LA TROBE UNIVERSITY

All kinds of clever

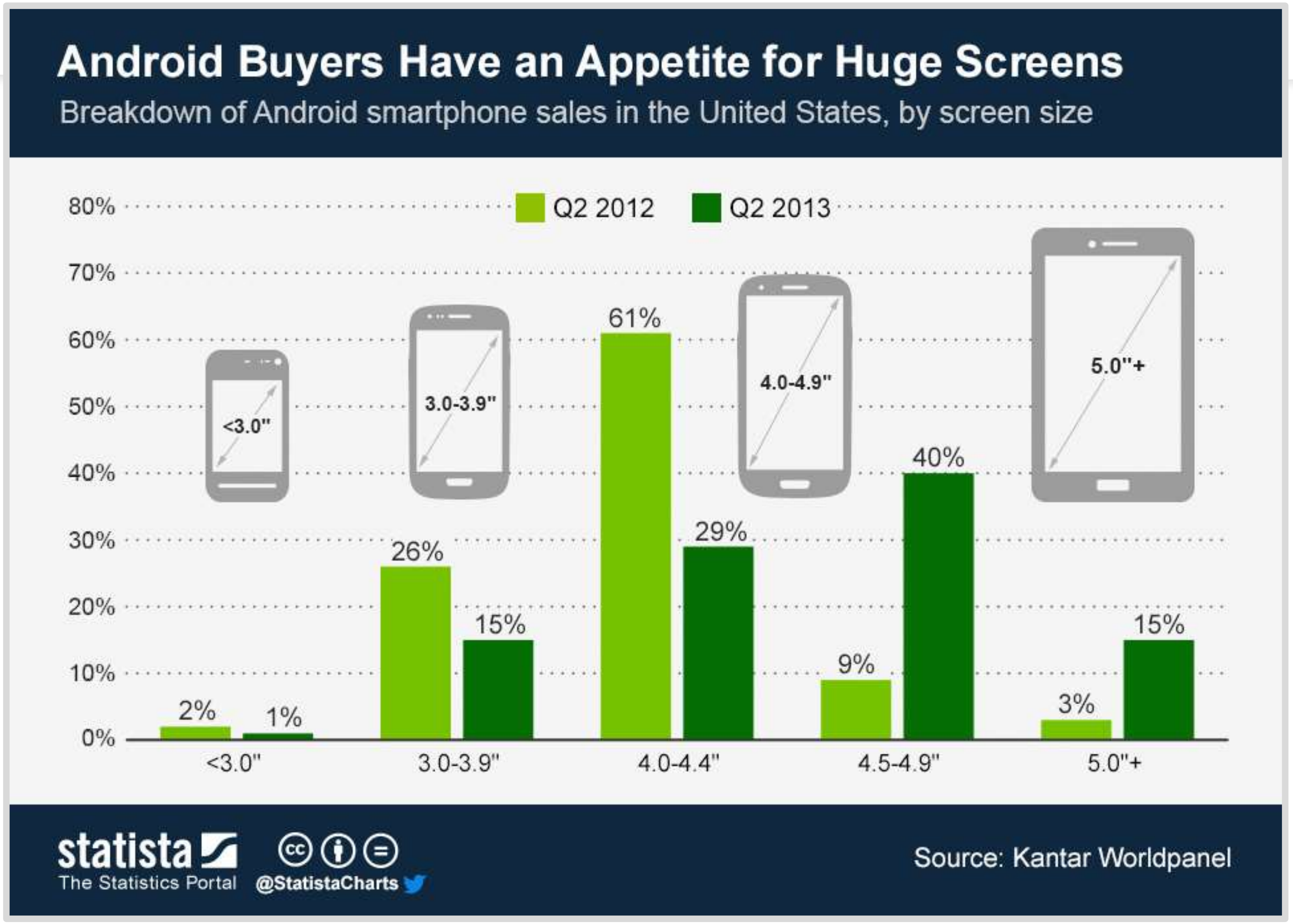
CSE2MAD

Mobile Application Development
Lecture 9

Outline

- Screen Size & Density
- Firebase UI
- Advanced Firebase
- Android Things

Supporting Multiple Screens



https://www.statista.com/chart/1396/android-phone-sales-by-screen-size/

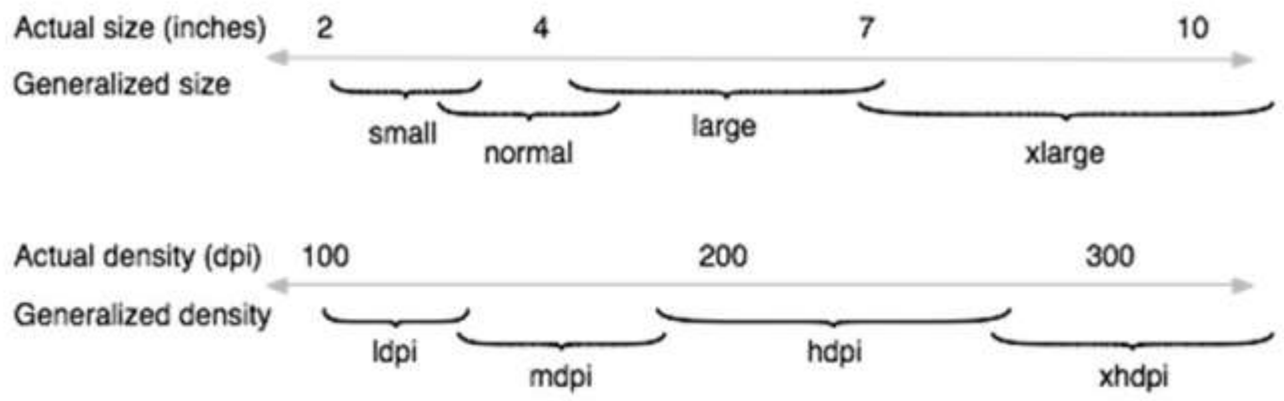
Screens Support Terms & concepts

- Screen size : physical size, measured diagonal. Groups : small, normal, large, and extralarge.
- Screen density: The quantity of pixels within a physical area of the screen; as dpi.
- Orientation: orientation of the screen from the user's POV. either landscape or portrait.
- Resolution: total number of physical pixels on a screen. When adding support for multiple screens, only consider screen size and density.
- Density-independent pixel (dp): A virtual pixel unit to use when defining UI layout, to express layout dimensions or position in a density-independent way.
  - ➢ A density-independent pixel = A physical pixel on a 160 dpi screen.
  - ➢ Conversion of dp units to screen pixels is given by-> px = dp * (dpi / 160)

The old way

Four generalized **sizes**:

1. small
2. normal
3. large
4. xlarge

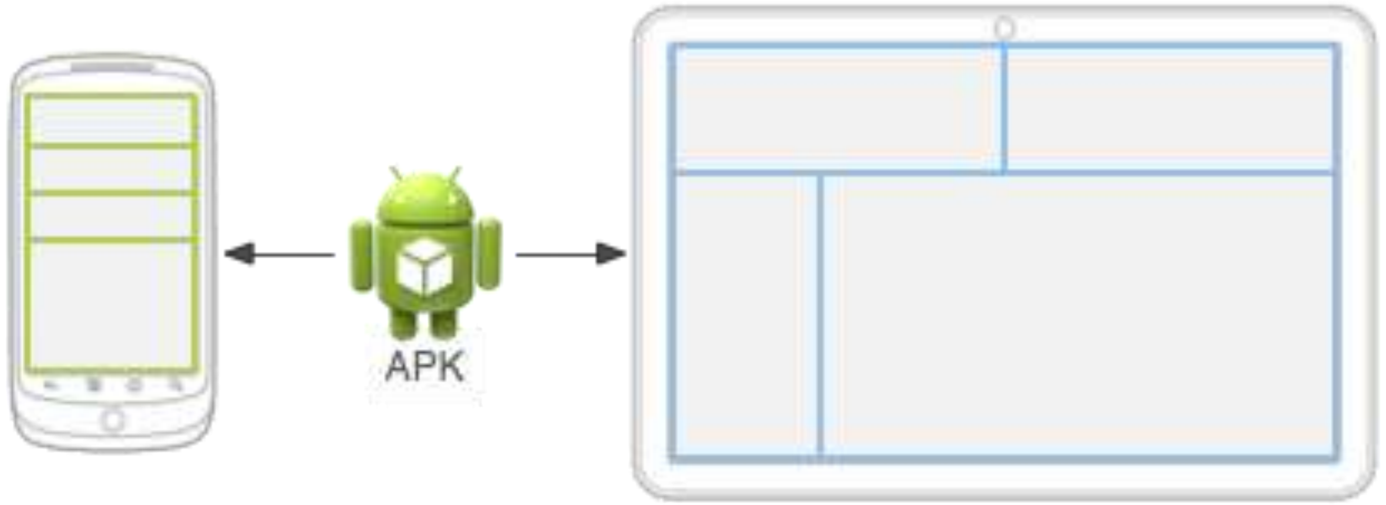


Six generalized **densities**:

