Virtual Machines for Security

What is (system) virtual machine?

- **Virtual machine:** Complete compute environment with its own isolated processing capabilities, memory and communication channels.
  - Virtual machine is an efficient isolated duplicate of the physical machine [Goldberg, Popek]
- **Virtual Machine Monitor/Hypervisor:** System software that creates and manages virtual machines
- **Desirable qualities of a VMM [Goldberg/Popek]:**
  - **Equivalence:** Virtual m/c interface similar to real m/c
  - **Safety/Isolation:** Each VM should be isolated from other
  - **Low performance overhead:** Perf. close to real m/c

Without virtualization (bare metal)

Application
1

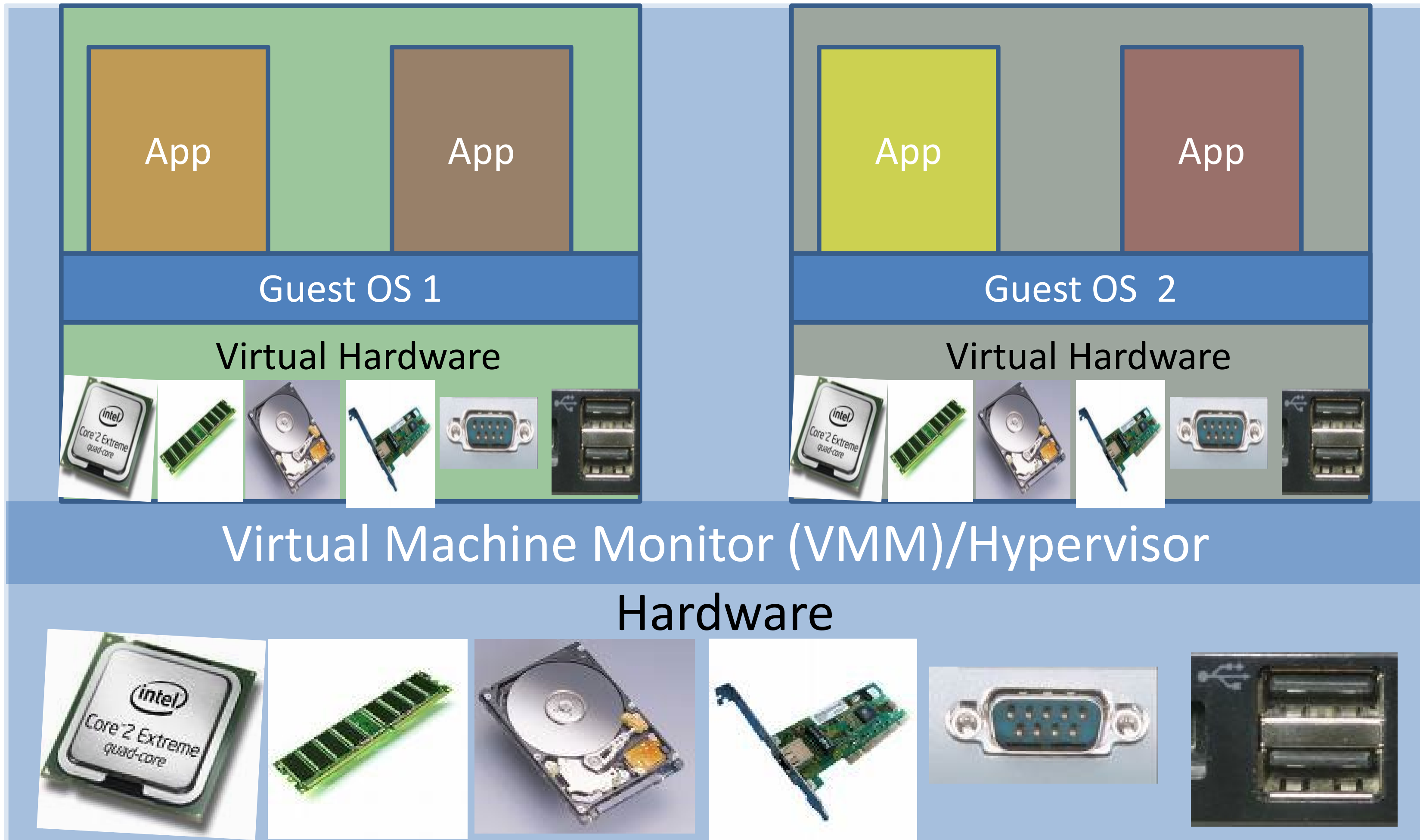
Application
2

OS

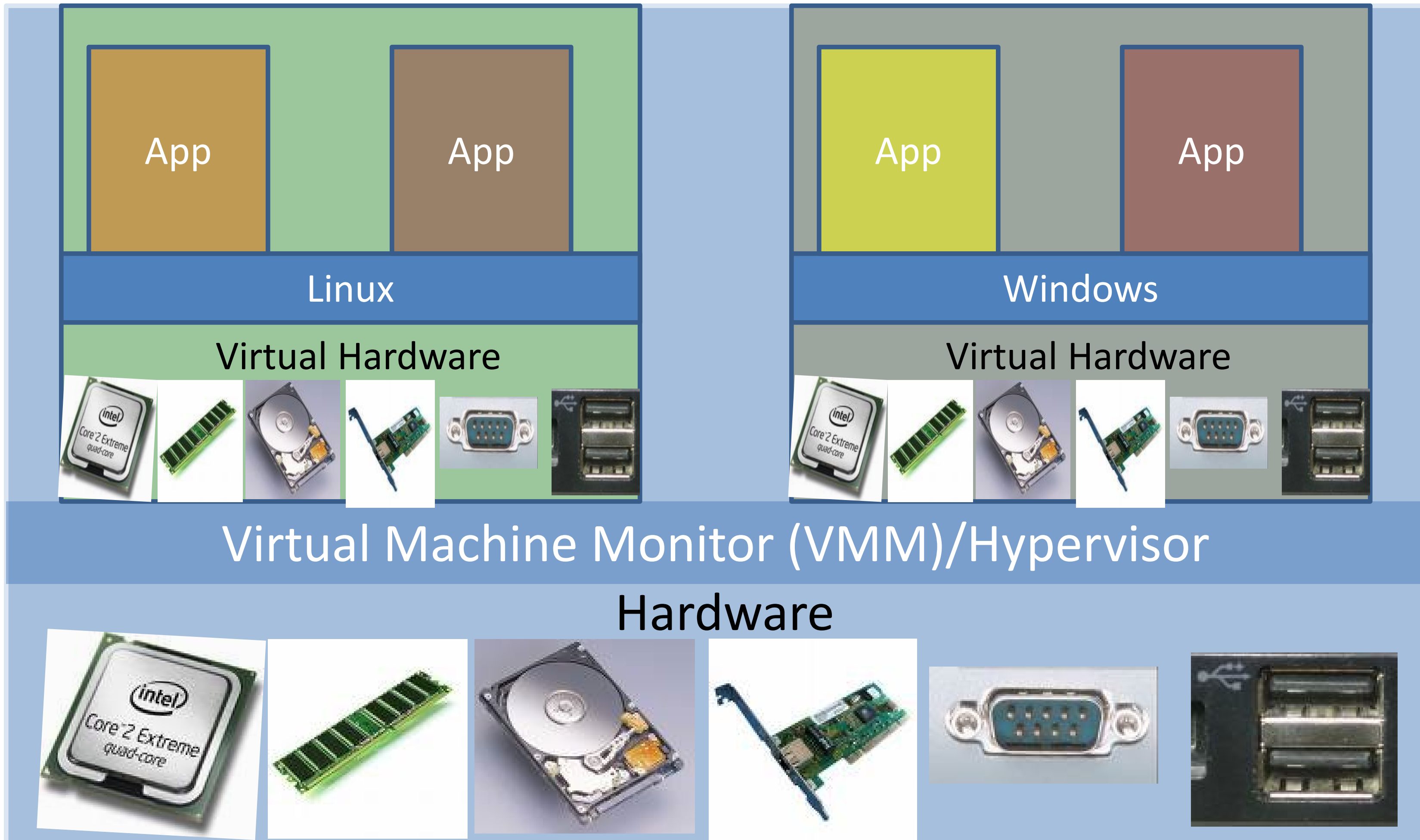
Hardware

Slide Sorav Bansal, IIT-Delhi

With virtualization



Different OS on same H/W



Why Virtual Machines?

- **Operating system diversity:** Can run both Linux and Windows on same h/w
- **Security/Isolation:** Hypervisor separates the VMs from each other and isolates VMs from H/W
- **Rapid provisioning/Cloud/ Server consolidation:** On demand provisioning of hardware resources

Why Virtual Machines?

- **Operating system diversity:** Can run both Linux and Windows on same h/w
- **Security/Isolation:** Hypervisor separates the VMs from each other and isolates VMs from H/W
- **Rapid provisioning/Server consolidation:** On demand provisioning of hardware resources
- **High availability/Load balancing:** Ability to live-migrate a VM to other physical server
- **Encapsulation:** The execution environment of an application is encapsulated within VM

Other “virtual machines(?)”

- Language VM: Language runtime focused on running a single application
  - e.g., Java Virtual Machine, Microsoft Common Language runtime, Javascripts
- “Lightweight” VM: Does not run guest OS; Isolate applications from other
  - e.g., Docker, FreeBSD’s Jail, Google’s Native Client
- Our focus is System virtual machine: presents a “copy” of whole machine

Types of system virtual machine

- Type 1 (Bare metal): VMM runs on bare metal; directly control the physical machine
  - e.g., Xen, VMWare ESX server

- Type 2 (Hosted): VMM runs as part of/on top of an OS
  - e.g., KVM, VMWare workstation

Implementing a VMM

- Three pieces of a system:
  - Instructions -- defined by the ISA (e.g., x86, ARM)
  - Memory
  - I/O (network, disk)

- Mostly a quick discussion of the first two. We won’t discuss the third in this course. For a detailed discussion of virtualization, take E0253!

