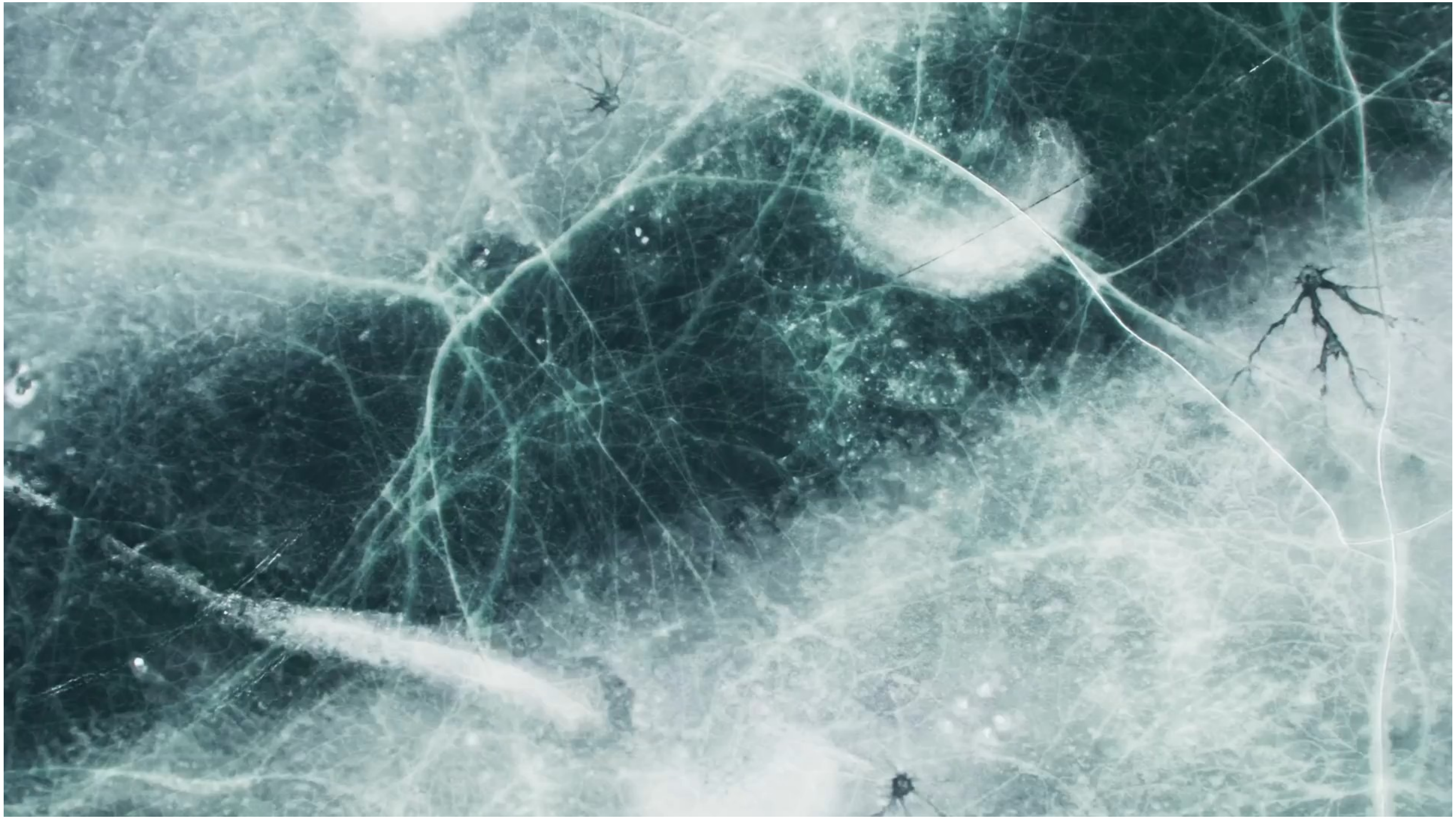
Challenges and Experiences Building a Software-Defined Vehicle Management System

Dr. Richard West
Professor, Boston University
Chief Software Architect, Drako Motors

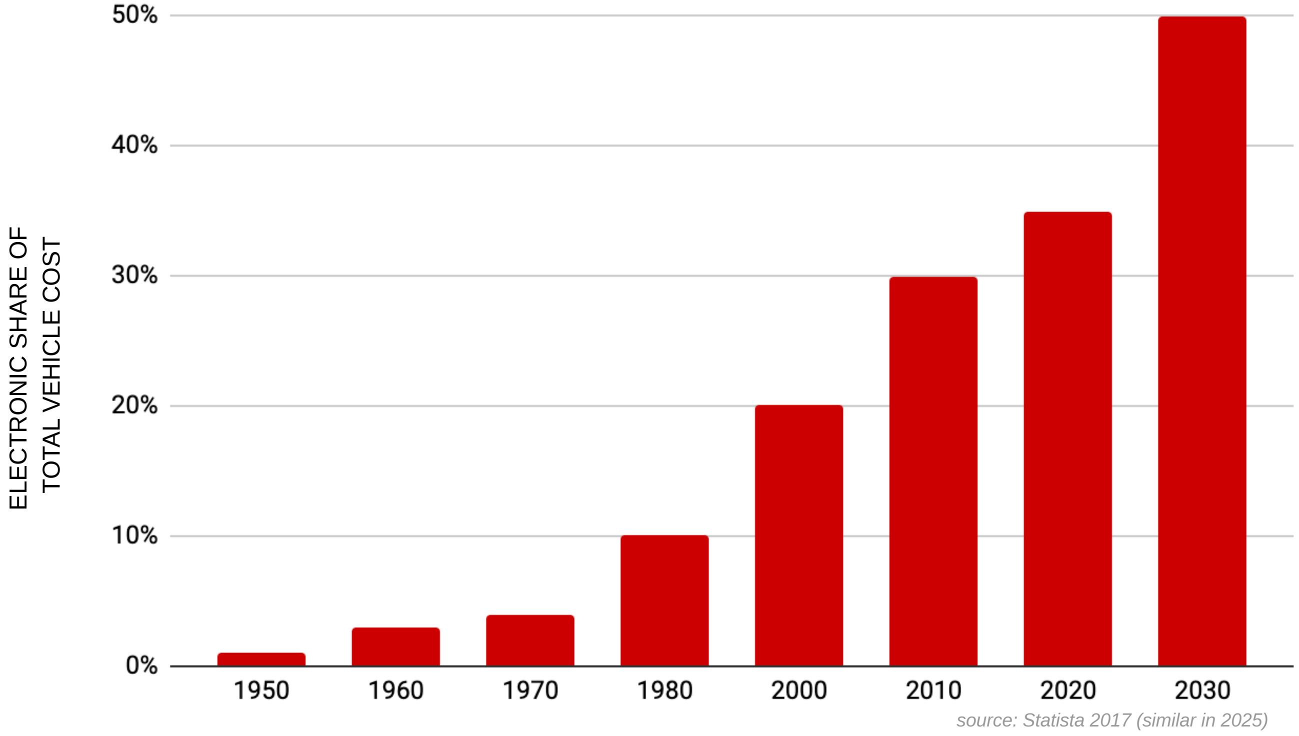
DriveOS Background

Drako GTE



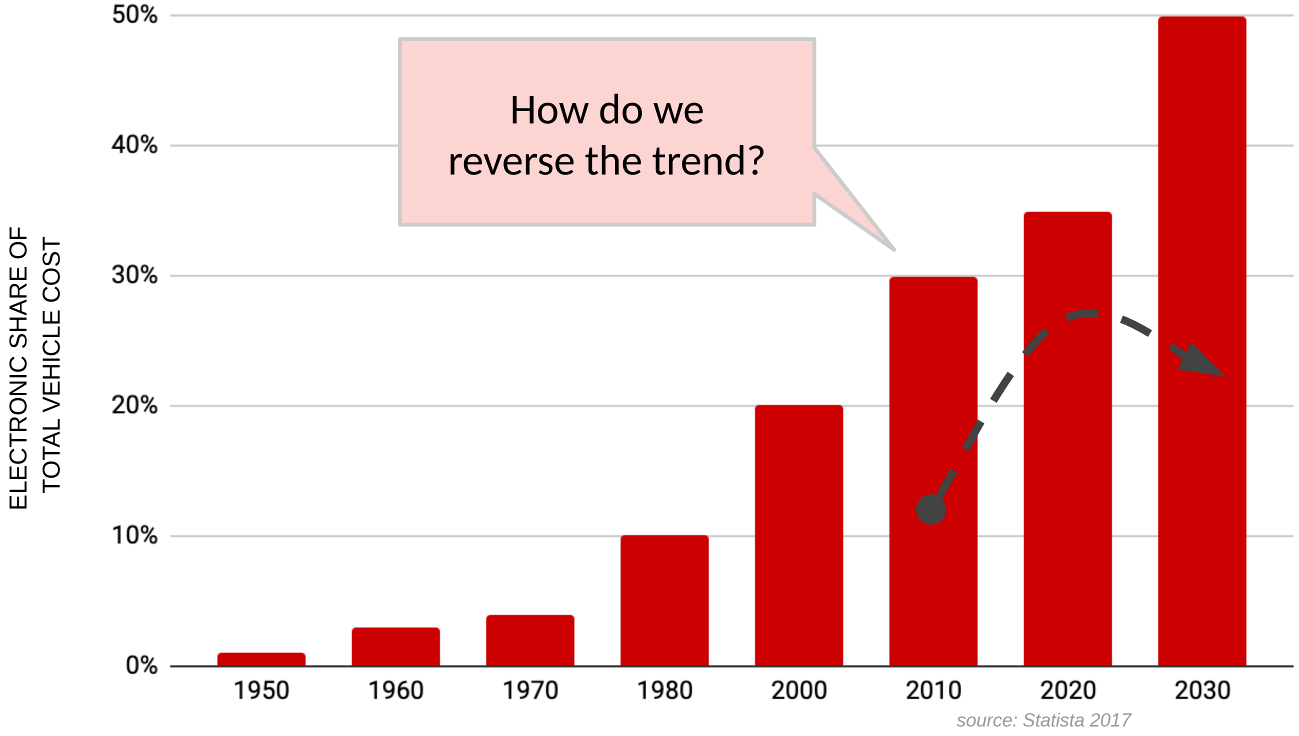
Vehicle Growth in Electronics

- Electric vehicles, ADAS, IVI, V2X driving up cost and complexity of electronics
- Modern luxury vehicles have 50-150 ECUs
  source: Strategy Analytics, IHS Markit
- Global ECU market $165.89 billion (2025)
  - Projected to be $219.19 billion (2030)
    source: Mordor Intelligence
- Electronic share of total vehicle cost is rising exponentially



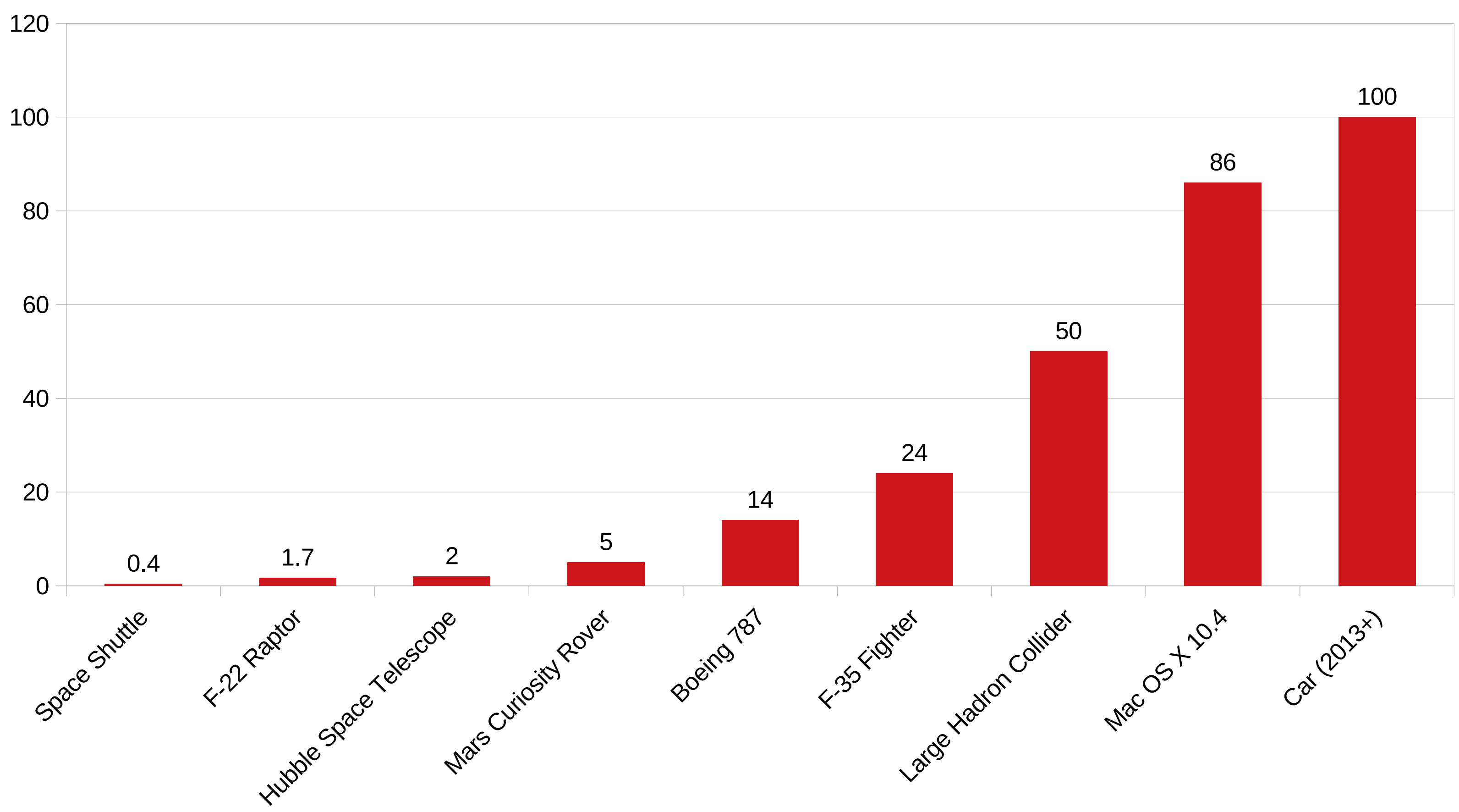
Vehicle Growth in Electronics

- Electric vehicles, ADAS, IVI, V2X driving up cost and complexity of electronics
- Modern luxury vehicles have 50-150 ECUs
  source: Strategy Analytics, IHS Markit
- Global ECU market $165.89 billion (2025)
  - Projected to be $219.19 billion (2030)
    source: Mordor Intelligence
- Electronic share of total vehicle cost is rising exponentially



Automotive Software Complexity

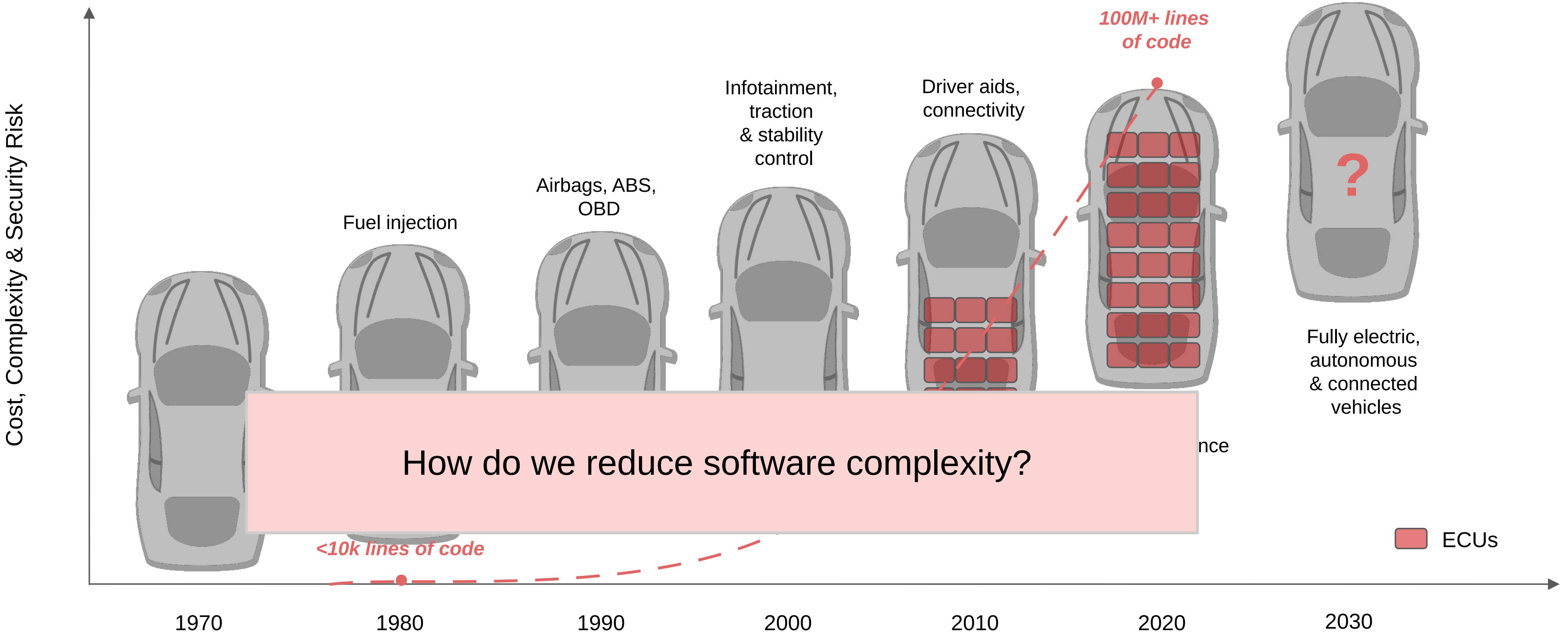
Growth in automotive electronics has given rise to growth in software complexity



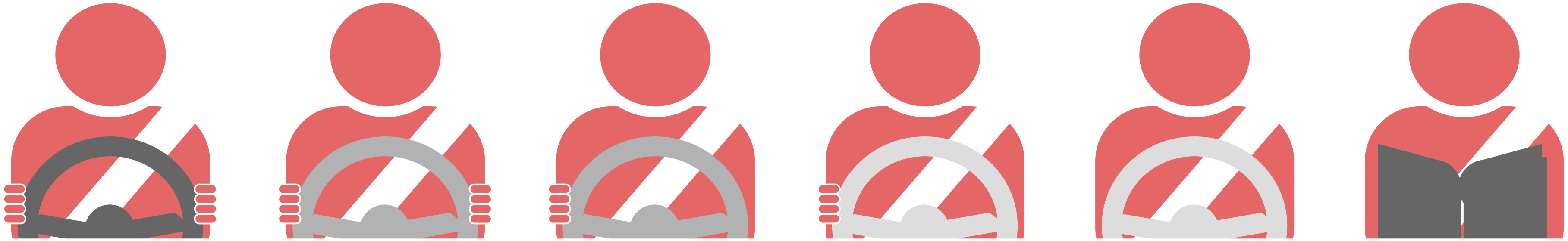
Source: https://informationisbeautiful.net/visualizations/million-lines-of-code/

Software Explosion

Software growth driven by increased vehicle functionality + increased ECU count



ADAS – SAE 6 Levels of Driving Automation



0	1	2	3	4	5
No Automation	Driver Assistance	Partial Automation	Conditional Automation	High Automation	Full Automation
Manual control	Single automated function e.g., active cruise control	Automated steering and acceleration; Human can take control	Environmental Detection; Human override required	Vehicle performs all driving tasks under specific circumstances; Requires geofencing and human override is still possible	Vehicle performs all driving tasks under all conditions, without human interaction

Human monitors

A vehicle management system is more than autonomous driving!

