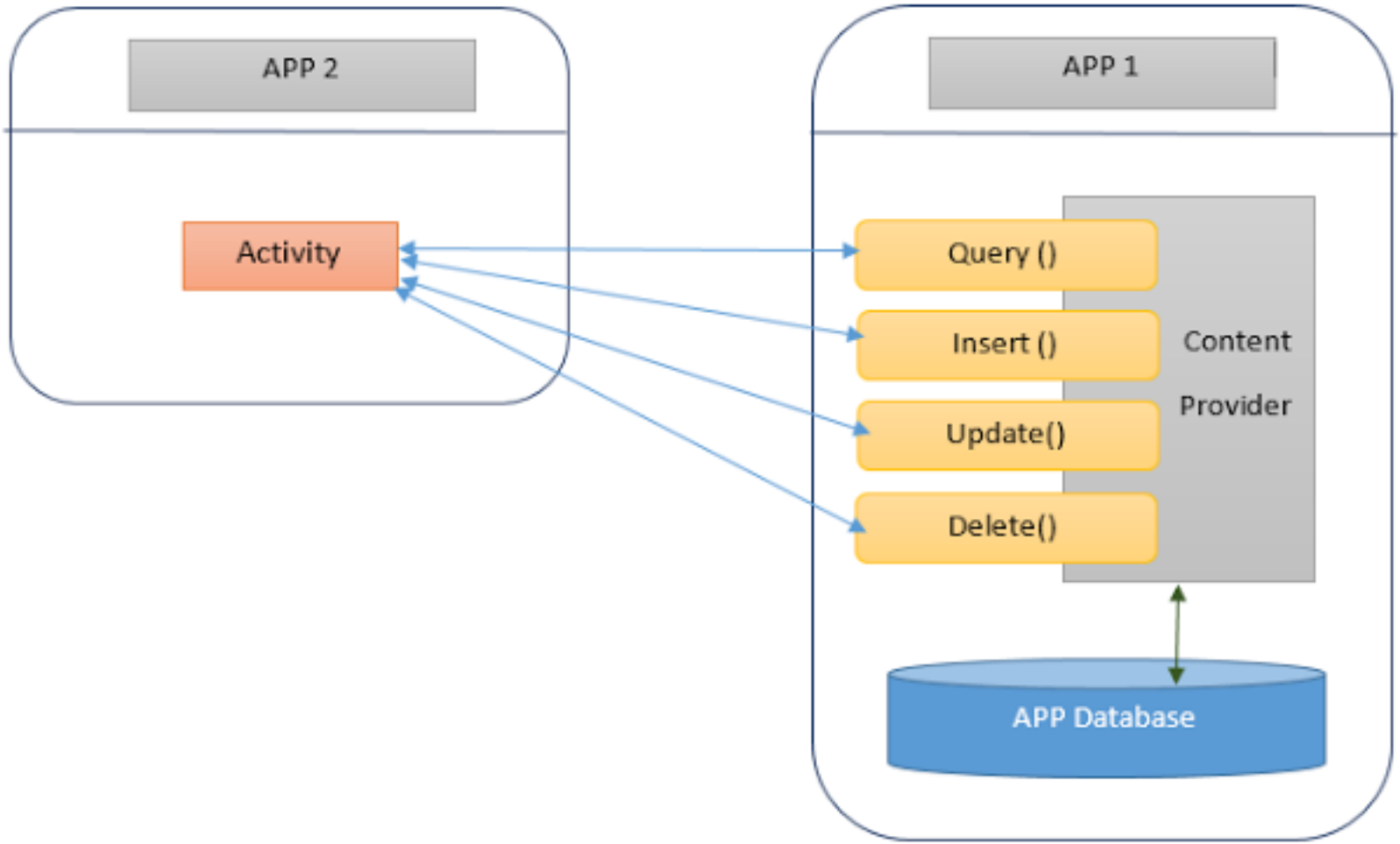
Content Providers

Content Providers

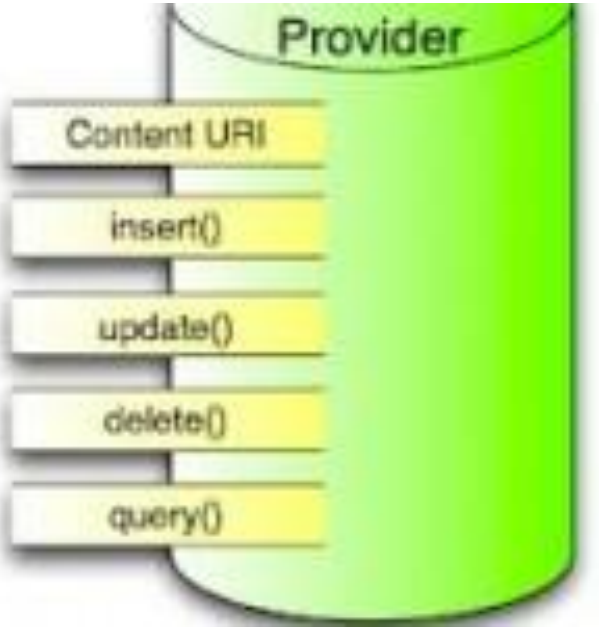


What is a Content Provider?