1. dpi (low) ~120dpi
2. mdpi (medium) ~160dpi
3. hdpi (high) ~240dpi
4. xhdpi (extra-high) ~320dpi
5. xxhdpi (extra-extra-high) ~480dpi
6. xxxhdpi (extra-extra-extra-high) ~640dpi

The new way

- From Android 3.2
- Use view dimensions that allow the layout to resize (flexible)
- Create alternative UI layouts according to the screen configuration
- Provide bitmaps that can stretch with the views

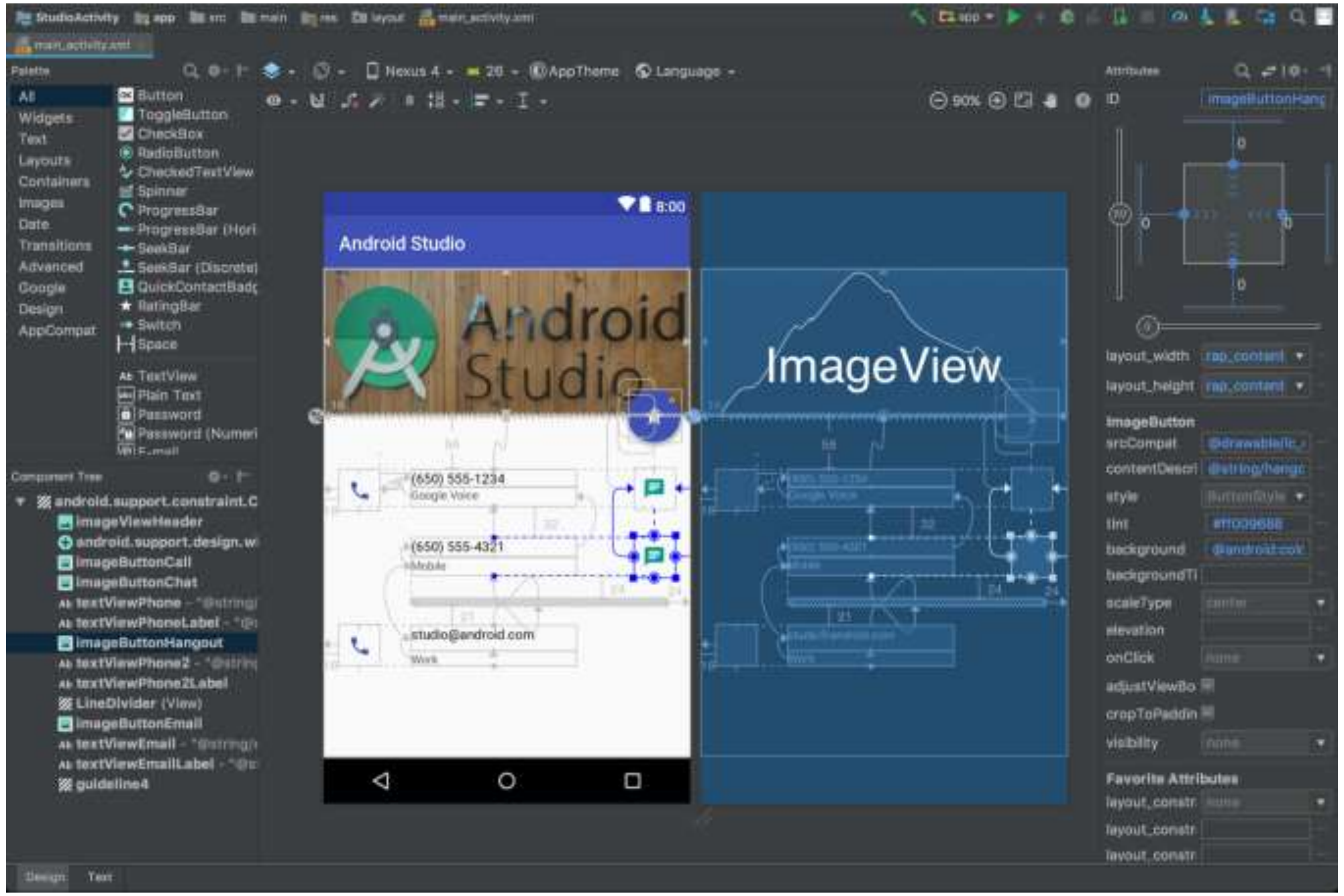


https://developer.android.com/guide/topics/resources/providing-resources

https://developer.android.com/training/multiscreen/screensizes

Create a flexible layout – Constraint Layout

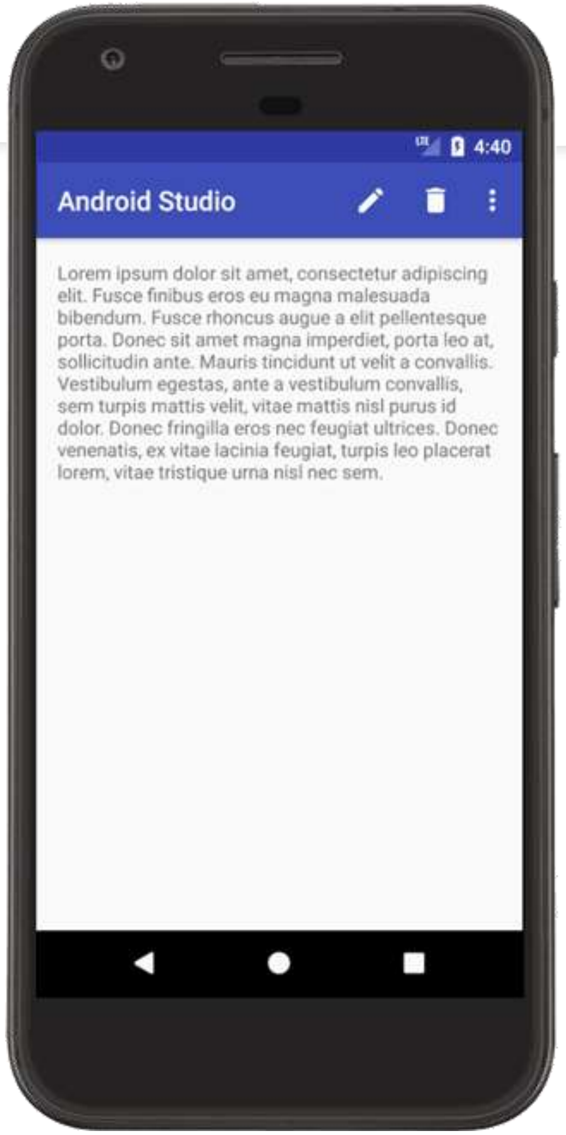
The best way to create a responsive layout for different screen sizes is to use ConstraintLayout as the base layout in your UI. ConstraintLayout allows you to specify the position and size for each view according to spatial relationships with other views in the layout. This way, all the views can move and stretch together as the screen size changes.



https://developer.android.com/training/multiscreen/screensizes#flexible-layout
https://developer.android.com/training/constraint-layout

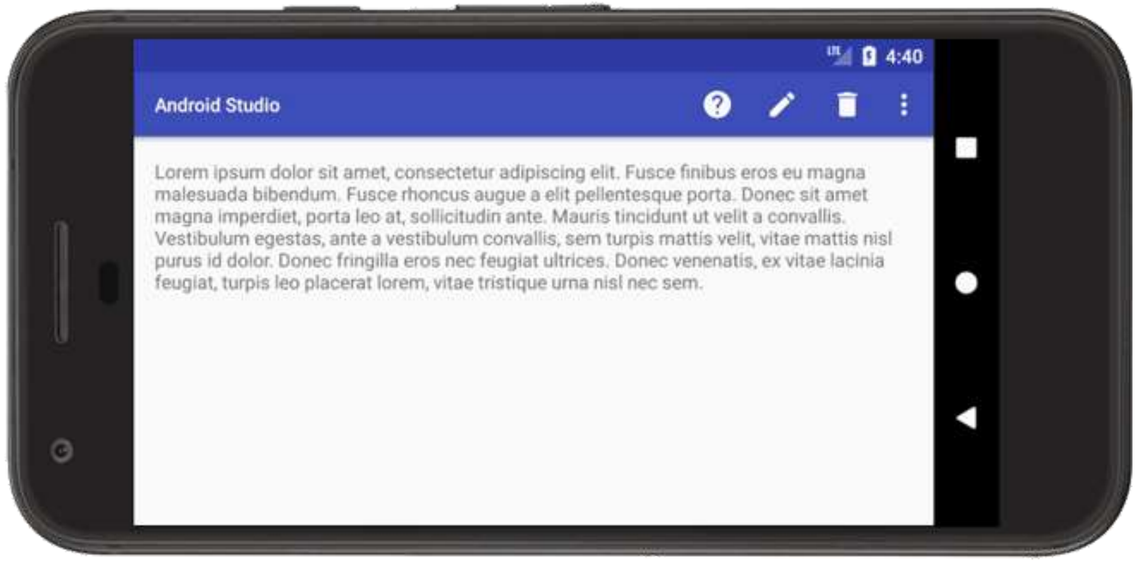
Create a flexible layout – Avoid hard-coded layout sizes

To ensure that your layout is flexible and adapts to different screen sizes, you should use "wrap_content" and "match_parent" for the width and height of most view components, instead of hard-coded sizes.



