KERNKONZEPT

DEVELOPING A MISSION-CRITICAL & SAFETY-CRITICAL OPERATING SYSTEM

Martin Děcký

INTRODUCTION

000

About the Speaker

- **Charles University in Prague, Faculty of Mathematics and Physics**
  - MSc. (2005), Ph.D. (2015)
  - Researcher at the Department of Distributed and Dependable Systems (2008 – 2017)
  - Co-author of the HelenOS microkernel multiserver operating system (since 2004)
- **Huawei Technologies**
  - Senior Research Engineer at the *Munich Research Center* (2017 – 2019)
  - Principal Research Engineer and co-founder of the *Dresden Research Center* (2019 – 2021)
  - Contributing to the HarmonyOS NEXT microkernel-based operating system
- **Kernkonzept GmbH**
  - Senior Software Engineer (since 2021)
  - Contributing to the L4Re microkernel-based operating system framework

About the Speaker

- **Invitation: Advanced Operating Systems**
  - NSWI161
  - Summer semester course
    - Originally since 2017
    - New form since 2022
  - Continuation of the Operating Systems winter semester course
    - Advanced algorithms and techniques
    - Focus on challenges and trade-offs of real-world operating systems
  - Lectures by yours truly and other invited speakers

About Kernkonzept

Owner-managed	Founded 2012	Spin-off from TU Dresden
International team of 35	Wide experience since 1996	Continuously growing
Close to research and innovative	Operating system specialists	Located in Dresden, Germany

Kernkonzept Markets

AUTOMOTIVE

HIGH ASSURANCE

CYBER SECURITY

SECURE ENDPOINT

SMART HOME

SECURE CLOUD

INDUSTRIAL IOT

AVIONICS

Kernkonzept Customers

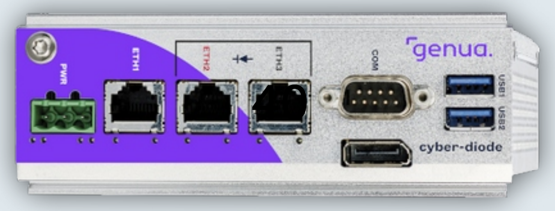
GERMAN
SECRET

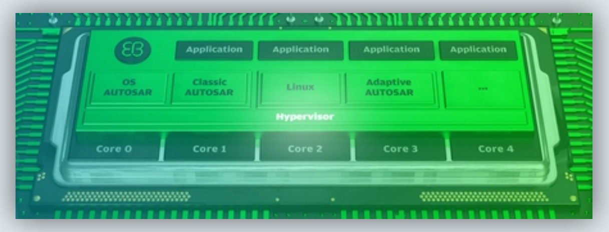
EU
SECRET

NATO
SECRET

Kernkonzept Customers

- **infodas**
  - *SDoT Security Gateway* and other products
    - German & NATO SECRET classification
- **genua**
  - *Secure laptop, Cyber data diode*
    - BSI approval for NATO SECRET & EU SECRET
- **Elektrobit**
  - Wholly-owned subsidiary of Continental
  - *EB Corbos Hypervisor*
    - Bare-metal mixed-criticality hypervisor for automotive systems (targeting Adaptive AUTOSAR)
    - Actually running in *Volkswagen ID.3* and other cars
- **Electrolux, Airbus, IABG, etc.**

