Quest-V – a Virtualized Multikernel

Richard West

richwest@cs.bu.edu

Ye Li, Eric Missimer

{liye, missimer}@cs.bu.edu

Computer Science

Goals

- Develop system for high-confidence (embedded) systems

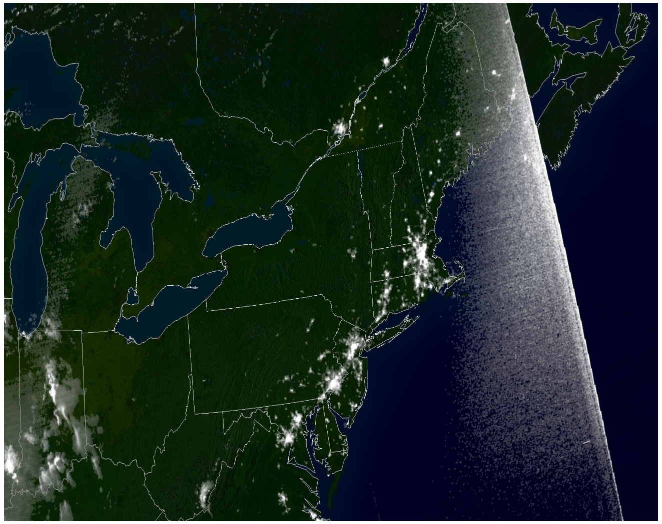
  - Predictable – real-time support
  - Resistant to component failures & malicious manipulation
  - Self-healing
  - Online recovery of software component failures

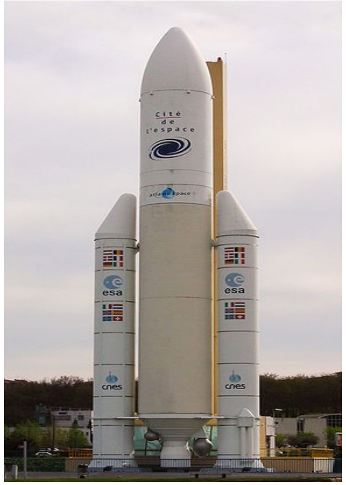
Target Applications

- 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

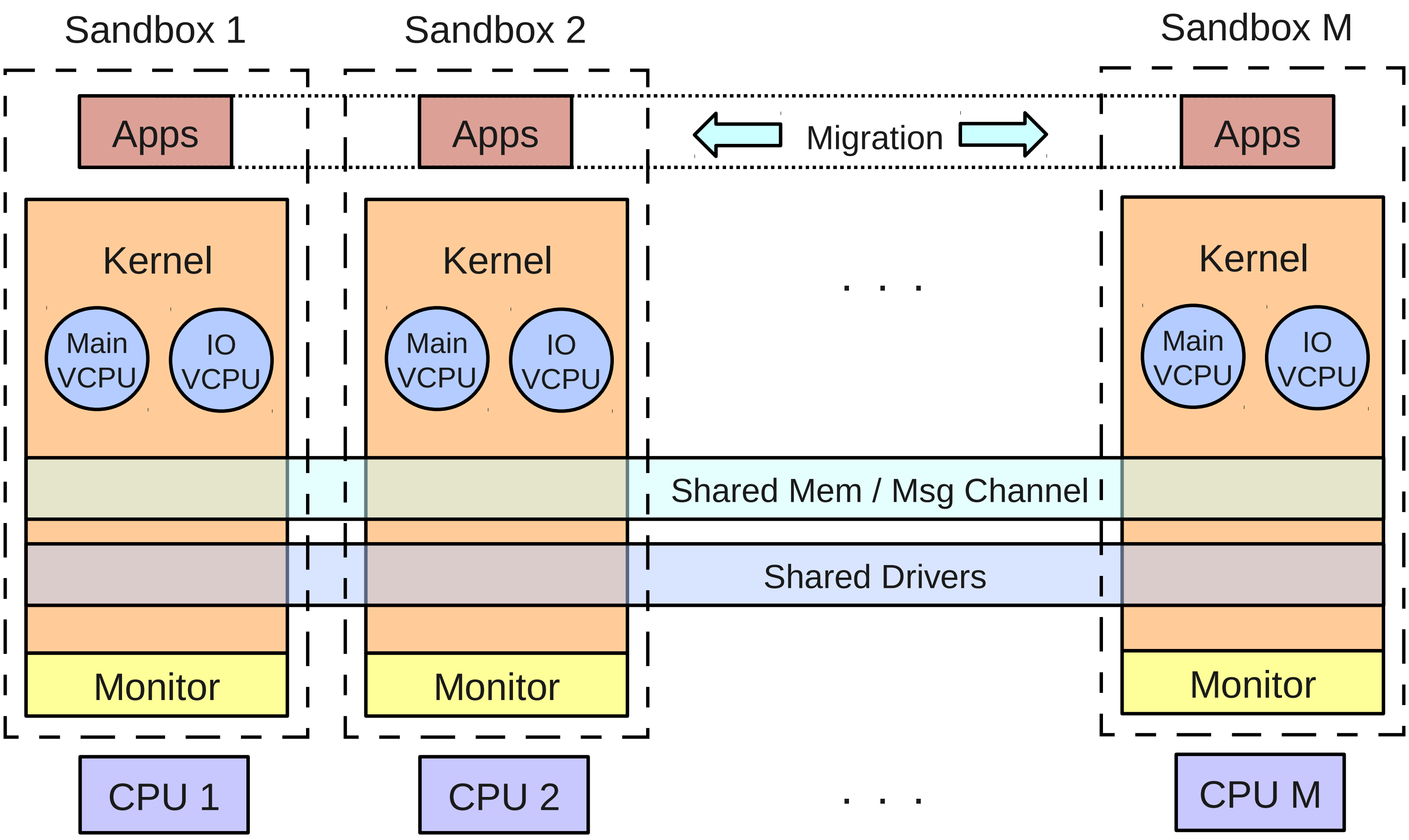




Approach

- Quest-V for multicore processors
  - Distributed system on a chip
  - Time as a first-class resource
    - Cycle-accurate time accountability
  - Separate sandbox kernels for system sub-components
  - Isolation using h/w-assisted memory virtualization
    - Extended page tables (EPTs – Intel)
    - Nested page tables (NPTs – AMD)
  - Security enforcible using VT-d + interrupt remapping (IR)
    - Device interrupts scoped to specific sandboxes
    - DMA xfers to specific host memory

