Android Jetpack, modern Android fejlesztés

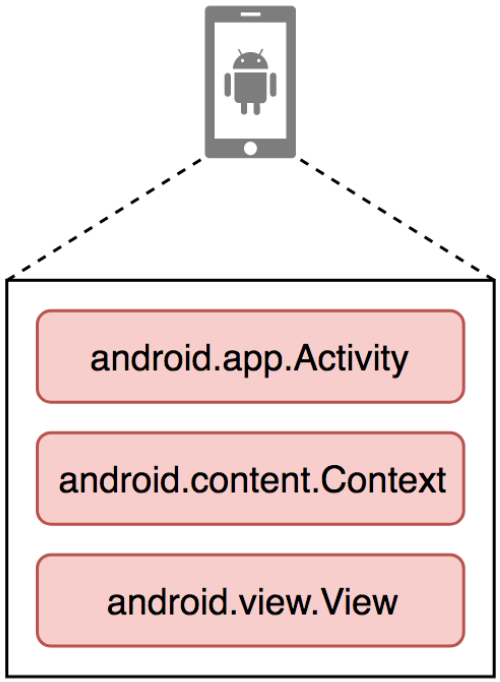
Braun Márton

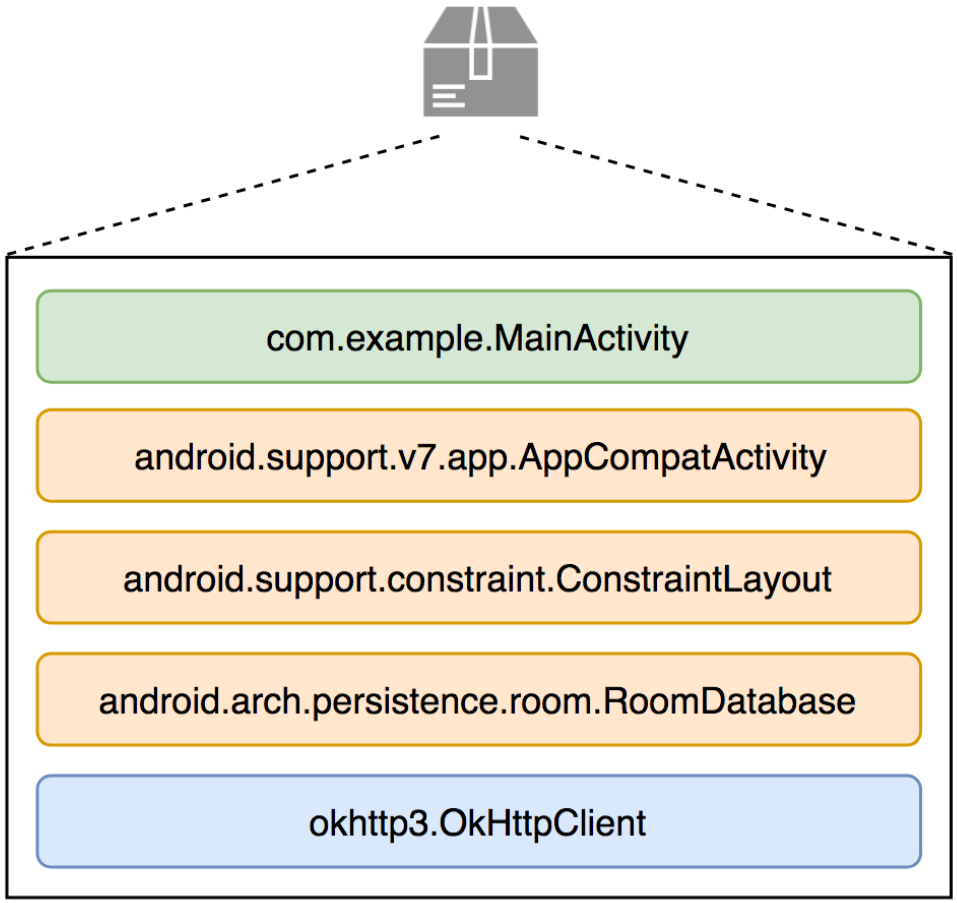
braun.marton@autsoft.hu

Az Android support library-k

- Alapvetően új funkciók elérésére régebbi platformokon
  - android.support.v7.app.AppCompatActivity
  - android.support.v4.app.Fragment
- Teljesen új funkciók, segédosztályok is
  - android.support.v7.widget.RecyclerView
  - android.support.constraint.ConstraintLayout