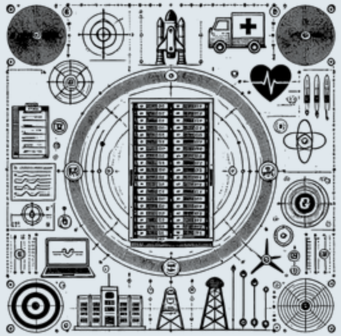




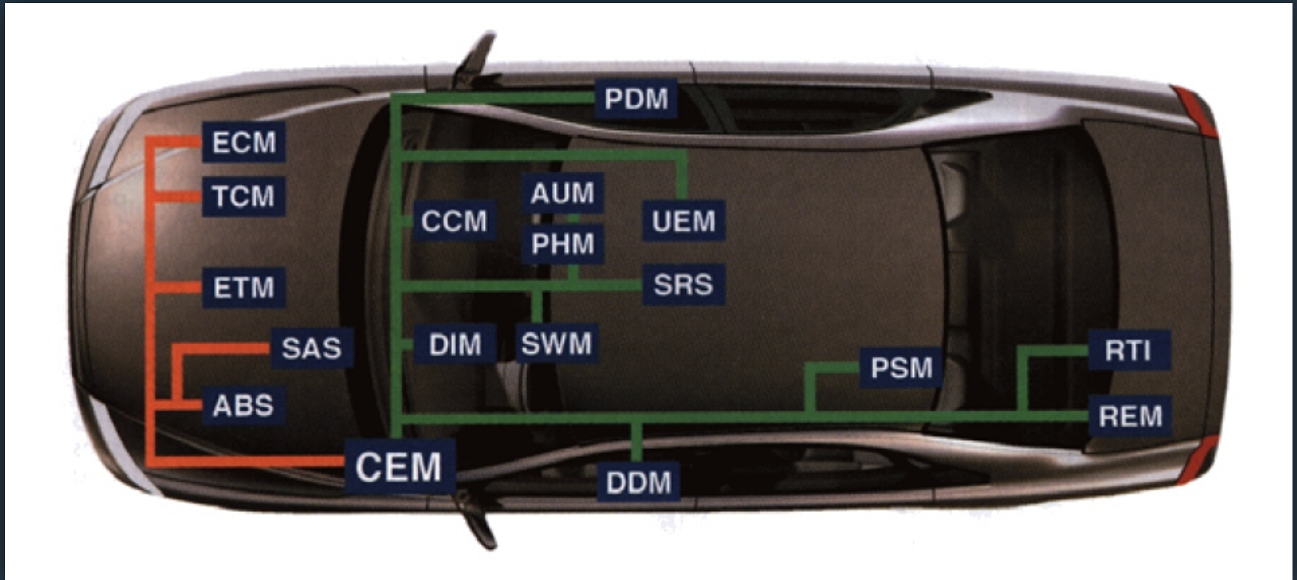
MISSION-CRITICAL
SAFETY-CRITICAL

001

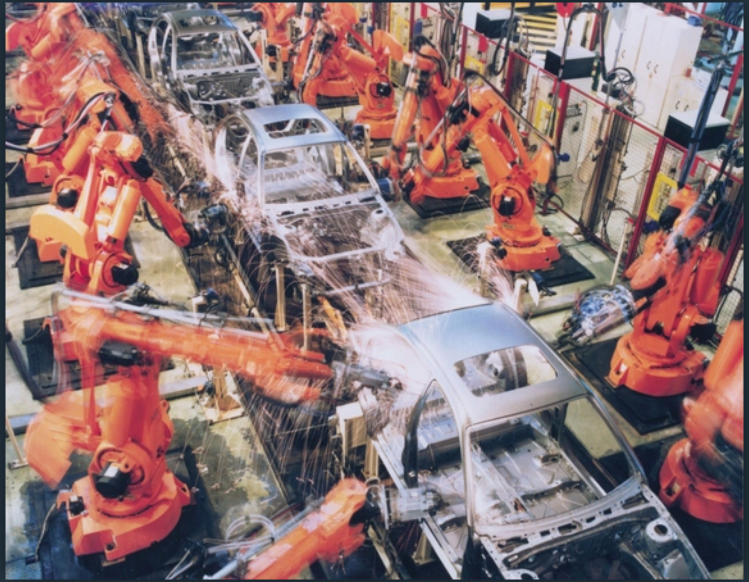
Critical Systems



- **Mission-critical systems**
  - Essential to business/organization survival
    - E.g. on-line banking, state secrets, transport operation, electric grid
  - Usually associated with *security* properties (protecting computers against humans)
    - Fail-safe design
- **Safety-critical systems**
  - Essential to human well-being and survival
    - E.g. medical devices, transport control, nuclear power plant control
  - Usually associated with *safety* properties (protecting humans against computers)
    - Fail-operational design

