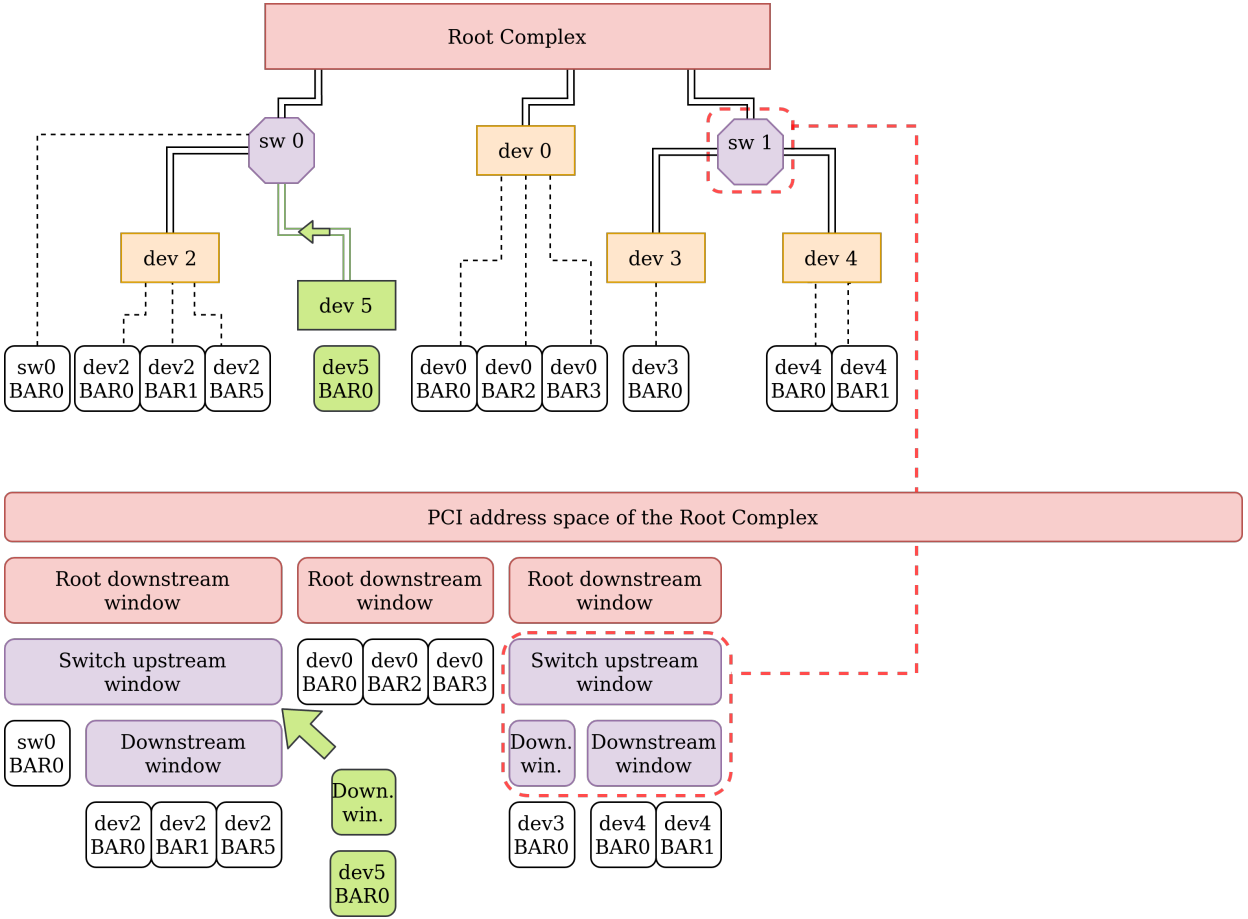
PCI hotplug: movable BARs and bus numbers

