Playing Audio and Video in Android

MediaPlayer

- Android Classes used to play sound and video
  - **MediaPlayer:** Plays sound and video
  - **AudioManager:** plays only audio

- Any Android app can create instance of/use MediaPlayerAPIs to integrate video/audio playback functionality

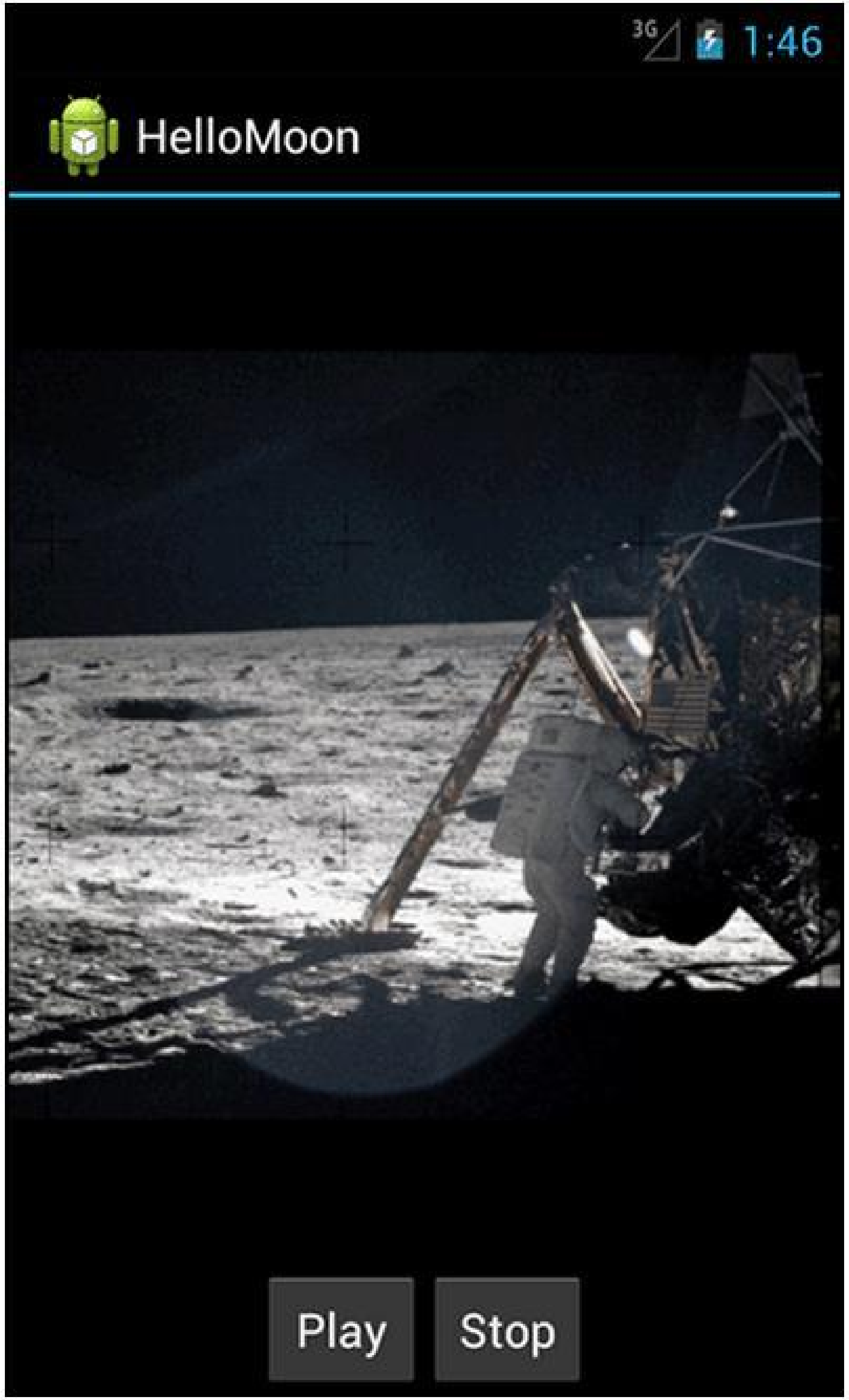
- MediaPlayercan fetch, decode and play audio or video from:
  - Audio/video files stored in app's resource folders (e.g. **res/raw/**folder)
  - External URLs (over the Internet)

MediaPlayer

- MediaPlayer supports:
  - **Streaming network protocols:** RTSP, HTTP streaming
  - **Media Formats:**
    - Audio (MP3, AAC, MIDI, etc),
    - Image (JPEG, GIF, PNG, BMP, etc)
    - Video (MPEG-4, H.263, H.264, H.265 AVC, etc)
- 4 major functions of a Media Player
  - **User interface**, user interaction
  - Handle **Transmission errors**: retransmissions, interleaving
  - **Decompress** audio
  - **Eliminate jitter:** Playback buffer (Pre-download 10-15 secs of music)

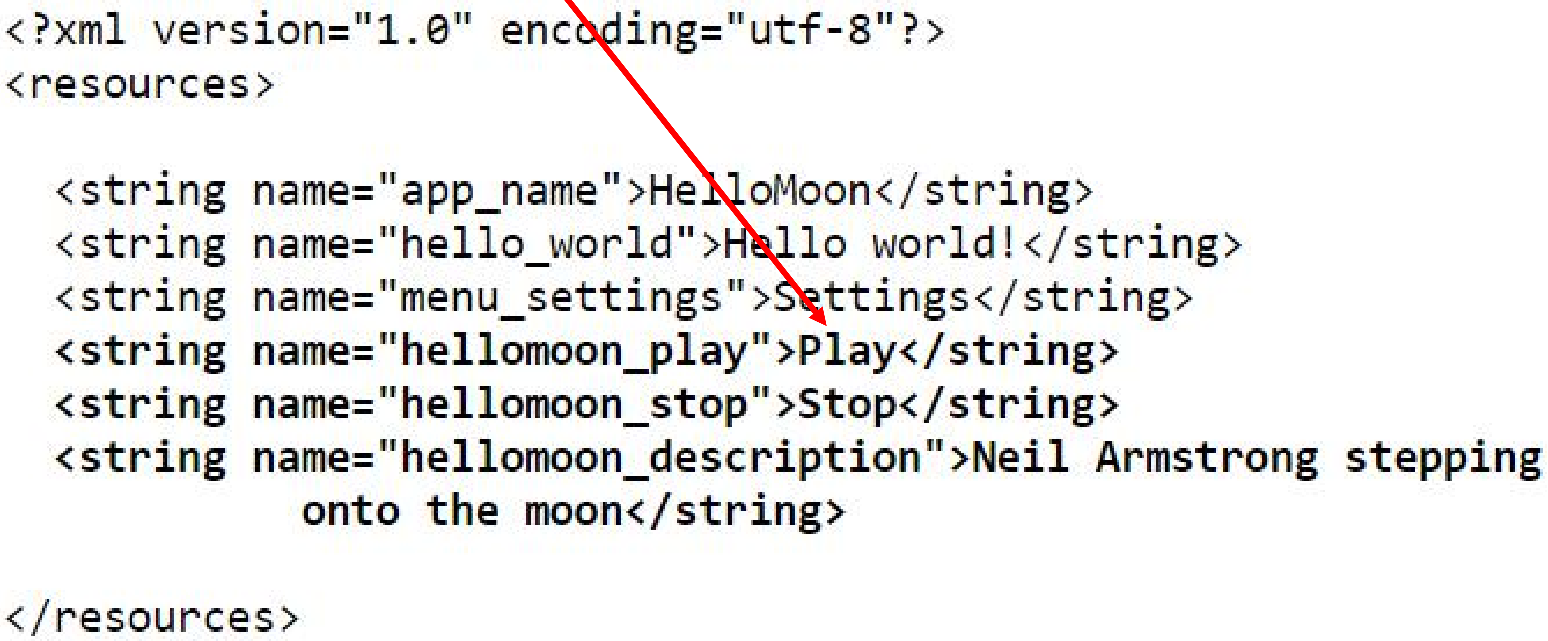
Example: Playing Audio File using MediaPlayer

