Activity Lifecycle

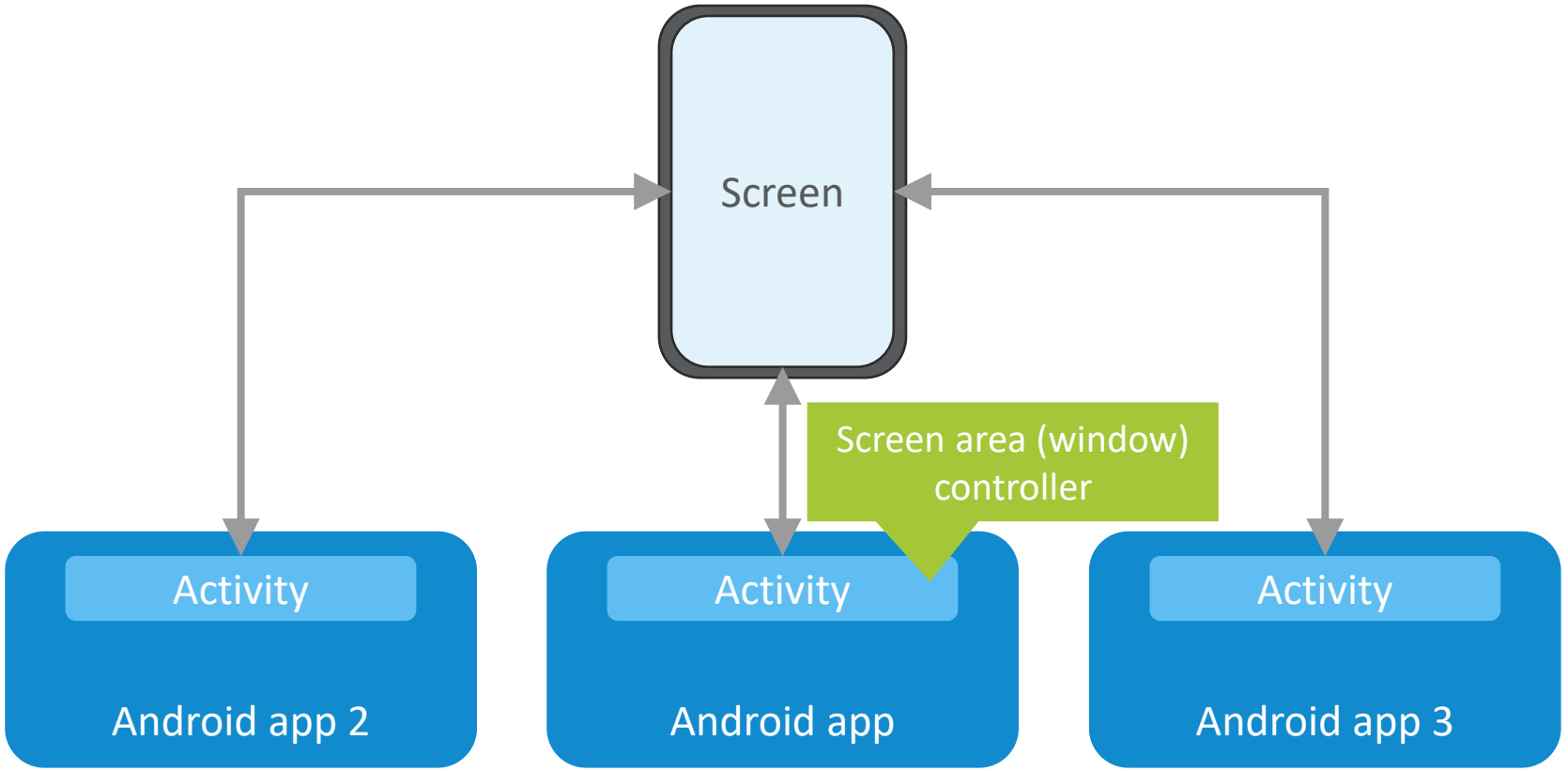
Activity?

From the official documentation:

- One screen in the application.
- A single, focused thing that the user can do.
- The entry point for an app's interaction with the user.

