Department of Information Engineering, CUHK
MScIE – 2nd Semester, 2015/16

IEMS 5722
Mobile Network Programming and Distributed Server Architecture

Lecture 4
HTTP Networking in Android

Lecturer: Albert C. M. Au Yeung

4th February, 2016

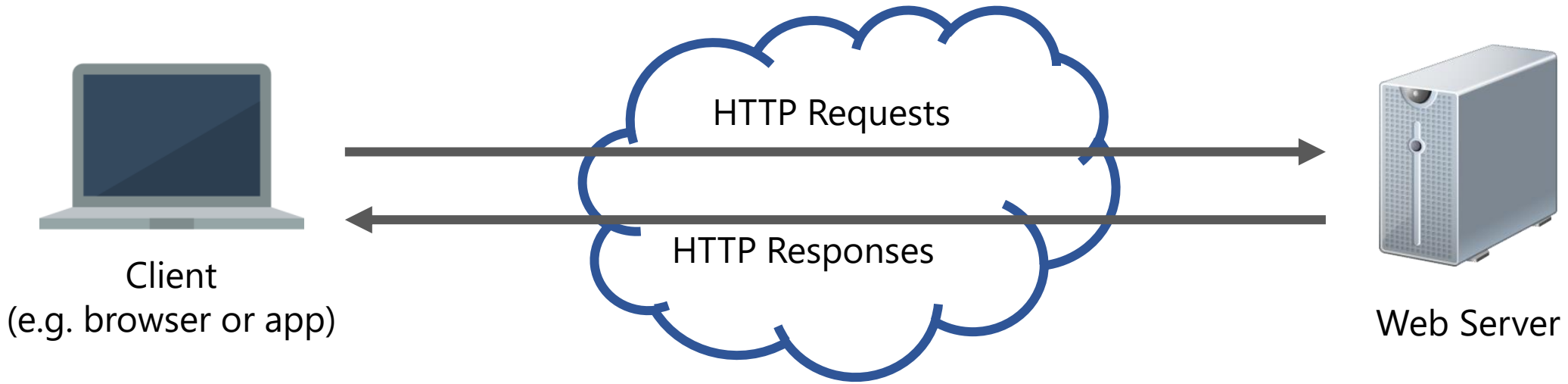
Online Applications

Examples?

- CUHK Website (http://www.cuhk.edu.hk/)
- Facebook Website (http://www.facebook.com/)
- HKO's RSS Feed (http://rss.weather.gov.hk/)
- Instagram App
- Twitter REST API
- …

Online Applications

- All the above services make use of the HTTP protocol

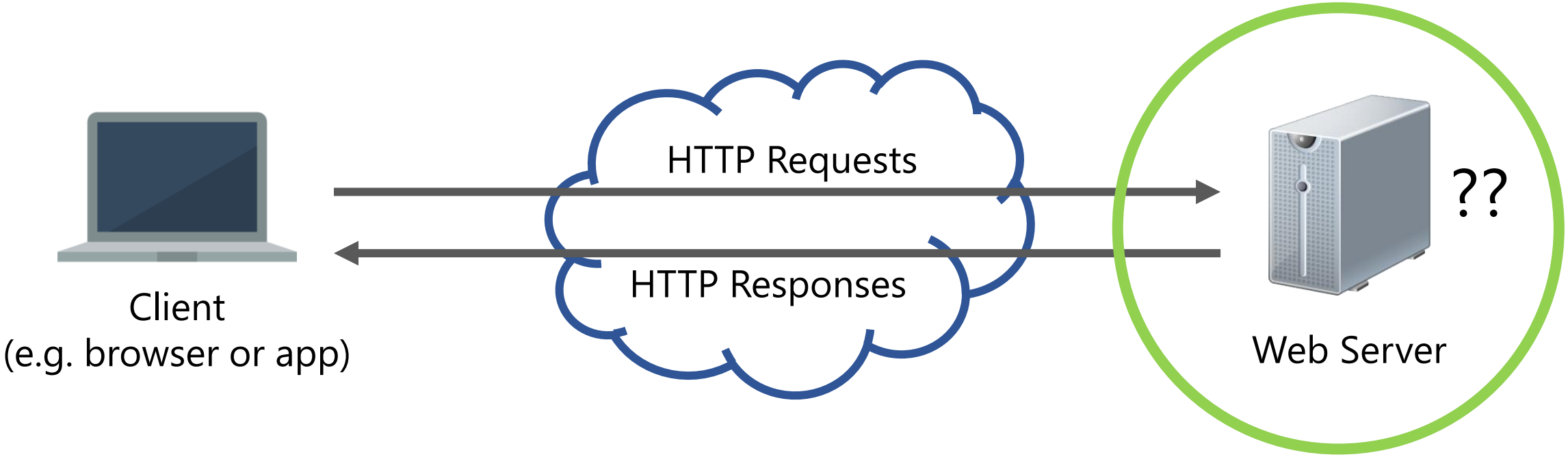


- Let's study more about HTTP first!

The Hypertext Transfer Protocol
HTTP

HTTP

- What is HTTP (Hypertext Transfer Protocol)?
- What happen between a request is made and a response is received?



HTTP

- Tim Berners-Lee, credited as the inventor of the World Wide Web, created the original HTTP and HTML in 1990 at CERN
- For combining the Internet and hypertext

Tim Berners-Lee

The first Web server

HTTP

- An application protocol for transferring hypertext and other file formats over the Internet
- Current widely used version: HTTP/1.1 (standardized in 1997)
- HTTP/2 specification was published as RFC 7540 in May 2015
- Client (e.g. Web browser) sends an HTTP request to a URL
- Server prepares and returns the requested resources

HTTP Requests

An HTTP request has the following components

- URL - the unique identifier of the online resource
- Method/Verb – the action of the request (e.g. GET something?)
- HTTP Version – the version of the protocol you are using
- Headers – the metadata of the request
- Body – Data to be sent to the server

HTTP Response

An HTTP response has the following components

- Status Code – indicate whether the request is successful
- HTTP Version - the version of the protocol you are using
- Headers – metadata of the response
- Body – data of the resource requested

URL

Uniform Resource Locator (URL)

- A specific type of URI (Uniform resource identifier)
- It implies the means to access a resource
- Syntax of a URL:

scheme **://** **[user:password@]** **domain** **:** **port** **/** **path** **?** **query_string** **#** **fragment_id**

- scheme: http or https
- [user:password@]: Usually not required
- domain: The name of the server
- port: 80 for http, 443 for https
- query_string: For passing parameters in the URL
- fragment_id: Referring to a section in the page

URL

Examples:

- CUHK Homepage
  http://www.cuhk.edu.hk/chinese/index.html
- YouTube Video
  https://www.youtube.com/watch?v=Q93o1yBr-Mc
- Apple Daily
  http://hkm.appledaily.com/list.php?category_guid=4104&category=daily
- Instagram API
  https://api.instagram.com/v1/users/self/feed?access_token=ACCESS-TOKEN

HTTP Request Methods

Indicate the desired action to be performed on the resource identified by the URL

- GET – retrieves data from the server
- HEAD – asks for a response same as GET, but without the body
- POST – asks the server to accept data enclosed in the request and apply it to the resource
- PUT – asks the server to store the data under the supplied URL
- Other methods: DELETE, TRACE, OPTIONS, CONNECT, PATCH

