SYSGO
EMBEDDING INNOVATIONS

REAL-TIME, SAFE AND CERTIFIED OS

FOR A SAFE & SECURE CONNECTED FUTURE

SYSGO

- Engineering company
- ~150 people (incl. support staff, management etc.)
- Flagship product
  - **PikeOS**
    - Real-time operating system certified for safety and security
    - Only other 4 such systems supporting certification available worldwide
- Secondary products
  - ElinOS – Embedded linux distribution maintained by Sysgo with smooth integration with PikeOS
  - PikeOS for MPU – PikeOS spin-off aimed for embedded platforms without MMU
  - CODEO – Eclipse-based IDE for Sysgo products

INTRODUCTION

- Realtime systems design patterns
  - Predictability as goal
  - Offline/integration time/fixed design
  - Simplicity
- Usual differences between Realtime vs General purpose operating systems
  - Scheduling
    - usually threads do not have quantums
    - predictable scheduling scheme
  - Resource management
    - stronger separation mechanisms
    - no malloc()/free() during regular operation of the device
    - no fork() or similar process creation API => OS image contains the apps to run
  - Features
    - lack of drivers, frequent customization for the hardware platform specifics

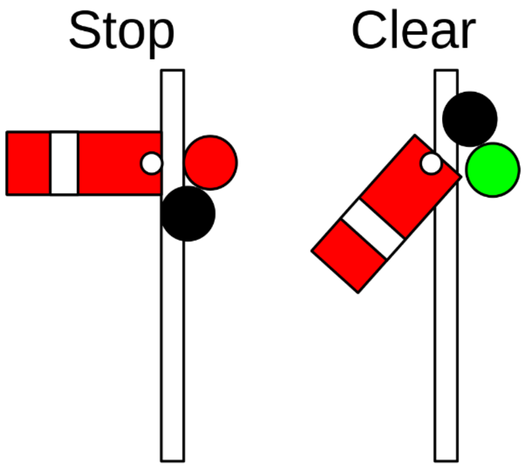
WHERE CAN I GET PIKEOS?

- Not available as consumer product (B2B only)
- Typical workflow:
  1. Customer evaluates the HW (System on Chip) and SW (the OS)
  2. We provide PikeOS either for QEMU or a SoC Development board and some training or support
  3. Customer builds a custom board for that SoC, with special peripherals
  4. We provide OS support for his custom board
  5. We provide certification documents (if necessary)
- Best for certified and mixed-criticality usage. Alternatives:
  - Linux with RT patches?
  - Lots of other RTOSes

CERTIFICATION

WHAT IS SAFETY AND SECURITY?

- System does not harm the environment
  - Czech translation overloaded: (provozní) bezpečnost
- Safety ≠ Flawless, if there is:
  - Safe backup
    - Airbus A340 rudder can still be controlled mechanically
  - Safe failure-mode
    - Stop position is safe failure mode for a rail signal
  - Or if it is harmless
    - In-flight entertainment
- Safety ≠ Security (i.e. System is resistant to cyberattacks)
  - But there are overlaps
    - Safety-critical device under control of an attacker is not safe
- Is certification perfect?
  - Boeing 787s must be restarted every 51 days
    - https://www.theregister.com/2020/04/02/boeing_787_power_cycle_51_days_stale_data



By Wiki user User:Stannered

SAFE AND SECURE SOFTWARE – MEANING?

- In Sysgo we mostly use administrative perspective on these terms
- Safety
  - We call software safe (synonymously reliable, dependent) when it has been developed, verified and certified according to the proper level of a safety standard.
  - Computations of reliability (e.g. faults per hour of service, ...) are applied very rarely.

- Security
  - We call software secure when it has been developed, verified and certified according to the proper level a of security standard and all this has been done in environment or conditions complient to that standard.

CERTIFICATION – WHY?

- Safety – In some domains safety-critical software cannot be put into service without being certified on proper level for the proper certification standard
  - And of course safety certification contributes to safety to certain extent

- Security – Decision makers holding responsibility may appreciate a solid argument why they did their best for security if bad things happen and serious disputes may occur
  - And of course security certification contributes to security to certain extent

SAFETY AND SECURITY CERTIFICATION

SYSGO
EMBEDDING INNOVATIONS

We provide Certification Kits for PikeOS for a wide range of industry domains and up to the highest levels

Safety: ECSS-E-40 - Space "Software Engineering"	
Safety: ISO 26262 - Automotive "Road vehicles - Functional Safety"	ASIL - Automotive Safety Integrity Level D C B A
Safety: DO-178C - Avionics „Software Considerations in Airborne Systems and Equipment Certification“	DAL - Design Assurance Level D C B A
Safety: EN 50128/29 - Railway "Software for train control and management systems"	SIL - Safety Integrity Level 1 2 3 4
Safety: IEC 61508 - Industry "Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems"	SIL - Safety Integrity Level 1 2 3 4
Security: SAR - Avionics „Airbus Security Standard“	SAL - Security Assurance Level 1 2 3 4
Security: ISO/IEC 15408-1/2/3 – Industry "Common Criteria for Information Technology Security Evaluation"	Evaluation Assurance Level 1 2 3 4 5 6 7

DO178 SAFETY LEVELS

EMBEDDING INNOVATIONS

- Design Assurance levels (DAL) from A to E

DAL	Failure condition	Consequences	Failure Rate
A	**Catastrophic**	May cause an airplane crash	10^-9/h
B	**Hazardous**	May cause fatal injuries	10^-7/h
C	**Major**	May cause injuries	10^-5/h
D	**Minor**	May cause inconvenience	10^-3/h
E	**No Effect**	No impact on safety	

SAFETY CERTIFICATION

- Software development
  - Requirement documents, Software architecture (a lot of documentation and paperwork)
  - Development processes (how to commit, peer review and testing, …)
  - Traceability, annotations
  - Coding standards
- Verification
  - Requirement-based testing (~80% of the verification efforts)
  - Analysis
    - Stack analysis, Partitioning analysis, Timing analysis
  - Formal reviews (a lot of paperwork)
    - Documents, System under, Tests
  - Independence on development (verification engineer cannot commit into the verified code)
- Process description, plans and other paperwork

