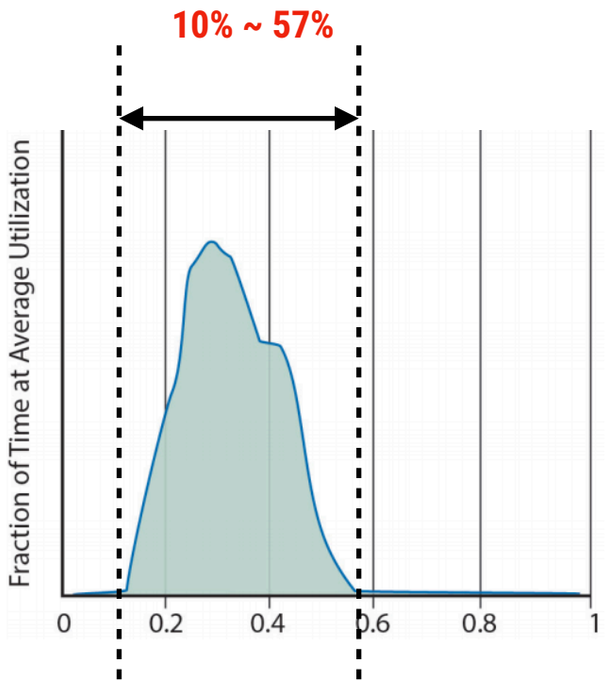
Scheduler Activations for Interference-Resilient SMP Virtual Machine Scheduling

Yong Zhao[1], Kun Suo[1], Luwei Cheng[2], Jia Rao[1]

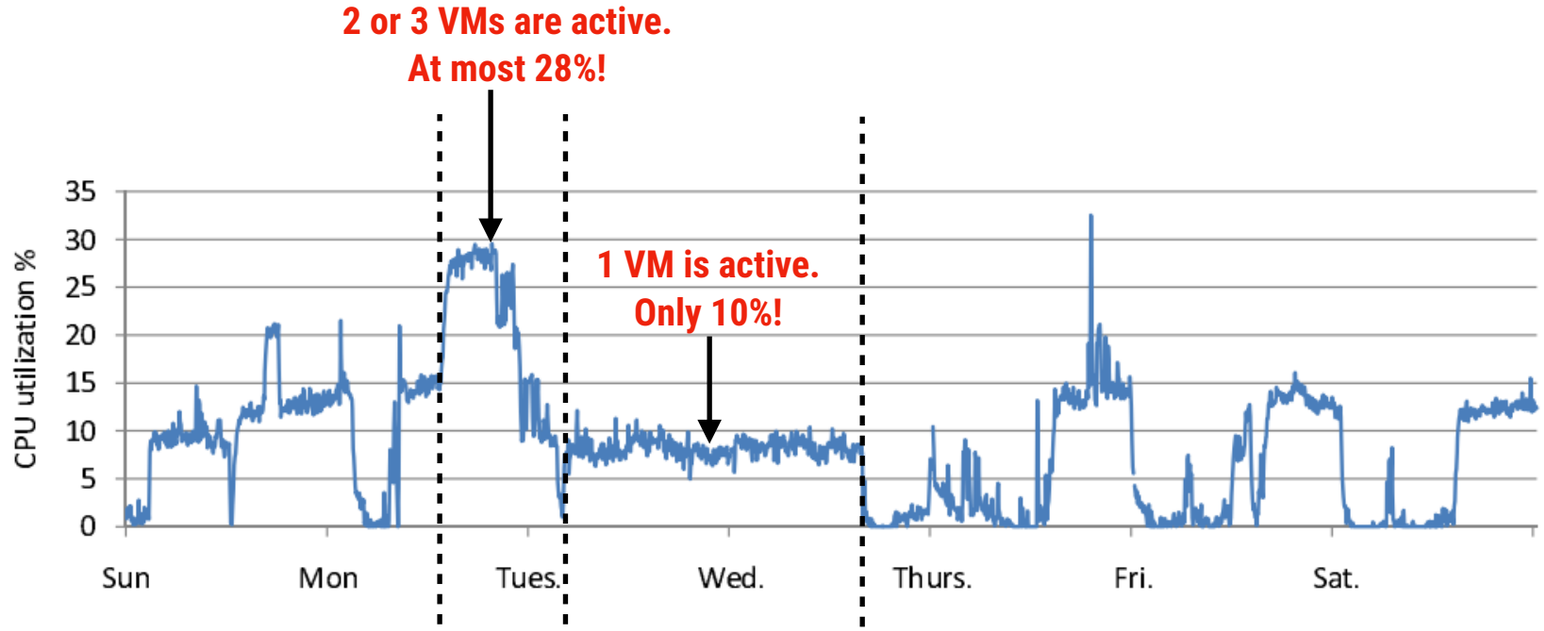
The University of Texas at Arlington[1] Facebook[2]

Middleware 2017

CPU Utilization in the Cloud



Average CPU utilization across over 20000 servers in **Google** data center (January ~ March. 2013).[1]



CPU utilization of a physical server on **Amazon EC2**.[2]

A high consolidation ratio is necessary for improving hardware utilization

1 *An Introduction to the Design of Warehouse-Scale Machines.*

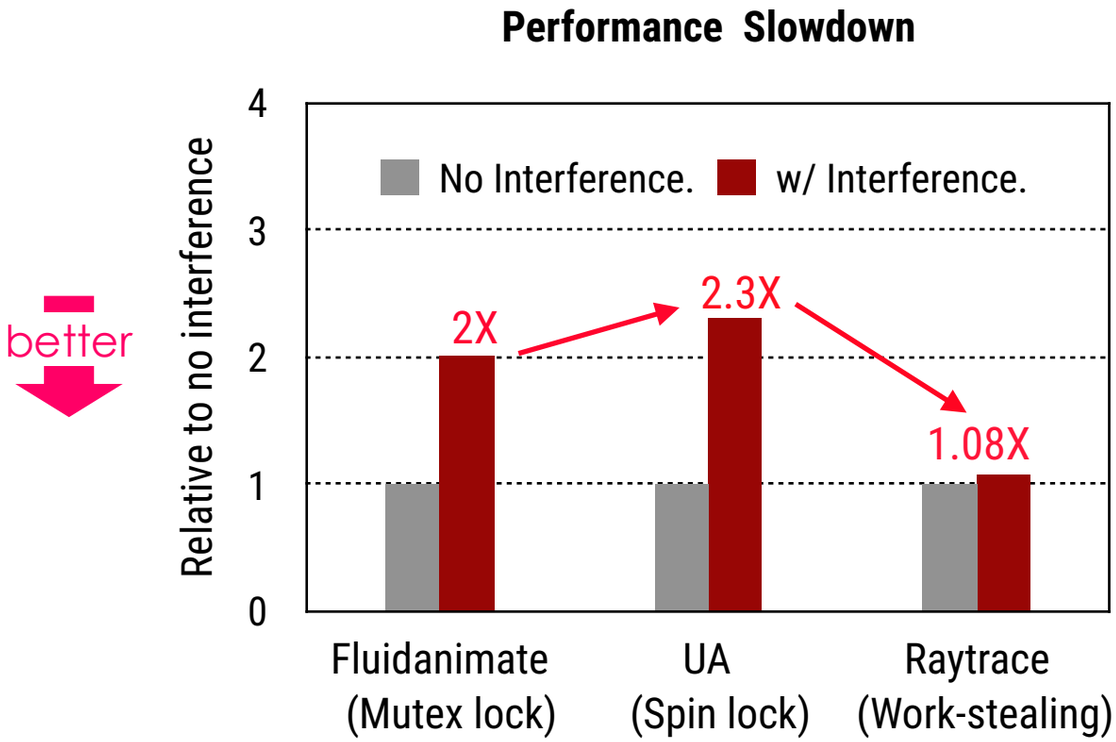
2 *A Measurement Study of Server Utilization in Public Clouds.*

