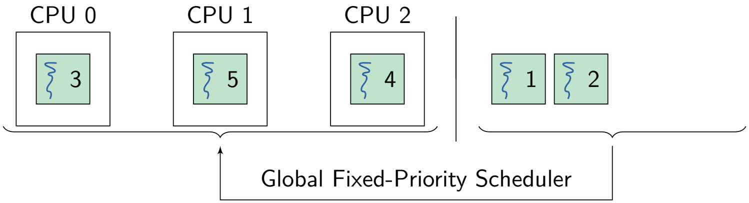
InterSloth: Global Hardware-Based Scheduling in a MultiCore-RTOS on RISC-V

Christian Dietrich

2019-09-24

Fixed-Priority Real-Time Scheduling

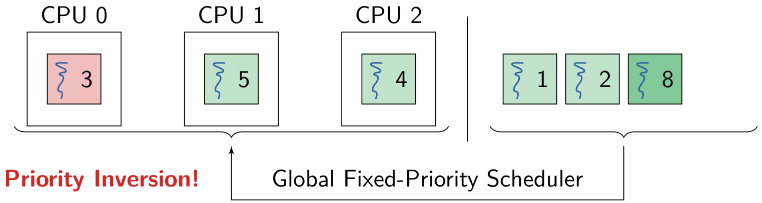
- Schedule Threads according to a Fixed Priority onto Processor(s)
  - Optimal priority assignments for unicore ({rate,deadline} monotonic)
  - For multicore: Global fixed-priority is most flexible schema



- Real-time analysts would like a zero RTOS overhead, but…
  - Global scheduling requires global synchronization (locking)
  - Dispatch on different CPU requires inter-processor interrupt

Fixed-Priority Real-Time Scheduling

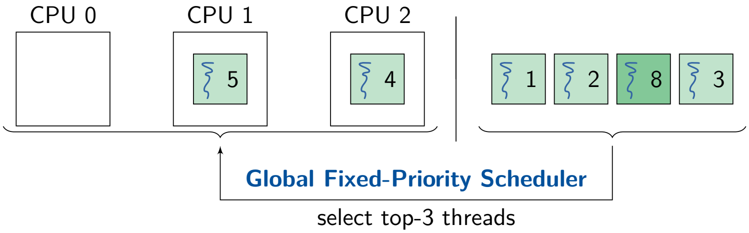
- Schedule Threads according to a Fixed Priority onto Processor(s)
  - Optimal priority assignments for unicore ({rate,deadline} monotonic)
  - For multicore: Global fixed-priority is most flexible schema



- Real-time analysts would like a zero RTOS overhead, but…
  - Global scheduling requires global synchronization (locking)
  - Dispatch on different CPU requires inter-processor interrupt

Fixed-Priority Real-Time Scheduling

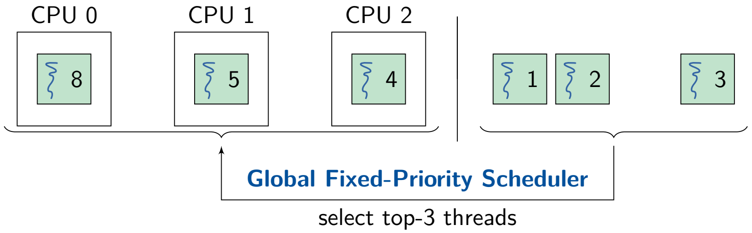
- Schedule Threads according to a Fixed Priority onto Processor(s)
  - Optimal priority assignments for unicore ({rate,deadline} monotonic)
  - For multicore: Global fixed-priority is most flexible schema



- Real-time analysts would like a zero RTOS overhead, but…
  - Global scheduling requires global synchronization (locking)
  - Dispatch on different CPU requires inter-processor interrupt

Fixed-Priority Real-Time Scheduling

- Schedule Threads according to a Fixed Priority onto Processor(s)
  - Optimal priority assignments for unicore ({rate,deadline} monotonic)
  - For multicore: Global fixed-priority is most flexible schema