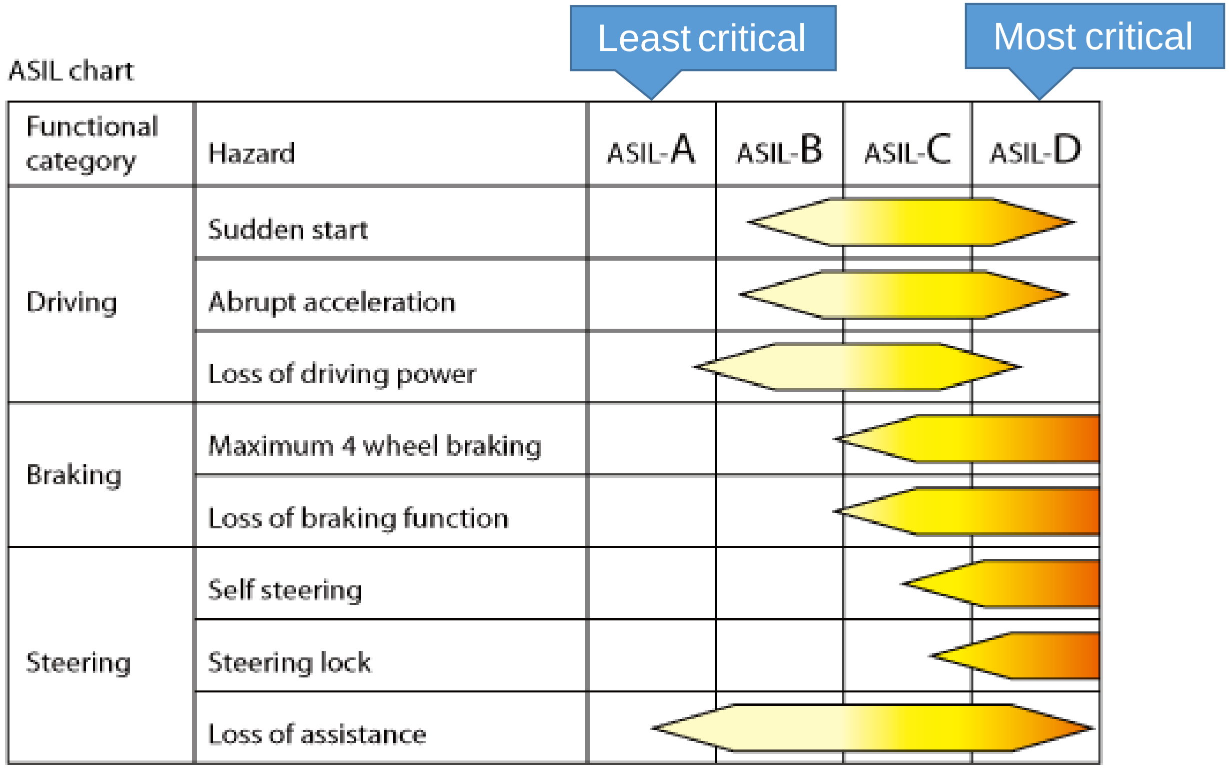
WHY CERTIFIED HYPERVISING KERNEL?

SYSGO EMBEDDING INNOVATIONS

- Separate critical and non-critical components
  - MMU/MPU required
- We need to certify
  - The critical components
  - The kernel
  - Smaller kernel = less work
- Non-critical parts can use
  - Off-the-shelf software
  - Linux
  - => Easier development and lowered certification costs

MIXED CRITICALITY EXAMPLE

- Typical examples of mixed criticality:
  - Control loop (critical) vs. diagnostics (non-critical)
  - Combined Control Unit for multiple functions in car



https://www.jnovel.co.jp/en/service/compiler/iso26262.html

PARTITIONING EXAMPLE: AIRBUS A400M

SYSGO
EMBEDDING INNOVATIONS

Level B	Level B	Level C	Level D
Ramp, Doors, Aerial Delivery, Cargo Locks ...	Graphics OpenGL GUI HMI	Winches, Crane ...	9 Applications incl. Waste & Water

PikeOS Virtualization Platform

Hardware

Pictures: Rheimetall Defense A400M

VERIFICATION EXAMPLE

- Testing of ANIS
  - ANIS = UDP/IP network stack certified for DO178C - DAL C (safety)
  - ANIS has 80 000 LOC of C code
  - 755 low-level design requirements, 587 interface requirements, 75 high-level requirements

- ANIS verification (tests only)
  - 2 test suites: Low-level test suite and Integration test suite
  - 694 low-level test cases, 25 integration test cases
  - Test suites have 125 000 LOC of C code
  - > 1000 pages of test suite description
  - ~ 5000 man-hours of verification effort
  - One test case 1-3 man-hours in simplest cases; man-weeks in most complex cases

EXAMPLES OF HIGH-LEVEL REQUIREMENTS

- The Ethernet driver shall forward and separate traffic between up to 3 physical ports (VLANs).
- A resource partition shall have a statically configurable set of memory requirements which specify physical memory, memory mapped I/O and port mapped I/O regions assigned to the partition.
- PikeOS shall mask an interrupt source if no thread is registered as handler for this interrupt.

EXAMPLES OF INTERFACE REQUIREMENTS

- vm_write() shall write an Ethernet message from the buffer "buff" to the device and return the number of bytes written in "written_size" and return P4_E_OK.

- The driver shall use interrupt specified by "Int" property.

- The driver shall raise a HM error of type P4_HM_TYPE_P4_E if the GEM hardware has unsupported version.

EXAMPLES OF LOW-LEVEL REQUIREMENTS

SYSGO
EMBEDDING INNOVATIONS

- anisUDP_checkChksum() shall return ANIS_ERR_OK if the computed checksum matches the value in the header.

- anisUDP_send() shall copy the message payload into the allocated buffer objects, prefixing the message with the UDP header and leaving sufficient space to prefix the IP header.

- anisIGMP_sendLeave() returns ANIS_ERR_SPACE if there is no internal buffer to store the message to send.

PIKEOS TECHNICAL OVERVIEW

HISTORY AND „PREDECESSOR“ - L4/X86

- Research Micro-Kernel in mid 1990
- http://os.inf.tu-dresden.de/L4/l4doc.html
- Focus on small API (7 syscalls, slightly overloaded)
  - Recent x86 Linux ~350 syscalls, PikeOS ~110
  - IPC, thread and task management
  - No mutexes, file descriptors, IPC used for everything
- Fast IPC for communication & configuration
  - IPC can send data
  - IPC can also map pages
- Hierarchical Tasks
  - The root task has access all the memory and distributes it to children
  - Tasks can directly IPC only to parents or siblings

