SYNCHRONISATION AND REPLICATION

OVERVIEW

Replication

ANDROID SQL

RAW SQL EMBEDDING

```
void execSQL (String sql)
```

```
db.execSQL("create table pets(name text, age integer)");
```

- multiple SQL statements **cannot** be batched in a single method call. Silently ignored !
- returns **void**, if the sql statement returns data, the method throws an exception.

RAW QUERIES

```
Cursor rawQuery (String sql, String[] selectionArgs)
```

```
Cursor c = db.rawQuery("pragma table_info(" + tableName + ")", null);
```

- Returns a `Cursor` to process the result
- Very low level (use as a last resort)

SQL

```
db.execSQL("delete from people where name='"
           + person
           + "' and added_date < "
           + String.valueOf(System.currentTimeMillis() - HISTORY));
```

SQL INJECTION [SECURITY]

```
db.execSQL("delete from people where name='"
           + person
           + "' and added_date < "
           + String.valueOf(System.currentTimeMillis() - HISTORY));
```

What happens if the variable person contains the value `fred';` ?

SYNTACTIC SQL

- API form of SQL
- Safer methods to minimise SQL hacking
- Support for most common SQL operations:

  `insert, update, delete and query`

DELETE

```
DELETE
FROM pets
WHERE age > 10 AND age < 20
```

```
db.delete("pets", "age > 10 AND age < 20", null);
```

DELETE

```
CREATE TABLE pets(name text, age integer)
INSERT INTO pets VALUES("linus", 14)
INSERT INTO pets VALUES("fellini", 15)
INSERT INTO pets VALUES("totoro", 8)
```

```
db.delete("pets", "age = ? OR name = ?", new String[] {"15", “linus”});

db.delete("pets", "age = ?2 OR name = ?1", new String[] {"linus", "15"});
```

UPDATE

```
UPDATE pets
SET age = 99
WHERE name = "linus" OR name = "fellini"
```

```
ContentValues newAges = new ContentValues();
newAges.put("age", Integer.valueOf(99));
db.update(
"pets",
newAges,
"name = ? OR name = ?",
new String[] {"linus", "fellini"});
```

INSERT

```
ContentValues newPet = new ContentValues();
newPet.put("name", "luna");
newPet.put("age", 99);
db.insert("pets", null, newPet);
```

QUERY

```
SELECT table1.name, sum(table2.price)
   FROM table1, table2
   WHERE table1.supplier = table2.id AND table1.type = "spigot"
   GROUP BY table1.name
   HAVING sum(table2.price) < 50
   ORDER BY table1.name ASC
   LIMIT 100
```

QUERY

```
query(String table,
      String[] columns,
      String selection,
      String[] selectionArgs,
      String groupBy,
      String having,
      String orderBy,
      String limit)
```

```
Cursor c = db.query( "pets",
new String[] { "name", "age" },
"age > ?", new String[] { "50" },
null, // group by
null, // having "name ASC");
```

table — The FROM clause: a table to query

columns — A list of columns to be included in the result the projection

selection, selectionArgs — The WHERE clause and its arguments

groupBy — The GROUP BY clause

having — The HAVING clause

orderBy — The ORDER BY clause

TRANSACTIONS

```
db.beginTransaction();
try {
  // sql...
  db.setTransactionSuccessful();
}
finally {
  db.endTransaction();
}
```

This is only important if you need to concurrently access the database …

ANDROID SQL PRACTICE

RSS FEED DATA REPRESENTATION

Entry

_ID	Title	Subtitle

“CONTRACT” CLASS

```
public final class FeedReaderContract {
    // To prevent someone from accidentally instantiating the contract class,
    // make the constructor private.
    private FeedReaderContract() {}

    /* Inner class that defines the table contents */
    public static class FeedEntry implements BaseColumns {
        public static final String TABLE_NAME = "entry";
        public static final String COLUMN_NAME_TITLE = "title";
        public static final String COLUMN_NAME_SUBTITLE = "subtitle";
    }
}
```

BASECOLUMNS

Summary

Constants	
`String`	`_COUNT` The count of rows in a directory.
`String`	`_ID` The unique ID for a row.

BENEFITS OF A “CONTRACT” CLASS

“The contract class allows you to use the same constants across all the other classes in the same package. This lets you change a column name in one place and have it propagate throughout your code.”

CREATING A DATABASE: THE SQLITEOPENHELPER CLASS

- In a typical web service, creating a database is likely to be a distinct, infrequent, and substantial task.
- On a mobile platform, the application has to create it's own database.
- The `SQLiteOpenHelper` class is Android's hedge against this edge case.

SQLITE OPEN HELPER

```
public class FeedReaderDbHelper extends SQLiteOpenHelper {
    // If you change the database schema, you must increment the database version.
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "FeedReader.db";

    public FeedReaderDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // This database is only a cache for online data, so its upgrade policy is
        // to simply to discard the data and start over
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

GET WRITEABLE DATABASE

```
FeedReaderDbHelper mDbHelper = new FeedReaderDbHelper(getContext());
// Gets the data repository in write mode
SQLiteDatabase db = mDbHelper.getWritableDatabase();

// Create a new map of values, where column names are the keys
ContentValues values = new ContentValues();
values.put(FeedEntry.COLUMN_NAME_TITLE, title);
values.put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle);

// Insert the new row, returning the primary key value of the new row
long newRowId = db.insert(FeedEntry.TABLE_NAME, null, values);
```

GET READABLE DATABASE

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    FeedEntry._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_SUBTITLE
    };

// Filter results WHERE "title" = 'My Title'
String selection = FeedEntry.COLUMN_NAME_TITLE + " = ?";
String[] selectionArgs = { "My Title" };

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_SUBTITLE + " DESC";

Cursor cursor = db.query(
    FeedEntry.TABLE_NAME,                     // The table to query
    projection,                               // The columns to return
    selection,                                // The columns for the WHERE clause
    selectionArgs,                            // The values for the WHERE clause
    null,                                     // don't group the rows
    null,                                     // don't filter by row groups
    sortOrder                                 // The sort order
    );
```

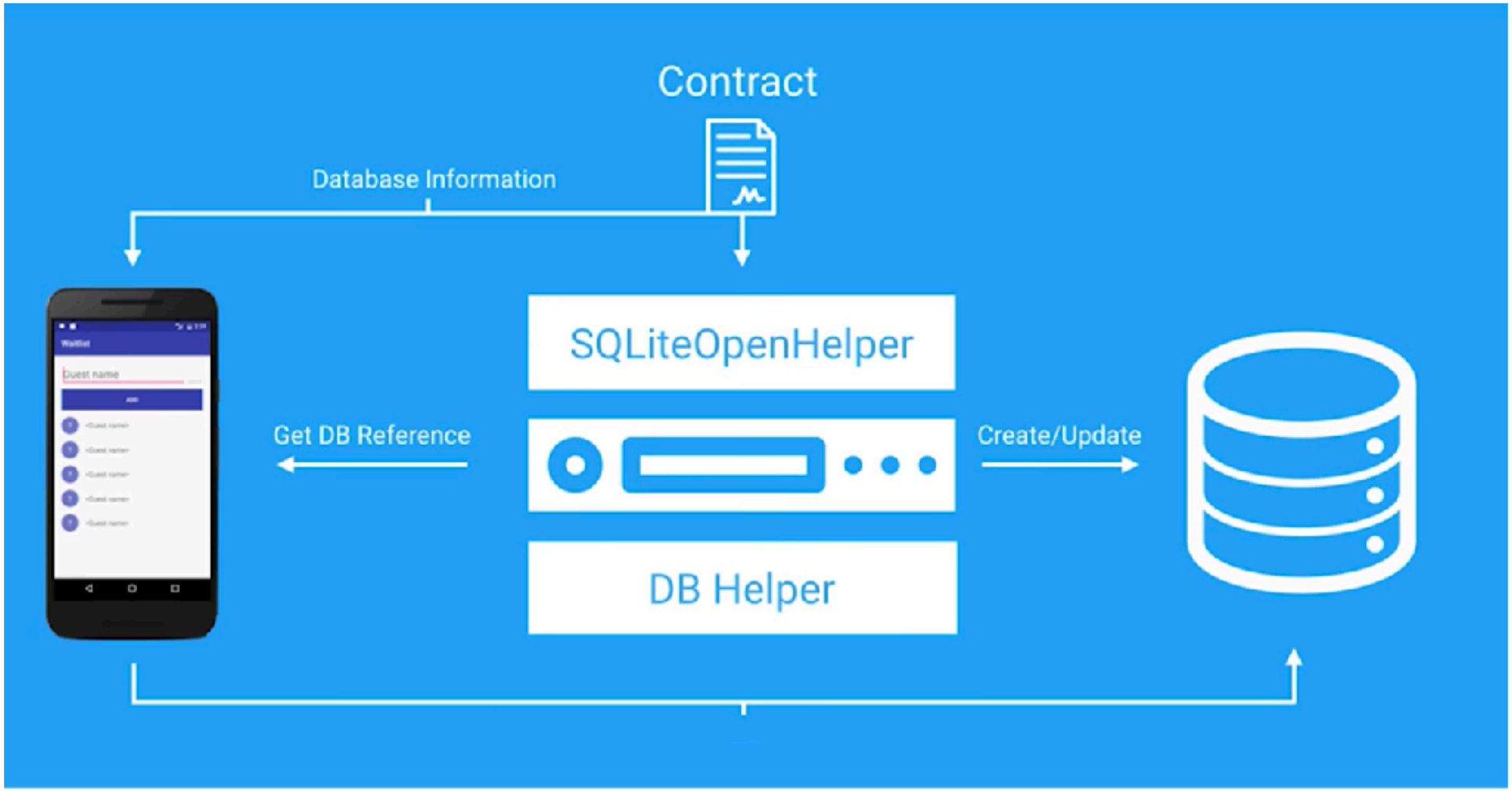
FIND THE PROBLEM

```
void insertKey(String key, int fk) {
  ContentValues r = new ContentValues();
  r.put(KeyValContract.Columns.KEY, key);
  r.put(COL_FK, Integer.valueOf(fk));
  getWriteableDatabase().insert(TAB_KEYS, null, r);
}

void insertVal(String val, int id) {
  ContentValues r = new ContentValues();
  r.put(COL_ID, Integer.valueOf(id));
  r.put(KeyValContract.Columns.VAL, val);
  getWriteableDatabase().insert(TAB_VALS, null, r);
}
```

```
public void onCreate(SQLiteDatabase db) {
  db.execSQL("CREATE TABLE " + TAB_VALS + "(" + COL_ID + " integer PRIMARY KEY," +
  KeyValContract.Columns.VAL + " text)");
  insertVal("bar", 1);
}
```