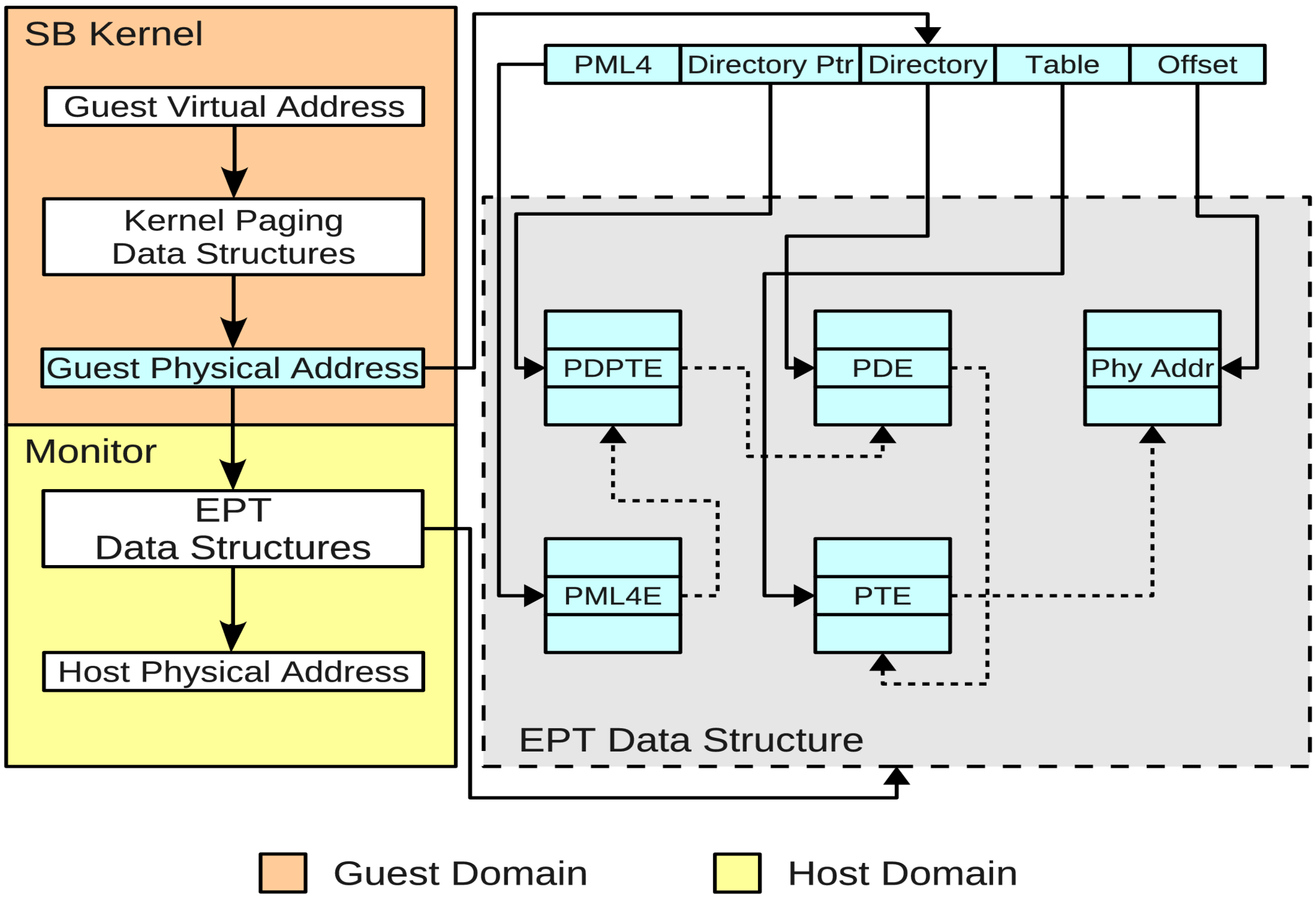
Architecture Overview



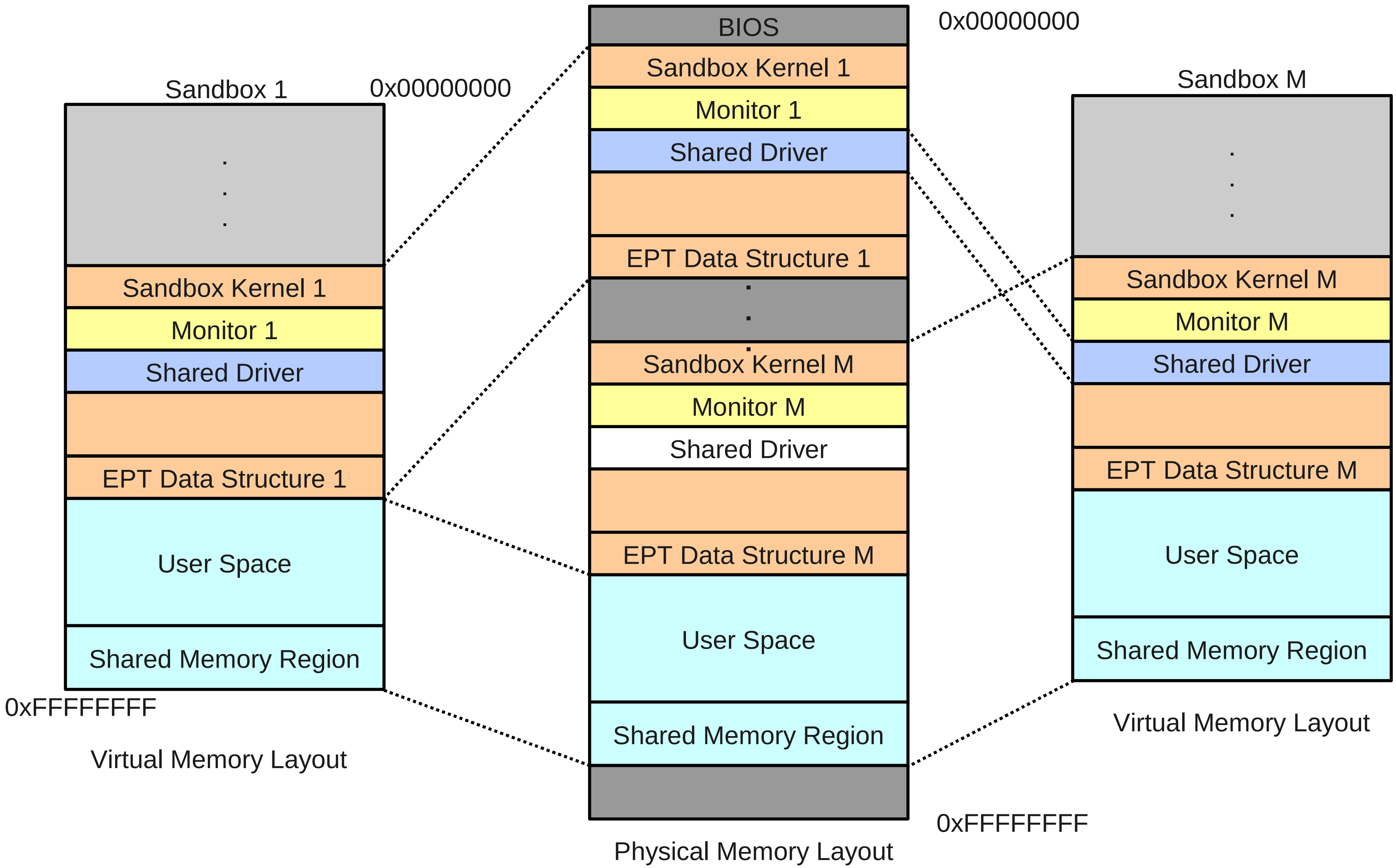
Isolation

- Memory virtualization using EPTs isolates sandboxes and their components
- Dedicated physical cores assigned to sandboxes
- Temporal isolation using Virtual CPUs (VCPUs)