Direct Execution + ???

- **Idea:** Run *most* instructions of the VM directly on the hardware
- **Advantage:** Performance *close* to native execution
- **Challenge:** How to ensure isolation/protection ?
  - If all instructions of the VM are directly executed then VMM has no control !

Direct Execution + Trap and Emulate

- **Idea:**
  - Trap to hypervisor when the VM tries to execute an instruction that could change the state of the system/take control (i.e., impact safety/isolation).
  - Emulate execution of these instruction in hypervisor
  - Direct execution of any other innocuous instructions on h/w that cannot impact other VMs or the hypervisor

How to do Trap and Emulate?

- Generally two categories of instructions:
  - User instructions:
    - Typically compute instructions
    - e.g., *add, mult, ld, store, jmp*
  - System instructions:
    - Typically for system management
    - e.g., *iret, invlpg, hlt, in, out*
- Two mode of CPU operation:
  - User mode (in x86-64 typically ring 3)
  - Privileged mode (in x86-64 ring 0)
  - Attempt to execute system instructions in user mode generates trap/general protection fault (gpf)

How to do Trap and Emulate?

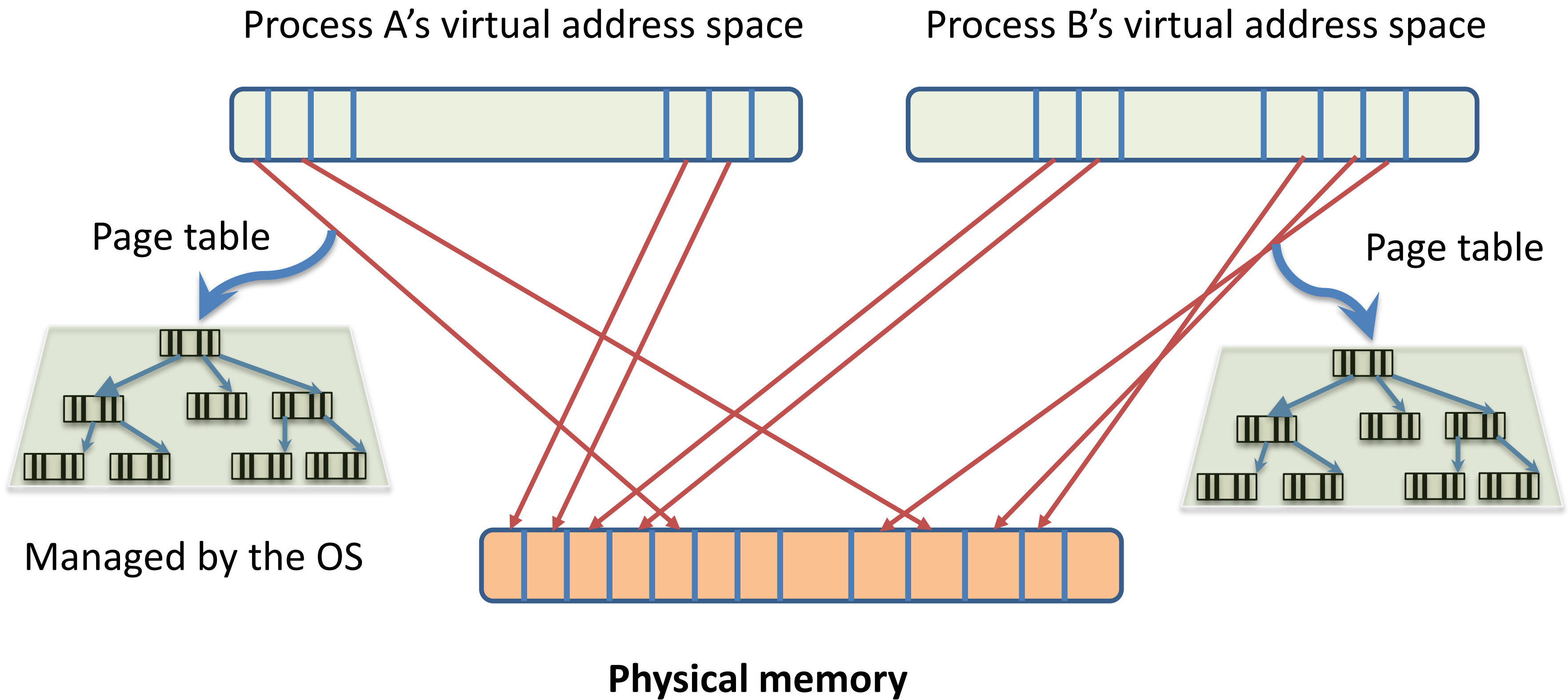
- System state:
  - Example, control registers like *cr3* (remember?)
  - Access to system state in user mode trigger gpf
- Idea:
  - Run the VM in user mode (ring 3), while the hypervisor in privileged mode (ring 0)
  - Anytime VM tries to execute an system instr. or tries to access system state, trap to VMM

Formalizing Direct exec. + Trap & emulate

- Goldberg & Popek theorem:
  - Control sensitive intr.: Instructions that can update system state
  - Behavior sensitive instr.: If instructions behavior depends upon system state
- Requirement for architecture/ISA to be virtualizable via trap & emulate:

{control sensitive} U {behavior sensitive} ⊆ {privileged}

Virtual Memory in un-virtualized system

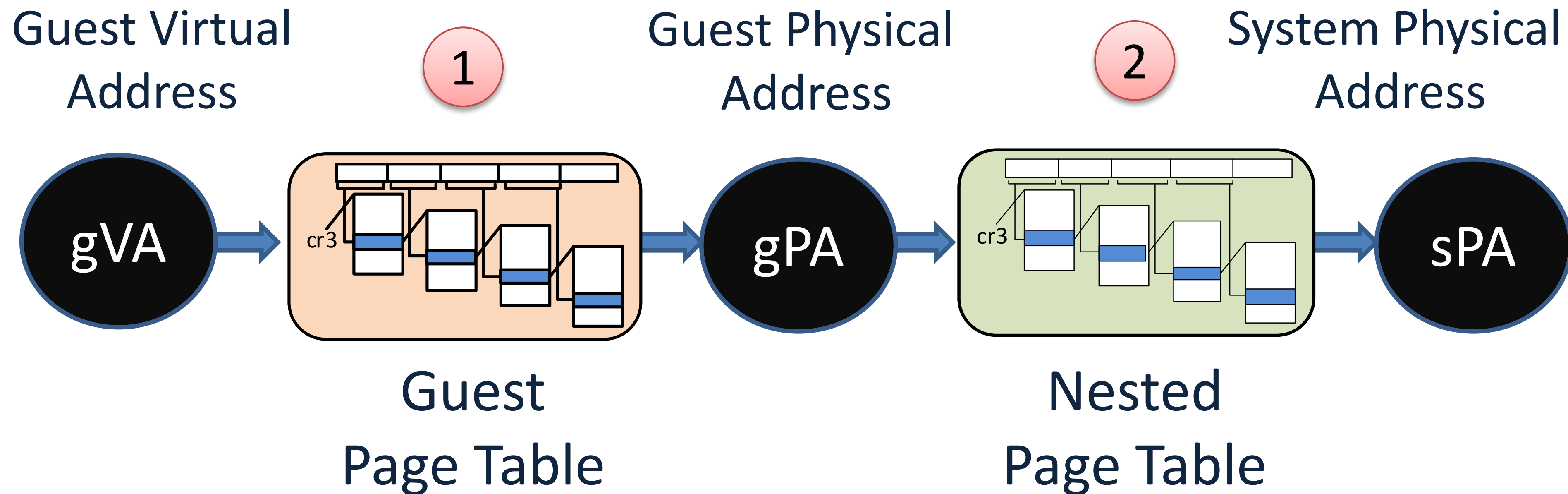


Requirement for memory virtualization

- **No** direct access to physical memory from VM
- **Only** hypervisor should manage physical memory
- But, fool the guest OS to think that it is accessing physical memory

Virtualizing Virtual Memory

Virtualizing Virtual Memory



Two levels of address translation on each memory access by an application running inside a VM

Based on "Efficient Memory Virtualization"
Gandhi et. al., MICRO'14

Applications of VM

- A.K.A. Virtual Machine Introspection (VMI)
- Rootkit (OS malware) detection
- Better network intrusion detection
- Challenges:
  - Semantic Gap problem.

Layered computer system design

Modern computer systems are built using layers of abstraction

	User app	User app	…	
Utilities & Libraries	`ls`, `ps`, & `bash` utilities	`libc`	`gcc`	…
Operating System	Kernel Code	Process List	Syscalls, IDT	…
Hardware	CPU	Memory	I/O devices	

Fundamental principle in security

The lower you go, the more control you have

Least control

Most control

User app

User app

. . .

Utilities & Libraries

`ls`, `ps`, & `bash` utilities

`libc`

`gcc`

. . .

Operating System

Kernel Code

Process List

Syscalls IDT

. . .