OVERVIEW


Contract
Database Information
SQLiteOpenHelper
Get DB Reference
Create/Update
DB Helper

CURSORS

CURSOR

Entry

_ID	Title	Subtitle

CURSORS

```
while (c.moveToNext()) {
// get data from a single row
}
```

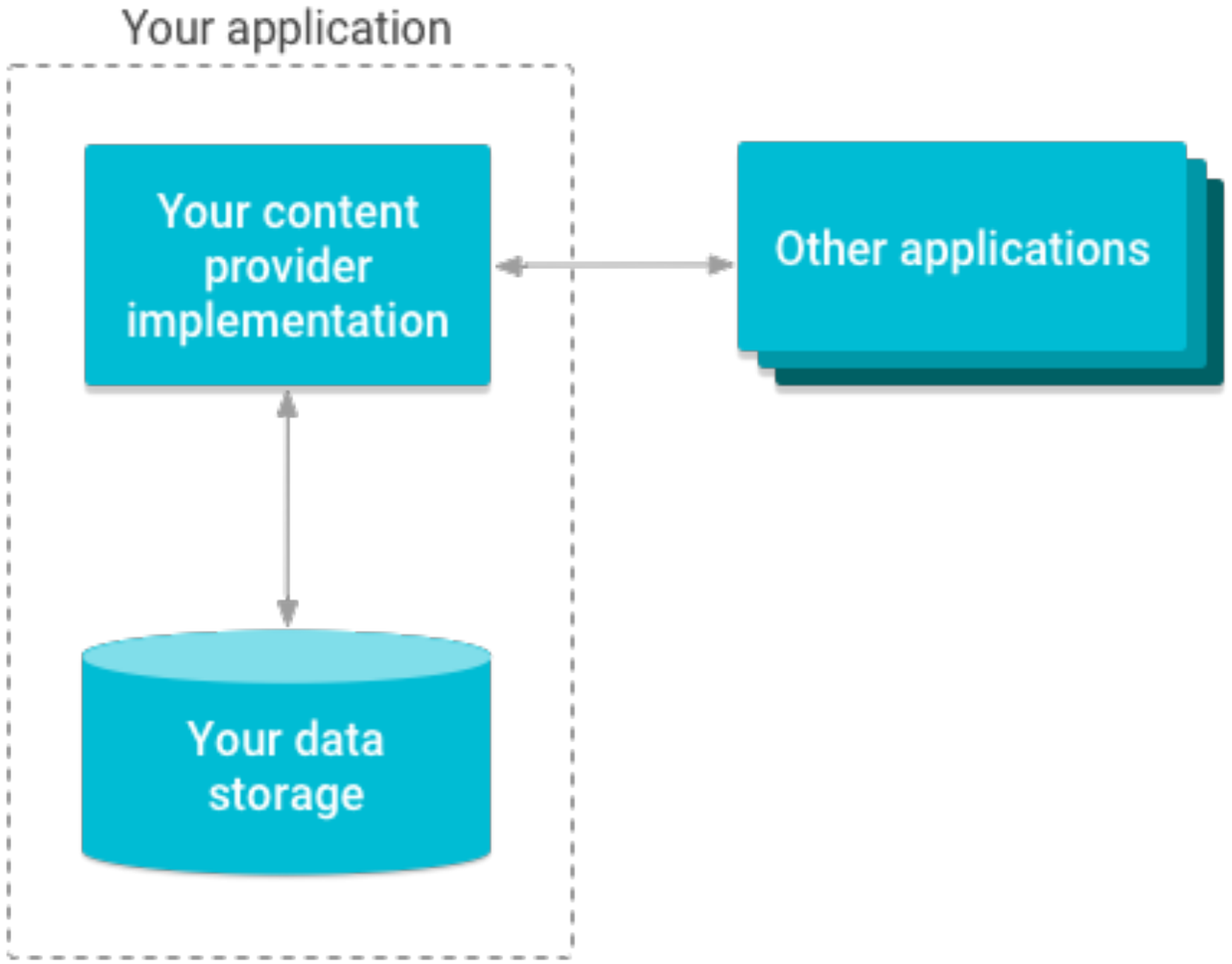
```
for (int i = 0; i < c.getCount(); i++) {
   c.moveToPosition(i);
  // get data from a single row
}
```

CURSORS EXAMPLE

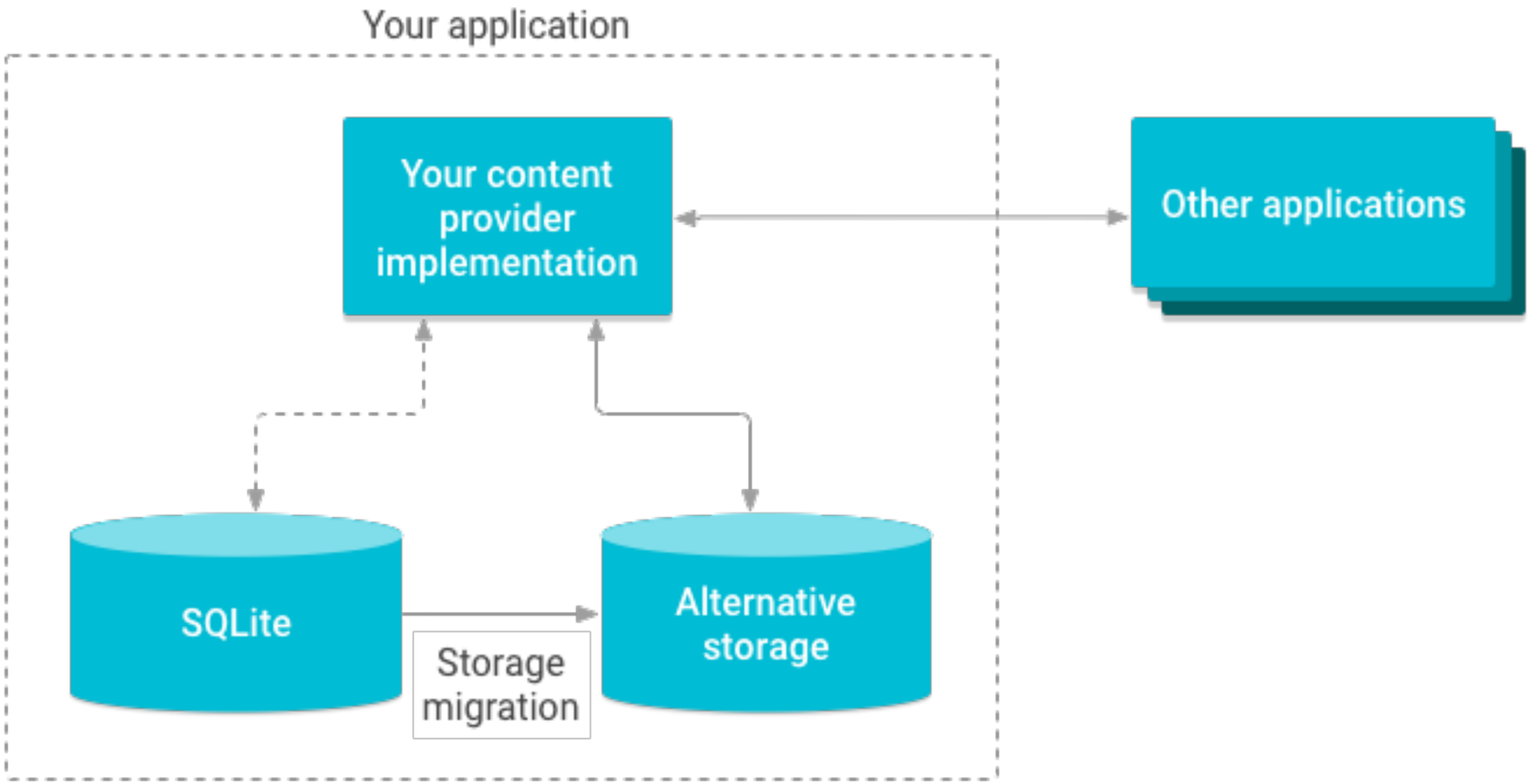
```
List itemIds = new ArrayList<>();
while(cursor.moveToNext()) {
   long itemId = cursor.getLong(
       cursor.getColumnIndexOrThrow(FeedEntry._ID));
   itemIds.add(itemId);
}
cursor.close();
```

CONTENT PROVIDER

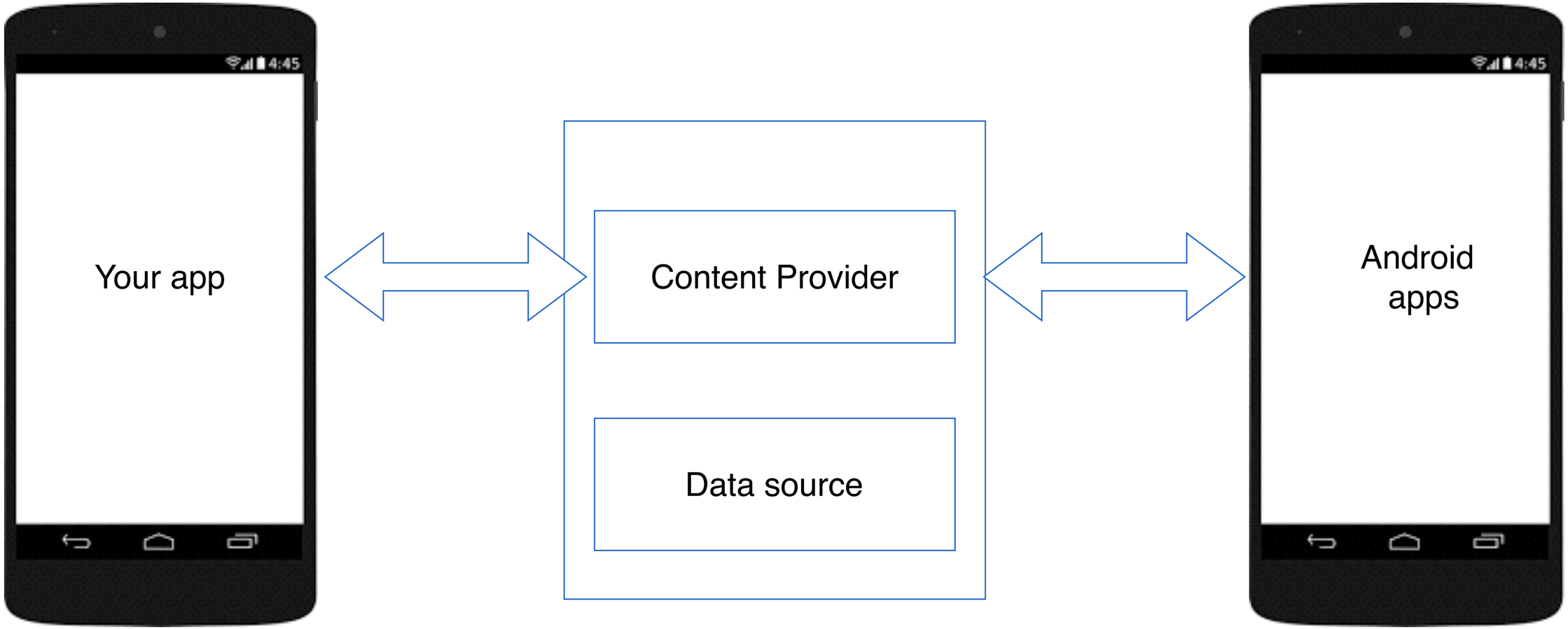
CONTENT PROVIDERS



CONTENT PROVIDERS



CONTENT PROVIDERS



WHY USE CONTENT PROVIDERS

- Easy to change the underlying data source
- To fit into the android ecosystem (Loaders)
- Allow multiple apps to use the same data in a secure way

