ELI: Bare-Metal Performance for I/O Virtualization

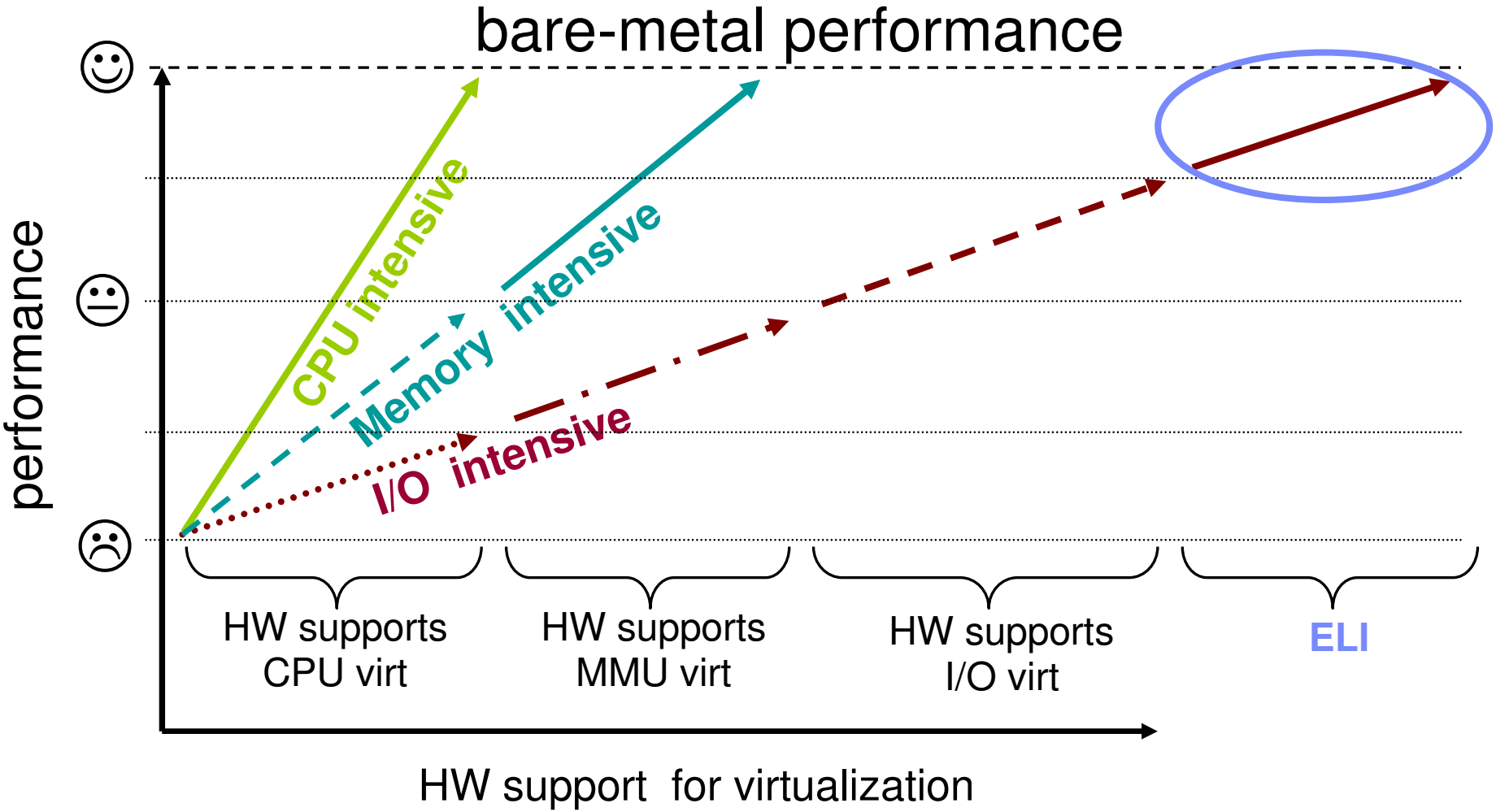
Abel Gordon[×] Nadav Amit[¤] Nadav Har'El[×]
Muli Ben-Yehuda[×,¤] Alex Landau[×] Assaf Schuster[¤] Dan Tsafrir[¤]

[×] IBM Research – Haifa
[¤] Technion – Israel Institute of Technology

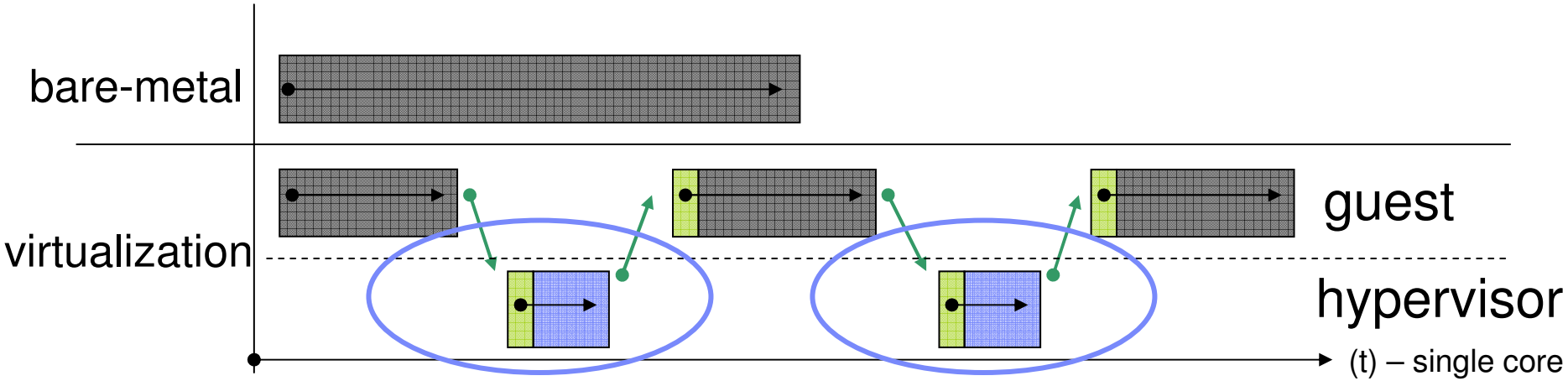
Why (not) Virtualization ?