- Use **MediaPlayer**to play audio file

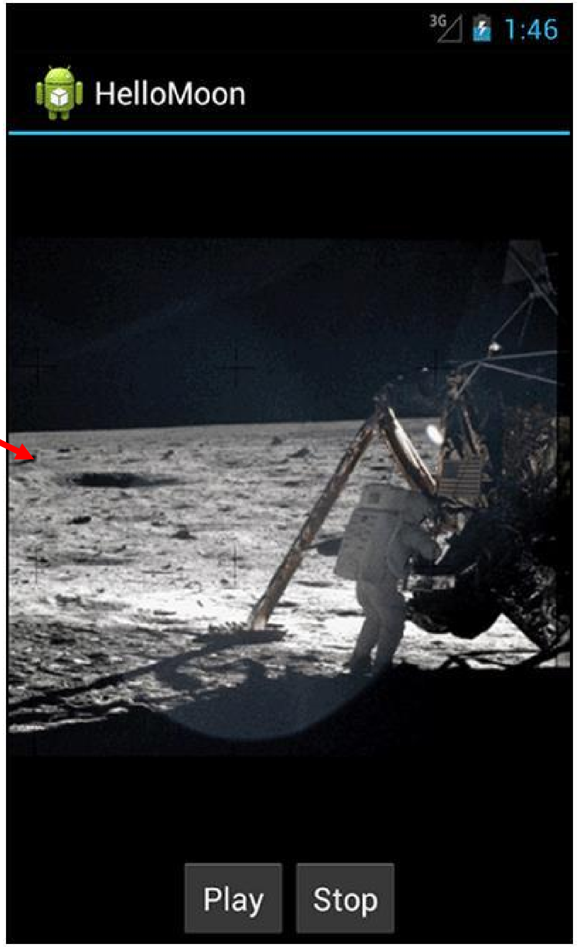


Resources

- Put image **armstrong_on_moon.jpg** in **res/drawable/**folders
- Place audio file to be played back (**one_small_step.wav**) in **res/raw** folder
- Create **strings.xml** file for app
  - Play, Stop, Image description..



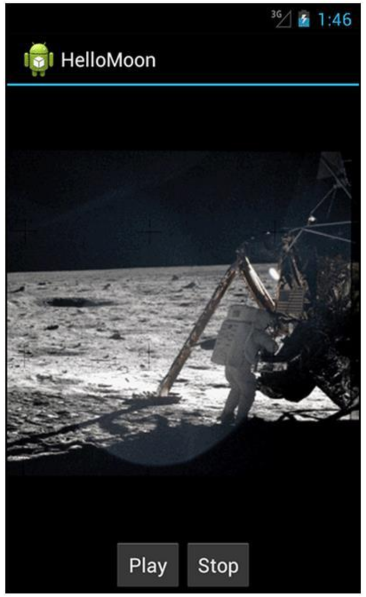
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="app_name">HelloMoon</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="hellomoon_play">Play</string>
  <string name="hellomoon_stop">Stop</string>
  <string name="hellomoon_description">Neil Armstrong stepping
            onto the moon</string>