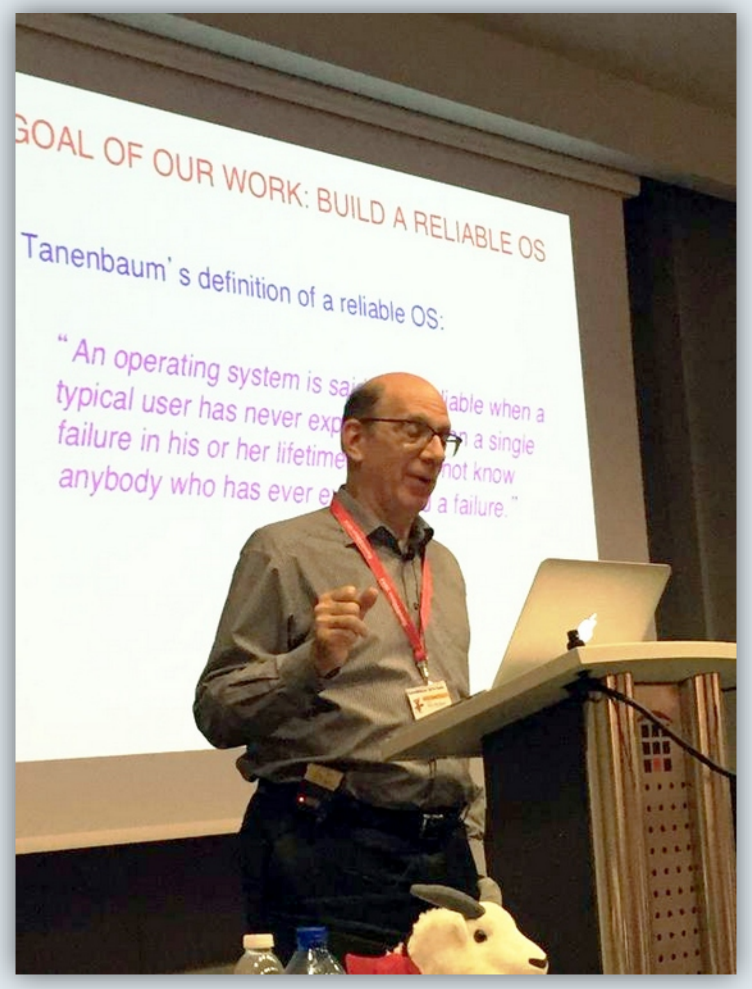

PDM
ECM
TCM
CCM
AUM
PHM
UEM
ETM
SRS
SAS
DIM
SWM
PSM
RTI
ABS
REM
CEM
DDM

Cité de l'espace
esa
cnes



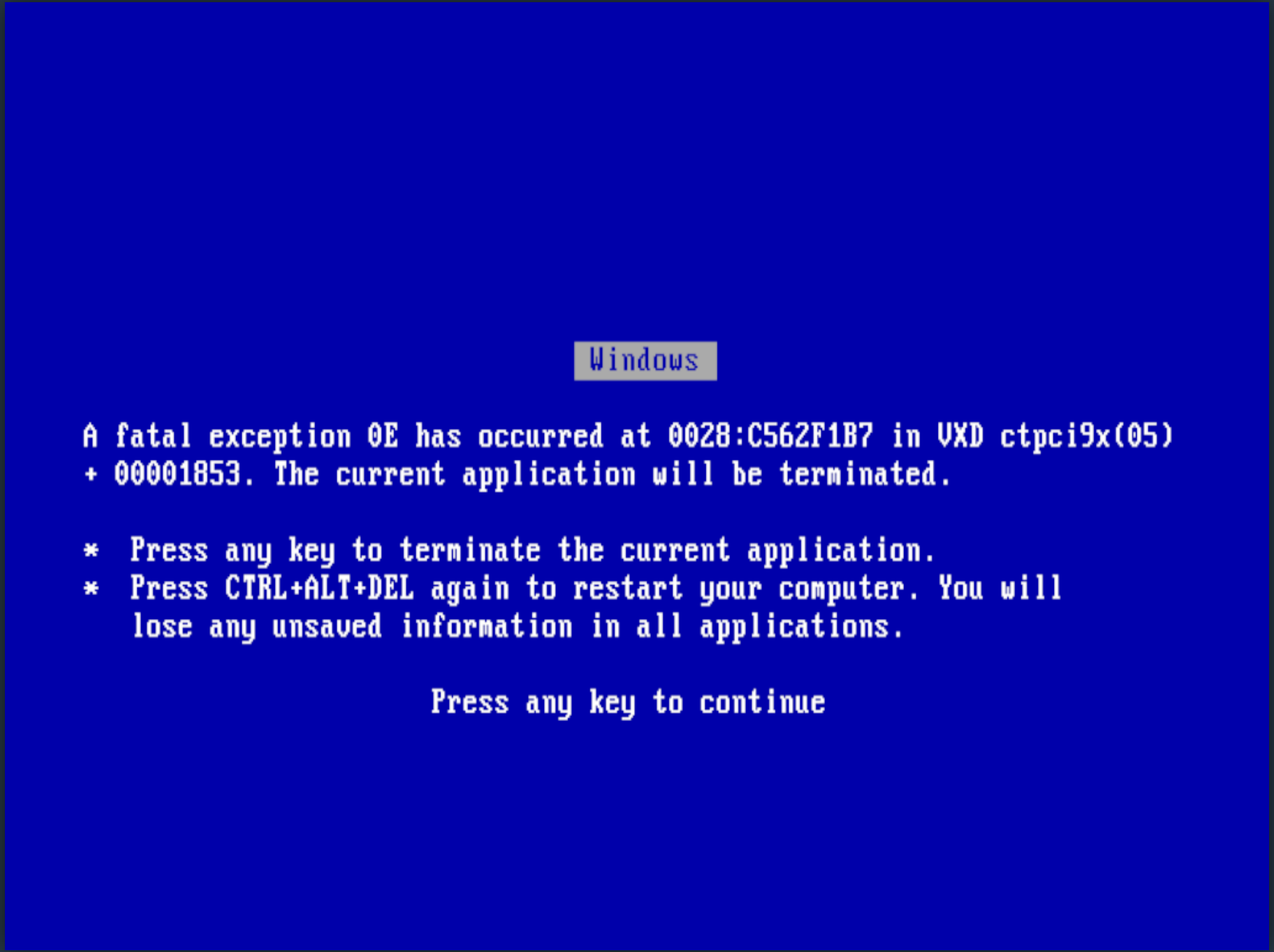
Operating System Reliability

- **Necessary* condition for general reliability**
  - Ability to perform its intended function without failure
    - Probability function depending on assumptions
    - *“An operating system is said to be reliable when a typical user has never experienced even a single failure in his or her lifetime and does not know anybody who has ever experienced a failure.”* [Tanenbaum 2014]
  - Dependability
    - *“Dependability is a measurable and provable degree of system's availability, reliability and its maintenance support”* [IEEE 2004]


GOAL OF OUR WORK: BUILD A RELIABLE OS
Tanenbaum's definition of a reliable OS:

Andy Tanenbaum at EuroBSDcon 2014, Sofia, Bulgaria
Photo by Ollivier Robert

* Unfortunately, not a satisfying condition.

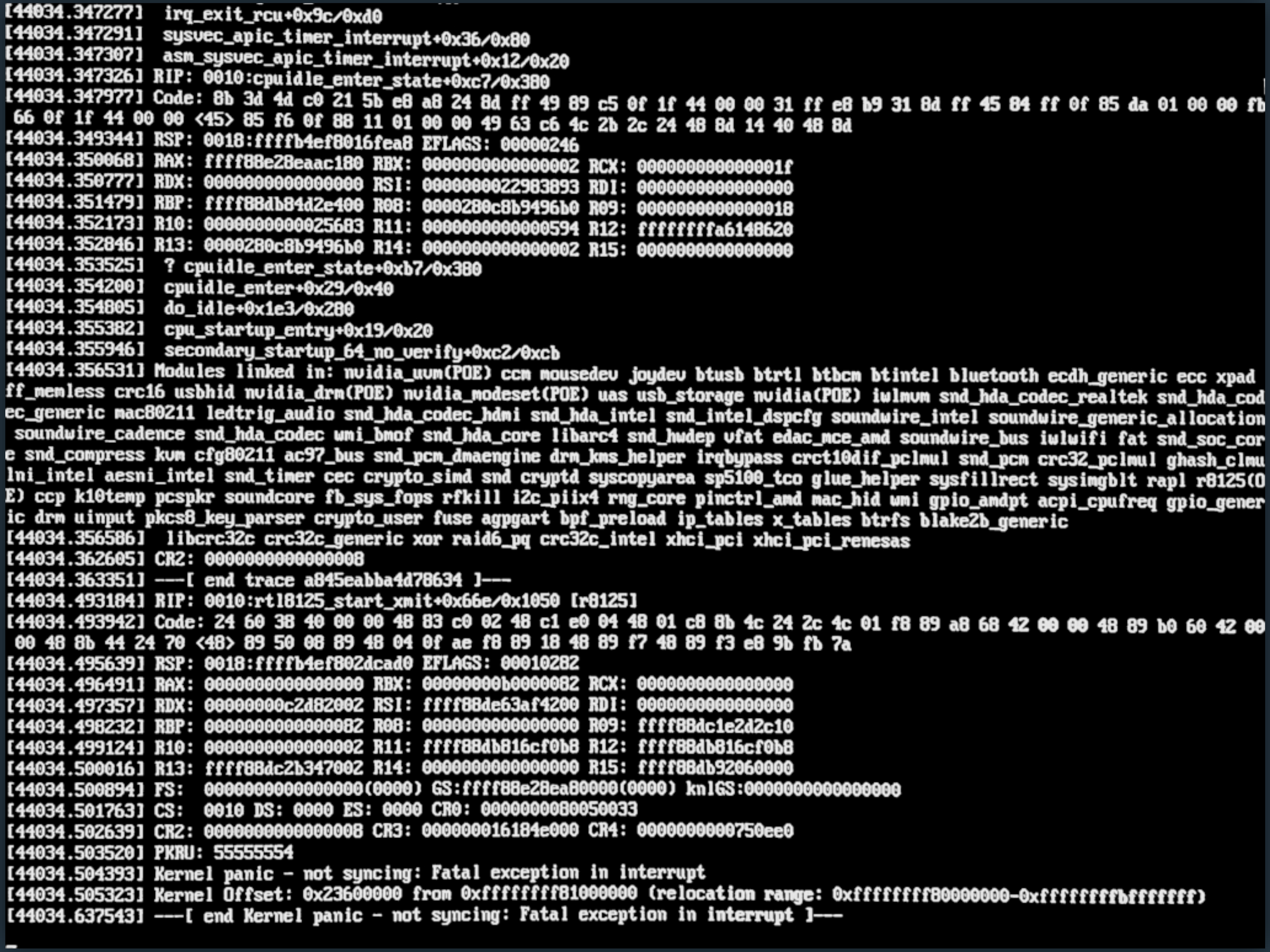


```
                              Windows

 A fatal exception 0E has occurred at 0028:C562F1B7 in VXD ctpci9x(05)
 + 00001853. The current application will be terminated.

 *  Press any key to terminate the current application.
 *  Press CTRL+ALT+DEL again to restart your computer. You will
    lose any unsaved information in all applications.

                         Press any key to continue
```



```
[44034.347277]  irq_exit_rcu+0x9c/0xd0
[44034.347291]  sysvec_apic_timer_interrupt+0x36/0x80
[44034.347307]  asm_sysvec_apic_timer_interrupt+0x12/0x20
[44034.347326] RIP: 0010:cpuidle_enter_state+0xc7/0x380
[44034.347977] Code: 8b 3d 4d c0 21 5b e8 a8 24 8d ff 49 89 c5 0f 1f 44 00 00 31 ff e8 b9 31 8d ff 45 84 ff 0f 85 da 01 00 00 fb
 66 0f 1f 44 00 00 <45> 85 f6 0f 88 11 01 00 00 49 63 c6 4c 2b 2c 24 48 8d 14 40 48 8d
[44034.349344] RSP: 0018:ffffb4ef8016fea8 EFLAGS: 00000246
[44034.350068] RAX: ffff88e28eaac180 RBX: 0000000000000002 RCX: 000000000000001f
[44034.350777] RDX: 0000000000000000 RSI: 0000000022983893 RDI: 0000000000000000
[44034.351479] RBP: ffff88db84d2e400 R08: 0000280c8b9496b0 R09: 0000000000000018
[44034.352173] R10: 0000000000025683 R11: 0000000000000594 R12: ffffffffa6148620
[44034.352846] R13: 0000280c8b9496b0 R14: 0000000000000002 R15: 0000000000000000
[44034.353525]  ? cpuidle_enter_state+0xb7/0x380
[44034.354200]  cpuidle_enter+0x29/0x40
[44034.354805]  do_idle+0x1e3/0x280
[44034.355382]  cpu_startup_entry+0x19/0x20
[44034.355946]  secondary_startup_64_no_verify+0xc2/0xcb
[44034.356531] Modules linked in: nvidia_uvm(POE) ccm mousedev joydev btusb btrtl btbcm btintel bluetooth ecdh_generic ecc xpad
ff_memless crc16 usbhid nvidia_drm(POE) nvidia_modeset(POE) uas usb_storage nvidia(POE) iwlmvm snd_hda_codec_realtek snd_hda_cod
ec_generic mac80211 ledtrig_audio snd_hda_codec_hdmi snd_hda_intel snd_intel_dspcfg soundwire_intel soundwire_generic_allocation
 soundwire_cadence snd_hda_codec wmi_bmof snd_hda_core libarc4 snd_hwdep vfat edac_mce_amd soundwire_bus iwlwifi fat snd_soc_cor
e snd_compress kvm cfg80211 ac97_bus snd_pcm_dmaengine drm_kms_helper irqbypass crct10dif_pclmul snd_pcm crc32_pclmul ghash_clmu
lni_intel aesni_intel snd_timer cec crypto_simd snd cryptd syscopyarea sp5100_tco glue_helper sysfillrect sysimgblt rapl r8125(O
E) ccp k10temp pcspkr soundcore fb_sys_fops rfkill i2c_piix4 rng_core pinctrl_amd mac_hid wmi gpio_amdpt acpi_cpufreq gpio_gener
ic drm uinput pkcs8_key_parser crypto_user fuse agpgart bpf_preload ip_tables x_tables btrfs blake2b_generic
[44034.356586]  libcrc32c crc32c_generic xor raid6_pq crc32c_intel xhci_pci xhci_pci_renesas
[44034.362605] CR2: 0000000000000008
[44034.363351] ---[ end trace a845eabba4d78634 ]---
[44034.493184] RIP: 0010:rtl8125_start_xmit+0x66e/0x1050 [r8125]
[44034.493942] Code: 24 60 38 40 00 00 48 83 c0 02 48 c1 e0 04 48 01 c8 8b 4c 24 2c 4c 01 f8 89 a8 68 42 00 00 48 89 b0 60 42 00
 00 48 8b 44 24 70 <48> 89 50 08 89 48 04 0f ae f8 89 18 48 89 f7 48 89 f3 e8 9b fb 7a
[44034.495639] RSP: 0018:ffffb4ef802dcad0 EFLAGS: 00010282
[44034.496491] RAX: 0000000000000000 RBX: 00000000b0000082 RCX: 0000000000000000
[44034.497357] RDX: 00000000c2d82002 RSI: ffff88de63af4200 RDI: 0000000000000000
[44034.498232] RBP: 0000000000000082 R08: 0000000000000000 R09: ffff88dc1e2d2c10
[44034.499124] R10: 0000000000000002 R11: ffff88db816cf0b8 R12: ffff88db816cf0b8
[44034.500016] R13: ffff88dc2b347002 R14: 0000000000000000 R15: ffff88db92060000
[44034.500894] FS:  0000000000000000(0000) GS:ffff88e28ea80000(0000) knlGS:0000000000000000
[44034.501763] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[44034.502639] CR2: 0000000000000008 CR3: 000000016184e000 CR4: 0000000000750ee0
[44034.503520] PKRU: 55555554
[44034.504393] Kernel panic - not syncing: Fatal exception in interrupt
[44034.505323] Kernel Offset: 0x23600000 from 0xffffffff81000000 (relocation range: 0xffffffff80000000-0xffffffffbfffffff)
[44034.637543] ---[ end Kernel panic - not syncing: Fatal exception in interrupt ]---
_
```

