A SOFTWARE AND HARDWARE ARCHITECTURE FOR NEXT-GENERATION AUTOMOTIVE SYSTEMS

DR RICHARD WEST
CHIEF SOFTWARE ARCHITECT
DRAKO MOTORS

PROFESSOR
BOSTON UNIVERSITY

Background

Drako GTE

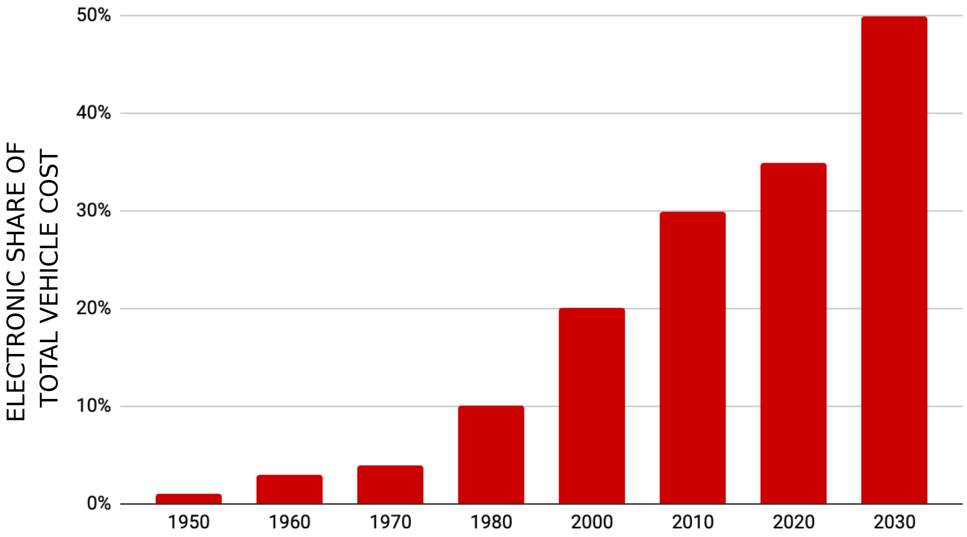
Vehicle Growth in Electronics

- 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 $63.6 billion (2018)
  source: grandviewresearch.com

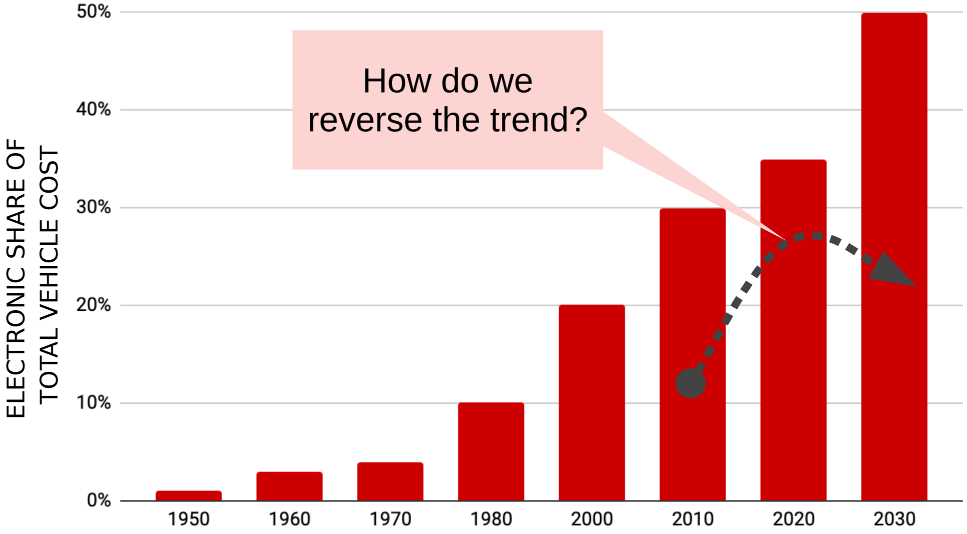
- Electronic share of total vehicle cost is rising exponentially



source: Statista 2017

Vehicle Growth in Electronics

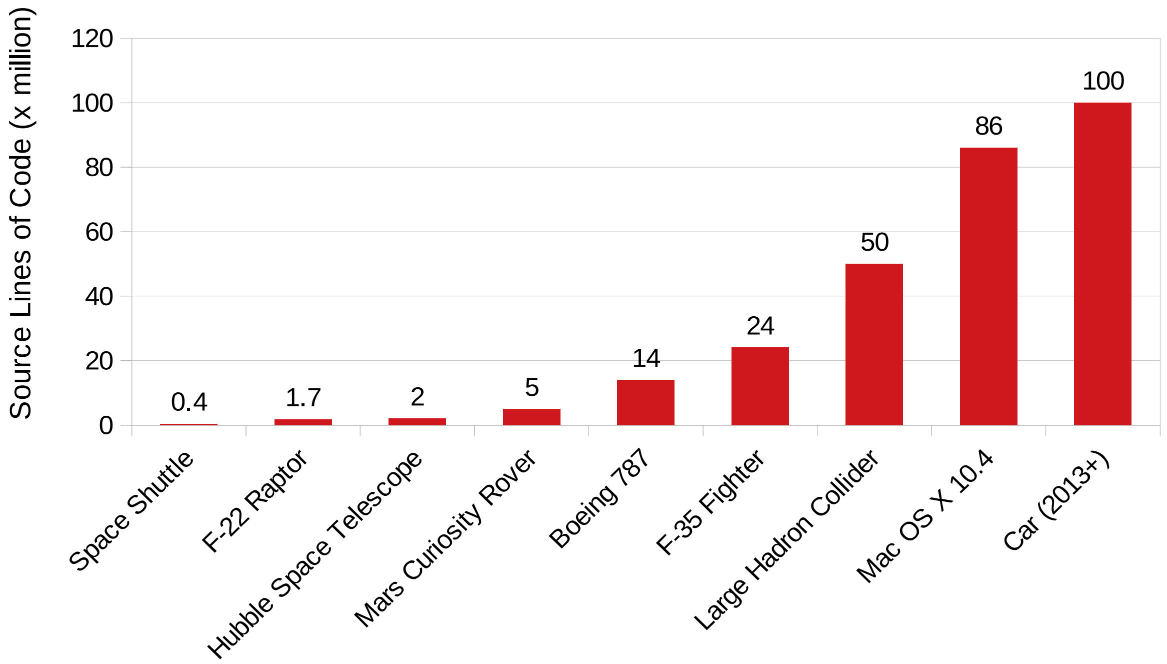
- Electric vehicles, ADAS, IVI, V2X driving up cost and complexity of electronics
- Modern luxury vehicles have 50-150 ECUs
  source: Strategy Analytics, IHS Markit
- Global ECU market $63.6 billion (2018)
  source: grandviewresearch.com
- Electronic share of total vehicle cost is rising exponentially



source: Statista 2017

Automotive Software Complexity

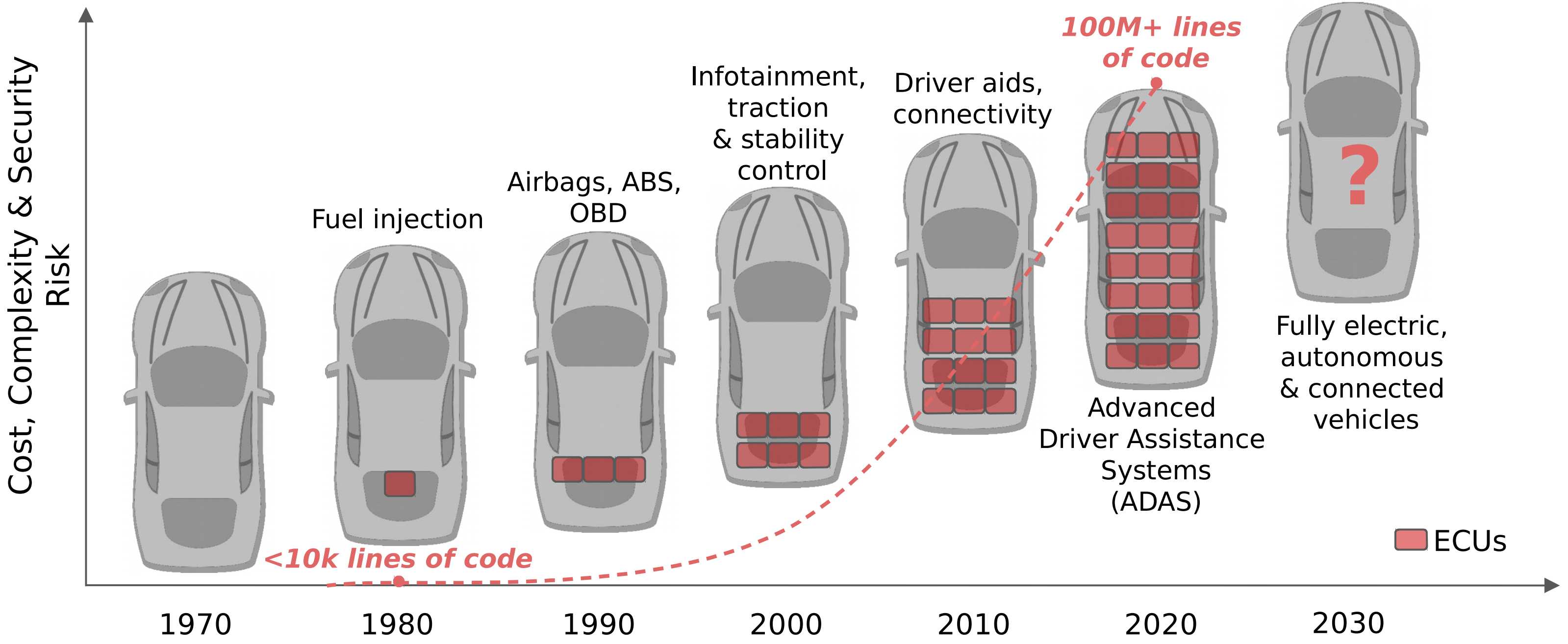
Growth in automotive electronics has given rise to growth in software complexity



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

Software Explosion

Software growth driven by increased vehicle functionality + increased ECU count



DRAKO

BOSTON UNIVERSITY

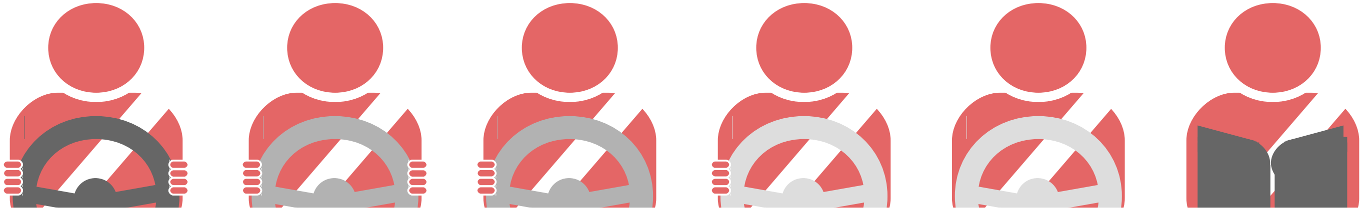
ADAS – SAE 6 Levels of Driving Automation

0	1	2
No Automation	**Driver Assistance**	**Partial Automation**
Manual control	Single automated function e.g., active cruise control	Automated steering and acceleration; Human can take control

Human monitors the driving environment

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

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 the driving environment (levels 0–2)

Automated system monitors the driving environment (levels 3–5)

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

Hardware & OS Evolution

AUTOMOTIVE DOMAIN

- 8→ 16→ 32 bit microcontrollers

PC DOMAIN

- 64-bit CPUs, integrated GPUs

Hardware & OS Evolution

AUTOMOTIVE DOMAIN

- 8→ 16→ 32 bit microcontrollers
- Mostly single core, single function

PC DOMAIN

- 64-bit CPUs, integrated GPUs
- Multicore, multiple tasks

Hardware & OS Evolution

AUTOMOTIVE DOMAIN

- 8→ 16→ 32 bit microcontrollers
- Mostly single core, single function
- Typically 10s-100s MHz

PC DOMAIN

- 64-bit CPUs, integrated GPUs
- Multicore, multiple tasks
- GHz clock speed, hardware virtualization

