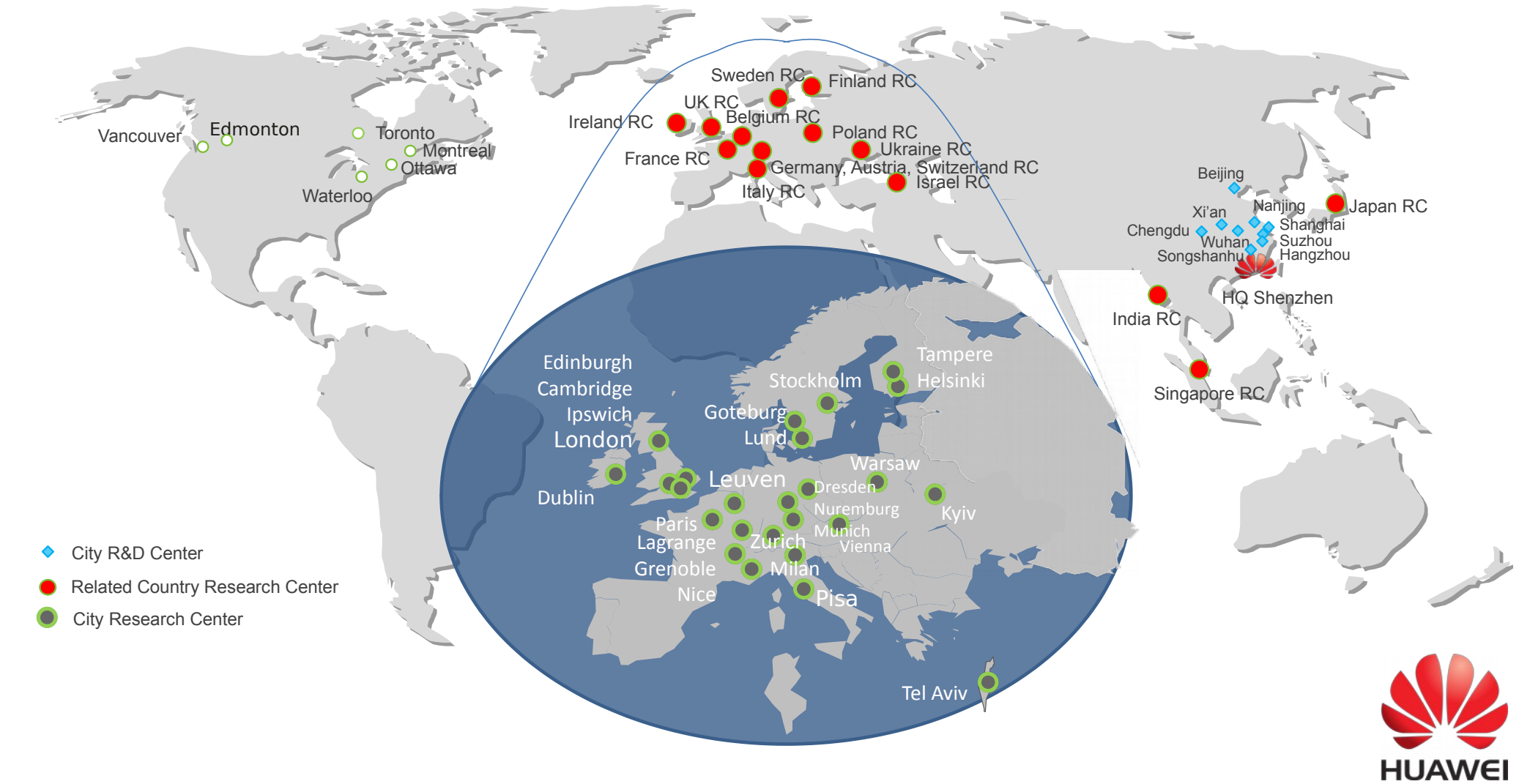
MICROKERNEL-BASED AND CAPABILITY-BASED OPERATING SYSTEMS

Martin Děcký
martin.decky@huawei.com

March 2021

About the Speaker

- **Charles University**
  - Research scientist at D3S (2008 – 2017)
  - Graduated (Ph.D.) in 2015
  - Co-author of the HelenOS (http://www.helenos.org/) microkernel multiserver operating system
- **Huawei Technologies**
  - Senior Research Engineer, Munich Research Center (2017 – 2018)
  - Principal Research Engineer, Dresden Research Center (2019 – present)


Vancouver
Edmonton
Toronto
Montreal
Ottawa
Waterloo
Sweden RC
Finland RC
UK RC
Ireland RC
Belgium RC
Poland RC
Ukraine RC
France RC
Germany, Austria, Switzerland RC
Israel RC
Italy RC
Beijing
Xi'an
Nanjing
Japan RC
Chengdu
Shanghai
Wuhan
Suzhou
Songshanhu
Hangzhou
HQ Shenzhen
India RC
Singapore RC
Edinburgh
Cambridge
Ipswich
London
Dublin
Stockholm
Tampere
Helsinki
Goteburg
Lund
Leuven
Warsaw
Dresden
Nuremburg
Munich
Vienna
Kyiv
Paris
Lagrange
Zurich
Grenoble
Milan
Nice
Pisa
Tel Aviv
City R&D Center
Related Country Research Center
City Research Center
HUAWEI

Huawei Dresden Research Center (DRC)

- Since 2019, ~20 employees (plus a virtualization team in Munich)

HUAWEI

Huawei Dresden Research Center (DRC) (2)

- **Focuses on R&D in the domain of operating systems**
  - Microkernels, hypervisors
    - Collaboration with the *OS Kernel Lab* in Huawei HQ
    - Collaboration with TU Dresden, MPI-SWS, ETH Zürich and other institutions
  - Formal verification of correctness, weak memory architectures
  - Safety and security certification
  - Many-core scalability, heterogeneous hardware
  - Flexible OS architecture

