

Modernizing Android USB

Userspace AOA, Multi-UDC, Type-C and Framework APIs

Neill Kapron
Android USB

George Chan
Android Security



**LINUX
PLUMBERS CONFERENCE**

PRAGUE, CZECHIA / OCT. 5-7, 2026

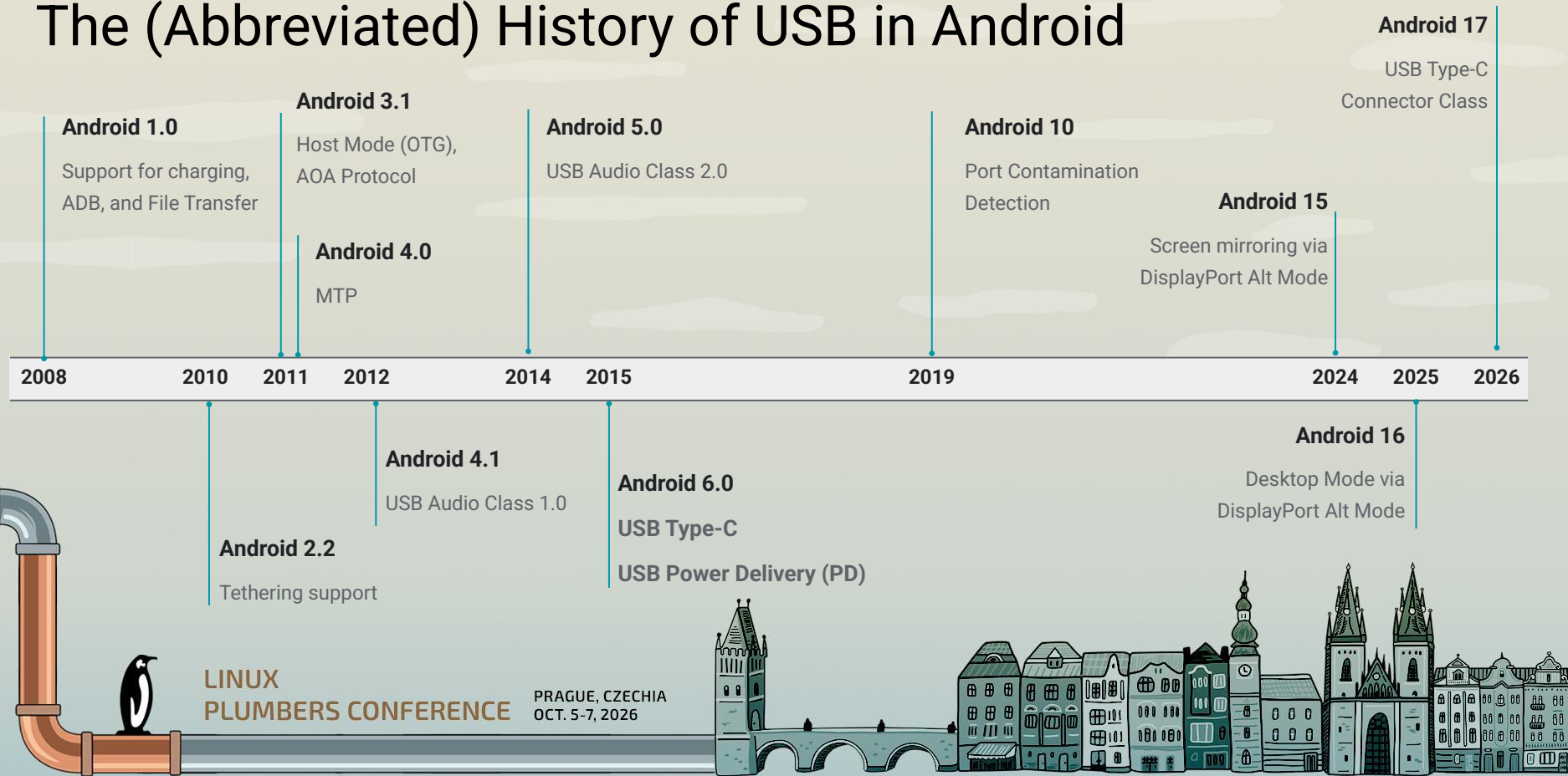


Agenda

- History of Android USB
- Current State
- Ongoing efforts
- Architectural Refactoring
 - Host Mode
 - Device Mode



The (Abbreviated) History of USB in Android



Android USB: The *Absence* Of An Architecture

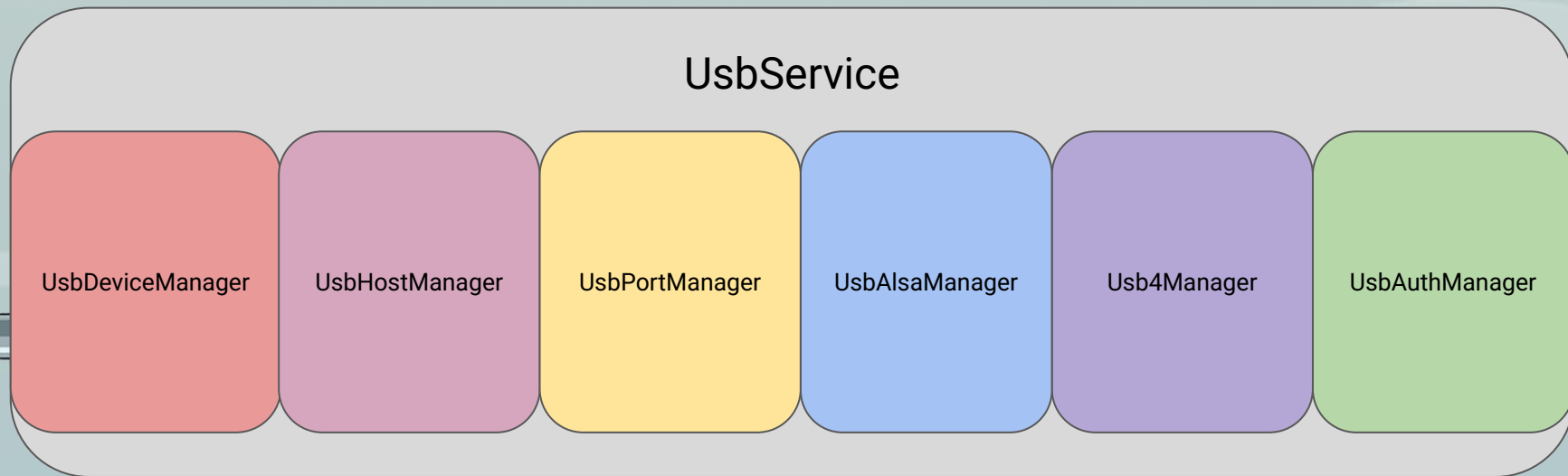
The USB stack in Android was designed for phones with a single USB port implemented as Mini-B, Micro-B, or a custom connector.

Each feature added over the last ~15 years brought with it its own control path, persistence, policy management, and API surface rather than extending a shared model.

