The Quest-V Separation Kernel for Mixed Criticality Systems

Richard West
richwest@cs.bu.edu

Ye Li, Eric Missimer
{liye, missimer}@cs.bu.edu

Computer Science

Background

- Multi- / many-core processors increasingly popular in embedded systems
- Many now feature hardware virtualization
  - ARM Cortex A15, Intel VT-x, AMD-V
- H/W Virtualization provides opportunity to partition resources amongst guest VMs

H/W Virtualization + Resource Partitioning = Platform for Mixed Criticality Systems

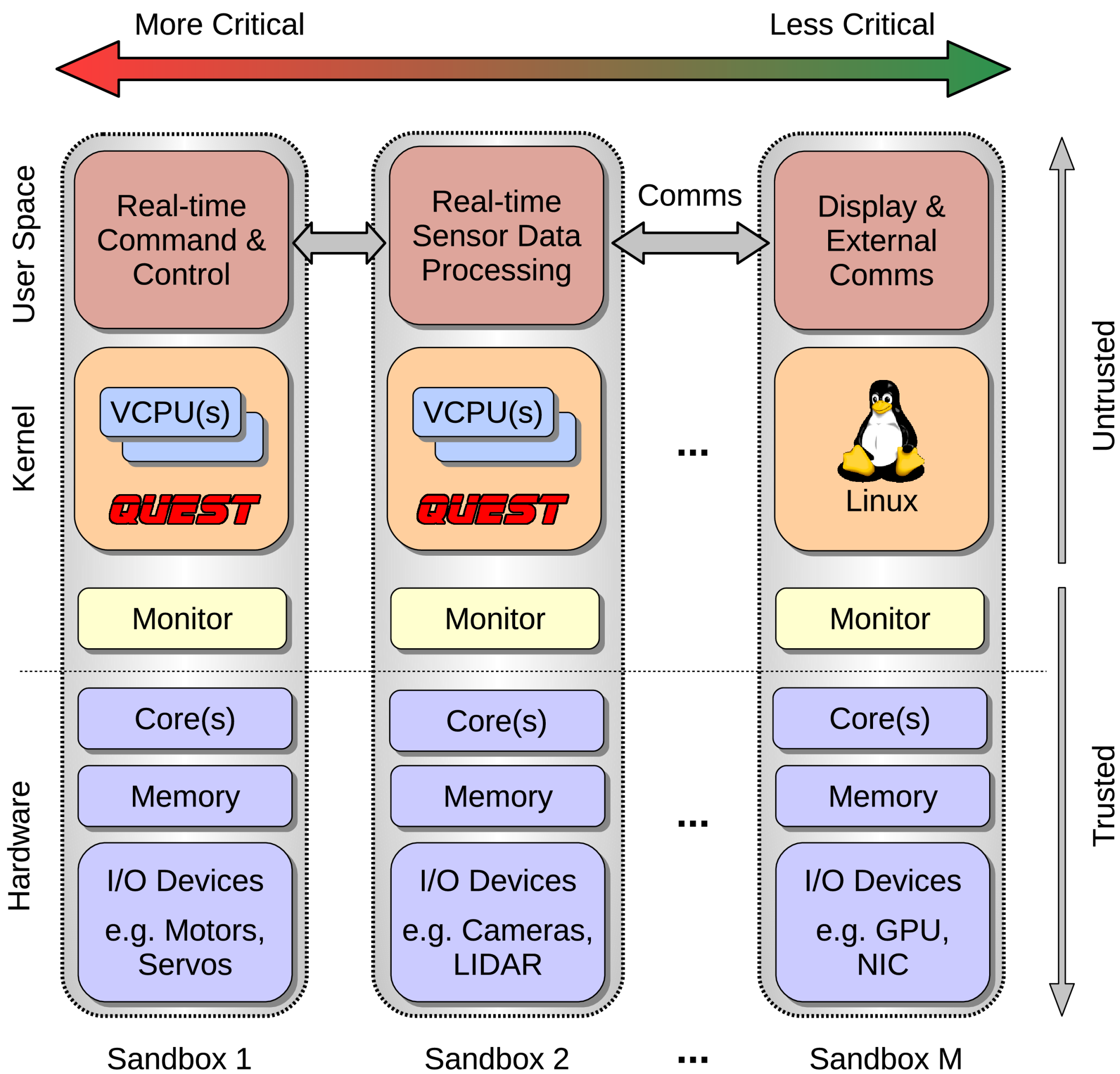
Problem

- Traditional Virtual Machine approaches too expensive
  - Require traps to VMM (a.k.a. hypervisor) to mux & manage machine resources for multiple guests
  - e.g., ~1500 clock cycles VM-Enter/Exit on Xeon E5506

Contributions

- Quest-V Separation Kernel
  - Uses H/W virtualization to partition resources amongst services of different criticalities
  - Each partition, or **sandbox**, manages its own CPU cores, memory area, and I/O devices w/o hypervisor intervention
  - Hypervisor typically only needed for bootstrapping system + managing comms channels b/w sandboxes

