CS 491/591: Introduction to Computer Security

Confinement

James Hook

(some slides adapted from Bishop)

Plan

- Confinement Problem (Lampson)
- Isolation
  - Virtual Machines
  - Sandboxes
- Covert Channels

The Confinement Problem

- Lampson, "A Note on the Confinement Problem", CACM, 1973.

  This note explores the problem of confining a program during its execution so that it cannot transmit information to any other program except its caller. A set of examples attempts to stake out the boundaries of the problem. Necessary conditions for a solution are stated and informally justified.

Possible Leaks

0. If a service has memory, it can collect data, wait for its owner to call it, then return the data
1. The service may write into a permanent file
2. The service may create a temporary file
3. The service may send a message to a process controlled by its owner [via ipc]
4. More subtly, the information may be encoded in the bill rendered for the service…

Possible Leaks (cont)

5. If the system has interlocks which prevent files from being open for writing and reading at the same time, the service can leak data if it is merely allowed to read files which can be written by the owner.

Leak 5 (cont)

The interlocks allow a file to simulate a shared Boolean variable which one program can set and the other can't

Given a procedure `open (file, error)` which does **`goto`** `error` if the file is already open, the following procedures will perform this simulation:

```
procedure settrue (file);
    begin loop1: open (file, loop1) end;
procedure setfalse (file);
    begin close (file) end;
Boolean procedure value (file);
    begin value : = true;
          open (file, loop2);
          value := false;
          close (file);
      loop2:
    end;
```

Leak 5 (cont)

Using these procedures and three files called data, sendclock, and receiveclock, a service can send a stream of bits to another concurrently running program. Referencing the files as though they were variables of this rather odd kind, then, we can describe the sequence of events for transmitting a single bit:

```
sender:     data : = bit being sent;
            sendclock : = true
receiver:   wait for sendclock = true;
            received bit : = data;
            receive clock : = true;
sender:     wait for receive clock = true;
            sendclock : = false;
receiver:   wait for sendclock = false;
            receiveclock : = false;
sender:     wait for receiveclock = false;
```

Leak 6

6. By varying its ratio of computing to input/output or its paging rate, the service can transmit information which a concurrently running process can receive by observing the performance of the system.
   …

One solution

- Just say no!
- Total isolation: A confined program shall make no calls on any other program
- Impractical

Confinement rule

- Transitivity: If a confined program calls another program which is not trusted, the called program must also be confined.

Classification of Channels:

- Storage
- Legitimate (such as the bill)
- Covert
  - I.e. those not intended for information transfer at all, such as the service program's effect on the system load

- In which category does Lampson place 5?

Mitigation

- Lampson proposes a mitigation strategy for 5
- Confined read makes a copy (this can be done lazily on a conflicting write)

Root Problem:

- Resource sharing enables covert channels
- The more our operating systems and hardware enable efficient resource sharing the greater the risk of covert channels

Lipner's Comments

- 1975 paper discusses how confidentiality models and access control address storage and legitimate channels
- Identifies time as "A difficult problem"
  - "While the storage and legitimate channels of Lampson can be closed with a minimal impact on system efficiency, closing the covert channel seems to impose a direct and unreasonable performance penalty."

Resources

- Lampson, A note on the Confinement Problem, CACM Vol 16, no. 10, October 1973.
  - http://doi.acm.org/10.1145/362375.362389
- Lipner, A Comment on the Confinement Problem, Proceedings of the 5th Symposium on Operating Systems Principles, pp 192 - 196 (Nov. 1975)
  - http://doi.acm.org/10.1145/800213.806537

Timing Channel: Kocher

- CRYPTO '96: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Kocher's Attack

- This computes $x = a^z \bmod n$, where $z = z_0 \ldots z_{k-1}$

```
x := 1; atmp := a;
for i := 0 to k–1 do begin
   if z_i = 1 then
       x := (x * atmp) mod n;
   atmp := (atmp * atmp) mod n;
end
result := x;
```