- Pros
  - Consolidation
  - Flexibility
  - Cloud computing
  - Simplified management
  - Migration
  - ….
- Cons
  - Performance degradation
  - Performance degradation
  - Performance degradation
  - …

x86 Virtualization Performance



The Cause of x86 I/O Virtualization Overhead

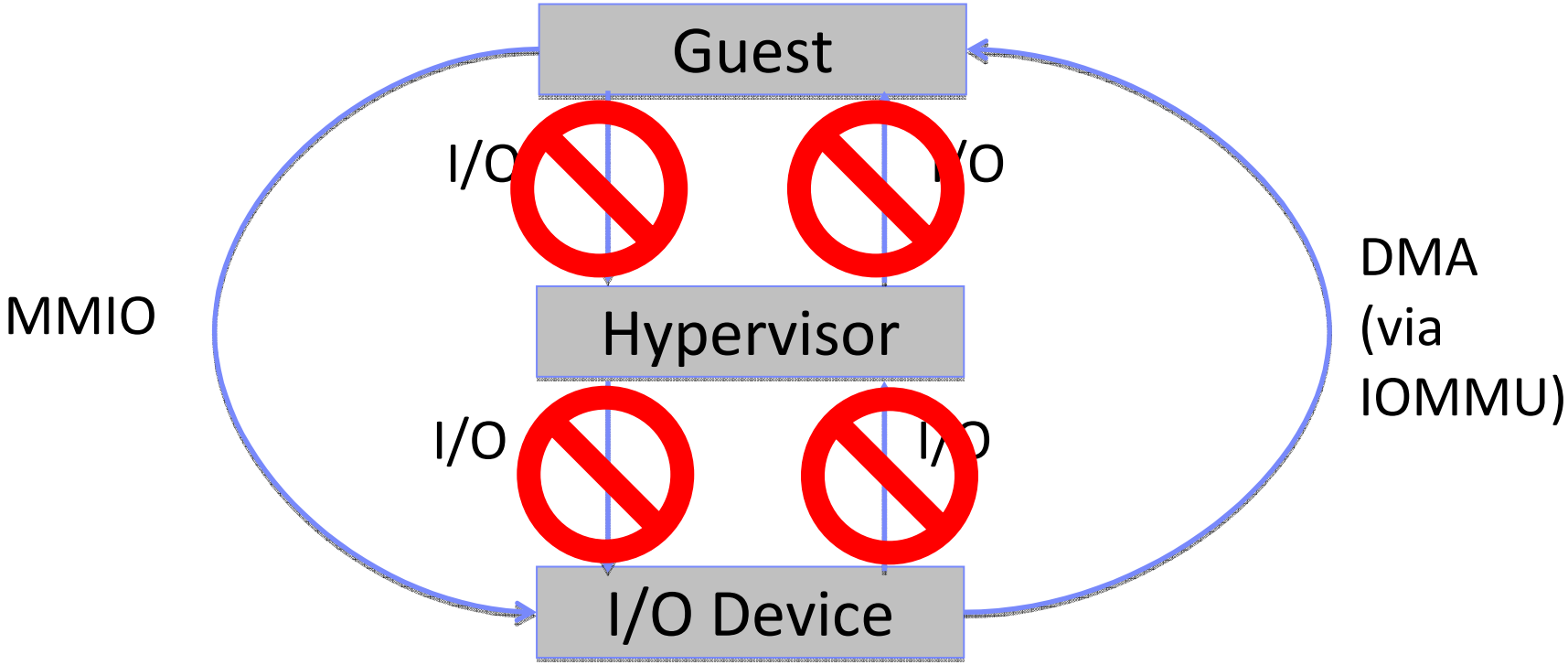


- Privileged instructions trap to the hypervisor
- Traps cause an *Exit* (switch to the hypervisor context)
- Hypervisor emulates their behavior
- Hypervisor resumes the guest (switch to the guest context)

I/O intensive workloads cause many exits = overhead!

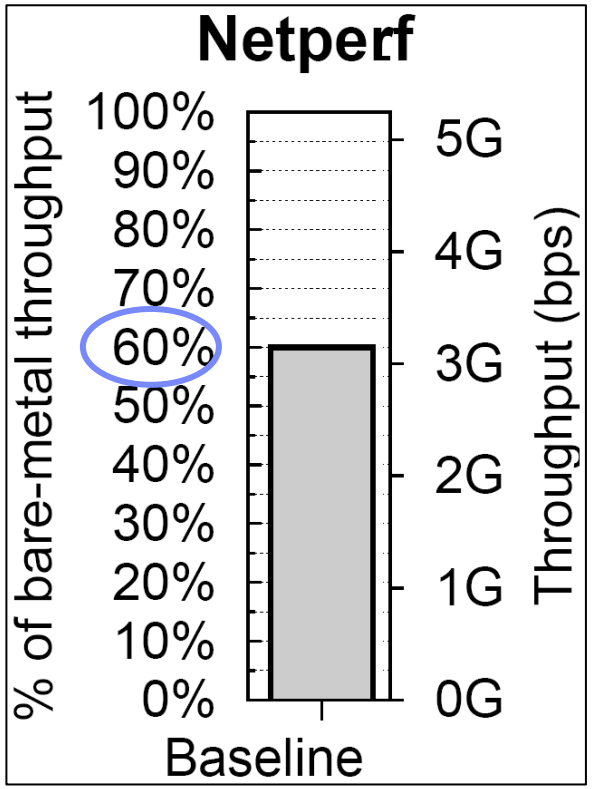
Direct Device Assignment

- Give the guest direct access to a physical device
- Devices nowadays know how to "self virtualize" (SR-IOV)
- Main I/O control and data paths do not require hypervisor intervention

