

# Mobile (Android) Computing CS371M

Concepts

# Android, a mobile OS

- Android is the world's most popular mobile OS
  - Controls phones, tablets, wearable

# Working with a programming language

- What does it mean to write idiomatic code?
  - In Kotlin, try to avoid explicit null checks
    - `highScores.sortWith(compareByDescending<Score>{it.score}.thenBy{it.name})`
    - `File("file.txt").copyTo(File("target_file.txt"));`
- Lateinit

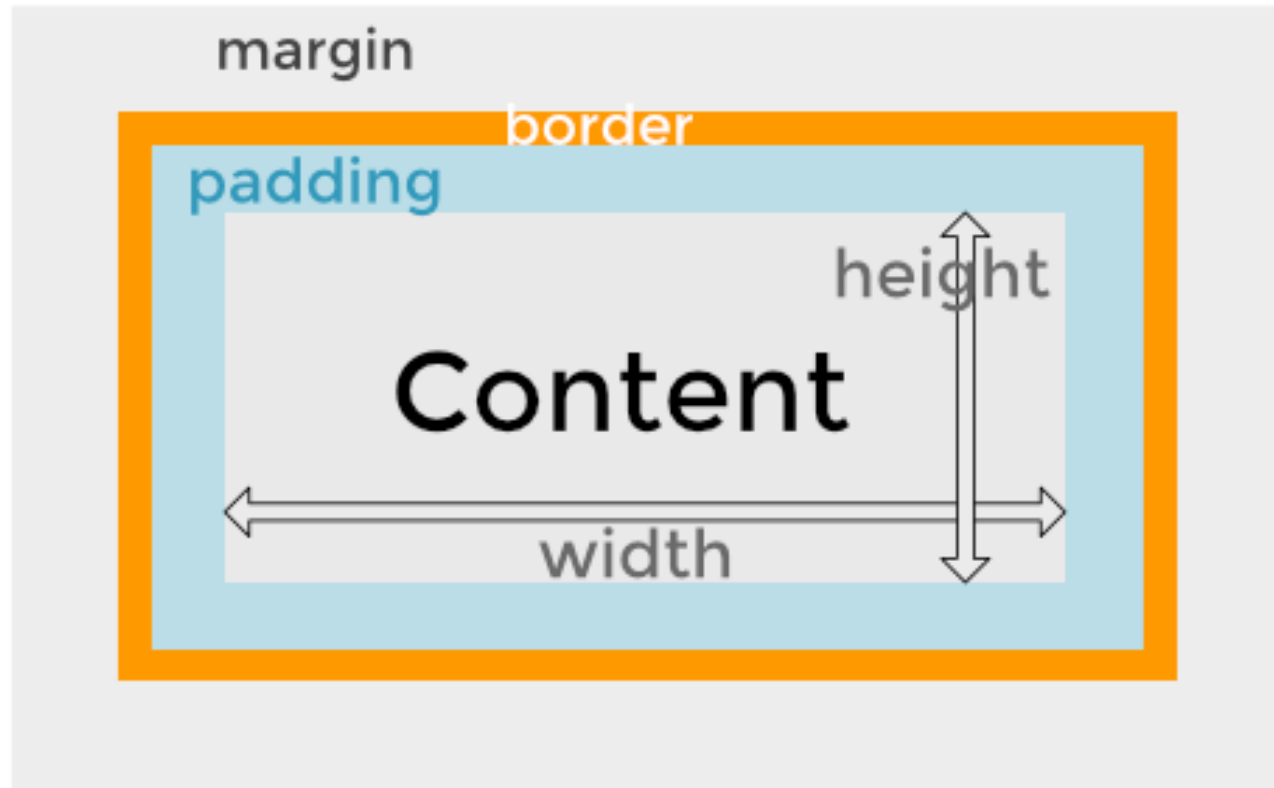
# How to succeed at programming

- Start with casting spells
  - Fake it till you make it
  - Pattern match from other code
- Manage your frustration
  - Keep trying new angles
  - Write notes
  - Explain your problem in an email (don't have to send it)
- Am I doing things correctly and efficiently?
  - Get something working quickly, then improve
  - Move from one working version to the next (you can always roll back)
  - Add functionality bit by bit

# Android Project Checklist

- Code up to date
- Layout files up to date
- Gradle file up to date, has proper dependences
- AndroidManifest.xml
- Drawables
- Values, like strings and colors, styles, etc.

# Widgets



# Starting a new Activity

- Activities in Android control the screen
  - Apps typically have several screens
    - E.g., Game play, high scores, settings
- Starting an activity > calling a function
  - Android runtime constructs the new activity
    - Eventually calls onCreate
  - How do we pass parameters?
- Returning from an activity isn't a function return
  - Where do we return to?

# Intents: Specifying operations

- An intent is an abstract description of an operation to be performed (Android docs)
  - Implicit intent: abstract action (e.g., dial this number)
  - Explicit intent: execute this class
- Provides a mechanism to pass parameters
  - Key/value bundles

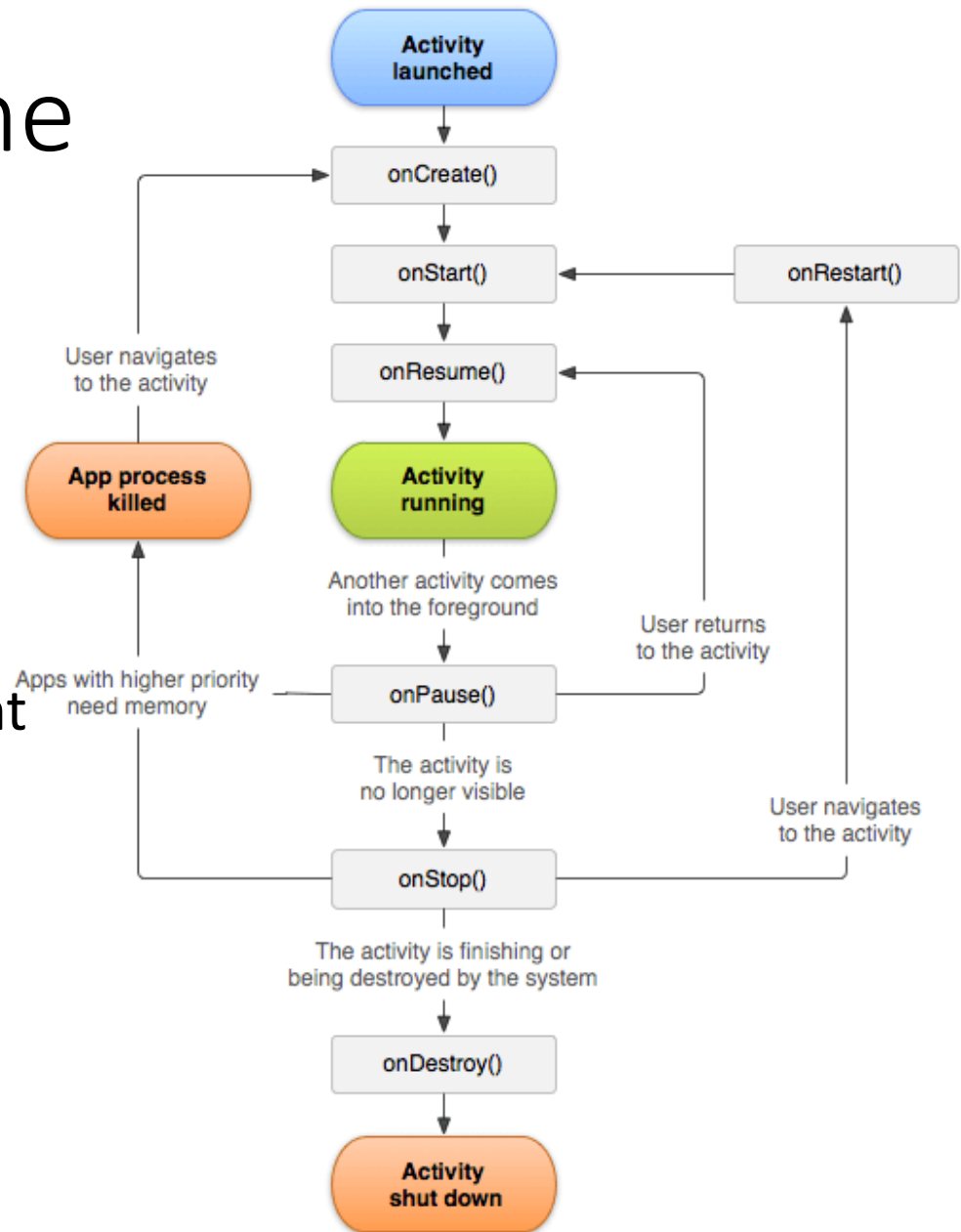
```
// Get the Intent that called for this Activity to open  
val activityThatCalled = intent  
// Get the data that was sent  
val callingBundle = activityThatCalled.extras  
callingTV.text = callingBundle?.getString("callingActivity")
```