- Length of run time related to number of 1 bits in z

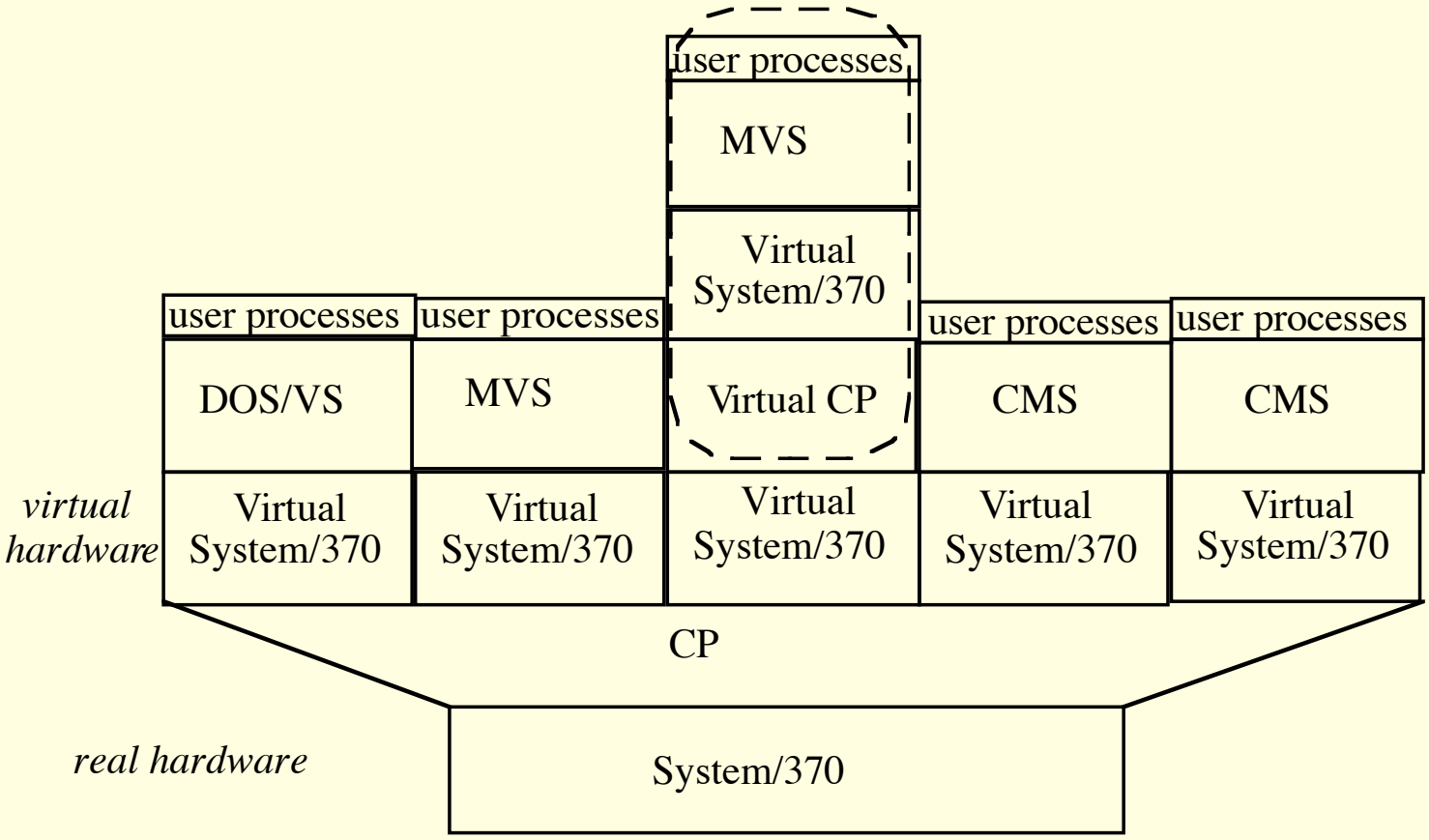
Isolation

- Virtual machines
  - Emulate computer
  - Process cannot access underlying computer system, anything not part of that computer system
- Sandboxing
  - Does not emulate computer
  - Alters interface between computer, process

Virtual Machine (VM)

- A program that simulates hardware of computer system
- *Virtual machine monitor* (VMM) provides VM on which conventional OS can run
  - Each VM is one subject; VMM knows nothing about processes running on each VM
  - VMM mediates all interactions of VM with resources, other VMS
  - Satisfies rule of transitive closure

Example: IBM VM/370



Adapted from Dietel, pp. 606–607

Example: KVM/370

- Security-enhanced version of IBM VM/370 VMM
- Goals
  - Provide virtual machines for users
  - Prevent VMs of different security classes from communicating
- Provides minidisks; some VMs could share some areas of disk
  - Security policy controlled access to shared areas to limit communications to those allowed by policy

DEC VAX VMM

- VMM is security kernel
  - Can run Ultrix OS or VMS OS
- Invoked on trap to execute privileged instruction
  - Only VMM can access hardware directly
  - VM kernel, executive levels both mapped into physical executive level
- VMM subjects: users, VMs
  - Each VM has own disk areas, file systems
  - Each subject, object has multilevel security, integrity labels

Sandbox

- Environment in which actions of process are restricted according to security policy
  - Can add extra security-checking mechanisms to libraries, kernel
    - Program to be executed is not altered
  - Can modify program or process to be executed
    - Similar to debuggers, profilers that add breakpoints
    - Add code to do extra checks (memory access, etc.) as program runs (*software fault isolation*)

Example: Limiting Execution

- Sidewinder
  - Uses type enforcement to confine processes
  - Sandbox built into kernel; site cannot alter it
- Java VM
  - Restricts set of files that applet can access and hosts to which applet can connect
- DTE, type enforcement mechanism for DTEL
  - Kernel modifications enable system administrators to configure sandboxes

Example: Trapping System Calls

- Janus: execution environment
  - Users restrict objects, modes of access
- Two components
  - *Framework* does run-time checking
  - *Modules* determine which accesses allowed
- Configuration file controls modules loaded, constraints to be enforced

Janus Configuration File

```
# basic module
basic
```

— *Load basic module*

```
# define subprocess environment variables
putenv IFS="\t\n " PATH=/sbin:/bin:/usr/bin TZ=PST8PDT
```

— *Define environmental variables for process*

```
# deny access to everything except files under /usr
path deny read,write *
path allow read,write /usr/*
```

— *Deny all file accesses except to those under /usr*

```
# allow subprocess to read files in library directories
# needed for dynamic loading
path allow read /lib/* /usr/lib/* /usr/local/lib/*
```

— *Allow reading of files in these directories (all dynamic load libraries are here)*

```
# needed so child can execute programs
path allow read,exec /sbin/* /bin/* /usr/bin/*
```

— *Allow reading, execution of subprograms in these directories*

Janus Implementation

- System calls to be monitored defined in modules
- On system call, Janus framework invoked
  - Validates system call *with those specific parameters* are allowed
  - If not, sets process environment to indicate call failed
  - If okay, framework gives control back to process; on return, framework invoked to update state
- Example: reading MIME mail
  - Embed "delete file" in Postscript attachment
  - Set Janus to disallow Postscript engine access to files

