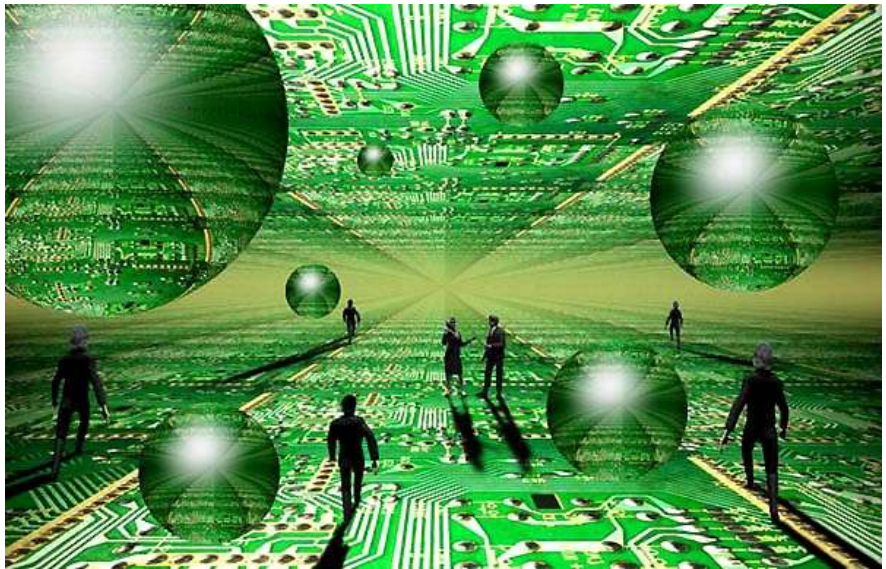
NYU
New York University

CSCI-GA.3033-015

Virtual Machines: Concepts & Applications

Lecture 1: So … What Is A Virtual Machine?

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com



Who Am I?

- Mohamed Zahran (aka Z)
- http://www.mzahran.com
- Research interest:
  - computer architecture
  - hardware/software interaction
  - Biologically-inspired machines
- Office hours: Tue 1-3 pm
  - or by appointment
- Room: WWH 320

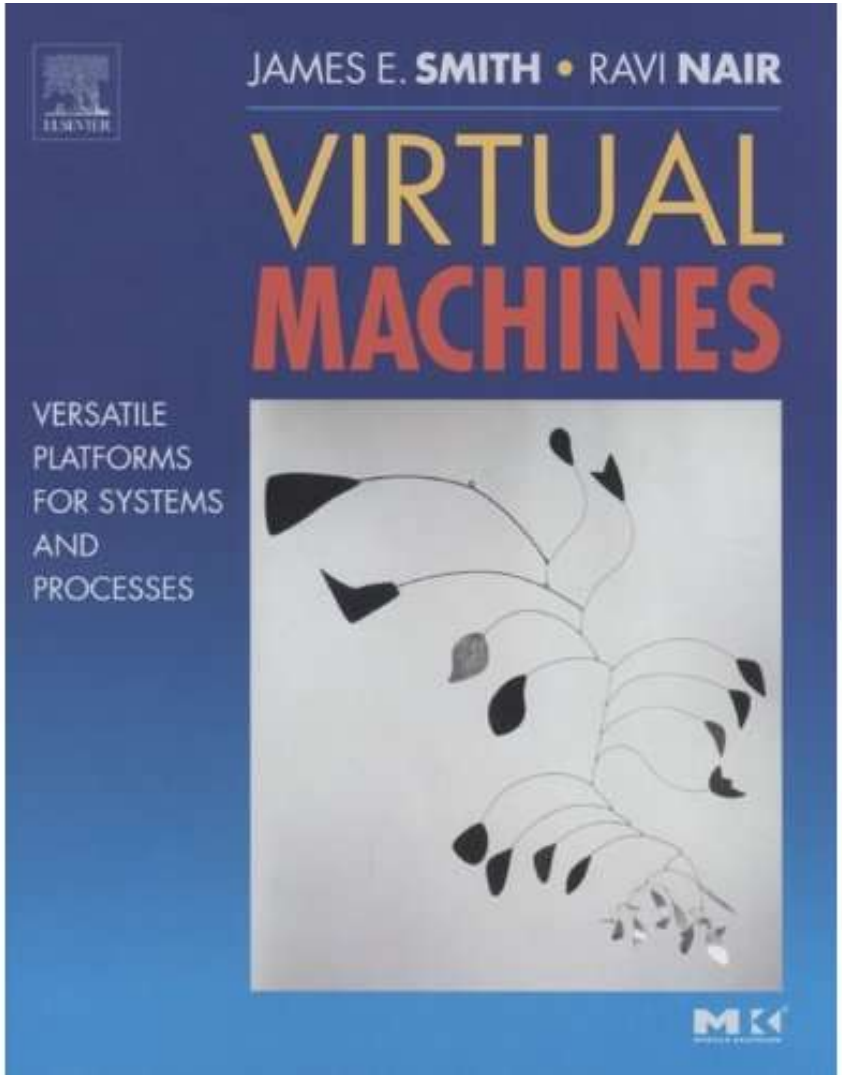
Formal Goals of The Course

- Understand VM architectures and applications
- Study key implementation technologies
- Focus on architecture and microarchitecture aspects; as well as software aspects
- Cover significant case studies

Informal Goals of This Course

- Be an expert in how programming languages, compilers, OS, and computer architecture interact together!
- Be able to use the technology learned in this course in many different situations
- Build a vision about technology and its future
- Enjoy the course

The Textbook



Grades

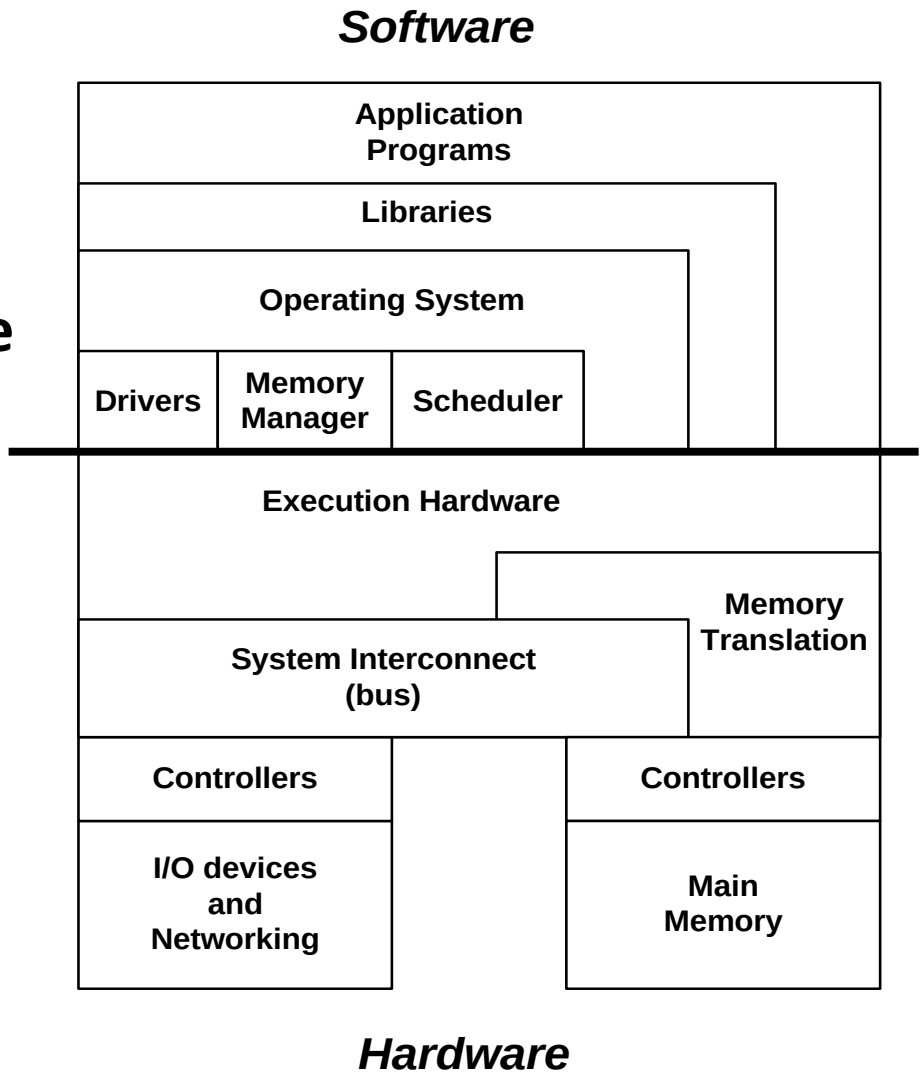
- You have 3 sources of study information:
  - Slides
  - Notes you take in class
  - Reading material from the textbook
- Exam is open book/notes
- Grade distribution:
  - Homework assignments 30%
  - Project 30%
  - Final exam 40%

We Have to Admit that …

- Computers are very complicated structures!
- In order to manage/design them, we need to be able to manage extreme complexity!
- The best way to do that is through: levels of abstraction with well defined interfaces