- Used on entry and exit of Activities

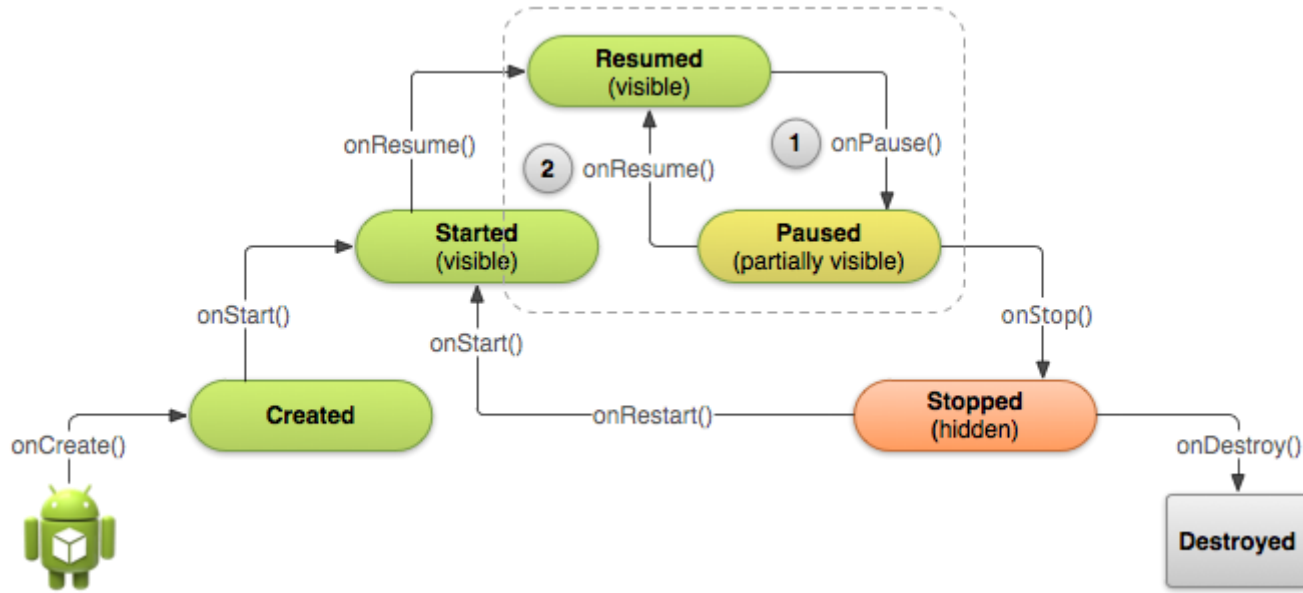


# Activity lifetime

- State machine for activities
  - Android tells you about significant events
  - Your app might want to clean up when it goes to the background
- Trend away from explicit callbacks



# Activity lifecycle

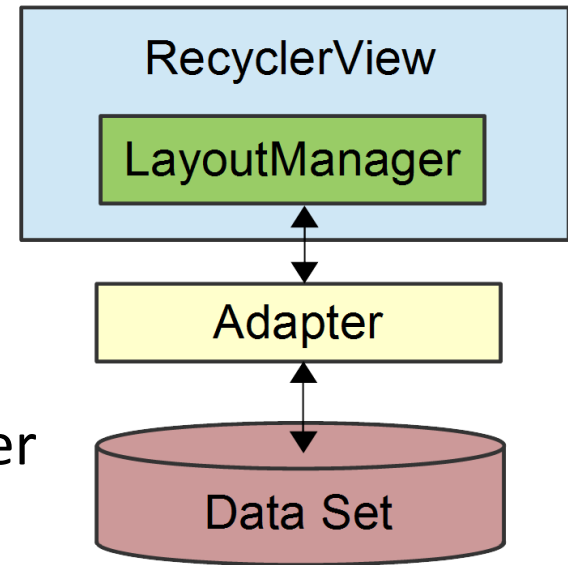


# Lists

- Simple case: One widget == one program state
  - A button has one `onClick` listener
  - A slider is on or off
  - A `TextView` contains a string
  - Everything about these widgets is visible all the time
- Lists
  - Lists contain many elements
  - Only some of them are on the screen at any time
  - How do we deal with many items?
  - How do we display lists efficiently?

# RecyclerView

- RecyclerViews display lists
  - RecyclerView contains a LayoutManager
    - Examples: Linear Layout, Grid Layout, etc.
    - Managers can nest, e.g., horizontal layout inside a vertical
  - RecyclerView only displays part of the list
    - An Adapter sees the full list and the part of the list currently on screen
  - Efficiency via ViewHolders
    - If 6 list elements are visible on the screen at once, we have 6 ViewHolders

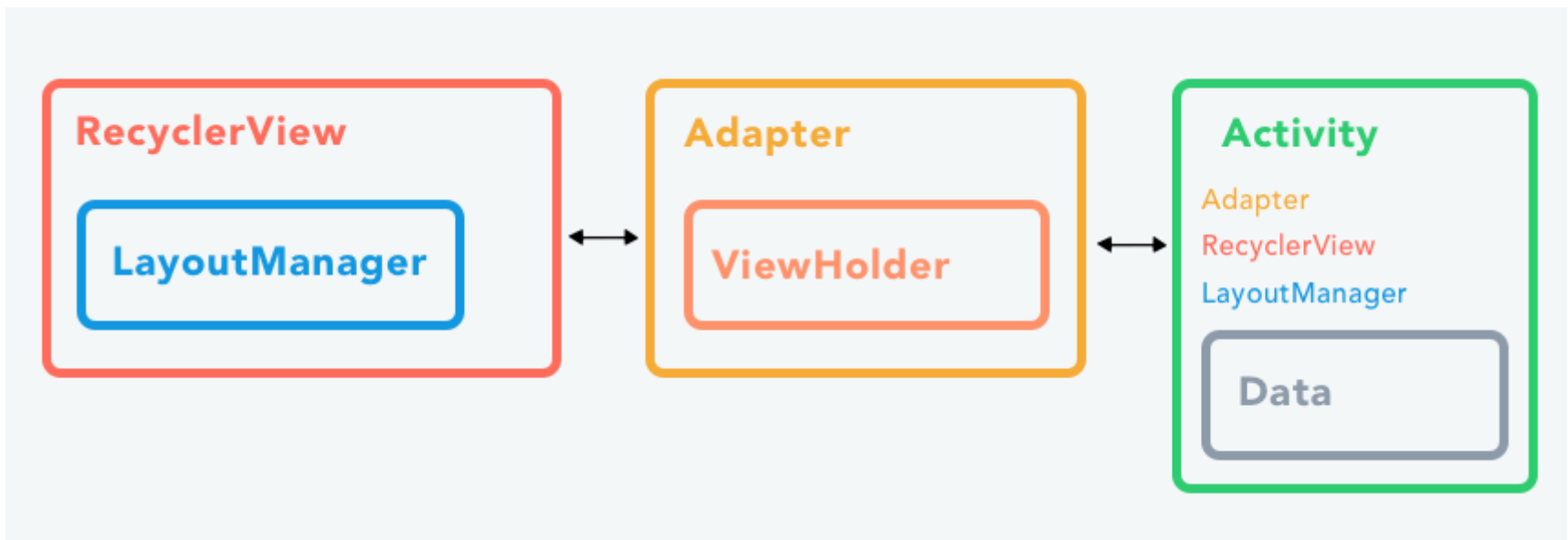


Many images from

<https://learn.microsoft.com/en-us/xamarin/android/user-interface/layouts/recycler-view/parts-and-functionality>

# Can't I just have a simple list?

- Before RecyclerView was ListView
  - Tried to be simple but failed
  - To avoid API fatigue, we are jumping straight to RVs