Additional Resources

- R. Wahbe, S. Lucco, T. Anderson, and S. Graham, Efficient Software-based Fault Isolation, http://www.cs.cornell.edu/home/jgm/cs711sp02/sfi.ps.gz
- Christopher Small, MiSFIT: A Tool for Constructing Safe Extensible C++ Systems, http://www.dogfish.org/chris/papers/misfit/misfit-ieee.ps

Going Deep on Virtualization

- Background (following Bishop Chapter 29)
- Virtualization and Intel architectures

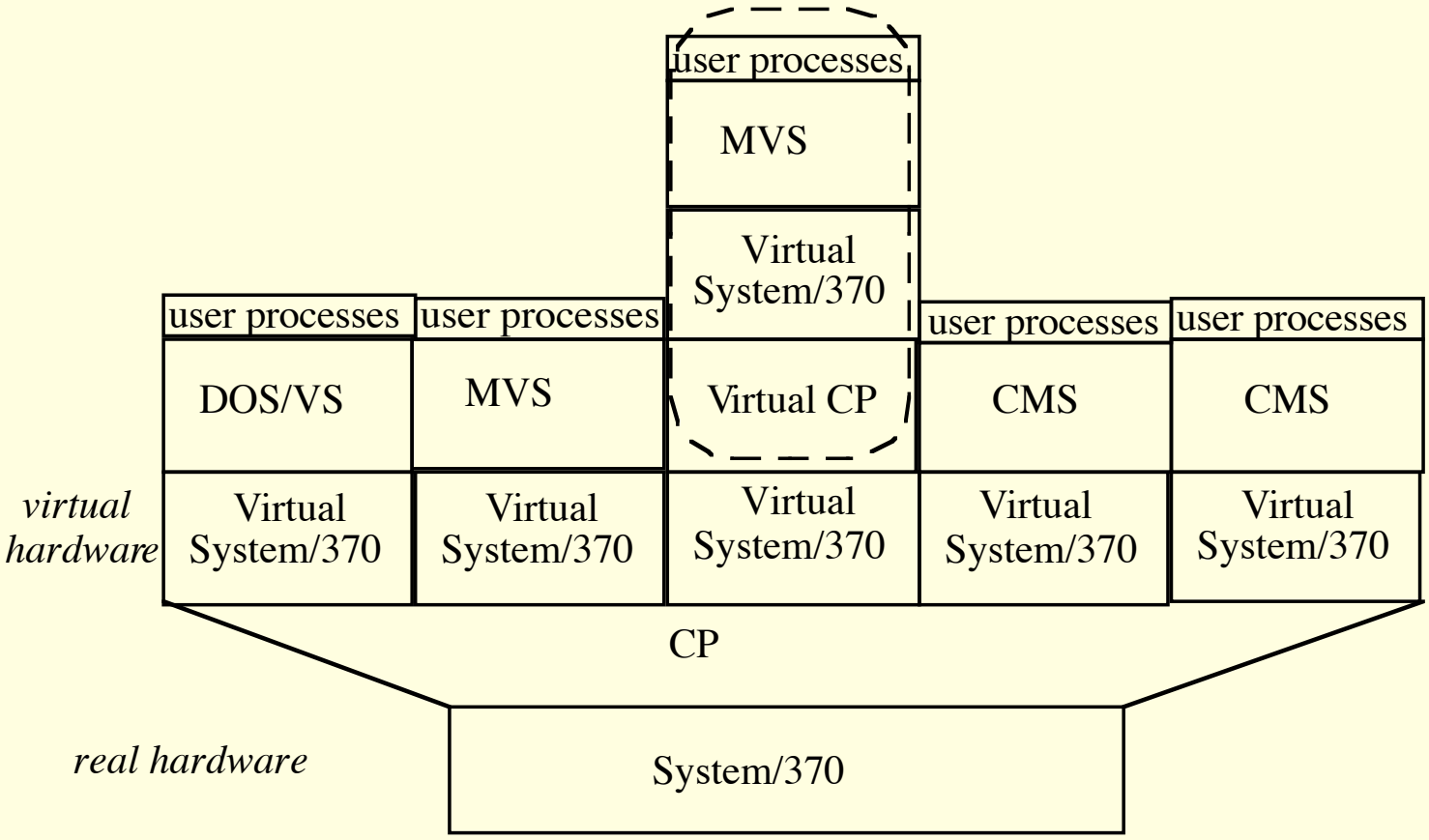
Overview

- Virtual Machine Structure
- Virtual Machine Monitor
  - Privilege
  - Physical Resources
  - Paging

What Is It?

- *Virtual machine monitor* (VMM) virtualizes system resources
  - Runs directly on hardware
  - Provides interface to give each program running on it the illusion that it is the only process on the system and is running directly on hardware
  - Provides illusion of contiguous memory beginning at address 0, a CPU, and secondary storage to *each* program