STEPS FOR USING A CONTENT PROVIDER

- Get permission to use the content provider
- Get the content Resolver
- Perform a basic operation query, insert, update, delete
- Use a URI to identify the data you want to read or manipulate
- Display the data

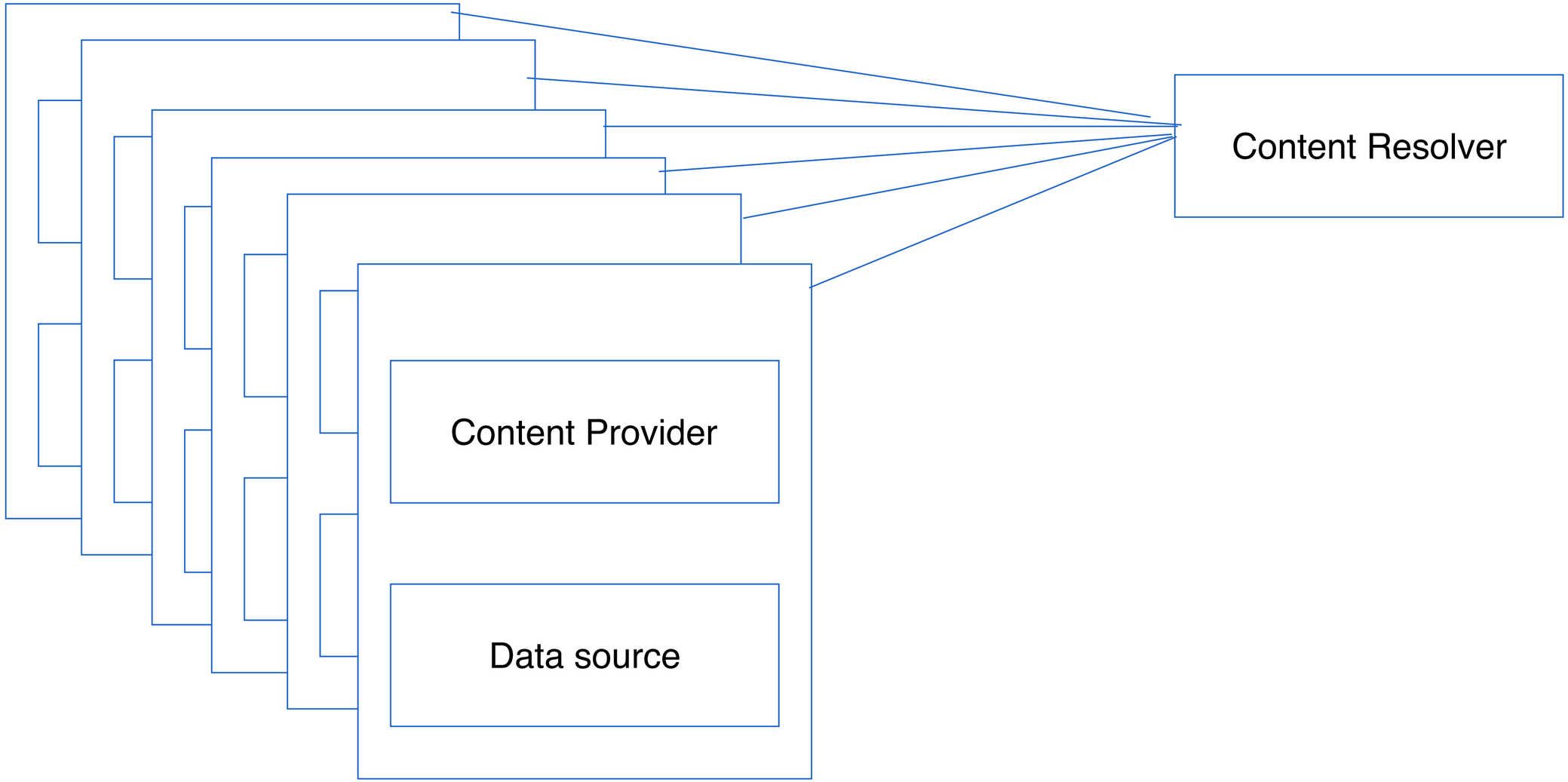
PERMISSIONS

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

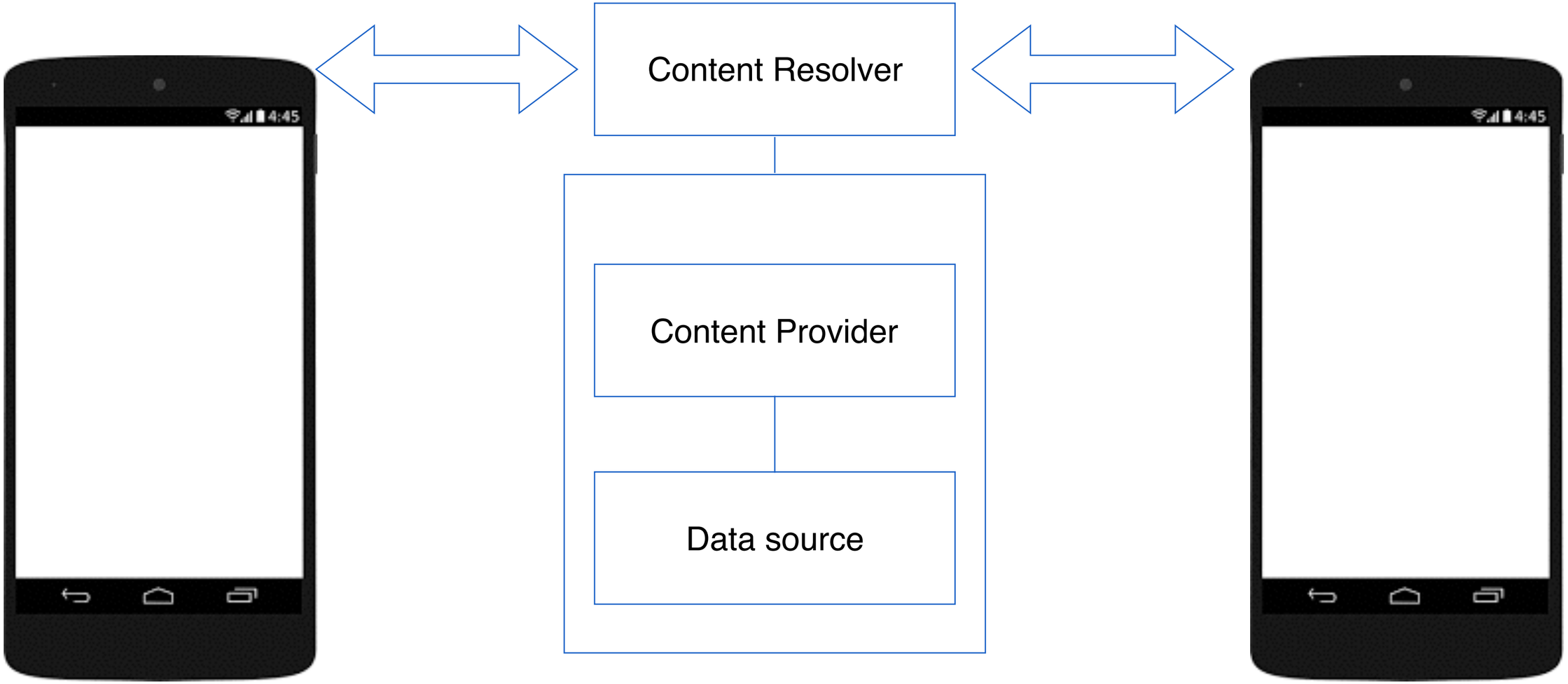
CONTENT RESOLVER

```
ContentResolver cr = getContentResolver();
Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
                      null, null, null, null);
```

CONTENT RESOLVER



CONTENT RESOLVER



BASIC OPERATIONS ON A CONTENT PROVIDER

- Read query()
- Add a row insert()
- Update the data update()
- Delete a row or rows form the data delete()

