Modulo 5

Programmazione base in Android

5.1

Applicazioni ed Activity

5.1 Applicazione (Riassunto)

Android package: file `.apk`

Vive nella sua *sandbox*:

- Ogni applicazione appartiene ad un utente Linux,
- Ha uno spazio dati privato,
- Viene eseguito in una istanza privata di Davik...
- ...in un processo isolato Linux.

5.1 Applicazione (Riassunto)

Android package: file `.apk`

Vive nella sua *sandbox*:

- Ogni applicazione appartiene ad un utente Linux,
- Ha uno spazio dati privato,
- Viene eseguito in una istanza privata di Davik...
- ...in un processo isolato Linux.

Le applicazioni *non* possono comunicare...

5.1 Applicazione (Riassunto)

Android package: file `.apk`

Se non attraverso i seguenti metodi:

- **Intent** (messaggio asincrono),
- **Dati condivisi** (spazio pubblico SD o *user ID* condiviso),
- **Service binding**,
- **Content provider**.

Le applicazioni *non* possono comunicare...

5.1 Applicazione (Eccezioni)

User ID condiviso:

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:sharedUserId="it.neunet.userId"
     android:sharedUserLabel="@string/sharedUserLabel"
```

Processo condiviso (o separato):

```
<activity
    android:name="eu.neunet.activitytest.PrimaryActivity"
    android:label="@string/app_name"
    android:process=":myProcess"
```

5.1 Applicazione (Componenti)

Non esiste un singolo *entry point*.

~~`int main(void);`~~

Ogni applicazione può avere 1+ di:

- Activity,
- Service,
- Content provider,
- Broadcast receivers.

5.1 Applicazione (Co

Non esiste un singolo *en*

~~`int main(`~~

Ogni applicazione può av

- Activity,
- Service,
- Content provider,
- Broadcast receivers.

Ognuno di questi componenti è un *entry point* a se.

Può essere avviato dal sistema operativo, a partire da un `Intent` lanciato da chiunque.

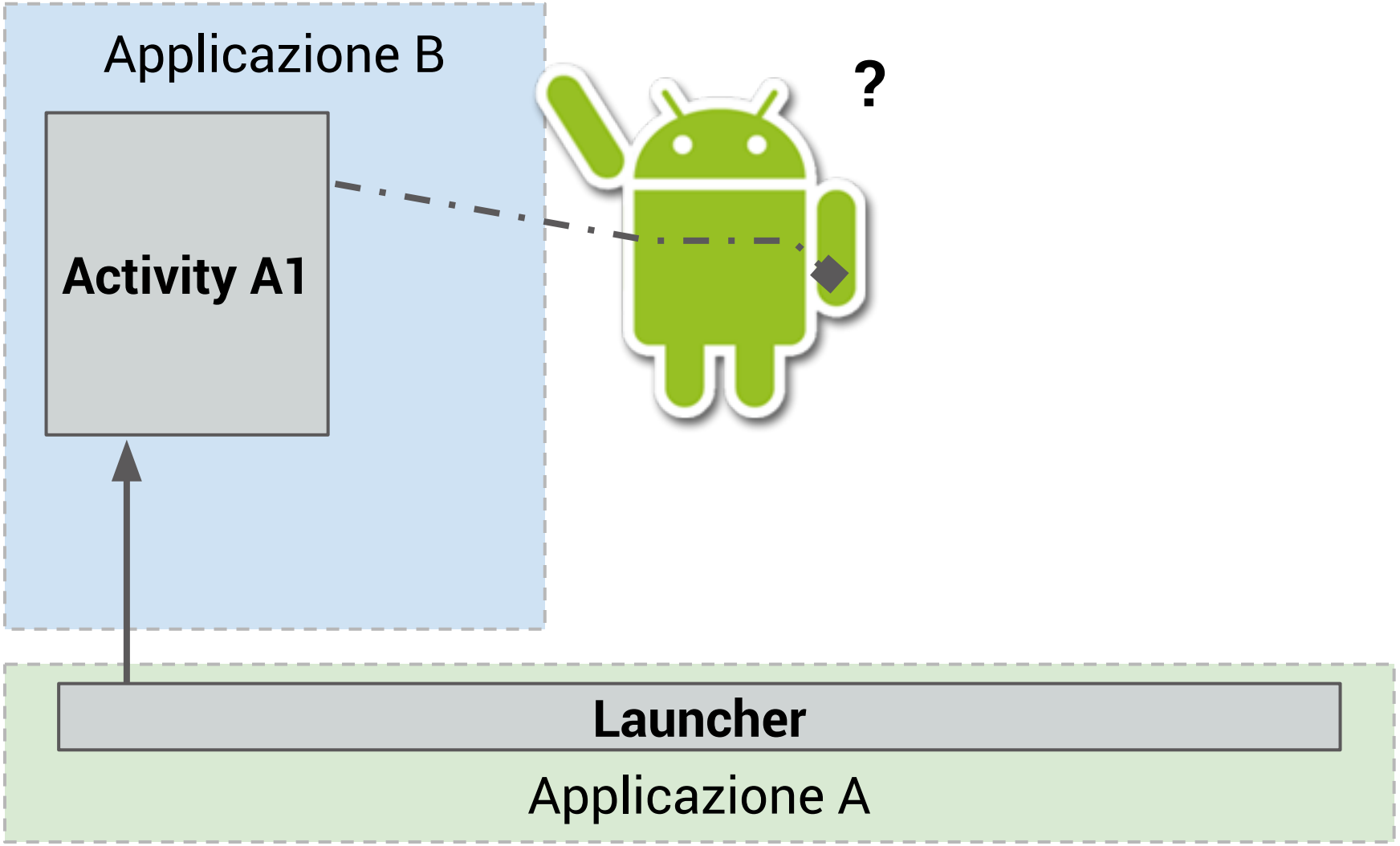
Stessa applicazione/ processo/thread.

5.1 Interazione tra applicazioni

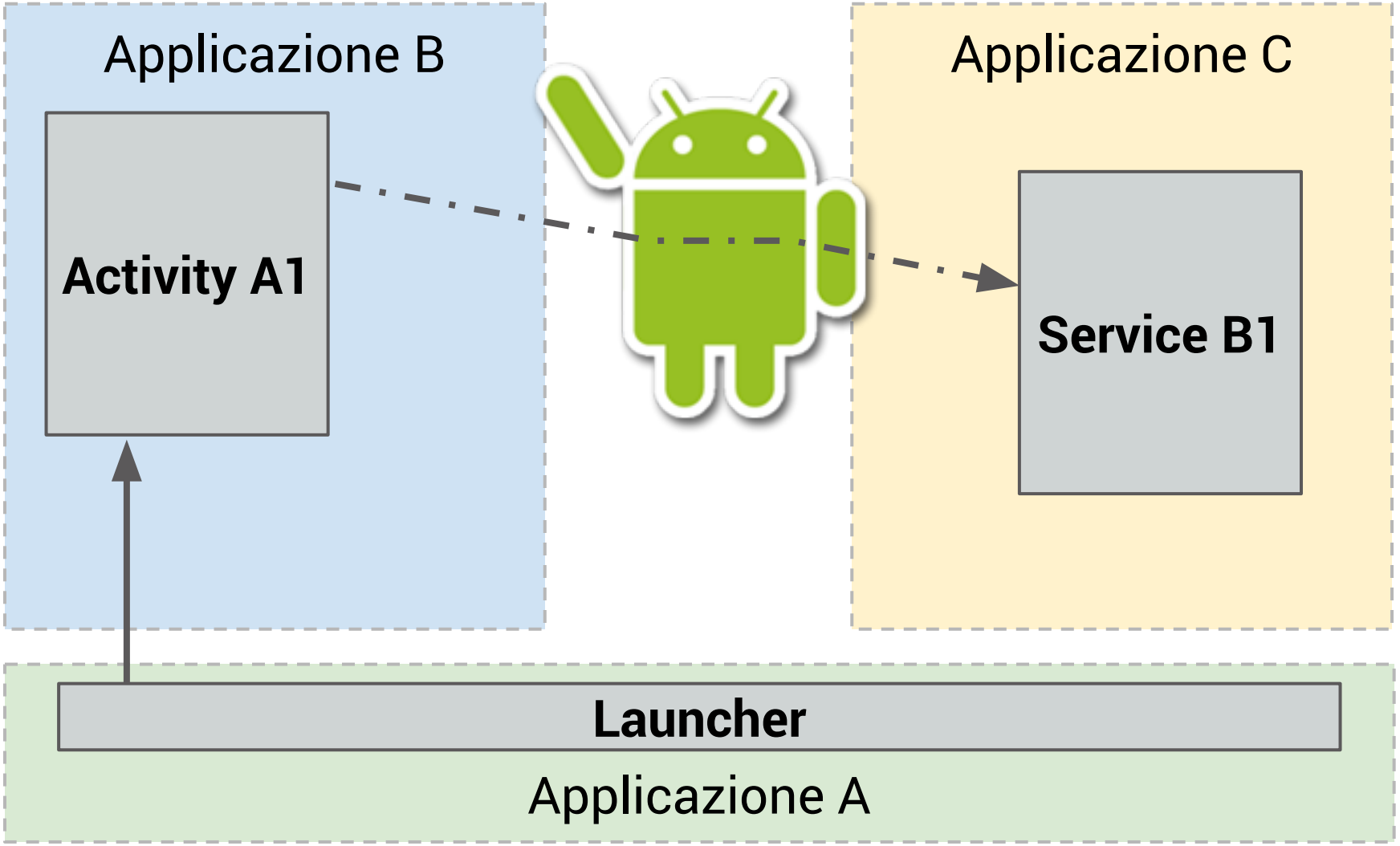
Launcher

Applicazione A

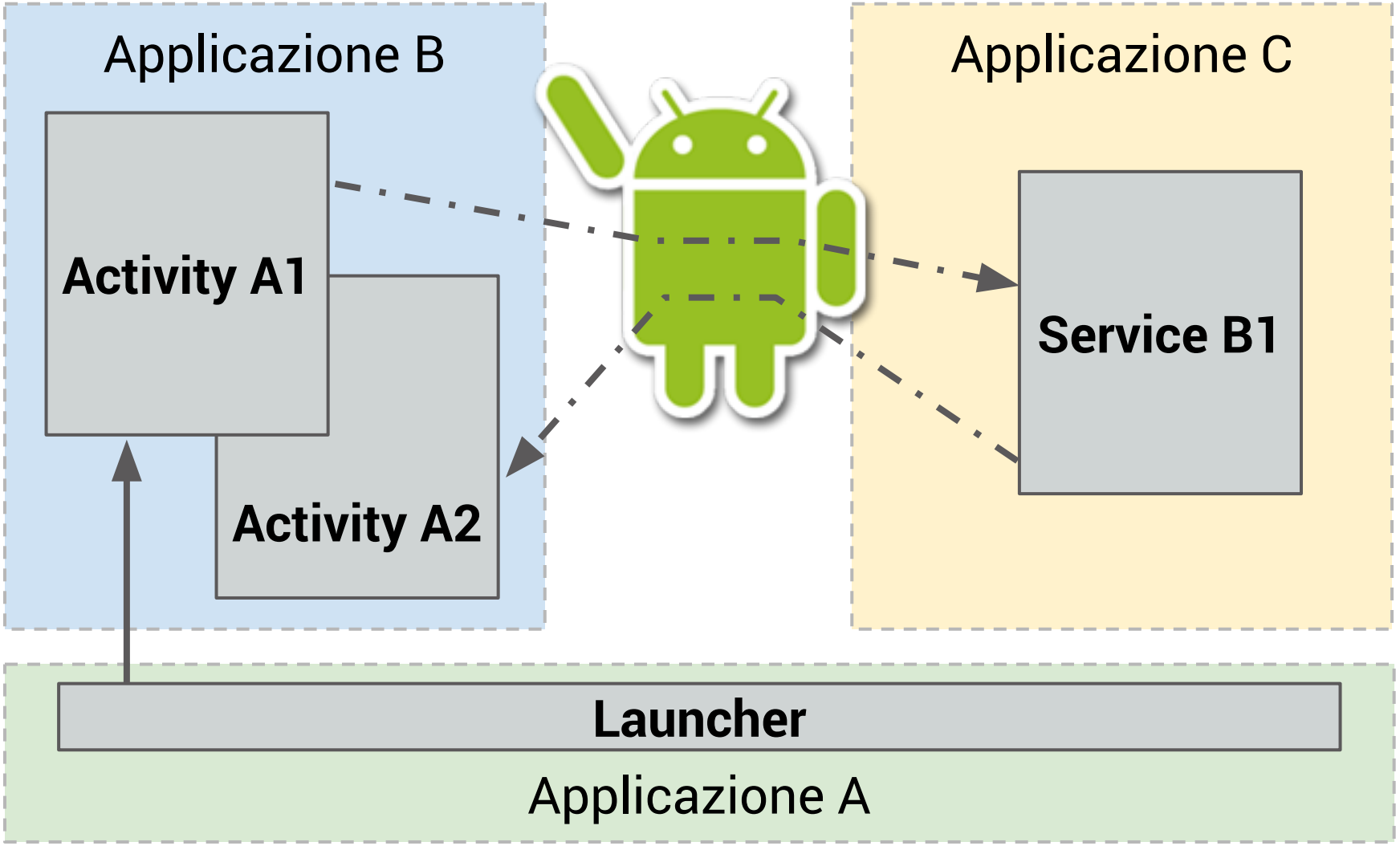
5.1 Interazione tra applicazioni



5.1 Interazione tra applicazioni



5.1 Interazione tra applicazioni



5.1 Ciclo di vita dei processi

Android tenta di tenere in vita i processi che contengono applicazioni attive.

1. Processi **foreground**
2. Processi **visibili**
3. Processi di **servizio**
4. Processi **background**
5. Processi **vuoti**

5.1 Ciclo di vita dei proc

Android tenta di tenere in vita
contengono applicazioni atti

1. Processi **foreground**
2. Processi **visibili**
3. Processi di **servizio**
4. Processi **background**
5. Processi **vuoti**

- Activity con cui l'utente interagisce.
- Service che ha fatto *bind* ad un activity attiva.
- Service in "foreground".
- Componente che sta eseguendo i suoi metodi per la gestione del ciclo di vita.

5.1 Ciclo di vita dei processi

Android tenta di tenere in vita i processi che contengono applicazioni attive.

1. Processi **foreground**
2. Processi **visibili**
3. Processi di **servizio**
4. Processi **background**
5. Processi **vuoti**

5.1 Ciclo di vita dei proc

Android tenta di tenere in vita
contengono applicazioni atti

1. Processi **foreground**
2. Processi **visibili**
3. Processi di **servizio**
4. Processi **background**
5. Processi **vuoti**

Nessun componente attivo, eccetto `Activity` non visualizzate (stop).

5.1 Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/..."
    package="it.neunet.csa.activitytest"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
```