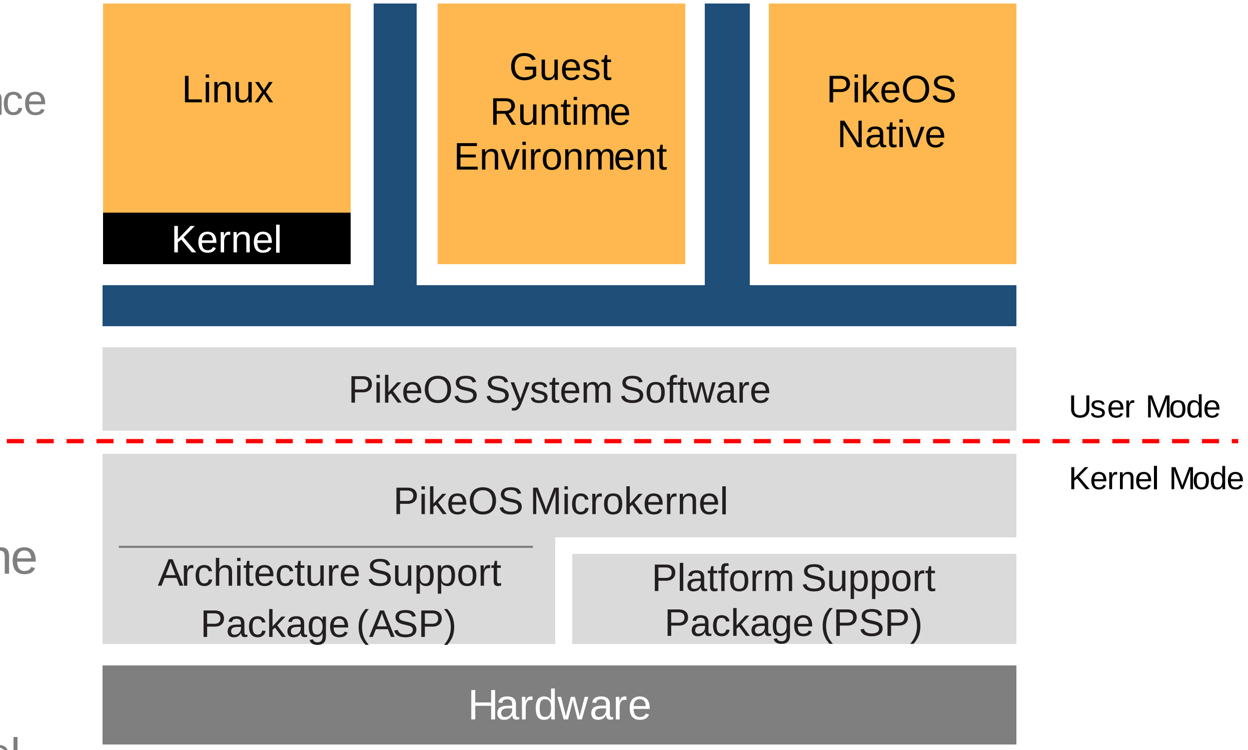
PIKEOS

SYSGO EMBEDDING INNOVATIONS

- Microkernel
  - Inspired by L4
    - https://www.researchgate.net/publication/285592141_Evolution_of_the_PikeOS_Microkernel
  - Lot of stuff added since then
    - Performance → larger kernel
    - Business requirements from customers
- Memory protection (MMU) required
- Includes virtualization hypervisor
- X86, ARM, SPARC, PowerPC, RISC-V
- Eclipse IDE for development and configuration

PIKEOS ARCHITECTURE

- Microkernel (may no longer be true)
  - Limited number of system calls
  - Only the kernel itself runs in protected mode (since PikeOS 4.2 not really)
- Userspace is split into „partitions“
  - Each partitions holds an application or even an operating system
- It is possible to put driver into every layer of the system
  - Most drivers are standalone user application
    - Thus, their fault will not threaten the kernel
  - Some drivers may be compiled into kernel
    - This may have improved performance



GUEST OS

- General
  - POSIX
  - Linux
    - Hardware virtualization
    - Para-virtualization
- Domain specific
  - ARINC653
  - PikeOS native
- Other semi-supported
  - Ada, RT JAVA, AUTOSAR, ITRON, RTEMS

HARD REAL-TIME

- System must meet deadlines
  - Missed deadline can affect safety
- Deadlines given by
  - Physics
    - Car must start breaking immediately
  - Hardware
    - Serial port buffer size – data loss
  - System design
- HW and SW must cooperate
- Apollo 11 had problems due to „irq storm“ from faulty radar
  - Src: https://www.doneyles.com/LM/Tales.html