URI STRUCTURE

<scheme> <Authority> <Path>

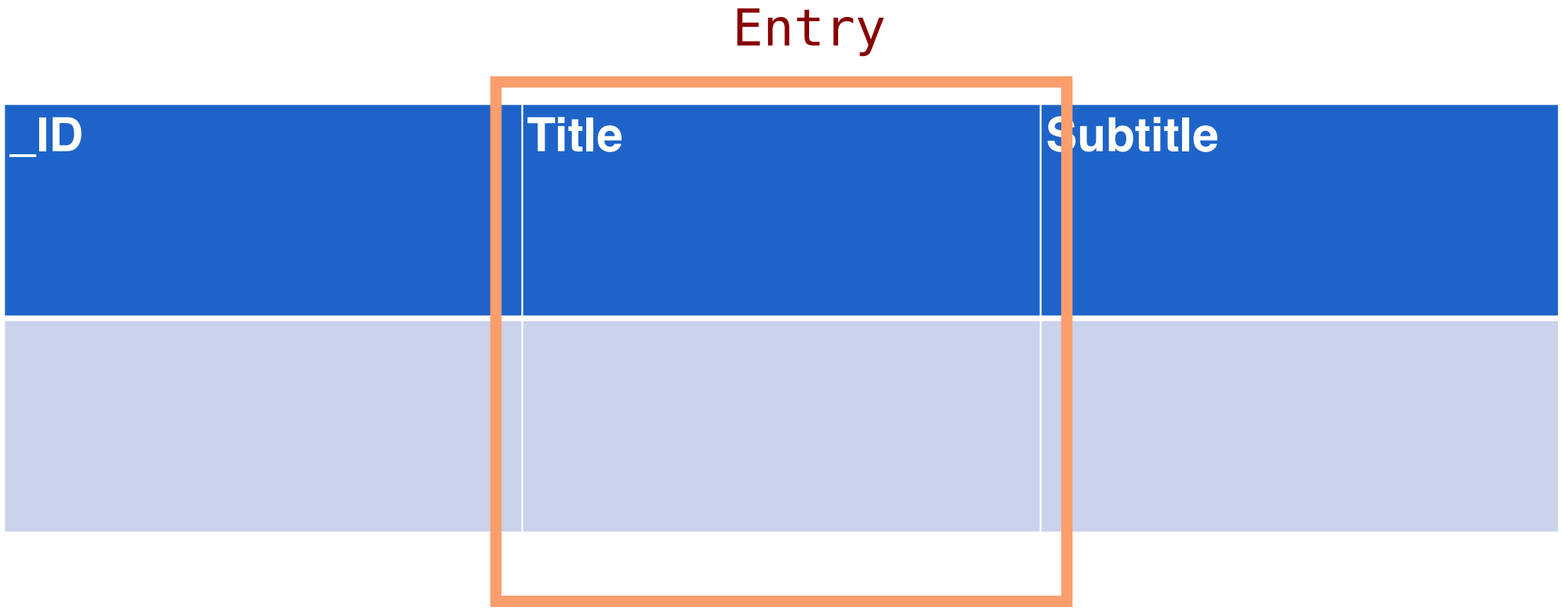
content://com.android.example.todolist/task

QUERY

```
Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
                      <projection>,
                      <selection>,
                      <selection args>,
                      <sort order>);
```

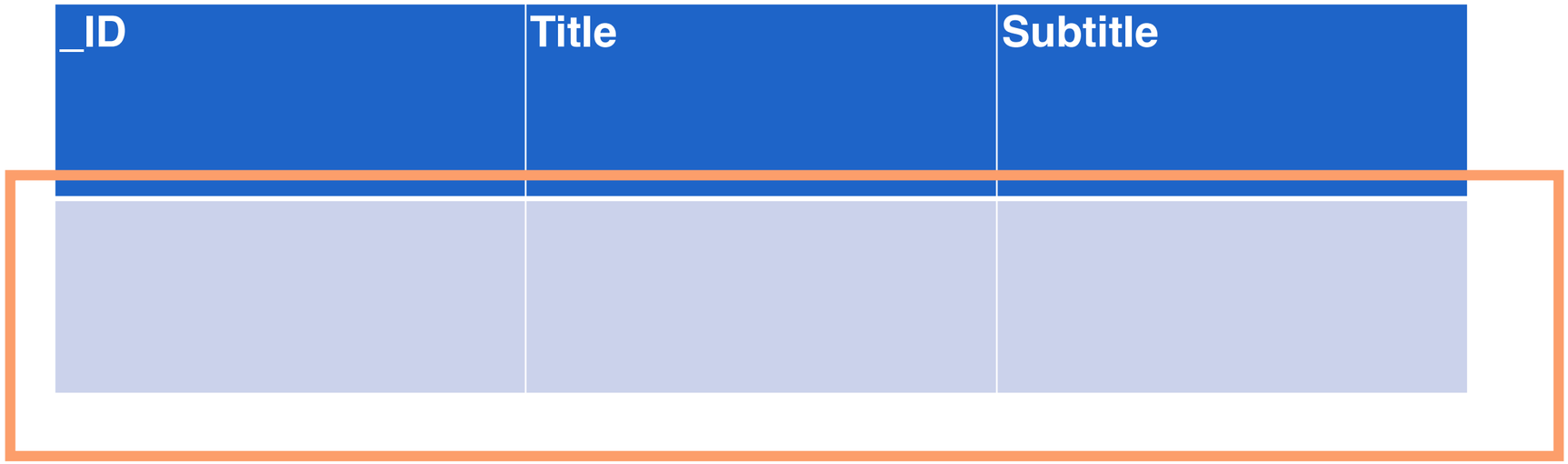
PROJECTION

```
Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
                      <projection>,
                      <selection>,
                      <selection args>,
                      <sort order>);
```



SELECTION

```
Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
                      <projection>,
                      <selection>,
                      <selection args>,
                      <sort order>);
```



SORT ORDER

```
Cursor cur = cr.query(ContactsContract.Contacts.CONTENT_URI,
                      <projection>,
                      <selection>,
                      <selection args>,
                      <sort order>);
```

_ID	Title	Subtitle

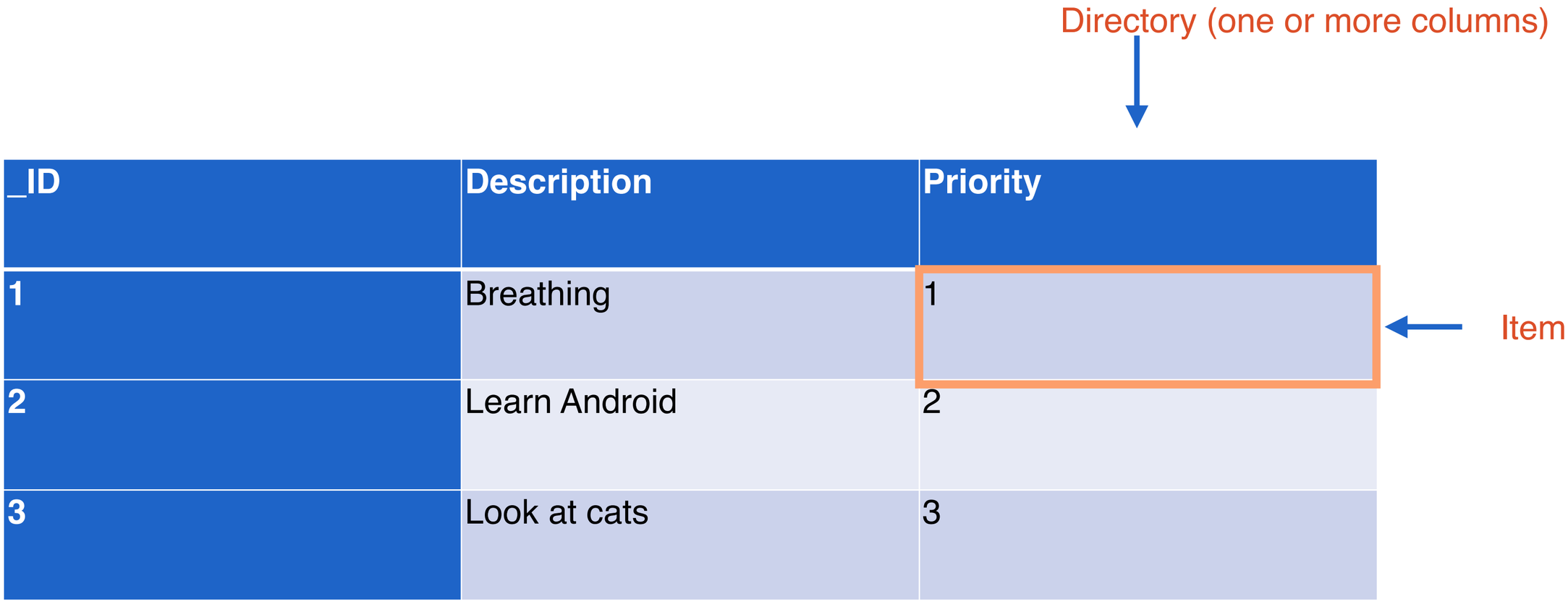
MAKING A CONTENT PROVIDER

TODO LIST APPLICATION

Tasks

_ID	Description	Priority
1	Breathing	1
2	Learn Android	2
3	Look at cats	3

TODO LIST APPLICATION



_ID	Description	Priority
1	Breathing	1
2	Learn Android	2
3	Look at cats	3

TASK CONTRACT

```
public class TaskContract {


    /* TaskEntry is an inner class that defines the contents of the task table */
    public static final class TaskEntry implements BaseColumns {


        // Task table and column names
        public static final String TABLE_NAME = "tasks";

        // Since TaskEntry implements the interface "BaseColumns",
        // it has an automatically produced
        // "_ID" column in addition to the two below
        public static final String COLUMN_DESCRIPTION = "description";
        public static final String COLUMN_PRIORITY = "priority";

    }
}
```

OVERVIEW OF CREATING A CONTENT PROVIDER

1. Create a class that extends the content provider
2. Register the content provider in the manifest
3. Define the URI's
4. Add the URI's to the Contract class
5. Build a URI matcher to match URI patterns to integers
6. Implement the required CRUD methods

TASKCONTENT PROVIDER

```
public class TaskContentProvider extends ContentProvider {
    private TaskDbHelper mTaskDBHelper;

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

    @Override
    public Uri insert(@NonNull Uri uri, ContentValues values) {

        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public Cursor query(@NonNull Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public int delete(@NonNull Uri uri, String selection, String[] selectionArgs) {

        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public int update(@NonNull Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {

        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public String getType(@NonNull Uri uri) {

        throw new UnsupportedOperationException("Not yet implemented");
    }

}
```

INITIALISE THE DBHELPER

```
private TaskDbHelper mTaskDBHelper;

@Override
public boolean onCreate() {
    Context context = getContext();
    mTaskDBHelper = new TaskDbHelper(context);
    return true;
}
```

REGISTER THE CONTENT PROVIDER

Package name of the application

```
<provider
  android:authorities="com.example.android.todolist"
  android:name="com.example.android.todolist.data.TaskContentProvider"
  android:exported="false"/>
```

Name of the content provider class

URI

- Identify the provider
- Identify the different type of data

URI STRUCTURE

<scheme> <Authority> <Path>

content://com.android.example.todolist/task

URI

Tasks

_ID	Description	Priority
1	Breathing	1
2	Learn Android	2
3	Look at cats	3

content://com.android.example.todolist/task/priority

URI

Tasks

_ID	Description	Priority
1	Breathing	1
2	Learn Android	2
3	Look at cats	3

content://com.android.example.todolist/task/1

URI WILDCARDS

Tasks

_ID	Description	Priority
1	Breathing	1
2	Learn Android	2
3	Look at cats	3

```
content://com.android.example.todolist/task/#
```

URI WILDCARDS

`#` Match a number

`*` Match a string

```
content://com.android.example.todolist/task/*
```

IN THE TODO LIST APP

Select one task

```
content://com.android.example.todolist/task/#
```

Select all tasks

```
content://com.android.example.todolist/task
```

STORE OUR URI'S IN THE TASK CONTRACT

```
public class TaskContract {
    public static final String AUTHORITY        = "com.android.example.todolist";
    public static final Uri BASE_CONTENT_URI    = Uri.parse("content://"+ AUTHORITY);
    public static final String PATH_TASKS       = "tasks";

    /* TaskEntry is an inner class that defines the contents of the task table */
    public static final class TaskEntry implements BaseColumns {
        public static final Uri CONTENT_URI = BASE_CONTENT_URI.buildUpon().appendPath(PATH_TASKS).build();

        // Task table and column names
        public static final String TABLE_NAME = "tasks";

        // Since TaskEntry implements the interface "BaseColumns", it has an automatically produced
        // "_ID" column in addition to the two below
        public static final String COLUMN_DESCRIPTION = "description";
        public static final String COLUMN_PRIORITY = "priority";
    }
}
```

MATCHING THE URI'S

```
if (UriString.equals("content://com.android.example.todolist/task/1")) {
      //matched
} else {
      //otherwise
}
```

Don't try this at home !