Extended Page Tables



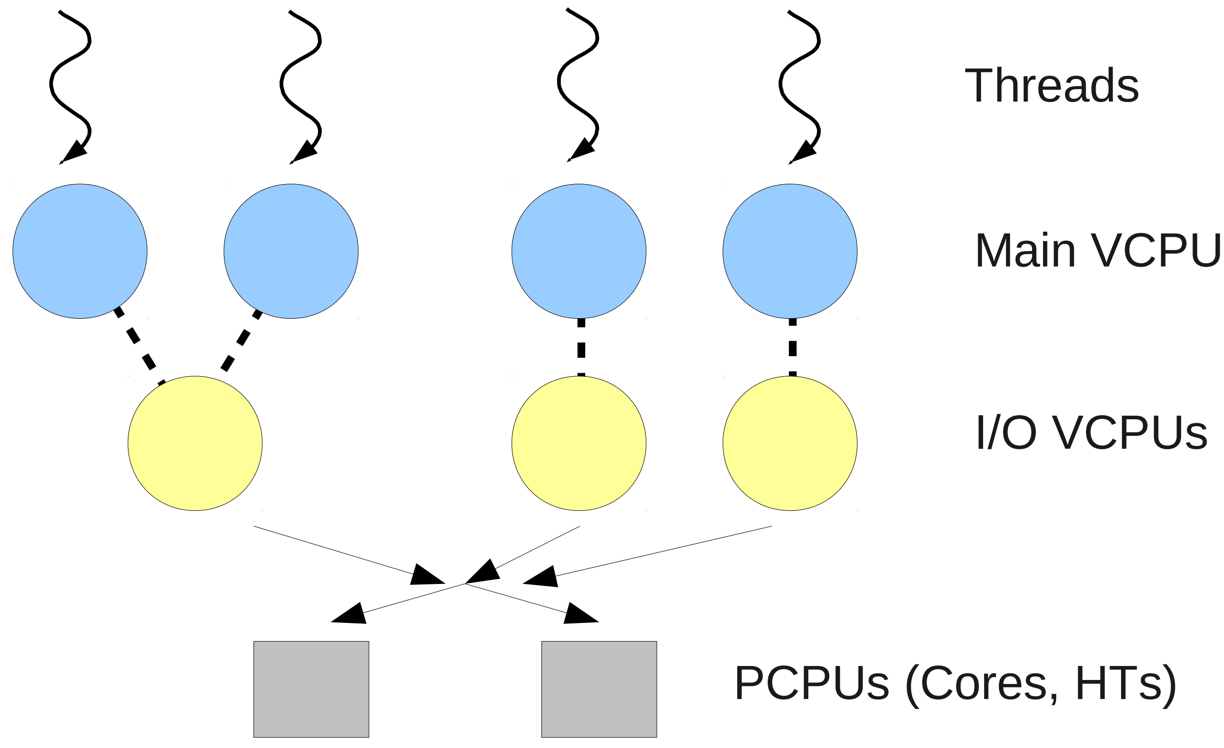
Quest-V Memory Layout



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

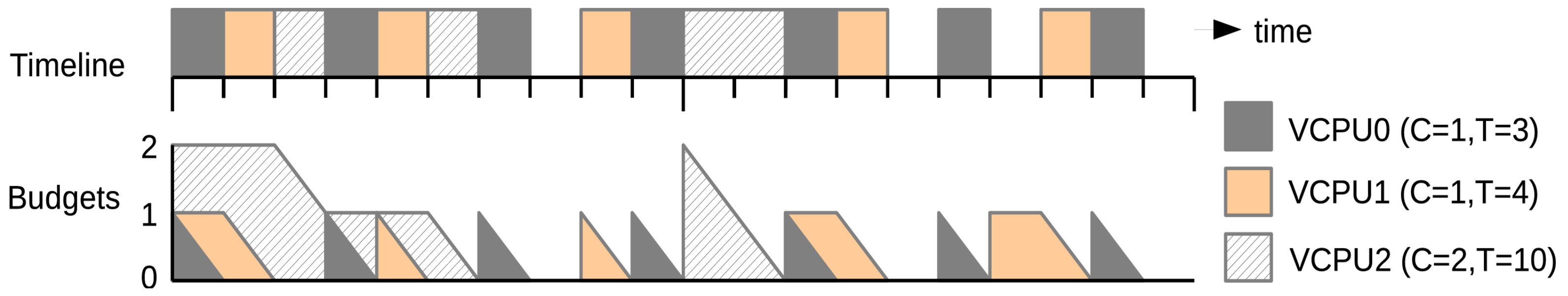
PIBS Scheduling

- 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

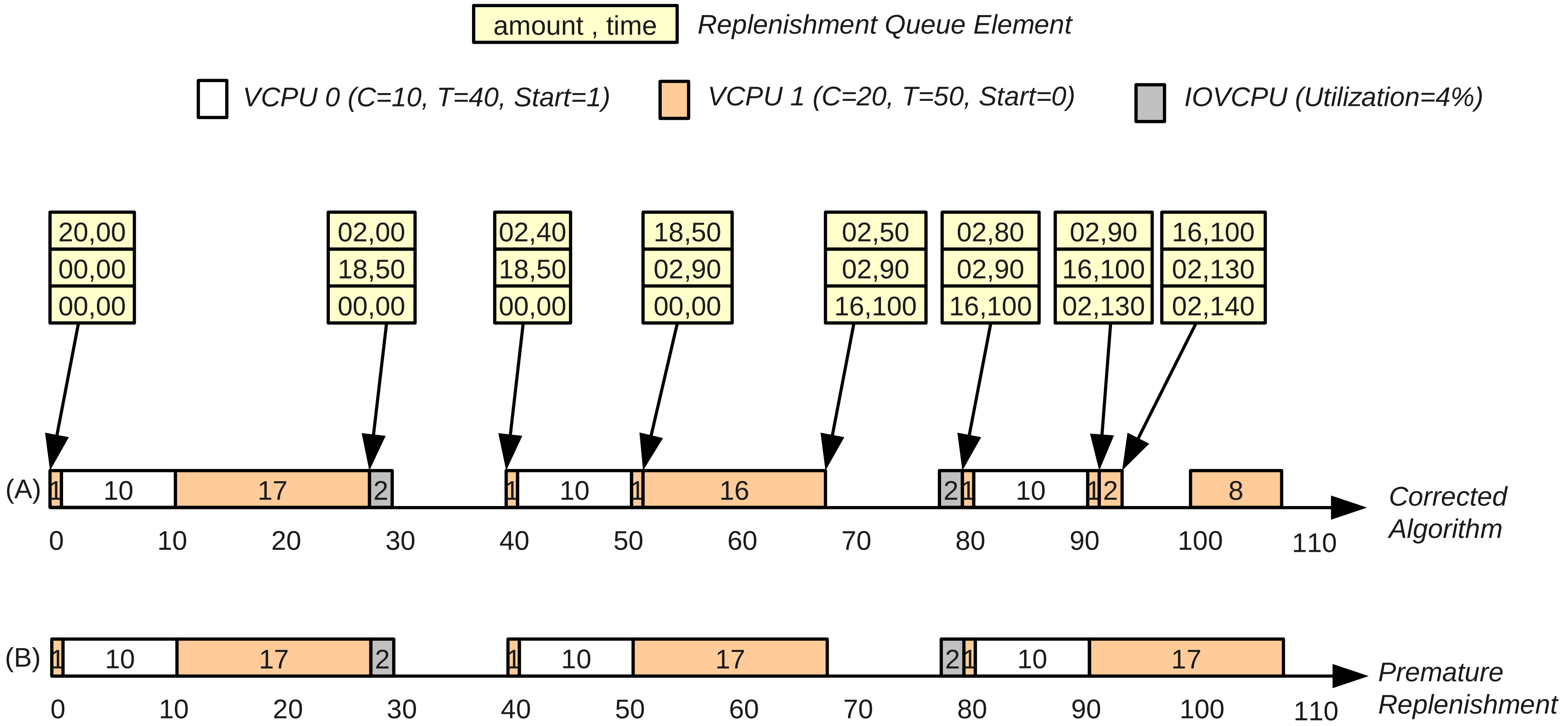
  (1) Replenishment, R must be deferred at least $t+T_V$

  (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

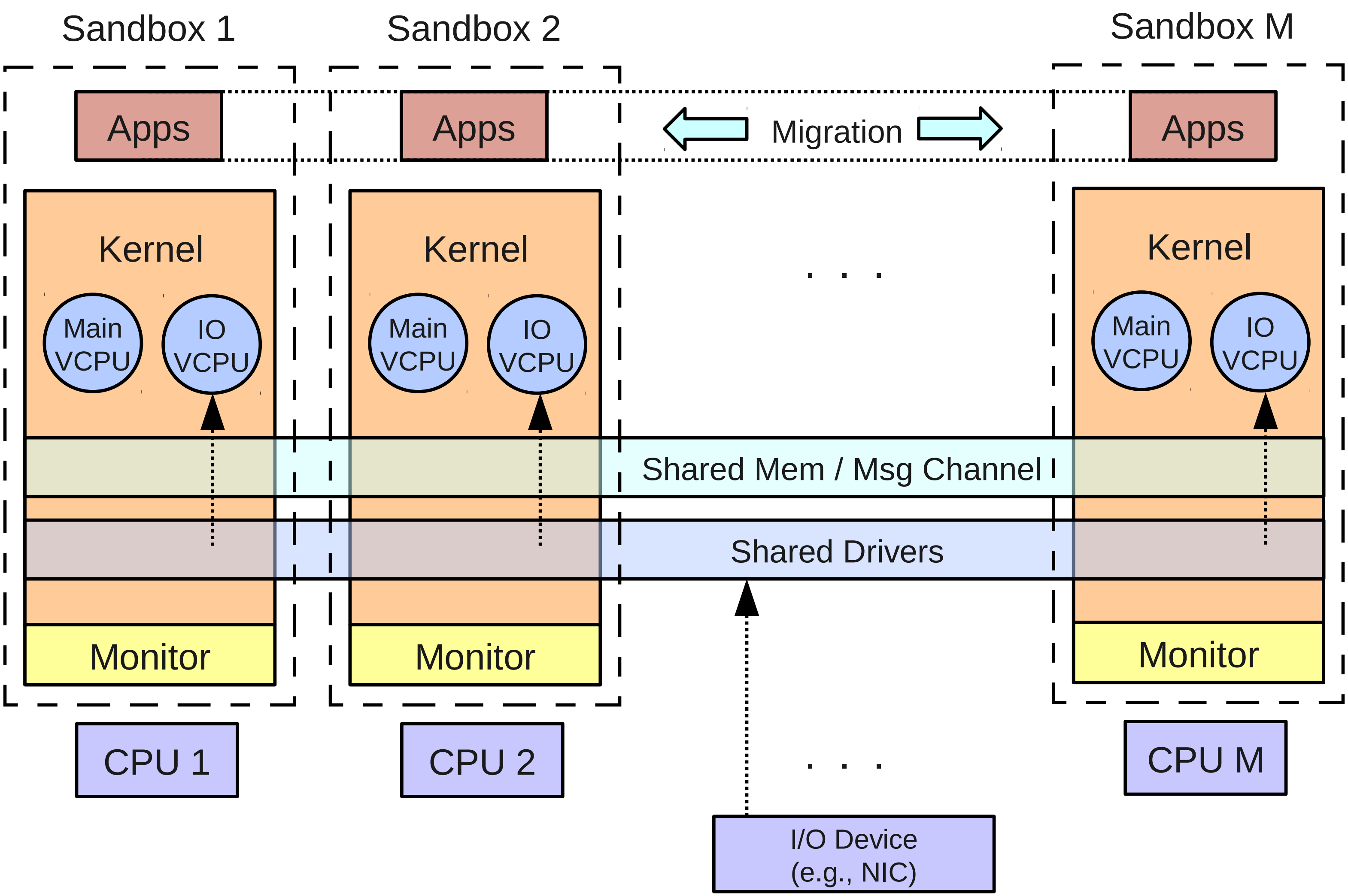
- Sandbox with 1 PCPU, n Main VCPUs, and m I/O VCPUs
  - Ci = Budget Capacity of Vi
  - Ti = Replenishment Period of Vi
  - Main VCPU, Vi
  - Uj = Utilization factor for I/O VCPU, Vj

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

Efficiency

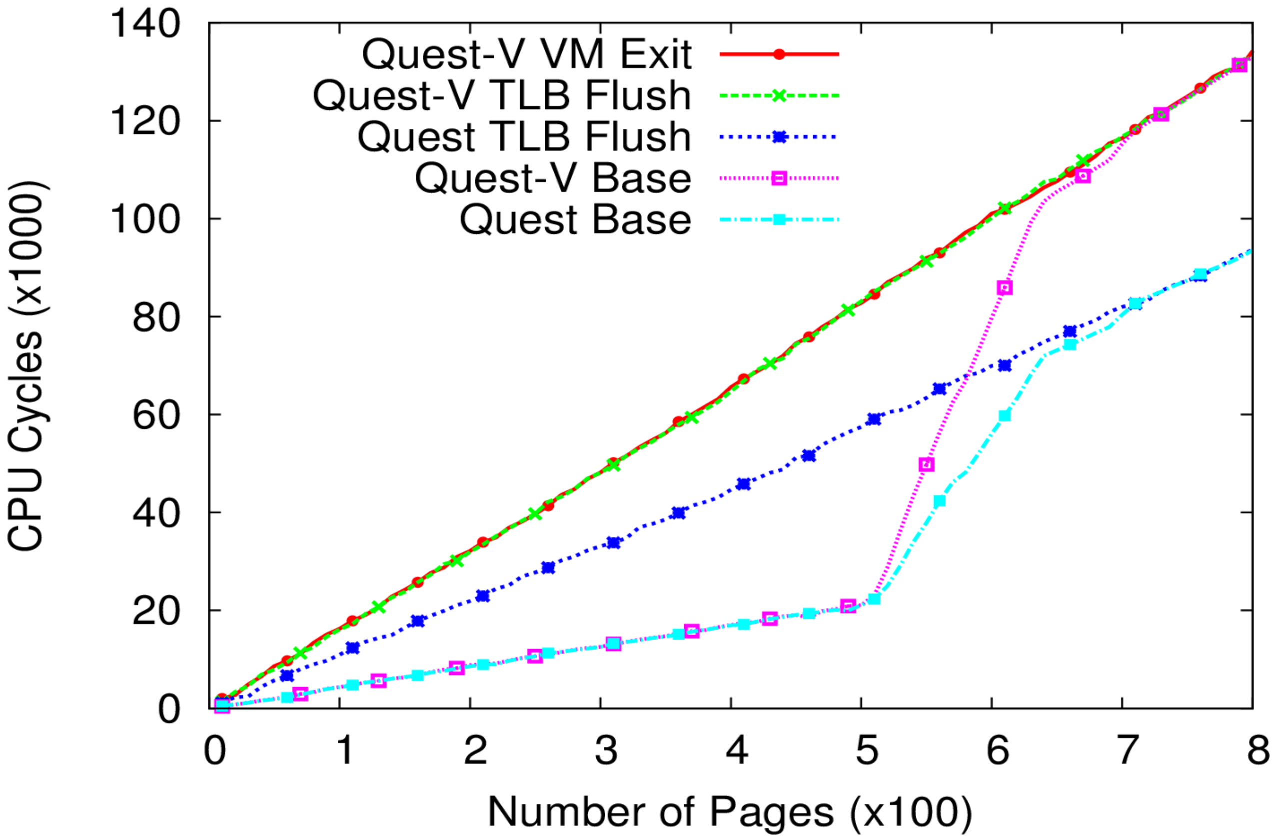
- Lightweight I/O virtualization & interrupt passthrough capabilities
  - e.g., VNICs provide separate interfaces to single NIC device
- Avoid VM-Exits into monitor for scheduling & I/O mgmt