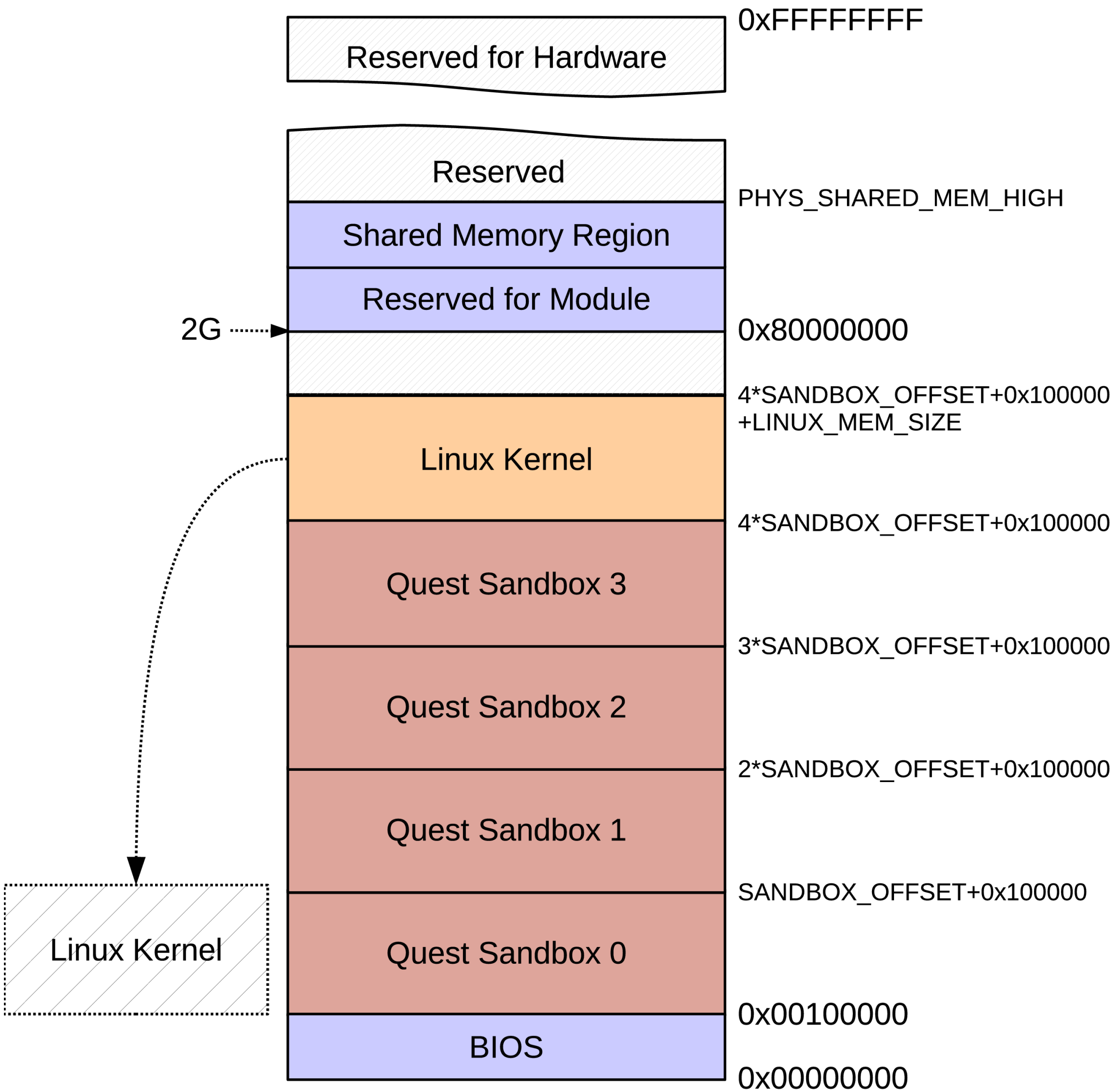
Architecture Overview



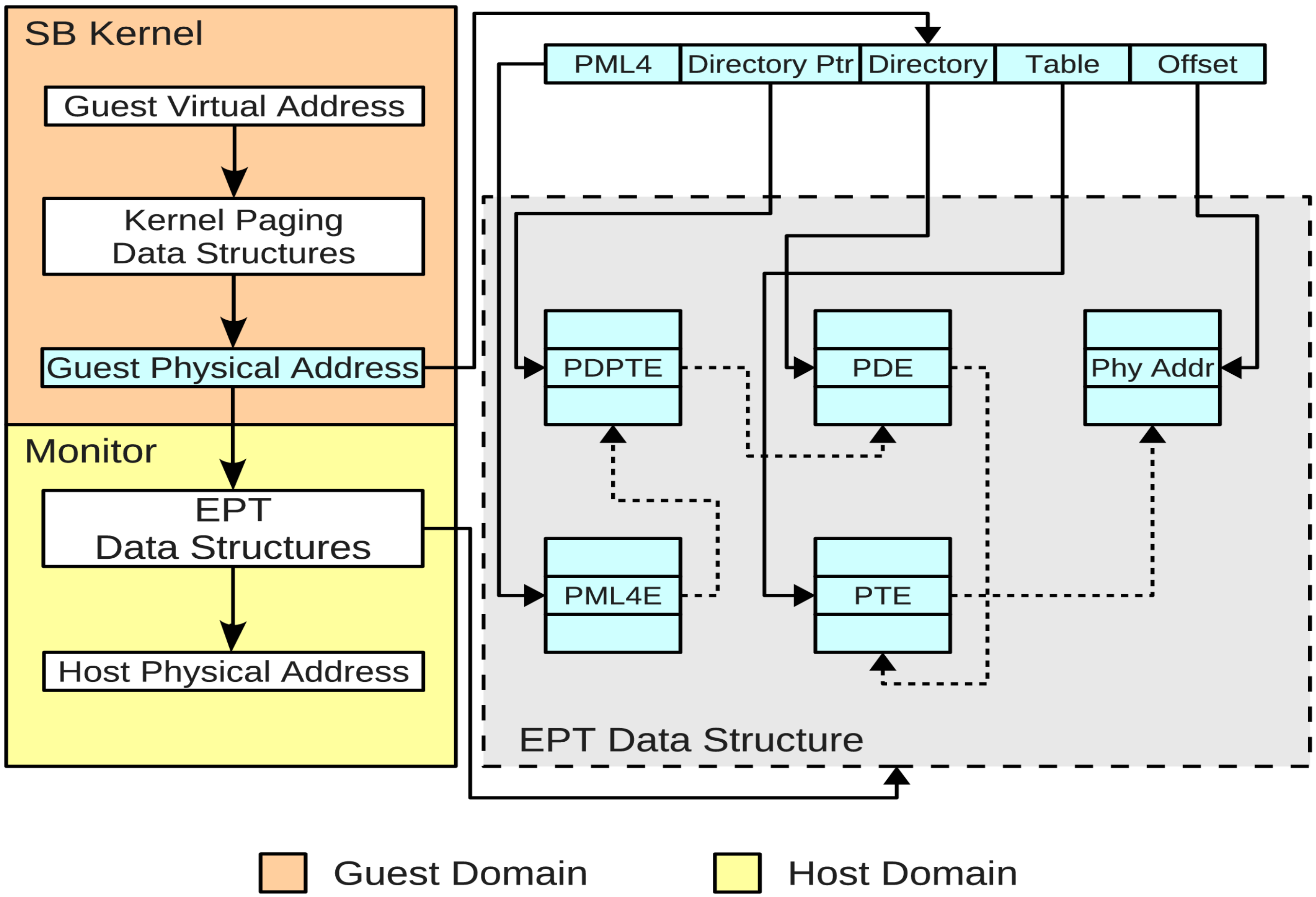
Memory Partitioning

- Guest kernel page tables for GVA-to-GPA translation
- EPTs (a.k.a. shadow page tables) for GPA-to-HPA translation
  - EPTs modifiable only by monitors
  -
  - Intel VT-x: 1GB address spaces require 12KB EPTs w/ 2MB superpaging