Linux Plumbers Conference 2020 – VFIO/IOMMU/PCI MC

Sergei Miroshnichenko <s.miroshnichenko@yadro.com>

August 24, 2020

Part I: Topology, windows and BARs



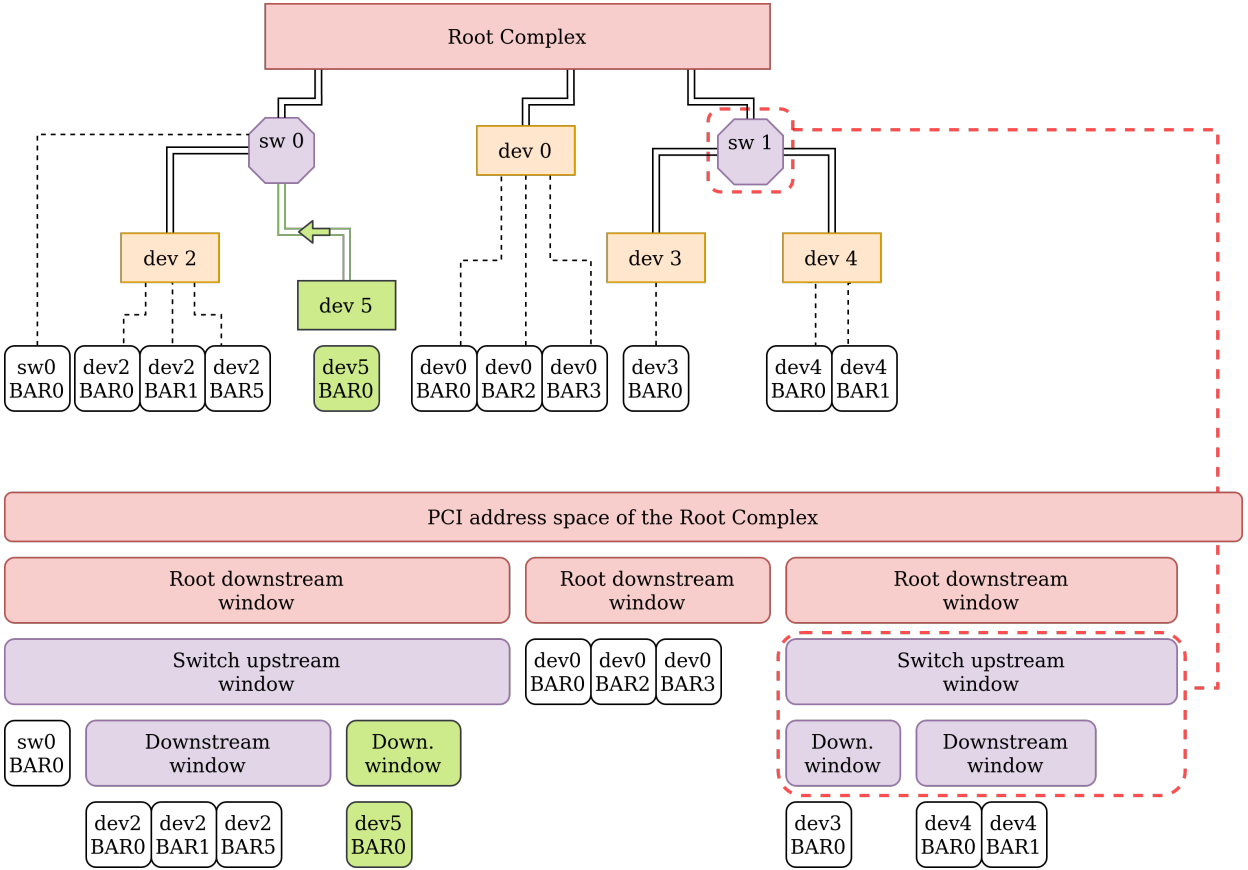
Constraints:

- Device's BAR may reside only in a window of its downstream port;
- Every switch port (upstream and downstreams) has 3 windows – one for each BAR type: IO, MEM and PREFETCH;
- Neighbouring windows can not intersect.
- Window of an upstream port covers all windows of downstream ports, and also switch's own BARs if any;
- BARs and windows in a parent window may be shuffled in arbitrary order.

The assignment algorithm:

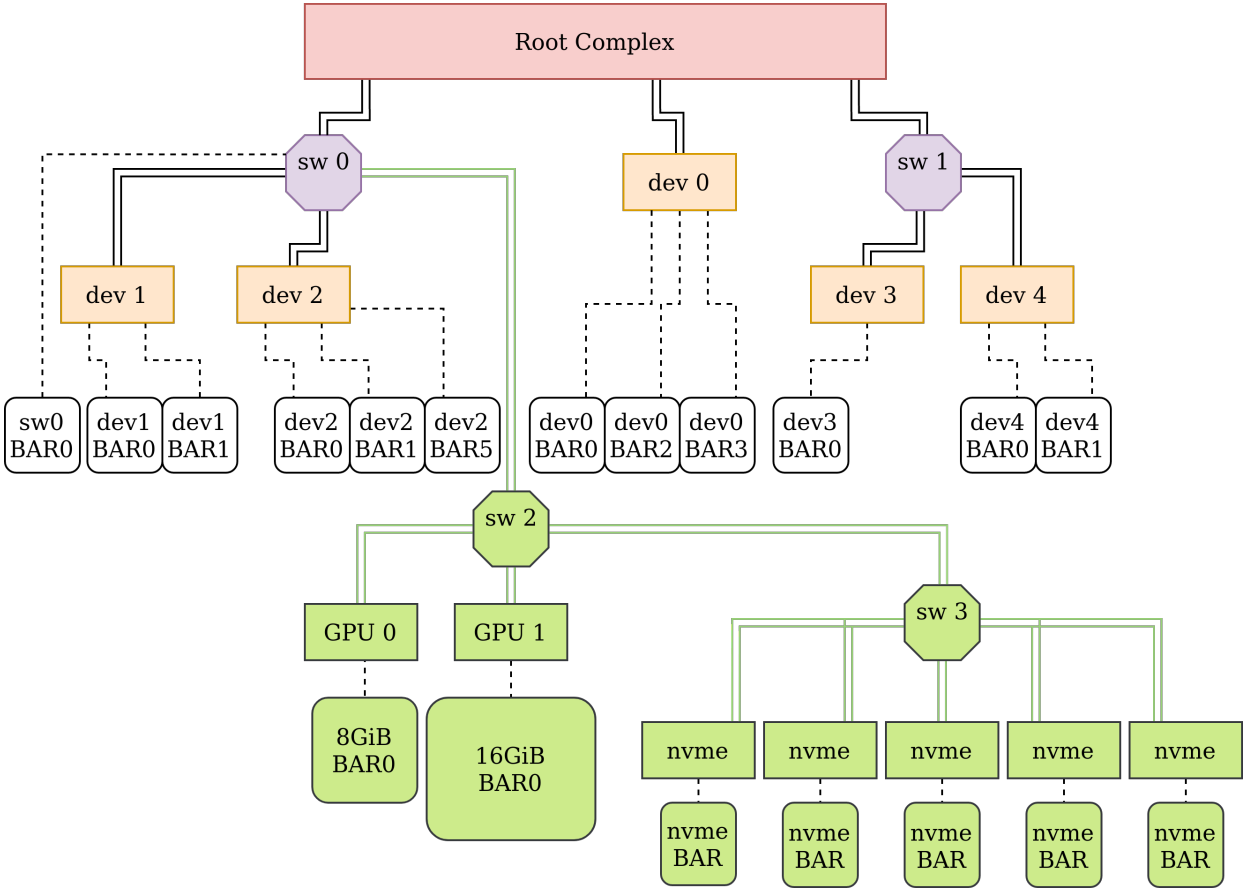
- Calculate window sizes bottom-up;
- Assign window addresses from top to bottom;
- Assign BARs to their windows.

