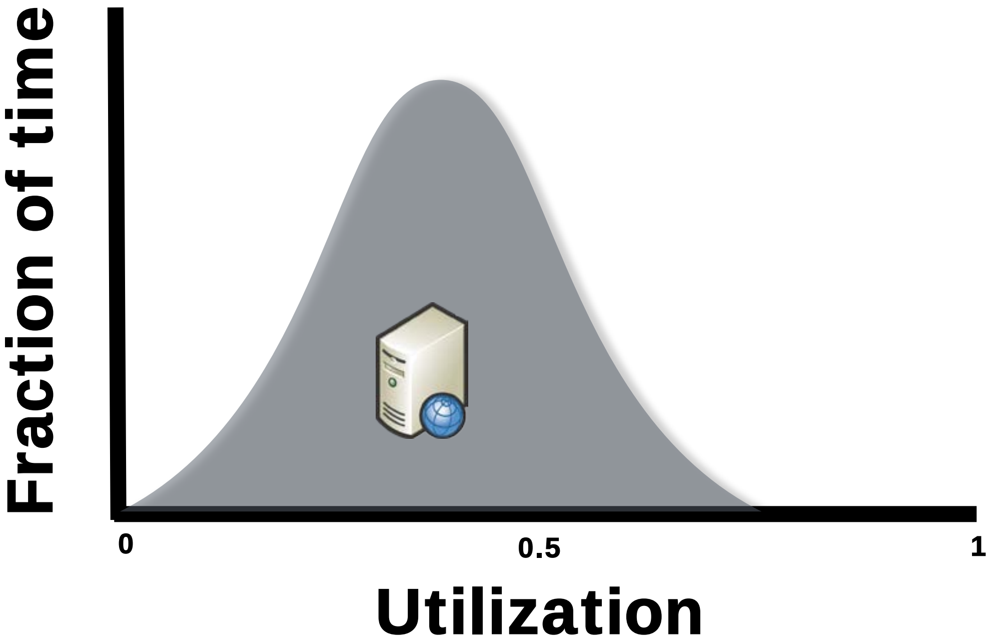
Accelerating Critical OS Services in Virtualized Systems with Flexible Micro-sliced Cores

Jeongseob Ahn*, Chang Hyun Park‡, Taekyung Heo‡, Jaehyuk Huh‡

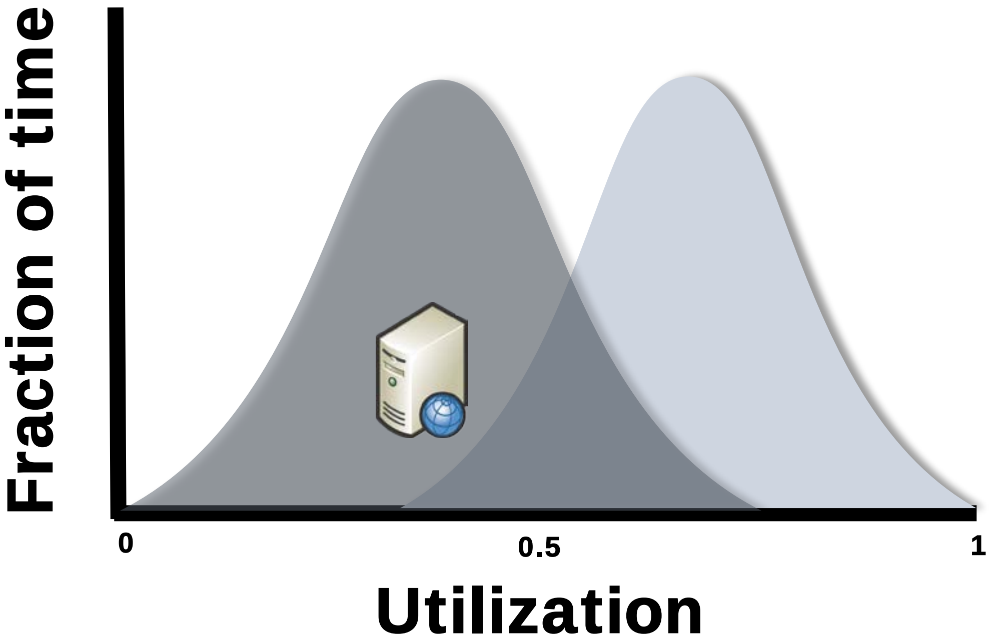
* AJOU UNIVERSITY

‡ KAIST

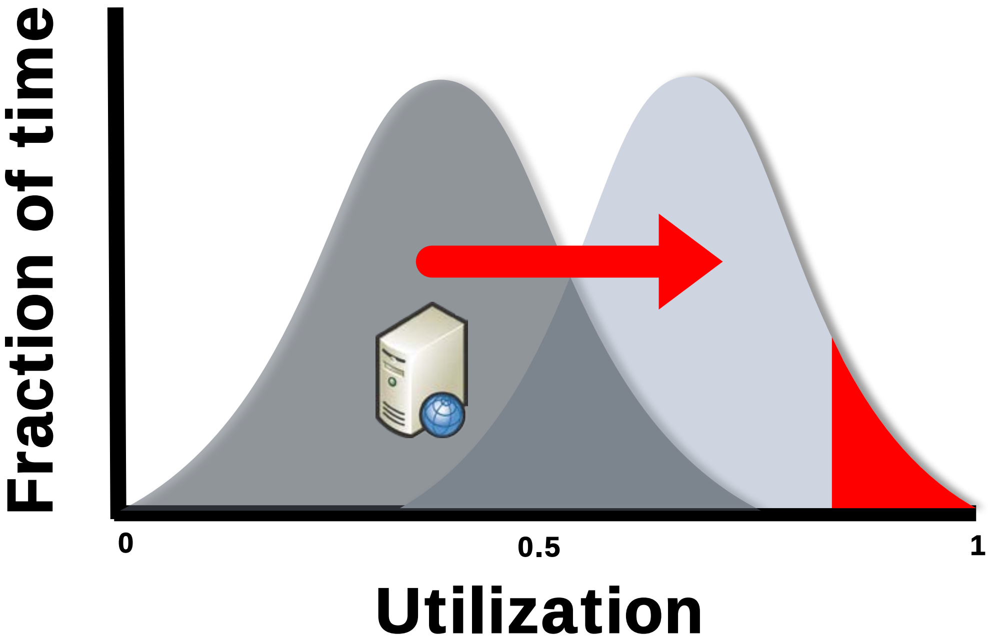
Challenge of Server Consolidation



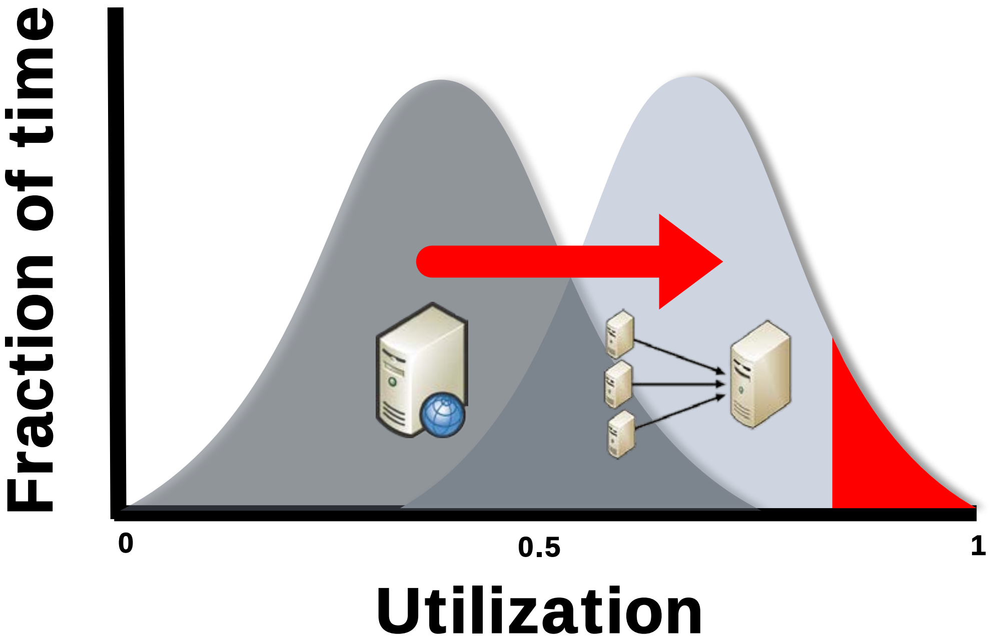
Challenge of Server Consolidation



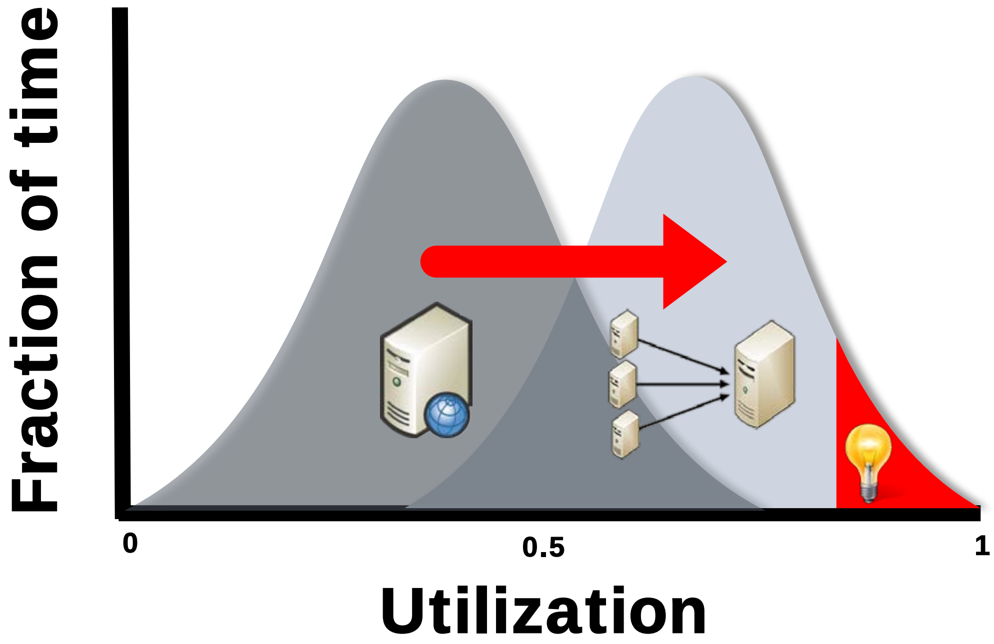
Challenge of Server Consolidation



Challenge of Server Consolidation

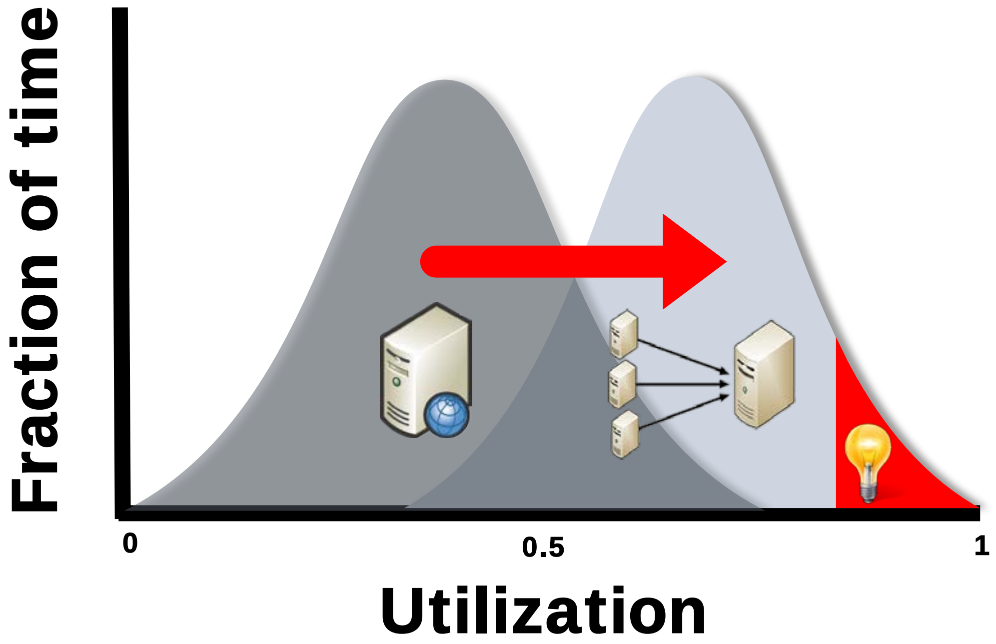


Challenge of Server Consolidation



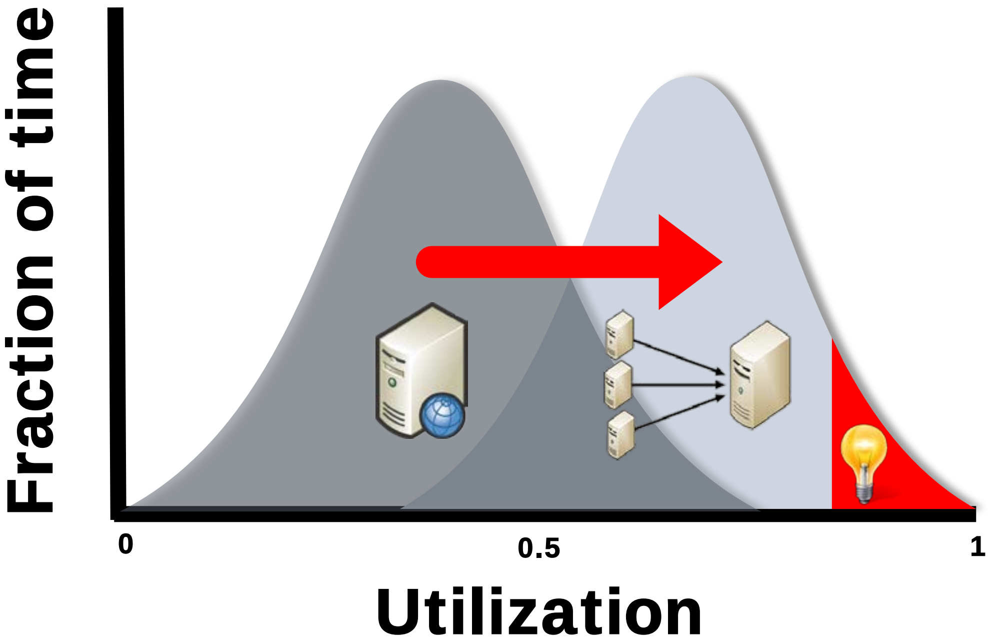
Consolidation improves system utilization

Challenge of Server Consolidation



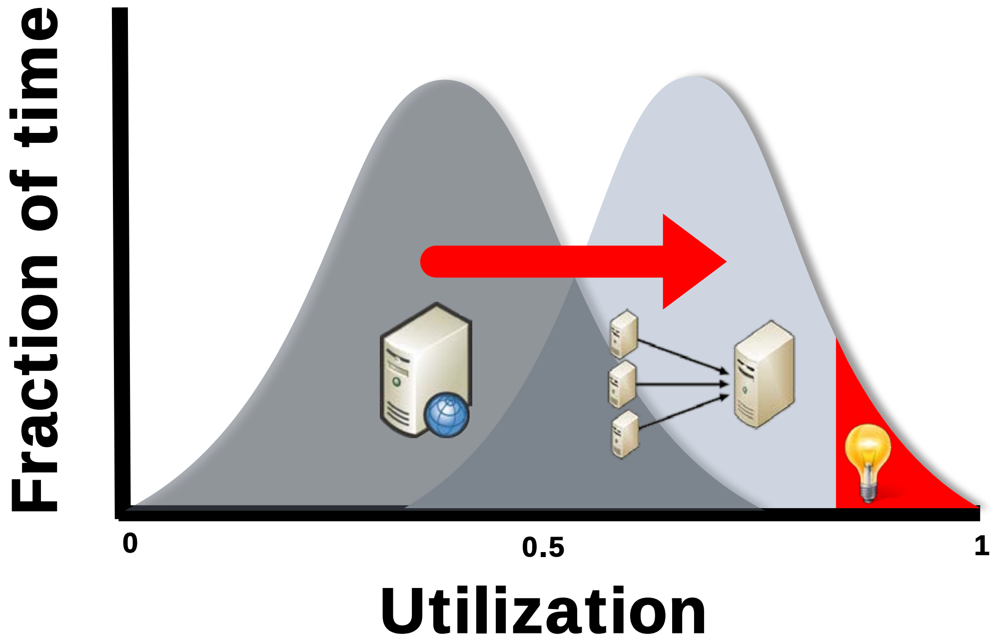
Consolidation improves system utilization

Challenge of Server Consolidation



 Consolidation improves system utilization

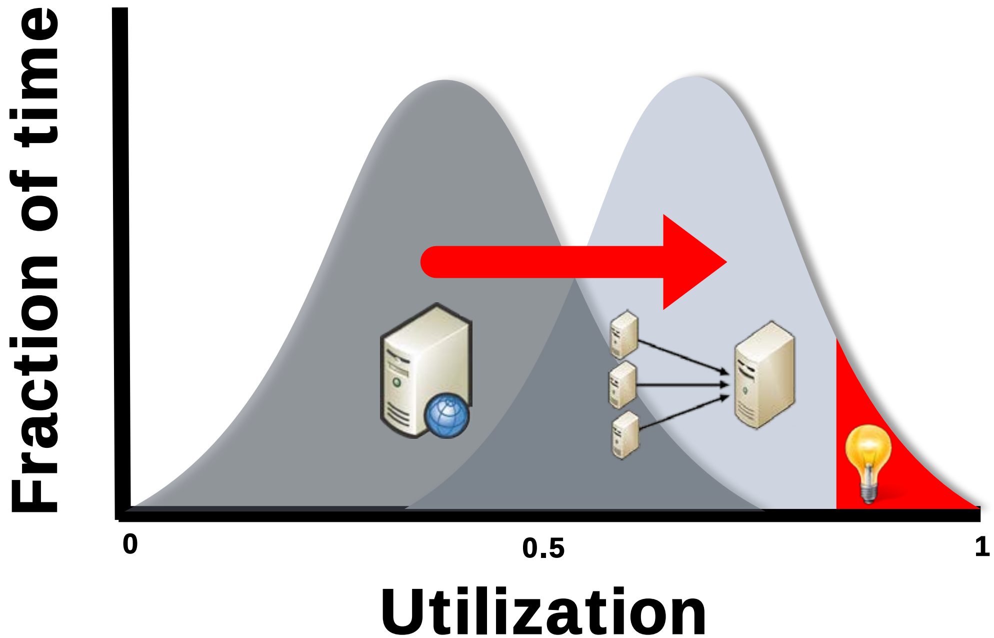
Challenge of Server Consolidation



 Consolidation improves system utilization

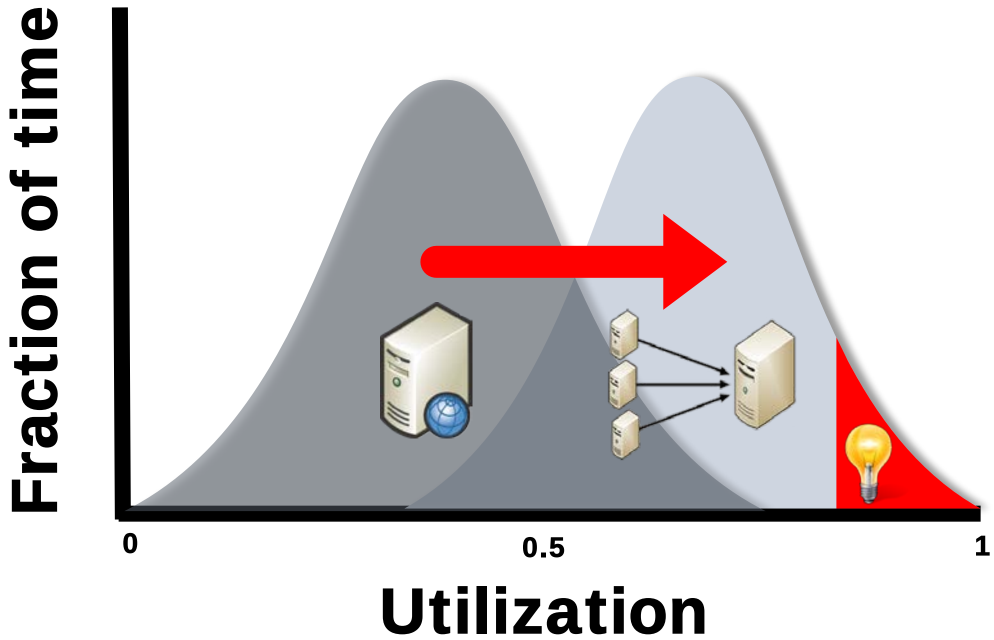
However, resources are contended

Challenge of Server Consolidation



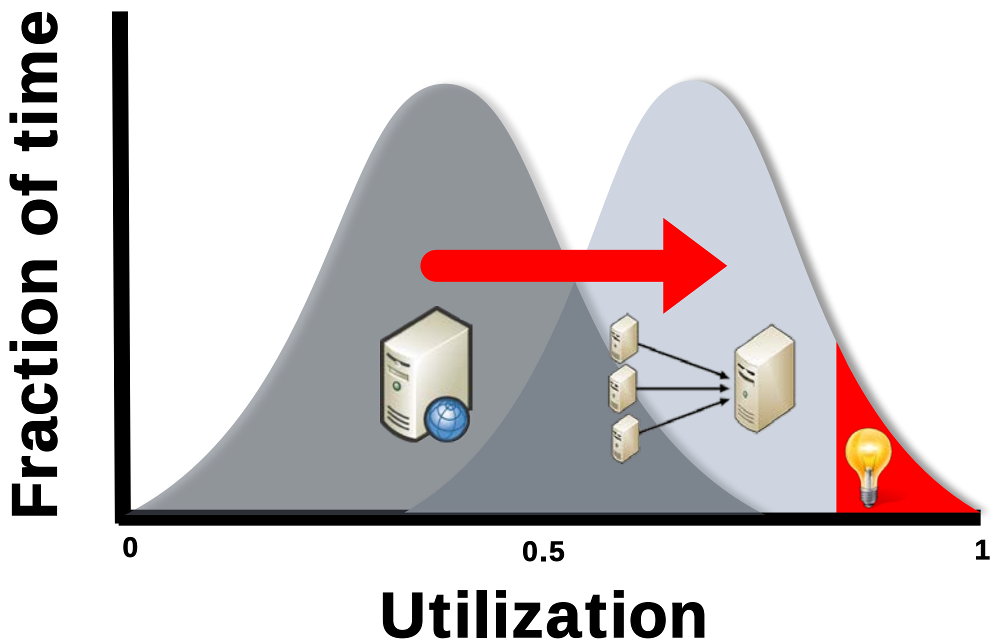
- Consolidation improves system utilization
- However, resources are contended

Challenge of Server Consolidation



- Consolidation improves system utilization
- However, resources are contended

Challenge of Server Consolidation



Consolidation improves system utilization

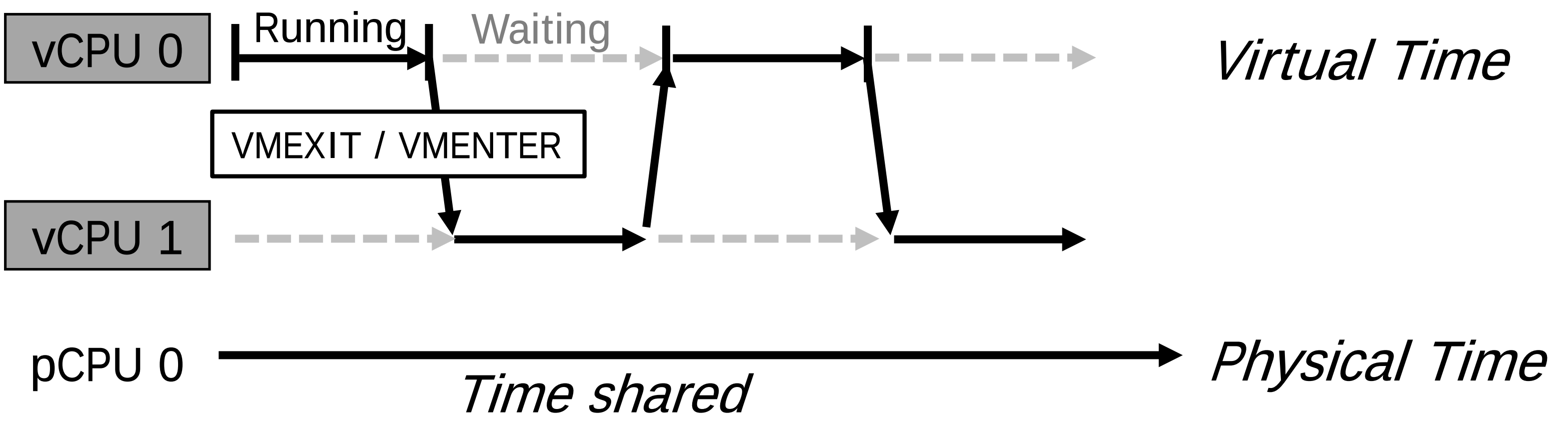
However, resources are contended

Challenge of Server Consolidation

- Consolidation improves system utilization
- However, resources are contended

So, What Can Happen?

- Virtual time discontinuity

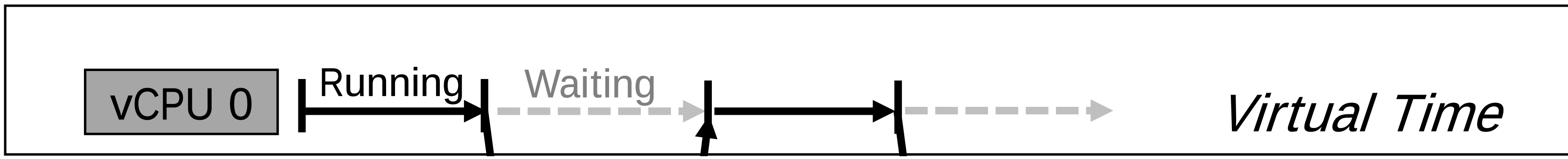


So, What Can Happen?

- Virtual time discontinuity

So, What Can Happen?

- Virtual time discontinuity



❶ Spinlock waiting time(`gmake`)

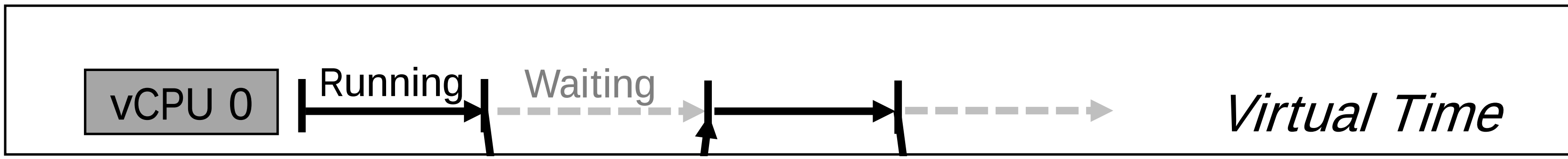
Kernel Component	Avg. waiting time (μsec)	
	solo	co-run*
Page reclaim	1.03	420.13
Page allocator	3.42	1,053.26
Dentry	2.93	1,298.87
Runqueue	1.22	256.07

** Concurrently running with Swaptions of PARSEC*

Processing time is amplified

So, What Can Happen?

- Virtual time discontinuity



❶ Spinlock waiting time(`gmake`)

Kernel Component	Avg. waiting time (μsec)	
	solo	co-run*
Page reclaim	1.03	420.13
Page allocator	3.42	1,053.26
Dentry	2.93	1,298.87
Runqueue	1.22	256.07

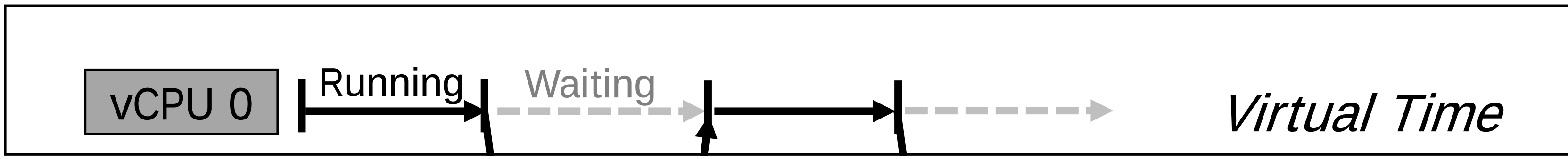
** Concurrently running with Swaptions of PARSEC*

		Avg.	Min.	Max.
dedup	solo	28	5	1927
	co-run*	6,354	7	74915
vips	solo	55	5	2052
	co-run*	14,928	17	121548

Processing time is amplified

So, What Can Happen?

- Virtual time discontinuity



❶ Spinlock waiting time(gmake)

Kernel Component	Avg. waiting time (μsec)	
	solo	co-run*
Page reclaim	1.03	420.13
Page allocator	3.42	1,053.26
Dentry	2.93	1,298.87
Runqueue	1.22	256.07

** Concurrently running with Swaptions of PARSEC*

Processing time is amplified

❷ TLB synchronization latency (μsec)

		Avg.	Min.	Max.
dedup	solo	28	5	1927
	co-run*	6,354	7	74915
vips	solo	55	5	2052
	co-run*	14,928	17	121548

So, What Can Happen?

- Virtual time discontinuity

❶ Spinlock waiting time(gmake)

Kernel Component	Avg. waiting time (μsec)	
	solo	co-run*
Page reclaim	1.03	420.13
Page allocator	3.42	1,053.26
Dentry	2.93	1,298.87
Runqueue	1.22	256.07

** Concurrently running with Swaptions of PARSEC*

Processing time is amplified

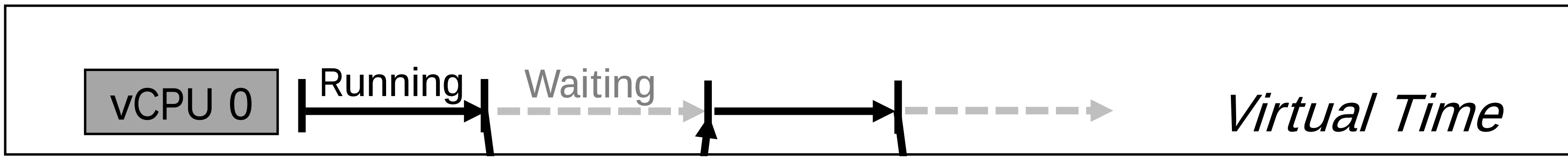
❷ TLB synchronization latency (μsec)

		Avg.	Min.	Max.
dedup	solo	28	5	1927
	co-run*	6,354	7	74915
vips	solo	55	5	2052
	co-run*	14,928	17	121548

	Jitters (ms)	Throughput (Mbits/sec)
solo	0.0043	936.3
mixed co-run*	9.2507	435.6

So, What Can Happen?

- Virtual time discontinuity



❶ Spinlock waiting time(`gmake`)

Kernel Component	Avg. waiting time (µsec)	
	solo	co-run*
Page reclaim	1.03	420.13
Page allocator	3.42	1,053.26
Dentry	2.93	1,298.87
Runqueue	1.22	256.07

** Concurrently running with Swaptions of PARSEC*

Processing time is amplified

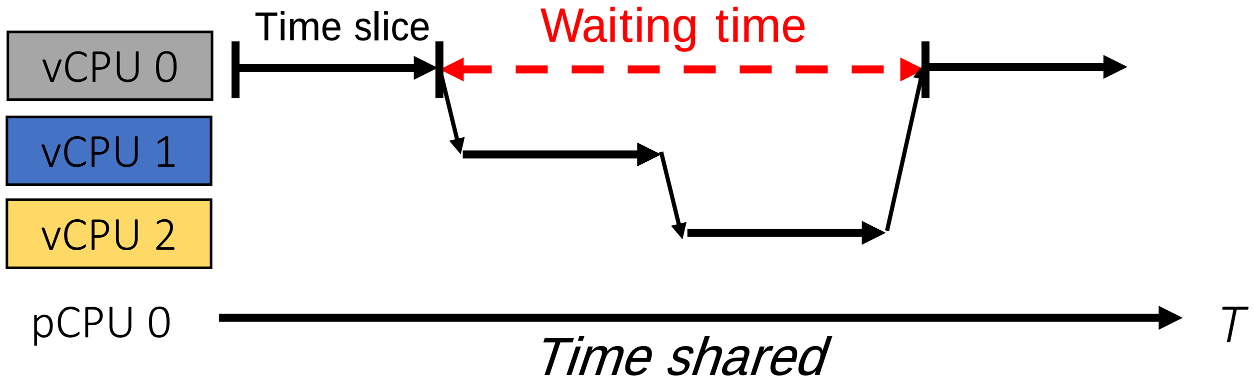
❷ TLB synchronization latency (µsec)

		Avg.	Min.	Max.
dedup	solo	28	5	1927
	co-run*	6,354	7	74915
vips	solo	55	5	2052
	co-run*	14,928	17	121548

❸ I/O latency & throughput (`iPerf`)

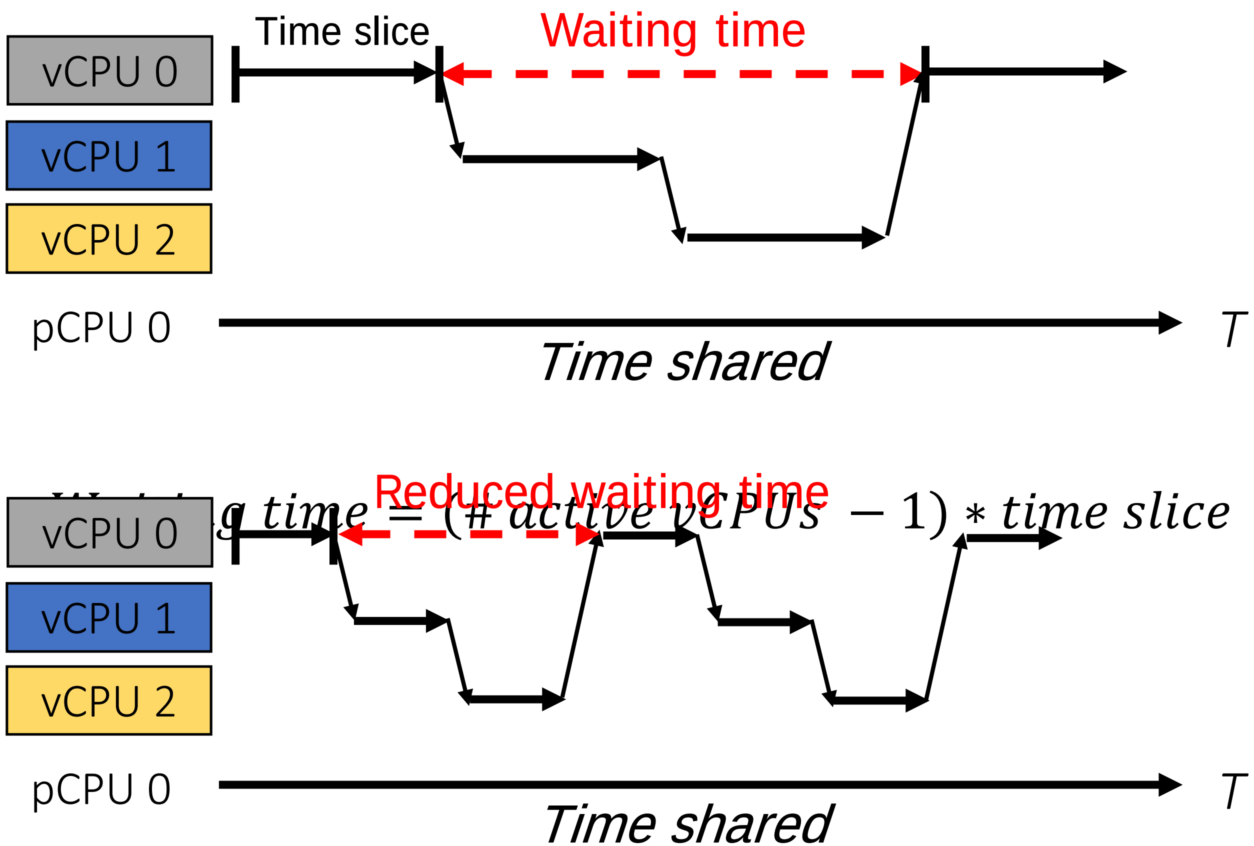
	Jitters (ms)	Throughput (Mbits/sec)
solo	0.0043	936.3
mixed co-run*	9.2507	435.6

How about Shortening Time Slice?



