SCIENCES SORBONNE UNIVERSITÉ

Views & view controllers

Fabrice.Kordon@lip6.fr

LIP6

As an introduction…

Remember, views are the canonical element

- Background, sensitivity, enclosed elements or drawings
- «Widget» : UILabel, UIButton, UISlider, UISwitch, etc.
- You may extend existing mechanisms
- You may invent new mechanisms

How to deal with view?

- Create these
- Associate some handler to the supported events
- WARNING:
  Some event are not supported by any widget…

Building simple Applications

Up to now

- Enrichment of a «ViewControllen»
  - **Creating data structure**
  - **Building the view with StoryBoard («IBOutlets»)**
  - **Implementation of «IBActions»**

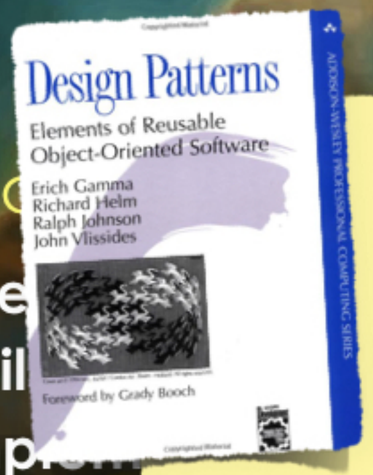
But…

- No separation between data and their display
- How to handle superposition of views
- How to manage efficiently various screen sizes
  - **Separate concerns : data, display, related actions**

Building simple Applications

Up

- Enric
  - Cre
  - Buil
  - Impl



But…

- No separation between data and their display
- How to handle superposition of views
- How to manage efficiently various

  - Separate conce

Model, View, Controller (MVC)

years 1970 (Smalltalk — Xerox)
«good practice» for programming

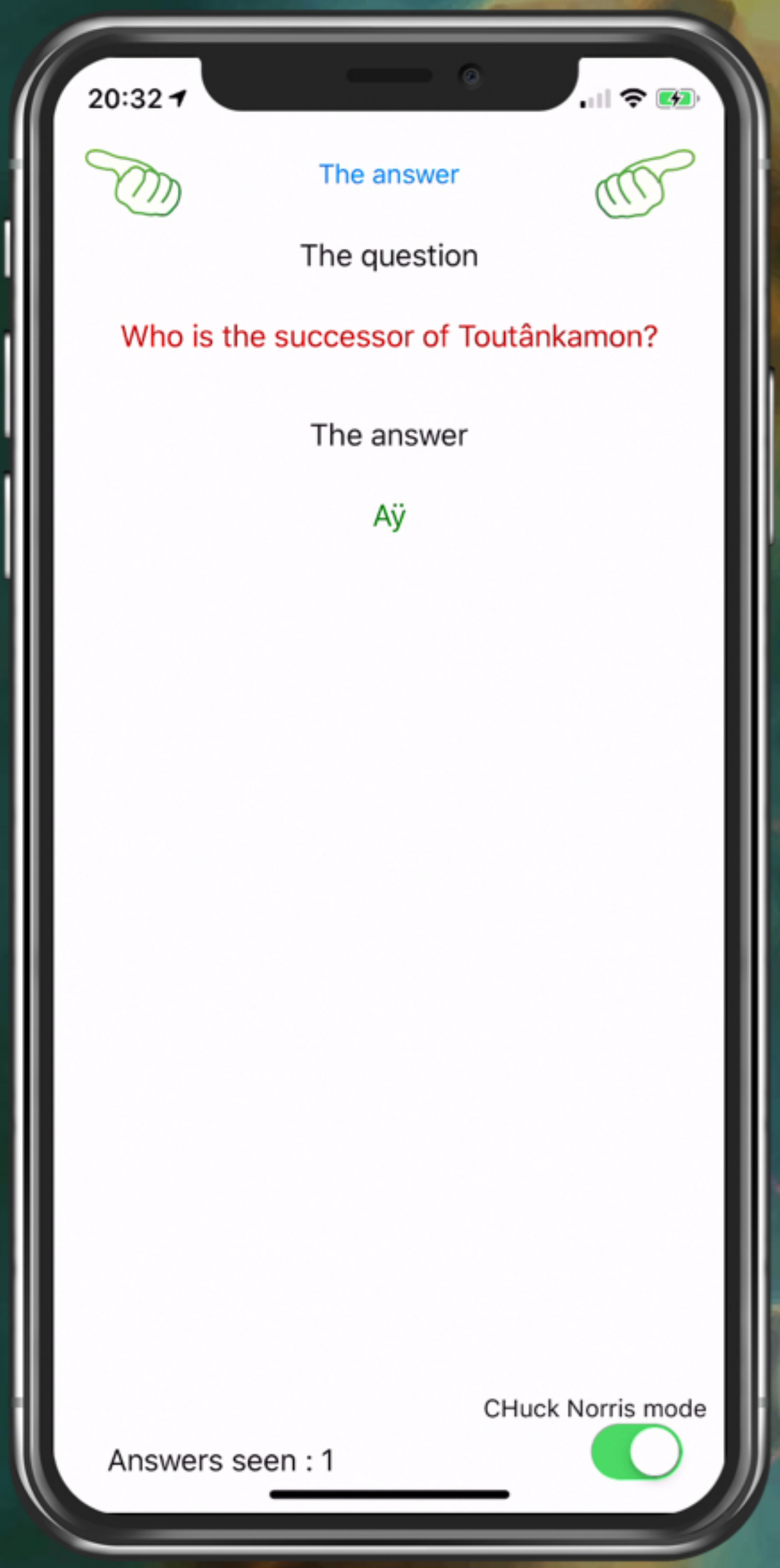
Ubiquitous in Cocoa (Objective-C & Swift)

