How to fix Usually Slightly Broken devices and drivers?

Krzysztof Opasiak

SAMSUNG

Samsung R&D Institute Poland

FOSDEM '17

Brussels 4 & 5 February 2017

Agenda

USB basics

Plug & Play

Plug & do what I want

Plug & tell me more

Summary

Q & A

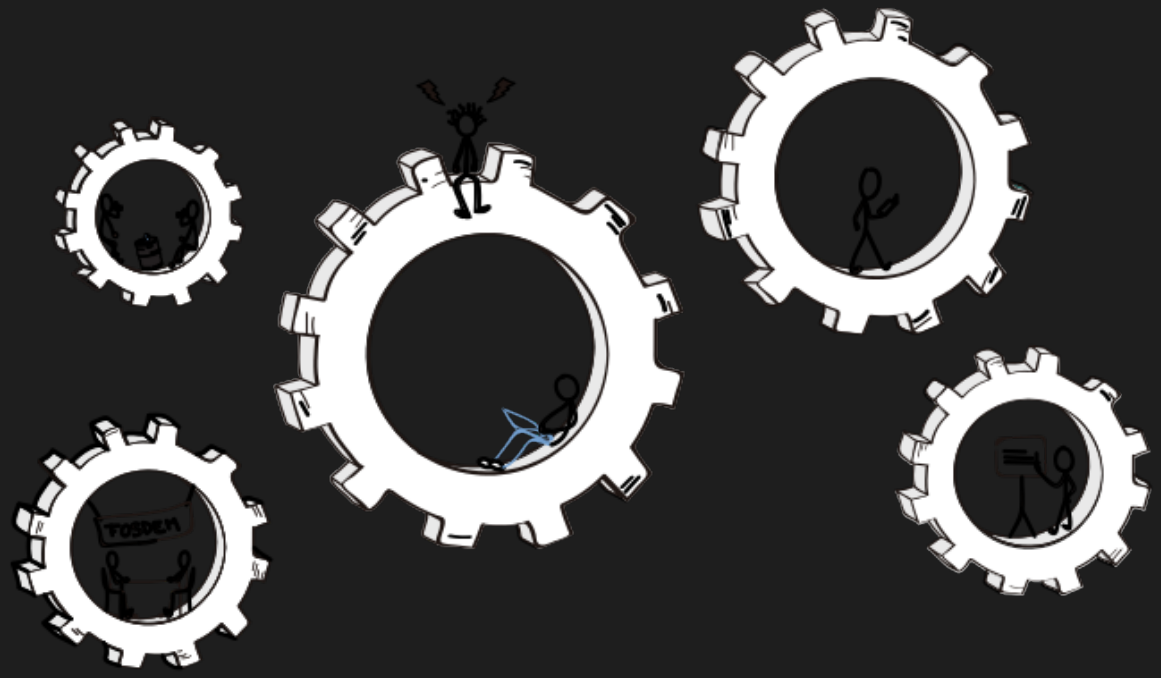
This presentation…

is about:

- **USB**
- **USB devices management**
- **USB drivers policy modification**
- **USB traffic sniffing**

is **NOT** about:

- **Kernel code debugging**
- **Using kgdb**
- **Using tracepoints**
- **Using JTAG**



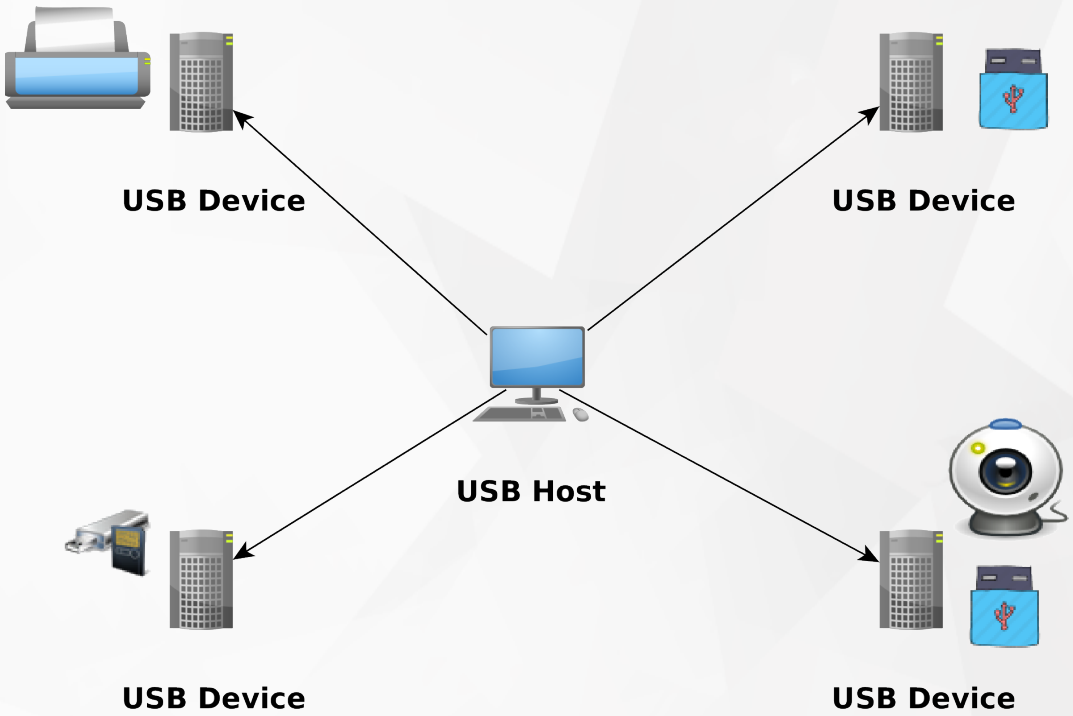
USB basics

FOSDEM '17
Brussels 4 & 5 February 2017

What USB is about?

It's about providing services!

- **Storage**
- **Printing**
- **Ethernet**
- **Camera**
- **Any other**



Endpoints…

- **Device may have up to 31 endpoints (including ep0)**
- **Each of them gets a unique endpoint address**
- **Endpoint 0 may transfer data in both directions**
- **All other endpoints may transfer data in one direction:**

IN	**Transfer data from device to host**
OUT	**Transfer data from host to device**