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce finibus eros eu magna malesuada bibendum. Fusce rhoncus augue a elit pellentesque porta. Donec sit amet magna imperdiet, porta leo at, sollicitudin ante. Mauris tincidunt ut velit a convallis. Vestibulum egestas, ante a vestibulum convallis, sem turpis mattis velit, vitae mattis nisl purus id dolor. Donec fringilla eros nec feugiat ultrices. Donec venenatis, ex vitae lacinia feugiat, turpis leo placerat lorem, vitae tristique urna nisl nec sem.

This example shows how the width of the text view using "match_parent" adjusts as the screen width changes with device orientation.



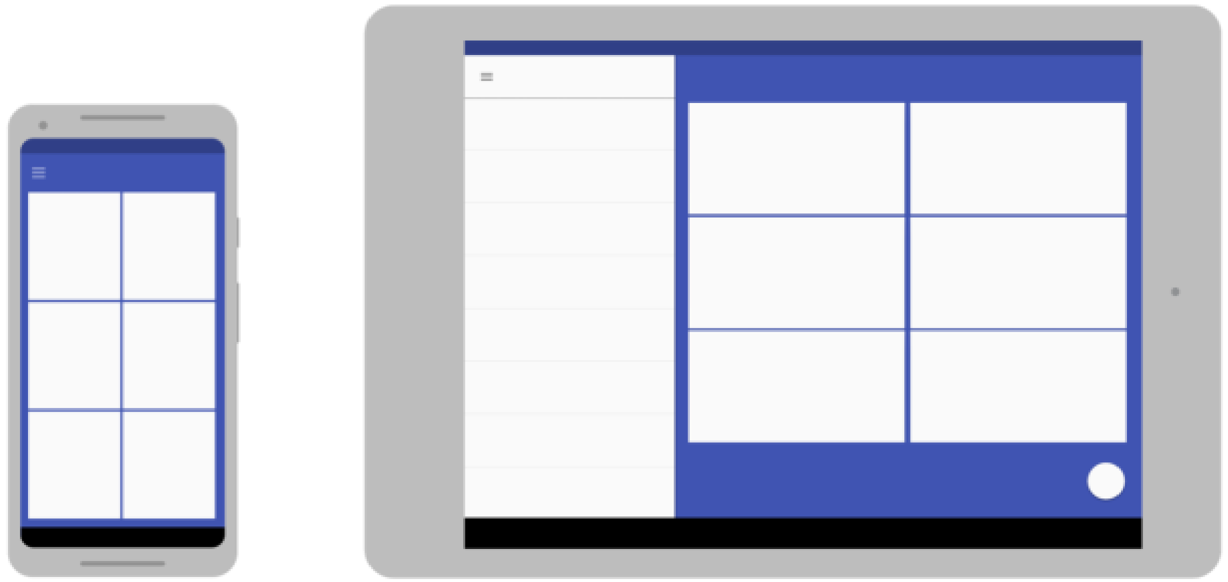
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce finibus eros eu magna malesuada bibendum. Fusce rhoncus augue a elit pellentesque porta. Donec sit amet magna imperdiet, porta leo at, sollicitudin ante. Mauris tincidunt ut velit a convallis. Vestibulum egestas, ante a vestibulum convallis, sem turpis mattis velit, vitae mattis nisl purus id dolor. Donec fringilla eros nec feugiat ultrices. Donec venenatis, ex vitae lacinia feugiat, turpis leo placerat lorem, vitae tristique urna nisl nec sem.

https://developer.android.com/training/multiscreen/screensizes

Create alternative layouts

Although your layout should always respond to different screen sizes by stretching the space within and around its views, that might not provide the best user experience for every screen size.

- For example, the UI you designed for a phone, probably doesn't offer a good experience on a tablet. Therefore, your app should also provide alternative layout resources to optimize the UI design for certain screen sizes.

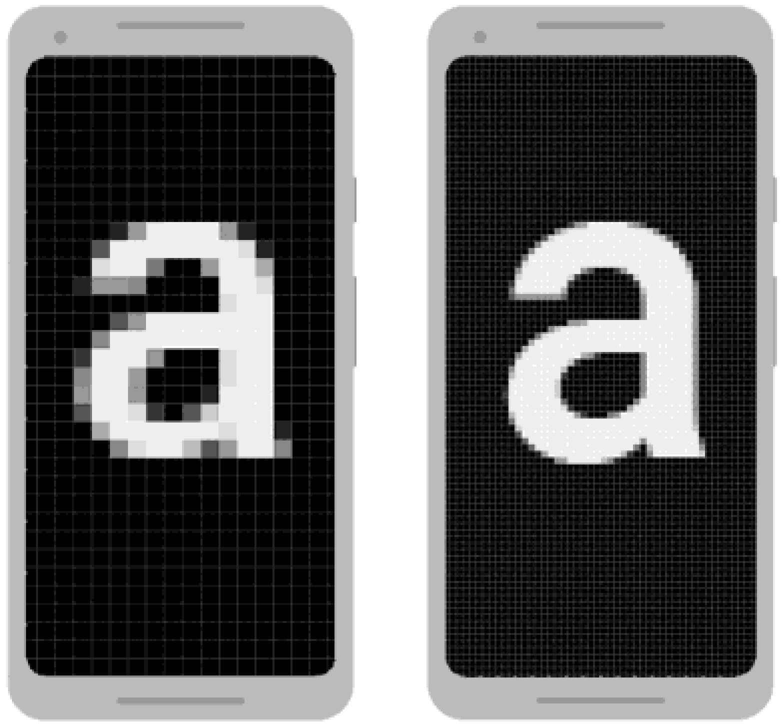


https://developer.android.com/training/multiscreen/screensizes

Use the smallest width qualifier

The "smallest width" screen size qualifier allows you to provide alternative layouts for screens that have a minimum width measured in density-independent pixels(dp or dip).

- By describing the screen size as a measure of density-independent pixels, Android allows you to create layouts that are designed for very specific screen dimensions while avoiding any concerns you might have about different pixel densities.