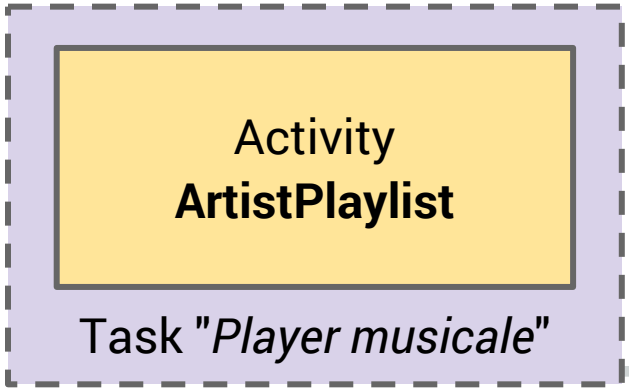
5.1 Dichiarazione Activity

```
<activity
    android:name="eu.neunet.csa.activitytest.PrimaryActivity"
    android:label="@string/app_name"
    android:launchMode="standard">

        <intent-filter>
            <!-- ... -->
        </intent-filter>
</activity>

<!-- altre activity... -->
```

5.1 Proprietà: taskAffinity

Gruppo concettuale dove l'Activity viene eseguita.

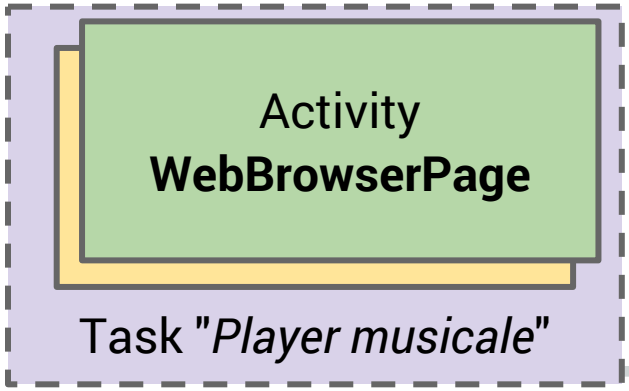
Se lanciata con `FLAG_ACTIVITY_NEW_TASK`, l'Activity viene sempre raggruppata con un task con l'affinità specificata.



5.1 Proprietà: taskAffinity

Gruppo concettuale dove l'Activity viene eseguita.

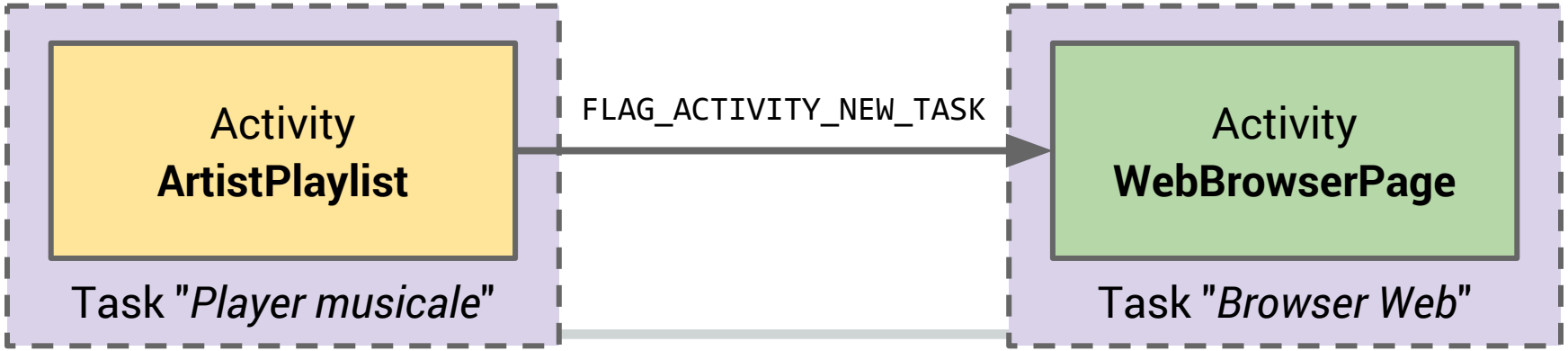
Se lanciata con `FLAG_ACTIVITY_NEW_TASK`, l'Activity viene sempre raggruppata con un task con l'affinità specificata.



5.1 Proprietà: taskAffinity

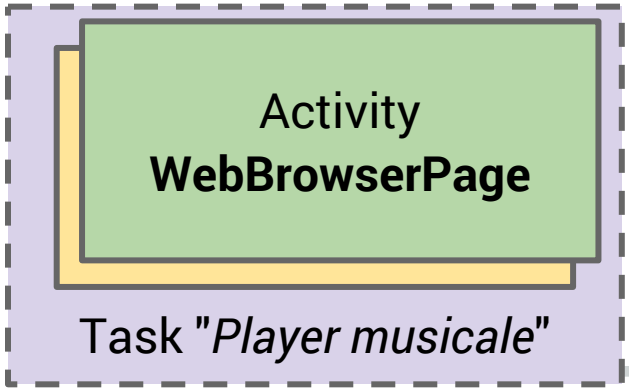
Gruppo concettuale dove l'Activity viene eseguita.

Se lanciata con `FLAG_ACTIVITY_NEW_TASK`, l'Activity viene sempre raggruppata con un task con l'affinità specificata.



5.1 Proprietà: allowTaskReparenting

Activity con questa proprietà possono essere "spostate" da un task ad un task per il quale hanno affinità (quando va in foreground).



5.1 Proprietà: allowTaskReparenting

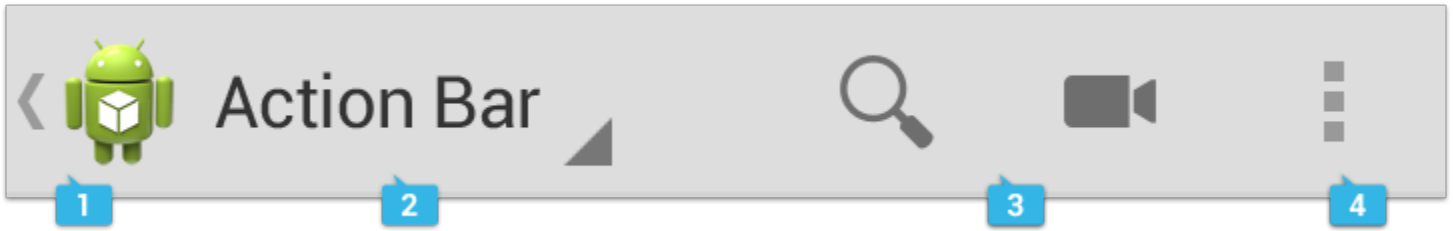
Activity con questa proprietà possono essere "spostate" da un task ad un task per il quale hanno affinità (quando va in foreground).

5.1 Proprietà: allowTaskReparenting

Activity con questa proprietà possono essere "spostate" da un task ad un task per il quale hanno affinità (quando va in foreground).