Az Android support library-k





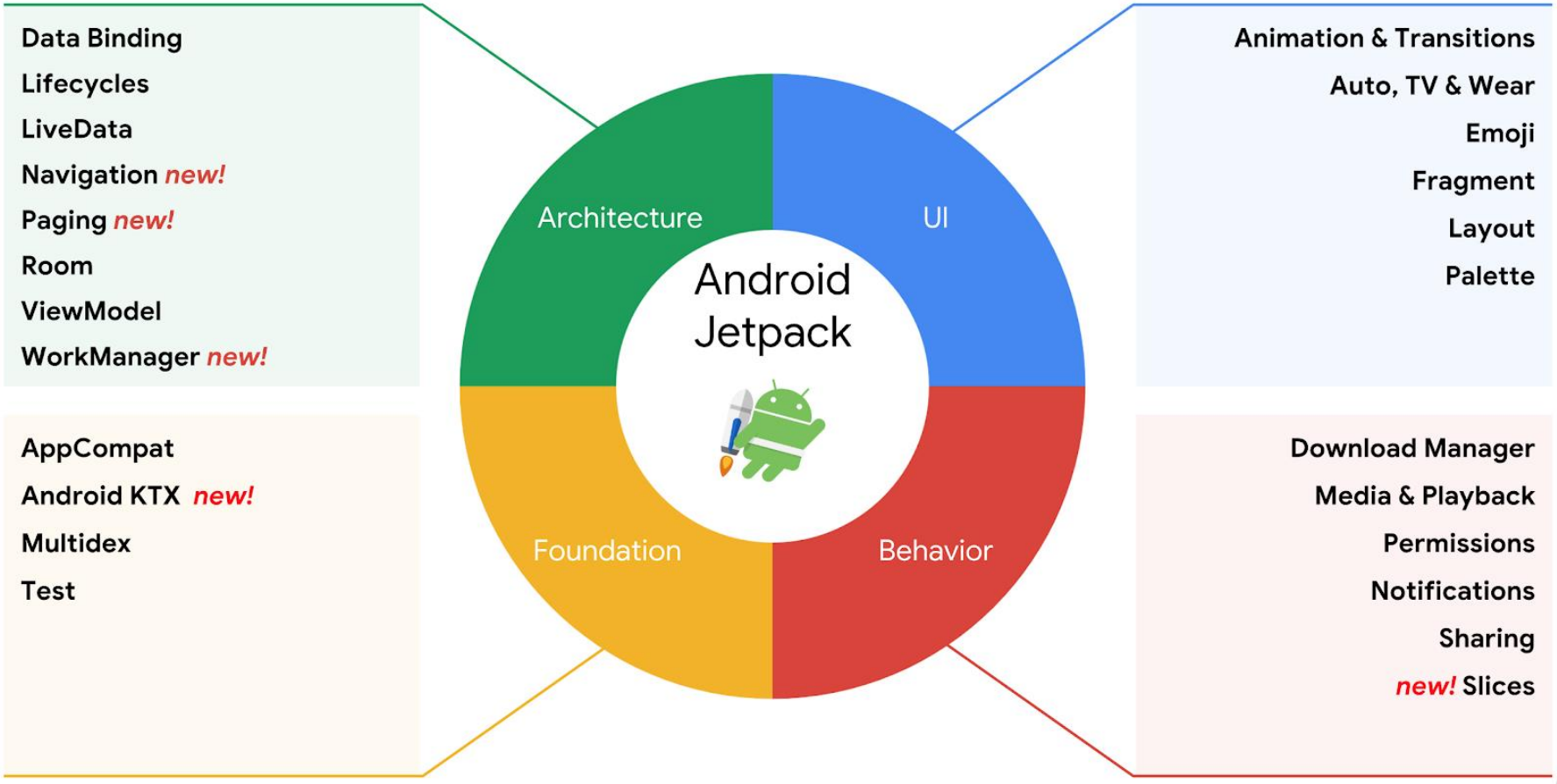
Architecture components

- Google I/O 2017
  - A Google válaszol a nagy kérdésre: milyen architektúrát használjunk az alkalmazásunkhoz?
  - “Opinionated guide to app architecture”
    - https://developer.android.com/jetpack/docs/guide
  - Architecture components
    - Room
    - ViewModel
    - LiveData
    - Lifecycle

Jetpack

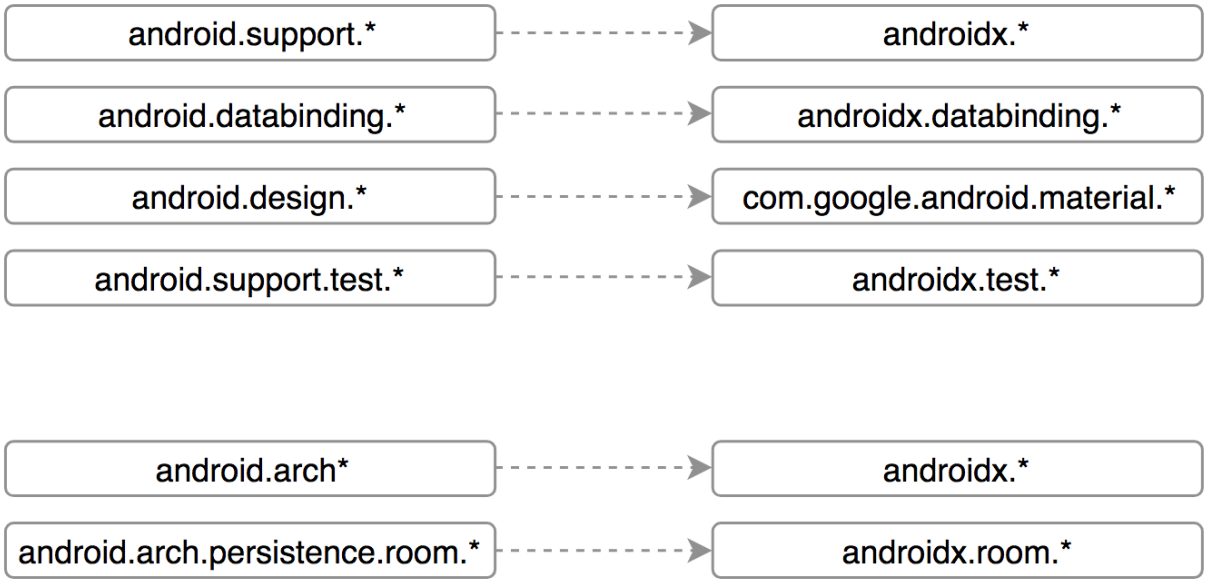
- Google I/O 2018
- Ajánlott könyvtárak és útmutatás Android fejlesztéshez
  - Tetszőlegesen válogathatók, egyenként is használhatók
- Ígéretek:
  - Gyorsabb fejlesztés
  - Modern architektúra
  - Boilerplate nélküli kód
  - Komplex feladatok egyszerűsítése
  - Visszafele kompatibilitás

Jetpack


Data Binding
Lifecycles
LiveData
Navigation new!
Paging new!
Room
ViewModel
WorkManager new!
AppCompat
Android KTX new!
Multidex
Test
Architecture
UI
Android Jetpack
Foundation
Behavior
Animation & Transitions
Auto, TV & Wear
Emoji
Fragment
Layout
Palette
Download Manager
Media & Playback
Permissions
Notifications
Sharing
new! Slices