Reserving gaps in address space



- If requested larger windows, new BARs can fit in the reserved gaps in the address space;
- Usually a bootloader/BIOS/UEFI provides an enumerated PCI topology, and the kernel may accept it;
- But the kernel is able to assign BARs by its own means.

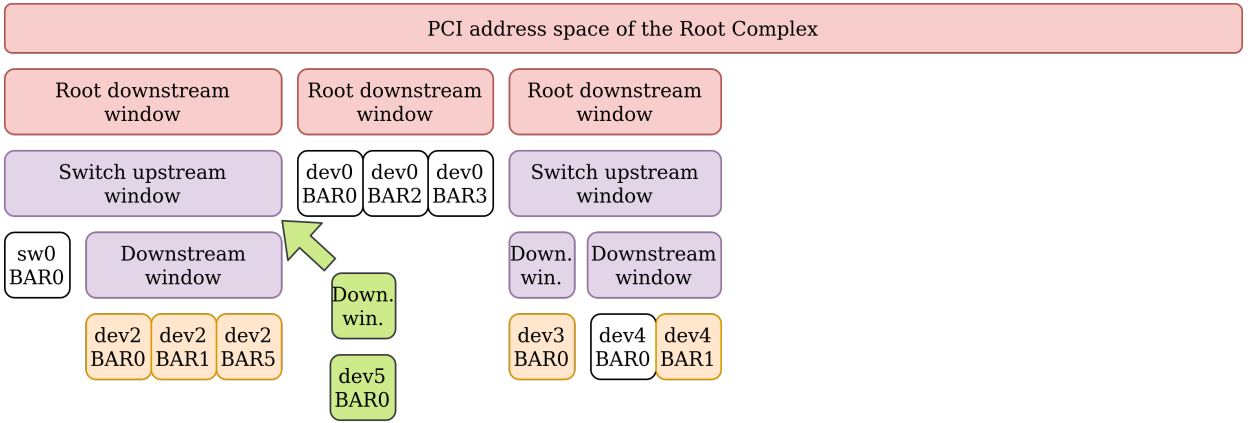
Hot-adding a chassis (switch)



Reserved gaps mostly works of endpoint devices, but adding even a single GPU may be a problem because of its large BARs.

Attaching a chassis (which starts with a switch) multiplies this problem.

Fixed BARs




```
bool pci_dev_bar_fixed(
  struct pci_dev *dev,
  struct resource *res)
{
  /* Bridge windows are never fixed */
  if (resno >= PCI_BRIDGE_RESOURCES)
    return false;

  if (res->flags & IORESOURCE_PCI_FIXED)
    return true;

  if (!pci_can_move_bars)
    return false;

  if (dev->driver &&
      dev->driver->bar_fixed)
    return dev->driver->bar_fixed(dev,
      resno);

  if (!dev->driver && !res->child)
    return false;

  return true;
}
```