Hardware

CPU

Memory

I/O devices

Example: Malware detection

User app

Utilities & Libraries

Operating System

Hardware

Example: Malware detection

User app

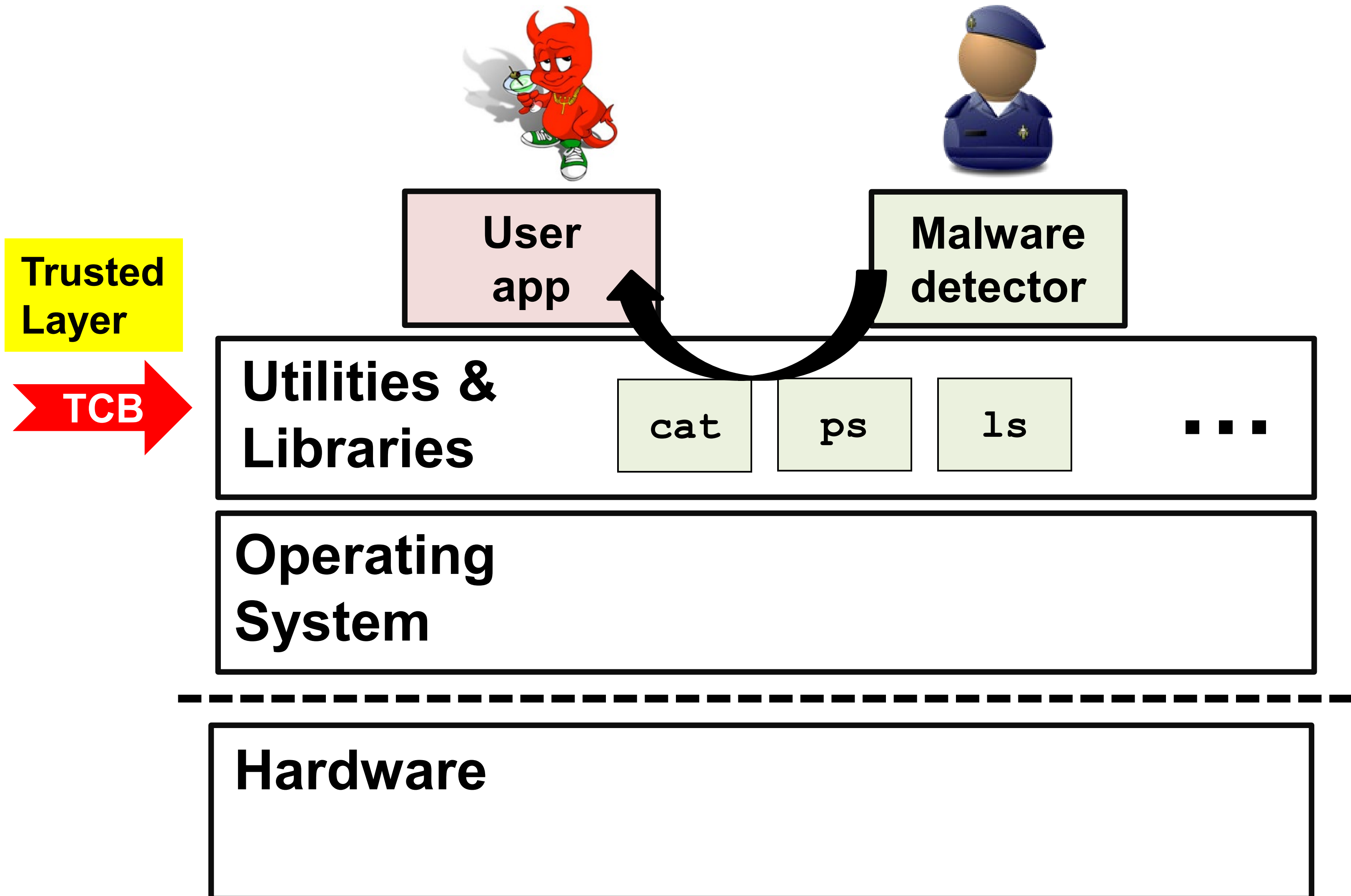
Malware detector

Utilities & Libraries

Operating System

Hardware

Example: Malware detection



But utilities may be compromised!

User app

Malware detector

Utilities & Libraries

`cat` `ps` `ls`

Operating System

Hardware

But utilities may be compromised!

(1) Show me file contents

User app

Malware detector

(1)

Utilities & Libraries

cat

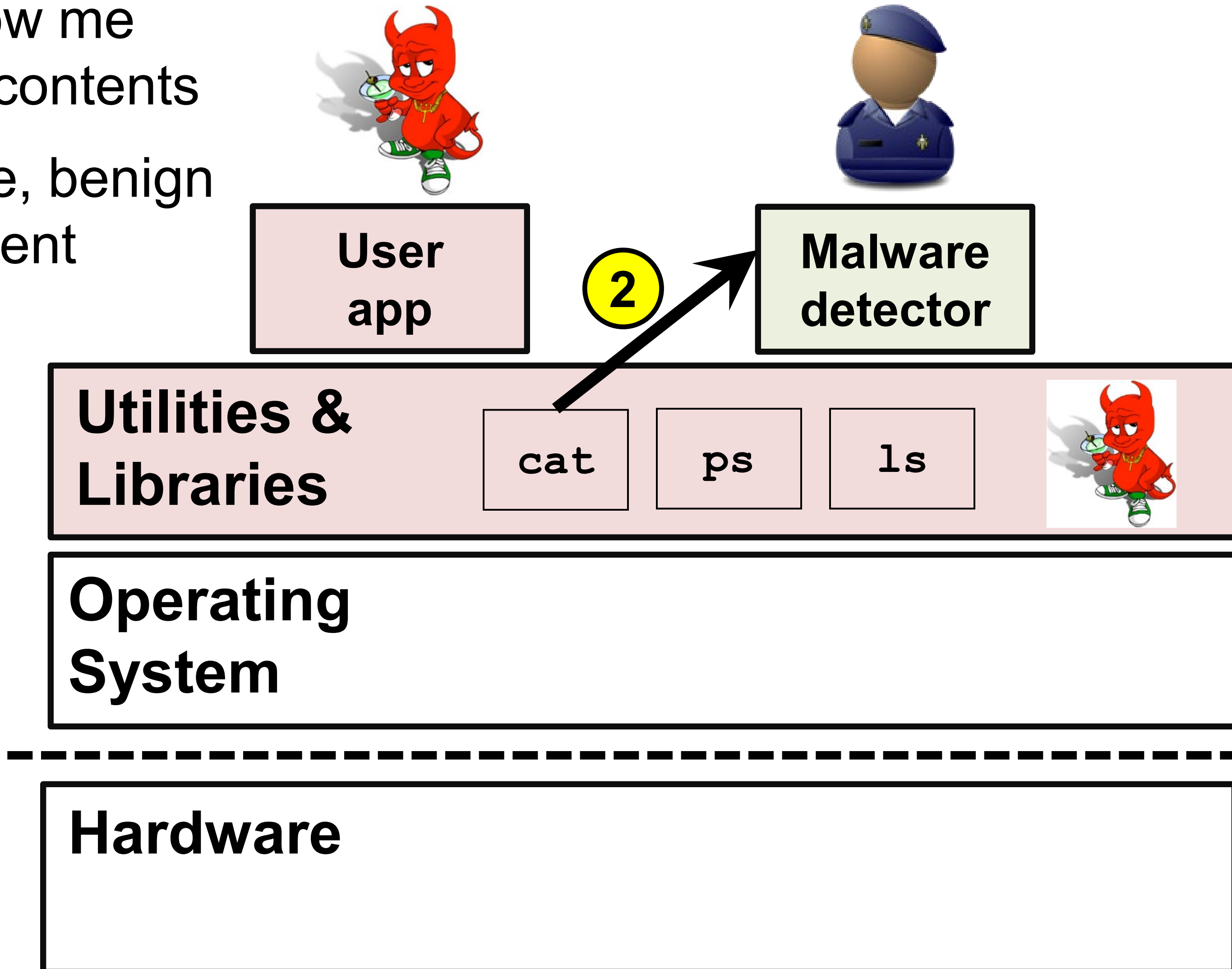
ps

ls

Operating System

Hardware

But utilities may be compromised!



