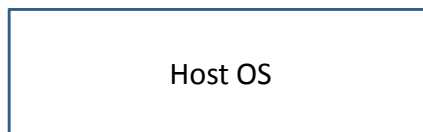
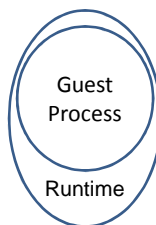


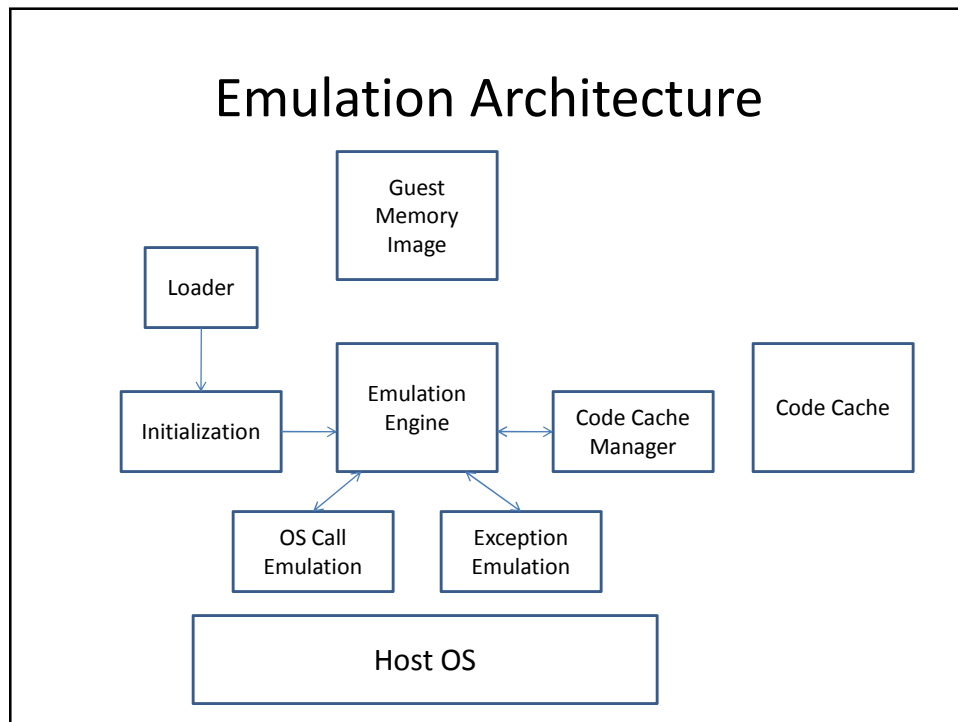
Process Virtual Machines

Introduction

Purpose

- Present the abstraction of a different machine and OS to a *process*.