</resources>
```

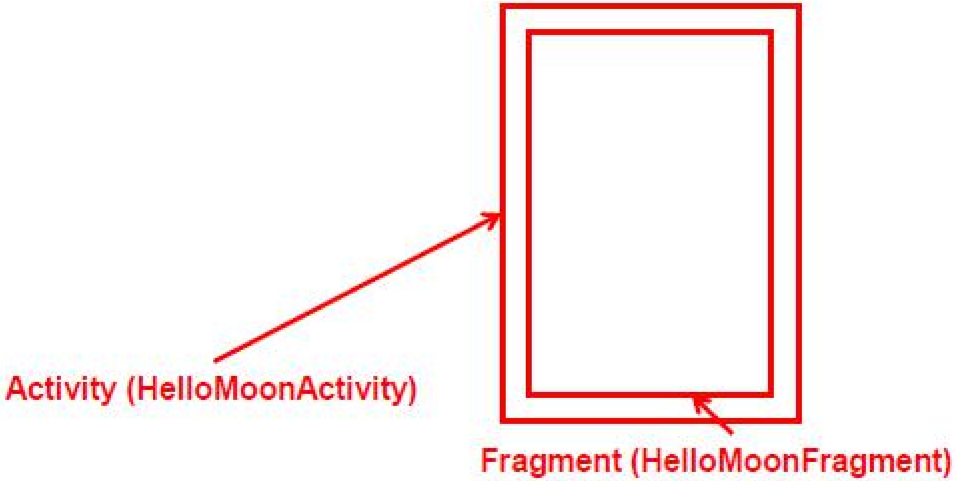


The UI

- 1 activity (**HelloMoonActivity**) that hosts **HelloMoonFragment**
- **AudioPlayer**class will be created to encapsulate **MediaPlayer**

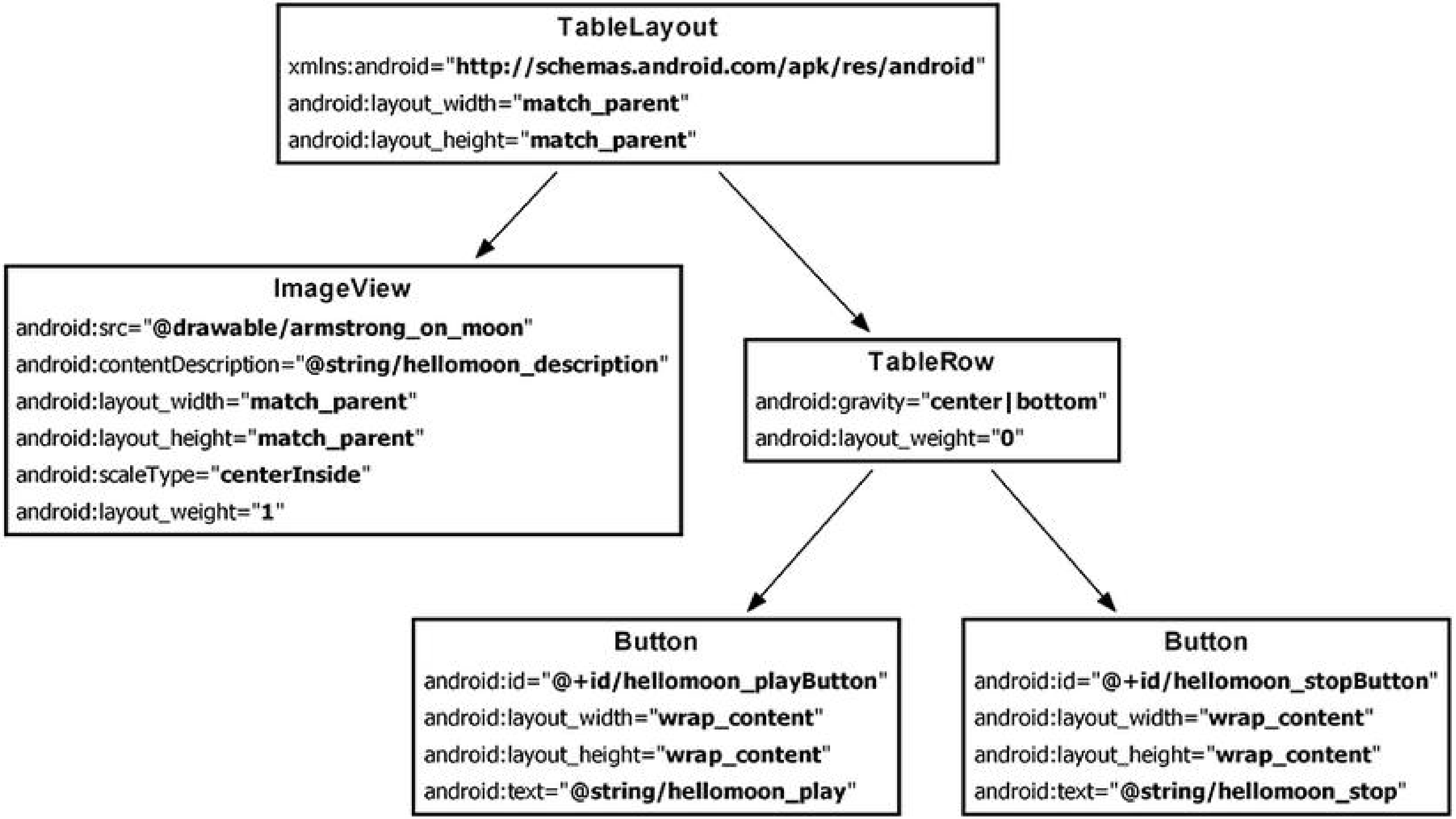
- First set up the rest of the app:
  - Define fragment’s XML layout
  - Create fragment java class
  - Modify the activity (java) and its XML layout to host the fragment




Activity (HelloMoonActivity)

Fragment (HelloMoonFragment)

Defining the Layout for HelloMoonFragment



TableLayout

xmlns:android="**http://schemas.android.com/apk/res/android**"

android:layout_width="**match_parent**"

android:layout_height="**match_parent**"

ImageView

android:src="**@drawable/armstrong_on_moon**"

android:contentDescription="**@string/hellomoon_description**"

android:layout_width="**match_parent**"

android:layout_height="**match_parent**"

android:scaleType="**centerInside**"

android:layout_weight="**1**"

TableRow

android:gravity="**center|bottom**"

android:layout_weight="**0**"

Button

android:id="**@+id/hellomoon_playButton**"

android:layout_width="**wrap_content**"

android:layout_height="**wrap_content**"

android:text="**@string/hellomoon_play**"

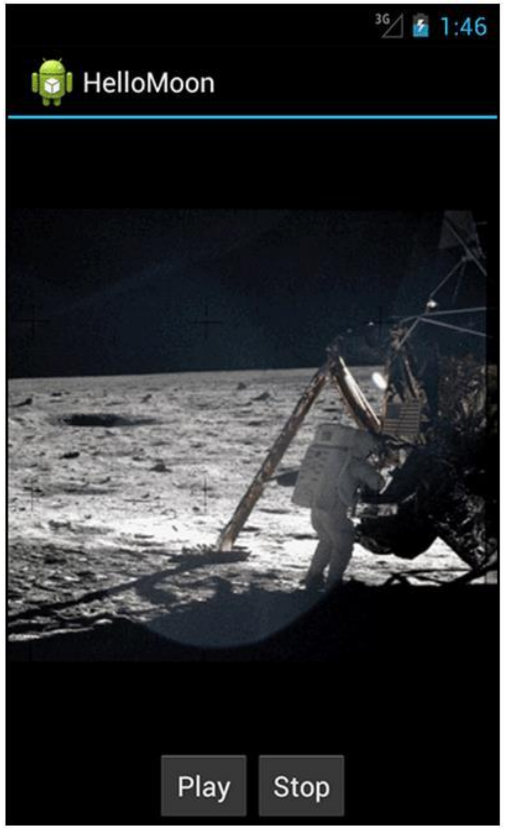
Button

android:id="**@+id/hellomoon_stopButton**"

android:layout_width="**wrap_content**"

android:layout_height="**wrap_content**"

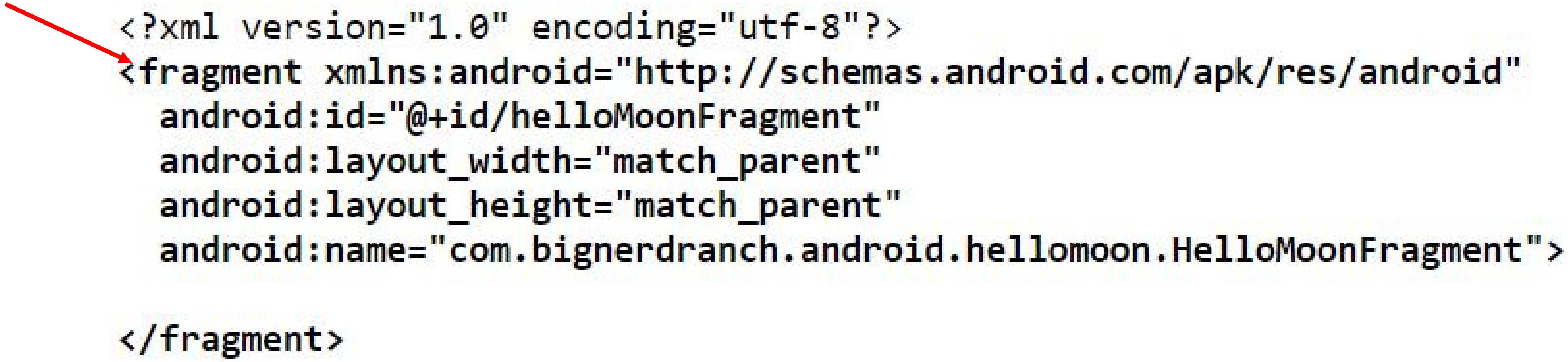
android:text="**@string/hellomoon_stop**"



Define XML for HelloMoon UI (fragment_hello_moon.xml)

Creating a Layout Fragment

- **Layout fragment:** Add fragments to hosting Activity's XML file
- Create activity's XML layout (**activity_hello_moon.xml**)
- **Activity's** XML layout file contains/hosts fragment

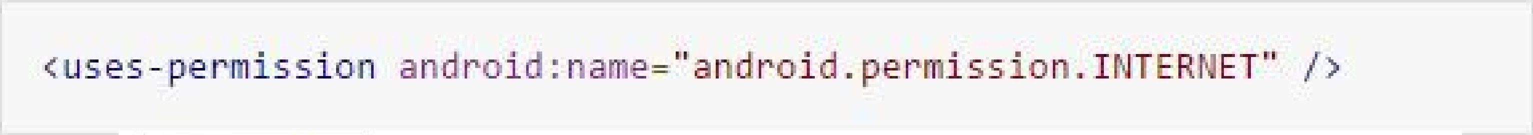