HTTP Request Methods

An example of GET:

https://www.youtube.com/watch?v=Q93o1yBr-Mc

- Retrieve a YouTube video page providing the value of the parameter v
- It has no effect on the resource to be retrieved, it simply retrieves a copy of the resource

"v=Q93olyBr-Mc" is the **query string**

HTTP Request Methods

Query String

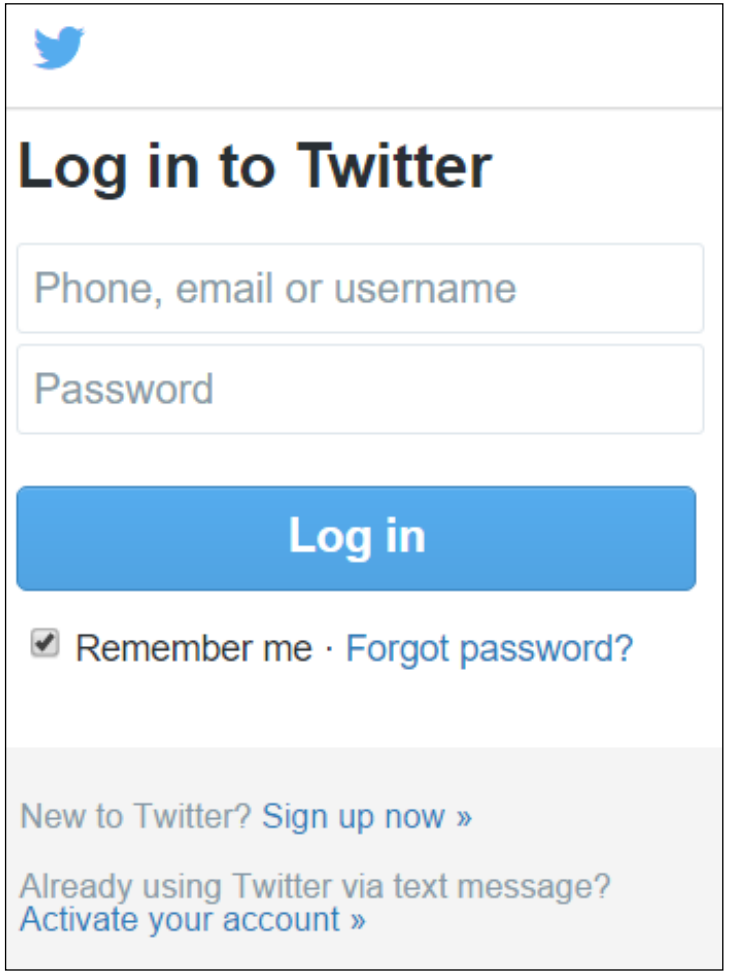
- Each parameter and its value are specified by **name**=**value**
- Parameters are separated by ampersand "**&**"
- The maximum amount of information that can be passed to the server using the query string depends on the maximum length allowed for an URL
  (The limits of different browsers are different, usually at about 64K characters)
- NOT for passing sensitive data (e.g. password)

HTTP Request Methods

An example of POST:

https://twitter.com/login

- After filling in the user name and password and clicking on the “Log in” button, the data will be sent to Twitter’s server using the “POST” method
- Usually used for submitting a form (e.g. online forms, leaving comments, etc.)



HTTP Request Methods

Recall that HTTP is a text protocol (i.e. everything sent using HTTP are assumed to be characters)

If you want to send files (binary data), you need to encode the binary data first before sending

In an HTML form, set enctype="multipart/form-data"
(see next slide)

HTTP Request Methods

```
<form method="post" enctype="multipart/form-data">
    <input type="text" name="name">
    <input type="file" name="file">
    <input type="submit" value="Send!">
</form>
```

Setting enctype="multipart/form-data" tells the server that the data are split into multiple parts, one for each file, plus one for the textual data in the form body.

Ref: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/Sending_and_retrieving_form_data

HTTP Headers

Headers contain metadata about the request/response, such as:

- Identity of the client
- Type of the content (e.g. plain text, HTML, CSS, image)
- Encoding of the content (e.g. ASCII, utf-8)
- Expiry date/time of the content
- Cookies
- …

For a list of HTTP request and response header fields, see:
https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

Inspecting HTTP Requests and Responses

Use the developer's tools in Firefox or Chrome:

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console

View: Options: Preserve log Disable cache

Filter All XHR Script Style Images Media Fonts Documents WebSockets Other Hide data URLs

Name
Path

index.html
/chinese

Headers Preview Response Cookies Timing

General

Remote Address: 137.189.11.73:80
Request URL: http://www.cuhk.edu.hk/chinese/index.html
Request Method: GET
Status Code: 200 OK

Response Headers view source

Accept-Ranges: bytes
Connection: Keep-Alive
Content-Type: text/html
Date: Sat, 15 Aug 2015 13:06:00 GMT
Keep-Alive: timeout=15, max=76
Server: Apache/2.0.52 (Unix) DAV/2 mod_ssl/2.0.52 OpenSSL/0.9.7d PHP/5.0.2
Transfer-Encoding: chunked

Request Headers view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-TW,zh;q=0.8,en-US;q=0.6,en;q=0.4
Cache-Control: max-age=0
Connection: keep-alive
Cookie: _ga=GA1.3.1345156499.1435416105; epws1-11000-PORTAL-PSJSESSIONID=3BvTVLZQ7phsLCFQKKft3vBk4dq1FCTL!-1390053081; __utmt=1; __utma=179935542.1879684639.1435279387.1435279387.1439643948.2; __utmb=179935542.2.10.1439643948; __utmc=179935542; __utmz=179935542.1435279387.1.1.utmcsr=google|utmccn=(organic)|utmcmd=organic|utmctr=(not%20provided)
Host: www.cuhk.edu.hk

Example: CUHK Homepage

Name
Path

- index.html /chinese
- css?family=Robot... fonts.googleapis.c...
- bootstrap.min.css /chinese/css
- main.css?20150901 /chinese/css
- google-analytics.js /chinese/js
- homepage.css

Headers | Preview | Response | Cookies | Timing

General

Remote Address: 27.126.208.30:80
Request URL: http://www.cuhk.edu.hk/chinese/index.html
Request Method: GET
Status Code: 200 OK

Response Headers view source

Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html
Date: Sun, 20 Sep 2015 06:59:29 GMT
Server: G2 server
Transfer-Encoding: chunked
X-CDN: BYPASS-BYPASS-0-0

IP Address and port number of the Web server

URL of the resource to be retrieved

The method (verb) of this request

Example: CUHK Homepage

Headers of the request

Name
Path

index.html
/chinese

css?family=Roboto:700,...
fonts.googleapis.com

bootstrap.min.css
/chinese/css

main.css?20150901
/chinese/css

google-analytics.js
/chinese/js

homepage.css
/chinese/css

dc.js
stats.g.doubleclick.net

60 requests | 280 KB transferre...