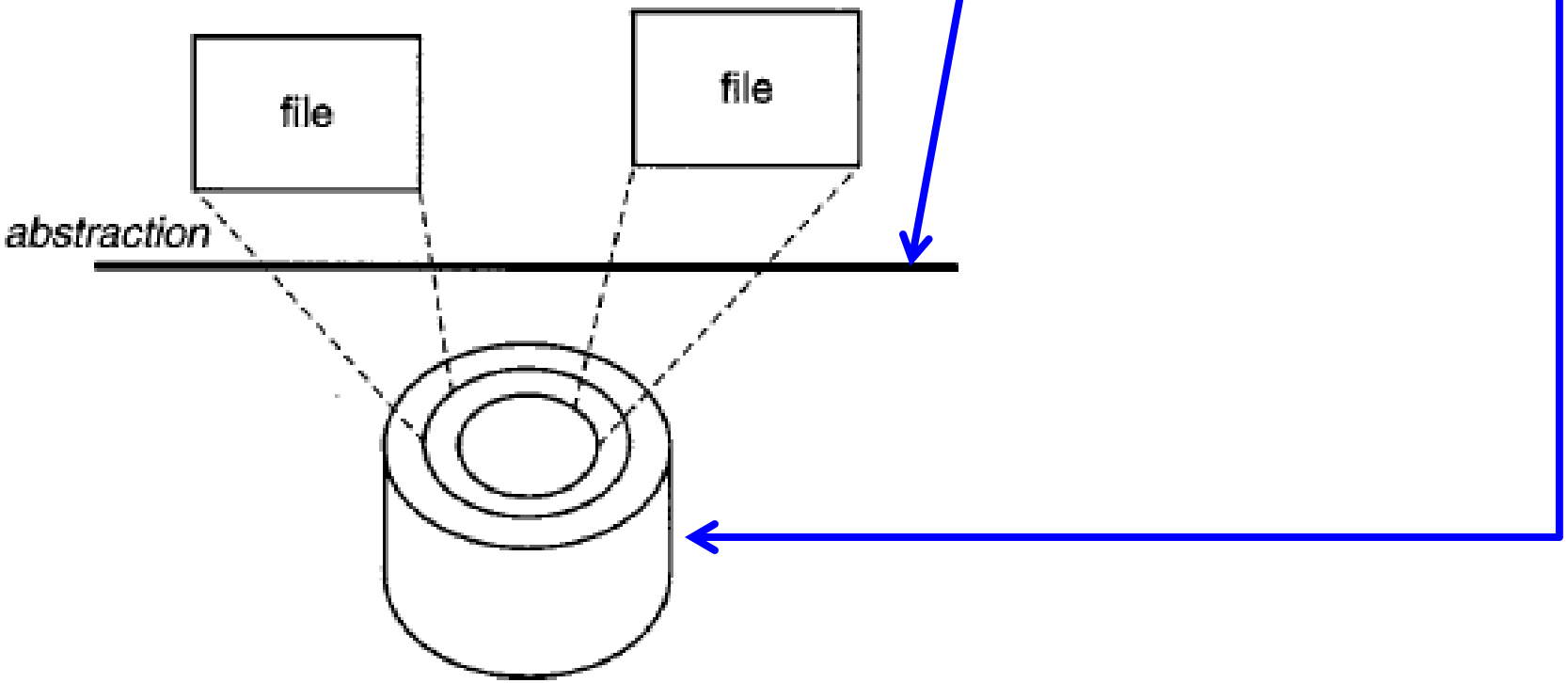
Abstraction

- Computer systems are built on levels of abstraction
  - **Higher level of abstraction hide details at lower levels**
  - **Example: files are an abstraction of a disk**

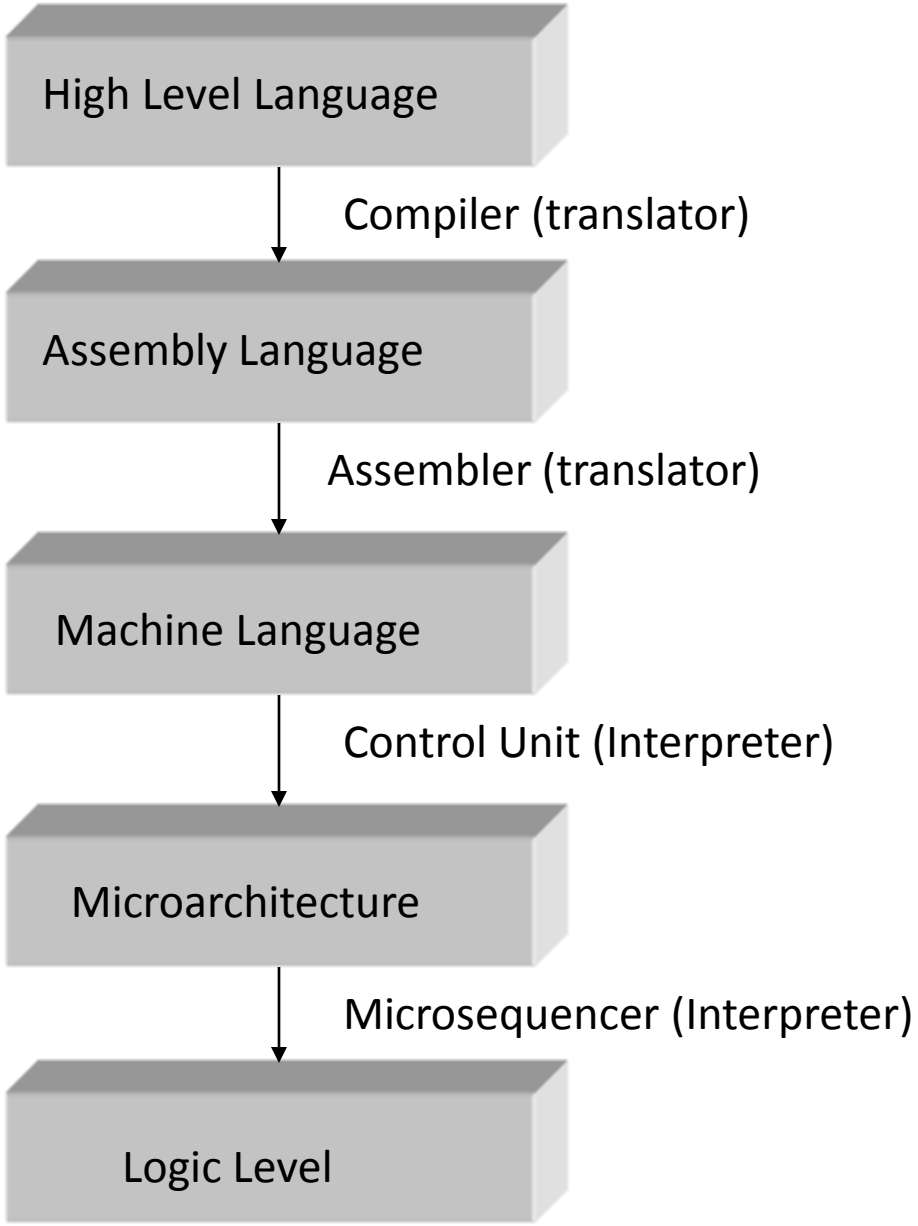


Abstraction provides:

Simplified interface to underlying resources.



Problem → Algorithm Development → Programmer

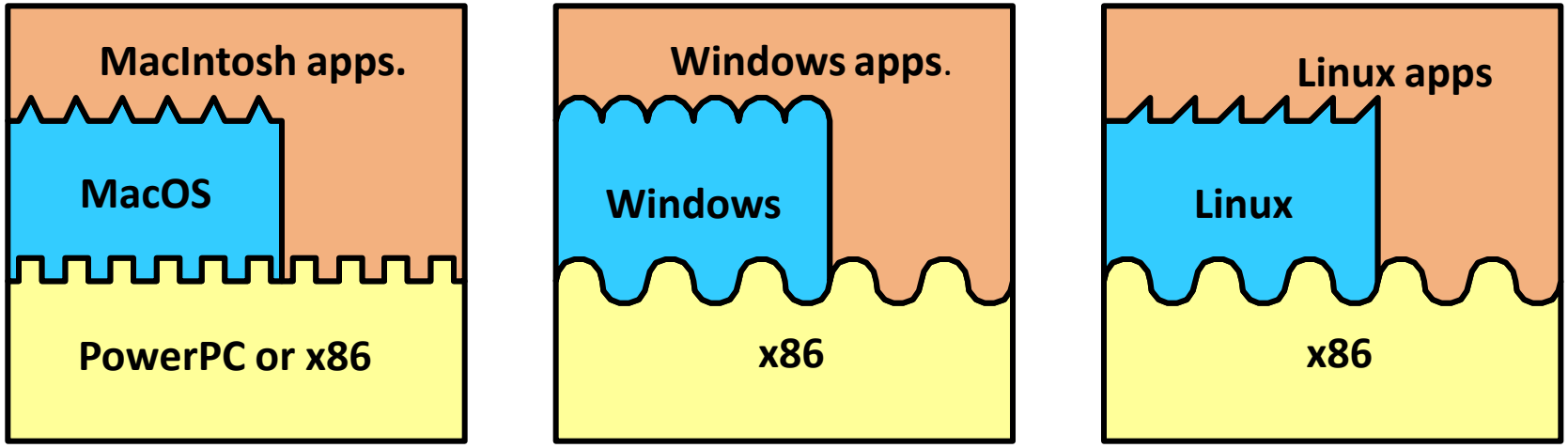


Device Level → Semiconductors → Quantum

Advantages of Well-defined Interfaces

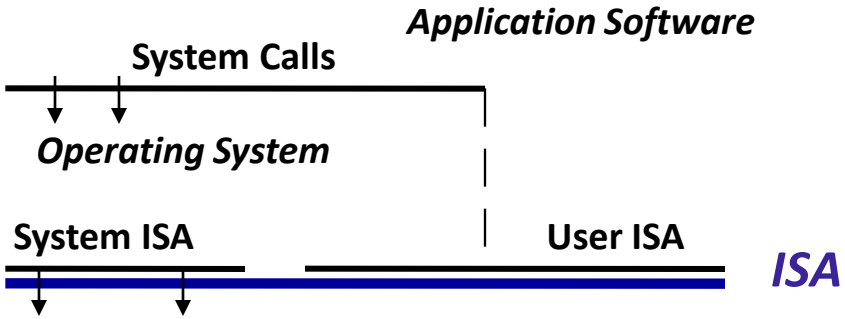
- Major design tasks are decoupled
- Different hardware and software development schedules
- Example of interfaces:
  - Instruction set architecture (ISA)
  - OS interface (system calls)

