SEMINARIO DE LENGUAJES OPCIÓN ANDROID

Ciclo de vida de una Activity

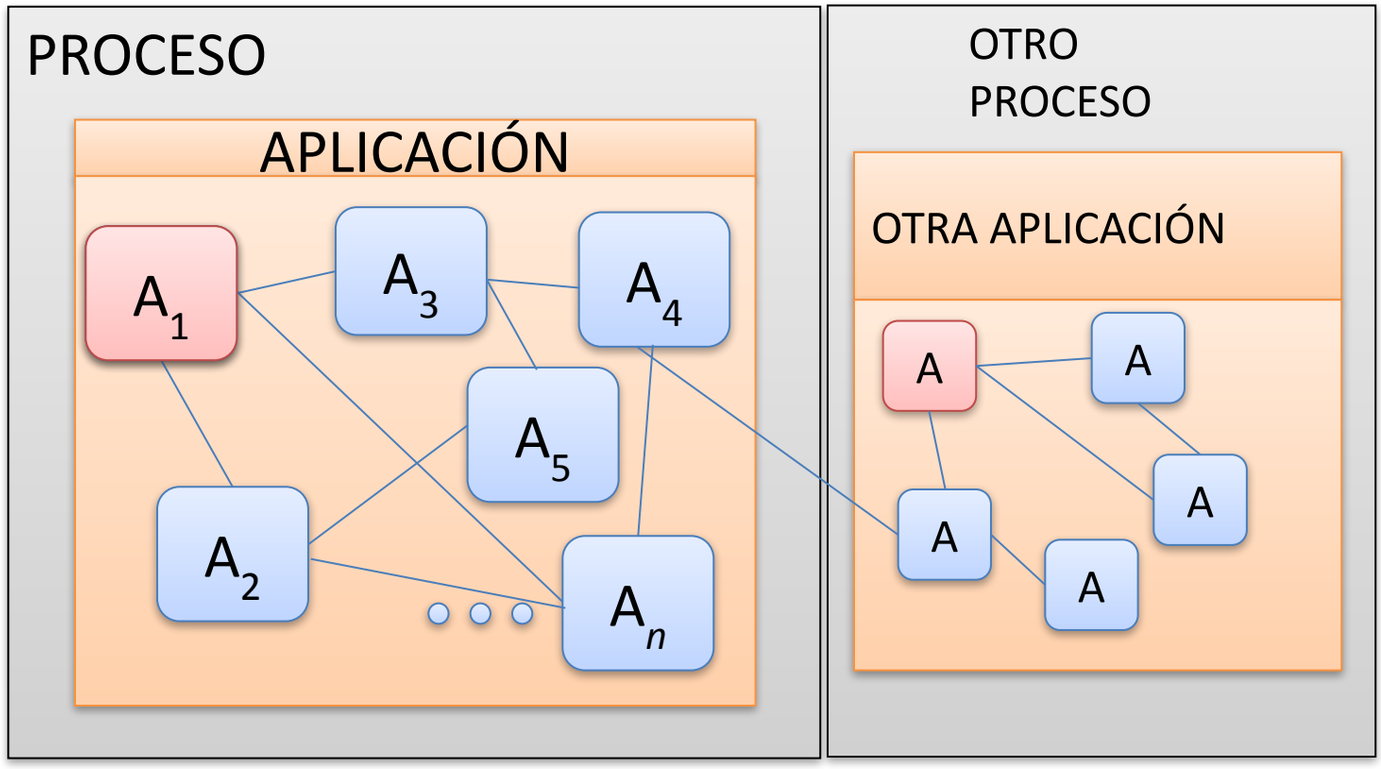
Esp. Fernández Sosa Juan Francisco

Ciclo de vida de una *activity*

Una aplicación generalmente consiste en múltiples *activities* vinculadas entre sí, corriendo en un único proceso del S.O.

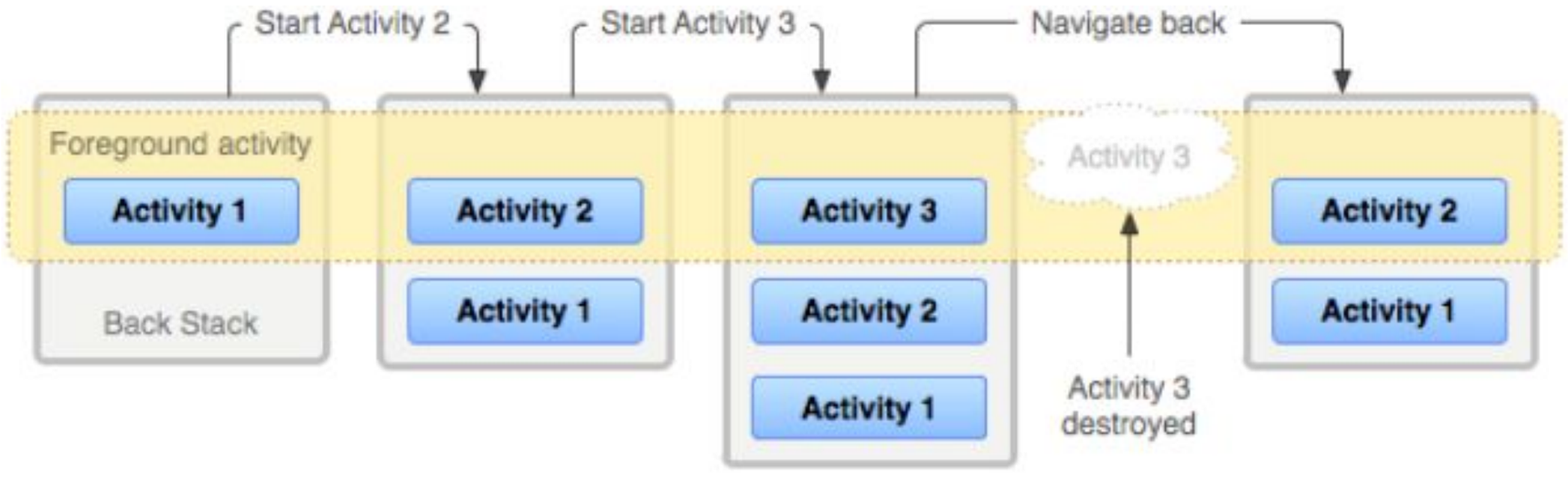
Normalmente, hay una *activity* principal que se presenta al usuario cuando éste inicia la aplicación por primera vez.

Nota: Una *activity* de un proceso puede invocar a otra *activity* en un proceso distinto



Cada vez que se inicia una *activity* nueva, se detiene la anterior, se la incluye en la pila de *activities* y obtiene el foco (atención del usuario).

Cuando el usuario presiona el botón Atrás, se quita de la pila, se destruye y se reanuda la *activity* anterior.

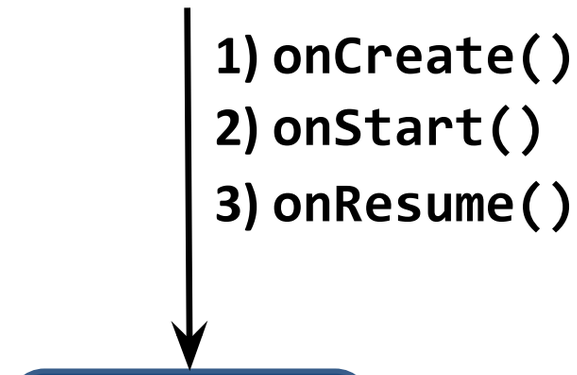


Ciclo de vida de una *activity*

- Una actividad puede pasar por distintos estados: created, paused, stopped, resumed o destroyed.
- Por cada transición de estado se ejecutarán una serie de métodos callbacks.
-