Headers Preview Response Cookies Timing

Server: G2 server
Transfer-Encoding: chunked
X-CDN: BYPASS-BYPASS-0-0

Request Headers view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: zh-TW,zh;q=0.8,en-US;q=0.6,en;q=0.4
Cache-Control: max-age=0
Connection: keep-alive
Cookie: epws1-11000-PORTAL-PSJSESSIONID=3BvTVLZQ7phsLCFQKKft3vBk4dq1FCTL!-1390053081; sFXQVYrJNCHnrh24fJQJJXsLkf1pG179!-780447180; resource_url=https://onepass.cuhk.edu.hk/fedgLLBj-v3O31who-; appName=BB1; _ga=GA1.3.1345156499.1435416105; visid_nxg=CCNZ/lUAAAAAGNQ5VcY3LcrZ+RXkvGPz9f+5TYftDro38VtQwPvL58/ukGlKVgXhViQi+EE=; __utmt=1; __utma=179935542.183.4; __utmb=179935542.1.10.1442732333; __utmc=179935542; __utmz=179935542.1442381471.3.2rganic|utmctr=(not%20provided)
Host: www.cuhk.edu.hk
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)

Example: CUHK Homepage

Name
Path

index.html
/chinese

css?family=Roboto:700,...
fonts.googleapis.com

bootstrap.min.css
/chinese/css

main.css?20150901
/chinese/css

google-analytics.js
/chinese/js

homepage.css
/chinese/css

dc.js
stats.g.doubleclick.net

Headers Preview Response Cookies Timing

General

Remote Address: 27.126.208.30:80
Request URL: http://www.cuhk.edu.hk/chinese/index.html
Request Method: GET
Status Code: 200 OK

Response Headers view source

Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html
Date: Sun, 20 Sep 2015 06:59:29 GMT
Server: G2 server
Transfer-Encoding: chunked
X-CDN: BYPASS-BYPASS-0-0

Request Headers view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.
Accept-Encoding: gzip, deflate, sdch

Headers of the response

Example: CUHK Homepage

Content of the response

× Headers Preview Response Cookies Timing

```
<!DOCTYPE html>
<html lang="zh-Hant-HK">
  <head>

    <title>香港中文大學</title>
    <meta name="DESCRIPTION" content="香港中文大學是一所研究型綜合大學，提供多類學士、碩士和博士課程。"
    <meta name="KEYWORDS" content="香港中文大學, 中大, 香港大學, 香港, cu, 香港大學, 學士, 碩士,  博士
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="x-ua-compatible" content="IE=edge" >

    <link rel="apple-touch-icon" href="/chinese/images/fav-icons/apple-touch-icon.png">
    <link rel="icon" type="image/png" href="/chinese/images/fav-icons/favicon-32x32.png" sizes="3
    <link rel="icon" type="image/png" href="/chinese/images/fav-icons/android-chrome-192x192.png"
    <link rel="icon" type="image/png" href="/chinese/images/fav-icons/favicon-96x96.png" sizes="9
    <link rel="icon" type="image/png" href="/chinese/images/fav-icons/favicon-16x16.png" sizes="1
    <link rel="manifest" href="/chinese/images/fav-icons/manifest.json">
    <meta name="msapplication-TileColor" content="#f4dfb4">
    <meta name="msapplication-TileImage" content="/chinese/images/fav-icons/mstile-144x144.png">
```

More on HTTP Headers

HTTP headers are sets of key-value pairs (field names and values)

Some of the request header "keys":

- Accept: the preferred format of the resource
  (e.g. text/html, application/json, application/xml)
- Accept-Language: the preferred language of the resource
  (e.g. zh-TW, zh-CN, en-US)
- User-Agent: the type of browser or device
  (e.g. indicate whether the client is on a PC or on a mobile)

More on HTTP Headers

Some of the response header "keys":

- Content-Length: length of the content of the resource
- Content-Type: format of the resource
  (e.g. text/html)
- Last-Modified: the time when the resource was last changed
- Server: The name of the Web server serving the resource

For a comprehensive list of header fields:
https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

HTTP Status Code

HTTP Status code is included in a HTTP response to indicate the outcome of an HTTP request

The different categories of HTTP status codes:

- 1XX: Informational
- 2XX: Successful
- 3XX: Redirection
- 4XX: Client-side error
- 5XX: Server-side error

HTTP Status Code

Examples of HTTP status codes

- 200: OK
  Everything is OK, results should be in the response
- 301: Moved Permanently
  The client should send request from the URL provided instead
- 403: Forbidden
  The client is not authorised to access the resource
- 404: Not Found
  The resource cannot be found
- 500: Internal Server Error
  Some problem with your server application

References

- Introduction to HTTP Basics https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Stateless-ness

HTTP is a stateless protocol

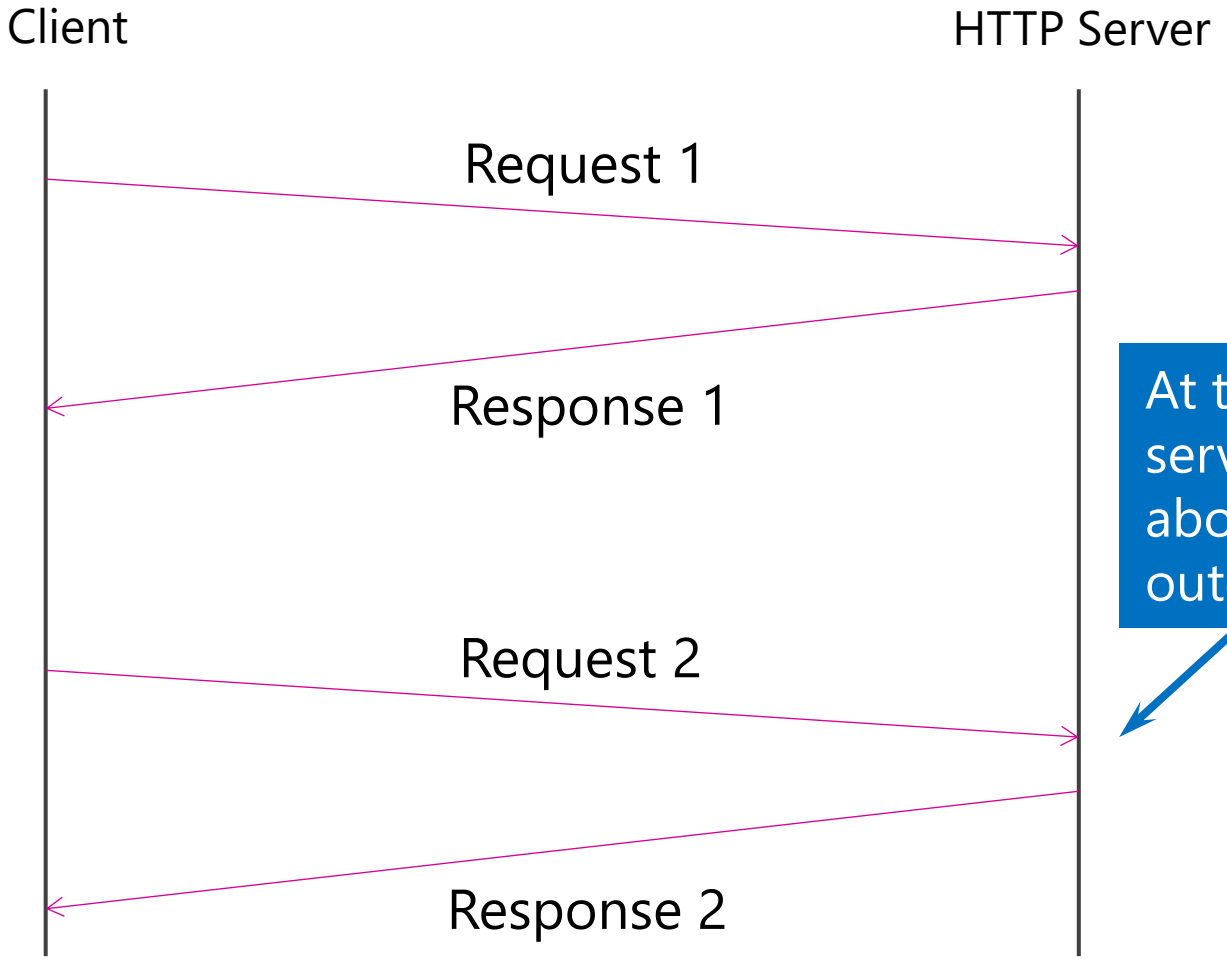
- The server does not retain information about clients between requests
- Each request is considered independent
- No session information stored on the server-side

(See illustration on the next slide)

Stateless-ness

In order to let the server know that the client has done something before, the client has to include some information in the request (header or content)

Example:
- The client has already logged in
- The client has filled in a form but some fields are missing



At this point, the server knows nothing about the content and outcome of Request 1.