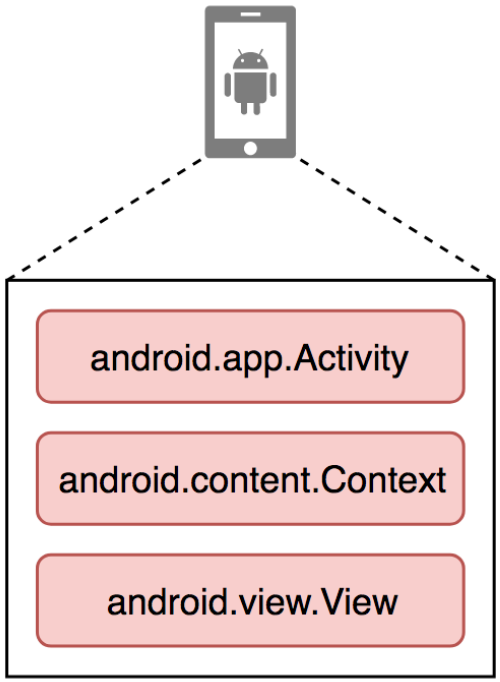
AndroidX

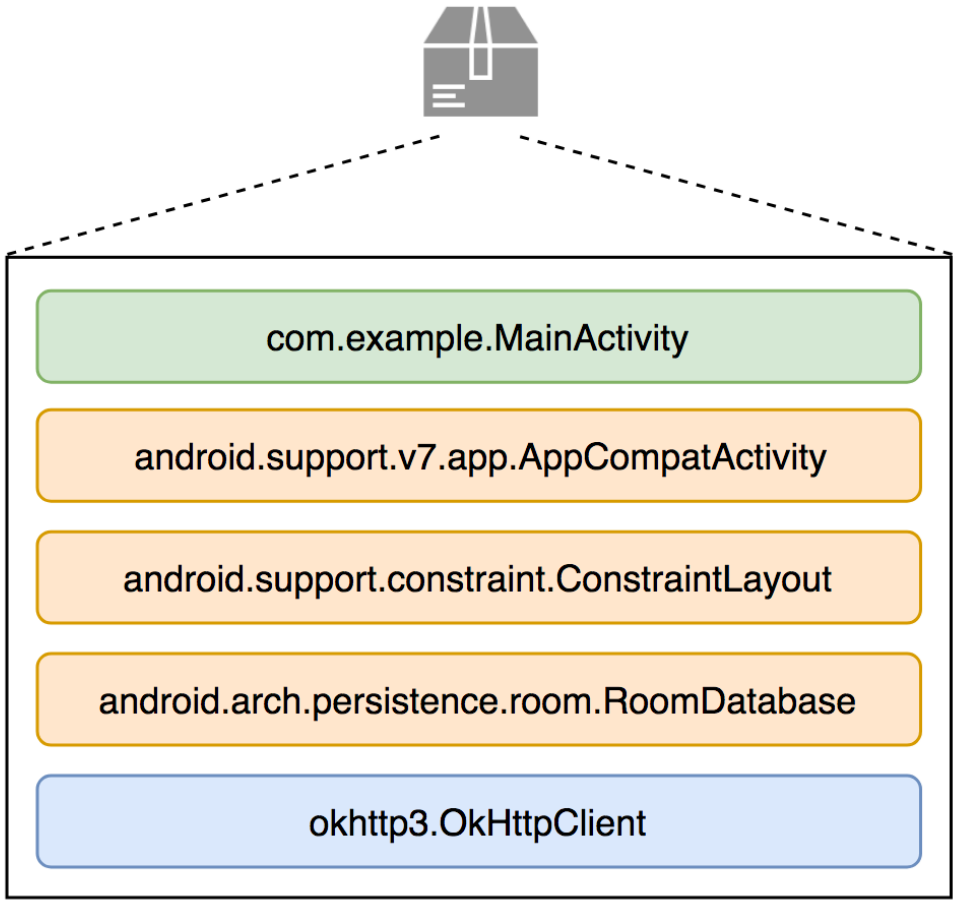
- Meglévő csomagok átszervezése új, egységes package név alá



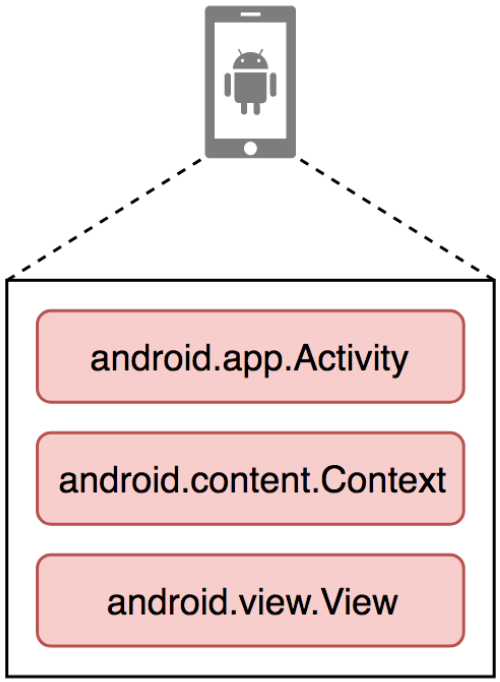
- Sokkal kisebb csomagok
  - Jelleg helyett adott funkció

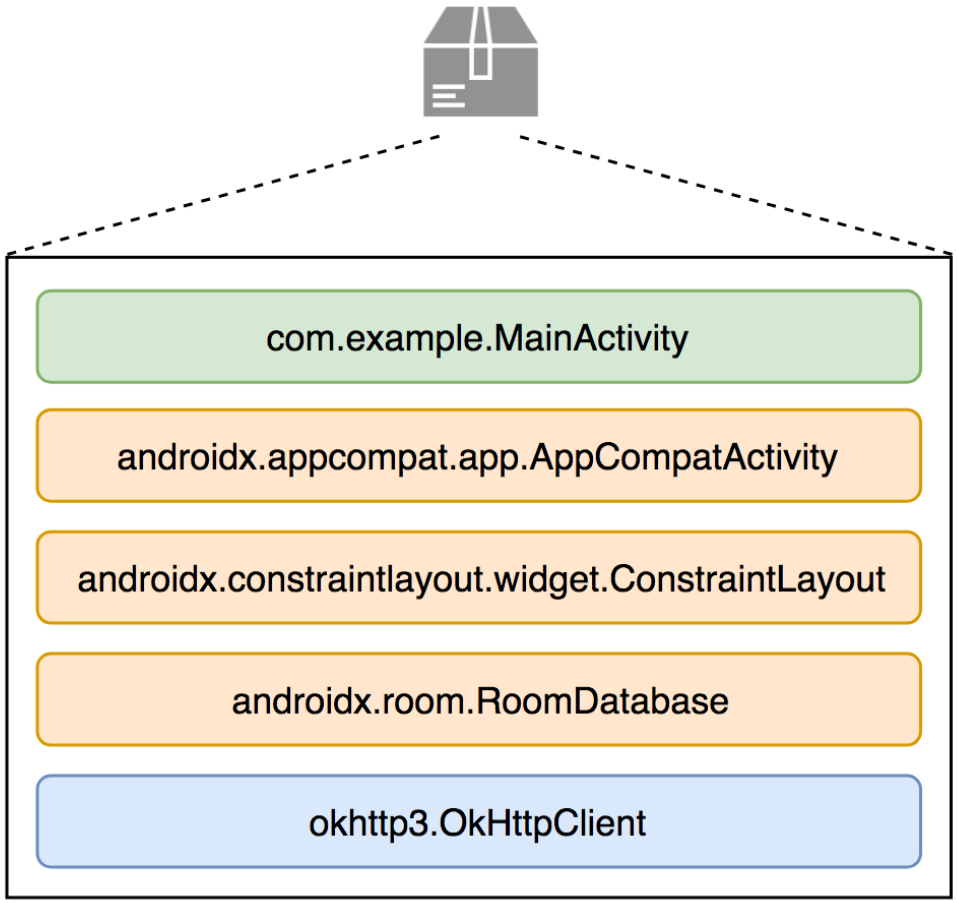
AndroidX előtt





AndroidX után





AndroidX: verziószámok

- Eddig
  - A support library az Android verziókat követte
    - Pl. API 27 használata esetén a 27.x.x-es support library-t használtuk
- Mostantól
  - 1.0.0-tól indulnak újra
    - Az 1.0.0 első stabil verziója még kiadásra kerül 28.0.0 számmal, a régi package nevekkel
  - Szemantikus verziószám
    - Major: kompatibilitás
    - Minor: új funkciók
    - Patch: bugfixek
  - Nem lesz mindnek ugyanaz a verziószáma