Por ejemplo, una *activity* que hace uso de una base de datos

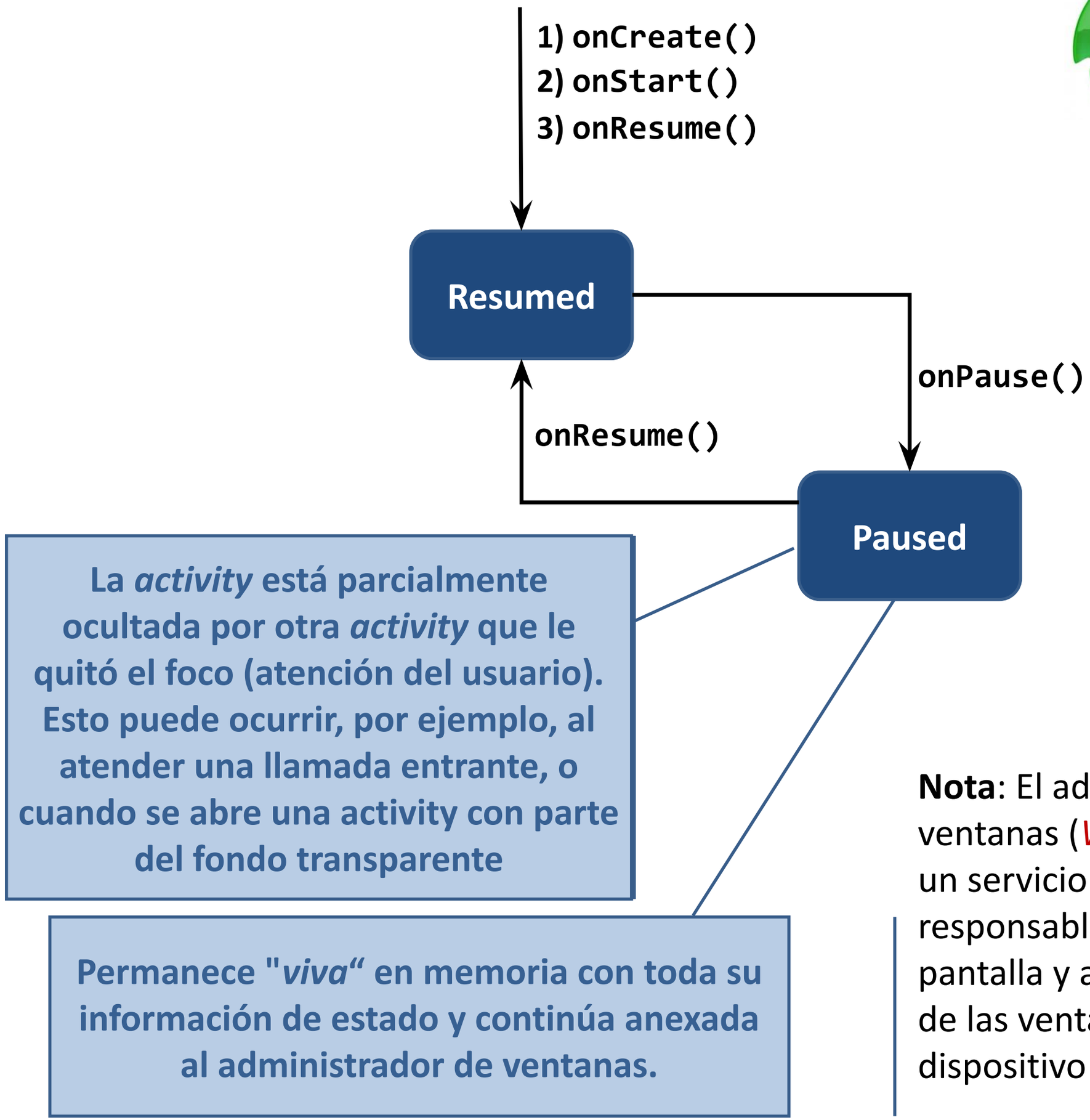
Ciclo de vida



La *activity* se encuentra en el primer plano y tiene el foco (la atención del usuario)

También se suele denominar *running*, *reanudada* o *en ejecución*

Ciclo de vida



Nota: El administrador de ventanas (*Window Manager*) es un servicio del sistema responsable de organizar la pantalla y administra el orden z de las ventanas visualizadas en el dispositivo

Ciclo de vida

1) onCreate()
2) onStart()
3) onResume()

Resumed

onPause()

3) onResume()
2) onStart()
1) onRestart()

onResume()

Stopped

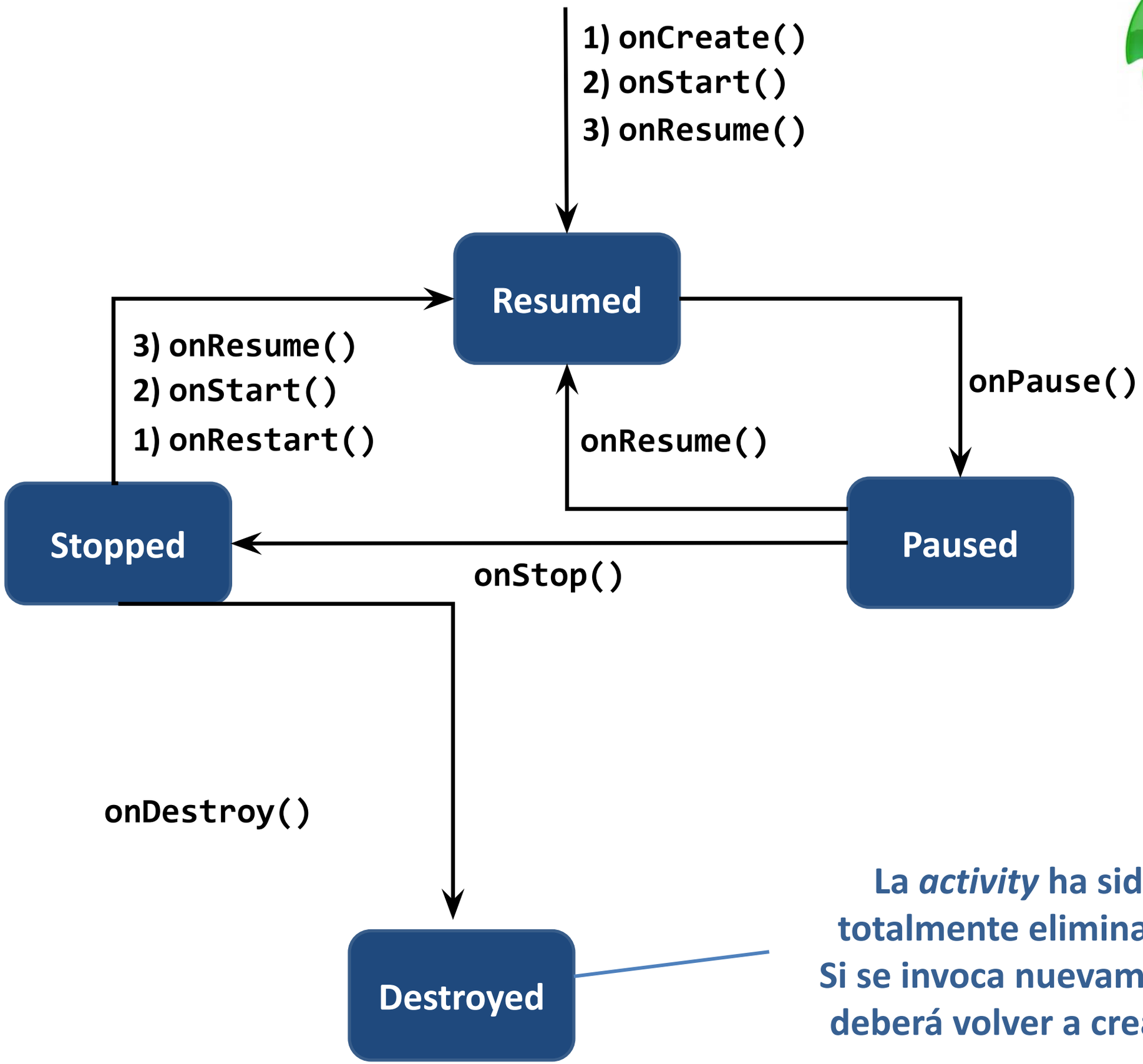
Paused

onStop()

La *activity* ya no está visible para el usuario, está completamente ocultada por otra *activity*.

Permanece "*viva*" en memoria con toda su información de estado, pero no está anexado al administrador de ventanas.

