Lock Holder Preemption Problem in Multiprocessor Virtualization

Burak Selcuk

RheinMain University of Applied Sciences
Informatik Master

August 24, 2017

Table of Contents

Motivation

- Virtualization is widely used and the fundamental of Cloud Computing
- Guest OS (should) behave like they are running on a physical machine without knowing they are virtualized
- Process synchronization techniques rely on assumptions that may not hold when an OS is virtualized
- Violating those assumptions can cause critical performance issues
- Spinlock mechanism is one of them!

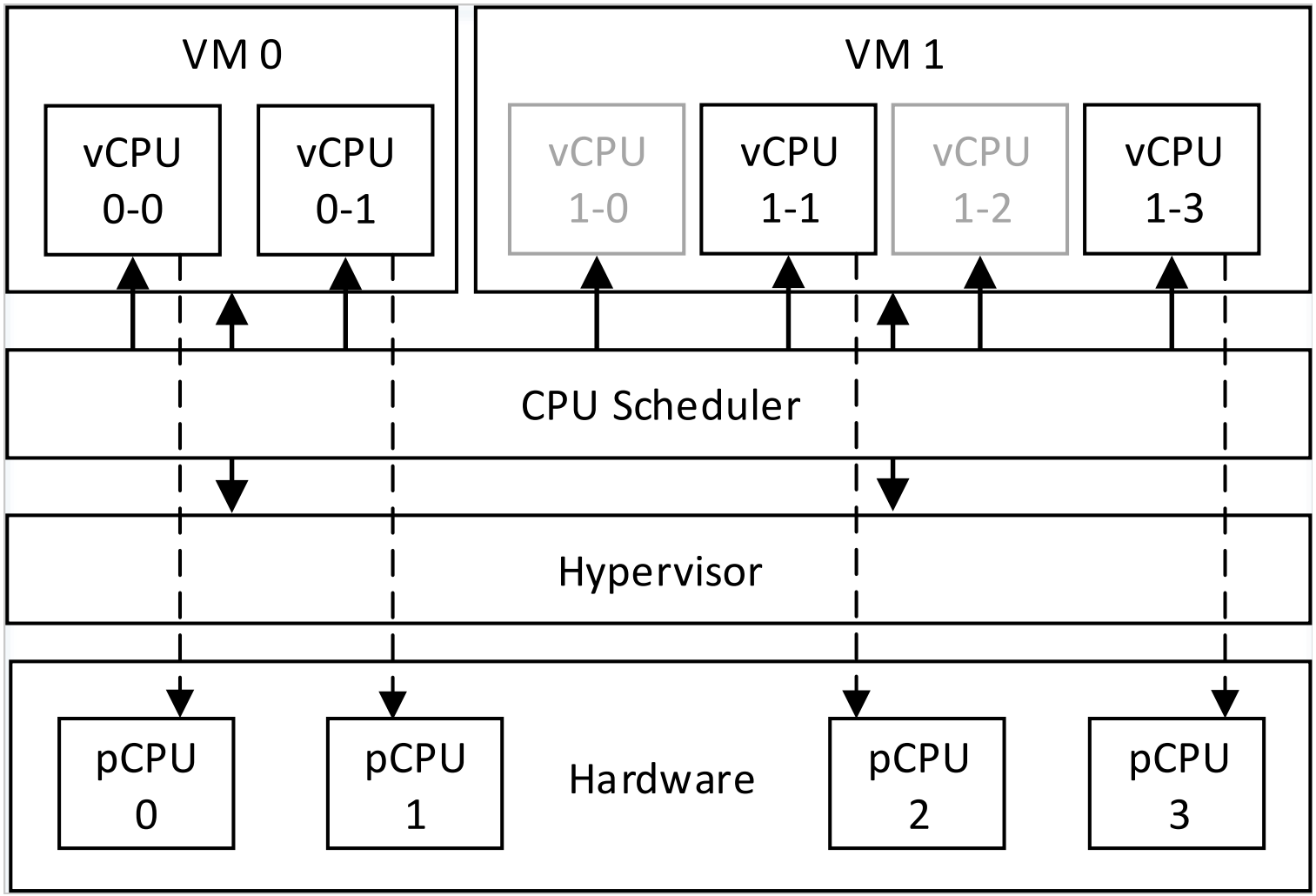
Spinlocks

- Inside the kernel, spinlocks are the lowest-level mutual exclusion mechanism
- Spinlocks uses busy waiting whenever the lock is taken by another thread
- This wastes CPU resources, but
  - critical sections in kernel are usually short and fast
  - busy waiting is less expensive than doing a context switch
- Most important: spinlocks rely on the assumption that the lock holder is not preempted while holding the spinlock

