



Managing IRQ mapping and deferred probing for ACPI static table devices

L.Pieralisi



LINUX
PLUMBERS
CONFERENCE

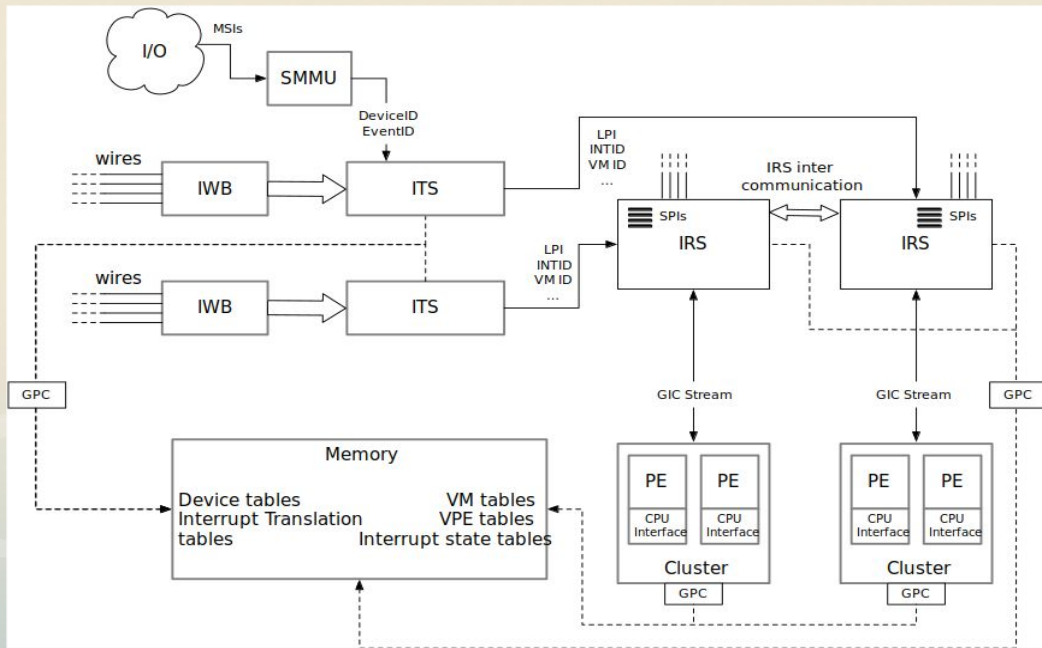
PRAGUE, CZECHIA / OCT. 5-7, 2026

A horizontal banner for the Linux Plumbers Conference. On the left, a penguin stands on a grey pipe. Above the penguin, the text "LINUX PLUMBERS CONFERENCE" is written in a bold, sans-serif font, followed by "PRAGUE, CZECHIA / OCT. 5-7, 2026" in a smaller font. On the right, a pressure gauge is mounted on the pipe. The background is a light blue gradient with faint, stylized clouds.

-
- The diagram illustrates the interrupt architecture. It shows the flow of interrupts from I/O devices through SMMU, IWB, ITS, and IRS blocks to GIC streams and finally to PE clusters. It also shows the interaction with Memory and GPCs.
- I/O** (cloud) sends **MSIs** to **SMMU**.
 - SMMU** outputs **DeviceID** and **EventID** to **ITS**.
 - IWB** (Input/Output Wire Block) receives **wires** and outputs to **ITS**.
 - ITS** (Interrupt Translation Service) outputs **LPI**, **INTID**, and **VMID** to **IRS**.
 - IRS** (Interrupt Remapping Service) contains **SPIs** (Service Processor Interrupts) and handles **IRS inter communication** with another **IRS**.
 - IRS** outputs **GIC Stream** to **Cluster** (PEs).
 - Cluster** contains **PE** (Processing Element) blocks with **CPU Interface** and **GPC** (Generic Interrupt Controller).
 - Memory** contains **Device tables**, **Interrupt Translation tables**, **VM tables**, **VPE tables**, and **Interrupt state tables**.
 - GPC** (Generic Interrupt Controller) interacts with **Memory** and **Cluster**.

