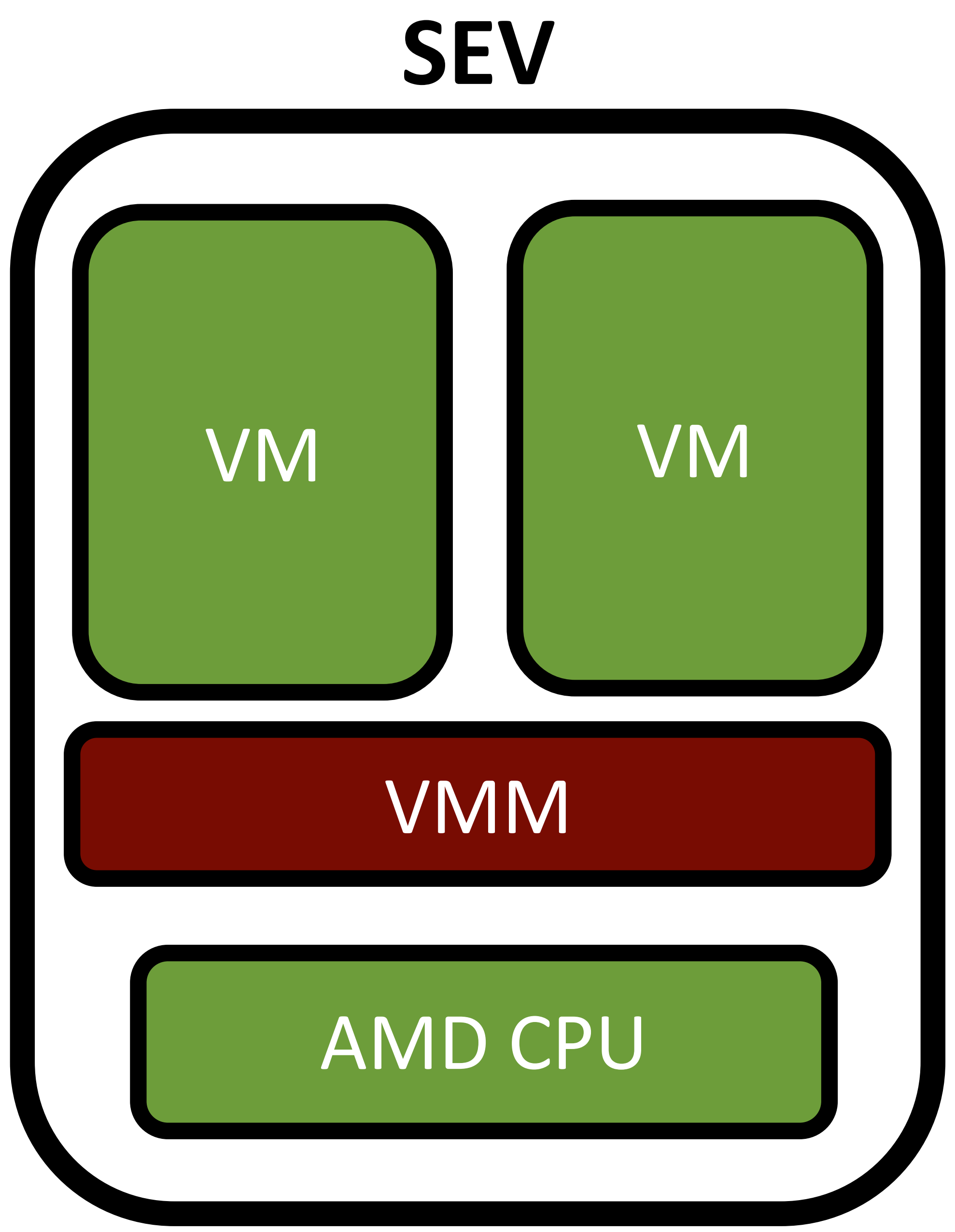
TLB Poisoning Attacks
on AMD Secure Encrypted Virtualization

Mengyuan Li[1], Yinqian Zhang[2], Huibo Wang[3], Kang Li[3] and Yueqiang Cheng[4]

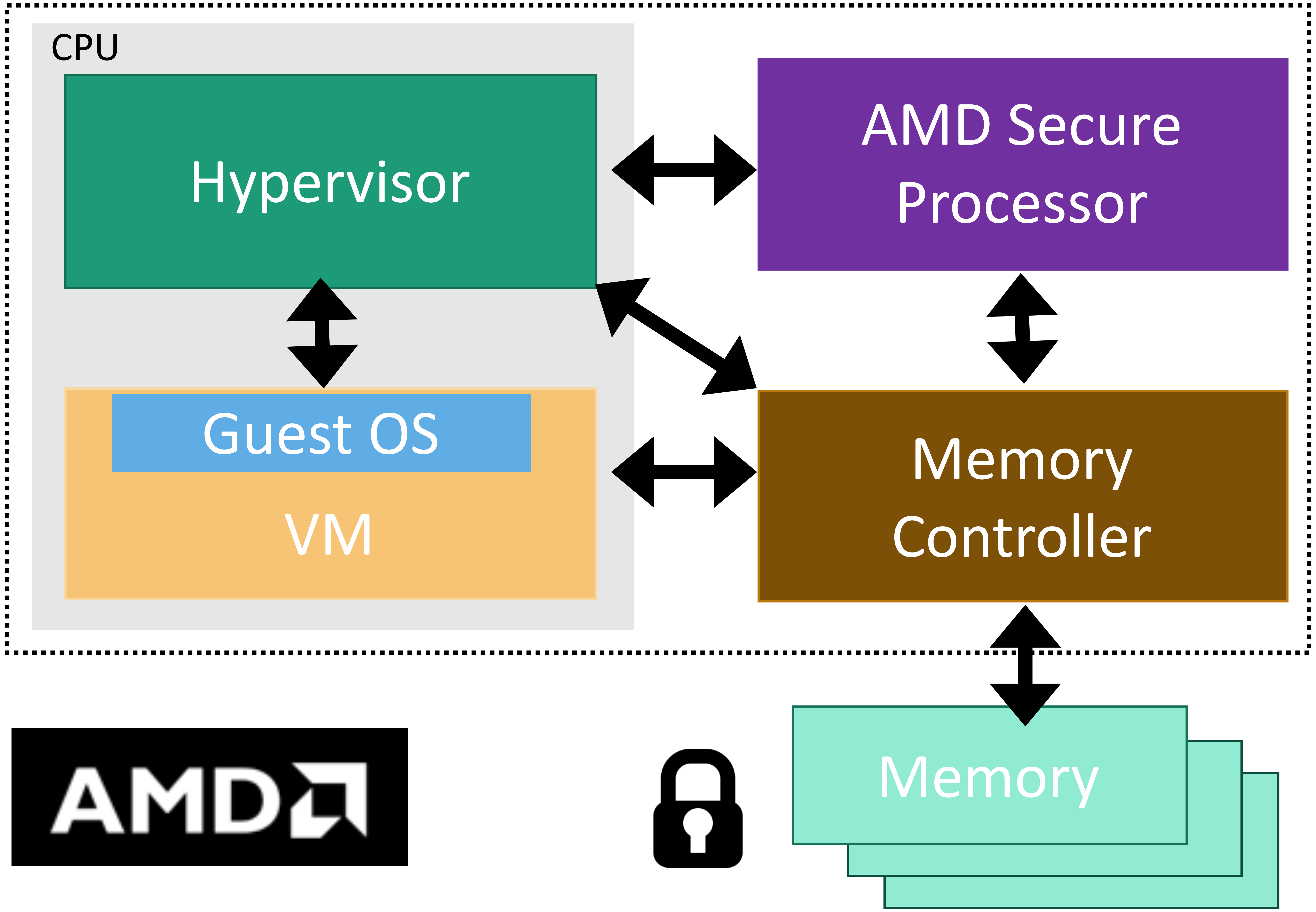
[1]The Ohio State University, [2]Southern University of Science and Technology, [3]Baidu Security, [4]NIO Security Research

AMD Secure Encrypted Virtualization (SEV)

"SEV technology is built around a threat model where an attacker is assumed to have access to not only execute user level privileged code on the target machine, but can potentially execute malware at the higher privileged hypervisor level as well."



Hardware Memory Encryption



❑CPU

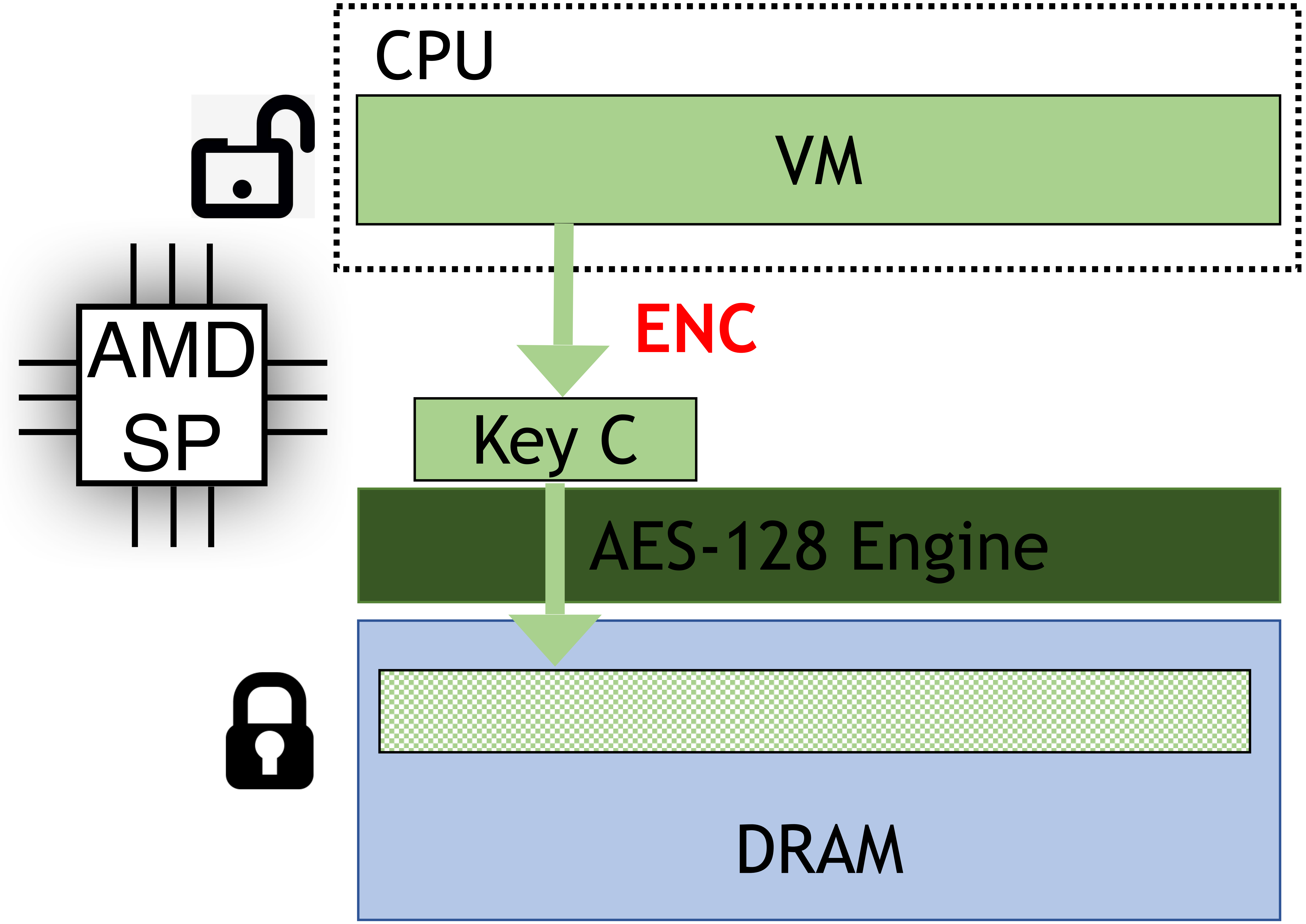
❑AMD Secure Processor

- Manages AES Keys
- Handle SEV API

❑Memory Controller

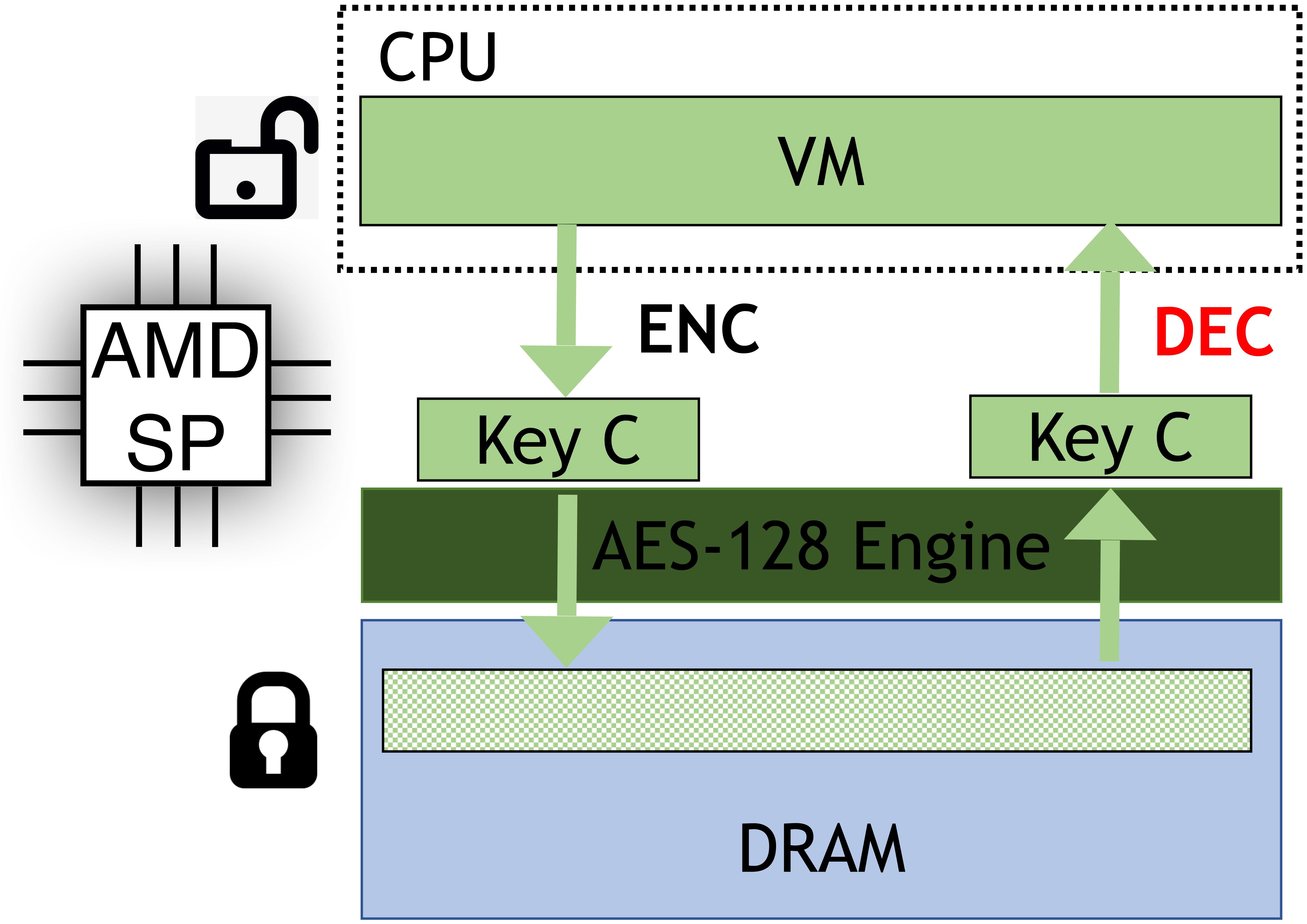
- Memory Encryption Engine(MEE)
- AES encryption/decryption

Hardware Memory Encryption



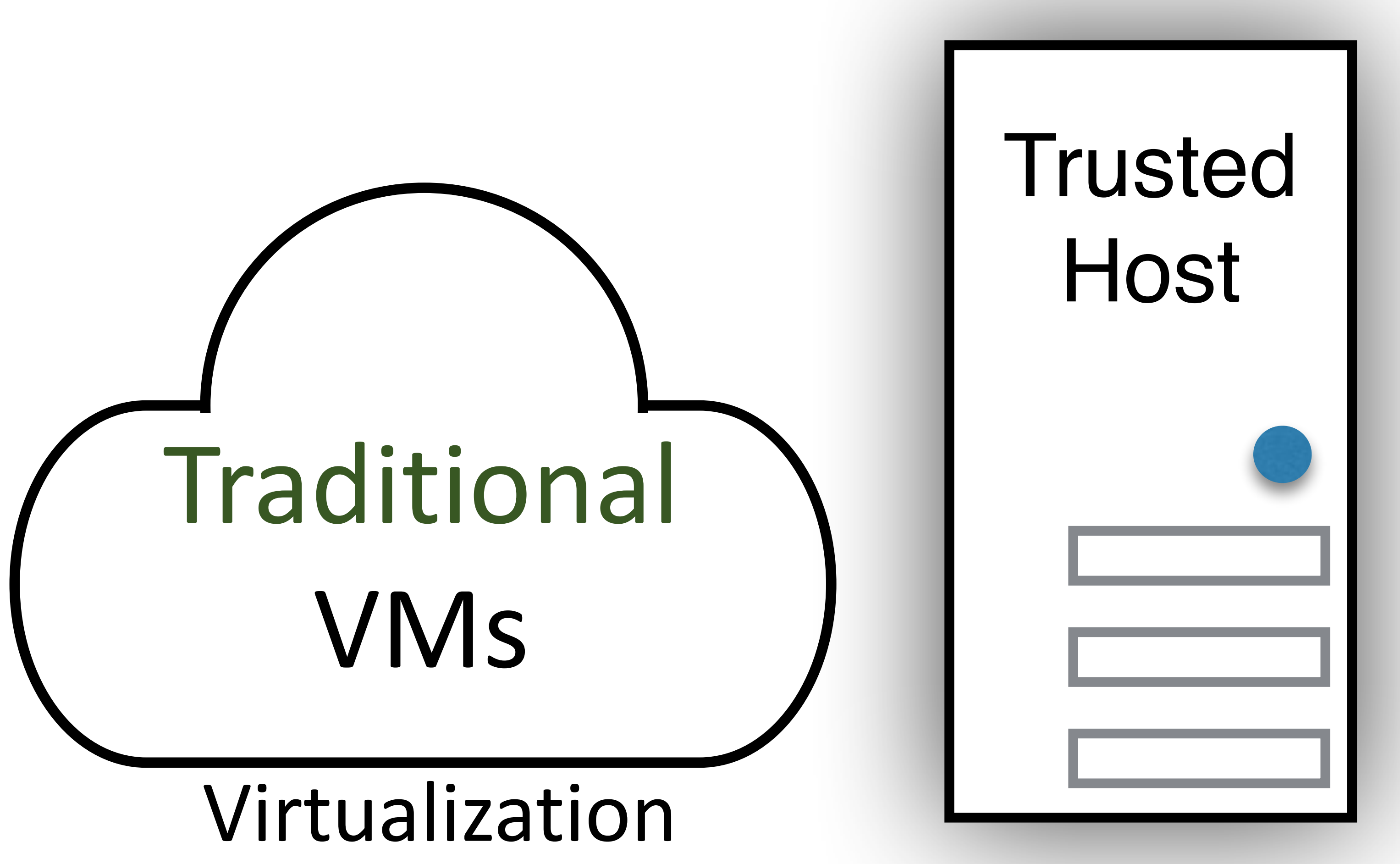
- Data are unencrypted in CPU.

Hardware Memory Encryption



- Data are encrypted in the memory.

New vulnerabilities?