Ciclo de vida

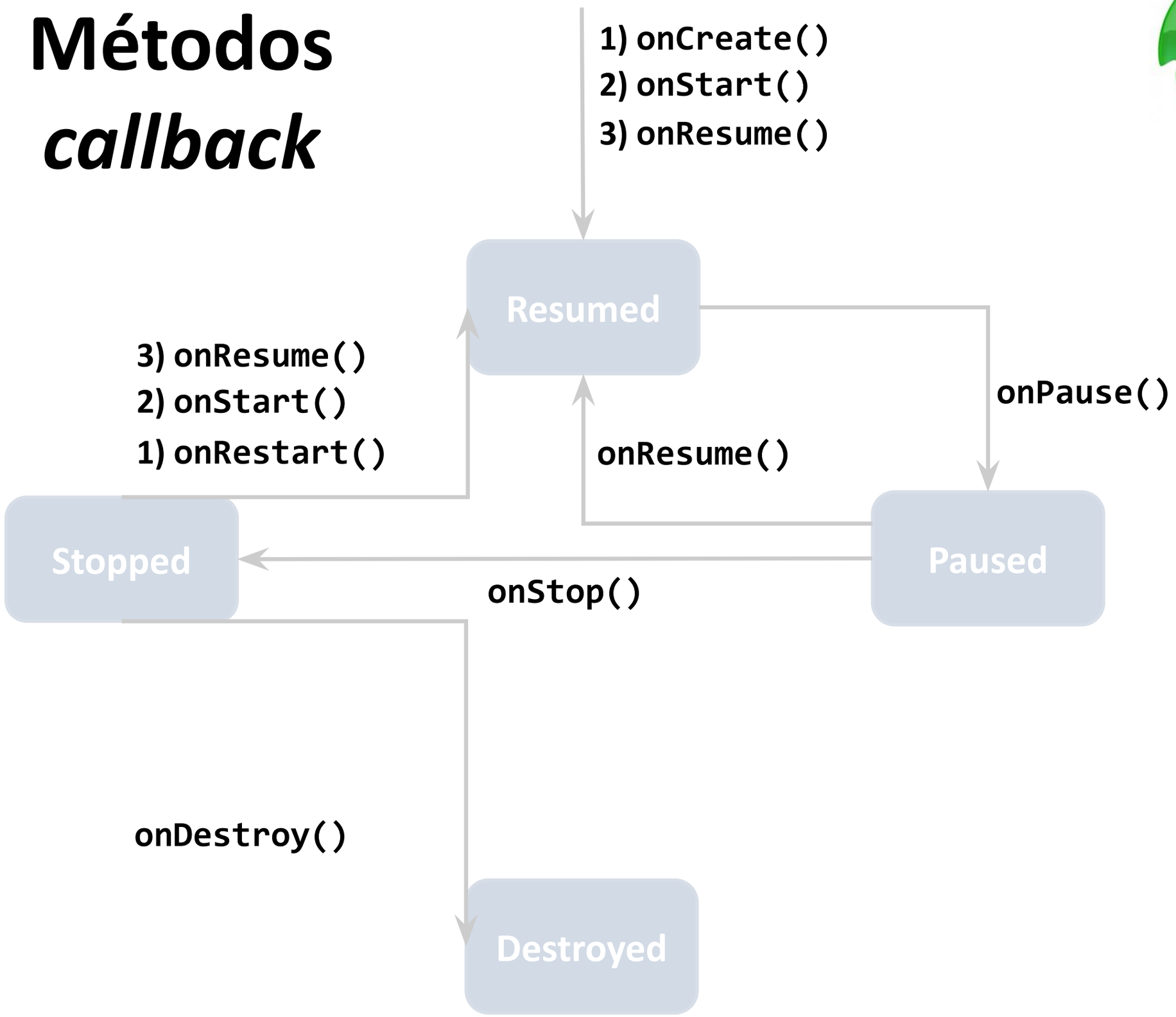


La *activity* ha sido totalmente eliminada. Si se invoca nuevamente deberá volver a crearse

¿Cómo aprovechar los métodos *callback* ?

