Keeping an Eye on Virtualization

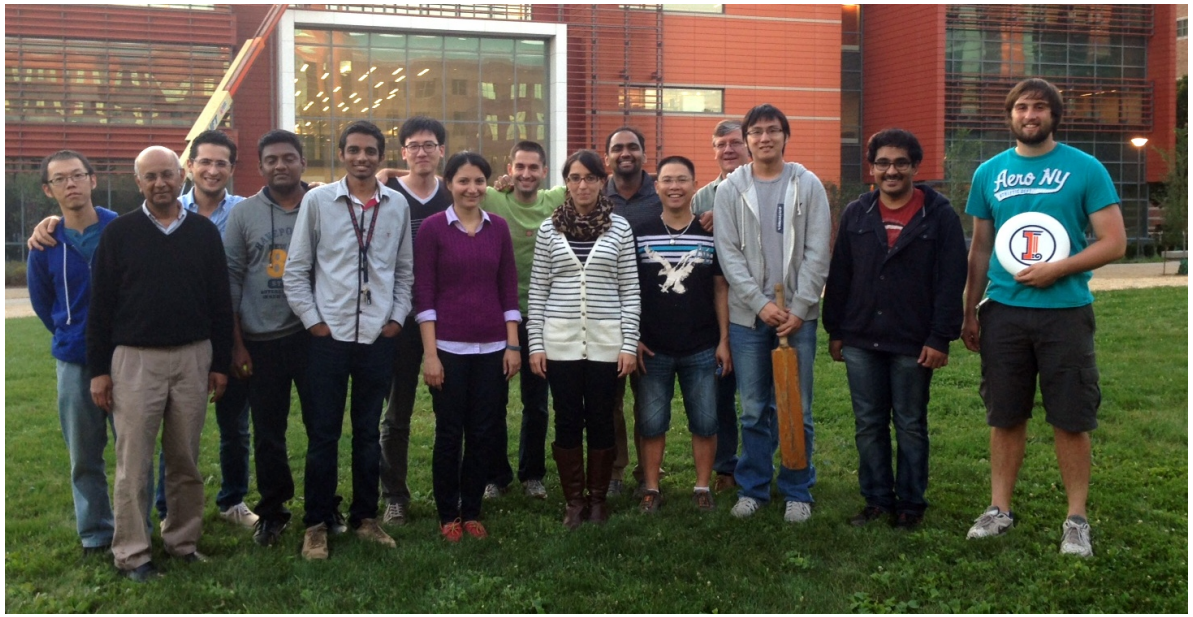
VM Monitoring Techniques & Applications

Zak Estrada
Gary Wang
Dr. Zbigniew Kalbarczyk

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

Who are these guys?

The DEPEND group focuses on the research, design and validation of highly-available, reliable and trustworthy computing systems and networks.



http://depend.csl.illinois.edu/

The Plan

8am - 9:15am

- Key Concepts
- Validation

15min break

9:30am-11am

- Virtual Machine Introspection
- Hardware Invariant Techniques

15min break

11:15am-noon

- Hook-based Monitoring

The Plan (cont.)

- All sections will end with discussion
- Ask questions at any time

Key Concepts: Virtualization

Virtual Machines

"VMs have finally arrived. Dismissed for a number of years as academic curiosities, they are now cost-effective techniques for organizing computer resources to provide system flexibility and support for unique applications."

Virtual Machines

"VMs have finally arrived. Dismissed for a number of years as academic curiosities, they are now cost-effective techniques for organizing computer resources to provide system flexibility and support for unique applications."

– Robert Goldberg, 1974

Virtual Machines

- Main idea: single machine $\Rightarrow$ many
- Only emulate privileged instructions

Virtualization Requirements[1]

[1] Popek and Goldberg, “Formal Requirements for Virtualizable Third Generation Architectures,” 1974

Virtualization Requirements[1]

- Efficiency
  - Most instructions execute natively

[1] Popek and Goldberg, “Formal Requirements for Virtualizable Third Generation Architectures,” 1974

Virtualization Requirements[1]

- Efficiency
  - Most instructions execute natively
- Equivalence
  - VM Behavior = Bare Metal Behavior

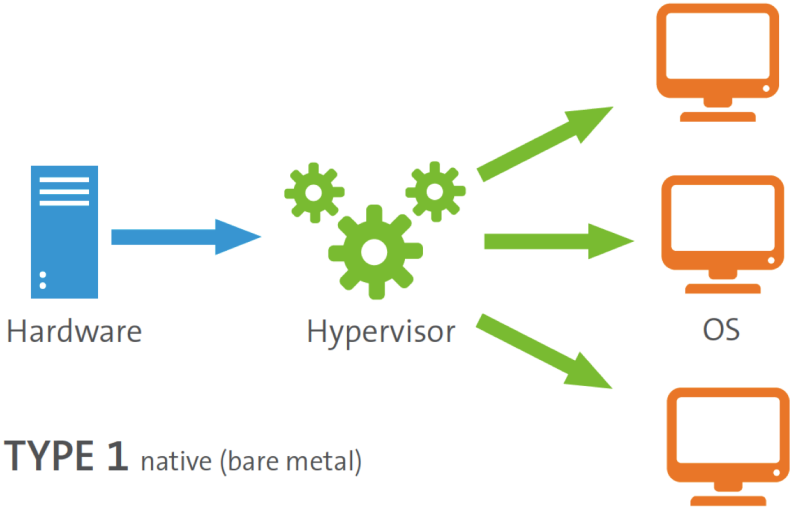
[1] Popek and Goldberg, “Formal Requirements for Virtualizable Third Generation Architectures,” 1974

Virtualization Requirements[1]

- Efficiency
  - Most instructions execute natively
- Equivalence
  - VM Behavior = Bare Metal Behavior
- Resource control
  - VMM has complete control

[1] Popek and Goldberg, “Formal Requirements for Virtualizable Third Generation Architectures,” 1974

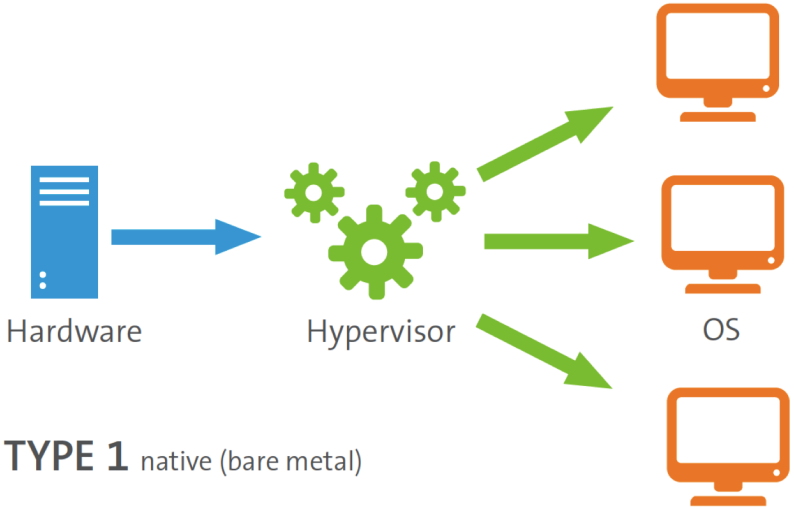
Types of hypervisors[2]



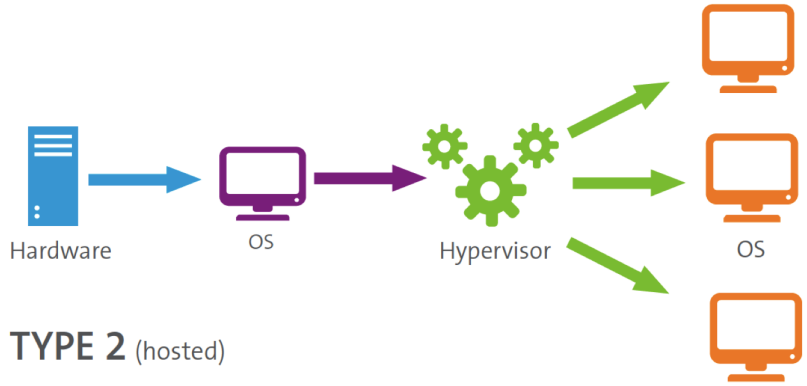
Xen, VMWare ESX

[2] http://www.flexiant.com/2014/02/05/what-does-a-hypervisor-do/

Types of hypervisors[2]



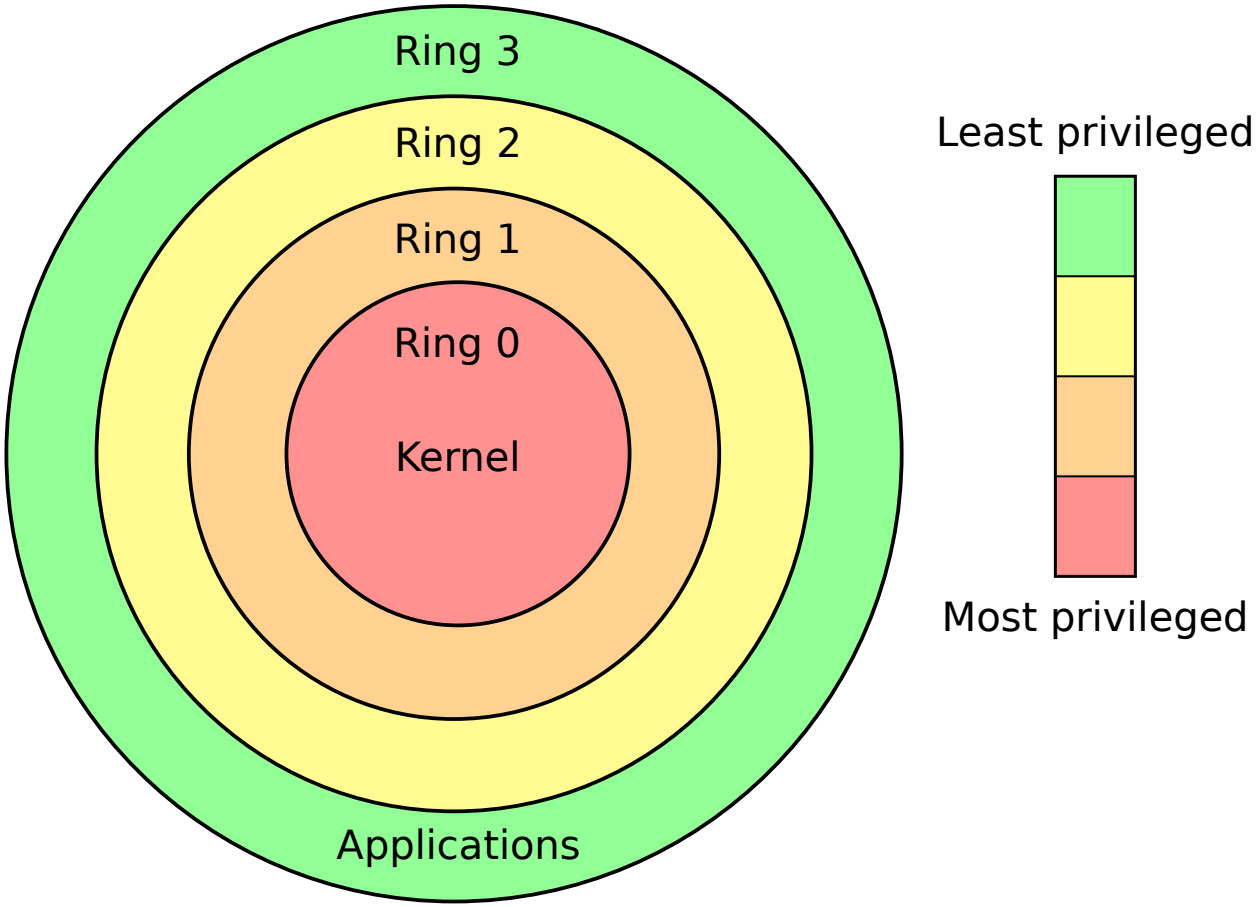
Xen, VMWare ESX



Virtualbox, QEMU

[2] http://www.flexiant.com/2014/02/05/what-does-a-hypervisor-do/

x86 Privilege Rings



[3]

[3] http://en.wikipedia.org/wiki/Protected_mode#Privilege_levels

Virtualization on x86

x86: unfriendly to virtualization

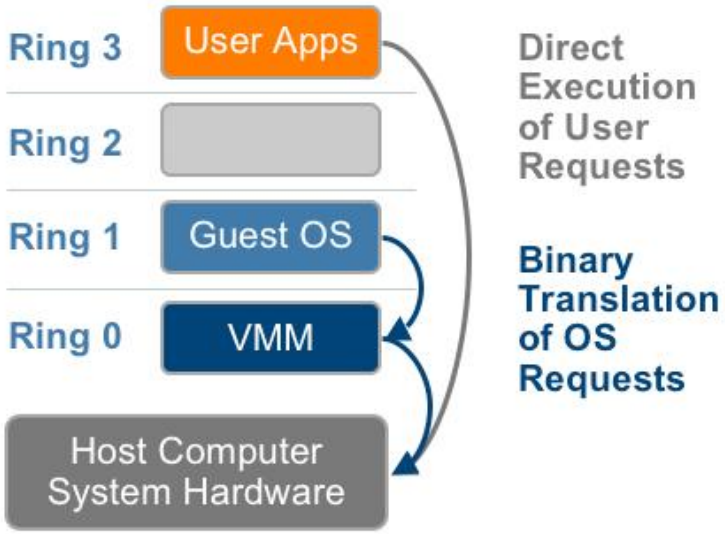
- Not all priv. ops generate traps (e.g. `popf`)
- No MMU virtualization
- 1998 VMware: binary translation

[4] http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

Virtualization on x86

x86: unfriendly to virtualization

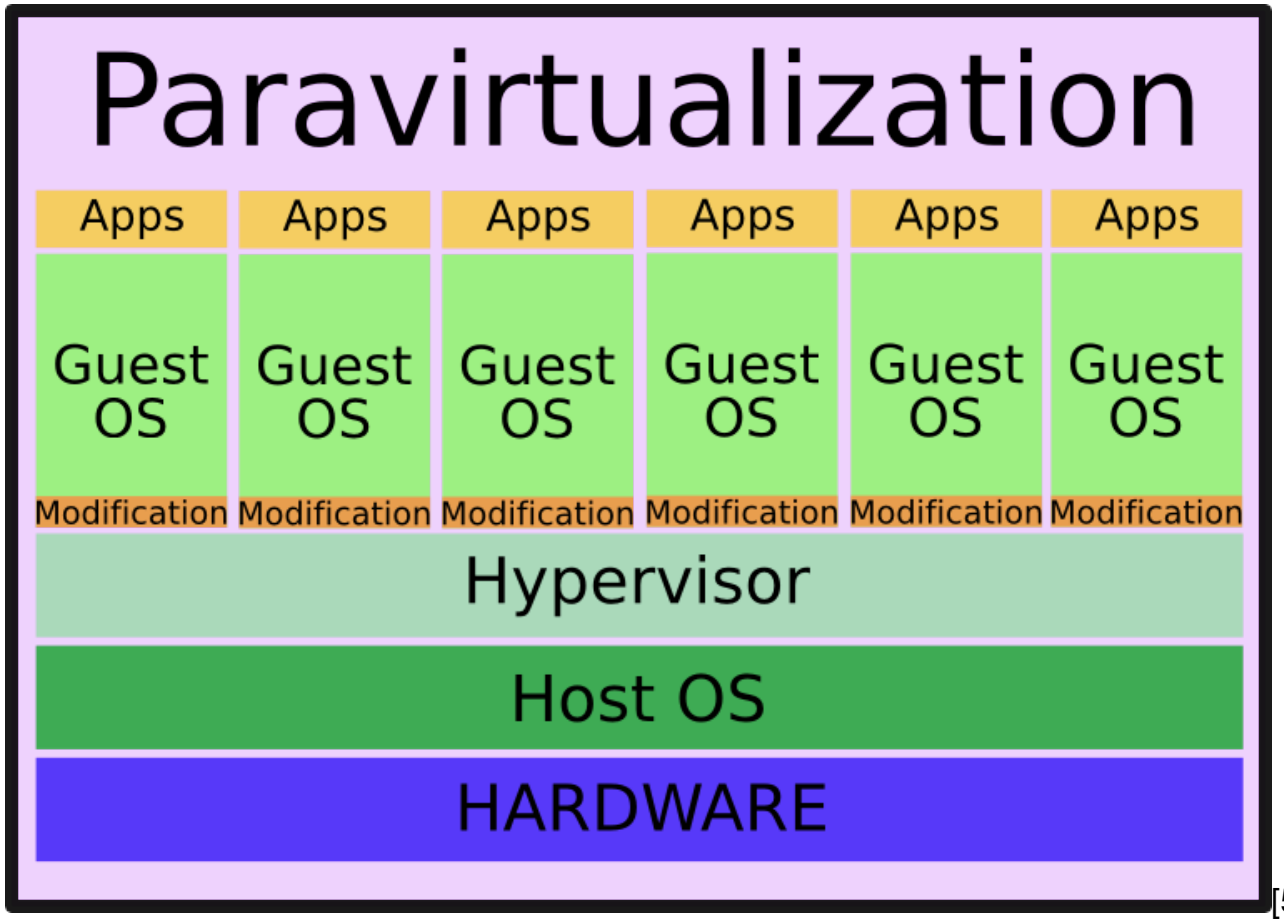
- Not all priv. ops generate traps (e.g. `popf`)
- No MMU virtualization
- 1998 VMware: binary translation



[4]

[4] `http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf`

Paravirtualization



[5]

[5] http://voz.vn/wp-content/uploads/2009/08/paravirtualization-copy.jpg

Paravirtualization

Question

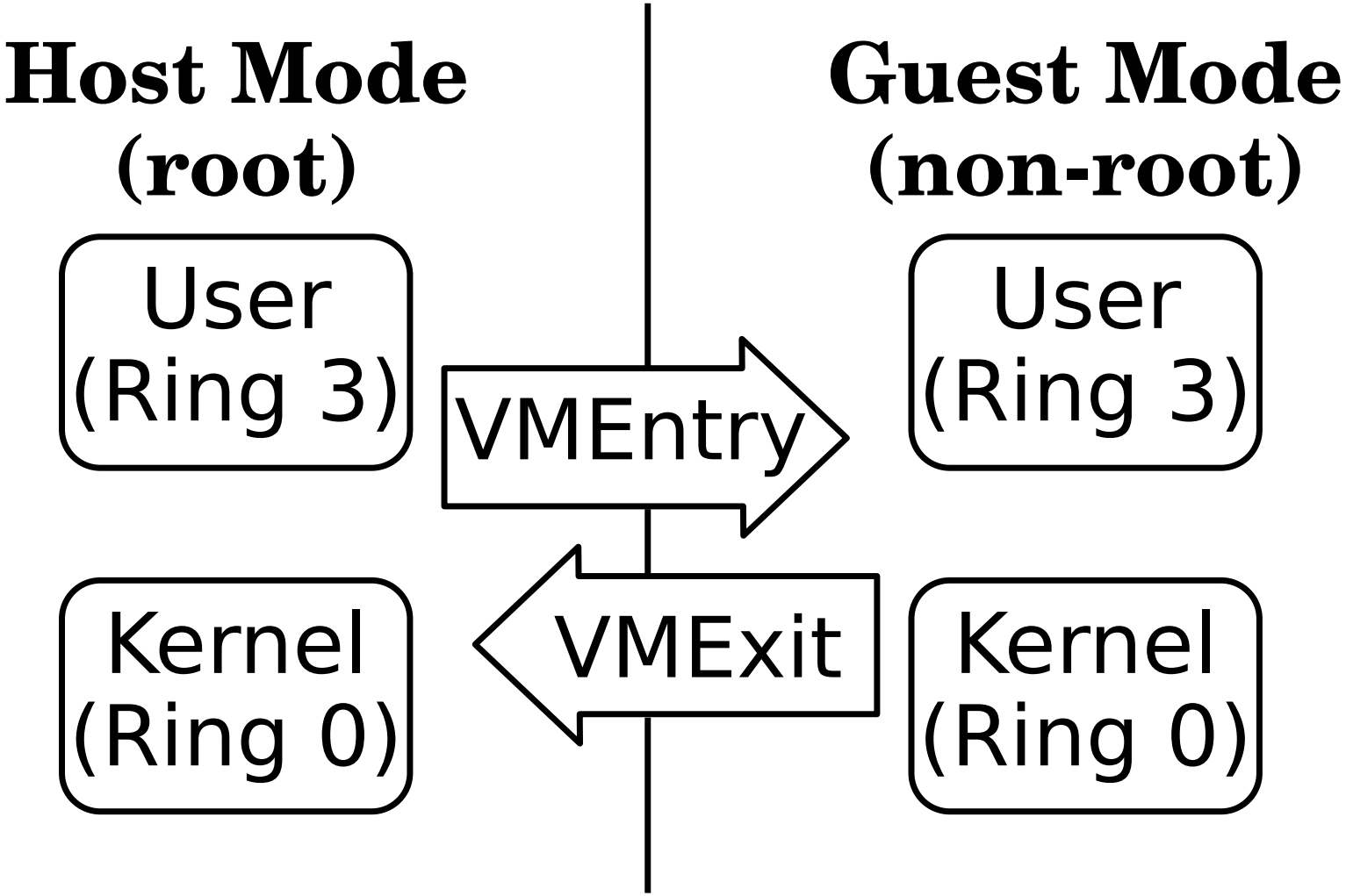
What are the advantages/disadvantages of Paravirtualization?

Hardware Assisted Virt.

In 2006, Intel VT-x and AMD AMD-V $\Rightarrow$ VMM much easier to implement

- Allowed for an unmodified guest OS
- Guest-mode/Host-mode
- **VM Entry** $\rightarrow$ Guest mode
- **VM Exit** on priv. operations
- Added instructions, e.g. `vmlaunch`/`vmcall`

Hardware Assisted Virt.