# Life is short, lists are long

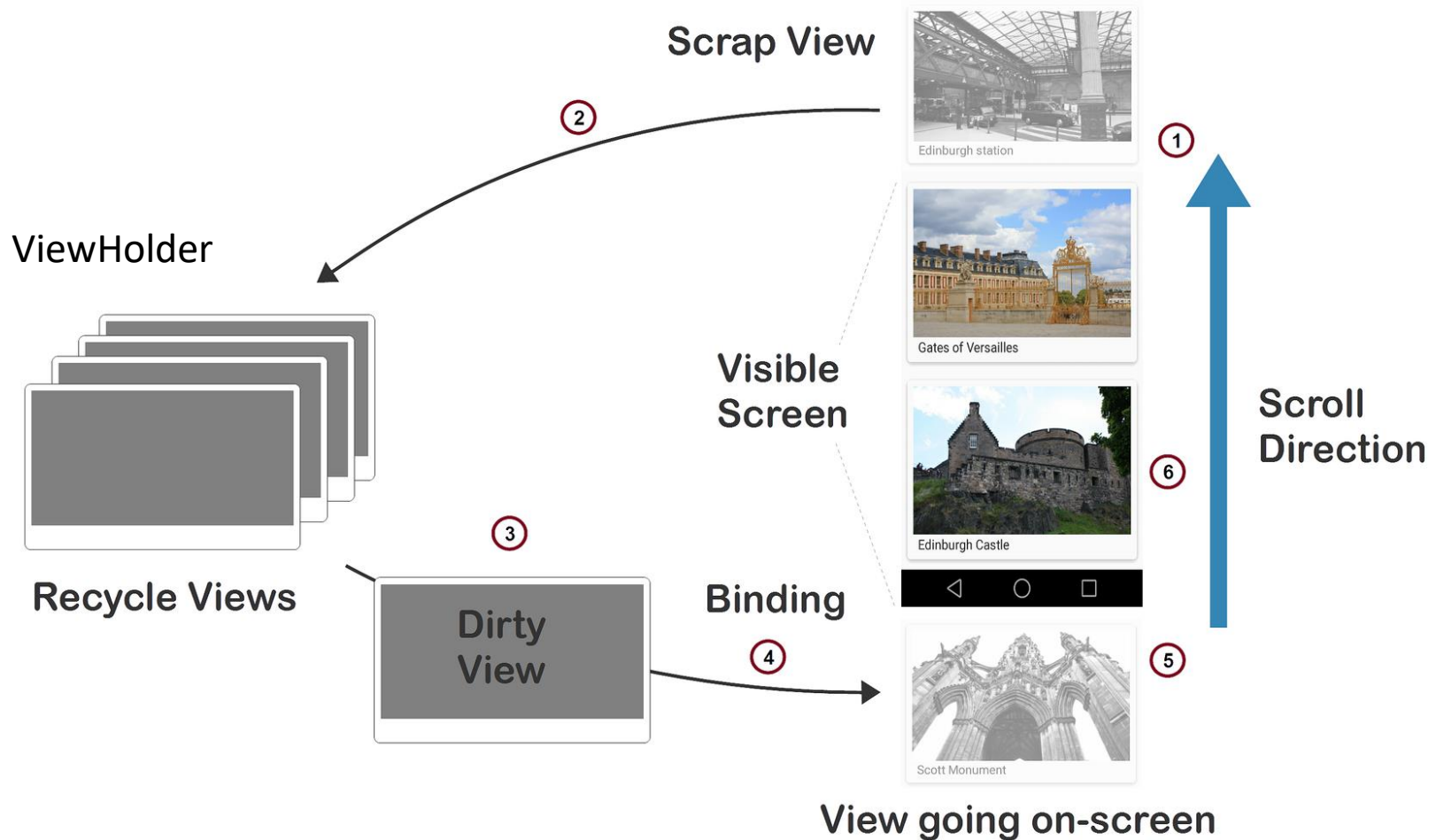
- Layout up to this point has been simple
  - A TextView object might have some default text
  - Then we change the text in our program
    - `textView.text = "Watch this space"`
- But how to we display a scrollable list of 100 items?
  - `recyclerView.list[1] = "Item 1"   ????`
  - `list = {"Item 0", "Item 1"};   ????`  
`screenView[1] = list[1]   ????`
- Spoiler alert, not like this, but using the same concepts

# Managing on-screen rows

- What is a list on screen?
  - A sequence of rows
- How do I lay out a row?
  - Use an XML file
- How do I draw a row on screen?
  - Inflate a view binding object
  - E.g., row.xml becomes RowBinding
- Inflation is expensive though, so recycle
  - Put view binding in ViewHolder & recycle them as user scrolls

# Limited pool of inflated views

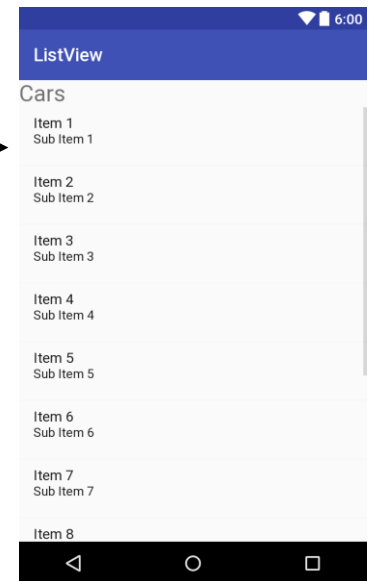
- Only inflate enough views to fill screen





# Adapting a list to your screen

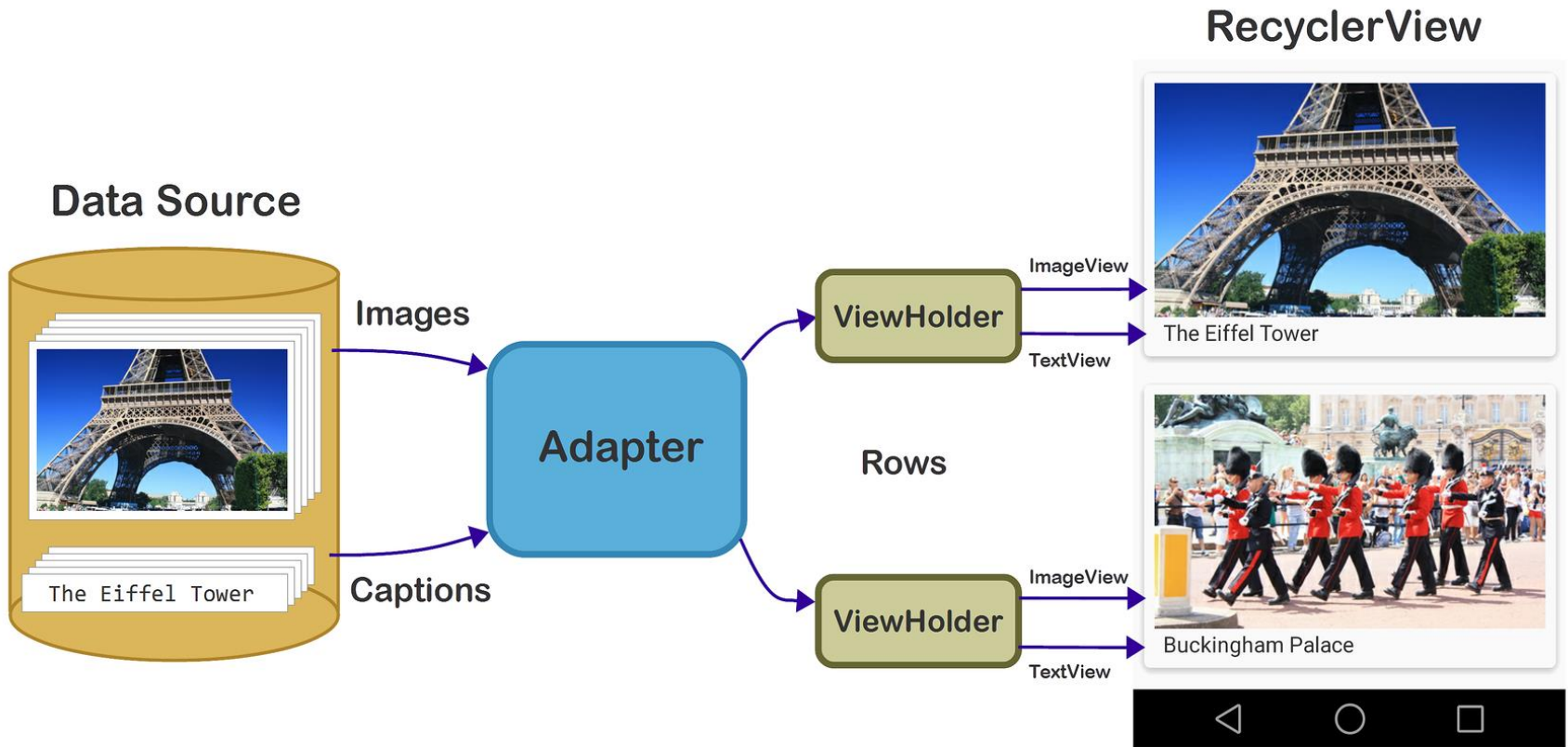
- The list is long and is in your code
  - `val list = List<Data>(100) {it}`
  - `val list = repository.fetchData()`
- Some elements are on the screen
  - E.g., 8 items in screen over here →
- The adapter marries your list to screen
  - Manage 8 on-screen row views
  - **Inflate**: create a view for an on-screen row
  - **Bind**: Put list element 9 into a view holder
    - This bind is different from view binding