- Software can run on any machine
  supporting a compatible interface



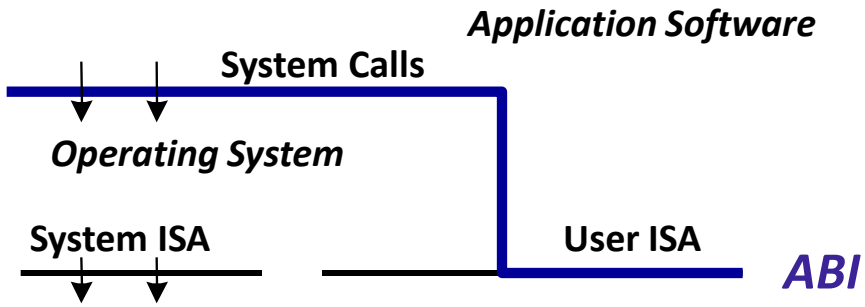
Major Program Interfaces

- ISA Interface -- supports all conventional software



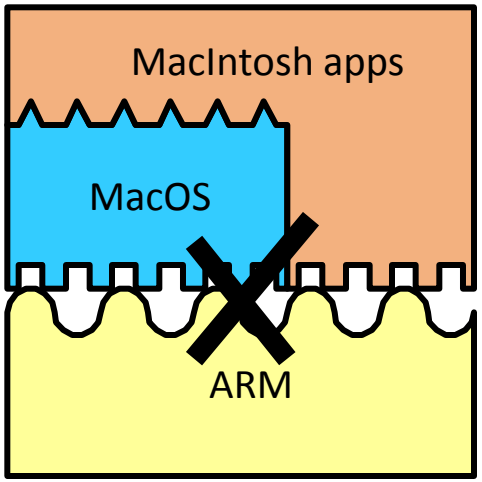
- **Application Binary Interface (ABI)**

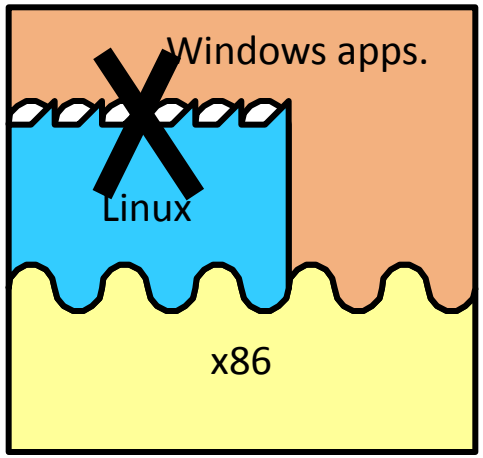
  -- supports application software only



There are also disadvantages…

- Software compiled for one ISA will not run on hardware with a different ISA
  - Apple Mac (PowerPC) binaries on an x86?
- Even if ISAs are the same, OSes may differ
  - Windows NT applications on a Solaris x86?
- Binary may not be optimized for the specific hardware platform it runs on
  - Intel Pentium 4 binaries on an AMD Athlon?

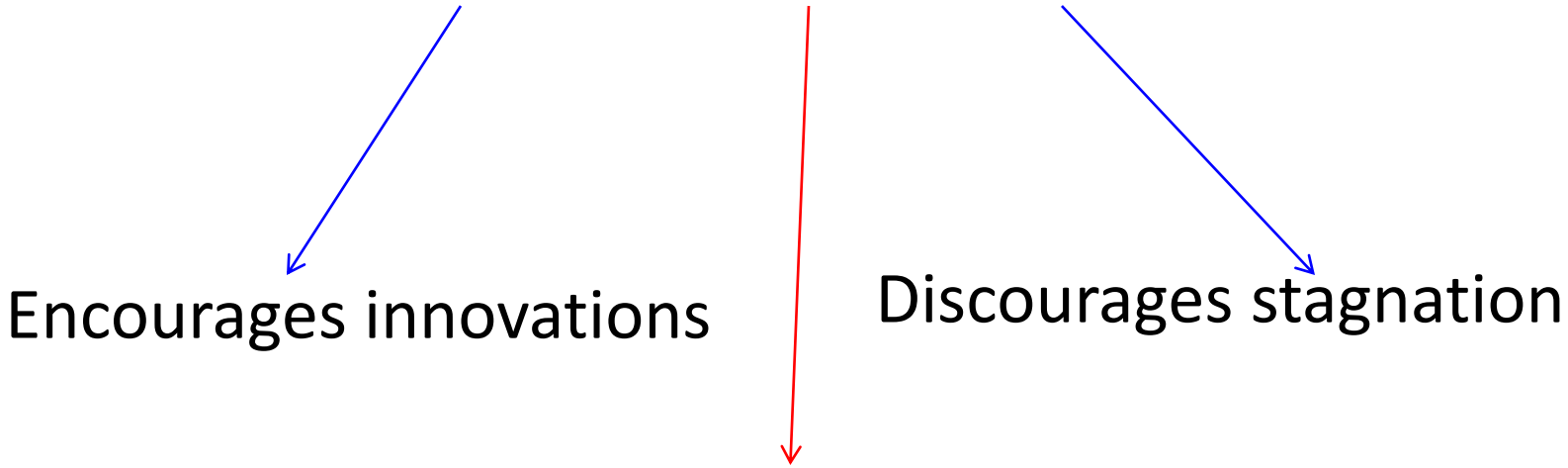




Disadvantages (contd.)

- Innovation may be inhibited by fixed ISA
  - Hard to add new instructions
    - *OR* remove obsolete ones
  - What was the most recent (successful) new ISA?
    Or new OS?
- Difficult for software to interact directly with implementation
  - Performance features
  - Power management
  - Fault tolerance
  - Software is *supposed* to be implementation independent

Diversity in instruction sets, OSes, and programming languages



BUT

In practice, diversity leads to reduced interoperability.

How to deal with this in our world of networked computers, where it is advantageous To move software as freely as data??

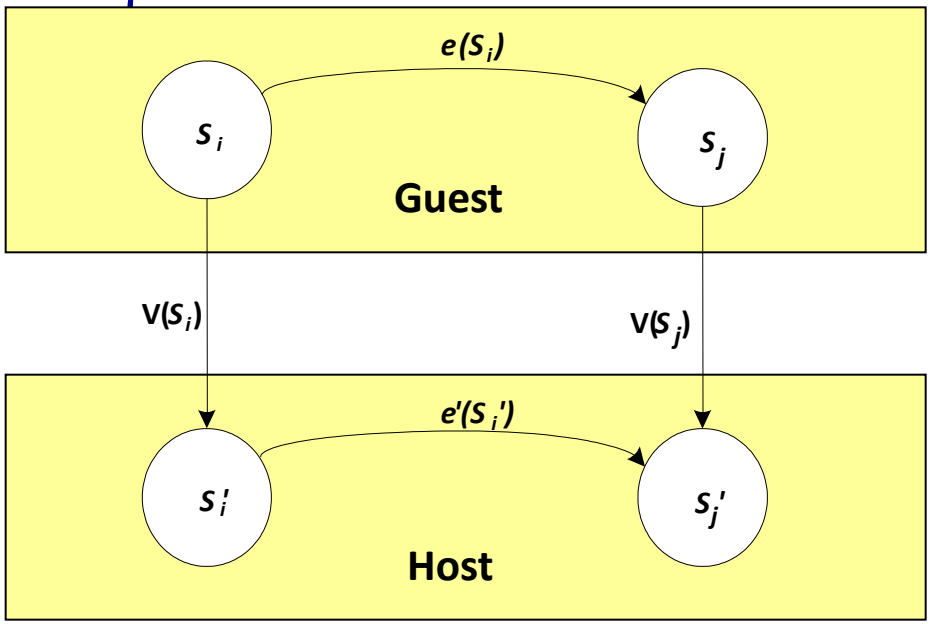
A Look at Hardware Resources

- Conventional system software manages hardware resources directly
  - An OS manages the physical memory of a specific size
  - I/O devices are managed as physical entities
- Difficult to share resources except through OS
  - All users of hardware must use the same OS
  - All users are vulnerable to attack from other users sharing the resource (via security holes in OS)

Can we do better?

Virtualization is the answer!