VM Entry/Exit Example

Guest mode

Host mode

VM Entry/Exit Example

Guest mode

Host mode

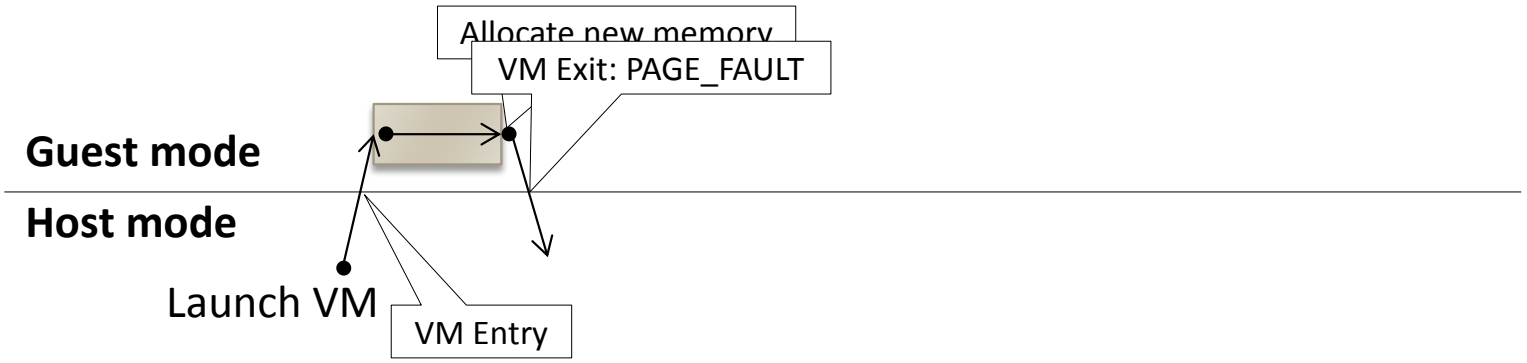
Launch VM

VM Entry/Exit Example

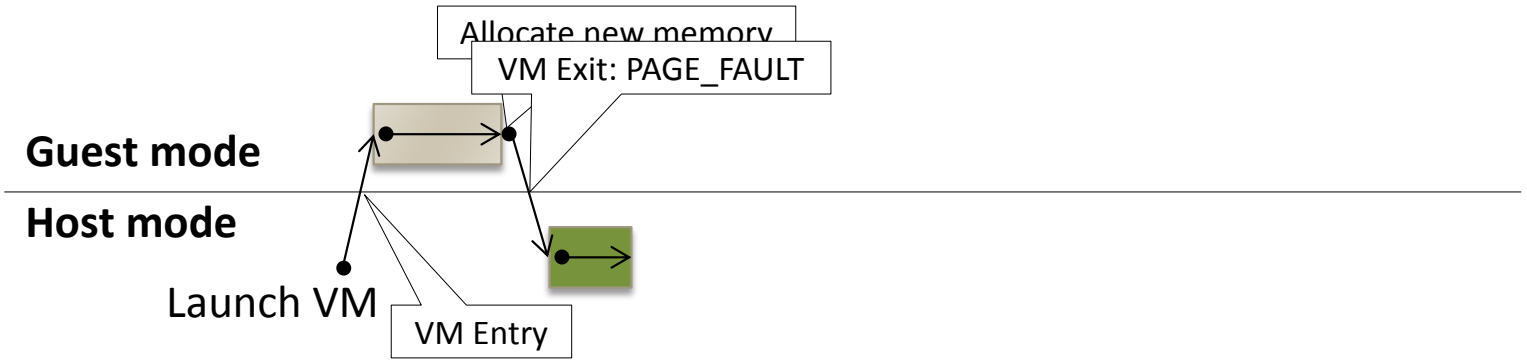
VM Entry/Exit Example

VM Entry/Exit Example

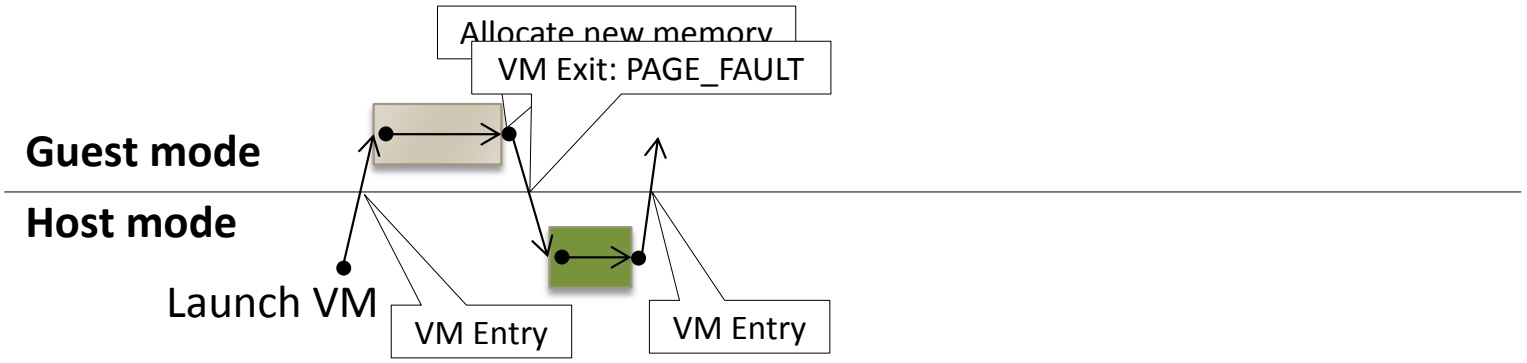
VM Entry/Exit Example



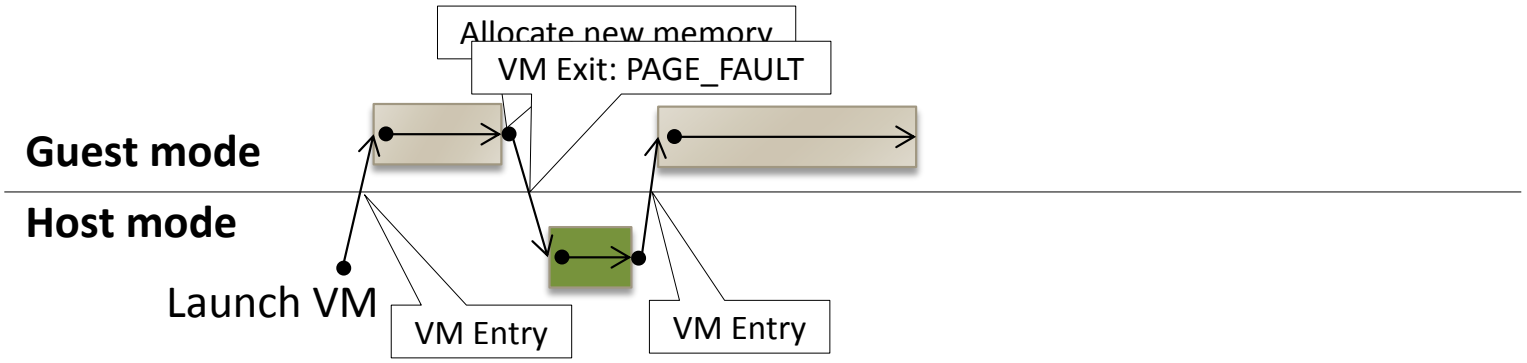
VM Entry/Exit Example



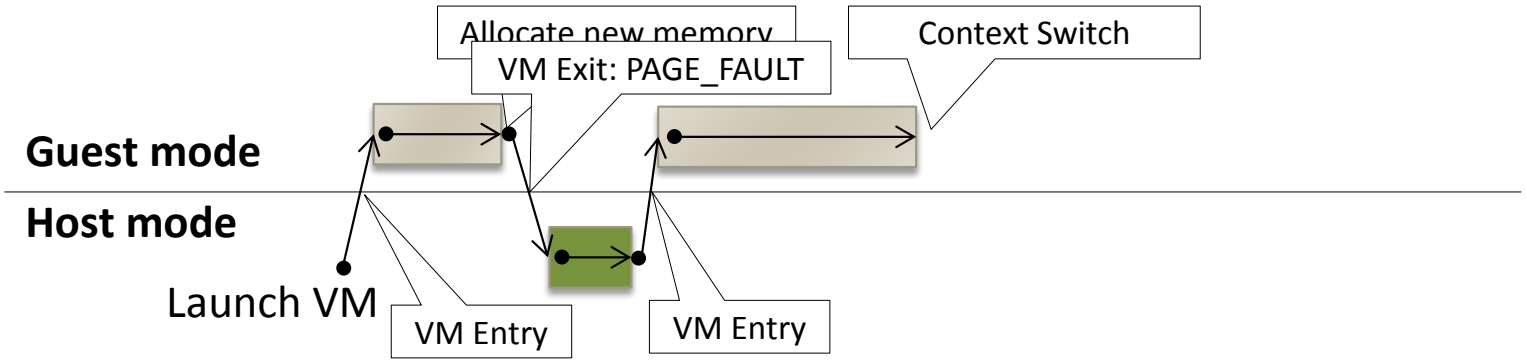
VM Entry/Exit Example



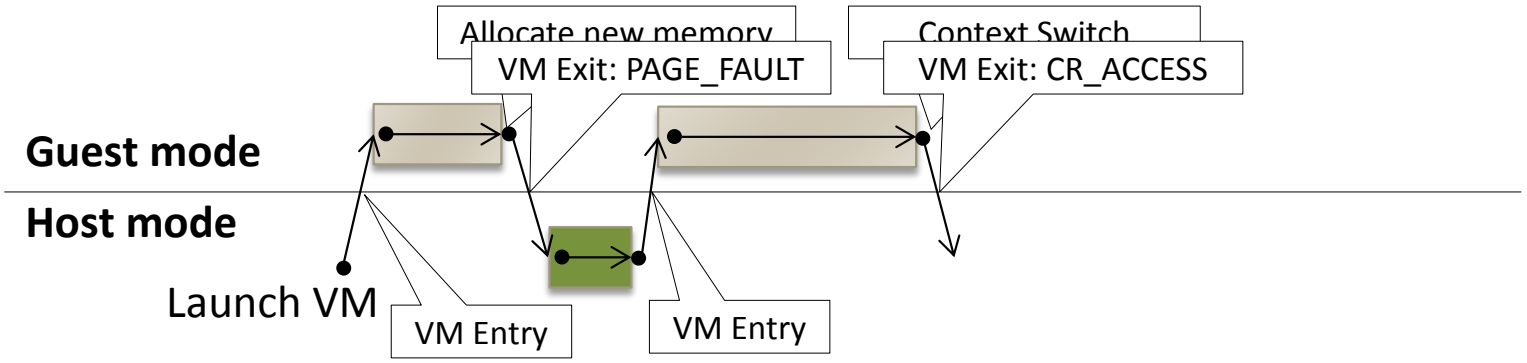
VM Entry/Exit Example



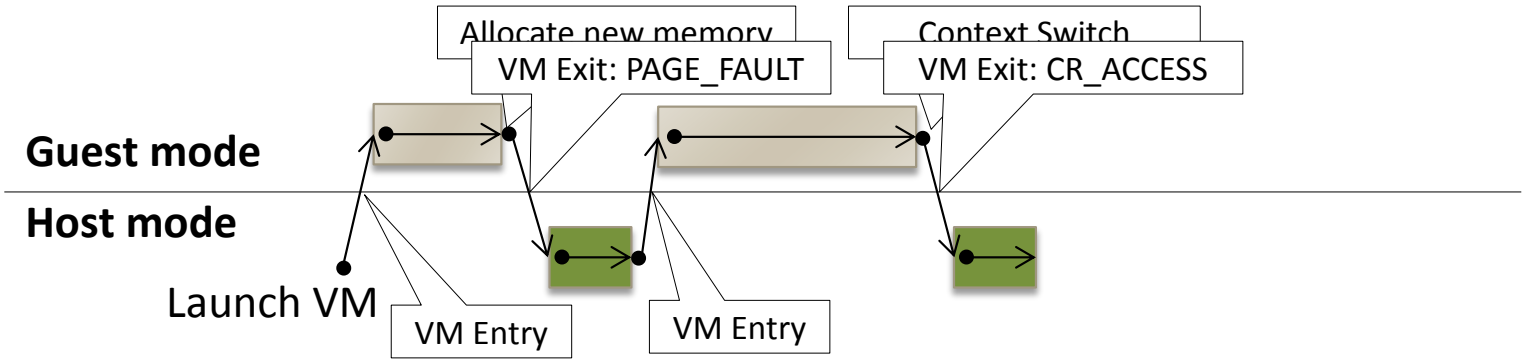
VM Entry/Exit Example



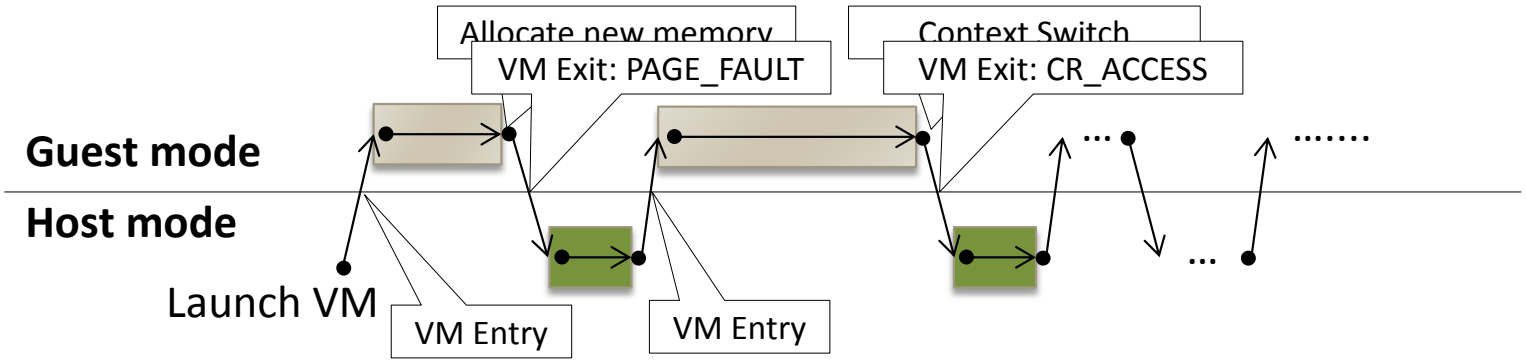
VM Entry/Exit Example



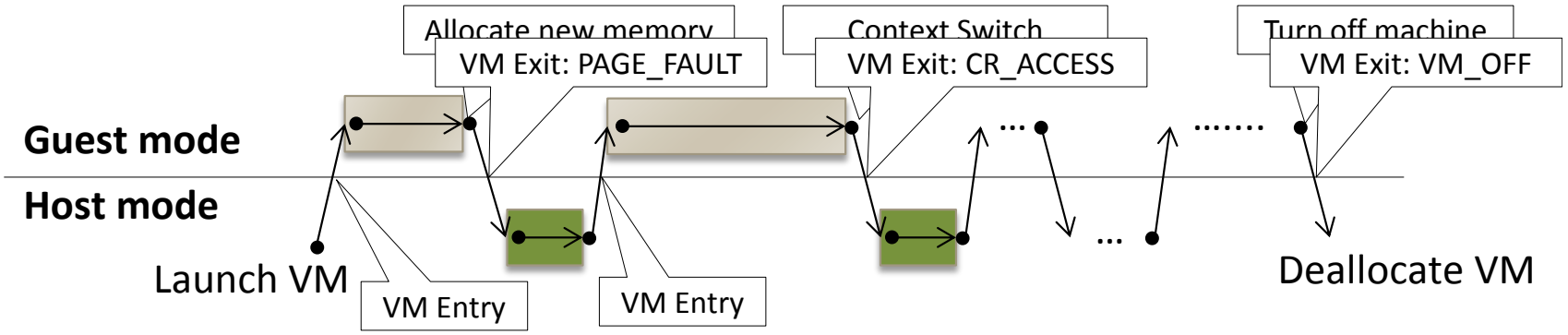
VM Entry/Exit Example



VM Entry/Exit Example

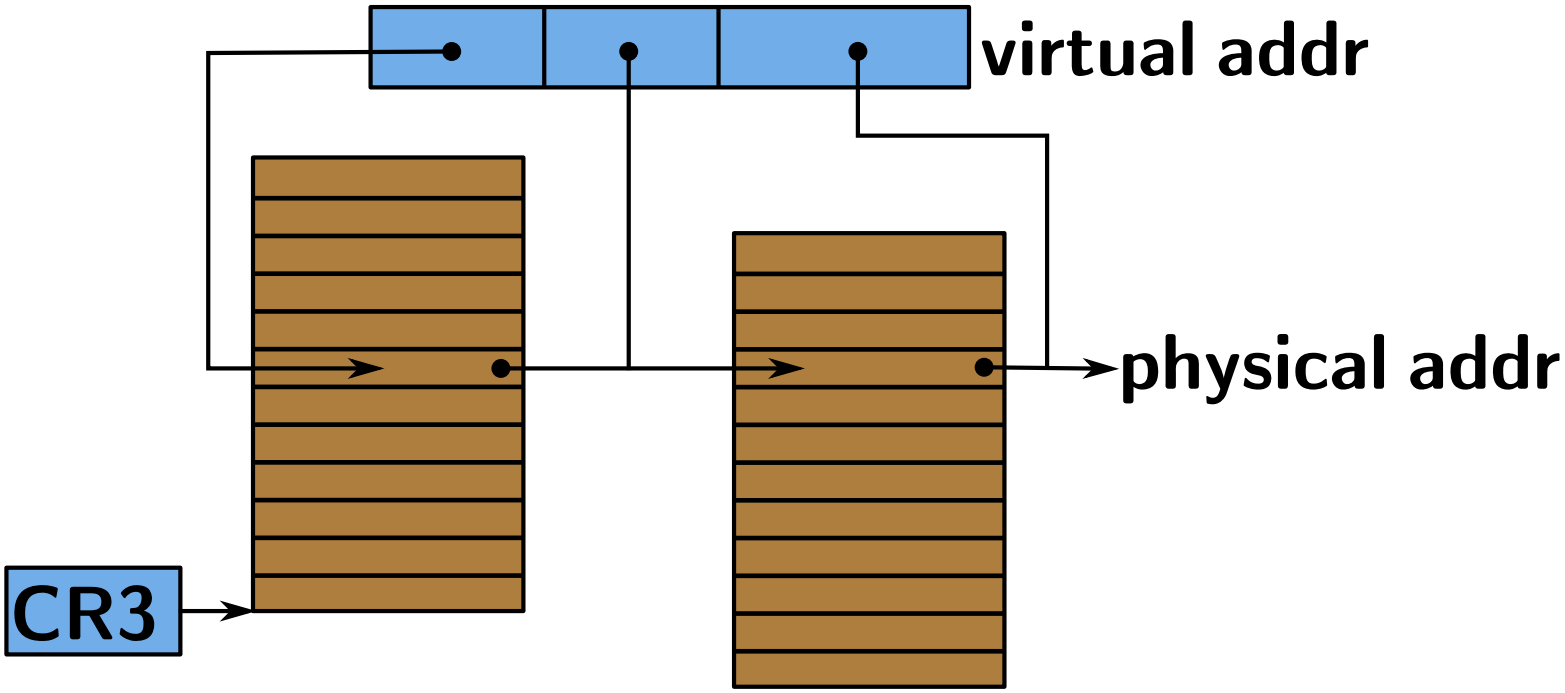


VM Entry/Exit Example



Paged Virtual Memory

Paged Virtual Memory



Shadow Page Tables

Question

Should we allow the VM to control page tables?

Shadow Page Tables

- Cannot let VM control page tables
  - But it still should think it does
- Use *Shadow* Tables in the Hypervisor
  - Update w/VM Exits on paging operations

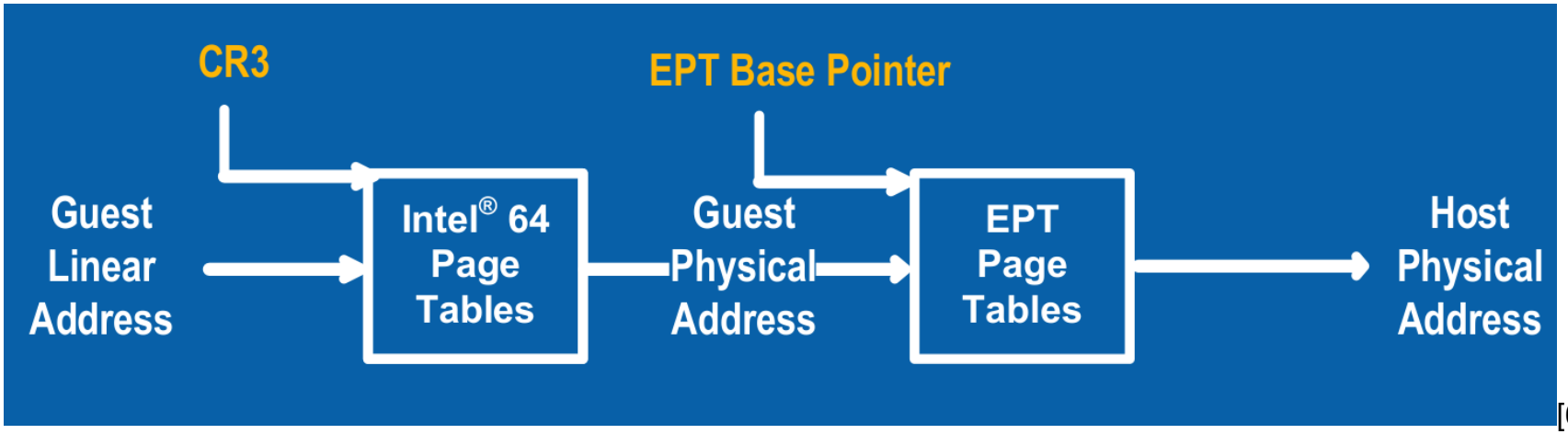
Shadow Page Tables

- Cannot let VM control page tables
  - But it still should think it does
- Use *Shadow* Tables in the Hypervisor
  - Update w/VM Exits on paging operations

Question

What are some drawbacks of Shadow Page Tables?

Second-Level Paging



[6]

- 2nd set of HW PTs for GPA→HPA
- Guest OS has full control over PTs

[6] http://www-archive.xenproject.org/files/xensummit_4/VT_roadmap_d_Nakajima.pdf

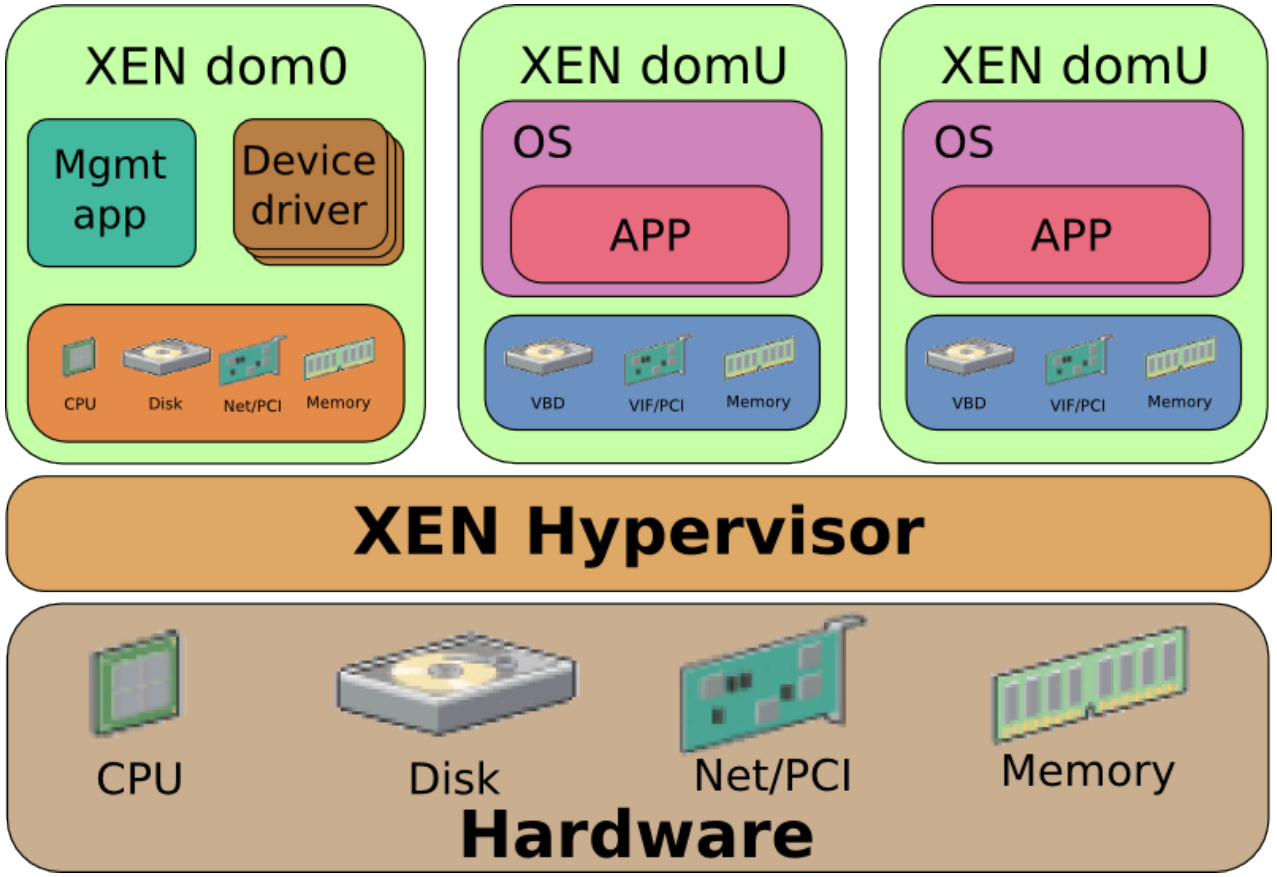
Hardware-Assisted Virtualization

Question

What are the advantages/disadvantages of Hardware-Assisted Virtualization?