```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/helloMoonFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</fragment>
```

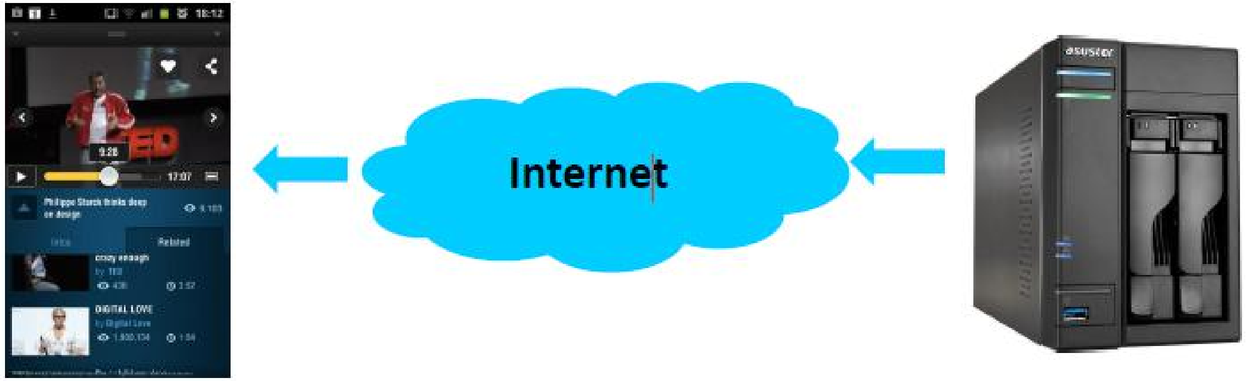
Using Media Player:

Step 1: Request Permission in AndroidManifest or Place video/audio files in res/raw

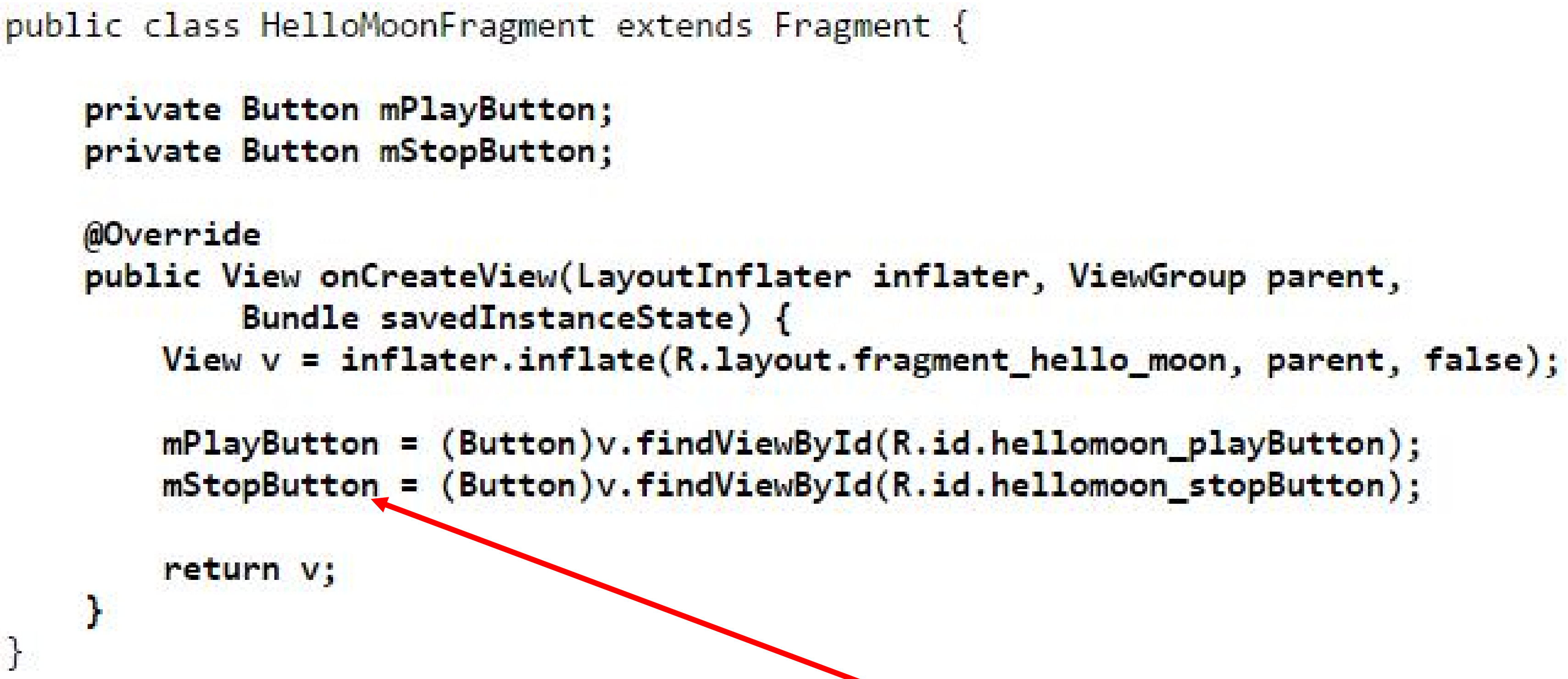
- If streaming video/audio over Internet (network-based content), request network access permission in AndroidManifest.xml:



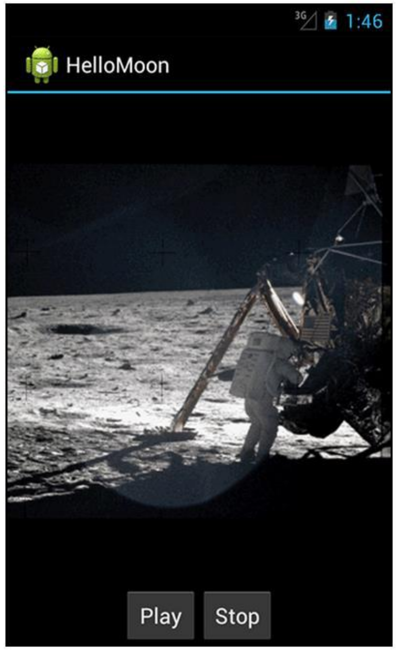
```
<uses-permission android:name="android.permission.INTERNET" />
```



Set up HelloMoonFragment.java



```
public class HelloMoonFragment extends Fragment {

    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);

        return v;
    }
}
```



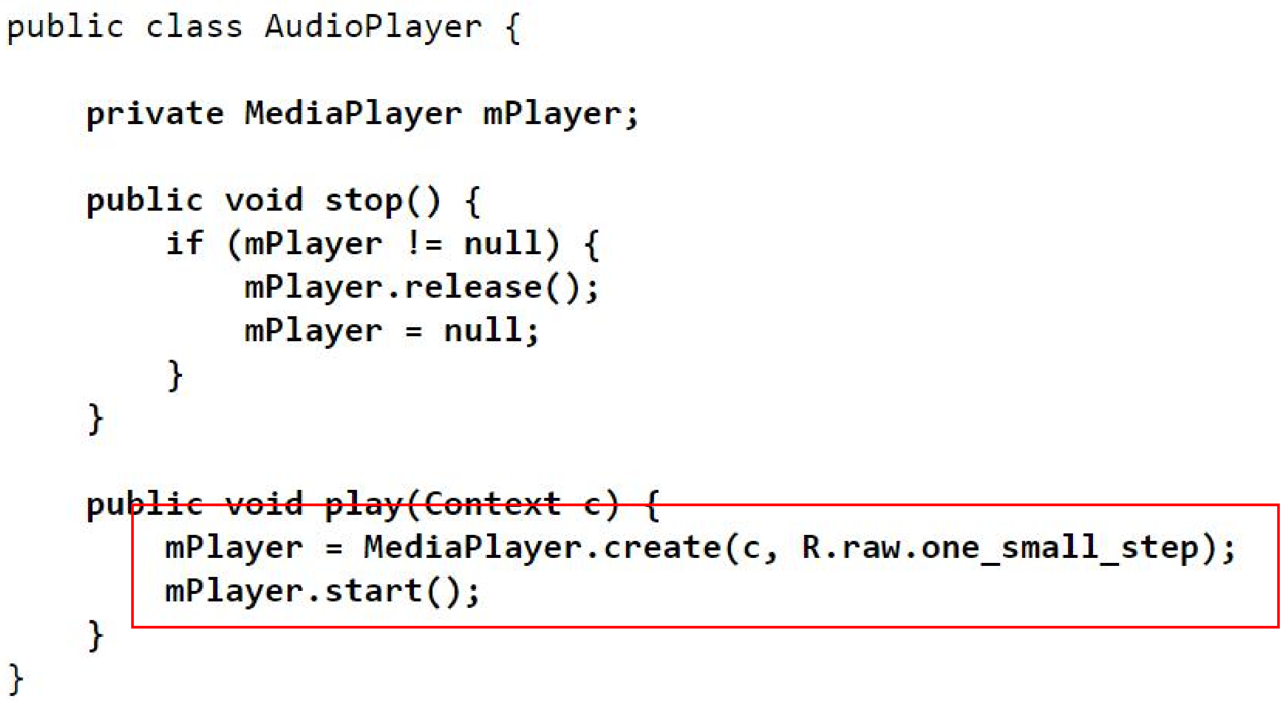
Get handle to Start, Stop buttons

- If playing back local file stored on user’s smartphone, put video/audio files in **res/raw** folder

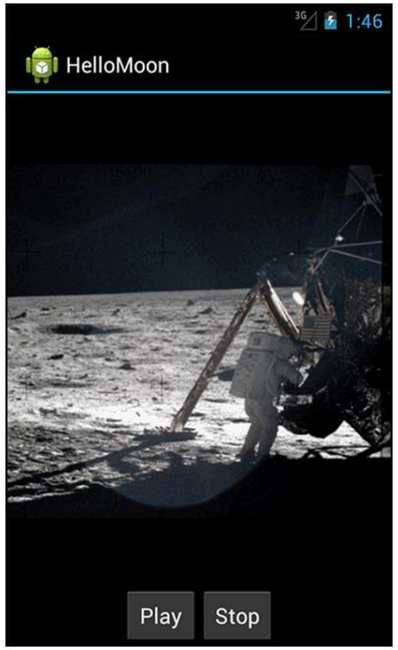
Step 2: Create MediaPlayer Object, Start Player

- To play audio file saved in app’s **res/raw/** directory
- **Note:** Audio file opened by create (e.g. one_small_step.mp3) must be encoded in one of supported media formats

Create AudioPlayer Class encapsulates MediaPlayer

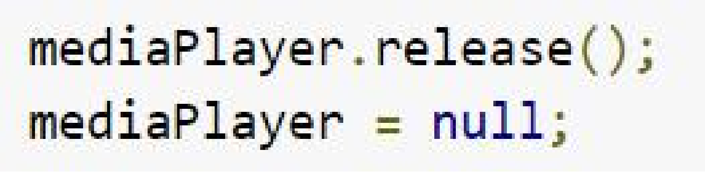