Based on: https://www.synopsys.com/automotive/autonomous-driving-levels.html

Hardware & OS Evolution

AUTOMOTIVE DOMAIN

- 8 → 16 → 32 bit microcontrollers
- Mostly single core, single function
- Typically 10s-100s MHz
- NXP/Freescale PowerPC, Infineon …
- Integrated CAN, GPIOs, ADCs

Simple RTOS

- OSEK, FreeRTOS, Tresos, ECOS ...

PC DOMAIN

- 64-bit CPUs, integrated GPUs
- Multicore, multiple tasks
- GHz clock speed, hardware virtualization
- Intel & AMD x86, ARM Cortex-A
- USB, PCIe, Ethernet, WiFi

Complex General Purpose OS

- Windows, Mac OS, Linux

Can we merge the two domains?

Vehicle Communication Networks & Data Processing

1Mbps and below:

+ I2C, CAN, LIN

Above 1Mbps:

+ Flexray, MOST
+ Ethernet (TSN, TTEthernet)

Emerging vehicles with multiple sensors:

+ Multiple cameras (USB 3.x, GMSL)
+ LIDAR
+ Ultrasonic

Need for low latency and high throughput

+ Google's self-driving car (2013) ~1GB/s data

A. D. Angelica: http://www.kurzweilai.net/googles-self-driving-car-gathers-nearly-1-gbsec

Automotive System Challenges

Reduce electronic costs

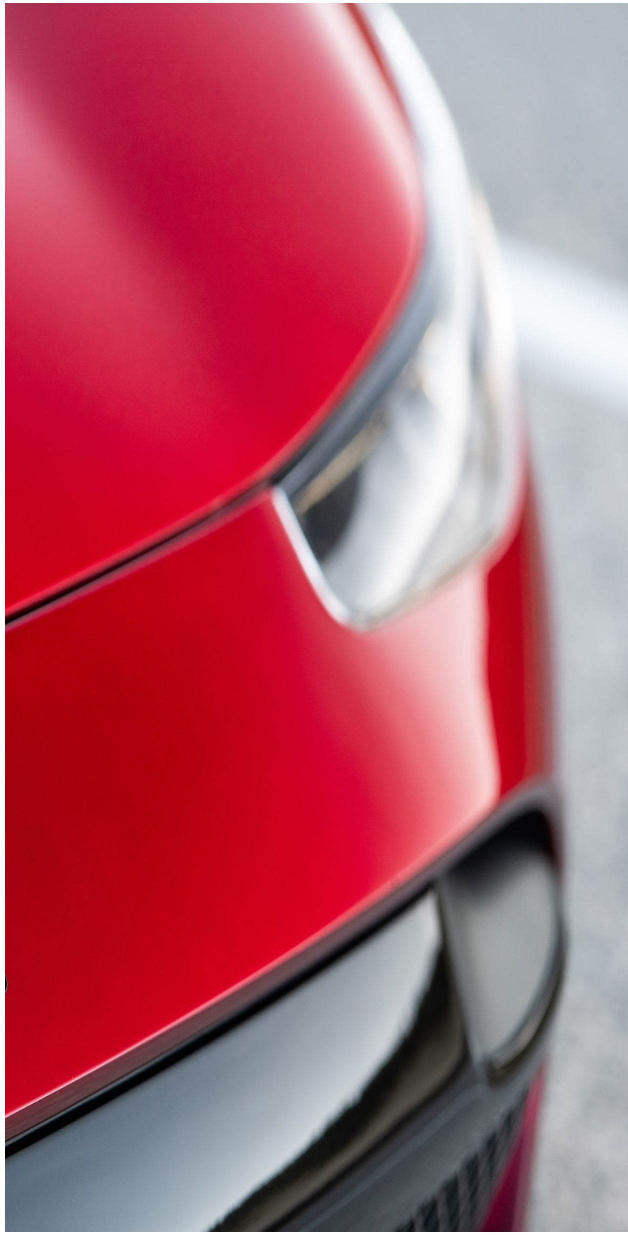
- Replace ECUs with fewer hardware components
  - e.g., multicore industrial PC

- Consolidate ECU functions as software tasks
  - Easier to update, reconfigure, extend

=> Need for **functional consolidation**

Address emerging real-time I/O needs

Functional safety and security (e.g., ISO26262 and 21434)



Automotive System Challenges

Functional Consolidation => Need new vehicle OS

- Manage 100s of tasks on multiple cores
- Handle real-time low & high bandwidth I/O
- Provide safety, security and predictability
- Support mixed-criticality, fast boot, power management

Prohibitive complexity to write new OS from scratch

- Combine real-time with legacy code
- e.g. small RTOS + Linux
- **Symbiotic** solution

Safety

Temporal and Spatial Isolation

- Ensure critical tasks are free from interference from less critical tasks

Timing and Functional Safety

- Ensure timing-critical tasks meet deadlines
- Functionally correct output values for given inputs

Correct Information Exchange

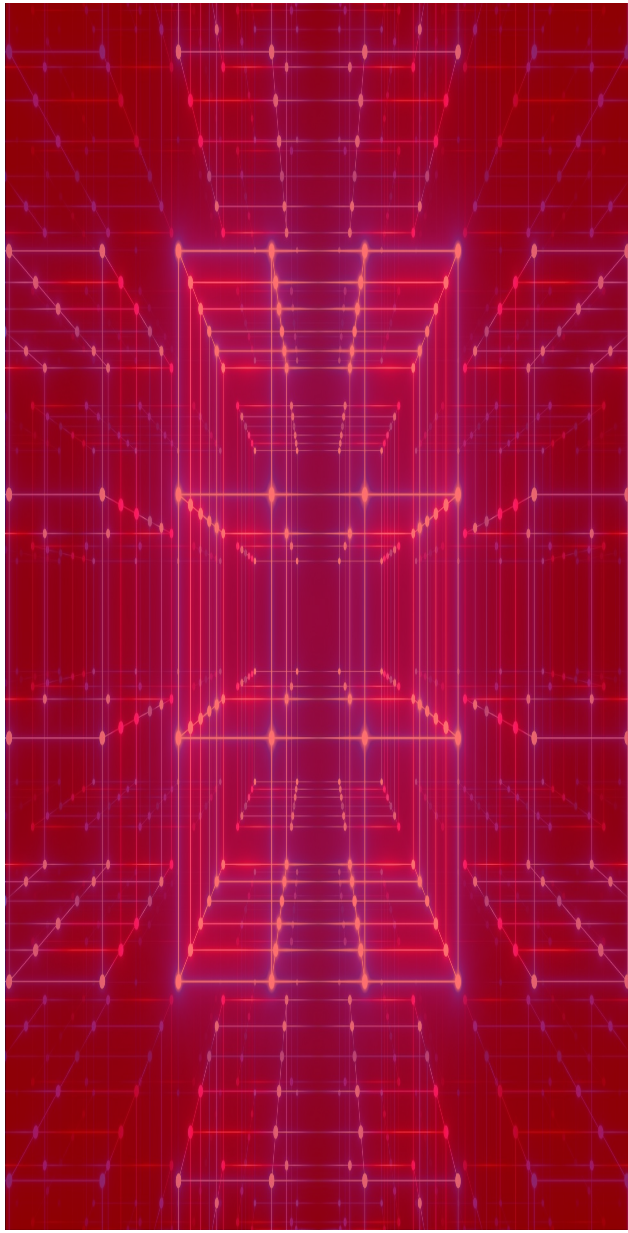
- No loss, duplication or corruption of data

Memory Safety

- No buffer overruns, stack under/overflow, invalid memory addressing

I/O Safety

- Controlled access to I/O devices

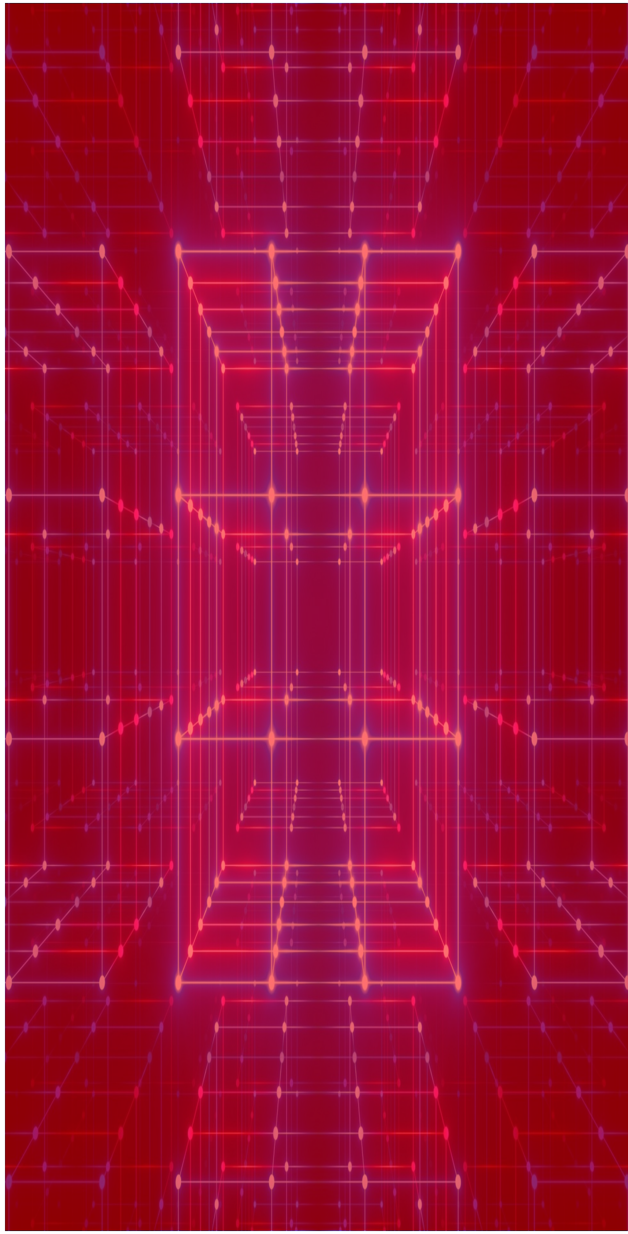


Security

Integrity
- Avoid attacker compromizing critical functionality
  - e.g., Miller & Valasek, 2014 Jeep Cherokee CAN attack via remote access to IVI

Confidentiality
- Avoid leaking sensitive data (CAN packets, personal information, app data,…)
- Eliminate side channels (e.g., via caches – possibly use cache/page coloring)

Access Rights
- Avoid user gaining elevated accesses to resources beyond allowed rights
  - e.g., CVE-2019-5736 Breaking out of Docker via RunC



Vehicle Vulnerabilities

Functional Safety (e.g., ISO26262) + Cybersecurity (e.g., ISO21434)

- ASIL classification based on Hazard Analysis and Risk Assessment
- ASIL = Exposure [E0-4] x Controllability [C0-3] x Severity [S0-3]

Example:

Remote Surface Attacks

Wi-Fi, Cellular, FM/AM radio, TPMS, Remote Keyless Entry, Bluetooth

ADAS Failures

Lane Keep Assist, Lane Departure Warning, Collision Avoidance

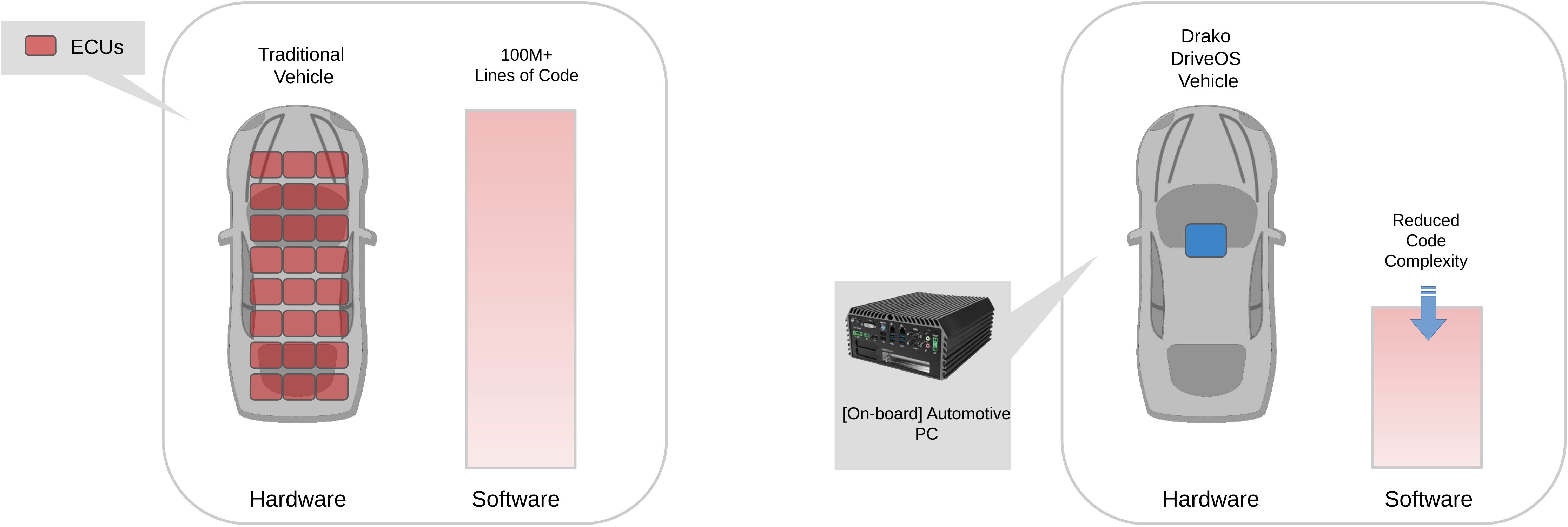
CAN Attacks

e.g. Miller & Valasek, 2014 Jeep Cherokee CAN attack via Uconnect IVI Head Unit

Moving Forward: DriveOS

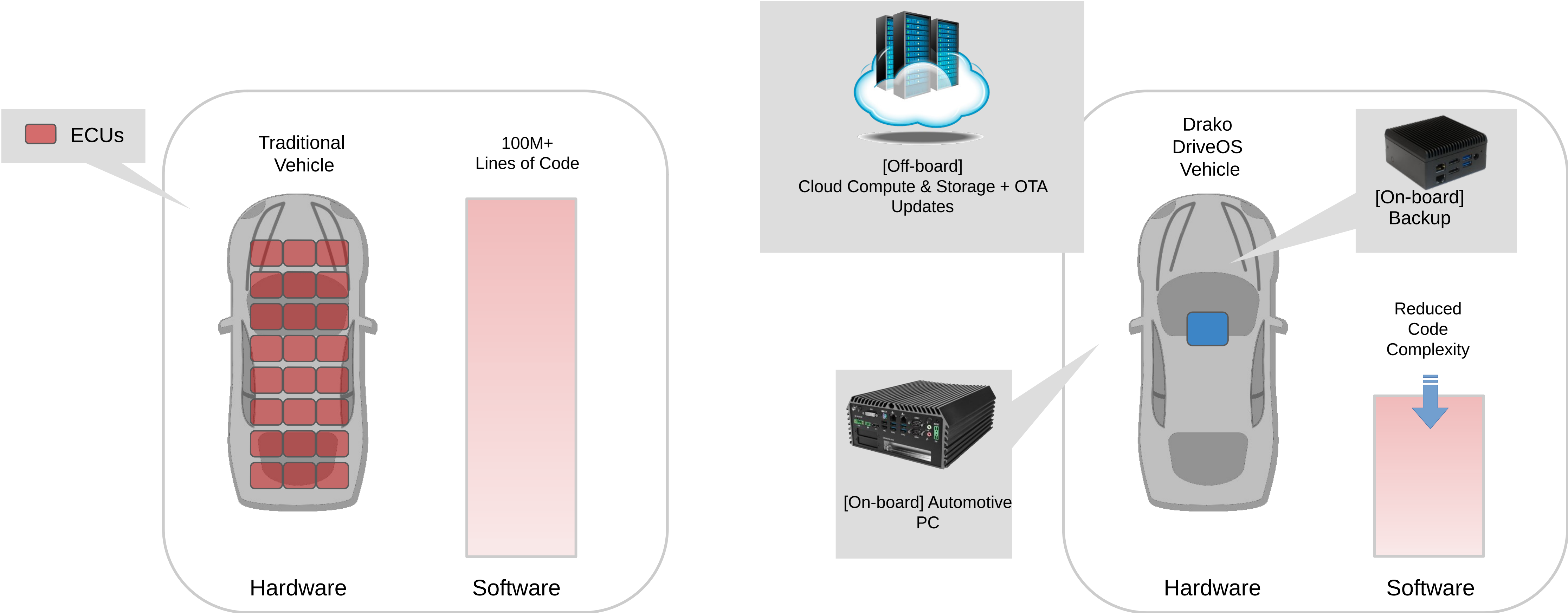
DRAKO DriveOS

DriveOS supports traditional hardware functions as software tasks running on a multicore virtualized platform



