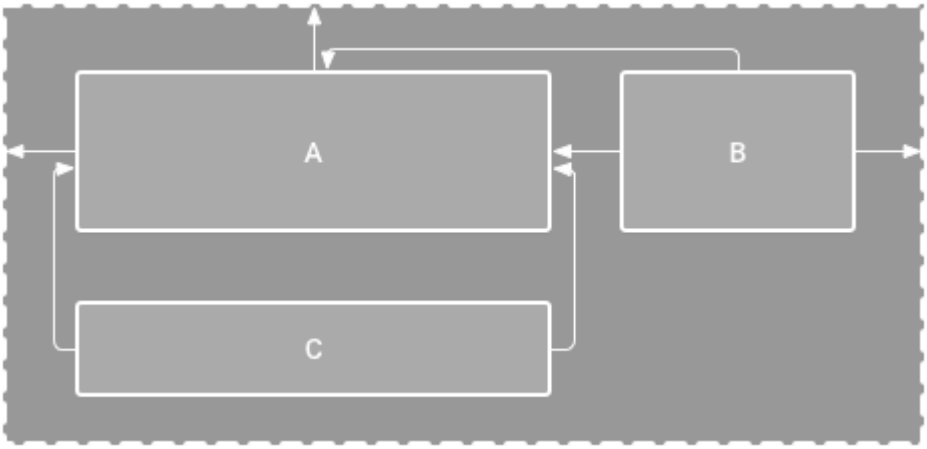
G → Services and intents

Layouts

Constraint Layout:

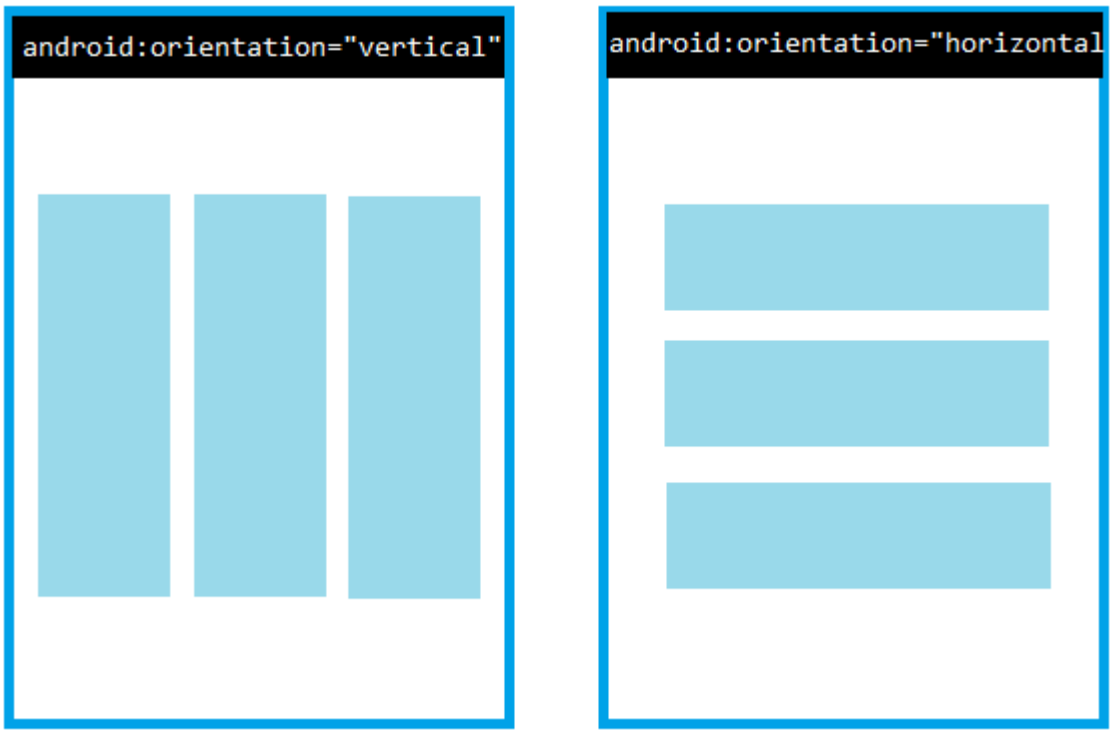
A flexible layout manager that allows you to create complex layouts with a flat view hierarchy. It’s highly customizable and efficient, and it’s often used for complex UI designs.