I/O Passthrough



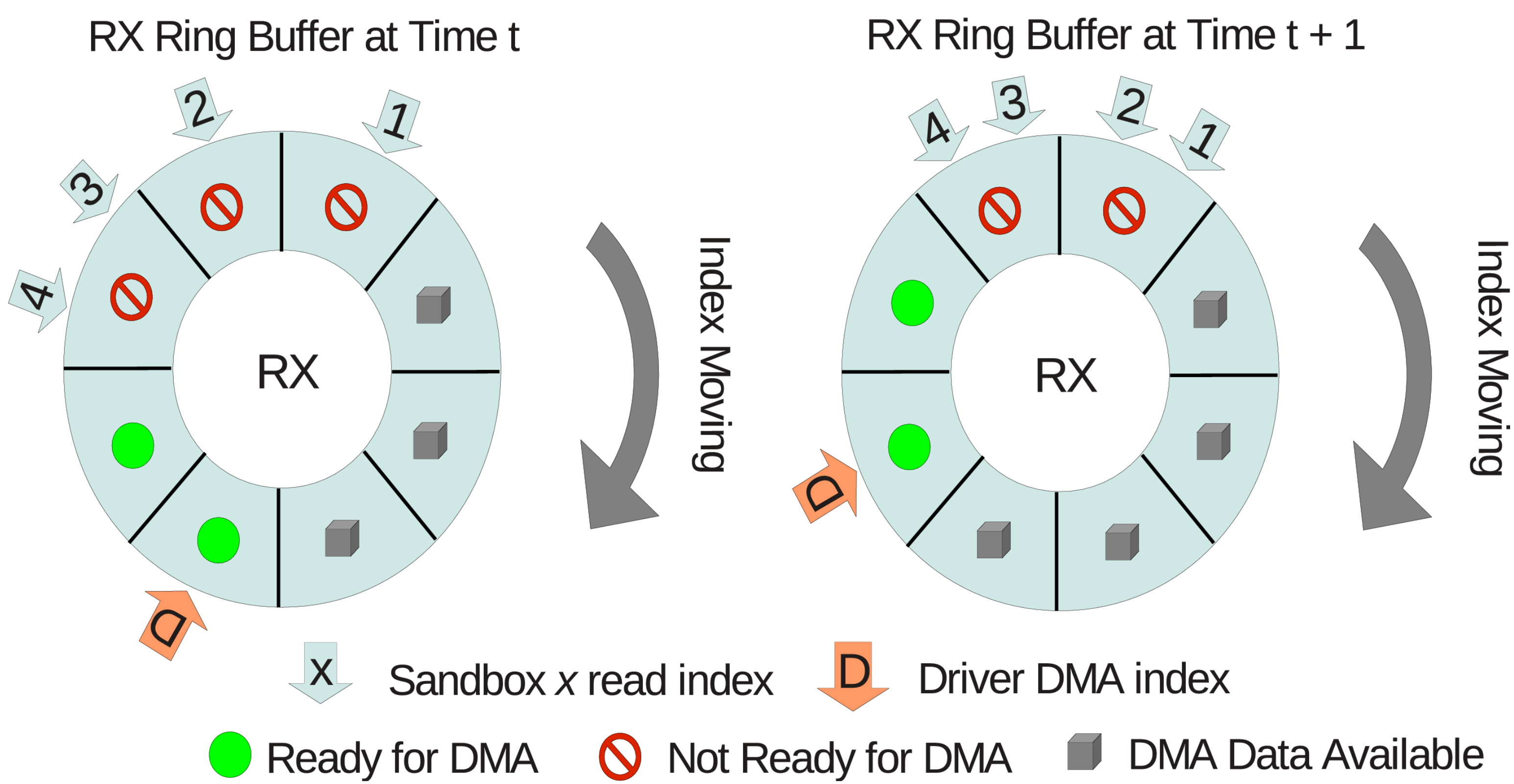
Virtualization Costs

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

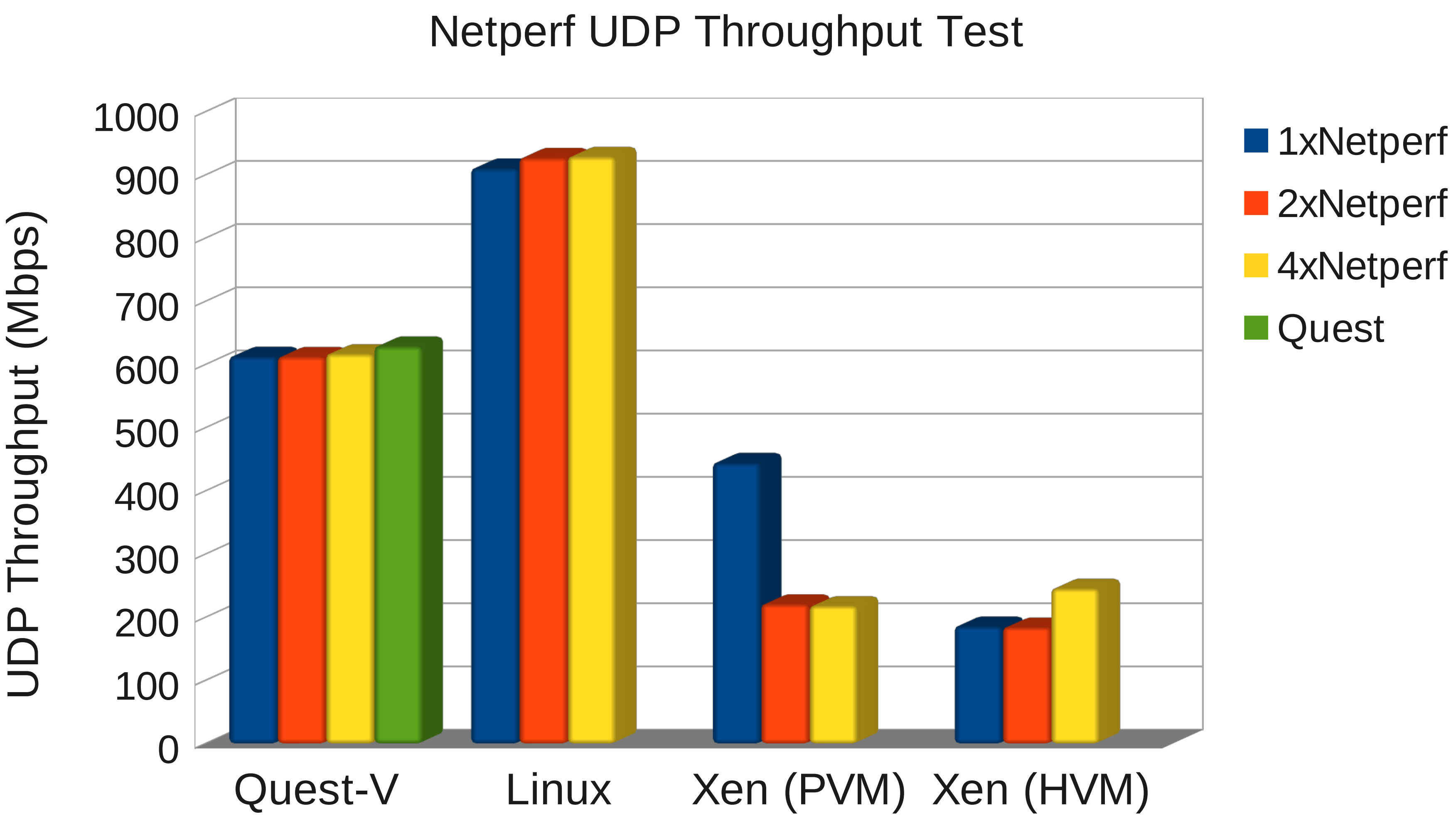


Device (Driver) Sharing

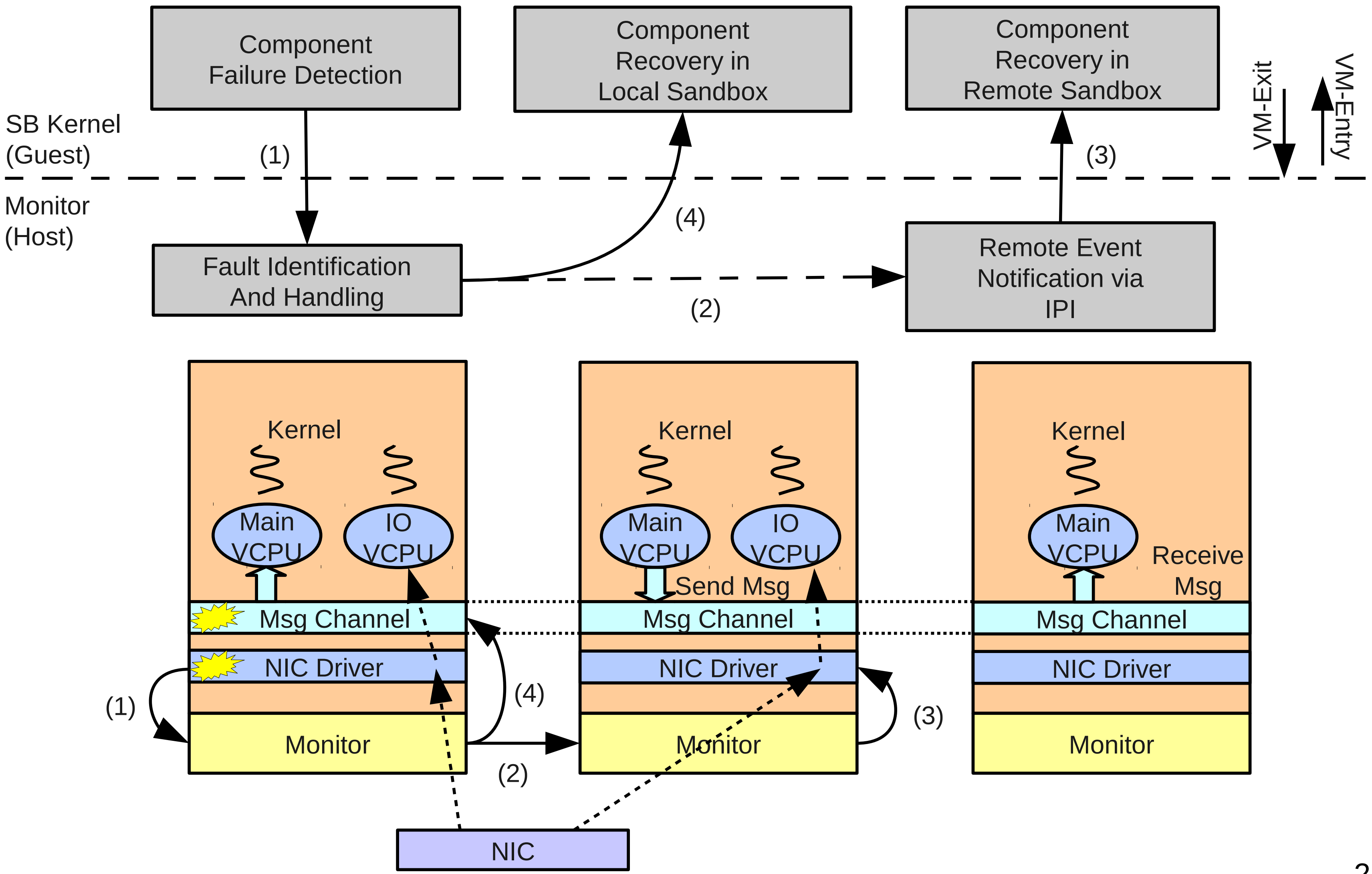
- Example NIC RX Ring Buffer



Shared Driver Costs

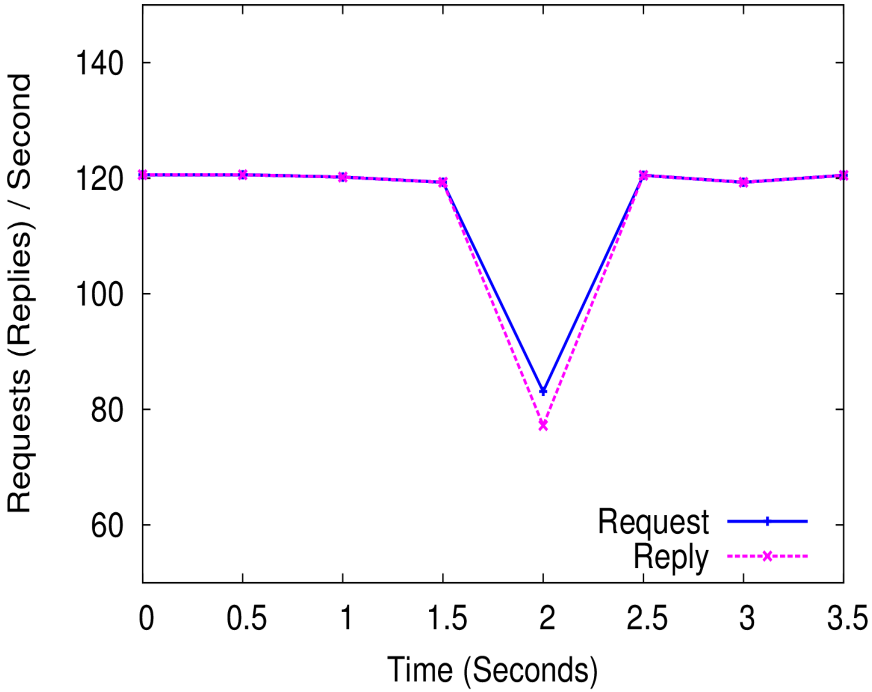


Example Fault Recovery



Faulting Driver for Web Server

- httperf with web server in presence of Realtek NIC driver fault
- Requests / replies set at 120/s under normal operation
  - Single-threaded server
  - Focus on one process
  - Recovery time rather than throughput



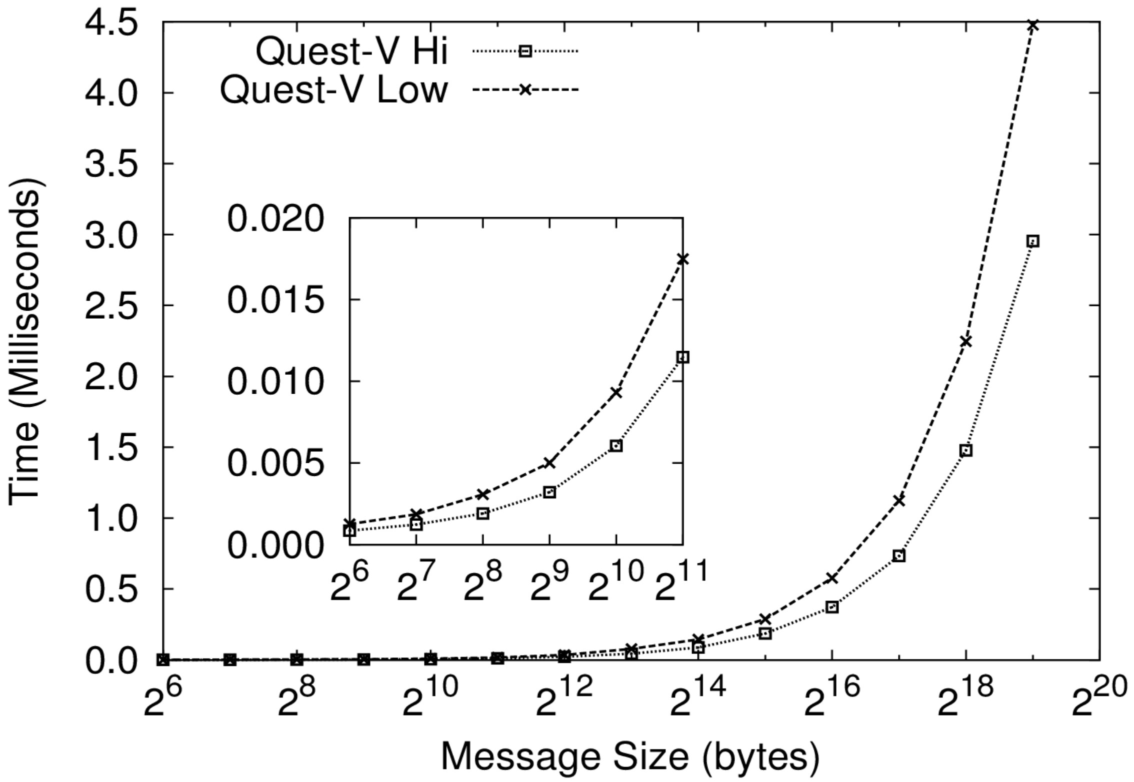
Performance Costs

- Core i5-2500K with 8GB RAM

Recovery Phases	CPU Cycles	
	Local Recovery	Remote Recovery
VM-Exit	885	
Driver Switch	10503	N/A
IPI Round Trip	N/A	4542
VM-Enter	663	