- Poor internal design
 - Monolithic classes, singletons, no modularity, and limited testing
- Limited API surface
 - Limited app APIs, reliance on broadcasts, missing USB class support in frameworks
 - Apps rely on external libusb/NDK
- Bespoke permission model
 - Lacking granularity, not integrated with permission manager, etc.



Existing Core “Architecture”

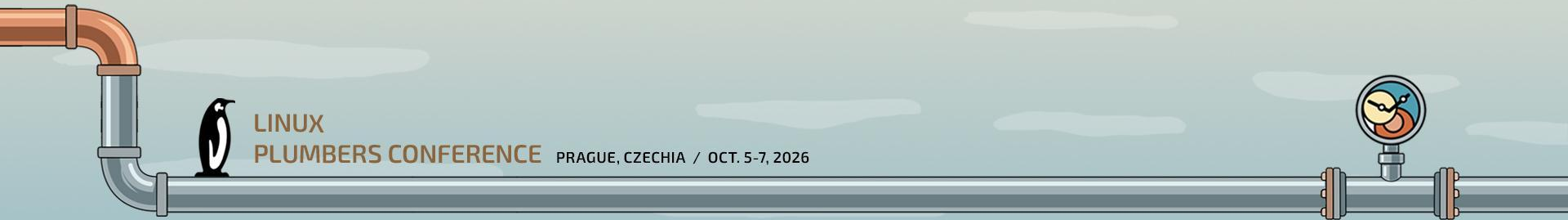


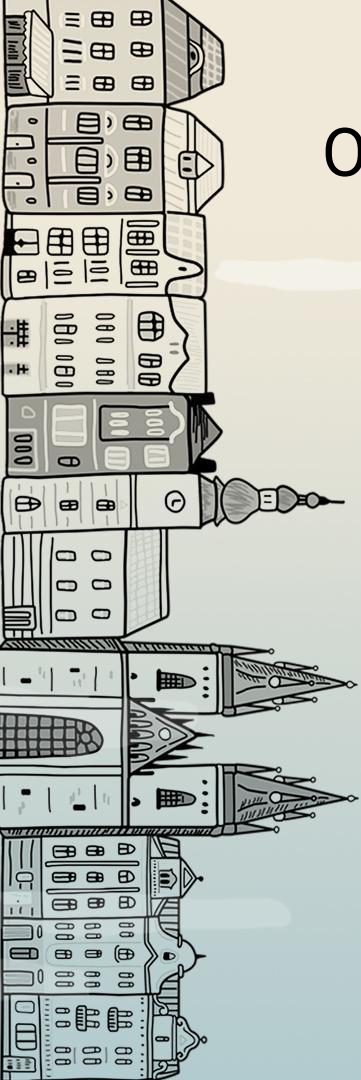
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

The Need for Architectural Refactoring

- Android has expanded past the handheld form factor
 - TVs, Watches, Laptops, Point of Sale devices, etc.
- Each form factor has its own unique requirements
- Each Android device has its own unique implementation
- We need modularity to support these different configurations
- Android frameworks and best practices have **evolved**, but the USB subsystem has **iterated**





Ongoing Projects

- **Userspace AOA**
 - Deprecate out-of-tree kernel driver and move AOA protocol handling to userspace
- **USB Authorization**
 - Restrict device access based on system policy and user interaction
- **USB Boot Security via AAPM**
 - Limit USB attack surface during boot
- **MTP Refactoring**
 - Improve performance and security
- **USB Visibility**
 - Expose USB capabilities via API
 - Update Settings App to make USB easier for users

LINUX PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Userspace Android Open Accessory (AOA) Protocol

- AOA protocol enables Android apps to communicate with USB hosts
 - Protocol negotiation occurs via vendor control requests
 - Exposes a simple read & write file descriptor to apps for data transmission
 - Support for HID interfaces
- Used by Android's Backup and Restore process, Android Auto, and custom hardware
- Historically implemented as an **out of tree USB gadget driver**
 - 6.1 and earlier: `f_accessory.c`
 - 6.6+: `android_f_accessory.c`

UAOA deprecates this driver by moving the protocol handling to userspace via FunctionFS

Upstream changes merged:

- [68e1e908cb76](#) selinux: enable per-file labeling for functionfs
- [82cfd4739011](#) usb: gadget: f_fs: Initialize epfile->in early to fix endpoint direction checks
- [8bdcf96eb135](#) usb: gadget: f_fs: Tie read_buffer lifetime to ffs_epfil
- [c3249c462b30](#) usb: gadget: f_fs: Add zero-length packet ioctl
- [0f0ea552cd21](#) usb: gadget: f_fs: Introduce rw_proxy file descriptors



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

USB Advanced Protection Mode

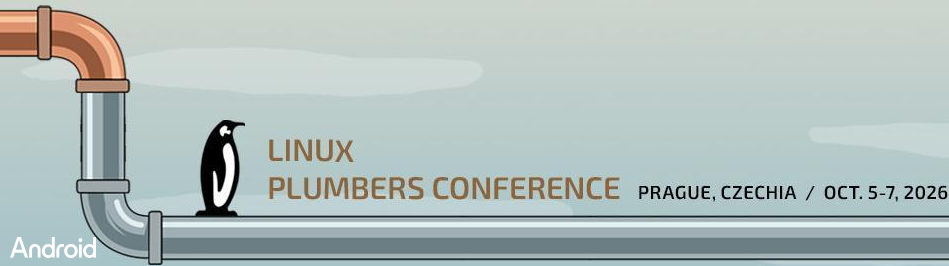
- Users can enable AAPM for increased USB security from boot
 - Disables ALL USB data signalling without limiting charging
- USB Type-C enables attack vectors beyond USB via Alt modes
 - USB4/Thunderbolt/DisplayPort
 - DMA/PCI
- Fragmentation of USB hardware implementation
 - TCPC + TCPM / EC (UCSI/Custom) / ?
 - Currently implemented via HALs
 - Difficult to validate across the ecosystem



