The Quest-V Separation Kernel

Richard West
richwest@cs.bu.edu

Computer Science

Goals

- Develop system for high-confidence (embedded) systems
  - Mixed criticalities (timeliness and safety)

- Predictable – real-time support
- Resistant to component failures & malicious manipulation
- Self-healing
- Online recovery of software failures

Target Applications

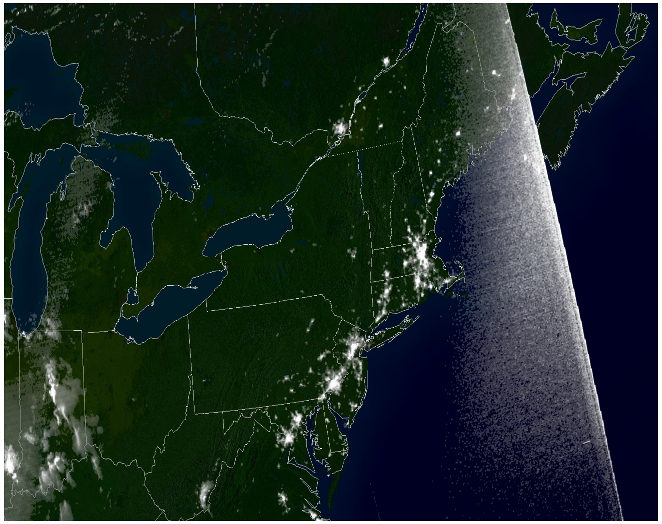
- Healthcare
- Avionics
- Automotive
- Factory automation
- Robotics
- Space exploration
- Other safety-critical domains

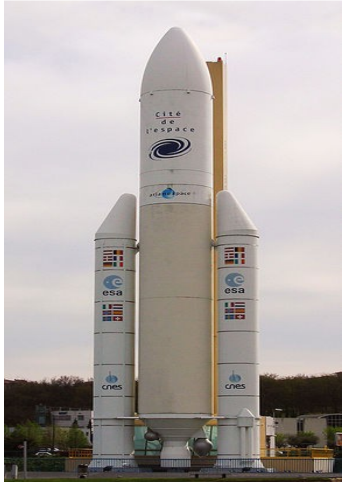





Case Studies

- $327 million Mars Climate Orbiter
    - Loss of spacecraft due to Imperial / Metric conversion error (September 23, 1999)
- 10 yrs & $7 billion to develop Ariane 5 rocket
    - June 4, 1996 rocket destroyed during flight
    - Conversion error from 64-bit double to 16-bit value
- 50+ million people in 8 states & Canada in 2003 without electricity due to software race condition





Approach

- Quest-V for multi-/many-core processors
  - Distributed system on a chip
  - Time as a first-class resource
    - Cycle-accurate time accountability
  - Separate sandbox kernels for system components
  - Memory isolation using h/w-assisted memory virtualization
    - Extended page tables (EPTs – Intel)
    - Nested page tables (NPTs – AMD)
  - Also need CPU, I/O, cache isolation, etc (later!)

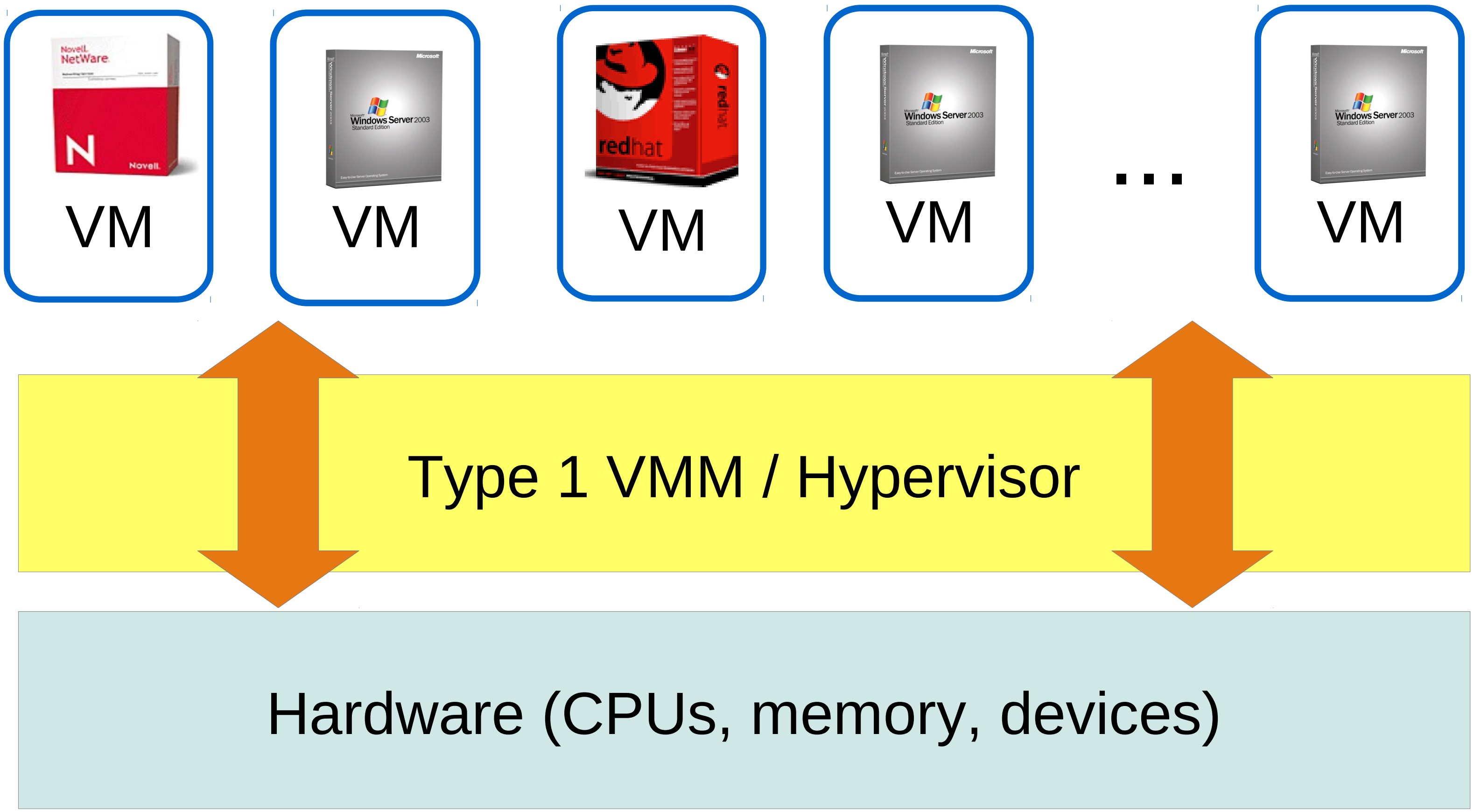
Related Work

- Existing virtualized solutions for resource partitioning
  - Wind River Hypervisor, XtratuM, PikeOS, Mentor Graphics Hypervisor
  - Xen, Oracle PDOMs, IBM LPARs
  - Muen, (Siemens) Jailhouse

