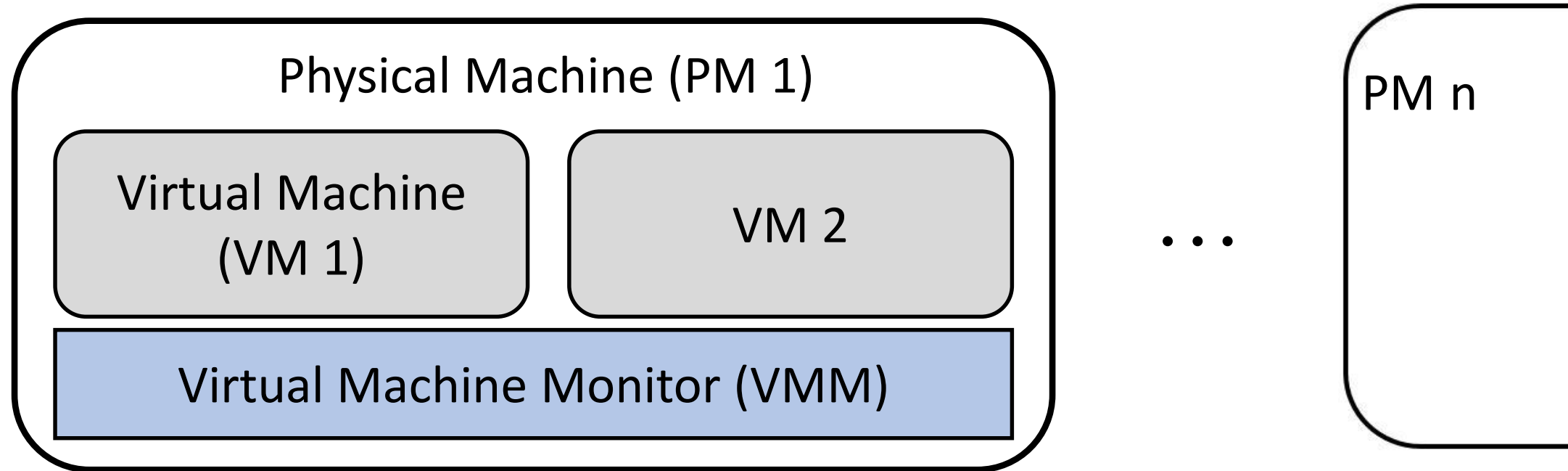


# Effectively Mitigating I/O Inactivity in vCPU Scheduling

*Weiwei Jia*<sup>1,2</sup>, Cheng Wang<sup>1</sup>, Xusheng Chen<sup>1</sup>, Jianchen Shan<sup>2</sup>, Xiaowei Shang<sup>2</sup>,  
Heming Cui<sup>1</sup>, Xiaoning Ding<sup>2</sup>, Luwei Cheng<sup>3</sup>, Francis C. M. Lau<sup>1</sup>, Yuexuan Wang<sup>1</sup>,  
Yuangang Wang<sup>4</sup>

Hong Kong University<sup>1</sup>, New Jersey Institute of Technology<sup>2</sup>, Facebook<sup>3</sup>, Huawei<sup>4</sup>

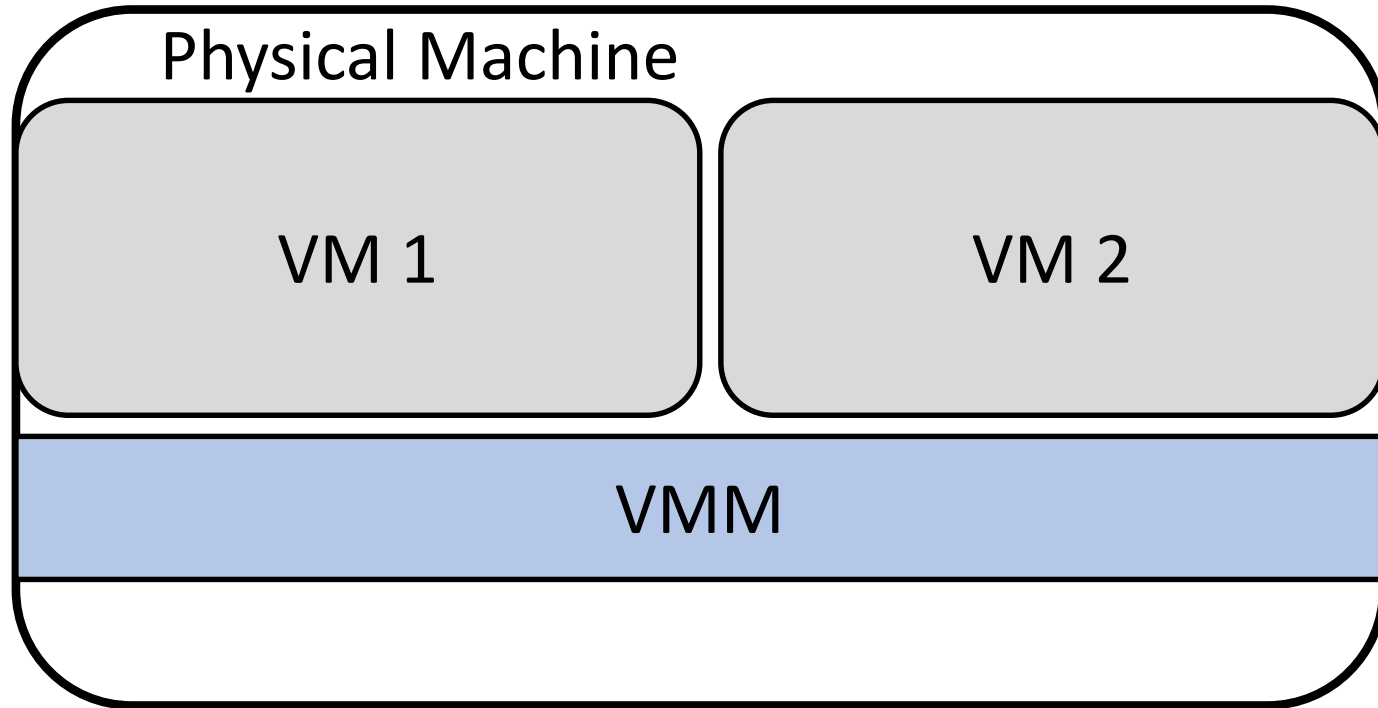
# VM consolidation is pervasive in clouds



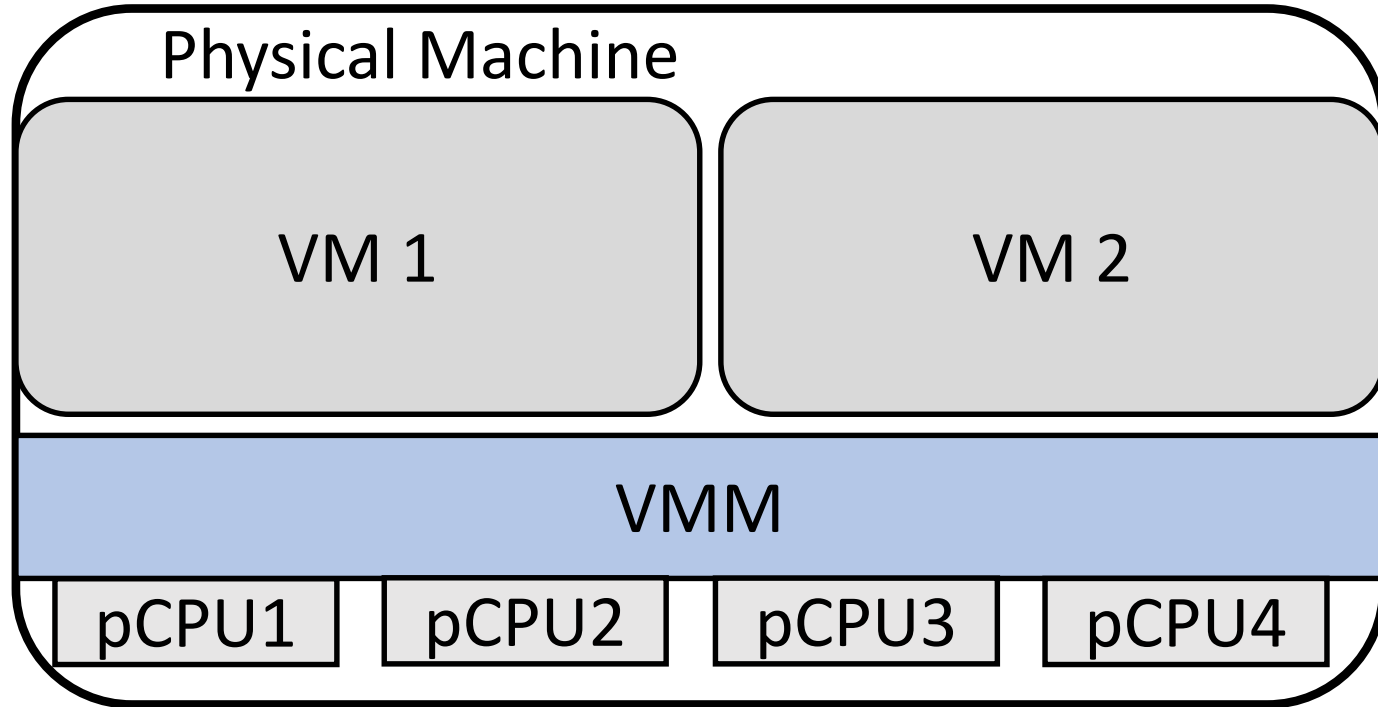
## Consolidation benefits:

- Ease management
- Save energy
- Improve resource utilization and system throughput

# Multiple vCPUs are scheduled to time-share a pCPU

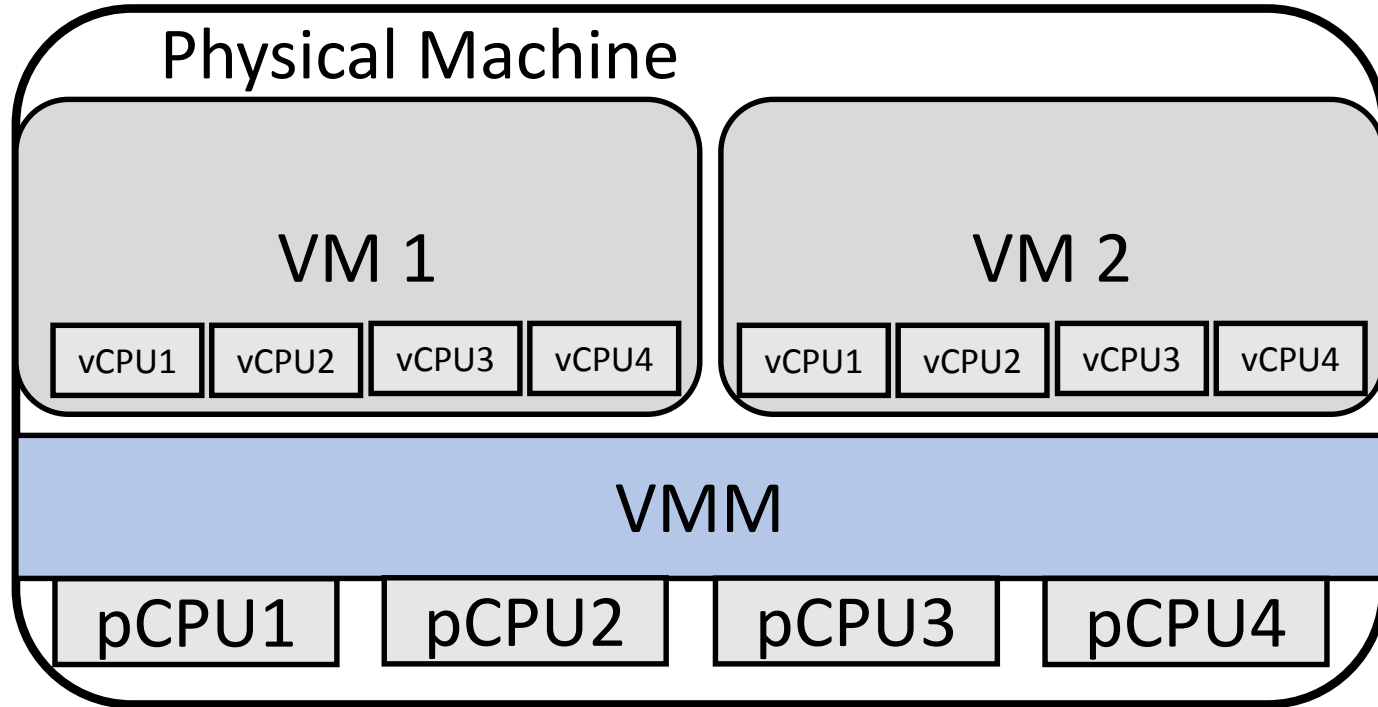


# Multiple vCPUs are scheduled to time-share a pCPU



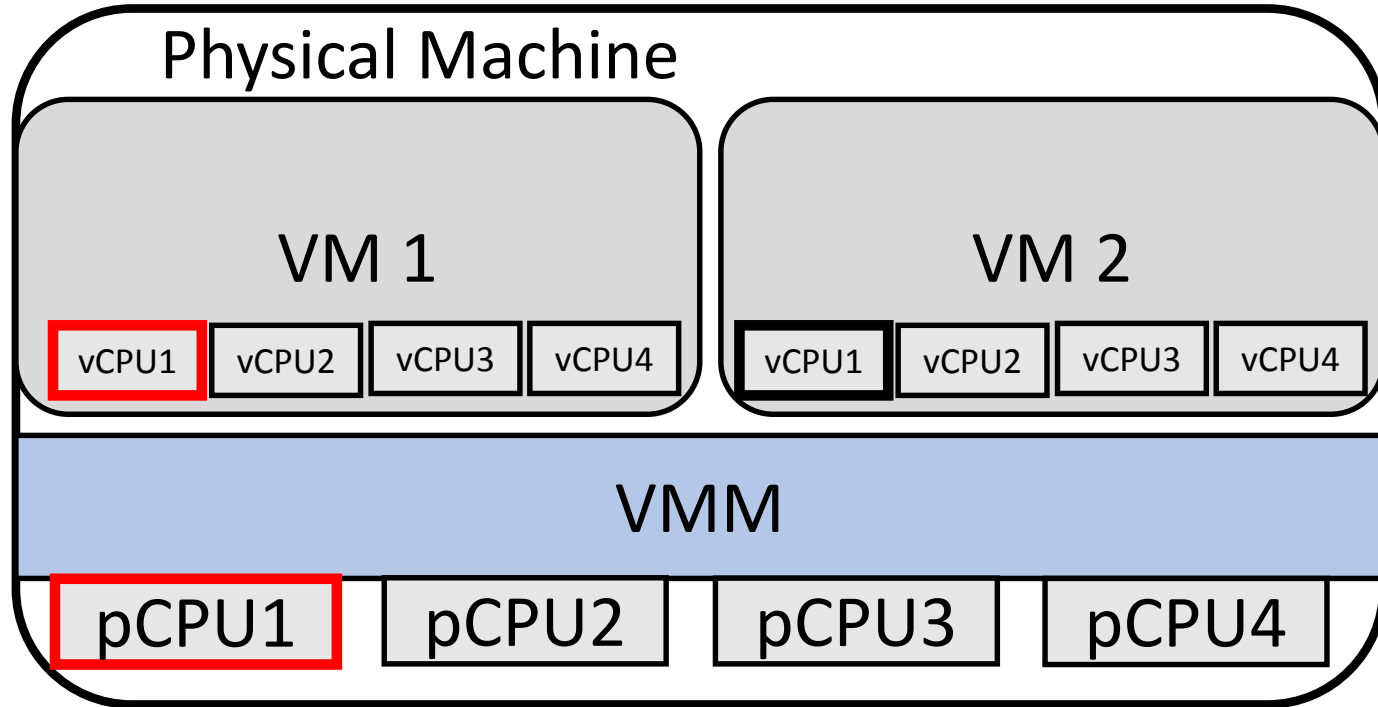
- Physical CPU (pCPU): hardware resources

# Multiple vCPUs are scheduled to time-share a pCPU



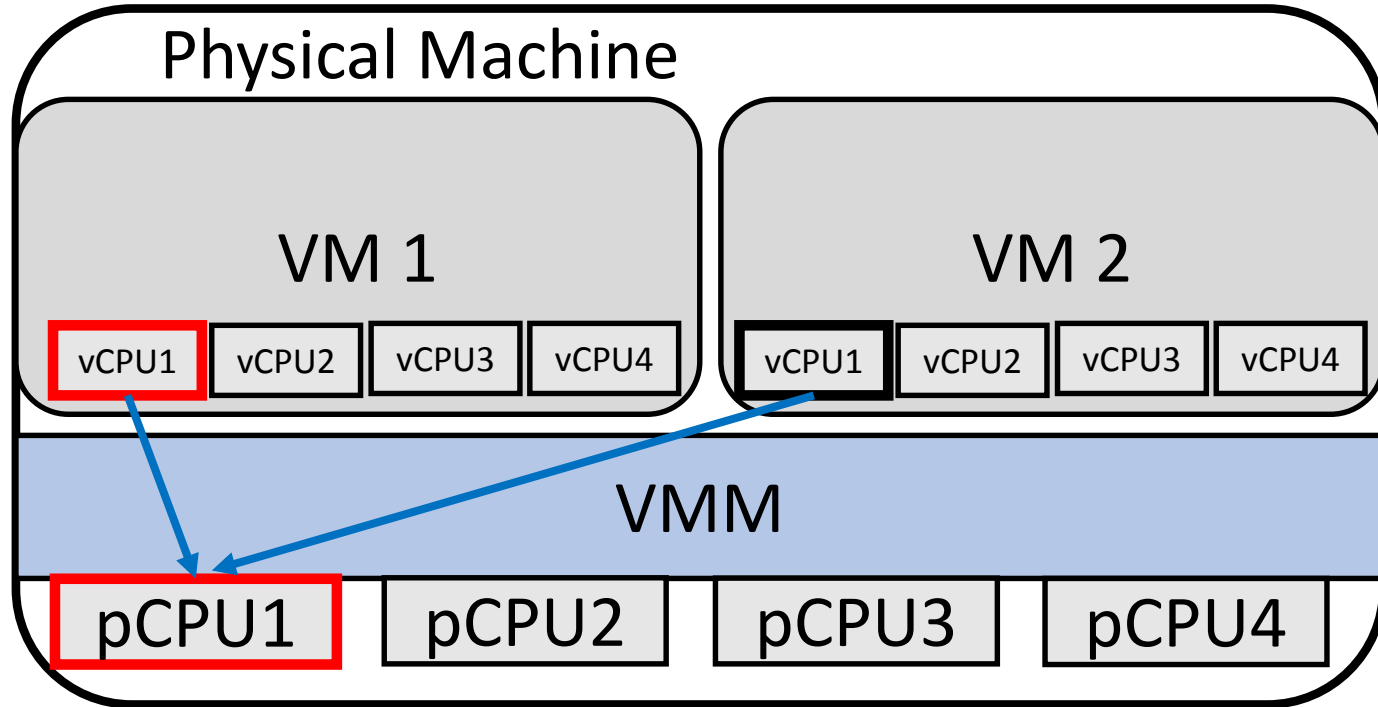
- Physical CPU (pCPU): hardware resources
- Virtual CPU (vCPU): processors in VM, threads in VMM
- Multiple vCPUs sharing one pCPU is often
  - E.g., VMWARE suggests 8-10 vCPUs to share one pCPU

# Multiple vCPUs are scheduled to time-share a pCPU



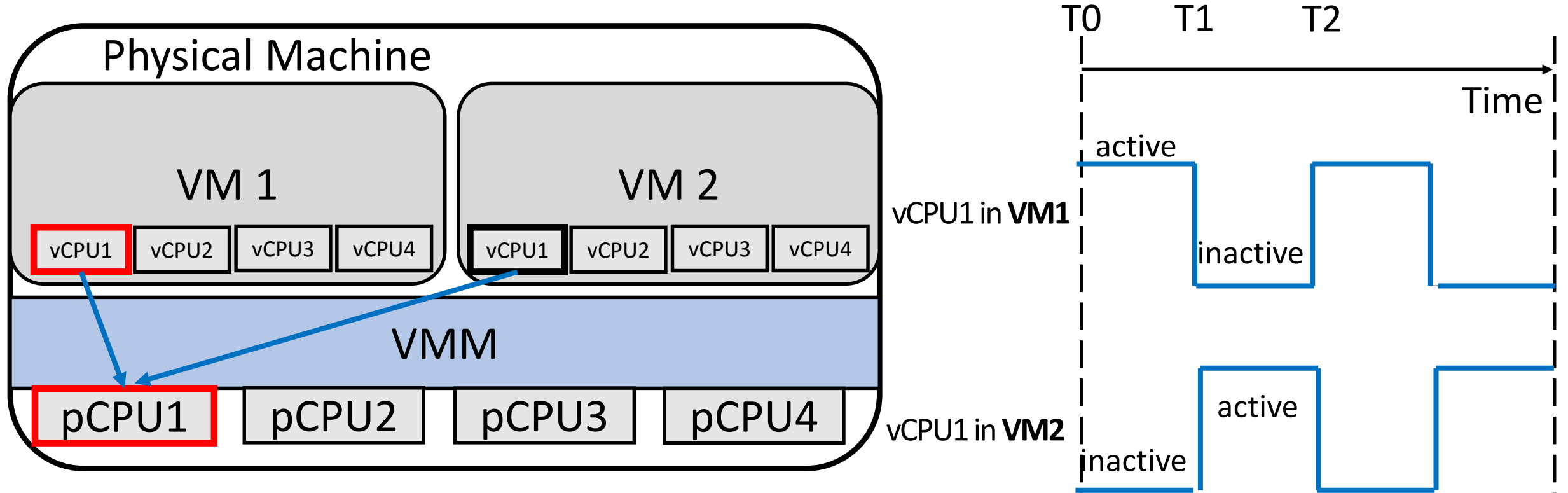
- Physical CPU (pCPU): hardware resources
- Virtual CPU (vCPU): processors in VM, threads in VMM
- vCPU scheduler schedules and deschedules vCPUs periodically

# Multiple vCPUs are scheduled to time-share a pCPU



- Physical CPU (pCPU): hardware resources
- Virtual CPU (vCPU): processors in VM, threads in VMM
- vCPU scheduler schedules and deschedules vCPUs periodically

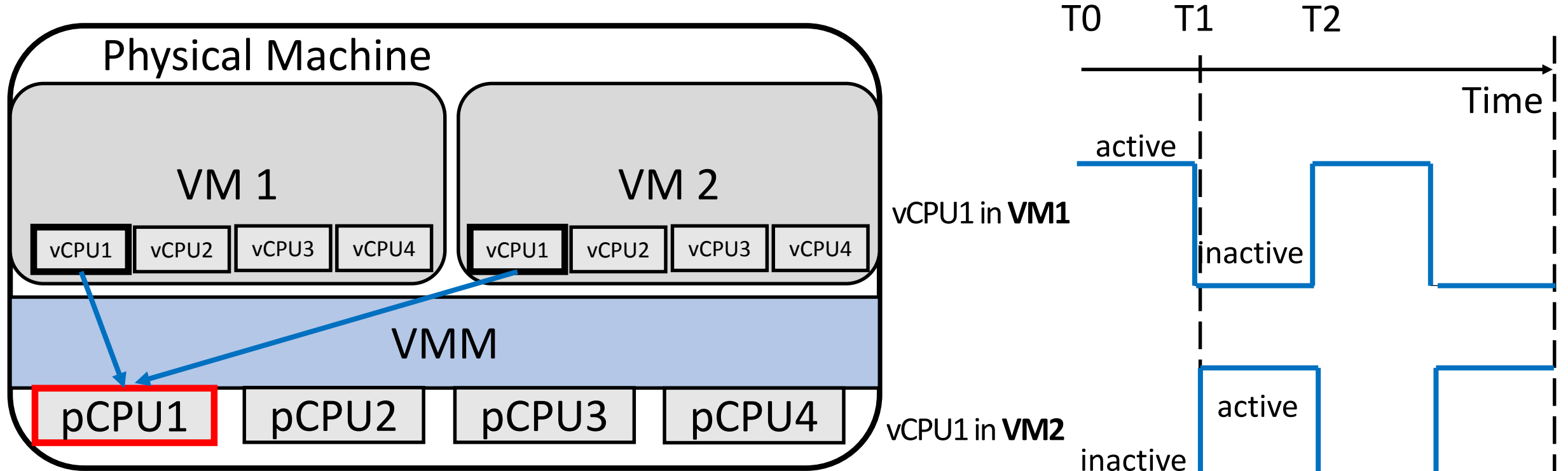
# Multiple vCPUs are scheduled to time-share a pCPU



- Physical CPU (pCPU): hardware resources
- Virtual CPU (vCPU): processors in VM, threads in VMM
- vCPU scheduler schedules and deschedules vCPUs periodically