Solution: Query the OS

① Query with syscall

User app

Malware detector

Utilities & Libraries

①

TCB

Operating System

System call API

Hardware

Solution: Query the OS

1 Query with syscall

2 OS reads file

User app

Malware detector

Utilities & Libraries

2

TCB

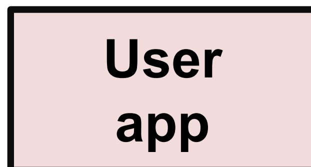
Operating System

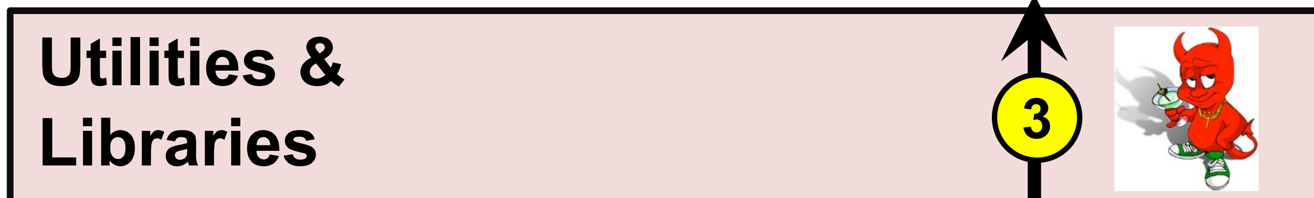
System call API

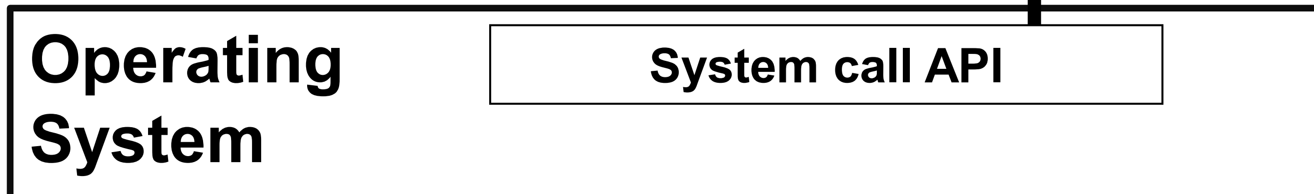
Hardware

Solution: Query the OS

1. Query with syscall
2. OS reads file
3. Returns true file content





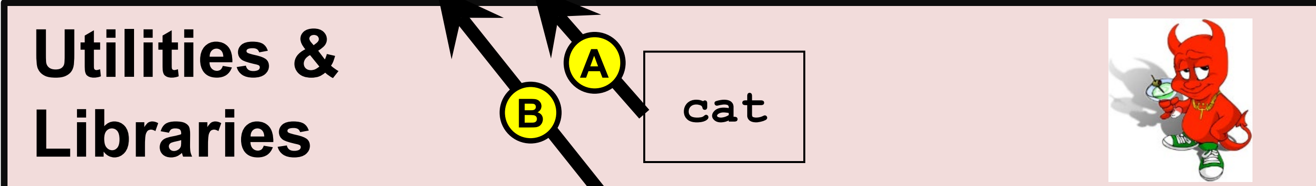


OS detects malicious utilities too

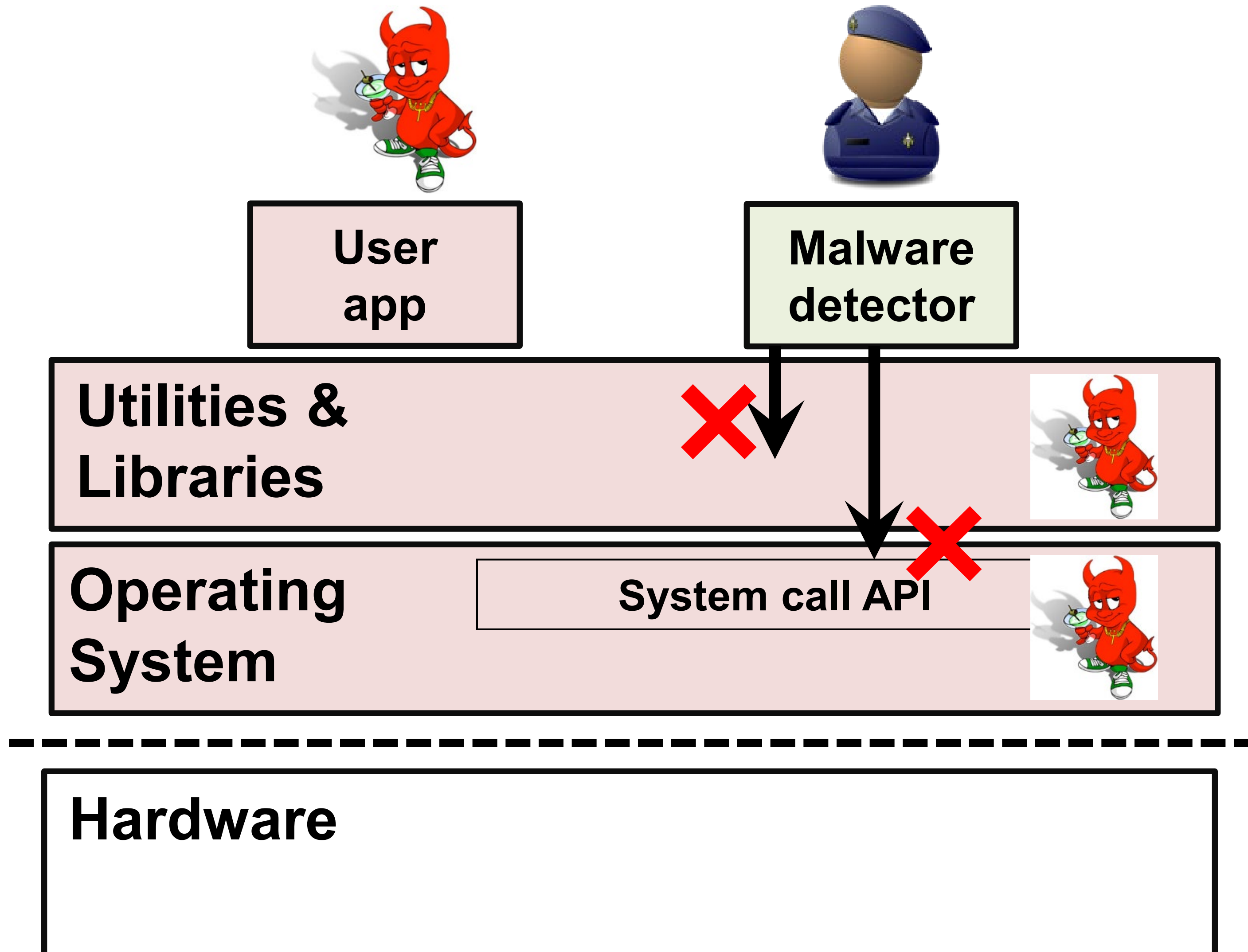
(A) `cat` file

(B) Read file

diff (A) vs (B) ?



What if the OS is malicious?



Rootkit = Malware that infects OS

Rootkits hide malware from detectors → Long-term stealth

BACK DOOR

...

Malware detector

Utilities & Libraries

Operating System

System call API

Hardware

How does an OS get infected?

- **Exploits of kernel vulnerabilities**:
  - Injecting malicious code by exploiting a memory error in the kernel
- **Privilege escalation attacks**:
  - Exploit a `root` process and use resulting administrative privileges to update the kernel
- **Social engineering attacks**:
  - Trick user into installing fake kernel updates
    - Defeated via signature verification of kernel updates
    - Trivial to perform prior to the Windows Vista OS

How prevalent are rootkits?

- **2010 Microsoft report**: 7% of all infections from client machines due to rootkits[1]
- **2016 HummingBad** Android rootkit:[2]
  - Up to 85 million Android devices infected?
  - Earns malware authors $300,000 each week through fraudulent mobile advertisements
- Used in many high-profile incidents:
  - Torpig and Storm botnets
  - Sony BMG (2005), Greek wiretapping (2004/5)

---

[1] Microsoft Malware Protection Center, "***Some Observations on Rootkits,***" January 2010, https://blogs.technet.microsoft.com/mmpc/2010/01/07/some-observations-on-rootkits

[2] CheckPoint Software, "***From HummingBad to Worse,***" July 2016, http://blog.checkpoint.com/wp-content/uploads/2016/07/HummingBad-Research-report_FINAL-62916.pdf

How can we detect rootkits?

Ask for help from the layers below

User app	Malware detector

Utilities & Libraries

Operating System — System call API

TCB → **Hypervisor (a.k.a. Virtual Machine Monitor)**

Hardware

How low can we go?

User app

Malware detector

Utilities & Libraries

Operating System

Hypervisor [Bluepill, Subvert]

Hardware

How low can we go?