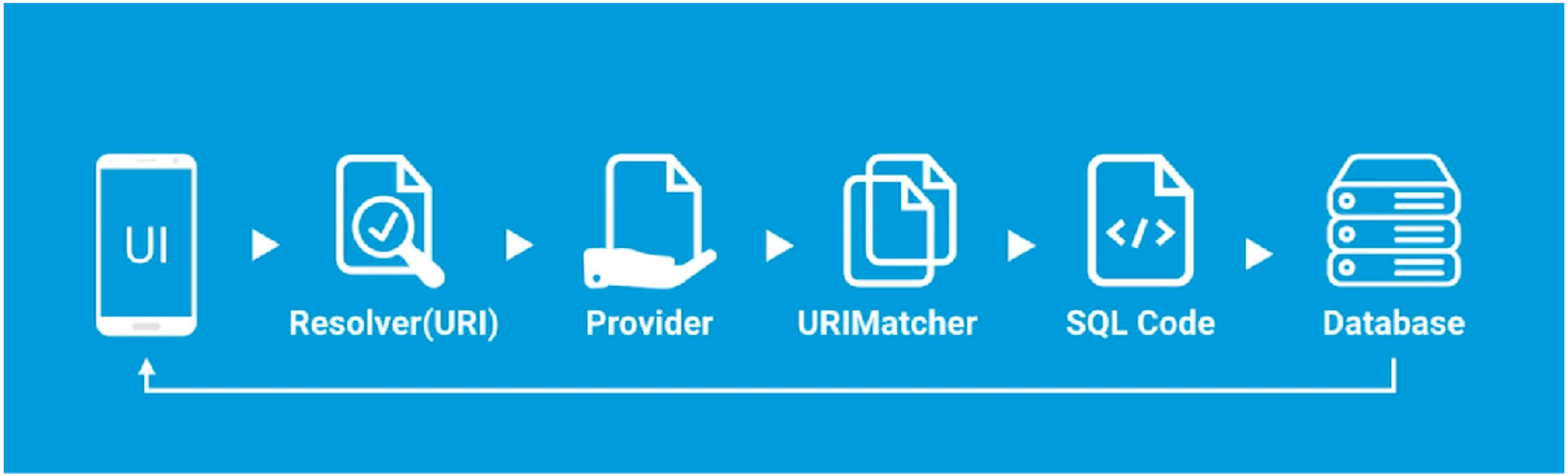
URIMATCHER

URI -> Num

```
public static final int TASK = 100;
public static final int TASK_WITH_ID = 101;
private static final UriMatcher sUriMatcher = buildUriMatcher();


public static UriMatcher buildUriMatcher() {
    UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI(TaskContract.AUTHORITY,TaskContract.PATH_TASKS,TASK);
    uriMatcher.addURI(TaskContract.AUTHORITY,TaskContract.PATH_TASKS+"/#",TASK_WITH_ID);
    return uriMatcher;
}
```

OVERVIEW



INSERT

```
public Uri insert(@NonNull Uri uri, ContentValues values) {
        // Get access to the task database (to write new data to)
        final SQLiteDatabase db = mTaskDbHelper.getWritableDatabase();

        // Write URI matching code to identify the match for the tasks directory
        int match = sUriMatcher.match(uri);
        Uri returnUri; // URI to be returned

        switch (match) {
            case TASKS:
                // Insert new values into the database
                // Inserting values into tasks table
                long id = db.insert(TABLE_NAME, null, values);
                if ( id > 0 ) {
                    returnUri = ContentUris.withAppendedId(TaskContract.TaskEntry.CONTENT_URI, id);
                } else {
                    throw new android.database.SQLException("Failed to insert row into " + uri);
                }
                break;
            // Set the value for the returnedUri and write the default case for unknown URI's
            // Default case throws an UnsupportedOperationException
            default:
                throw new UnsupportedOperationException("Unknown uri: " + uri);
        }

        // Notify the resolver if the uri has been changed, and return the newly inserted URI
        getContext().getContentResolver().notifyChange(uri, null);

        // Return constructed uri (this points to the newly inserted row of data)
        return returnUri;
    }
```

QUERY

```
public Cursor query(@NonNull Uri uri, String[] projection, String selection,
                    String[] selectionArgs, String sortOrder) {

        // (1) Get access to underlying database (read-only for query)
        final SQLiteDatabase db = mTaskDbHelper.getReadableDatabase();

        // (2) Write URI match code and set a variable to return a Cursor
        int match = sUriMatcher.match(uri);
        Cursor retCursor;

        // (3) Query for the tasks directory and write a default case
        switch (match) {
            // Query for the tasks directory
            case TASKS:
                retCursor =  db.query(TABLE_NAME,
                        projection,
                        selection,
                        selectionArgs,
                        null,
                        null,
                        sortOrder);
                break;
            // Default exception
            default:
                throw new UnsupportedOperationException("Unknown uri: " + uri);
        }

        // (4) Set a notification URI on the Cursor and return that Cursor
        retCursor.setNotificationUri(getContext().getContentResolver(), uri);

        // Return the desired Cursor
        return retCursor;
    }
```

MAKING A FILECONTENT PROVIDER IN XML

FILE PROVIDER

```
<application>
    ...
    <provider
         android:name="android.support.v4.content.FileProvider"
         android:authorities="com.example.android.fileprovider"
         android:exported="false"
         android:grantUriPermissions="true">
         <meta-data
             android:name="android.support.FILE_PROVIDER_PATHS"
             android:resource="@xml/file_paths"></meta-data>
     </provider>
     ...
</application>
```

Since the default functionality of FileProvider includes content URI generation for files, you don't need to define a subclass in code. Instead, you can include a FileProvider in your app by specifying it entirely in XML.

FILE PROVIDER

```
File imagePath = new File(Context.getFilesDir(), "images");
File newFile = new File(imagePath, "default_image.jpg");
Uri contentUri = getUriForFile(getContext(), "com.example.android.fileprovider", newFile);
```

Make sure that the authorities string matches
the second argument to `getUriForFile(Context, String, File)`.

SYNC ADAPTORS

SYNC ADAPTORS

- **Plug-in architecture**: Allows you to add data transfer code to the system in the form of callable components.
- **Automated execution**: Allows you to automate data transfer based on a variety of criteria, including data changes, elapsed time, or time of day. In addition, the system adds transfers that are unable to run to a queue, and runs them when possible.
- **Automated network checking**: The system only runs your data transfer when the device has network connectivity.
- **Improved battery performance**: Allows you to centralize all of your app's data transfer tasks in one place, so that they all run at the same time. Your data transfer is also scheduled in conjunction with data transfers from other apps. These factors reduce the number of times the system has to switch on the network, which reduces battery usage.
- **Account management and authentication:** If your app requires user credentials or server login, you can optionally integrate account management and authentication into your data transfer.

OVERVIEW

- Create an authenticator (*See android documentation*)
- Create a content provider (*We know that now*)
- Create a sync adaptor
- Create a service for the sync adaptor
- Run the sync adaptor

SYNC ADAPTOR

```
public class SyncAdapter extends AbstractThreadedSyncAdapter {
    ...
    // Global variables
    // Define a variable to contain a content resolver instance
    ContentResolver mContentResolver;
    /**
     * Set up the sync adapter
     */
    public SyncAdapter(Context context, boolean autoInitialize) {
        super(context, autoInitialize);
        /*
         * If your app uses a content resolver, get an instance of it
         * from the incoming Context
         */
        mContentResolver = context.getContentResolver();
    }
    ...
    /**
     * Set up the sync adapter. This form of the
     * constructor maintains compatibility with Android 3.0
     * and later platform versions
     */
    public SyncAdapter(
            Context context,
            boolean autoInitialize,
            boolean allowParallelSyncs) {
        super(context, autoInitialize, allowParallelSyncs);
        /*
         * If your app uses a content resolver, get an instance of it
         * from the incoming Context
         */
        mContentResolver = context.getContentResolver();
        ...
    }
```

SYNC ADAPTOR

```
/*
 * Specify the code you want to run in the sync adapter. The entire
 * sync adapter runs in a background thread, so you don't have to set
 * up your own background processing.
 */
@Override
public void onPerformSync(
        Account account,
        Bundle extras,
        String authority,
        ContentProviderClient provider,
        SyncResult syncResult) {
/*
 * Put the data transfer code here.
 */
...
}
```

SYNC ADAPTOR

```
 public void onPerformSync(Account account, Bundle extras, String authority,
                           ContentProviderClient provider, SyncResult syncResult) {
           final URL location = new URL(FEED_URL);
           InputStream stream = null;
           Log.i(TAG, "Streaming data from network: " + location);
           stream = downloadUrl(location);
           updateLocalFeedData(stream, syncResult);
}

 /* <p>Merge strategy:
 * 1. Get cursor to all items in feed<br/>
 * 2. For each item, check if it's in the incoming data.<br/>
 *    a. YES: Remove from "incoming" list. Check if data has mutated, if so, perform
 *            database UPDATE.<br/>
 *    b. NO: Schedule DELETE from database.<br/>
 * (At this point, incoming database only contains missing items.)<br/>
 * 3. For any items remaining in incoming list, ADD to database.
 */
public void updateLocalFeedData(final InputStream stream, final SyncResult syncResult)
        throws IOException, XmlPullParserException, RemoteException,
        OperationApplicationException, ParseException {
       …
 }
```

RUNNING THE SYNC ADAPTOR

```
public class MainActivity extends FragmentActivity {
    ...
    // Constants
    // Content provider authority
    public static final String AUTHORITY = "com.example.android.datasync.provider";
    // Account
    public static final String ACCOUNT = "default_account";
    // Sync interval constants
    public static final long SECONDS_PER_MINUTE = 60L;
    public static final long SYNC_INTERVAL_IN_MINUTES = 60L;
    public static final long SYNC_INTERVAL =
            SYNC_INTERVAL_IN_MINUTES *
            SECONDS_PER_MINUTE;
    // Global variables
    // A content resolver for accessing the provider
    ContentResolver mResolver;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        // Get the content resolver for your app
        mResolver = getContentResolver();
        /*
         * Turn on periodic syncing
         */
        ContentResolver.addPeriodicSync(
                ACCOUNT,
                AUTHORITY,
                Bundle.EMPTY,
                SYNC_INTERVAL);
        ...
    }
    ...
}
```

REPLICATION

THE TWO GENERALS PROBLEM

- *Two generals need to coordinate the time of an attack*
- *They can only win if they attack simultaneously*
- *They communicate through messengers*
- *Messengers may be killed on their way*

THE TWO GENERALS PROBLEM

- *Two generals need to coordinate the time of an attack*
- *They can only win if they attack simultaneously*
- *They communicate through messengers*
- *Messengers may be killed on their way*

E.g. in databases both parties have to agree on {rollback, commit} (called atomic commit)

Two nodes have to <u>agree</u> on a value while relying on unreliable communication

How?