# Recycler View APIs

- row.xml has the row layout
  - Compiler generated RowBinding object via view binding
- Inherit from RecyclerView.Adapter
  - `class RecyclerViewAdapter : RecyclerView.Adapter<RecyclerViewAdapter.VH>()`
  - But now must define inner VH class
- ViewHolder holds view, which is in binding class
  - `inner class VH(val rowBinding: RowBinding) : RecyclerView.ViewHolder(rowBinding.root)`
- Inflate enough rows to fit on screen, no more
  - `onCreateViewHolder`
- Set holder.rowBinding to contents in list[position]
  - `onBindViewHolder(holder: VH, position: Int)`

# One more zoomed-out view



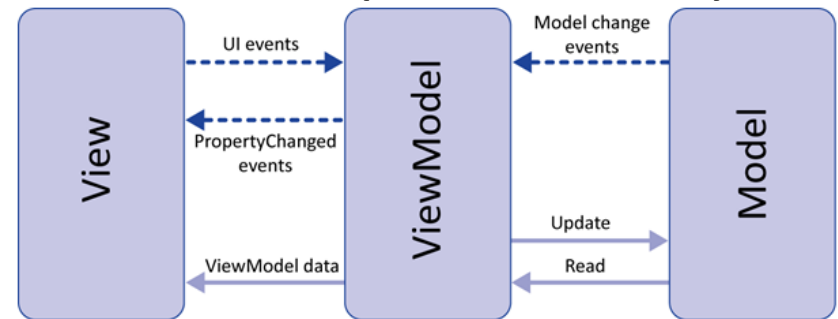
# RecyclerView glossary

- Row layout: XML file describing a row
- Adapter: Views for on-screen rows from full list
- Position: The position of a data item within the list
- Inflate: XML → View (uses view binding)
- ViewHolder: View/binding for one on-screen row
- Binding view holder: Put data for list element X into view holder when user scrolls to item X
- Recycle: ViewHolders are reused as the user scrolls. Minimize inflation, which requires lots of CPU & memory

# Why is activity/fragment killed?

- **1. You meant to navigate away permanently:** The user navigates away or explicitly closes the activity — say by pushing the back button or triggering some code that calls `finish()`. *The activity is permanently gone.*
- **2. There is a configuration change:** The user rotates the device or does some other configuration change. *The activity needs to be immediately rebuilt.*
- **3. The app is put in the background and its process is killed:** This happens when the device is running low on memory and needs to quickly free some up. *When the user navigates back to your app, the activity will need to be rebuilt.*

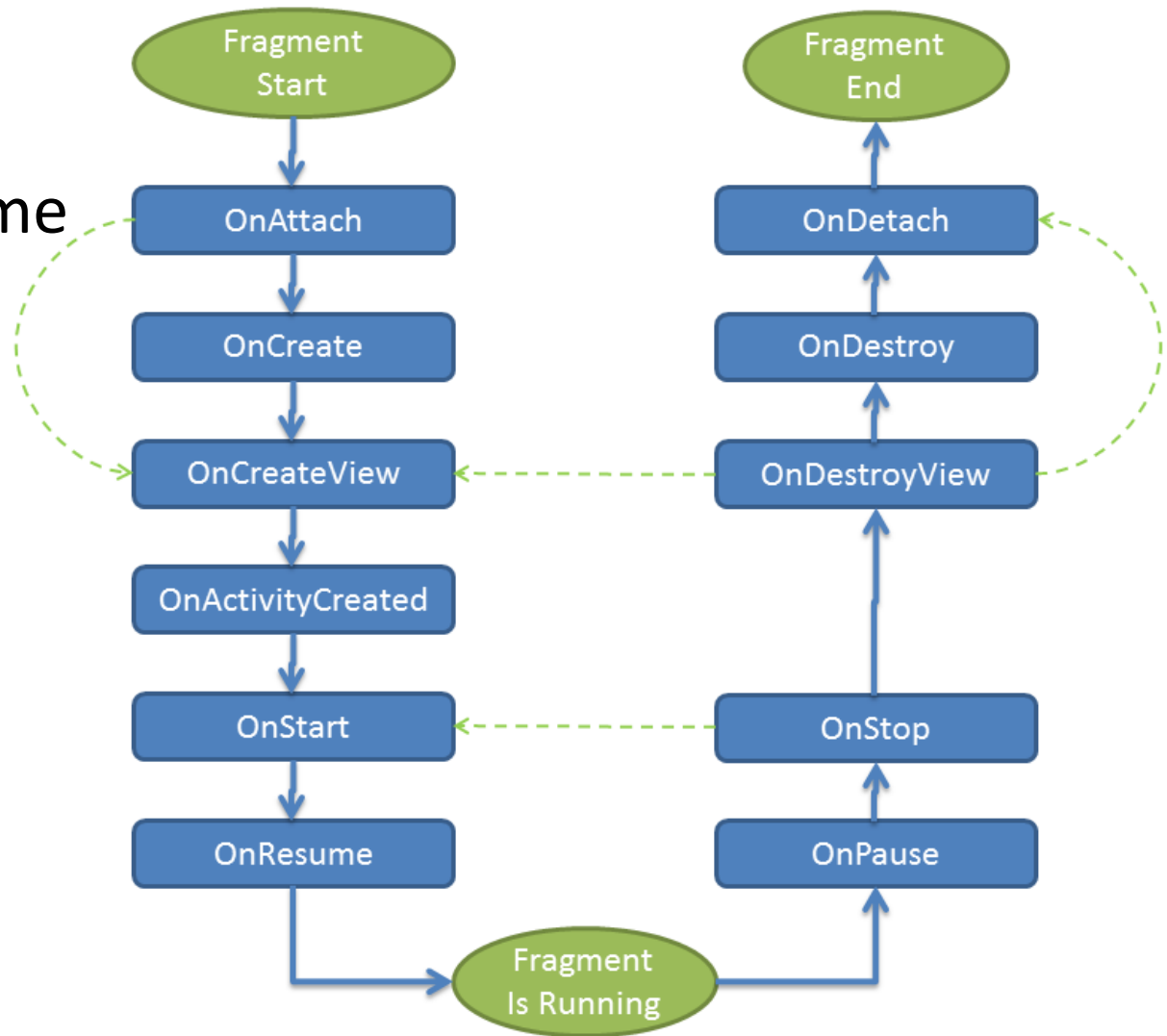
# View, ViewModel, Model (MVVM)



- View is your fragment
  - It manages the screen, inflates XML
  - Observes LiveData, reflects changes to the display
  - Observes display, feeds events to ViewModel
- ViewModel manages LiveData, exports it.
  - Contains most of the application logic.
- Model is the data
  - Source from network or database/files
- Might feel weird at first, but wisdom becomes clear

