KVM FORUM

Mitigating Excessive Pause-Loop-Exiting in VM-Agnostic KVM

Kenta Ishiguro / Keio University

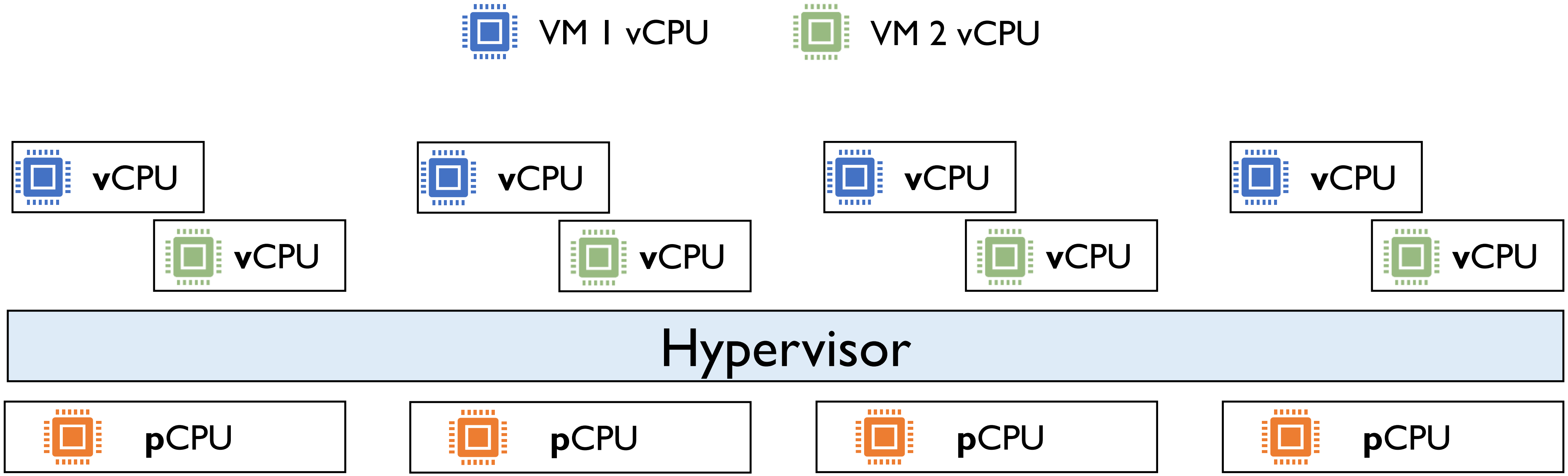
THE LINUX FOUNDATION

About Me

- 3rd year PhD student at Keio University, Japan
- Research interests:
  - performance and security of hypervisors
  - particularly KVM

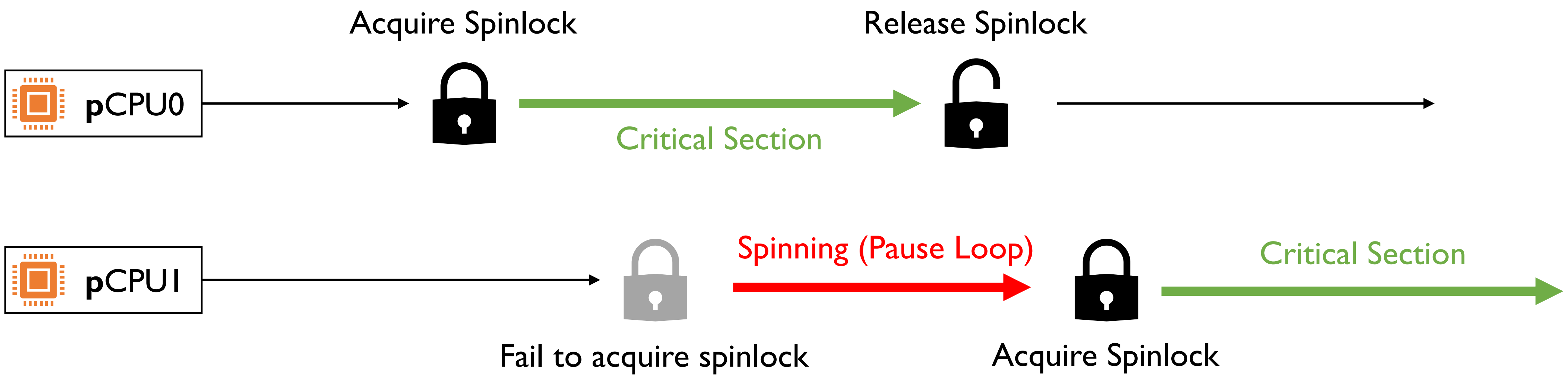
Oversubscribing **virtual** CPUs

- Enables better hardware utilization
- Requires multiplexing of virtual CPUs (vCPUs) on physical CPUs (pCPUs)