Hardware & OS Evolution

AUTOMOTIVE DOMAIN

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

PC DOMAIN

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

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

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

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

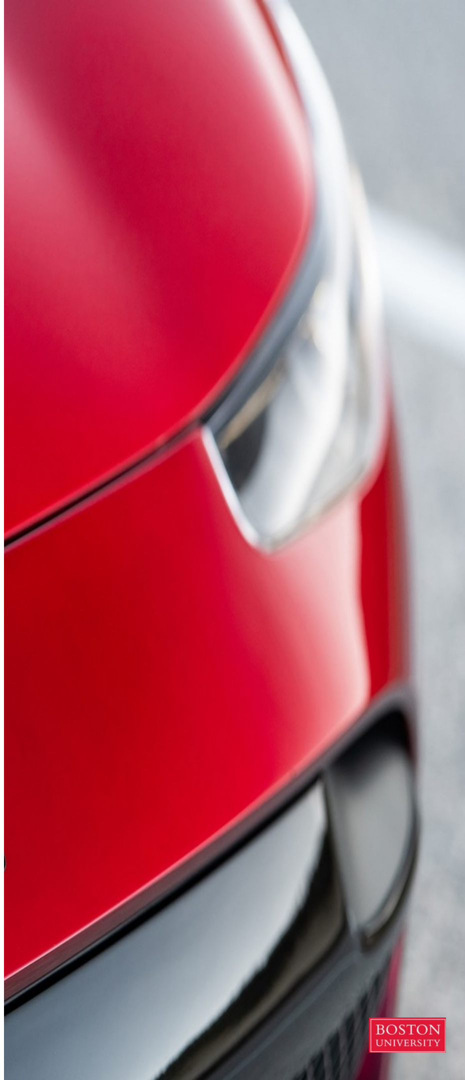
Complex General Purpose OS

- Windows, Mac OS, Linux

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**



Automotive System Challenges

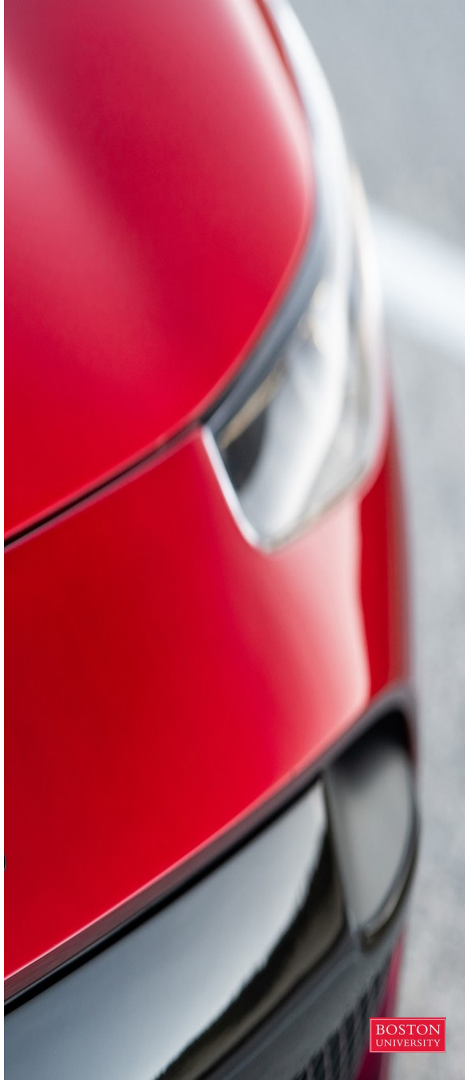
Reduce electronic costs

- 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

- Combined low-latency & high bandwidth data processing
- 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

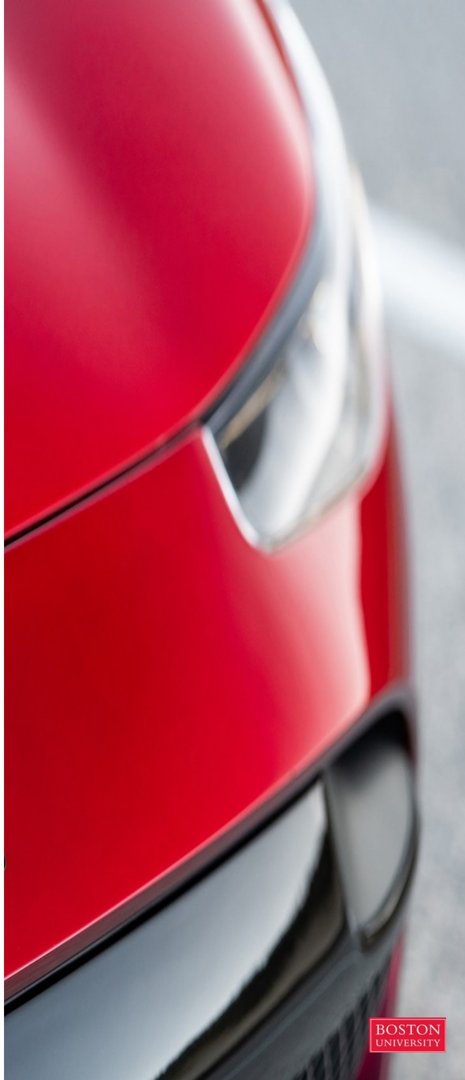
=> Need for **functional consolidation**

Address emerging real-time I/O needs

- Combined low-latency & high bandwidth data processing
- Google's self-driving car (2013) ~ 1GB/s data

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

Functional safety and security (e.g., ISO26262, ISO21434)



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

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:

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

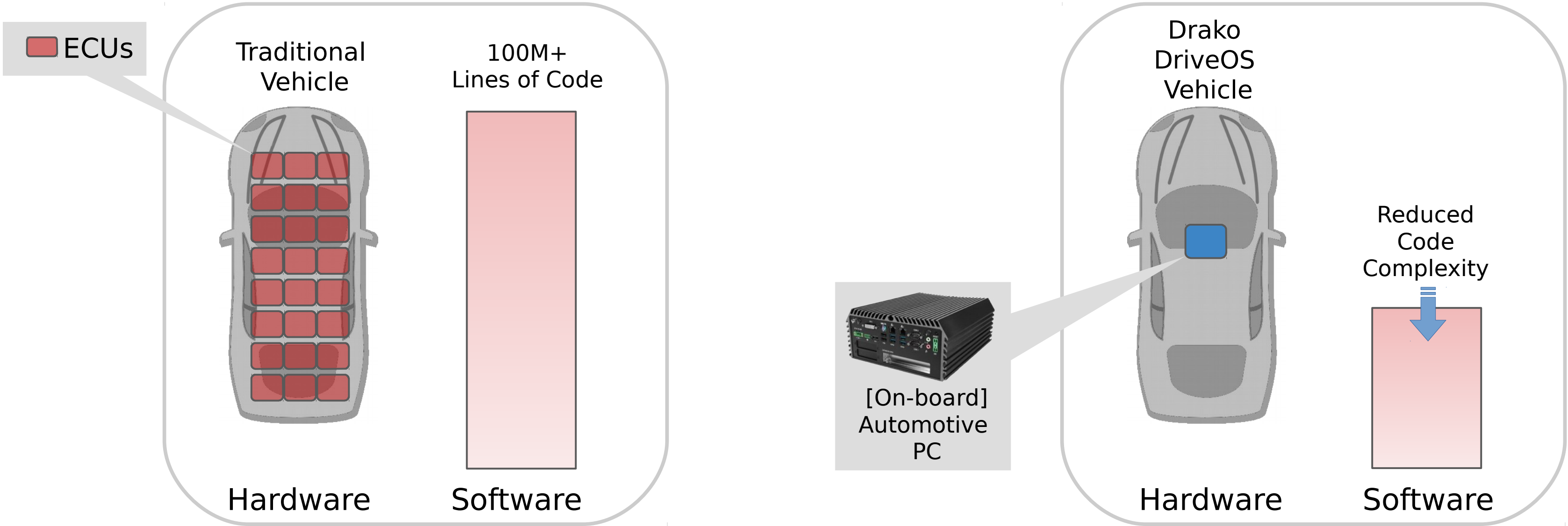
ADAS Failures
Lane Keep Assist, Lane Departure Warning, Collision Avoidance

