Designing Interactive Systems 2

Lecture 9: Post-Desktop Window Systems

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/dis2

RWTH AACHEN UNIVERSITY

CHAPTER 27

Post-Desktop Devices

Mobile Device Characteristics

- Compact screen size
- An app has one screen at a time
- Minimal onscreen help
- Restrictive memory management
- Use case specific hardware
- Efficiency is crucial for good battery life
- Context is key

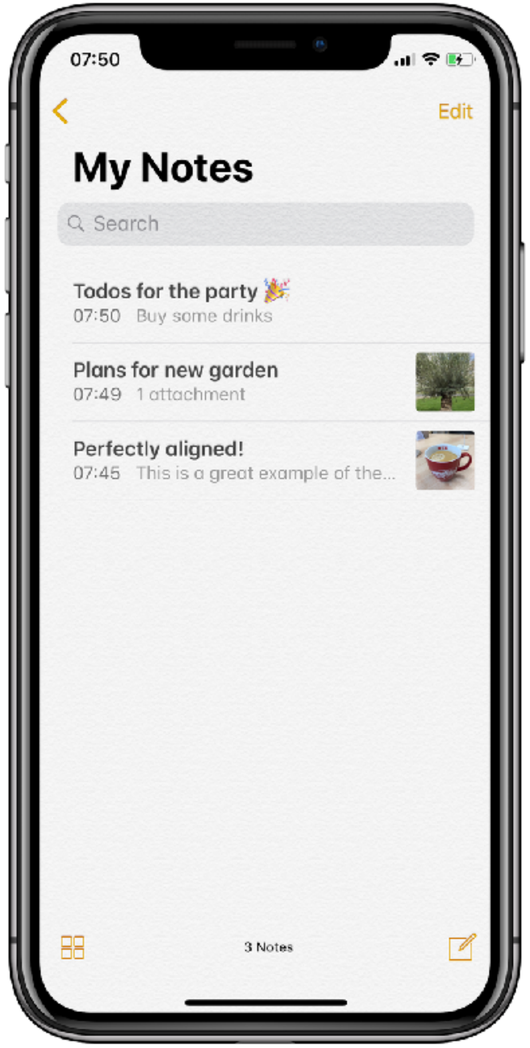
Context is Key

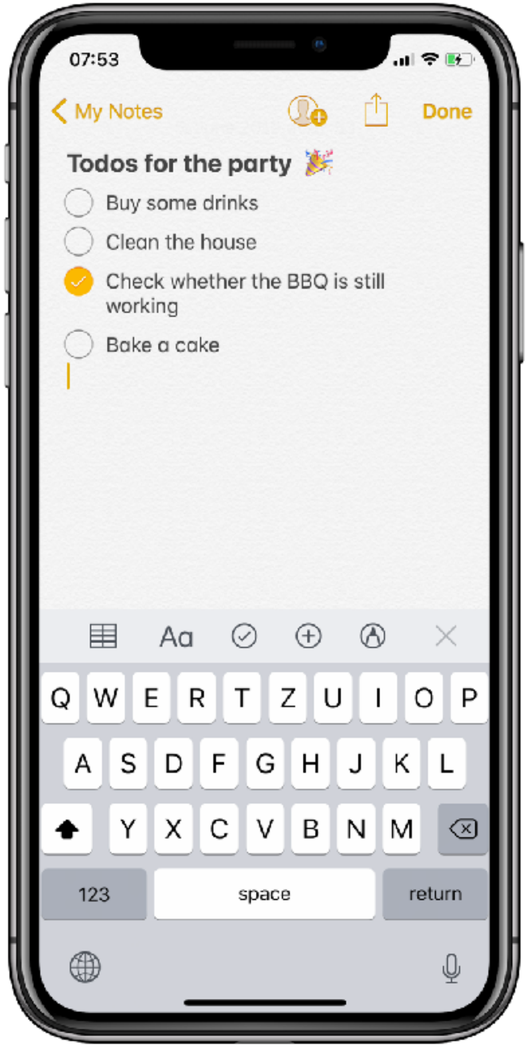
- It’s unlikely to be hit by a bus while working on your desktop
- Or, to fall into a fountain...

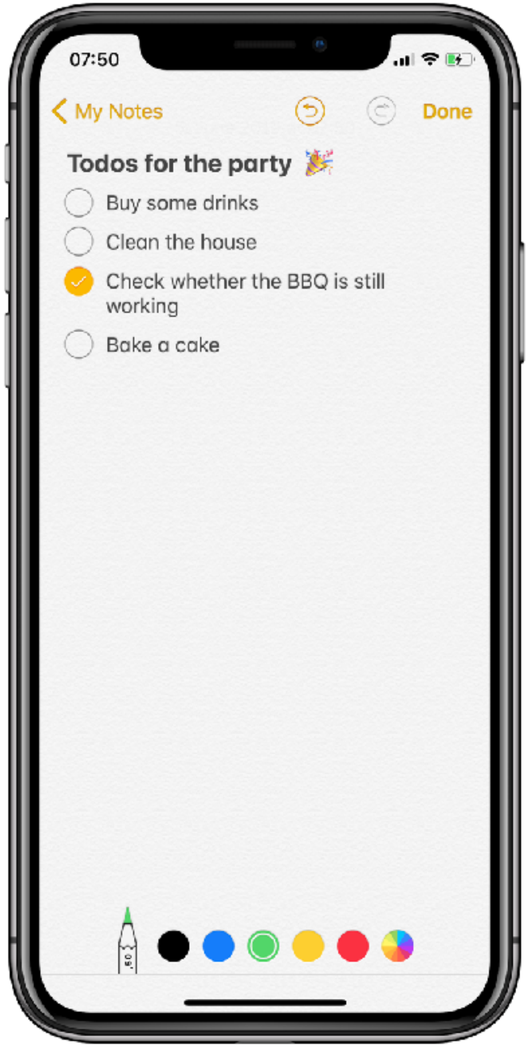
10 Golden Rules of Interface Design (see DIS 1)

1. Keep the interface simple!
2. Speak the user's language!
3. Be consistent and predictable!
4. Provide feedback!
5. Minimize memory load!
6. Avoid errors, help to recover, offer undo!
7. Design clear exits and closed dialogs!
8. Include help and documentation!
9. Offer shortcuts for experts!
10. Hire a graphics designer!

A Typical Mobile App







Life as an App

- A mobile OS is an **app-centric environment**
- **One app per task**
  Hence, do one thing but do it well
- **Sandboxing**
  Data is typically stored per app and not visible to others
- Data exchange between apps difficult
  Use integrated Files app to prevent duplication of data in the 2nd app
- Define the task that users want to accomplish in your app

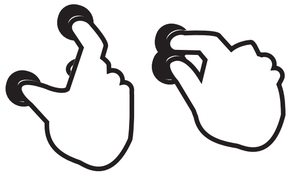
Designing the UI

- Make it obvious how to use your app
- Sort information from top to bottom
- Use alignment to ease scanning and communicate groupings
- Minimize text input
- Be prepared for changes in text size
- Provide fingertip-size targets

Interaction Design

- There are no on-screen signifiers how and where to perform multitouch gestures

- On some targets pressure input possible



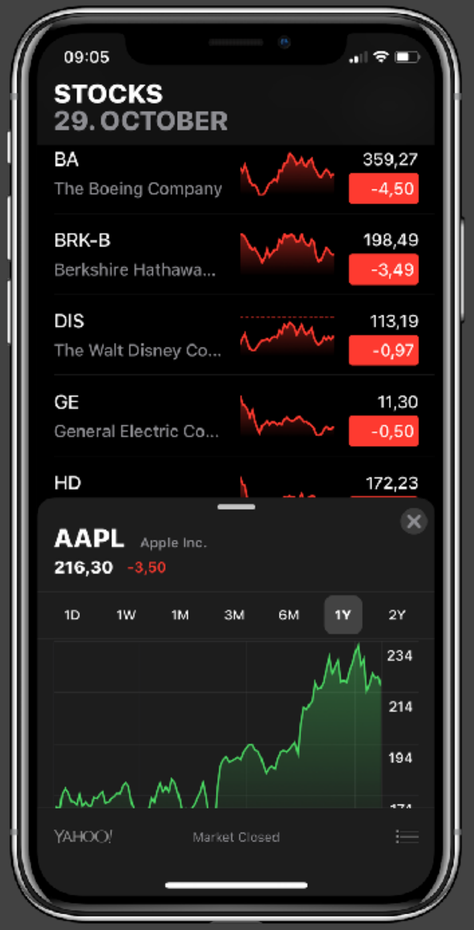
- Interaction patterns vary between different platforms: Follow the respective guidelines for intuitive operation
- If you use complex gestures, help the user