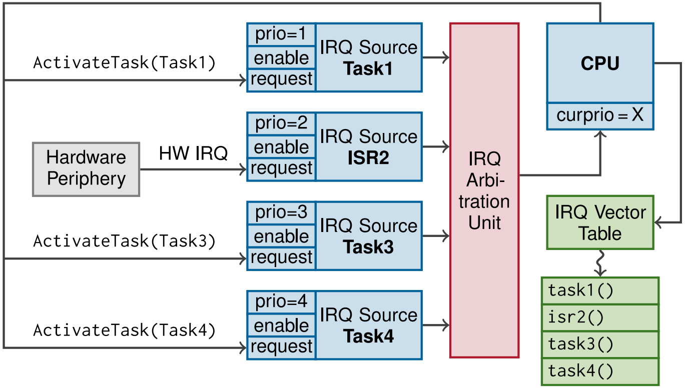


- Real-time analysts would like a zero RTOS overhead, but…
  - Global scheduling requires global synchronization (locking)
  - Dispatch on different CPU requires inter-processor interrupt

Sloth: Threads as Interrupts

[Hof+09; HLSP11; Hof+12] [Dan+14; Mül+14]

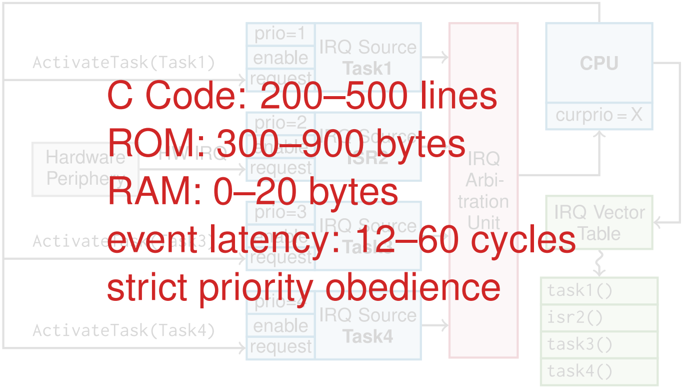
- Sloth RTOS in a Nutshell: Threads $\equiv$ ISR
  - Interrupt controller already selects high-prio interrupt source.
  - Interrupt service routine performs context switch between threads.
  - Supports only unicore and partitioned multicore scheduling.



Sloth: Threads as Interrupts

[Hof+09; HLSP11; Hof+12] [Dan+14; Mül+14]

- Sloth RTOS in a Nutshell: Threads $\equiv$ ISR
  - Interrupt controller already selects high-prio interrupt source.
  - Interrupt service routine performs context switch between threads.
  - Supports only unicore and partitioned multicore scheduling.



C Code: 200–500 lines

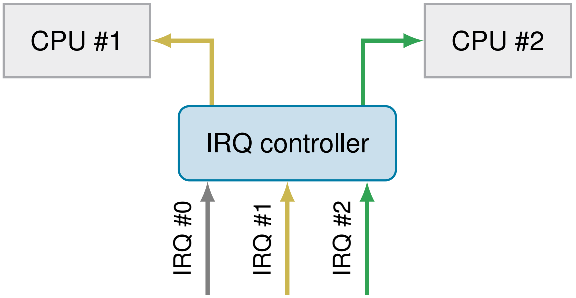
ROM: 300–900 bytes

RAM: 0–20 bytes

event latency: 12–60 cycles

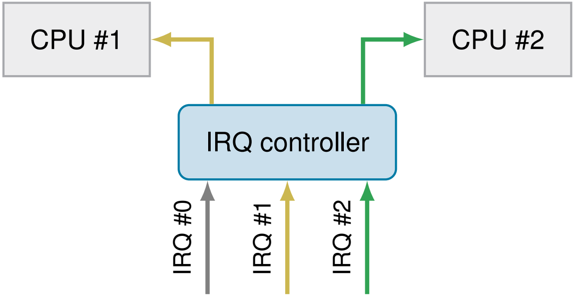
strict priority obedience

InterSloth: Extension to Global FP Scheduling



- We require a strict priority-obedient IRQ controller, but existing…
  - …use a threshold and choose at random (ARM)
  - …do not support re-delivery of IRQs (Intel)
  - …support only fixed CPU–IRQ mapping (Infineon AURIX)

InterSloth: Extension to Global FP Scheduling



- We require a strict priority-obedient IRQ controller, but existing…
  - …use a threshold and choose at random (ARM)
  - …do not support re-delivery of IRQs (Intel)
  - …support only fixed CPU–IRQ mapping (Infineon AURIX)

