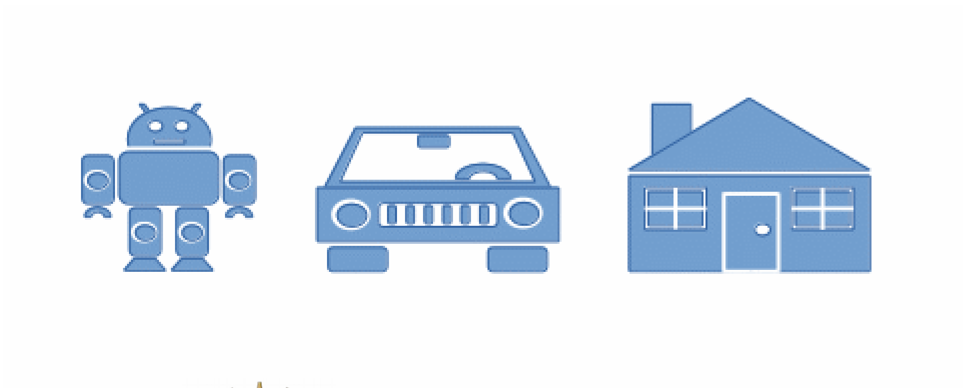
The Quest OS for Real-Time Computing

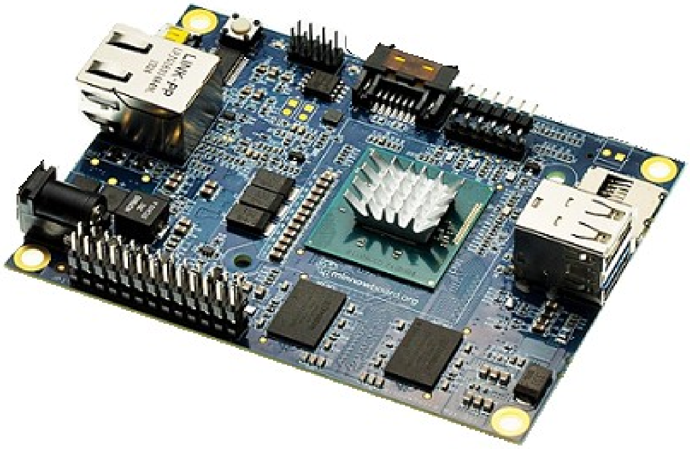
Richard West

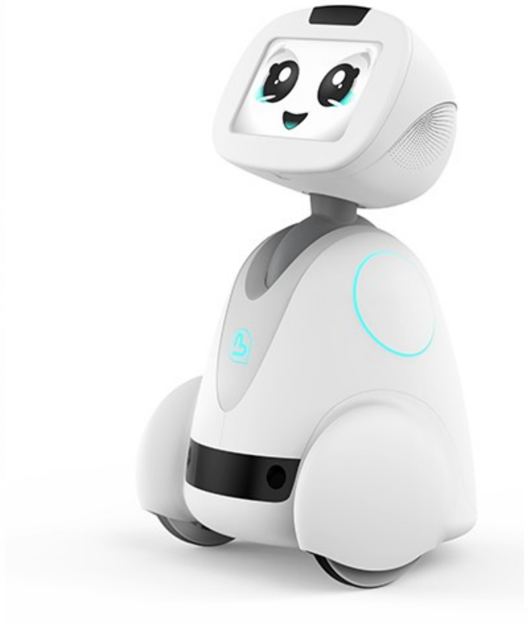
richwest@cs.bu.edu

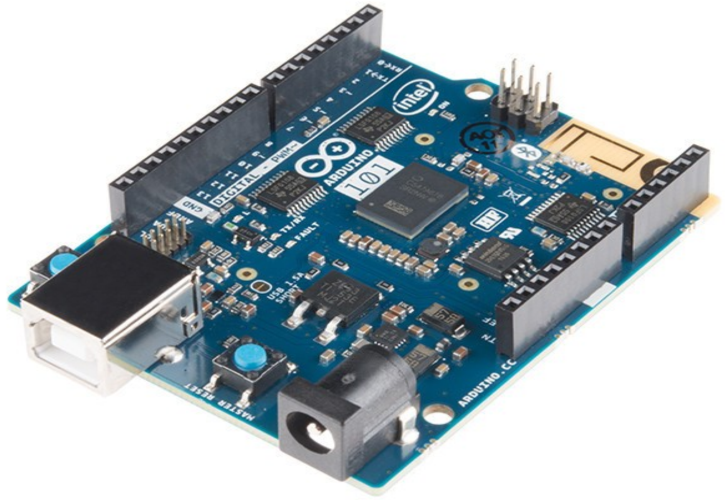


Computer Science

Emerging “Smart” Devices







Goals

- High-confidence (embedded) systems
  - **Mixed criticalities** – timeliness and safety
  - **Predictable** – real-time support
  - **Secure** – resistant to component failures & malicious attacks
  - **Fault tolerant** – online recovery from soft errors and timing violations

Target Applications

- Healthcare
- Avionics
- Automotive
- Factory automation
- Robotics
- Space exploration
- Internet-of-Things (IoT)
- Industry 4.0 “smart factories”

Internet of Things

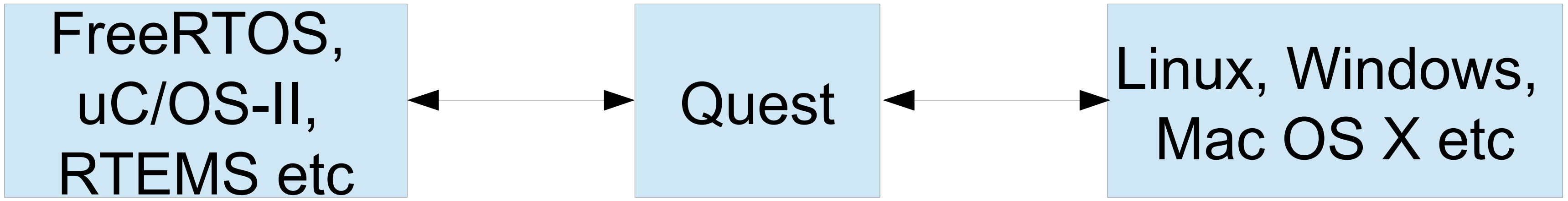
- Number of Internet-connected devices
  > 12.5 billion in 2010
- World population > 7 billion (2015)
- Cisco predicts 50 billion Internet devices by 2020

Challenges:

- **Secure** management of data
- **Reliable** + **predictable** data exchange
- Device **interoperability**

In the Beginning...Quest

- Initially a “small” RTOS
- ~30KB ROM image for uniprocessor version
- Page-based address spaces
- Threads
- Dual-mode kernel-user separation
- Real-time Virtual CPU (VCPU) Scheduling
- Later SMP support
- LAPIC timing



From Quest to Quest-V

- 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
  - Also CPU, I/O, cache partitioning
- Focus on **safety**, **efficiency**, **predictability** + **security**