Jetpack vs AndroidX

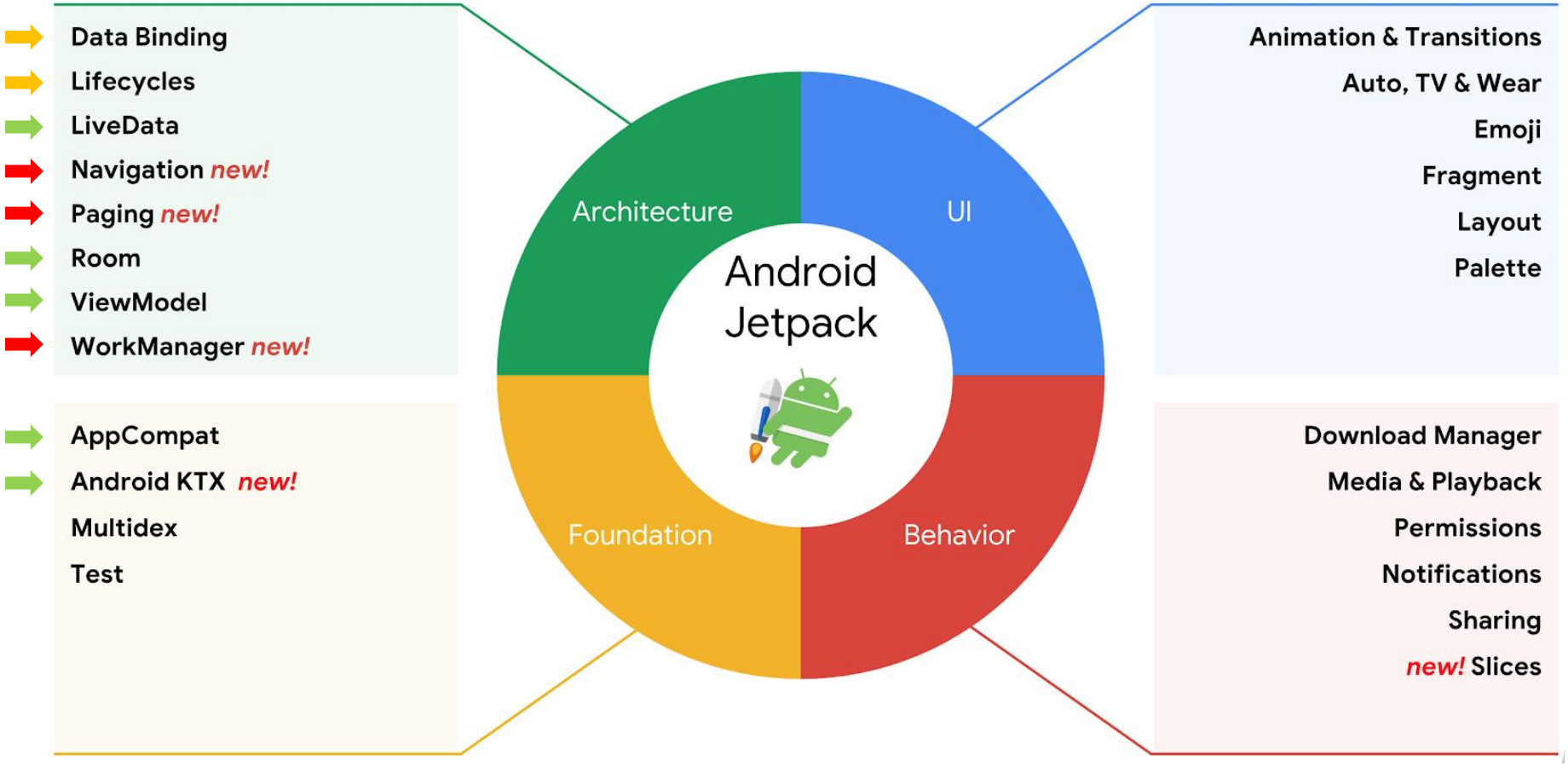
Jetpack

- Guidance
- Recommended libraries and tools
- Has a cute logo

AndroidX

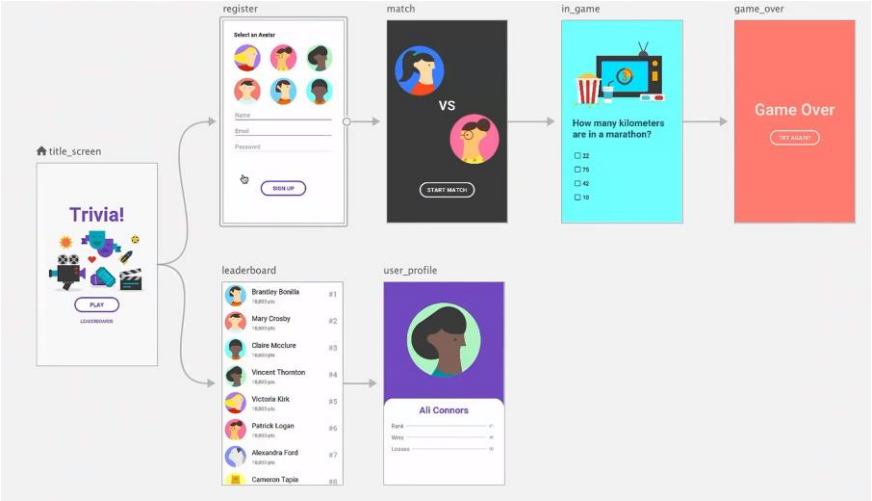
- Libraries
- Technical guarantees
- Does not have a cute logo

A mai alkalommal...



Navigation

- Hivatalos megoldás alkalmazáson belüli navigációra
- Funkciók:
  - Gráf alapú leírás
  - Argumentum átadás
  - Fel és vissza navigáció kezelése
  - Olvasható XML kód



- https://codelabs.developers.google.com/codelabs/android-navigation/

WorkManager

- Hosszan tartó háttérben futó műveletekre
  - Előzmények: JobScheduler, Firebase JobDispatcher
- Együttműködik a rendszerrel
  - Garantált végrehajtás
  - Betartja a háttérfolyamatokra vonatkozó korlátozásokat
  - Energiatakarékos
  - Visszafele kompatibilis
- Funkciók:
  - Futtatási feltételek: wifi, töltő, tárhely, stb.
  - Láncolás
  - Állapot lekérdezés
- https://codelabs.developers.google.com/codelabs/android-workmanager/

WorkManager

```
class MyWorker : Worker() {
    override fun doWork(): Result {
        // Do work here
        return Result.SUCCESS // SUCCESS, FAILURE, RETRY
    }
}

val constraints = Constraints.Builder()
        .setRequiredNetworkType(NetworkType.UNMETERED)
        .setRequiresBatteryNotLow(true)
        .build()

val workRequest = OneTimeWorkRequest.Builder(MyWorker::class.java)
        .setConstraints(constraints)
        .build()

WorkManager.getInstance()?.enqueue(workRequest)
```

Paging

- Nagyon hosszú (akár végtelen) listák kezelése
  - Nem praktikus vagy nem lehetséges előre betölteni az összes adatot
- Integrációk:
  - RecyclerView
  - LiveData, RxJava
- Központi objektuma a DataSource
  - Aszinkron betölti az adatokat
    - Adatbázisból, hálózatról, fájlból, bárhonnan
  - Automatikusan meghívódik, amikor szükség van több adatra
- https://codelabs.developers.google.com/codelabs/android-paging/