HUAWEI

We Are Hiring

- **Operating System Engineer / Researcher** (Dresden)
  - https://apply.workable.com/huawei-16/j/3BAC3458E6/
- **Formal Verification Engineer / Researcher** (Dresden)
  - https://apply.workable.com/huawei-16/j/95CCAD4EC5/
- **Virtualization Engineer / Researcher** (Munich)
  - https://apply.workable.com/huawei-16/j/51F90678EA/
- **Industrial Ph.D. Student** (Dresden)
  - In collaboration with TU Dresden

HUAWEI

Systems Software Innovations Summit 2021

- **March 30th – 31st 2021**
  - https://huawei-events.de/, on-line, no participation fee


Greg Kroah-Hartman
Linux Foundation


Arnd Bergmann
Linaro


Jules Villard
Facebook


Diogo Behrens
Huawei


Olaf Spinczyk
Osnabrück University


June Andronick
University of New South Wales (UNSW)


Ding Yuan
University of Toronto


Jeronimo Castrillon
Dresden University of Technology


Timothy Roscoe
ETH Zurich


Viktor Vafeiadis
Max Planck Institute for Software Systems (MPI-SWS)


Jan Reineke
Saarland University


Hermann Härtig
Dresden University of Technology


Sanidhya Kashyap
École Polytechn. Fédérale de Lausanne (EPFL)


Ronghui Gu
Columbia University


Vasily A. Sartakov
Imperial College London

HUAWEI

MICROKERNELS

Microkernel-based Operating Systems

- **Motivation**
  - Safety, security, reliability, dependability
    - Proper software architecture
    - Formal verification of correctness
  - Modularity, customization
  - Virtualization, paravirtualization
    - Tasks and virtual machines are quite similar types of entities
  - Partitioning, support for mixed criticality

HUAWEI

Monolithic OS Design Is Flawed

- **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."*
    - https://dl.acm.org/doi/10.1145/3265723.3265733

HUAWEI

Some Data Points from History

- **Compatible Time-Sharing System (CTSS)**
  - John McCarthy, MIT Computation Center, 1961
    - Probably one of the earliest “real” operating system
      - Not just a loader, jobs manager or batch manager
- **RC 4000 Multiprogramming System**
  - Per Brinch Hansen, Regnecentralen, 1969
    - Separation of mechanism and policy, modularity via isolated concurrently running processes, message passing
- **Multics**
  - MIT, General Electric, Bell Labs, 1969
    - Traceable influence on UNIX

HUAWEI

Some Data Points from History (2)

- **HYDRA**
  - William Wulf, Carnegie Mellon University, 1971
    - Capability-based, object-oriented, separation of mechanism and policy
    - Probably the earliest peer-reviewed publication of the design principles
- **UNIX**
  - Ken Thompson, Dennis Ritchie, Brian Kernighan et al., Bell Labs, 1973
    - Architecture and design traceable in many current monolithic systems
- **VMS**
  - Digital Equipment, 1977
    - Architecture and design traceable in Microsoft Windows

HUAWEI

Some Data Points from History (3)

- **EUMEL / L2**
  - Jochen Liedtke, University of Bielefeld, 1979
    - Proto-microkernel based on bitcode virtual machines
- **QNX**
  - Gordon Bell, Dan Dodge, 1982
    - Earliest commercially successful microkernel multiserver OS
      - Still in active use and development today
- **CMU Mach**
  - Richard Rashid, Avie Tevanian, Carnegie Mellon University, 1985
    - Arguably the most widespread microkernel code base
      - Still a core part of macOS, iOS and other OS clones by Apple today (but not in a microkernel configuration)
      - Despite its well-publicized shortcomings, it remains highly influential

HUAWEI

Microkernel-based Operating Systems

- **Definition**
  - Operating system that follows specific design principles that, in effect, minimize the amount of code running in the privileged (kernel) mode
    - Hence the name
  - Every microkernel-based OS follows slightly different specific design principles
    - Two design principles are probably universally common
      - Minimality principle
      - Split of mechanism and policy principle

Minimality Principle

- **The obvious criterion**
  - The kernel needs to implement the functionality than cannot be possibly implemented in user space
    - On typical commodity hardware, this includes
      - Bootstrapping
      - Fundamental part of hardware exception and interrupt handling
      - Configuration of certain control registers (possibly including MMU)
      - Fundamental part of mode switching (e.g. related to hardware virtualization, trusted execution environments, etc.)

Minimality Principle (2)

- **The necessary criterion**
  - The kernel needs to implement the functionality than cannot be delegated only to a trusted user space component without also delegating it to any untrusted user space component (thus undermining the fundamental guarantees that the operating system provides)
    - On typical commodity hardware, this includes
      - Configuration of the forced preemption mechanism (e.g. timer interrupt routing)
      - Fundamental part of interacting with a hypervisor, firmware and some hardware components
        - Hardware components are tricky: Without IOMMU, almost any interaction with hardware might potentially undermine the OS guarantees

Minimality Principle (3)

- **The practicality criterion**
  - The kernel **might** also implement the functionality that would be unpractical (while still technically possible) to be safely delegated to user space
    - This is where microkernels differ, but there are still some universal examples
      - Context switching
      - Basic scheduling
      - System timer configuration
      - Observability and (optional) debugging support