MIRQ-V: A Strict Priority-Obedient Multi-Core IRQ Controller

Motivation

MIRQ-V: A Strict Priority-Obedient Multi-Core IRQ Controller

InterSloth: An RTOS on Top of MIRQ-V

Evaluation

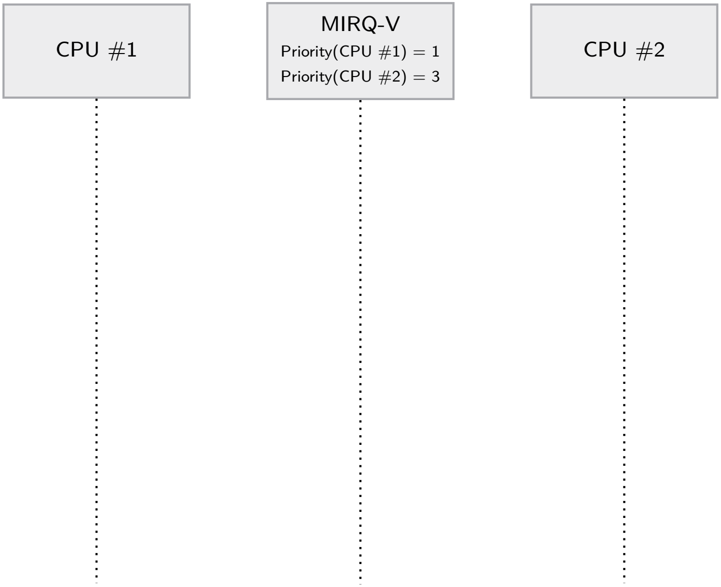
Conclusion

MIRQ-V: Features and Rocket Integration

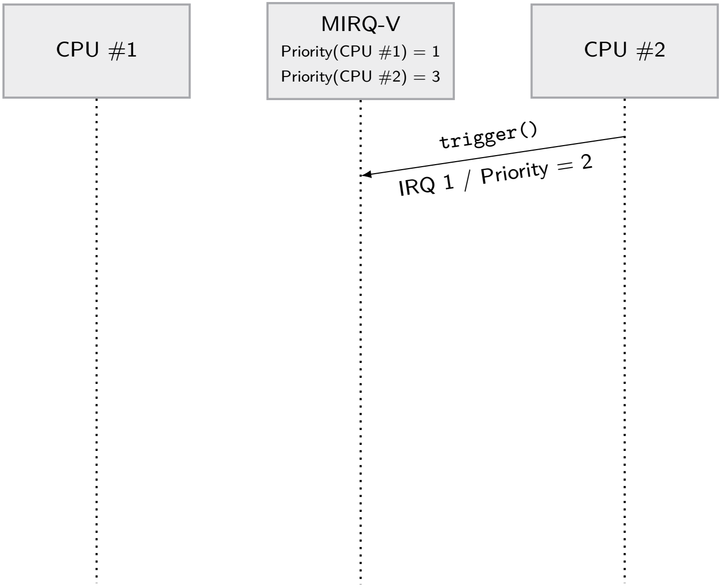
- Feature set is designed for hard real-time systems
  - Freely configurable interrupt and CPU interfaces
  - Up to 255 interrupt/CPU priority levels
  - Highest-priority IRQ is always delivered to lowest-priority CPU
  - Software-triggered IRQ sources and IRQ migration

- Integration with a RISC-V processor
  - Rocket Chip Generator is written in Chisel HDL (Scala DSL)
  - MIRQ-V replaces the *Platform-Level Interrupt Controller* (PLIC)
  - Existing prototype and work on a more efficient implementation

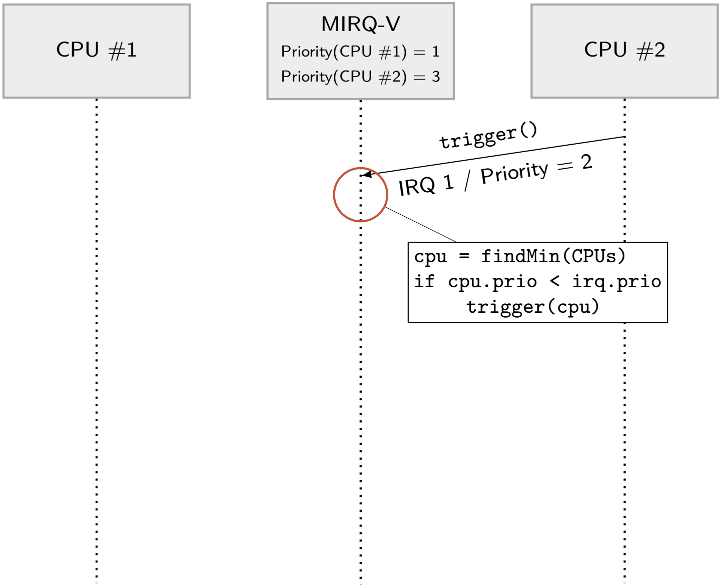
Example Interrupt Delivery with Software IRQ