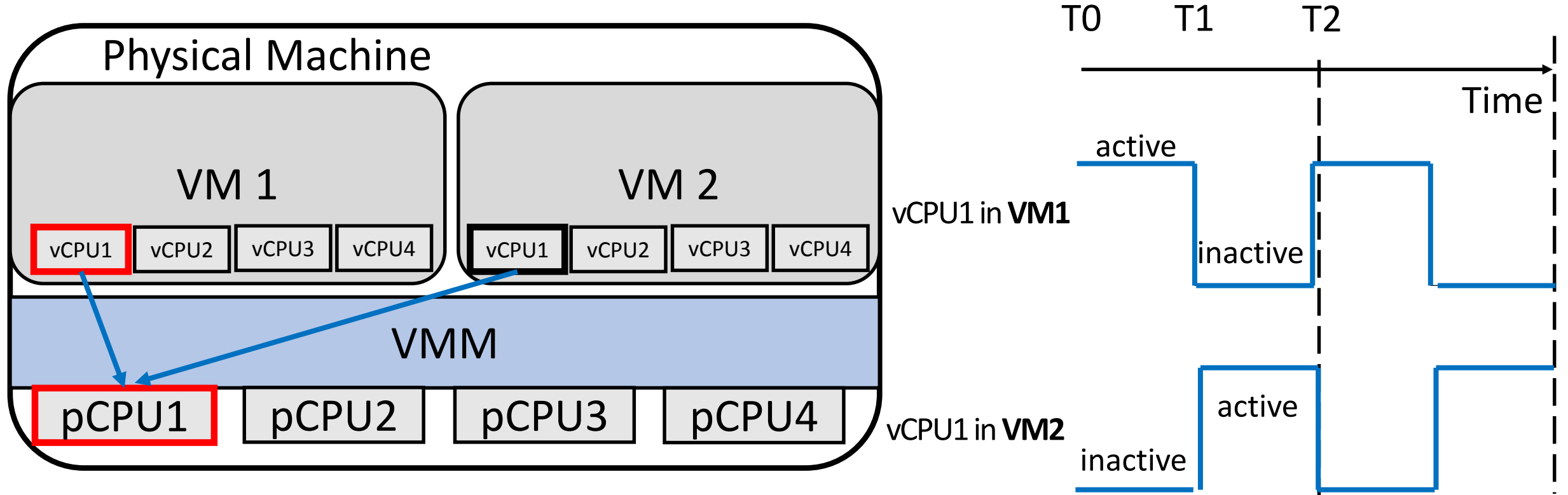


# Multiple vCPUs are scheduled to time-share a pCPU



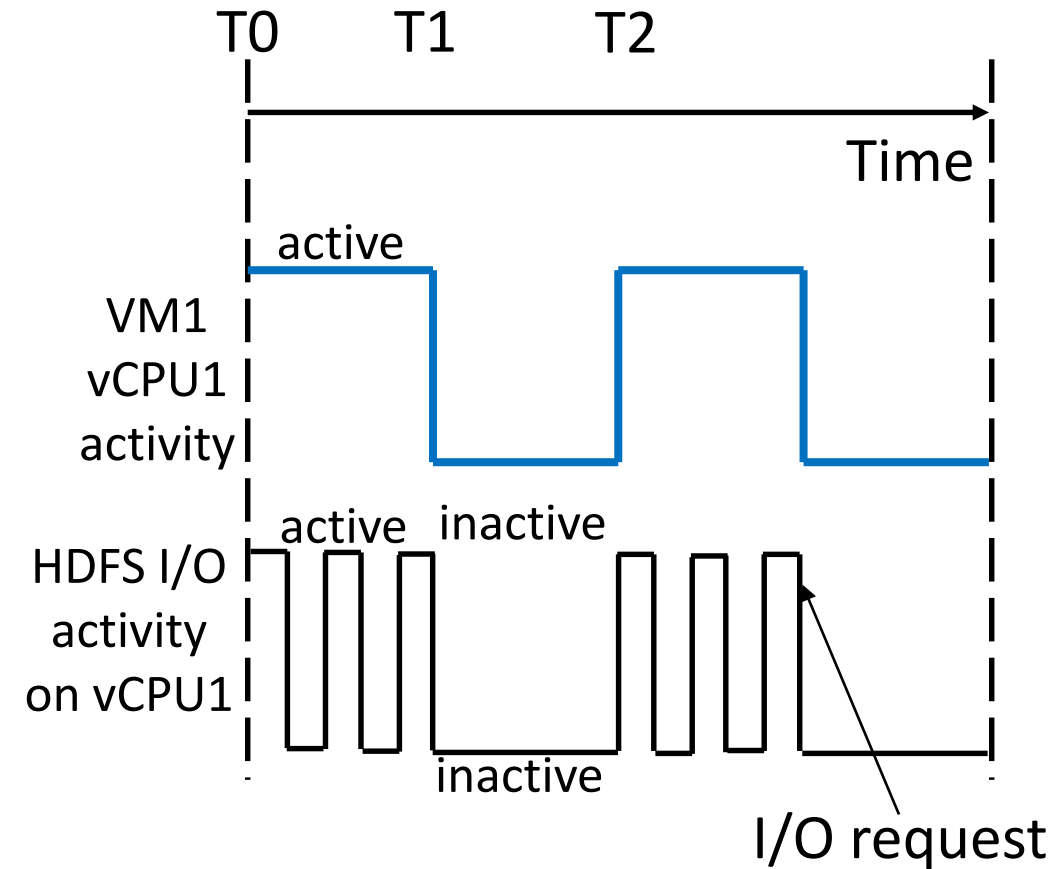
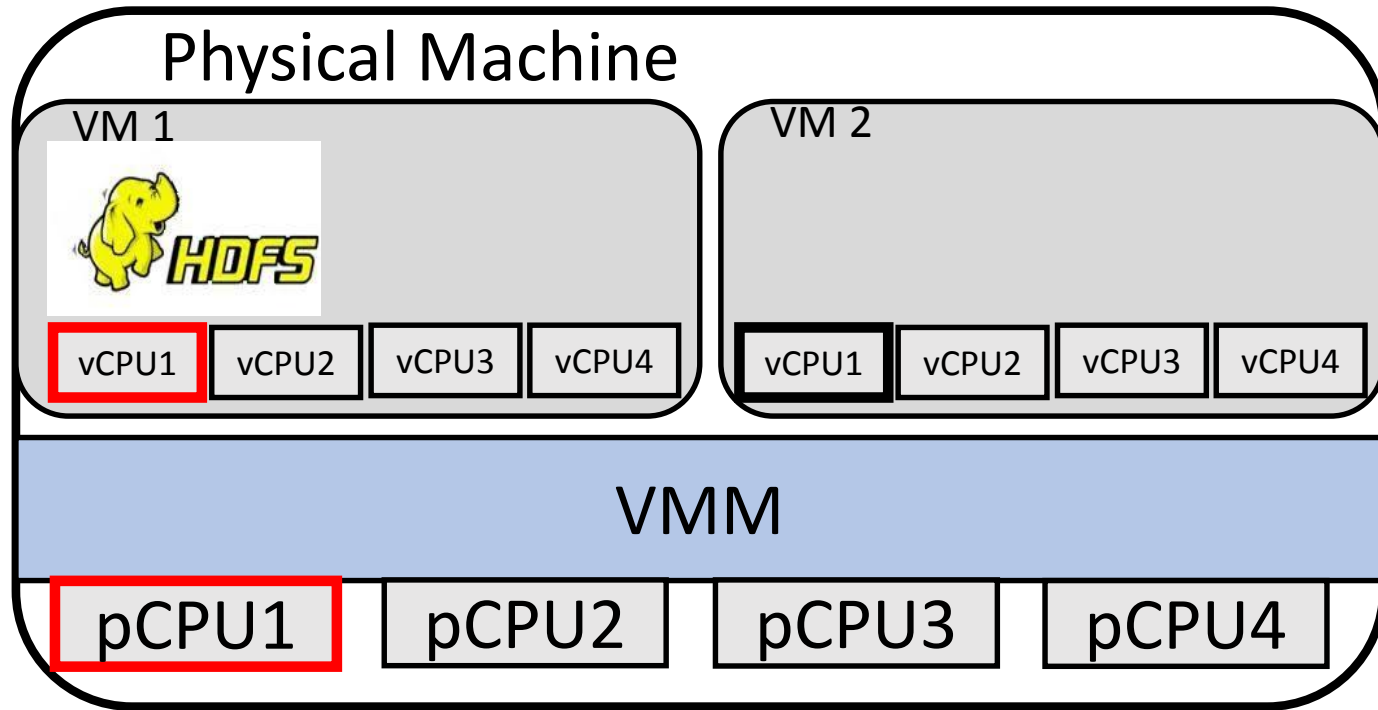
- Physical CPU (pCPU): hardware resources
- Virtual CPU (vCPU): processors in VM, threads in VMM
- vCPU scheduler schedules and deschedules vCPUs periodically

# Multiple vCPUs are scheduled to time-share a pCPU

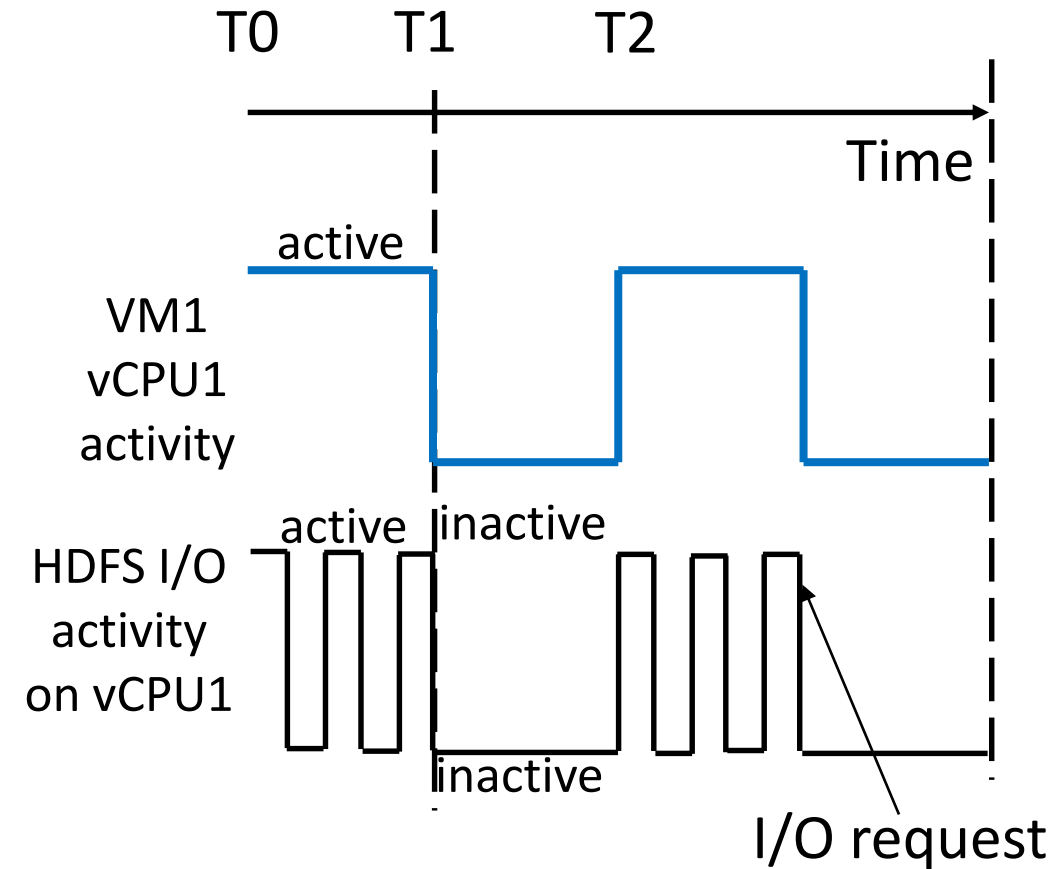
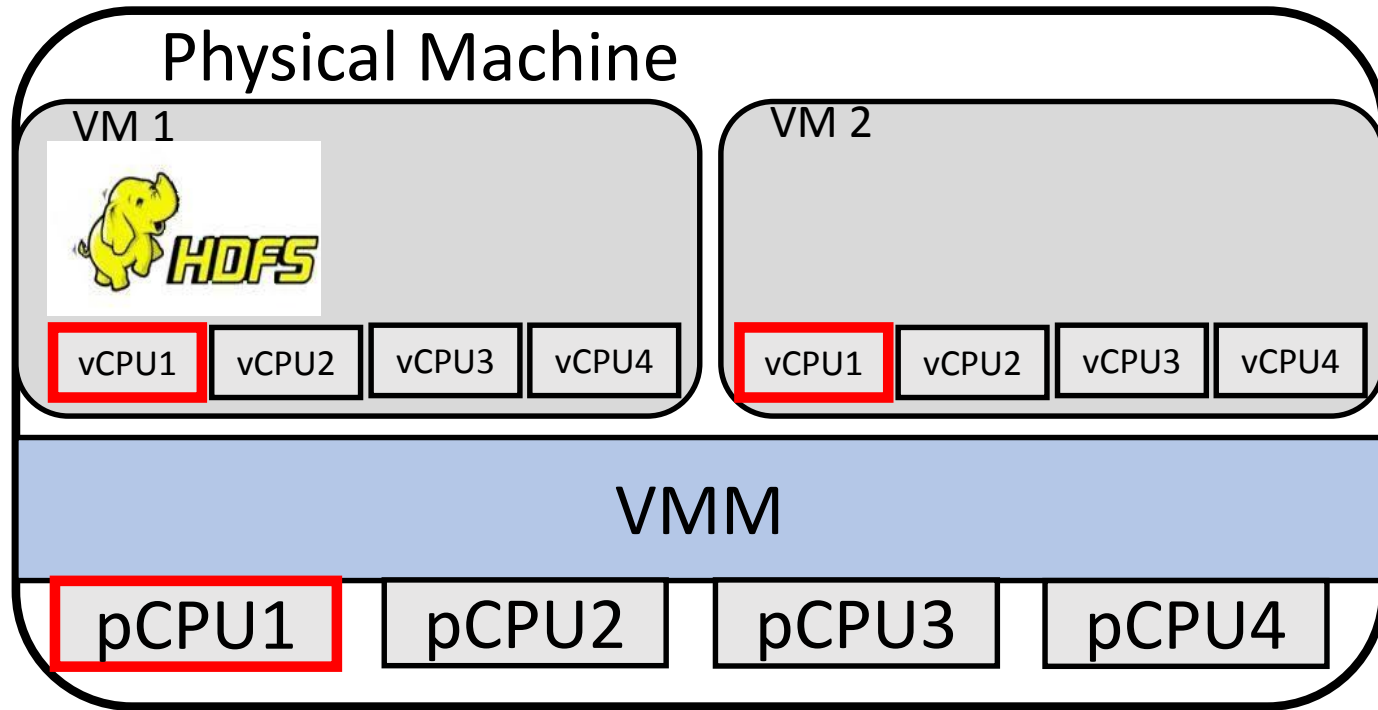


- Physical CPU (pCPU): hardware resources
- Virtual CPU (vCPU): processors in VM, threads in VMM
- vCPU scheduler schedules and deschedules vCPUs periodically

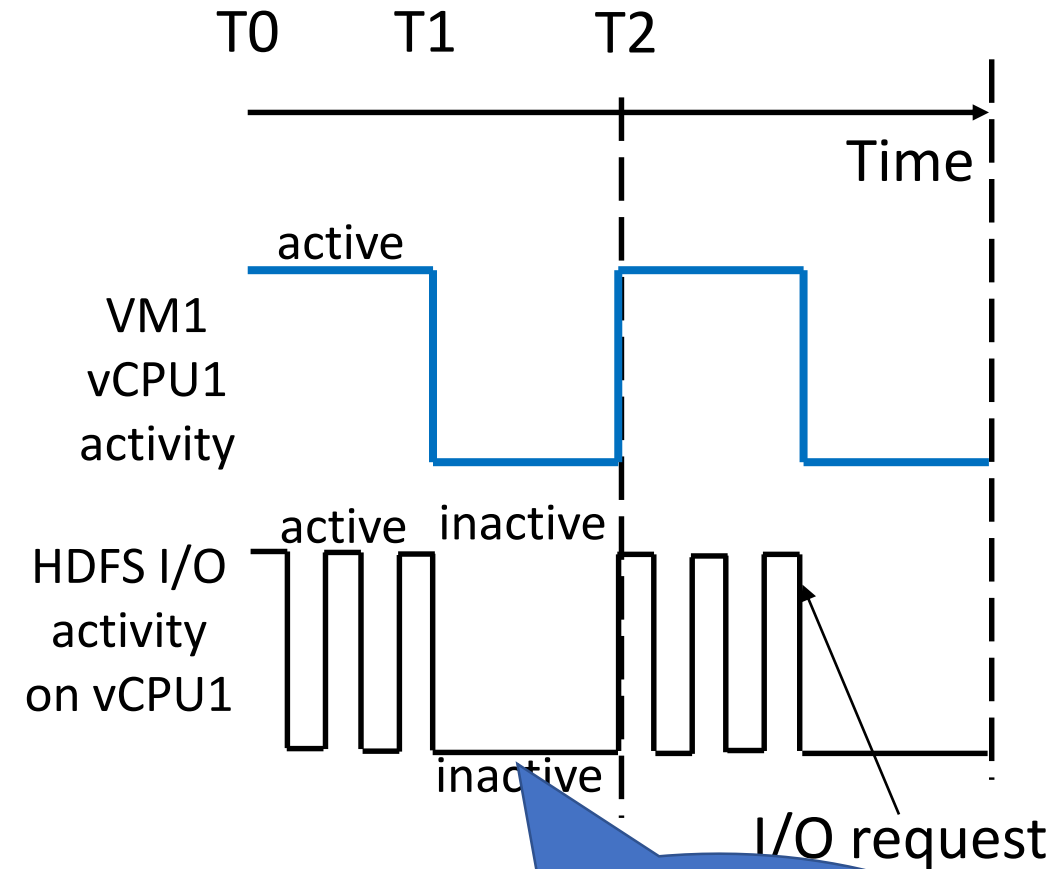
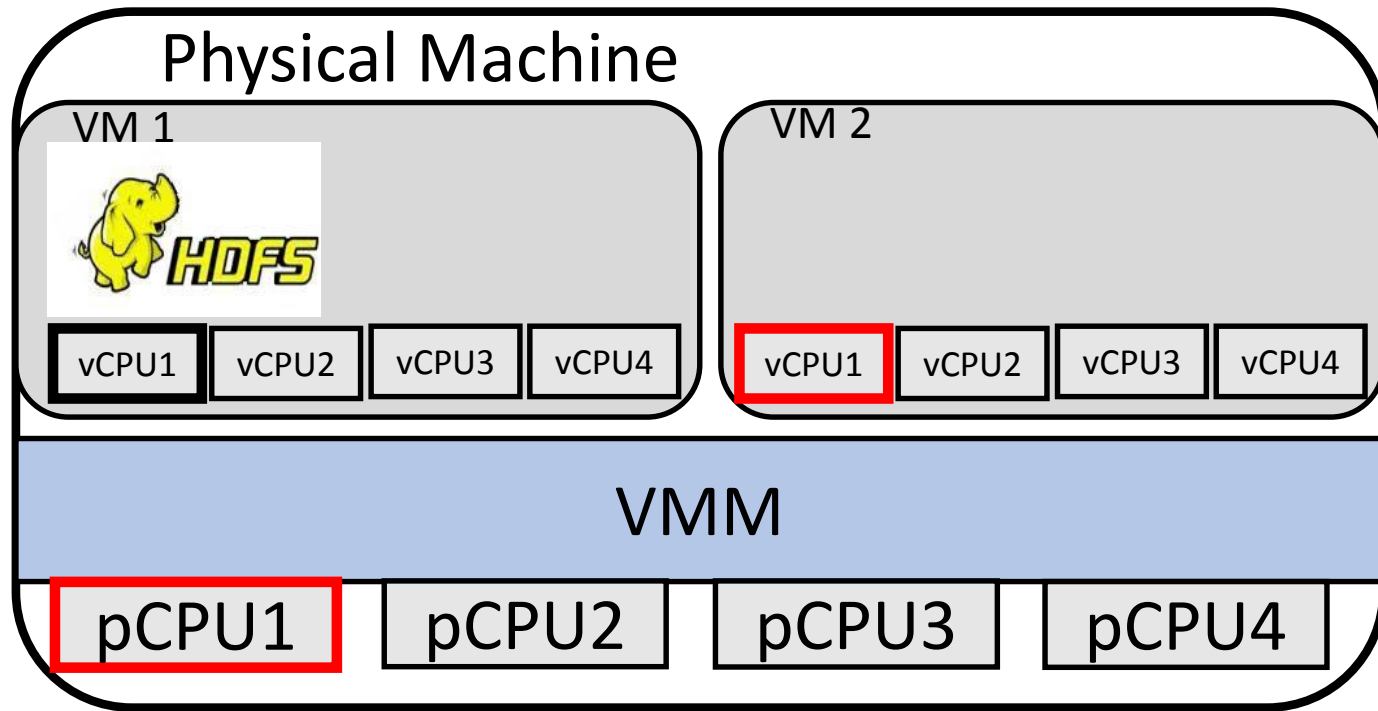
# An understudied problem: I/O inactivity



# An understudied problem: I/O inactivity



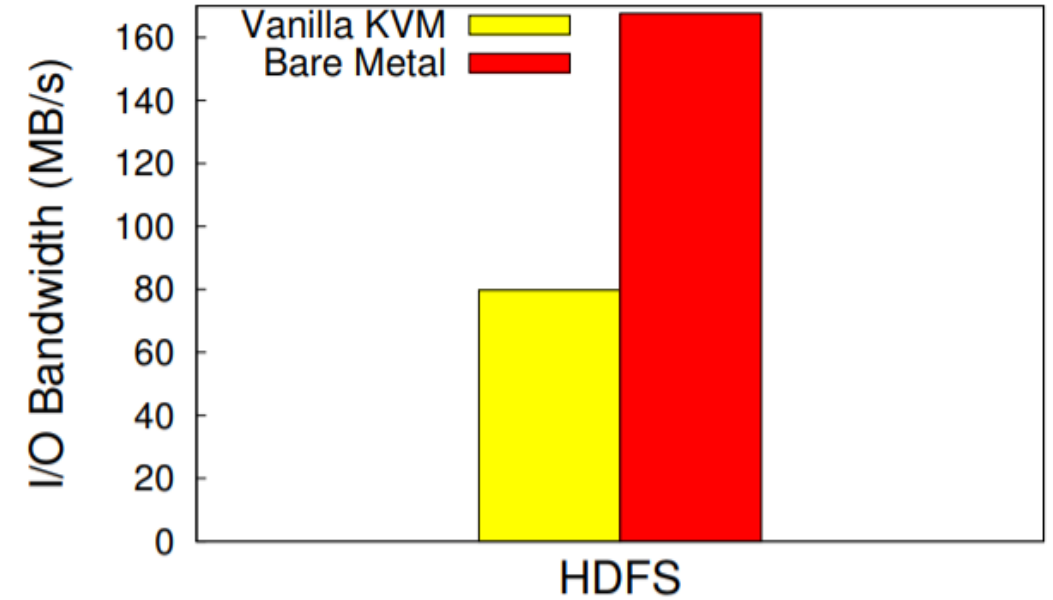
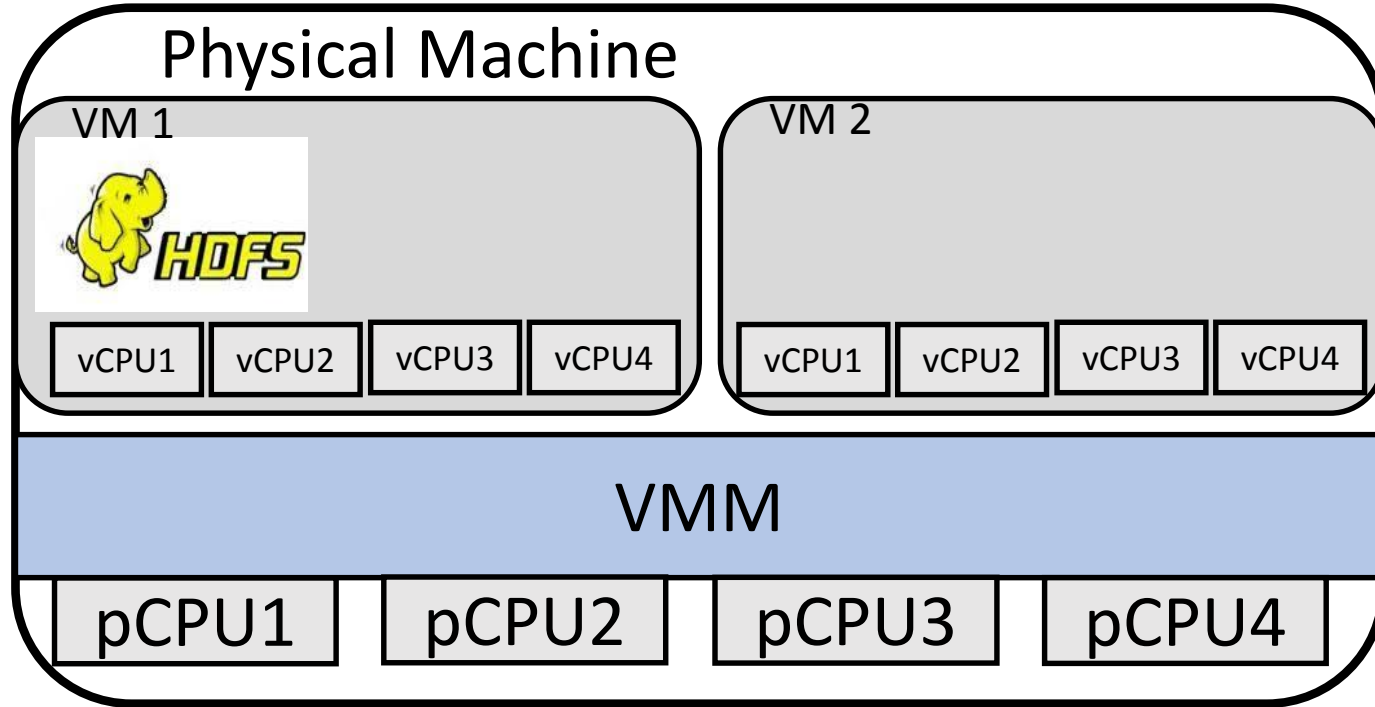
# An understudied problem: I/O inactivity



No I/O requests can be issued on an inactive vCPU

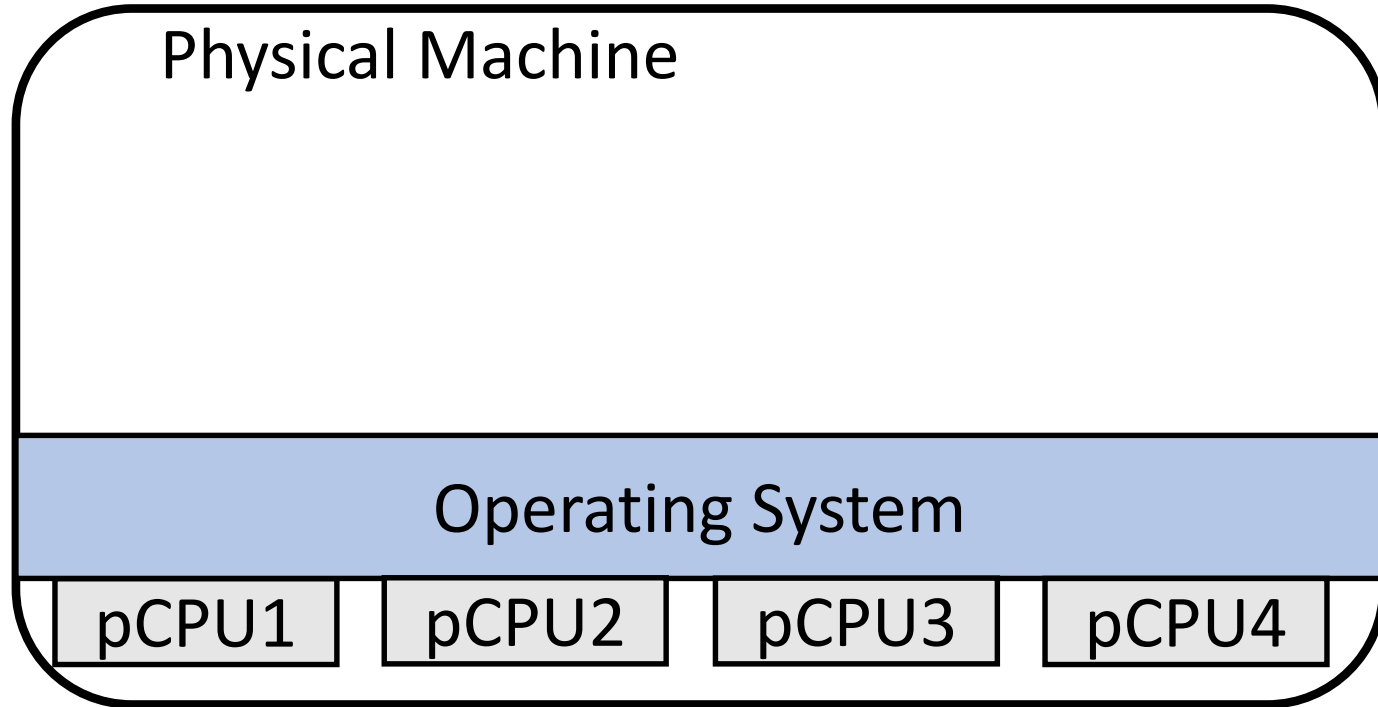
Underutilization  
of I/O device

# I/O inactivity causes low I/O performance in VMs

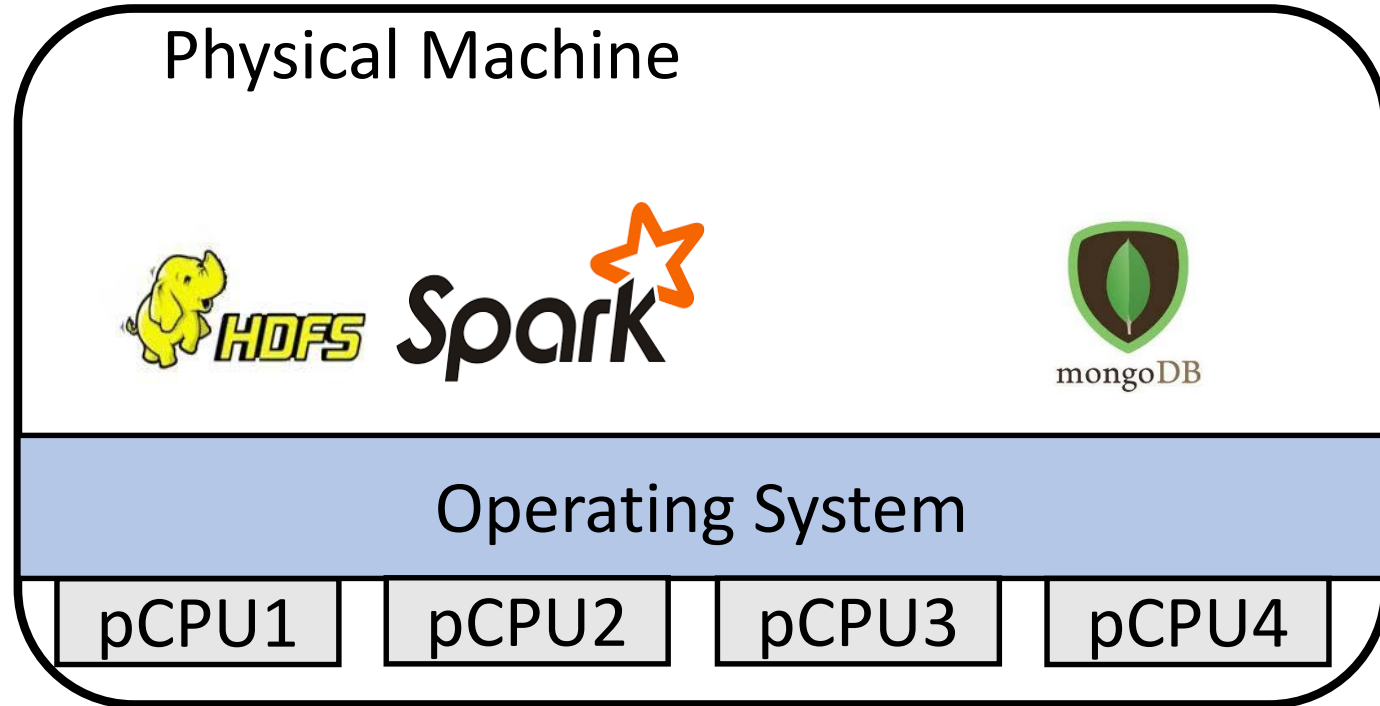


- VM1 can only use 50% of I/O bandwidth even when VM2 does not use I/O device.
- I/O throughput of HDFS in VM1 is only 55% of that on bare-metal.