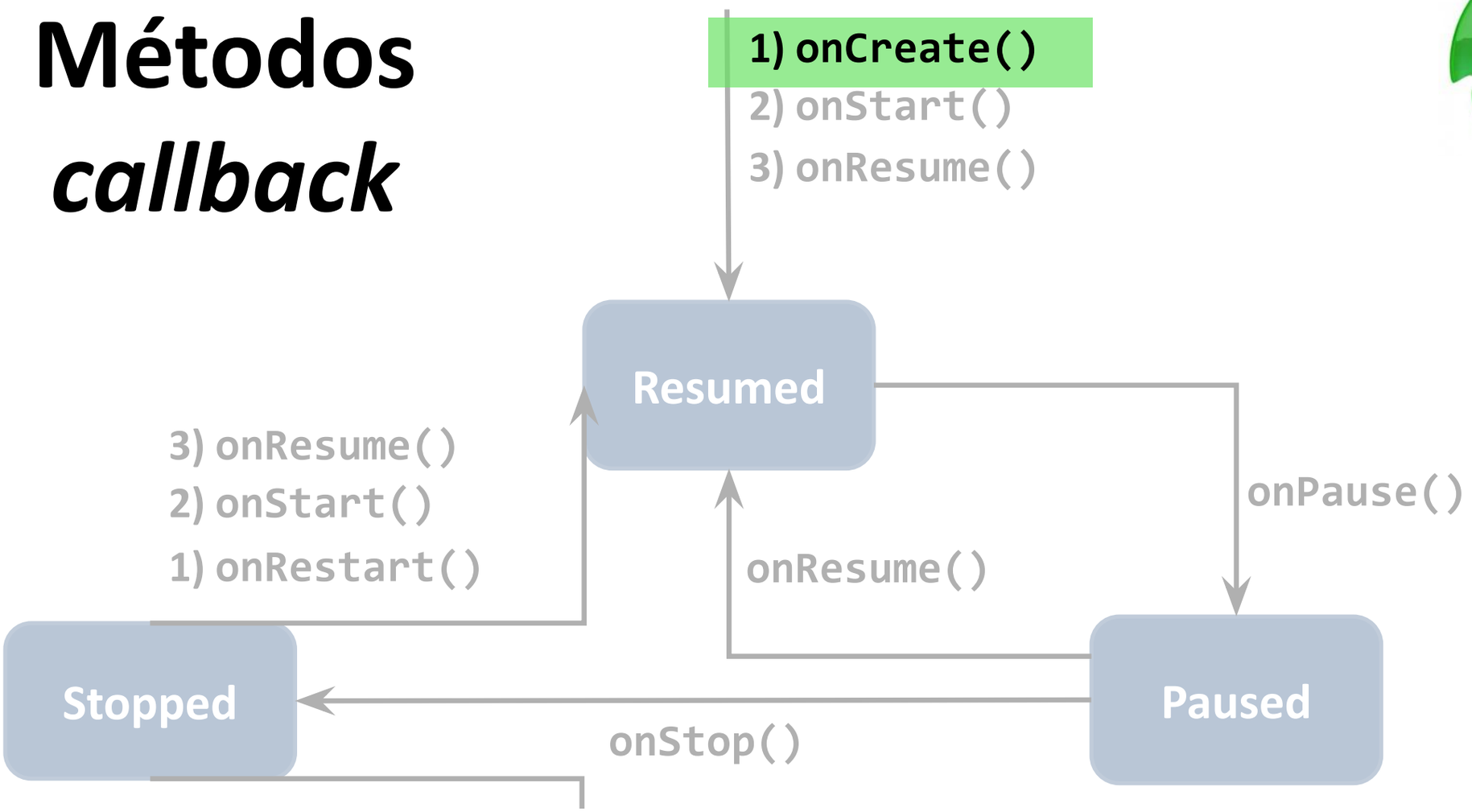
Métodos *callback*

Ciclo de vida



Métodos *callback*

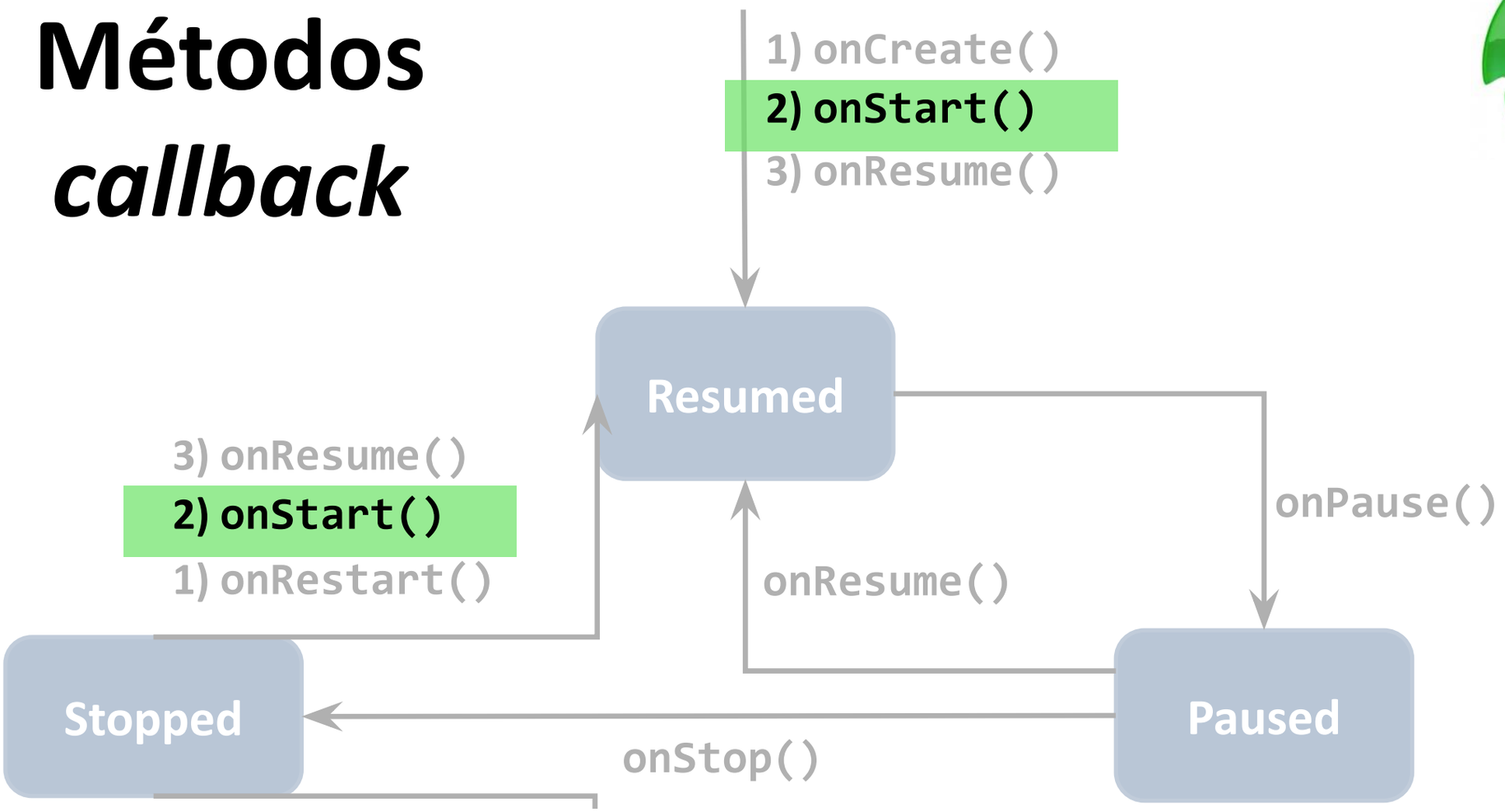
Ciclo de vida



- **onCreate**() se ejecuta justo después del constructor. Aquí se inicializa la *activity* y se define la vista de la misma.
- Este método recibe un parámetro nulo si es la primera vez que se crea la *activity* o con datos para recuperar el estado anterior en caso contrario.

Métodos *callback*

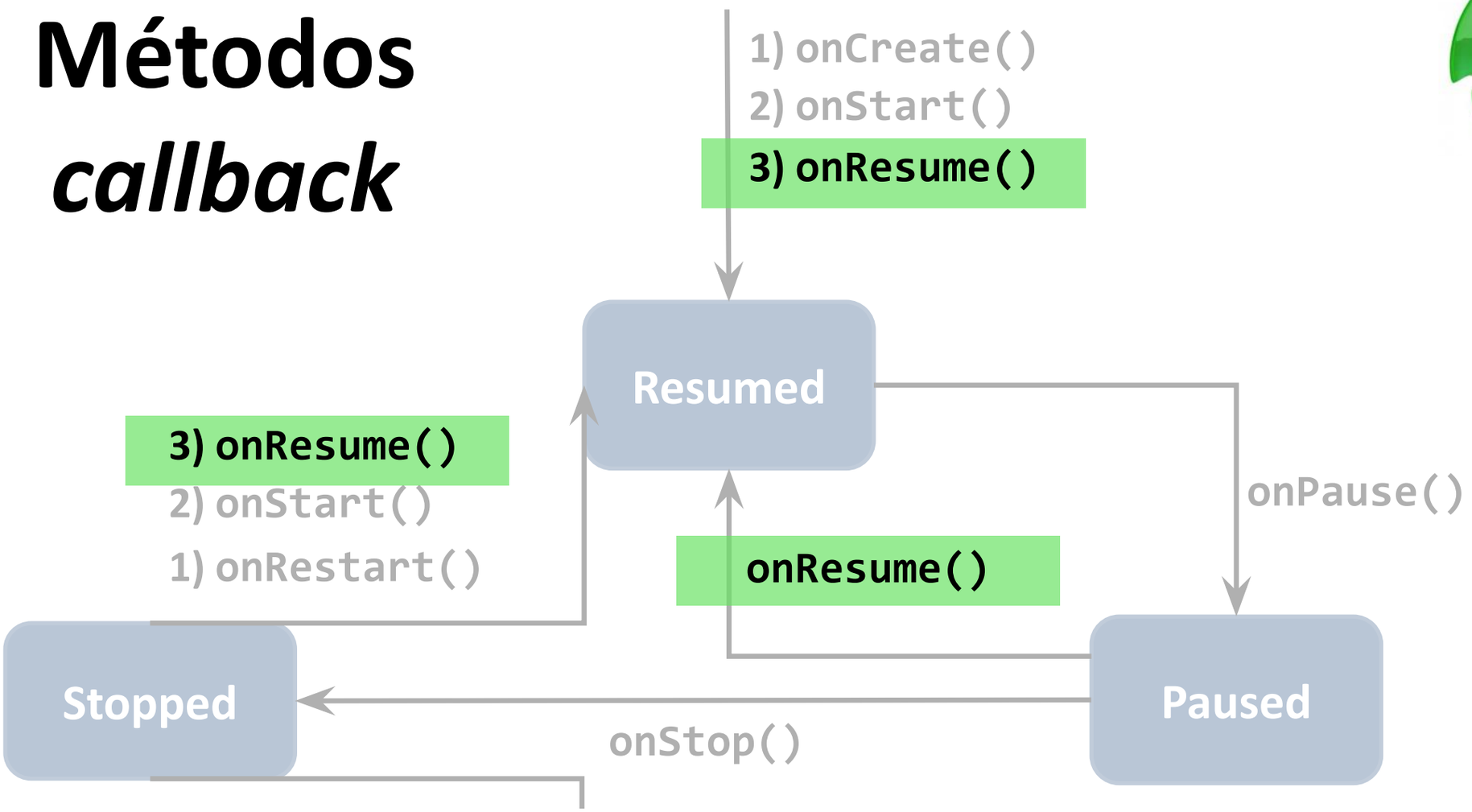
Ciclo de vida



- **onStart**() hace que el usuario pueda ver la actividad, mientras la app se prepara para que esta entre en primer plano y se convierta en interactiva. Por ejemplo, este método es donde la app inicializa el código que mantiene la IU.
- Puede utilizarse para crear procesos cuyo objetivo es actualizar la interfaz de usuario: animaciones, temporizadores, localización GPS etc.