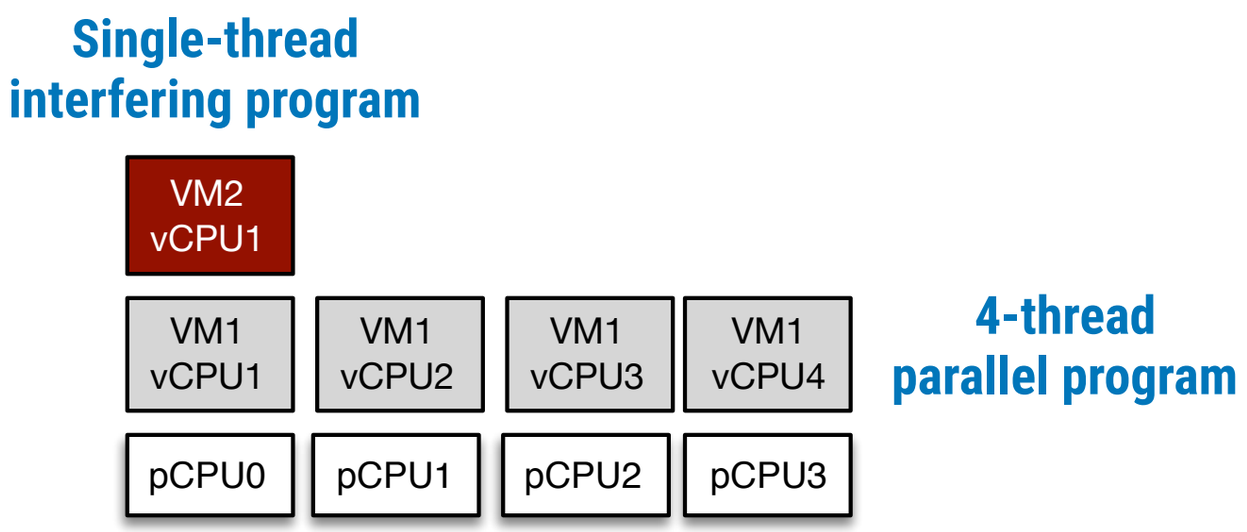
Conservative Consolidation Policy

- Partitions resources rather than sharing resources
  - **No CPU sharing** among **SMP VMs** or **CPU oversubscription** in Amazon EC2, Microsoft Azure, Google Compute Engine, and Alibaba Cloud
- Leads to low resource utilization and high user cost
  - Existing clouds do not have economic competitive advantage compared to DOE centers in scientific computing [Magellan report]
  - **Rule of thumb:** it is more economic to do it in-house rather than on cloud if you can keep a machine more than half busy for more than half of the time [NAG consulting]

Why cloud providers are overly conservative in workload consolidation, especially for SMP VMs?

Performance Loss and Variability

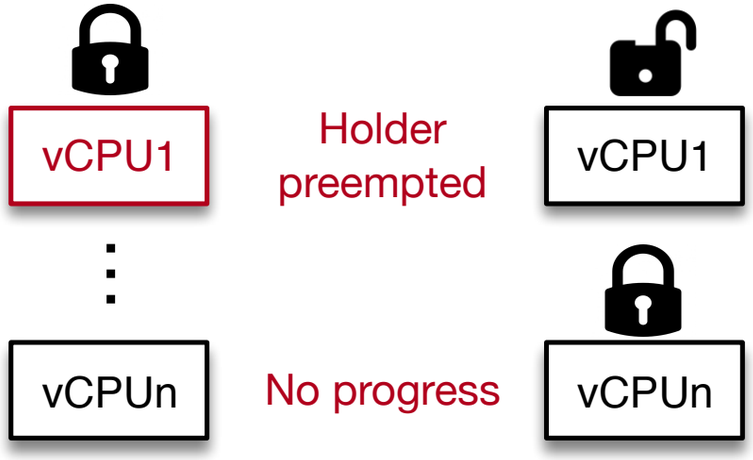




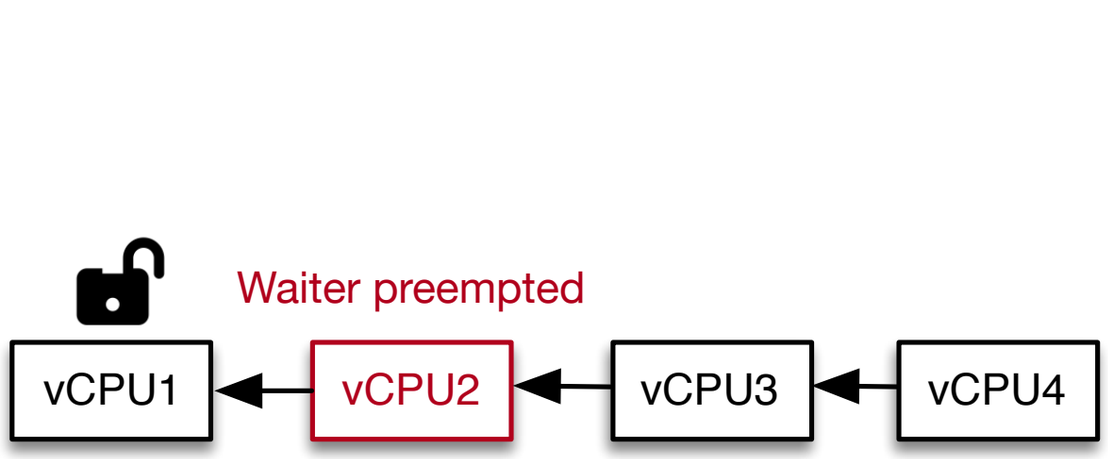
Observations:

- **Significant performance loss, disproportionate to the level of contention**
- **Programs respond differently to interference**

LHP and LWP



Lock-holder preemption (LHP)



Lock-waiter preemption (LWP)

The delay of one vCPU affects the progress of all other vCPUs, thereby slowing down the entire program

Existing Efforts

- Hypervisor Level Approaches
  - Co-scheduling, relaxed co-scheduling **[VMware'10]**
  - Balance scheduling **[Sukwong-EuroSys'11]**
  - Demanded-based coordinated scheduling **[Kim-ALPLOS'13]**

- Guest OS-Assisted Approaches
  - Dynamic adaptive scheduling **[Weng-HPDC'10]**
  - Delay scheduling **[Uhlig-VM'04]**

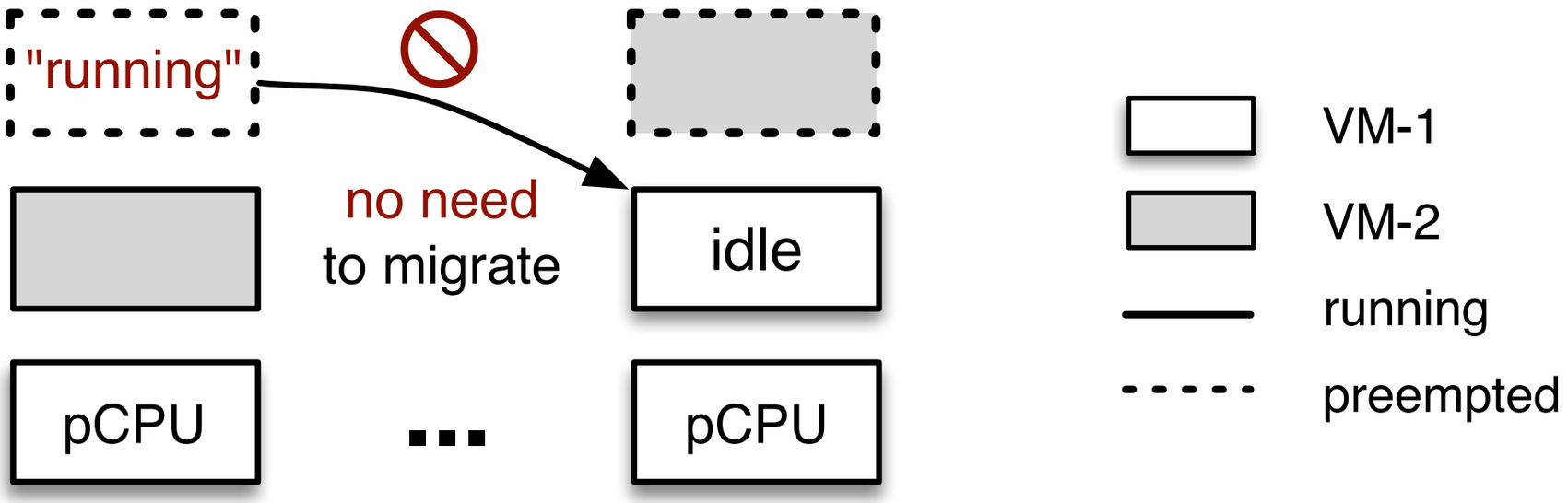
- Hardware-Assisted Approaches
  - Intel Pause-Loop Exiting (PLE) **[Riel'11]**

