udev overview for newbies

Haifa Linux Club

March 27, 2006

Paramahansa Polo

paramahansa@gmail.com

History: A Past Since Forgotten

- The steps for installing a new hardware peripheral on a Mac might go a bit like this:
  - Step 1: plug hardware in to Mac.
  - Step 2: begin using hardware.
- Most of us would not even include these two items as steps. The first is a physical necessity; the second is the original and ultimate goal.

History: A Past Since Forgotten

- At some point in Linux's history, support for new hardware could easily require compiling a new kernel module, becoming root, editing configuration files, loading said module, checking dmesg, cursing, removing the module, unplugging the hardware, plugging the hardware back in, reloading the module and so on.

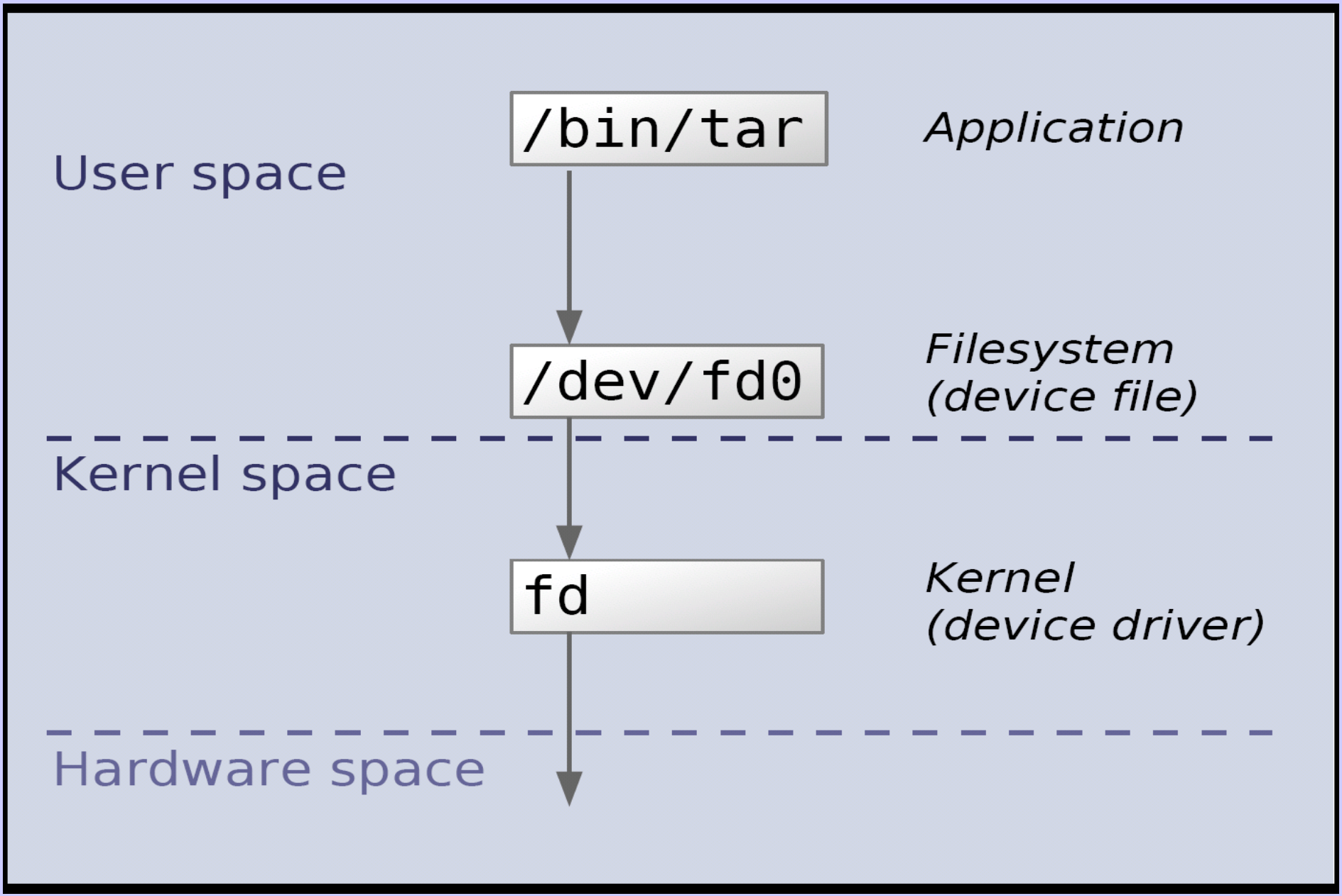
The Problem

- **Hot Swapping**: the ability to remove and replace components of a machine, usually a computer, while it is operating

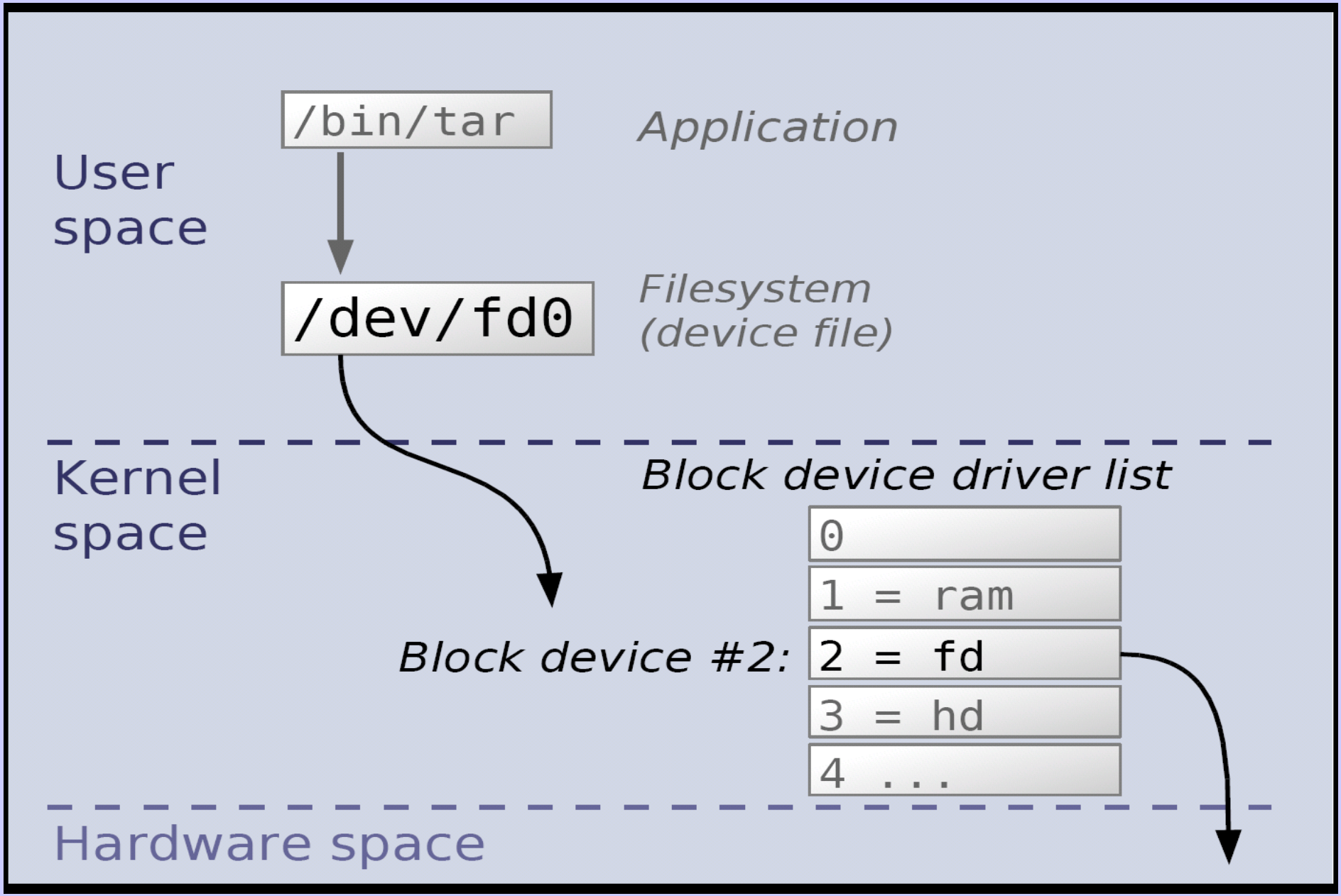
  Example:

  the Universal Serial Bus (USB), that allows a user to add or remove peripheral components such as a mouse, keyboard, or printer.

How was it in the past ?



How was it in the past ?



What entry is which device ?

- When the kernel finds a new piece of hardware, it typically assigns the next major/minor pair for that kind of hardware to the device.
  Example:
  on boot, the first USB printer found would be assigned the major number 180 and minor number 0, which is referenced in /dev as /dev/usb/lp0. The second USB printer would be assigned major number 180 and minor number 1, which is referenced in /dev as /dev/usb/lp1

Advantages

- Simple
- Flexible
- Easy to understand
- Easy to configure
- Power abstraction at low cost

But, what if ?

- *ACTION*: the user rearranges the USB topology, perhaps adding a USB hub to support more USB devices in the system.
- *RESULT*: the USB probing order of the printers might change the next time the computer is booted, reversing the assignment of the different minor numbers to the two printers

Requirements !

- Support for “hot plug” devices (e.g, usb)
  - consistent device names
  - dynamic system config
- Good interactive user experience:
  - Easy, consistent access
  - Report device status
  - Easy installation

What we have ?

- Dynamic system configuration:
  - It was limited to <500 diferent devices, in the entire linux universe
  - Limited to 16-250 instances of any particular device.
  - With linux 2.6 the valid range of major numbers increased to 4,096. And more than a million minor numbers are available per major number.

What we have ?

- Dynamic system configuration
  - Device files are created for every possible device, including devices that are not physically installed
    Example:
    On a machine running Red Hat's Fedora release 1, the /dev directory holds more than 18,000 different entries.

What we have ?

- Consistent device names:
  - Device names (specially SCSI and USB), may change acroos systems boots.

... remember the example?

What we have ?

- Good interactive user experience

I just love it !!!

New Alternatives

- **devfs**: an automated and dynamic manager of device nodes. A RAM-based filesystem, alternative to "real" character and block special devices on your root filesystem. Kernel device drivers can register devices by name rather than major and minor numbers.
- **udev**: allows Linux users to have a dynamic /dev directory and it provides the ability to have persistent device names. It uses sysfs and /sbin/hotplug and runs entirely in userspace.

Goals for udev

- run in userspace
- dynamically creates/removes device files (dynamic /dev)
  - allow everyone to not care about major/minor numbers
- provide consistent naming, if desired
  - provide LSB standard names
- provide a user-space API to access information about current systems devices.

In the middle game

- The device model allowed, for the first time, the kernel to build an in-memory tree of the devices it supported.

  Example:

  Both my mouse and my keyboard are connected to my USB hub, which is connected to my third USB port, which is on my first PCI bus.

- Such a rich hierarchy provides all sorts of opportunities to the kernel. One of the most promising, however, was sysfs

SYSFS EXPORTS THE IN-MEMORY TREE DEVICE HIERARCHY AS A FILESYSTEM

- One directory lists all the buses on a system.
- For each bus, another directory lists all of the devices on a given bus.
- Files for a given device could link to the associated module.
- Walking the sysfs tree, therefore, would allow user space to build a comprehensive picture of the system's physical device hierarchy, exactly as the kernel sees it.

HOTPLUG, ANOTHER KERNEL FEATURE

- The kernel's hotplug infrastructure notifies user space whenever a device is added to or removed from the system.
- This allows applications to become aware of changes to sysfs in real time. It also allowed for the creation of udev.

Enter HAL

- HAL, originally hardware abstraction layer but now not an abstraction of anything whatsoever, is a system-level dæmon that ties together hotplug, sysfs and udev in order to provide a Linux system with a single, comprehensive view of hardware, accessible via a standardized interface

Enter HAL

- HAL makes it possible for an application to say, "give me the device nodes of all input devices" or to ask, "is there a camera connected to this computer?"
- HAL uses a nascent but always-promising project called D-BUS as its communications mechanism.

DBUS

- On one side, D-BUS is a run-of-the-mill interprocess communication (IPC) system-like CORBA, but a lot easier to use.
- On the other side, however, D-BUS introduces the concept of the system-wide message bus.

DBUS

- In addition to per-user process-to-process communication, D-BUS allows components in a Linux system to send out signals, announcing events or providing information to all who care to listen.
- Signals can announce when a network connection is obtained or when the laptop battery is running low. Interested applications higher up the stack can listen for these signals and, upon receipt, react.

UDEV FEATURES

- Are separated in three projects:
  - namedev
  - libsysfs
  - udev

NAMEDEV

- allows you to define the device naming separately from the udev program.
- Currently only a single naming scheme is provided by namedev; the one provided by LANANA, used by the majority of Linux systems.