CHAPTER 28

Mobile Application Styles

Utility Applications

The Elements

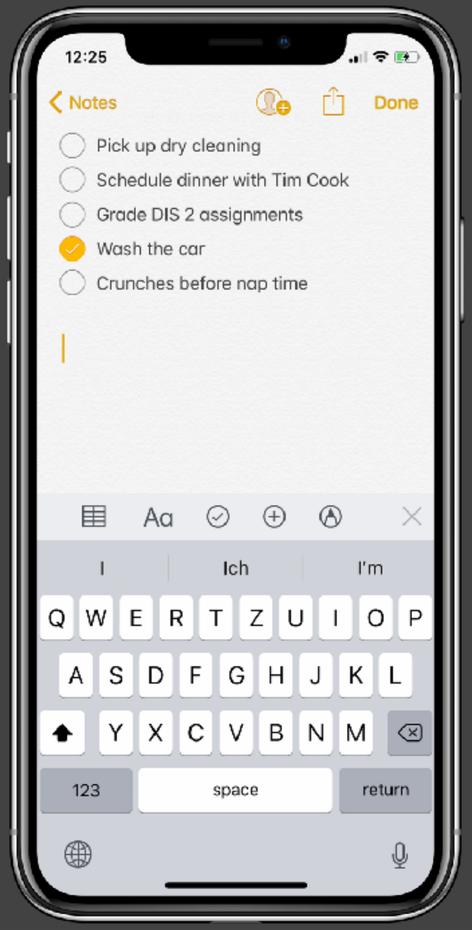


Stocks

Compass

Productivity Applications

Calendar



Notes

Microsoft Excel

Immersive Applications

Google Earth

Pokémon GO

Monument Valley 2

CHAPTER 29

iOS

iOS History: Breakthroughs in Interaction

2007
Multitouch

2010
iPad

2014
Apple
Watch

2015
3D Touch

2017
ARKit

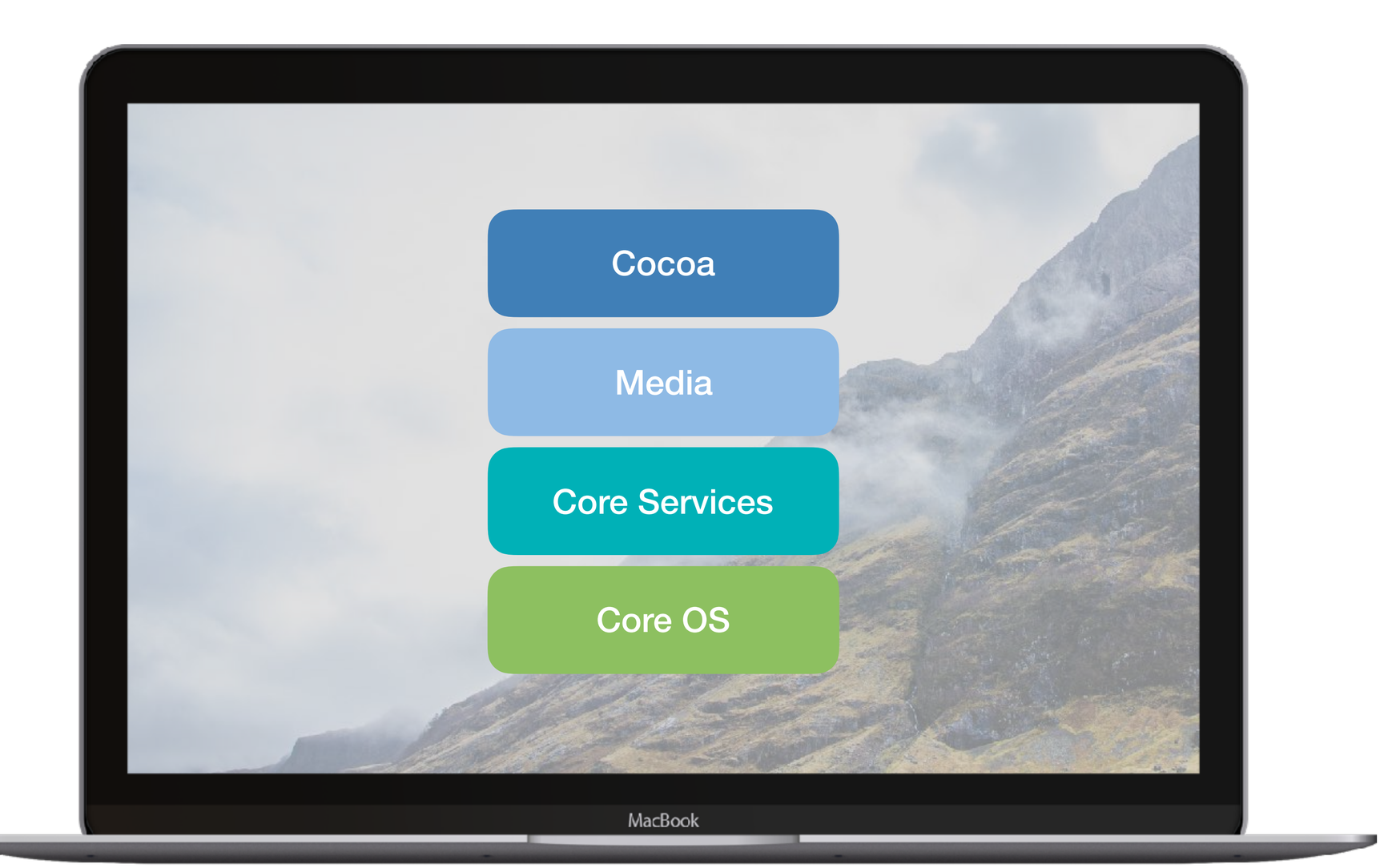
2008
App Store

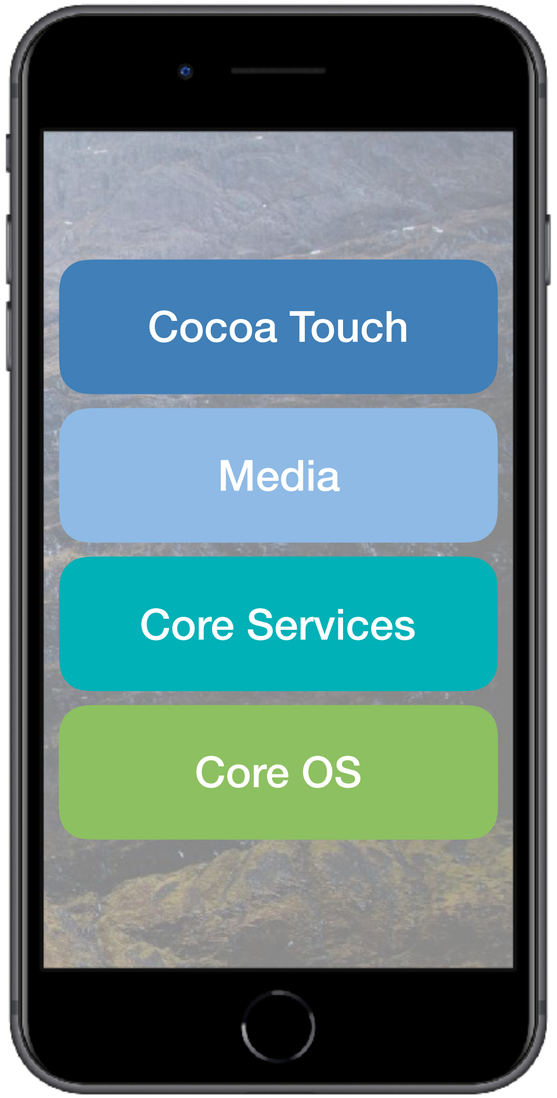
2011
Siri

2015
iPad Pro & Pencil
Split-Screen Mode

2019
iPadOS

iOS Architecture





iPhone Development

- **From AppKit to UIKit**
  - Different event handling:
    Views support multiple input events and have no hover menus
  - No main menu for applications
  - More modern framework, e.g. target-action is no longer 1:1 but 1:n
  - Originally only RGB color space

- **Changes in Foundation**
  - No Cocoa bindings
  - No distributed objects

- **New Frameworks**
  to interact with phone hardware

iOS: Handling Touch Input

- Override **touch methods**

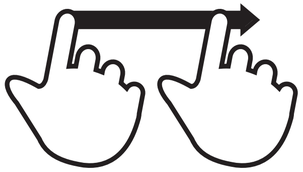
```
override func touchesBegan(_ touches: Set<UITouch>,
                           with event: UIEvent?) {
    if let touch = touches.first {
        print(touch.location(in: self))
    }
}
```

- Use a **gesture recognizer**

```
self.recognizer = UIPinchGestureRecognizer(target: self,
                        action: #selector(action(_:)))
self.view.addGestureRecognizer(recognizer)
```

Gestures

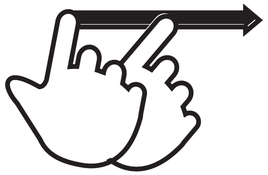
Tap



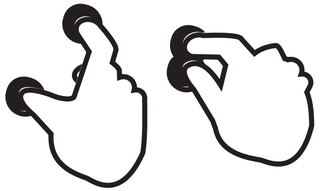
Drag

Pinch open

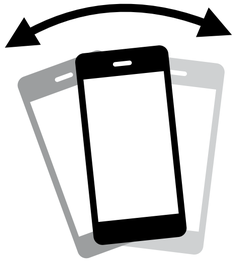
Double tap



Flick

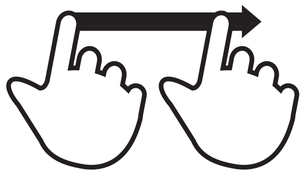


Pinch close

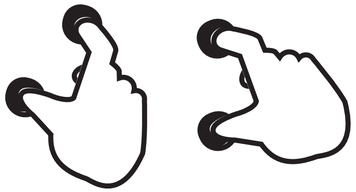


Shake

Touch and hold



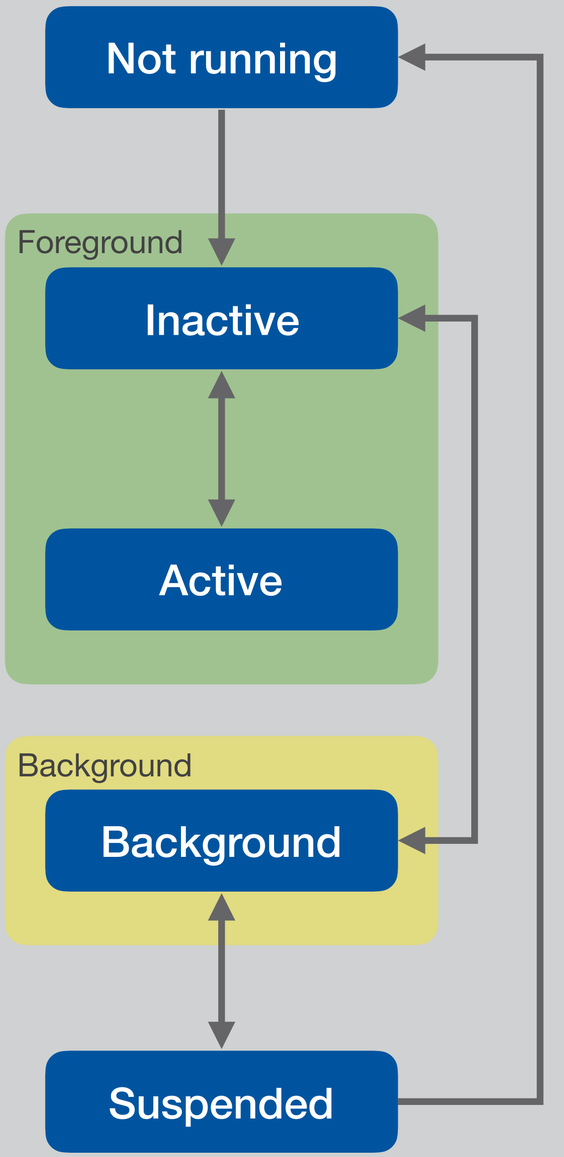
Swipe



Rotate

iOS App Lifecycle

- Lifecycle delegate methods allow the app to react to state changes
- Implicit termination by user only possible in app switcher (and by the way: it is not required)
- The OS decides when to terminate an app, mostly depending on the memory footprint, and the app might not be aware of the termination