Two screens of the same size may have a different number of pixels

https://developer.android.com/training/multiscreen/screensizes

Use the smallest width qualifier

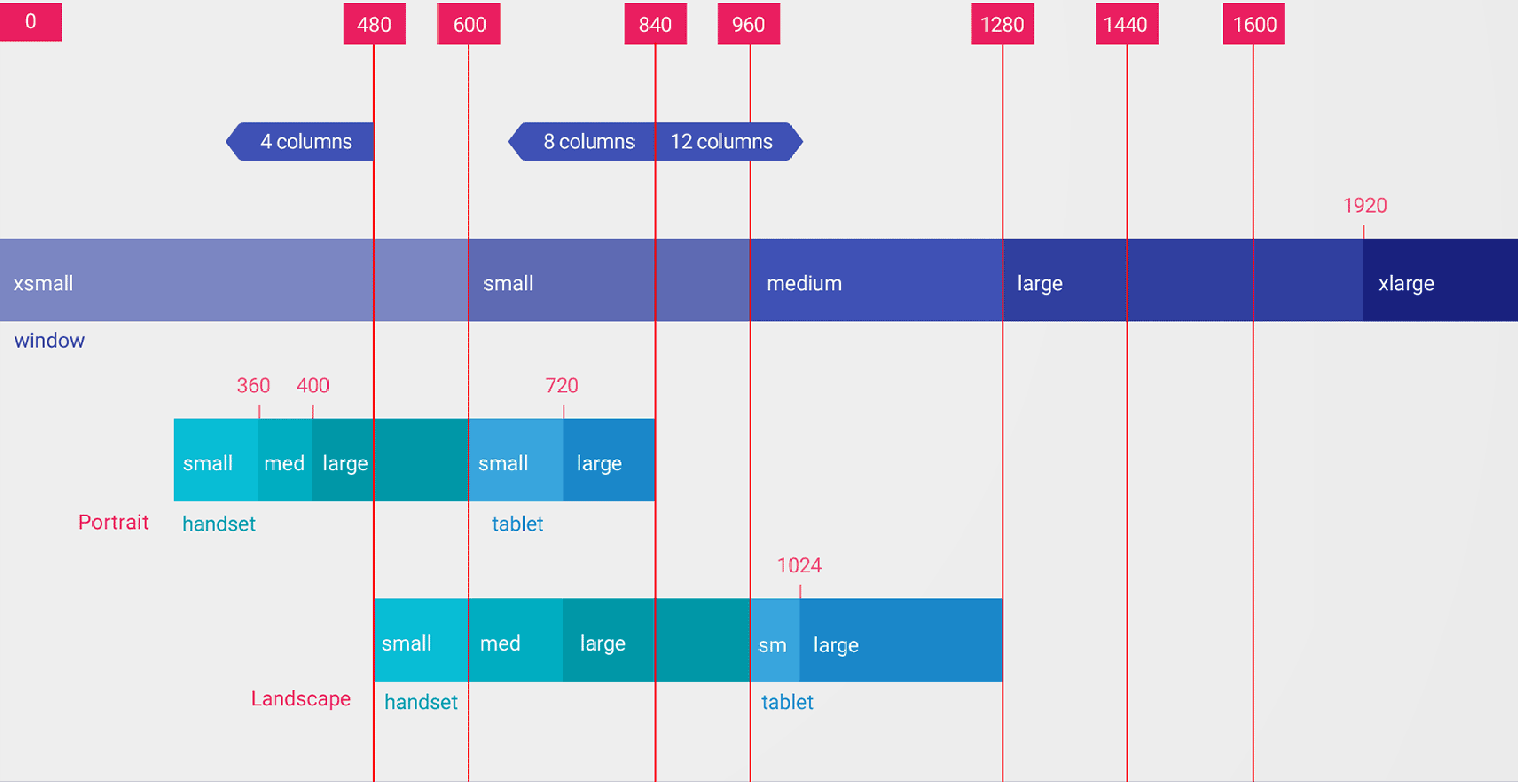
For example, you can create a layout named main_activity that's optimized for handsets and tablets by creating different versions of the file in directories as follows:

```
res/layout/main_activity.xml # For handsets (smaller than 600dp available width)
res/layout-sw600dp/main_activity.xml # For 7” tablets (600dp wide and bigger)
```

Here's how other smallest width values correspond to typical screen sizes:

- 320dp: a typical phone screen (240x320 ldpi, 320x480 mdpi, 480x800 hdpi, etc).
- 480dp: a large phone screen ~5" (480x800 mdpi).
- 600dp: a 7” tablet (600x1024 mdpi).
- 720dp: a 10” tablet (720x1280 mdpi, 800x1280 mdpi, etc).

https://developer.android.com/training/multiscreen/screensizes



https://developer.android.com/training/multiscreen/screensizes#TaskUseSWQuali

Use the available width qualifier

Instead of changing the layout based on the smallest width of the screen, you might want to change your layout based on how much width or height is currently available. For example, if you have a two-pane layout, you might want to use that whenever the screen provides at least 600dp of width, which might change depending on whether the device is in landscape or portrait orientation.

```
res/layout/main_activity.xml # For handsets (smaller than 600dp available width)
res/layout-w600dp/main_activity.xml # For 7” tablets or any screen with 600dp # available width (possibly landscape handsets)
```

If the available height is a concern for you, then you can do the same using the "available height" qualifier. For example, layout-h600dp for screens with at least 600dp of screen height.

https://developer.android.com/training/multiscreen/screensizes

Add orientation qualifiers

Although you may be able to support all size variations using only combinations of the "smallest width" and "available width" qualifiers, you might also want to change the user experience when the user switches between portrait and landscape orientations.

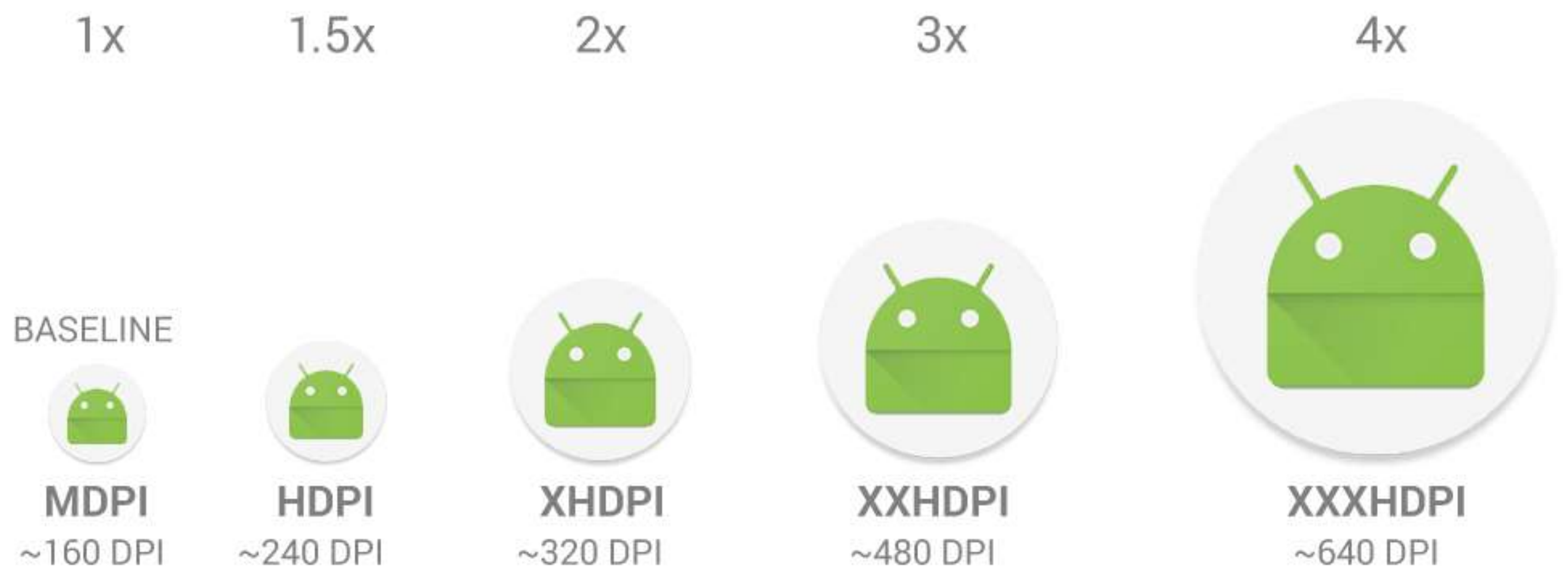
For example that you can add the port or land qualifiers to your resource directory names. Just be sure these come after the other size qualifiers.

```
res/layout/main_activity.xml # For handsets
res/layout-land/main_activity.xml # For handsets in landscape
.res/layout-sw600dp/main_activity.xml # For 7” tablets
res/layout-sw600dp-land/main_activity.xml # For 7” tablets in landscape
```

https://developer.android.com/training/multiscreen/screensizes
https://developer.android.com/guide/topics/resources/providing-resources#AlternativeResources