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

Traditional Approach (Type 1 VMM)



Contributions

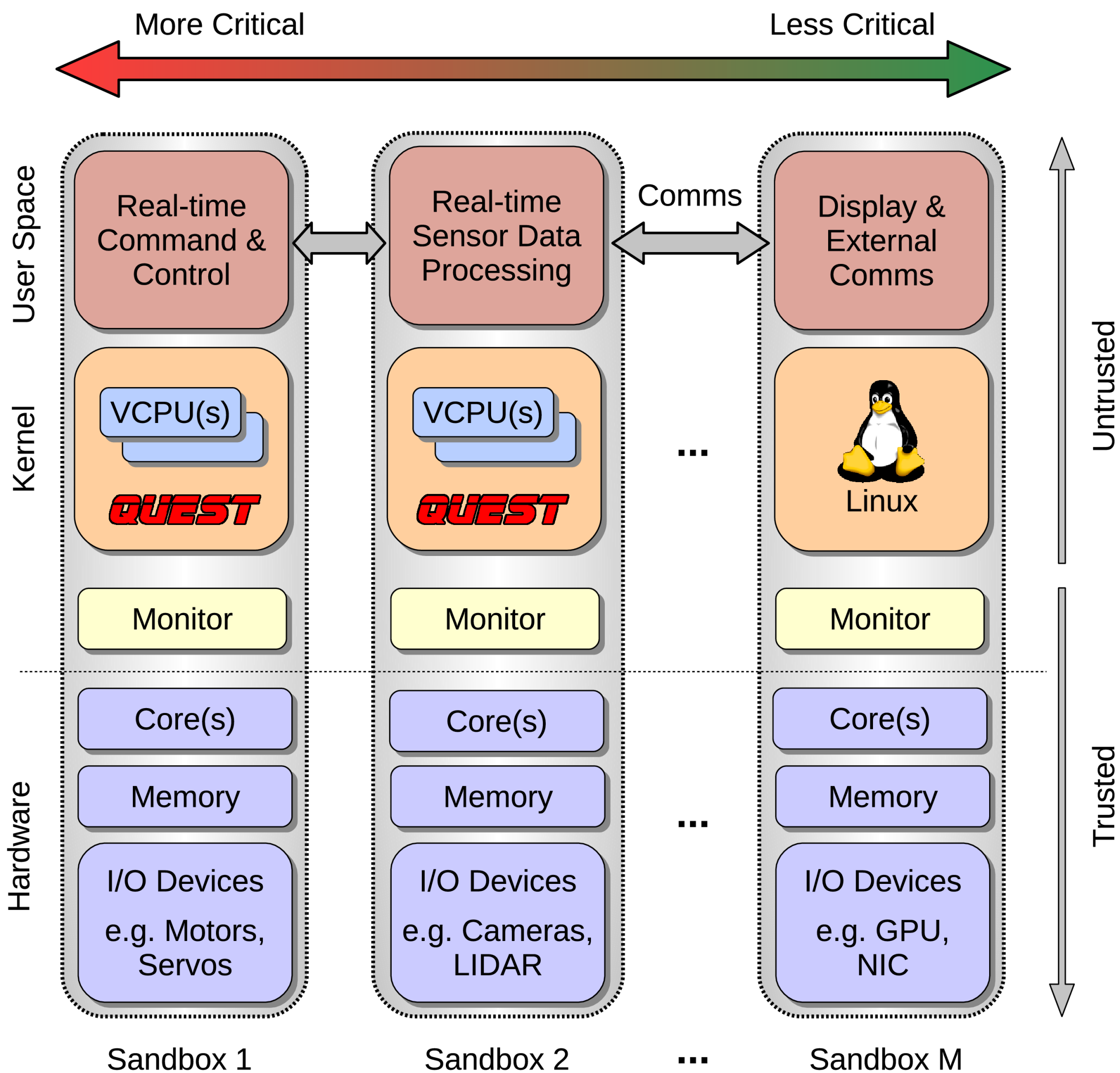
- Quest-V Separation Kernel **[WMC'13, VEE'14]**
  - Uses H/W virtualization to partition resources amongst services of different criticalities
  - Each partition, or **sandbox**, manages its own CPU cores, memory area, and I/O devices w/o hypervisor intervention
  - Hypervisor typically only needed for bootstrapping system + managing comms channels b/w sandboxes