NAMEDEV

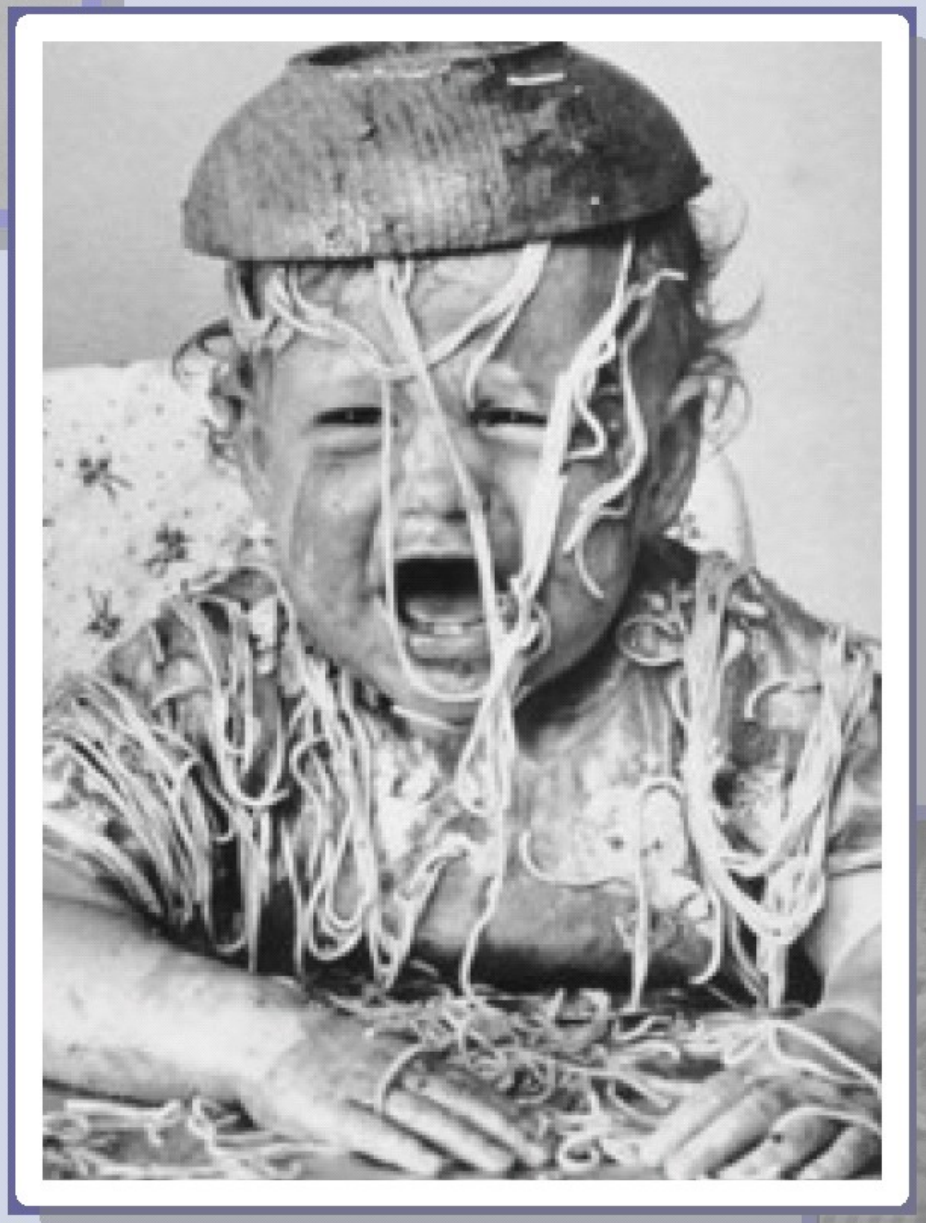
- Namedev uses a 5-step procedure to find out the name of a given device. If the device name is found in one of the given steps, that name is used. The steps are:
  - label or serial number
  - bus device number
  - bus topology
  - statically given name
  - kernel provided name