“What is Activity” isn’t a simple question!

Activity in Android applications



Screen is a shared resource!

Multiple Activities in a single app is just a special (simpler) case

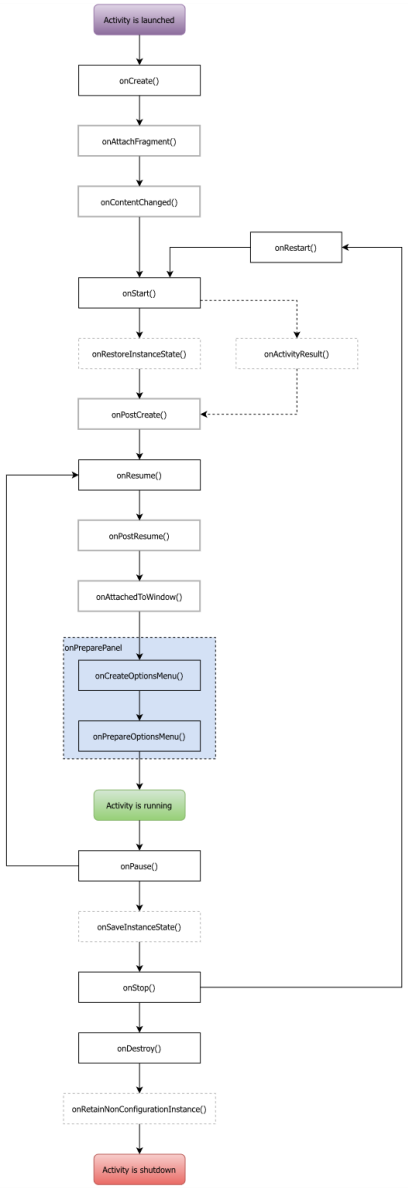
Implications of the screen being a shared resource

- OS should be able to transfer control over windows from one Activity to another.
- Activities should be notified when the state of their control changes.
- Activities should be able to exchange data between them.
- More…

Activity lifecycle supports these requirements!

Activity lifecycle

https://github.com/xxv/android-lifecycle



Activity as Context

Context

Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

God Object!

Activity as Context

- Activity extends Context, so it’s a God Object as well.
- Goes against Favor Composition over Inheritance rule from Effective Java book.

Context and lifecycles are orthogonal concepts!

Activity
onCreate() and onDestroy()

Activity onCreate()

- Background: Activity objects are instantiated by the system.
- The first method invoked after a new instance of Activity is created is onCreate().
- onCreate() replaces constructor.

Activity onDestroy()

- This method is invoked to let the Activity know that it isn't needed anymore.
- Despite its name, even after this method returns, the actual destruction of Activity object isn't guaranteed (will be discussed later in the course).
- A better name for this method would be onRelease().

Dos and don’ts in onCreate()

- Do: call through to super.onCreate().
- Do: set content View.
- Do: initialize fields and properties.
- Do: change attributes that affect the eventual user interface (e.g. set Window flags).