I/O scheduler cannot effectively enforce fairness between VMs

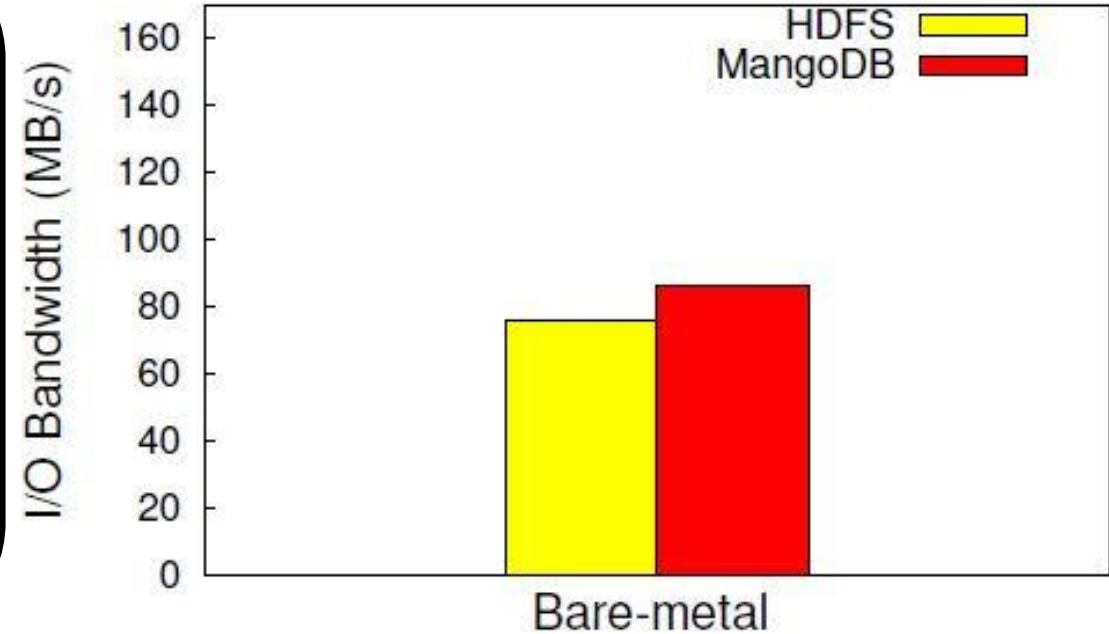
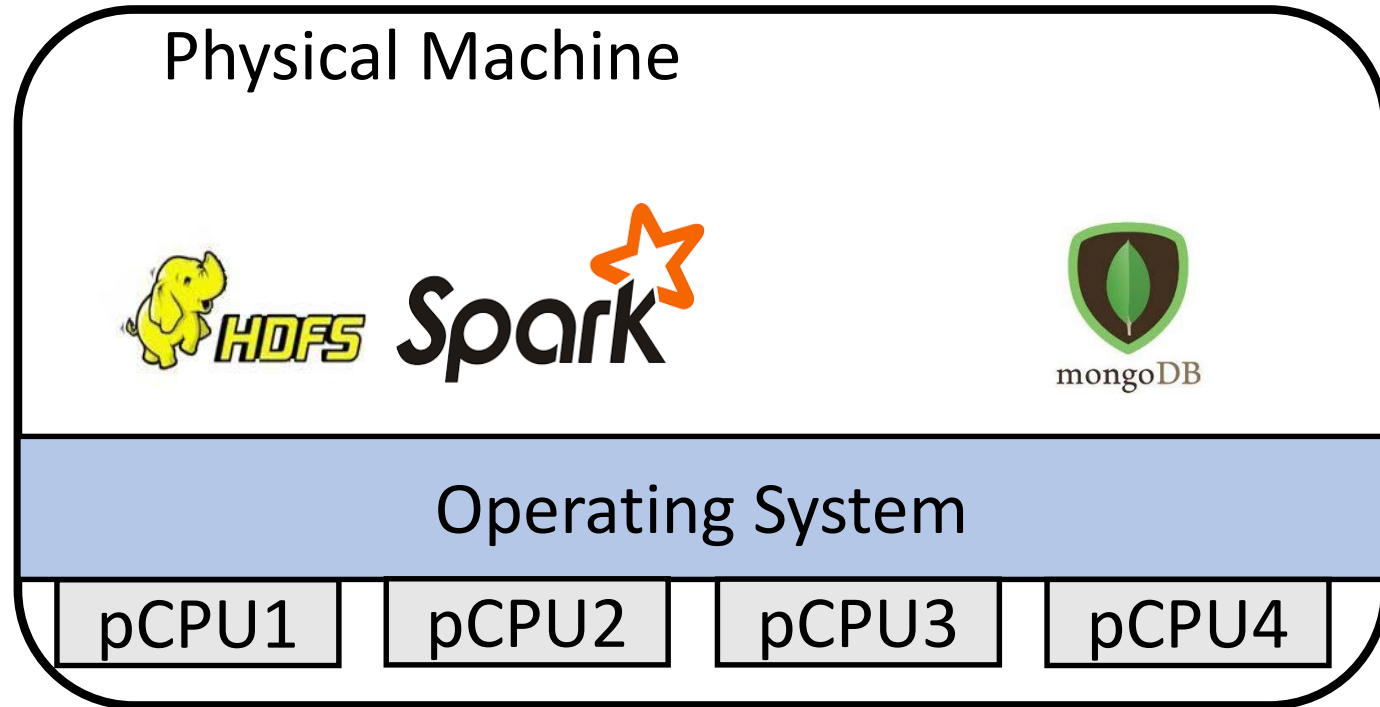


# I/O scheduler cannot effectively enforce fairness between VMs

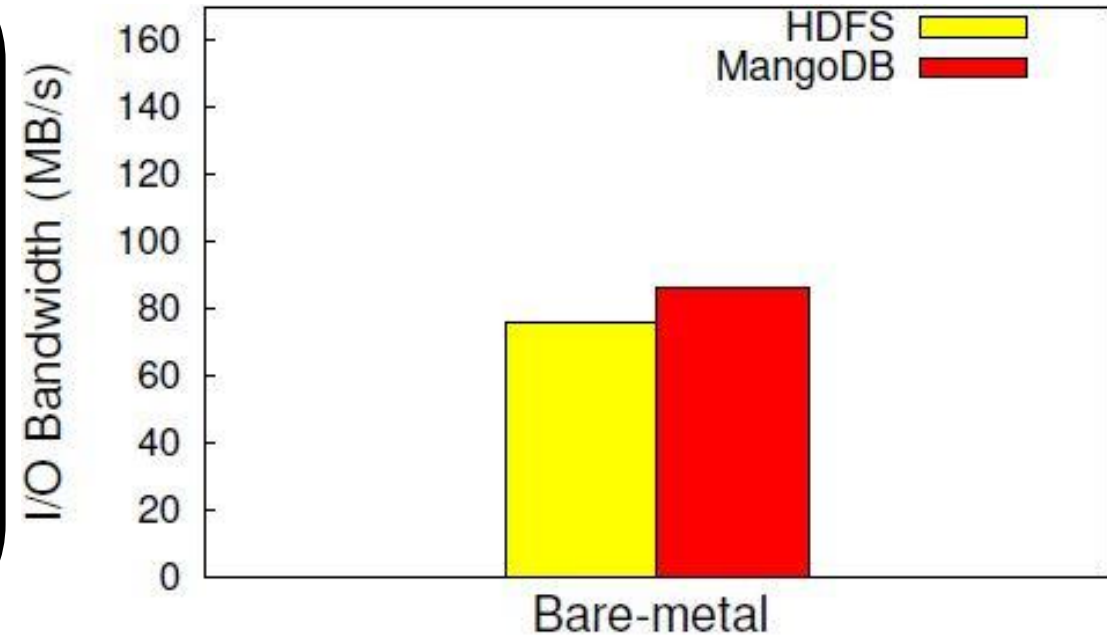
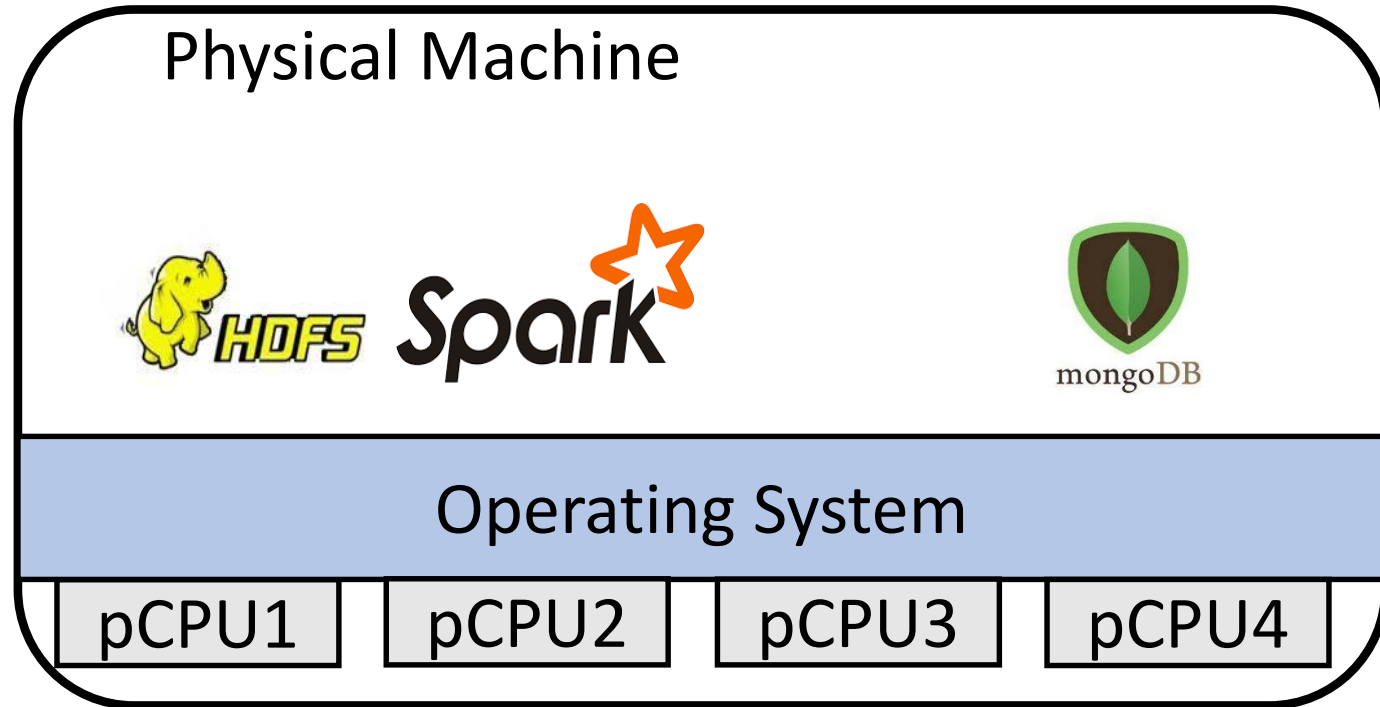




# I/O scheduler cannot effectively enforce fairness between VMs

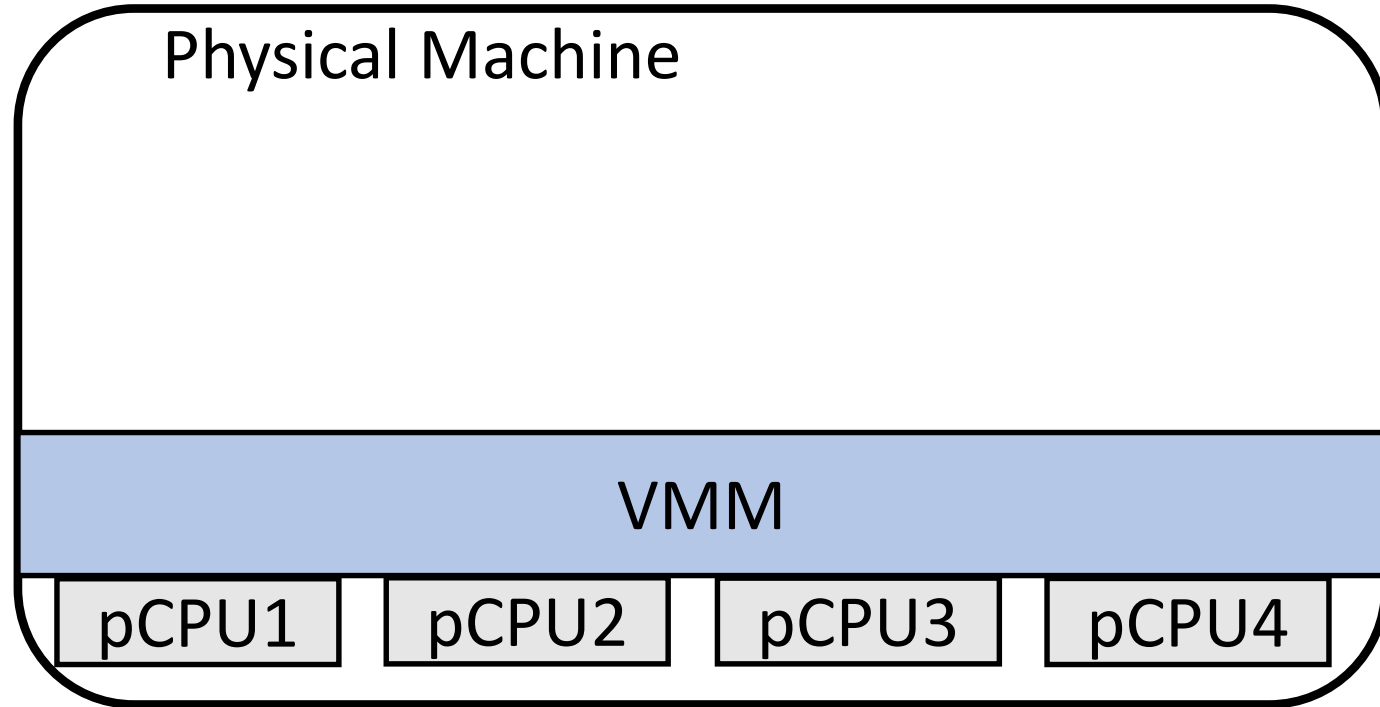


# I/O scheduler cannot effectively enforce fairness between VMs

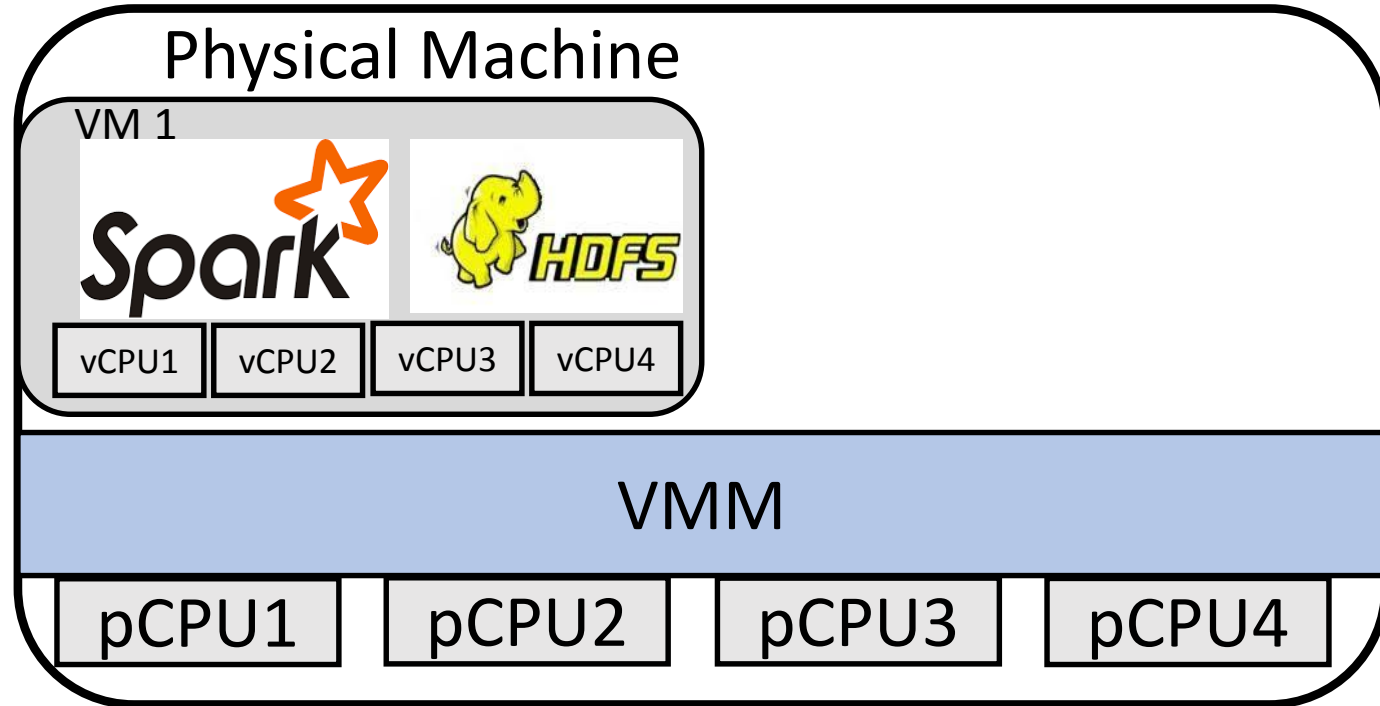


Two I/O-intensive applications (HDFS and MongoDB) show similar I/O throughput on bare-metal

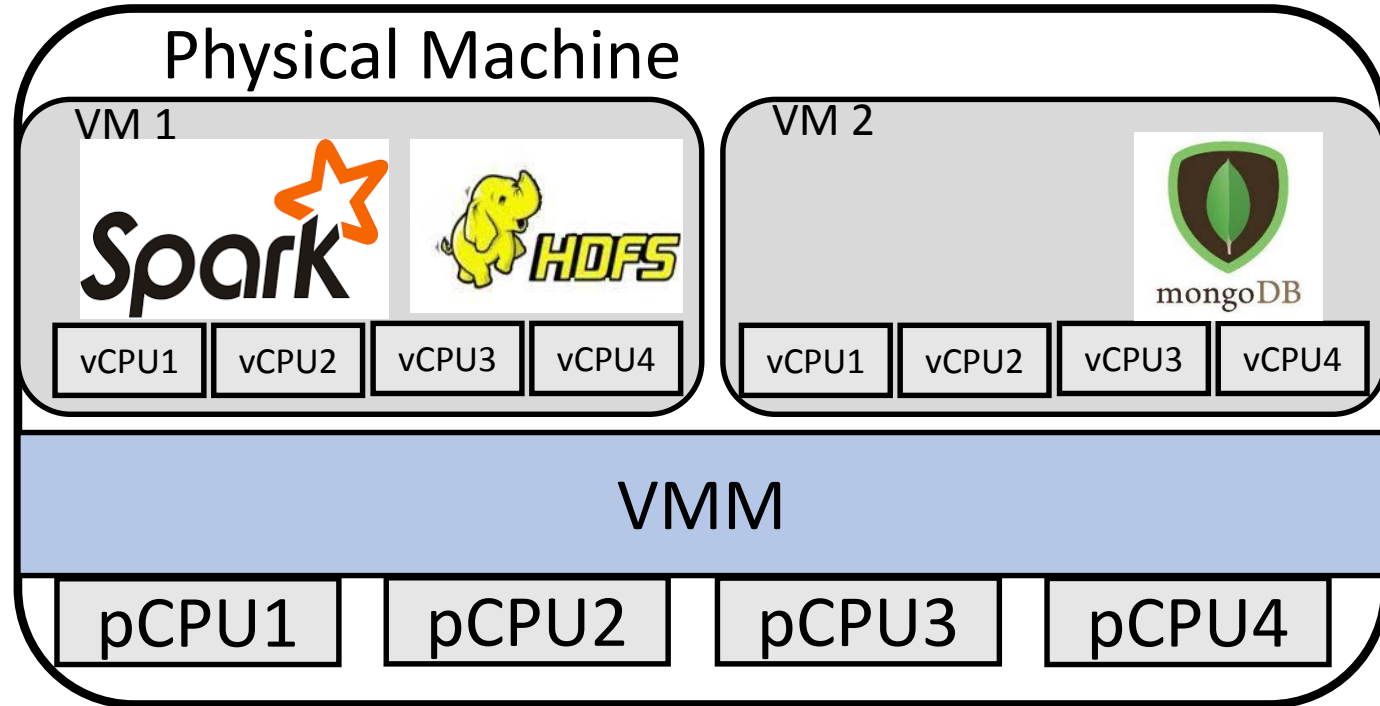
I/O scheduler cannot effectively enforce fairness between VMs



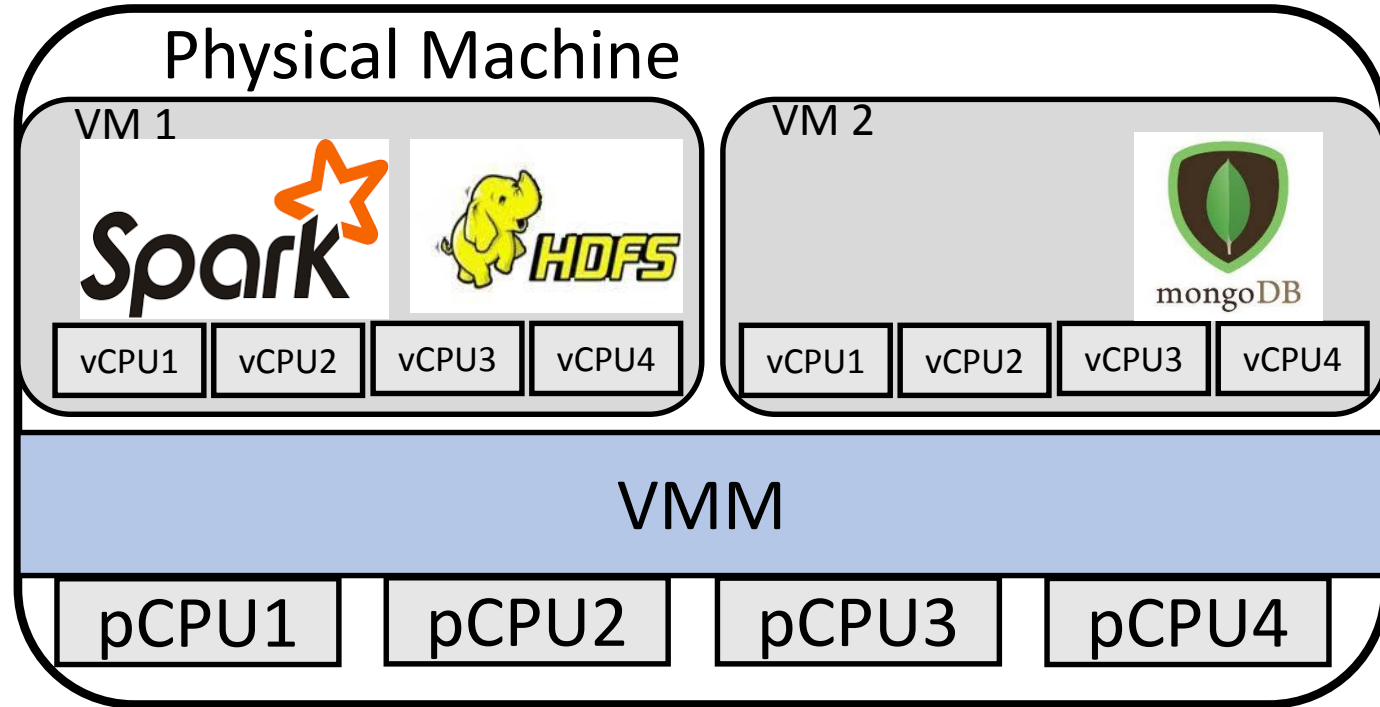
# I/O scheduler cannot effectively enforce fairness between VMs



# I/O scheduler cannot effectively enforce fairness between VMs

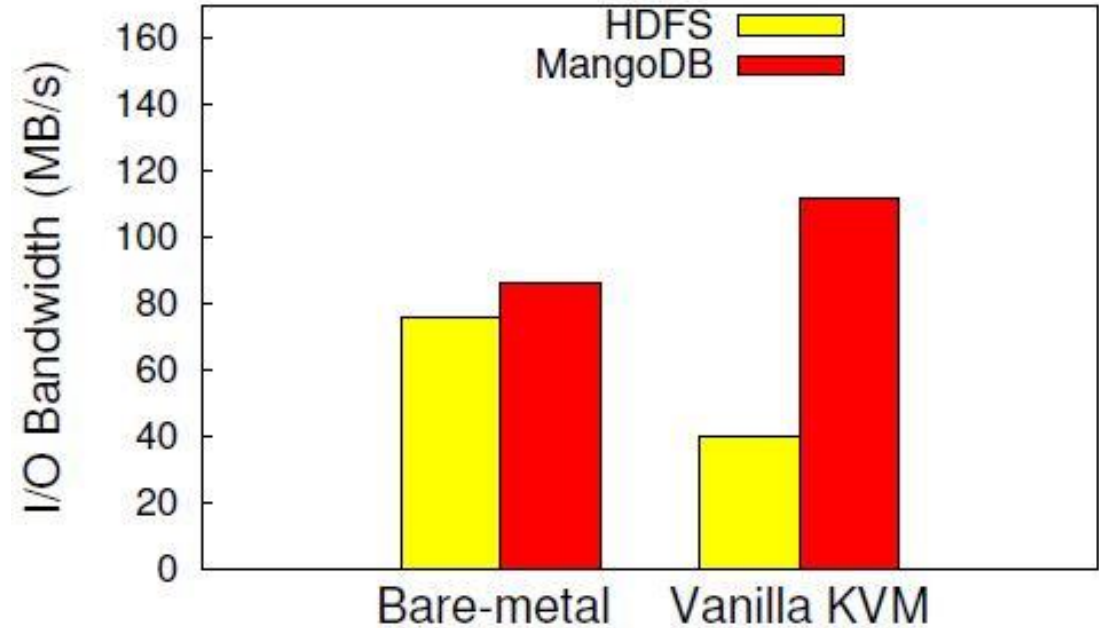
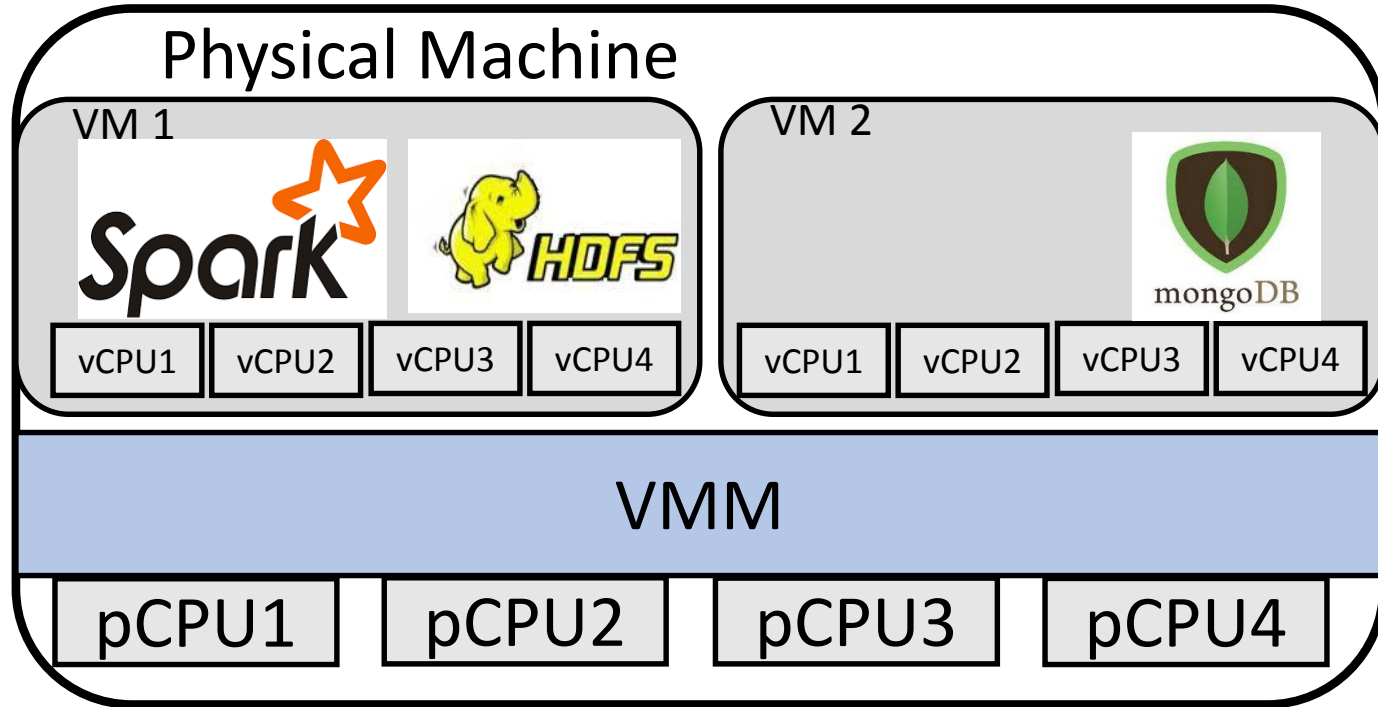