The Unexploited Potential of the Guest OS

Sibling vCPUs block/idle if LHP or LWP occurs

If the guest OS can balance load among running and preempted vCPUs and timely schedule critical threads, any application can be made resilient to interference

A Hidden Semantic Gap



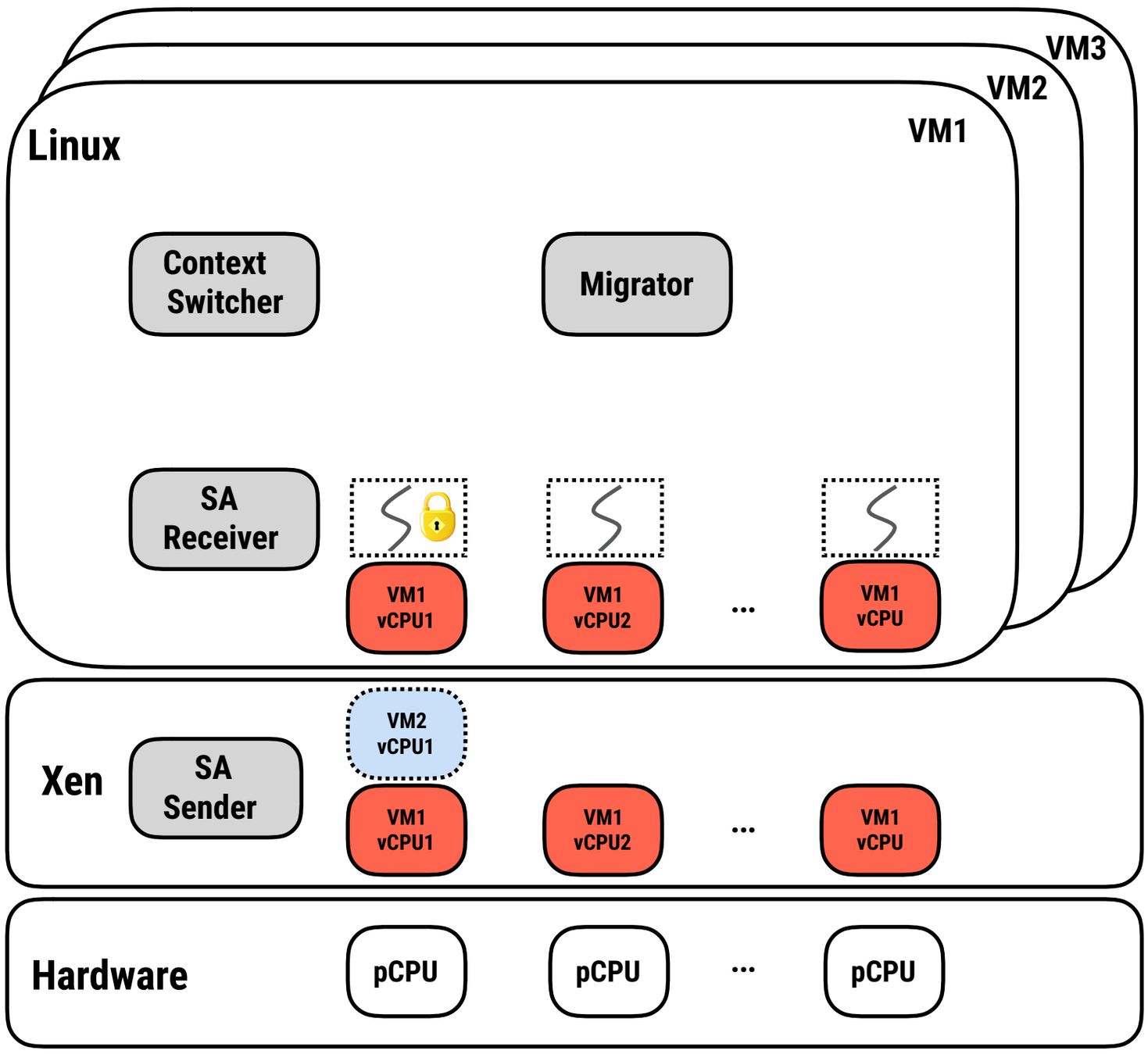
The semantic gap: the guest OS is unaware of the scheduling activities at the hypervisor.

To the guest OS, a critical thread on a **preempted** vCPU still appears to be **"running"**. Therefore, it thinks that there is **no need** to migrate it.

Interference-Resilient Scheduling (IRS)

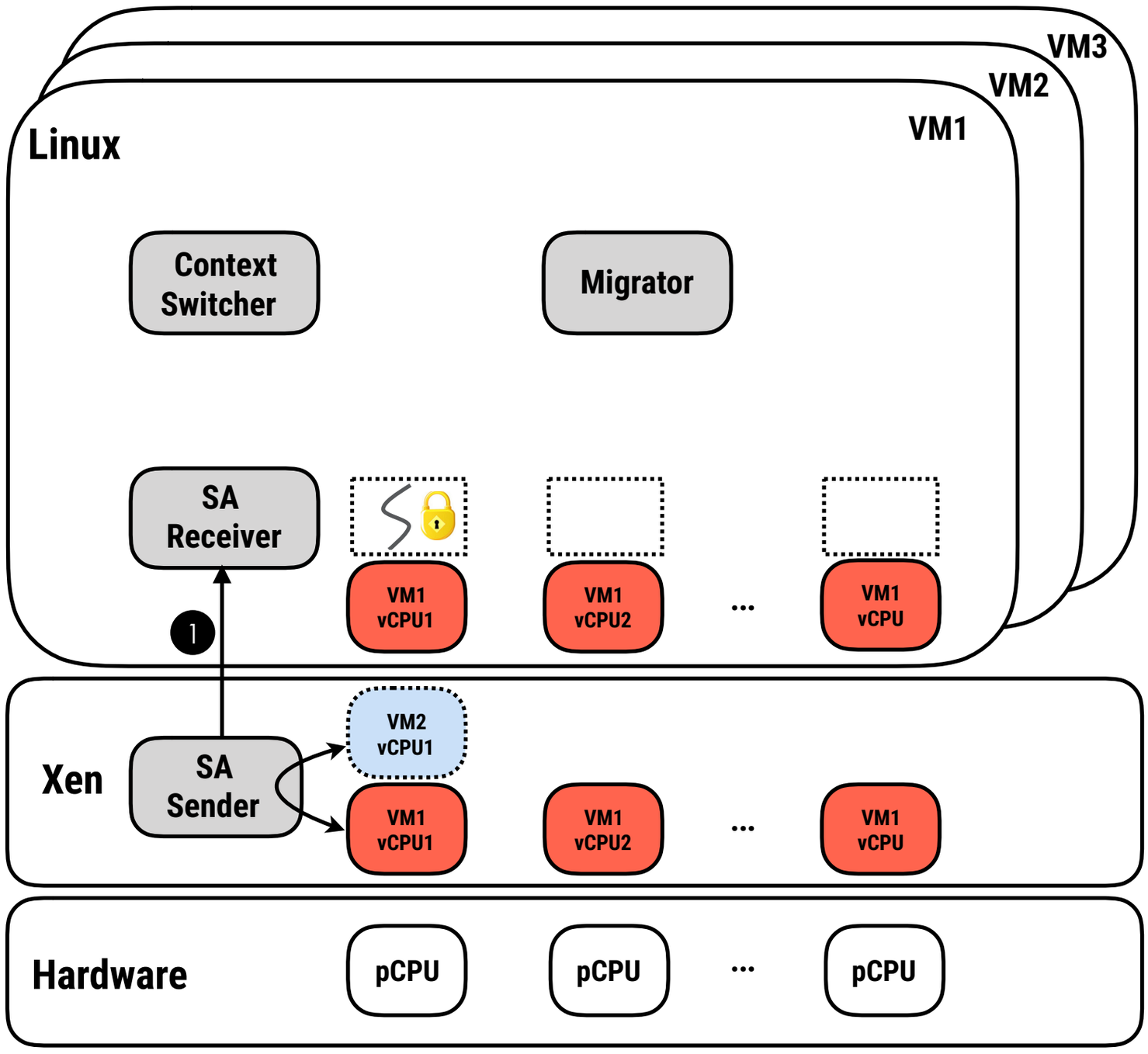
- **Idea**
  - before a vCPU is preempted, the guest OS migrates the critical thread on this vCPU to another running vCPU to avoid LHP or LWP
- **Motivation & objective**
  - Inspired by ***scheduler activation* (SA)** [Anderson TOCS'92] in hybrid threading
  - Minimize interference-induced idling and CPU waste
- **Results**
  - IRS outperformed state-of-the-art relaxed co-scheduling and PLE approaches
  - IRS mitigated CPU stacking