Quest-V Linux Memory Layout



Quest-V Memory Partitioning

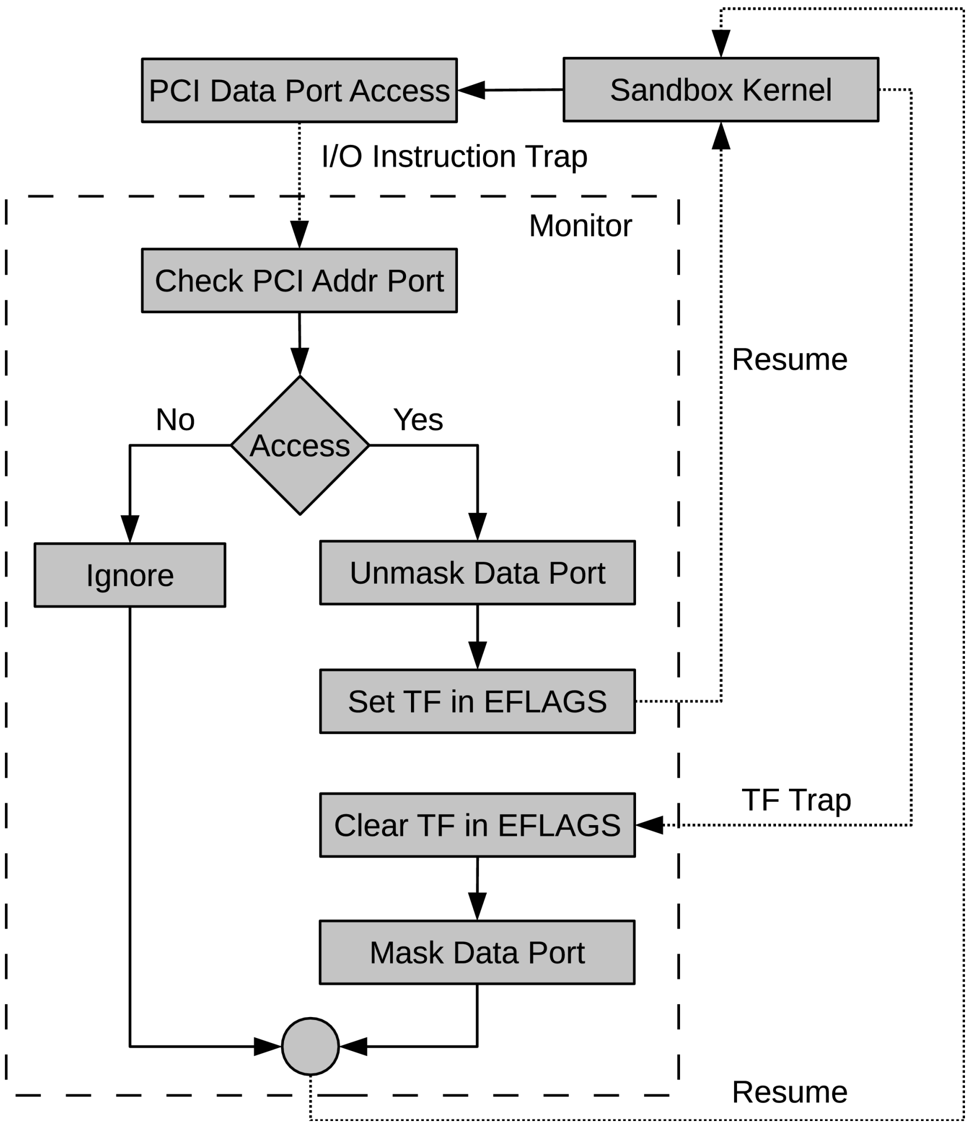


I/O Partitioning

- Device interrupts directed to each sandbox
  - Use I/O APIC redirection tables
  - Eliminates monitor from control path
- EPTs prevent unauthorized updates to I/O APIC memory area by guest kernels

- Port-addressed devices use in/out instructions
- VMCS configured to cause monitor trap for specific port addresses
- Monitor maintains device "blacklist" for each sandbox
  - DeviceID + VendorID of restricted PCI devices

Quest-V I/O Partitioning



CPU Partitioning

- Scheduling local to each sandbox
  - partitioned rather than global
  - avoids monitor intervention

- Uses VCPU approach for Quest native kernels (real-time)

