Real-Time, Safe and Certified OS

SYSGO
EMBEDDING INNOVATIONS

Roman Kapl <rka@sysgo.com>
drivers, customer projects, development

Tomas Martinec <rka@sysgo.com>
testing and certification

Introduction

- **PikeOS – real-time, safety certified OS**
- **Desktop and Server vs.**
  - Embedded
  - Real-Time
  - Safety-Critical
  - Certified
- **Differences**
  - Scheduling
  - Resource management
  - Features
  - Development

Certification

- **Testing**
- **Analysis**
- **Lot of time**
- **Even more paper**
- **Required for safety-critical systems**
  - Trains
  - Airplanes

PikeOS

- Embedded, real-time, certified OS
- ~150 people (not just engineers)
- Rail
- Avionics
- Space
- This presentation is not about PikeOS specifically

PikeOS technical

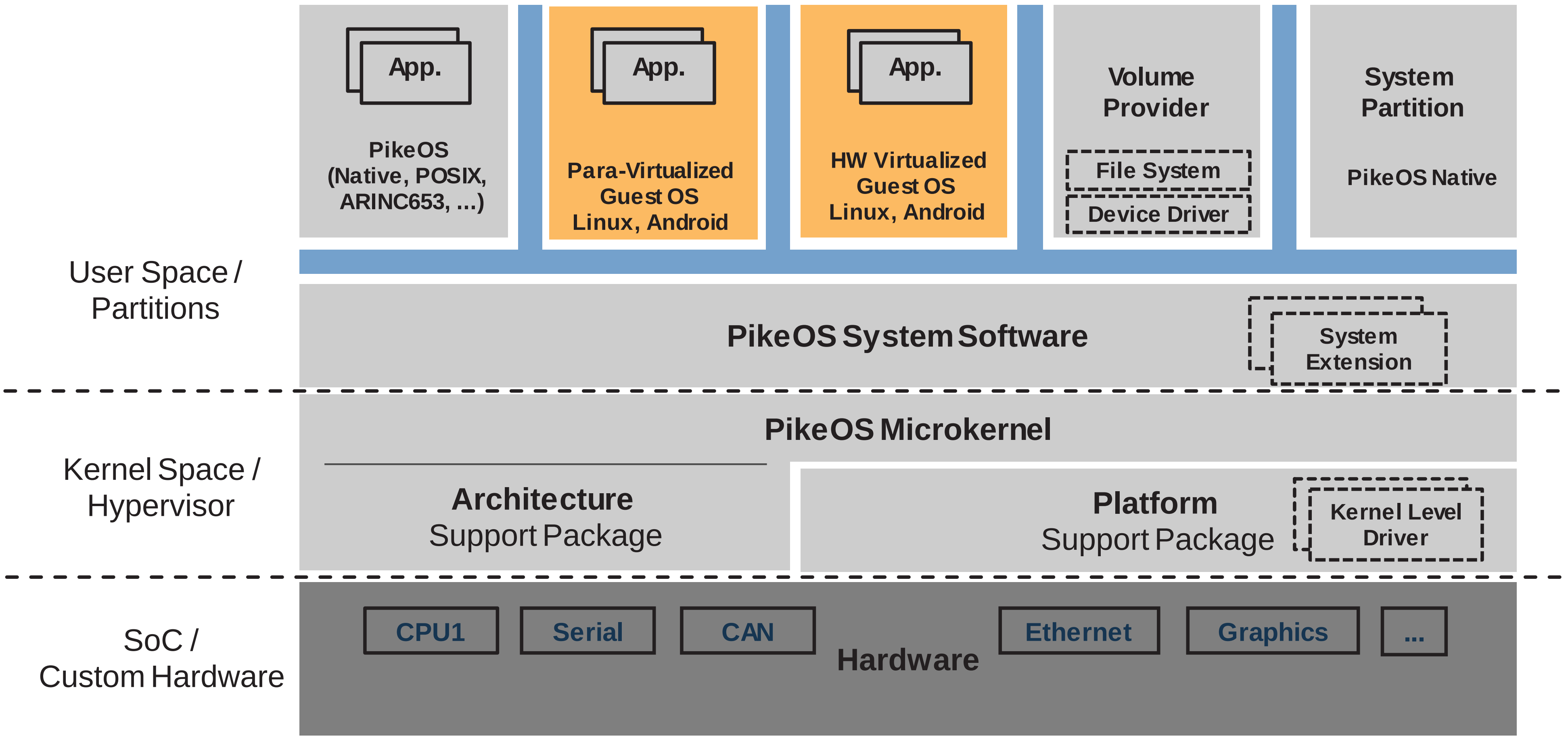
- **Microkernel**
  - Inspired by L4
- **Memory protection (MMU)**
  - More complex than FreeRTOS ☺
- **Virtualization hypervisor**
- **X86, ARM, SPARC, PowerPC**
- **Eclipse IDE for development**