Constraint Layout

Linear Layout:

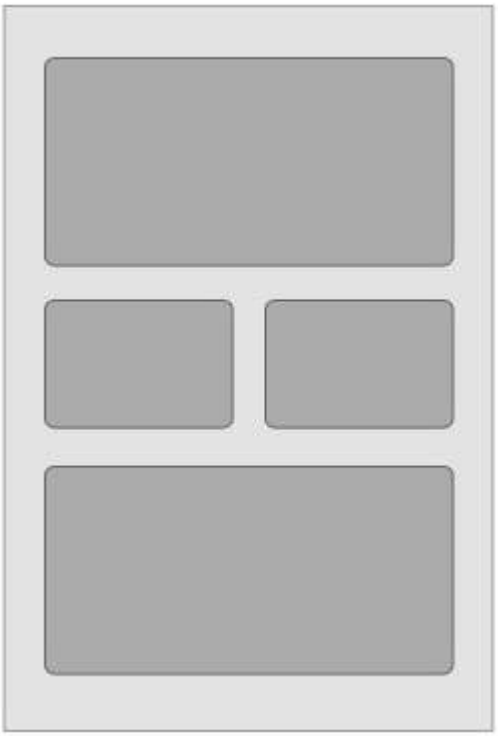
A simple layout manager that arranges views in a single row or column. It’s easy to use and great for simple UI designs.



Linear Layout(Vertical and Horizontal)

Relative Layout:

A layout manager that arranges views relative to each other or relative to the parent layout. It's highly customizable and great for complex UI designs.



Relative Layout

Intents

- There are three fundamental use cases:
- **Starting an activity**

  An `Activity` represents a single screen in an app. You can start a new instance of an `Activity` by passing an `Intent` to `startActivity()`. The `Intent` describes the activity to start and carries any necessary data.

  If you want to receive a result from the activity when it finishes, call `startActivityForResult()`. Your activity receives the result as a separate `Intent` object in your activity's `onActivityResult()` callback. For more information, see the Activities guide.

- **Starting a service**

  A `Service` is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a

service with `JobScheduler`. For more information about `JobScheduler`, see its `API-reference documentation`.

For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the `Service` class. You can start a service to perform a one-time operation (such as downloading a file) by passing an `Intent` to `startService()`. The `Intent` describes the service to start and carries any necessary data.

If the service is designed with a client-server interface, you can bind to the service from another component by passing an `Intent` to `bindService()`. For more information, see the Services guide.

- **Delivering a broadcast**

  A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an `Intent` to `sendBroadcast()` or `sendOrderedBroadcast()`.

Services

A *service* is an entry point for keeping an app running in the background for all kinds of reasons.

It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface.
For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it to interact with it.

There are two types of services that tell the system how to manage an app: started services and bound services.

Started services tell the system to keep them running until their work is completed. This might be to sync some data in the background or play music even

after the user leaves the app. Syncing data in the background or playing music represent different types of started services, which the system handles differently:
• Music playback is something the user is directly aware of, and the app communicates this to the system by indicating that it wants to be in the foreground, with a notification to tell the user that it is running. In this case, the system prioritizes keeping that service's process running, because the user has a bad experience if it goes away.
• A regular background service is not something the user is directly aware of, so the system has more freedom in managing its process. It might let it be killed, restarting the service sometime later, if it needs RAM for things that are of more immediate concern to the user.

Bound services run because some other app (or the system) has said that it wants to make use of the service. A bound service provides an API to another process, and the system knows there is a dependency between these processes. So if process A is bound to a service in process B, the system knows that it needs to keep process B and its service running for A. Further, if process A is something the user cares about, then it knows to treat process B as something the user also cares about.
Because of their flexibility, services are useful building blocks for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they run.