GICv5 IWB: An ACPI device (2/2)

- Driver might probe later than code requiring the IRQ mapping to be resolved (`acpi_register_gsi()`)



LINUX

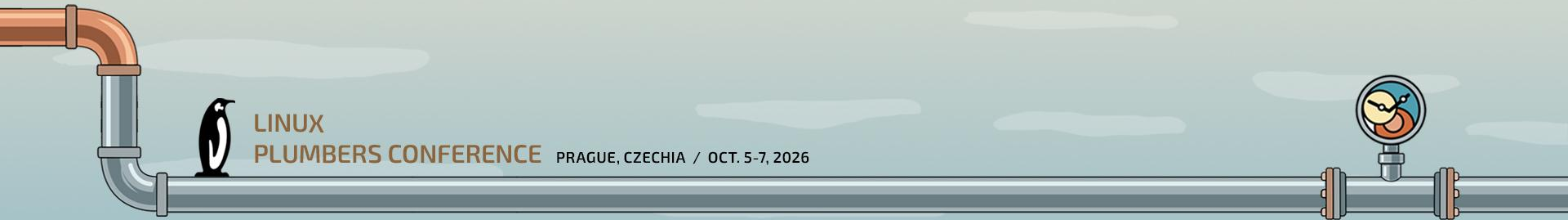
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



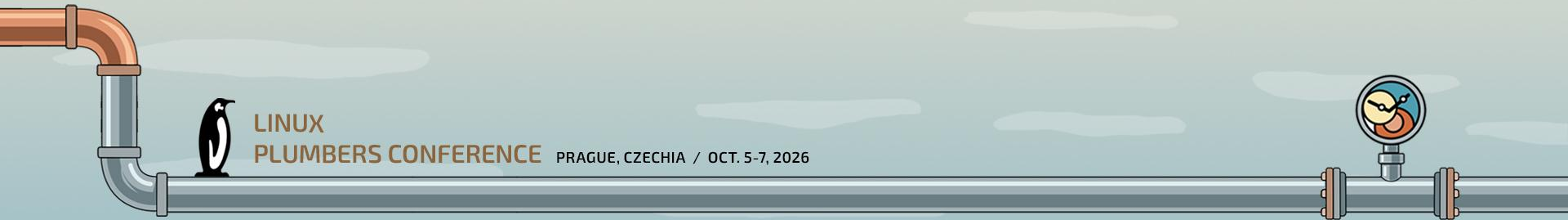
GICv5 IWB: IRQ consumers

- Any device drivers requiring a wired IRQ connected to an IWB (ie most of GICv3 SPIs)
- ACPI namespace devices
- Platform devices created out of static tables (SMMU, MPAM, arch MEM timer)



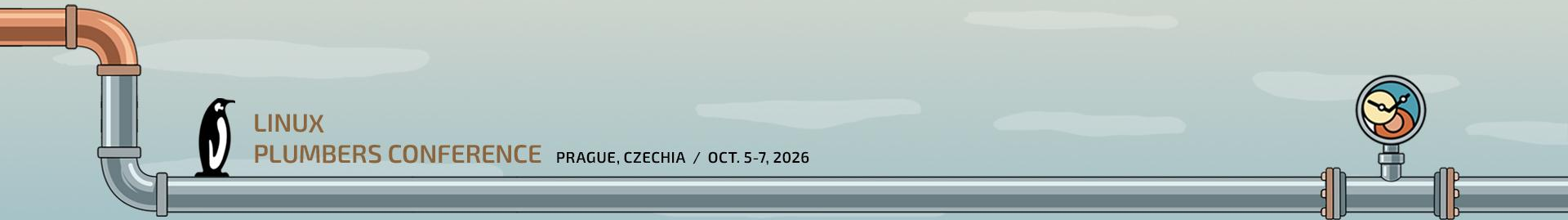
GICv5 IWB: ACPI namespace devices dep

- Resolved using ACPI autodep mechanism already implemented for RISC-V and moved to generic code
 - Handle probe dependency by automatically detecting acpi_handle dependency through the GSI number
 - It covers most of the devices (inclusive of PCI)
 - <https://lore.kernel.org/linux-acpi/CAJZ5v0hDK+C+ZJ4YKd0xbyHW+vEot9z-YrivWSrg1ag7Ebx4cA@mail.gmail.com/>
 - **Not applicable to platform devices created out of static tables, no ACPI object**



GICv5 IWB: Static table devices IRQ resolution (1/2)

- Platform devices for eg SMMU created when arch static tables (eg IORT) are parsed at `acpi_arch_init()` (subsys_initcall)
- IRQ mappings for those devices carried out through `acpi_register_gsi()` and stashed into the platform device resources (`IORESOURCE_IRQ`)
- The IWB driver might not have probed yet - which results in `acpi_register_gsi()` failures
- How to solve this is what we are here for today



GICv5 IWB: Static table devices IRQ resolution (2/2)

```
int acpi_register_gsi(struct device *dev, u32 gsi, int trigger, int polarity)
{
    struct irq_fwspec fwspec;
    unsigned int irq;

    fwspec.fwnode = acpi_get_gsi_domain_id(gsi);
    if (WARN_ON(!fwspec.fwnode)) {
        pr_warn("GSI: No registered irqchip, giving up\n");
        return -EINVAL;
    }

    fwspec.param[0] = gsi;
    fwspec.param[1] = acpi_dev_get_irq_type(trigger, polarity);
    fwspec.param_count = 2;

    irq = irq_create_fwspec_mapping(&fwspec);
    if (!irq)
        return -EINVAL;

    return irq;
}
```