# I/O scheduler cannot effectively enforce fairness between VMs



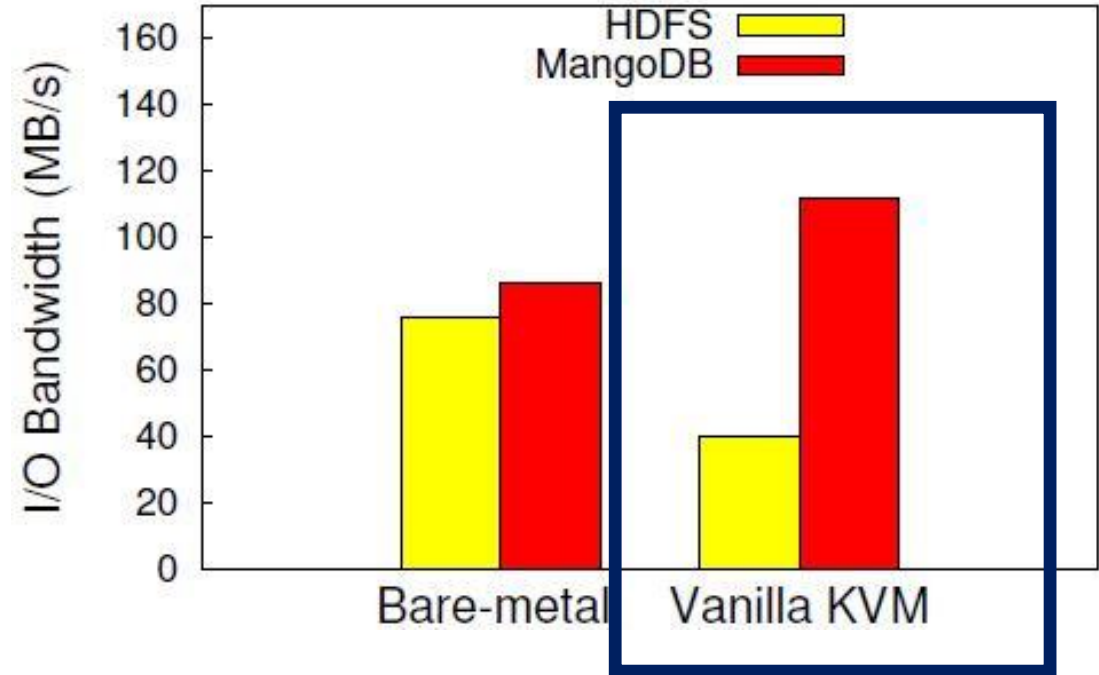
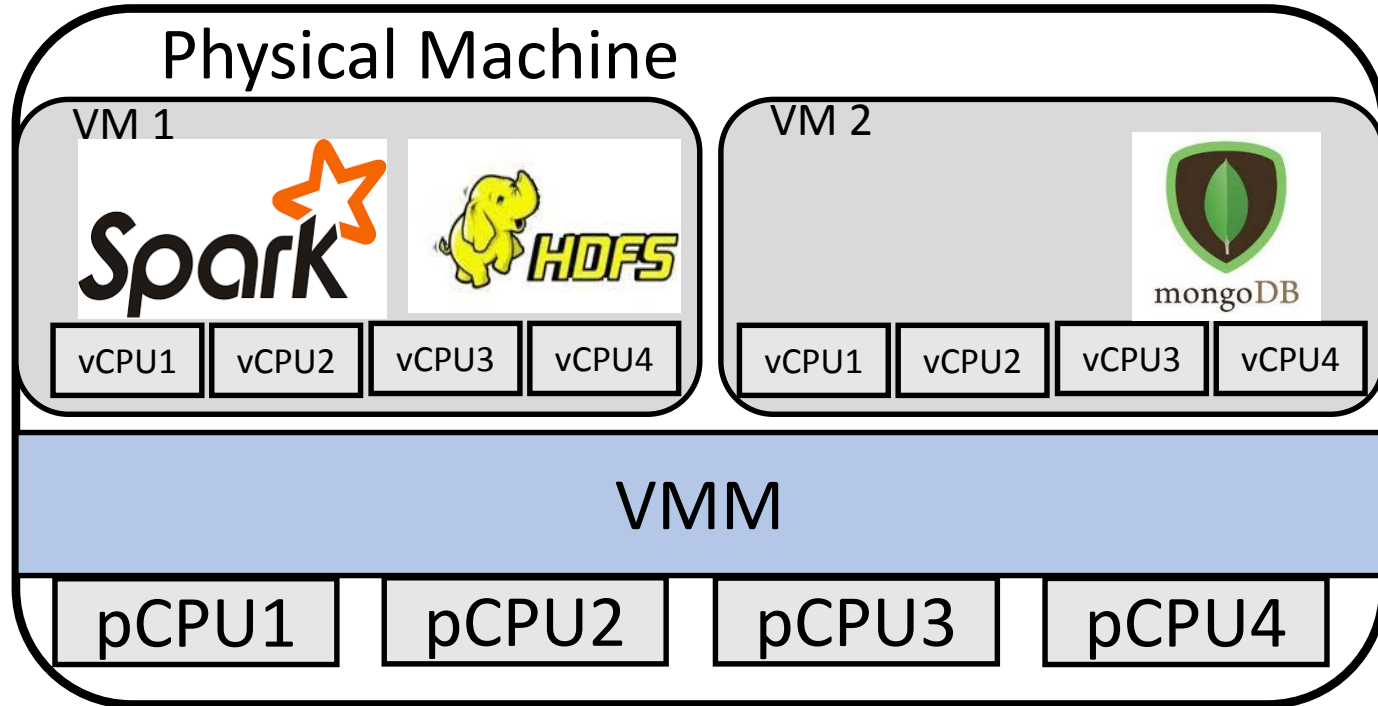
VM1 and VM2 are assigned with the same I/O priority, but MongoDB (VM2) achieves much higher I/O throughput.

# I/O scheduler cannot effectively enforce fairness between VMs



VM1 and VM2 are assigned with the same I/O priority, but MongoDB (VM2) achieves much higher I/O throughput.

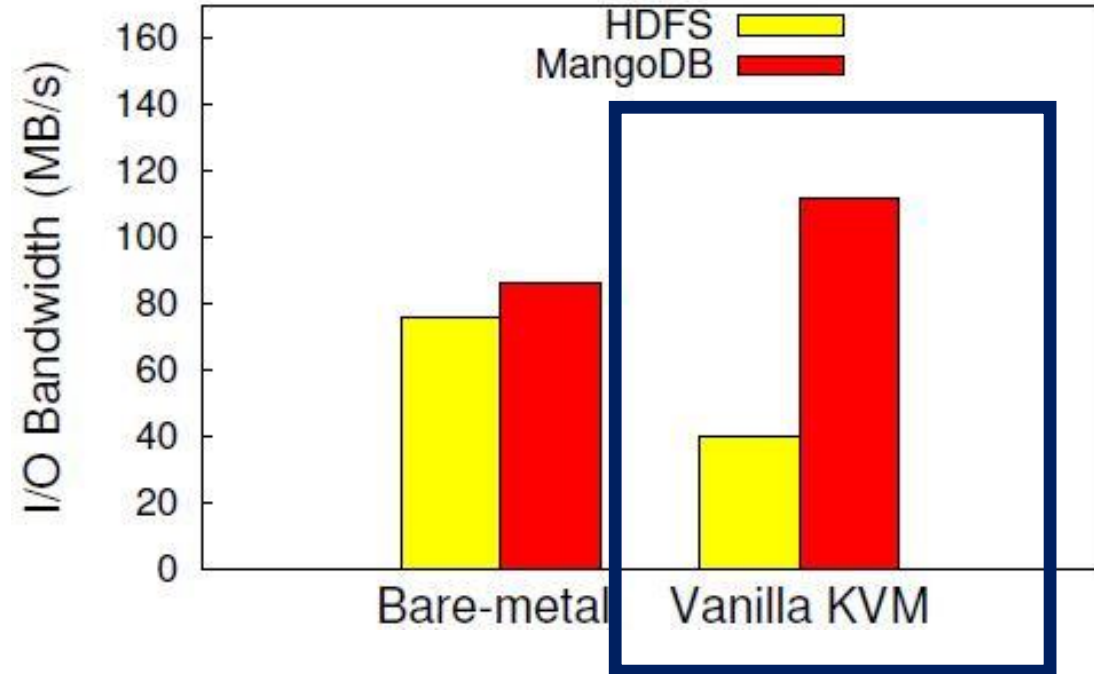
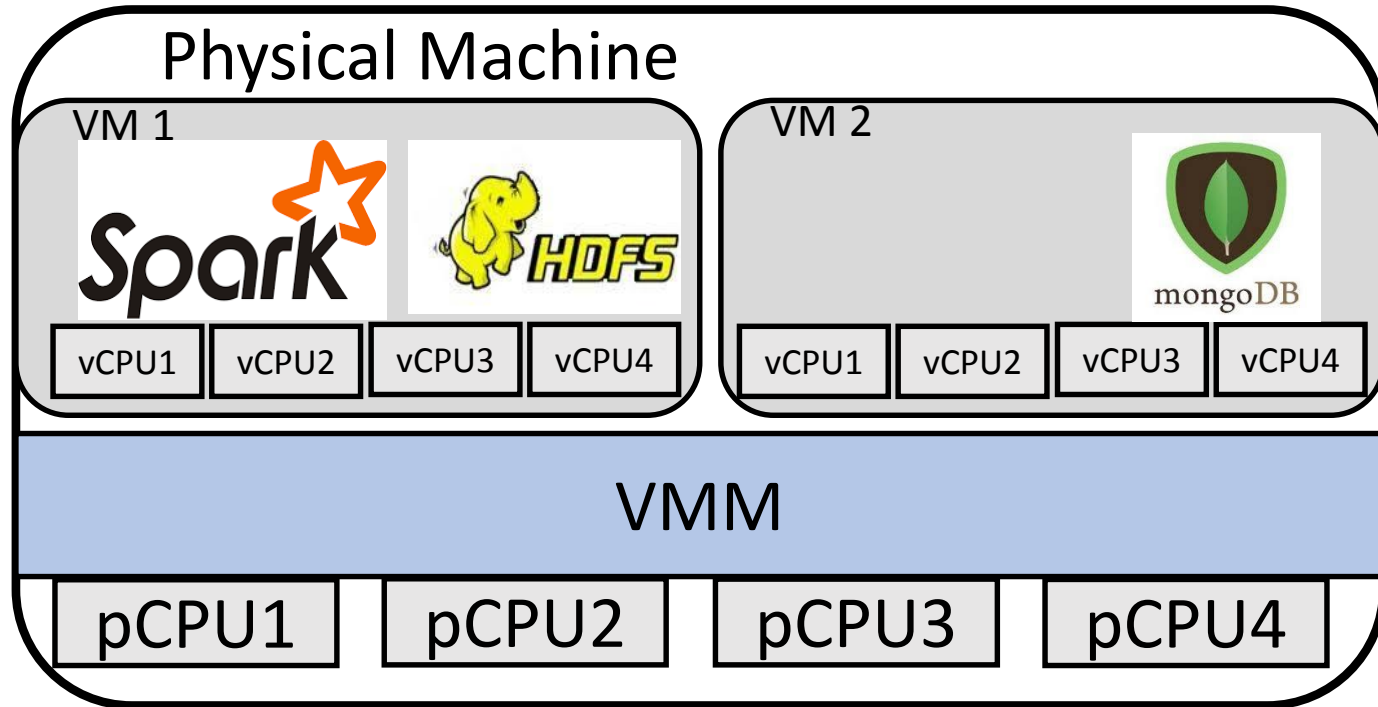
# I/O scheduler cannot effectively enforce fairness between VMs



VM1 and VM2 are assigned with the same I/O priority, but MongoDB (VM2) achieves much higher I/O throughput.



# I/O scheduler cannot effectively enforce fairness between VMs



VM1 and VM2 are assigned with the same I/O priority, but MongoDB (VM2) achieves much higher I/O throughput.

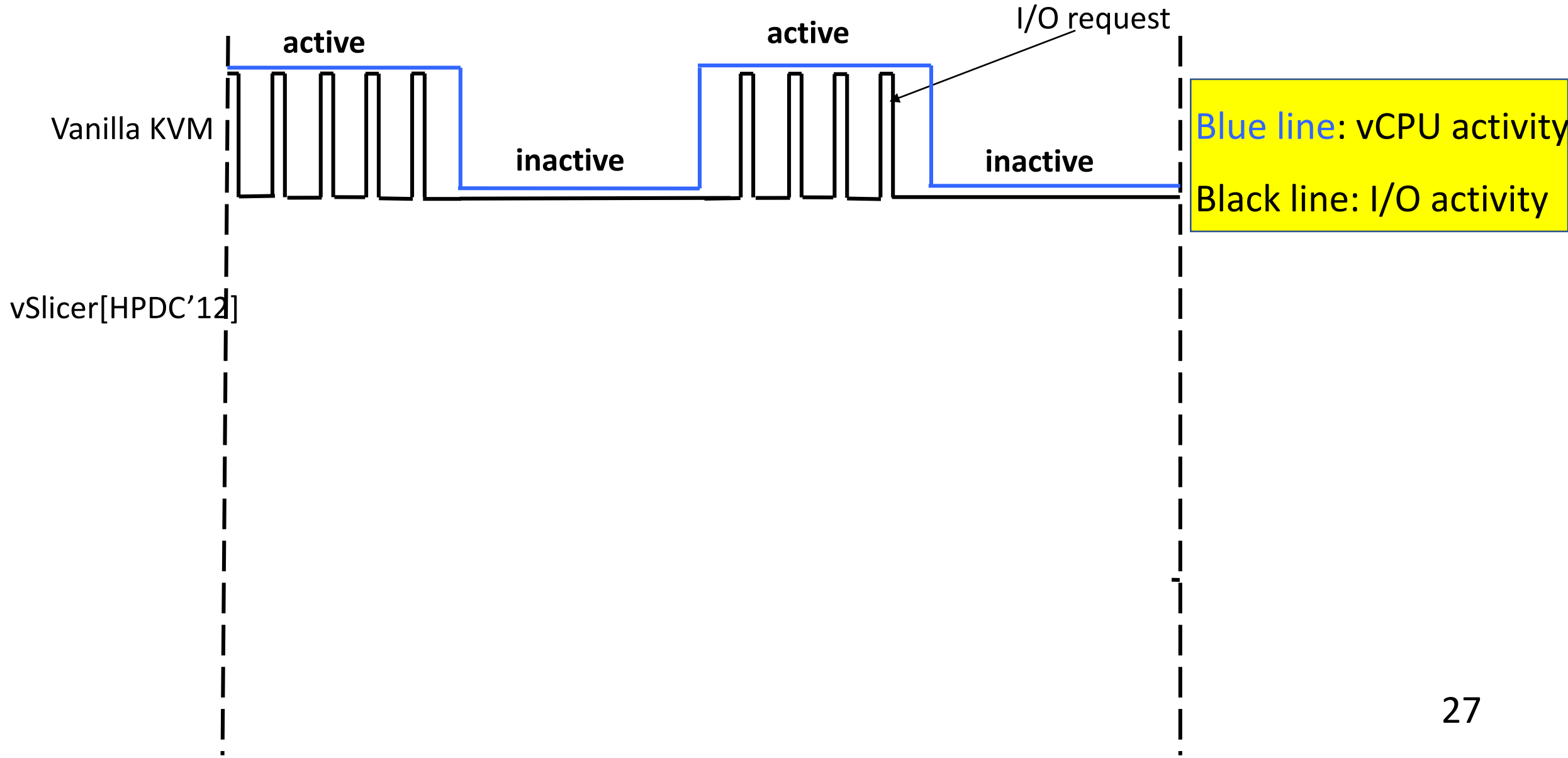
- vCPUs running I/O tasks in MongoDB in VM2 have more time slice to run.
- vCPUs running HDFS in VM1 have less time slice to run

# Key: decoupling I/O activities from vCPU scheduling

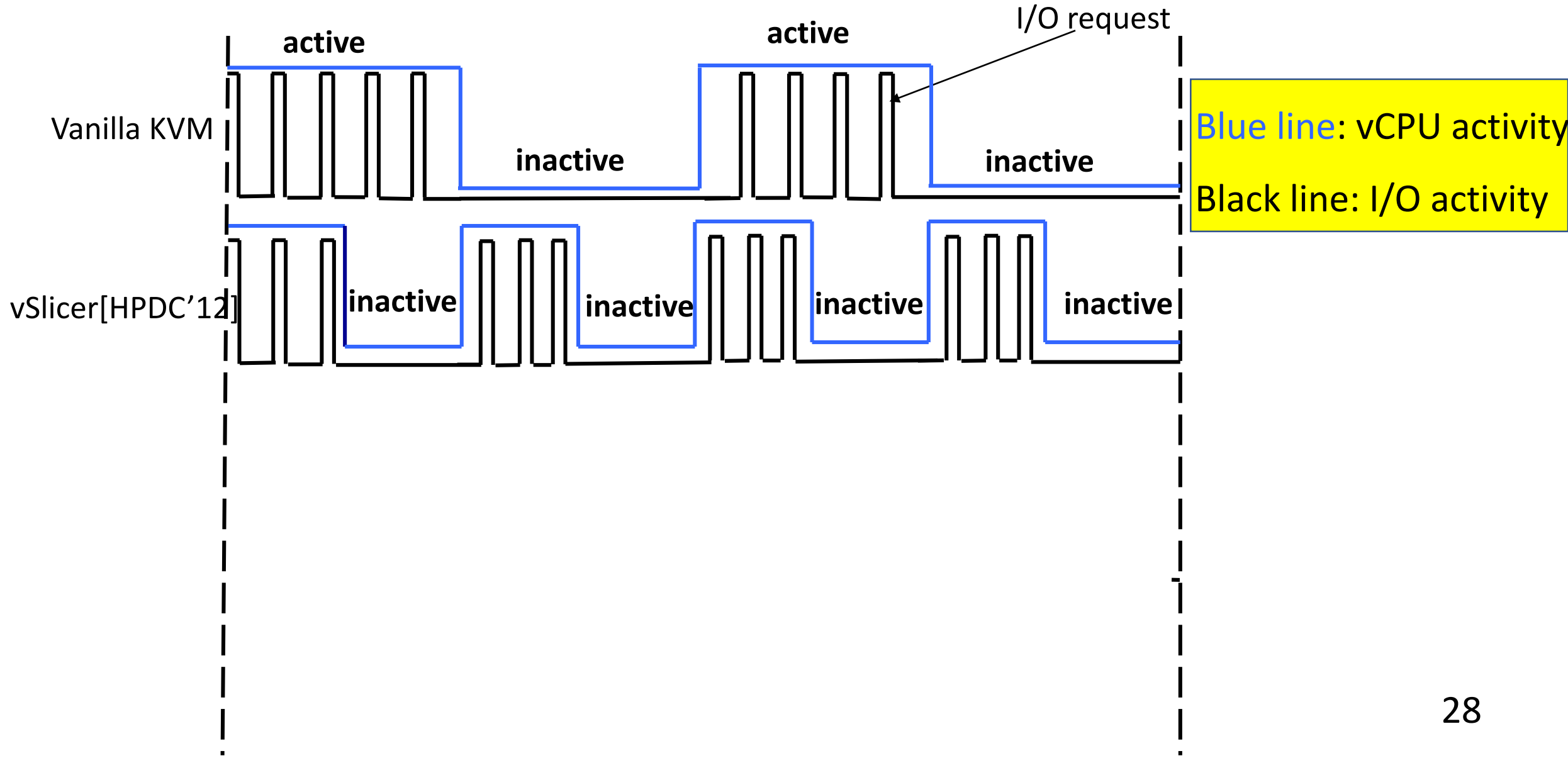
vSlicer[HPDC'12]

The diagram consists of two vertical dashed lines. The left line is positioned to the right of the text 'vSlicer[HPDC'12]'. The right line is positioned to the right of the page number '26'. The lines are parallel and extend across most of the vertical height of the slide.

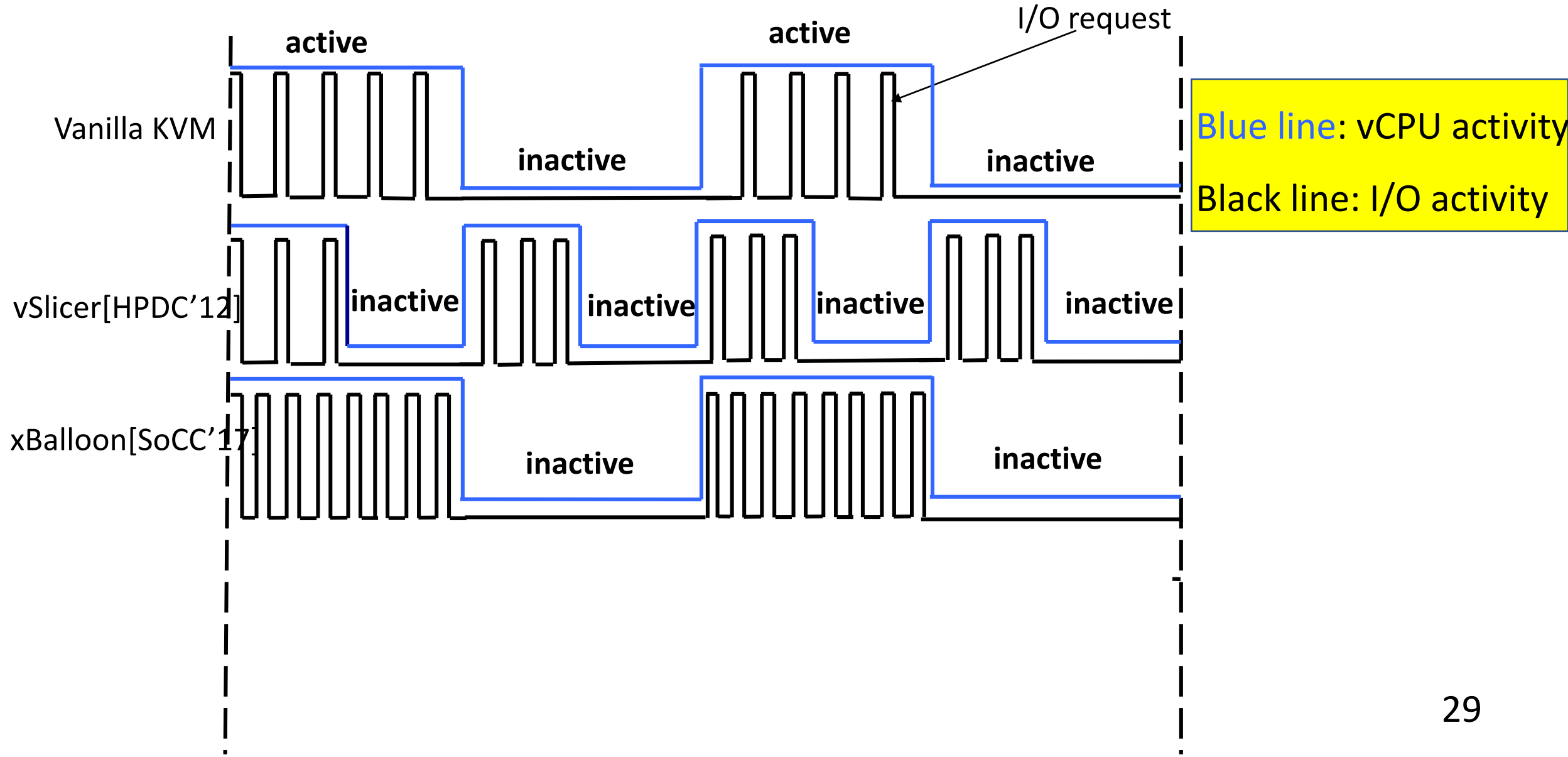
# Key: decoupling I/O activities from vCPU scheduling



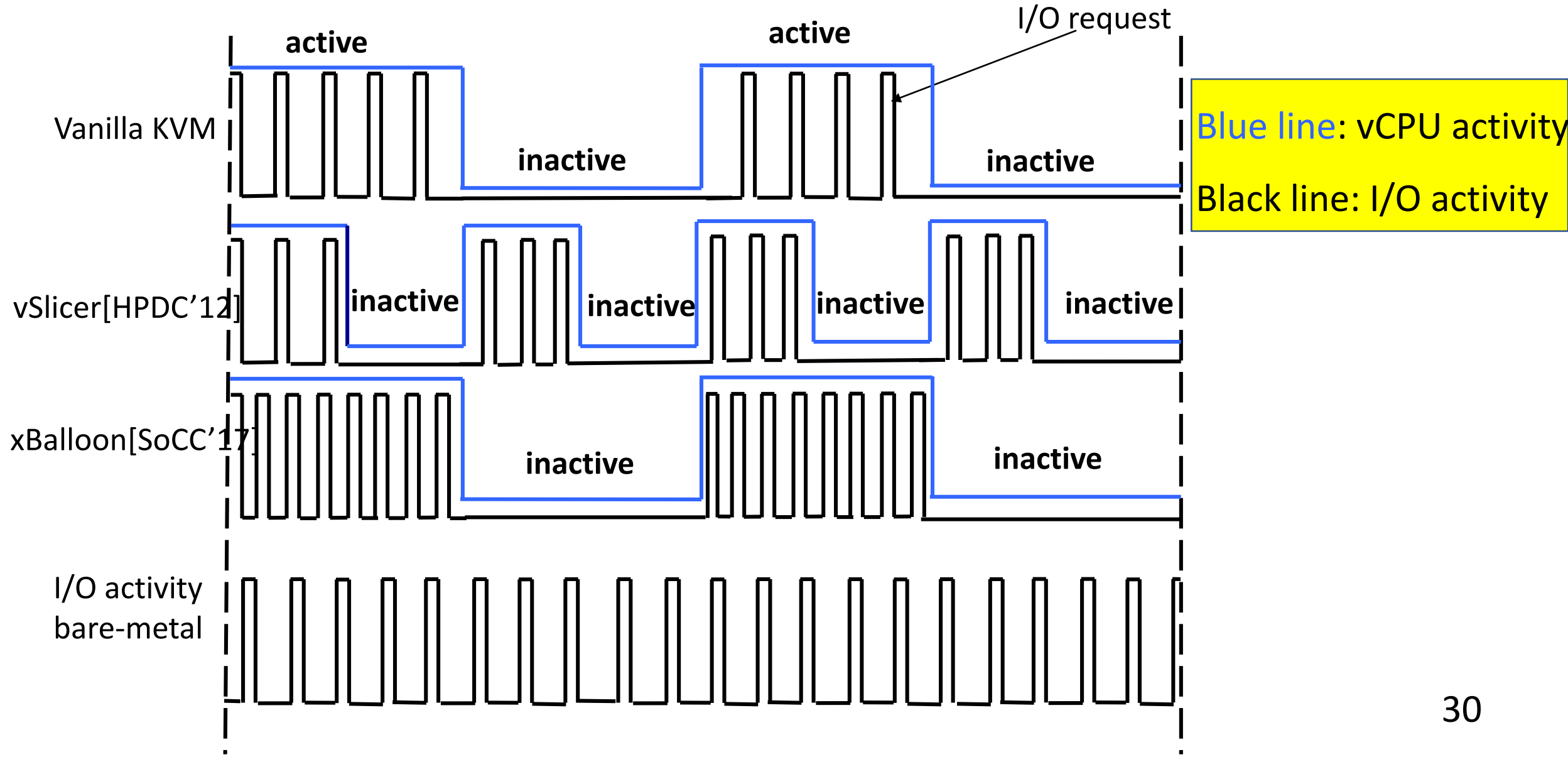
# Key: decoupling I/O activities from vCPU scheduling



# Key: decoupling I/O activities from vCPU scheduling



# Key: decoupling I/O activities from vCPU scheduling



# Can we decouple I/O activities from vCPU scheduling?

- Our answer is YES!