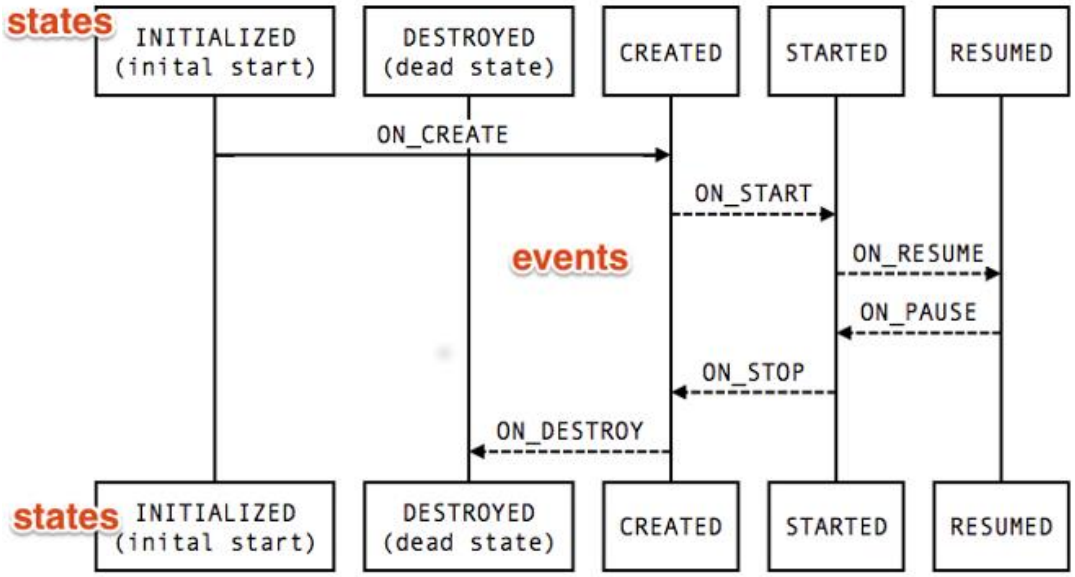
Data Binding

- A UI értékeinek hozzákötése modell objektumokhoz

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
        <variable
            name="user"
            type="com.example.User" />
    </data>

    <TextView
        android:id="@+id/tvFirstName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@{user.name}" />

</layout>
```

Lifecycles

- Az Activity/Fragment életciklus közvetlen elérésére más komponensekből
  - Feliratkozhatunk a változásokra, vagy elkérhetjük az aktuális állapotot (!)



Lifecycles

- Előnyök:
  - Rövidebb kód az életciklussal rendelkező osztályokban

```
override fun onStart() {
    super.onStart()
    compassListener.start()
    gpsService.getUpdates()
    mediaPlayer.resume()
}
```

```
override fun onStop() {
    super.onStop()
    compassListener.stop()
    gpsService.pauseUpdates()
    mediaPlayer.stop()
}
```

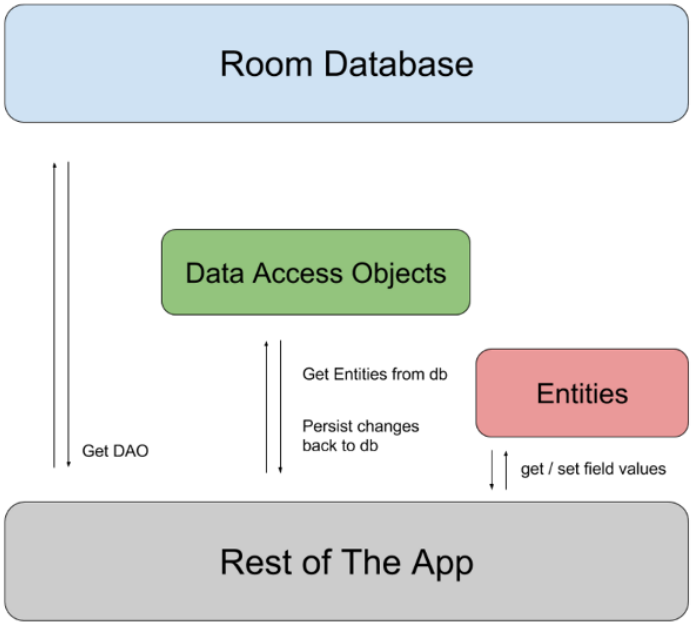
  - A komponens tudja, hogy mikor melyik függvényét kell futtatni
    - Nem felejtjük el meghívni őket amikor kell

Lifecycles

```
class LifecycleAwareComponent : LifecycleObserver {
    @OnLifecycleEvent(Lifecycle.Event.ON_START)
    fun startDoingThings() {
        // ...
    }
    @OnLifecycleEvent(Lifecycle.Event.ON_STOP)
    fun stopDoingThings() {
        // ...
    }
}

...

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    lifecycle.addObserver(LifecycleAwareComponent())
}
```

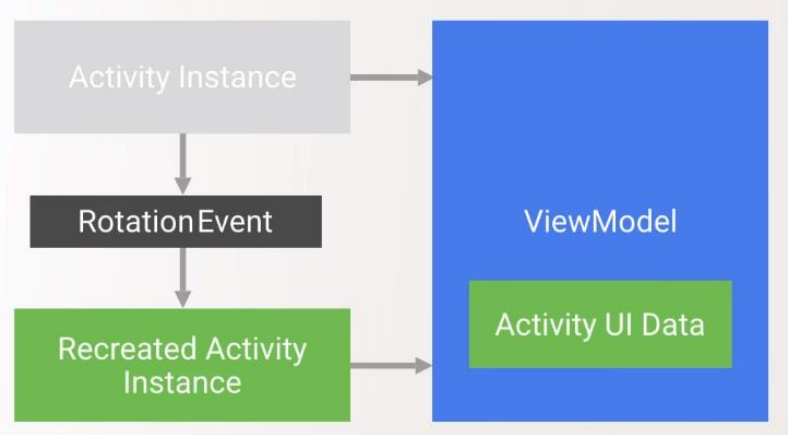
Room Persistence Library

- Absztrakciós réteg az SQLite felett
- SQLite teljes képességeinek használata
- Room architektúra:



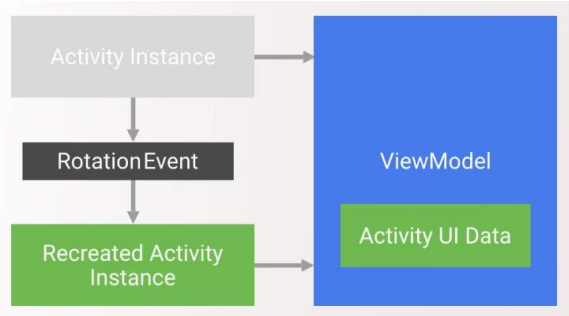
ViewModel

- Felhasználó felület adatainak tárolására, életciklus tudatos módon
  - Tetszőleges adatot tartalmazhat, ami a UI feltöltésére szolgál
  - Túléli a konfiguráció változásokat, az Activity/Fragment újraindulását



ViewModel

- Előnyök:
  - Az Activity-nek a felelőssége így csak a megjelenítés
  - A ViewModel tárolja (ideiglenesen) és menedzseli az adatokat



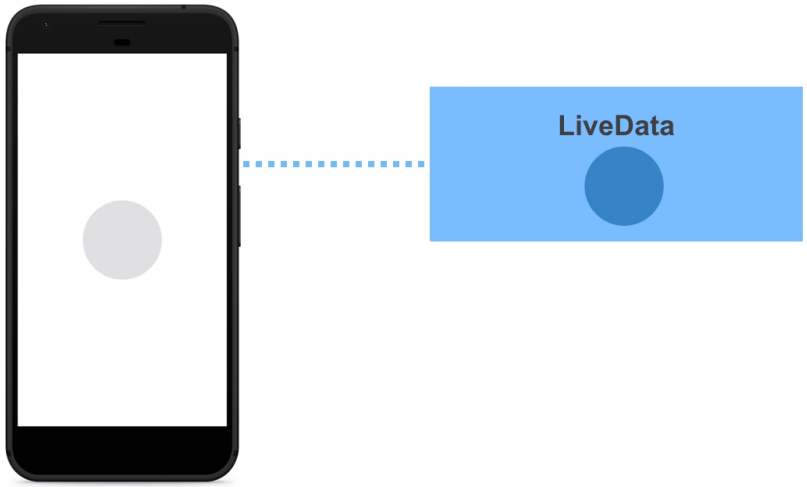