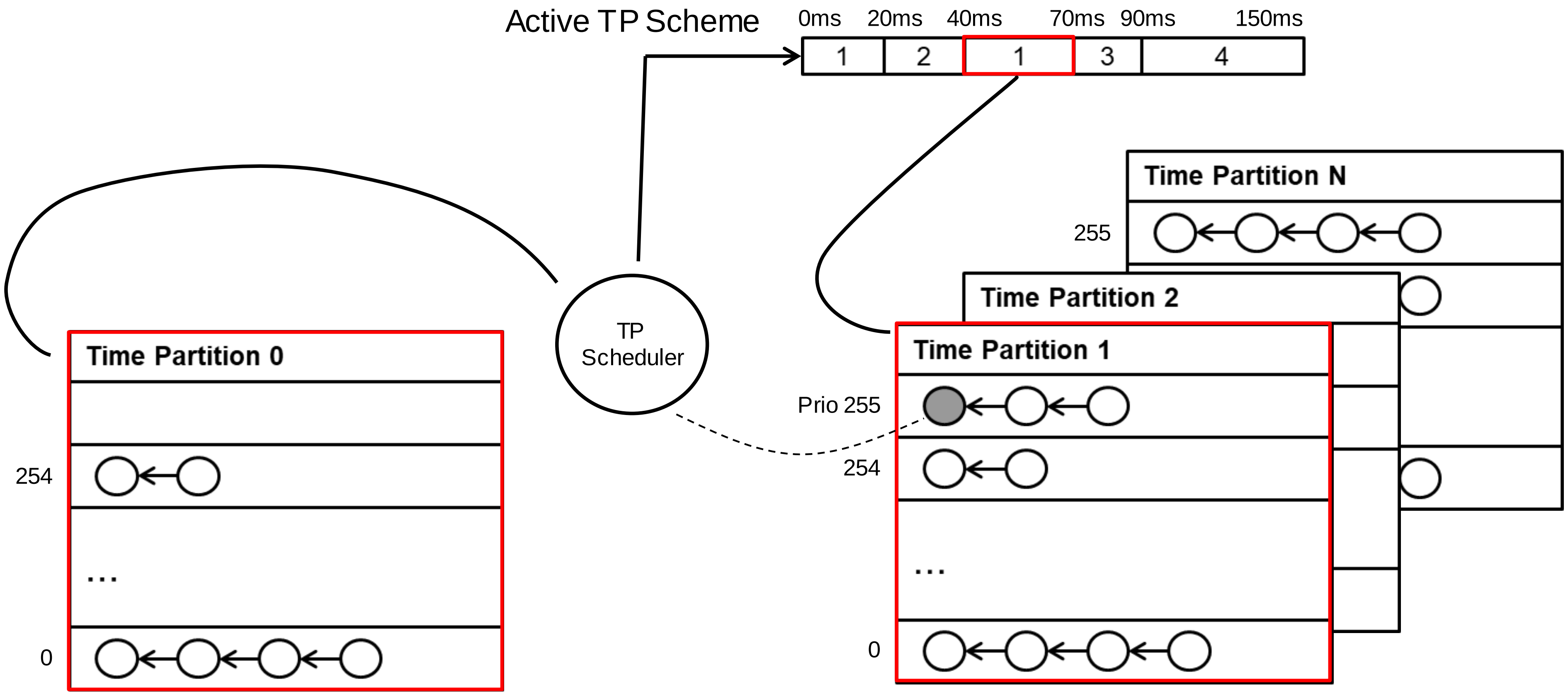
REAL-TIME SCHEDULING

- Lot of theory about running the tasks in correct order
  - MFF UK, NSWE001 - Embedded and Real Time Systems
    - Earlist deadline first scheduling, Rate monotonic scheduling
- In practice simple thread priorities
  - QNX, FreeRTOS, PikeOS, VxWorks …
  - + Some extensions
- Often without classical time quantum
  - Unlike Linux
- On Linux-RT, use pthread_attr_setschedpolicy
  - SCHED_FIFO, SCHED_RR, SCHED_DEADLINE
  - Documentation: https://bit.ly/3yY0GeP
  - API part of POSIX, 1003.13, PSE51, PSE52

PIKEOS SCHEDULING

SYSGO EMBEDDING INNOVATIONS

Time partitions + priorities



TP0 is PikeOS extension

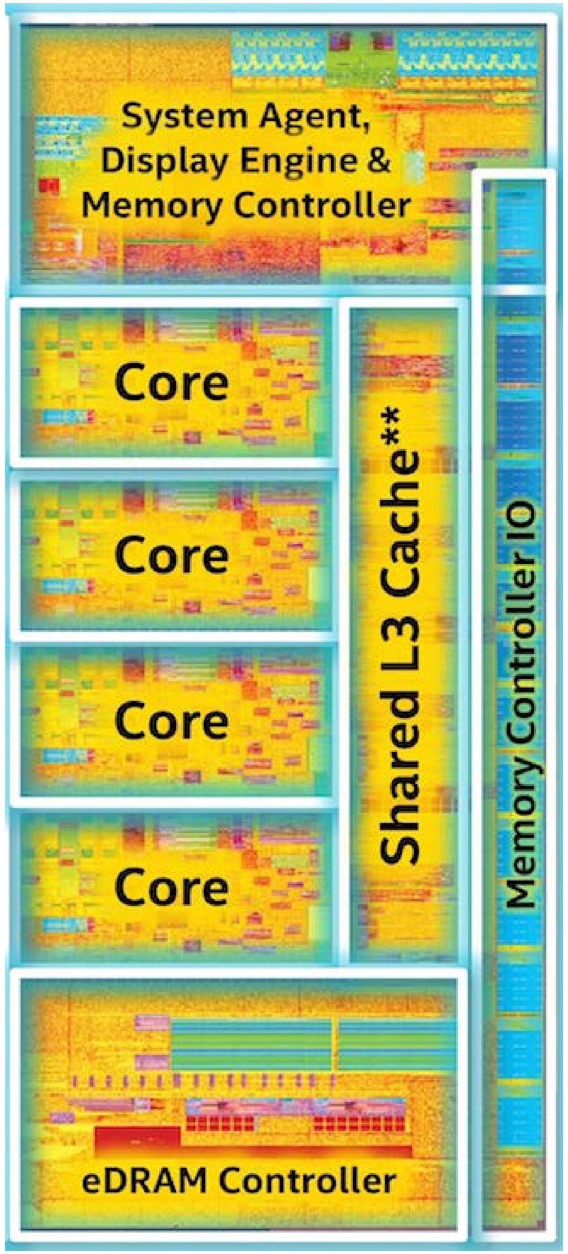
WCET

= Worst-Case Execution Time

- How long will the code run?
  - Will we satisfy the deadline?
  - Upper bound (worst-case) is important
- Combination of code analysis and measurement
  - PikeOS API function with expected use in real-time scenarios
- Jitter
  - Time partition switch
- Tools e.g.: https://www.rapitasystems.com/wcet-tools#rt

ENEMIES OF REAL-TIME

- Shared resources
  - Heap, devices, scheduler, CPU time
  - Unpredictable state
  - Locking
- Multi-processor
  - Locking less predictable
  - Shared
    - Cache
    - Memory bandwidth
    - Other processor units?
- Devices
  - Interrupts



Broadwell die map, copyright Intel

MORE ENEMIES

- Modern hardware
  - Lazy algorithms
  - Branch predictors
  - Out-of-order execution
    - Unpredictable pipeline
  - TLB, caches
  - SMI, ARM Trust Zone etc.
- Modern OS features
  - Swap, overcommit
  - Copy on Write
  - Thread migration
- Complexity in general

MEMORY MANAGEMENT IN RTOS

- Sometimes no MMU at all
  - FreeRTOS, some VxWorks variants
  - Or just MPU – memory protection units
    - Memory regions without paging
- PikeOS: Simple virtual to physical mapping
  - Mmap-like syscalls directly fill in page tables, no unmap
  - X Swap, memory mapped files, copy on write …
  - ✓ Shared memory
  - ✓ Memory protection (NX bit etc.)
- Compared with Linux… (correct me if wrong)
  - Mmap-like syscalls prepare *struct vm_area,* page tables on-demand
  - Each physical page has a descriptor to track refcounts and other state

PIKEOS KERNEL MEMORY

- User-space needs kernel memory for:
  - Threads (kernel stacks)
  - Processes
  - Memory mappings
- Pre-allocated pools
  - Safe limit
  - Avoids extra locks

USER-SPACE MEMORY ALLOCATION

- Heap allocator problems
  - Locking
  - Allocator latency
  - Fragmentation
  - Unpredictable failures
- General rule: Avoid malloc/free
  - Except for initialization
  - Pre-allocate everything
  - Malloc/free is error prone anyway
- Or use task-specific allocator

MULTI-PROCESSOR

- Threads are bound to single CPU
  - No automatic balancing of tasks
  - PikeOS has implicit migration on IPC
  - Scheduler ready queues per-CPU
- Kernel should avoid locks
- Especially in real-time syscalls
- If locks are fair (FIFO queue), WCET is
  - *num_cpus * lock_held_time*

MULTI-PROCESSOR

- Predicting resources like caches and memory is difficult
- Disable HyperThreading
  - it is not worth the trouble
- SYSGO’s recommendation “avoid the problem”
- Better solutions are being investigated

CPU 2	Non-real-time APP1	Idle	Non-real-time APP3
CPU 1	Linux	Real-time APP	Non-real-time APP2

OTHER CONSIDERATIONS

- Worst-case complexity
  - Hash-map is O(1) in practice, O(n) in worst case
  - AVL or RB trees are always O(log n)
- Log messages may slow you down
- Keep the code small (certification)
  - Sadly, it often is better to copy and specialize the code
- General guiding principle:
  Configure/initialize most things statically
  - Static number of FDs, buffers etc.