Contributions

- Quest-V Separation Kernel

Eliminates hypervisor intervention during normal virtual machine operations

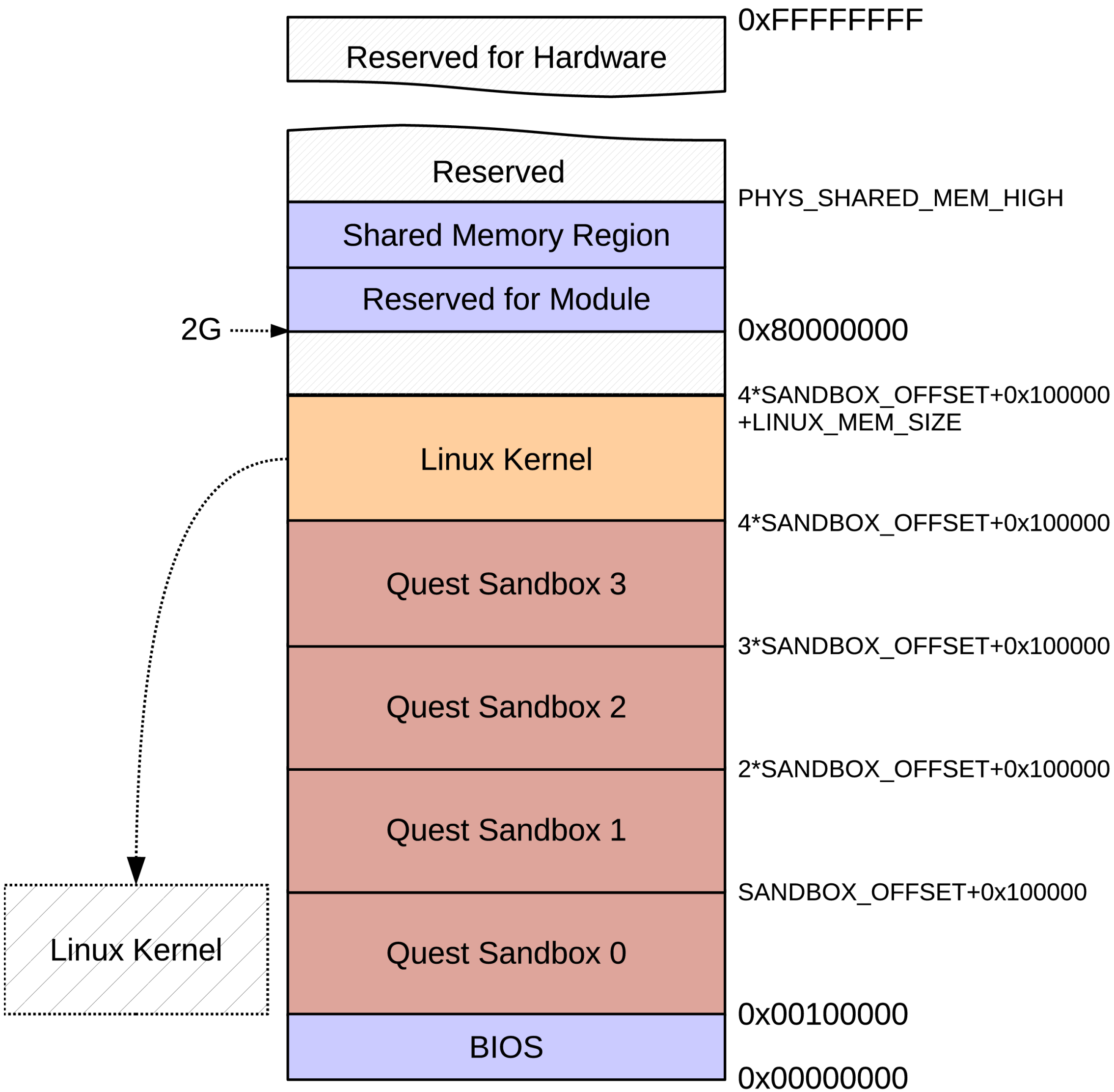
Architecture Overview



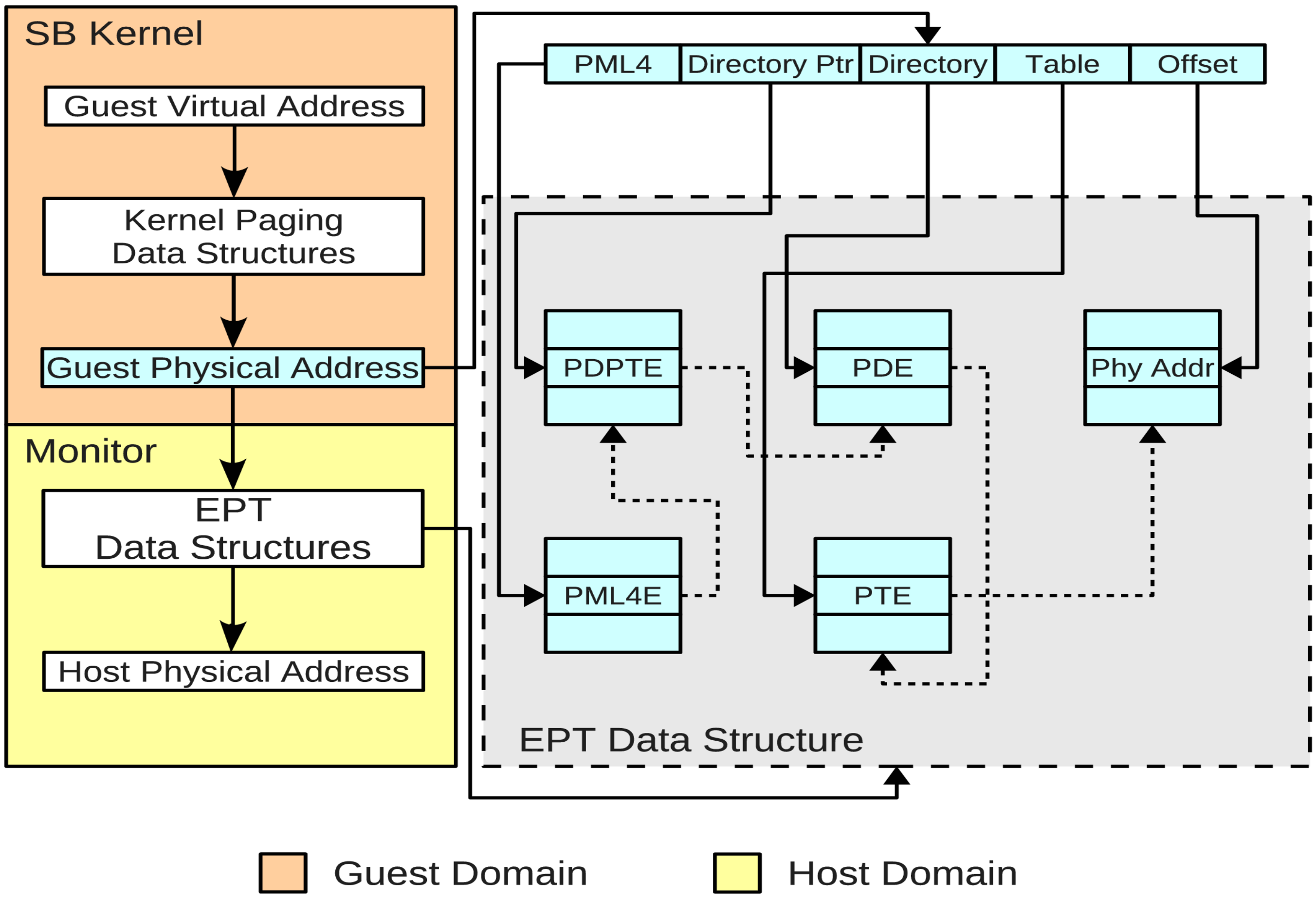
Memory Partitioning

- 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

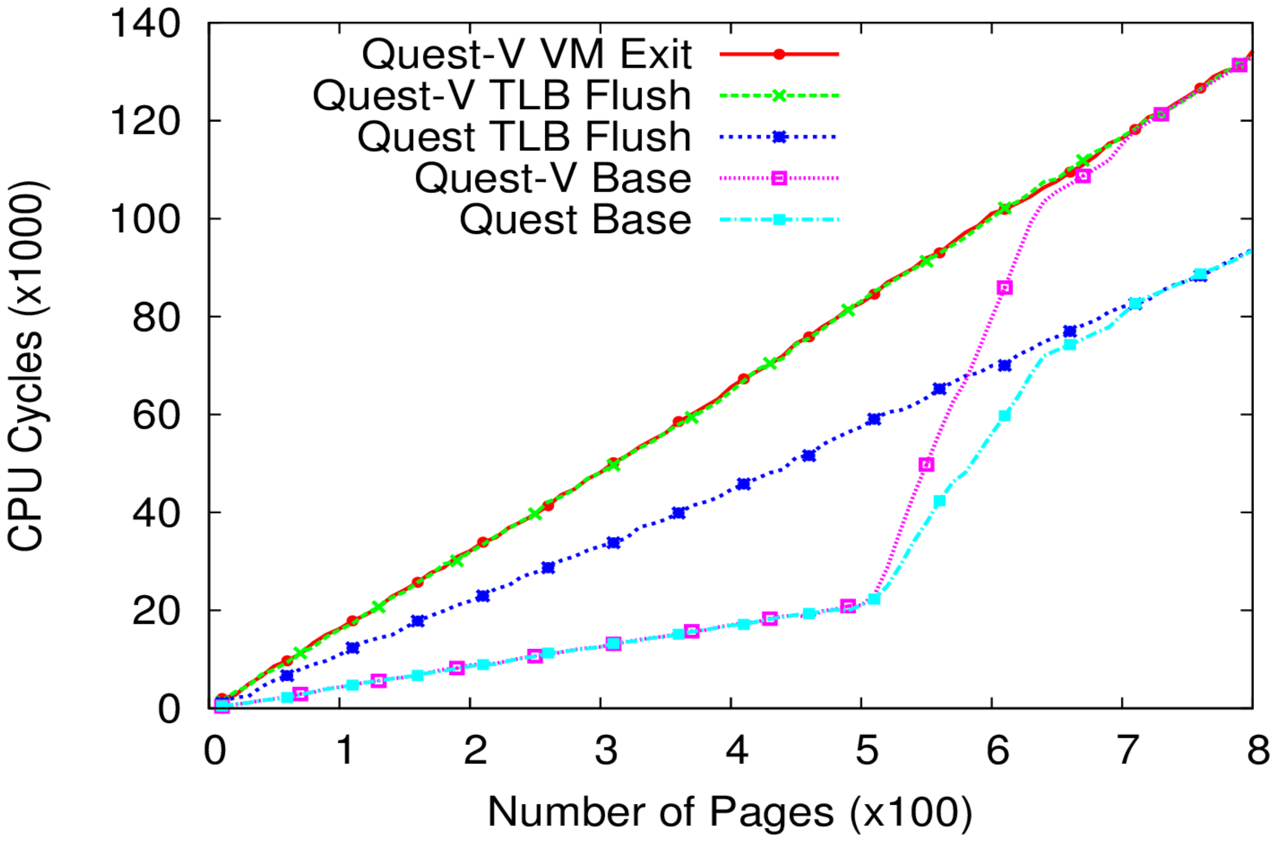


Quest-V Memory Partitioning



Memory Virtualization Costs

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

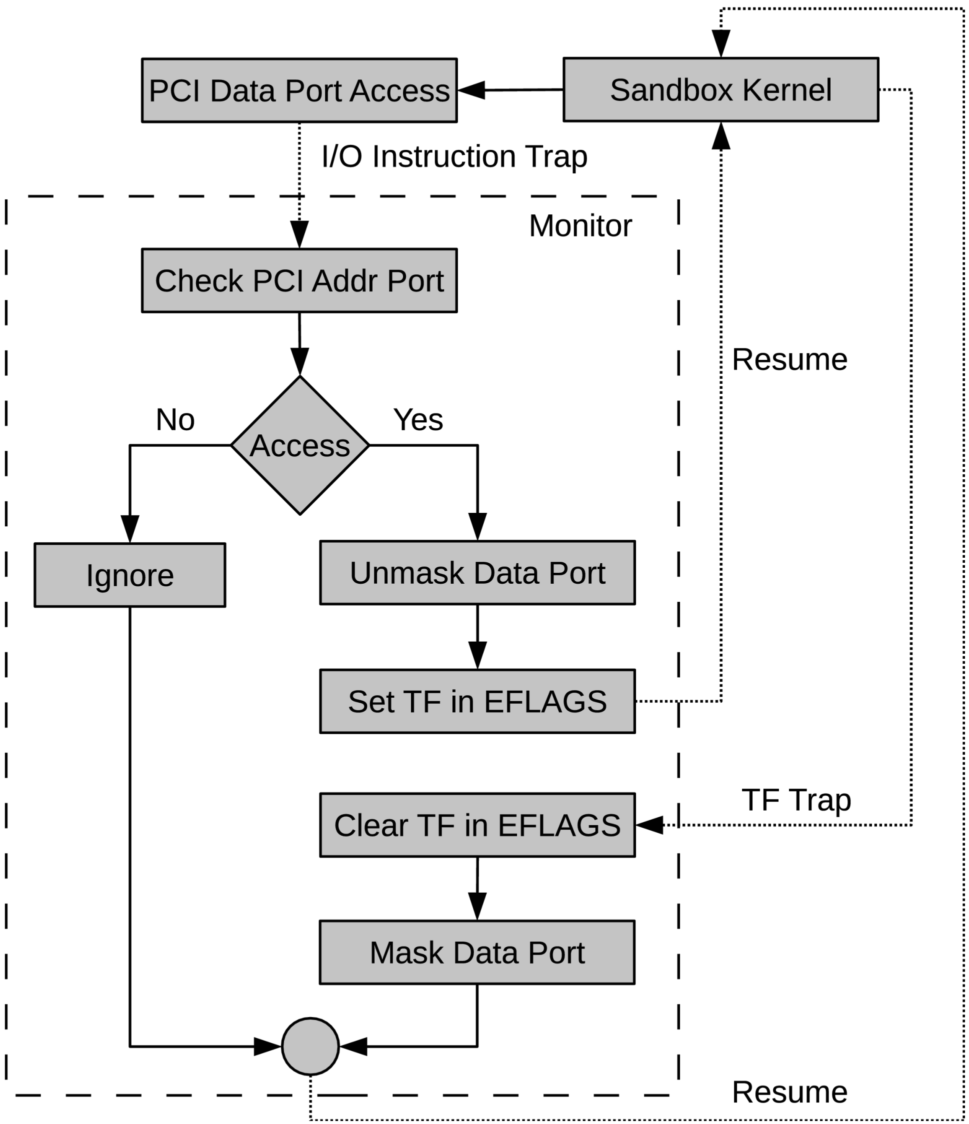


