Responsive and Enforced Interrupt Handling for Real-Time System Virtualization

Hyoseung Kim* Shige Wang† Raj Rajkumar *

*Carnegie Mellon University

† General Motors R&D

Workload Consolidation

- **Multi-core CPUs for embedded real-time systems**
  - **Automotive:**
    - Freescale i.MX6 4-core CPU
    - NVIDIA Tegra K1 platform
  - **Avionics and defense:**
    - Rugged Intel i7 single board computers
    - Freescale P4080 8-core CPU



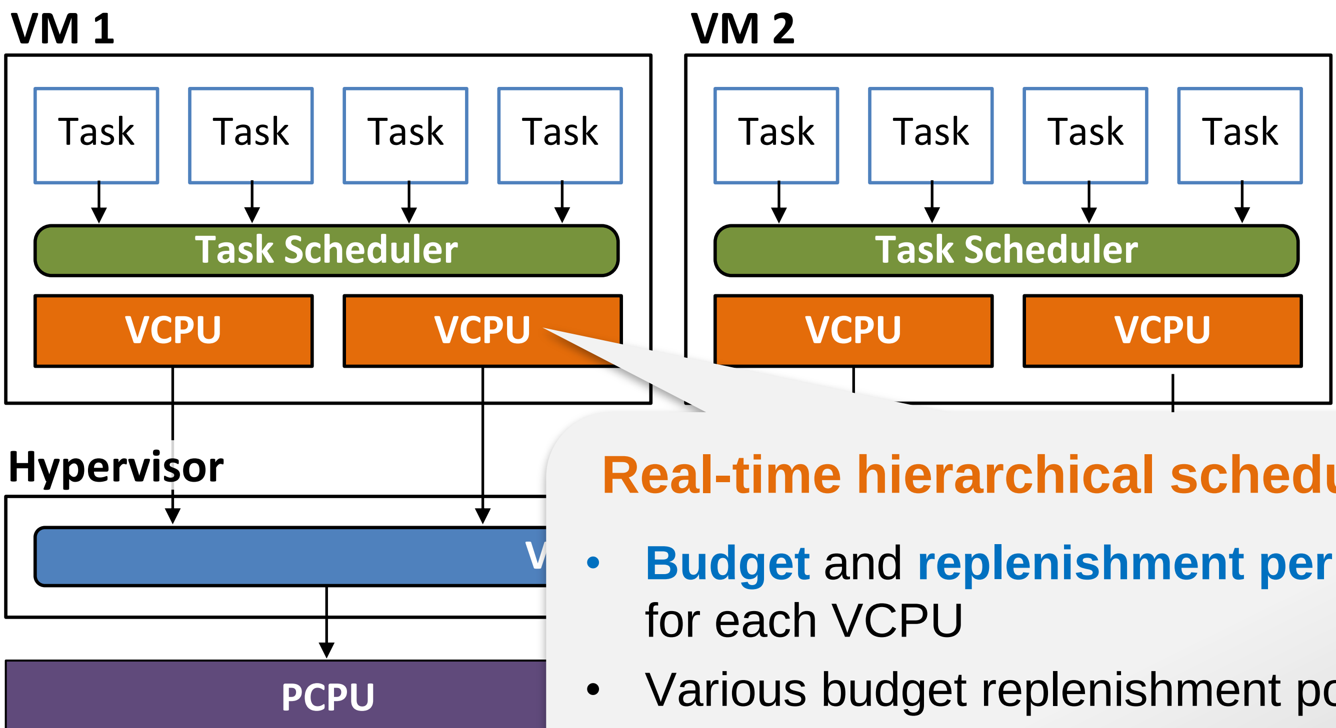

- **Consolidation of real-time applications onto a single hardware platform**
  - Reduces the number of CPUs and wiring harness among them
  - Leads to a significant reduction in cost and space requirements

Benefits of Real-Time Virtualization

- **Barrier to consolidation**
  - Each app. could have been developed independently by different vendors
    - Heterogeneous S/W infrastructure
    - Bare-metal / Proprietary OS
    - Linux / Android
  - Different license issues
- **Consolidation via virtualization**
  - Each application can maintain its own implementation
  - Minimizes re-certification process
  - IP protection, license segregation
  - Fault isolation