Theorem

This is impossible to solve. A general can only be sure after an “ack”, but the “ack” can also be killed and so do “acks of the ack” etc

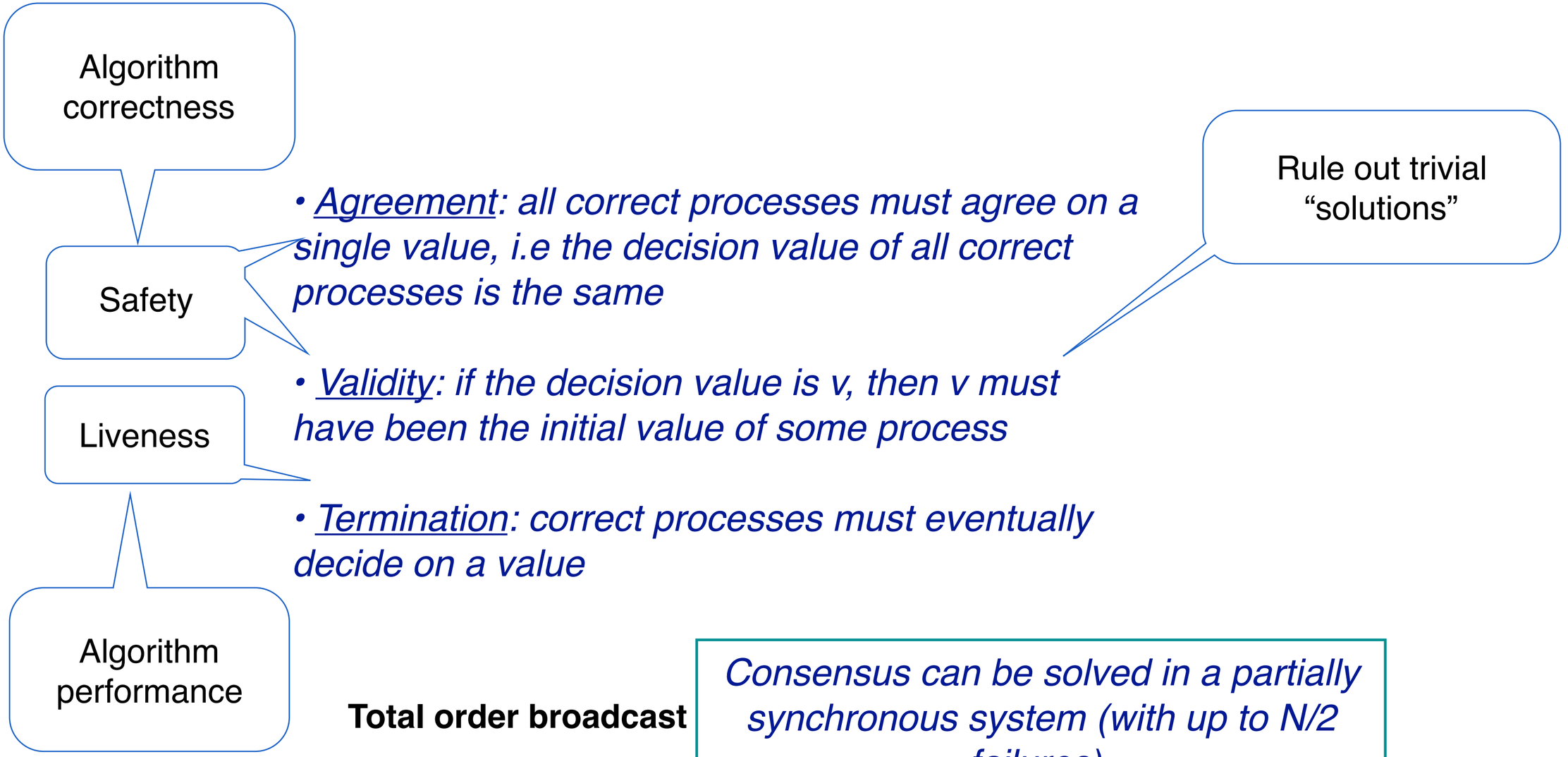
THE CONSENSUS PROBLEM

All distributed nodes propose a value and may crash. The algorithm must ensure that all non-crashed nodes agree on one of the proposed values.

E.g. all nodes send a message to all other nodes. All of them should process the same messages in the same order (called atomic broadcast)

Basis for implementing e.g. replication (requiring atomic commit)

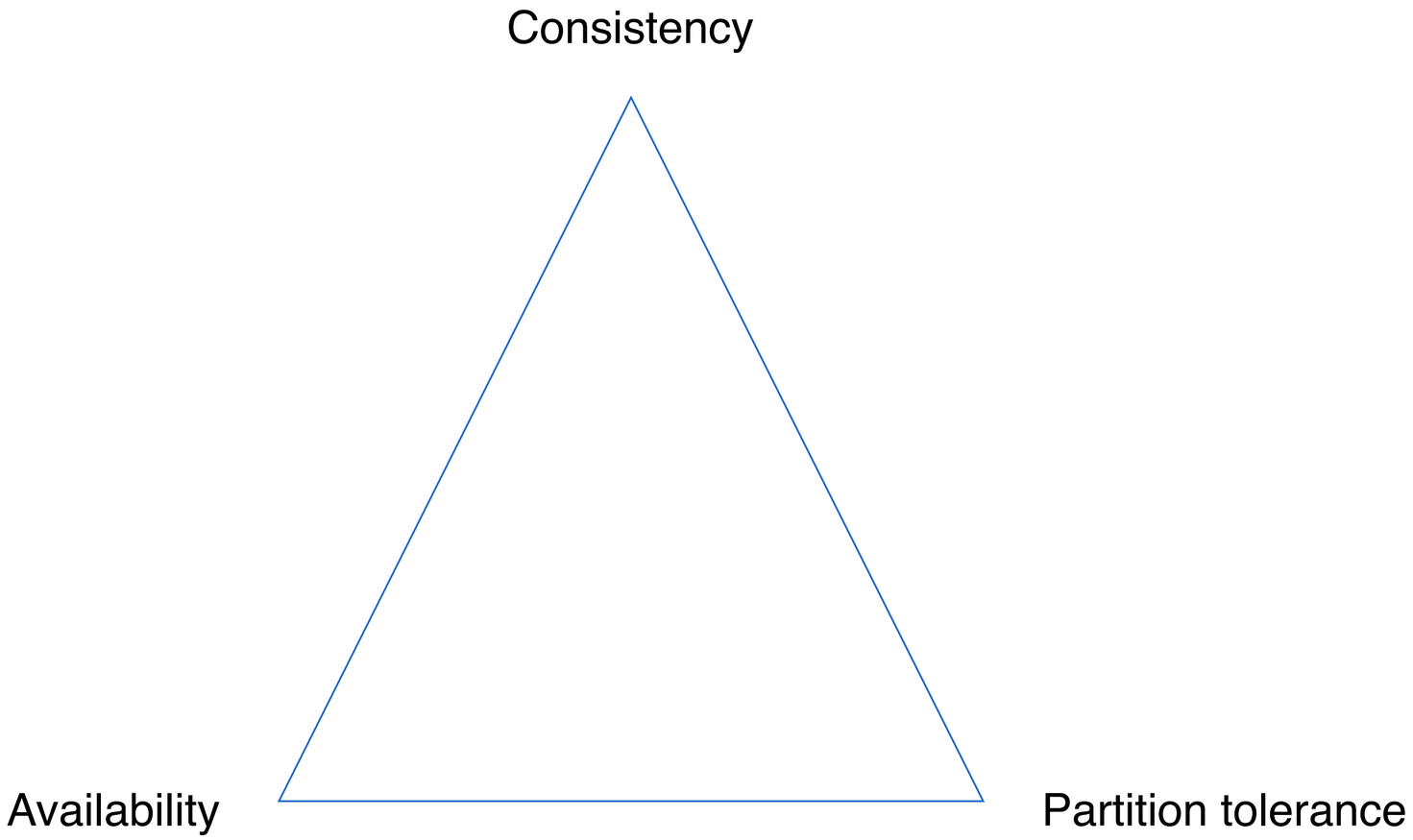
A SOLUTION REQUIRES...



