

# Lecture #6

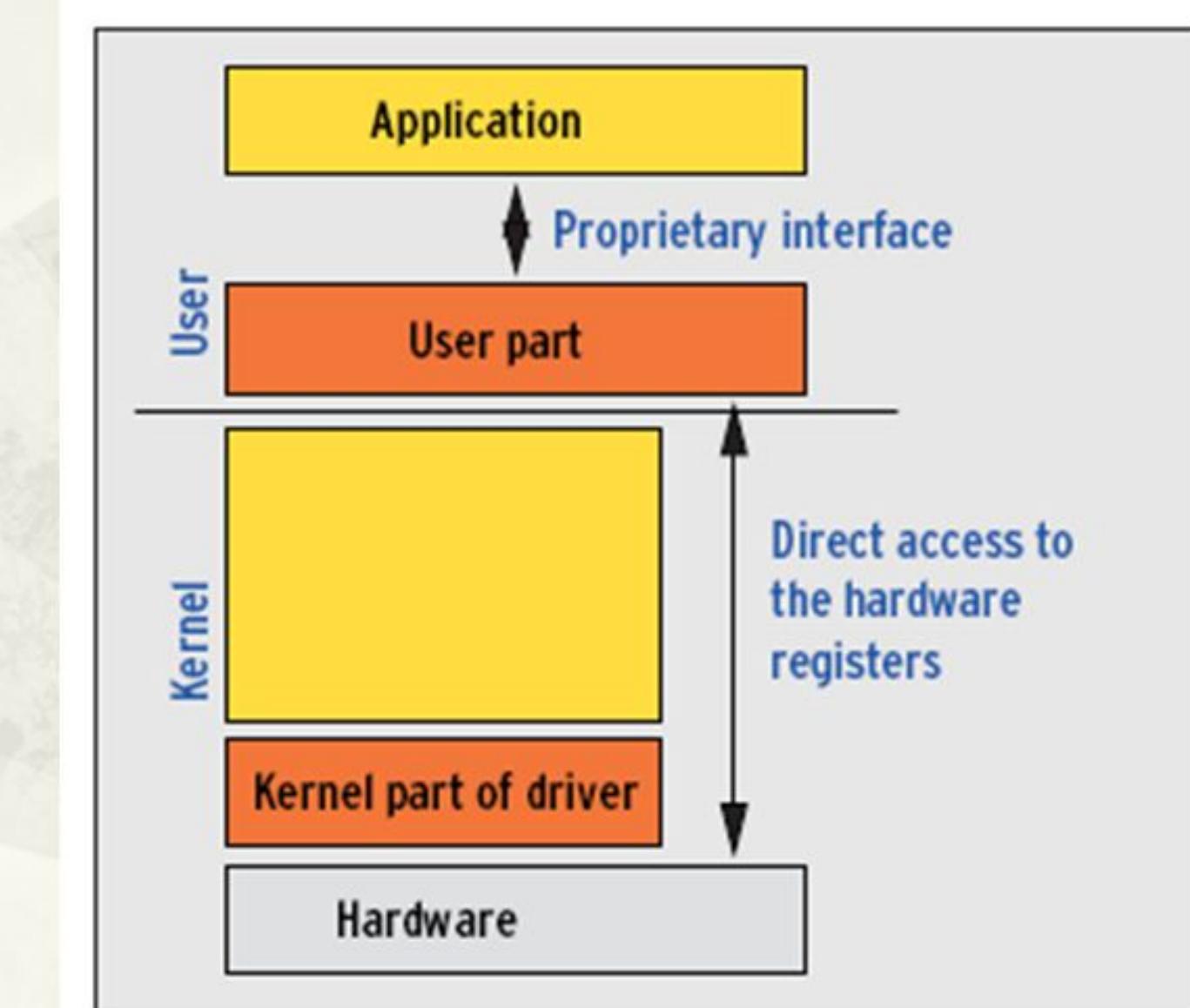
# User-Space Drivers

Spring 2024

# What are User Space Drivers?

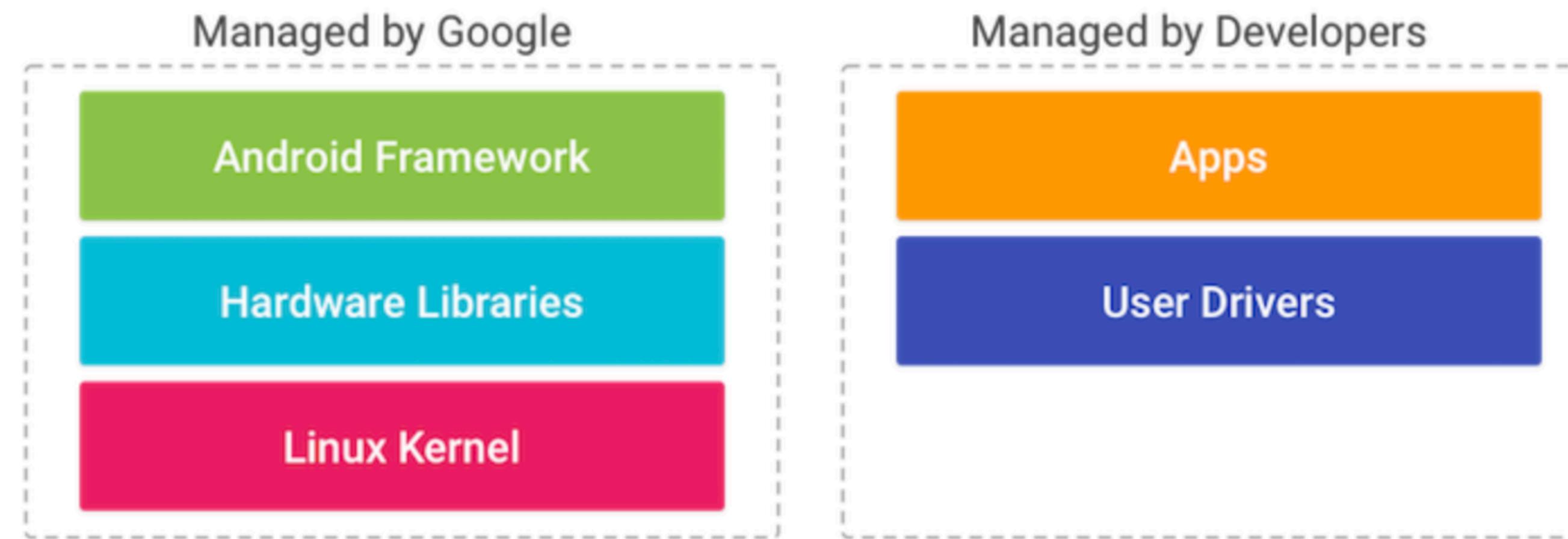
- Definition of user space drivers
- Why user space drivers are important in IoT projects
- Examples of user space drivers