User app

Malware detector

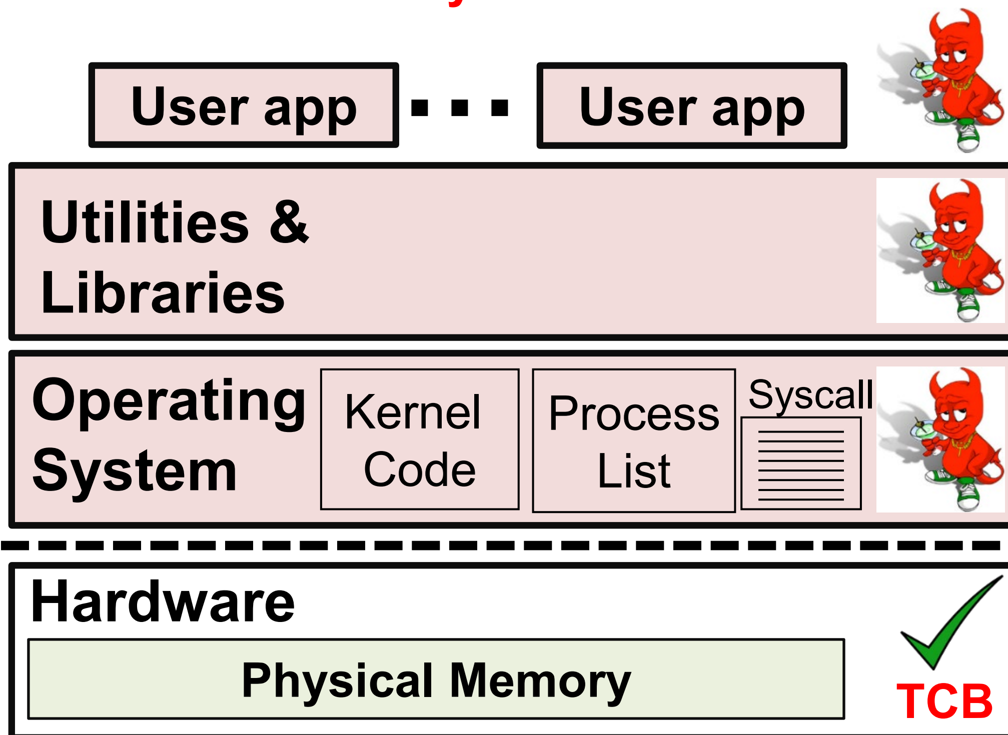
Utilities & Libraries

Operating System

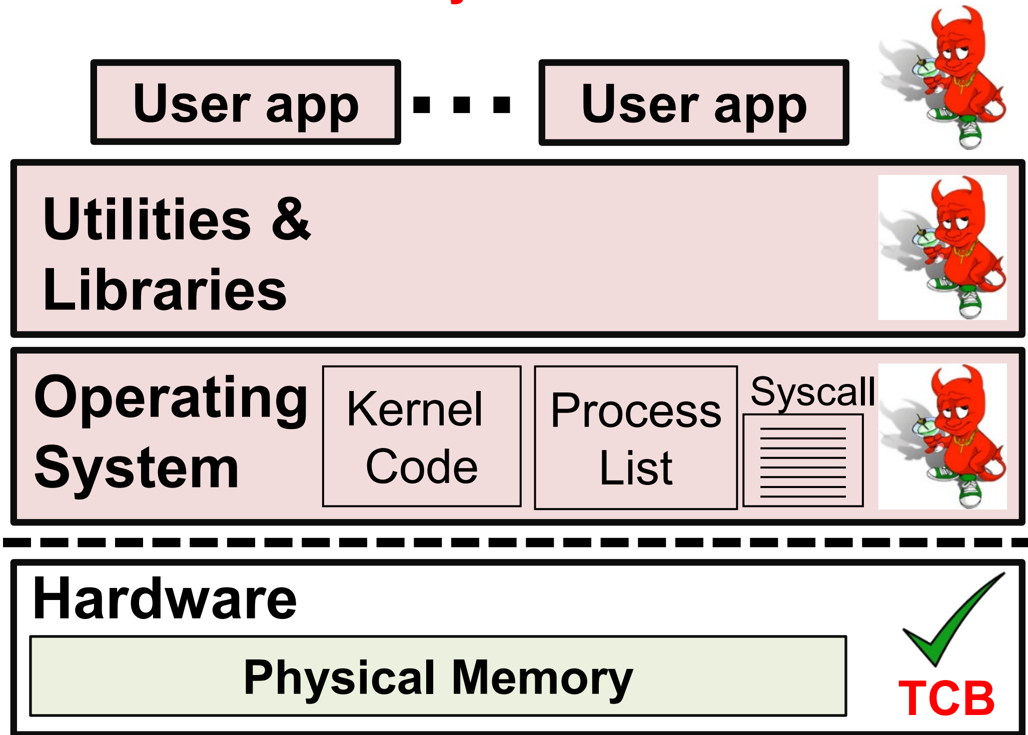
Hardware
[Stuxnet, Trojaned ICs]

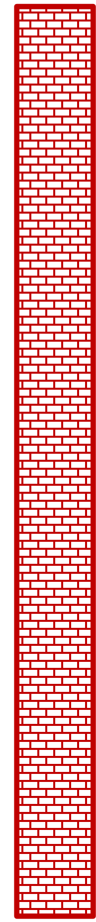
???

Analysis of memory snapshots obtained from target machine



Analysis of memory snapshots obtained from target machine





Analysis of memory snapshots obtained from target machine

Target machine
Potentially rootkit infected

User app ... User app

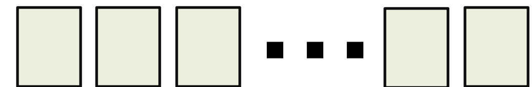
Utilities & Libraries

Operating System | Kernel Code | Process List | Syscall

Hardware
Physical Memory
TCB

Analysis machine
Trusted

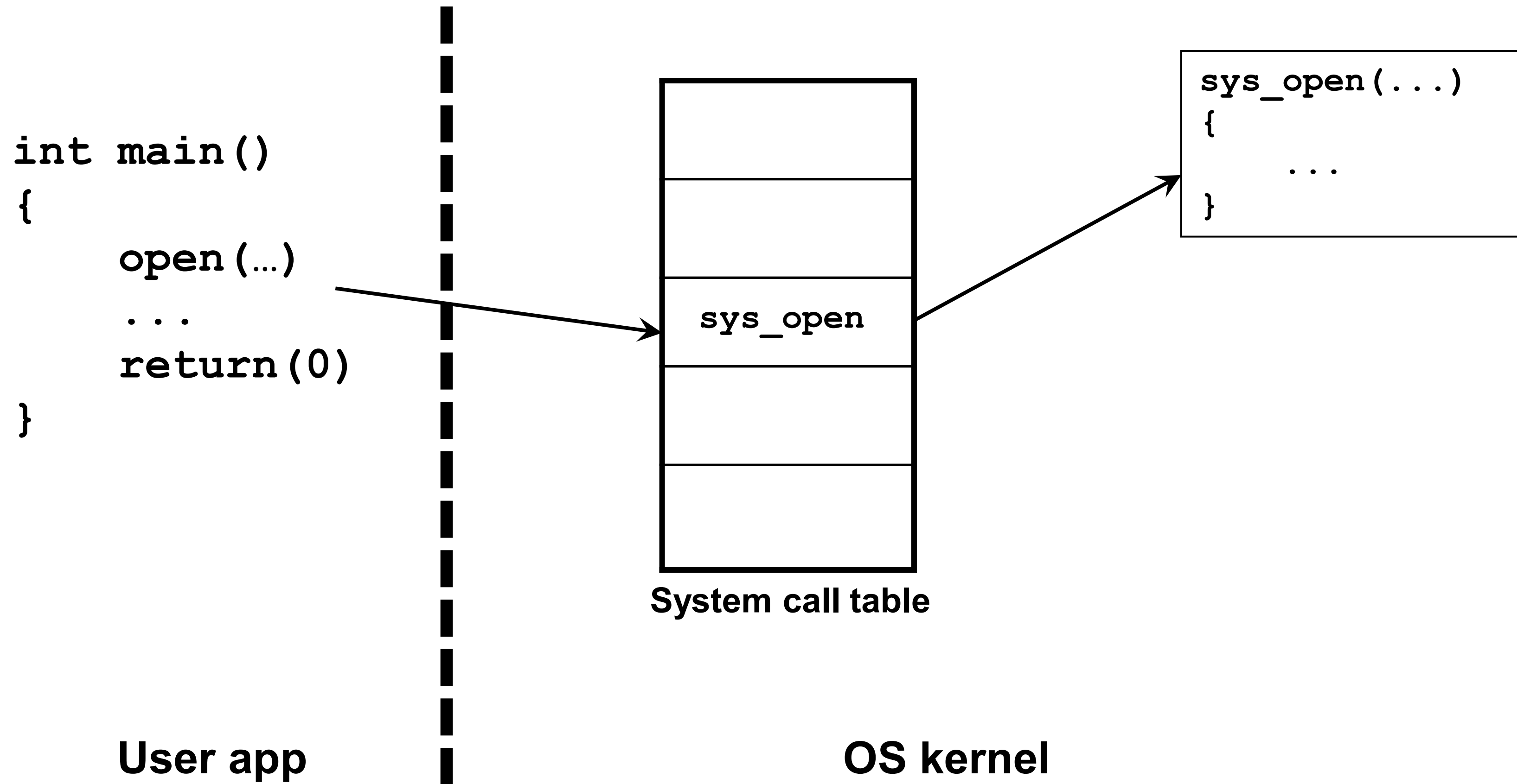
Snapshot of memory pages



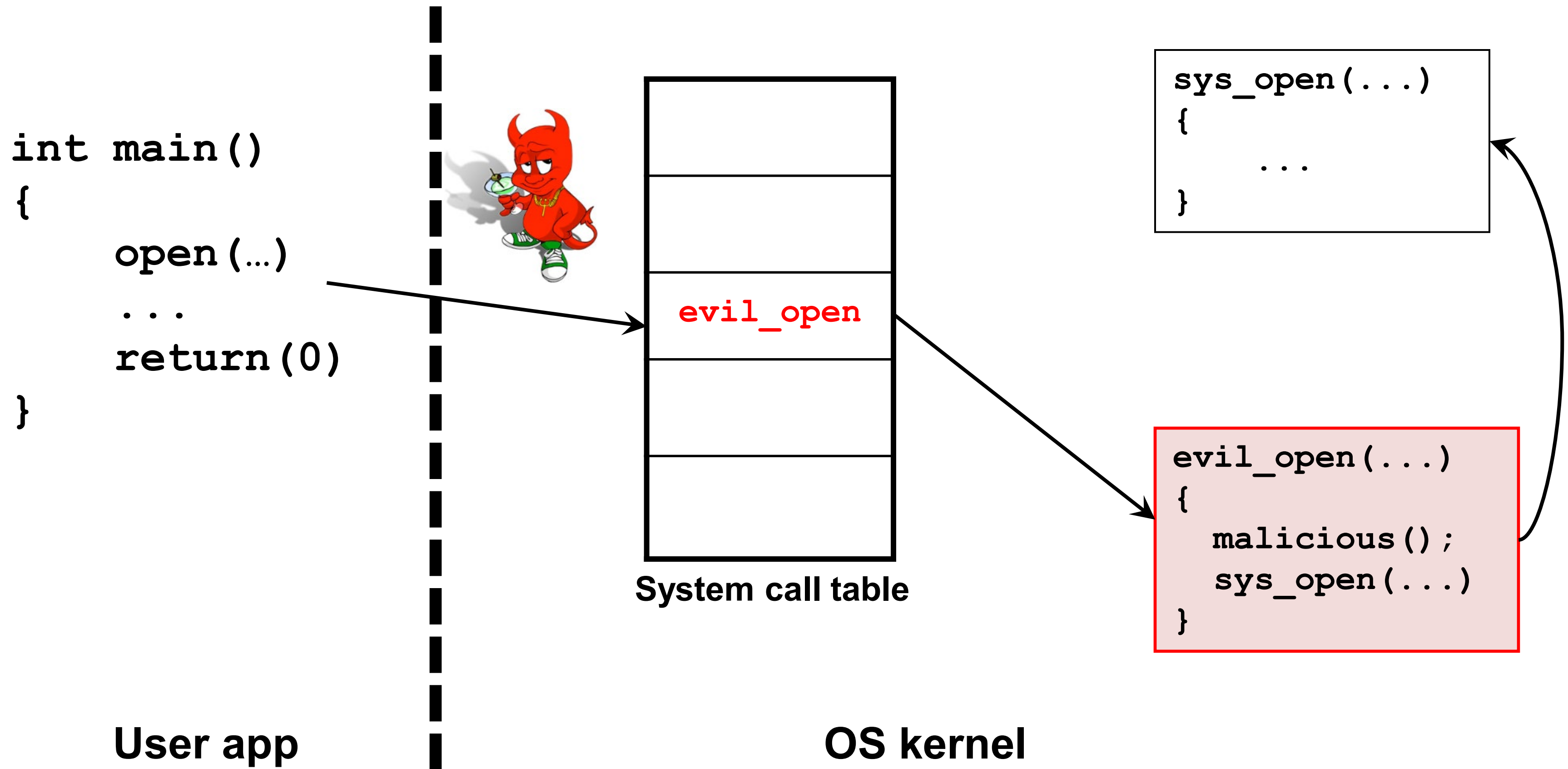
How to detect rootkits?