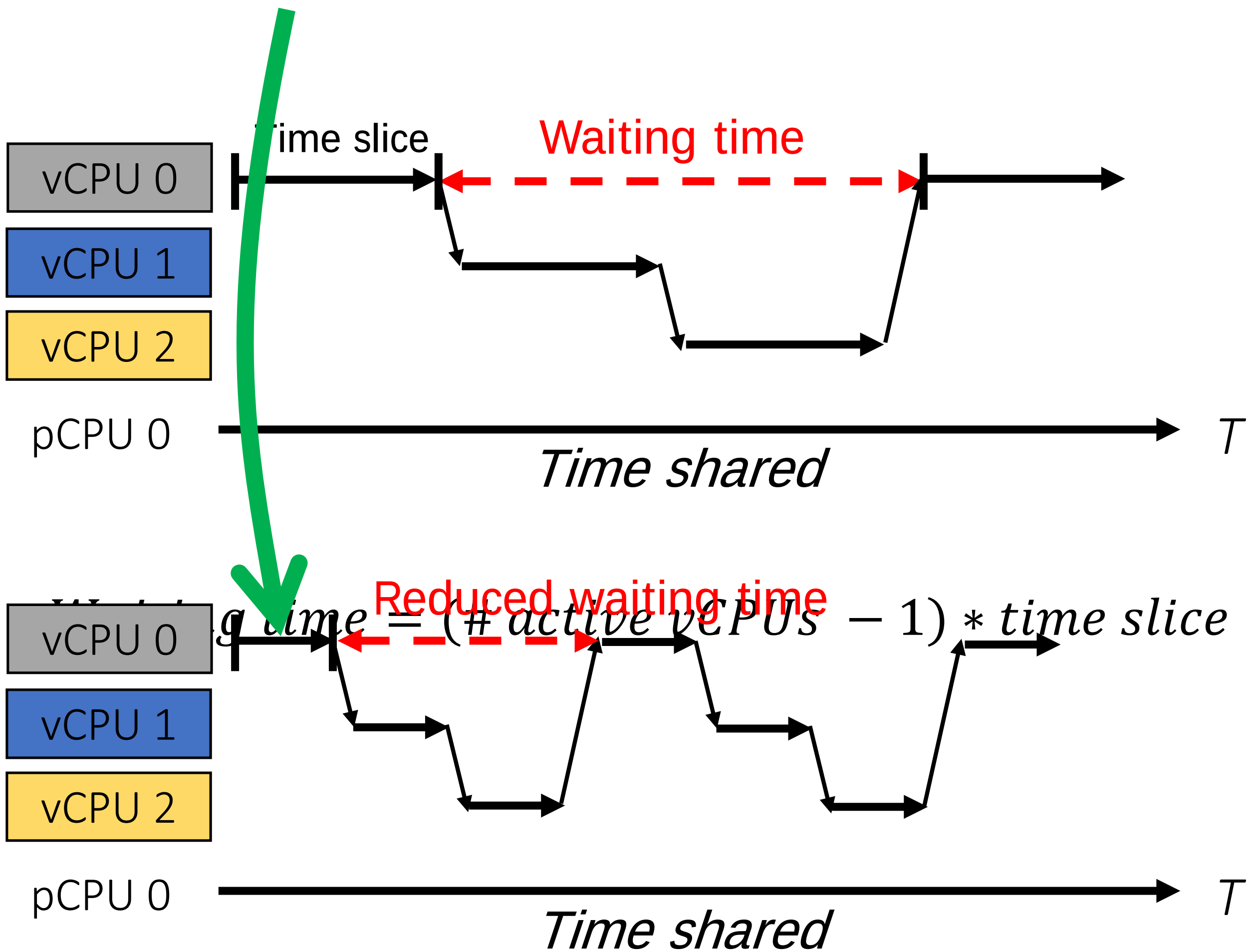
$$Waiting\ time = (\#\ active\ vCPUs\ - 1) * time\ slice$$

How about Shortening Time Slice?

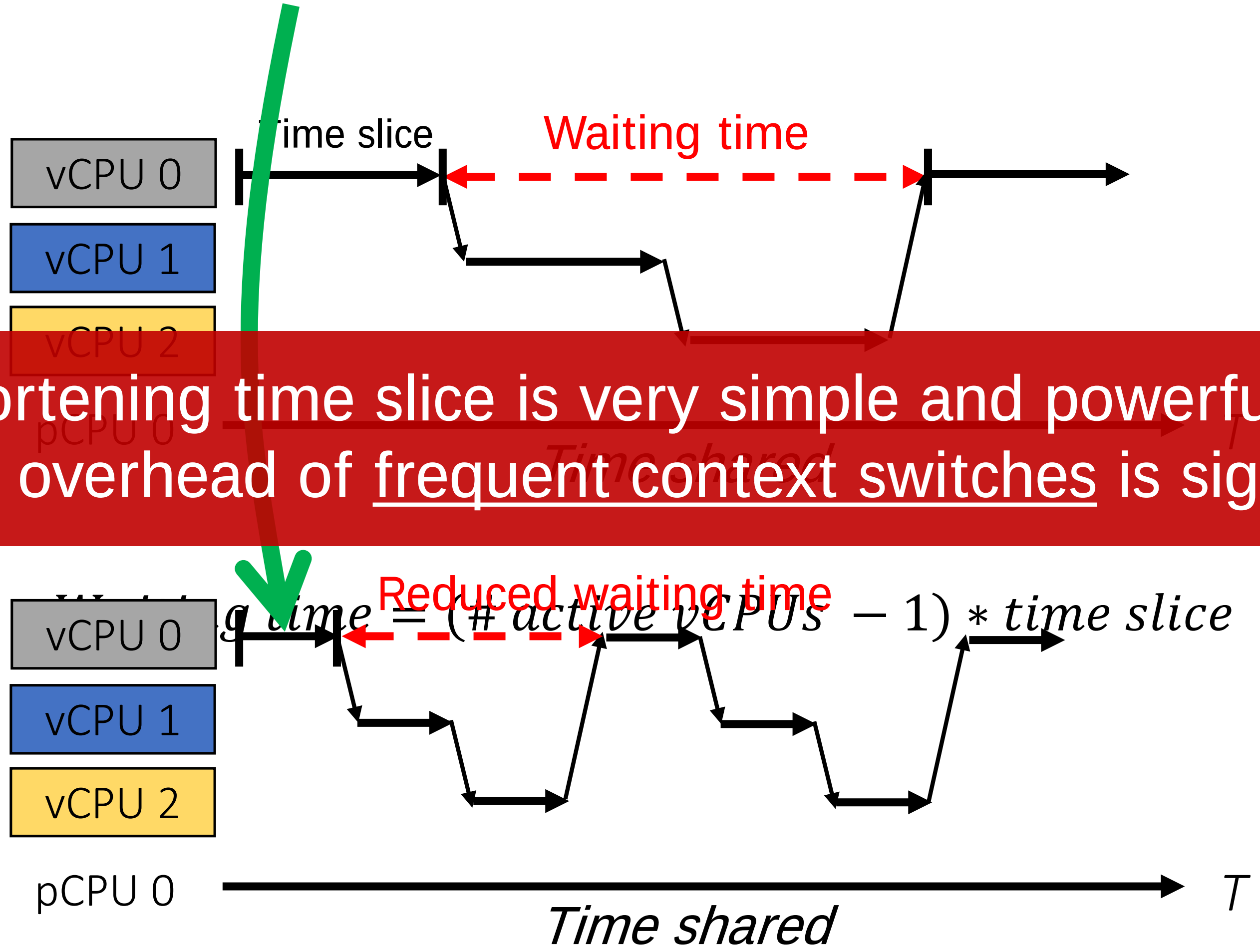


$$Waiting\ time = (\#\ active\ vCPUs - 1) * time\ slice$$

How about Shortening Time Slice?



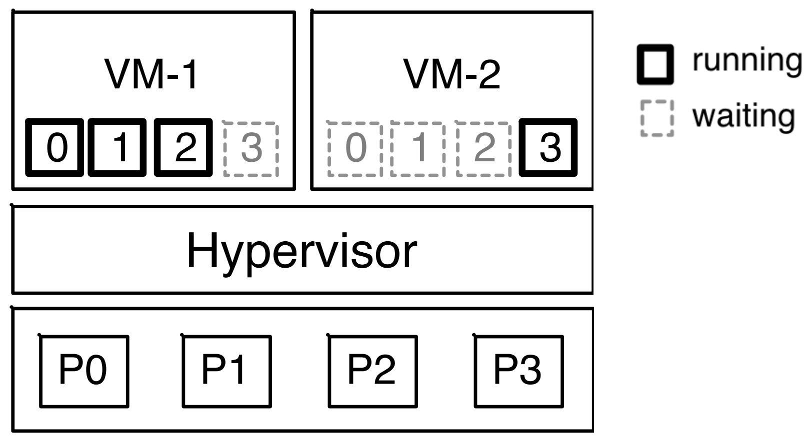
How about Shortening Time Slice?



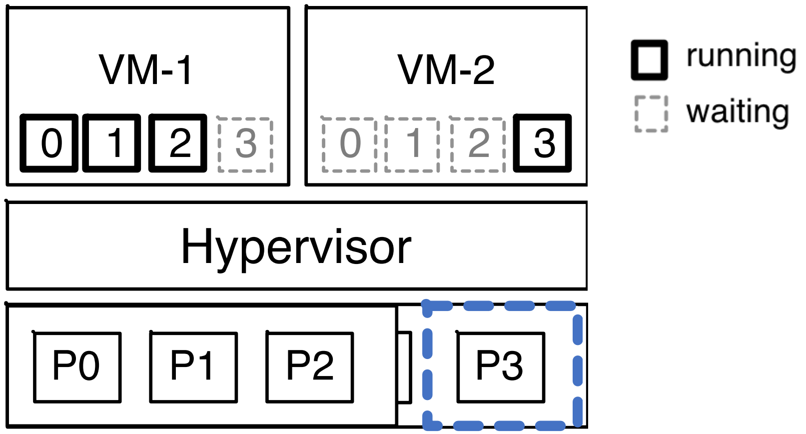
$$Waiting\ time = (\#\ active\ vCPUs - 1) * time\ slice$$

Shortening time slice is very simple and powerful, but the overhead of <u>frequent context switches</u> is significant

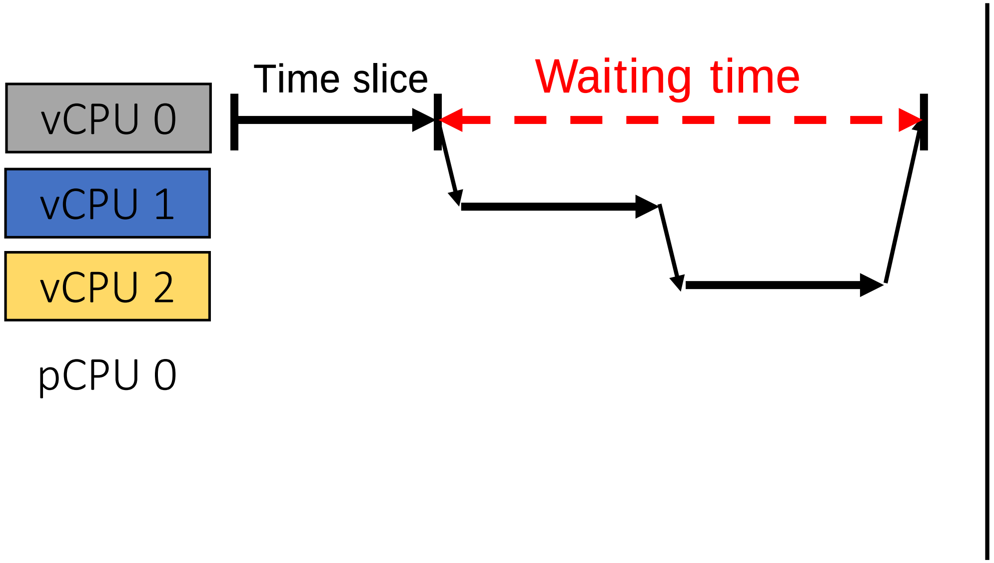
Approach: Dividing CPUs into Two Pools



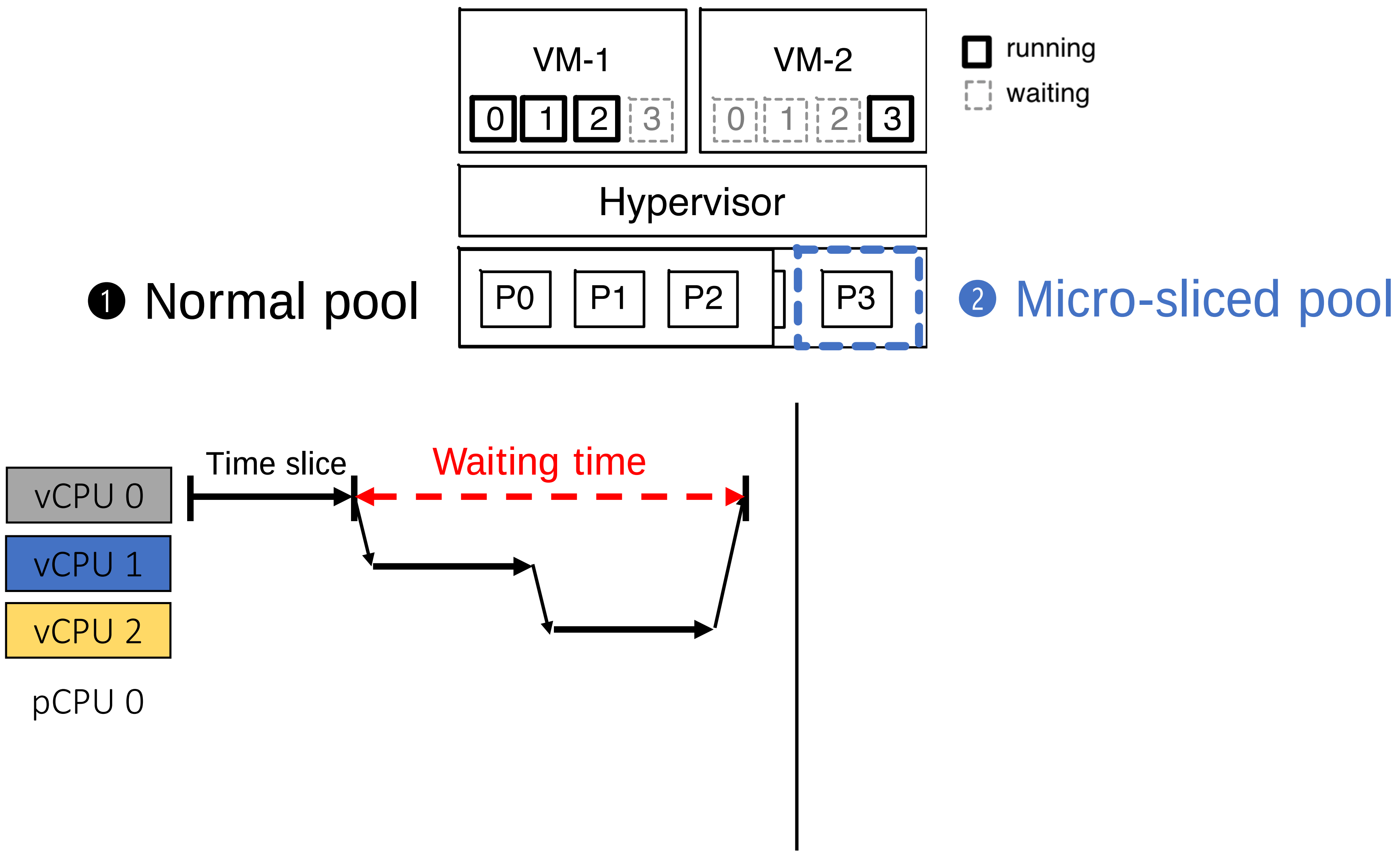
Approach: Dividing CPUs into Two Pools



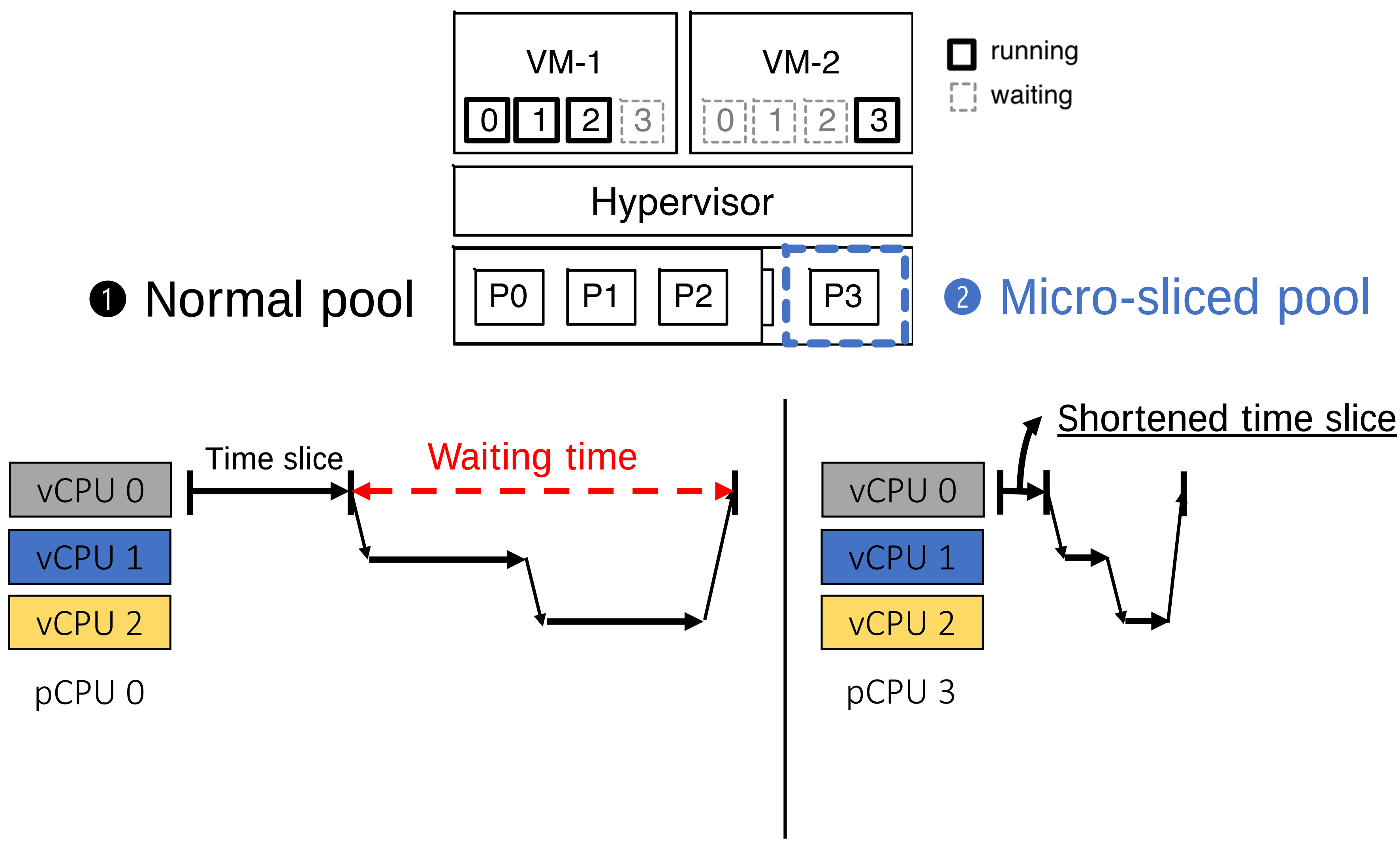
❶ Normal pool



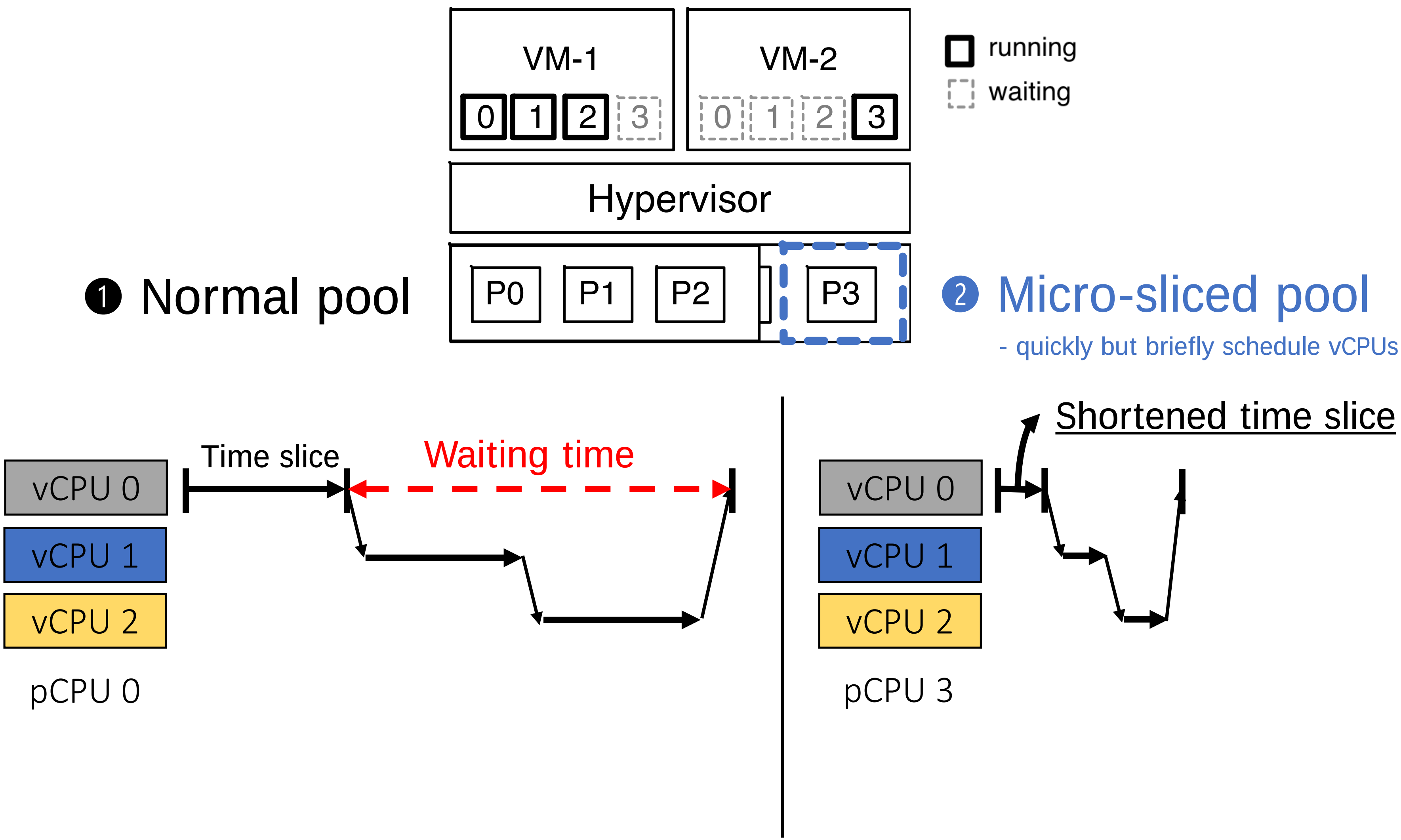
Approach: Dividing CPUs into Two Pools



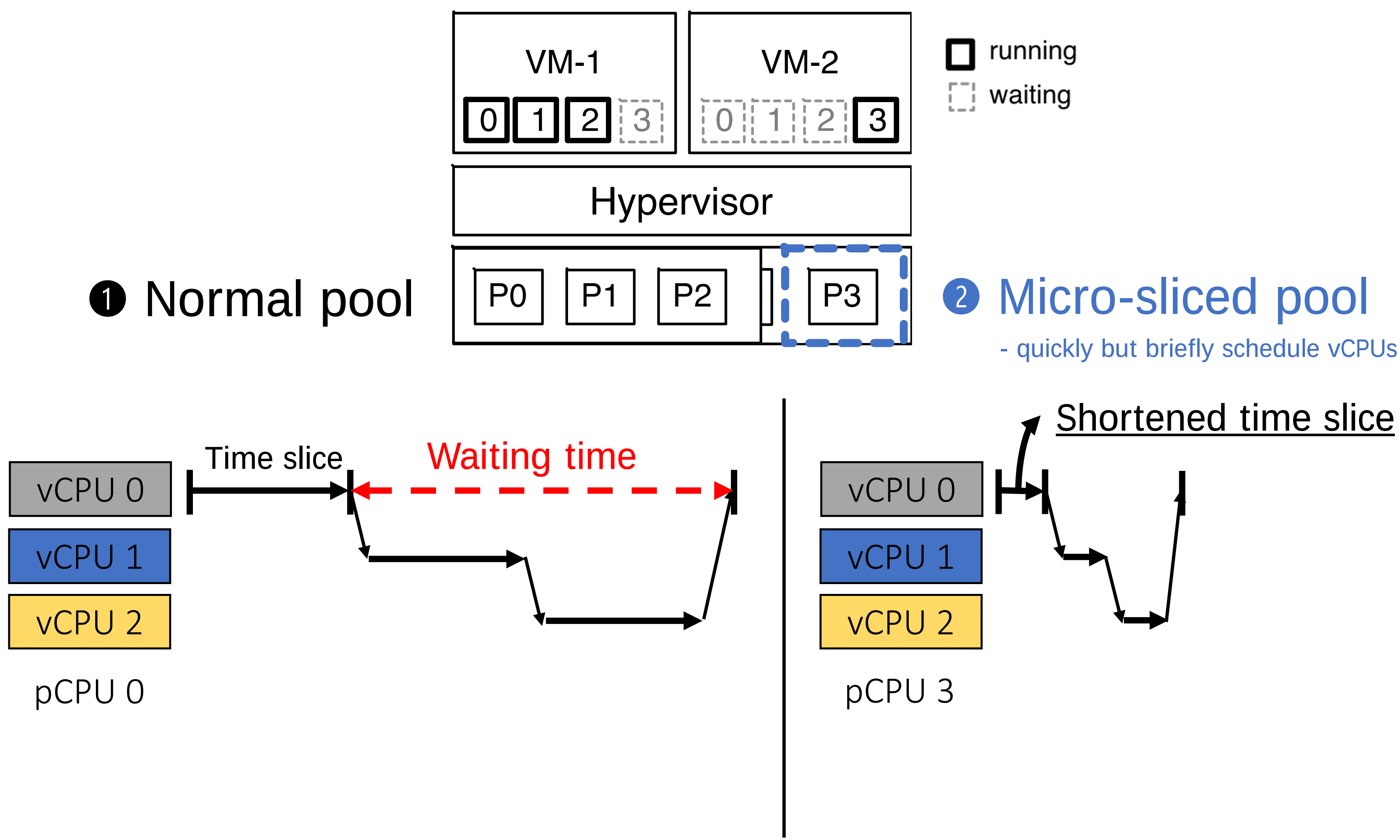
Approach: Dividing CPUs into Two Pools



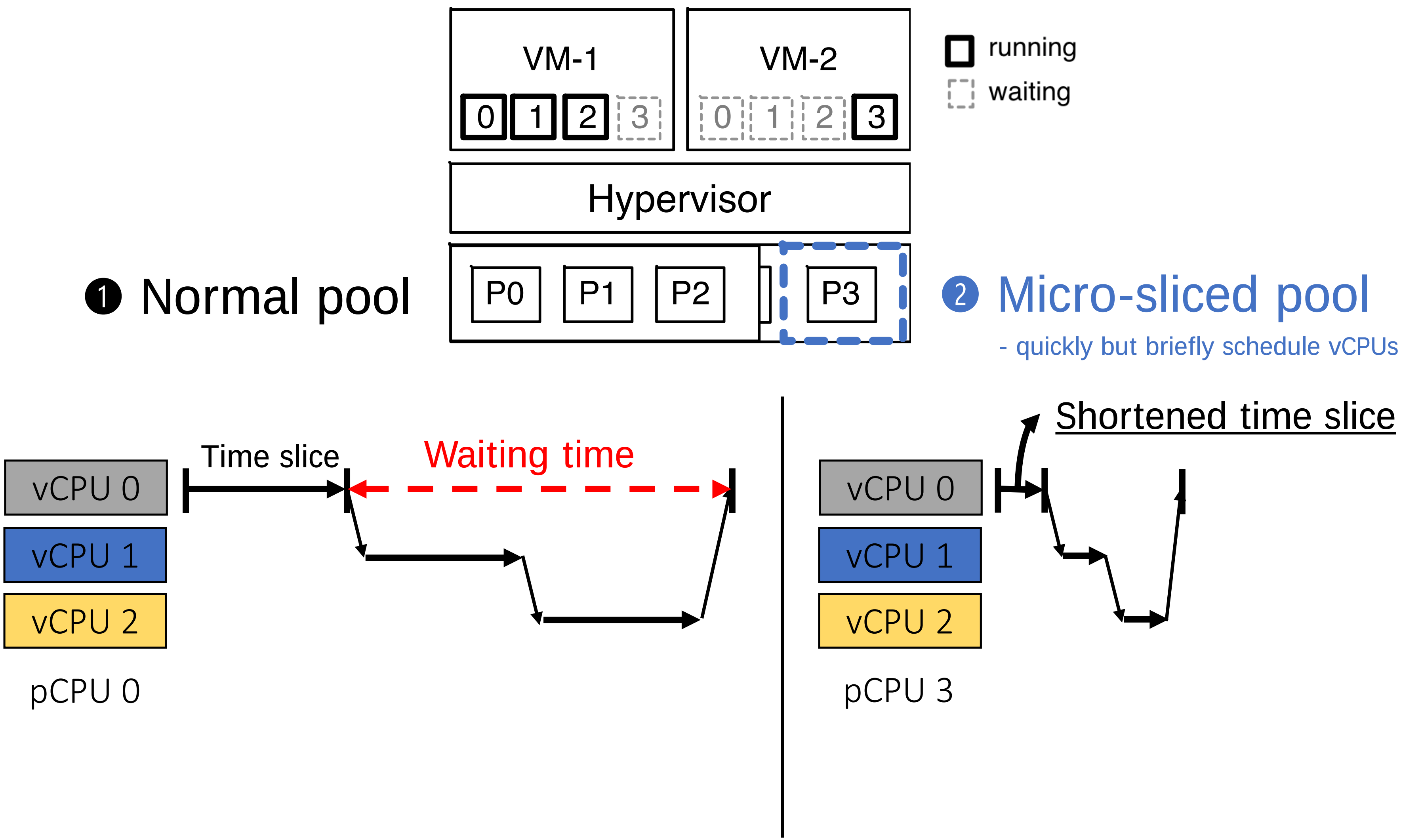
Approach: Dividing CPUs into Two Pools



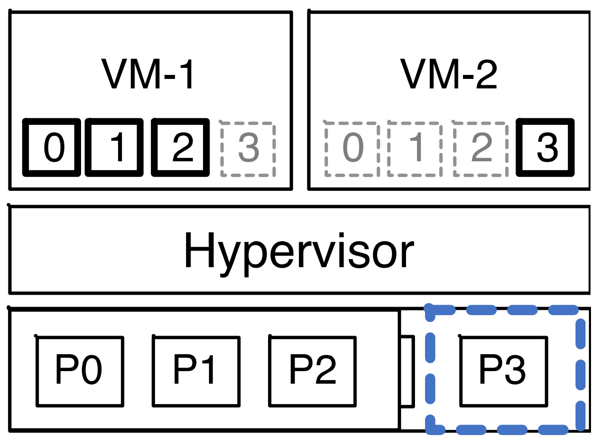
Approach: Dividing CPUs into Two Pools



Approach: Dividing CPUs into Two Pools



Approach: Dividing CPUs into Two Pools

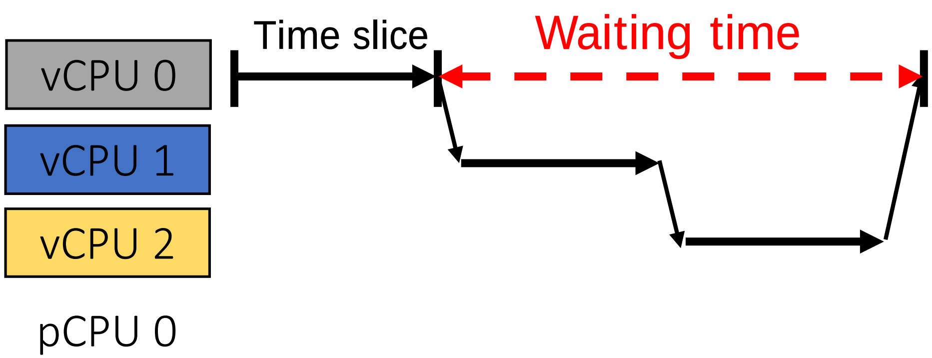


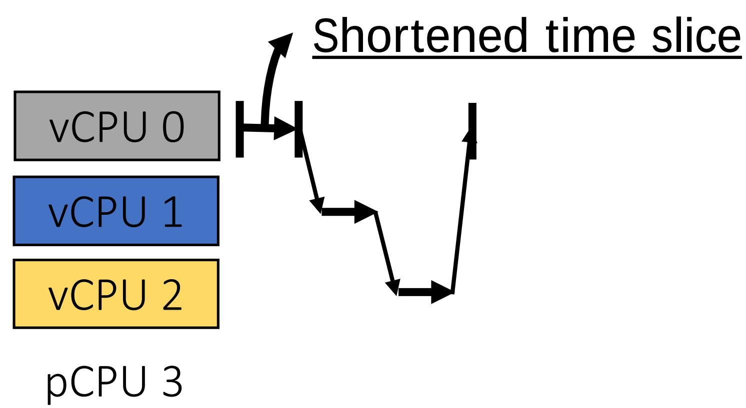
running
waiting

❶ Normal pool

❷ Micro-sliced pool

- quickly but briefly schedule vCPUs

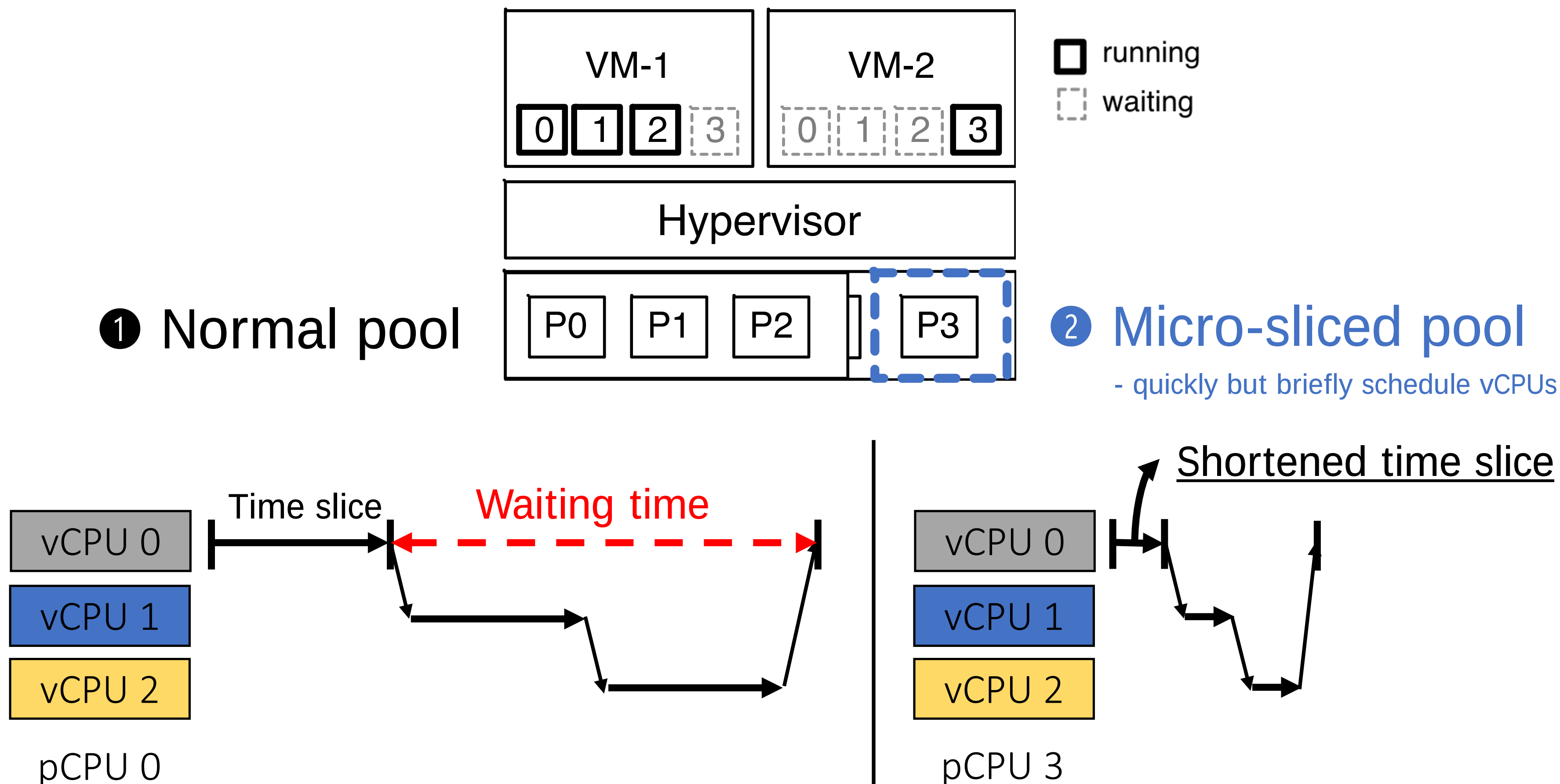




Serving the main work of applications

Serving critical OS services to minimize the *waiting time*

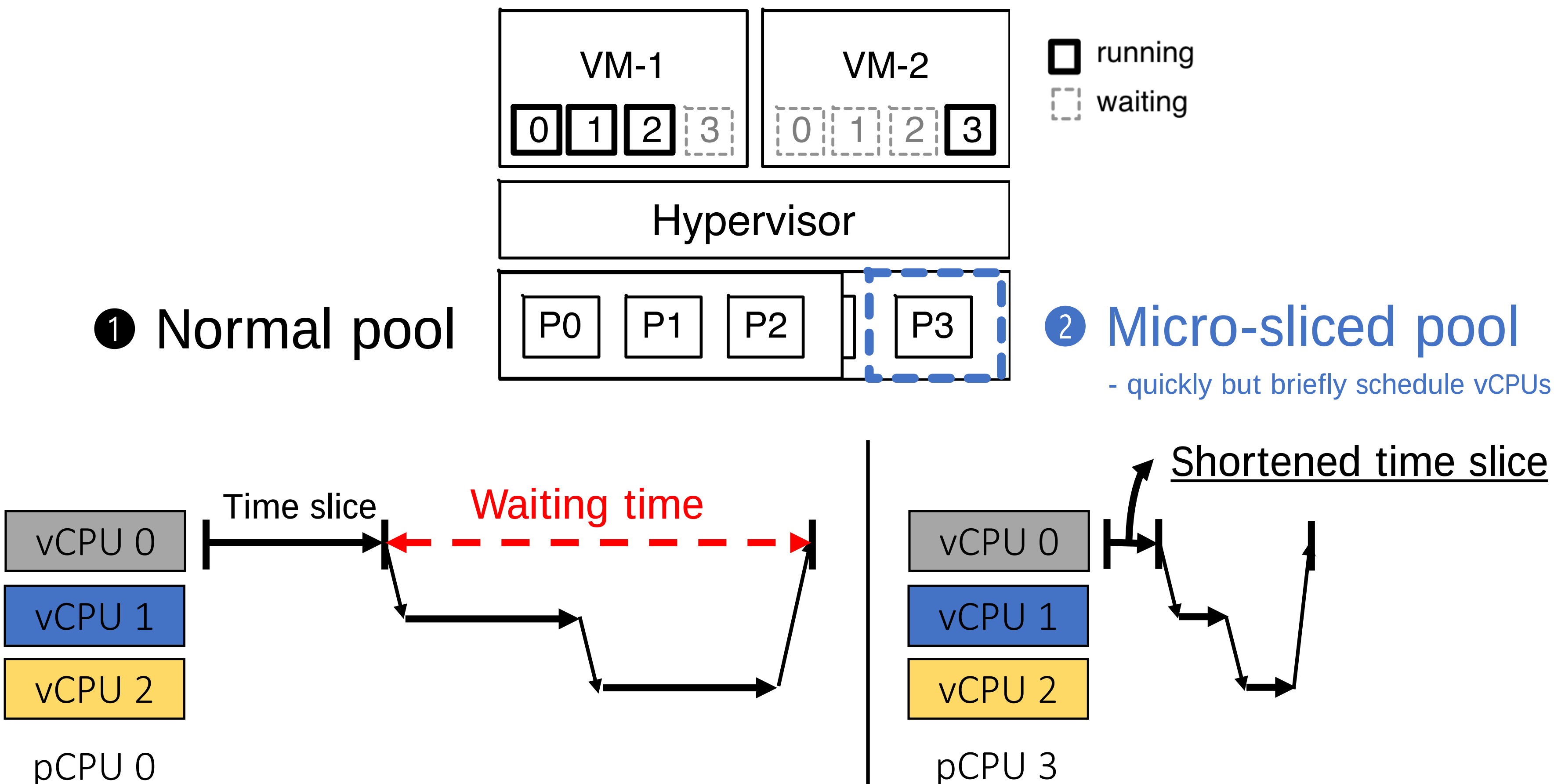
Approach: Dividing CPUs into Two Pools



Serving the main work of applications

Serving critical OS services to minimize the *waiting time*

Approach: Dividing CPUs into Two Pools



Serving the main work of applications

Serving critical OS services to minimize the *waiting time*

Challenges in Serving Critical OS Services on Micro-sliced Cores

Challenges in Serving Critical OS Services on Micro-sliced Cores

1. Precise detection of urgent tasks

Challenges in Serving Critical OS Services on Micro-sliced Cores

1. Precise detection of urgent tasks

2. Guest OS transparency

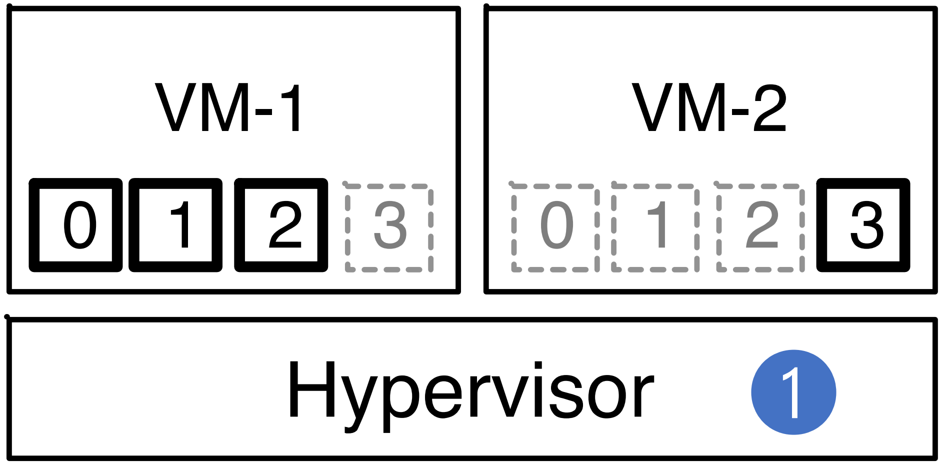
Challenges in Serving Critical OS Services on Micro-sliced Cores

1. Precise detection of urgent tasks

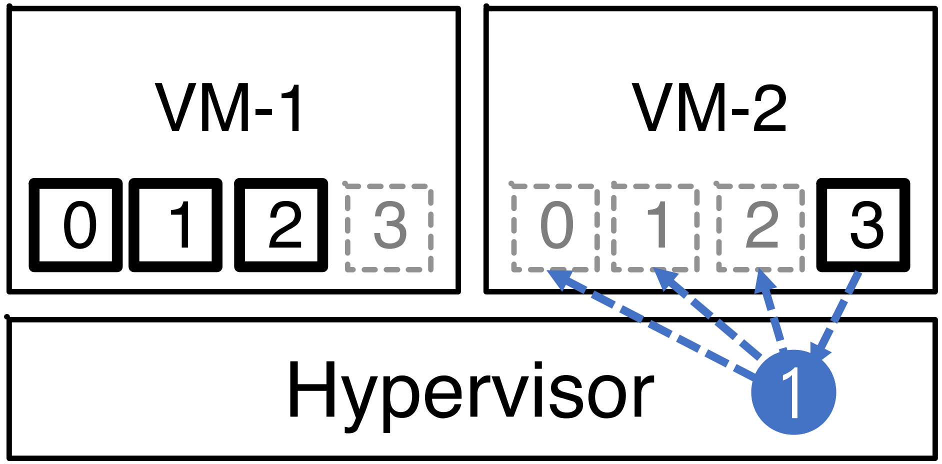
2. Guest OS transparency

3. Dynamic adjustment of micro-sliced cores

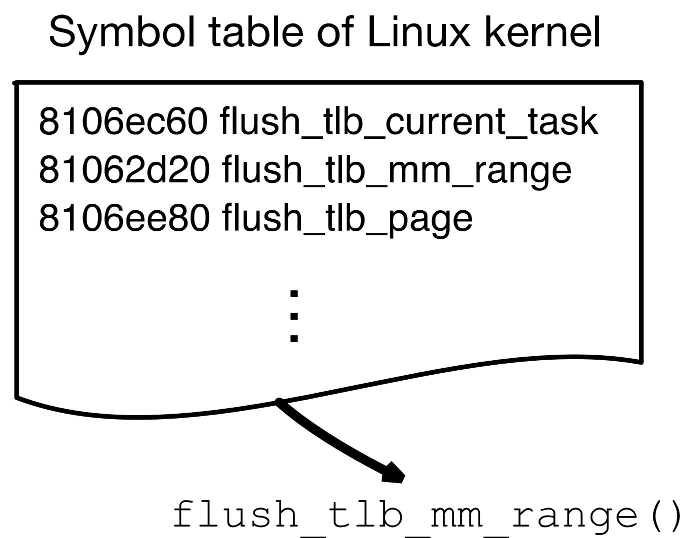
Detecting Critical OS Services



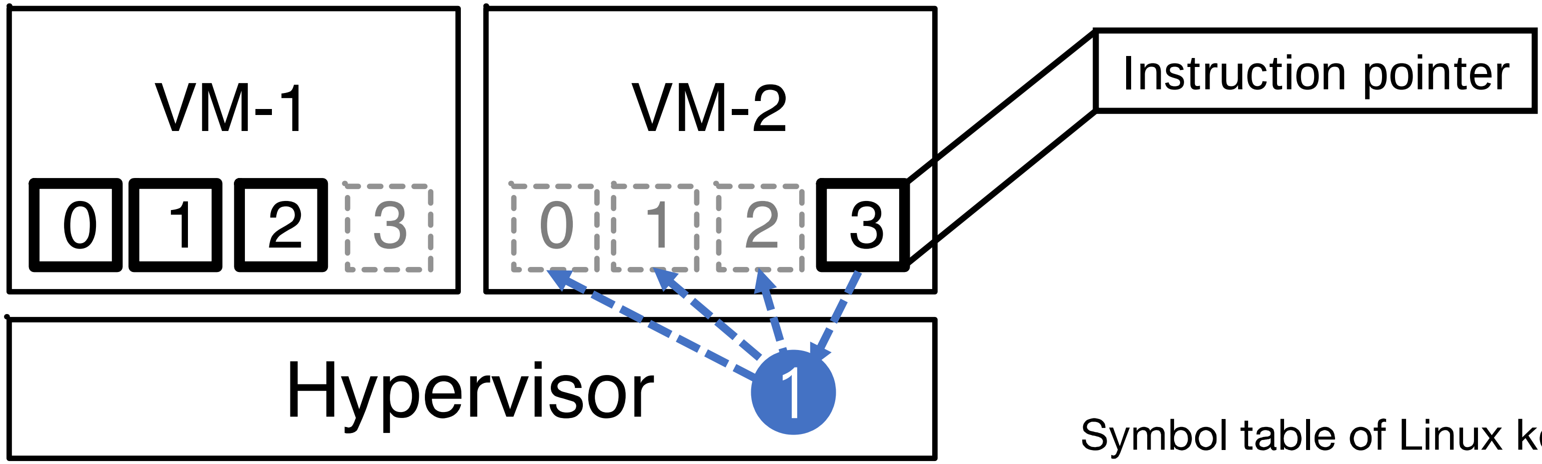
Detecting Critical OS Services



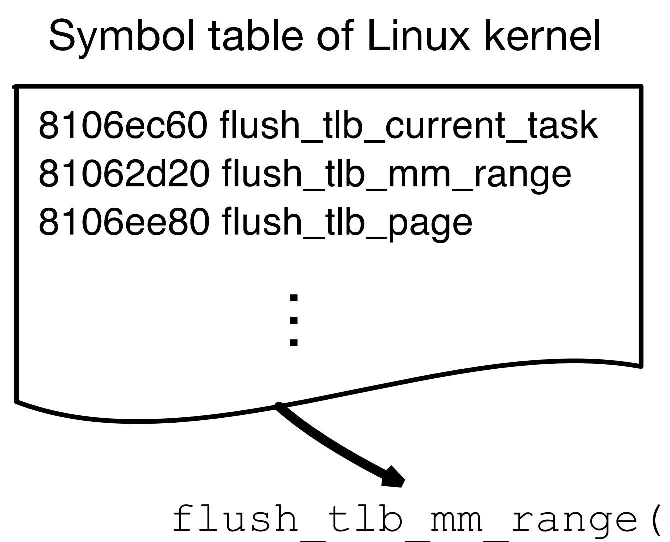
Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU



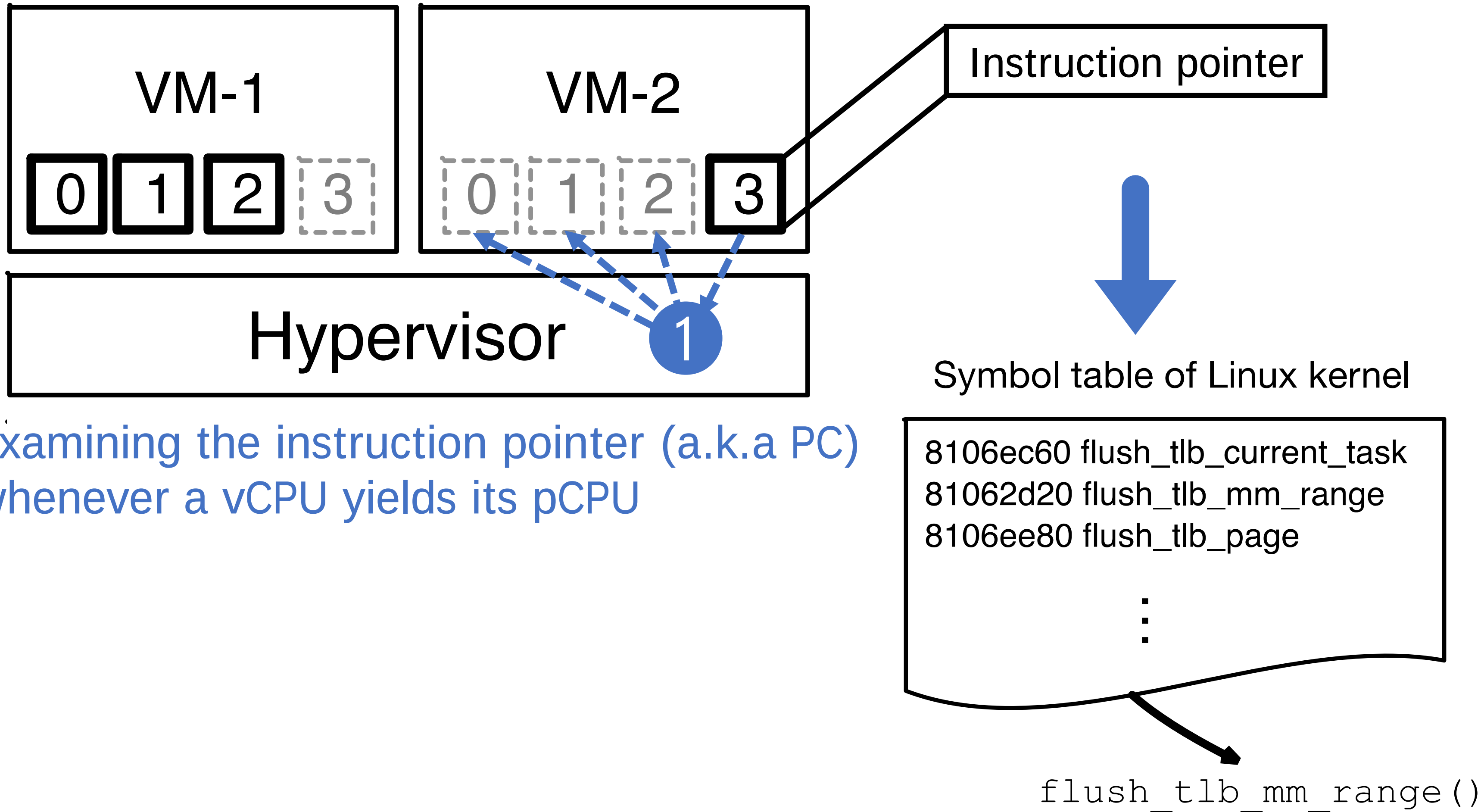
Detecting Critical OS Services



Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

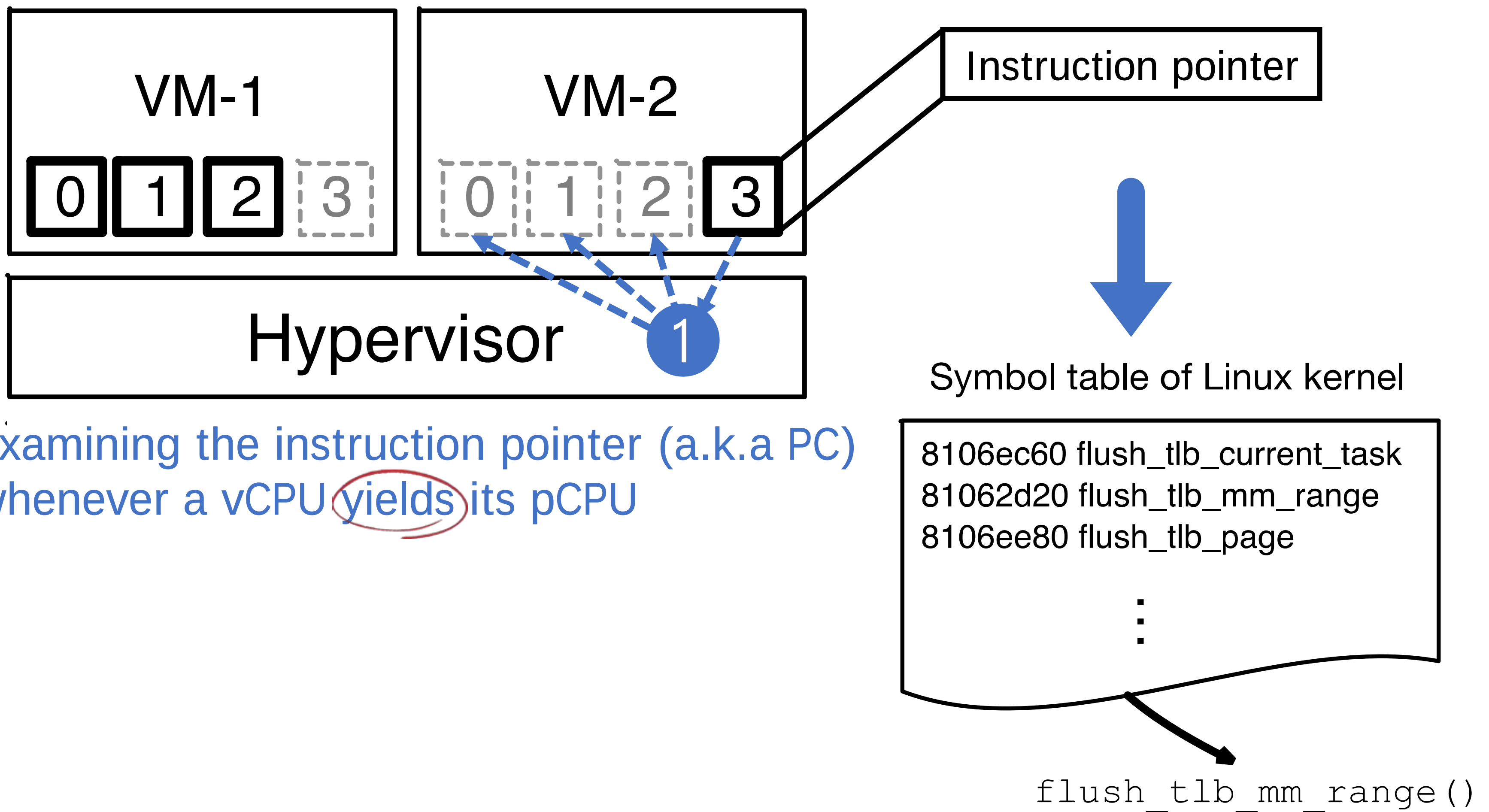


Detecting Critical OS Services



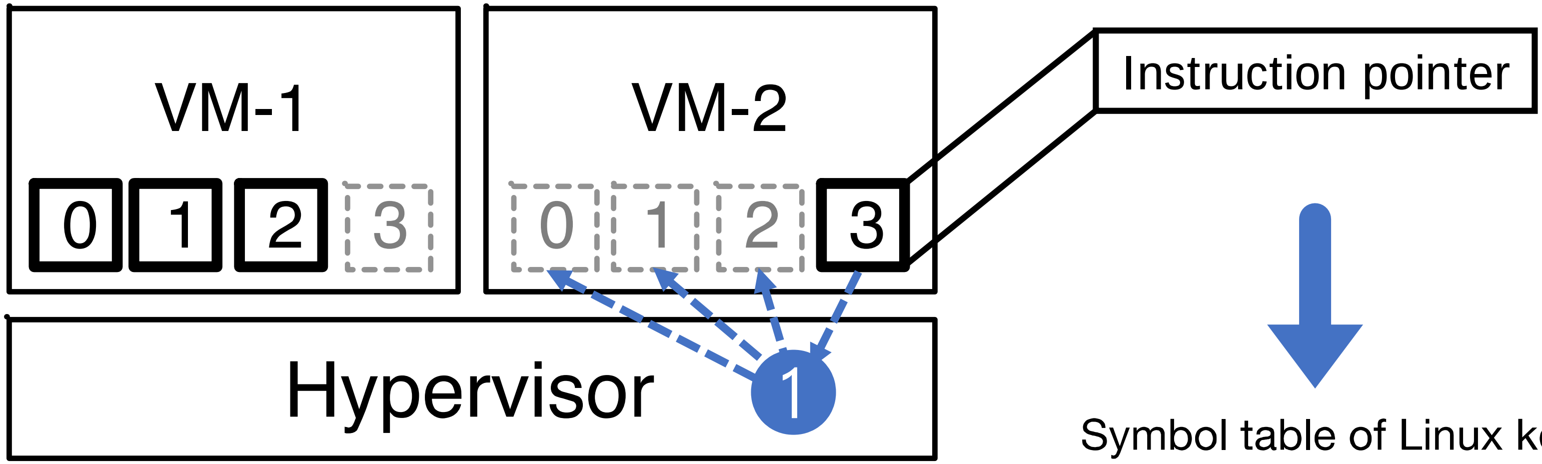
Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

Detecting Critical OS Services

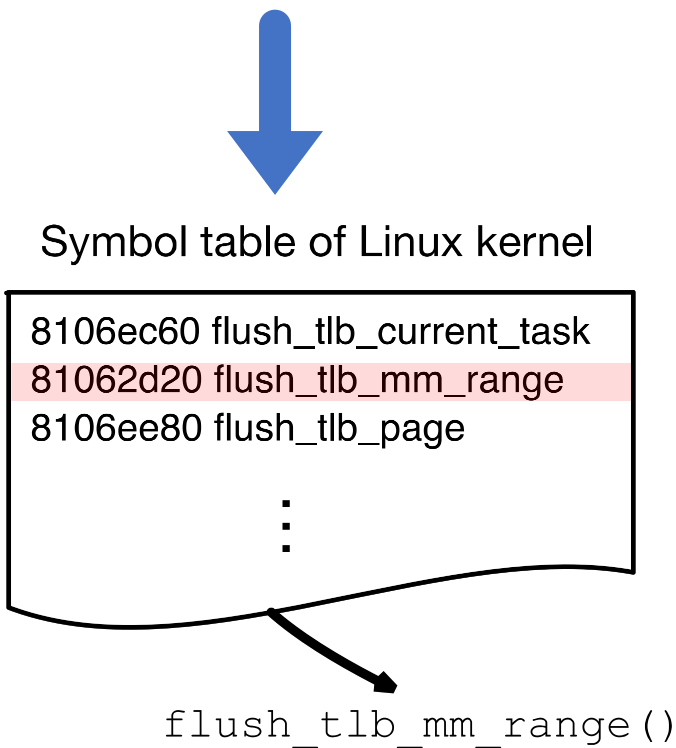


Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

Detecting Critical OS Services

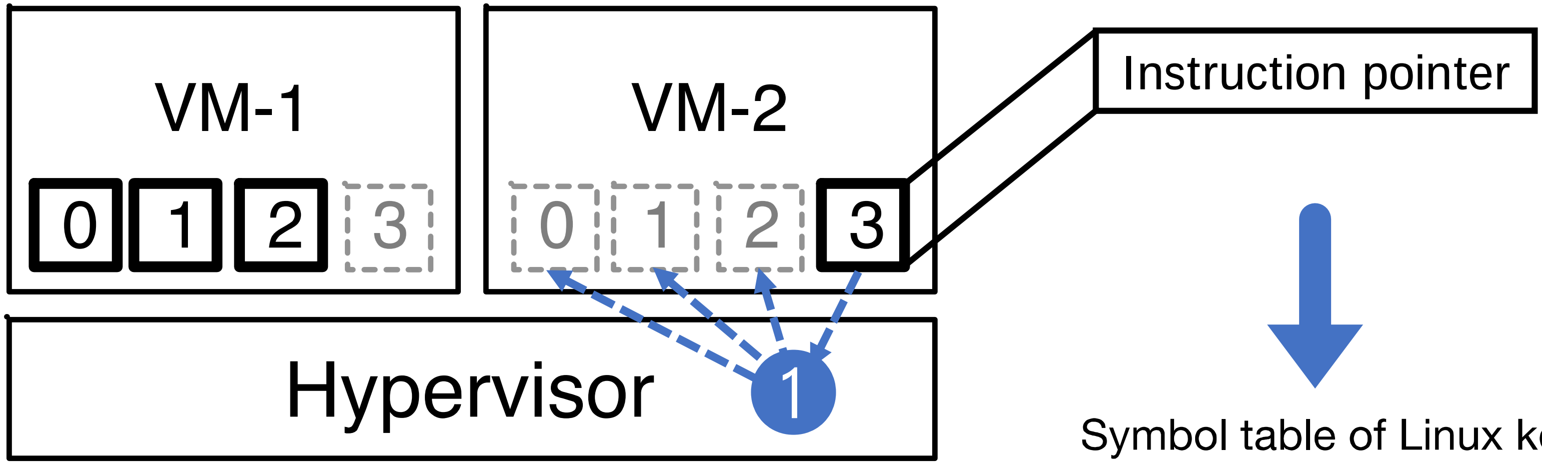


Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU



workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

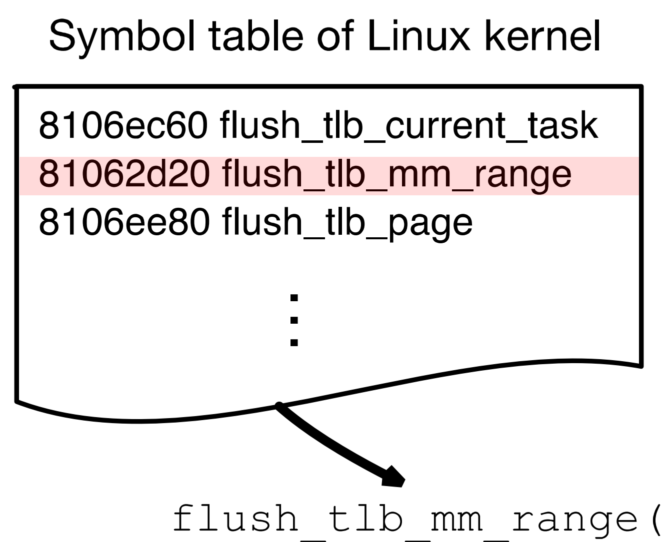
Detecting Critical OS Services



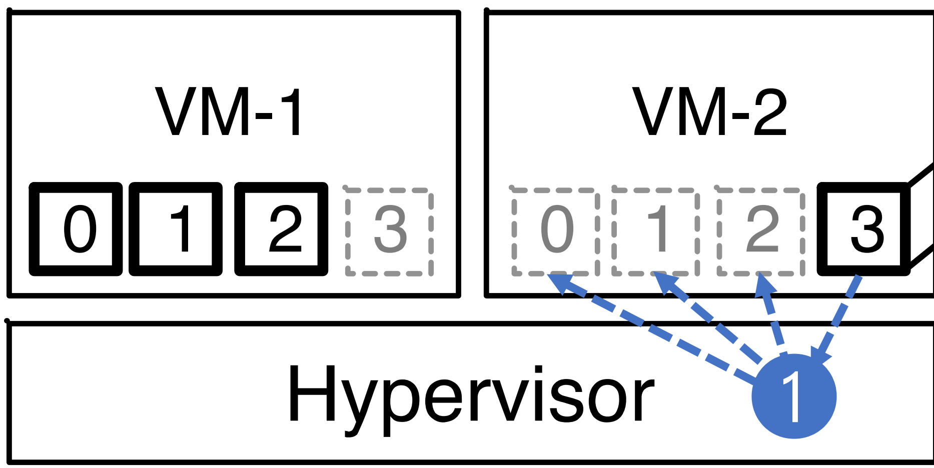
Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

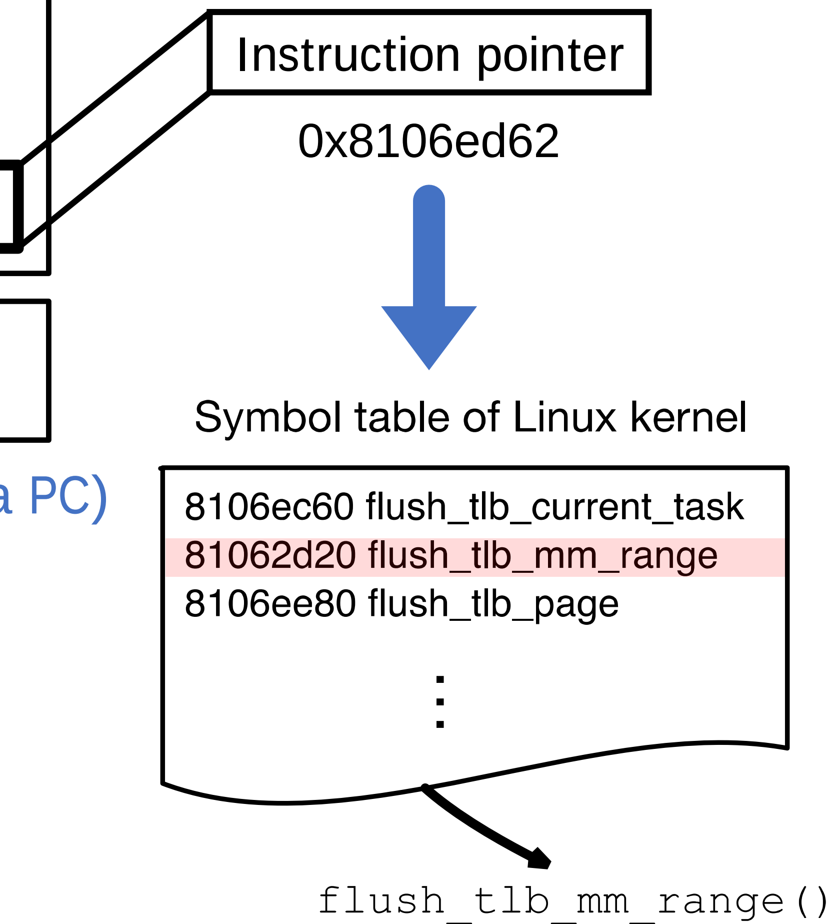
workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

** Concurrently running with Swaptions of PARSEC*



Detecting Critical OS Services





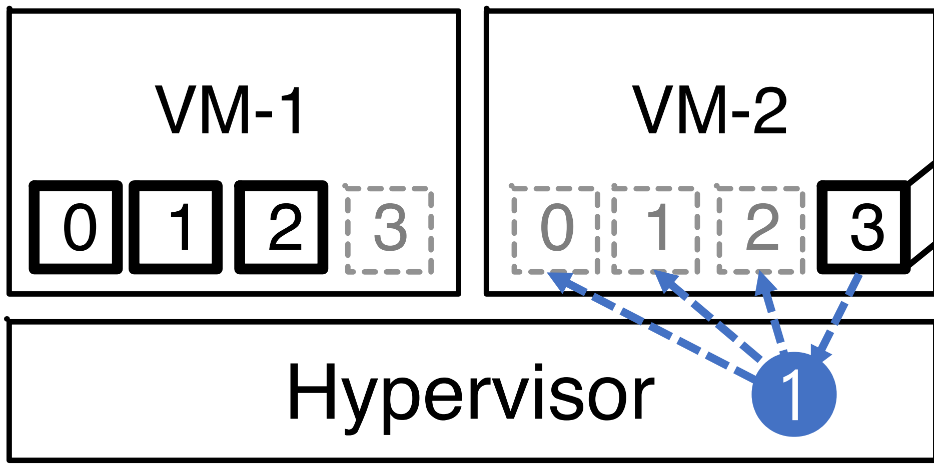
Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

** Concurrently running with Swaptions of PARSEC*

May need to inspect vCPU siblings

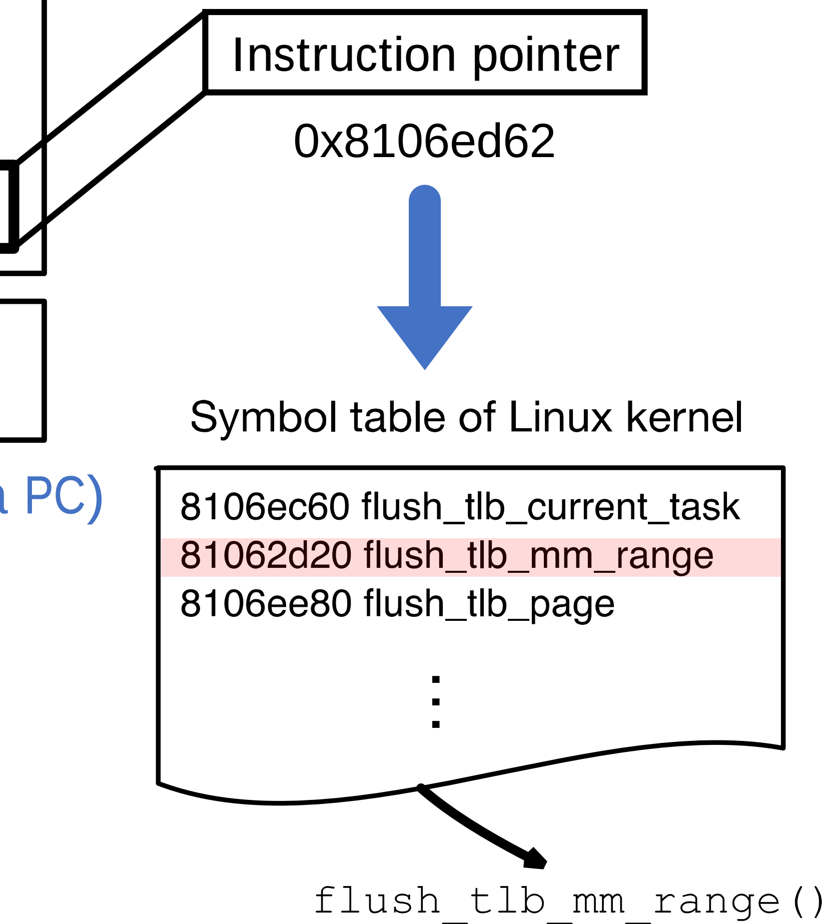
Detecting Critical OS Services



Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

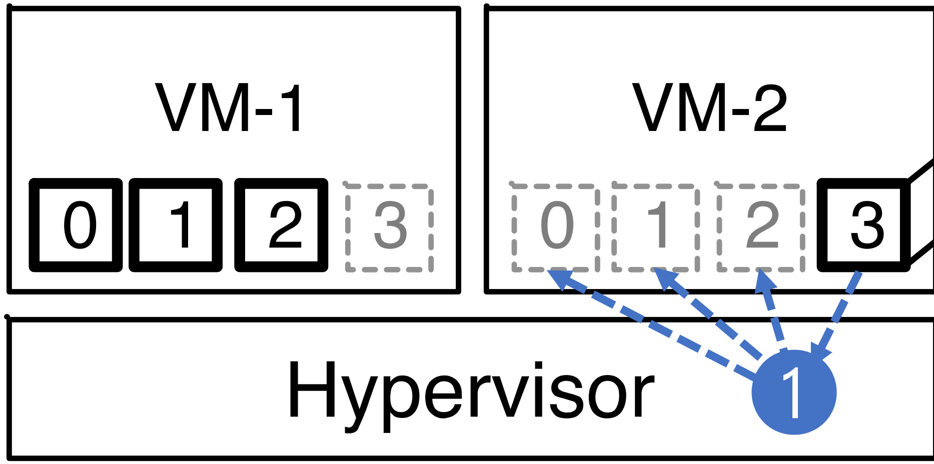
workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

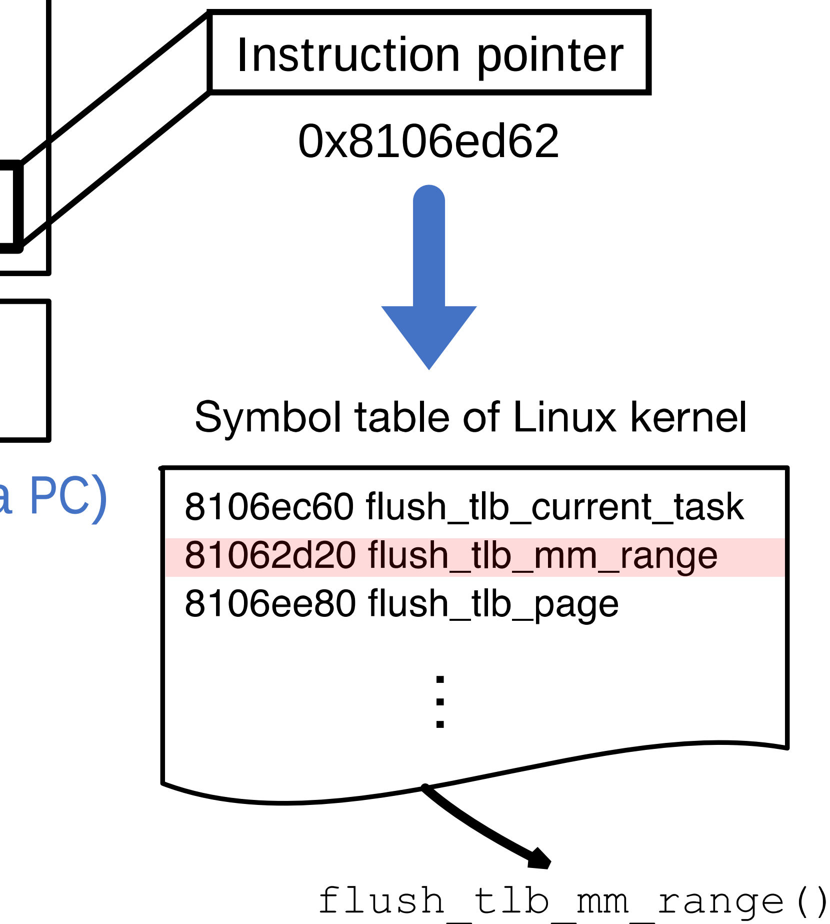
** Concurrently running with Swaptions of PARSEC*



May need to inspect vCPU siblings

Detecting Critical OS Services





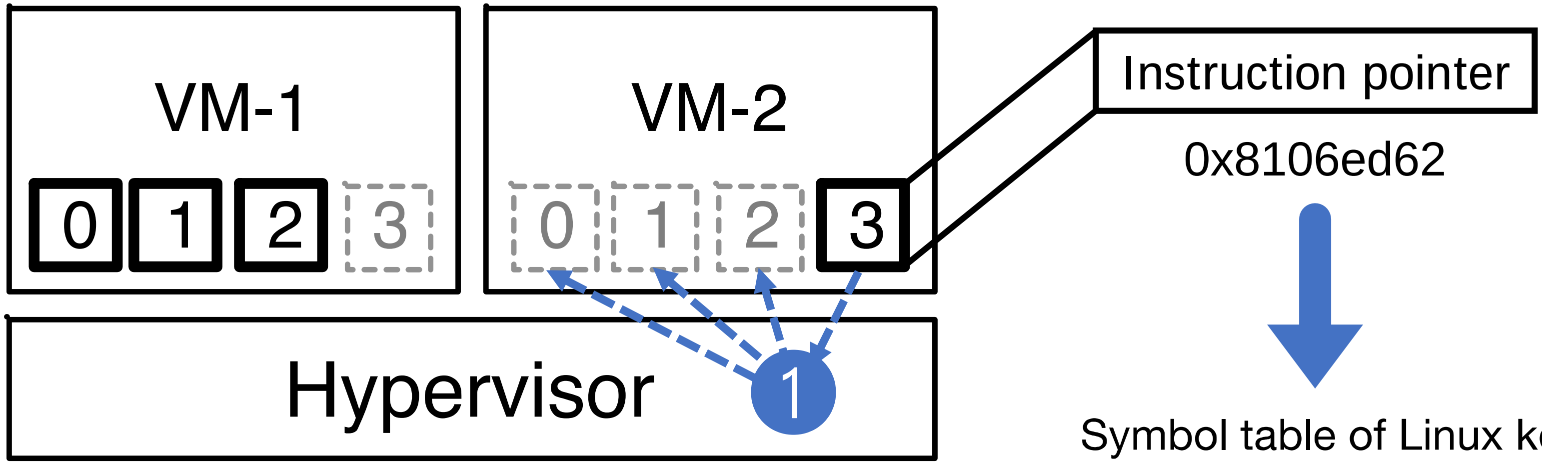
Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

** Concurrently running with Swaptions of PARSEC*

May need to inspect vCPU siblings

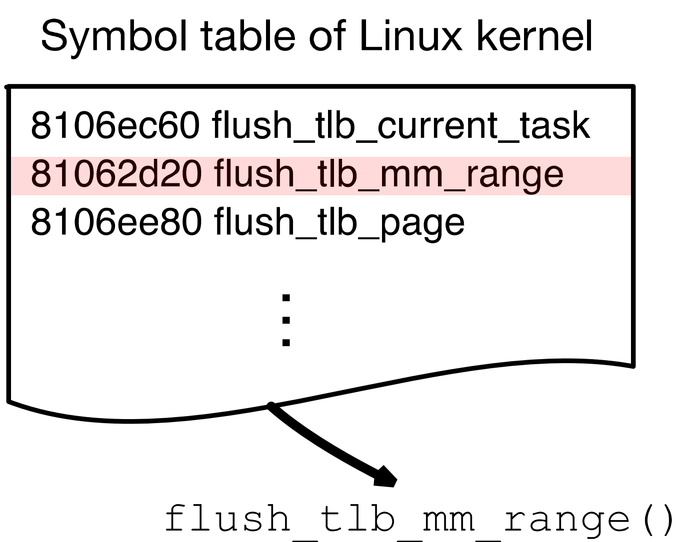
Detecting Critical OS Services



Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

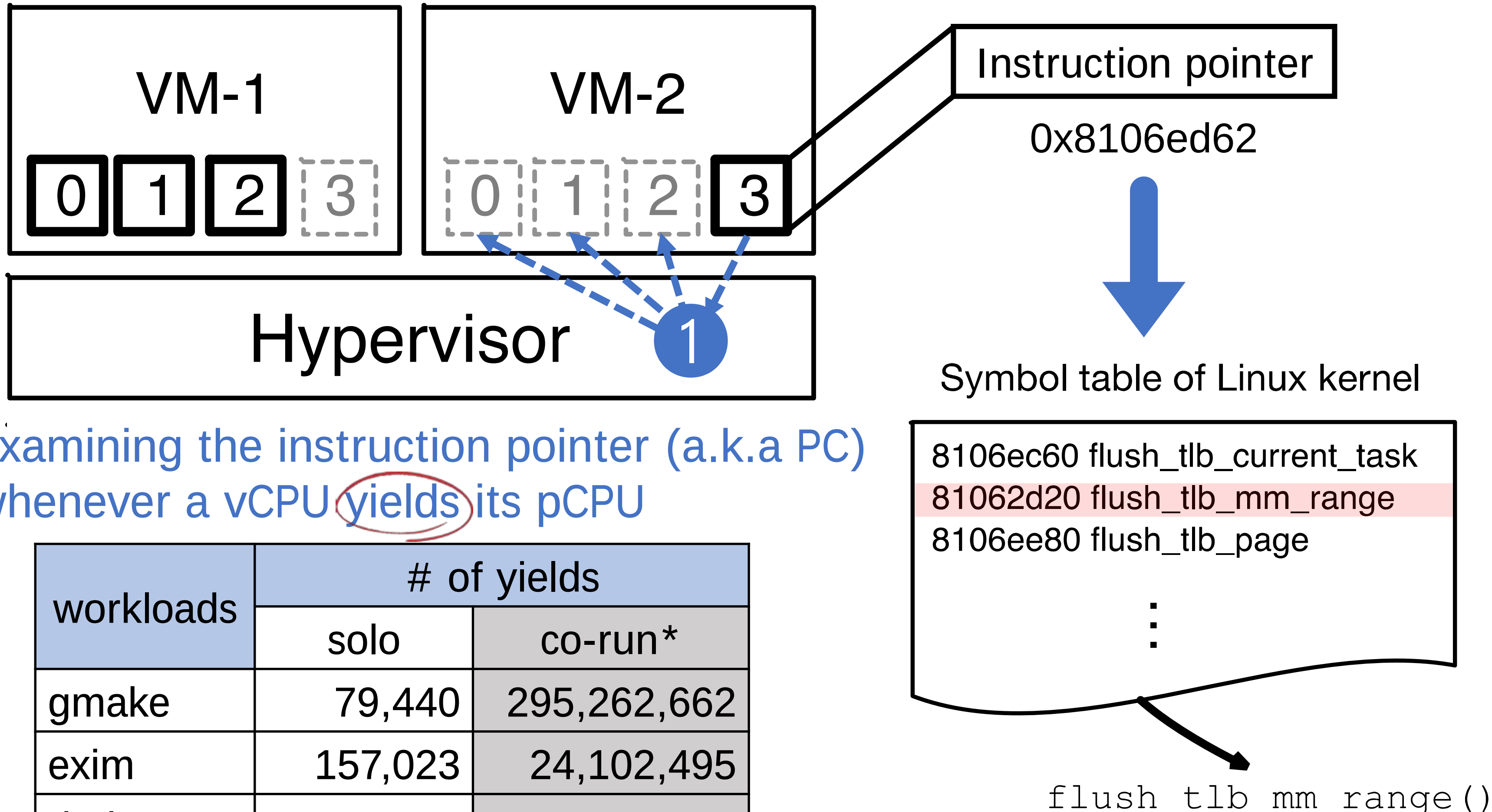
workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

** Concurrently running with Swaptions of PARSEC*



May need to inspect vCPU siblings

Detecting Critical OS Services



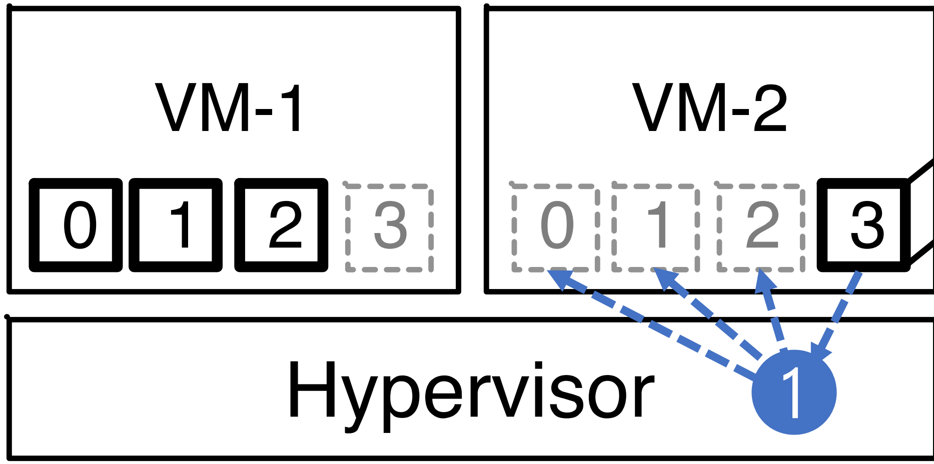
Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

** Concurrently running with Swaptions of PARSEC*

May need to inspect vCPU siblings

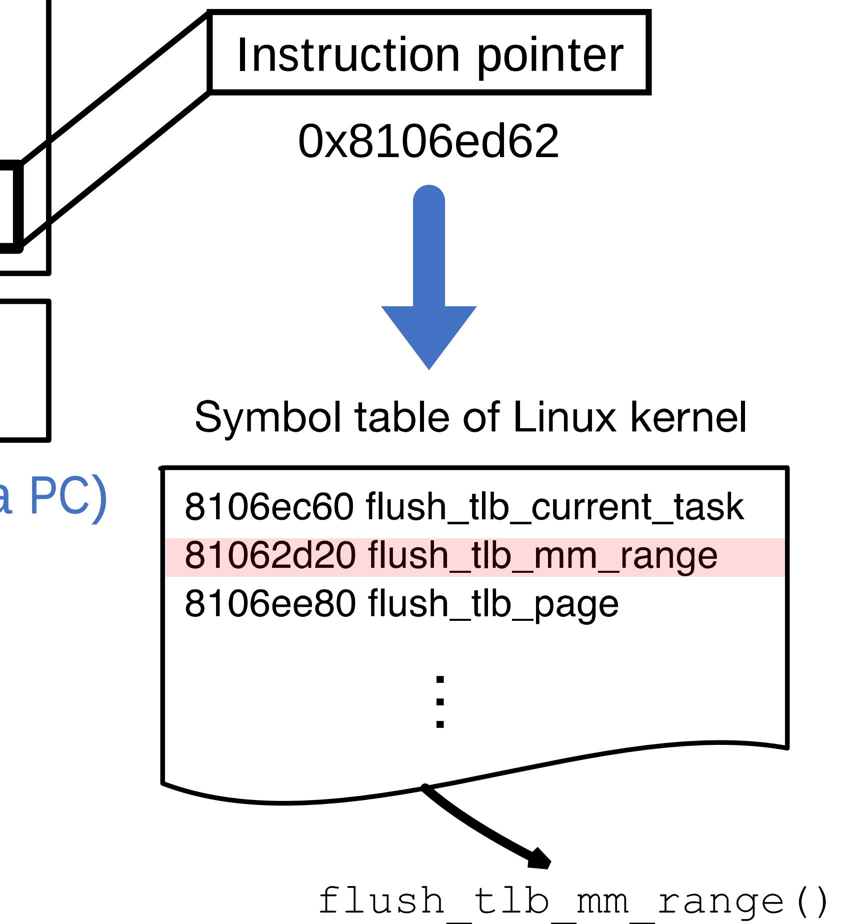
Detecting Critical OS Services



Examining the instruction pointer (a.k.a PC) whenever a vCPU yields its pCPU

workloads	# of yields	
	solo	co-run*
gmake	79,440	295,262,662
exim	157,023	24,102,495
dedup	290,406	164,578,839
vips	644,643	57,650,538

** Concurrently running with Swaptions of PARSEC*

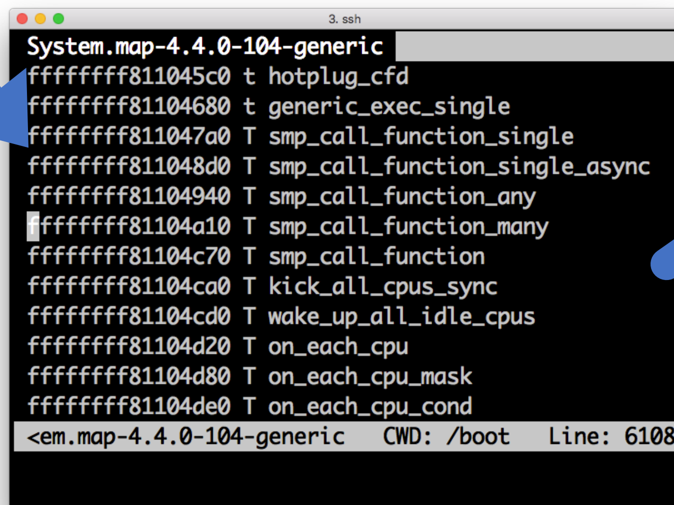


May need to inspect vCPU siblings

Profiling Virtual CPU Scheduling Logs

- Investigating frequently preempted regions

vCPU scheduling trace
(w/ `Inst. Pointer`)



Kernel symbol tables

Critical Guest OS Components

Module	Operation

sched	`scheduler_ipi()`
	`resched_curr()`
	…

mm	`flush_tlb_all()`
	`get_page_from_freelist()`
	…

irq	`irq_enter()`
	`irq_exit()`
	…

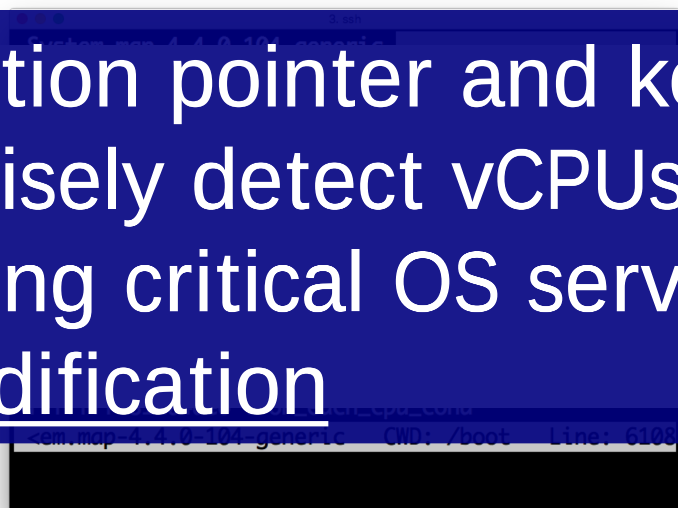
spinlock	`__raw_spin_unlock()`
	`__raw_spin_unlock_irq()`
	…

In our paper, you can find the table in details

Profiling Virtual CPU Scheduling Logs

- Investigating frequently preempted regions

vCPU scheduling trace
(w/ `Inst. Pointer`)



Kernel symbol tables

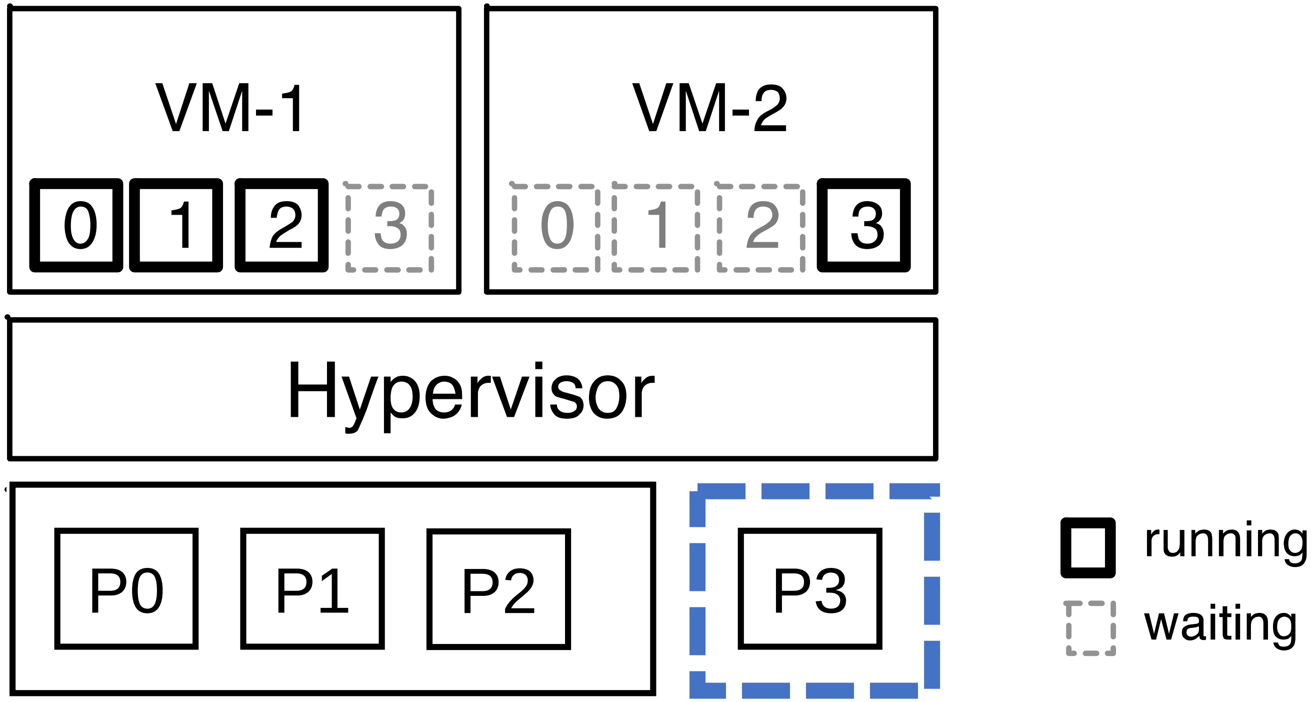
Critical Guest OS Components

Module	Operation
sched	`scheduler_ipi()`
	`resched_curr()`
spinlock	`__raw_spin_unlock_irq()`
	...

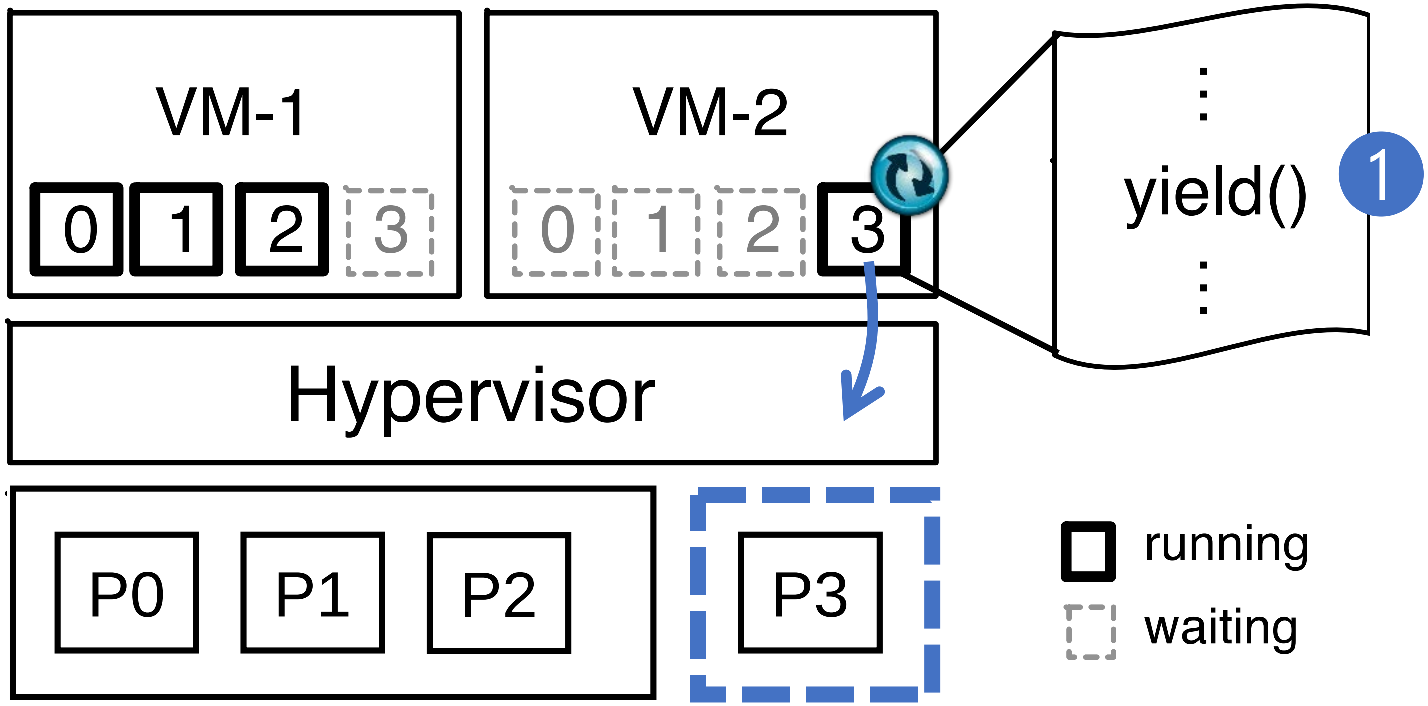
In our paper, you can find the table in details

Instruction pointer and kernel symbols enable to precisely detect vCPUs preempted while executing critical OS services <u>without guest OS modification</u>

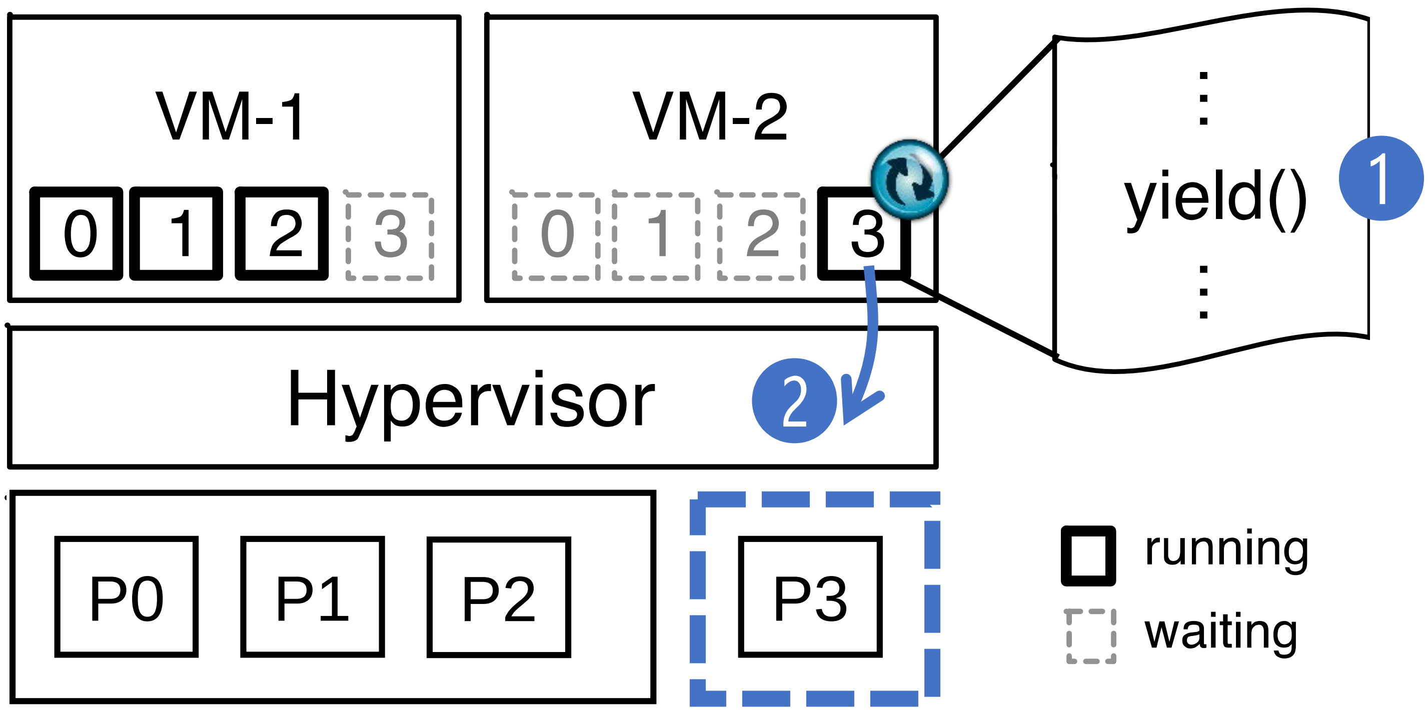
Accelerating Critical Sections



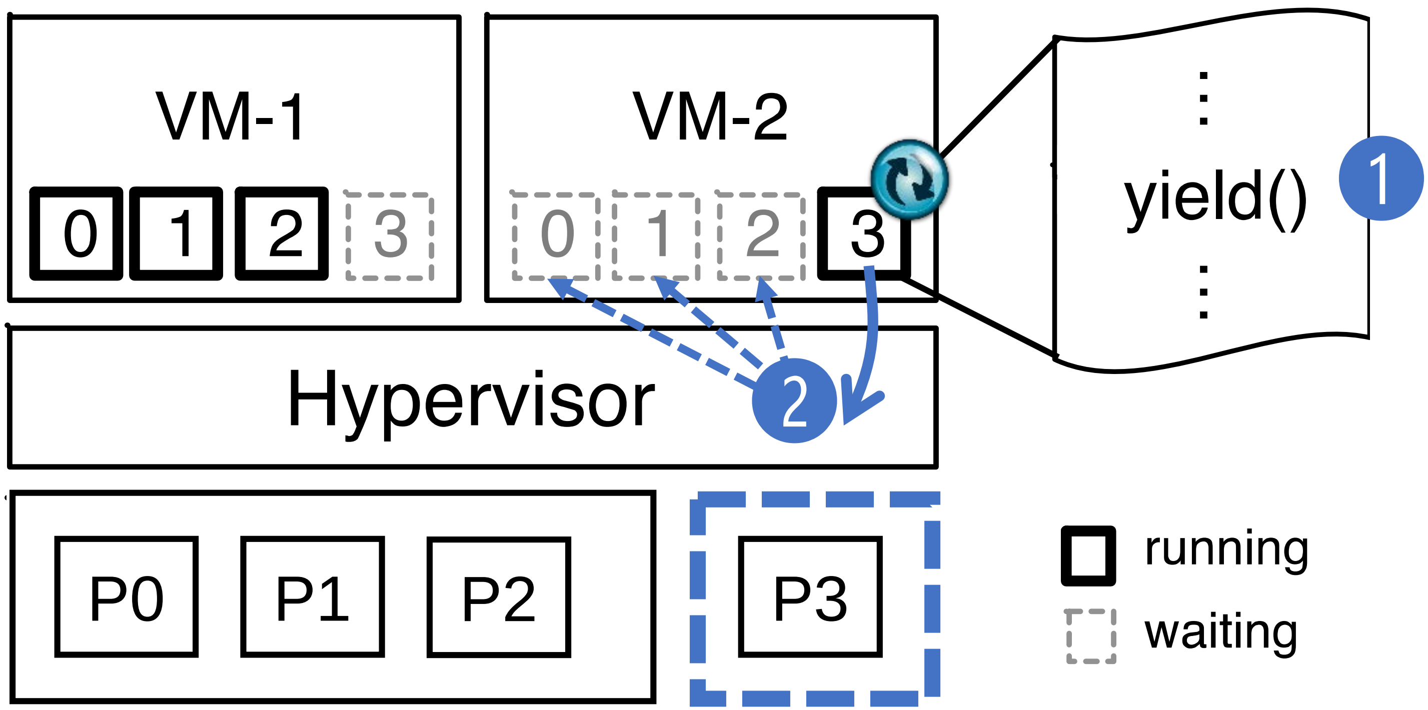
Accelerating Critical Sections



Accelerating Critical Sections

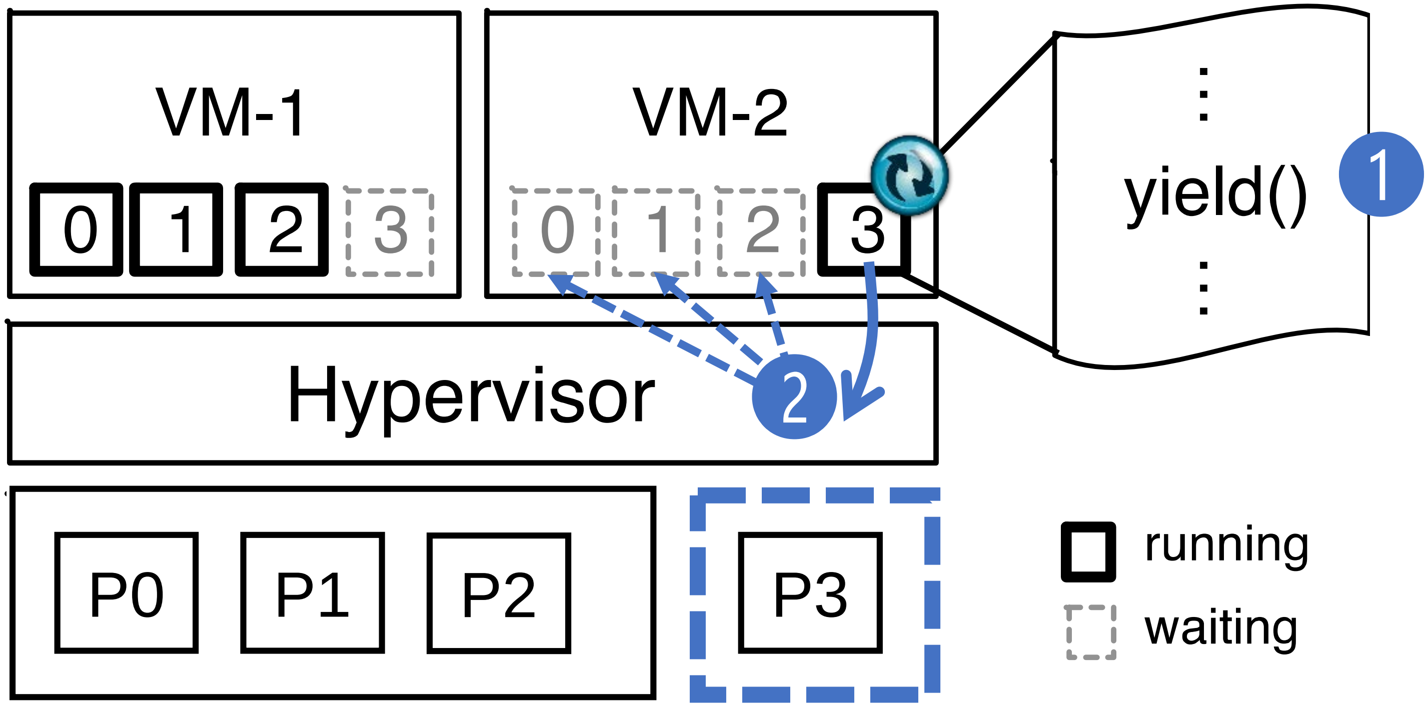


Accelerating Critical Sections



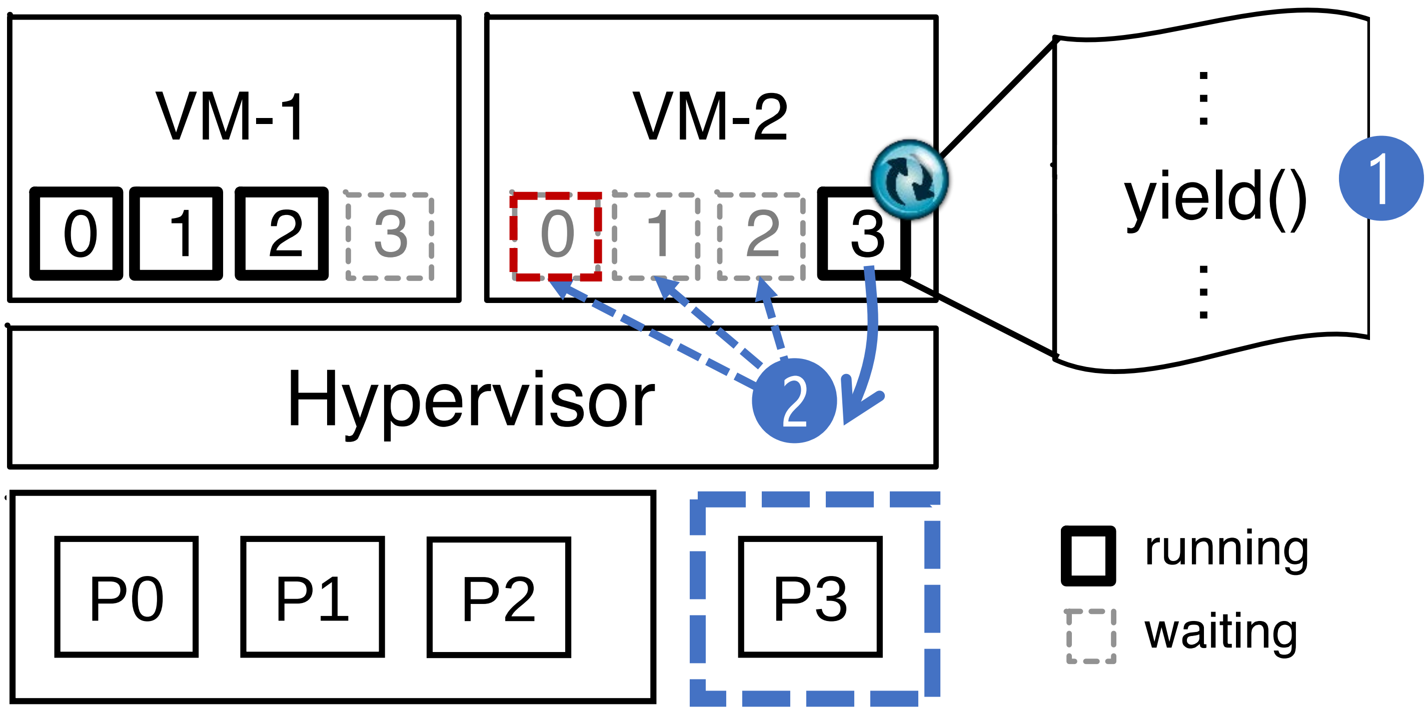
❶ Yield occurring

Accelerating Critical Sections



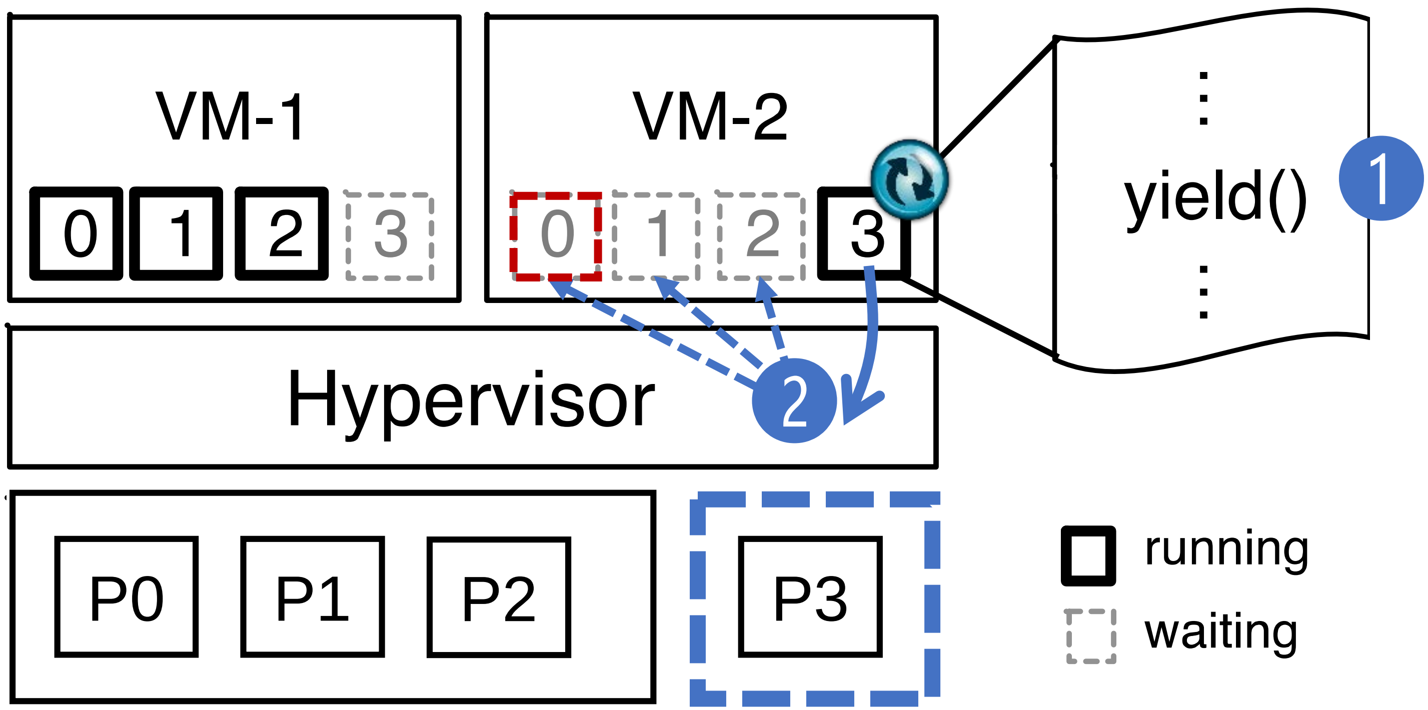
❶ Yield occurring

Accelerating Critical Sections



❶ Yield occurring
❷ Investigating the preempted vCPUs

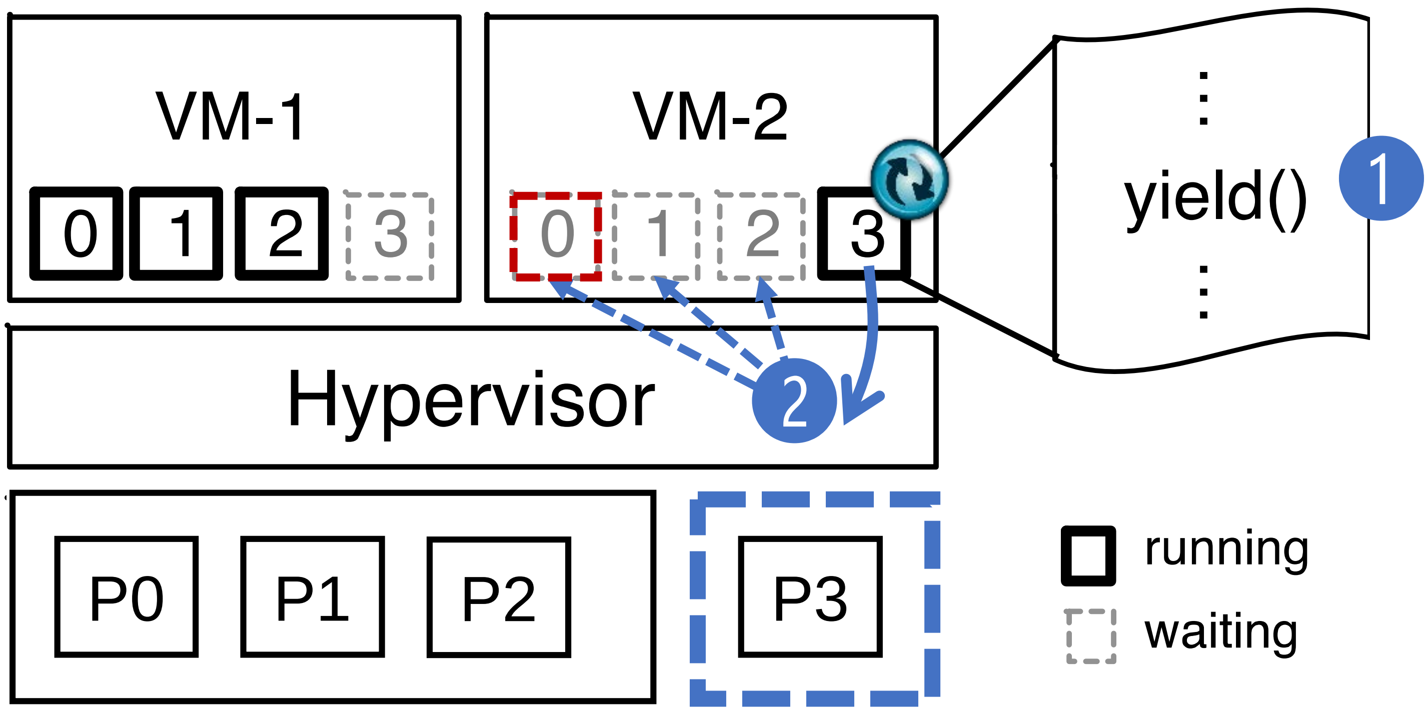
Accelerating Critical Sections



❶ Yield occurring

❷ Investigating the preempted vCPUs

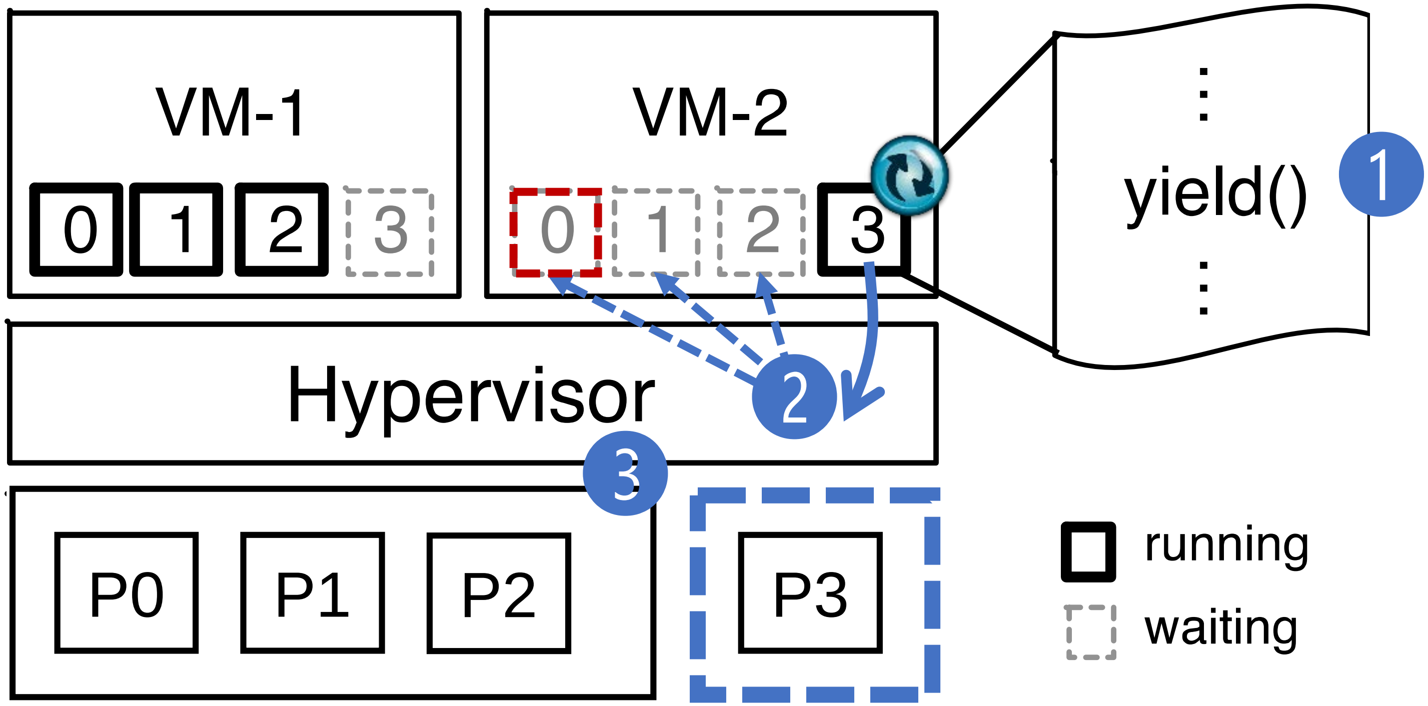
Accelerating Critical Sections



❶ Yield occurring

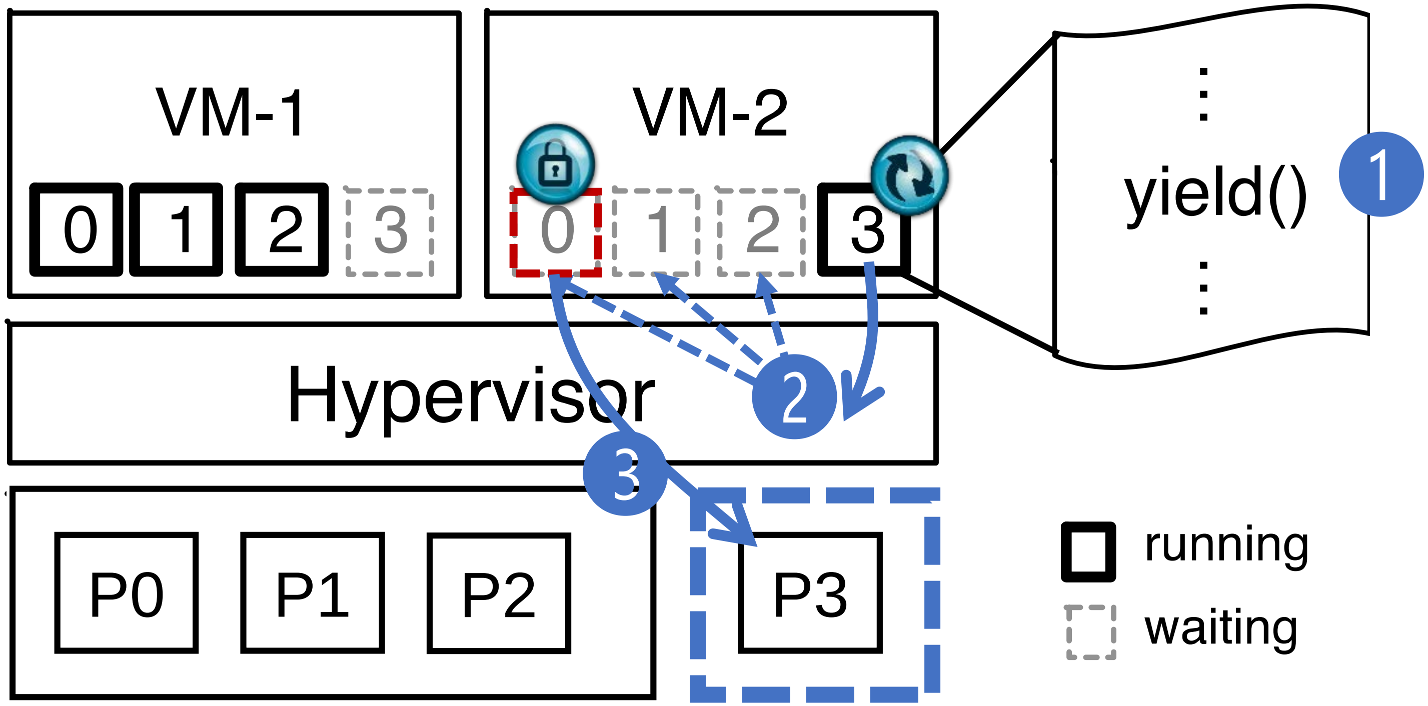
❷ Investigating the preempted vCPUs

Accelerating Critical Sections



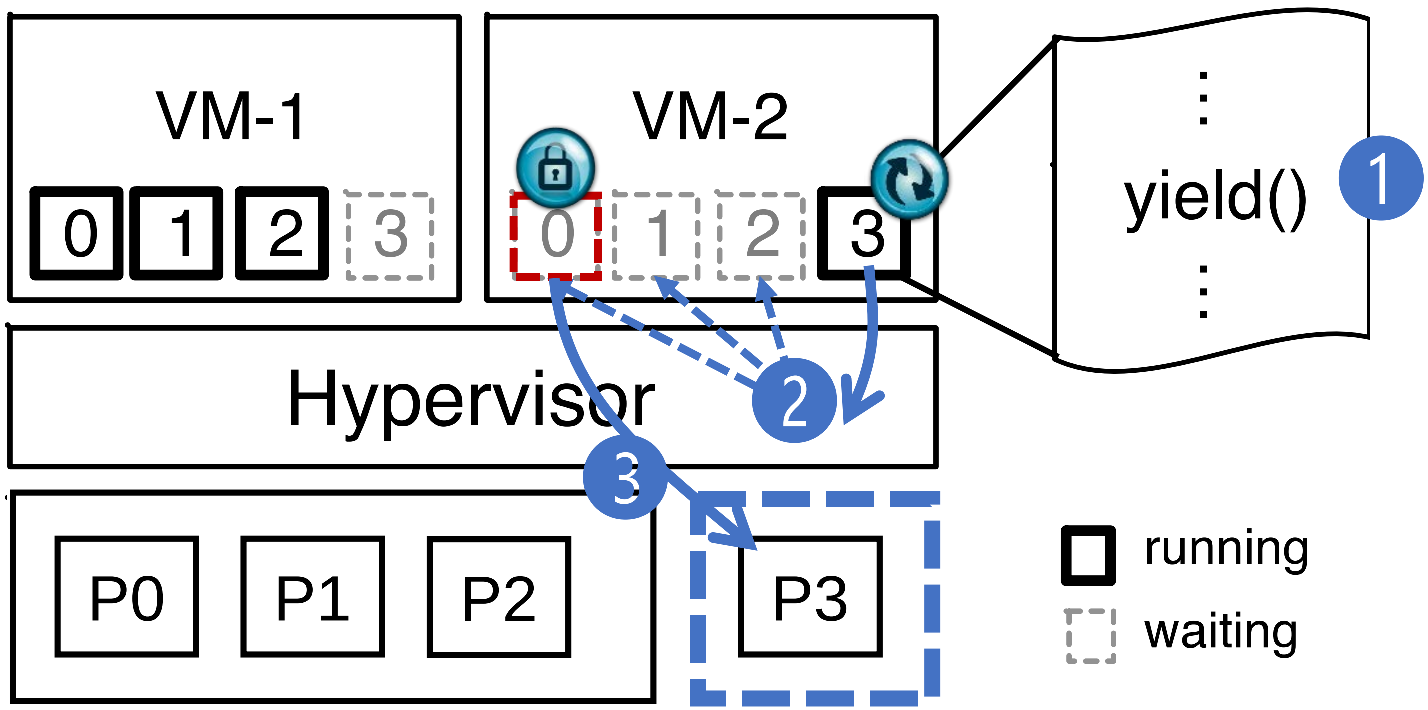
❶ Yield occurring
❷ Investigating the preempted vCPUs
❸ Scheduling the selected vCPU on the micro-sliced pool

Accelerating Critical Sections



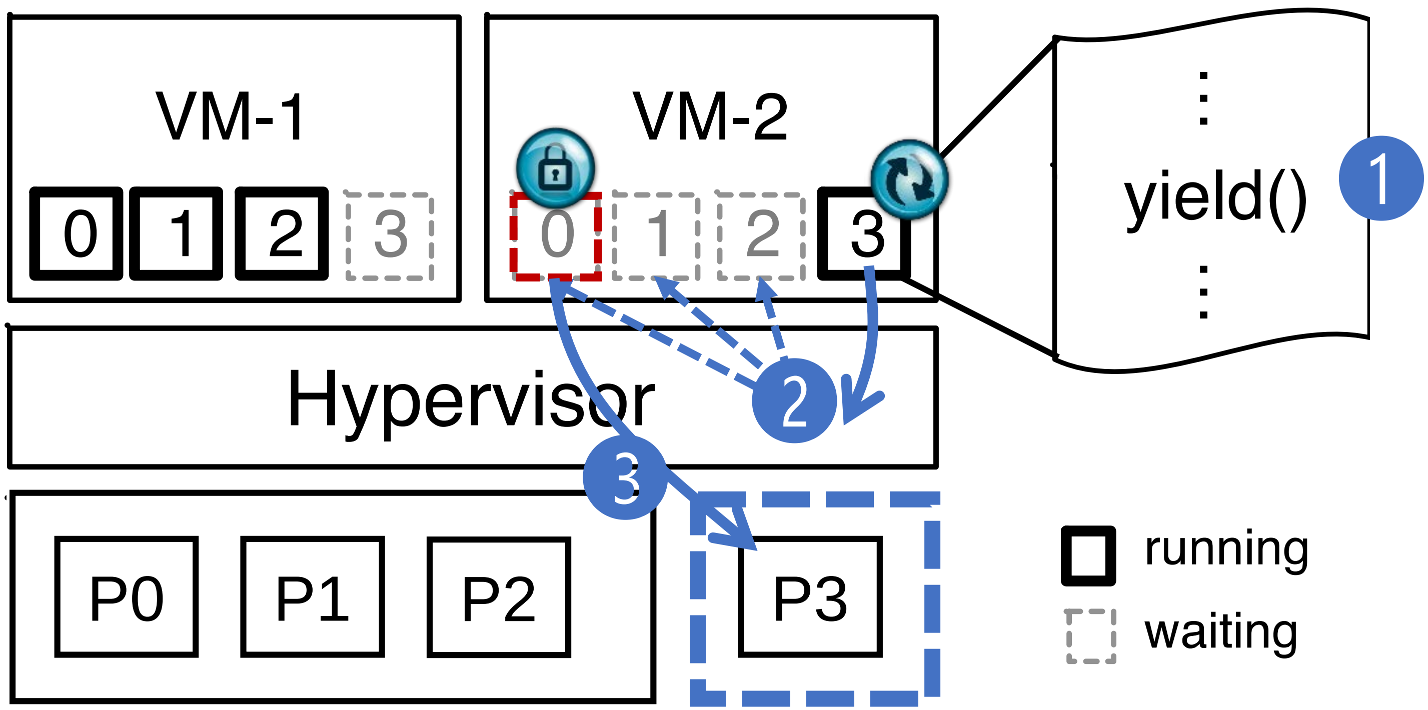
❶ Yield occurring
❷ Investigating the preempted vCPUs
❸ Scheduling the selected vCPU on the micro-sliced pool

Accelerating Critical Sections



❶ Yield occurring
❷ Investigating the preempted vCPUs
❸ Scheduling the selected vCPU on the micro-sliced pool

Accelerating Critical Sections

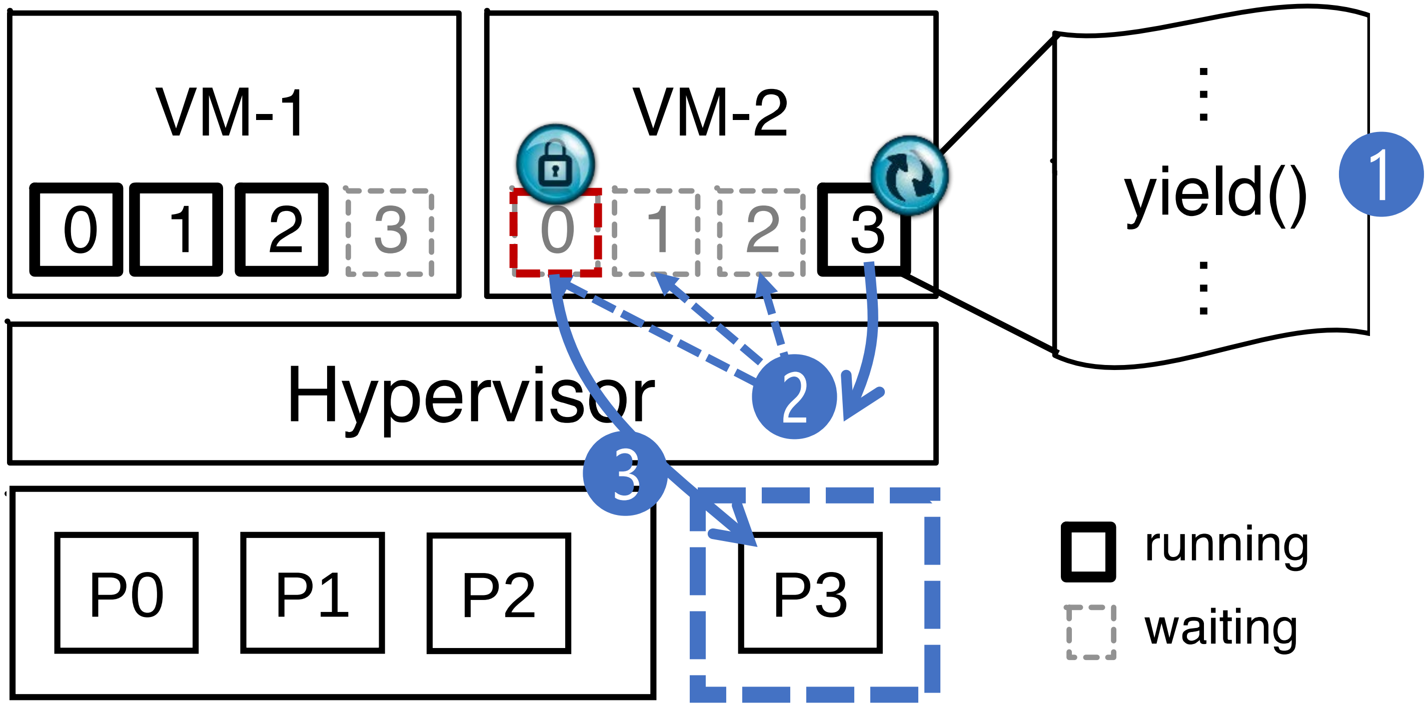


❶ Yield occurring

❷ Investigating the preempted vCPUs

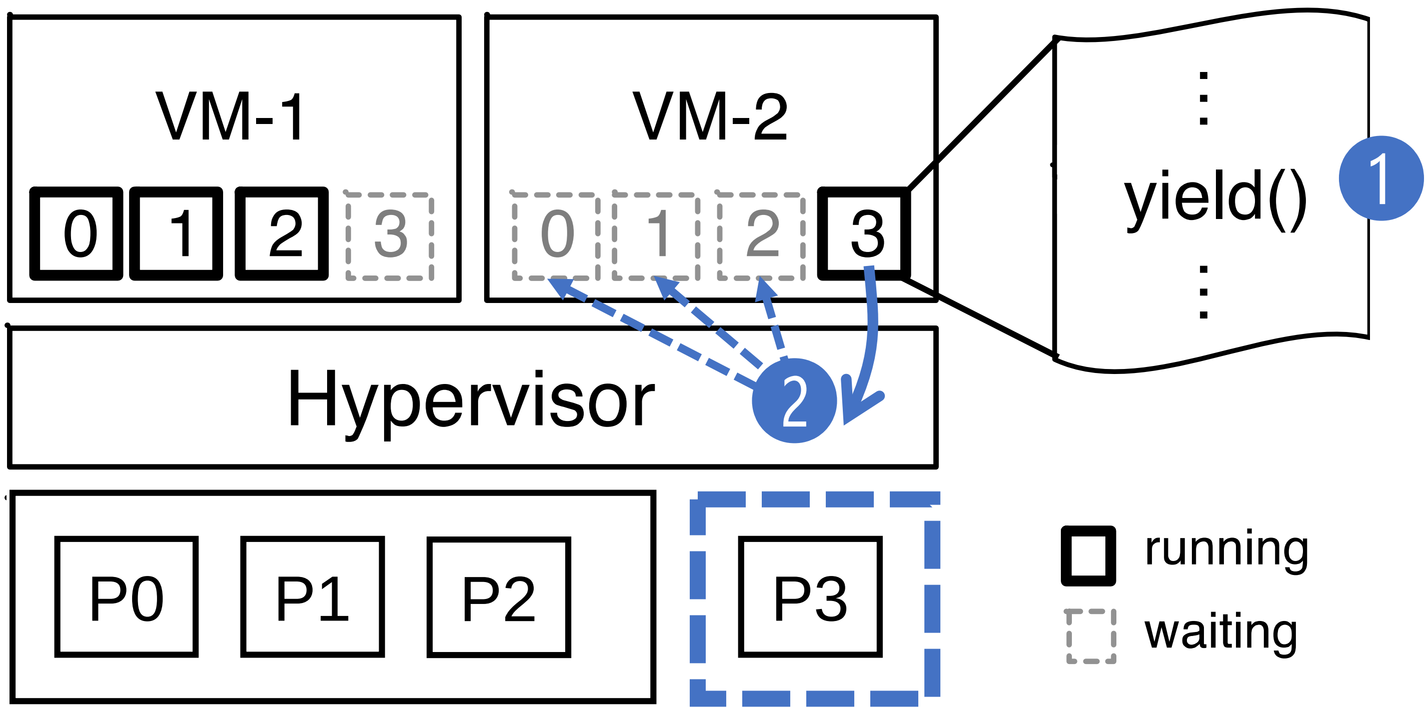
❸ Scheduling the selected vCPU on the micro-sliced pool

Accelerating Critical Sections



❶ Yield occurring
❷ Investigating the preempted vCPUs
❸ Scheduling the selected vCPU on the micro-sliced pool

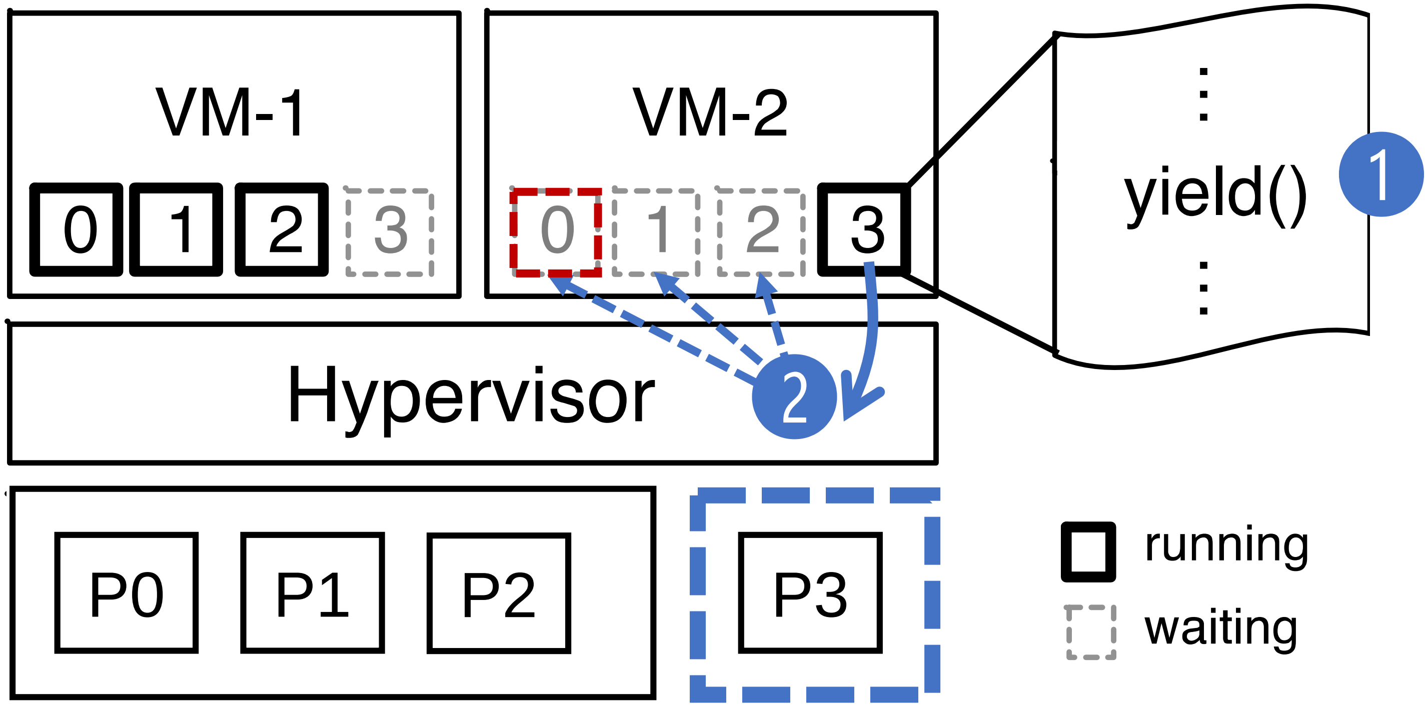
Accelerating Critical TLB Synchronizations



❶ Yield occurring

❷ Investigating the preempted vCPUs

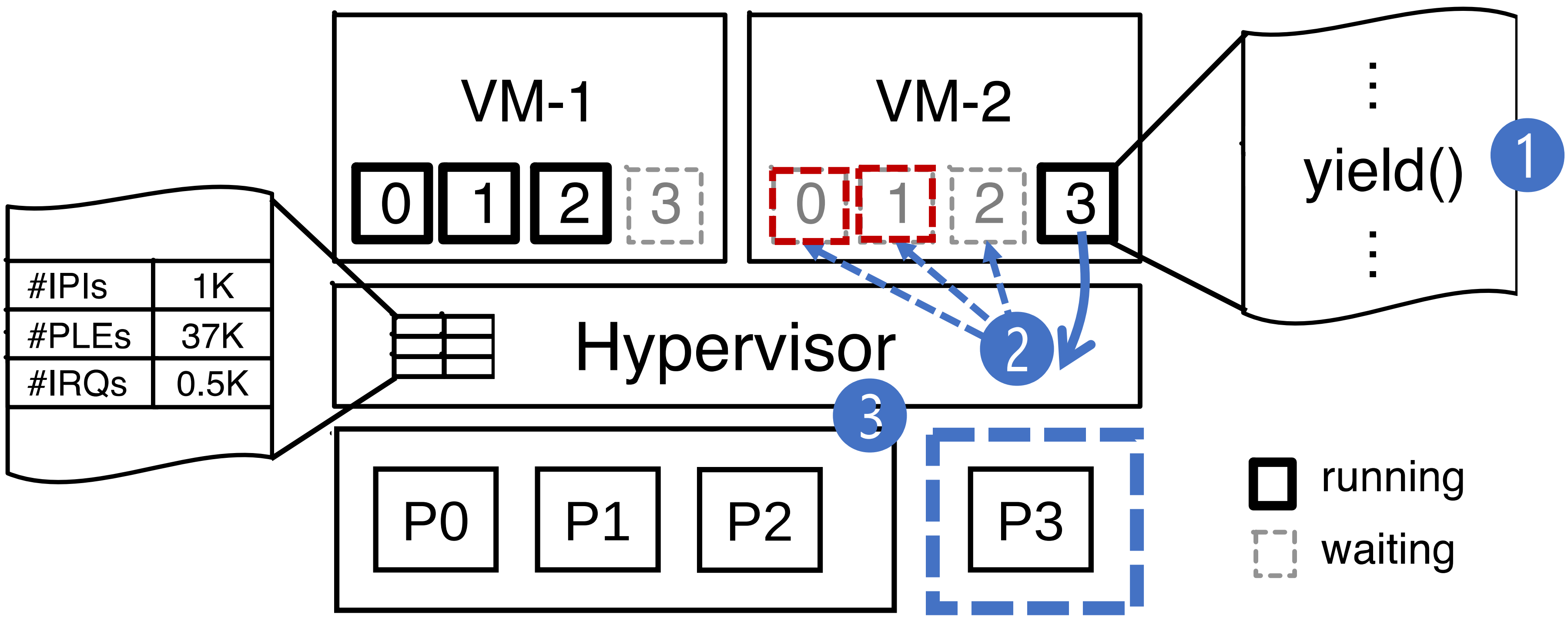
Accelerating Critical TLB Synchronizations



❶ Yield occurring

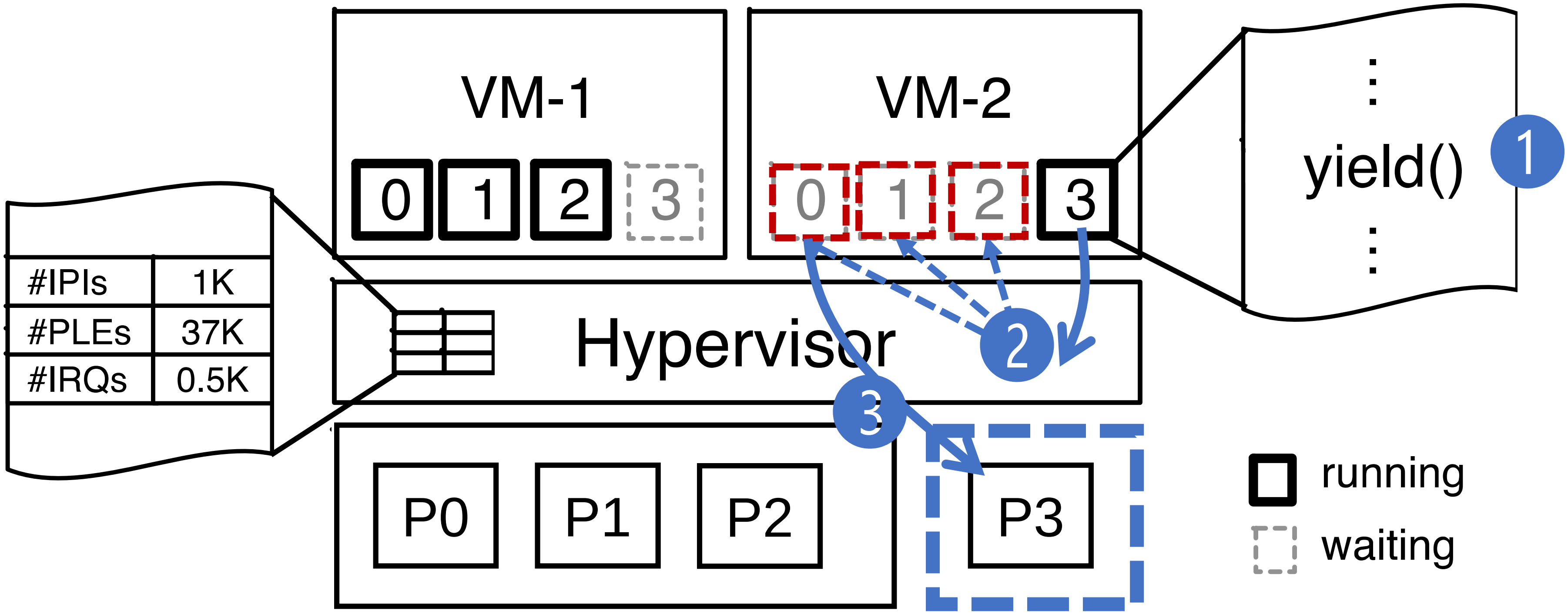
❷ Investigating the preempted vCPUs

Accelerating Critical TLB Synchronizations



1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool

Accelerating Critical TLB Synchronizations

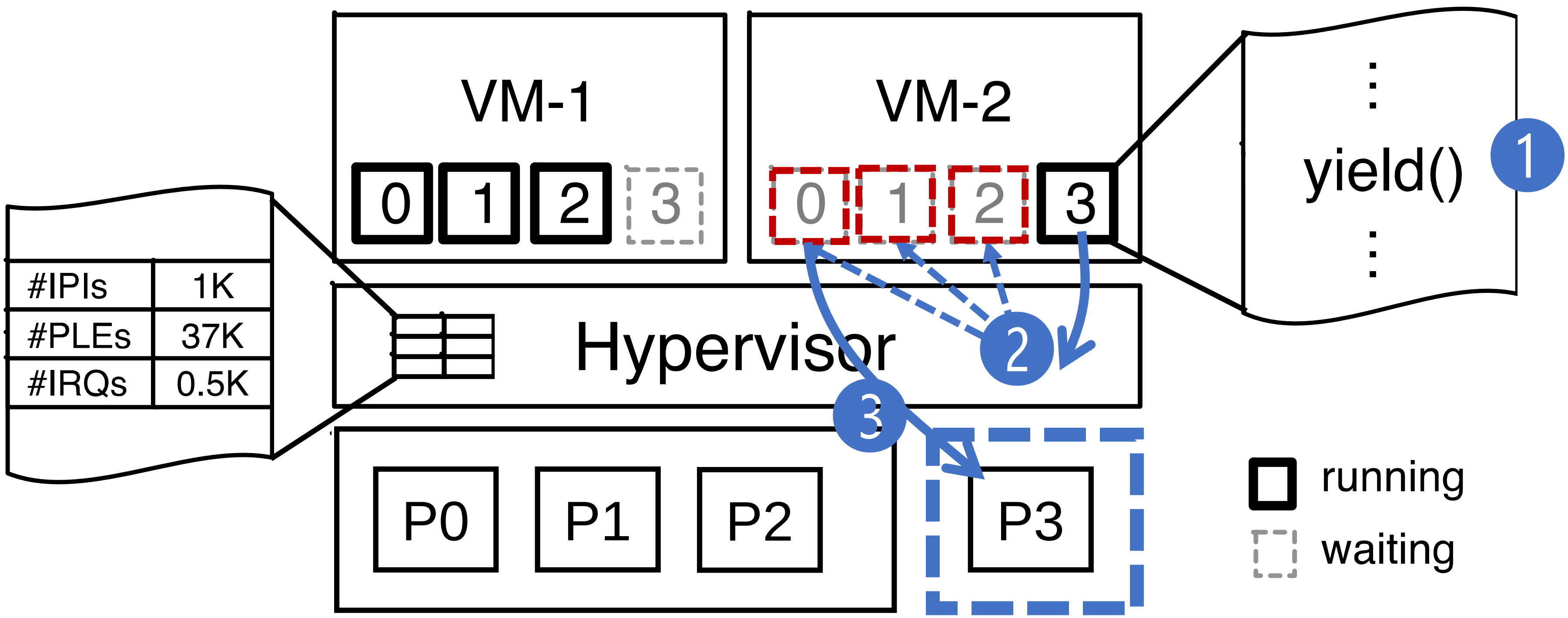


❶ Yield occurring

❷ Investigating the preempted vCPUs

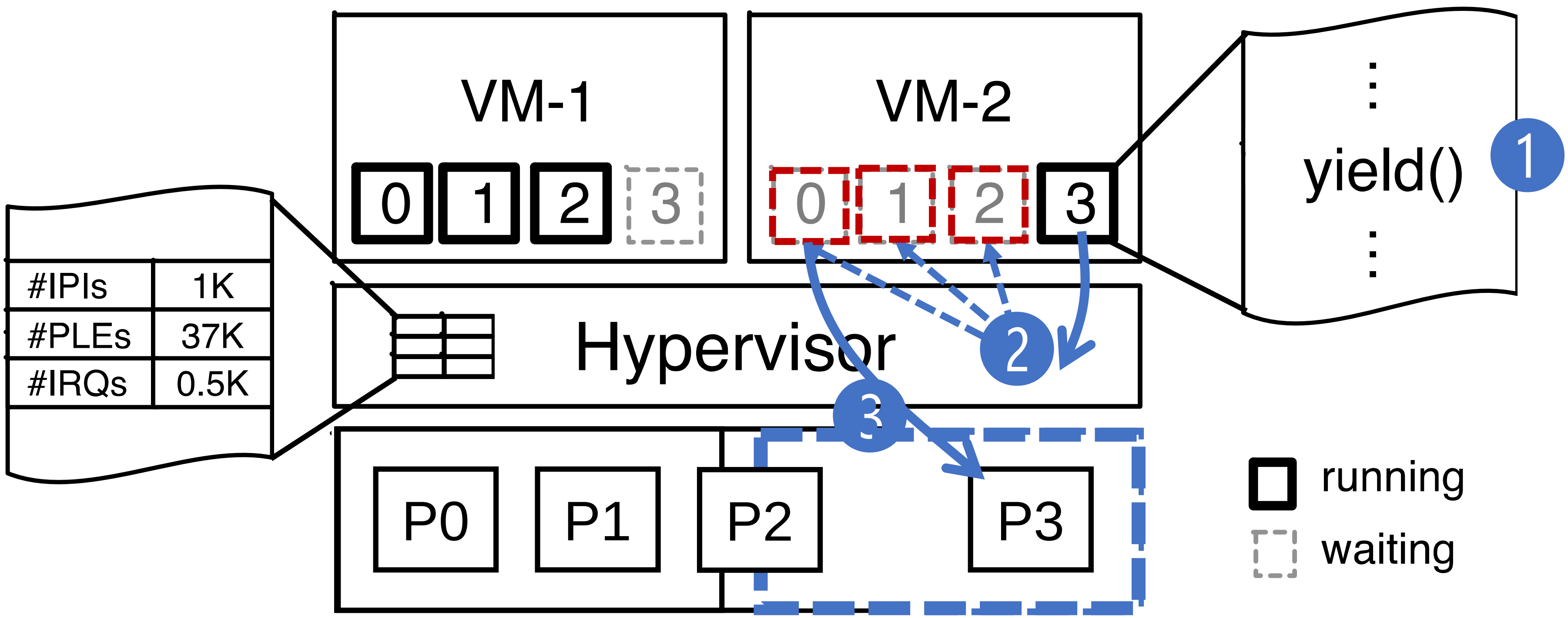
❸ Scheduling the selected vCPU on the micro-sliced pool

Accelerating Critical TLB Synchronizations



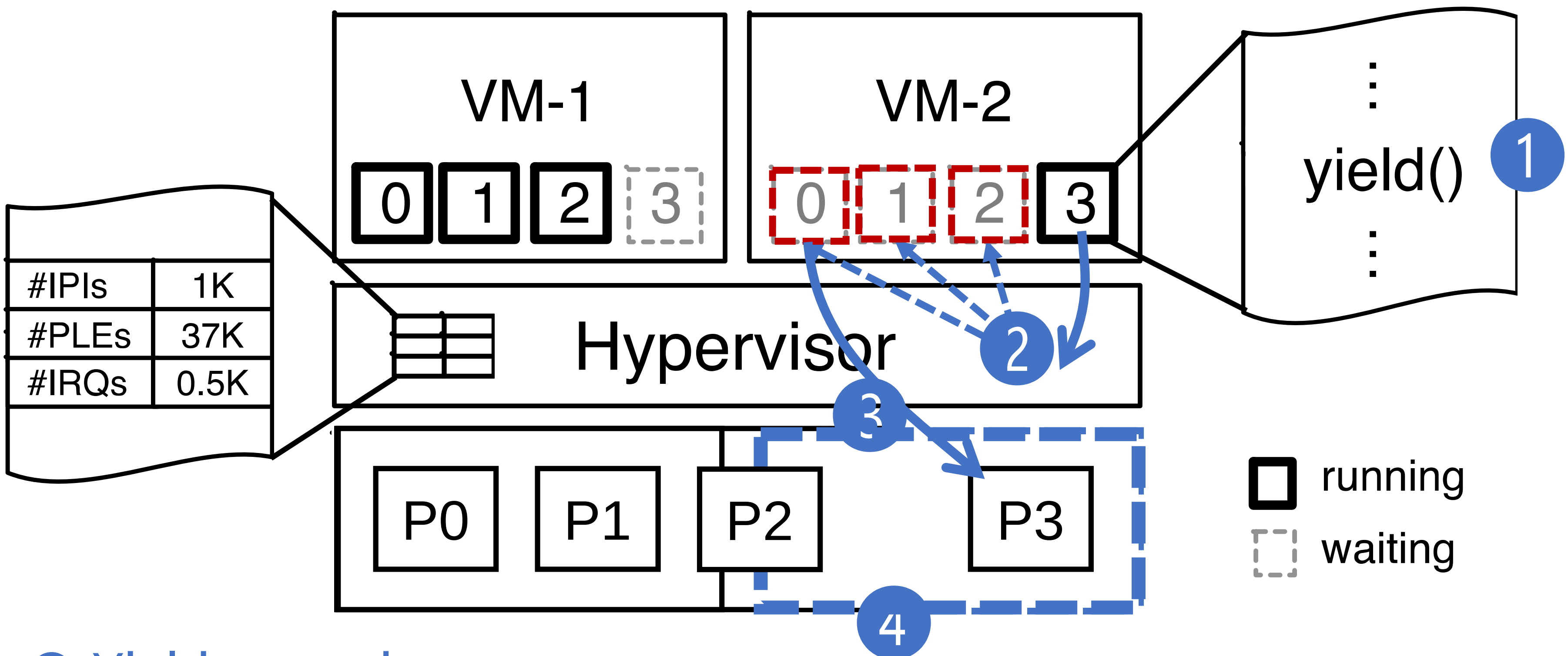
❶ Yield occurring
❷ Investigating the preempted vCPUs
❸ Scheduling the selected vCPU on the micro-sliced pool
❹ Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



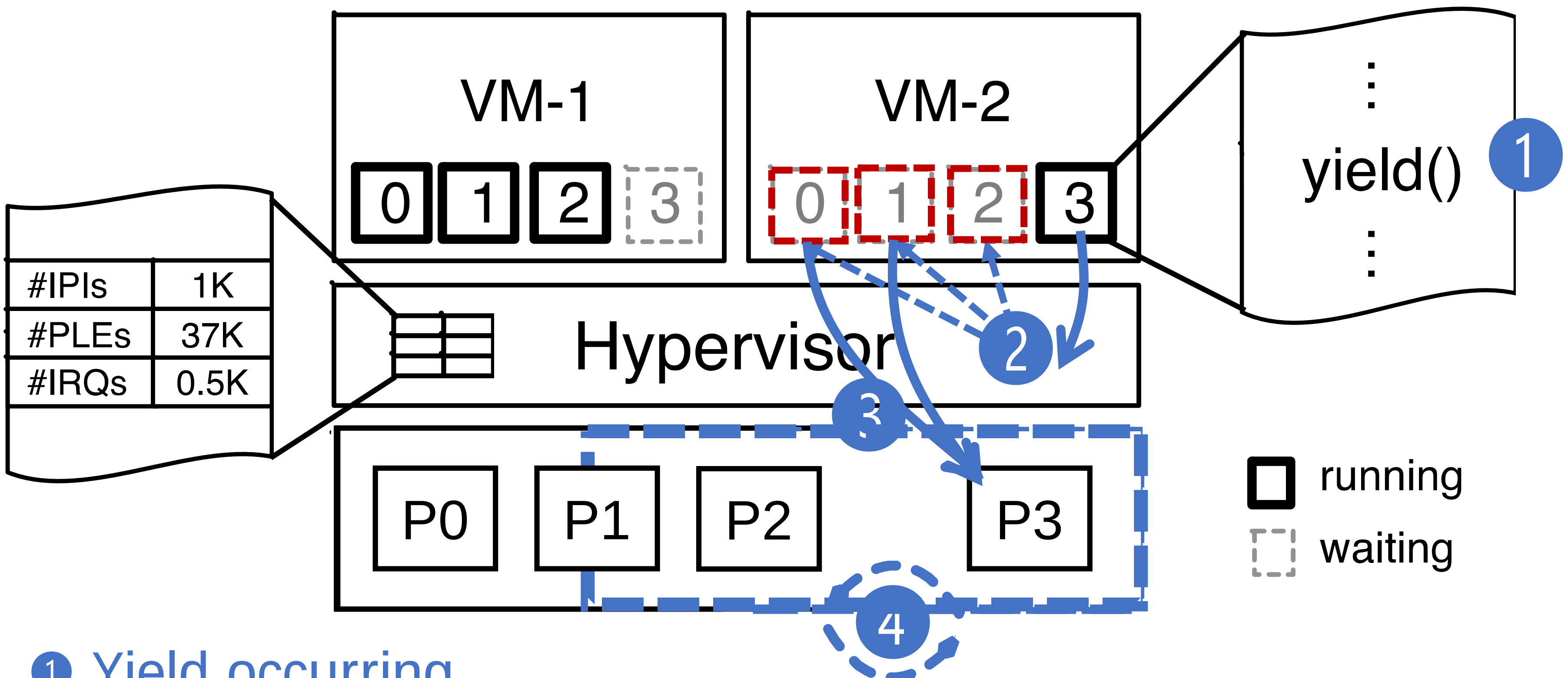
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



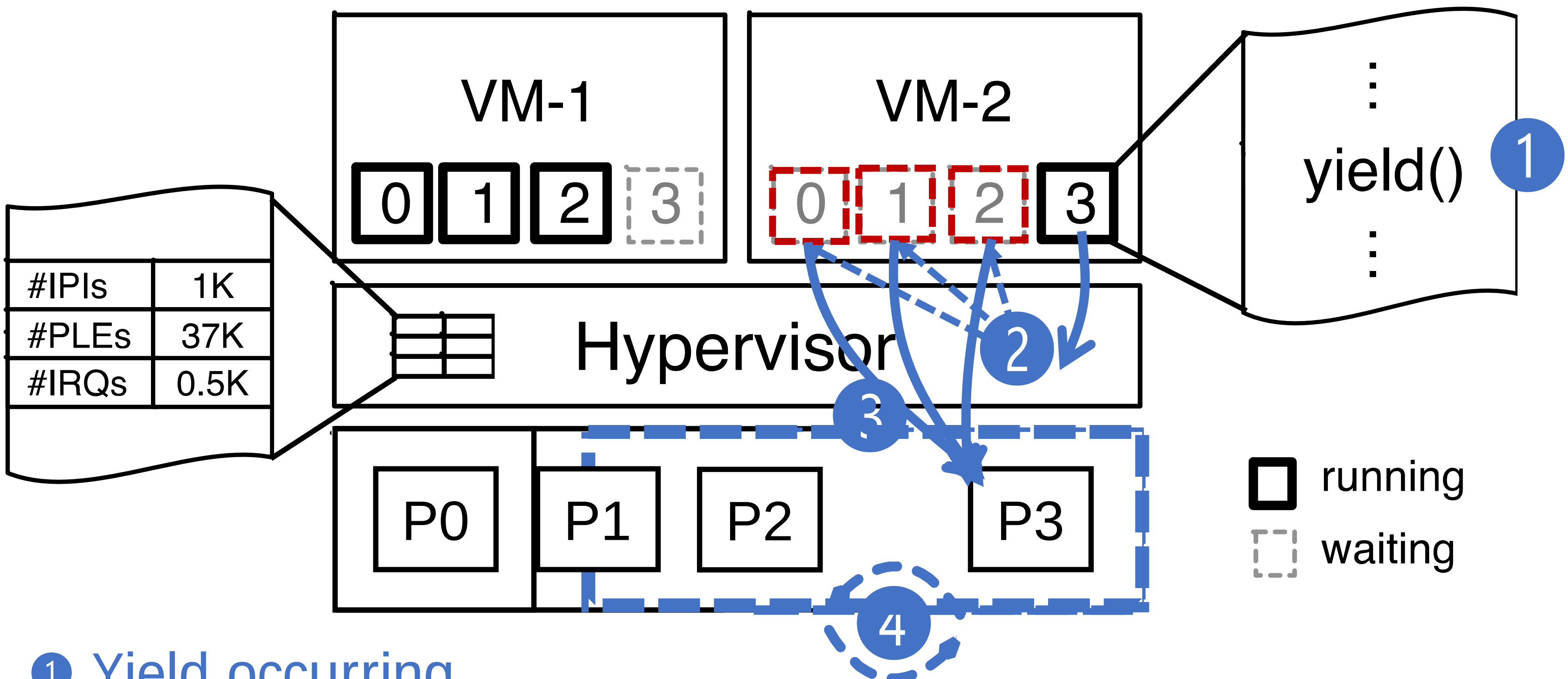
❶ Yield occurring
❷ Investigating the preempted vCPUs
❸ Scheduling the selected vCPU on the micro-sliced pool
❹ Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



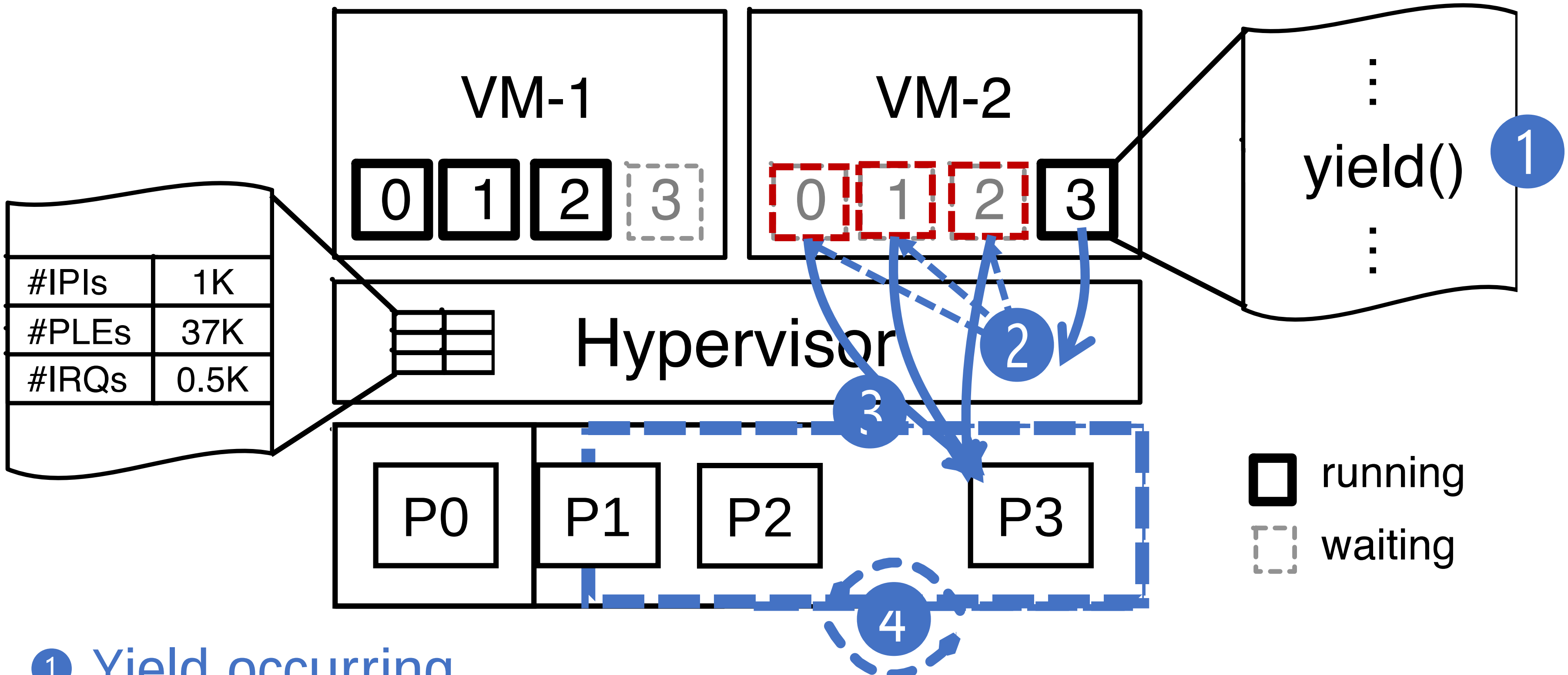
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



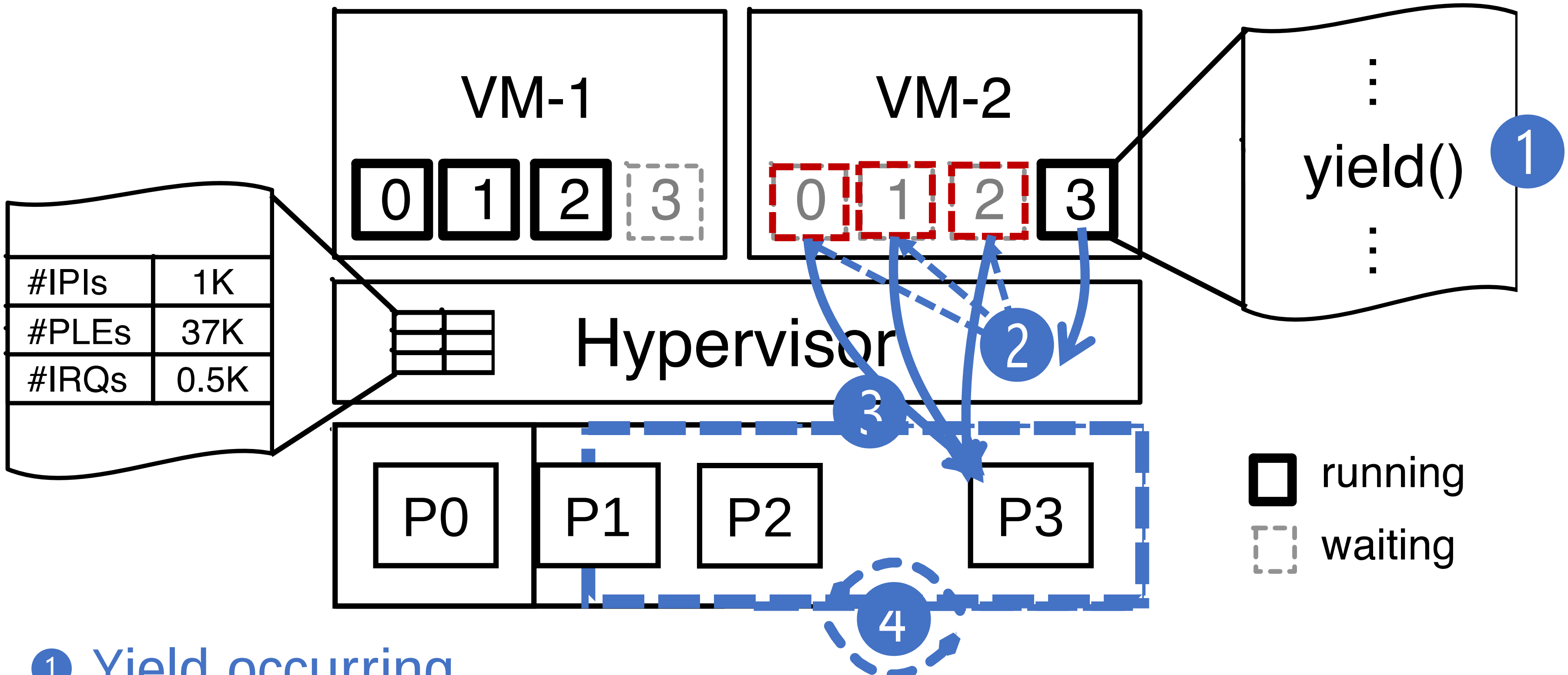
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