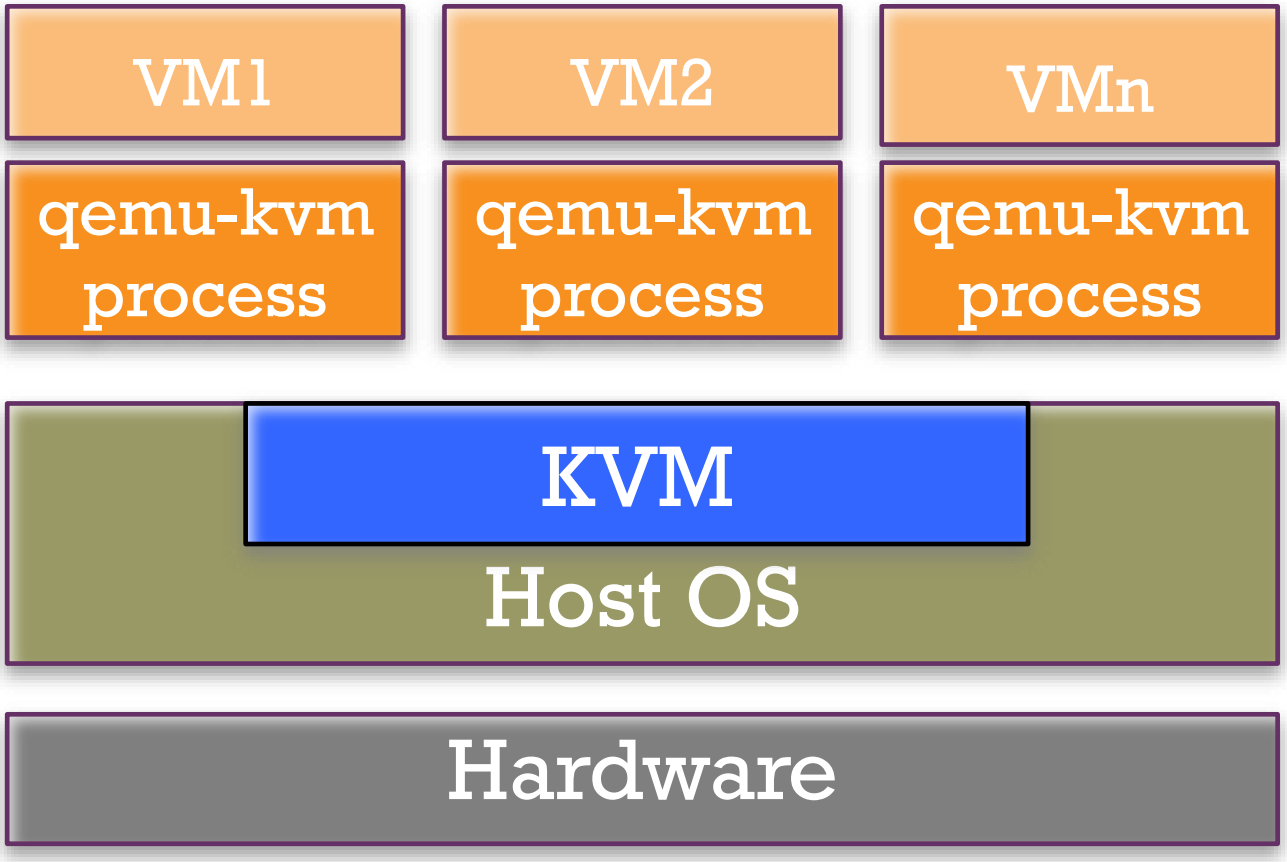
Open-Source Virtual Machine Monitors (VMMs)

Xen



- Amazon EC2

Kernel Virtual Machine



- Default for Openstack

Thoughts on Virtualization

- Virtualization has overhead
- Many VMs run only one application
- Many people running Linux on Linux (e.g. KVM w/Linux guest)

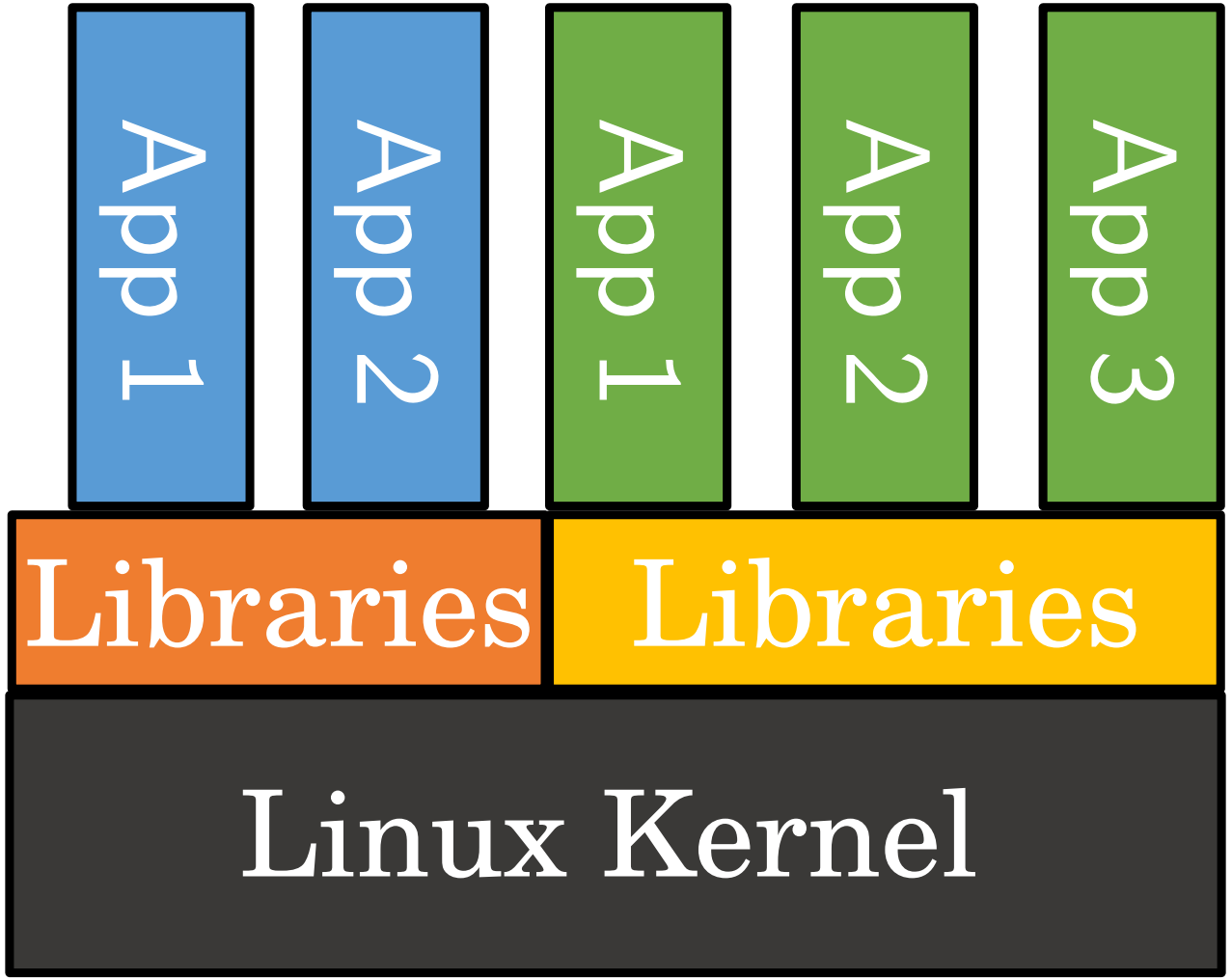
Question

Can we do better?

Linux Containers (LXC)

- Containers share kernel
  - no “trap and emulate”
- Isolation via cgroups/namespaces
- No HAV involved $\rightarrow$ always running in “host mode”

Linux Containers (LXC)



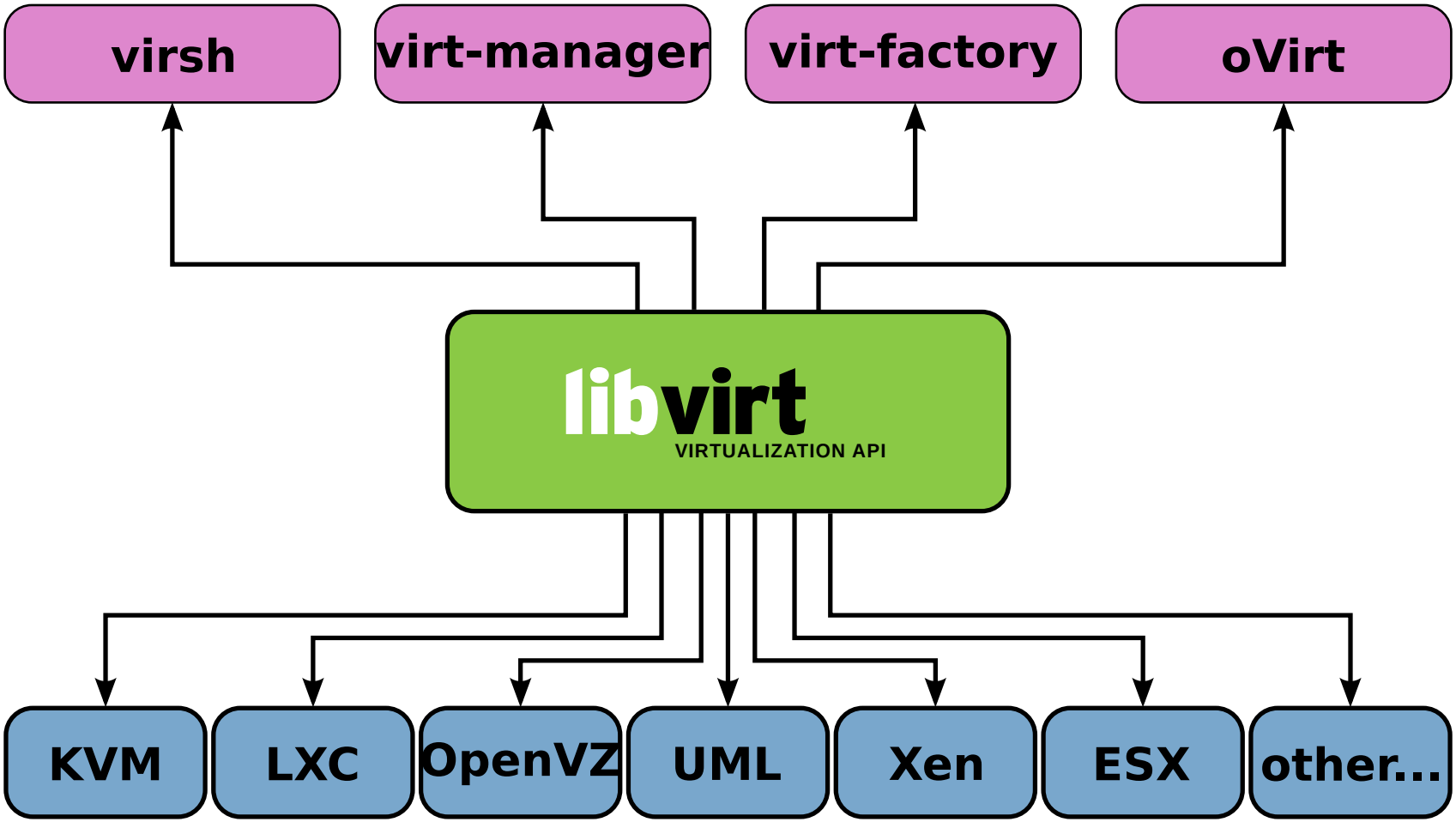
Linux Containers (LXC)

Question

What are the advantages/disadvantages of containers?

