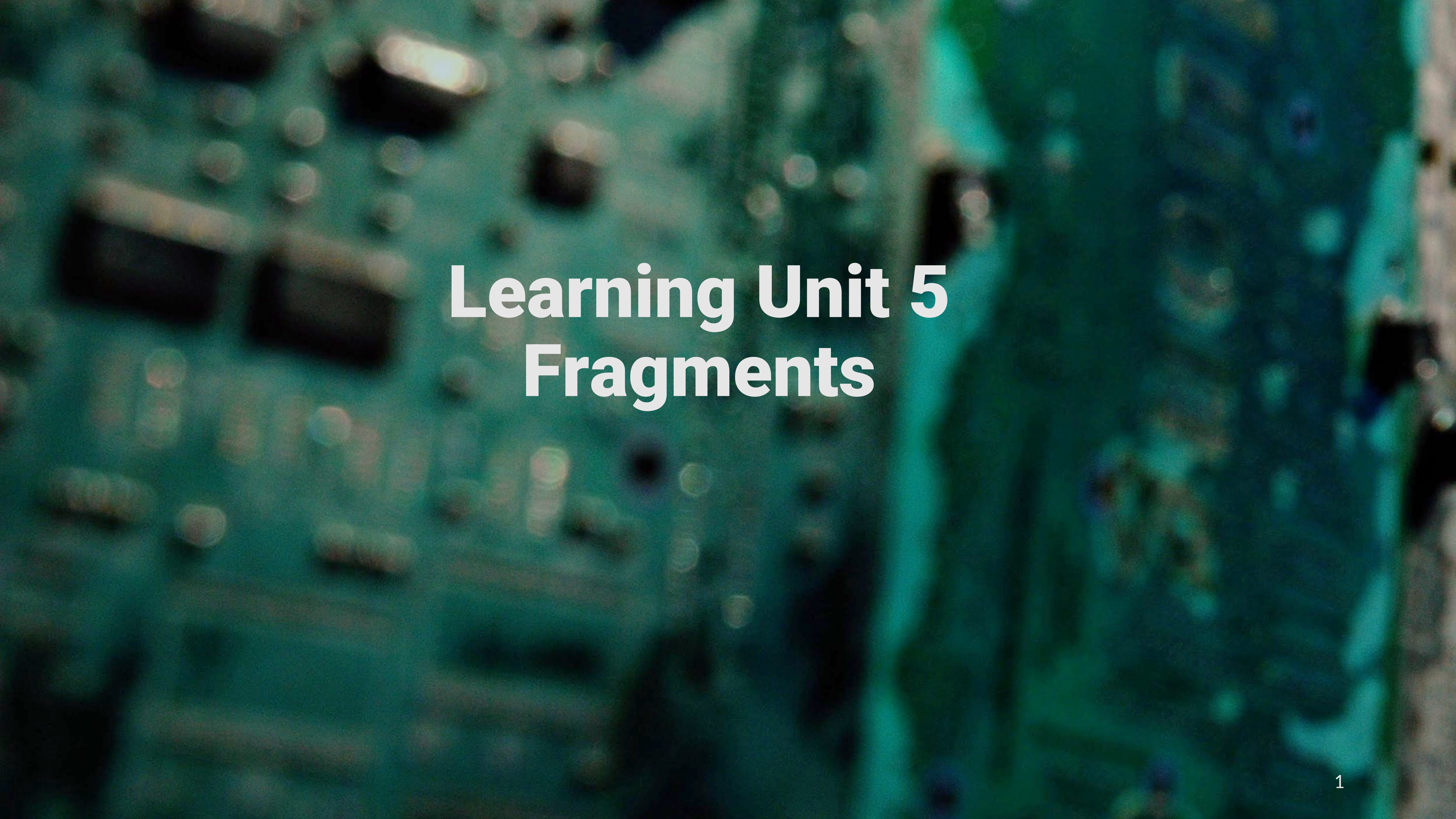