Scheduling in Virtualization

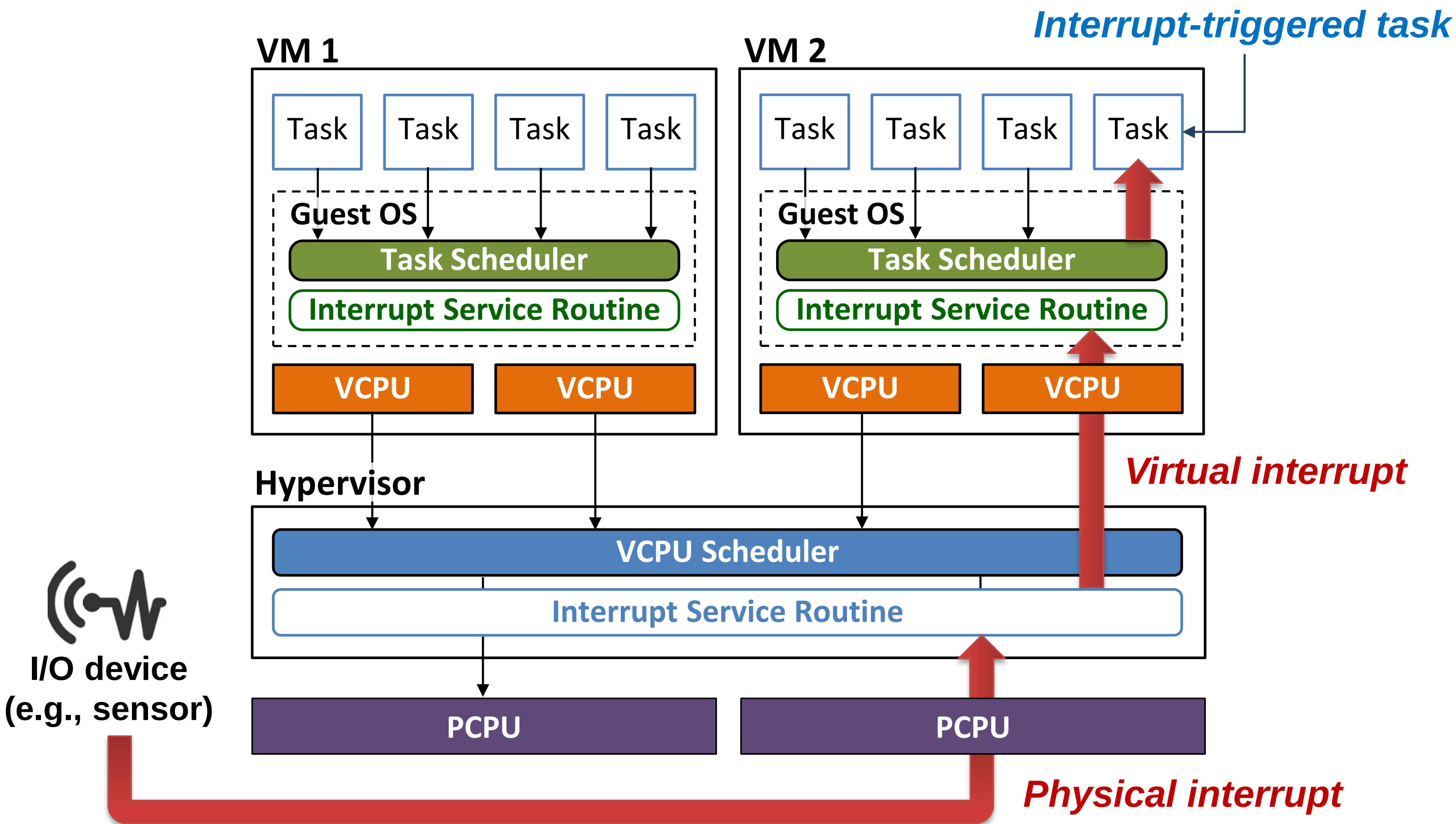
- Two-level hierarchical scheduling structure
  - Task scheduling and VCPU scheduling



Real-time hierarchical scheduling

- **Budget** and **replenishment period** for each VCPU
- Various budget replenishment policies (e.g., deferrable server)

Interrupt Handling in Virtualization



Requirements for Interrupt Handling

- **R1: Responsive and bounded interrupt handling time**
  - Timing penalties to interrupt handling in virtualization
- **R2: Protect real-time tasks from interrupt storms**
  - Task schedulability should be guaranteed
- **R3: Support unmodified guest OSs**
  - Many commercial RTOSs are closed-source

Previous Work

	R1			R2		R3
	Priority based sched.	VCPU temporal isolation	Bounded Interrupt handling	Interrupt storm protection	Task sched. guarantee	Unmodified guest OSs
[1]			✓	✓		✓
[2]	✓	✓				
[3]	✓	✓				
[4]	✓	✓				
Ours	✓	✓	✓	✓	✓	✓

[1] M. Beckert et al. Sufficient temporal independence and improved interrupt latencies in a real-time hypervisor. In DAC, 2014.
[2] J. Kiszka. Towards Linux as a real-time hypervisor. In RTLWS, 2009.
[3] A. Lackorzy´nski, A. Warg, M. Volp, and H. H¨artig. Flattening hierarchical scheduling. In EMSOFT, 2012.
[4] R. Ma et al. Performance tuning towards a KVM-based embedded real-time virtualization system. J. Inf. Sci. Eng., 29(5):1021–1035, 2013.

Our Approach

- **vINT**: an analyzable interrupt handling framework for real-time system virtualization
  - Provides responsive, bounded, and enforced interrupt handling
  - Does not require any change to the guest OS code
    - Easily applicable to virtualizing proprietary, closed-source RTOSs

- Contributions
  - vINT framework design
  - Analysis on interrupt handling time and VCPU/task schedulability
  - Implementation and case study on the KVM hypervisor of Linux/RK

KVM + LINUX/RK

GM

Carnegie Mellon

Outline

- Introduction
- vINT Framework
  - System model
  - Problems with interrupt handling
  - vINT details
  - Analysis
- Evaluation
- Conclusion

System Model (1)

- Partitioned fixed-priority scheduling for both VCPUs and tasks
  - Widely supported in many real-time OSs and hypervisors
  - e.g., OKL4, PikeOS, …
- VCPU $v_i: (C_i^v, T_i^v)$
  - C_i^v: Maximum execution budget
  - T_i^v: Budget replenishment period