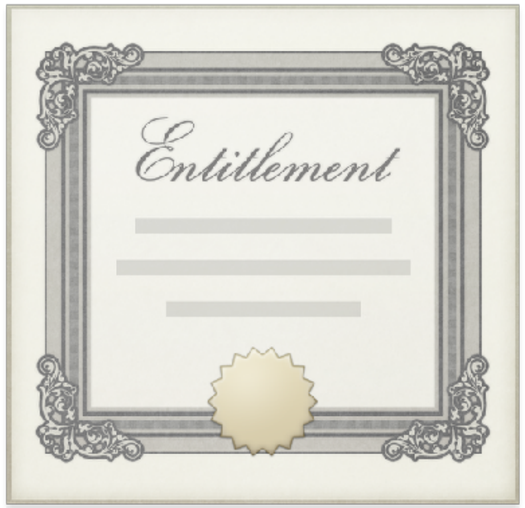


iOS Apps: Components

Info.plist

AppDelegate

View Controllers

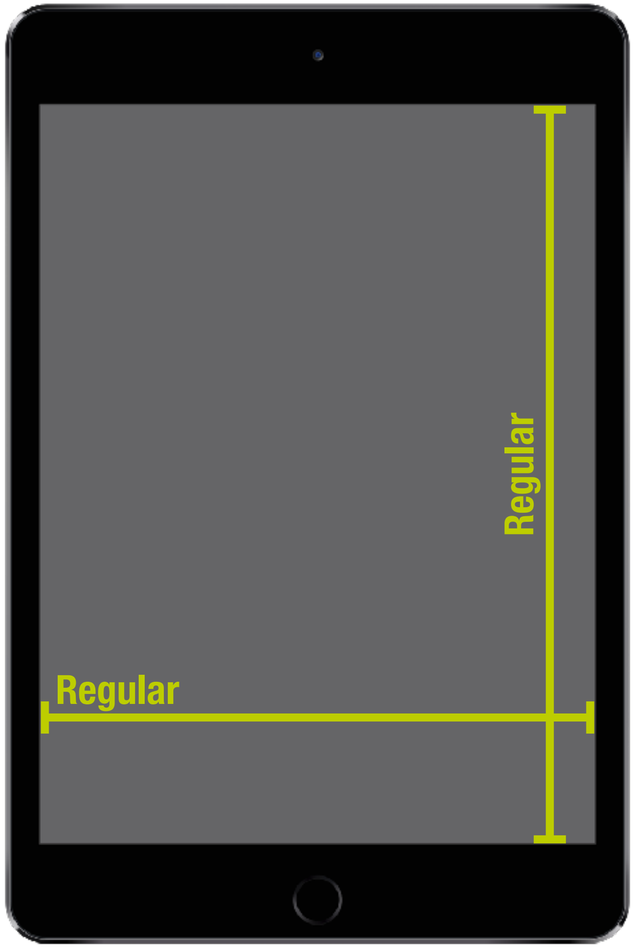


Entitlement

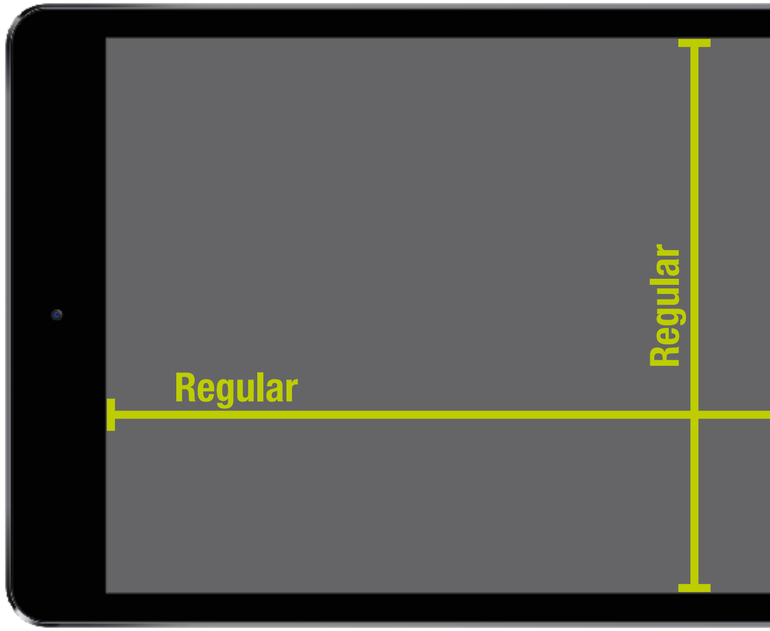
Adapting to Different Devices

- How to create an interface that works well on different aspect ratios?
- Storing pixel coordinates is not flexible, requires values for every different screen size
- Instead describe the relationships of the UI widgets to each other
- Key addition:
  Declare variations in UI depending on **size classes**

iOS: Trait Variations



iOS: Trait Variations



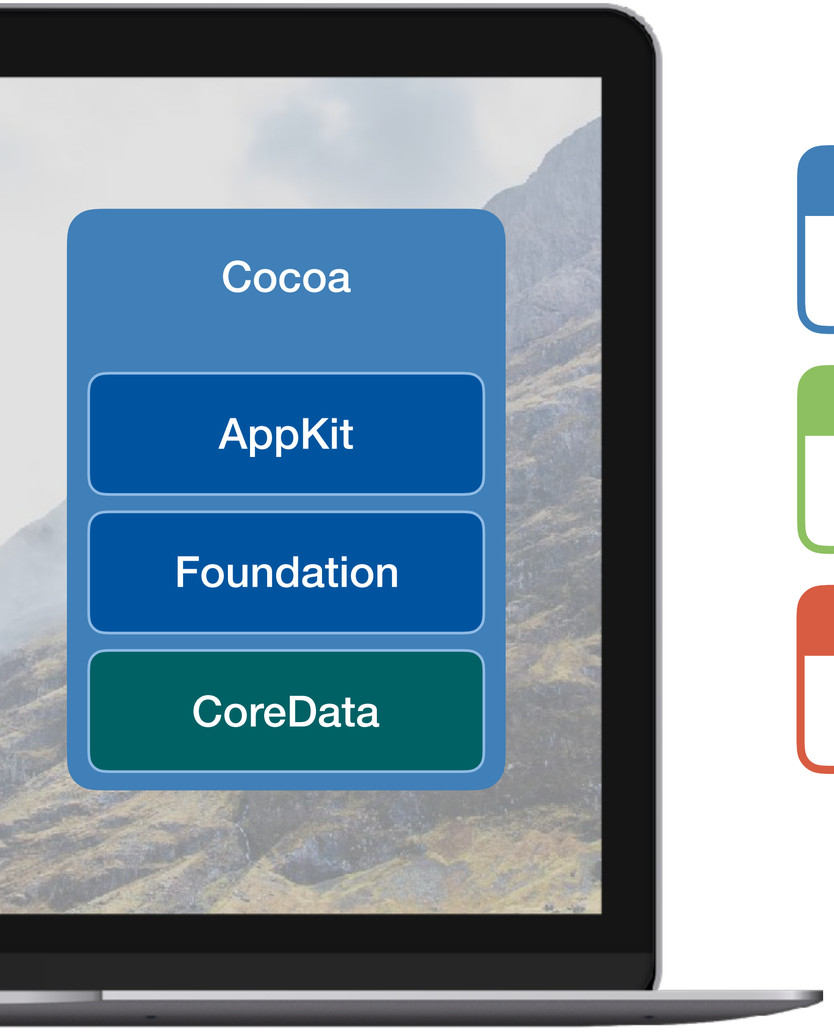
Some iOS SDKs

App Frameworks	 App Extensions	 Handoff	 Multitasking	 Notifications
Media & Web	 AirPlay	 MapKit	 MusicKit	 WebKit
App Services	 CloudKit	 HealthKit	 Machine Learning	 SiriKit
Graphics & Games	 GameCenter	 Metal	 SceneKit	 SpriteKit

iOS Design Themes

- **Clarity**
  - Subtle decorations
  - Sharpened focus on functionality
  - Use negative space, colors, fonts to highlight important content
  - Direct manipulation to support understanding
- **Deference**
  - UI does not compete with content
  - Content fills entire screen
  - Fluid motion
  - Typically no bezels
  - Translucency hints at more content
- **Depth**
  - Distinct visual layers
  - Realistic motion
  - Navigational transitions provide sense of depth

iOS Architecture



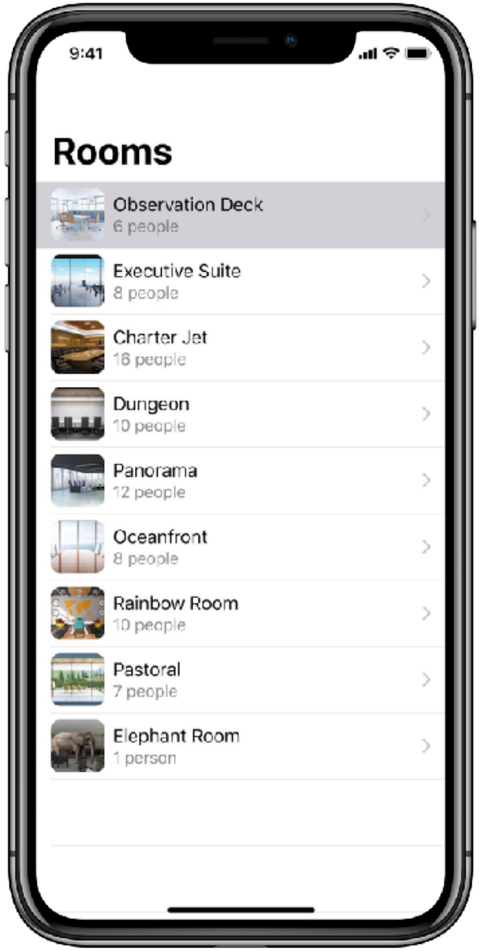
View	View
AppKit	UIKit
Controller	**Controller**
AppKit	UIKit
Model	
Foundation, CoreData	

SwiftUI

- Unified framework across all Apple platforms
- Declarative instead of imperative code
  - For many things the framework decides how to display them
  - But, hence, customization can sometimes be limited
- Results in platform-specific native interfaces