## Linux Userspace Driver Model



# User-Space Drivers

- Components registered from within apps that extend existing Android framework services.



- Inject hardware events into the framework that other apps can process using the standard Android APIs.



**Note:** You cannot customize the behavior of device drivers in the [Linux kernel](#) or [Hardware Abstraction Layer \(HAL\)](#) to add new functionality to a device.

# Benefits

- Portability
- Reuse
- Integration



# Device driver types

- Location
- Human Interface Devices (HI)
- Sensors

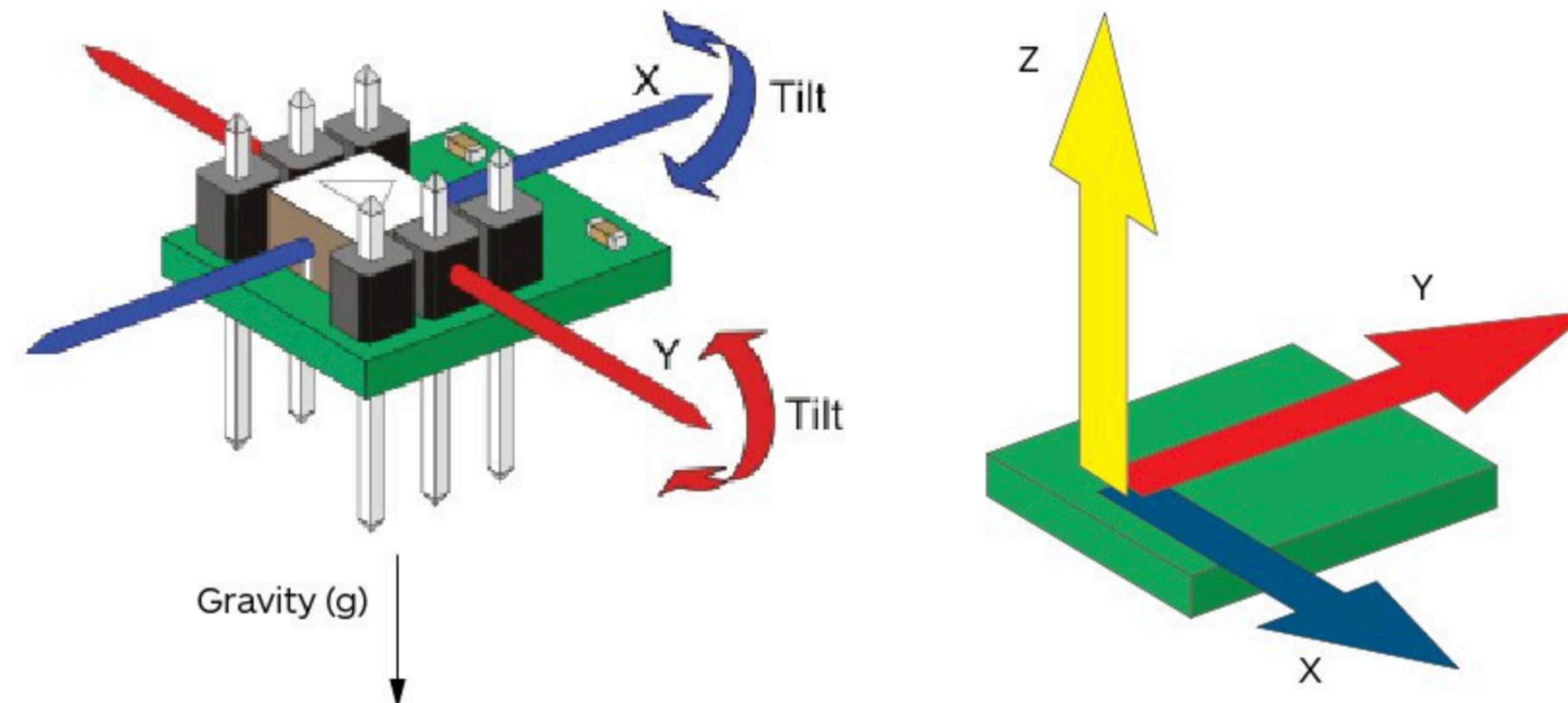


Image source: <https://insights.globalspec.com>

Image source: <https://www.geotab.com>

# Device driver types

- Location