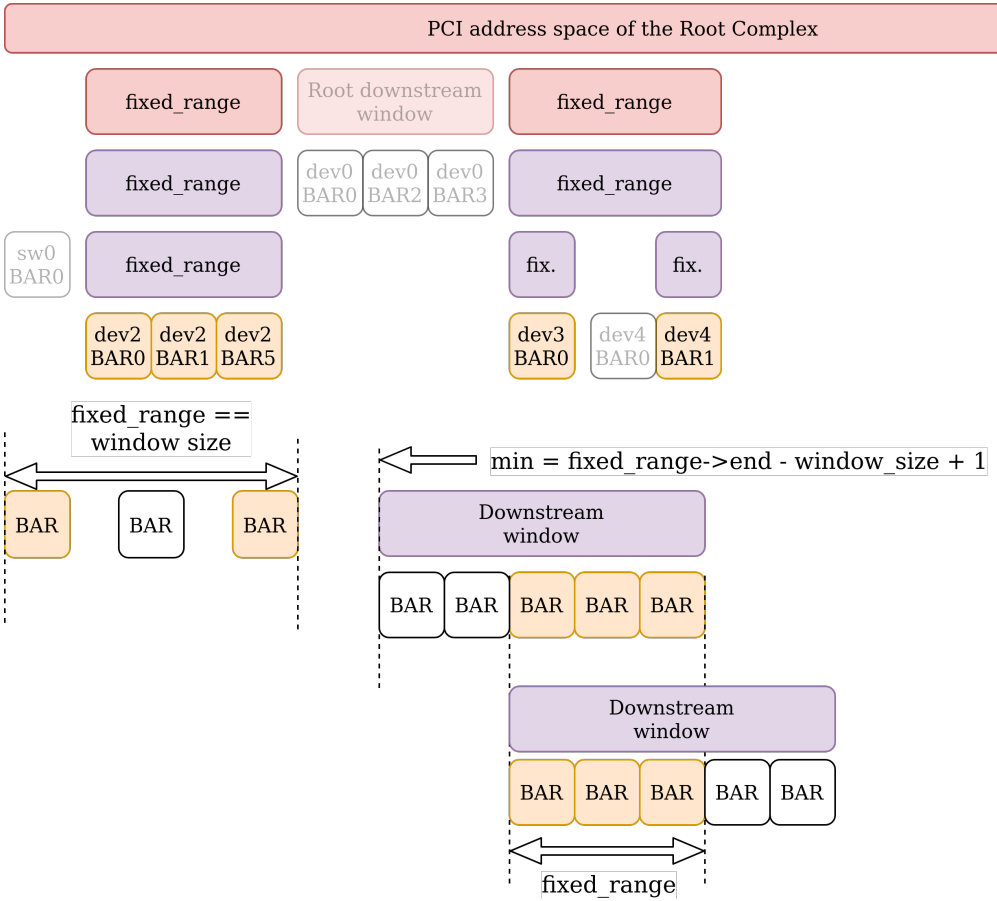
Effects of these changes

Before, a hotplug event on a slot was local to its direct switch only, just utilizing the reserved gaps between the used BARs, without altering the switch's registers.

But when BARs are movable, hotplug event in the middle of the PCIe tree affects almost every bridge window of almost every switch in the whole topology.

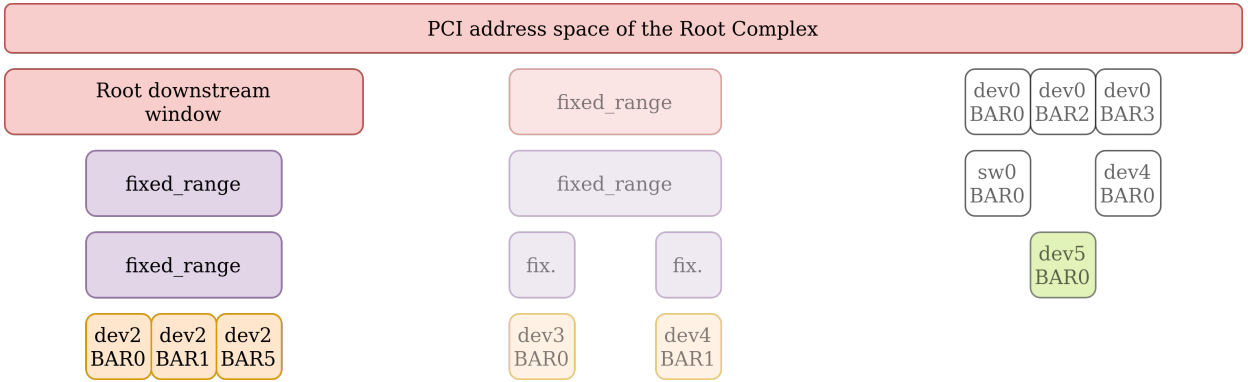
An interrupt from pciehp (standard hotplug driver) will lead to a full domain rescan.

