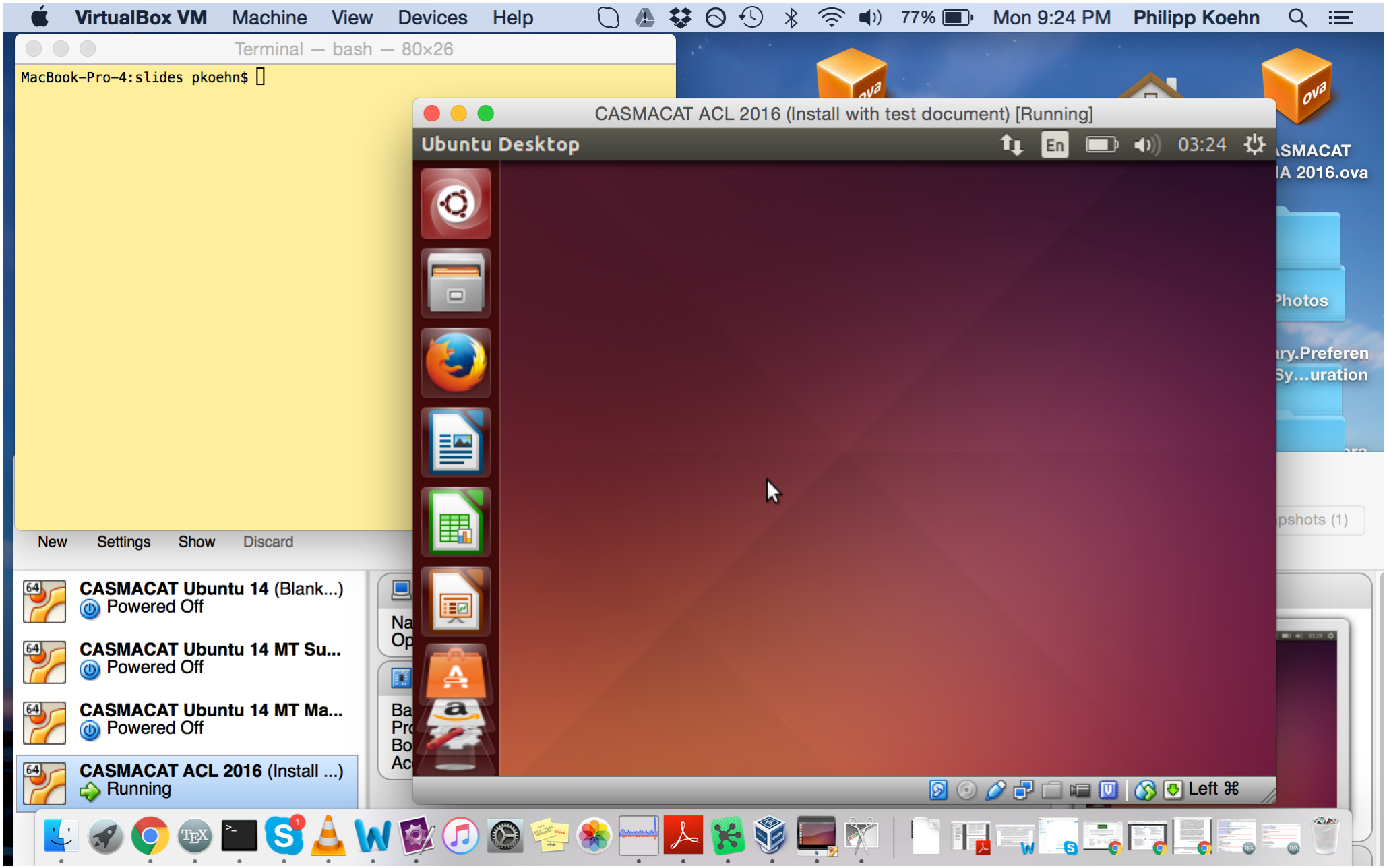
Virtual Machines

Philipp Koehn

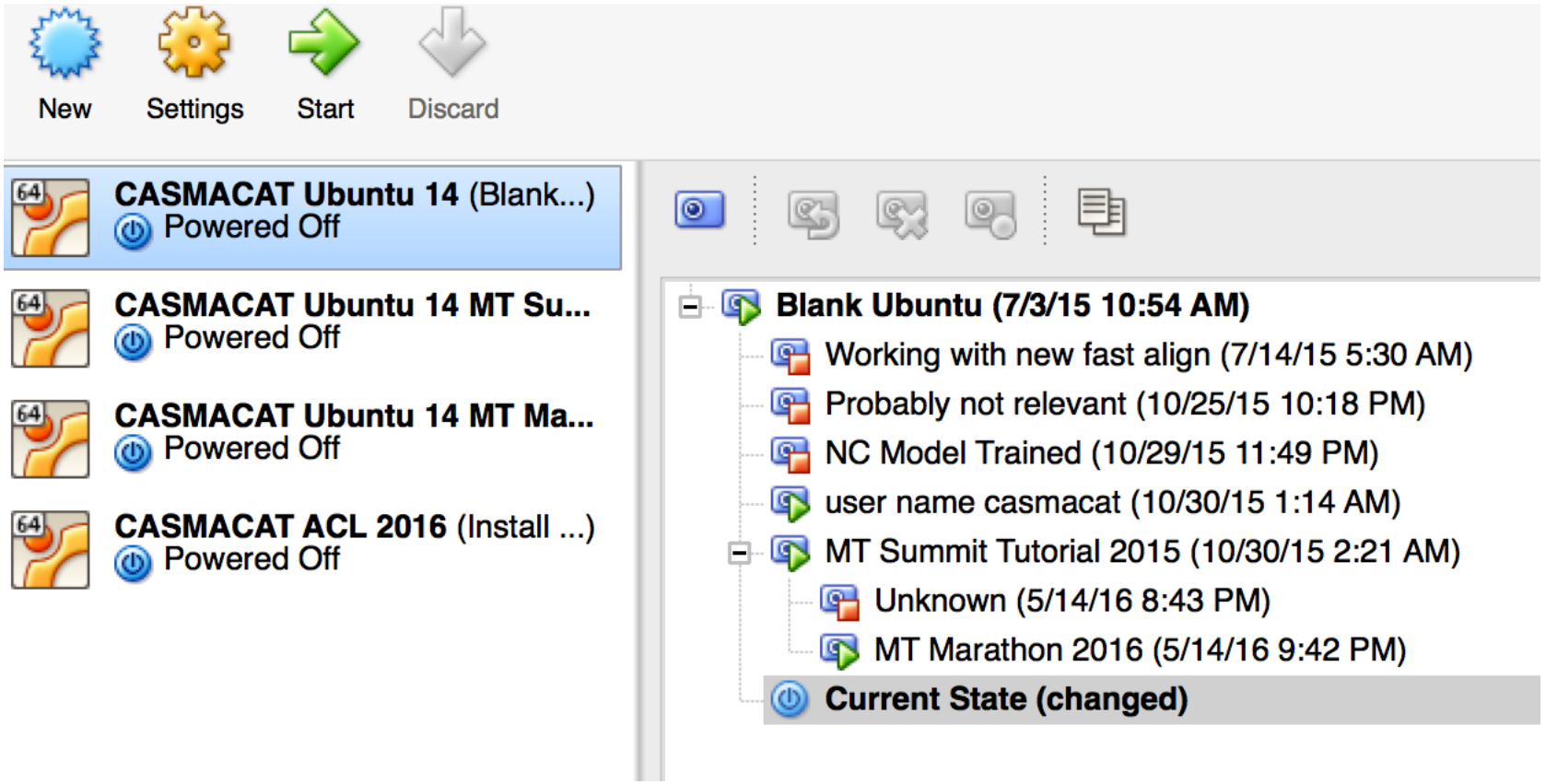
30 April 2018

Basic Idea

- Run multiple instances of full operating systems on a machine
- Example: run Windows and Linux on a Mac
- Not to be confused with Java Virtual Machines


VirtualBox VM
Machine
View
Devices
Help
77%
Mon 9:24 PM
Philipp Koehn
Terminal — bash — 80×26
MacBook-Pro-4:slides pkoehn$
CASMACAT ACL 2016 (Install with test document) [Running]
Ubuntu Desktop
En
03:24
New
Settings
Show
Discard
CASMACAT Ubuntu 14 (Blank...)
Powered Off
CASMACAT Ubuntu 14 MT Su...
Powered Off
CASMACAT Ubuntu 14 MT Ma...
Powered Off
CASMACAT ACL 2016 (Install ...)
Running
Left ⌘

Snapshots



- Freeze copy of a virtual machine
- Copy of file system and memory

Migration

- Migration: move a VM to another host
  (maybe because of spike of VM usage overloads current machine)

- Steps
  - take snapshot (fast)
  - copy all pages of snapshot (not so fast)
  - copy modified pages (fast)
  - freeze virtual machine and copy VM memory

- Very fast, fractions of a second

Why?

