ReVirt

Enabling Intrusion Analysis through Virtual Machine Logging and Replay

George Dunlap
Samuel Talmadge King
Sukru Cinar
Murtaza Basrai
Peter M. Chen

University of Michigan

Introduction: Security

- Administrators routinely deal with intrusions
- 'Post-mortem' analysis
  - How the attack worked
  - What they saw, what they changed
- Available data
  - Disk image
  - Security logs
  - Firewall logs

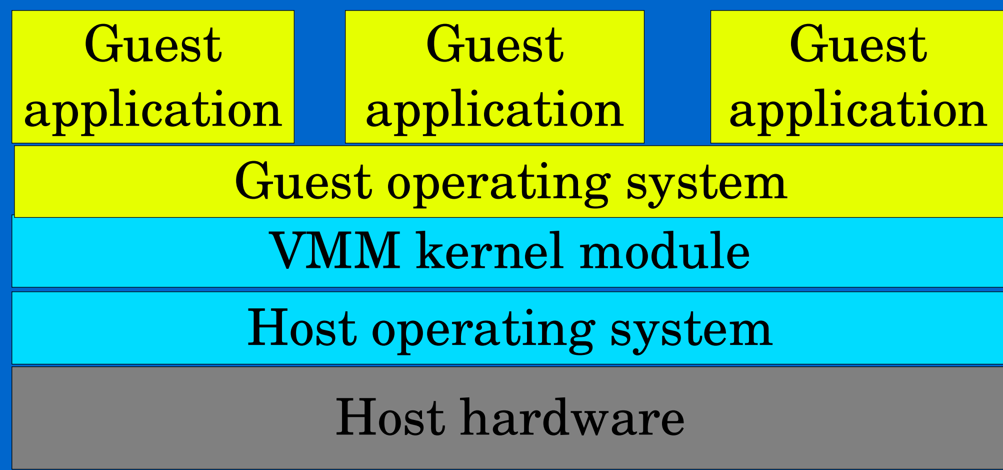
The Weakness of Security Logs

- Integrity
  - Attacker's first move is to subvert the logs
  - Delete or modify, or at least disable
- Completeness
  - Still require lots of educated guesses
  - Can't account for non-determinism
  - Encryption renders even a packet log useless

CoVirt and ReVirt

- The CoVirt project
  - Add security services to virtual machines
- ReVirt
  - Log enough to reconstruct and replay the entire execution of a system
  - Instruction-by-instruction replay of entire virtual machine
  - View the entire state of the system at an arbitrary point in history
  - Watch the execution as it progressed

Virtual Machine Overview

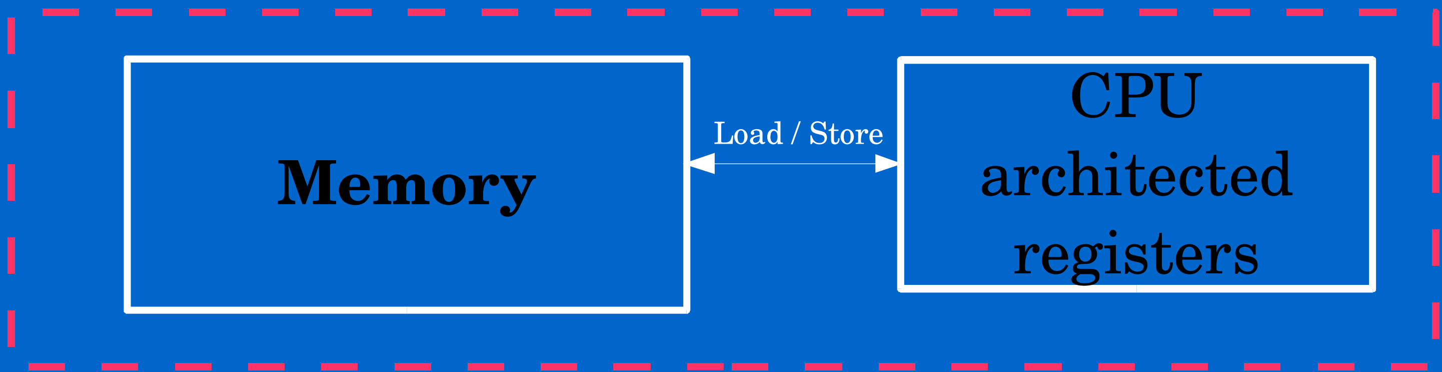


- Virtual machine monitor
  - Current system: “Hosted” VMM architecture
- Security aspects of virtual machines
  - Simpler interface, smaller codebase
  - VMM limits access to host functionality