Related Work

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

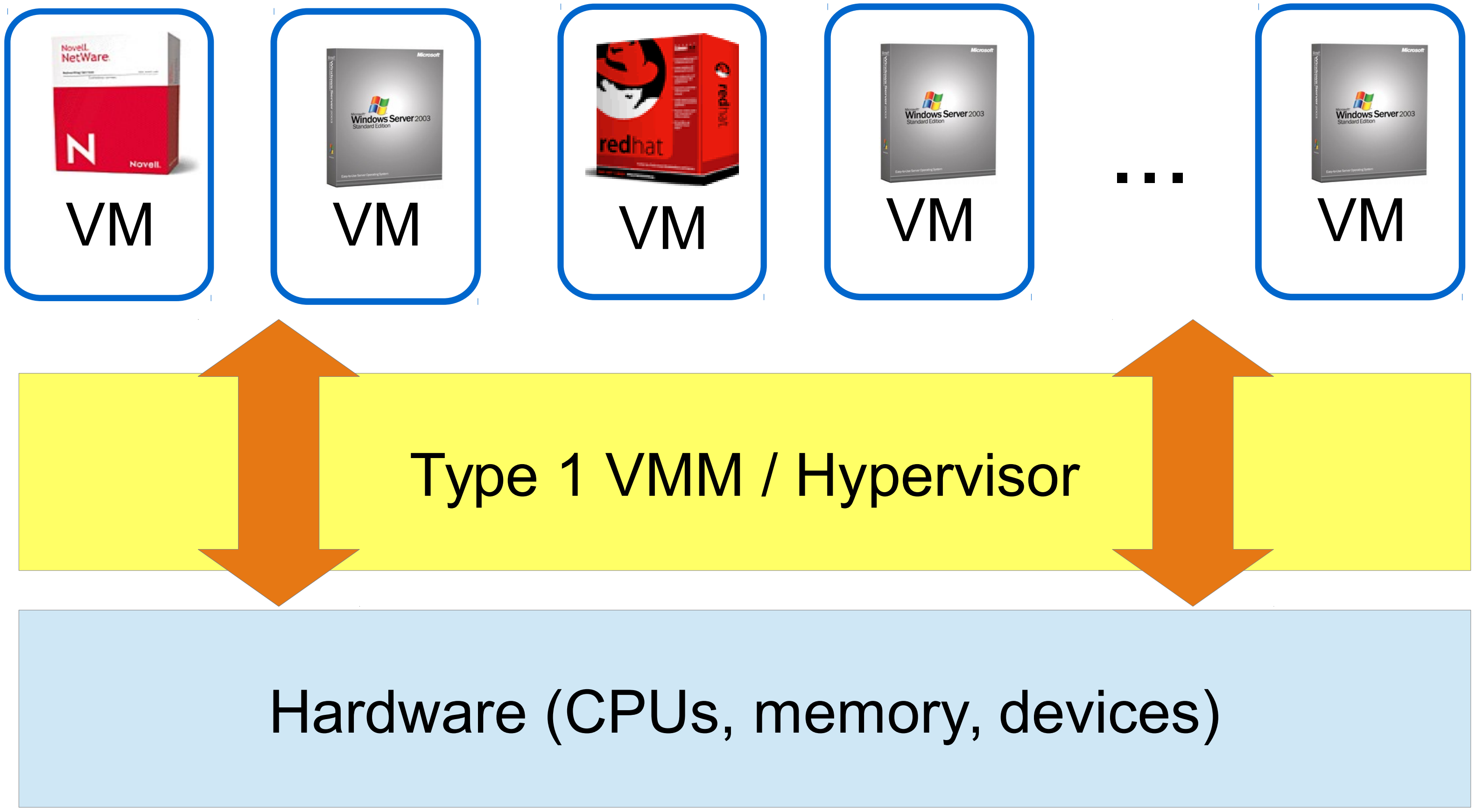
Problem

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
-

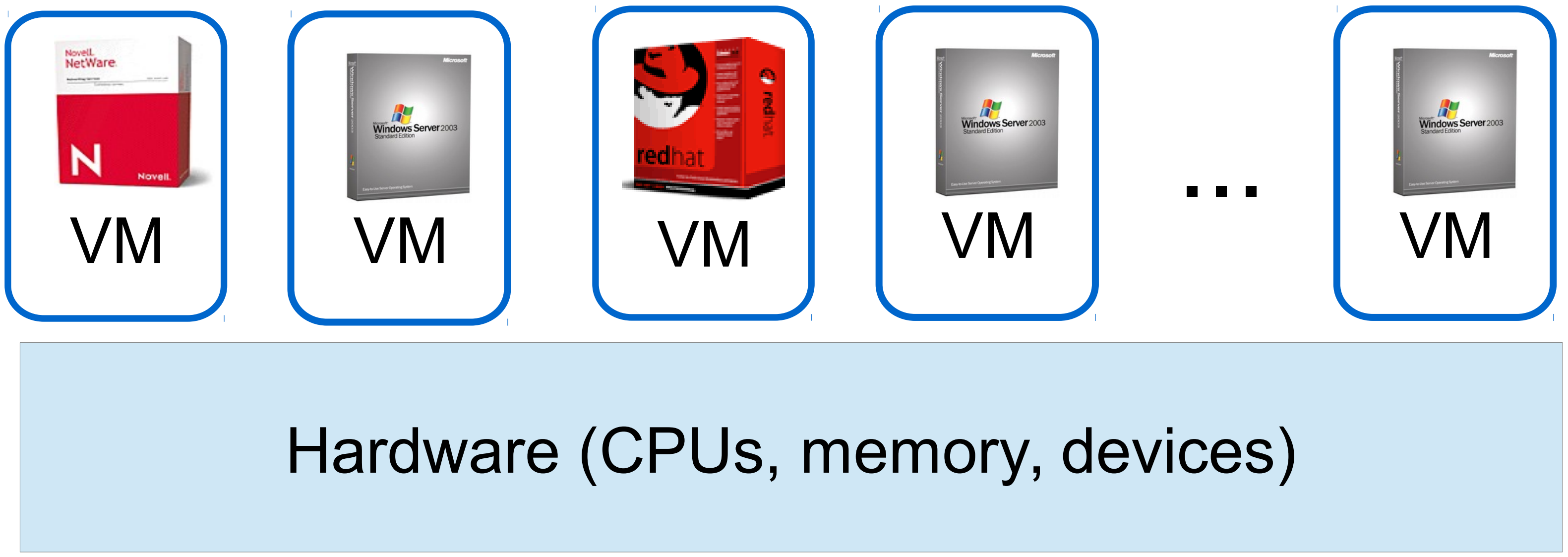
Traditional Virtual Machine approaches **too memory intensive** for many real-time embedded systems

Traditional Approach (Type 1 VMM)

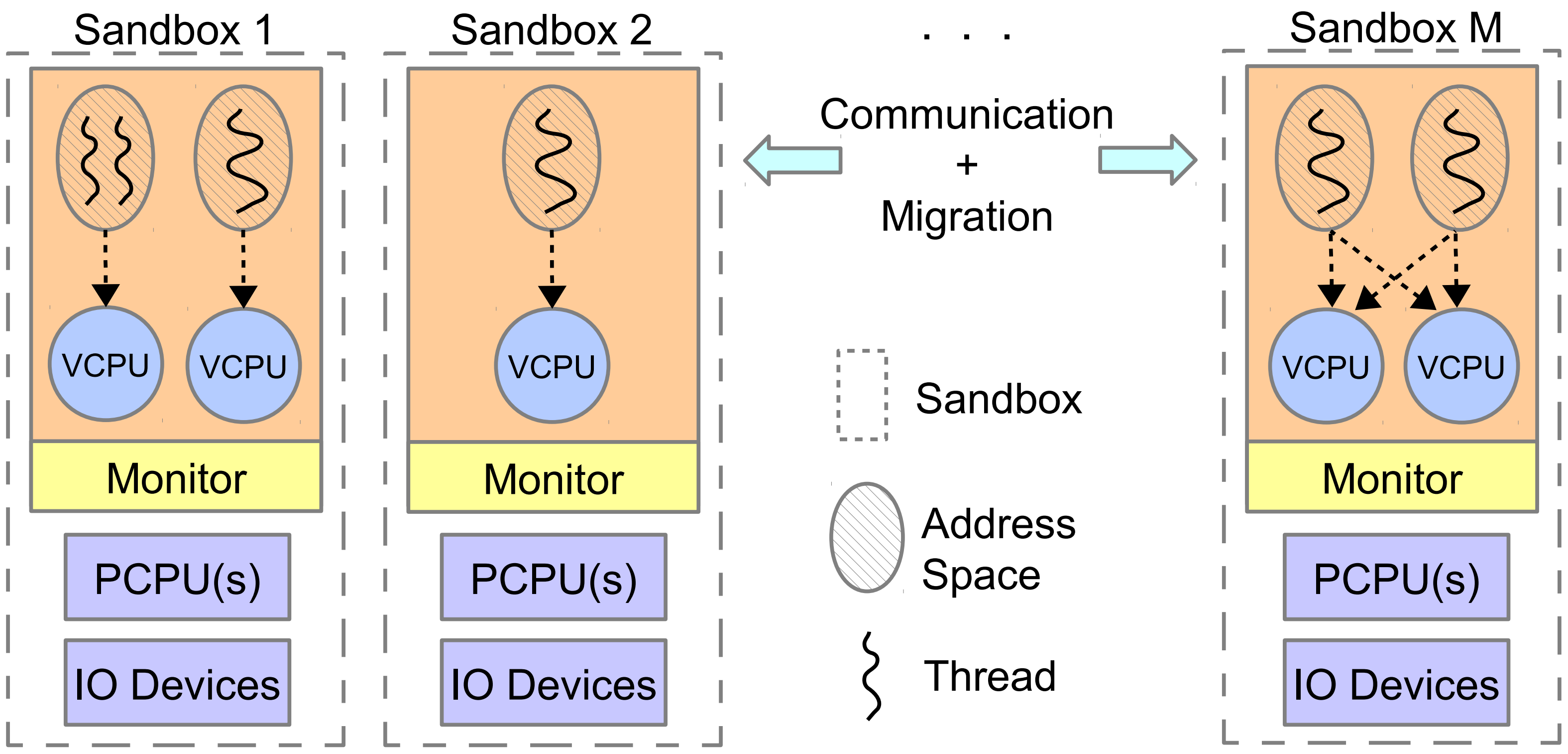


Quest-V Approach

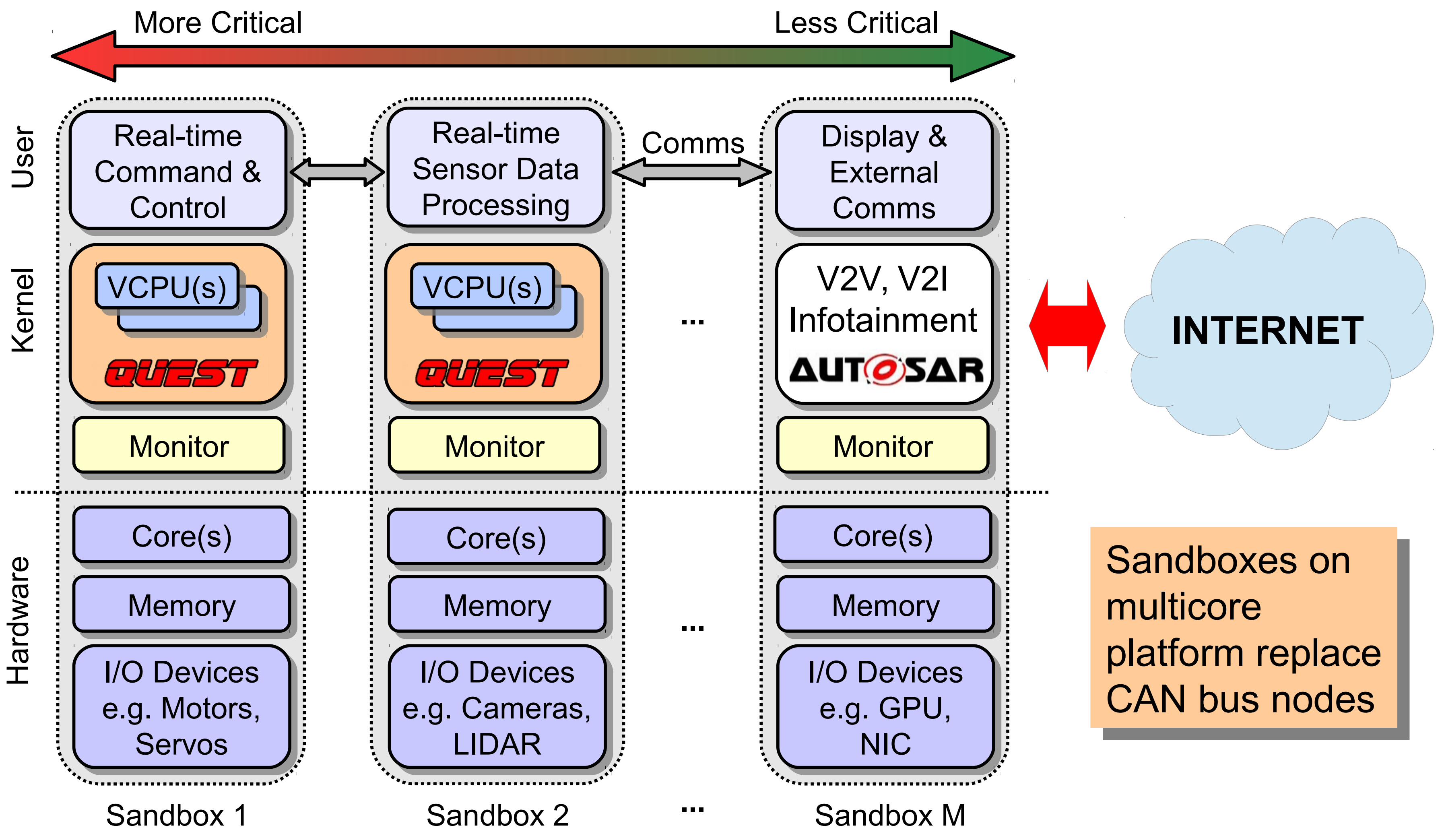
Eliminates hypervisor intervention during normal virtual machine operations