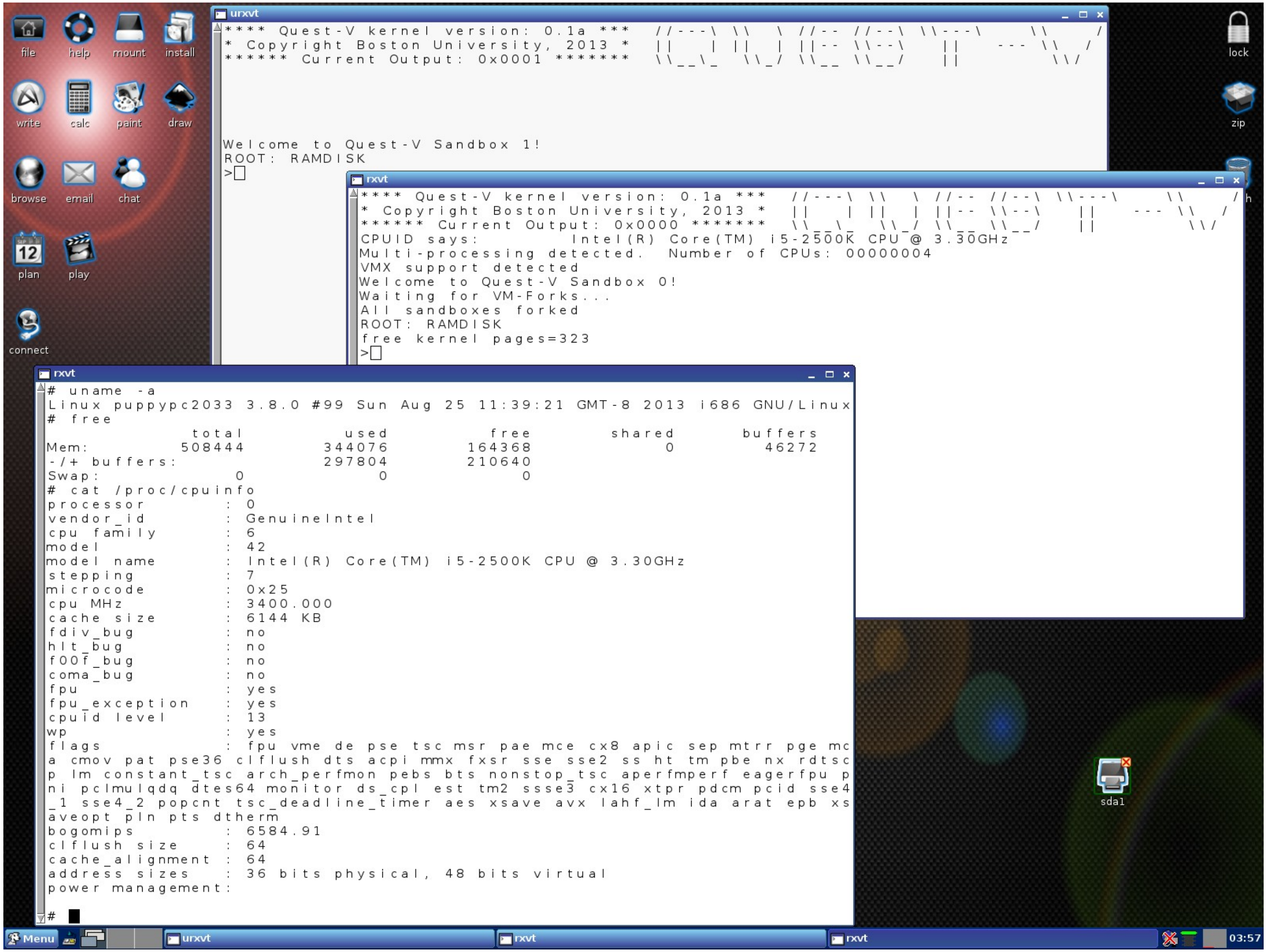
Cache Partitioning

- Shared caches controlled using color-aware memory allocator
- Cache occupancy prediction based on h/w performance counters
  - $E' = (1-E/C) * m_l - E/C * m_o$

Linux Front End

- For low criticality legacy services
- Based on Puppy Linux 3.8.0
- Runs entirely out of RAM including root filesystem
- Low-cost paravirtualization
  - less than 100 lines