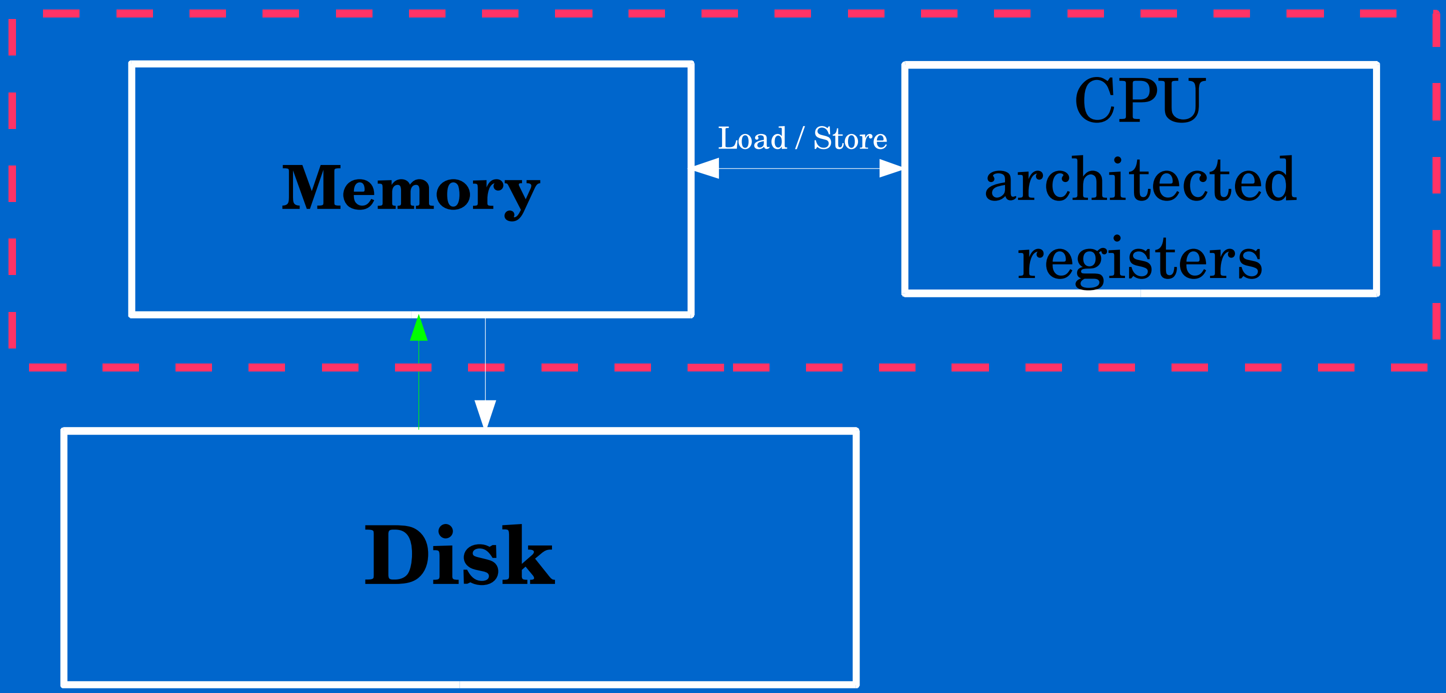
UMLinux: Linux on Linux

- Linux ported to run on 'Linux' architecture
- Guest OS and all applications run within a single host process
- Virtual devices
  - Disk: host raw partition. RTC: gettimeofday().
  - Network: host TUN/TAP.
- Virtual interrupts implemented with signals
  - Timer: SIGALRM. Device I/O: SIGIO.
  - Page fault: SIGSEGV. Syscall(`int80`): SIGUSR1.