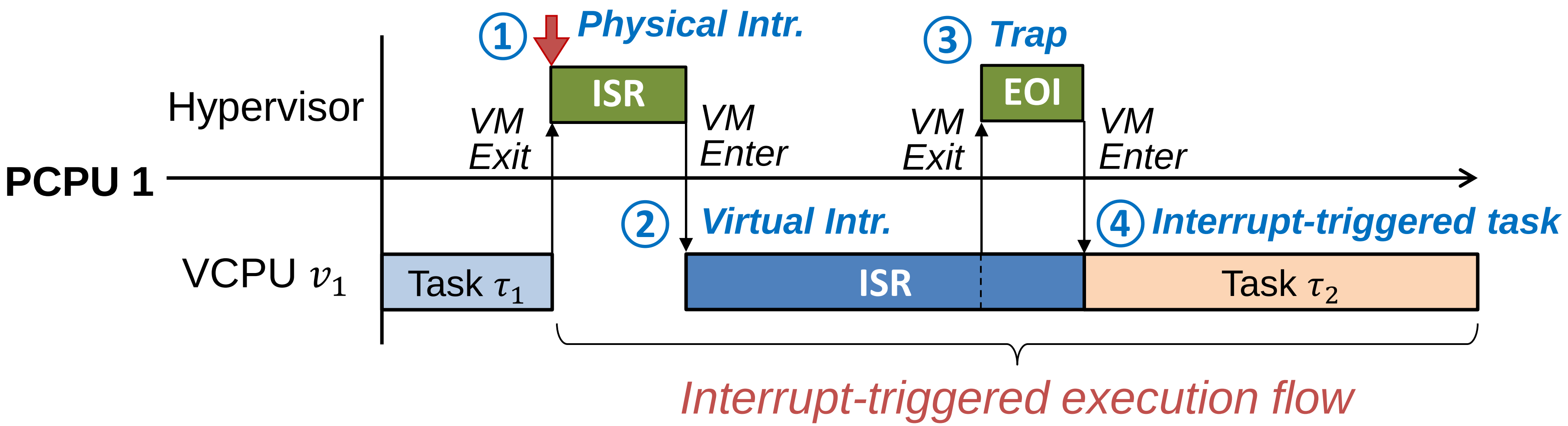
Any task or OS code can execute only if the corresponding VCPU has a non-zero remaining budget

- VCPU budget replenishment policies
  - Deferrable server & sporadic server
- Task $\tau_i: (C_i, T_i)$
  - C_i: Worst-case execution time (WCET)
  - T_i: Minimum inter-arrival time

System Model (2)

Min. inter-arrival time expected at design time
→ Interrupt storms may happen at runtime

- Physical interrupt $I_i^{pi}: (C_i^{pi}, T_i^{pi})$
  - A signal issued from a hardware device to a PCPU
  - Handled by the corresponding ISR of the hypervisor
- Virtual interrupt $I_j^{vi}: (C_i^{vi}, T_i^{vi})$
  - A software signal from the hypervisor to a VCPU
  - Handled by the ISR of the guest OS while consuming the VCPU budget

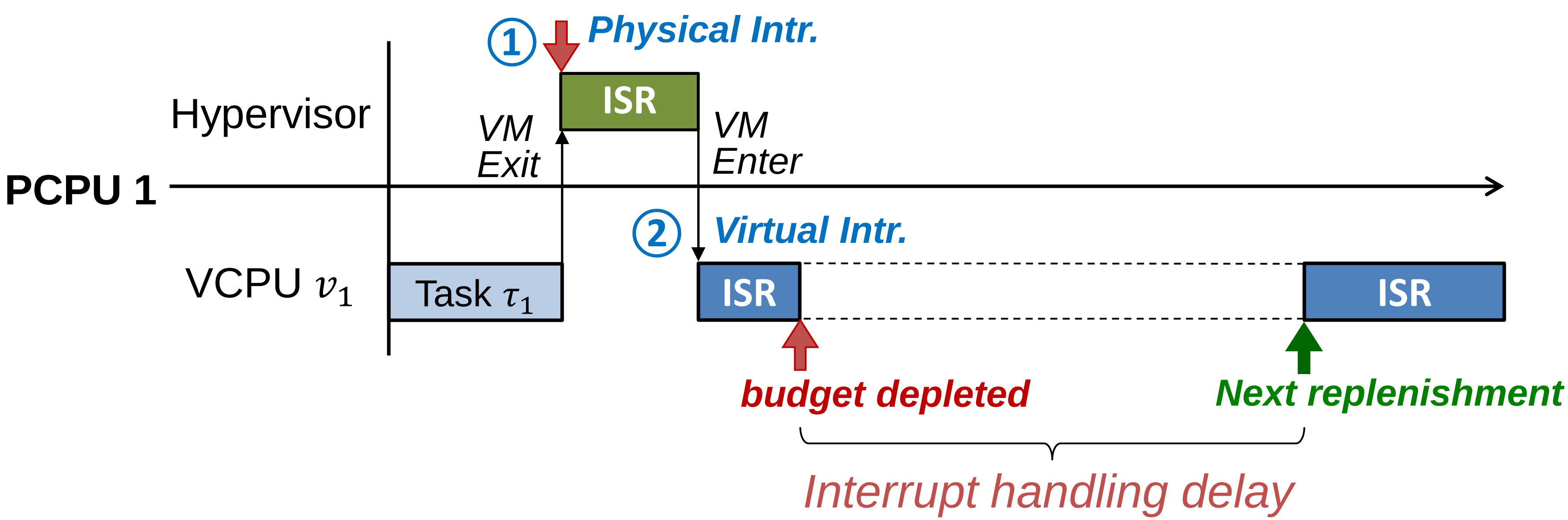


GM

Carnegie Mellon

Problems with Virtual Interrupts (1)

