Trap-less Virtual Interrupt for KVM on RISC-V

Design, Implementation and Challenges

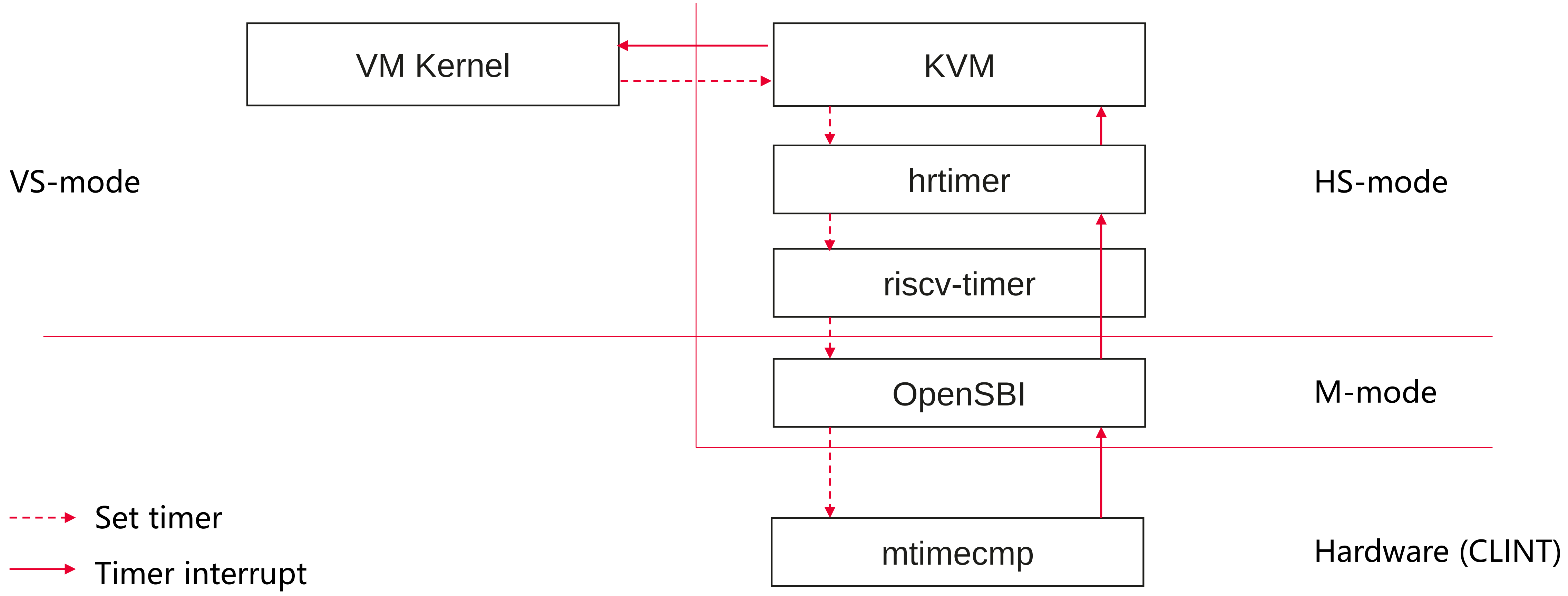
Siqi Zhao <zhaosiqi3@huawei.com>

Contents

Background & Motivation

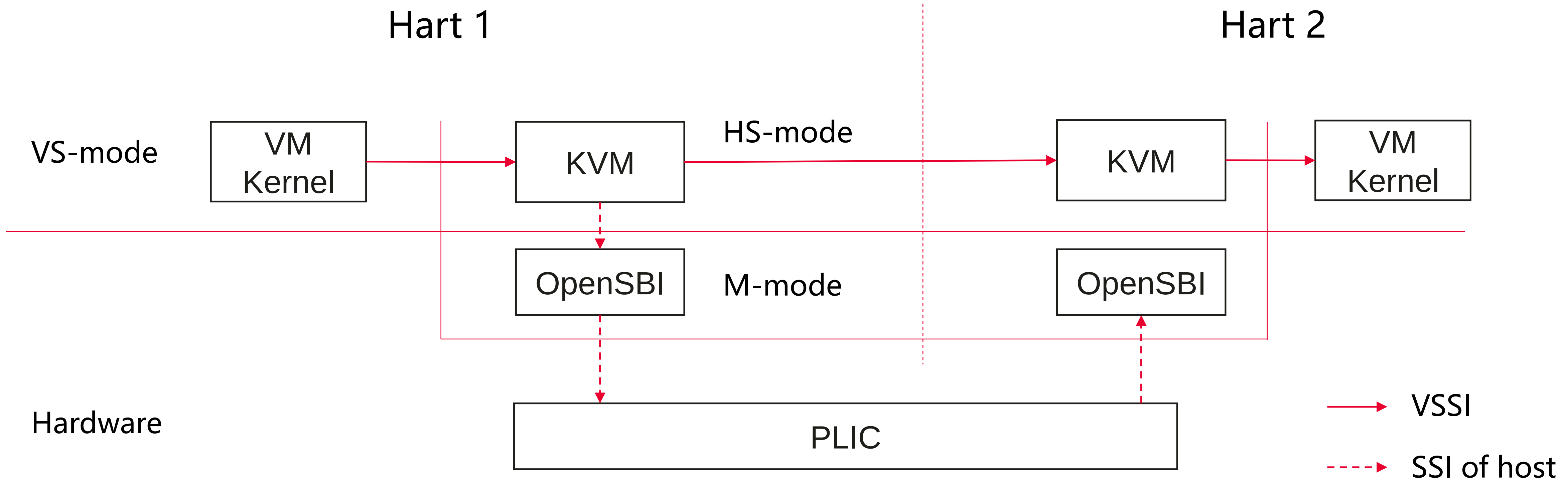
Background

- RISC-V’s current approach to virtual interrupt is based on trap-n-emulate.
    - Three types: timer, software-generated, external
- The virtual timer interrupt flow on current KVM architecture:



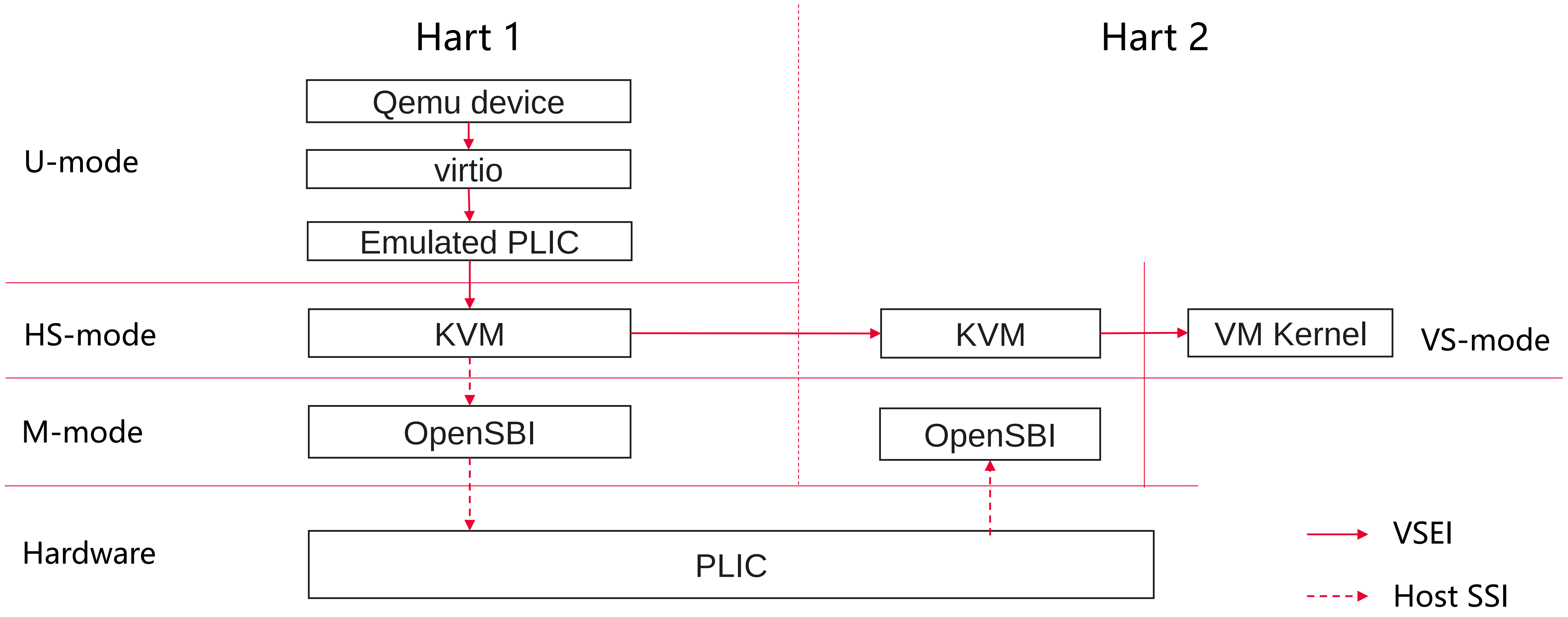
Background

- The virtual supervisor software-generated interrupt (VSSI) flow on current KVM architecture:



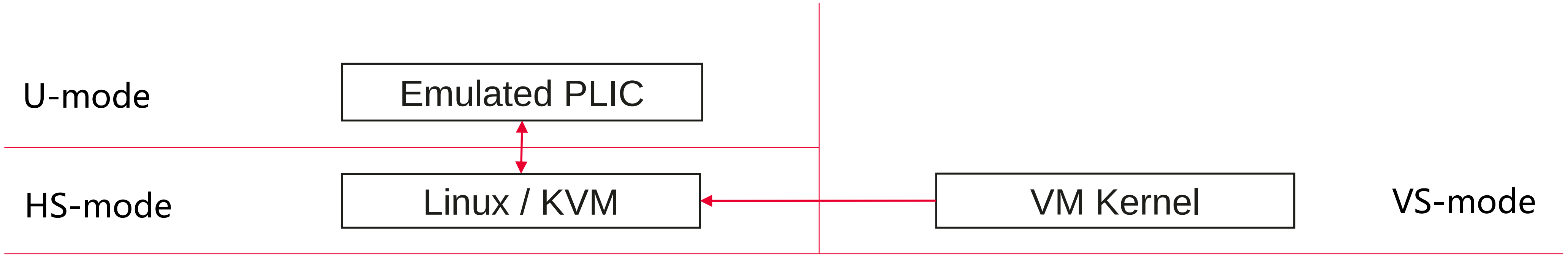
Background

- The virtual supervisor external interrupt (VSEI) flow on current KVM architecture:



Background

- The flow of acknowledging VSEI on current KVM architecture:



Motivation

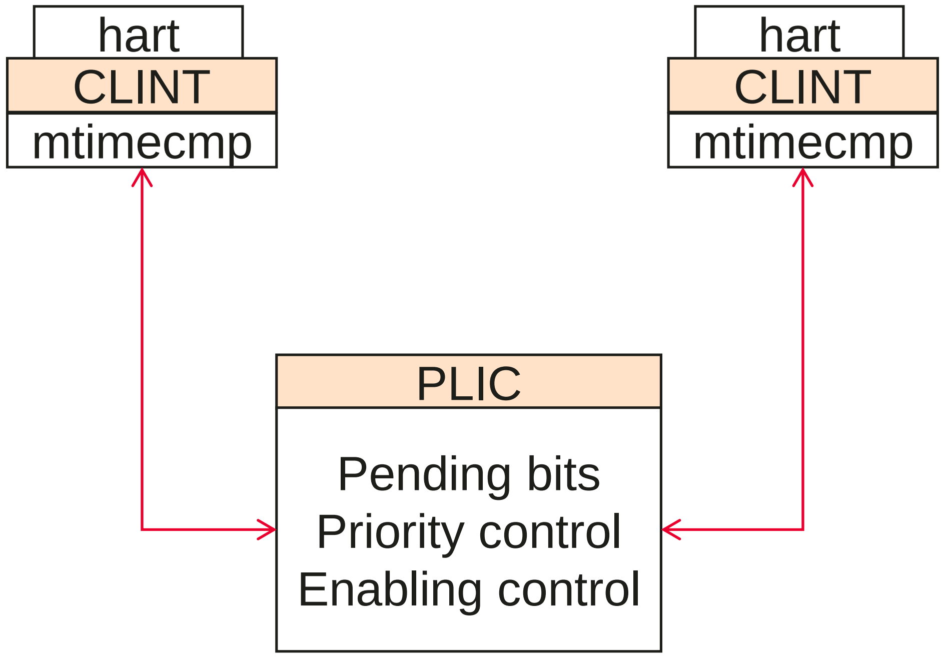
- Trap-n-emulate introduces a number of traps for all three types of virtual interrupt.
- Performance critical workloads will suffer from frequent traps thus lost CPU cycles.
- Existing virtualization technology on other architectures such as x86 and ARM already introduced some kind of interrupt pass-through mechanism, such as Posted Interrupt and ITS

- What about RISC-V?

Overview

Overview

- We designed a virtualization-aware interrupt controller for RISC-V
- We implemented KVM support for the virtualization-aware interrupt controller
- Background: existing RISC-V interrupt controller:



Virtualization-aware Core-Local Interrupt Controller (CLINT)

Interrupt	Name	Type	Mode	Purpose
Timer	stimecmp	CSR	HS-mode	The clock event source for HS-mode
	vstimecmp	CSR	VS-mode	The clock event source for VS-mode
	stimecmp	Alias	VS-mode	Alias of vstimecmp
Software	sgenipi	CSR	HS-mode	Write generates HS-mode SGIs
	vsgenipi	CSR	HS-mode	Write generates VS-mode SGIs
	sgenipi	Alias	VS-mode	Alias of vsgenipi
	vshartid	CSR	VS-mode	the identifier of the vCPU running on a hart
	shartid	Alias	VS-mode	Alias of vshartid
External	ugenvsei	CSR	U-mode	Write generates VS-mode VSEIs
	vsclaim	CSR	HS-mode	For guest to claim the pending VSEIs
	sclaim	Alias	VS-mode	Alias of vsclaim