# Can we decouple I/O activities from vCPU scheduling?

- Our answer is YES!
- Key observation: each VM often has active vCPUs



# Can we decouple I/O activities from vCPU scheduling?

- Our answer is YES!
- Key observation: each VM often has active vCPUs
- Solution: keep I/O tasks on active vCPUs
  - Migrate an I/O task when its vCPU is about to be descheduled

# Can we decouple I/O activities from vCPU scheduling?

- Our answer is YES!
- Key observation: each VM often has active vCPUs
- Solution: keep I/O tasks on active vCPUs
  - Migrate an I/O task when its vCPU is about to be descheduled
  - Migrate the I/O task to a vCPU that is NOT to be descheduled soon.

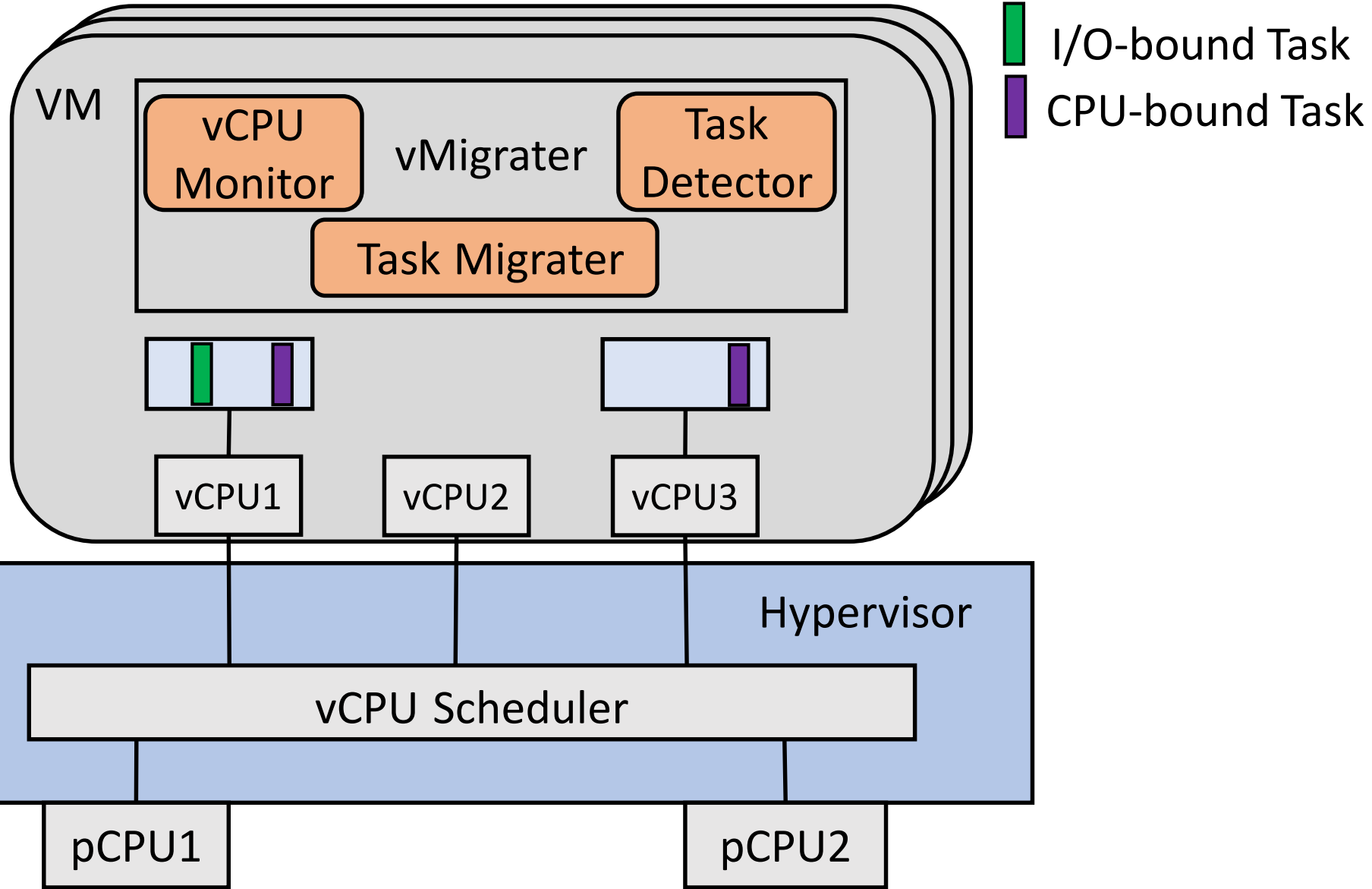
# Can we decouple I/O activities from vCPU scheduling?

- Our answer is YES!
- Key observation: each VM often has active vCPUs
- Solution: keep I/O tasks on active vCPUs
  - Migrate an I/O task when its vCPU is about to be descheduled
  - Migrate the I/O task to a vCPU that is NOT to be descheduled soon.
- Benefits: I/O tasks can make continuous progress like on bare-metal

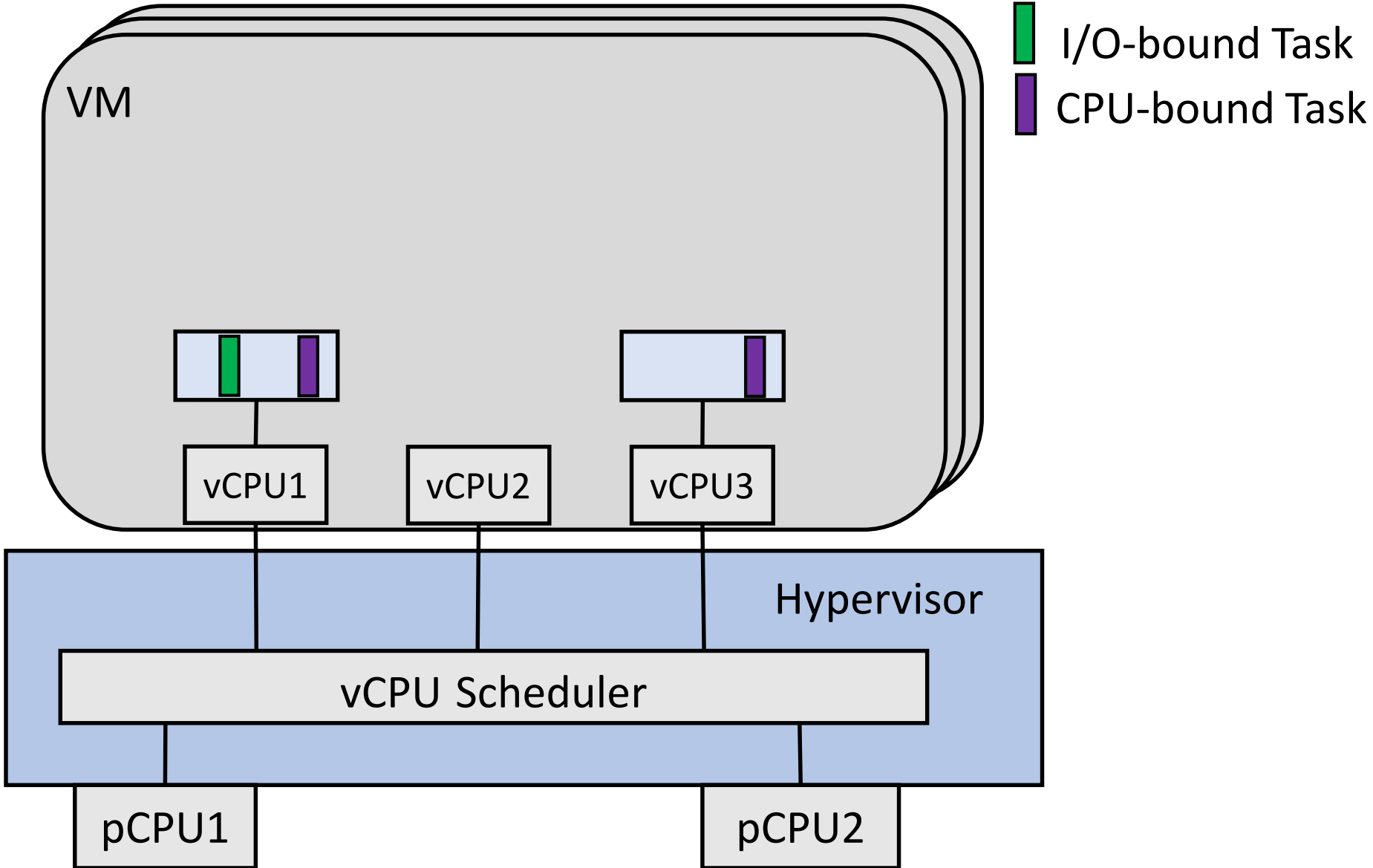
# Can we decouple I/O activities from vCPU scheduling?

- Our answer is YES!
- Key observation: each VM often has active vCPUs
- Solution: keep I/O tasks on active vCPUs
  - Migrate an I/O task when its vCPU is about to be descheduled
  - Migrate the I/O task to a vCPU that is NOT to be descheduled soon.
- Benefits: I/O tasks can make continuous progress like on bare-metal
  - Migration can be efficient because I/O tasks usually have small working sets.