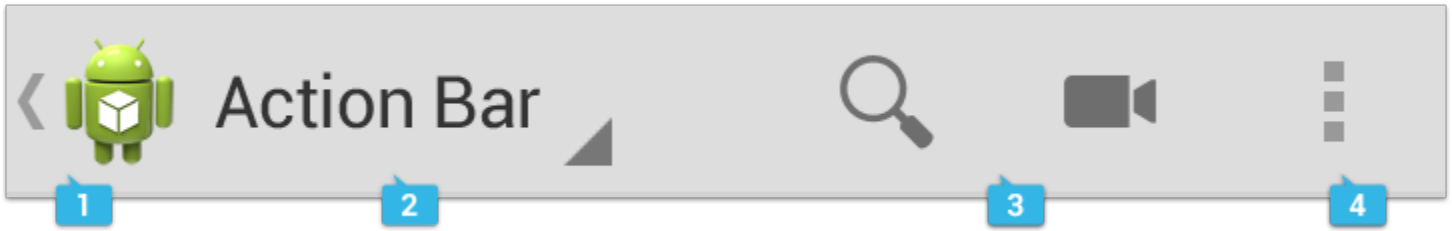
5.1 *Preview:* navigazione "up"

Da Honeycomb (API level 11): **Action Bar**!



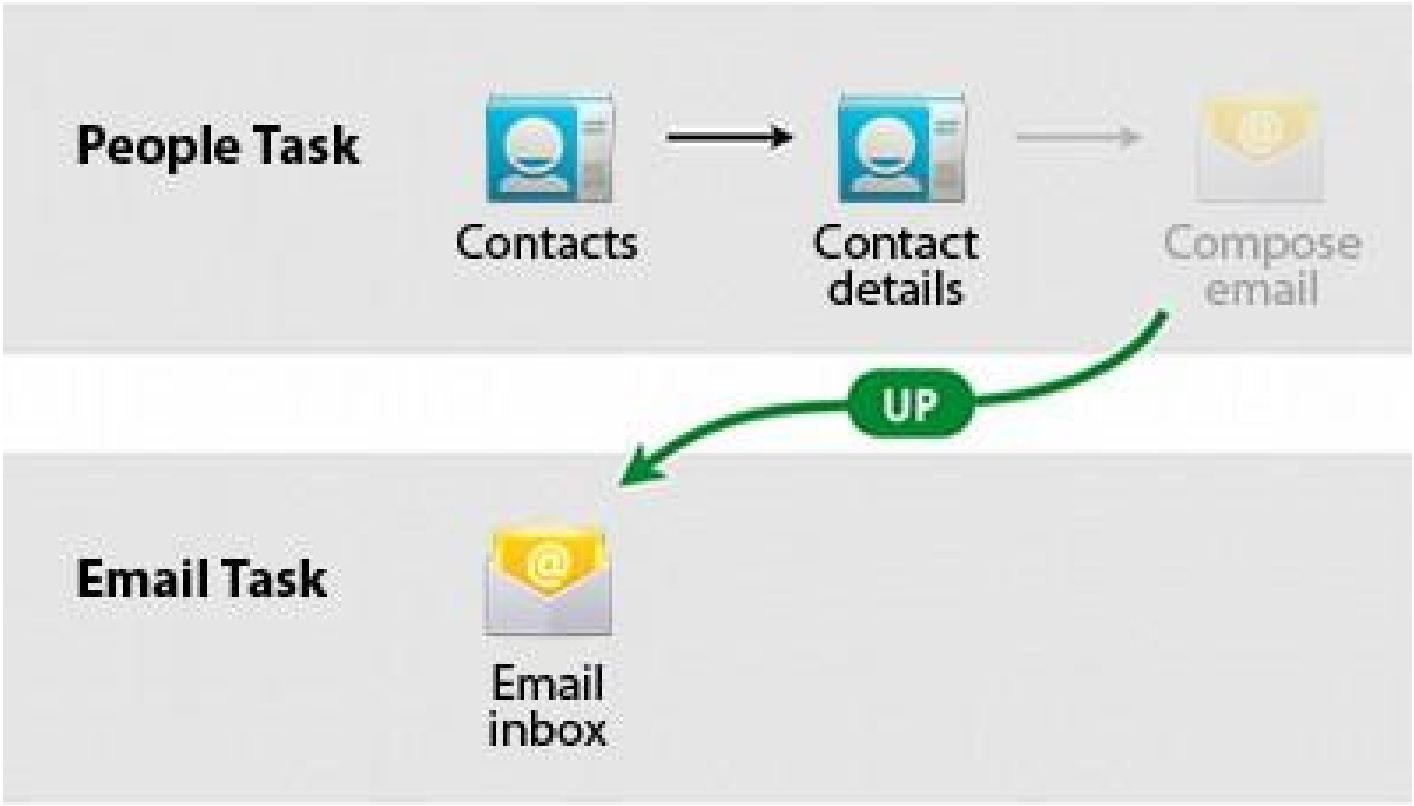
5.1 *Preview:* navigazione "up"

Da Honeycomb (API level 11): **Action Bar**!



5.1 *Preview:* navigazione "up"

Da Honeycomb (API level 11): **Action Bar**!



5.1 Proprietà: finishOnTaskLaunch

Tutte le istanze dell'Activity vengono terminate quando un nuovo task è avviato con questa Activity.

È più stringente di `allowTaskReparenting`:

activity che verrebbero spostate vengono distrutte ed una nuova istanza viene creata nel nuovo task.

5.1 Proprietà: exported

Default: `true` se ha un `intent-filter`.

Solo Activity esportate possono essere instanziate da un Intent *esterno* all'applicazione stessa.

Limita l'esposizione di componenti esterni dell'applicazione.

5.1 Proprietà: alwaysRetainTaskState

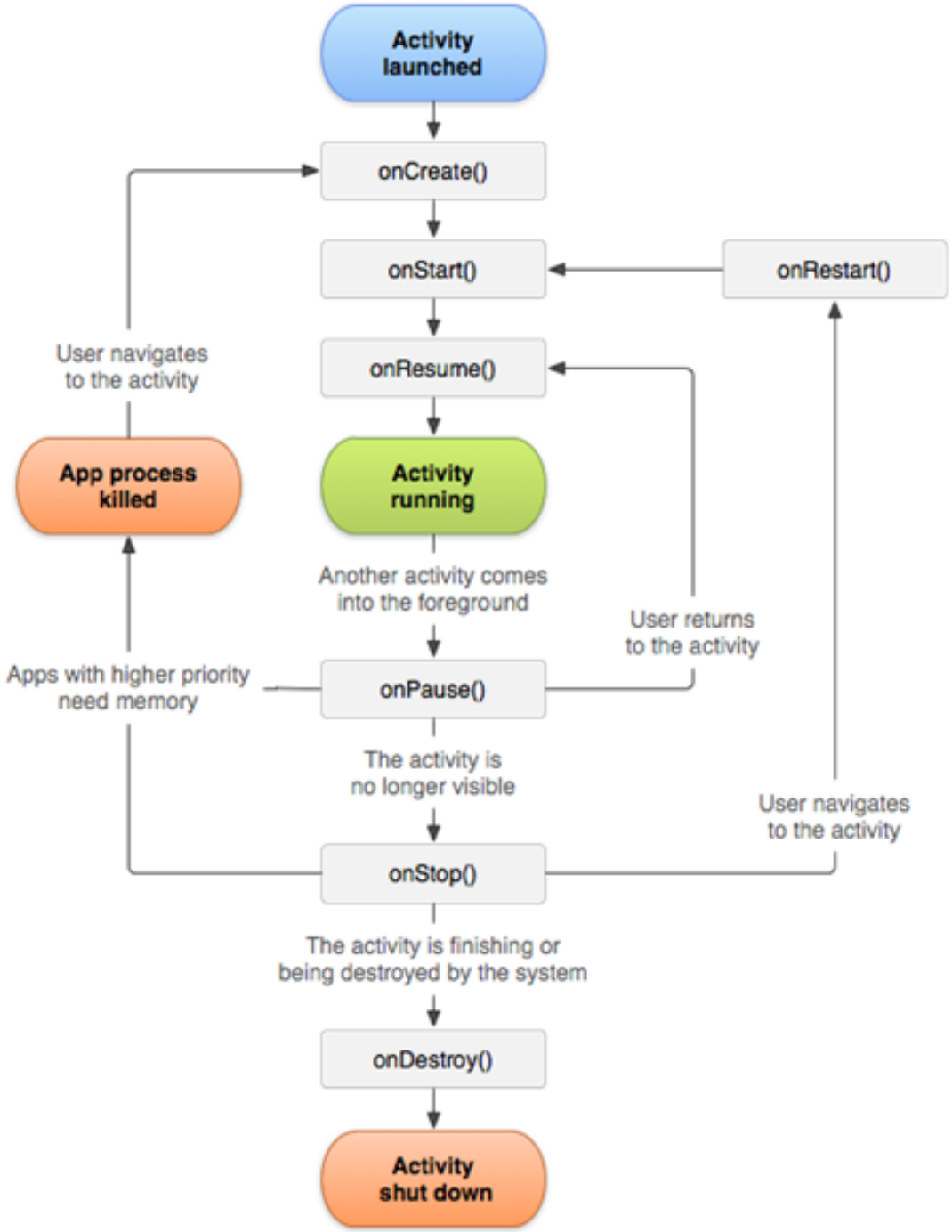
Default: `false`.

Forza il sistema a mantenere lo stato di tutte le Activity del task (va specificato per l'Activity root) - anche a lungo termine.

Esempio: web browser.

5.1 Proprietà: launchMode

Modalità	Conseguenze
Default	Viene creata una nuova activity. Istanze diverse della stessa possono appartenere a diversi task. Ogni task può avere più istanze della stessa.
SingleTop	Viene creata una nuova istanza solo se l'activity non è al top dello stack (onNewIntent()). Il resto come Default.
SingleTask	Se c'è già un'istanza in un task viene riavviata quella altrimenti viene avviata una nuova activity in un nuovo task. Può esserci sempre solo un'istanza dell'activity nel sistema.
SingleInstance	Come sopra ma l'activity è sempre l'unica presente nel suo task.



Ciclo di vita

5.1 Demo

Ciclo di vita e persistenza

Intermezzo 1

Factory pattern

5.1 Factory method

Pattern creazionale, permette la creazione di oggetti tramite chiamata a metodo.

Di solito:

```
Foo foo = new Foo();
```

5.1 Factory method

Pattern creazionale, permette la creazione di oggetti tramite chiamata a metodo.

Di solito:

```
Foo foo = new Foo();
```

L'allocazione è strettamente accoppiata a quel particolare costruttore di `Foo`.

5.1 Factory method

Pattern creazionale, permette la creazione di oggetti tramite chiamata a metodo.

Di solito:

```
Foo foo = new Foo();
```

Factory statica:

```
public class FooFactory {
   public static Foo make();
}
```

5.1 Factory method

Pattern creazionale, permette la creazione di oggetti tramite chiamata a metodo.

Di solito:

```
Foo foo = FooFactory.make();
```

5.1 Factory method

Pattern creazionale, permette la creazione di oggetti tramite chiamata a metodo.

Di solito:

```
Foo foo = FooFactory.make();
```

oppure

```
Foo foo = Foo.makeDefault();
Foo foo = Foo.makeWithGusto();
```

5.1 Factory method

Impedire allocazione via costruttore.

```
public class Foo {
   Foo() { }
}

public class FooFactory {
   private FooFactory() { }
   public static Foo make();
}
```

5.1 Factory method in Android

```
Fragment.instantiate(Context, String);

AndroidHttpClient.newInstance(String uri);

Toast.makeText(Context, String);

…
```

5.1 Factory method

Usato spesso: permette al framework di rigenerare le classi (`onResume`) utilizzando il costruttore pubblico.