Driver Re-initialization	1.45E+07	
Network Re-initialization	78351	

Inter-Sandbox Communication

- Via Communication VCPUs
  - High rate VCPUs: 50/100ms
  - Low rate VCPUs: 40/100ms



The Quest Team

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

Further Information

- Quest website
  - http://www.cs.bu.edu/fac/richwest/quest.html

- Github public repo
  - http://questos.github.com

Quest(-V) Summary

- About 11,000 lines of kernel code
- 175,000+ 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

Final Remarks

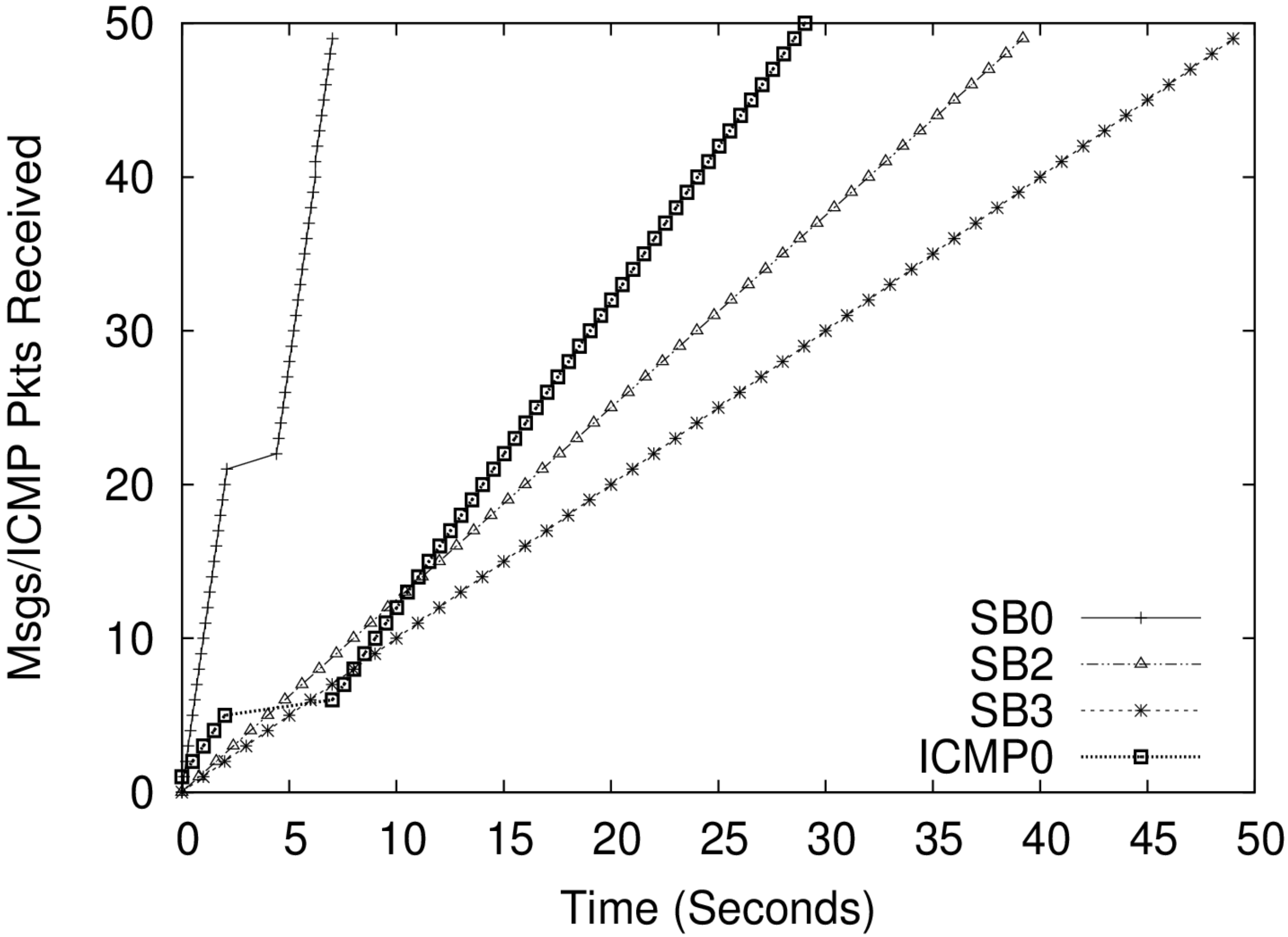
- Quest-V multikernel
  - Leverages H/W virtualization for safety/isolation
  - Avoids VM-Exits for VCPU/thread scheduling
  - Online fault recovery
  - Shared memory communication channels
  - Lightweight I/O virtualization
  - Predictable VCPU scheduling framework