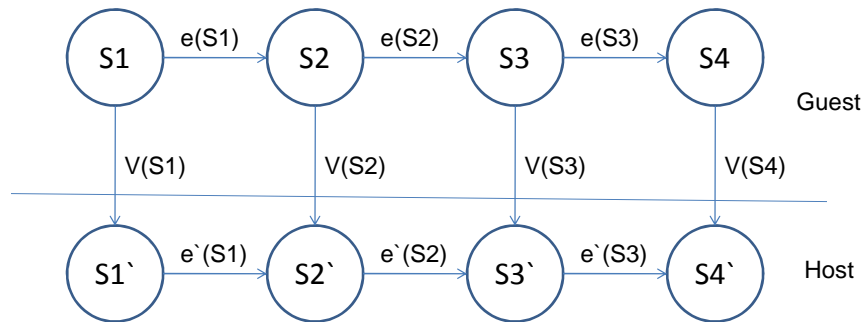




A Note on Compatibility

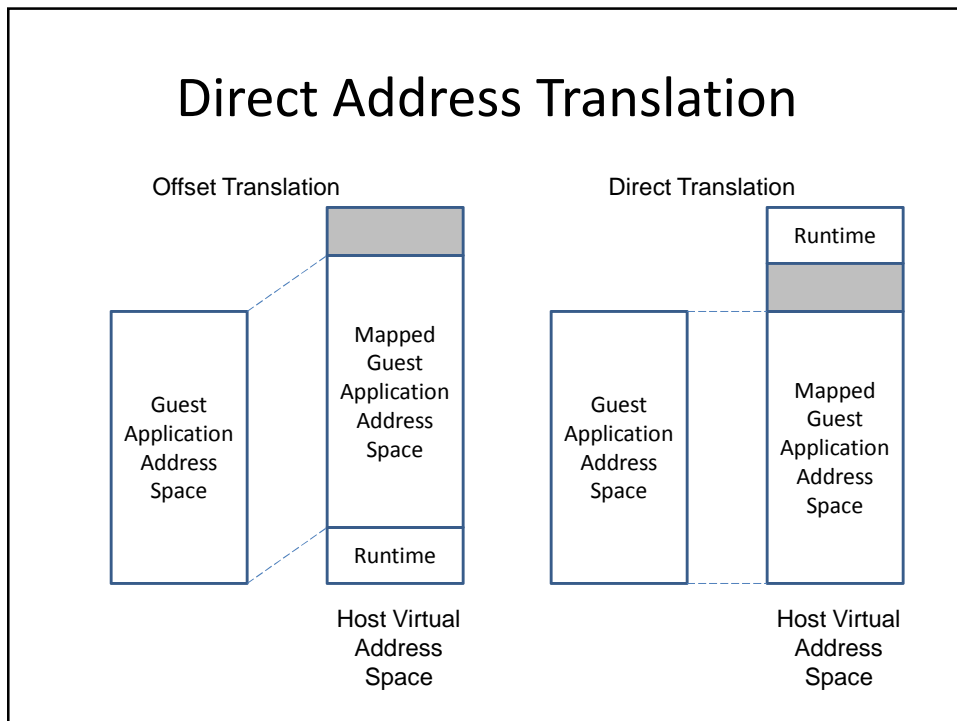
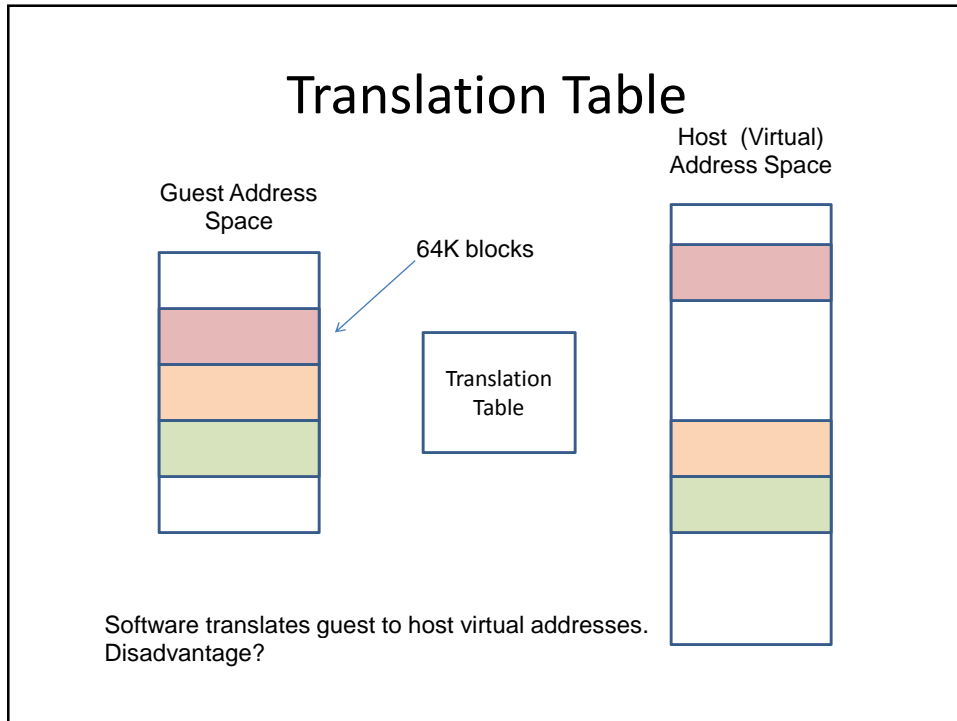
- Process state equivalence at the point of interaction with the “external world”
 - When control transfers from guest process to host OS, state equivalence must hold
 - When control transfers back to guest process, state equivalence must hold (both of user managed and OS managed state)
- Consequences:
 - State does not need to be mapped correctly in between interactions with OS
 - Conservative: why?