Métodos *callback*

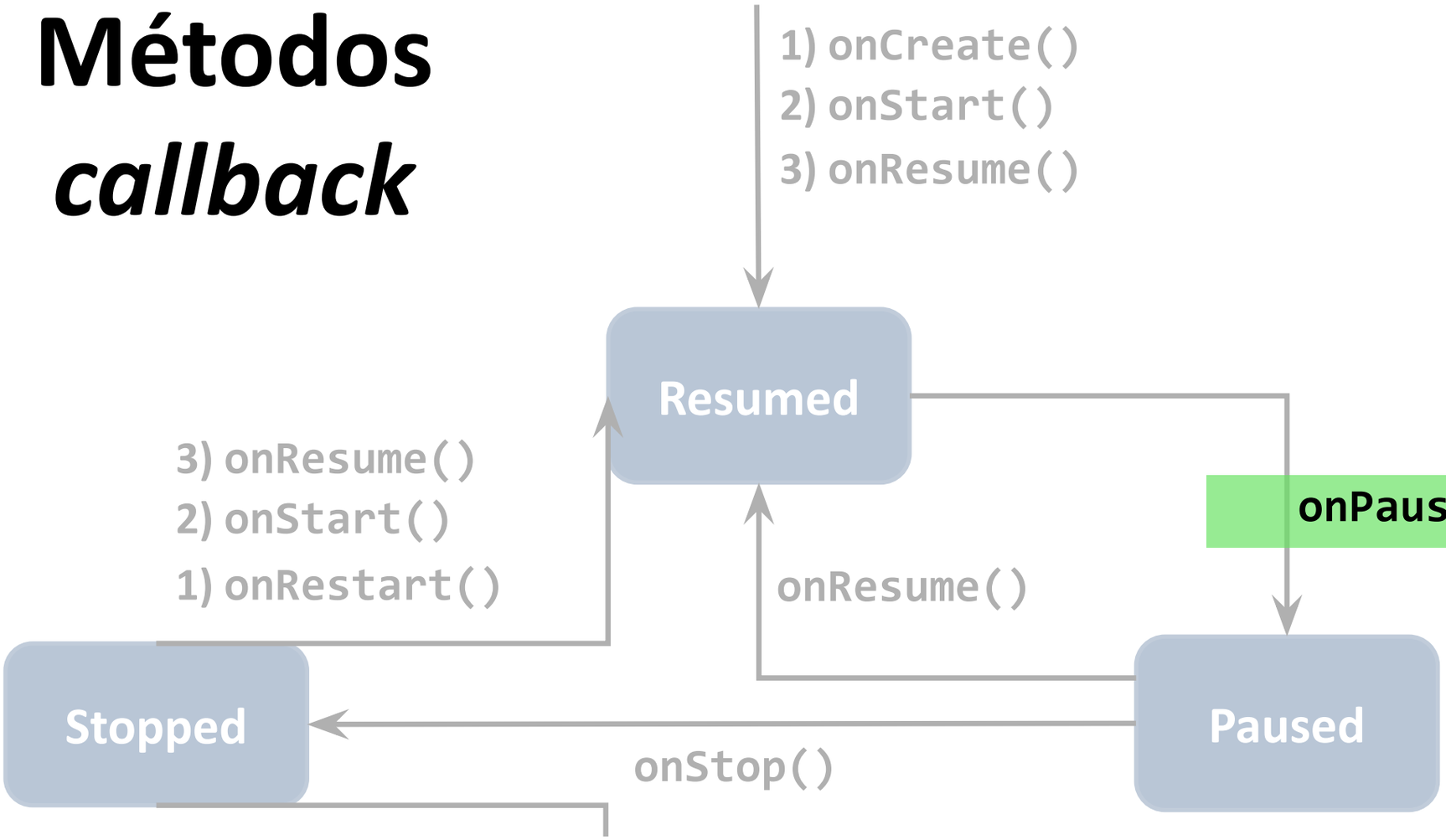
Ciclo de vida



- **onResume**() se ejecuta justo antes de que la actividad sea completamente visible y obtenga el foco.
- Es el sitio indicado para iniciar animaciones, acceder a recursos de forma exclusiva como la cámara, etc.

Métodos *callback*

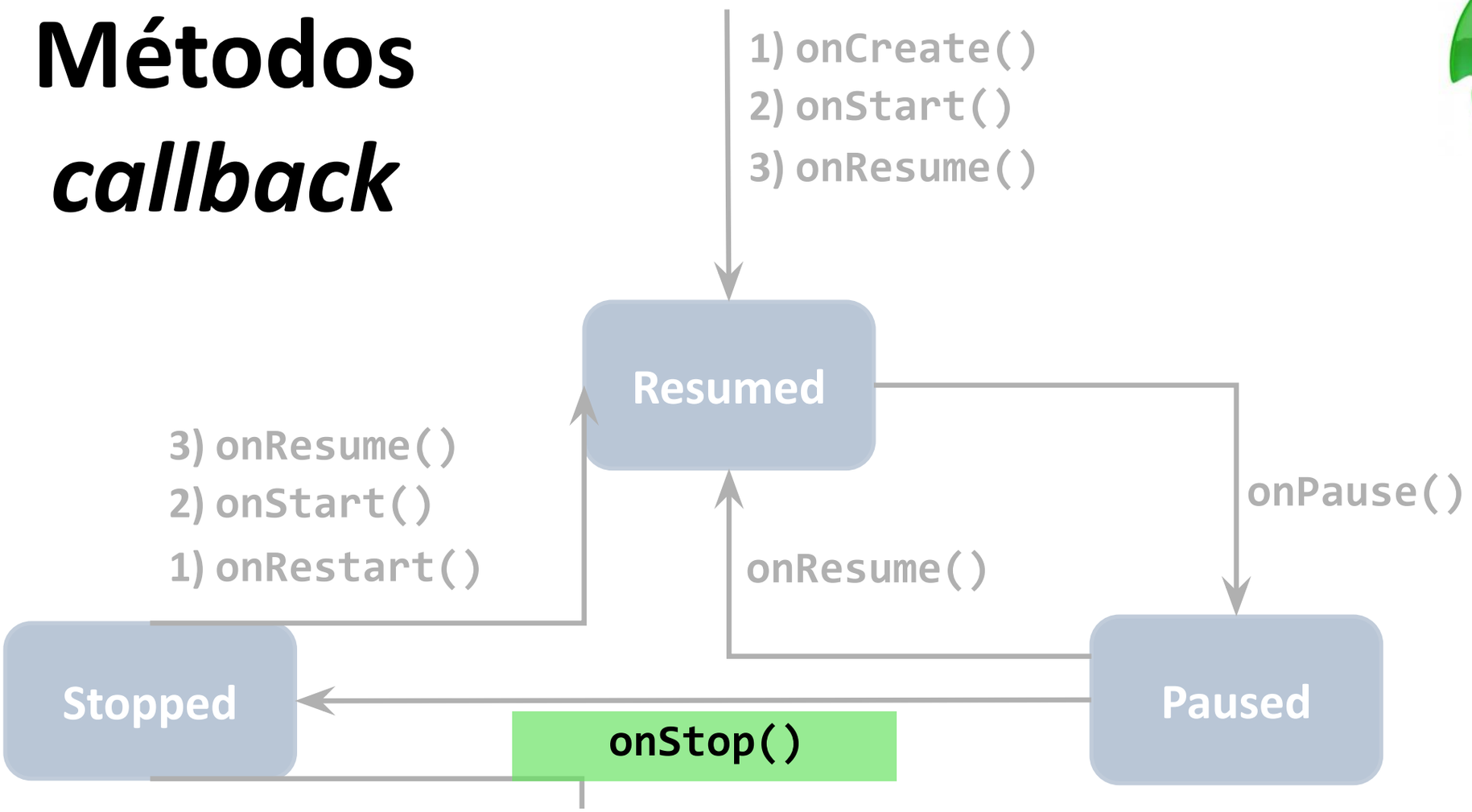
Ciclo de vida



- **onPause**() se ejecuta cuando la *activity* actual pierde el foco, porque va a ser reemplazada por otra. Es el sitio ideal para detener todo lo que se ha activado en **onResume**() y liberar los recursos de uso exclusivo.
- La ejecución de este método debería ser lo más rápida posible, puesto que el usuario no podrá utilizar la nueva actividad hasta que éste finalice

Métodos *callback*

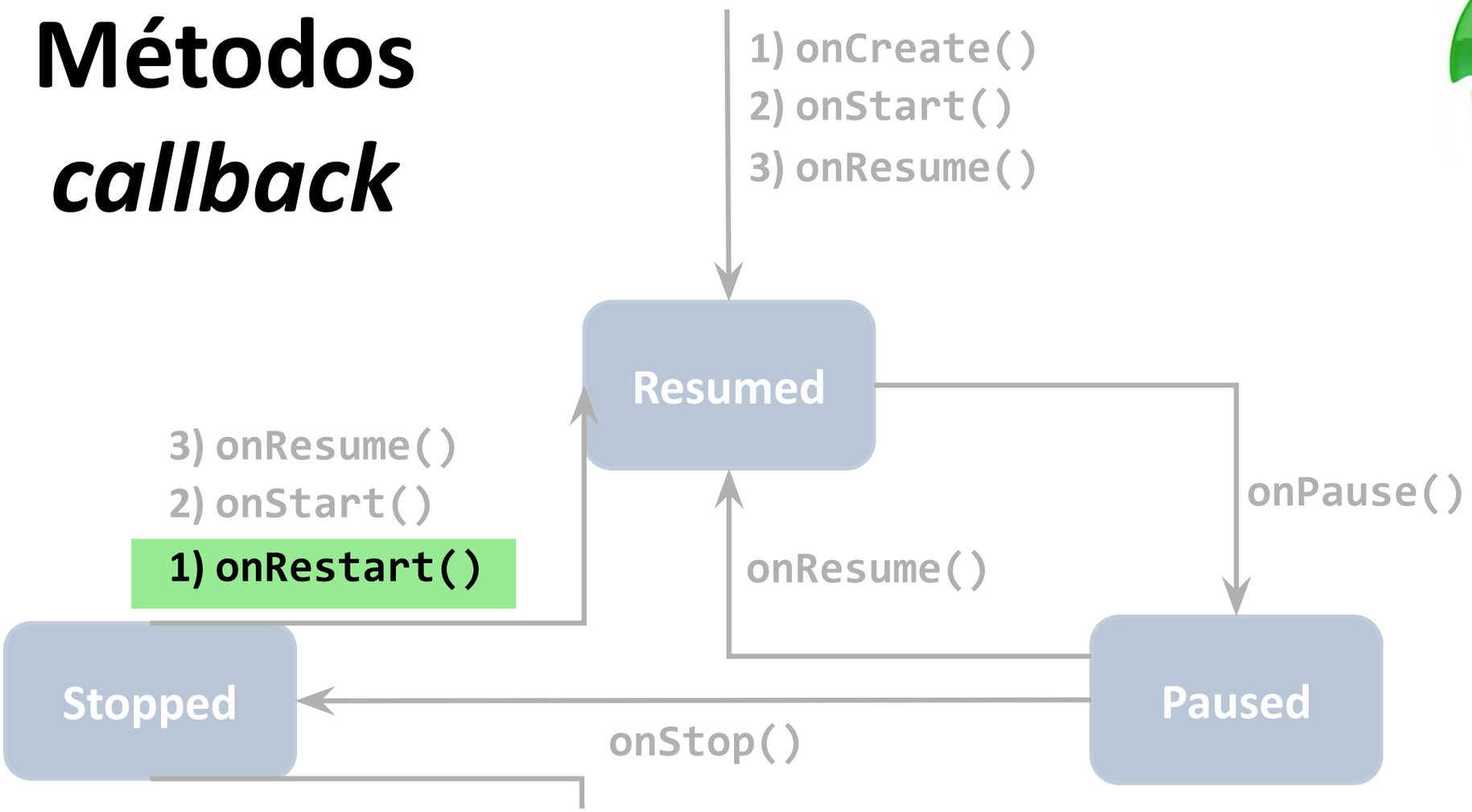
Ciclo de vida



- **onStop**() es invocado cuando la *activity* ha sido ocultada completamente por otra que ya está interactuando con el usuario.
- Aquí se suelen destruir los procesos creados en el método **onStart**().

