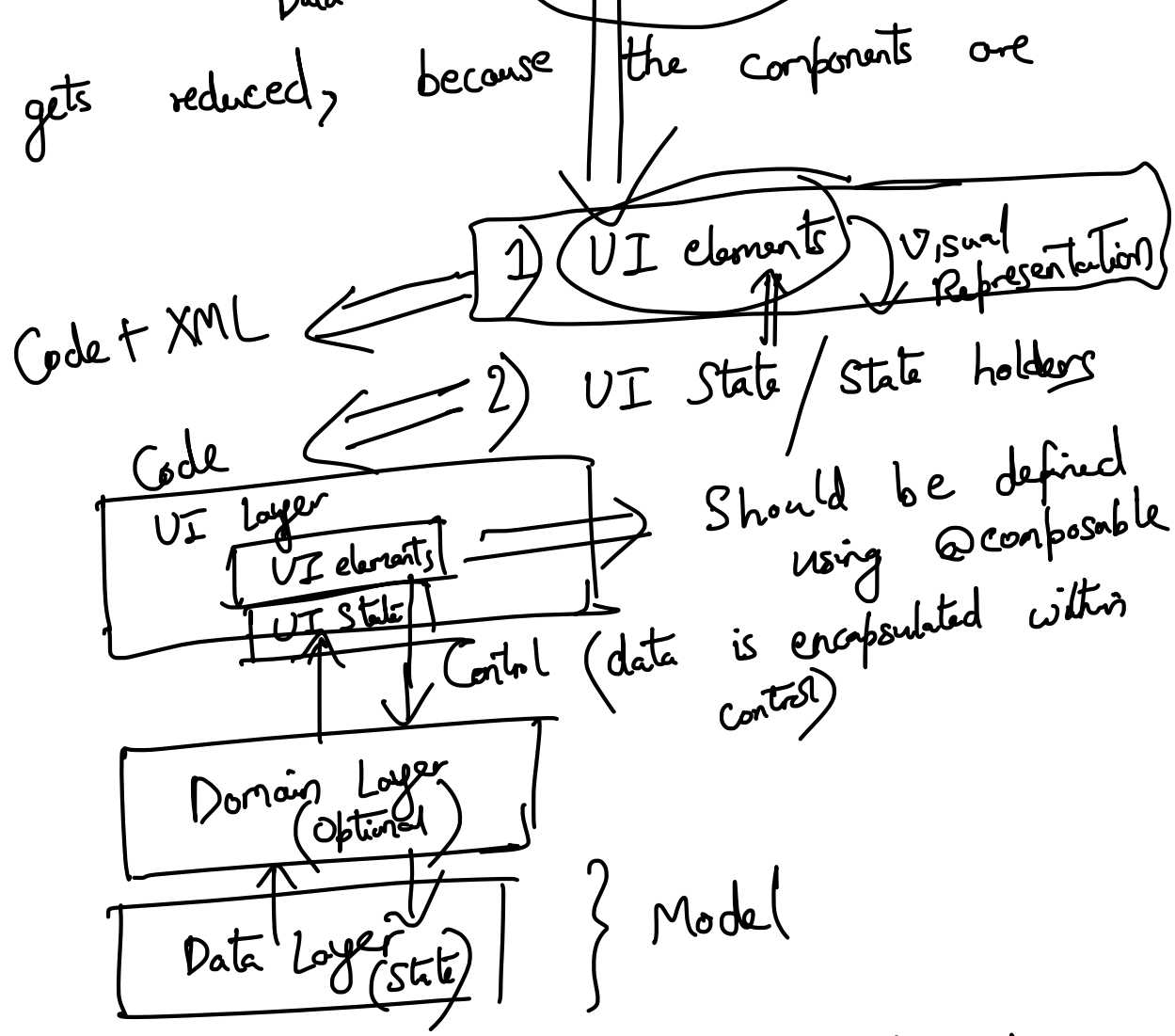
UI's logic is dependent on the app's logic; so hidden complexities get involved.

UI's control messages are asynchronous in nature.

Model - View - Control Design



Complexity gets reduced, because the components are separated.