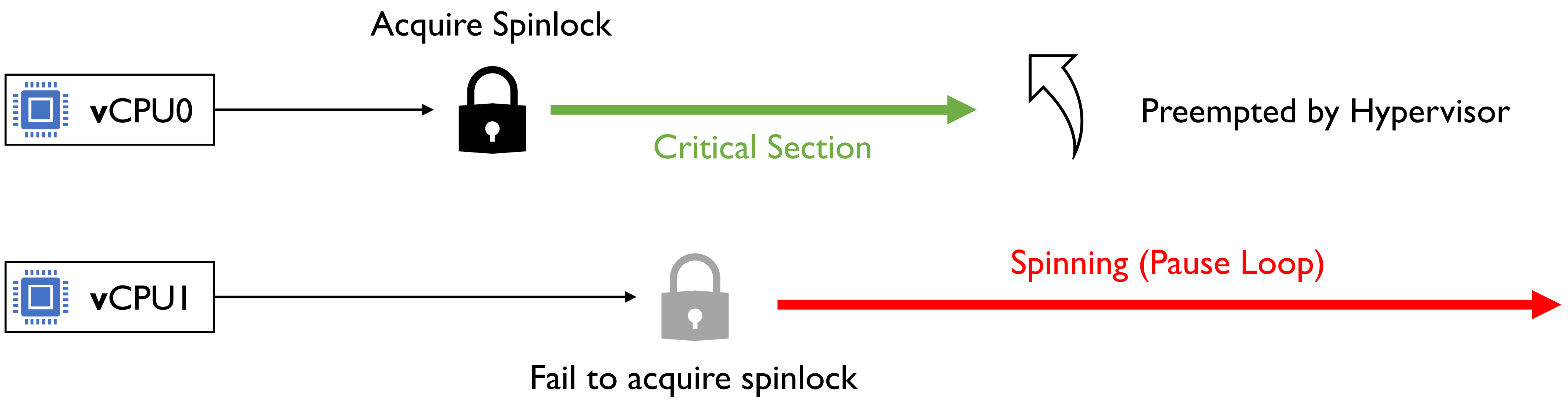
Critical section on pCPU

- Operating System assumption:
  - pCPUs are **always active**



Excessive vCPU spinning

- Oversubscription incurs **excessive spinning**
- **vCPU pre-emption** by Hypervisor **violates OS assumption**
- Lock-holder pre-emption problem



Improving guest performance by solving excessive vCPU spinning

- Problem
  - vCPU **wastes** its execution time by **excessive spinning**
- Ideal case
  - Hypervisor knows which vCPU should be scheduled right now
- But, hard to solve excessive vCPU spinning due to **semantic gap**
  - between KVM and Linux scheduler
  - between KVM and guest VMs

Improving guest performance by solving excessive vCPU spinning

- Problem
  - vCPU **wastes** its execution time by **excessive spinning**
- Ideal ca
  - Hype
- But, hard to solv essive vCPU spinning due to **semantic gap**
  - between KVM and Linux scheduler
  - between KVM and guest VMs

Boosting a target vCPU can be impeded by the Linux scheduler to keep fairness between vCPUs

Improving guest performance by solving excessive vCPU spinning

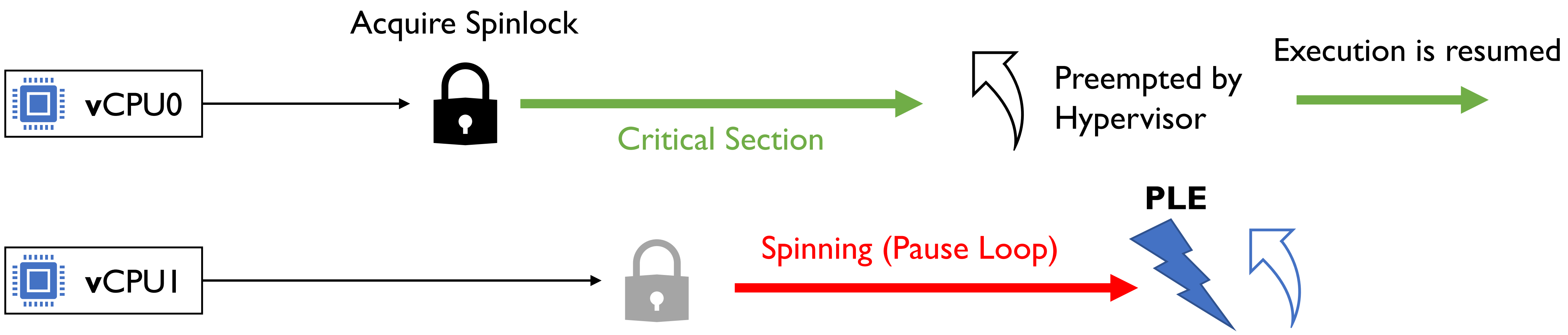
- Problem
  - vCPU **wastes** its execution time by **excessive spinning**
- Ideal ca
  - Hype

Hard to build comprehensive vCPU candidate set for boosting

- But, hard to solv ive vCPU spinning due to **semantic gap**
  - between KVM d Linux scheduler
  - between KVM and guest VMs

KVM approach with hardware feature

- KVM leverages Pause Loop Exiting (**PLE**) on Intel x86
  - **detects** excessive vCPU spinning
  - gives hypervisors a chance to **re-schedule** vCPUs



PLE **detects** excessive spinning and
transfers the control to the hypervisor for re-scheduling

KVM's strategy to suppress PLE events

- Reschedule PLE-ed vCPU to another preempted vCPU
  - in kvm_vcpu_on_spin

- Co-operative rescheduling with Linux scheduler
  - leverages yield_to provided by CFS scheduler
  - makes a request to Linux scheduler to yield and boost vCPUs

- Selects a vCPU to boost in round-robin from candidates
  - resource-waiter, lock-waiter, and IPI-receiver vCPUs are candidates

What happens in the worst case?

KVM trace with running two 8-vCPU VMs simultaneously

```
CPU 2/KVM  2679 [006]  276.731784:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731819:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731823:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731845:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731849:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731871:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731875:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731897:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731901:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731923:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731927:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731949:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731954:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731975:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731980:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732001:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732006:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732027:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732032:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732053:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732058:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
```

PLE events occur continuously (> 100 times)

What happens in the worst case?

KVM trace with running two 8-vCPU VMs simultaneously

```
CPU 2/KVM  2679 [006]  276.731784:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731819:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731823:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731845:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731849:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731871:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731875:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731897:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731901:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731923:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731927:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731949:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731954:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731975:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731980:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732001:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732006:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732027:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732032:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732053:   kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732058:    kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
```

Continuous PLE occur in the short period

What happens in the worst case?

KVM trace with running two 8-vCPU VMs simultaneously

```
CPU 2/KVM  2679 [006]  276.731784:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731819:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731823:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731845:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731849:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731871:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731875:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731897:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731901:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTI    ip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731923:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731927:   kvm:kvm_exit: vcpu 2 reason PAUSE_I     ION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731949:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.731954:   kvm:kvm_                          N rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.731975:  kvm:kvm_e
CPU 2/KVM  2679 [006]  276.731980:   kvm:kvm_                          N rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732001:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732006:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732027:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732032:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]  276.732053:  kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]  276.732058:   kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
```

What happens in the worst case?