  - Restrict observable memory
  - Adjust DMA offsets
- Grant access to VGA framebuffer + GPU
- Quest native SBs tunnel terminal I/O to Linux via shared memory using special drivers

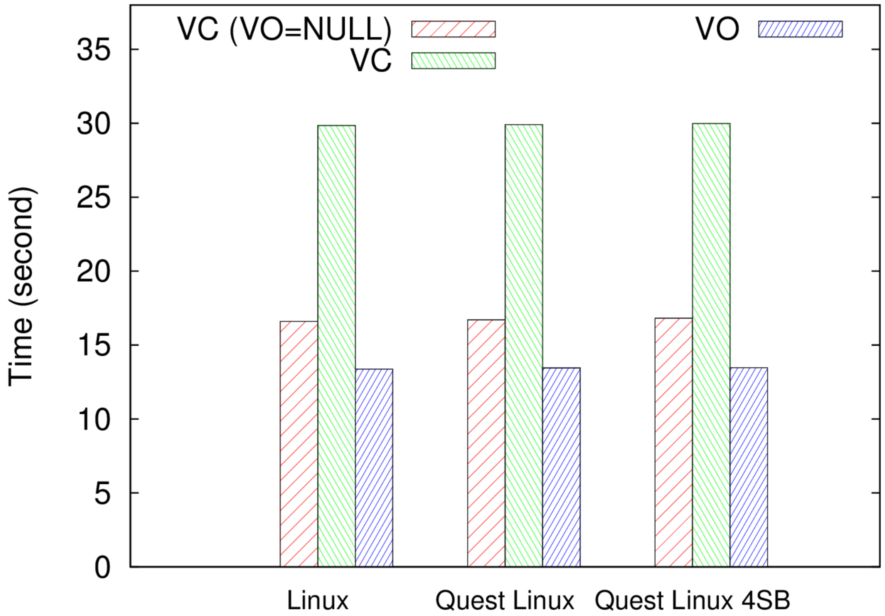
Quest-V Linux Screenshot



Quest-V Performance Overhead

- Measured time to play back 1080P MPEG2 video from the x264 HD video benchmark
- Mini-ITX Intel Core i5-2500K 4-core, HD3000 graphics, 4GB RAM