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

Moving Forward: DriveOS

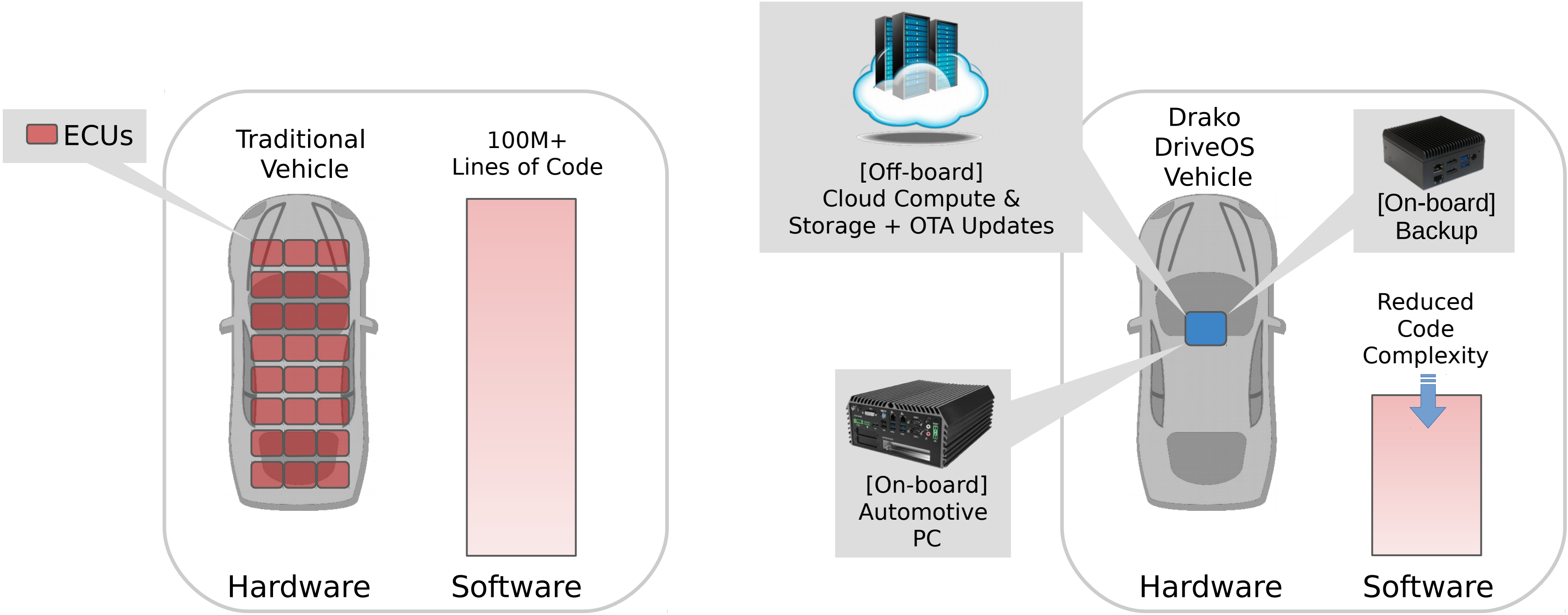
DRAKO DriveOS

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



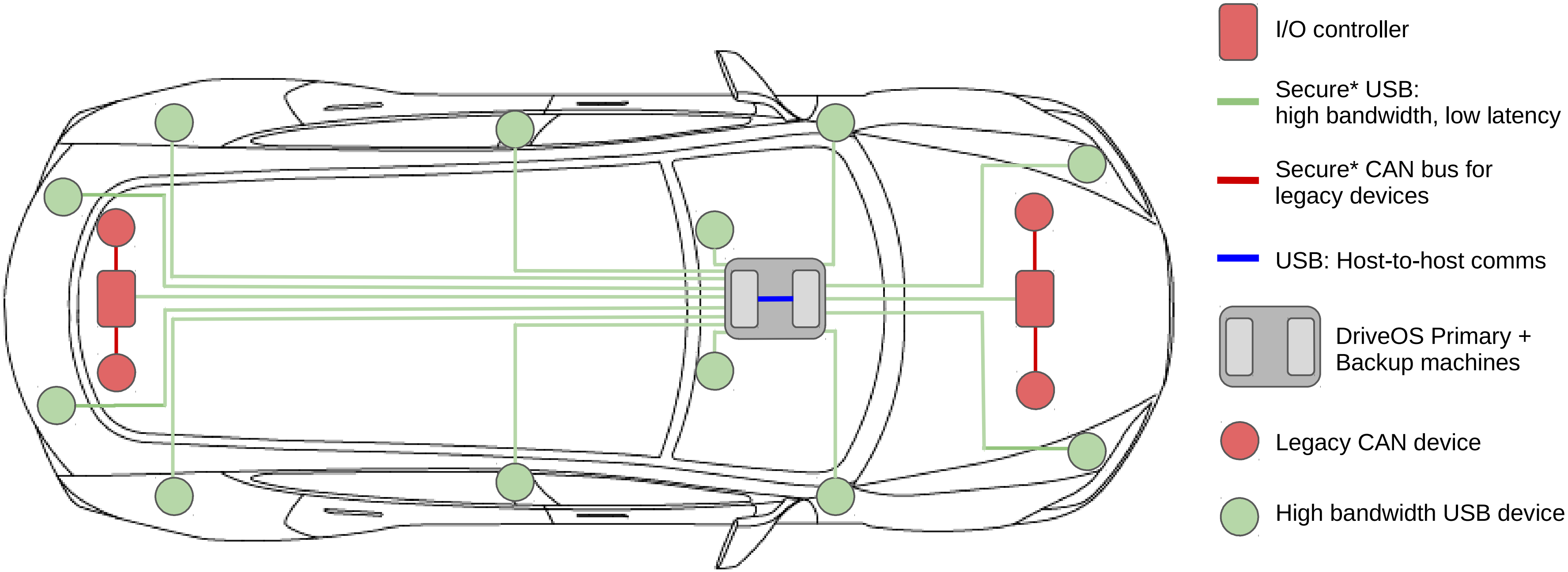
DRAKO DriveOS

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



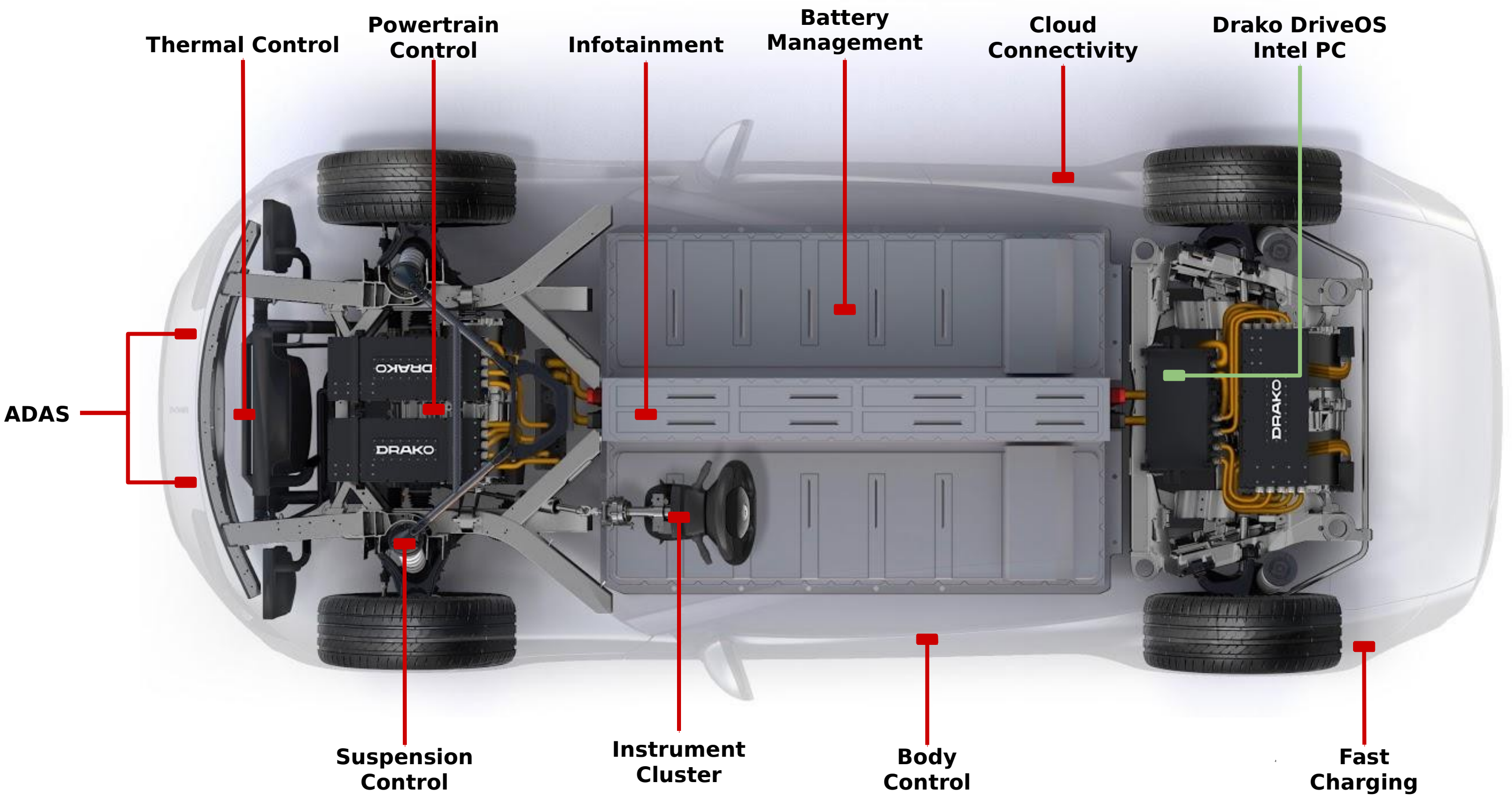
DRAKO DriveOS I/O

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

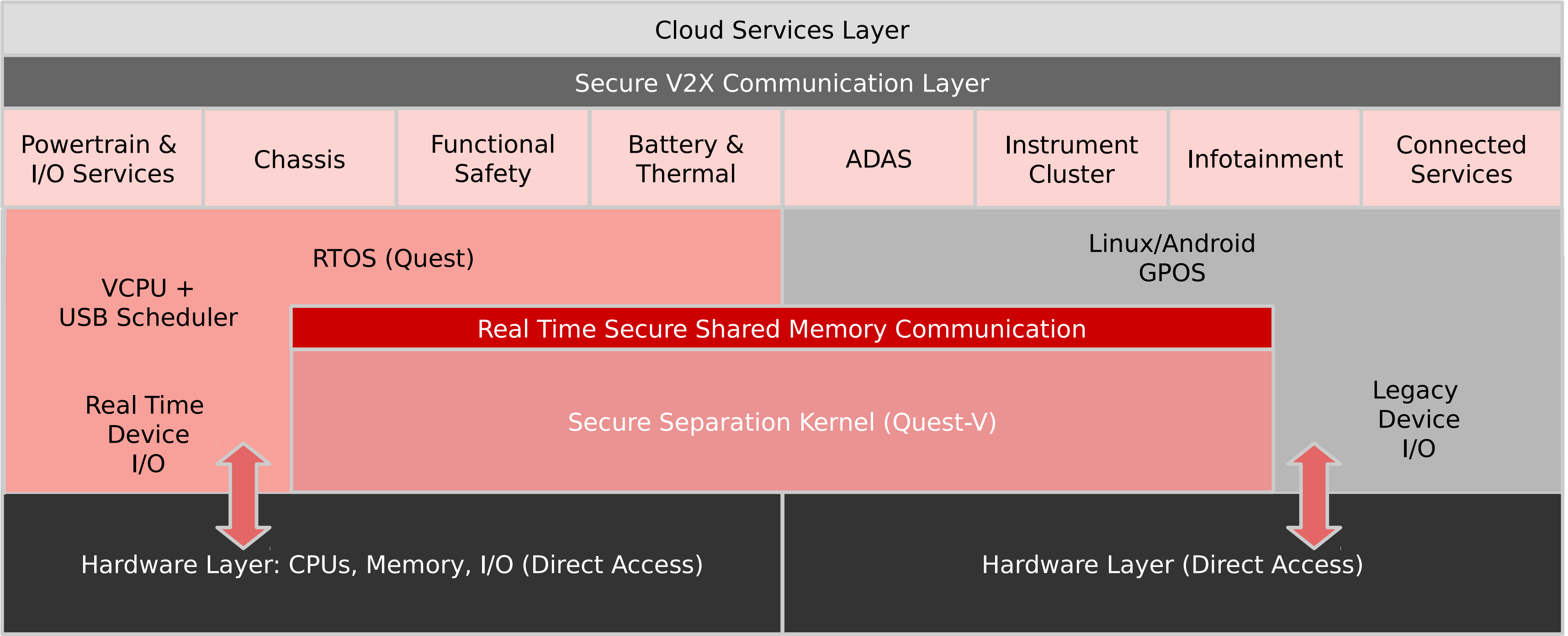


*Secure access to USB + CAN mediated by trusted I/O sandbox in DriveOS

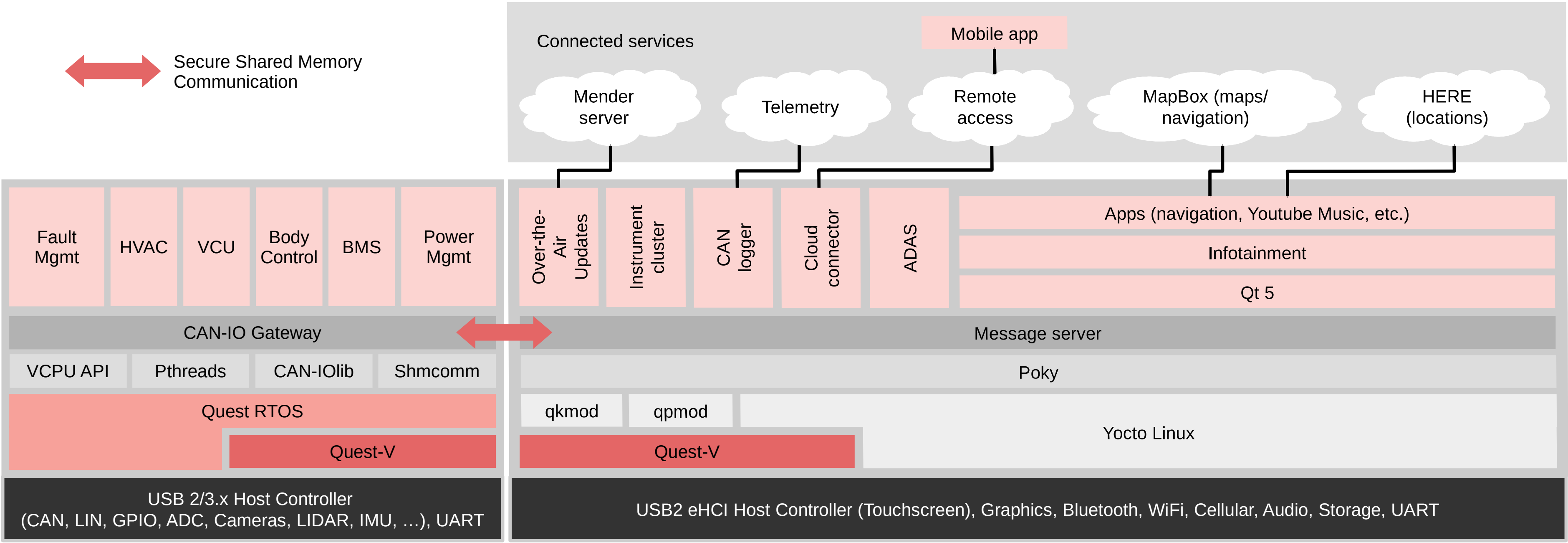
Reference Design: DRAKO GTE DriveOS



DRAKO DriveOS Reference Stack

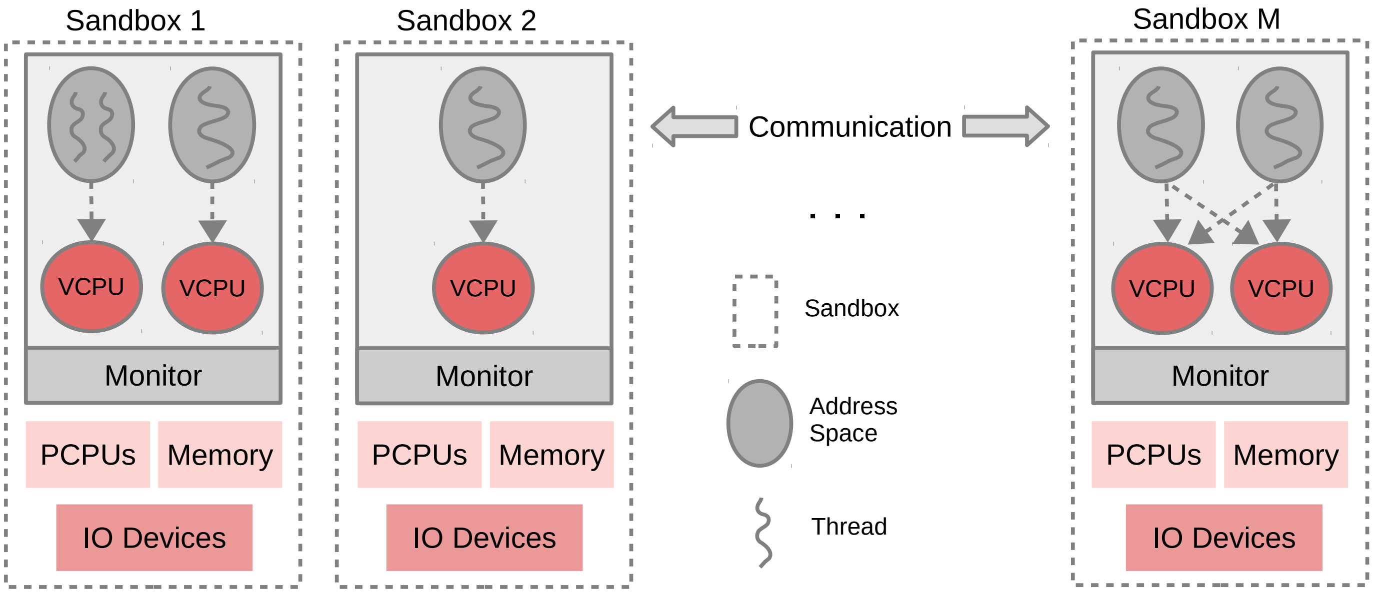


DRAKO DriveOS Functional Overview



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

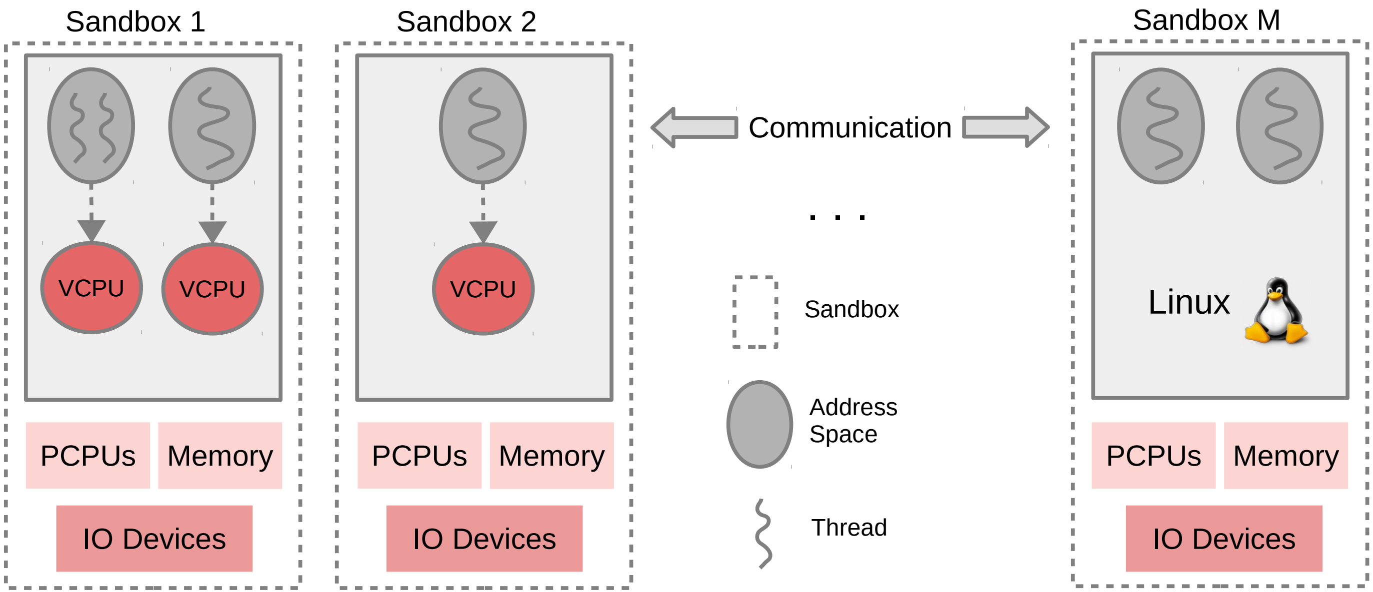


DRAKO

BOSTON UNIVERSITY

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