Isolation

- 4 sandboxes: SB0,..., SB3
  - SB1 sends msgs to SB0, SB2 & SB3 at 50ms intervals
    - SB0, SB2 & SB3 rx at 100, 800, 1000ms intervals, respectively
  - SB0 handles ICMP requests
    - sent remotely at 500ms intervals
  - Observe failure + recovery in SB0

  - Messaging threads on Main VCPUs: 20ms/100ms
  - NIC driver I/O VCPU: 1ms/10ms

Isolation



Msgs/ICMP Pkts Received
50
40
30
20
10
0
0
5
10
15
20
25
30
35
40
45
50
Time (Seconds)
SB0
SB2
SB3
ICMP0

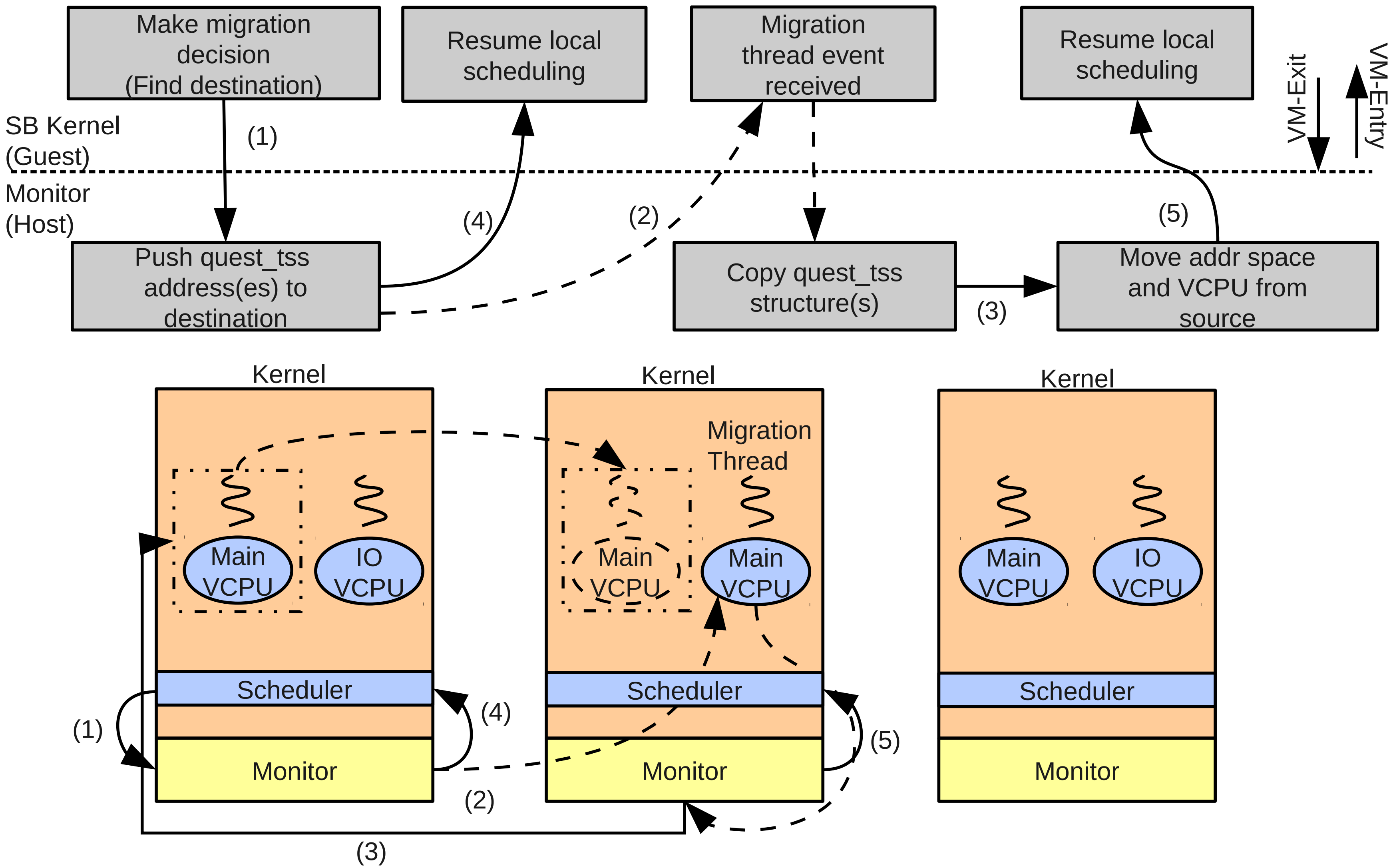
Next Steps

- VCPU/thread migration
- API extensions
- Application development
- Hardware performance monitoring
- RT-USB sub-system
- Fault detection

Real-Time Migration

- At t, guarantee VCPU, V_{src}, moves from $SB_{src} \rightarrow SB_{dest}$ without violating:
  - (a) Remote VCPU requirements, $\forall V_{dest} \in SB_{dest}$
  - (b) Requirements of V_{src}
- Use migration VCPUs, $V_{migrate}$ $[C_{mig}, T_{mig}]$
- Ensure: $U_{dest} + \frac{C_{src}}{T_{src}} \leq (n+1)\left(\sqrt[n+1]{2}-1\right), |V_{dest}| = n \, @ \, t' < t$
- Ensure: C[memcpy of V_{src}+thread(s)] <= C_{mig}
  - while V_{src} is ineligible for execution

Real-Time Migration



VCPU API

- Full thread support
  - NB: Limit a VCPU to one address space
  - Reduces migration costs

```
int VCPU_create(struct vcpu_param *param)
struct vcpu_param {
  int vcpuid;
  policy; // SCHED_SPORADIC, SCHED_PIBS
  int mask; // affinity mask
  int C; // budget
  int T; // period
}
```

VCPU API

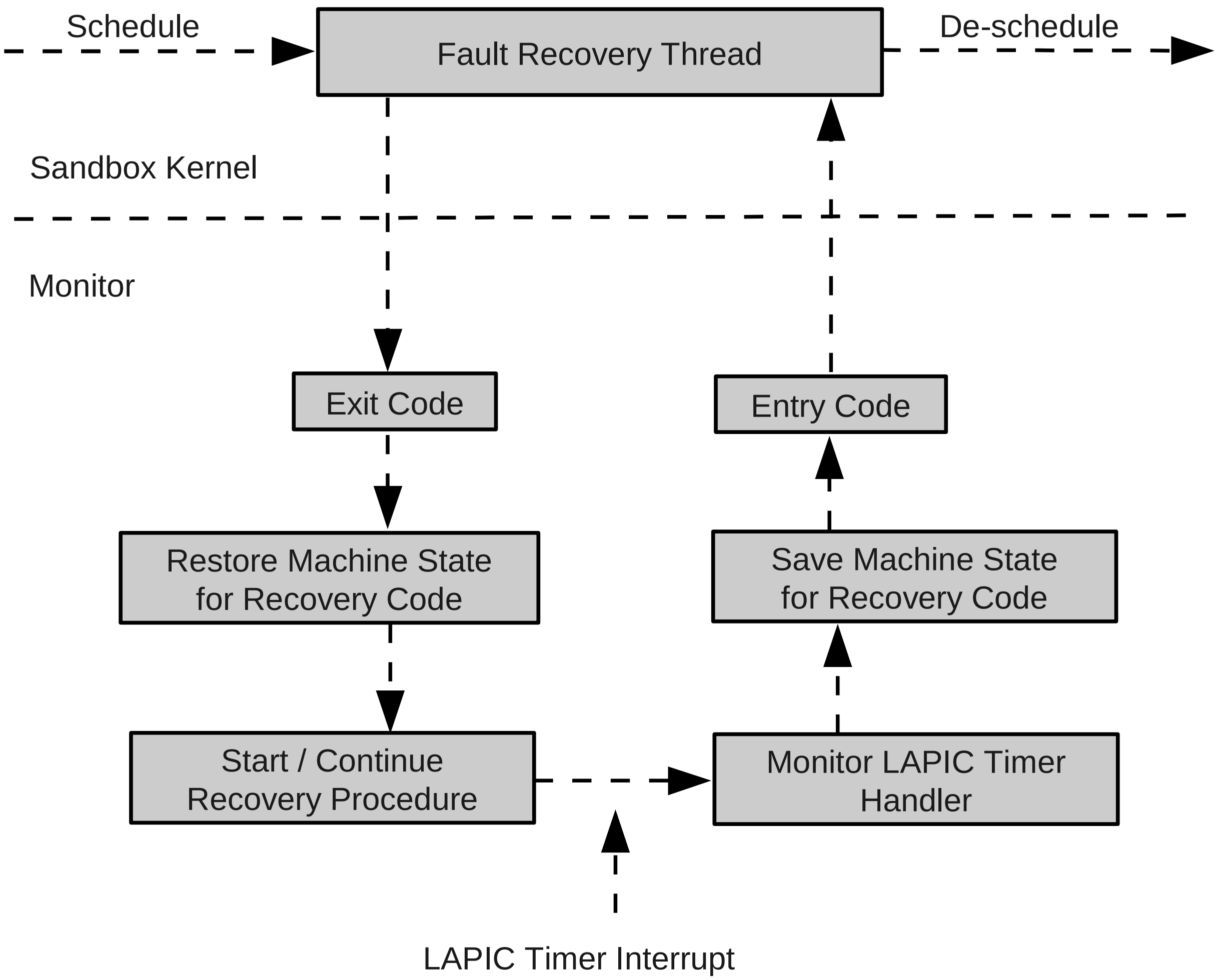
- `int VCPU_destroy(int vcpuid, int force);`
- `int VCPU_setparam(int vcpuid, struct vcpu_param *param);`
- `int VCPU_getparam(struct vcpu_param *param);`
- `Int VCPU_bind_task(int vcpuid);`

- Policy:
  - Which sandboxes assigned which VCPUs?
    - Utilization considerations
    - Cache usage (perfmon)
    - Have SBs announce their utilization (bidding)

Real-Time Fault Recovery

- Real-time fault recovery
  - Local & remote
  - Requires SB with working scheduler for predictable recovery
  - Remote recovery can avoid re-initialization of faulting service

Real-Time Fault Recovery



Applications

- RacerX
- TORCS
- Benchmarks
  - Web server
  - Netperf
  - Canny
  - Others?