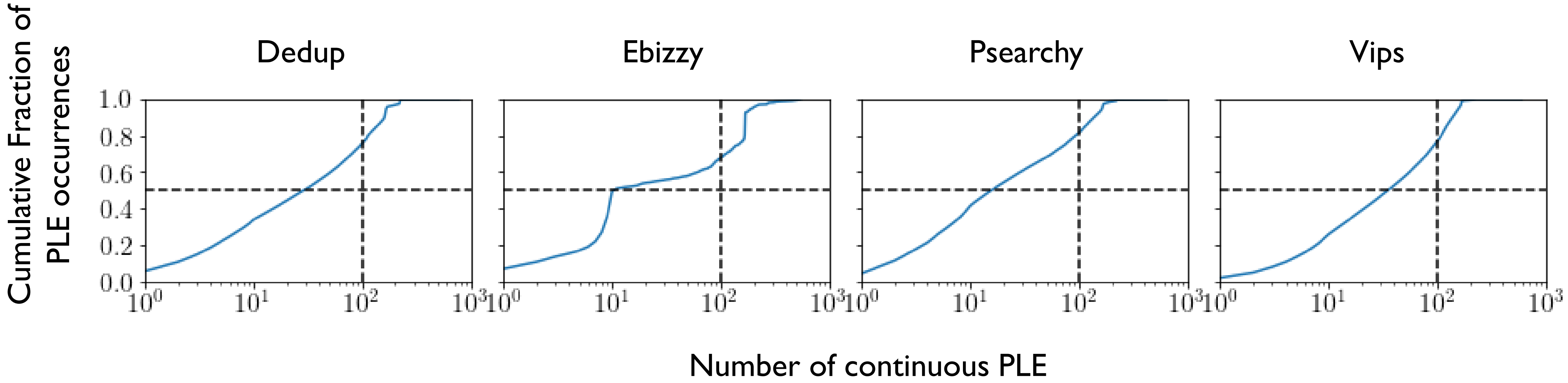
KVM trace with running two 8-vCPU VMs simultaneously

```
CPU 2/KVM  2679 [006]   276.731784:     kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]   276.731819:    kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]   276.731823:     kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]   276.731845:    kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]   276.731849:     kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM  2679 [006]   276.731871:    kvm:kvm_entry: vcpu 2
CPU 2/KVM  2679 [006]   276.731875:     kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
CPU 2/KVM                                                                                                            info 0 0
CPU 2/KVM
CPU 2/KVM                                                                                                              fo 0 0
CPU 2/KVM
CPU 2/KVM                                                                                                              fo 0 0
CPU 2/KVM
CPU 2/KVM                                                                                                              fo 0 0
CPU 2/KVM
CPU 2/KVM                                                                                                              fo 0 0
CPU 2/KVM
CPU 2/KVM                                                                                                            info 0 0
CPU 2/KVM
CPU 2/KVM  2679 [006]   276.732058:     kvm:kvm_exit: vcpu 2 reason PAUSE_INSTRUCTION rip 0xffffffffb72e29e3 info 0 0
```

- PLE events occur continuously (> 100 times)
  - in the short period
  - at the same code location

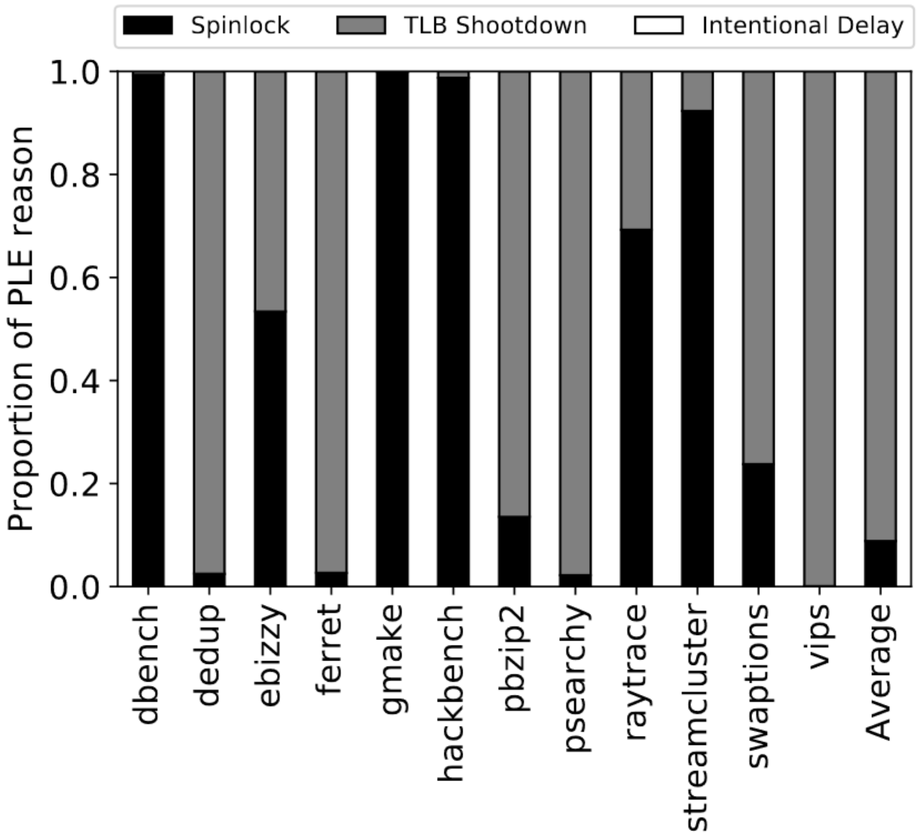
Continuous PLE events are **NOT** rare

- Most number of PLE events are from continuous PLE events
- Cause of PLE events: both spinlock and TLB shootdown



Continuous PLE events are NOT rare

- Most number of PLE events are from continuous PLE events
- Cause of PLE events: both spinlock and TLB shootdown



Functionality	Function causing PLE events
spinlock	**native_queued_spin_lock_slowpath** queued_write_lock_slowpath queued_read_lock_slowpath try_to_wake_up etc…
TLB shootdown	**smp_call_function_many** smp_call_function_single

Why PLE events occur continuously?