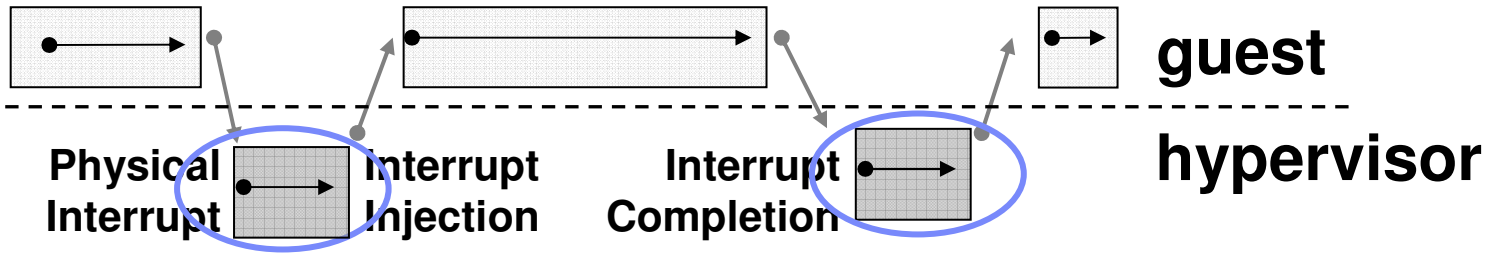


The Solution (?) to I/O Virtualization Overhead

- Device assignment is by far the best performing option, but it only achieves 60% of bare-metal performance

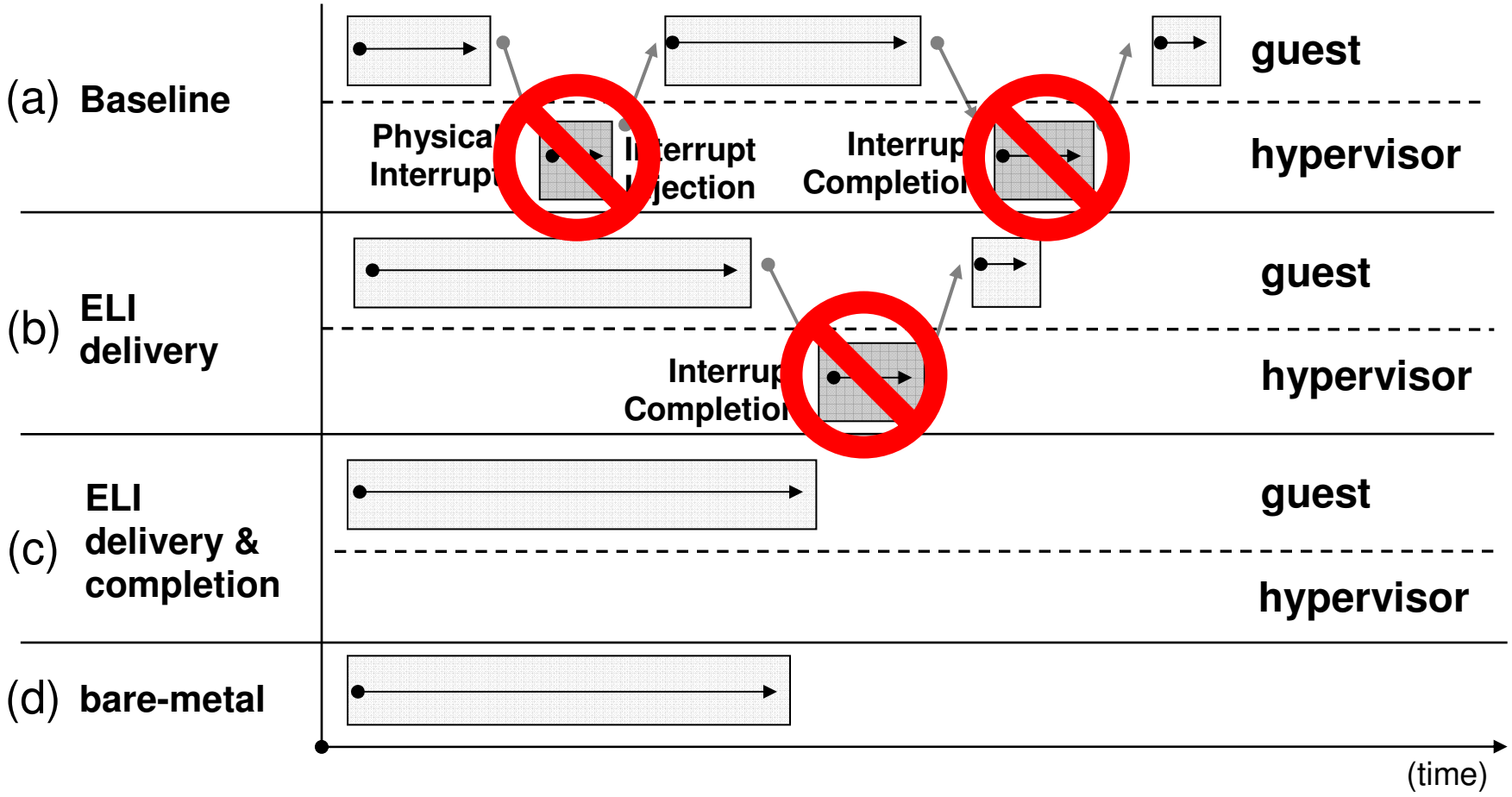


Interrupt, Exits, Interrupt, Exits, Interrupt, Exits...



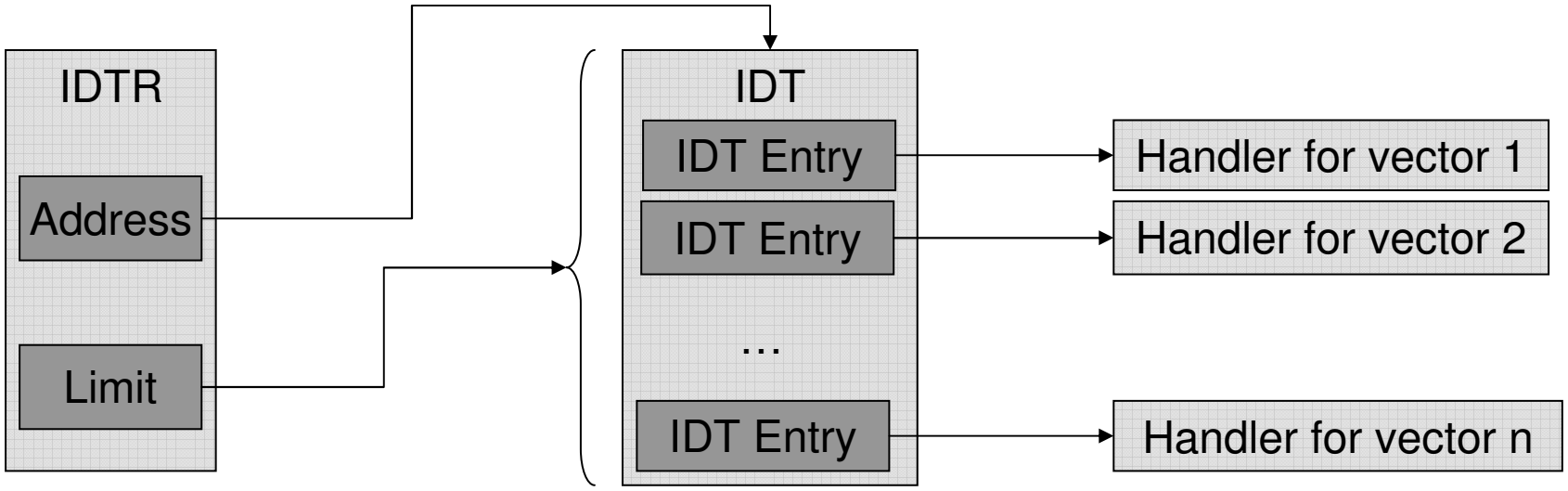
- No hypervisor intervention (?!)… **except for handling more than 48,000 interrupts per second!**
  - Exits due to external interrupts
  - Exits due to interrupt completions