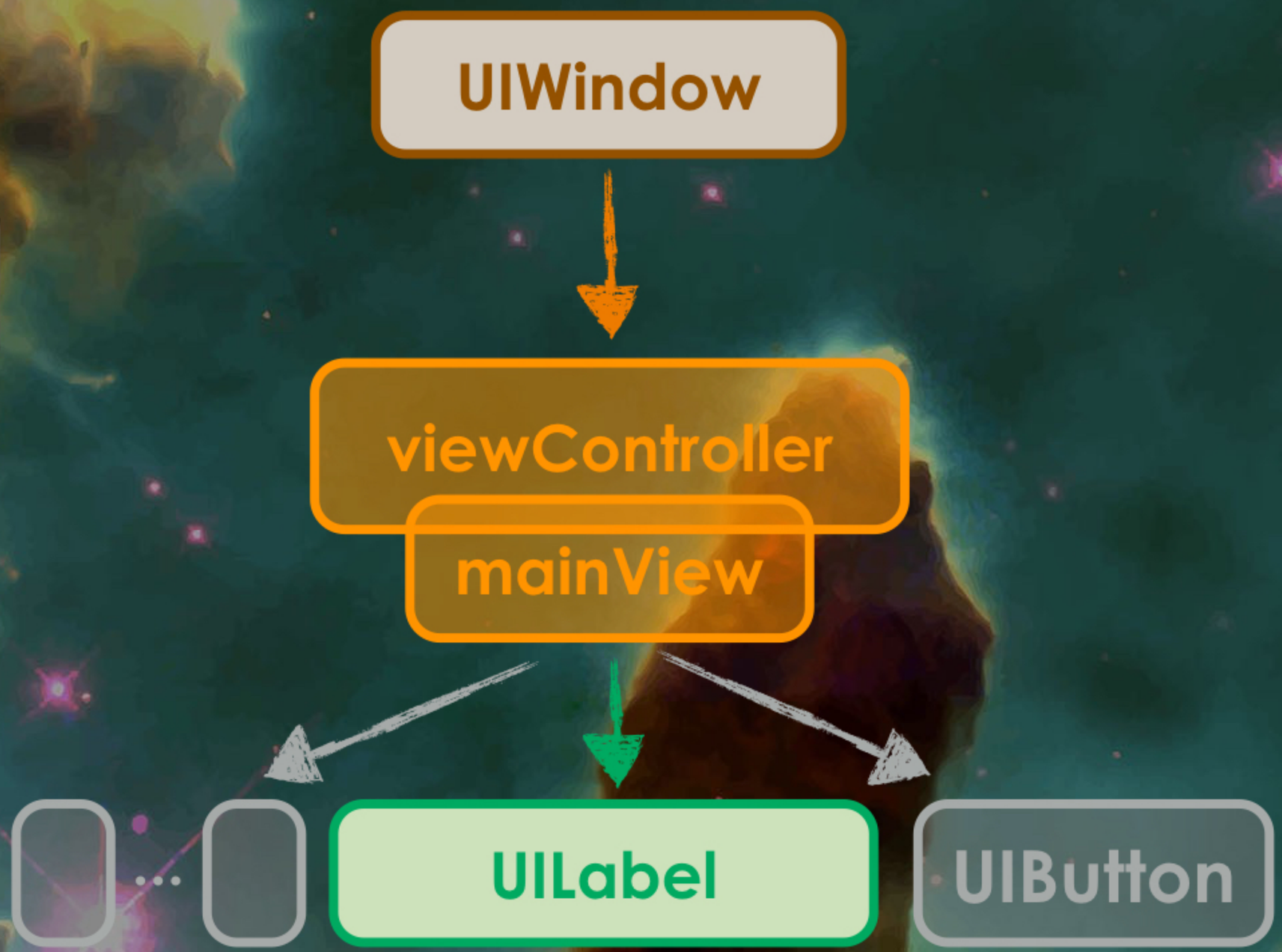
«pre-defined» class : UIViewController

Concretely

Controller = UIViewController
Vue = UIView
Model = a class

Structure of a view (example)





Life cycle of a view controller

- **init**
  - Several variants (various parameters)
- **ViewDidLoad**
  - Elaboration of the view (before it appears)
- **Managing steps of the view's life**
  - viewWillAppear / viewDidAppear
  - viewWillDisappear / viewDidDisappear
  - etc.
- **Other events**
  - didReceivedMemoryWarning
    - **To release some memory for iOS**
  - Handling orientations
    - **Detailed later**

Create a view

- **The view, a class inherited from UIView**
  - May support UIScrollViewDelegate (detailed later)
- **In the ViewController (inherited from UIViewController)**
  - Complete loadView or viewDidLoad
  - set the rootView


```
override func viewDidLoad() {
    super.viewDidLoad() // by Xcode
    let screen = UIScreen.main // The view fills the screen
    let rect = screen.bounds
    let v = MyMainView(frame: rect)
    self.view = v
}
```

```
- (void)viewDidLoad {
    [super viewDidLoad];  // by Xcode
    UIScreen *screen = [UIScreen mainScreen];
    CGRect rect = [screen bounds]; // the view fills the screen
    MyMainView *v = [[MyMainView alloc] initWithFrame:rect];
    [self setView:v];
    [v release];
}
```

As a conclusion…

Simple is’n’t it?

- You just have to deal with positions

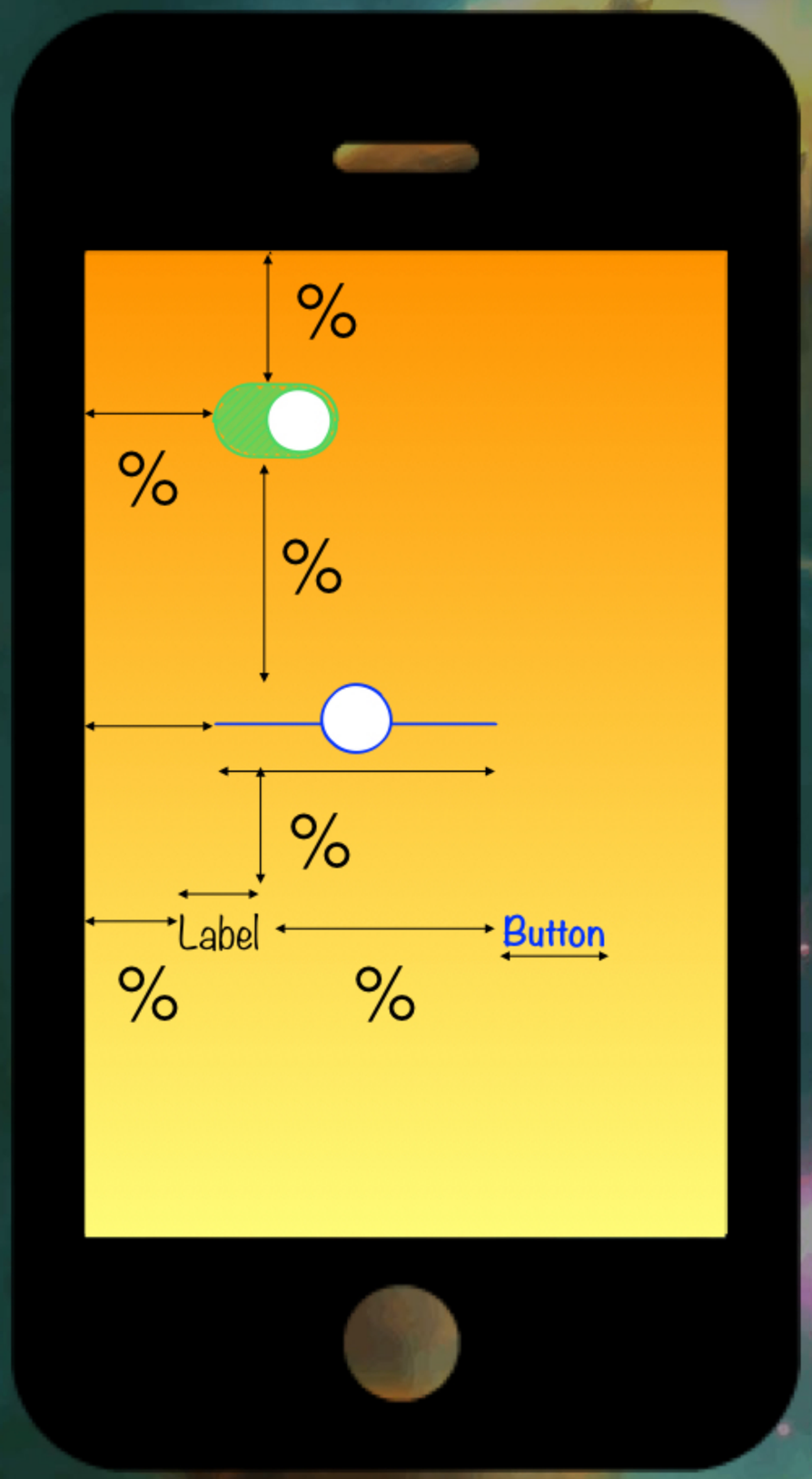
Advice

- Take your time and «draw» (by hand)
- Think «relative positions»

Remember constraints

- With AutoLayouts

One more thing…
handling coordinates



Three approaches

I compute everything

- Tedious + device handling + maintenance

Think incrementally

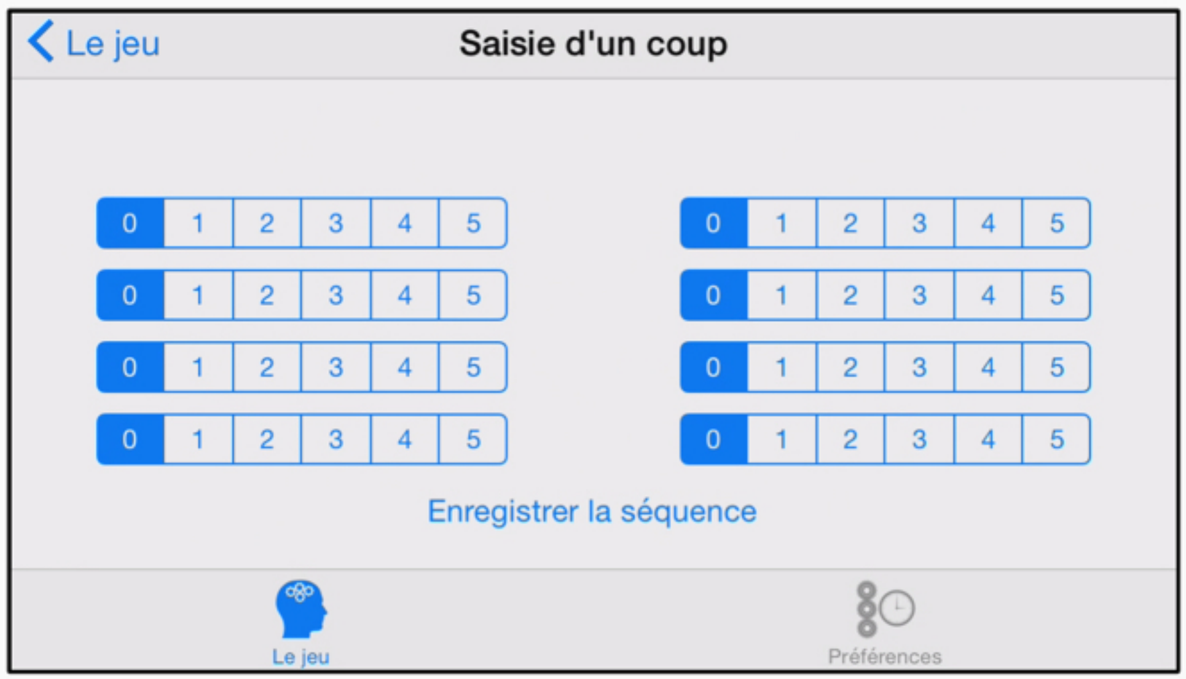
- Vertical & Horizontal
- Increment may be proportional to the screen

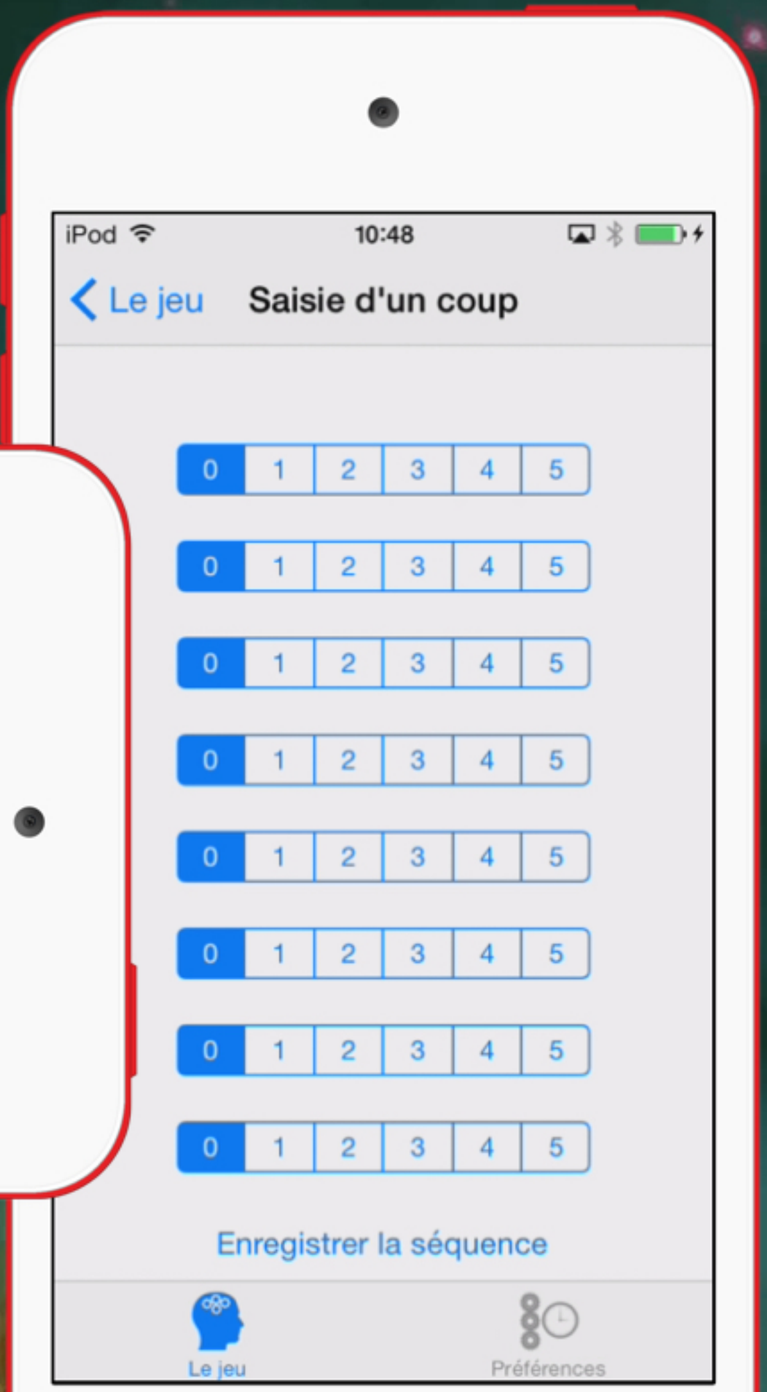
Proportional to the screen

- Ratio on height & width
- Gives flexibility (unless big changes with orientation)

One more more thing…
when it becomes complex

How do you do this?





One more more thing…
when it becomes complex

How do you do this?

what about animations?

Not presented here (RTFM 😇)