Split of Mechanism and Policy Principle

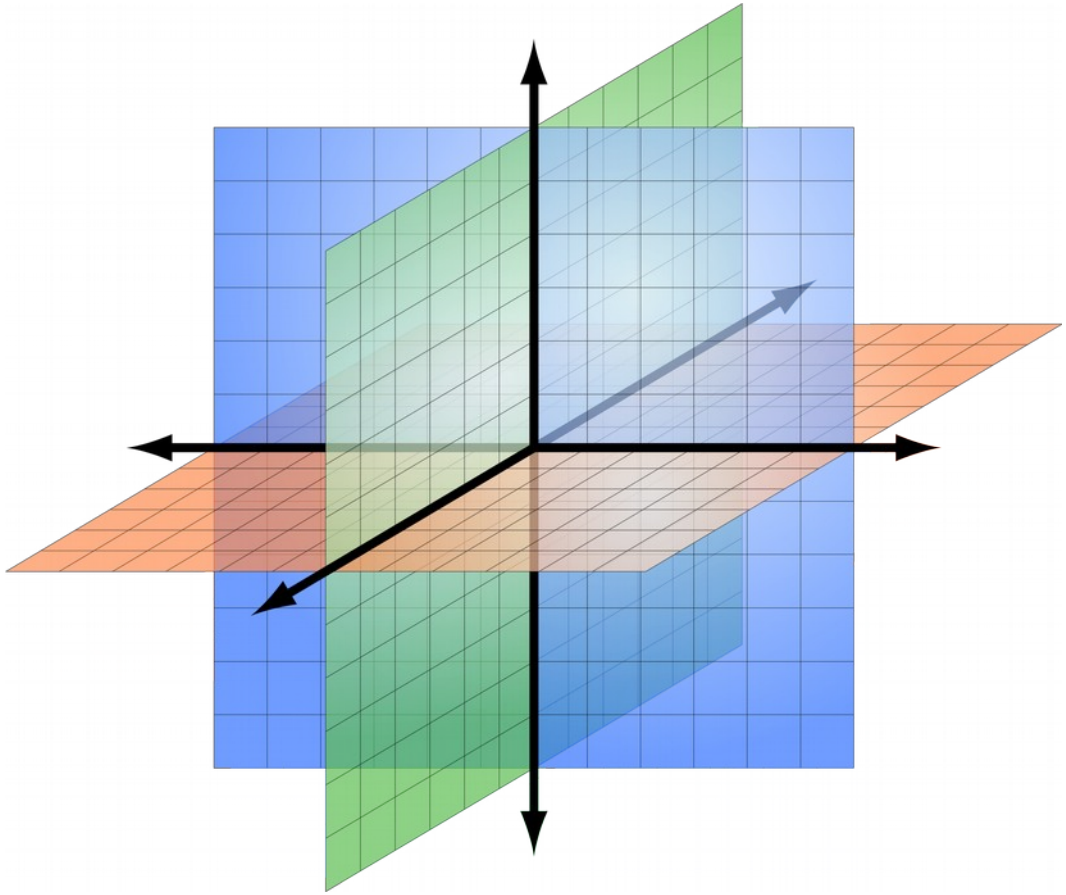
- **Orthogonal to the minimality principle**
  - The microkernel is not an indivisible entity
    - Composed of instructions, basic blocks, language constructs, etc.
    - The code inevitably follows some patterns that form architecture, design, abstractions, parametrization, etc.
  - Separation of concerns
    - The kernel implements only pure and universal mechanisms (“the what”) while the policies (“the how/when”) are delegated to user space
      - This is where microkernels differ
        - Does “arbitrary policy” equal “no policy”?
        - Is it fine to have a default (but replaceable) policy?