Possible IRQ domain look-up failure



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



GICv5 IWB: Static table devices IRQ resolution - RFC

- Every platform device GSI number, flag and interrupt name are available in different ACPI static tables in different format
 - not a single chunk of data to look into to map the IRQ when the consumer driver requests it rather than at platform device creation time
- Consumer drivers (eg SMMUv3) request IRQs through standard platform calls (`platform_get_irq()`)
- So far, mapped IRQs were stashed in platform resource created when the platform device was added, that does not work anymore
- Where should we stash GSI number, flag and name so that platform generic code can map the IRQ when the consumer needs it (and possibly defer the probe) ?



GICv5 IWB: Static table devices IRQ probe defer - RFC

- We could use platform device resources to store HWIRQ instead of virtual IRQ (that's frowned upon and hackish)
- Enhance ACPI static fwnode (that are used as fwnodes for ACPI static table devices) to add GSI [number, flags, name] storage to them
 - Option 1: Attach GSI [number, flags, name] “properties” to a software node that is added as ACPI static fwnode secondary
<https://lore.kernel.org/lkml/20260925-acpi-static-table-irq-probe-defer-v1-0-2c62125d0085@kernel.org/>
 - Option 2: Add GSI [number, flag, name] to the ACPI static fwnode itself (struct irq_fwspec + name ?) (are we basically reinventing software nodes ?)



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



GICv5 IWB: Static table devices IRQ probe defer - RFC

```
int acpi_register_gsi(struct device *dev, u32 gsi, int trigger, int polarity)
{
    struct irq_domain *domain;
    struct irq_fwspec fwspec;
    unsigned int irq;

    fwspec.fwnode = acpi_get_gsi_domain_id(gsi);
    if (WARN_ON(!fwspec.fwnode)) {
        pr_warn("GSI: No registered irqchip, giving up\n");
        return -EINVAL;
    }

    domain = irq_find_matching_fwnode(fwspec.fwnode, DOMAIN_BUS_ANY);
    if (!domain)
        return -EPROBE_DEFER;

    fwspec.param[0] = gsi;
    fwspec.param[1] = acpi_dev_get_irq_type(trigger, polarity);
    fwspec.param_count = 2;

    irq = irq_create_fwspec_mapping(&fwspec);
    if (!irq)
        return -EINVAL;

    return irq;
}
```



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



GICv5 IWB: Static table devices IRQ probe defer - RFC

```
int platform_get_irq_affinity(struct platform_device *dev, unsigned int num,  
                             const struct cpumask **affinity)  
{  
    int ret;  
  
    if (is_acpi_static_node(fwnode)) {  
        ret = acpi_static_fwnode_irq_get(fwnode, num);  
        if (ret > 0 || ret == -EPROBE_DEFER)  
            goto out;  
    }  
    ...  
}
```



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



GICv5 IWB: Static table devices IRQ probe defer - RFC

```
static int acpi_static_fwnode_irq_get(const struct fwnode_handle *fwnode, unsigned int index)
{
    u32 gsi, trigger, polarity;
    int ret;

    ...
    fwnode = fwnode->secondary;

    ret = acpi_static_fwnode_read_u32_prop_index(fwnode, ACPI_IRQ_PROP_GSI,
                                                index, &gsi);

    if (ret)
        return ret == -ENOENT ? -ENXIO : ret;

    ...
    return acpi_register_gsi(NULL, gsi, trigger, polarity);
}
```



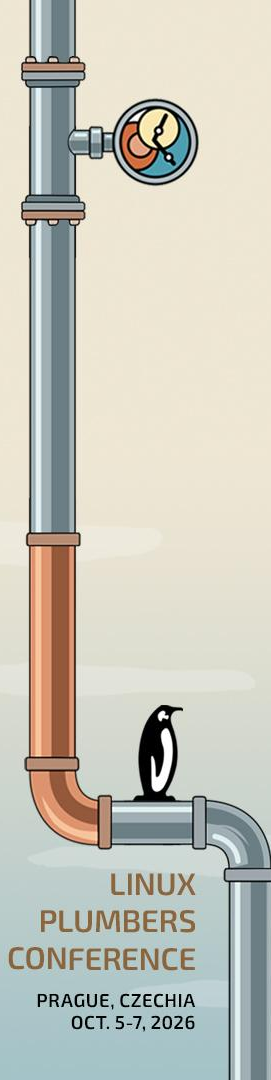
LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



THANKS !



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026