- Better resource utilization:
  sharing of a single computer among several users
- Isolation and security in clouds
- Security limitations of standard operating systems
- Faster processors make overhead acceptable

History

- Virtual machines popular in mainframes in 1970s
- Not on "personal computer" Intel x86 for a long time
- First x86 virtualization: VMWare 1999
- Intel and AMD added hardware support 2005/2006
- Used in cloud computing (e.g., Amazon web services)

basics

Virtual Machine Monitor

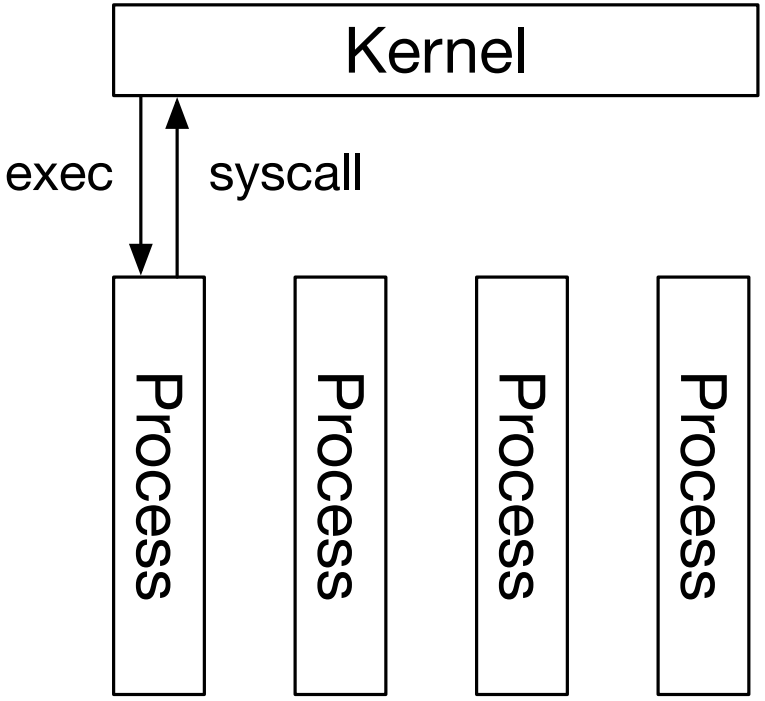
- Host machine runs a regular operating system

- Virtual machine monitor (VMM)
  - runs as a process of the operating system
  - has privileged access to CPU

- VMM runs other operating systems (guest machine)
  - manages their access to hardware
  - intercepts exceptions and interrupts