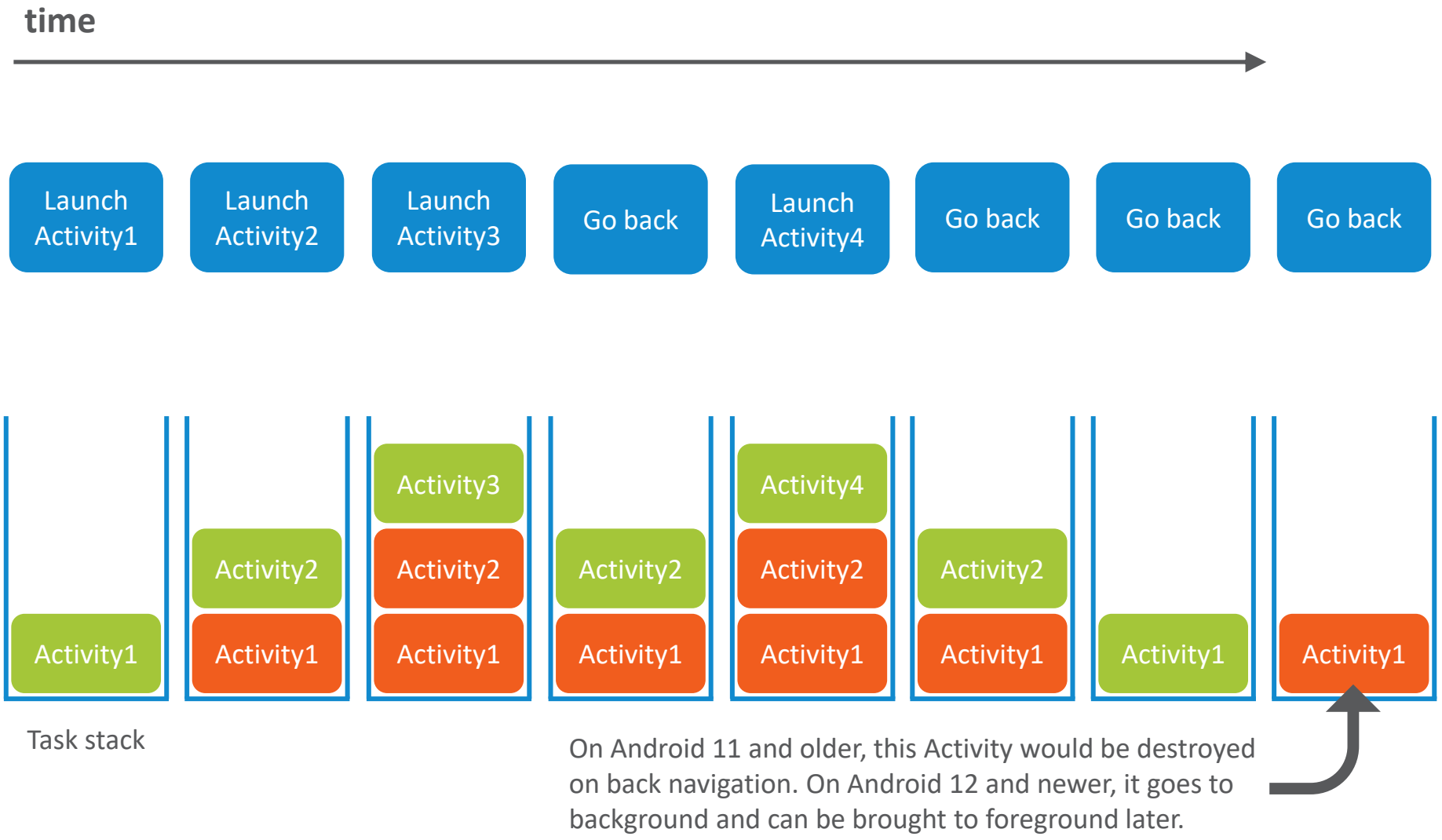
- Don’t: start animations.
- Don’t: start functional flows.
- Don’t: allocate resources.

Dos and don’ts in onDestroy()