- What algorithm should we use for memory snapshot analysis?
  - Concerns the **policy**
  - Formulate rootkit detection problem as one of detecting *invariant violations*
- How can we fetch memory pages without involving the target's OS?
  - Concerns the **mechanism**
  - **That is the feature provided by virtual machines**

Example 1: Linux Adore rootkit

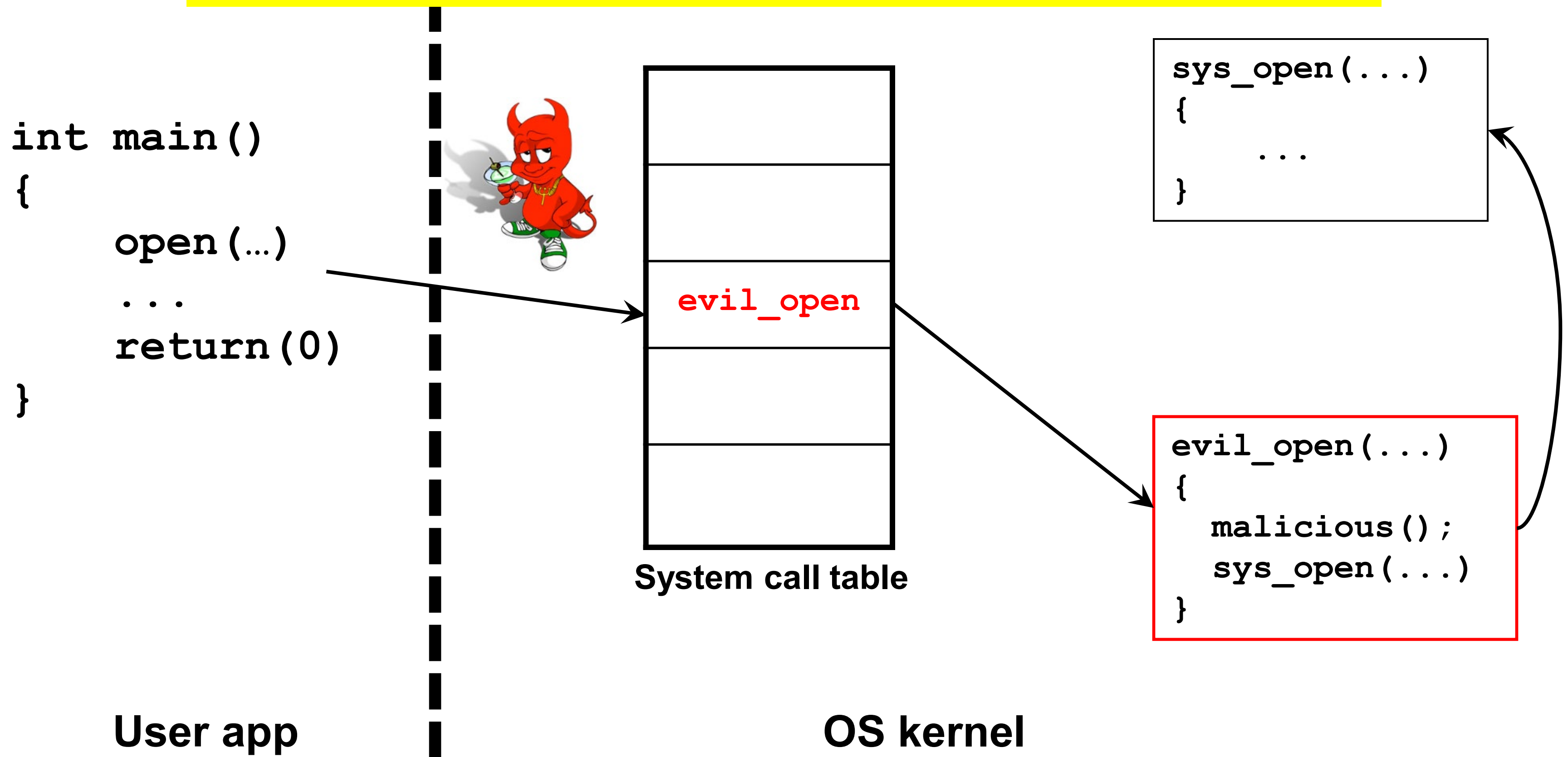


Example 1: Linux Adore rootkit



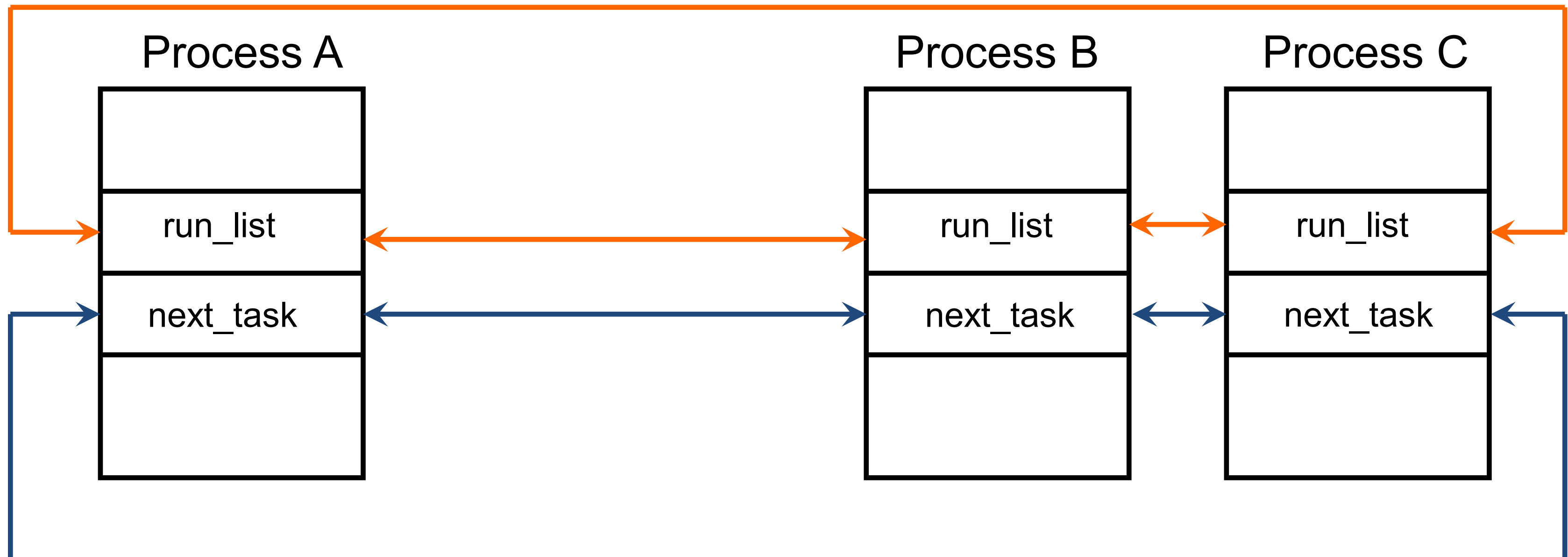
Example 1: Linux Adore rootkit

Violated: Function pointer values in system call table should not change



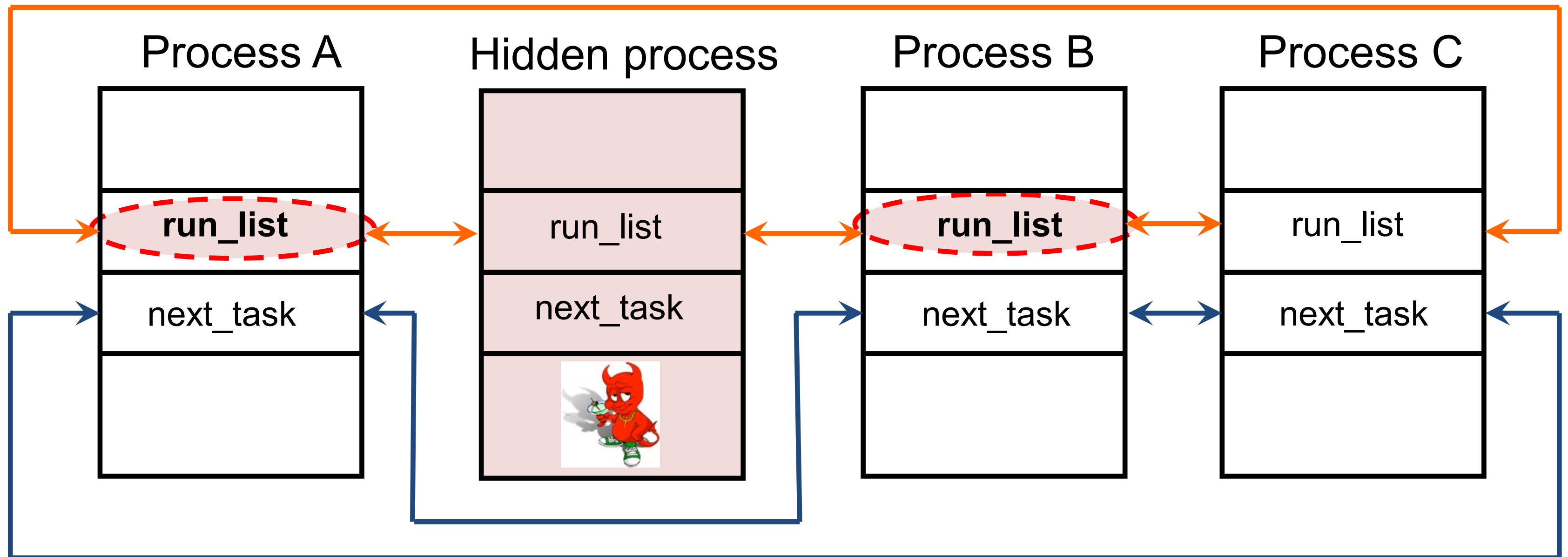
Example 2: Windows Fu rootkit

run-list: Used by the scheduler to select processes for execution



all-tasks: Used for process accounting

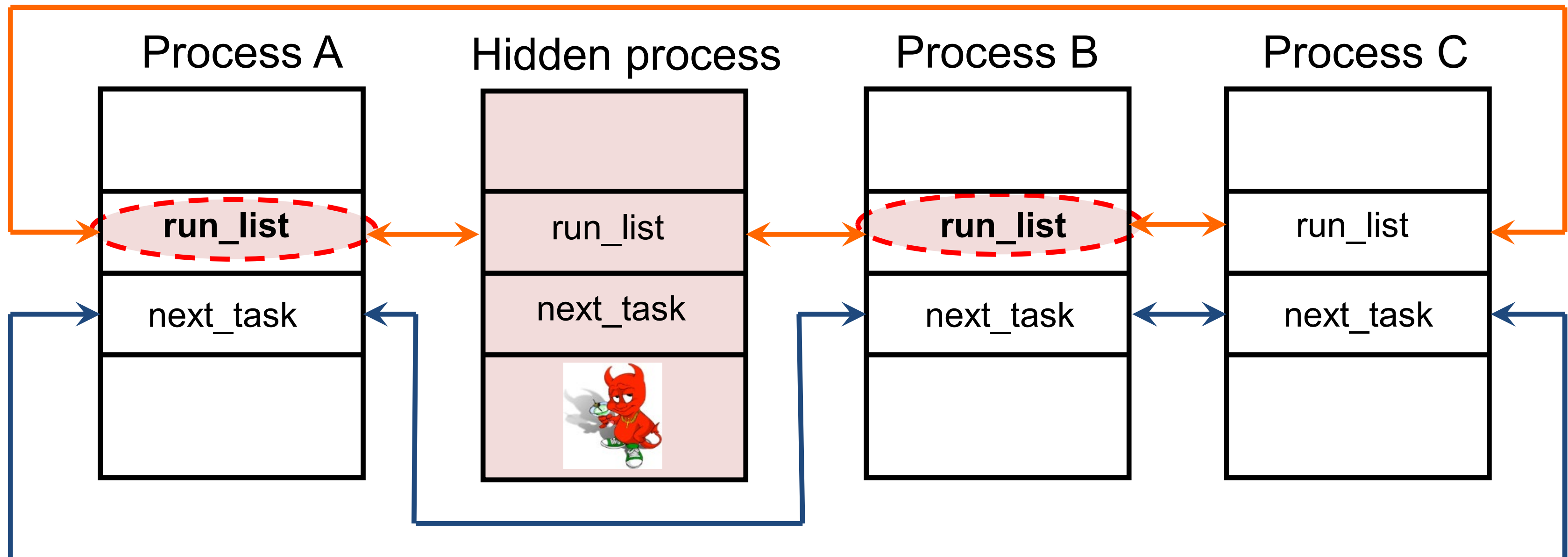
Example 2: Windows Fu rootkit



Example 2: Windows Fu rootkit

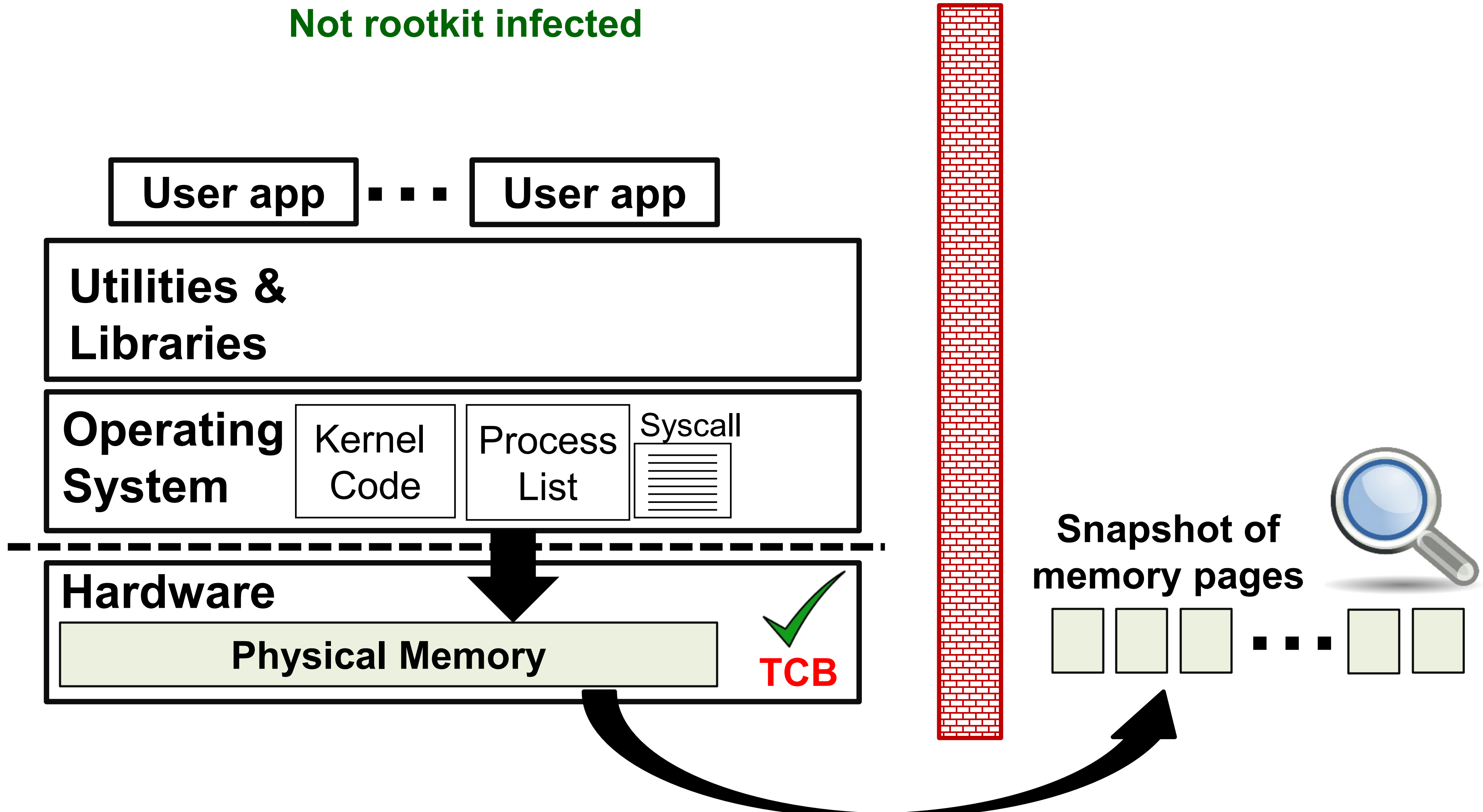
Violated: run-list ⊆ all-tasks

run-list: Used by the scheduler to select processes for execution

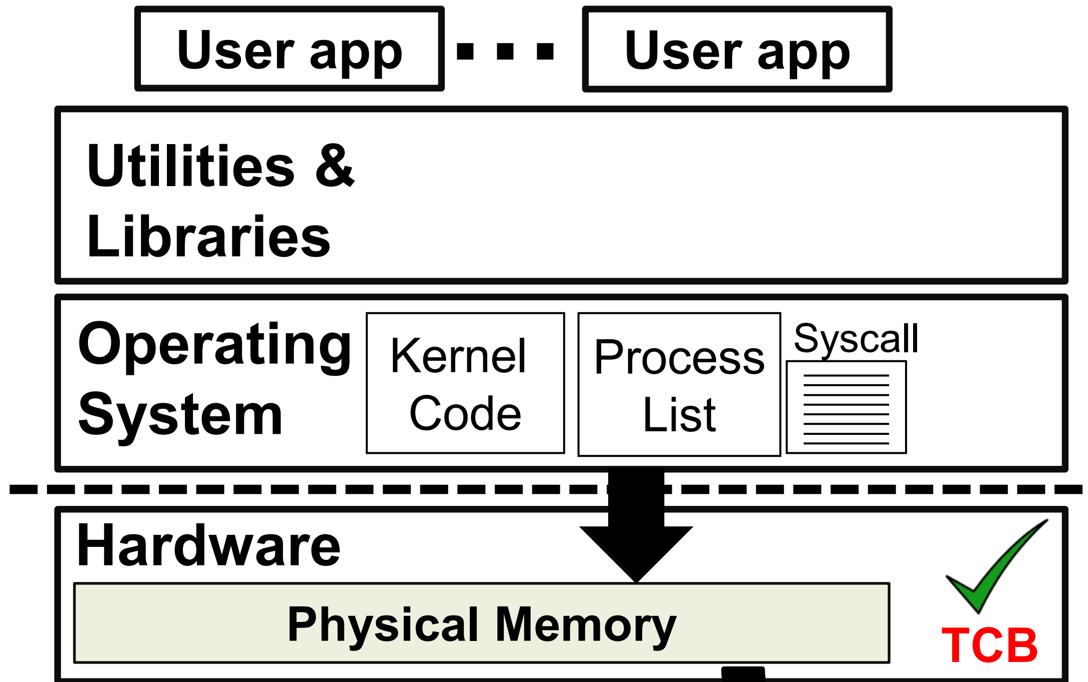


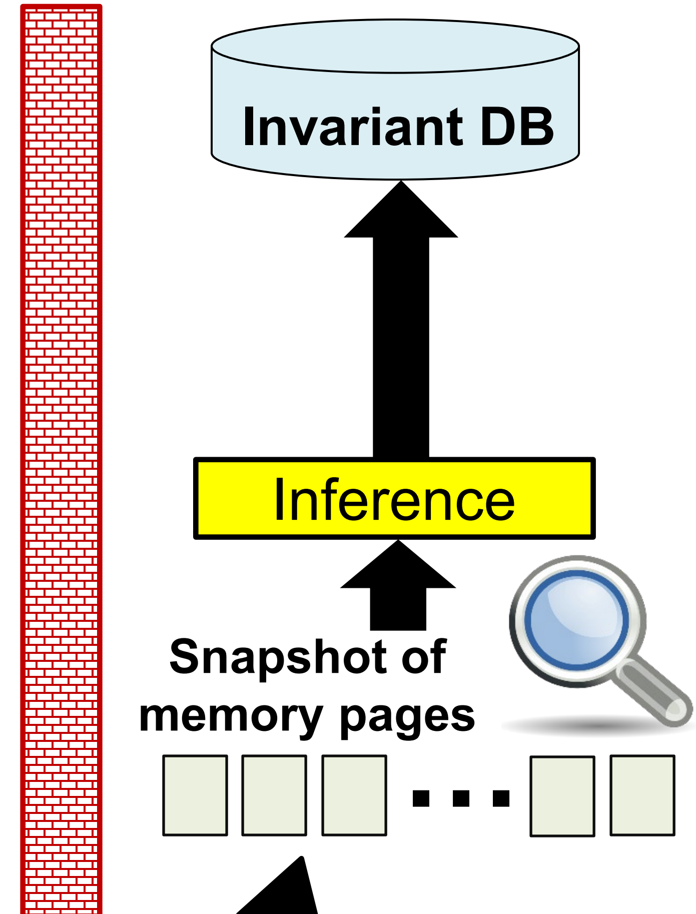
all-tasks: Used for process accounting