Example Interrupt Delivery with Software IRQ

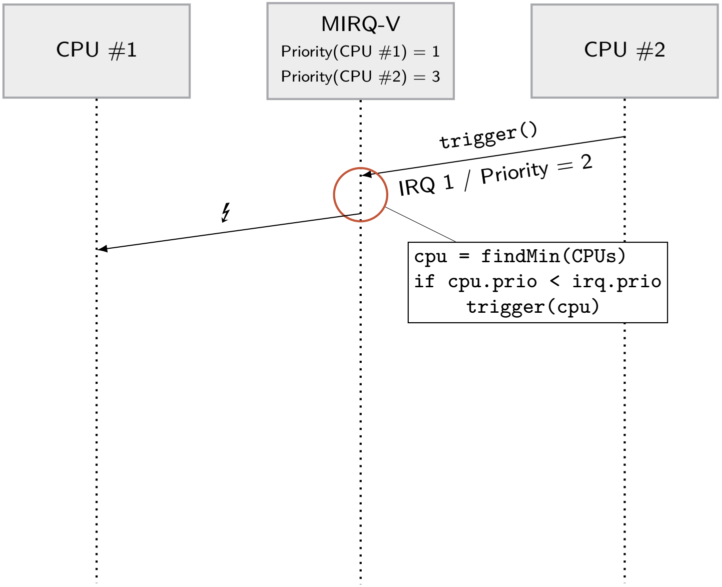


Example Interrupt Delivery with Software IRQ



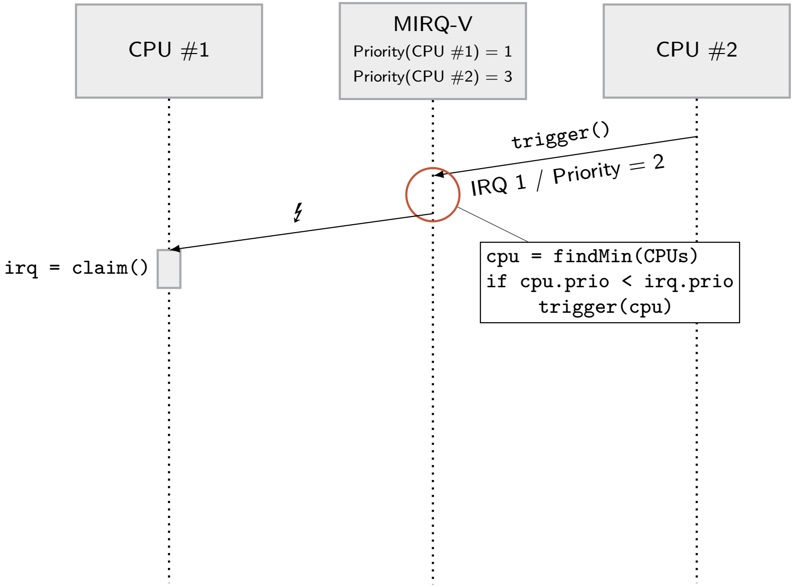
```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

Example Interrupt Delivery with Software IRQ



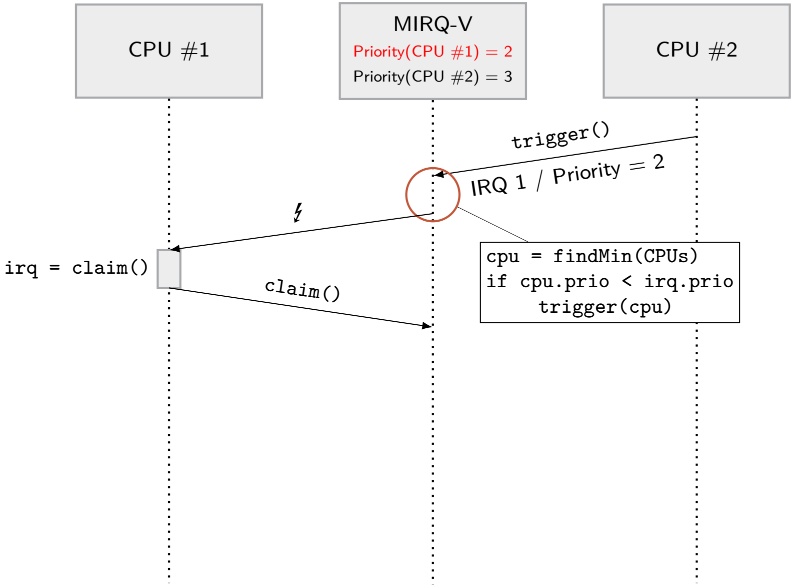
```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