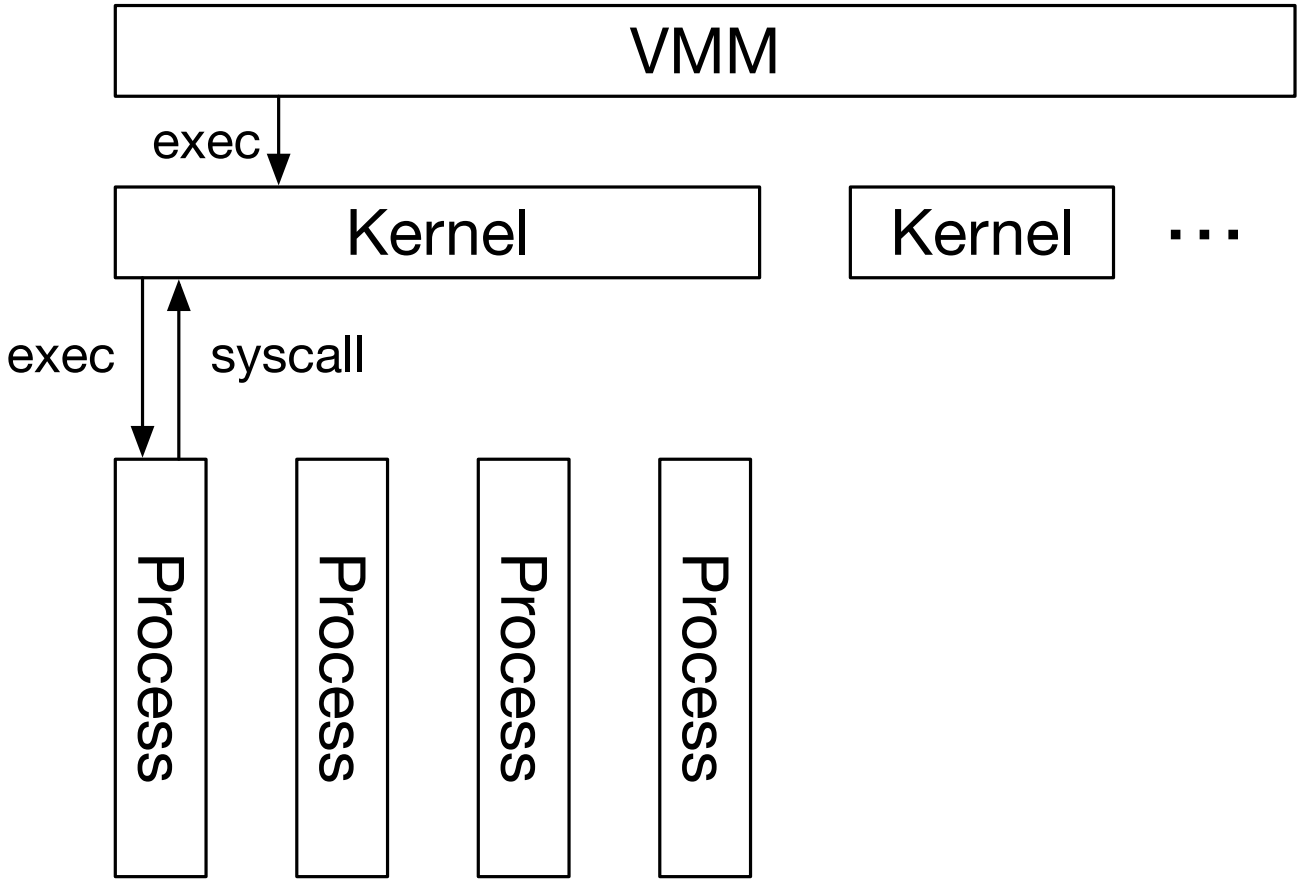
Virtual Machine Monitor

Normal OS



Virtual Machine Monitor

Virtual Machine



Basic Functions of Operating System

- User mode
  - process runs in own virtual memory
  - makes systems calls to kernel

- Kernel mode
  - manages processes
  - handles interrupts and exceptions
    e.g., page faults

- Hardware supports this with "privileged" mode for instructions
  e.g., allow access to physical memory

User Processes

- Run already in "virtual mode"
- Memory access is channeled through virtual memory
- Device interactions are handled by kernel via system calls

$\Rightarrow$ Very little overhead when running inside virtual machine
(unless very I/O intensive)