Overcommitment

- Virtual machines share the hardware resources of the physical machine
- A VM gets a number of virtual CPUs (vCPUs) assigned, which are then assigned to the physical CPUs (pCPU) by the hypervisor
- The total number of vCPUs is usually larger than the number of pCPUs, this situation is called overcommitment
- Whenever an overcommitment happens, there are always inactive vCPUs

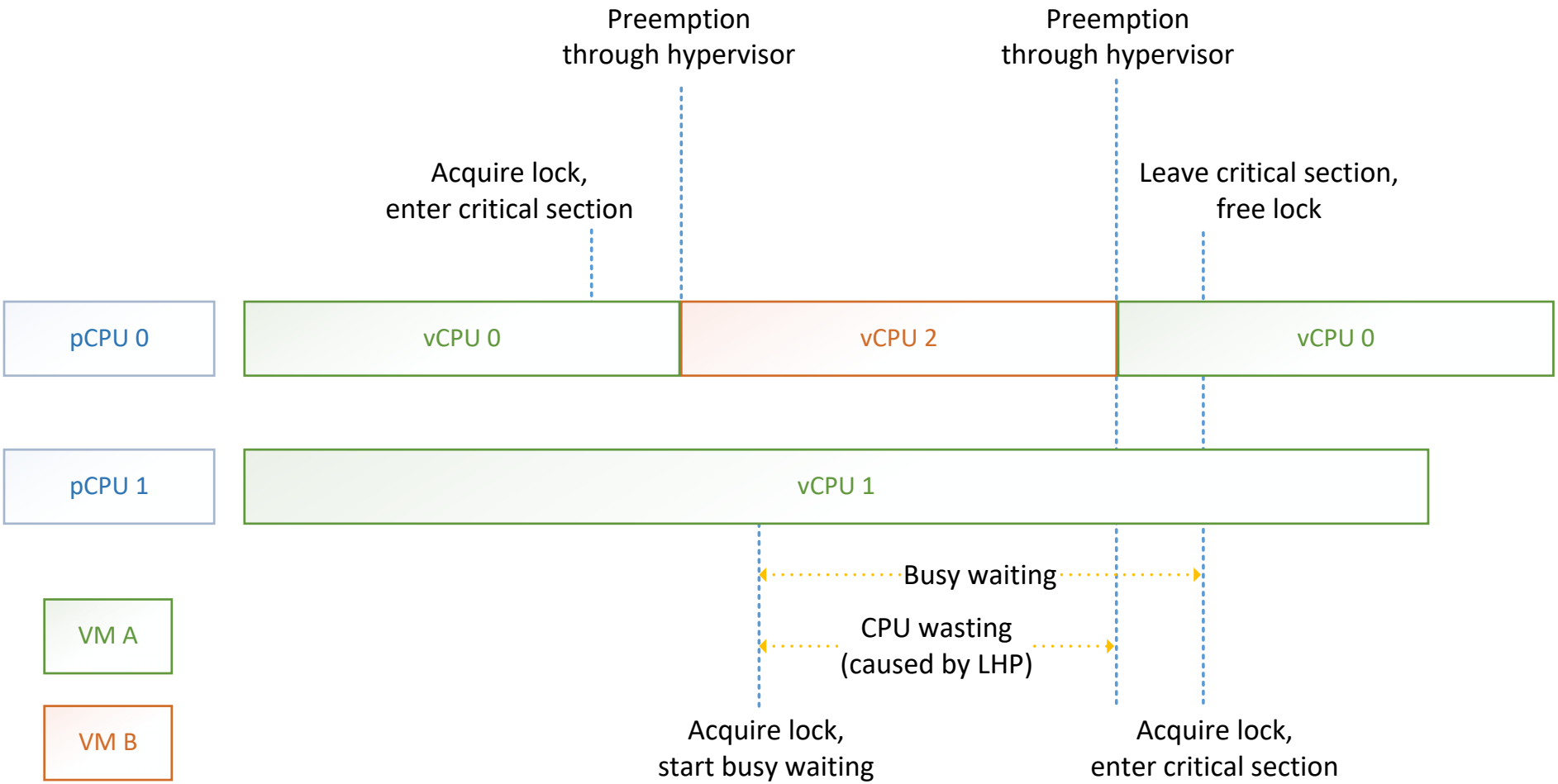
Example of Overcommitment



Scheduling