Quest-V Architecture Overview



Mixed-Criticality Automotive System

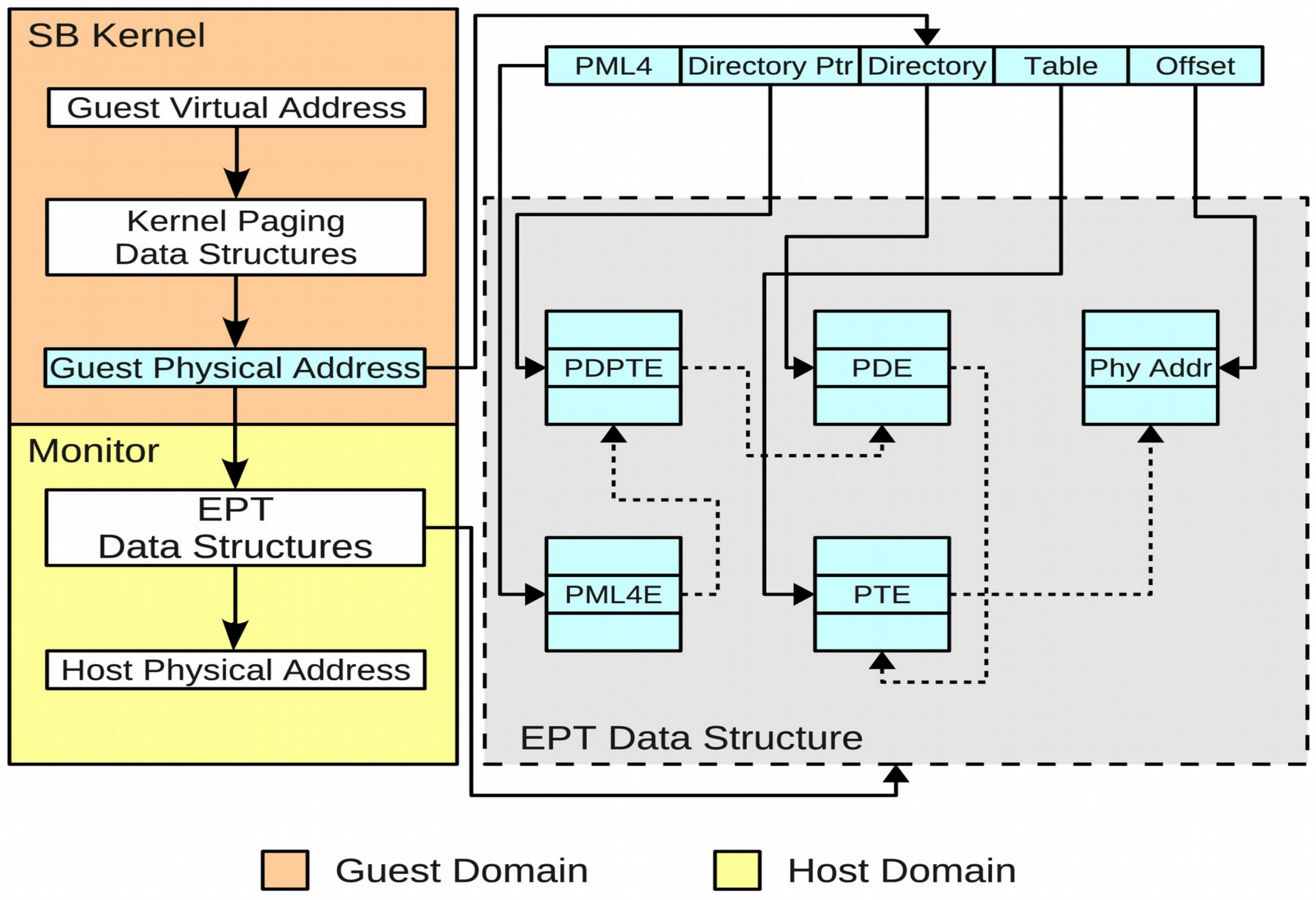


Sandboxes on multicore platform replace CAN bus nodes

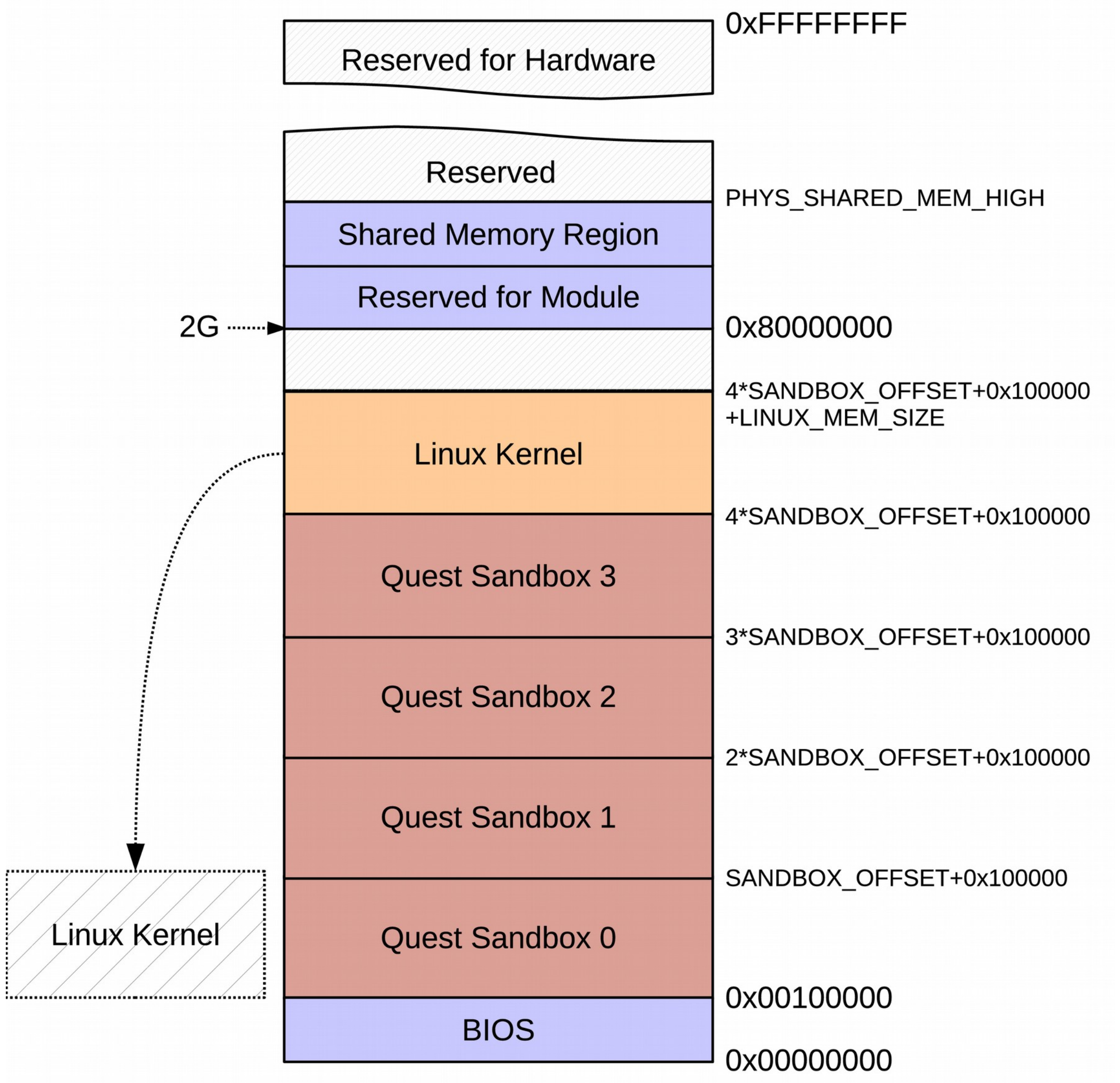
Memory Partitioning

- 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 Memory Partitioning

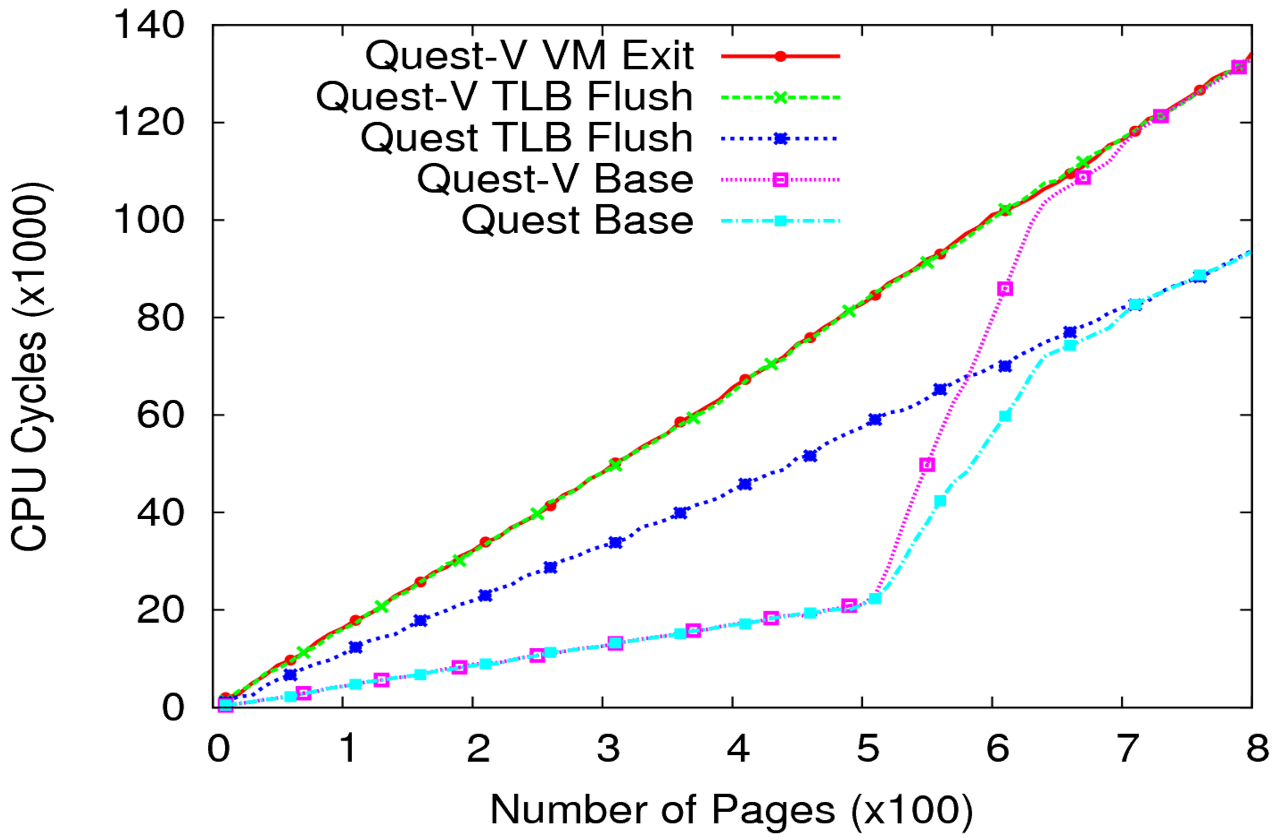


Quest-V Linux Memory Layout



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

CPU Partitioning

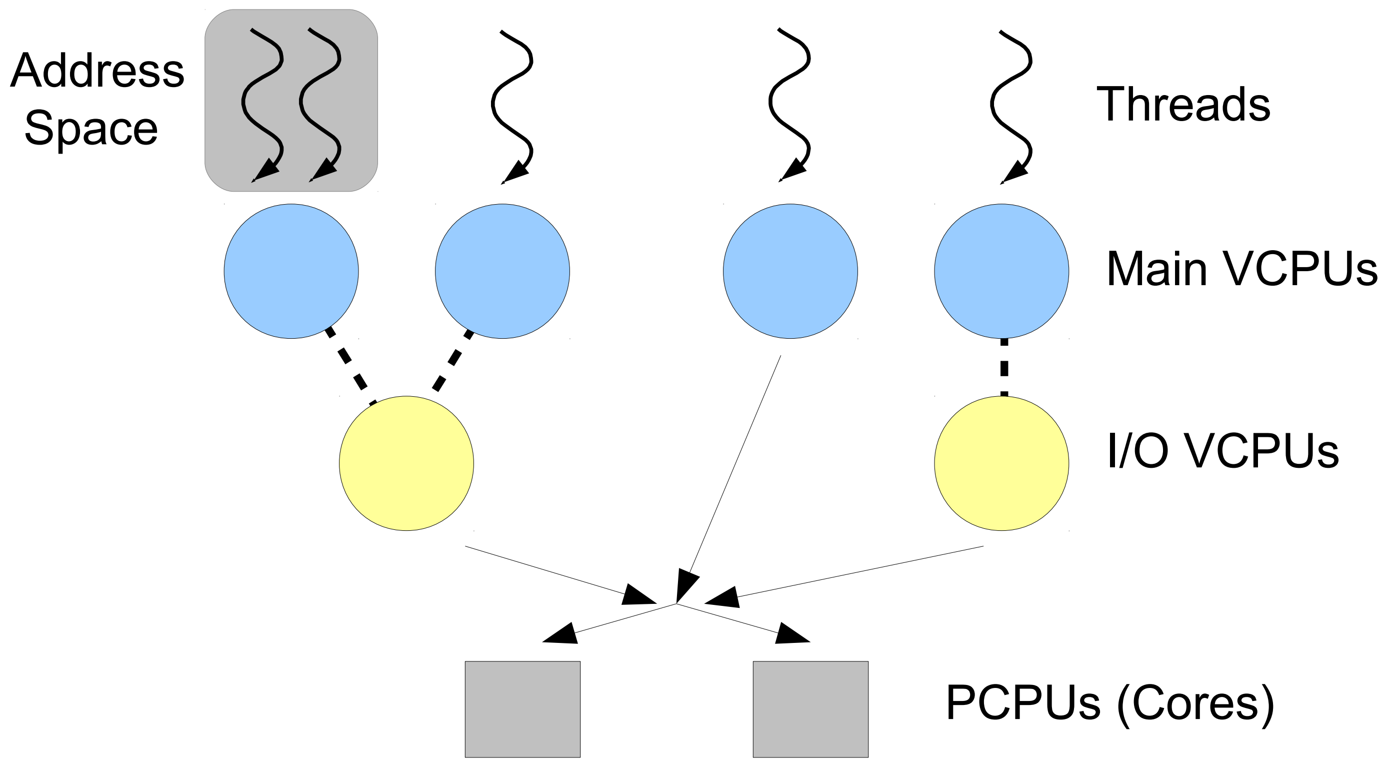
- Scheduling local to each sandbox