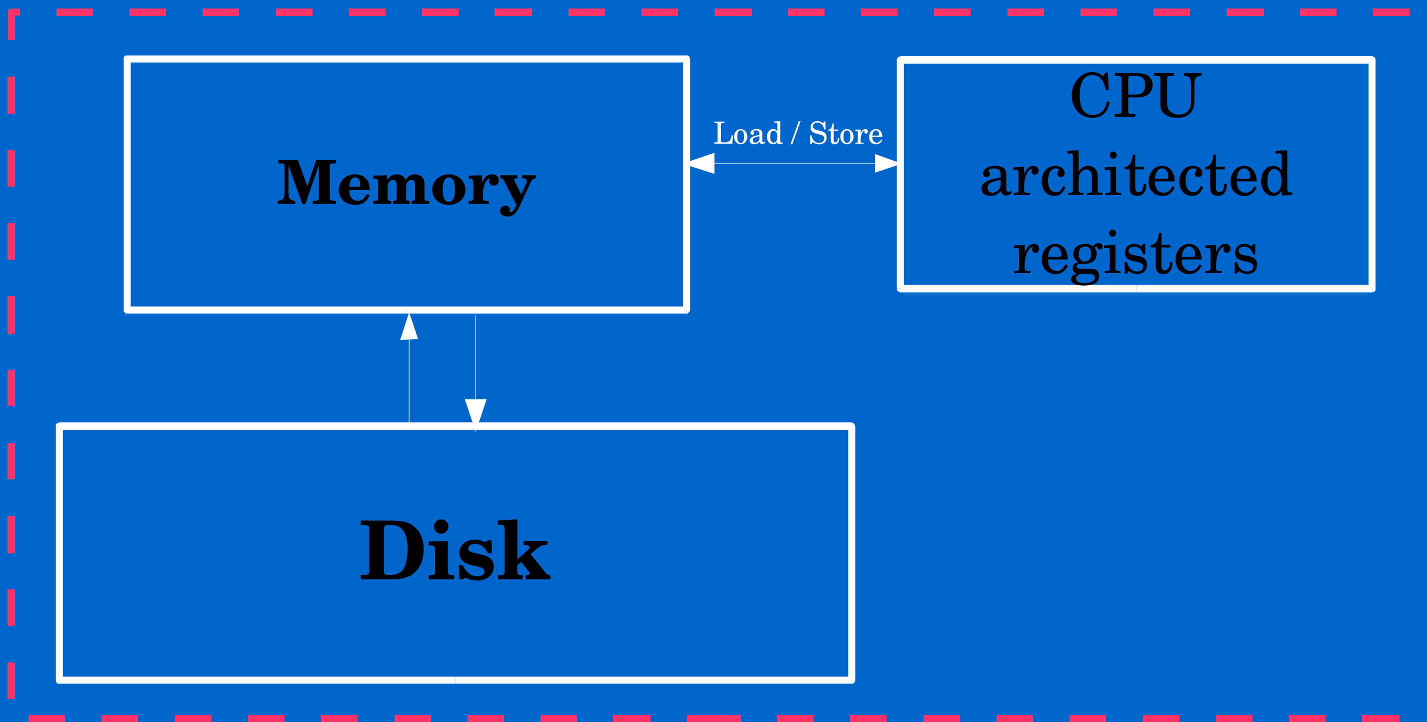
Complete Replay



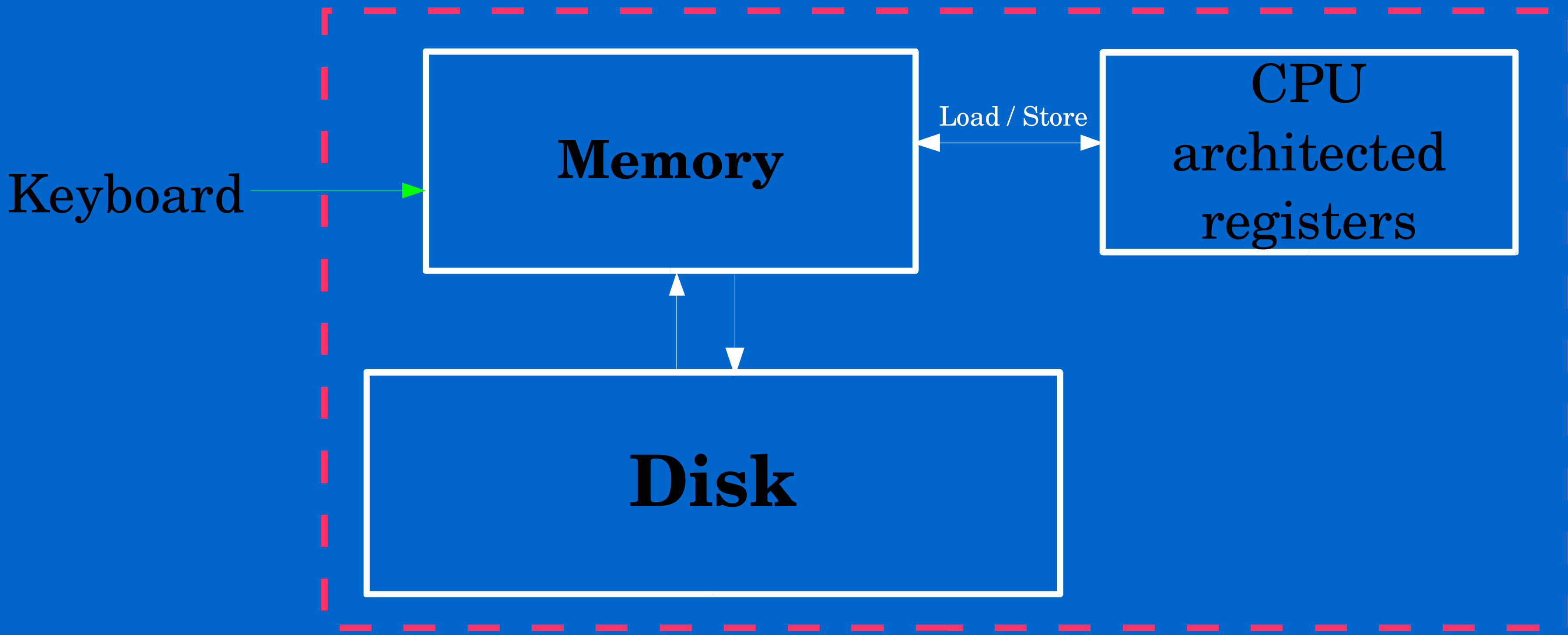
Complete Replay



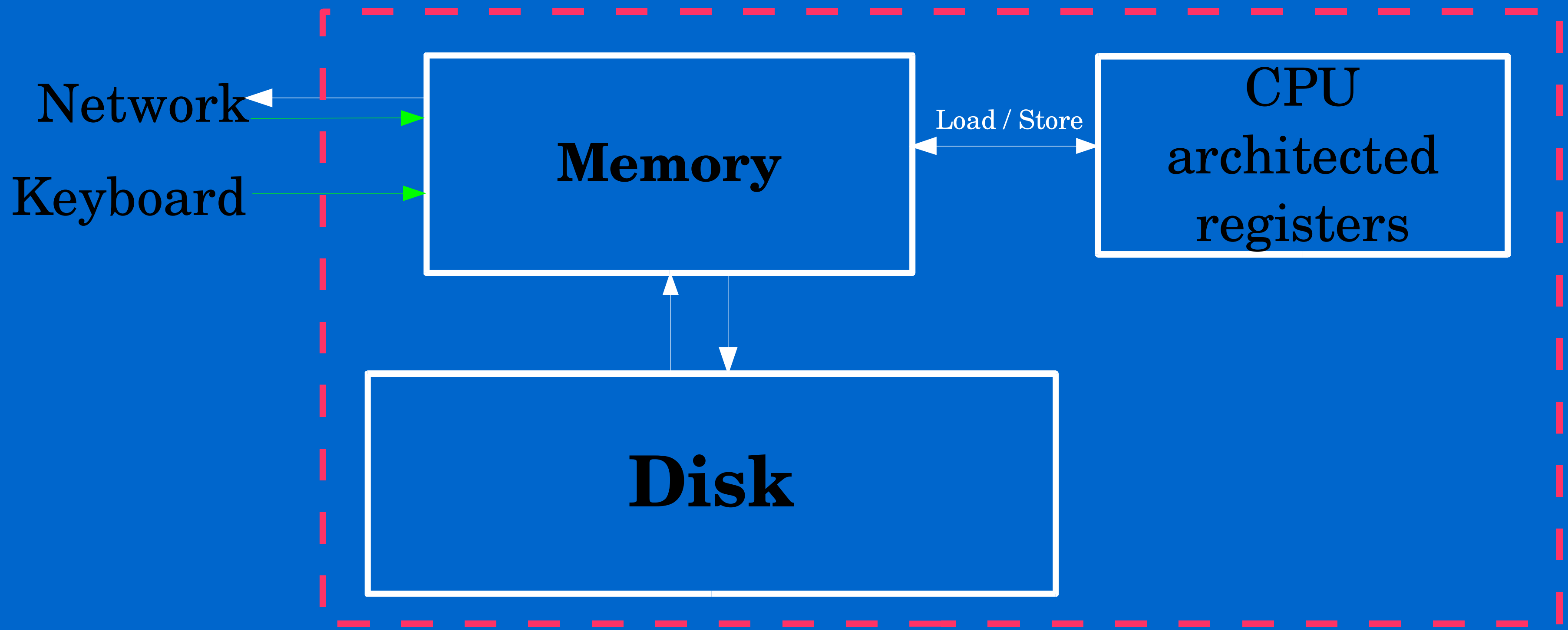
Complete Replay



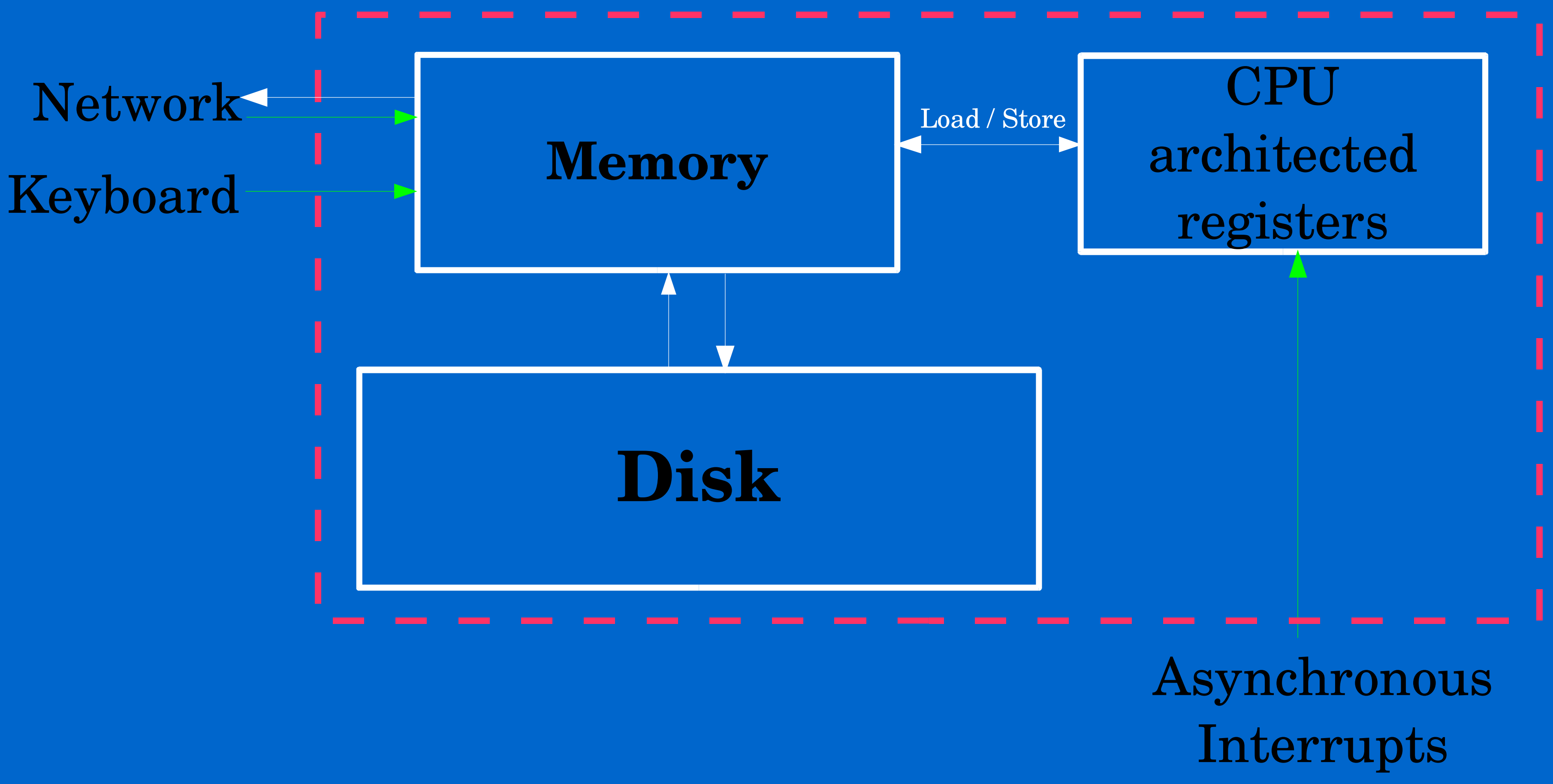
Complete Replay



Complete Replay



Complete Replay



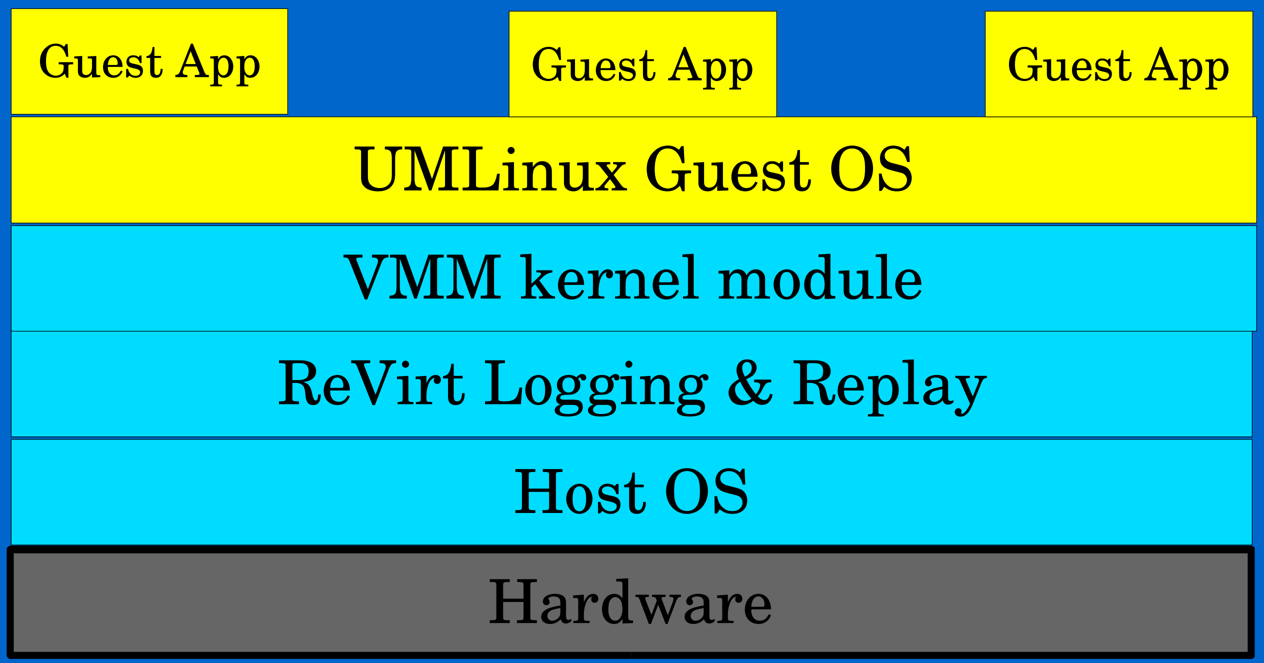
Complete Replay: Summary

- Architecturally visible state transitions
  - Same starting state + same input => same state transitions
- Checkpoint and restore the initial state
- Log when non-deterministic input happens, and make it happen the same way on replay.
  - External data: keyboard, network, external clock
  - Time: when asynchronous interrupts happen

Replaying Interrupts

- Asynchronous virtual interrupts
  - Must be delivered at the exact point in the instruction stream
  - Performance counters available on P4, Athlon
  - (instruction pointer, branch count) unique identifier for an instruction in the stream
  - Before delivering an interrupt, record (`eip,bc`)
  - During replay, deliver at the same (`eip,bc`)

The ReVirt System



- Log syscalls containing external data
  - Give same data during replay
- Deliver virtual interrupts at same point

Details, Details...

- Intel "Repeat String" (`repz`) instructions
  - Log `ecx` register as well
- Hardware performance counters count interrupts
  - Have the OS count interrupts and compensate
- `RDTSC` instruction
  - Disable or emulate with gettimeofday()

Experiment Questions

- How do we know it's doing the same thing?
- What's the overhead of virtualization?
  - Doesn't running in a VM make it too slow?
- What's the overhead of logging?
  - Don't you have to log too much data?
  - Doesn't it slow things down too much?
- How fast can I replay?