Example Interrupt Delivery with Software IRQ



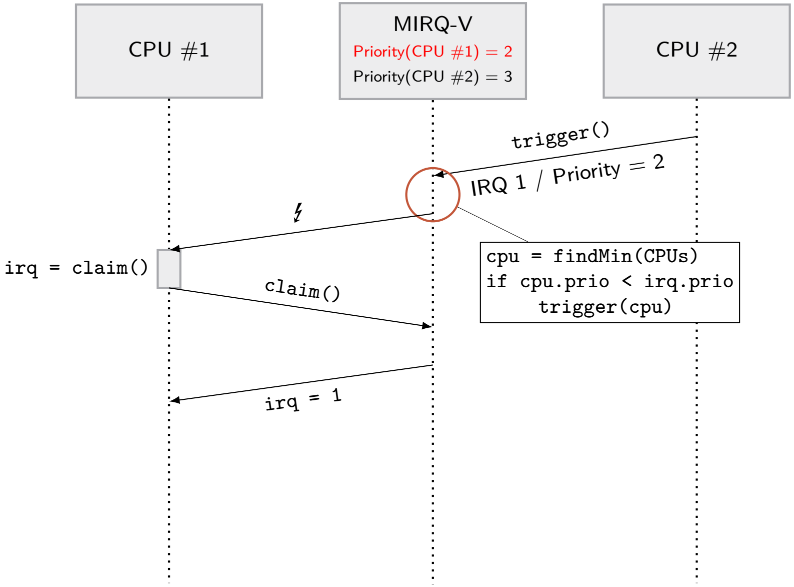
```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

Example Interrupt Delivery with Software IRQ



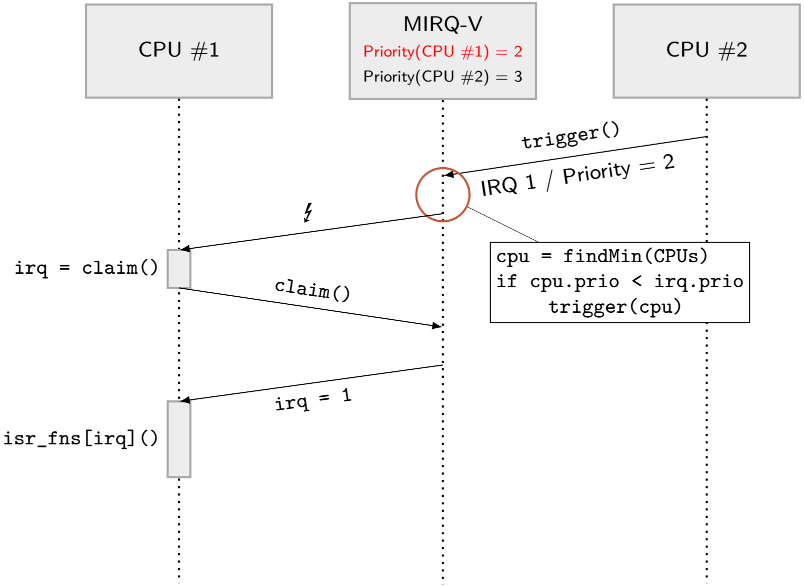
```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