- Lost opportunity and Overboost
  - Comprehensive approach to identify the root cause of PLE does not exist
  - Mitigation: Strict Boost

- Scheduler mismatch
  - Linux scheduler **ignores hints** from hypervisor
  - Mitigation: Debooster

How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

} Optimizations against **spinlock**

- Boost halted vCPUs where they have received an IPI [2019]
  - Optimization against **TLB shootdown**

How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

} Optimizations against **spinlock**

- Boost halted vCPUs where they have received an IPI [2019]
  - ...on against **TLB shootdown**

Insights

- Spinlock still incurs continuous PLE events
- vCPUs in user mode also need to ack when TLB shootdown happens
- No need to boost IPI-receivers when lock-holder preemption happens

How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

Optimizations against **spinlock**

- Boost halted vCPUs
  on again

due to scheduler mismatch (described later)

Insights

- **<u>Spinlock</u>** <u>still incurs continuous PLE events</u>
- vCPUs in user mode also need to ack when TLB shootdown happens
- No need to boost IPI-receivers when lock-holder preemption happens

How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

} Optimizations against **spinlock**

- Boost halted vCPUs where they have received an IPI [2019]

Insights

Lost opportunity

- Spinlock still incurs … PLE events
- <u>**vCPUs in user mode** also need to ack when **TLB shootdown happens**</u>
- **No need** to boost **IPI-receivers** when **lock-holder preemption** happens

How to select candidate vCPU

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

Optimizations against **spinlock**

- Boost halted vCPUs where they have received an IPI [2019]

on against **TLB shootdown**

Insights

Overboost

- Spinlock still in
- **vCPUs in user mo** need to ack when **TLB shootdown happens**
- **No need** to boost **IPI-receivers** when **lock-holder preemption** happens

Mitigation: Strict Boost

- PLE-ed vCPU sleeps (NOT boost any vCPUs) [2009]
- Directed yield [2011]

- Less prioritize recently PLE-ed vCPUs [2012]
- Boost only preempted vCPUs [2013]
- Boost only vCPUs in kernel mode [2017]

Optimizations against **spinlock**

- B 019]

Introduce new strategies for candidate vCPU selection

- Boost IPI-receiver vCPUs in user mode
- Not boost halted vCPUs if the yielded vCPU has NOT sent an IPI to it

Why PLE events occur continuously?

- ~~Lost opportunity and Overboost~~ Strict Boost

- Scheduler mismatch

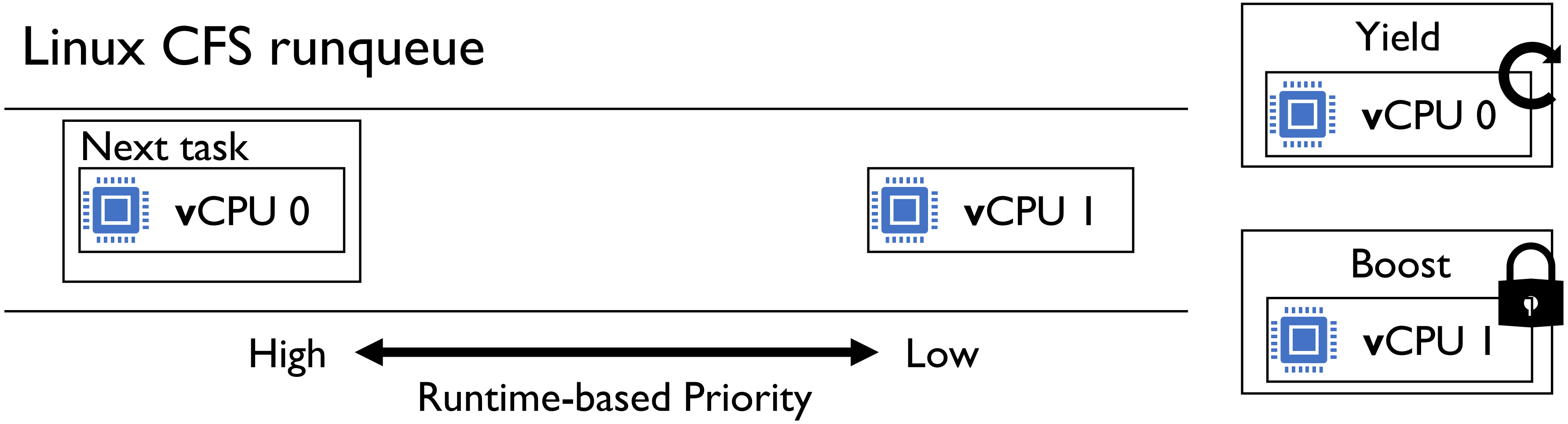
Problem: Scheduler Mismatch

- Linux CFS does **not distinguish** between vCPUs and other threads
  - KVM makes request to Linux CFS for boosting vCPU
  - But Linux CFS always keeps fairness between vCPUs

```
if (cfs_rq->next && wakeup_preempt_entity(cfs_rq->next, left) < 1) {
        /*
         * Someone really wants this to run. If it's not unfair, run it.
         */
        se = cfs_rq->next;
```