Métodos *callback*

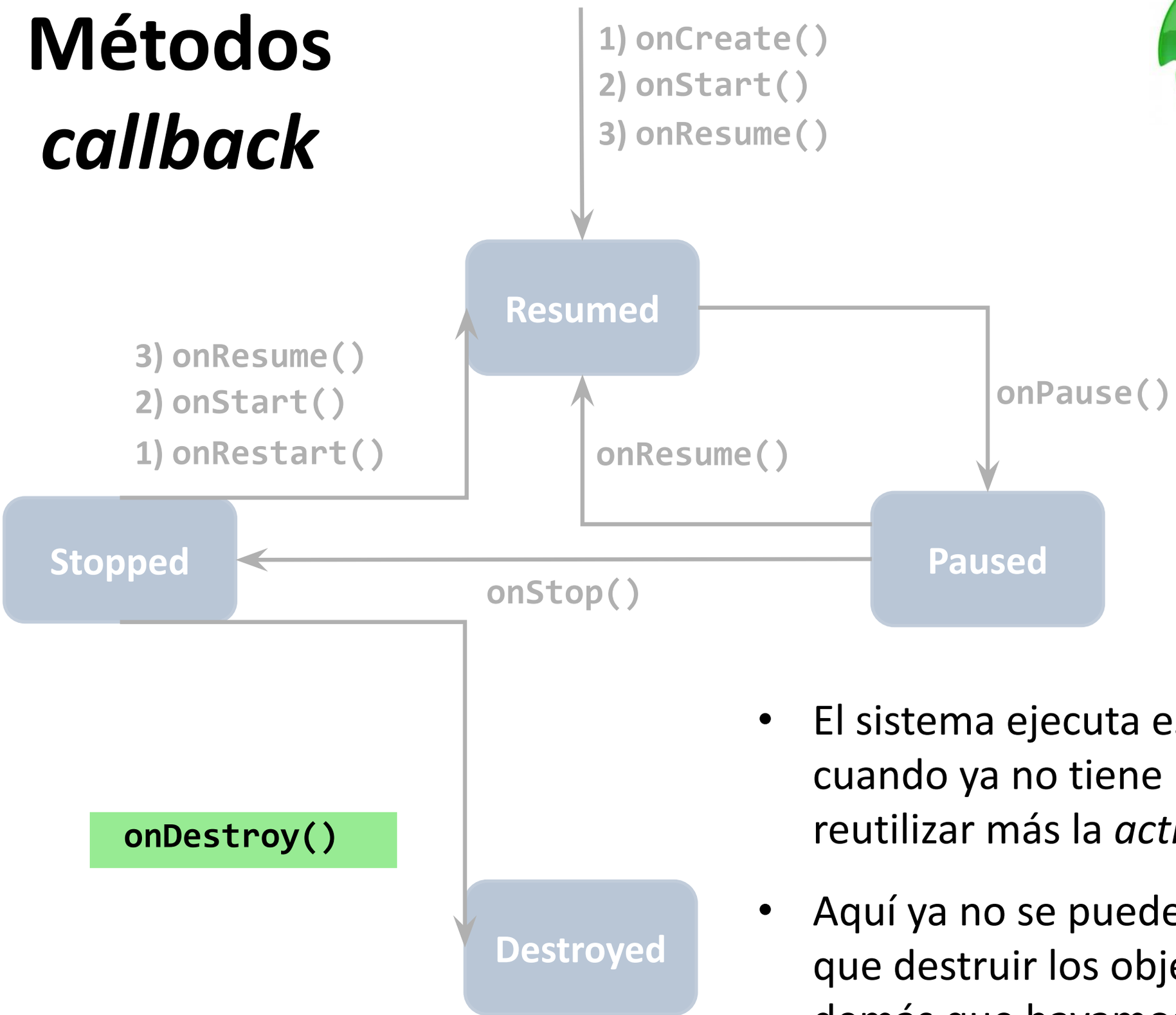
Ciclo de vida



- **onRestart**() se ejecuta cuando una *activity* que había sido ocultada (pero no destruida) tiene que mostrarse de nuevo.
- Es poco utilizado pero puede ser útil en algunos casos.

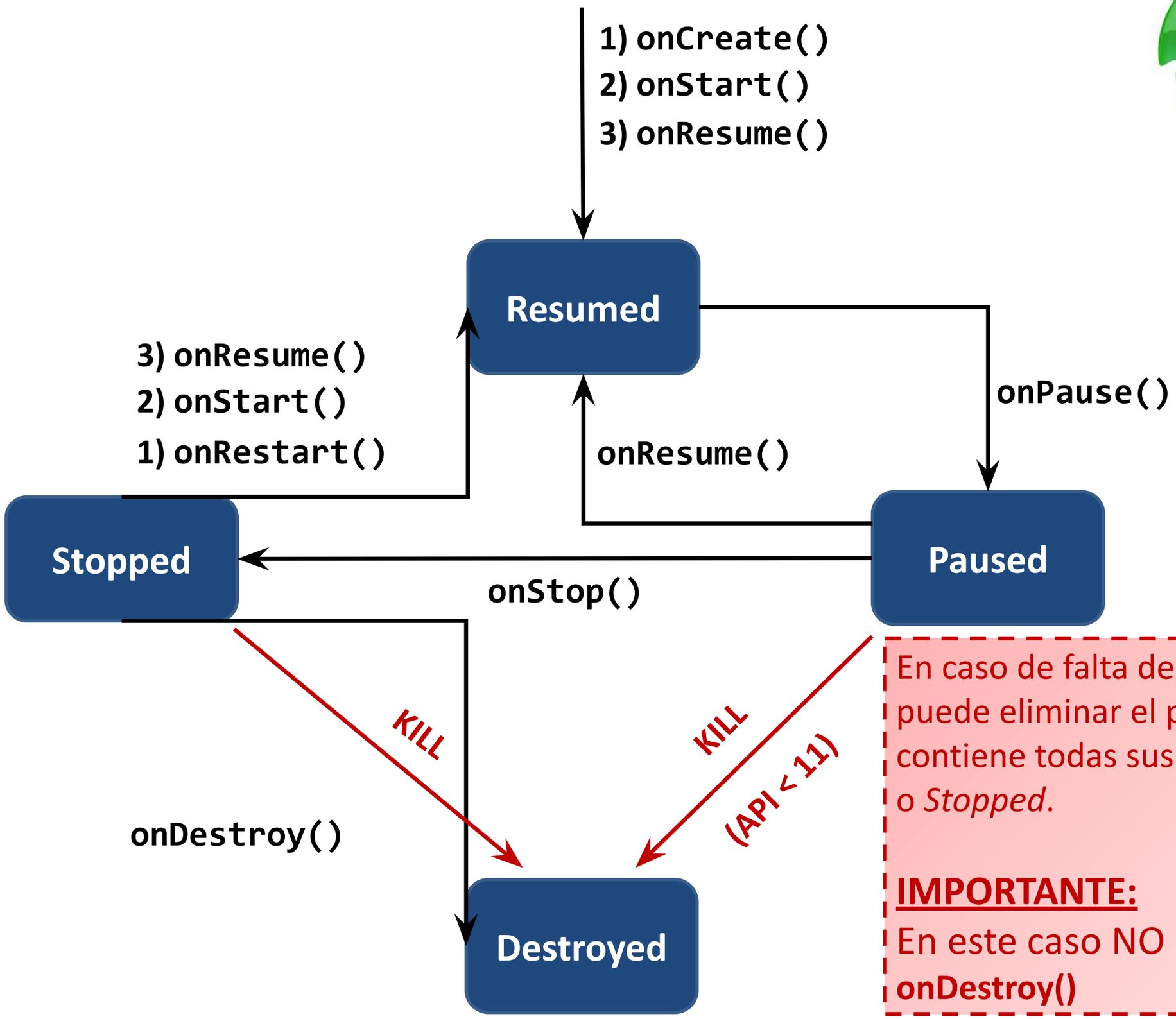
Métodos *callback*

Ciclo de vida



- El sistema ejecuta este método cuando ya no tiene intención de reutilizar más la *activity*.
- Aquí ya no se puede hacer otra cosa que destruir los objetos, hilos y demás que hayamos creado en **onCreate**().