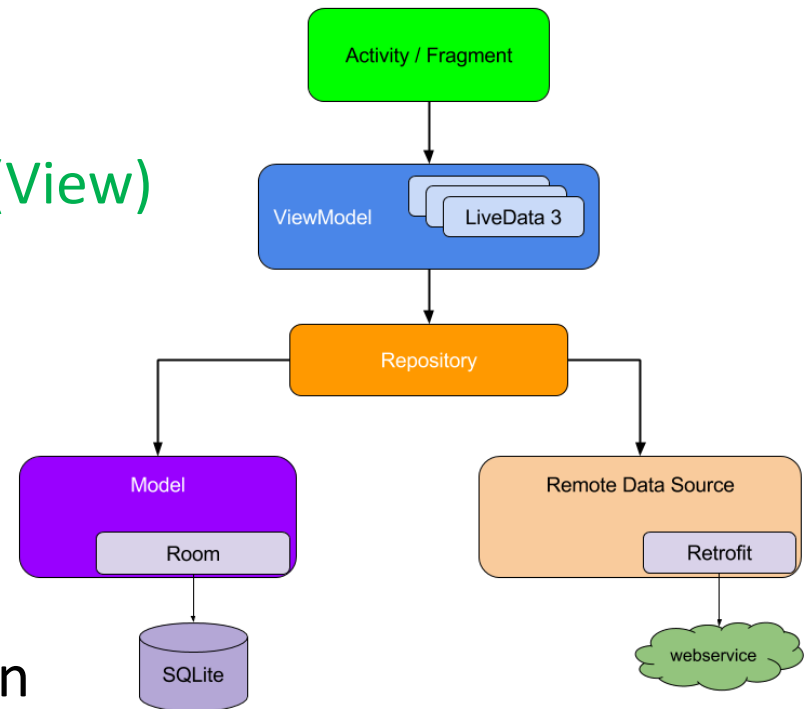
# Fragments

- Fragment lifetime



# ViewModels, MVVM<sub>(model, view, viewModel)</sub>

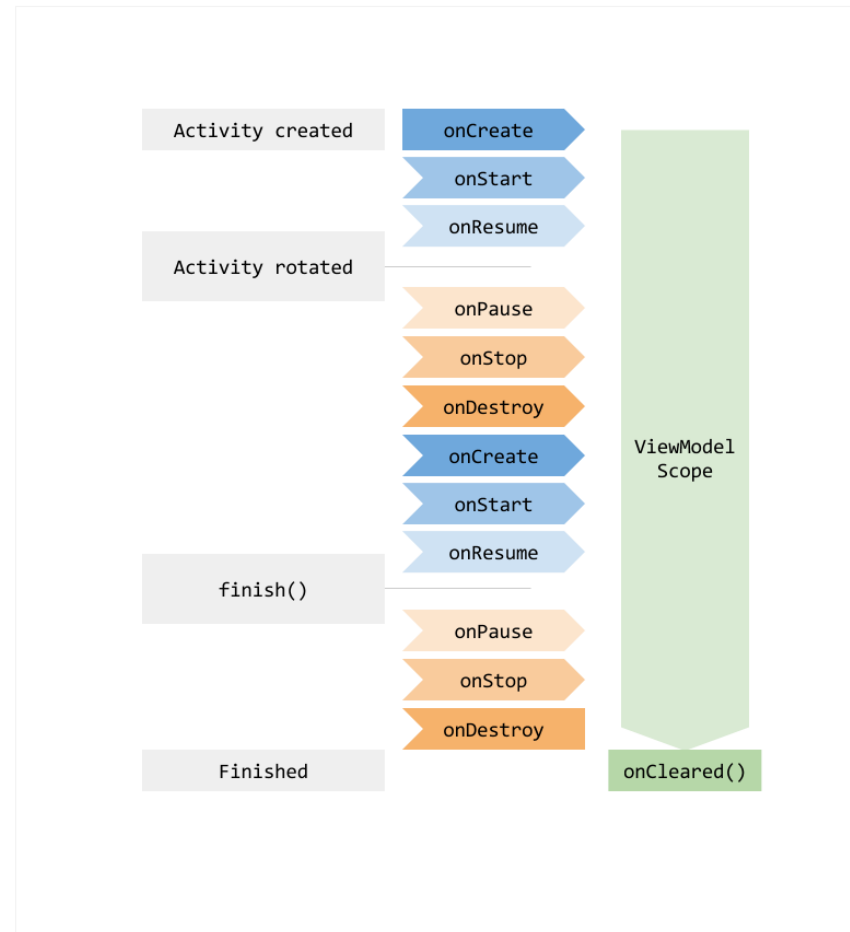
- Separate logic from display
  - Display in activity/fragment (**View**)
  - “Application logic” goes in ViewModel (**ViewModel**)
  - Model holds the “real data” (**Model**)
- Encourages an app style
  - Raises the level of abstraction
  - Enables powerful apps quickly





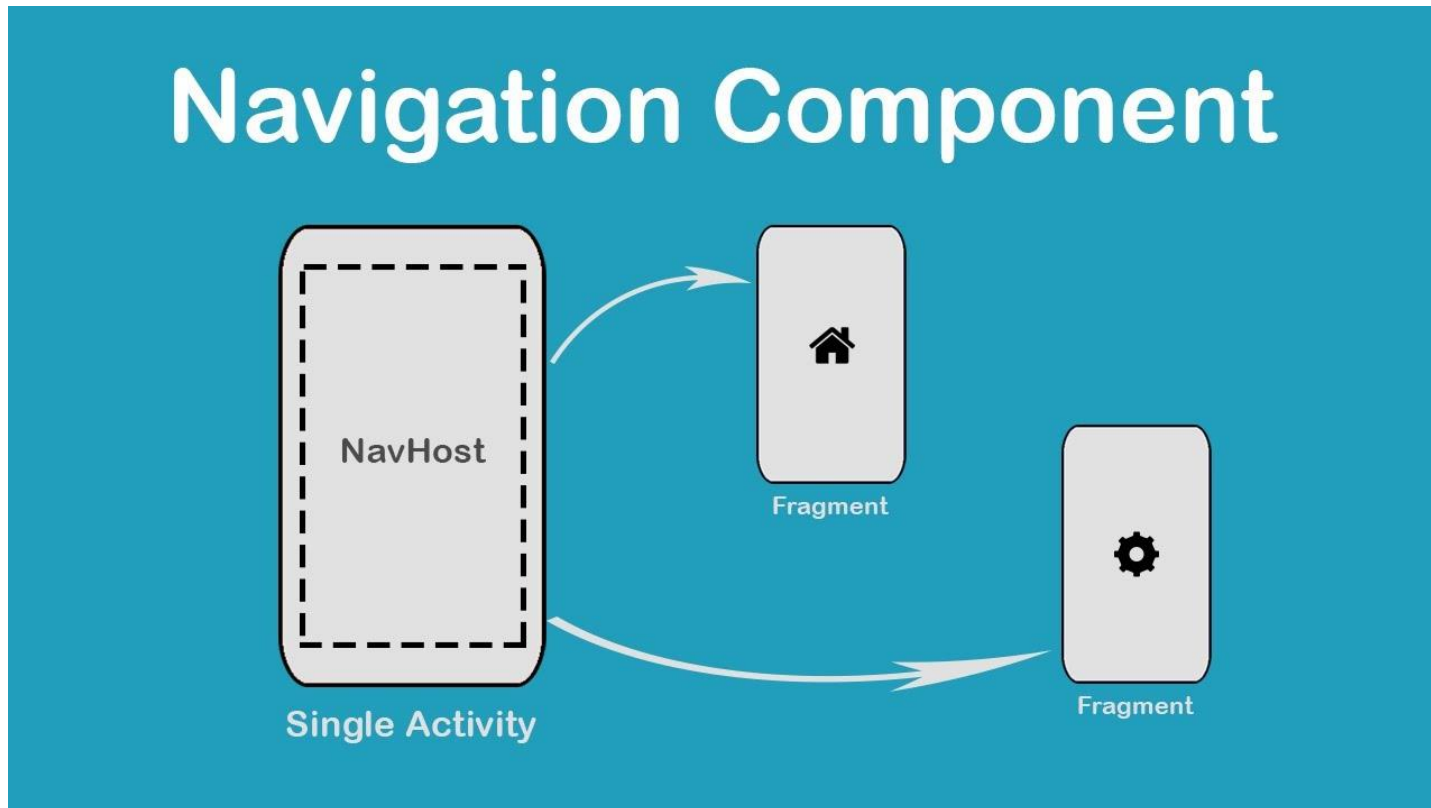
# ViewModels have long lifetimes

- Data in a ViewModel is in memory even if device rotates or is paused
  - Fewer surprises at runtime!
  - You still sometimes need `onSaveInstanceState()` for persistence



# Navigation

- One activity moves among fragments



# Navigation key ideas

- Fragments and actions in a single graph
  - Actions transition among fragments
- In a fragment: `findNavController()`
- In an activity it is more complicated
  - There can be multiple controllers
  - `findNavController(R.id.nav_host_fragment_activity_main)` - ID of <fragment>
  - `view.findNavController()` - use view
- Exit a fragment: `findNavController().popBackStack()`
  - All fragments form a stack