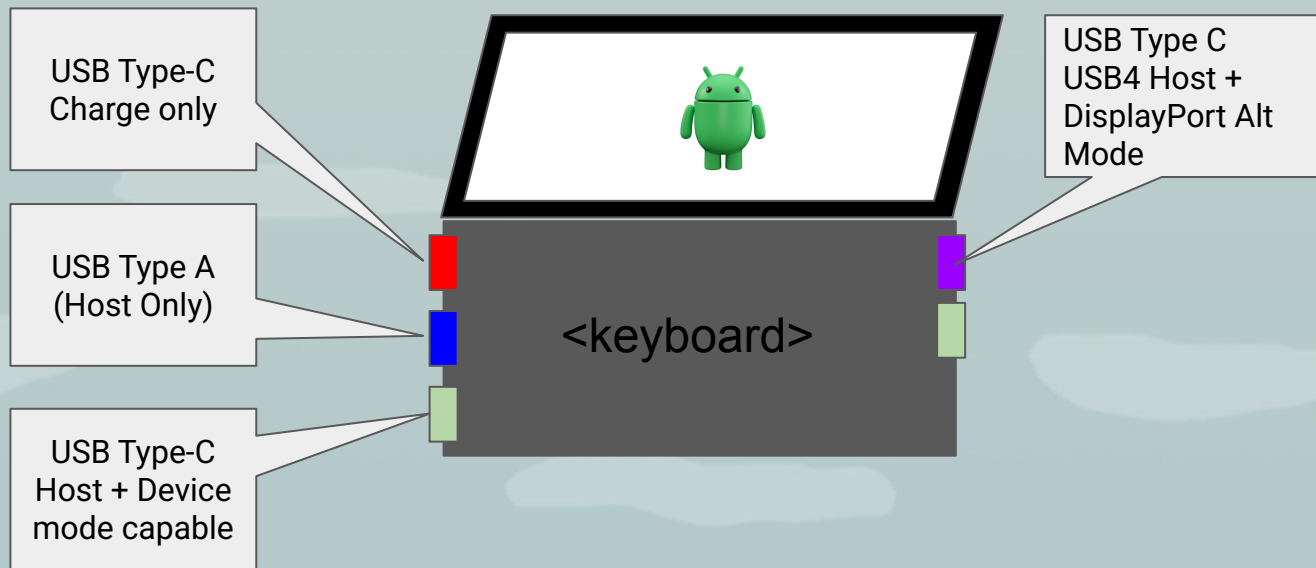
USB Advanced Protection Mode

Pursuit of a universal switch

- New Type-C Connector class UAPI
- Need to identify the right trust boundary
 - Type-C exists outside USB subsystem boundaries
 - Driver Binding?
 - Enumeration?



Multi Port USB



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

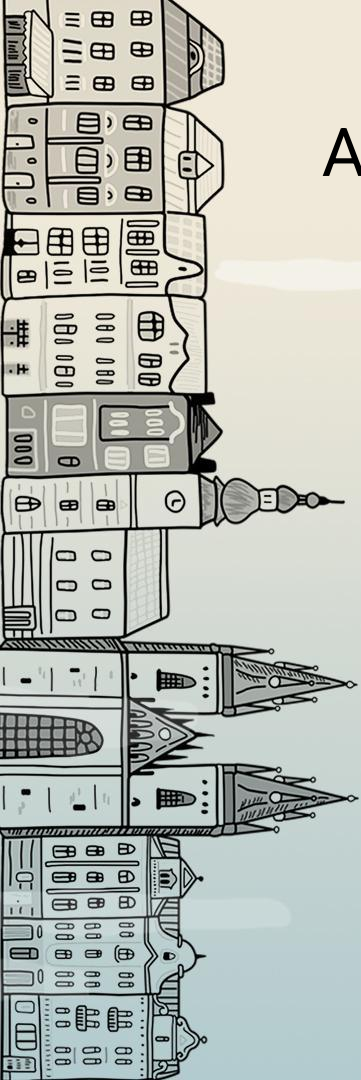
Multi Port USB

- Desktop class devices often have more than one USB port
- Android Watch or TV devices may not have a USB Type-C port
- Each USB port has its own capabilities
 - Host Mode
 - Device Mode
 - USB4/Thunderbolt/DisplayPort Alt modes
 - Vendor Alt modes
 - System and port power capabilities/limits
- Each port may have different hardware topologies
 - Embedded Controllers
 - UCSI
 - Hardware Multiplexers
 - ...?



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026



Architectural Requirements

- Topology focused
- Modularized features
- Extensible
- Merged peripheral permission model
 - Per-port
 - Per-interface
 - Host / Device mode
 - Alt Modes
 - Enterprise hooks
- Rich APIs for both System and App use cases
- User mode (app) drivers?
- Multiple UDC support



Goal: Make USB HALs 'Thin'

Move standardized kernel logic into the USB core, reducing platform bringup and maintenance costs

~60%

Existing USB HAL
covered by
standardized
kernel interfaces

~15%
Glue Logic

~25%

Vendor Specific
Hardware Logic



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Goal: Make USB HALs 'Thin'

Current HALs provide several mechanisms not found in upstream kernels:

- Contaminant/moisture detection (Type-C connector class?)
- Data-path disable (TCPM / UCSI / EC ?)
- Topology linkage: Connector -> UDC / root-hub port / DRM connector / power supply
- Connector physical location (mirror ACPI_PLD in DT?)
- Charger compliance detection/reporting



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Goal: Align Gadget HAL with Upstream Kernel ConfigFS

- Paradigm shift
 - Dynamic function allocation/mounting by core code
 - Vendor code needs access to some f_fs dirs for configuration
 - Core would need access for linking config to UDC
- Challenge: UDC Endpoint Resource Validation
 - UDC hardware resources vary device to device
 - Not straightforward <X> IN endpoints, <Y> OUT endpoints - other resources in play
- Unknown: SELinux policy ownership