ELI: Exitless Interrupts for unmodified, untrusted guests



Background: x86 Interrupt Handling

- I/O devices raise interrupts to notify the system software about incoming events
- CPU temporarily stops the currently executing code and jumps to a pre-specified interrupt handler

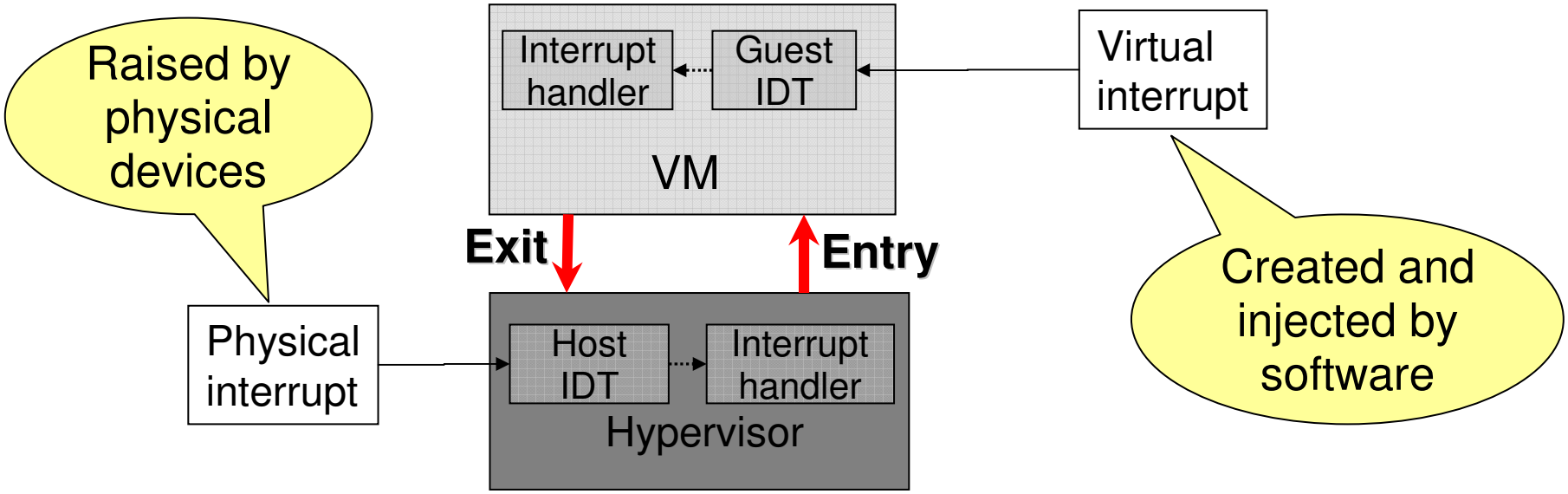


IDT Register

Interrupt **D**escriptor **T**able

Interrupt handlers

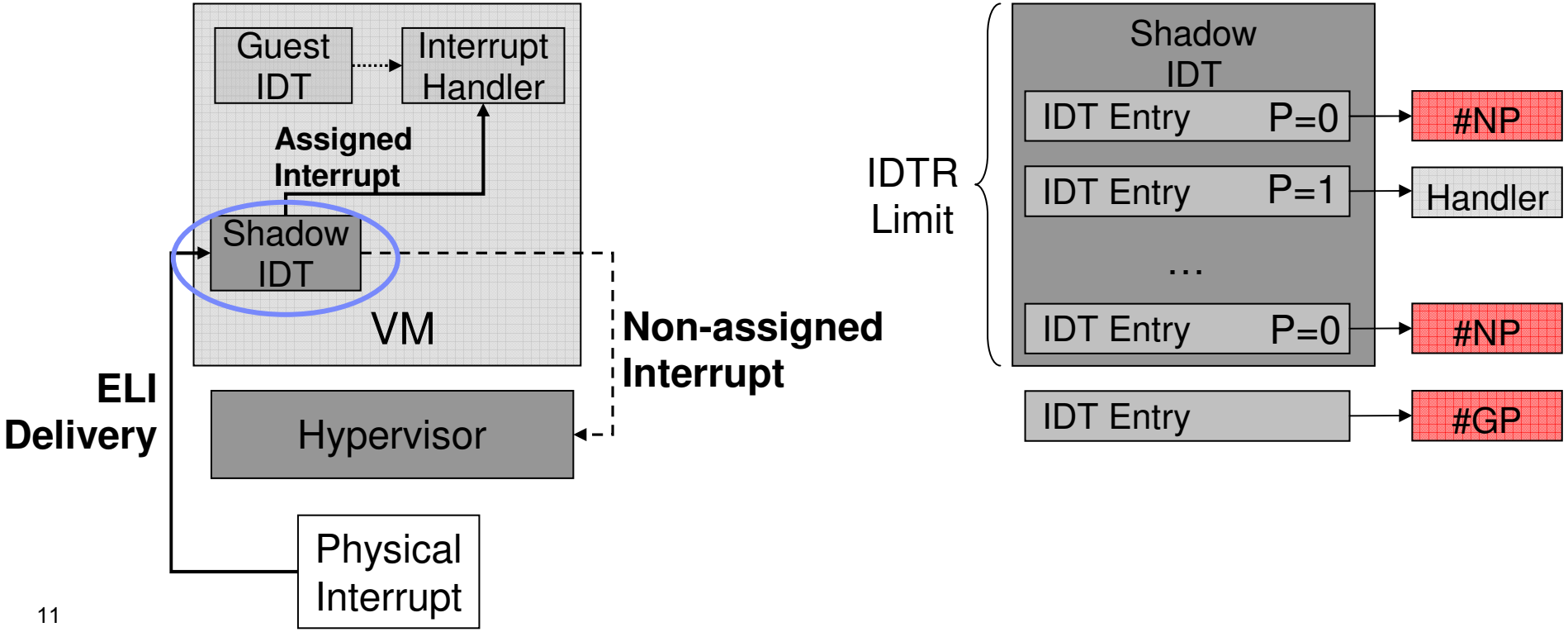
x86 Interrupt Handling in Virtual Environments