- Human Interface Devices (HID)
- Sensors
- LoWPAN

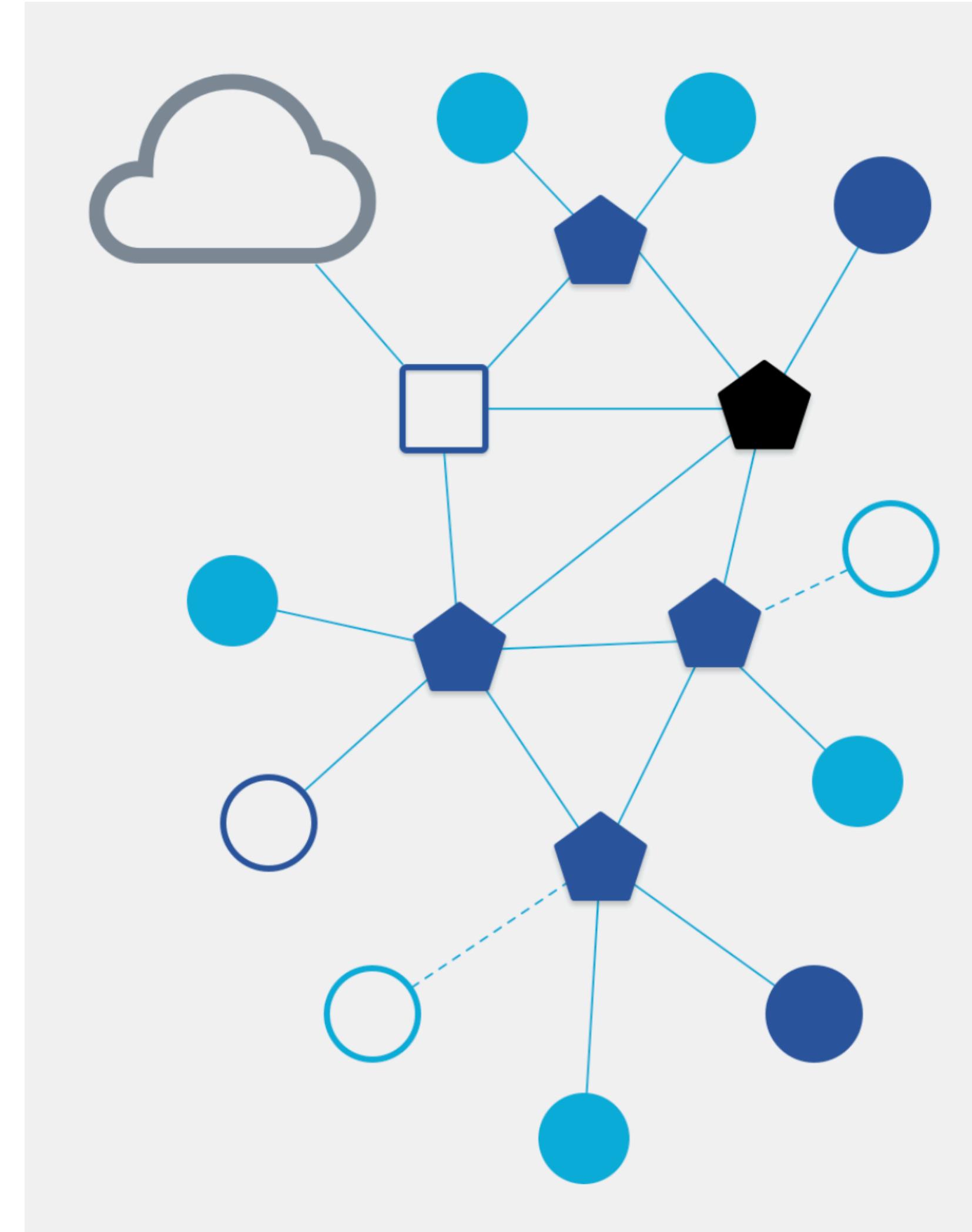


Image source: <https://openthread.io/>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/hid.h>

int main(void)
{
    int fd;
    struct hidraw_report_descriptor rpt_desc;
    struct hidraw_devinfo info;
    char *device = "/dev/hidraw0";

    fd = open(device, O_RDWR|O_NONBLOCK);

    ioctl(fd, HIDIOCGRRAWINFO, &info);
    ioctl(fd, HIDIOCGRDESCSIZE, &rpt_desc.size);
    ioctl(fd, HIDIOCGRDESC, &rpt_desc);

    printf("Raw Input Data:\n");

    while(1)
    {
        unsigned char buf[8];
        int nbytes;
```

```
ioctl(fd, HIDIOCGRRAWINFO, &info);
ioctl(fd, HIDIOCGRDESCSIZE, &rpt_desc.size);
ioctl(fd, HIDIOCGRDESC, &rpt_desc);
```