mplayer Benchmark



Conclusions

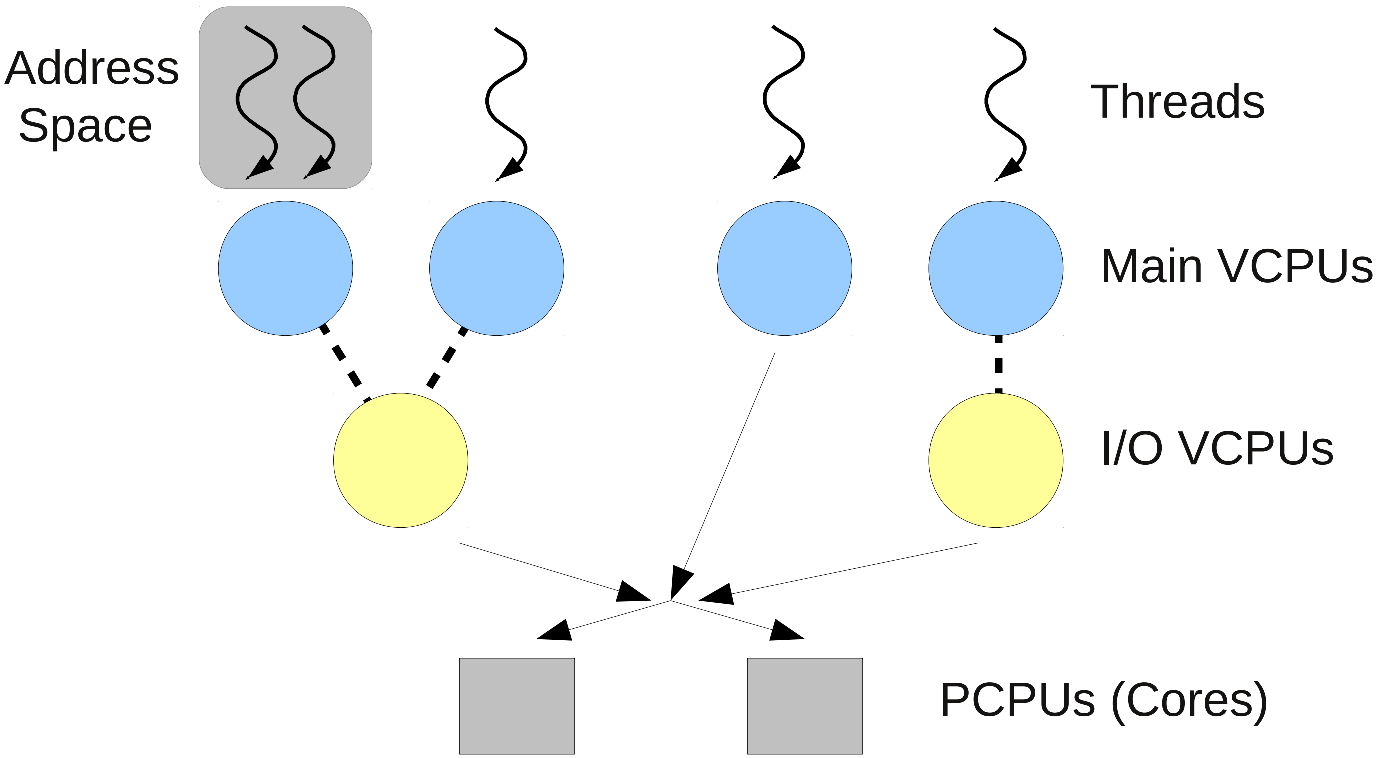
- Quest-V separation kernel built from scratch
  - Distributed system on a chip
  - Uses (optional) h/w virtualization to partition resources into sandboxes
  - Protected comms channels b/w sandboxes
- Sandboxes can have different criticalities
  - Linux front-end for less critical legacy services
- Sandboxes responsible for local resource management
  - avoids monitor involvement

See: www.questos.org for more details

Future Work

- Online fault detection and recovery
- Technologies for secure monitors
  - e.g., Intel TXT

VCPUs in Quest(-V)



Predictability

- VCPUs for budgeted real-time execution of threads and system events (e.g., interrupts)
  - Threads mapped to VCPUs
  - VCPUs mapped to physical cores
- Sandbox kernels perform local scheduling on assigned cores
  - Avoid VM-Exits to Monitor – eliminate cache/TLB flushes

Memory Virtualization Costs

- Example Data TLB overheads
- Xeon E5506 4-core @ 2.13GHz, 4GB RAM