- Real system is transformed so that it appears to be different!
- An isomorphism from guest to host
  - Map guest state to host state
  - Implement "equivalent" functions

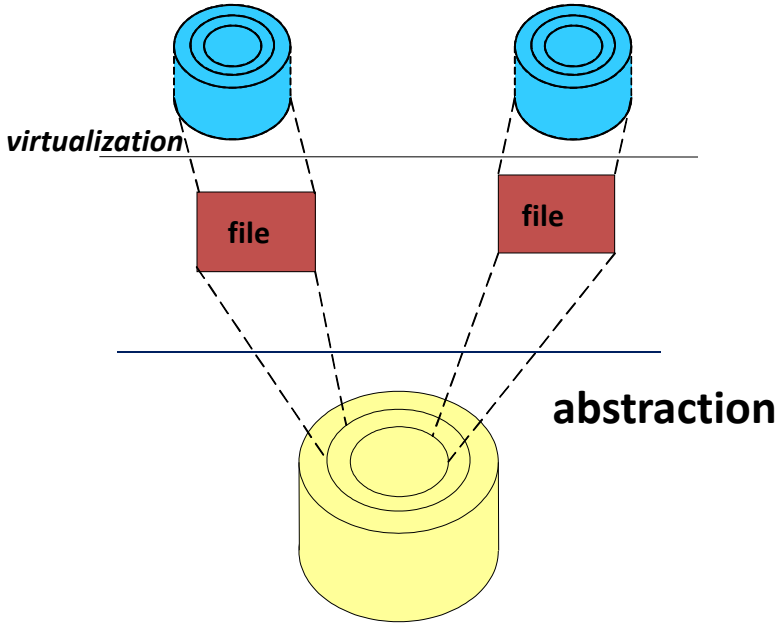


Virtualization

- Similar to abstraction

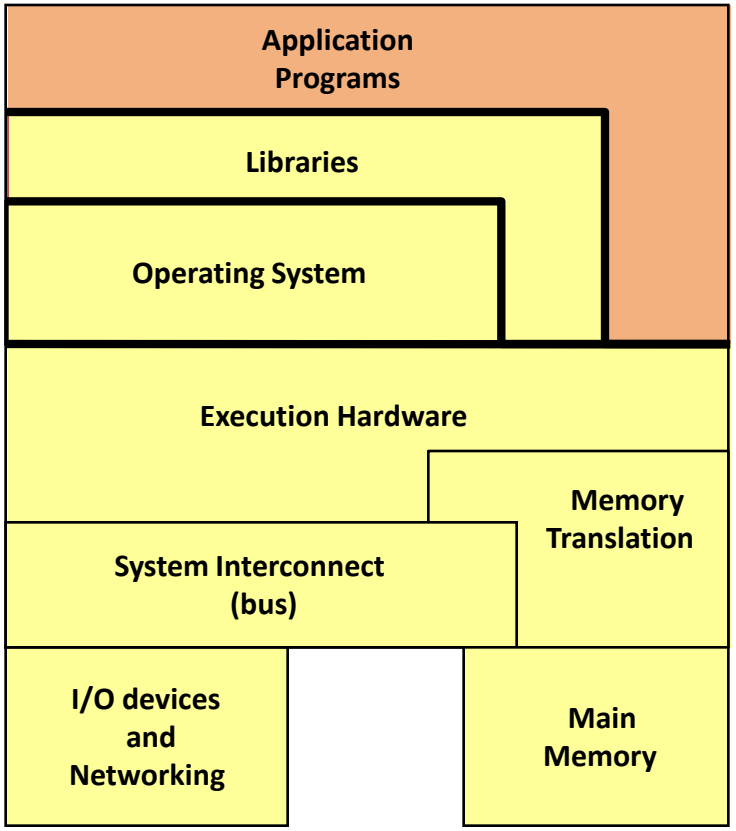
  Except

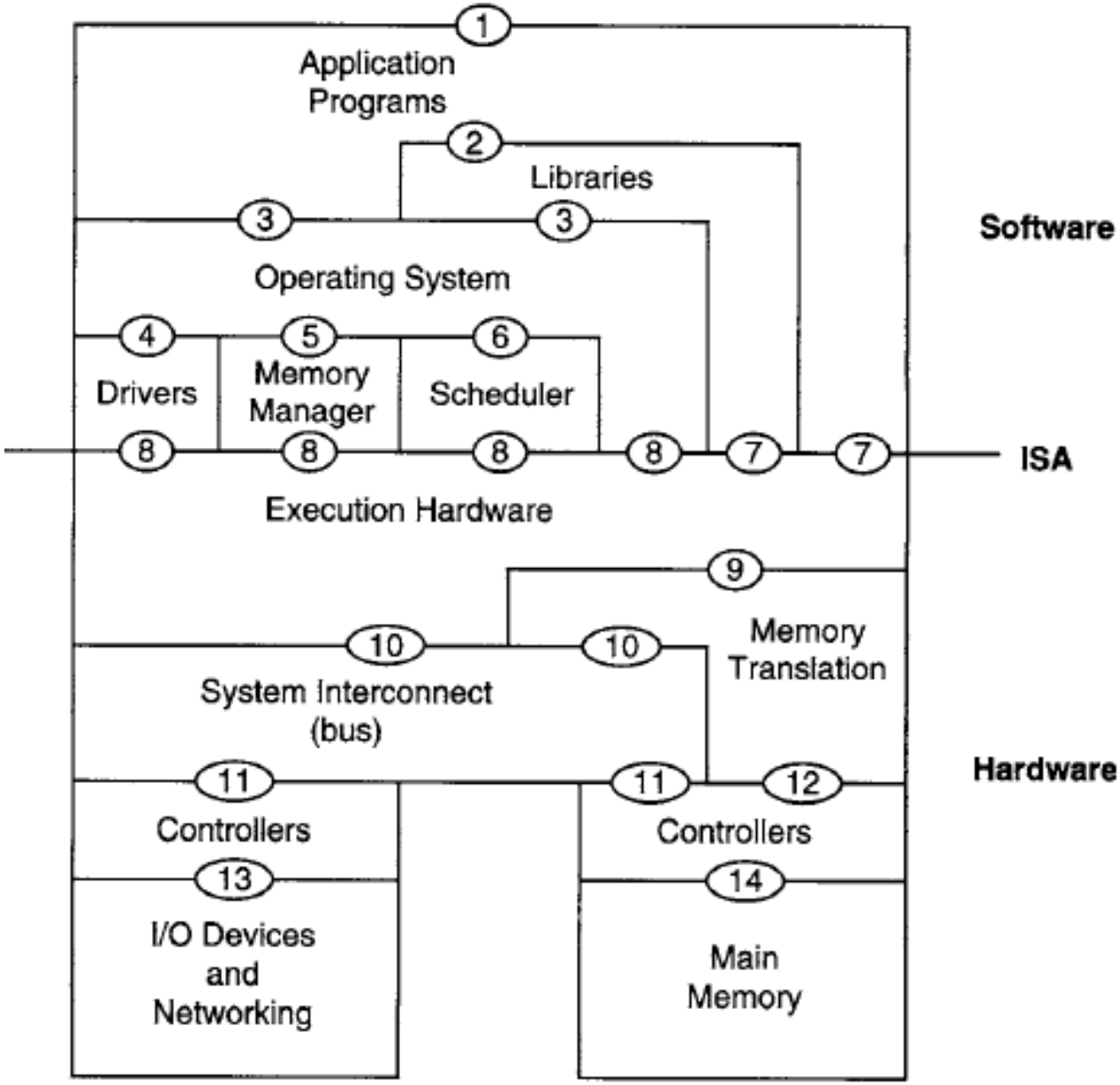
  – Details not necessarily hidden
- Construct Virtual Disks
  – As files on a larger disk
  – Map state
  – Implement functions



The "Machine"

- Different perspectives on what the *Machine* is:
  - OS developer
  - Compiler developer
  - Application programmer