Provide alternative bitmaps

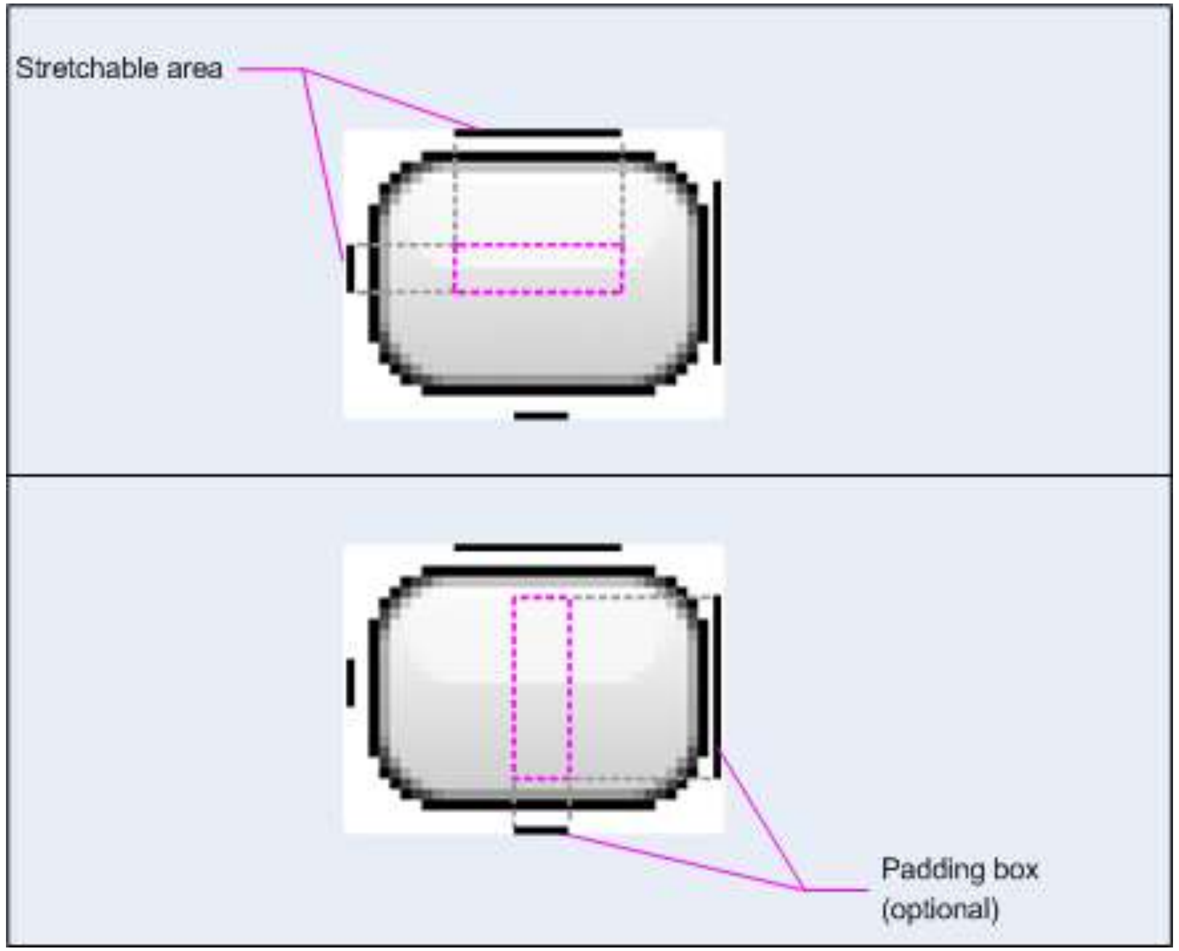
To provide good graphical qualities on devices with different pixel densities, you should provide multiple versions of each bitmap in your app—one for each density bucket, at a corresponding resolution. Otherwise, Android must scale your bitmap so it occupies the same visible space on each screen, resulting in scaling artifacts such as blurring.



https://developer.android.com/training/multiscreen/screendensities#TaskProvideAltBmp

Create stretchable nine-patch bitmaps

If you use a bitmap as the background in a view that changes size, you will notice Android scales your images as the view grows or shrinks based on the size of the screen or content in the view. This often leads to visible blurring or other scaling artifacts. The solution is using nine-patch bitmaps, which are specially formatted PNG files that indicate which areas can and cannot be stretched.



https://developer.android.com/training/multiscreen/screensizes#TaskUse9Patch
https://developer.android.com/studio/write/draw9patch

Test on all screen sizes

It's important to test your app on a variety of screen sizes so you can ensure your UI scales correctly. If you don't have access to physical devices for all the different screen sizes, you can use the Android Emulator to emulate any screen size.

If you would rather test on a physical device, but don't want to buy the devices, you can use Firebase Test Lab to access devices in a Google data center.

https://developer.android.com/training/multiscreen/screensizes

Layout Examples

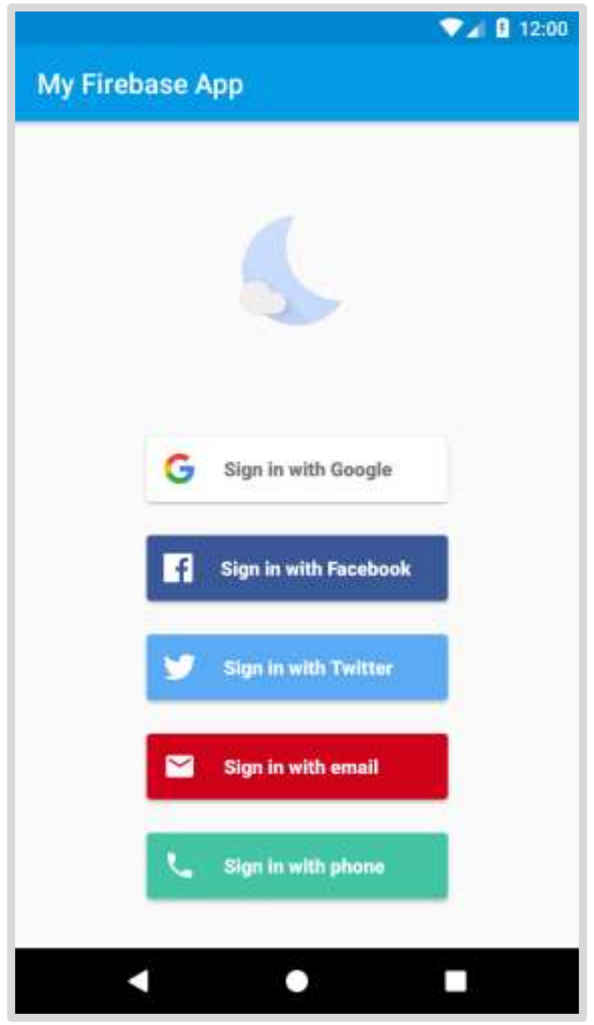
https://github.com/latrobe-cs-educator/CSE2MAD_Lecture9_Layoutdemo

FirebaseUI for Android — UI Bindings for Firebase

FirebaseUI is an open-source library for Android that allows you to quickly connect common UI elements to Firebase APIs.

FirebaseUI has separate modules for using Firebase Realtime Database, Cloud Firestore, Firebase Auth, and Cloud Storage.