SwiftUI: Teaser

```
struct ContentView : View {
    var rooms: [Room] = []

    var body: some View {
        NavigationView {
                List(rooms) { room in
                    NavigationButton(destination: DetailView(room)) {
                        Image(room.thumbnailName)
                            .cornerRadius(8)

                        VStack(alignment: .leading) {
                            Text(room.name)
                            Text("\(room.capacity) people")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                    }
                }
                .navigationBarTitle(Text("Rooms"))
        }
    }
}
```



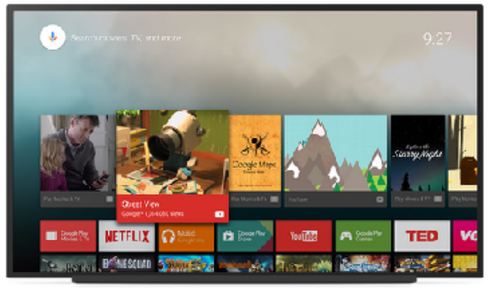
CHAPTER 30

Android

Android History

2007
Roots of Android

2009
Android Market



2014
Support for TVs and watches

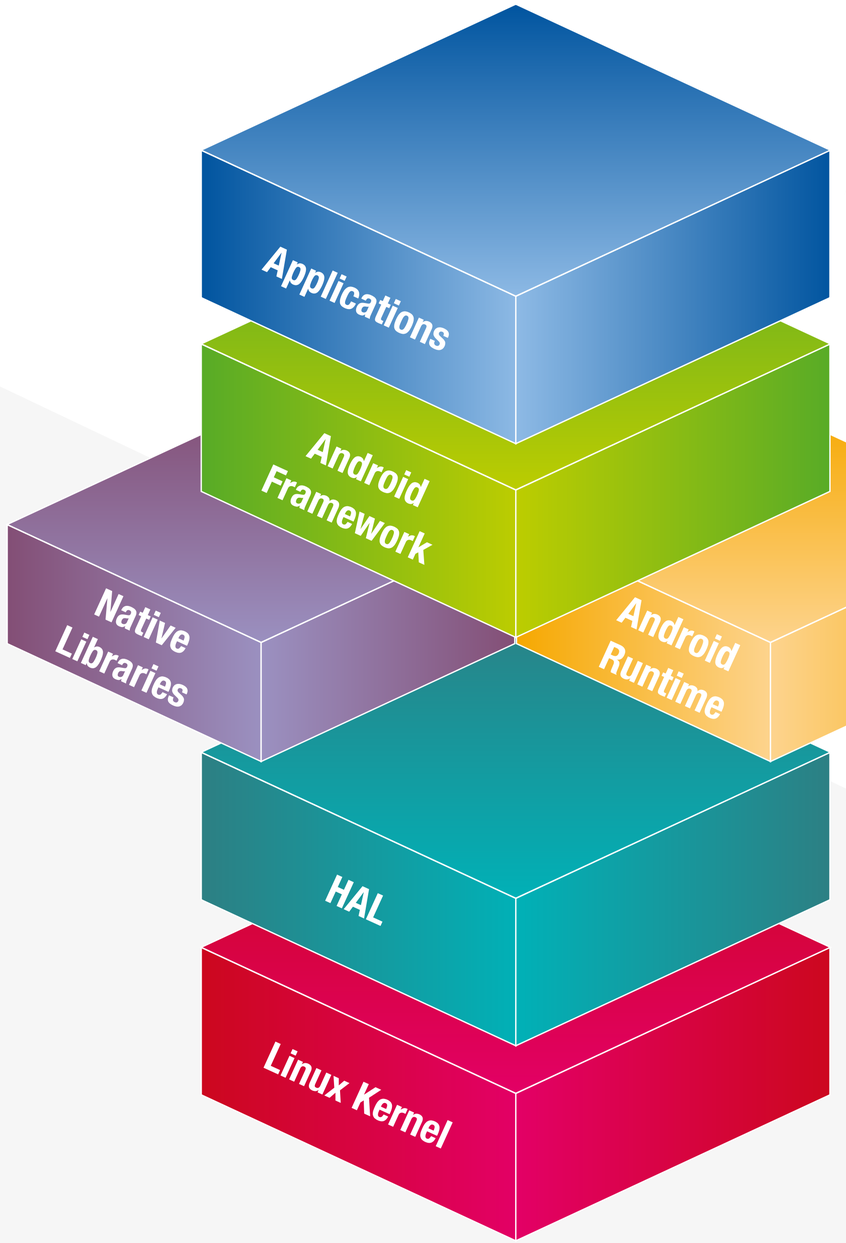
2008
First Android device

2011
Support for tablets

2013
“OK Google”

2015
Fingerprint gestures

Android Stack



Alarm, Browser, Calendar, Photo Album, SMS, ...

Content Providers, Activities, Resources, Telephony

Audio Manager, Freetype, OpenGL, SQLite, SSL, WebKit

ART, Core Libraries

Audio, Bluetooth, Camera, Storage, Graphics, Sensors

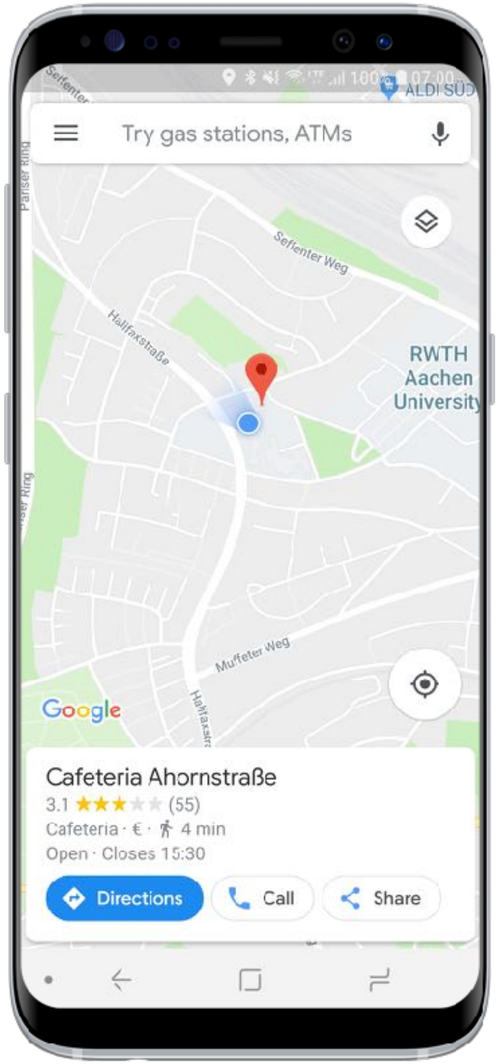
Drivers, Shared Memory, Power Management, USB, WiFi

Application Fundamentals

- Idea: **Share** elements of applications
  - No single entry point
- All of the four different **types of component**s are entry points for the system or user
  - Activities
  - Services
  - Broadcast receivers
  - Content providers

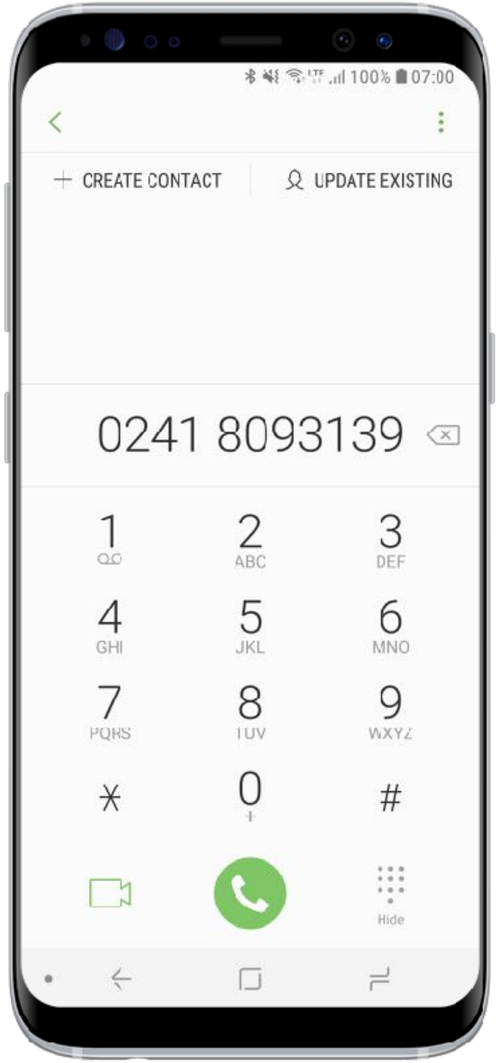
Activity

- Single screen of your application’s UI
  - Contains a tree of views
  - Defines menus
- Starts & stops services
- Calls other activities via intents



Intent

- Messaging object to request an action from another app (component)
- **Explicit intent**
  - Open another activity in the same app
- **Implicit intent**
  - Requesting an abstract “service”
  - Caller does not know callee
- Intent filters expose functionalities to other components