```
public class AudioPlayer {

    private MediaPlayer mPlayer;

    public void stop() {
        if (mPlayer != null) {
            mPlayer.release();
            mPlayer = null;
        }
    }

    public void play(Context c) {
        mPlayer = MediaPlayer.create(c, R.raw.one_small_step);
        mPlayer.start();
    }
}
```



- **Releasing the MediaPlayer**

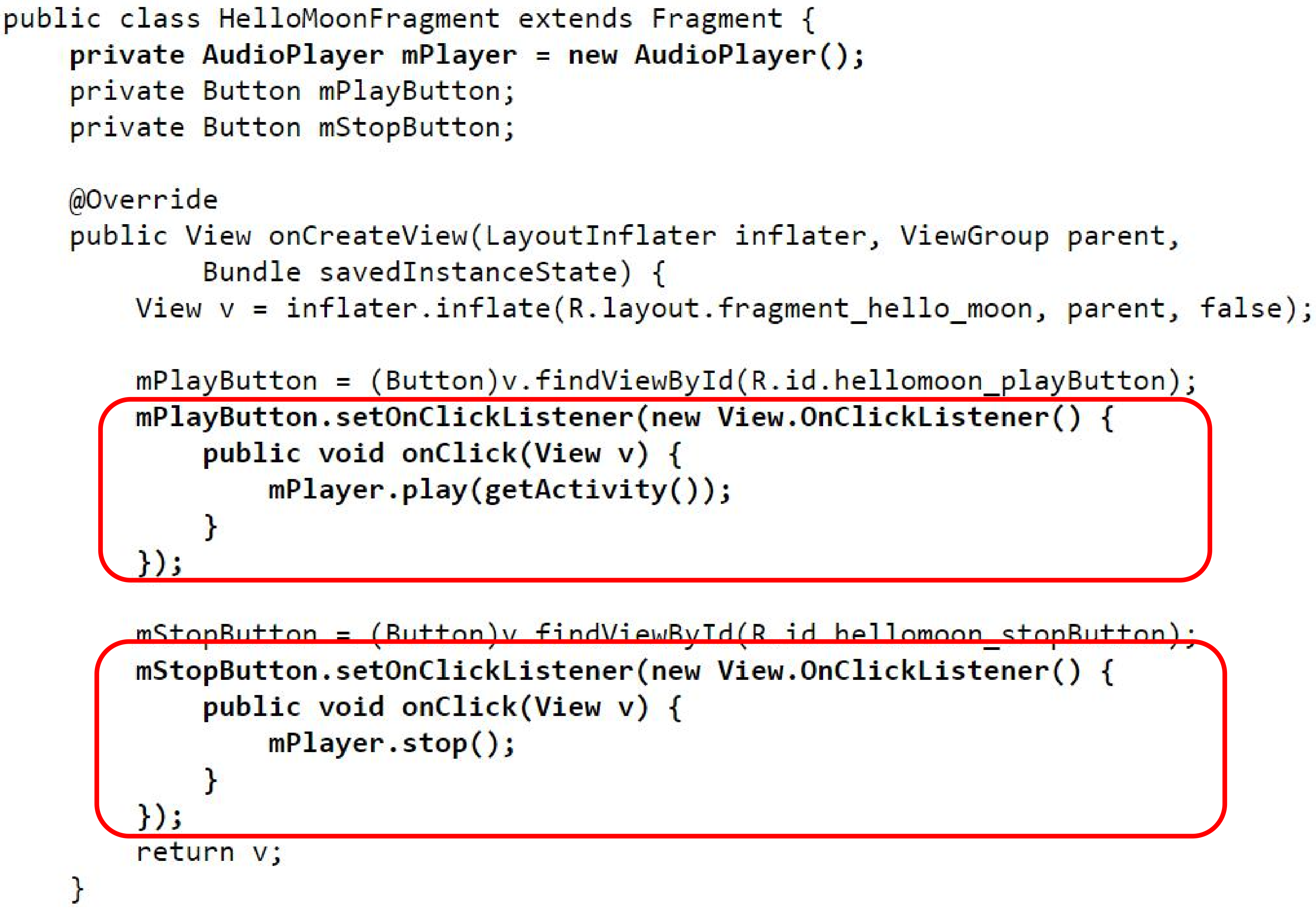
- MediaPlayer can consume valuable system resources
- When done, call **release( )** to free up system resources
- In **onStop( )** or **onDestroy( )** methods, call



```
mediaPlayer.release();
mediaPlayer = null;
```

- **MediaPlayer in a Service:** Can play media (e.g. music) in background while app is not running
  - Start MediaPlayer as service

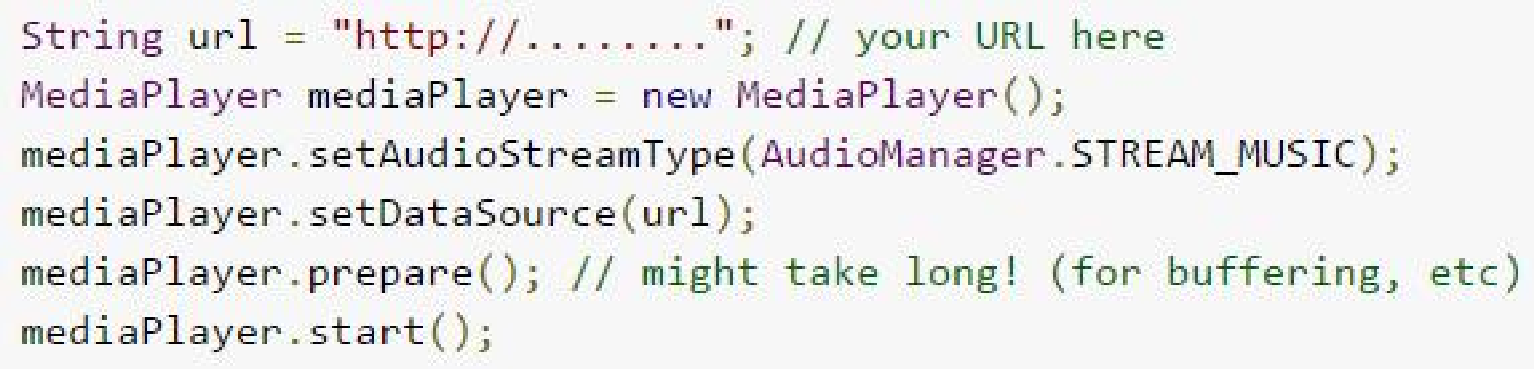
Hook up Play and Stop Buttons



```
public class HelloMoonFragment extends Fragment {
    private AudioPlayer mPlayer = new AudioPlayer();
    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mPlayButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.play(getActivity());
            }
        });

        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.stop();
            }
        });
        return v;
    }
```

Extra: stream audio from internet

- To play audio from remote URL via HTTP streaming over the Internet



```
String url = "http://........"; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

Multimedia Networking: Basic Concepts

Multimedia networking: 3 application types

- Multimedia refers to audio and video. 3 types

1. *streaming, stored*audio, video
   - *streaming:* transmit in batches, begin playout before downloading entire file
   - e.g., YouTube, Netflix, Hulu
   - Streaming Protocol used (e.g. Real Time Streaming Protocol (RTSP), HTTP streaming protocol (DASH))
2. *streaming live* audio, video
   - e.g., live sporting event
3. *conversational* voice/video over IP
   - Requires minimal delays due to interactive nature of human conversations
   - e.g., Skype, RTP/SIP protocols