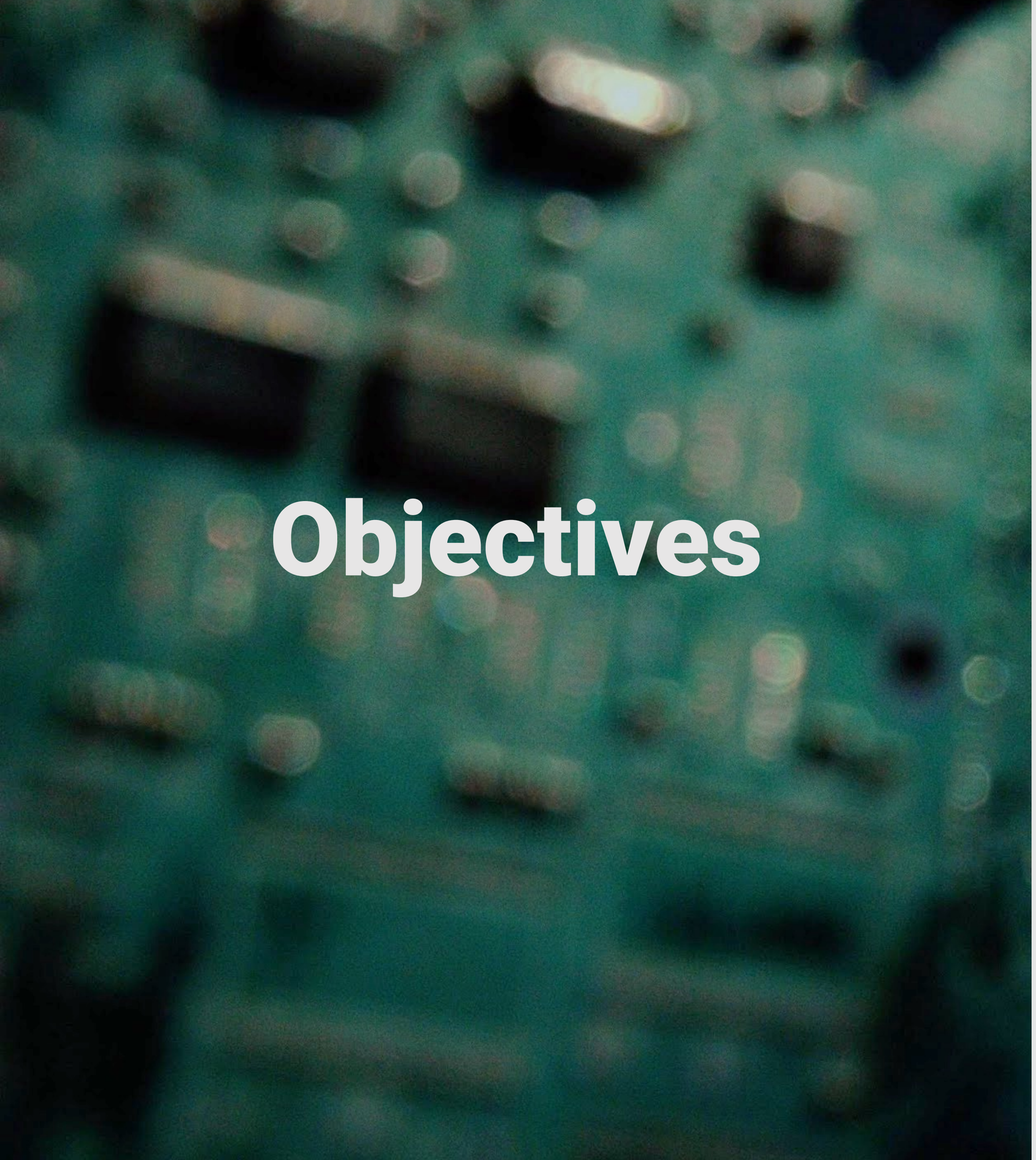