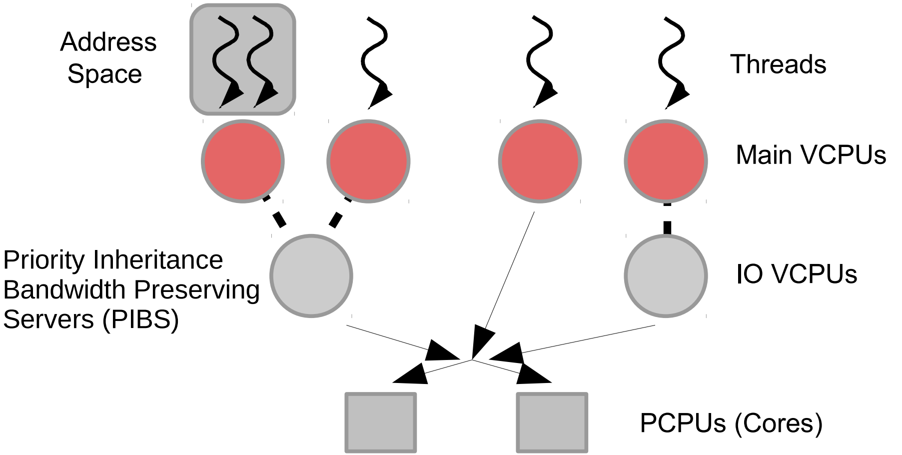


Quest RTOS (RTAS'11)

VCPUs are first-class entities within the RTOS

- 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)



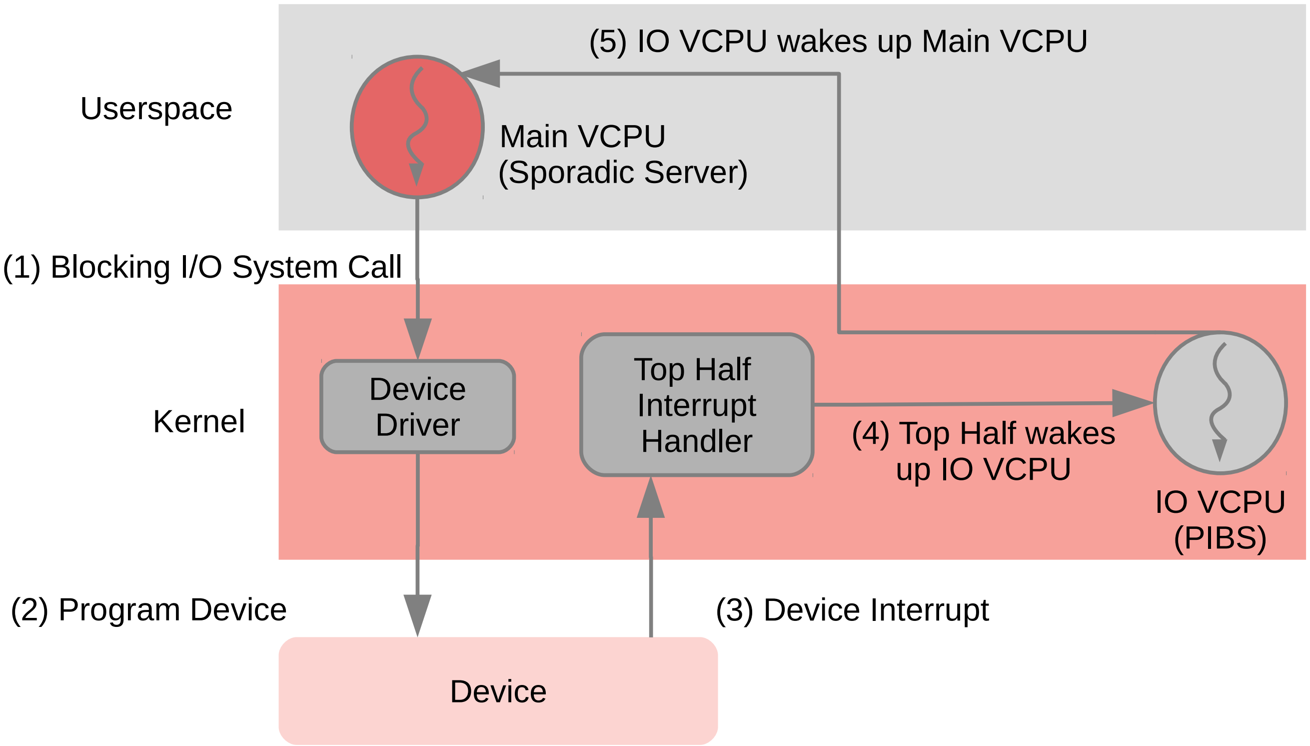
Real-time IO

IOVCPU_CLASS_USB
IOVCPU_CLASS_NET
IOVCPU_CLASS_GPIO
...

DRAKO

BOSTON UNIVERSITY

VCPU Control Flow



VCPU Scheduling (RTAS'11)

Sandbox with 1 PCPU, n Main VCPUs (SS), and m IO VCPUs (PIBS)

- Ci = Budget Capacity of Main VCPU, Vi
- Ti = Replenishment Period of Vi
- Uj = Utilization factor for I/O VCPU, Vj

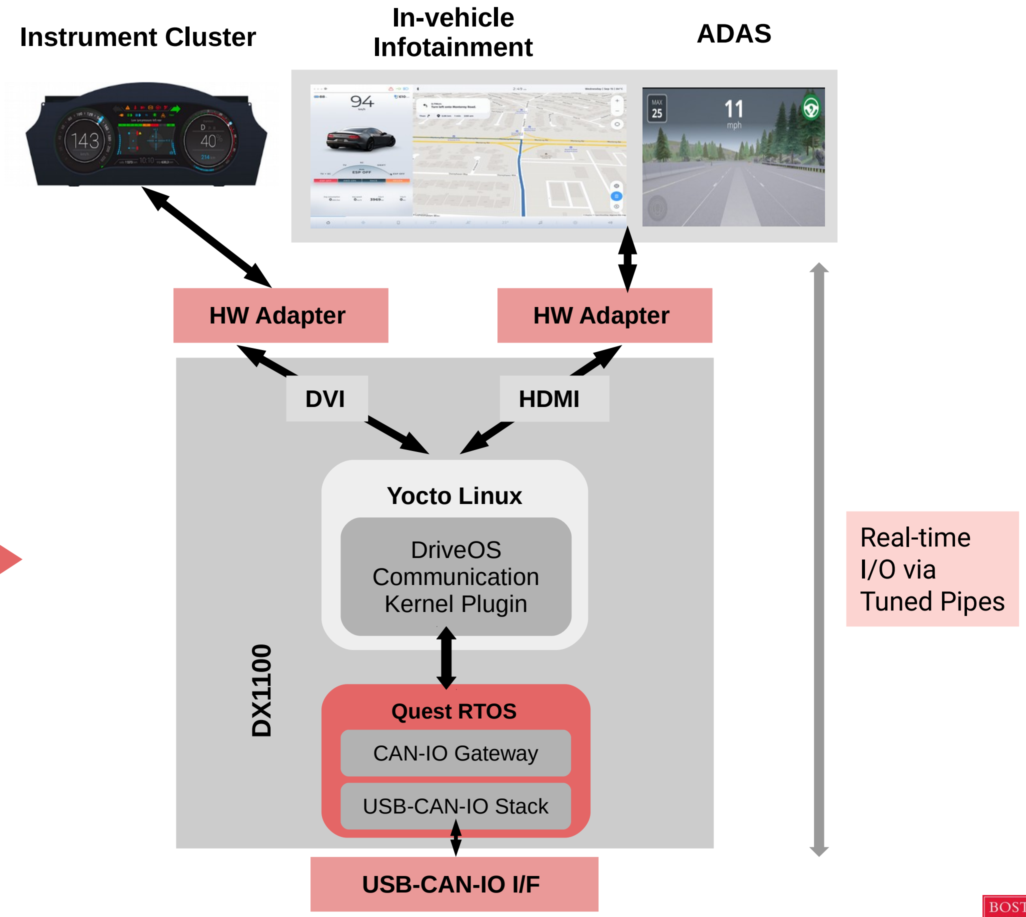
- Utilization bound feasibility test (with rate-monotonic scheduling of VCPUs):

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

Single x86 Multicore PC Solution

Map all services to a
single industrial automotive PC

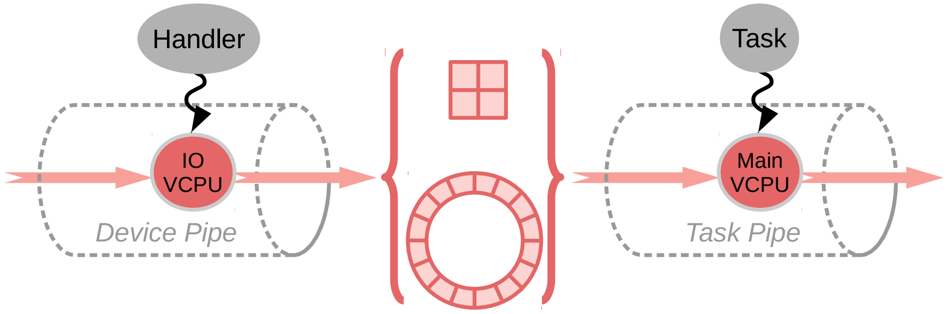
Cincoze DX1100



Tuned Pipes (RTSS'18)