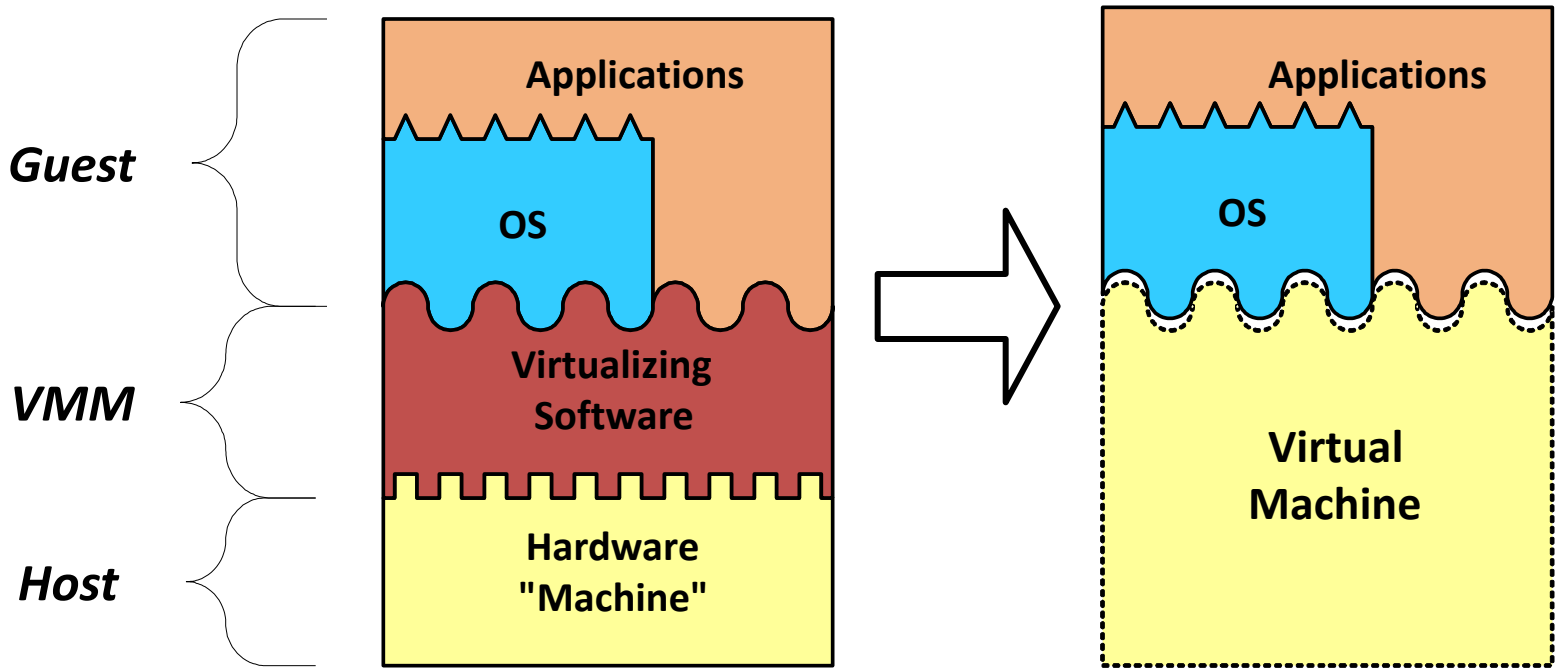



1
Application Programs
2
Libraries
3
3
Operating System
Software
4
5
6
Drivers
Memory Manager
Scheduler
8
8
8
8
7
7
ISA
Execution Hardware
9
10
10
Memory Translation
System Interconnect (bus)
Hardware
11
11
12
Controllers
Controllers
13
14
I/O Devices and Networking
Main Memory

Virtual Machines

add *Virtualizing Software* to a *Host* platform
and support *Guest* process or system on a *Virtual Machine* (VM)

Example: System Virtual Machine



Example of VM Usages

- A virtualizing software installed on an Apple Macintosh can provide a Windows/IA-32 VM capable of running PC application programs.
- Multiple, replicated VMs can be implemented on a single hardware platform to provide groups/individuals with their own OS environments

Example of VM Usages

- A large multiprocessor server can be divided into smaller virtual servers.
- VM can provide dynamic, on-the-fly optimization of program binaries.
- …

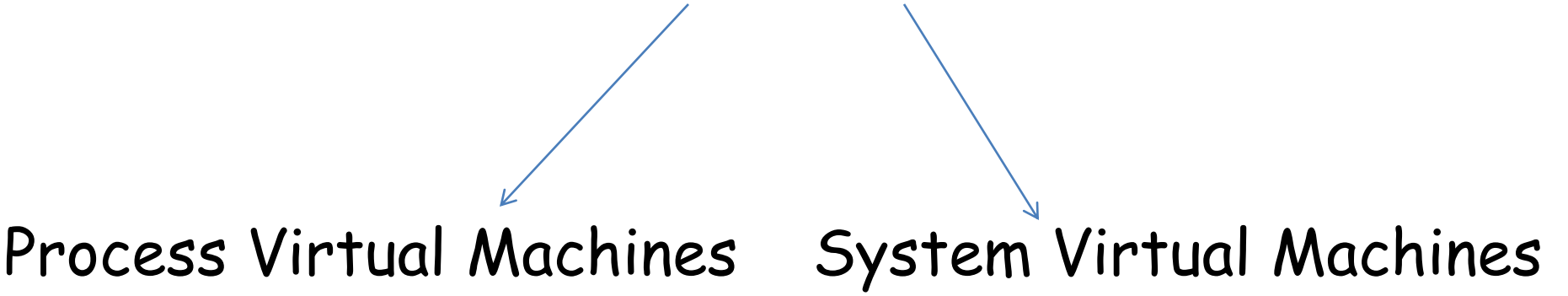
The Family of Virtual Machines

- Lots of things are called "virtual machines"
  - IBM VM/370
  - Java
  - VMware

Some things *not* called "virtual machines", *are* virtual machines

- IA-32 EL
- Dynamo
- Transmeta Crusoe

Virtual Machines

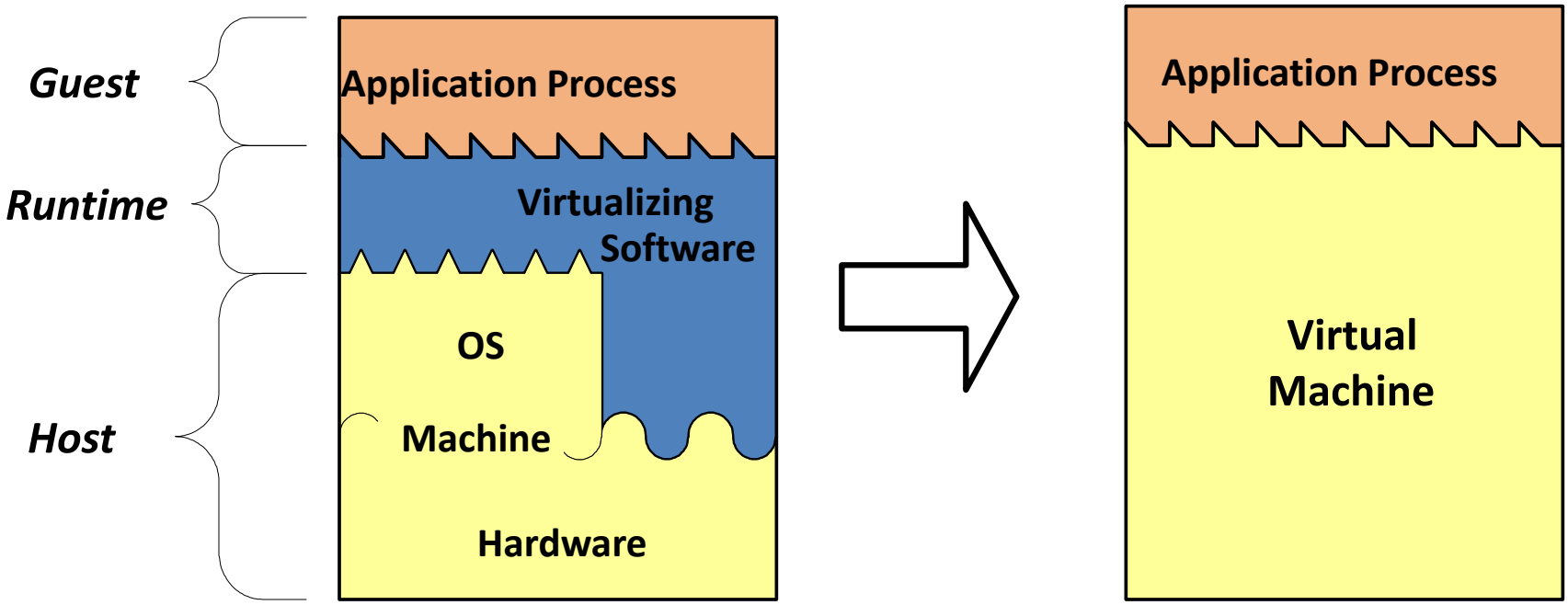


The process of virtualization involves two steps:

1. Mapping of virtual resources or state to real resources of the underlying machine.
2. Using real machine instructions and/or system calls to carry out the actions specified by the VM instructions.

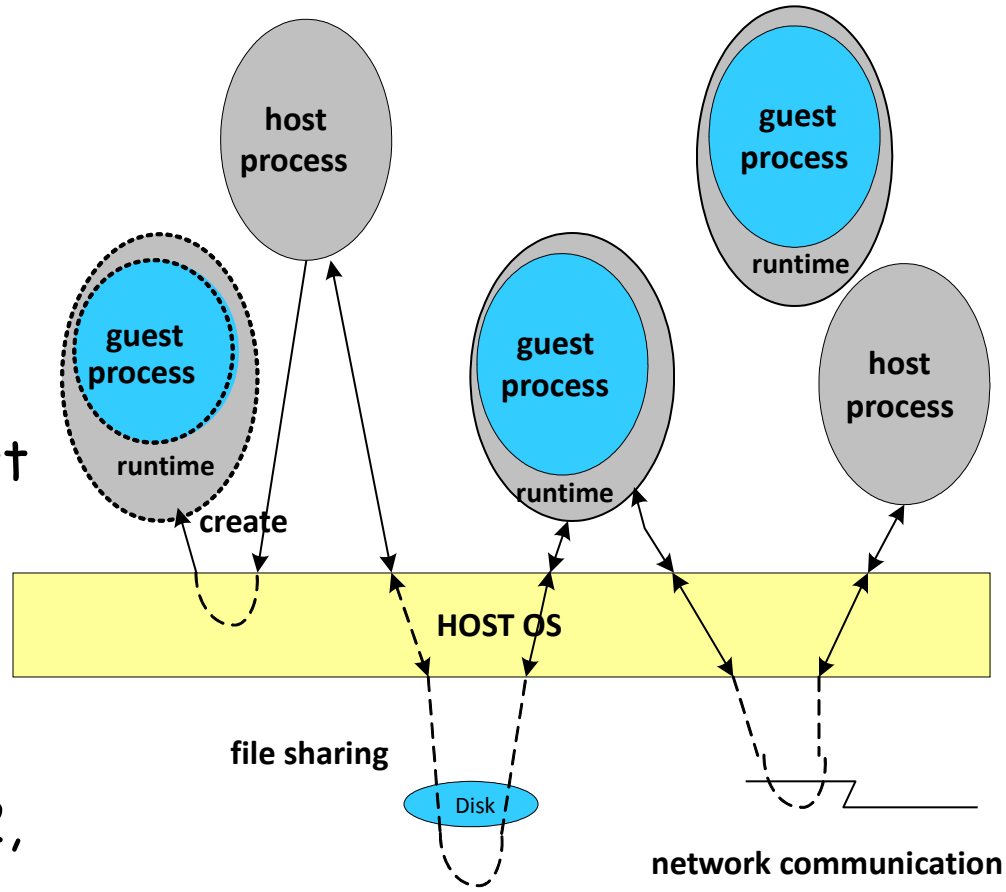
Process VMs

- Execute application binaries with an ISA *different* from hardware platform
- Provide user application with a virtual ABI environment
- Can provide: replication, emulation, and optimization
- Examples: IA-32 EL, FX!32