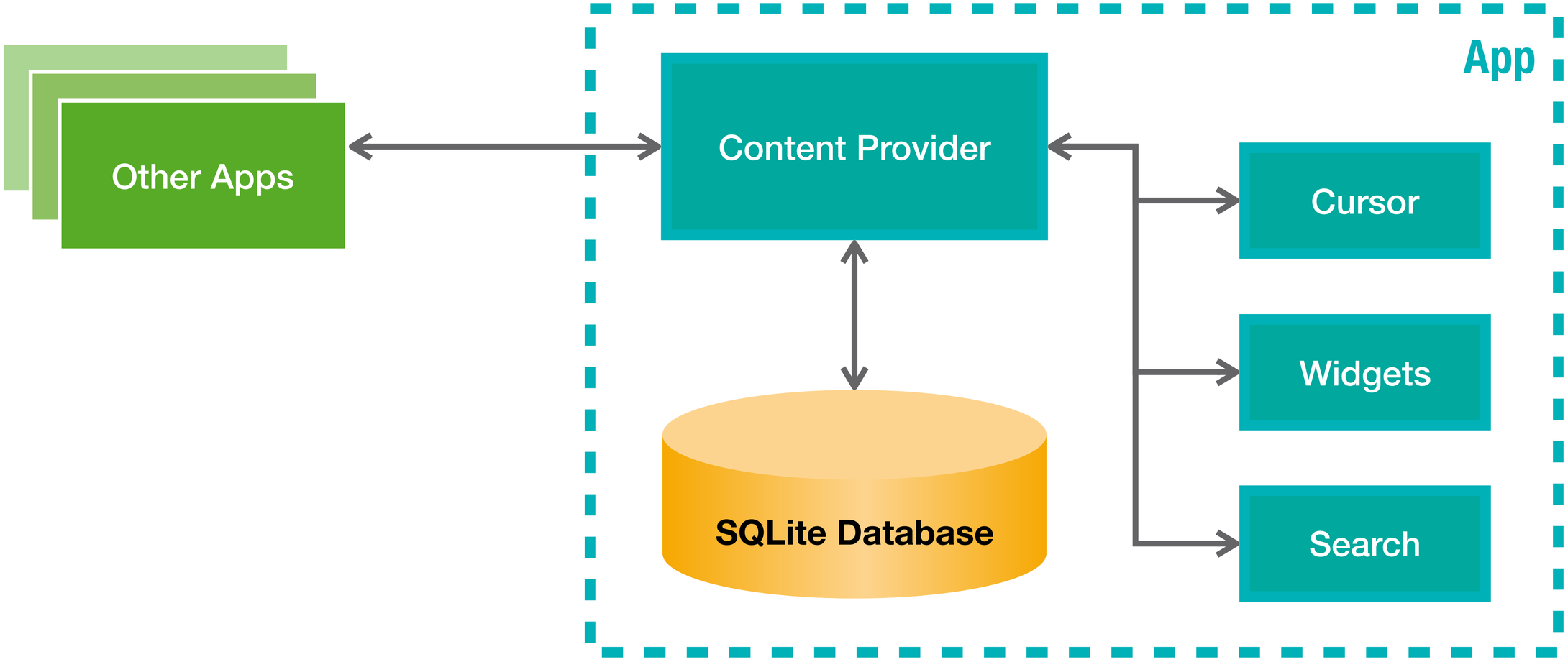
Broadcast Receiver

- Broadcasts are implicit intents
  - e.g. for system events like timezone change, device shutdown, ...
- Broadcast receivers are used to register from system or application events
- Use a **dynamic** broadcast receiver
  to make your app react to changes during runtime
- Use a **static** broadcast receivers
  to start your app on a specific broadcast

Service

- Long-running operation in the background and does not provide a UI
  - e.g., network transactions, play music, perform file I/O
- **Unbound service**
  - Is kept alive by the system even if the starting Activity has finished executing
- **Bounded service**
  - Components can bind to a service and interact with it through an interface exposed by the Service
  - Client Server architecture
  - When the last client unbinds from the service, the system destroys it

Content Provider



Android Manifest

- XML file that defines a black box view of an application
- Interface between the OS and the app
  - Icon
  - Requirements (e.g., minimum API level)
  - Permissions (e.g., making calls)
- Exposes app’s functionality
  - Available activities
  - Intent filters (e.g., entry point)

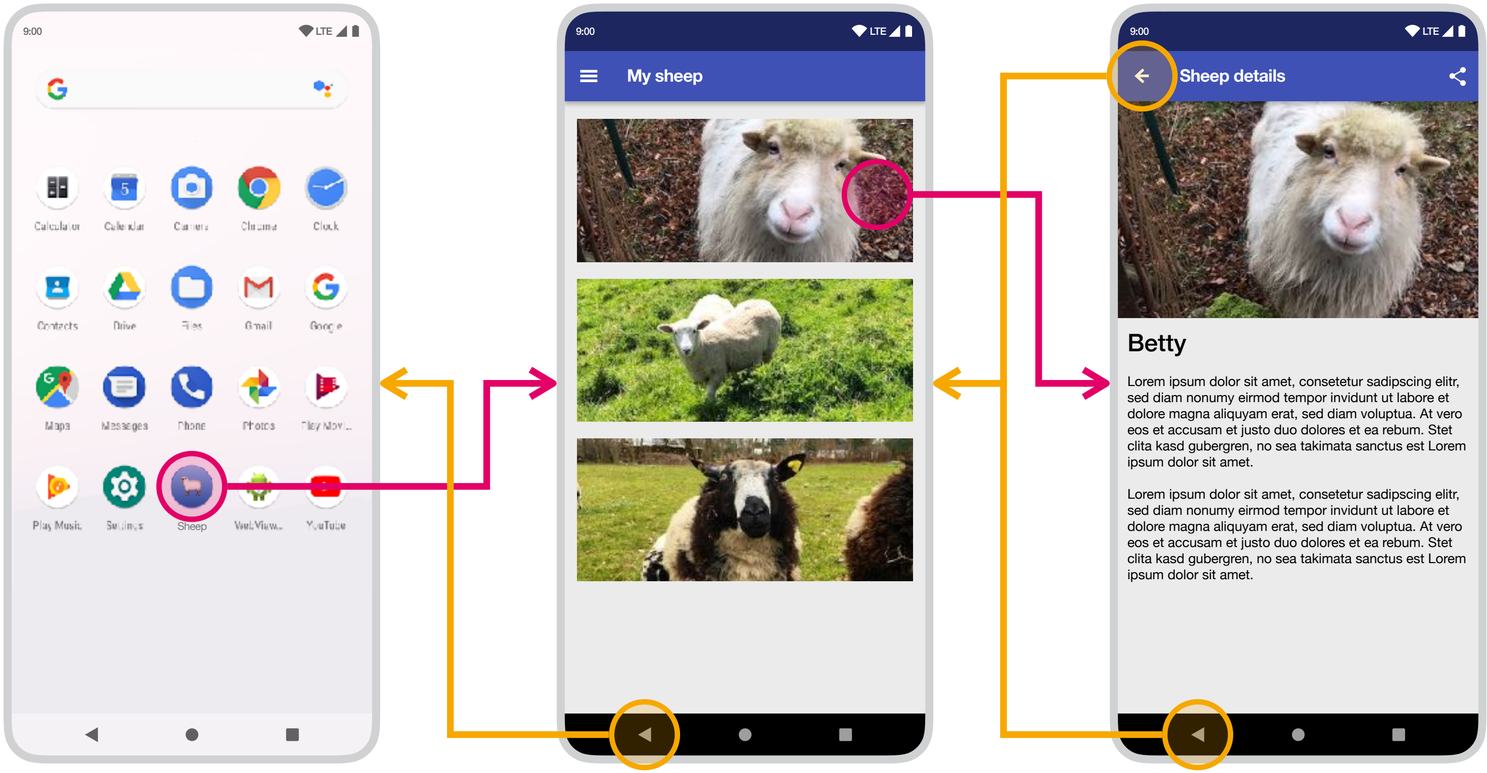
Tasks and Multitasking

- **Tasks** are a sequence of activities (possibly from different apps)

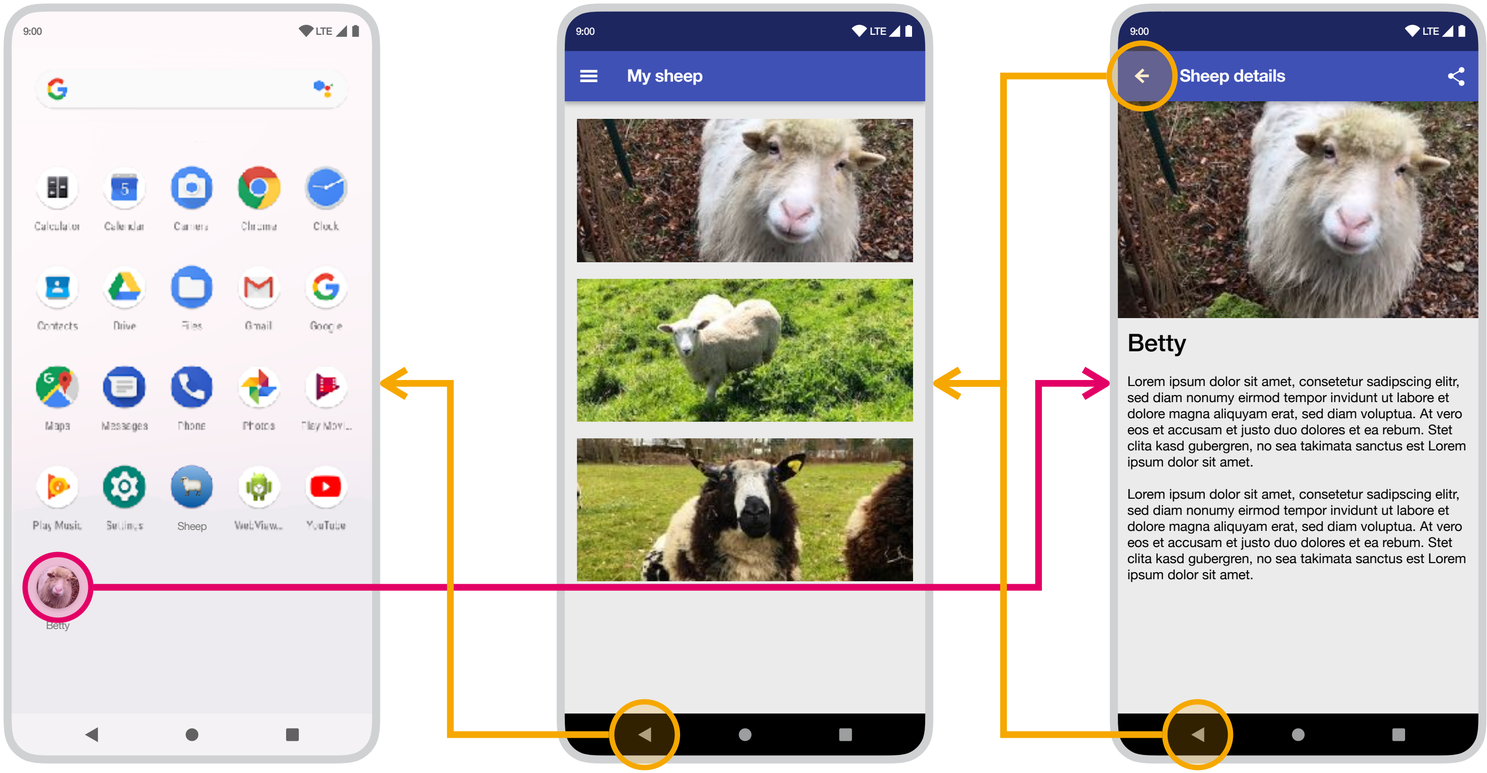
- Every time a new activity is started, the previous one is moved to the task's **back stack**
  - The same activity can be instantiated multiple times in one back stack
  - The back button switches to the previous activity in the stack