Example: IBM VM/370



Adapted from Dietel, pp. 606–607

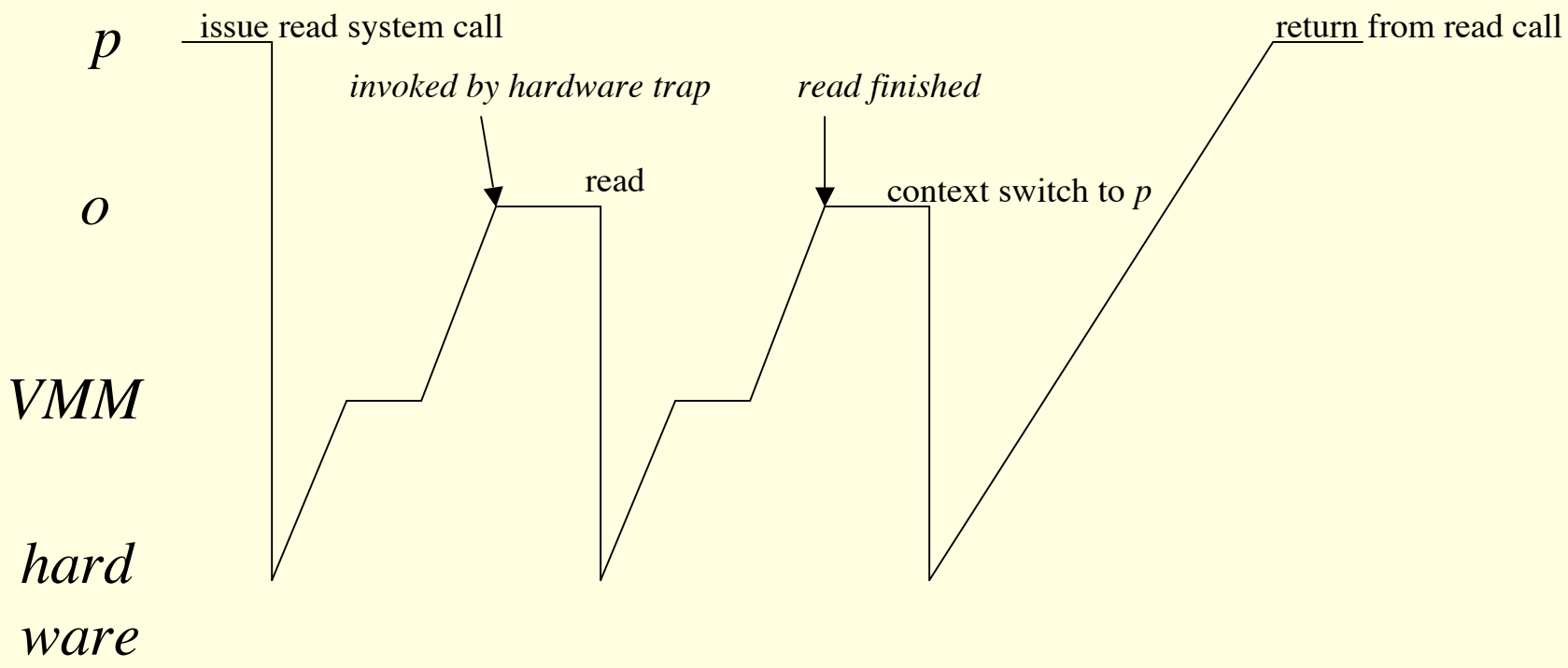
Privileged Instructions

1. VMM running operating system *o*, which is running process *p*
   - *p* tries to read—privileged operation traps to hardware
2. VMM invoked, determines trap occurred in *o*
   - VMM updates state of *o* to make it look like hardware invoked *o* directly, so *o* tries to read, causing trap
3. VMM does read
   - Updates *o* to make it seem like *o* did read
   - Transfers control to *o*

Privileged Instructions

4. *o* tries to switch context to *p*, causing trap
5. VMM updates virtual machine of *o* to make it appear *o* did context switch successfully
   - Transfers control to *o*, which (as *o* apparently did a context switch to *p*) has the effect of returning control to *p*

Privileged Instructions



Privilege and VMs

- *Sensitive instruction* discloses or alters state of processor privilege
- *Sensitive data structure* contains information about state of processor privilege

When Is VM Possible?

- Can virtualize an architecture when:
  1. All sensitive instructions cause traps when executed by processes at lower levels of privilege
  2. All references to sensitive data structures cause traps when executed by processes at lower levels of privilege

Example: VAX System

- 4 levels of privilege (user, supervisor, executive, kernel)
  - CHMK changes privilege to kernel level; sensitive instruction
    - Causes trap *except* when executed in kernel mode; meets rule 1
  - Page tables have copy of Processor Status Longword (PSL), containing privilege level; sensitive data structure
    - If user level processes prevented from altering page tables, trying to do so will cause a trap; this meets rule 2

Multiple Levels of Privilege

- Hardware supports *n* levels of privilege
  - VM must also support *n* levels
  - VM monitor runs at highest level, so *n*–1 levels of privilege left!
- Solution: virtualize levels of privilege
  - Called *ring compression*

Example: VAX VMM System

- VMM at kernel level
- VMM maps virtual kernel and executive level to (real) executive mode
  - Called *VM kernel level*, *VM executive level*
  - Virtual machine bit added to PSL
    - If set, current process running on VM
  - Special register, VMPSL, records PSL of currently running VM
  - All sensitive instructions that *could* reveal level of privilege get this information from VMPSL or trap to the VMM, which then emulates the instruction

Alternate Approach

- Divide users into different classes
- Control access to system by limiting access of each class

Example: IBM VM/370

- Each control program command associated with user privilege classes
  - "G" (general user) class can start a VM
  - "A" (primary system operator) class can control accounting, VM availability, other system resources
  - "Any" class can gain or surrender access to VM

Physical Resources and VMs

- Distributes resources among VMs as appropriate
  - Each VM appears to have reduced amount of resources from real system
  - Example: VMM to create 10 VMs means real disk split into 10 minidisks
    - Minidisks may have different sizes
    - VMM does mapping between minidisk addresses, real disk addresses

Example: Disk I/O

- VM's OS tries to write to disk
  - I/O instruction privileged, traps to VMM
- VMM checks request, services it
  - Translates addresses involved
  - Verifies I/O references disk space allocated to that VM
  - Services request
- VMM returns control to VM when appropriate
  - If I/O synchronous, when service complete
  - If I/O asynchronous, when service begun