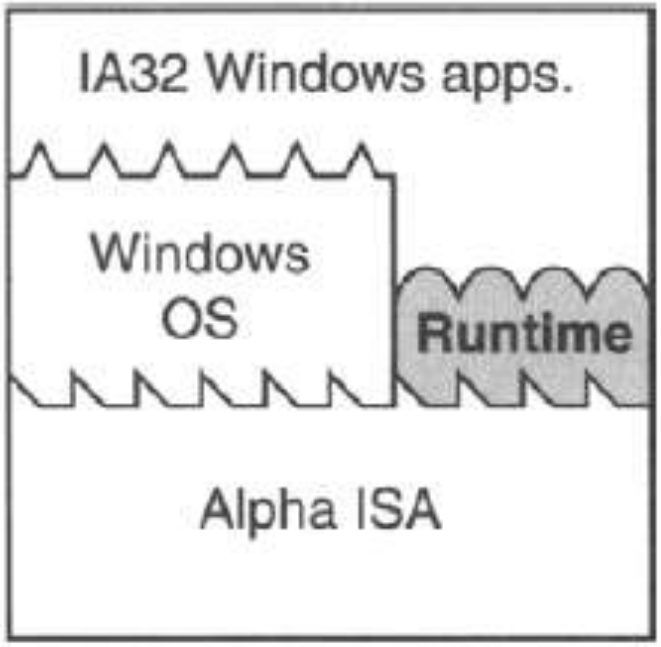


Process Virtual Machines

- Constructed at ABI level
- *Runtime* manages guest process
- Not persistent
- Guest processes may intermingle with host processes
- As a practical matter, guest and host OSes are often the same
- Dynamic optimizers are a special case
- Examples: IA-32 EL, FX!32, Dynamo



Example of Process VM: FX!32



Application compiled for source ISA
(in this example IA-32)

Executed on a machine with target ISA
(in this example Dec Alpha)

Process VM: Replication

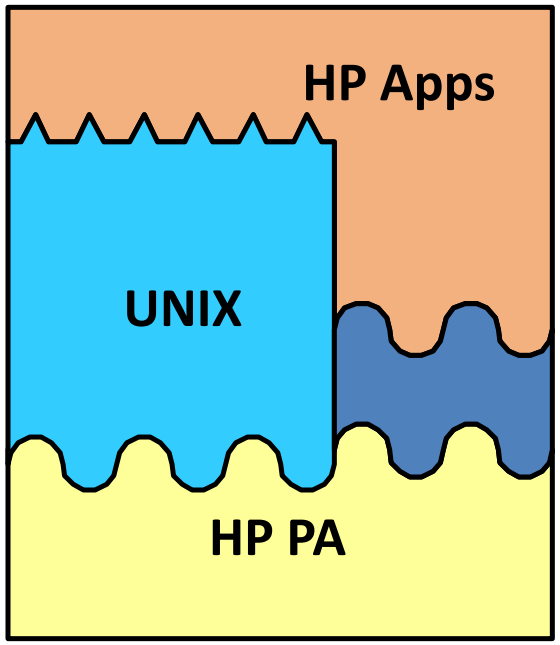
- OS providing the illusion of multiprogramming
- Each user process thinks it has the complete machine to itself

Process VM: Emulation

- Supports program binaries compiled for a different instruction set than the host hardware → emulates one instruction set on hardware designed for another instruction set
- Interpretation Vs Translation

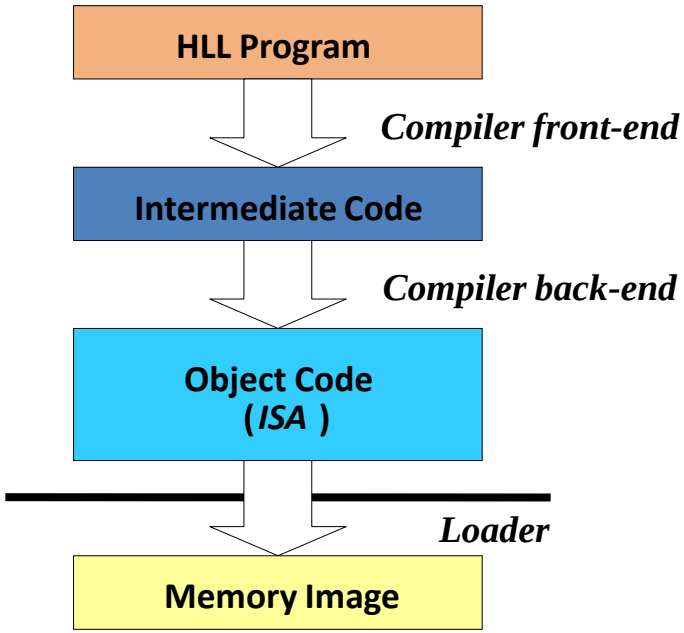
Process VM: Optimization
Same-ISA Dynamic Binary Optimizers

- Optimize Binary at Runtime
- Instruction sets for host and guest are the same
- Example HP Dynamo
  - Can optimize for dynamic properties of program
  - Can optimize for a specific processor implementation

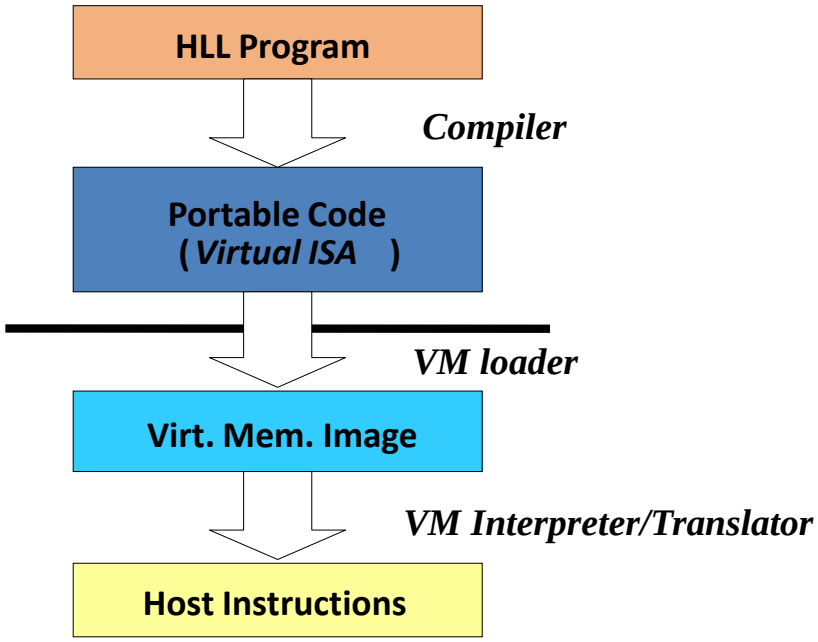


Process VM: High Level Language Virtual Machines

- VM environment does not directly correspond to any real platform
- VM environment designed for:
  - ease of portability
  - to match features of HLL used for program development.