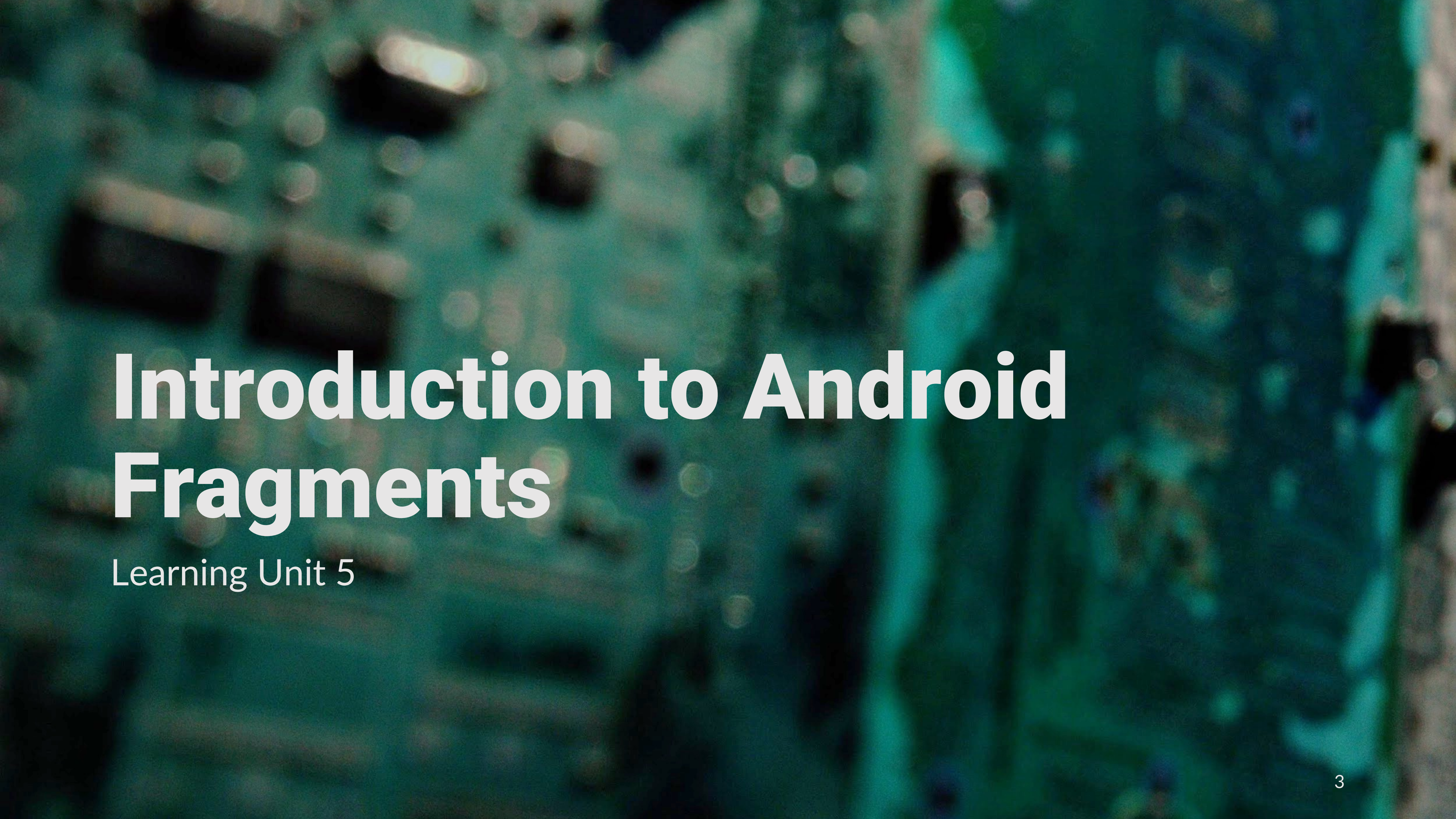