Paging and VMs

- Like ordinary disk I/O, but 2 problems
  - Some pages may be available only at highest level of privilege
    - VM must remap level of privilege of these pages
  - Performance issues
    - VMM paging its own pages is transparent to VMs
    - VM paging is handled by VMM; if VM's OS does lots of paging, this may introduce significant delays

Example: VAX/VMS

- On VAX/VMS, only kernel level processes can read some pages
  - What happens if process at VM kernel level needs to read such a page?
    - Fails, as VM kernel level is at real executive level
  - VMM reduces level of page to executive, then it works
    - Note: security risk!
      - In practice, OK, as VMS allows executive level processes to change to kernel level

Example: IBM VM/370

- Supports several different operating systems
  - OS/MFT, OS/MVT designed to access disk storage
    - If jobs being run under those systems depend on timings, delay caused by VM may affect success of job
  - If system supports virtual paging (like MVS), either MVS or VMM may cause paging
    - The VMM paging may introduce overhead (delays) that cause programs to fail that would not were the programs run directly on the hardware

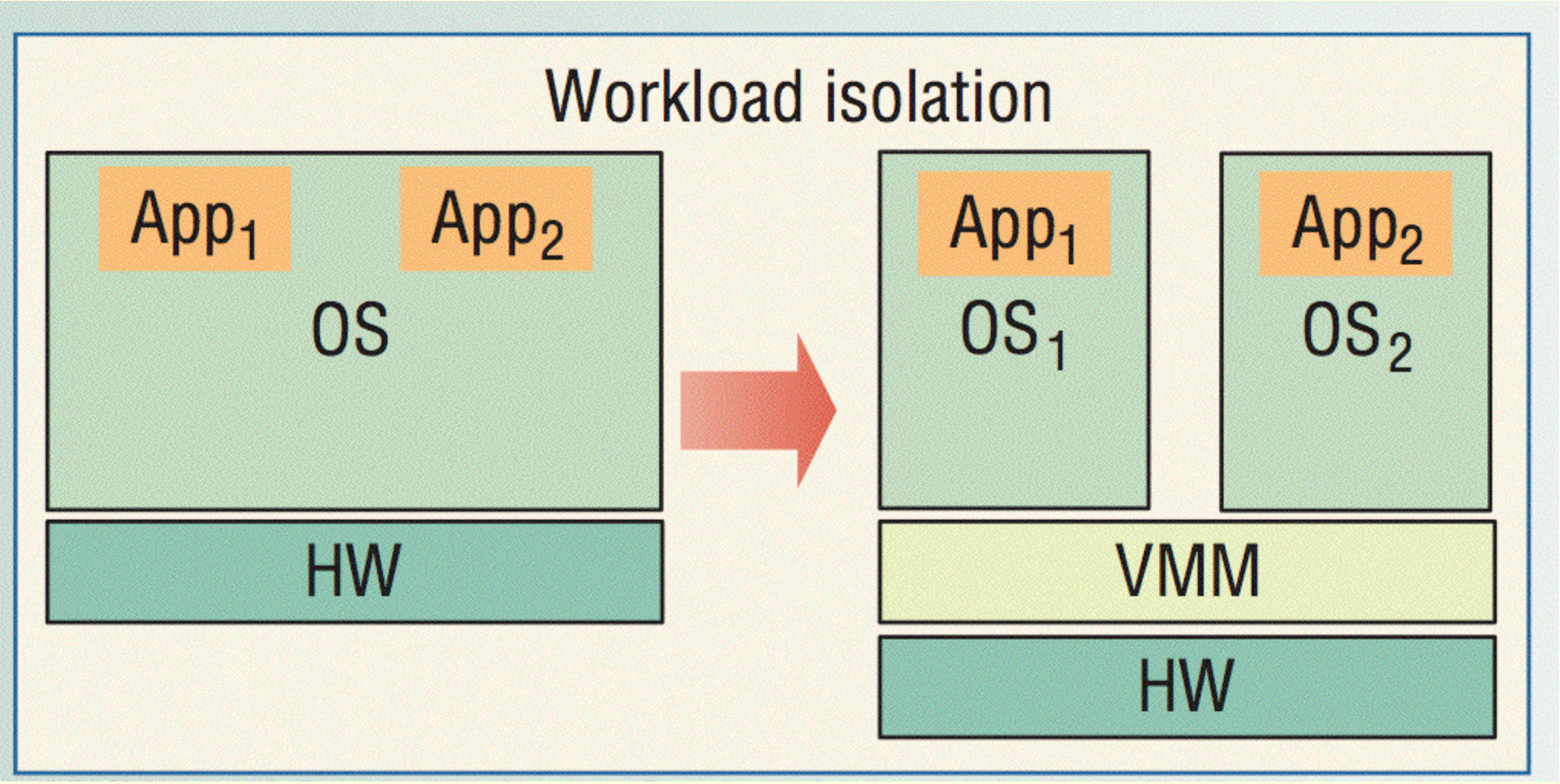
Virtualization Returns

- Intel's Vanderpool architecture brings Virtual Machines back to the mainstream
- Intel Virtualization Paper
  - ftp://download.intel.com/technology/computing/vptech/vt-ieee-computer-final.pdf