- Host IDT: handles physical interrupts
- Guest IDT: handles virtual interrupts
- If a physical interrupt –**including interrupts from assigned devices**– arrives while the guest is running the **CPU forces an Exit!**

ELI: Exitless Interrupts - Delivery

- ELI configures the CPU to deliver all interrupts to the guest
- ELI runs the guest using a shadow IDT
- Host interrupts are bounced back to the host in the form of exceptions
- …without the guest being aware of it



ELI: Exitless Interrupts - Completion

- Guests write to the LAPIC EOI register
- Old LAPIC interface:
  - Hypervisor traps memory accesses → **page granularity**
- New LAPIC interface (x2APIC), required for Exitless Completions
  - Hypervisor traps accesses to MSRs → **register granularity**

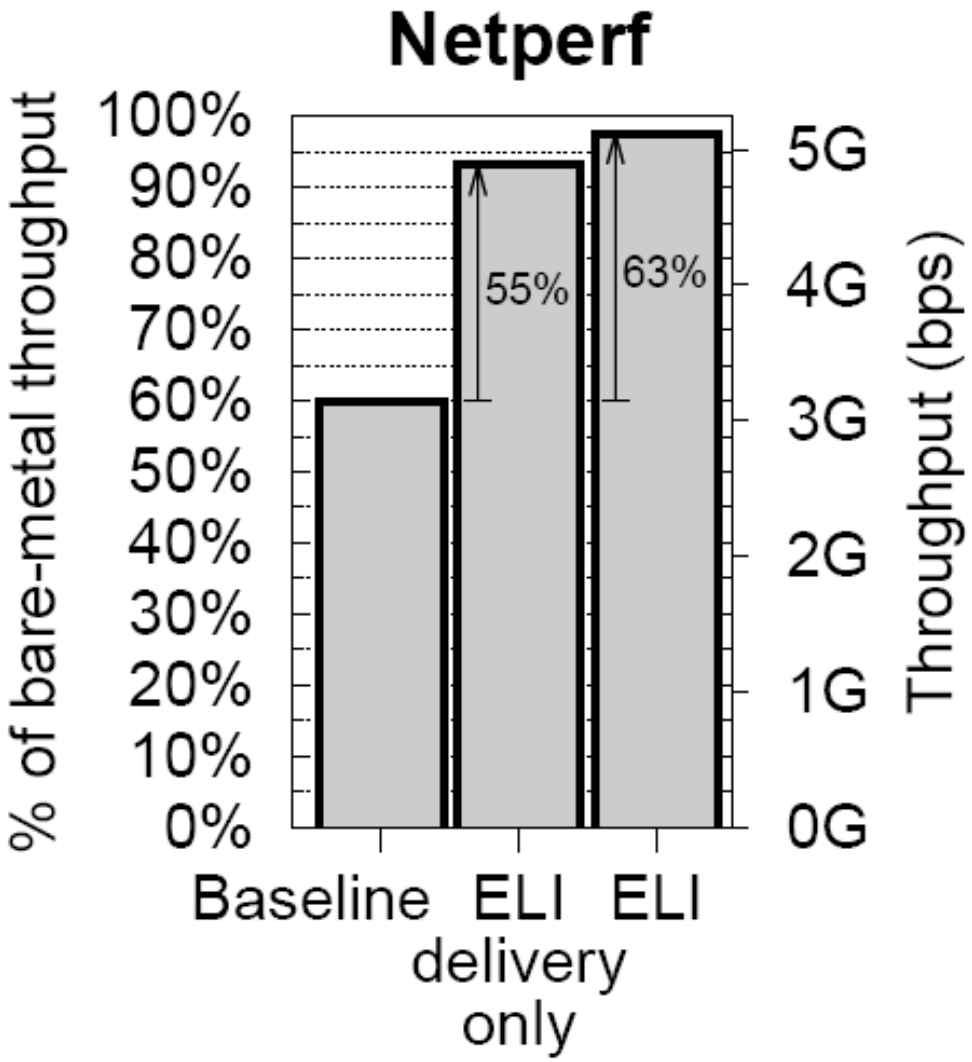
ELI gives direct access only to the EOI register

Evaluation: Netperf

102,000 Exits/sec (+) → (-) **800**

60% Time in guest (-) → (+) **98%**

bare-metal performance!



