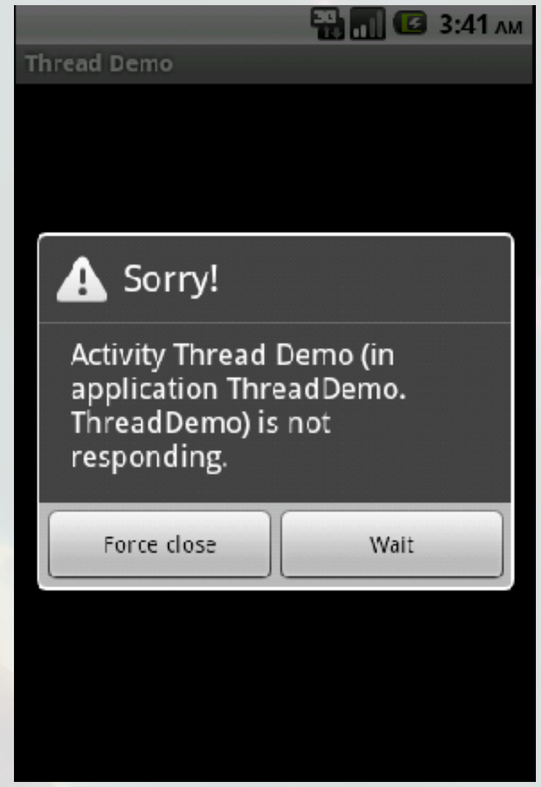
SCIENCES SORBONNE UNIVERSITÉ

Asynchronous Tasks

F. Kordon - Sorbonne Université - CC2018

Renault@lrde.epita.fr

Technical Considerations



In an Application, only the UI Thread can update the GUI

- Must **NOT** be blocked

For long processing, 2 options

- Build a new thread that will perform computation
  - **You have to communicate with UI thread if you want to display the result**
  - **may be hard**
  - **Not in this chapter**
- Use asynchronous tasks
  - **Dedicated component for performing background operation with GUI updates**

Asynchronous Tasks

Ease manipulation of the UI thread

- Background operation
- Builtin callbacks to update the GUI
  - **Useful for progress bar for instance**