- The home button signals that the user switches to a new task

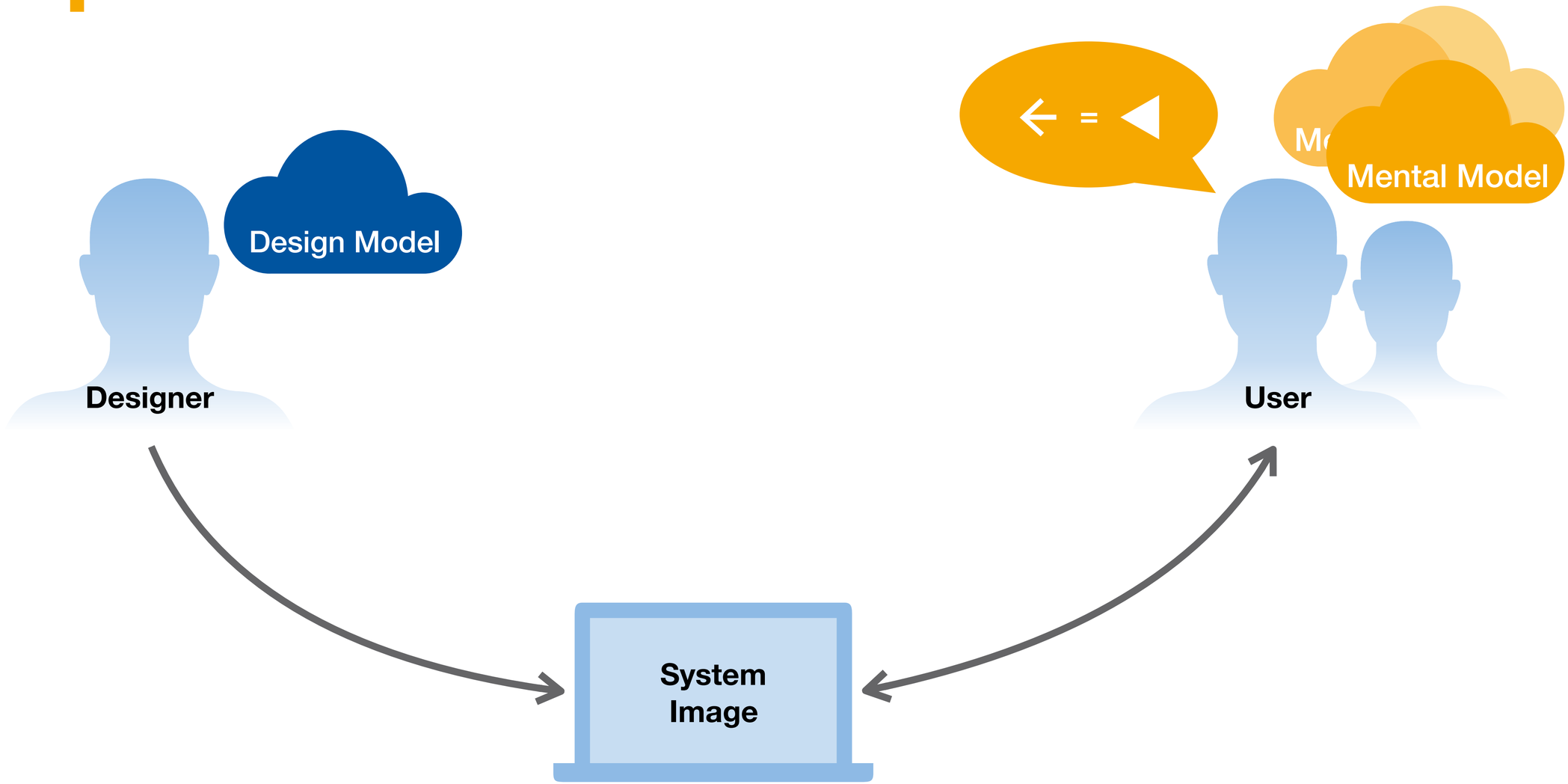
Back Navigation



Deep Links



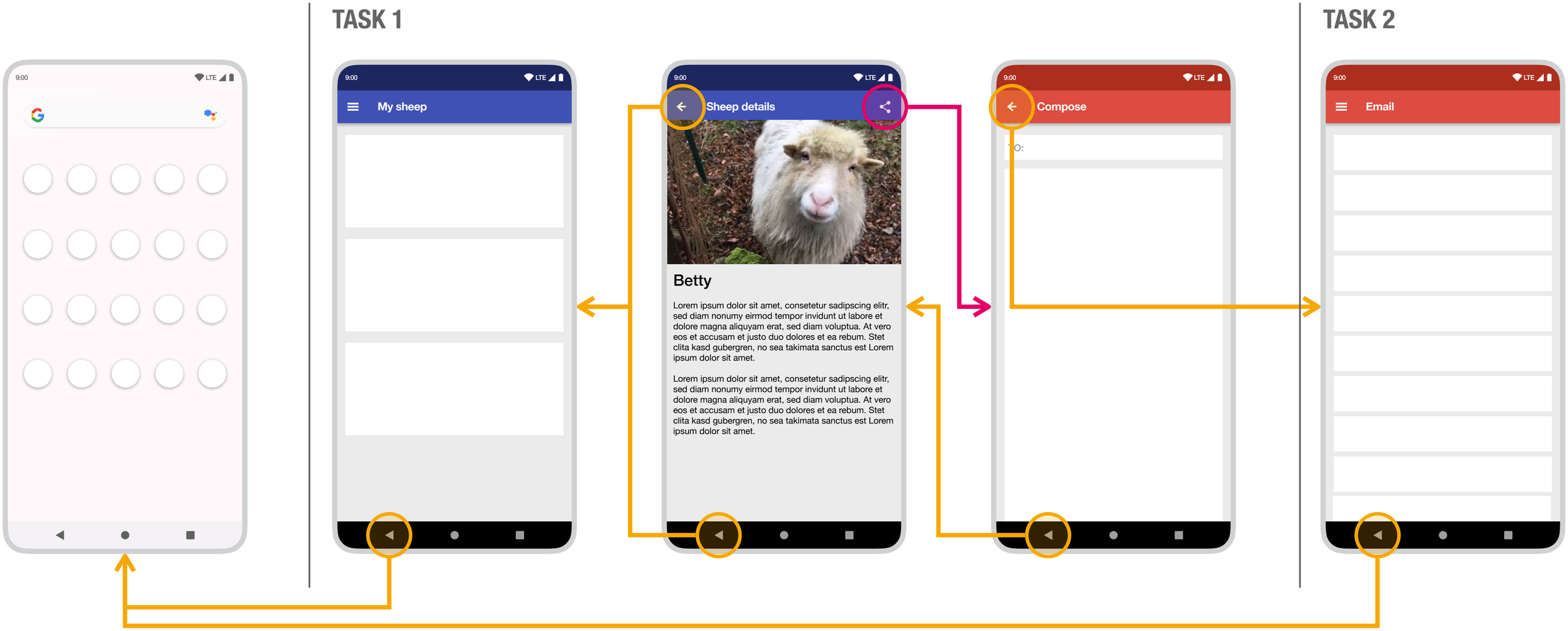
Conceptual Models



The Up Button

- The on-screen up button navigates to a parent activity that is statically defined by the developer
- It never exits the app and hence does not exist on root activity
- For tasks that remain in one app, up and back behave identically
- Pressing the up button creates a new task if the current activity was presented from an activity of a different app
- The up button cannot be used to navigate between sibling contents, e.g. paged contents inside of an activity

Switching Tasks



CHAPTER 31

Designing for the Big Screen

Design Themes for the TV

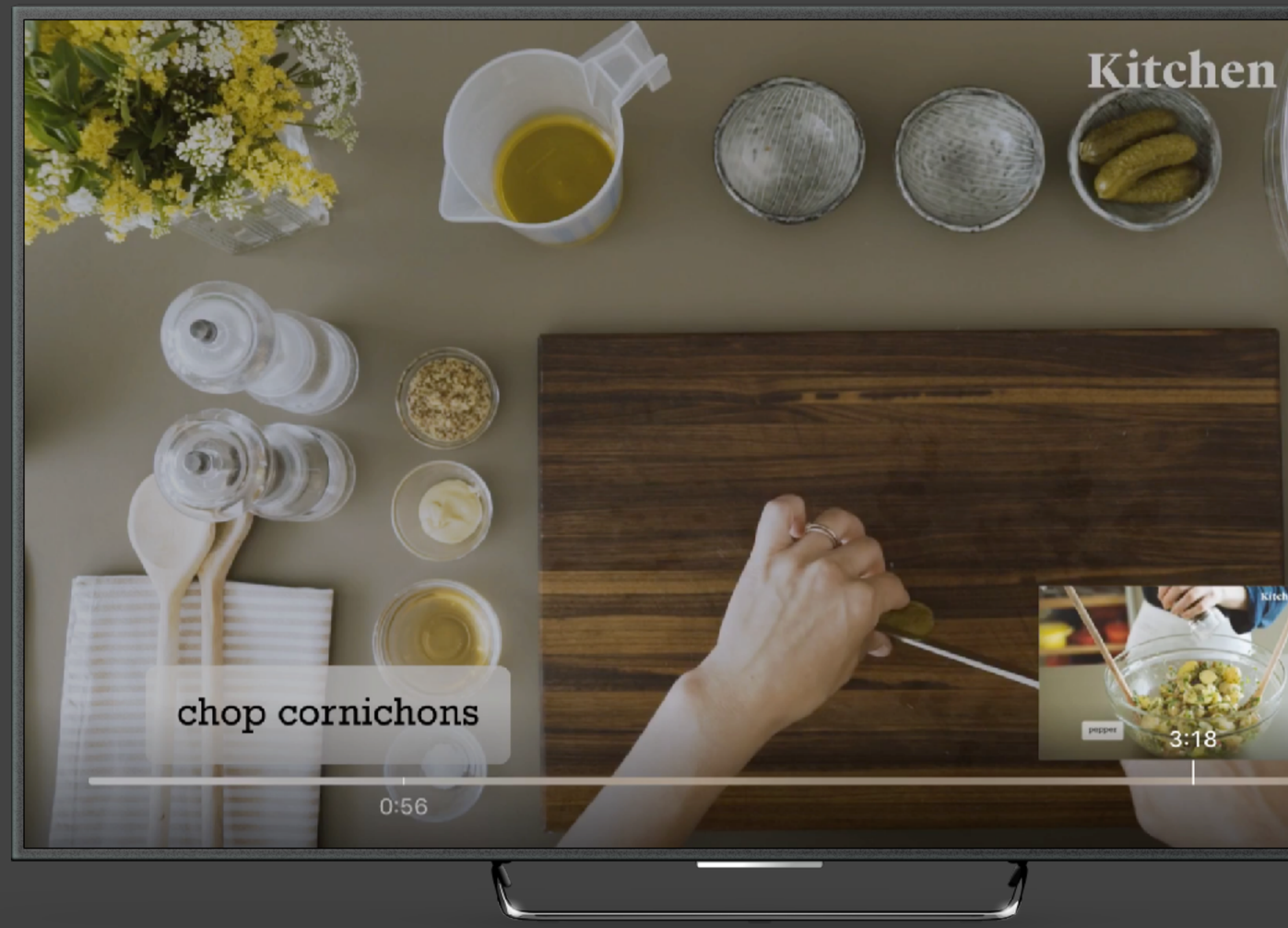
- **Immersive**
  - Engulf people in a cinematic experience
  - Exploit the massive canvas with edge-to-edge scenery
  - Use fluid animations, captivating audio and vibrant colors