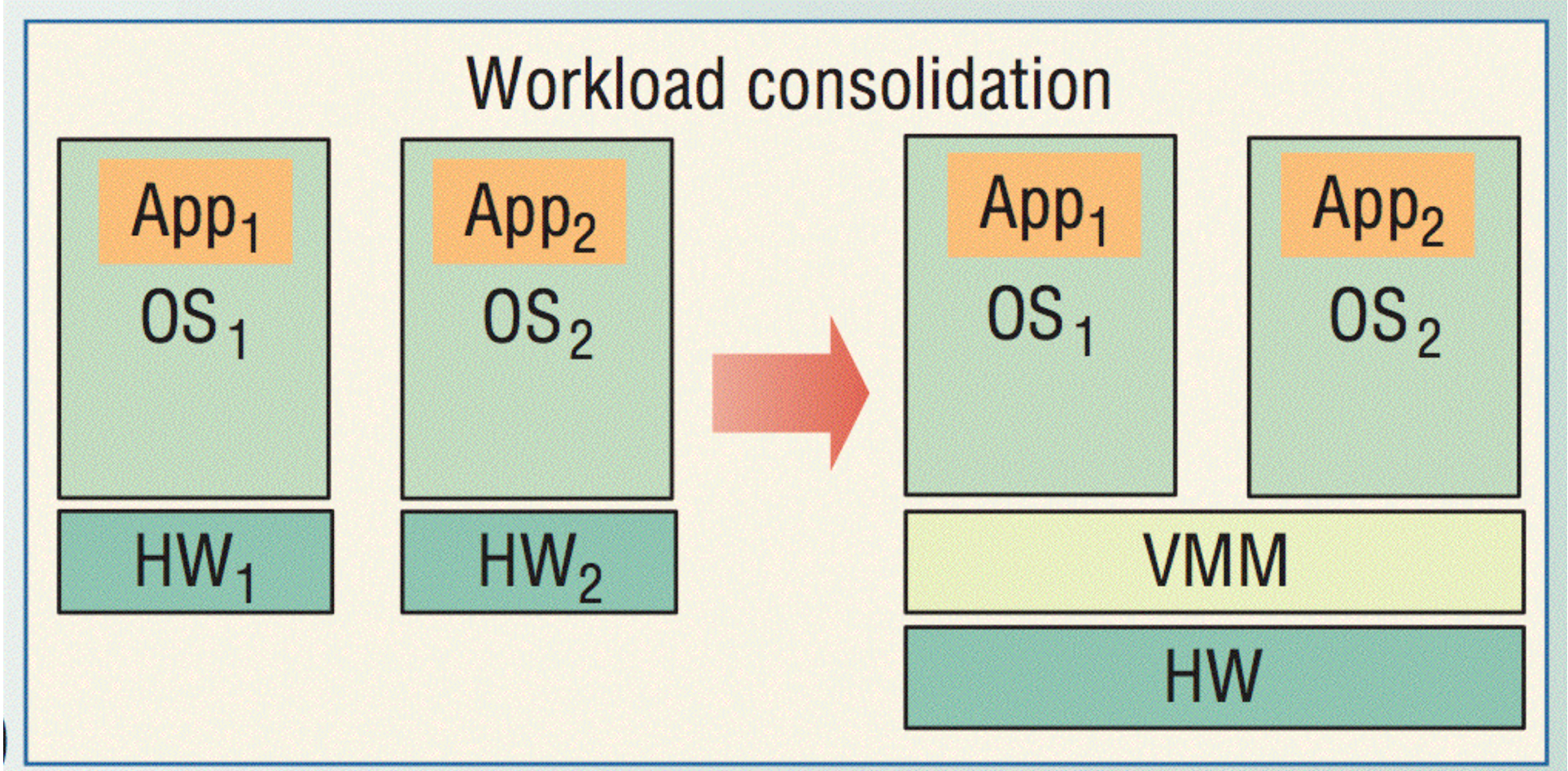
Applications of Virtualization

- Workload isolation
- Workload consolidation
- Workload migration

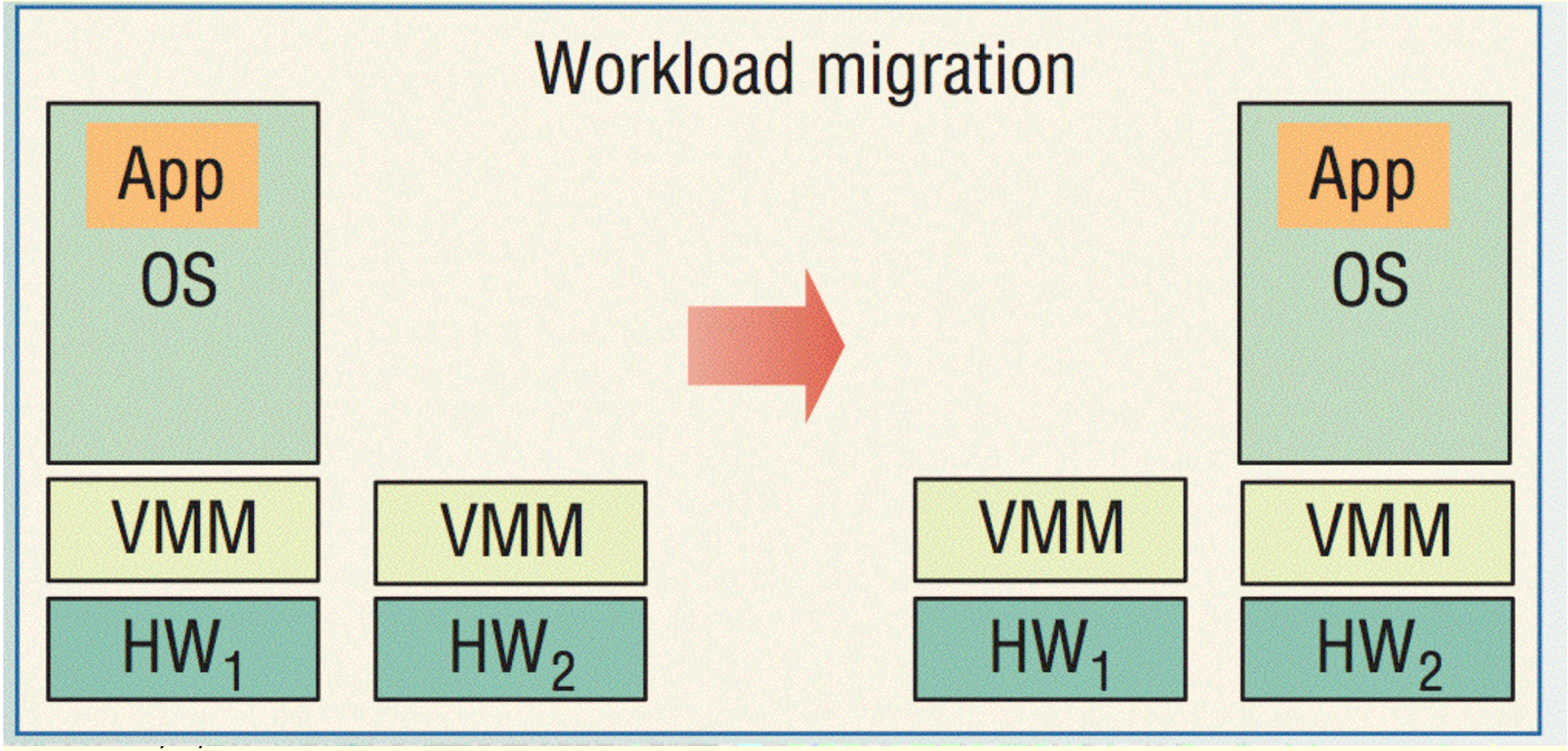
Isolation



Consolidation



Migration


Workload migration
App
OS
VMM
HW1
VMM
HW2
VMM
HW1
App
OS
VMM
HW2

Virtualizing Intel architectures

- As is, Intel architectures do not meet the two requirements:
  - Nonfaulting access to privileged state
    - IA-32 has registers that describe and manipulate the “global descriptor table”