Il costruttore **non** va utilizzato per l'inizializzazione delle classi!

Utilizzare gli eventi: `onCreate`, `onResume`...

5.2

Navigazione ed Intent

5.2 Demo

Intent espliciti

Intent con dati extra

Intent impliciti

Activity con risultato

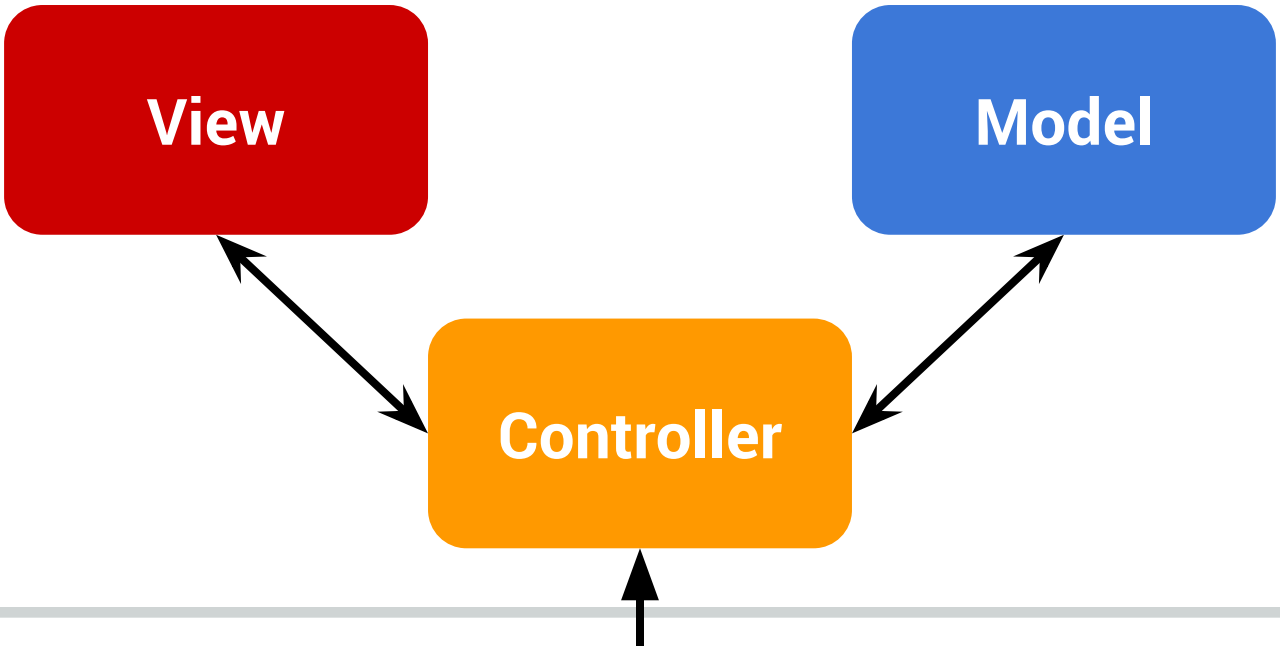
Intermezzo 2

MVC pattern

5.2 Architettura MVC

Pattern *Model View Controller* per UI:

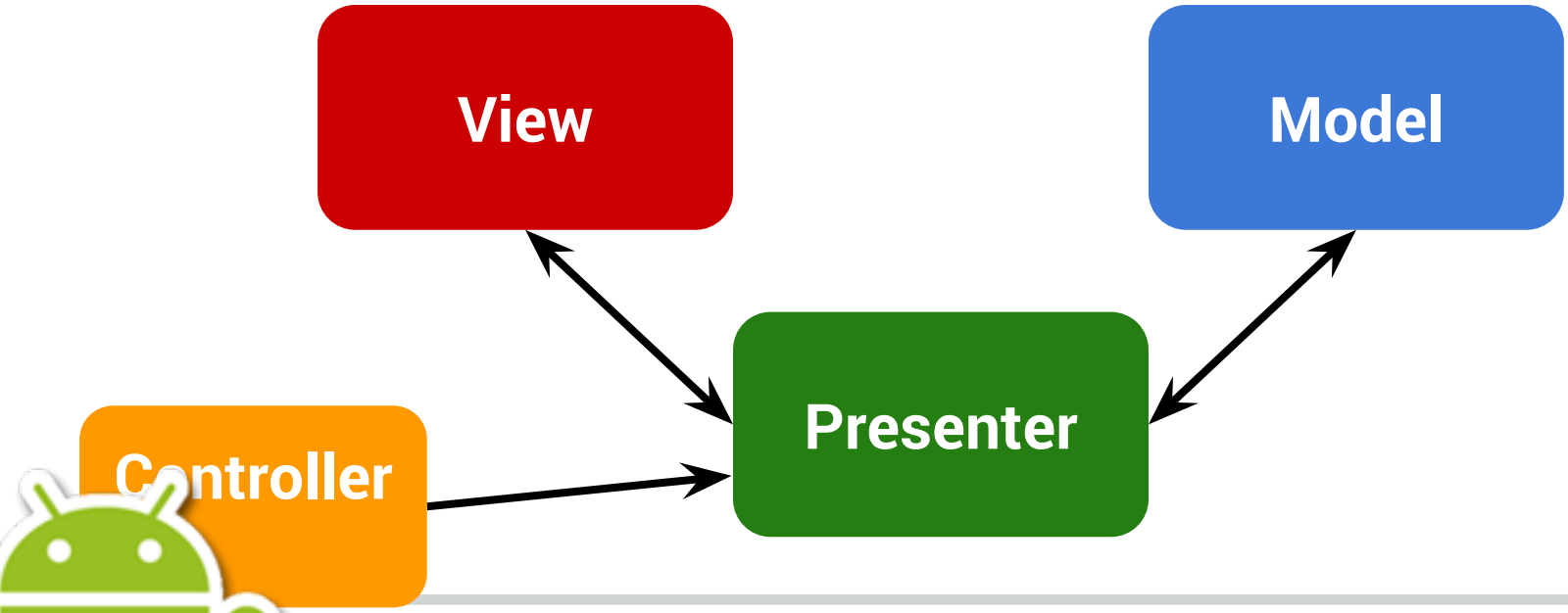
- Utente accede a Controller.
- Presenta View senza legami a Model.
- Controller *media* tra Model e View. (Ma come?)



5.2 Model View Presenter

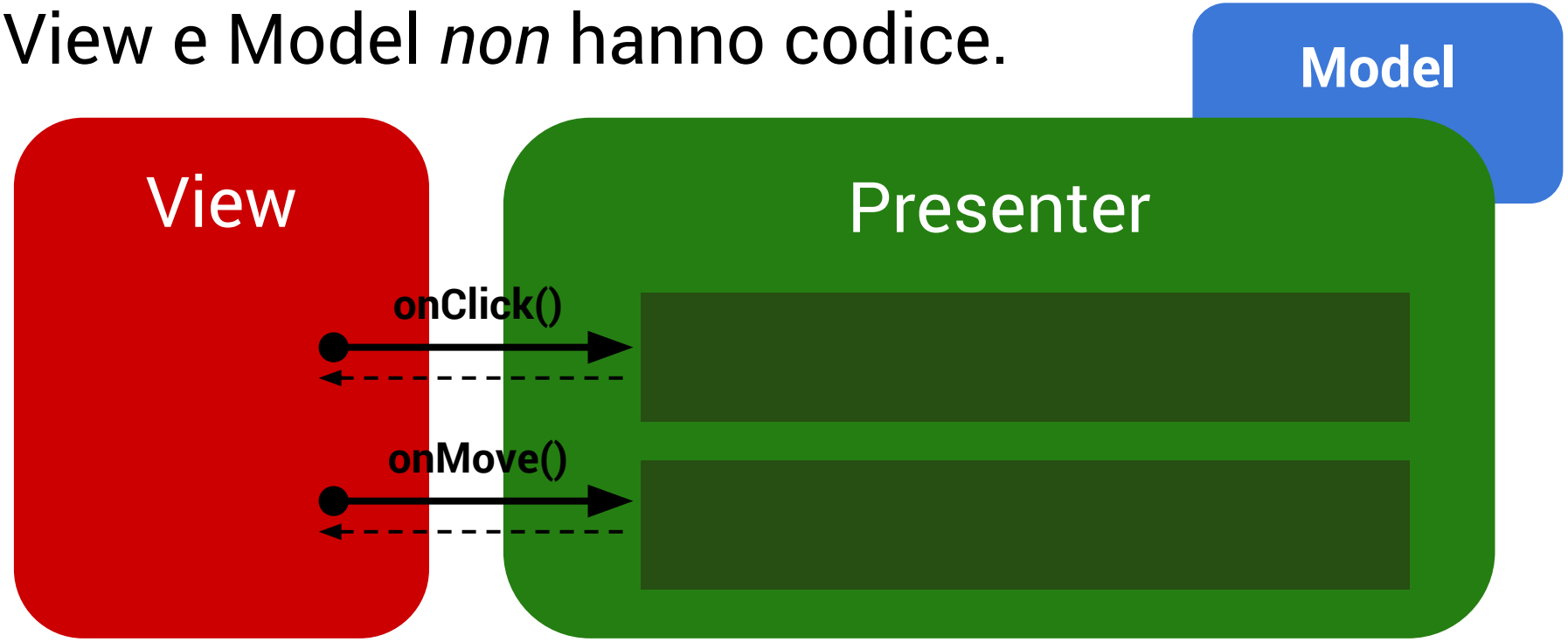
Presenter agisce come *"code behind"* nel gestire eventi della View.

Mediatore attivo.



5.2 MVP: passive view

View e Model *non* hanno codice.



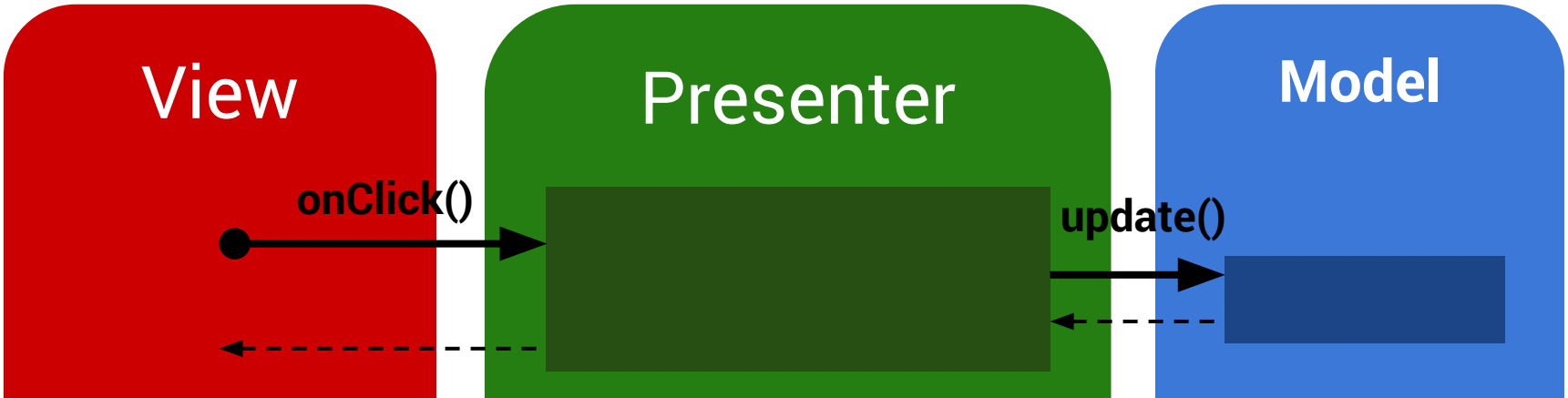
Eventi gestiti da Presenter, che aggiorna la View. No *data binding*. Stato nel Presenter.