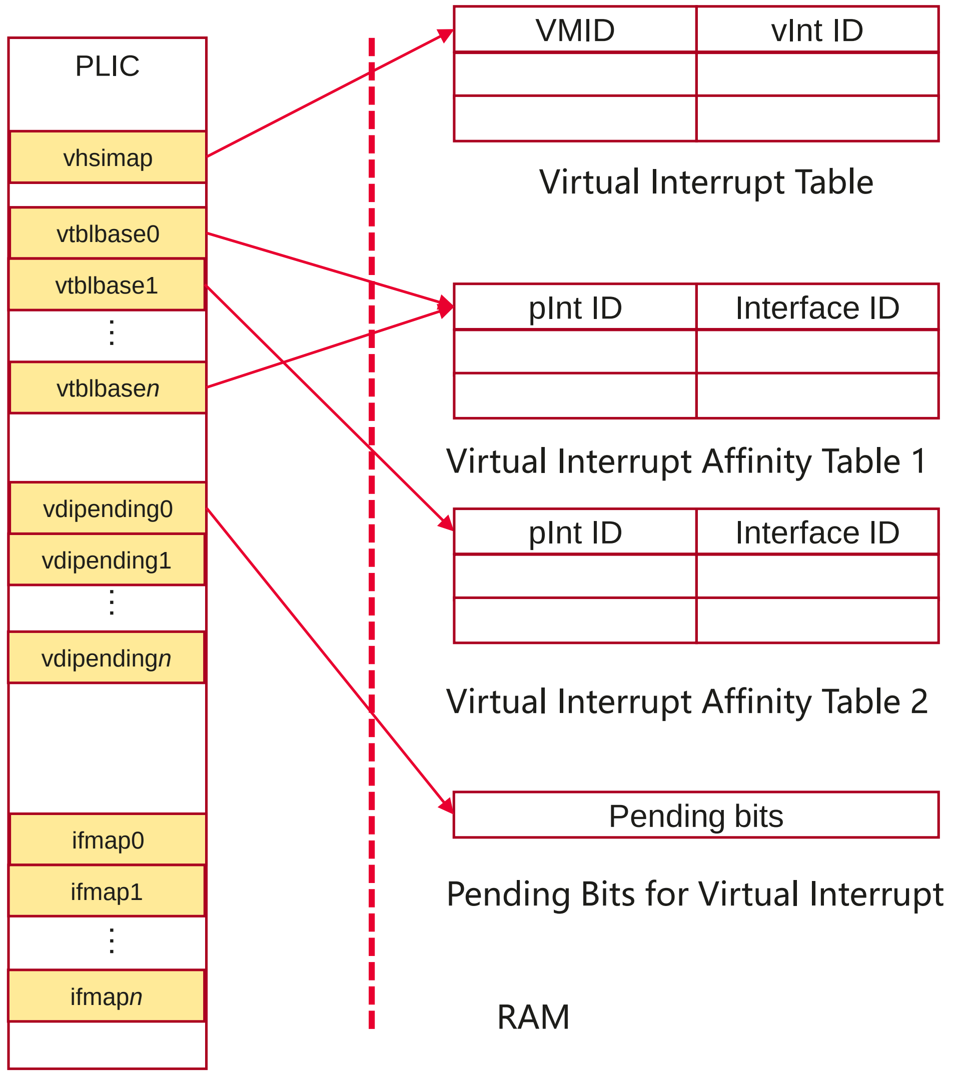
CLINT
stimecmp
vstimecmp
sgenipi
vsgenipi
vshartid
ugenvsei
vsclaim
stimecmp
sgenipi
shartid
sclaim

🟨 New CSRs

🟧 New aliases

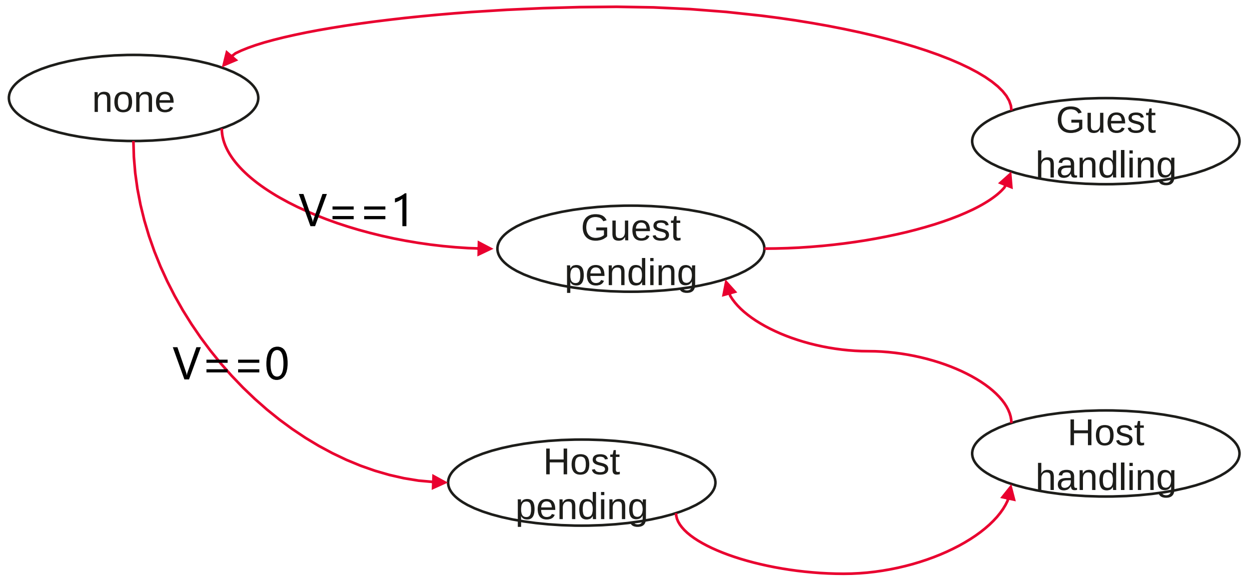
Virtualization-aware PLIC

- Virtualization-aware PLIC
  - The goal: minimize vCPU thread contention during vCPU scheduling
- New registers
  - *vhsimap*: holds the base address to the virtual interrupt table
  - *vtblbase*[n]: holds the base address to the virtual interrupt affinity table of the VM to which the vCPU on the hart n belongs.
  - ifmap[n]: holds the VMID and the vCPU ID of the vCPU on the hart n.
  - vdipending[n]: MMIO register, holds the pending VSEIs for a guest



Direct Injection of Virtual Interrupts

- The hart is modified so that the CLINT can directly set the pending bits for the virtual interrupt
- When the virtual interrupt becoming pending when V==0, the hart still takes it and the hypervisor handles the interrupt, which later injects this interrupt to a vCPU.