Calculating bridge windows



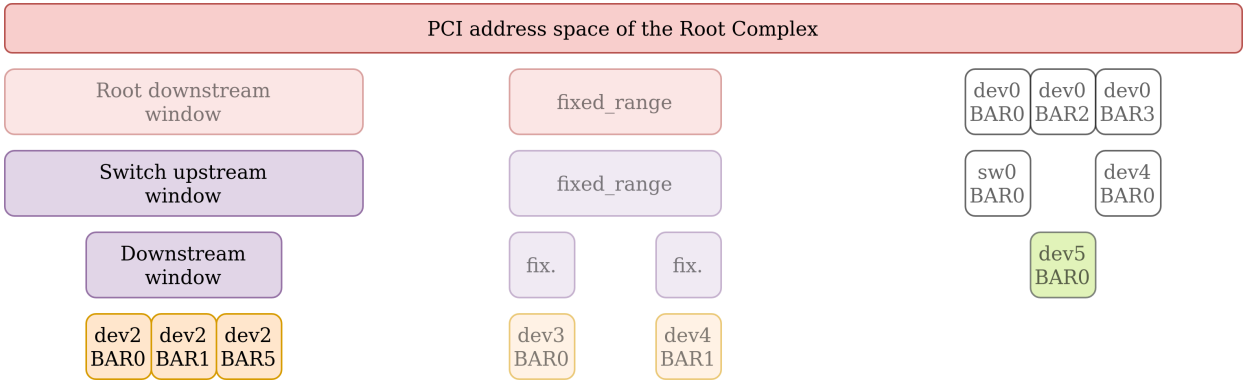
- Release movable BARs;
- Release windows;
- **Don't touch hardware registers yet!**
- Fill the new fixed_range field for every struct pci_bus;
- fixed_range is propagated to parent windows;
- Recalculate window sizes taking fixed_range into account;
- Minimal start address is needed to make the window cover its fixed BARs: pci_bus_alloc_resource(bus, res, size, align, **min**, 0, pcibios_align_resource, dev);
- As windows are assigned now based on the min start address, they must be sorted beforehand.

Assigning windows example 1/5



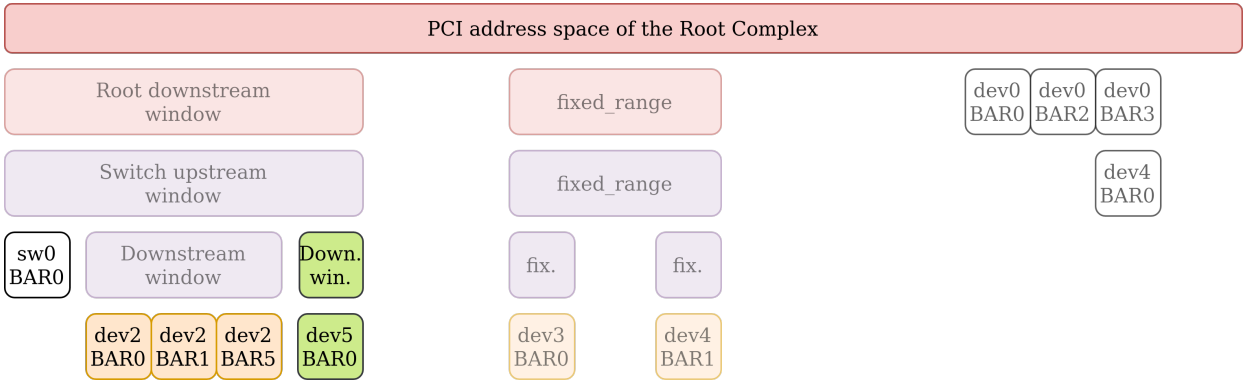
- A window set with the leftmost fixed BAR is assigned first, starting from the downstream root port;

Assigning windows example 2/5



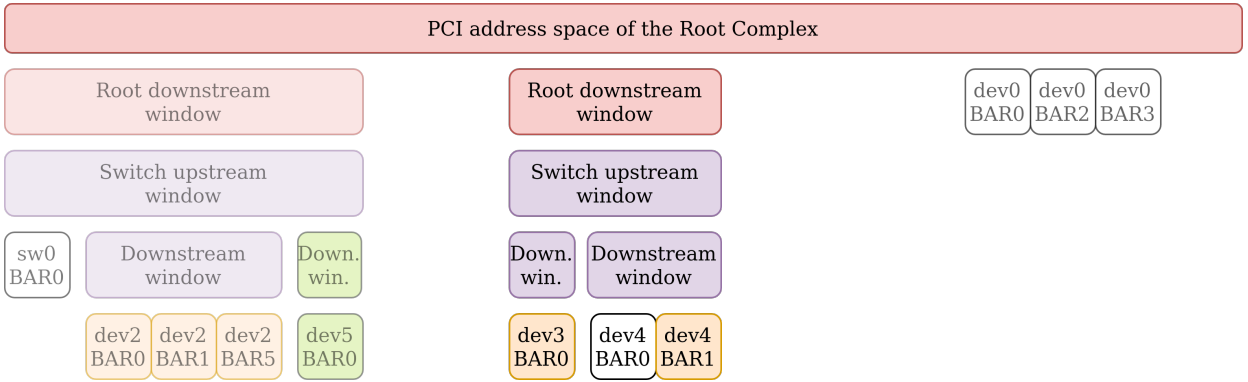
- A window set with the leftmost fixed BAR is assigned first, starting from the downstream root port;
- Next are windows below, leading to fixed BARs;