https://github.com/firebase/FirebaseUI-Android

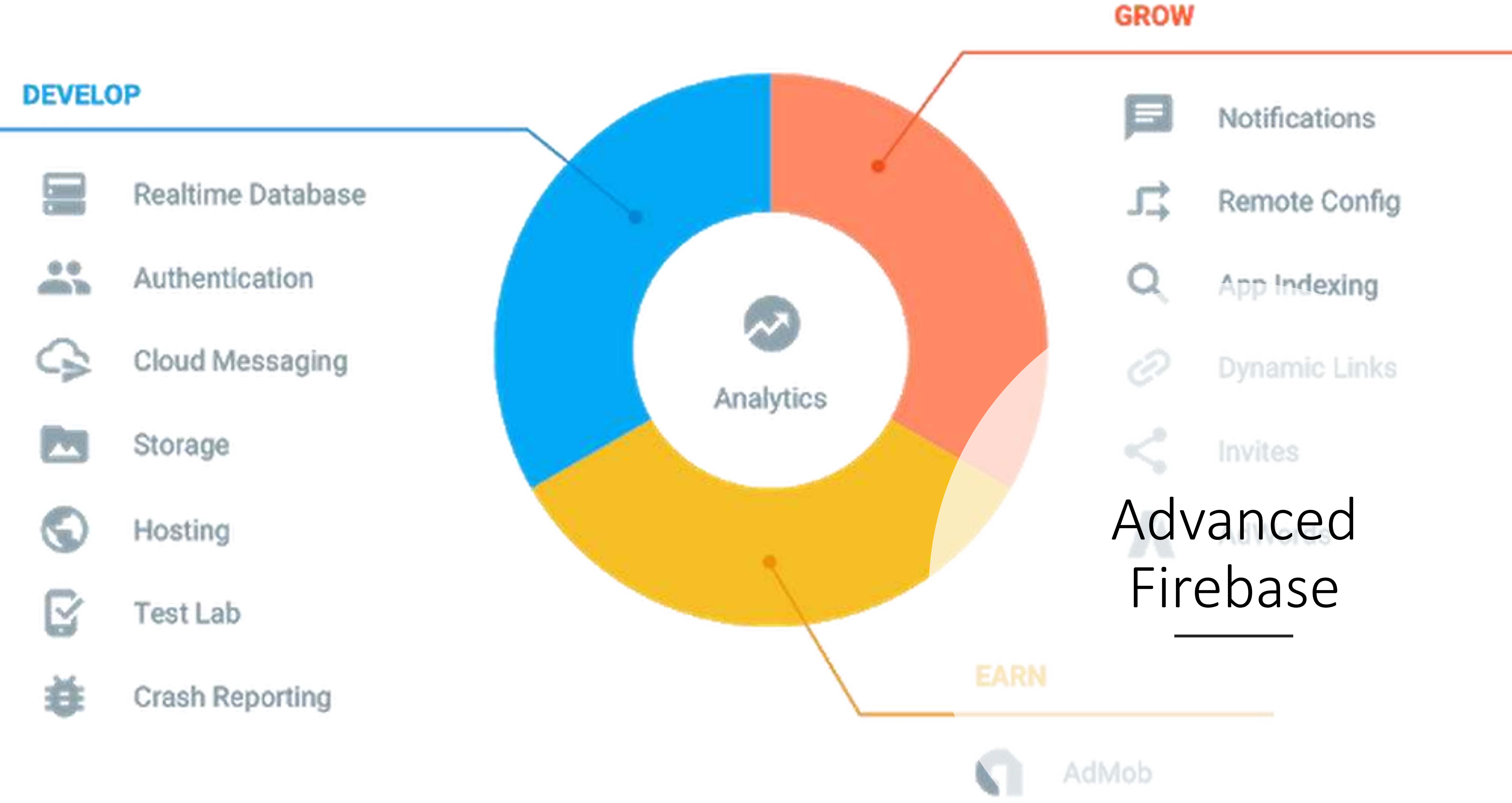


https://firebase.google.com/docs/auth/android/firebaseui

https://youtu.be/sJw-ok4W_hE example code: https://github.com/firebase/FirebaseUI-Android/tree/master/app

FirestoreRecyclerAdapter

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture6_FirebaseStore



Advanced Firebase

https://www.blog.google/products/admob/how-using-firebase-can-help-you-earn-more

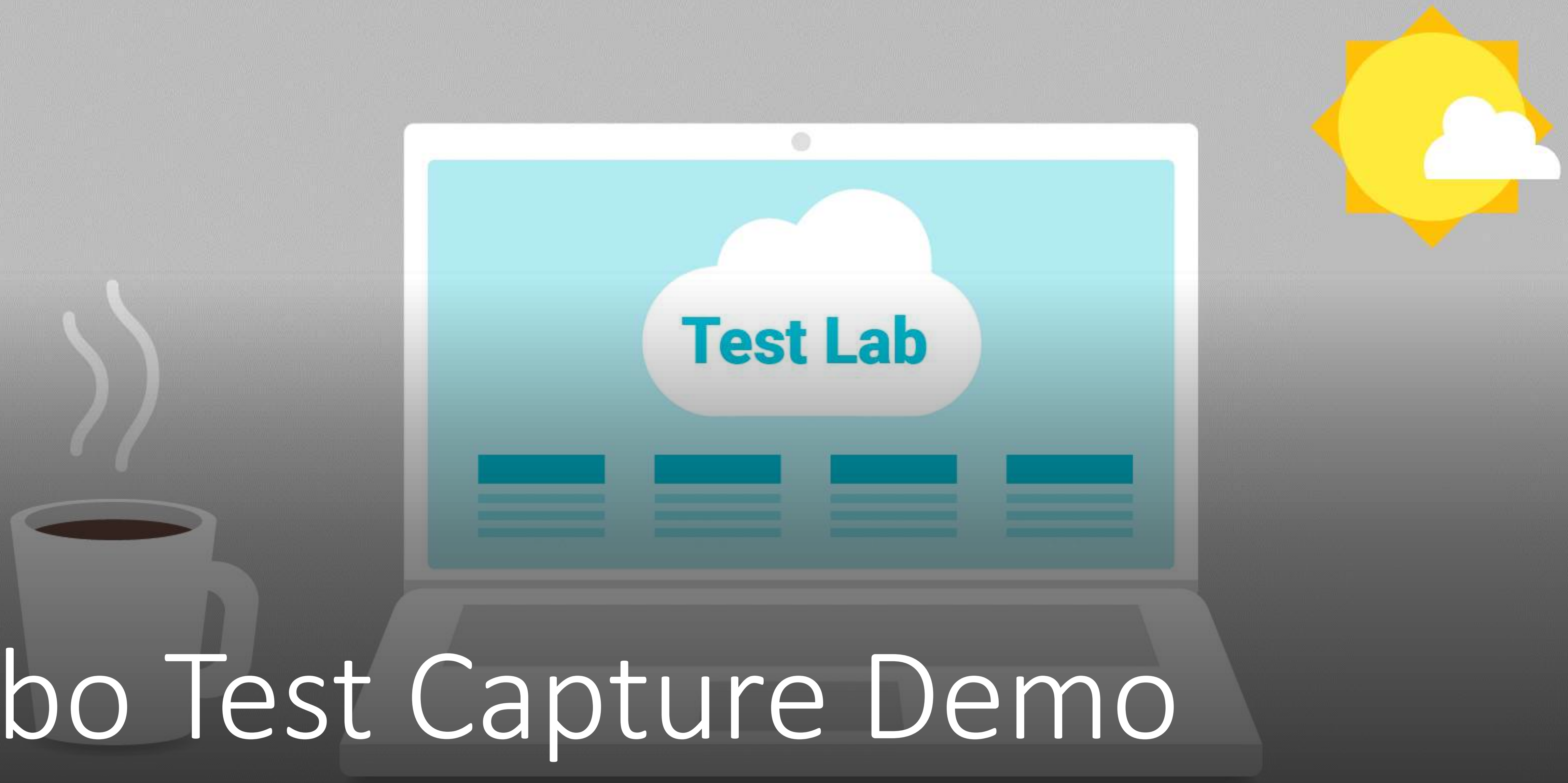
https://youtu.be/4_ZEEX1x17k

Firebase Test Lab

Key capabilities

Test Android and iOS apps	If your app has both an Android and iOS version, no worries. Test Lab now offers iOS devices to test on.
Run on real devices	Test Lab exercises your app on devices installed and running in a Google data center, so you can find issues that only occur on specific devices and configurations.
Workflow integration	Test Lab is integrated with the Firebase console, Android Studio, and the gcloud command line tool. You can even use it with Continuous Integration (CI) systems.