HUAWEI

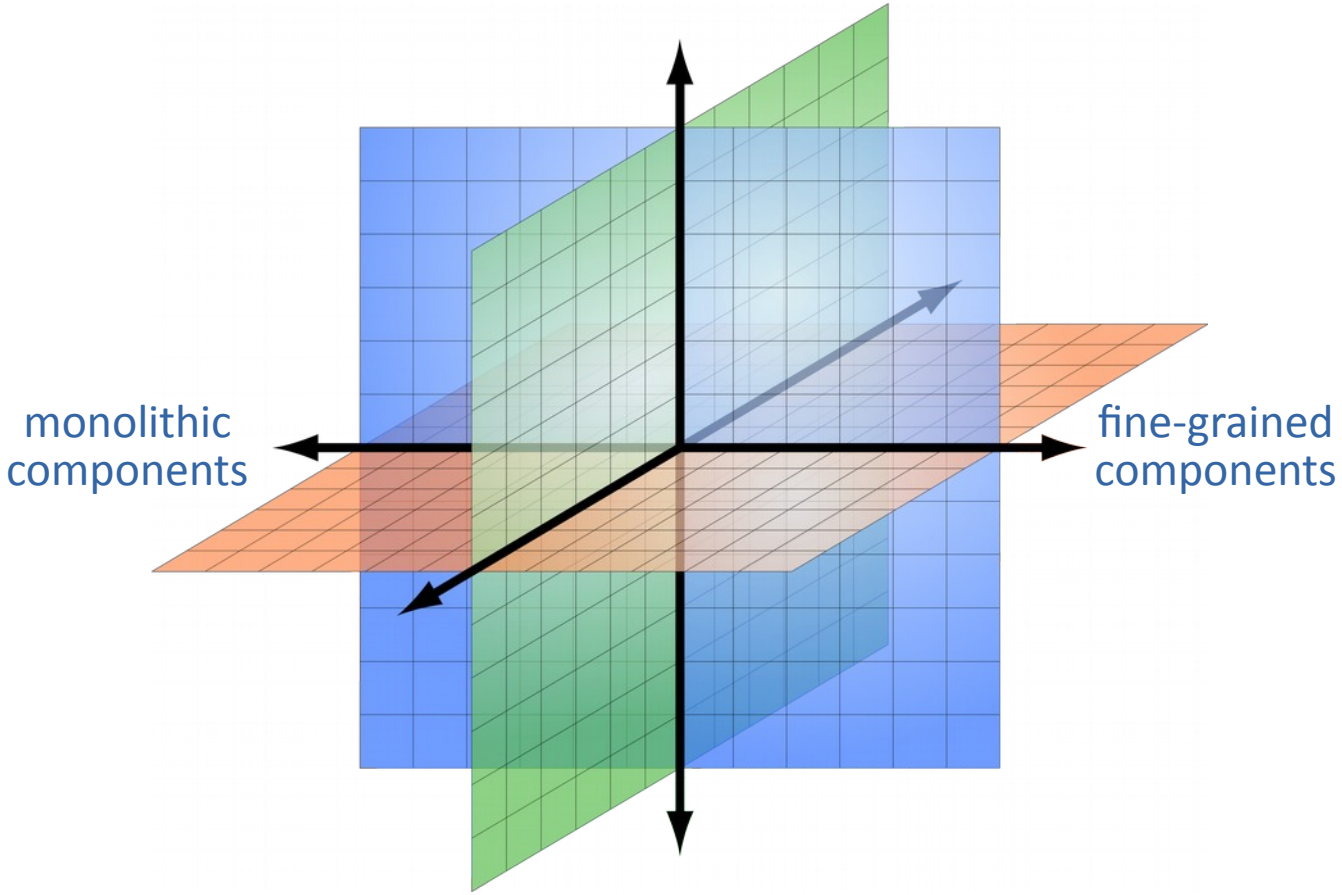
Practical Differences

- **Monolithic kernel**
  - Configurability via compile-time options and parametrization
  - Modularity via run-time dynamic linking
  - Tight module coupling, weak module cohesion
  - Structure is implicit and not enforced (especially at run time)
- **Microkernel**
  - Configurability via different use (policy in user space)
  - Modularity via extension in user space
  - Loose module coupling, strong module cohesion
  - Structure is explicit and enforced (even at run time)