Stateless-ness

Discussion

- What are the pros and cons of stateless protocols?
- Does this help scalability?

Using HTTP in Android

Data Communication using HTTP

How can we perform HTTP requests to a Web server in Android?

- First of all, you need to ask for permission in the AndroidManifest.xml file

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

- Use the HttpURLConnection HTTP client to perform HTTP requests

* Note: some online resources may mention the use the DefaultHTTPClient class, however that class should not be used as it is deprecated.

HttpURLConnection

HttpURLConnection can be used to perform both GET and POST actions

- Check the following link for usage examples
  http://developer.android.com/training/basics/network-ops/connecting.html
- Data is returned in the form of InputStream
- Depending on the data type of the data (e.g. image, text, file, etc.), you need to decode the data into appropriate format

HttpURLConnection

Example

Performing a GET request to http://www.cuhk.edu.hk/

```
InputStream is = null;

try {
    URL url = new URL(“http://www.cuhk.edu.hk/”);
    HttpURLConnection conn = (HttpURLConnection) url.openConnection();
    conn.setReadTimeout(10000); // 10,000 milliseconds
    conn.setConnectTimeout(15000); // 15,000 milliseconds
    conn.setRequestMethod("GET"); // Use the GET method
    conn.setDoInput(true);
    // Starts the query
    conn.connect();
    int response = conn.getResponseCode(); // This will be 200 if successful
    is = conn.getInputStream();

    // Convert the InputStream into a string
    String data = “”;
    String line;
    BufferedReader br = new BufferedReader(new InputStreamReader(is));
    while ((line = br.readLine()) != null) {
        results += line;
    }

} finally {
    if (is != null) {
        is.close(); // Close InputStream after using it.
    }
}
```

HttpURLConnection

Example

Performing a POST request to http://www.example.com/submit_form

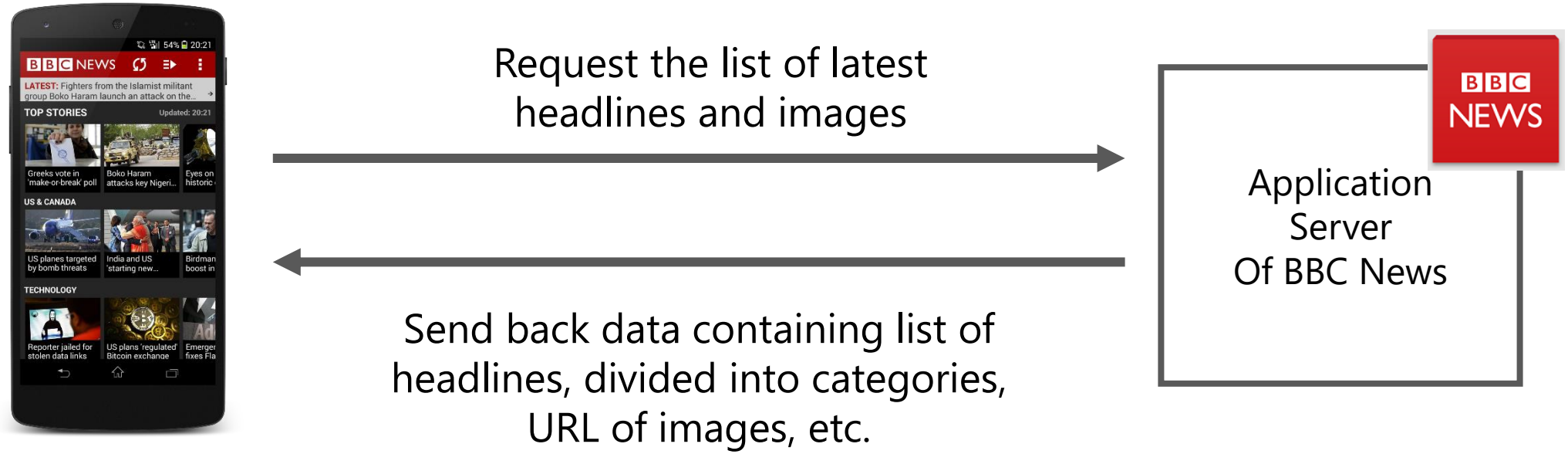
```
URL url_object = new URL("http://www.example.com/submit_form");
HttpURLConnection conn = (HttpURLConnection)url_object.openConnection();
conn.setReadTimeout(15000);
conn.setConnectTimeout(15000);
conn.setRequestMethod("POST");
conn.setDoInput(true);
conn.setDoOutput(true);

OutputStream os = conn.getOutputStream();
BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os, "UTF-8"));

Uri.Builder builder = new Uri.Builder();
for (int i = 0; i < para_names.size(); i++) {
    builder.appendQueryParameter(para_names.get(i), para_values.get(i));
}
String query = builder.build().getEncodedQuery();

writer.write(query);
writer.flush();
writer.close();
os.close();

int responseCode = conn.getResponseCode();
if (responseCode == HttpsURLConnection.HTTP_OK) {
    ...
}
```

Exchanging Data between Server & Client

Exchange Data

In app programming, very often we are not requesting Web pages. Instead, we request data from the server, such as:

- List of latest news (news app)
- History of conversation (instant messaging app)



Exchange Data

In order to exchange structured data, we need to have a common data format

Common data exchange formats include:

1. XML (Extensible Markup Language)
   - Using different tags (e.g. <title></title>) to give meanings to the data
   - May result in a significant increase in the length of the data
2. JSON (JavaScript Object Notation)
   - JavaScript objects encoded as strings, can handle several data types such as strings, numbers, Booleans and arrays.
   - More compact compared to XML, still easy to read by human

JSON

Below is an example of data coded in JSON format

```
{
    “status”: “100”,
    “message”: “Request processed without error.”,
    “data”: [
        {
            “title”: “News title 001”,
            “content”: “...”
        },
        {
            “title”: “News title 002”,
            “content”: “...”
        },
        {
            “title”: “News title 003”,
            “content”: “...”
        }
    ]
}
```

Objects are indicated by **{...}**

Arrays are indicated by **[...]**

All values should be enclosed by quotation marks “”

Parsing JSON Data

If you have a string containing encoded JSON data, you can extract data by using the following method:

```
JSONObject json = new JSONObject(json_string);

int status = json.getInt("status");
String message = json.getString("message");

JSONArray array = json.getJSONArray("data");
for (int i = 0; i < array.length(); i++) {
    String title = array.getJSONObject(i).getString("title");
    ...
    ...
}
```