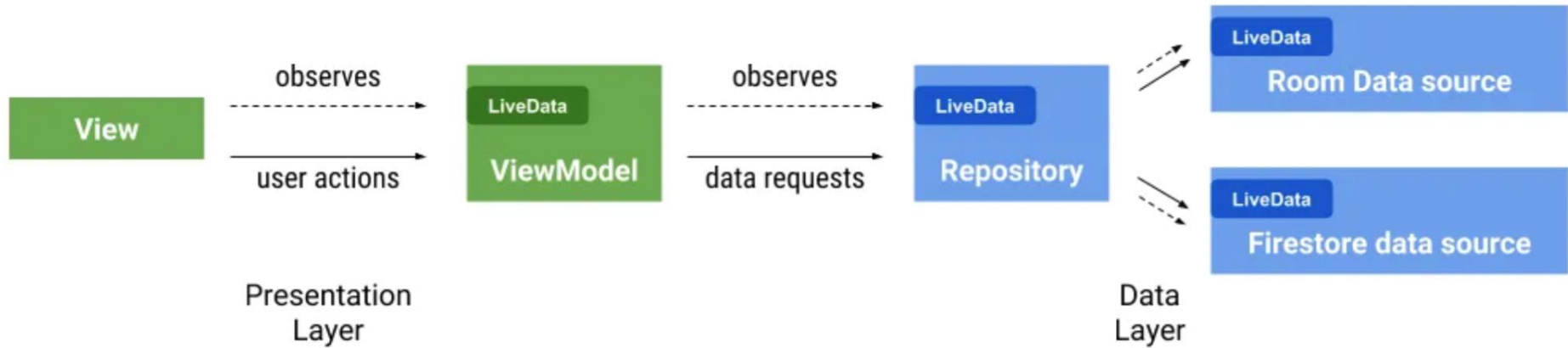
# Navigation: moving around

- To move to another fragment
  - `view.findNavController().navigate(R.id.targetFragment)`
  - `findNavController(R.id.view).navigate(R.id.targetFragment)`
  - Where `targetFragment` is an id in your nav graph
  - Nav graph is in `navigation/nav_graph.xml`
- To pass arguments use a Directions objects
  - Compiler generates them from nav graph—type safe!
  - An action carries the parameters
  - ```
val action = SpecifyAmountFragmentDirections  
    .actionSpecifyAmountFragmentToConfirmationFragment(foo)
```
  - `findNavController().navigate(action)`

# Live data

- View models contain the state of your app
- Views get the state from view model
- What is the problem?
  - How does the view know when state changes?
  - Could provide explicit callbacks
  - But what happens if view listening to changes dies?
  - Register callback on create, deregister on destroy
  - Bugs!

# Live data



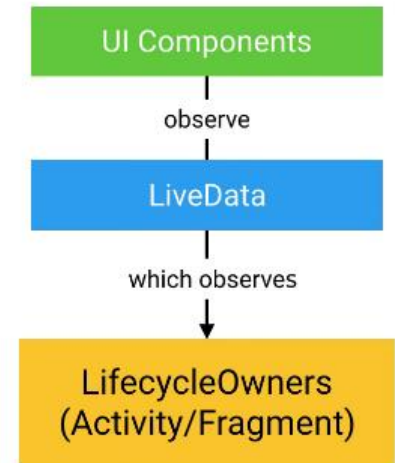
- Live data lives in view model
- View observes live data
  - Modifies view model state based on user actions
- Repository/model also has live data
  - View model updates its data from repository
- App responds to data changes minimal coupling

# Live data

- ```
// In the view model
private var liveList = MutableLiveData()
fun observeLiveQuotes():
LiveData<List<Quote>> {
    return liveList
}
```
- ```
// In the view
viewModel.observeLiveQuotes().observe(viewLifecycleOwner) { quoteList ->
    adapter.submitList(quoteList)
    adapter.notifyDataSetChanged()
}
```

# MVVM, lifetime and scope

- LiveData allows communication without dependence
  - Multiple observers ok
  - Observers can die, be reborn, no code changes
- ViewModels live as long as application
  - Fragment (can die when goes off screen)
  - Activity (longer than fragment, but can die off screen)
  - Application (activities can share state and communicate)





# Networking(!)

- What “language” do we speak with servers?
  - Representational state transfer (REST)
  - Use text (JSON)
  - Servers are “stateless”
    - Requests give them all the context they need
- What tools do we need in Android
  - Retrofit2: Talk to servers
  - Gson: parse JSON
  - Coroutines & Dispatchers: Fetch on background thread
  - Glide: display & cache images given a URL
    - Glide is independent, but fetching images is common

# Coroutine dispatchers

- Dispatcher.Main
  - Main thread
  - Must be quick: required if you have to update the UI
- Dispatcher.Default
  - Background thread, possibly on other processor
  - For background work, like diffing lists, image filters
- Dispatcher.IO
  - Background thread, possibly on other processor
  - For network access or file system access

# Specifying coroutine dispatchers

- `withContext` – allows you to switch dispatchers

```
withContext (Dispatchers.Main) {  
    textView.text = "hi there" }
```

- This also works!

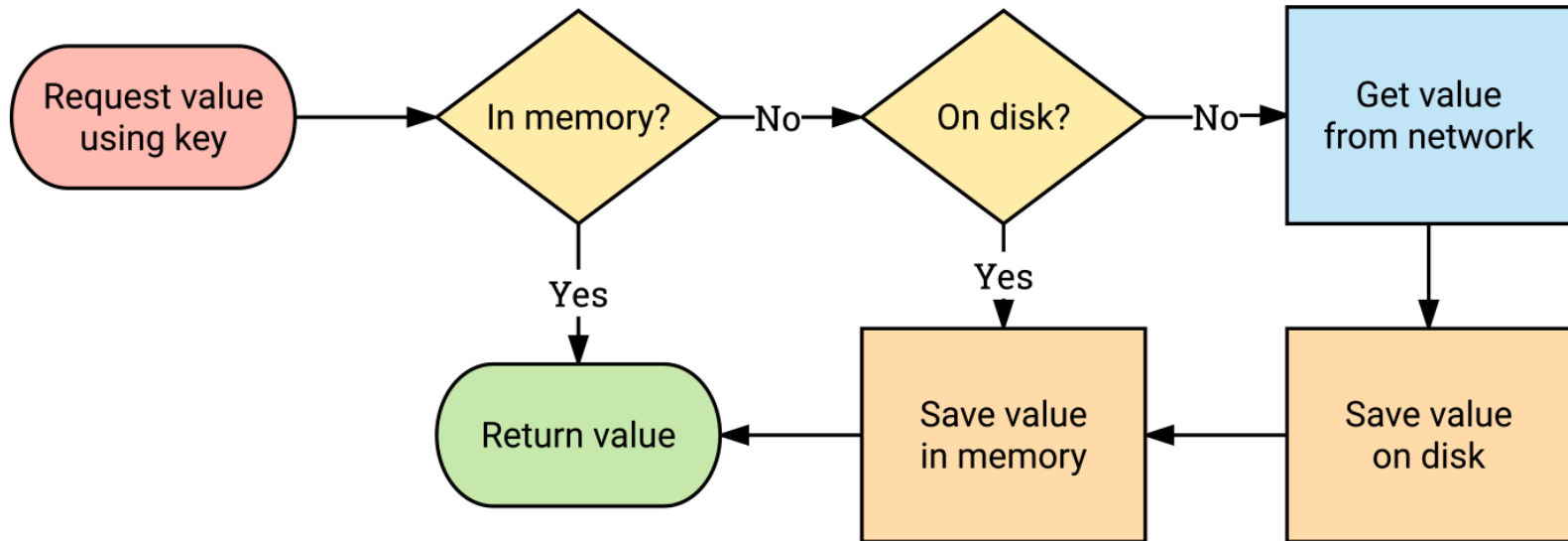
```
runOnUiThread {...}
```

- Launch with `viewModelScope.launch(`  
`context = viewModelScope.coroutineContext`  
`+ Dispatchers.IO) {`  
scope

- Live data works with threads

- Main thread can use `setValue` or `.value`
- Background threads **must** use `postValue`

# Glide for caching images



- Use a key, like the image URL
- Library handles storing images locally

# Retrofit2 for web services

- Retrofit2 turns your HTTP API into a Kotlin interface.
- ```
interface GitHubApi {  
    @GET("users/{user}/repos")  
    listRepos(@Path("user") String user) :  
    List<Repo>
```
- Retrofit2 generates the implementation of `listRepos` (not you!)
  - Bridge between Kotlin interface (`listRepos`)
  - And http call  
(<https://api.github.com/users/octocat/repos>)

# Retrofit2 for web services

- ```
interface GitHubApi {  
    @GET ("users/{user}/repos&sort=updated")  
    listRepos (@Path ("user") String user) :  
    List<Repo>
```
- **@GET**
  - Defines the path of the URL
    - Retrofit2 can plug in values (user, above)
    - String can contain parameters (sort=updated, above)
- **@Path** fills in path component from parameter
- You need to define return type
  - Look at JSON given by the web service

