Project 2

- Many clouds are used for big data processing. Machine Learning is an example use case.
- Machine Learning: train a model and then use it for inference.
- Before training, ETL (Extract Transform Load) is needed and it often consumes a large fraction of human time:
  - Data cleaning: extract the useful content, transform it to an efficient format
  - Data exploration: compute the main statistical characteristics
- The aforementioned programming systems are meant to make data processing in the cloud easy.
- We care about the performance of those systems.

Project 2 Data Engineering On the Cloud

- Explore workflow for Big Data Machine Learning using a framework
- Hands-on experience with second generation MapReduce: Spark
  - Develop distributed applications using Spark.
  - Practice Spark performance diagnosis (and optimization).
- Part 1 – ETL (Extract Transform and Load) processing with Spark
  - Releasing soon.
- Part 2 & 3 – develop an iterative ML training program and improve its performance and scalability

Project 2.1 ETL Processing on AWS using Apache Spark

- The task:
  - Given a large set of crawled webpages stored in HDFS.
  - Preprocess them to a format that can be used for training ML models.
  - Compute basic statistics about the corpus.
  - You don't need to know ML models nor how to train them for part 1.
  - This is a good use case for MapReduce & Spark (big data, non-iterative).
- Your program will be tested against datasets of various size and on various number of EC2 m4.xlarge instances.
- Grading is based on both correctness and performance.
  - Your program needs to complete within the required time otherwise you lose points.

Project 2.1 Why ETL?

- Topic modeling:
  - Classify documents based on their topic (sports, political, etc).
- Examples (detailed specification in the handout):
  - Webpages might not be in English (not ASCII).
  - Webpages might contain invalid words (misspelling, etc).
  - There are words that are not helpful for identifying topics – stop words: you, me, that, the, etc.
  - Topic modeling uses the bag-of-words model to represent documents, which is typically more compact.

Project 2.1 Dataset

- Common Crawl: http://commoncrawl.org/
  - An open repository of web crawl data
  - It's regularly updated. We'll use the December 2016 version.
- It's in various formats. You will process the WET format.
- WET files are gzip-compressed, ~170MB each.
- Your program will be tested against up to 1000 WET files (about 420 GB total uncompressed).
- Start with a small data size, make sure your program is correct before improving its performance.

Project 2.1 Dataset – WET format

```
WARC/1.0
WARC-Type: conversion
WARC-Target-URI: http://news.bbc.co.uk/2/hi/africa/3414345.stm
WARC-Date: 2014-08-02T09:52:13Z
WARC-Record-ID:
WARC-Refers-To:
WARC-Block-Digest: sha1:JROHLCS5SKMBR6XY46WXREW7RXM64EJC
Content-Type: text/plain
Content-Length: 6724
```

Header

```
BBC NEWS | Africa | Namibia braces for Nujoma exit
...
President Sam Nujoma works in very pleasant surroundings in the small but beautif
ul old State House...
```

Payload

- In part 1 you will parse & transform the dataset and compute stats
- Informally, data is sequences of records, each with a header and payload corresponding to the response to one HTTP request, with HTML already removed

Project 2.1 Apache Spark

- You will use its python API (PySpark).
- We provide you with an AMI and tools to launch Spark (2.1.0) + HDFS clusters on EC2.
  - Do not use AWS EMR

Project 2.1 Apache Spark

- Spark programming is declarative
  - Programs that are logically equivalent may have vastly different performance.
- To complete this project, you need to understand how Spark operates from project handout and learn how to perform basic performance diagnosis.
  - (Optional) recitation on Friday to discuss handouts & answer.
  - Slides will be available on theproject.zone beforehand.
  - Don’t wait until the recitation to start.

Project 2.1 Apache Spark First Steps

- Quick start: http://spark.apache.org/docs/latest/quick-start.html
- Programming guide: http://spark.apache.org/docs/latest/programming-guide.html
- Submitting a job to Spark: http://spark.apache.org/docs/latest/submitting-applications.html
- There will be a few important configuration parameters: http://spark.apache.org/docs/latest/configuration.html
- Spark monitoring web UI: http://spark.apache.org/docs/latest/monitoring.html

ENCAPSULATION

(15-719 – Advanced Cloud Computing)

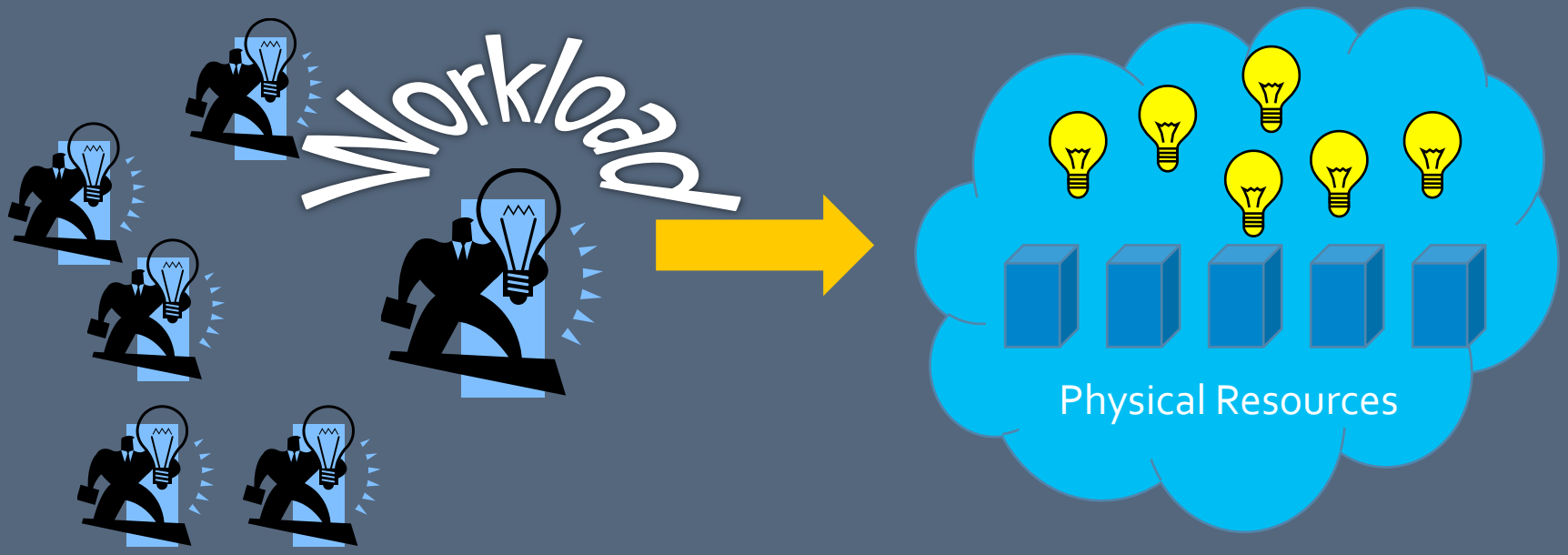
Dr. Michael Kozuch

Intel Labs, Pittsburgh

MOTIVATION

CLOUD 101

Cloud workloads may have many different users and applications



Cloud *providers* want users to share a unified infrastructure:

- To ease infrastructure management
- To multiplex users/applications
- To provide a common programming interface

WHAT DO CLOUD USERS WANT?

Instance Properties

- Security isolation
- Performance isolation
- Portability
- Software flexibility

Infrastructure Properties

- Reliability
- Scalability
- Ease of management
- Tool/component availability

Solution:
workloads encapsulated
on shared infrastructure

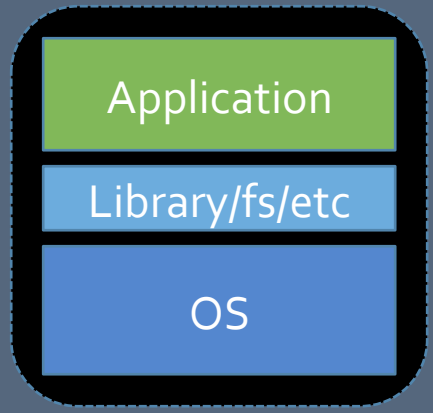
ENCAPSULATION COMPONENTS

- Specification of an "image"
  - What will run within the container
  - E.g., application binaries, data files, filesystem images, disk images
- Specification of resources needed
  - Hardware resources, such as CPU, memory, devices
  - Networking configuration
  - Other dependencies

ENCAPSULATION OPTIONS

OPTION #1: BARE METAL

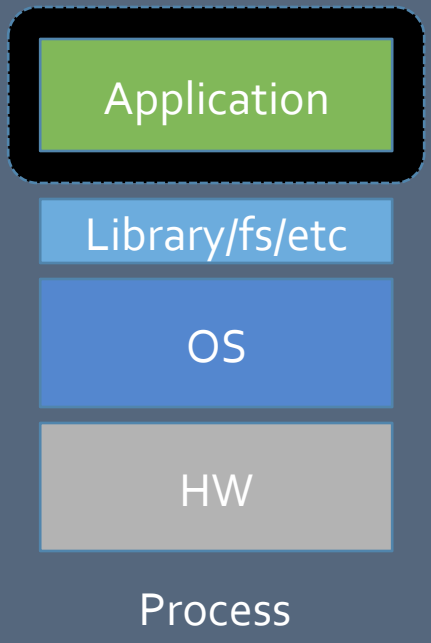
- **Bare Metal**: Give users entire machines
  - Pro: Good isolation, Software freedom, Best performance
  - Con: Limits allocation granularity, Software management tricky (drivers, debugging OS)



Bare Metal

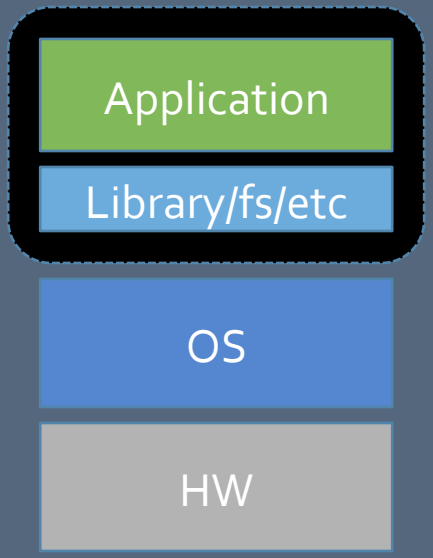
OPTION #2: PROCESS

- **Process**: Users are allocated traditional OS processes
  - Pro: Well-understood, Good performance, Debugging "easy"
  - Con: Performance isolation poor, Security questionable, Software freedom poor



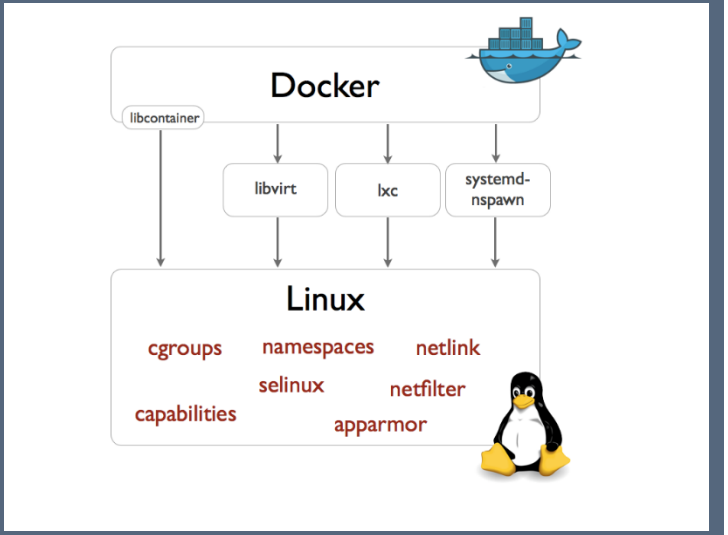
OPTION #3: CONTAINERS

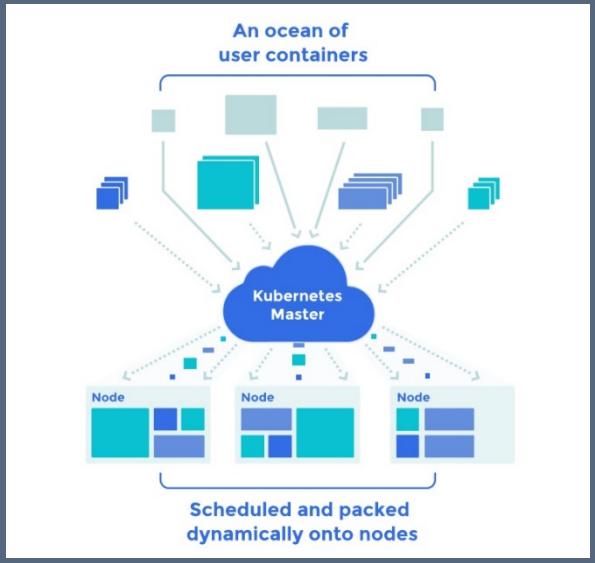
- **Containers**: Traditional OS hosts process containers, where each container looks like an "empty" OS
  - Pro: Decent software freedom, Good performance
  - Con: Possible security problems



CONTAINERS

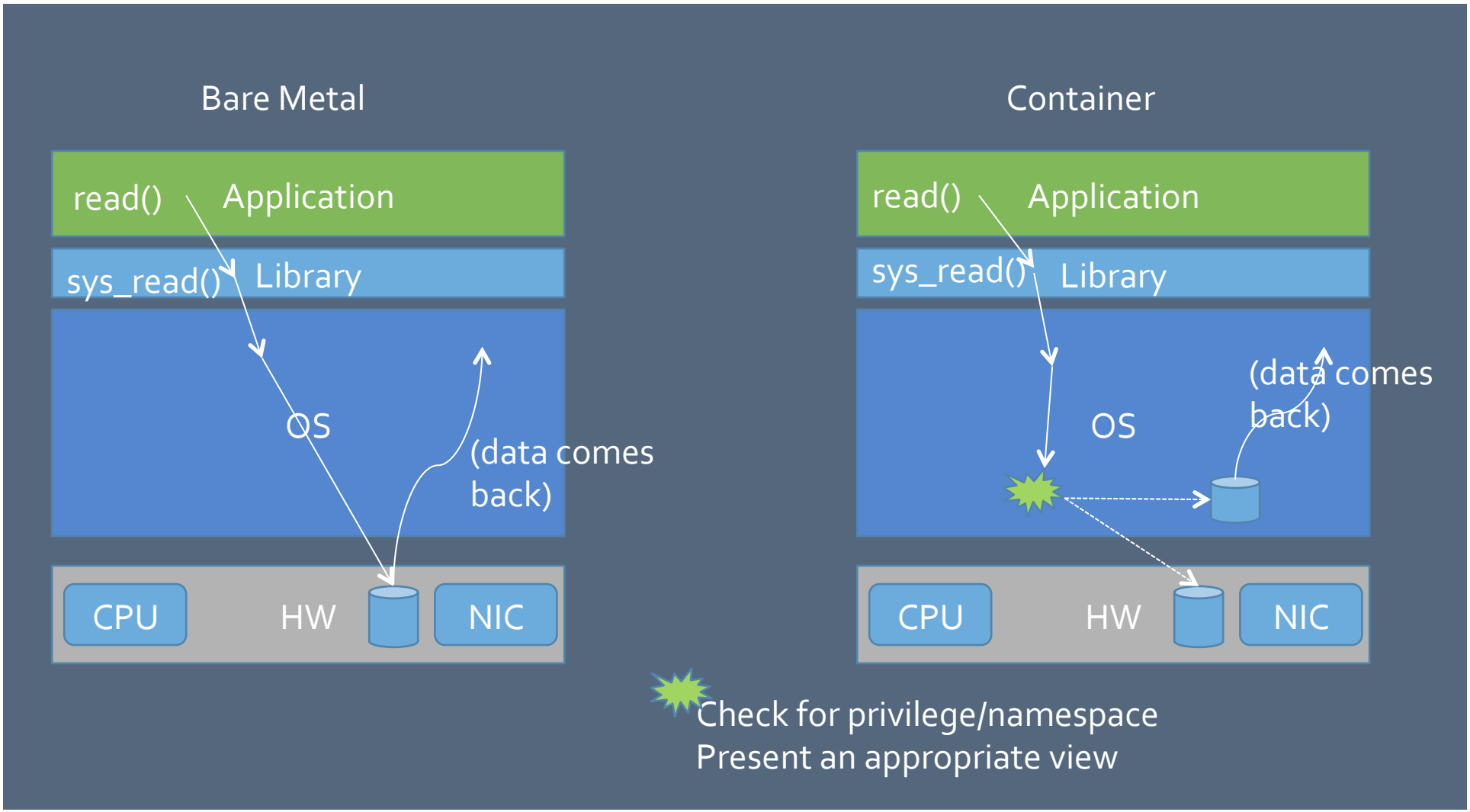
- Need *OS-based* protection and namespaces to limit power of "guest" application
- Leverage layered file system to enable easy composition of images (e.g. OverlayFS)





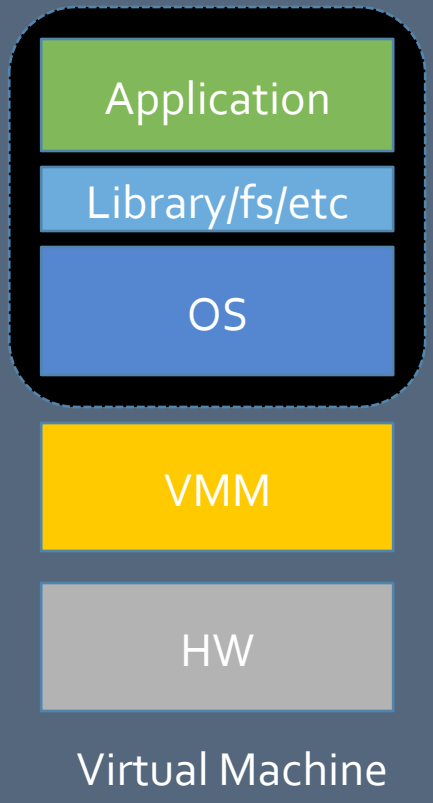
- Still need platform to deploy and manage running instances

EXAMPLE

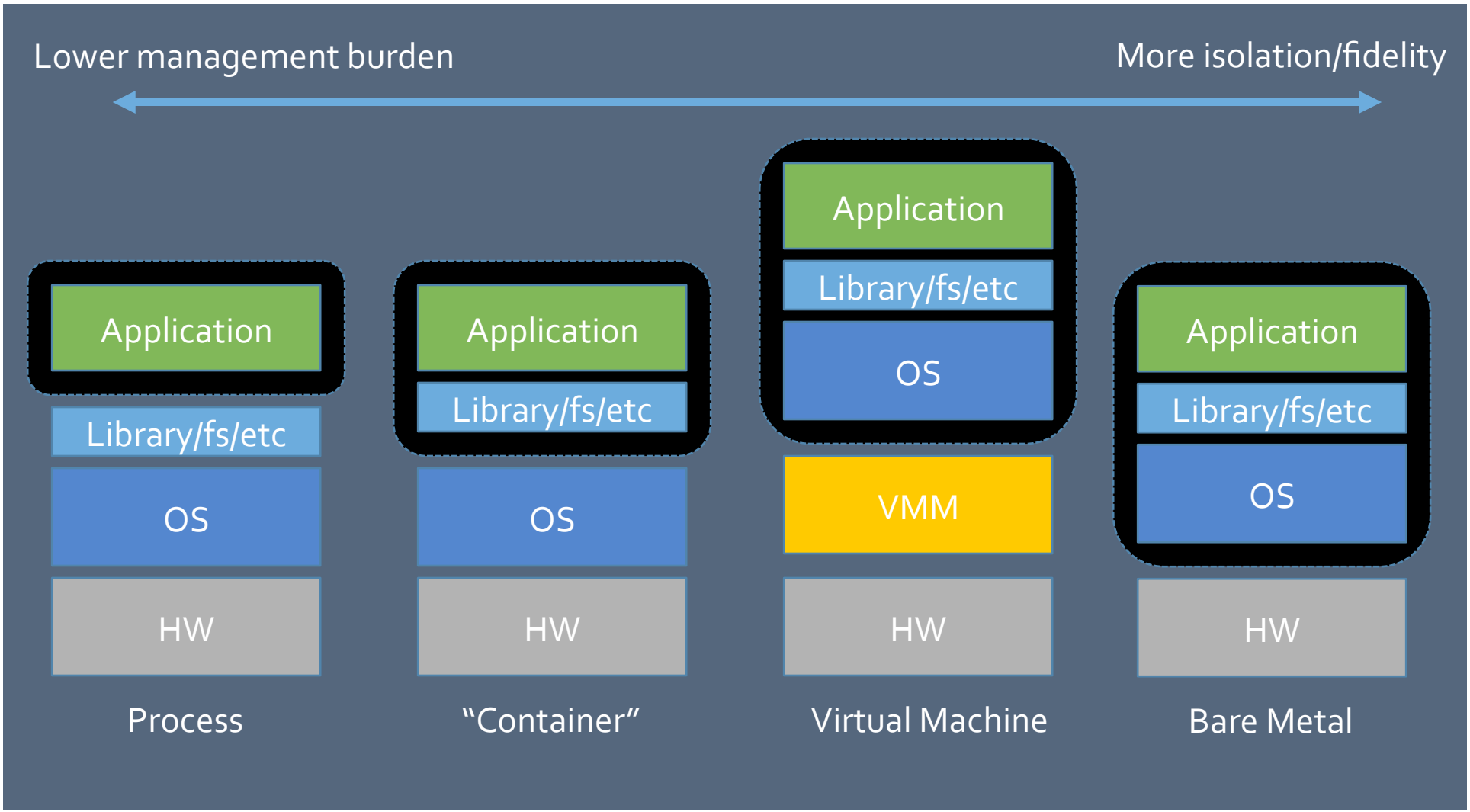

Bare Metal
read()
Application
sys_read()
Library
OS
(data comes back)
CPU
HW
NIC
Container
read()
Application
sys_read()
Library
OS
(data comes back)
CPU
HW
NIC
Check for privilege/namespace
Present an appropriate view

OPTION #4: VIRTUAL MACHINES

- **Virtual machines**: Users get a software container that acts like a physical machine
  - Pro: Decent isolation properties, Good software freedom
  - Con: Performance overhead, Imperfect performance isolation



ENCAPSULATION OPTIONS

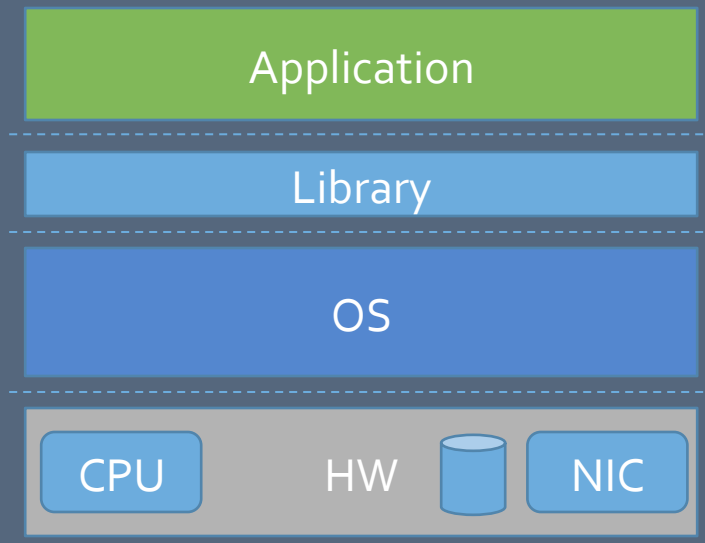


VIRTUAL MACHINES

SOFTWARE CONTRACTS

- CS101: Software layers provide clean interfaces

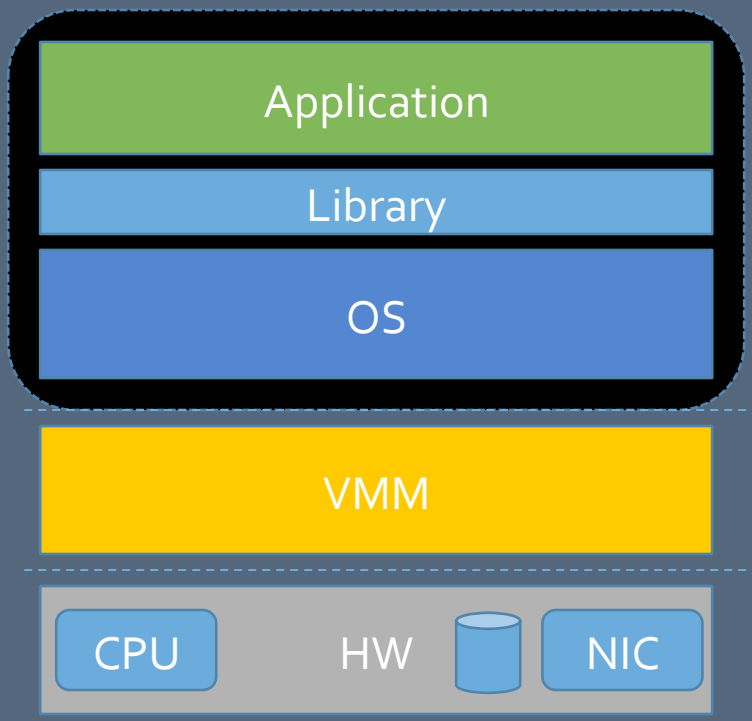
Hardware provides a similar interface



Example Interfaces	Examples
API	read()
ABI, System calls	sys_read()
ISA, device interface	io: in 0x1f7 mmio: mov eax, [0xdeadbeef]

VIRTUAL MACHINES

- Key idea: Add software layer which emulates hardware interface



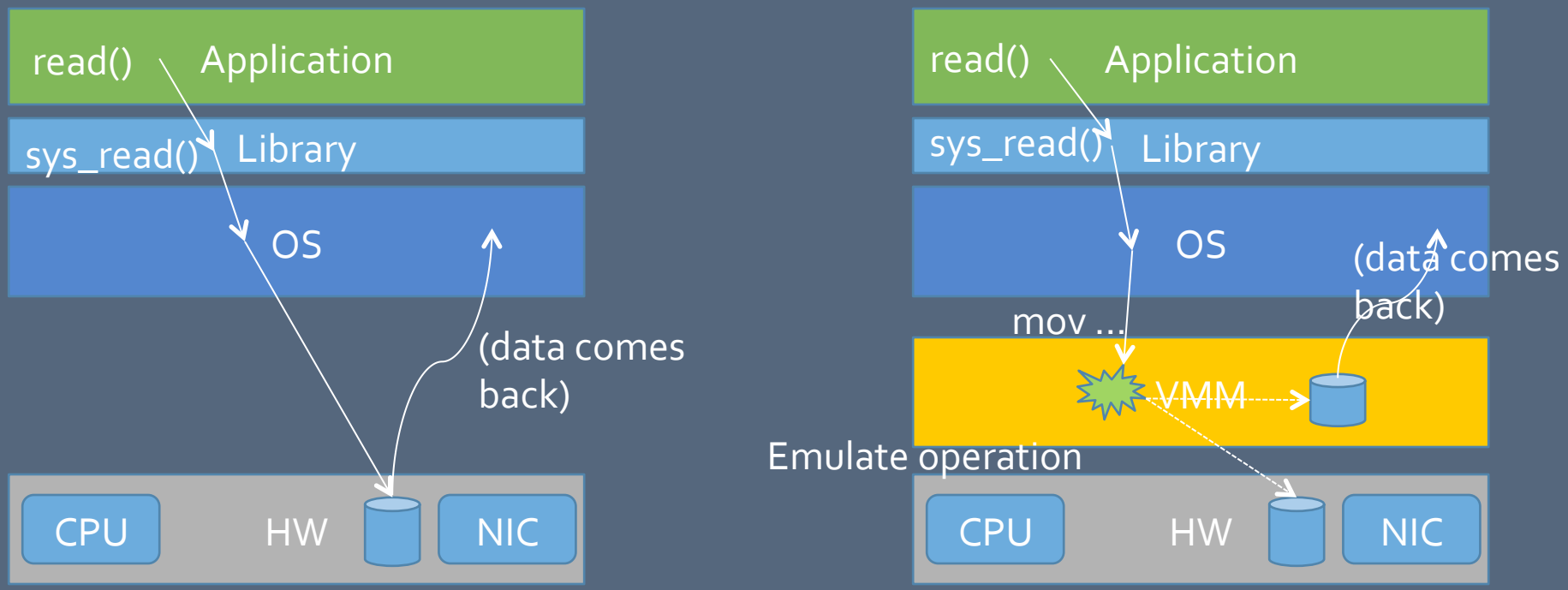
"Virtual machine instance"
or
"Virtual machine"
or
"Guest"

Virtual machine Interface

Machine Interface

Note: The machine and virtual machine interfaces may be different

EXAMPLE


read()
Application
sys_read()
Library
OS
(data comes back)
CPU
HW
NIC
read()
Application
sys_read()
Library
OS
mov …
VMM
(data comes back)
Emulate operation
CPU
HW
NIC

HW VIRTUALIZATION PRINCIPLES

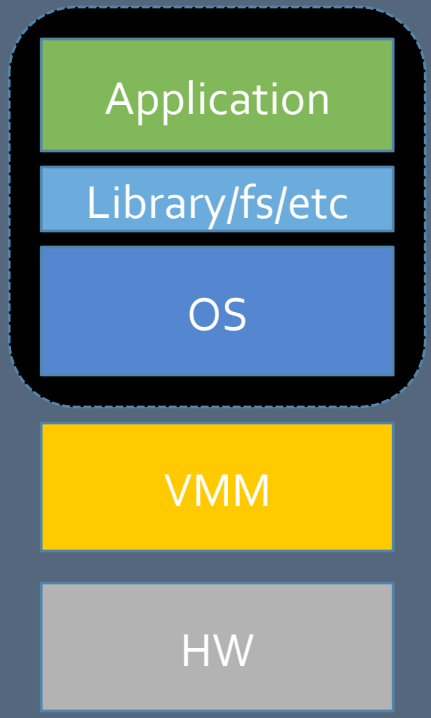
- Hardware virtual machine interfaces should provide:

- Fidelity
  - Software operation identical when virtualized/not
- Isolation
  - A guest may not directly affect VMM or other guests
- Performance
  - Providing fidelity and isolation must not yield unacceptable performance
  - Implies that most operations must execute natively

ASIDE: VIRTUALIZATION OPPORTUNITIES

- VM encapsulation provides broad advantages:

- Compatibility
  - Run software A on system B (e.g. VMM translates IDE to SCSI)
  - Rapid deployment (e.g. scale out on EC2)
- Consolidation
  - Multiple VMs may run on same host
- State capture
  - Suspension of running VM
  - Migration
  - Checkpointing/replication
- Observability
  - Record/replay for debugging/forensics
  - Fault-tolerance
- Others? (many more..)



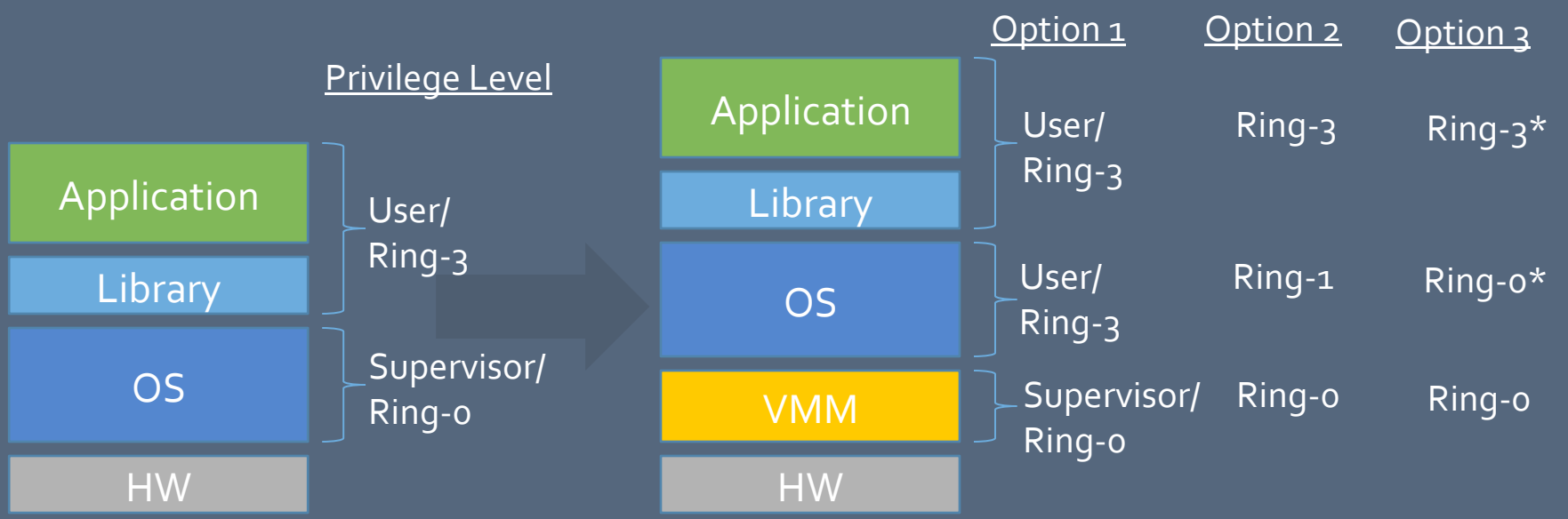
VIRTUAL MACHINES– TECHNIQUES

PRIVILEGED OPERATIONS

- For performance, most instructions may execute directly (e.g. add)
- However, to maintain guest "sandboxes"
  - VMM controls page tables, interrupts, access to devices, etc.
- Typically, a VMM must prevent guest OSes from executing:
  - Privileged instructions (e.g. mov CR3)
  - Privileged instructions that silently fail (e.g. popf)
  - Non-privileged instructions that reveal privileged state (e.g. pushf)
- Non-CPU devices must be treated carefully as well
  - e.g. a VM must not be allowed to cause a DMA into memory not belonging to that VM

ENCAPSULATION PRINCIPLE

- Basic principle: Execute VM software in de-privileged mode
  - Prevents privileged instructions from escaping containment



HANDLING PRIVILEGED INSTRUCTIONS

- Trap-and-emulate
  - Sensitive operations executed in the VM trap to VMM for handling
  - The VMM emulates the behavior of the operation
  - Includes modern HW virtualization such as VT-x
- Static software re-writing/"paravirtualization"
  - Avoid issuing sensitive operations in the VM by re-writing guest OS to leverage VMM "hypercalls"
  - Better performance by sacrificing transparency
- Dynamic software re-writing
  - VMM transparently re-writes portions of the guest's privileged code– coalescing traps
  - Extensively used in PC VMMs prior to HW virtualization maturity
  - Good performance, but VMM may be complex

HW VIRTUALIZATION OF X86 CPUS

- Intel introduced VT-x in 2005
- Essentially, trap-and-emulate with new "non-root" privilege levels (Option 3)
- Redefined behavior of some sensitive operations in non-root mode
- Interrupts are typically delivered to VMM (and vectored to guests as needed)

- A VM Control Structure (VMCS) defines CPU operation
  - *Guest-state area*. Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
  - *Host-state area*. Processor state is loaded from the host-state area on VM exits.
  - *VM-execution control fields*. These fields control processor behavior in VMX non-root operation. They determine in part the causes of VM exits.
  - *VM-exit control fields*. These fields control VM exits.
  - *VM-entry control fields*. These fields control VM entries.
  - *VM-exit information fields*. These fields receive information on VM exits and describe the cause and the nature of VM exits. They are read-only.

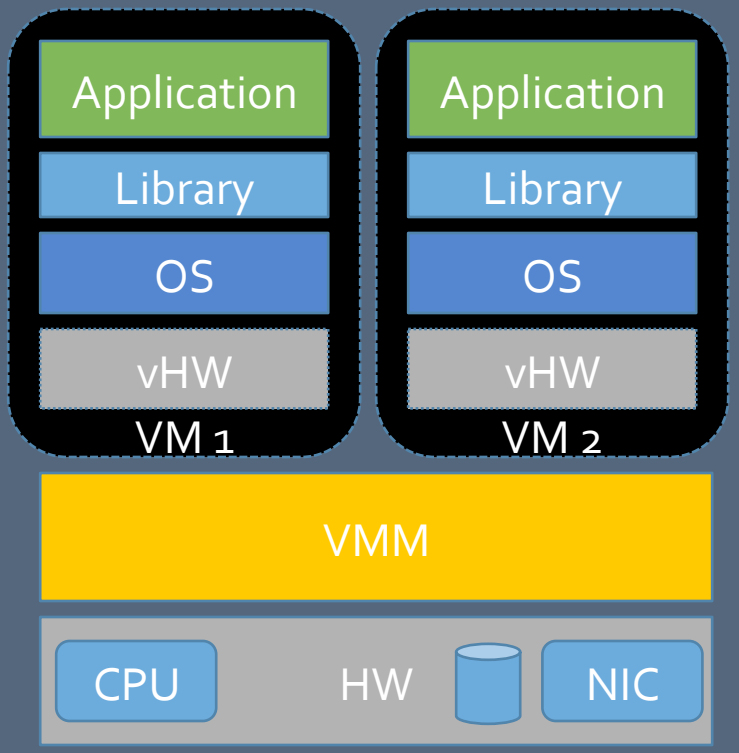
VIRTUALIZING DEVICES

- In principle, all devices may be fully virtualized
  - I.e., trap accesses and emulate
- However, for performance, many alternative techniques exist
  - *Mapping*: Give control to a particular guest
    - May require hardware support (e.g. VT-d)
  - *Partitioning*: e.g. disk drive partitions
  - *Guest enhancements*: provide special VM→VMM calls
  - *Virtualization-enhanced devices*: e.g. NICs with VMDq

VMM ISSUES

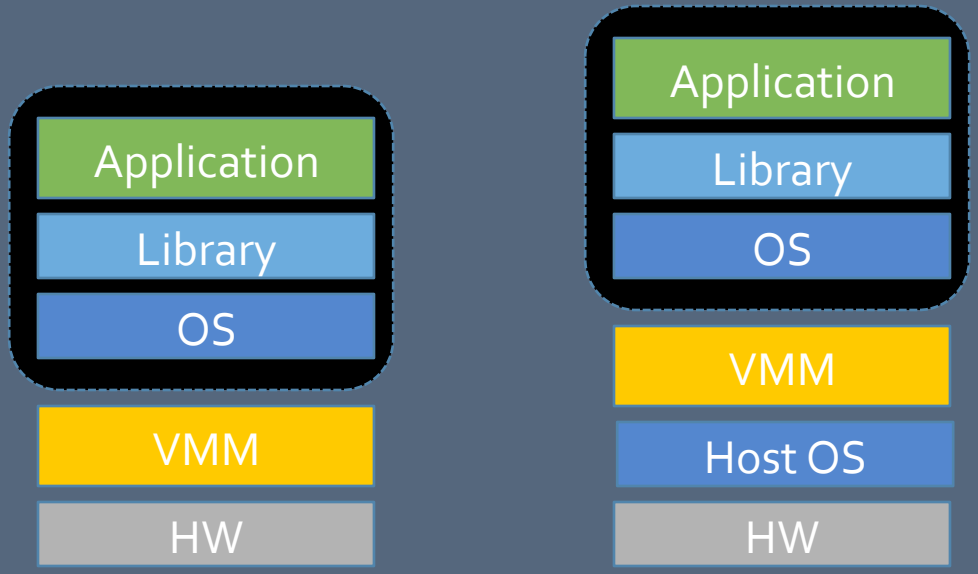
MANAGING MULTIPLE VMS

- Multiple VMs may be handled through either partitioning or time-slicing.
- *Note: the number of virtual cores in the VMs is orthogonal to the number of physical cores.*
- Time-slicing is required if the number of virtual cores exceeds the number of physical cores.
- When the VMs are multiprocessor, *gang-scheduling* the virtual cores may improve performance.



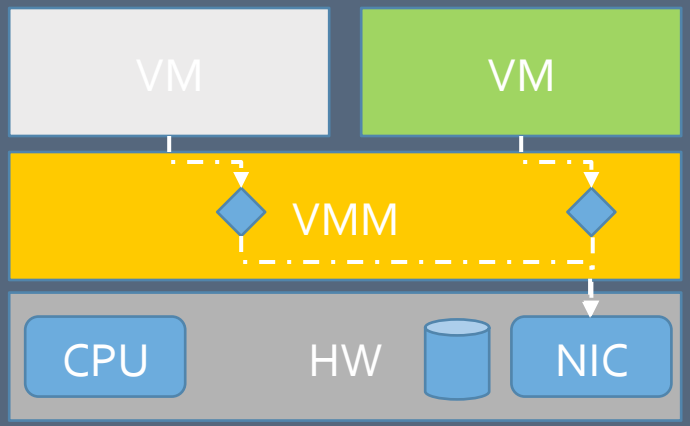
TYPE I VS TYPE II VMMS

- Type I advantages: performance, smaller code base
- Type II advantages: convenience
- One important issue: Where are the device drivers?