OTHER CONSIDERATIONS

- Control over the platform
  - You are not alone on X86
    - System Management Mode
    - Intel Management Engine
    - AMD Platform Security Processor

INTERRUPT HANDLING

- Interrupt handling sequence:
  1. HW signals interrupt
  2. CPU runs kernel’s interrupt handler
  3. Kernel masks (disables) the interrupt
  4. Unblocks the thread blocked in *wait_for_interrupt*
  5. Thread handles interrupt
  6. Calls wait_for_interrupt
  7. Kernel blocks the thread
  8. Unmasks the interrupt

  \+ variations for different platforms

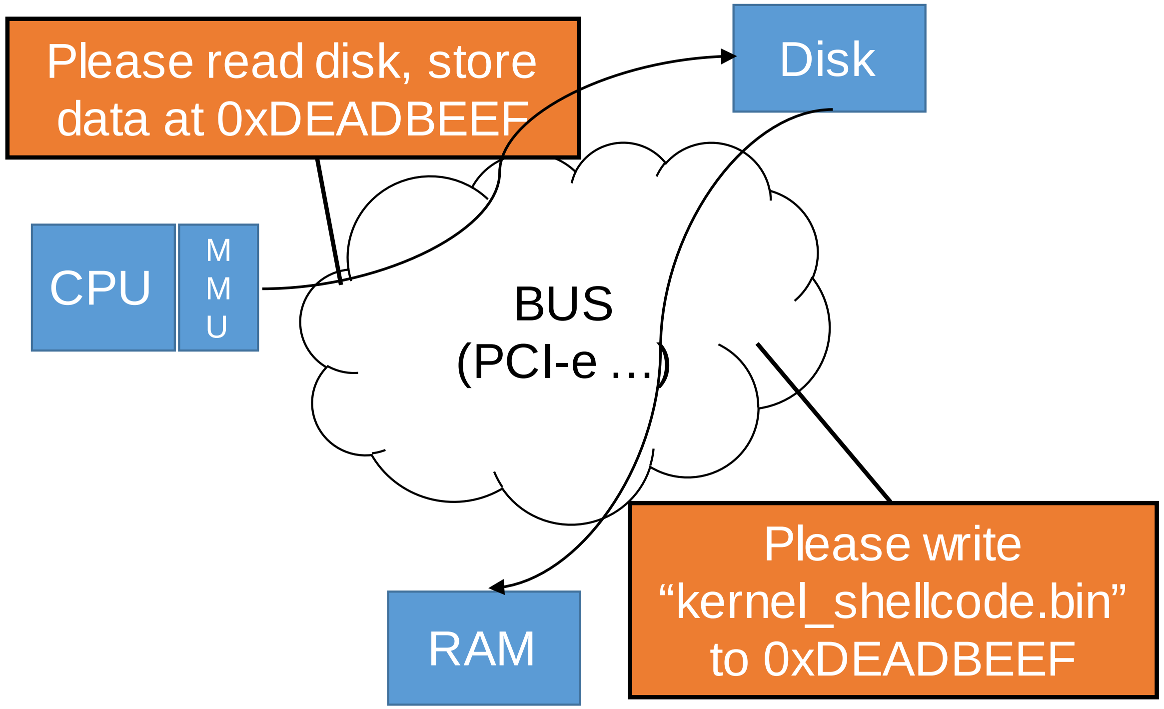
USER-SPACE DRIVERS

- Modern hardware looks like a memory (MMIO)
- Can be mapped to user-space using MMU
  – PikeOS is configured to map the IO memory into the driver’s partition address space
- Most drivers use file API as interface with its client application
  – open(“eth0:0”, O_WR_RD, &fd); // open the Ethernet driver device
  – read(fd, ethernet_frame_buf, 1536); // receive a frame from Ethernet network
  – write(fd, my_frame_buf, 100); // send a frame to Ethernet network
  – ioctl(fd, NET_IOCTL_GET_LINK_STATUS, &status); // check if the network link is up or down
- PikeOS interrupt handler is a user-space thread
  – with regular scheduling

```
for(;;) {
  wait_for_interrupt();
  /* handle the interrupt */
}
```

IOMMU

- Q: Is MMU enough to isolate drivers?
- A: No, because of DMA
- The driver can tell device to read/write memory
  - Bypasses CPU MMU
- We can
  - Ignore the problem
  - Disable DMA
  - Use IOMMU



IOMMU

- IOMMU is MMU for the Non-CPU Bus Masters
- Available on modern x86, ARM and PowerPC
  - Different hardware, same goal

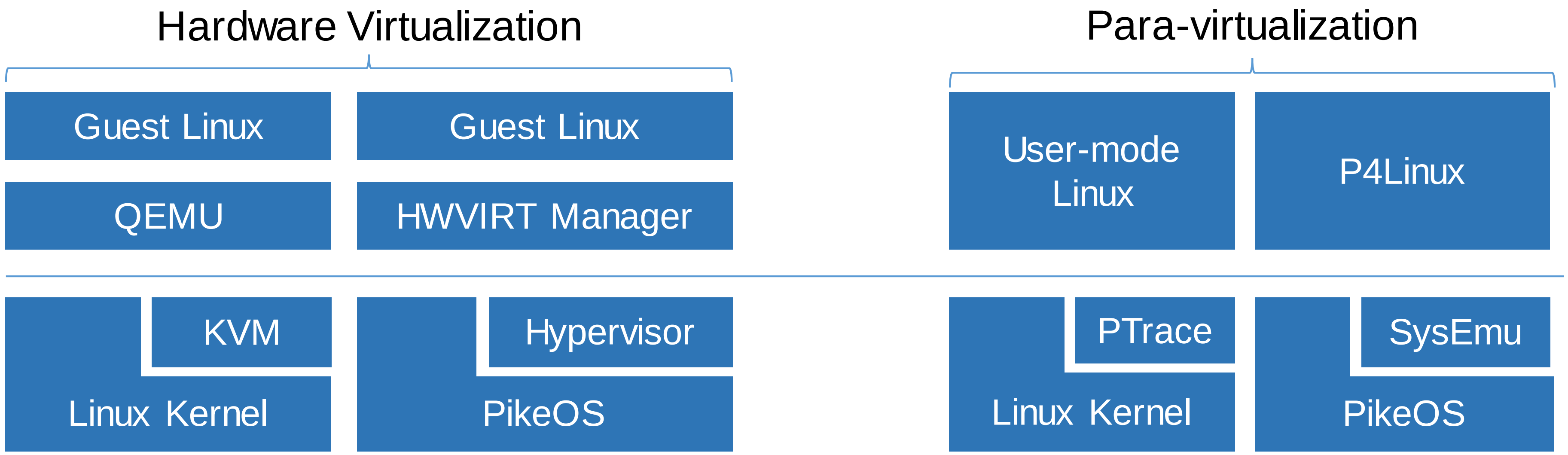
WHY VIRTUALIZATION?