SYSGO EMBEDDING INNOVATIONS

Personalities

- **General**
  - POSIX
  - Linux
- **Domain specific**
  - ARINC653
  - PikeOS native
- **Other**
  - Ada, RT JAVA, AUTOSAR, ITRON, RTEMS

PikeOS Architecture



Embedded

- **Examples**
  - Tamagochi
  - Rail signal
  - ABS brake controller
- **Usually does not have**
  - Lots of RAM
  - Beefy CPU
  - Keyboard and mouse
  - PC Case
  - Monitor

Embedded peripherals

- **Ethernet**
  - Sometimes with hardened connectors
  - May be real-time
- **CAN**
- **I2C**
- **UART (Serial port)**
- **JTAG for debugging**

Safety

- **System does not harm the environment**
  - Safe aircraft does not harm or kill people during the flight
- **≠ flawless**
  - Safe backup
    - Airbus A340 rudder can still be controlled mechanically
  - Safe failure-mode
    - “Closed” rail signal is safe
  - Harmless
    - In-flight entertainment

Safety

- **≠ security**
  - but there are overlaps
- **Safety needs to be certified**
- **<u>More</u> important than features or performance**

Hard-realtime

- **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**

Real-Time Scheduling

- **Lot of theory about running the tasks in correct order**
  - NSWE001 - Embedded and Real Time Systems
- **In practice simple thread priorities**
  - QNX, FreeRTOS, PikeOS, VxWorks …
- **Often without time quantum**
  - Unlike Linux