  - partitioned rather than global
  - avoids monitor intervention

- 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)



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

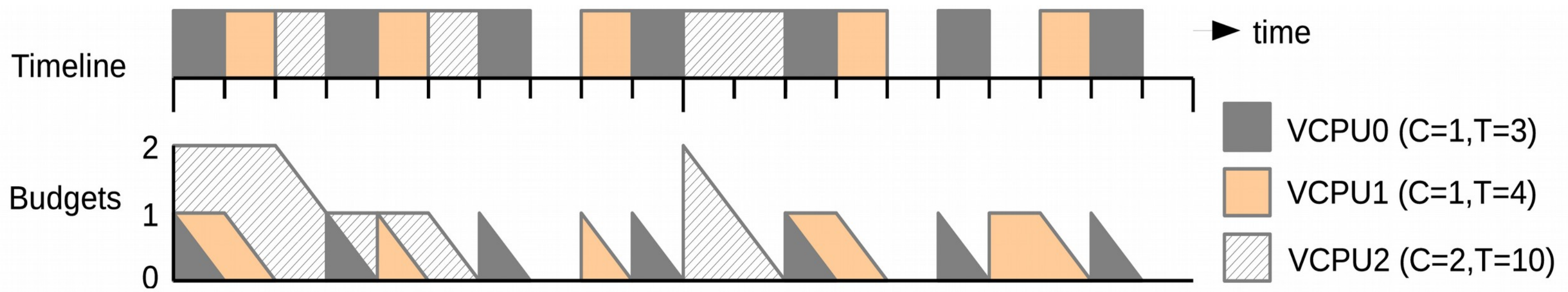
PIBS Scheduling

- 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

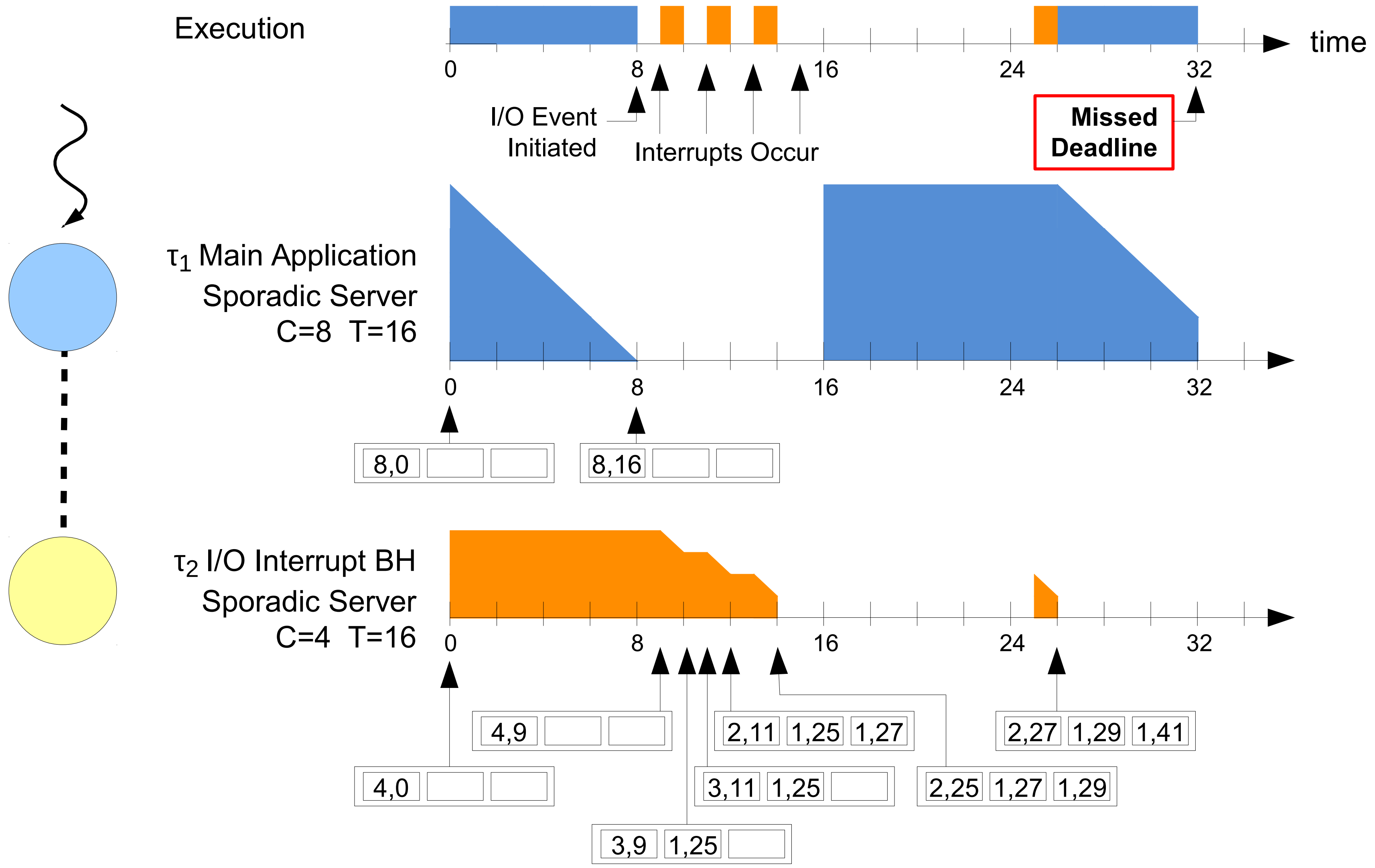
  (1) Replenishment, R must be deferred at least $t+T_V$

  (2) Can be deferred longer

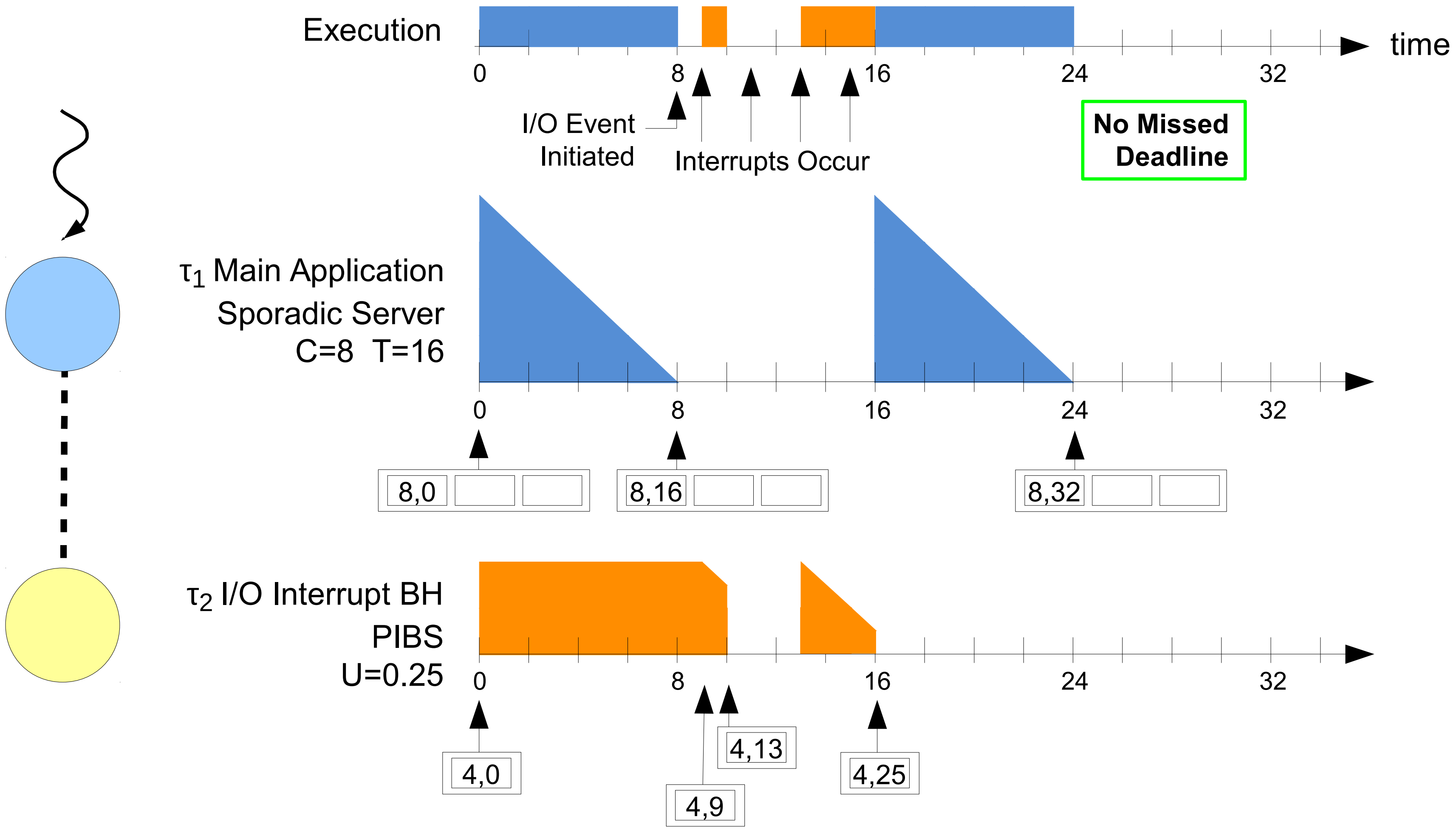
  (3) Can merge two overlapping replenishments