Endpoint types

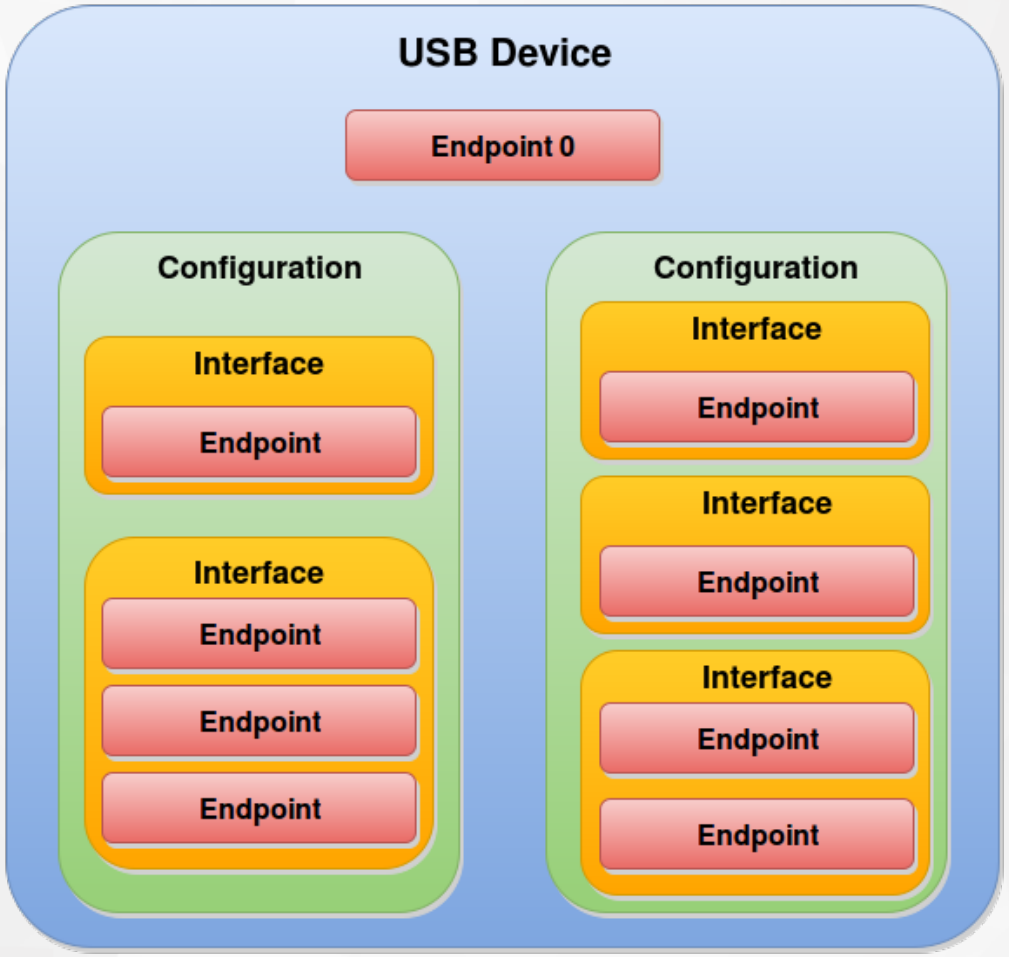
- **Control**
  - Bi-directional endpoint
  - Used for enumeration
  - Can be used for application
- **Interrupt**
  - Transfers a small amount of low-latency data
  - Reserves bandwidth on the bus
  - Used for time-sensitive data (HID)

Endpoint types

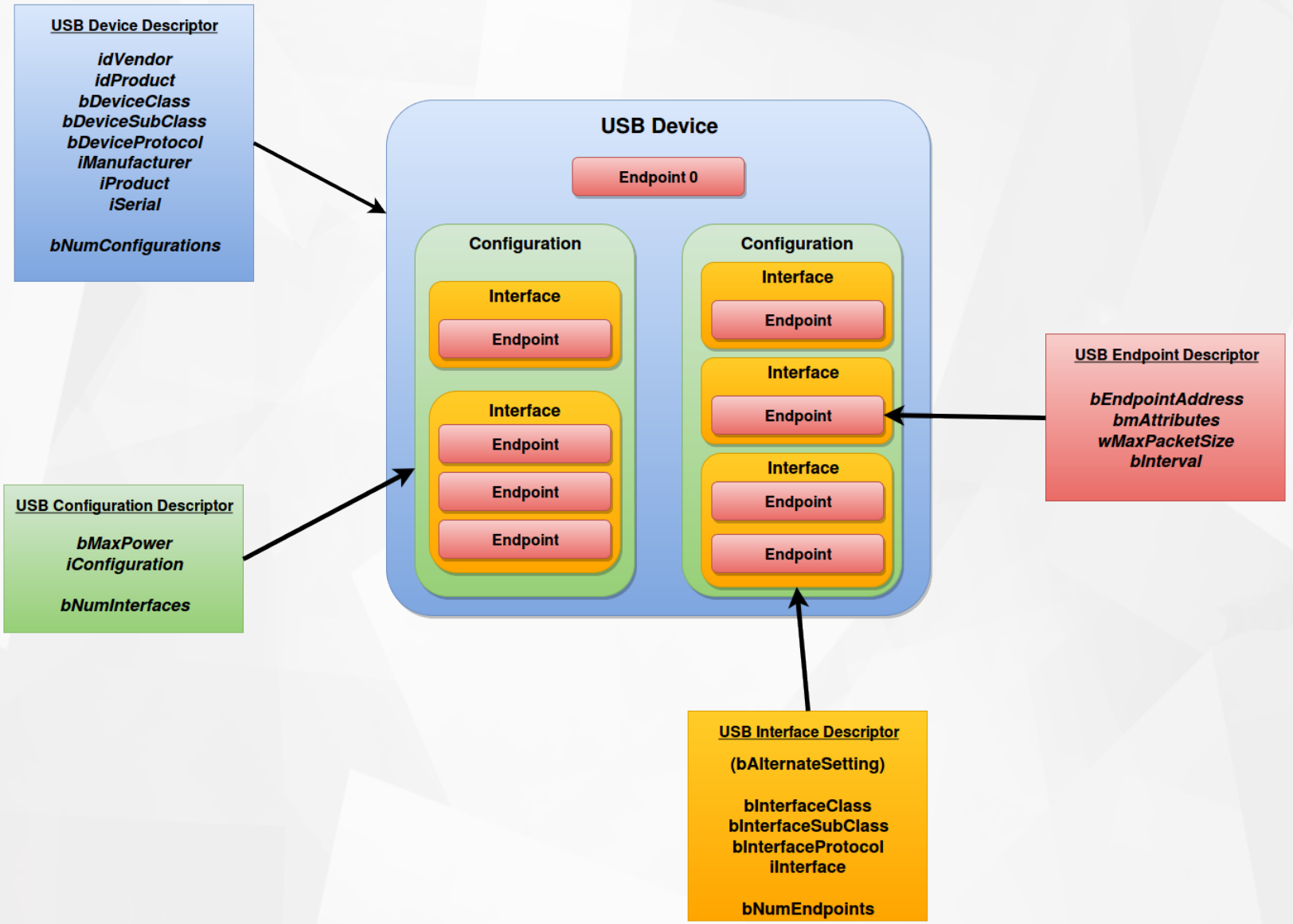
- **Bulk**
  - Used for large data transfers
  - Used for large, time-insensitive data (Network packets, Mass Storage, etc).
  - Does not reserve bandwidth on bus, uses whatever time is left over

- **Isochronous**
  - Transfers a large amount of time-sensitive data
  - Delivery is not guaranteed (no ACKs are sent)
  - Used for Audio and Video streams
  - Late data is as good as no data
  - Better to drop a frame than to delay and force a re-transmission

USB device


USB Device
Endpoint 0
Configuration
Interface
Endpoint
Interface
Endpoint
Endpoint
Endpoint
Configuration
Interface
Endpoint
Interface
Endpoint
Interface
Endpoint
Endpoint