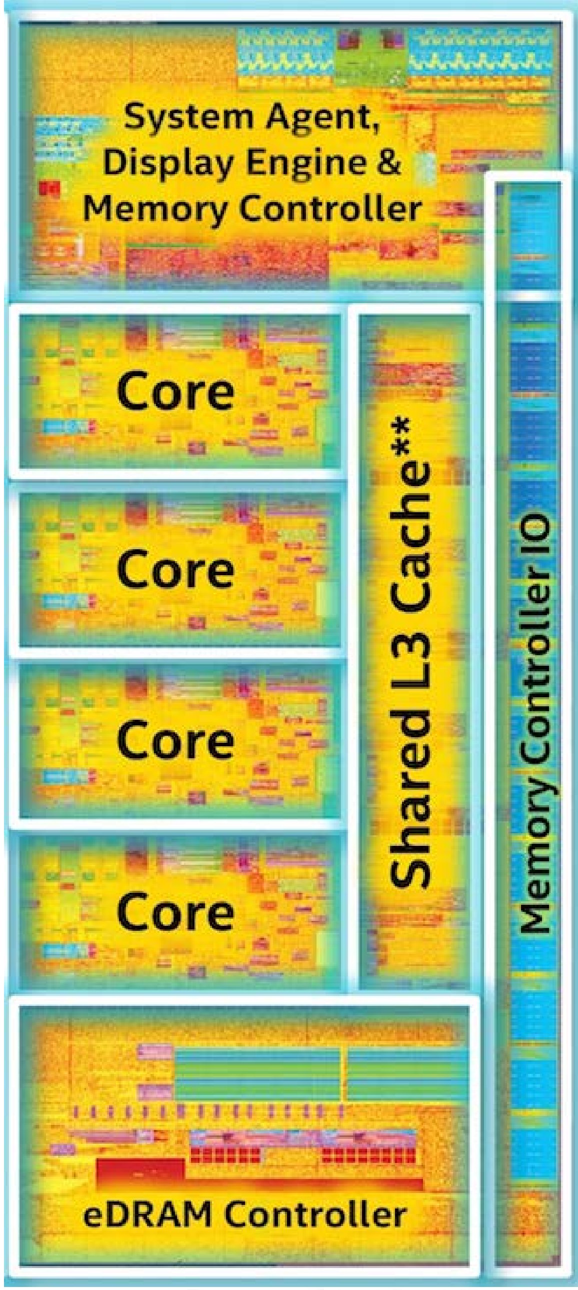
WCET

- **=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**
- **Jitter**
  - Context switches
  - Interrupt duration
  - Interrupt latencies

SYSGO EMBEDDING INNOVATIONS

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?



More enemies

- **Modern hardware**
  - Lazy algorithms
  - Branch predictors
  - Out-of-order execution
    - Unpredictable pipeline
  - TLB, caches
- **Modern OS features**
  - Paging, overcommit
  - Copy on Write
  - Thread migration