# Retrofit2 for web services

```
interface GitHubApi{
    @GET("users/{user}/repos&sort=updated")
    listRepos(@Path("user") String user) : List<Repo>

    companion object Factory {
        fun create(): GitHubApi {
            val retrofit: Retrofit = Retrofit.Builder()
                .addConverterFactory(GsonConverterFactory.create())
                // Must end in /!
                .baseUrl("https://api.github.com/")
                .build()
            return retrofit.create(GitHubApi::class.java)
        }
    }
}
```

# JSON (JavaScript Object Notation) Example

```
{  
  "Title": "The Cuckoo's Calling",  
  "Author": "Robert Galbraith",  
  "Detail": { "Pages": 494 },  
  "Price": [  
    { "type": "Hardcover",  
      "price": 16.65 },  
    { "type": "Kidle Edition",  
      "price": 7.03 }  
  ]  
}
```

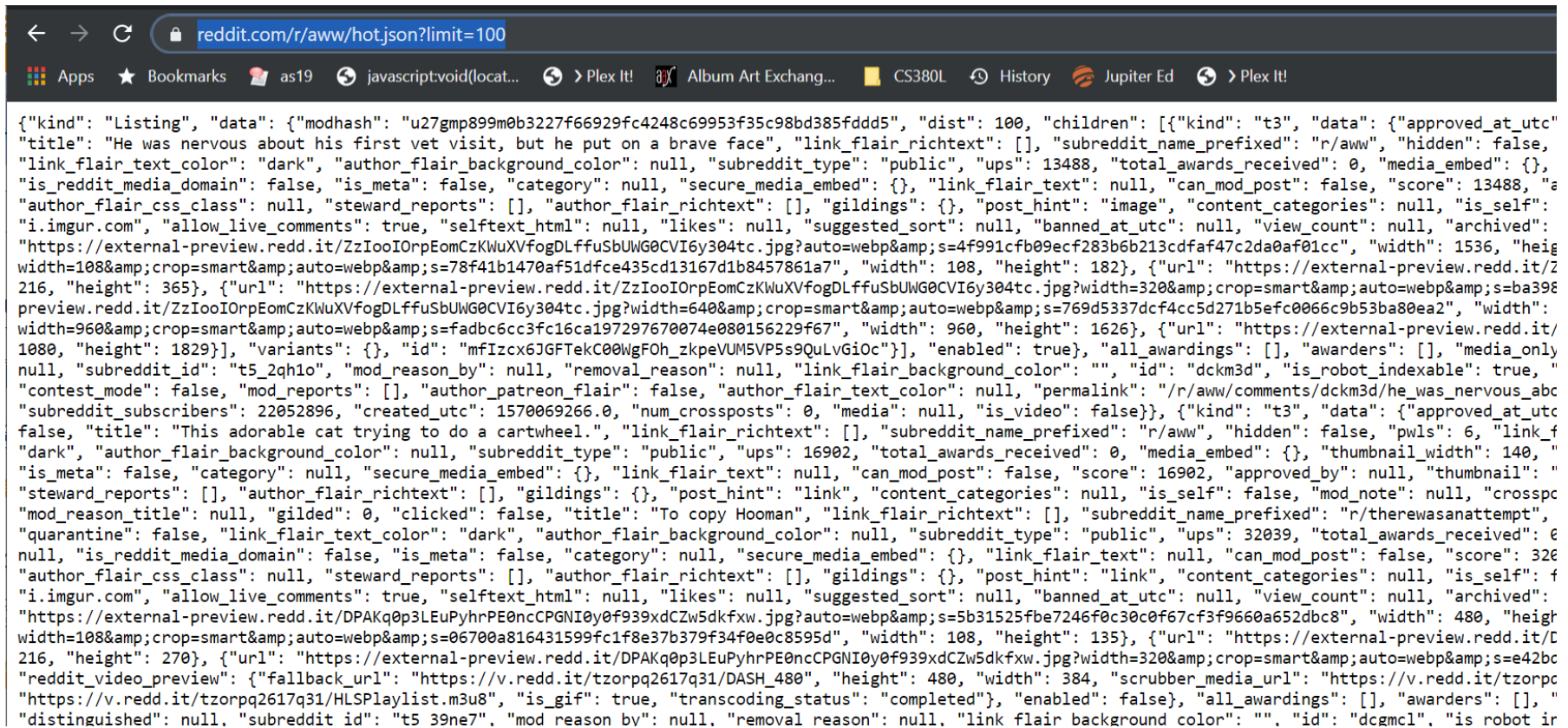
<http://www.w3resource.com/JSON/introduction.php>



# Elements of JSON

- Four basic and built-in data types in JSON.
  - Strings, numbers, Booleans (i.e true and false) and null.
  - These can be used as values
- Two structured data types
- Objects are wrapped within '{' and '}'.
  - Objects are list of label-value pairs.
- Arrays are enclosed by '[' and ']'.
  - Arrays are list of values.
- Both objects and arrays can be nested.

# Web services often return JSON



```
{"kind": "Listing", "data": {"modhash": "u27gmp899m0b3227f66929fc4248c69953f35c98bd385fddd5", "dist": 100, "children": [{"kind": "t3", "data": {"approved_at_utc": "title": "He was nervous about his first vet visit, but he put on a brave face", "link_flair_richtext": [], "subreddit_name_prefixed": "r/aww", "hidden": false, "link_flair_text_color": "dark", "author_flair_background_color": null, "subreddit_type": "public", "ups": 13488, "total_awards_received": 0, "media_embed": {}, "is_reddit_media_domain": false, "is_meta": false, "category": null, "secure_media_embed": {}, "link_flair_text": null, "can_mod_post": false, "score": 13488, "author_flair_css_class": null, "steward_reports": [], "author_flair_richtext": [], "gildings": {}, "post_hint": "image", "content_categories": null, "is_self": "i.imgur.com", "allow_live_comments": true, "selftext_html": null, "likes": null, "suggested_sort": null, "banned_at_utc": null, "view_count": null, "archived": "https://external-preview.redd.it/ZzIooIOrpEomCzKwUXVfogDLffuSbUWG0CVI6y304tc.jpg?auto=webp&s=4f991cfb09ecf283b6b213cdfaf47c2da0af01cc", "width": 1536, "height": 1080, "crop": "smart", "auto": "webp", "s": "78f41b1470af51dfce435cd13167d1b8457861a7", "width": 108, "height": 182}, {"url": "https://external-preview.redd.it/ZzIooIOrpEomCzKwUXVfogDLffuSbUWG0CVI6y304tc.jpg?width=320&crop=smart&auto=webp&s=ba398preview.redd.it/ZzIooIOrpEomCzKwUXVfogDLffuSbUWG0CVI6y304tc.jpg?width=640&crop=smart&auto=webp&s=769d5337dcf4cc5d271b5efc0066c9b53ba80ea2", "width": 640, "height": 365}, {"url": "https://external-preview.redd.it/1080, "height": 1829}], "variants": {}, "id": "mfIzcx6JGfTEkC00WgF0h_zkpeVUM5VP5s9QuLvGi0c"}, {"enabled": true}, {"all_awardings": [], "awarders": [], "media_only": null, "subreddit_id": "t5_2qh1o", "mod_reason_by": null, "removal_reason": null, "link_flair_background_color": "", "id": "dckm3d", "is_robot_indexable": true, "contest_mode": false, "mod_reports": [], "author_patreon_flair": false, "author_flair_text_color": null, "permalink": "/r/aww/comments/dckm3d/he_was_nervous_about_subreddit_subscribers": 22052896, "created_utc": 1570069266.0, "num_crossposts": 0, "media": null, "is_video": false}, {"kind": "t3", "data": {"approved_at_utc": false, "title": "This adorable cat trying to do a cartwheel.", "link_flair_richtext": [], "subreddit_name_prefixed": "r/aww", "hidden": false, "pwl": 6, "link_flair_text_color": "dark", "author_flair_background_color": null, "subreddit_type": "public", "ups": 16902, "total_awards_received": 0, "media_embed": {}, "thumbnail_width": 140, "is_meta": false, "category": null, "secure_media_embed": {}, "link_flair_text": null, "can_mod_post": false, "score": 16902, "approved_by": null, "thumbnail": "steward_reports": [], "author_flair_richtext": [], "gildings": {}, "post_hint": "link", "content_categories": null, "is_self": false, "mod_note": null, "crosspost_mod_reason_title": null, "gilded": 0, "clicked": false, "title": "To copy Hooman", "link_flair_richtext": [], "subreddit_name_prefixed": "r/therewasanattempt", "quarantine": false, "link_flair_text_color": "dark", "author_flair_background_color": null, "subreddit_type": "public", "ups": 32039, "total_awards_received": 0, "is_reddit_media_domain": false, "is_meta": false, "category": null, "secure_media_embed": {}, "link_flair_text": null, "can_mod_post": false, "score": 32039, "author_flair_css_class": null, "steward_reports": [], "author_flair_richtext": [], "gildings": {}, "post_hint": "link", "content_categories": null, "is_self": false, "i.imgur.com", "allow_live_comments": true, "selftext_html": null, "likes": null, "suggested_sort": null, "banned_at_utc": null, "view_count": null, "archived": "https://external-preview.redd.it/DPAKq0p3LEuPyhrPE0ncCPGNI0y0f939xdCZwSdKfXw.jpg?auto=webp&s=5b31525f8e7246f0c30c0f67cf3f9660a652dbc8", "width": 480, "height": 270}, {"url": "https://external-preview.redd.it/DPAKq0p3LEuPyhrPE0ncCPGNI0y0f939xdCZwSdKfXw.jpg?width=320&crop=smart&auto=webp&s=42bc/reddit_video_preview": {"fallback_url": "https://v.redd.it/tzorpq2617q31/DASH_480", "height": 480, "width": 384, "scrubber_media_url": "https://v.redd.it/tzorpq2617q31/HLSPlaylist.m3u8", "is_gif": true, "transcoding_status": "completed"}, {"enabled": false}, {"all_awardings": [], "awarders": [], "distinguished": null, "subreddit_id": "t5_39ne7", "mod_reason_by": null, "removal_reason": null, "link_flair_background_color": "", "id": "dcmcl", "is_robot_indexable": true}
```

# Web pretty-printers

- <https://jsonformatter.org/json-pretty-print>

```
1 {
2   "kind": "Listing",
3   "data": {
4     "modhash": "1er6wkluqe34886b4be68e3d40284ea2027c8b9553d12a5db",
5     "dist": 100,
6     "children": [
7       {
8         "kind": "t3",
9         "data": {
10          "approved_at_utc": null,
11          "subreddit": "aww",
12          "selftext": "",
13          "author_fullname": "t2_kn5k6ai",
14          "saved": false,
15          "mod_reason_title": null,
16          "gilded": 0,
17          "clicked": false,
18          "title": "Fluffy Bunny",
19          "link_flair_richtext": [],
20          "subreddit_name_prefixed": "r/aww",
21          "hidden": false,
22          "pwls": 6,
23          "link_flair_css_class": null,
24          "downs": 0,
25          "thumbnail_height": 140,
26          "hide_score": false,
27          "name": "t3_dcfzdz",
28          "quarantine": false,
29          "link_flair_text_color": "dark",
30          "author_flair_background_color": null,
31          "subreddit_type": "public",
32          "ups": 7542,
33          "total_awards_received": 0,
34          "media_embed": {},
35          "thumbnail_width": 140,
36          "author_flair_template_id": null,
37          "is_original_content": false,
38          "user_reports": [],
39          "secure_media": {
40            "reddit_video": {
41              "fallback_url": "https://v.redd.it/n8zsn3umt6q31/DASH_720?source=f",
42              "height": 720,
43              "width": 404,
44              "scrubber_media_url": "https://v.redd.it/n8zsn3umt6q31/DASH_96",
45              "dash_url": "https://v.redd.it/n8zsn3umt6q31/DASHPlaylist.mpd",
46              "duration": 25,
47              "hls_url": "https://v.redd.it/n8zsn3umt6q31/HLSPlaylist.m3u8",
48              "is_gif": true,
49              "transcoding_status": "completed"
50            }
51          },
52          "is_reddit_media_domain": true,
53          "is_meta": false,
54          "category": null,
55          "secure_media_embed": {},
56          "link_flair_text": null,
57          "can_mod_post": false,
58          "score": 7542,
59          "approved_by": null,
60          "thumbnail": "https://a.thumbs.redditmedia.com/xP73kMwvCnFF4s3Mc6f113
```

# Network services for mobile

- User authentication
  - Anonymous users
  - Email/Password
  - Google/facebook/phone number,...
- Storage
  - Files (e.g., images, video)
  - Structured data (e.g., documents, collections)
  - Access control
- Cloud functions (not in this class)
  - Code executes on your behalf in the cloud. No VMs

# Authentication

- How do you prove you are who you say you are?
  - Know a secret (e.g., password)
  - Possess a token (e.g., phone number)
  - Vouched for by third party (e.g., google or facebook)
- Firebase authentication
  - Manages multiple users without a server
  - Configure a web-based console
  - Open source activity that does best practices login

# Google firestore

- Store data in a flexible format in the cloud
  - No server, web console
- Collections and documents
  - Documents are key/value pairs and sub-collections
  - Alternate collection/document/collection/document...
- Real-time updates!
  - Cool for features like chat
  - Works well with live data

# Working with a database

- Know what parts of your model are written by client code and what by the db/server
  - E.g., timestamps, ids are often db/server only
- Expose fetched data via your viewModel
  - LiveData super useful if update is asynchronous
- Understand how items are cached/modified
- Generalize your notion of pointer
  - A pathname to an image is a “pointer.”
    - If pathname is in database, file better exist (referential integrity)
  - A foreign key points from one table to row of another

# Taking pictures

- Your app can control the camera
  - Lots of control (filters, etc.)
  - Complex code
- You can launch an intent to take a picture
  - Much like an implicit intent
  - Input: URI location to store picture
    - File name must end in .jpg
  - Output: success Boolean
- Problem: Output callback does not know location!
  - If you have other metadata, like a tag, that is lost



# Taking pictures, problems

- `private val tag: String = ""`
- `private val location: String = ""`
- `private val cameraLauncher = registerForActivityResult(  
 ActivityResultContracts.TakePicture()) { success ->  
 // use tag and location freely`
- `// Here we use the above declarations`
- `tag = "funny"`  
`location = "foobar.jpg"`  
`cameraLauncher.launch(uri)`
- **What is the problem here?**
  - Our code probably in fragment (view) that can be killed
  - Especially because we are launching a camera activity
  - If killed, we lose tag and location!

# Taking pictures, solutions

- ```
private val cameraLauncher = registerForActivityResult(  
    ActivityResultContracts.TakePicture()) { success ->  
    // use viewModel.tag and viewModel.location freely
```
- ```
// Here we use the above declarations
```
- ```
viewModel.tag = "funny"  
viewModel.location = "foobar.jpg"  
cameraLauncher.launch()
```
- **Store state in the view model**
  - Safe even if camera activity kills our fragment
  - View model holds strings—simple data type

# Taking pictures, details

- Just remember the last component of the file name
  - timestamp in Notebook
  - Universally unique identifier (UUID) for cloud database
  - We call it fileName and it is a string
- Useful functions
  - fileNameToFile(fileName: String): File
  - There is a way to create a URI from a File object
- Application manages permissions via URI
  - Each app gets its own area to write for security
- Example (fileName, absolute path, URI)
  - 35751550-1edd-4738-afd1-fbf41cc918a7
  - /storage/emulated/0/Pictures/35751550-1edd-4738-afd1-fbf41cc918a7.jpg
  - content://edu.utap.firechat/my\_images/Pictures/35751550-1edd-4738-afd1-fbf41cc918a7.jpg

# Notes: dealing with dependences

- Consider a simple note.
  - `text: String`
  - `imageFiles: List<String>`
- `imageFiles` is a list of path names
  - Each path name refers to an image
  - Image might be a local file (sqlite)
  - Image might be a cloud file (firebase)
- Note depends on images
  - Danger: a note that refers to a non-existent image

# How to create/edit a note (sqlite)

- In-memory note has list of image files (newList)
- In-DB note has list of image files (oldList)
- Algorithm to modify image table
  - Leave members shared between oldList & newList
  - In oldList and not in newList -> delete file
  - In newList and not in oldList -> add row

# How to create a note (firestore)

- For list of image files -> start upload
- Write note
- What is wrong with this picture?
  - `note.pictureUUIDs = [ffjsjsjsj-sjfjfj]`
  - `storage/images/ffjsjsjsj-sjfjfj` does not exist
  - Violates referential integrity!
  - Error when displaying new note (via live data)

# How to create a note (firestore)

- Take a picture
- Start file upload
- When file successfully uploaded...
  - Delete local file
  - Update note to have new pictureUUID
  - Write note to server
    - Maintains referential integrity!
    - Write data before writing pointers to it
  - Might be user-visible delay for their picture to show up