Managing VMs/Hypervisors

Managing VMs/Hypervisors



Virtualization Summary

Paravirtualization

- Xen

HAV

- VMExit
- Paging/EPT
- KVM

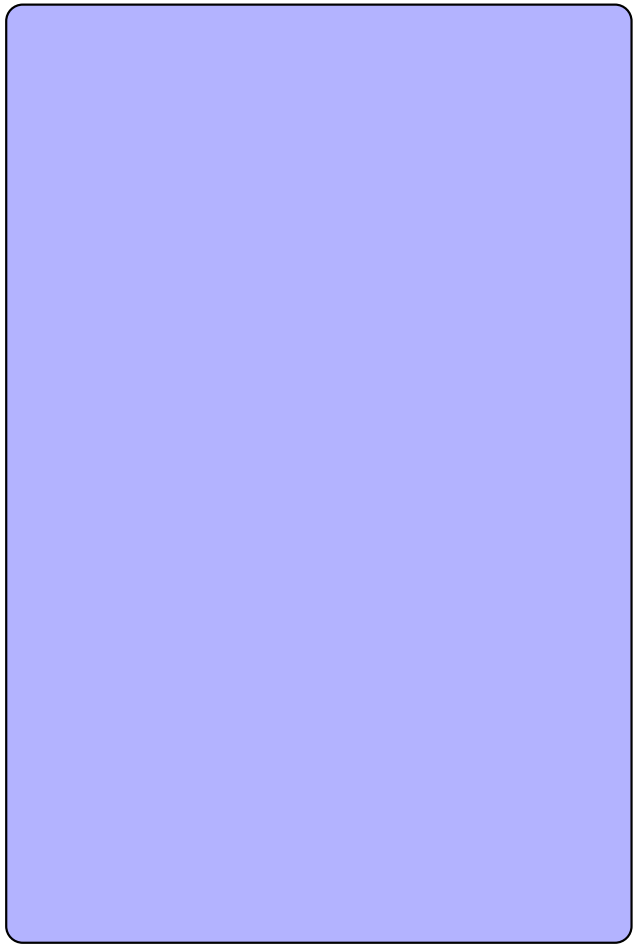
Lightweight Replacement

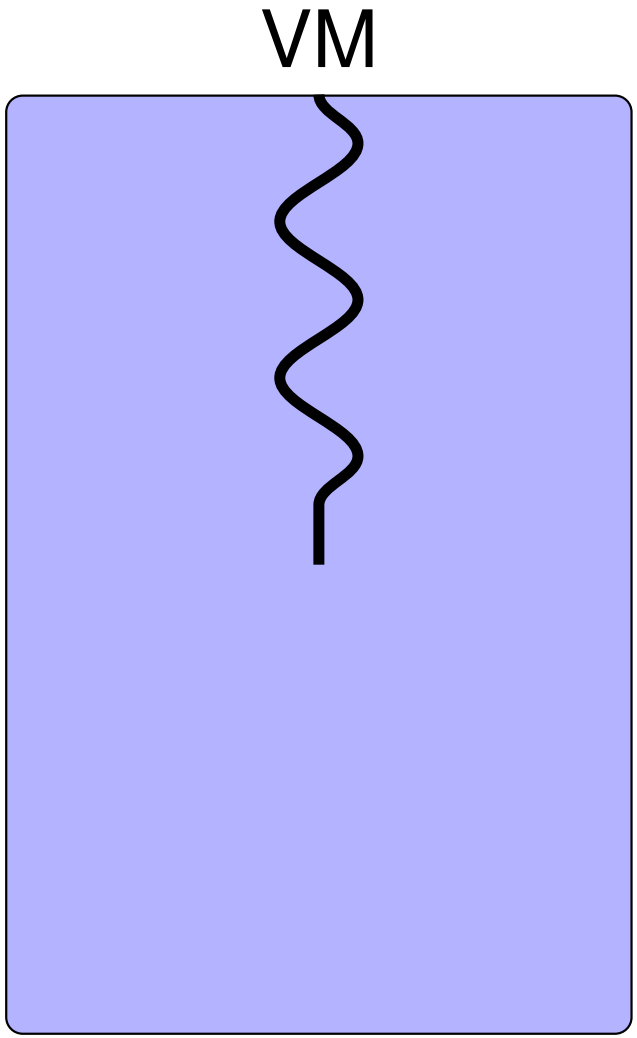
- Linux Containers

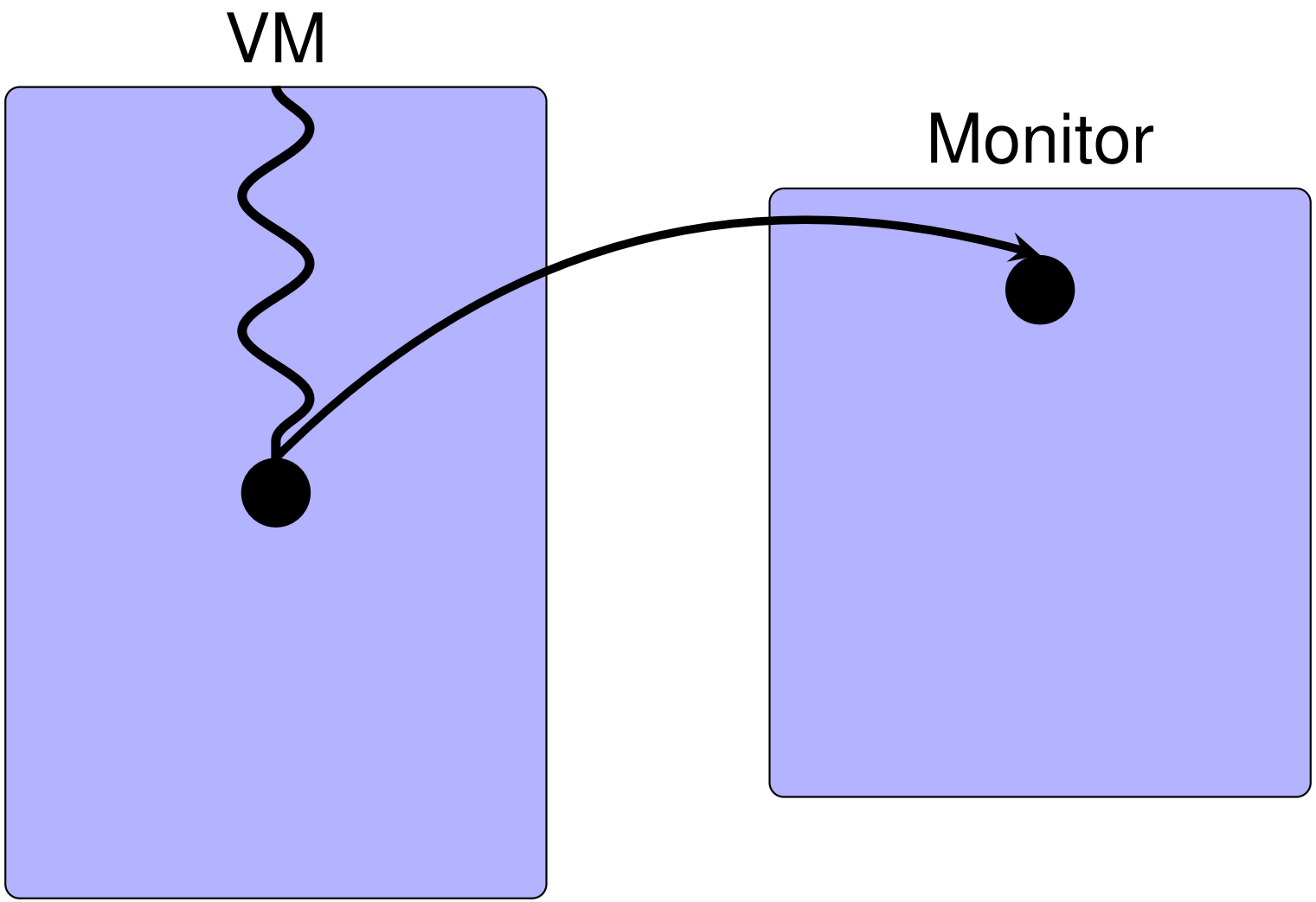
libvirt

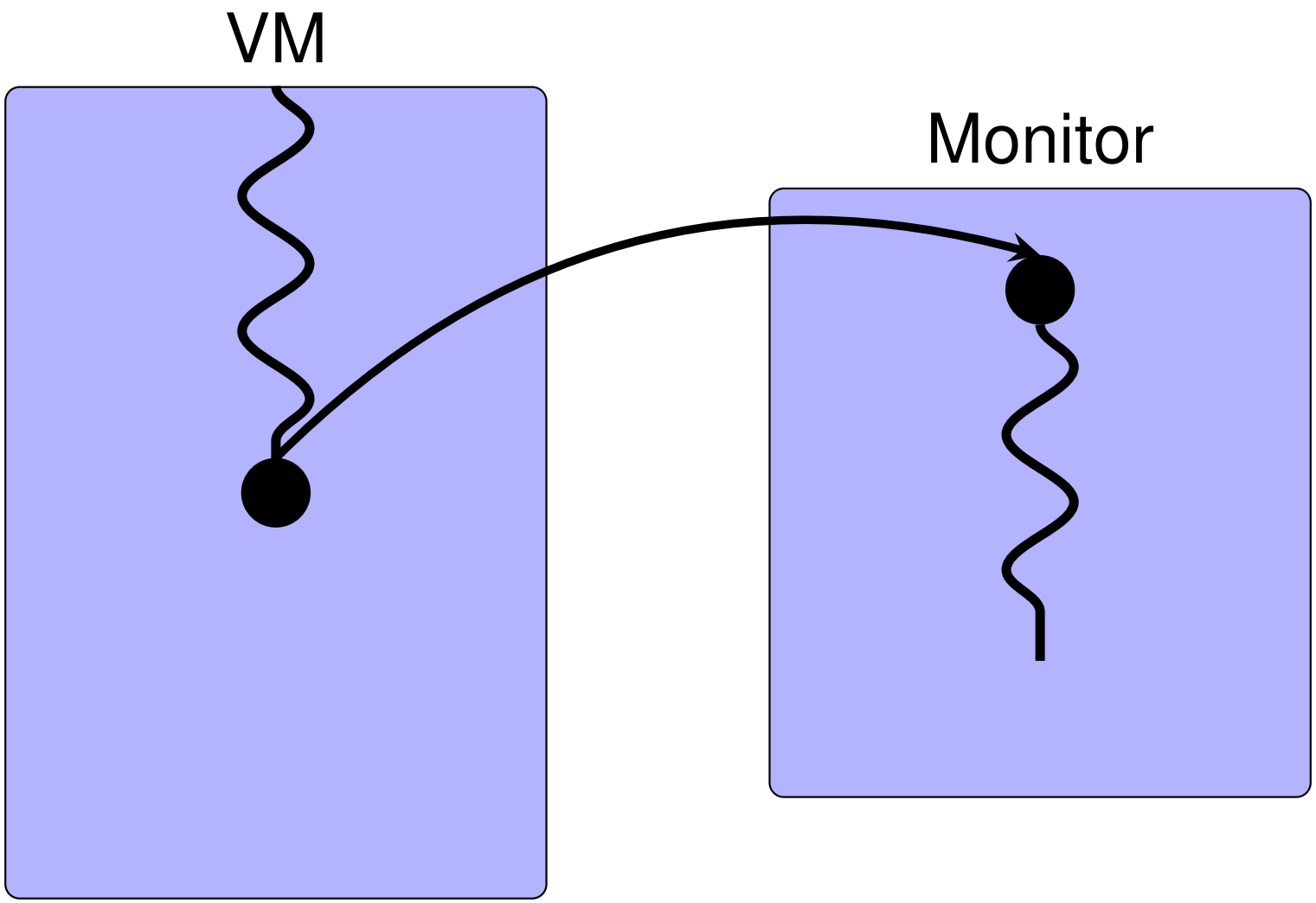
Hook-based Monitoring

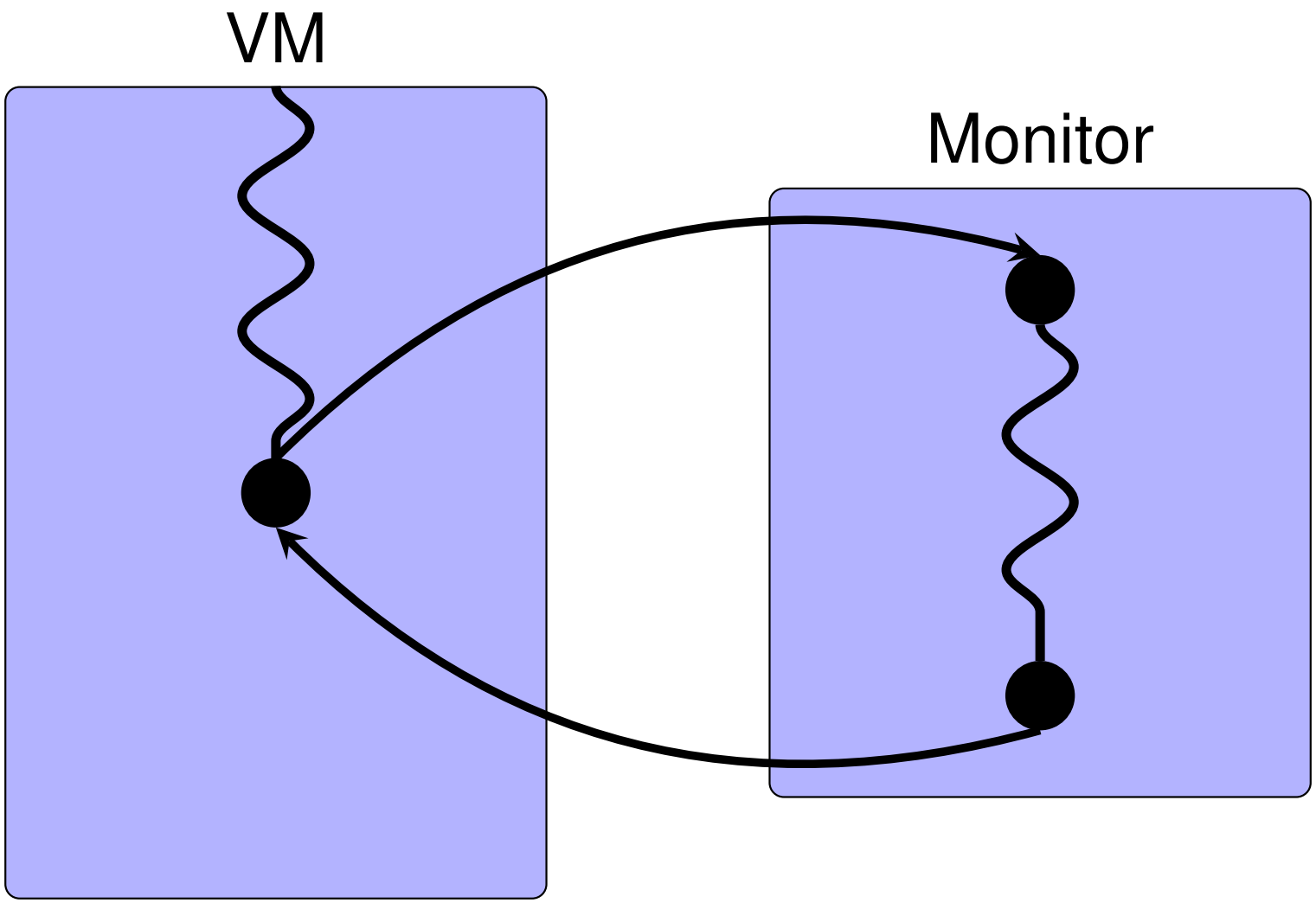
VM

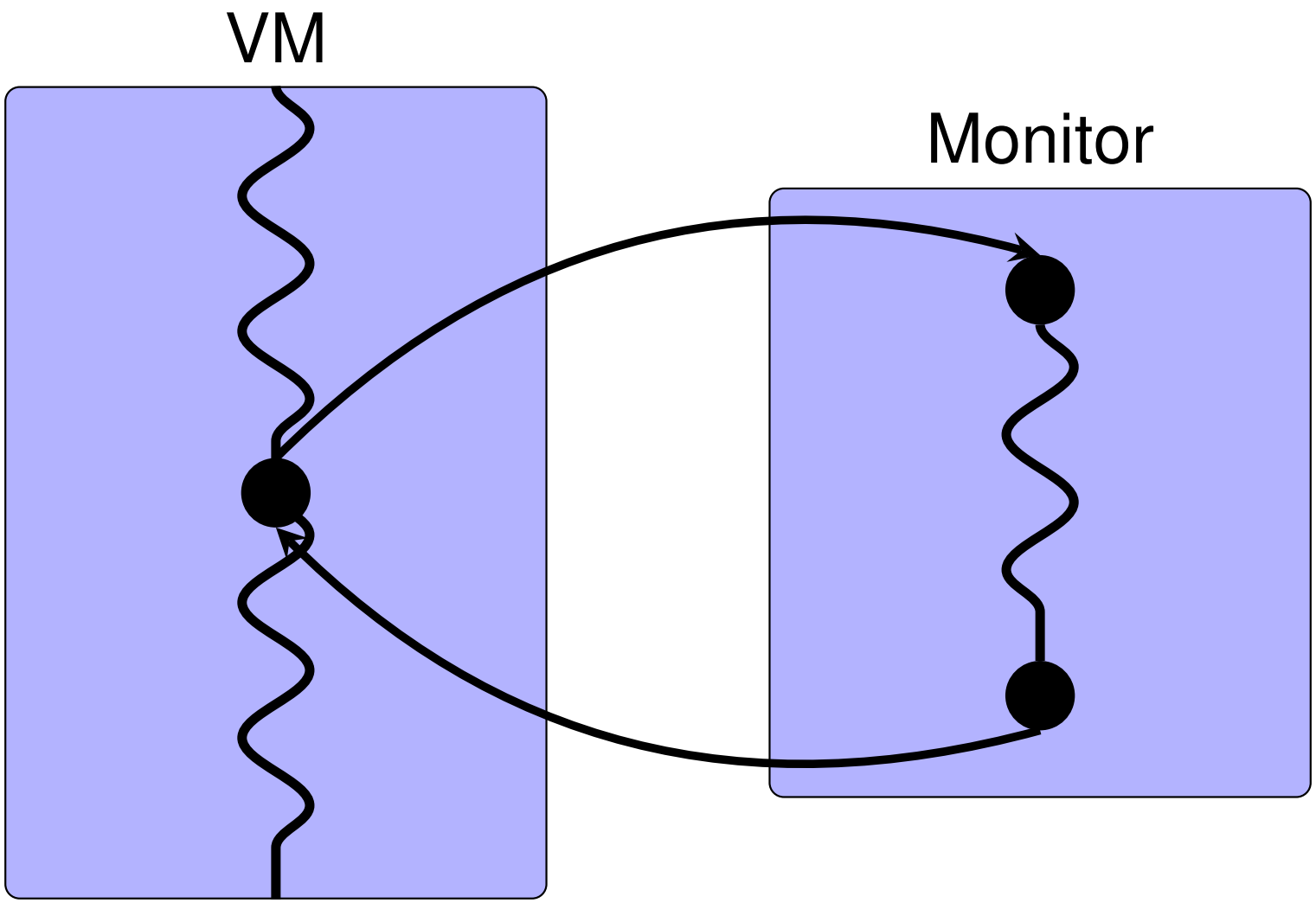



VM


VM
Monitor


VM
Monitor


VM
Monitor


VM
Monitor

Formal Model for Hooks

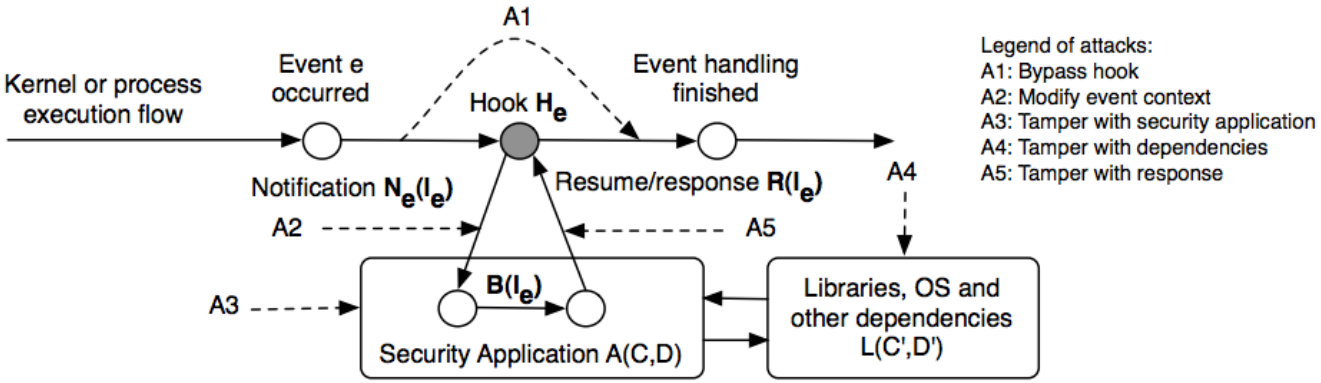


Figure 1: Formal model of secure active monitoring shown with potential attacks. [7]

[7] Payne *et al.*, "Lares: An architecture for secure active monitoring using virtualization," 2008

Formal Model for Hooks

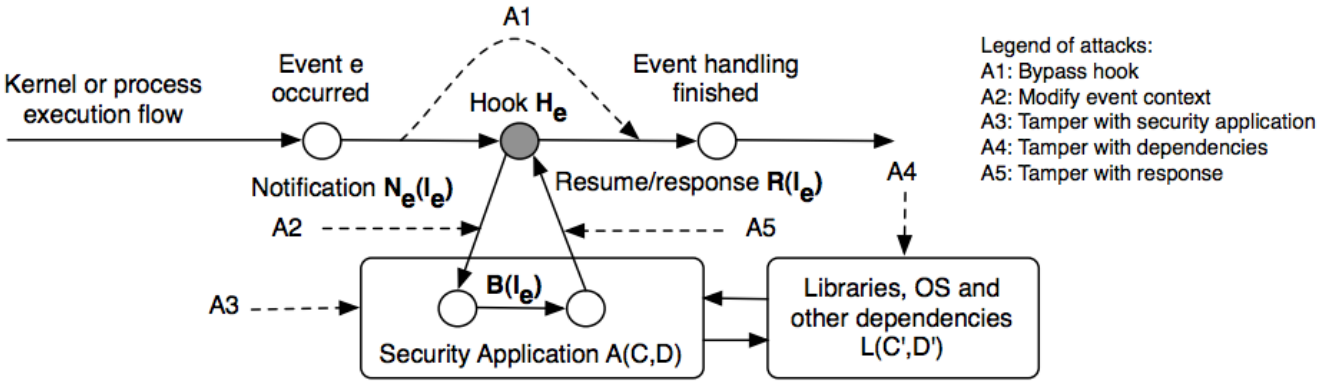


Figure 1: Formal model of secure active monitoring shown with potential attacks. [7]

- Notification only on legitimate events
- VM state fixed during processing
- Monitor behavior can't be alterered
- Response is enforced

[7] Payne *et al.*, "Lares: An architecture for secure active monitoring using virtualization," 2008

Hook-based VM Monitoring Examples

Hooks - VM Monitoring

When desigining, have to consider:

- How to hook?
- Where to run the monitor?

Lares

- How to hook?
  - `VMCALL` to hypervisor
- Where to run the monitor?
  - Separate security VM

Lares

- Xen with windows guest

Lares

- Xen with windows guest
- Hook $\Rightarrow$ replace entries in dispatch table

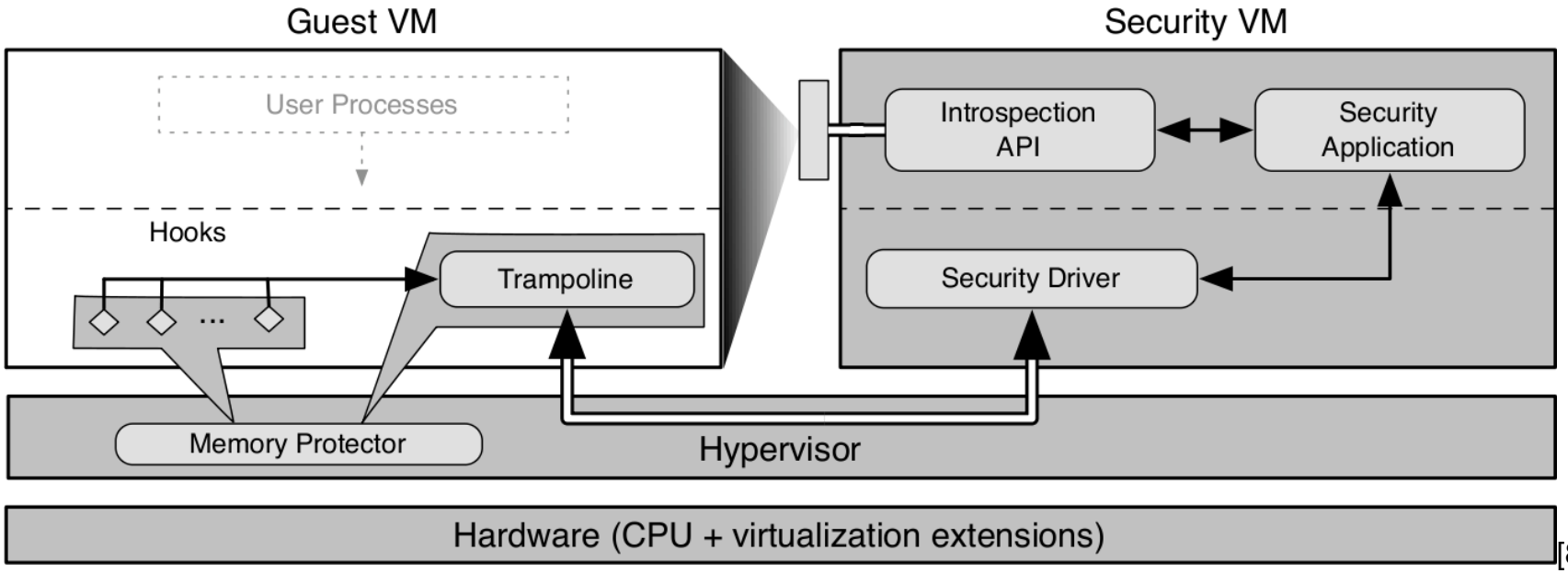
Lares

- Xen with windows guest
- Hook $\Rightarrow$ replace entries in dispatch table
- Entries go to trampoline that does a `VMCALL`

Lares

- Xen with windows guest
- Hook $\Rightarrow$ replace entries in dispatch table
- Entries go to trampoline that does a `VMCALL`
- Command from target is passed to security driver in secure VM

Lares - Diagram



[8]

[8] Payne *et al.*, “Lares: An architecture for secure active monitoring using virtualization,” 2008

Secure-In VM Monitoring

- How to hook?
  - jump to entry gate
- Where to run the monitor?
  - Separate address space inside VM

Secure-**I**n VM **M**onitoring

- Hardware-Assisted Virtualization w/KVM

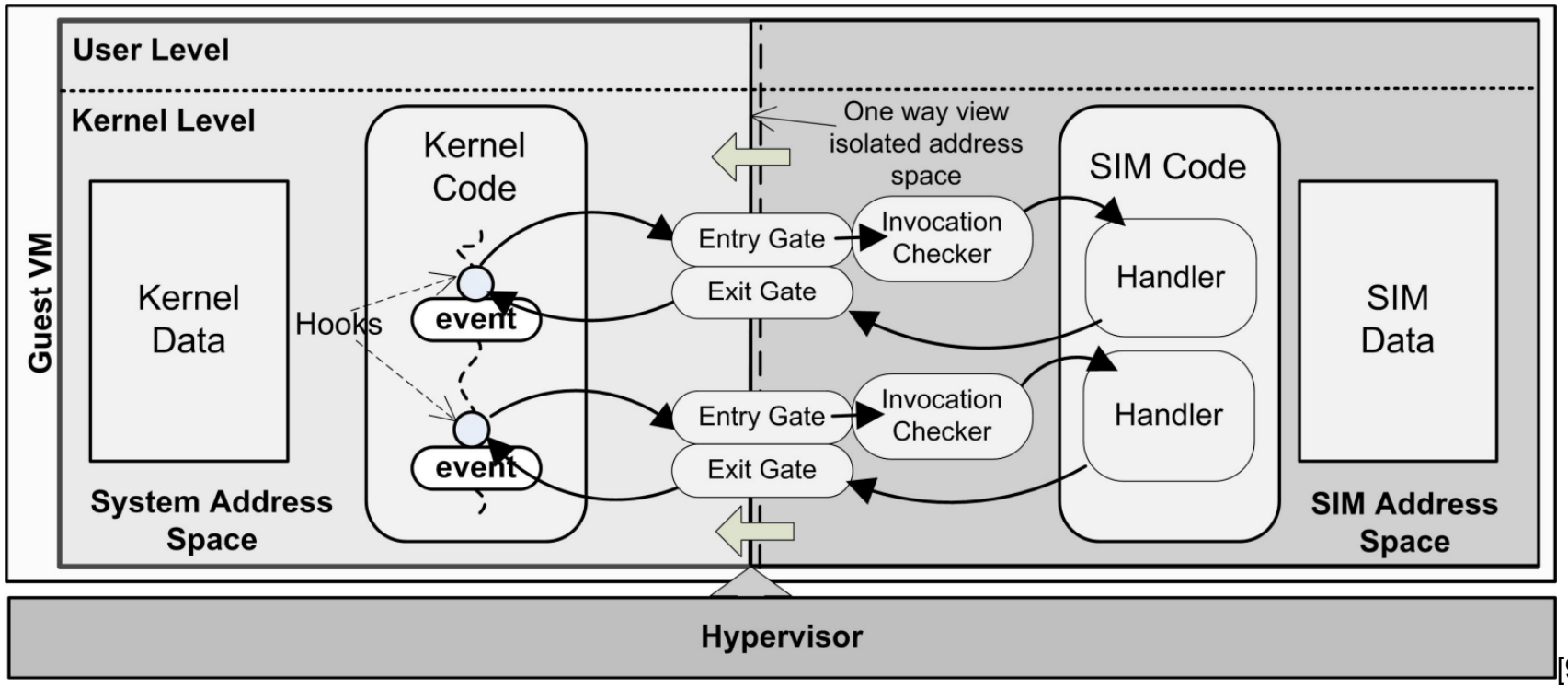
Secure-**I**n VM **M**onitoring

- Hardware-Assisted Virtualization w/KVM
- Sets up a protected address space in target

Secure-In VM Monitoring

- Hardware-Assisted Virtualization w/KVM
- Sets up a protected address space in target
- Hooks are inserted into kernel functions

Hooks - SIM



[9]

[9] Sharif *et al.*, “Secure in-VM Monitoring Using Hardware Virtualization,” 2009

Properties of Lares, SIM, ...

Pros/cons?

Properties of Lares, SIM, ...

Pros/cons?

- + Active monitoring
- + Protected hooks
- – guest OS only - no userspace
- – Not dynamic - boot time config
- – Require guest OS modifications
- – Lares - only dispatch tables?
- – SIM - limit on number of VMs

Detailed Example: hprobes

Goals

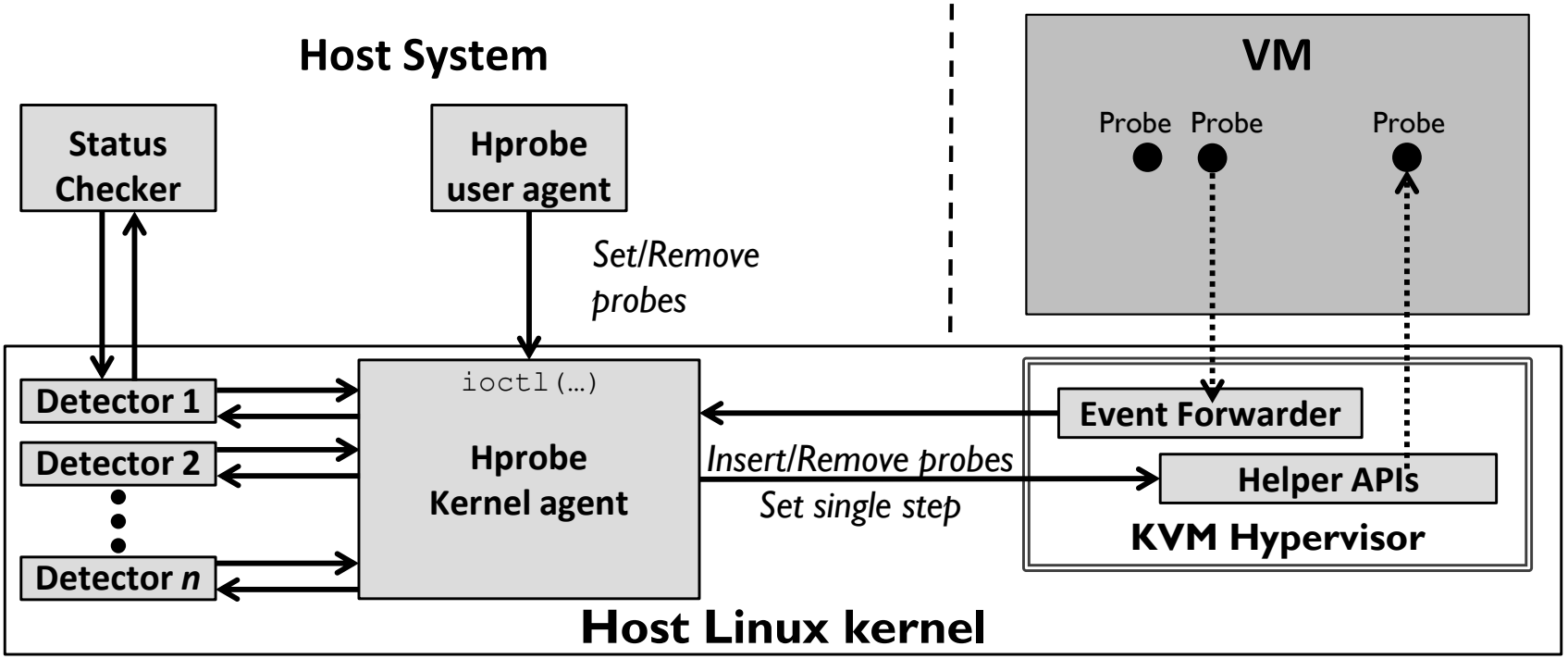
Hook-based monitoring should:

+ be protected from attacks in the VM
+ be simple to use
+ not require guest OS modification
+ be runtime adaptable
+ allow for arbitrary hook placement

Hypervisor Probes

- VMExit on instruction execution (`int3`)
  - execute a `handler()`
- Hooks added/removed at runtime
- Can monitor userspace and the guest OS

Hprobe Architecture



Hprobes API

```
int HPROBE_add_probe(addr_info, vmid,
          (*)handler(vcpu_type *VCPU));

int HPROBE_remove_probe(addr_info, vmid);
```