Assigning windows example 3/5



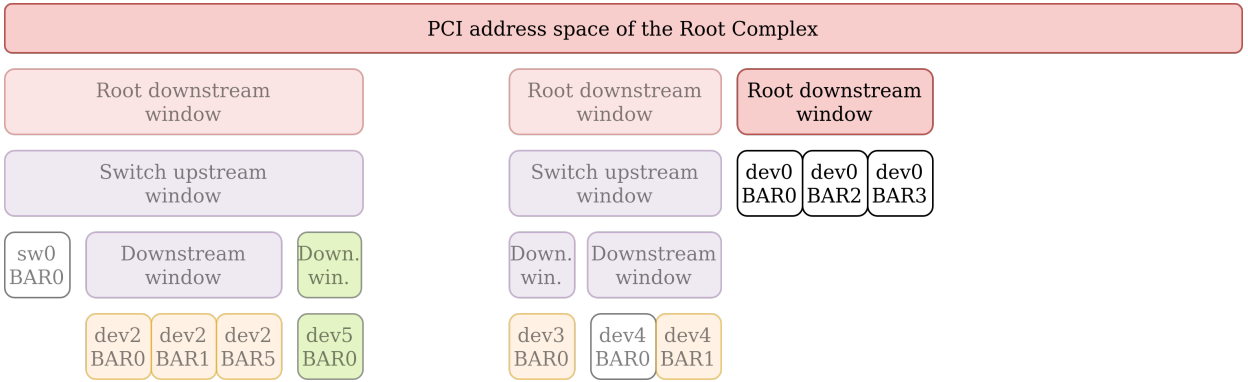
- A window set with the leftmost fixed BAR is assigned first, starting from the downstream root port;
- Next are windows below, leading to fixed BARs;
- Then movable and new BARs;

Assigning windows example 4/5



- A window set with the leftmost fixed BAR is assigned first, starting from the downstream root port;
- Next are windows below, leading to fixed BARs;
- Then movable and new BARs;
- The second set must start at the beginning of its first fixed BAR;

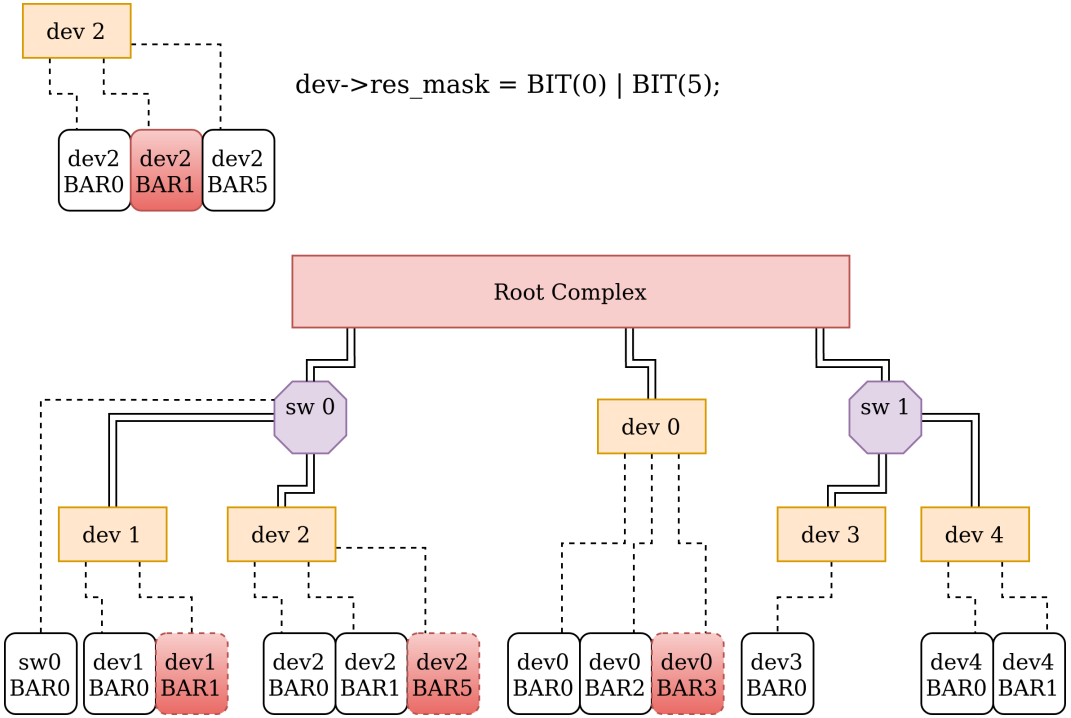
Assigning windows example 5/5



- A window set with the leftmost fixed BAR is assigned first, starting from the downstream root port;
- Next are windows below, leading to fixed BARs;
- Then movable and new BARs;
- The second set must start at the beginning of its first fixed BAR;
- And the rest – the movable ones – are assigned last.