```
printf("Raw Input Data:\n");
```

```
while(1)
```

```
{
```

```
    unsigned char buf[8];
```

```
    int nbytes;
```

```
    nbytes = read(fd, buf, sizeof(buf));
```

```
    if (nbytes == sizeof(buf))
```

```
{
```

```
        printf("Received data: ");
```

```
        for (int i = 0; i < nbytes; i++)
```

```
{
```

```
            printf("%02x ", buf[i]);
```

```
}
```

```
        printf("\n");
```

```
}
```

```
        usleep(1000);
```

```
}
```

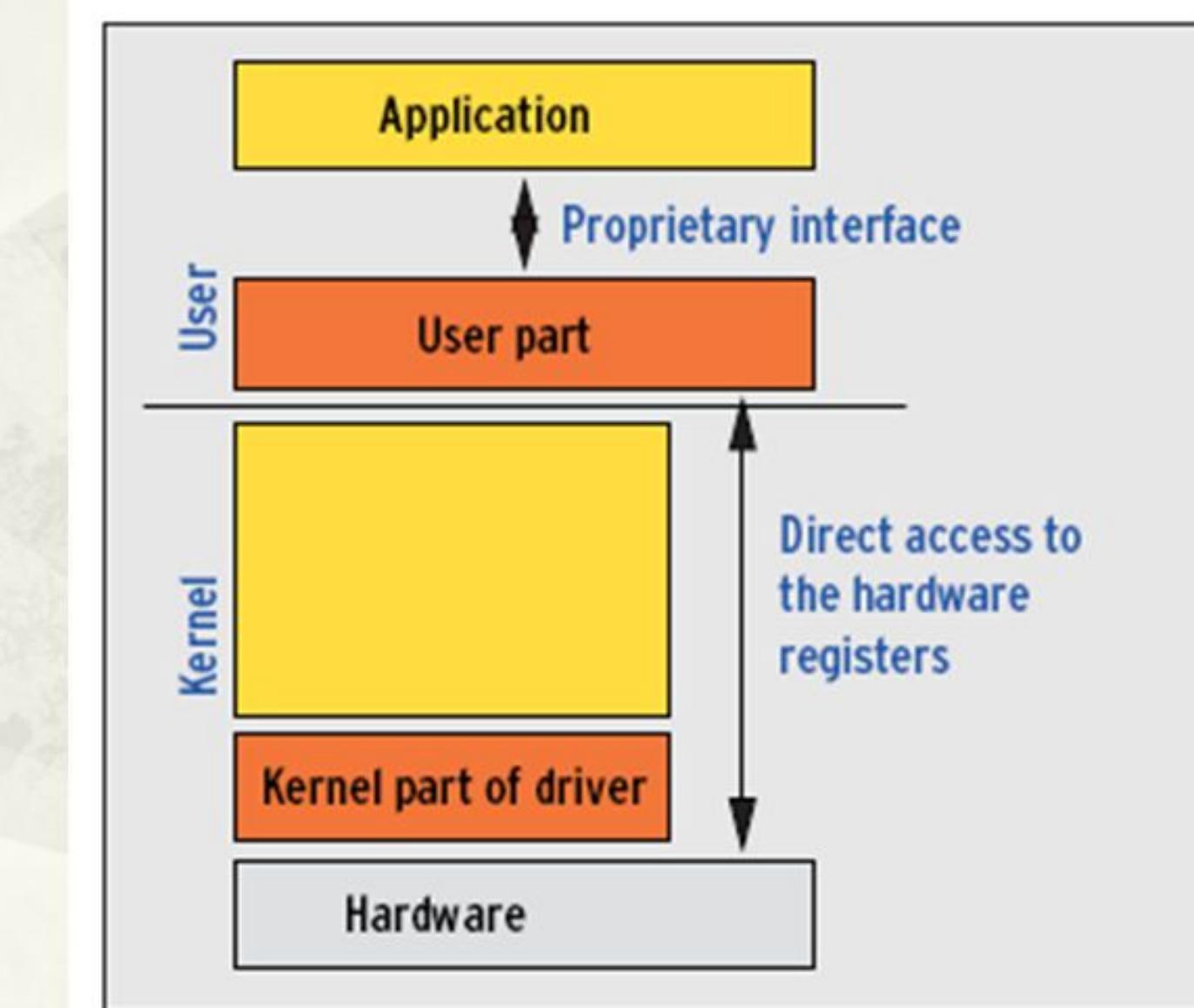
```
return 0;
```

```
}
```

# Advantages of User Space Drivers

- More control over the hardware
- Easier to develop custom drivers
- Greater flexibility in the development process

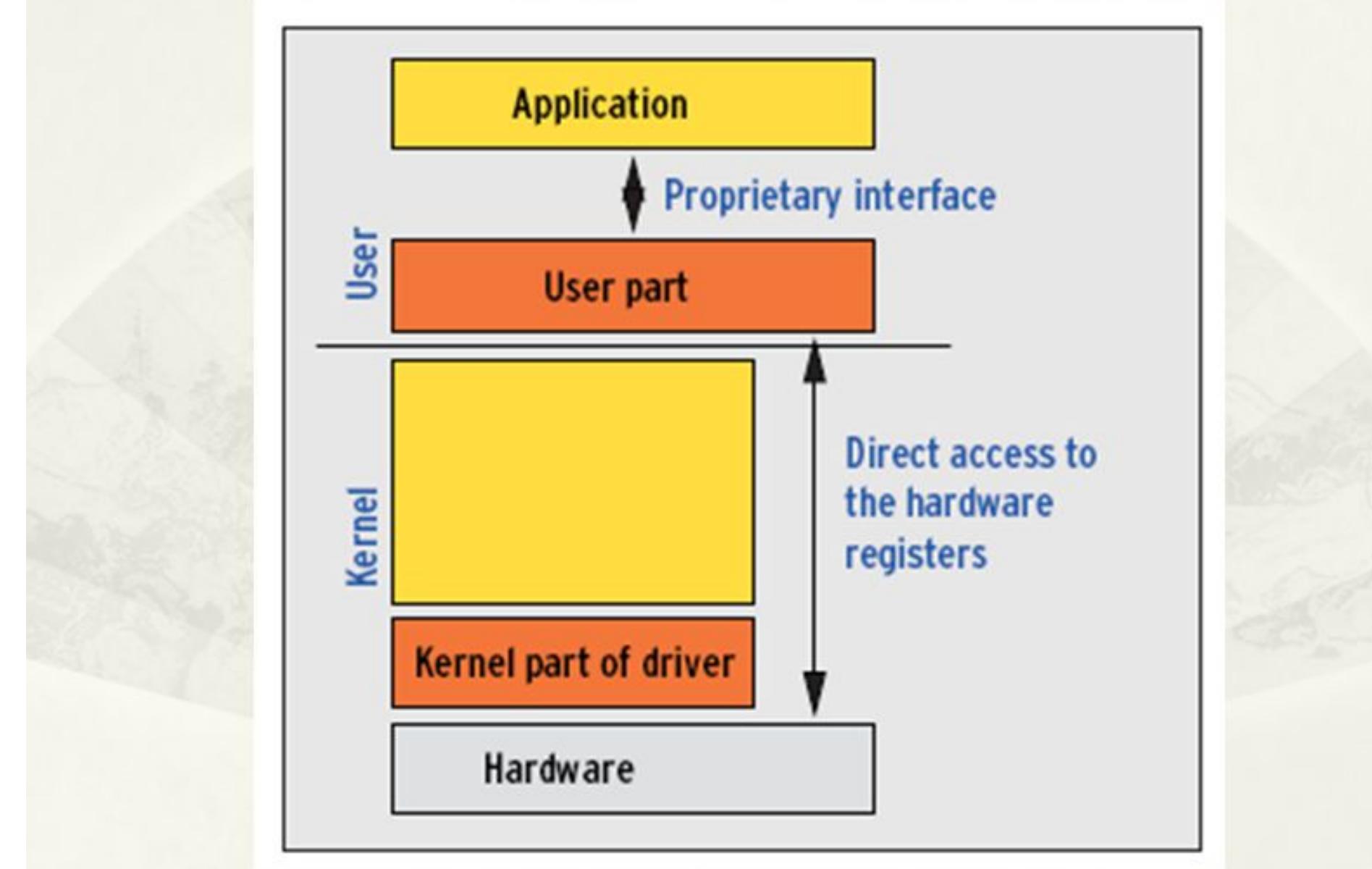
## Linux Userspace Driver Model



# Challenges of User Space Drivers

- Limited access to system resources
- Lower performance compared to kernel space drivers
- Increased risk of system crashes

## Linux Userspace Driver Model



# WiringPi

```
#include <wiringPi.h>
int main(void)
{
    wiringPiSetup();
    pinMode(0, OUTPUT);
    while(1)
    {
        digitalWrite(0, HIGH);
        delay(1000);
        digitalWrite(0, LOW);
        delay(1000);
    }
    return 0;
}
```

# Pi4J

```
import com.pi4j.component.temperature.TemperatureSensor;
import com.pi4j.temperature.TemperatureScale;
import com.pi4j.io.gpio.RaspiPin;
import com.pi4j.io.w1.W1Device;
import com.pi4j.io.w1.W1Master;
public class TemperatureReader {
    public static void main(String[] args) throws Exception {
        W1Master master = new W1Master();
        TemperatureSensor sensor = new TemperatureSensor(master.getDevice("28-00000737b5a5"));
        System.out.println("Temperature: " + sensor.getTemperature(TemperatureScale.CELSIUS));
    }
}
```

# BCM2835

```
#include <bcm2835.h>
int main(void)
{
    bcm2835_spi_begin();
    bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_256);
    bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);
    bcm2835_spi_chipSelect(BCM2835_SPI_CS0);
    bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW);
    uint8_t tx[3] = {0x01, 0x80, 0x00};
    uint8_t rx[3];
    bcm2835_spi_transfernb(tx, rx, 3);
    uint16_t value = ((rx[1] & 0x03) << 8) | rx[2];
    printf("ADC value: %d\n", value);
    bcm2835_spi_end();
    return 0;
}
```

# pigpio

```
import pigpio  
pi = pigpio.pi()  
pi.set_mode(17, pigpio.INPUT)  
while True:  
    if pi.read(17) == pigpio.HIGH:  
        print("Button pressed!")  
pi.stop()
```

# OpenCV

```
import cv2
cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    cv2.imshow('Video', frame)
    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

# TensorFlow Lite

```
import numpy as np
import tensorflow as tf
from PIL import Image
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
image = Image.open("test.jpg")
image = np.array(image.resize((224, 224))) / 255.0
image = np.expand_dims(image, axis=0).astype(np.float32)
interpreter.set_tensor(input_details[0]['index'], image)
interpreter.invoke()
output_data = interpreter.get_tensor(output_details[0]['index'])
print(output_data)
```

# Global Navigation Satellite System (GNSS)

```
import com.google.android.things.udriver.location.GnssDriver;
import com.google.android.things.udriver.UserDriverManager;
...
public class LocationDriverService extends Service {
    private GnssDriver mDriver;
    @Override
    public void onCreate() {
<uses-permission android:name="com.google.android.things.permission.MANAGE_GNSS_DRIVERS" />
        // Create a new driver implementation
        mDriver = new GnssDriver();
        // Register with the framework
        UserDriverManager manager = UserDriverManager.getInstance();
        manager.registerGnssDriver(mDriver);
    }
}
```