I/O Partitioning

- 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



Data Port:
0xCFC

Address Port:
0xCF8

Monitor Intervention

During normal operation only one monitor trap every 3-5 mins by CPUID

	No I/O Partitioning	I/O Partitioning (Block COM and NIC)
Exception (TF)	0	9785
CPUID	502	497
VMCALL	2	2
I/O Instruction	0	11412
EPT Violation	0	388
XSETBV	1	1

Table: Monitor Trap Count During Linux Sandbox Initialization

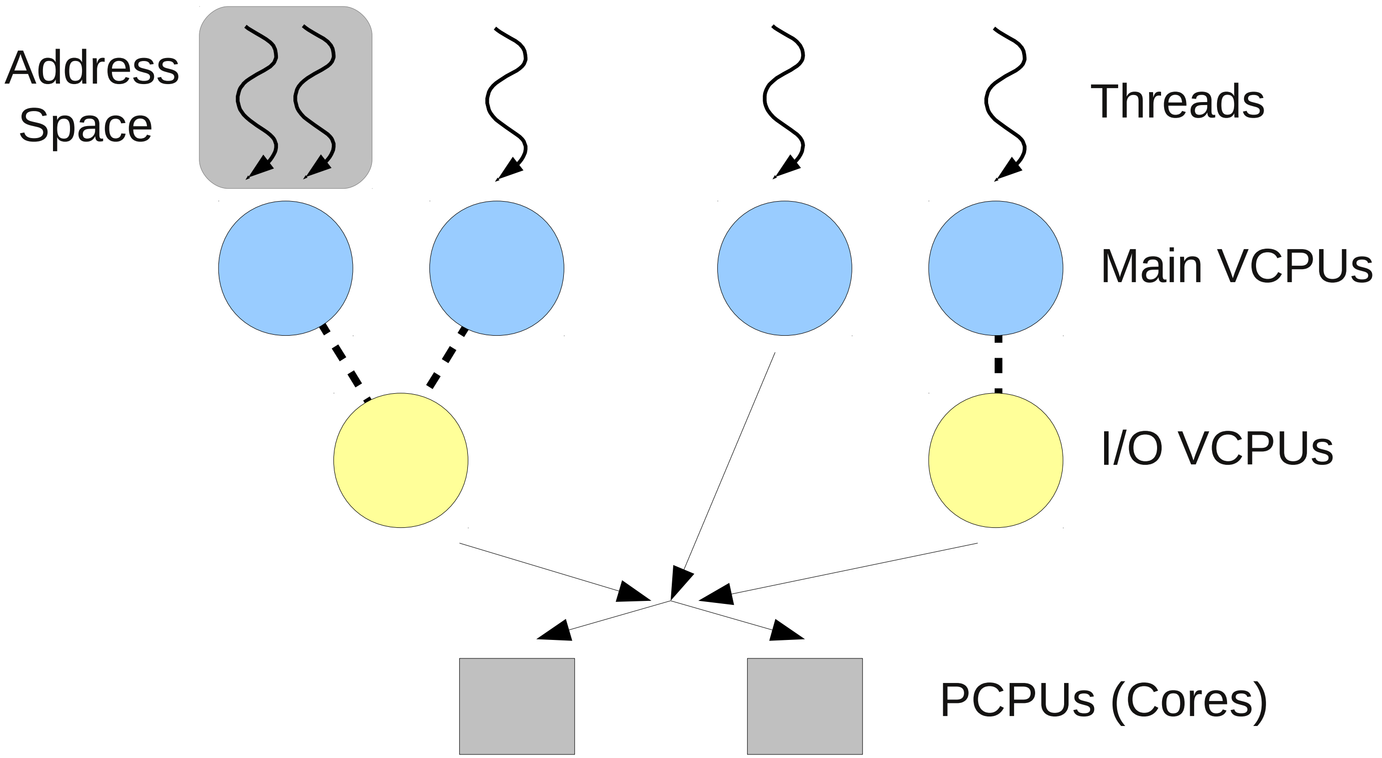
CPU Partitioning

- Scheduling local to each sandbox
  - partitioned rather than global
  - avoids monitor intervention
- Uses real-time VCPU approach for Quest native kernels **[RTAS'11]**

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

VCPUs in Quest(-V)



VCPUs in Quest(-V)

- Two classes
  - **Main** $\rightarrow$ for conventional tasks
  - **I/O** $\rightarrow$ for I/O event threads (e.g., ISRs)

- Scheduling policies
  - **Main** $\rightarrow$ sporadic server (SS)
  - **I/O** $\rightarrow$ priority inheritance bandwidth-preserving server (PIBS)