Performance Monitoring

- (LLC) Cache hits, misses, instrs retired, TSC,...
- Can predict s/w thread LLC occupancy in real-time
  - $E' = E + (1-E/C)*M_l - E/C*M_0$
  - See West, Zaroo, Waldspurger & Zhang
    - OSR, December 2010

Experiments

- Intel Core2 Extreme QX6700 @ 2.66GHz
- 4GB RAM
- Gigabit Ethernet (Intel 8254x “e1000”)
- UHCI USB Host Controller
  - 1GB USB memory stick
- Parallel ATA CDROM in PIO mode

- Measurements over 5sec windows using bandwidth-preserving logging thread

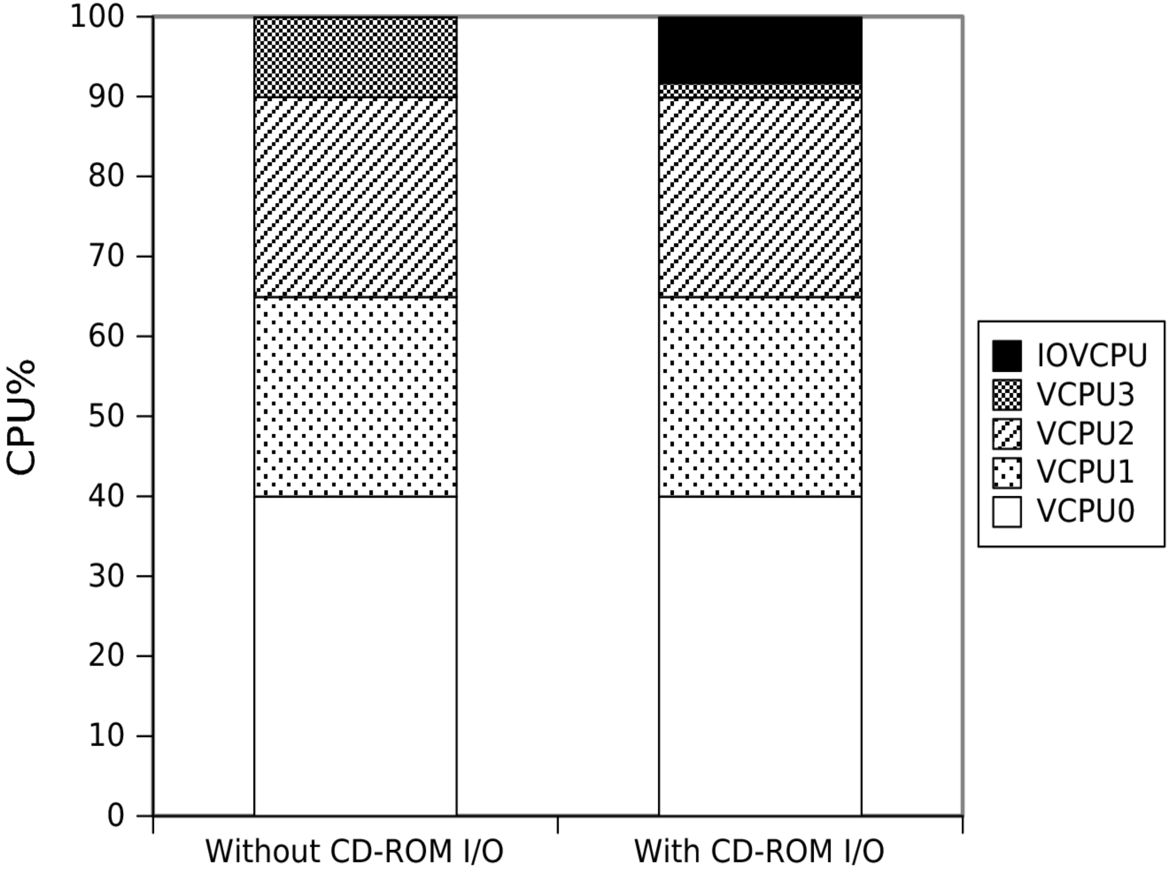
Experiments

- CPU-bound threads: increment a counter
- CD ROM/USB threads: read 64KB data from filesystem on corresponding device

I/O Effects on VCPUs

VCPU	V_C	V_T	threads
VCPU0	2	5	CPU-bound
VCPU1	2	8	Reading CD, CPU-bound
VCPU2	1	4	CPU-bound
VCPU3	1	10	Logging, CPU-bound
IOVCPU	10%	ATA	

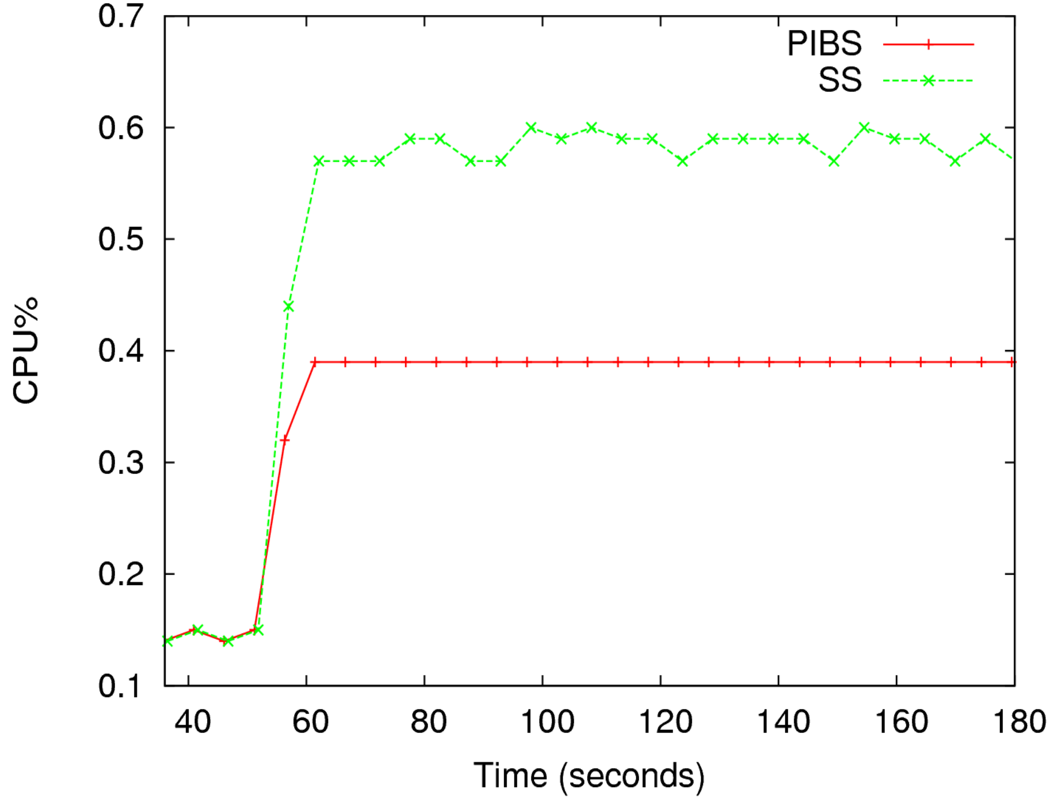
I/O Effects on VCPUs



PIBS vs SS IO VCPU Scheduling

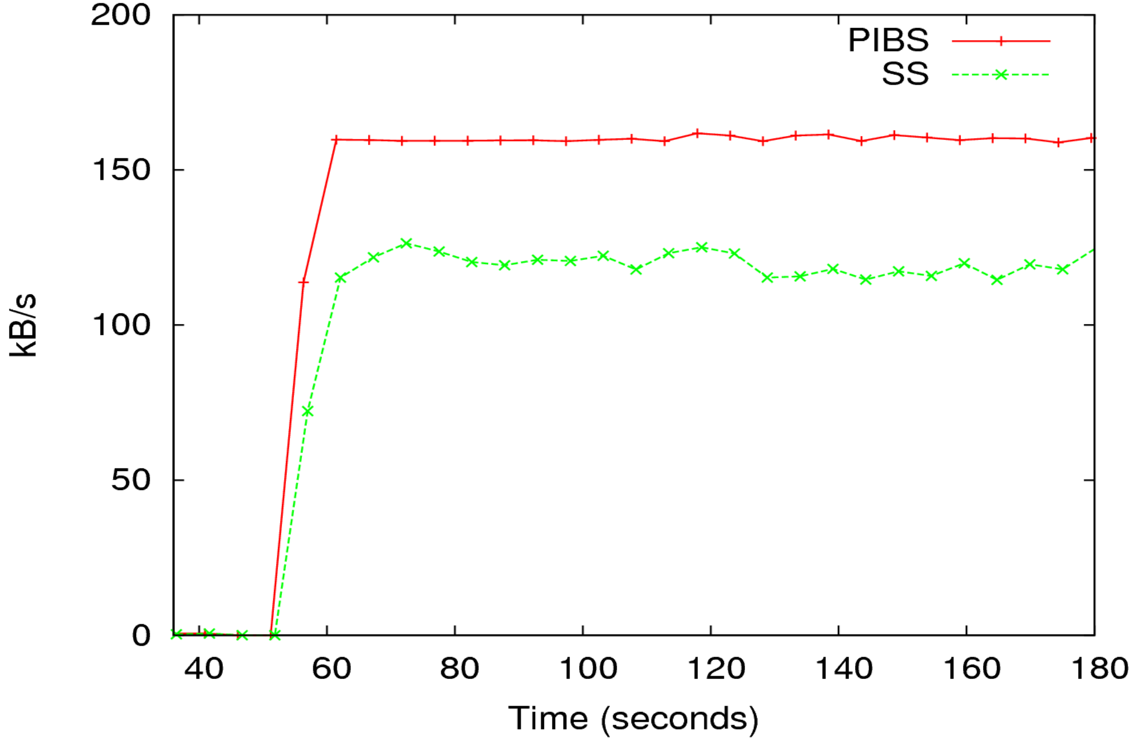
VCPU	V_C	V_T	threads
VCPU0	1	20	CPU-bound
VCPU1	1	30	CPU-bound
VCPU2	10	100	Network, CPU-bound
VCPU3	20	100	Logging, CPU-bound
IOVCPU	1%	Network	