Learning Unit 5
Fragments



Objectives

1. Understand and use of fragments.
2. Understand fragments' lifecycle
3. Creating a UI fragment and Defining Fragment's layout
4. Adding a UI fragment to the FragmentManager
5. Use Fragment transactions
6. Introduction to storing data



Introduction to Android Fragments

Learning Unit 5

Introduction to Fragments

- A Fragment represents a behavior or a portion of user interface in a FragmentActivity.

- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.

- You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

- A fragment must always be hosted in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.

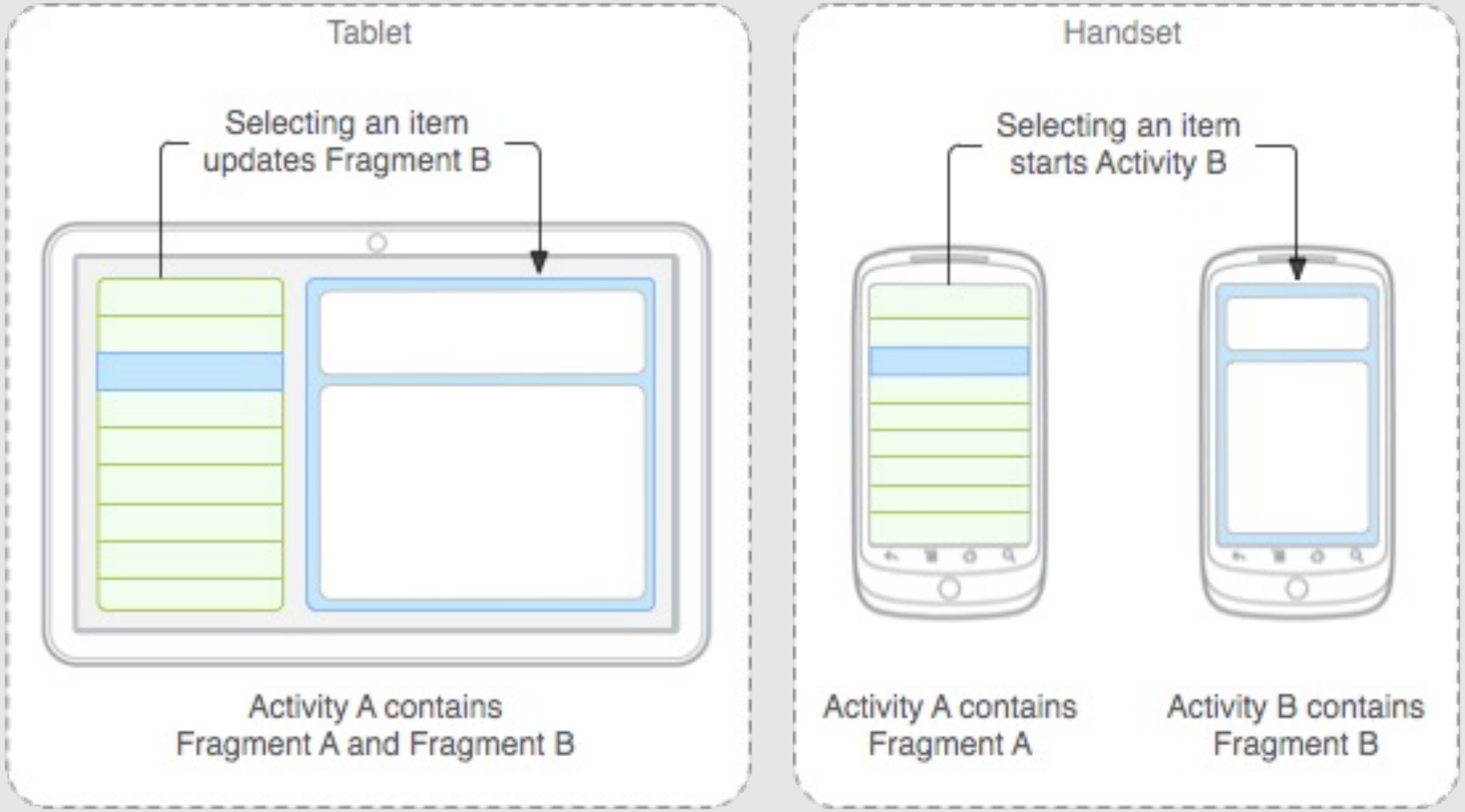
Introduction to Fragments cont.

- Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets.

- Because a tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components.

- Fragments allow such designs without the need for you to manage complex changes to the view hierarchy.

Fragments: Why they are useful?

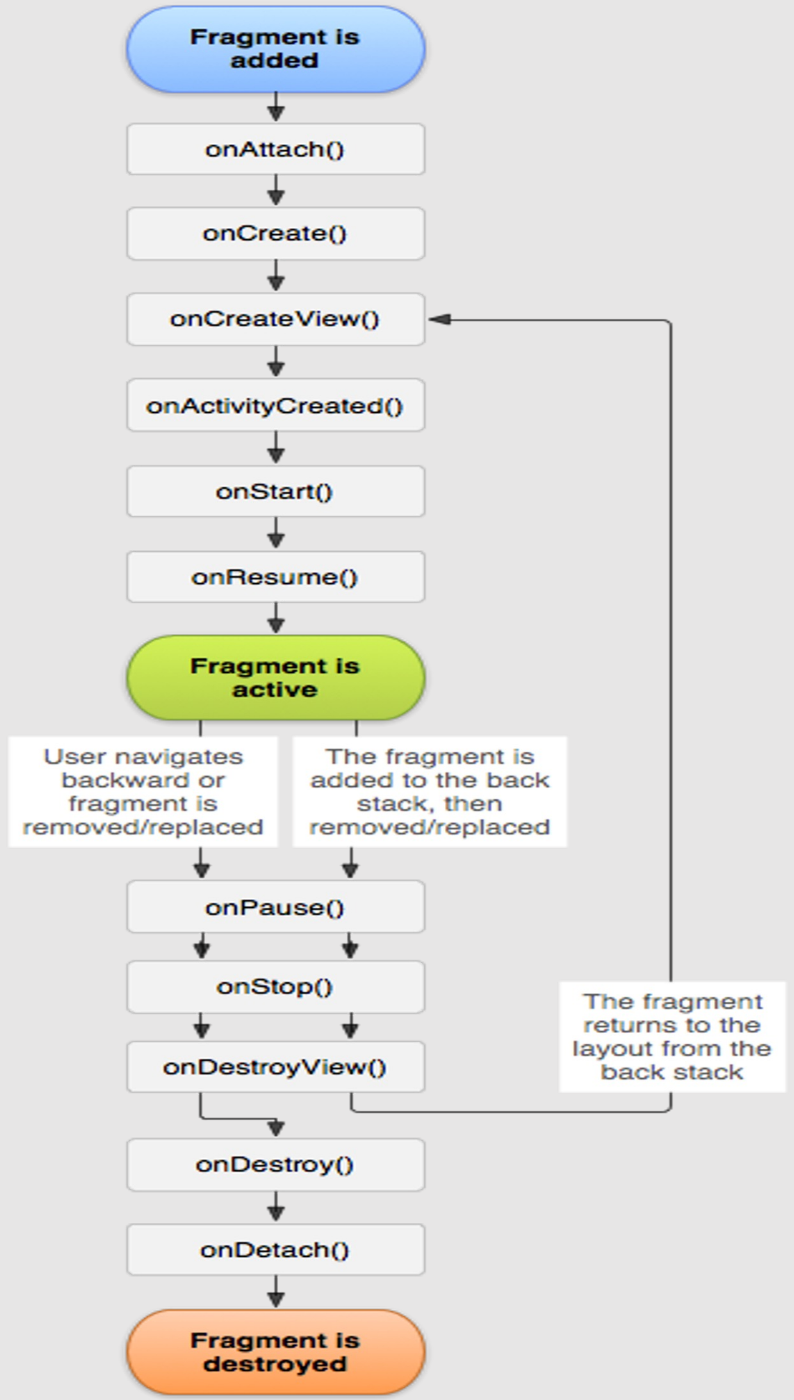

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Fragments: why they are useful? cont.

- A fragment is used to render and manage a single UI view..
- Fragments are parts of the application's UI or behavior that are integrated into activities -- a fragment needs an activity to run.
- Consider each fragment as a **lightweight mini-activity**.
- Fragments are **flexible and reusable components** defined by the programmer.