Interrupt Handling

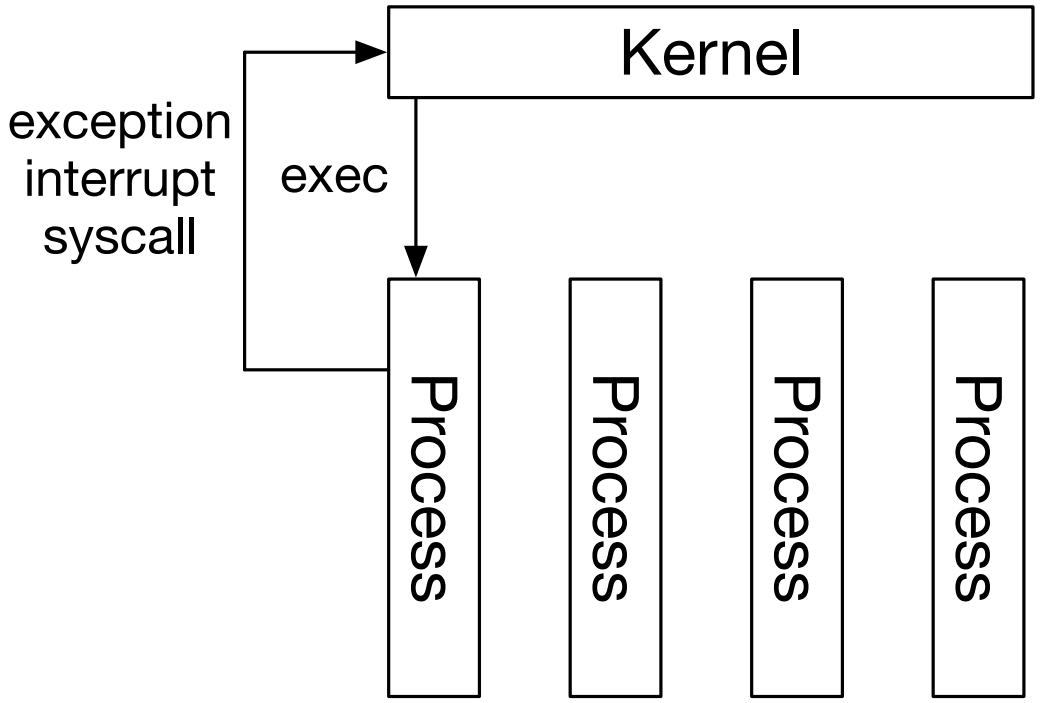
- VMM controls access to
  - privileged CPU state
  - input/output devices
  - exceptions
  - interrupts

- "Trap and emulate"

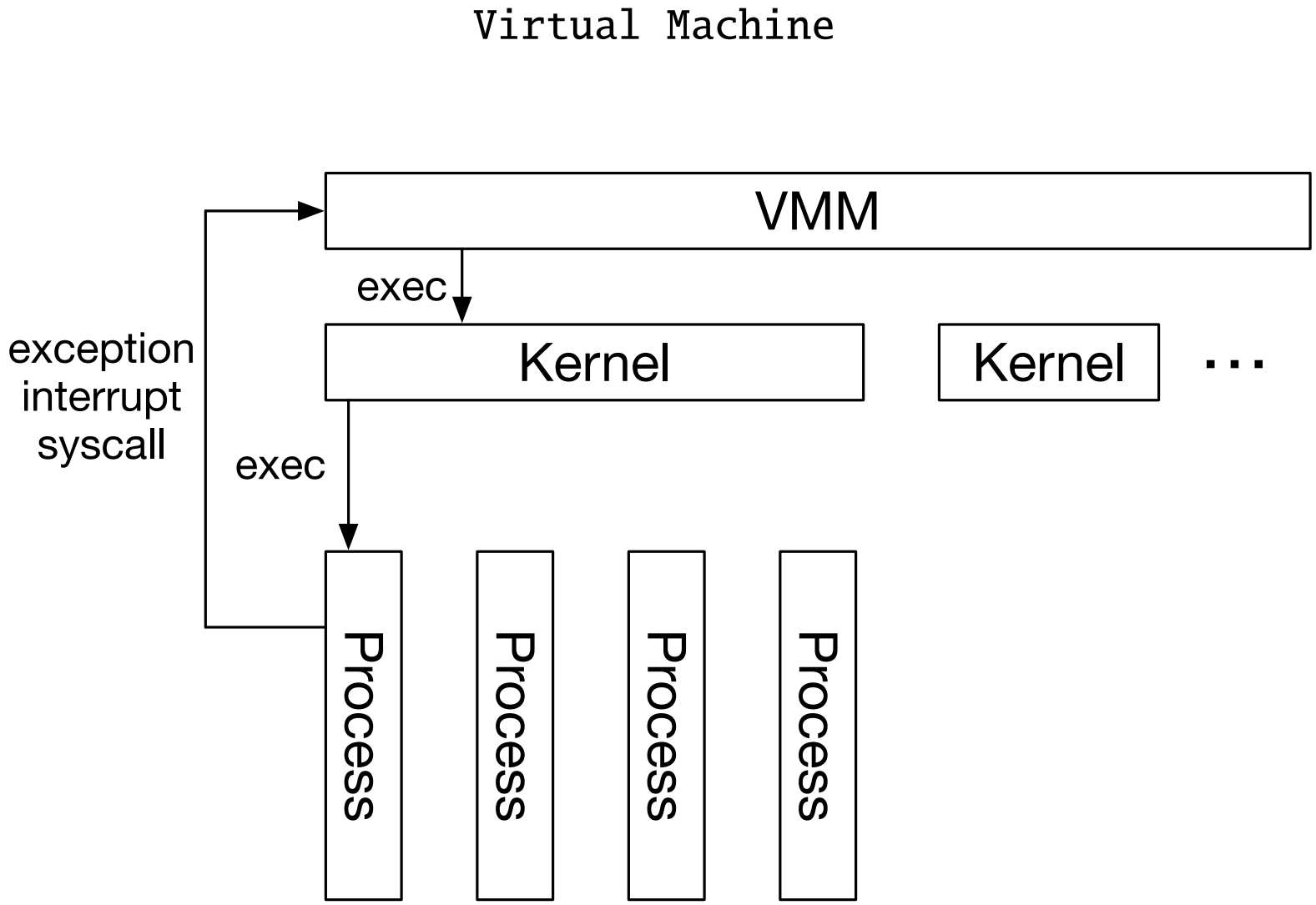
  VMM catches exceptions and directs them to the right guest