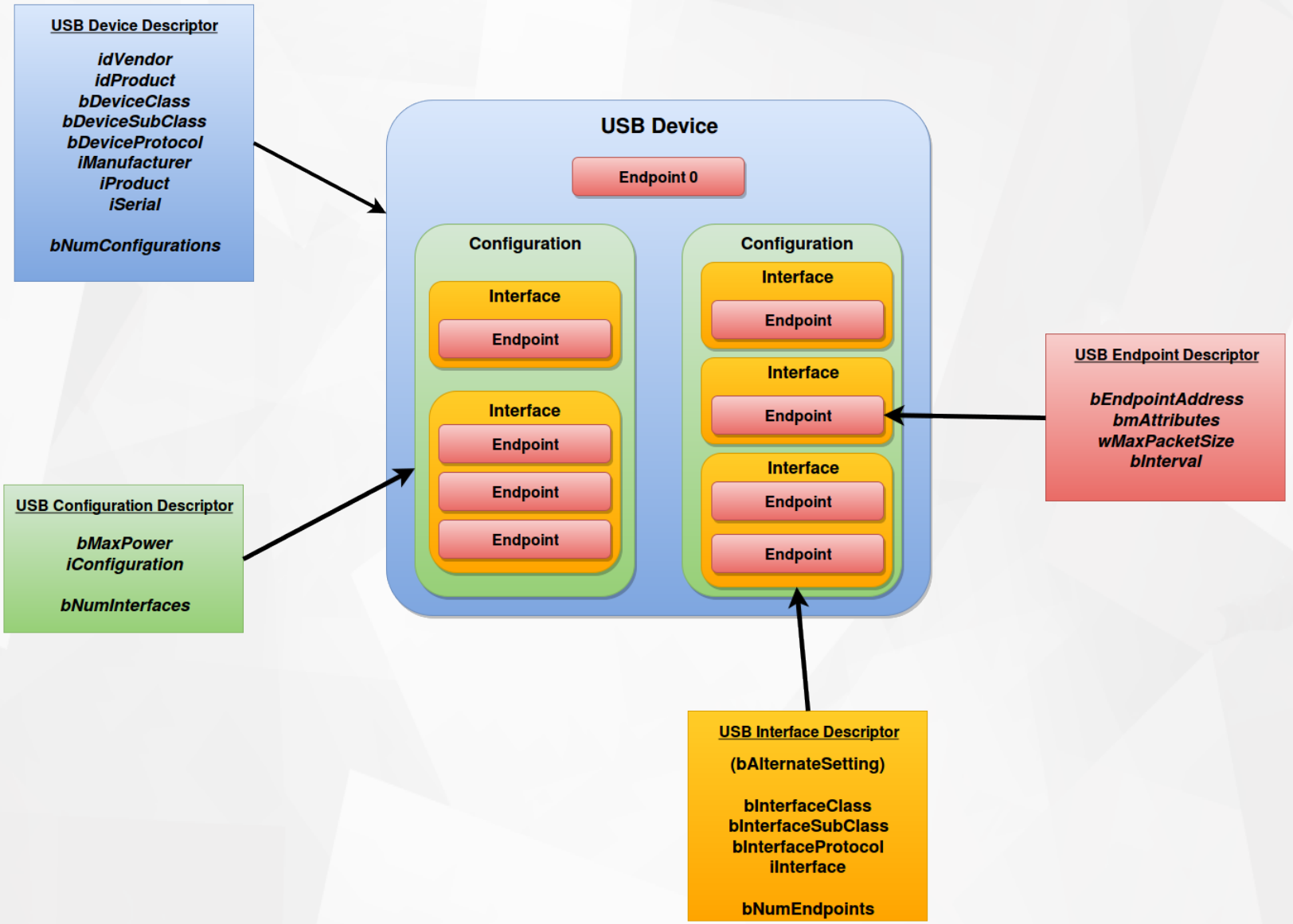
USB descriptors



USB classes

00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video
0Fh	Interface	Personal Healthcare
10h	Interface	Audio/Video Devices
11h	Device	Billboard Device Class
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