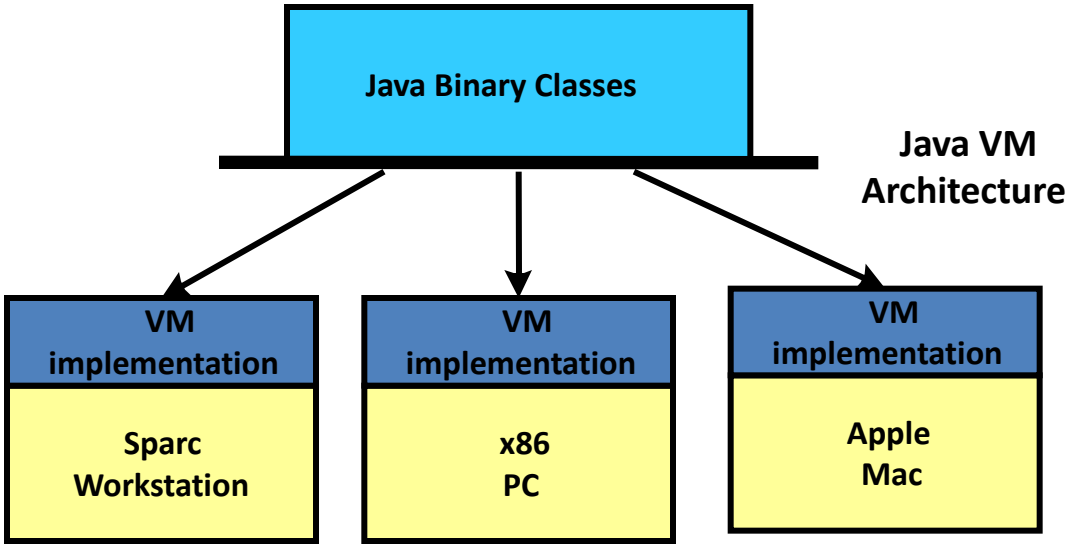


Traditional



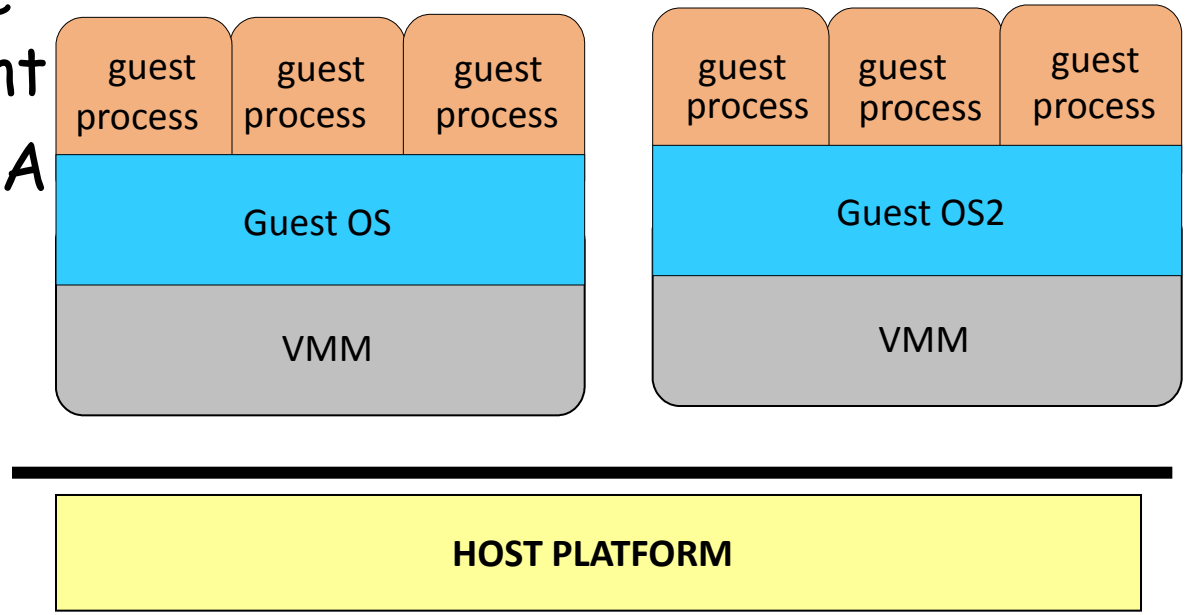
HLL VM

HLL VM: Example JVM



System Virtual Machines

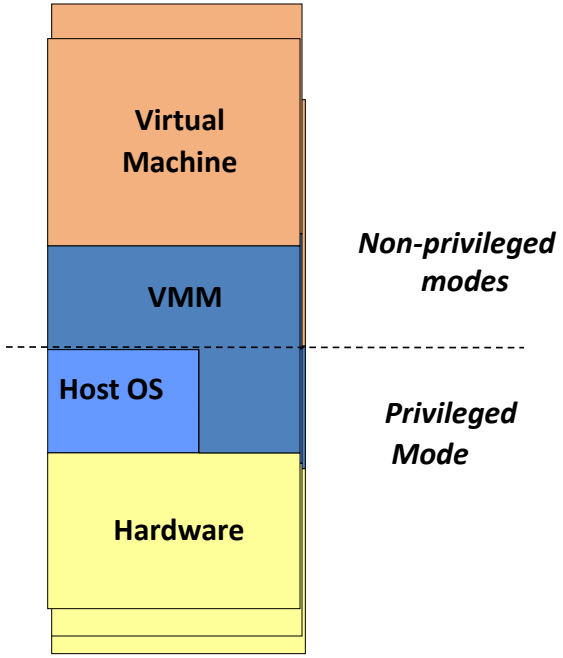
- Provide a complete system environment
- Constructed at ISA level
- Persistent
- Examples: IBM VM/360, VMware, Transmeta Crusoe



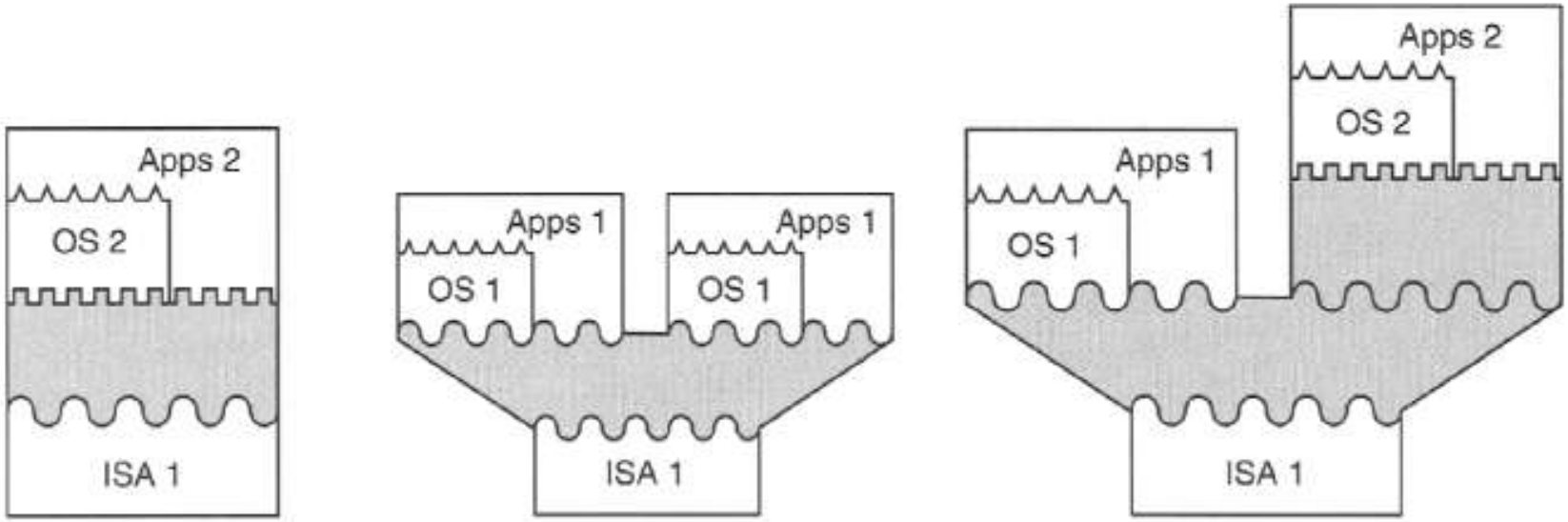
A single host hardware platform can support multiple guest OS environments simultaneously.

System Virtual Machines

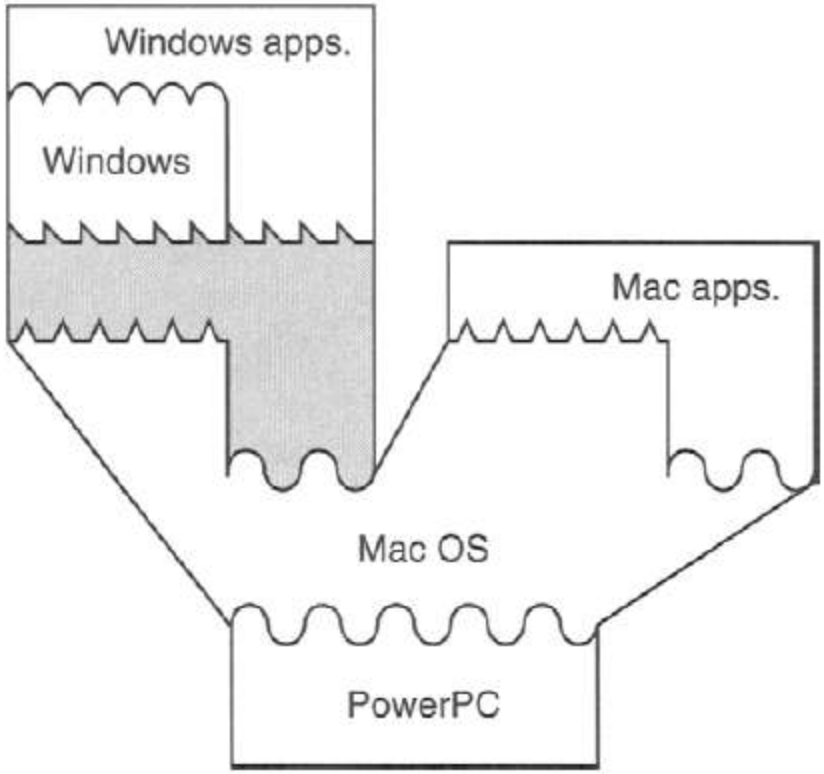
- Native VM System
  - VMM privileged mode
  - Guest OS user mode
  - Example: classic IBM VMs
- User-mode Hosted VM
  - VMM runs as user application
- Dual-mode Hosted VM
  - Parts of VMM privileged; parts non-privileged
  - Example VMware



