Разработка аксессуаров для Android

Темы

- Разработка приложений для Android™
  - Инструменты
  - Основные концепции разработки приложений для Android app development concepts and tips
- Интерфейсы для подключения акскссуаров
  - Что доступно
  - Как подключить аксессуар через каждый из интерфейсов
  - Что важно знать

Разработка приложений для Android

android

Java

- Android приложения создаются на Java
- Google предлагает полную документацию, описывающую все классы системы
  - http://developer.android.com/reference/packages.html
  - Полное описание всех API, членов классов, конструкторов, иерархии и .т.д.

Инструменты

- Google предлагает sdk, включающий также эмулятор и отладчик для связи с реальными устройствами
- Наиболее распространенным IDE для отладки Java приложений является Eclipse
- Google предлагает plugin для Eclipse, упрощающий разработку приложений для Android

Threads: что это и зачем ?

```
while(true) {
    if(button1_pressed == 1) { DoSomethingLong1(); }
    if(button2_pressed == 1) { DoSomethingLong2(); }
}
```

- После того, как кнопка нажата пользовательский интерфейс «зависает».
- Пользователь может пытается нажимать другие кнопки, но никакой реакции не будет пока выполняется функция DoSomethingLong()

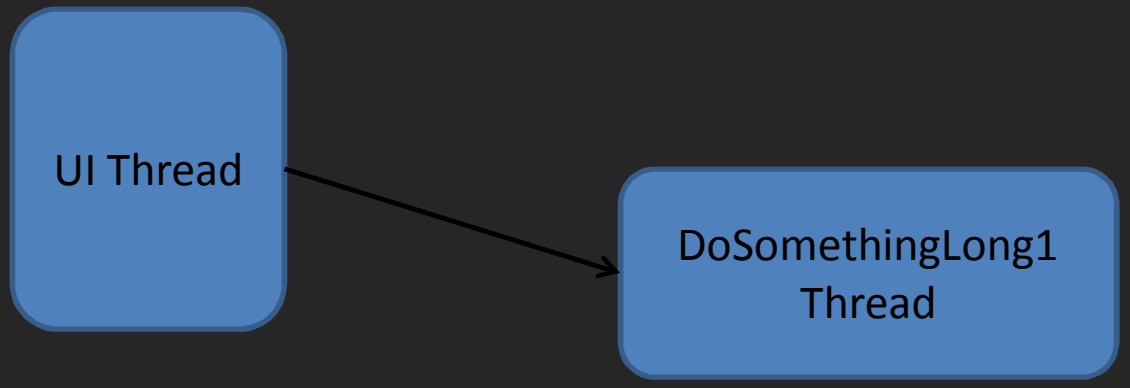
Threads: Что это и зачем ?

- Thread (программный поток) – это разветвление выполнения программы внутри процесса
- Позволяет системе быстро выполнять критические ко времени задачи (например, пользовательский интерфейс), а менее критические задачи выполнять когда на это есть время

Threads: Что это и зачем ?

```
while(true) {
    if(button1_pressed == 1) { new Thread(DoSomethingLong1()); }
    if(button2_pressed == 1) { new Thread(DoSomethingLong2()); }
}
```

- При нажатии кнопки создается отдельный поток (Thread), в котором будет выполняться «длинная» задача. А задача пользовательского интерфейса продолжает воспринимать события от пользователя.



Threads: Что это и зачем ?

- Зачем мне это знать?
  - Операции потокового чтения и записи в файл (filestream), при помощи которых происходит обмен с аксессуаром через большинство интерфейсов, являются блокирующими !!!
    - Обмен данными должен быть организован в отдельном от пользовательского интерфейса потоке (thread)

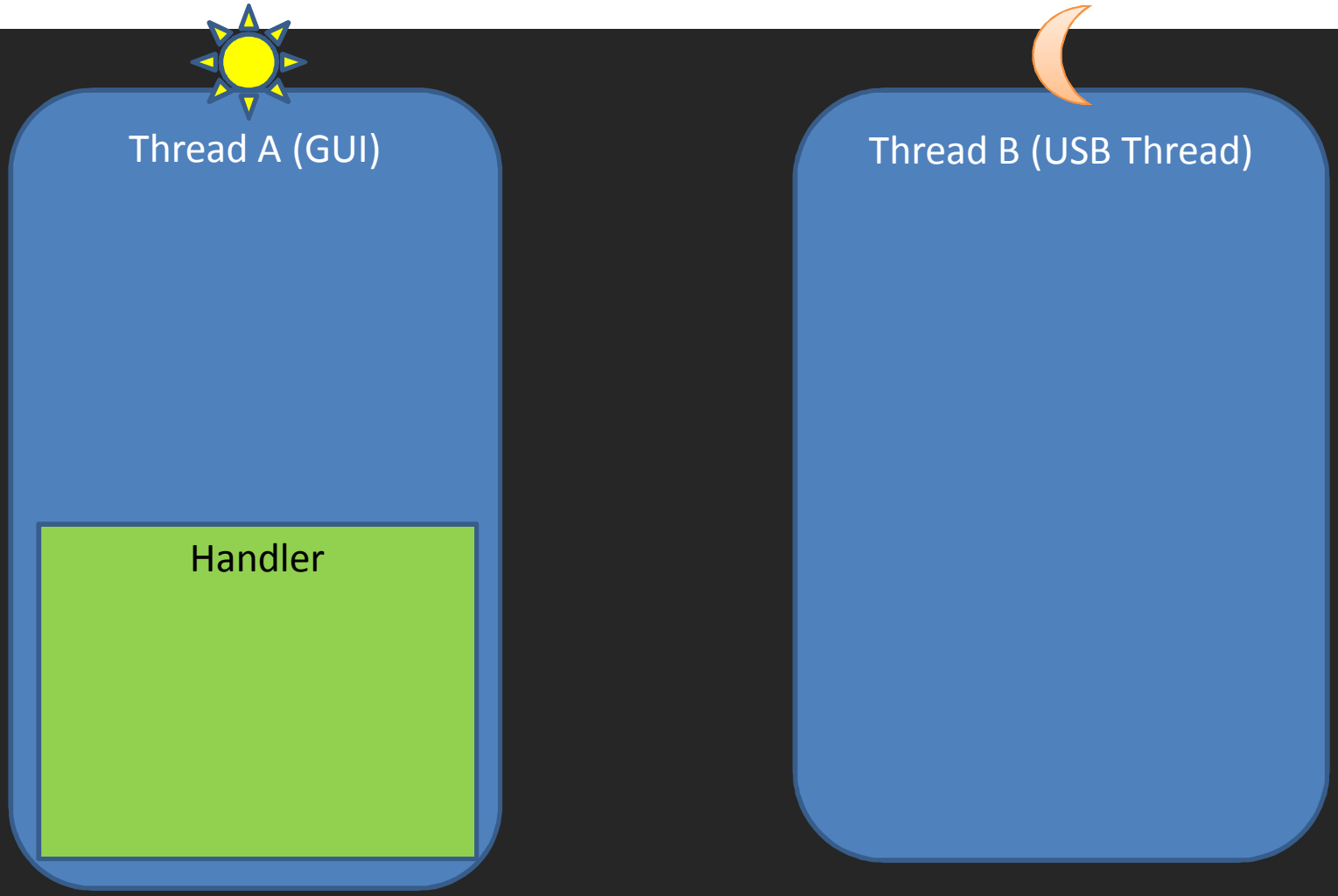
Threads: Обработчики (Handlers)

- Обработчик – это способ обмена данными между различными потоками
- Обработчик принимает сообщение или фрагмент выполняемого кода из другого потока и обрабатывает это сообщение или выполняет код в принимающем потоке,

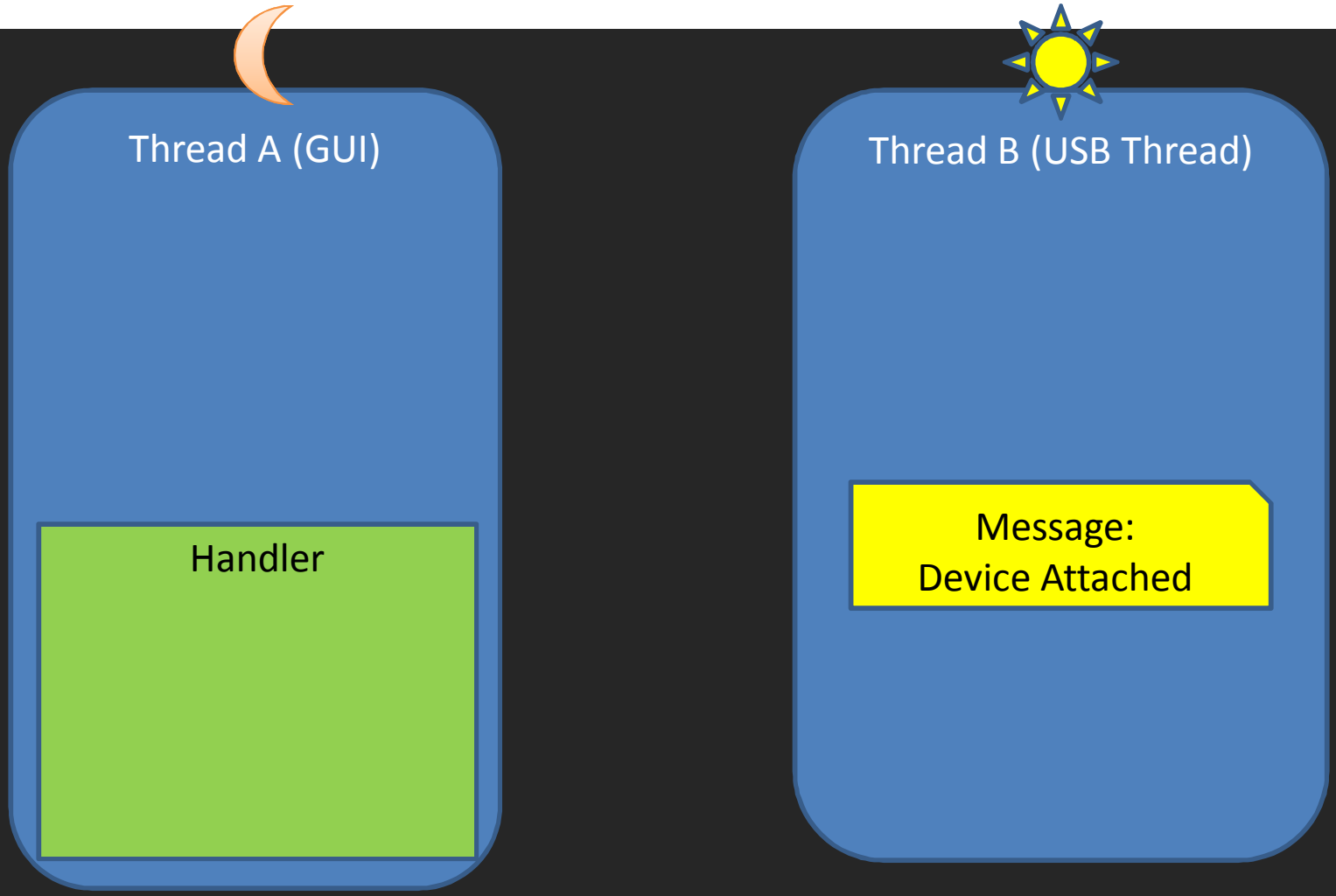
Threads: Сообщения (Messages)

- Сообщение принимается когда поток свободен.
- Сообщения формируют очередь до тех пор, пока не будут обработаны.
- Члены сообщений:
  - what: int, описывающий что это за сообщение
  - arg1: int, который используется как параметр
  - arg2: int, который используется как параметр
  - obj: любой объект