https://firebase.google.com/docs/test-lab



Robo Test Capture Demo

https://firebase.google.com/docs/test-lab/android/robo-ux-test

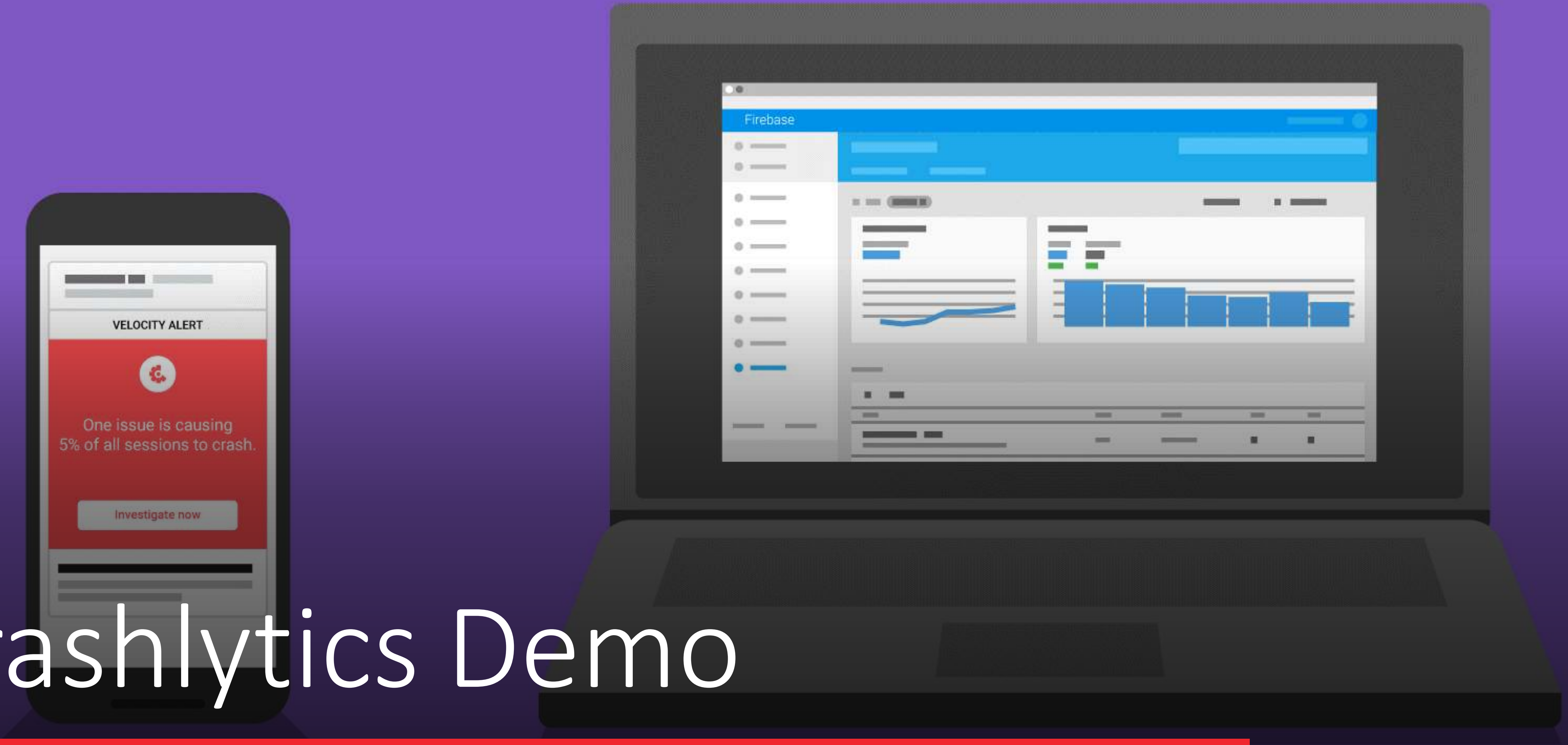
https://youtu.be/k_mdNRZzd30

Firebase Crashlytics

Key capabilities

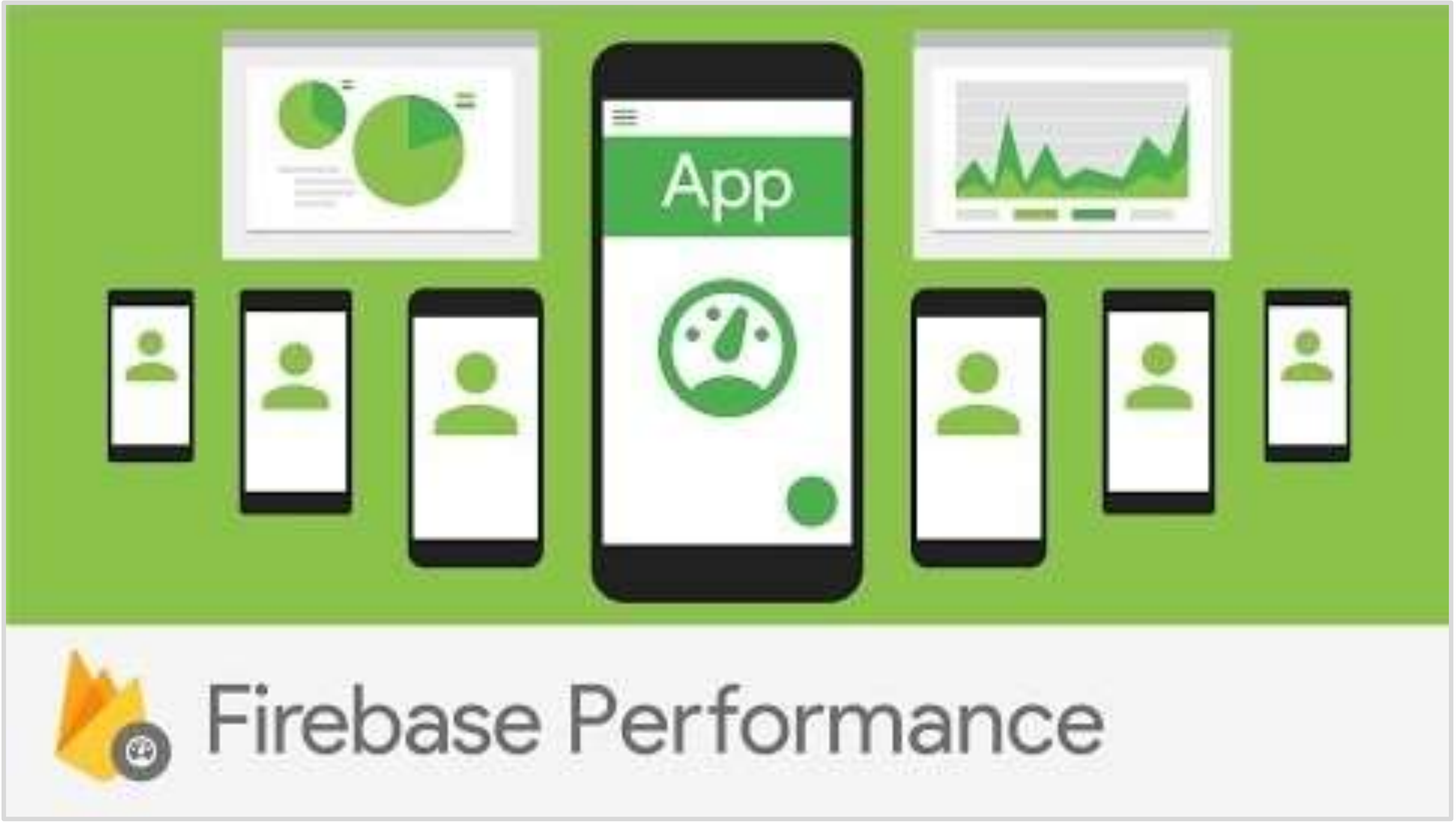
Curated crash reports	Crashlytics synthesizes an avalanche of crashes into a manageable list of issues, provides contextual information, and highlights the severity and prevalence of crashes so you can pinpoint the root cause faster.
Cures for the common crash	Crashlytics offers Crash Insights, helpful tips that highlight common stability problems and provide resources that make them easier to troubleshoot, triage, and resolve.
Integrated with Analytics	Crashlytics can capture your app's errors as app_exception events in Analytics. The events simplify debugging by giving you access a list of other events leading up to each crash, and provide audience insights by letting you pull Analytics reports for users with crashes.
Realtime alerts	Get realtime alerts for new issues, regressed issues, and growing issues that might require immediate attention.

https://firebase.google.com/docs/crashlytics



Crashlytics Demo

https://firebase.google.com/docs/crashlytics/get-started



https://youtu.be/0EHSPFvH7vk

Firebase Performance Monitoring

Key capabilities

Automatically measure app startup time, HTTP network requests, and more	When you integrate the Performance Monitoring SDK into your app, you don't need to write any code before your app starts automatically monitoring several critical aspects of performance. For native apps, the SDK logs startup time, rendering data by screen, and activity while in the foreground or background. For web apps, the SDK logs aspects like first contentful paint, ability for users to interact with your app, and more.
Gain insight into situations where app performance could be improved	Optimizing the performance of your app can be challenging when you don't know exactly why it is falling short of user expectations. That's why Performance Monitoring lets you see performance metrics broken down by *attributes*, like country, device, app version, and OS level.
Customize monitoring for your app	You can instrument *custom code traces* to capture your app's performance in specific situations, like when you load a new screen or display a new interactive feature. And, you can create *custom metrics* on these custom code traces to count events that you define (like cache hits) during those traces.

https://firebase.google.com/docs/perf-mon

android
things

https://developer.android.com/things/get-started

Android Things

Android Things lets you experiment with building smart, connected device applications.