5.2 Supervising presenter

Presenter gestisce la sincronizzazione, ma parte della responsabilità passa a View/Model.

Primitive di *data binding*:

- View di Android si limitano a stile.
- Esistono progetti che lo estendono (XML).
- Model che implementa `Observable`.

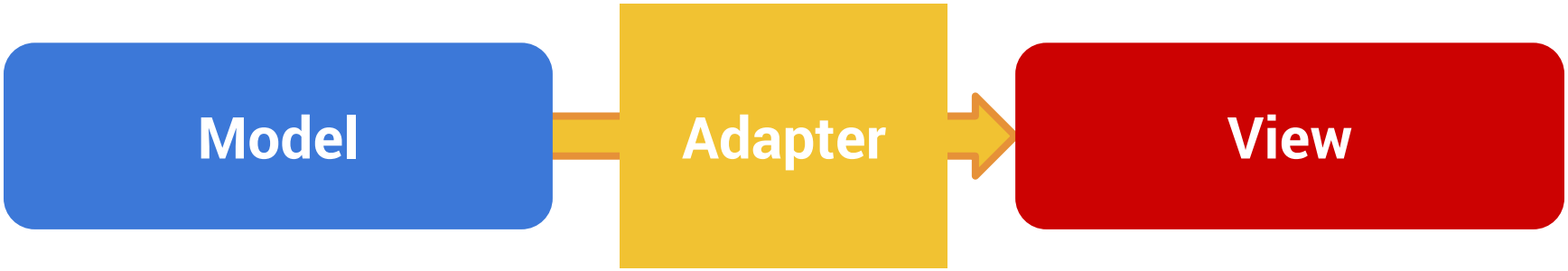


5.2 Adapter

Classe che traduce un'interfaccia in un'altra.

Necessariamente parte del Presenter, riduce la purezza raggiungibile in codice UI.

In Android: tramite (di codice) tra dati e UI.



5.3

Notifiche

5.3 Demo

Notifiche base con PendingIntent

5.4

Broadcast Receiver

5.4 Broadcast Receiver

Componente Android che permette di registrarsi per eventi di sistema o di livello applicazione.

Per esempio potremmo registrarci per l'evento ACTION_BOOT_COMPLETED lanciato dal sistema appena la procedura di avvio è completata.

5.4 Broadcast Receiver

Per crearne uno dobbiamo estendere la classe astratta BroadcastReceiver e fare l'override del metodo onReceive(Context c, Intent intent).

```
@Override
public void onReceive(Context c, Intent intent){
        // do something
        ...
}
```

Quando l'evento per cui ci siamo registrati accade, il metodo onReceiver viene invocato con i parametri corretti.

5.4 Broadcast Receiver

Il metodo onReceive deve essere molto "leggero" e non può effettuare operazioni asincrone.

Se vogliamo comportamenti più elaborati possiamo avviare un servizio dal suo interno e spostare lì la logica.

5.4 Broadcast Receiver

Esistono due tecniche per registrare un Broadcast Receiver:

- staticamente tramite il manifest;
- dinamicamente tramite Java.

5.4 Broadcast Receiver

Registrazione statica nel manifest

```
<!-- ... -->
<receiver android:name=".MyBReceiver" >
        <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
</receiver>
<!-- ... -->
```

5.4 Broadcast Receiver

Registrazione dinamica in Java

```
IntentFilter iF = new IntentFilter(Intent.ACTION_BOOT_COMPLETED);
MyBReceiver br = new MyBReceiver();

// registrazione
this.registerReceiver(br, iF);

// deregistrazione
this.unregisterReceiver(br);
```

5.4 Broadcast Receiver

Alcuni tra i più importanti eventi di sistema per cui possiamo registrarci:

- Intent.ACTION_BOOT_COMPLETED
- Intent.ACTION_POWER_CONNECTED
- Intent.ACTION_POWER_DISCONNECTED
- Intent.ACTION_SHUTDOWN
- Intent.ACTION_DEVICE_STORAGE_LOW
- Intent.ACTION_DEVICE_STORAGE_OK
- Intent.ACTION_BATTERY_LOW
- Intent.ACTION_BATTERY_OKAY
- ConnectivityManager.CONNECTIVITY_ACTION
- AudioManager.AUDIO_BECOMING_NOISY

5.4 Broadcast Receiver

Esistono due tipi di intent broadcast:

- Normal Broadcast
  - inviato con *sendBroadcast()*;
  - asincrono
  - ordine di ricezione non specificato
- Ordered Broadcast
  - inviato con sendOrderedBroadcast();
  - sincrono, un receiver alla volta
  - ordine dipende dalla priorità specificata nel filtro
  - un receiver può bloccare l'inoltro dell'intent

5.4 Broadcast Receiver

Se vogliamo scambiare dati in maniera privata

all'app esiste la classe *LocalBroadcastManager:*

- gli intent broadcast rimangono nell'app
- altre app non possono inviare intent broadcast per interferire con la nostra app
- più efficiente di intent globali

5.5

Servizi

5.5 Servizi

Componente Android che esegue in background, senza interazione diretta con l'utente.

Android esegue alcuni servizi predefiniti per le proprie attività, ma ogni applicazione può implementare i propri servizi specifici.

5.5 Servizi

Per poter essere avviato un servizio deve essere dichiarato nel manifest, nella seguente maniera:

```
<service
      android:name="MyService">
</service>
```

5.5 Servizi

Per poter essere avviato un servizio deve essere dichiarato nel manifest, nella seguente maniera:

```
<service
      android:name="MyService"
      android:process=":processName">
</service>
```

Attributo *process* per specificare l'esecuzione in un processo separato.

5.5 Servizi

Necessario estendere la classe base Service...

```
public class MyService extends Service{

// ..

}
```

5.5 Servizi

..e fare l'override dei metodi per gestire il ciclo di vita

```
public class MyService extends Service{

    @Override
    public void onCreate(){
        super.onCreate();
        // il servizio è stato appena creato.
    }

}
```

5.5 Servizi

..e fare l'override dei metodi per gestire il ciclo di vita

```
public class MyService extends Service{
	@Override
	public int  onStartCommand(Intent intent, int flags, int startId){
		super. onStartCommand(intent, flags, startId);
		// al servizio è stato appena chiesto di reagire
		// alla richiesta contenuta nell'intent.
		return START_NOT_STICKY; // vediamo meglio fra poco
	}
}
```

5.5 Servizi

..e fare l'override dei metodi per gestire il ciclo di vita

```
public class MyService extends Service{

    @Override
    public int  onDestroy(){
        super. onDestroy();
        // il servizio è stato appena distrutto dal sistema
        // quindi dovremmo liberare eventuali risorse.
    }
}
```

5.5 Servizi

Flag di ritorno nell'onStartCommand:

START_NOT_STICKY	Se il servizio viene ucciso mentre è avviato (dopo l'onStartCommand), non riavviarlo fino a che non è presente una nuova richiesta esplicita
START_REDELIVERY_INTENT	Se il servizio viene ucciso mentre è avviato (dopo l'onStartCommand), schedula un riavvio fornendogli l'ultimo intent che gli è stato passato nell'onStartCommand
START_STICKY	Se il servizio viene ucciso mentre è avviato (dopo l'onStartCommand), schedula un riavvio fornendogli un intent nullo
START_STICKY_COMPATIBILITY	Versione di START_STICKY per la compatibilità con le versioni meno recenti di Android (< 2.0)

5.5 Servizi

Avviare un servizio:

```
// creiamo l'intent esplicito per l'avvio del servizio desiderato.
Intent service = new Intent(context, MyService.class);
// inseriamo eventuali extra per dettagliare le operazioni richieste al servizio.
service.putExtra("DOWNLOAD_URI", "http://www...");
// avviamo il servizio che soddisfa l'intent appena creato.
context.startService(service);
```

5.5 Servizi

Fermare un servizio dall'esterno:

```
// creiamo l'intent esplicito per l'arresto del servizio desiderato.
Intent service = new Intent(context, MyService.class);
// chiediamo ad Android l'arresto del servizio specificato.
context.stopService(service);
```

NB: non è rilevante il numero di avvi di un servizio. Android arresterà il servizio immediatamente.

5.5 Servizi

Fermare un servizio dall'interno:

```
// un oggetto Service può indicare autonomamente la sua terminazione.
stopSelf();
```

NB: non è rilevante il numero di avvi di un servizio. Android arresterà il servizio immediatamente.

5.5 Servizi

Avviare un servizio in **foreground**:

```
// un oggetto Service può indicare di essere di notevole importanza per l'UX,
// quindi può attivarsi in FOREGROUND fornendo la notifica
// che verrà mostrata all'utente durante la sua esecuzione
Notification not = new Notification();
// avviamo il servizio in foreground.
startForeground(1, not);
```

5.5 Servizi

Comunicazione Activity-Service:

- Intent e BroadcastReceiver
  - massimo disaccoppiamento
  - classico schema di comunicazione Android
- Binding in memoria
  - forte accoppiamento
  - activity e servizio devono risiedere nello stesso processo
  - maggiore semplicità e velocità di esecuzione

5.5 Servizi

Binding in memoria, da parte del servizio...

```
public class MyService extends Service{
    // creiamo un oggetto che fa da collante tra servizio ed activity.
    private final IBinder myBinder = new MyBinder();

    public class MyBinder extends Binder{
        public MyService getService(){
            return MyService.this;
        }
    }
...
```

5.5 Servizi

Binding in memoria, da parte del servizio...

```
public class MyService extends Service{
        ...

        // e lo restituiamo quando un componente si collega a questo servizio.
        public IBinder onBind(Intent intent){
                return myBinder;
        }

}
```

5.5 Servizi

Binding in memoria, da parte dell'activity...

```
public class MyActivity extends Activity{
        private MyService myService;
        public void onStart(){
                super.onStart();
                // richiediamo il collegamento in memoria.
                Intent svcInt = new Intent(this, MyService.class);
                bindService(svcInt, myConnection, Context.BIND_AUTO_CREATE);
        }
        ...
}
```

5.5 Servizi

Binding in memoria, da parte dell'activity...

```
public class MyActivity extends Activity{
        ...
        public void onStop(){
                super.onStop();
                // rimuoviamo il collegamento in memoria.
                unbindService(myConnection);
        }
        ...
}
```