- **Virtual interrupt**
  - Main difference between interrupt handling in virtualized and non-virtualized environments
- **Problem 1: Timing penalties to virtual interrupt handling**
  - VCPU budget depletion and VCPU preemption



Carnegie Mellon

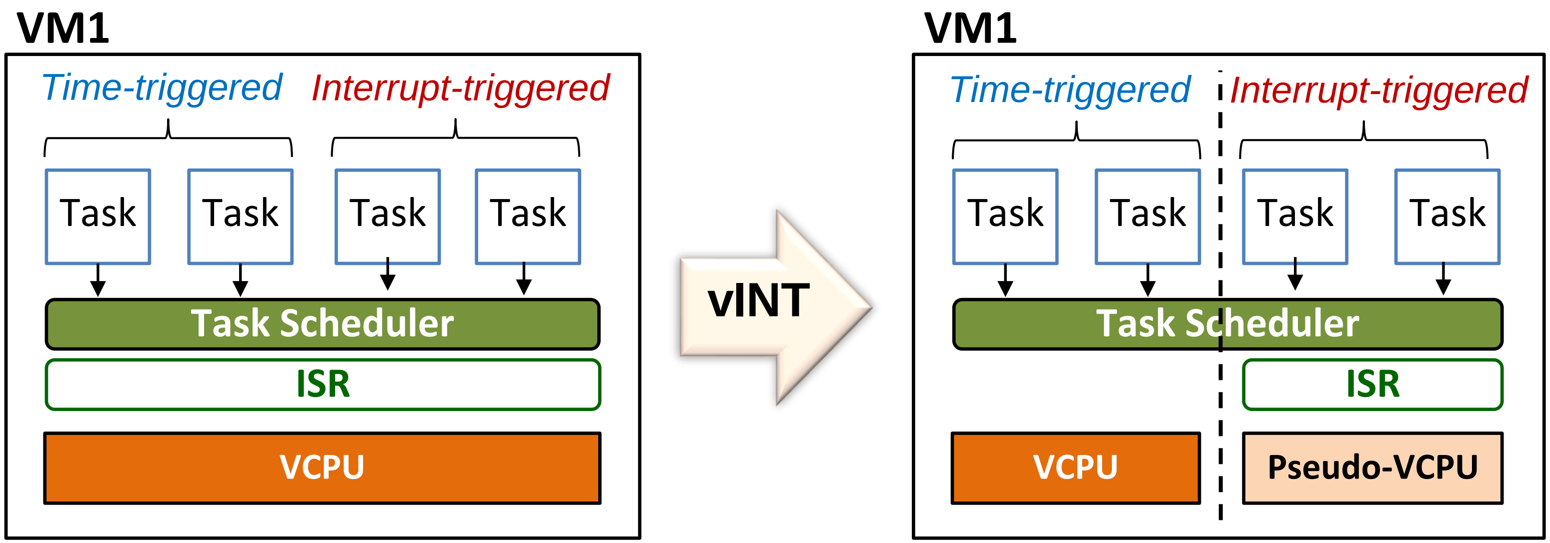
Problems with Virtual Interrupts (2)

- **Problem 2: Virtual interrupt storms**
  - VCPU typically has a *fraction* of physical CPU time as its budget
  - Negative impact of virtual interrupt storm can be much significant than physical interrupt storms

- Prior work developed for non-virtualized systems
  - Cannot address virtual interrupt storms due to the unawareness of the passage of physical time within a VM

vINT Overview

- Conceptually splits virtual interrupt handling from the VCPU of regular tasks in an analyzable way
  - Used pseudo-VCPU abstraction
  - Prioritizes virtual interrupt handling
  - Does not require any guest OS modification



Pseudo-VCPU Parameters

- Same types of parameters as a regular VCPU: (C_p^v, T_p^v)
- Budget replenishment period T_p^v
  - Equal to or greater than the minimum inter-arrival time of the associated interrupt
- Execution budget C_p^v

$$C_p^v = \left\lceil \frac{T_p^v}{T_i^{vi}} \right\rceil \left(\mathbb{C}_i^{vi} + \sum_{\substack{I_j^{vi} \in \mathbb{V}(I_i^{vi}) \wedge \\ pseudo(I_j^{vi}) = \emptyset}} \left\lceil \frac{T_i^{vi}}{T_j^{vi}} \right\rceil C_j^{vi} \right)$$