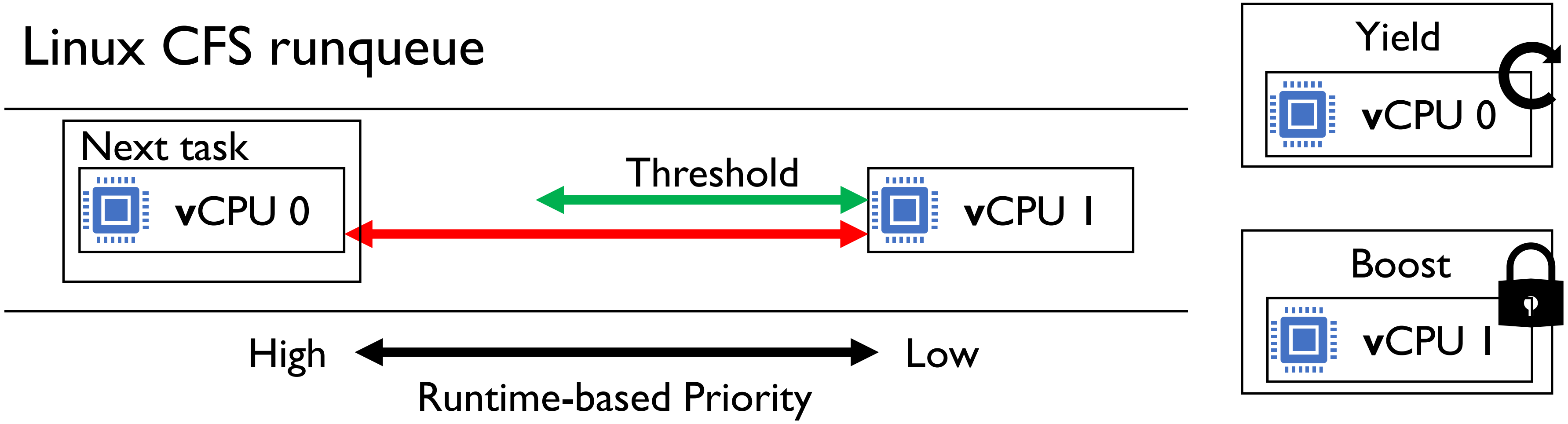
in kernel/sched/fair.c

Case Study: Scheduler Mismatch



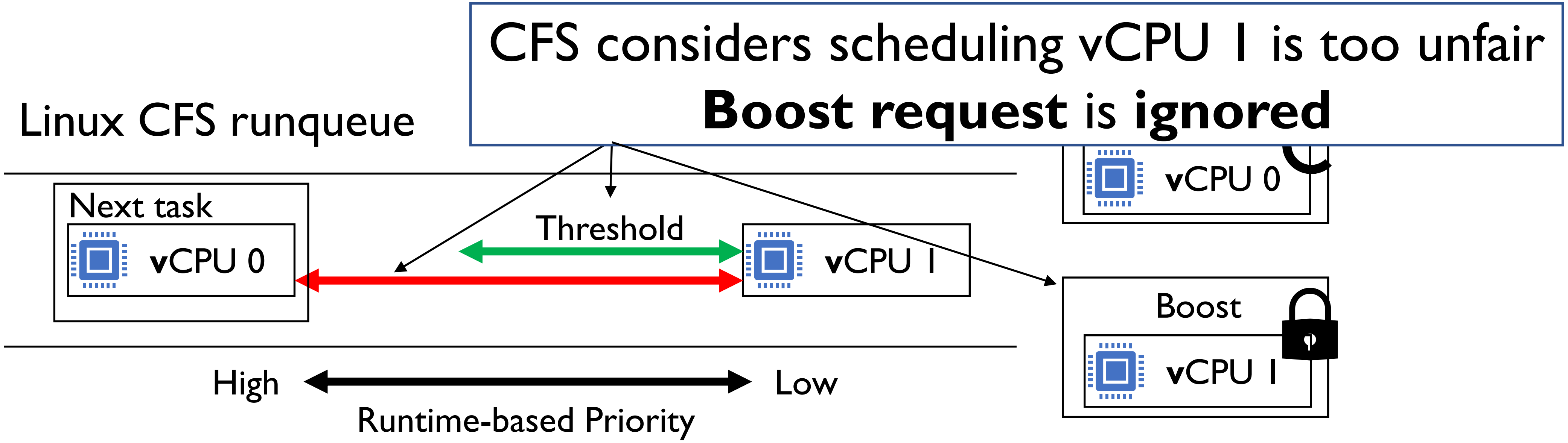
1. Picks the highest priority task: vCPU 0
2. $cputime_{vCPU1} - cputime_{vCPU0} > threshold$
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock

Case Study: Scheduler Mismatch



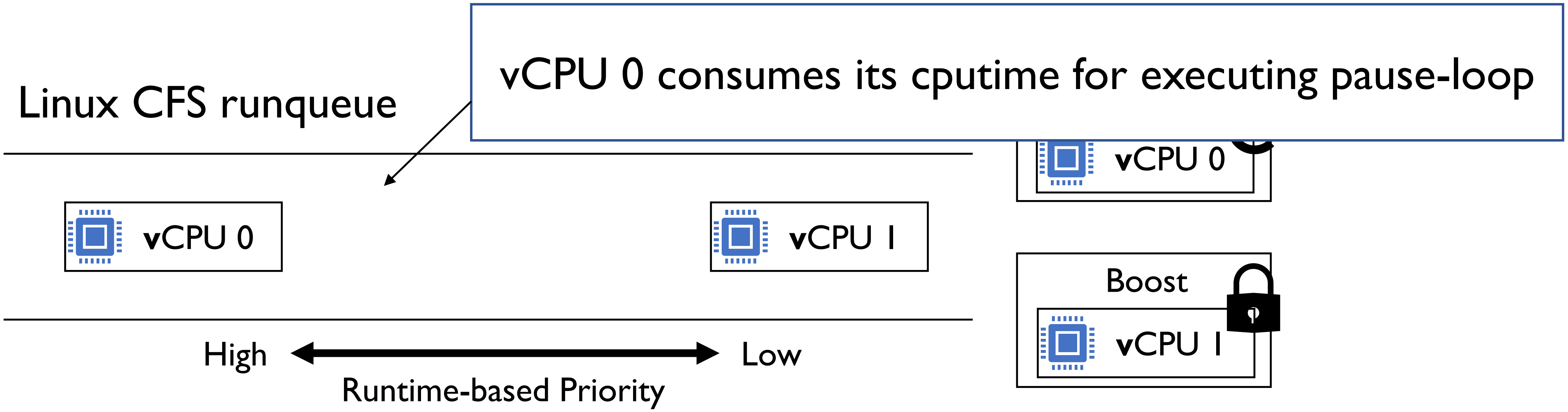
1. Picks the highest priority task: vCPU 0
2. $cputime_{vCPU1} - cputime_{vCPU0} > threshold$
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock

Case Study: Scheduler Mismatch



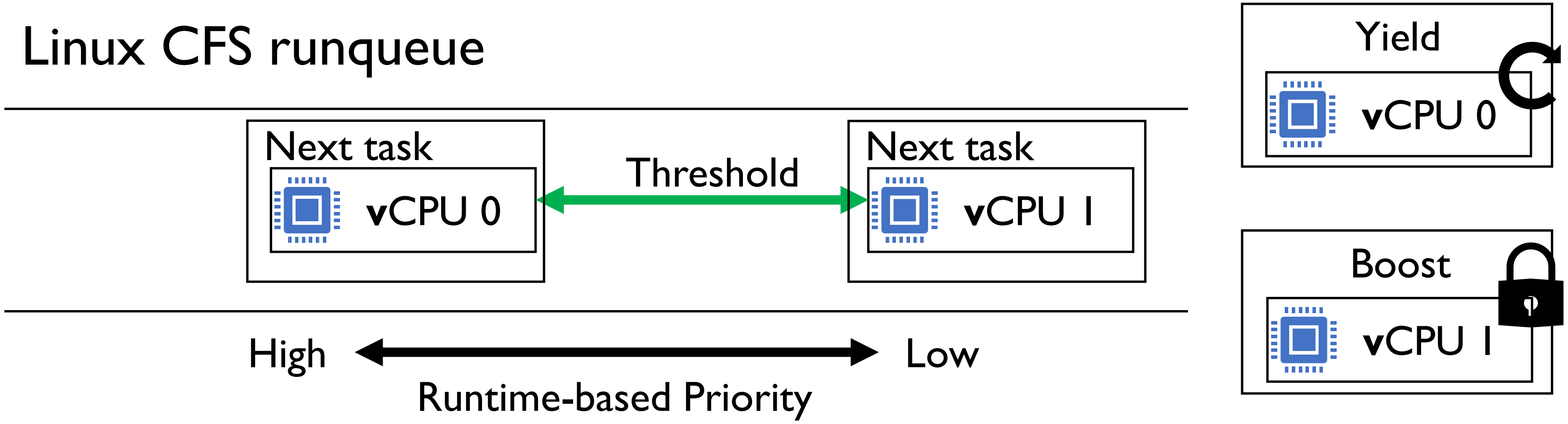
1. Picks the highest priority task: vCPU 0
2. $cputime_{vCPU1} - cputime_{vCPU0} > threshold$
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock

Case Study: Scheduler Mismatch



1. Picks the highest priority task: vCPU 0
2. $cputime_{vCPU1} - cputime_{vCPU0} > threshold$
3. Decides not to yield vCPU 0 and not to boost vCPU 1
4. vCPU 0 triggers PLE again because vCPU 1 still has the lock

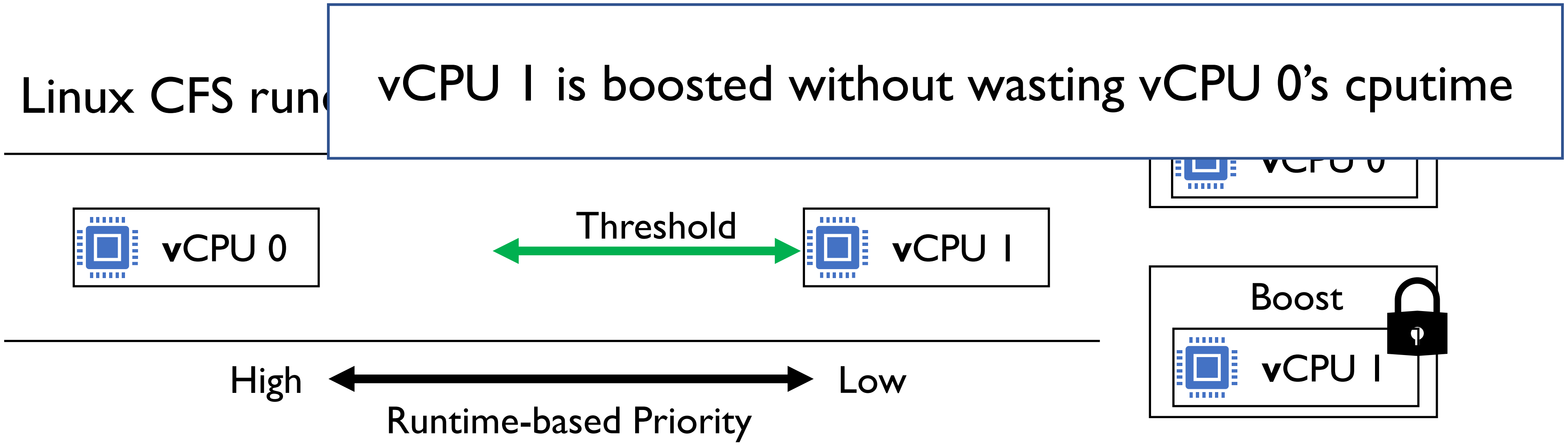
Case Study: Scheduler Mismatch



1. Picks the highest priority task: vCPU 0
2. vCPU 1 is boosted eventually because
   $cputime_{vCPU1} - cputime_{vCPU0} < threshold$

Mitigation: Debooster

- Debooster makes CFS not to hesitate to boost another vCPU instead of vCPU which exits due to PLE
  - by lowering PLE-ed vCPU priority



Implementation

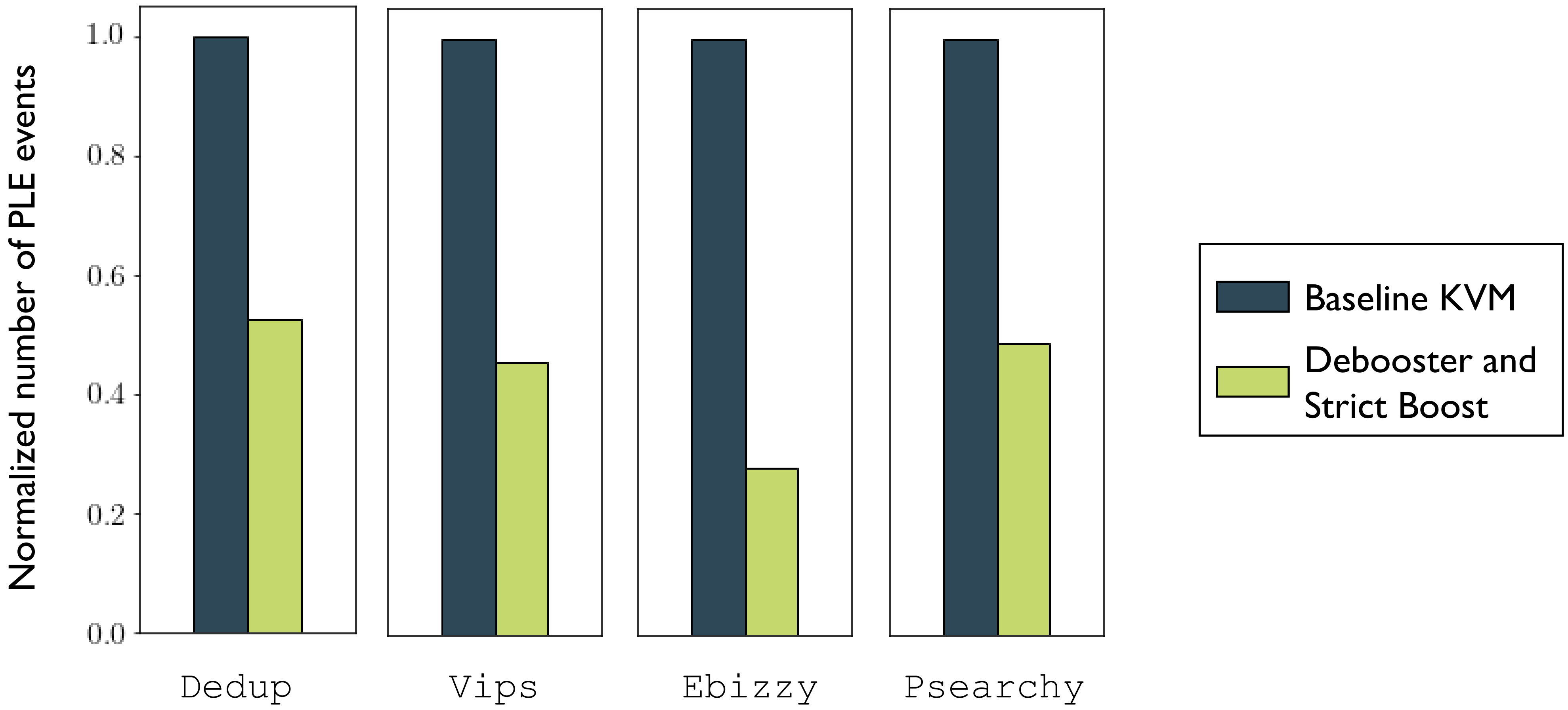
- Modified Linux KVM 5.6.0
- Our mitigations require 41 LoC modification
  - Debooster: yield_to_task() interface in CFS
  - Strict Boost: vCPU candidate selection and IPI handler in KVM

Experimental Setup

- Testbed
  - 8-core (Intel Xeon)
  - 2 VMs with 8 vCPU for each

- Benchmarks
  - `dedup` and `vips` from parsec
  - `psearchy` from mosbench
  - `ebizzy`
  - `swaptions`: CPU-intensive workload co-runner in another VM

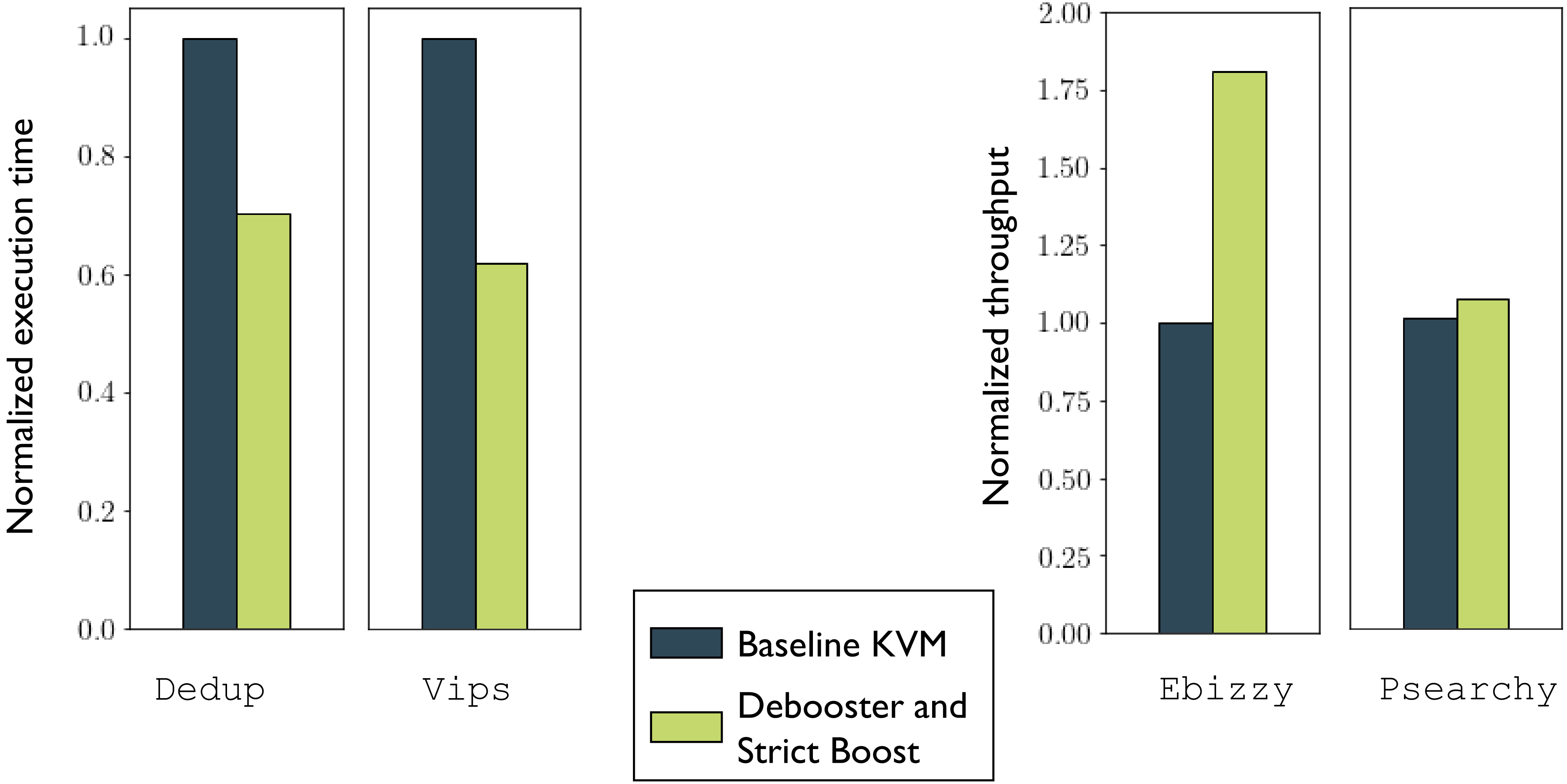
Evaluation: Reduction of PLE Occurrences



Evaluation: Performance Improvement

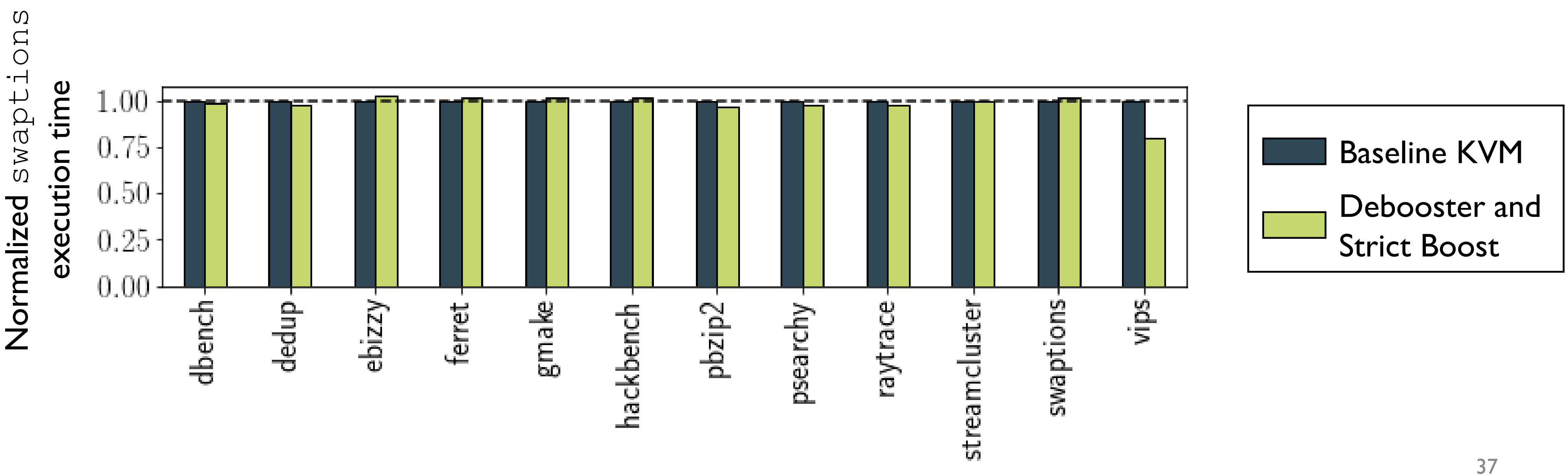
Execution time is **reduced by 40 %**

Throughput is **improved by 75 %**



Evaluation: System Fairness

- Co-runner's performance degradation cannot be seen
- Mitigations do not raise the priority of the boosted vCPU



Conclusion

- Oversubscribing vCPUs incurs excessive spinning
- Pause-Loop-Exiting (PLE) unfortunately does not fix it
- due to the semantic gap between
  - KVM and guest VMs
  - KVM and Linux scheduler

- Introduced mitigations against identified problems improve apps throughput by up to 75 %
  - Strict Boost against lost opportunity and overboost
  - Debooster against scheduler mismatch

- Problem investigated by the KVM community
  - https://lore.kernel.org/kvm/20210421150831.60133-1-kentaishiguro@sslab.ics.keio.ac.jp/
  - https://lore.kernel.org/kvm/1618542490-14756-1-git-send-email-wanpengli@tencent.com/

- Ishiguro, K., Yasuno, N., Aublin, P. L., & Kono, K. (2021, April). Mitigating excessive vCPU spinning in VM-agnostic KVM (VEE ’21)

Thanks & QA

Mitigating Excessive Pause-Loop-Exiting in VM-Agnostic KVM

Contact: kentaishiguro@sslab.ics.keio.ac.jp