Live Streaming

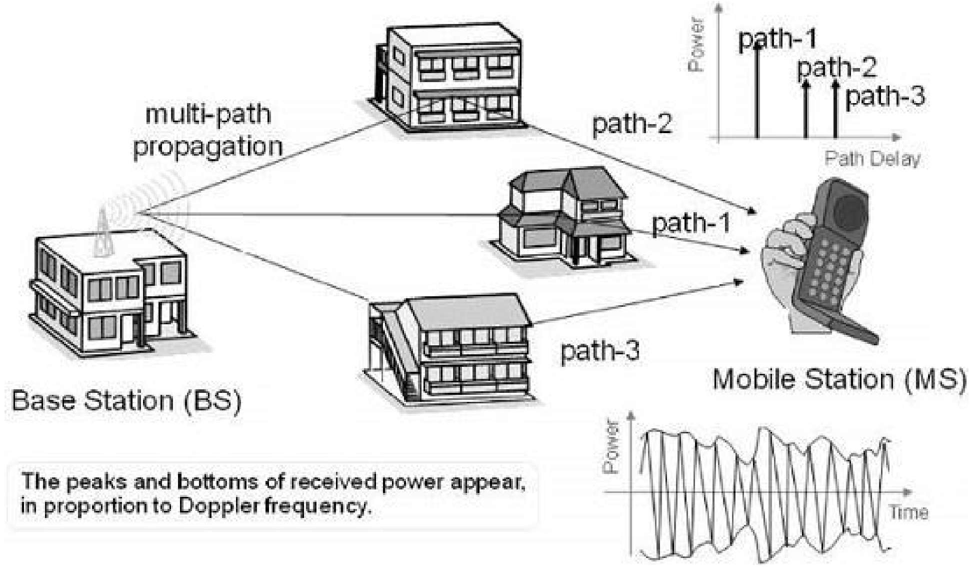
- Live streaming extremely popular now (E.g. going Live on Facebook)
- A person can share their experiences with friends
- Popular live streaming apps include Facebook, Periscope
- Also possible on devices such as Go Pro
- Uses RTMP (real time protocol by Adobe), or other 3 rd party APIs

Facebook Live

Live GoPro

Live Streaming Bandwidth Issues

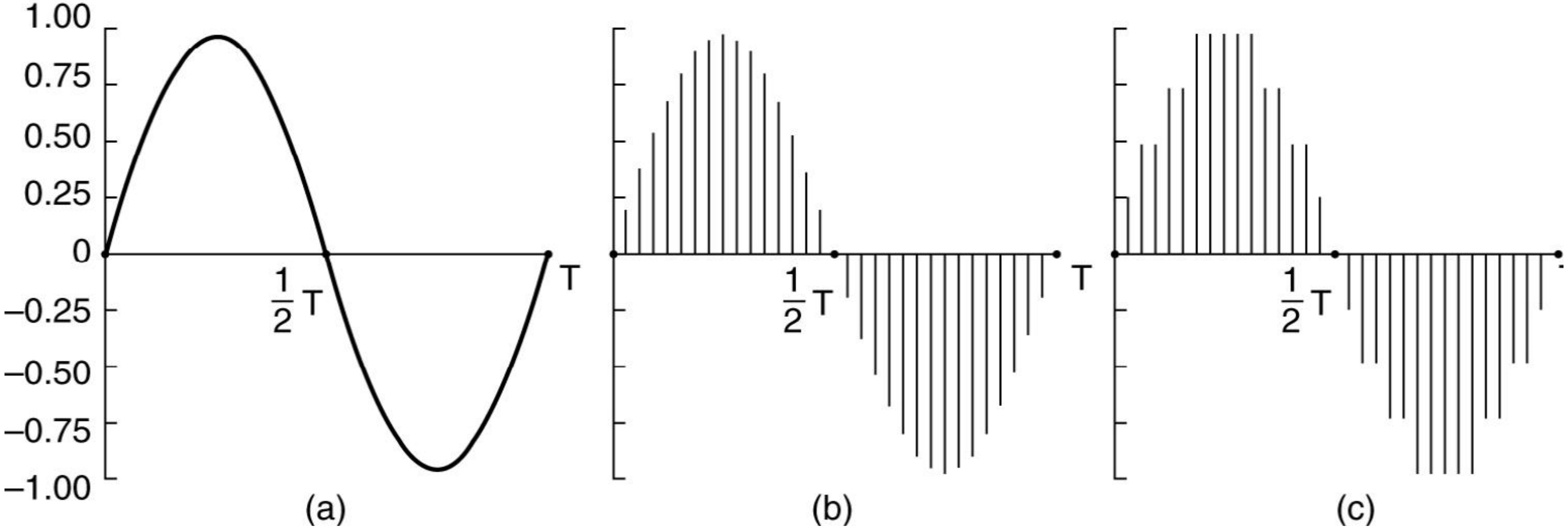
- On WiFi, bandwidth is adequate, high quality video possible
- Cellular links:
  - Low bandwidth,
  - Variable bandwidth (multi-path fading)
    - Even when standing still
  - Optimized for download not upload
- Video quality increasing faster than cellular bandwidths
  - Ultra HD, 4k cameras makes it worse, now available on many smartphones



The peaks and bottoms of received power appear, in proportion to Doppler frequency.

Digital Audio

- Sender converts audio from analog waveform to digital signal
- E.g PCM uses 8-bit samples 8000 times per sec
- Receiver converts digital signal back into audio waveform



Audio Compression

- Audio CDs:
  - 44,100 samples/second
  - Uncompressed audio, requires 1.4Mbps to transmit real-time
- Audio compression reduces transmission bandwidth required
  - E.g. MP3 (MPEG audio layer 3) compresses audio down to 96 kbps

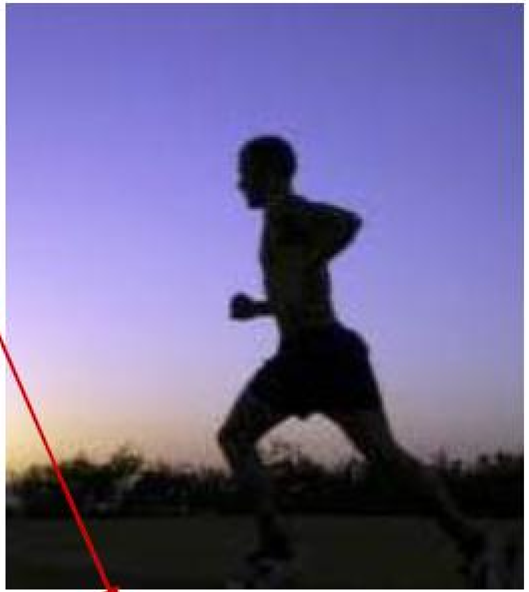
Video Encoding

- **Digital image:** array of <R,G,B> pixels
- **Video:** sequence of images
- **Redundancy:** Consecutive frames mostly same (1/30 secs apart)
- **Video coding (e.g. MPEG):**use redundancy *within*and *between*images to decrease # bits used to encode video
  - **Spatial**(within image)
  - **Temporal**(from 1 image to next)

spatial coding example: instead of sending *N* values of same color (all purple), send only two values: color value (*purple*) *and number of times repeated* (N)

temporal coding example: instead of sending complete frame at i+1, send only differences from frame i