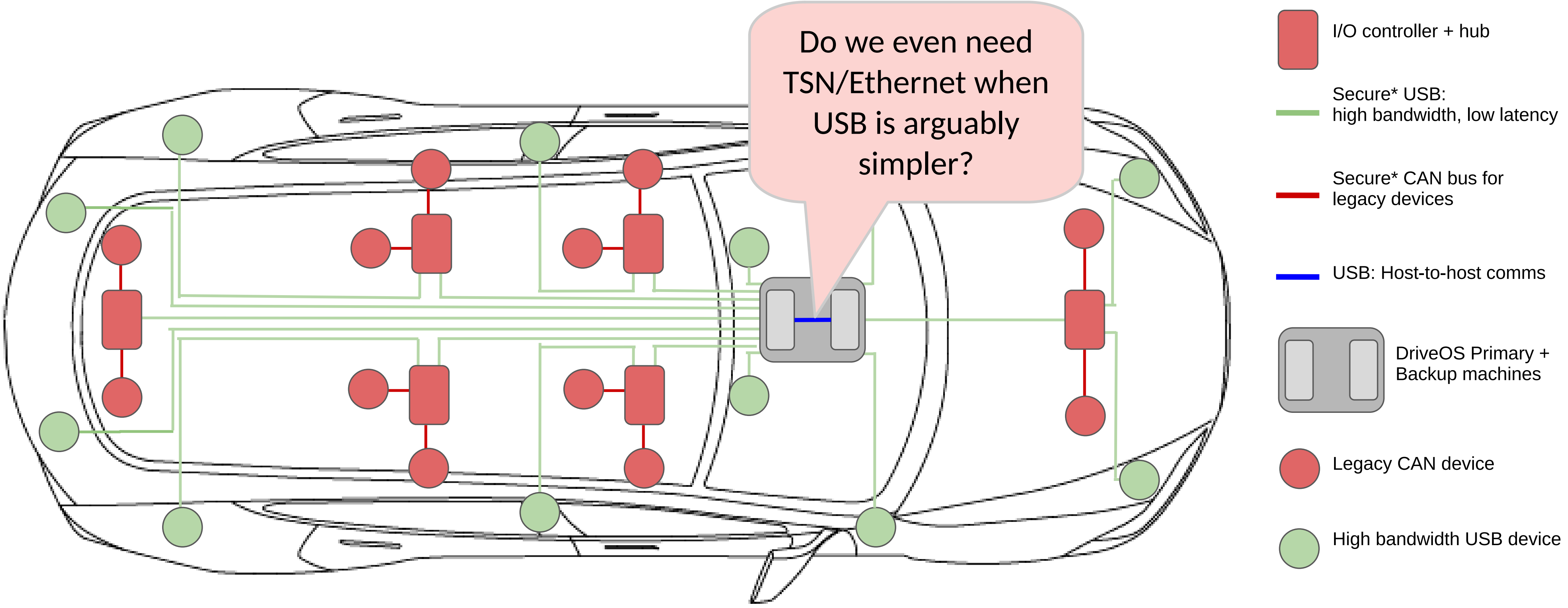
DRAKO DriveOS

DriveOS supports traditional hardware functions as software tasks running on a multicore virtualized platform

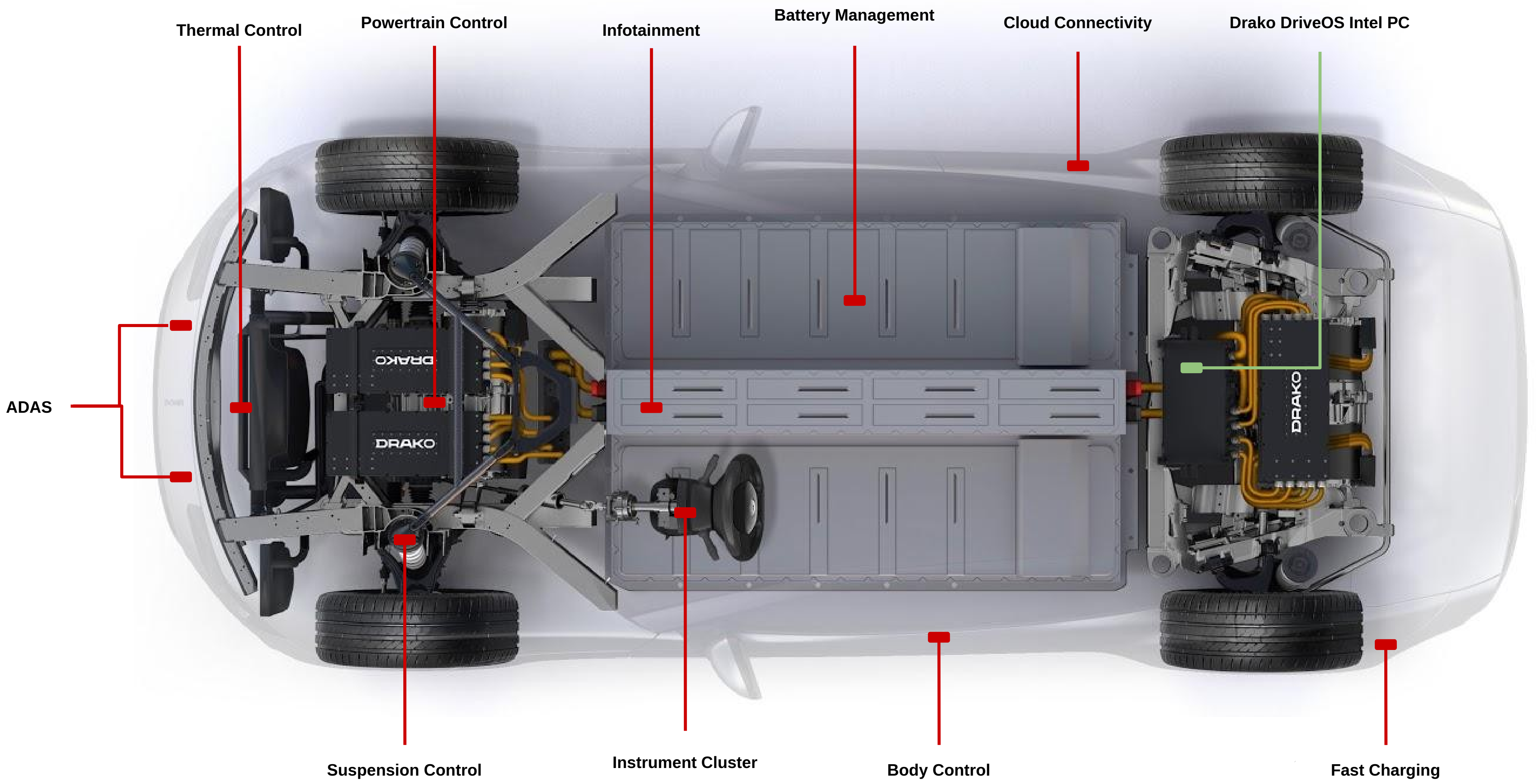


DRAKO DriveOS I/O

USB-centric solution: works with legacy devices + supports higher bandwidth future needs



Reference Design: DRAKO GTE DriveOS



DRAKO DriveOS

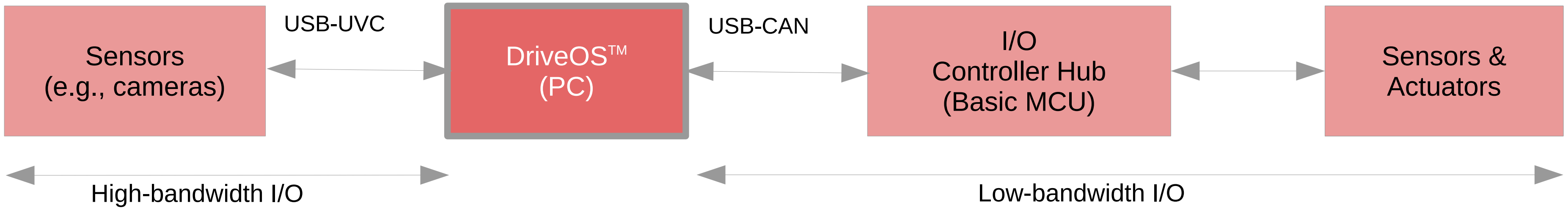
Leverage the **Quest-V** separation kernel

- Open Source
- Partitions CPU cores, RAM, I/O devices among guests

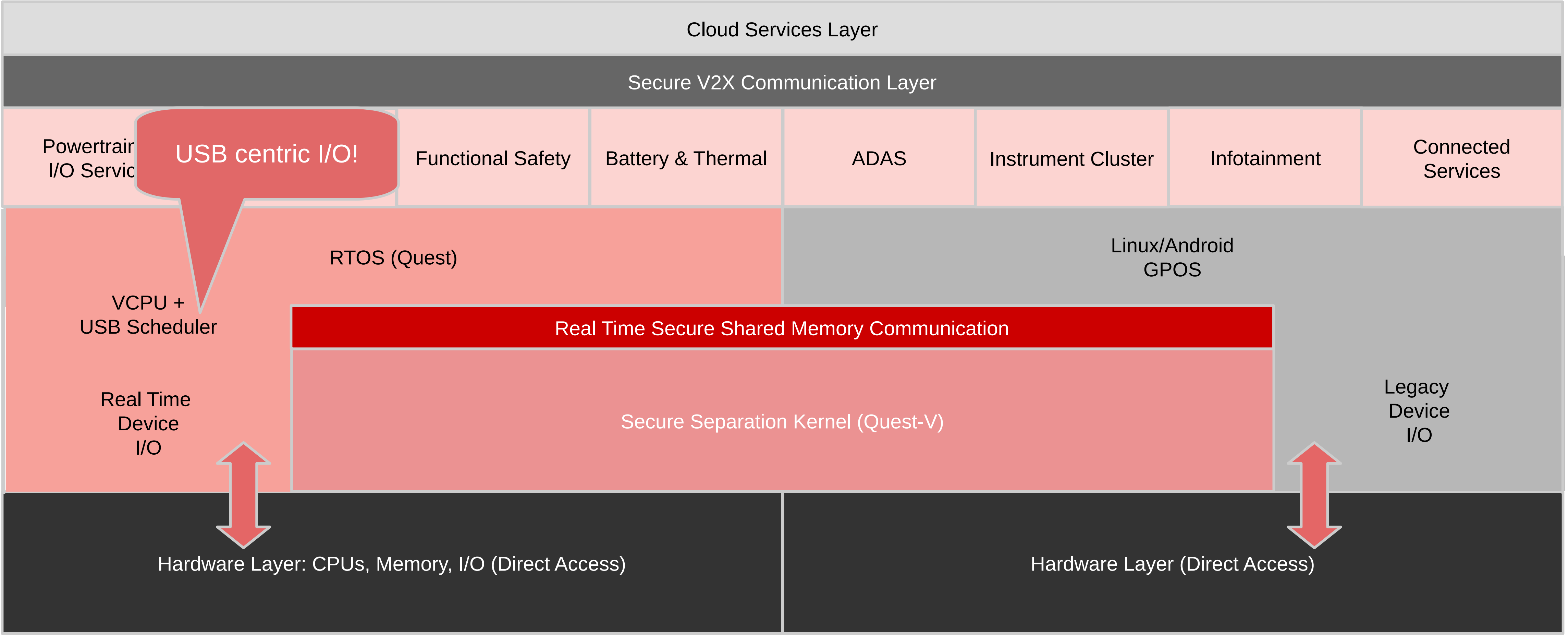
Co-locate **Quest RTOS** with Linux and Android guests on same hardware

Real-time interface for device I/O

+ Processing moved to PC
+ I/O via e.g. USB-CAN or custom control-class interface



DRAKO DriveOS Reference Stack

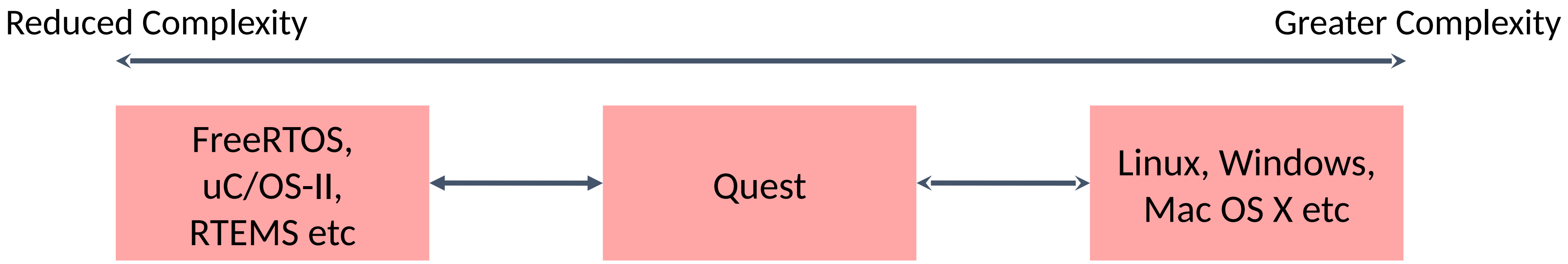


DRAKO

BOSTON UNIVERSITY

The Quest RTOS

- Open source (GPL v3), GRUB bootable either with legacy or EFI firmware
- Initially a “small” RTOS
- ~30KB ROM image for uniprocessor version
- Page-based address spaces
- Kernel threads (simple POSIX implementation)
- Dual-mode kernel-user separation
- Real-time Virtual CPU (VCPU) task and interrupt scheduling
- Later SMP support (defaults up to 8 cores, expandable to 256 or higher)
- LAPIC timing
- Semaphores and spinlocks
- Tuned pipes
- Real-time USB 2 (EHCI) and 3.x (xHCI) stack



Quest Virtual CPUs (RTAS'11)

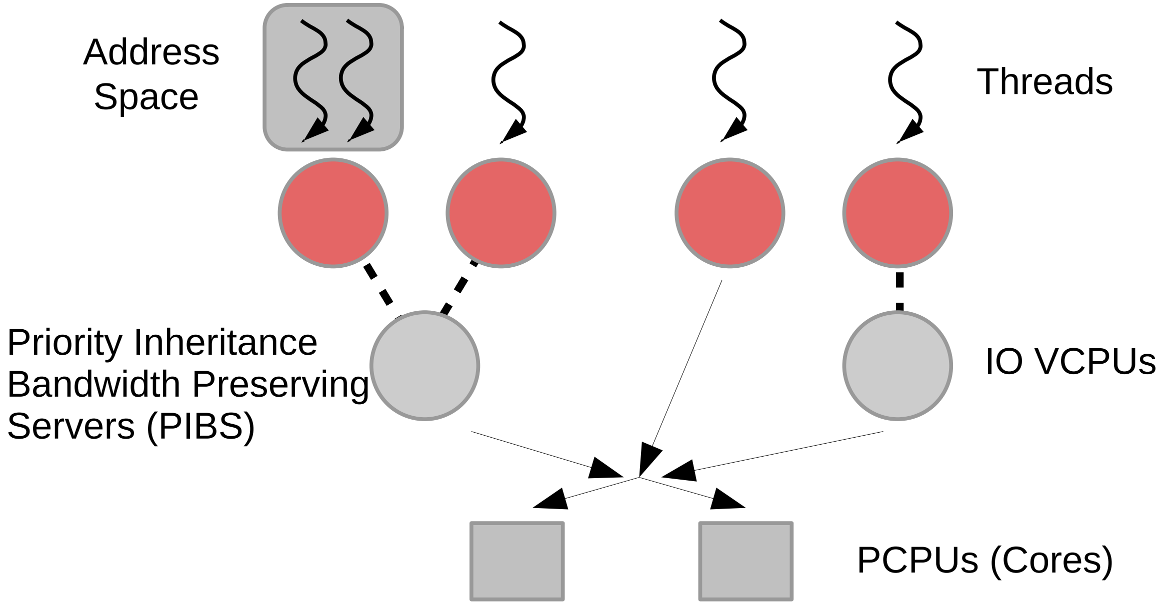
VCPUs are first-class entities within the RTOS

- Provide CPU resource reservations
- Budgeted real-time execution of threads and interrupts

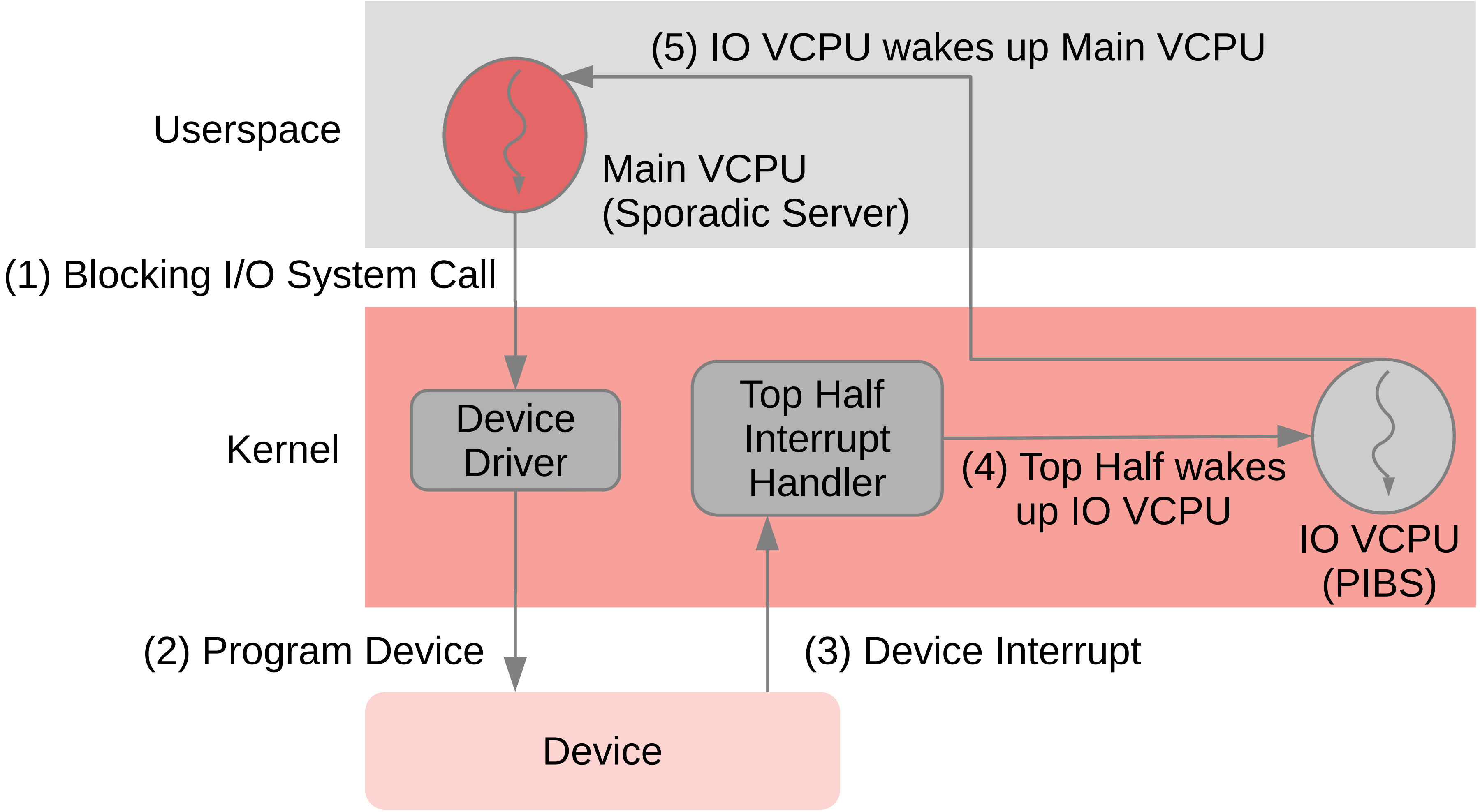
- Tasks → Main VCPUs (Sporadic servers: budget & period)
- Interrupts → IO VCPUs (PIBS: derive budget & period from Main VCPU)