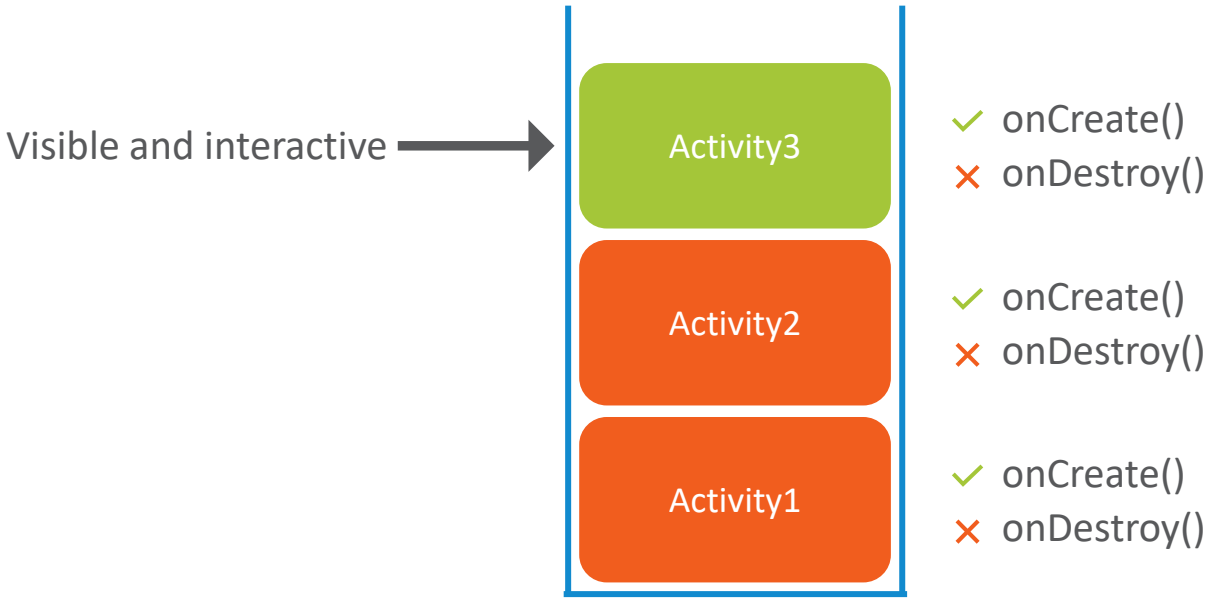
- Do: unregister from any Observable that you registered to in onCreate() (discussed later in the course).
- Don’t: you don’t need this method in absolute majority of your Activities.

Activities Back Stack

Activities back stack operation



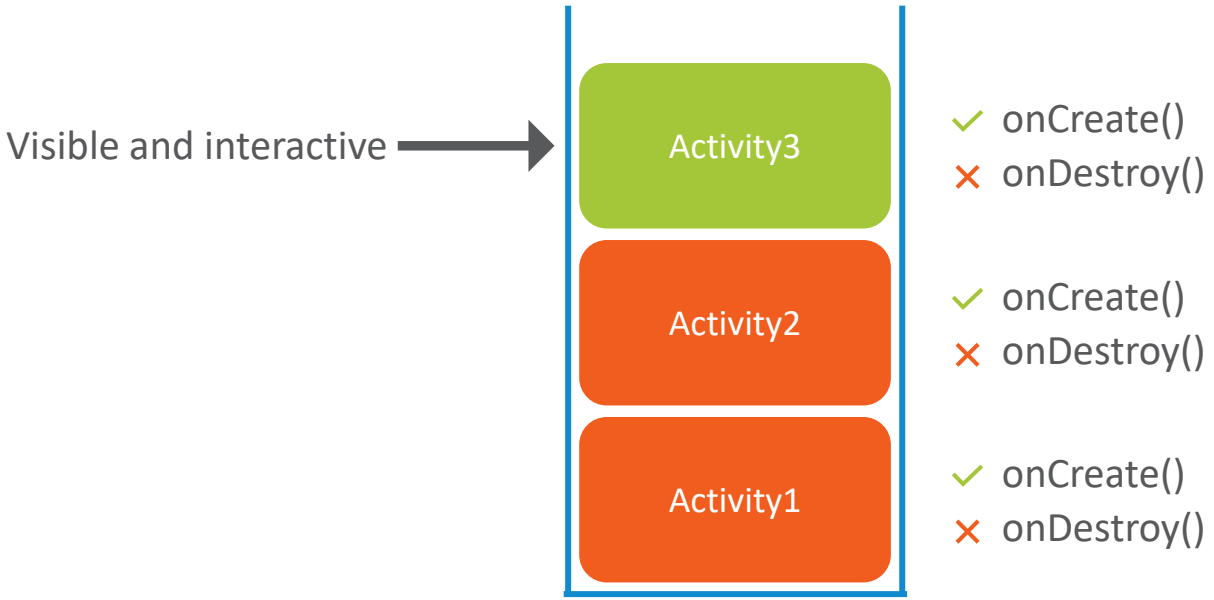
Foreground Activity vs back-stacked Activity



How is the foreground Activity different from the back-stacked Activities?