Correctness: Sanity Checks

- Output
- System behavior
  - UMLinux makes 14 host system calls regularly
  - Check to see that they're in the same order
- Internal data
  - Compare registers at each system call
- Sparse (`instruction pointer`, `branch count`) space
  - Check (`eip`, `bc`) at each system call
  - Check for (`eip`, `bc`) existence at virtual interrupts

Correctness: Experiments

- Microbenchmark
  - Several guest processes with shared memory, with an explicit race condition
  - Check for same output during replay
- Macrobenchmark
  - Boot, start Gnome session, two concurrent builds over NFS, surf the web simultaneously
  - 15,000,000 host system calls
  - 55,000 virtual interrupts

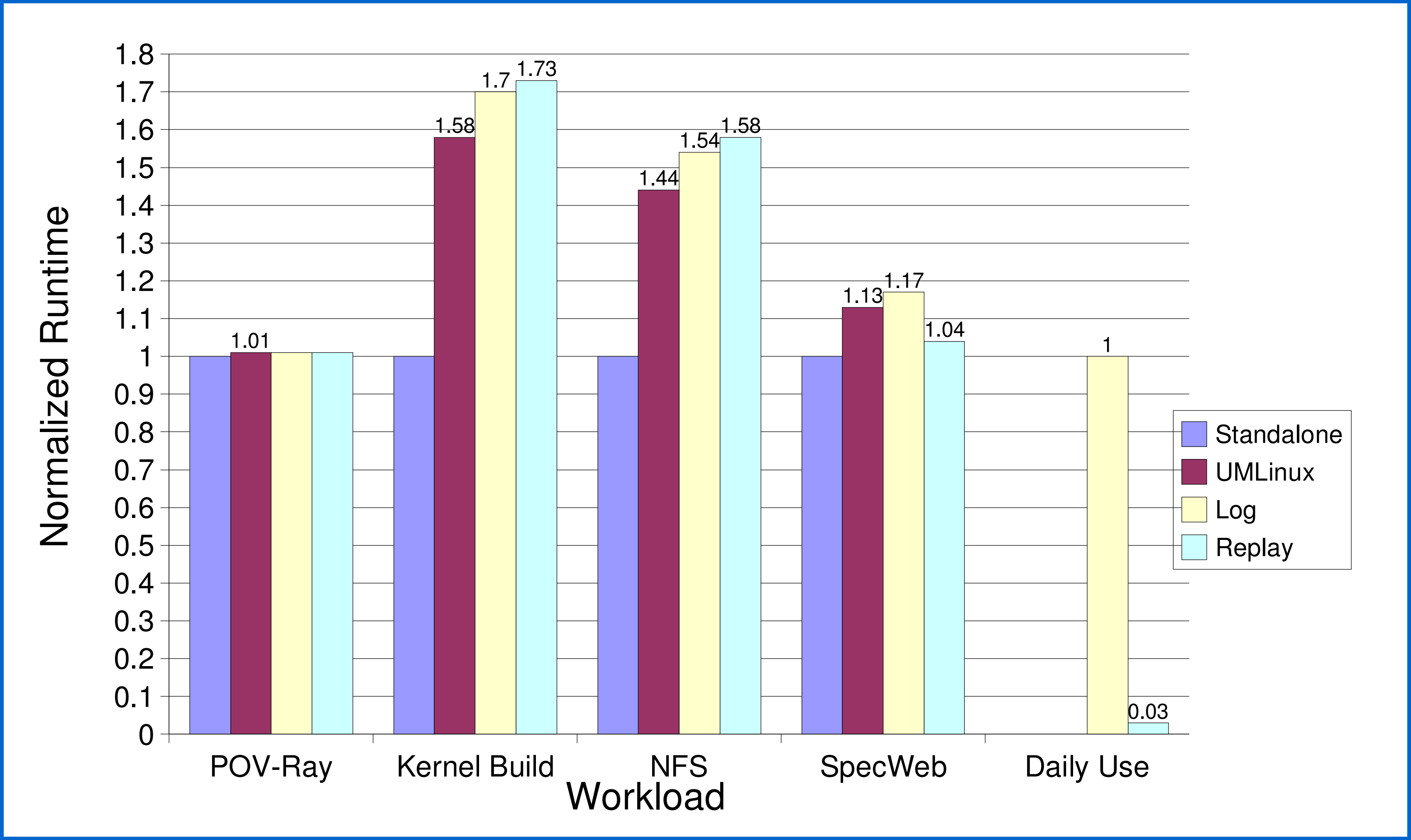
Experiments: Performance Setup

- AMD Athlon 1800+
- Samsung SV4084 IDE Disk
- Linux 2.4.18 guest / host / standalone kernel
- Redhat 6.2 install for guest / standalone system
- Standalone: 256MB
- ReVirt: Host total 256MB, Guest 192MB
  - Factor in memory overhead of virtualization
- Virtual HD on a raw partition to avoid host caching effects

Experiments: Workload

- POV-Ray raytracer
  - CPU-bound, few processes, little disk I/O
- Kernel build: 2.4.18 stock kernel
- NFS kernel build
  - Warm cache numbers reported
- SPEC Web 99
  - Apache 2.0.36; 2 clients, 15 simultaneous connections
- Daily use test: 24 hrs

Performance Results



Log Size

Workload	Compressed log growth rate	Time to fill a 100 GB disk
POV-Ray	0.04 GB/Day	7.4 years
Kernel-build	0.08 GB/Day	3.4 years
NFS kernel-build	1.2 GB/Day	2.9 months
SPECweb99	1.4 GB/Day	2.4 months
Daily use	0.2 GB/Day	1.5 years

Analysis

- Can roll back to any arbitrary point in the attack
- Look in from outside
  - Complete memory & disk state
- Look from inside
  - Start the VM running from an arbitrary point
  - Log in to system and look around

Future Work

- Analysis tools
- Checkpointing a “live” system
- More creative uses of replay
  - Partial replay & continue
  - Hypothesis testing
  - Binary search
- Cooperative logging
  - Extend “the box” to other logged systems

Conclusions

- Current logging systems lack integrity and completeness
- CoVirt enhances integrity by moving services beneath a virtual machine