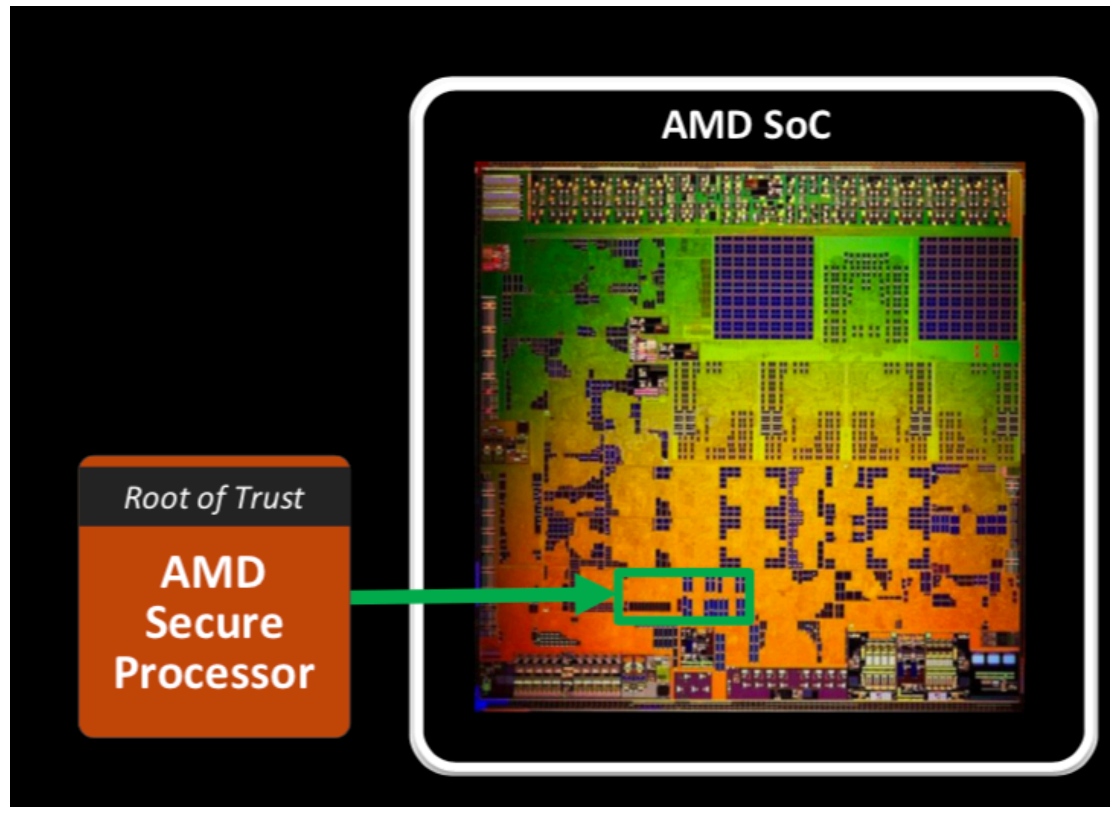


Traditional VMs

Virtualization

Trusted Host

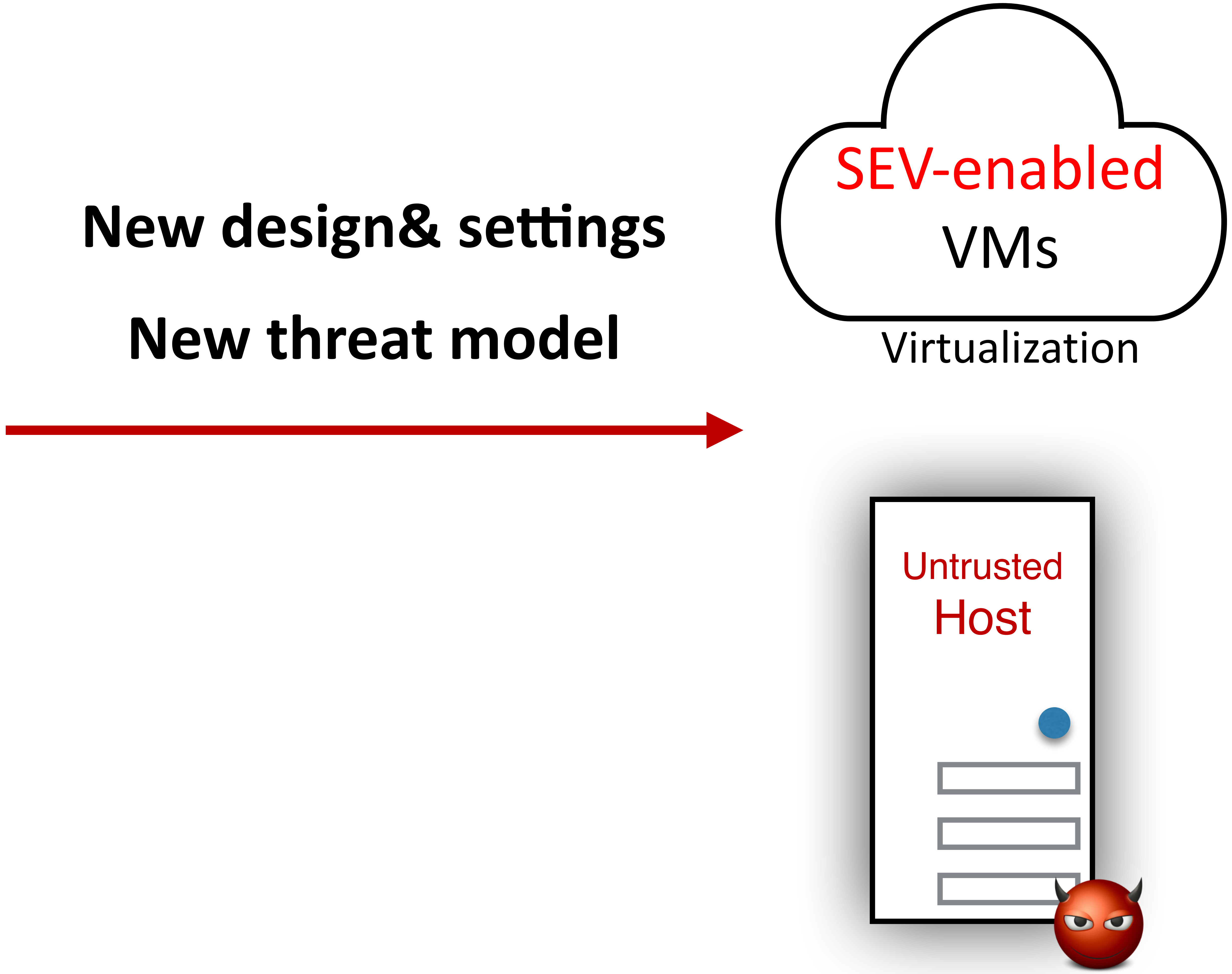
+



Hardware Memory Encryption

New design& settings

New threat model

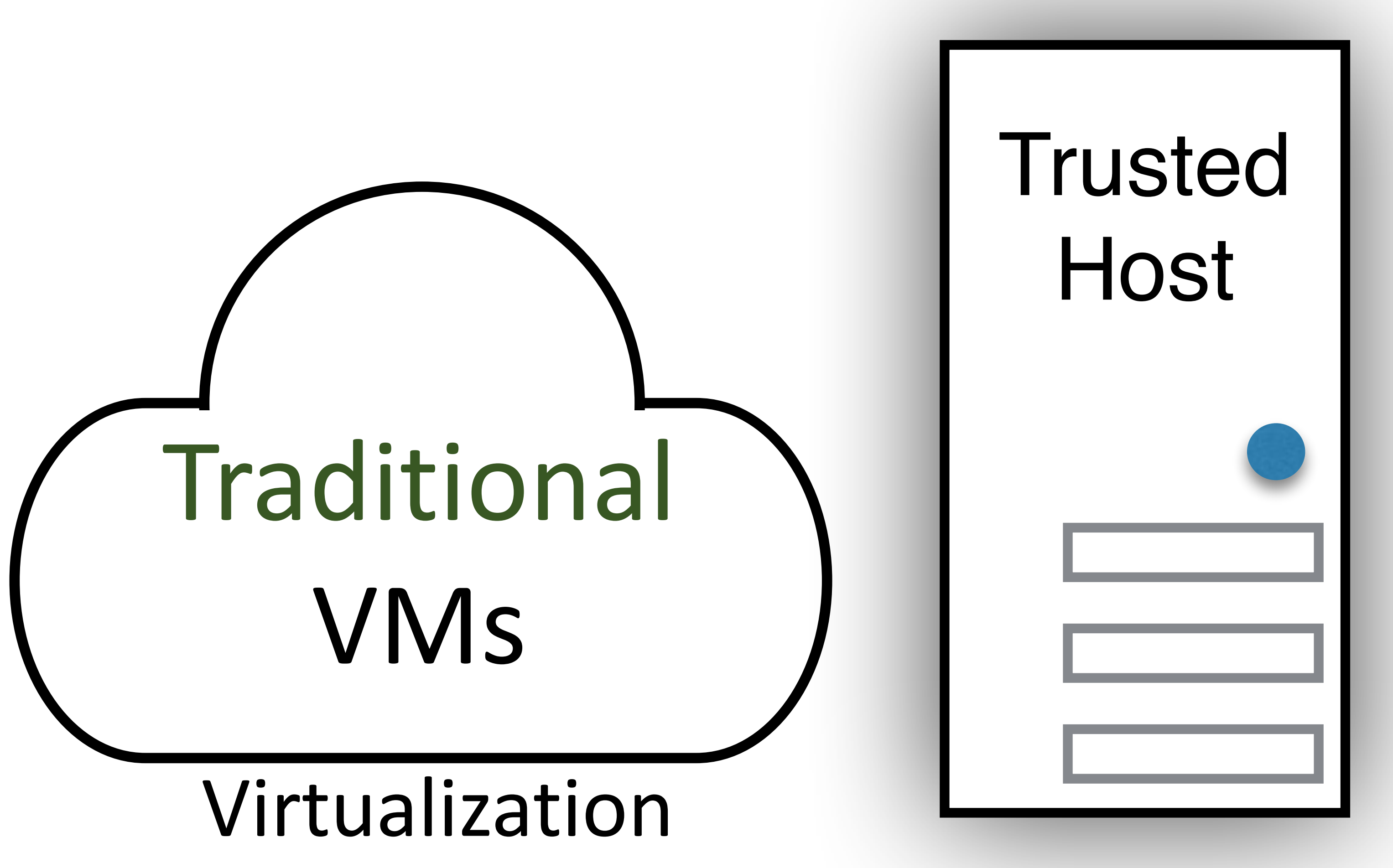


SEV-enabled VMs

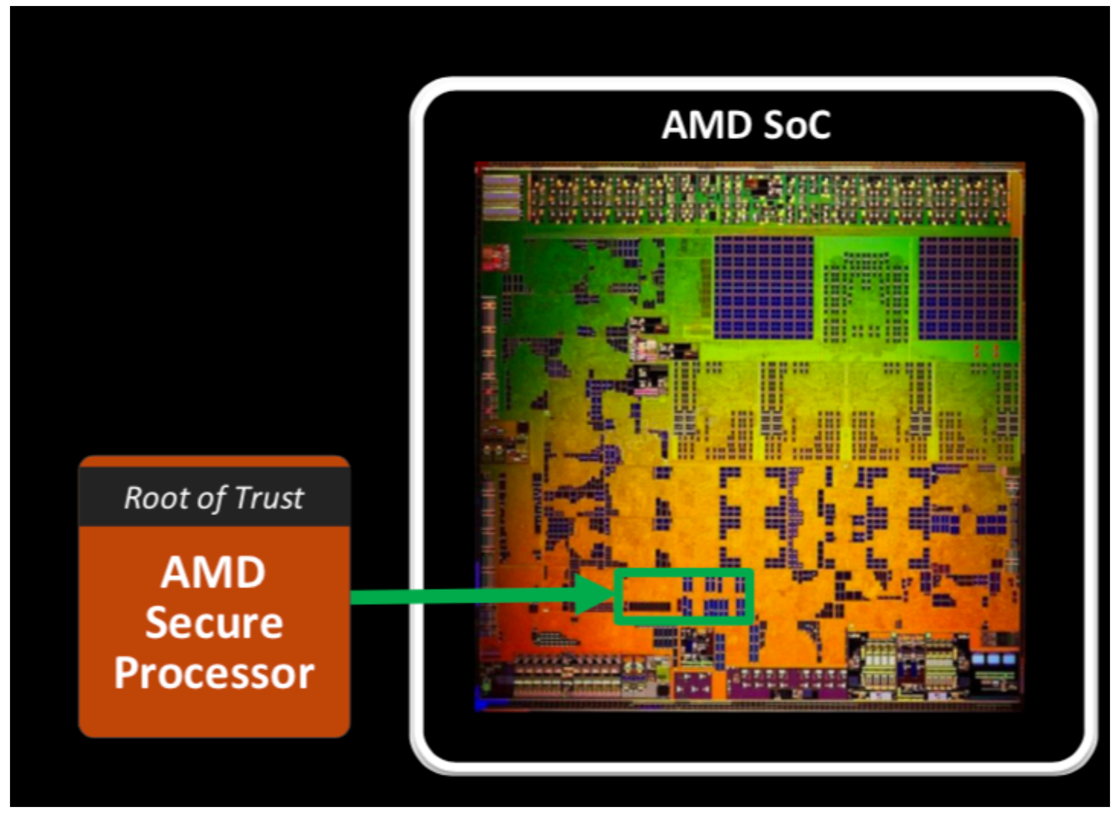
Virtualization

Untrusted Host

New vulnerabilities?



+

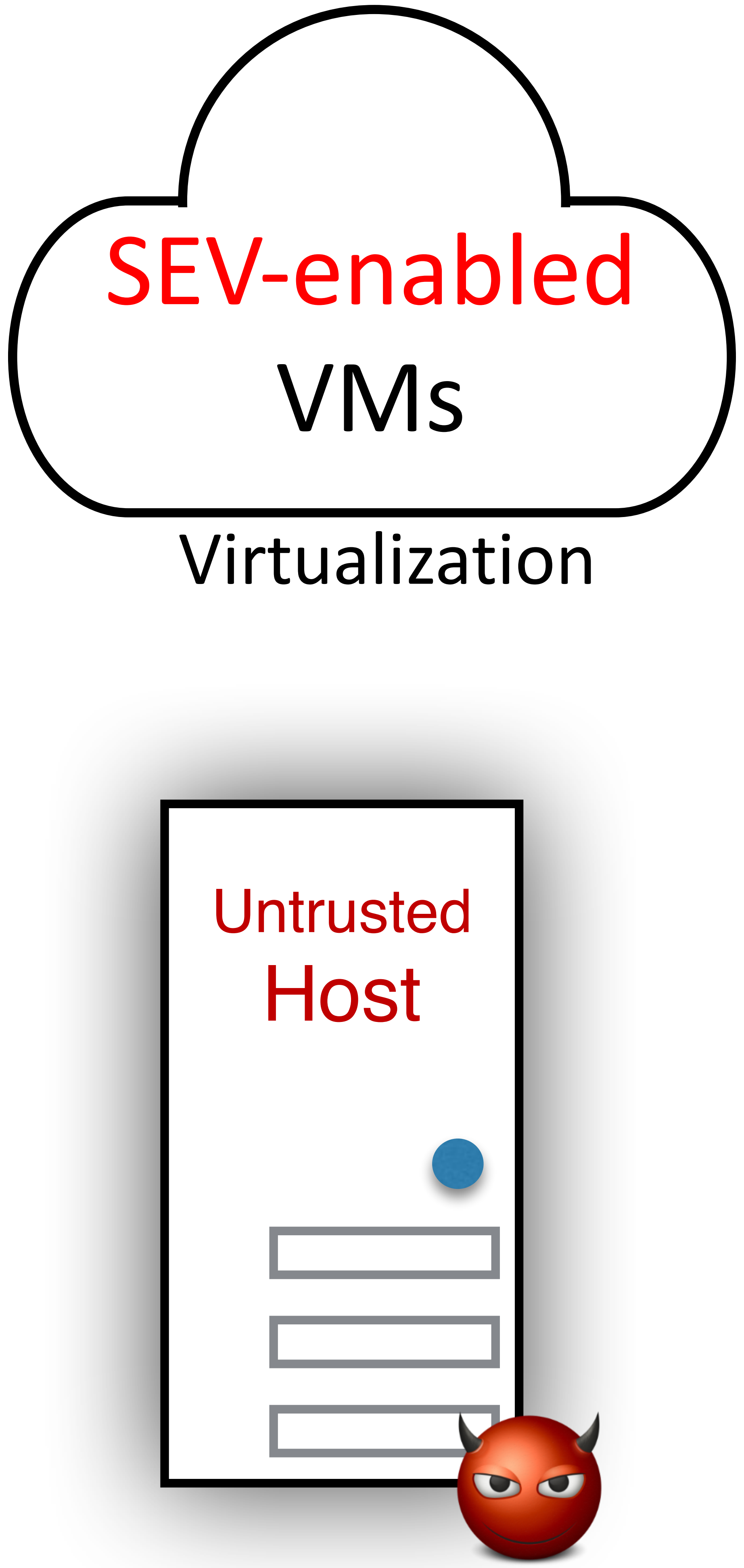


Hardware Memory Encryption

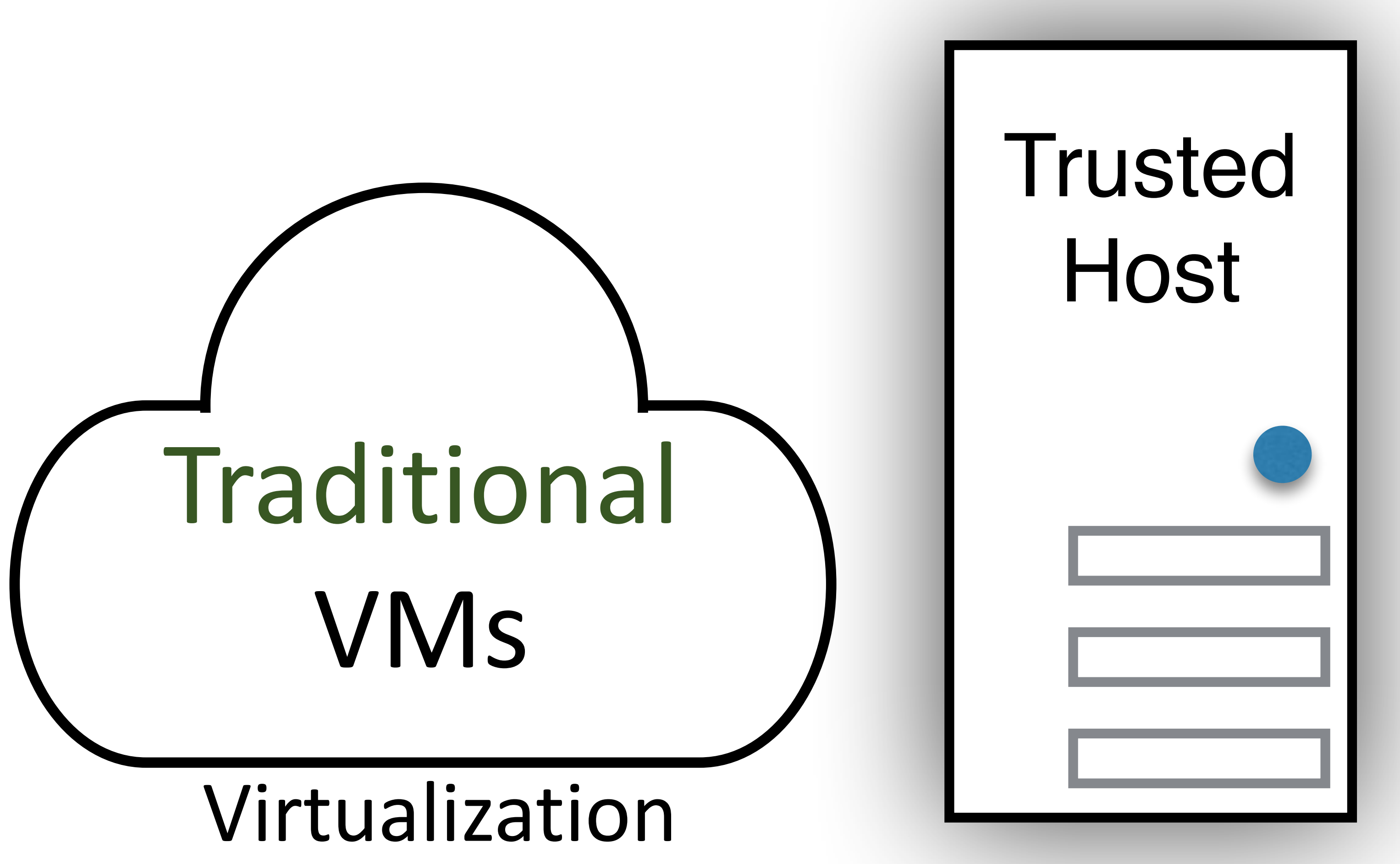
New design& settings

New threat model

New vulnerabilities caused by the inconsistency?



New vulnerabilities?