Traps

Normal OS



Virtual Machine Monitor Catches Traps



emulation

Emulation

- Binary translation
- Shaddowing
- Device emulation

Binary Translation

- Some instructions require supervisor mode
  - access to physical memory
  - handling interrupt flags
- Raw kernel code instructions need to be translated
  i.e., rewritten into user mode instructions
- This is tricky...

Shadowing

- Guest kernel data structures need to be duplicated by VMM

- Example: page tables of virtual memory
  - VMM maintains copy of page tables
  - traps access attenpts
  - emulating them instead in software

- VMM tracks changes by guest kernel

Device Emulation

- Kernel accesses devices directly, e.g.,
  - network adapter
  - disk
  - keyboard
  - video/audio i/o
- VMM talks directly to these
- Guest OS interactions with hardware have to go through VMM
- Guest OS has access only to generic devices

Hardware Support

- Intel and AMD implement virtualization support for x86
- Direct execution model
  - new execution mode: guest mode
    → direct execution of guest OS code
    incl. privileged instructions
  - virtual machine control block (VMCB)
    → controls what operations trap
    records info to handle traps in VMM
- Steps
  - new instruction "vmrun" enters guest mode, runs VM code
  - when VM traps, CPU executes new "exit" instruction
  - enters VMM, which emulates operation

shadow page tables

Virtualizing Memory

- OS assumes it has full control over memory
  - managing it: OS assumes it owns it all
  - mapping it: OS assumes it can map to any physical page

- VMM partitions memory among VMs
  - VMM needs to assign hardware pages to VMs
  - VMM needs to control mappings for isolation