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

Example SS-Only Schedule



Example SS+PIBS Schedule



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

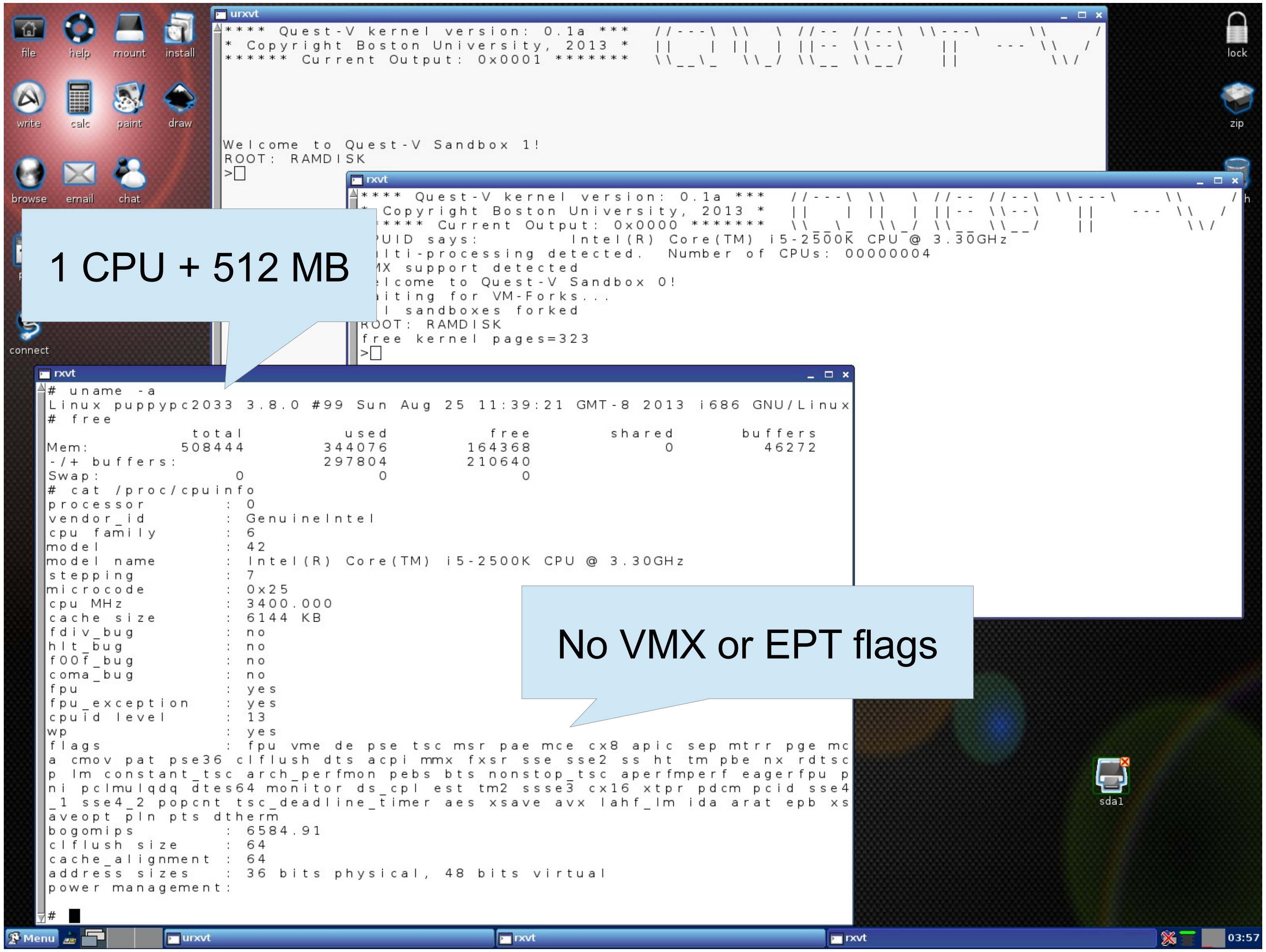
- Shared caches controlled using color-aware memory allocator **[COLORIS – PACT'14]**
- Cache occupancy prediction based on h/w performance counters
  - $E' = E + (1-E/C) * m_l - E/C * m_o$
  - Enhanced with hits + misses

    [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



1 CPU + 512 MB

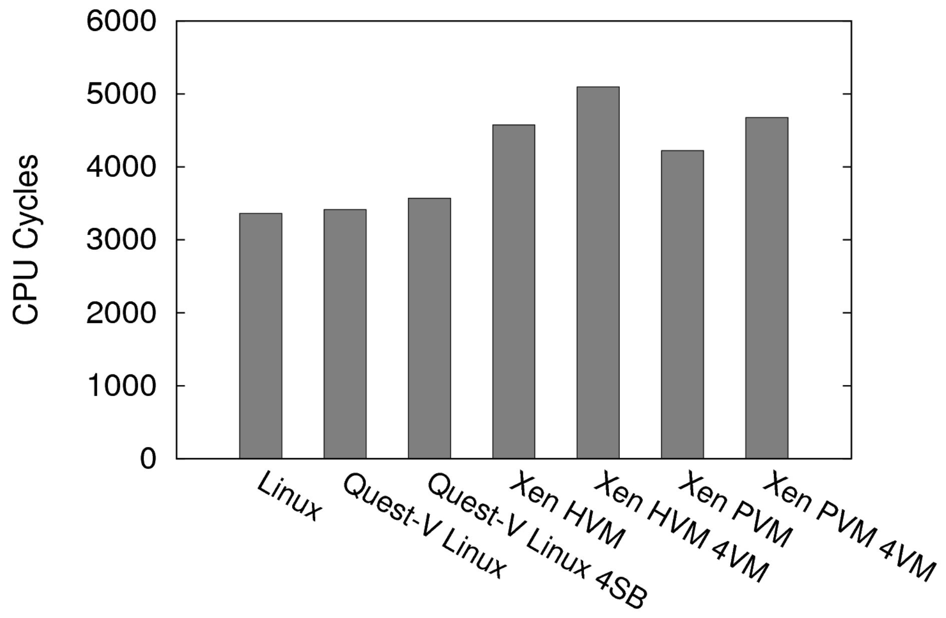
No VMX or EPT flags

```
**** Quest-V kernel version: 0.1a ***   //---\ \\   \ //-- //--\ \\---\     \\    /
*  Copyright Boston University, 2013 *  ||    | ||   | ||-- \\--\    ||    --- \\   /
****** Current Output: 0x0001 *******   \\__\_  \\_/ \\__ \\__/    ||          \\/



Welcome to Quest-V Sandbox 1!
ROOT: RAMDISK
>
```

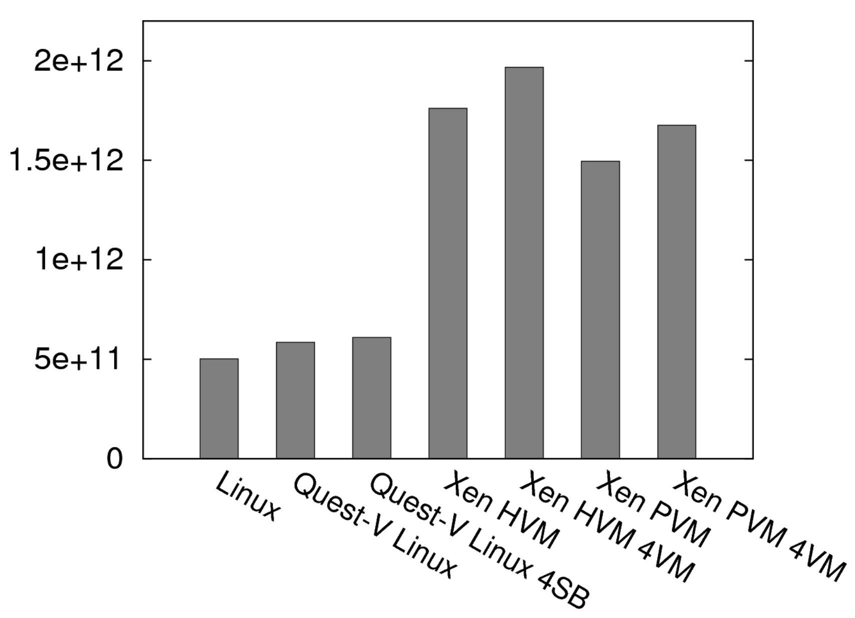
```
**** Quest-V kernel version: 0.1a ***   //---\ \\   \ //-- //--\ \\---\     \\    /
*  Copyright Boston University, 2013 *  ||    | ||   | ||-- \\--\    ||    --- \\   /
***** Current Output: 0x0000 *******    \\__\_  \\_/ \\__ \\__/    ||          \\/
UID says:          Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz
lti-processing detected.  Number of CPUs: 00000004
MX support detected
elcome to Quest-V Sandbox 0!
iting for VM-Forks...
l sandboxes forked
ROOT: RAMDISK
free kernel pages=323
>
```

```
# uname -a
Linux puppypc2013 3.8.0 #99 Sun Aug 25 11:39:21 GMT-8 2013 i686 GNU/Linux
# free
             total       used       free     shared    buffers
Mem:        508444     344076     164368          0      46272
-/+ buffers:           297804     210640
Swap:            0          0          0
# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 42
model name      : Intel(R) Core(TM) i5-2500K CPU @ 3.30GHz
stepping        : 7
microcode       : 0x25
cpu MHz         : 3400.000
cache size      : 6144 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mc
a cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx rdtsc
p lm constant_tsc arch_perfmon pebs bts nonstop_tsc aperfmperf eagerfpu p
ni pclmulqdq dtes64 monitor ds_cpl est tm2 ssse3 cx16 xtpr pdcm pcid sse4
_1 sse4_2 popcnt tsc_deadline_timer aes xsave avx lahf_lm ida arat epb xs
aveopt pln pts dtherm
bogomips        : 6584.91
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

# 
```