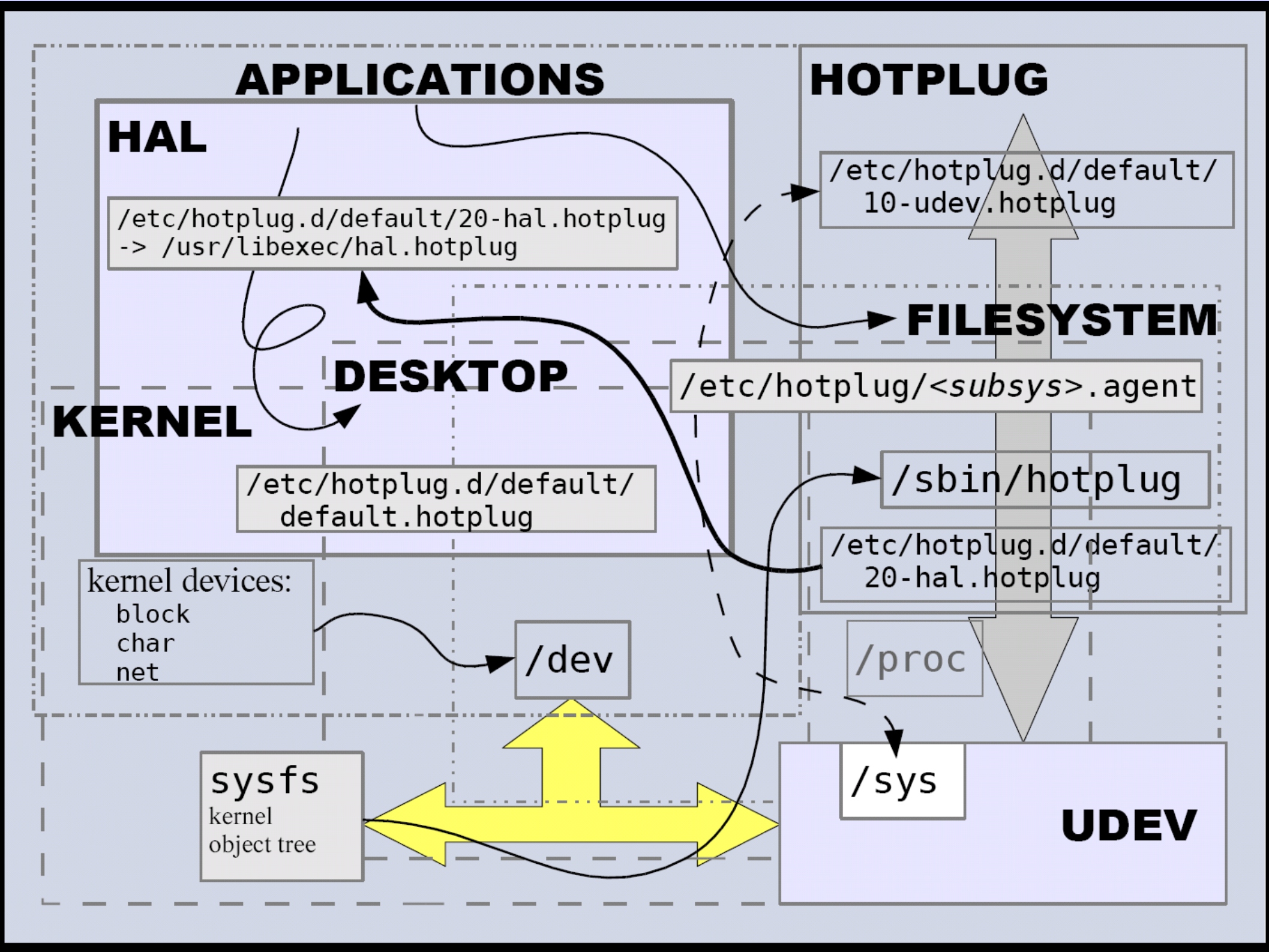
LIBSYSFS

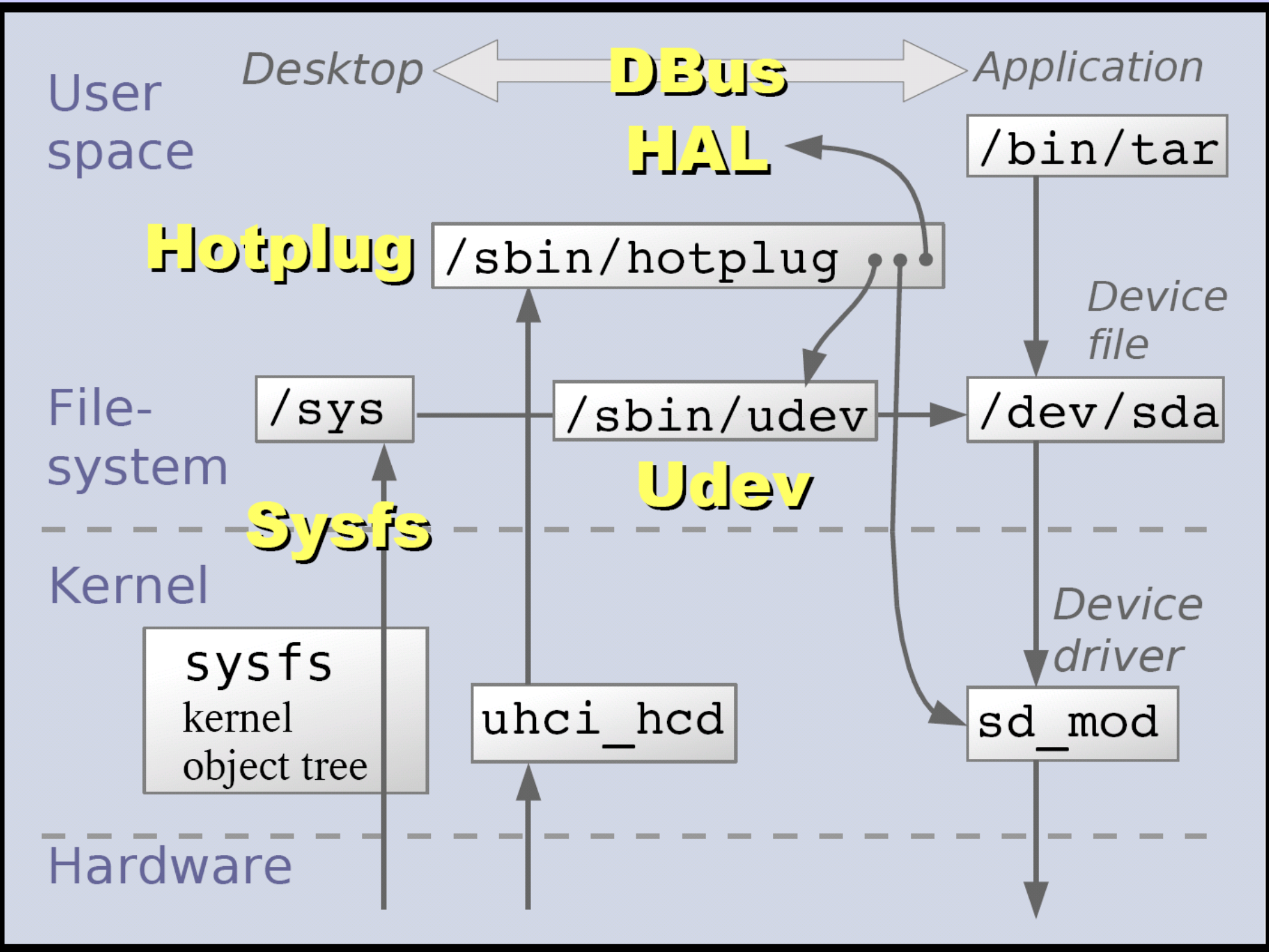
- udev interacts with the kernel through the sysfs pseudo filesystem.
- The libsysfs project provides a common API to access the information given by the sysfs filesystem in a generic way. This allows for querying all kinds of hardware without having to make assumptions on the kind of hardware.

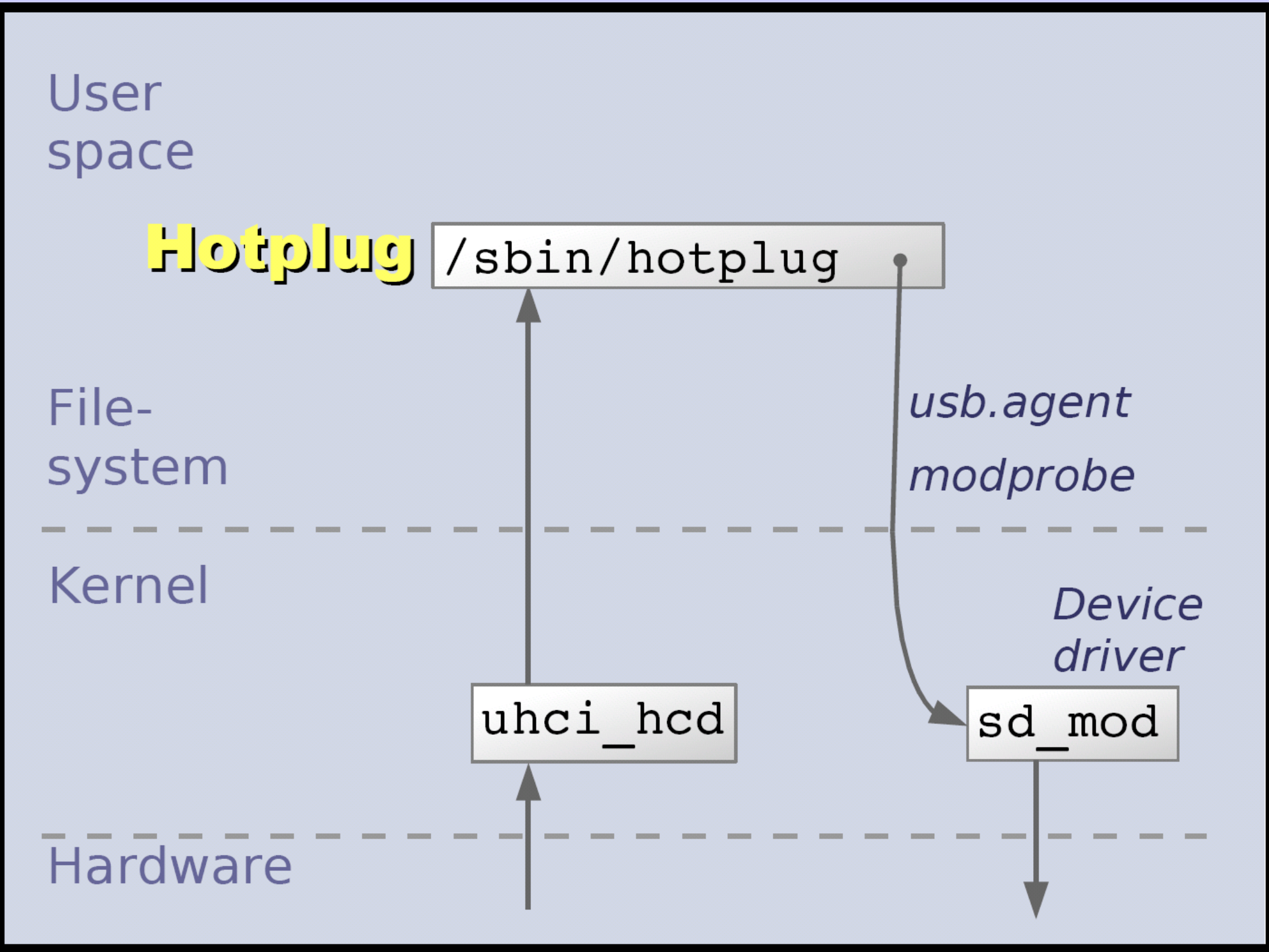
Where we've been . . .