# Global Navigation Satellite System (GNSS)

```
public class LocationDriverService extends Service {  
    ...  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        UserDriverManager manager = UserDriverManager.getInstance();  
        manager.unregisterGnssDriver();  
    }  
}
```

# GNSS - Location Reporting

```
import android.location.Location;  
...  
public class LocationDriverService extends Service {  
    private GnssDriver mDriver;  
    ...  
    private Location parseLocationFromString(String data) {  
        Location result = new Location(LocationManager.GPS_PROVIDER);  
        // ...parse raw GNSS information...  
        return result;  
    }  
    public void handleLocationUpdate(String rawData) {  
        // Convert raw data into a location object  
        Location location = parseLocationFromString(rawData);  
        // Send the location update to the framework  
        mDriver.reportLocation(location);  
    }  
}
```

# Location Attributes

Required Attributes	Optional Attributes
Accuracy	Altitude
Timestamp	Bearing
Latitude	Speed
Longitude	

# Human Interface Devices

```
import com.google.android.things.userdriver.input.InputDriver;
import com.google.android.things.userdriver.UserDriverManager;
...
public class ButtonDriverService extends Service {
    // Driver parameters
    private static final String DRIVER_NAME = "EscapeButton";
    private static final int DRIVER_VERSION = 1;
    <uses-permission android:name="com.google.android.things.permission.MANAGE_INPUT_DRIVERS" />
    private static final int KEY_CODE = KeyEvent.KEYCODE_ESCAPE;
    private InputDriver mDriver;
    @Override
    public void onCreate() {
        super.onCreate();

        // Create a new driver instance
        mDriver = new InputDriver.Builder()
            .setName(DRIVER_NAME)
            .setSupportedKeys(new int[] {KEY_CODE})
        .build();
    }
}
```

<https://developer.android.com/things/sdk/drivers/input.html>

# Sensors

```
UserSensorDriver driver = new UserSensorDriver() {  
    // Sensor data values  
    float x, y, z;  
  
    @Override  
    public UserSensorReading read() {  
        <uses-permission android:name="com.google.android.things.permission.MANAGE_SENSOR_DRIVERS" />  
        // ...read the sensor hardware...  
  
        // Return a new reading  
        return new UserSensorReading(new float[]{x, y, z});  
    } catch (Exception e) {  
        // Error occurred reading the sensor hardware  
        throw new IOException("Unable to read sensor");  
    }  
};
```

# Low Power Sensor

```
UserSensorDriver driver = new UserSensorDriver() {  
    ...  
  
    // Called by the framework to toggle low power modes  
    @Override  
    public void setEnabled(boolean enabled) {  
        if (enabled) {  
            // Exit low power mode  
        } else {  
            // Enter low power mode  
        }  
    }  
};
```

# Describing the Sensor

```
UserSensor accelerometer = UserSensor.builder()  
    .setName("KoolAccelerometerSensor")  
    .setVendor("UBBFactory")  
    .setType(Sensor.TYPE_ACCELEROMETER)  
    .setDriver(driver)  
    .build();
```

# Registering the Sensor

```
import com.google.android.things.userdriver.UserDriverManager;  
...  
public class SensorDriverService extends Service {  
    UserSensor accelerometer;  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        ...  
        UserDriverManager manager = UserDriverManager.getInstance();  
        // Create a new driver implementation  
        accelerometer = ...;  
        // Register the new driver with the framework  
        manager.registerSensor(accelerometer);  
    }  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        ...  
        UserDriverManager manager = UserDriverManager.getInstance();  
        // Unregister the driver when finished
```

# Access the Sensors

```
public class SensorActivity extends Activity implements SensorEventListener {  
    private SensorManager sensorManager;  
    private SensorCallback callback = new SensorCallback();  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
        sensorManager.registerDynamicSensorCallback(callback);  
    }  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        ...  
        sensorManager.unregisterDynamicSensorCallback(callback);  
    }  
    @Override  
    public void onSensorChanged(SensorEvent event) {  
        ...  
    }  
}
```

# LoWPAN



## 1. Introduction

# OPEN THREAD

released by Nest

[OpenThread](#) is an open-source implementation of the [Thread](#) networking protocol. Nest has released OpenThread to make the technology used in Nest products broadly available to developers to accelerate the development of products for the connected home.

The [Thread specification](#) defines an IPv6-based reliable, secure and low-power wireless device-to-device communication protocol for home applications. OpenThread implements all Thread networking layers including IPv6, 6LoWPAN, IEEE 802.15.4 with MAC security, Mesh Link Establishment, and Mesh Routing.

This Codelab will walk you through simulating a Thread network on emulated devices using Docker.

### What you'll learn

- ✓ How to set up the OpenThread build toolchain
- ✓ How to simulate a Thread network
- ✓ How to authenticate Thread nodes
- ✓ How to manage a Thread network with `wpantund`

### What you'll need

- Docker
- Basic knowledge of Linux, network routing

# Create a Driver

```
apply plugin: 'com.android.library'

android {
    compileSdkVersion 27
    buildToolsVersion '28.0.3'

    defaultConfig {
        minSdkVersion 27
        targetSdkVersion 27
        // Other default stuff...
    }
}

dependencies {
    provided 'com.google.android.things:androidthings:0.1-devpreview'
}
```