Offline training phase

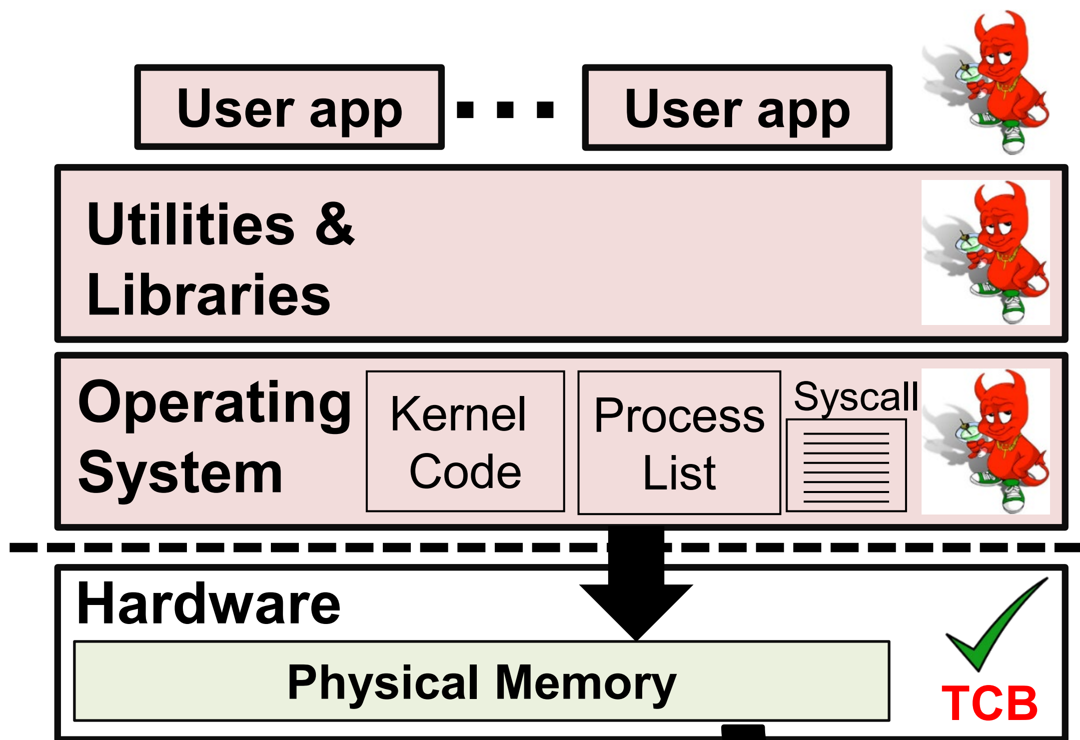


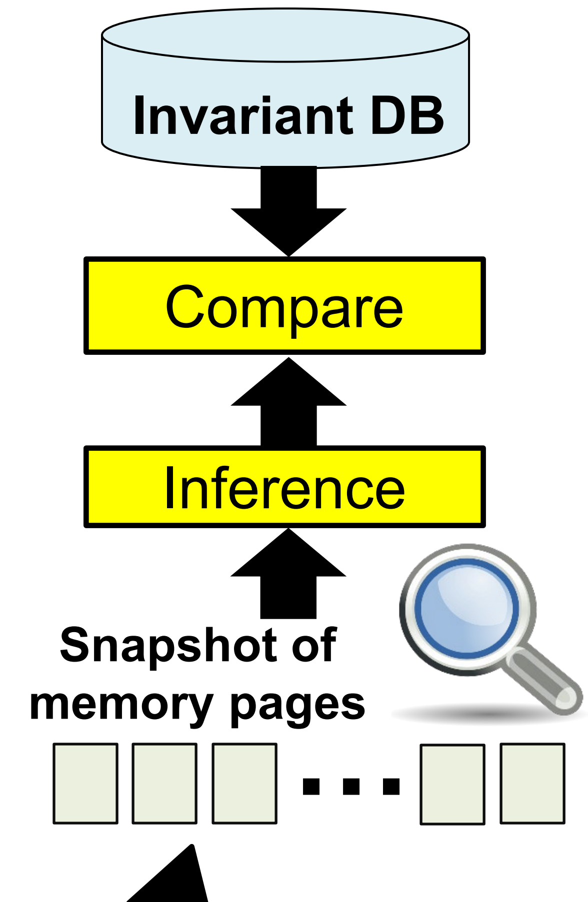
Offline training phase





Online enforcement phase





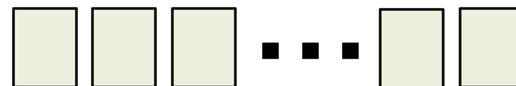
Reconstructing data structures

(1) Kernel data structure type definitions

```
struct task_struct {...}
struct list_head {...}
struct siginfo {...}
...
```

(2) Entry-points into the kernel

```
ffffe400 init_task
ffffe410 phys_base
ffffe420 loops_per_jiffy
...
```



(3) Snapshot of memory pages

How to interpret raw dump?

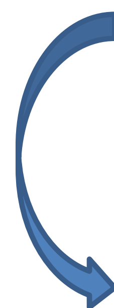
- This is the **semantic gap** problem
- Hypervisor only sees:
  - Raw memory snapshots.
  - If doing live monitoring, only sees low level hardware events (e.g., a fault)
- Has to figure out how to interpret these snapshots or events
  - May need help from the (untrusted) guest OS!
  - Soln: Use a semantic gap bridging library

Reconstructing data structures

(1) Kernel data structure type definitions

```
struct task_struct {...}
struct list_head {...}
struct siginfo {...}
...
```

(2) Entry-points into the kernel



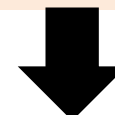
```