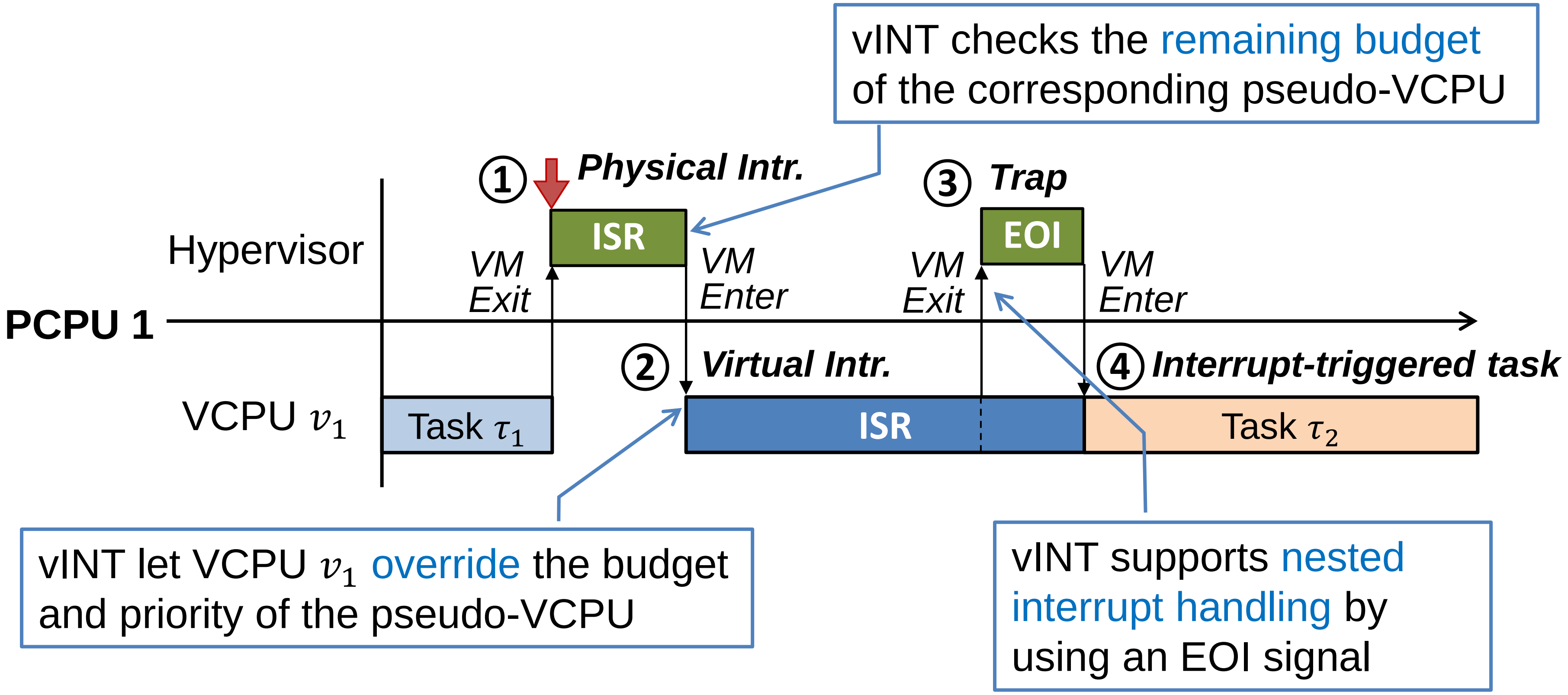
Sum of execution times of ISR and interrupt-triggered task

Extra budget to reduce blocking time on interrupt handling

Carnegie Mellon

Pseudo-VCPU Realization

- Pseudo-VCPU does not have an execution context
  - vINT handles a virtual interrupt as if it was handled in its pseudo-VCPU



Carnegie Mellon

Analysis

- **Scope of our analysis**
  - Interrupt handling time
  - VCPU schedulability
  - Task schedulability
- Considers four different use cases

VCPU budget replenish policies	With vINT	Without vINT
Deferrable server	YES	YES
Sporadic server	YES	YES

Interrupt Handling Time Analysis

- **Interrupt handling time**
  - Sum of physical and virtual interrupt response times
- **Physical interrupt response time**

$$W_i^{pi,n+1} = C_i^{pi} + \sum_{I_h^{pi} \in \mathbb{P}(I_i^{pi}) \wedge \pi_h^{pi} > \pi_i^{pi}} \left\lceil \frac{W_i^{pi,n}}{T_h^{pi}} \right\rceil C_h^{pi}$$

Similar to interrupt handling time in a non-virtualized environment

- **Virtual interrupt response time**

[without vINT]

$$W_j^{vi,n+1} = \mathbb{C}_j^{vi} + \sum_{\substack{\tau_h \in \mathbb{V}(I_j^{vi}) \wedge \pi_h > \check{\pi}_{\mathbb{D}} \\ \wedge pseudo(\tau_h) = \emptyset}} \left\lceil \frac{W_j^{vi,n} + J_h}{T_h} \right\rceil C_h + \left\lceil \frac{W_j^{vi,n} + C_k^v}{T_k^v} \right\rceil (T_k^v - C_k^v) + \sum_{\substack{I_u^{vi} \in \mathbb{V}(I_j^{vi}) \wedge u \neq j \\ pseudo(I_u^{vi}) = \emptyset}} \left\lceil \frac{W_j^{vi,n} + J_u^{vi}}{T_u^{vi}} \right\rceil C_u^{vi}$$

Delay from VCPU budget depletion

Delay from time-triggered tasks

[vINT]

$$W_j^{vi,n+1} = \mathbb{C}_j^{vi} + B_{p,j}(W_j^{vi,n}) + \sum_{I_u^{pi} \in \mathbb{P}(v_p)} \left\lceil \frac{W_j^{vi,n}}{T_u^{pi}} \right\rceil C_u^{pi} + \sum_{v_h \in \mathbb{P}(v_p) \wedge \pi_h^v > \pi_p^v} \left\lceil \frac{W_j^{vi,n} + J_h^v}{T_h^v} \right\rceil C_h^v$$