Real-time IO

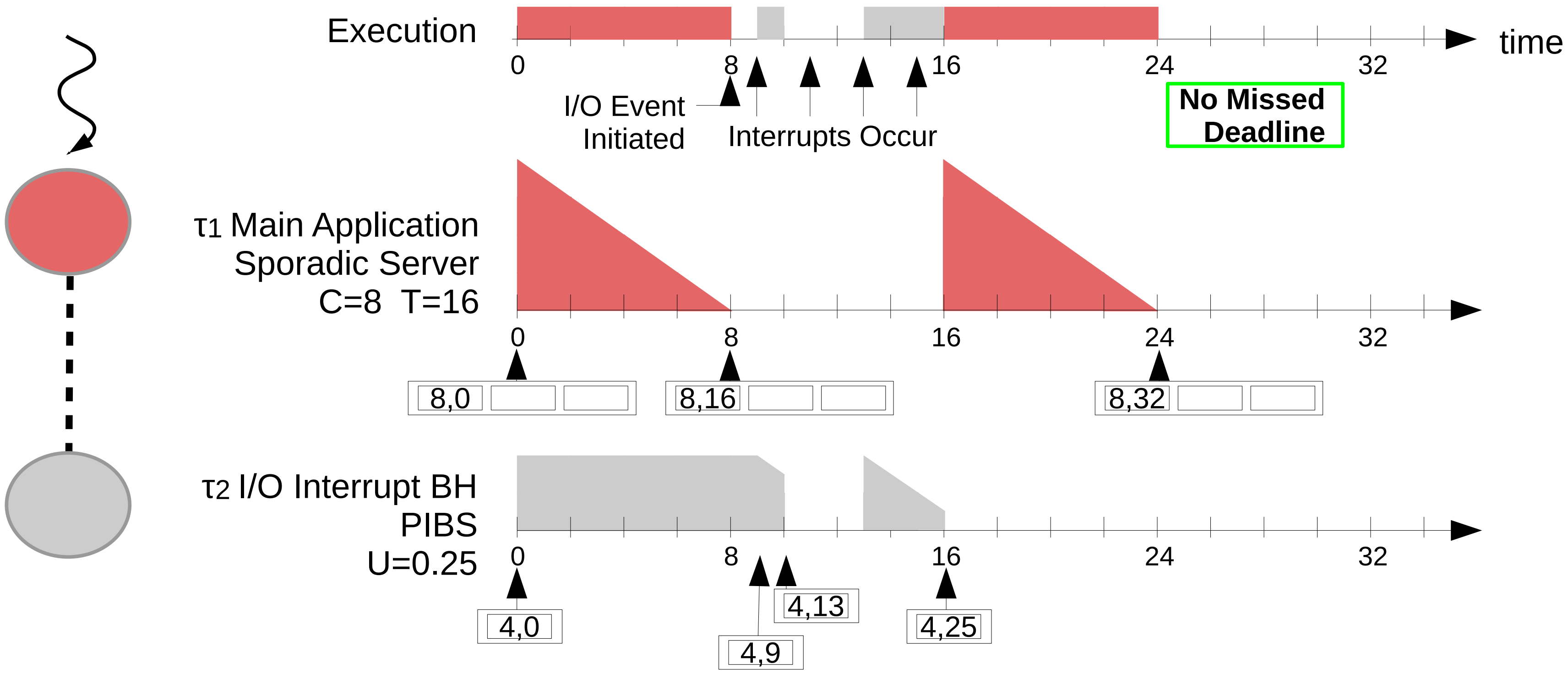
IOVCPU_CLASS_USB
IOVCPU_CLASS_NET
IOVCPU_CLASS_GPIO
...



VCPU Control Flow



Example SS+PIBS Schedule



- PIBS use one replenishment
- No merging of replenishments required and only one (LAPIC) timer event to program
- Although theoretically inferior to SS-only scheduling, practically better with more servers

Quest USB Stack (RTAS'13, RTSS'18, ACM TECS'23)

- USB ubiquitous for I/O devices
- 480 Mbps (USB 2) to 5-20 Gbps (USB 3.x), integration with PCIe/DisplayPort (USB 4&5)
- Quest supports EHCI and xHCI
- Supports xDBC for host-to-host communication
- Working on xDCI support

- Real Time Capability
  - USB 2 (EHCI) & 3 (xHCI) Scheduling
  - Differentiated Service of Interrupts

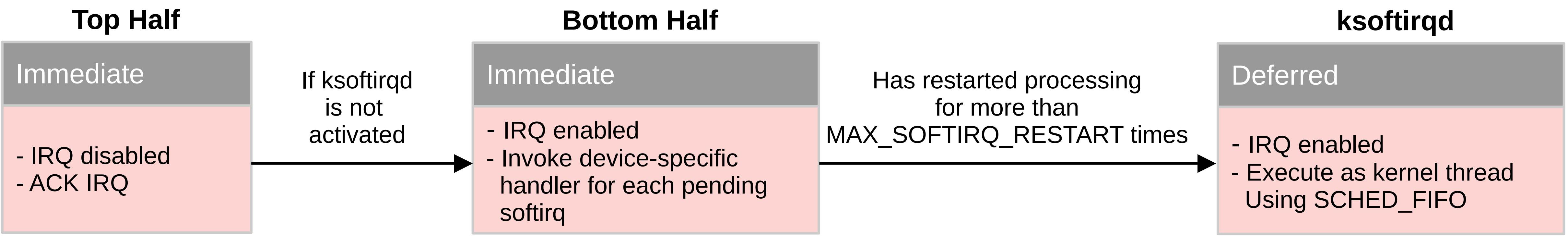
Interrupt Handling

- Problem: how are interrupts associated with service requests
- How do we then prioritize them correctly?
- **[RTAS'24]** USB provides way to achieve early demultiplexing (in hardware!)
  - Interrupts are correctly processed at priority of task causing them
  - With Message Signaled Interrupts (MSI-X*), USB host controller can support up to **1024 interrupters**

*MSI-X can potentially support 2048 interrupts per device, if device is capable of that many interrupts

Linux Interrupt Handling

- Interrupts split in top halves and bottom halves
- Preempted tasks are charged for the time spent handling interrupts, causing potential deadline misses