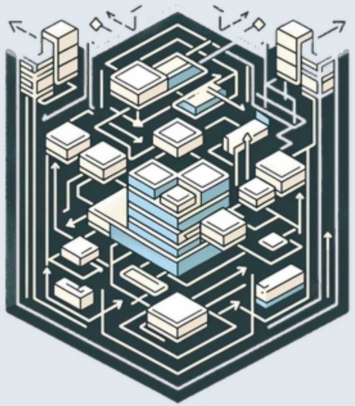
FUNDAMENTALLY RELIABLE OPERATING SYSTEMS

010

Motivation

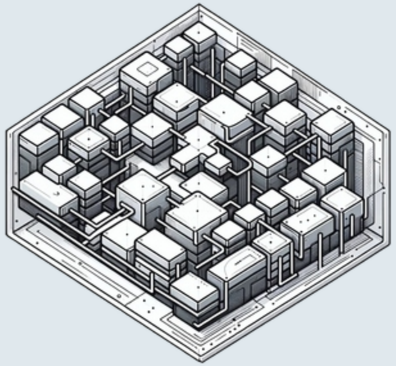
- **Avoiding fundamentally unreliable software architecture**
  - *"To me, writing a monolithic system in 1991 is a truly poor idea."* [Tanenbaum 1991]
  - *"There are no demonstrated examples of highly secure or highly robust unstructured (monolithic) systems in the history of computing."* [Shapiro 2006]
  - Biggs S., Lee D., Heiser G.: *The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-based Designs Improve Security*, ACM 9th Asia-Pacific Workshop on Systems (APSys), 2018
    - *"While intuitive, the benefits of the small TCB have not been quantified to date. We address this by a study of critical Linux CVEs, where we examine whether they would be prevented or mitigated by a microkernel-based design. We find that almost all exploits are at least mitigated to less than critical severity, and 40 % completely eliminated by an OS design based on a verified microkernel, such as seL4."*

Microkernel-Based Operating Systems



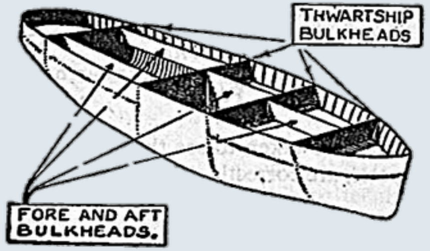
- **Built according to coherent design principles**
  - **Component-based architecture**
    - Operating system composed of isolated components that communicate via well-defined interfaces
  - **Separation of concerns**
    - Each component takes care of a specific well-defined functionality and implements it well
  - **Split of mechanism and policy**
    - Components implement generic mechanisms without implicitly imposing a specific policy on the client components
  - **Least privilege**
    - Components have a minimal set of privileges required to do their job