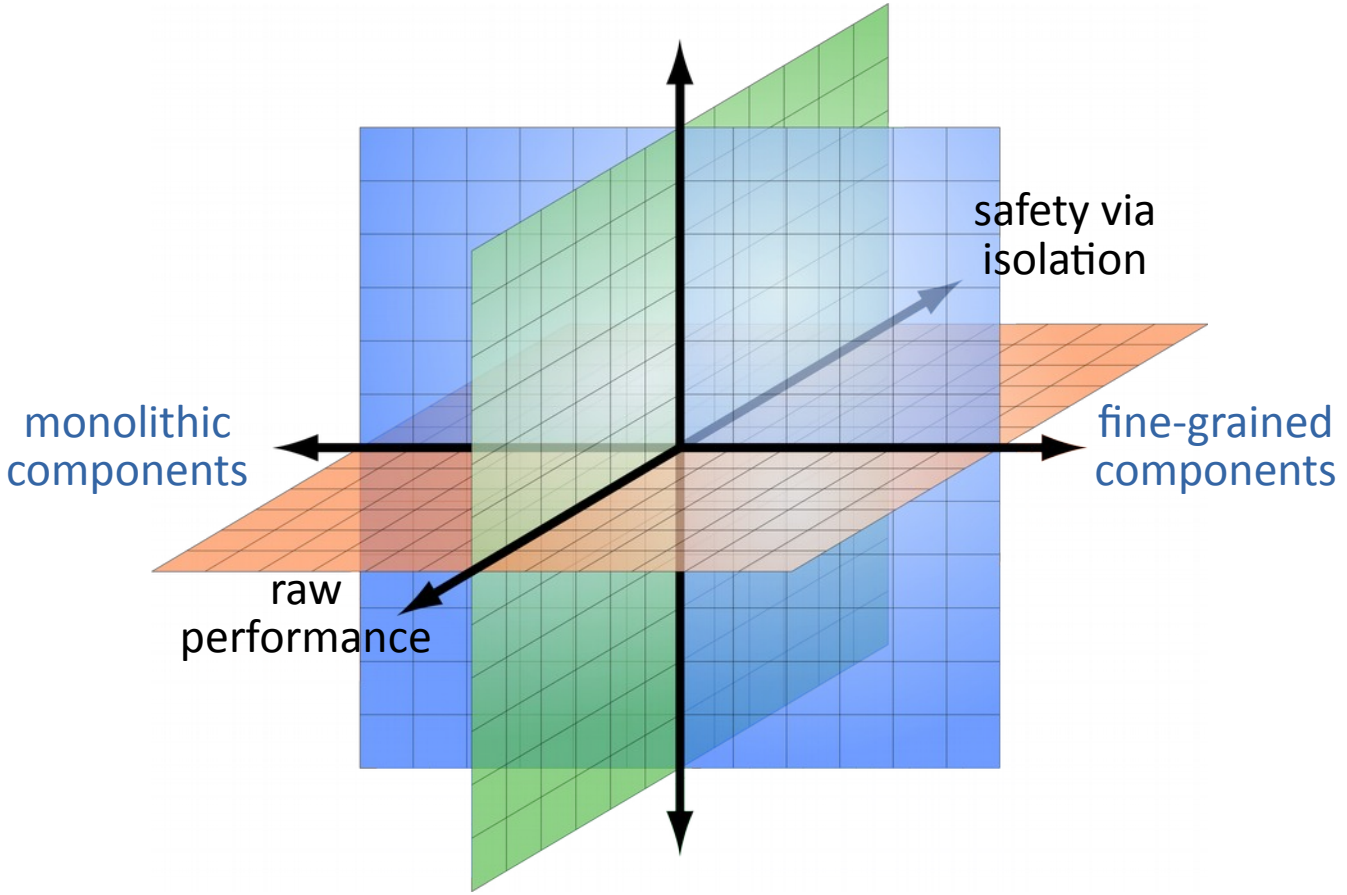
Design Space of Operating Systems



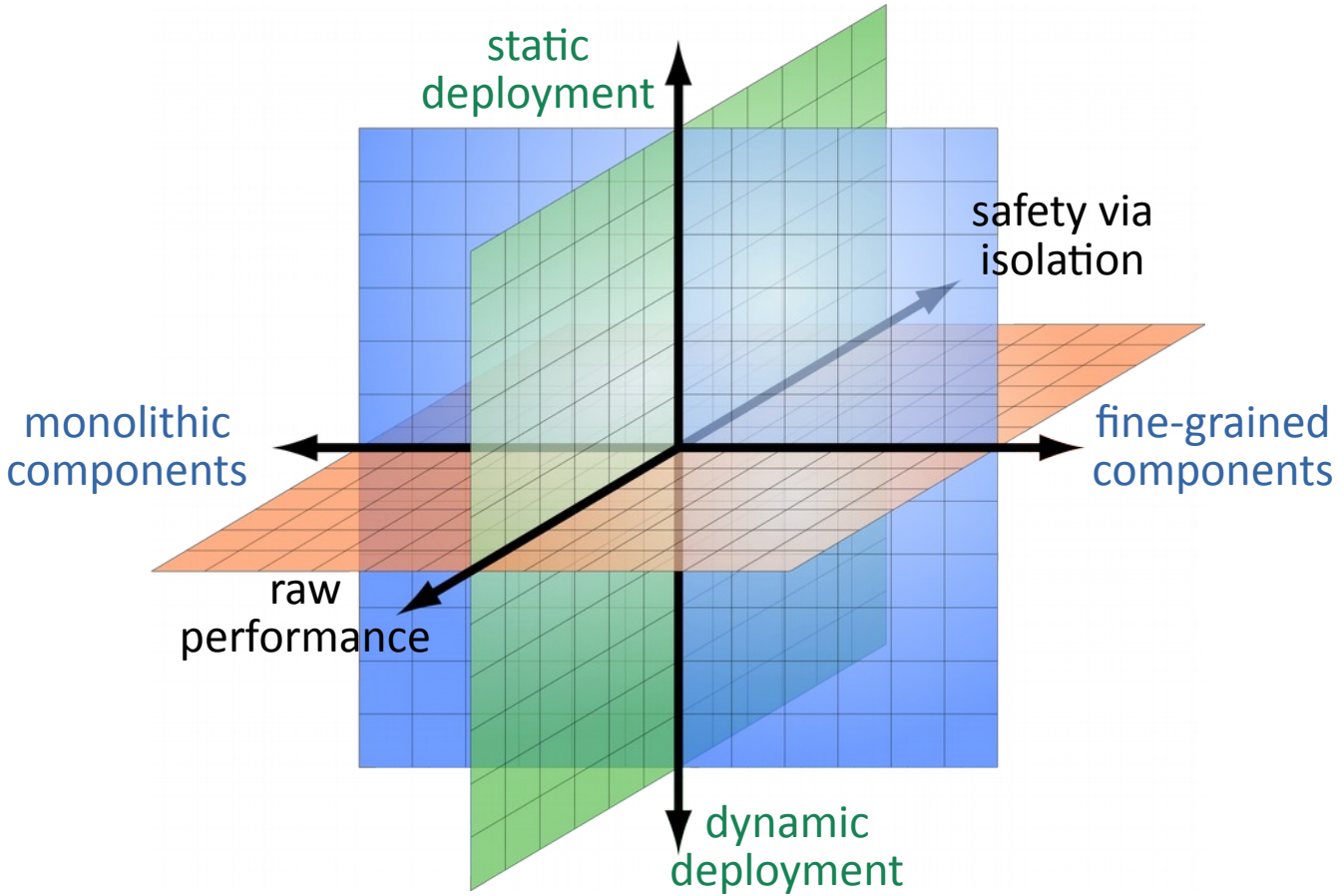
Design Space of Operating Systems



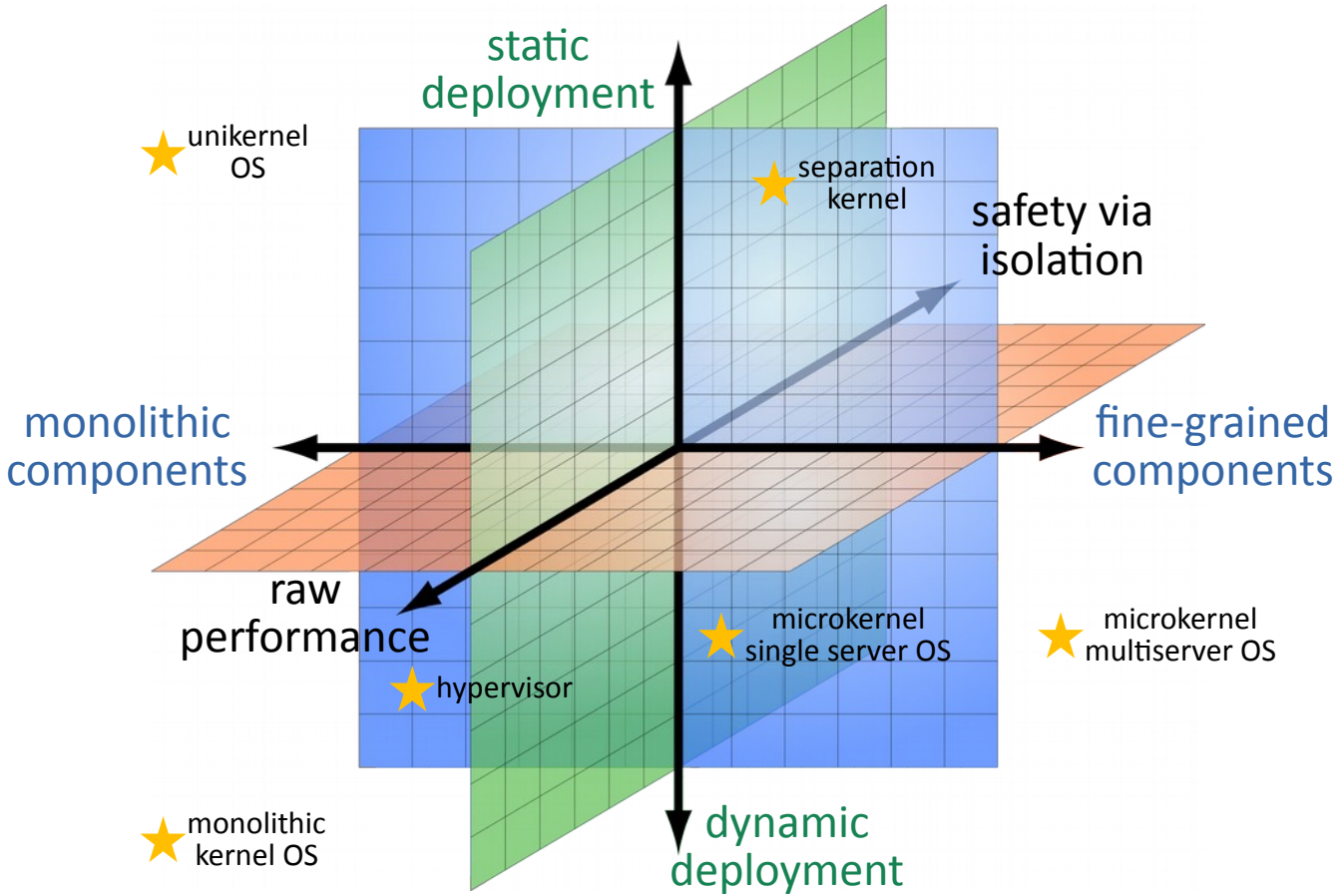
Design Space of Operating Systems