```
class UserViewModel : ViewModel() {
    val user = User(name ="Sally", age = 25)
}

class UserActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val userViewModel =
                ViewModelProviders.of(this).get(UserViewModel::class.java)
    }
}
```

ViewModel

- Tippek: a ViewModel…
  - … *soha* ne tartson referenciát UI elemekre, vagy például egy Activity-re
    - Mivel hosszabb életű az Activity-nél, ez memória szivárgást jelentene
  - … nem helyettesíti a savedInstanceState-et
    - ViewModel: több adatot tárolhat
      - Tárolja a UI megjelenítéséhez szükséges összes adatot
    - savedInstanceState: túléli a konfiguráció változást, és még a teljes alkalmazás halálát is ha túl kevés a memória, de kis mennyiségű adatot tárol
      - Tároljon minimális mennyiségű adatot a UI visszaállításához (például ID-kat)

LiveData

- Lifecycle aware, observable data holder
  - > Életciklus tudatos, megfigyelhető adat tároló…
- Megfigyelhető:
  - > Becsomagol egy értéket, és a feliratkozott megfigyelőit értesíti, ha az érték megváltozott



LiveData

- Jellemzően egy ViewModel tartalmazza
  - Így megfigyelhetjük az adatait, anélkül hogy a ViewModel hivatkozna az Activity-re

```
class UserViewModel : ViewModel() {
    val user: LiveData<User> = ...
}

class UserActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val userViewModel = ViewModelProviders.of(this)
                                    .get(UserViewModel::class.java)

        userViewModel.user.observe(this, Observer { user ->
            // show user on UI
        })
    }
}
```

LiveData

- Életciklus tudatos
  - Csak akkor hívja meg a megfigyelőt, ha az aktív
    - Nem kap frissítést az Activity, ha épp nem látszik
    - Amikor újra láthatóvá válik, csak a legutolsó értéket kapja meg
  - A feliratkozást automatikusan bontja, ha az átadott LifecycleOwner életciklusának vége lesz
    - Nem felejtjük el
    - Nincs memória szivárgás

```
userViewModel.user.observe(this, Observer { user ->
    // show user on UI
})
```

LiveData

- Példány létrehozása:

```
val user: MutableLiveData<User> = MutableLiveData<User>()
```

- Értékek frissítése UI szálról:

```
user.setValue(User("Ann", 56))
user.value = User("Jim", 41)
```

- Értékek frissítése háttérszálról:

```
user.postValue(User("Zoe", 24))
```

Android KTX

Nehezen használható Android API-k

+

A Kotlin nyelvi elemei

=

Android KTX

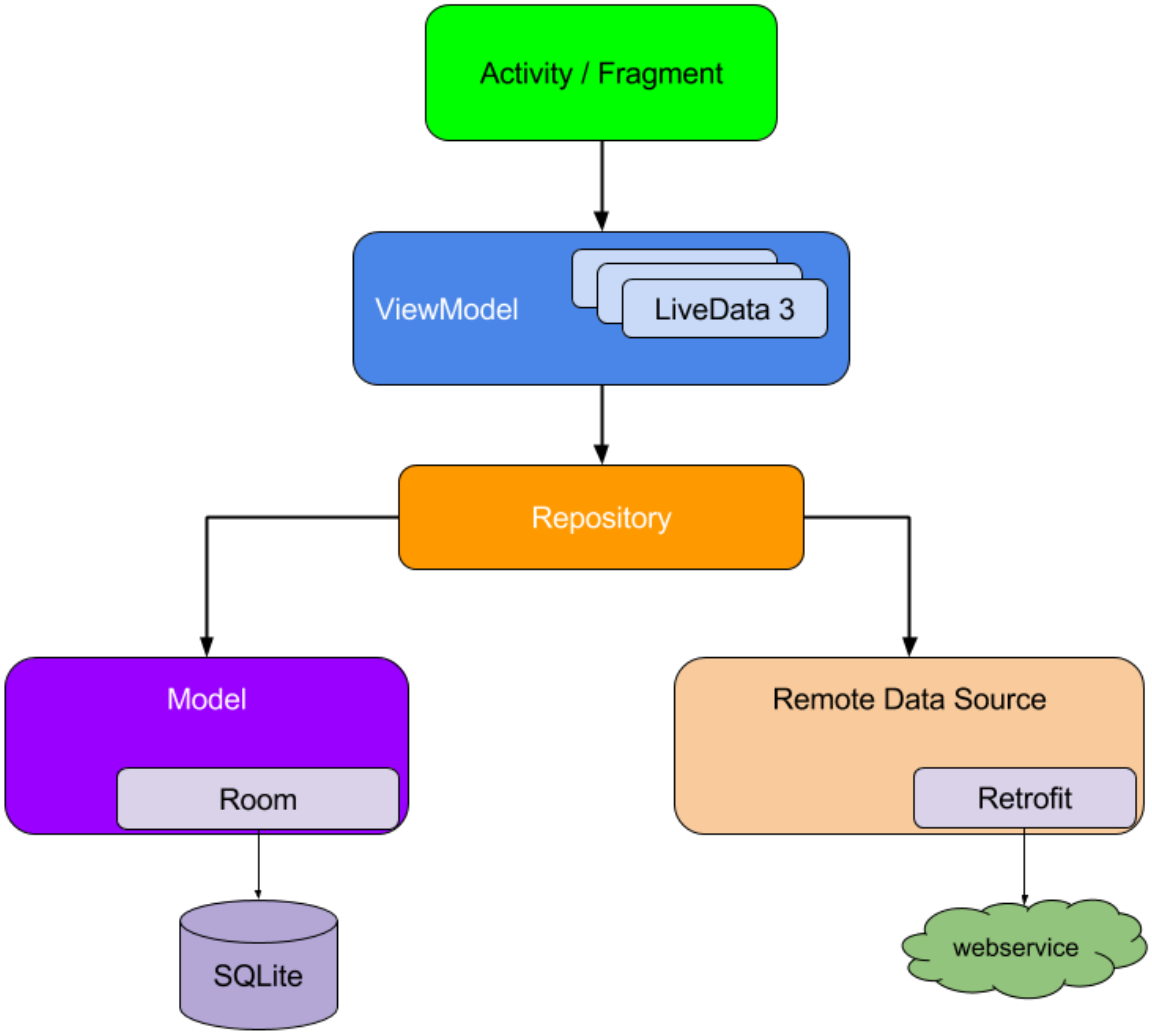
Android KTX

- Extension function és -property gyűjtemény
- Egyszerűbbé teszik az Android API-k használatát
- Irányelvek:
  - Nem tartalmaznak új funkciókat, csak API változtatások Kotlin felhasználóknak
  - Kifejezetten csak Kotlinnal lehetséges dolgok
  - Minimális teljesítmény költséggel (“zero overhead abstractions”)

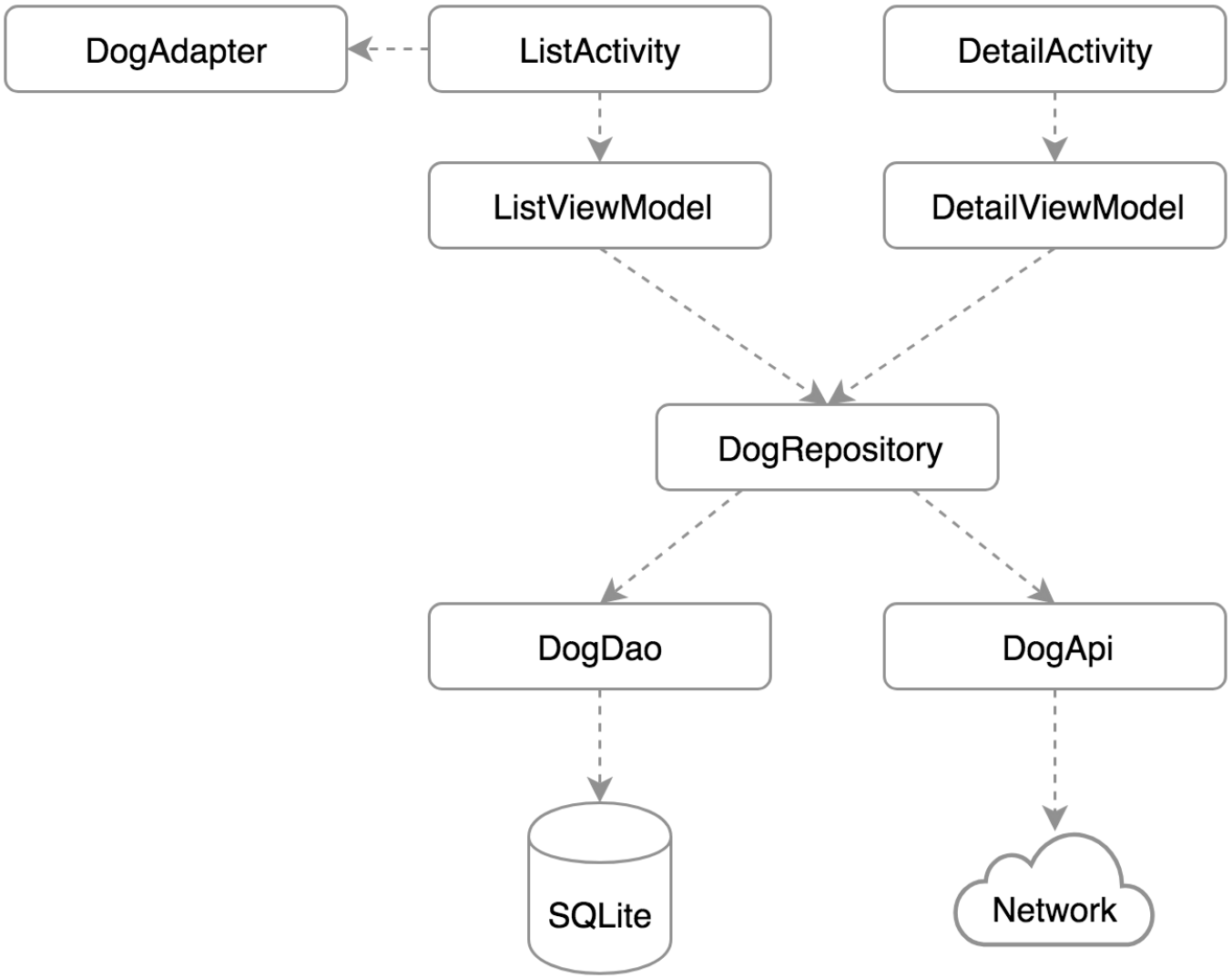
Android KTX

```
textView.visibility = View.VISIBLE
textView.isVisible = true


textView.visibility = if(showText) View.VISIBLE else View.GONE
textView.isVisible = showText


editText.setPadding(20, editText.paddingTop,
                    40, editText.paddingBottom)
editText.updatePadding(left = 20, right = 40)