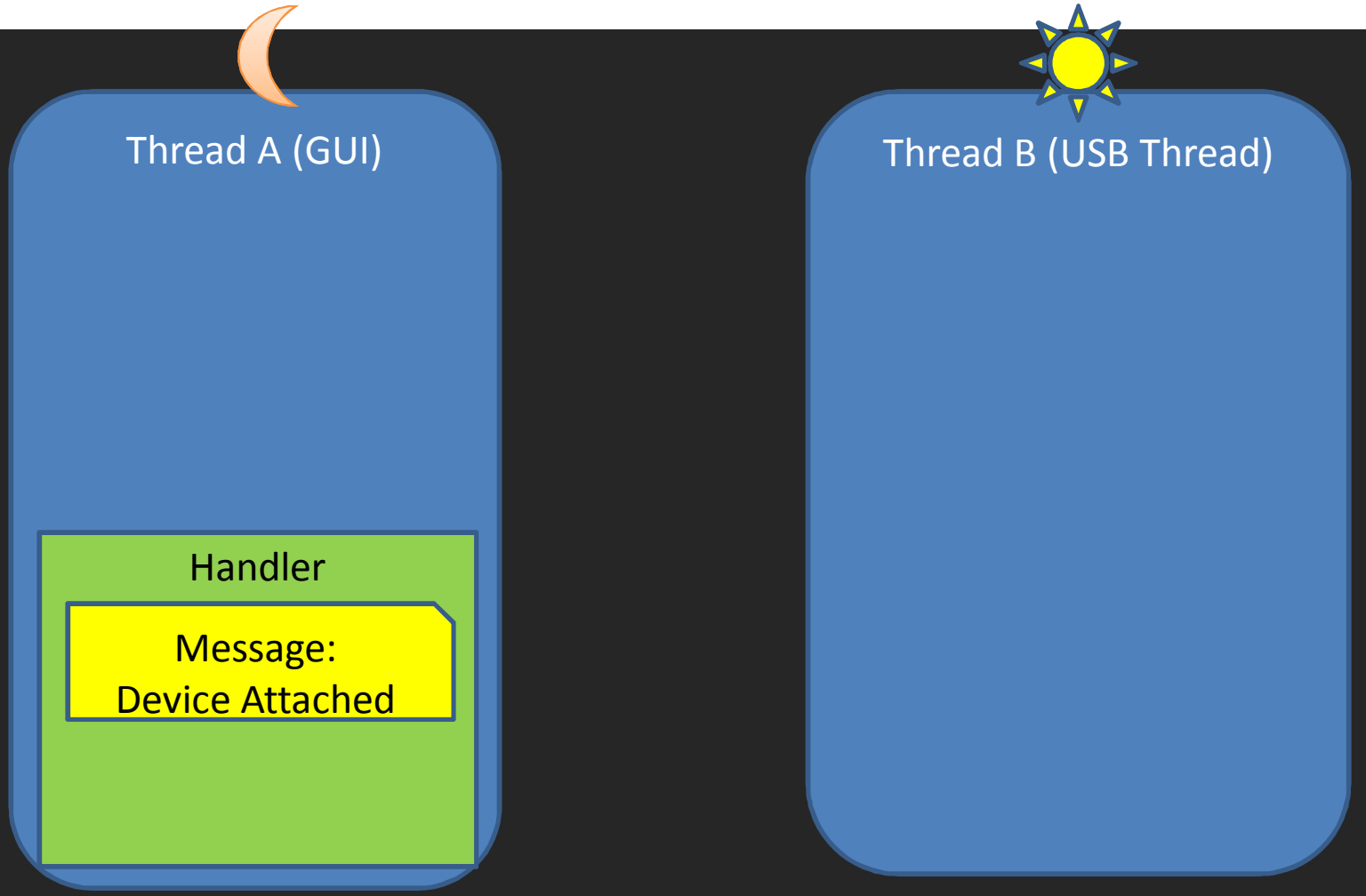
Threads:Обработчики и сообщения



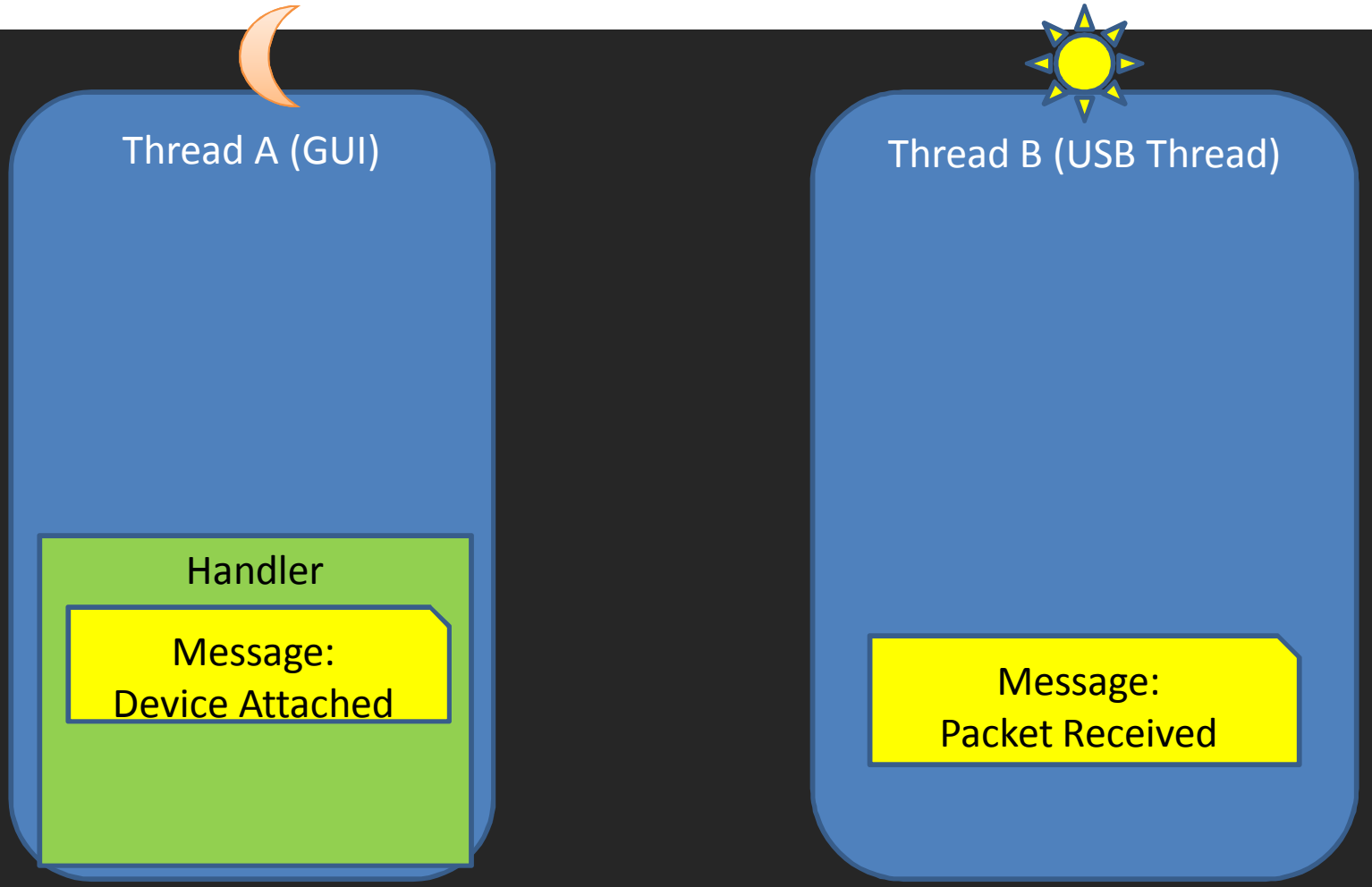
Threads:Обработчики и сообщения



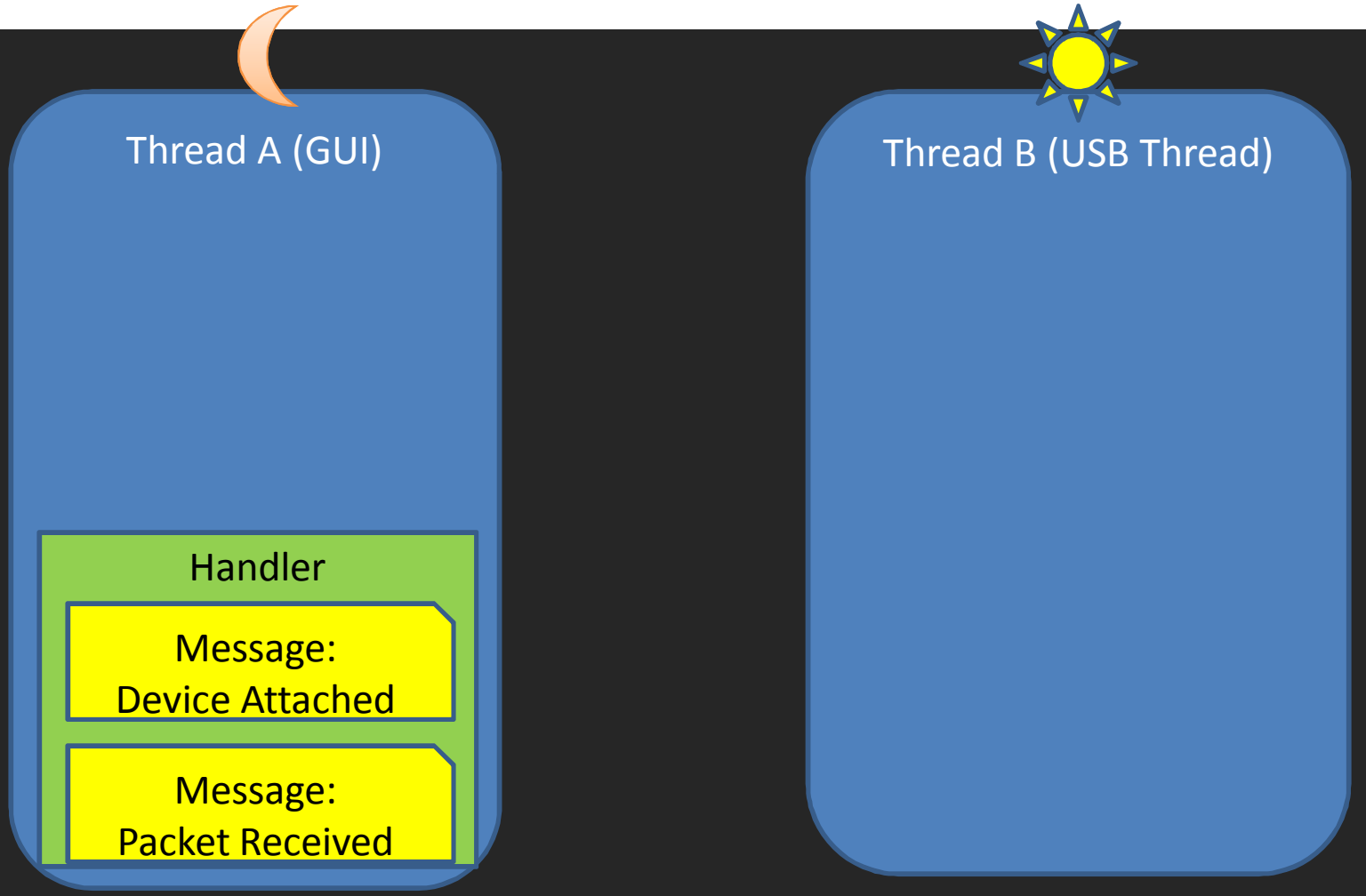
Threads:Обработчики и сообщения



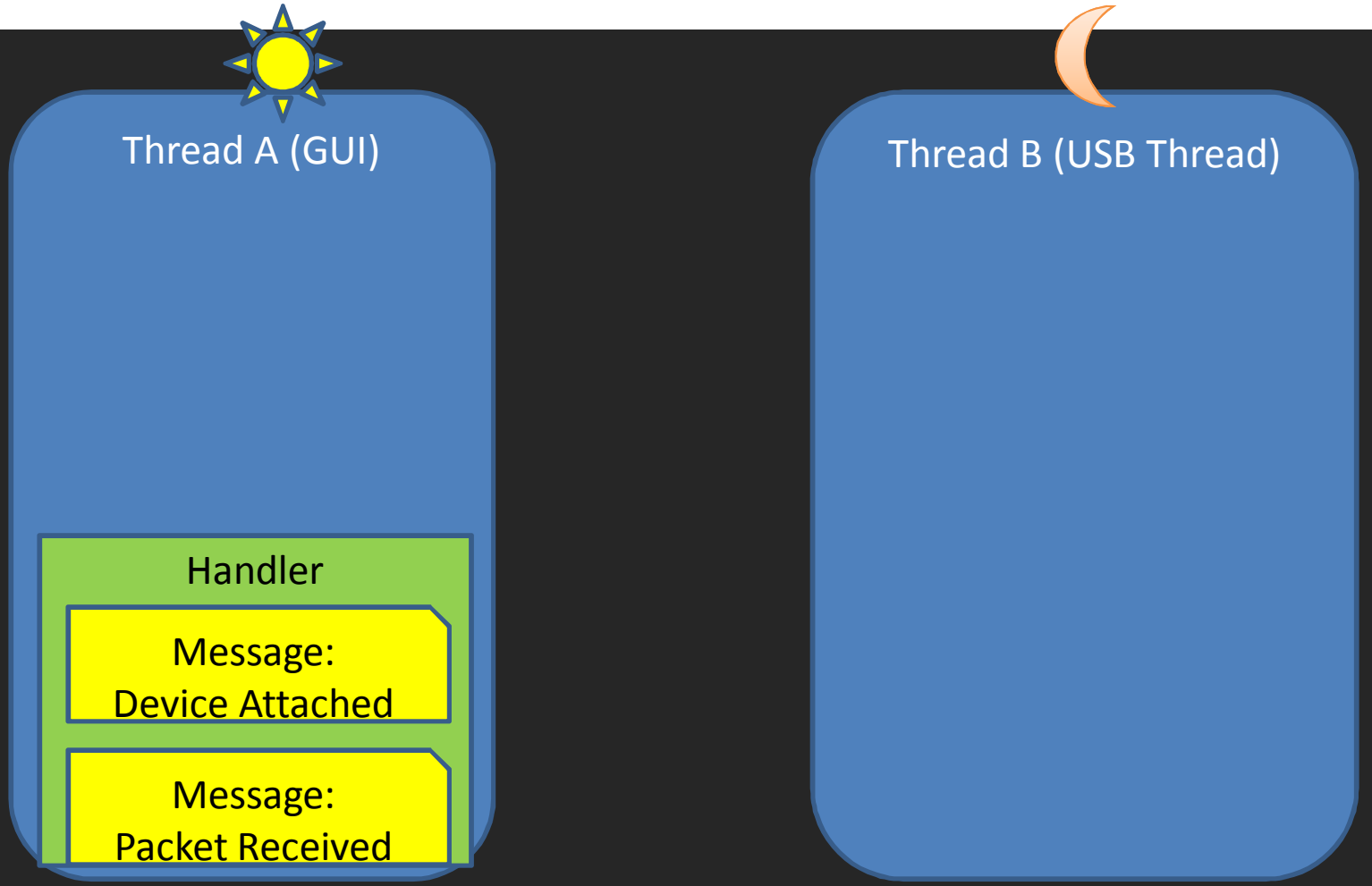
Threads:Обработчики и сообщения



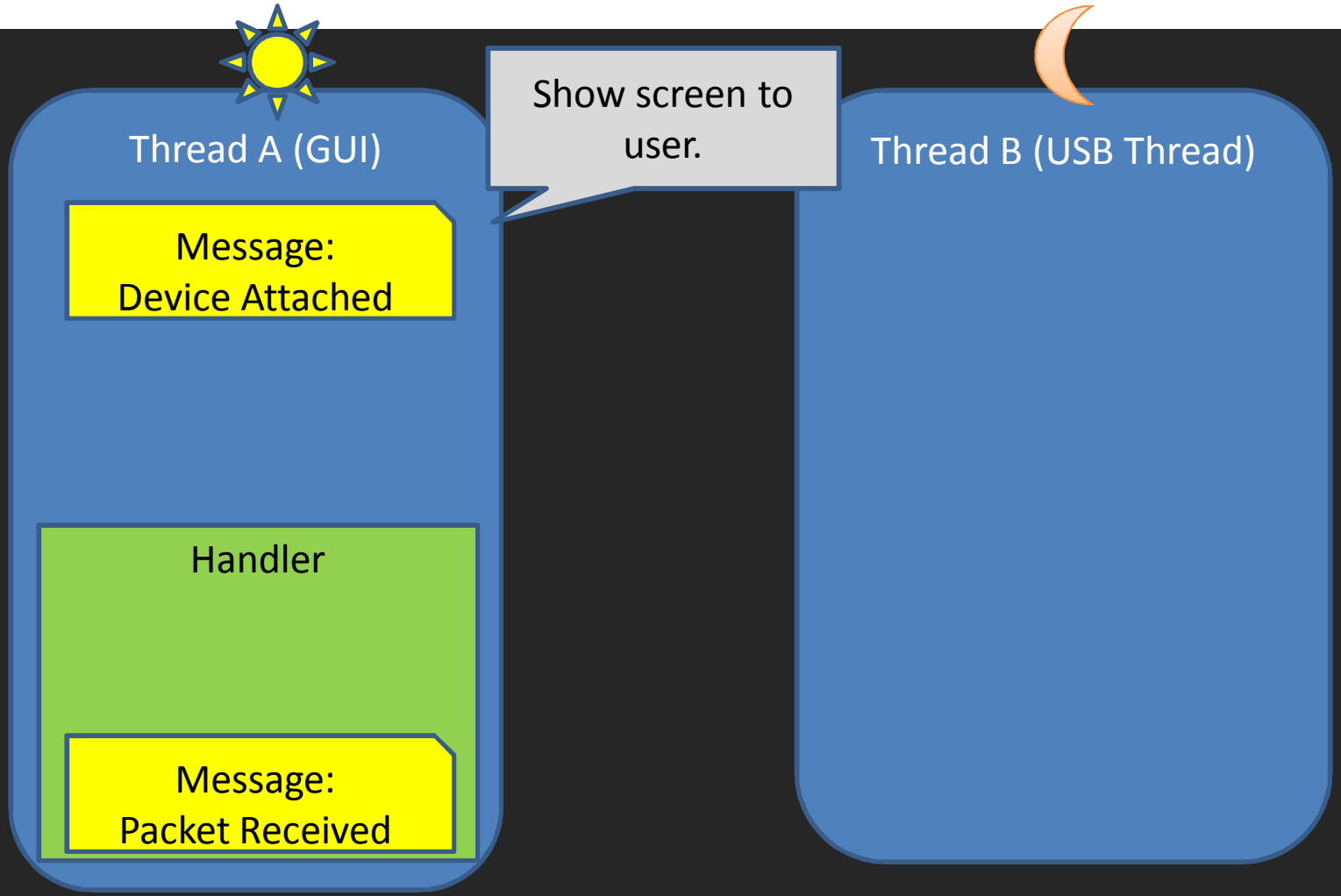
Threads:Обработчики и сообщения



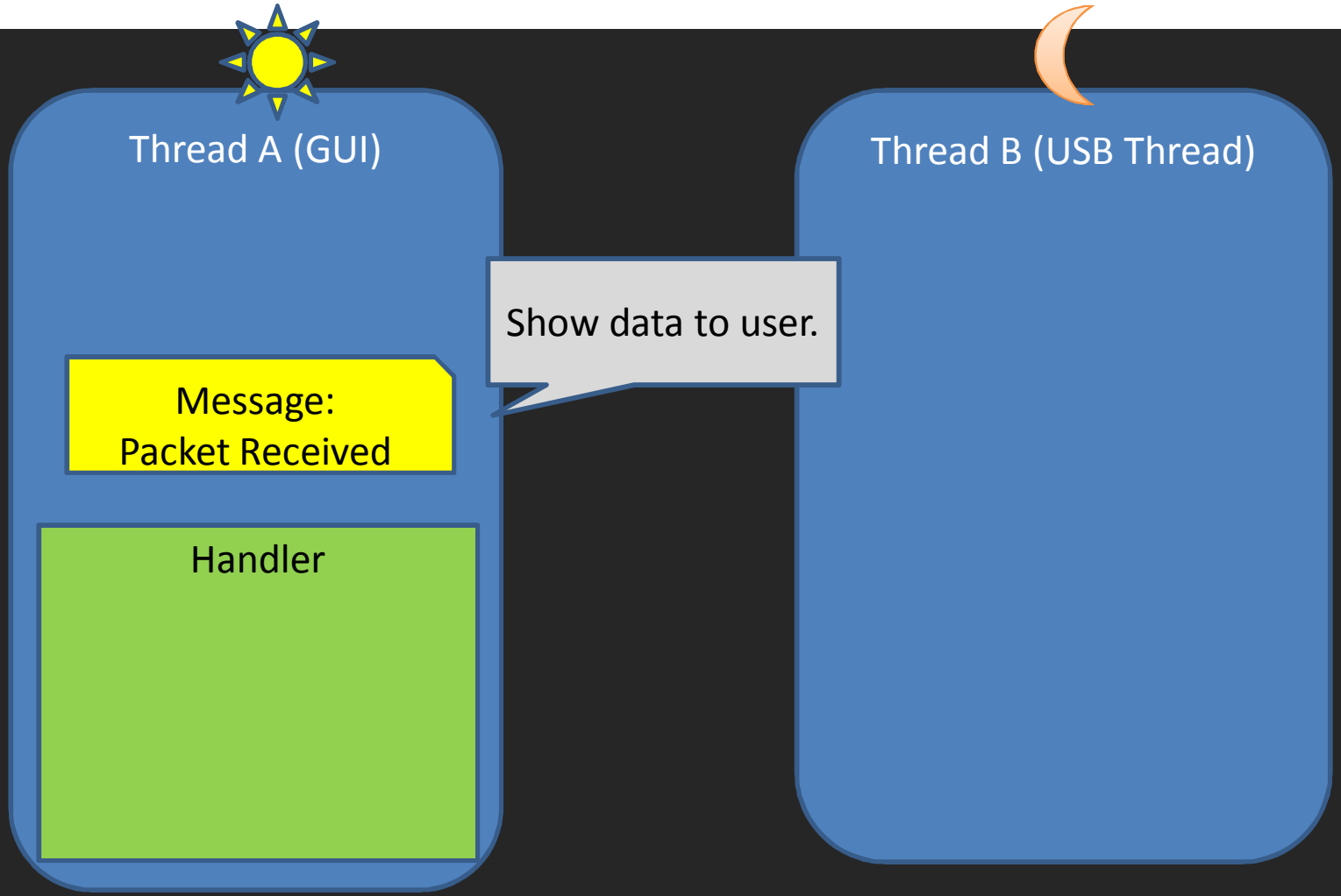
Threads:Обработчики и сообщения



Threads:Обработчики и сообщения



Threads:Обработчики и сообщения



Реализация сообщения

```
public class PushbuttonMessage {
   private boolean pressed = false;

   public PushbuttonMessage (boolean state) {
      pressed = state;
   }

   public boolean isPressed() {
      return pressed;
   }
}
```

Передача сообщения

```
private final static int BUTTON_EVENT = 1;

PushbuttonMessage pbMsg = new PushbuttonMessage(false);
Message msg = handler.obtainMessage(BUTTON_EVENT, pbMsg);
msg.sendToTarget();
```

Реализация обработчика

```
private final static int BUTTON_EVENT = 1;
private Handler handler = new Handler() {
   @Override
   public void handleMessage(Message msg) {
      switch(msg.what) {
      case BUTTON_EVENT:
            PushbuttonMessage pbMessage =
   (PushbuttonMessage) msg.obj;
            if(pbMessage.isPressed() == true) {
                  //do something here
            }
      }
   }
};
```

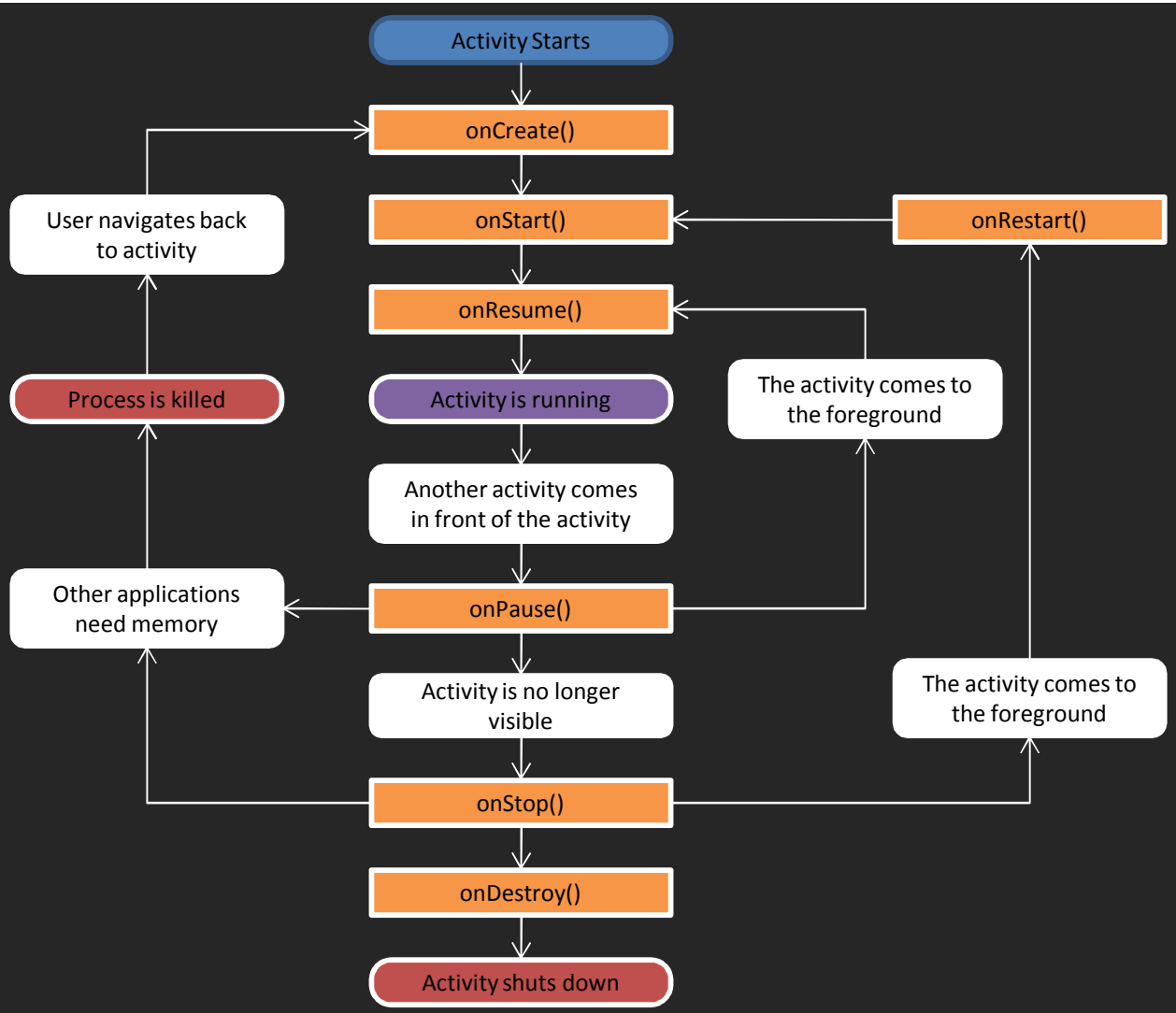
Жизненный цикл приложения

- Все приложения проходят через т.н. «жизненный цикл» (lifecycle) – набор различных состояний приложения
- Жизненным циклом приложения управляет ОС
- При изменении состояния жизненного цикла приложение должно адекватно
- Приложение не знает когда ОС запросит изменение состояния

Жизненный цикл приложения

- **Зачем мне это знать ?**
- Любые события вне приложения могут привести к смене состояния жизненного цикла (например, поворот устройства)
- Приложение может быть полностью закрыто и вновь открыто операционной системой
  - Требуется сохранять важные промежуточные данные при таком переходе
  - Приложения должны зарегистрировать в ОС доступ к «железу». Такой доступ может быть прекращен при изменениях цикла. Для повторного доступа нужна новая регистрация

Жизненный цикл приложения



android

Жизненный цикл приложения

- События изменения жизненного цикла могут быть обработаны
- Для этого необходимо добавить свой код в обработчик события:

```
@Override
public void onResume() {
        super.onResume();
        //your stuff here
}
```

Манифест

- «Манифест» – это XML файл AndroidManifest.xml, который «прикладывается» к каждому приложению
- В манифесте содержится информация о приложении, которую должна знать ОС до запуска приложения
- Информация из манифеста также влияет на фильтрацию в Google Play

Манифест

uses- feature

- Использование возможностей ОС (uses-feature)
  - В манифесте требуется указать, какие возможности ОС и «железа» использует приложение. (Например: Bluetooth, USB, Wi-Fi)
  - Можно указать, обязательно ли наличие данного «железа» для работы приложения, или опционально

Манифест

uses-permissions

- Использование разрешений (uses-permissions)
  - В ОС определен ряд потенциально «опасных» функций, работа с которыми требует получения разрешений. Например: выход в интернет, работа с Bluetooth, изменение конфигурации Bluetooth, изменение конфигурации WiFi и .т.д.
  - В манифесте приложения должны быть указаны такие функции, использующиеся в приложении.
  - Приложение может также запросить разрешение во время работы в виде диалога, позволяющего пользователю подтвердить разрешение.