    - These registers can only be set in ring 0
    - They can be queried in any ring without generating a fault
  - This violates rule 2 (all references to sensitive data traps)
- Software products to virtualize Intel hardware had to get around this.
  - Vmware and Virtual PC dynamically rewrite binary code!
  - Xen requires source changes (paravirtualization)

Intel solutions

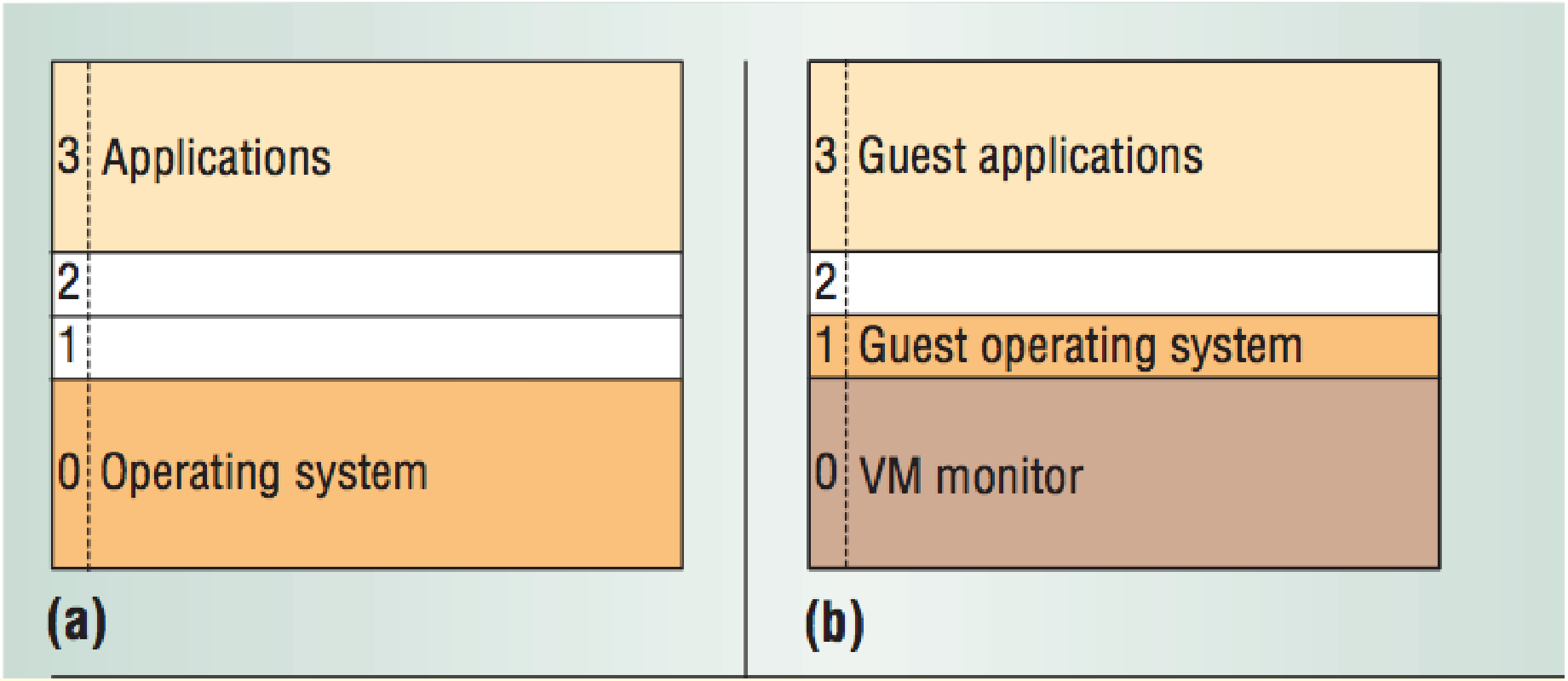
- VT-x, virtualization for IA-32
- VT-i, virtualization for Itanium