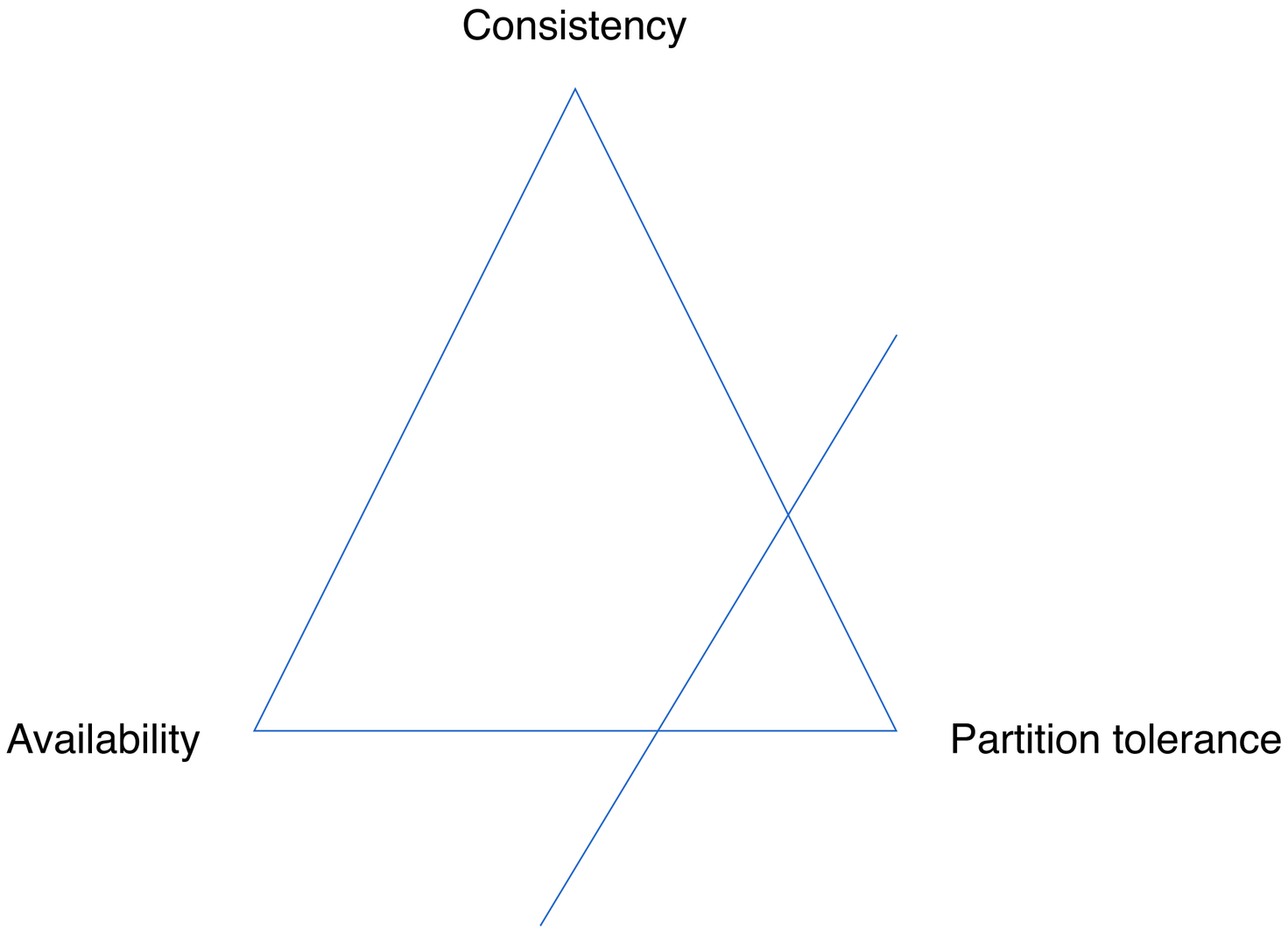
Total order broadcast

Consensus can be solved in a partially synchronous system (with up to N/2 failures).

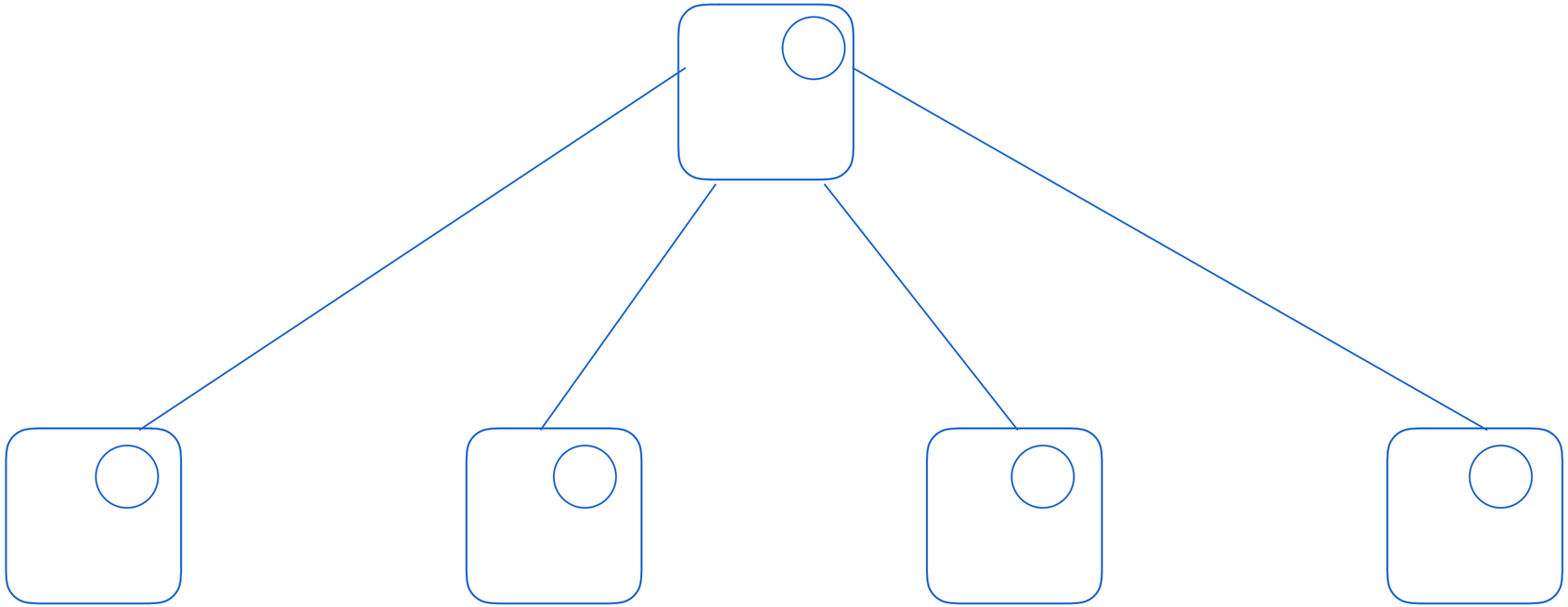
CAP THEOREM



CAP THEOREM



EVENTUAL CONSISTENCY



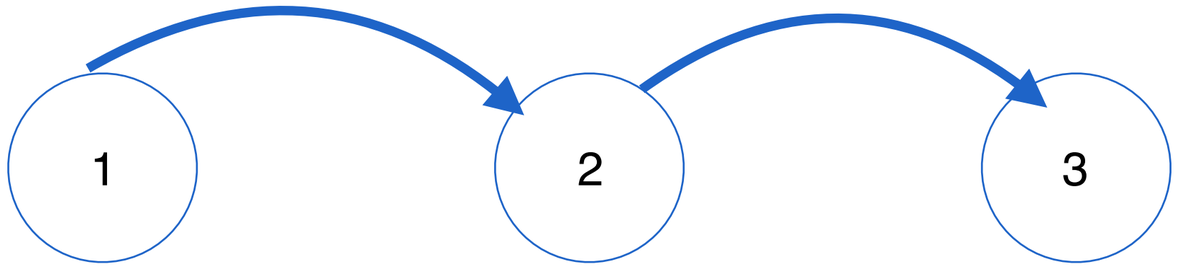
If no new updates are made to a given data item, **eventually** all accesses to that item will return the last updated value.

CORE GPS

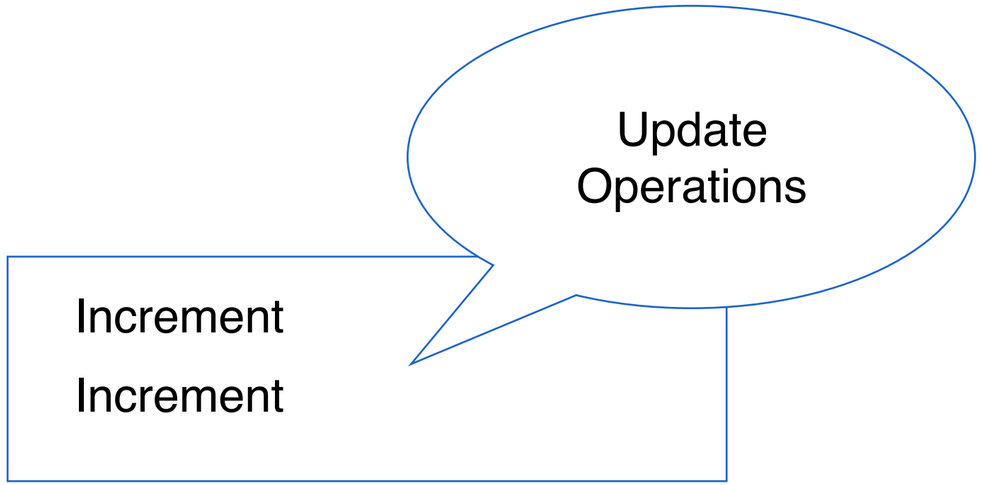
GLOBAL SEQUENCE PROTOCOL

- **Reasoning**. *Above all, what is needed is a simple mental reference model to understand how to use the mechanism appropriately to write correct programs.*
- **Judicious Synchronization**. *It is important that programmers can easily choose between synchronous and asynchronous reads and updates.*
- **Robust Implementations.** *Implementations must be carefully engineered to hide failures of clients, the network, and the server, and to minimize the amount of data stored and transmitted.*

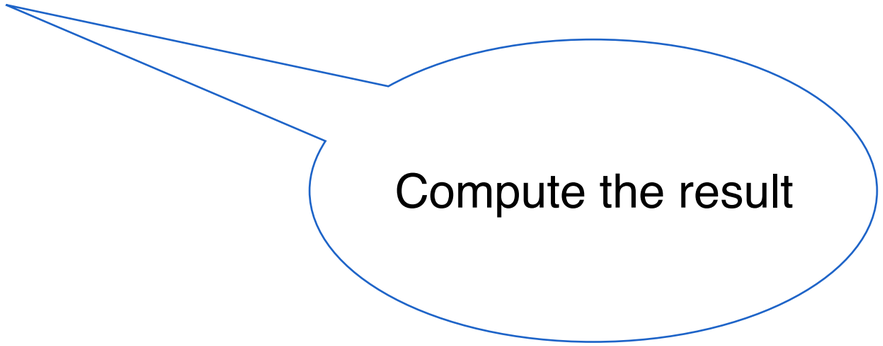
REPLICATING DATA



State view



Sequence of updates



SEQUENCE NOTATIONS

$T*$	Type of sequences of type T
$[]$	the empty sequence
$s1 \cdot s2$	the concatenation of two sequences
$s[i]$	the element at position i