SS Scheduling

- Model periodic tasks
  - Each SS has a pair (C,T) s.t. a server is guaranteed **C** CPU cycles every period of **T** cycles when runnable
    - Guarantee applied at *foreground* priority
    - *background* priority when budget depleted
  - Rate-Monotonic Scheduling theory applies

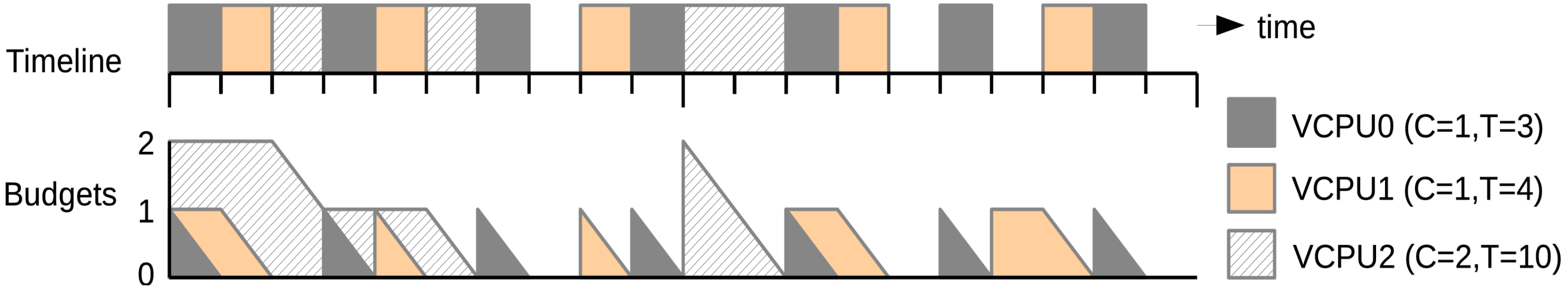
PIBS Scheduling

- IO VCPUs have utilization factor, $U_{V,IO}$
- IO VCPUs inherit priorities of tasks (or Main VCPUs) associated with IO events
  - Currently, priorities are f(T) for corresponding Main VCPU
  - IO VCPU budget is limited to:
    - $T_{V,main} * U_{V,IO}$ for period $T_{V,main}$

PIBS Scheduling

- IO VCPUs have *eligibility* times, when they can execute
- $t_e = t + C_{actual} / U_{V,IO}$
  - t = start of latest execution
  - t >= previous eligibility time

Example VCPU Schedule



Sporadic Constraint

- Worst-case preemption by a sporadic task for all other tasks is not greater than that caused by an equivalent periodic task

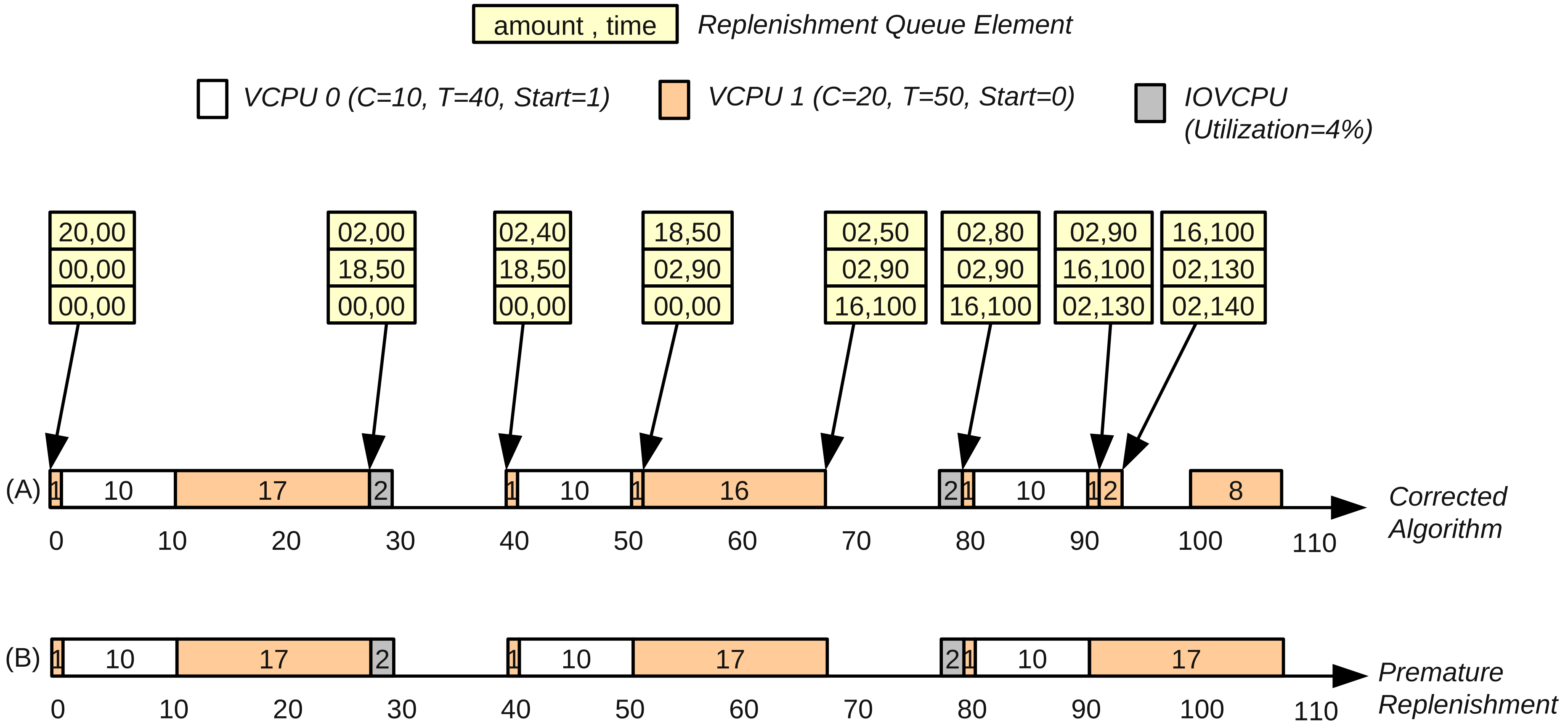
  (1) Replenishment, R must be deferred at least $t+T_V$