- Process and CPU scheduling happens on two layers:
  - Guest OS scheduler assigns the process to the vCPUs like usually
  - Hypervisor schedules the vCPUs to the pCPUs
- Each vCPU gets a time slice of execution until it gets preempted by the hypervisor
- A time slice duration is several microseconds (30-50ms)

Lock Holder Preemption Problem



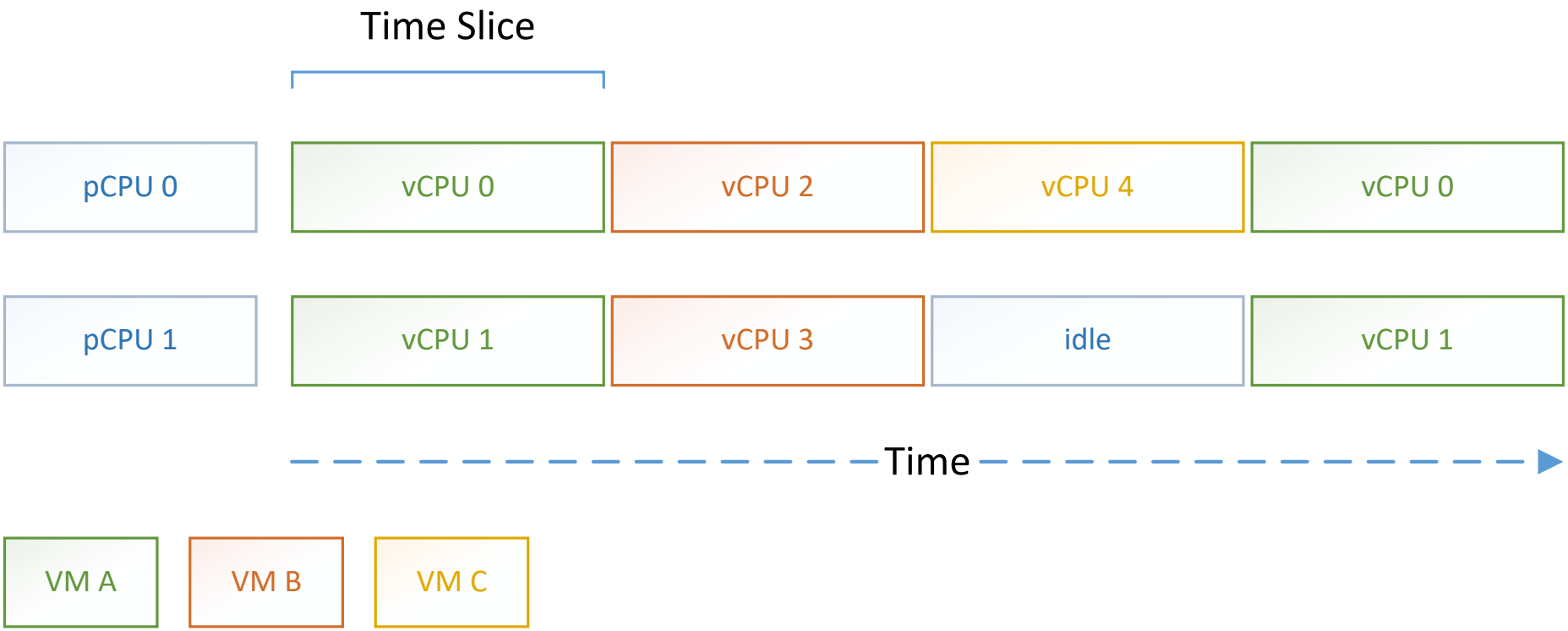
Solutions for LHP problem

- Co-Scheduling / Gang-Scheduling
- Guest OS modification
- Monitoring through hypervisor