Манифест

intent-filter

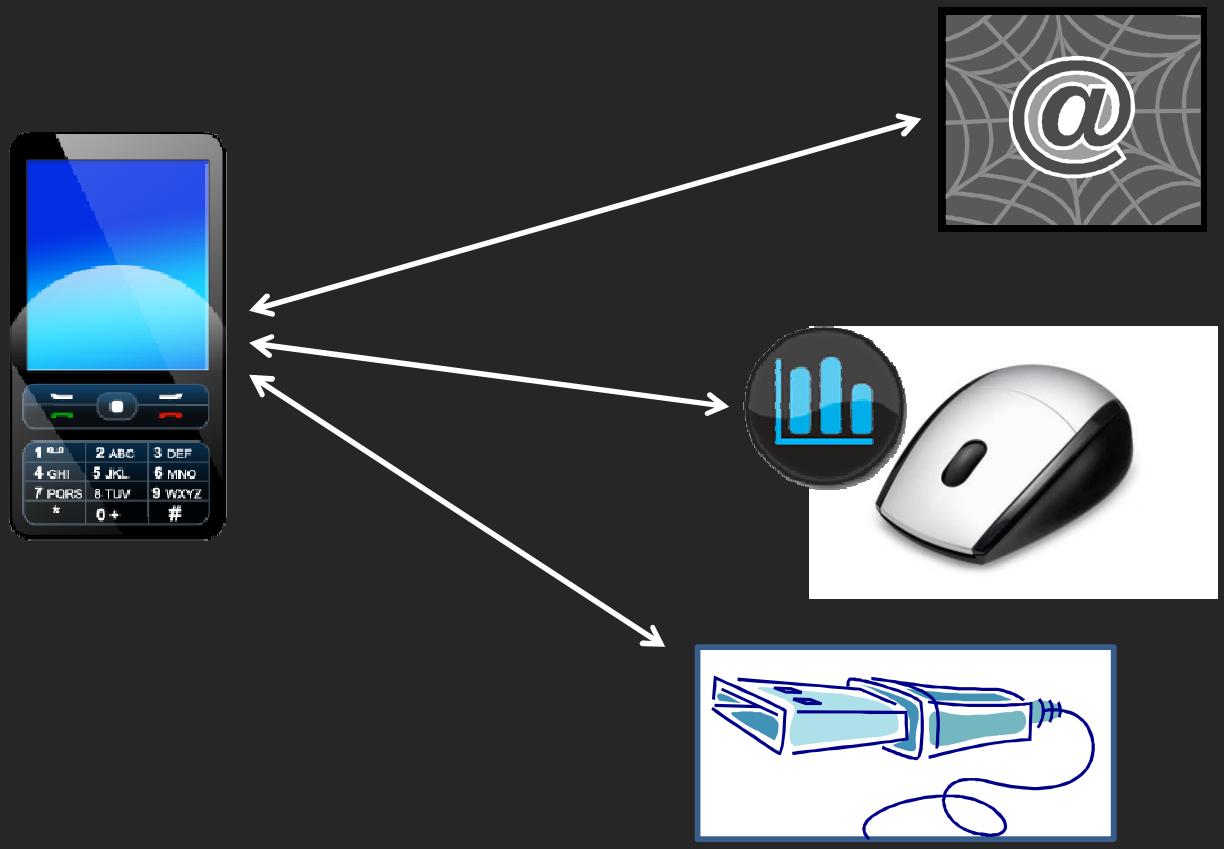
- Фильтр запросов (intent-fitler)
  - Взаимодействие ОС-приложение и приложение-приложение происходит через передачу запросов (Intents)
  - В манифесте приложения может быть указано какие запросы ОС приложение обрабатывает. (Например, запрос подключения устройства по USB).
  - Если системное событие указано в манифесте приложения, то оно будет передано приложению и может быть в нем обработано

Манифест

Пример

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     . . .
    <uses-feature android:name="andriod.hardware.usb.accessory" />
     . . .
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.INTERNET" />
     . . .
    <application
     <activity
          . . .
       <intent-filter>
          <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
        </intent-filter>
        <meta-data
           android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
           android:resource="@xml/accessory_filter" />
          . . .
      </activity>
          . . .
    </application>
</manifest>
```

Интерфейсы для подключения аксессуаров



Подключение аксессуаров:
По USB

USB: Что доступно?

- Штатная поддержка OS
  - Мышь, Клавиатура, и т.д.
- USB Host
  - Требует v3.1+ и устройство с поддержкой USB host
  - Аксессуар – USB периферия
- OpenAccessory Framework
  - Требует v3.1+ or v2.3.4+
  - Аксессуар – USB хост

OpenAccessory

- USB Accessory Protocol v1 (v2.3.4+, v3.1+)
  - Аксессуар-хост, Android - устройство
  - Использует vendor-class драйвер
    - 1 bulk endpoint in
    - 1 bulk endpoint out
- Microchip предлагает библиотеку, реализующую работу аксессуара в Open Accessory Framework

OpenAccessory
API библиотеки Microchip

```
void AndroidAppStart(ANDROID_ACCESSORY_INFORMATION *info)
```

```
BYTE AndroidAppWrite(void* handle, BYTE* data, DWORD size)
```

```
BOOL AndroidAppIsWriteComplete(void* handle, BYTE* errorCode, DWORD* size)
```

```
BYTE AndroidAppRead(void* handle, BYTE* data, DWORD size)
```

```
BOOL AndroidAppIsReadComplete(void* handle, BYTE* errorCode, DWORD* size)
```

```
BOOL USB_ApplicationEventHandler(      BYTE address, USB_EVENT event,
                                       void *data, DWORD size )
```

```
void AndroidTasks(void)
```

- **Новое событие:** EVENT_ANDROID_ATTACH
  - Data = handle на устройство Android
    - Этот handle используется в дальнейшем для всех других функций !!

OpenAccessory AndriodAppStart

- Функция AndroidAppStart() должна быть вызвана до подключения устройства Android
- Параметр info должен содержать информацию, совпадающую с той, которая объявлена в приложении Android

```
typedef struct
{
  char* manufacturer;
  BYTE manufacturer_size;
  char* model;
  BYTE model_size;
  char* description;
  BYTE description_size;
  char* version;
  BYTE version_size;
  char* URI;
  BYTE URI_size;
  char* serial;
  BYTE serial_size;
} ANDROID_ACCESSORY_INFORMATION;
```

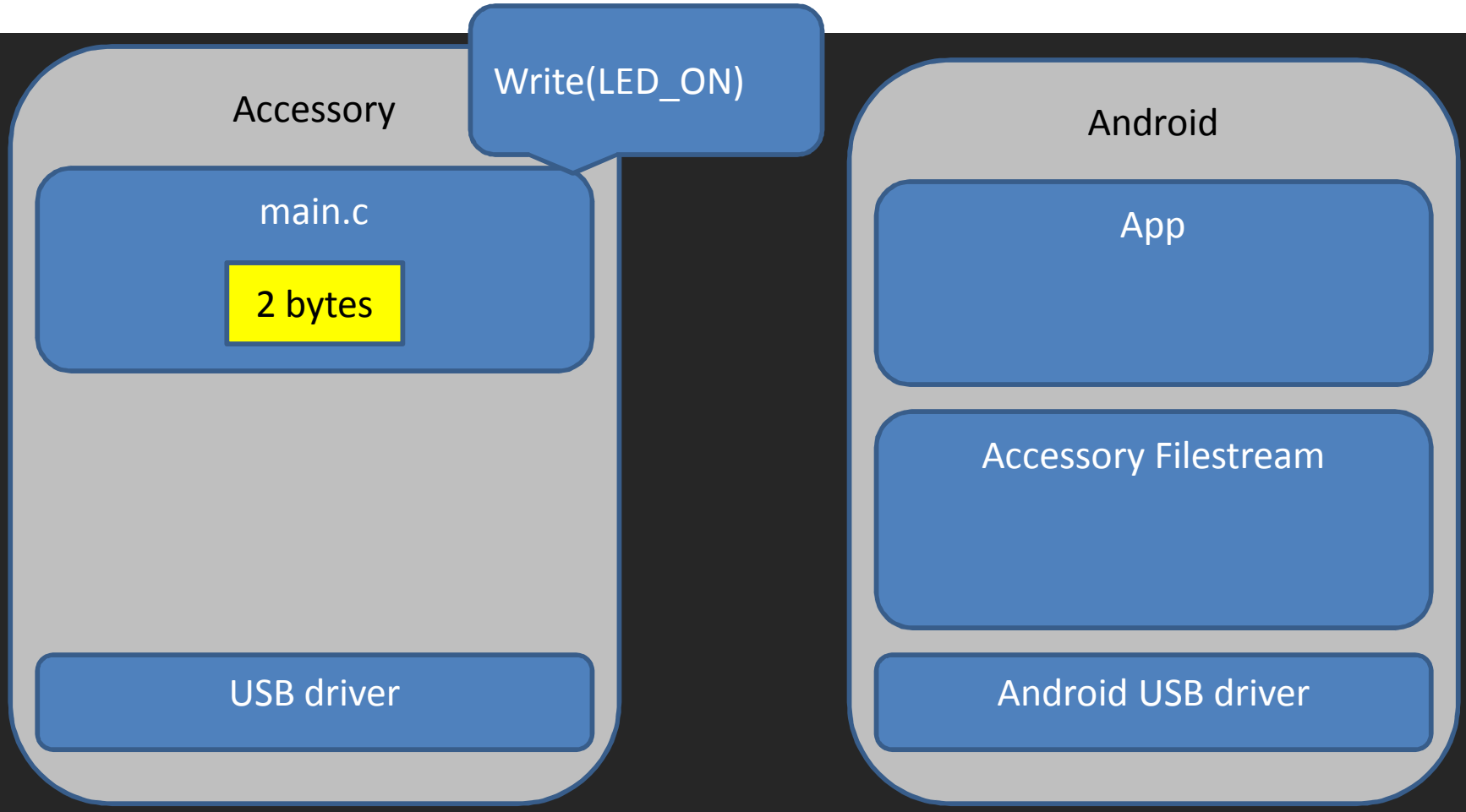
```
char manufacturer[] = "Microchip Technology Inc.";
char model[] = "Basic Accessory Demo";
char description[] = "PIC24F Android Starter kit";
char version[] = "1.0";
char uri[] = "http://www.microchip.com/android";
char serial[] = "N/A";