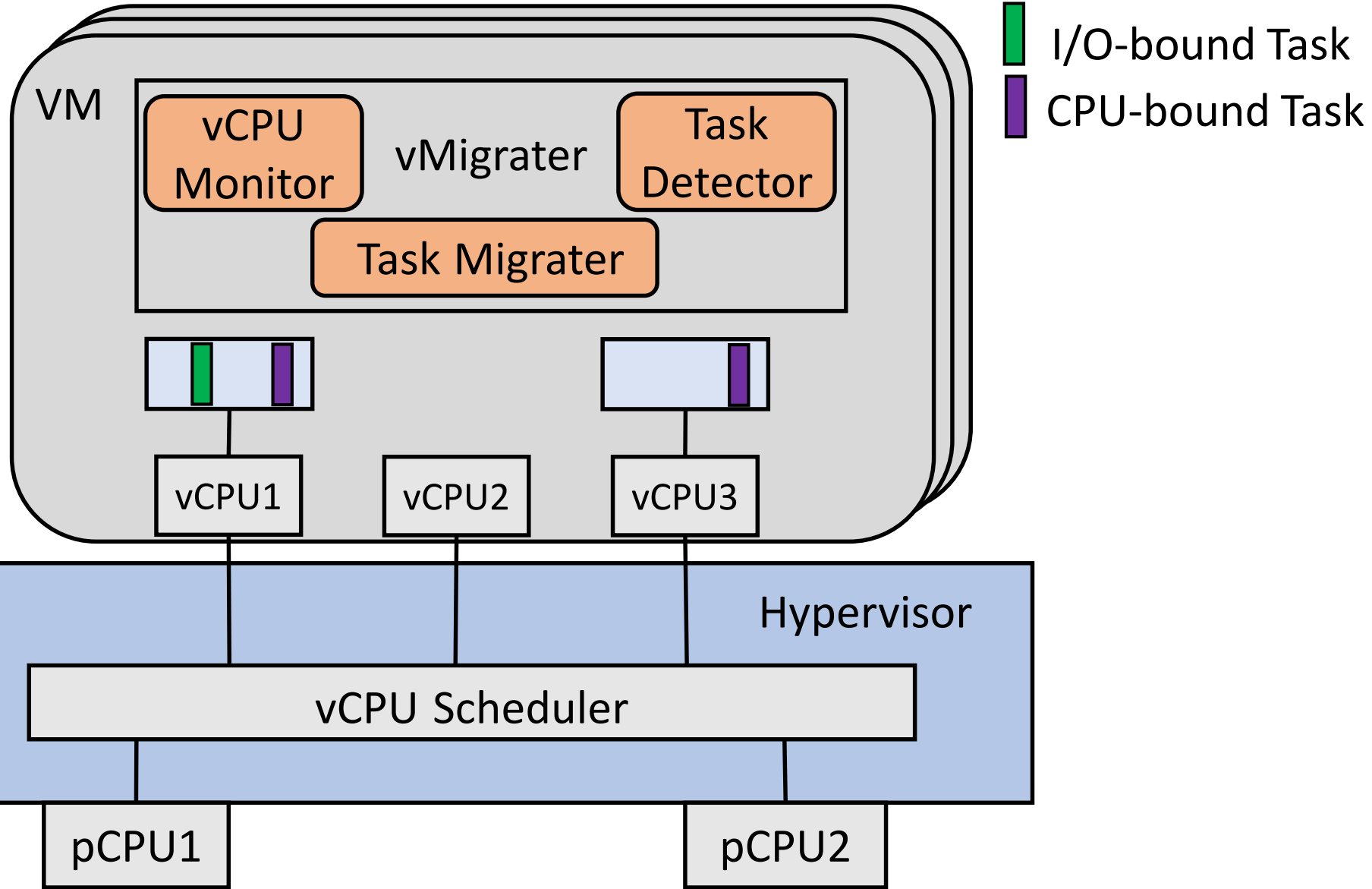
# vMigrater: a user-level design



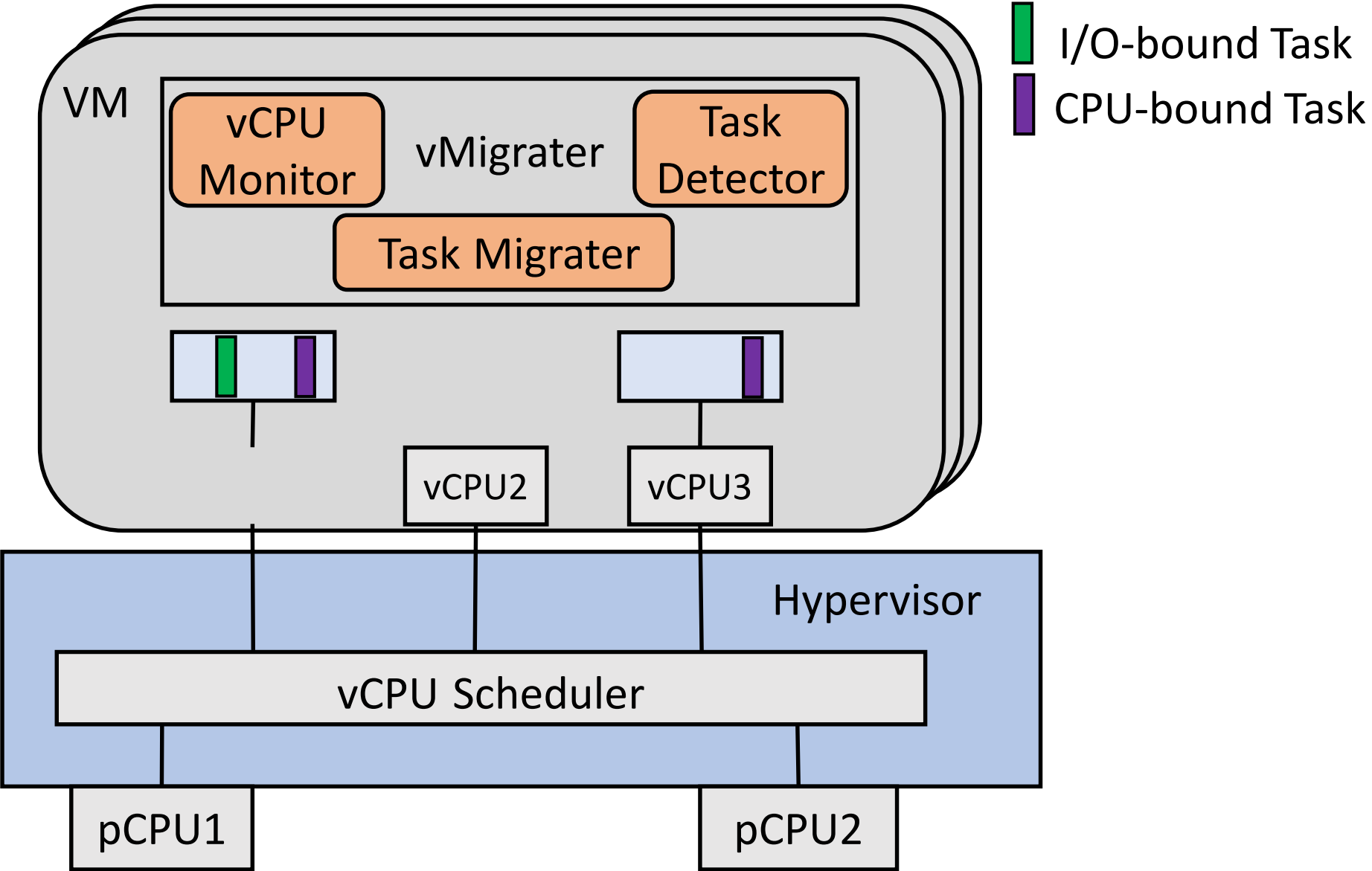
# vMigrater: a user-level design



# vMigrater: a user-level design

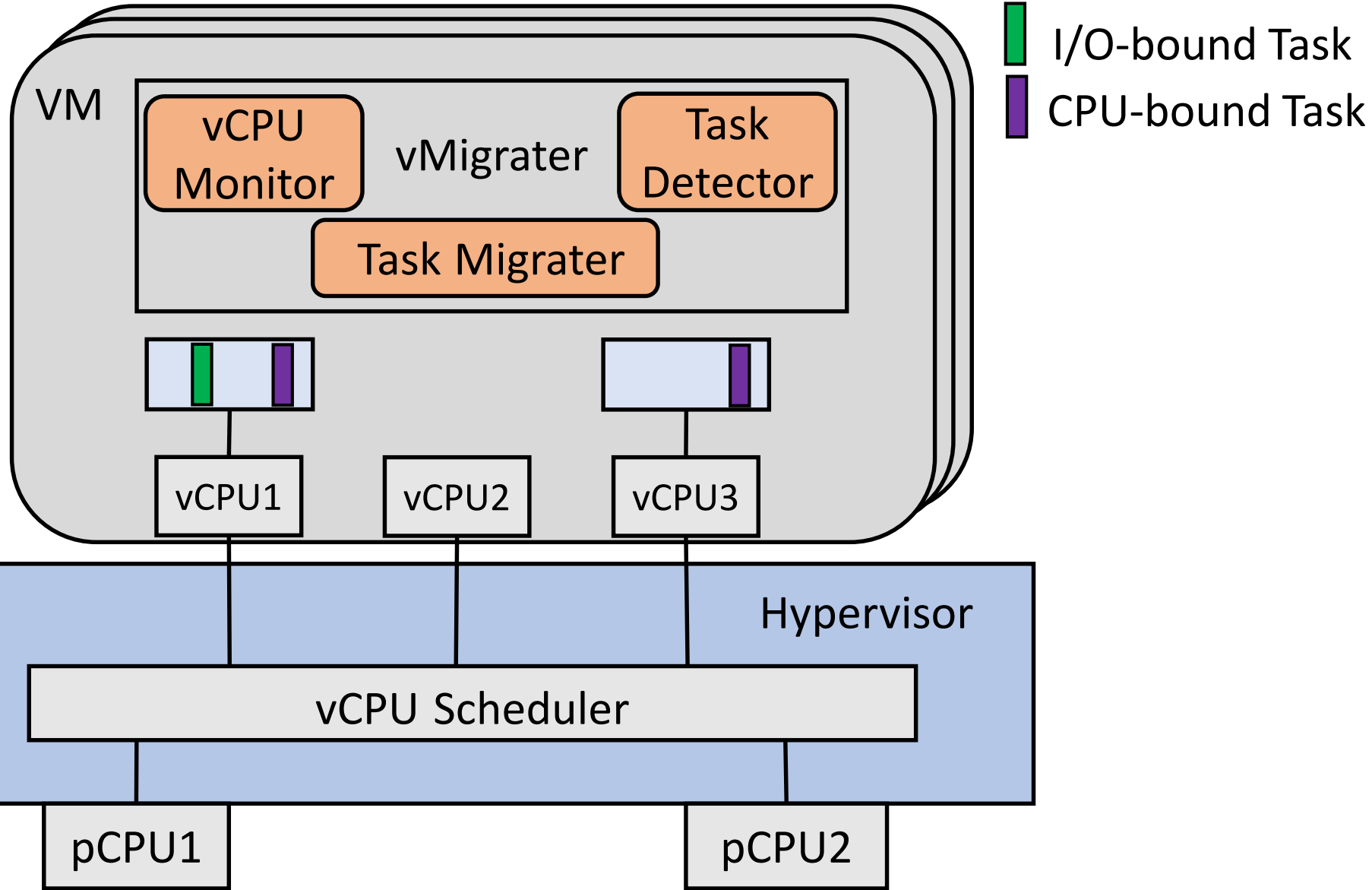


# vMigrater: a user-level design

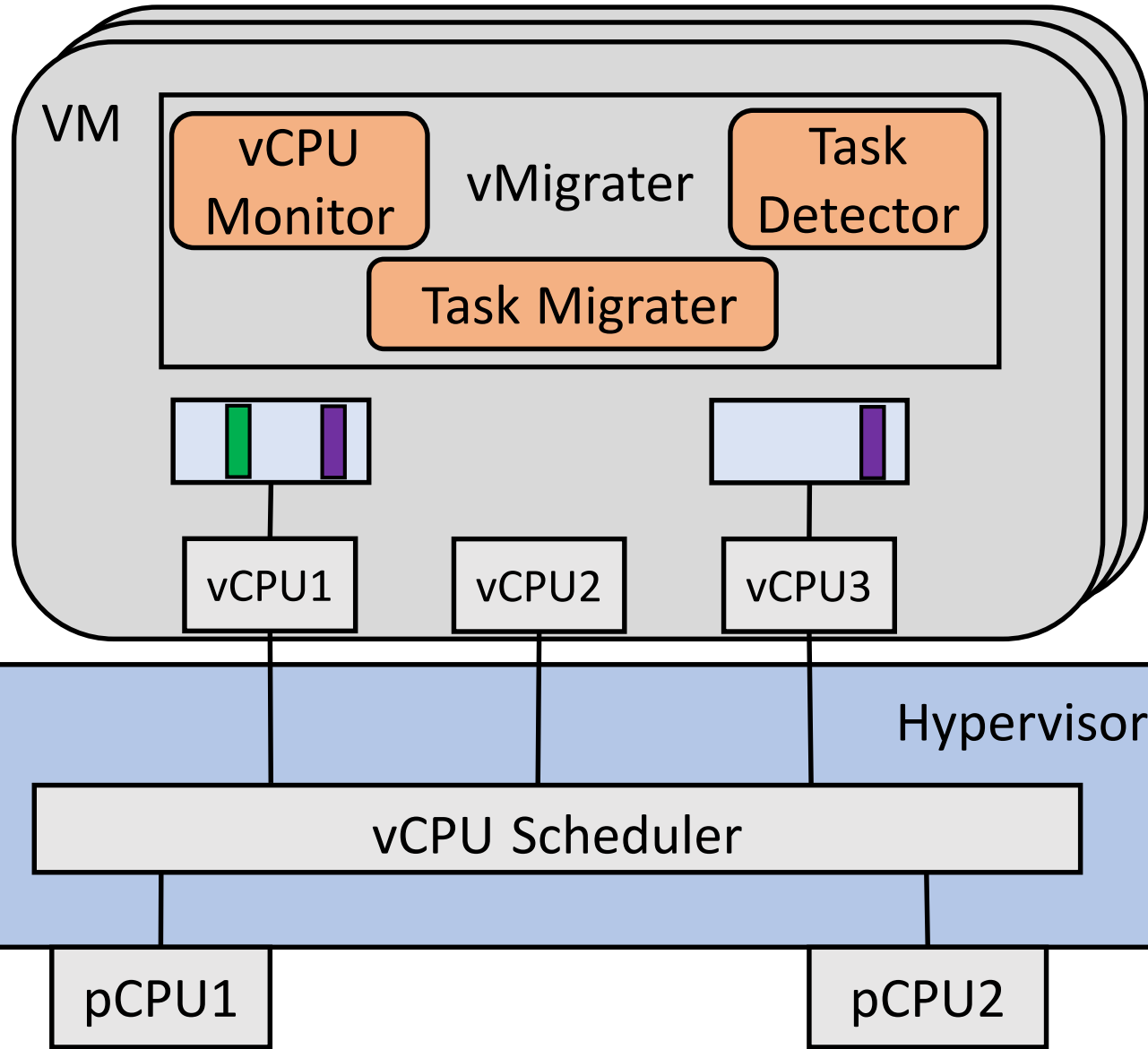






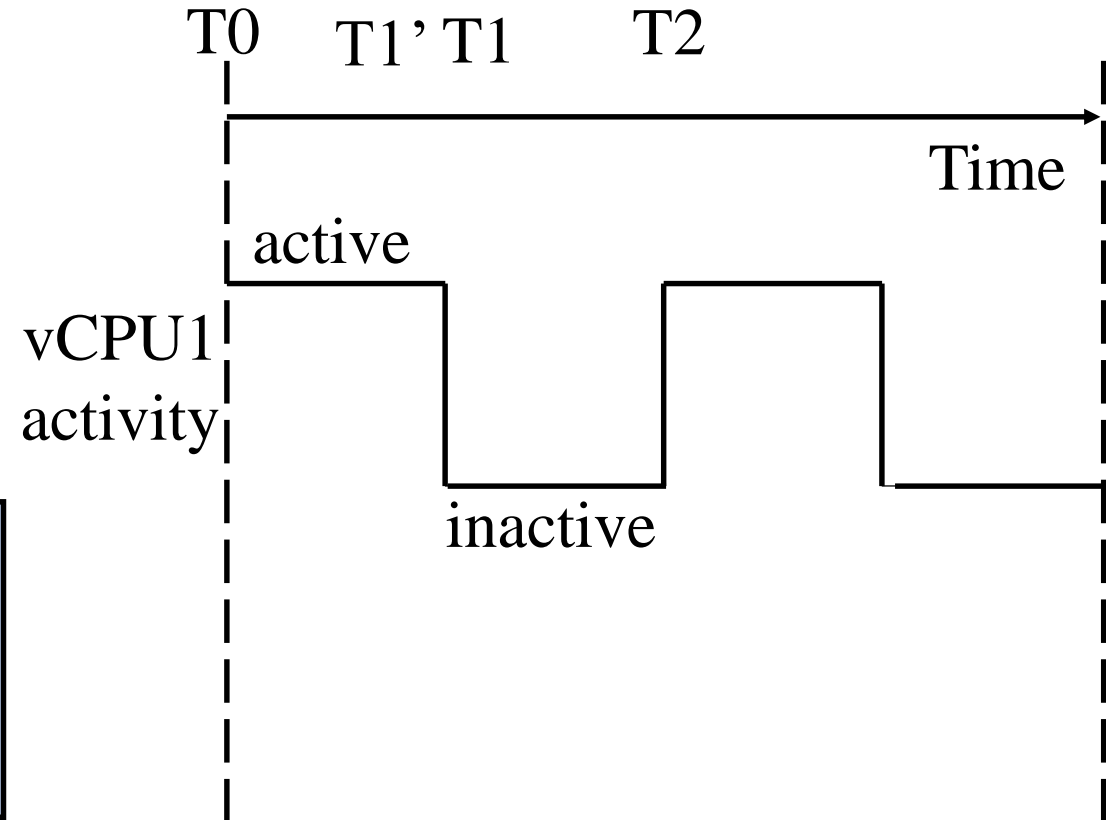
# vMigrater: a user-level design



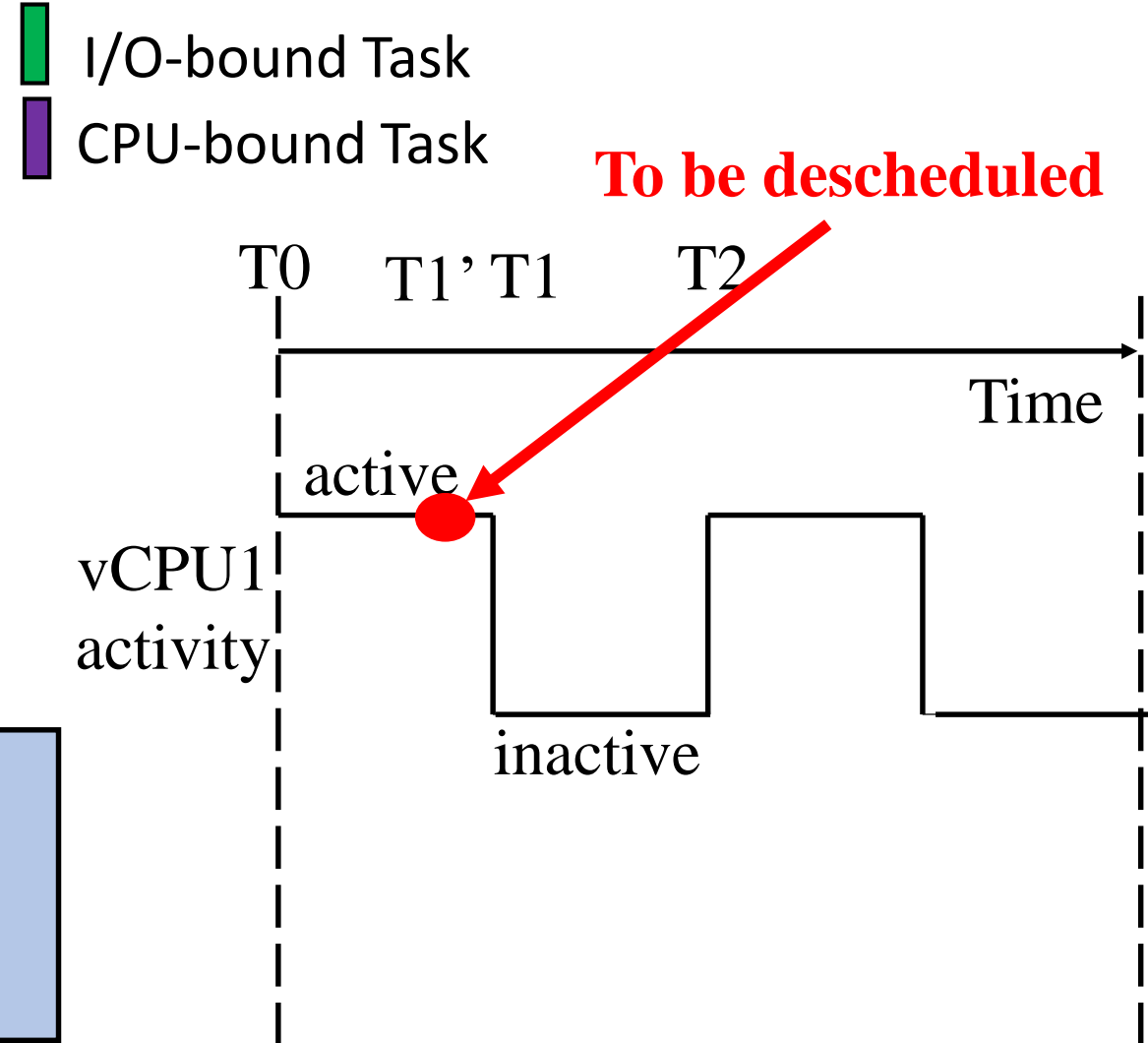
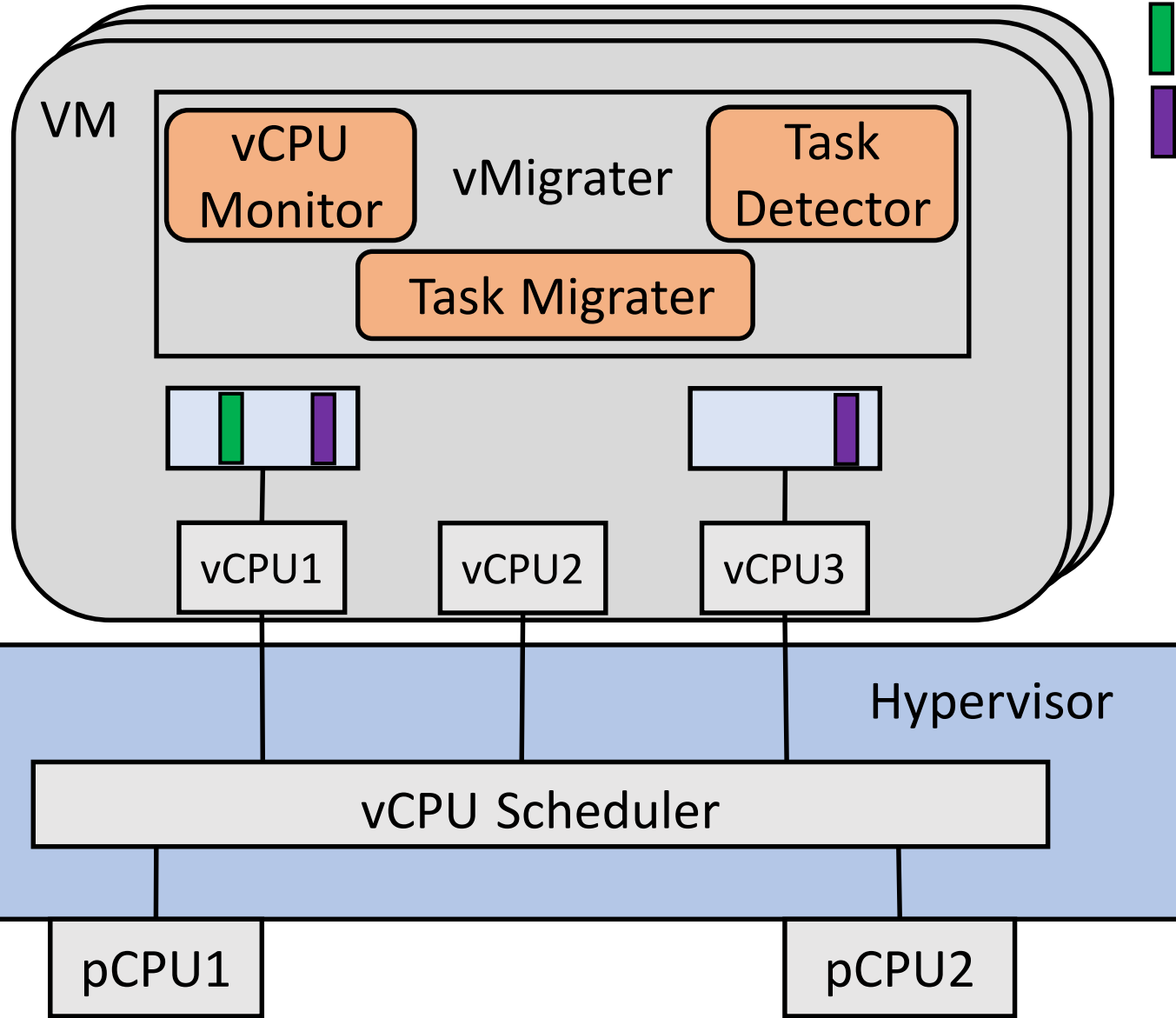
# vMigrater: a user-level design



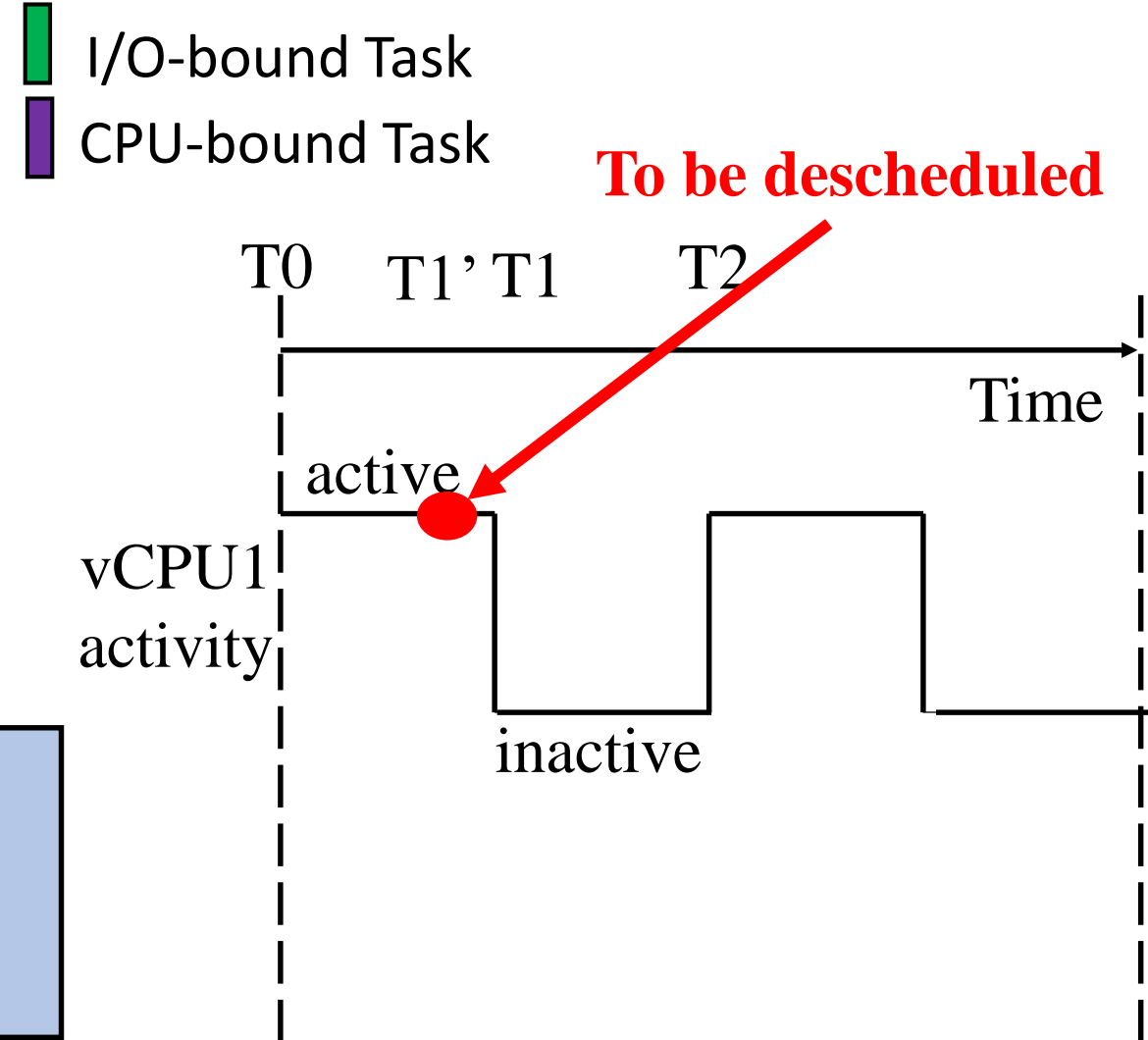
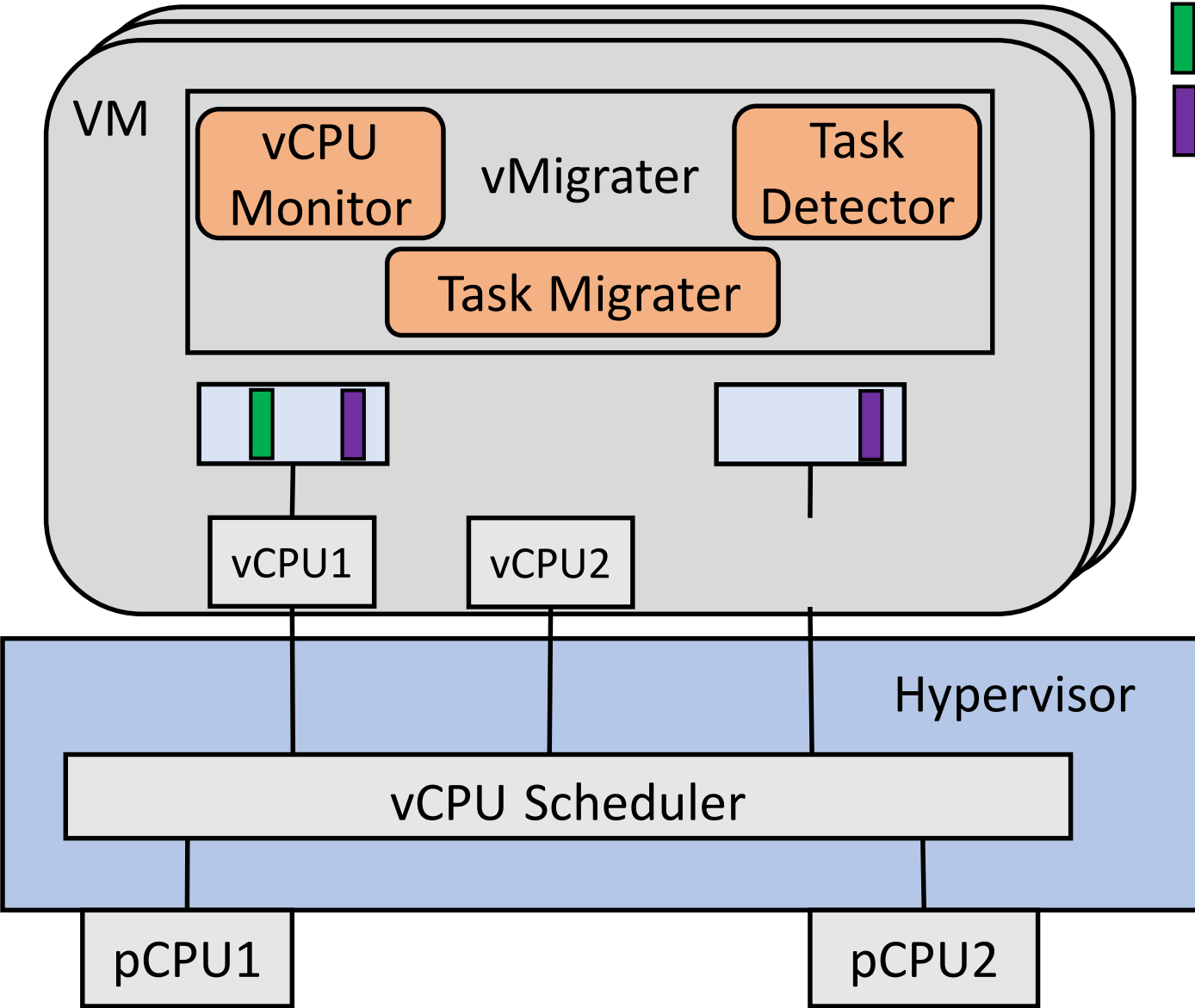
 I/O-bound Task  
 CPU-bound Task



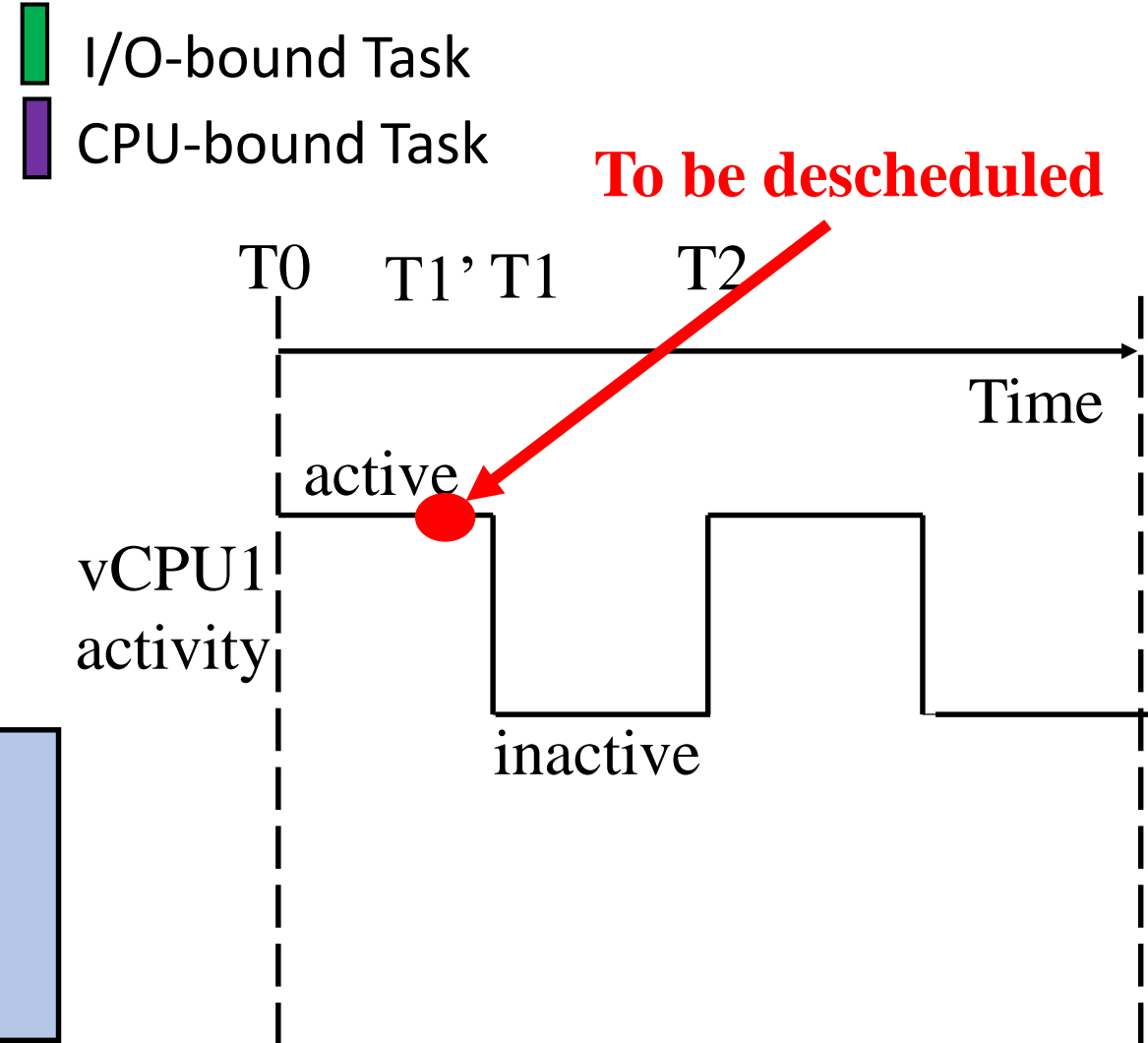
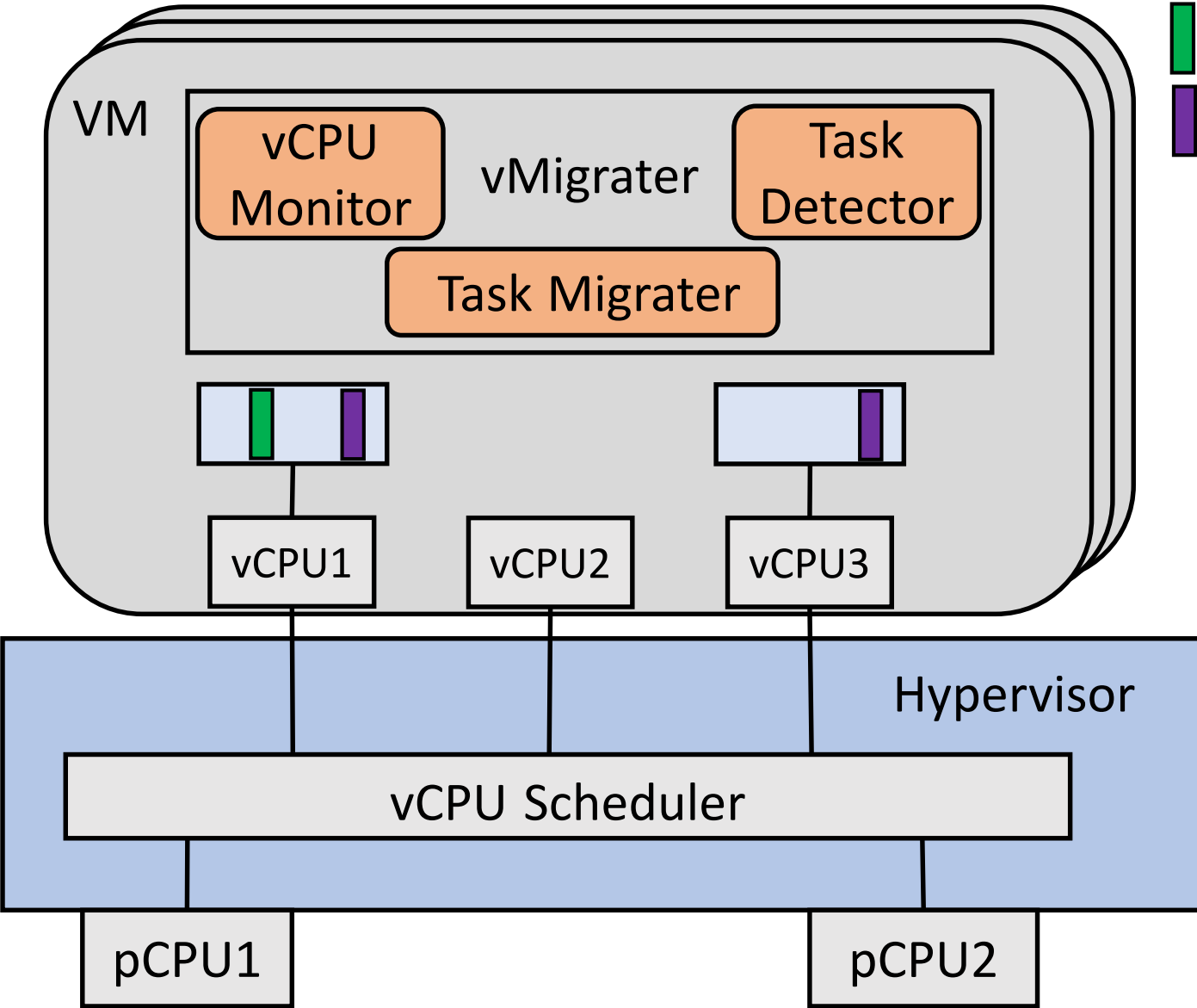
# vMigrater: a user-level design



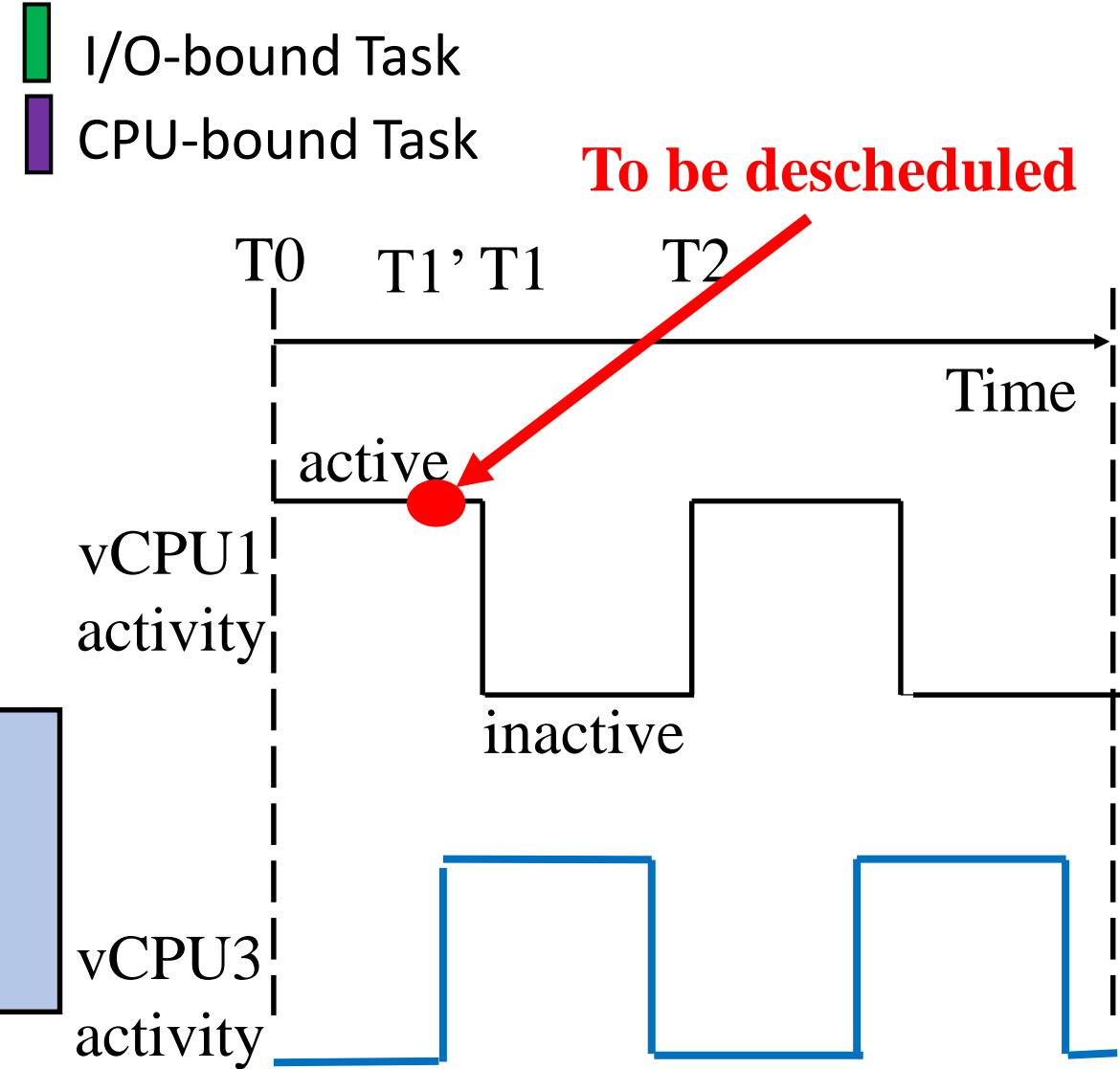
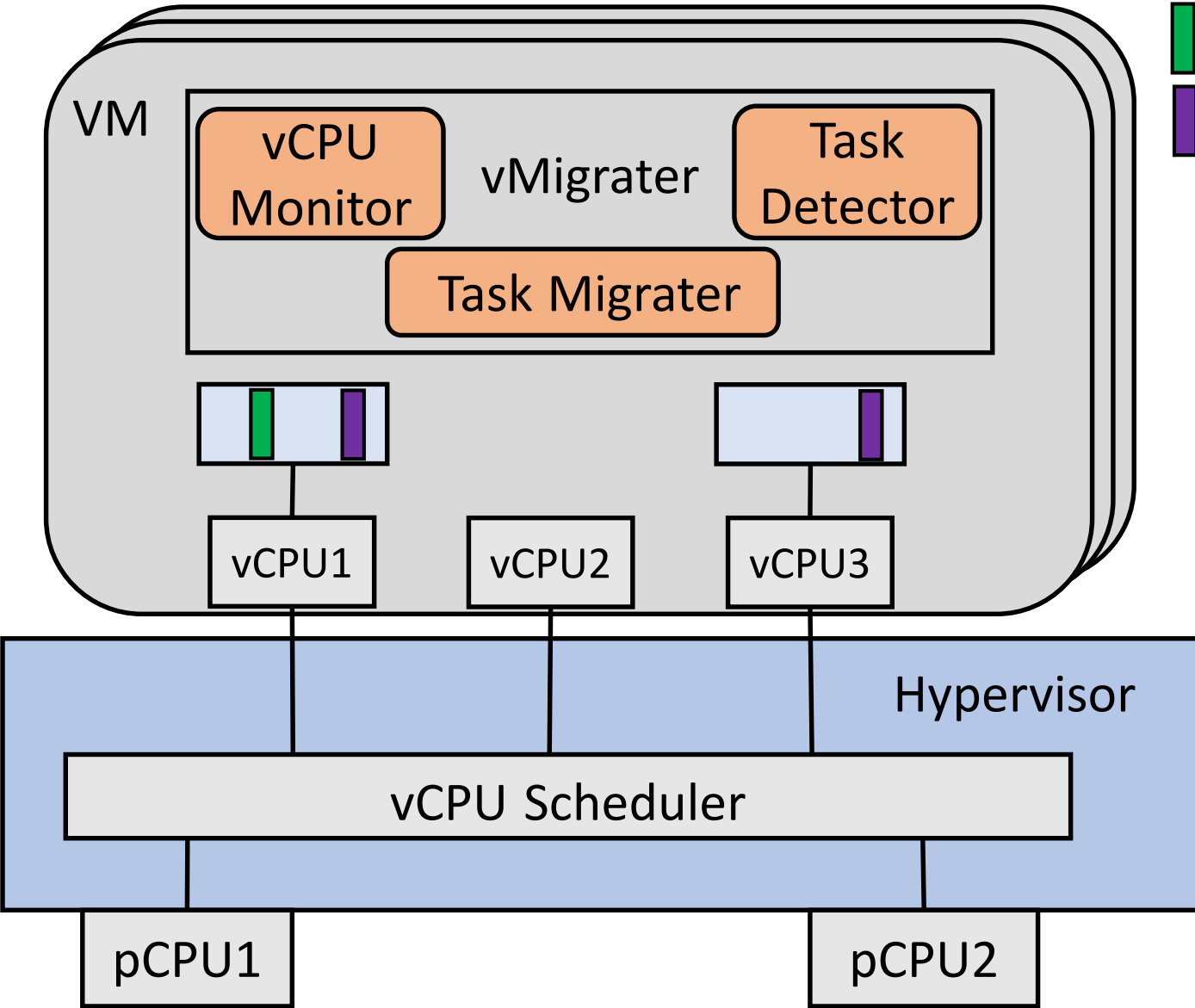
# vMigrater: a user-level design