Like POSIX pipes but guarantee throughput and delay on communication

- Simpson's 4-slot (asynchronous) & FIFO (synchronous) buffering



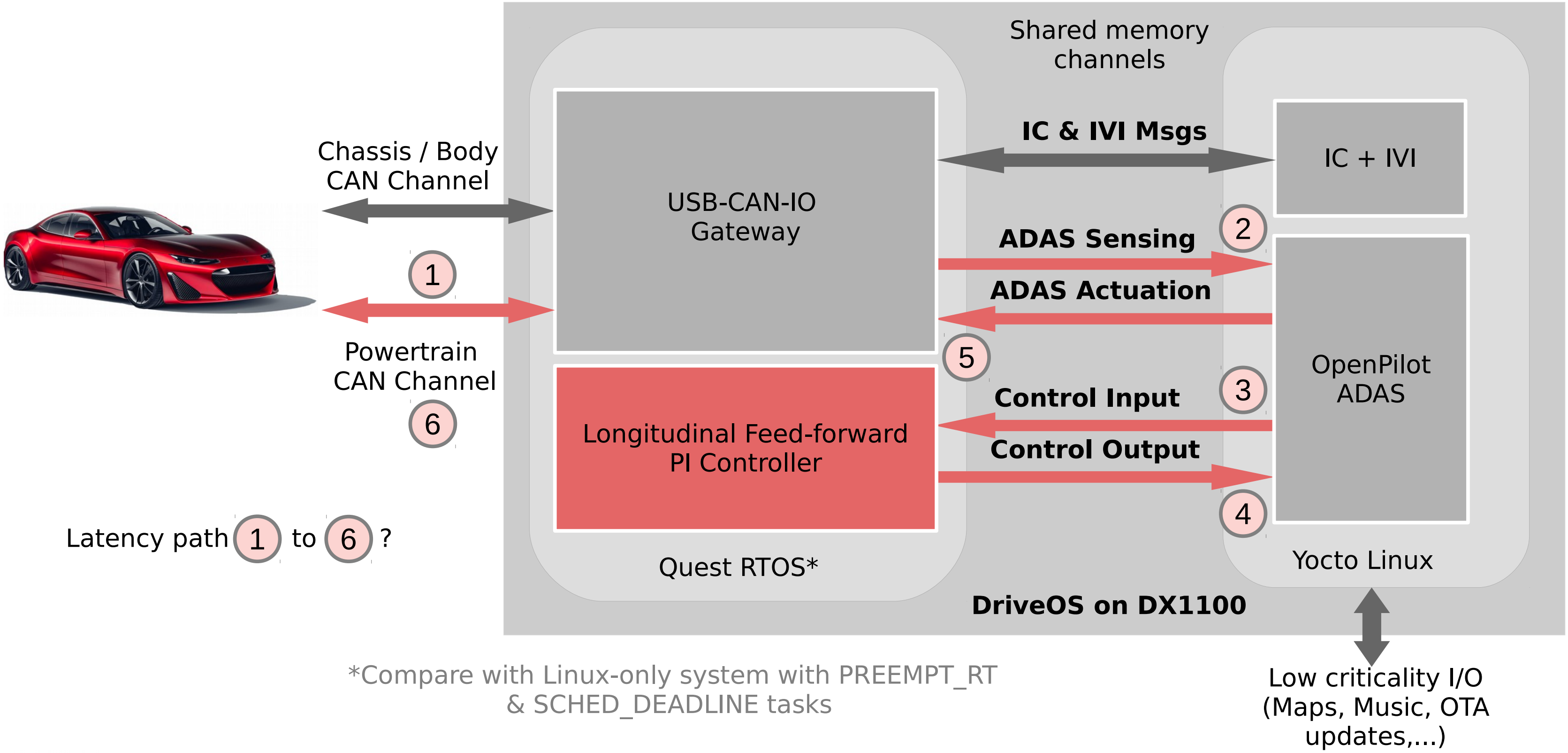
Boomerang I/O subsystem in Quest-V supports real-time pipelines across Quest RTOS and legacy OSes

- Rate match tasks in pipeline to avoid blocking or missed data
- Quest appears as a **real-time virtual device interface** to Linux/Android

DRAKO

BOSTON UNIVERSITY

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

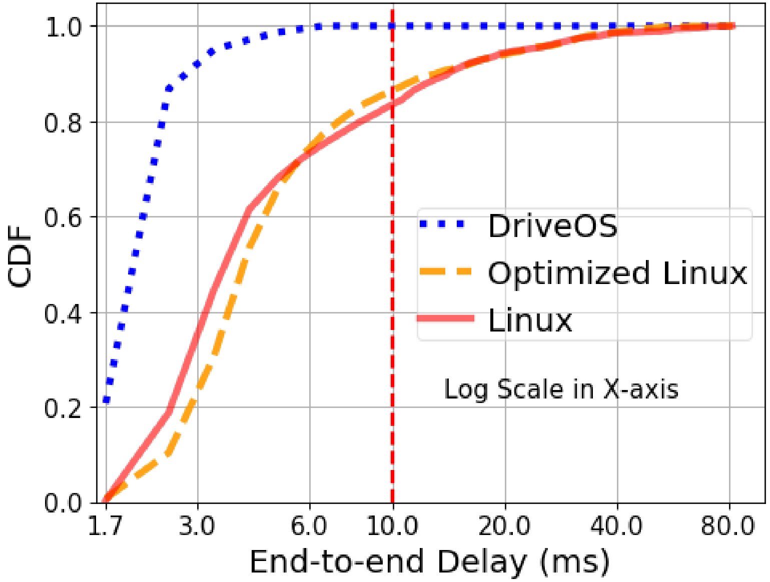


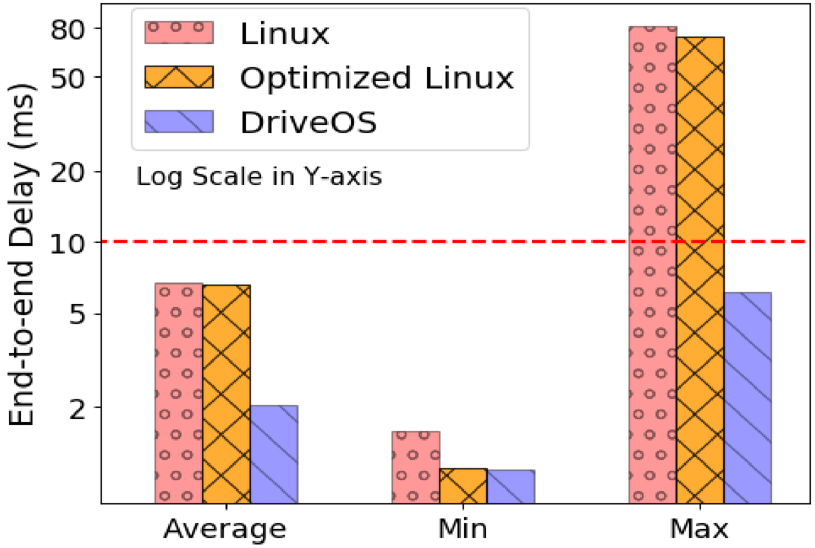
*Compare with Linux-only system with PREEMPT_RT & SCHED_DEADLINE tasks

DriveOS: OpenPilot Control Loop Latency (EMSOFT'21)

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

Target bound = 10ms





*Both Linux cases use PREEMPT_RT. Optimized Linux maps USB interrupts to separate core

Conclusions

Now is the time to look to alternative hardware + OS automotive solutions

DriveOS uses hardware virtualization for real time temporal and spatial isolation of software functions

- Multicore PC-class platform replaces ECUs with software tasks
- USB-centric I/O control
- Symbiosis between RTOS & legacy OS
- Real-time I/O & task pipeline processing

Fast startup of critical services on PC-class hardware (RTAS'22)

Key Points

- **Functional consolidation** to drive down costs of electronics
- **Centralized software stack** to reduce hardware + code complexity
- **Must consider OS challenges**
  – More than just supporting driverless & connected cars
  – ML is great for object detection but an RTOS is needed to avoid objects!
- **Real-time I/O is critical**

Related Work

Automotive Company / System	Operating System	Features
DRAKO DriveOS™	Quest RTOS, Quest-V Separation Kernel + Yocto Linux / Android	**Centralized**; Quest/Linux/Android sandboxes IC, IVI, HVAC, Powertrain,ADAS, etc Simulink Multi-OS Support
Toyota Entune 3.0 (Future: Arene)	Automotive Grade Linux (Arene: Apex.OS)	Infotainment (Arene will support autonomy)
BMW OS7 and OS8 (iX)	Greenhills Integrity RTOS + Linux	[Linux] Infotainment, IC [RTOS] RT vehicle control functions
Polestar + Google	Automotive Android	Infotainment
Nvidia Drive OS	Nvidia Hypervisor, QNX Neutrino RTOS, Linux	ADAS, Linux + QNX SDK
Ford Sync 3	QNX (current); Android (future)	Microkernel, RTOS, Infotainment
TTTech	Car.OS	Supports AUTOSAR, Linux, QNX + others **Centralized**; IC, IVI, ADAS, HVAC, Powertrain
Mercedes Benz	MB.OS	**Centralized**; RTOS + Linux support IVI, Powertrain, ADAS, Body Control, HVAC
Tesla	Linux + FSD (Full Self Driving)	Infotainment (AMD for Model 3 & Y), ADAS

Questions?

Extra Details

System Software 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

IO Safety

- Controlled access to IO devices

System Security

Integrity

- Avoid attacker compromizing critical functionality
  - e.g., Miller & Valasek, 2014 Jeep Cherokee CAN attack via remote access to IVI
- Resource partitioning and access only via secure interfaces
- Validate arguments to functional interfaces

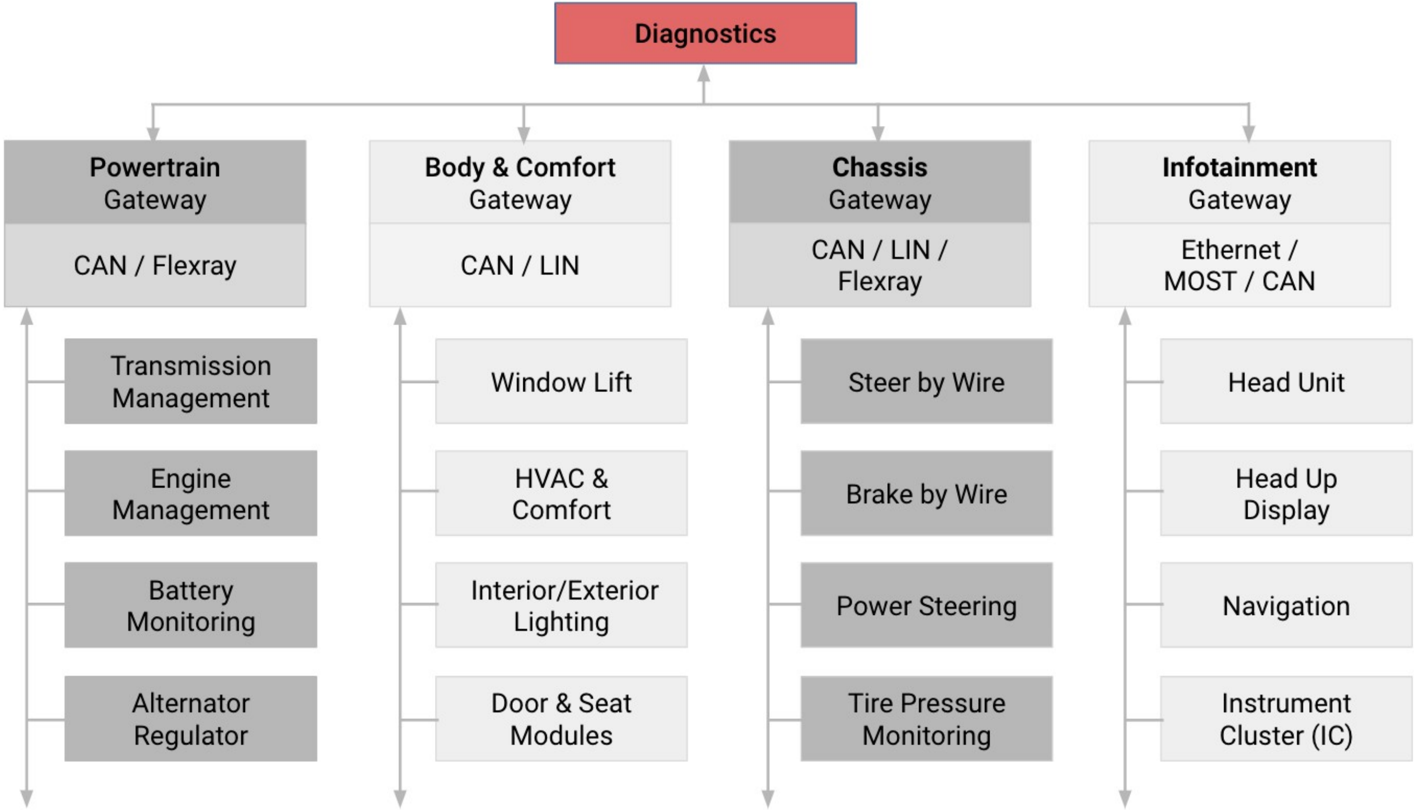
Confidentiality

- Avoid leaking sensitive data (CAN packets, personal information, app data,...)
- Encrypt data or enforce information flow policies
- Eliminate side channels (e.g., via caches – possibly use cache/page coloring)
- Use containerization for critical components

Access Rights

- Avoid user gaining elevated accesses to resources beyond allowed rights
  - e.g., CVE-2019-5736 Breaking out of Docker via RunC
- Enforce a capability mechanism on access to resources
- Digitally sign software images