IRS Design


VM3
VM2
VM1
Linux
Context Switcher
Migrator
SA Receiver
VM1 vCPU1
VM1 vCPU2
...
VM1 vCPU
Xen
SA Sender
VM2 vCPU1
VM1 vCPU1
VM1 vCPU2
...
VM1 vCPU
Hardware
pCPU
pCPU
...
pCPU

IRS Design

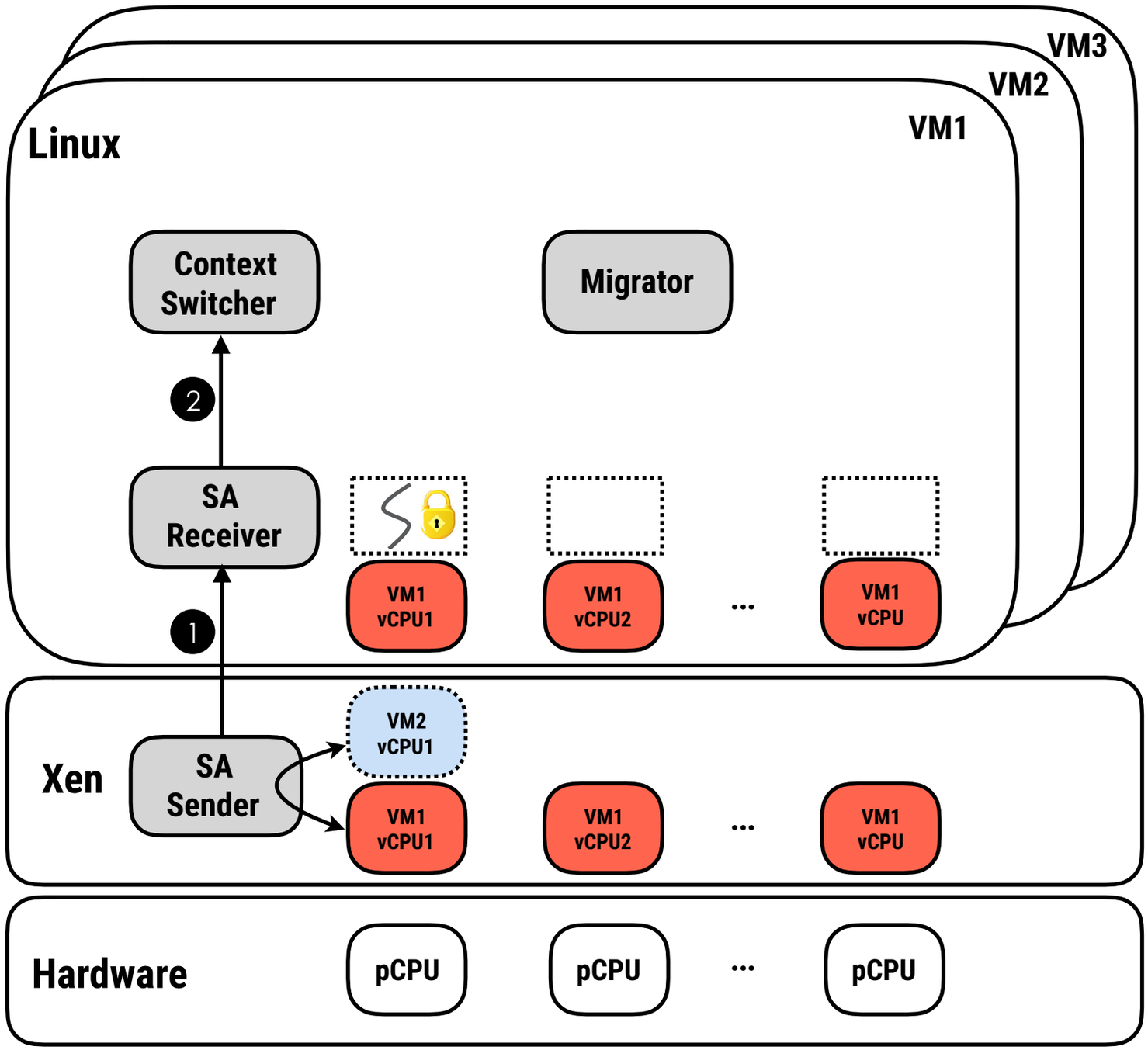
1. Before Xen preempts a vCPU, it sends a notification to the guest OS via SA sender