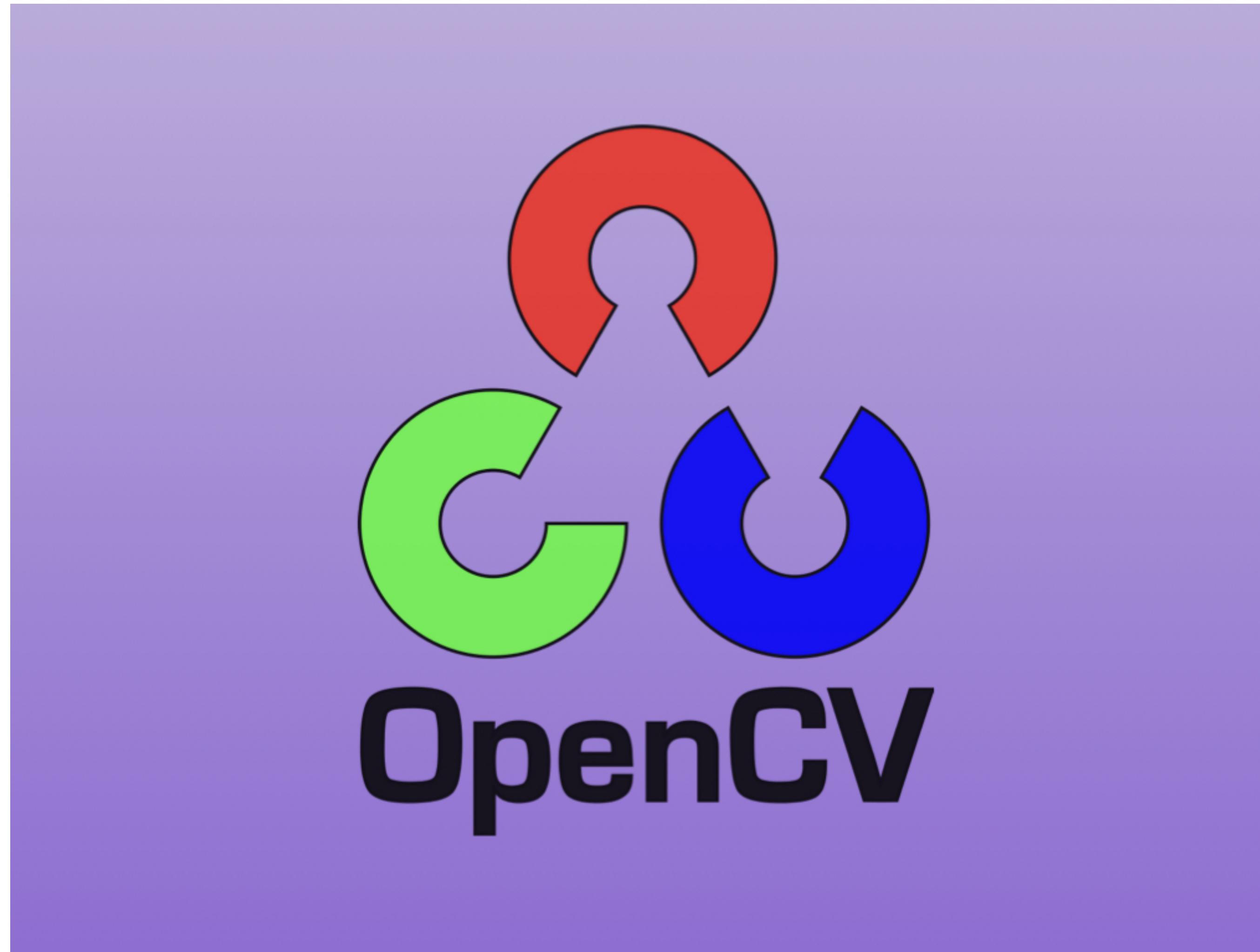
# Manifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mydriver">
    <application>
        <uses-library android:name="com.google.android.things" />
    </application>
</manifest>
```

# Driver

```
class LEDSensor : AutoCloseable {  
    private lateinit var redPin: Gpio  
    private lateinit var greenPin: Gpio  
    private lateinit var bluePin: Gpio  
  
    fun init(redGPIOName: String = "BCM17", greenGPIOName: String = "BCM27", blueGPIOName: String = "BCM22") {  
        val pioManager = PeripheralManager.getInstance()  
  
        redPin = pioManager.openGpio(redGPIOName)  
        greenPin = pioManager.openGpio(greenGPIOName)  
        bluePin = pioManager.openGpio(blueGPIOName)  
        initGPIO(redPin)  
        initGPIO(greenPin)  
        initGPIO(bluePin)  
    }  
  
    fun write(red: Boolean = false, green: Boolean = false, blue: Boolean = false) {  
        redPin.value = red  
        greenPin.value = green  
        bluePin.value = blue  
    }  
}
```

# OpenCV



<https://opencv.org/>

# Install

```
pip3 install opencv-python
```

<https://opencv.org/>

# Verification

```
pip3 install opencv-python
```

```
import cv2 as cv
img = cv.imread("path/to/image")

cv.imshow("Display window", img)
k = cv.waitKey(0) # Wait for a keystroke in the window
```

# Install on Raspberry Pi

```
sudo apt-get update && sudo apt-get upgrade
```

```
#To install image I/O packages that will allow us to load various image file formats from disk:  
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev
```

```
#These video I/O packages enables various video file formats  
# to be read and enables you to work with video streams:  
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev  
sudo apt-get install libxvidcore-dev libx264-dev
```

```
#The following command installs the GTK development library which is required to compile the highgui module,  
#used to display images to the screen and build basic GUIs:  
sudo apt-get install libgtk2.0-dev
```