frame *i*



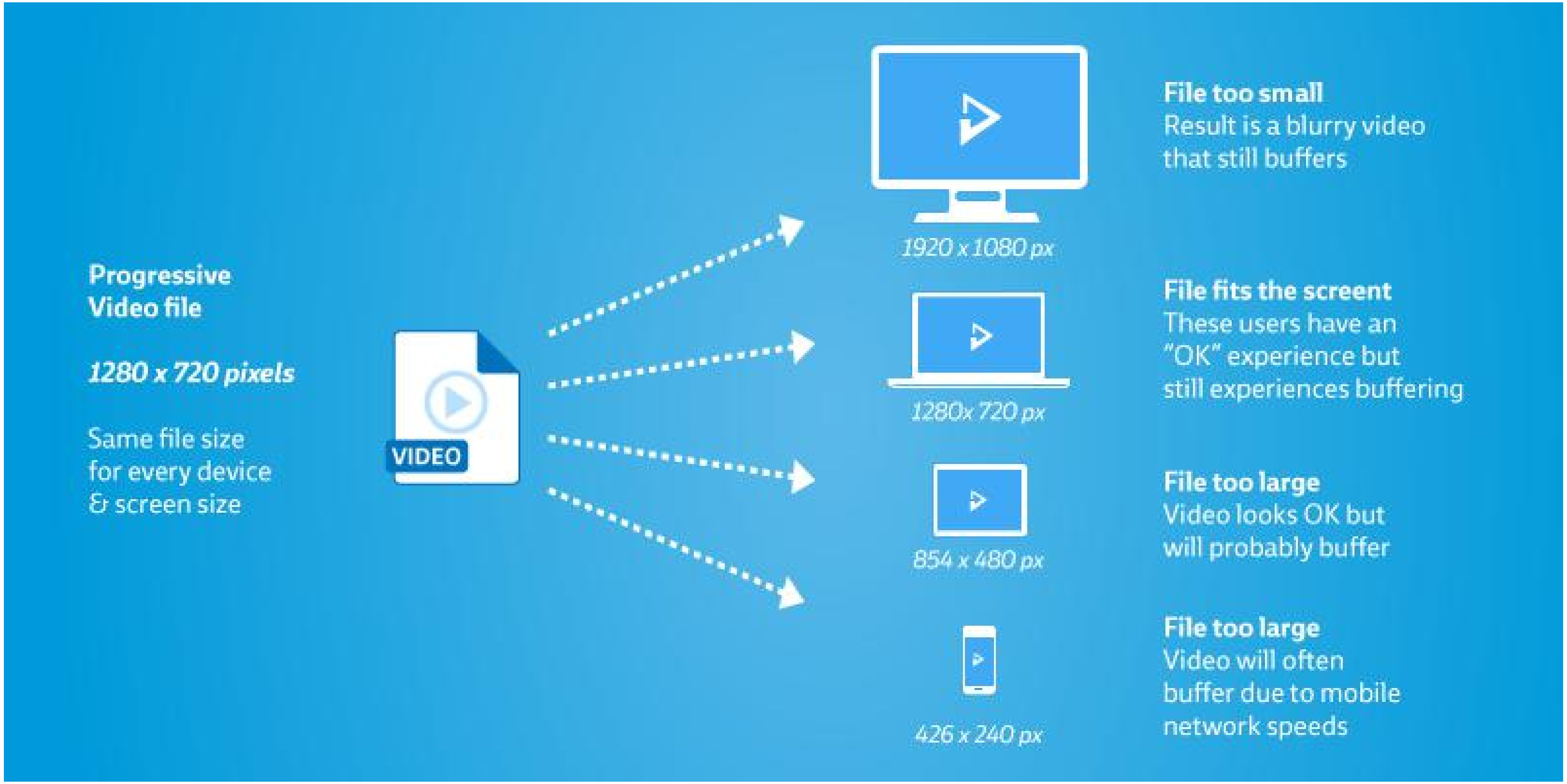
frame *i+1*

Adaptive Video Streaming

Adaptive Video Streaming

- E.g., Dynamic Adaptive Streaming over HTTP (DASH) in Youtube
- Motivation: one size does not fit all

- Baseline: a progressive video stream is simply one single video file being streamed over the internet, then stretch to different screens.



Progressive Video file

1280 x 720 pixels

Same file size
for every device
& screen size

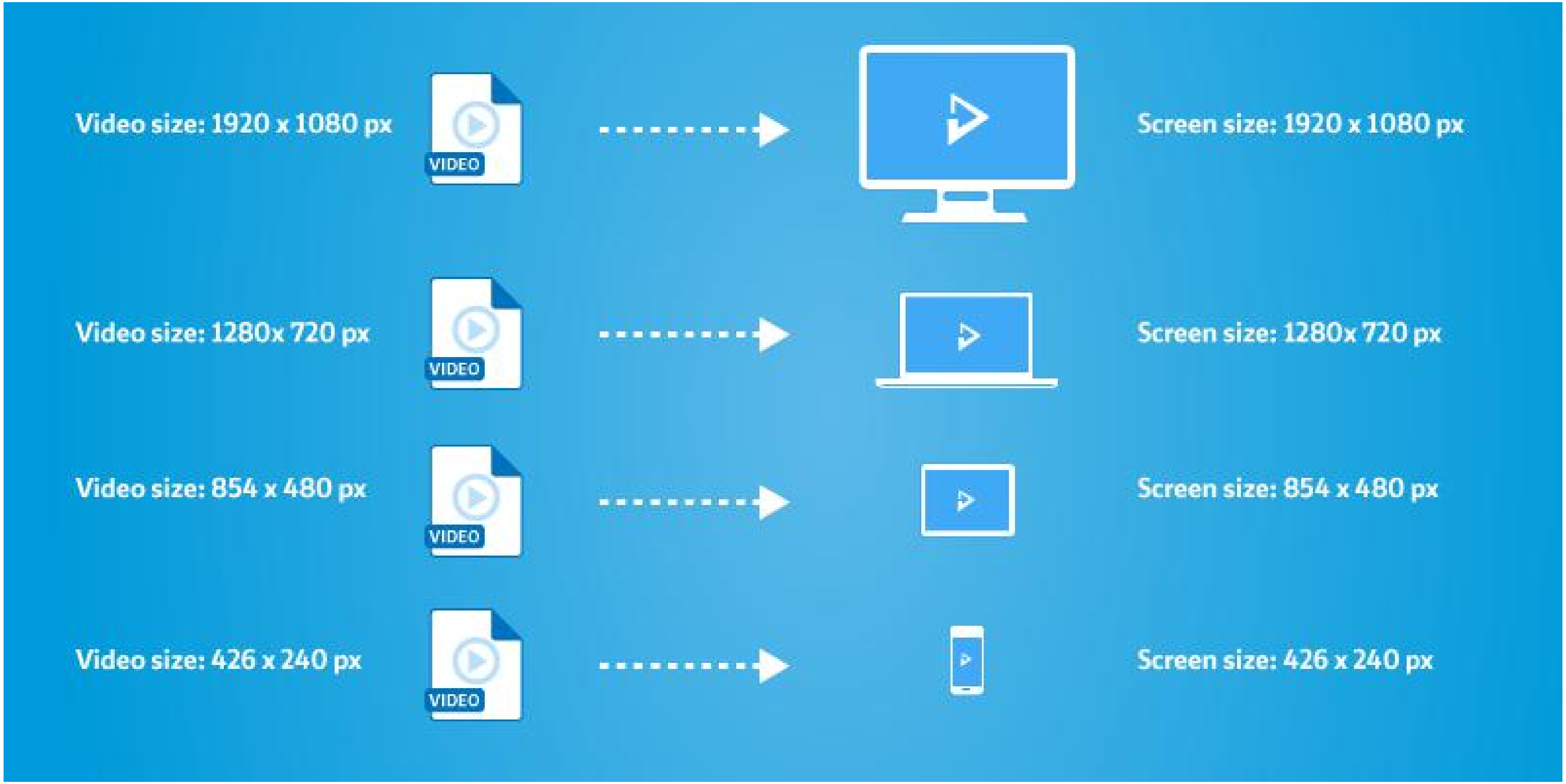
File too small
Result is a blurry video
that still buffers

File fits the screent
These users have an
"OK" experience but
still experiences buffering