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Android Host Mode Stack



LINUX
PLUMBERS CONFERENCE

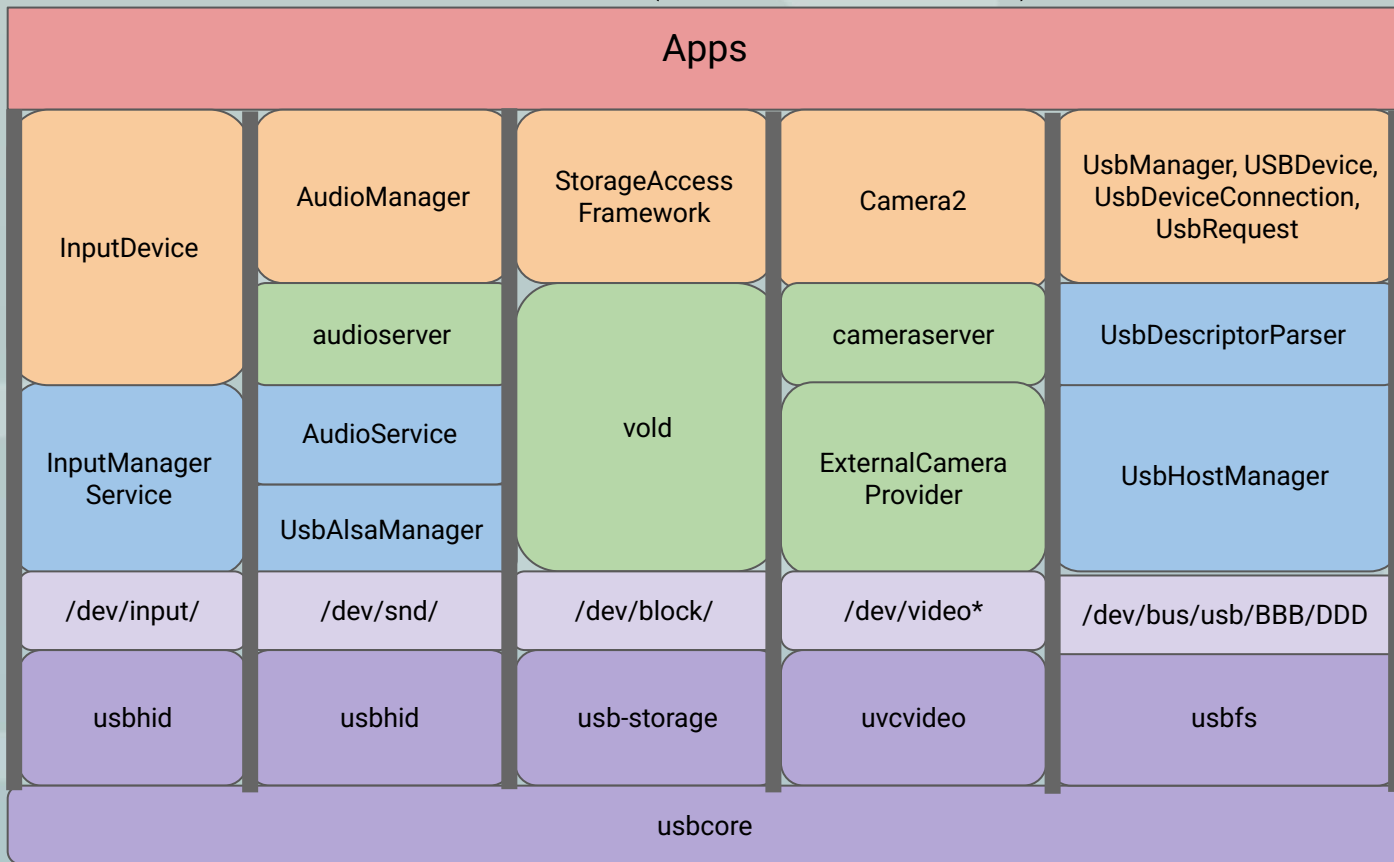
PRAGUE, CZECHIA / OCT. 5-7, 2026



Current Host Mode Architecture (Abbreviated)

Color Key:

- App
- Framework API
- Native
- System Server
- Kernel UAPI
- Kernel



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Current Host Mode Architecture Issues

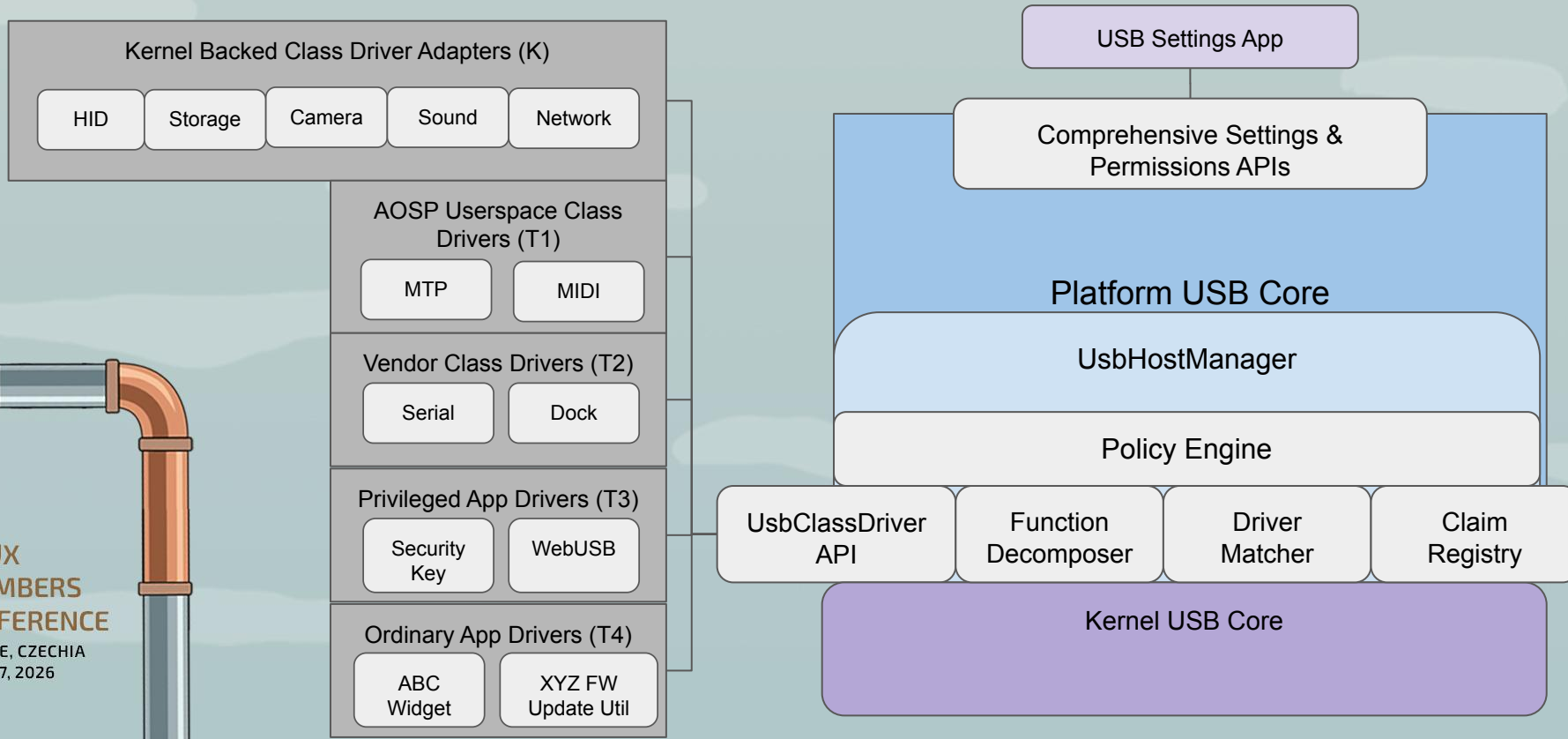
- Separate domains:
 - Kernel drivers (usbhid, snd-usb-audio, usb-storage, uvcvideo, etc)
 - AOSP class drivers (MTP, MIDI)
 - App leveraging raw Usbfs FD
- Apps can claim devices
 - unbind/bypass class drivers
- Inconsistent permissions
 - An app granted permission to the device can access all interfaces
 - Permissions may not match those required via other frameworks
- No communication between system drivers/subsystems and USB core
 - When an app claims the device via usbfs, no notification to class subsystem
 - App1 using Camera2 API -> App2 claims device via usbfs -> force unbind from kernel driver backing Camera2



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Proposed Host Mode Architecture



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Proposed Host Mode Architecture