Example 1: Priority Inversion with Linux

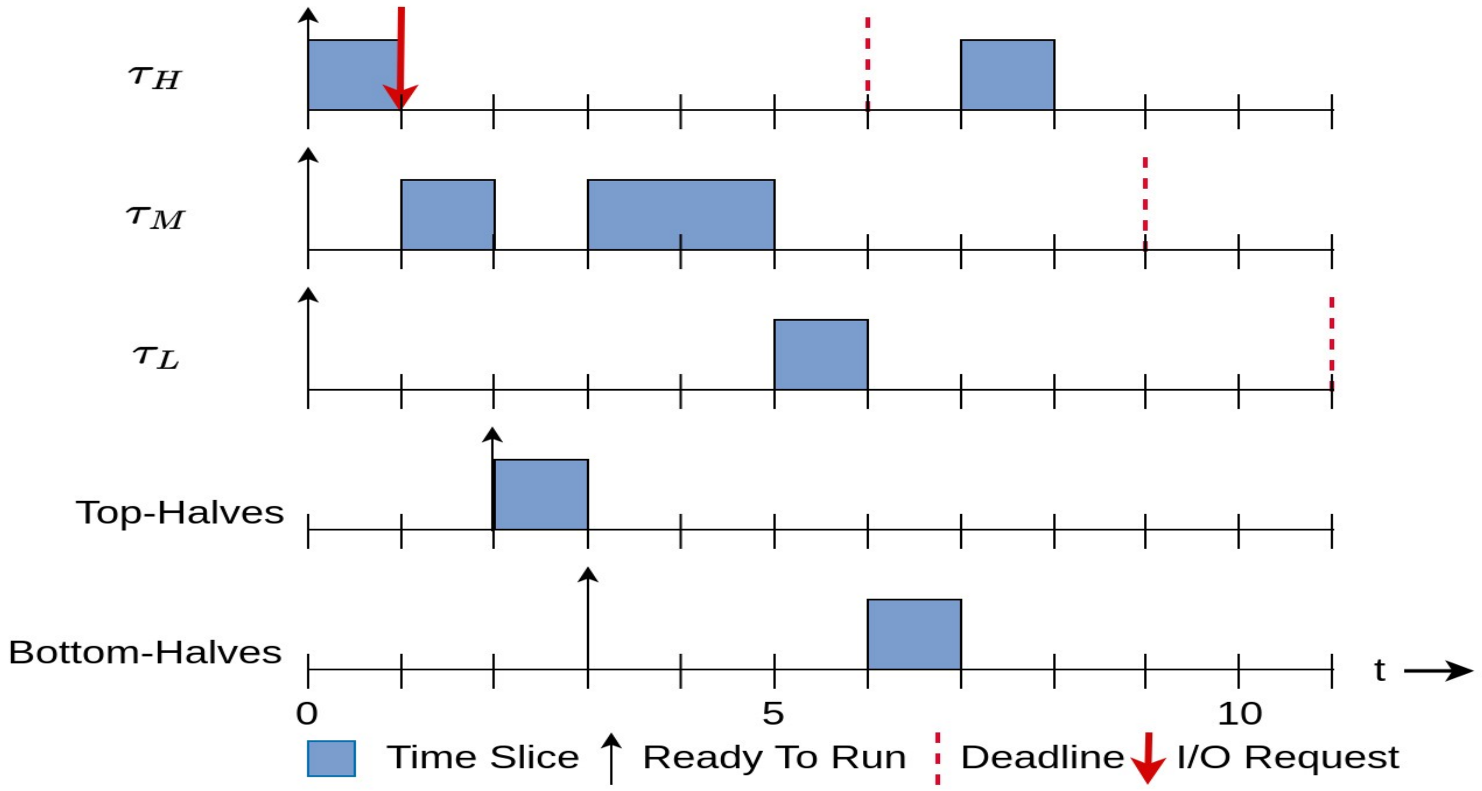


Figure 1: Linux deadline miss

Example 1: Quest Fixes Priority Inversion

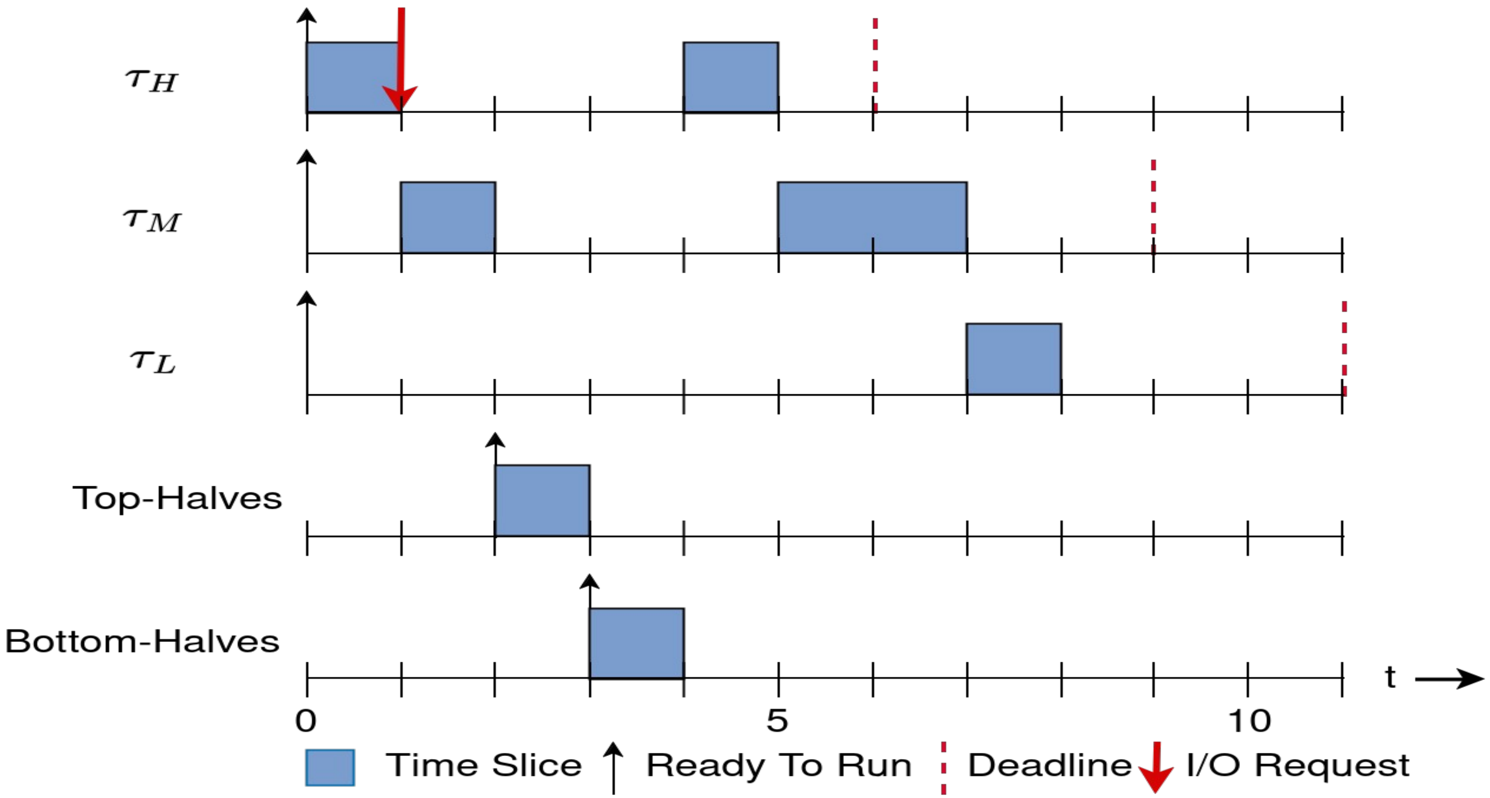


Figure 2: Quest (No Differentiated Service): No deadline miss

Example 2: Without Differentiated Service Quest Still Suffers Priority Inversion

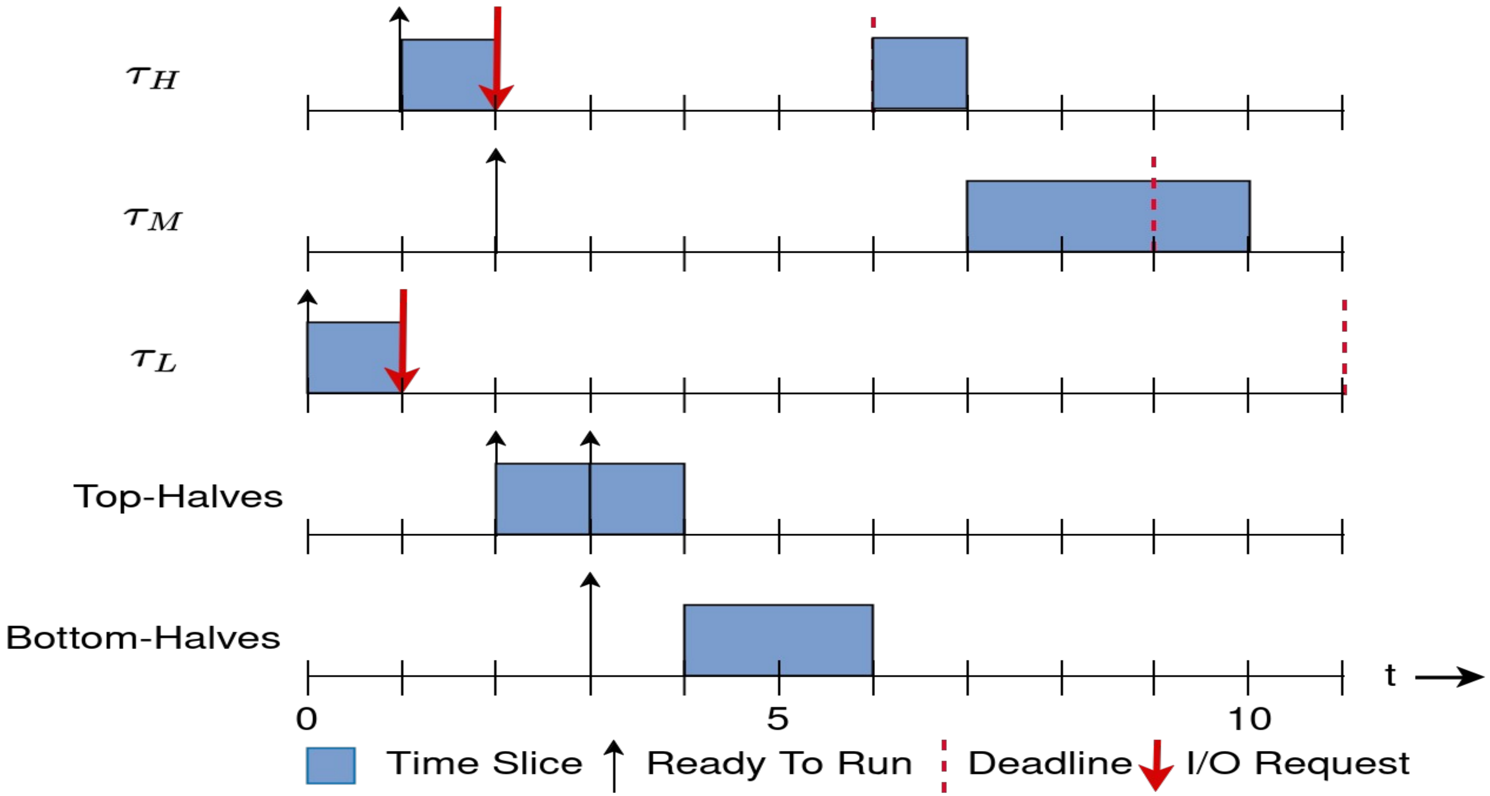


Figure 3: Quest (No Differentiated Service): Deadline miss

Example 2: With Differentiated Service Quest Avoids Priority Inversion

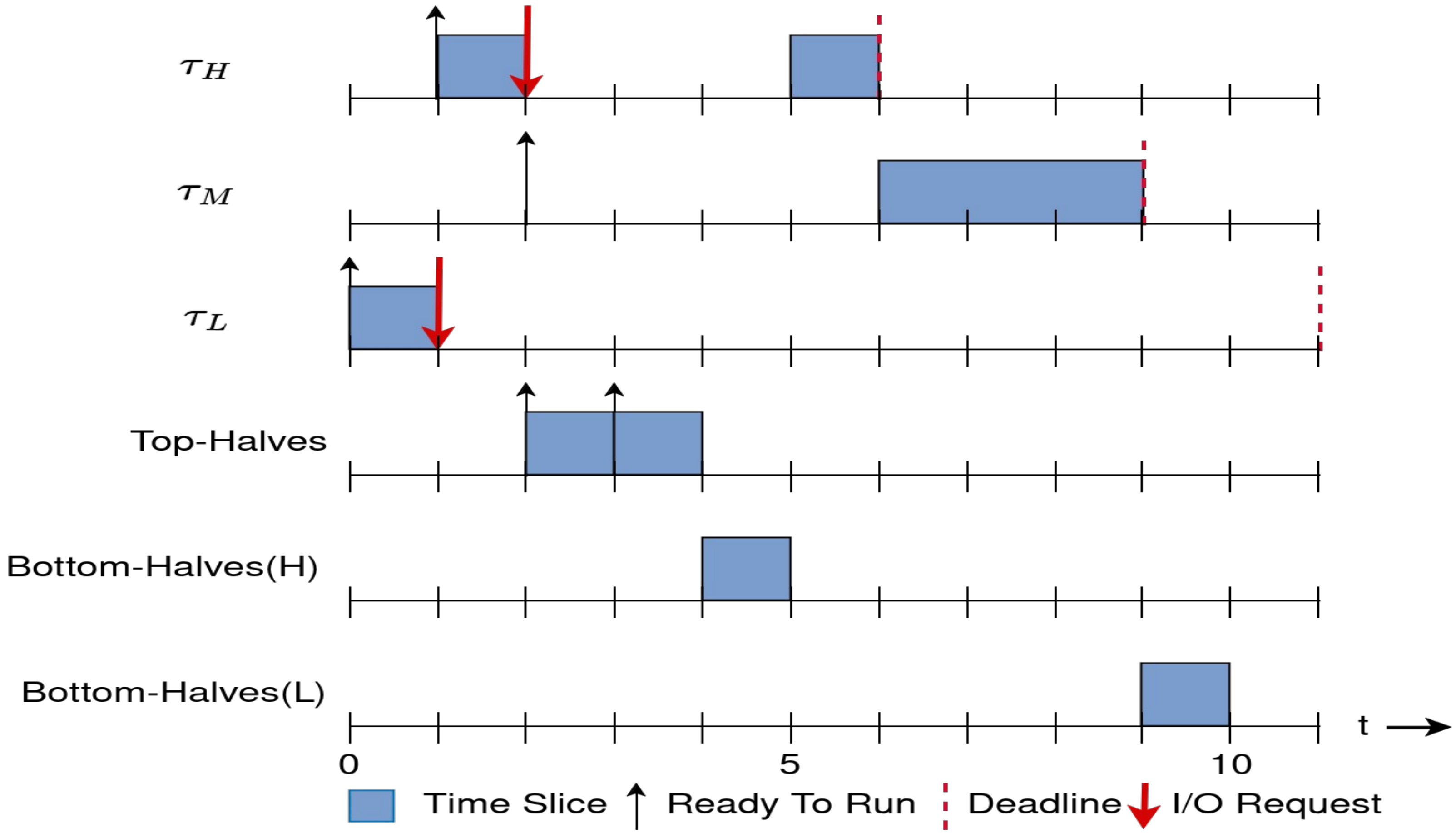
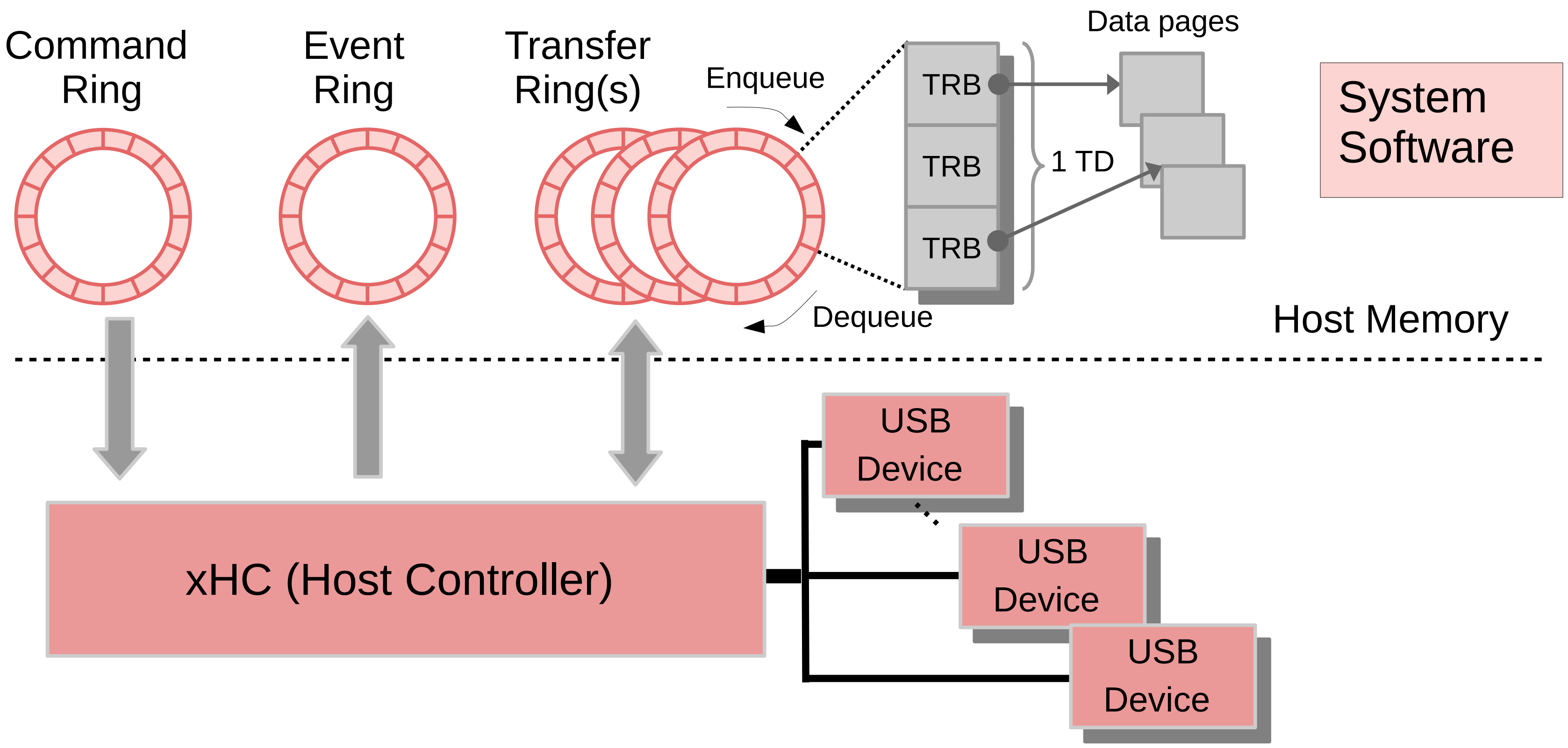


Figure 4: Quest (Differentiated Service): No Misses

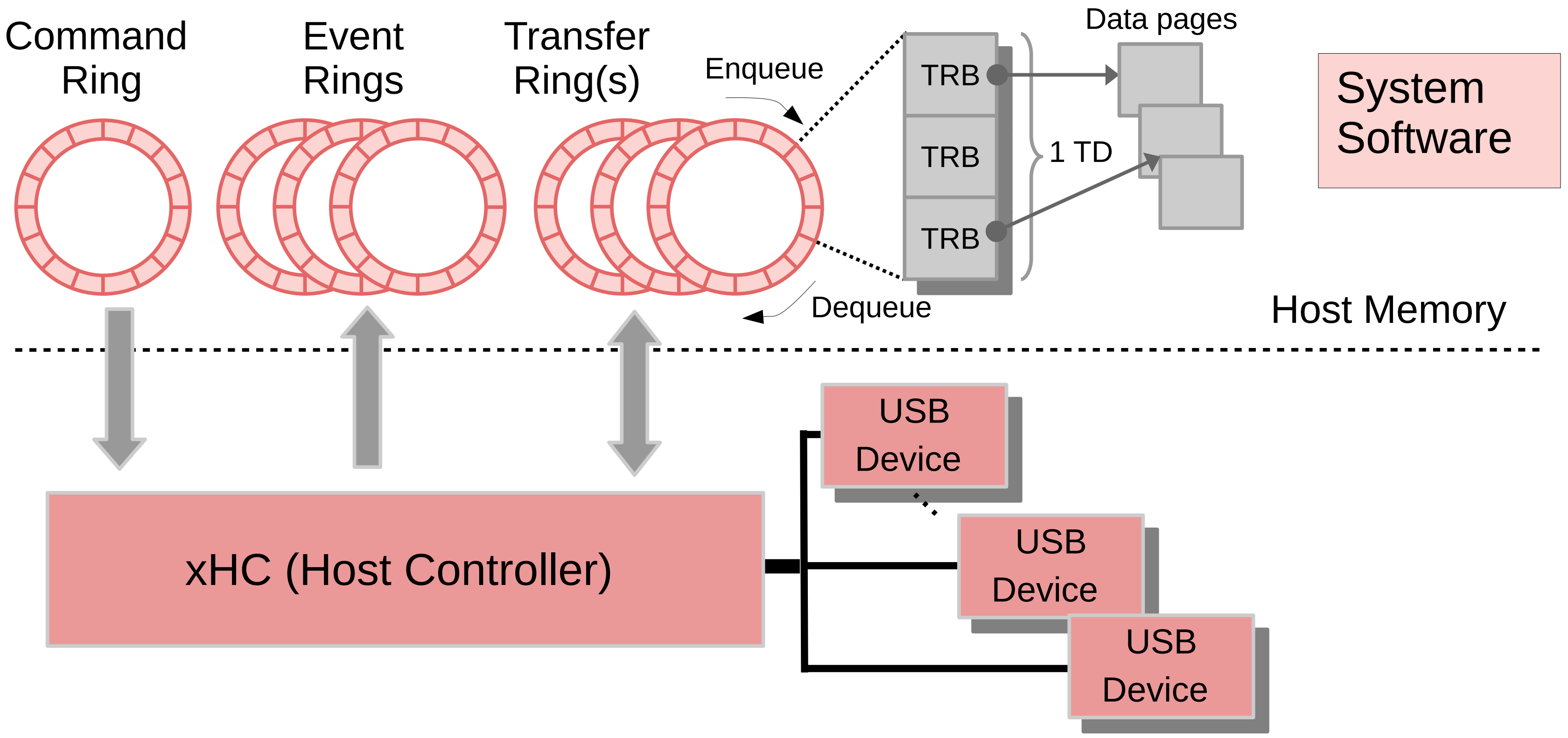
USB Differentiated Services (RTAS'24)

No differentiation (Linux approach) – one event ring for all interrupts on completion

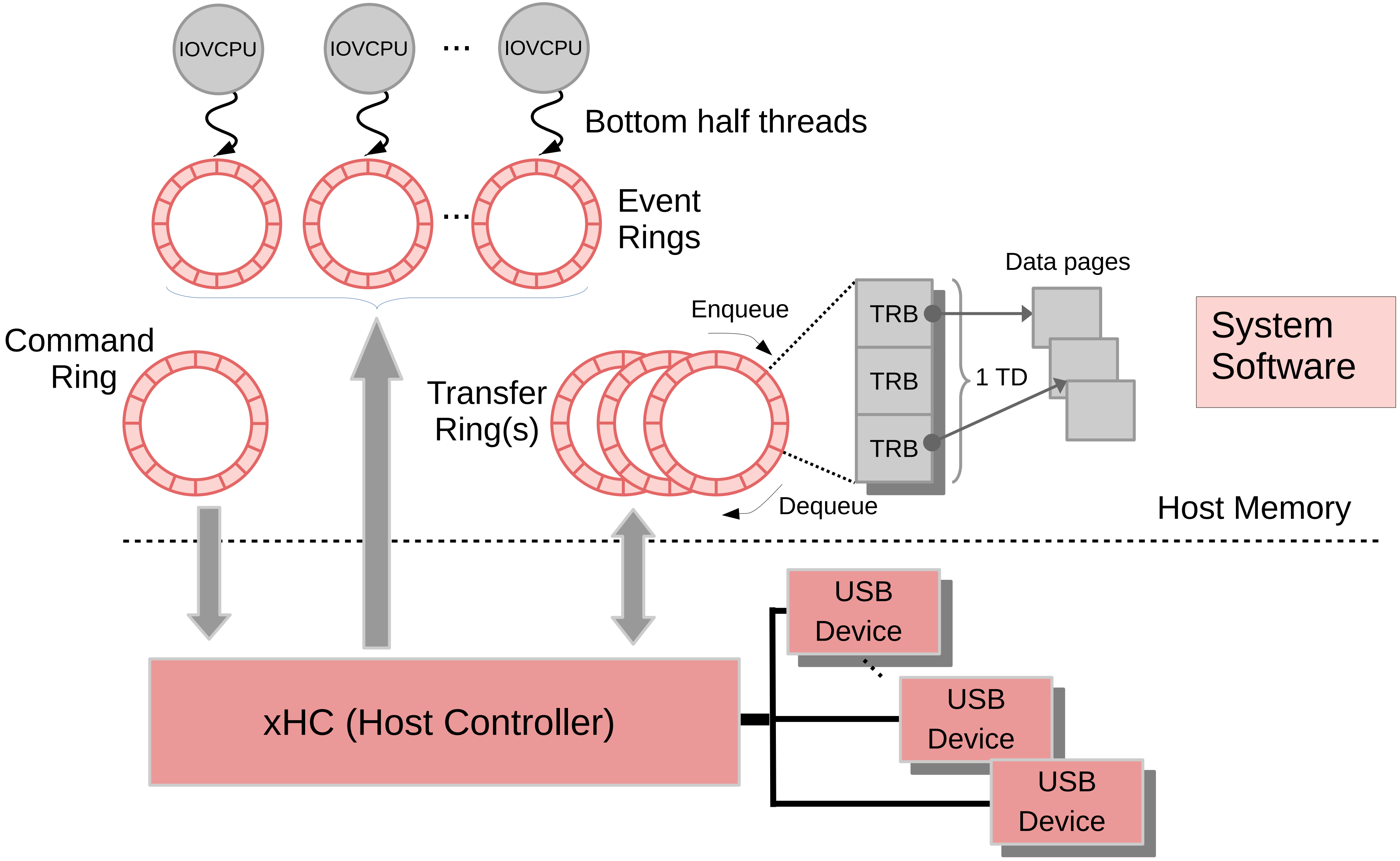


USB Differentiated Services (RTAS'24)

Differentiation – one event ring per interrupter



USB Differentiated Services with Bandwidth Preservation (RTAS’24)



Throughput Differentiated Service

- Perform test with varying I/O VCPU utilization parameters
- Ratio between each throughput corresponds to the ratio of I/O VCPU parameters
- Shows we can guarantee differentiated throughput

DX1100 (2.4 GHz Intel Core i7)

Teensy 4.1 (Arm Cortex-M7 600MHz)