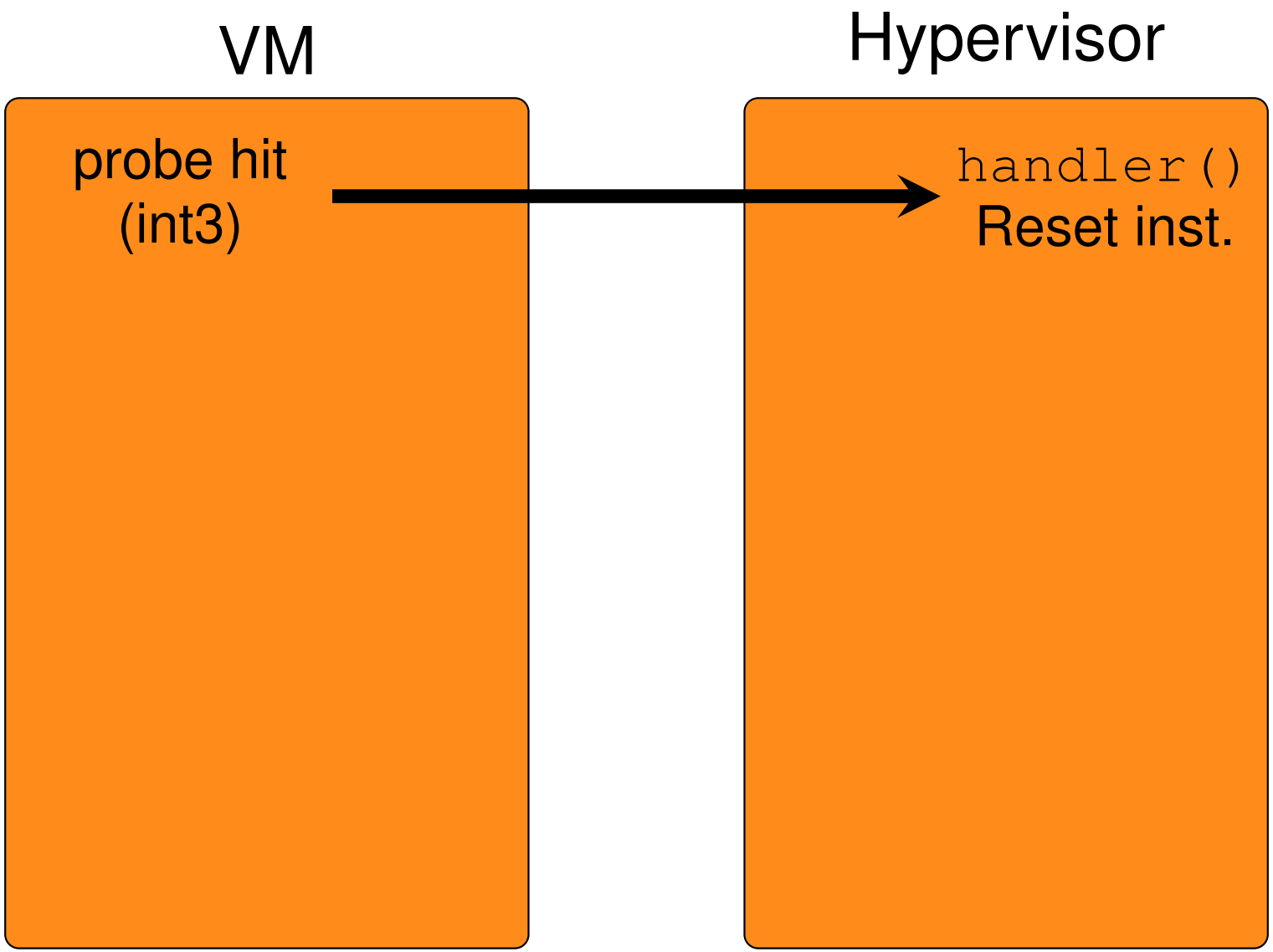
Hprobes API

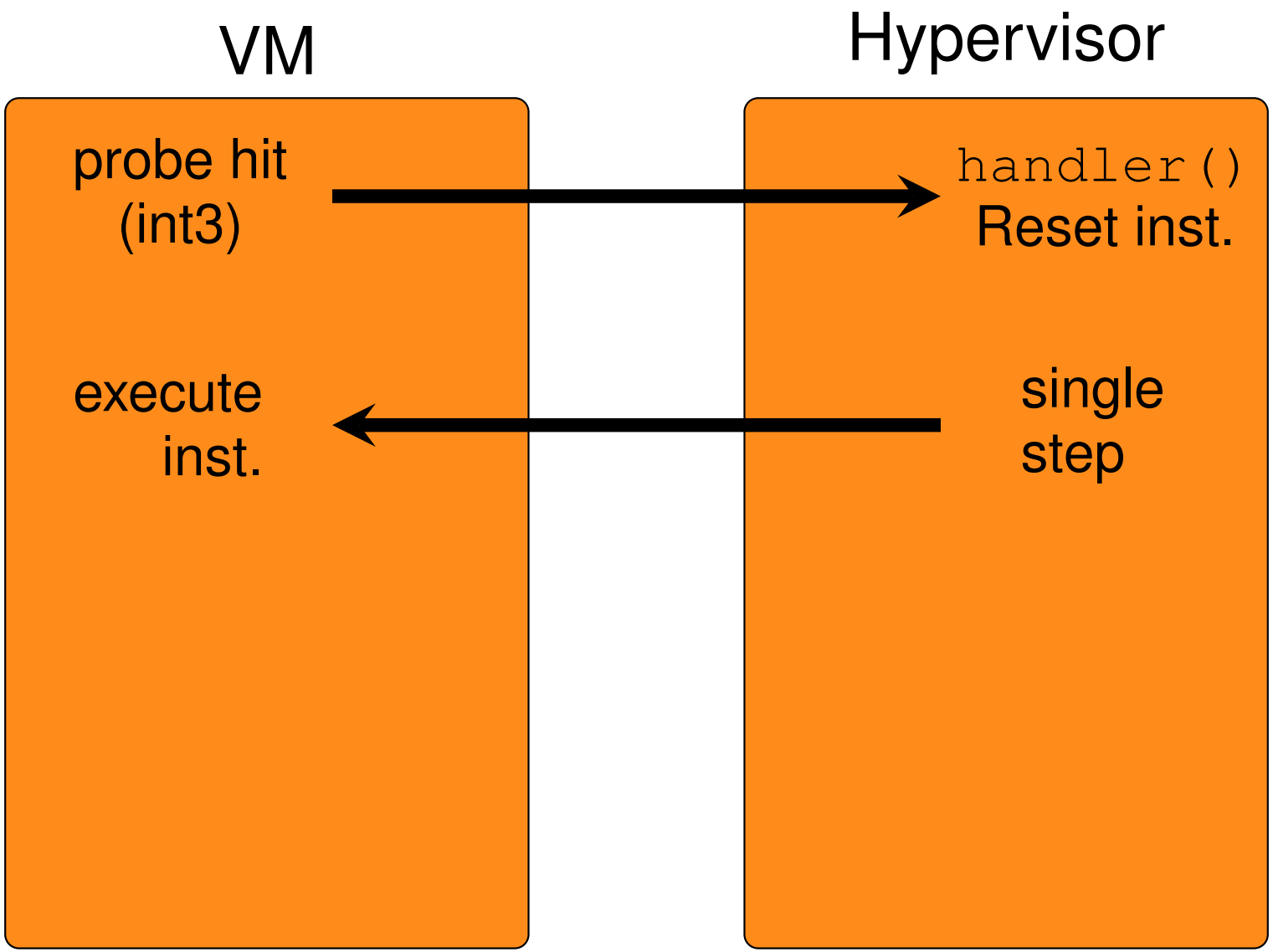
```
int HPROBE_add_probe(addr_info, vmid,
          (*)handler(vcpu_type *VCPU));

int HPROBE_remove_probe(addr_info, vmid);
```

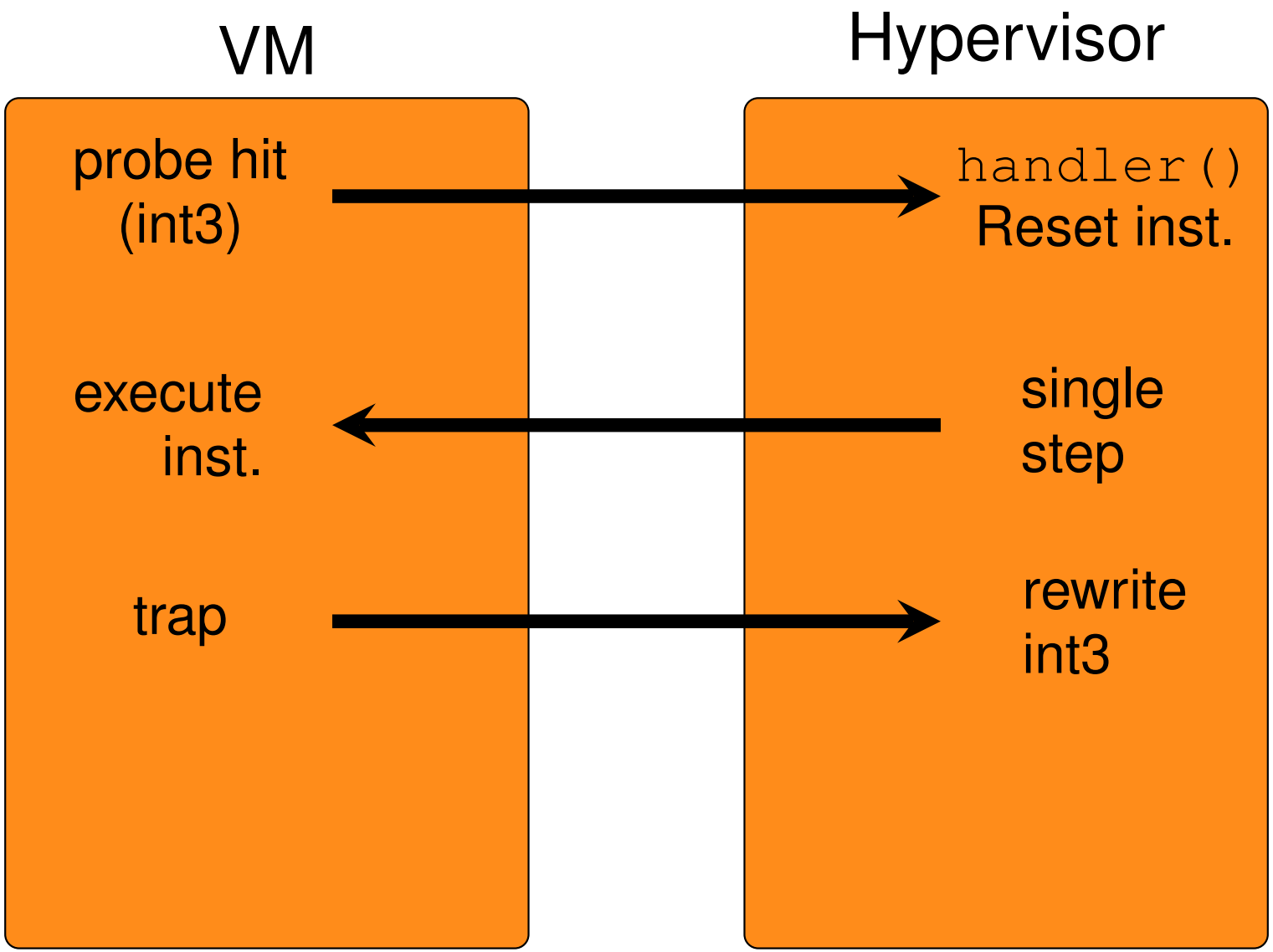
- `addr_info`: gva+cr3
- `vmid`: unique id for VM
- `vcpu_type`: vcpu state

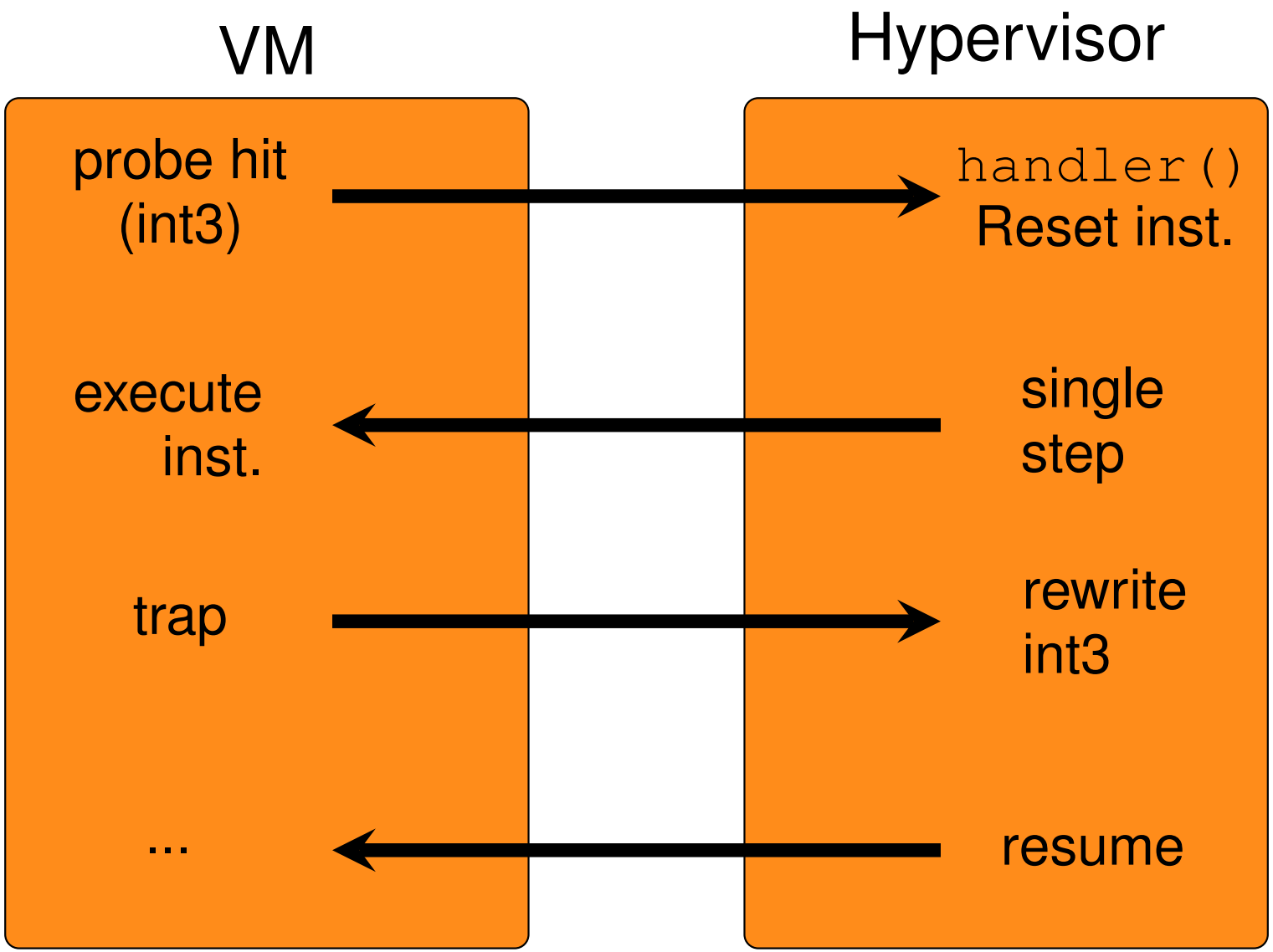
VM

Hypervisor


VM
Hypervisor
probe hit
(int3)
handler()
Reset inst.


VM
Hypervisor
probe hit
(int3)
handler()
Reset inst.
execute
inst.
single
step


VM
Hypervisor
probe hit
(int3)
handler()
Reset inst.
execute
inst.
single
step
trap
rewrite
int3


VM
Hypervisor
probe hit
(int3)
handler()
Reset inst.
execute
inst.
single
step
trap
rewrite
int3
...
resume

What an Hprobe Looks Like

Original

```
...
pushl %eax
incl %eax
decl %ebx
...
```

What an Hprobe Looks Like

With Probe

```
...
pushl %eax
int3
decl %ebx
...
```

What an Hprobe Looks Like

With Probe

```
...
pushl %eax
int3
decl %ebx
...
```

What an Hprobe Looks Like

With Probe	Execute
...	...
pushl %eax	pushl %eax
int3	incl %eax
decl %ebx	decl %ebx
...	...

What an Hprobe Looks Like

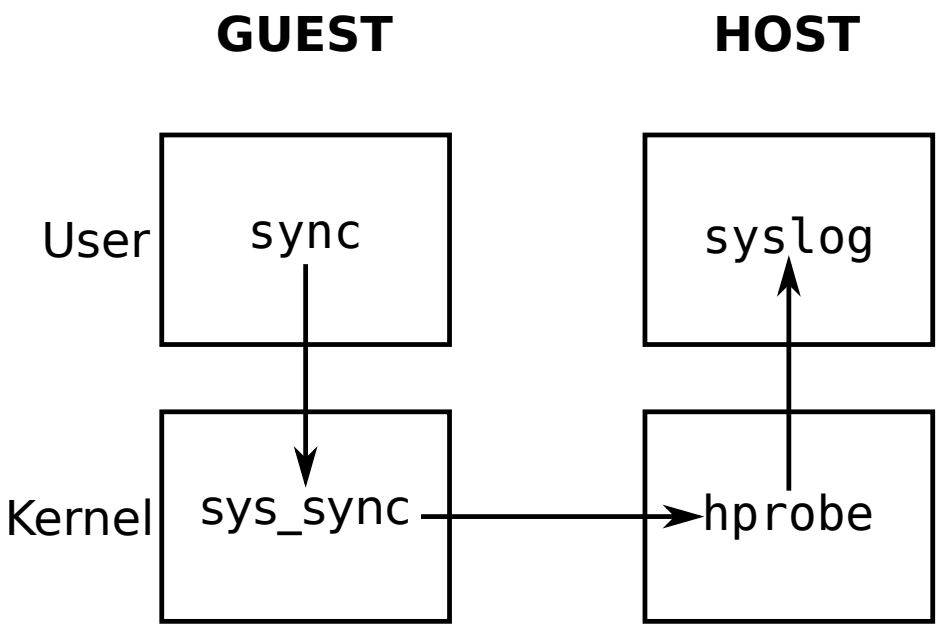
With Probe	Execute	Resume
...	...	...
pushl %eax	pushl %eax	pushl %eax
int3	incl %eax	int3
decl %ebx	decl %ebx	decl %ebx
...	...	...

Demo: Hprobe Example

- Single kernelspace probe
- `sync` system call
- Probe added at runtime

Demo: Hprobe Example

- Single kernelspace probe
- `sync` system call
- Probe added at runtime



Userspace Programs

- Need to overwrite instructions in user programs
- Use CR3 to identify process
- But... just using CR3 isn't good enough

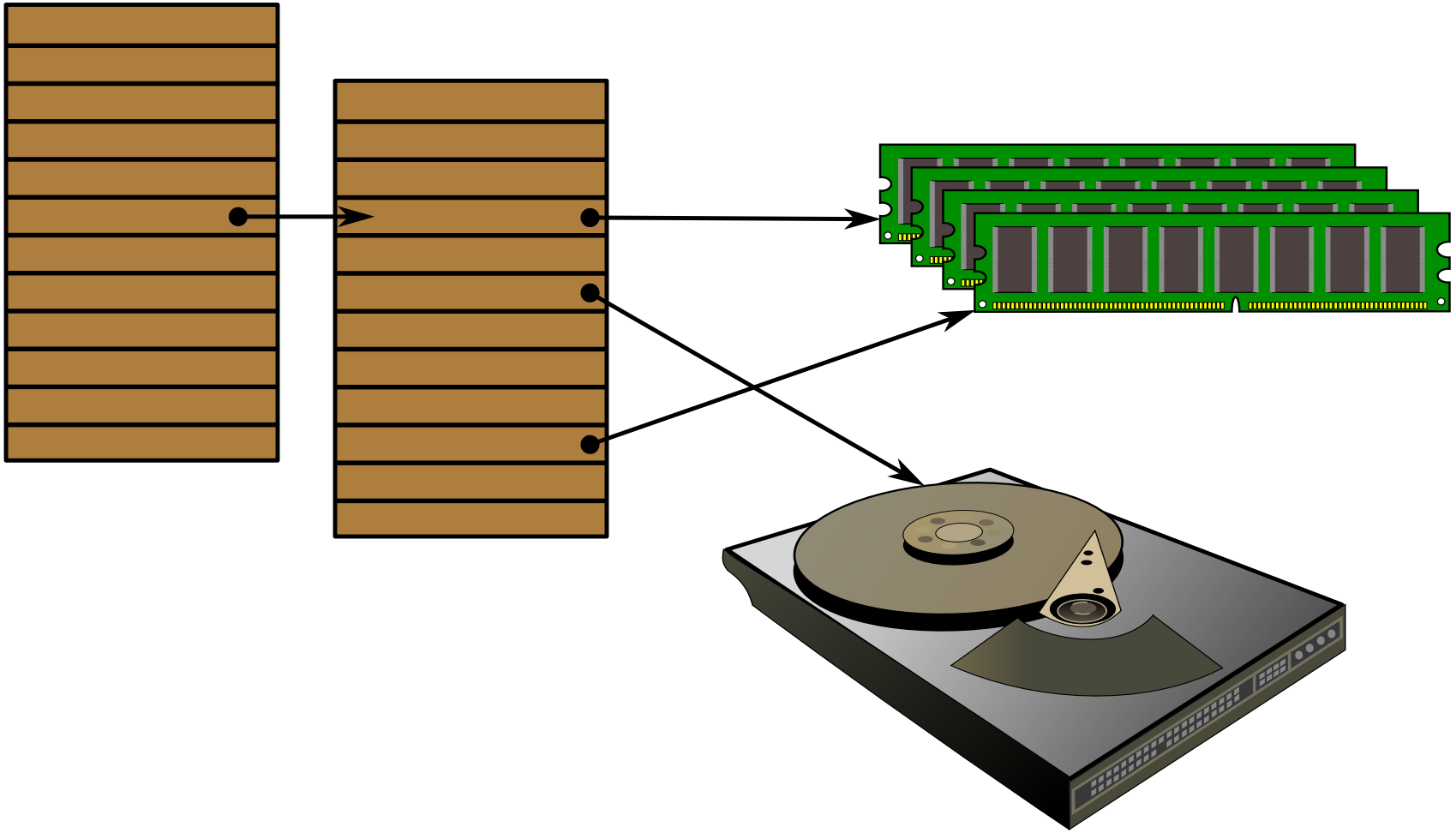
Userspace Programs

- Need to overwrite instructions in user programs
- Use CR3 to identify process
- But... just using CR3 isn't good enough

Question

Why might CR3 not be enough?

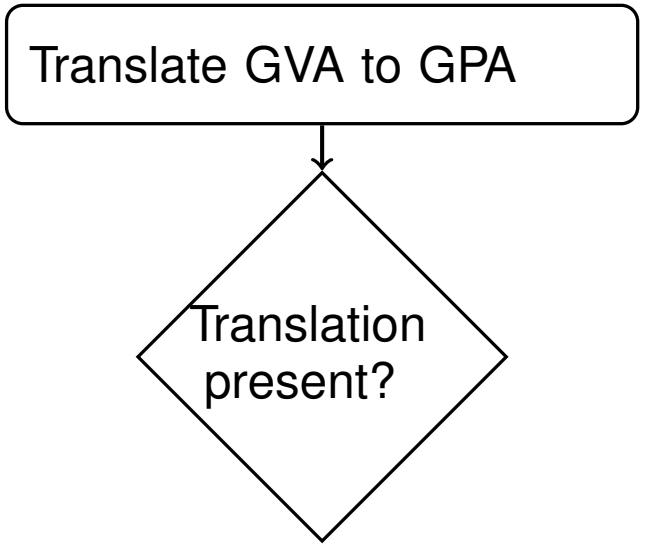
Pages Aren't Always in RAM!