Evaluation: Apache

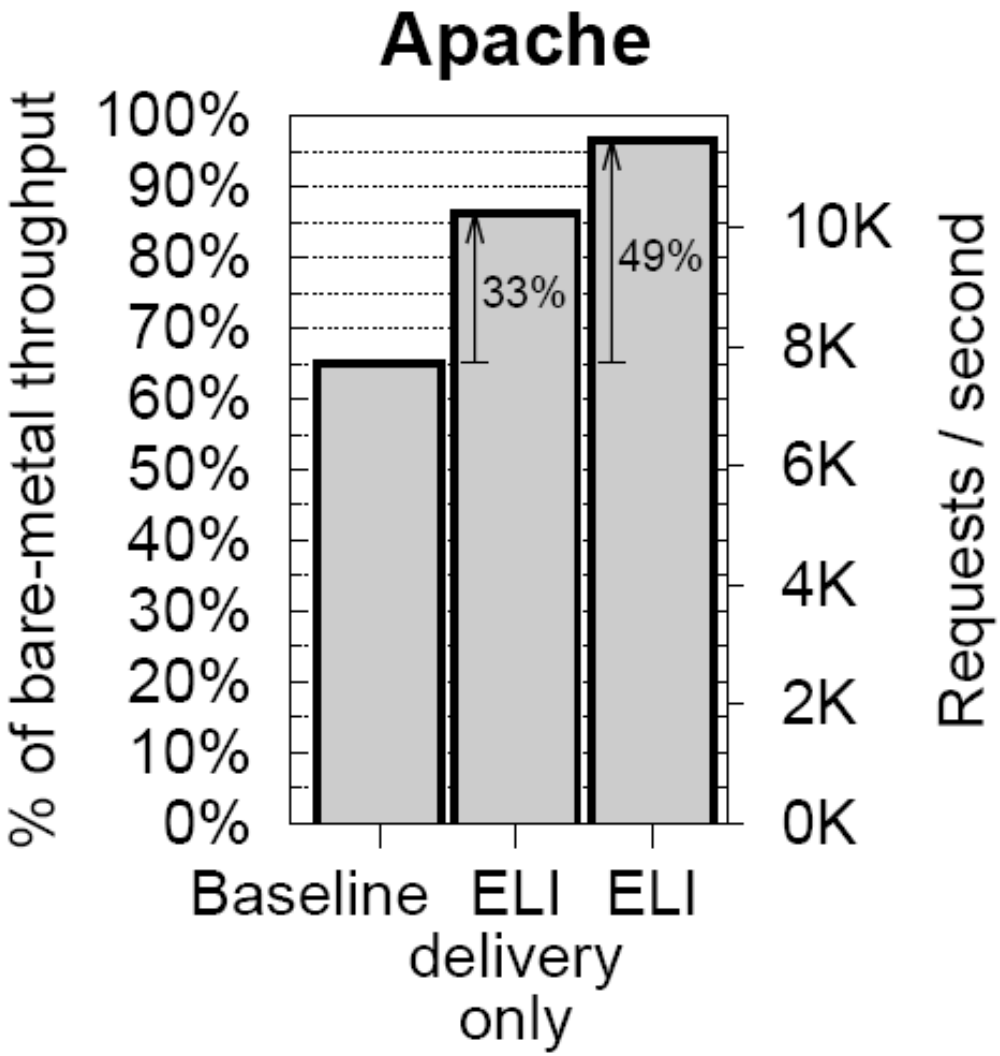
91,000 —(+)→ **1,100** Exits/sec

↓

65% —(-)→ **97%** Time in guest

↓

bare-metal performance!

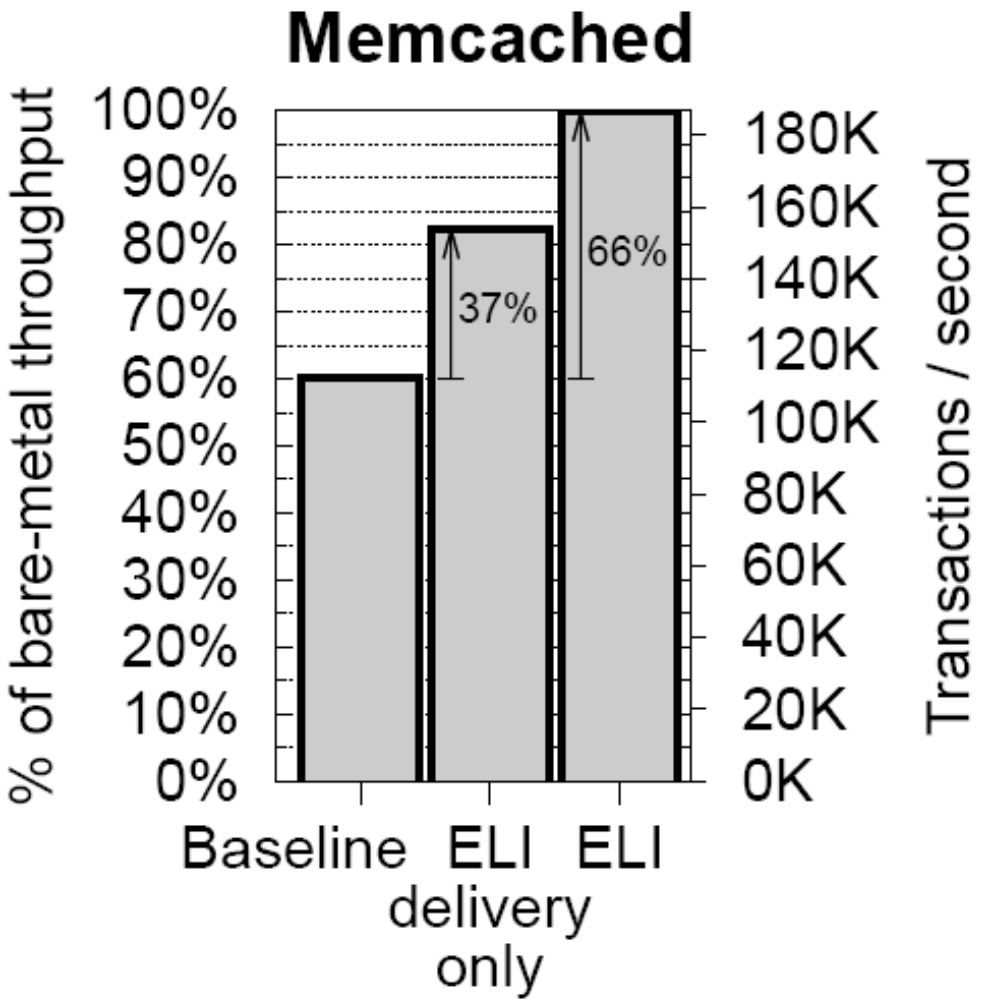


Evaluation Memcached

123,000 —(+ / −)→ 1,000 Exits/sec

60% —(− / +)→ 100% Time in guest

bare-metal performance!