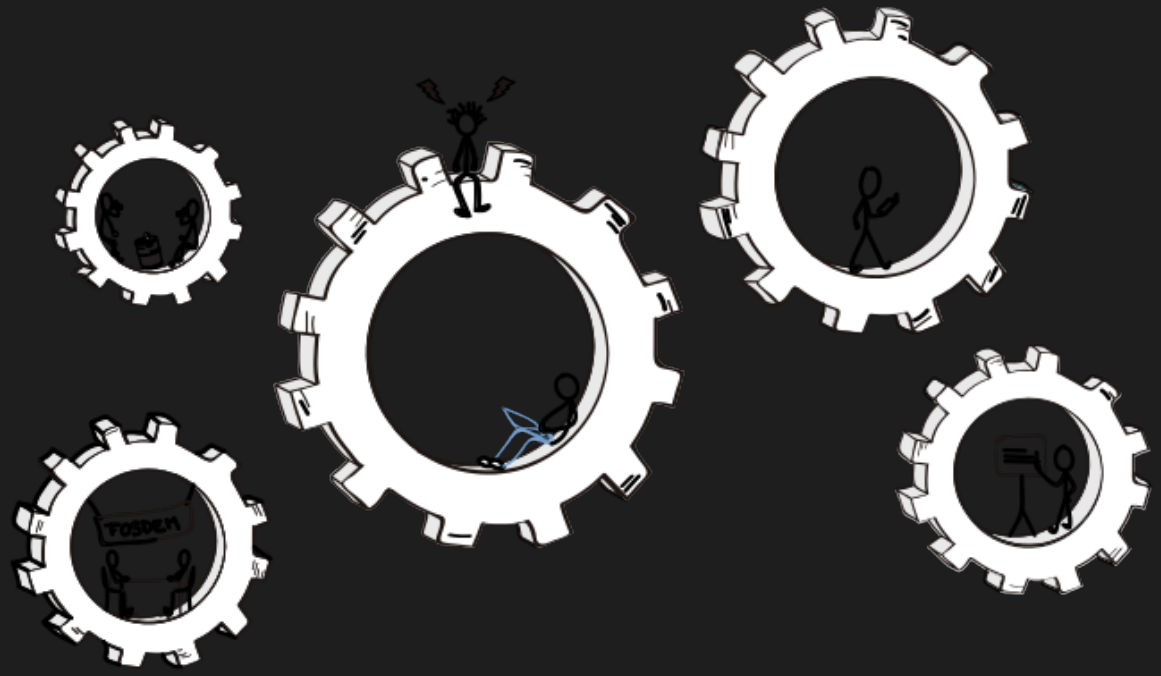
USB descriptors



USB device example

dmesg & lsusb

DEMO



Plug & Play

FOSDEM '17

Brussels 4 & 5 February 2017

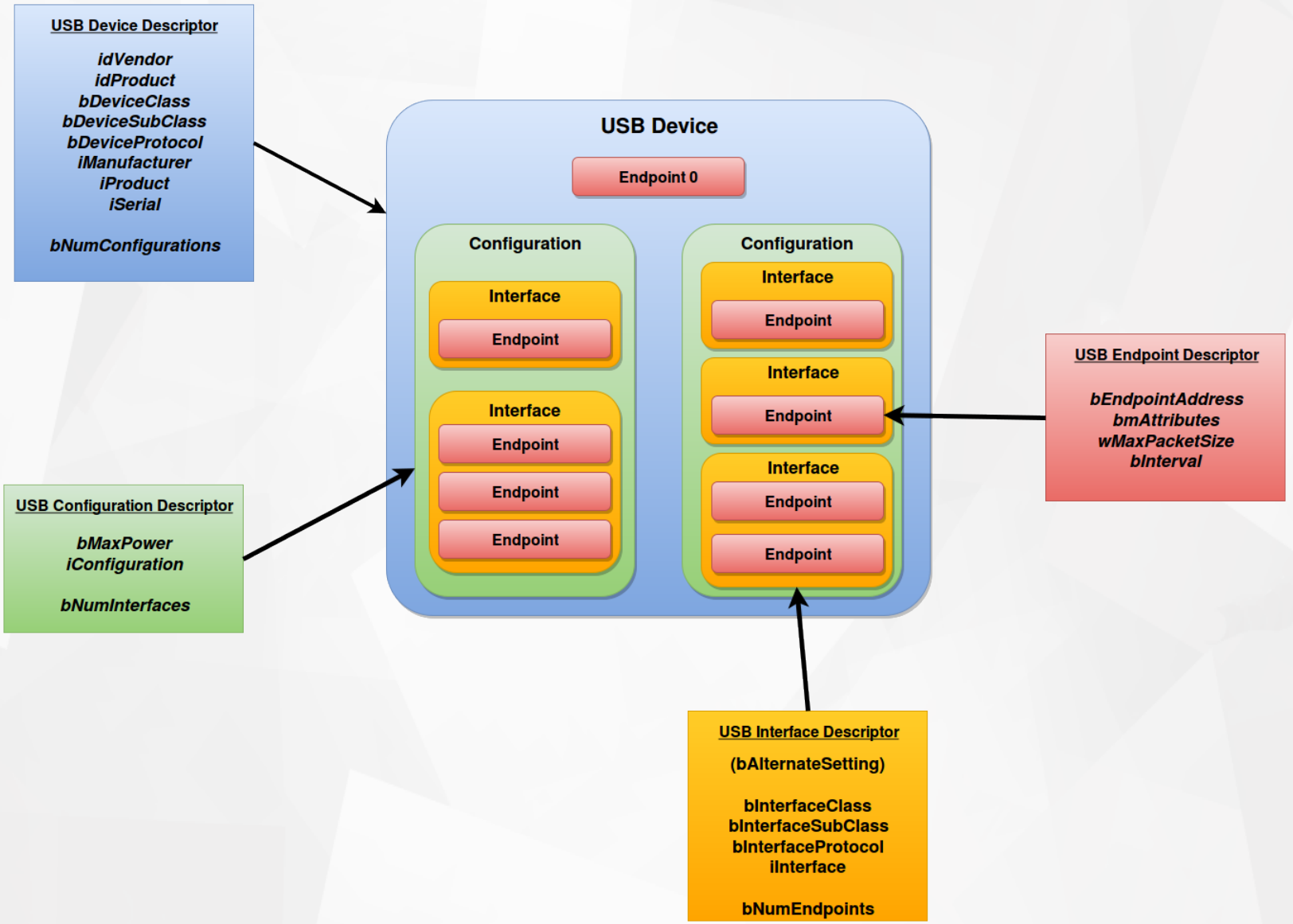
Step by step

- Plug in device
- Detect Connection
- Set address
- Get device info
- Choose configuration
- Choose drivers for interfaces
- Use it ;)

Set address

- On plug-in device uses default address 0x00
- Only one device is being enumerated at once
- Hosts assigns unique address for new device
- Usually it's just the next one (dev.addr = addr++)

USB Device Details



Which configuration is the most suitable?

- **Do we have enough power for it (bMaxPower)?**
- **Does it have at least one interface?**
- **If the device has only one config**
  - The first one!
- **If the device has multiple configs**
  - The first one which first interface class is **different** than **Vendor Specific**
- **All interfaces of chosen configuration become available so let's use them**

What USB driver really is?

- **Piece of kernel code (often a module)**
- *struct usb_driver*
- **Usually it provides something to userspace (network interface, block device, tty, etc.)**
- **Implementation of some communication protocol**
- **…so it's a little bit equivalent of web browser, ssh client etc.**

How driver is chosen?

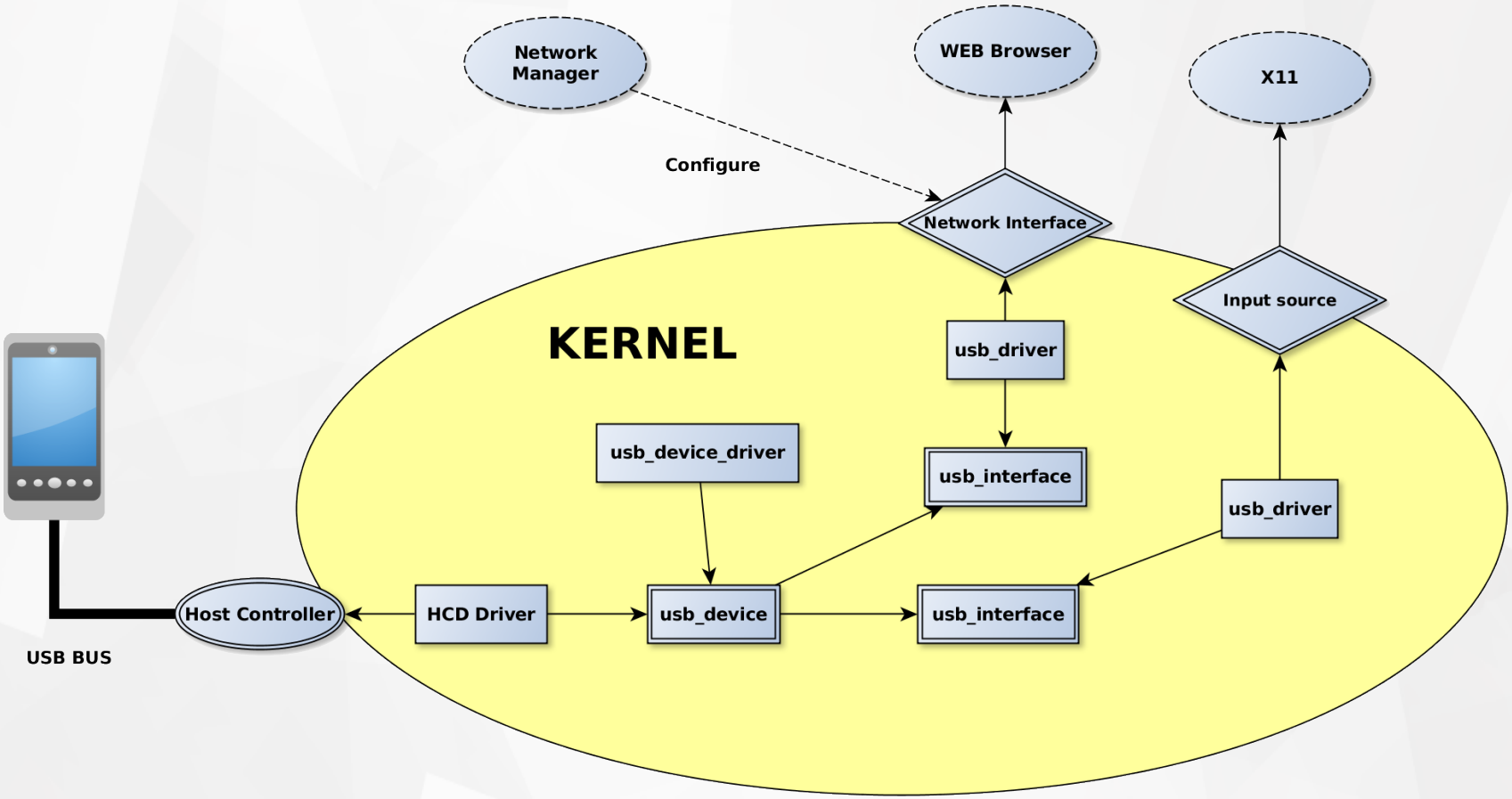
- **Kernel has a list of registered drivers**
- **Each driver has an array of acceptable device IDs**
- **Kernel goes through the list and if some id matches calls driver's** *probe()*
- **If driver is not ther udev may load it's module based on alias**
- **Module aliases are generated based on acceptable device IDs**