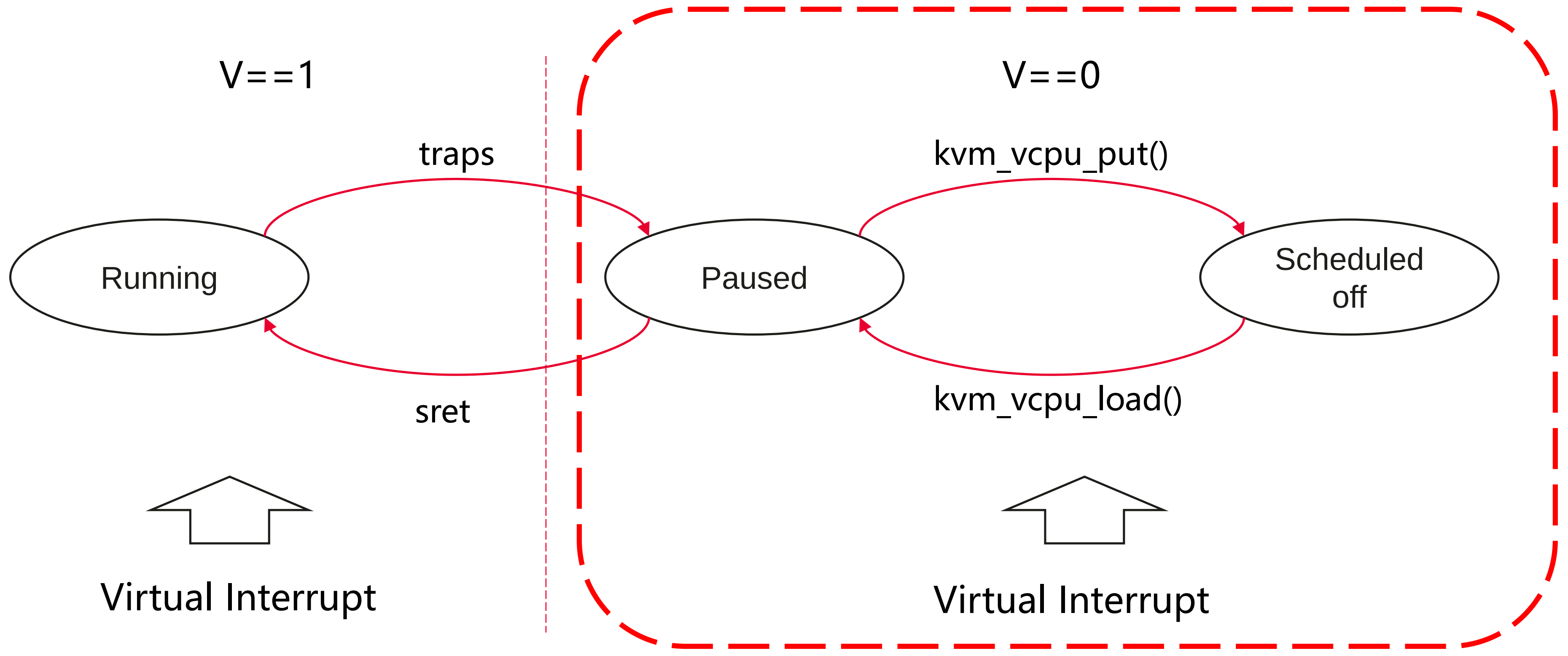


Direct Injection of Virtual Interrupts

- There are some challenges posed by the existing CSRs in the H-extension
- For virtual timer interrupt, we managed with only hip.VSTIP
  - The hypervisor doesn't really 'claim' a virtual timer interrupt because it is level-triggered.
  - We do need to separate the enable bits, vsie.STIE is no longer an alias of hie.VSTIE.
- For VSSI, the separation is needed.
  - The vsip.SSIP bit indicates whether there is a VSSI pending for V==1.
  - The PLIC always sets hip.VSSIP. When the hart is in V==1, it converts hip.VSSIP to vsip.SSIP, the guest subsequently receives an SSI
  - When V==0, the host receives an VSSI.
  - vsie.SSIE bit enables / disables VSSI when V==1
- For VSEI, when V==0, the host receives an notification interrupt, it then injects a SEI to the guest.

KVM Support Overview

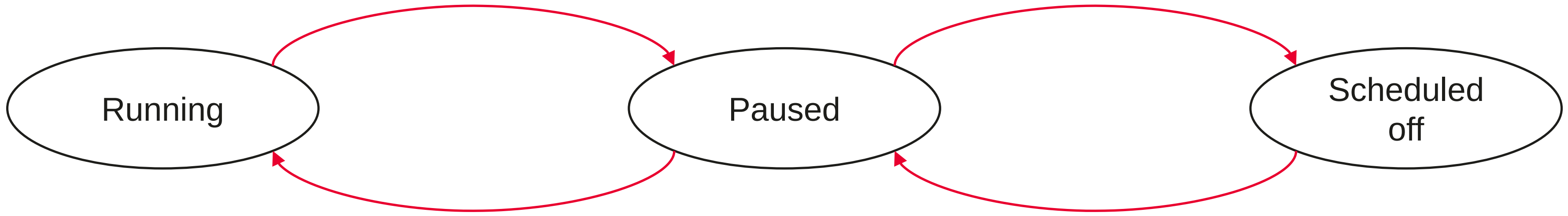
- Handle virtual interrupts when V==0, saving them for injection
- Maintain a consistent interrupt context for each vCPU across all possible context switches.



Trap-less Virtual Interrupt

Trap-less Virtual Interrupt – Timer Interrupt

- Setting the timer
  - The guest writes to *vstimecmp* via the alias *stimecmp* to set the timer
  - The host writes directly to *vstimecmp* to switch the vCPU context
- Interrupt delivery
  - In running state, V==1, *hip.VSTIP* is set, the guest handles it directly.
  - In paused state, V==0, *hip.VSTIP* is set, the host handles it, e.g. to allow priority adjustment for scheduling.
  - In scheduled off state, the host tracks the vCPU's timer using hrtimer, no timer virtual interrupt is fired.



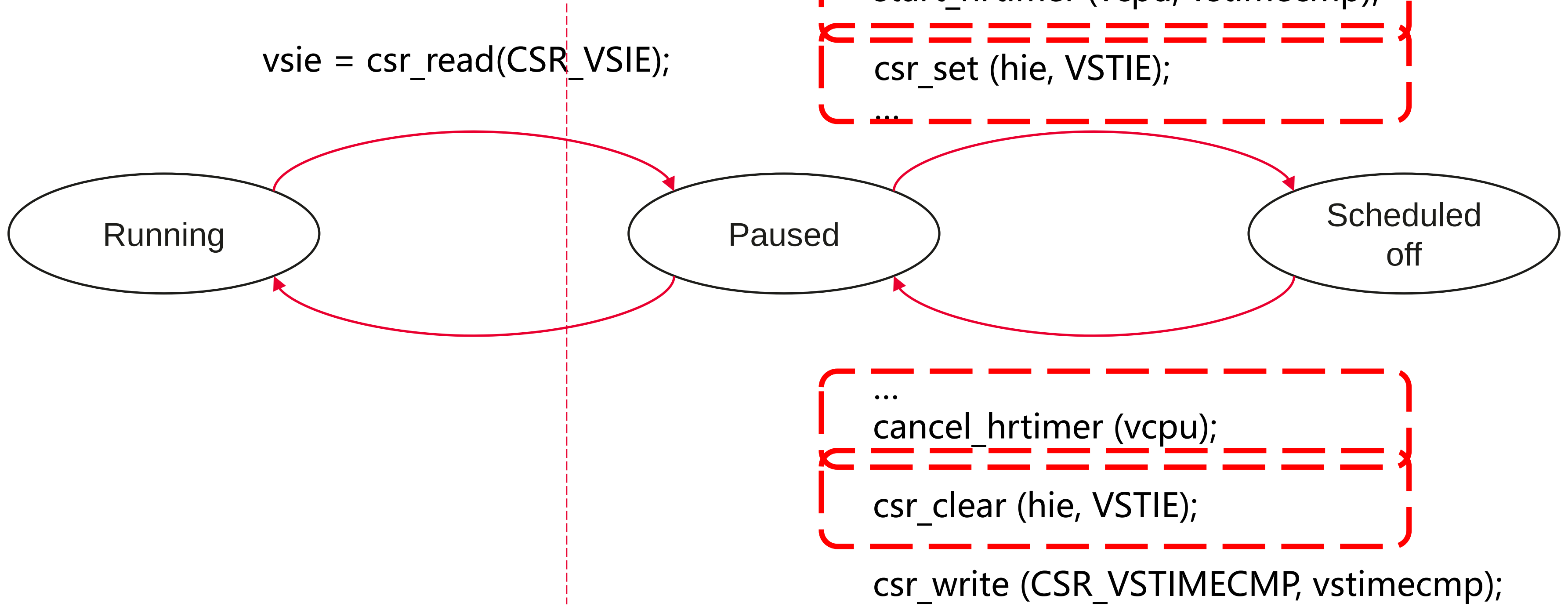
```
...
case IRQ_VSTIMER:

    csr_write (hie, VSTIE);

    kvm_riscv_cpu_inject (IRQ_VS_TIMER);
...
```