Design Space of Operating Systems

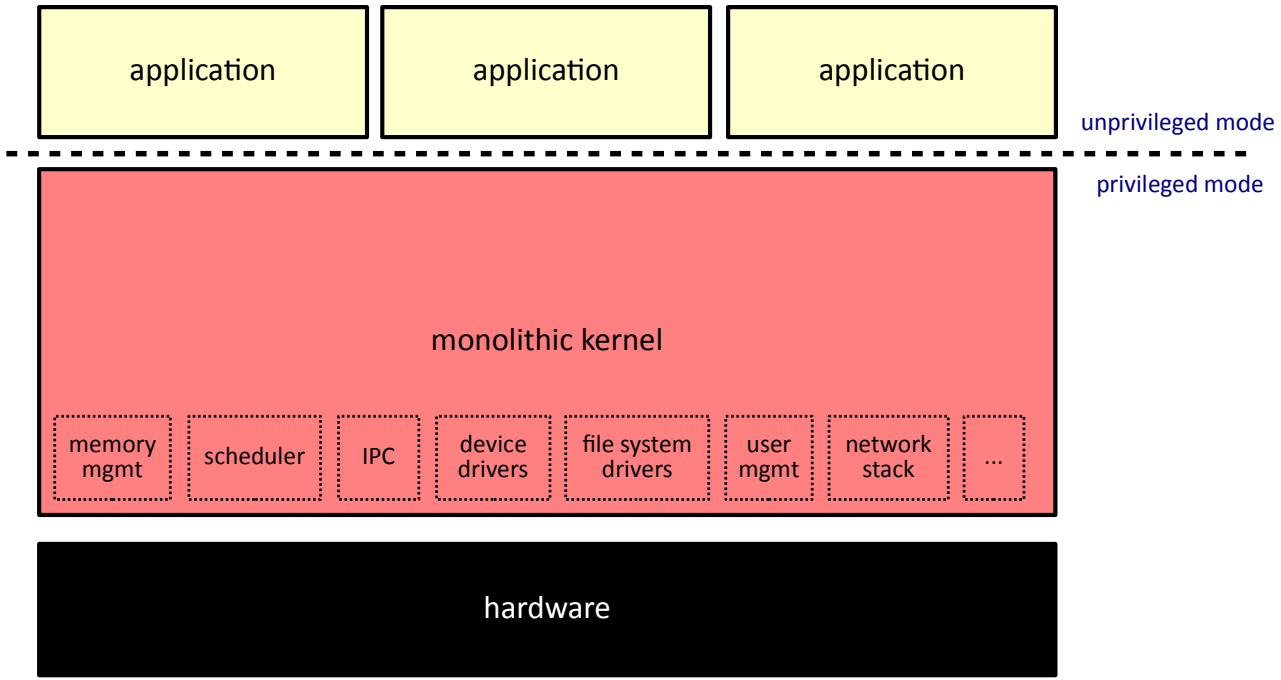


Design Space of Operating Systems



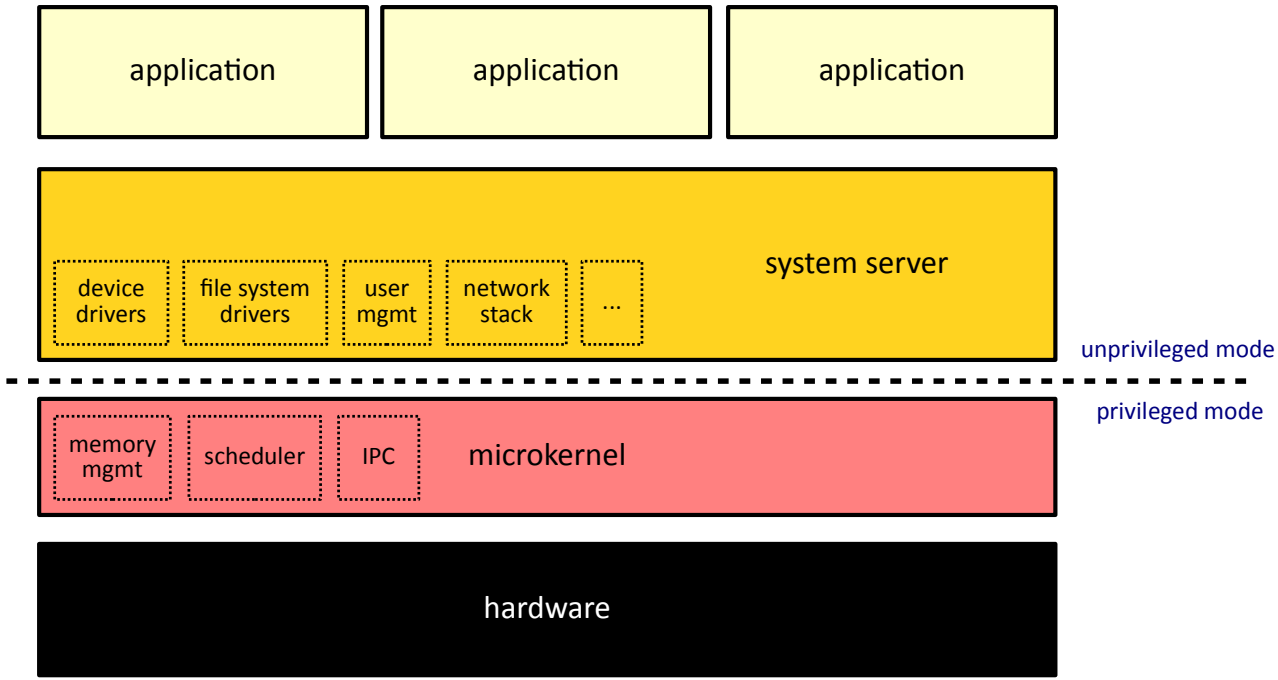
Architecture of User Space

- **Monolithic OS**