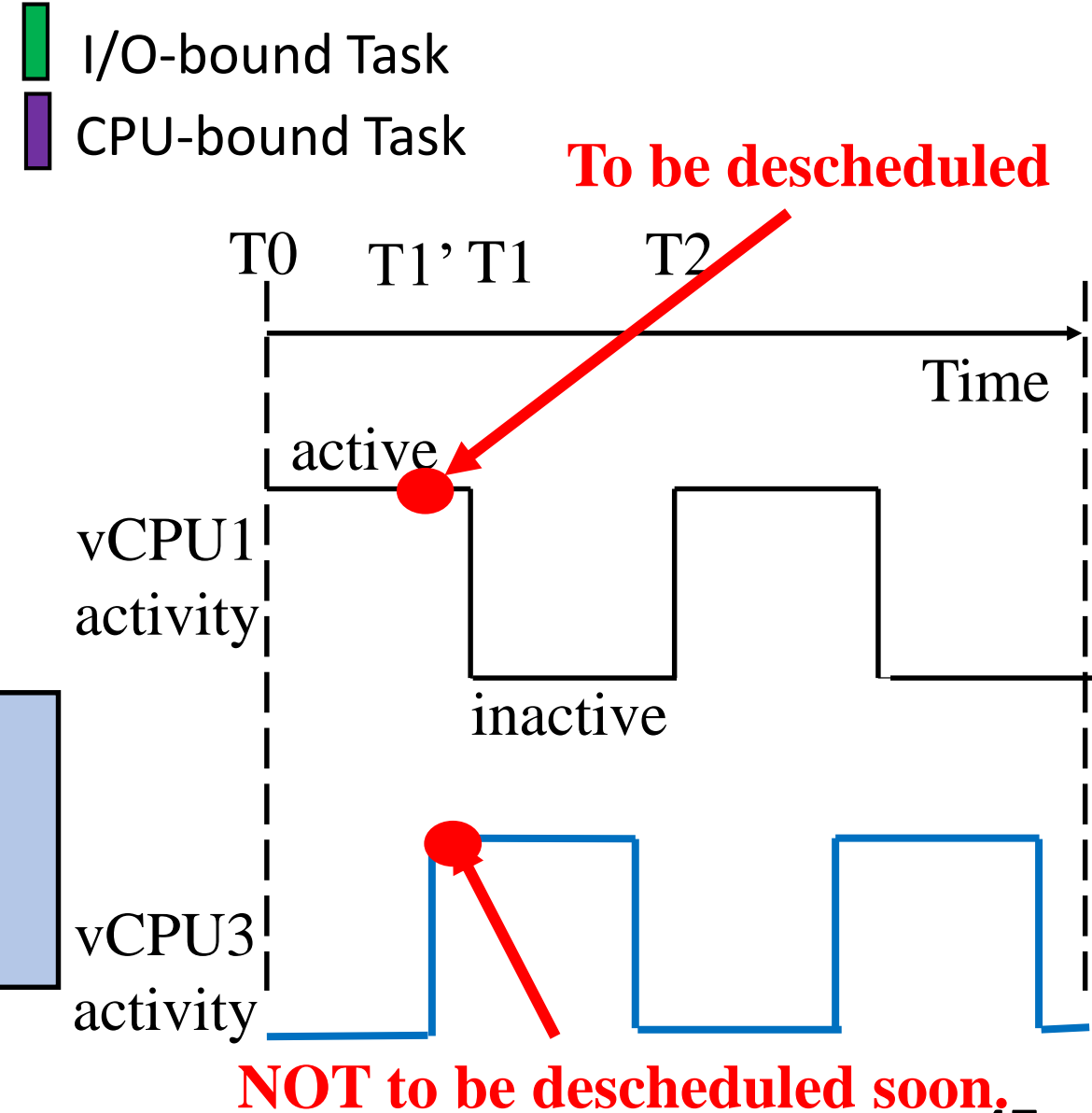
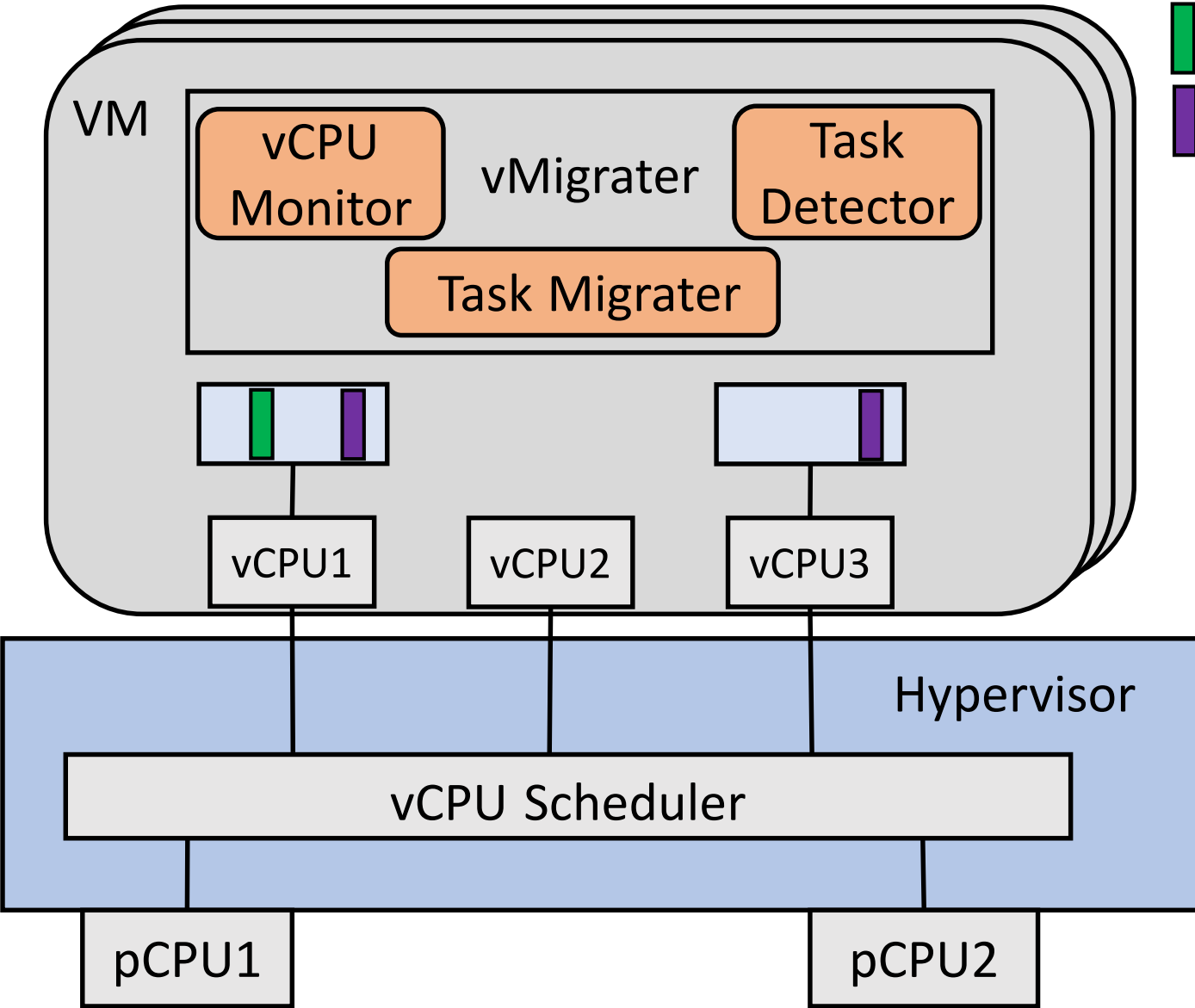
# vMigrater: a user-level design



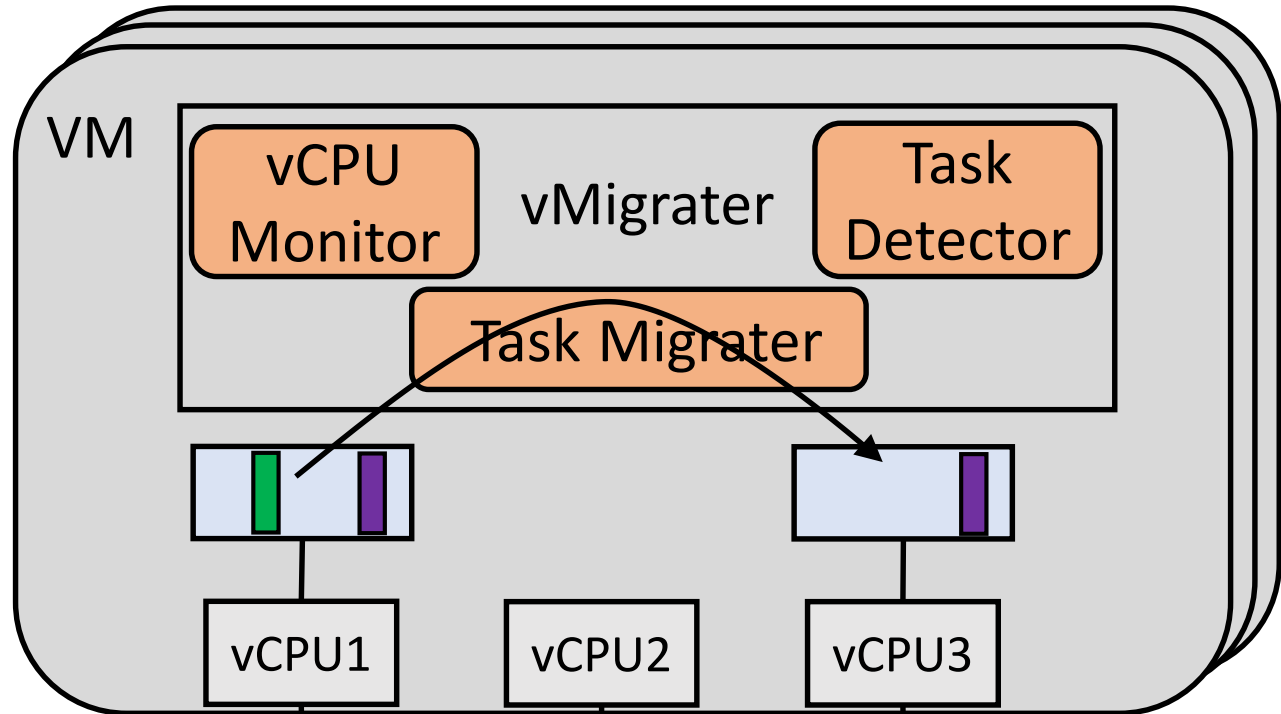
# vMigrater: a user-level design



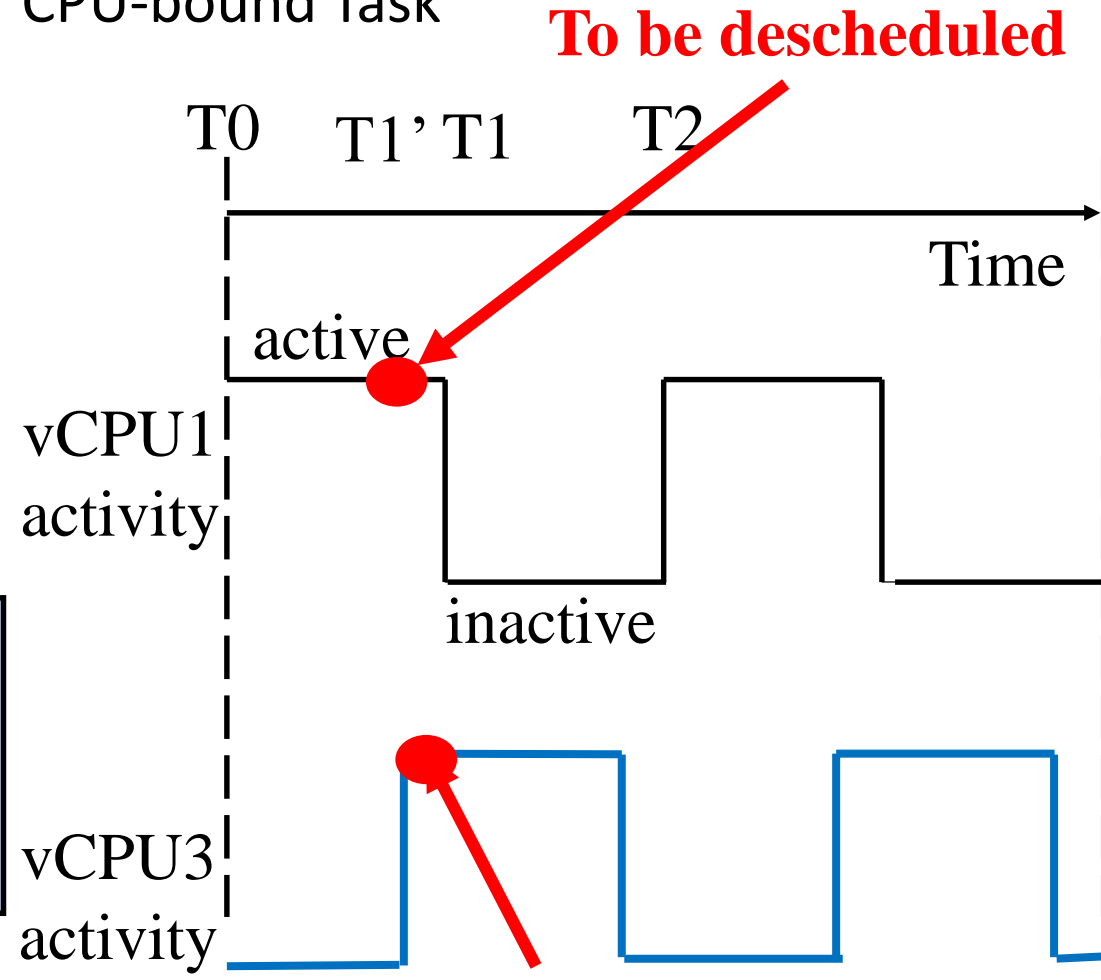
# vMigrater: a user-level design



# vMigrater: a user-level design



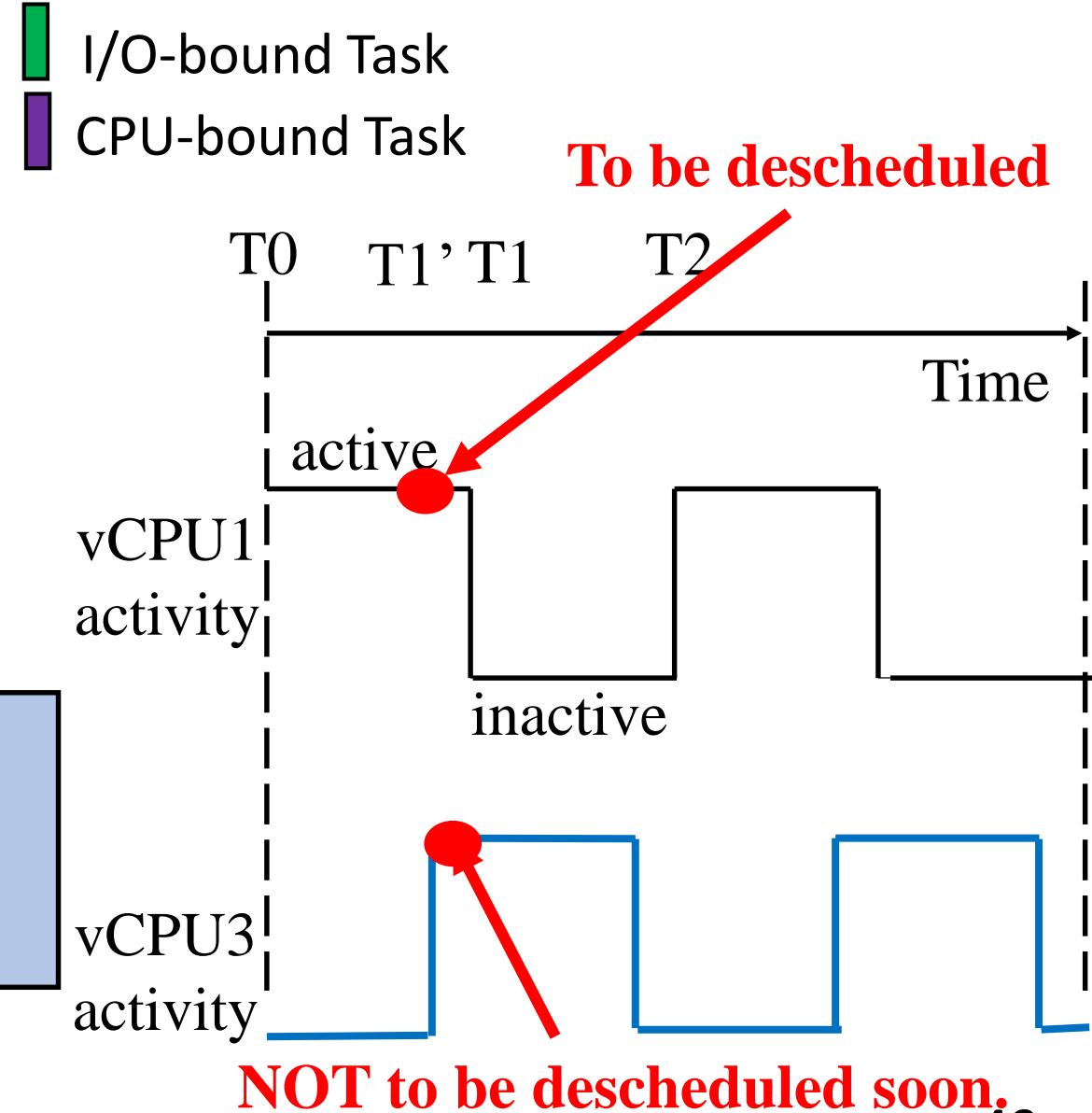
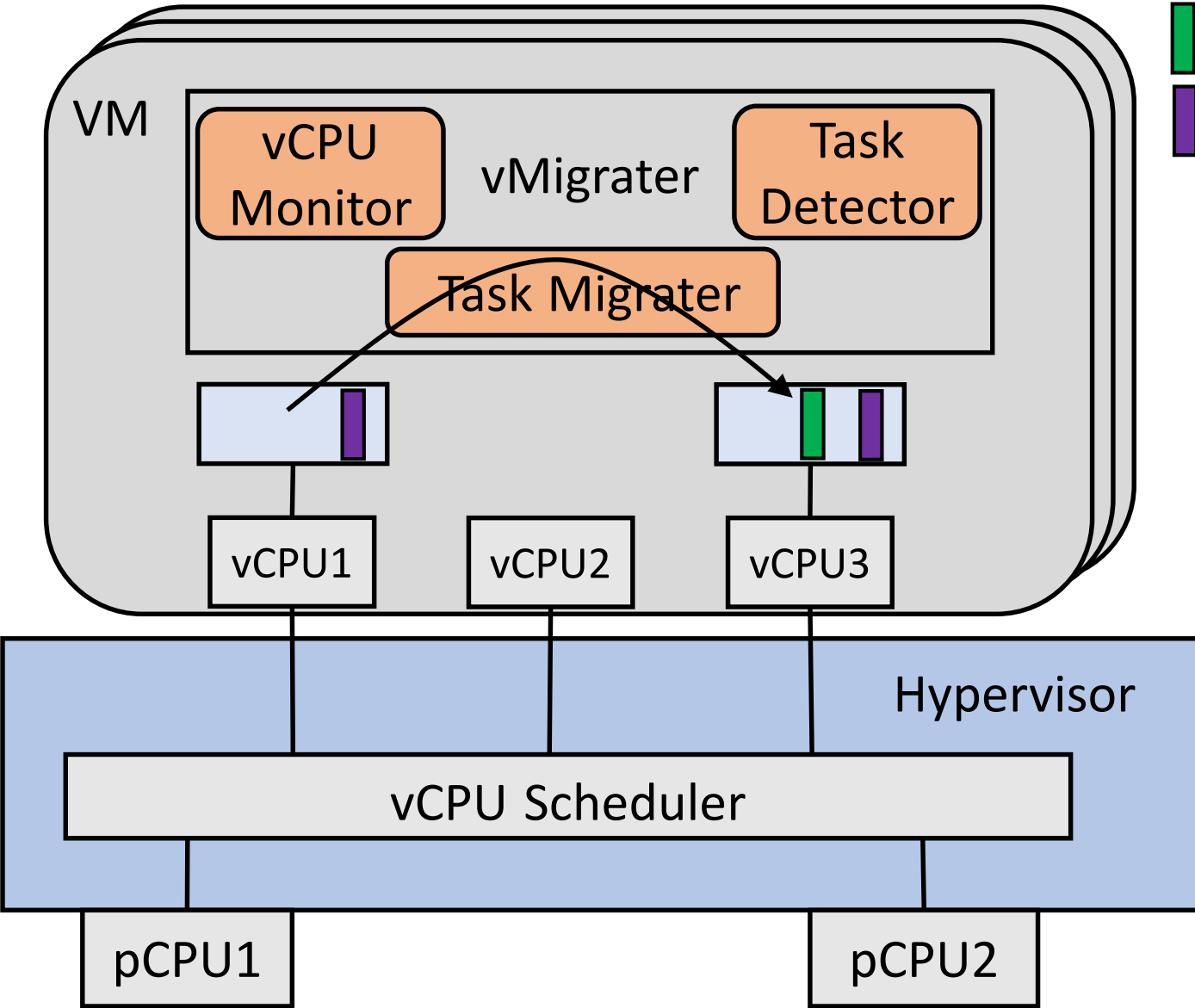
■ I/O-bound Task  
■ CPU-bound Task



**NOT to be descheduled soon.**



# vMigrater: a user-level design



# Challenges with the user-level design (1/3)

- How to detect I/O tasks quickly?

- Existing resource monitors cannot respond to execution phase changes quickly
  - eg., Linux top and iotop refresh measurements periodically every a few seconds
  - Applications often have bursty I/O phases finished within a refreshing period.

# Challenges with the user-level design (1/3)

## - How to detect I/O tasks quickly?

- Existing resource monitors cannot respond to execution phase changes quickly
  - eg., Linux top and iotop refresh measurements periodically every a few seconds
  - Applications often have bursty I/O phases finished within a refreshing period.
- Event-driven method in vMigrator
  - Monitor I/O events time at OS block I/O layer
  - Calculate the fraction between the I/O events time and the whole period
  - respond quickly when task becomes I/O intensive (<1 millisecond)

# Implementation challenges (2/3)

## - when to migrate?

- When the vCPUs are to be inactive
- Naïve approach: monitor inactive/active vCPUs in hypervisor layer: not secure and portable
- Our approach
  - a heartbeat-like mechanism: timer events as heartbeats
  - a vCPU cannot process timer events when it is inactive
  - vCPU time slice: timer differences between the start timer and end timer when the vCPU is active

# Challenges with the user-level design (3/3)

- migrate to which vCPU(s)?

- Migrate to vCPUs with enough remaining time slice
  - Estimation of time slice still relies on the heartbeat-like mechanism
- Naïve approach: consolidate all I/O tasks to the vCPU with the longest remaining time slice
  - Problem: the vCPU may be overloaded
- Our approach: distribute I/O tasks to vCPUs based on I/O workload and remaining time slice.
  - tasks with heavier I/O workload on vCPUs with more time slice

# Experimental Setup

- Dell PowerEdge R430 with 12 cores, a 1TB HDD, and a 1TB SSD
- Both VMs and VMM (linux QEMU/KVM) use Ubuntu Linux 16.04
- Each VM has 12 vCPUs and 4GB DRAM
- Compared with vSlicer [HPDC'12] and xBalloon [SoCC'17]

# Evaluation applications and workloads

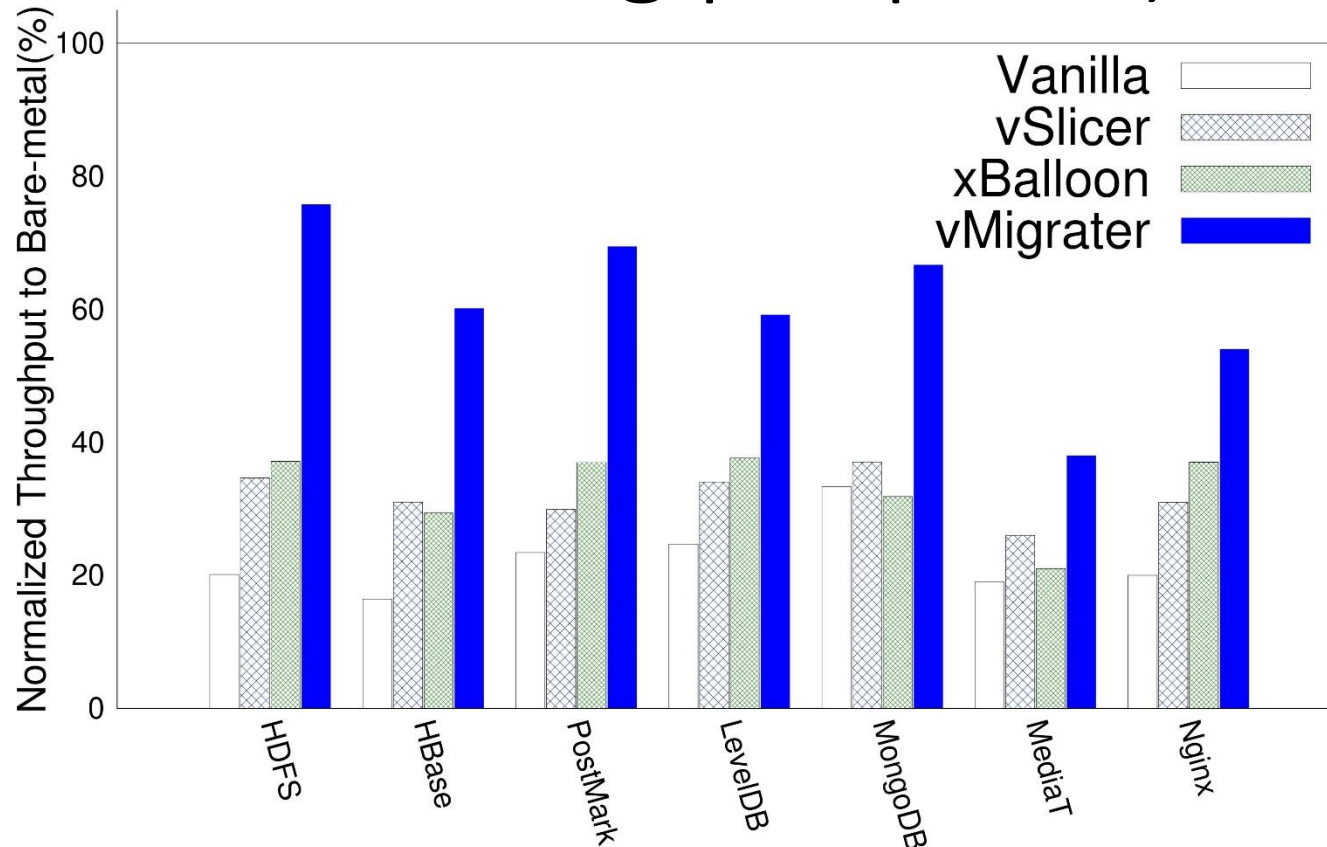
Application	Workload
HDFS	Sequentially read 16GB with HDFS TestDFSIO
LevelDB	Randomly scan table with db_bench
MediaTomb	Concurrent requests on transcoding a 1.1GB video
HBase	Randomly read 1GB with Hbase PerfEval
PostMark	Concurrent requests on a mail server
Nginx	Concurrent requests on watermarking images
MongDB	Sequentially scan records with YCSB

# Evaluation questions

- How much performance improvement can be achieved with vMigrater, compared with vanilla KVM and two related systems?
- Can vMigrater help the I/O scheduler in the VMM to achieve fairness between VMs?
- How robust is vMigrater to varying workloads?
- What is the overhead incurred by vMigrater?
- What is vMigrater's performance when the workload in a VM varies over time
- How does vMigrater scale to the number of shared vCPUs on a pCPU?