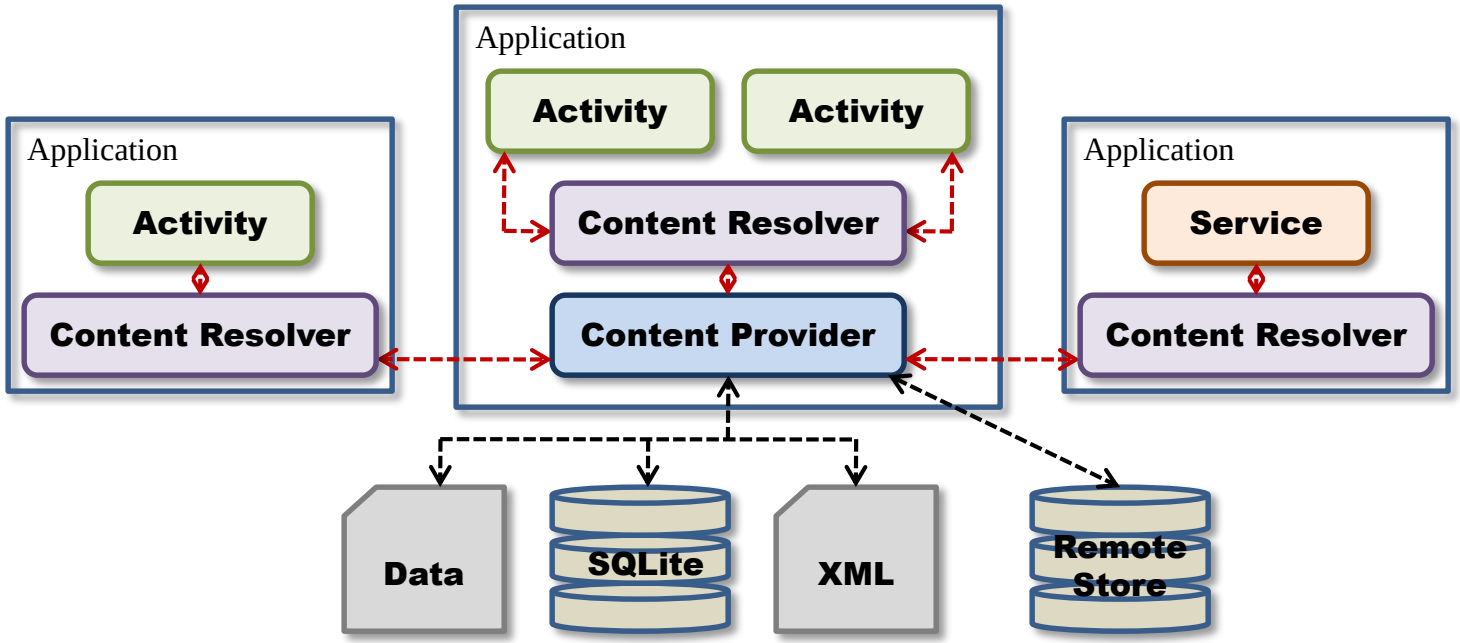
- A content provider provides access to structured data between different Android applications.
- This data is exposed to applications either as tables of data (in much the same way as a SQLite database) or as a handle to a file.
- This essentially involves the implementation of a client /server arrangement whereby the application seeking access to the data is the client and the content provider is the server, performing actions and returning results on behalf of the client.

Content Providers

- Android applications can access data directly from a SQLite database using the database helper but for other applications to get access to the same data you have to create a Content Provider.
- Content Providers encapsulate the data access, provide security using the Android Manifest and standardize the access via a Content URI.
- If the data is used only by your application there is no need to create a content provider but you can always do that in case a need that arises in the future.