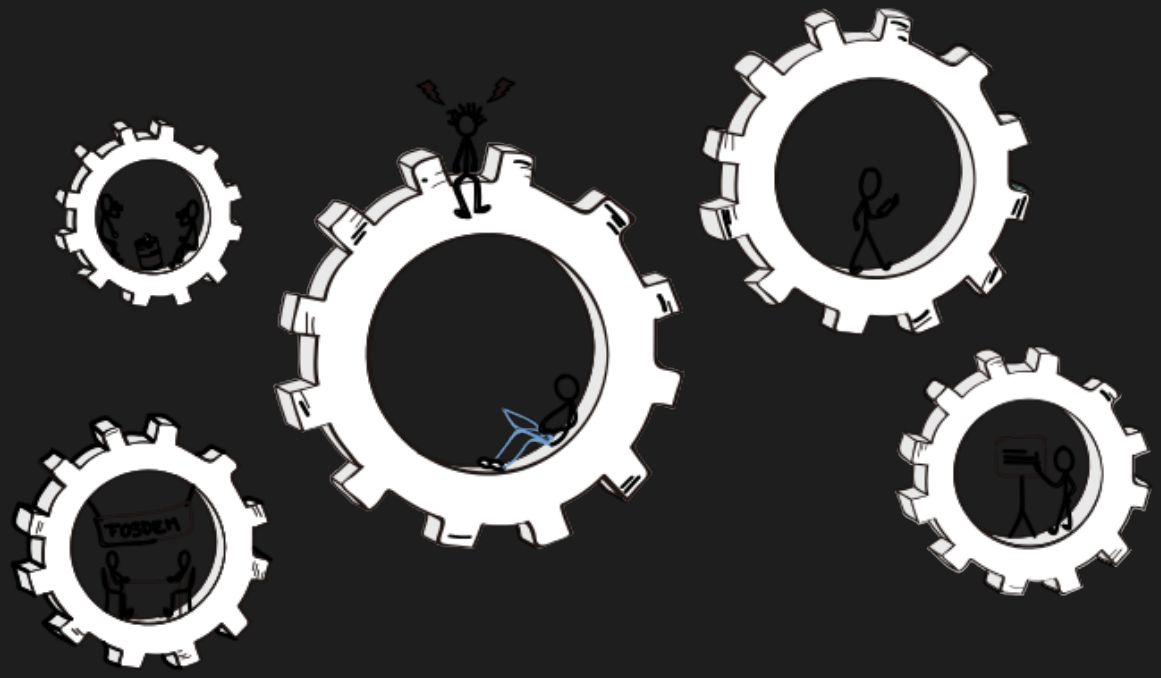
USB device identity

```
struct usb_device_id {
    /* which fields to match against? */
    __u16 match_flags;

    /* Used for product specific matches */
    __u16 idVendor;
    __u16 idProduct;
    __u16 bcdDevice_lo;
    __u16 bcdDevice_hi;

    /* Used for device class matches */
    __u8 bDeviceClass;
    __u8 bDeviceSubClass;
    __u8 bDeviceProtocol;

    /* Used for interface class matches */
    __u8 bInterfaceClass;
    __u8 bInterfaceSubClass;
    __u8 bInterfaceProtocol;

    /*
     * Used for vendor-specific
     * interface matches
     */
    __u8 bInterfaceNumber;

    /* not matched against */
    kernel_ulong_t driver_info;
};
```

```
#define USB_DEVICE_ID_MATCH_VENDOR 0x0001
#define USB_DEVICE_ID_MATCH_PRODUCT 0x0002
#define USB_DEVICE_ID_MATCH_DEV_LO 0x0004
#define USB_DEVICE_ID_MATCH_DEV_HI 0x0008
#define USB_DEVICE_ID_MATCH_DEV_CLASS 0x0010
#define USB_DEVICE_ID_MATCH_DEV_SUBCLASS 0x0020
#define USB_DEVICE_ID_MATCH_DEV_PROTOCOL 0x0040
#define USB_DEVICE_ID_MATCH_INT_CLASS 0x0080
#define USB_DEVICE_ID_MATCH_INT_SUBCLASS 0x0100
#define USB_DEVICE_ID_MATCH_INT_PROTOCOL 0x0200
#define USB_DEVICE_ID_MATCH_INT_NUMBER 0x0400
```

USB Host Big Picture





Plug & do what I want

FOSDEM '17
Brussels 4 & 5 February 2017

Automation is good…

…but not always:

- **Too many devices allowed**
- **Only part of device functionality is needed**
- **Wrong config chosen**
- **No matching driver found**
- **Wrong driver bound**

What kernel gives us?

- **SysFS infrastructure**
  - Device Information
  - Device Management
  - Drivers Information
- **Device node**
  - Device Information
  - Device Communication
  - Used by *libusb*

/sys/bus/usb/devices/ demystified

- `usbX`

  X **ID of host controller on your machine**

- `X-A.B.C`

  X **HCD ID (as above)**

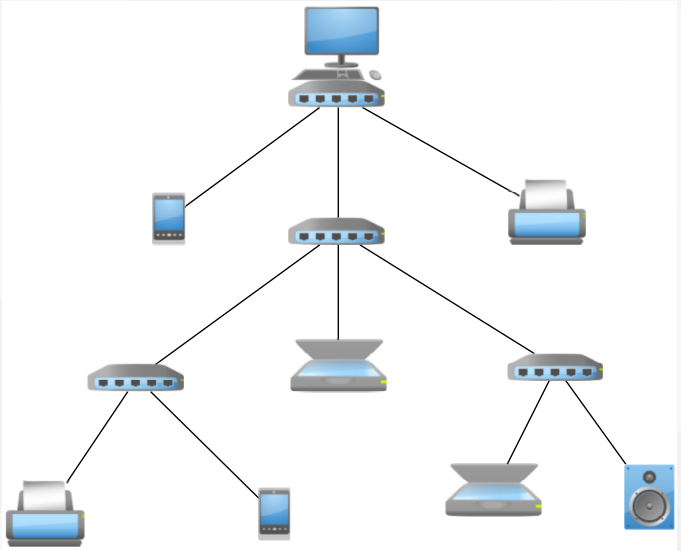
  A.B.C **Physical path to port where your USB device is connected**

- `X-A.B.C:Y.Z`

  X-A.B.C **Device path (as above)**

  Y **Active configuration**

  Z **bInterfaceNumber**



Limit number of allowed devices

- **Let's use USB Device Authorization!**
- **Each USB device has** *authorized* **attribute in sysfs directory**
- **Each HCD (`usbX`) has** *authorized_default* **attribute**
- **If** *authorized* == *0* **then device is left in unconfigured state**
- **When authorized, drivers probed automatically**
- **Can be automated using** *usbguard* **project**