Activity
onStart() and onStop()

Foreground Activity vs back-stacked Activity



How is the foreground Activity different from the back-stacked Activities?

Activity onStart()

- Called when Activity gets control of any visible area of the screen.

Activity onStop()

- Called when Activity no longer has control of any visible part of the screen.

Dos and don’ts in onStart()

- Do: call through to super.onStart().
- Do: register as observer to receive events that affect the user interface.
- Do: update the user interface to reflect the most up to date state.
- Do: start functional flows (e.g. fetch data from the database).

- Don’t: initialize fields and properties*.
- Don’t: start animations*.
- Don’t: access mutex shared resources.

Dos and don’ts in onStop()

- Do: call through to super.onStop().
- Do: unregister from observables.
- Do: pause or cancel functional flows.

Activity
onResume() and onPause()

Activity onResume()

- Called when Activity becomes interactive to the user.

Activity onPause()

- Called when Activity becomes non-interactive to the user.

Dos and don’ts in onResume()

- Do: call through to super.onResume().
- Do: start animations.
- Do: access shared mutex resources.
- Do: use in special cases (e.g. lock task mode)

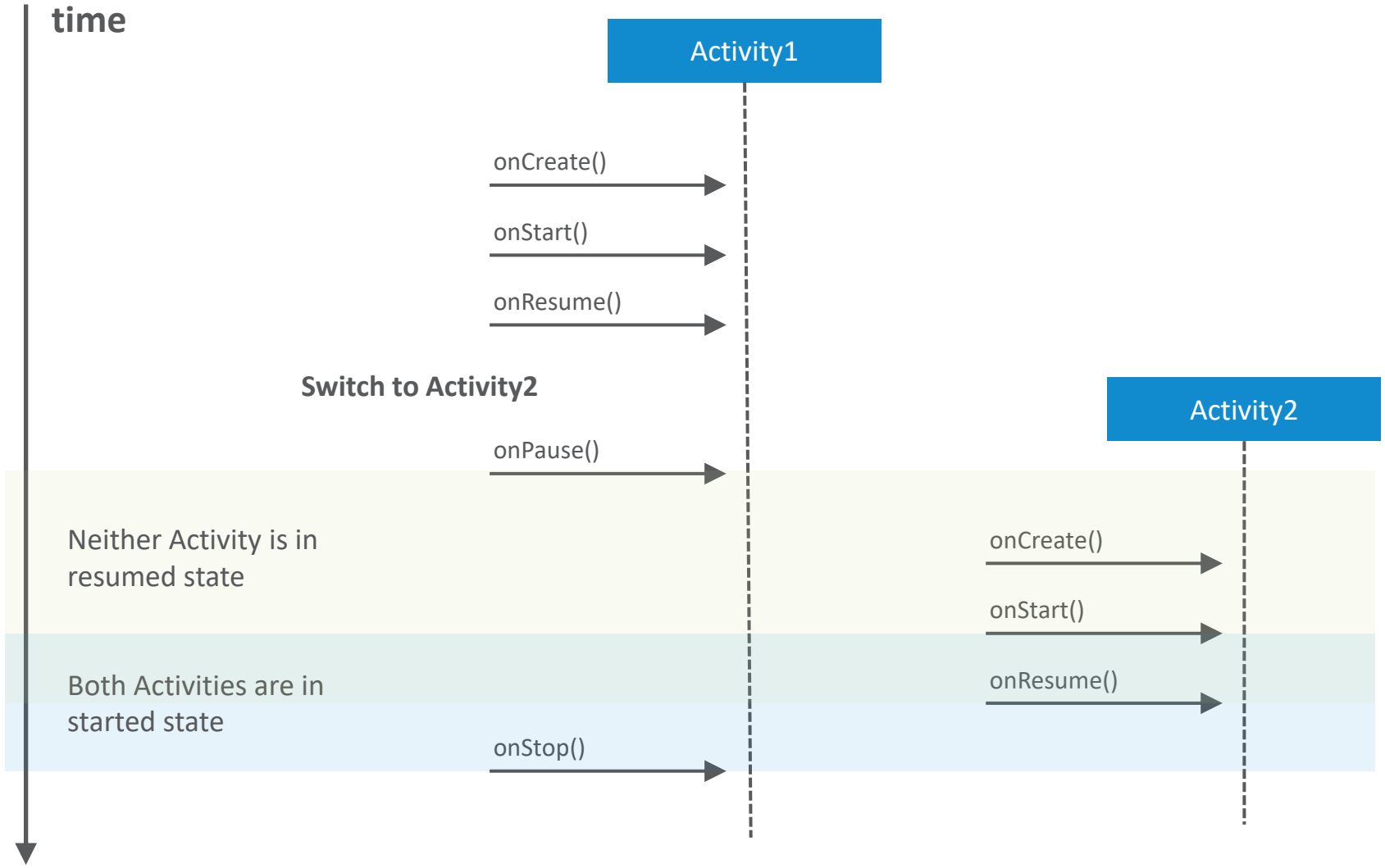
- Don’t: you don’t need this method in absolute majority of your Activities.