ANDROID_ACCESSORY_INFORMATION  myDeviceInfo =
{
  manufacturer,   sizeof(manufacturer),
  model,   sizeof(model),
  description,  sizeof(description),
  version,   sizeof(version),
  uri,   sizeof(uri),
  serial,   sizeof(serial)
};
```

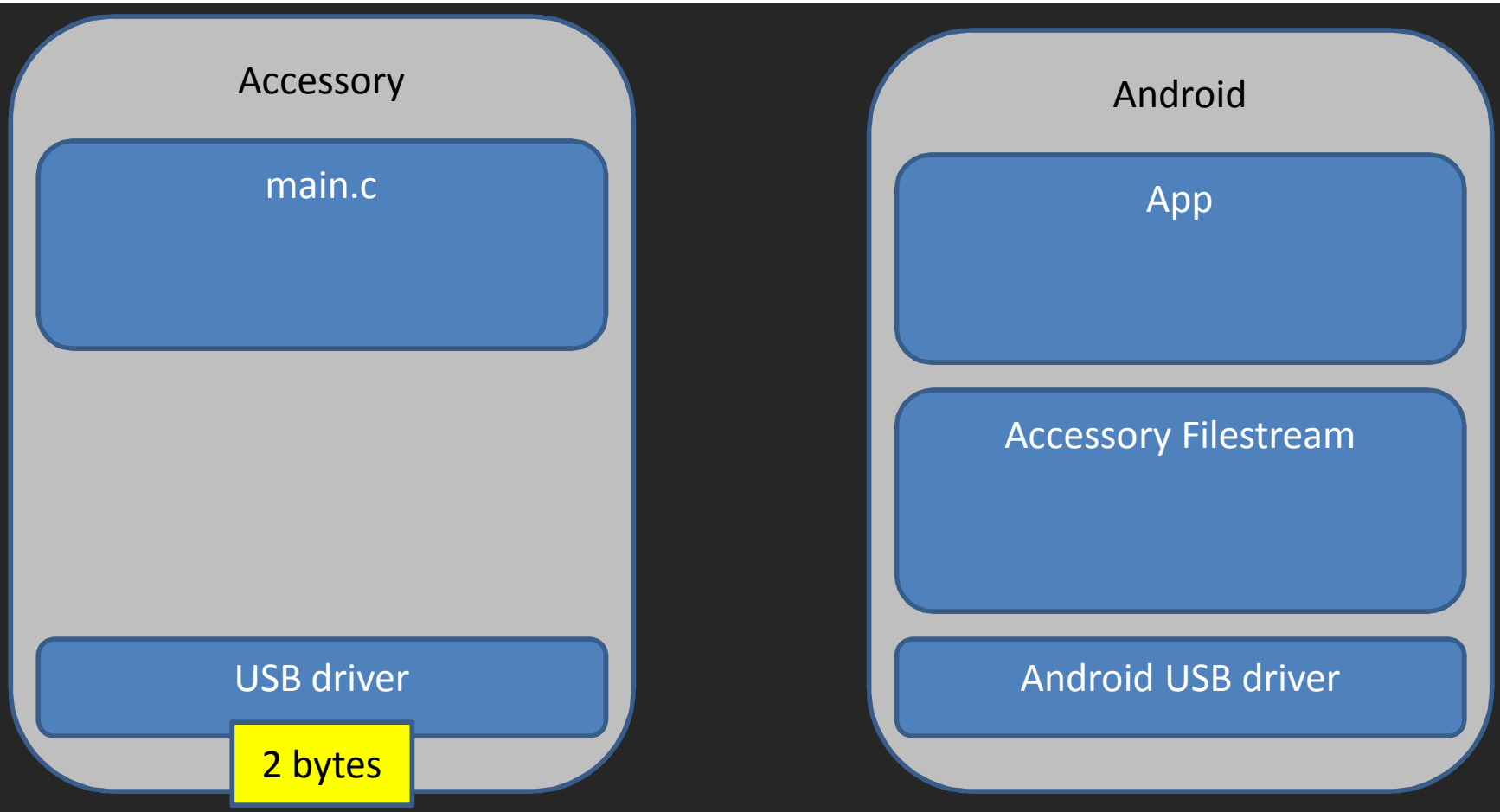
Filestreams

- Аксессуар взаимодействует с Andrioid через файловые потоки (Filestreams)
- Filestreams работают с данными как с потоком
  - Реальное разбиение по пакетам не видно приложению

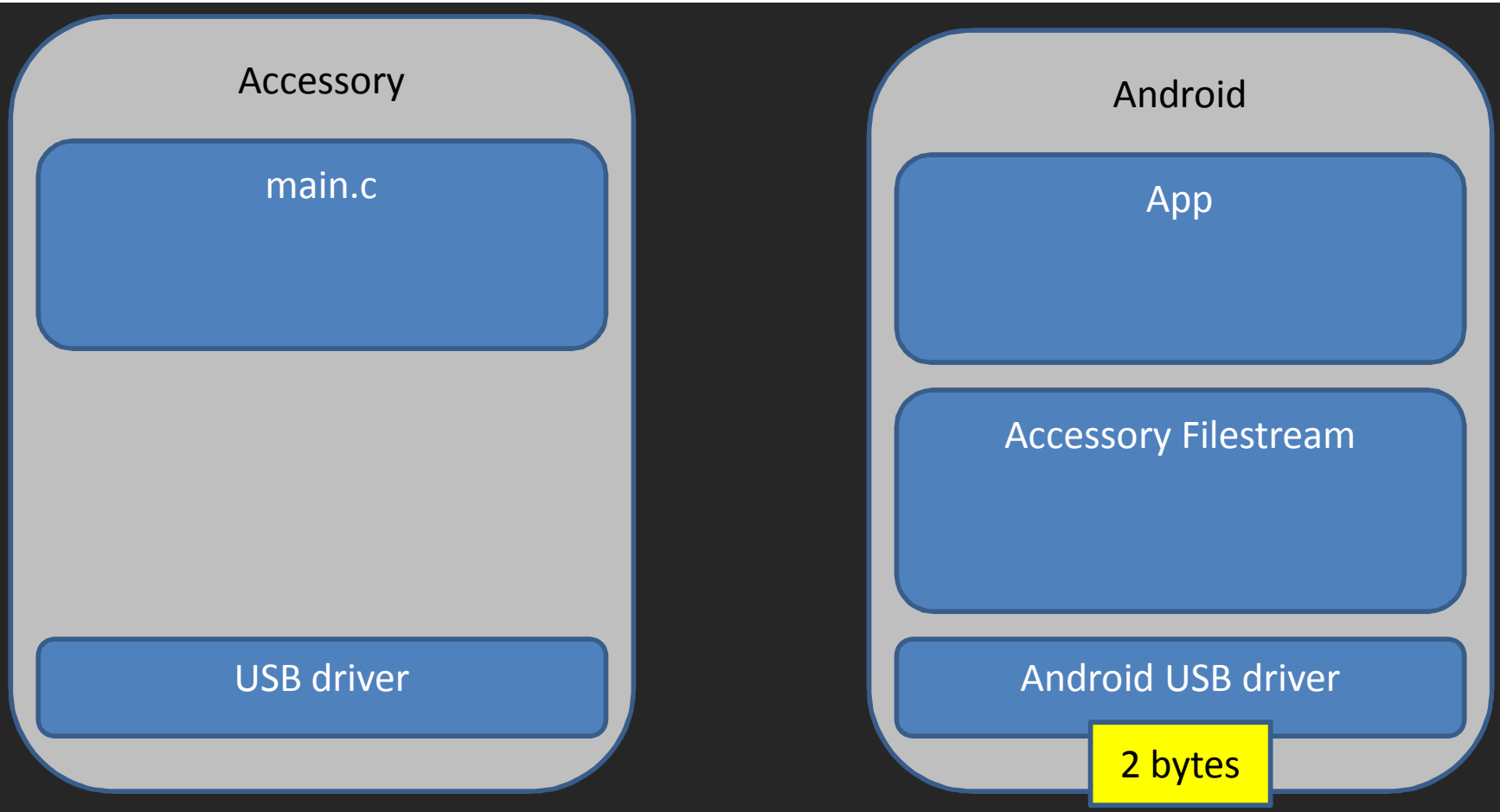
Filestreams: Packing



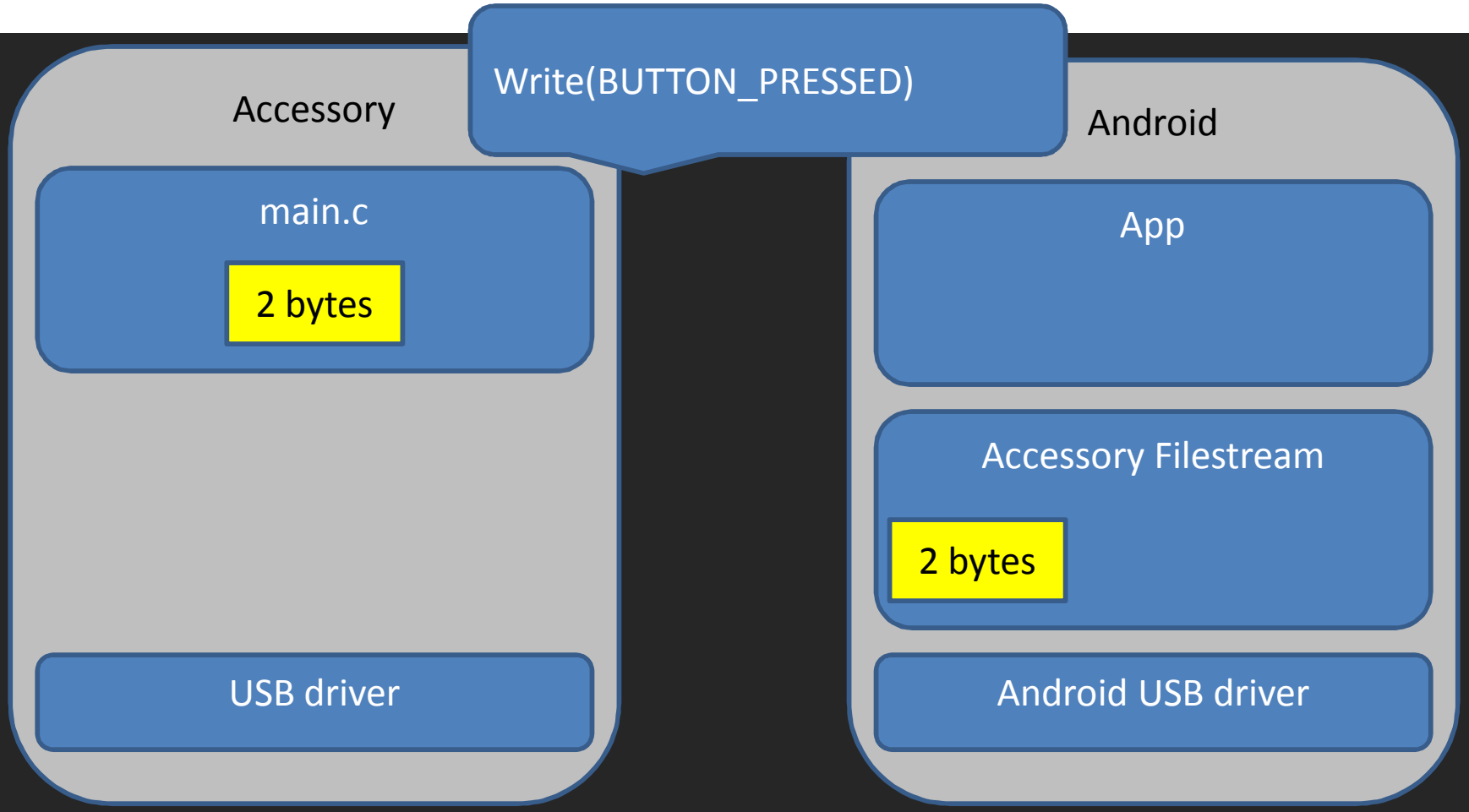
Filestreams: Packing



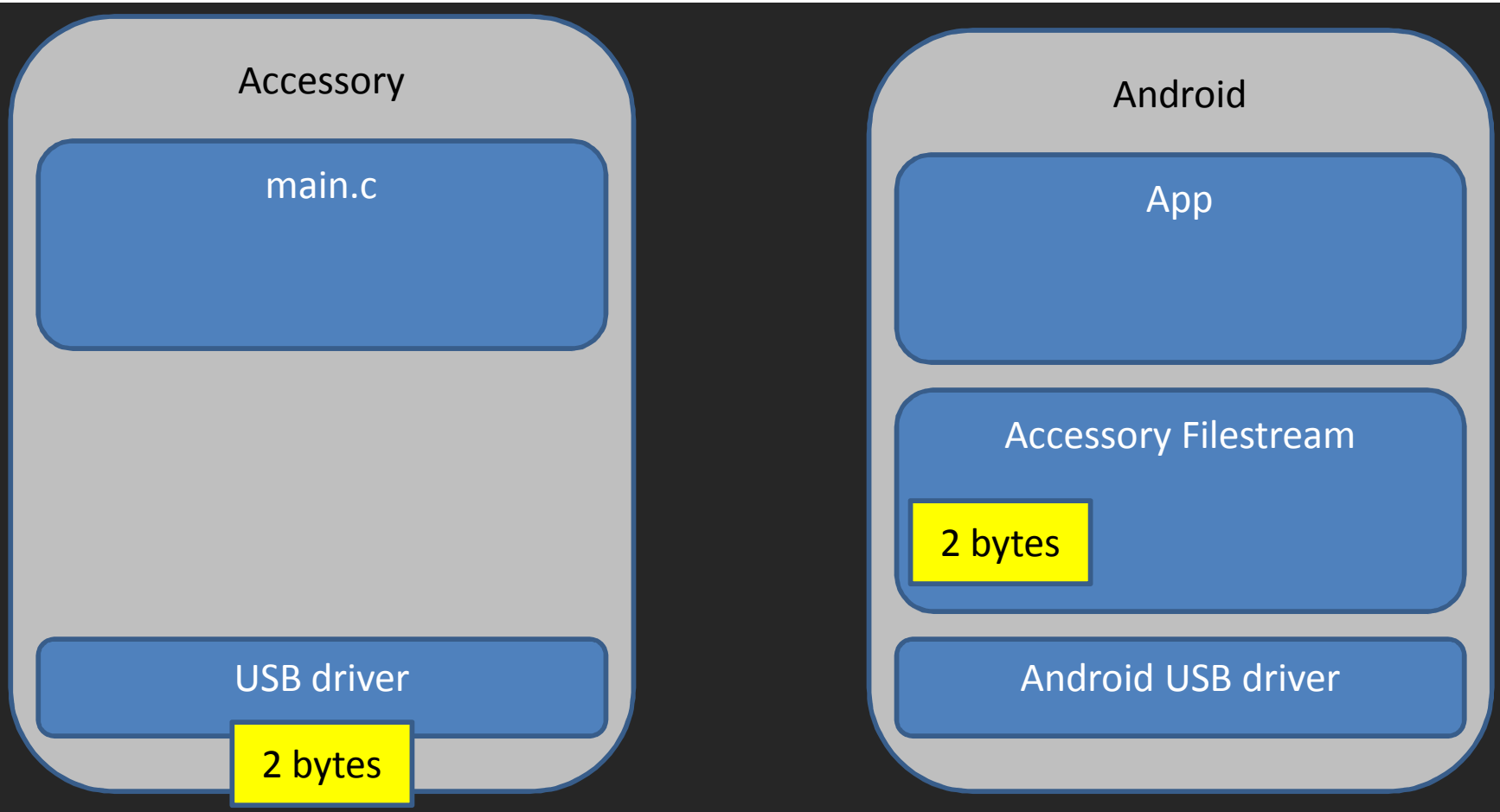
Filestreams: Packing



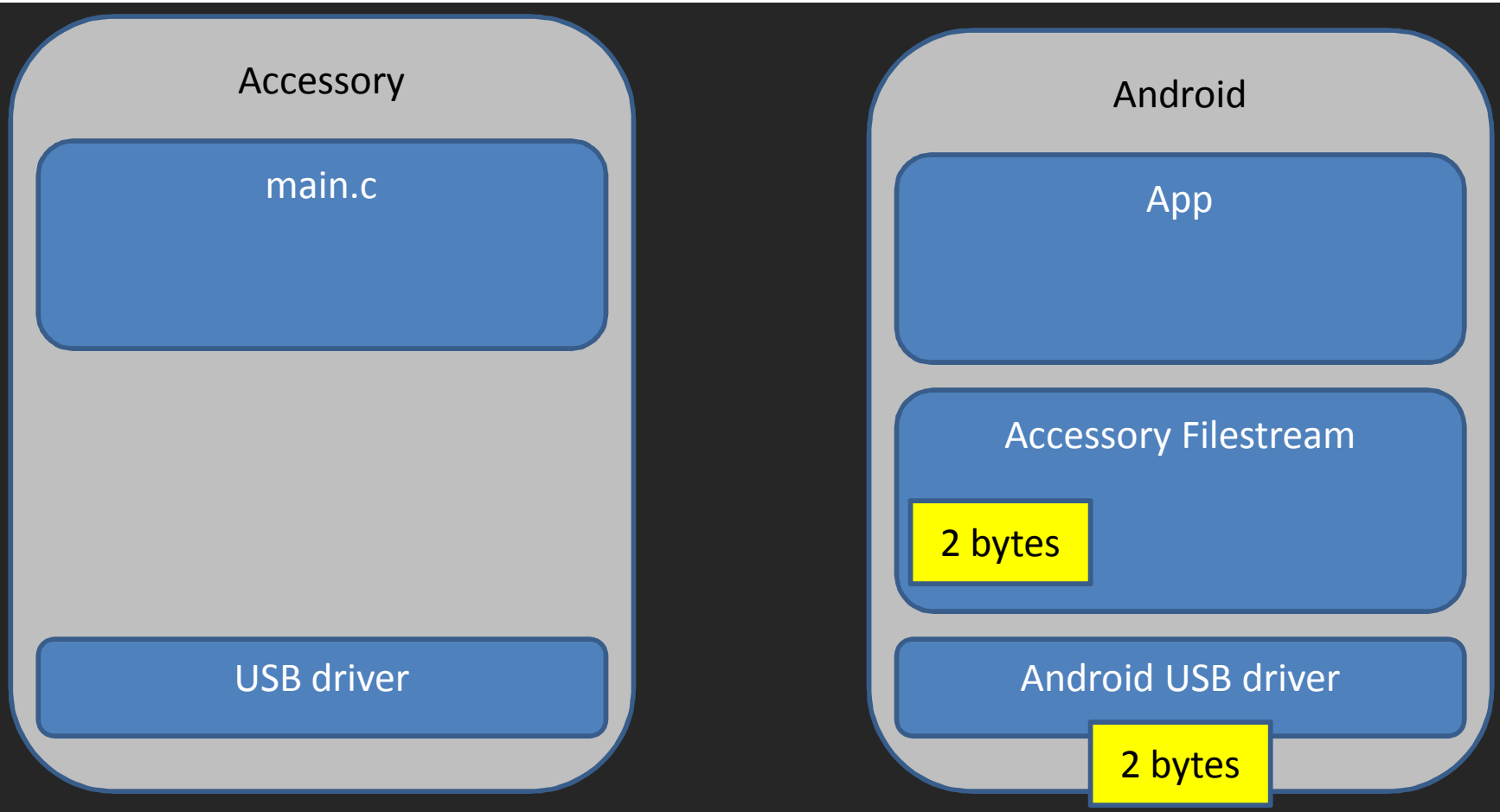