Quest-V Performance



100 Million Page Faults



1 Million *fork-exec-exit* Calls

Quest-V Summary

- 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

Qduino

- **Qduino** – Enhanced Arduino API for Quest
  - **Parallel** and **predictable** loop execution
  - **Real-time** communication b/w loops
  - Predictable and efficient interrupt management
  - Real-time event delivery
  - **Backward compatible** with Arduino API
  - Simplifies multithreaded real-time programming

Interleaved Sketches

```
// Sketch 1: toggle GPIO pin 9
// every 2s
int val9 = 0;

void setup() {
	pinMode(9, OUTPUT);
}

void loop() {
	val9 = !val9; //flip the output value
	digitalWrite(9, val9);
	delay(2000); //delay 2s
}
```

```
//Sketch 2: toggle pin 10 every 3s
int val10 = 0;

void setup() {
	pinMode(10, OUTPUT);
}

void loop() {
	val10 = !val10; //flip the output
value
	digitalWrite(10, val10);
	delay(3000); //delay 3s
}
```

How do you merge the sketches
and keep the correct delays?

Interleaved Sketches

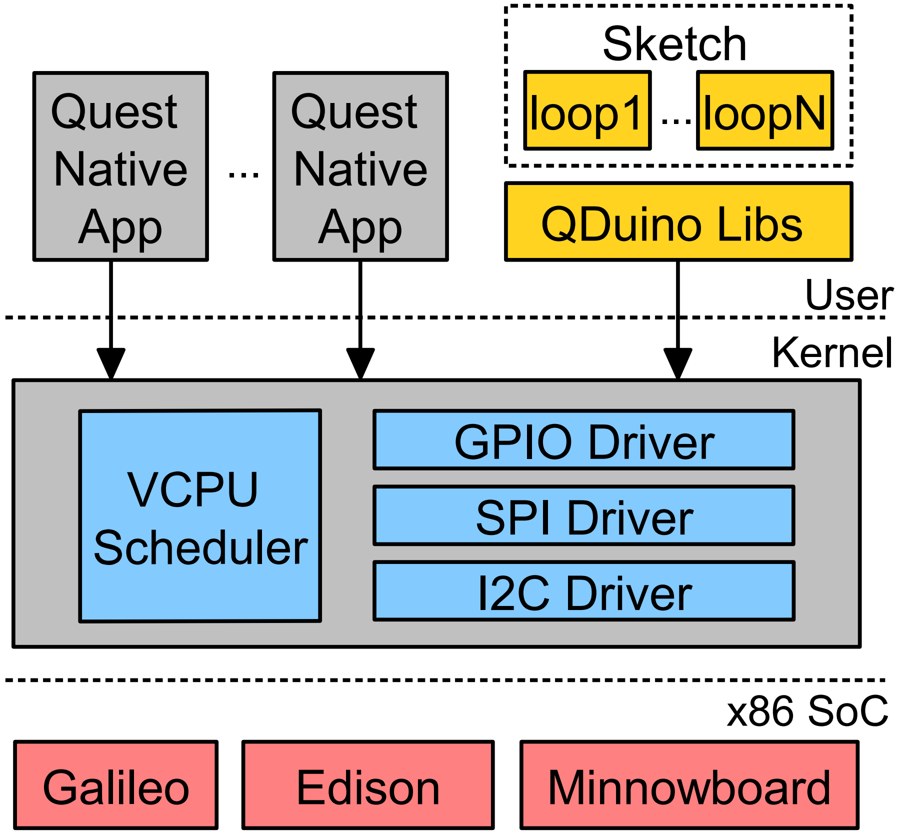
- Do scheduling by hand
- Inefficient
- Hard to scale

```
int val9, val10 = 0;
int next_flip9, next_flip10 = 0;

void setup() {
	pinMode(9, OUTPUT);
	pinMode(10, OUTPUT);
}
void loop() {
	if (millis() >= next_flip9) {
		val9 = !val9; //flip the output value
		digitalWrite(9, val9);
		next_flip9 += 2000;
	}
	if (millis() >= next_flip10) {
		val10 = !val10; //flip the output value
		digitalWrite(10, val10);
		next_flip10 += 3000;
	}
}
```

Qduino Multi-threaded Sketch

```
int val9, val10 = 0;
int C = 500, T = 1000;

void setup() {
        pinMode(9, OUTPUT);
        pinMode(10, OUTPUT);
}

void loop(1, C, T)  {
        val9 = !val9; // flip the output value
        digitalWrite(9, val9);
        delay(2000);
}

void loop(2, C, T)  {
        val10 = !val10; // flip the output value
        digitalWrite(10, val10);
        delay(3000);
}
```

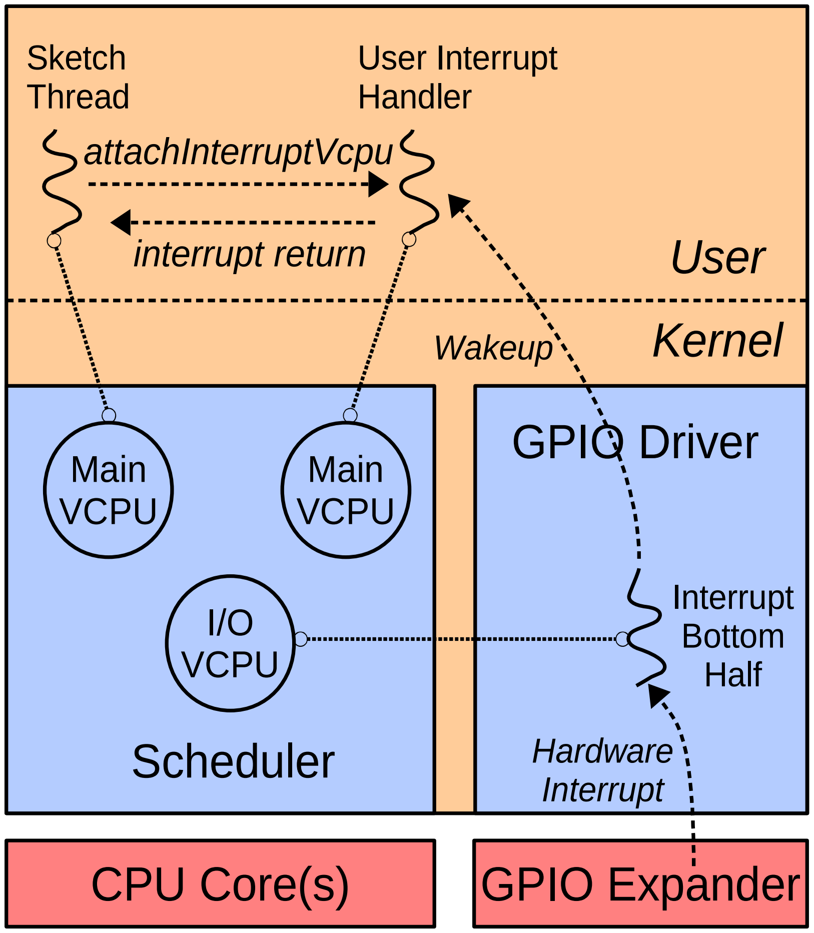
Qduino Organization



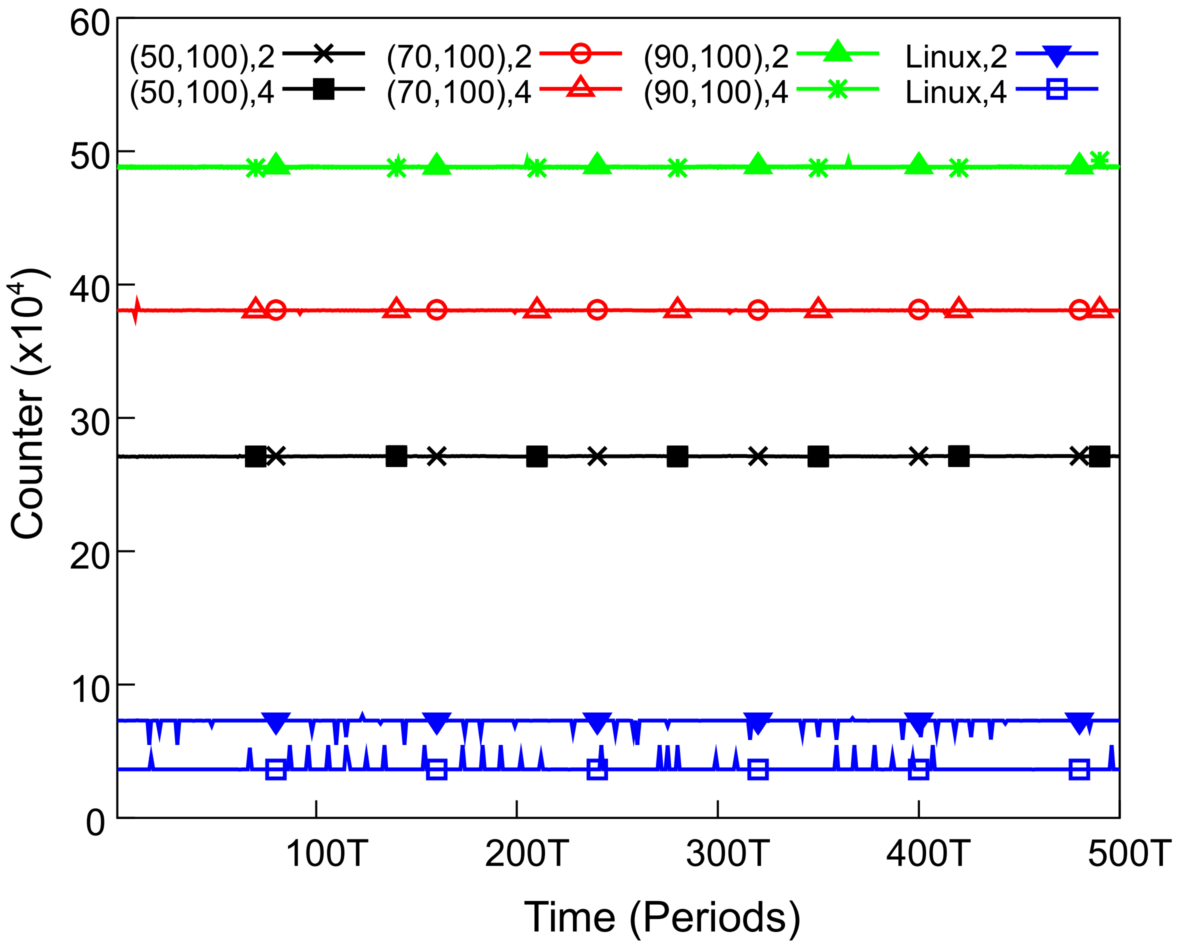
Qduino New APIs

Function Signatures	Category
• loop(loop_id, C, T)	Structure
• interruptsVcpu(C,T) ← **I/O VCPU** • attachInterruptVcpu(pin,ISR,mode,C,T) ←**Main VCPU**	Interrupt
• spinlockInit(lock) • spinlockLock(lock) • spinlockUnlock(lock)	Spinlock
• channelWrite(channel,item) • item channelRead(channel)	Four-slot
• ringbufInit(buffer,size) • ringbufWrite(buffer,item) • ringbufRead(buffer,item)	Ring buffer