- Fragments present a **consistent UI across different apps** and devices; fragments are flexible in adapting the user experience across these different environments (phone, tablet); therefore, allowing to group UI views.
- Each fragment has **its own lifecycle** and associated UI (for example, a fragment for the user to enter profile information, another one to render maps).
- How many fragments do you need? Rule of thumb is one per different type of data displayed (e.g., preferences, profile, maps, list of exercises) or user input.

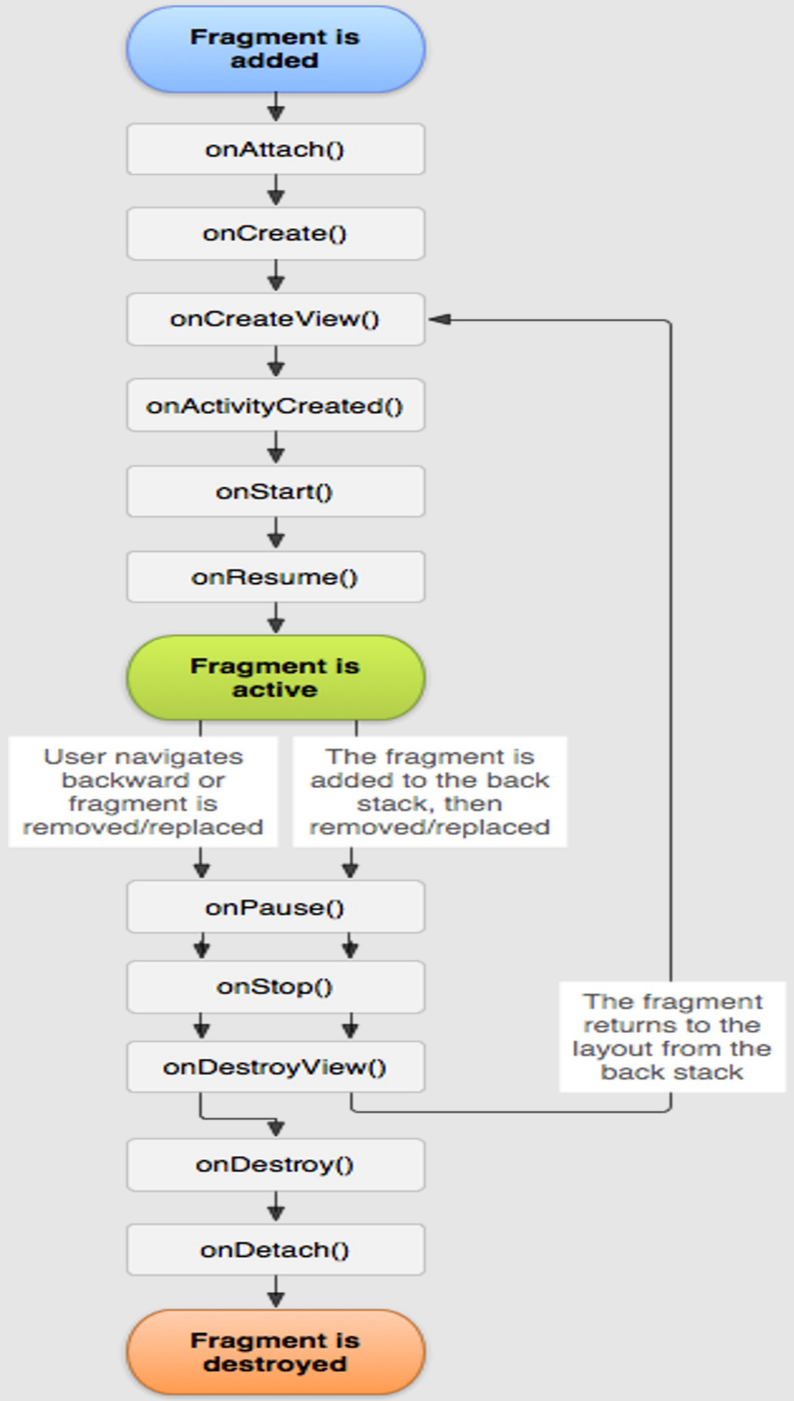
**Treat a fragment as a mini-activity with its own
own independent UI and Lifecycle (to manage its internal state).**