Filestreams: Packing



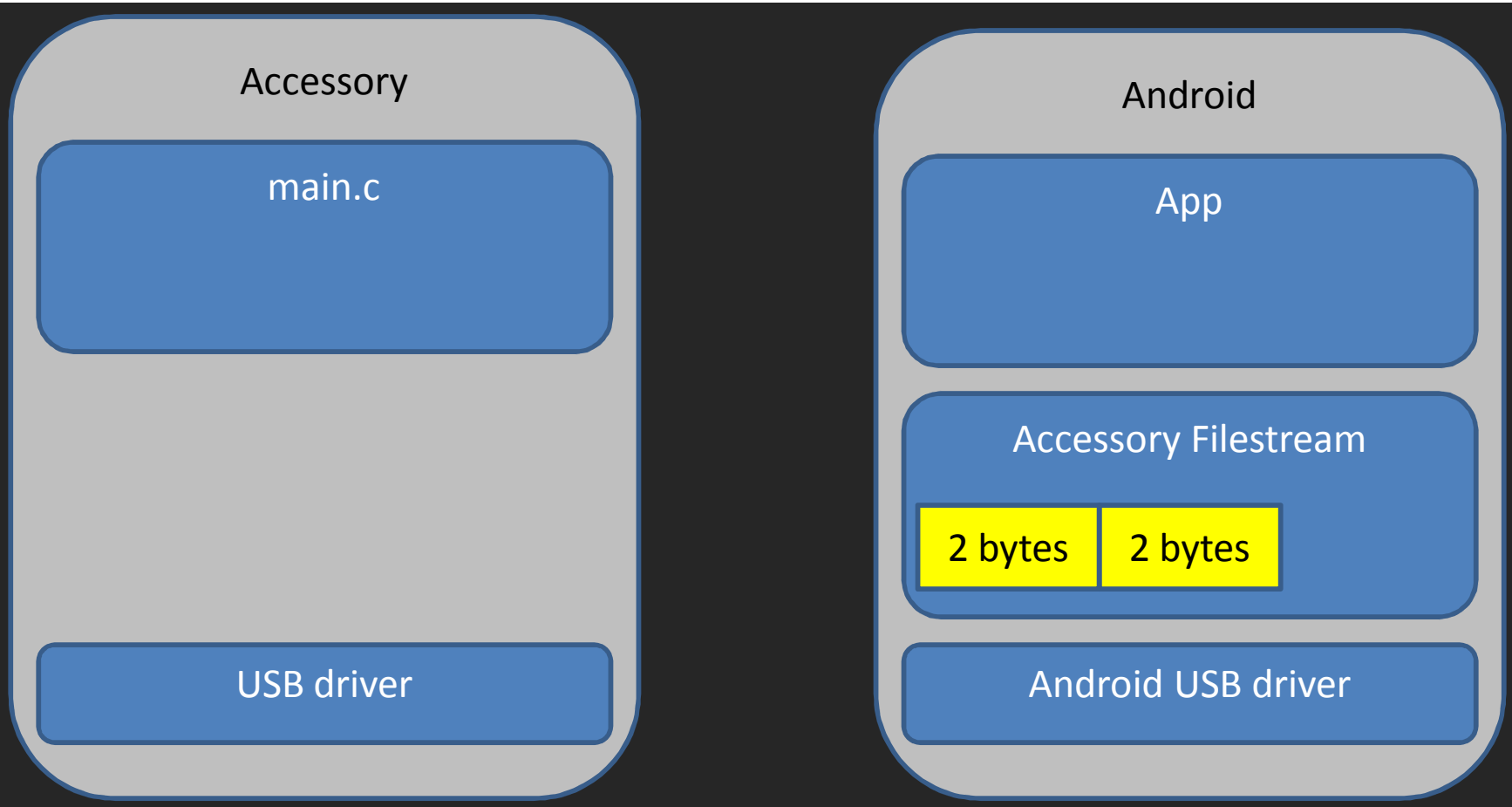
Filestreams: Packing



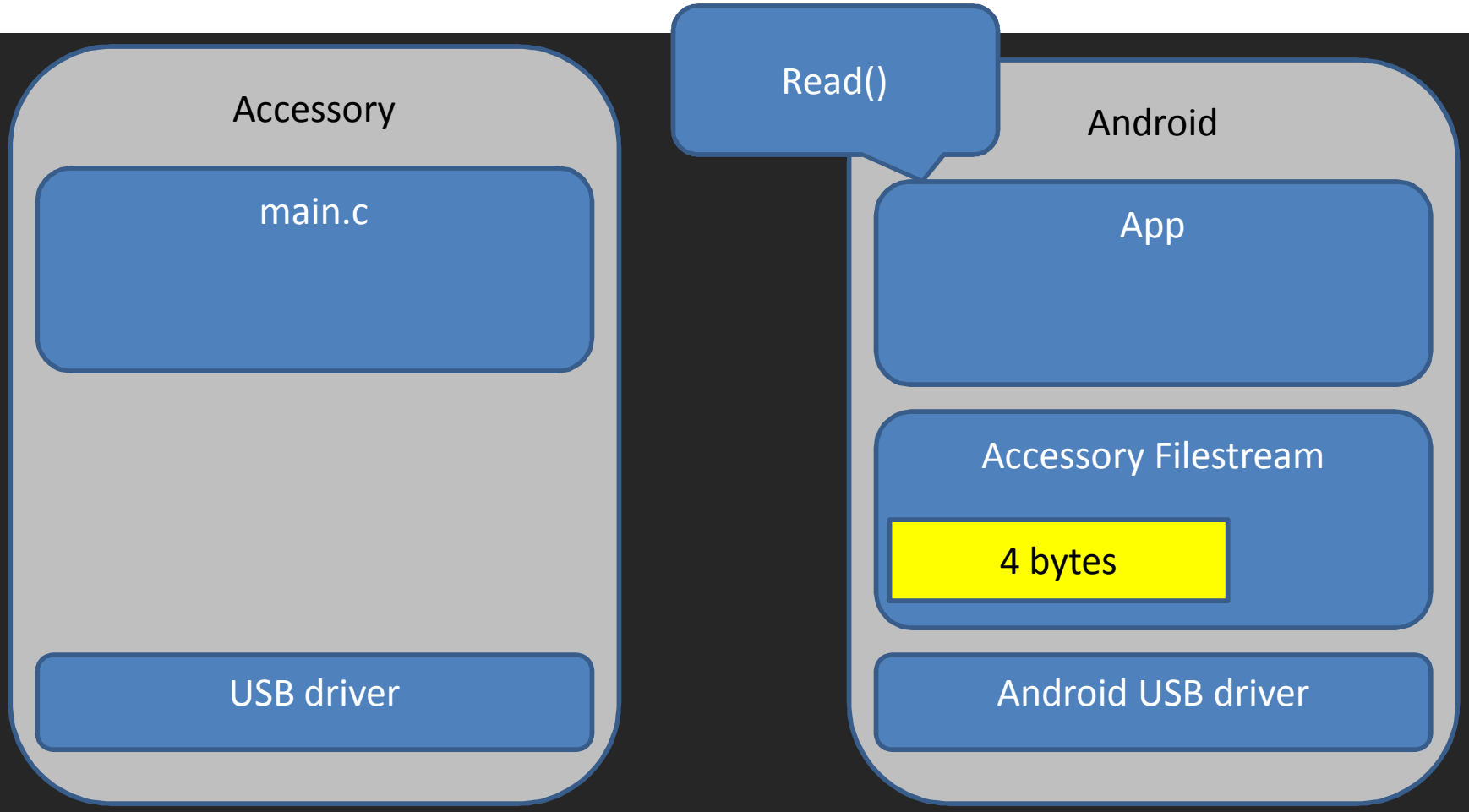
Filestreams: Packing



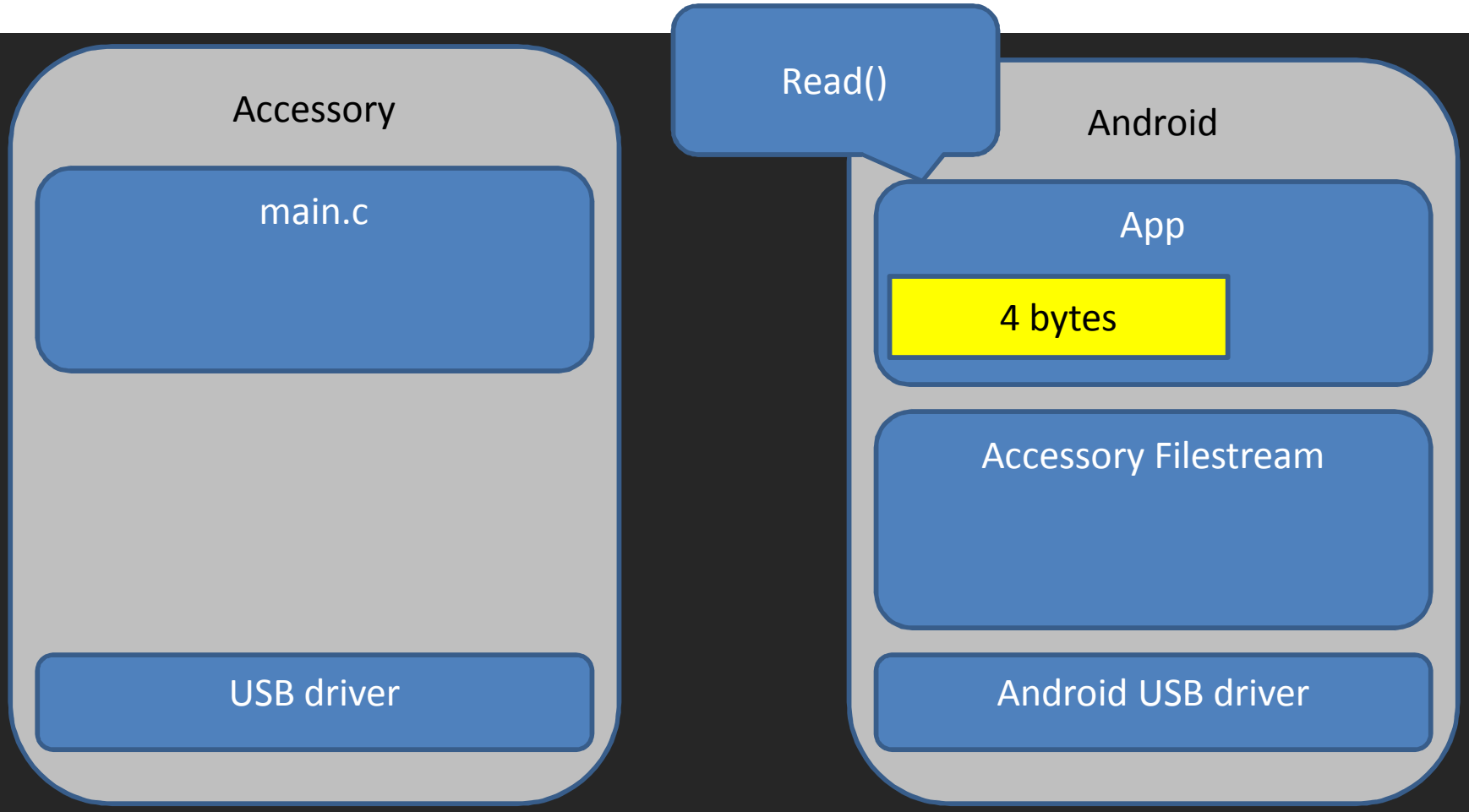
Filestreams: Packing



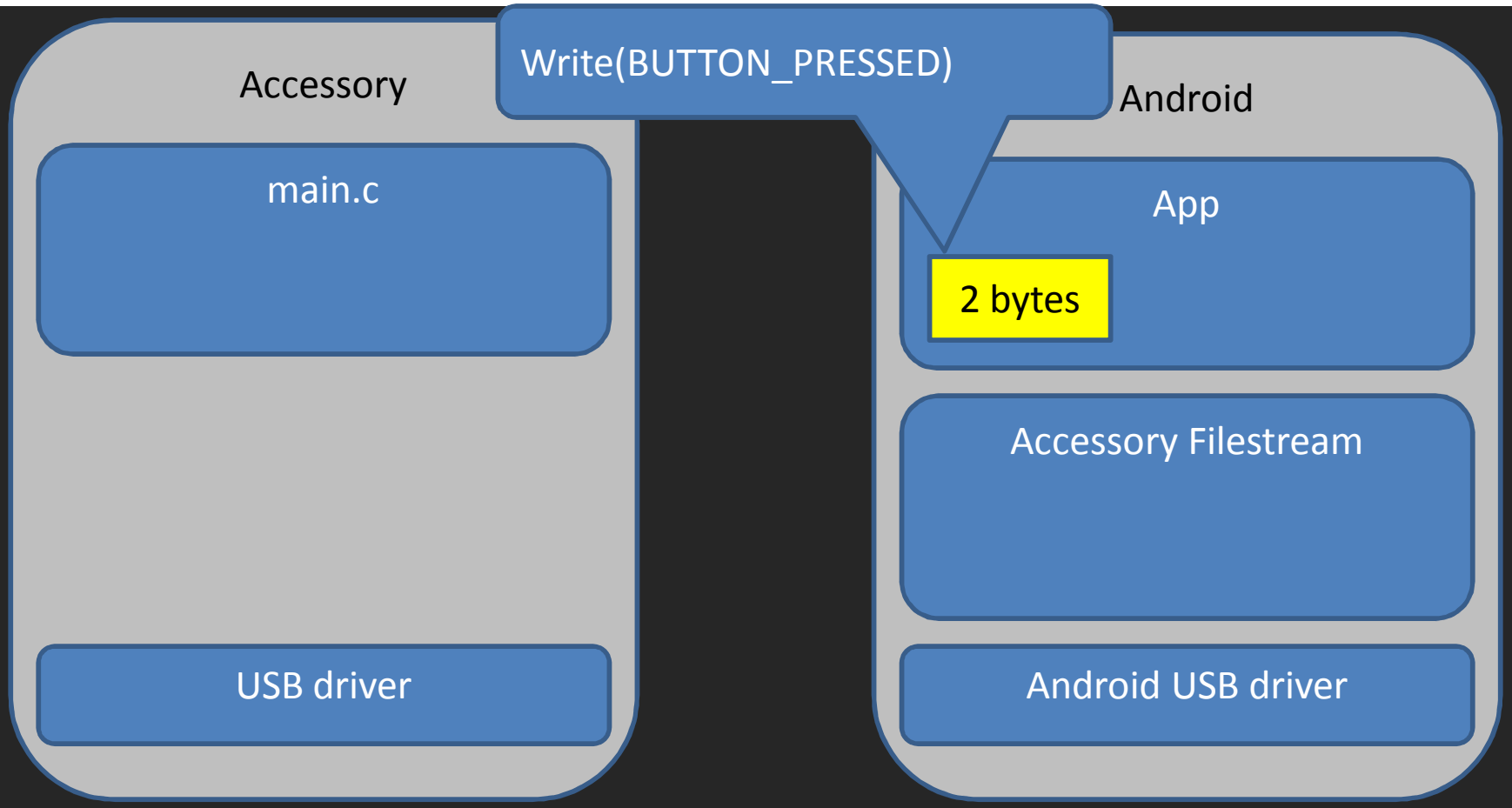
Filestreams: Packing



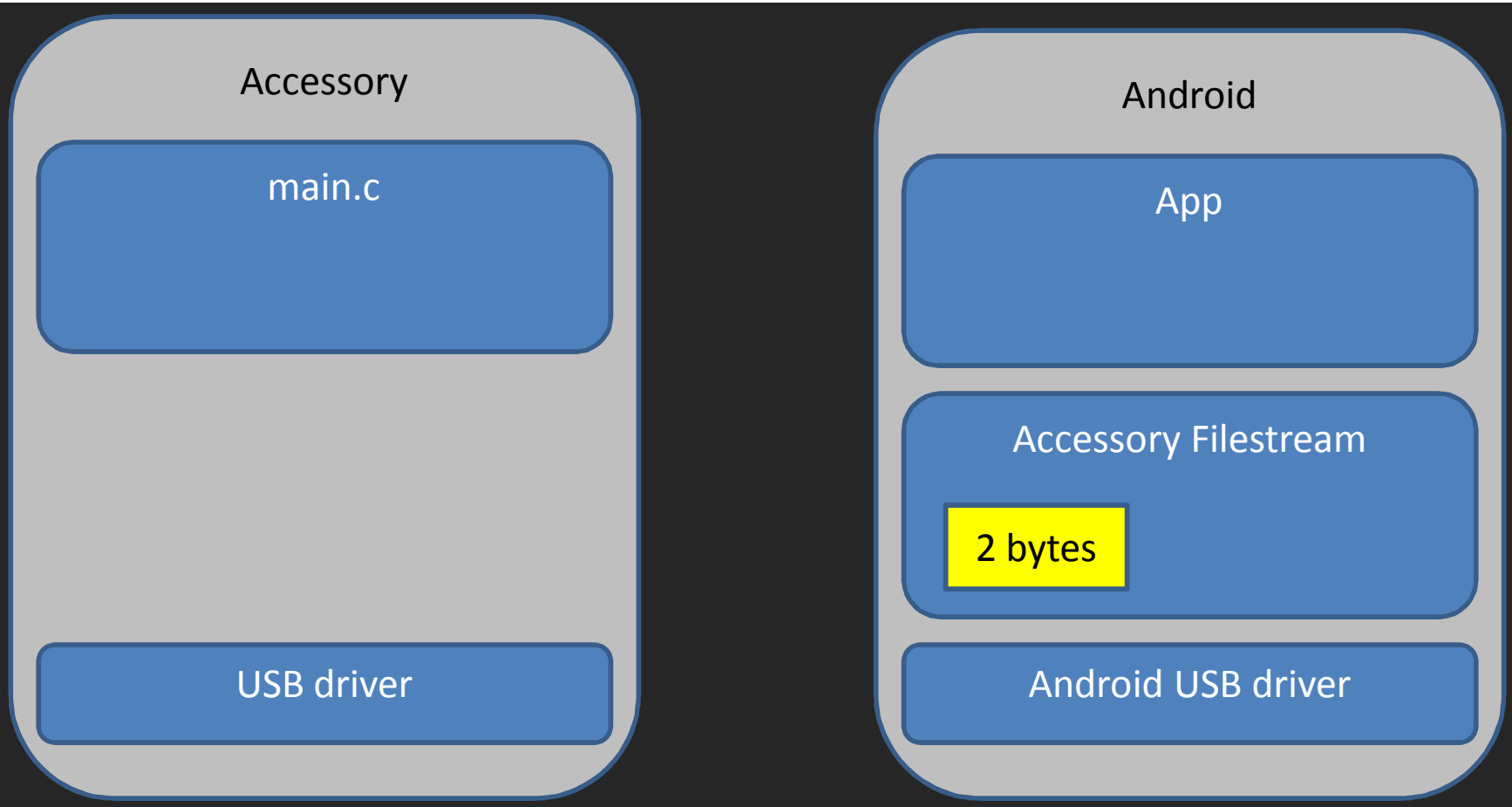
Filestreams: Packing



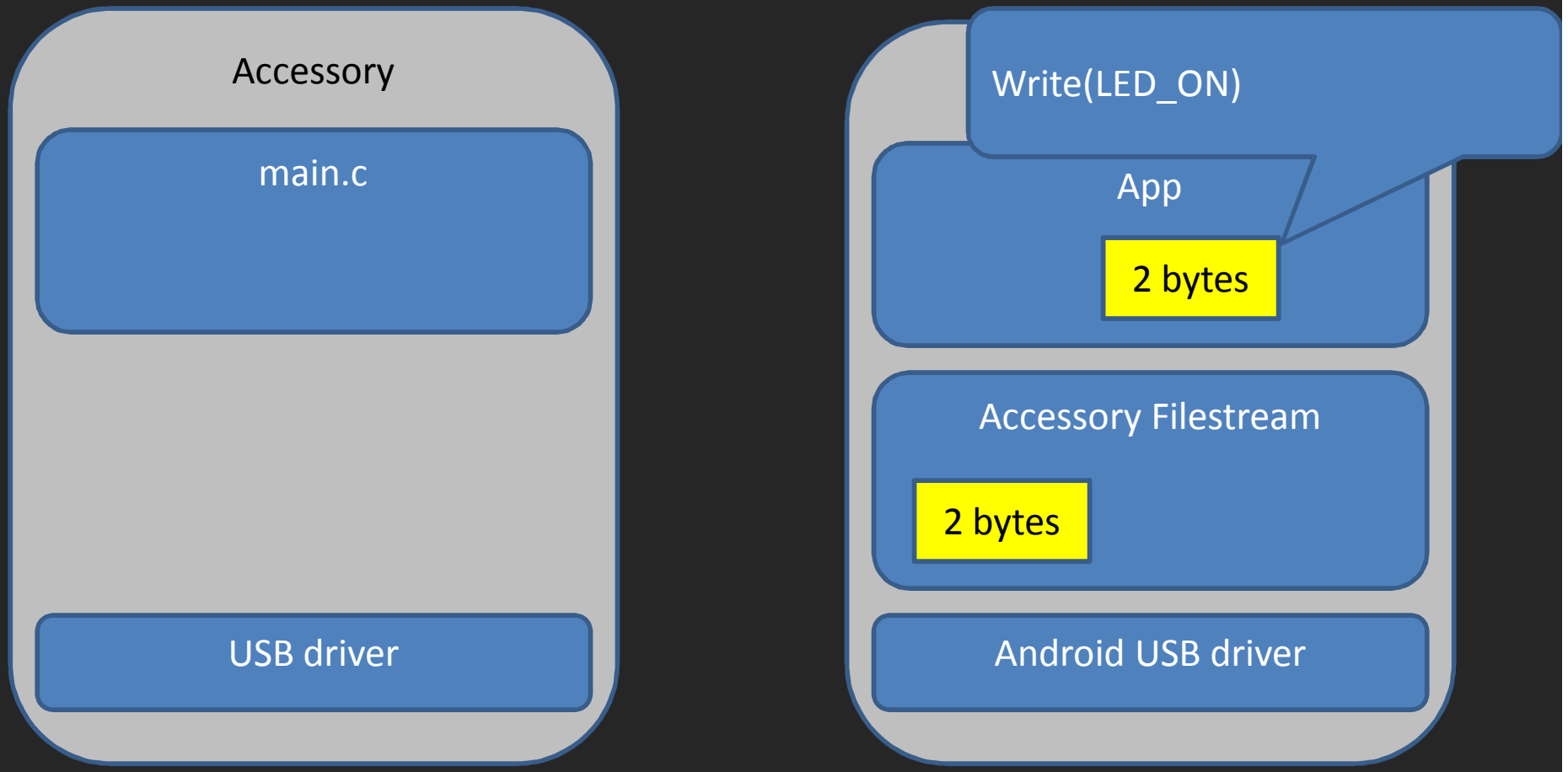
Filestreams: Packing



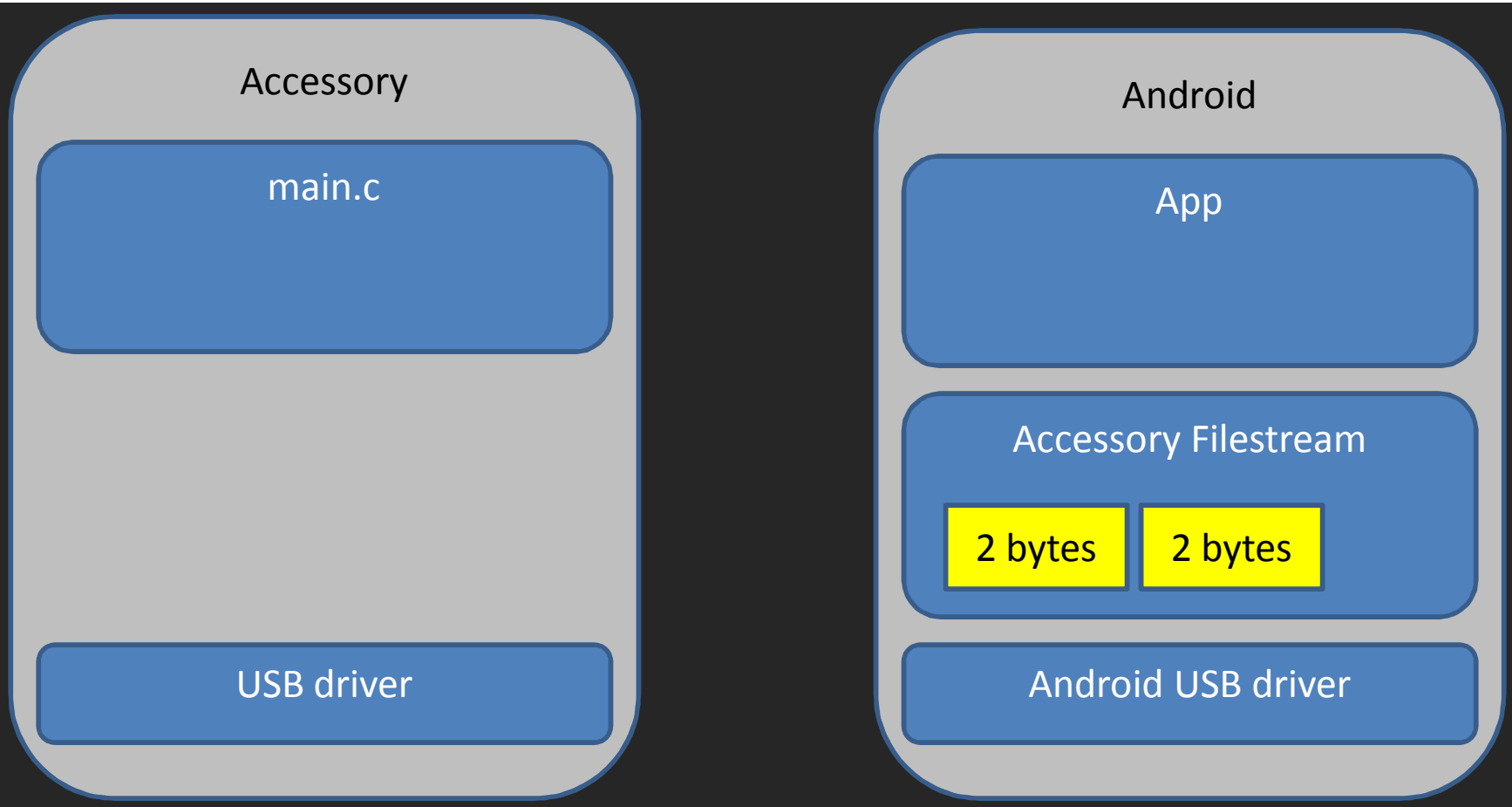
Filestreams: Packing



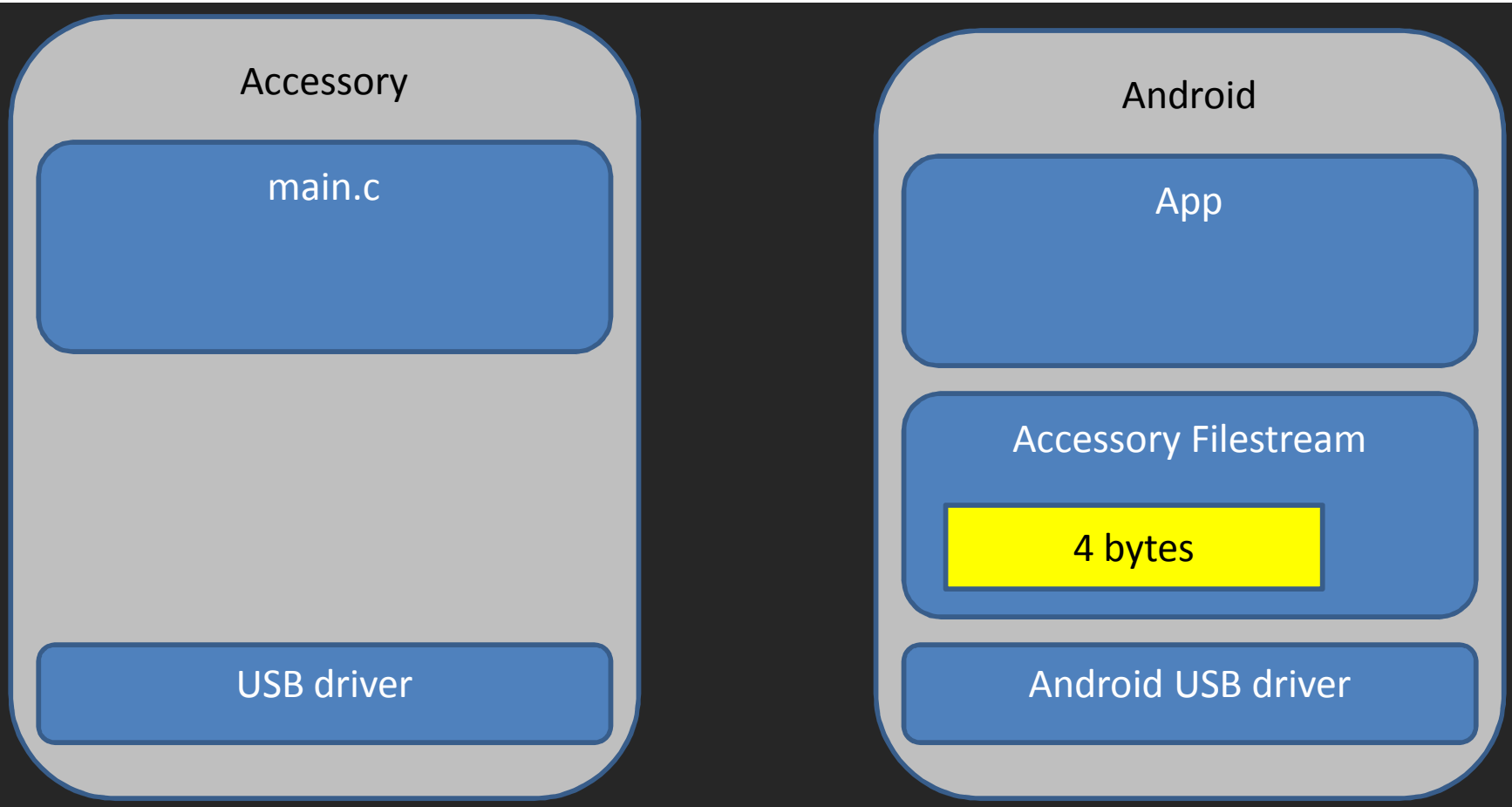
Filestreams: Packing



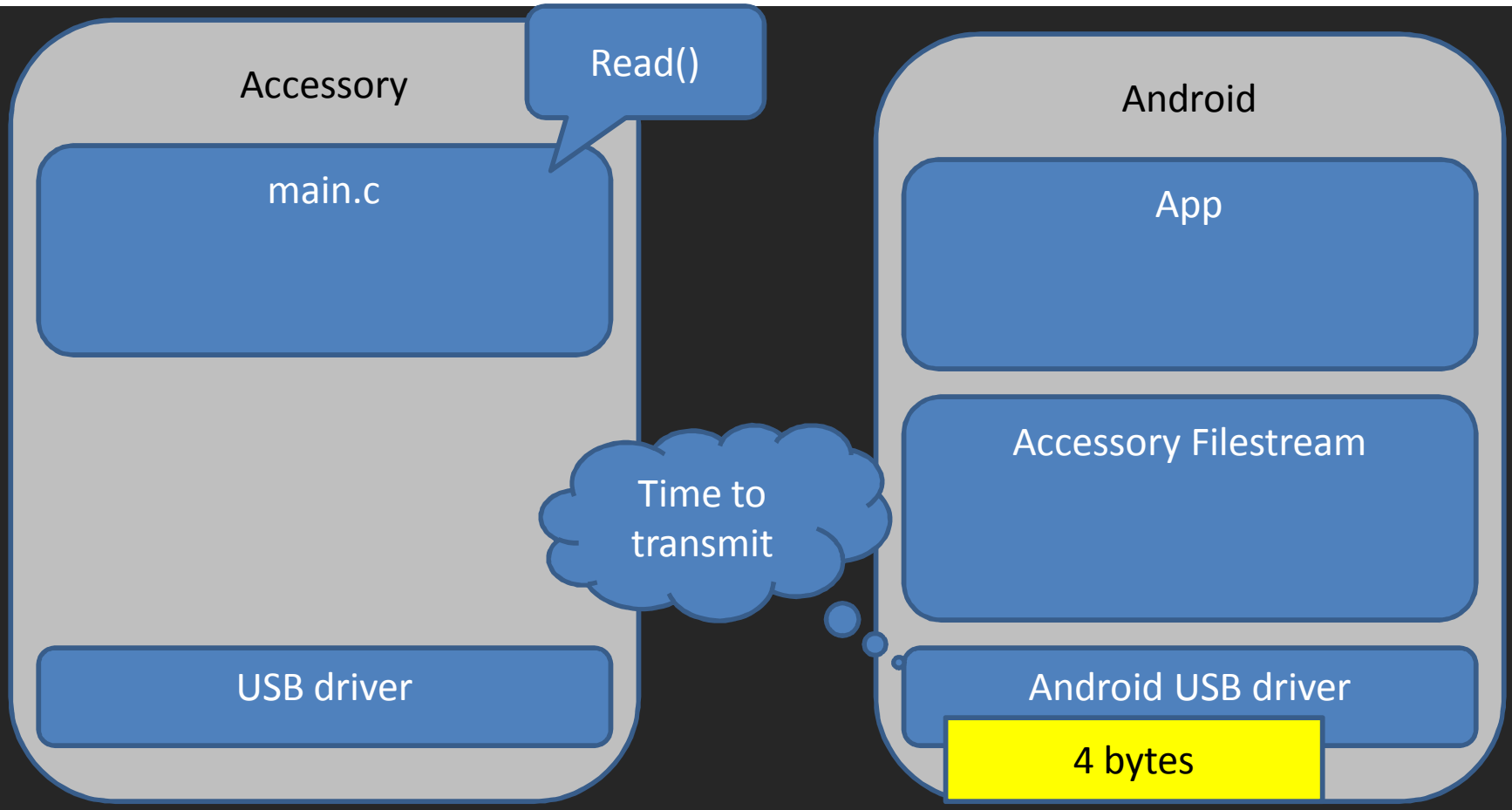
Filestreams: Packing



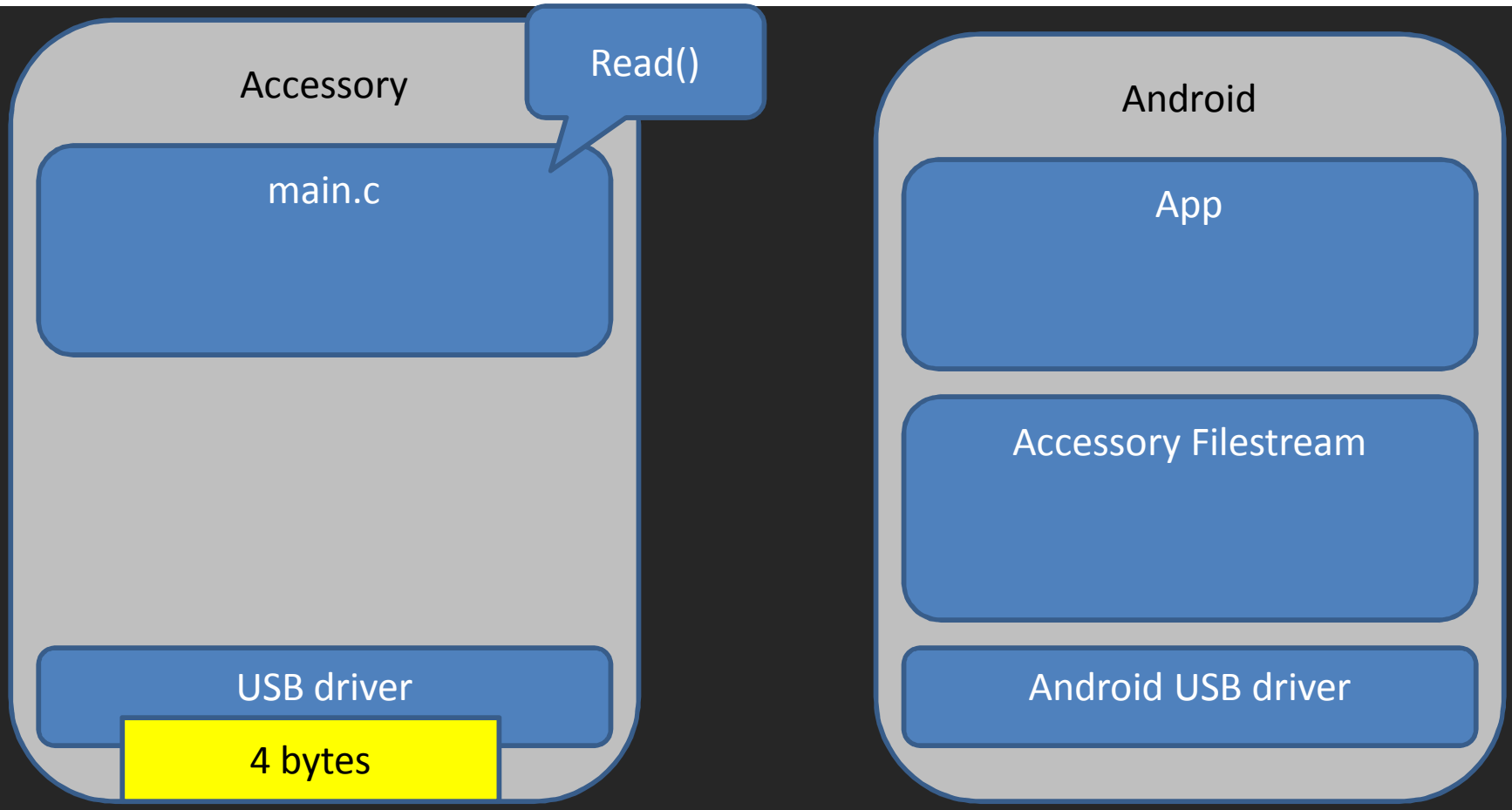
Filestreams: Packing