Trap-less Virtual Interrupt – Virtual Timer Interrupt

- vCPU context maintenance

```
...
csr_clear (hie, VSTIE);
vstimecmp = csr_read (CSR_VSTIMECMP);
csr_write (CSR_VSTIMECMP, -1);
start_hrtimer (vcpu, vstimecmp);
csr_set (hie, VSTIE);
...
```

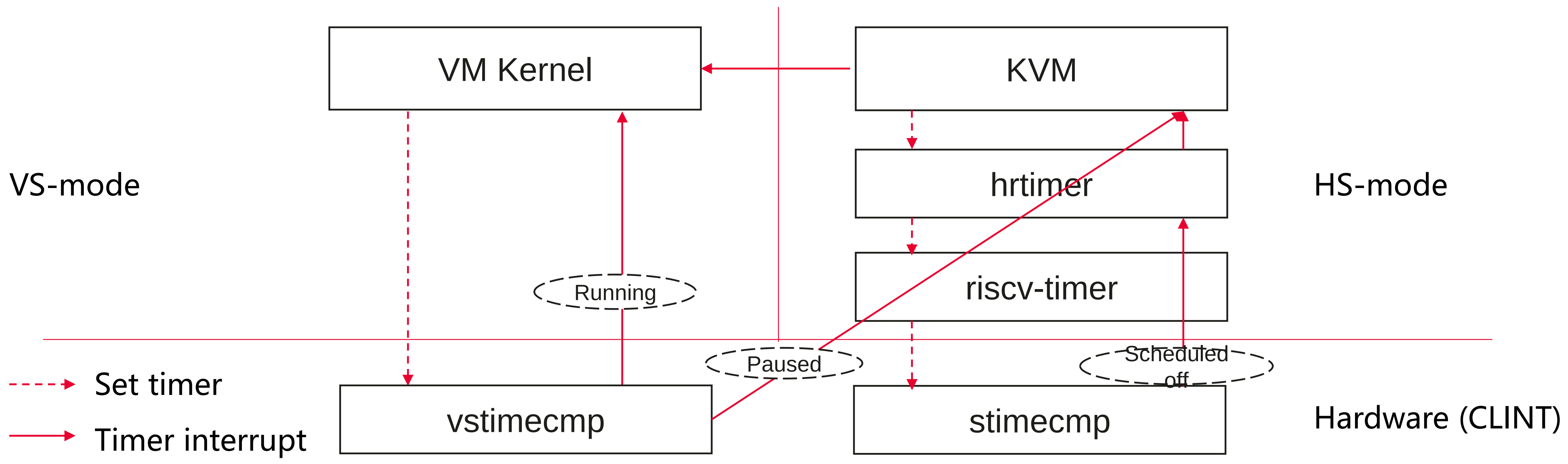
```
vsie = csr_read(CSR_VSIE);
```



```
...
cancel_hrtimer (vcpu);
csr_clear (hie, VSTIE);
csr_write (CSR_VSTIMECMP, vstimecmp);
csr_write (CSR_VSIE, vsie);
...
```

Trap-less Virtual Interrupt – Virtual Timer Interrupt

- The virtual timer interrupt flow with the new controller
- Bypasses the entire host when the vCPU is in running state
- Comparable to the existing approach when vCPU is not running