  (2) Can be deferred longer

  (3) Can merge two overlapping replenishments

    - R1.time + R1.amount >= R2.time then MERGE
    - Allow replenishment of R1.amount +R2.amount at R1.time

Example Replenishments



Interval [t=0,100] (A) VCPU 1 = 40%, (B) VCPU 1 = 46%

Utilization Bound Test

- Sandbox with 1 PCPU, n Main VCPUs, and m I/O VCPUs
  - Ci = Budget Capacity of Vi
  - Ti = Replenishment Period of Vi
  - Main VCPU, Vi
  - Uj = Utilization factor for I/O VCPU, Vj

$$\sum_{i=0}^{n-1} \frac{Ci}{Ti} + \sum_{j=0}^{m-1} (2-Uj) \cdot Uj \leq n \cdot \left(\sqrt[n]{2}-1\right)$$

Cache Partitioning

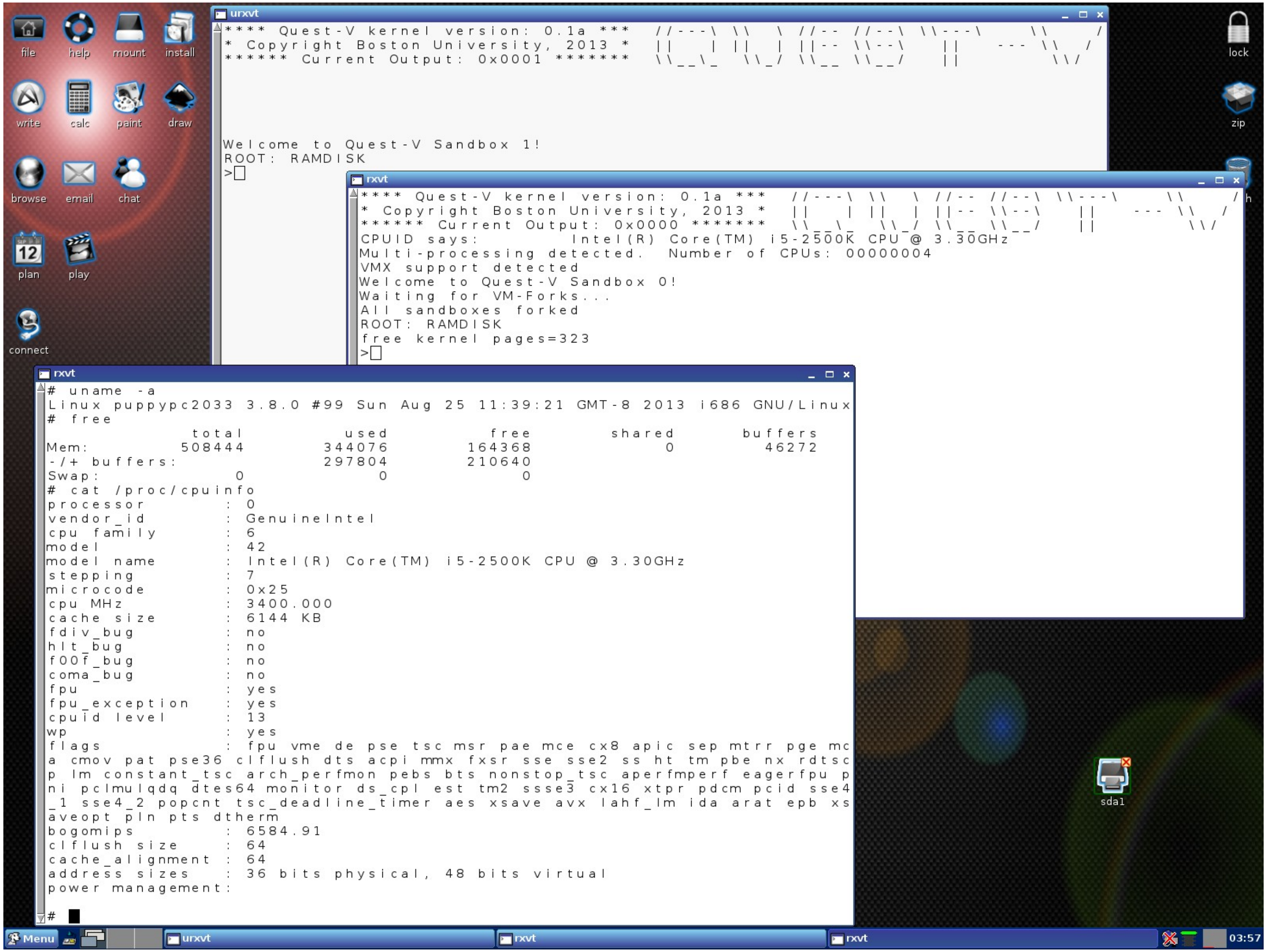
- Shared caches controlled using color-aware memory allocator
- Cache occupancy prediction based on h/w performance counters
  - $E' = E + (1-E/C) * m_l - E/C * m_o$
  - Enhanced with hits + misses

  [Book Chapter, OSR'11, PACT'10]

