LPC2025 LIVE UPDATE MC, DEC 13 2025 TOKYO

PCSC: Caching PCI Config Space Accesses for faster Live Updates

Evangelos Petrongonas

He/Him

Kernel/Hypervisor Engineer

AWS

epetron@amazon.de

Alexander Graf

He/Him

Principal Engineer

AWS

graf@amazon.de

Diary of Bad Ideas

Navigating Tokyo without a smartphone

Caching PCI Configuration Space Accesses

Motivation

Tokyo

Ignorance

PCSC

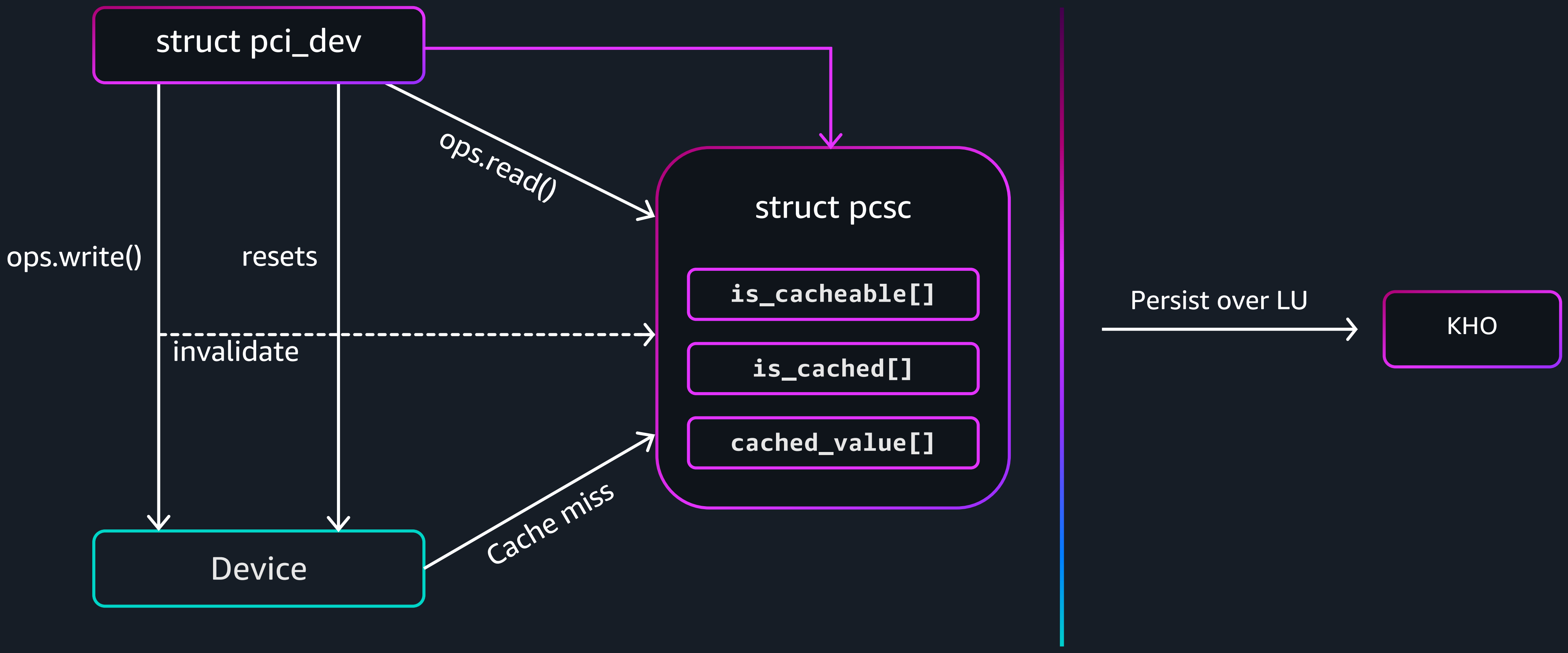
Ignorance

Bus Enumeration / VF creation

> 1K devices (PFs + VFs)

VMM actions

PCSC: How does it work ?



Is it a bad Idea if it works* ?

```
cat sys/bus/pci/pcsc/stats

Cache Hits: 21063
Cache Misses: 510
Uncachable Reads: 4398
Writes: 1049
Cache Invalidations: 584
Device Resets: 0
Total Reads: 25971
Hardware Reads: 4908
Hit Rate: 81%
Total Cache Access Time: 30952 us
Cache Access Time (without HW reads due to Misses): 16126 us
HW Access Time due to misses: 14826 us
Total Hardware Access Time: 101819 us
KHO Restore Statistics:
  Restored Devices: 2819
  Total Restore Time: 1362 us
  Hashtable Initial Entries: 2819
  Hashtable Build Time: 1000 us
```

Up to 450ms savings* in PCI Configuration Space Accesses

Current Status

RFC Posted: [RFC PATCH 00/13] Introduce PCI Configuration Space Cache (PCSC)

LKML

Discussion

Is this the right approach?

ops.map_bus()

How to handle it ?

LUO integration

[PATCH v2 00/10] LUO: PCI subsystem (phase I)

```
 static int pci_liveupdate_freeze(void
*arg, u64 *data)
```

THANK YOU!

PCSC: Caching PCI Config Space Accesses for faster Live Updates

Evangelos Petrongonas

He/Him

Kernel/Hypervisor Engineer

AWS

epetron@amazon.de

Alexander Graf

He/Him

Principal Engineer

AWS

graf@amazon.de

LKML

Backup Slides

map_bus

```c
/**
 * pcsc_map_bus - Map PCI configuration space for memory-mapped access
 * @bus: PCI bus structure
 * @devfn: Device and function number
 * @where: Offset in configuration space
 *
 * WARNING: Cache Bypass Issue
 * This function returns a memory-mapped I/O address that provides direct
 * access to PCI configuration space, completely bypassing the PCSC cache.
 *
 * Any reads or writes performed through the returned MMIO address will NOT:
 * - Use cached values for reads
 * - Update cached values on reads
 * - Invalidate cached values on writes
 *
 * This can lead to cache inconsistency where:
 * 1. PCSC cache contains stale data after MMIO writes
 * 2. Subsequent cached reads return outdated values
 * 3. Cache coherency is lost until the next cache invalidation
 *
 * Current users include:
 * - (pci_generic_config_{read,write}{,32}) which are already handled
 * - operations on RCs that are not supported by PCSC.
 * Therefore, there is no risk of cache inconsistency here.
 * However, any future use of map_bus after cache population poses risks.
 *
 * IMPORTANT: Callers using the returned MMIO address are responsible for
 * maintaining cache consistency. Consider invalidating relevant cache entries
 * after MMIO operations if the device's cache may be active.
 *
 * Return: Virtual address for memory-mapped config space access, or NULL
 */
static void __iomem *pcsc_map_bus(struct pci_bus *bus, unsigned int devfn,
                                  int where)
{
    if (!bus->orig_ops || !bus->orig_ops->map_bus)
        return NULL;
    return bus->orig_ops->map_bus(bus, devfn, where);
}
```