Example Interrupt Delivery with Software IRQ



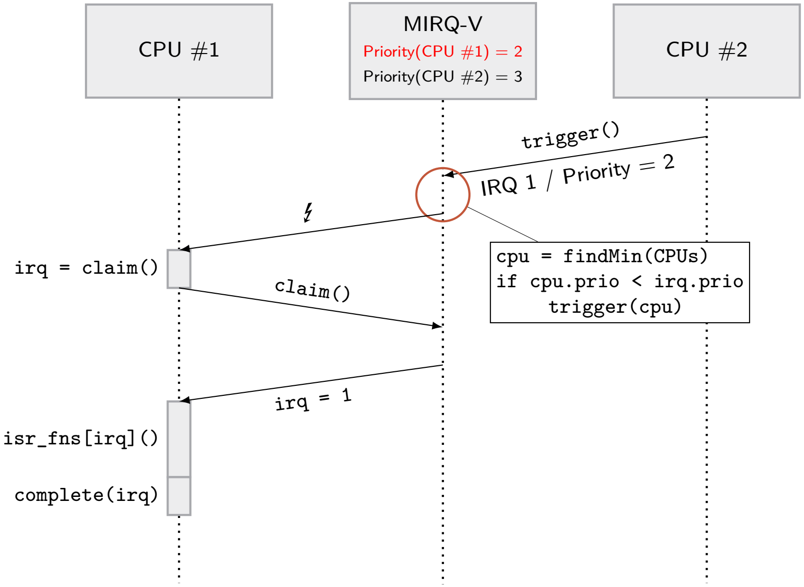
```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

Example Interrupt Delivery with Software IRQ

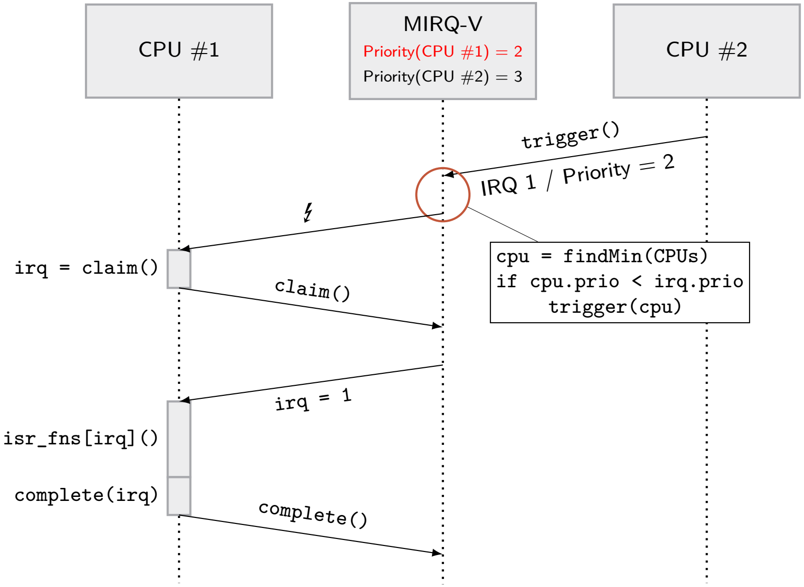


```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

Example Interrupt Delivery with Software IRQ



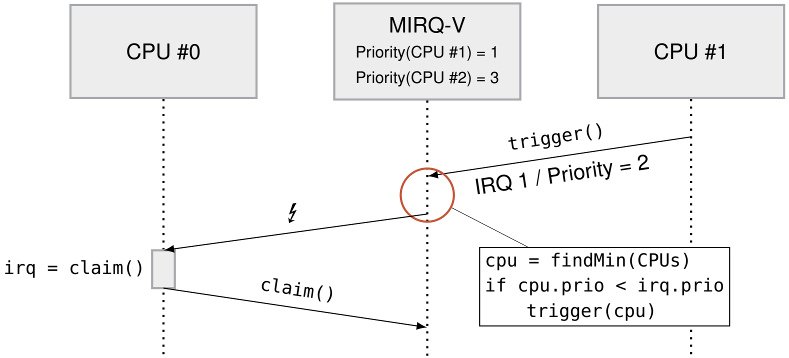
Example Interrupt Delivery with Software IRQ



```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

MIRQ-V: A few technical details

- Race conditions between delivery and CPU-priority changes
  - Normal PLIC IRQ Source has two states: *pending* and *in service*
  - Introduce *delivered* if CPU is informed but has not `claim()`ed.
  - Automatic re-delivery of delivered IRQs if a CPU priority changes.

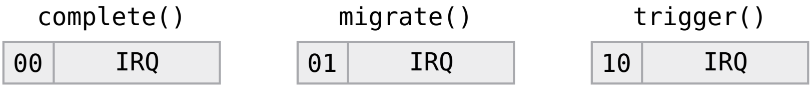