Note: There is a possibility that the string is not well formatted, you have to put these codes inside a try/catch block to catch JSONExceptions

Creating JSON Objects

You can also create a JSONObject and populate it with data

```
JSONObject json = new JSONObject();
json.put("title", "...");
json.put("content", "...");

JSONObject json2 = new JSONObject();
json2.put("x", "...");
json2.put("y", "...");

json.put("data", json2);

String json_string = json.toString();
```

Note: There is a possibility that the string is not well formatted, you have to put these codes inside a try/catch block to catch JSONExceptions

JSON for Data Communication

JSON has been used in many APIs for returning structured data

For example, Google Map's Geocoding API serves JSON data

http://maps.googleapis.com/maps/api/geocode/json
?**address=chinese%20university%20of%20hong%20kong**&**sensor=false**

```
{
    "status" : "OK“,
    "results" : [
        {
            "address_components" : [
            {
                "long_name" : "香港中文大學",
                "short_name" : "香港中文大學",
                "types" : ["establishment" ]
            },
...
```

More on Multi-threading in Android

Performing HTTP Requests

Very often, you need to perform network operations in the background, and then display the data fetched from the network to the users

The two rules again:

- You should not block the UI thread
- You should not manipulate UI components from other threads

Performing HTTP Requests

One option is to use Runnable and the View.post() function:

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            text = fetchDatafromNetwork();
            ...
            ...
            text_view.post(new Runnable() {
                public void run() {
                    textview.setText(text);
                }
            });
        }
    }).start();
}
```

Performing HTTP Requests

The down-side of using runnable + view.post:

- Codes become difficult to manage
- Need to design parameter passing, or using a lot of global variables
- Not suitable if you need to update a lot of UI components after the network operation

AsyncTask

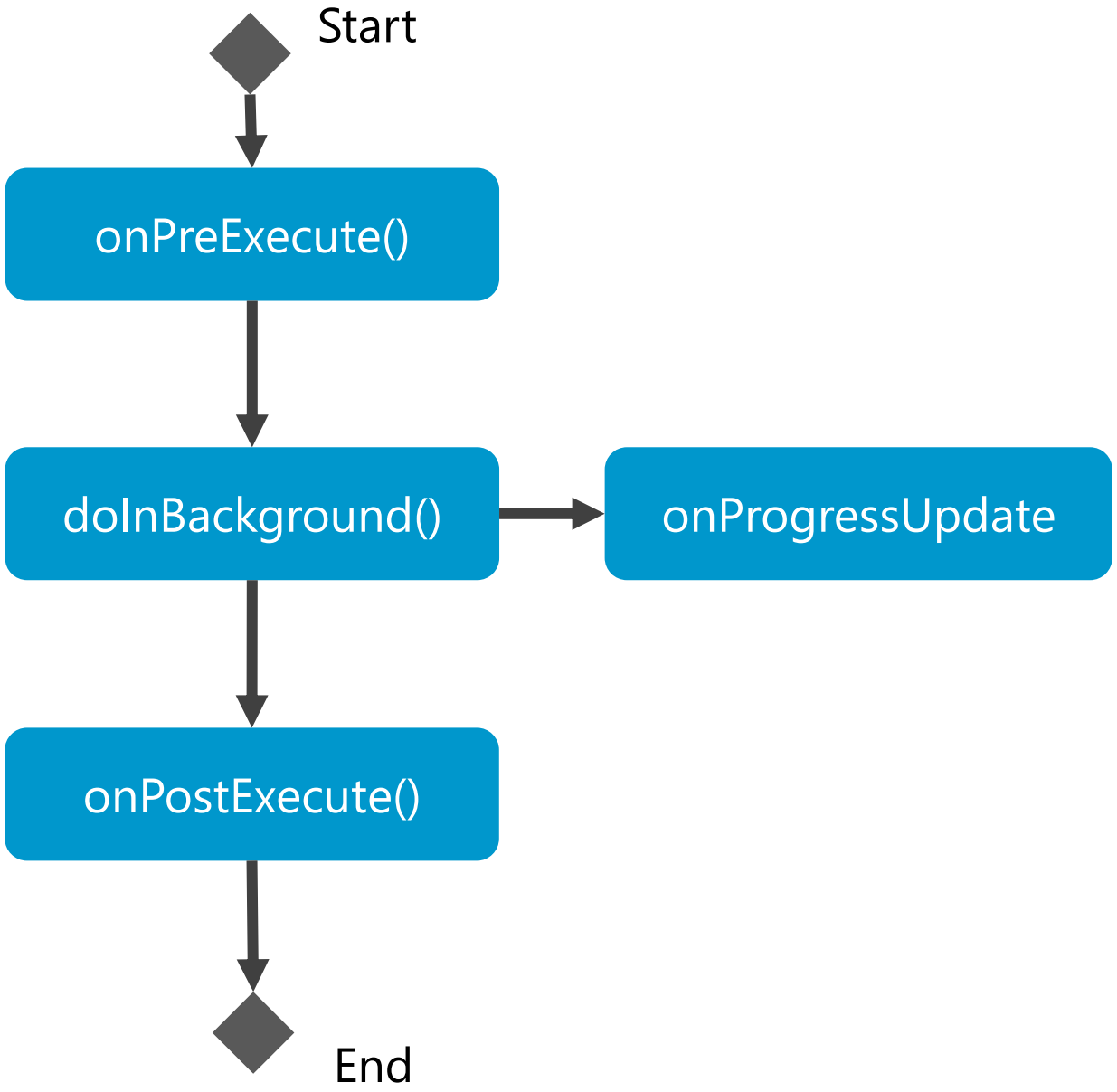
AsyncTask provides a proper and easy-to-use method to perform background operations and manipulate UI components, without worrying about creating threads.

- It allows you to perform **asynchronous** tasks in the background
- It is suitable for **short operations** (e.g. a few seconds)
- It **must be sub-classed** to be used (extends AsyncTask)

AsyncTask

4 Steps in an AsycnTask

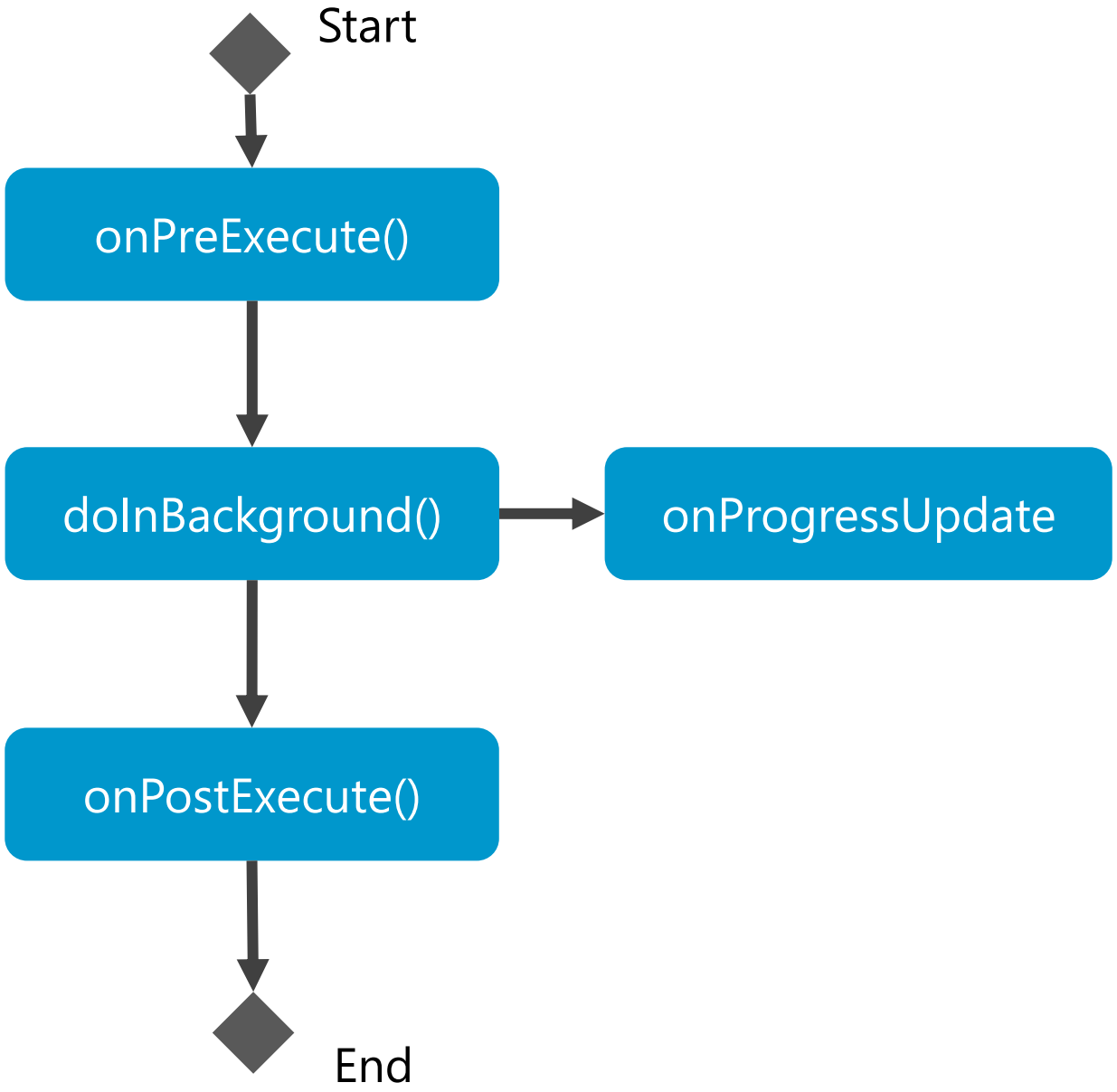
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()



AsyncTask

onPreExecute(): Do preparations before the operation is performed, for example:

- Start displaying a progress dialog (e.g. now loading…, please wait…)
- Initialise variables
- Etc.

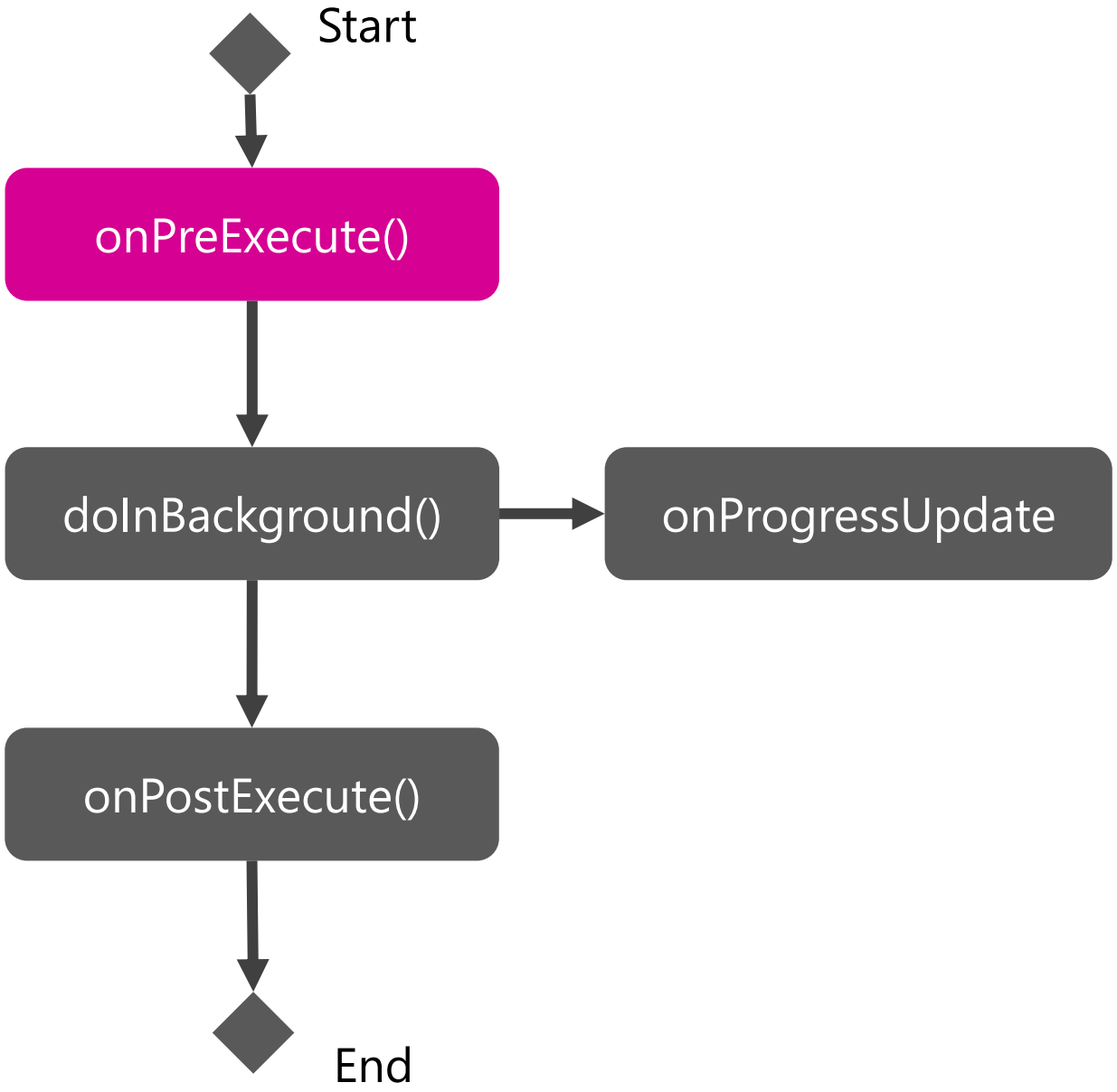


AsyncTask

onPreExecute(): Do preparations before the operation is performed, for example:

- Start displaying a progress dialog (e.g. now loading…, please wait…)
- Initialise variables
- Etc.

NOTE
this function runs on the **UI thread**

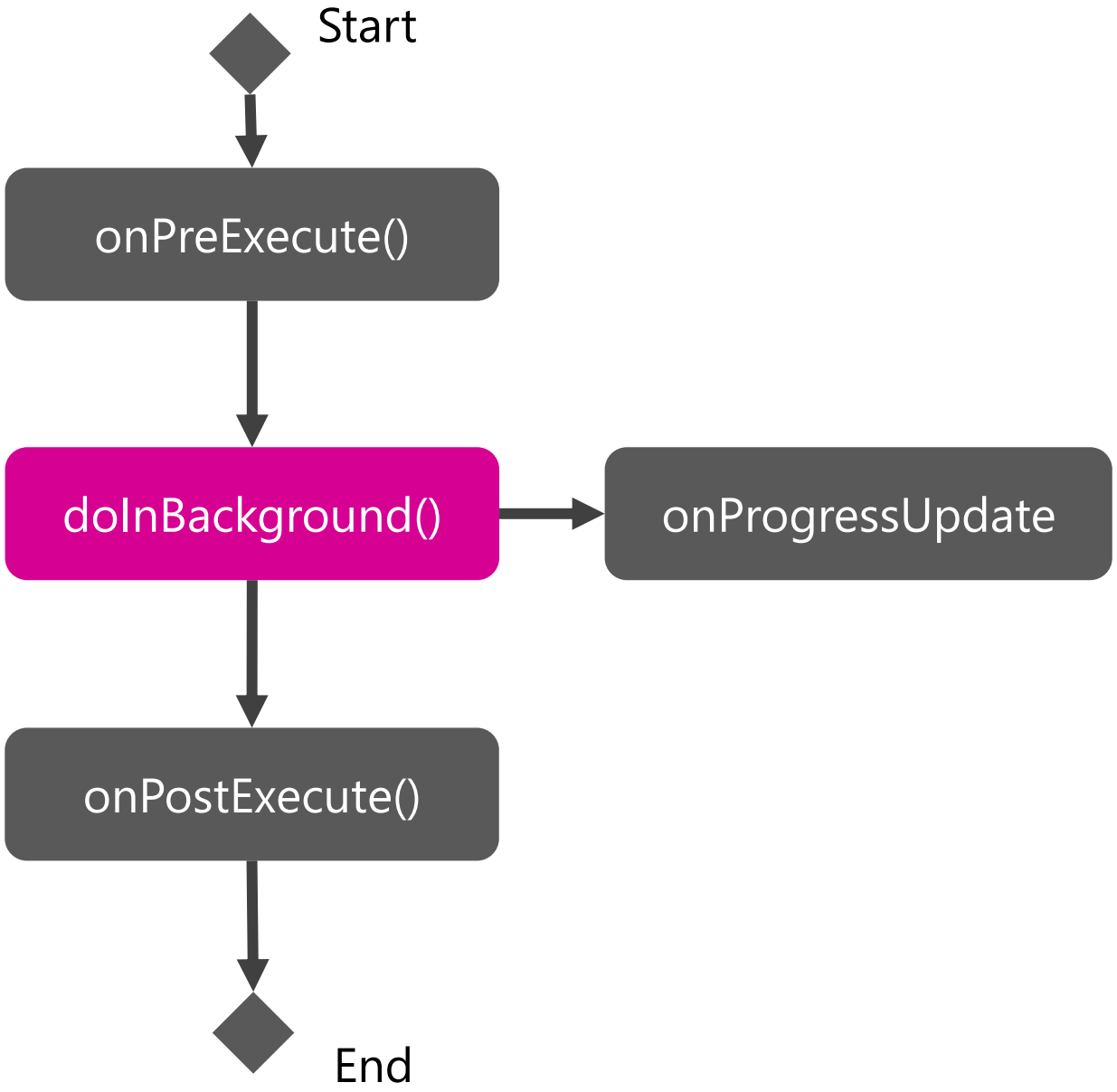


AsyncTask

doInBacktround(): perform the background operation that takes some time, for example:

- Connecting to a server to fetch data
- Performing some heavy operations

NOTE
this function runs on a **new thread**

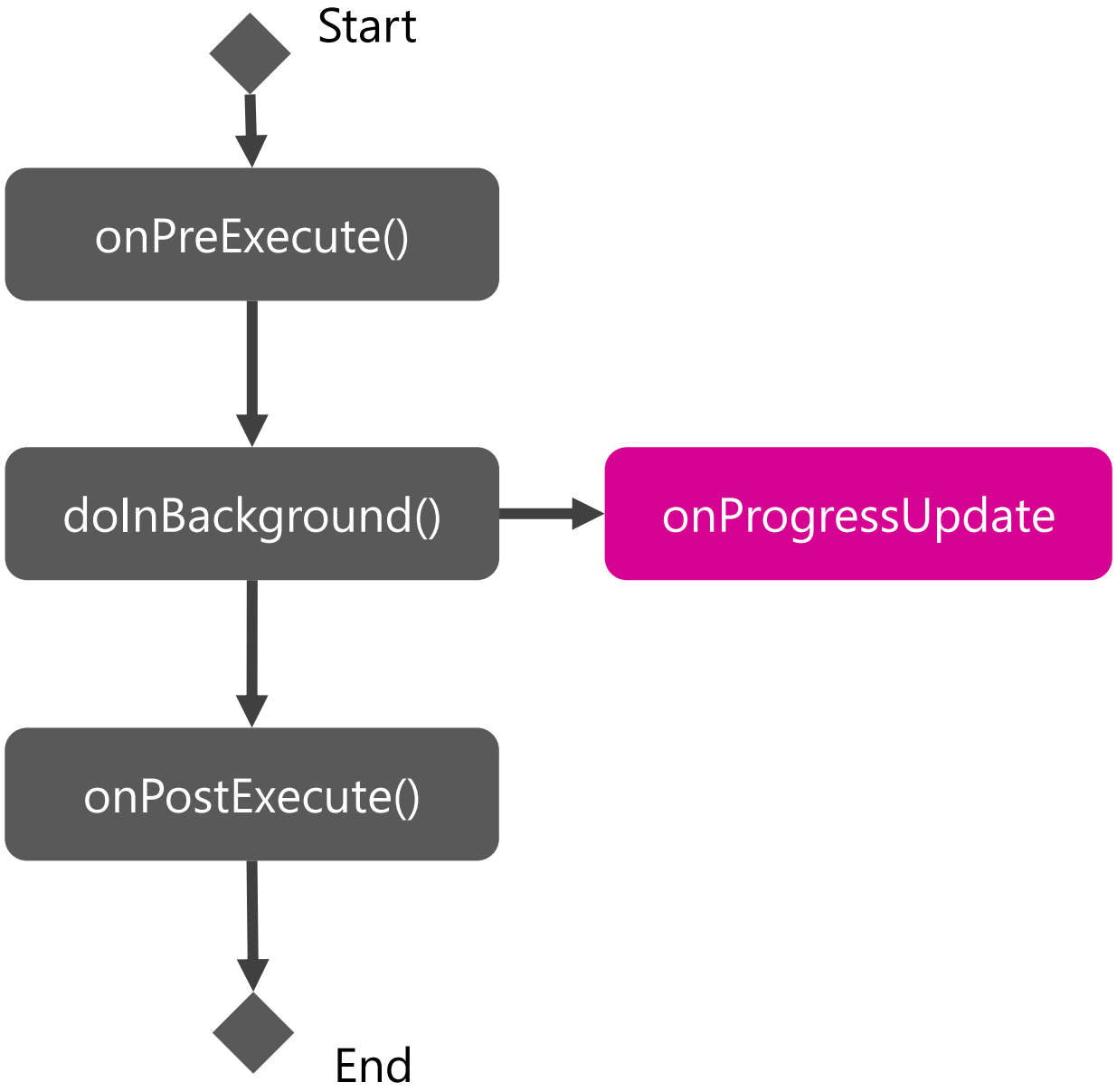


AsyncTask

onProgressUpdate(): when you call publishProgress() in doInBackground(), this function will be called, here you may:

- Update the percentage in the progress dialog
- Show intermediate result to the user

NOTE
this function runs on the **UI thread**

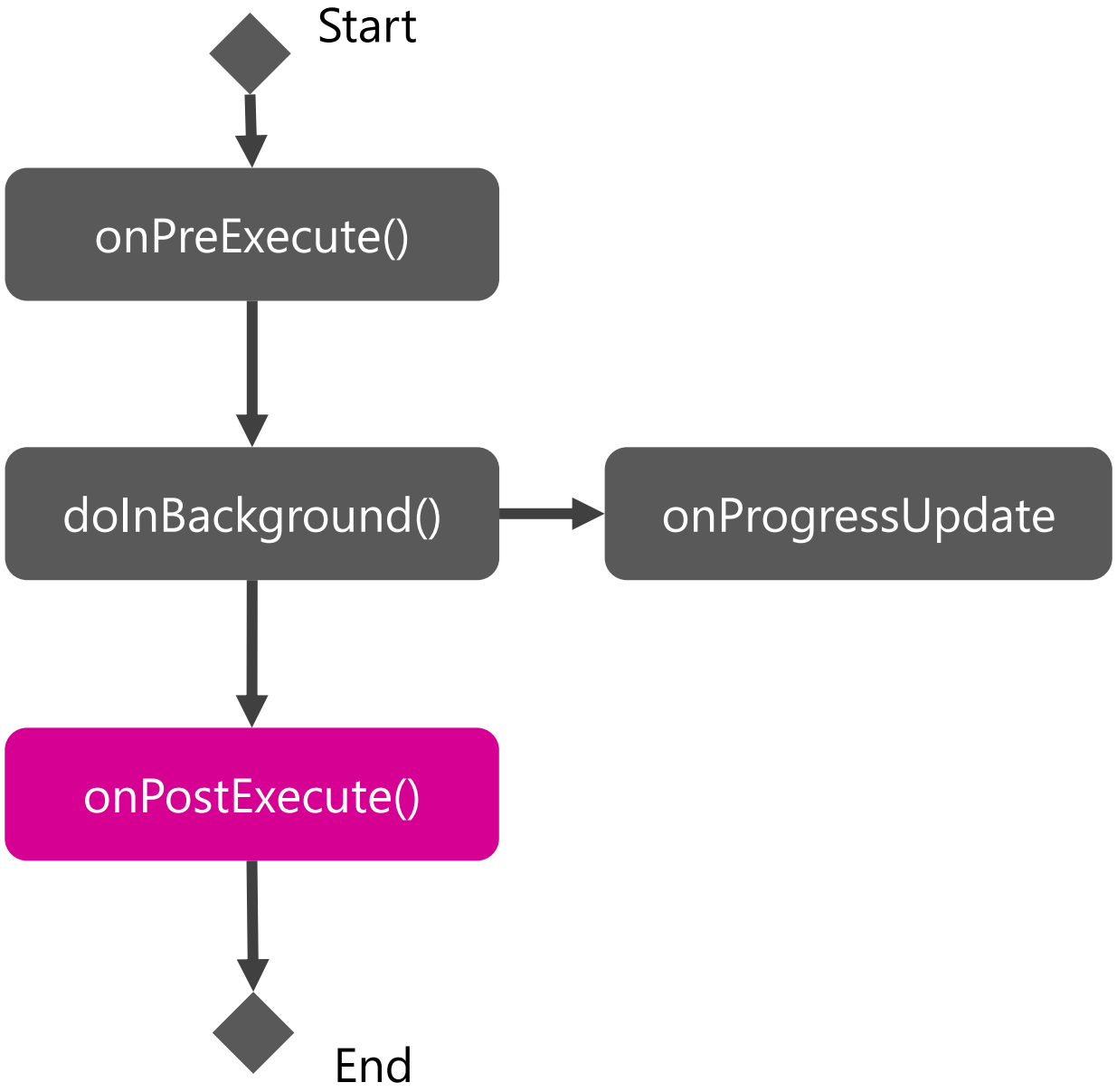


AsyncTask

onPostExecute(): will be called after doInBackground() finishes its job, here we can:

- Update UI components using the data received
- Notify the user that data has been updated (if necessary)

NOTE
this function runs on the **UI thread**



AsyncTask

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            if (isCancelled()) break;
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

AsyncTask

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100)
            if (isCancelled()) break;
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

These are data types of parameters and return types of the functions

Volley

An HTTP library that provides simpler and more efficient method to make networking easier in Android

- Available through the Android Open Source Project
- You need to download the code and include it in your own project
- Similar to AsyncTask, suitable for short operations
- Not suitable for large downloads, as all responses are kept in memory by Volley

Reference: http://developer.android.com/training/volley/index.html

Checking Availability of the Network

Network Availability

Before using the network to carry out data communication, it is a good practice to first check for the availability of the network

- On a mobile phone, the network can be unstable
- The user may have switched off data transmission, or switched to airplane mode
- By checking the availability of the network, you can prompt the user first before you carry out any network operation

Network Availability

To be able to check the state of the network, you need to ask for the following permission in the AndroidManifest.xml file

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

To check whether network is available, you can use:

```
public static boolean isNetworkAvailable(Context context) {
    ConnectivityManager cm = (ConnectivityManager)
    context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo info = cm.getActiveNetworkInfo();
    if (info == null) {
        return false;
    } else {
        return true;
    }
}
```

Network Availability

In addition, you may want to know if the user is connected using Wi-Fi. This is useful when you plan to transmit large amount of data

The object **NetworkInfo** returned by the **getActiveNetworkInfo()** function contains this information

You can check the type of connection by:

```
...
NetworkInfo info = cm.getActiveNetworkInfo();
int ntype = info.getType();
if (ntype == ConnectivityManager. TYPE_WIFI) {
    ...
    ...
}
...
```

Loading Multiple Images using HTTP

Loading Images

Consider the case when the data requested from server contains both text and URLs to images

```
{
    “data”: [
        {
            “title”: “News title 001”,
            “content”: “...”,
            “image”: “http://www.myserver.com/images001.jpg”
        },
        {
            “title”: “News title 002”,
            “content”: “...”,
            “image”: “http://www.myserver.com/images002.jpg”
        },
        ...
    ]
}
```

Loading Images

```
{
    “data”: [
        {
            “title”: “News title 001”,
            “content”: “...”,
            “image”: “http://www.myserver.com/images001.jpg”
        },
        {
            “title”: “News title 002”,
            “content”: “...”,
            “image”: “http://www.myserver.com/images002.jpg”
        },
        ...
    ]
}
```

For each of these images, you need to download the image in a new thread, and show the bitmap in an ImageView

Loading Images

Two third-party Android libraries are particularly useful:

1. Universal Image Loader
   https://github.com/nostra13/Android-Universal-Image-Loader
2. Picasso
   http://square.github.io/picasso/

- Both handle downloading the image in a new thread, and update an ImageView on the UI thread
- Handle caching of images
- Transformation of the image before displaying

Next Lecture:
Web and Application Servers

Next Lecture

We will be talking about Web and Application servers in the next lecture, try to get familiar with the following things before that:

- Creating a Ubuntu VM in Amazon AWS
- Some basic Linux commands (e.g. installing new software)
- Python
  - https://docs.python.org/2/tutorial/
  - https://www.youtube.com/playlist?list=PLS1QulWo1RIaJECMeUT4LFwJ-ghgoSH6n

End of Lecture 4