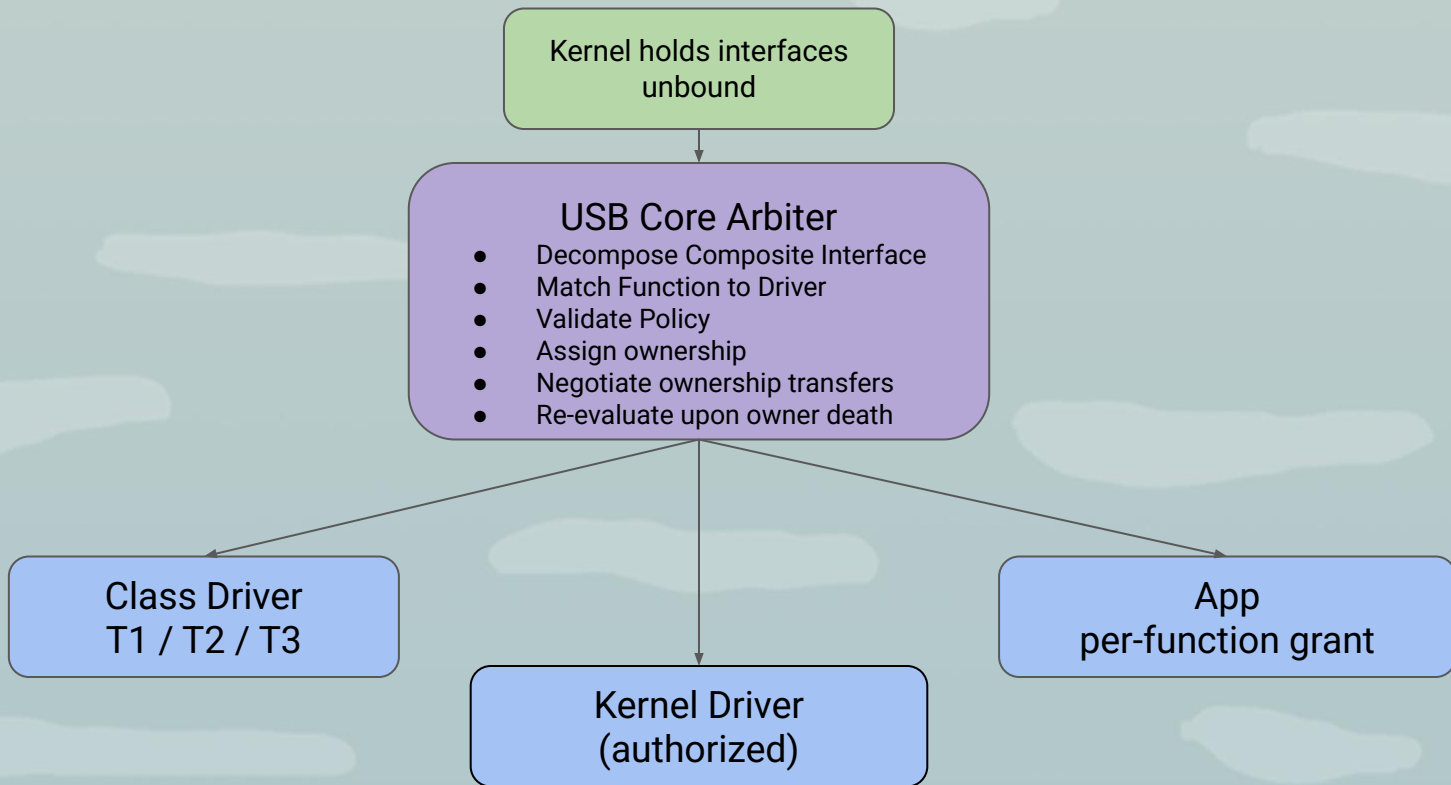
- USB Core acts as an arbitrator to class drivers
- Userspace Class drivers register with Core
- Core decomposes composite devices into usable function groups based on IAD
 - UVC: VideoControl + VideoStreaming
 - UAC: AudioControl + AudioStreaming In/Out
- Tiered driver ownership priorities
 - Kernel/AOSP/Vendor/Privileged App/App
 - Ownership can be transferred via Core arbitration
- Lifecycle APIs cover matching/setup/teardown/errors



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Proposed Host Mode Architecture



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Proposal: Host Mode Class Drivers

```
parcelable ClassDriverDescriptor {  
    String driverId;  
    MatchRule[] matchRules;  
    int priority;  
    boolean exclusive;  
    int[] protectedClassesRequested;  
}
```

```
parcelable MatchRule {  
    int vendorId;  
    int productId;  
    int functionClass;  
    int functionSubClass;  
    int functionProtocol;  
}
```

```
interface IUsbClassDriver {  
    ClassDriverDescriptor getDescriptor();  
    /** Can you take this? */  
    void onFunctionOffered(in UsbFunction fn, long epoch, int offerTimeoutMs);  
    /** It is yours. */  
    void onFunctionClaimed(in UsbFunction fn, in ParcelFileDescriptor[]  
interfaceFds, long epoch);  
    /** Give it back. Reason: policy change, higher-priority  
    * driver registered, user override, enterprise policy, shutdown. */  
    void onFunctionRevoked(in UsbFunction fn, int reason, long deadlineMs, long  
epoch);  
    /** Device physically gone. Best-effort cleanup; no reply expected. */  
    void onFunctionDetached(in UsbFunction fn, long epoch);  
}  
  
interface IUsbClassDriverCallback {  
    void onOfferAccepted(String functionId, long epoch);  
    void onOfferDeclined(String functionId, long epoch);  
    void onReleased(String functionId, long epoch);  
    void onDriverError(String functionId, int error, long epoch);  
}
```



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Android Device Mode Stack



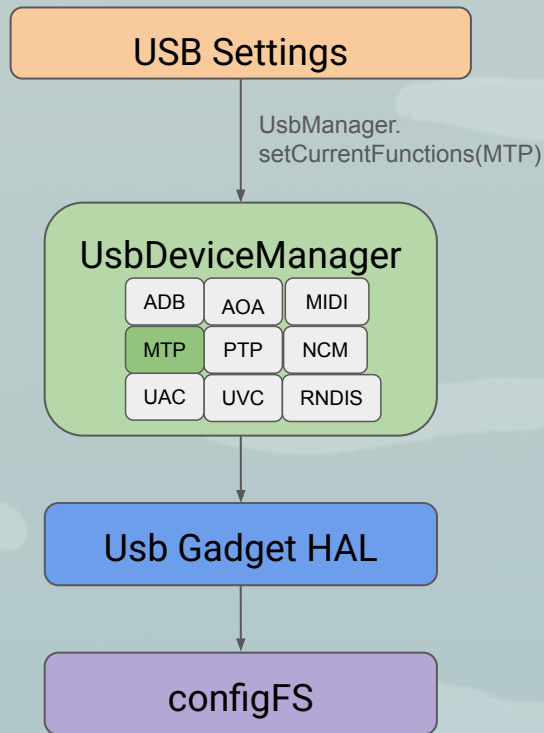
LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Android USB Device Stack Lacks Modularity