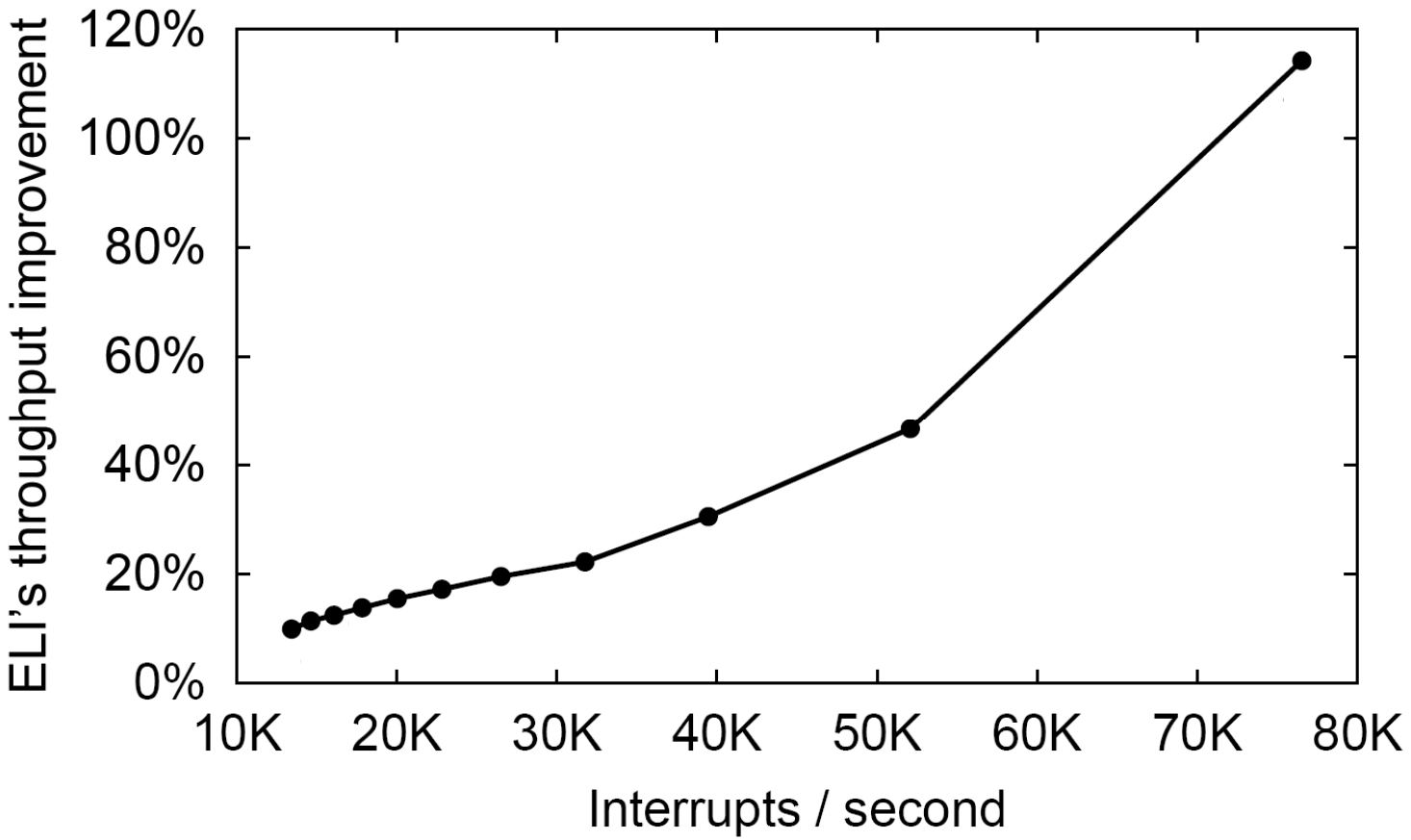
Evaluation: Latency

Configuration	Avg. Latency	% of bare-metal
Baseline	36.14µs	**129%**
ELI delivery-only	30.10µs	108%
ELI	28.51µs	**102%**
Bare-metal	27.93µs	100%

Netperf UDP Request Response

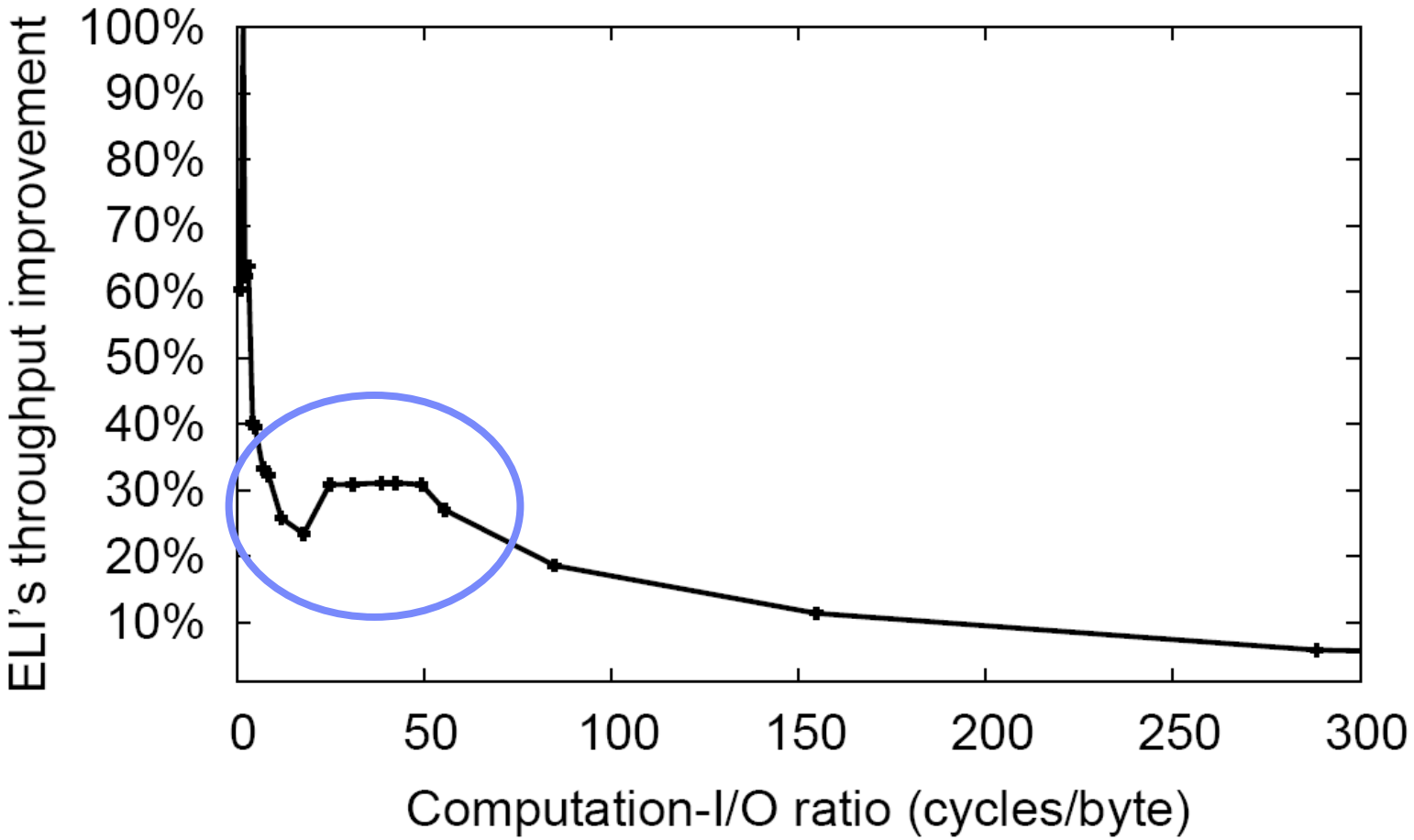
ELI reduces the time it takes to deliver interrupts, critical for latency-sensitive workloads.