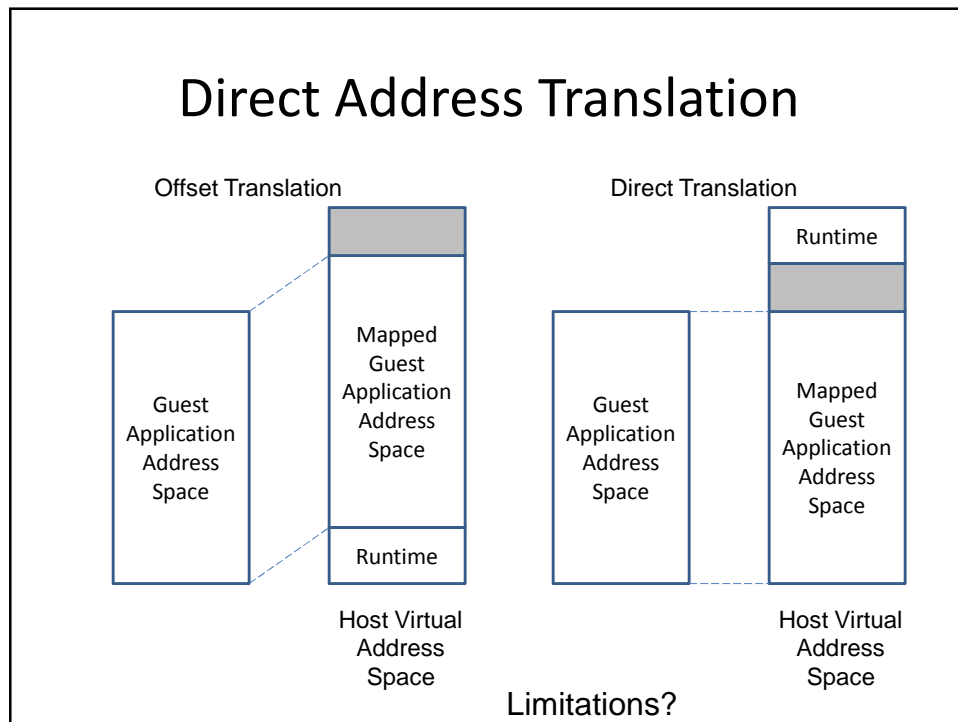
Isomorphism Revisited



State Mapping

- Guest registers → Host registers/Memory
 - Guest context (and context switch)
 - Depends on who has more registers
- Memory address space mapping
 - Guest application (virtual) address space
 - Host application (virtual) address space





Memory Architecture Emulation

- Host OS Offers:
 - A system call to set memory protection (specifies page and access privileges)
 - A signal for a memory protection violation that can be delivered to the application (runtime)
- Memory protection
 - Each page has protection bits such as read/write or read/write/execute (e.g., you cannot execute data, or overwrite code)
 - What if guest architecture has read/write/execute protection whereas host has read/write only?

Page Size Issues

- What if page size on guest is a multiple of page size on host?
- What if page size on host is a multiple of page size on guest?

Page Size Issues

- What if page size on guest is a multiple of page size on host?
 - No problem. Just replicate page protection
- What if page size on host is a multiple of page size on guest?
 - Different guest pages mapped to same host page? Problems?
 - Pad guest pages to size of host page? Problems?