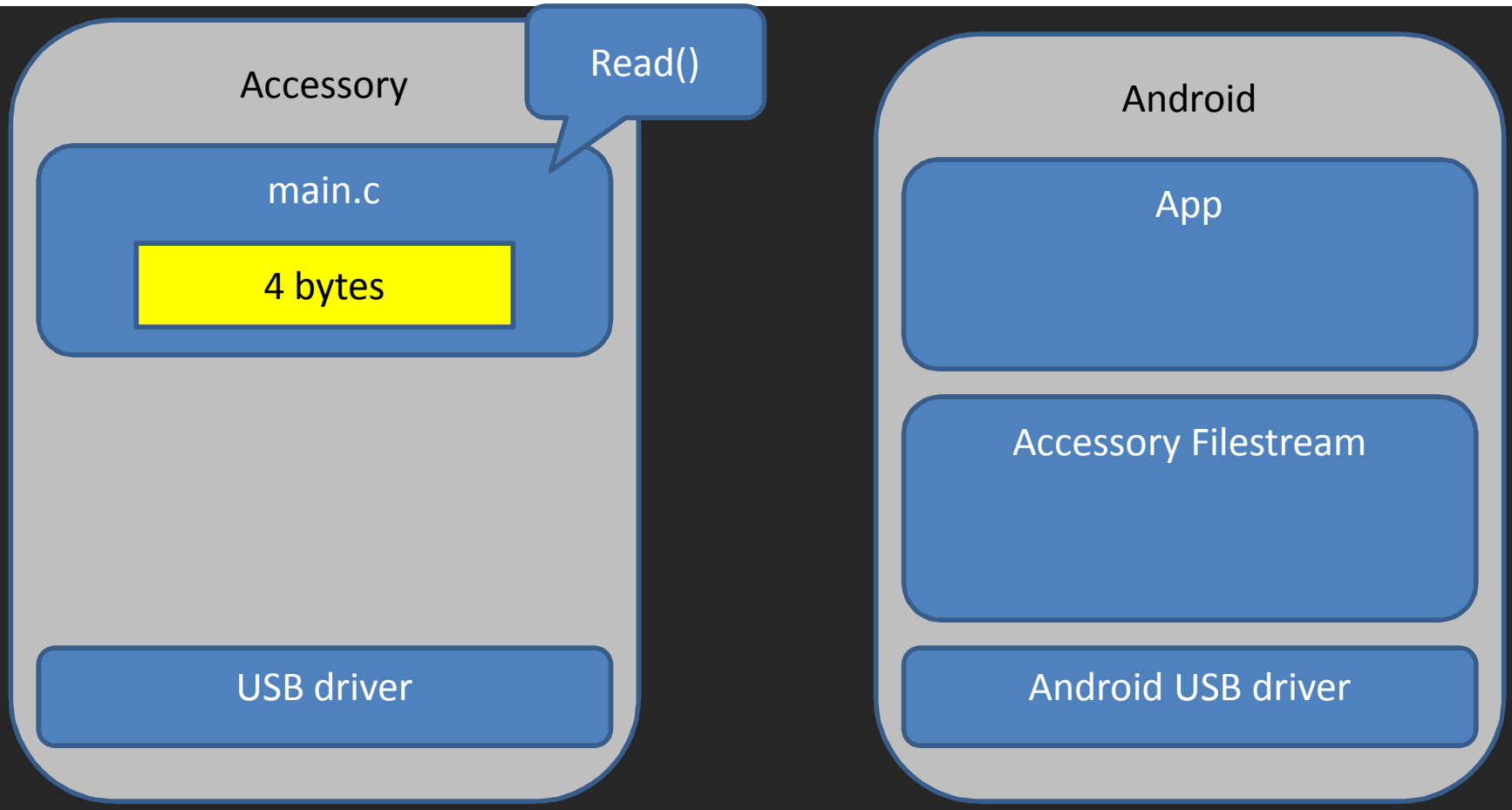
Filestreams: Packing



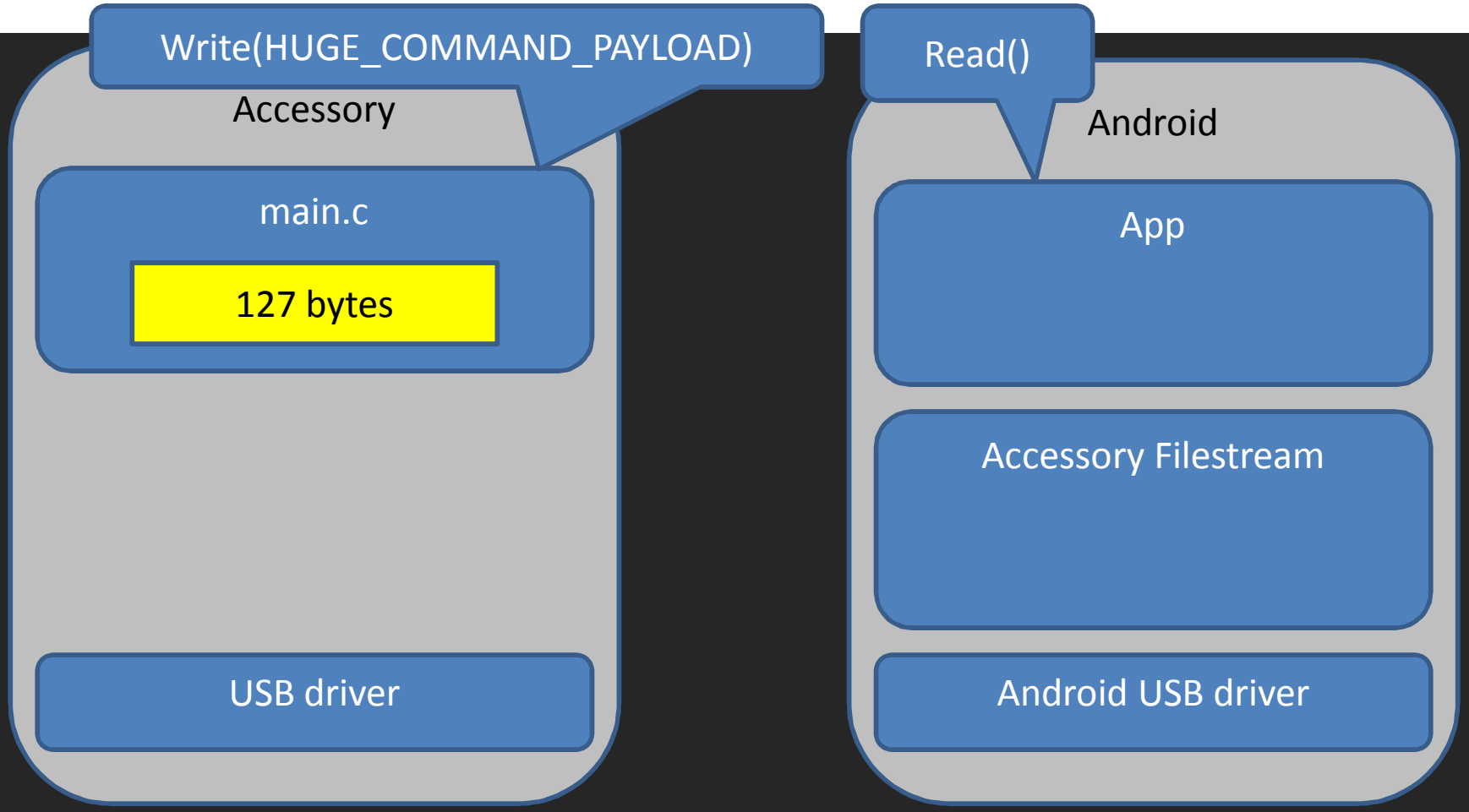
Filestreams: Packing



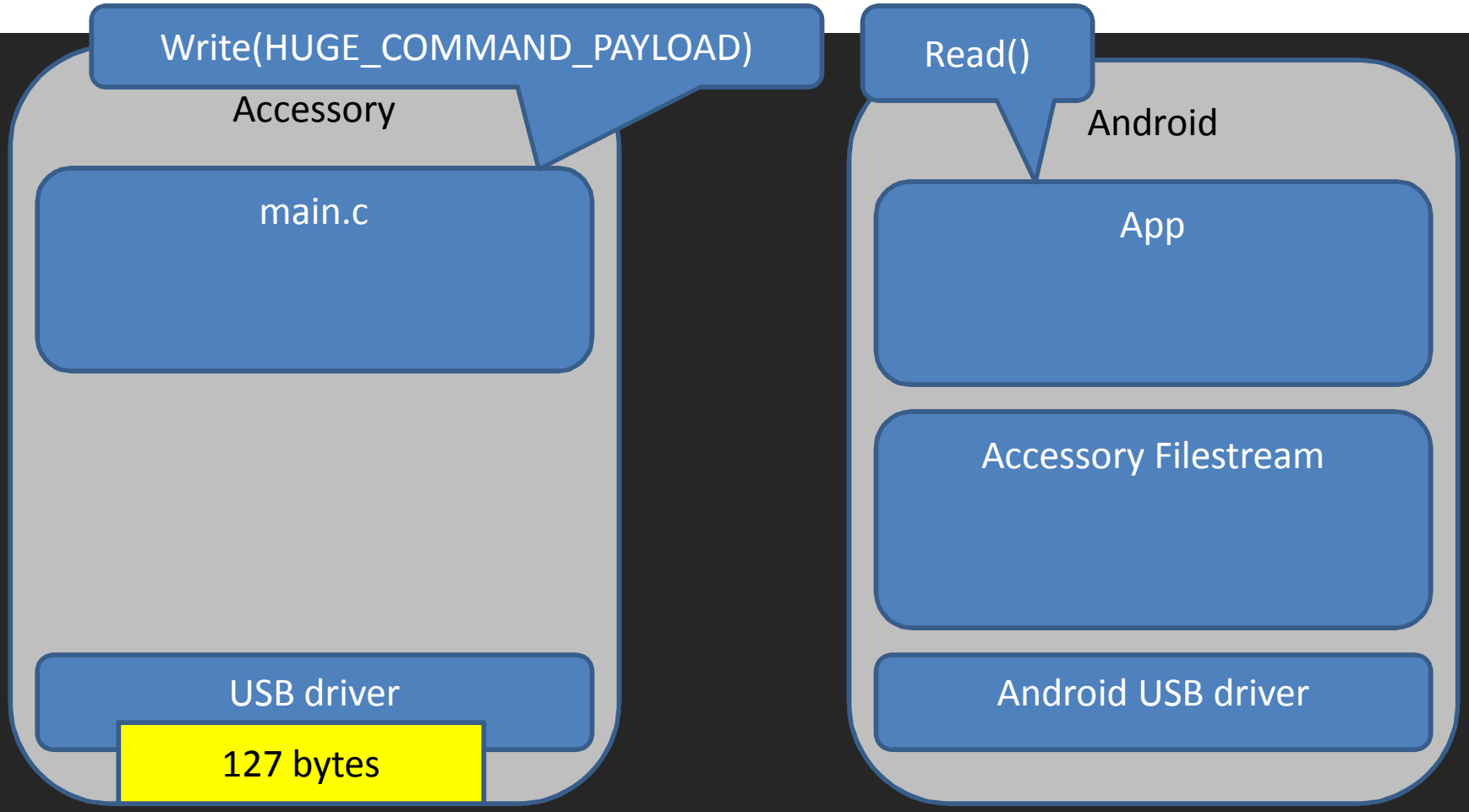
Filestreams: Packing



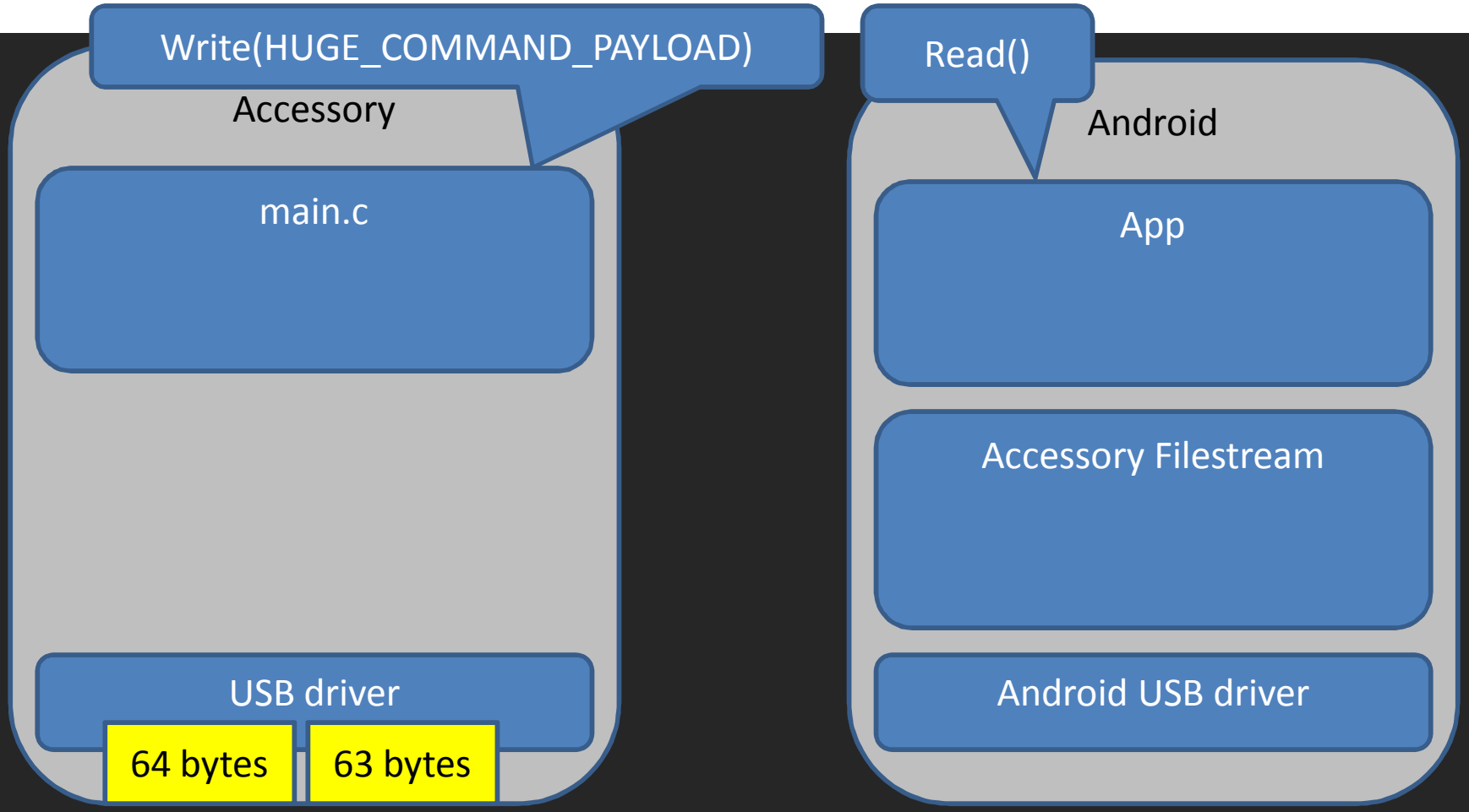
Filestreams: Fragmentation



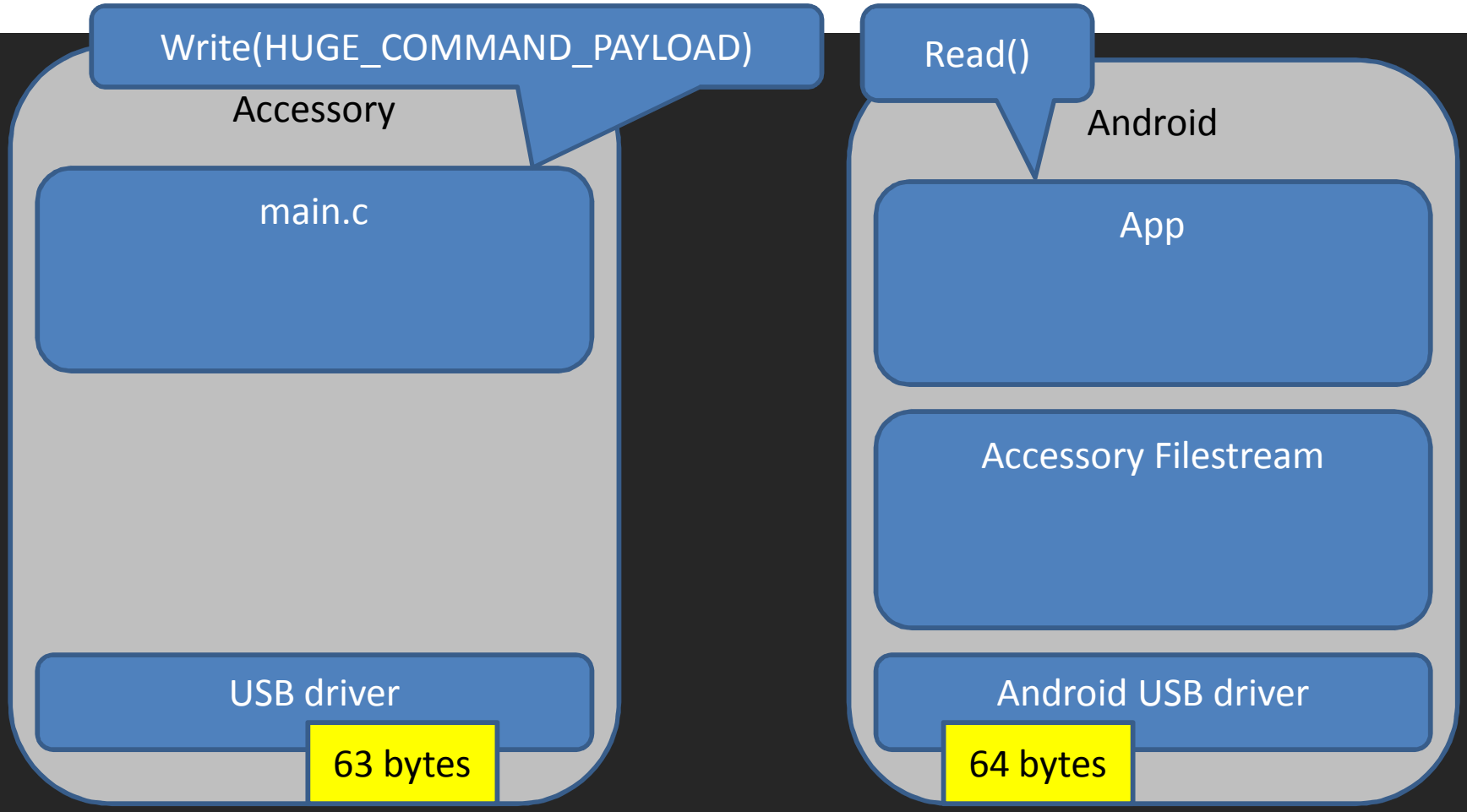
Filestreams: Fragmentation



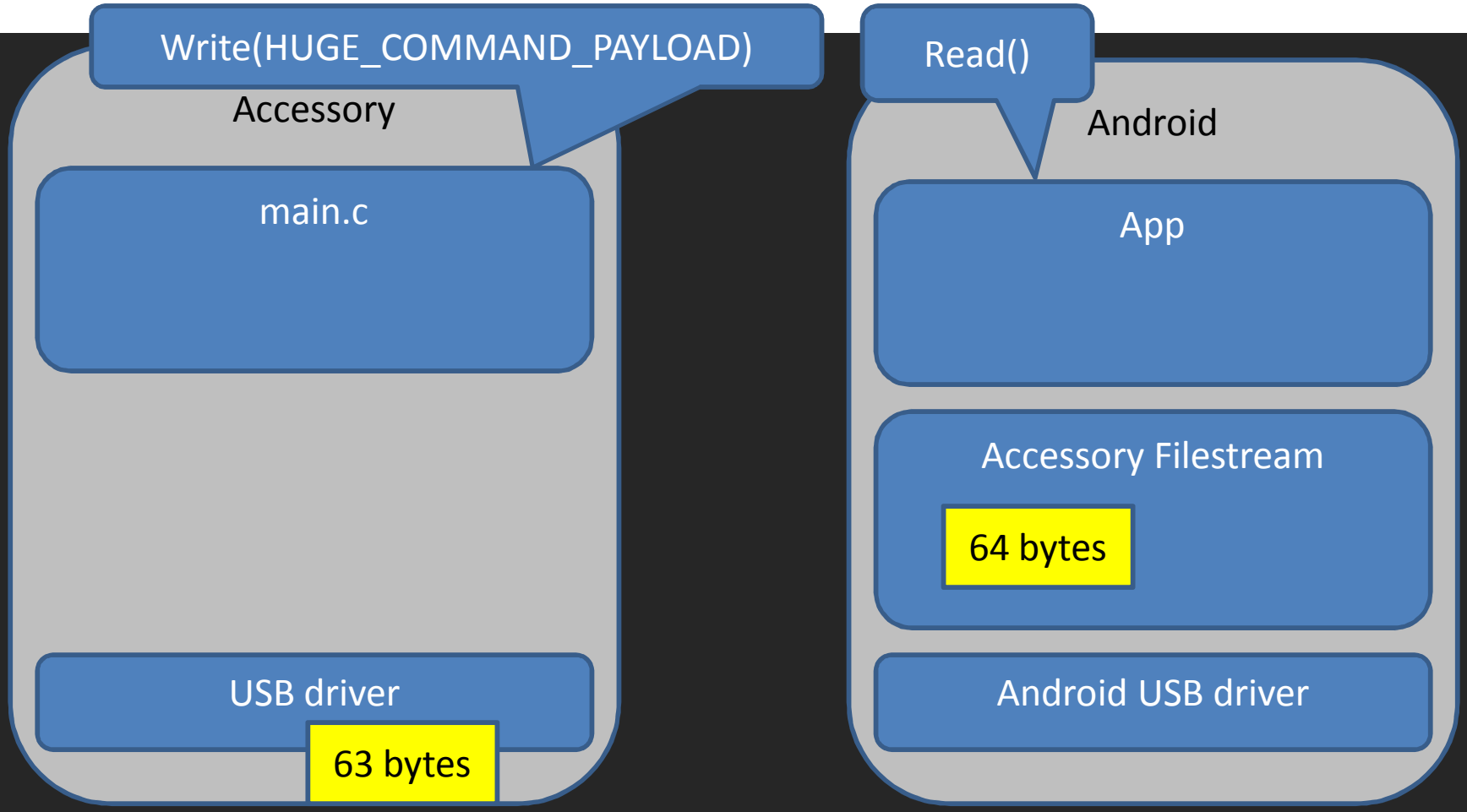
Filestreams: Fragmentation



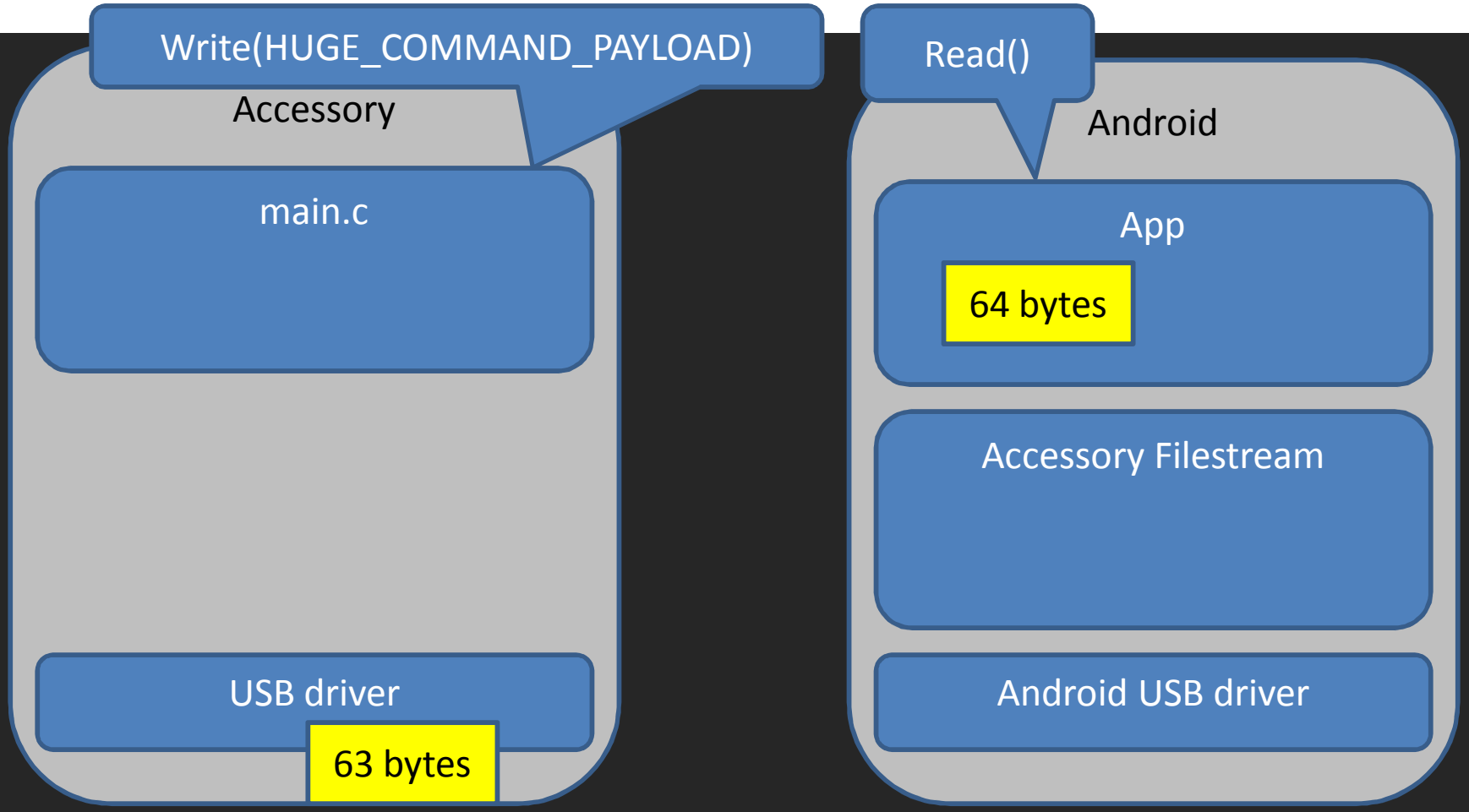
Filestreams: Fragmentation



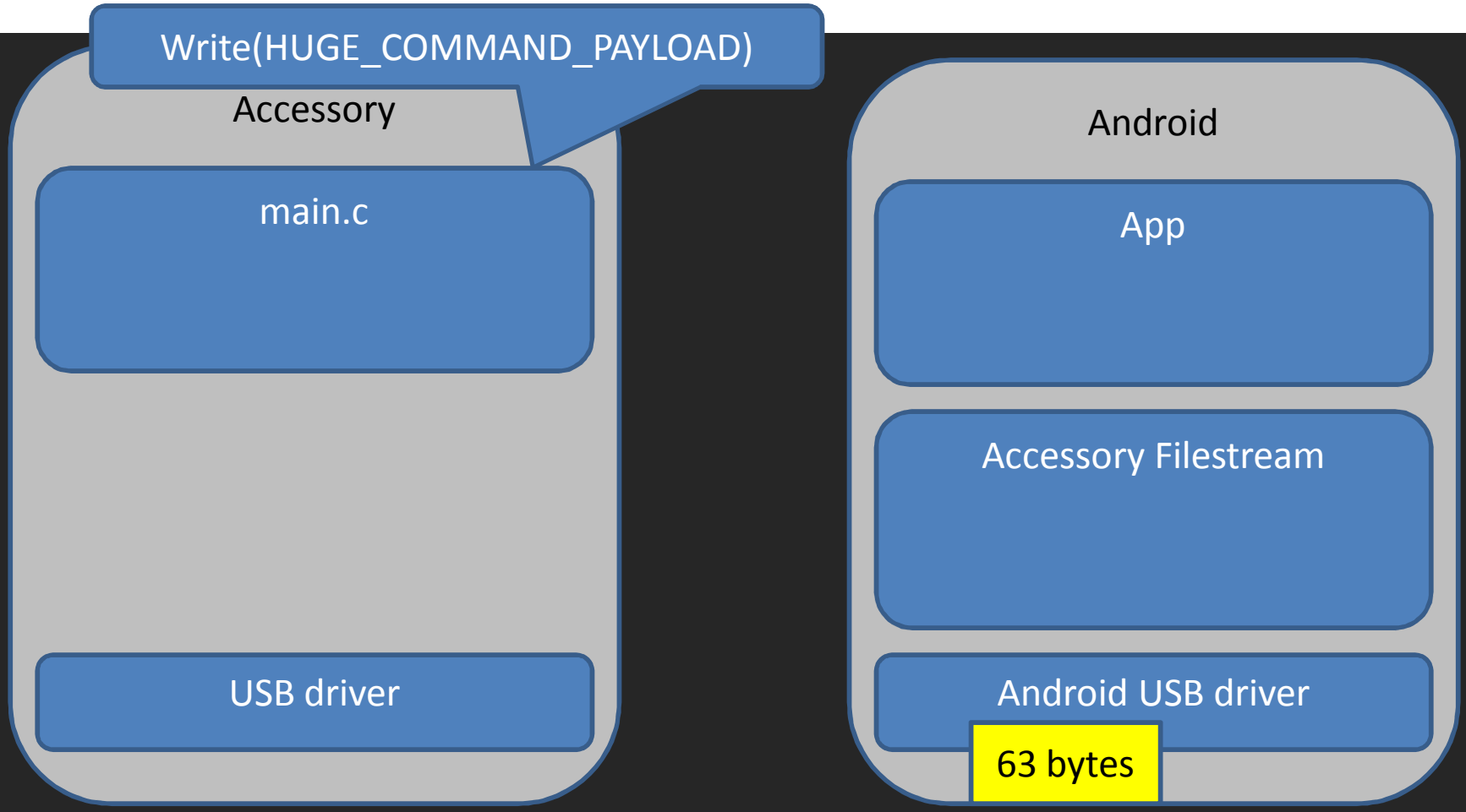
Filestreams: Fragmentation



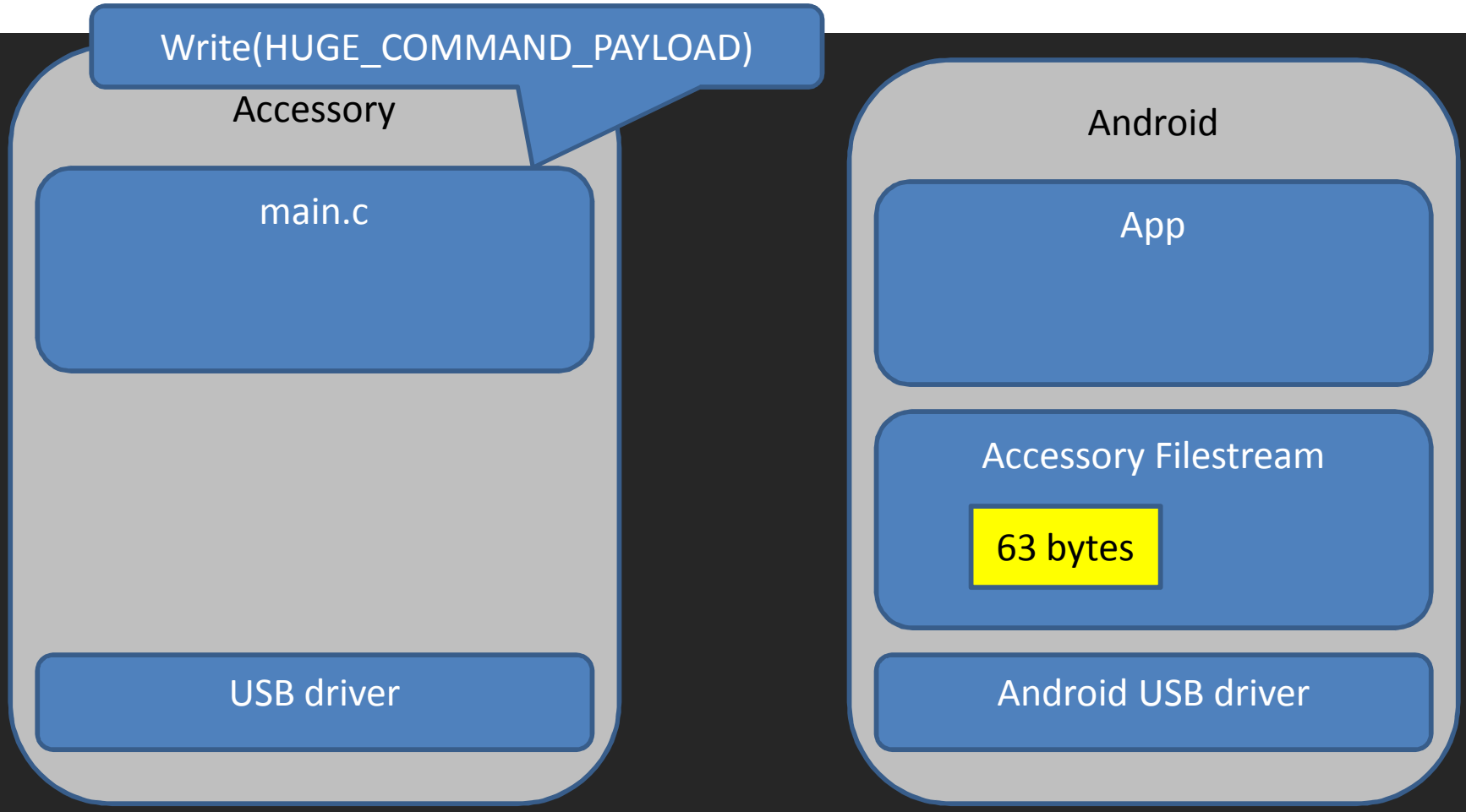
Filestreams: Fragmentation



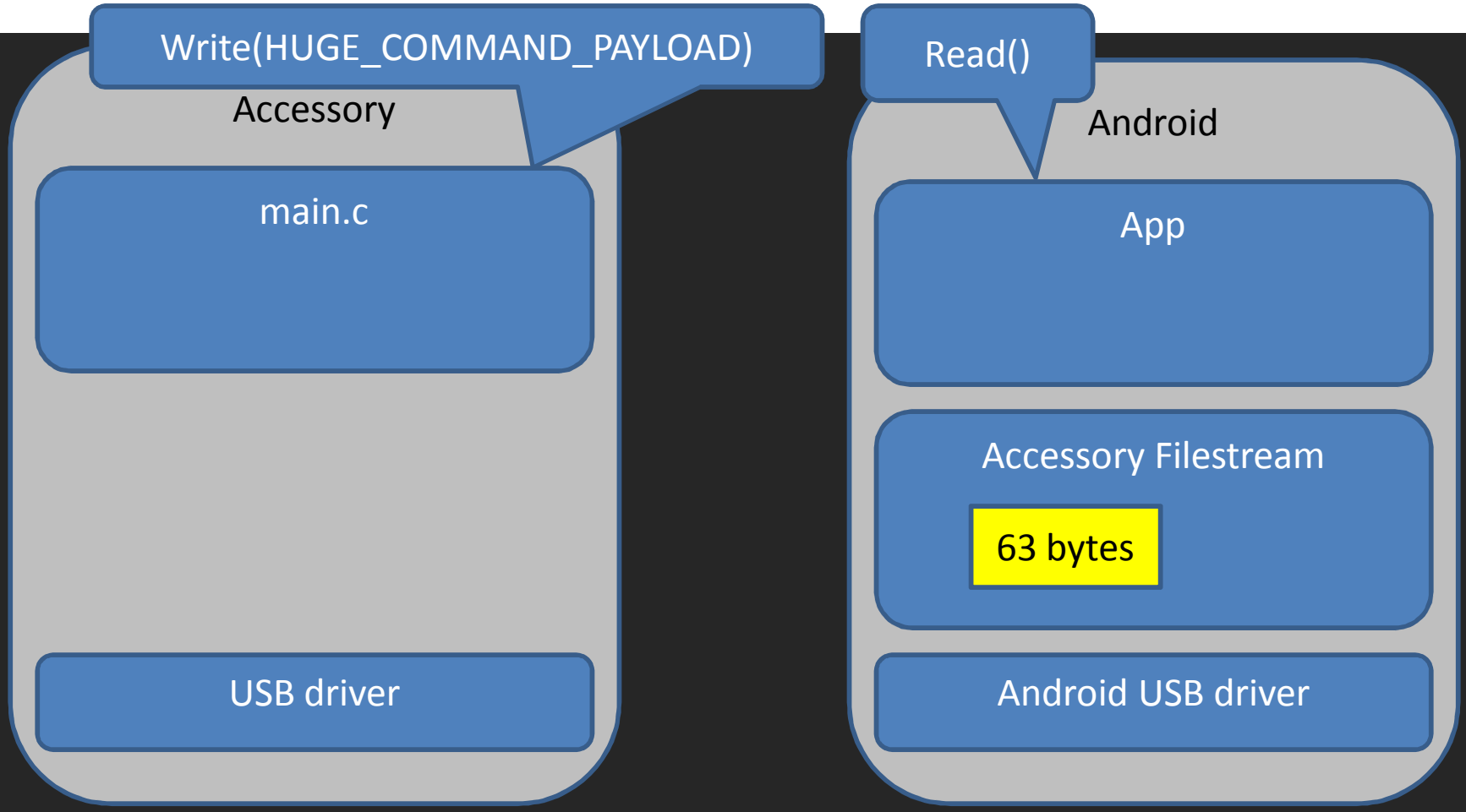
Filestreams: Fragmentation