Evaluation: Interrupt Coalescing (netperf)

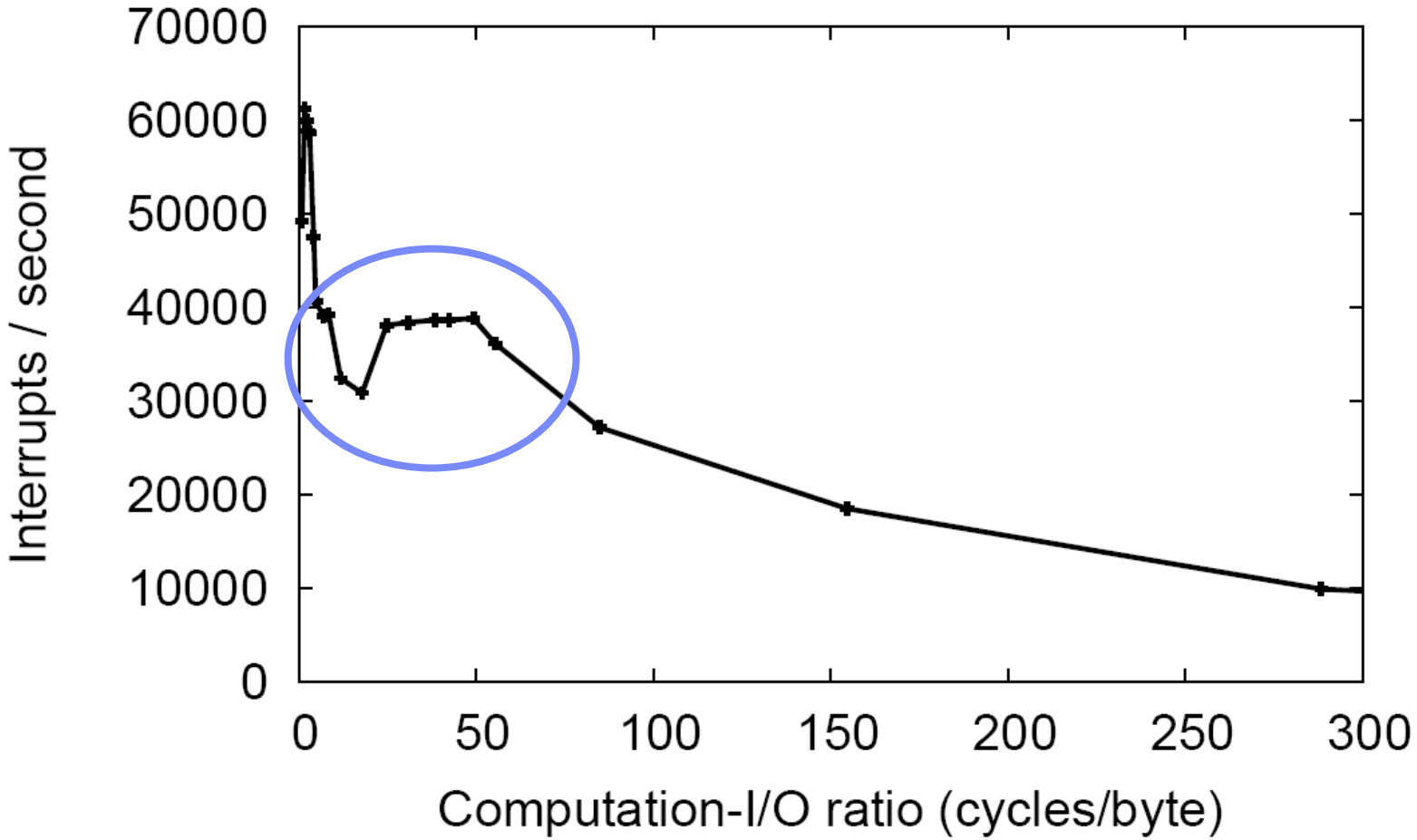


Aggressive Interrupt coalescing → No Interrupt coalescing

Evaluation: Computation vs. I/O Ratio (throughput)



Evaluation: Computation vs. I/O Ratio (interrupt rate)



Mitigating Interrupts Overhead - Related Work

- Polling
- Hybrid
- Interrupt Coalescing

(1) ELI is **complementary** to these approaches. **Removes the virtualization overhead** caused by the costly exits and entries during interrupt handling.

(2) ELI allows the guest to choose the desired method and achieve bare-metal performance in any-case.

Coming Soon….

- Reduce frequency of exits caused by para-virtual I/O devices
- Reduce frequency of exits caused by non-assigned interrupts
- Reduce exits required to inject a virtual interrupt
- Improve performance of SMP workloads with high Inter-processor interrupt (IPI) rate
- Improve performance of Nested Virtualization

Conclusions

- High virtualization performance requires the CPU to spend most of the time running the guest (useful work) and not the host (handling exits = overhead)
- x86 virtualization requires host involvement (exits) to handle interrupts (critical path for I/O workloads)
- ELI lets the guest handle interrupts directly (no exits) and securely, making it possible for untrusted and unmodified guests reach near bare-metal performance

Questions ?