```
cpu = findMin(CPUs)
if cpu.prio < irq.prio
    trigger(cpu)
```

MIRQ-V: A few technical details

- Race conditions between delivery and CPU-priority changes
  - Normal PLIC IRQ Source has two states: *pending* and *in service*
  - Introduce *delivered* if CPU is informed but has not `claim()`ed.
  - Automatic re-delivery of delivered IRQs if a CPU priority changes.

- Backward Compatibility with the original PLIC
  - `migrate()` IRQ to other CPU, `trigger()` from software.
  - Encode new commands into `claim/complete`-register values



Motivation

MIRQ-V: A Strict Priority-Obedient Multi-Core IRQ Controller

InterSloth: An RTOS on Top of MIRQ-V

Evaluation

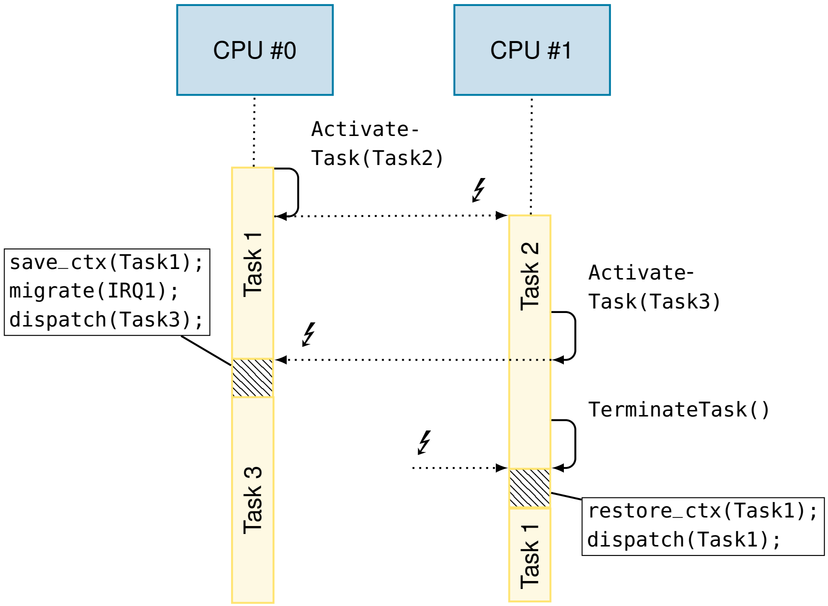
Conclusion

InterSloth: An RTOS for MIRQ-V

- MIRQ-V already performs most of the heavy lifting
  - No global synchronization need, as MIRQ-V is single source of truth.
  - Scheduling and re-scheduling decisions are calculated in parallel.
  - The CPU with the lowest priority is informed about high-priority IRQ.

- InterSloth must handle preemption of already running ISRs
  Remember: Sloth $\Rightarrow$ Thread $\equiv$ ISR
  - Preemption can happen at any time and must be performed transparent.
  - ISR prologue always safes old thread context.
  - Start new thread or resume to old thread context.

Example of Thread Activation and Preemption



Motivation

MIRQ-V: A Strict Priority-Obedient Multi-Core IRQ Controller

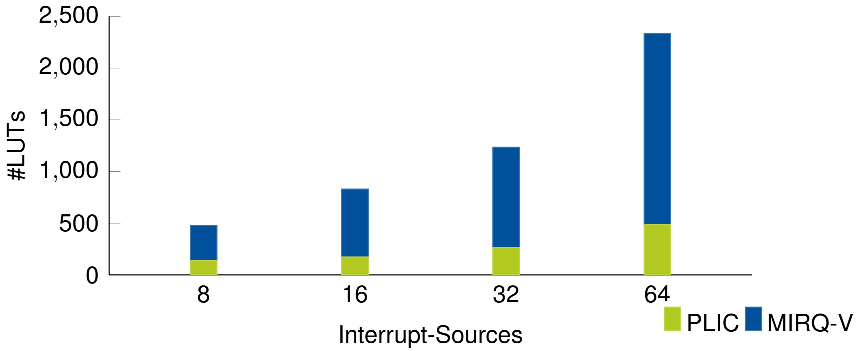
InterSloth: An RTOS on Top of MIRQ-V

Evaluation

Conclusion

Evaluation: MIRQ-V

- IRQ arbitration in 1 cycle, delivery in 4 cycles
- MIRQ-V Design with parametrizable number of IRQs/CPUs
  - LUT demand is linearly larger than PLIC LUT demand
  - Full Rocket Chip uses 23.000 LUTs



Evaluation: InterSloth

- Cycle-Accurate Simulator on Verilog level (Verilator)
  - Timing Measurements from Software with RISC-V's `mcycle` register
  - Measure time that is spent in the InterrSloth kernel

- Measure three characteristic InterSloth system calls
  - Context switch must save and restore 32 general-purpose registers.

System Call	Cycles	Instructions
`ActivateTask()` *Trigger thread and dispatch on different CPU*	318	130
`TerminateTask()` *Start the idle thread*	93	60
`ChainTask()` *Destroy current thread and dispatch in this CPU*	261	148

Conclusion

- Problems for Multi-Core Real-Time Scheduling