- **Complexity in general**

Memory Management

- **Sometimes no MMU at all**
  - FreeRTOS, some VxWorks
- **Simple virtual to physical mapping**
  - X Paging, memory mapped files, copy on write …
  - ✓ Shared memory
  - ✓ Memory protection (NX bit etc.)
- **No (ab-)use of free memory for buffers**

PikeOS Kernel Memory

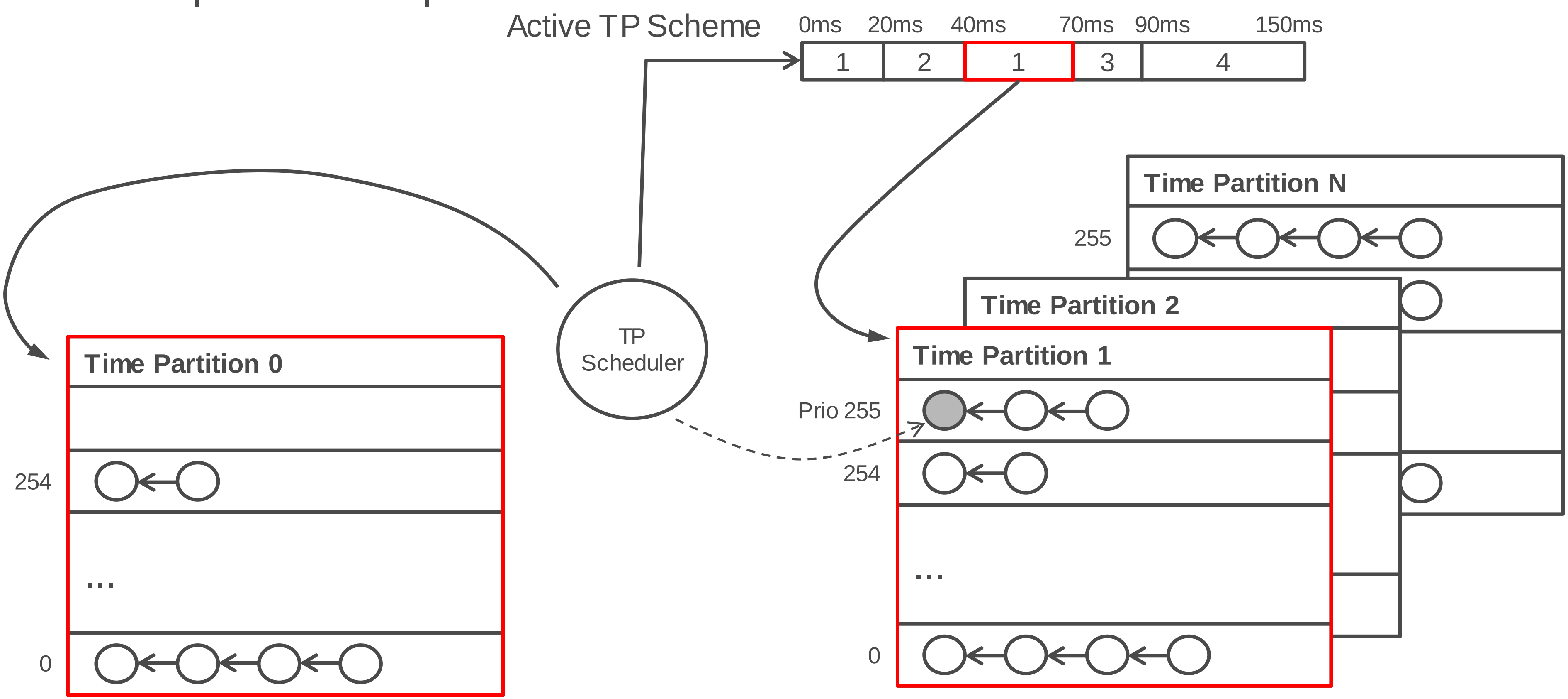
- **User-space needs kernel memory**
  - Threads
  - Processes
  - Memory mappings
- **Pre-allocated pools**
  - Safe limit
  - Avoids extra locks