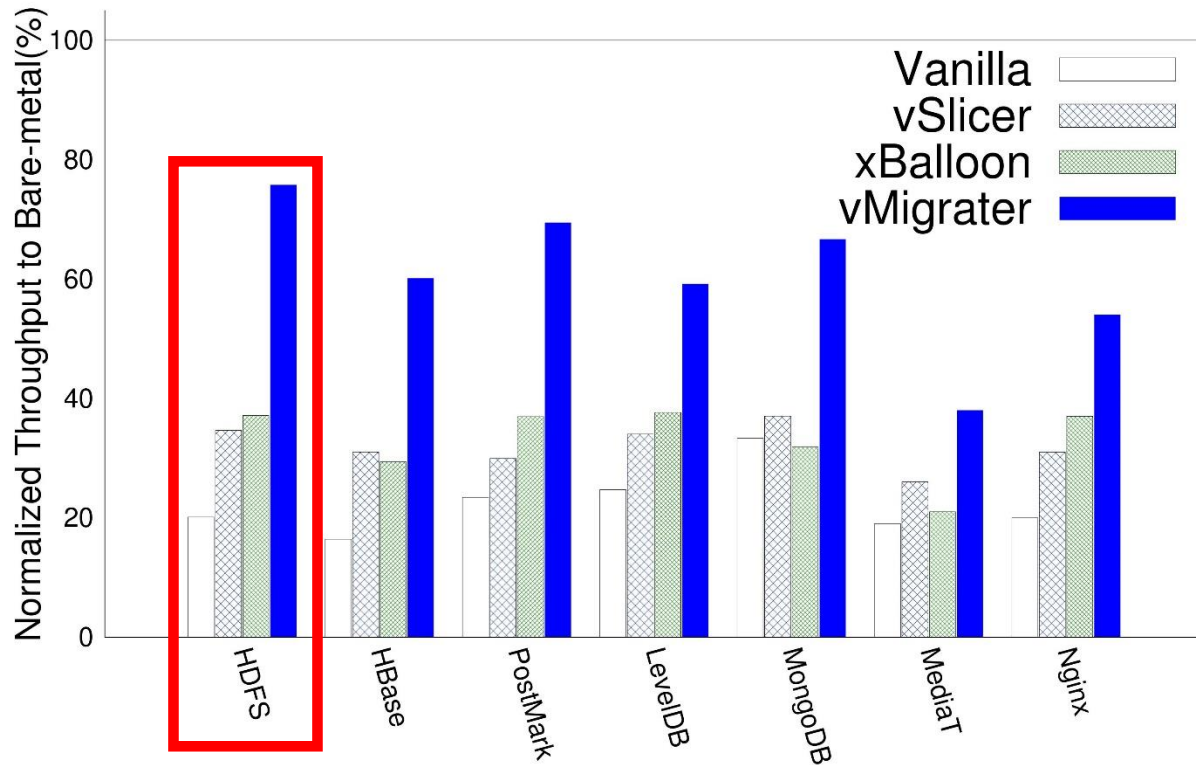


# Throughput on seven applications (4 vCPUs sharing per pCPU)



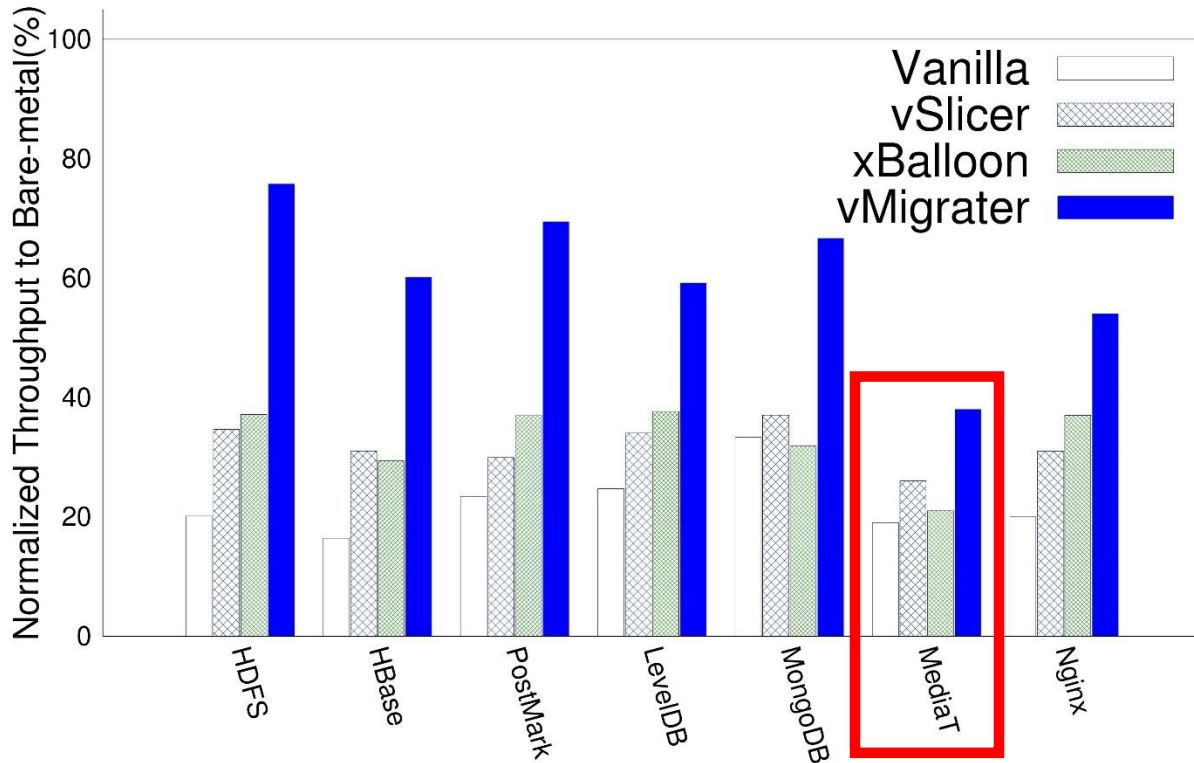
vMigrater's throughput is 192% higher than Vanilla KVM, 75% higher than vSlicer, 84% higher than xBalloon on average

# HDFS performance analysis



HDFS	Vanilla	vSlicer	xBalloon	vMigrater
I/O inactivity time (seconds)	121.82	92.91	75.27	6.62

# MediaTomb performance analysis



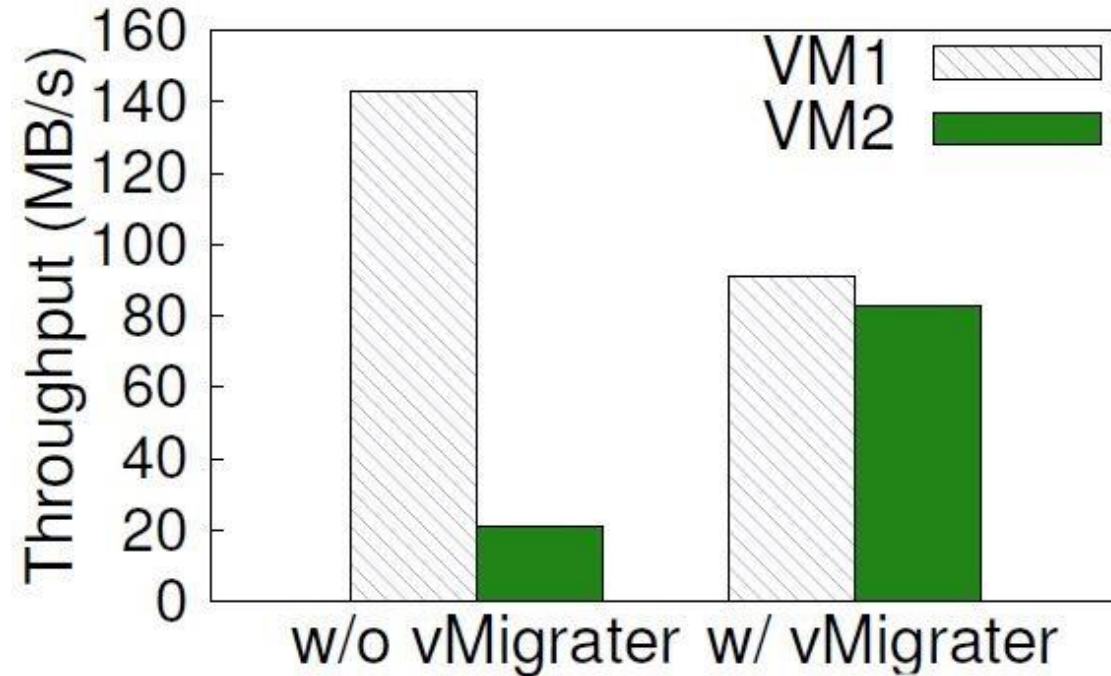
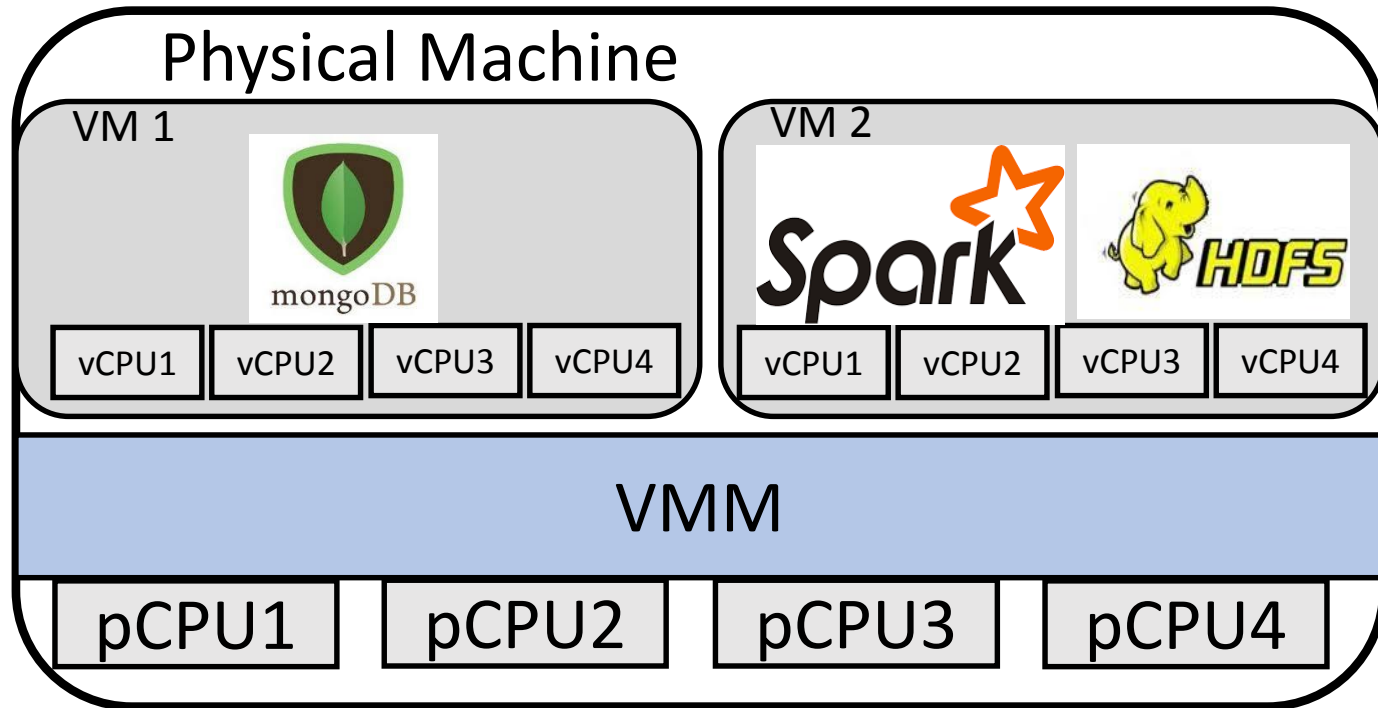
HDFS	Vanilla	vSlicer	xBalloon	vMigrater
I/O inactivity time (seconds)	121.82	92.91	75.27	6.62

MediaTomb	Vanilla	vSlicer	xBalloon	vMigrater
I/O inactivity time (seconds)	108.61	89.46	116.96	34.95

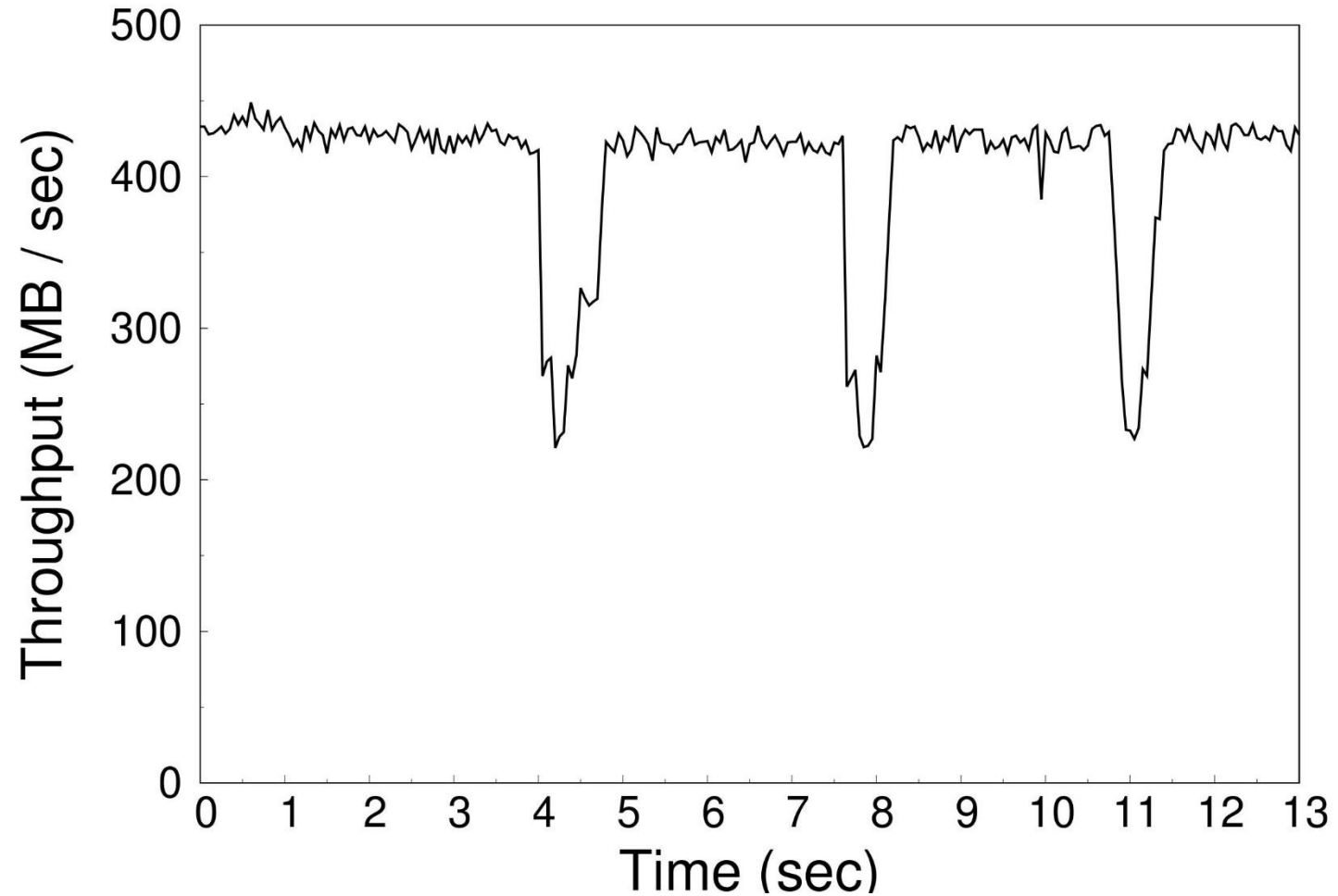
vMigrater has big I/O inactivity periods on MediaTomb

- MediaTomb combine computation and I/O in each thread so the migration cost is higher

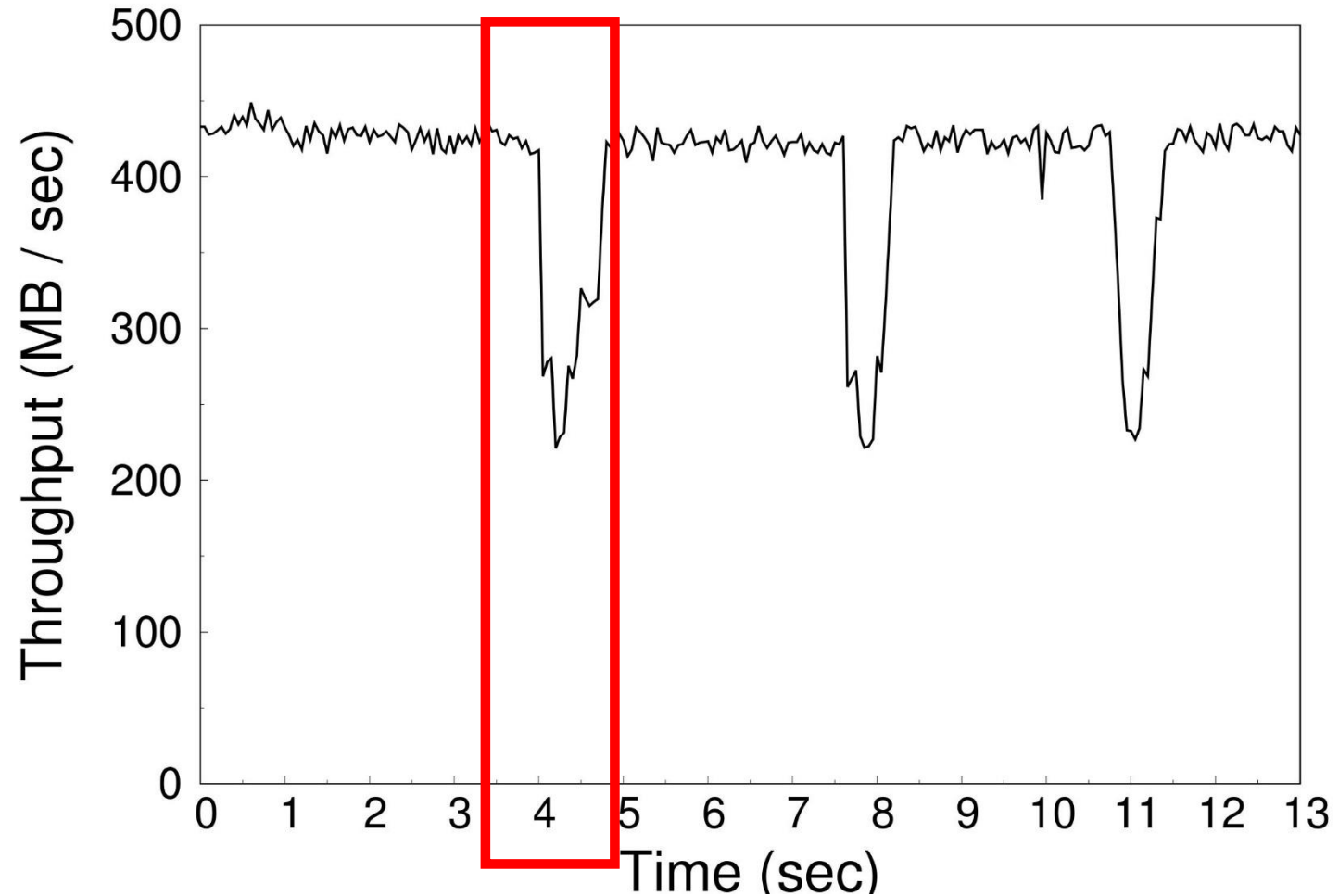
# Fairness of I/O Scheduler in VMM



# Robustness to varying workloads

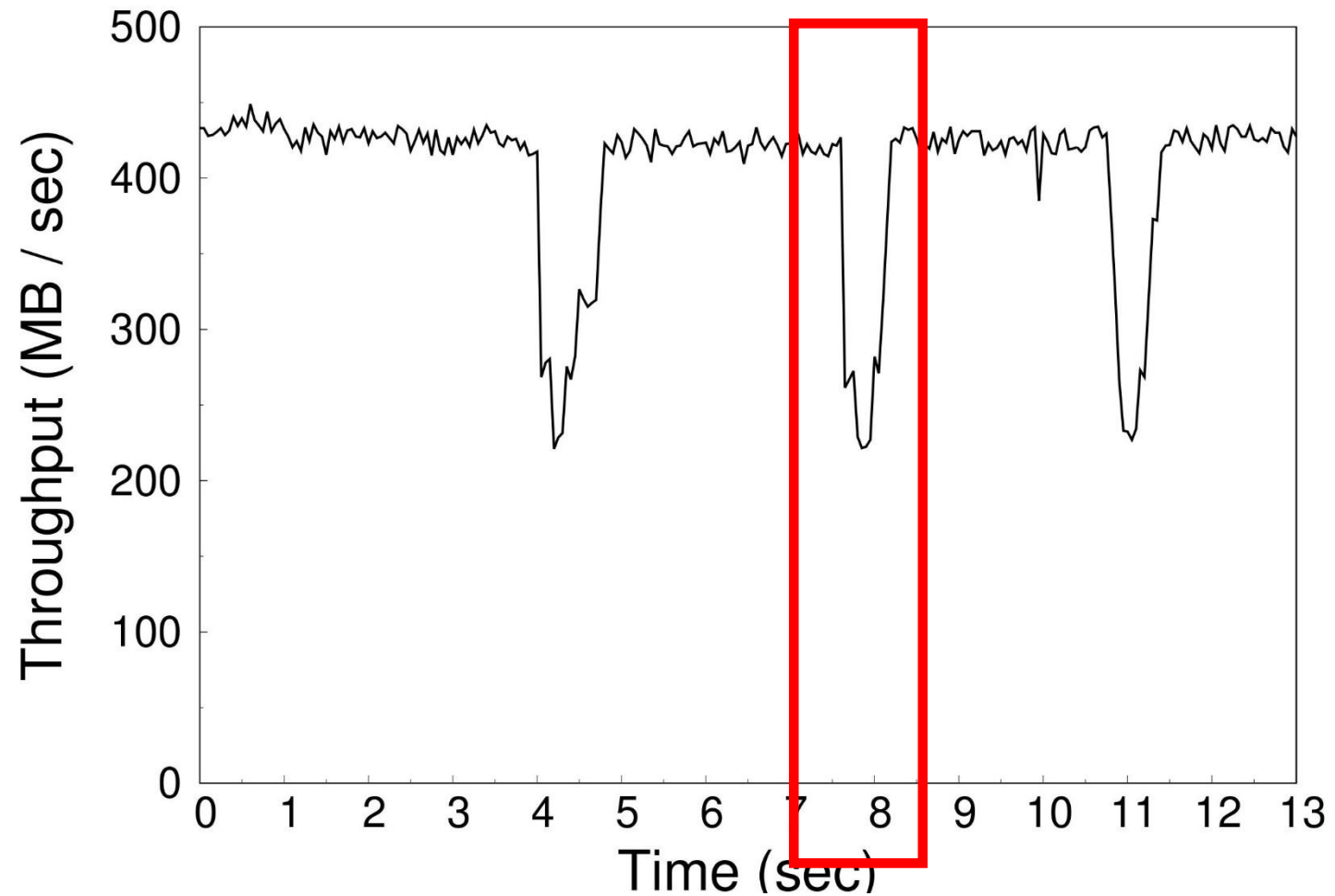


# Robustness to varying workloads



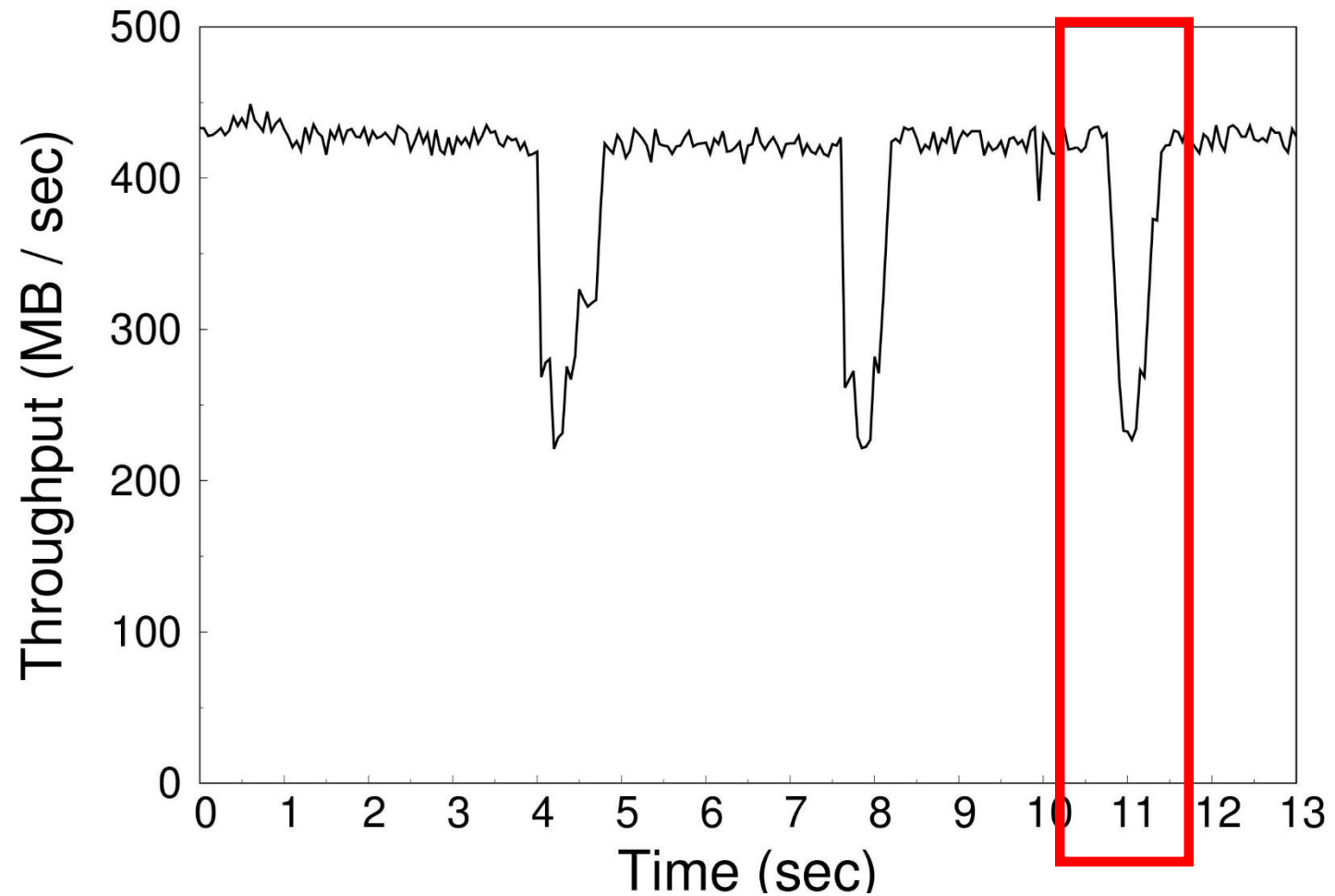
Added 4 clients

# Robustness to varying workloads



Added 8 clients

# Robustness to varying workloads



Added 16 clients



# Conclusions

- I/O inactivity problem
  - Performance degradation
  - I/O scheduler unfairness
- vMigrater: effectively mitigating I/O inactivity problem
  - Performance is close to bare-metal
  - Regain fairness
- vMigrater's source code: <https://github.com/hku-systems/vMigrater>



Thank you  
Questions?