Device Authorization HOWTO

```
# Choose USB bus
$ cd /sys/bus/usb/devices/usb$X

# Stop authorizing devices by default
$ echo 0 > authorized_default

# Connect new device, do other stuff

# Authorize device of your choice
$ cd /sys/bus/usb/devices/$DEV_DIR
$ echo 1 > authorized
```

Use only subset of functionality

- **Let's use USB Interface Authorization! (v4.4+)**
- **Each USB interface has *authorized* attribute in sysfs directory**
- **Each HCD (`usbX`) has *authorized_default* attribute**
- **If *authorized == 0* then drivers are not allow to bind**
- **Driver probing has to be triggered manually after authorization**

Interface Authorization HOWTO

```
# Choose USB bus
$ cd /sys/bus/usb/devices/usb$X

# Stop authorizing devices by default
$ echo 0 > interface_authorized_default

# Authorize interface of your choice
$ cd /sys/bus/usb/devices/$INTERFACE_DIR
$ echo 1 > authorized

# Trigger driver search
$ echo -n $INTERFACE_DIR \
                    > /sys/bus/usb/drivers_probe
```

Change configuration

- **Configuration is chosen by kernel**
- **Choice is based on hardcoded heuristic**
- **But we may change it:**

```
$ cd $DEV_DIR

# Check current config
$ cat bConfigurationValue
1

# Set new one
$ echo $NEW_CONFIG > bConfigurationValue
```

Add device ID to driver

- **Sometimes you get a device which is compatible with another one…**
- **But has a little bit different VID:PID info**
- **This new VID:PID is not listed in driver's id table**
- **This means that your driver is not going to bind to it:(**

Dynamic IDs - formats

- **VID+PID:**

```
echo $VID $PID
```

- **VID+PID+Intf Class:**

```
echo $VID $PID $IntfClass
```

- **VID+PID+Intf Class+dev_info:**

```
echo $VID $PID $IntfClass $RefVID $RefPID
```

Dynamic IDs - formats

- **VID+PID:**

```
echo $VID $PID
```

- **VID+PID+Intf Class:**

```
echo $VID $PID $IntfClass
```

- **VID+PID+Intf Class+dev_info:**

```
echo $VID $PID $IntfClass $RefVID $RefPID
```

- **All umbers interpreted as HEX!**

Dynamic IDs - handling

- **Add new device ID**

```
$ echo $VID $PID > \
        /sys/bus/usb/drivers/$DRV_NAME/new_id
```

- **Show the list of dynamic IDs**

```
$ cat /sys/bus/usb/drivers/$DRV_NAME/new_id
```

- **Remove previously added device ID**

```
$ echo $VID $PID > \
        /sys/bus/usb/drivers/$DRV_NAME/remove_id
```

Bind/Unbind particular interface

- **Check which driver is bound**

```
$ readlink \
    /sys/bus/usb/devices/$INTERFACE_DIR/driver
```

- **Unbind driver**

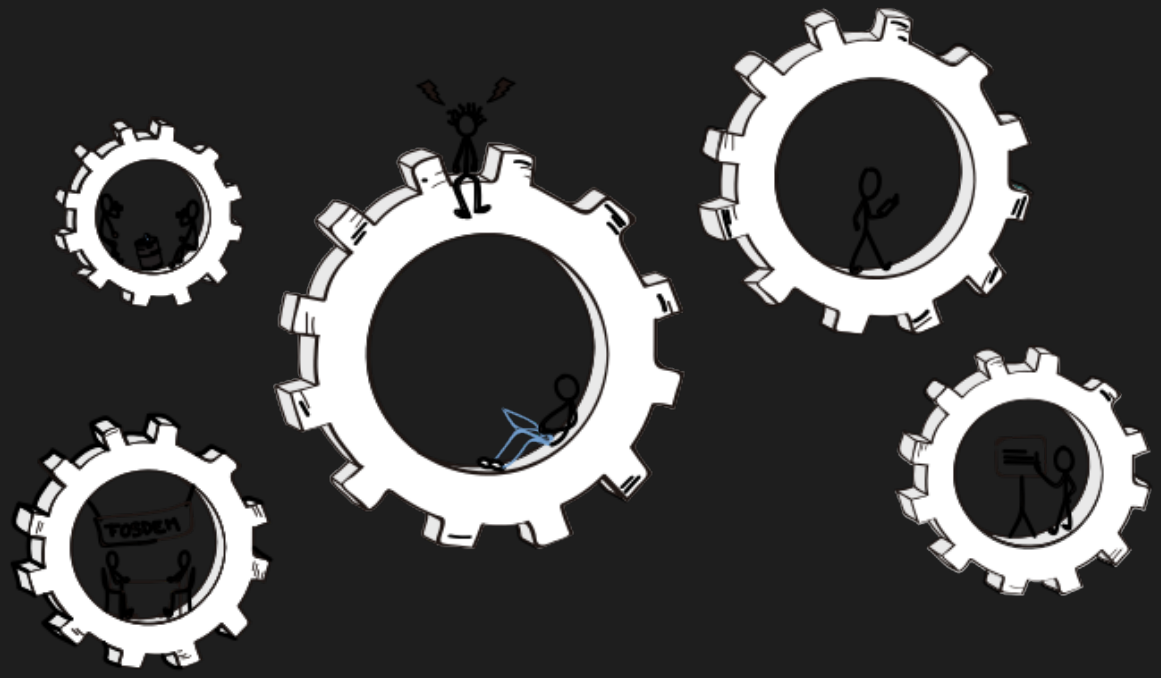
```
$ echo -n  $INTERFACE_DIR > \
    /sys/bus/usb/drivers/$DRV_NAME/unbind
```

- **Bind driver (device id must match)**

```
$ echo -n  $INTERFACE_DIR > \
    /sys/bus/usb/drivers/$DRV_NAME/unbind
```

Let's try this

DEMO



Plug & tell me more

FOSDEM '17

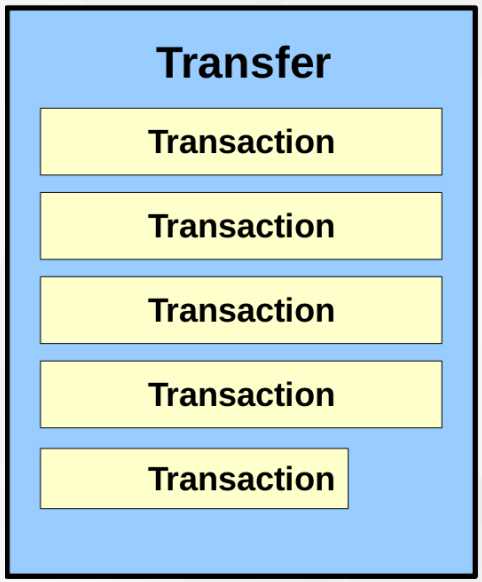
Brussels 4 & 5 February 2017

USB bus

- **USB is a Host-controlled bus**
- **Nothing on the bus happens without the host first initiating it.**
- **Devices cannot initiate any communication.**
- **The USB is a Polled Bus.**
- **The Host polls each device, requesting data or sending data.**

USB transfer vs transaction

- **Transaction**
  - Delivery of data to endpoint
  - Limited by wMaxPacketSize
- **Transfer**
  - One or more transactions
  - May be large or small
  - Completion conditions



USB Request Block

- Kernel provides hardware independent API for drivers
- This API is asynchronous
- URB is a kind of envelope for USB data

```
struct urb {
    struct list_head urb_list;

    struct usb_device *dev;
    unsigned int pipe;

    int status;
    unsigned int transfer_flags;
    void *transfer_buffer;
    u32 transfer_buffer_length;
    u32 actual_length;

    unsigned char *setup_packet;

    void *context;
    usb_complete_t complete;
};
```

Typical USB driver

Where?