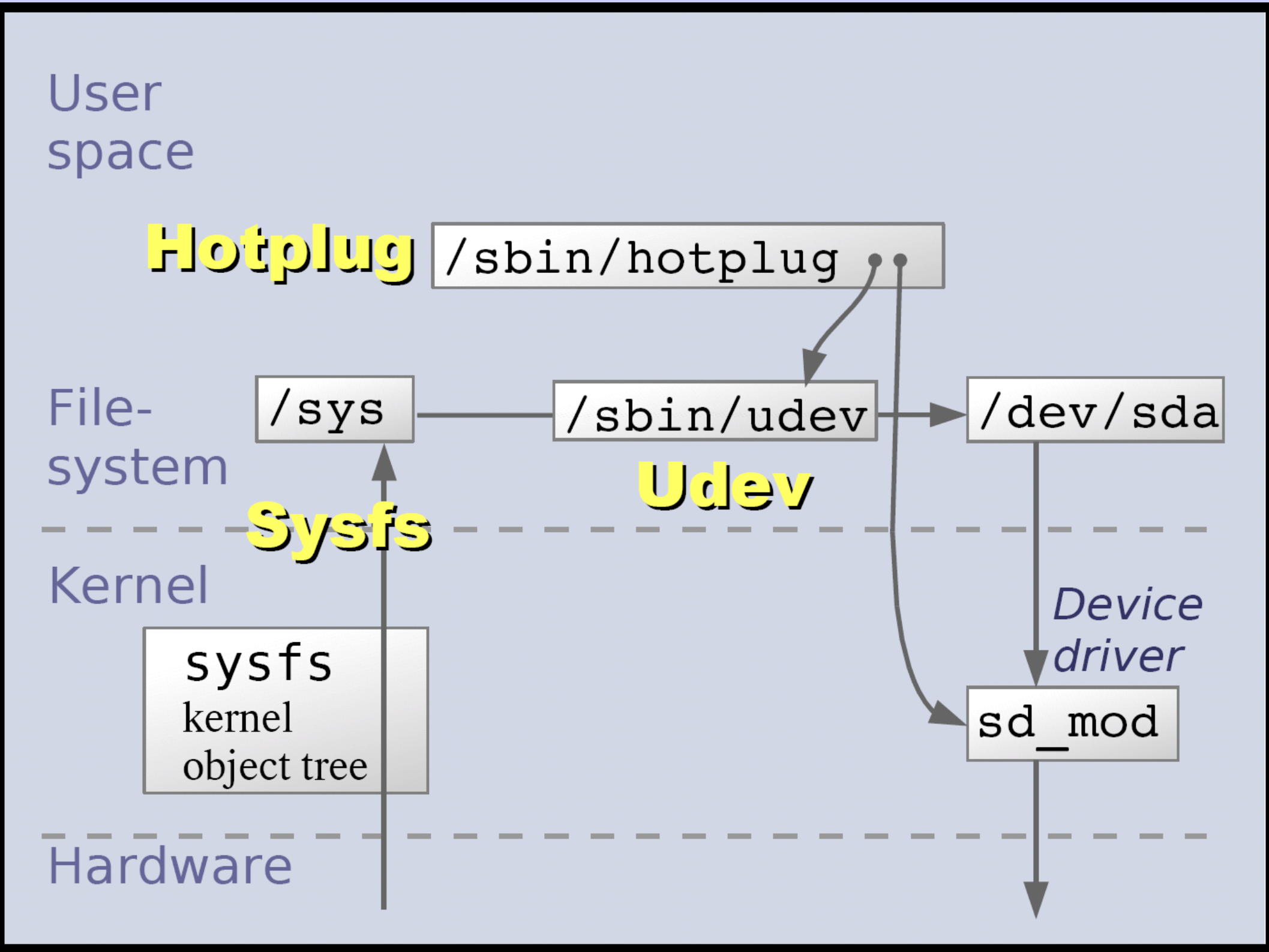
Where we'd like to be . . .