Examples


Apps 2
OS 2
ISA 1
Apps 1
OS 1
Apps 1
OS 1
ISA 1
Apps 2
OS 2
Apps 1
OS 1
ISA 1

Examples



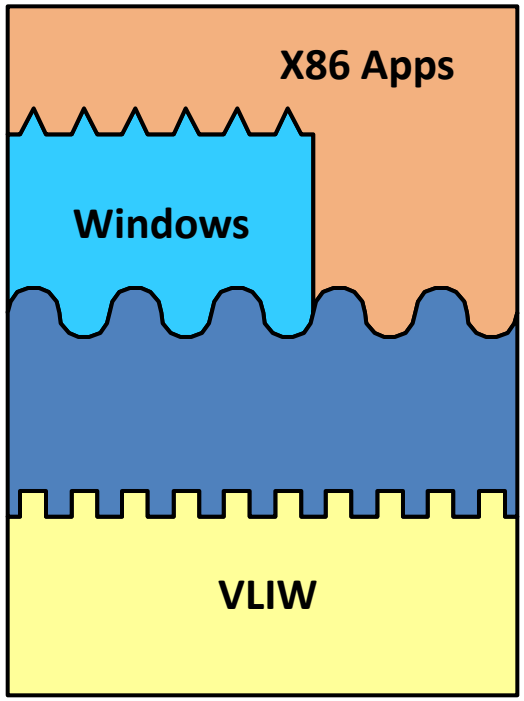
Co-Designed VMs

Different objective:

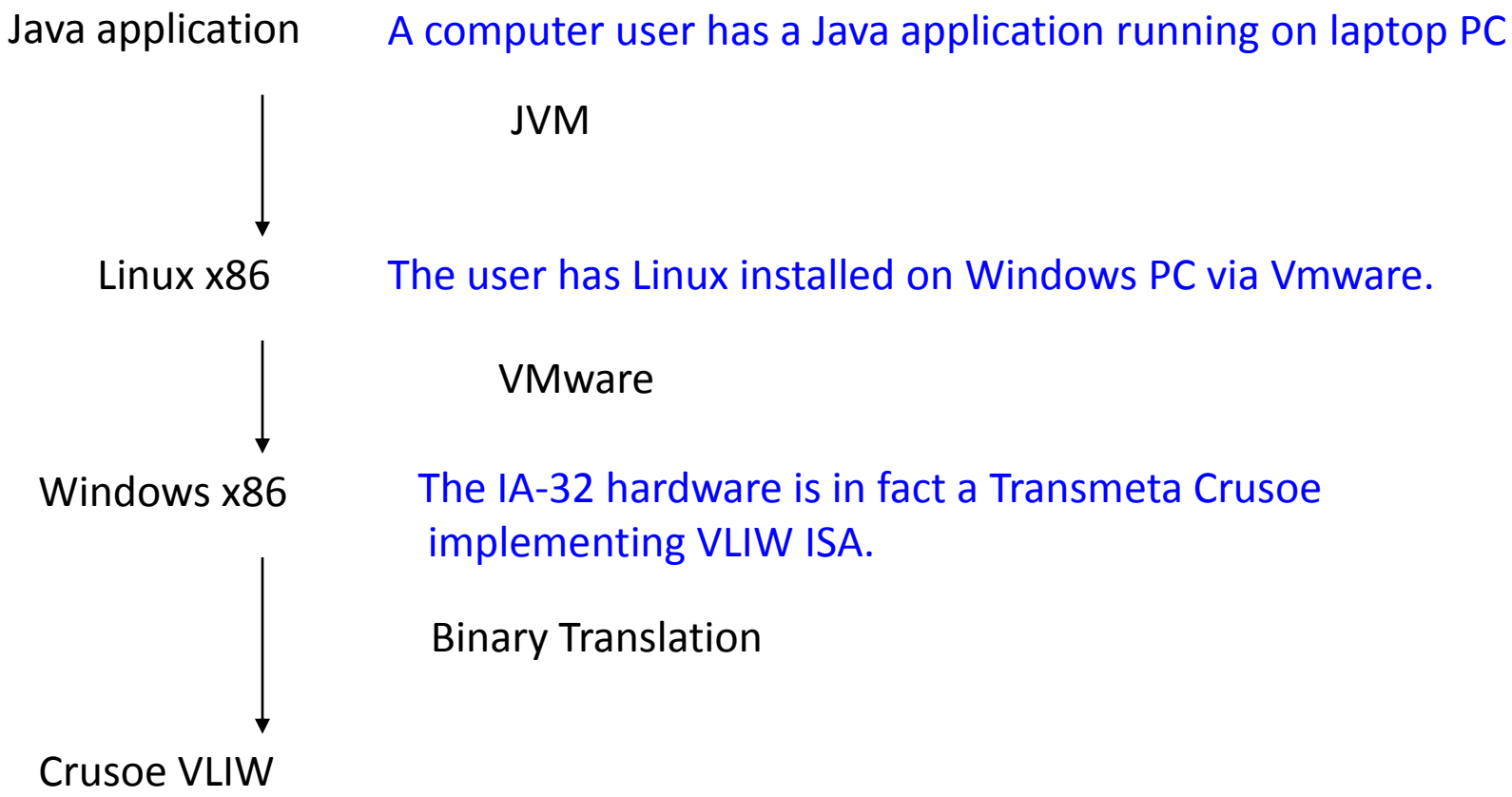
Enable innovative ISA and/or hardware implementation for improved performance and/or power.

Co-Designed VMs

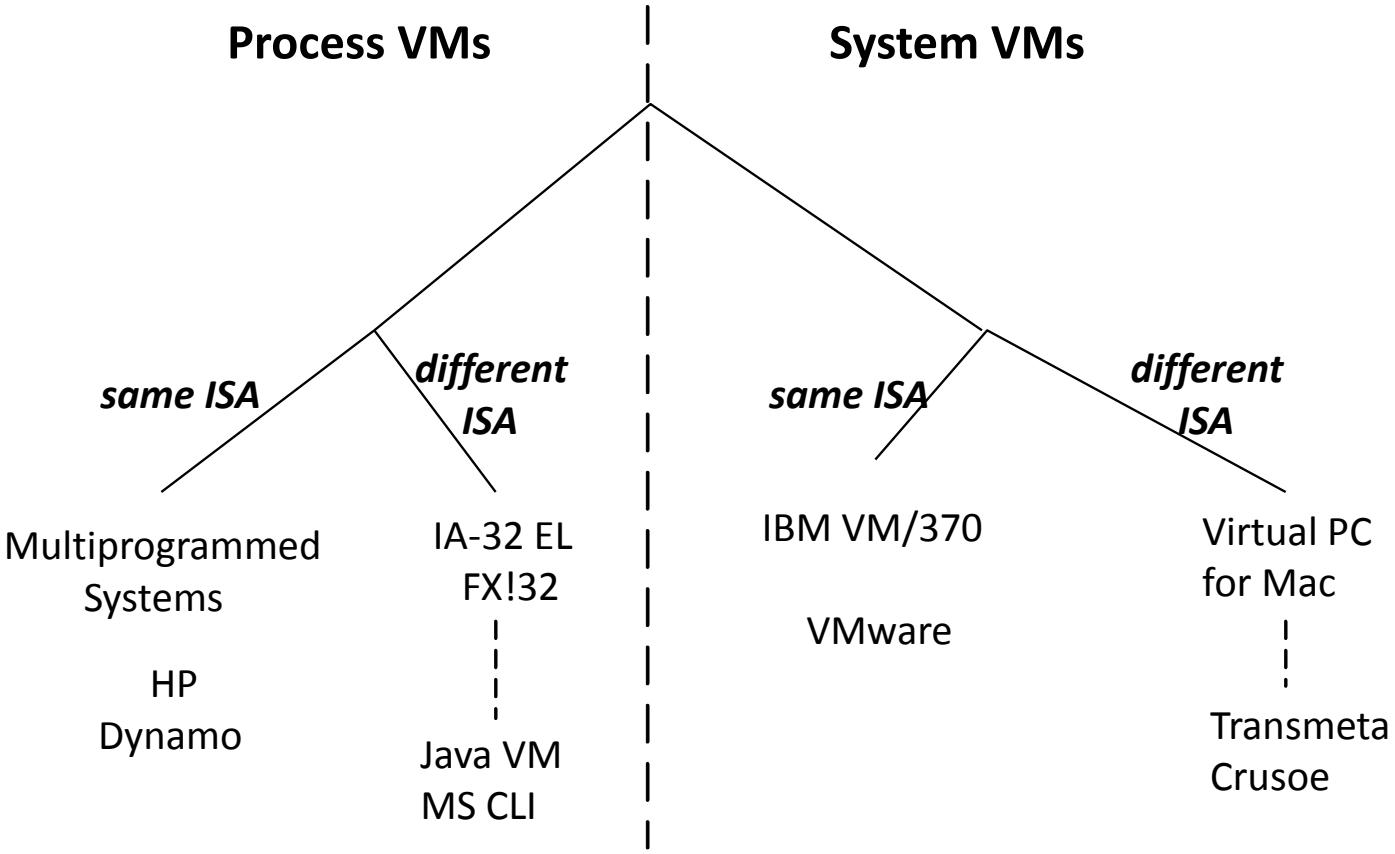
- Perform both translation and optimization
- VM provides interface between standard ISA software and implementation ISA
- Transmeta Crusoe and IBM Daisy best-known examples



Composition: Example



Taxonomy


Process VMs
System VMs
same ISA
different ISA
same ISA
different ISA
Multiprogrammed Systems
HP Dynamo
IA-32 EL
FX!32
Java VM
MS CLI
IBM VM/370
VMware
Virtual PC for Mac
Transmeta Crusoe

Conclusions

- Virtualization will be a key part of future computer systems.
- A fourth major discipline? (with HW, System SW, Application SW)
- VMs have been investigated and built by OS developers, language designers, compiler developers, and hardware designers!
- In this course, we will study the underlying concepts and technologies that are common across the whole spectrum.