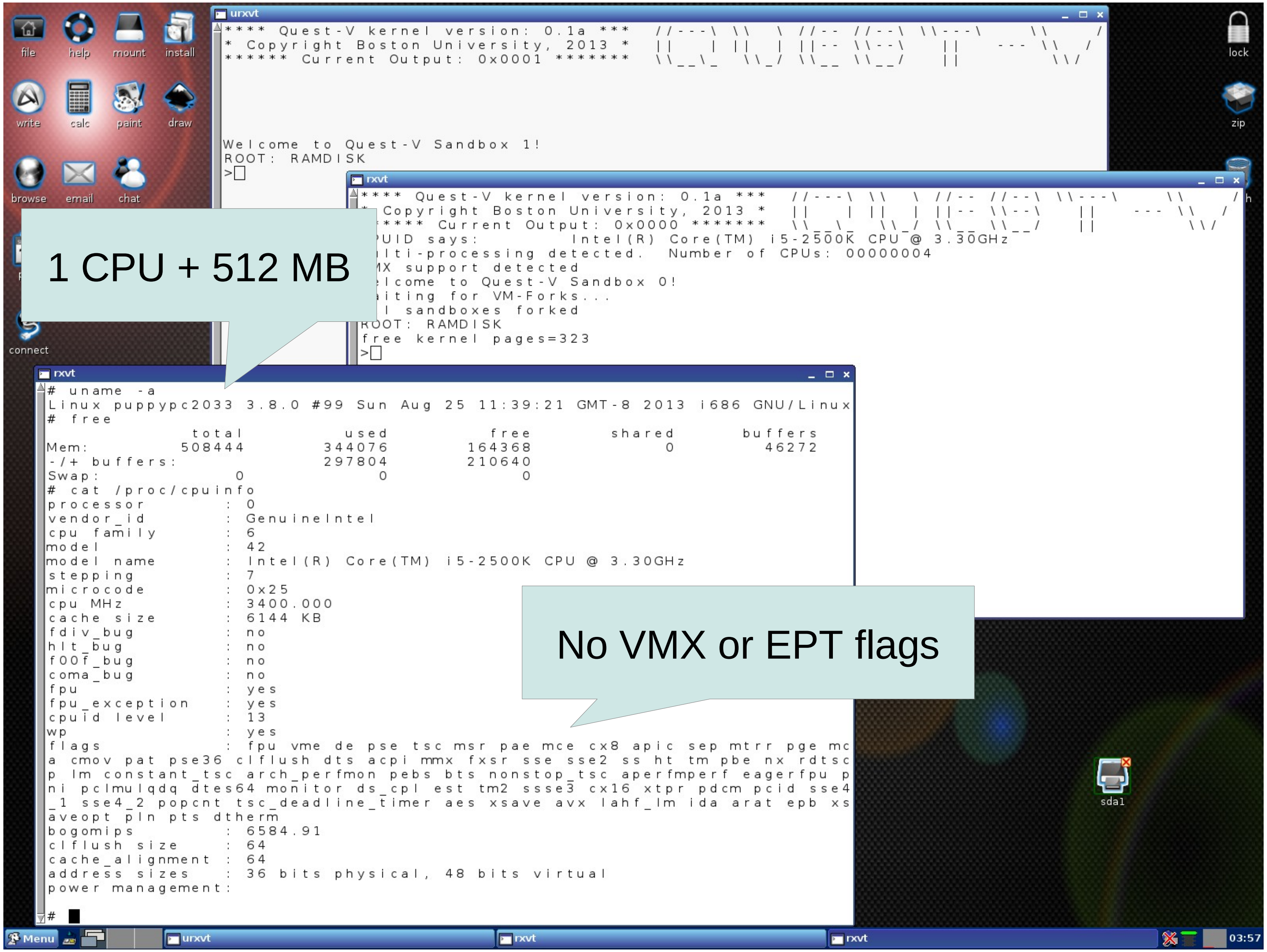
Linux Front End

- 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



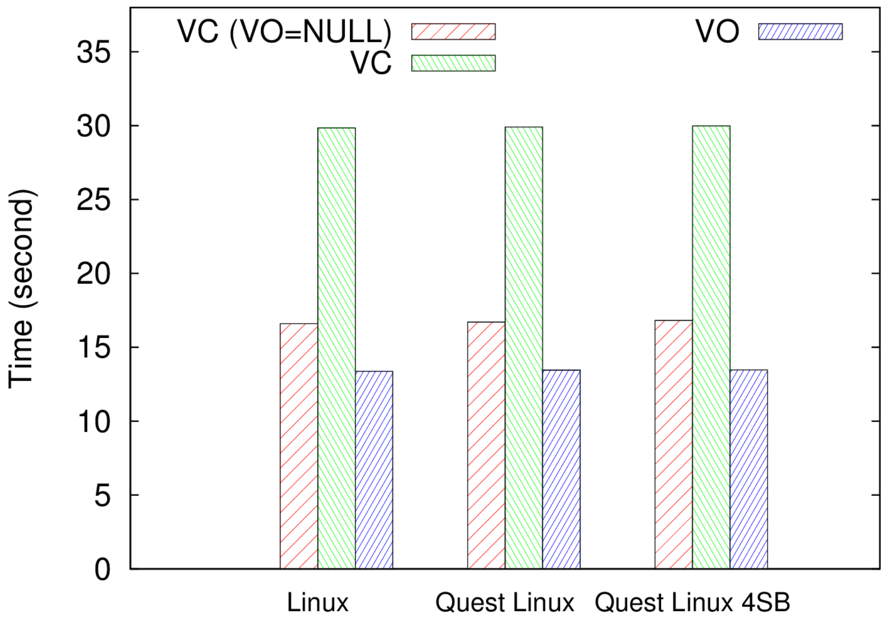
Quest-V Linux Screenshot



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

Quest-V Status

- About 11,000 lines of kernel code
- 200,000+ lines including lwIP, drivers, regression tests
- SMP, IA32, paging, VCPU scheduling, USB, PCI, networking, etc
- Quest-V requires BSP to send INIT-SIPI-SIPI to APs, as in SMP system
  - BSP launches 1st (guest) sandbox
  - APs "VM fork" their sandboxes from BSP copy

Future Work

- Online fault detection and recovery
- Technologies for secure monitors
  - e.g., Intel TXT + VT-d
- Separation kernel support for:
  - Accelerators / GPUs (time partitioning)
  - NoCs
  - Heterogeneous platforms (ala Helios satellite kernels)

See www.questos.org for more details

Quest-V Demo

- Bootstrapping Quest native kernel (core 0) + Linux (core 1)
  - Linux kernel + filesystem in RAM
  - Secure comms channel b/w Quest SB & Linux SB using a pseudo-char device
  - /dev/qSBx device for each sandbox x
- Triple modular redundancy (TMR) fault recovery for unmanned aerial vehicle (UAV)

http://quest.bu.edu/demo.html

The Quest Team

- Richard West
- Ye Li
- Eric Missimer
- Matt Danish
- Gary Wong

Other (Current) Developments

- Port of Quest to Intel Galileo Arduino
- Quest RT-USB host controller stack **[RTAS'13]**

10+ Years On...

- Intelligent transportation systems
  - V2V and V2I communications
  - Driverless cars (e.g., Google, CMU, Stanford, Oxford RobotCar, etc)

- Humanoid robots
  - Complex sensing + processing networks