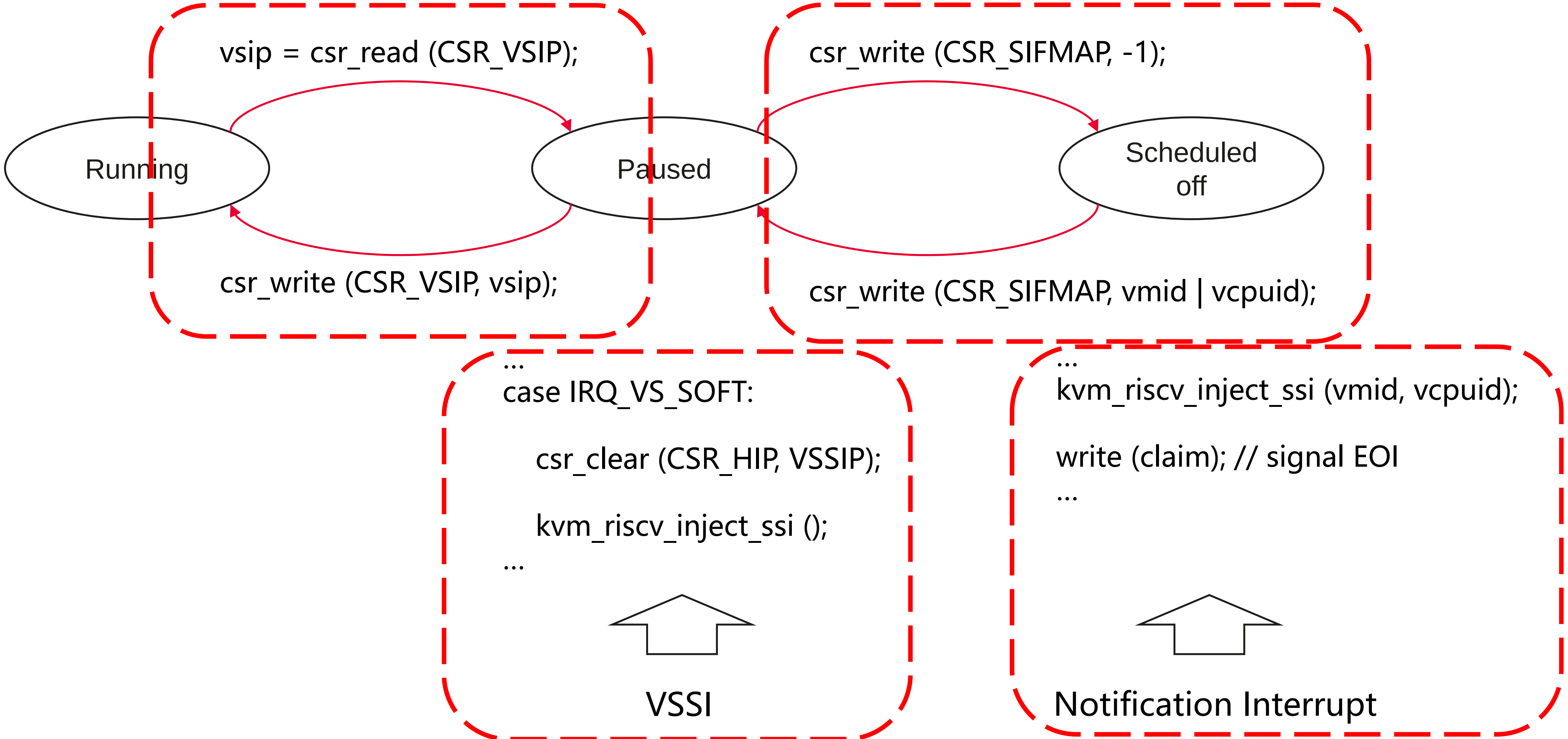


Trap-less Virtual Interrupt – VSSI

- Sending VSSI
  - A guest vCPU writes to *sgenipi* to send a VSSI to another vCPU in the same VM, specifying the vCPU ID.
- Routing VSSI
  - The PLIC looks up in ifmap[n] registers whether there is one that contains the specified vCPU ID
  - If found, deliver the VSSI to the hart with mhartid value n. The vCPU can be in either running state or paused state.
    - If running, take the interrupt directly by guest.
    - If paused, host takes the interrupt and injects it to the vCPU
  - If not found, deliver a notification interrupt to any host hart for injection. The vCPU must be in scheduled off state.

Trap-less Virtual Interrupt – VSSI

- Interrupt handling and vCPU context maintenance by host



```
vsip = csr_read (CSR_VSIP);
csr_write (CSR_VSIP, vsip);
```

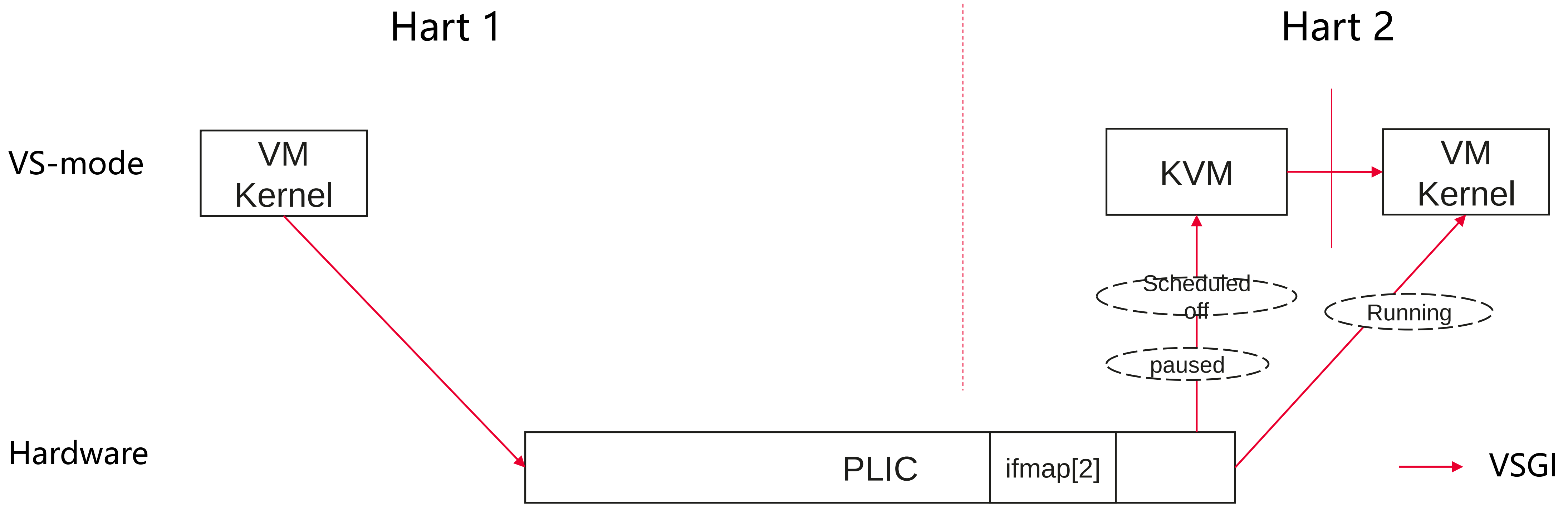
```
csr_write (CSR_SIFMAP, -1);
csr_write (CSR_SIFMAP, vmid | vcpuid);
```

```
...
case IRQ_VS_SOFT:
    csr_clear (CSR_HIP, VSSIP);
    kvm_riscv_inject_ssi ();
...
```

```
...
kvm_riscv_inject_ssi (vmid, vcpuid);
write (claim); // signal EOI
...
```