Content Providers

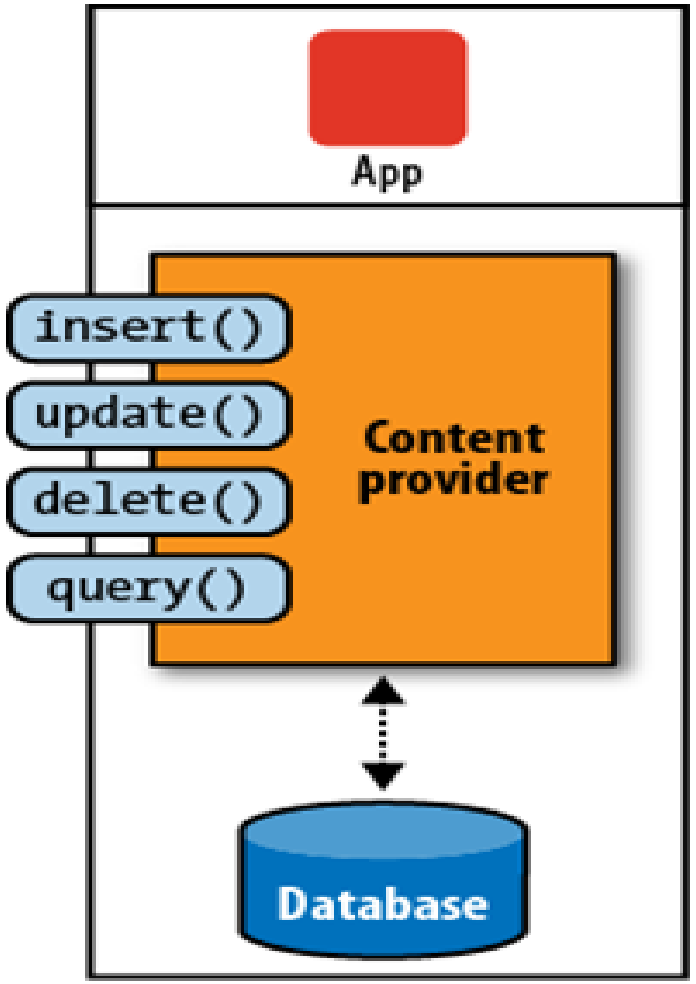


- A content provider makes a specific set of the application's data available to other applications.
  - ✓ The data can be stored in the file system, in an SQLite, or in any other manner that makes sense.

Content Providers (Cont)

- Using a content provider is the only way to share data between Android applications.
- It extends the ContentProvider base class and implements a standard set of methods to allow access to a data store.
  - ✓ Querying
  - ✓ Delete, update, and insert data
- Applications do not call these methods directly.
  - ✓ They use a ContentResolver object and call its methods instead.
  - ✓ A ContentResolver can talk to any content provider.
- Content is represented by URI and MIME type.

What are the operations supported by a Content Provider?