5.5 Servizi

Binding in memoria, da parte dell'activity...

```
public class MyActivity extends Activity{
        ...
        private  ServiceConnection myConnection = new ServiceConnection(){
                public void onServiceConnected(ComponentName name, IBinder s){
                        MyService.MyBinder binder = (MyService.MyBinder) s;
                        myService = binder.getService();
                }
                ...
        }
}
```

5.5 Servizi

Binding in memoria, da parte dell'activity...

```
public class MyActivity extends Activity{
        ...
        private  ServiceConnection myConnection = new ServiceConnection(){
                ...
                public void onServiceDisconnected(ComponentName name){
                        myService = null;
                }
        }
}
```

5.6

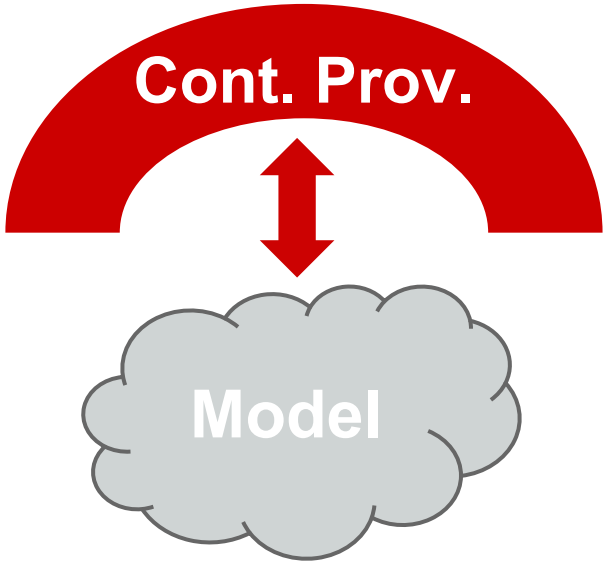
Content Provider Usage

5.6 Content Provider (riassunto)

È un'astrazione di insiemi di **dati strutturati**.

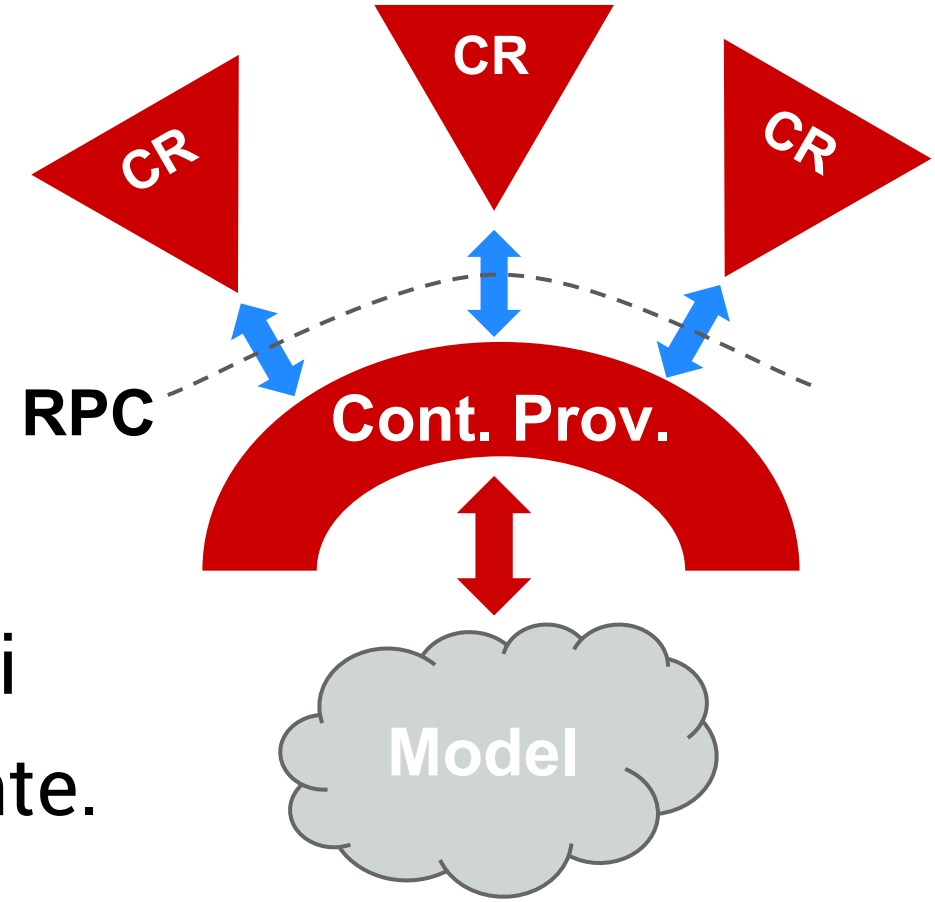
Gestisce l'accesso ai dati.

Garantisce la sicurezza
delle operazioni.



5.6 Content Provider (riassunto)

Un'applicazione non può accedere ai dati di un'altra.



Usato per **condividere dati** ad altre applicazioni o per gestirli privatamente.

5.6 Content Provider

Viene individuato da un Uri

`content://user_dictionary/words`

scheme

Uguale per tutti i content provider

5.6 Content Provider

Viene individuato da un Uri

`content://user_dictionary/words`

authority

Individua il Content Provider

Simile ad un nome di dominio o all'indirizzo IP di un DB.

5.6 Content Provider

Viene individuato da un Uri

```
content://user_dictionary/words
```

path

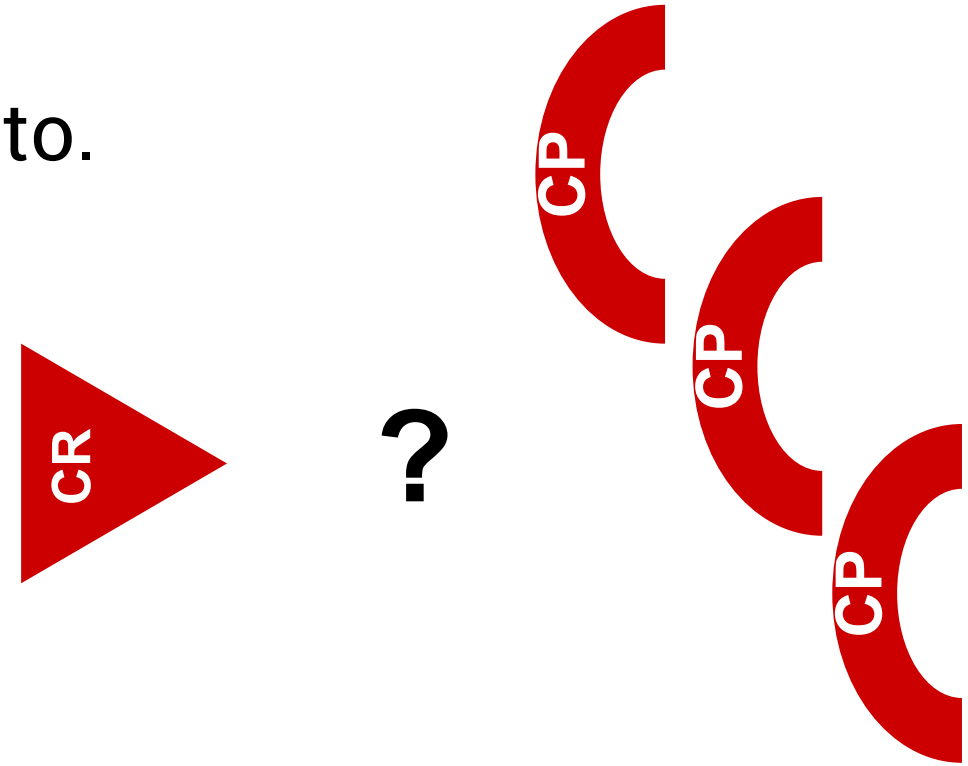
Individua la tabella

Simile al nome di una directory, ecc.

5.6 Content Resolver

Ogni applicazione accede ai CP tramite un Content Resolver.

Individua il CP corretto.
(tramite l'Uri)



5.6 Content Resolver

Ogni componente può ottenere un Content Resolver tramite il contesto

```
ContentResolver resolver = getApplicationContext().getContentResolver();
```

.

Espone metodi **CRUD**:
create, retrieve, update, delete.

5.6.1 Content Resolver - Create

```
ContentValues mNewValues = new ContentValues();
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "frak");
mNewUri = getContentResolver().insert(
   UserDictionary.Word.CONTENT_URI,   // the user dictionary content URI
   mNewValues                         // the values to insert
);
```

5.6.1 Content Resolver - Create

```
ContentValues mNewValues = new ContentValues();
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "frak");
mNewUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI,   // the user dictionary content URI
    mNewValues                         // the values to insert
);
```

content://user_dictionary/words

5.6.1 Content Resolver - Create

```
ContentValues mNewValues = new ContentValues();
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "frak");
mNewUri = getContentResolver().insert(
  UserDictionary.Word.CONTENT_URI,   // the user dictionary content URI
  mNewValues                         // the values to insert
);
```

```
INSERT INTO Word (APP_ID, LOCALE, WORD)
        VALUES ('example.user', 'en_US', 'frak');
```

5.6.1 Content Resolver - Create

```
ContentValues mNewValues = new ContentValues();
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "frak");
mNewUri = getContentResolver().insert(
   UserDictionary.Word.CONTENT_URI,   // the user dictionary content URI
   mNewValues                         // the values to insert
);
```

content://user_dictionary/words/377

5.6.2 Content Resolver - Update

```
mUpdateValues.put(UserDictionary.Words.LOCALE, "it_IT");
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};

mRowsUpdated = getContentResolver().update(
   UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
   mUpdateValues                       // the columns to update
   mSelectionClause                    // the column to select on
   mSelectionArgs                      // the value to compare to
);
```

5.6.2 Content Resolver - Update

```
mUpdateValues.put(UserDictionary.Words.LOCALE, "it_IT");
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};

mRowsUpdated = getContentResolver().update(
   UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
   mUpdateValues                       // the columns to update
   mSelectionClause                    // the column to select on
   mSelectionArgs                      // the value to compare to
);
```

5.6.2 Content Resolver - Update

```
mUpdateValues.put(UserDictionary.Words.LOCALE, "it_IT");
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};

mRowsUpdated = getContentResolver().update(
   UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
   mUpdateValues                       // the columns to update
   mSelectionClause                    // the column to select on
   mSelectionArgs                      // the value to compare to
);
```

5.6.2 Content Resolver - Update

```
mUpdateValues.put(UserDictionary.Words.LOCALE, "it_IT");
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};

mRowsUpdated = getContentResolver().update(
   UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
   mUpdateValues                       // the columns to update
   mSelectionClause                    // the column to select on
   mSelectionArgs                      // the value to compare to
);
```

5.6.2 Content Resolver - Update

```
mUpdateValues.put(UserDictionary.Words.LOCALE, "it_IT");
String mSelectionClause = UserDictionary.Words.LOCALE +  "LIKE ?";
String[] mSelectionArgs = {"en_%"};
mRowsUpdated = getContentResolver().update(
   UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
   mUpdateValues                       // the columns to update
   mSelectionClause                    // the column to select on
   mSelectionArgs                      // the value to compare to
);
```

```
UPDATE Word
            SET LOCALE = 'it_IT'
            WHERE LOCALE LIKE 'en_%'
```