Dos and don’ts in onPause()

- Do: call through to super.onPause().
- Do: stop animations.
- Do: release shared mutex resources.

- Don’t: you don’t need this method in absolute majority of your Activities.

Overlapping vs non-overlapping lifecycle phases


time
Activity1
onCreate()
onStart()
onResume()
Switch to Activity2
Activity2
onPause()
Neither Activity is in resumed state
onCreate()
onStart()
Both Activities are in started state
onResume()
onStop()

Multi-resume

Multi-resume

- Prior to Android 10 (API level 29), only one Activity could be resumed at any given instant.
- Starting with Android 10, there can be multiple resumed Activities simultaneously.

Accessing mutex resources prior to Android 10

- Mutex resources must be accessed only during non-overlapping lifecycle phase.
- Request access to mutex resources in onResume().
- Release mutex resources in onPause().

Accessing mutex resources in Android 10 or later

- Mutex resources must be accessed only during non-overlapping lifecycle phase.
- Request access to mutex resources in **onTopResumedActivityChanged()** when isTopResumedActivity is true, **and in onResume() for API level < 29.**
- Release mutex resources in **onTopResumedActivityChanged()** when isTopResumedActivity is false, **and in onPause() for API level < 29**.

Conditions when onResume()/ onPause() won’t work

- Android 10 or later.
- Multiple resumed Activities in the system (e.g split-screen mode).
- Two or more resumed Activities request access to the same mutex resource, while at least one of them doesn’t account for isTopResumedActivity state.

Rare scenario

Memory leaks

Lifecycle

Specification of object's creation and destruction times, as well as a set of special callback methods which the system invokes automatically in response to specific events.

Memory leak

Non-intentional code (a bug) that keeps a reference to an object after the object is no longer needed, thus preventing that object from being garbage collected, and leading to excessive memory consumption.

Activity Lifecycle

Summary

Activity lifecycle methods

- **onCreate():** called after instantiation, replaces constructor
- **onStart():** called when the Activity becomes visible
- **onResume():** called when the Activity becomes interactive (topmost Activity on the stack of the active task)
- **onTopResumedActivityChanged():** called when the “top” resumed Activity changes in API levels with multi-resume support
- **onPause():** called when the Activity becomes non-interactive
- **onStop():** called when the Activity is no longer visible
- **onDestroy():** called when the Activity is no longer

Activity lifecycle methods responsibilities

- **onCreate():** initialize fields and properties, set content view, window customization
- **onStart():** register observers, update UI with the latest data, start functional flows
- **onResume():** start animations, access mutex resources (API level < 29), special cases and hacks
- **onTopResumedActivityChanged():** access mutex resources (API level >= 29)
- **onPause():** stop animations, release mutex resources (API level < 29)
- **onStop():** unregister observers, cancel or pause functional flows (if required)
- **onDestroy():** rarely ever needed

Memory leaks

References to Activity instances from global objects, static state and other objects with different lifecycles, can prevent garbage collector from destroying those instances, even after their lifecycle completes.

More info about Activity lifecycle in the next modules…