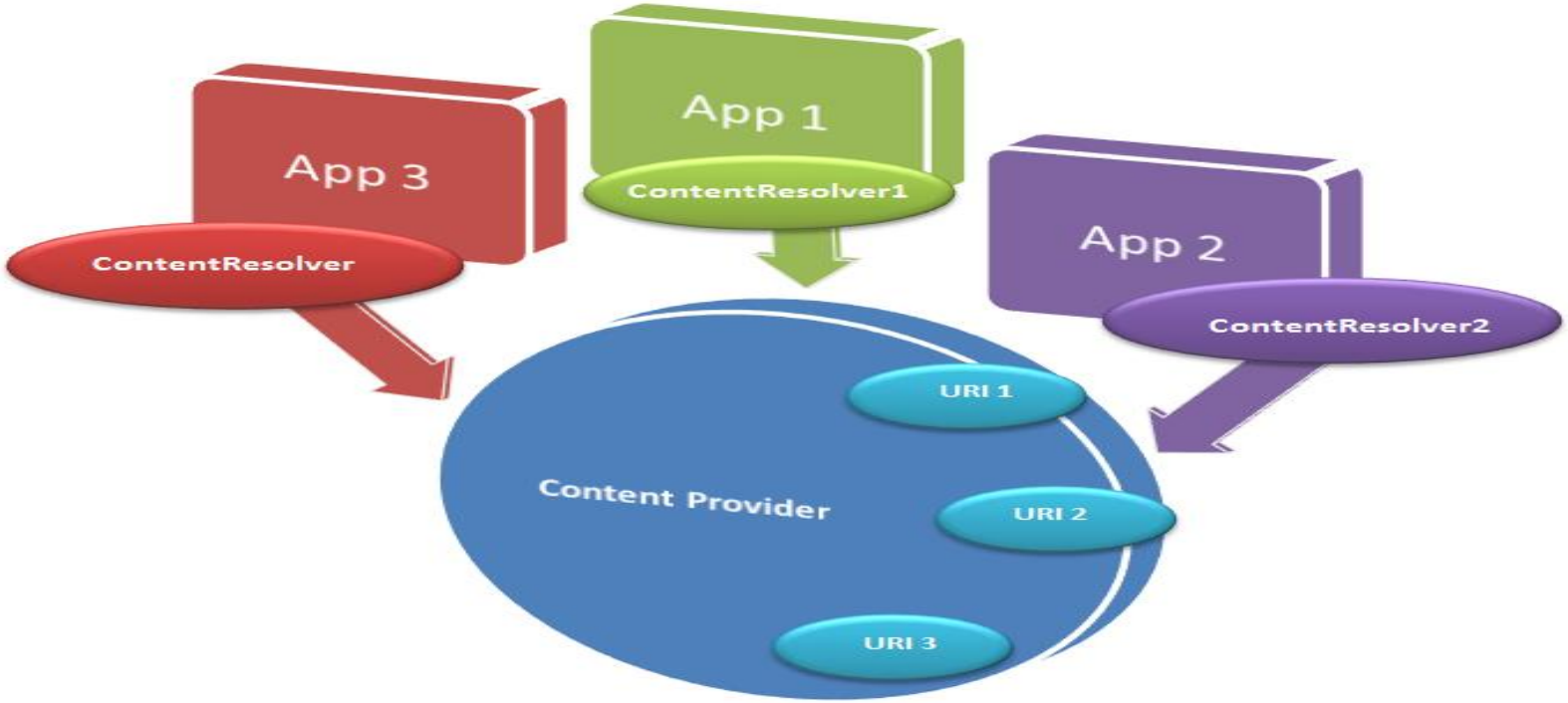
Step 1: Accessing a Content Provider

- **ContentResolver client object** is used to access data from Content Provider. It communicates with provider object, which in turn accepts the request to access data and returns the desired results. The data exchange interface provided by the provider and provider client object allows communication across different processes/applications.
- The application that needs to access the database has to declare this and request permissions in its manifest file.

- **Content URI**
- Content URI is one of the arguments used to identify the data in a provider. It has four parts:
- **1) Scheme:** The scheme for content provider has a constant value: “content”.
- **2) Authority:** It is the symbolic name of the provider, and is unique for each one. This is how we single out the desired content provider from a list of so many.
- **3) Path:** Path **helps distinguish the required data from the complete database**. For instance, the Call Log Content Provider differentiates between Missed Calls, Received calls etc. using different paths.
- **4) ID:** It is not a mandatory component, and may not be present in the URI; but if present, it should be numeric. For instance, *if you want to access a specific music file from your Media Content Provider, you would specify an ID as well.*

- The ContentResolver object parses out the URI's authority, and uses it to "resolve" the provider by comparing the authority to a system table of known providers.
- The ContentResolver can then dispatch the query arguments to the correct provider.

- **The process**
- Using the provider authority, the ContentResolver identifies the correct content provider (as authority is unique for each content provider). Having done that, the path component of URI is used to select the correct (requested) data table. In case an ID is present, the provider would know what exact data is being requested.



Step 2: How to retrieve data from a Content Provider

- Even though the ContentResolver has access to the data table now, it cannot retrieve the required data unless the application has *"read access permission "* for that particular provider.
- This permission is defined in the manifest file of each content provider.
- All that an application *(that wants to access this database)* has to do is to request this permission.
- Now as discussed earlier, ***four operations can be carried out using a content provider***.

QUERYING

- **1) URI**: It works exactly as explained above.
- **2) Projection:** The query should **return a set of columns from the entire database table**. This is known as projection. Passing null will return all columns, which is inefficient.
- **3) Selection Clause:** A **filter declaring which rows to return**, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given URI.
- **4) Selection Argument:** You may include "?s" in selection, which will be replaced by the values from selection Args, in the order that they appear in the selection.
- **5) SortOrder:** SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will fetch the results which may be unordered.

- Create a class that extends ContentProvider
- Create a contract class
- Create the UriMatcher definition
- Implement the onCreate() method
- Implement the getType() method
- Implement the CRUD methods
- Add the content provider to your AndroidManifest.xml

```
import android.content.ContentProvider;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
public class  myProvider extends ContentProvider {

    @Override
    public String getType(Uri uri) {
        return "";
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String
selection, String[] selectionArgs, String sortOrder) {
        return null;
    }

    @Override
    public Uri insert(Uri uri,
ContentValues values) {
        return null;
    }

    @Override
    public int delete(Uri uri, String
selection, String[] selectionArgs) {
        return 0;
    }

    @Override
    public int update(Uri uri,
ContentValues values, String
selection, String[] selectionArgs) {
        return 0;
    }
}
```