Data should always move from lower to higher in the heirarchy.

Control should flow from upwards to downwards in the hierarchy.

Unidirectional Data Flow Model

A lot of logic is connected not to the underlying data, but only to the UI.

Cache of a web browser ⇒ Data Layer.

Some webpages are bookmarked

⇒ Remove a bookmark

⇒ { UI State (not part of the data layer)

How does Compose help in implementing the above?

Domain Layer ⇒ Adapt the data shown to the device that is being shown

⇒ Browsers (mobile & desktop version)

⇓

show less data on webpages (adaptation performed by the domain layer)

⇒ WhatsApp (Desktop / Mobile version) ⇒ The data sent changes depending on the device.

@Composable
fun ( ) ⇒ Used for describing the UI

Within the same file, create a class with data variables that can be modified by the listeners.

To respond to the control messages, utilize Mutable State.

Performance of Android Systems

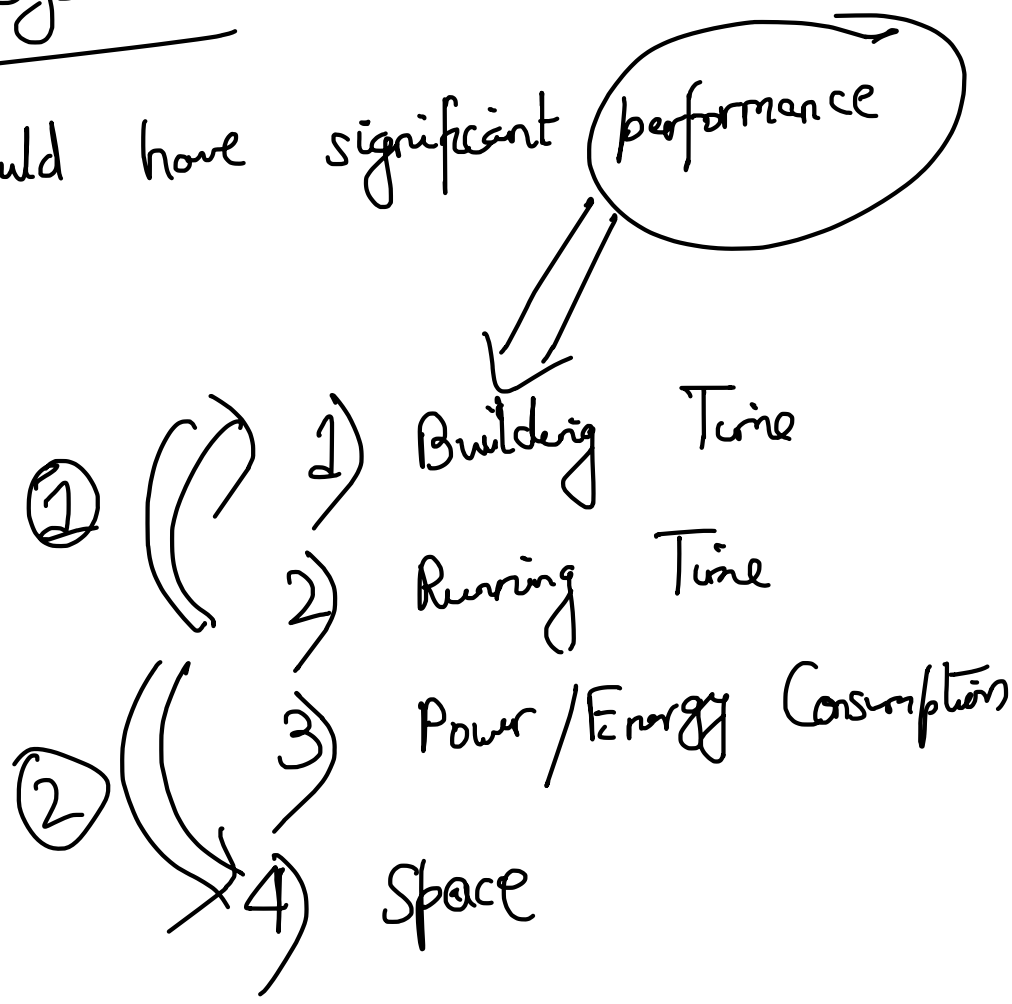
1) Android's layers should have significant performance disadvantages.