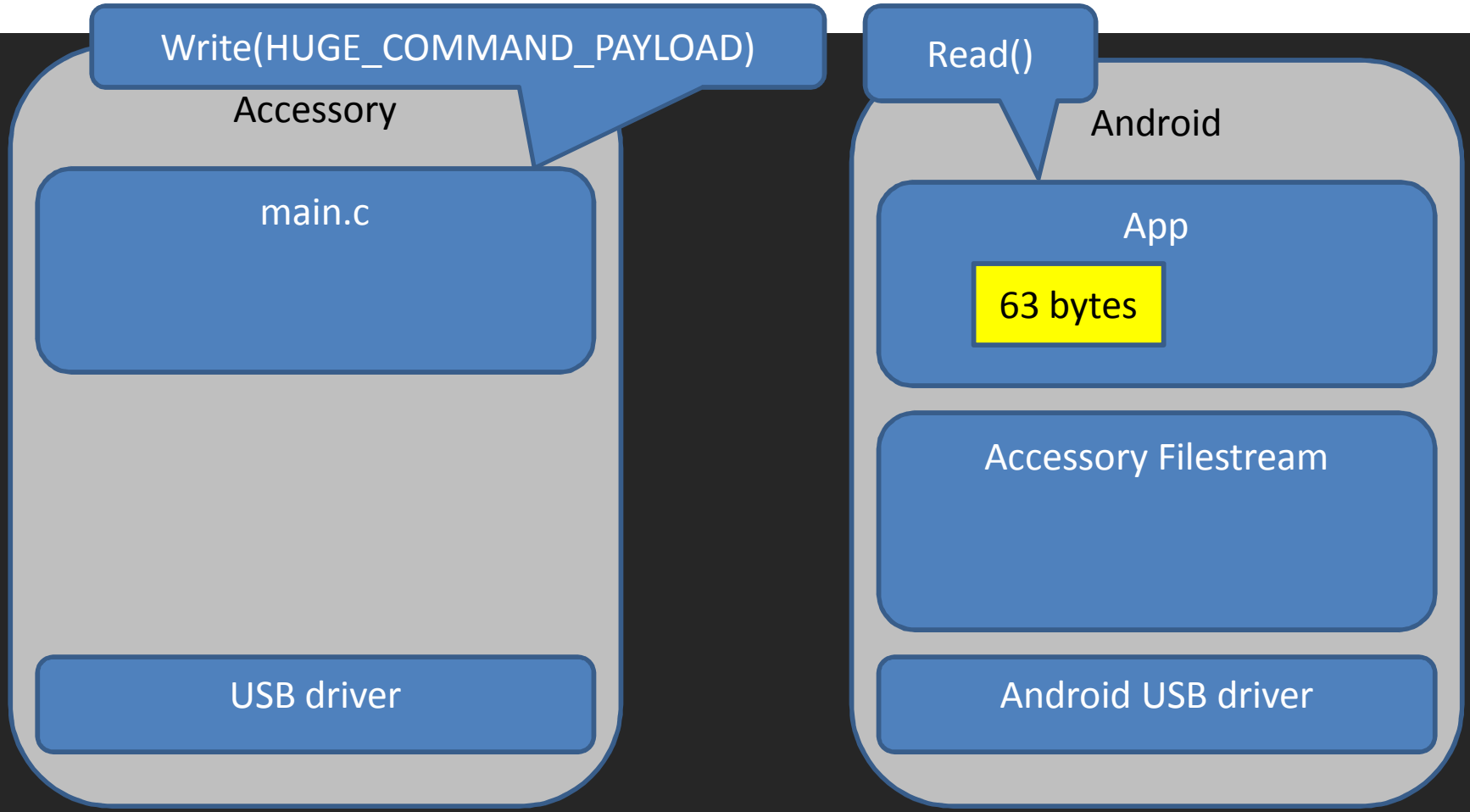
Filestreams: Fragmentation



Filestreams: Fragmentation



Filestreams: Fragmentation



Передача данных по USB

Длина данных

- OpenAccessory endpoints имеют размер 64 байта
- USB Bulk передача считается законченной, если переданы все данные и длина последнего пакета не равна размеру endpoint
- При передаче данных аксессуар должен добавлять в передачу пакет нулевой длины, если длина данных кратна размеру endpoint (64 байта)
- Несоблюдение этого правила приводит к тому, что данные не передаются из USB драйвера Android к OpenAccessory Filestream

Манифест

- Приложение Android Apps должно добавить в манифест необходимые данные