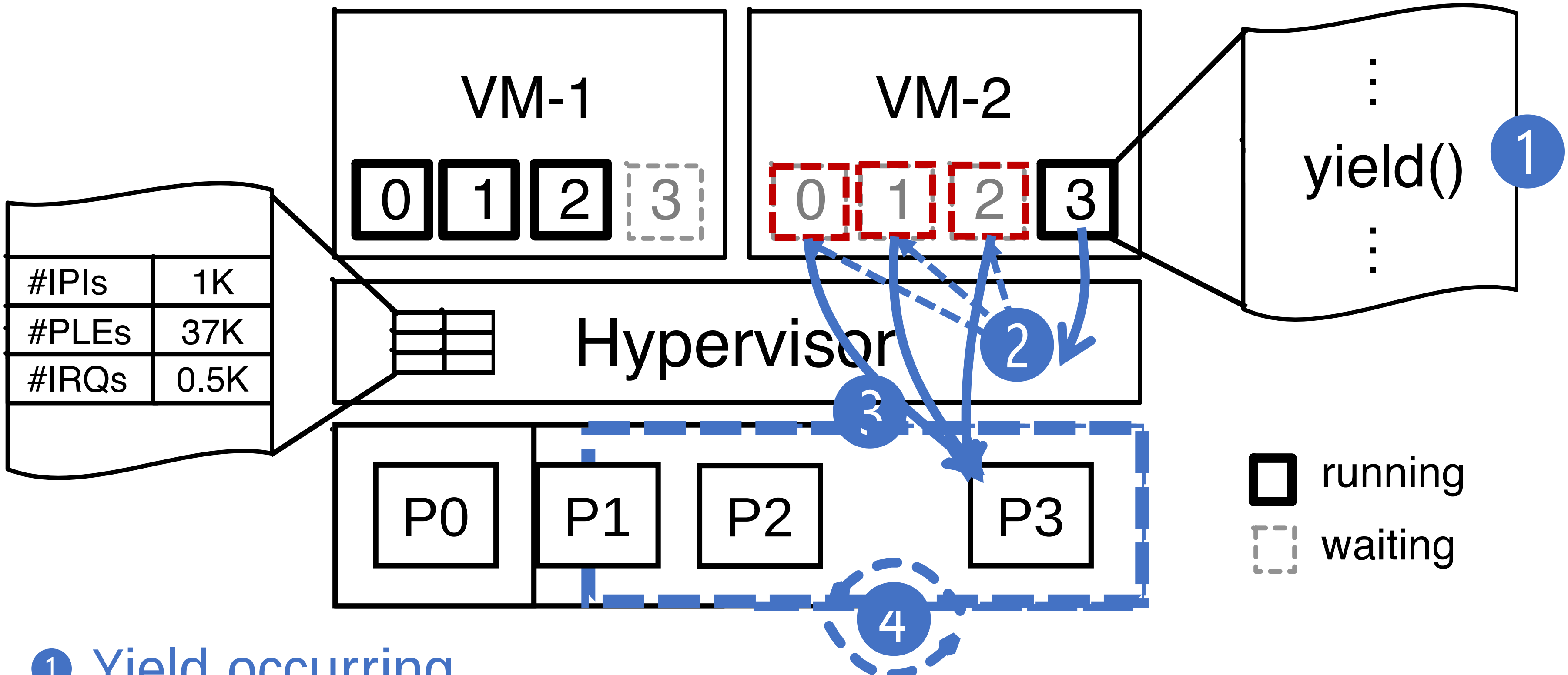
❶ Yield occurring

❷ Investigating the preempted vCPUs

❸ Scheduling the selected vCPU on the micro-sliced pool

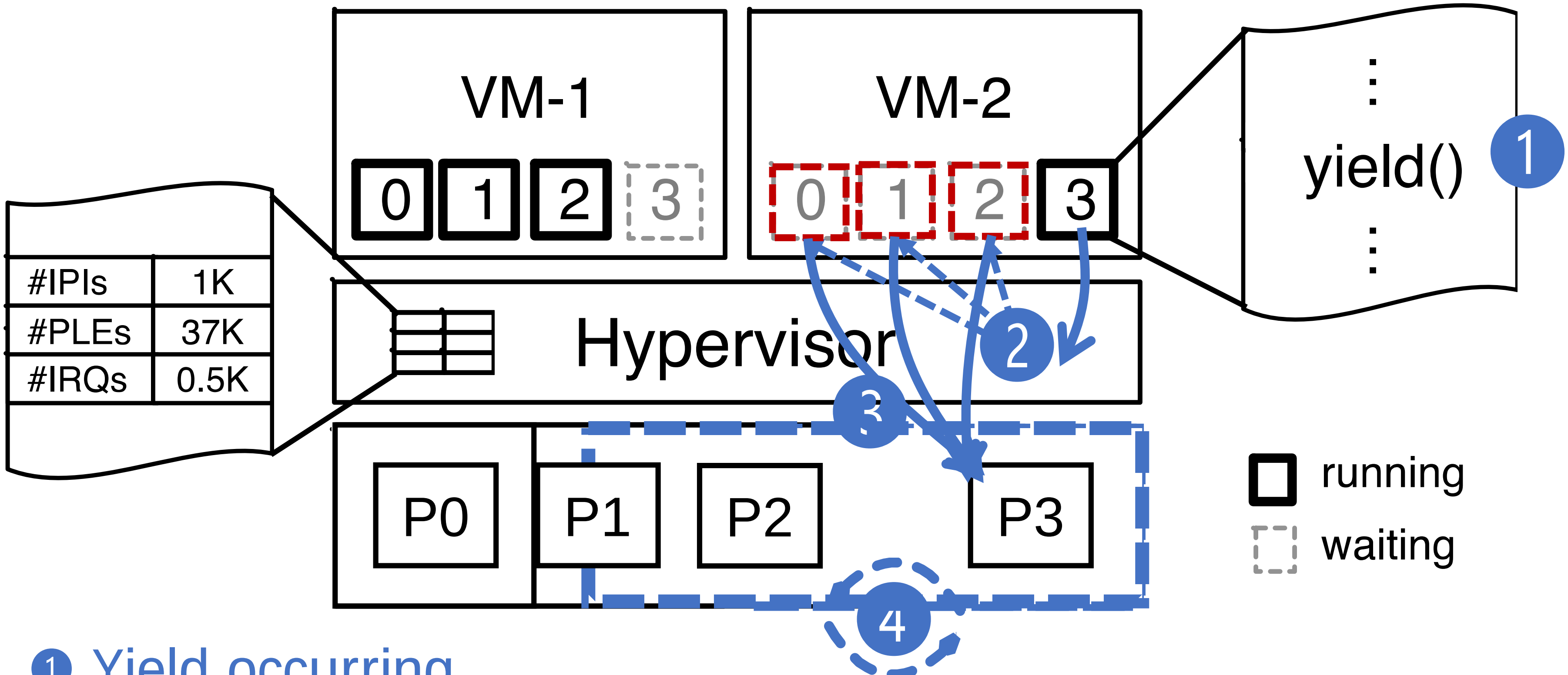
❹ Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



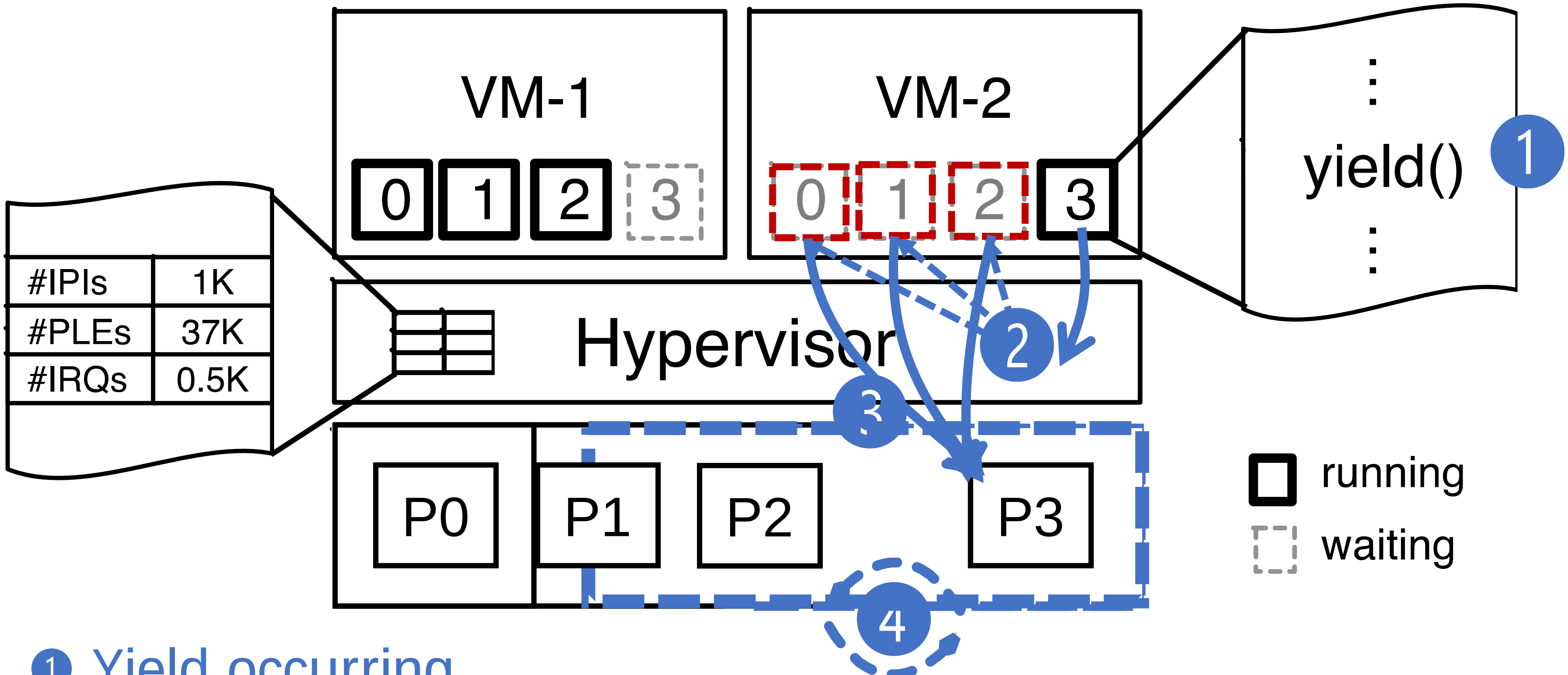
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



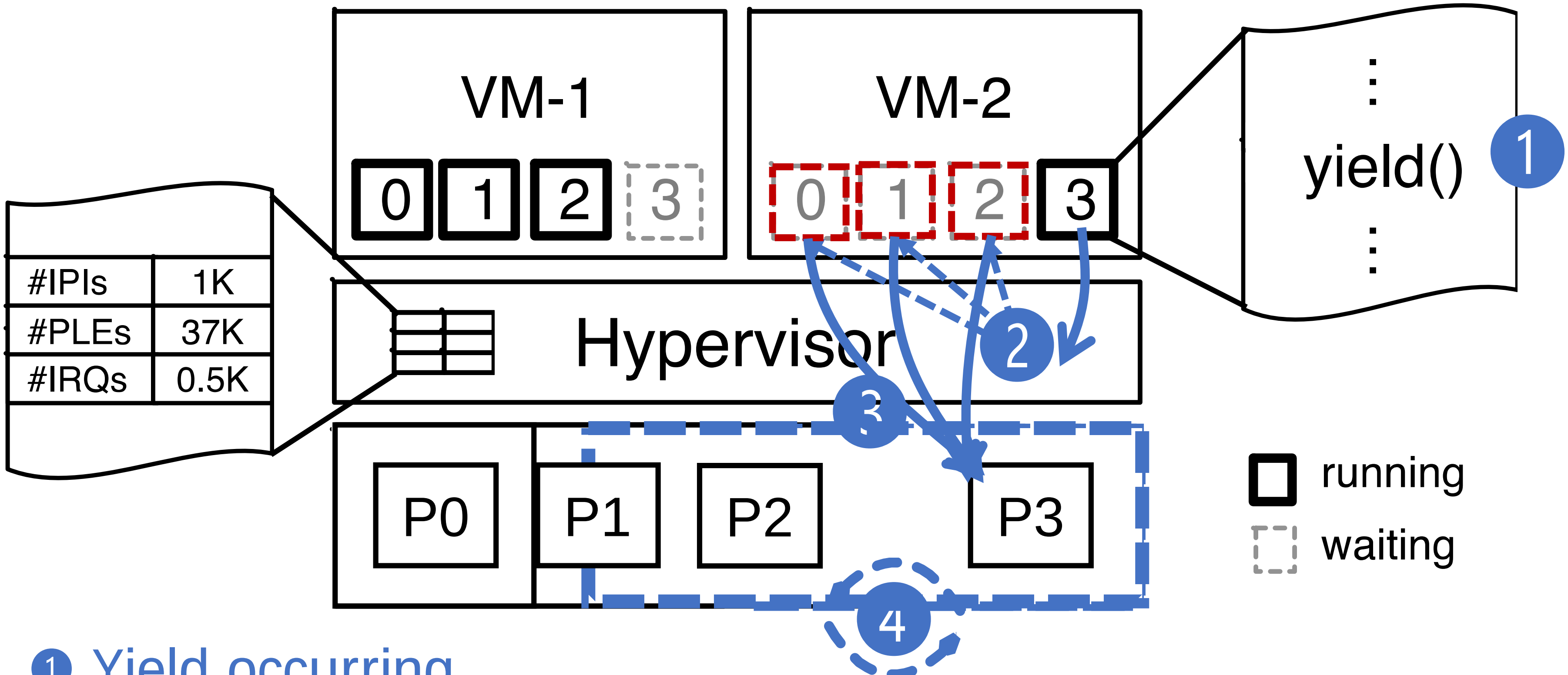
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



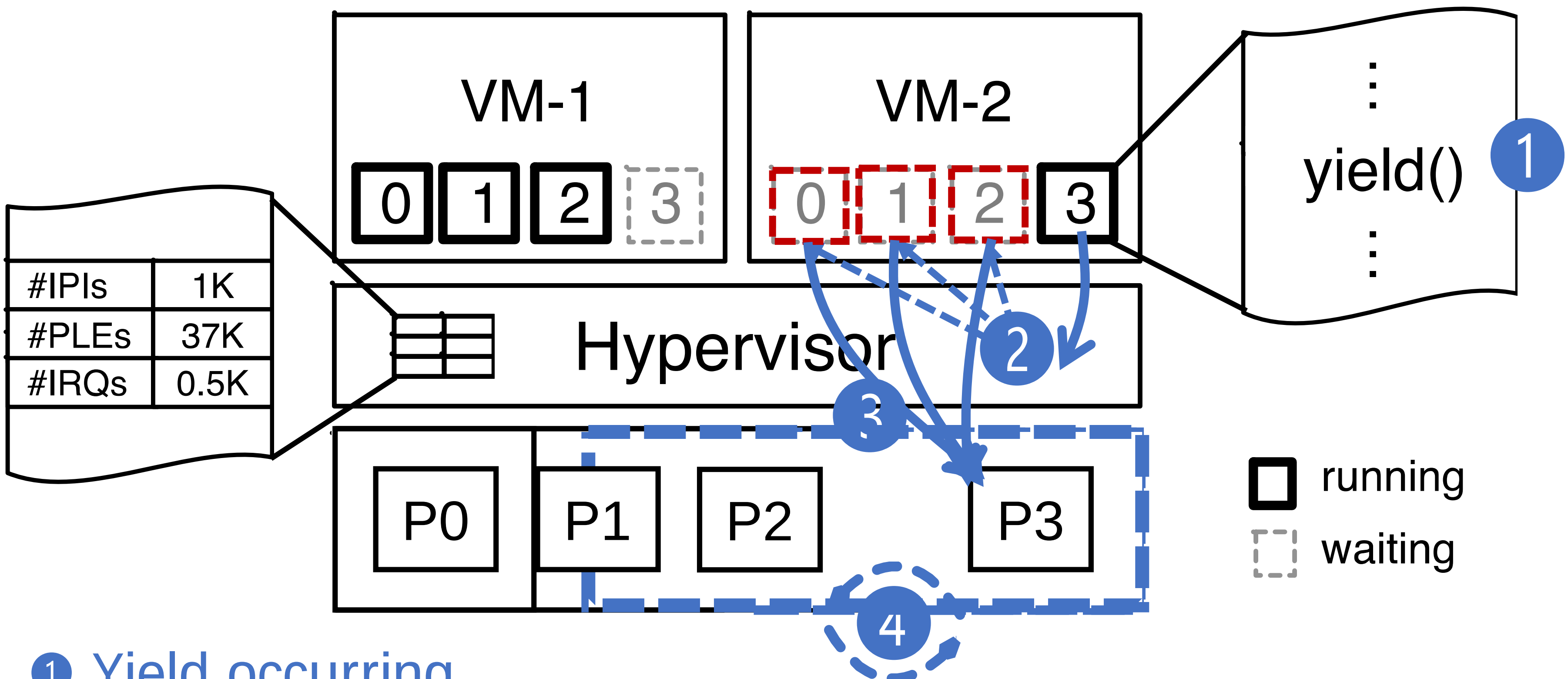
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



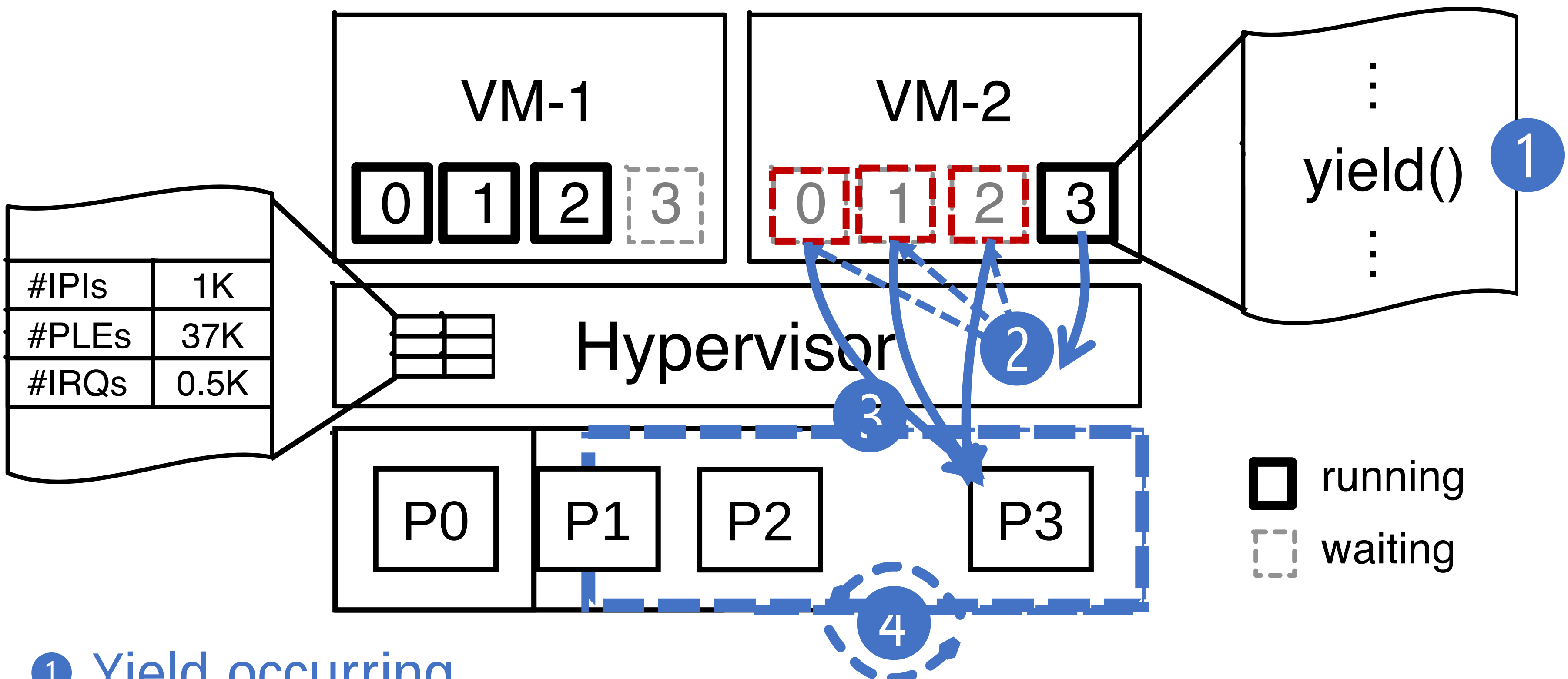
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



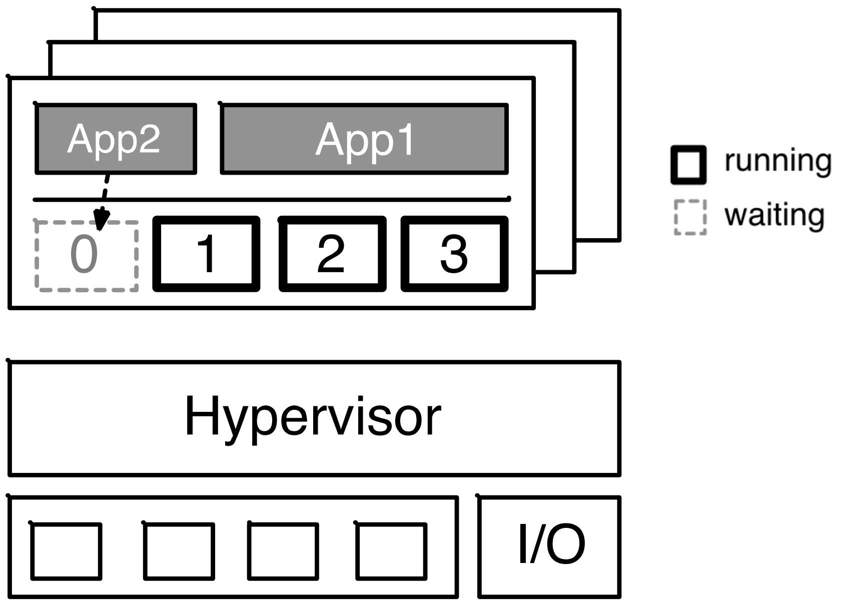
1. Yield occurring
2. Investigating the preempted vCPUs
3. Scheduling the selected vCPU on the micro-sliced pool
4. Dynamically adjusting micro-sliced cores based on profiling

Accelerating Critical TLB Synchronizations



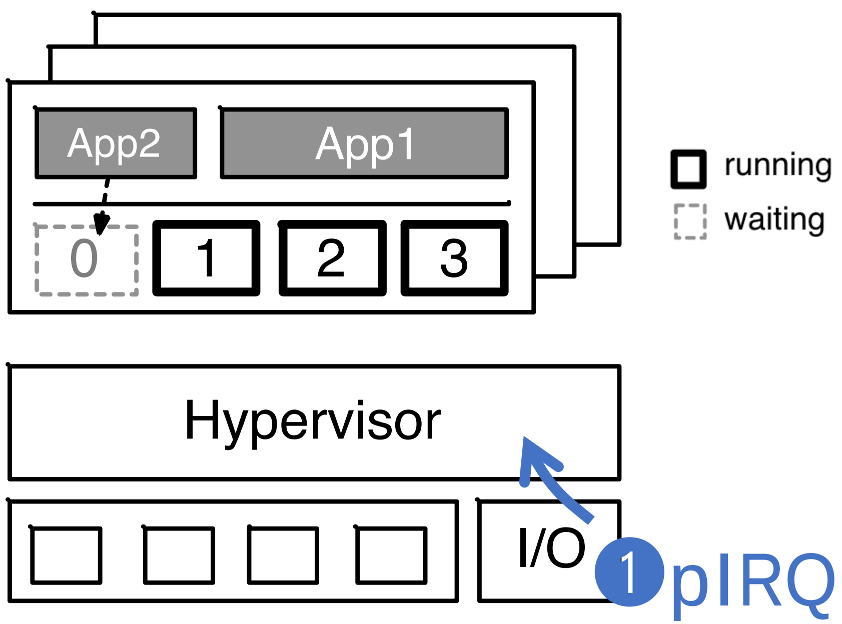
❶ Yield occurring
❷ Investigating the preempted vCPUs
❸ Scheduling the selected vCPU on the micro-sliced pool
❹ Dynamically adjusting micro-sliced cores based on profiling

Detecting Critical I/O Events



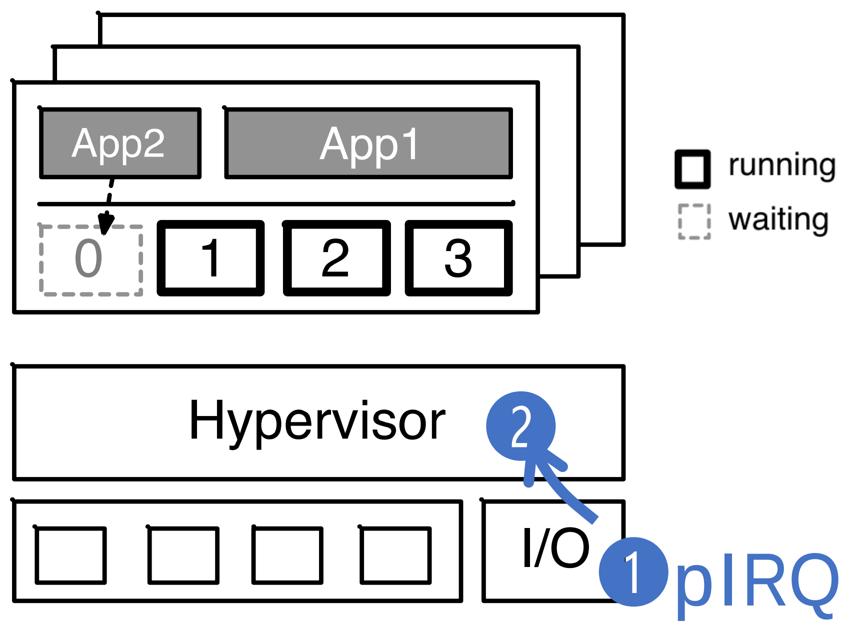
I/O handling consists of a chain of operations involving potentially multiple vCPUs

Detecting Critical I/O Events



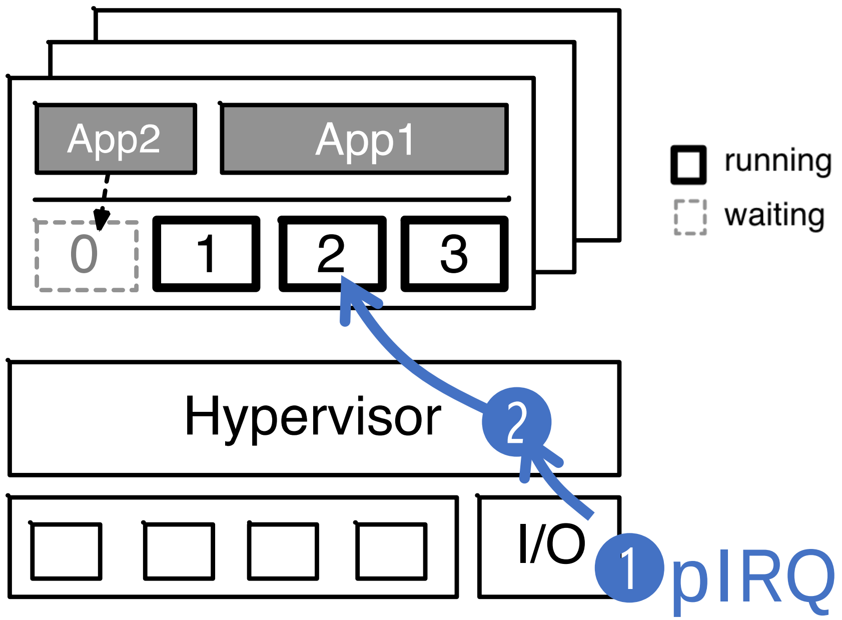
I/O handling consists of a chain of operations involving potentially multiple vCPUs

Detecting Critical I/O Events



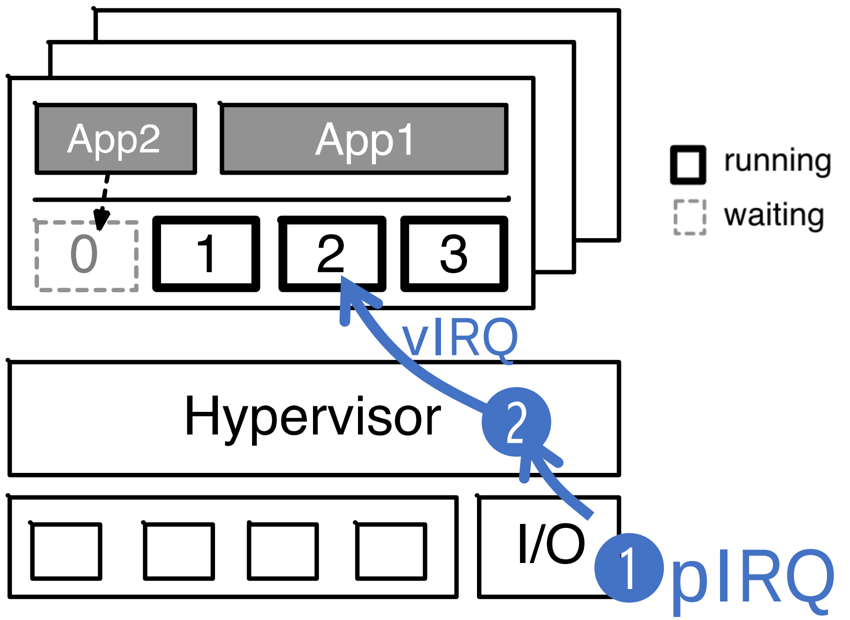
I/O handling consists of a chain of operations involving potentially multiple vCPUs

Detecting Critical I/O Events



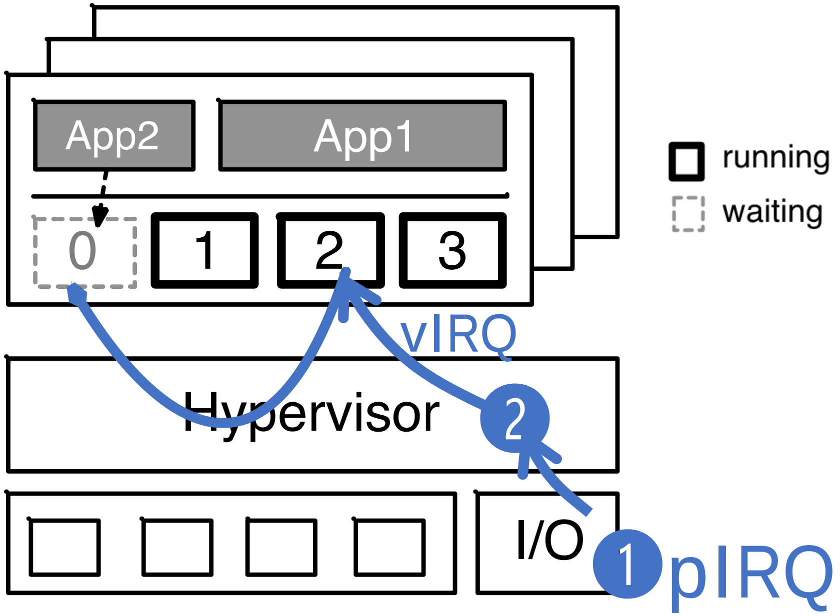
I/O handling consists of a chain of operations involving potentially multiple vCPUs

Detecting Critical I/O Events



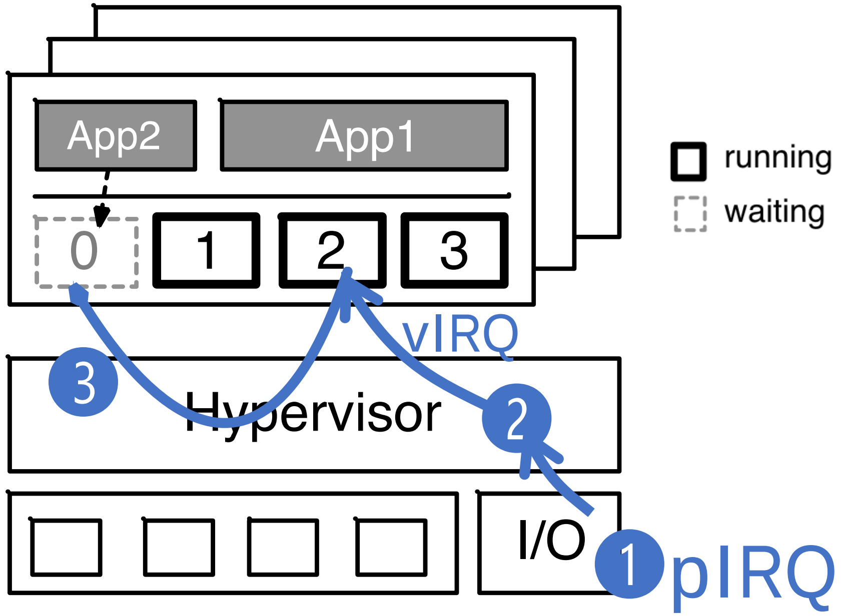
I/O handling consists of a chain of operations involving potentially multiple vCPUs

Detecting Critical I/O Events



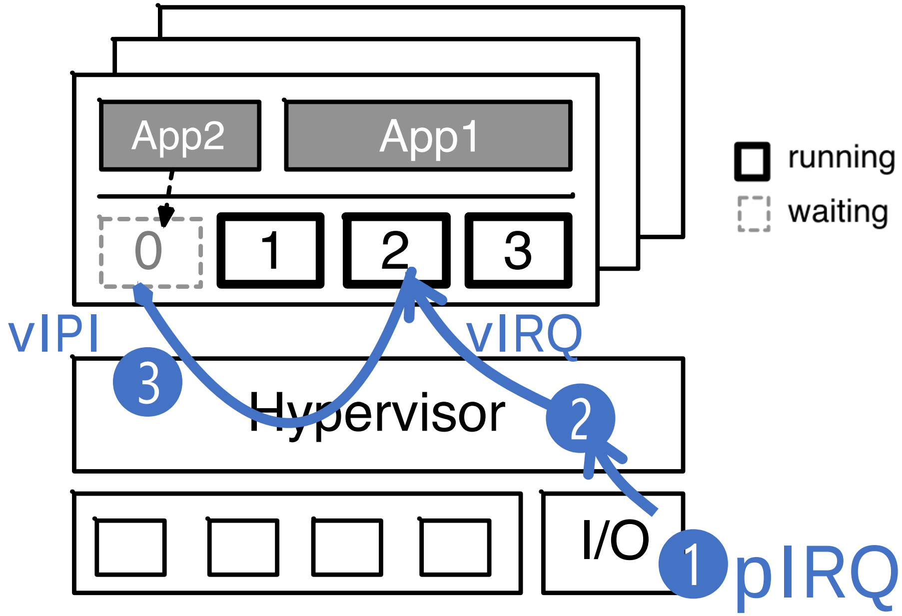
I/O handling consists of a chain of operations involving potentially multiple vCPUs