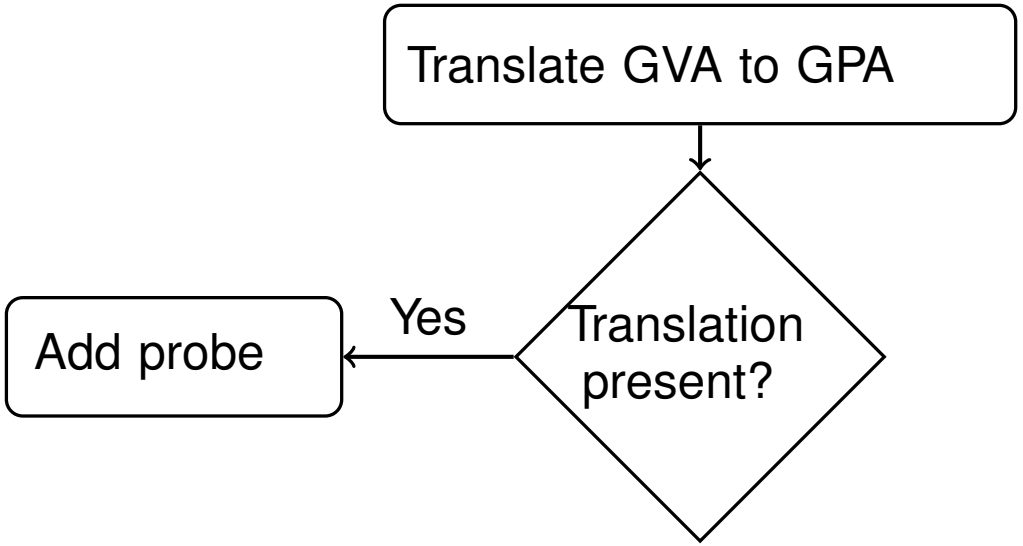
Use Page Permissions

Translate GVA to GPA

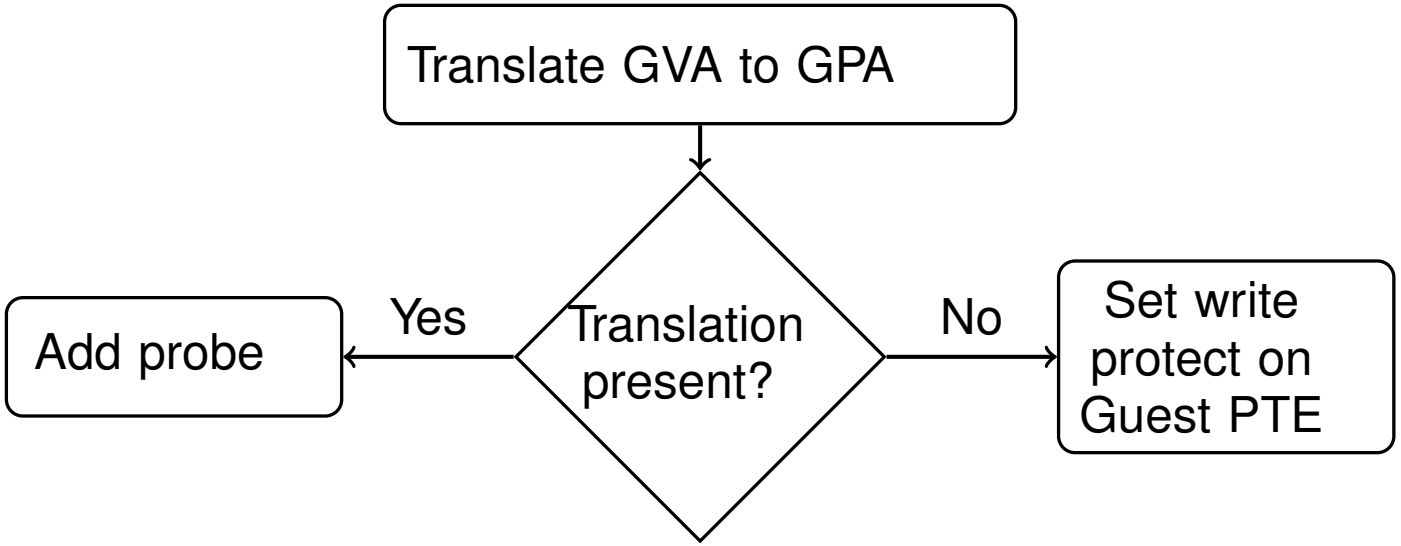
Use Page Permissions



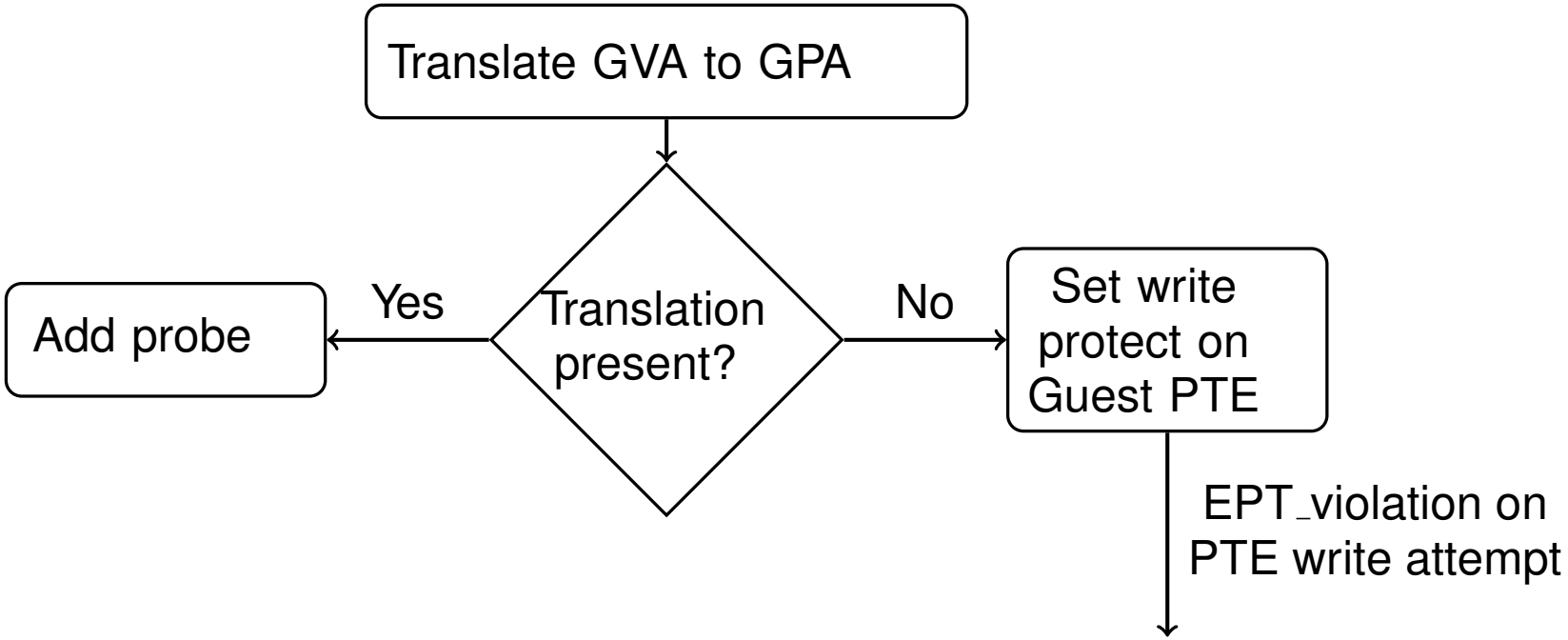
Use Page Permissions



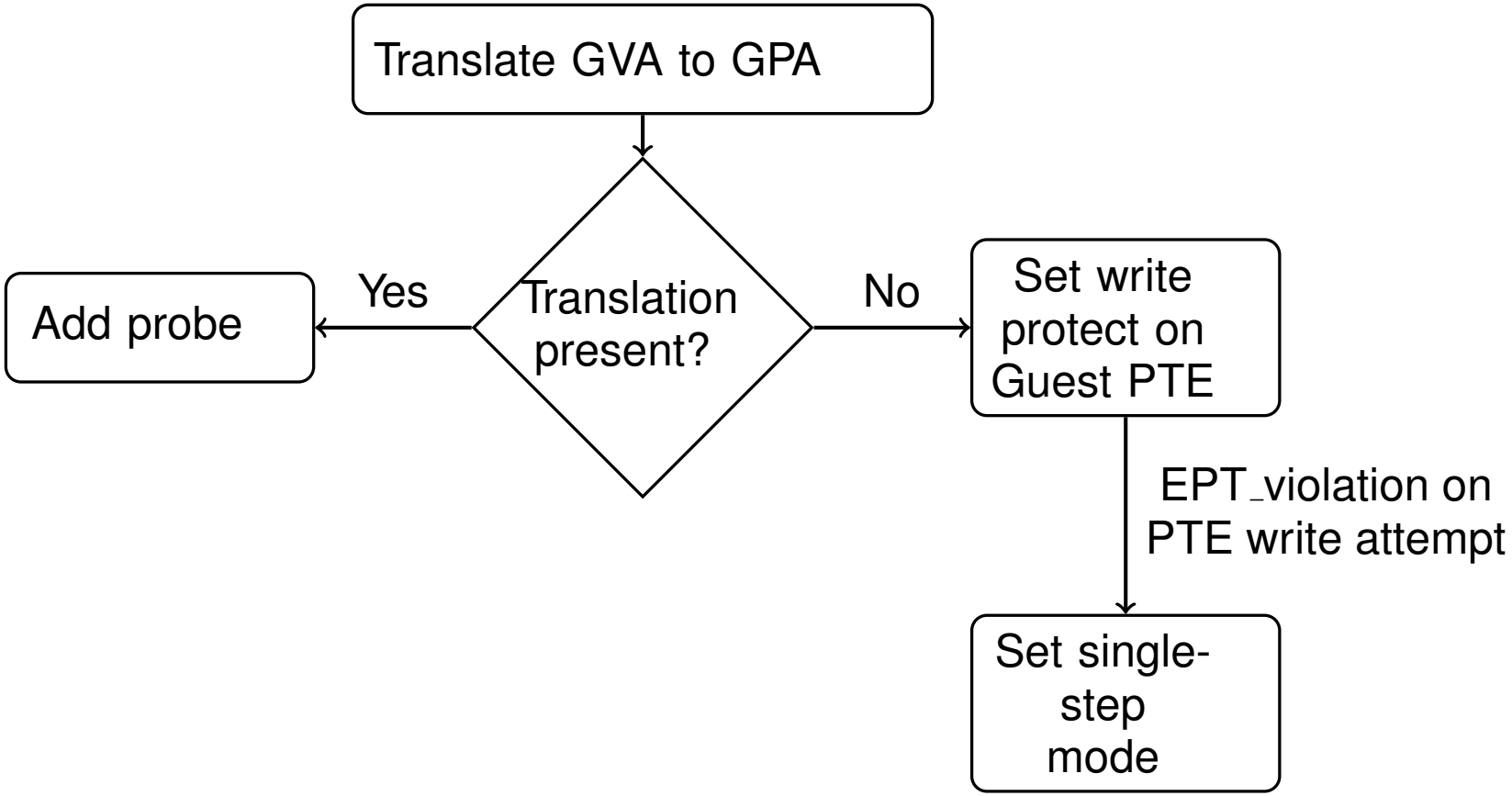
Use Page Permissions



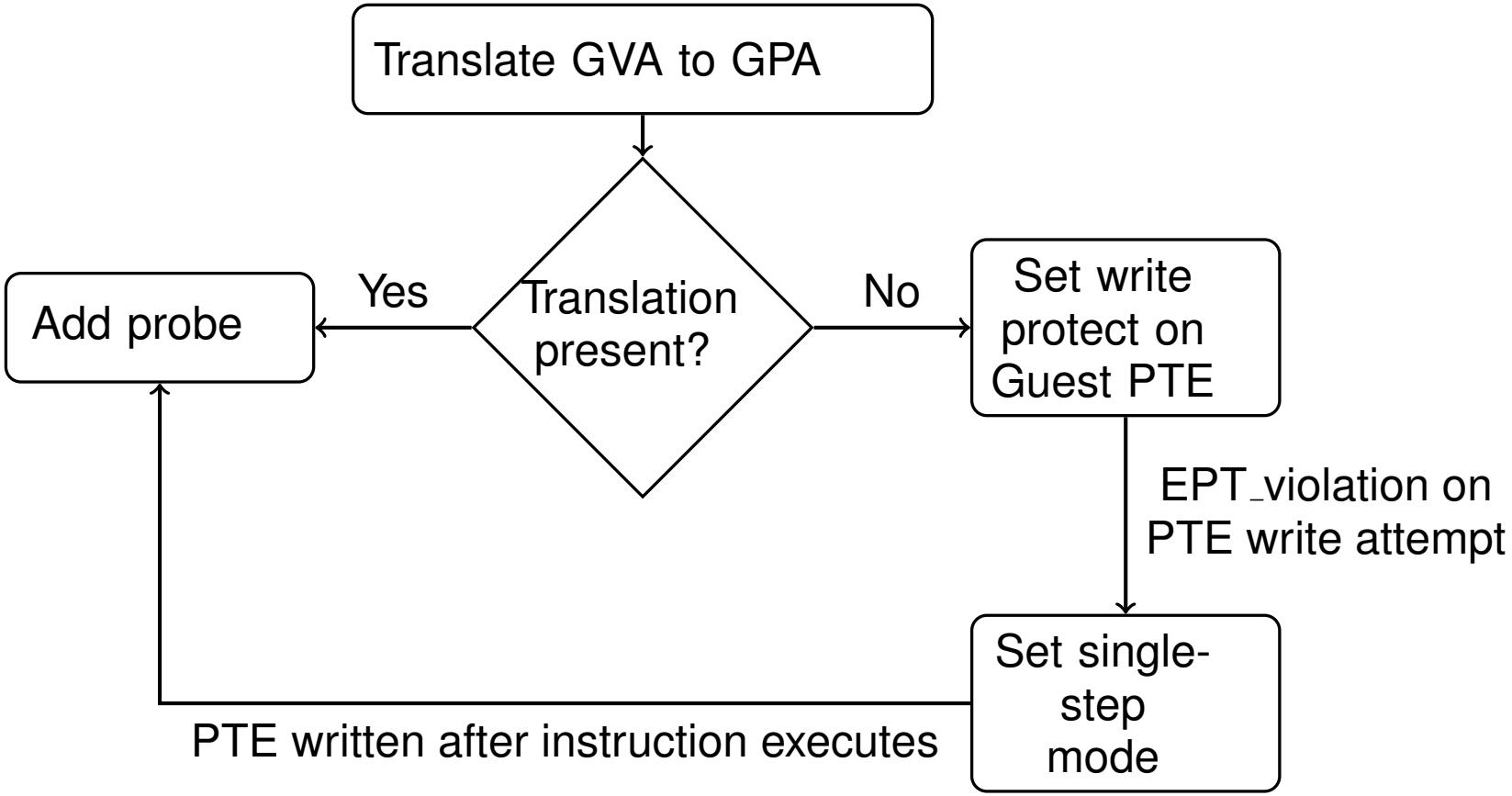
Use Page Permissions



Use Page Permissions



Use Page Permissions

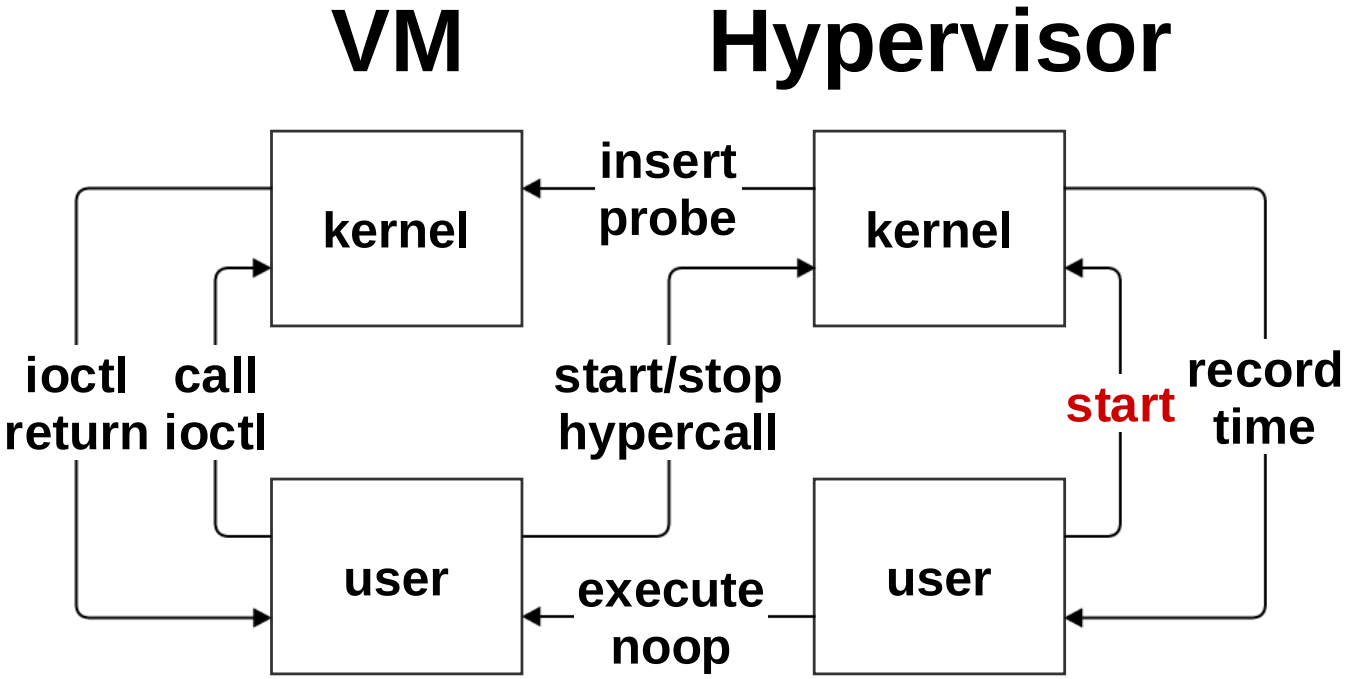


Hprobes

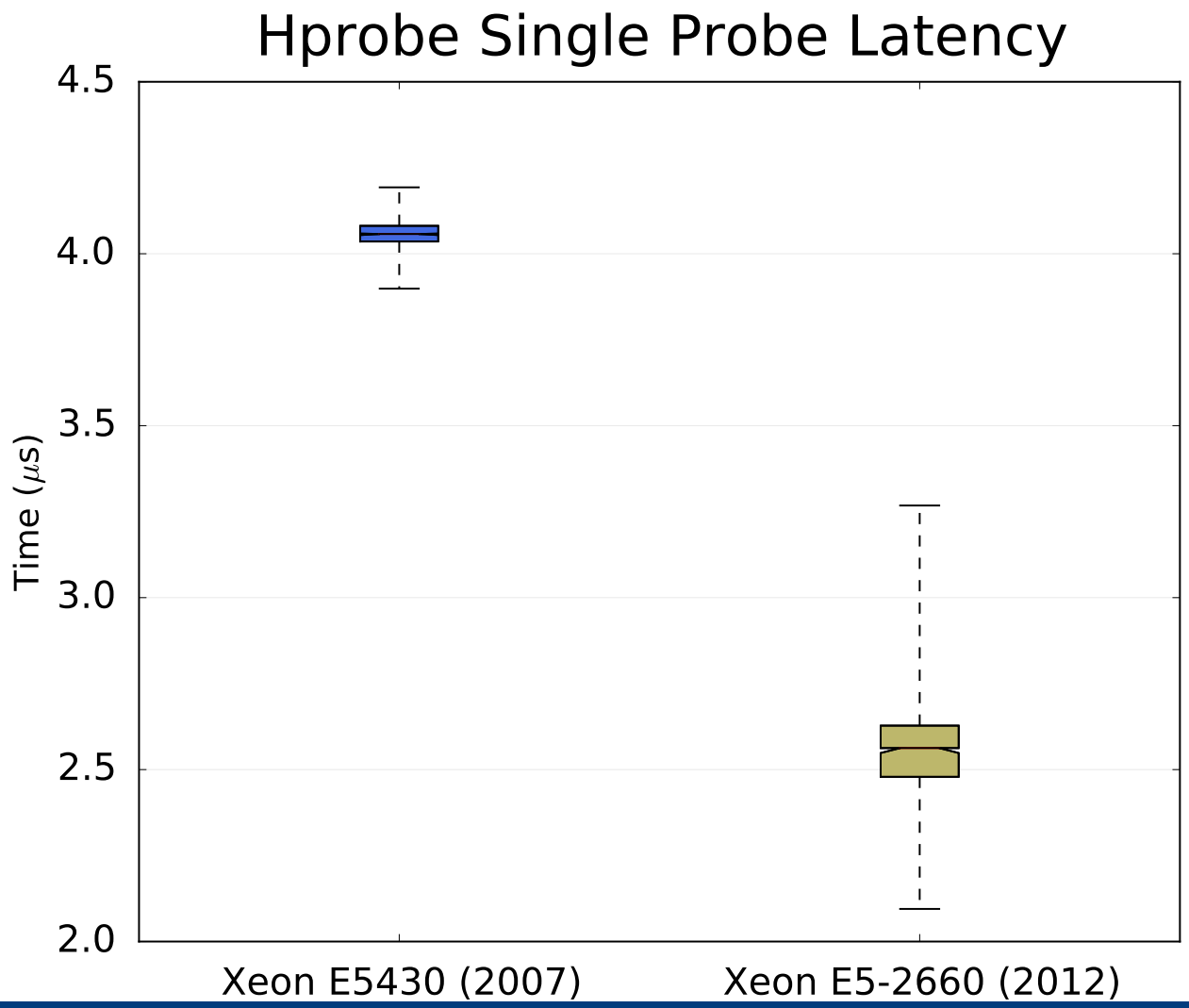
- Active monitoring
- Transparent - OS agnostic
- Dynamic
- Protect w/EPT
- Simple

Hprobe microbenchmarks

- probe @ `noop` kernel function
- execute 1M times



Hprobe microbenchmarks



Hook-based Techniques

Name	User	Latency	Dynamic	Modifications
Lares	No	28μs	No	Hypervisor/Guest
SIM	No	0.40μs	No	Hypervisor/Guest
hprobes	Yes	2.6μs	Yes	Hypervisor

Detectors

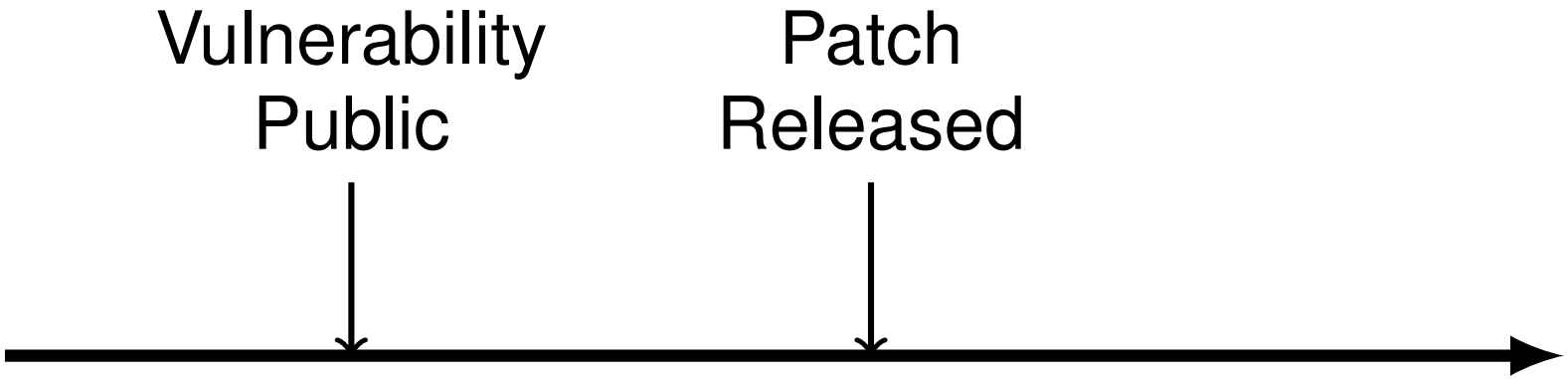
What can we do with this?