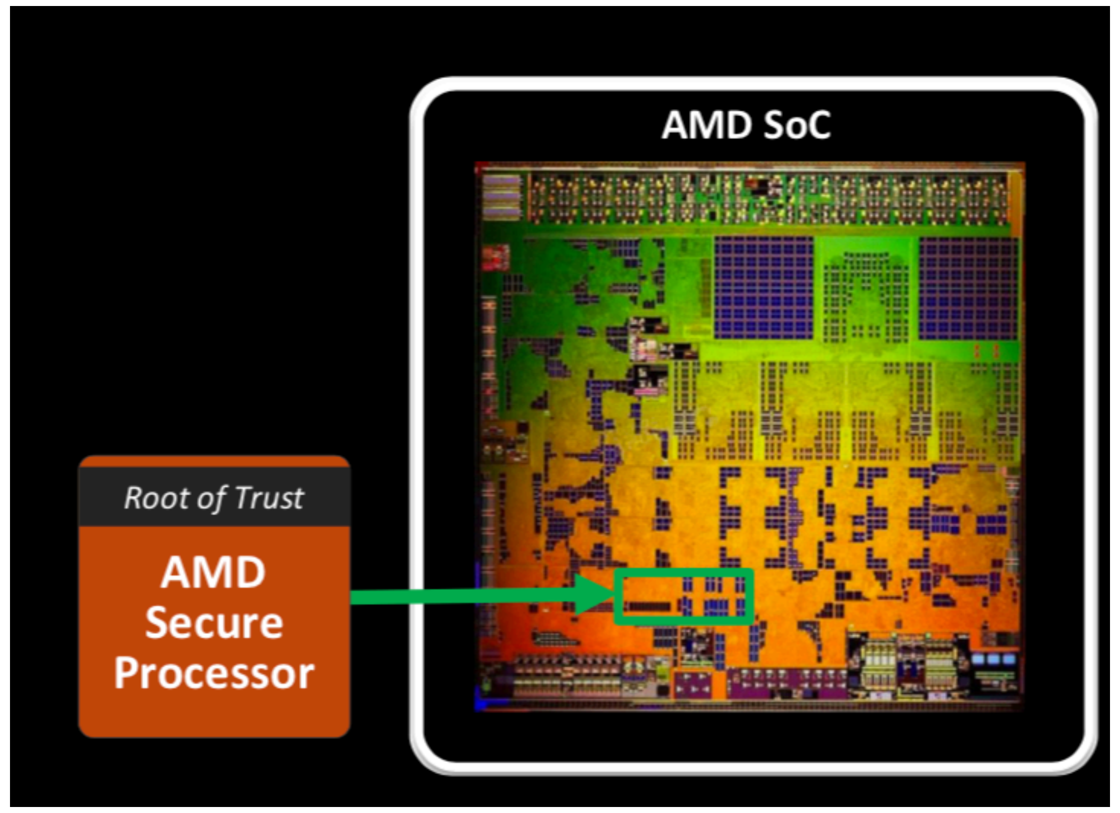


Traditional VMs

Virtualization

Trusted Host

+



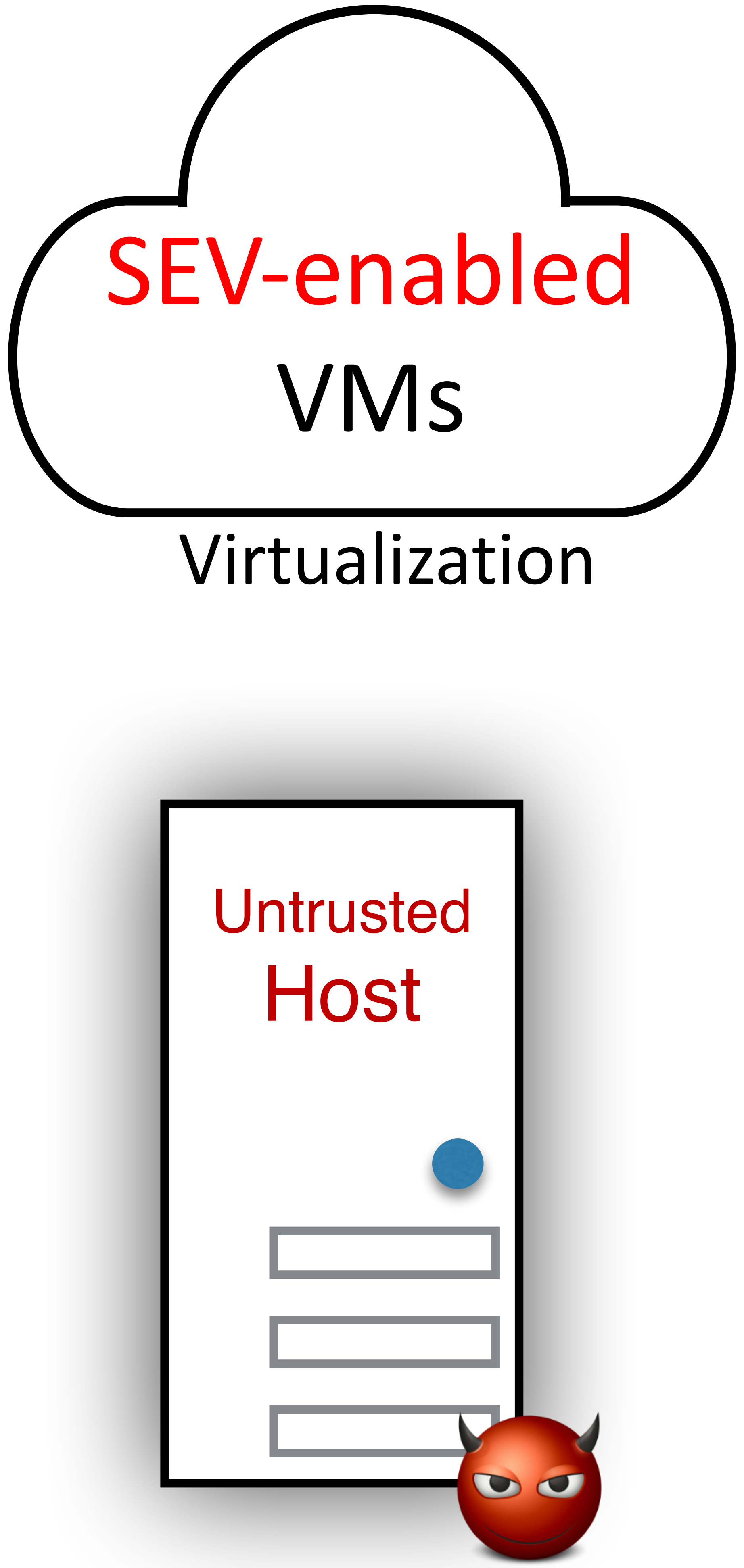
Hardware Memory Encryption

New design& settings

New threat model

New vulnerabilities caused by the inconsistency?

TLB isolation mechanism



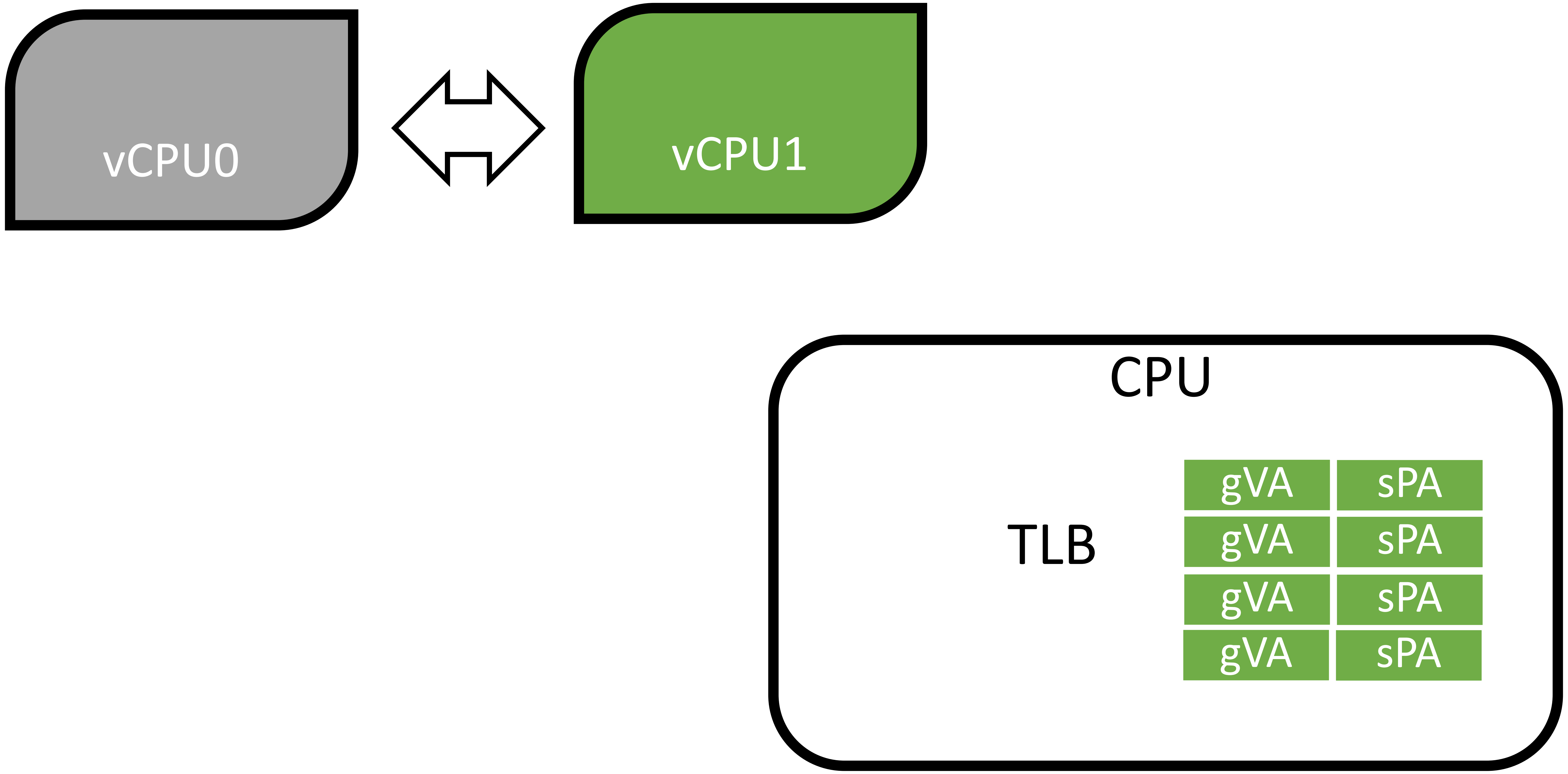
SEV-enabled VMs

Virtualization

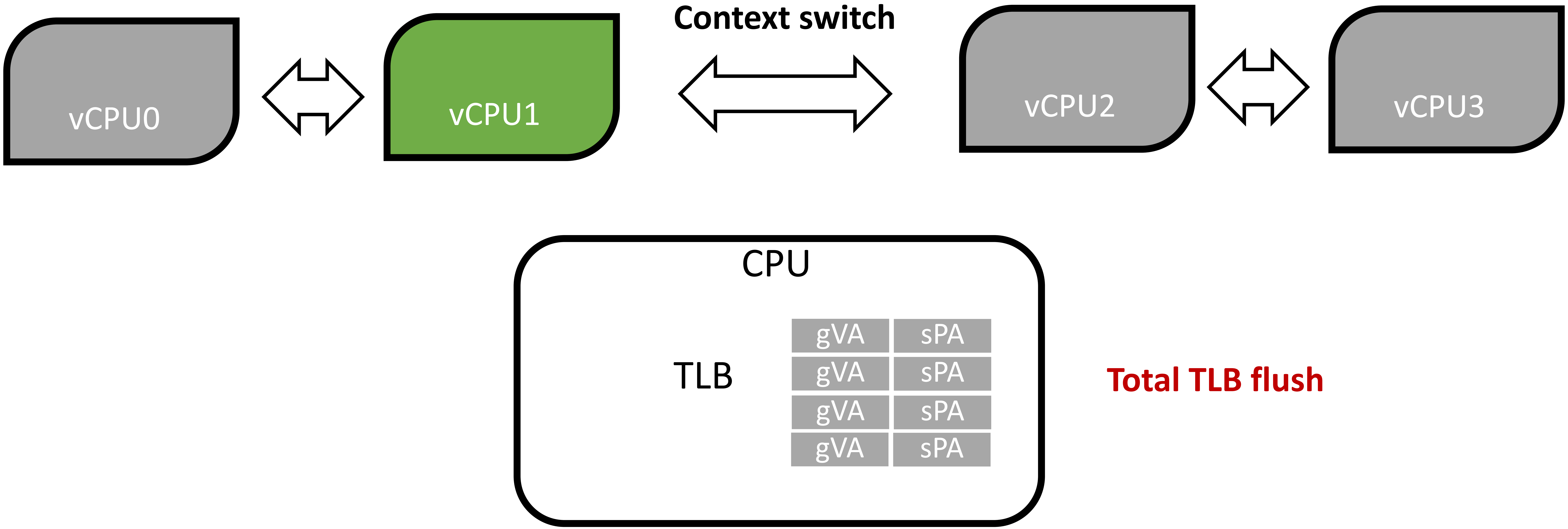
Untrusted Host

ASID-based TLB Isolation in VM’s Lifetime

Address Space Identifier (ASID)



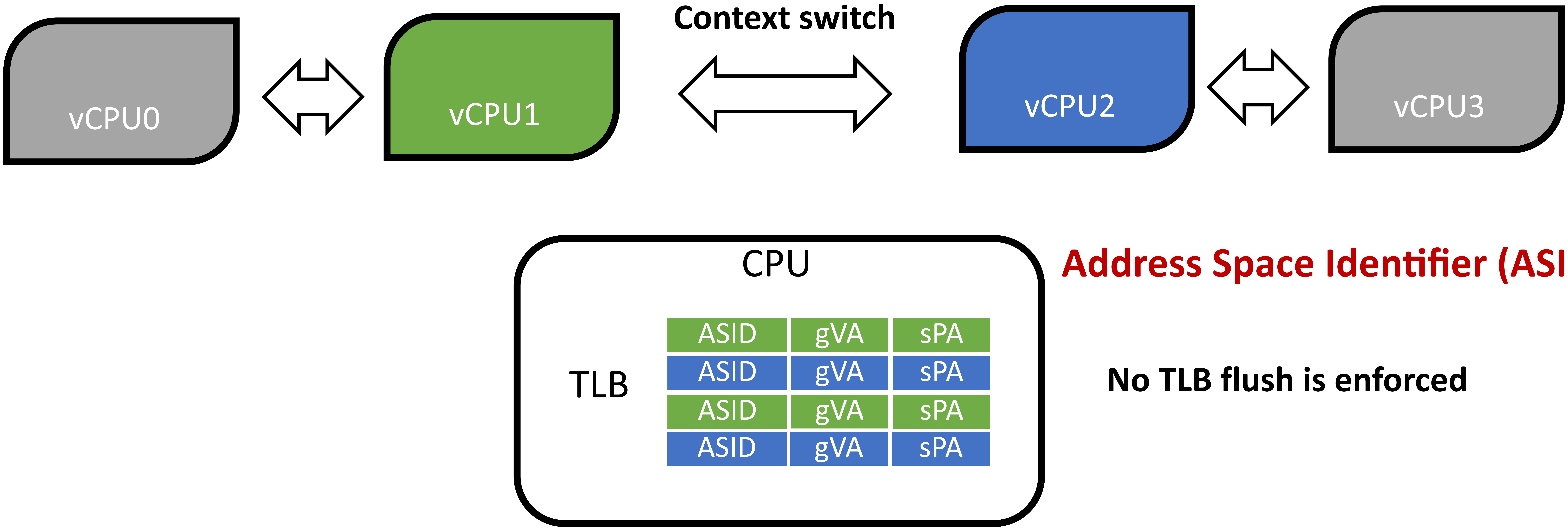
ASID-based TLB Isolation in VM’s Lifetime



gVA	sPA
gVA	sPA
gVA	sPA
gVA	sPA

Total TLB flush

ASID-based TLB Isolation in VM’s Lifetime

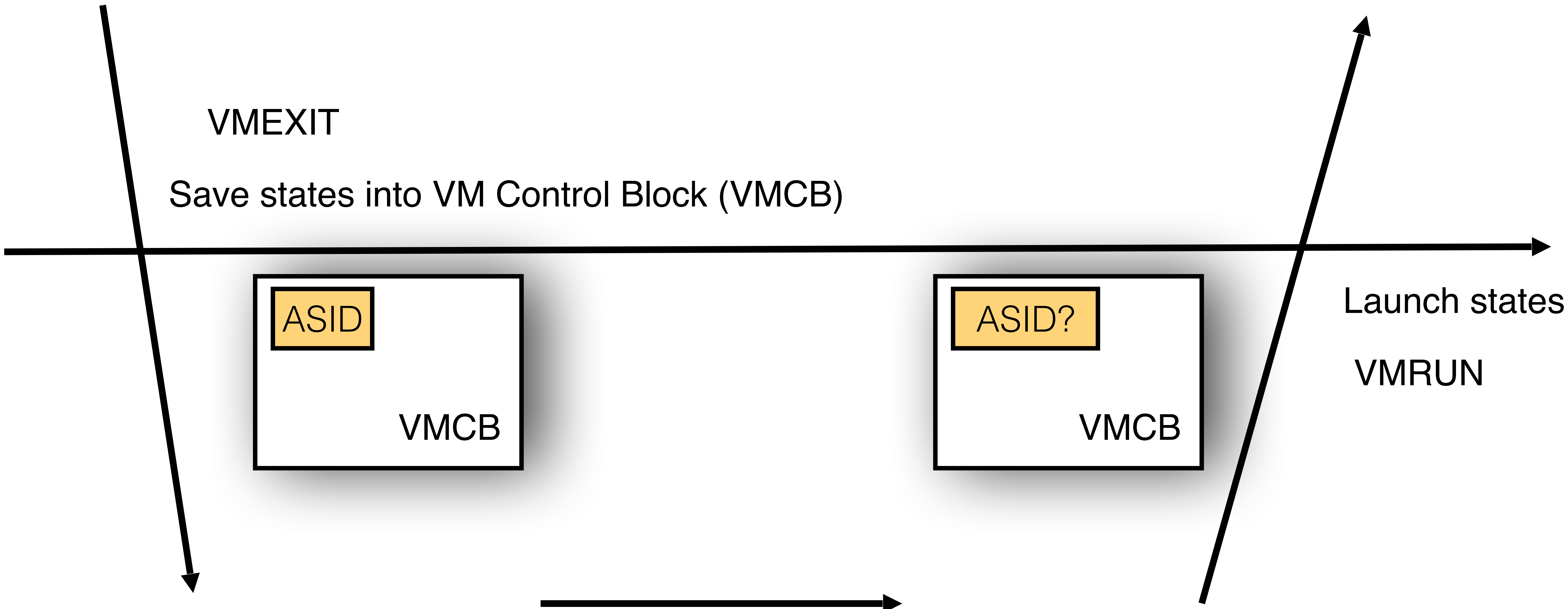


ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA

Address Space Identifier (ASID)

No TLB flush is enforced

ASID assignment in traditional VM



Guest VM's world

VMEXIT

Save states into VM Control Block (VMCB)

ASID

VMCB

ASID?