Today’s ECU Vehicle Network



DRAKO DriveOS

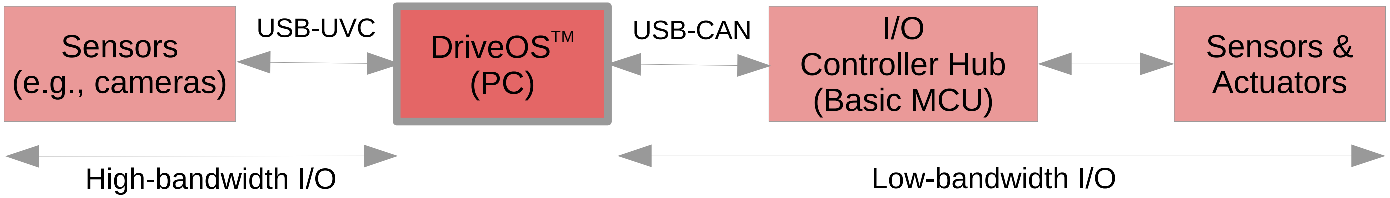
Leverage the Quest-V separation kernel

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

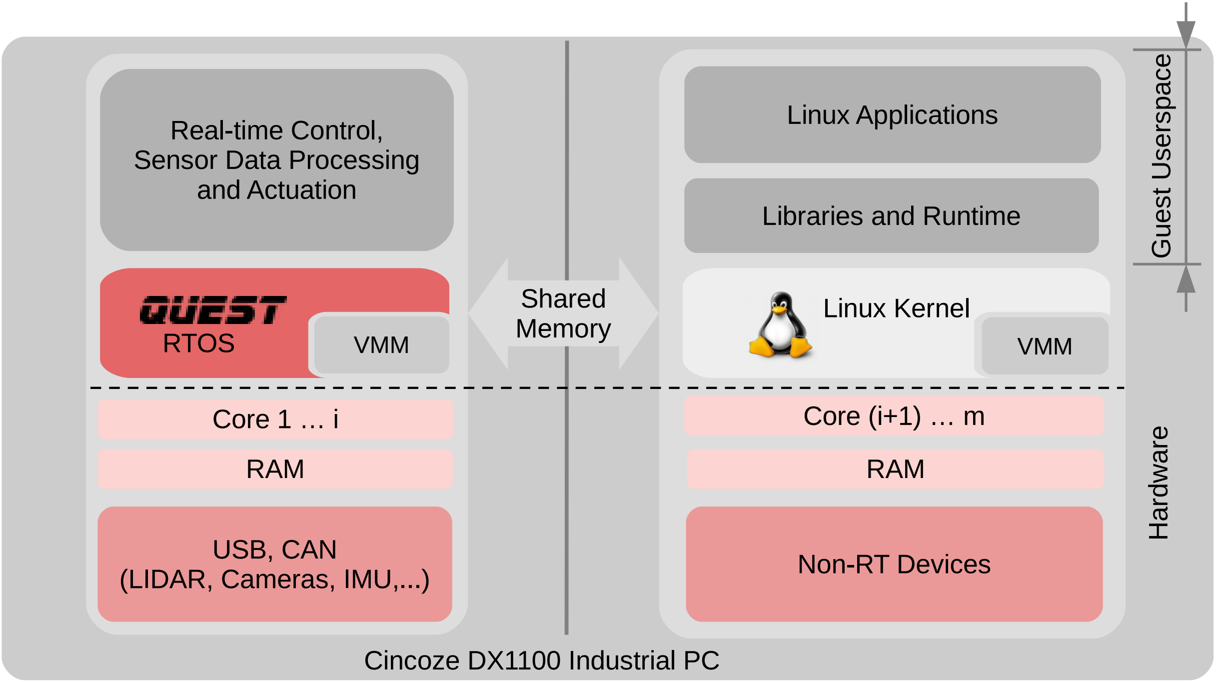
Co-locate Quest RTOS with Linux and Android guests on same hardware

Real-time interface for device I/O

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



Example: Quest-V for DriveOS

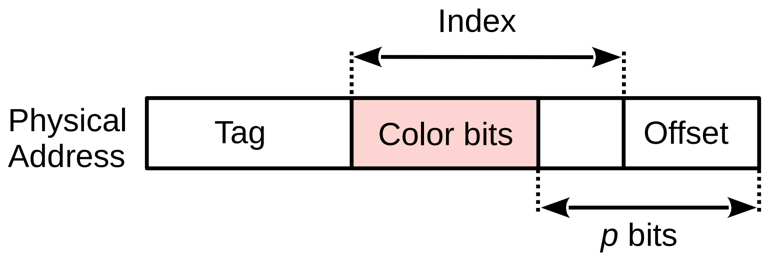


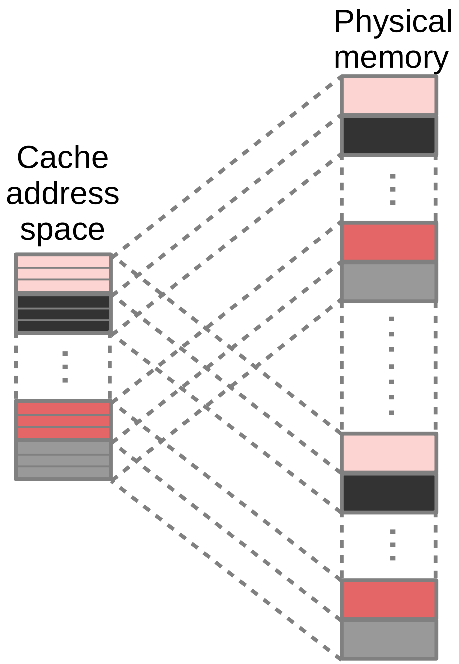
DRAKO

BOSTON UNIVERSITY

Cache Partitioning (Spatial and Temporal Isolation)

- Shared caches controlled using color-aware memory allocator **[COLORIS – PACT'14]**
- Quest-V uses EPTs to map guest physical to machine physical addresses





- Last-level cache occupancy prediction based
- on h/w performance counters
  - local (core) + global (all core)
    cache hits and misses between scheduling points
     [Book Chapter, OSR'11, PACT'10]

Quest RTOS – USB Scheduling (RTAS'13)

USB 2/3.x Bus scheduler

Each periodic request represented as a tuple (wi, ti)

- wi – time to send transaction i
- ti – time interval of transaction i

Given set of n tuples {(w1,t1), (w2,t2),…,(wn,tn)}, is there an assignment of USB transactions to 125uS microframes, such that no frame is over-committed?

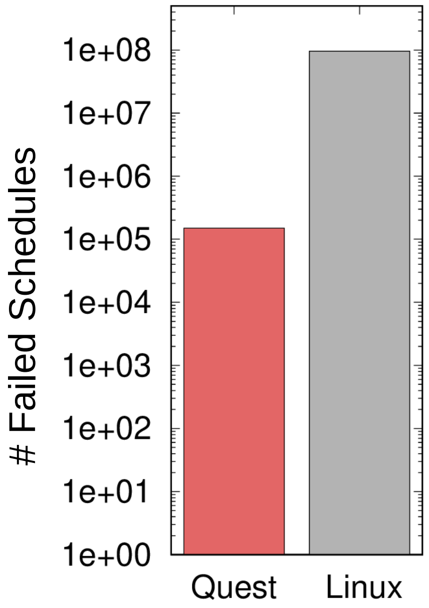
A request assigned to microframe f is also assigned to microframe f+n*ti, n ∈ N

Using variant of first-fit decreasing packing algorithm, shown to outperform Linux

- Sort by decreasing wi (largest first)
- First pick request based on smallest ti, breaking ties with largest wi

Quest RTOS – USB Scheduling (RTAS'13)

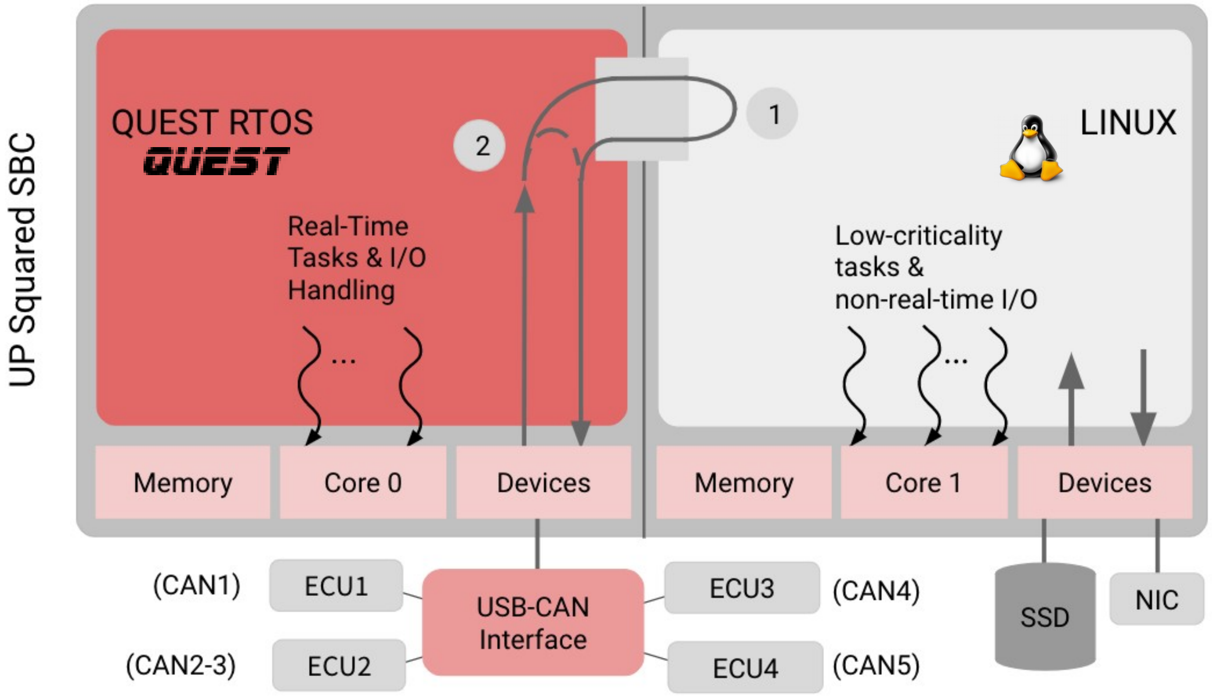
- Consider all permutations of 1 to 5 requests
- Intervals: 2, 4, 8, 16 microframes
- Packet Sizes: 32, 64,…,1024 bytes
- Quest ≈ 150 thousand failed schedules
- Linux ≈ 95 million failed schedules



Boomerang Inter-OS Task Pipeline Example (RTAS'20)

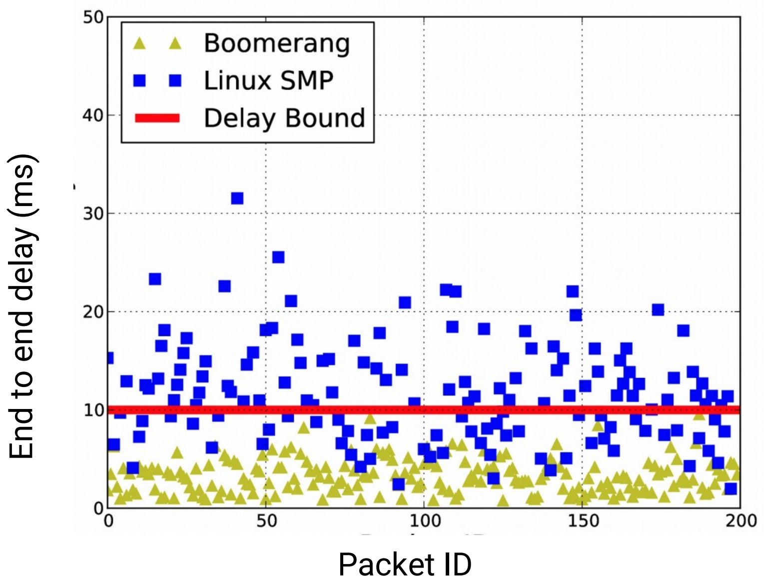
Boomerang tuned pipe path (1) spans Quest + Linux + USB-CAN

Boomerang tuned pipe path (2) spans Quest + USB-CAN

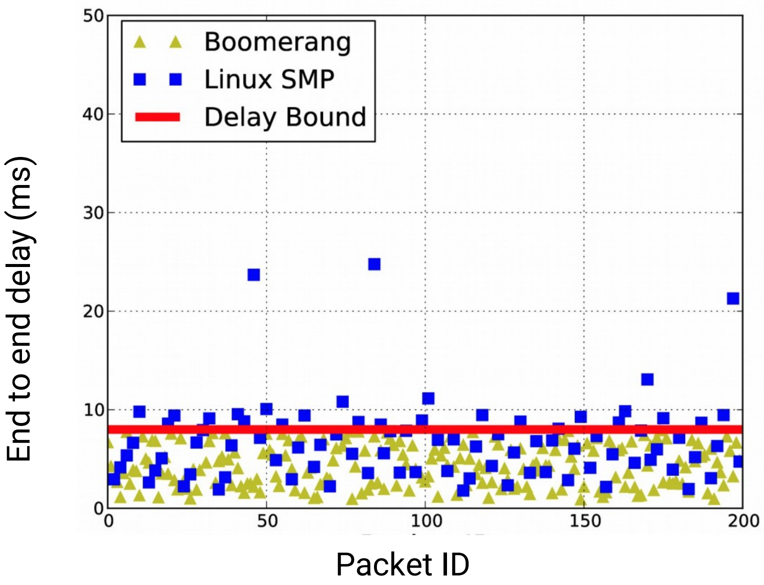


DriveOS: Boomerang Results

Pipeline 1 (dual core)



Pipeline 2 (single core)