Delay from higher-priority interrupt handling

GM

Carnegie Mellon

Outline

- Introduction
- vINT Framework
- Evaluation
  - Performance characteristics of vINT
  - Implementation
  - Case study
- Conclusion

Performance Characteristics of vINT

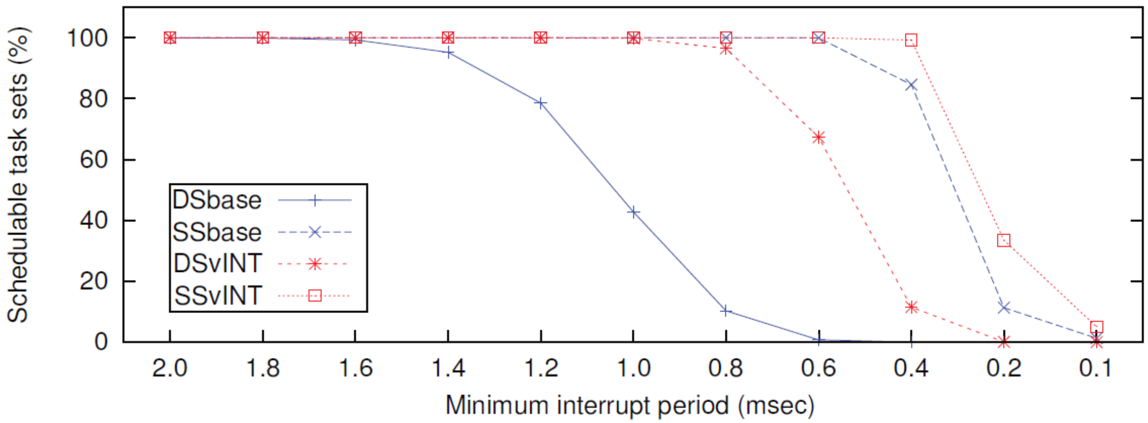
- **Purpose**: Empirically investigate the performance characteristics and benefits of vINT

DSbase	Deferrable Server without vINT (baseline)
SSbase	Sporadic Server without vINT (baseline)
DSvINT	Deferrable Server with vINT
SSvINT	Sporadic Server with vINT

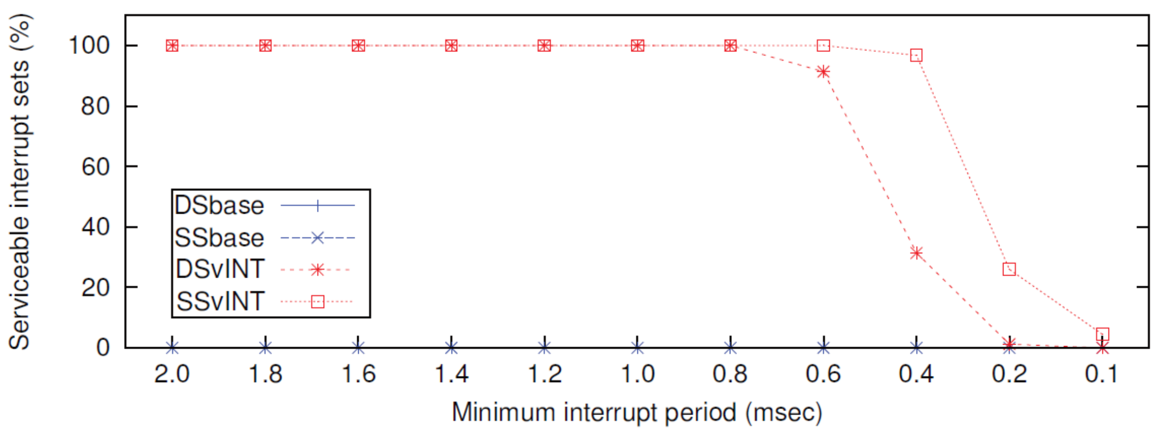
- **Experimental setup**
  - Used randomly-generated task sets and interrupt sets
  - Metrics
    - Percentage of schedulable task sets
    - Percentage of serviceable interrupt sets

Experimental Results (1)