- **probe()**
- **disconnect()**
- **complete()**
- **related to other subsystem**

What?

- **check device + allocate resources**
- **release resources**
- **check status, get data, resubmit**
- **depends on susbsys**

Typical bugs?

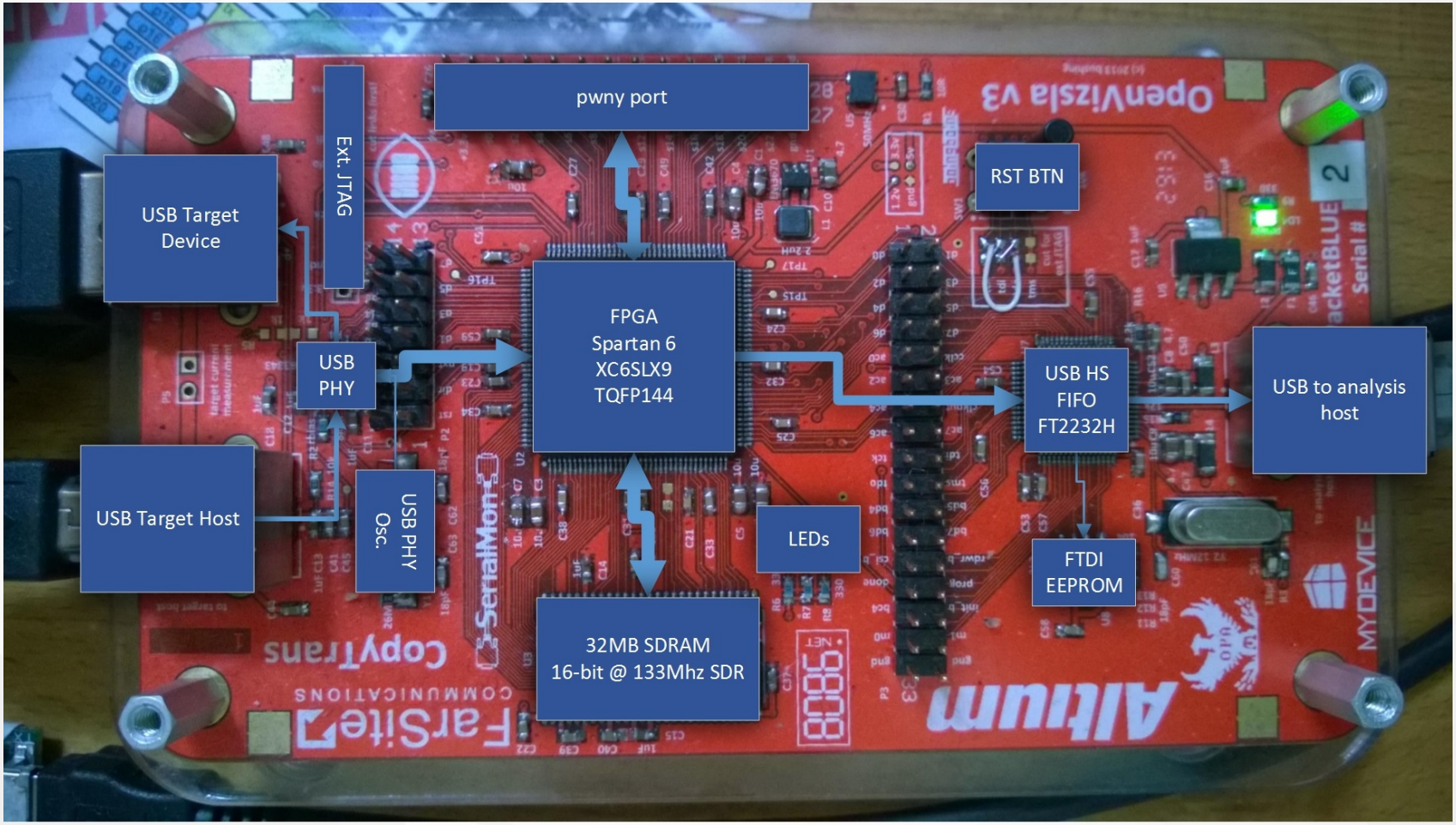
- **Missing descriptors**
- **No error path on missing entities**
- **No correct error handling in complete()**
- **Malformed packets**

HW USB sniffers - Commercial

2850$

1400$

HW USB sniffers - Open Hardware



about 100$

USBMon

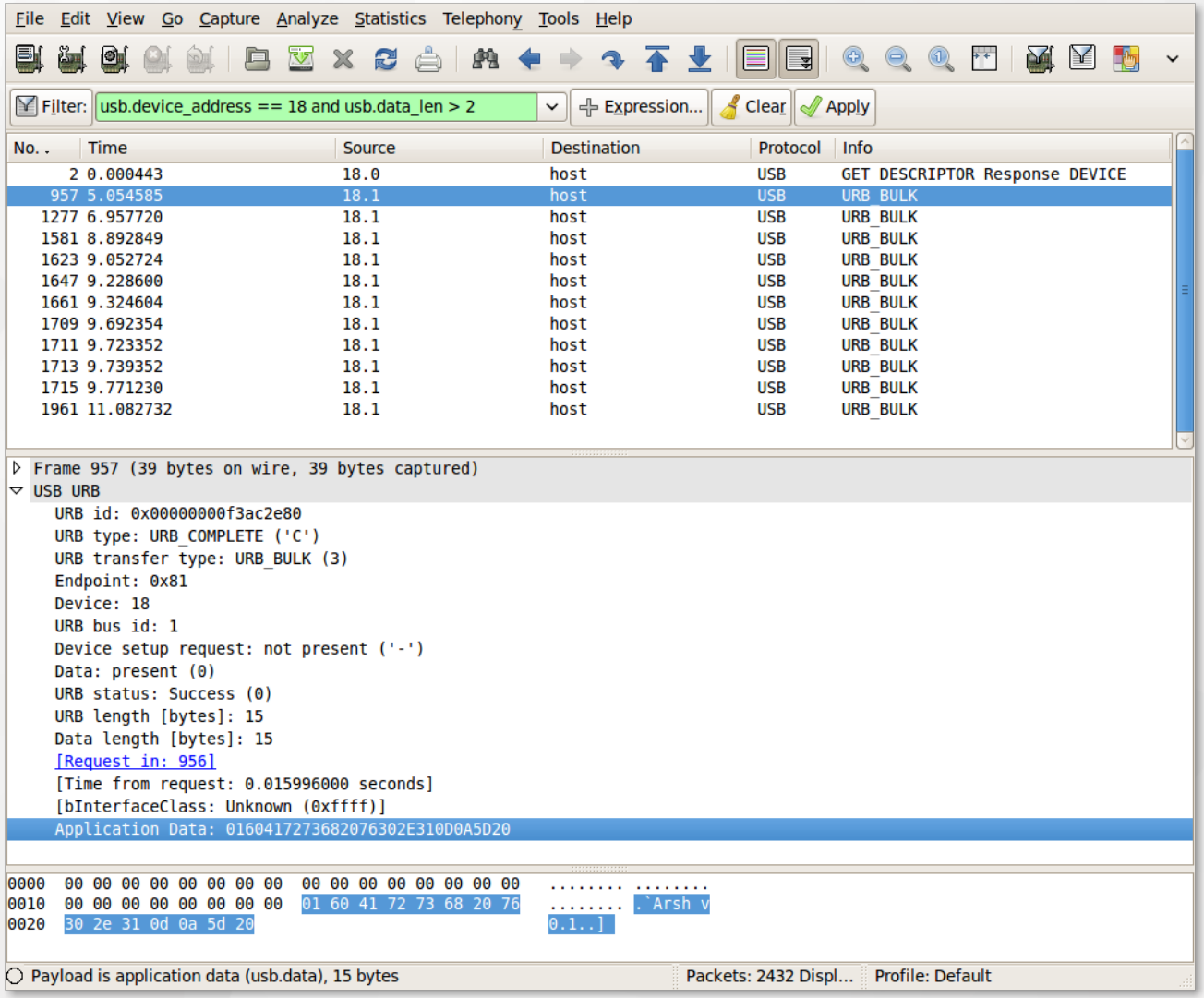
- **Kind of logger for URB related events:**
  - submit()
  - complete()
  - submit_error()
- **Text interface**
- **Binary Interface**
- **One instance for each USB bus**