IRS Design

1. Before Xen preempts a vCPU, it sends a notification to the guest OS via SA sender

2. Upon receiving the notification, the SA receiver in the guest activates load balancing

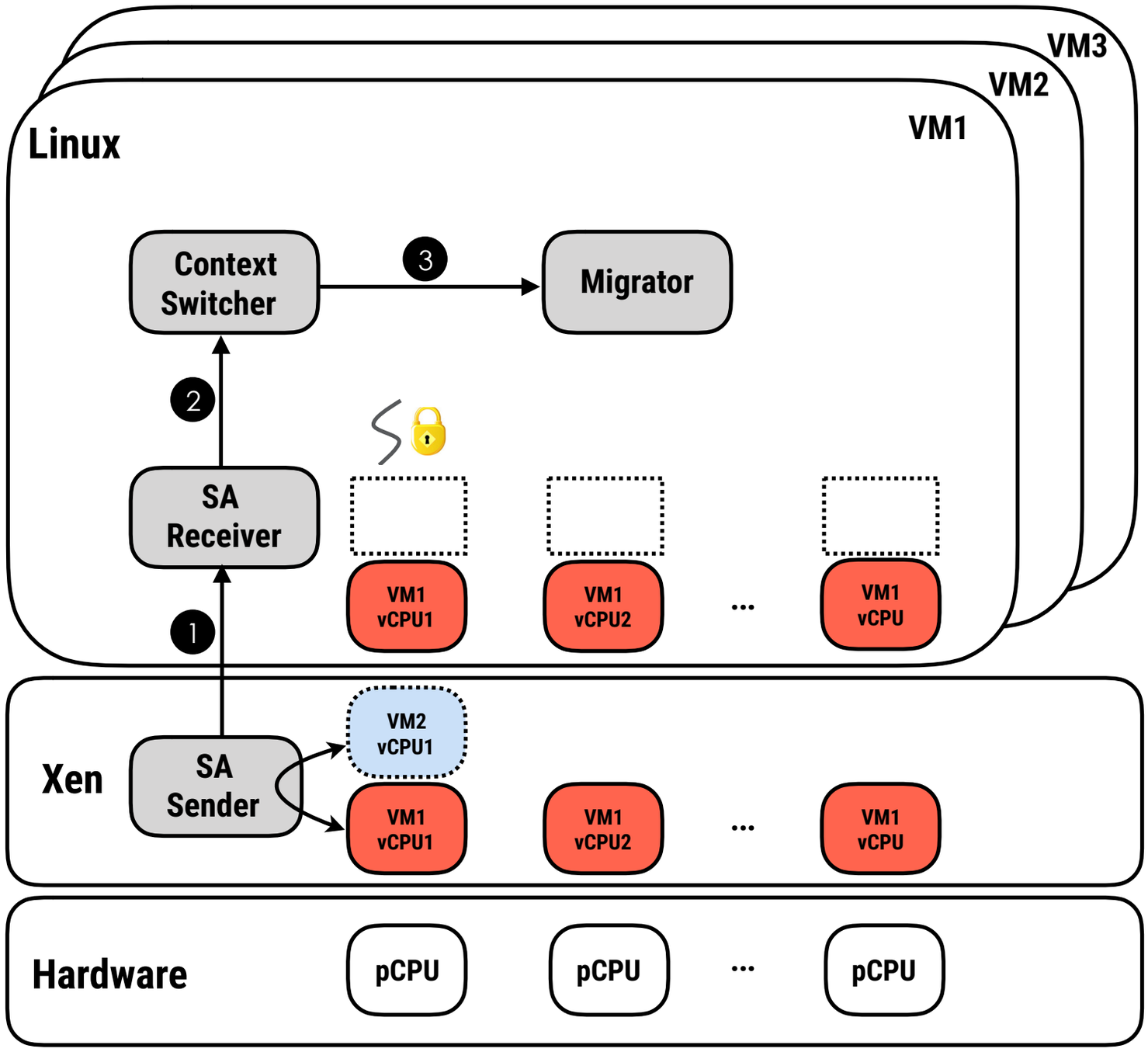


IRS Design

1. Before Xen preempts a vCPU, it sends a notification to the guest OS via SA sender

2. Upon receiving the notification, the SA receiver in the guest activates load balancing

3. CS deschedules the thread on the to-be-preempted vCPU and marks it as migrating



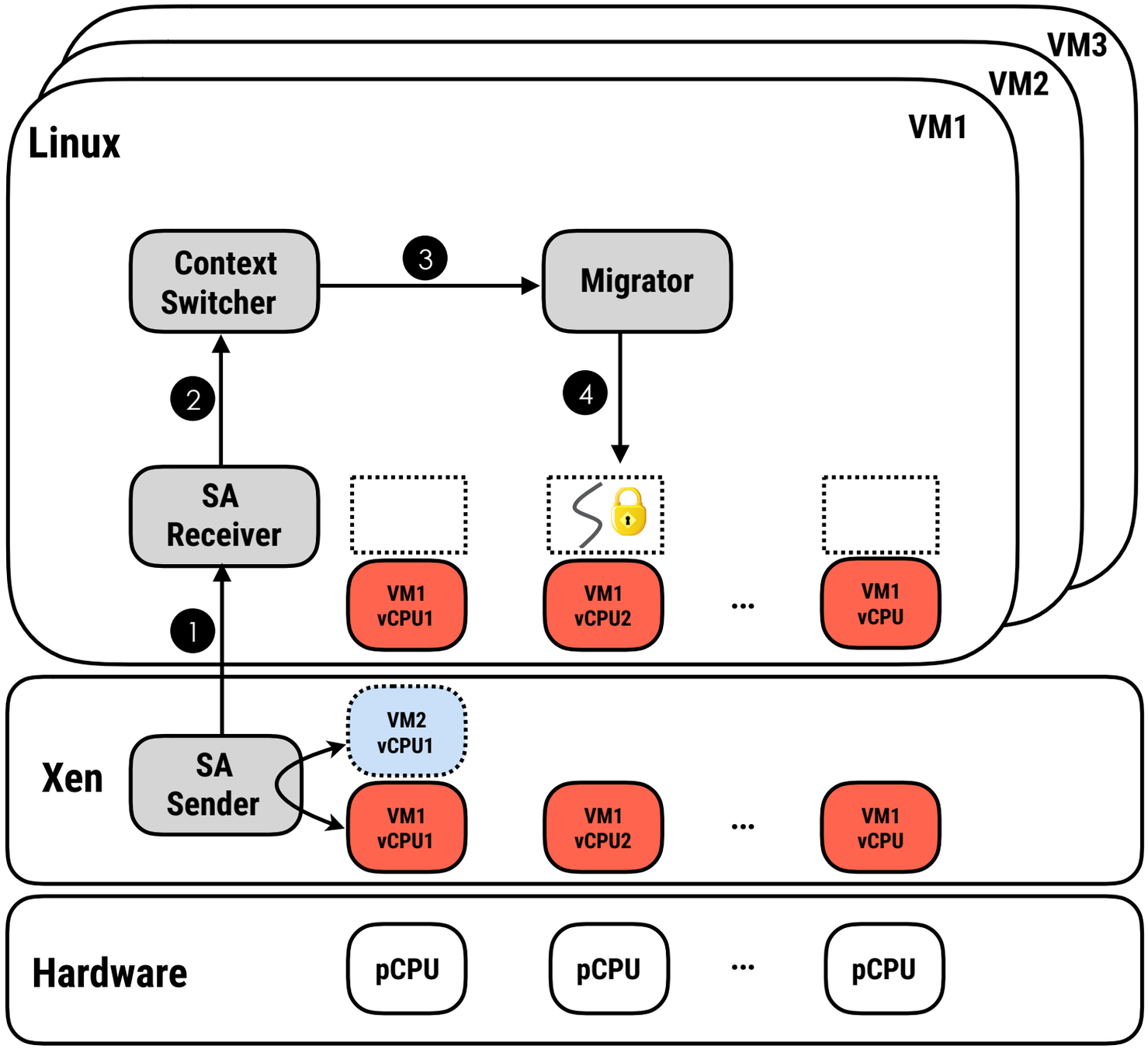
IRS Design

1. Before Xen preempts a vCPU, it sends a notification to the guest OS via SA sender

2. Upon receiving the notification, the SA receiver in the guest activates load balancing

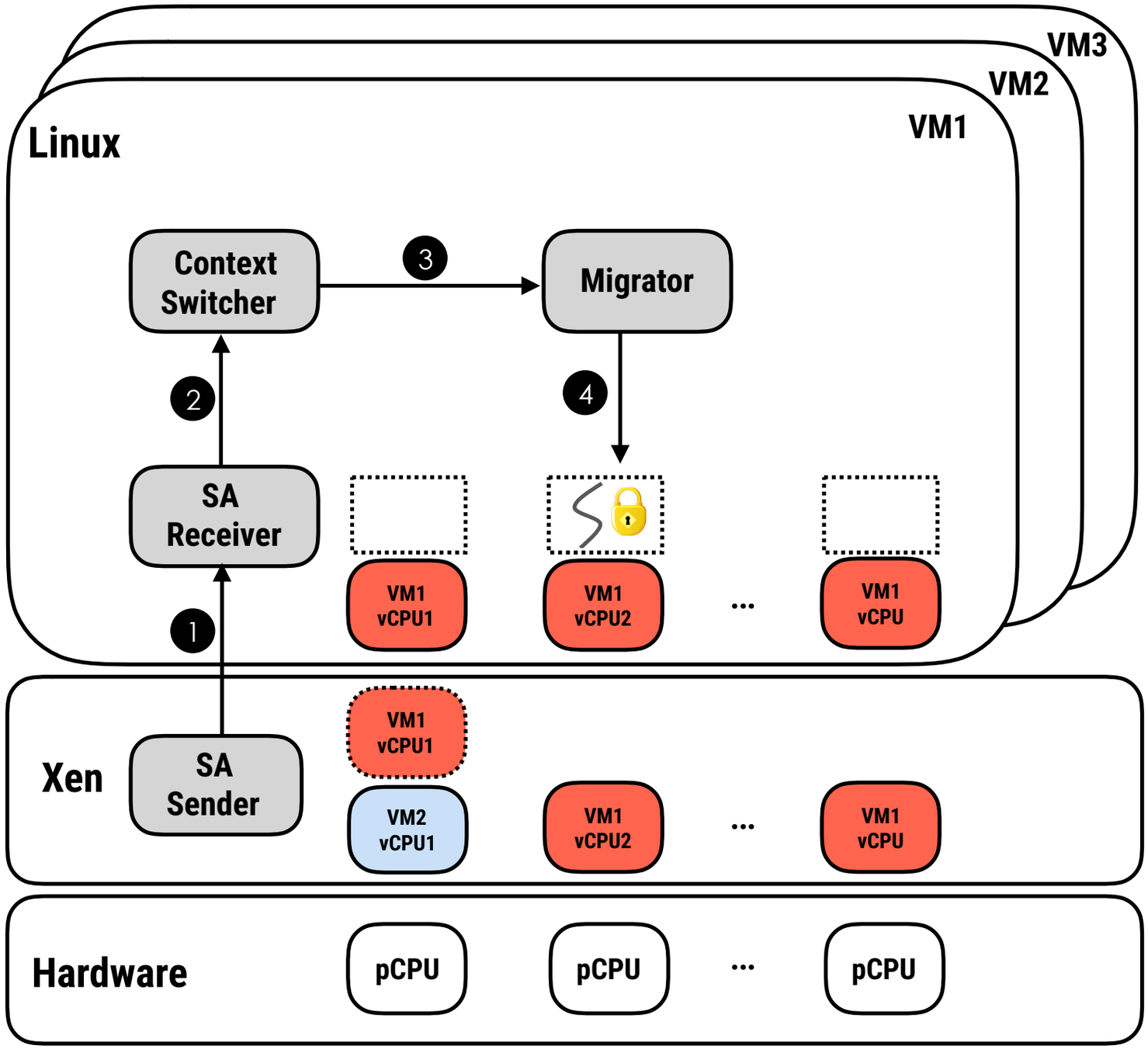
3. CS deschedules the thread on the to-be-preempted vCPU and marks it as migrating