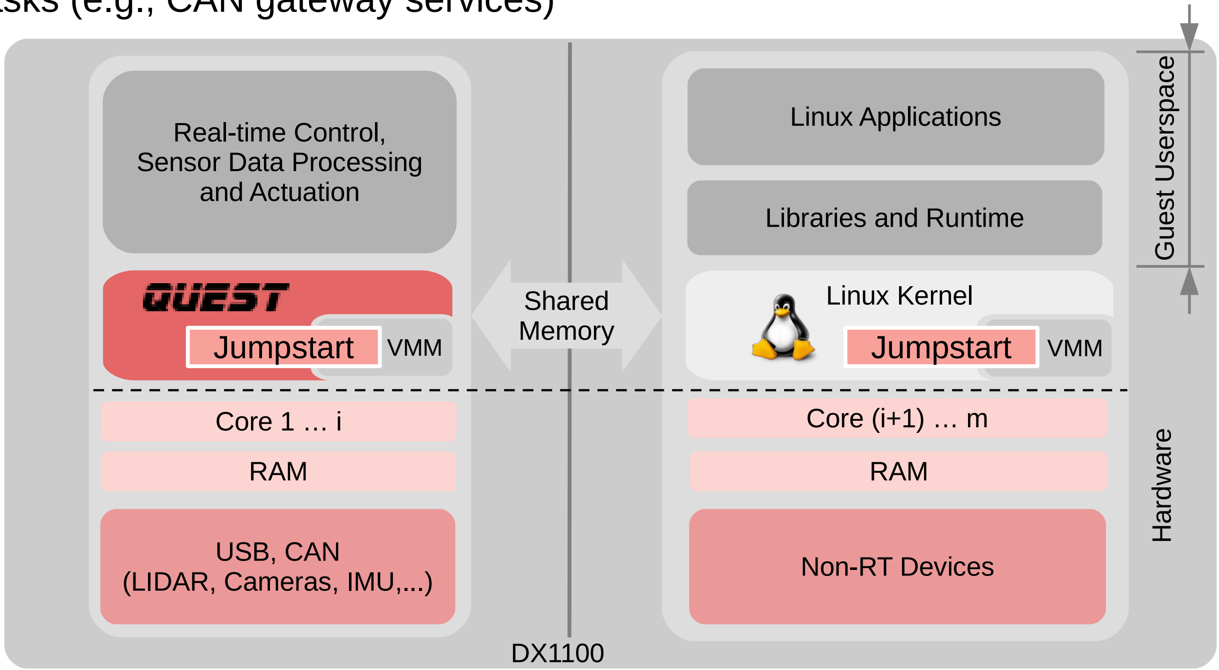


Boomerang sub-system in DriveOS meets communication timing guarantees

A Linux SMP (multicore) OS with real-time extensions cannot perform I/O predictably

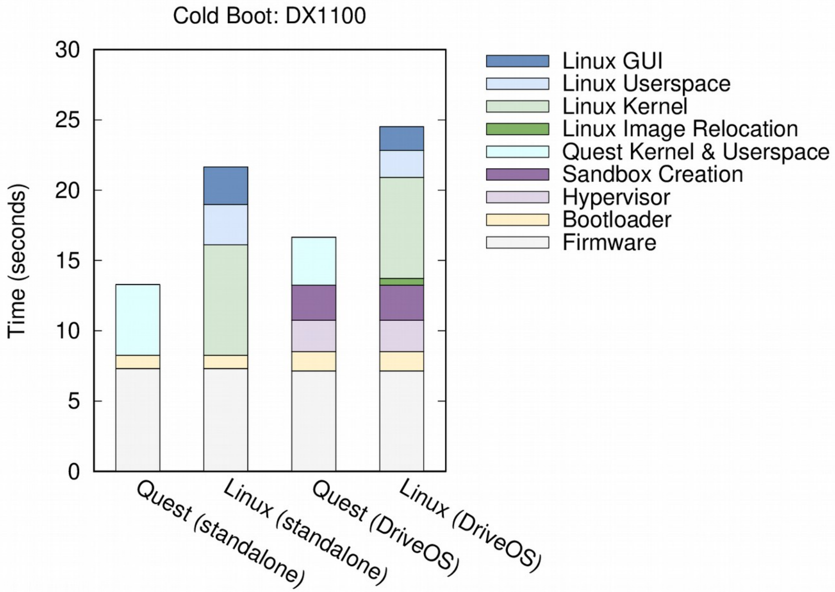
Jumpstart Power Management (RTAS'22)

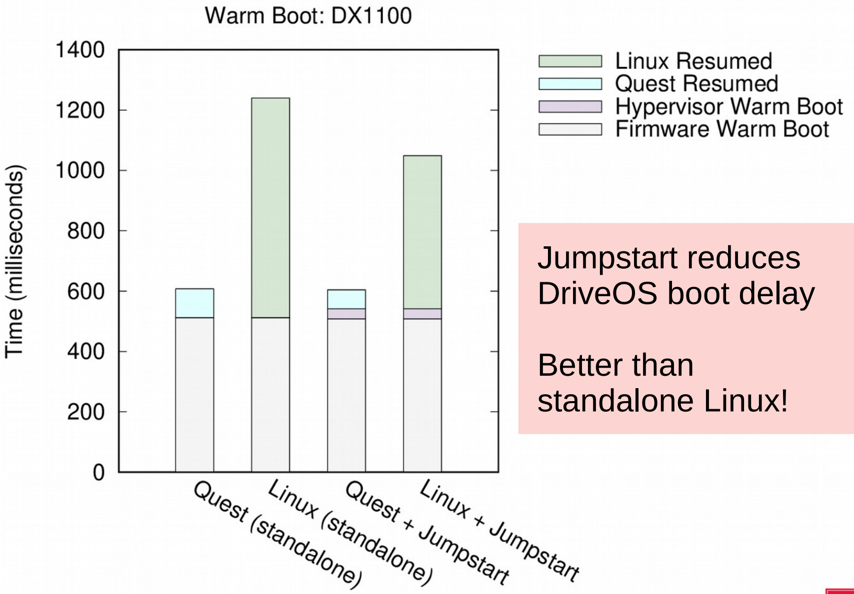
- PC hardware requires Firmware POST, bootloader, device & service initialization to boot OS
- DriveOS uses Jumpstart ACPI S3 suspend-to-RAM & resume-from-RAM for low latency restart of critical tasks (e.g., CAN gateway services)



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

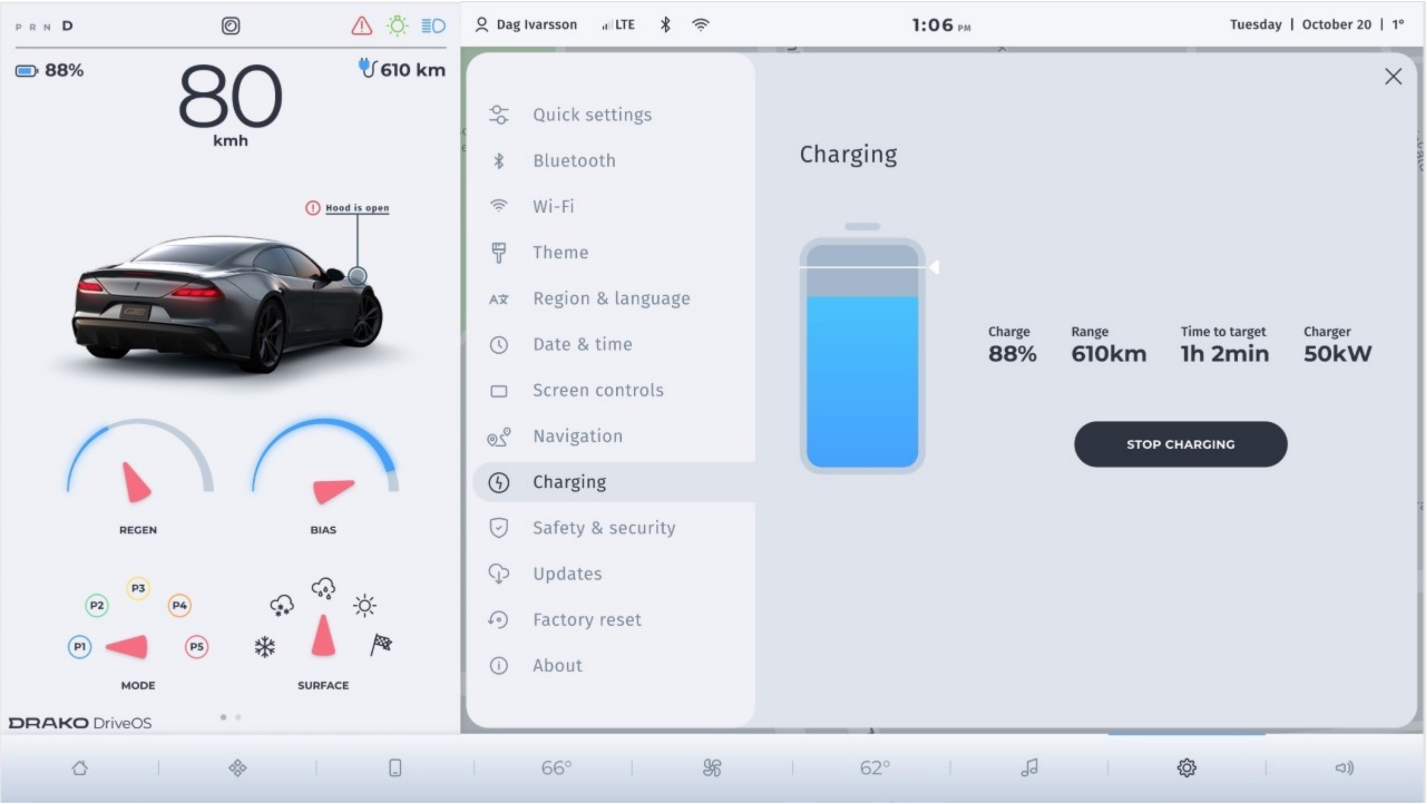




Jumpstart reduces DriveOS boot delay

Better than standalone Linux!