Microkernel-Based Operating Systems

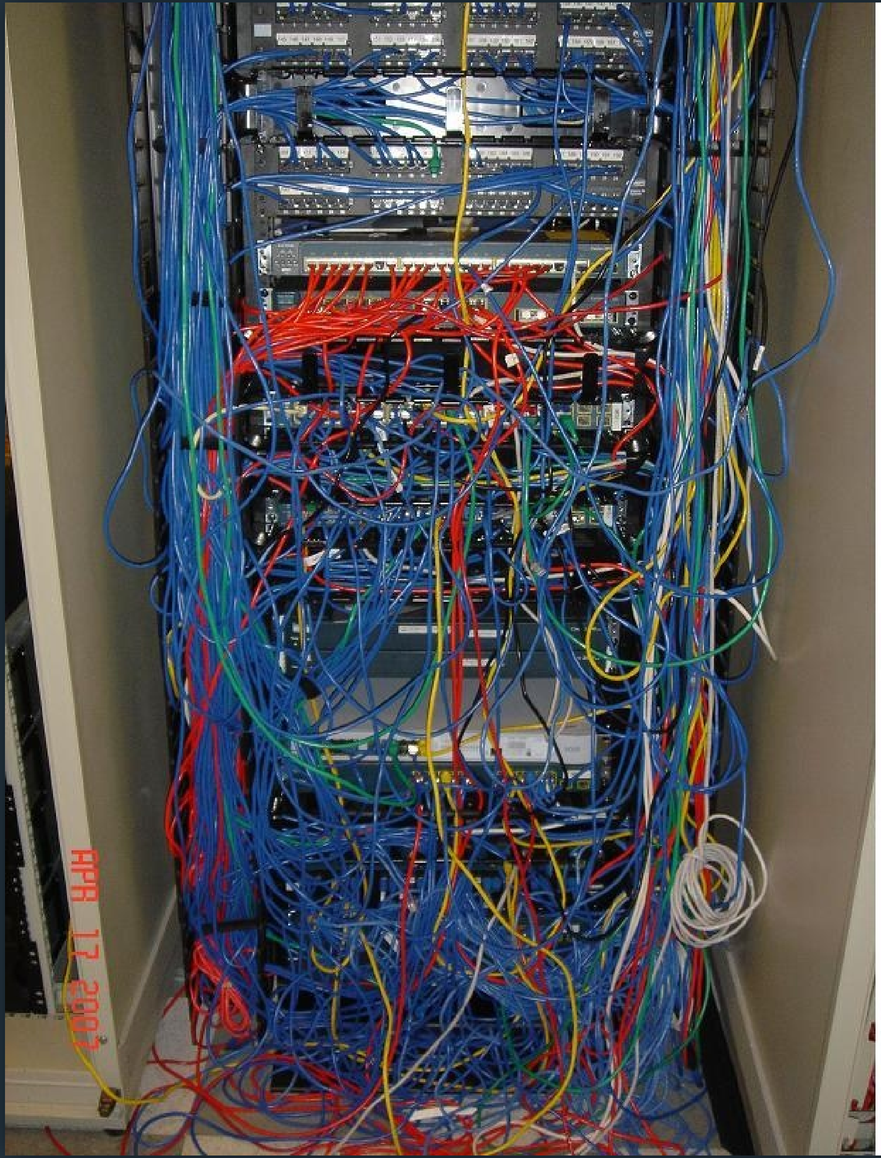


- **Typical emerging properties**
  - **Fine-grained components**
    - As opposed to monolithic components
  - **Minimality of the kernel & trusted computing base**
    - Most mechanisms do not require the privileged CPU mode
    - File systems, most device drivers, security policies, etc., run as user mode components
  - **Modularity**
    - Replacing component implementation while keeping the interface
  - **Seamless virtualization**
    - VMs and tasks are essentially similar entities

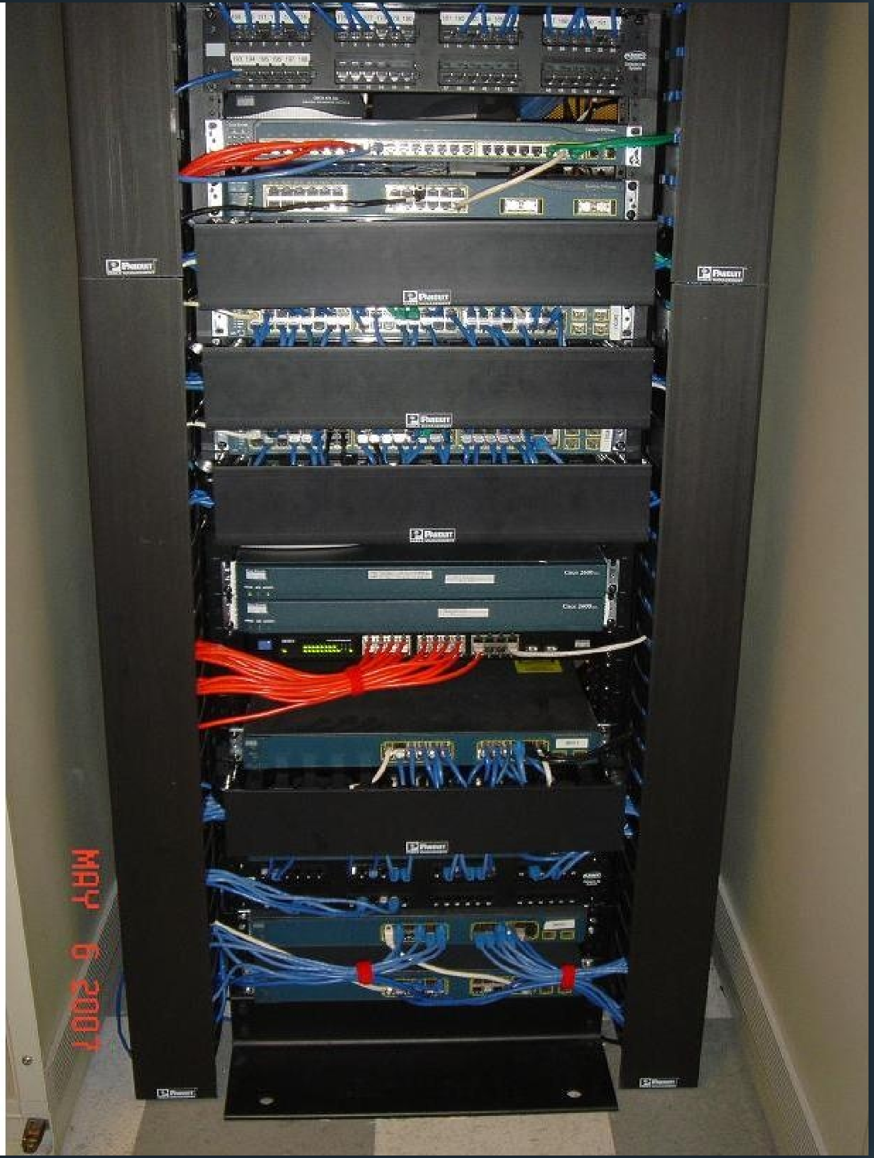
Microkernel-Based Operating Systems



- **Typical emerging properties**
  - **Loose module coupling**
    - Configurability via different composition of modules
    - Policies in user space and distributed
  - **Architectural safety, security, reliability and dependability guarantees**
    - Limiting the “blast radius” of faults at run time
  - **Architectural enabler for advanced reasoning about correctness**
    - Certification
    - Real-time guarantees
    - Formal verification