4. Migrator moves the thread to a sibling vCPU with the least waiting time



IRS Design

1. Before Xen preempts a vCPU, it sends a notification to the guest OS via SA sender
2. Upon receiving the notification, the SA receiver in the guest activates load balancing
3. CS deschedules the thread on the to-be-preempted vCPU and marks it as migrating
4. Migrator moves the thread to a sibling vCPU with the least waiting time
5. After the thread if migrated, Xen finishes vCPU switching



Optimizations & Practical Considerations

- **Preserving data locality**
  - Prevent ping-pong migration by preferably moving threads that were migrated from preempted vCPUs since their locality is already lost
  - Timely scheduling **>** data locality
- **Preventing security exploit**
  - Set an upper bound of 1ms at the hypervisor for SA completion

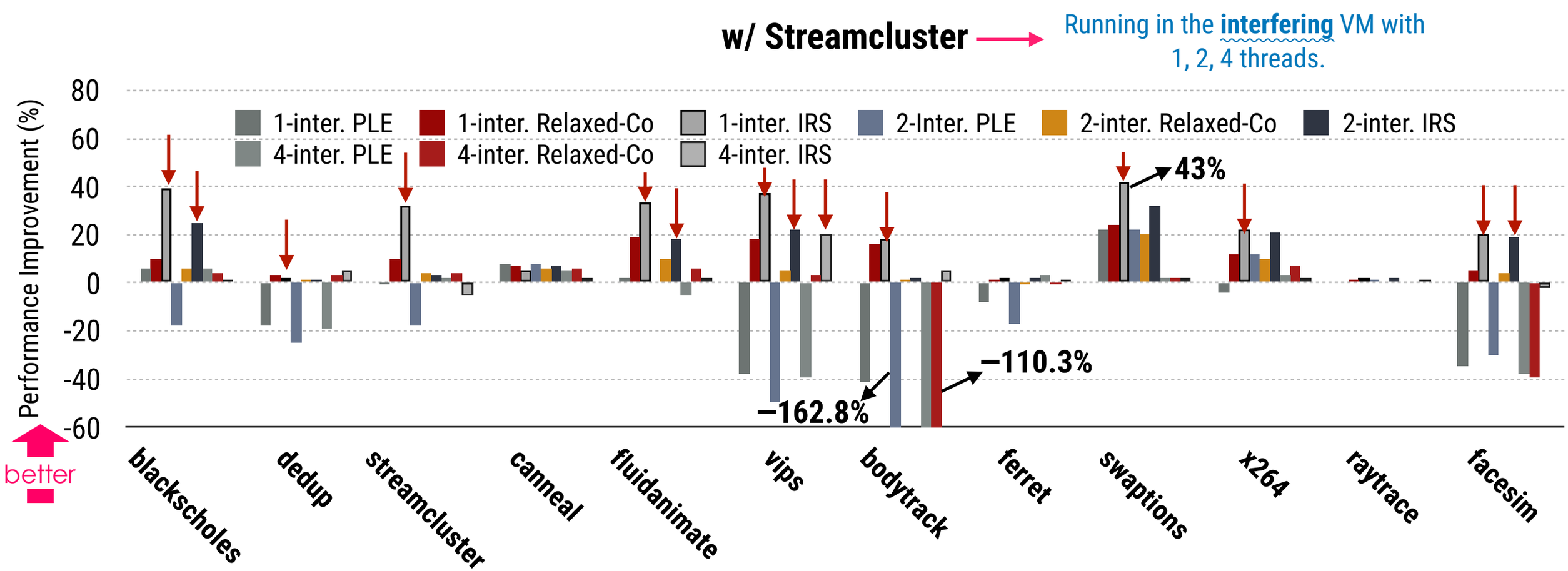
Limitations

- **Inability to eliminate all idle or wasted CPU time**
  - Load estimate is not always accurate
  - Proactively migrating the critical thread from to-be-preempted vCPU does not guarantee an optimal placement of the thread
- **(slightly) undermined fairness**
  - Rogue users can exploit the 1ms SA completion upper bound to gain an additional 1ms time slice
- **The delay of vCPU preemption can hurt I/O latency**

Evaluation

- **Baseline systems**
  - Vanilla Xen 4.5.0
  - VMware relaxed co-scheduling (Relaxed-co)
  - Intel Pause-Loop Exiting (PLE)
- **Benchmarks**
  - PARSEC benchmarks (gcc-pthreads+ blocking sync + native input)
  - NASA parallel benchmarks (spin sync + class C)
  - Apache HTTP benchmarks
  - SPECjbb 2005
- **Experiments**
  - Controlled experiments: vCPUs pinned to pCPUs, increasing level of interference
  - Realistic experiments: vCPUs free to run any pCPUs

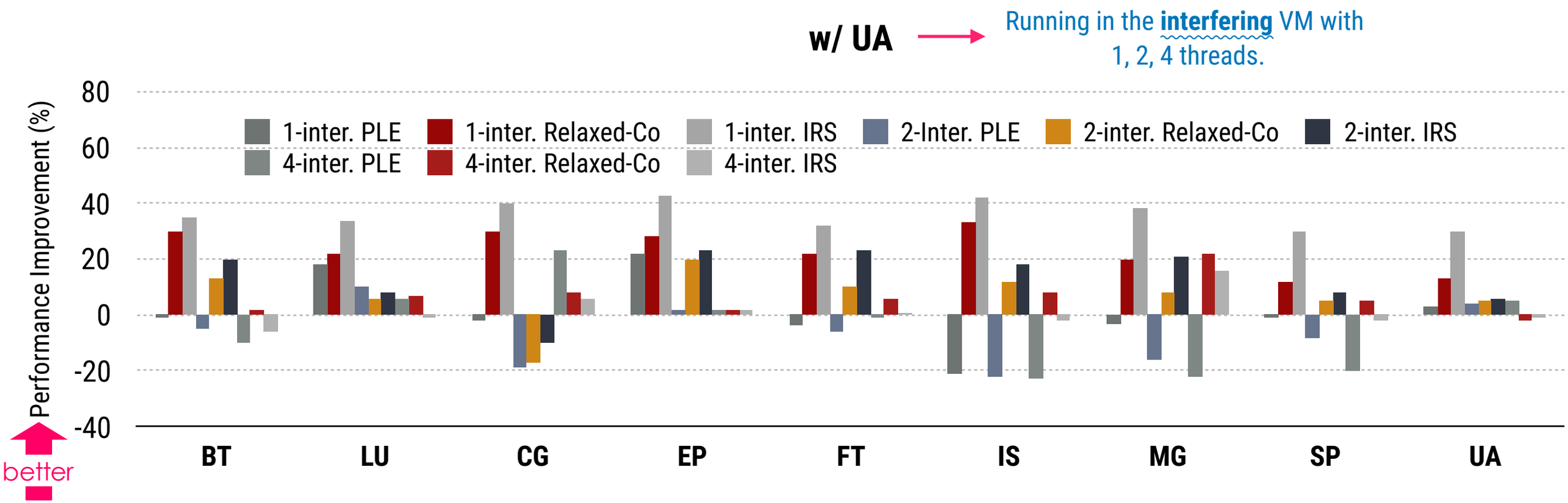
PARSEC benchmarks (blocking)



Running in the **foreground** VM with 4 threads in 4 vCPUs

- **IRS improved performance compared to Xen**
- **IRS consistently outperformed co-scheduling and PLE**
- **IRS had diminishing gain as interference ramped up**

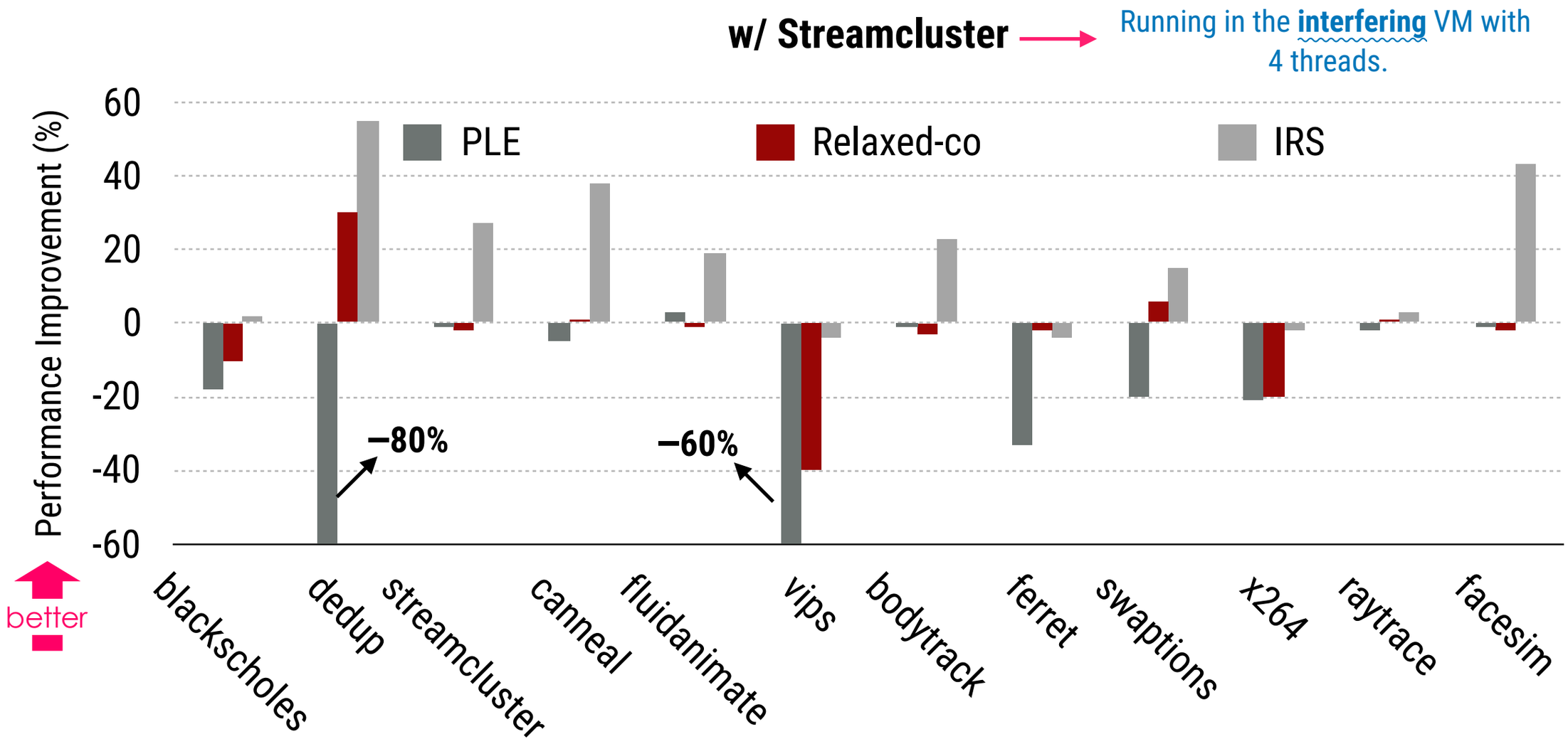
NASA parallel benchmarks (spinning)



Running in the **foreground** VM with 4 threads in 4 vCPUs

- **IRS effectively reduces futile spinning**
- **IRS consistently outperformed co-scheduling and PLE**
- **IRS had diminishing gain as interference ramped up**

Mitigating CPU Stacking



Running in the **foreground** VM with 4 threads in 4 vCPUs

- **IRS greatly mitigated CPU stacking**
- **Co-scheduling and PLE incurred more serious CPU stacking compared to vanilla Xen**

Conclusion

- **Interference Resilient Scheduling (IRS):** a coordinated approach that bridges the guest-hypervisor semantic gap at the Guest OS side.

  - Inspired by Scheduler Activation (SA) [Anderson TOCS'92]

  - Enhances Guest OS load balancing to make any parallel applications resilient to interference

  - Mitigates the LHP and LWP problems

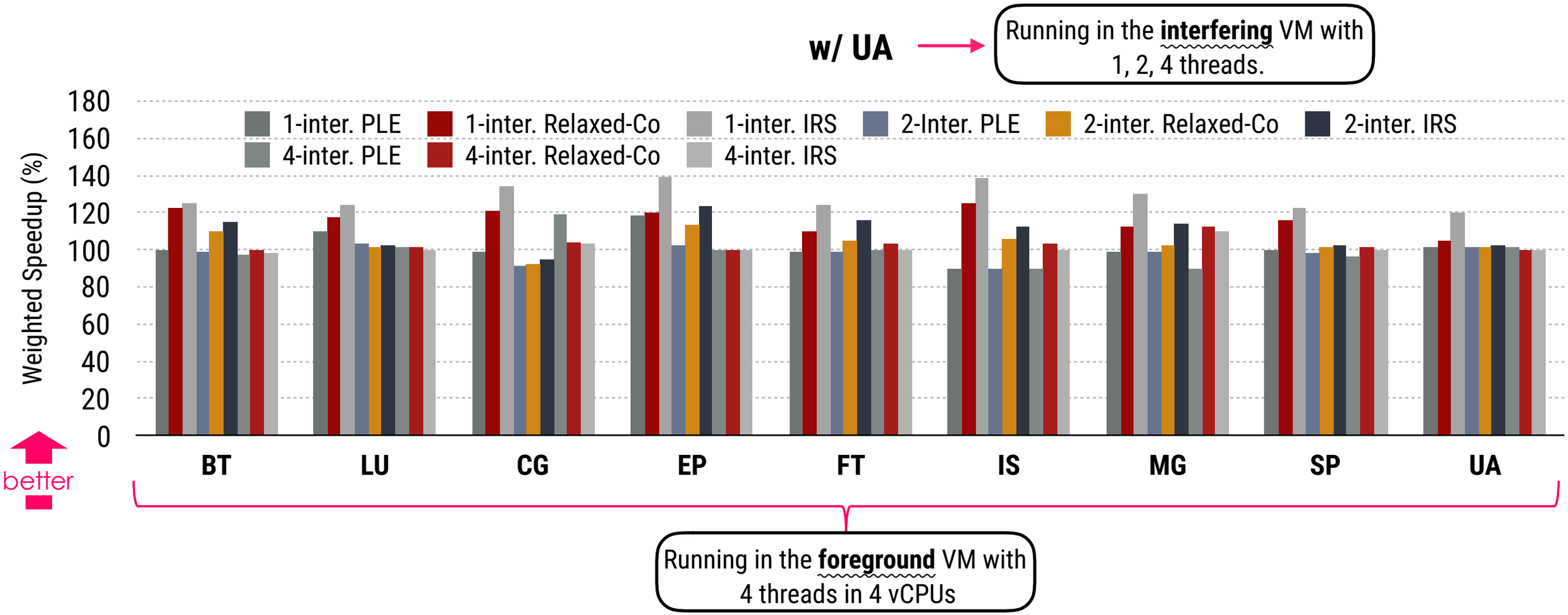
  - Alleviates the CPU stacking problem

  - Outperforms relaxed co-scheduling and PLE

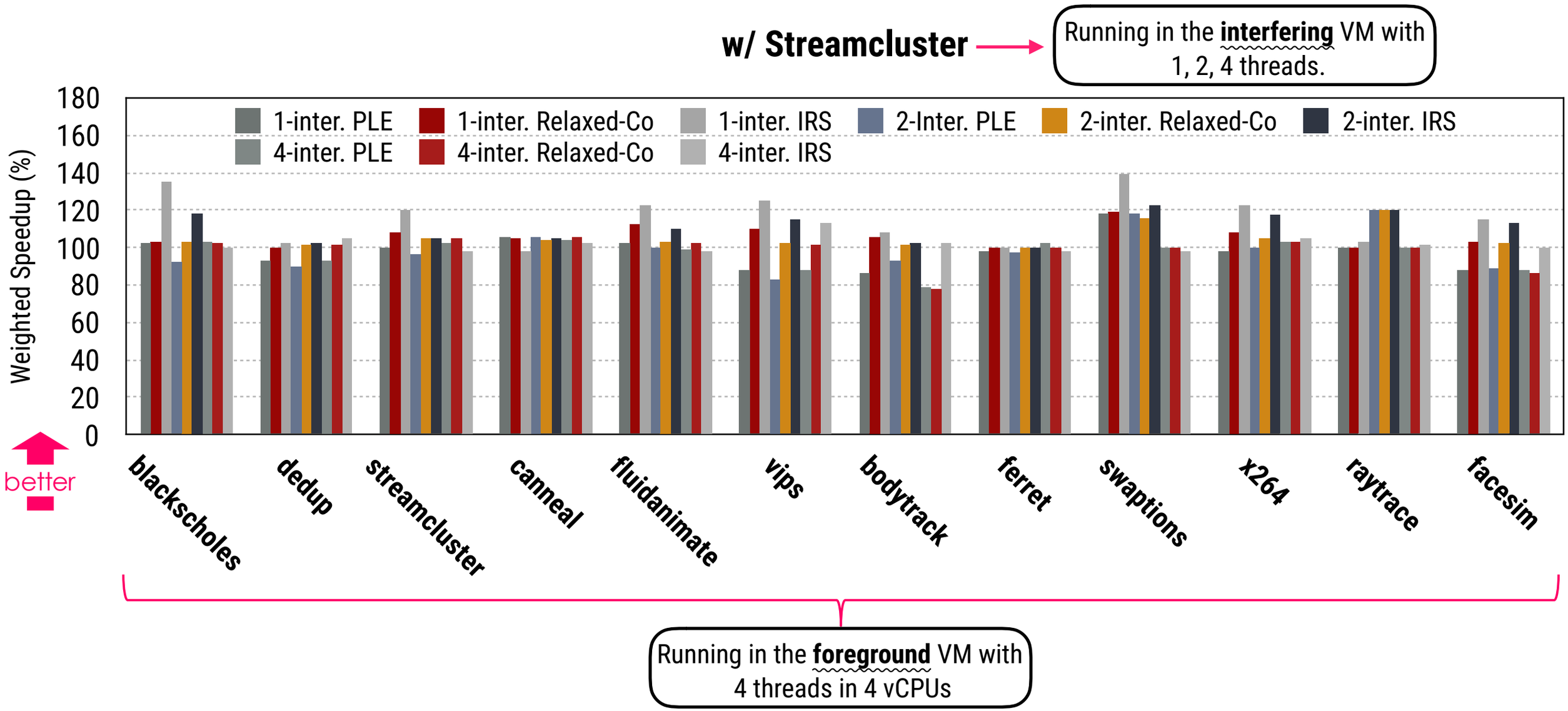
Thanks!
&
Questions?