- To use Linux
  - … and Linux device drivers
  - Safely
- Offered by
  - SYSGO
  - GreenHills
  - VxWorks …
- Minimal hypervisor part of the kernel
- VMs subject to access rights
  - … and scheduling

VIRTUALIZATION COMPARISON

SYSGO EMBEDDING INNOVATIONS

- PikeOS offers
  - Para-virtualization (similar to User-mode Linux)
  - HW Assisted virtualization



P4LINUX

- Linux kernel as a PikeOS process
- Runs unmodified Linux executables
- Inspired by User Mode Linux
- Virtual CPUs backed by PikeOS threads
- Linux processes backed by PikeOS processes
- *sysemu_enter* syscall to “run the userspace”
  - Use address space of other PikeOS process
  - Start executing code in this context
  - Returns control on exceptions, privileged instructions etc.
    - Also returns to the old address space

P4LINUX

- Full Linux memory management
  - Paging, CoW, memory mapped files …
  - Page tables simulated by PikeOS processes
- Linux kernel not mapped in user-space at all
  - Now pretty standard with Spectre & Meltdown mitigations
- Para-virtual drivers for PikeOS devices
- Code to access passed-through devices
  - Most drivers are well behaved and use proper APIs to map device memory and handle interrupts
  - => can be used unchanged
  - E.g. You can play OpenArena on an Intel GPU

OTHER PIKEOS FEATURES

- Interpartition communication
  - Shared memory
  - Queuing ports, Sampling ports
- Synchronization primitives
  - Mutexes
  - Condition variables
  - Barriers
  - Semaphores
- Volume providers
  - CFS (Certifiable filesystem)
- Integration-time xml configuration
  - PikeOS, drivers and optionally applications have build-time xml configuration
  - Integrated with CODEO for pleasant user experience

PIKEOS AS SECURING COMPONENT

SECURITY AND REAL-TIME SYSTEMS

- Connecting embedded devices to internet (internet of things)
  - Increasing trend in the last decade
  - Somewhat limited know-how about how to secure embedded software among device manufacturers
- Connecting safety-critical software to internet extends the possibility to disable the device by a third-party
- How much is this real today?
  - Jeep Cherokee, 2015, documented a possibility of disabling brakes over Internet (cellular phone connection)
  - http://illmatics.com/Remote%20Car%20Hacking.pdf

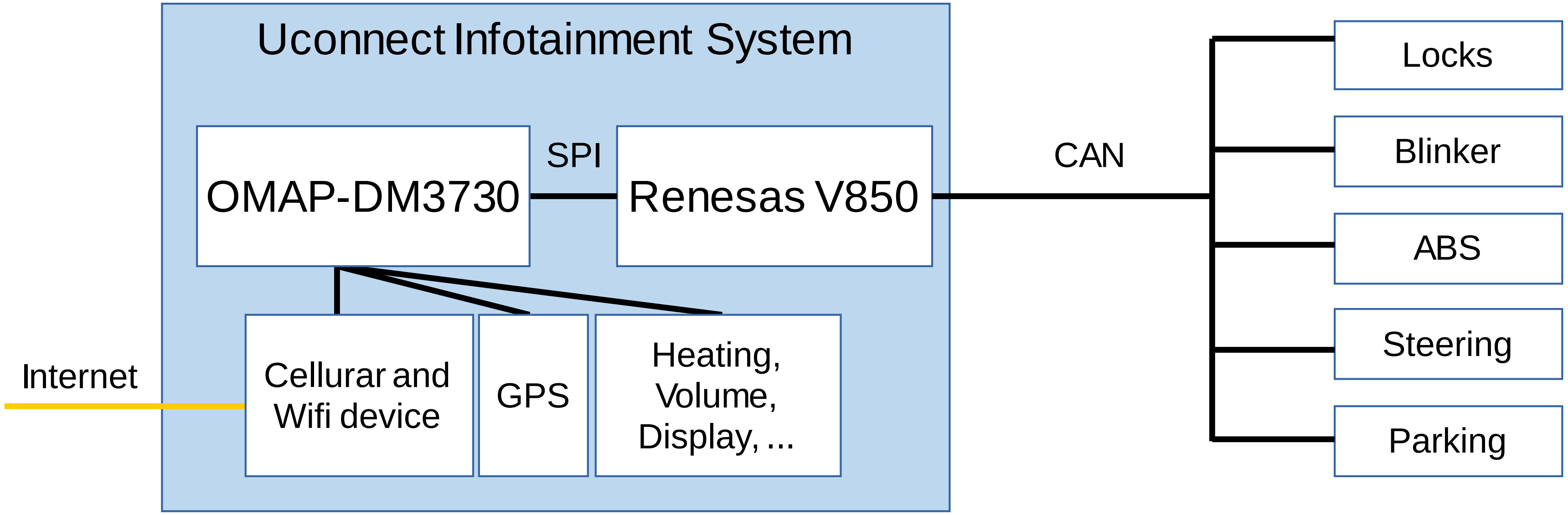
CERTIFICATION / SECURITY

SYSGO EMBEDDING INNOVATIONS

- Common Criteria, Security Target
- Trusted world (kernel, PSP, some partitions)
- Untrusted world (partitions with low security demands (e.g. Linux))
- Well-defined interface between the two worlds
  - Attack surface syscalls to kernel, ioctl and other communication channels between the trusted and untrusted world
  - Verification approach
    - Some safety requirements marked as security relevant, these are then tested more extensively or just differently
    - Vulnerability analysis instead of some safety-related analyses
- Security board monitors reported vulnerabilities for other operating systems
- Fuzz tests
- Increased demands for physical security