APR 17 2007



MAY 6 2007

Abridged History of Microkernels

- **1969**
  - *RC 4000 Multiprogramming System*
    - Per Brinch Hansen (Regnecentralen)
    - Separation of mechanism and policy, modularity via isolated concurrently running processes, message passing
    - Same year as Multics
- **1971**
  - *HYDRA*
    - William Wulf (Carnegie Mellon University)
    - Capability-based, object-oriented kernel
    - Around the same time as UNIX

Abridged History of Microkernels

- **1979**
  - *EUMEL / L2*
    - Jochen Liedtke (University of Bielefeld)
    - Microkernel running bitcode virtual machines
- **1982**
  - *QNX*
    - Gordon Bell, Dan Dodge (University of Waterloo, later Quantum Software Systems)
    - Earliest commercially successful microkernel-based OS (still in active development and use today, owned by BlackBerry)

Abridged History of Microkernels

- **1985**
  - *CMU Mach*
    - Richard Rashid, Avie Tevanian (Carnegie Mellon University)
    - Arguably the most widespread microkernel code base
      - Core part of the operating systems by Apple (no longer following the original design principles) and GNU/Hurd
    - Highly influential
      - Affected the design of Windows NT
      - Establishing the usual terminology and conventions
    - Well-publicized shortcomings

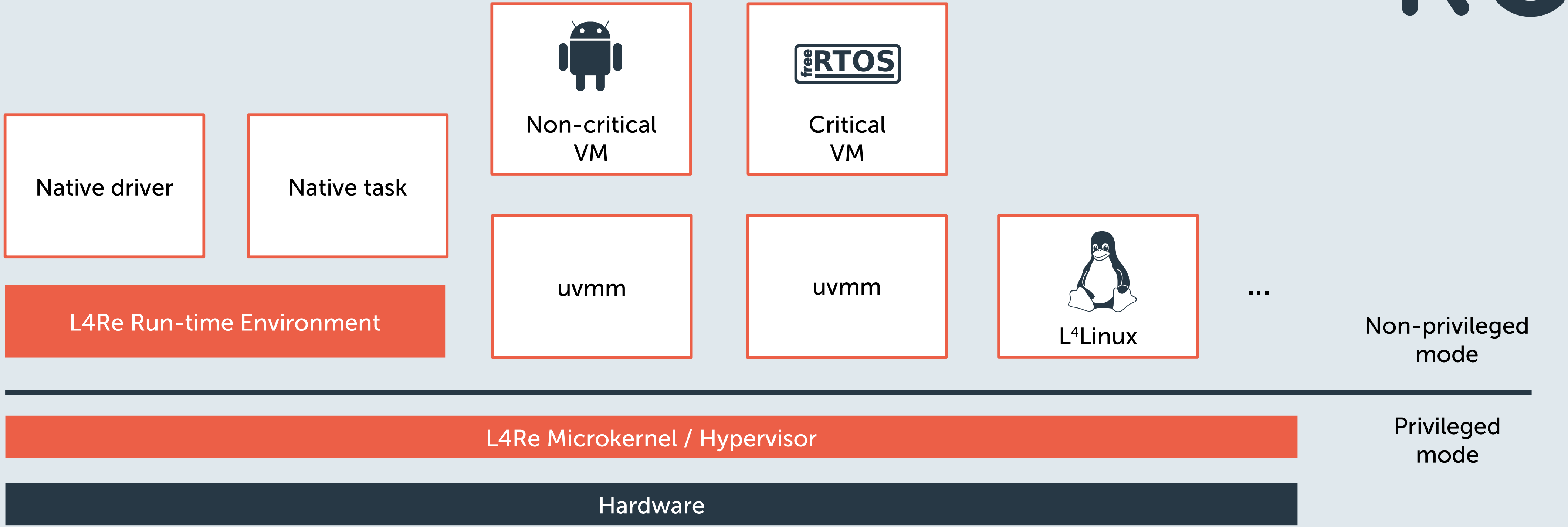
Abridged History of Microkernels

- **1988**
  - *L3*
    - Jochen Liedtke (Gesellschaft für Mathematik und Datenverarbeitung, later known as Fraunhofer)
    - Addressing the main performance issues of CMU Mach
      - Synchronous rendezvous-style remote calls instead of asynchronous in-kernel buffered message passing
- **1993**
  - *L4*
    - Order of magnitude performance improvement compared to CMU Mach
      - Small and cache-friendly kernel working set, fast-path IPC without complex processing (access rights, data interpretation, etc.)
    - User mode pagers and recursive address spaces
    - Non-portable hand-written assembly implementation (for 486 and Pentium)
    - Liedtke J.: *Improving IPC by Kernel Design*, ACM SIGOPS Operating Systems Review, Volume 27, Issue 5, 1993

L4
Re

011

L4Re in a Nutshell



Native driver

Native task

Non-critical VM

Critical VM

L4Re Run-time Environment

uvmm

uvmm

L^4Linux

...