Develop apps for your devices with existing Android development tools, APIs, and resources along with new APIs that provide low level I/O and libraries for common components like temperature sensors, display controllers, and more.

https://youtu.be/HxRv_w5DcxM

https://developer.android.com/things

Internet of Things (IoT)

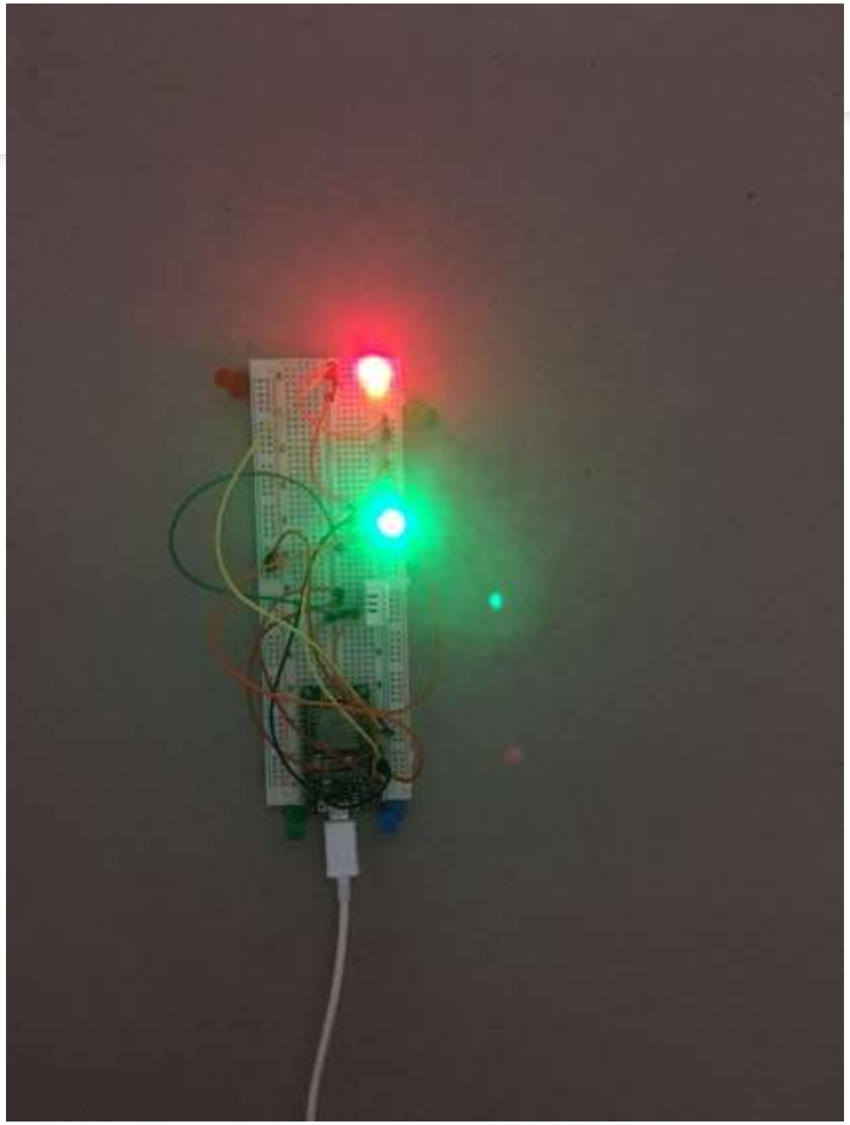
Embed connectivity into everyday objects

Furniture, Household appliances, clothing … ,…., ….

Internet connected send/receive

Include sensors (temperature, light ..) and effectors

(motors, lights, buzzers)



IoT Device Temp/Humidity/Dew Point

Libelium Smart World

Air Pollution
Control of CO_2 emissions of factories, pollution emitted by cars and toxic gases generated in farms.

Forest Fire Detection
Monitoring of combustion gases and preemptive fire conditions to define alert zones.

Wine Quality Enhancing
Monitoring soil moisture and trunk diameter in vineyards to control the amount of sugar in grapes and grapevine health.

Offspring Care
Control of growing conditions of the offspring in animal farms to ensure its survival and health.

Sportsmen Care
Vital signs monitoring in high performance centers and fields.

Structural Health
Monitoring of vibrations and material conditions in buildings, bridges and historical monuments.

Smartphones Detection
Detect iPhone and Android devices and in general any device which works with Wifi or Bluetooth interfaces.

Perimeter Access Control
Access control to restricted areas and detection of people in non-authorized areas.

Radiation Levels
Distributed measurement of radiation levels in nuclear power stations surroundings to generate leakage alerts.

Electromagnetic Levels
Measurement of the energy radiated by cell stations and and WiFi routers.

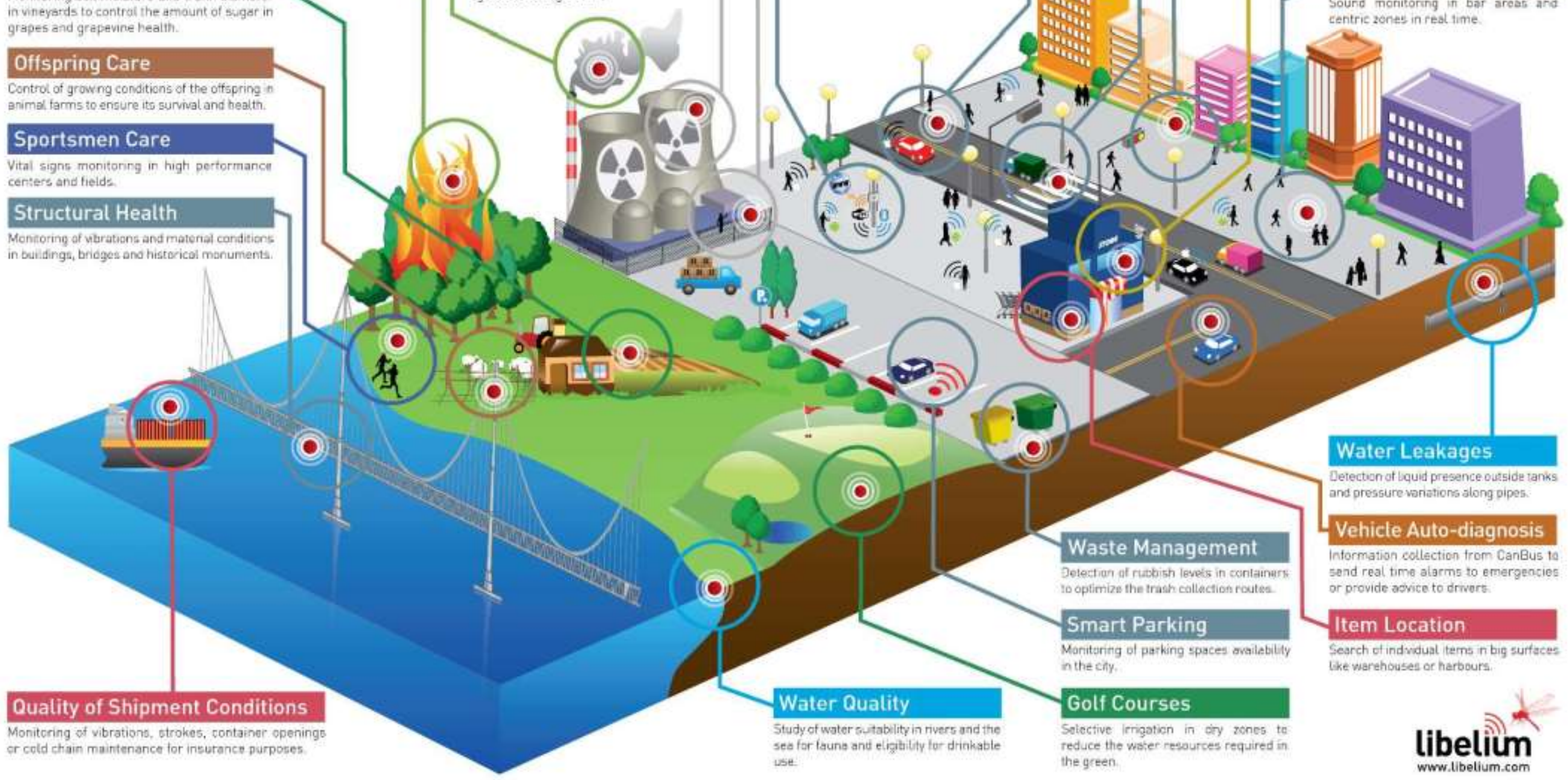
Traffic Congestion
Monitoring of vehicles and pedestrian affluence to optimize driving and walking routes.

Smart Roads
Warning messages and diversions according to climate conditions and unexpected events like accidents or traffic jams.

Smart Lighting
Intelligent and weather adaptive lighting in street lights.

Intelligent Shopping
Getting advices in the point of sale according to customer habits, preferences, presence of allergic components for them or expiring dates.

Noise Urban Maps
Sound monitoring in bar areas and centric zones in real time.



Water Leakages
Detection of liquid presence outside tanks and pressure variations along pipes.

Vehicle Auto-diagnosis
Information collection from CanBus to send real time alarms to emergencies or provide advice to drivers.

Item Location
Search of individual items in big surfaces like warehouses or harbours.

Waste Management
Detection of rubbish levels in containers to optimize the trash collection routes.

Smart Parking
Monitoring of parking spaces availability in the city.

Golf Courses
Selective irrigation in dry zones to reduce the water resources required in the green.

Water Quality
Study of water suitability in rivers and the sea for fauna and eligibility for drinkable use.

Quality of Shipment Conditions
Monitoring of vibrations, strokes, container openings or cold chain maintenance for insurance purposes.

libelium
www.libelium.com