Detecting Critical I/O Events



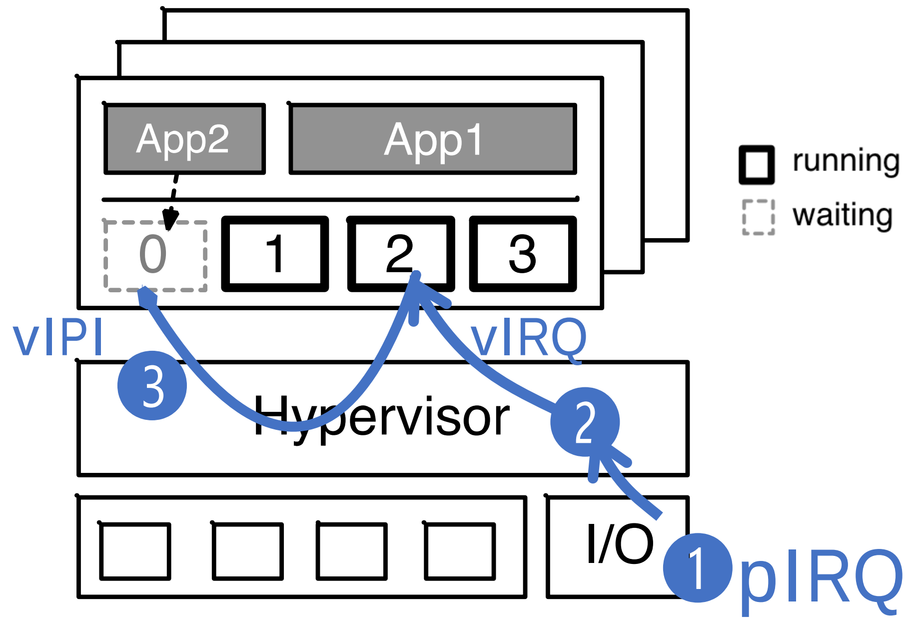
I/O handling consists of a chain of operations involving potentially multiple vCPUs

Detecting Critical I/O Events



I/O handling consists of a chain of operations involving potentially multiple vCPUs

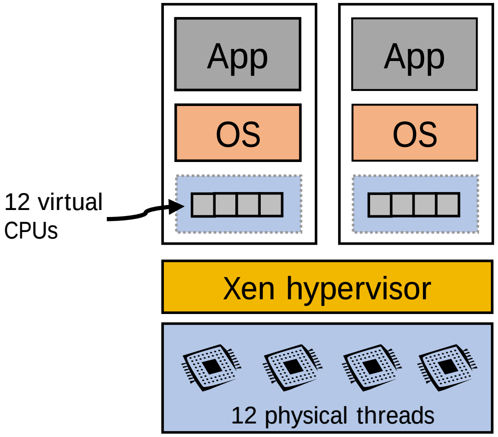
Detecting Critical I/O Events



I/O handling consists of a chain of operations involving potentially multiple vCPUs

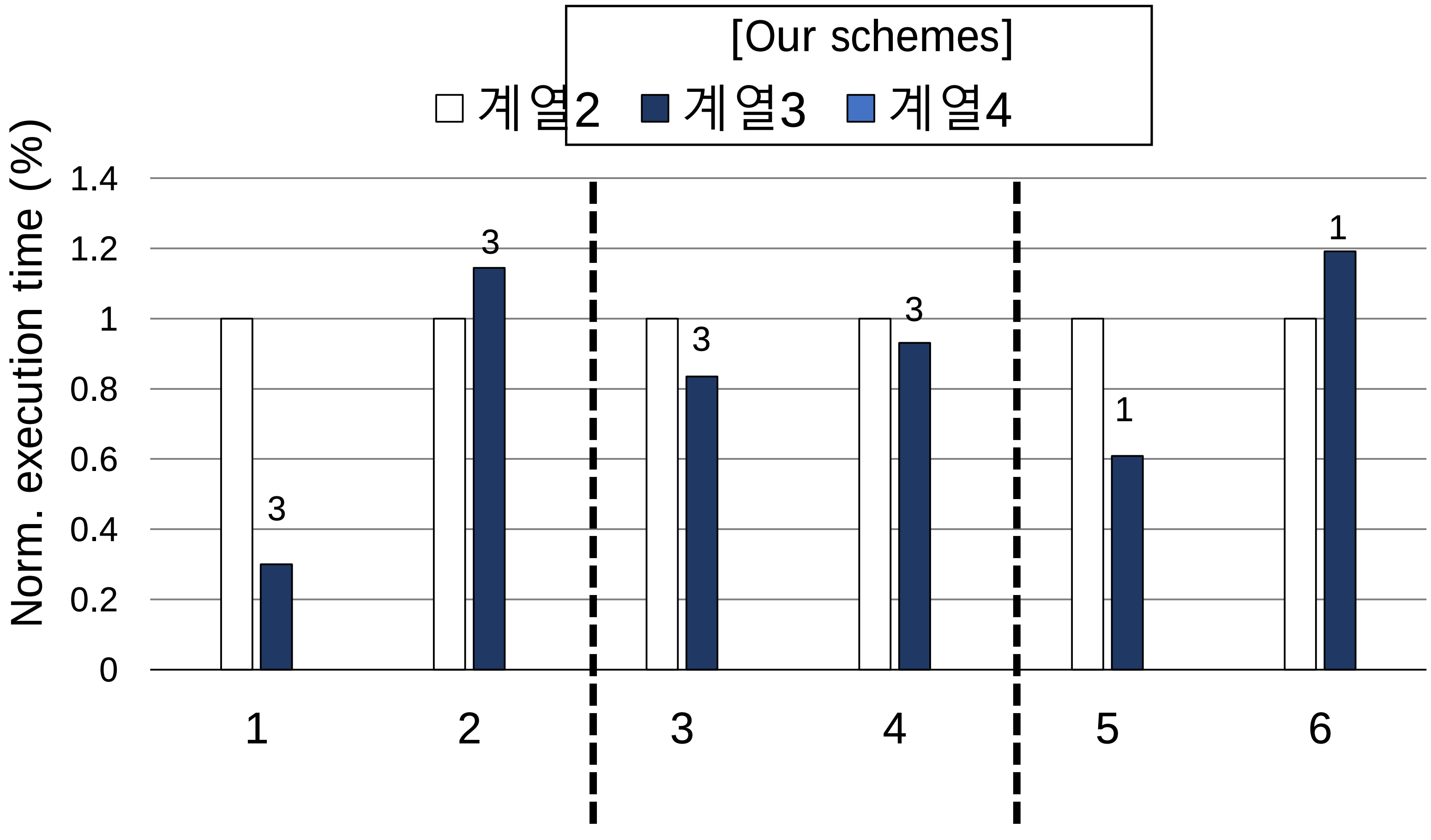
Experimental Environments

- Testbed
  - 12 HW threads (Intel Xeon)
  - 2 VMs with 12 vCPUs for each
  - Xen hypervisor 4.7
- Benchmarking workloads
  - `dedup` and `vips` from PARSEC
  - `exim` and `gmake` from MOSBENCH
- Pool configuration
  - Normal: 30ms (Xen default)
  - Micro-sliced: 0.1ms

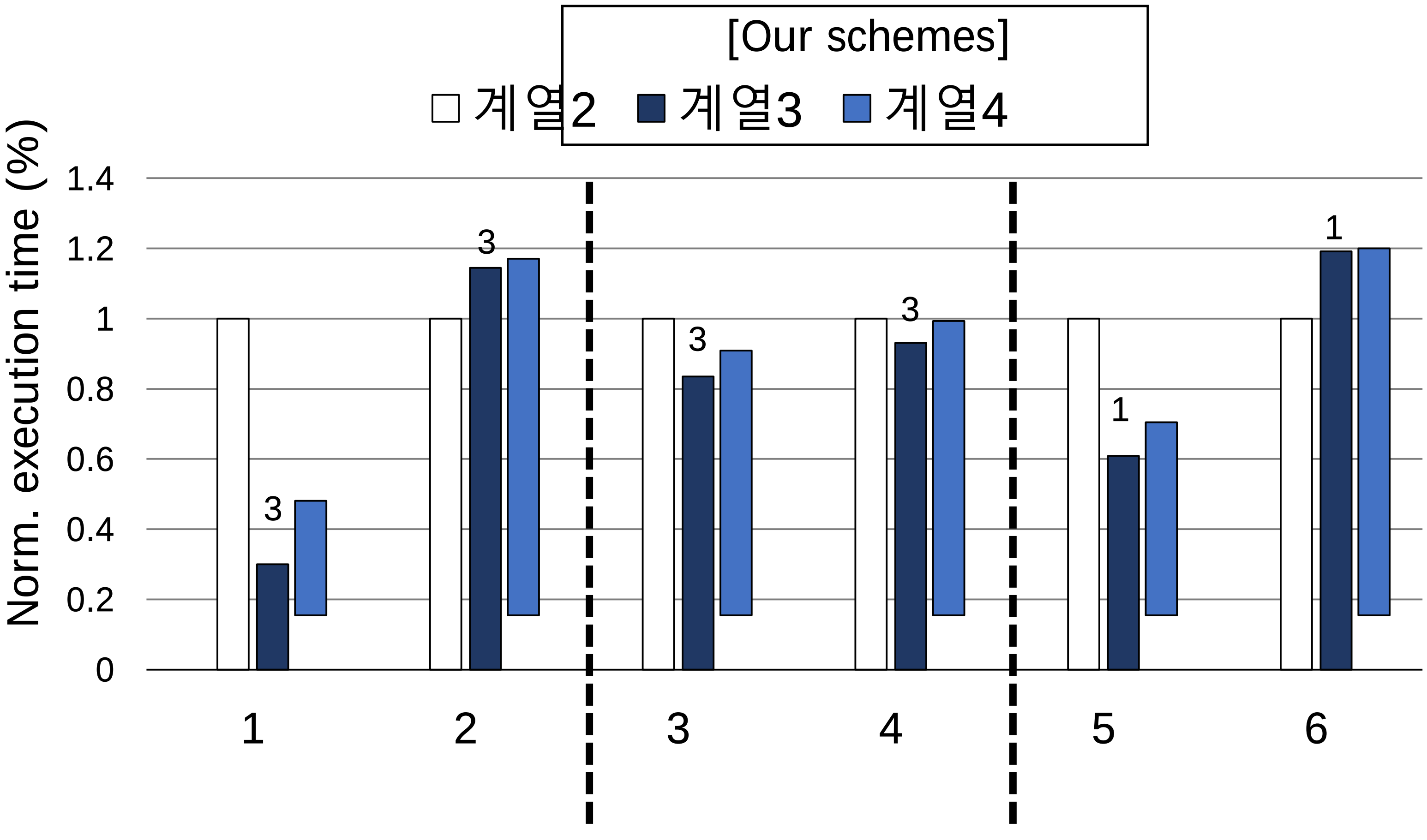


2-to-1 consolidation ratio

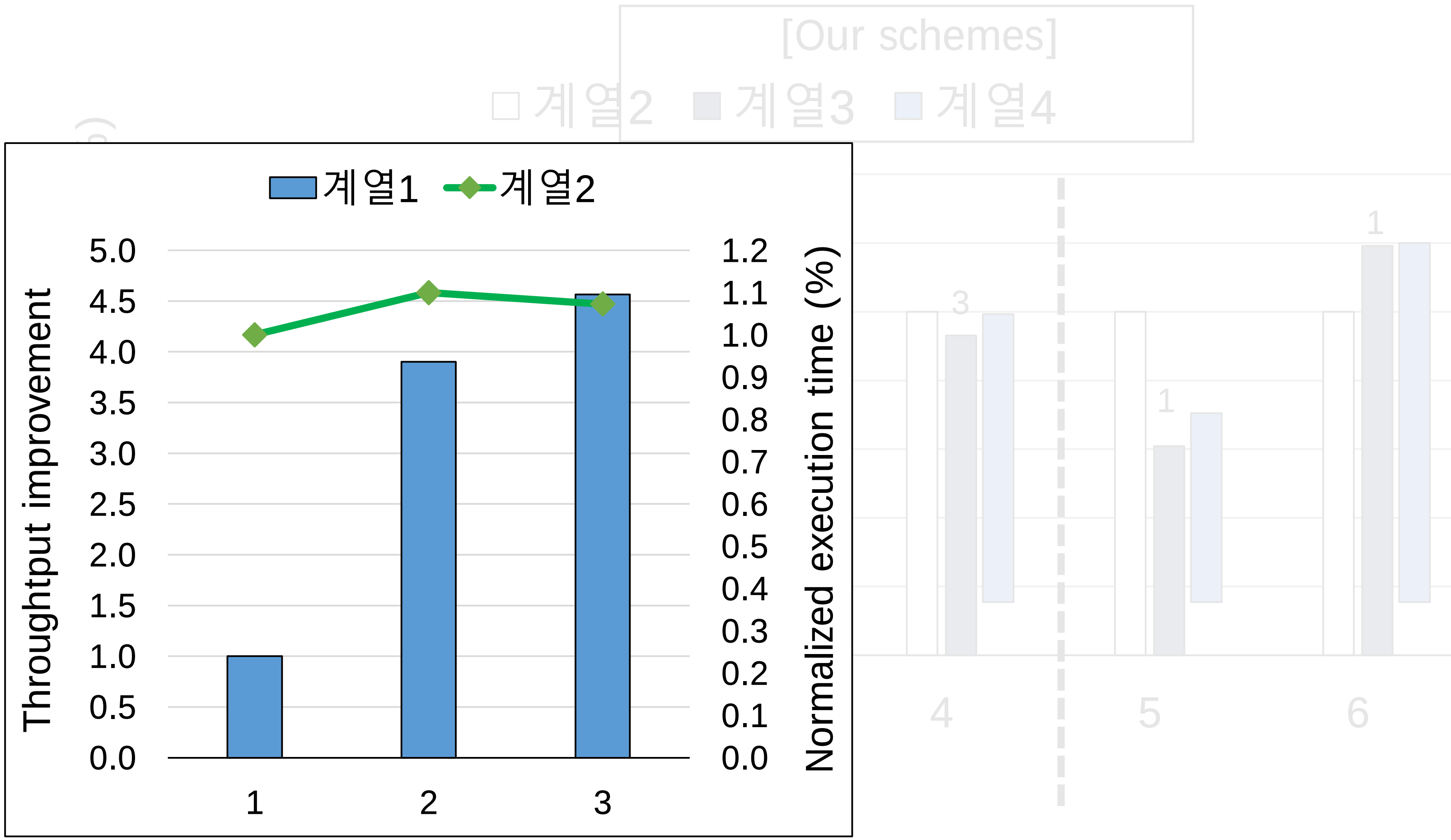
Performance of Micro-sliced Cores



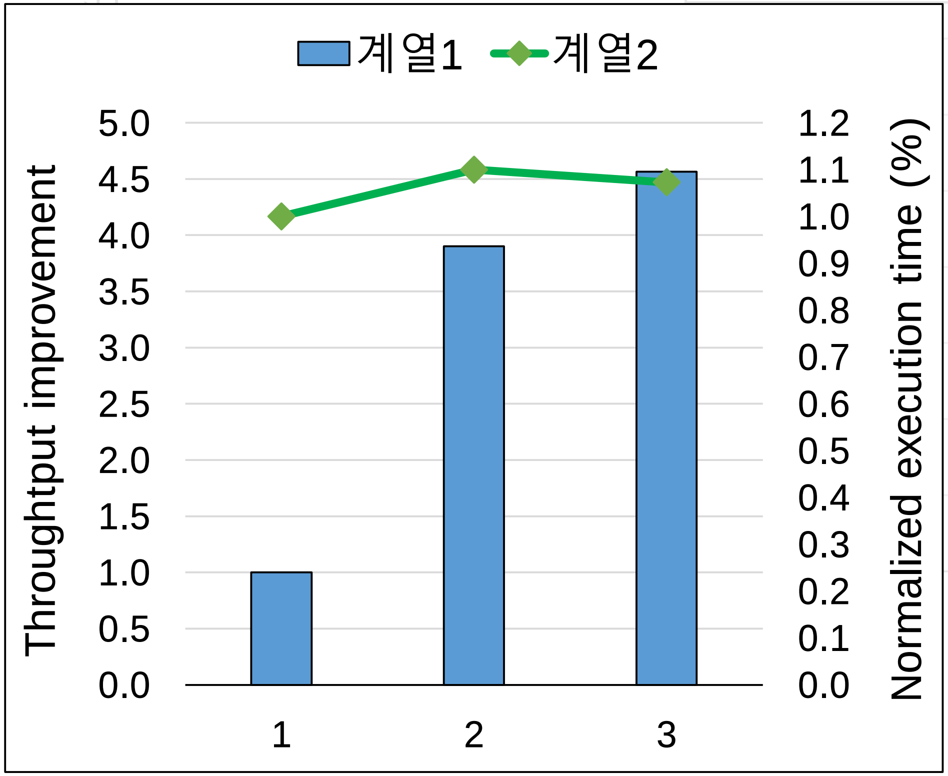
Performance of Micro-sliced Cores

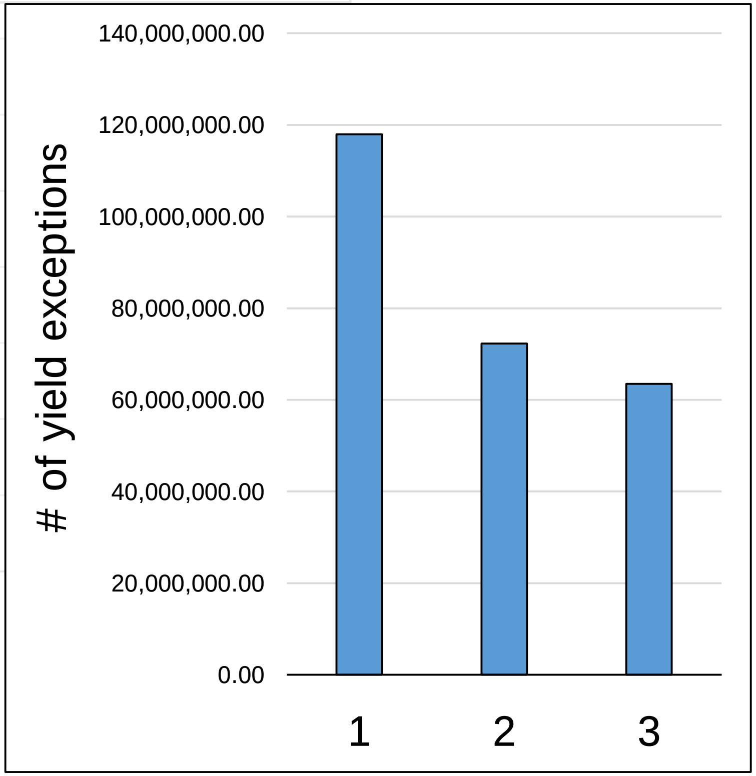


Performance of Micro-sliced Cores

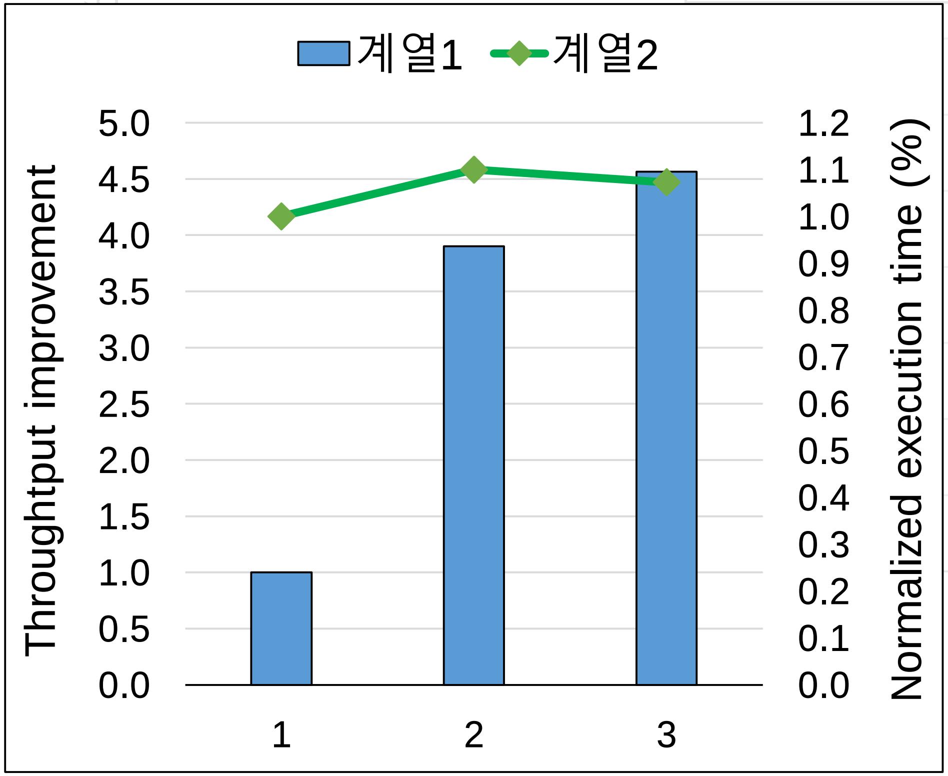


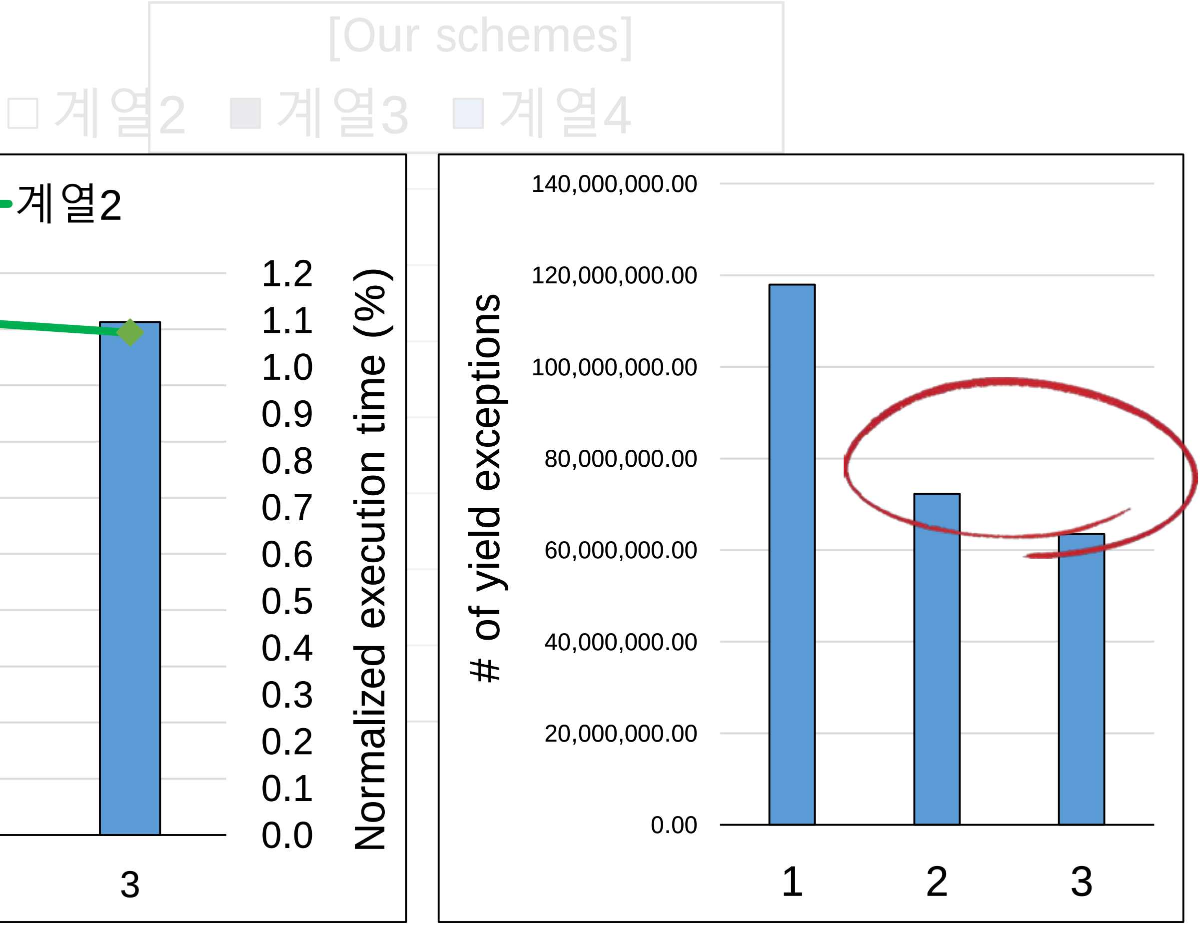
Performance of Micro-sliced Cores



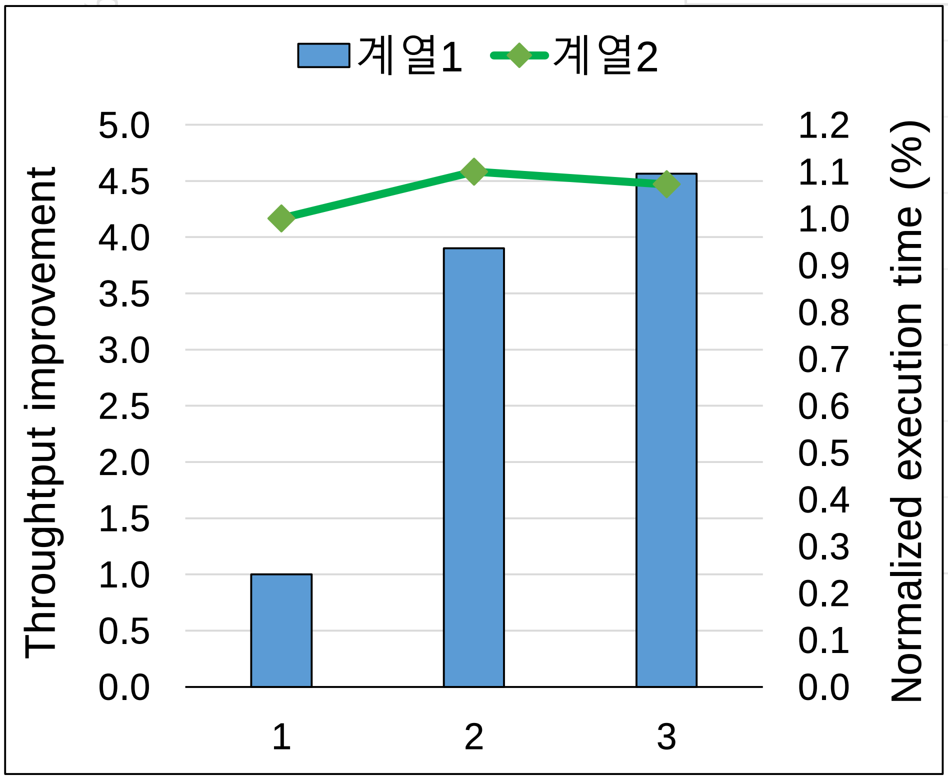


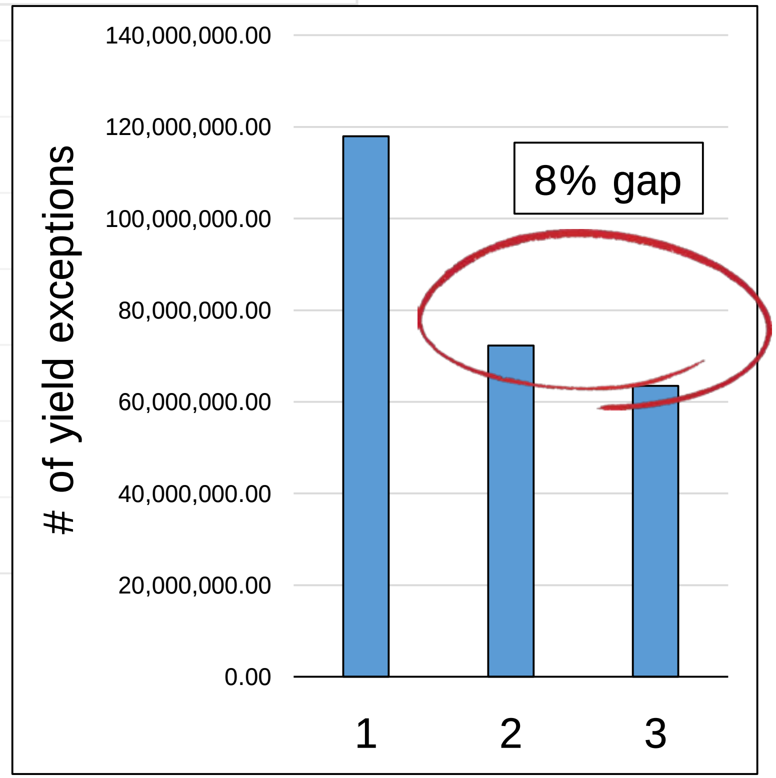
Performance of Micro-sliced Cores



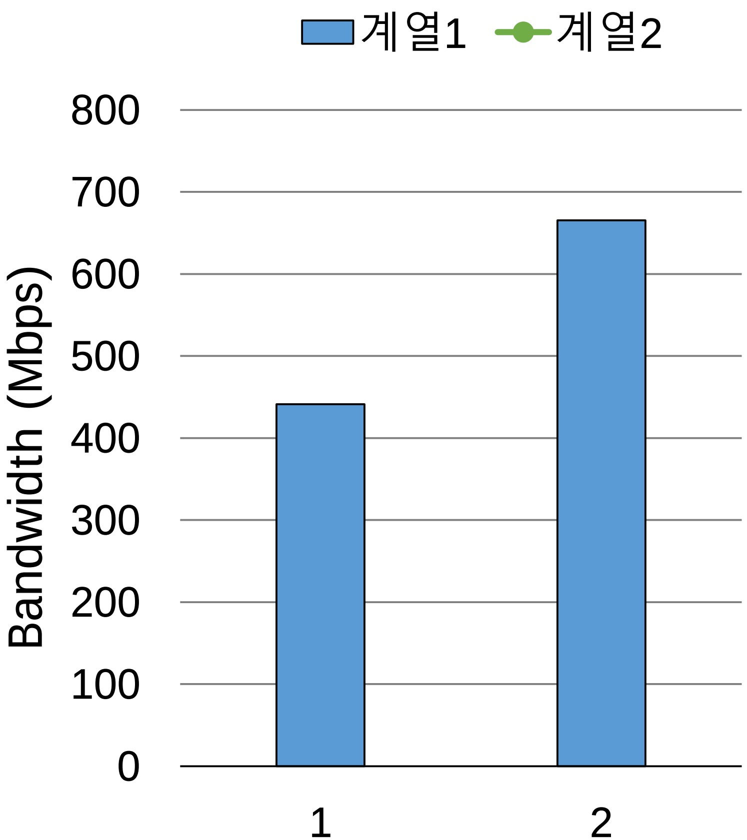


Performance of Micro-sliced Cores



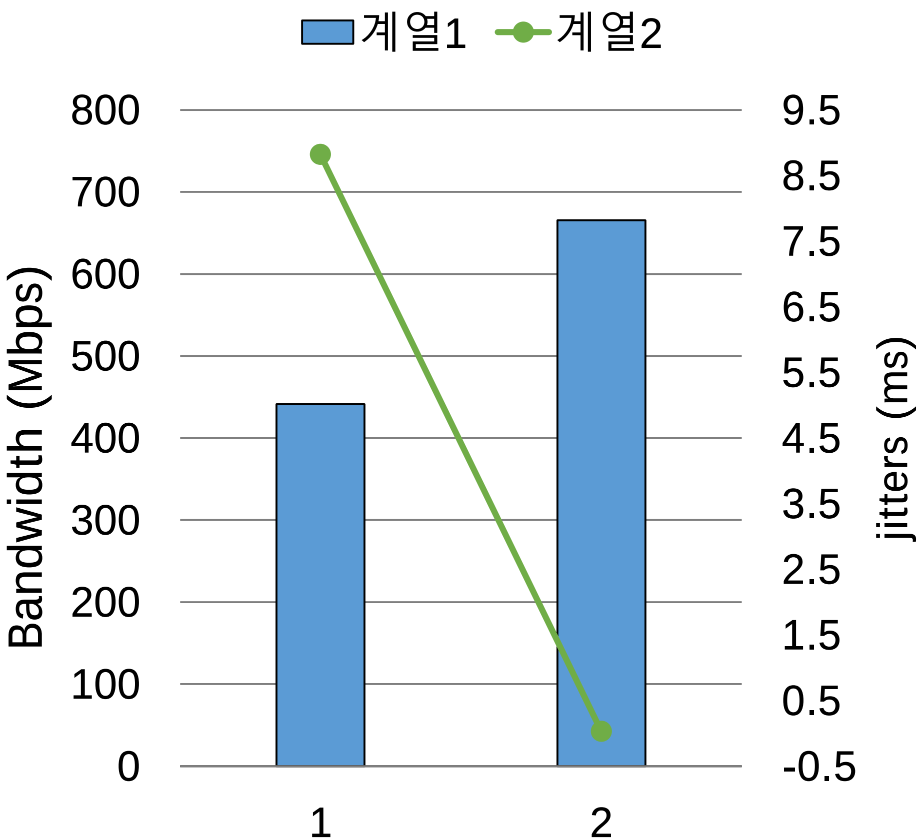


I/O Performance



	Workloads
VM-1	iPerf lookbusy
VM-2	lookbusy

I/O Performance



	Workloads
VM-1	iPerf lookbusy
VM-2	lookbusy

Conclusions

- We introduced a new approach to mitigate the virtual time discontinuity problem
- Three distinct contributions
  - Precise detection of urgent tasks
  - Guest OS transparency
  - Dynamic adjustment of the micro-sliced cores
- Overhead is very low for applications which do not frequently use OS services

Thank You!

jsahn@ajou.ac.kr

Accelerating Critical OS Services in Virtualized Systems with Flexible Micro-sliced Cores

Jeongseob Ahn[*], Chang Hyun Park[‡], Taekyung Heo[‡], Jaehyuk Huh[‡]

[*] AJOU UNIVERSITY

[‡] KAIST