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**

Scheduling

- **ARINC653 (avionics standard) is common**
  - Time partitions + priorities



TP0 is PikeOS extension

Multi-Processor

- **Threads are bound to single CPU**
  - Explicit migration
  - 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-realtime APP1	Idle	Non-realtime APP3
CPU 1	Linux	Real-time APP	Non-realtime 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
- **Build time design**
  - Static number of FDs, buffers etc.

Other considerations

- **Choose a suitable HW**
  - NXP, Xilinx …
- **Control over the platform**
  - You are not alone on X86
  - System Management Mode
  - Intel Management Engine

Coding guidelines

- **MISRA C coding standard**
  - Ex. Rule: Initializer lists shall not contain persistent side effects
- **In OS development, you have to break some of them**
  - Ex. Rule: A conversion should not be performed between a pointer to object and an integer type

Mixing critical and non-critical …

Why microkernel?

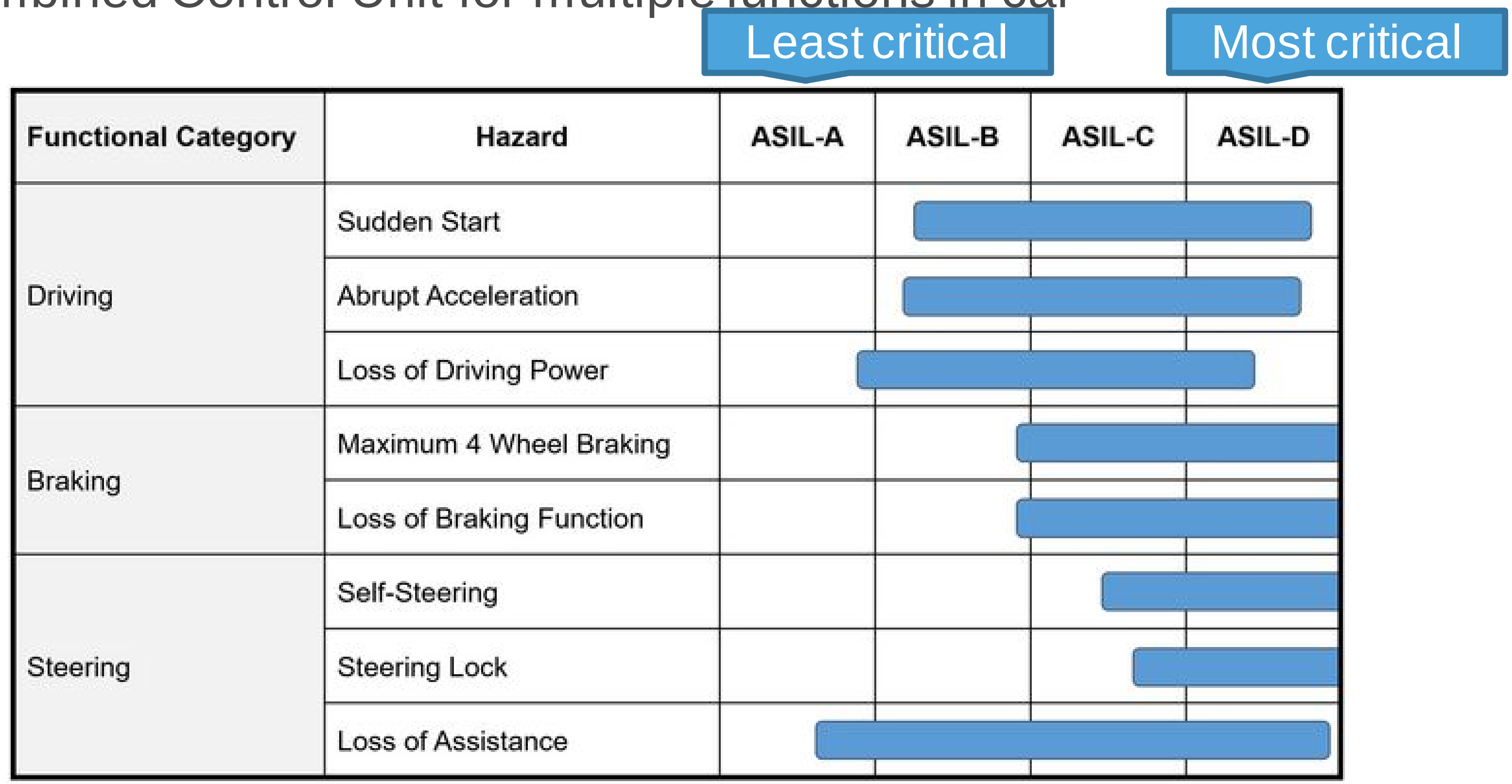
- **Separate critical and non-critical components**
  - MMU required
- **We need to certify**
  - The critical components
  - The kernel
  - Smaller kernel = less work
- **Non-critical parts can use**
  - Off-the-shelf software
  - Linux
  - => Easier development

Why microkernel?

- **Alternatives**
  - Certify everything
  - Build two physically separate systems
- **In PikeOS you can choose**
  - Kernel driver
  - User-space driver
- **Clear(er) line between levels of criticality**
  - Desktop PC crash is not fatal if you save your work

Mixed criticality ex.

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



Partitioning example - Airbus A400M

Level B

Ramp, Doors,
Aerial Delivery,
Cargo Locks
...

Level B

Graphics
OpenGL
GUI
HMI

Level C

Winches,
Crane
....

Level D

9 Applications
incl.
Waste&Water

PikeOS Virtualization Platform

Hardware

Pictures: Rheimetall Defense A400M

User-space drivers

- **Modern hardware looks like a memory (MMIO)**
- **Can be mapped to user-space using MMU**
- **PikeOS interrupt handler is a user-space thread**
  - with regular scheduling