VMCB

Launch states from VMCB

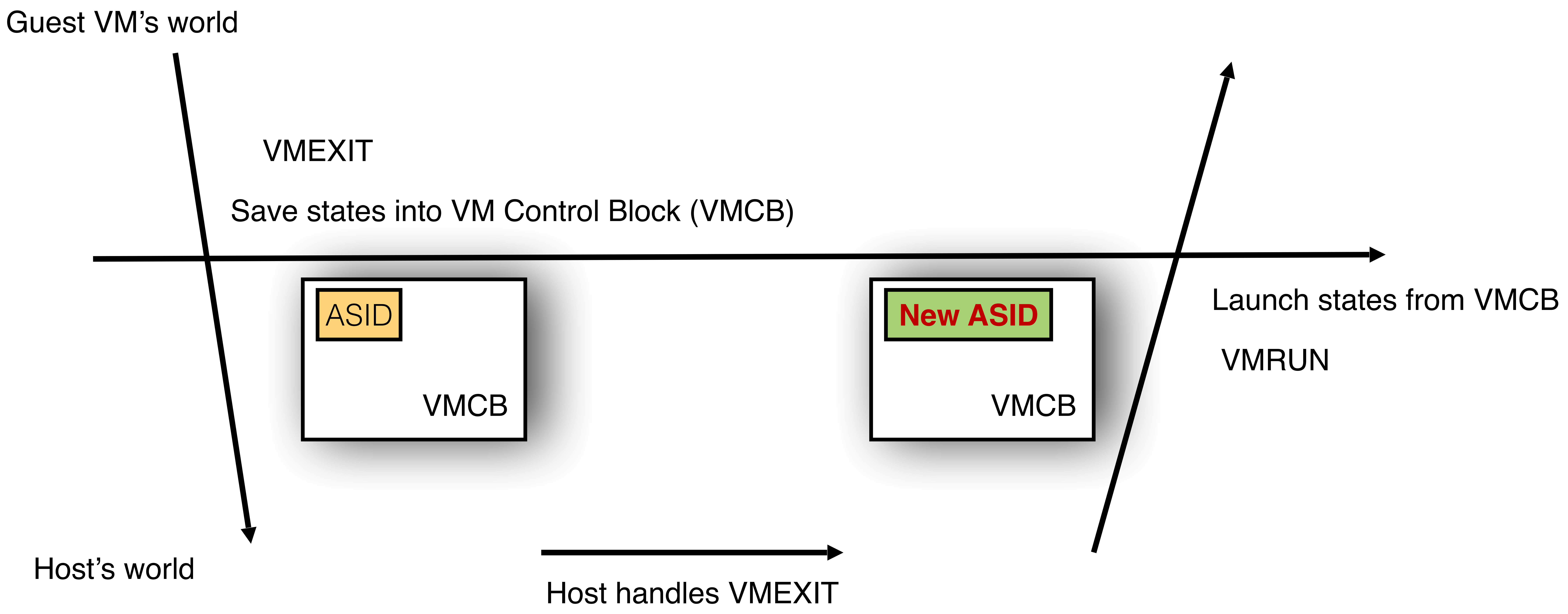
VMRUN

Host's world

Host handles VMEXIT

Check TLB condition before VMRUN

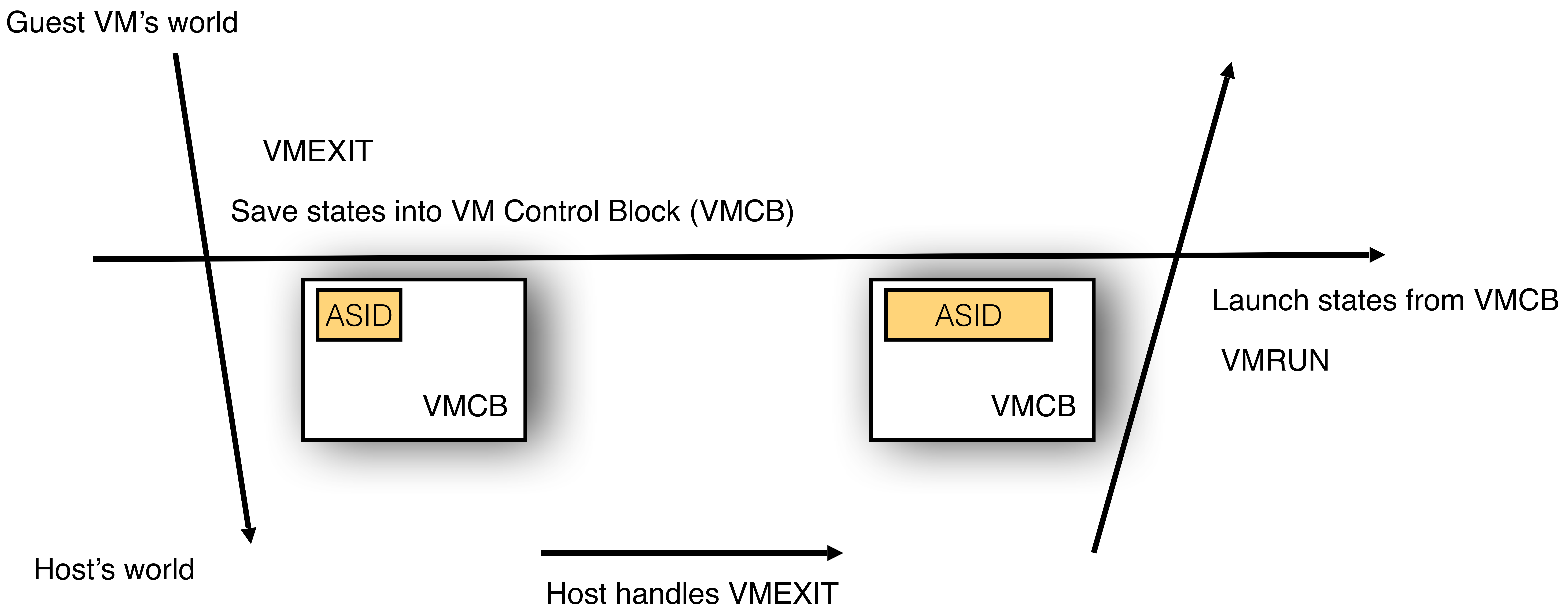
ASID-based TLB Isolation in traditional VM



Check TLB condition before VMRUN

a) **move to a new CPU core => Assign a new ASID**
b) **Observed vCPU-switch => ASID++**

ASID-based TLB Isolation in traditional VM



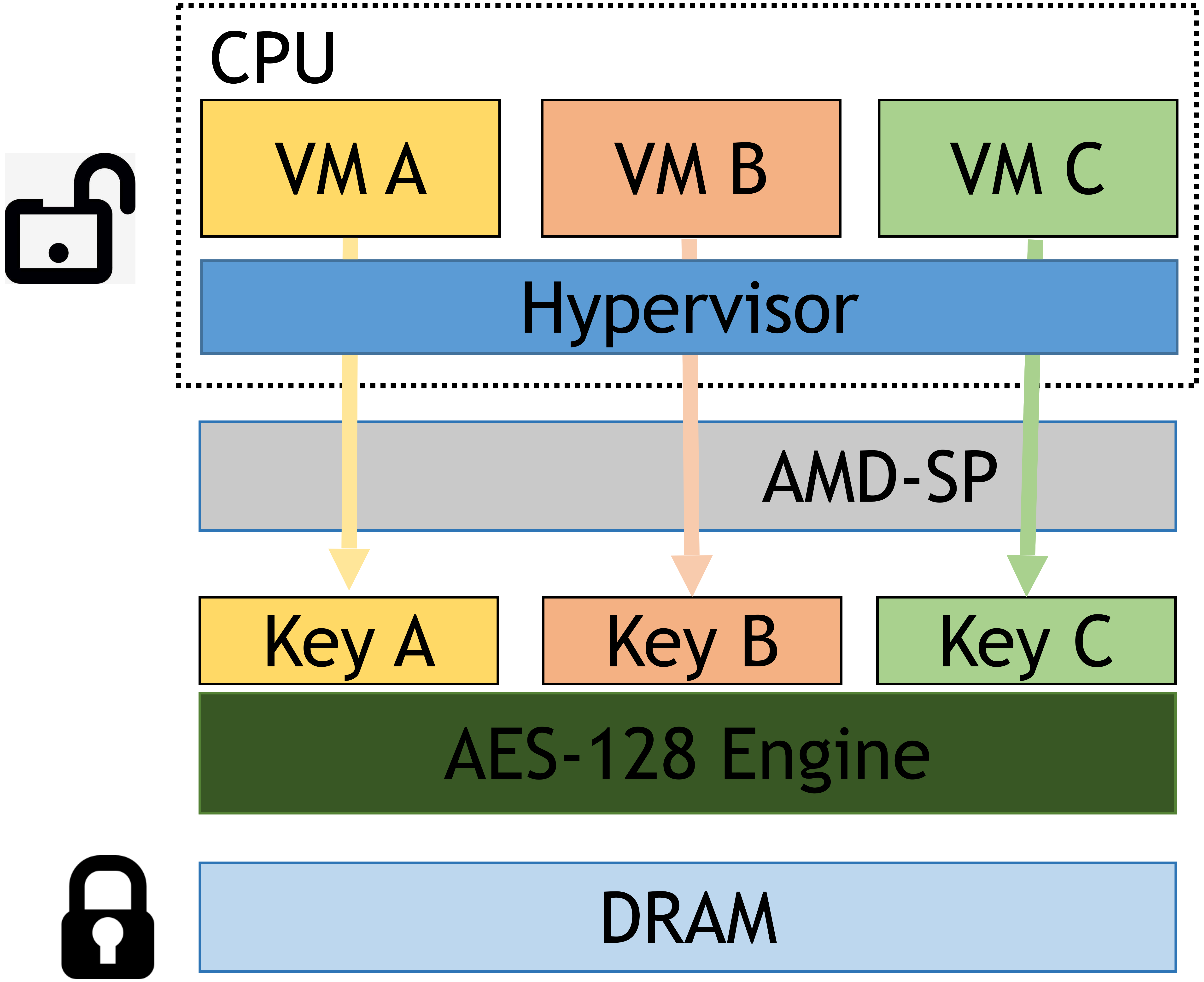
Check TLB condition before VMRUN

a) move to a new CPU core => Assign a new ASID

b) Observed vCPU-switch => ASID++

c) Otherwise => Unchanged ASID

ASID’s new role in SEV



CPU

VM A

VM B

VM C

Hypervisor

AMD-SP

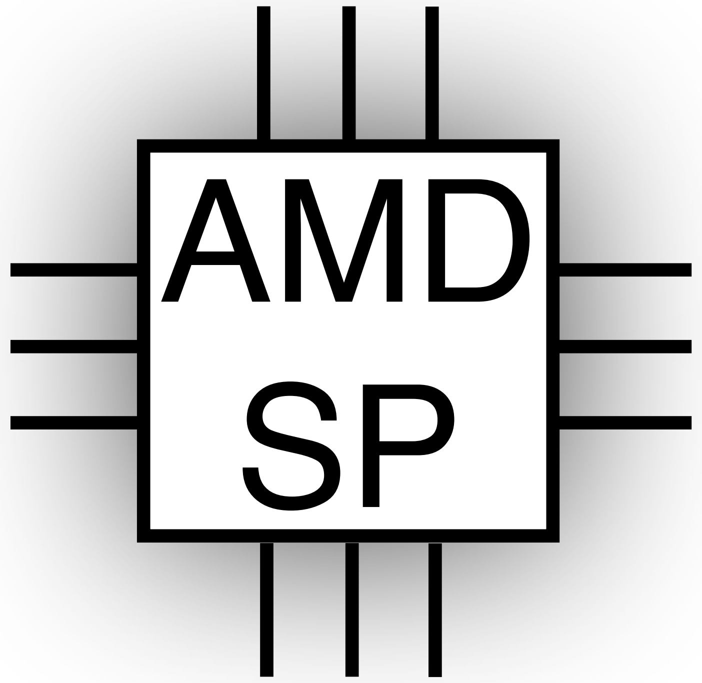
Key A

Key B

Key C

AES-128 Engine

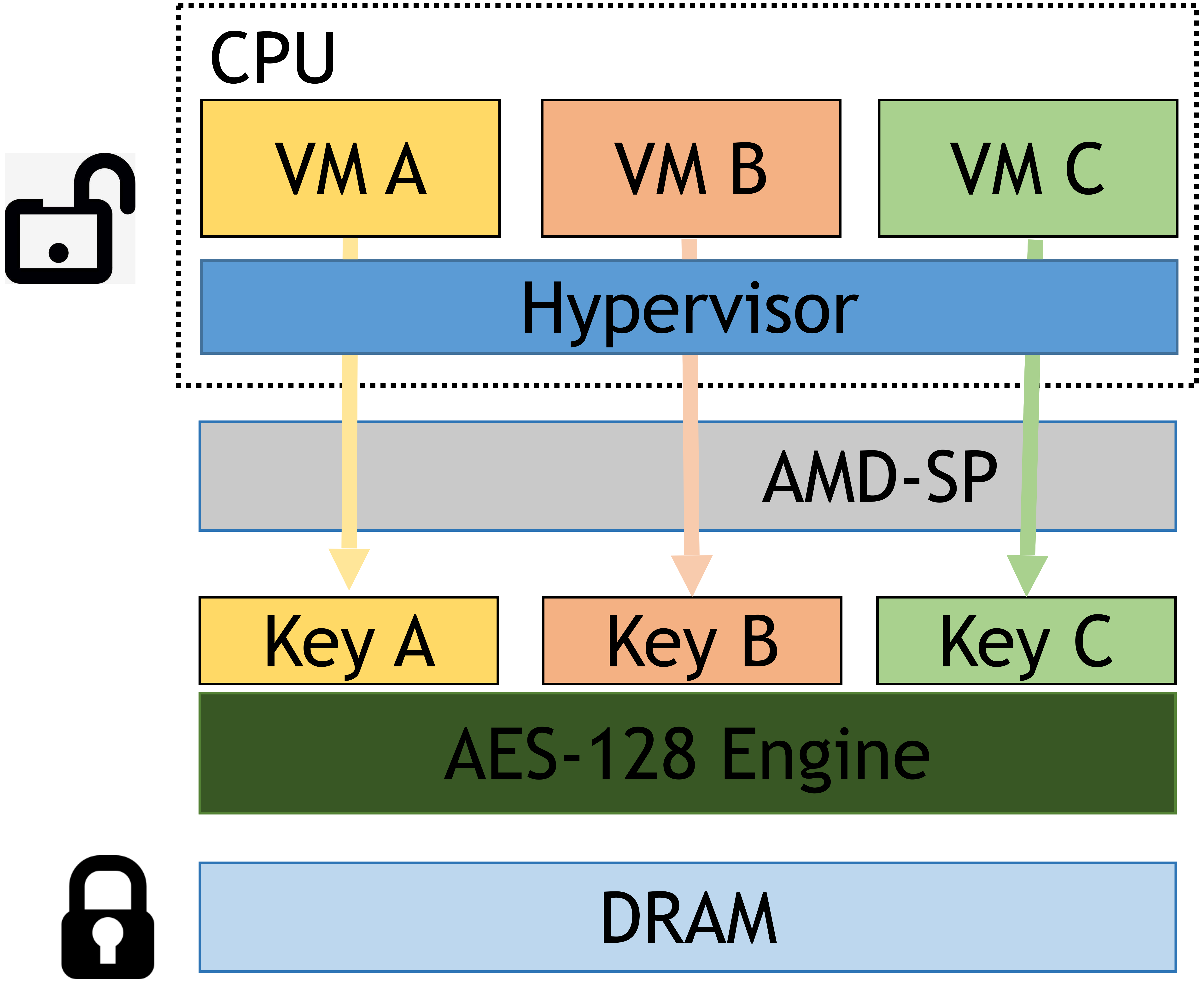
DRAM

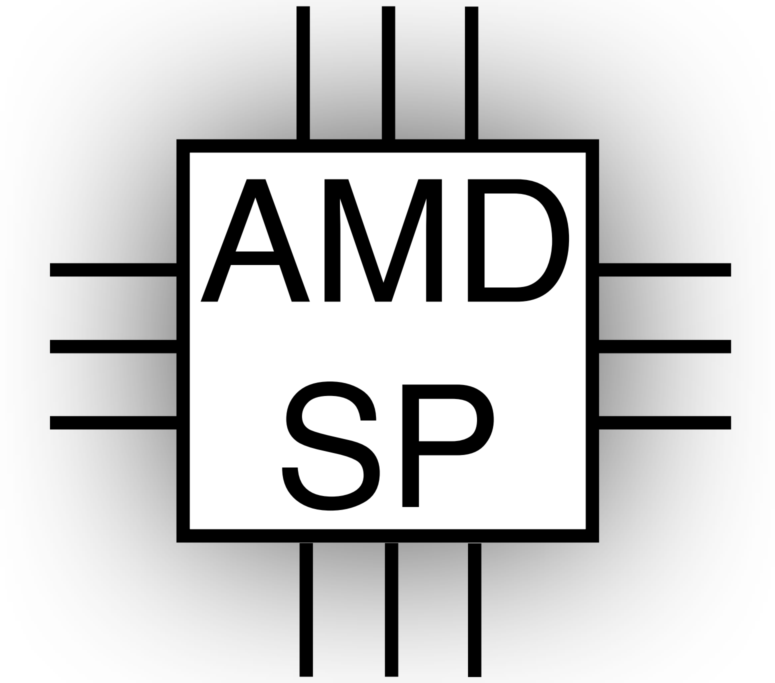


- Each VMs as well as hypervisor have their own and unique AES keys. Those VM Encryption Keys (VEKs) are stored in AMD-SP.

Address Space Identifier (ASID)

ASID’s new role in SEV



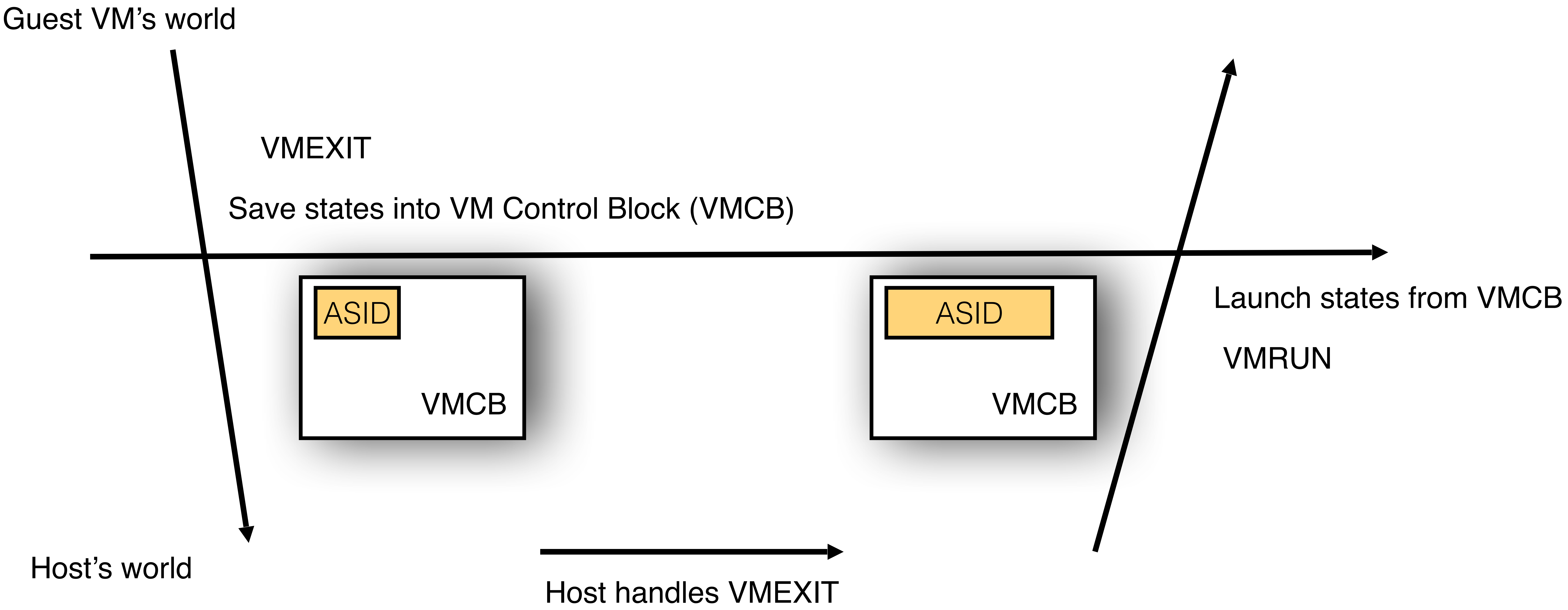


- Each VMs as well as hypervisor have their own and unique AES keys. Those VM Encryption Keys (VEKs) are stored in AMD-SP.

Address Space Identifier (ASID)

- **SEV VM’s vCPUs need to have the same ASID**

ASID-based TLB Isolation in SEV VM



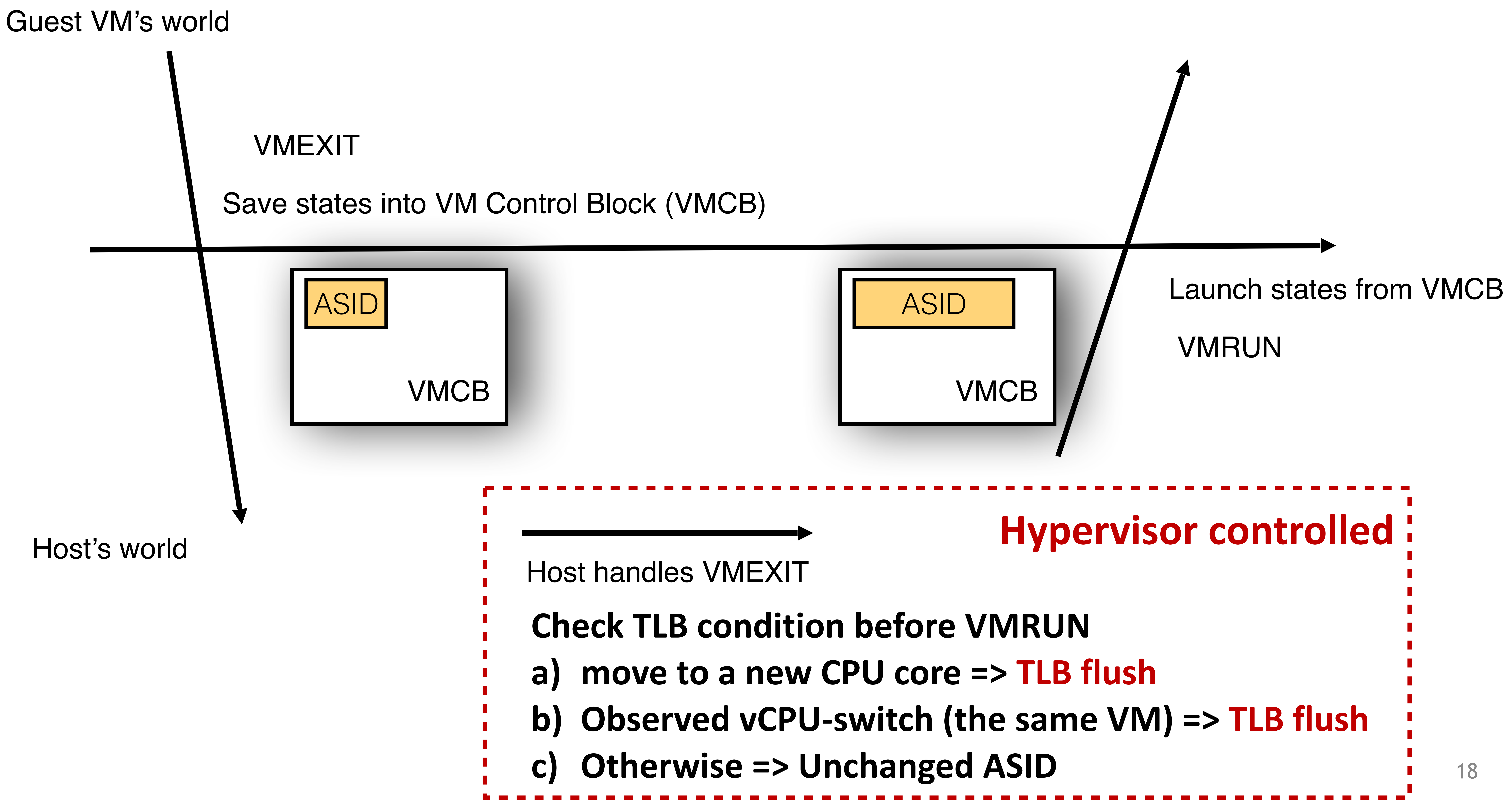
Check TLB condition before VMRUN

a) **move to a new CPU core => TLB flush**

b) **Observed vCPU-switch (the same VM) => TLB flush**

c) **Otherwise => Unchanged ASID**

ASID-based TLB Isolation in SEV VM



Hypervisor controlled

Host handles VMEXIT

Check TLB condition before VMRUN

a) move to a new CPU core => TLB flush

b) Observed vCPU-switch (the same VM) => TLB flush

c) Otherwise => Unchanged ASID

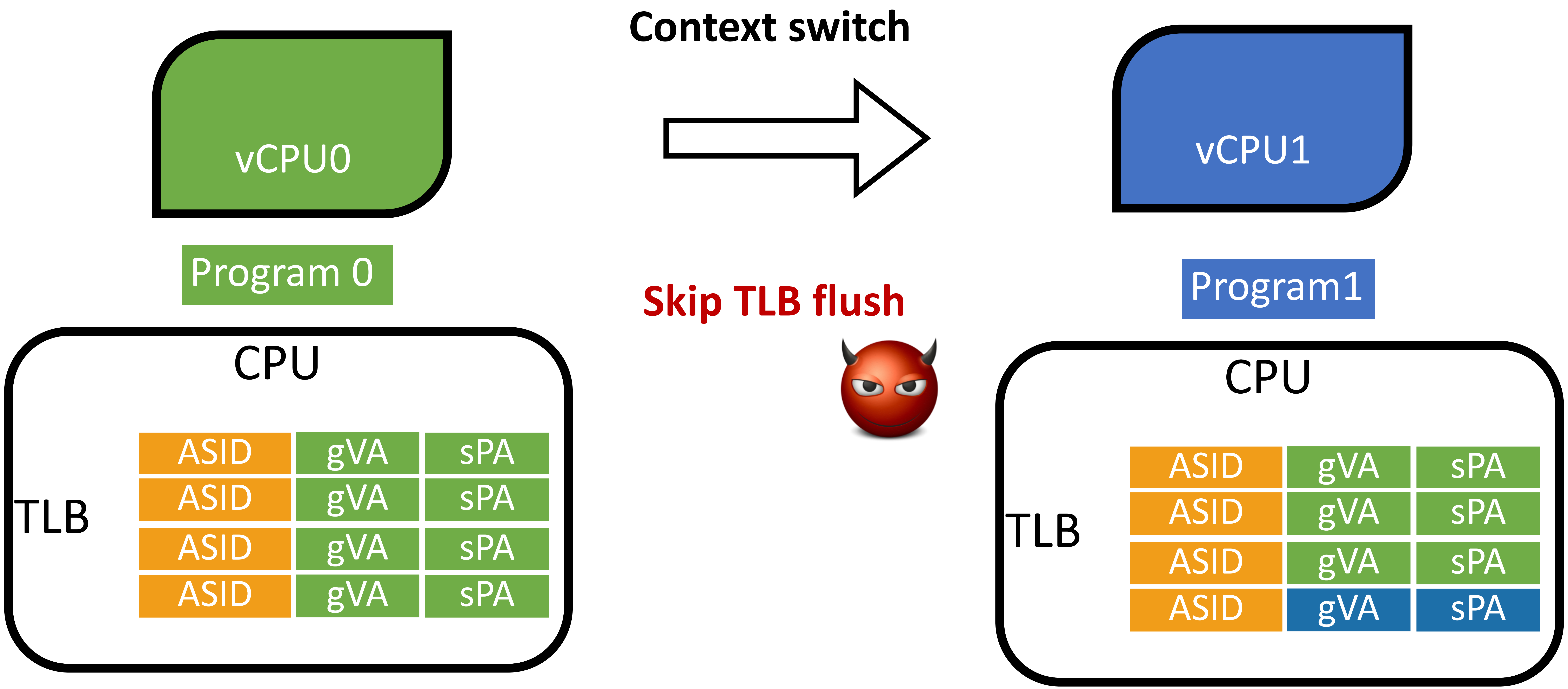
TLB POISONING ATTACKS - OUTLINE

- Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU
- TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process
- Discussion
- Conclusion

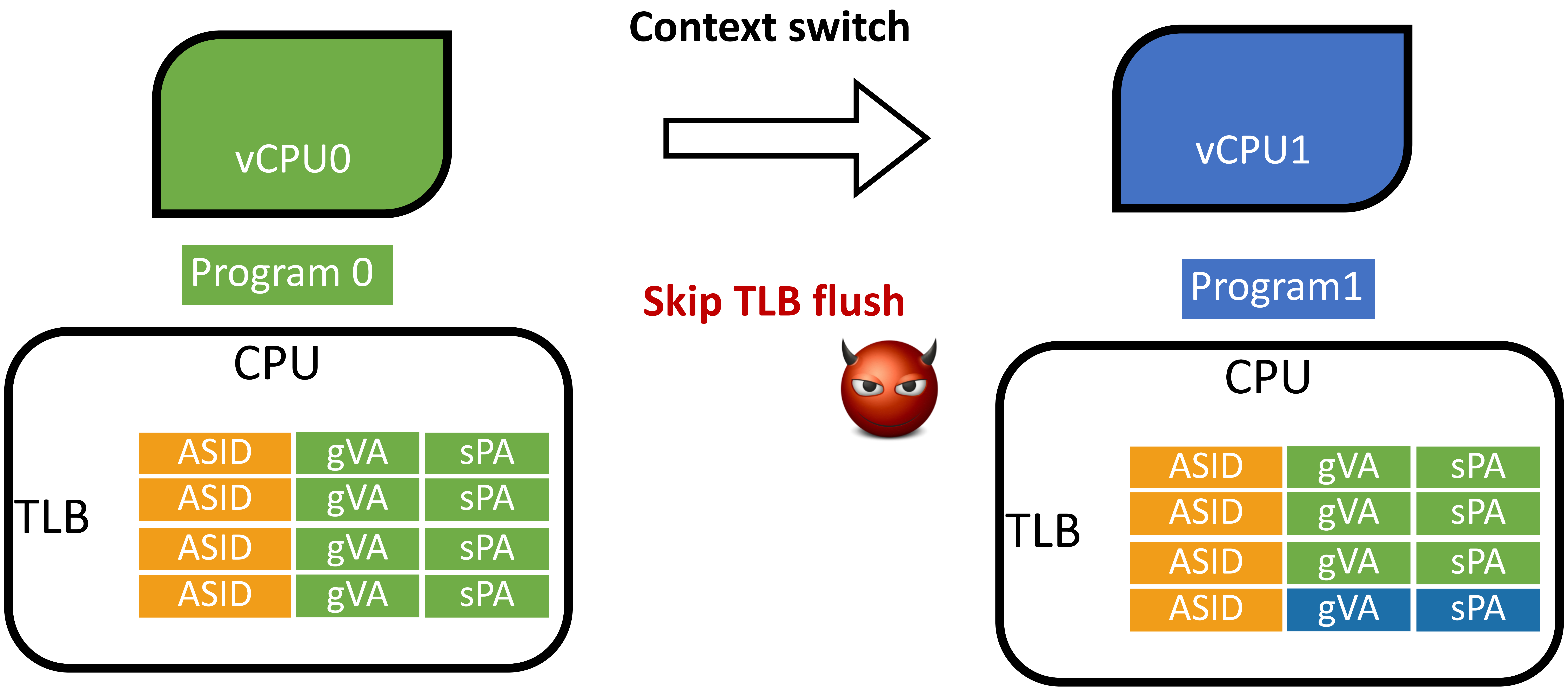
TLB POISONING ATTACKS - OUTLINE

- Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU
- TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process
- Discussion
- Conclusion

TLB Misuse across vCPUs



TLB Misuse across vCPUs



CPU

TLB

ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA

CPU

TLB

ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA

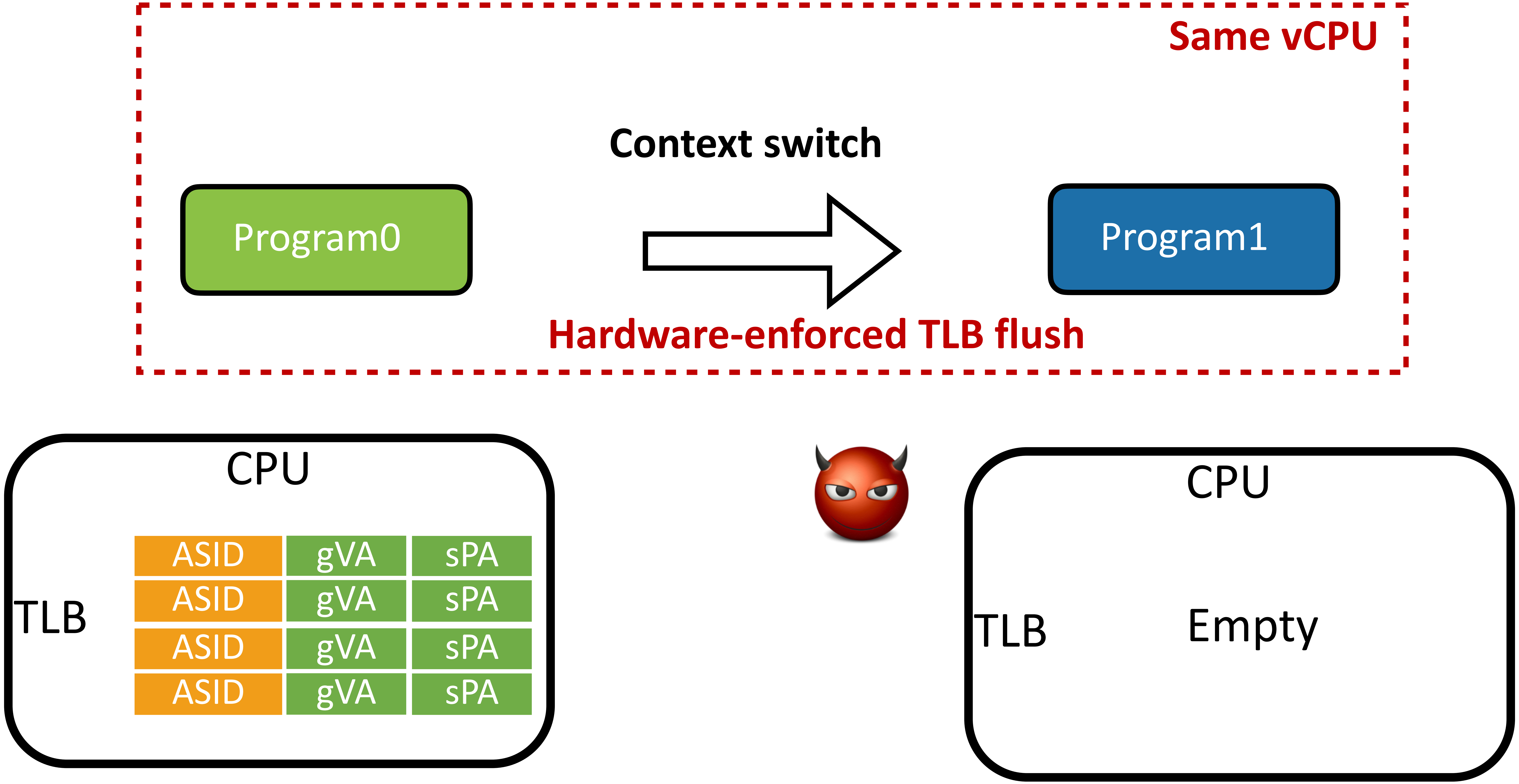
Program 1 can

- **Execute P0’s instruction**
- **Read P0’ data**

TLB POISONING ATTACKS - OUTLINE

- Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU
- TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process
- Discussion
- Conclusion

TLB Misuse within the Same vCPU



Same vCPU

Context switch

Program0

Program1

Hardware-enforced TLB flush

CPU

TLB

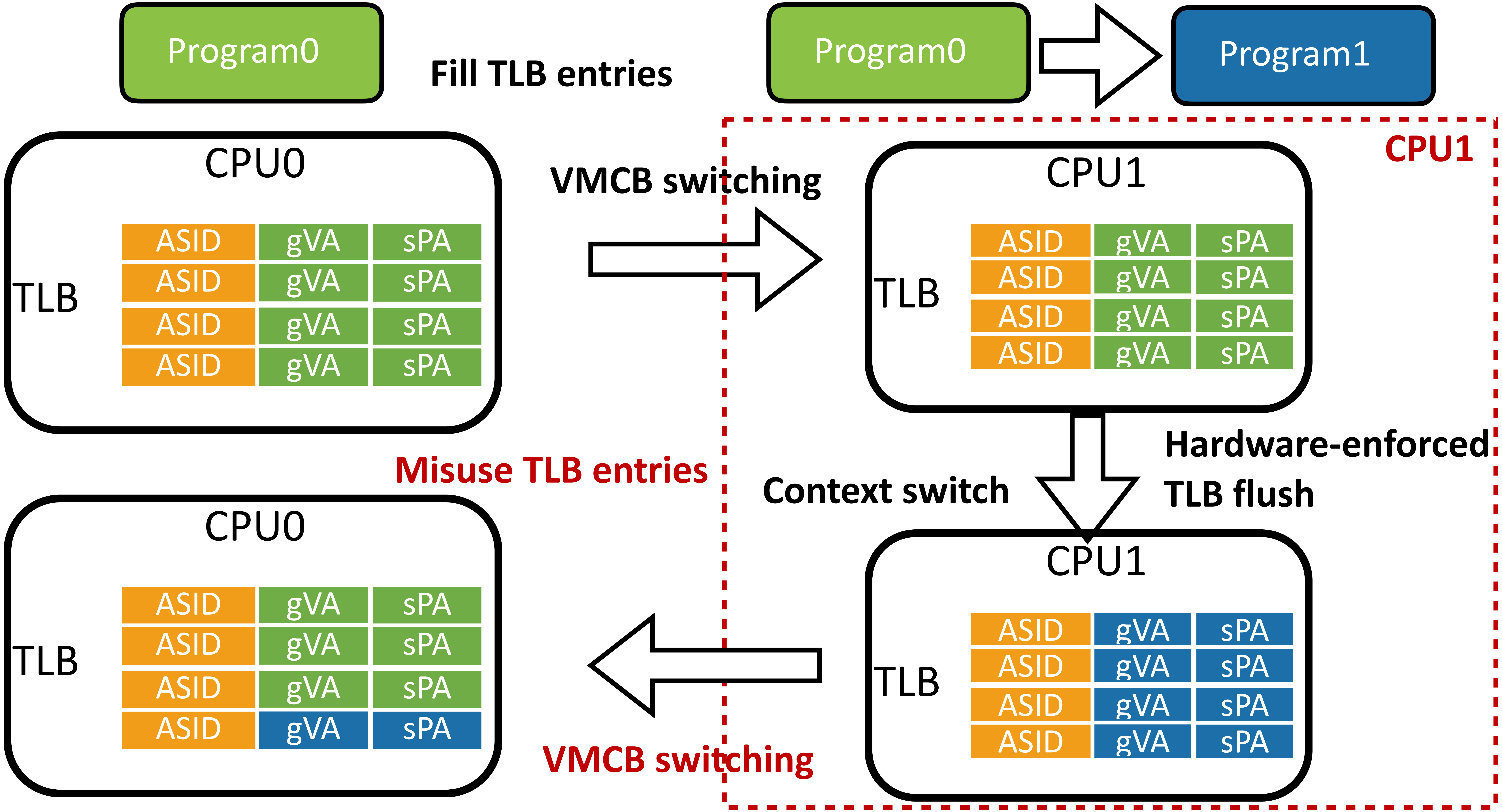
ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA
ASID	gVA	sPA

CPU

TLB

Empty

TLB Misuse within the Same vCPU



Program 1 can

- **Execute P0's instruction**
- **Read P0' data**

TLB POISONING ATTACKS - OUTLINE

- Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU
- TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process
- Discussion
- Conclusion

TLB poisoning with assisting process

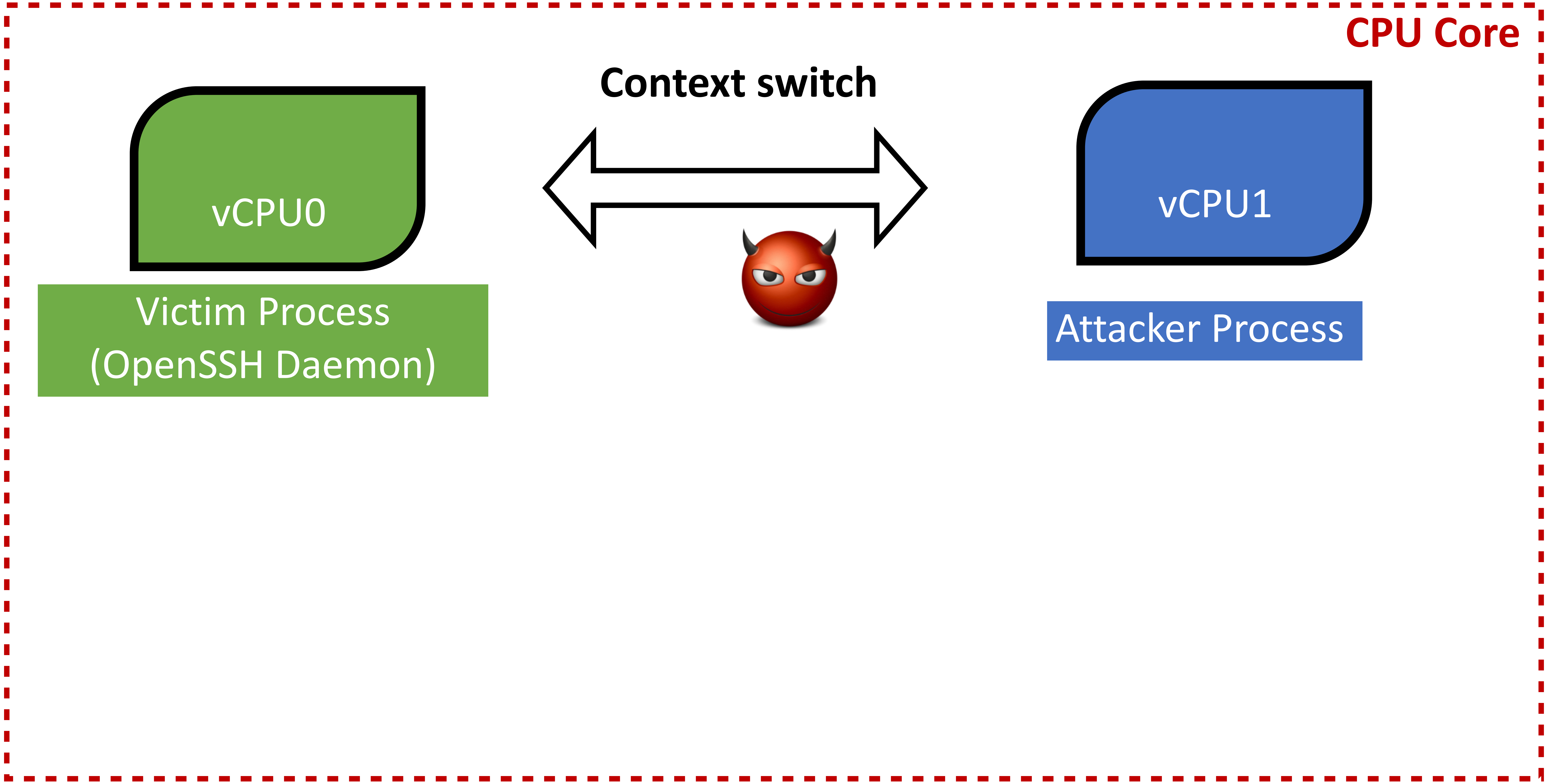
Assume:

- Attacker process inside the VM and controlled by the attacker
- Attacker process is unprivileged
- Attacker process and victim process are on different vCPUs
- Attacker process know victim process's address space (e.g., Crossline attack)

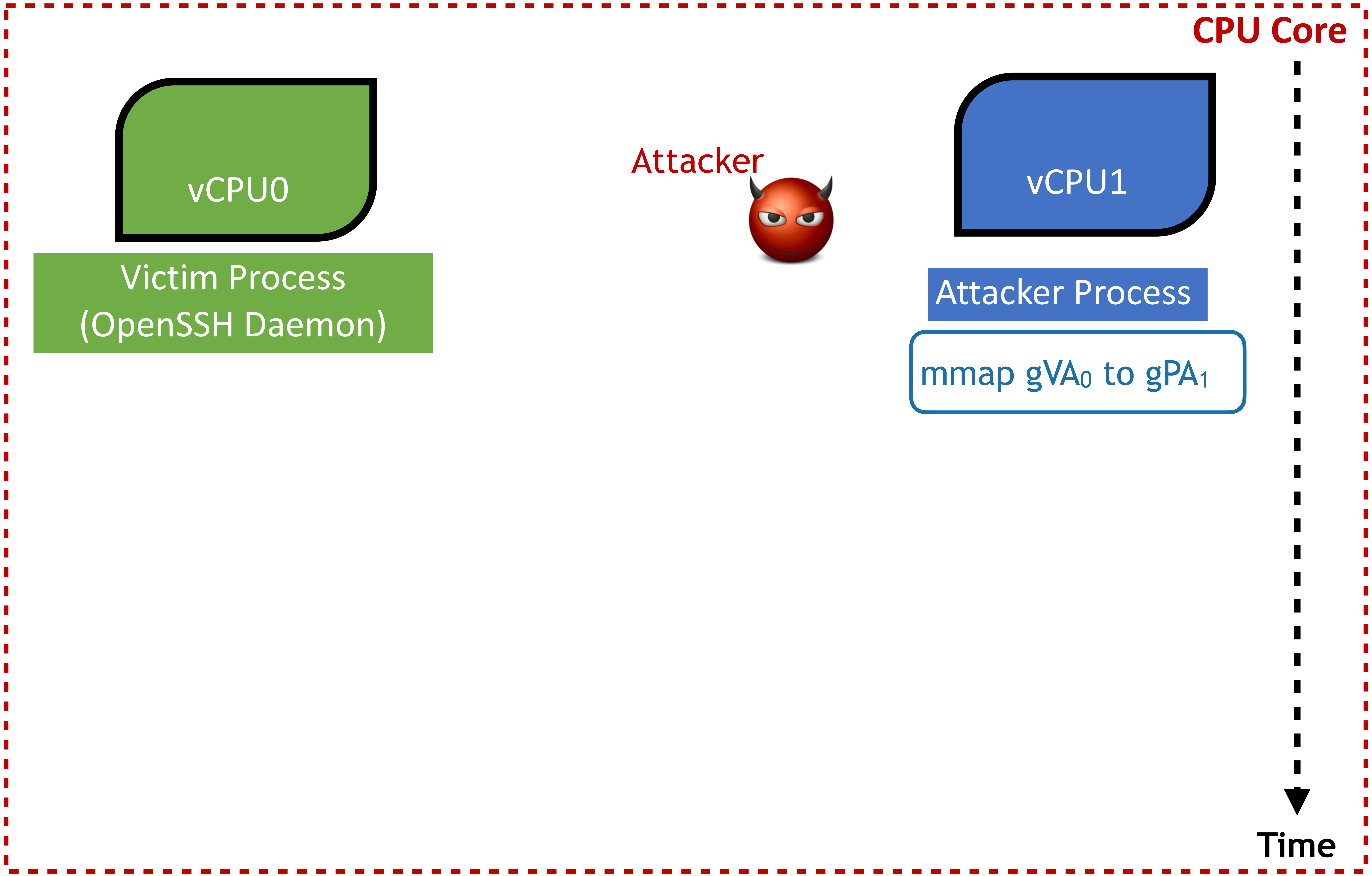
Goal:

- Control privileged process's execution

TLB poisoning with assisting process (Openssh)



TLB poisoning with assisting process (Openssh)



CPU Core

vCPU0

Victim Process (OpenSSH Daemon)

Attacker

vCPU1

Attacker Process

mmap gVA_0 to gPA_1

Time

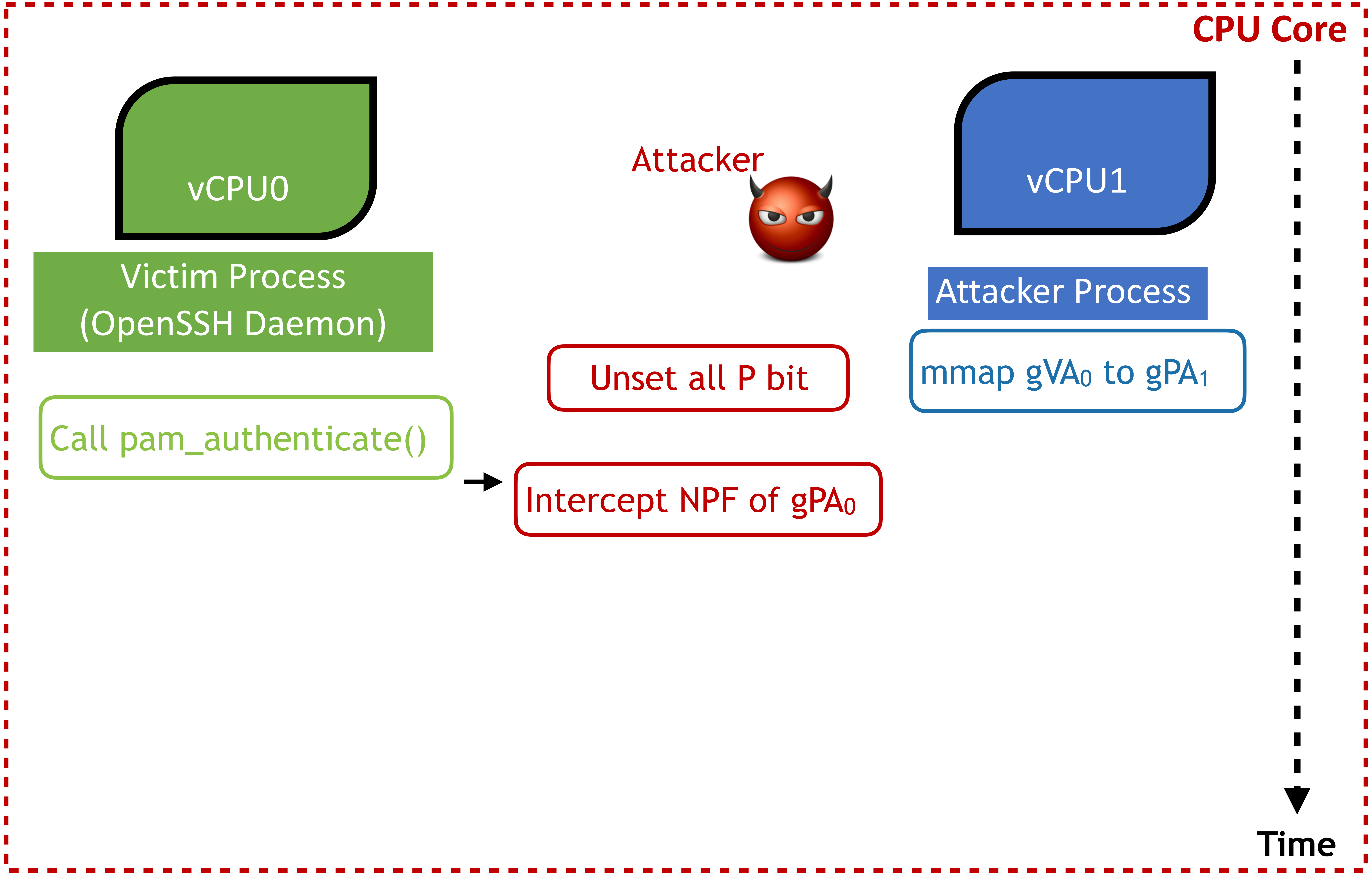
gVA_0: Virtual Address of *pam_authenticate*

gPA_0: Physical Address of *pam_authenticate*

gPA_1: Physical Address of malicious code

Step1:
Creat a mapping
in attacker process's
address space

TLB poisoning with assisting process (Openssh)



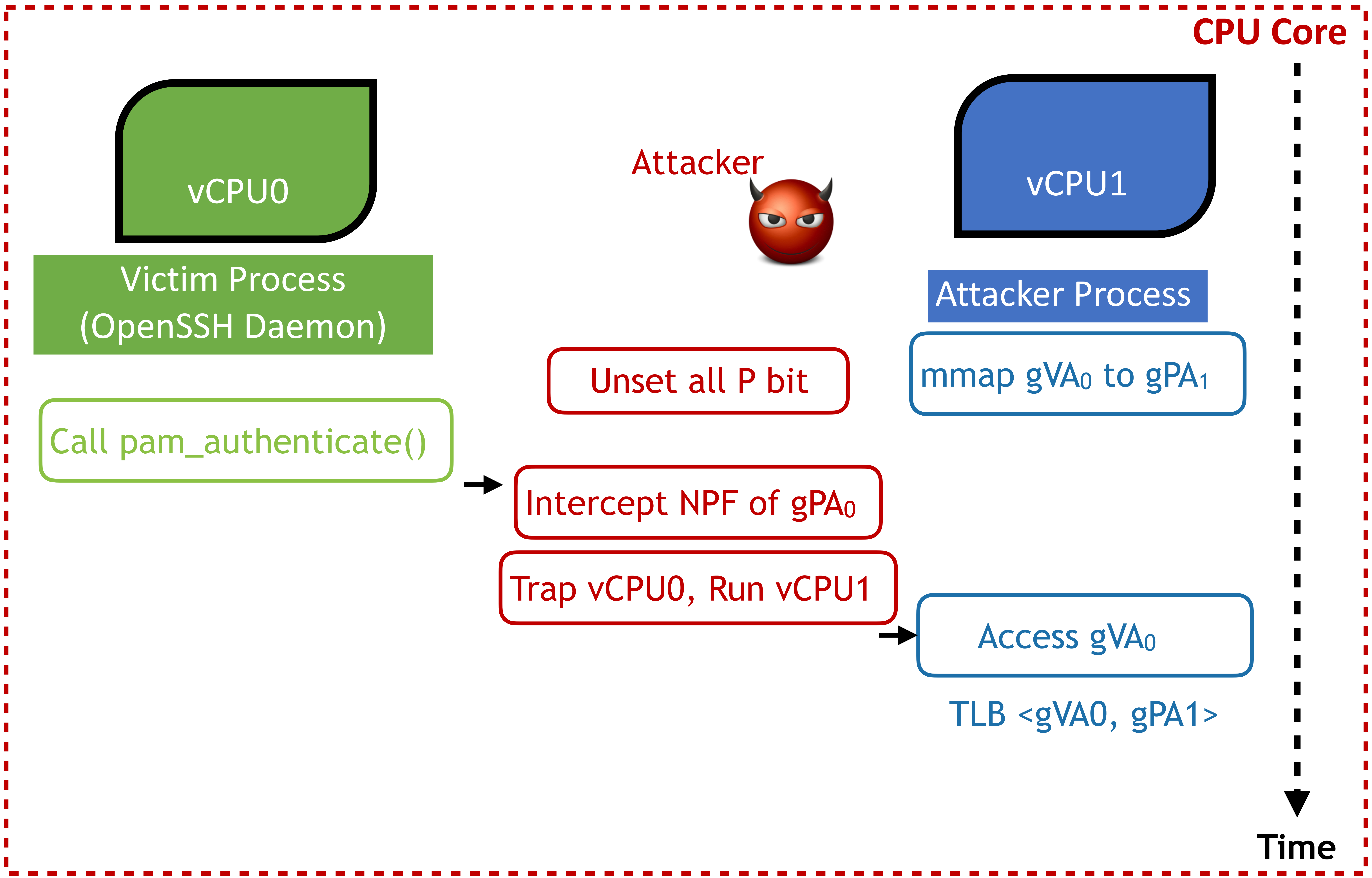
gVA_0: Virtual Address of *pam_authenticate*

gPA_0: Physical Address of *pam_authenticate*

gPA_1: Physical Address of malicious code

Step2:
Unset all P bit, and Intercept target function (*pam_authenticate*)

TLB poisoning with assisting process (Openssh)



CPU Core

vCPU0

Victim Process (OpenSSH Daemon)

Call pam_authenticate()

Attacker

Unset all P bit

Intercept NPF of gPA$_0$

Trap vCPU0, Run vCPU1

vCPU1

Attacker Process

mmap gVA$_0$ to gPA$_1$

Access gVA$_0$

TLB <gVA0, gPA1>

Time

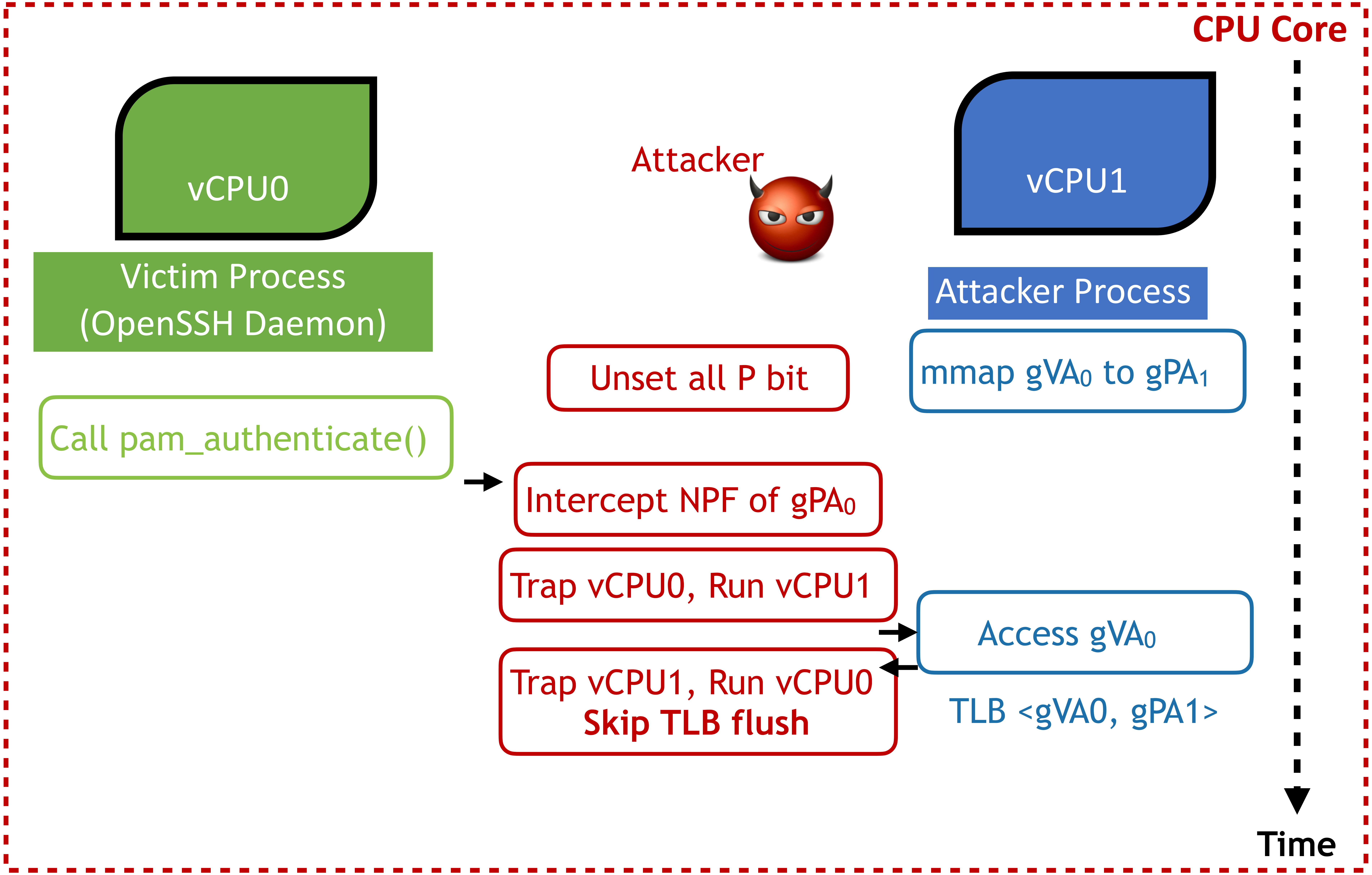
gVA$_0$: Virtual Address of *pam_authenticate*

gPA$_0$: Physical Address of *pam_authenticate*

gPA$_1$: Physical Address of malicious code

Step3:
Poison TLB entries by accessing gVA$_0$

TLB poisoning with assisting process (Openssh)



CPU Core

vCPU0

Victim Process (OpenSSH Daemon)

Attacker

vCPU1

Attacker Process

Unset all P bit

mmap gVA_0 to gPA_1

Call pam_authenticate()

Intercept NPF of gPA_0

Trap vCPU0, Run vCPU1

Access gVA_0

Trap vCPU1, Run vCPU0
Skip TLB flush

TLB <gVA0, gPA1>

Time

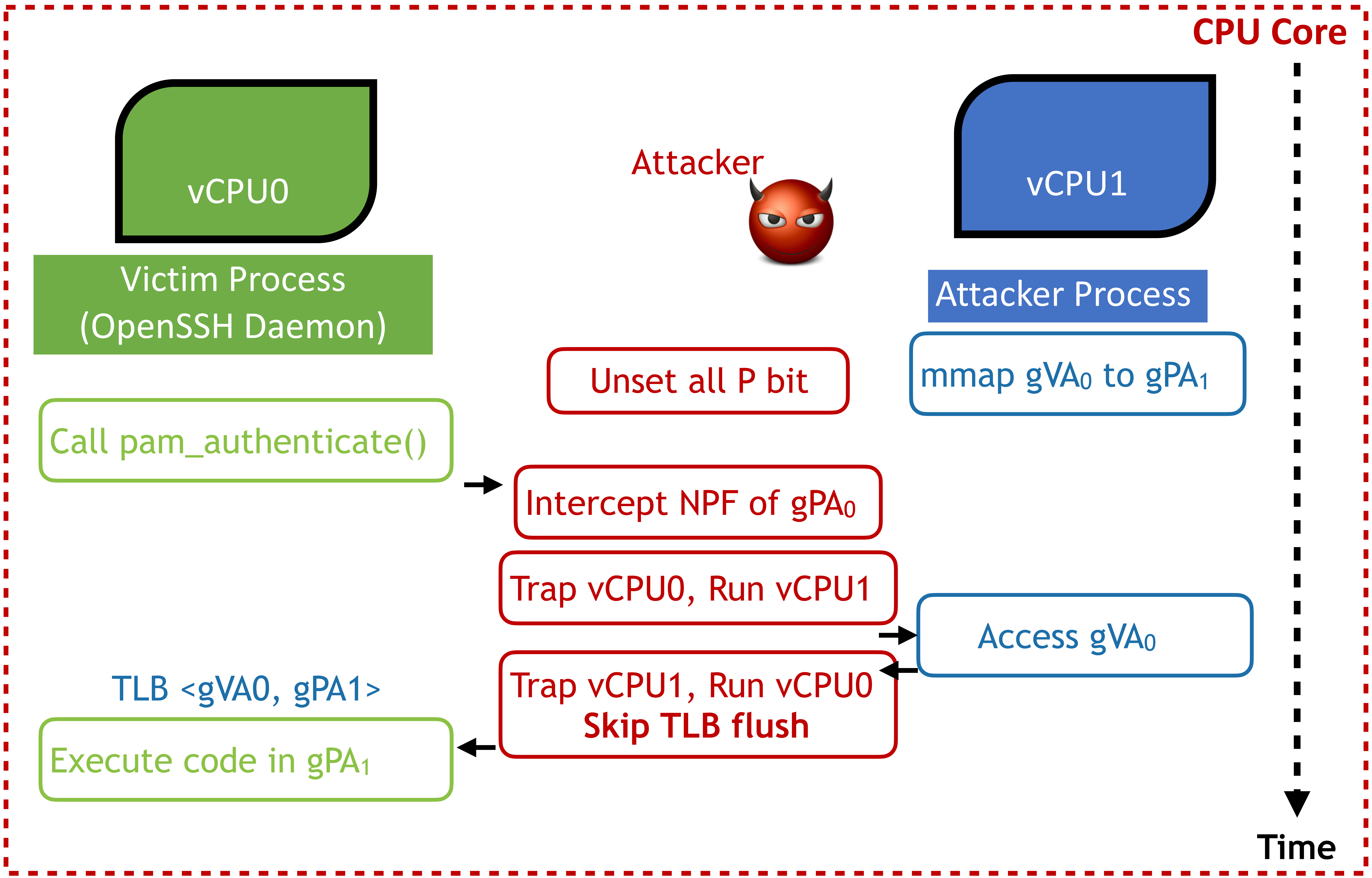
gVA_0: Virtual Address of *pam_authenticate*

gPA_0: Physical Address of *pam_authenticate*

gPA_1: Physical Address of malicious code

Step4:
Skip TLB flush caused by vCPU switching.

TLB poisoning with assisting process (Openssh)



CPU Core

vCPU0

Victim Process (OpenSSH Daemon)

Attacker

vCPU1

Attacker Process

Unset all P bit

mmap gVA_0 to gPA_1

Call pam_authenticate()

Intercept NPF of gPA_0

Trap vCPU0, Run vCPU1

Access gVA_0

TLB <gVA0, gPA1>

Trap vCPU1, Run vCPU0
Skip TLB flush

Execute code in gPA_1

Time

gVA_0: Virtual Address of *pam_authenticate*

gPA_0: Physical Address of *pam_authenticate*

gPA_1: Physical Address of malicious code

Result:
Bypass authentication
(execute arbitrary code)

TLB POISONING ATTACKS - OUTLINE

- Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU
- TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process
- Discussion
- Conclusion

TLB poisoning without assisting process

Network-interface applications:

- Use fork() to serve different requests
- Children processes have similar VMA

Target:

Dropbear SSH: lightweight open-source SSH server

TLB poisoning without assisting process

Network-interface applications:

- Use fork() to serve different requests
- Children processes have similar VMA

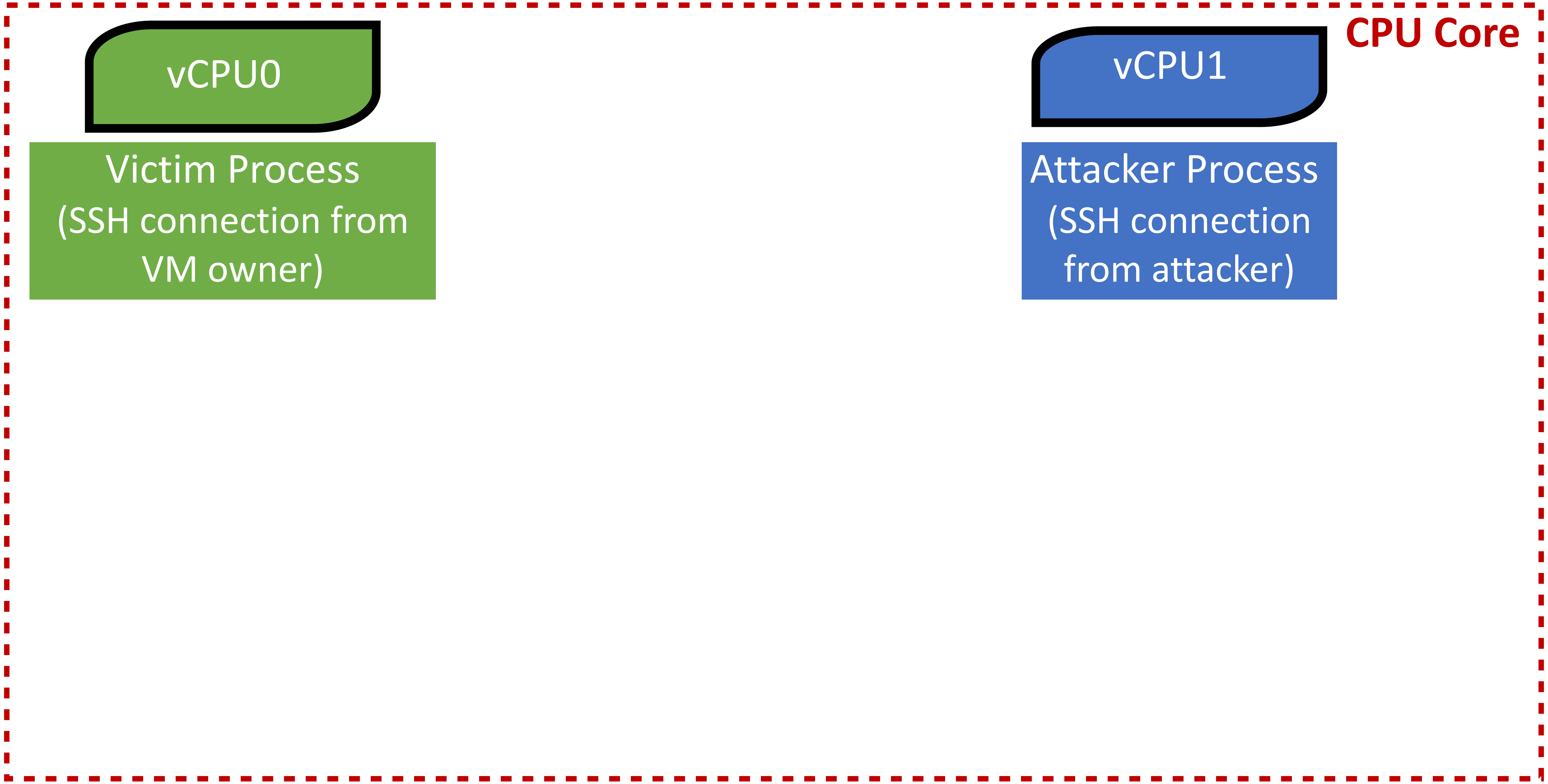
Target:

Dropbear SSH: lightweight open-source SSH server

Goal:

- Bypass password authentication without assisting process when ASLR is enabled

TLB poisoning without assisting process



CPU Core

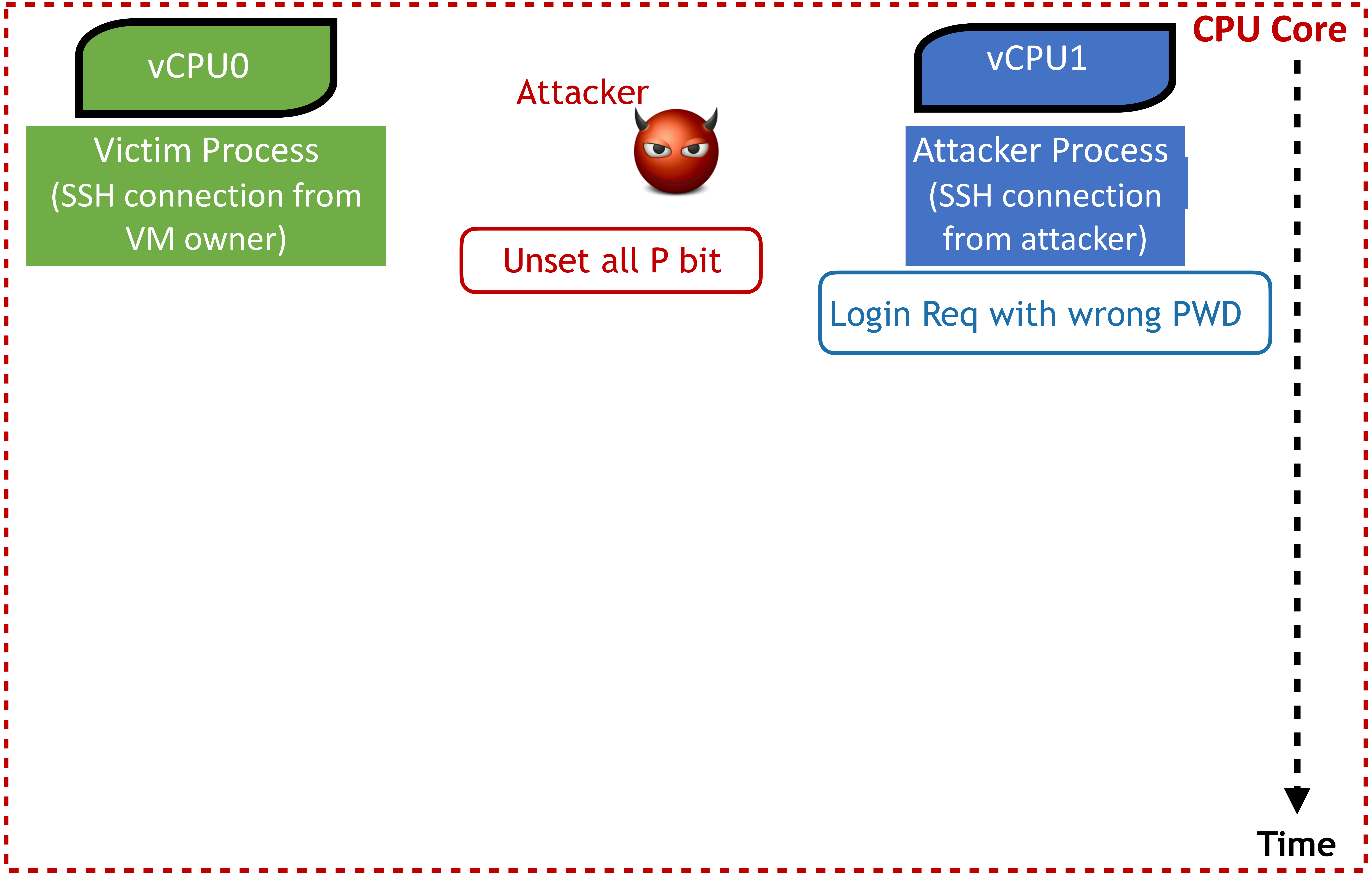
vCPU0

Victim Process (SSH connection from VM owner)

vCPU1

Attacker Process (SSH connection from attacker)

TLB poisoning without assisting process

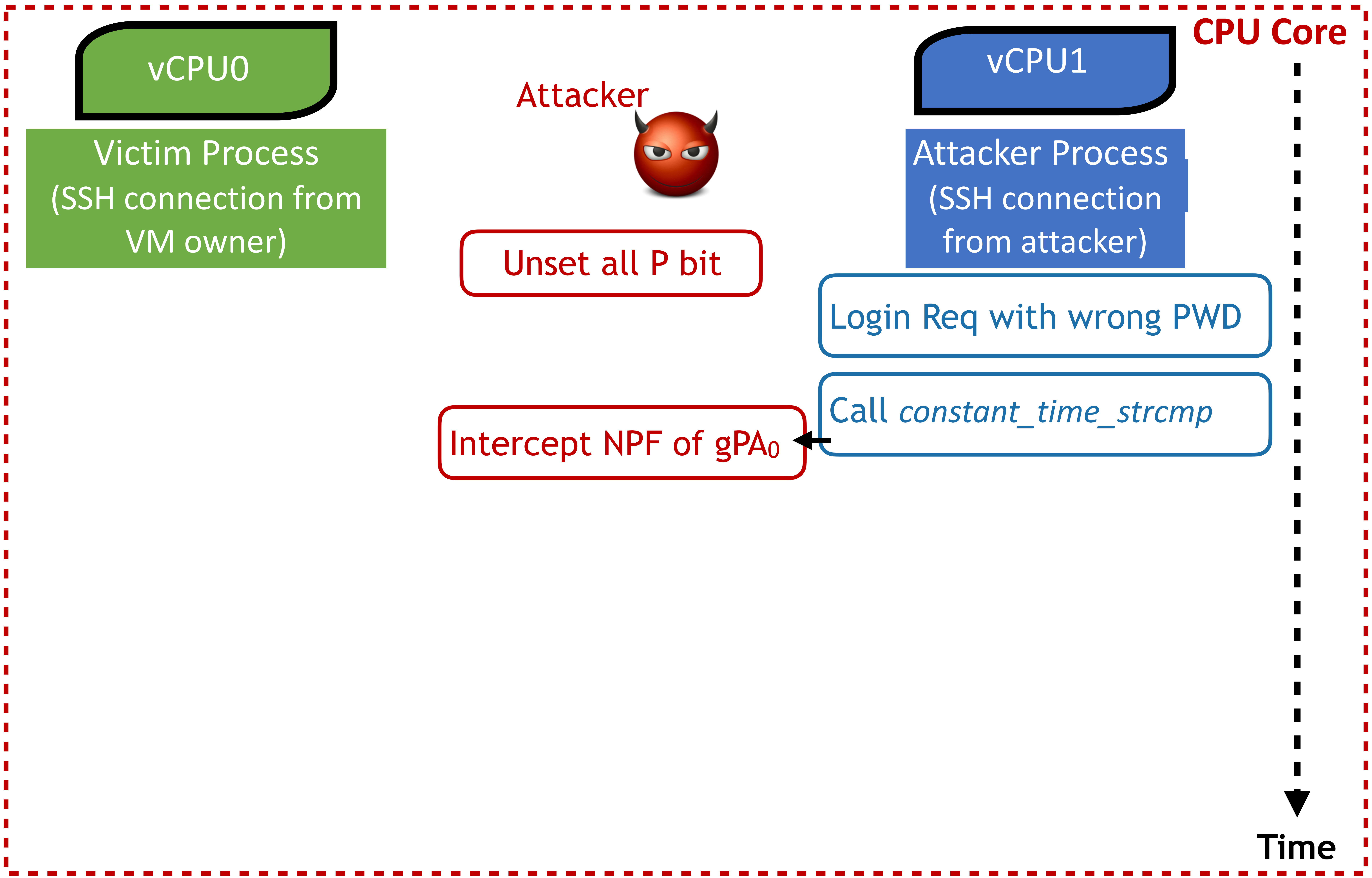


gPA_0: Physical Address of
constant_time_strcmp

gPA_1: Physical Address of
svr_auth_pwd

Step1:
Attacker login with
a wrong PWD

TLB poisoning without assisting process



CPU Core

vCPU0

Victim Process (SSH connection from VM owner)

Attacker

Unset all P bit

Intercept NPF of gPA_0

vCPU1

Attacker Process (SSH connection from attacker)

Login Req with wrong PWD

Call *constant_time_strcmp*

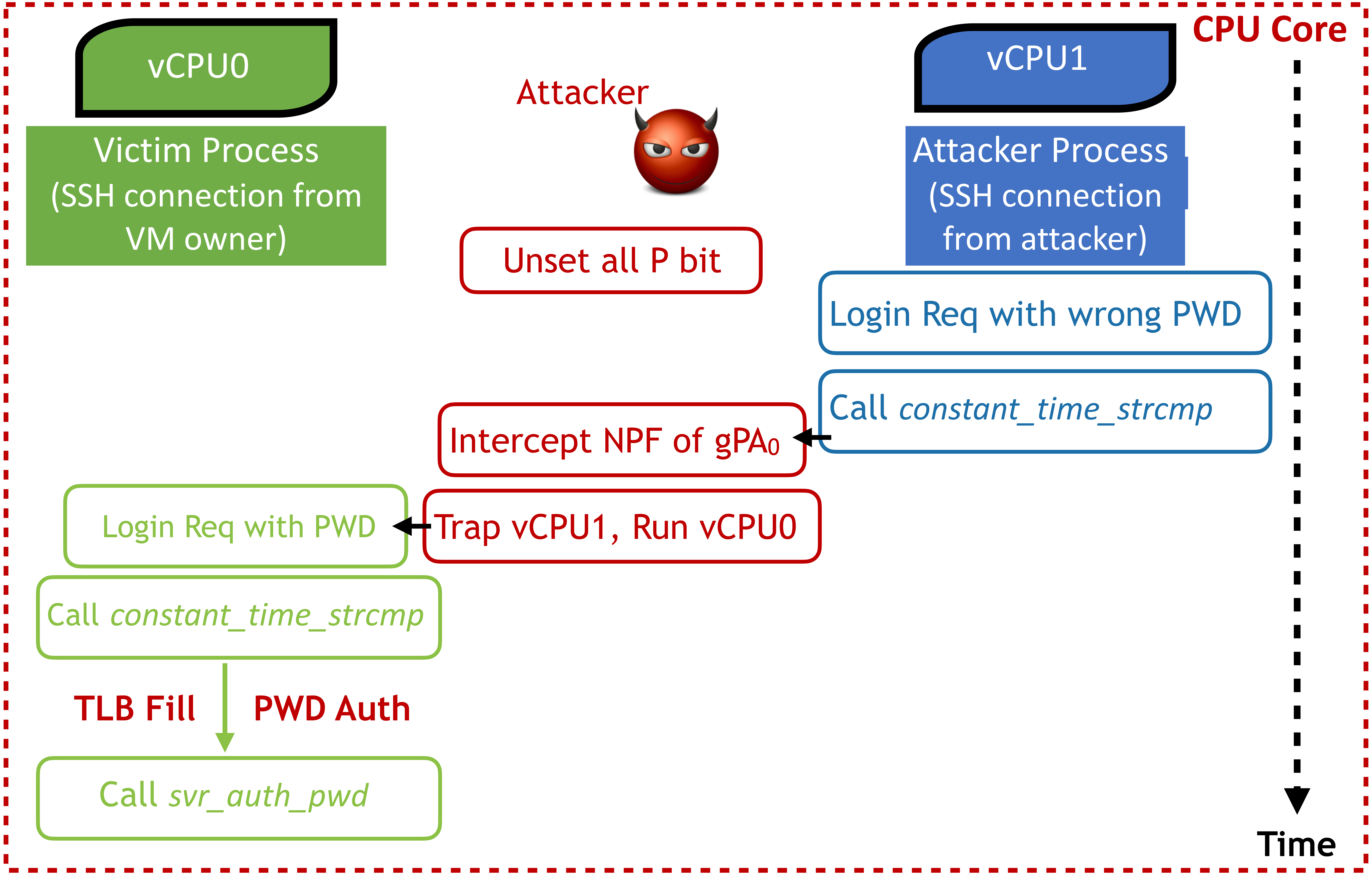
Time

gPA_0: Physical Address of *constant_time_strcmp*

gPA_1: Physical Address of svr_auth_pwd

Step2:
Pause attacker process before PWD authentication

TLB poisoning without assisting process



CPU Core

vCPU0

Victim Process (SSH connection from VM owner)

Attacker

Unset all P bit

vCPU1

Attacker Process (SSH connection from attacker)

Login Req with wrong PWD

Call *constant_time_strcmp*

Intercept NPF of gPA_0

Trap vCPU1, Run vCPU0

Login Req with PWD

Call *constant_time_strcmp*

TLB Fill

PWD Auth

Call *svr_auth_pwd*

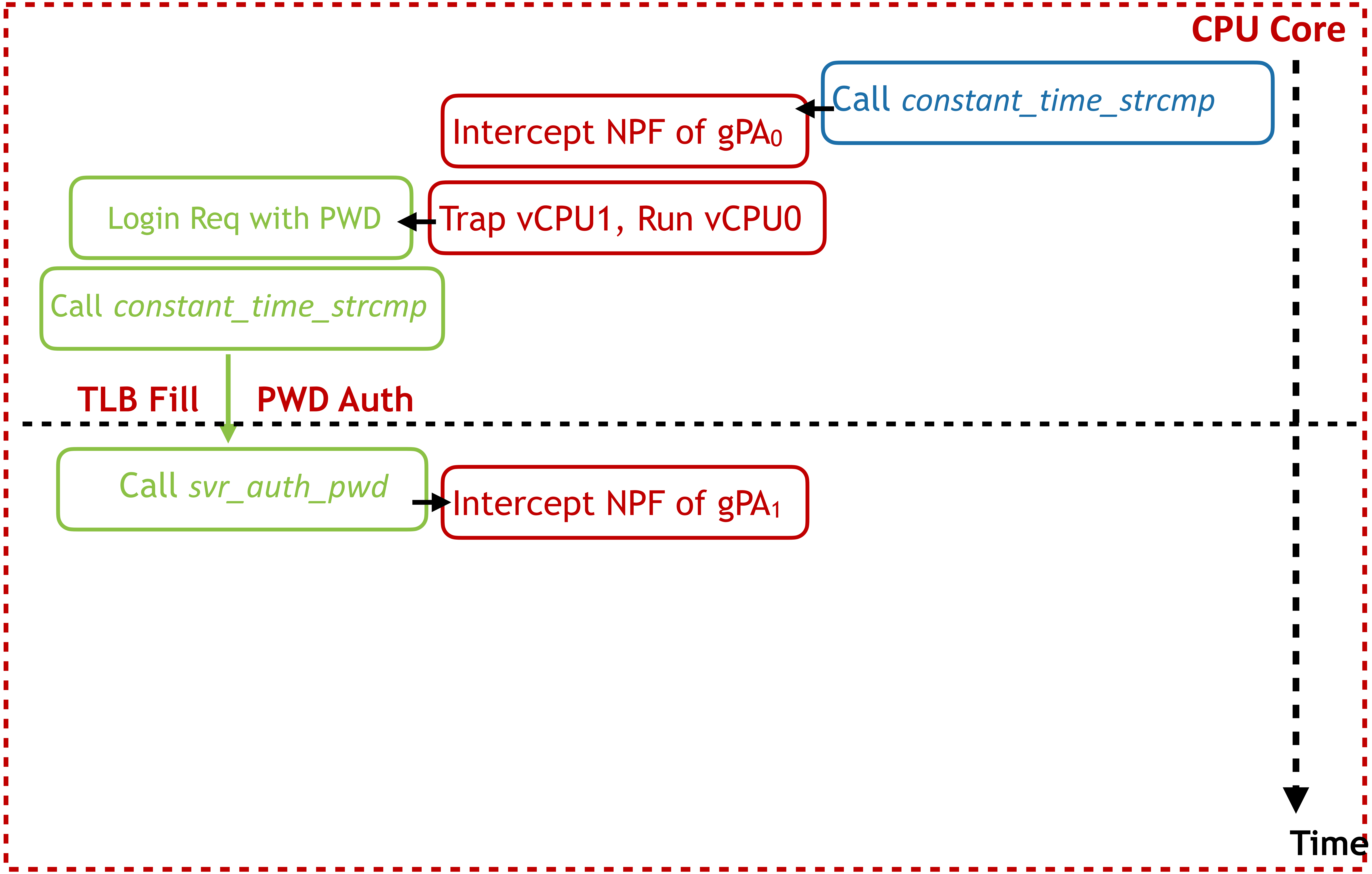
Time

gPA_0: Physical Address of *constant_time_strcmp*

gPA_1: Physical Address of svr_auth_pwd

Step3:
Pause Victim process after Victim process's PWD auth

TLB poisoning without assisting process

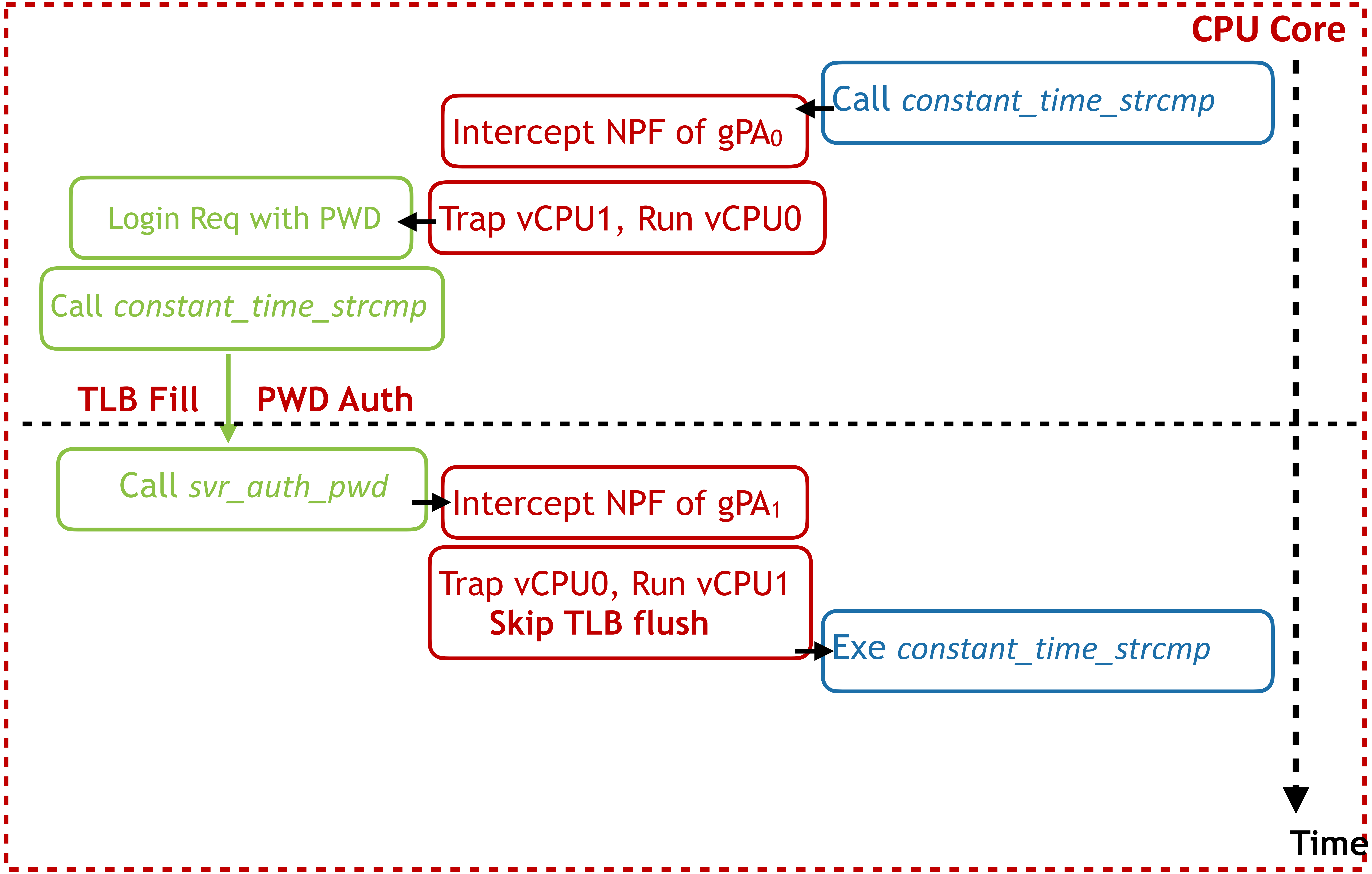


gPA_0: Physical Address of *constant_time_strcmp*

gPA_1: Physical Address of svr_auth_pwd

Step3:
Pause Victim process after Victim process's PWD auth

TLB poisoning without assisting process



CPU Core

Call *constant_time_strcmp*

Intercept NPF of gPA_0

Login Req with PWD

Trap vCPU1, Run vCPU0

Call *constant_time_strcmp*

TLB Fill

PWD Auth

Call *svr_auth_pwd*

Intercept NPF of gPA_1

Trap vCPU0, Run vCPU1
Skip TLB flush

Exe *constant_time_strcmp*

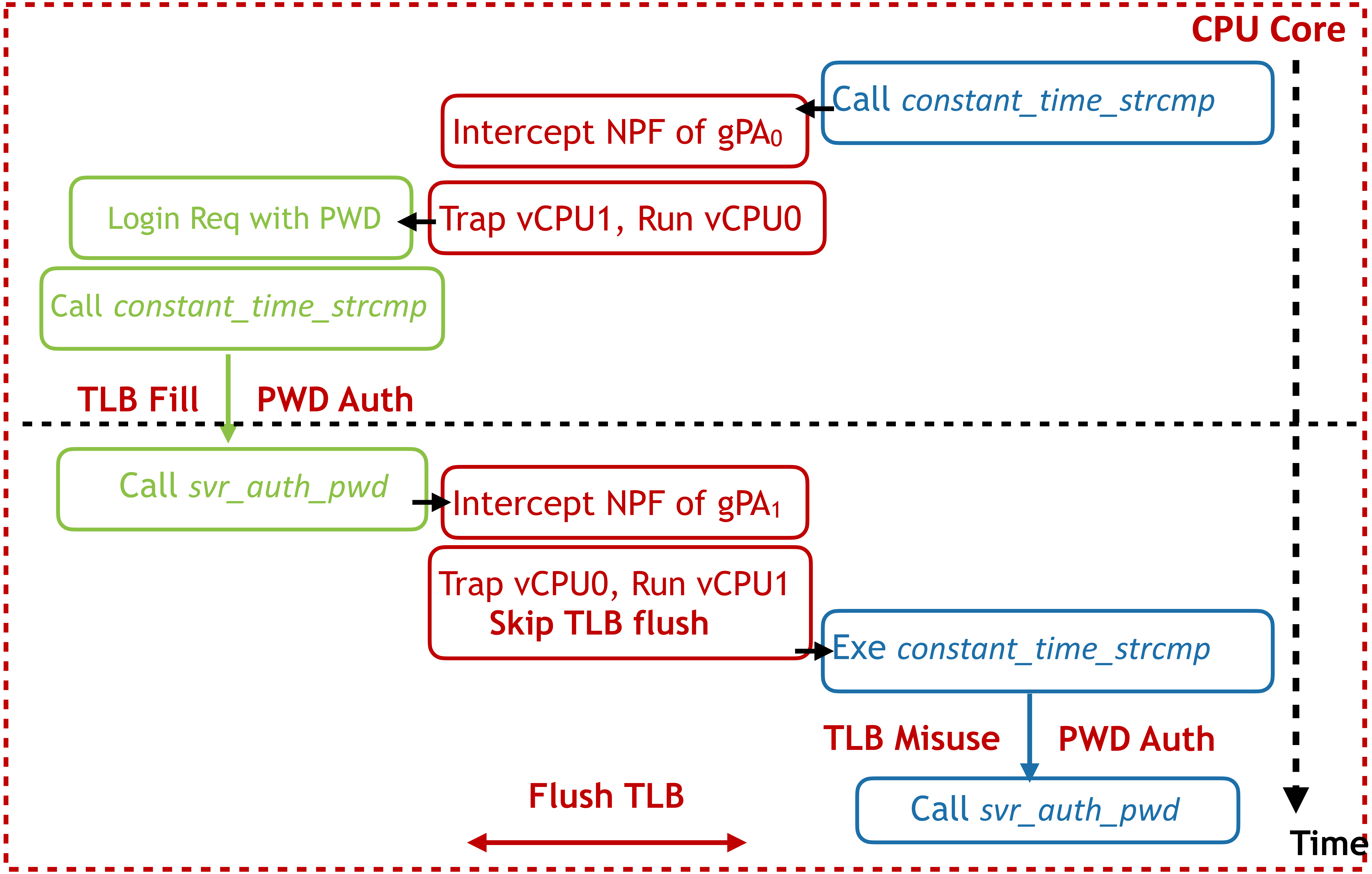
Time

gPA_0: Physical Address of
constant_time_strcmp

gPA_1: Physical Address of
svr_auth_pwd

Step4:
Skip TLB flush caused by vCPU switching and resume Attacker process

TLB poisoning without assisting process

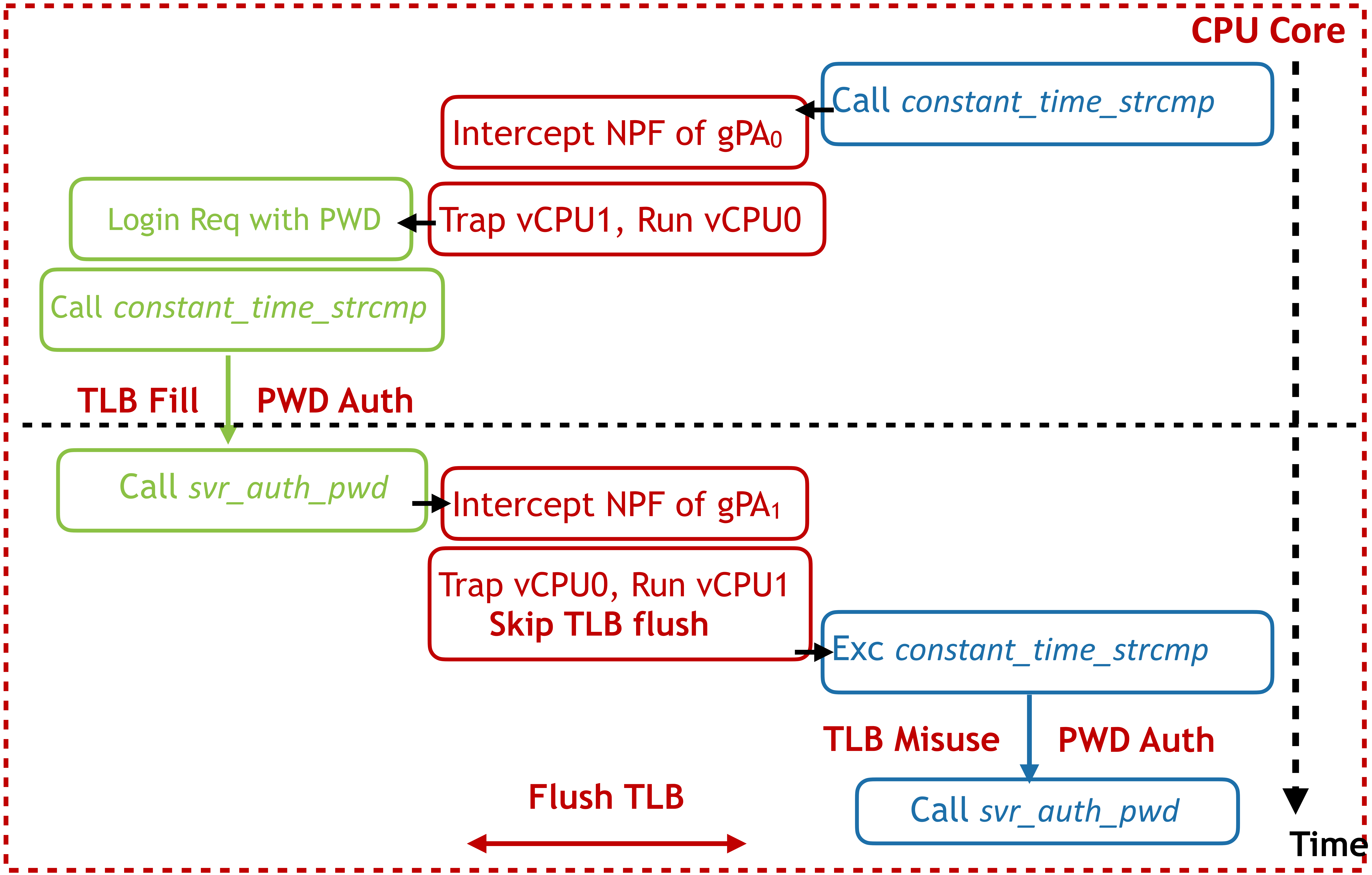


gPA_0: Physical Address of *constant_time_strcmp*

gPA_1: Physical Address of svr_auth_pwd

Step5:
Attacker process pass PWD auth by using victim process's PWD Buffer

TLB poisoning without assisting process



CPU Core

Call *constant_time_strcmp*

Intercept NPF of gPA_0

Trap vCPU1, Run vCPU0

Login Req with PWD

Call *constant_time_strcmp*

TLB Fill

PWD Auth

Call *svr_auth_pwd*

Intercept NPF of gPA_1

Trap vCPU0, Run vCPU1
Skip TLB flush

Exc *constant_time_strcmp*

TLB Misuse

PWD Auth

Call *svr_auth_pwd*

Flush TLB

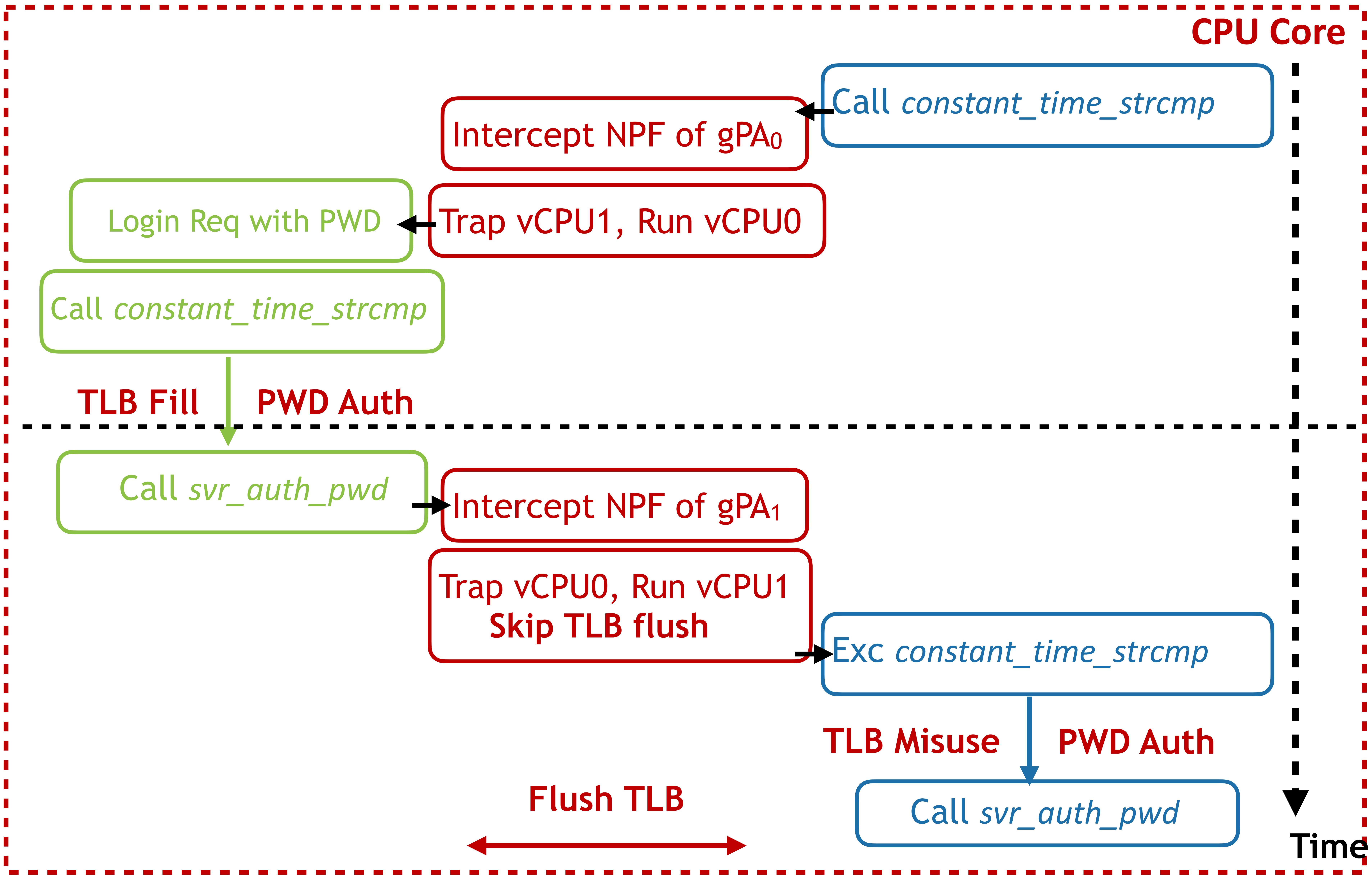
Time

gPA_0: Physical Address of
constant_time_strcmp

gPA_1: Physical Address of
svr_auth_pwd

Step6:
Flush TLB

TLB poisoning without assisting process



gPA_0: Physical Address of
constant_time_strcmp

gPA_1: Physical Address of
svr_auth_pwd

Result:
17 out of 20 connections
Bypass the PWD auth

TLB POISONING ATTACKS - OUTLINE

- Attack Primitives
  - TLB Misuse across vCPUs
  - TLB Misuse within the Same vCPU
- TLB Poisoning Attacks
  - TLB poisoning with assisting process
  - TLB poisoning without assisting process
- Discussion
- Conclusion

Discussion

- **TLB Poisoning attacks on SEV-SNP**
  - SEV-SNP add additional TLB identifier fields in protected VMSA
  - TLB-flush mechanism is now controlled by hardware

Discussion

- **TLB Poisoning attacks on SEV-SNP**
  - SEV-SNP add additional TLB identifier fields in protected VMSA
  - TLB-flush mechanism is now controlled by hardware

- **Countermeasure on SEV/SEV-ES**
  - Network-related application should use exec() to ensure a completely new VMA for different connections (like OpenSSH)

Summary

- This work Demystifies AMD SEV’s mechanism for TLB management
- This work proposes the TLB Poisoning attacks
- This work discusses potential countermeasures

Confidential VMs

AMD

Q & A

Mengyuan Li: *li.7533@osu.edu*