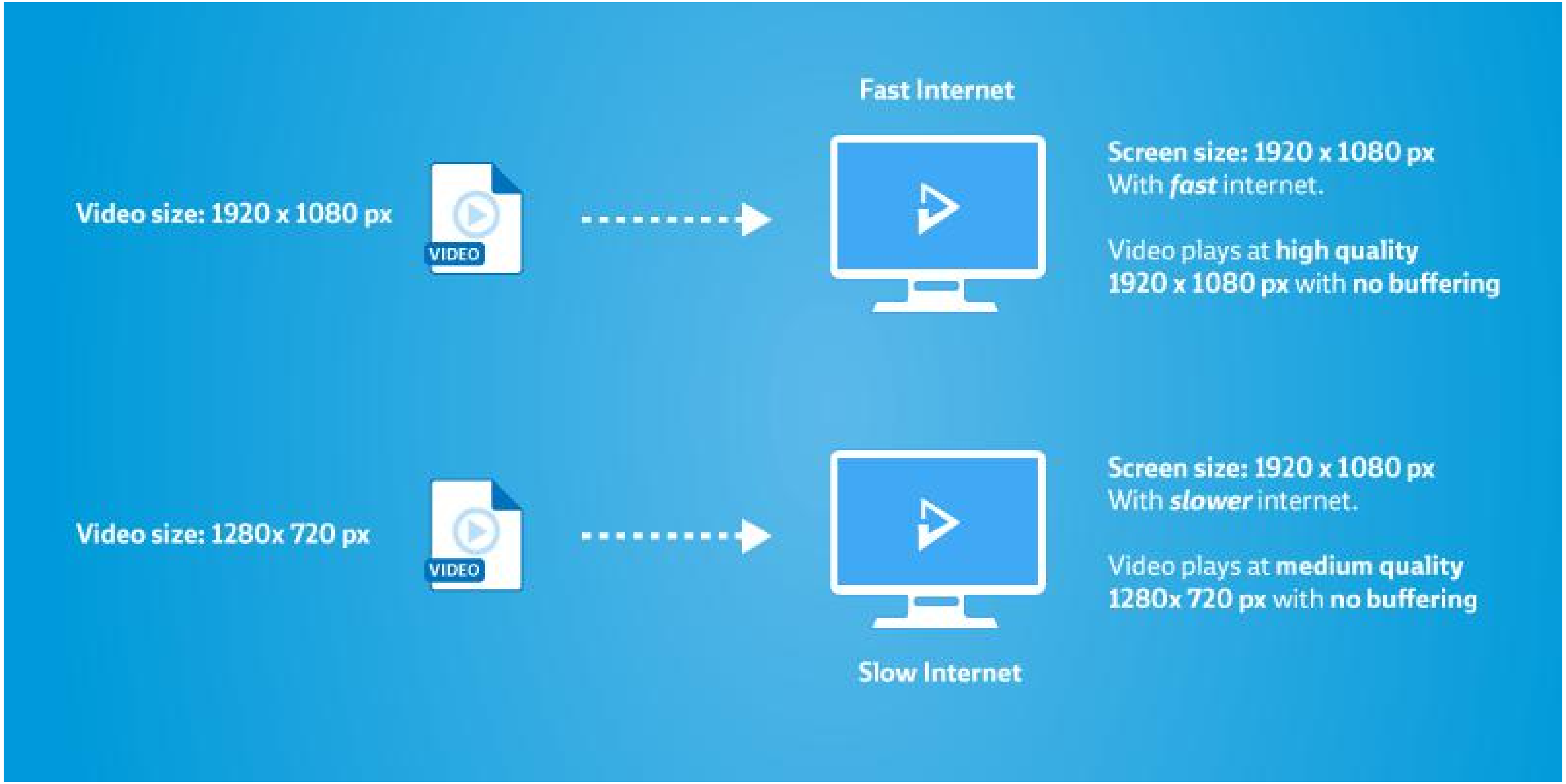
File too large
Video looks OK but
will probably buffer

File too large
Video will often
buffer due to mobile
network speeds

- Adaptive streaming: allows the video provider to create a different video for each of the screen sizes


Video size: 1920 x 1080 px
VIDEO
Screen size: 1920 x 1080 px
Video size: 1280x 720 px
VIDEO
Screen size: 1280x 720 px
Video size: 854 x 480 px
VIDEO
Screen size: 854 x 480 px
Video size: 426 x 240 px
VIDEO
Screen size: 426 x 240 px

- Adjust the video size based on network connection quality



Fast Internet

Video size: 1920 x 1080 px

VIDEO

Screen size: 1920 x 1080 px
With ***fast*** internet.

Video plays at **high quality**
1920 x 1080 px with **no buffering**

Video size: 1280x 720 px

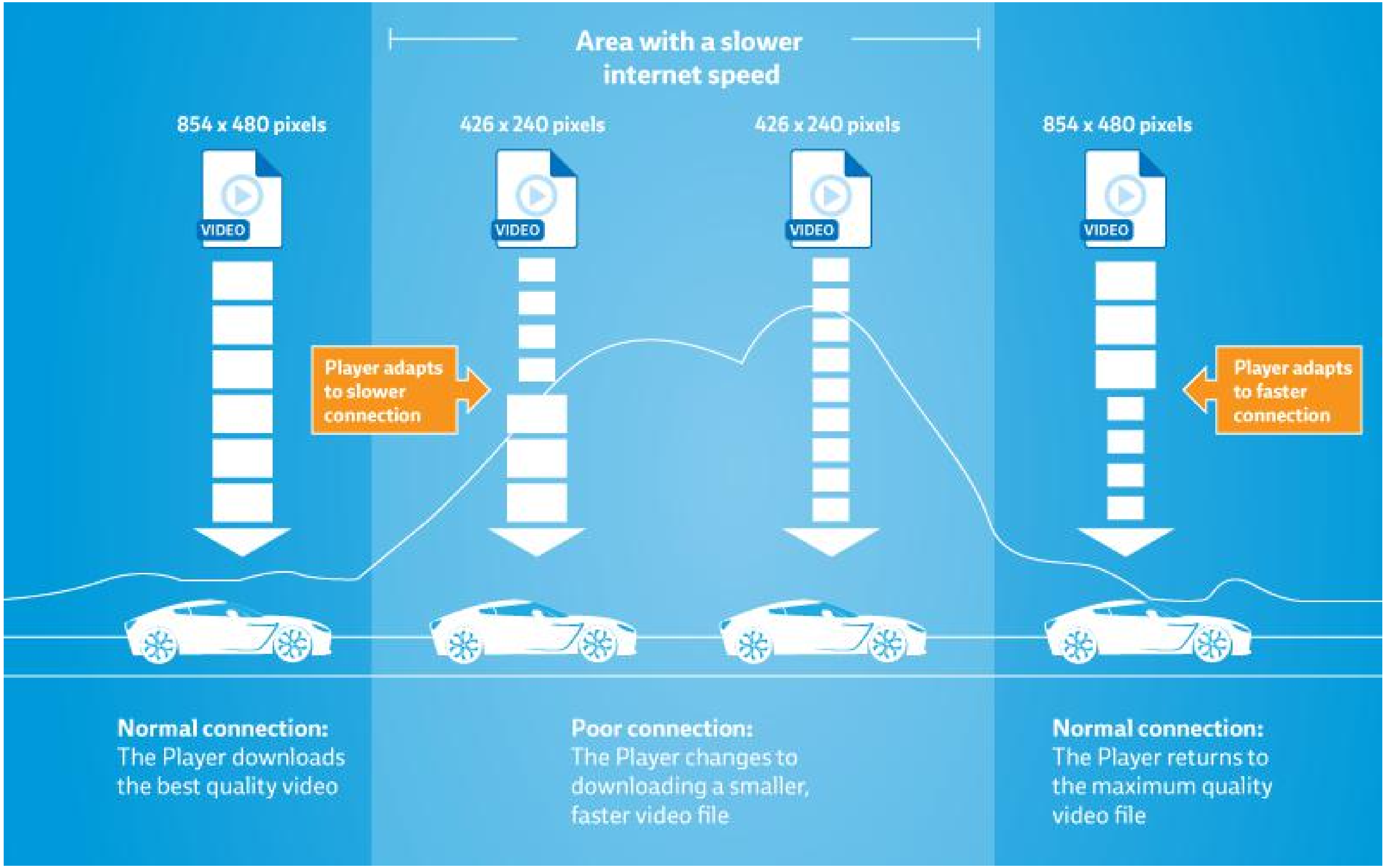
VIDEO

Screen size: 1920 x 1080 px
With ***slower*** internet.

Video plays at **medium quality**
1280x 720 px with **no buffering**

Slow Internet

- The biggest strength: adaptive bitrate


Area with a slower internet speed
854 x 480 pixels
426 x 240 pixels
426 x 240 pixels
854 x 480 pixels
VIDEO
VIDEO
VIDEO
VIDEO
Player adapts to slower connection
Player adapts to faster connection
Normal connection: The Player downloads the best quality video
Poor connection: The Player changes to downloading a smaller, faster video file
Normal connection: The Player returns to the maximum quality video file