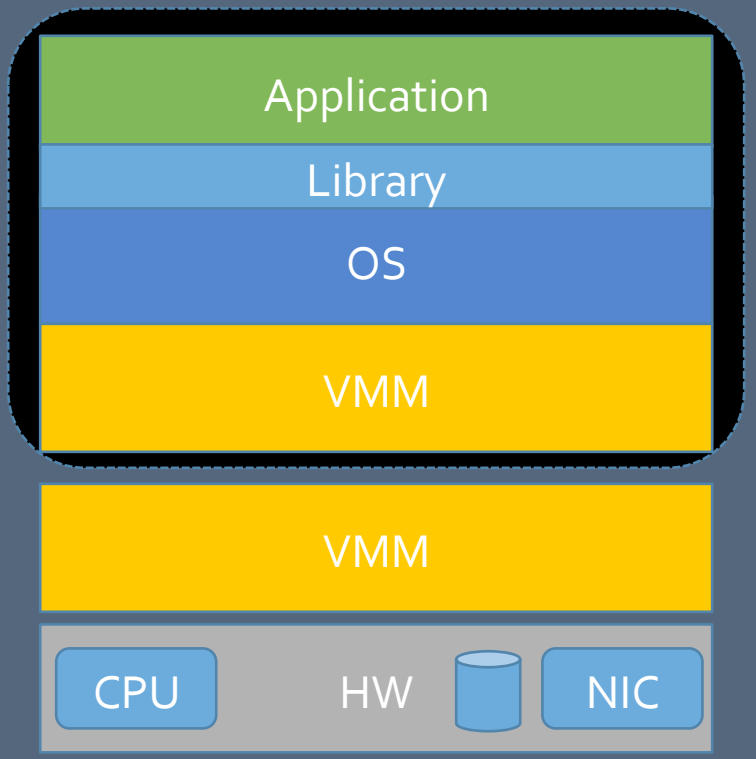
A WORD ON ISOLATION

- Ideally, isolation means that a VM behaves exactly as a PM
  - One guest can not observe another guest– mostly true
  - Performance is independent of number of guests– not true
    - Resources are shared, including caches and network BW
- Some good news: network interference can be reduced
  - All traffic can be intercepted by VMM
  - The VMM can create virtual networks– routing, VLANs



ADVANCED TOPIC

- Recursive Virtualization
  - Virtualizing a virtualized system



- Thought experiment: What system properties would be needed to enable recursive virtualization?

DISCUSSION

- "Xen and the art of virtualization," Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield, *SOSP'03,* http://doi.acm.org/10.1145/945445.945462

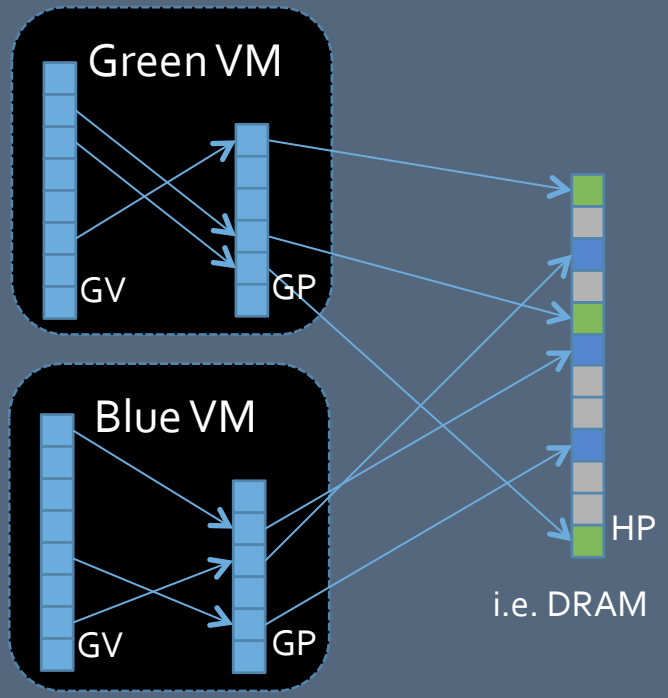
- "Survey of virtual machine research," Goldberg, Robert P., *Computer*, June 1974, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6323581&isnumber=6323564
- "The architecture of virtual machines," Smith, J.E.; Nair, R., *Computer*, May 2005, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1430629&isnumber=30853
- "Container-based operating system virtualization: a scalable high-performance alternative to hypervisors," Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, Larry Peterson, *Eurosys'07,* http://www.win.tue.nl/~tozceleb/2IN05/essay_collection/7%20Container-based%20Operating%20System%20Virtualization.pdf
- "When virtual is better than real," Chen, P.M., Noble, B.D., *Hot Topics in Operating Systems (HotOS 2001),* http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=990073&isnumber=21324

- "kvm: the Linux virtual machine monitor," Kivity, Avi, et al., *Proceedings of the Linux Symposium.* http://www.cs.columbia.edu/~cdall/candidacy/pdf/Kivity2009.pdf
- "Memory resource management in VMware ESX server," Carl A. Waldspurger, *OSDI'02*, http://www.waldspurger.org/carl/papers/esx-mem-osdi02.pdf
- "Resource Containers: A New Facility for Resource Management in Server Systems," Gaurav Banga, Peter Druschel, Jeff Mogul, *OSDI'99,* https://www.usenix.org/legacy/events/osdi99/full_papers/banga/banga.pdf
- "Jails: Confining the omnipotent root," Poul-Henning Kamp, Robert N. M. Watson., http://www.watson.org/~robert/freebsd/sane2000-jail.pdf

BACKUP

VIRTUALIZING MEMORY

- Guest OSes manage "guest physical" memory
  - Mapping "guest virtual" to "guest physical" memory
- VMM manages "host physical" memory
  - Mapping "guest physical" to "host physical"

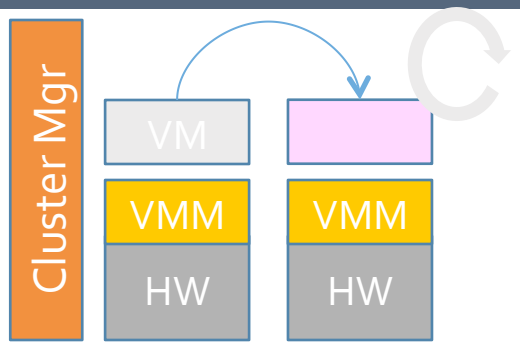


- Management mechanisms include:
  - SW: Shadow page tables
  - HW: Extended page tables (EPT)

VM STATE CAPTURE

- One can capture the state of a running VM for replication, migration, offline debugging, etc.
- Capturing the state of a running VM implies all *architectural* state
  - Processor state: registers (incl. general purpose, privileged, EIP), hidden state
  - Device state: config, buffers, transactions in flight, etc
  - Memory state: config, contents
  - Disk image

COOL TECHNIQUE: LIVE MIGRATION



- VM encapsulation enables migration of running software
  - Useful in data center for load balancing, upgrades, etc
  - Not new, just "easier" (see *process migration*)
- Metric: machine "downtime"
  - Success if less than typical network hiccups (TCP timeout)
- Challenge: a lot of memory to move
  - (not disk, disk images are typically available at source and target through SAN)
  - (not network reconfig if source and target on same subnet)
- Strategy: pre-transfer the memory pages
  - Then, check for additional dirty pages, wash, rinse, repeat
  - When # dirty pages < threshold, send remaining, jump