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

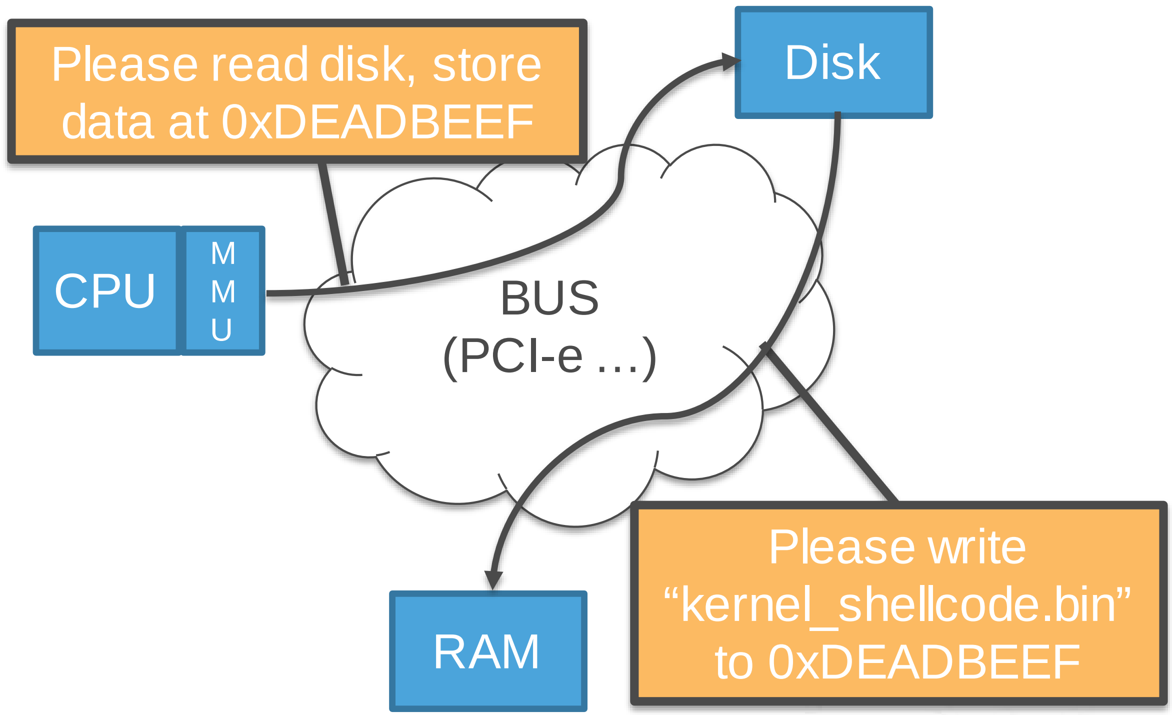
Interrupt handling

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

  \+ variations for different platforms
- **Solaris, FreeBSD and others also run interrupt routines in threaded context**

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
- **Commonly used for PCI pass-throught**

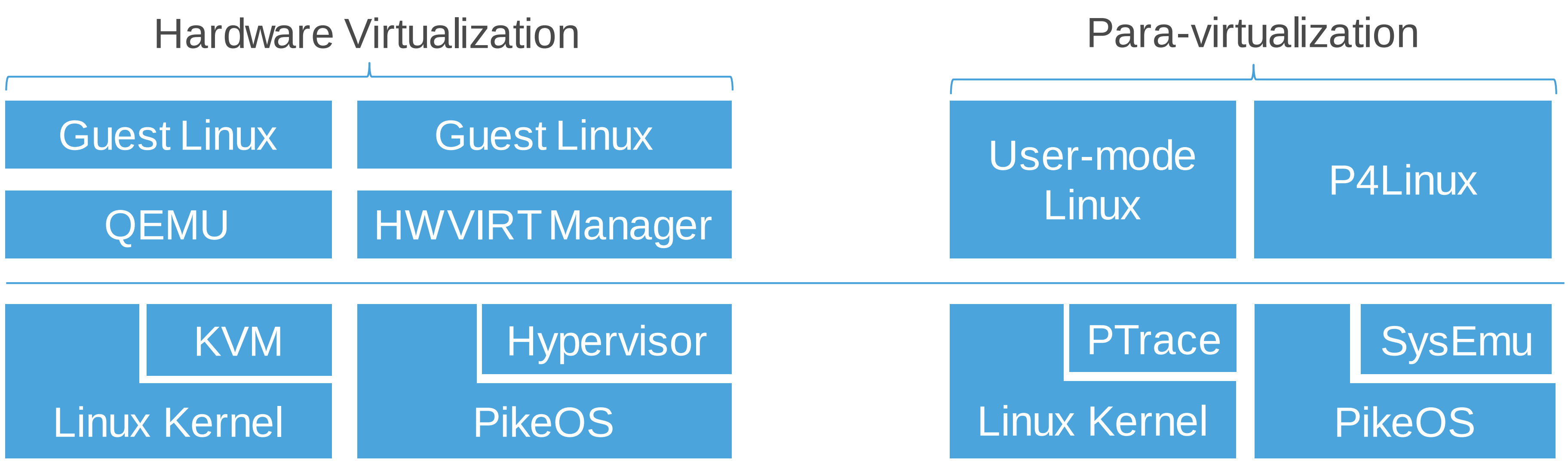
Why virtualization?