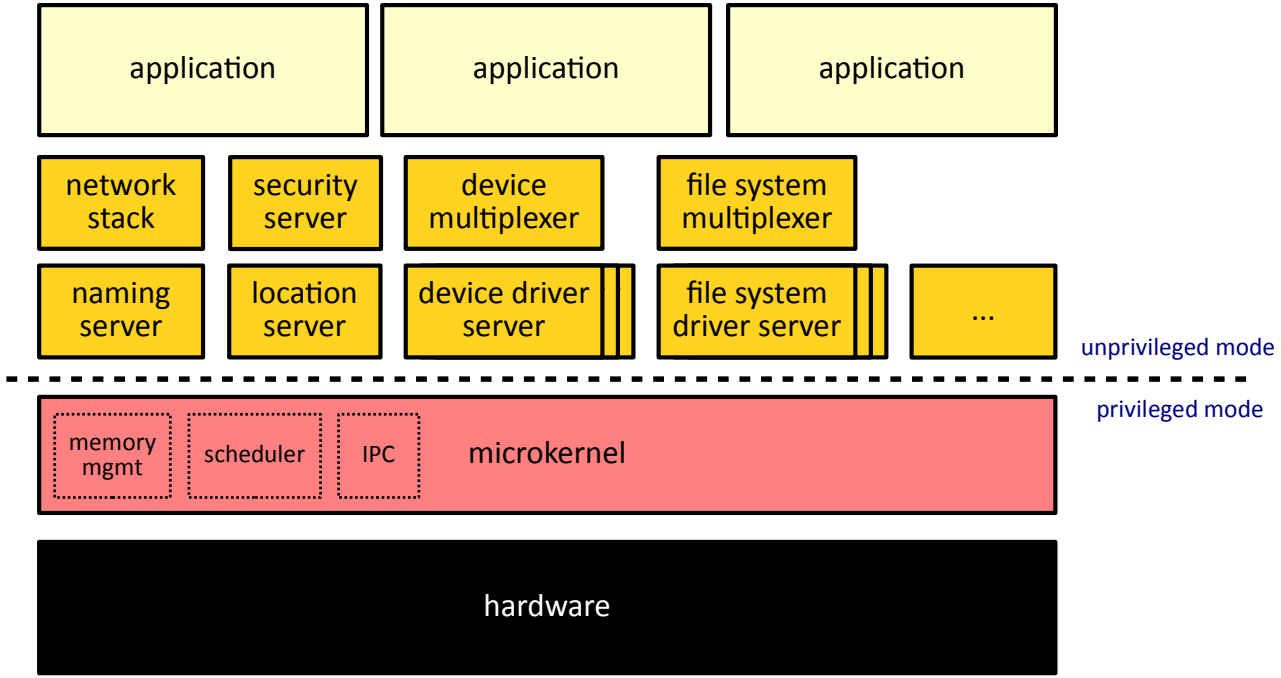
Architecture of User Space

- **Single-server microkernel OS**



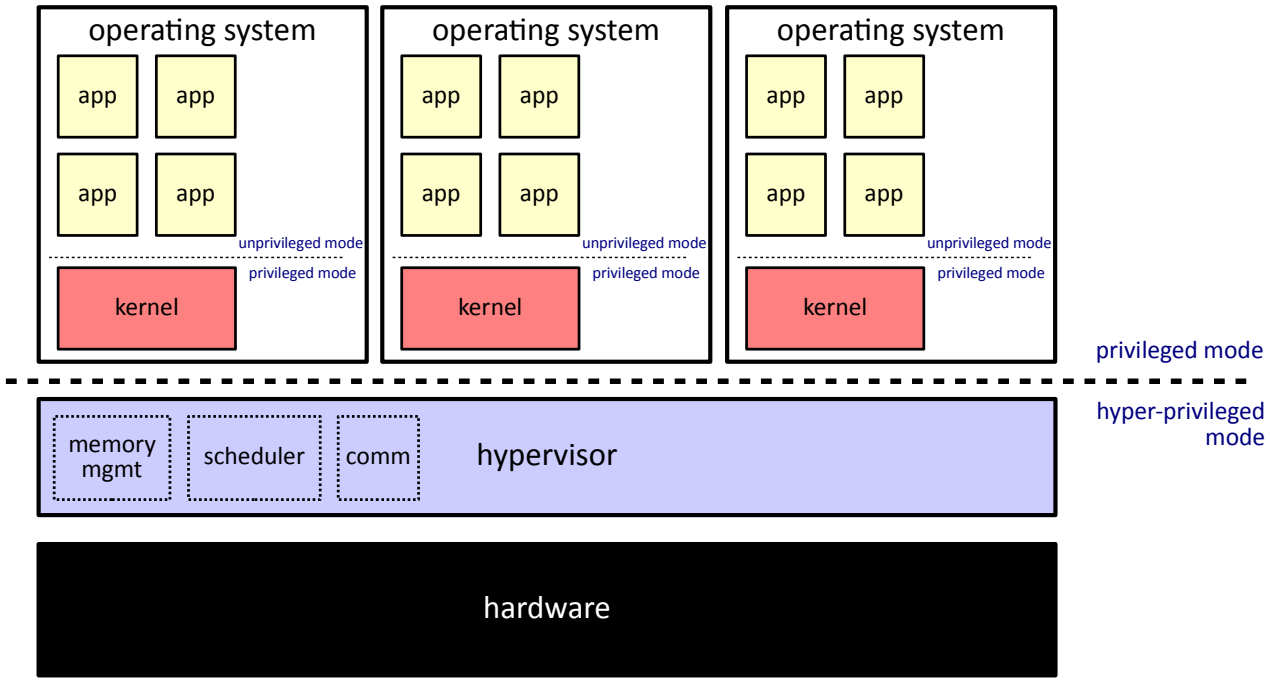
Architecture of User Space

- **Multiserver microkernel OS**