- Changed architecture to meet the criteria

Ring aliasing and ring compression

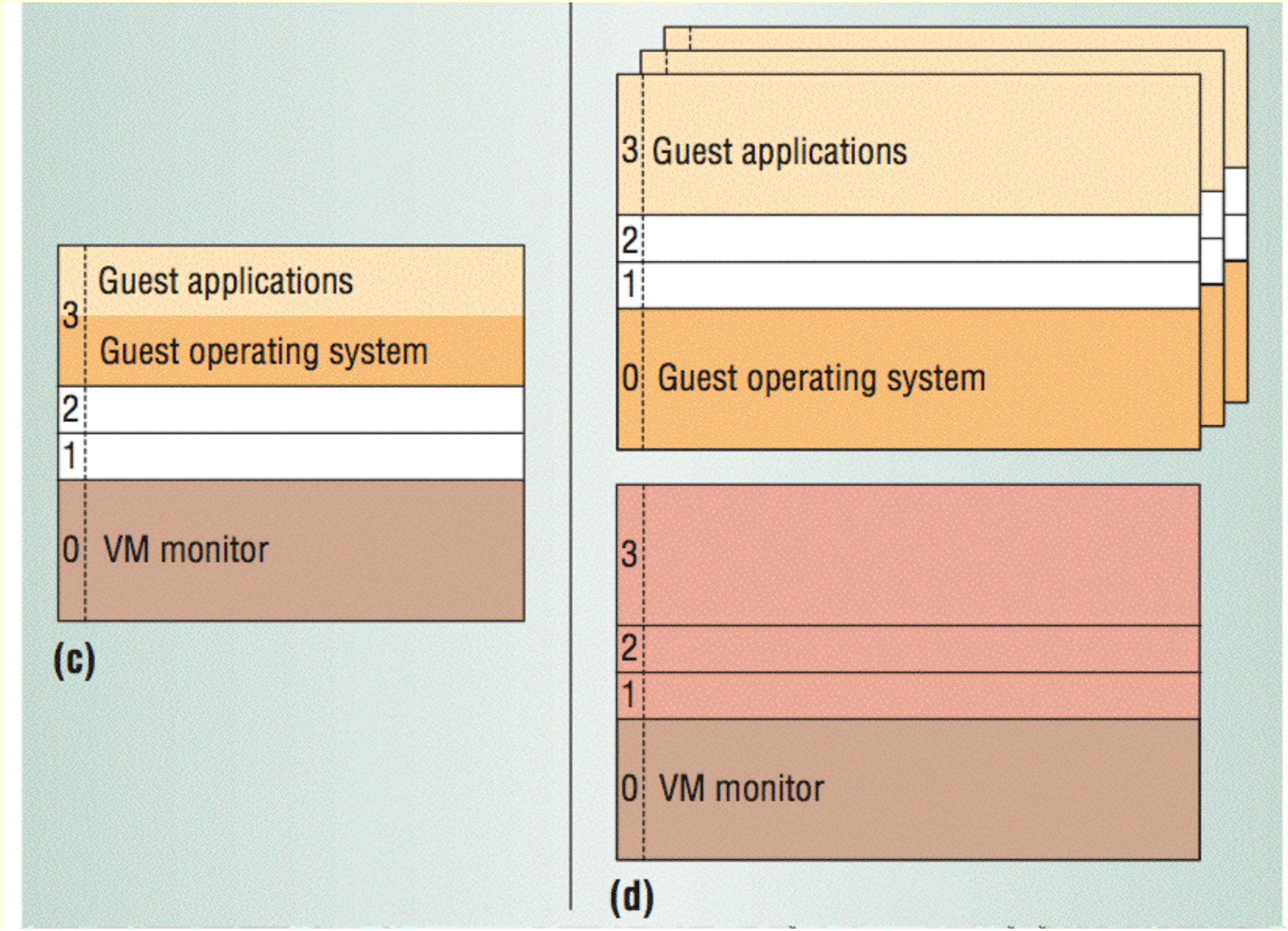
- Solution is to allow guest to run at intended privilege level by augmenting privilege levels.
- See Figure 2(d).

Nonvirtuallized and 0/1/3



- (a) is typical of x86 operating systems
- (b) and (c) give two strategies for virtualization in software

0/3/3 and VT-x


3
Guest applications
Guest operating system
2
1
0
VM monitor
(c)
3
Guest applications
2
1
0
Guest operating system
3
2
1
0
VM monitor
(d)

Nonfaulting access to privileged state

- Two kinds of changes
  - Make access fault to the VM
  - Allow nonfaulting access, but to state under the control of the VMM

- Intel Virtualization Paper
  - ftp://download.intel.com/technology/computing/vptech/vt-ieee-computer-final.pdf

Dark Side

- Malware and Virtual Machines
  - SubVirt: Implementing malware with virtual machines,
  - King, Chen, Wang, Verbowski, Wang, Lorch
  - Describes the construction of a "virtual-machine based rootkit" and potential defenses.