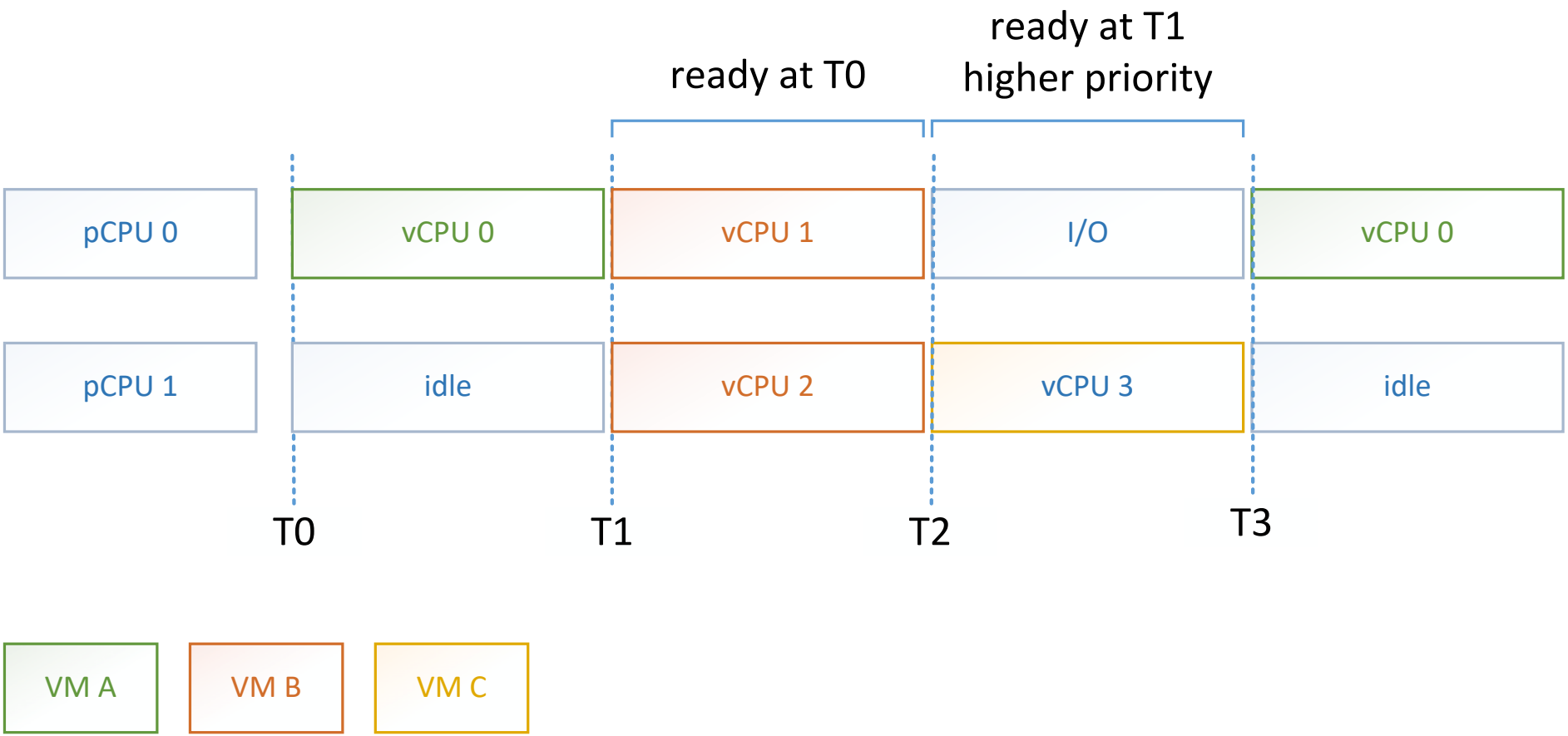
Co-scheduling

- Scheduling technique for multiprocessor systems
- Published by John Ousterhout in 1982
- Idea: Threads that communicate should be scheduled concurrently
- Transferred to virtualization for CPU scheduling
- Hypervisor assigns all vCPUs of a VM simultaneously, LHP problem can not occur
- ...but it can cause CPU fragmentation and priority inversion!