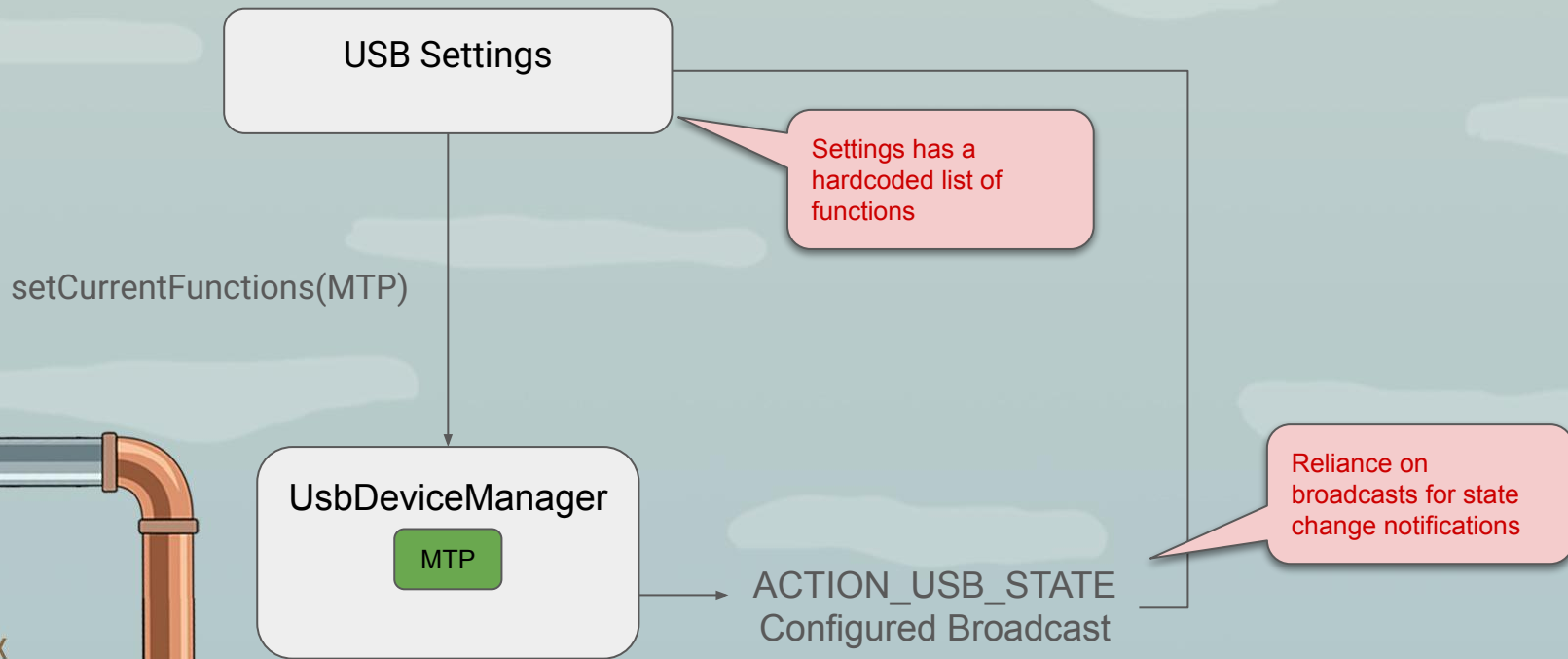
- Enabling UsbDeviceManager opts-in device to all functions
- UsbDeviceManager is a monolithic entity in system server
- Vendor customization requires modifying system code
- USB Settings has a list of hardcoded available function enum values
- Vendors' Gadget HALs are often mostly copy/paste copies device to device
- No clear owner of ConfigFS access