DATA MODEL

abstract type $Update, Read, Value;$

function $rvalue : Read \times Update * \rightarrow Value$

REGISTER

$$Update = \{\, wr(v) \mid v \text{ in } \mathrm{Value} \}$$

$$Read = \{\, rd \,\}$$

$$rvalue(rd, s) = \textbf{match } s \textbf{ with}$$

$$[] \rightarrow \textbf{undefined}$$

$$s0 \cdot wr(v) \rightarrow v$$

COUNTER

$$Update = \{ inc \}$$
$$Read = \{ rd \}$$

$$rvalue(rd, s) = s.length$$

KEY-VALUE STORE

$$Update = \{\ wr(k,v) \mid k,v \text{ in } Value\ \}$$

$$Read = \{rd(k) \mid k \text{ in } Value\}$$

$$rvalue(rd(k),\ s) = \textbf{match } s \textbf{ with}$$

$$[\,] \rightarrow \textbf{undefined}$$

$$s0 \cdot wr(k0,v) \rightarrow \textbf{if } (k = k0) \textbf{ then}$$

$$v$$

$$\textbf{else}$$

$$rvalue(rd(k),s0)$$

CORE GSP

```
role Core_GSP_Client {
   known   :  Round * := []; // known prefix of global sequence
  pending  : Round * := []; // sent, but unconfirmed rounds
  round    :  N := 0; // counts submitted rounds

  // client program interface
  update(u : Update)    { …  }
  read(r : Read) : Value { … }
  // network interface
  onReceive(r : Round) {…}
}

 // rounds data structure
 class Round { origin : Client, number : N, update : Update }
 function updates(s : Round * ) : Update * { return s[0].update · · · s[s.length - 1].update; }
```

Each client stores a currently known prefix of the global update sequence in known, and a sequence of pending updates in pending.

UPDATE

```
update(u : Update) {
   pending := pending · u;
   RTOB_submit( new Round { origin = this, number = round ++ , update = u } );
}
```

When the client program issues an update, we (1) append this update to the sequence of pending updates, and (2) wrap the update into a Round object, which includes the origin and a sequence number, and broadcast the round.

READ

```
read(r : Read) : Value {
    var compositelog := known · pending;
    return rvalue(r, updates(compositelog));
}
```

Reading a value boils down to calculating its rvalue based on both the *known* and *pending* update log.

ONRECEIVE

```
// network interface
onReceive(r : Round) {
    known := known · r;
    if (r.origin = this) {
        assert(r = pending[0]);    // due to RTOB total order
        pending := pending[1..];  // remove first
    }
}
```

When receiving a round, we append the contained update to the known prefix of updates. Moreover, if this round is an echo (it originated on the same client), we remove it from the pending queue.

TRANSACTIONAL GSP

TRANSACTIONAL GSP

```
role GSP_Client {

  known : Round * := []; // known prefix of global sequence
  pending : Round * := []; // sent, but unconfirmed rounds
  round : N := 0; // counts submitted rounds
  transactionbuffer : Update* := [];
  receivebuffer : Round * := [];

  // client program interface
  update(u : Update) { }
  read(r : Read) : Value { }
  confirmed() : boolean { }
  push() { }
  pull() { }
}

// network interface
onReceive(r : Round) { receivebuffer := receivebuffer · r; }
// rounds data structure
class Round { origin : Client, number : N, updates : Update }
function updates(s : Round * ) : Update * { return s[0].updates · · · s[s.length - 1].updates;} }
```

UPDATE

$$
\begin{aligned}
&update(u : Update)\ \{\\
&\quad transactionbuffer := transactionbuffer \cdot u;\\
&\}
\end{aligned}
$$

PUSH

```
push() {
  var r := new Round { origin = this,
                       number = round ++ ,
                       updates = transactionbuffer
                     };
  tob_submit( r );
  pending := pending · r;
  transactionbuffer := [];
}
```

TRANSACTIONAL GSP

```
pull() {
    foreach(var r in receivebuffer) {
        known := known · r;
        if (r.origin = this) { pending := pending[1..]; }
}
```

READ

```
read(r : Read) : Value {
    var compositelog := updates(known) · updates(pending) · transactionbuffer;
    return rvalue(r, compositelog);
}
```

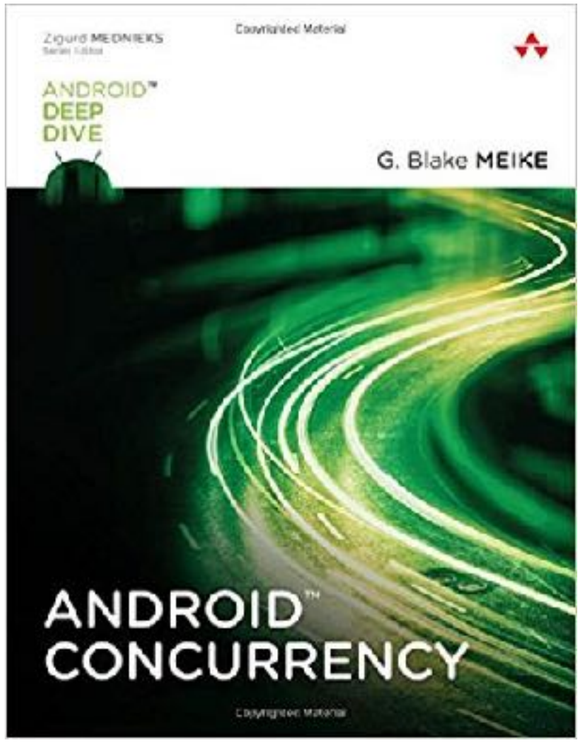

That's all Folks!

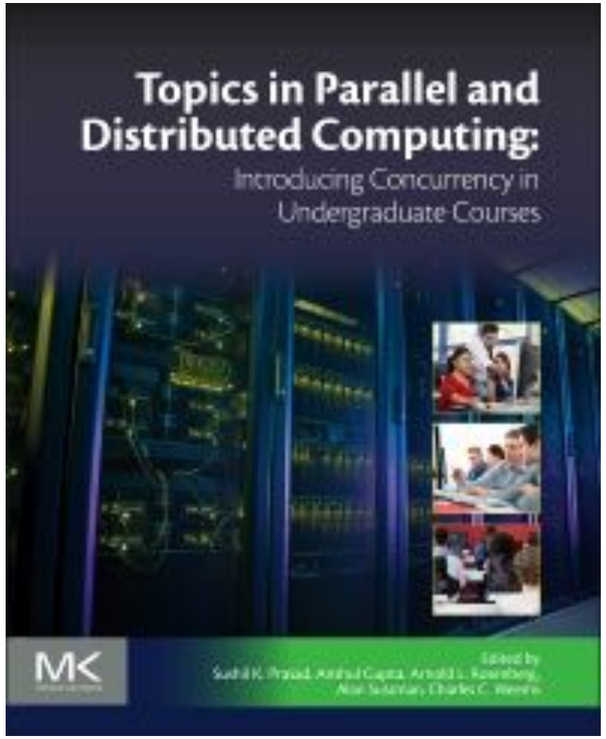
USED RESOURCES

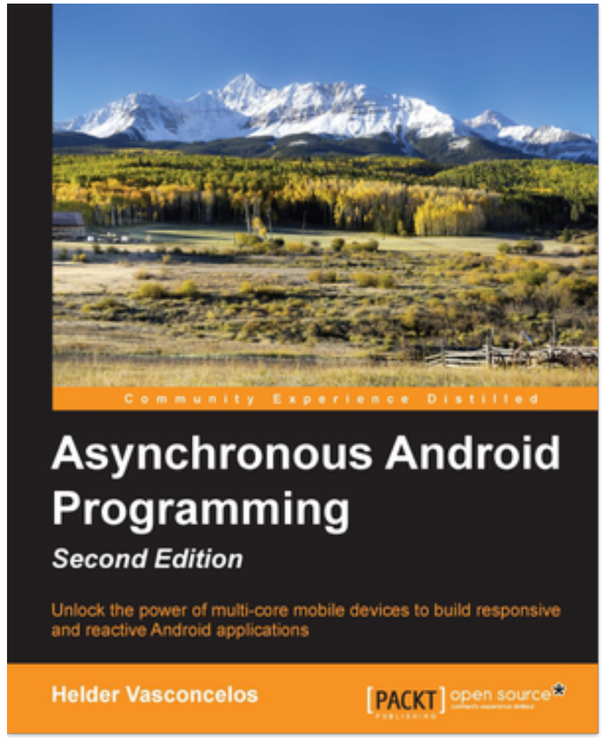
Some slides in this lecture were adopted from lectures by

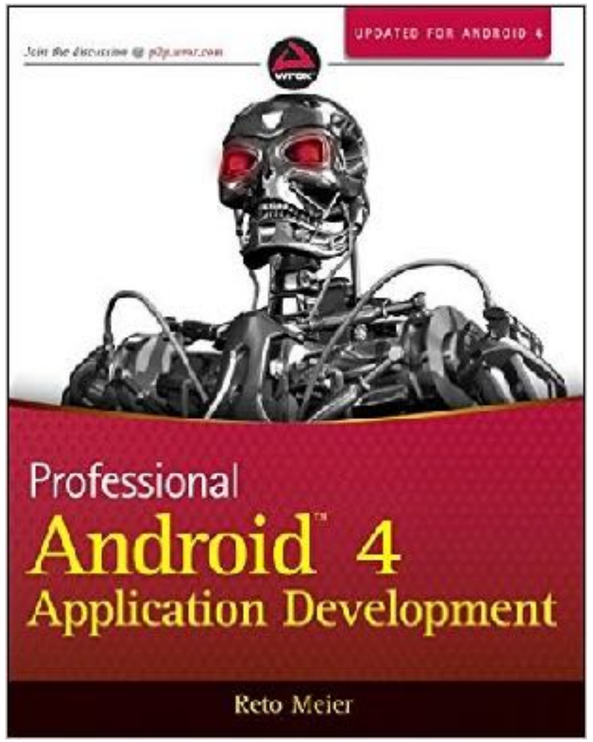
Elisa Gonzalez Boix, Wolfgang De Meuter & Tom Van Cutsem

BOOKS









LOT'S OF EXPLANATION WITH THE CODE EXAMPLES

INTERNET RESOURCES

https://developer.android.com/training/building-connectivity.html

Easy to follow tutorials ! However, concurrency and synchronicity is not always explained well!

https://developer.android.com/guide/index.html

Read the API’s they are really good !

https://www.udacity.com/course/new-android-fundamentals--ud851

Well explained video lectures !