CDC-ACM + Interrupter-aware xHCI Driver

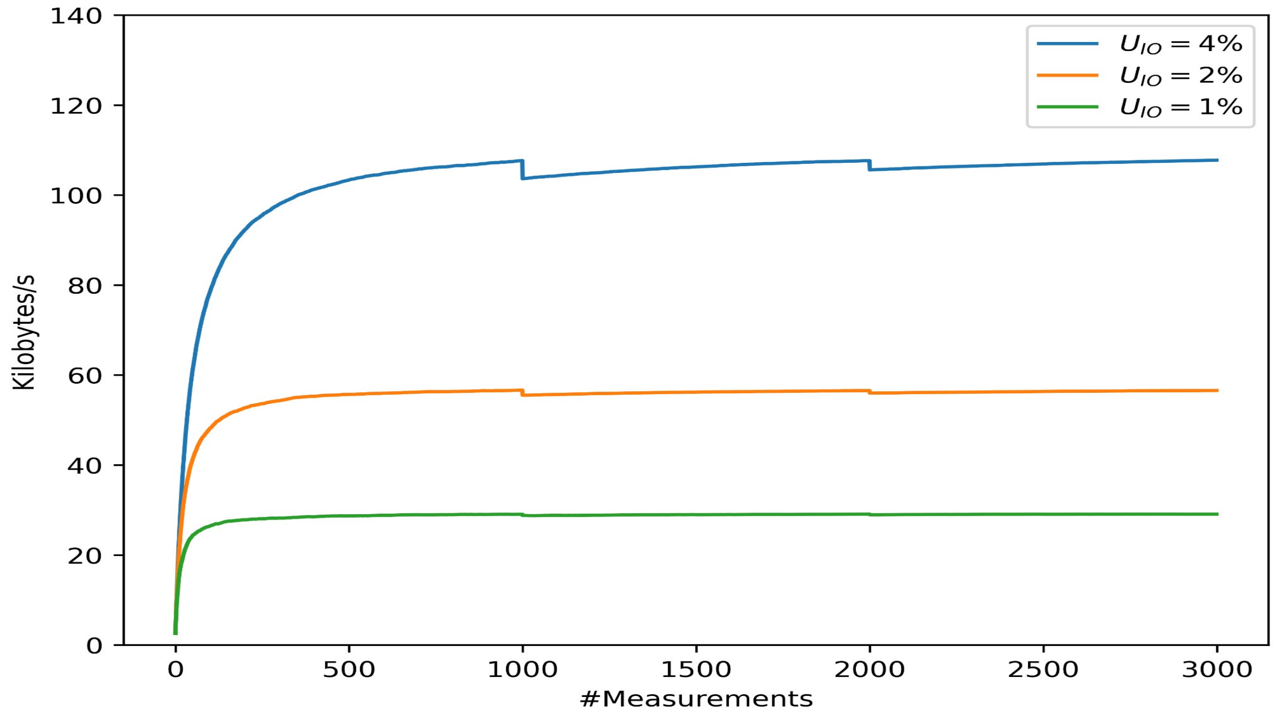


Figure 14: Read throughput for different I/O VCPU utilizations

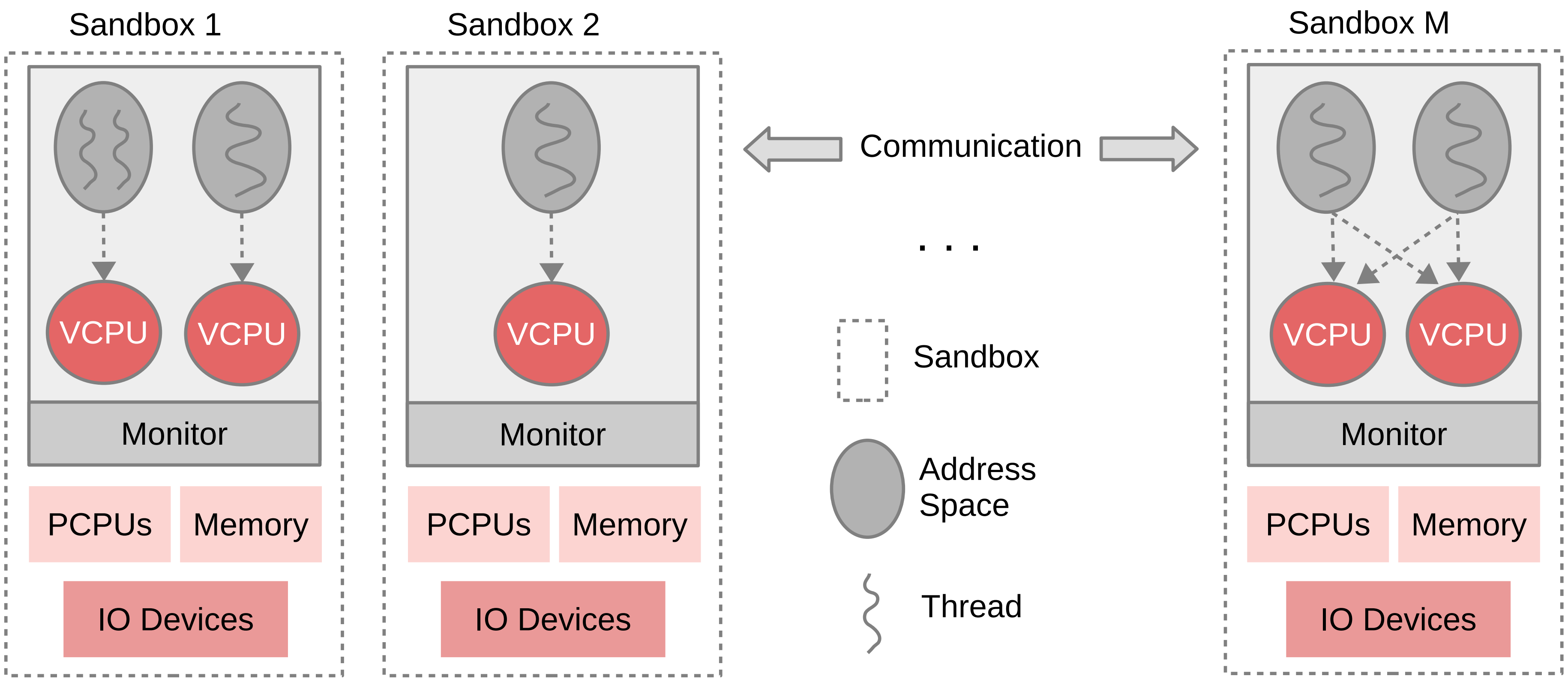
From Quest to Quest-V

- Distributed system on a chip
- Uses Intel VT-x capabilities found on PCs and SBCs/SoCs:
  - Galileo, MinnowBoard, Edison, Joule, Intel Aero, Up boards, Intel Automotive SoC (Malibou Lake),…
- Separate sandbox kernels for system components
- Memory isolation using hardware-assisted memory virtualization
- Also CPU, I/O, cache partitioning

- Supports symbiotic union between Quest RTOS and other legacy systems such as Linux or Android
- Supports horizontal scaling of multiple “small” OSes

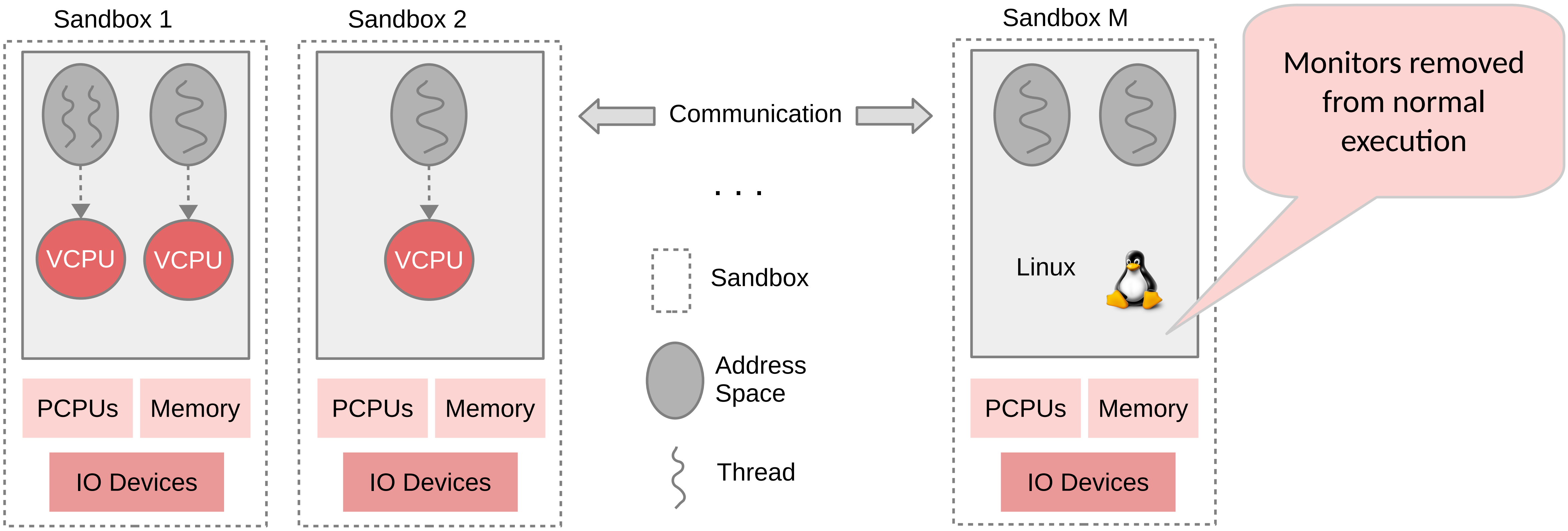
Quest-V Separation Kernel (VEE'14, ACM TOCS'16)

- Monitors partition CPU cores, RAM, I/O devices among sandboxed guests
- Monitors have small trusted compute base – no runtime resource management



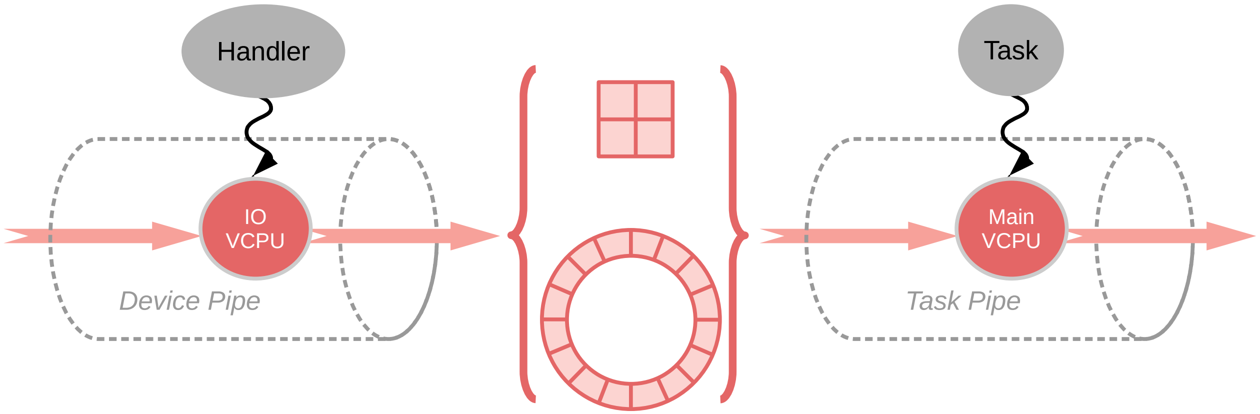
Quest-V Separation Kernel (VEE’14, ACM TOCS’16)

- Partitioning hypervisor – statically partitions resources
- Separation kernel – distributed collection of sandboxed components, indistinguishable from separate private machines for each component



Tuned Pipes (RTSS'18, RTAS'20)

- E2E guarantees on task pipelines are critical
- **Tuned Pipes** like POSIX pipes but guarantee throughput and delay on communication
- Simpson's 4-slot (asynchronous) & FIFO (synchronous) buffering

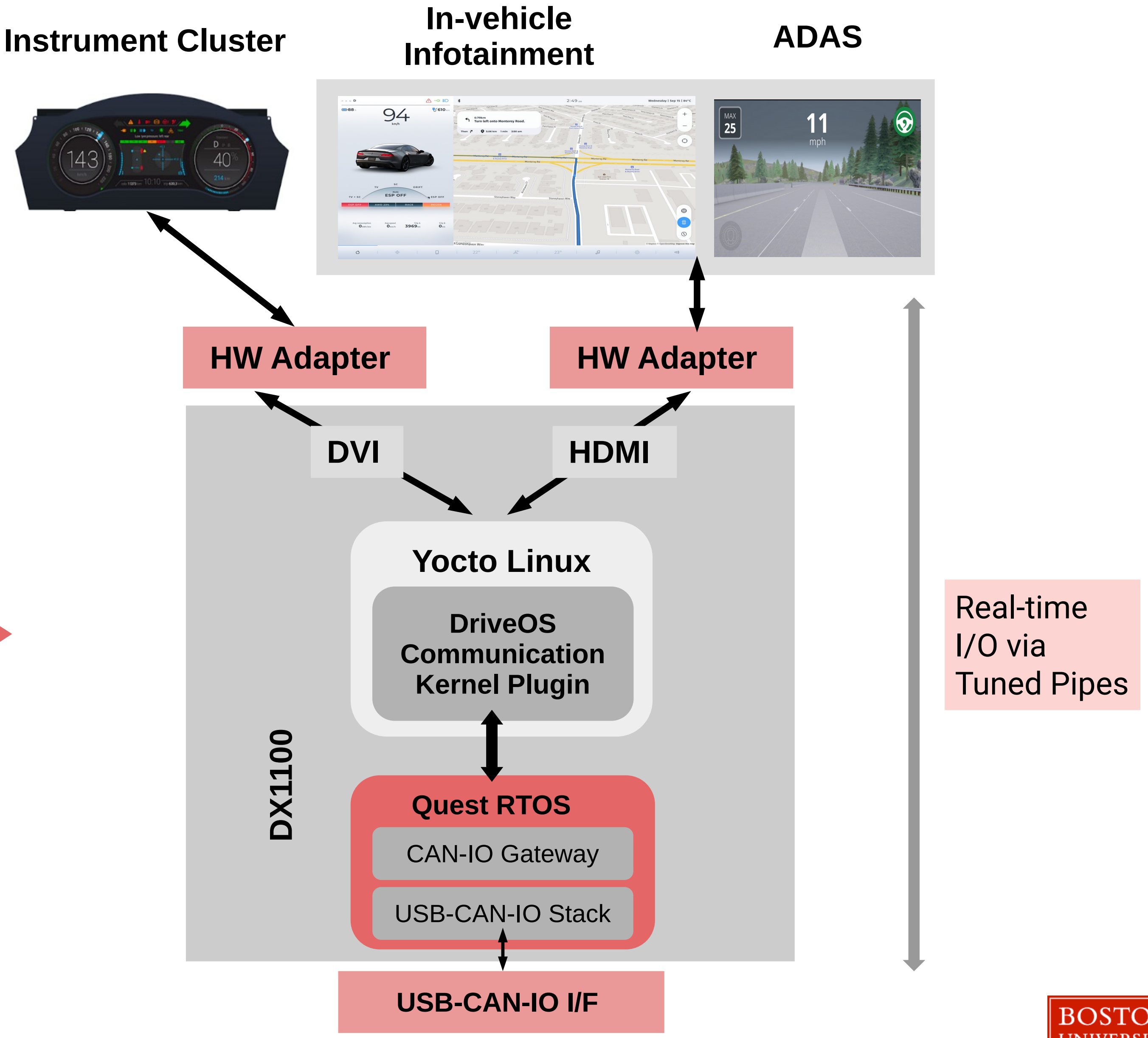


- **Boomerang** I/O subsystem in Quest-V supports real-time pipelines across Quest RTOS and legacy OSes
- Rate match tasks in pipeline to avoid blocking or missed data
- Quest appears as a **real-time virtual device interface** to Linux/Android

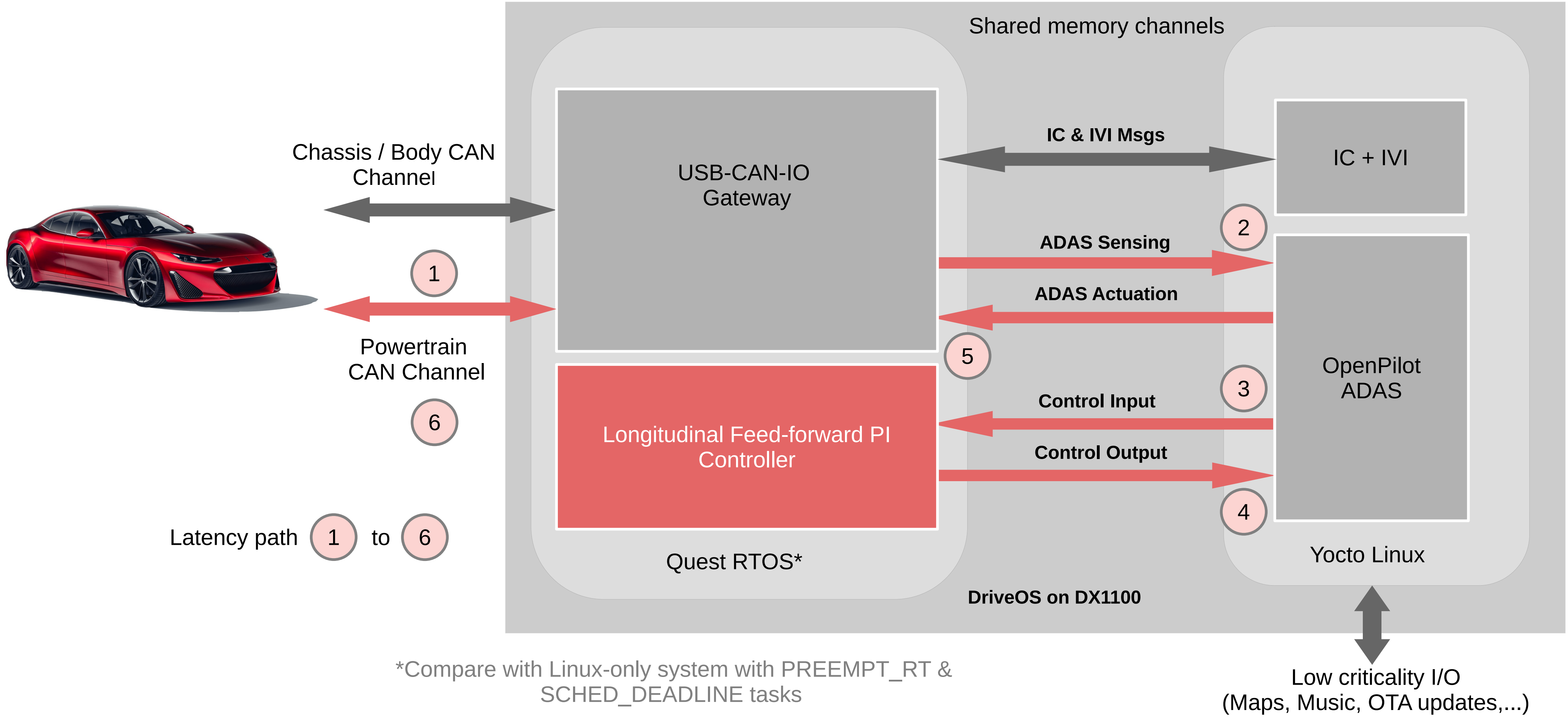
DriveOS Example (EMSOFT’21)

Map all services to a
single industrial automotive PC

Cincoze DX1100



DriveOS: Example OpenPilot ADAS+IC+IVI (EMSOFT’21)