ffffe400 init_task
ffffe410 phys_base
ffffe420 loops_per_jiffy
...
```

Data at `0xffffe400`

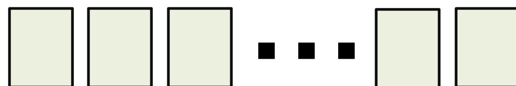
Definition of `task_struct`

```
struct task_struct {
    int state;
    int counter;
    struct task_struct *next;
    ...
}
```

```
00000001
0034ea23
ac3456bc
...
```



init_task.state = 1
init_task.counter = `0x34ea23`
init_task.next = `0xac3456bc`



(3) Snapshot of memory pages

Reconstructing data structures

(1) Kernel data structure type definitions

```
struct task_struct {...}
struct list_head {...}
struct siginfo {...}
...
```

(2) Entry-points into the kernel

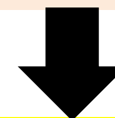
```
ffffe400 init_task
ffffe410 phys_base
ffffe420 loops_per_jiffy
...
```

Definition of `task_struct`

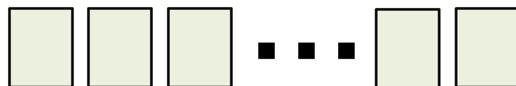
```
struct task_struct {
    int state;
    int counter;
    struct task_struct *next;
    ...
}
```

★ **Data at `0xac3456bc`**

```
00000000
0056ae71
bf6723ae
...
```



init_task.next.state = 0
init_task.next.counter = **`0x56ae71`**
init_task.next.next = **`0xbf6723ae`**



(3) Snapshot of memory pages

Preventing malicious code execution