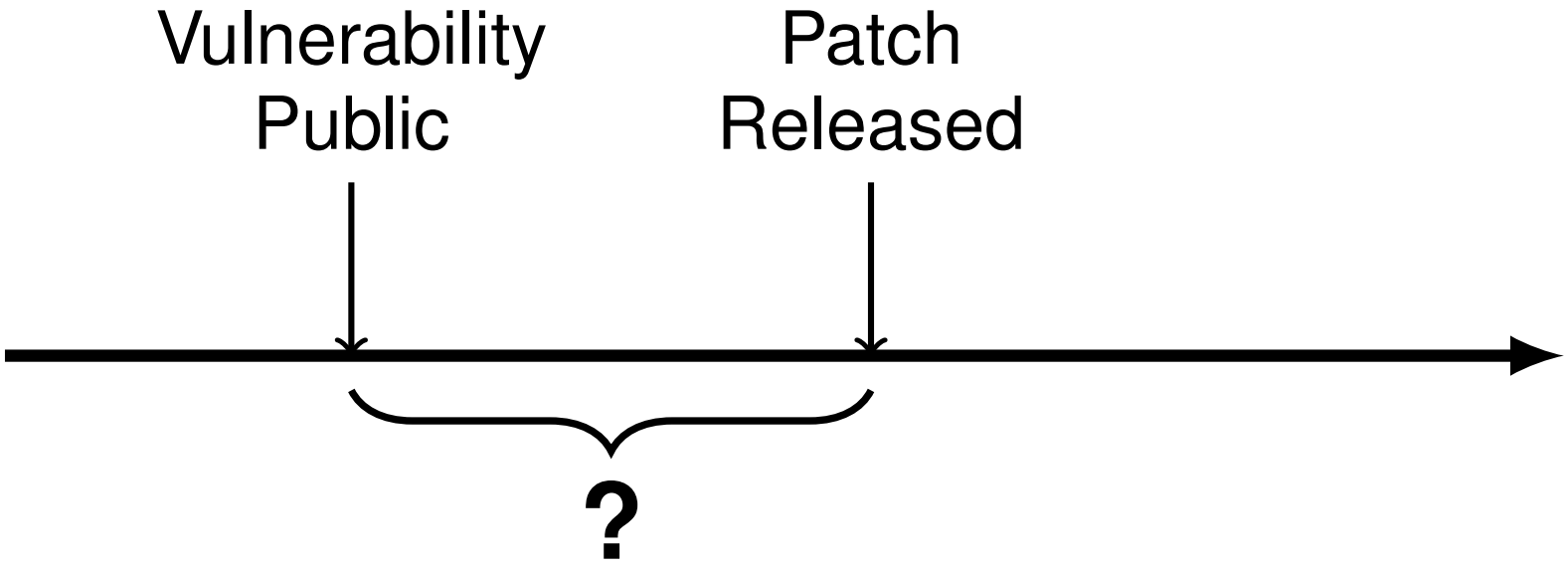
Detectors

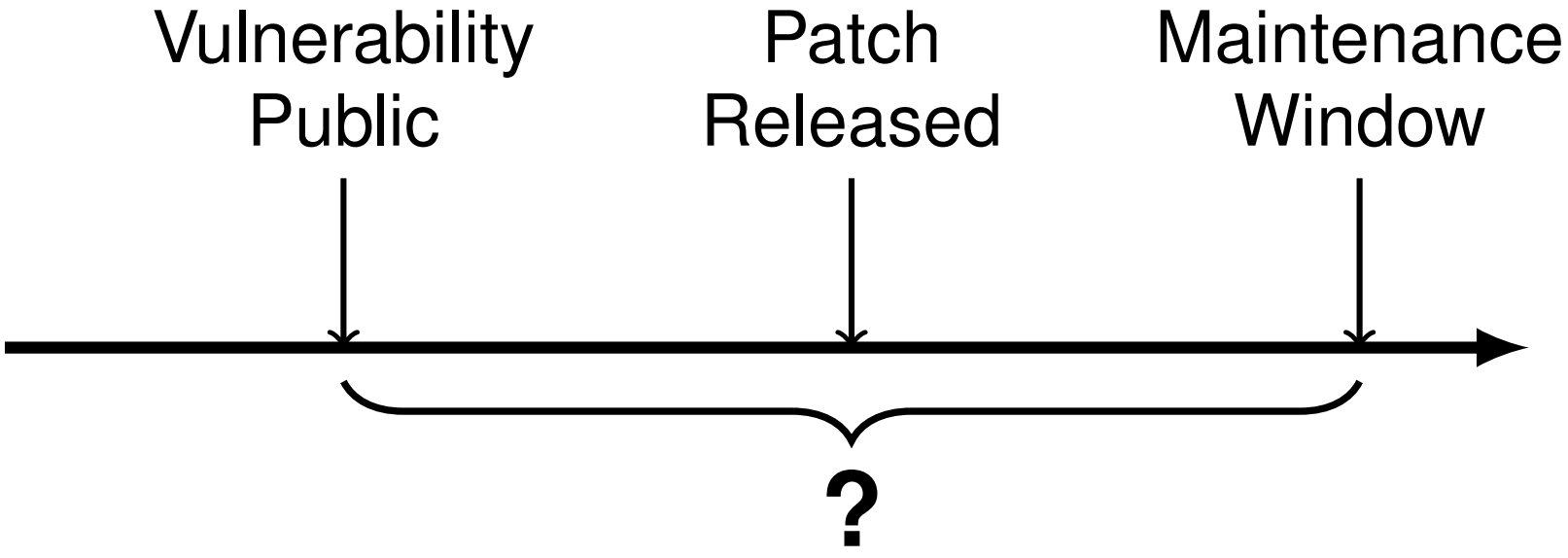
What can we do with this?

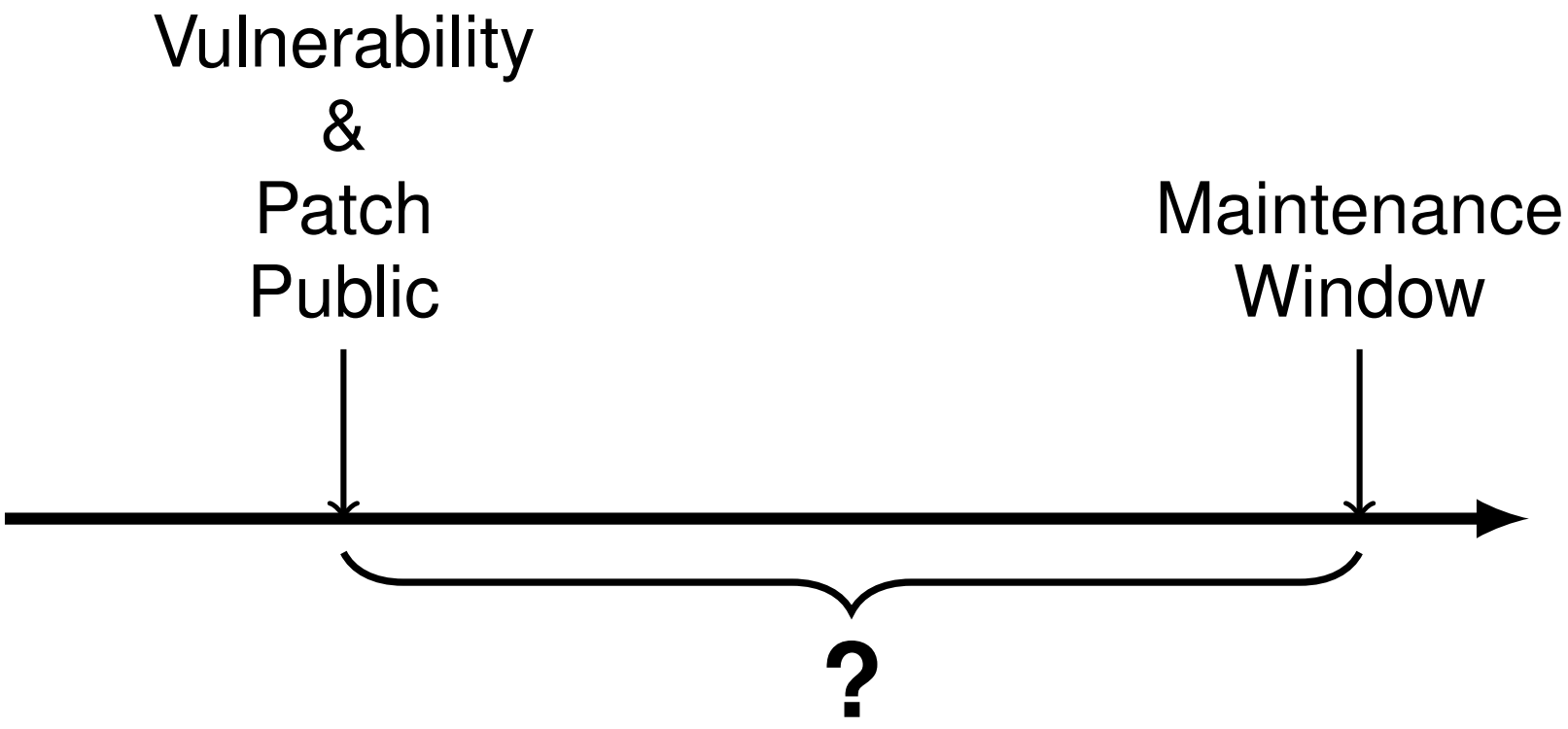
- Arbitrarily chose events
  - At runtime
- App-level monitoring

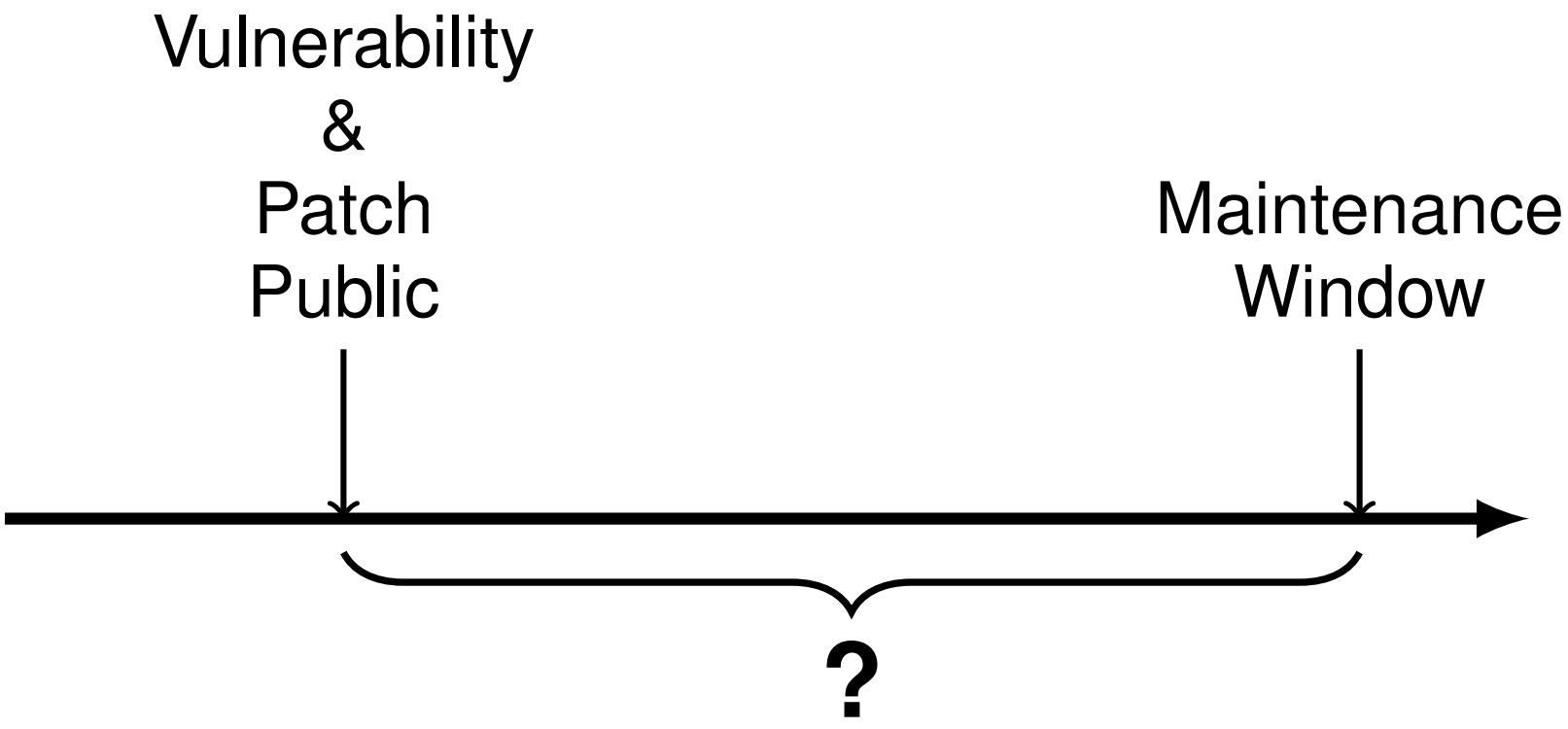

Vulnerability
Public


Vulnerability
Public
Patch
Released


Vulnerability
Public
Patch
Released
?


Vulnerability Public
Patch Released
Maintenance Window
?


Vulnerability
&
Patch
Public
Maintenance
Window
?



Solution

Use an *Emergency Detector*

Emergency Vulnerability Detector

- CVE-2008-0600 - Privilege Escalation in `vmsplice()` [10]
- integer overflow in a `struct iovec` system call argument
- insert probe at `vmsplice()` syscall
- get value of argument off of the stack

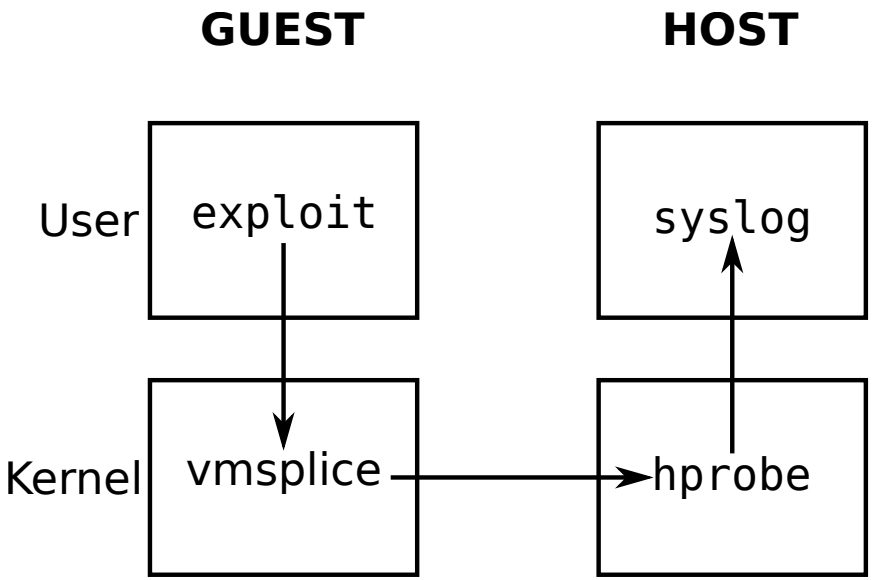
[10] `http://www.win.tue.nl/~aeb/linux/hh/hh-12.html#ss12.4`

Emergency Vulnerability Detector

```
procedure VMSPLICE_HANDLER(vcpu)
    iov_pointer ← read_guest(esp+arg_offset)
    iov_len ← read_guest_virt(iov_pointer)

    if iov_len ≥ 0xFFFFFFFF then
        HANDLE_EXPLOIT_ATTEMPT(vcpu)
    end if
end procedure
```

Demo: Emergency Vulnerability Detector

- Single kernelspace probe, added at runtime
- One **HOST** and one **GUEST**
- “notroot” user in **GUEST** escalates privileges



Checkpoint/Restart In Userspace

Checkpoint two test applications

Checkpoint/Restart In Userspace

Checkpoint two test applications

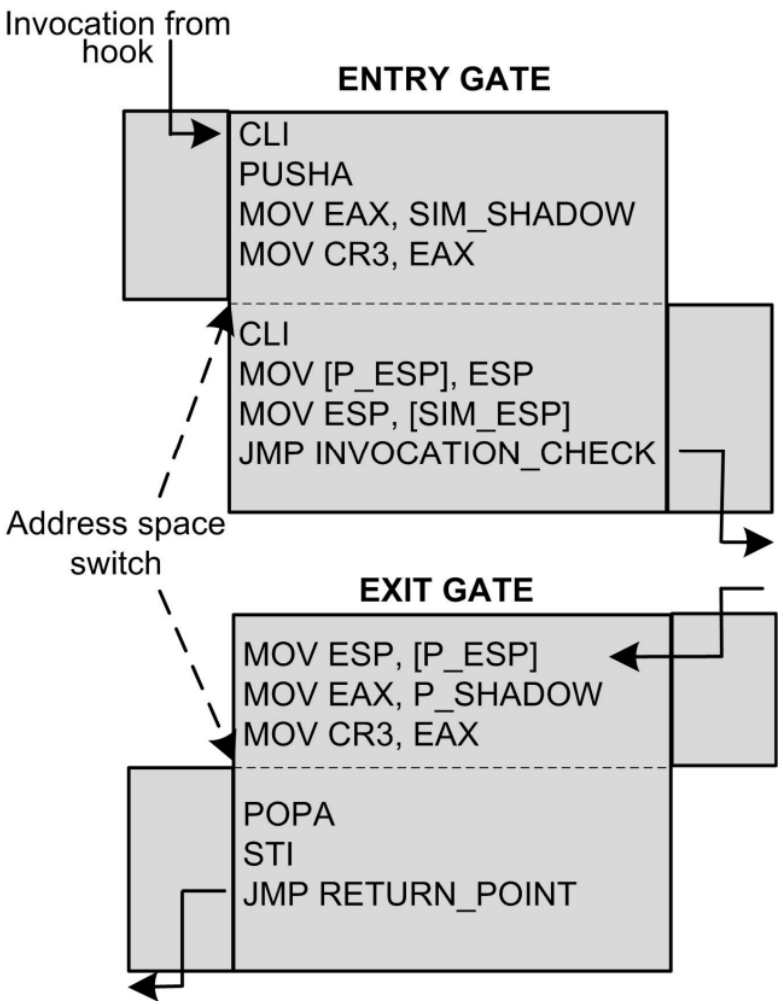
Application	**Runtime (s)**	**95% CI** (s)	**% overhead**
F@H Normal	0.221	0.00922	0
F@H w/hprobe	0.228	0.0122	3.30
F@H w/Naïve	0.253	0.00851	14.4
pi-qmc Normal	0.137	0.00635	0
pi-qmc w/hprobe	0.140	0.00736	1.73
pi-qmc w/Naïve	0.152	0.00513	11.1

Summary

- Hooks
- Lares, SIM
- Hprobes
  - How hprobes work
  - Performance
  - Demo
  - Emergency Detector

BACKUP

SIM Entry/Exit Gates



Why hprobes vs. OS-level?

Why hprobes vs. OS-level?

- Isolation/Protection

Why hprobes vs. OS-level?

- Isolation/Protection
- More views

Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor

Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes

Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes
  - Less variance in hypervisors

Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes
  - Less variance in hypervisors
  - Reliability-as-Service

Why hprobes vs. OS-level?

- Isolation/Protection
- More views
  - Other VMs/hypervisor
- No guest OS changes
  - Less variance in hypervisors
  - Reliability-as-Service
- Unikernel/OSv

“Hello World” Detector

Heartbeat/watchdog

- App-level

“Hello World” Detector

Heartbeat/watchdog

- App-level
- Critical code sections

“Hello World” Detector

Heartbeat/watchdog

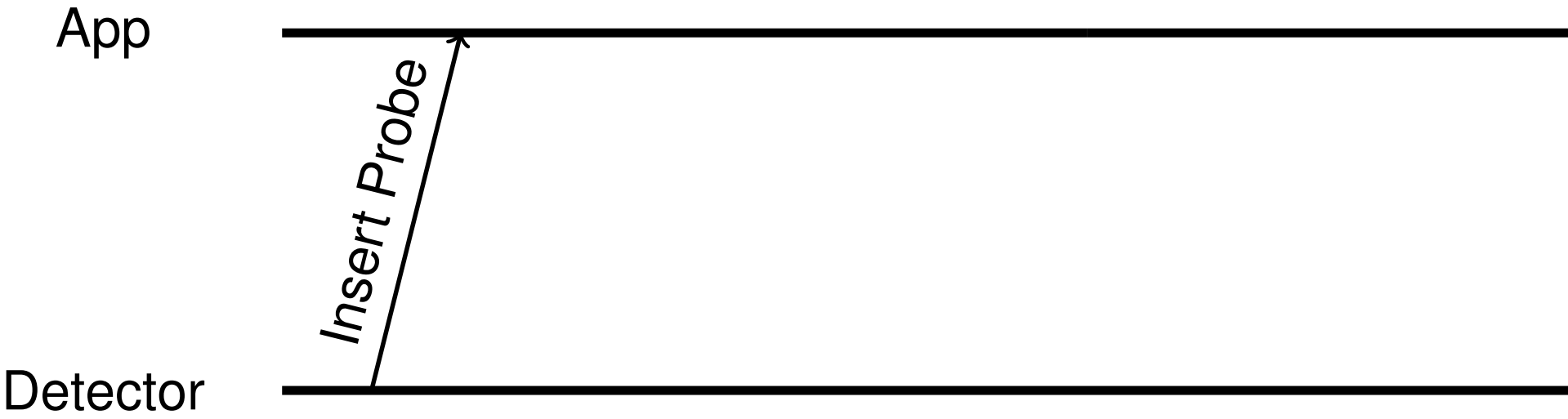
- App-level
- Critical code sections
- Act?