- Interrupts with short inter-arrival times
  - Task schedulability



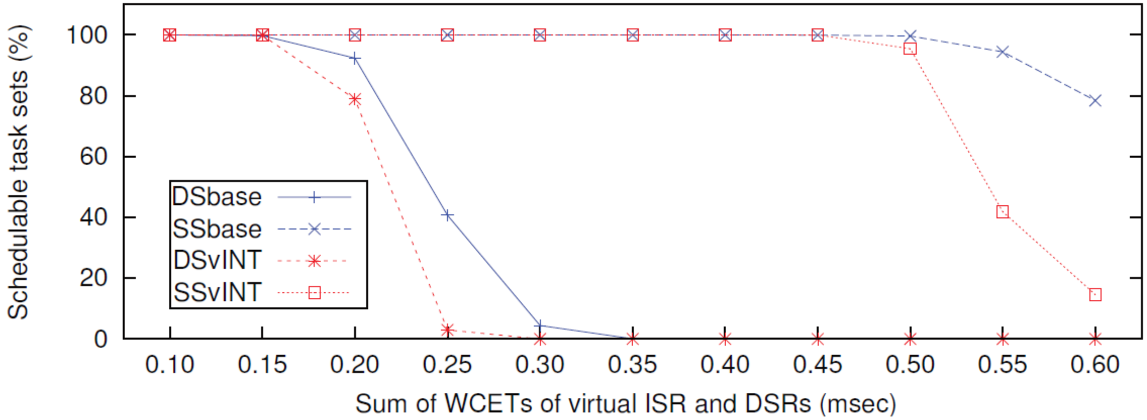
  - Interrupt service rate



vINT has benefits in both task scheduling and interrupt handling

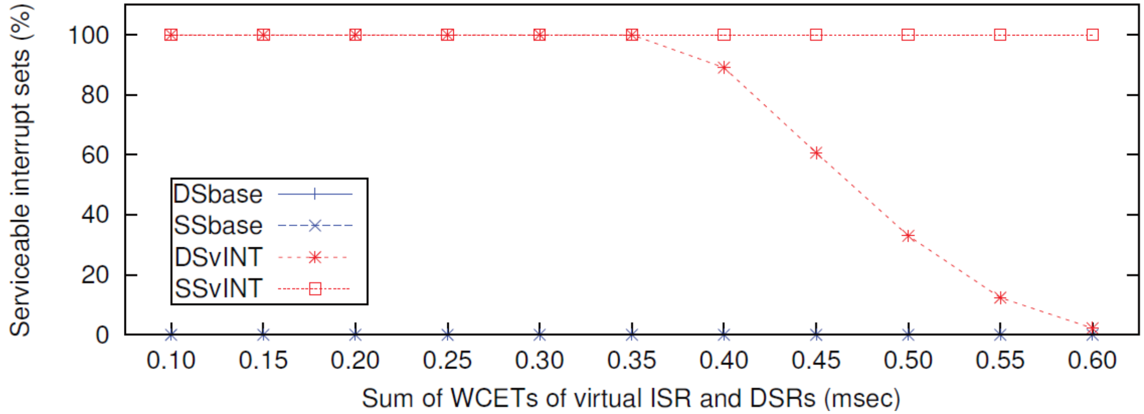
Experimental Results (2)

- WCET of interrupt handlers
  - Task schedulability



vINT shows slightly lower task schedulability

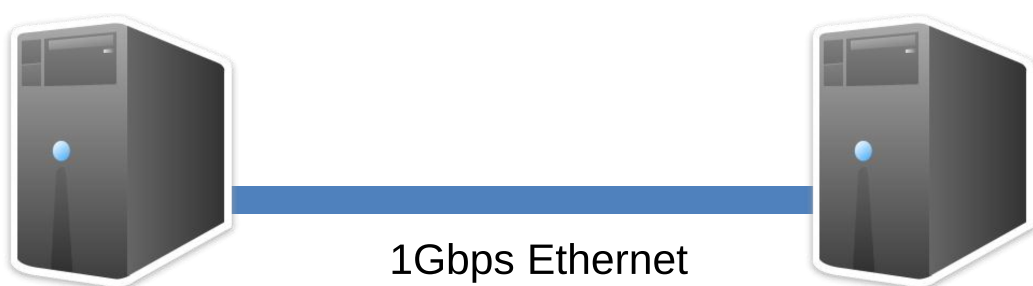
  - Interrupt service rate



But vINT provides significantly higher interrupt service rates

Case Study

- System configuration
  - Hypervisor: KVM of Linux/RK
    - Chosen for convenience
    - vINT applied to a Gigabit PCI NIC
  - Guest VM
    - OS: Unmodified Linux kernel 3.10
    - Tasks: Netperf (network benchmark tool), Mplayer (movie player), Busyloop (background task)



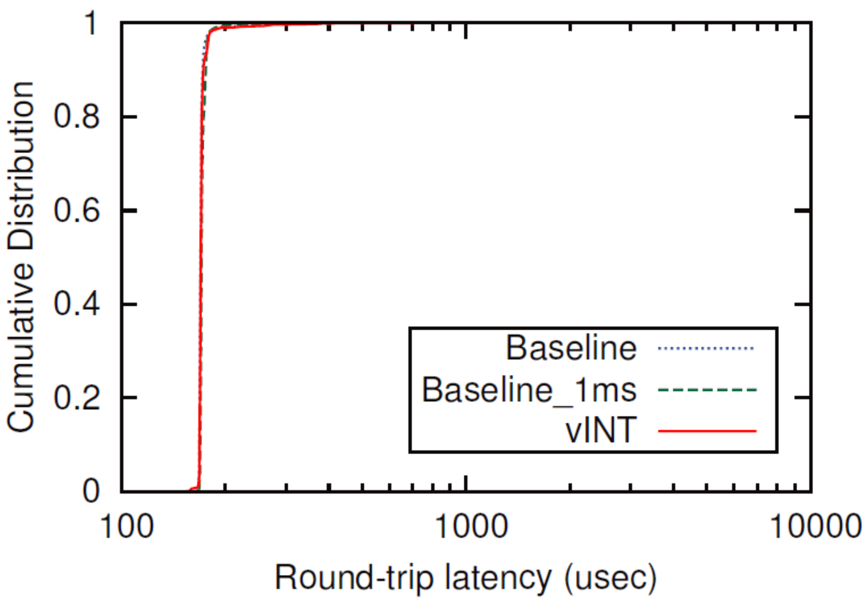
Netperf receiver, Mplayer and Busyloop running in a VM

Remote machine (Netperf sender)