sharedPreferences
        .edit()
        .putInt("clicks", clickCount)
        .putBoolean("clicked", true)
        .apply()
sharedPreferences.edit {
    putInt("clicks", clickCount)
    putBoolean("clicked", true)
}
```

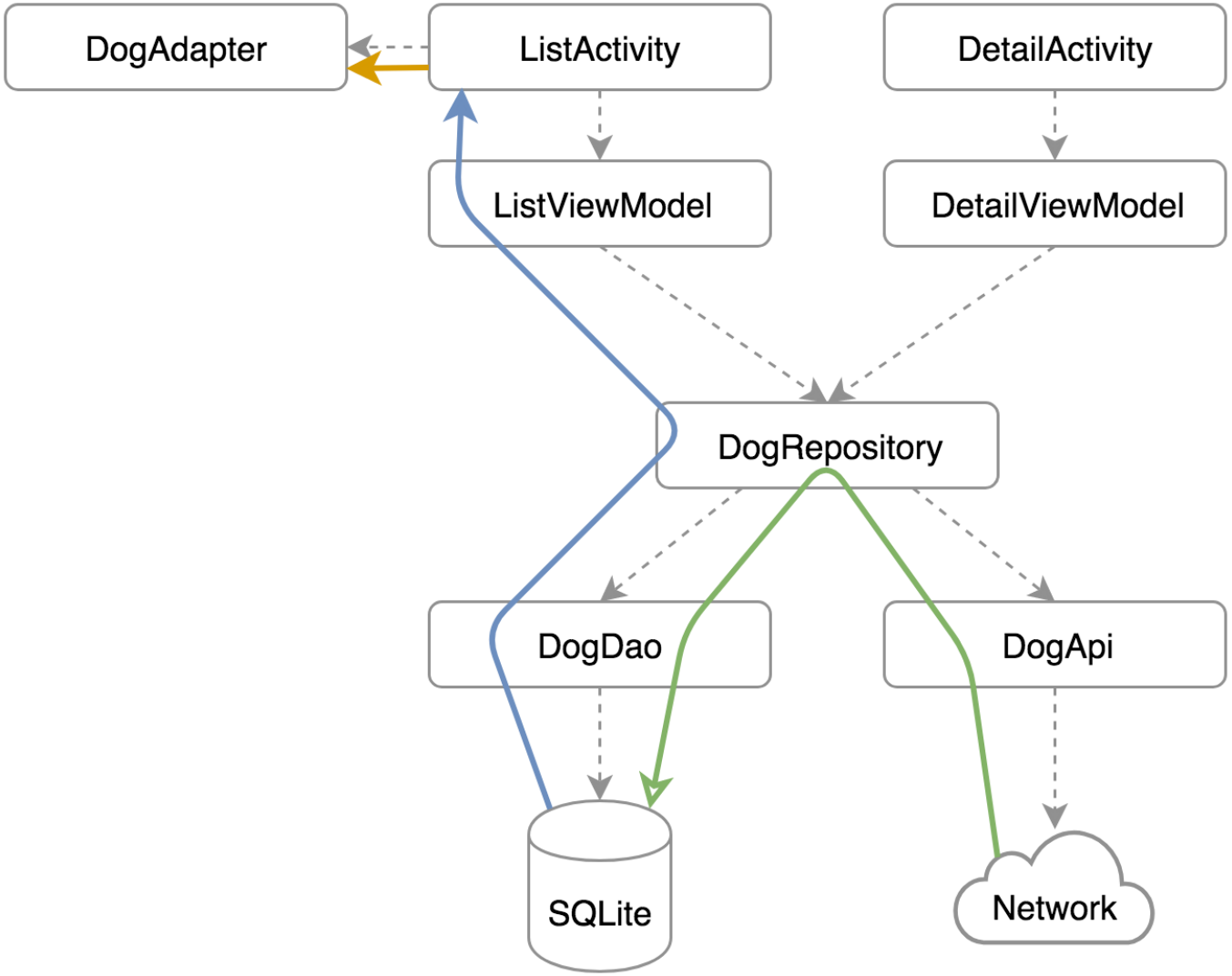
Az ajánlott architektúra



DogCatalog


DogAdapter
ListActivity
DetailActivity
ListViewModel
DetailViewModel
DogRepository
DogDao
DogApi
SQLite
Network

DogCatalog


DogAdapter
ListActivity
DetailActivity
ListViewModel
DetailViewModel
DogRepository
DogDao
DogApi
SQLite
Network

Kotlin nyelv bevezetése

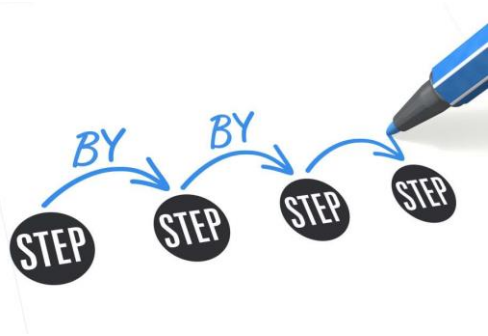
Avagy hogyan győzzem meg a főnököt?

Kotlin nyelv bevezetése 1/2

- Előnyök bemutatása
  - Produktivitás növelése
  - Kevesebb hibalehetőség
  - Motivált fejlesztők
  - Vonzó pozíciók
- Tisztázni az alacsony kockázatot
  - Java tudás teljes mértékben felhasználható
  - Egyszerű és könnyen tanulható nyelv (~)

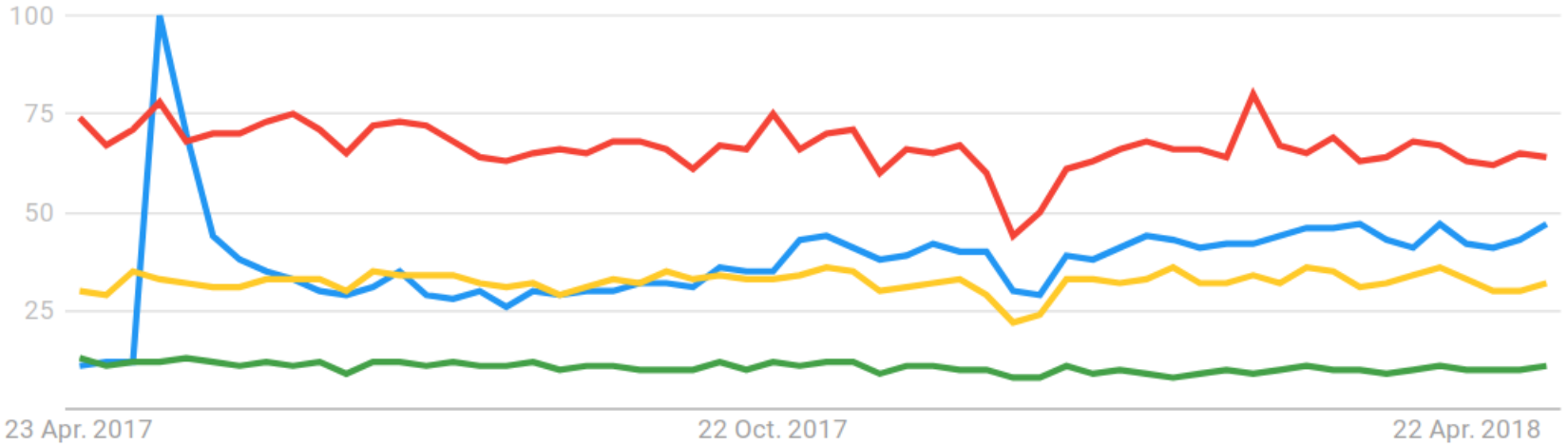
Kotlin nyelv bevezetése 2/2

- Esetleges hátrányok
  - JetBrains-től függ a továbbfejlesztés
  - Alacsony támogatás Eclipse és NetBeans környezetben
- Bevezetés lehetősége kis lépésekben
  - Kezdjünk kis kódbázison, alacsony üzleti érdekeltségű modulban
  - Prototípusok, belső eszközök
  - Kis kompromisszumokra egyszerűbb jóváhagyást szerezni☺
  - Referencia után már könnyebb a használat kiterjesztése



Kotlin terjedése

- Forrás: Google Trends



Köszönöm a figyelmet!

braun.marton@autsoft.hu

AutSoft