```
<uses-library android:name="com.android.usb.accessory" />
<intent-filter>
     <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
</intent-filter>
<meta-data android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
            android:resource="@xml/accessory_filter" />
```

- В файле /res/xml/accessory_filter.xml необходимо указать данные аксессуара:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<usb-accessory manufacturer="Microchip Technology Inc." model="Basic Accessory Demo"
     version="1.0" />
</resources>
```

- Данные должны совпадать с теми, которые аксессуар передает функции AndriodAppStart()

OpenAccessory Framework v2.0

- USB Accessory Protocol v2 (v4.1+)
  - Аксессуар - Host, Android - Slave
  - USB Vendor-class драйвер
    - 1 bulk endpoint in
    - 1 bulk endpoint out
  - Audio драйвер
    - 1 isochronous endpoint out
  - HID control interface
    - Выполнено с помощью передачи команд через EP0 через vendor class драйвер

OpenAccessory Framework v2.0

- Допускается комбинация интерфейсов
  - Только Audio
  - Audio + Accessory
  - Только Accessory (как в AOA v1.0)

OpenAccessory Framework v2.0

- Разрешает передачу HID команд
  - Посылается через специальные EP0 команды
  - Позволяет подключить мышь/клавиатуру/ джойстик к устройствам Android, которые не имеют возможности USB host подключения
  - Удобны для управления аудио (вперед, назад, воспроизведение, …)

Android App Development for Accessories: USB Host

android

Подключение к устройству

- Выполняется на низком уровне через интерфейсы и конечные точки

```
UsbManager = (UsbManager)getSystemService(Context.USB_SERVICE);
UsbInterface intf = device.getInterface(0);
UsbDeviceConnection connection = manager.openDevice(device);
connection.claimInterface(intf, true);
UsbEndpoint endpointOUT = intf.getEndpoint(0);
connection.bulkTransfer(  endpointOUT,
                          buffer,
                          buffer.length,
                          timeout_ms);
```

Манифест

- Приложение Android Apps должно добавить в манифест необходимые данные

```
<uses-library android:name="com.android.usb.host" />
<intent-filter>
     <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
</intent-filter>
<meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
           android:resource="@xml/device_filter" />
```

- В файле /res/xml/device_filter.xml необходимо указать данные устройства:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
     <usb-device vendor-id="1240" product-id="516"/>
     <usb-device class="256" subclass="256" protocol="5"/>
</resources>
```

Подключение аксессуаров: По Wi-Fi®

Что доступно?

- Большинство устройств Android не поддерживают ad-hoc сети
- Чтобы работать через Wi-Fi аксессуары должны подключиться к имеющейся сети
- С версии v4.1 поддерживается Wi-Fi Direct для связи устройств точка-точка

Создавать или не создавать приложения?

- Можно использовать существующие программы
  - Web браузеры
  - telnet/ftp клиенты
- При создании собственных приложений используется стандартный интерфейс Java для работы с сетями
  - http://developer.android.com/reference/java/net/Socket.html
  - Масса литературы доступна

Манифест

- Для работы в манифест надо добавить:
  - <uses-permission android:name="android.permission.INTERNET" />

Подключение аксессуаров:
По : Bluetooth®

Что доступно?

- Android v2.x and раньше
  - Использует RFCOMM (SPP Profile)
    - Эмуляция RS232 через Bluetooth
    - Только связь точка-точка
    - Android - клиент, аксессуар -сервер
- Android v3.x
  - Добавлена поддержка для Headset and Advanced Audio Distribution Profile (A2DP)
- Android v4.x
  - Добавлена поддержка для Health Device Profile (HDP).

Решения от Microchip

- Примеры ПО для мкроконтроллера, приложений для Android, документация
- OpenAccessory: www.microchip.com/android
- Android is USB Host: www.microchip.com/usb
- Wi-Fi: www.microchip.com/wifi
- Bluetooth: www.microchip.com/bluetooth