PIKEOS SECURITY USE CASE - CAR INFOTAINMENT UNIT

SYSGO EMBEDDING INNOVATIONS

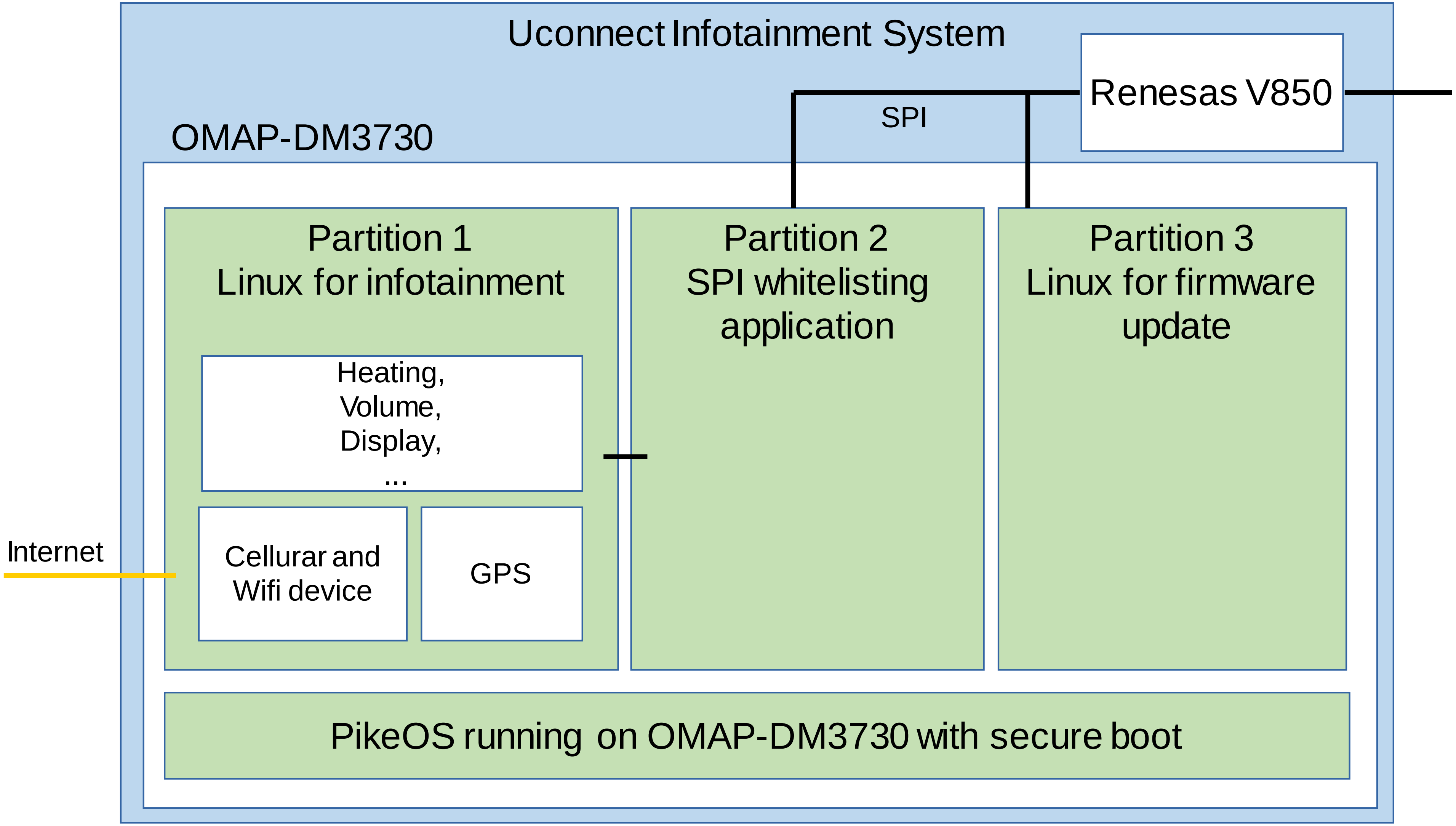


OMAP-DM3730 is controlled by embedded Linux that manages:

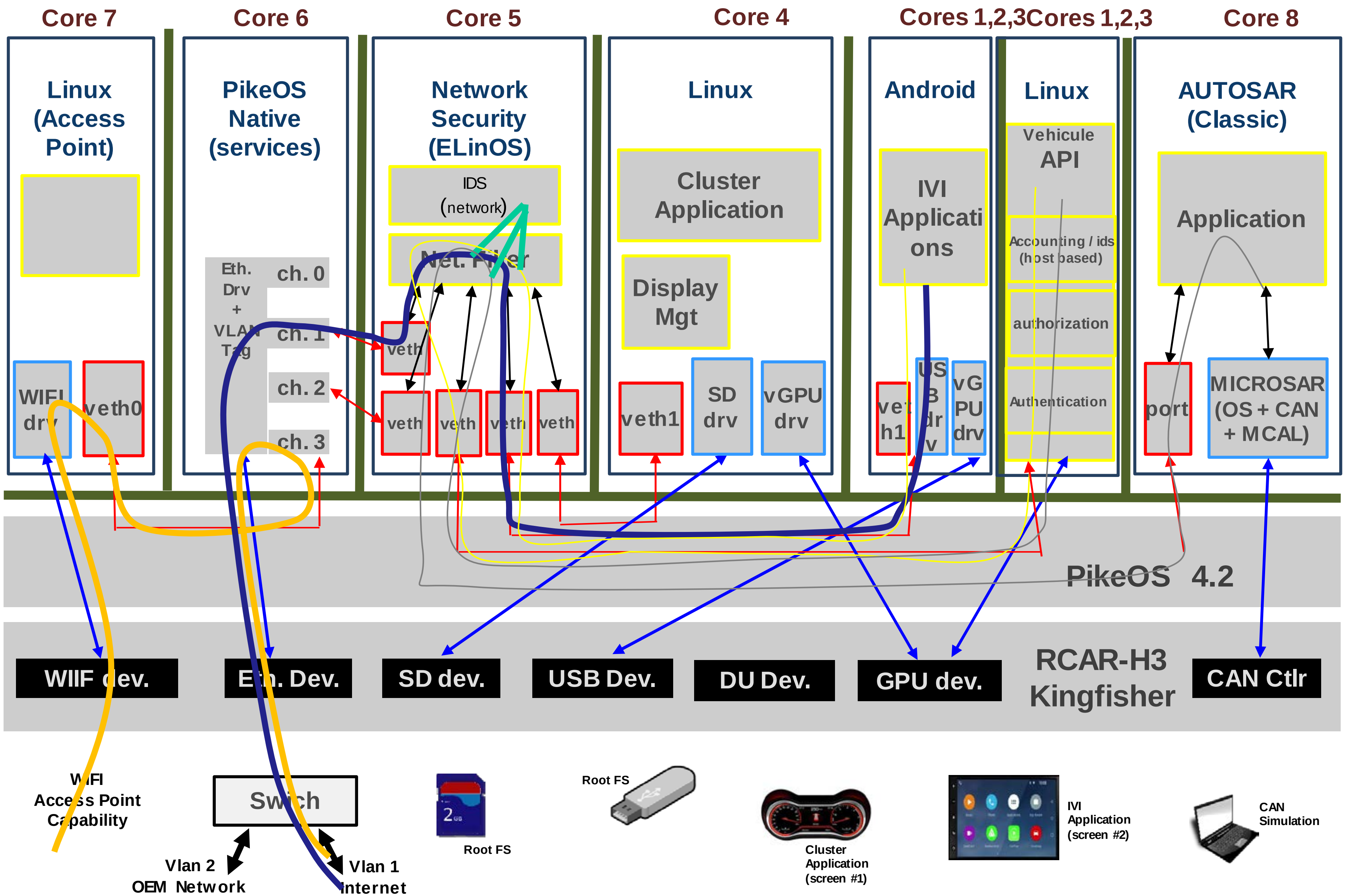
- Infotainment devices
- Internet access
- Renesas V850 local firmware update (update-v850-firmware.sh)