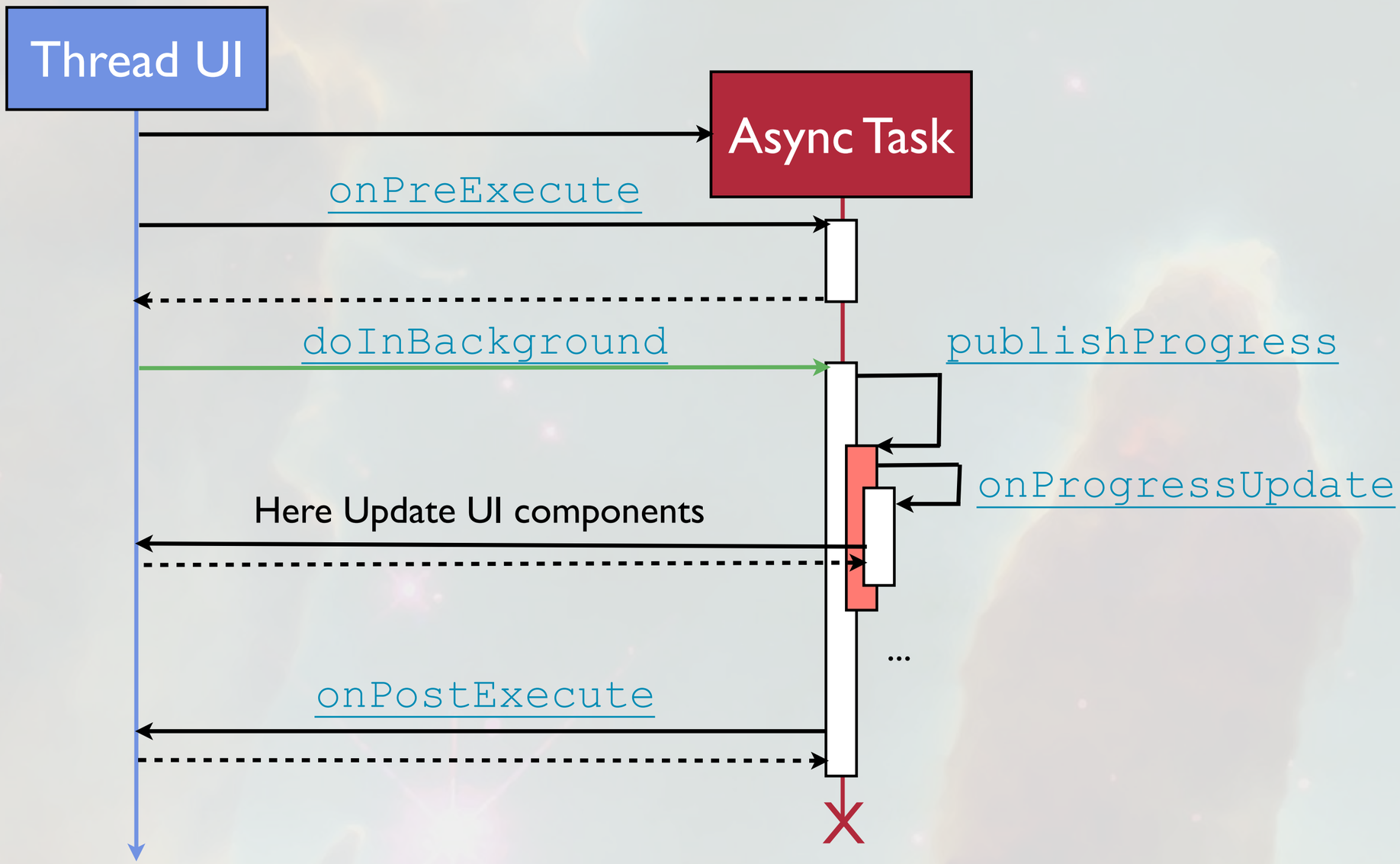
MUST NEVER REPLACE THREADS

- Should not be used for more than few seconds operations

Use three parameters

- Param:
  - **The type of the parameter used for the computation**
- Progress:
  - **The unit used to notify a progress**
- Results:
  - **The type of the result of an asynchronous task**

Lifecycle


Thread UI
Async Task
onPreExecute
doInBackground
publishProgress
onProgressUpdate
Here Update UI components
...
onPostExecute

AsyncTask and ProgressBar (1/2)


```
public class BigCompute
                    extends AsyncTask<Void, Integer, Void> {
  Context mContext;
  ProgressBar mProgressBar;

  BigCompute(Context c, ProgressBar p){
      mContext = c;
      mProgressBar = p;
  }

  @Override
  protected void onPreExecute() {
      Toast.makeText(mContext, "OnPreExecute",
                     Toast.LENGTH_SHORT).show();
      super.onPreExecute();
  }
  @Override
  protected void onPostExecute(Void aVoid) {
      super.onPostExecute(aVoid);
  }
```

AsyncTask and ProgressBar (2/2)


```java
@Override
protected void onProgressUpdate(Integer... values) {
    super.onProgressUpdate(values);
    mProgressBar.setProgress(values[0]);
}
```

```java
@Override
protected Void doInBackground(Void... params) {
    for (int progress = 0; progress < 100; ++progress){
        try{
            Thread.currentThread().sleep(1000);
        }catch (Exception e){}
        publishProgress(progress);
    }
    return null;
}
```

Building an AsyncTask

Instantiation

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Button b = (Button) findViewById(R.id.button_launch);
  b.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        BigCompute bc = new BigCompute(getApplicationContext(),
              (ProgressBar)findViewById(R.id.progressBar));
        bc.execute();
      }
 });
}
```

AsyncTask and Rotation

AsyncTask does not support rotation as-is

Possible workarounds

- Define the asynchronous task in a Fragment an Apply a setRetainedInstance
- Associate AsyncTasks to Services
- Relaunch the AsyncTask

No best solutions

Summary

Asynchronous Tasks

- Easy way to release UI Thread
- Already predefined handlers
- Not for more than few seconds computing
- Well suited for mixing with ProgressBar

Mix bad with rotation

- Even if some solution exist

Should never replace threads