Now it's time to write new BARs and windows to PCI registers

Fallback



Reassignment may fail: blocked by fixed BARs, or not enough address space. A working layout must be restored.

Need to track which BARs were working before a PCI rescan with the new res_mask bitmask in struct pci_dev;

Some BAR can remain unassigned since previous attempt (e.g., not provided by BIOS), need to retry them.

Try different policies of BAR assignments:

- try every BAR, even those that weren't assigned before;
- if that fails, retry without those failed BARs;
- if that fails, retry without one of hotplugged devices.

Support of movable BARs in drivers

NVMe need to pause before a rescan, and remap its BAR0 after:

```
static bool nvme_bar_fixed(struct pci_dev *pdev, int resno)
{
        return false;
}

static void nvme_rescan_prepare(struct pci_dev *pdev)
{
        struct nvme_dev *dev = pci_get_drvdata(pdev);
        nvme_dev_disable(dev, true);
        nvme_dev_unmap(dev);
        dev->bar = NULL;
}

static void nvme_rescan_done(struct pci_dev *pdev)
{
        struct nvme_dev *dev = pci_get_drvdata(pdev);
        nvme_dev_map(dev);
        nvme_reset_ctrl(&dev->ctrl);
}

static struct pci_driver nvme_driver = {
        .bar_fixed      = nvme_bar_fixed,
        .rescan_prepare = nvme_rescan_prepare,
        .rescan_done    = nvme_rescan_done,
};
```

Some switches have BARs, but portdrv doesn't use them:

```
static bool pcie_portdrv_bar_fixed(struct pci_dev *pdev, int resno)
{
        return false;
}

static struct pci_driver pcie_portdriver = {

        .bar_fixed = pcie_portdrv_bar_fixed,
}
```

lockdep

The enable_mutex is added to struct pci_dev to help with the race in pci_enable_bridge() + pci_is_enabled(). But the kernel test robot reported a possible recursive locking, because the bridge also enables its parent.

But if replace by mutex_lock_nested(&dev→enable_mutex, X), the max supported value of a subclass argument is (MAX_LOCKDEP_SUBCLASSES - 1) = 7, so BDF address is not suitable – bus number range is {0 – 255}.

Probably using depth, like in i2c, may help with that, if the current PCI topology is not very deep.

Other issues addressed

Few things had to be solved before getting to BARs:

- Race in pci_enable_bridge() + pci_is_enabled(): reproduces when drivers are starting simultaneously for devices in a non-pre-enabled bridge – fixed by a per-device mutex;
- PCIe-specific settings (Max Payload Size) weren't not applied to hot-added bridges;
- The IO and MEM bits of bridges weren't re-checked after hotplug events;
- Some BIOSes report PCIBIOS_MIN_MEM in the middle of the PCI address space.

TODO: Resizable BARs

Work in progress on integrating the support of Resizable BARs.

BIOS sets only 256MiB size for GPU's BAR for compatibility, but AMD GPUs use this feature to map the entire VRAM.

Part II: Hot-adding a large switch

Any bus number assignment policy can be broken by hotplugging nested bridges:

```
+-[0020:00]---00.0-[01-20]--+-00.0-[02-20]--+-00.0-[03-07]--    <--  *BRIDGE*
|                           |               +-...
|                           |               \-05.0-[1c-20]--



04.0-[xx-xx]--+-00.0-[xx-xx]--
              +-04.0-[xx-xx]--
              +-09.0-[xx-xx]--
              +-0c.0-[xx-xx]----00.0-[xx-x]--+-01.0-[xx-xx]--
              |                              +-02.0-[xx-xx]--
              |                              +-04.0-[xx-xx]--
              |                              +-05.0-[xx-xx]--
              |                              +-06.0-[xx-xx]--
              |                              +-07.0-[xx-xx]--
              |                              +-08.0-[xx-xx]--
              |                              +-09.0-[xx-xx]--
              |                              +-0a.0-[xx-xx]--
              |                              +-0b.0-[xx-xx]--
              |                              +-0c.0-[xx-xx]--
              |                              +-0d.0-[xx-xx]--
              |                              +-0e.0-[xx]----00.0 NVMe
              |                              +-0f.0-[xx-xx]--
              |                              +-10.0-[xx-xx]--
              |                              \-11.0-[xx-xx]--
              +-10.0-[xx-xx]--
              +-11.0-[xx-xx]--
```