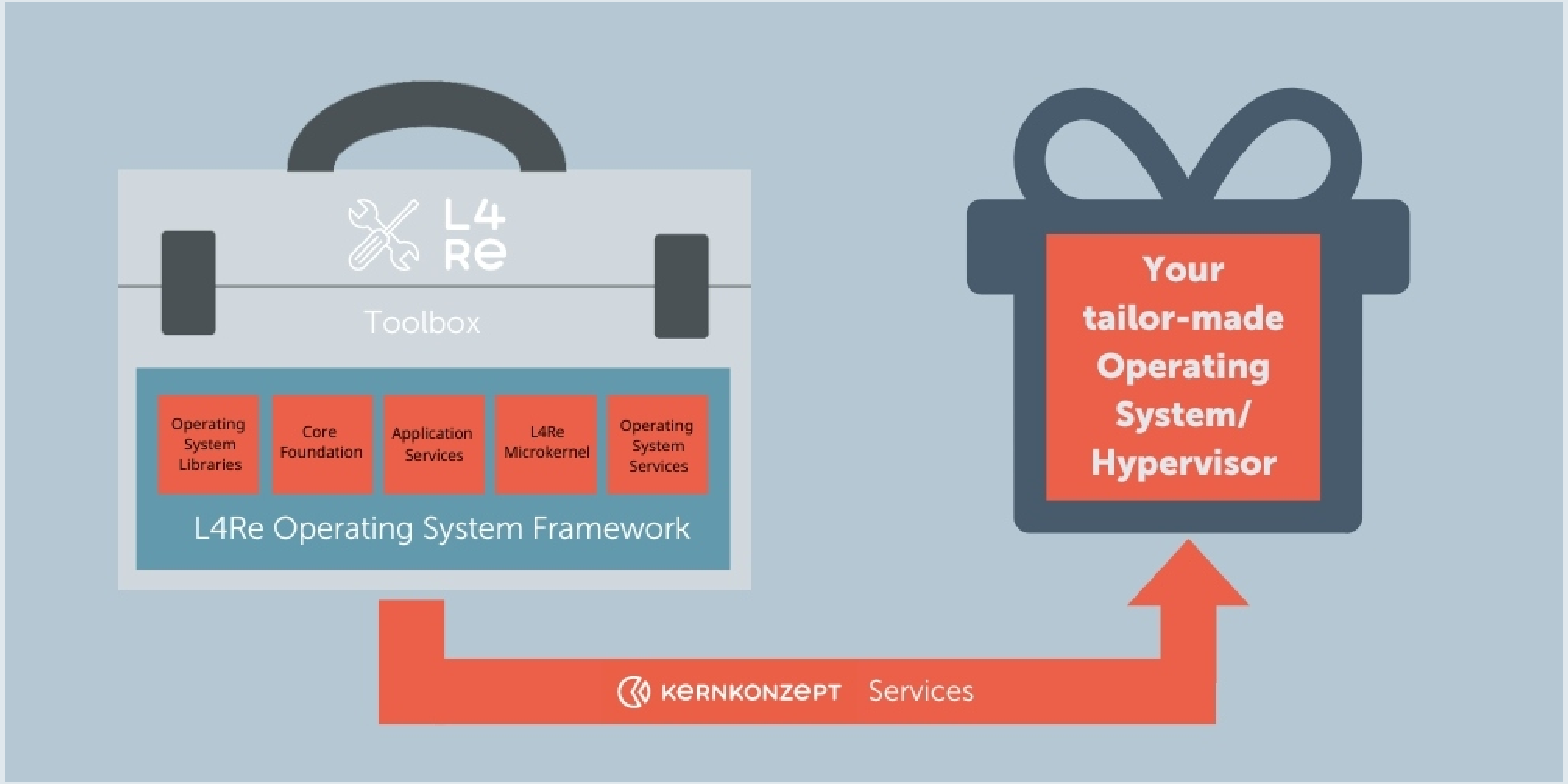
Non-privileged mode

Privileged mode

L4Re Microkernel / Hypervisor

Hardware

L4Re in a Nutshell


L4 Re
Toolbox
Operating System Libraries
Core Foundation
Application Services
L4Re Microkernel
Operating System Services
L4Re Operating System Framework
Your tailor-made Operating System/ Hypervisor
KERNKONZEPT Services

L4Re

- **Microkernel**
  - Designed at TU Dresden, follows the historical lineage from **L4/x86**
    - Continuity in design, not in code, API or ABI
  - Direct predecessor: **L4/Fiasco**
    - Original implementation by Michael Hohmuth and others
    - Fully preemptive kernel targeting real-time workloads
    - Portable C++ with a custom preprocessor
    - The name refers to the legal struggles of releasing the original L4/x86 code as open source
  - Current: **L4Re Microkernel** (previously known as *Fiasco.OC*)
    - Original implementation by Alexander Warg and others
    - Object capabilities (popularized by Jonathan S. Shapiro)
    - Support for x86, x86-64, ARM (32/64), MIPS (32/64) and RISC-V

L4Re

- **User space**
  - Original implementation by Alexander Warg and others
  - Follows the historical lineage from *L4Env*
  - Specifically targets the object capability API of the L4Re Microkernel
  - L4Re-core
    - User space run-time environment (primarily C and C++)
    - *sigma0* (default pager)
    - *Moe* (root task)
    - *Ned* (initialization task)
  - Catalogue of other user space components / packages / libraries
    - *IO*, *uvmm*, *L^{4}Linux*, device drivers, file system drivers, etc.

KERNKONZEPT

100

Academic Roots

- **Vastly different (even conflicting) criteria of success**
  - **Academia:** publications, citations
    - *Software project* as a vehicle for hosting the research on novel radical ideas
      - Publications are the actual products
      - Only needs to be sufficiently usable and practical for the evaluation and benchmarking
        - No need to cover all real-world corner cases
  - **Industry:** revenue
    - *Software project* as a vehicle for customer satisfaction
      - The actual product itself
      - Usable and practical for all real-world corner cases
        - Pragmatism and down-to-earth approaches might win over novel radical ideas

Commercial Environment

- **Reliably fulfilling the specific needs of (our) customers**
- **Better customizability and less unnecessary baggage than the competition**
  - Modularity helps by itself, but sometimes individual product lines are needed
- **Balance between principled and pragmatic design decisions**
  - Design principles are the means, not the ends
  - Perfection is the enemy of the good
- **State-of-the-art software engineering is at least as important as state-of-the-art software architecture**
  - Work efficiency via processes and tooling
  - Avoiding technical debt

Commercial Environment

- **Stronger safety/security guarantees than the competition**
  - Already academically demonstrated, but the guarantees need to be practically attested and certified
  - Hard to convince an average vendor that more security/safety is needed than Linux can ever provide
    - Very few companies actually paid a fine* because of a software safety failure or a security vulnerability
      - But that day will come as more and more critical infrastructure relies on software
  - No-brainer in mission-critical and safety-critical domains
    - But traditional reliance on hardware solutions
- **Cultivation of research projects**
  - Infineon, Bosch, Continental, Siemens, Airbus, Fraunhofer, etc.
  - ETH KIT, FZI, TU Munich, TU Dresden, University of Postdam, University of Leipzig, University of Bologna, Barcelona Supercomputing Center, etc.

* Very few people actually went to jail, too.

Commercial Environment

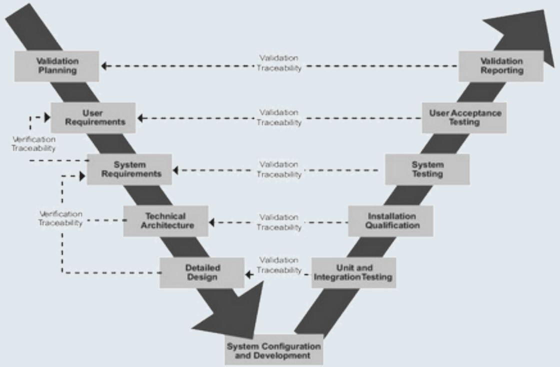
- **Interacting with the community**
  - Dresden has been the hub for operating systems research and development for decades
    - TU Dresden, Barkhausen Institut, Genode Labs, Cyberus Technology, Huawei DRC, etc.
  - Universally adopted the open source development model
  - Participating both in academic and community events (OSDI, FOSDEM, etc.)
- **Reaching out to customers**
  - Somewhat traditional means of increasing visibility
    - Trade fairs (Embedded World, etc.)
    - Industry events (Omnisecure, Bitkom Forum, SOAFEE, etc.)
    - Industrial partnerships (ST, NXP, ARM, etc.)