Trap-less Virtual Interrupt – VSSI

- The virtual software-generated interrupt flow with the new controller

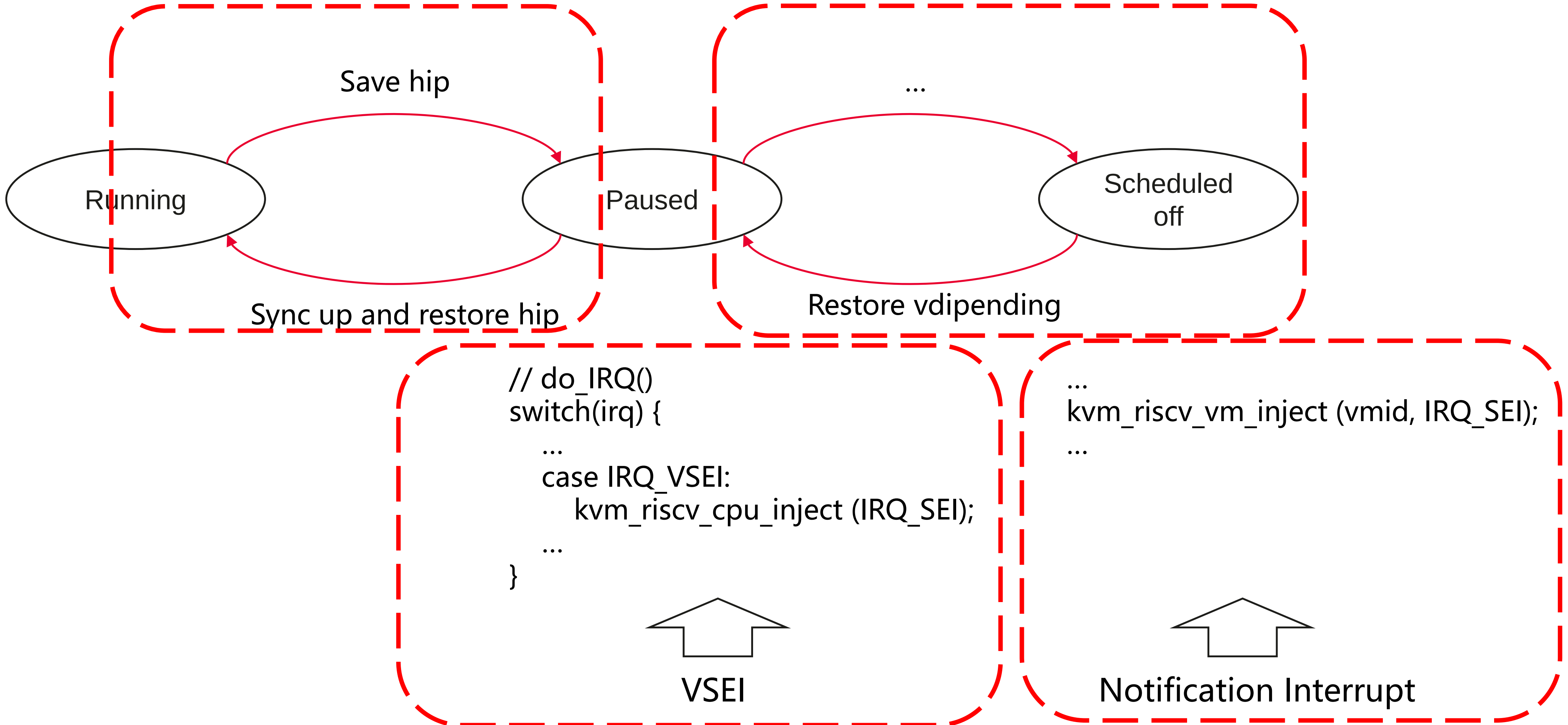


Trap-less Virtual Interrupt – VSEI

- VM Creation
  - The host allocates memory for pending bits for the VM
- Sending VSEI
  - Backend driver in the host’s user space writes to ugenvsei CSR to send an external device interrupt to the guest
- Delivering VSEI
  - The PLIC looks up the virtual interrupt affinity tables pointed to by the *vtblbase[n]* registers and find the vCPU that this virtual interrupt should be delivered to.
  - The PLIC looks up in ifmap[n] registers whether there is one that contains the specified vCPU ID
  - If found, deliver the VSEI to hart with mhartid n. The vCPU can be in either running state or paused state.
    - If running, take the interrupt directly by guest.
    - If paused, host takes the interrupt and injects an SEI to the vCPU
  - If not found, deliver an notification interrupt to any host hart for injection. The vCPU must be in scheduled off state.
- VSEI Claim and EOI
  - The guest directly read from and write to the vsclaim register to claim and signal end-of-interrupt during handling of the VSEI.

Trap-less Virtual Interrupt – VSEI

- Interrupt handling and vCPU context maintenance

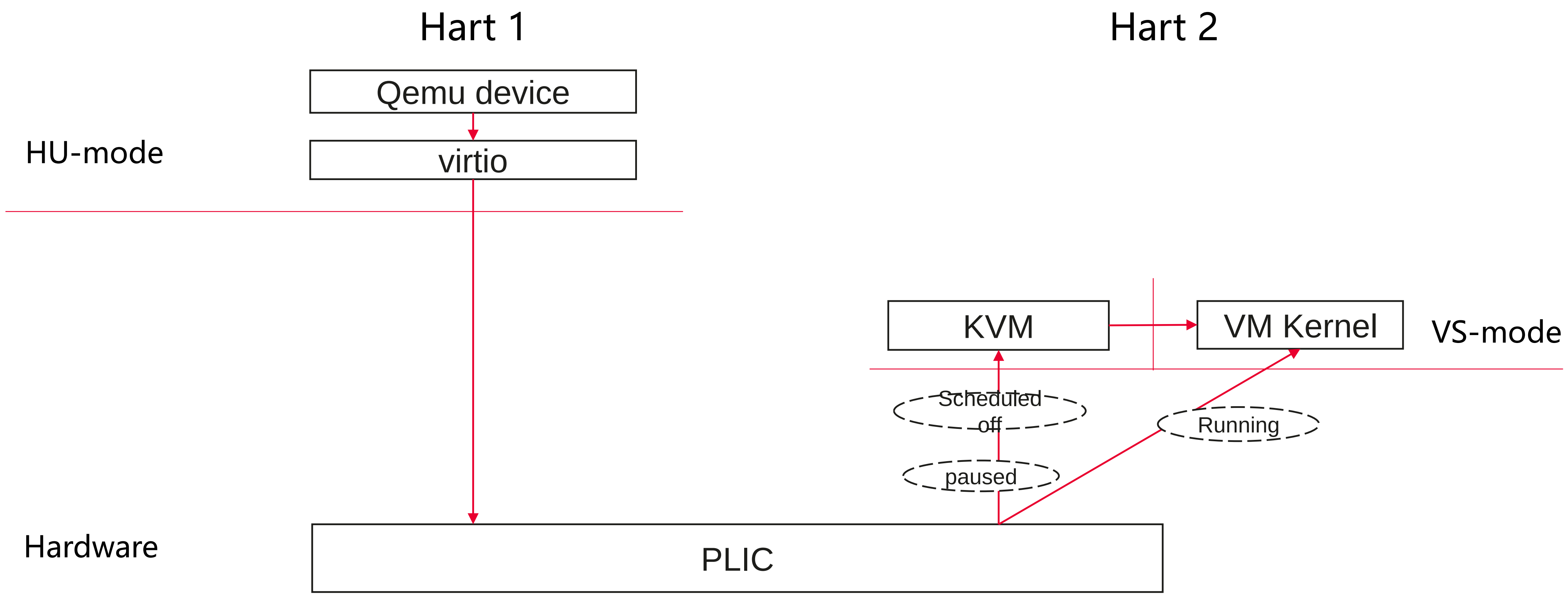


```
// do_IRQ()
switch(irq) {
    ...
    case IRQ_VSEI:
        kvm_riscv_cpu_inject (IRQ_SEI);
    ...
}
```

```
...
kvm_riscv_vm_inject (vmid, IRQ_SEI);
...
```