Heartbeat/watchdog

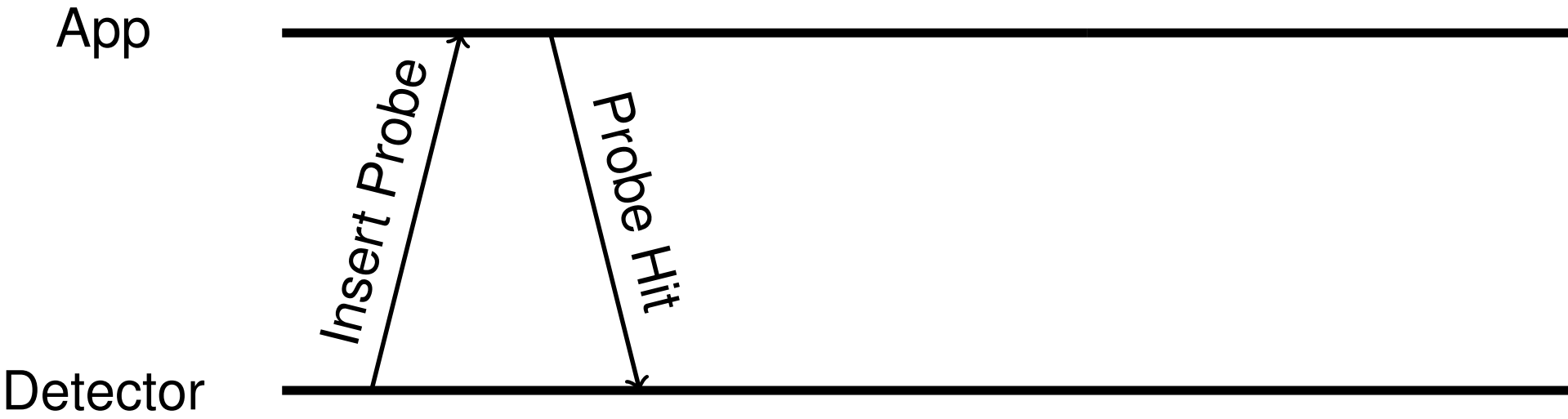
App

Detector

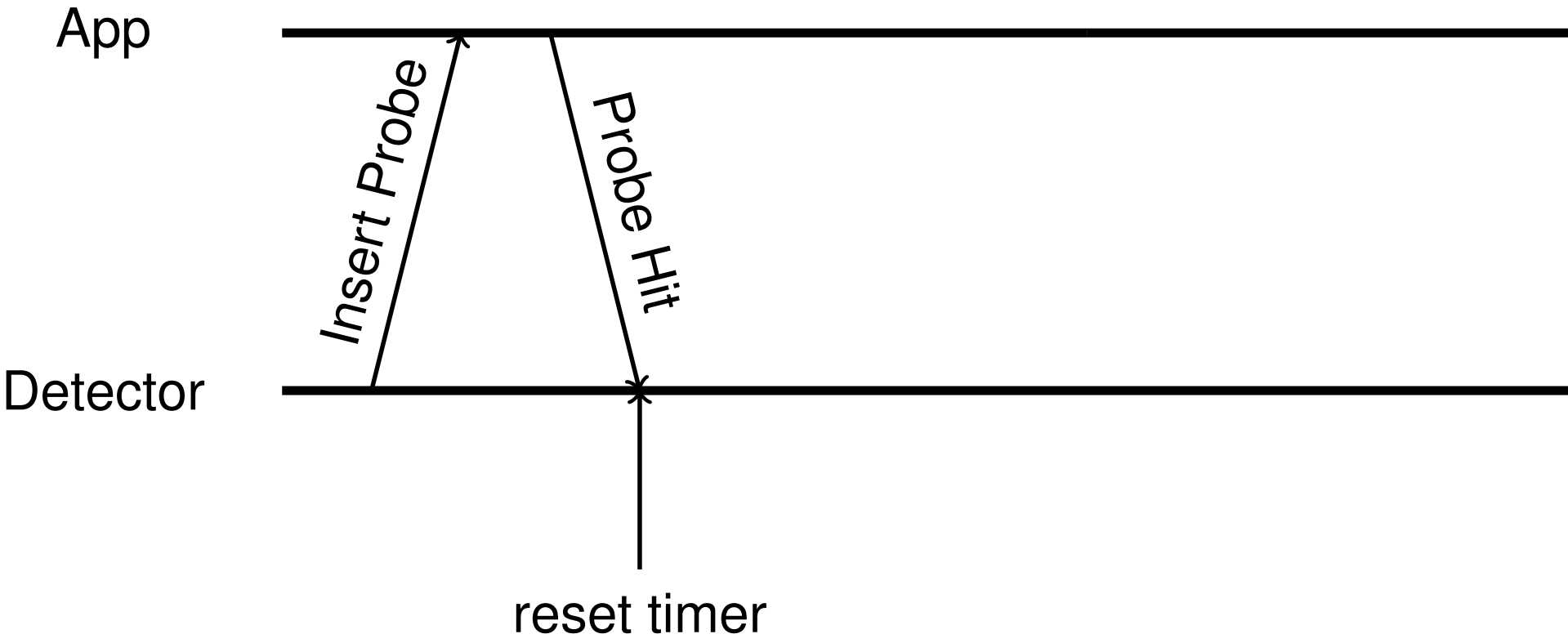
Heartbeat/watchdog



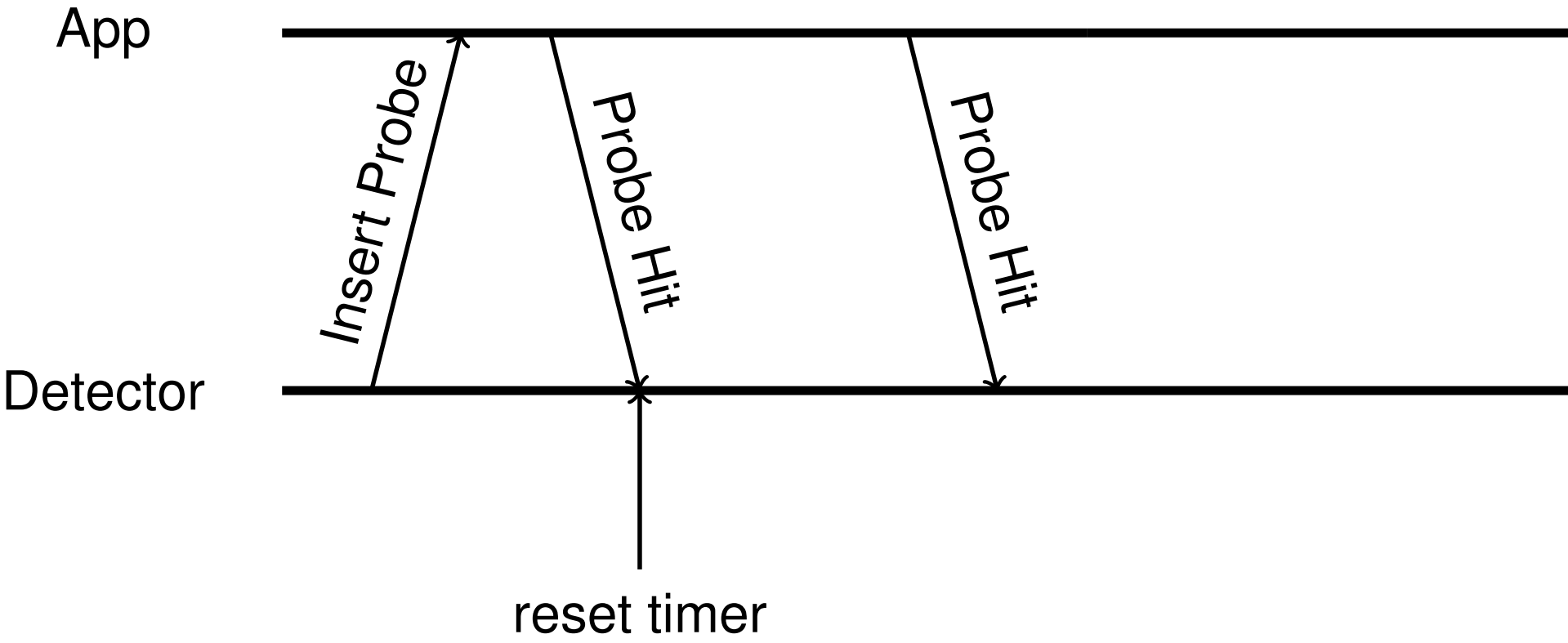
Heartbeat/watchdog



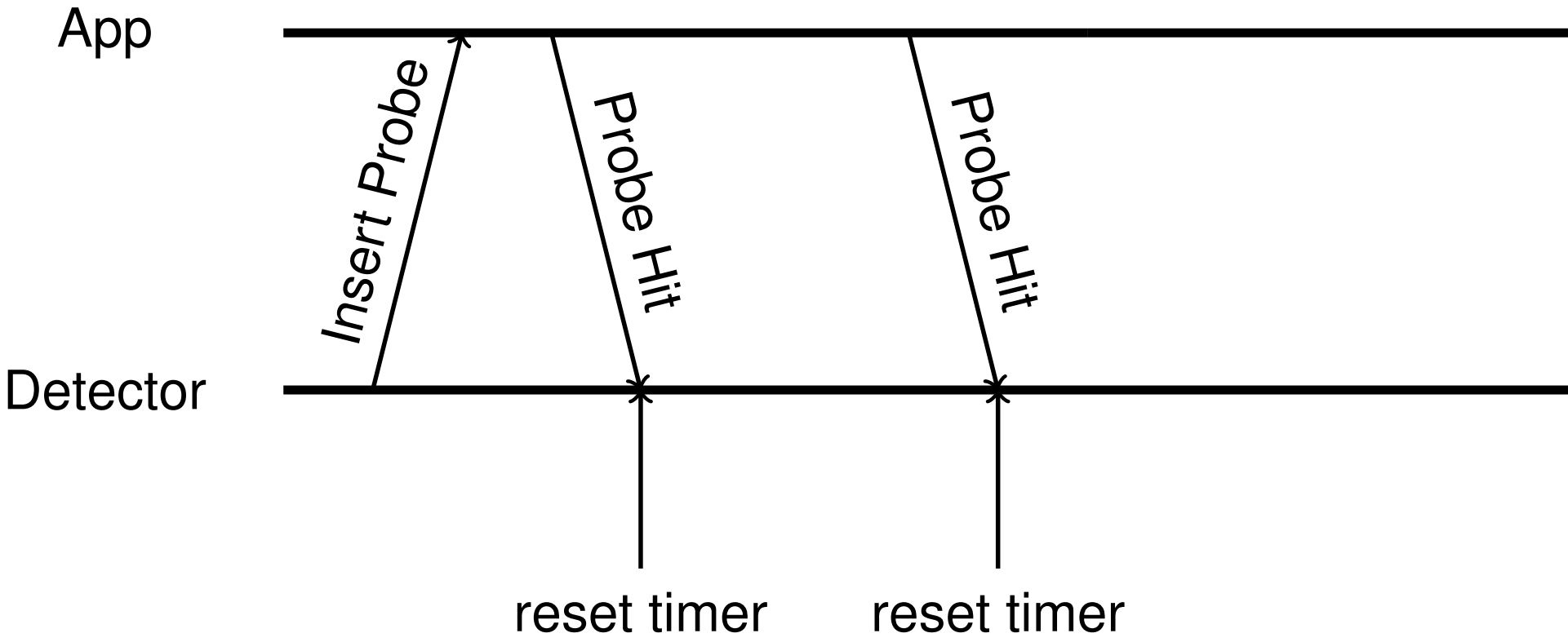
Heartbeat/watchdog



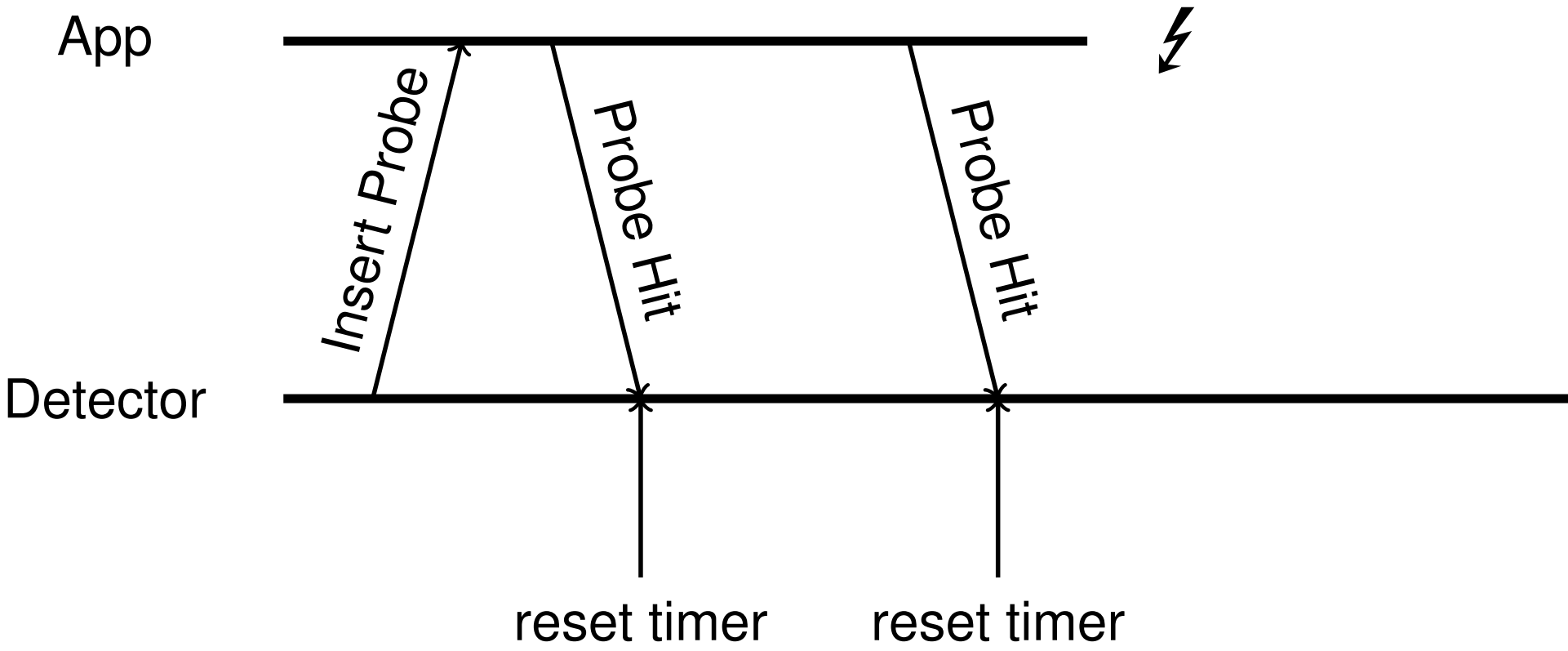
Heartbeat/watchdog



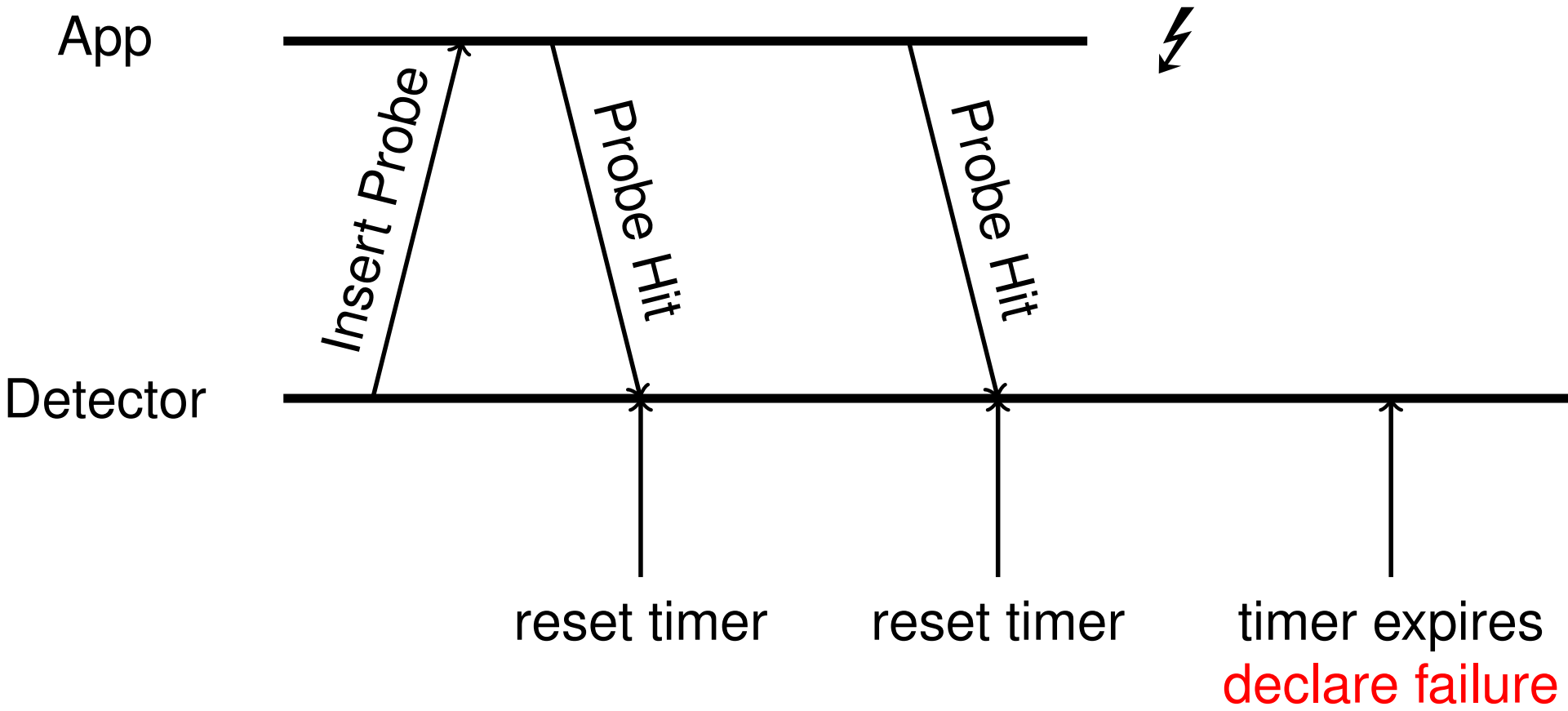
Heartbeat/watchdog



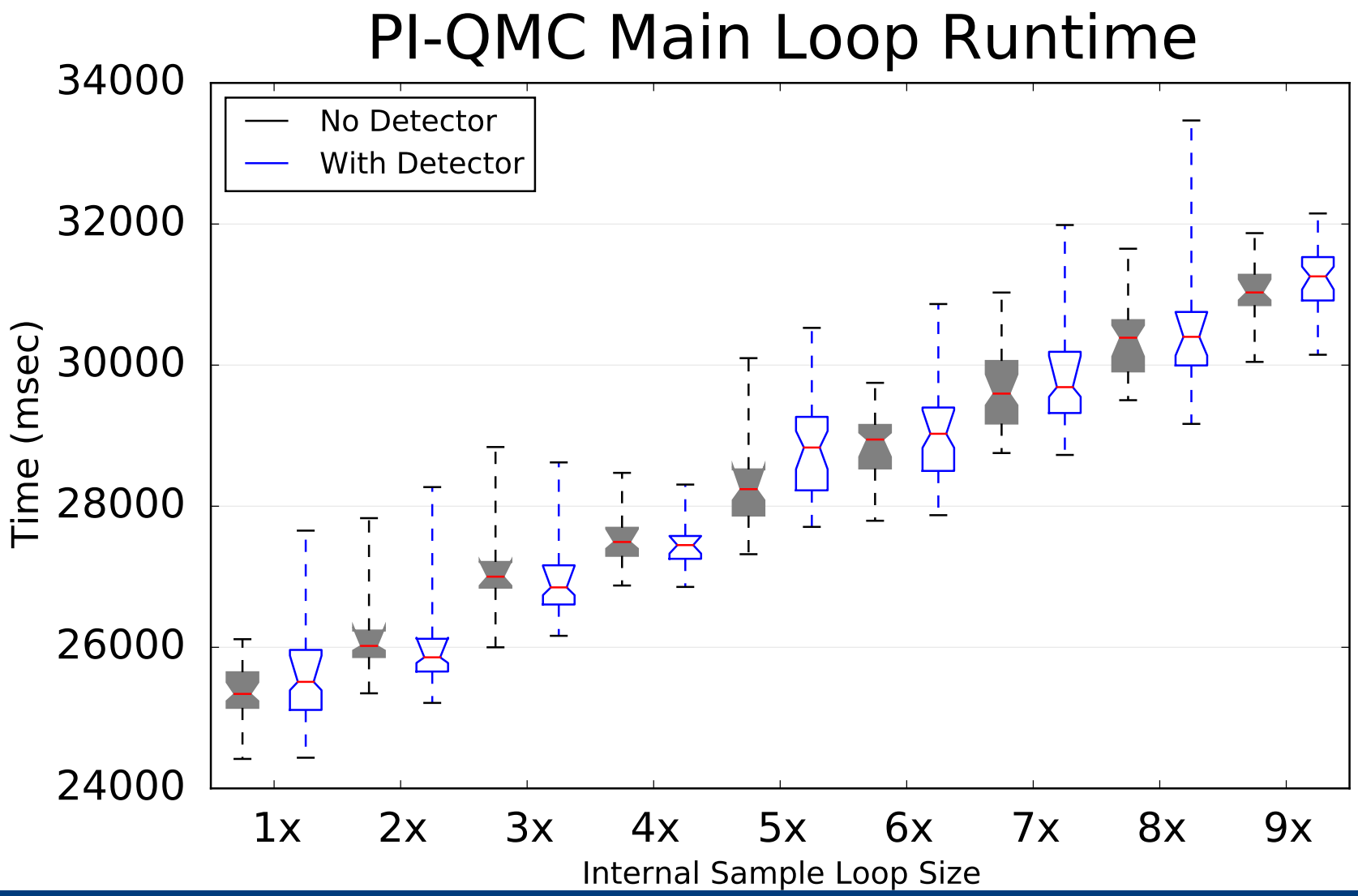
Heartbeat/watchdog



Heartbeat/watchdog



Watchdog - Performance



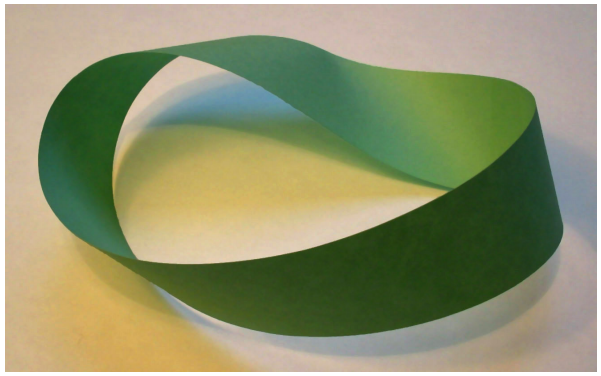
Detectors

Infinite Loop Detector

Detectors

Infinite Loop Detector

- Kernel or App-level
- Previously determined threshold
- Or register



Infinite Loop Detector

```
for(i=0; i<10; i++) {
  ...
}
//after loop
```

Infinite Loop Detector

1st Probe
(counter) →

```
for(i=0; i<10; i++) {
  ...
}
//after loop
```

Infinite Loop Detector

1st Probe (counter) →

2nd Probe (reset) →

```
for(i=0; i<10; i++) {
  ...
}
//after loop
```

Without Infinite Loop

Application	**Time (s)**	**95% CI** (s)	**% overhead**
Normal	1.13	0.0325	N/A
Naïve ILD - Page	1.26	0.0229	11.5
Naïve ILD - No Page	1.26	0.0265	11.8
Smart ILD - Page	1.14	0.0267	1.15
Smart ILD - No Page	1.15	0.0215	1.9

Conclusions

VMI

- Aims to achieve high fidelity monitoring while maintaining monitor fety
- Simplifies writing memory introspection based VMI monitors
- Passive monitoring has some inherent weaknesses

HW Invariant Techniques

- Ties high-level operations to fundamental HW functionality
- Protected against in-VM attackers
- Limited in scope?

Hook-Based Techniques

- Detectors are invoked on execution events
- Simple and flexible
- But... you need to understand execution

VM Monitoring Overview

	HyperTap (TBD)	Hprobes (DSN'14)	LiveWire (NDSS'03)	VMWatcher (CCS'07)	LibVMI (ACSAC'07)	SIM (CCS'09)	Lares (SP'08)	Lycosid (VEE'08)	Antfarm (ATC'06)	Nitro (AICS'11)	Ether (CCS'08)	Osck (ASPLOS'11)	Virtuoso (SP'11)	VMST (SP'12)	TxIntro (HPCA'14)
Root-of-trust (invariant)	**HW**	OS	OS	OS	OS	OS	OS	HW	OS	OS	OS	OS	OS	OS	OS
Active/**P**olling Mon.	**A**	**A**	P	P	P	**A**	**A**	P	P	P	P	P	P	P	P
Changes to VM	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Custom Auditors	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓
Online Detection	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
Auto-generate Invariants	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗
Userspace Monitoring	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗

Disclaimer: there are some gray areas!

Thank You!

Summary

- Key Concepts
- Validation
- Virtual Machine Introspection
- Hardware Invariant Techniques
- Hook-based Monitoring