Design Themes for the TV

- **Clear**
  - Use consistent layouts
  - Make the focus clear and unmistakable, even from at distance
  - Movement across space is consistent and predictable

Design Themes for the TV

- **Across the Room**
  - Users sit a few meters away from the screen
  - Resolution and viewing distance make it difficult to process too much information
  - “Connect” the user with the content

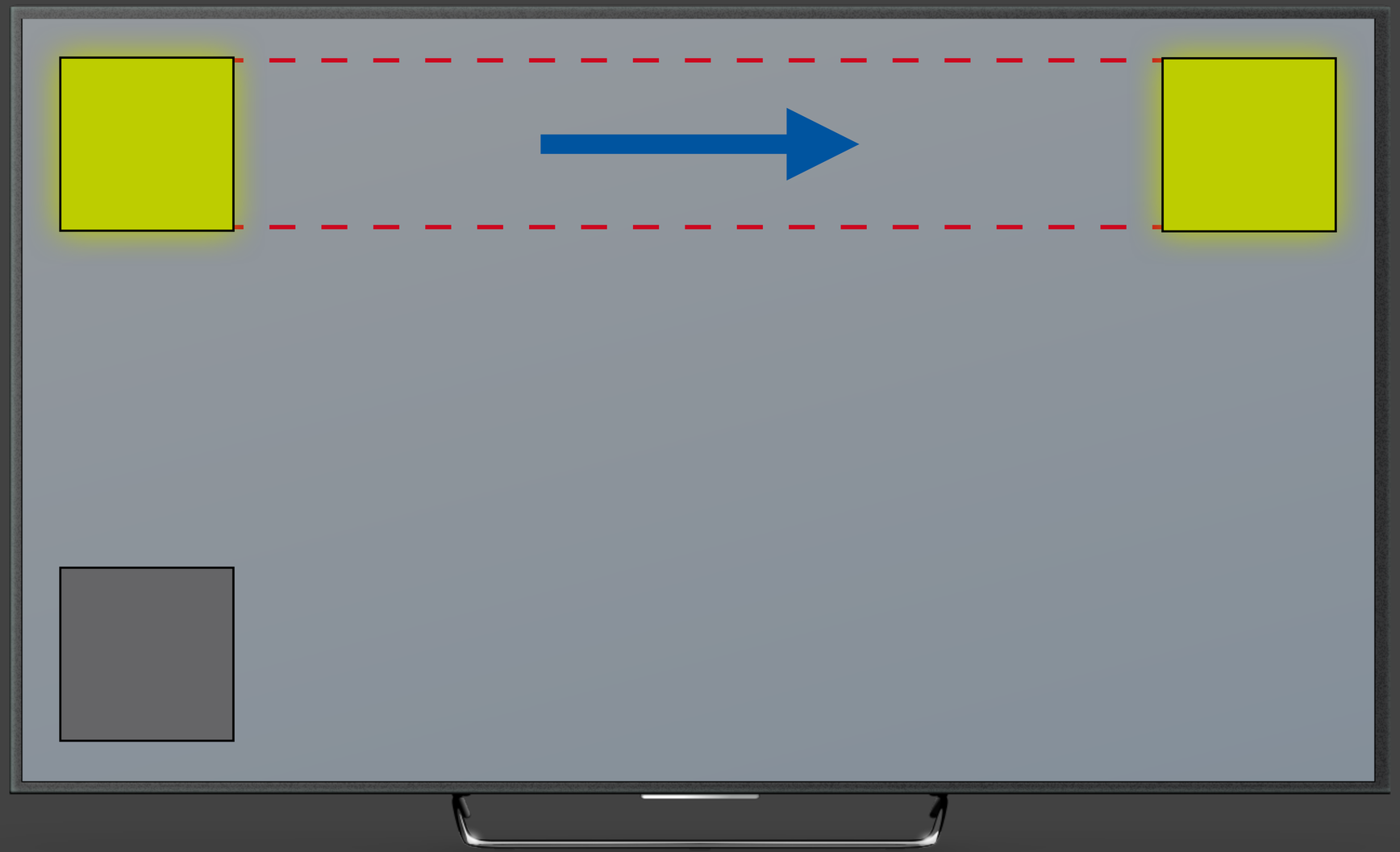


Input?

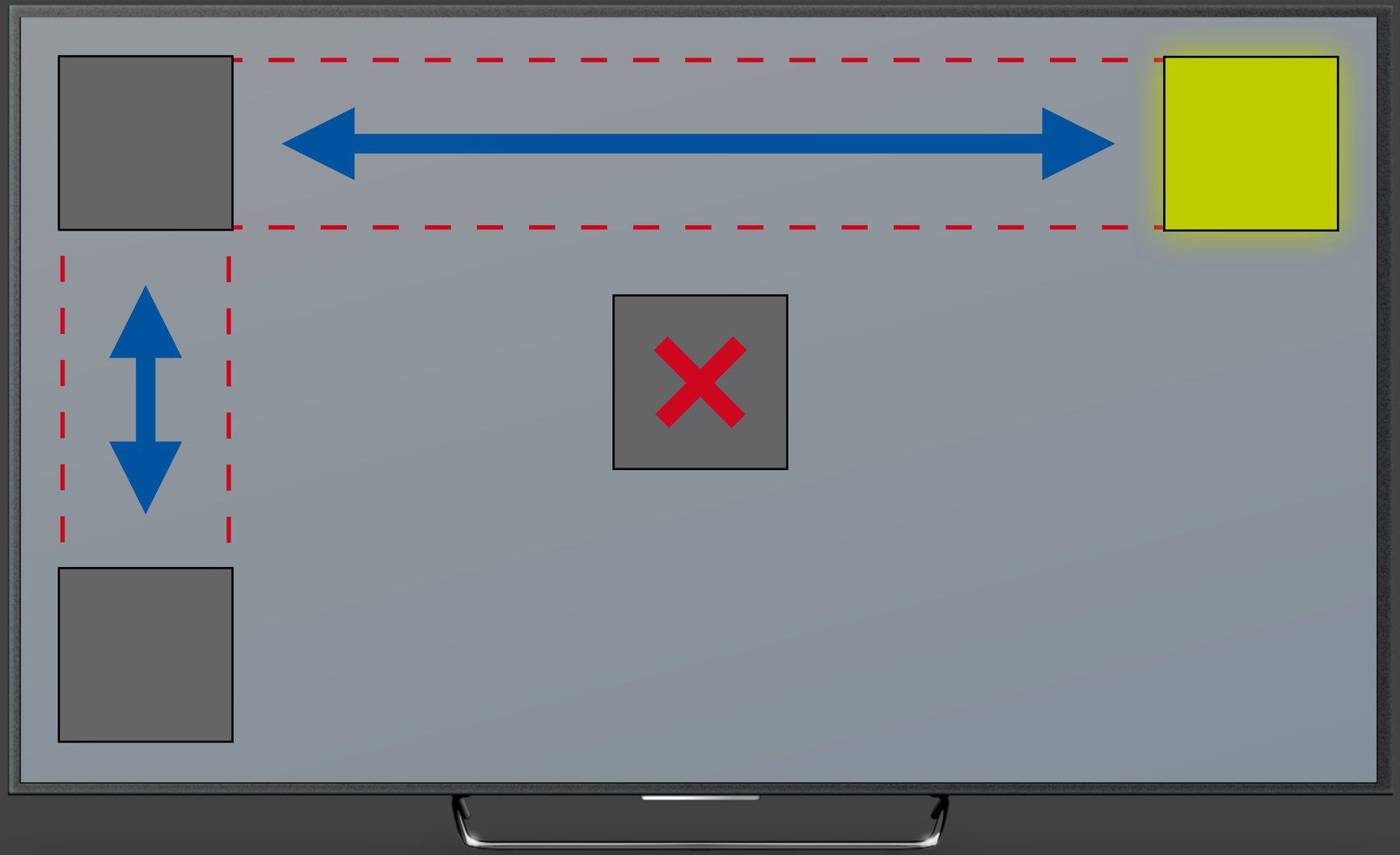
Focus Model

- Interaction with TV UIs is based on a focus model
  - Always one element highlighted
  - tvOS: Parallax effect makes focused items more responsive to user input
- Always move focus in the expected direction
  - Focus moves in the direction of the gesture
    Content might move in the opposite direction of the focus
  - (Fullscreen) objects move in the direction of the gesture
- Make the focused item obvious

Focus Model



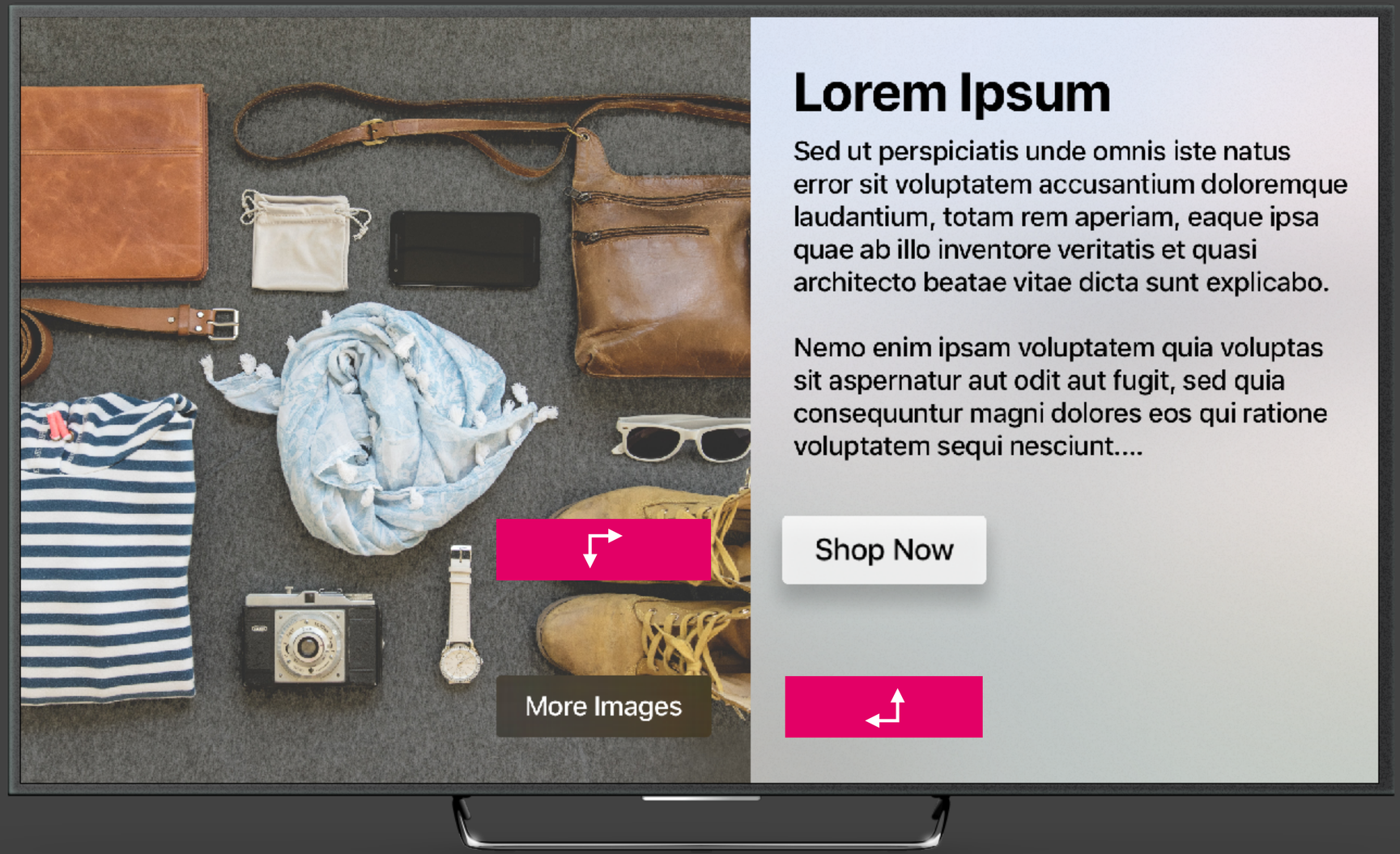
Focus Model



Overriding the Default Navigation

- Overriding the default navigation is needed, …
  - … if some UI elements are not accessible by the focus model
  - … if the semantic order of contents does not fit their physical arrangement (e.g. two column designs)
- Possible solutions
  - Statically defining successor of items
  - Adding dynamic layout guides

tvOS Focus Engine: Layout Guides

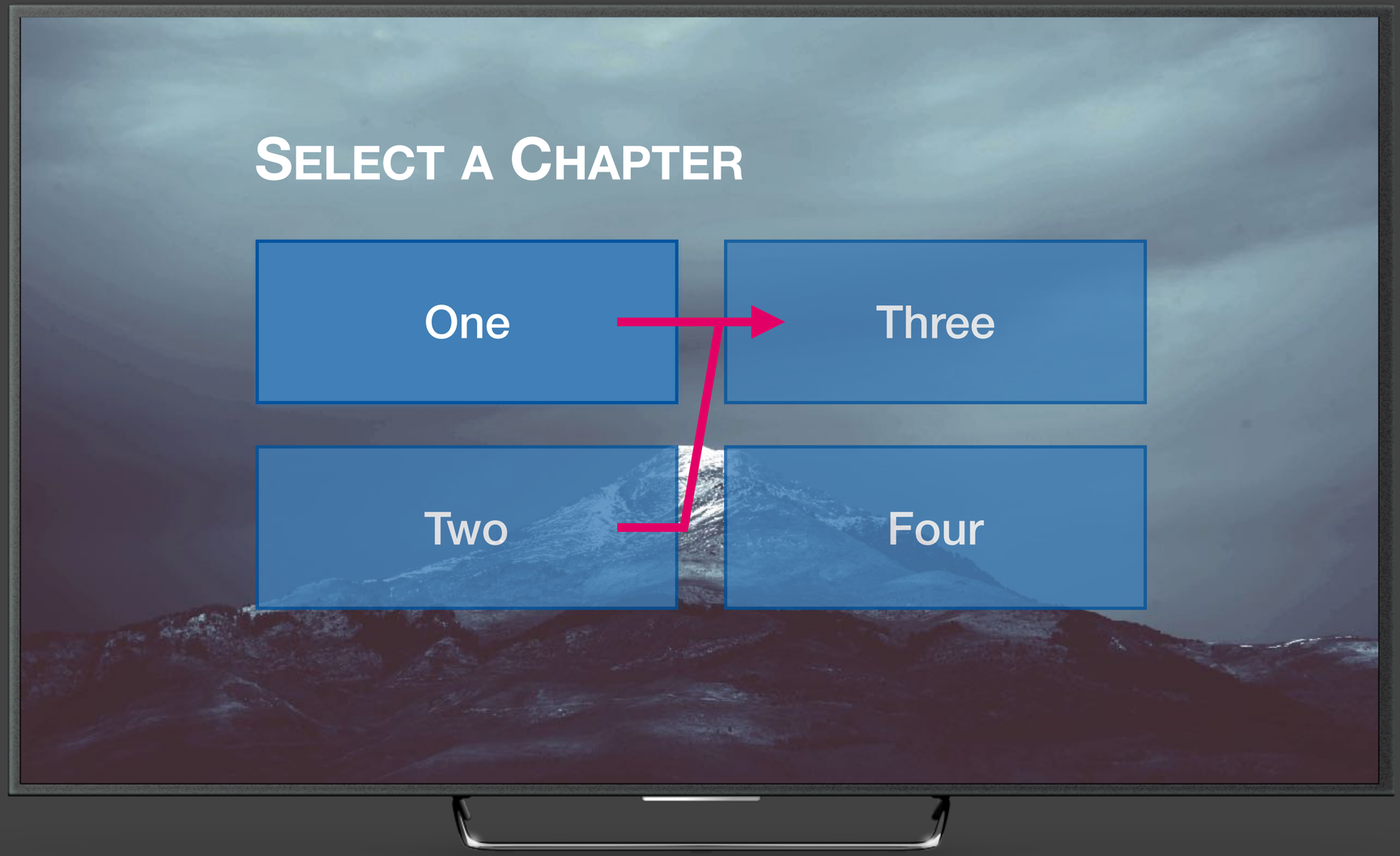


tvOS Focus Engine: Layout Guides

```
var focusGuide: UIFocusGuide = {
    let fg = UIFocusGuide()
    self.view.addLayoutGuide(fg)

    fg.rightAnchor.constraint(equalTo: shopButton.rightAnchor).isActive = true
    fg.bottomAnchor.constraint(equalTo: moreButton.bottomAnchor).isActive = true
    fg.leftAnchor.constraint(equalTo: shopButton.leftAnchor).isActive = true
    fg.topAnchor.constraint(equalTo: moreButton.topAnchor).isActive = true

    return fg
}()
```

```
override func didUpdateFocus(in context: UIFocusUpdateContext, with coordinator:
UIFocusAnimationCoordinator) {
    super.didUpdateFocus(in: context, with: coordinator)

    switch context.nextFocusedView {
        case self.moreButton:  focusGuide.preferredFocusEnvironments = [shopButton]
        default:               focusGuide.preferredFocusEnvironments = [moreButton]
    }
}
```

Setting XYFocus on Xbox



Setting XYFocus on Xbox

```
<StackPanel Orientation="Horizontal" Margin="300,300">
    <UserControl XYFocusRight="{x:Bind ButtonThree}">
        <StackPanel>
            <Button Content="One"/>
            <Button Content="Two"/>
        </StackPanel>
    </UserControl>
    <StackPanel>
        <Button x:Name="ButtonThree" Content="Three"/>
        <Button Content="Four"/>
    </StackPanel>
</StackPanel>
```

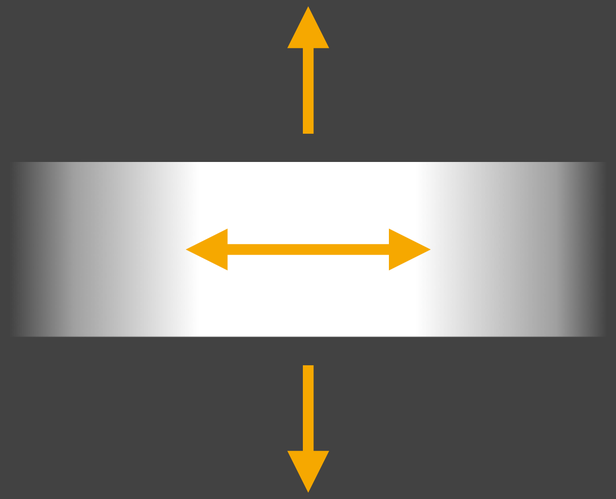
Understanding Focus Navigation

Understanding Focus Navigation

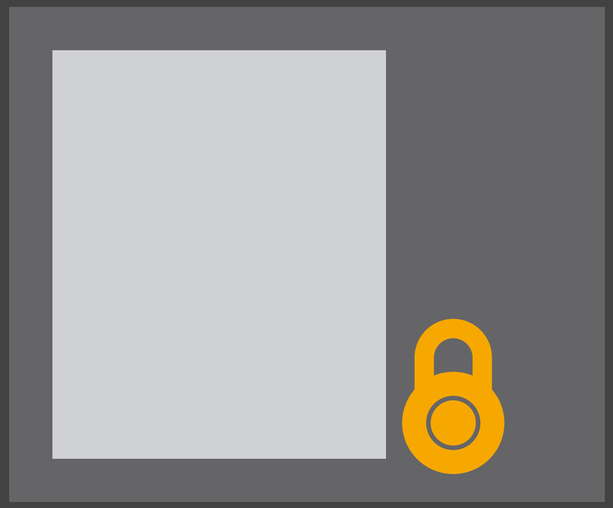
Understanding Focus Navigation

Imagine this list to have 50 items

Techniques



Use different axes for navigation and content



Lock focus in a specific area

Use a cursor for contents that scroll in two directions

What's beyond smartphones and TVs?

New Frontiers: In-Car Interaction

New Frontiers: Wearables

Limitations repeat themselves in history.