Formalizing Interrupt Systems

11th EuroSys Doctoral Workshop – Lukas Humbel
The Ideal System

- Core 0
- Core 1
- Interrupt controller
- GFX
A Real Intel System

Virtual Core 0

Core 0

Core 1

IOMMU

IOAPIC

LNKA

GFX

RTC

EHCI

Different
Configurable

Software generated interrupts

Virtualization

Memory write

Virtualization
A Real ARM SoC

Not all cores are reachable

accepts

interrupt, different vector
Trends

Hardware **complexity** and **diversity**.

New **demands**:
- Direct-device access
- CPU hot-plug

→ **A lot of C code that might** be correct
→ **Inflexible OS**

---

**No formal description** of hardware.
Hard to handle heterogeneity (M3 cores)
Problem

- How to write generic and correct interrupt configuration code?
- What is good interrupt hardware?
Our approach

Generic Hardware Model

“Test”

Reduction

Additional properties (loop-free, …)

OS representation

HW Feedback

Improve OS Design by:
• Being clear what HW properties are assumed
• Separating concerns
• Providing basis for verification

Improve HW:
• By deriving what properties lead to simpler abstractions.
Reto Achermann: Provable correct memory management.
Overview – Formal methods

 Configuration Updates

 Atomicity of updates

 Configuration Options

 Optimality of configuration algorithms

 Performance Metrics

 Beyond interrupts: RDMA Notifications / USB

 Static Model

 Equivalency (explicit extra assumptions)

 Dynamic

 More concrete

 OS Representation / Algorithms
Thanks for listening, Questions?