-> once hacked the hacker has direct access to Renesas V850 and consequently the CAN bus

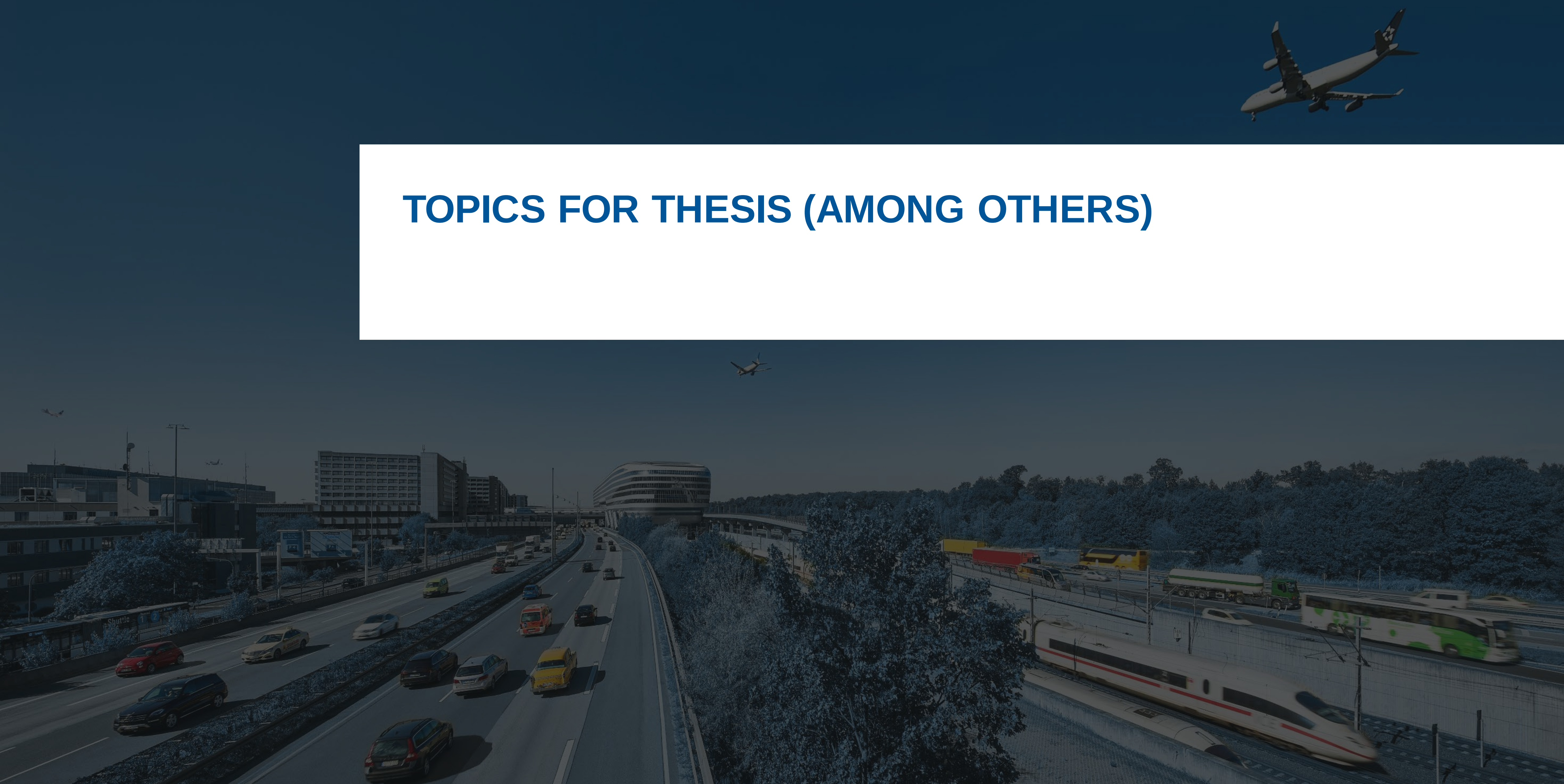
HARDENING WITH HYPERVISOR AND PARTITIONING



ANOTHER HARDENING EXAMPLE



TOPICS FOR THESIS (AMONG OTHERS)



POSSIBLE TOPICS

- **IAT0131** Modify the HWVIRT to allow a more modular approach to VMM-drivers and P4BUS-drivers.
- **IAT0132** Support Intel Processor Trace (PT) in PikeOS.
- **IAT0137** Pluggable Scheduling Policies in the PikeOS kernel.
- **IAT0142** Implement RDMA / RoCEv2 Support for PikeOS
- **IAT0143** Power Management in PikeOS (suspend/resume)
- **IAT0144** VirtIO Interface for PikeOS HWVIRT
- **IAT0145** Precision Time Protocol for PikeOS
- **IAT0147** Implement fuzz testing for certified network stack (ANIS)

QUESTIONS OR COMMENTS?

SYSGO GmbH

Am Pfaffenstein 8
55270 Klein-Winternheim
Germany

Student Contact (CZ)

tomas.martinec@sysgo.com

Subscribe, Like and Follow:

www.sysgo.com

www.sysgo.com/twitter

www.sysgo.com/linkedin

www.sysgo.com/youtube