```
#For the further optimisation of more operations in OpenCV, a few more extra dependencies are required.  
#So run the command:  
sudo apt-get install libatlas-base-dev gfortran
```

# Install on Raspberry Pi

#The package management tool, pip by running the following command:

```
sudo apt-get install python3-pip
```

#Numpy library adds mathematical and number handling functionality

```
pip install numpy
```

#Install OpenCV:

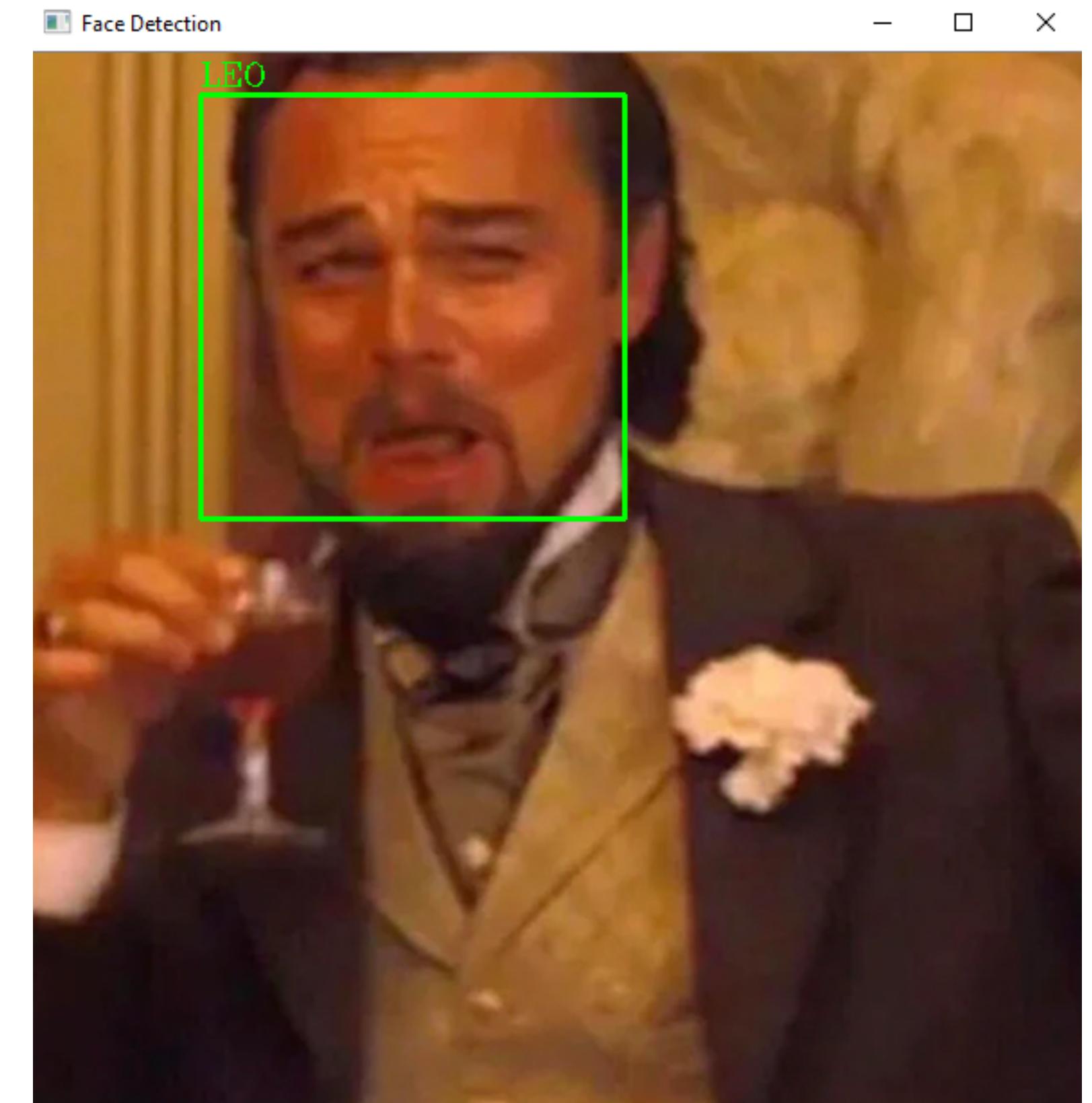
```
sudo apt install python3-opencv
```

# Example

```
import cv2
faceCascade = cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
img = cv2.imread("Resources/picture.png")
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = faceCascade.detectMultiScale(imgGray, 1.1, 39)
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
    cv2.putText(img, "LEO", (x,y-5), cv2.FONT_HERSHEY_COMPLEX, 0.6, (0,255,0), 1)
    cv2.imshow("Face Detection", img)
cv2.waitKey(0)
```

# Example

```
import cv2
faceCascade = cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
img = cv2.imread("Resources/picture.png")
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = faceCascade.detectMultiScale(imgGray, 1.1, 39)
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
    cv2.putText(img, "LEO", (x,y-5), cv2.FONT_HERSHEY_COMPLEX, 0.6, (0,255,0), 1)
cv2.imshow("Face Detection", img)
cv2.waitKey(0)
```



# Using the Camera

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
faceCascade = cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
while True:
    success, img = cap.read()
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(imgGray, 1.1, 39)
    for (x,y,w,h) in faces:
        cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)
        cv2.putText(img, "Face", (x,y-10), cv2.FONT_HERSHEY_COMPLEX, 0.6, (0,255,0), 1)
    cv2.imshow("Output", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

# Lecture outcomes

- How to create a user driver.
- How to use the existing user-defined drivers.