  - Overheads for Inter-Core Communication and Synchronization
  - Multi-Core IRQ Controllers are not strictly enforcing priorities.
- MIRQ-V: A Strict Priority-Obedient Multi-Core IRQ Controller
  - Always route highest-priority IRQ to lowest-priority CPU.
  - Parametrizable Hardware Design with Rocket (RISC-V) Integration
  - Software-Triggered IRQs and IRQ Migration
- InterSloth: Minimal-Effort Global Fixed-Priority Scheduling
  - ISRs save the old and restore/start the new thread context.
  - Requires no global synchronization between CPUs
- Future Work
  - MIRQ-V: Decrease LUT Demand by Optimized Delivery Invalidation
  - InterSloth: Support more RTOS Primitives (Mutexes, Alarms, Events)

Bibliography

[Dan+14] Daniel Danner et al. “Safer Sloth: Efficient, Hardware-Tailored Memory Protection”. In: *Proceedings of the 20th IEEE International Symposium on Real-Time and Embedded Technology and Applications (RTAS ’14)*. Washington, DC, USA: IEEE Computer Society Press, 2014, pp. 37–47.

[HLSP11] Wanja Hofer, Daniel Lohmann, and Wolfgang Schröder-Preikschat. “Sleepy Sloth: Threads as Interrupts as Threads”. In: *Proceedings of the 32nd IEEE International Symposium on Real-Time Systems (RTSS ’11)* (Vienna, Austria, Nov. 29, 2011–Dec. 2, 2011). IEEE Computer Society Press, Dec. 2011, pp. 67–77. ISBN: 978-0-7695-4591-2. DOI: 10.1109/RTSS.2011.14.

[Hof+09] Wanja Hofer et al. “Sloth: Threads as Interrupts”. In: *Proceedings of the 30th IEEE International Symposium on Real-Time Systems (RTSS ’09)* (Washington, D.C., USA, Dec. 1, 2009–Dec. 4, 2009). IEEE Computer Society Press, Dec. 2009, pp. 204–213. ISBN: 978-0-7695-3875-4. DOI: 10.1109/RTSS.2009.18.

[Hof+12] Wanja Hofer et al. “Sloth on Time: Efficient Hardware-Based Scheduling for Time-Triggered RTOS”. In: *Proceedings of the 33rd IEEE International Symposium on Real-Time Systems (RTSS ’12)* (San Juan, Puerto Rico, Dec. 4, 2012–Dec. 7, 2012). IEEE Computer Society Press, Dec. 2012, pp. 237–247. ISBN: 978-0-7695-4869-2. DOI: 10.1109/RTSS.2012.75.

[Mül+14] Rainer Müller et al. “MultiSloth: An Efficient Multi-Core RTOS using Hardware-Based Scheduling”. In: *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS ’14)* (Madrid, Spain). Washington, DC, USA: IEEE Computer Society Press, 2014, pp. 289–198. ISBN: 978-1-4799-5798-9. DOI: 10.1109/ECRTS.2014.30.