Page Size Issues

- What if page size on host is a multiple of page size on guest?
 - Different guest pages mapped to same host page? Problems?
 - What if pages have different protection?
 - Use the more conservative bits and handle violations accordingly
 - Pad guest pages to size of host page? Problems?
 - Makes address translation more difficult

Instruction Emulation

- Interpretation versus binary translation?
 - Interpretation:
 - no startup overhead
 - High overhead per instruction
 - Binary translation:
 - High startup overhead
 - Low overhead per instruction
 - Can we combine the best of both worlds?

Instruction Emulation

- Interpretation versus binary translation?

- Interpretation:

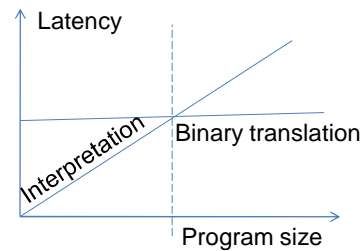
- no startup overhead
- High overhead per instruction

- Binary translation:

- High startup overhead
- Low overhead per instruction

- Can we combine the best of both worlds?

- Small program: Do interpretation
- Large program: Do binary translation



Instruction Emulation

- Initially assume small program
 - Do Interpretation
- Count the number of times each block is executed
- If a block is executed more than N times, do binary translation on this block

Exception Emulation

- Two types of exceptions:
 - Traps (caused by instructions in the program)
 - Interrupts (caused by asynchronous external events)
- For traps:
 - Ensure that all instructions prior to trap have been executed
 - Ensure that none of the instructions after the trap have been executed
- For Interrupts:
 - Emulated code must be in interruptible state

Traps

- How to detect them?
 - Both guest and host support same trap (e.g., page fault). Map guest trap to host trap: capture trap signal, execute the translated guest handler
 - Runtime intercepts all signals and handles them
 - Guest supports trap that host does not support (or does not deliver to the application). Check for trap conditions in the emulated software explicitly

Interrupts

- When an interrupt occurs:
 - Interpretation: When an interrupt occurs, finish interpreting the current instruction and execute the interrupt handler
 - Binary translation: When an interrupt occurs, the emulated code may be in non-interruptible state (what does that mean?)
 - Need well-defined boundaries where emulated code is interruptible.
 - What is a suitable boundary?
 - When interrupt occurs, execute emulate guest code until boundary is reached, then execute the interrupt handler.

Interrupts

- When an interrupt occurs:
 - Interpretation: When an interrupt occurs, finish interpreting the current instruction and execute the interrupt handler
 - Binary translation: When an interrupt occurs, the emulated code may be in non-interruptible state (what does that mean?)
 - Need well-defined boundaries where emulated code is interruptible.
 - What is a suitable boundary? *Block boundaries*
 - When interrupt occurs, execute emulate guest code until boundary is reached, then execute the interrupt handler.

Interrupts

- When an interrupt occurs:
 - Interpretation: When an interrupt occurs, finish interpreting the current instruction and execute the interrupt handler
 - Binary translation: When an interrupt occurs, the emulated code may be in non-interruptible state (what does that mean?)
 - Need well-defined boundaries where emulated code is interruptible.
 - What is a suitable boundary? *Block boundaries* What if blocks are chained?
 - When interrupt occurs, execute emulate guest code until boundary is reached, then execute the interrupt handler.

Interrupts in Binary Translation

- When an interrupt occurs, the emulated code may be in non-interruptible state
 - Determine which block is currently running
 - Unchain the block from the next by replacing the jump at the end of the block to a transfer of control to the emulation manager.
 - Let the block finish
 - Control is transferred to emulation manager which invoked interrupt handler.

A Note on Guest OS Emulation

- Does not have to translate guest OS instructions one a time
 - Translate entire functions into equivalent ones
 - Example: replace a disk I/O system call on the guest with an equivalent disk I/O call on the host
- Not all guest system calls need to be translated to host calls; some are handled by the runtime.
 - Example: Calls installing a new signal handler may be handled by the runtime since runtime intercepts all signals and maintains their handlers.
- Generally an ad hoc process (case-by-case).