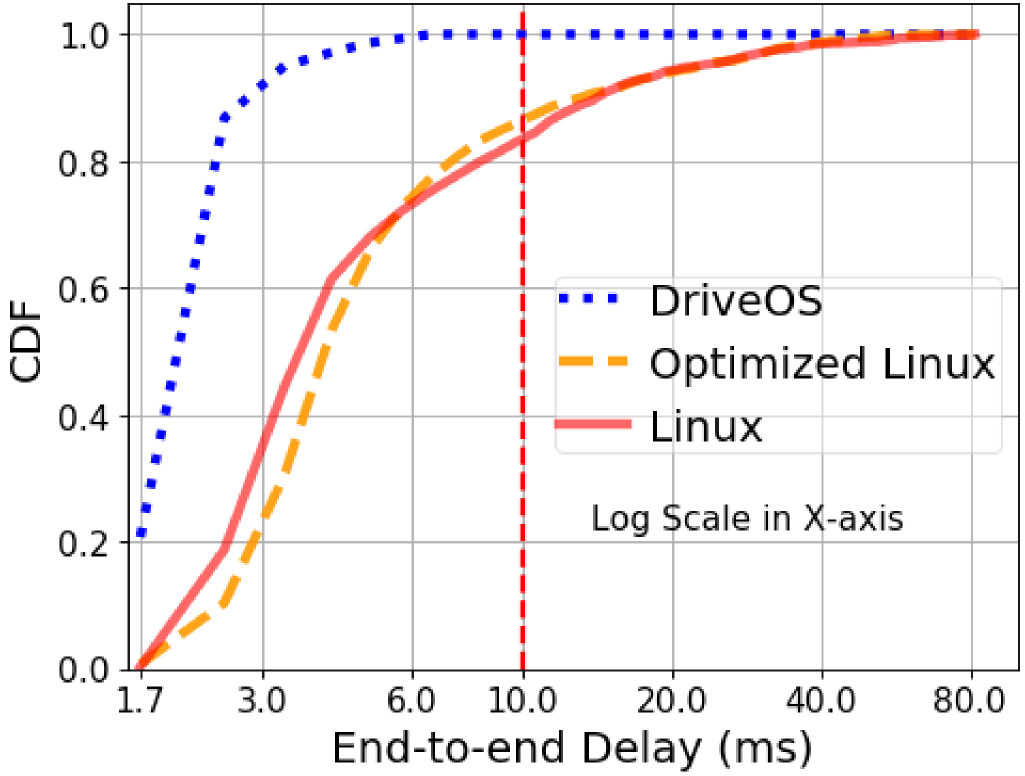


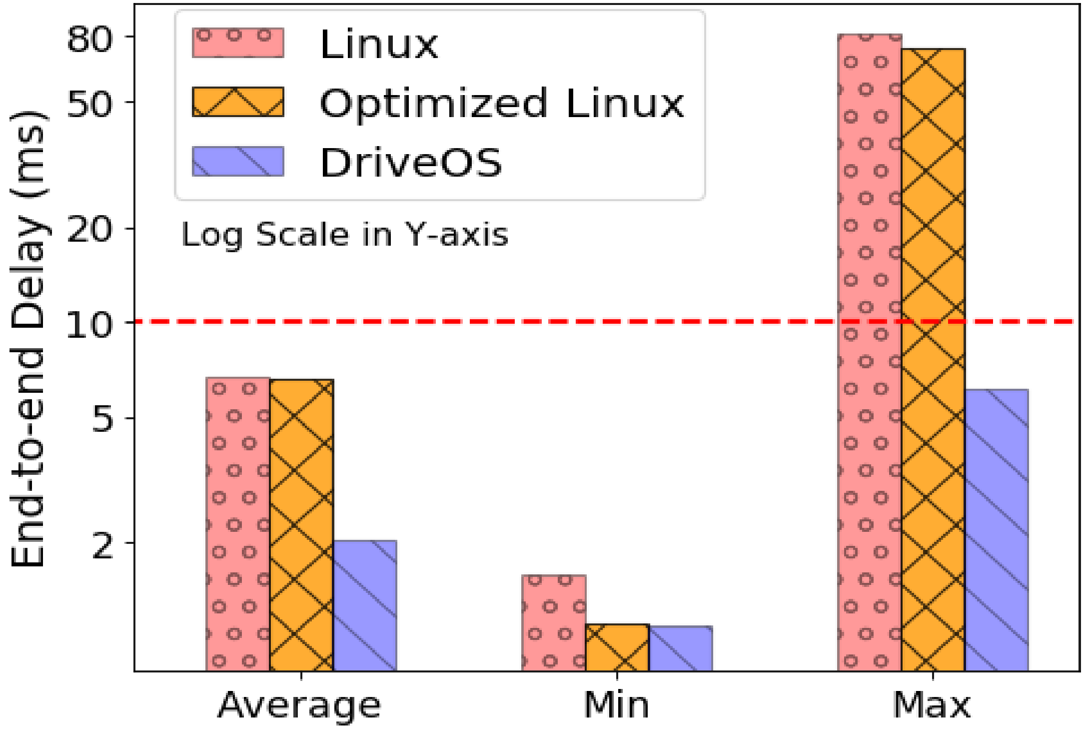
*Compare with Linux-only system with PREEMPT_RT & SCHED_DEADLINE tasks

DriveOS: OpenPilot Control Loop Latency (EMSOFT'21)

ADAS Control Loop End-to-end Latency in presence of background Linux tasks

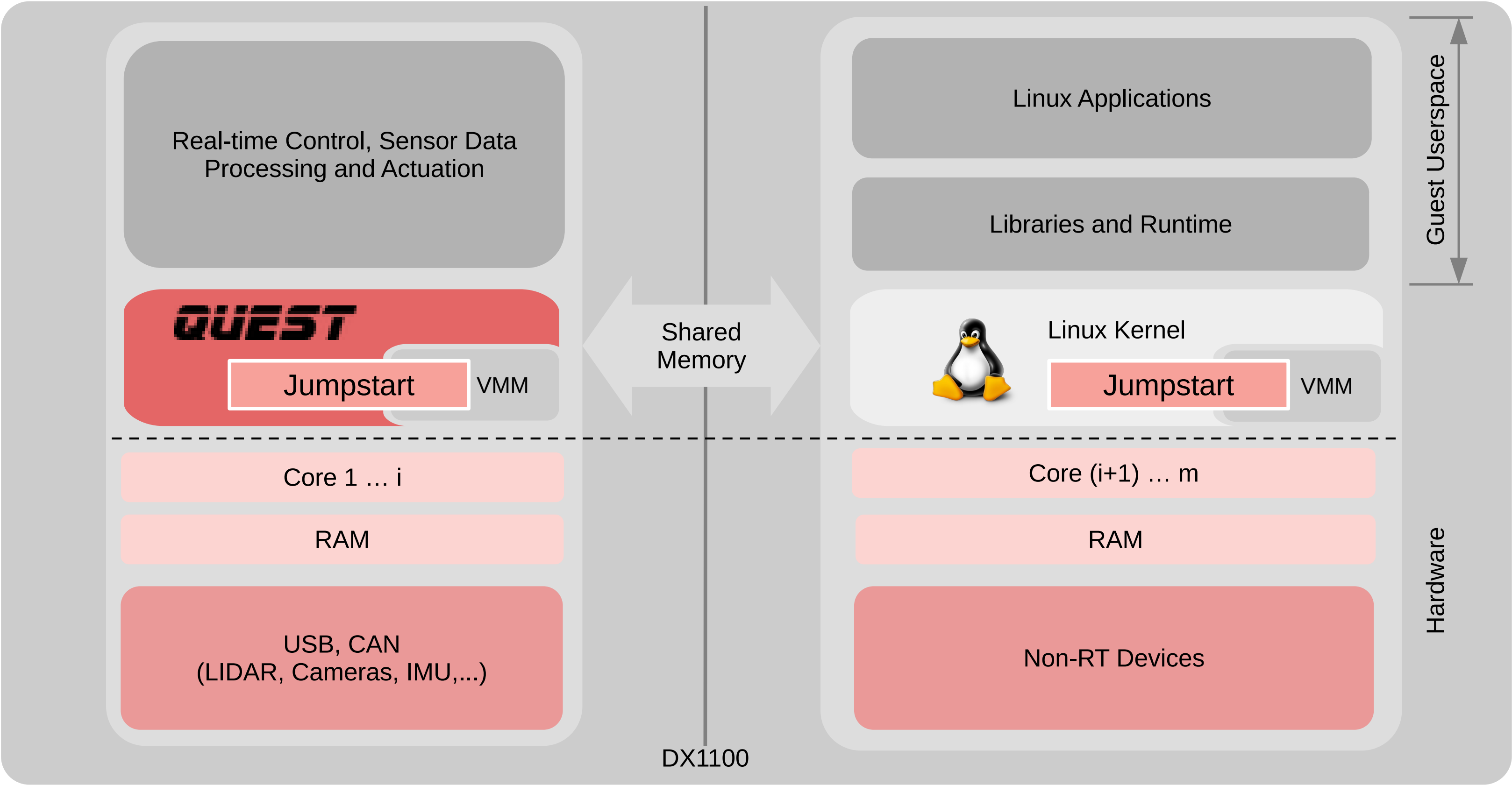
Target bound = 10ms





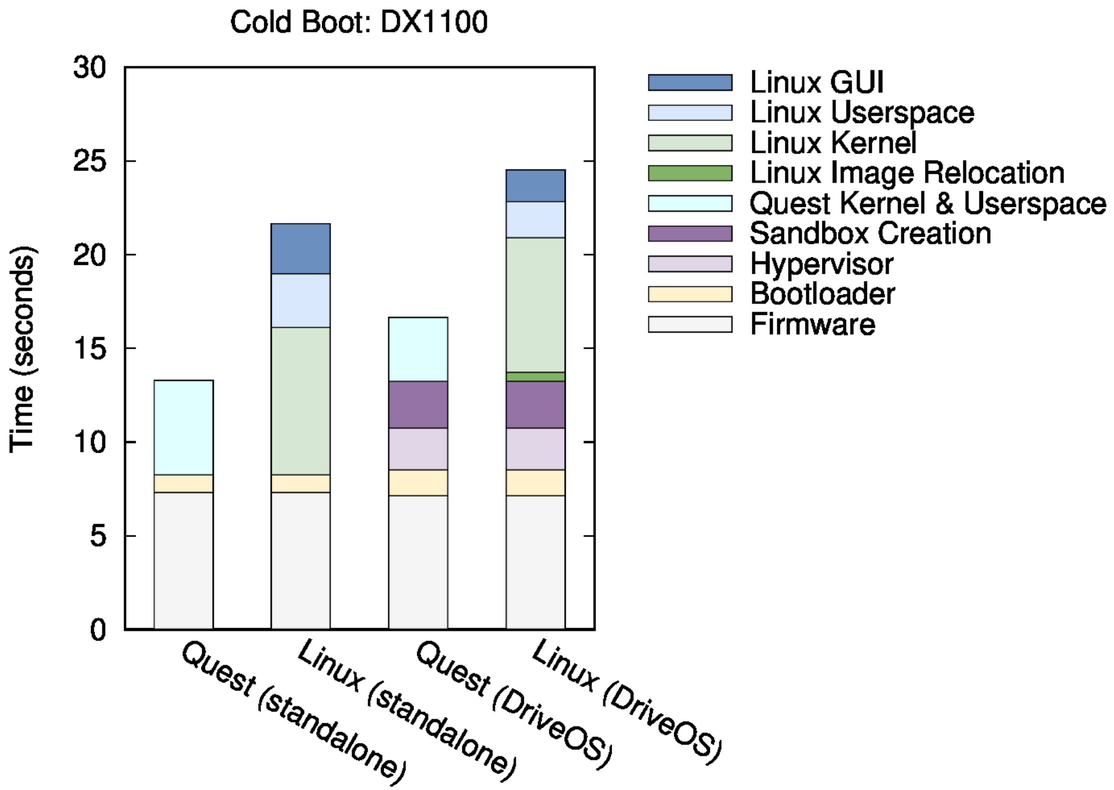
Jumpstart Power Management (RTAS’22, JuMP2start -- ECRTS’24)

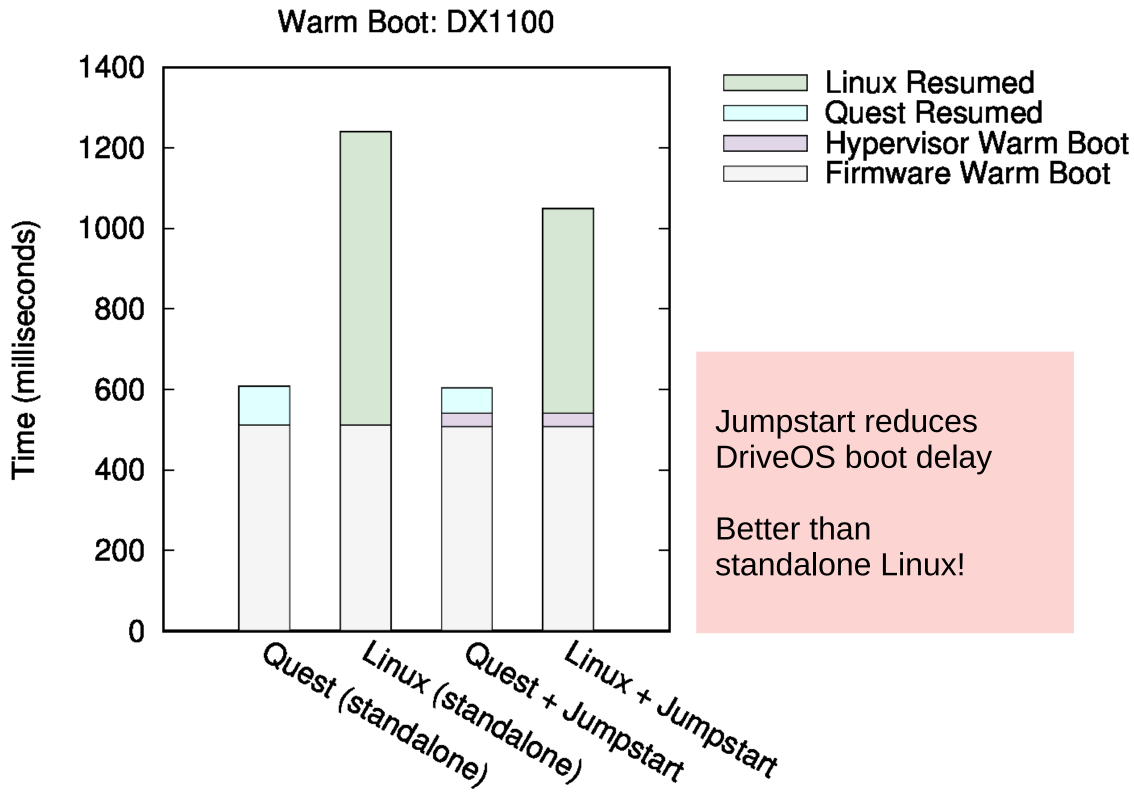
- PC hardware requires Firmware POST, bootloader, device & service initialization to boot OS
- DriveOS uses Jumpstart ACPI S3 suspend-to-RAM & resume-from-RAM for low latency restart of critical tasks (e.g., CAN gateway services)



Jumpstart Power Management (RTAS'22)

- Jumpstart services span all guests
  - RTOS coordinates suspension but enables parallel reboot
- Potential for ACPI S4 suspend-to-disk using non-volatile memory (e.g., Intel Optane)
  - Eliminates system power usage during suspension



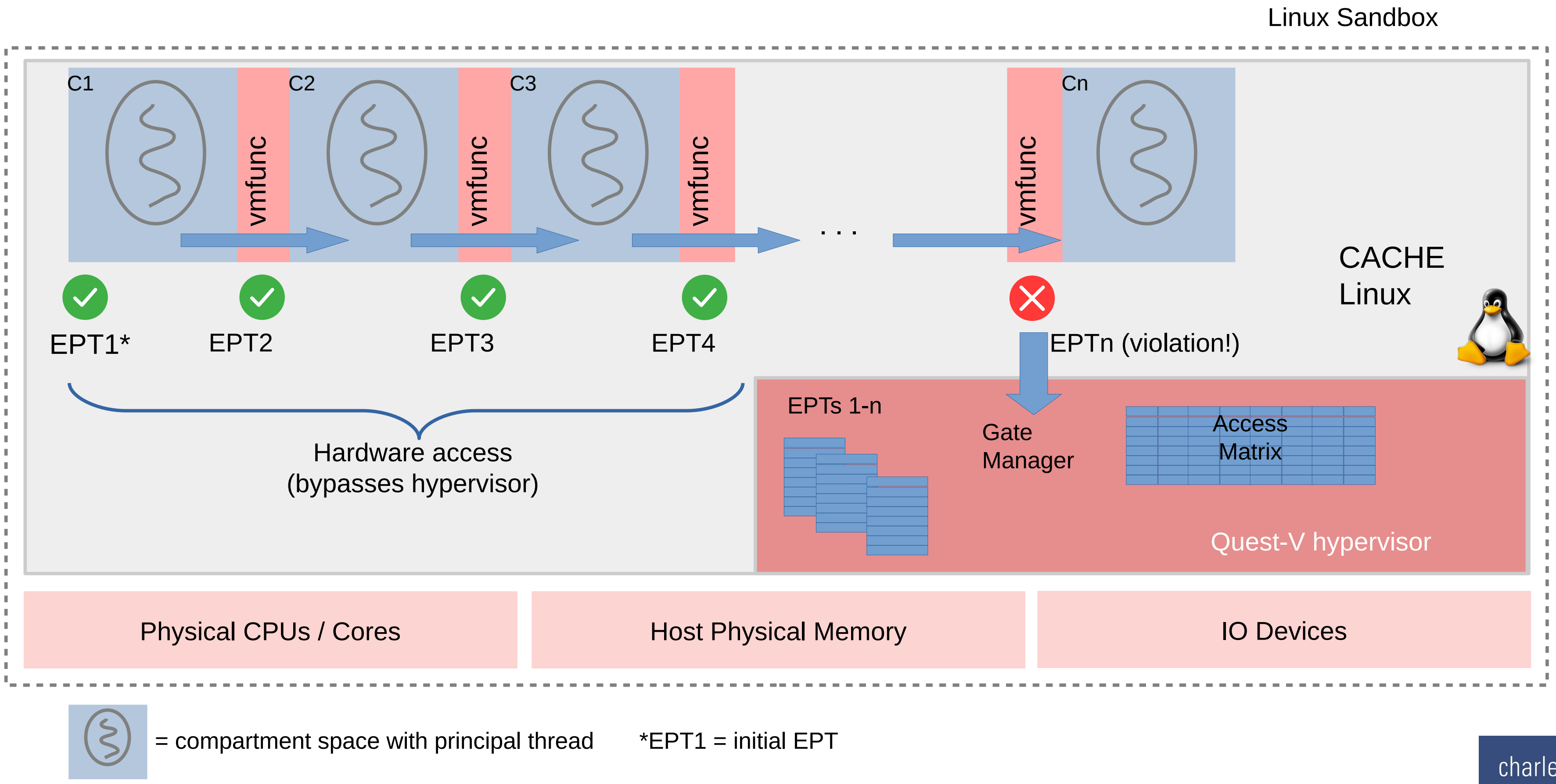


Jumpstart reduces DriveOS boot delay

Better than standalone Linux!