5.6.3 Content Resolver - Delete

```
String mSelectionClause = UserDictionary.Words.APP_ID + " = ?";
String[] mSelectionArgs = {"user"};

mRowsDeleted = getContentResolver().delete(
   UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
   mSelectionClause                    // the column to select on
   mSelectionArgs                      // the value to compare to
);
```

5.6.3 Content Resolver - Delete

```
String mSelectionClause = UserDictionary.Words.APP_ID + " = ?";
String[] mSelectionArgs = {"user"};

mRowsDeleted = getContentResolver().delete(
   UserDictionary.Words.CONTENT_URI,   // the user dictionary content URI
   mSelectionClause                    // the column to select on
   mSelectionArgs                      // the value to compare to
);
```

```
DELETE FROM Words
            WHERE APP_ID = 'user';
```

5.6.4 Content Resolver - Retrieve

```
String[] projection = {ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME,
        ContactsContract.Contacts.HAS_PHONE_NUMBER,
        ContactsContract.Contacts.PHOTO_THUMBNAIL_URI};
String selection = ContactsContract.Contacts.HAS_PHONE_NUMBER + " = '1'"
         + " AND " + ContactsContract.Contacts.PHOTO_ID + " NOT NULL ";
String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " DESC" +
        "LIMIT 100";
Cursor resultsCursor = rsvlr.query(ContactsContract.Contacts.CONTENT_URI,
projection, selection, null, sortOrder);
```

5.6.4 Content Resolver - Retrieve

```
String[] projection = {ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME,
        ContactsContract.Contacts.HAS_PHONE_NUMBER,
        ContactsContract.Contacts.PHOTO_THUMBNAIL_URI};
String selection = ContactsContract.Contacts.HAS_PHONE_NUMBER + " = '1'"
         + " AND " + ContactsContract.Contacts.PHOTO_ID + " NOT NULL ";
String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " DESC" +
        "LIMIT 100";
Cursor resultsCursor = rsvlr.query(ContactsContract.Contacts.CONTENT_URI,
projection, selection, null, sortOrder);
```

5.6.4 Content Resolver - Retrieve

```
String[] projection = {ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME,
        ContactsContract.Contacts.HAS_PHONE_NUMBER,
        ContactsContract.Contacts.PHOTO_THUMBNAIL_URI};
String selection = ContactsContract.Contacts.HAS_PHONE_NUMBER + " = '1'"
        + " AND " + ContactsContract.Contacts.PHOTO_ID + " NOT NULL ";
String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " DESC" +
        "LIMIT 100";
Cursor resultsCursor = rsvlr.query(ContactsContract.Contacts.CONTENT_URI,
projection, selection, null, sortOrder);
```

5.6.4 Content Resolver - Retrieve

```
String[] projection = {ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME,
        ContactsContract.Contacts.HAS_PHONE_NUMBER,
        ContactsContract.Contacts.PHOTO_THUMBNAIL_URI};
String selection = ContactsContract.Contacts.HAS_PHONE_NUMBER + " = '1'"
        + " AND " + ContactsContract.Contacts.PHOTO_ID + " NOT NULL ";
String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " DESC" +
        "LIMIT 100";
Cursor resultsCursor = rsvlr.query(ContactsContract.Contacts.CONTENT_URI,
projection, selection, null, sortOrder);
```

5.6.4 Content Resolver - Retrieve

```
String[] projection = {ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME,
        ContactsContract.Contacts.HAS_PHONE_NUMBER,
        ContactsContract.Contacts.PHOTO_THUMBNAIL_URI};
String selection = ContactsContract.Contacts.HAS_PHONE_NUMBER + " = '1'" + " AND " +
        ContactsContract.Contacts.PHOTO_ID + " NOT NULL ";
String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " DESC" +  "LIMIT 100";
Cursor resultsCursor = rsvlr.query(ContactsContract.Contacts.CONTENT_URI,
        projection, selection, null, sortOrder);
```

```
SELECT _ID, DISPLAY_NAME, ...  FROM Contacts
    WHERE HAS_PHONE_NUMBER = '1'
        AND PHOTO_ID NOT NULL
        ORDER BY DISPLAY_NAME DESC LIMIT 100
```

5.6.5 Content Resolver - Retrieve

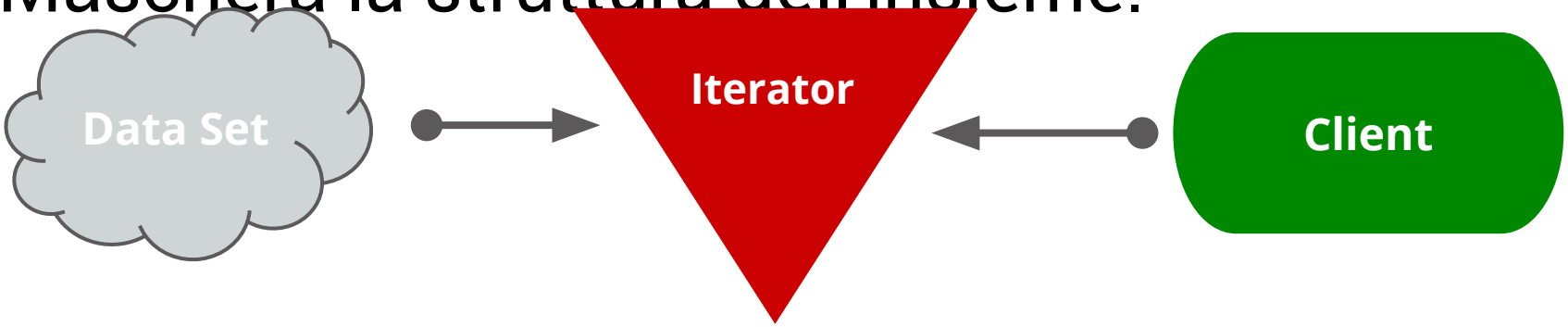
```
String[] projection = {ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME,
        ContactsContract.Contacts.HAS_PHONE_NUMBER,
        ContactsContract.Contacts.PHOTO_THUMBNAIL_URI};
String selection = ContactsContract.Contacts.HAS_PHONE_NUMBER + " = '1'"
         + " AND " + ContactsContract.Contacts.PHOTO_ID + " NOT NULL ";
String sortOrder = ContactsContract.Contacts.DISPLAY_NAME + " DESC" +
        "LIMIT 100";
Cursor resultsCursor = rsvlr.query(ContactsContract.Contacts.CONTENT_URI,
projection, selection, null, sortOrder);
```

5.6.5 Cursor (Iterator)

Caso particolare del pattern *Iterator* (GoF)

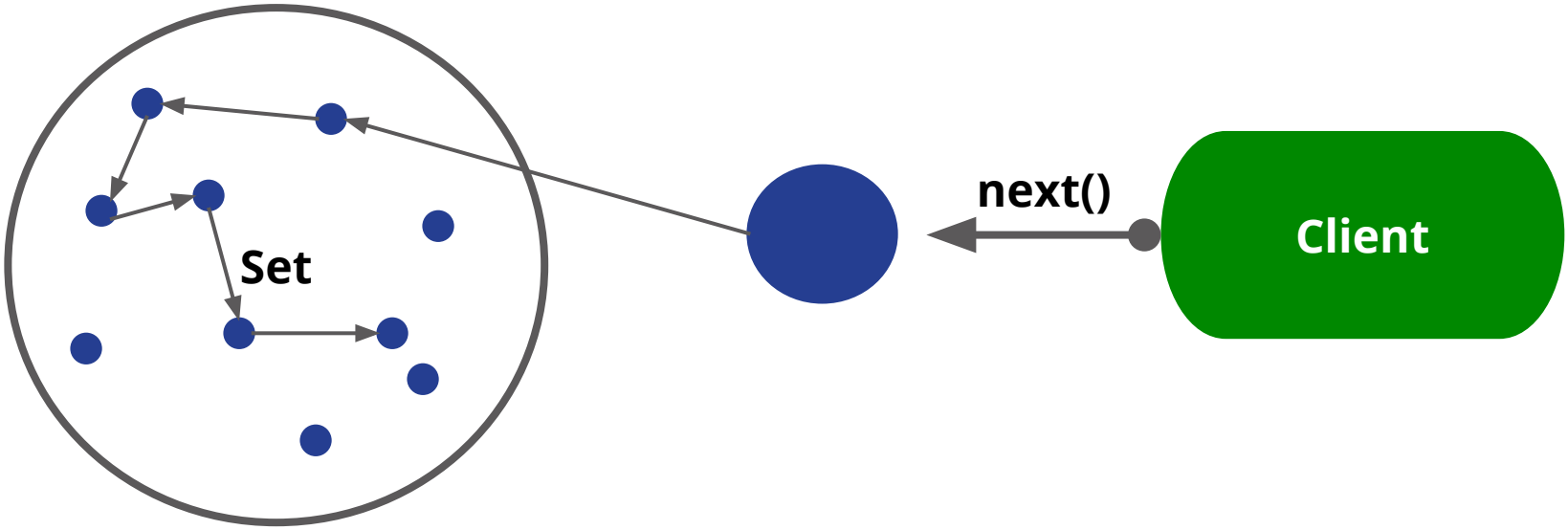
Fornisce un accesso sequenziale ad un insieme di oggetti.

Maschera la struttura dell'insieme.



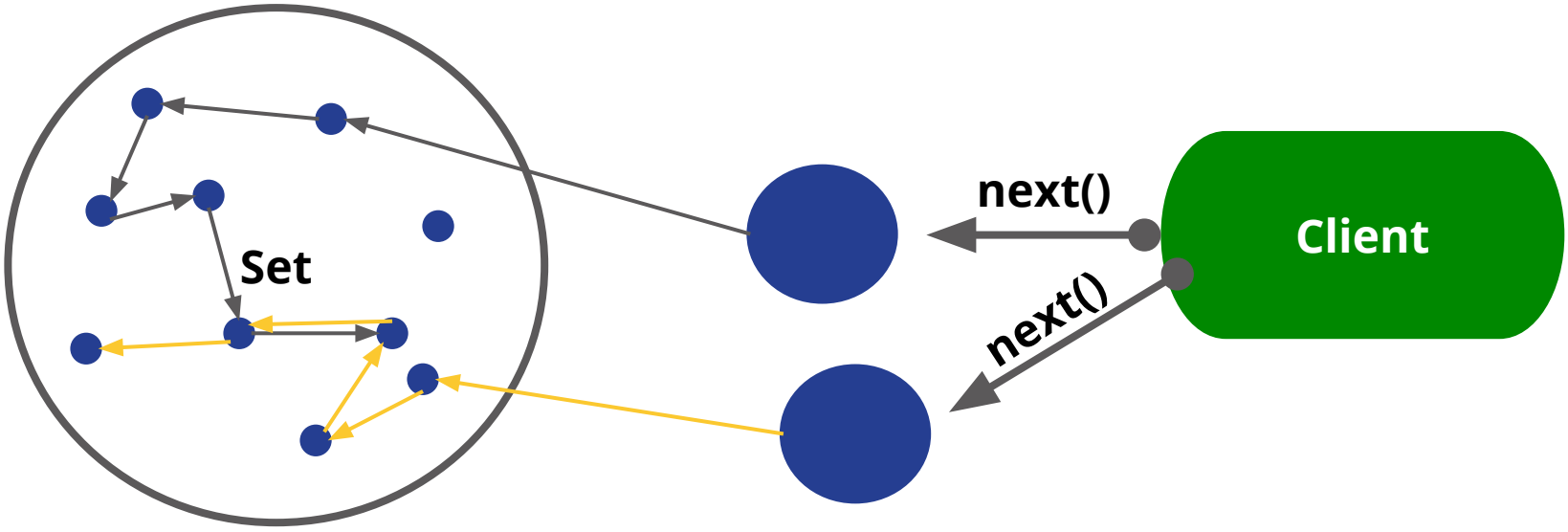
5.6.5 Cursor (Iterator)

Un puntatore all'elemento corrente nell'insieme.