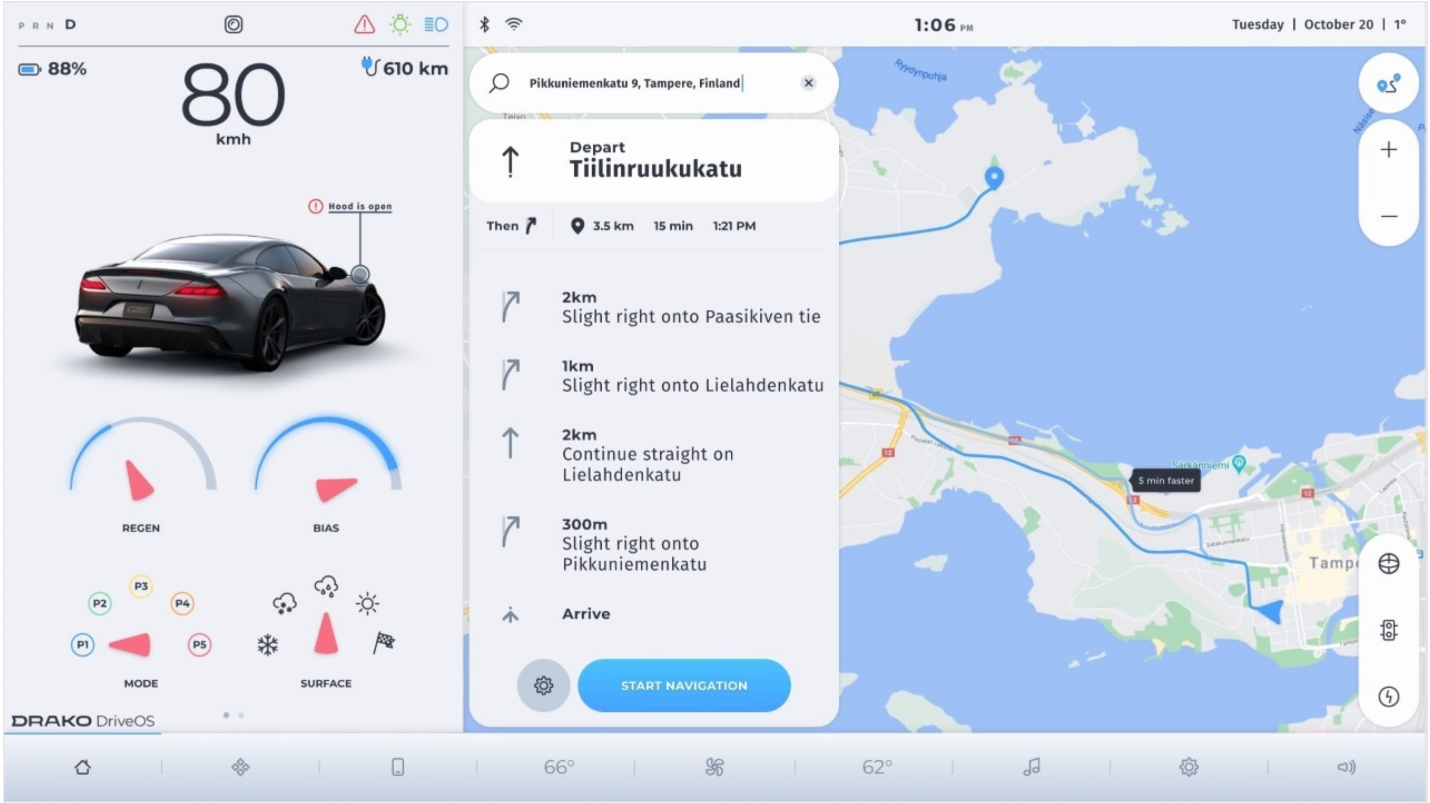
DriveOS: Screenshot 1/4



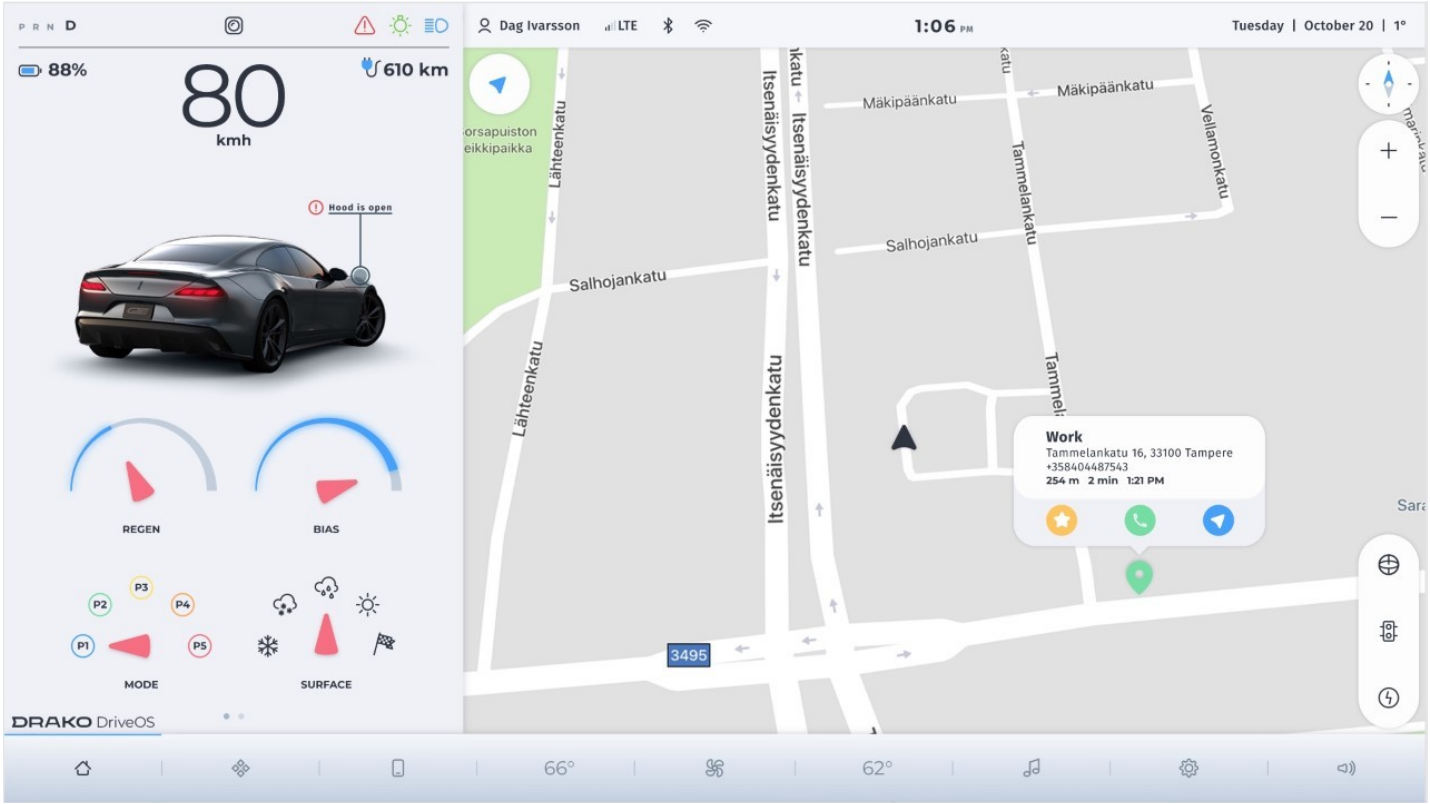
DRAKO

BOSTON UNIVERSITY

DriveOS: Screenshot 2/4



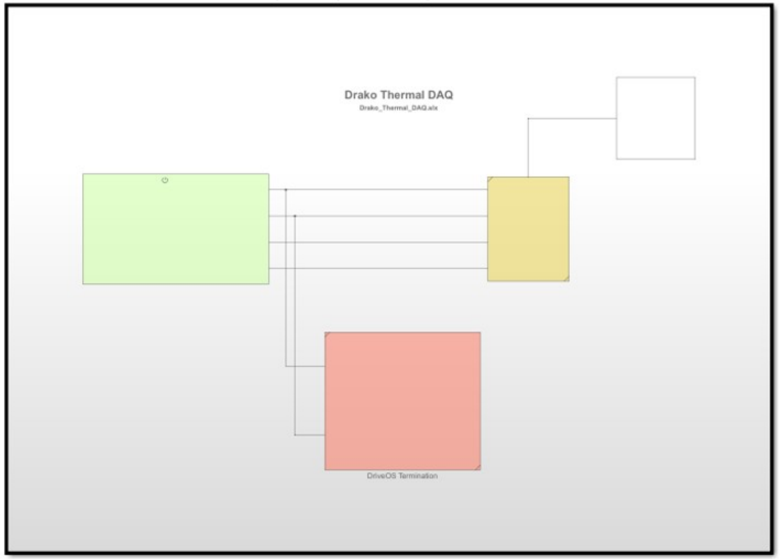
DriveOS: Screenshot 3/4



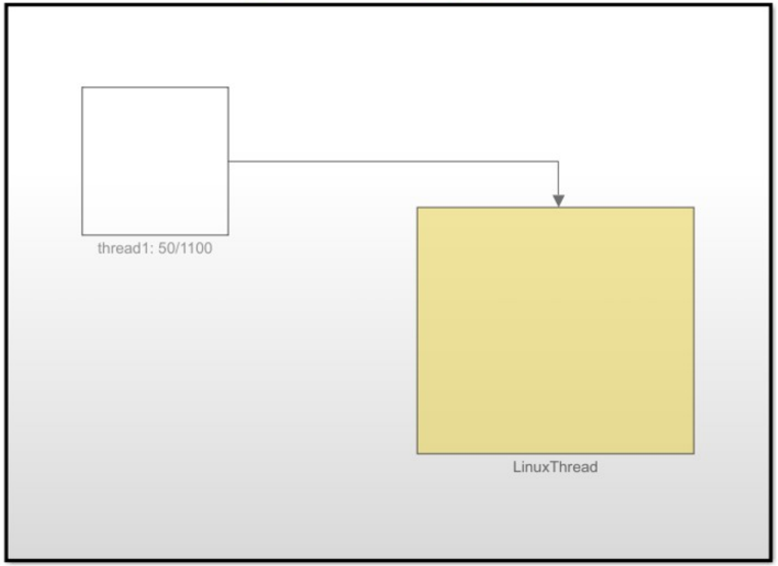
DriveOS: Screenshot 4/4

Simulink Multi-OS Modeling and Code Generation

- Model-based design for Multi-OS target
- Automatic support for nested ELF binaries with inter-sandbox RPC bindings



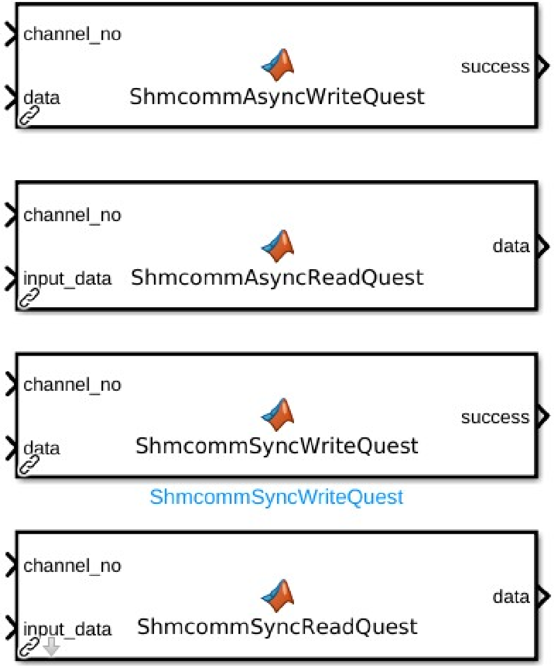
QuestSandbox



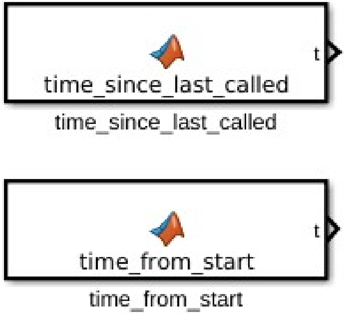
LinuxSandbox

Quest(-V) Simulink Blocks

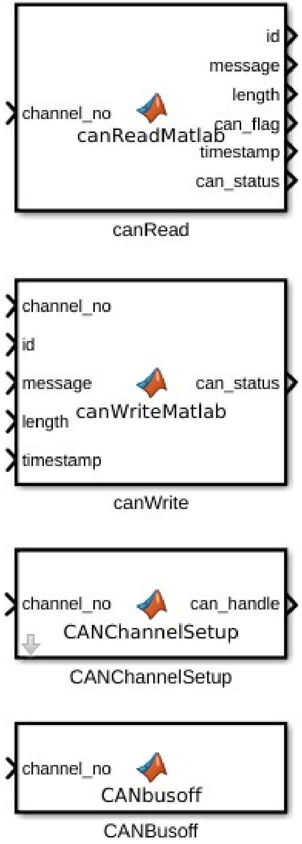
Shared memory inter-task/sandbox communication



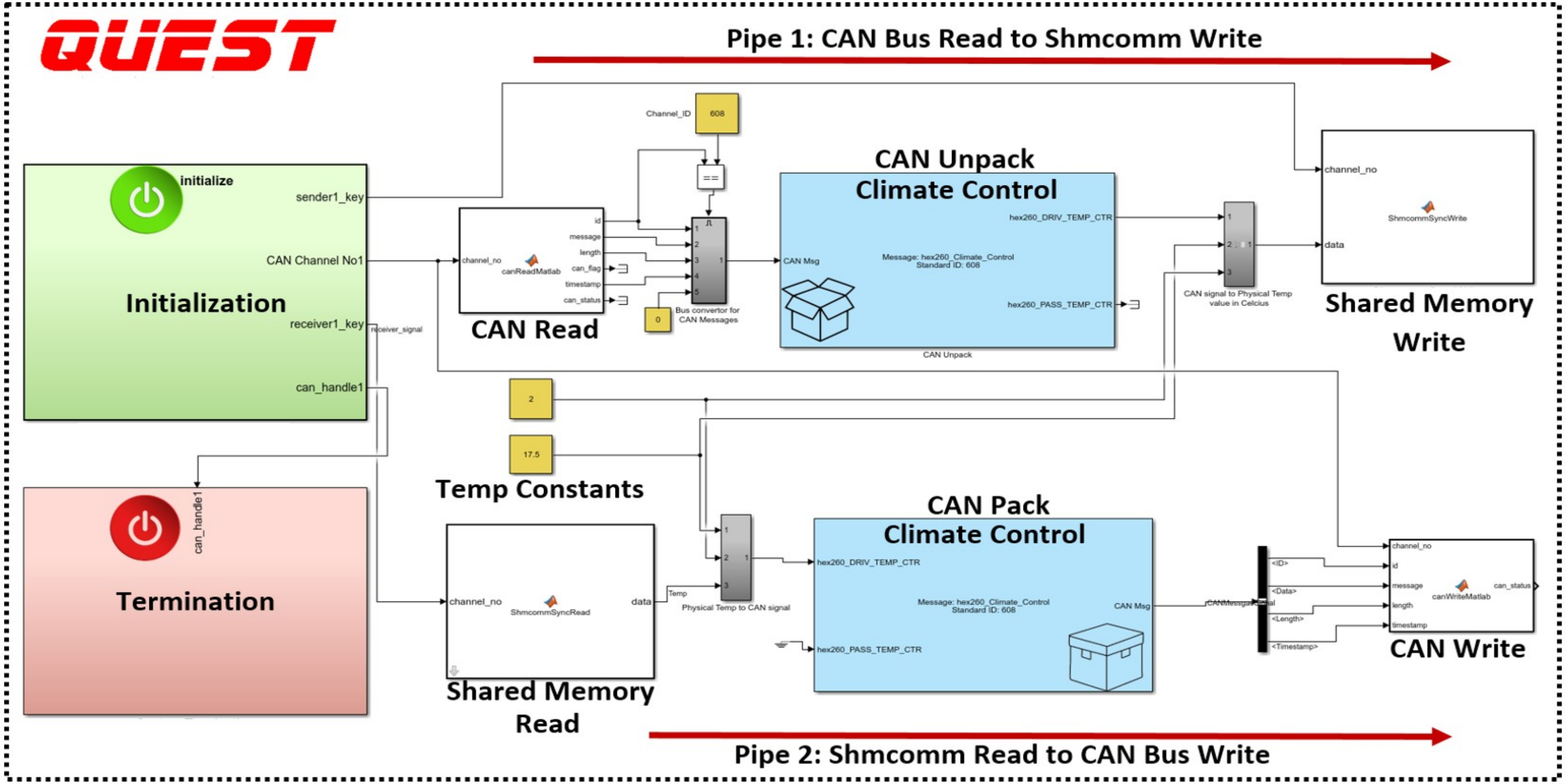
Time management



CAN-bus Management



Example: Quest HVAC CAN◄—►Shared Memory Logic



Mapping a Function to a Quest VCPU

Configurable Parameters:

1. Target Sandbox:
2. Task Budget (C)
3. Execution Period (T)

A VCPU is bound to an automotive function via the output signal link of the **vcpusetup** block

Set up new channel or connect to an existing one