- ReVirt adds completeness by allowing complete replay of a VM
- Virtualization & logging adds 1-70% overhead
- A single disk can store a log for several months

Questions

Trusted Computing Base

- TCB of current research implementation
  - VMM, host kernel, X server
  - Guest OS use of host kernel limited
    - UMLinux given access to only one host file
    - Can't interact directly with other host programs
- Other possible implementations
  - VMWare ESX Server
  - Microkernel, exokernel
  - Denali

Related Work

- Hypervisor
  - Many similar techniques and ideas, different goals
  - Hypervisor duplicates input and throws it away
  - We log input and replay it later
- S4: Self-Securing Storage
  - Complete log of disk states

Issues: Removable Media

- Solution 1: Log it as an external data source
  - CDs: ~700 MB
    - One per hour = 17GB/day
    - 6 days to fill 100GB even uncompressed
  - DVDs: up to 87GB?
- Solution 2: Jukebox
  - Bring “inside the box”, don't need to log...
  - ...but can't change
- Solution 3: Require user to re-insert media

Issues: Log Flooding

- What if you run out of space for the log?
  - Must stop the system or abandon replay
  - Turns break-in into DoS attack
    - Can still see what the attacker did
  - Attacker has no direct control over log
    - Network data most likely way of flooding log
    - Noticeable

Technical Stuff

- Micro-architectural non-determinism
  - Only care about architecturally visible state
  - Architecturally in-visible state:
    - Branch prediction
    - Cache misses
    - Out-of-order execution
- Memory: DMA, Alpha prefetching & reordering
  - Don't allow DMA to guest OS
  - Don't allow access to mmio (pre-fetched reads)

Shared-Memory Multiprocessors

- True SMM is very hard
  - Log/Replay memory write/read interleaving?
  - We know of no good way to do this
- Disco
  - Ran non-SMM kernels on SMM
  - Takes partial advantage of SMM hardware

Performance Counters

- Are the performance counters accurate?
  - We don't need correctness, only consistency
  - (eip,bc) cross-checking: one or the other is wrong
- Don't they count interrupts, which are non-det?
  - AMD: interrupts; P4s: iret instructions
  - Kernel sees most interrupts
    - SMM
  - Compensate for non-deterministic events