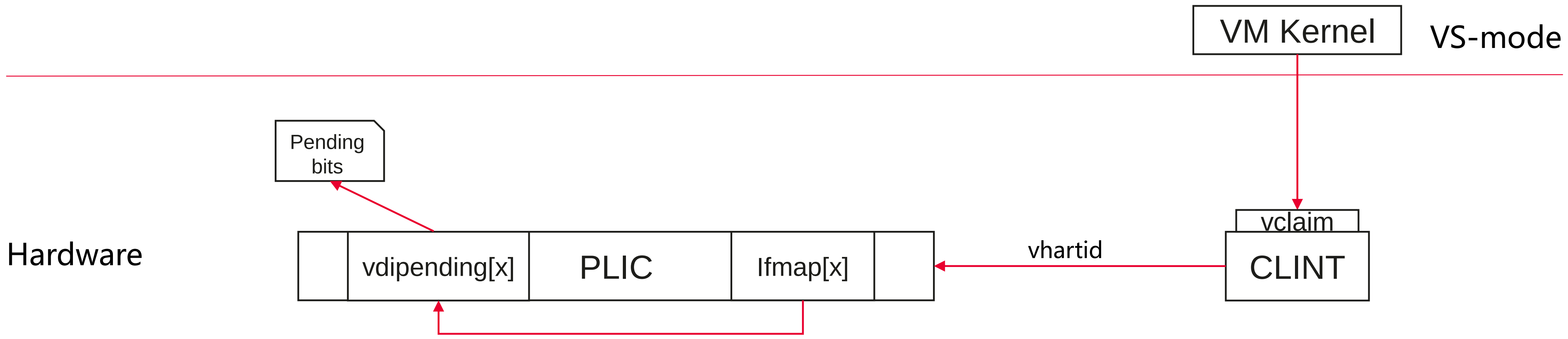
Trap-less Virtual Interrupt – VSEI

- The VSEI sending flow with the new controller



Trap-less Virtual Interrupt – VSEI

- The VSEI claim flow with the new controller

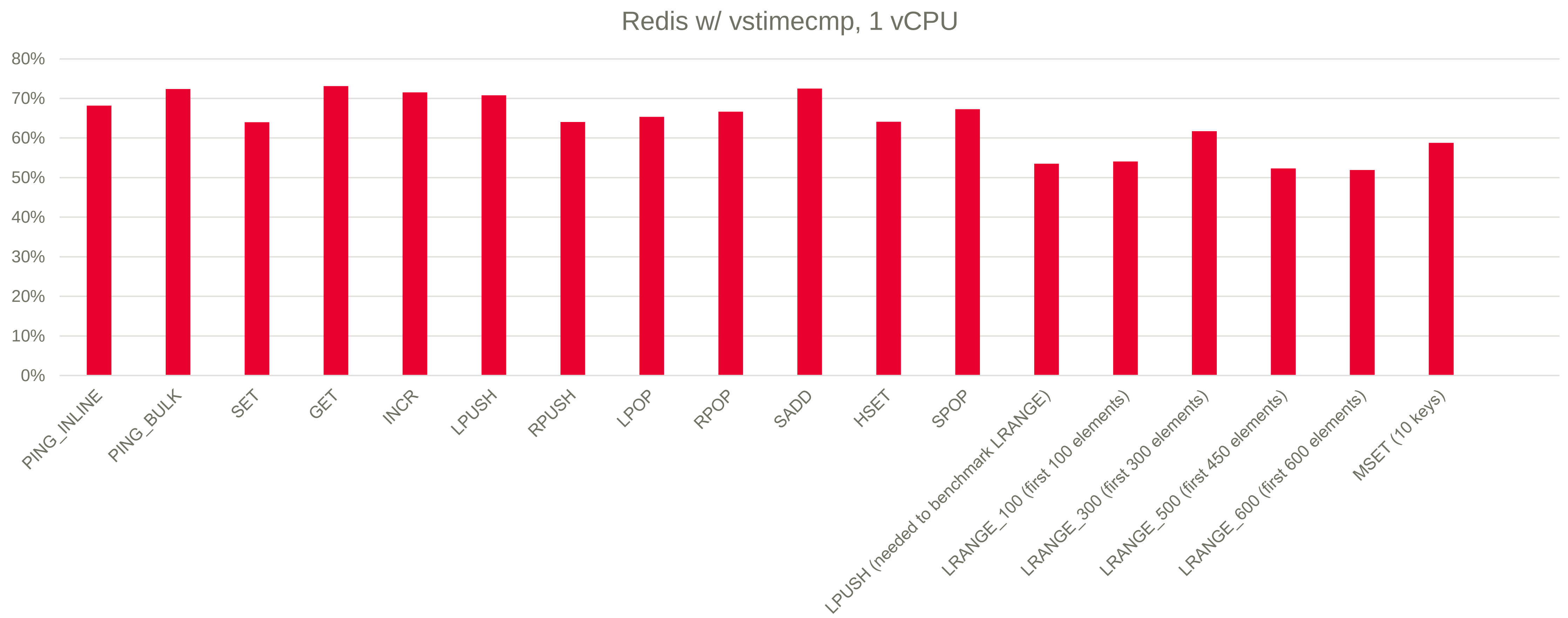


Implementation & Results

Implementation

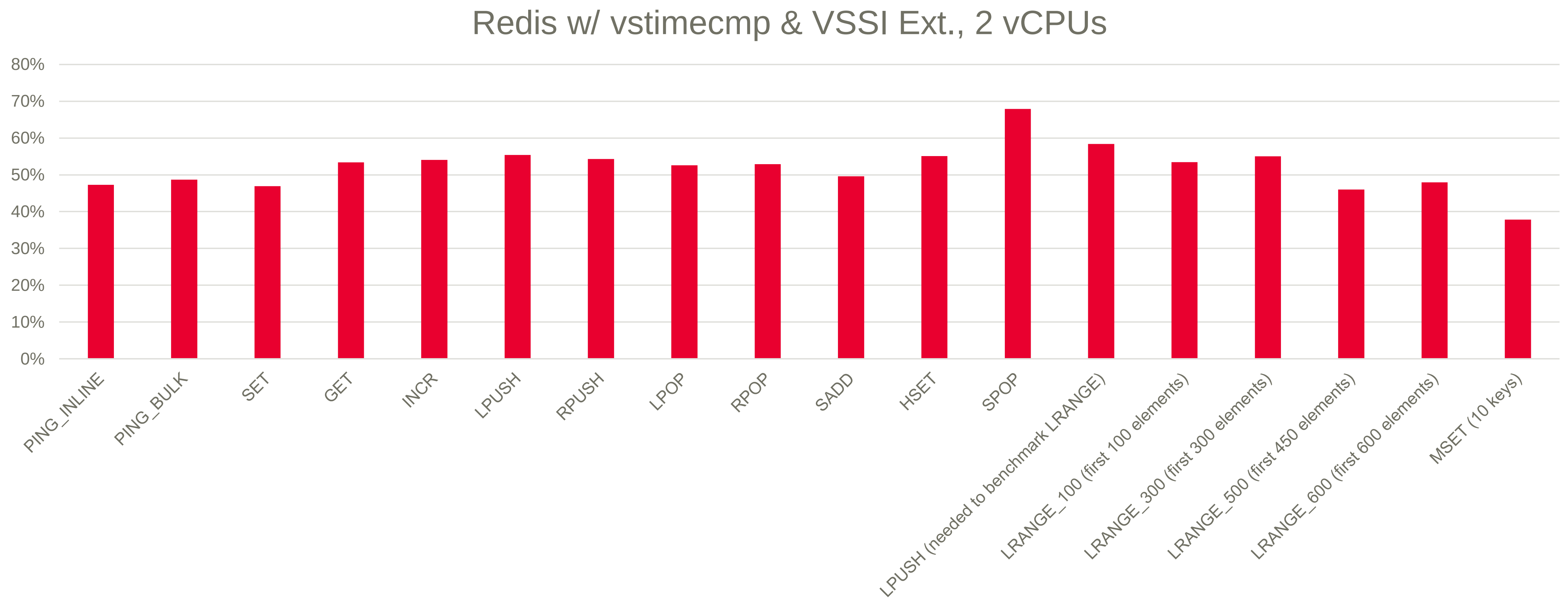
- We have implemented the above interrupt extension in RISC-V QEMU v5, which provides an emulated RISC-V environment with the hypervisor extension.
- We have implemented necessary KVM support.
- We ran benchmarks that can be compiled and ran in the emulator with reasonable effort, and obtained some performance figures.
  - We managed to compile and run Redis and UnixBench

Benchmark Results – Virtual Timer Interrupts



- Performance is compared to *mtimecmp*-based timer
- Similar results are observed for UnixBench

Benchmark Results – Timer & VSSI



- Performance is compared to *mtimecmp*-based timer and the original SBI-based IPI

Benchmark Results – VSEI

- We ping the virtual machine 100,000 times
- Ping latency is reduced 11% on average
- Traps due to MMIO are reduced by ~300,000 times, as expected.

Future Work

- Pass-through device interrupts: need IOMMU
- More design details need to be iron out
  - Priority controls, secure virtual interrupts etc.
- Integration with future RISC-V interrupt controllers
- Validate the design on more hypervisors

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI