Muen – Design

Inzemamul Haque

25 Nov 2016

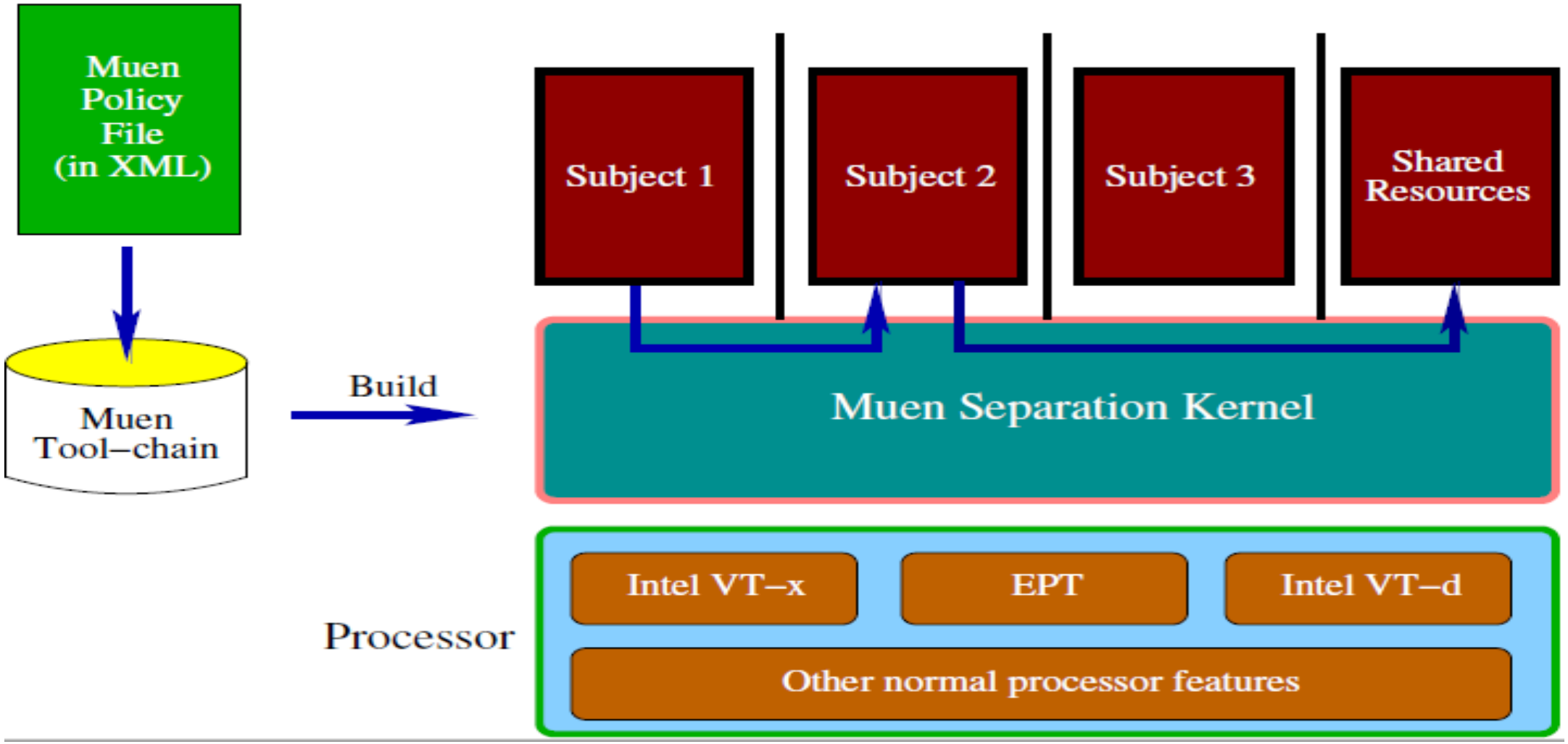
Introduction

- Muen is an open-source separation kernel for x86 platform
- Uses Intel hardware support for virtualization

What Muen does?

- Takes a policy as input and works according to it
- Policy contains information like
  - No. of subjects
  - Information about memory
  - Scheduling policy
  - Communication channels

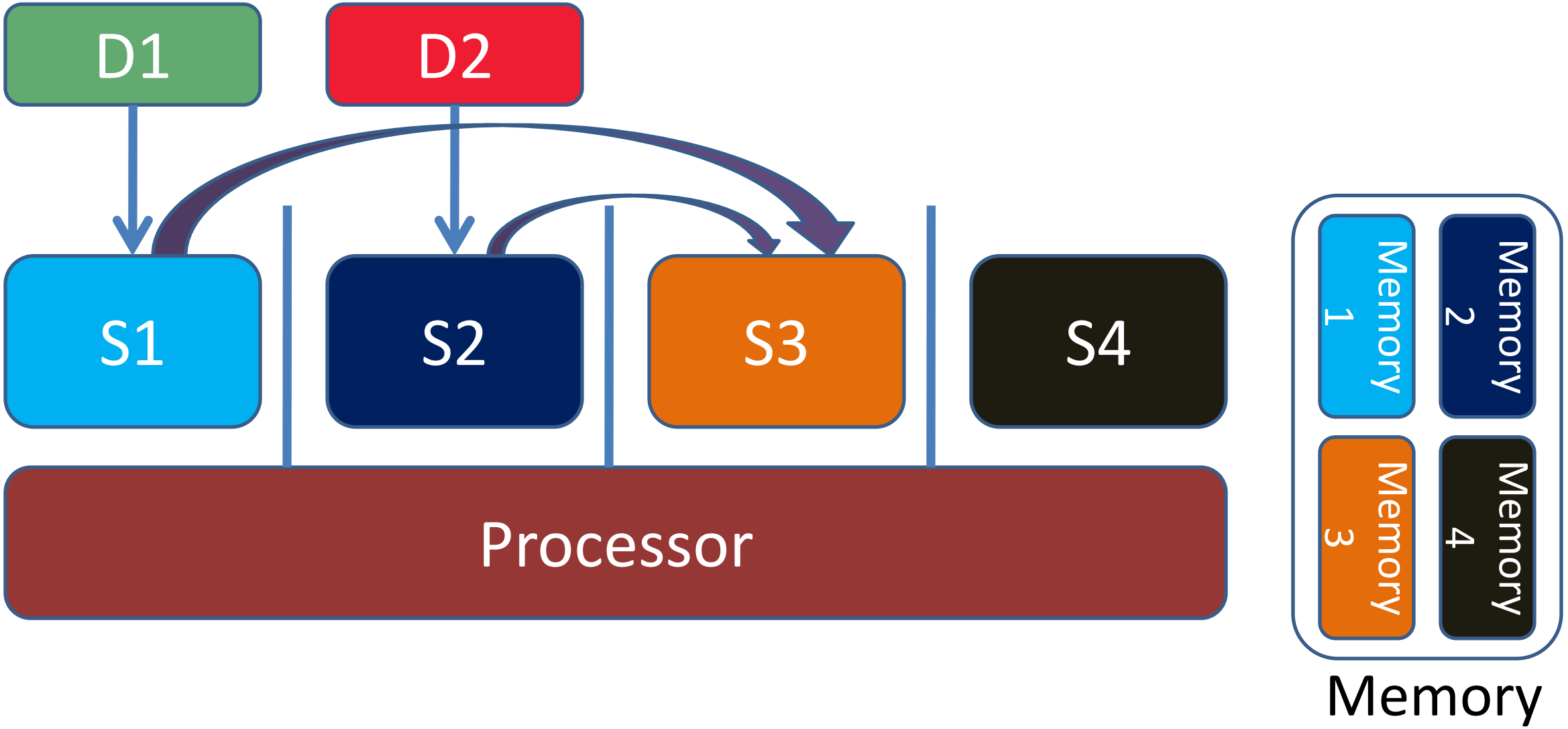
What Muen does?



Example Policy

- 4 subjects, say S1, S2, S3 and S4
- 2 communication channels
  - S1 to S3
  - S2 to S3
- 2 devices D1 and D2 connected to subjects S1 and S2 respectively
- Memory for each subject is of size 512 MB

Example Muen system



Policy

- Contains the following information
  - Memory areas
  - Communication channels
  - Subjects
    - Memory for the subject
    - Devices attached to the subject
    - Channels where it can read or write

Subject

- Abstract view: a full stand-alone machine running a software
- Similar to a virtual machine on a hypervisor
- Can be a bare metal program or an OS
- Also called partition or regime in a separation kernel

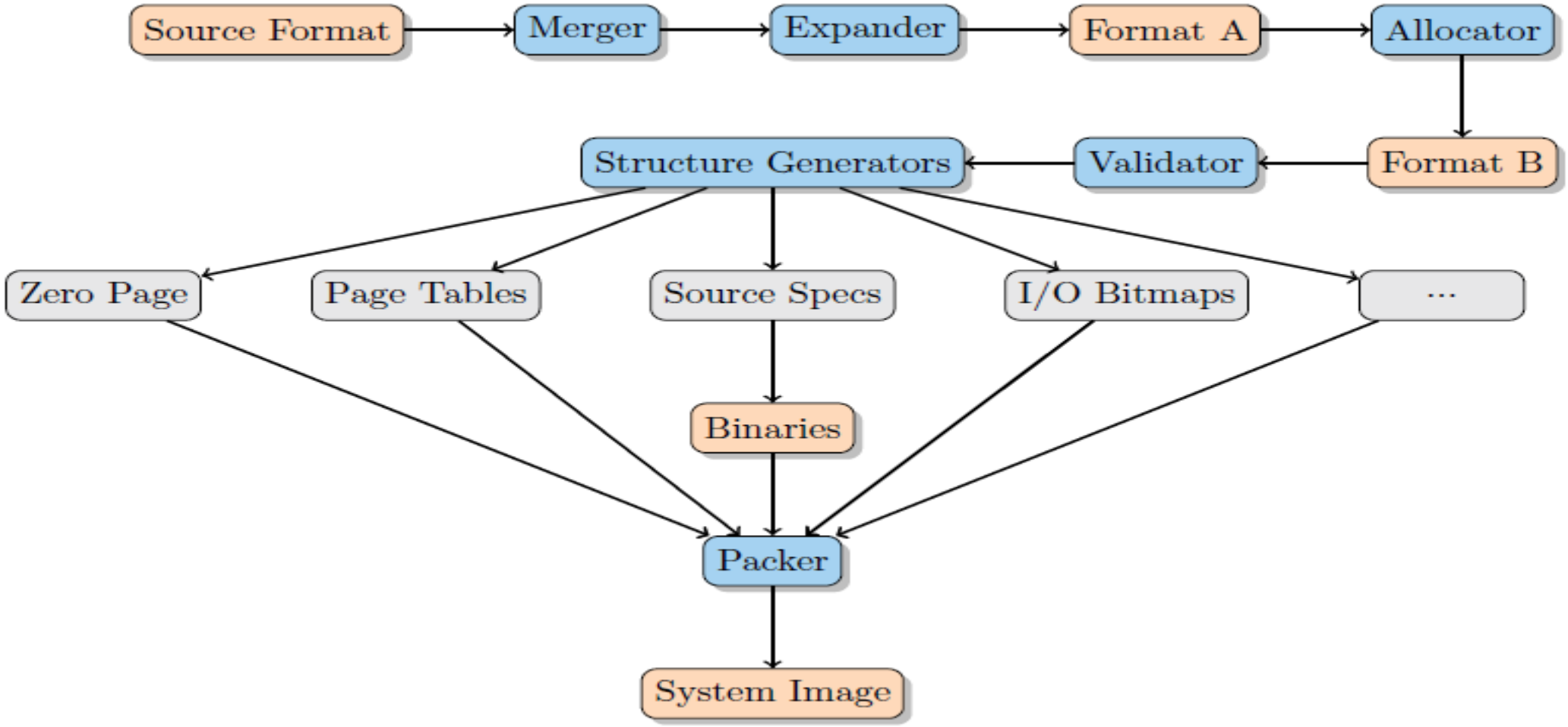
Channels

- Can be either through
  - Shared memory
  - Events
- Complete isolation between subjects except these communication channels
- One-way channels

Scheduling

- Static scheduling
- Round-robin scheduling
- Uses two kinds of frames
  - Minor frames – a subject runs for one minor-frame
  - Major frames – used for synchronization on multiple processors
- 1 Major frame can consist of multiple minor frames

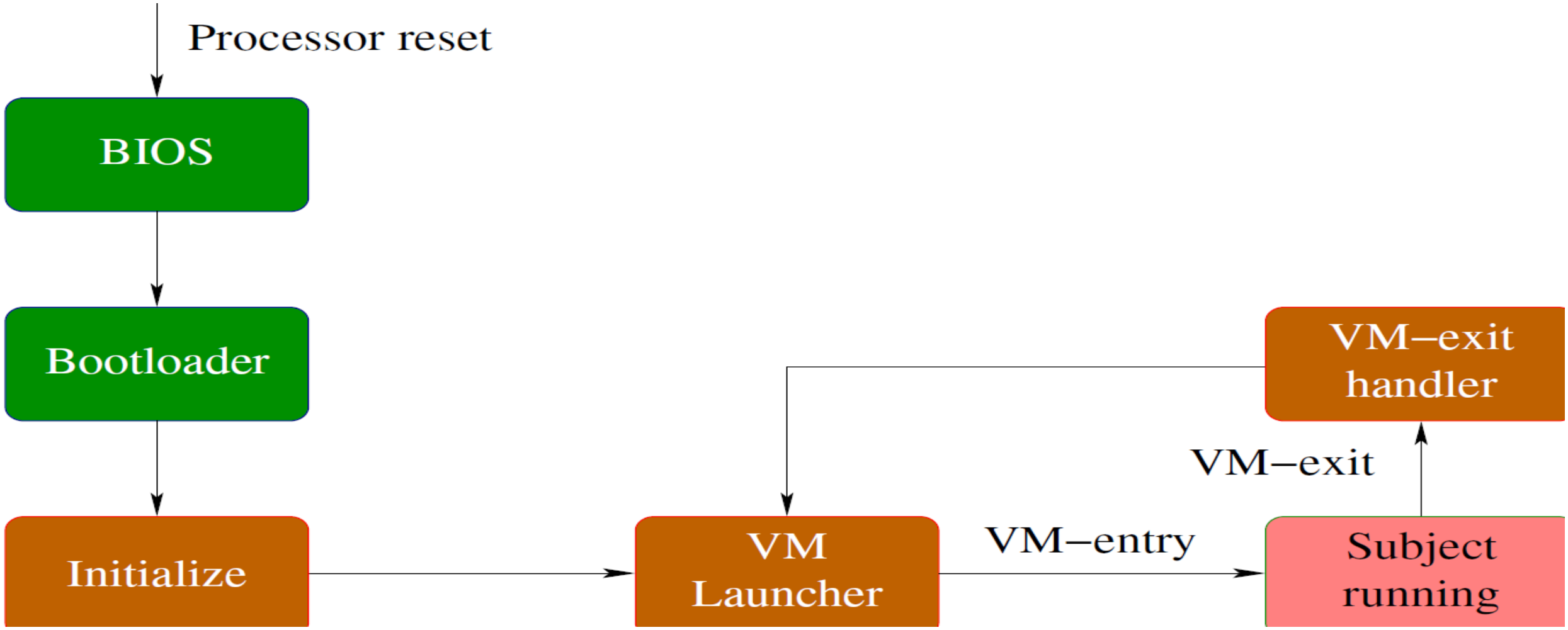
Tool-chain



Source: Muen tool-chain document

Address map

Overview of working of Muen



Initialization

- Set up segmentation and paging
- Set up IDT
- Checking validity of system
- Performing VMXON
- Configure VMCS for each subject
- Initialize scheduler and VMX timer

System after initialization

EPTP
Host state area
RIP:VMexithandlerptr
Guest state area
Interrupt-exiting:1
Exit info
Interrupt info

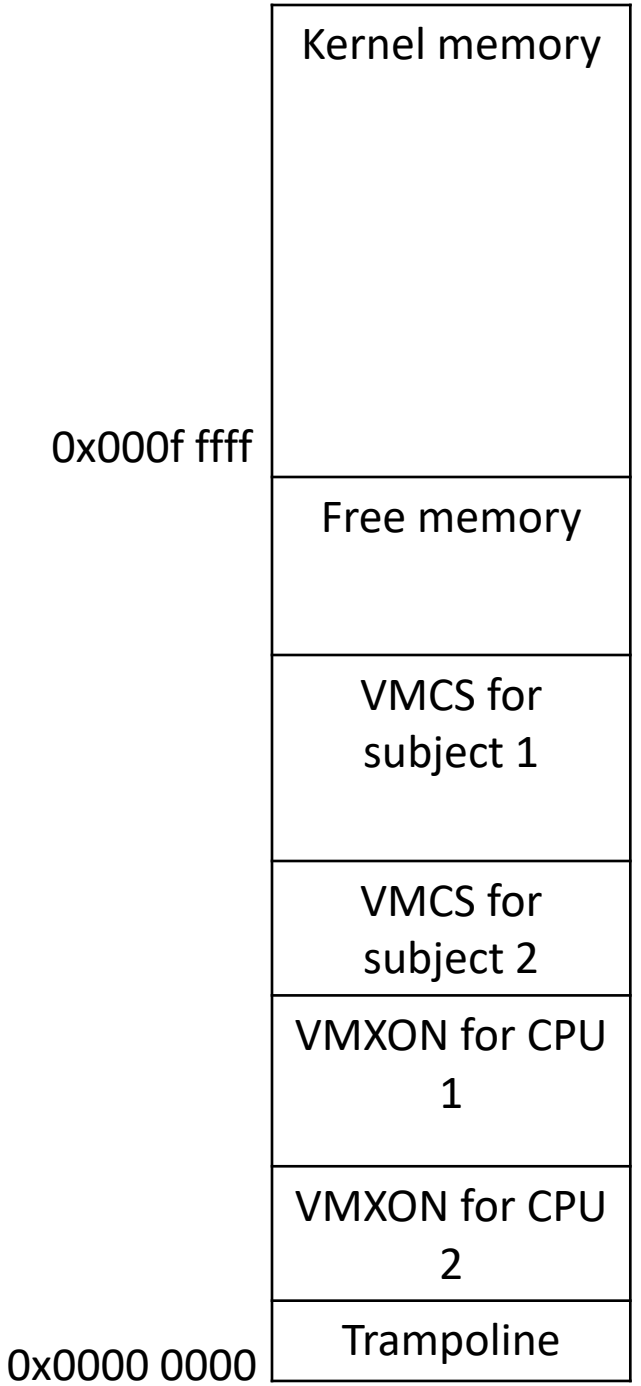
VMCS 1

EPTP
Host state area
RIP:VMexithandlerptr
Guest state area
Interrupt-exiting:1
Exit info
Interrupt info

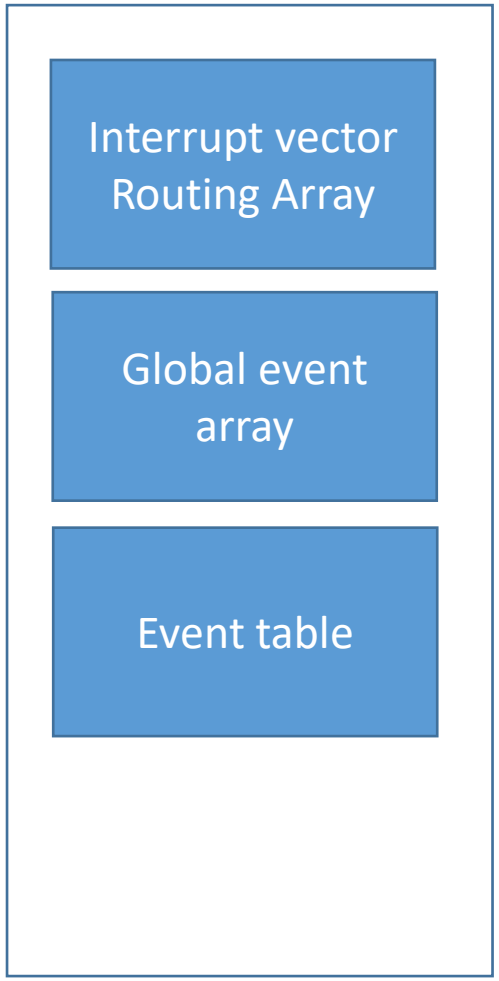
VMCS 2

CurrentVMCS:VMCS1

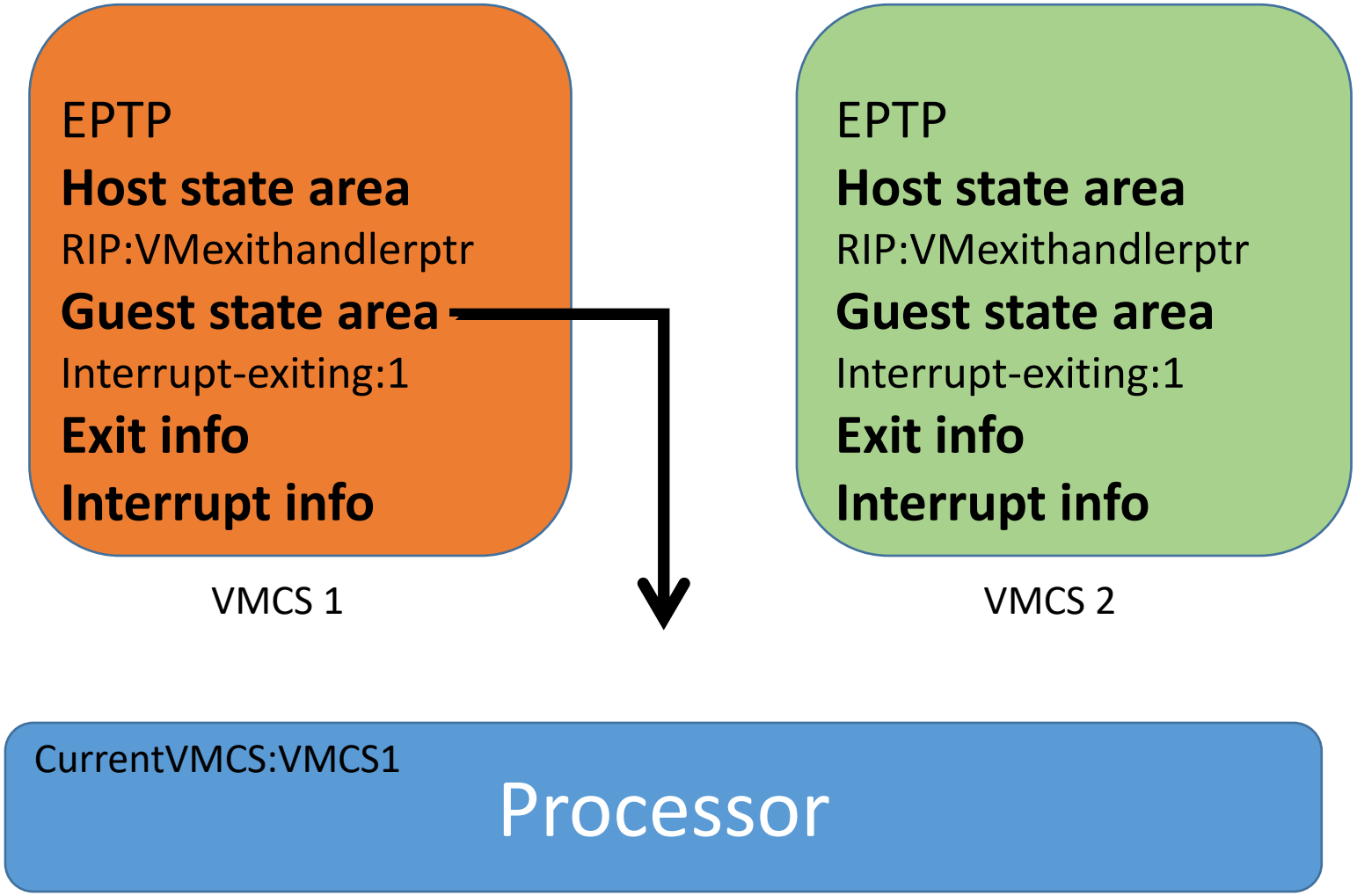
Processor

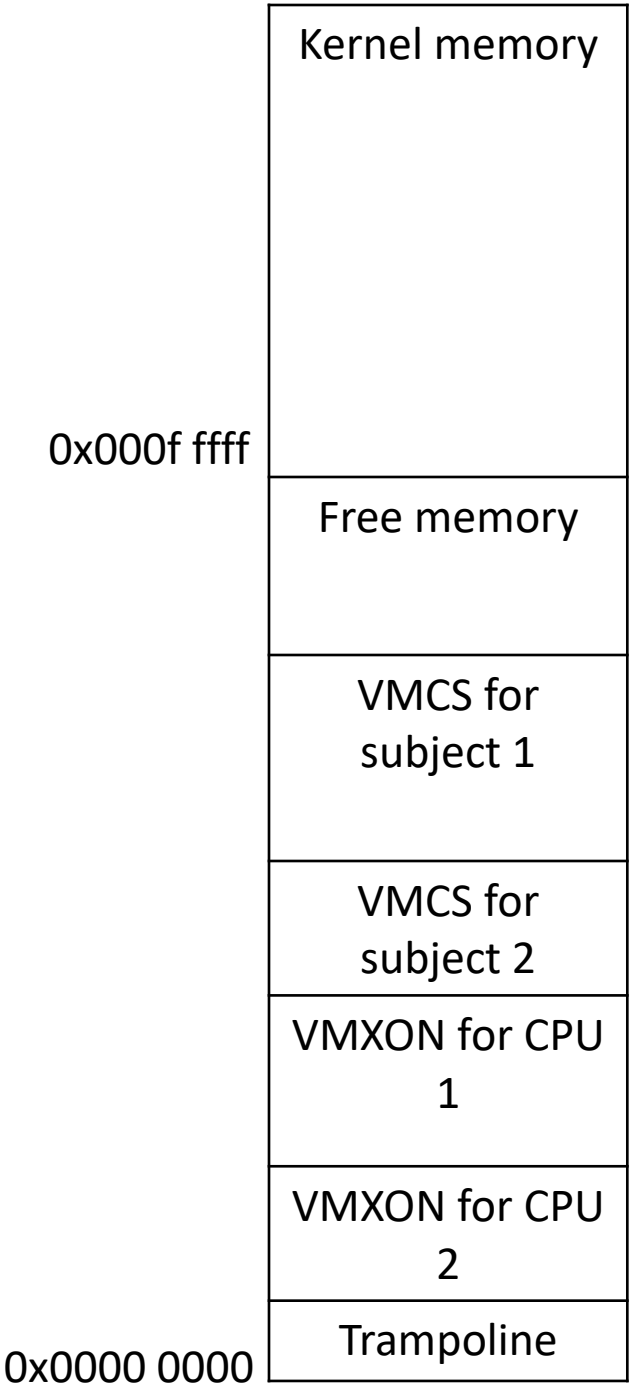


Kernel data structures

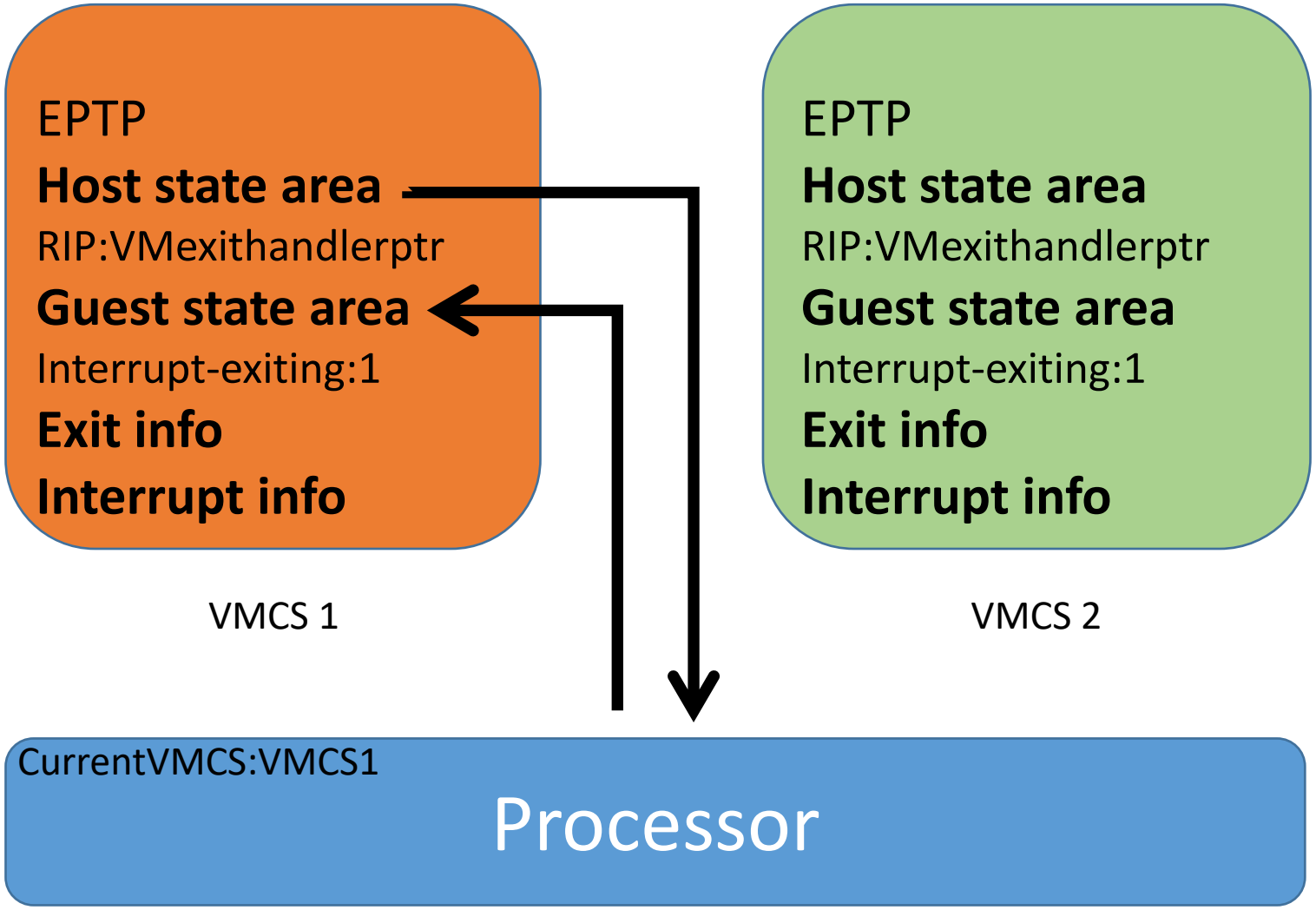


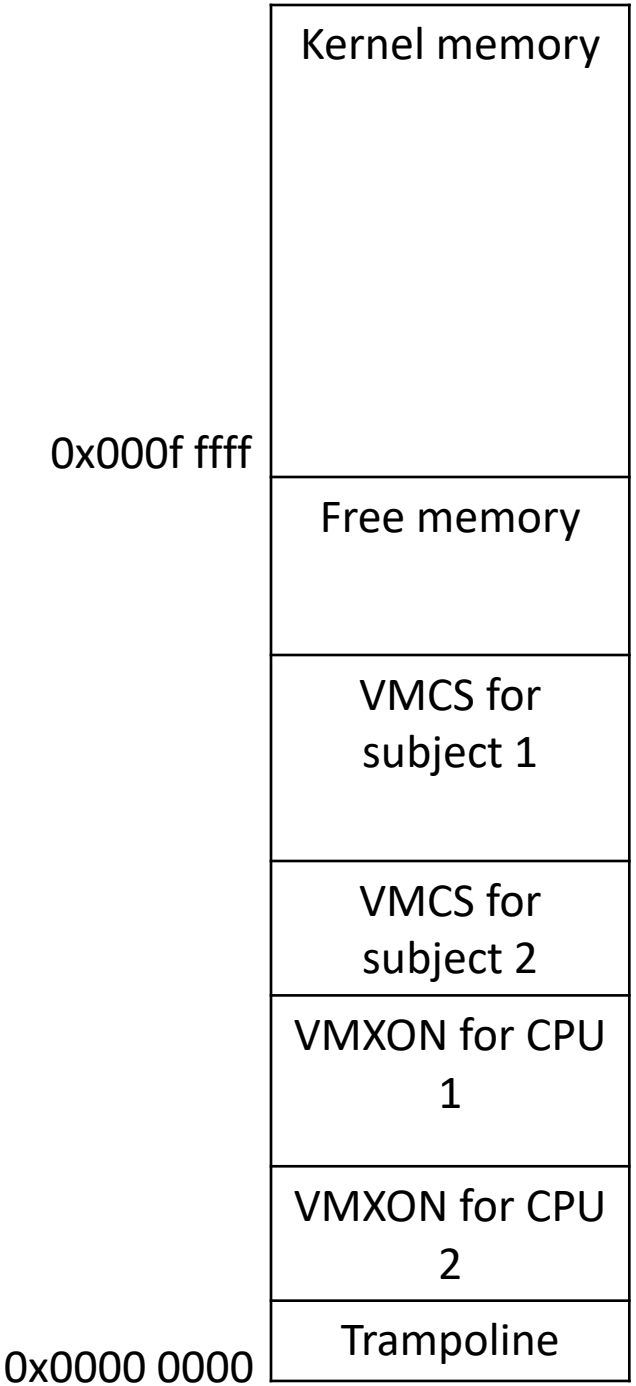
VM Entry


EPTP
Host state area
RIP:VMexithandlerptr
Guest state area
Interrupt-exiting:1
Exit info
Interrupt info
VMCS 1
EPTP
Host state area
RIP:VMexithandlerptr
Guest state area
Interrupt-exiting:1
Exit info
Interrupt info
VMCS 2
CurrentVMCS:VMCS1
Processor


Kernel memory
0x000f ffff
Free memory
VMCS for subject 1
VMCS for subject 2
VMXON for CPU 1
VMXON for CPU 2
Trampoline
0x0000 0000

VM Exit

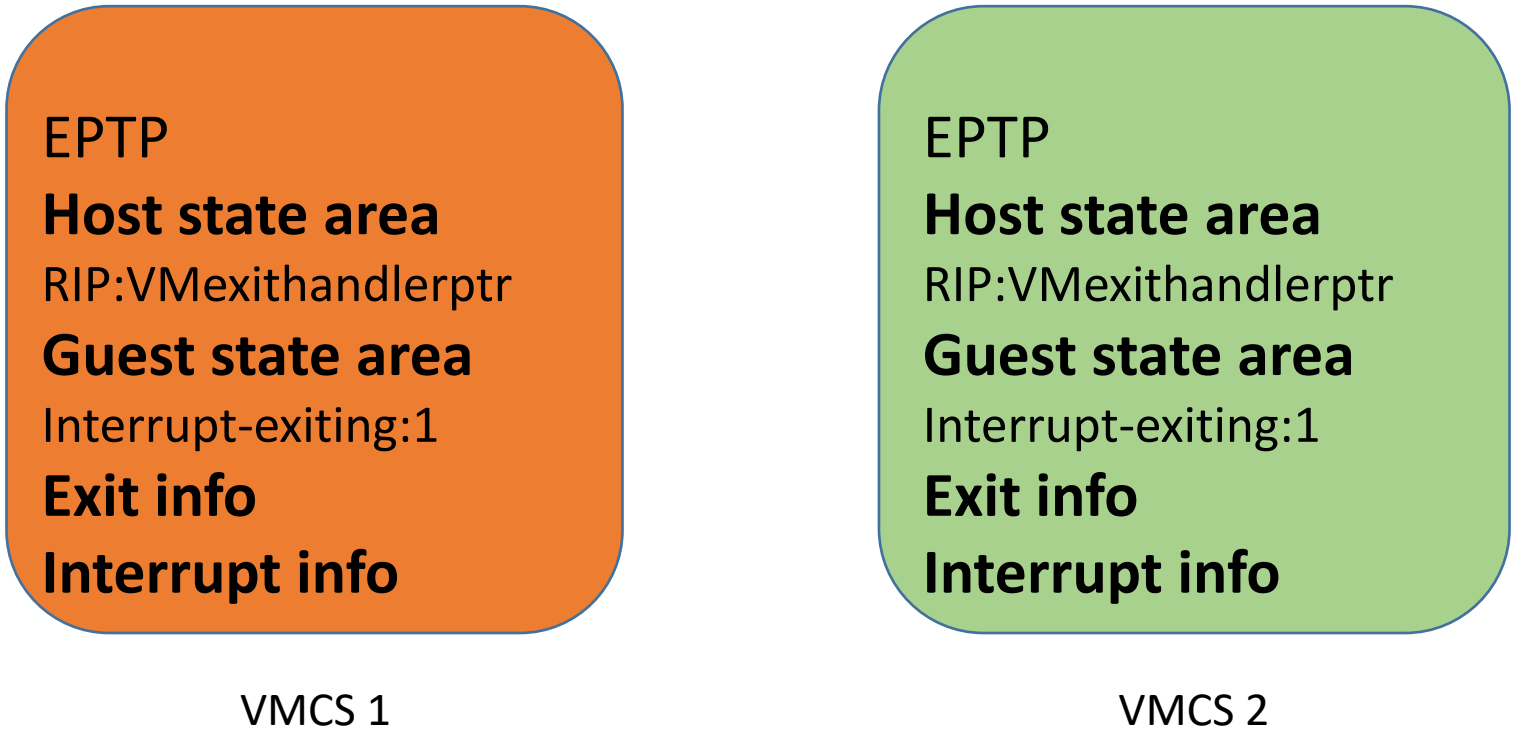

EPTP
Host state area
RIP:VMexithandlerptr
Guest state area
Interrupt-exiting:1
Exit info
Interrupt info
VMCS 1
EPTP
Host state area
RIP:VMexithandlerptr
Guest state area
Interrupt-exiting:1
Exit info
Interrupt info
VMCS 2
CurrentVMCS:VMCS1
Processor

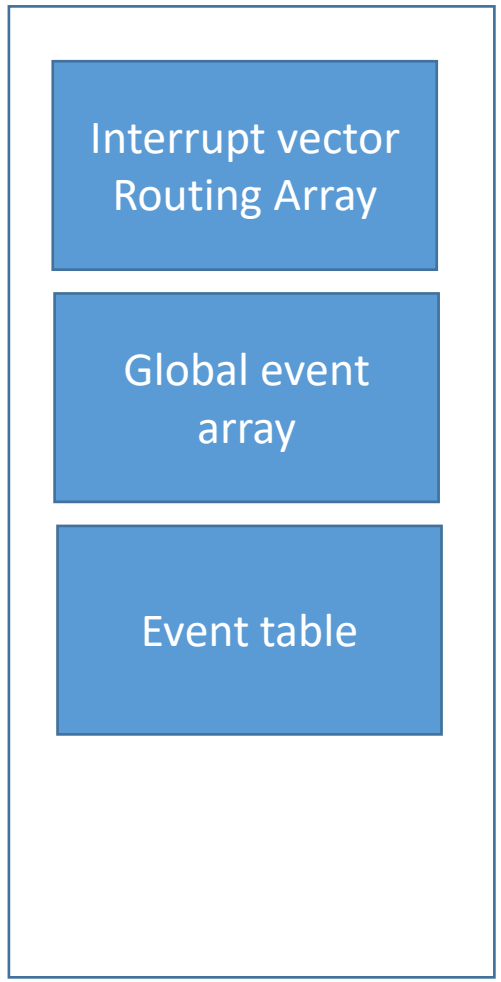

Kernel memory
0x000f ffff
Free memory
VMCS for subject 1
VMCS for subject 2
VMXON for CPU 1
VMXON for CPU 2
Trampoline
0x0000 0000

VM Exit

- Various reasons of VM-exit
  - External interrupts
  - VMX preemption timer expiry
  - VMCALL instruction
  - Interrupt-window exiting

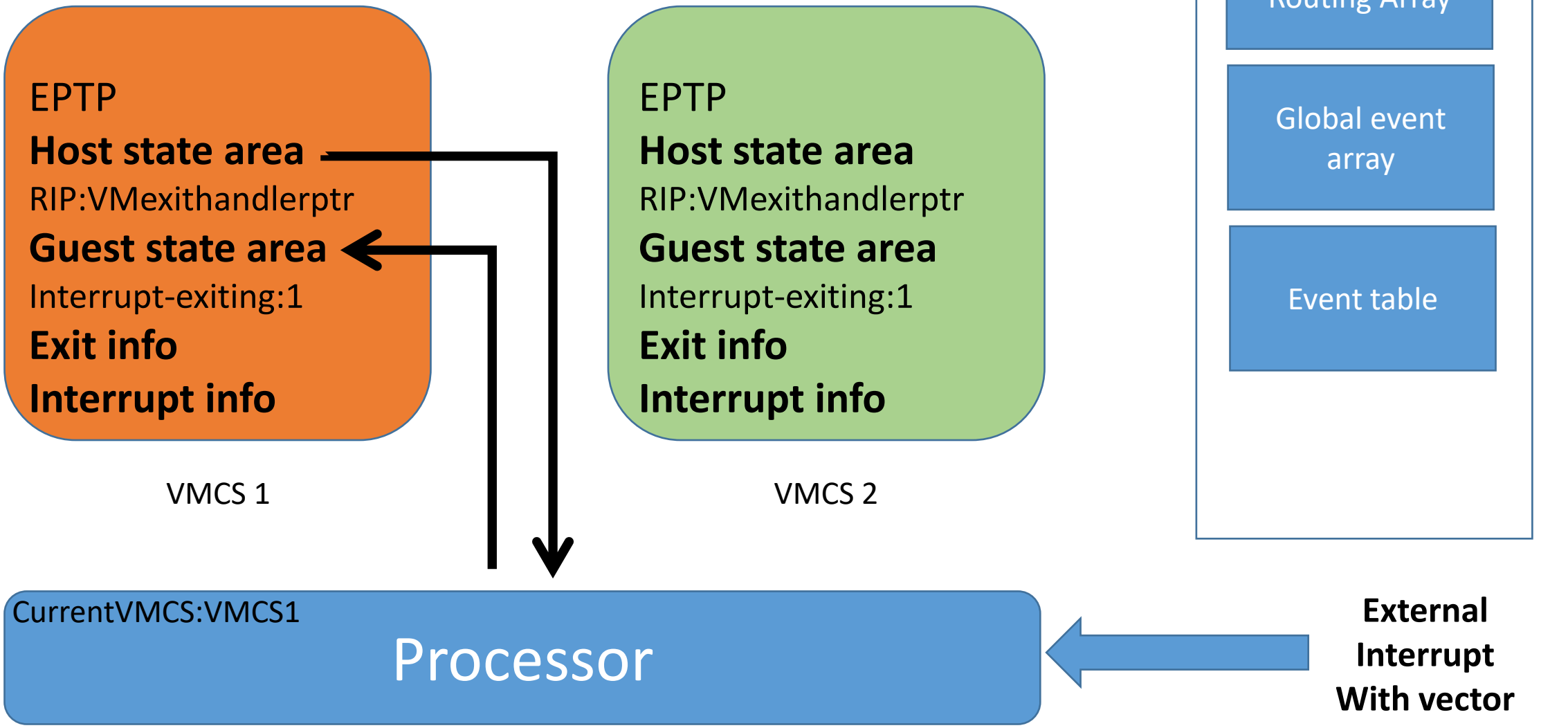
VM Exit – External Interrupt



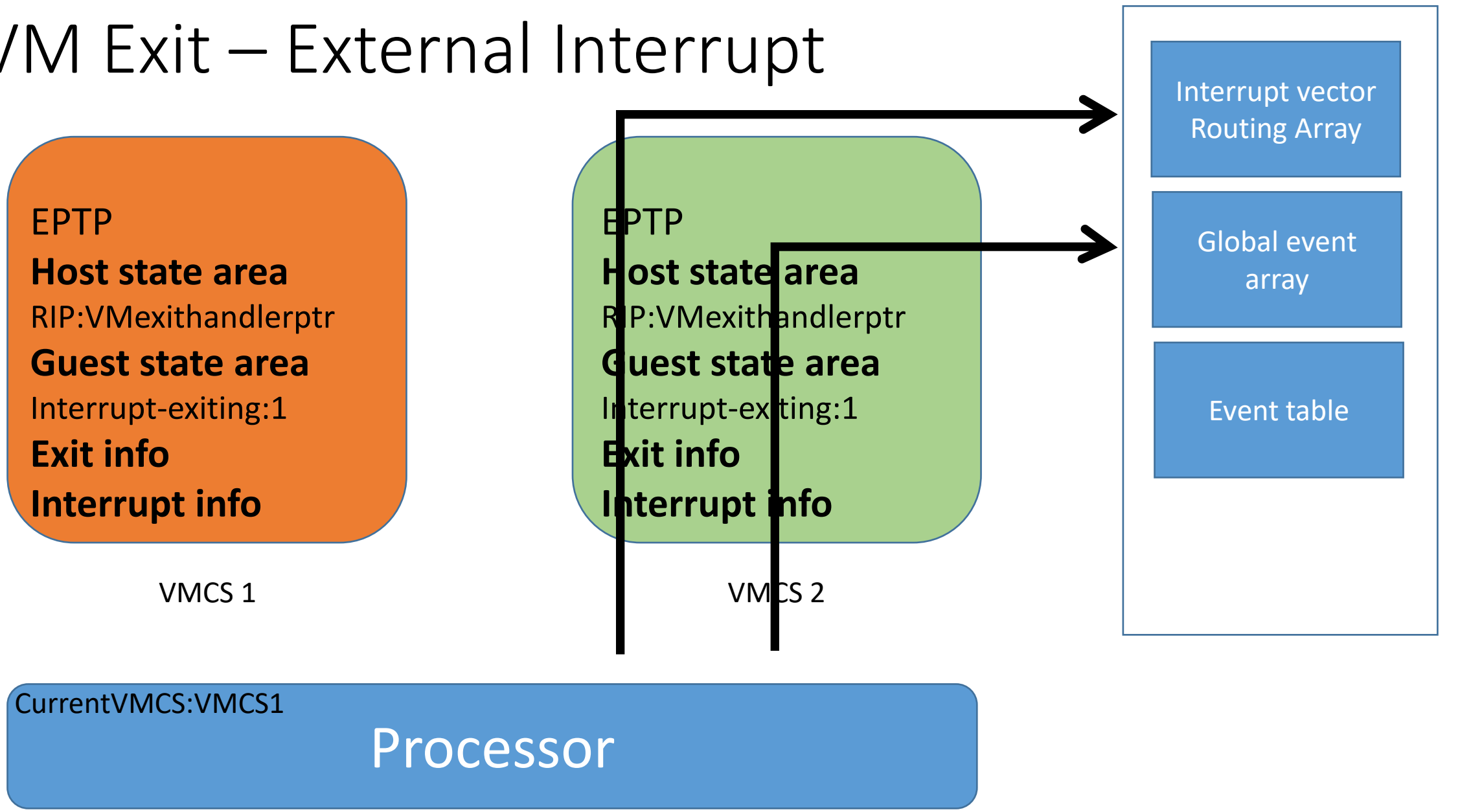


External Interrupt With vector

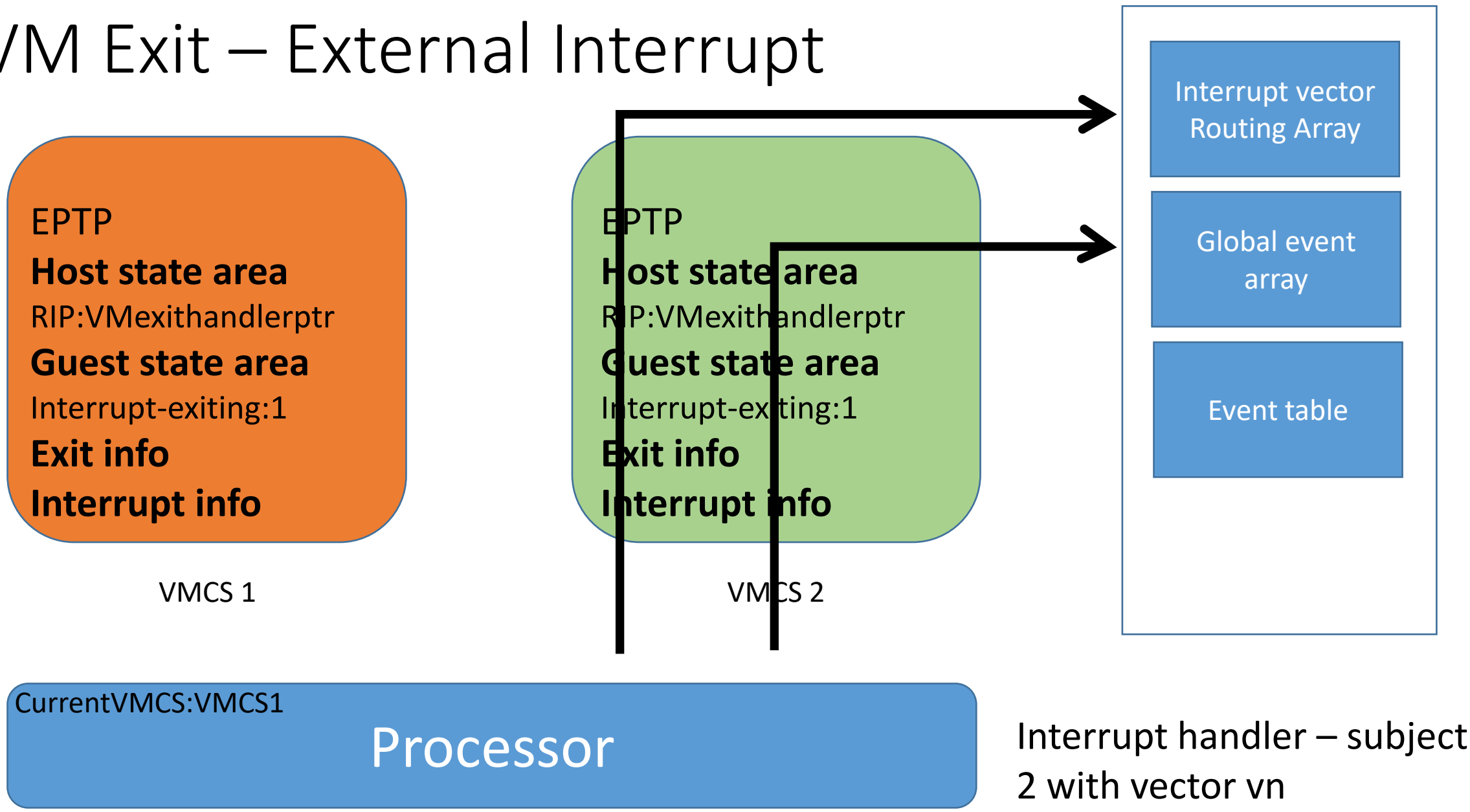
VM Exit – External Interrupt



VM Exit – External Interrupt

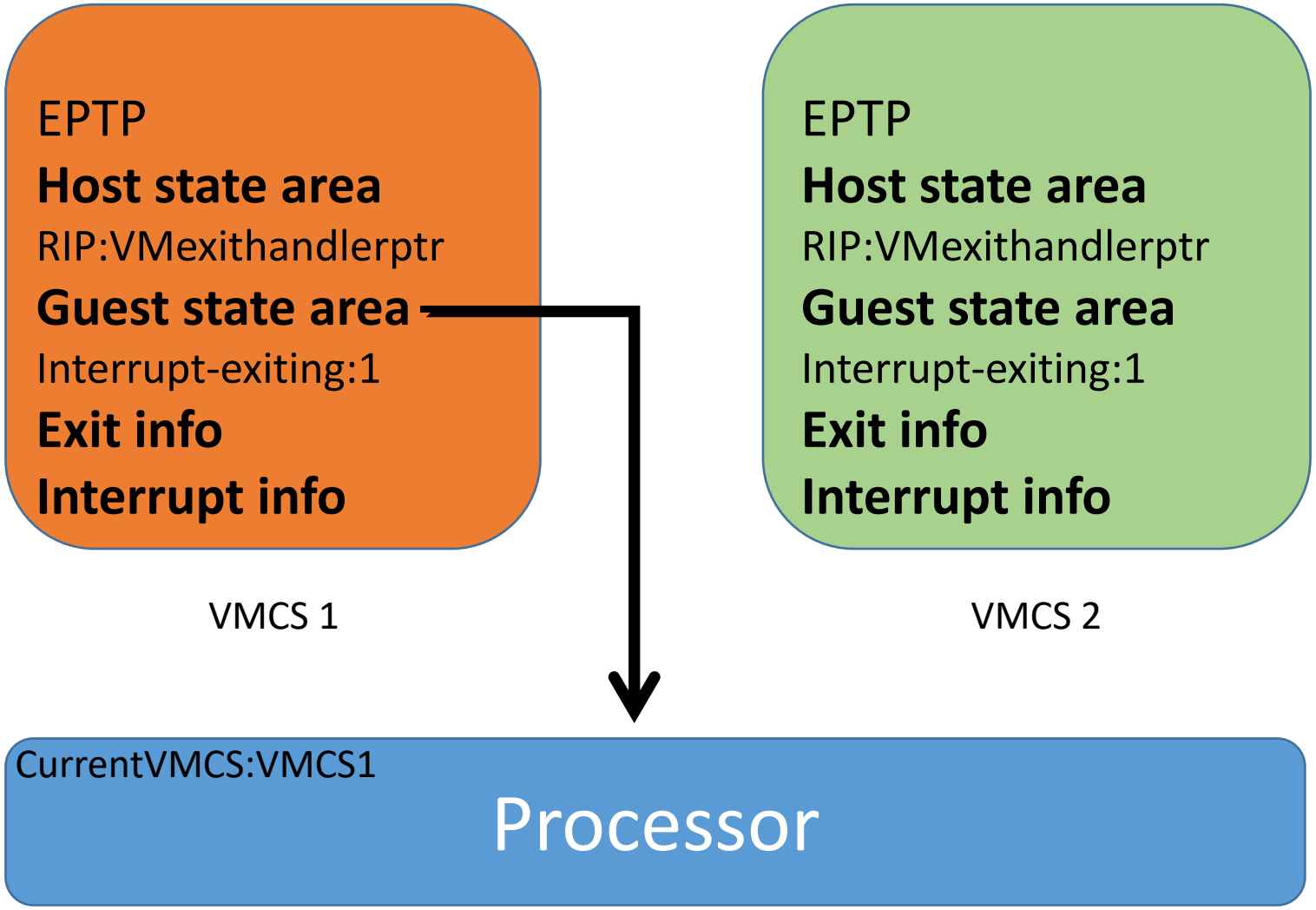


VM Exit – External Interrupt

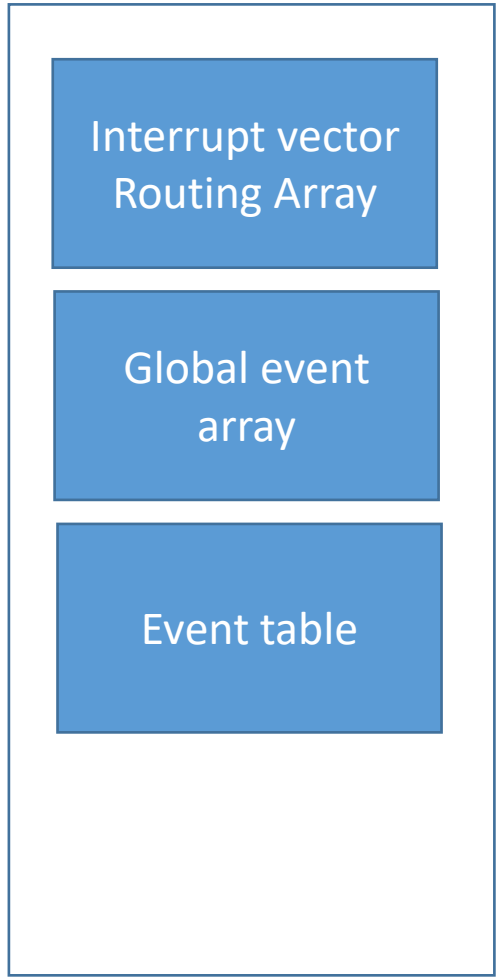


Interrupt handler – subject 2 with vector vn

VM Exit – External Interrupt

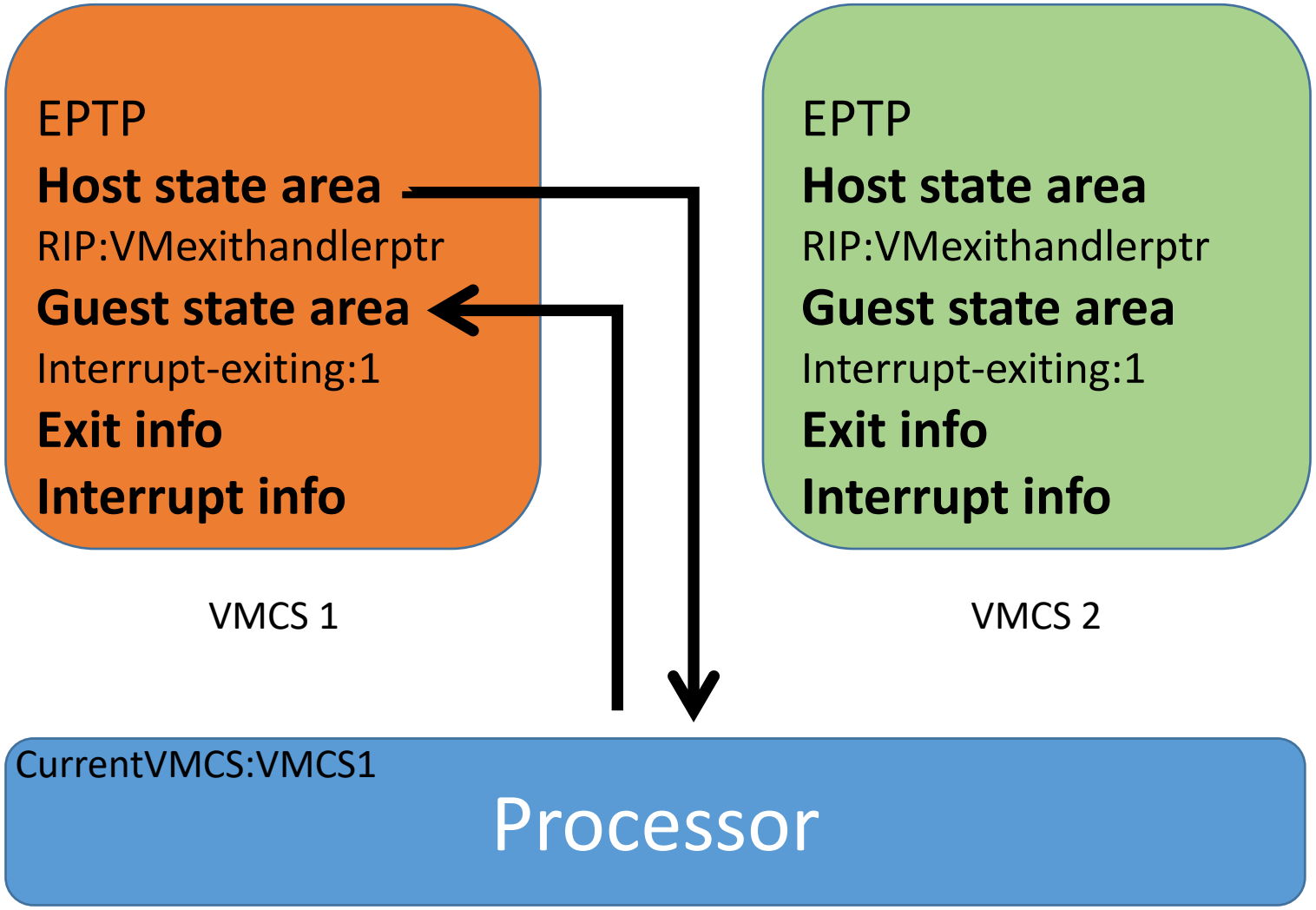


Kernel data structures

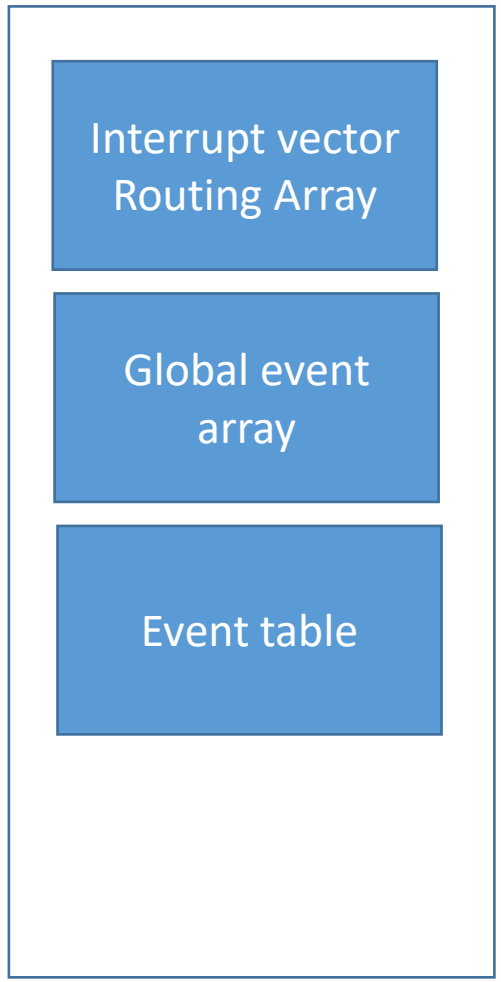


Subject 1 starts running again

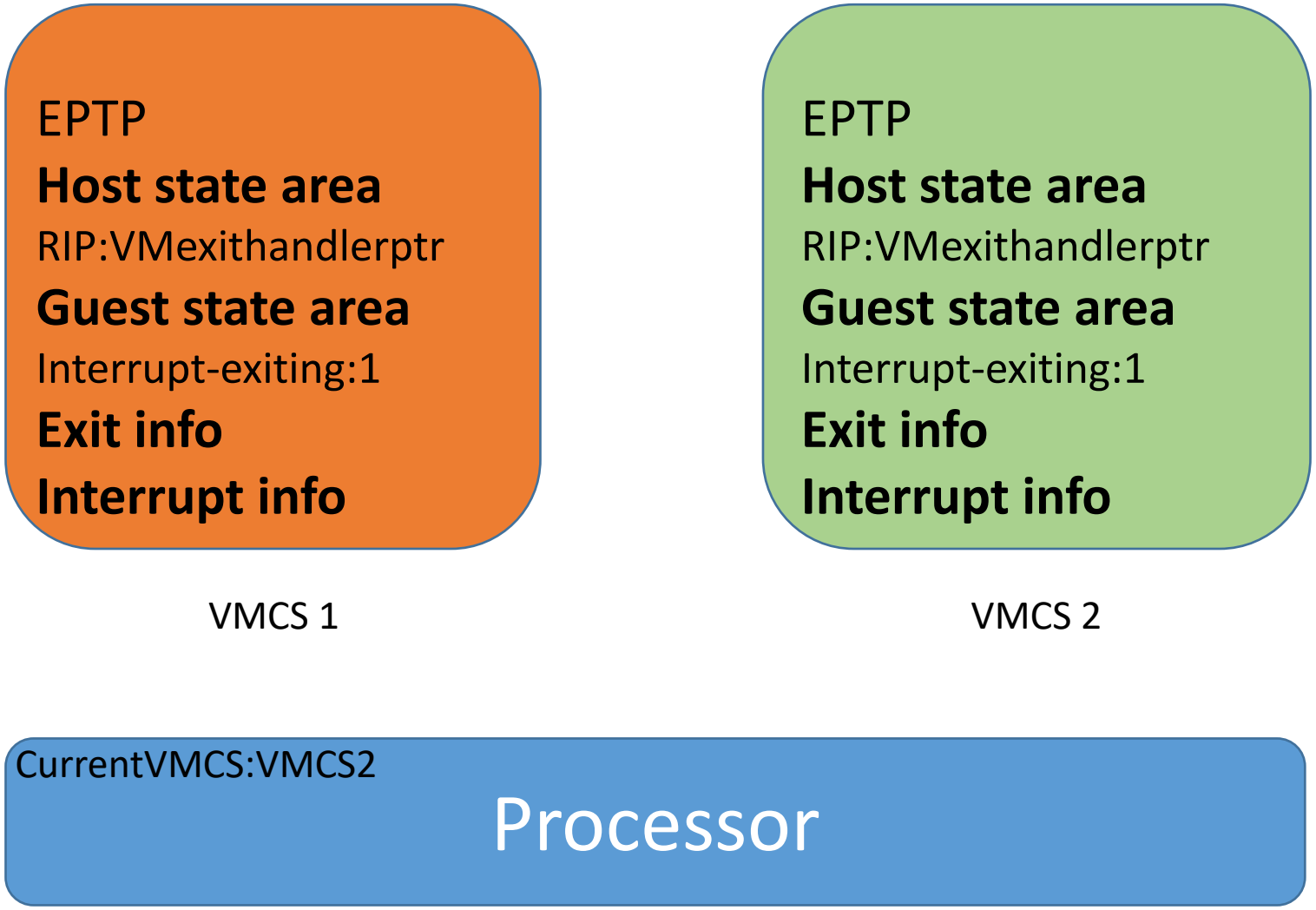
VM Exit – Timer Expiry

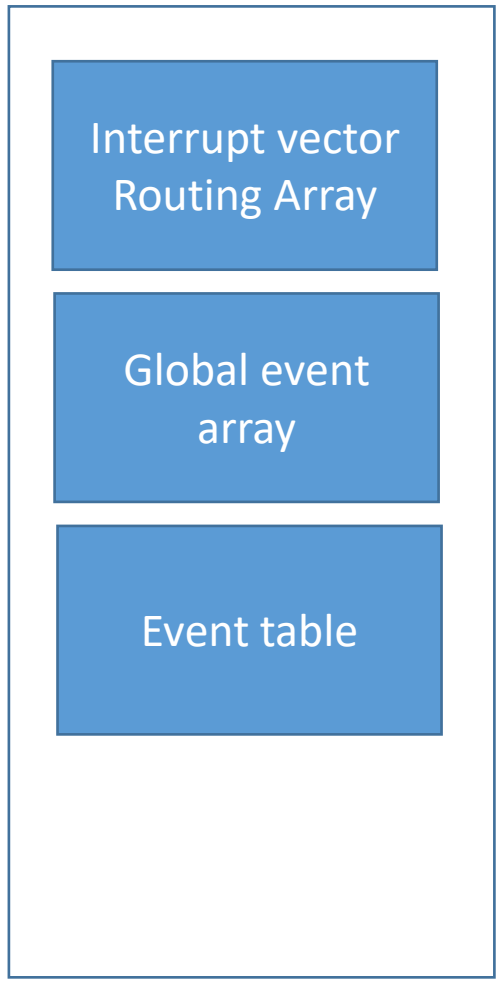


Kernel data structures

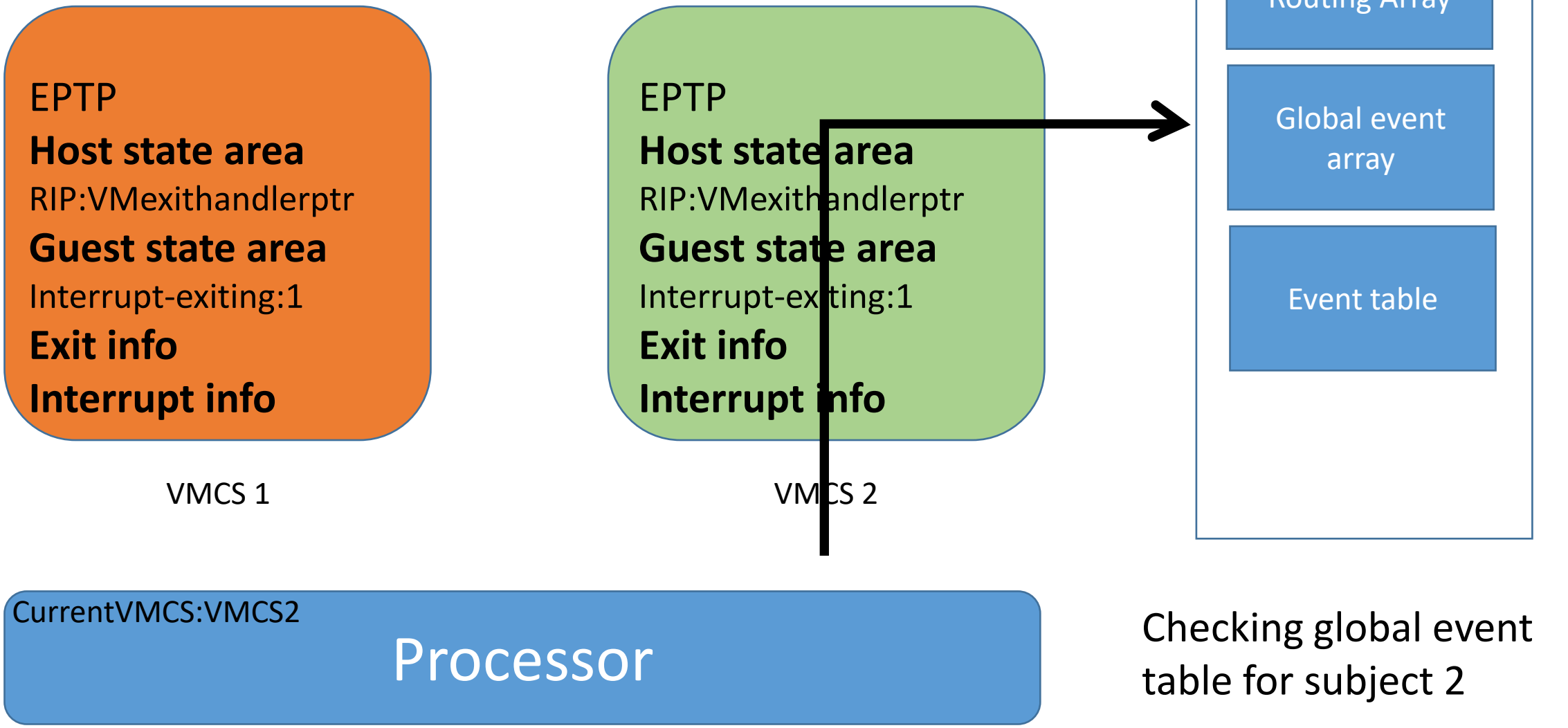


VM Exit – Timer Expiry

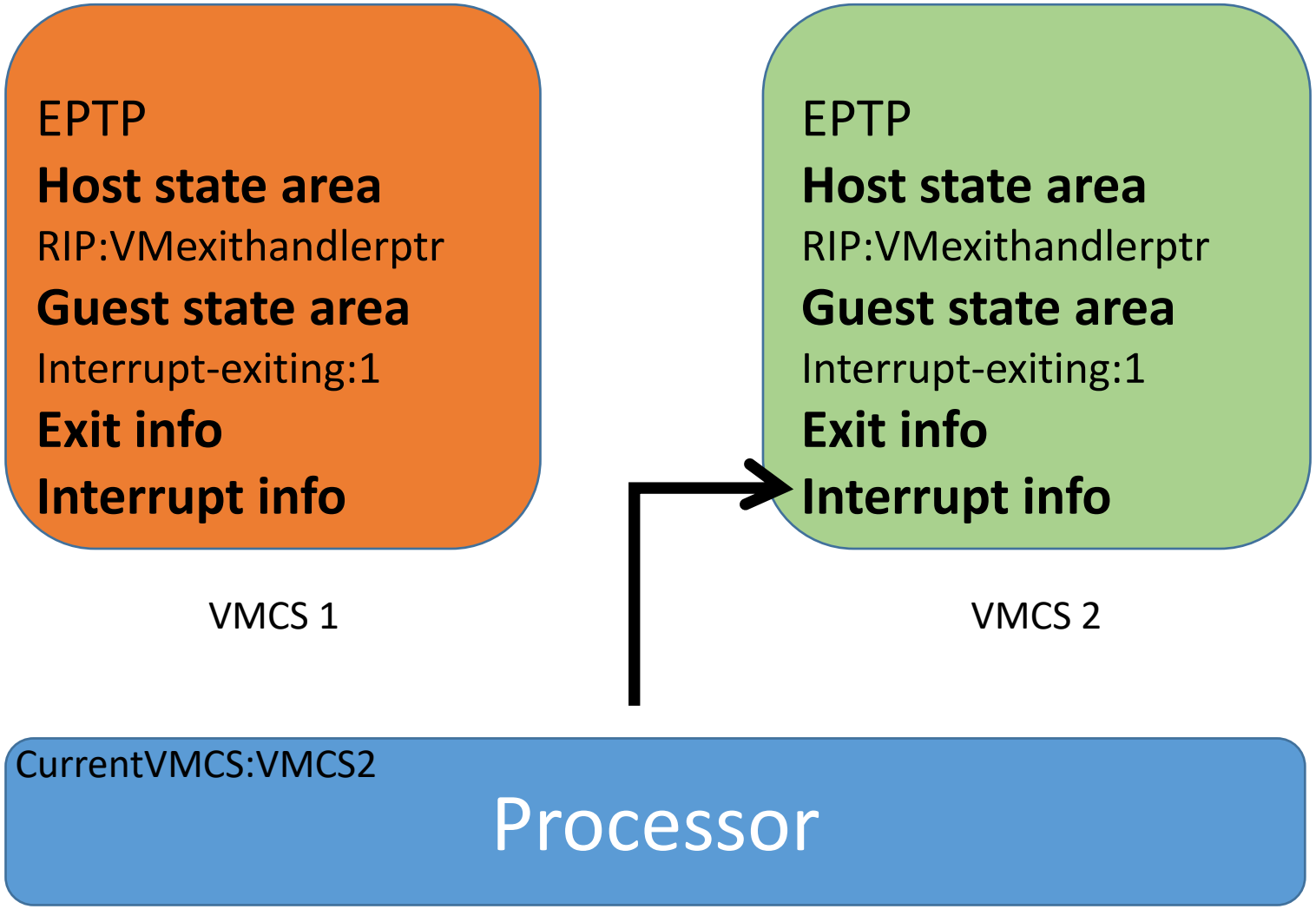




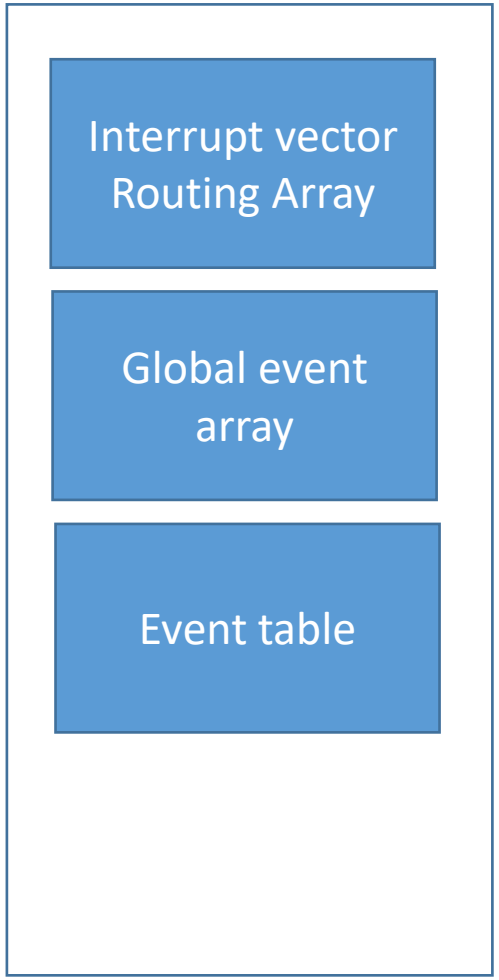
VM Exit – Timer Expiry



VM Exit – Timer Expiry

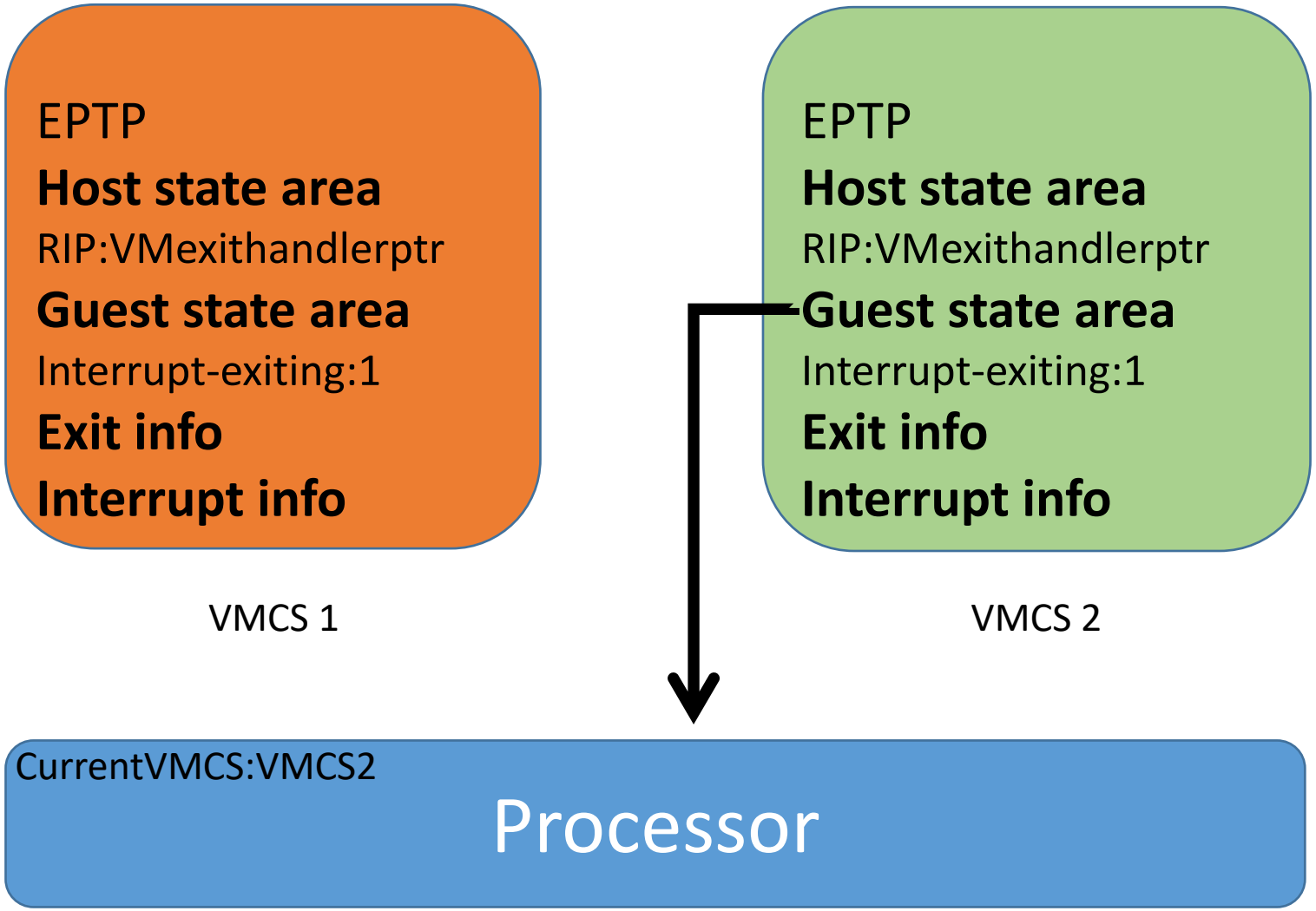


Kernel data structures

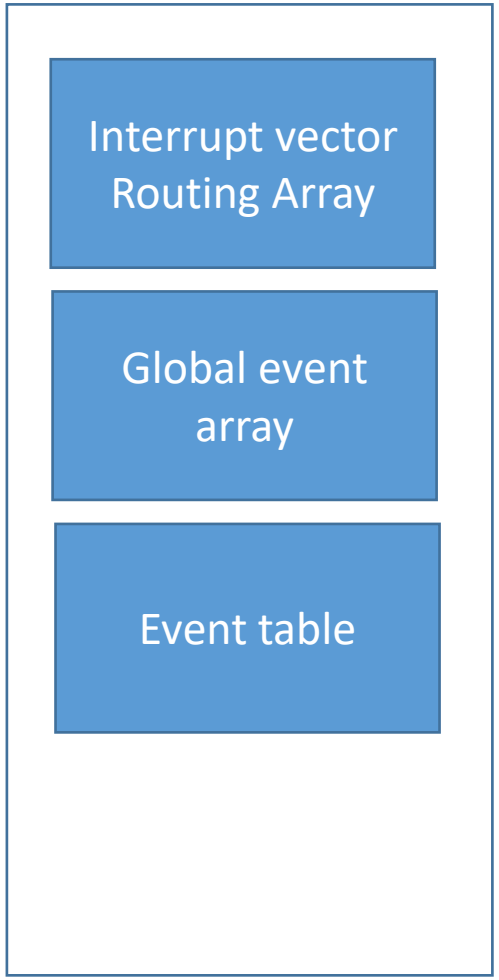


Writing interrupt info in VMCS of subject 2

VM Exit – Timer Expiry

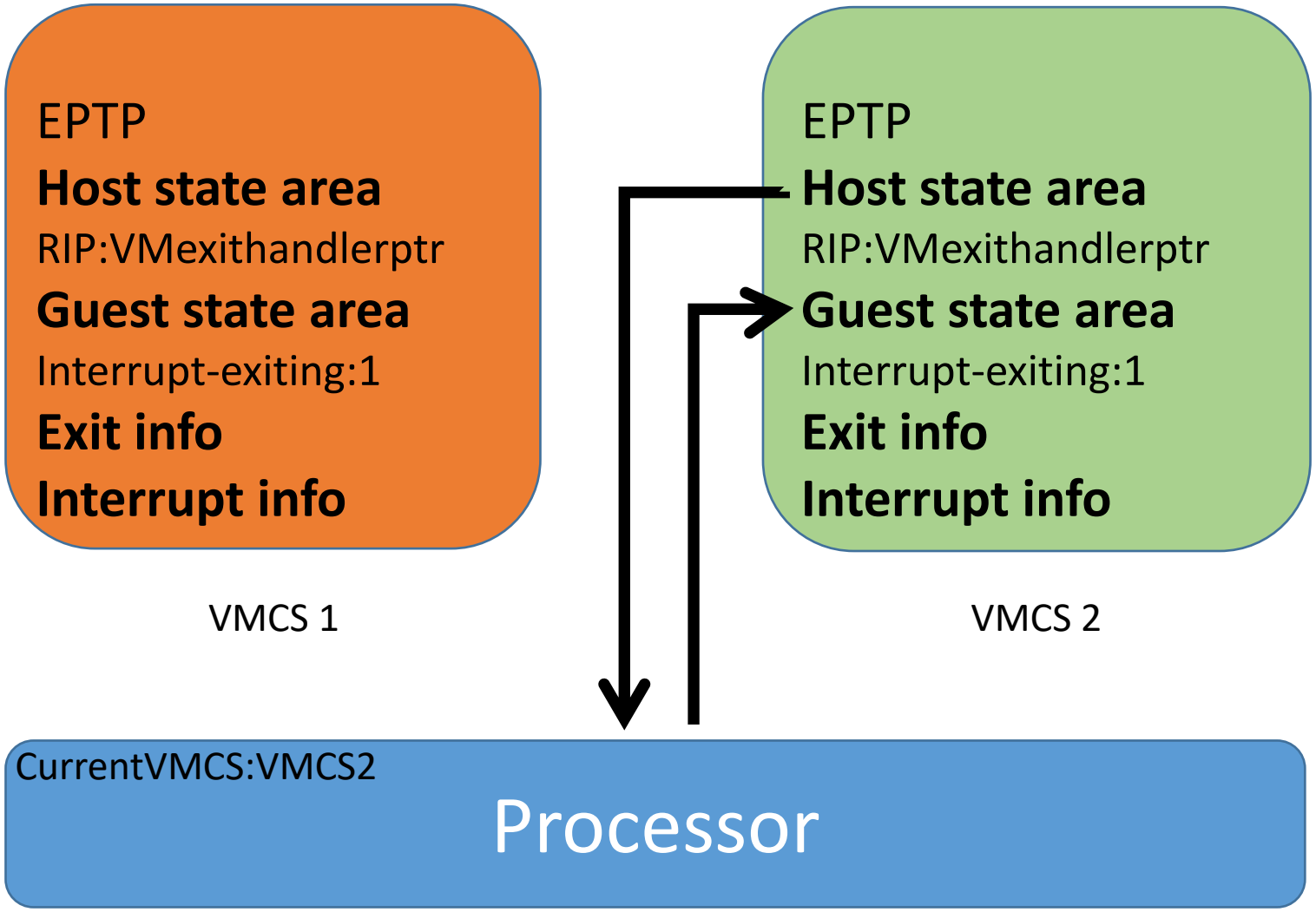


Kernel data structures

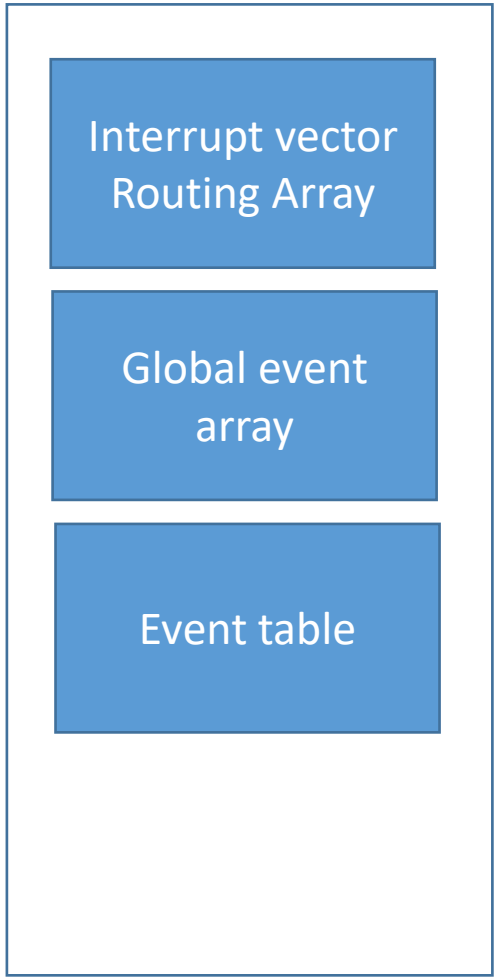


Checking interrupt info field before entry

VM Exit - Hypercall

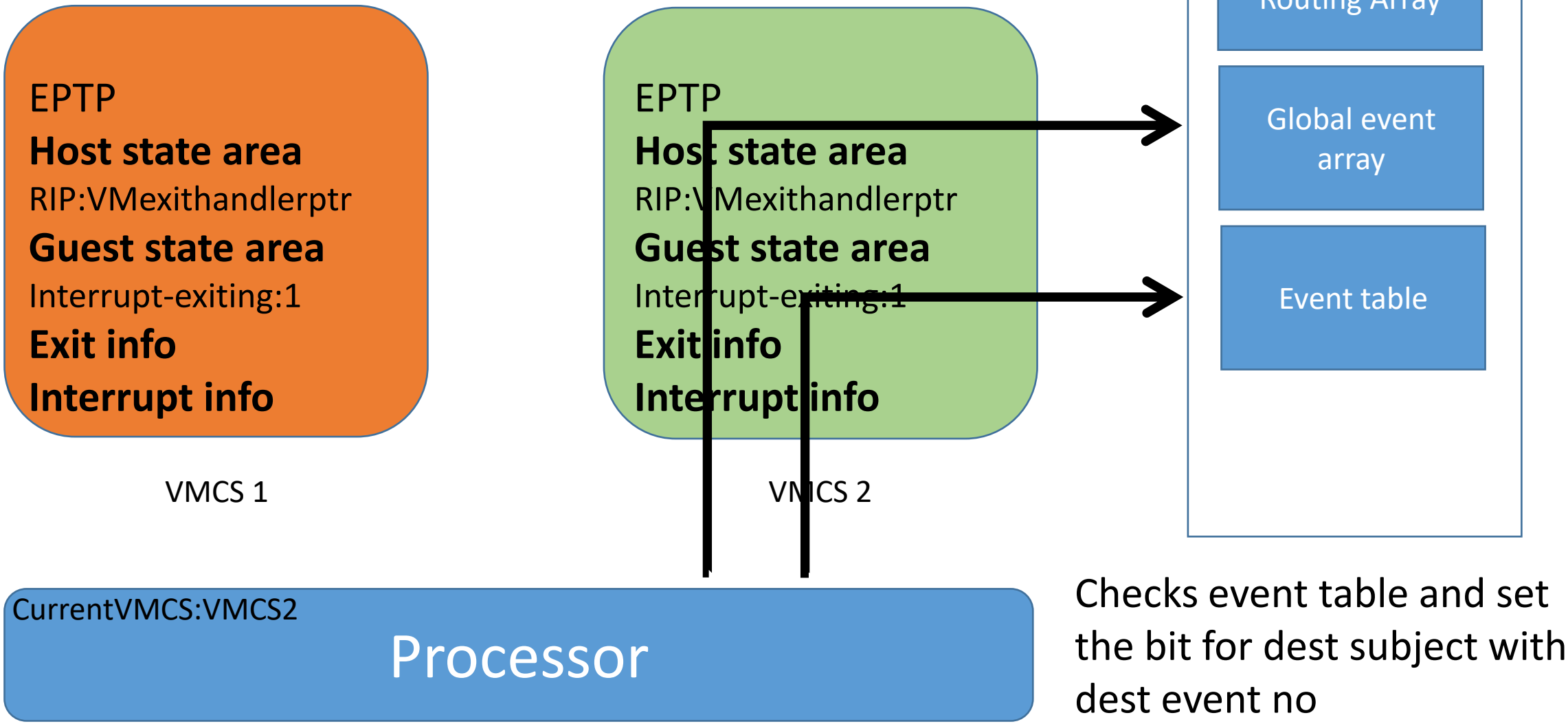


Kernel data structures



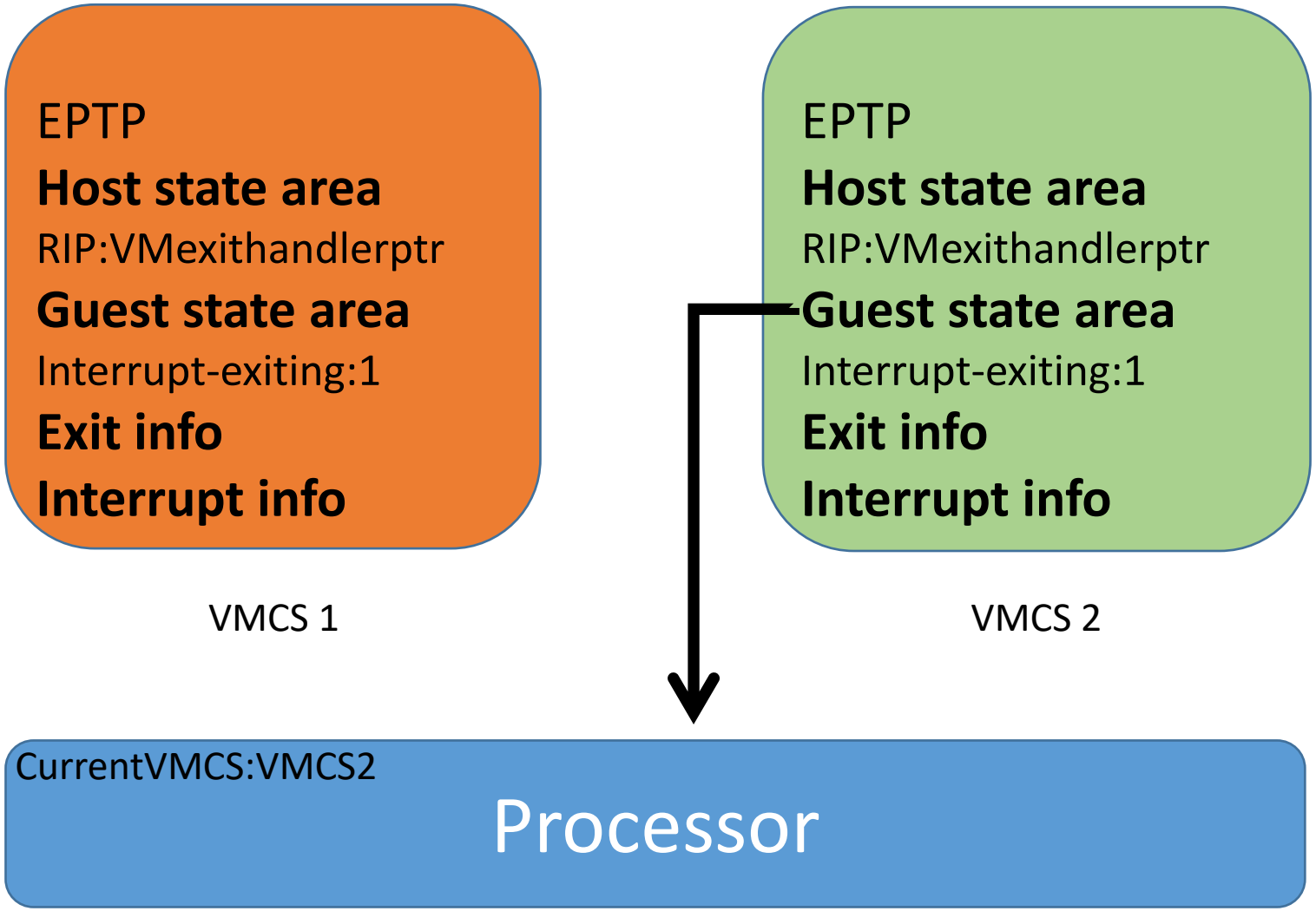
Guest 2 executed VMCALL instruction with operand in A register

VM Exit - Hypercall

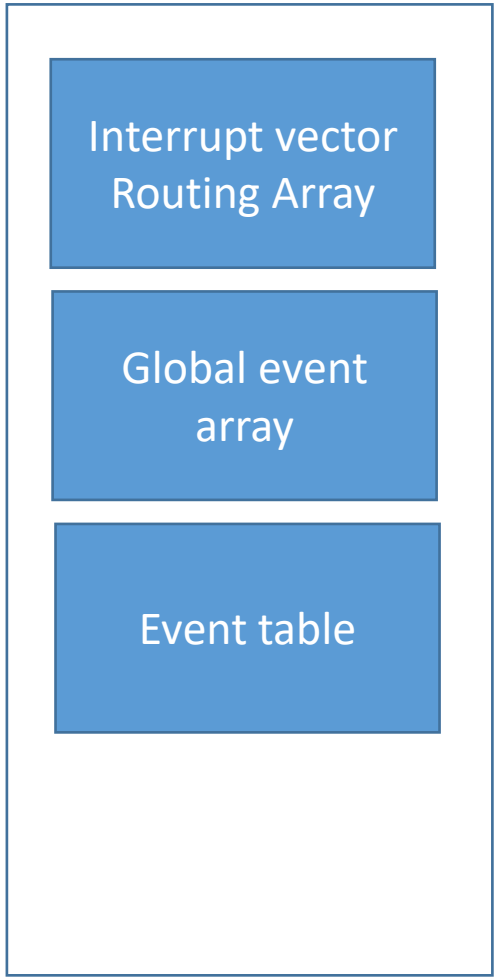


Checks event table and set the bit for dest subject with dest event no

VM Exit - Hypercall



Kernel data structures



Starts running again if not handover event