PIBS vs SS IO VCPU Scheduling



t=50 start ICMP ping flood. Here, we see comparison overheads of two scheduling policies

PIBS vs SS IO VCPU Scheduling



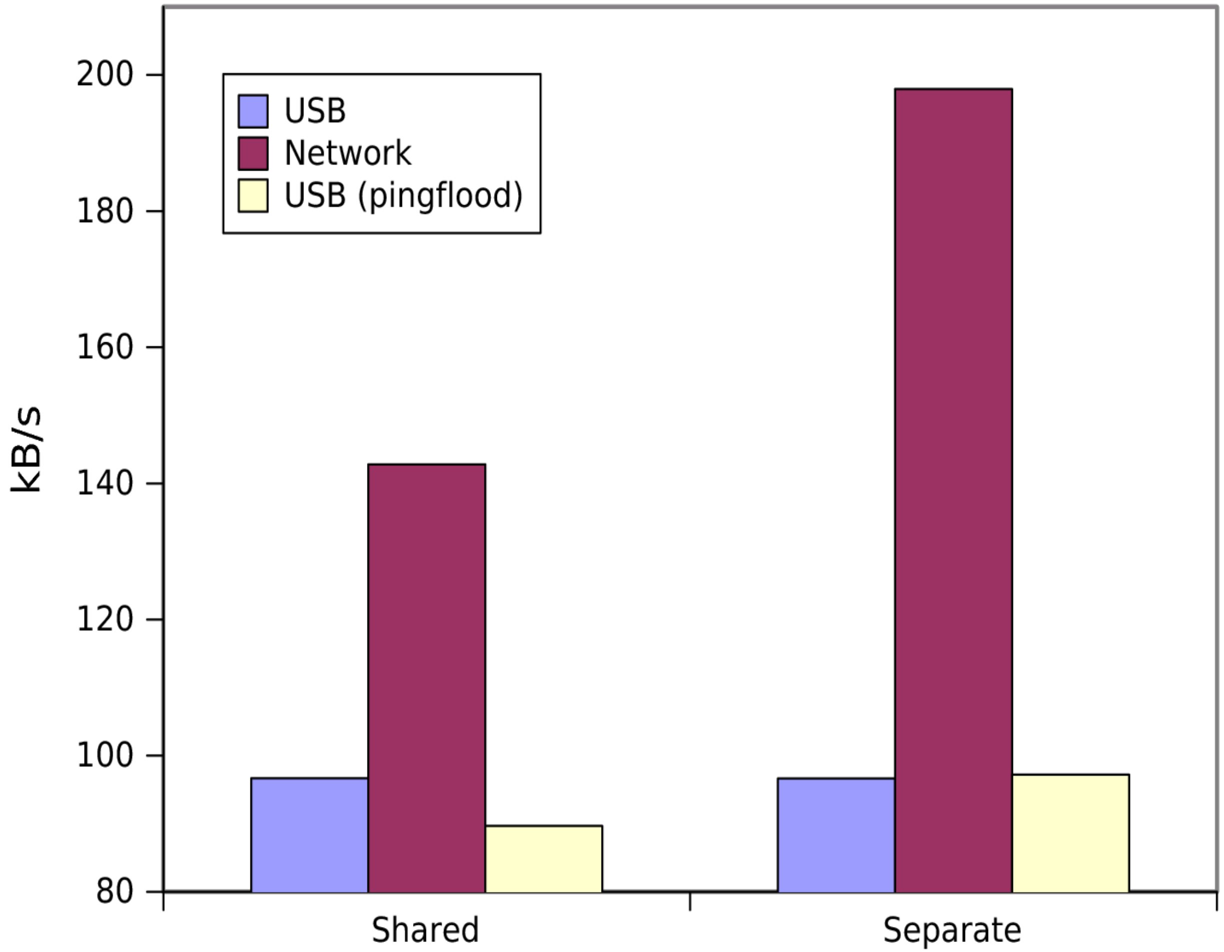
Network bandwidth of two scheduling policies

IO VCPU Sharing

VCPU	V_C	V_T	threads
VCPU0	30	100	USB, CPU-bound
VCPU1	10	110	CPU-bound
VCPU2	10	90	Network, CPU-bound
VCPU3	100	200	Logging, CPU-bound
IO VCPU	1%		USB,Network

VCPU0	30	100	USB, CPU-bound
VCPU1	10	110	CPU-bound
VCPU2	10	90	Network, CPU-bound
VCPU3	100	200	Logging, CPU-bound
IO VCPU1	1%		USB
IO VCPU2	1%		Network

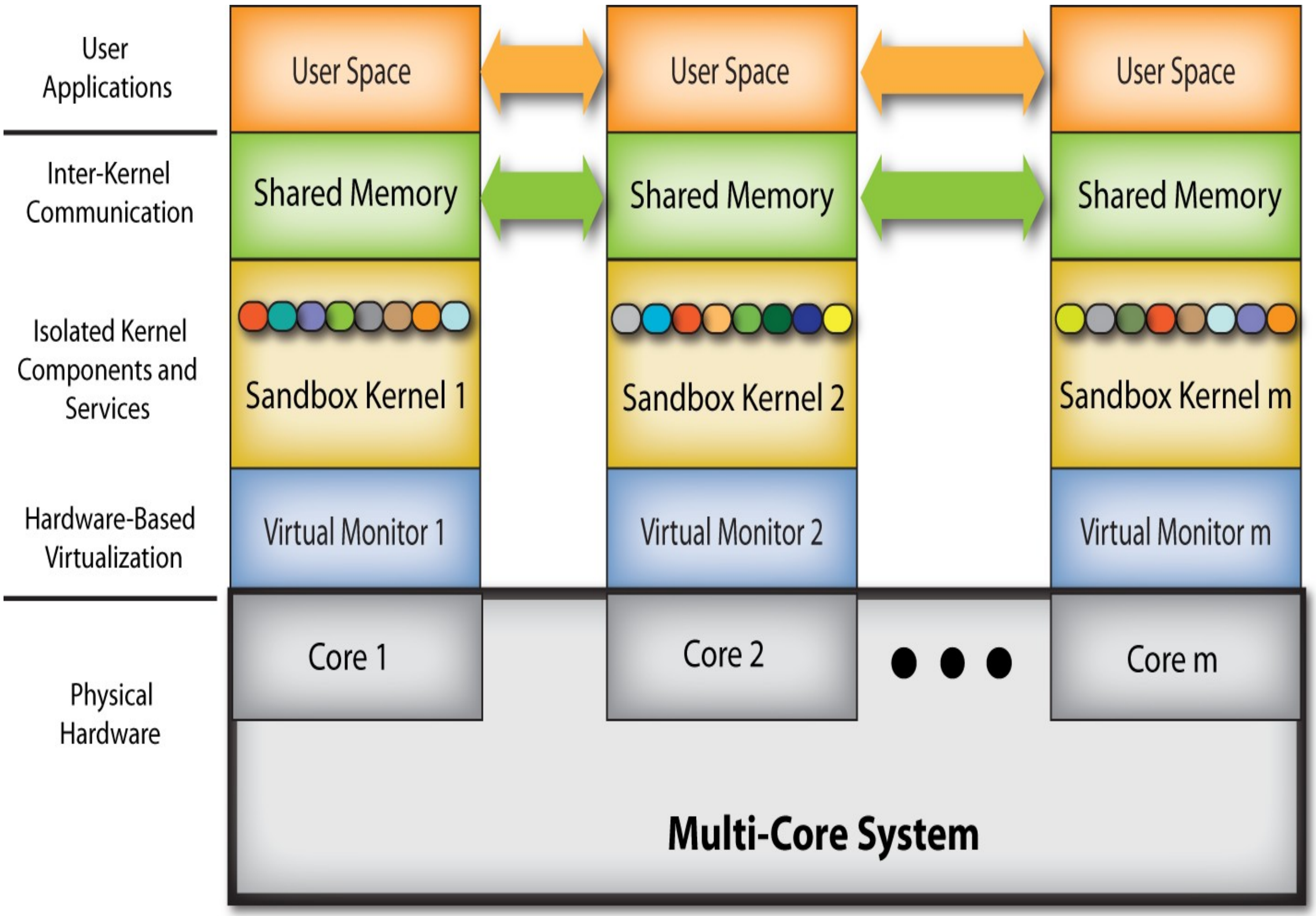
IO VCPU Sharing



Conclusions

- Temporal isolation on IO events and tasks
- PIBS + SS Main & IO VCPUs can guarantee utilization bounds
- Future investigation of higher-level policies
- Future investigation of h/w performance counters for VCPU-to-PCPU scheduling

Architecture Overview



Example Fault Recovery