5.6.5 Cursor (Iterator)

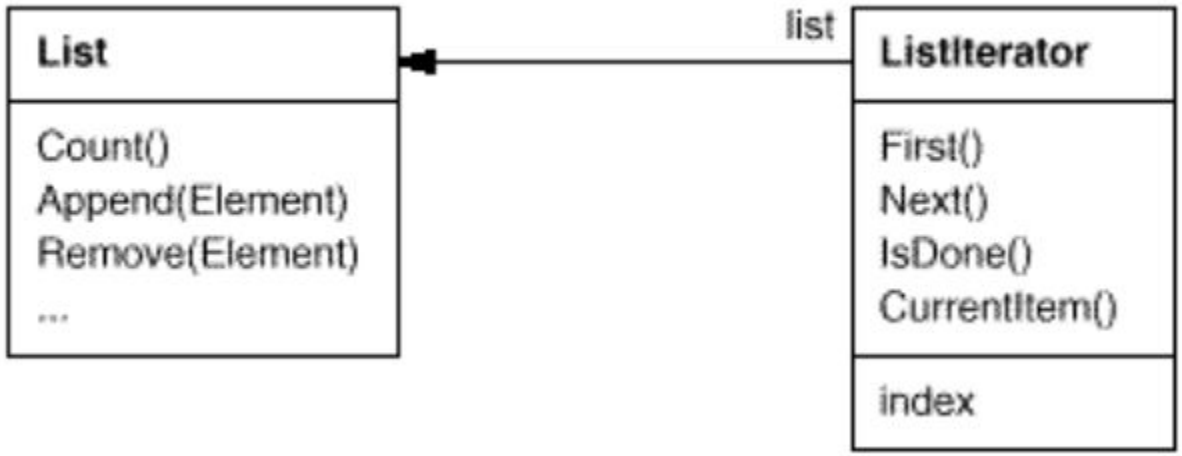
Permette diversi tipi di traversamento.



5.6.5 Cursor (Iterator)

Iteratore esterno: `inititalize(), currentItem(), moveNext(), hasNext(), ...`

Iteratore interno: `inititalize(), next().`



5.7

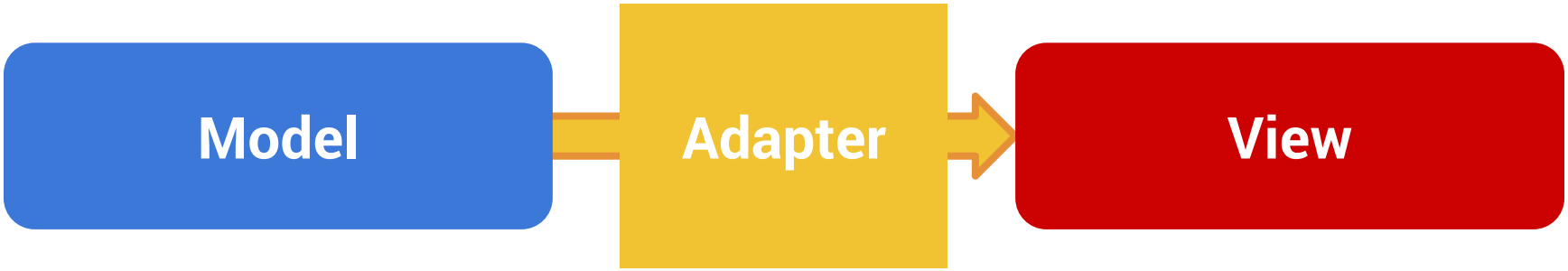
Adapter

5.7 Adapter

Classe che traduce un'interfaccia in un'altra.

- Necessariamente parte del Presenter, riduce la purezza raggiungibile in codice UI.

In Android: tramite (di codice) tra dati e UI.



5.8

AlarmManager

5.8 AlarmManager

È un servizio di sistema che permette di schedulare l'invio di Intent a certe scadenze e intervalli.

Viene utilizzato per limitare il tempo di vita dell'applicazione, in modo da consumare meno risorse energetiche e computazionali.

5.8 AlarmManager

Per esempio potrebbe essere utilizzato per controllare periodicamente la disponibilità di aggiornamenti dalla rete:

```
public class MyActivity extends Activity{
        ...
        // creiamo il PendingIntent da passare all'AlarmManager
        Intent intent = new Intent(this, MyService.class);
        PendingIntent pIntent = PendingIntent.getService(this, 0, intent, 0);
        ...
}
```

5.8 AlarmManager

Per esempio potrebbe essere utilizzato per controllare periodicamente la disponibilità di aggiornamenti dalla rete:

```
public class MyActivity extends Activity{
...
// quindi scheduliamo l'invio del PendingIntent
AlarmManager a = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
long now = System.currentTimeMillis();

a.setRepeating(AlarmManager.RTC_WAKEUP, now , 60 * 60 * 1000, pIntent);

}
```

5.8 AlarmManager

Per evitare di avere tanti eventi vicini tra di loro nel tempo ma non completamente coincidenti, Android permette di specificare intervalli ***inesatti***:

- INTERVAL_FIFTEEN_MINUTES
- INTERVAL_HALF_HOUR
- INTERVAL_HOUR
- INTERVAL_HALF_DAY
- INTERVAL_DAY

5.8 AlarmManager

Per evitare di avere tanti eventi vicini tra di loro nel tempo ma non completamente coincidenti, Android permette di specificare intervalli ***inesatti***:

```
public class MyActivity extends Activity{

AlarmManager a = (AlarmManager) getSystemService(Context.ALARM_SERVICE);
long now = System.currentTimeMillis();

a.setInexactRepeating(AlarmManager.RTC_WAKEUP, now,

                    AlarmManager.INTERVAL_HOUR, pIntent);

}
```

5.9

DownloadManager

5.9 DownloadManager

Servizio di sistema per la gestione dei long-running HTTP download.

```
DownloadManager dm = (DownloadManager)getSystemService
(DOWNLOAD_SERVICE);
```

Espone un'API semplificata per il recupero di dati su server remoti.

```
Uri uri = Uri.parse("http://google.it/wholegooglearchive.zip");
DownloadManager.Request dr = new DownloadManager.Request(uri);
dm.enqueue(dr);
```

5.9 DownloadManager

Gestisce la visualizzazione dello stato del download tramite notifiche.

```
DownloadManager.Request dr = new DownloadManager.Request(uri);
dr.setTitle("The whole Google archive's");
dr.setDescription("we can do it!");
dr.setNotificationVisibility(DownloadManager.Request.VISIBILITY_VISIBLE);
dm.enqueue(dr);
```

5.9 DownloadManager

Notifica il completamento dei download tramite *intent broadcast*

```
registerReceiver(myBroadcastReceiver,
        new IntentFilter(DownloadManager.ACTION_DOWNLOAD_COMPLETE));


...
public void onReceive(Context context, Intent intent) {
        long downloadId = Intent.getLongExtra(DownloadManager.
EXTRA_DOWNLOAD_ID, -1);
        logStatusOnUI(String.format("Download %d Completed",downloadId));
}
```

5.10

Location API

5.10 Location API

Android mette a disposizione classi e metodi per ottenere la ***posizione geografica attuale***, essere notificati ai ***cambi di posizione***, trovare latitudine e longitudine dato un indirizzo (***forward geocoding***), ricavare un indirizzo dati latitudine e longitudine (***reverse geocoding***).

5.10 Location API

Posizione geografica attuale:

```
// Acquisiamo il servizio di sistema per localizzarsi
LocationManager lm = (LocationManager) getSystemService
                                (Context.LOCATION_SERVICE);
// Definiamo un criterio per avere una buona precisione
Criteria crit = new Criteria();
crit.setAccuracy(Criteria.ACCURACY_FINE);
// Ed otteniamo l'ultima posizione valida dal miglior LocationProvider identificato
String provider = lm.getBestProvider(crit, true);
Location loc = lm.getLastKnownLocation(provider); // loc potrebbe essere null
```

5.10 Location API

Cambi di posizione:

```
// Acquisiamo il servizio di sistema per localizzarsi
LocationManager lm = (LocationManager) getSystemService
                                (Context.LOCATION_SERVICE);
// Definiamo un ascoltatore per i cambiamenti di posizione
LocationListener listener = new LocationListener(){
      public void onLocationChanged(Location location){
            Log.d(TAG, "Nuova posizione rilevata");
      }
      ...
}
```

5.10 Location API

Cambi di posizione:

```
LocationListener listener = new LocationListener(){
        ...
        public void onStatusChanged(String provider, int status, Bundle extras){}
        public void onProviderEnabled(String provider){}
        public void onProviderDisabled(String provider){}
};
// Registriamo l'ascoltatore al LocationManager per ricevere aggiornamenti
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, listener);
```

5.10 Location API

Cambi di posizione:

public void requestLocationUpdates(String ***provider***, long ***minTime***, float ***minDistance***, LocationListener ***listener***);

provider = a chi affidarsi per la gestione della posizione

minTime = l'intervallo minimo di tempo tra due posizioni notificate

minDistance = la distanza minima espressa in metri tra due posizioni notificate

listener = l'oggetto che verrà notificato degli eventi

5.10 Location API

Posizione attuale e cambi di posizione:

un problema per la privacy!

È necessario specificare i permessi nel manifest

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

5.10 Location API

Forward geocoding

```
List<Address> indirizzi = null;
// Dato un indirizzo ottiene latitudine e longitudine relative ad esso
indirizzi = new Geocoder(this).getFromLocationName("Viale Dante, Riccione", 1);
Log.d(TAG, "Viale Dante a latitudine " + indirizzi.get(0).getLatitude() +
                    " longitudine " + indirizzi.get(0).getLongitude());
```

5.10 Location API

Reverse geocoding

```
List<Address> indirizzi = null;
// Dato un indirizzo ottiene latitudine e longitudine relative ad esso
indirizzi = new Geocoder(this).getFromLocation(43.907505, 12.911167, 1);
if(indirizzi.size() > 0){
        Log.d(TAG, "Indirizzo a latitudine" + indirizzi.get(0).getLatitude() +
                        " longitudine " + indirizzi.get(0).getLongitude() + ": " +
                        indirizzi.get(0).getLocality() + ", " +
                        indirizzi.get(0).getCountryName());
}
```