⇒

Composible, Layers, etc ⇒
where do they eventually translate to ??

⇒ Bytecode

⇒ Very optimized code; but for the VM that Android is compatible with (Dalvik VM) ⇒ Performance becomes better with more iterations



Does Android have this disadvantage ?

⇒ Till Android v4
(Just in-time compilation)
⇒ Translate the bytecode into machine code as requests to execute instructions come in).

Building ⇒ { Ahead-of-Time Compilation
(Get the app ready for each individual HW even before it is downloaded)

⇒ Extensive caching is involved.

All apps are statically built.
{ ⇒ All the dependencies are part of the app package.

UI State { How does the UI layer adapt to such performance constraints?

⇒ Phone's Theme ⇒ Dark Theme and Light Theme
⇒ The UI layer can access the theme used on the smartphone, and accordingly adapt its display.

Adapt the colors

⇒ Adjust size/ refresh rate of display according to the requirements

(refresh rate integrated into modern smartphones)

UI State { ⇒ Video & multimedia applications ⇒ can reduce quality.

Lazy Fields ⇒ Scrollable ⇒ Very large amount of content possible

⇒ Might crash your phone

(At a conceptual level)

⇓

Render only when necessary.

Infinite amount of scrolling in the terminal; soon runs out of memory.

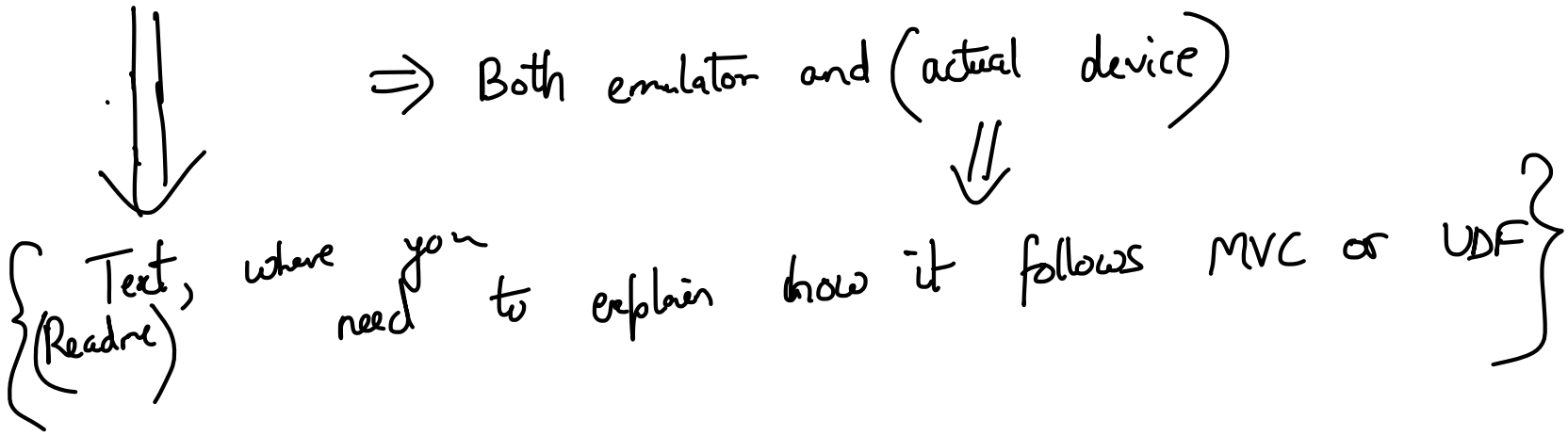
{ LazyColumn; Has a smaller Composable function that is triggered only when a signal is sent to it ⇒ Goes to render content again.

---

1) Calculator App ⇒ +, −, ..., etc.

⇓

Vehicle's Calculator ⇐ (Distance; velocity; acceleration;) and convert them into different units.


⇒ Both emulator and (actual device)
Text, where you need to explain how it follows MVC or UDF
(Readme)