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

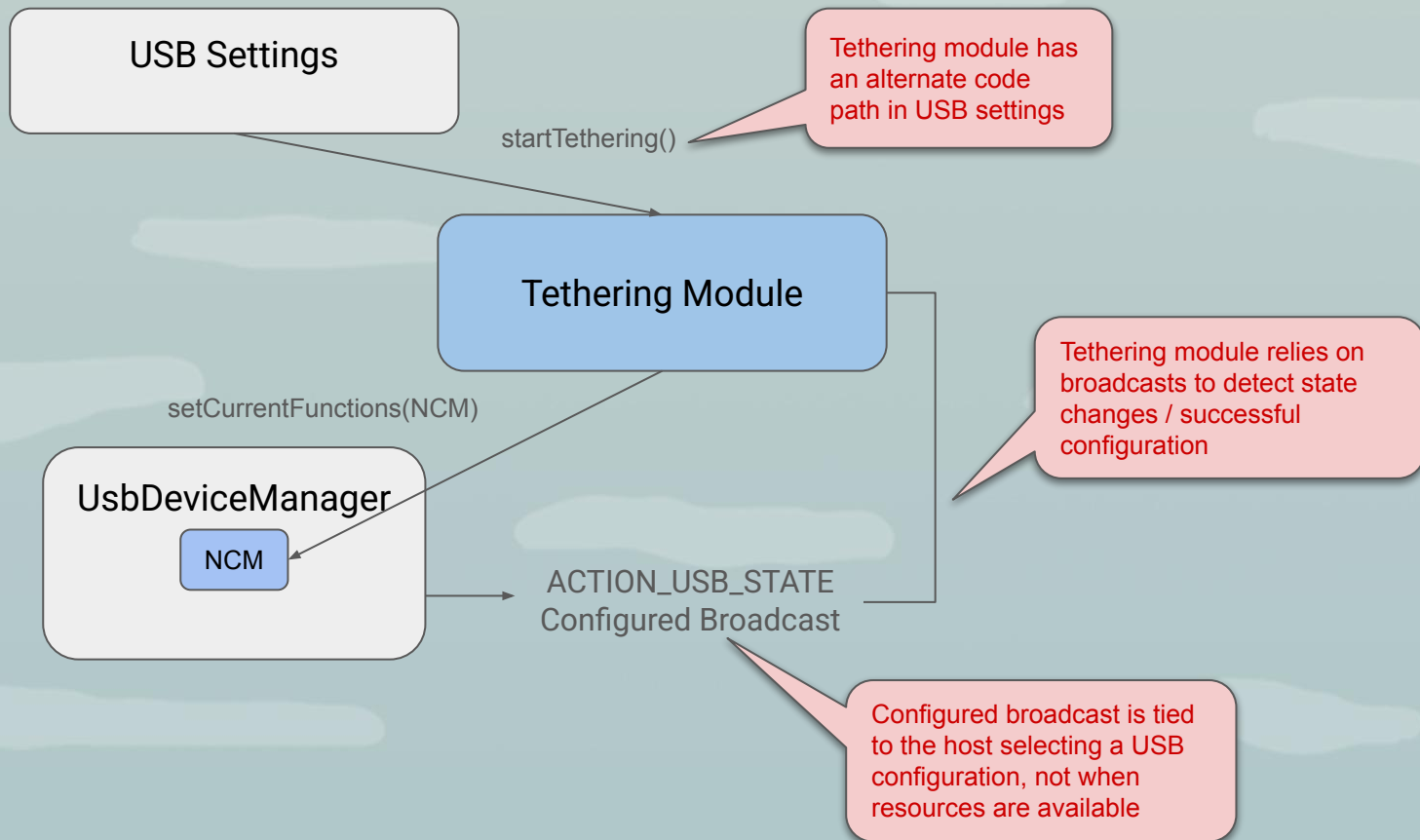
Existing Architectural Issues



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

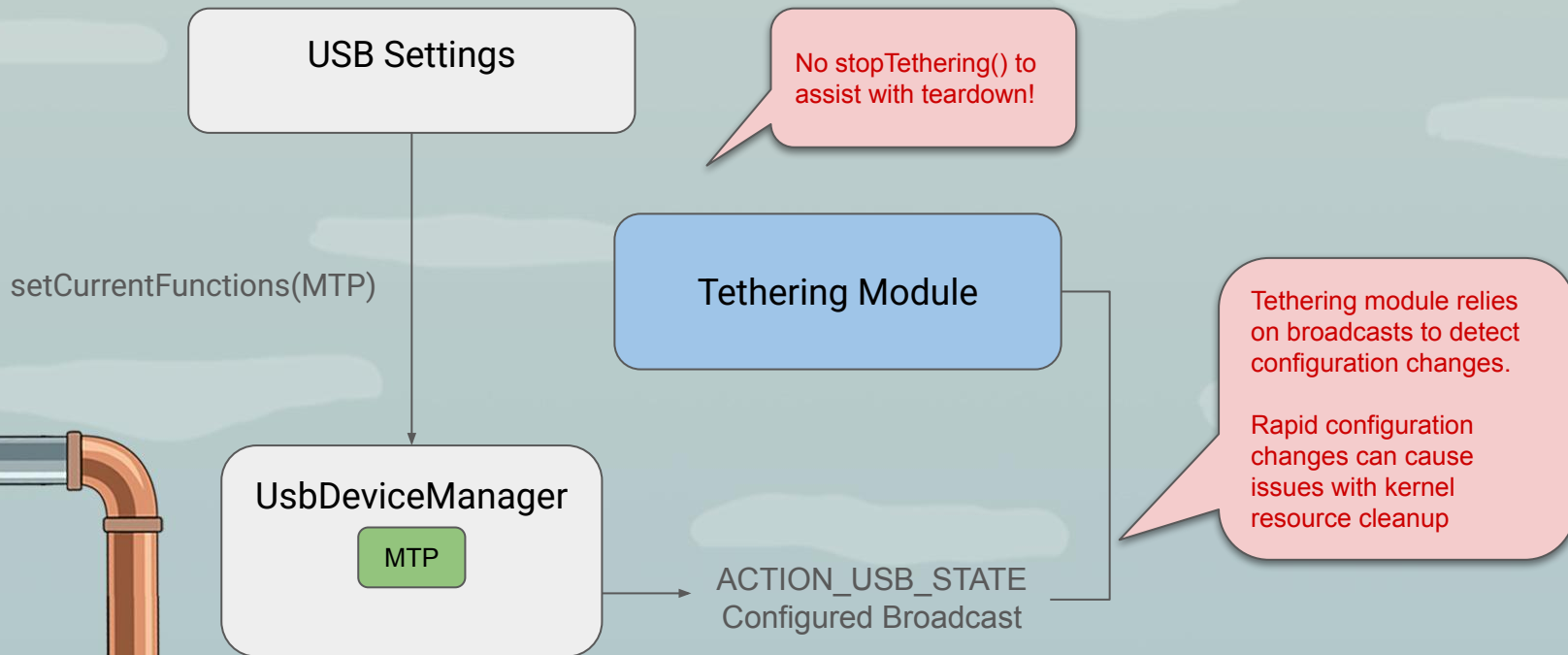
Existing Architectural Issues



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Existing Architectural Issues

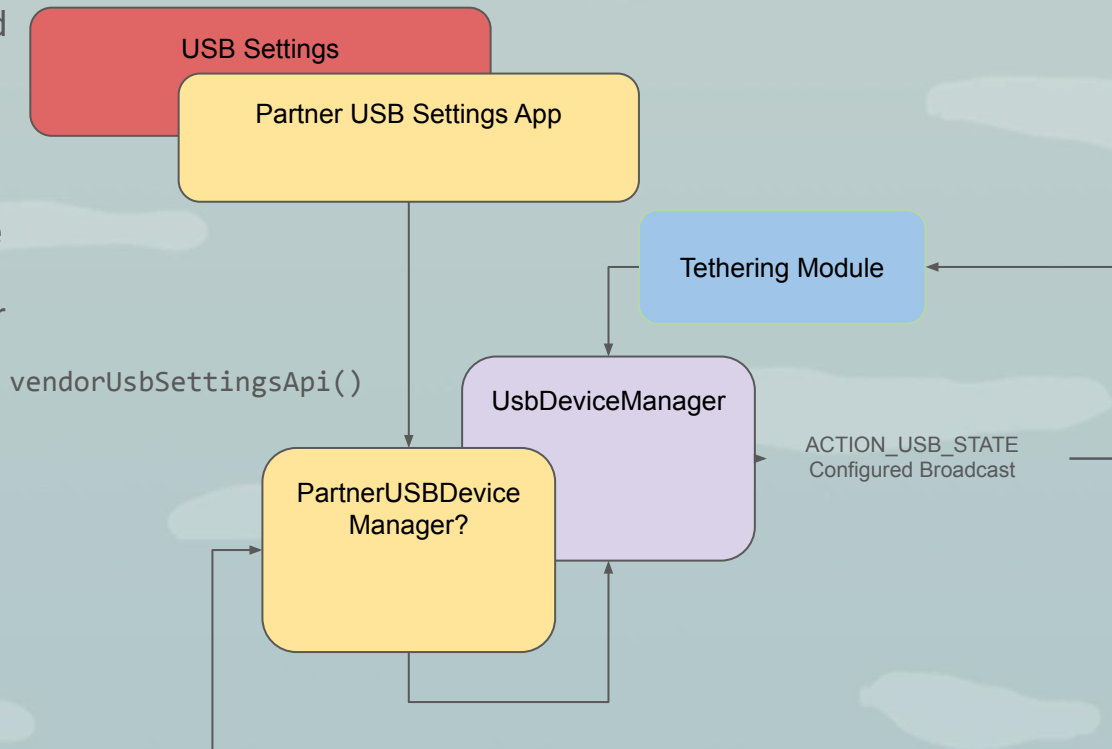


LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Partner implementations

- Due to the lack of modularity and modern API surfaces, partners replace or extend core functionality
- This extensive customization causes issues when features are added to the core, slowing adoption and increasing work for OEMs with each new platform release