Ciclo de vida



En caso de falta de recursos el S.O. puede eliminar el proceso que contiene todas sus *activities* en *Paused* o *Stopped*.

IMPORTANTE:

En este caso NO se ejecuta **onDestroy()**

Ciclo de vida

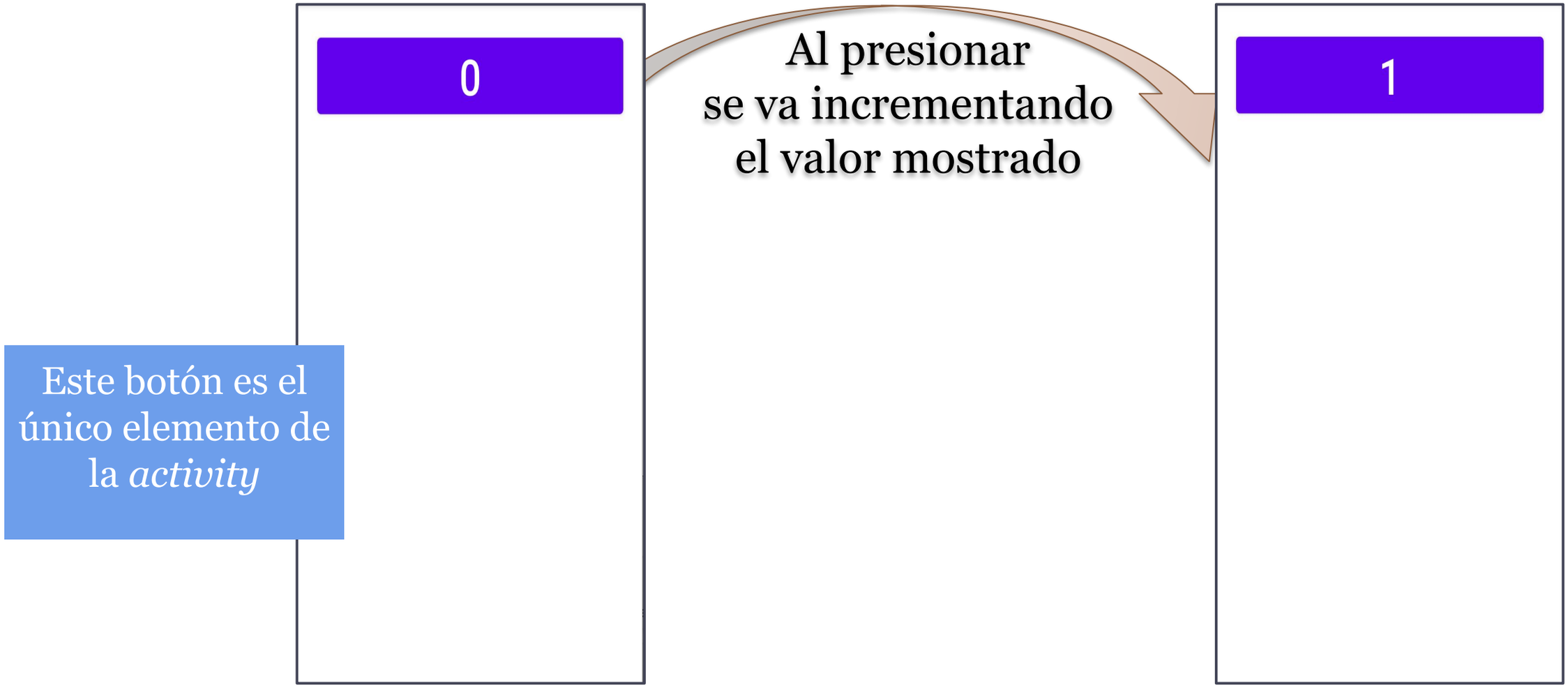
Además existen otras situaciones en las que programáticamente se pueden conseguir **circuitos infrecuentes** en el ciclo de vida, por ejemplo:

Si en el método **onCreate()** llamamos a **this.finish()** se invoca inmediatamente **onDestroy()** y se pasa directamente al estado *Destroyed.*

Si en el método **onStart()** invocamos **this.finish()** se saltea el estado *Resumed* y pasa directamente a *Stopped* para luego alcanzar *Destroyed*, es decir, a **onStart**() le sigue **onStop()** y luego **onDestroy().**

Ejercicio

- Programar una activity con un botón (Button) cuyo texto indique la cantidad de veces que ha sido pulsado. Cada vez que el usuario lo presiona debe incrementarse en uno dicho valor.



Ejercicio - Resolución

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
  <Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="0"
    android:onClick="incrementar"
    android:layout_margin="20sp"
    android:textSize="40sp"
    android:id="@+id/boton"/>


</LinearLayout>
```

En **`onClick`** se especifica el nombre del método de la *activity* que se ejecutará al presionar el botón

Ejercicio - Resolución

```
class MainActivity : AppCompatActivity() {
    var contador = 0;
    lateinit var boton: Button;

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        boton = findViewById<Button>(R.id.boton);
    }