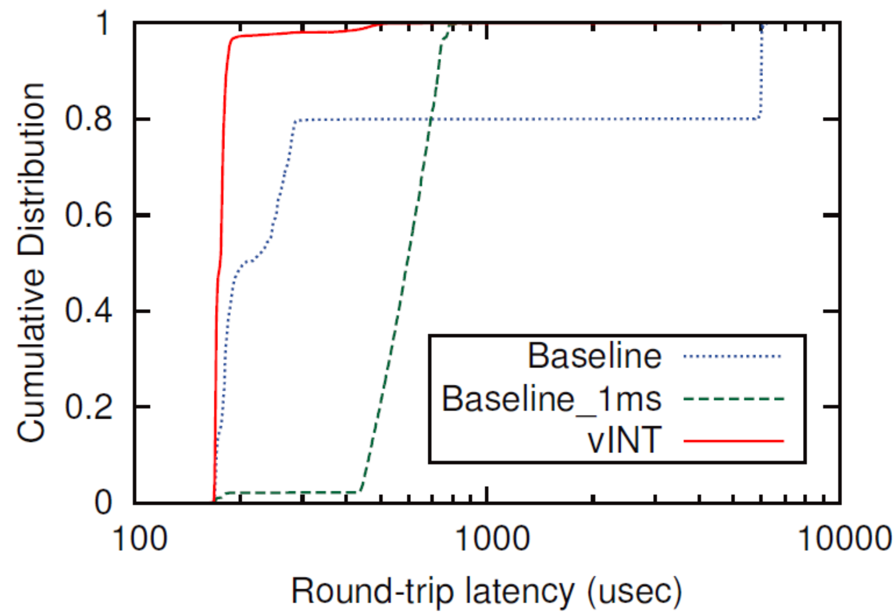
Netperf Round-Trip Latency

- Highly affected by system's interrupt handling time

[Idle]



[Mplayer + Busyloop]



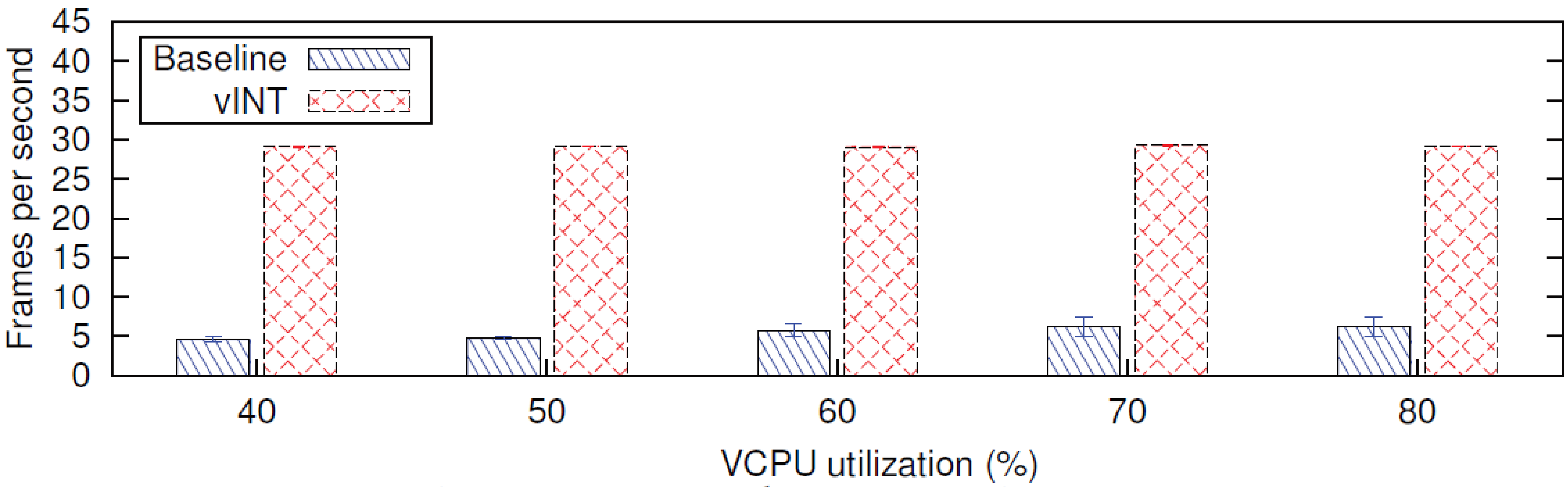
- Netperf with vINT: handles 95% of round-trips in 200 μsec
- Netperf without vINT: only 50% during that time

GM

Carnegie Mellon

Mplayer QoS under Interrupt Storms

- Measured fps(frames-per-second) of video playback
  - MPEG2 video stream recorded in 29.97 fps
  - X-axis: total VCPU budget assigned



- Mplayer with vINT: nearly unaffected
- Mplayer without vINT: dropped from 29.97 fps to 6 fps

Conclusions

- **vINT**: an interrupt handling framework for RT virtualization
  - Provides responsive and bounded interrupt handling time
  - Protects real-time tasks from interrupt storms
  - Supports unmodified guest OSs
- Analysis and Experimental Results
  - Timely interrupt handling and good task schedulability in most cases
  - A system designer can choose a trade off between task schedulability and interrupt handling time for each interrupt
- Implementation and Case study
  - KVM + Linux/RK: https://rtml.ece.cmu.edu/redmine/projects/rk/
- Future Work
  - Memory interference, efficient VCPU resource allocation