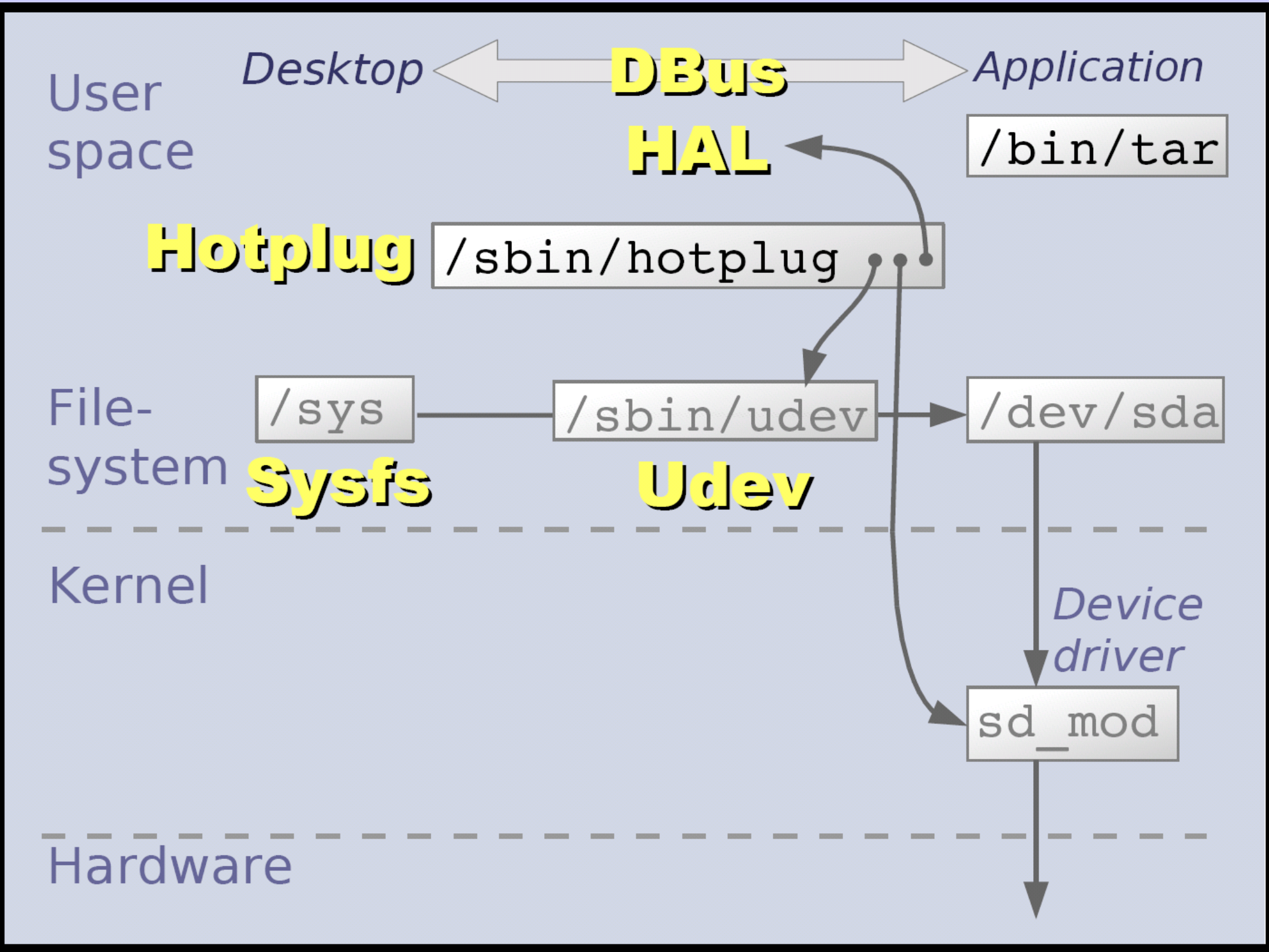



APPLICATIONS
HOTPLUG
HAL
/etc/hotplug.d/default/20-hal.hotplug
-> /usr/libexec/hal.hotplug
/etc/hotplug.d/default/
10-udev.hotplug
FILESYSTEM
DESKTOP
/etc/hotplug/<subsys>.agent
KERNEL
/sbin/hotplug
/etc/hotplug.d/default/
default.hotplug
/etc/hotplug.d/default/
20-hal.hotplug
kernel devices:
block
char
net
/dev
/proc
/sys
sysfs
kernel
object tree
UDEV

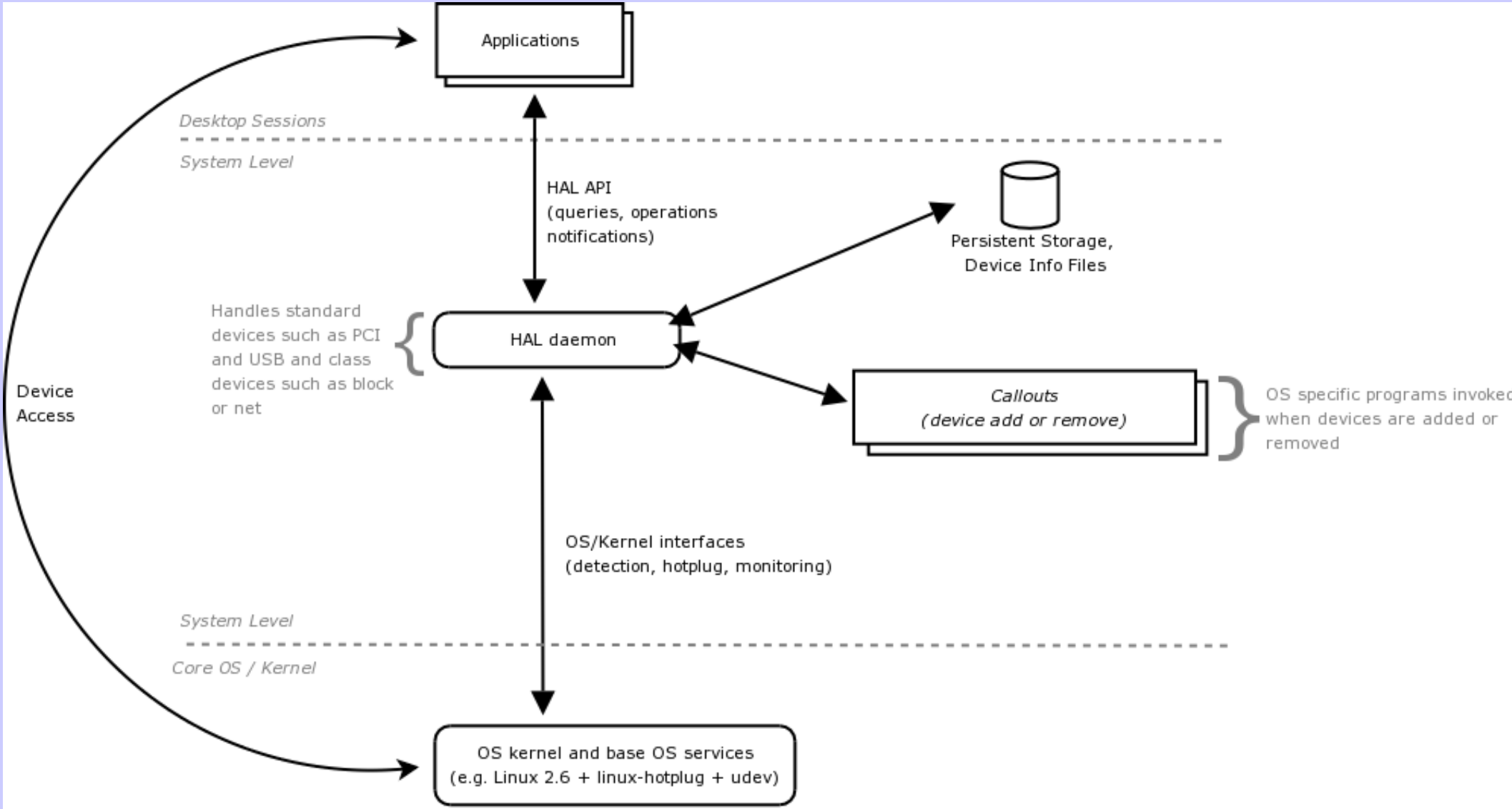

User space
Desktop
DBus
Application
HAL
/bin/tar
Hotplug
/sbin/hotplug
Device file
File-system
/sys
/sbin/udev
/dev/sda
Udev
Sysfs
Kernel
Device driver
sysfs
kernel
object tree
uhci_hcd
sd_mod
Hardware