Fragments should be considered as part of the controller layer in the MVC architecture



Fragment Lifecycle

- onAttach() is called when the Fragment is attached to its parent activity
- **onCreate()** is called to initially create the fragment
- onCreateView() is called to created its user interface -- it inflates the fragment UI
- onActivityCreated() is called once the parent activity and the fragment UI are created.
- **onStart()** is called the start of the visible lifetime, any UI changes can be applied before the fragment goes visible
- **onResume()** is called at the start of the active lifetime such as resuming any paused UI updates needed by the fragment that were suspended when it became inactive.



Fragment Lifecycle

- **onPause()** called at the end of the active lifetime; persist all edits or state changes, suspend UI updates, threads, or CPU intensive processes that don't need to be updated when the Activity isn't the active foreground activity
- **onSaveInstanceState()** called to save UI state changes at the end of the active lifecycle
- void onStop() called at the end of the visible lifetime
- onDestroyView() called when the fragment's view is detached
- **onDestroy()** called at the end of the full lifetime
- onDetach() called when the Fragment has

Types of Fragments

There are two kinds of fragments based on the the way being added to an activity layout:

- **Static:** by declaring the fragment inside the activity's layout file.

- **Dynamic:** or sometimes refered to as flexible UI
  - At any time while your activity is running, you can add fragments to your activity layout. You simply need to specify a ViewGroup in which to place the fragment.

Static Fragments

- A static Fragment is included in an XML file using a fragment tag within another XML layout.

- When you use a Fragment statically it is very like a custom UI component that behaves like the other components - Buttons, TextFields and so on.

Static Fragments cont.

- Example:
- Assuming that you want to create an ArticleFragment such as:

-
  ```
  class Fragment01 : Fragment(R.layout.fragment_01) {

  }
  ```

- Then the static Fragment link to the activity layout is through the XML file as in:

-
  ```
  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">


      <fragment android:name="com.example.android.fragments. Fragment01"
              android:id="@+id/article_fragment"
              android:layout_weight="2"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

  </LinearLayout>
  ```

Dynmaic Fragments (Flexible UI)

- Add a Fragment to an Activity at Runtime
- You need to use the FragmentManager class.
- The FragmentManager class provides methods that allow you to add, remove, and replace fragments to an activity at runtime in order to create a dynamic experience.
- To perform a transaction such as add or remove a fragment, you must use the FragmentManager to create aFragmentTransaction, which provides APIs to add, remove, replace, and perform other fragment transactions.
- The activity's layout includes an empty FrameLayout that acts as the fragment container.