Security Challenge

- Split sandboxes into fine-grained compartments -- “principle of least privilege”
- **Is it possible to automatically convert a monolithic kernel into a micro-kernel with compartmentalized capabilities?**
- Use separate extended page tables (EPTs) per compartment rather than one per guest
- **vmfunc calls optimize EPT switching without trapping into hypervisor**

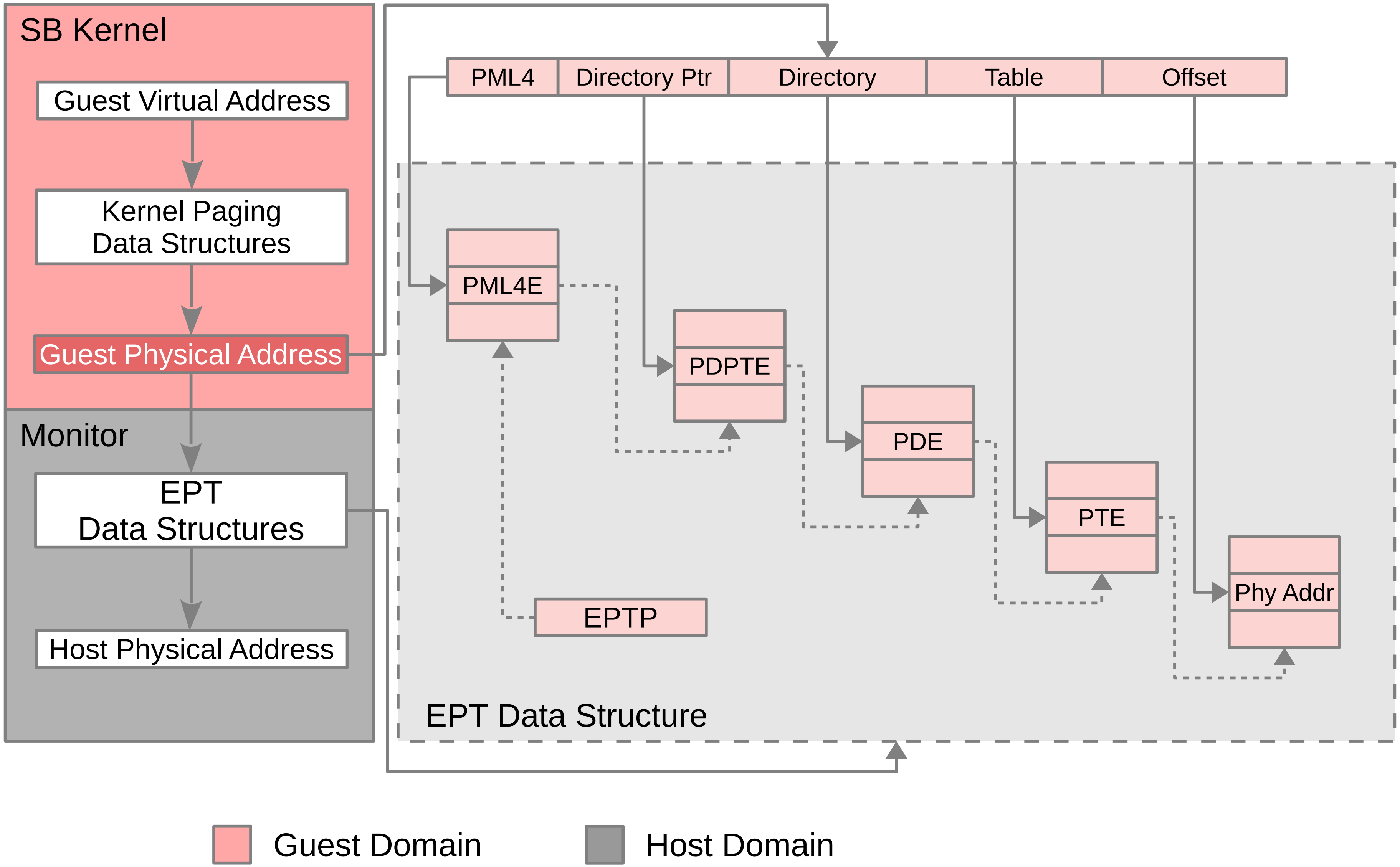


DRAKO

charles river analytics

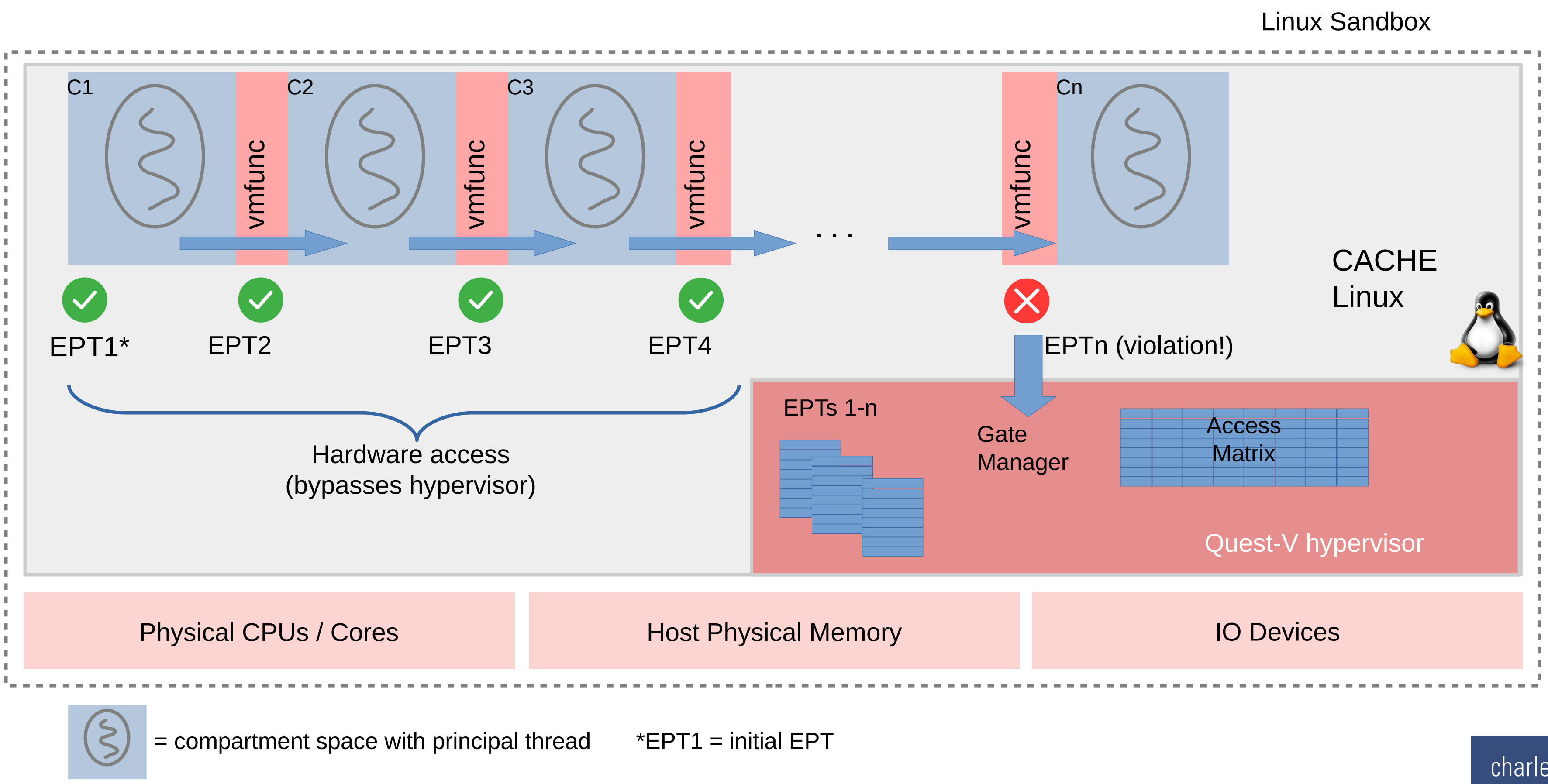
BOSTON UNIVERSITY

Memory Partitioning using Extended Page Tables (EPTs)



CACHE – Compartmentalization Architecture using Commodity Hardware Enforcement

- Quest-V annotates Linux kernel with vmfunc calls using Gate Manager access matrix
- Each vmfunc invocation triggers a new EPT mapping for the next compartment
- EPT violations are trapped by the Gate Manager in the Quest-V hypervisor



DRAKO

charles river analytics

BOSTON UNIVERSITY

CACHE – Test Platform

- Supermicro X13SRN-H 13th Gen Intel Core (i7-1370 PE) Processor
- Yocto Linux 5.15.137 and Quest run on the Quest-V hypervisor

CACHE – Quest EPT Switching Benchmark

- EPT Switching with VMCALL or VMFUNC
  - (1) VMCALL into hypervisor to switch to different EPT by passing index of EPTP in EAX
  - (2) VMFUNC in guest to switch to different EPT
    - EAX = 0x0 → EPTP switching VMFUNC operation
    - ECX = index of EPTP to switch to (**can have up to 512 EPTs compared to 16 MPK regions per core!**)

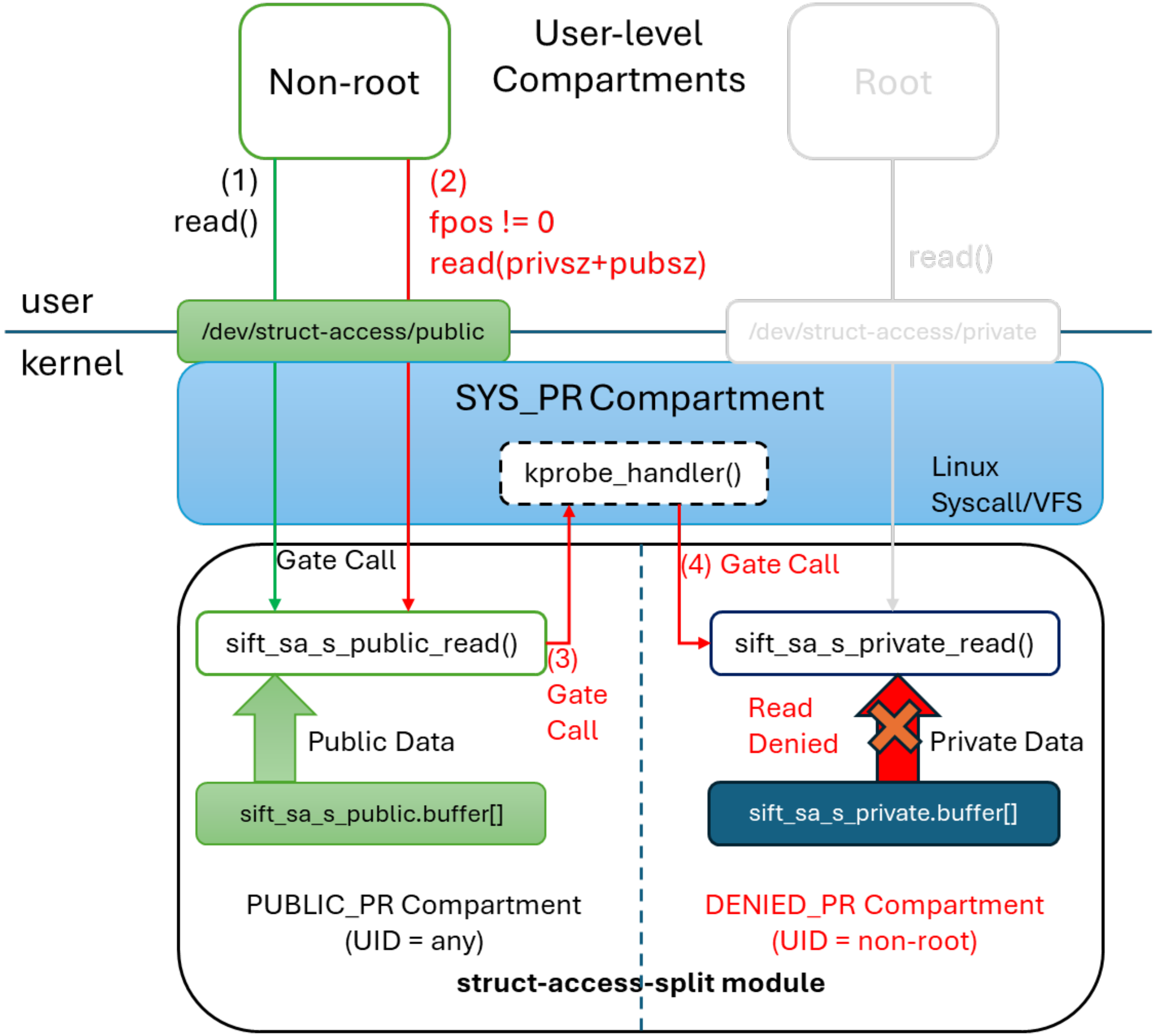
- Excluding **average RDTSC overhead of 35 cycles**, the EPT switching time (averaged over 1,000 iterations) is:
  - **(1) 1271 cycles** with VMCALL (w/o VPIDs), **1013 cycles** w/ VPIDs
  - **(2) 287 cycles** with VMFUNC (w/o VPIDs), **133 cycles** w/ VPIDs

  - Time measures the base cost of a gate call from one compartment to another
    - Discounts checking access rights

CACHE – Example Linux + Kernel Module

- Three main compartments:
  - SYS_PR: default EPT (index 0) encompasses the core kernel; accessible to all users
  - PUBLIC_PR: contains the public buffer; can be accessed by any user (index 1)
  - PRIVATE_PR: contains the private buffer; only accessible to the root user (index 2)

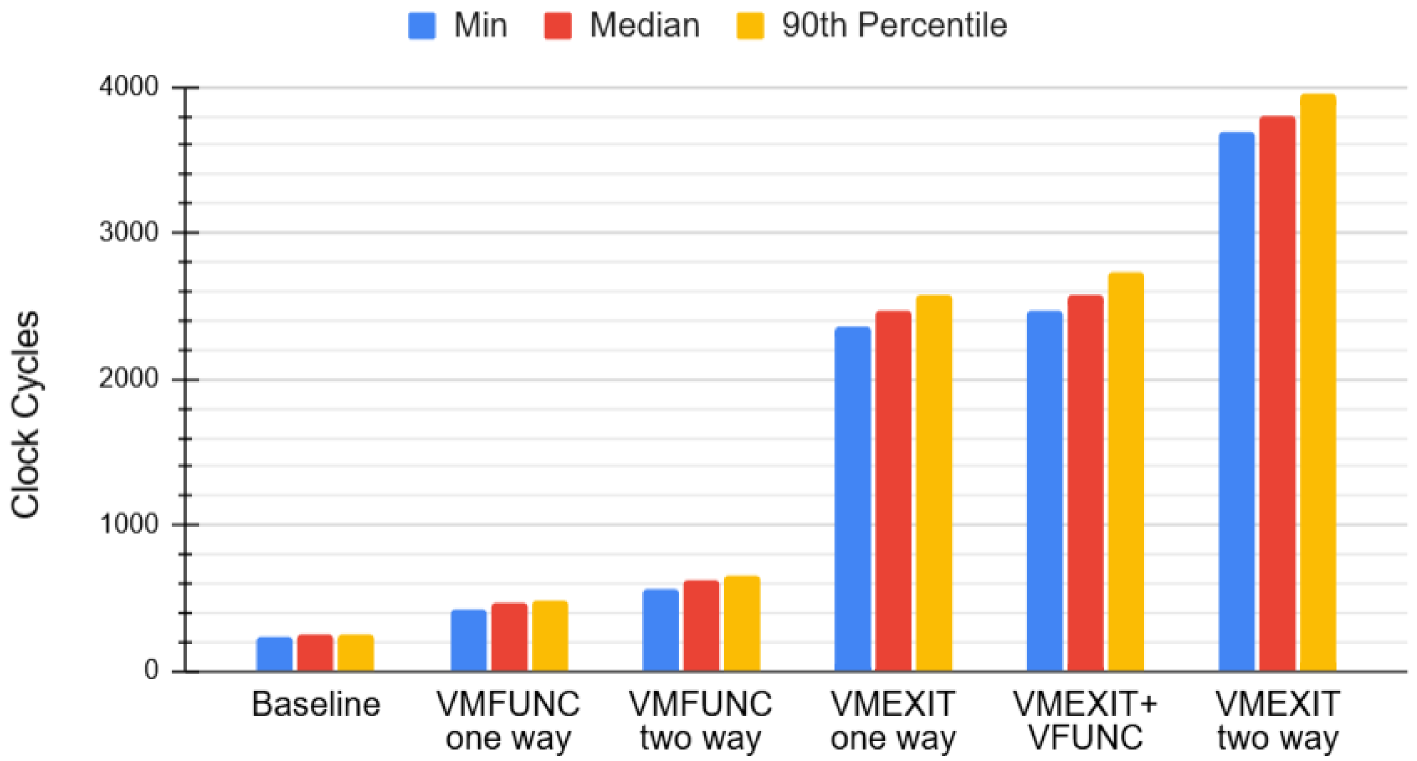
  - DENIED_PR: implicitly created when PUBLIC_PR attempts to access the private buffer



CACHE – Cost of Linux Compartmentalization

Time to execute the private write function under different conditions:

- Baseline: no compartmentalization
- VMFUNC one way: VMFUNC is used to switch to PRIVATE_PR
- VMFUNC two way: VMFUNC is used to switch to PRIVATE_PR and back to SYS_PR
- VMEXIT one way: EPT violation causes a trap, which leads to gate switch
- VMEXIT+VMFUNC: cost of trapping an EPT violation, and a VMFUNC-based switch to SYS_PR
- VMEXIT two way: cost of trapping due to EPT violation and then an explicit gate call to SYS_PR



Quest-V Summary

- Separation kernel – a.k.a. distributed system on a chip
- Uses hardware virtualization to partition resources into sandboxes
- Can use multiple EPTs to enforce finer-grained compartmentalization
- Secure communication channels b/w sandboxes and compartments
- Sandboxes responsible for resource mgmt – avoids monitor involvement

DriveOS Takehome Messages

- **Functional consolidation requires a multicore architecture**
  - Less about ARM vs x86 (or RISC-V) and more about capabilities
    - e.g., VT-x, AMD-V, IOMMU (VT-d, AMD-Vi), security (VT-rp, VMFUNC, MPKs,…)
- **Scheduling is only part of the problem**
  - Multiple cores hard to fully utilize
  - We need pipeline scheduling (dataflow management)
  - Real-time data distribution necessary ("real-time ROS")
- **Real-time I/O is necessary**
  - USB makes sense here, also for networking primary + backup
  - Do we really need TSN in a centralized system? At best, a zonal approach could benefit from a simple USB network or a hybrid of TSN and USB
- **Linux alone isn't enough** – more than just trying to make it certifiably real-time
  - Need to address security