NZEPT
INEER
RITY
L4
Re
L4Re
OPERATING SYSTEM
AND HYPERVISOR
GERMANY
GEHEIM
EU
SECRET
NATO
SECRET
+ Common Criteria EAL 4+
+ BS
nel
WE ENGINEER SECURITY
NATIONAL SECURE CLOUD
KERNKONZEPT
Steffen
Liebergeld
EN SIE
INS

Open Source

- **Double-edged sword**
  - Openness
    - Enabling community contributions
      - Although not that frequent and requiring additional effort
    - Enabling research without centralized coordination
  - Transparency
    - Actual selling point (no *security by obscurity*)
    - Often expected in the operating systems domain (but not universally)
  - Sometimes seen as an undesired liability
    - Some people do not fully understand the GPL license and it might scare them
      - Thus moving towards the MIT license

Certification



- **Independently reviewing compliance to requirements**
  - State-of-the-art software engineering practices
    - Similar to other engineering fields (e.g. rolling stock certification)
  - External audit of code, documentation, development processes, test coverage, etc.
  - Requirements defined by a specific standard document
    - Usually informal and semi-formal qualitative and quantitative requirements
      - Formal methods only part of the highest levels of certification (and never the sole part)
    - Adherence to coding standards and best practices

Certification

- **L4Re Separation Kernel accreditation (BSI)**
  - Requirements for a microkernel-based OS for processing classified data up to a level *secret*
    - Specifically a scenario with at most one untrusted partition on x86-64
  - Accreditation artifacts
    - Security target, platform specification, secure boot documentation, high-level design, low-level design, functional specification, configuration specification, secure operations, vulnerability analysis, etc.
    - Tests covering the functional specification
  - Completed

Certification

- **L4Re Common Criteria EAL4+ certification**
  - Requirements for strong security and capability separation
  - Security target similar to the BSI accreditation
  - Many (but not all) artifacts shared with the BSI accreditation
    - x86-64 and ARM, but no secure boot
  - Close to being completed

Certification

- **L4Re ISO 26262 ASIL-B certification**
  - Safety requirements for automotive safety
    - Relying on informal requirements
    - Sufficient for controlling less critical systems (e.g. headlights, brake lights)
  - Requirements to follow a quality-managed development process (such as ASPICE) and to follow a coding standard (such as MISRA)
  - Certification artifacts
    - Safety case, high-level design, low-level design, 4 levels of functional requirements, safety analysis, hazard and operability study, dependency failure analysis, safety test specification
  - 100% line, function and branch coverage using unit tests
  - Completed via the *EB Corbos Hypervisor* (Elektrobit/Continental)

Formal Methods

- **Double-edged sword**
  - Mathematically-strong guarantees of the formally-verified properties under formally-specified proof assumptions
    - Much stronger than any degree of testing can ever provide
    - Highly appreciated by critical use cases
      - Although their integration into existing certification processes might not be so straightforward
  - False sense of guarantees when the proof assumptions cannot be always made to hold
    - Unless the proof assumptions are completely incorrect, the formal proofs still provide some conditional assurances
    - But the price might be unfavorable compared to informal methods
      - Tests, although non-exhaustive, actually inherently verify their own assumptions

Formal Methods

- **Current Kernkonzept approach**
  - Incremental steps
    - Specifying an abstract model and a meaningful separation property
    - Verifying compliance between the abstract model and the implementation
      - Model-based testing
      - Exhaustive comparison
    - Improving baseline guarantees (e.g. switching from C++ to Rust)
  - Proactive approach, but further steps to be determined by customer needs
    - Currently there seems to be more supply than demand
      - Extremely costly and time-consuming
      - Lack of automation in tooling

PRACTICAL MATTERS

101

Kernkonzept Practically

- **SME in the traditional sense**
  - Not a start-up, but a long-term sustainable business
    - Organic growth, no external investors
  - Almost flat hierarchy
    - Everybody has a reasonable awareness of what everybody else is doing
  - Pleasant working environment
    - No “big corporate BS”
      - No processes for the sake of processes
      - Do whatever it takes to get the job done
    - Meritocracy, technical challenges and self-learning
      - Budget for training, annual hackathon
    - Work/life balance

Working Remotely Practically

- **Great option, but not a silver bullet**
  - Ideal for certain life periods (e.g. having small children, requiring time flexibility)
  - Less ideal for other life periods (e.g. junior positions, developing a fast career)
  - Some job agenda more suitable than other
    - Works well for tasks with longer stretches of individual work and less frequent coordination (researching, coding, etc.)
    - Works less well for tasks with frequent and irregular coordination (people management, intense teamwork, etc.)
      - Modern technologies help
      - Face-to-face interaction still more efficient, with less friction and overhead

Working Remotely Practically

- **Personal tips**
  - Define and respect physical, temporal and virtual boundaries
    - Imagine actually going to/from the office every day (despite the commute time in seconds)
    - Procrastinating is easy, but working long after the “business hours” is even easier (workaholism is not uncommon)
    - Use separate user accounts (or even physical machines) for work and non-work
  - Make conscious decisions regarding the flexibility
    - Except for emergencies, always plan your “away from keyboard” moments ahead of time
    - Let everyone (especially children) understand that time borrowed during “business hours” needs to be repaid during “free hours”
      - Work/life balance should not turn into work/life inbalance

OUTRO

110

Outro

- **Kernkonzept is successful in …**
  - … developing the microkernel-based L4Re in the mission-critical & safety-critical industrial context
  - … balancing pragmatic use cases and research
  - … achieving certification goals
  - … supporting formal verification efforts
  - … being a significant part of the community
  - … improving the state-of-the-art via proper software architecture and engineering
- **Kernkonzept is open for collaboration**
  - Assignments, theses, internships, jobs
  - Research, EU projects

KERNKONZEPT

THANK YOU

Questions?

KERNKONZEPT

CONTACT US

www.kernkonzept.com
info@kernkonzept.com

BACKUP SLIDES

111

Microkernel Overhead

- **A.k.a. the unfounded anxiety that refuses to die**
  - Liedtke has shown 29 years ago that the overhead is **negligeable** (assuming proper microkernel design)
  - Bershad has argued 32 years ago that the IPC overhead is **increasingly irrelevant** (since the real-world performance of computer systems is dominated by other factors)
  - The market share of monolithic operating systems is hardly caused by the lack of IPC overhead alone
    - The market share of Coca Cola is hardly caused by the taste alone
- **Our customers simply “do not care about the overhead”**
  - The overall performance of L4Re is satisfactory to them
  - Whatever measurable overhead is there, it is considered a **reasonable price** for the run-time component isolation and the safety/security guarantees that are fundamentally not available in monolithic operating systems
  - The typical deployment of L4Re does not need extremely fine-grained components