Example of Co-scheduling



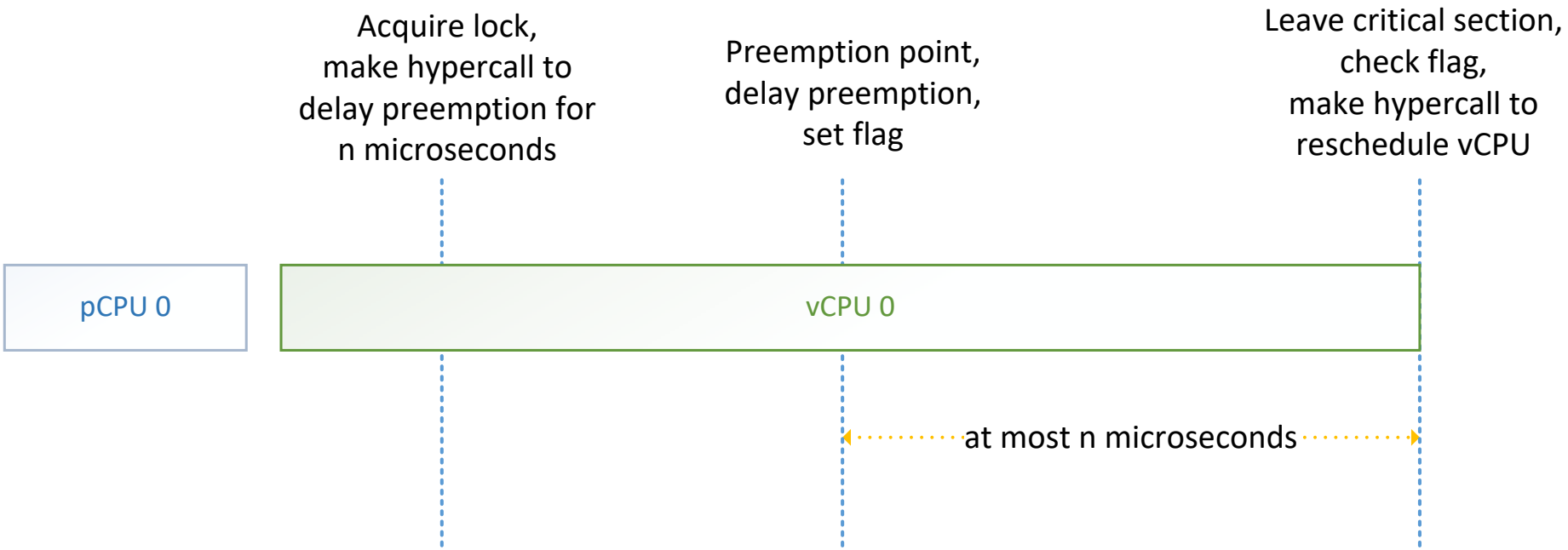
CPU Fragmentation and Priority Inversion



Guest OS Modification

- Hypervisor offers hypercalls to inform him about lock holding or spinning
- Source code necessary, each hypervisor may has different interface
- Suitable for para-virtualization
- Two possibilities:
  - Preemption delay (lock holder)
  - Notify hypervisor during long spinning (lock waiter)

Preemption delay



Threshold for Spinning

- Stop spinning after a number of iterations
- Will not avoid LHP but limit side effects of it
- After n iterations, make a hypercall to inform hypervisor about long spinning
- Hypervisor decides to preempt lock waiter vCPU and schedules another one of the same VM
  - Good choice: lock holder vCPU
  - Bad choice: another lock waiter, resulting in spinning again

Which solution is used?

- VMware vSphere:
  - Uses co-scheduling as *customized, relaxed* version
  - Limits LHP and CPU fragmentation
- Xen and KVM:
  - Both hypervisor offer interfaces to notify about long spinning
  - Implemented in Linux spinlock code
- Microsoft Hyper-V:
  - No co-scheduling necessary, Windows is major guest OS
  - Offers hypercalls for long spinning like in Xen/KVM
  - Probably used in Windows, usage in Linux may be possible

The End

Thanks for your attention!

Any questions?

Monitoring through Hypervisor

- Approaches whenever guest OS modification is not possible
- Recognize every entering and leaving of the kernel mode
  - Safe state: Guest OS in user mode. No spinlock is hold. Preemption safe.
  - Unsafe state: Guest OS in kernel mode. Spinlock may be hold. Preemption unsafe.
- Monitor guest OS instructions, e.g. IA-32 HLT (power saving)
- Use fake device driver, guest OS is in user mode whenever protocol is handled