    fun incrementar(v: View){
        contador ++;
        boton.setText(contador.toString())
    }
}
```

Este método se ejecuta cada vez que el usuario presiona el botón

Ejercicio - Cuestionario

- **P:** ¿Qué ocurre con el contador que se visualiza en el texto del botón cuando se cambia la orientación del dispositivo entre vertical y horizontal? ¿A qué se debe tal comportamiento?

R: El contador vuelve a cero. Ello se debe a que cada vez que rotamos el dispositivo la *activity* en primer plano (*Resumed*) es destruida y creada nuevamente perdiéndose los valores de sus variables de instancia.

Ejercicio - Cuestionario

- Este comportamiento Destrucción-Creación también ocurre cuando se cambia alguna opción global de configuración del sistema, como por ejemplo el idioma del dispositivo.

P: ¿Por qué cree que ocurre esto?

R: La razón es que cualquier recurso de la aplicación, puede cambiar en función de los valores de configuración. La forma más segura de manejar un cambio de configuración es volver a recuperar todos los recursos (incluidos los *layouts*, *drawables* y strings) y reconstruir la *activity*

¿Cómo podemos recuperar el estado de una activity luego de un proceso de Destrucción-Creación ?

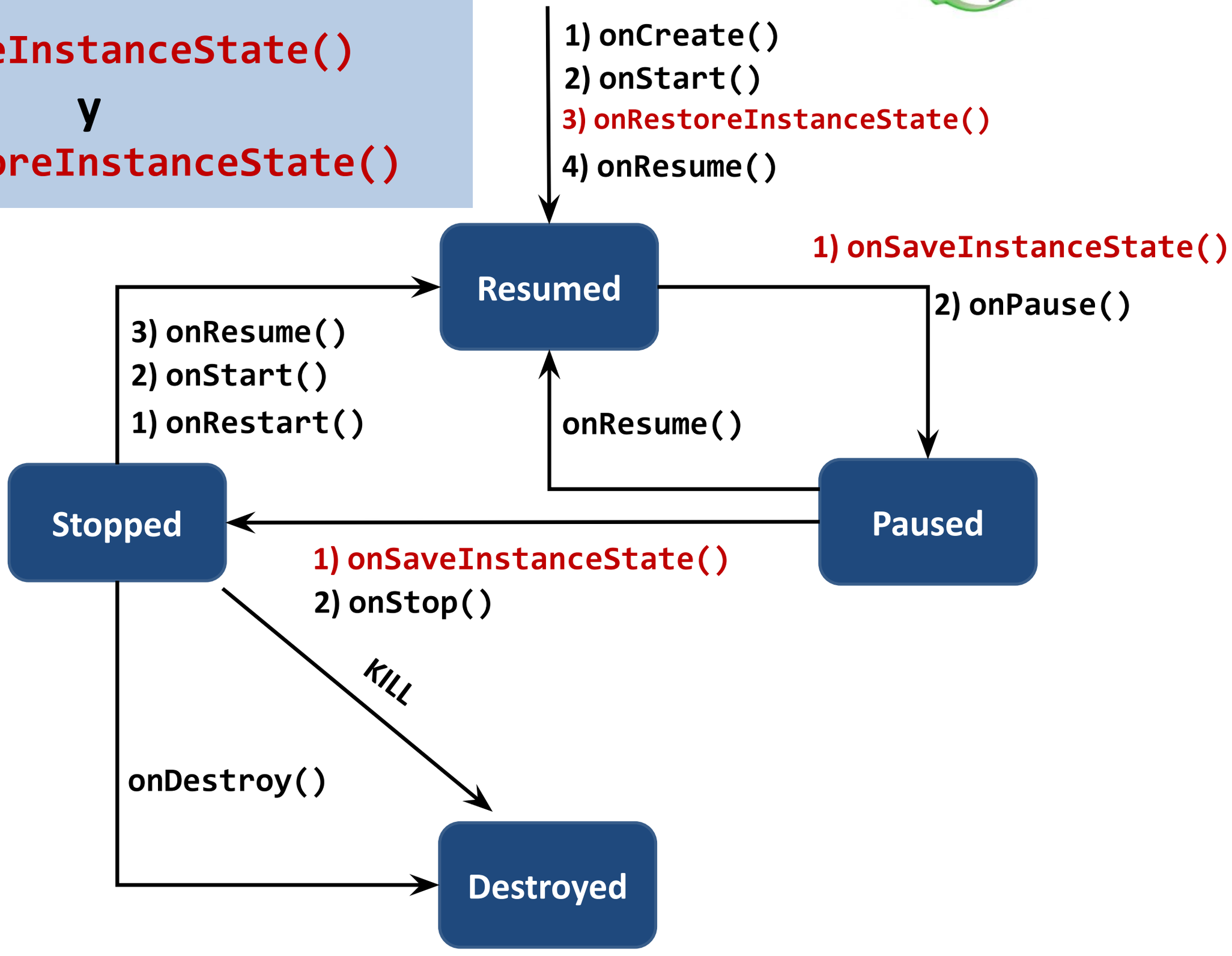
Ciclo de vida

Con dos nuevos *callbacks*:

`onSaveInstanceState()`

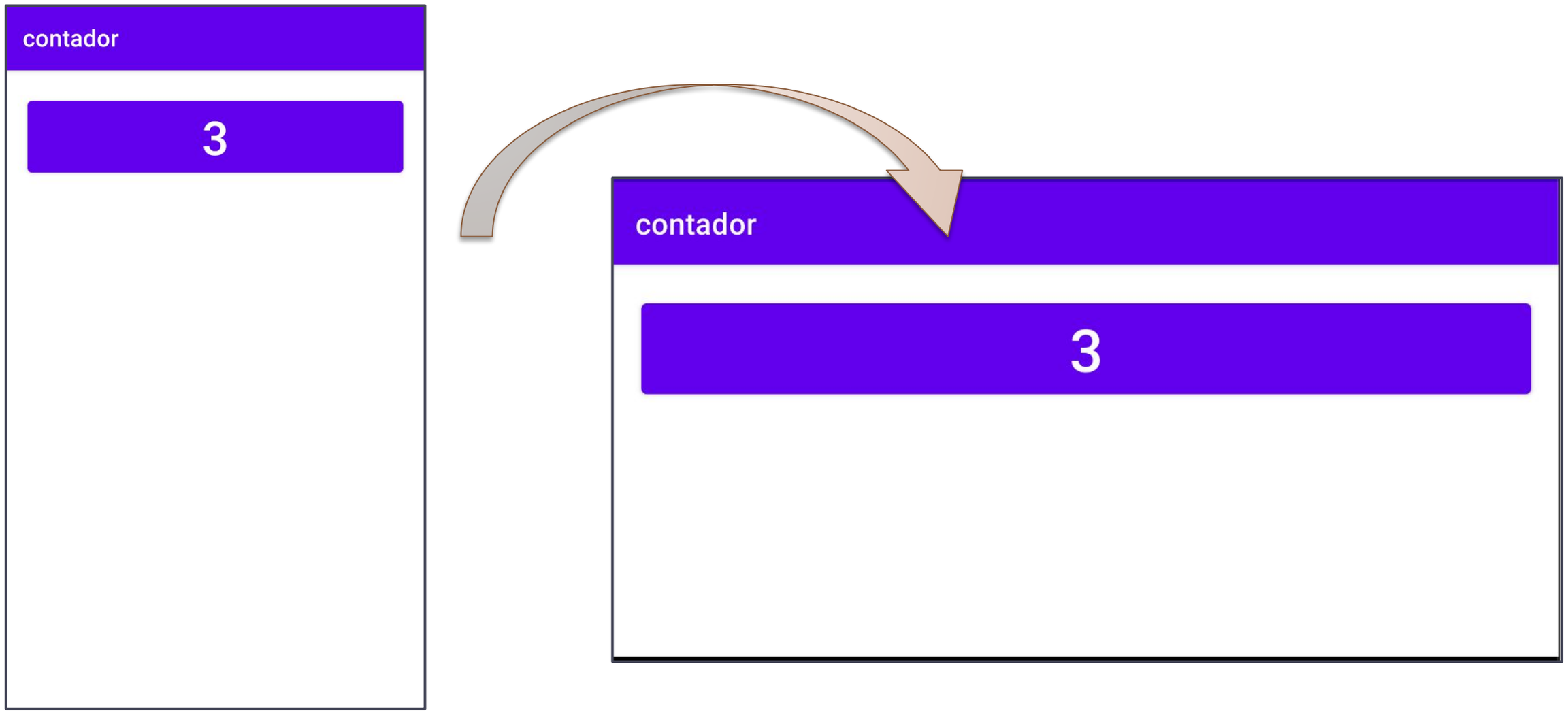
y

`onRestoreInstanceState()`



Ejercicio

- Modificar la *activity* del ejercicio anterior para que al cambiar la orientación del dispositivo se reconstruya adecuadamente (no se debe perder el valor de contador)



Ejercicio - Solución

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putInt("contador", contador)
}
```

```
override fun
onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)
    contador = savedInstanceState.getInt("contador");
    boton.setText(contador.toString());
}
```

Agregar estos dos métodos

Ejercicio

- Modificar la *activity* del ejercicio anterior, agregando un EditText.
- Se debe agregar orientation="vertical" en el LinearLayout para que los widget se muestren correctamente.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Modificame..."
    />
```

Ejercicio - Cuestionario

Modificar el valor del EditText y cambiar la orientación del dispositivo

- **P:** ¿Qué ocurre con el valor ingresado?

R: El EditText vuelve a “Modificame”.
Observamos el mismo problema que en el ejercicio del contador.
Ahora… agregar un ID al EditText y volver a probar…

La implementación predeterminada del método **onSaveInstanceState()** en la clase (superclass) **Activity** llama al método del mismo nombre de cada **View** del diseño. Prácticamente todos los *widgets* del *framework* de Android implementan este método según les convenga. Sin embargo, para que funcione correctamente es necesario establecer el atributo **android:id** para cada *widget* cuyo estado se desee guardar.

Por este motivo es que, en nuestras redefiniciones, debemos invocarlos (usando **super**) y encargarnos de guardar y recuperar los datos particulares de nuestra clase.

Ejercicio - Cuestionario

- **P:** ¿Qué ocurre con el contador y con el editText si cierro la aplicación (back button) y luego la ejecuto nuevamente? ¿A qué se debe tal comportamiento?

R: Se pierden ambos valores. Ello se debe a que no se persiste el estado de la aplicación una vez que la aplicación fue “destruida”. Para ese caso se debería utilizar otro mecanismo de persistencia de datos (en clases posteriores veremos alternativas)

Por lo tanto… **onSaveInstanceState ()** y **onRestoreInstanceState() NO siempre** son invocados**.**

onSaveInstanceState () se invoca cuando una *activity* que pierde el foco y espera ser reanudada (el usuario no la cerró explícitamente) está en riesgo de ser destruida.

- Esto se da por ejemplo cuando se lanza una *activity* sobre la actual. La *activity*, ahora en segundo plano, podría ser destruida si el sistema necesita memoria. Sin embargo, al presionarse el botón "Atrás" debería *re-crearse* la primera *activity* recuperando el estado anterior a ser destruida.
- También ocurre al cambiar la orientación del dispositivo o al realizar algún cambio de configuración

onRestoreInstanceState() se utiliza para recuperar los datos almacenados en **onSaveInstanceState()**. Muchas veces no se usa porque la recuperación también se puede hacer en el **onCreate()**.

IMPORTANTE

Ciclo de vida

Nota 1: No se garantizan llamadas a **onSaveInstanceState()** antes de destruir la *activity*, ya que hay casos en los cuales no será necesario guardar el estado (como cuando el usuario sale de la *activity* con el botón *Atrás*, porque cierra la *activity* explícitamente). Si el sistema llama a **onSaveInstanceState()** , lo hace antes de llamar a **onStop()** y posiblemente antes de llamar a **onPause()**

Nota extraída del sitio Android Developers
https://developer.android.com

IMPORTANTE

Ciclo de vida

Nota 2: Dado que no se garantiza la llamada a **onSaveInstanceState()**, debe usarse sólo para registrar el estado transitorio de la *activity* (el estado de la IU); nunca se debe usar para almacenar datos persistentes. En su lugar, debe usarse **onPause()** para guardar datos persistentes (como datos que deben guardarse en una base de datos) cuando el usuario abandona la actividad.

Nota extraída del sitio Android Developers
https://developer.android.com