User space
Hotplug
/sbin/hotplug
File-system
usb.agent
modprobe
Kernel
Device driver
uhci_hcd
sd_mod
Hardware


User space
Hotplug
/sbin/hotplug
File-system
/sys
/sbin/udev
/dev/sda
Udev
Sysfs
Kernel
Device driver
sysfs
kernel
object tree
sd_mod
Hardware


User space
Desktop
DBus
Application
HAL
/bin/tar
Hotplug
/sbin/hotplug
File-system
/sys
/sbin/udev
/dev/sda
Sysfs
Udev
Kernel
Device driver
sd_mod
Hardware

HALL AQUITECTURE



Udev:

Provides the ability to name devices in a persistent manner using a flexible rule-based system.

Udev Rules:

A udev rule defines the mapping between a device's physical attributes and the desired device filename.

Udev Rules:

Found in files under

`/etc/udev/rules.d`

Describe any devices to be named in a way that differs from the default kernel name.

Sample Udev Rules

```
BUS="usb", \
SYSFS_serial="W09090207101241330", \
NAME="lp_color"

BUS="usb", \
SYSFS_serial="HXOLL0012202323480", \
NAME="lp_plain"
```

Sample Udev Rules

```
BUS="usb", SYSFS_serial="W09090207101241330", \
NAME="lp_color"

BUS="usb", SYSFS_serial="HXOLL0012202323480", \
NAME="lp_plain"
```

Map to:

```
mknod lp_color c 180 1
mknod lp_plain c 180 0
```

Keys available in udev rules:

BUS matches the bus type of the device; examples of this include PCI, USB or SCSI.

KERNEL matches the "kernel name" of the device.

ID matches the device number on the bus; for example, the PCI bus ID or the USB device ID.

Keys available in udev rules:

PLACE matches the topological position on bus, such as the physical port a USB device is plugged in to.

SYSFS_filename, SYSFS{filename} match any sysfs device attribute, such as label, vendor, USB serial number or SCSI UUID.

Keys available in udev rules:

PROGRAM

allows udev to call an external program and check the result. This key is valid if the program returns successfully.

RESULT

matches the returned string of the last PROGRAM call. This key may be used in any rule following a PROGRAM call.

Advanced udev rules

udev allows a number of printf-like **string substitutions** to be used in the NAME, SYMLINK and PROGRAM fields.

Also, a number of keys support a simple form of **shell-style pattern matching**.

Sample udev rule

```
$ cat /etc/udev/rules.d/90-sg.rules

# sg[0-9] scsi scanner
KERNEL="sg[0-9]*", BUS="scsi", \
SYMLINK="scanner"

$ ls -l /dev/scanner
lrwxrwxrwx 8 Feb 15 08:26 scanner -> sg0
```

Silly udev rule

```
KERNEL="[hs]d[a-z]", \
PROGRAM="name_cdrom.pl %M %m", \
NAME="%1c", SYMLINK="cdrom"
```

- On match of any disk device
- Run the Perl program **name_cdrom.pl**
- Use the first word of the program's output to name the device file
- Create a symlink called cdrom.

Silly udev rule

/dev might look like the following when using this rule:

```
$ cd /dev
$ ls -l cdrom
lrwxrwxrwx       8 Feb 15 08:26 cdrom -> Samiam-Astray
$ ls -l Samiam-Astray
brw-------  22, 64 Feb 15 08:26 Samiam-Astray
```

Hotplug

Udev

Where & how to get info to write udev rules?

Sysfs

HAL & DBus

Where? From SysFS

```
$ udevinfo -q path -n /dev/sda
/block/sda

$ ls /sys/block/sda
dev      queue/  removable  size
device   range   sda1/        stat

$ udevinfo -a -p /block/sda
```

Getting info from SysFS:

$ udevinfo -a -p /block/sda

udevinfo starts with the device the node belongs to and then walks up the device chain, to print for every device found, all possibly useful attributes in the udev key format.

```
looking at class device '/sys/block/sda':
    SYSFS{dev}="8:0"
    SYSFS{range}="16"
    SYSFS{removable}="1"
    SYSFS{size}="64000"
    SYSFS{stat}="       40        36
377      1224           0          0          0
    0          0      1224       1224"

...
```

Getting info from SysFS:

```
... looking at the device chain at
'/sys/devices/pci0000:00/0000:00:11.3/usb2/2-2':
    BUS="usb"
    ID="2-2"
    SYSFS{bConfigurationValue}="1"
    SYSFS{bDeviceClass}="00"
    SYSFS{bDeviceProtocol}="00"
    SYSFS{bDeviceSubClass}="00"
    SYSFS{bMaxPower}="100mA"
    SYSFS{bNumConfigurations}="1"
    SYSFS{bNumInterfaces}=" 1"
    SYSFS{bcdDevice}="0100"
    SYSFS{bmAttributes}="80"
    SYSFS{detach_state}="0"
    SYSFS{devnum}="2"
    SYSFS{idProduct}="0100"
    SYSFS{idVendor}="0d7d"
    SYSFS{manufacturer}="        "
    SYSFS{maxchild}="0"
    SYSFS{product}="USB DISK"
    SYSFS{serial}="073A1D252211"
    SYSFS{speed}="12"
    SYSFS{version}=" 1.10"
```

A new udev rule

```
# USB pen drive
KERNEL="sd[a-z]", BUS="scsi", \
SYSFS{serial}="073A1D252211" \
NAME="usb_pen", LINK="my_pen"
```

Does it work?