Movable bus numbers: Concept

Instead of reserving numbers, let's "push" the "tail" of the tree to free numbers for the new switch (bus 04→1e, 05→1f, etc.):

```
+-[0020:00]---00.0-[01-08]--+-00.0-[02-08]--+-00.0-[03]--    <--  *BRIDGE*
|                           |               +-01.0-[04]--
|                           |               +-02.0-[05]--
|                           |               +-03.0-[06]--
|                           |               +-04.0-[07]--
|                           |               \-05.0-[08]--
```

```
+-[0020:00]---00.0-[01-22]--+-00.0-[02-22]--+-00.0-[03-1d]----04.0-[04-1d]--+-00.0-[05]--
|                           |               |                               +-04.0-[06]--
|                           |               |                               +-09.0-[07]--
|                           |               |                               +-0c.0-[08-19]----00.0-[09-19]--+-01.0-[0a]--
|                           |               |                               |                               +-02.0-[0b]--
|                           |               |                               |                               +-04.0-[0c]--
|                           |               |                               |                               +-05.0-[0d]--
|                           |               |                               |                               +-06.0-[0e]--
|                           |               |                               |                               +-07.0-[0f]--
|                           |               |                               |                               +-08.0-[10]--
|                           |               |                               |                               +-09.0-[11]--
|                           |               |                               |                               +-0a.0-[12]--
|                           |               |                               |                               +-0b.0-[13]--
|                           |               |                               |                               +-0c.0-[14]--
|                           |               |                               |                               +-0d.0-[15]--
|                           |               |                               |                               +-0e.0-[16]----00.0 NVMe
|                           |               |                               |                               +-0f.0-[17]--
|                           |               |                               |                               +-10.0-[18]--
|                           |               |                               |                               \-11.0-[19]--
|                           |               |                               ...
|                           |               |                               \-15.0-[1d]--
|                           |               +-01.0-[1e]--
|                           |               +-02.0-[1f]--
|                           |               +-03.0-[20]--
|                           |               +-04.0-[21]--
|                           |               \-05.0-[22]--
```

Movable bus numbers: drivers

linux/include/linux/pci.h: BDFs are not used by the drivers explicitly, but via handlers:

```
int pci_read_config_byte(const struct pci_dev *dev, int where, u8 *val);
int pci_read_config_word(const struct pci_dev *dev, int where, u16 *val);
int pci_read_config_dword(const struct pci_dev *dev, int where, u32 *val);
int pci_write_config_byte(const struct pci_dev *dev, int where, u8 val);
int pci_write_config_word(const struct pci_dev *dev, int where, u16 val);
int pci_write_config_dword(const struct pci_dev *dev, int where, u32 val);
```

Probably still need to block the writes to registers except of the Subordinate Bus Number, Secondary Bus Number, BARx and Base+Limit.

Movable bus numbers, huge problem: sysfs and procfs

sysfs and procfs entries and symlinks of devices are all based on their BDFs, which may change after a PCI rescan

```
% ls -la /sys/bus/pci/devices
0000:00:00.0 -> ../../../devices/pci0000:00/0000:00:00.0
0000:00:02.0 -> ../../../devices/pci0000:00/0000:00:02.0
...
0000:04:00.0 -> ../../../devices/pci0000:00/0000:00:1c.6/0000:04:00.0
0000:40:00.0 -> ../../../devices/pci0000:00/0000:00:1d.2/0000:40:00.0
```

```
% ls -la /proc/bus/pci/*
/proc/bus/pci/00:
00.0
02.0
...
1f.4
1f.6

/proc/bus/pci/04:
00.0

/proc/bus/pci/40:
00.0
```

```
% ls -la /sys/devices/pci0000:00/
0000:00:00.0
0000:00:02.0
...
0000:00:1f.3
0000:00:1f.4
0000:00:1f.6

% ls -la /sys/devices/pci0000:00/0000:00:1c.6/0000:04:00.0/driver
driver -> ../../../../bus/pci/drivers/iwlwifi
```

Movable bus numbers: bus renaming

The only renaming function found in the kernel is device_rename(&dev→dev, new_name), it is used now only to rename network interfaces, and is marked with "Note: Don't call this function".

How should udev react on renamed devices? "remove"+"add" implies a completely new device, but that is not the case.

Sysfs entries are created by the kernel during bus_add_device(dev)+pci_create_sysfs_dev_files(dev), removed during bus_remove_device(dev)+pci_remove_sysfs_dev_files(dev)

Sysfs symlinks – device_add_class_symlinks(dev) and device_remove_class_symlinks(dev)

Procfs entries – pci_proc_attach_device(dev) and pci_proc_detach_device(dev)

Movable bus numbers: brutal way

Supposing that sysfs renaming is permissible, the following proof of concept has been constructed: destroy all sysfs and procfs entries and symlinks before renaming the device, and then recreate them back, based on new BDFs.

The bus_remove_device(dev) is too aggressive - it also detaches the device from its driver, which is preferred to avoid.

As an experiment, the bus_disconnect_device(dev) was added to the Base API, which is the same as original bus_remove_device(dev), but without the call to device_release_driver(dev).

That's not the only change in the Base kernel API required for this approach: the existing bus_add_device(dev), device_add_class_symlinks(dev) and device_remove_class_symlinks(dev) had to be made public.

As a result, it is used to hot-add chassis full of NVMe drives, SAS HBA and other devices. It was totally worth it! :)