Backup Slides …

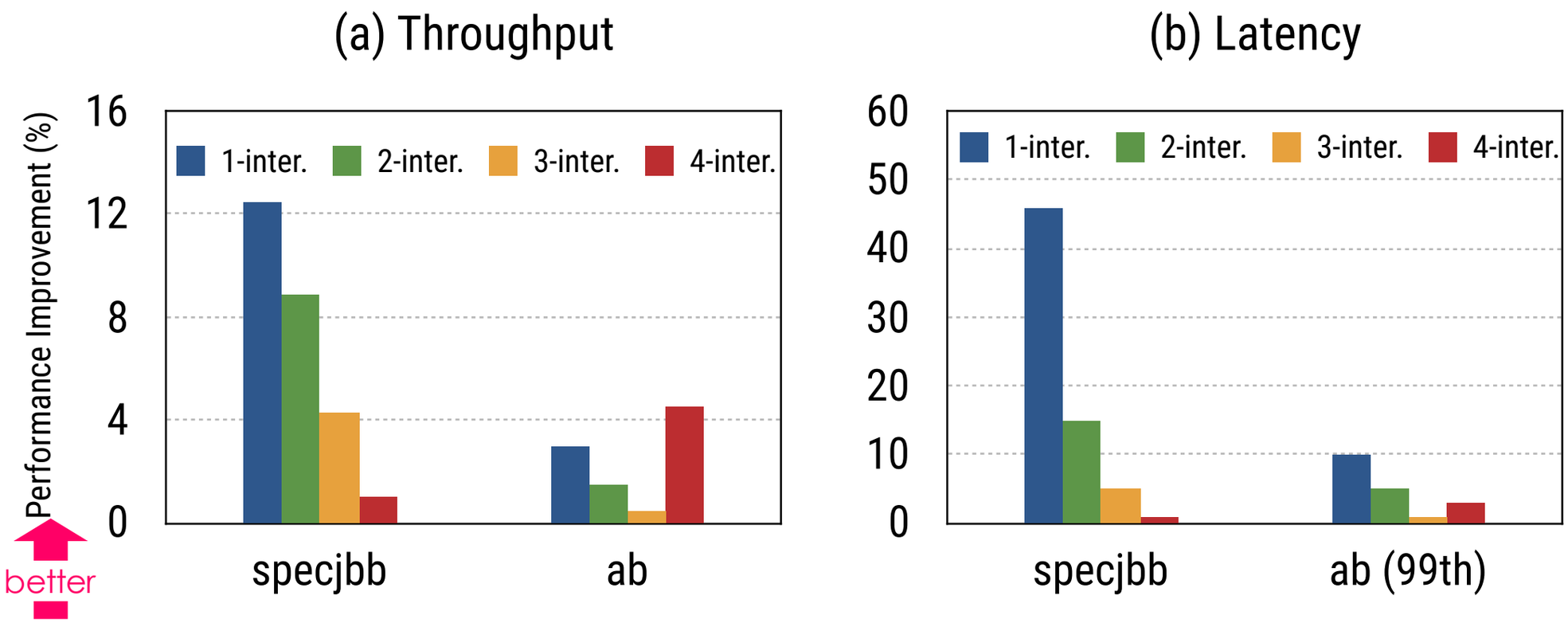
NASA parallel benchmarks



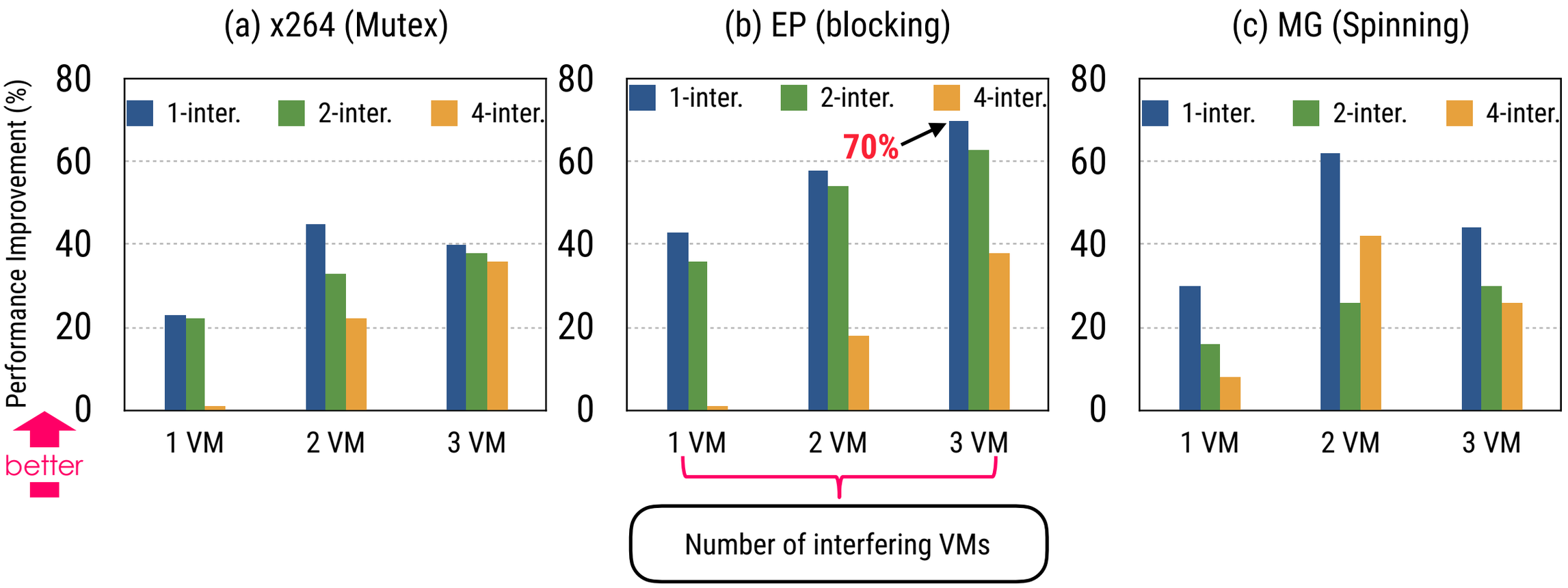
PARSEC benchmarks



Apache HTTP benchmarks



PARSEC benchmarks



Details about CPU Stacking

- Caused by an inherent deficiency in existing multicore schedulers

  - Multicore schedulers balance threads/vCPUs based on load — threads' CPU usage in previous rounds

  - **Deceptive idleness:** parallel programs experience idleness on lock waiter threads due to LHP or LWP, showing decreasing CPU usage as LHP or LWP worsens

  - Parallel threads are moved to a few cores to consolidate the load that cannot be justified to run a dedicated core

  - This exacerbates LHP or LWP