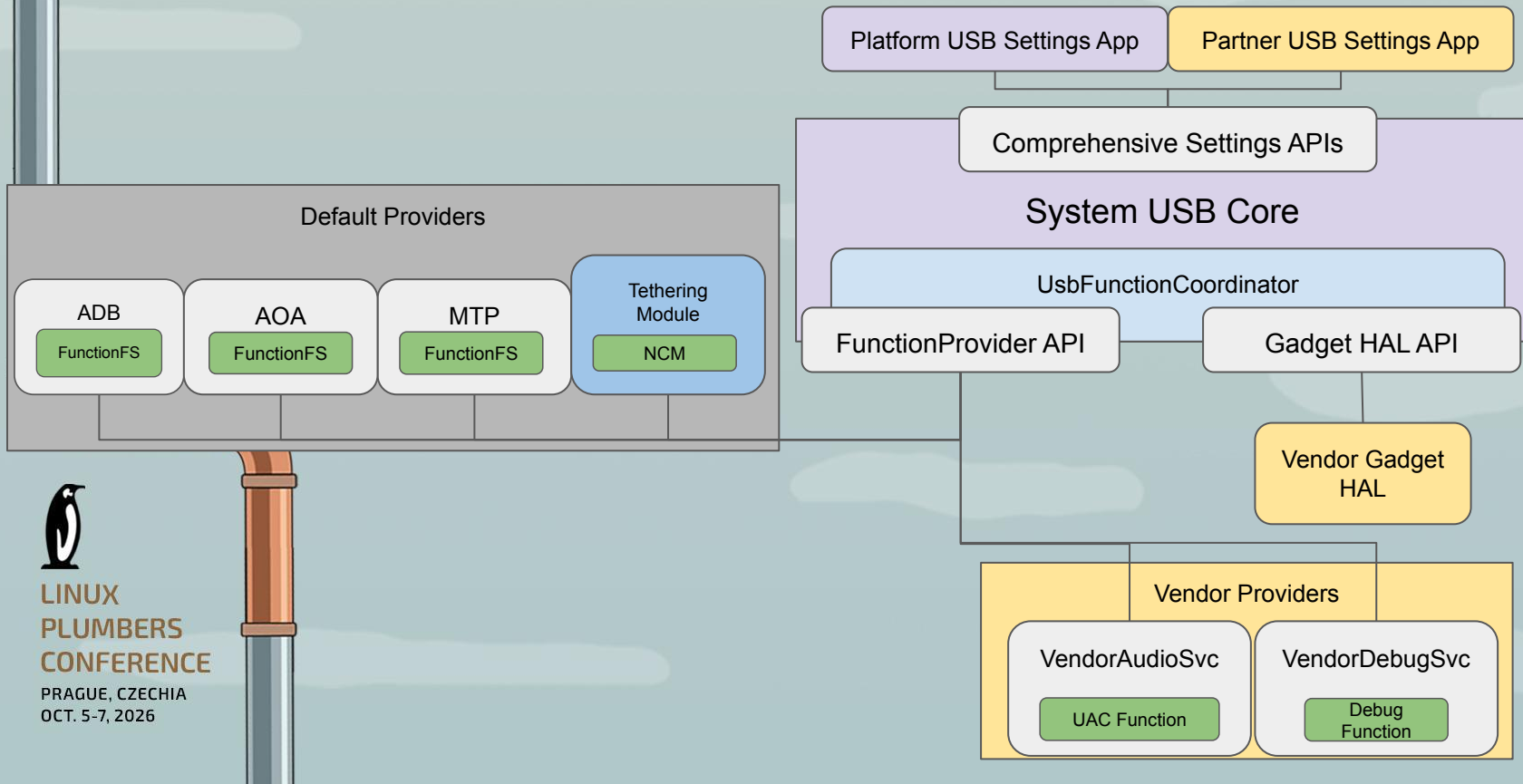
LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Color Key:

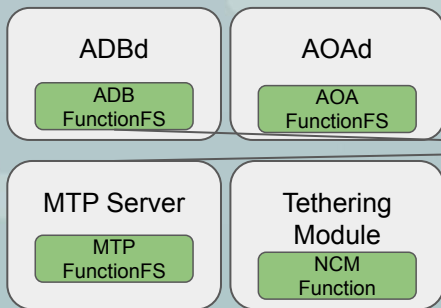
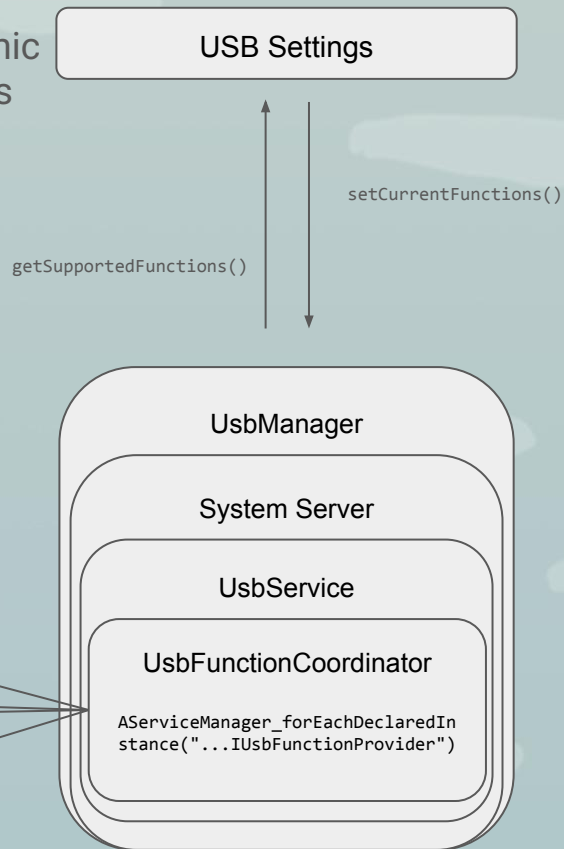
- Core Platform
- Mainline
- Partner
- Unused

Proposed Device Mode Architecture



Proposal: De-couple Functions with Provider API

- Introduce new `UsbFunctionCoordinator` Class which allows dynamic registration of gadget functions through standard AOSP interfaces
- Vendor opt-in to new modular architecture via system property
- Enables custom gadget functions to be registered
- Providers exist outside of system server for stability and security
- New APIs for Settings applications to dynamically query and set function configurations



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Proposal: De-couple Functions with Provider API

```
interface IUsbFunctionProvider {  
    FunctionDescriptor getDescriptor();  
    /** Called once at registration, before any lifecycle call. */  
    void setCoordinator(in IUsbFunctionCoordinator coordinator);  
  
    // Requests (provider must ack via IUsbFunctionCoordinator)  
    oneway void prepare(in FunctionInstance instance, in PrepareParams  
params);  
    oneway void start(in FunctionInstance instance);  
    oneway void stop(in FunctionInstance instance, in StopReason  
reason);  
  
    // Host-originated notifications (No Ack)  
    oneway void onConfigured(in FunctionInstance instance);  
    oneway void onDeconfigured(in FunctionInstance instance);  
    oneway void onSuspended(in FunctionInstance instance);  
    oneway void onResumed(in FunctionInstance instance);  
}
```

Gadget Descriptor Fields retrieved by
coordinator calling
FunctionProvider.getDescriptor()

```
functionId          "vendor.acme.diag"  
kernelFunctionName  "ffs.acme_diag"  
userspaceImplemented true  
requiredInEndpoints / requiredOutEndpoints  
conflictsWith       ["ncm", "rndis"]  
supportedPorts      [port ids or ALL]  
userSelectable      false  
label               <localized CharSequence>  
requiredPermission  "com.acme.permission.USB_DIAG"
```

```
interface IUsbFunctionCoordinator {  
    oneway void onPrepared(in FunctionInstance instance);  
    oneway void onStarted(in FunctionInstance instance);  
    oneway void onStopped(in FunctionInstance instance);  
  
    // Failure for this instance. Triggers rollback.  
    oneway void onFunctionError(  
        in FunctionInstance instance,  
        in FunctionError error,  
        in String detail);  
}
```

Questions?



LINUX
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