Qduino Event Handling



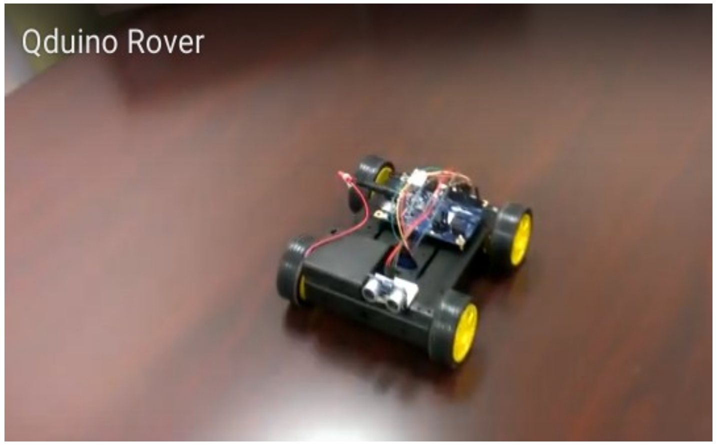
Qduino Temporal Isolation



- Foreground loop increments counter during loop period
- 2-4 background loops act as potential interference, consuming remaining CPU capacity
- No temporal isolation or timing guarantees w/ Linux

Qduino Rover

- Autonomous Vehicle
  - Collision avoidance using ultrasonic sensor

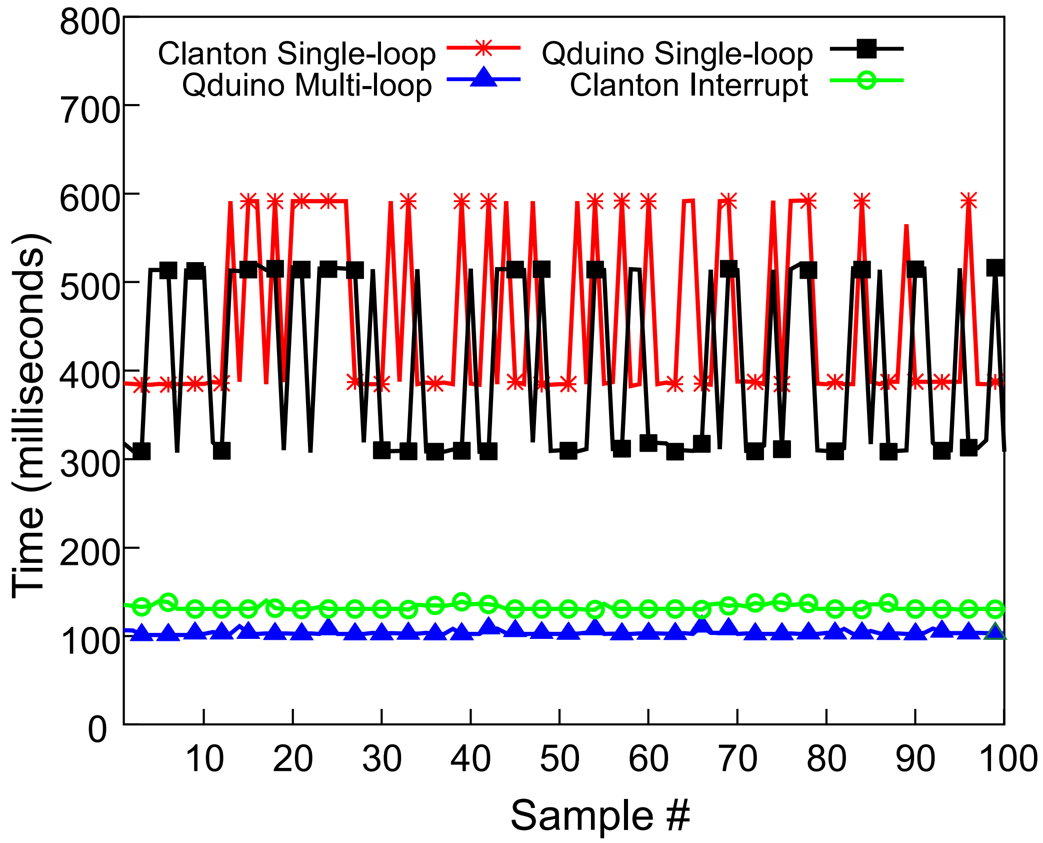


- Two tasks:
  - A **sensing task** detects distance to an obstacle – delay(200)
  - An **actuation task** controls the motors - delay(100)

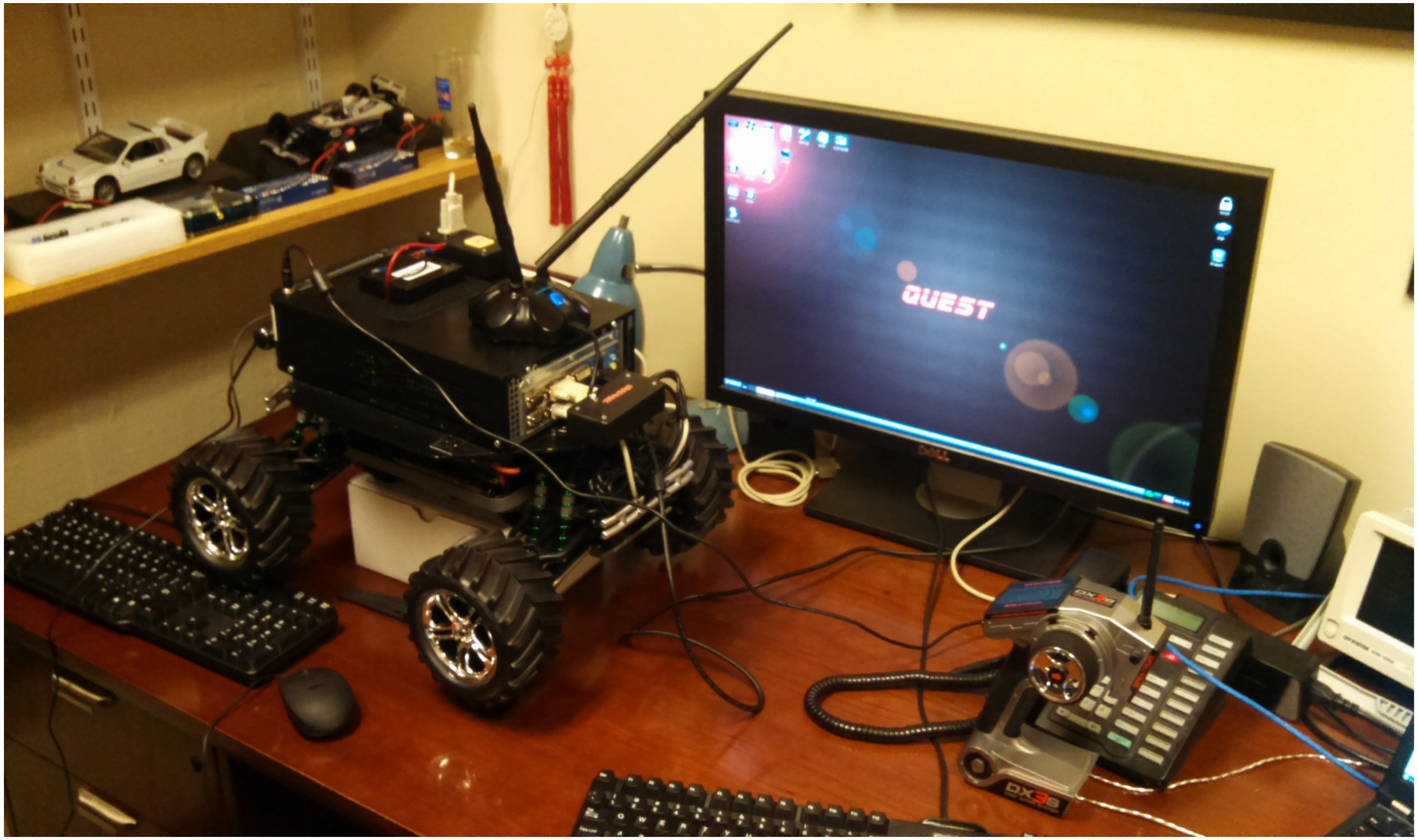
Rover Performance

- Measure the time interval between two consecutive calls to the motor actuation code
  - **Clanton Linux single loop**
    - delay from both sensing and actuation task
  - **Qduino multi-loop**
    - No delay from sensing loop
    - No delay from sensor timeout