- **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

SYSGO
EMBEDDING INNOVATIONS

Virtualization comparison

- **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

SYSGO
EMBEDDING INNOVATIONS

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**
  - Copes surprisingly well with it
- **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
  - You can play OpenArena on an Intel GPU

Where can I get PikeOS?

- **Not Commercial, Off-the-shelf product**
- **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 mixed-criticality certified usage. Alternatives:**
    - Linux with RT patches? FreeRTOS?
    - Lots of other RTOSes

SYSGO EMBEDDING INNOVATIONS

Certification/safety I

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“

Automotive Safety Integrity Level

D	C	B	A

Safety: DO-178C - Avionics
„Software Considerations in Airborne Systems and Equipment Certification“

Design Assurance Level

D	C	B	A

Safety: EN 50128/29 - Railway
„Software for Traincontrol and –management systems“

Safety Integrity Level

1	2	3	4

Safety: IEC 61508 - Industry
„Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems“

Safety Integrity Level

1	2	3	4

Security: SAR - Avionics
„Airbus Security Standard“

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

SYSGO EMBEDDING INNOVATIONS

Certification/safety II

- DO178, …
  - DO178 DAL C (medium) – 2-3 verification engineers on 1 developer
- Requirement-based testing
  - High-level requirements, interface requirements, low-level requirements
    - Traceability between all levels of requirements, code and tests is essential
      - Code is annotated (by corresponding requirement name)
  - 80% of verification efforts writing automated tests
    - Minority of tests can be manual or rarely just code analysis
    - From DAL C all code must be covered by tests
  - The rest formal reviews (of documents, code, tests), WCET analysis, stack analysis
  - Independence between development and verification (verification engineer cannot commit into the verified code, …)
- Bunch of other documents (plans, standards, …)

Certification/security I

- 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 Cherokie, 2015, documented a possibility of disabling brakes over Internet (cellular phone connection)
  - http://illmatics.com/Remote%20Car%20Hacking.pdf

Certification/security II

- 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

Possible topics for intership, thesis or project

- Applied research topics (thesis, research paper)
  - IAT0134 MPLockingProtocol
  - IAT0136 EvaluationOfFormalMethodsToolsForVVDepartment
  - IAT0104 SchedulerFormalVerificationDiplomaThesis
- Implementation topics (student project, thesis)
  - IAT0133 PSPraspberrypi3
  - IAT0132 IPT-PikeOs-support
  - IAT0135 IntegrateLWTinCodeo

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

- 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.

Testsuite example

- **TS_ANIS**
  - **ANIS = UDP/IP network stack certified for DAL C**
  - **Low-level testsuite**
  - **694 test cases**
  - **587 interface requirements, 755 design requirements**
  - **125 000 LOC of C code**
  - **> 1000 pages of test suite description**
  - **~ 4000 manhours**
- **ANIS itself has 80 000 LOC of C code**
- One test case 1-3 manhours in simplest cases; manweeks in most complex cases