submit vs complete

- **Data in URB buffer may is not always valid**
- **Validity depends on transfer results**
- **And on endpoint direction:**

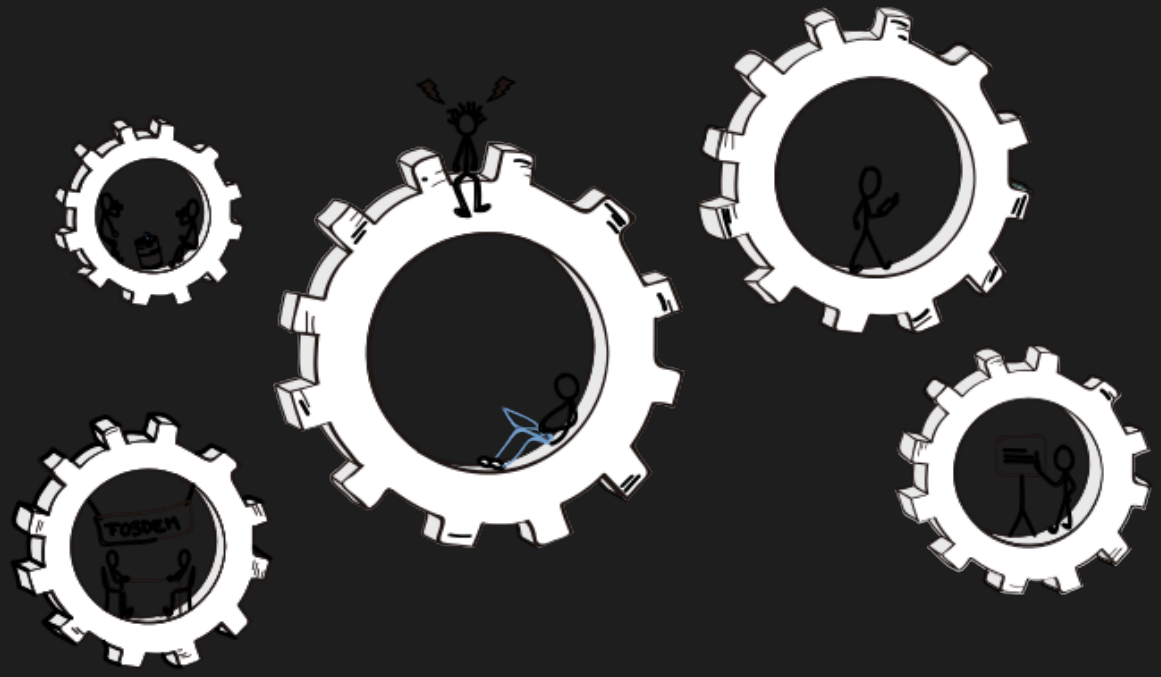
	IN	OUT
submit()	NO	YES
complete()	YES	NO

Good old friend Wireshark



Let's catch sth

DEMO



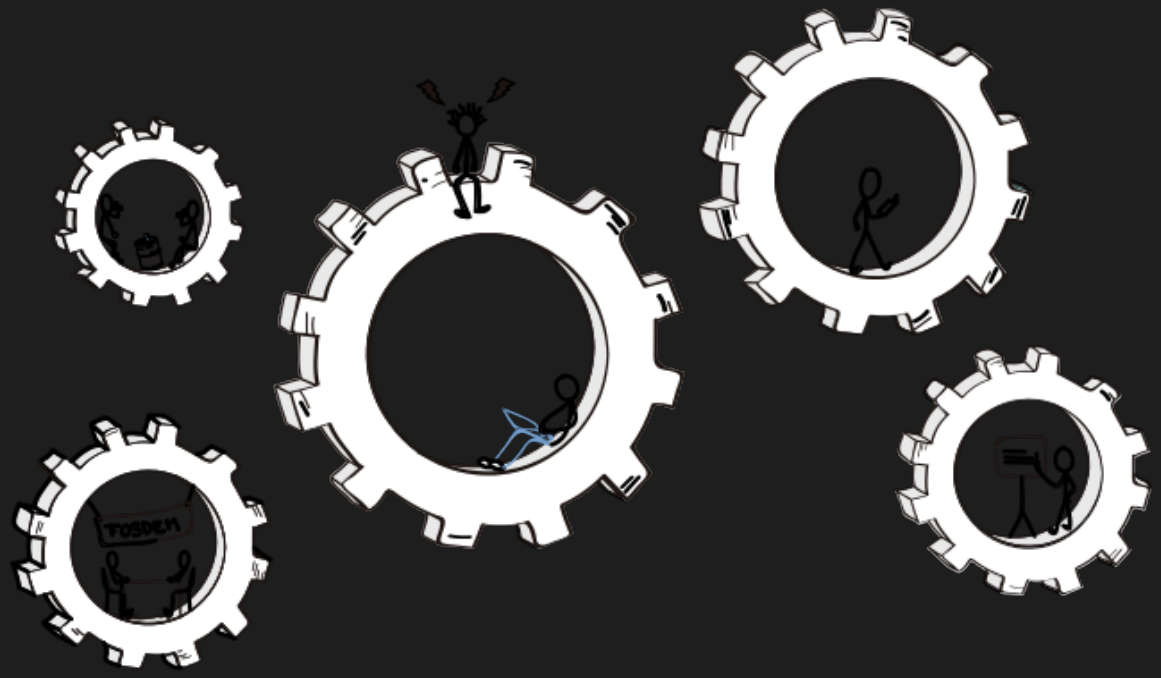
Summary

FOSDEM '17

Brussels 4 & 5 February 2017

Summary

- **USB descriptors are a device ID**
- **You can get them using *lsusb***
- **Drivers declares list of compatible devices**
- **USB devices are manageable via SysFS:**
  - Change active config
  - Add new device to driver
  - Black list device
  - Bind/Unbind driver
  - Device/Interface authorization
- **Drivers communicate using URBs**
- **In some cases USBMon can be used instead of expensive HW analyzers**



Q & A

FOSDEM '17

Brussels 4 & 5 February 2017

Thank you!

Krzysztof Opasiak

Samsung R&D Institute Poland

+48 605 125 174
k.opasiak@samsung.com