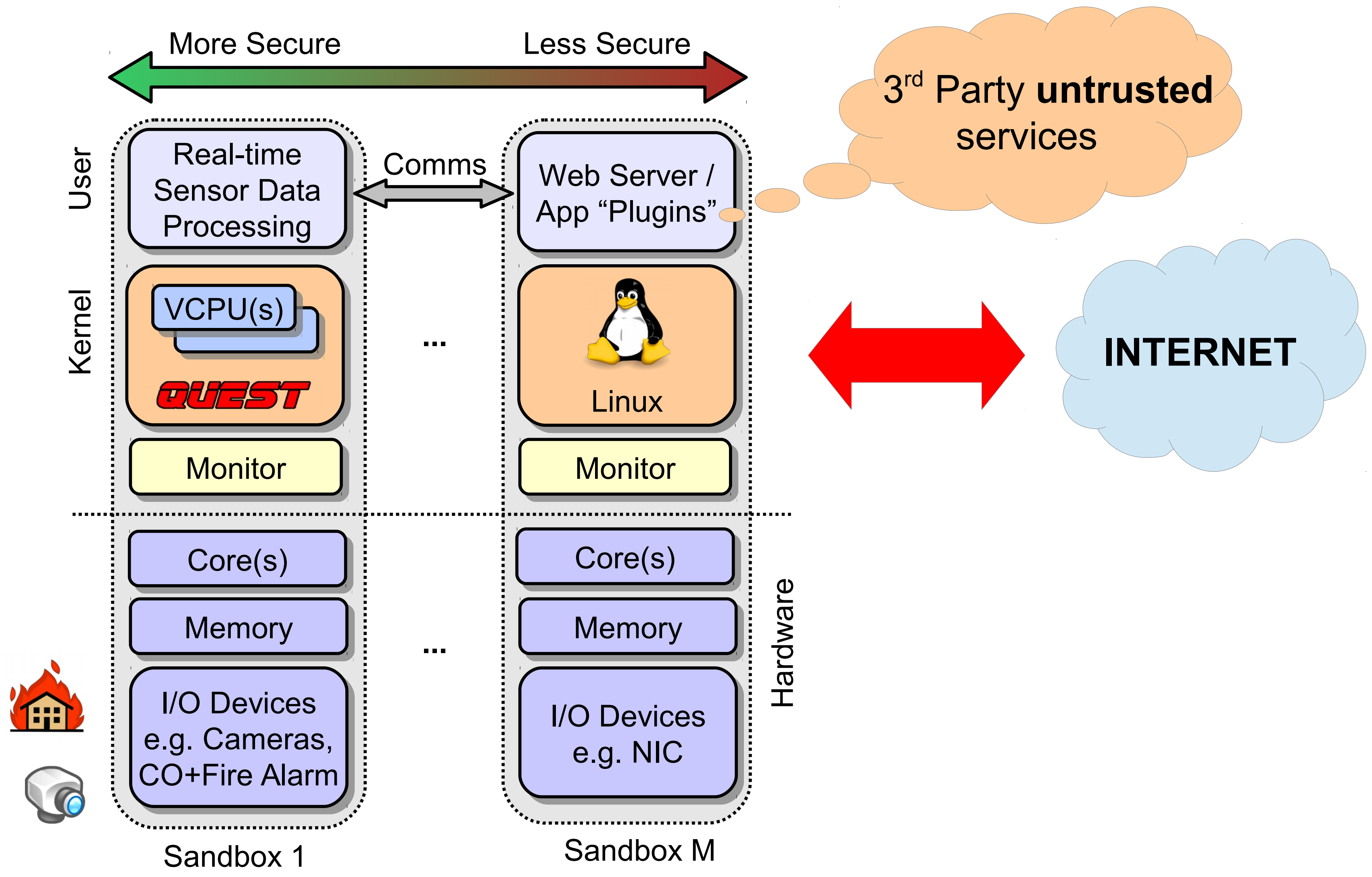
- The shorter the worst case time interval, the faster the vehicle can drive



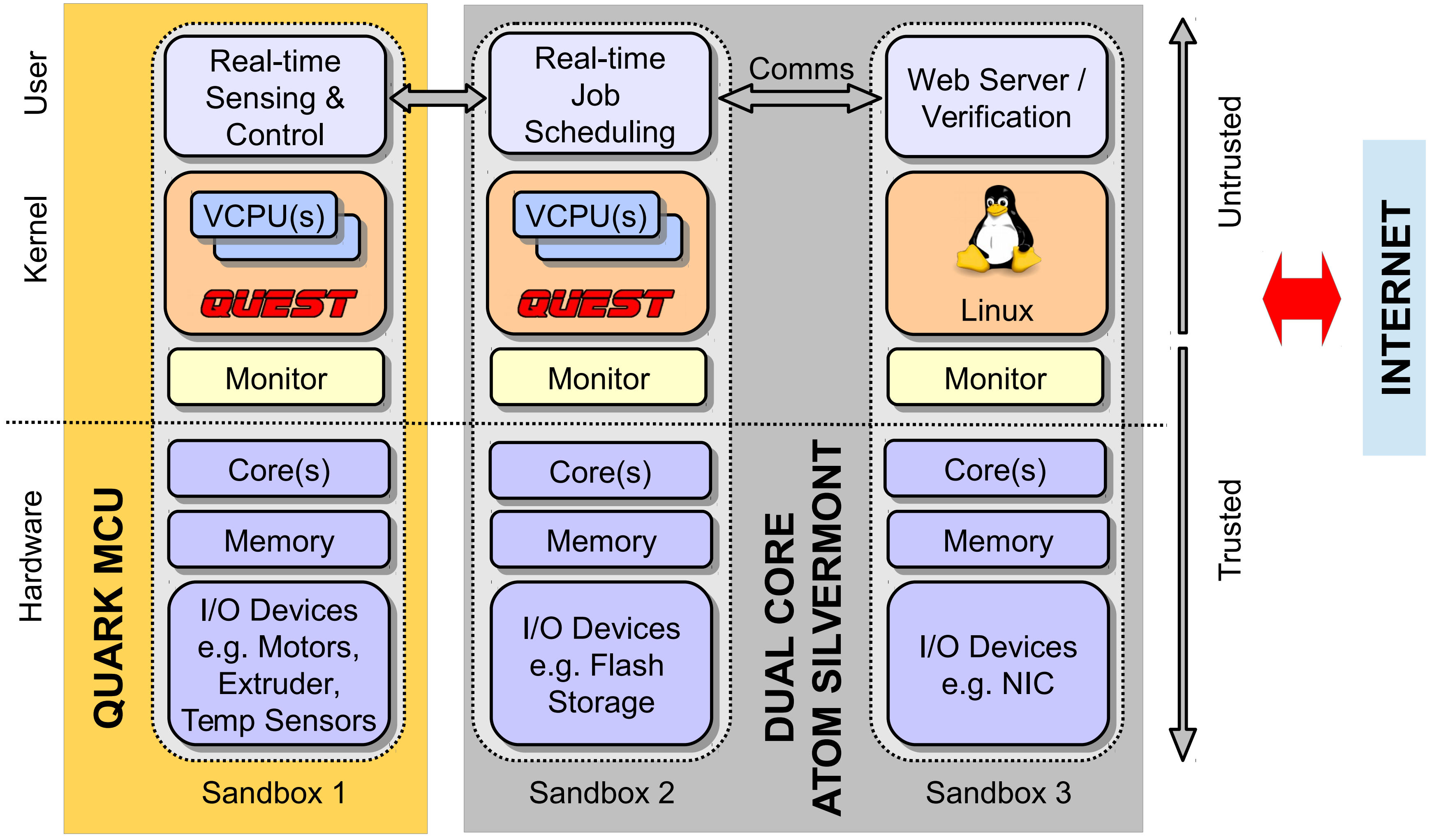
RacerX Autonomous Vehicle



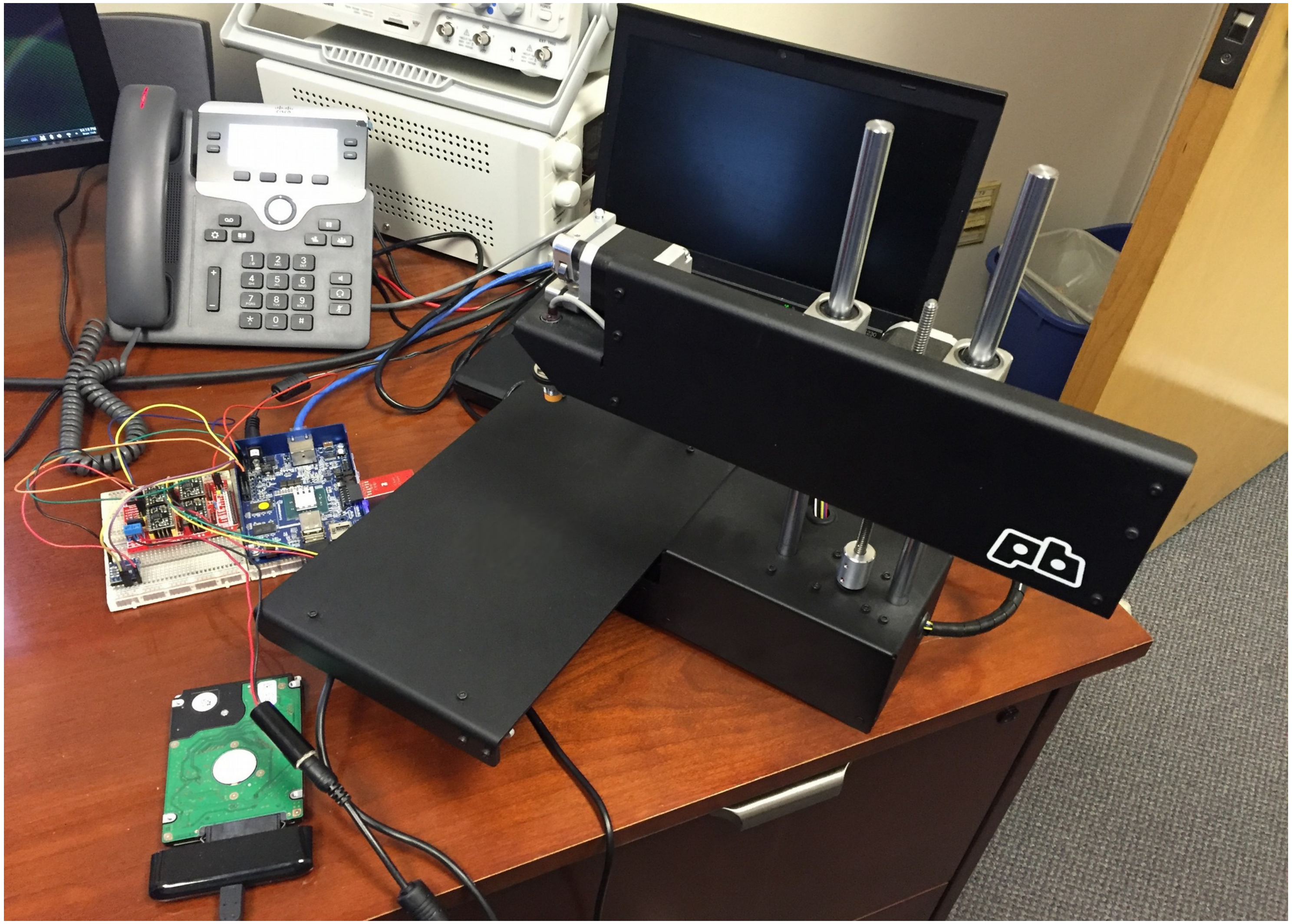
Secure Home Automation



Edison 3D Printer Controller



Minnowboard 3D Printer Controller



Demos

- Qduino Rover
- Quest-V Triple modular redundancy (TMR) fault recovery for unmanned aerial vehicle (UAV)

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

Conclusions

- Quest-V uses one monitor per sandbox
  - Heightens security & safety
  - Monitors are small
    - Not needed for resource multiplexing
- Possible to refactor legacy apps into separate components mapped to different sandboxes
  - Eases transition to a new OS
- Qduino real-time multi-loop programming

The Quest Team

- Richard West
- Ye Li
- Eric Missimer
- Matt Danish
- Gary Wong
- Ying Ye
- Zhuoqun Cheng

QUEST

BOSS