  → OS can only map to a hardware page given to it by the VMM

Additional Abstraction

- Three abstractions of memory

  machine: actual hardware memory, e.g., 16 GB of DRAM

  physical: abstraction of hardware memory managed by OS

  - VMM allocates 2 GB to a VM

  $\rightarrow$ OS thinks the computer has 2 GB of contiguous physical memory

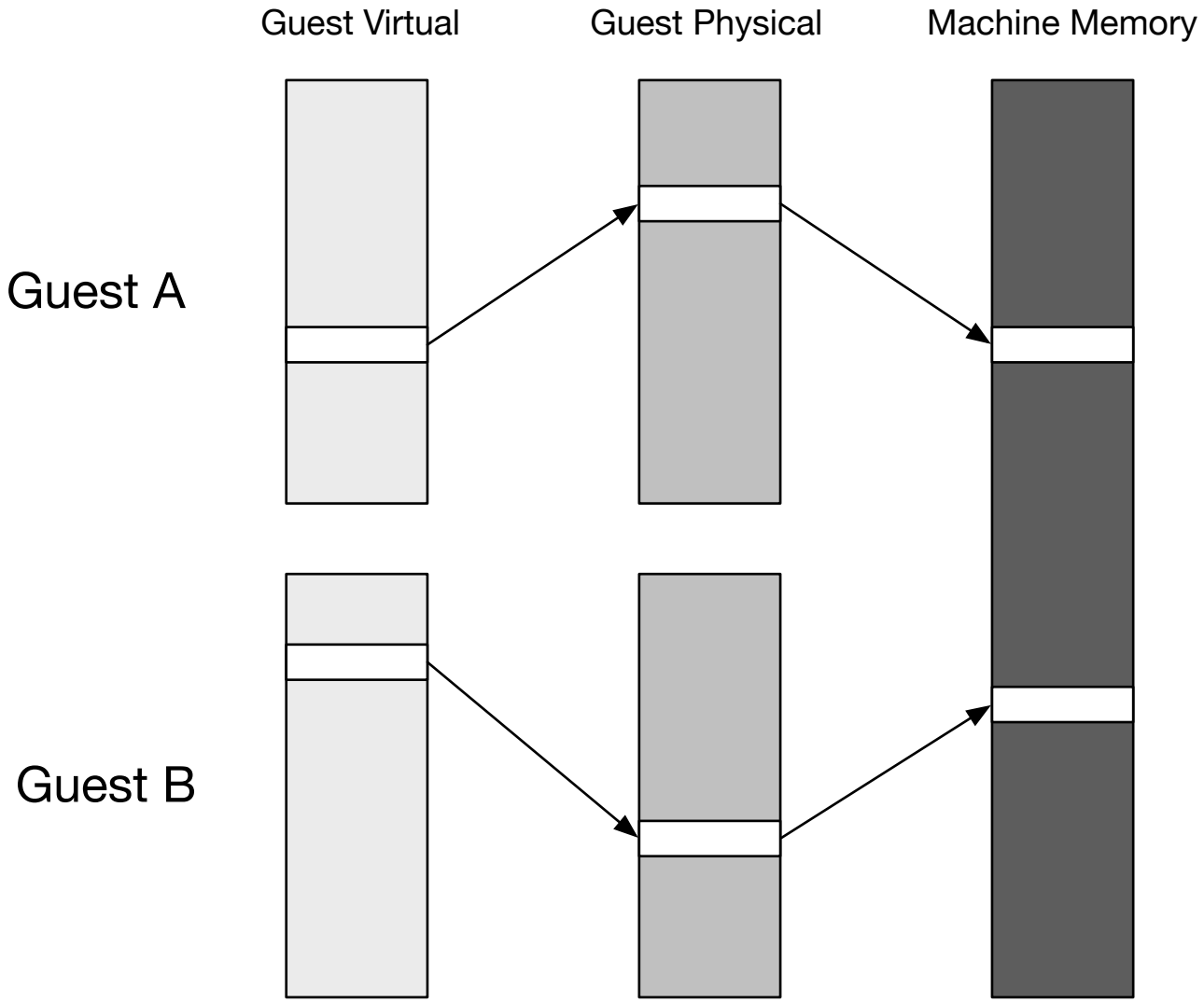
  - note: underlying machine memory may be discontiguous

  virtual: virtual address spaces of process (48 bit $\rightarrow$ 256TB)

- Guest OS creates and manages page tables

  but: these page tables are not used by the MMU hardware

Address Translation



Shadow Page Tables

- VMM manages page tables that map virtual pages to machine pages ("shadow page tables")

- These tables are loaded into the MMU on a context switch

- VMM needs to keep its V→M tables consistent with changes made by OS to its V→P tables
  - VMM maps OS page tables as read only
  - when OS writes to page tables, trap to VMM
  - VMM applies write to shadow table and OS table, returns

Hardware Support

- Intel extended page tables (EPT), AMD nested page tables (NPT)
- Original page tables map virtual to (guest) physical pages
  - Managed by OS in VM, backwards-compatible
  - No need to trap to VMM when OS updates its page tables
- New tables map physical to machine pages: Managed by VMM
- Translation lookup buffer (TLB)
  - tagged TLB w/ virtual process identifiers (VPIDs)
  - tag VMs with VPID, no need to flush TLB on VM/VMM switch

containers

Deploying Services

- Often the goal is to deploy complex software applications
- Many dependencies: specific versions of libraries
- Example: "web service" answers HTTP request to fulfill complex tasks
- One solution: virtual machine
  - package all the software into a virtual machine
  - deployment: run virtual machine
  - but: relatively large overhead (runs entire operating system)
- Light-weight solution: containers

Docker Containers

- One (host) operating system
- Containers include application and all dependencies
- But share the kernel with other containers
- Each containers runs as isolated process in user space
- Initial release of open source software in 2013

Containers vs. Virtual Machine