Adding a fragment to an activity dynamically

- To make fragment transactions in your activity (such as add, remove, or replace a fragment), you must use APIs from FragmentTransaction. You can get an instance of FragmentTransaction from your Activity like this:

```
supportFragmentManager.beginTransaction().apply{
    replace<ExampleFragment>(R.id.fragment_container)
    setReorderingAllowed(true)
    addToBackStack("name") // name can be null
}
```

Introduction to Storing Data

Learning Unit 5

Saving Data

Most Android apps need to save data, even if only to save information about the app state during onPause() so the user's progress is not lost.

Most non-trivial apps also need to save user settings, and some apps must manage large amounts of information in files and databases.

This unit introduces you to the principal data storage options in Android, including:

1. Saving key-value pairs of simple data types in a **shared preferences file.**
2. Saving arbitrary files in Android's file system.
3. Using databases managed by SQLite

Saving Key-Value Sets 1/4

For relatively small amount of data of K-V then:

- Use the SharedPreferences APIs.
- A SharedPreferences object points to a file containing key-value pairs and provides simple methods to read and write them.
- Each SharedPreferences file is managed by the framework and can be private or shared.

Saving Key-Value Sets 2/4

Get a Handle to a SharedPreferences

You can create a new shared preference file or access an existing one by calling one of two methods:

- **getSharedPreferences()** — Use this if you need multiple shared preference files identified by name, which you specify with the first parameter. You can call this from any Context in your app.

Saving Key-Value Sets 3/4

```
val sharedPreference = getSharedPreferences("PREF_NAME",Context.MODE_PRIVATE)
var editor = sharedPreference.edit()
editor.putString(”NAME",”Moaath")
editor.commit()
```

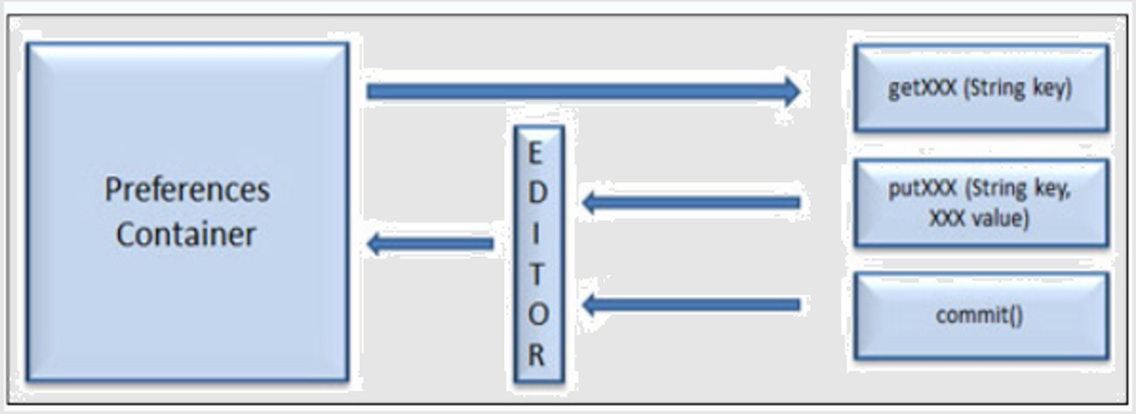
When naming your shared preference files, you should use a name that's uniquely identifiable to your app, such as

"com.example.myapp.PREFERENCE_FILE_KEY"

Saving Key-Value Sets 4/4

Write to Shared Preferences

```
val sharedPreference = getSharedPreferences("PREF_NAME",Context.MODE_PRIVATE)
var editor = sharedPreference.edit()
editor.putString(”NAME",”Moaath")
editor.commit()
```



Read from Shared Preferences

```
sharedPreference.getString(" NAME ","defaultName")
sharedPreference.getLong("l",1L)
```

What have we learnt this unit?

- Fragments static and Dynamic.
- Fragments’ lifecycle
- Using fragments in applications (two tutorials)
- Storing data using shared preferences