A new udev rule:

```
... looking at the device chain at
'/sys/devices/pci0000:00/0000:00:11.3/usb2/2-2':
    BUS="usb"
    ID="2-2"
    SYSFS{bConfigurationValue}=
    SYSFS{bDeviceClass}="00"
    SYSFS{bDeviceProtocol}="00"
    SYSFS{bDeviceSubClass}="00"
    SYSFS{bMaxPower}="100mA"
    SYSFS{bNumConfigurations}="
    SYSFS{bNumInterfaces}=" 1"
    SYSFS{bcdDevice}="0100"
    SYSFS{bmAttributes}="80"
    SYSFS{detach_state}="0"
    SYSFS{devnum}="2"
    SYSFS{idProduct}="0100"
    SYSFS{idVendor}="0d7d"
    SYSFS{manufacturer}="          "
    SYSFS{maxchild}="0"
    SYSFS{product}="USB DISK"
    SYSFS{serial}="073A1D252211"
    SYSFS{speed}="12"
    SYSFS{version}=" 1.10"
```

```
KERNEL="sd[a-z]",
BUS="scsi",
SYSFS{serial}="073A1D252211"
NAME="usb_pen",
LINK="my_pen"
```

A new udev rule:

```
... looking at the device chain at
'/sys/devices/pci0000:00/0000:00:11.3/usb2/2-2':
    BUS="usb"
    ID="2-2"
    SYSFS{bConfigurationValue}=
    SYSFS{bDeviceClass}="00"
    SYSFS{bDeviceProtocol}="00"
    SYSFS{bDeviceSubClass}="00"
    SYSFS{bMaxPower}="100mA"
    SYSFS{bNumConfigurations}="
    SYSFS{bNumInterfaces}=" 1"
    SYSFS{bcdDevice}="0100"
    SYSFS{bmAttributes}="80"
    SYSFS{detach_state}="0"
    SYSFS{devnum}="2"
    SYSFS{idProduct}="0100"
    SYSFS{idVendor}="0d7d"
    SYSFS{manufacturer}="          "
    SYSFS{maxchild}="0"
    SYSFS{product}="USB DISK"
    SYSFS{serial}="073A1D252211"
    SYSFS{speed}="12"
    SYSFS{version}=" 1.10"
```

```
KERNEL="sd[a-z]",
BUS="usb",
SYSFS{serial}="073A1D252211"
NAME="usb_pen",
LINK="my_pen"
```

The fields in a udev rule must all match in the same sysfs directory

Comparing: /dev, defs and udev

Install & Configure “by Hand”

- ☐ Hardware installed ☐ working
- ☐ Driver code installed ☐ working
- ☐ Other kernel code installed ☐ working
- ☐ Device file installed ☐ working

Install & Configure “by Hand”

☑ Hardware installed ☑ working

☐ Driver code installed ☐ working

```
modprobe foo_dev
```

Check kernel messages:

```
dmesg
tail -f /var/log/messages
```

Install & Configure “by Hand”

- ☑ Hardware installed ☑ working
- ☑ Driver code installed ☑ working
- ☐ Other kernel code installed ☐ working

```
modprobe fat
```

Check kernel messages:

```
dmesg
tail -f /var/log/messages
```

Install & Configure “by Hand”

- ☑ Hardware installed ☑ working
- ☑ Driver code installed ☑ working
- ☑ Other kernel code installed ☑ working
- ☑ Device file installed ☑ working

Ready to rock!

THE PROBLEMS OF STATIC /dev

P1 - A static /dev is unwieldy and big. It would be nice to only show the /dev entries for the devices we actually have running in the system.

P2 - We are (well, were) running out of major and minor numbers for devices.

P3 - Userspace programs want to know when devices are created or removed, and what /dev entry is associated with them.

THE PROBLEMS OF STATIC /dev

P4 - Users want a way to name devices in a persistent fashion (i.e. "This disk here, must _always_ be called "boot_disk" no matter where in the scsi tree I put it", or "This USB camera must always be called "camera" no matter if I have other USB scsi devices plugged in or not.")

CONSTRAINS TO CONSIDER

C1 - No policy in the kernel!

C2 - Follow standards (like the LSB)

C3 - Must be small so embedded devices will use it.

DEVFS SOLVING THE THE PROBLEMS

P1 - It would be nice to only show the /dev entries for the devices we actually have running in the system.

devfs only shows the dev entries for the devices in the system.

P2 - We are running out of major and minor numbers for devices.

devfs does not handle the need for dynamic major/minor numbers

DEVFS SOLVING THE PROBLEMS

P3 - Users want a way to name devices in a persistent fashion.

devfs does not provide a way to name devices in a persistent fashion.

P4 - Userspace programs want to know when devices are created or removed,and what /dev entry is associated with them.

devfs does provide a deamon that userspace programs can hook into to listen to see what devices are being created or removed.

DEVFS AND THE CONSTRAINSTS

C1 - No policy in the kernel!

devfs forces the devfs naming policy into the kernel.

C2 - Follow standards (like the LSB).

devfs does not follow the LSB device naming standard.

DEVFS AND THE CONSTRAINSTS

C3 - Must be small so embedded devices will use it.

devfs is small, and embedded devices use it. However it is implemented in non-pagable memory.

UDEV SOLVING THE PROBLEMS

P1 - It would be nice to only show the /dev entries for the devices we actually have running in the system.

using udev, the /dev tree only is populated for the devices that are currently present in the system.

UDEV SOLVING THE PROBLEMS

P2 - We are running out of major and minor numbers for devices.

udev does not care about the major/minor number schemes. If the kernel tomorrow switches to randomly assign major and minor numbers to different devices, it would work just fine

UDEV SOLVING THE PROBLEMS

P3 - Users want a way to name devices in a persistent fashion.

This is the main reason udev is around. It provides the ability to name devices in a persistent manner

UDEV SOLVING THE PROBLEMS

P4 - Userspace programs want to know when devices are created or removed,and what /dev entry is associated with them.

udev emits D-BUS messages so that any other userspace program (like HAL) can listen to see what devices are created or removed.

It also allows userspace programs to query its database to see what devices are present and what they are currently named as.

udev and the constrainsts

C1 - No policy in the kernel!

udev moves _all_ naming policies out of the kernel and into userspace.

udev and the constrainsts

C2 - Follow standards (like the LSB).

udev defaults to using the LSB device naming standard. If users want to deviate away from this standard (for example when naming some devices in a persistent manner), it is easily possible to do so.

udev and the constrainsts

P3 - Users want a way to name devices in a persistent fashion.

udev is small (49Kb binary) and is entirely in userspace, which is swapable, and doesn't have to be running at all times.

How to “Just make it work”?

☑ Hardware installed ☑ working

☑ Driver code installed
☑ Other kernel code installed

Hotplug 2.4

☑ Device file installed **Udev** 2.6

☑ Desktop & app support **HAL & DBus** >2.6?

Kernel changes in 2.6:

Expanded device numbers: **Sysfs**

- **Major: 4,095**
- **Minor: >1M**

udev a user-space implementation of devfs

- Instead of a /dev created once, statically, udev updates /dev on the fly, in response to the exact hardware available to the system.
- More important, however, is that udev places intimate knowledge of devices and their device nodes in user space.
- Hotplug, sysfs and udev together allow user space a complete view of the system's hardware.

Some links

- Udev HomePage
  http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev.html
- udev faq
  http://www.kernel.org/pub/linux/utils/kernel/hotplug/udev-FAQ
- udev writing rules
  http://reactivated.net/writing_udev_rules.html
- Page with good reference links about udev and friends
  http://developer.osdl.org/maryedie/DCL/PSDN/Testing_udev_notes.html

Finally

Thank you ... for your patient.