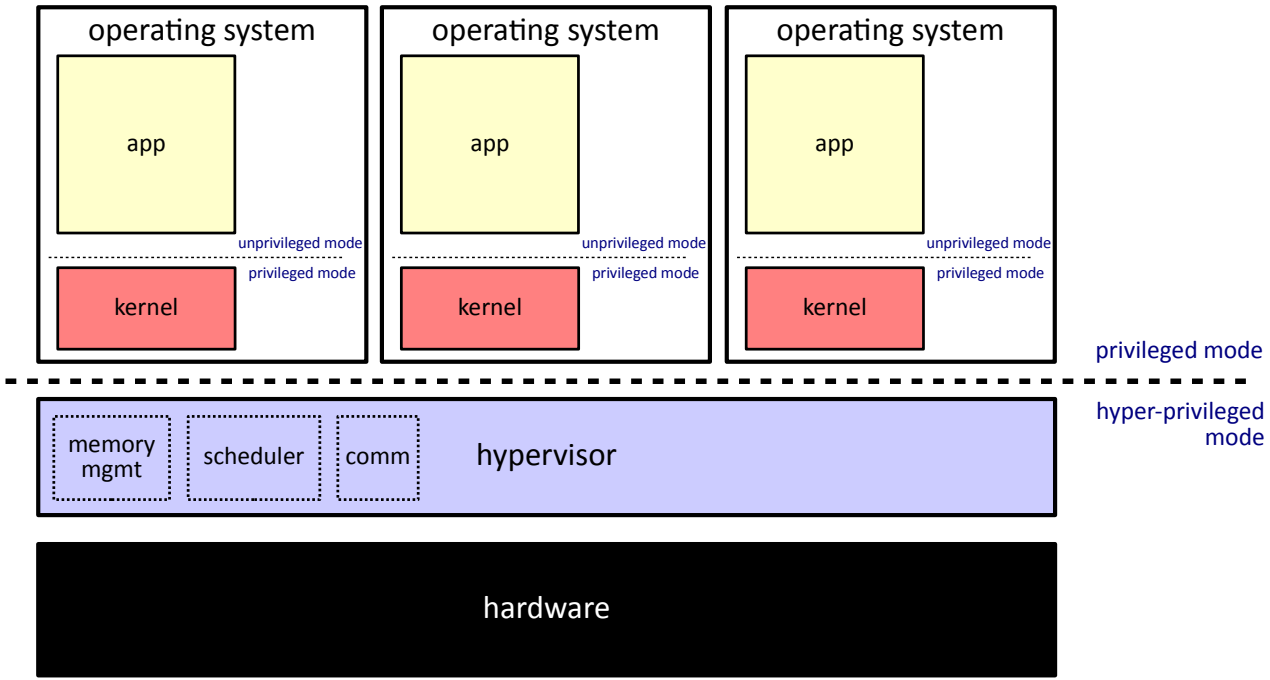
Architecture of User Space

- **Type-1 hypervisor**



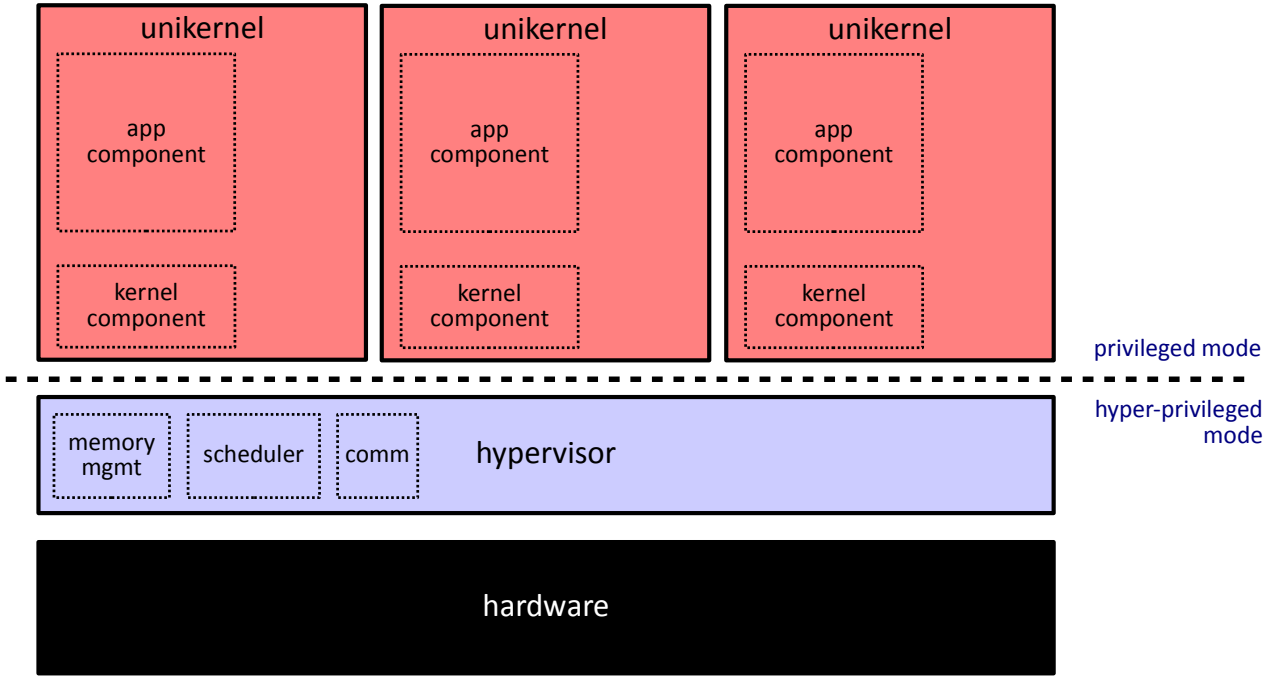
Architecture of User Space

- **Type-1 hypervisor** (in common deployment)



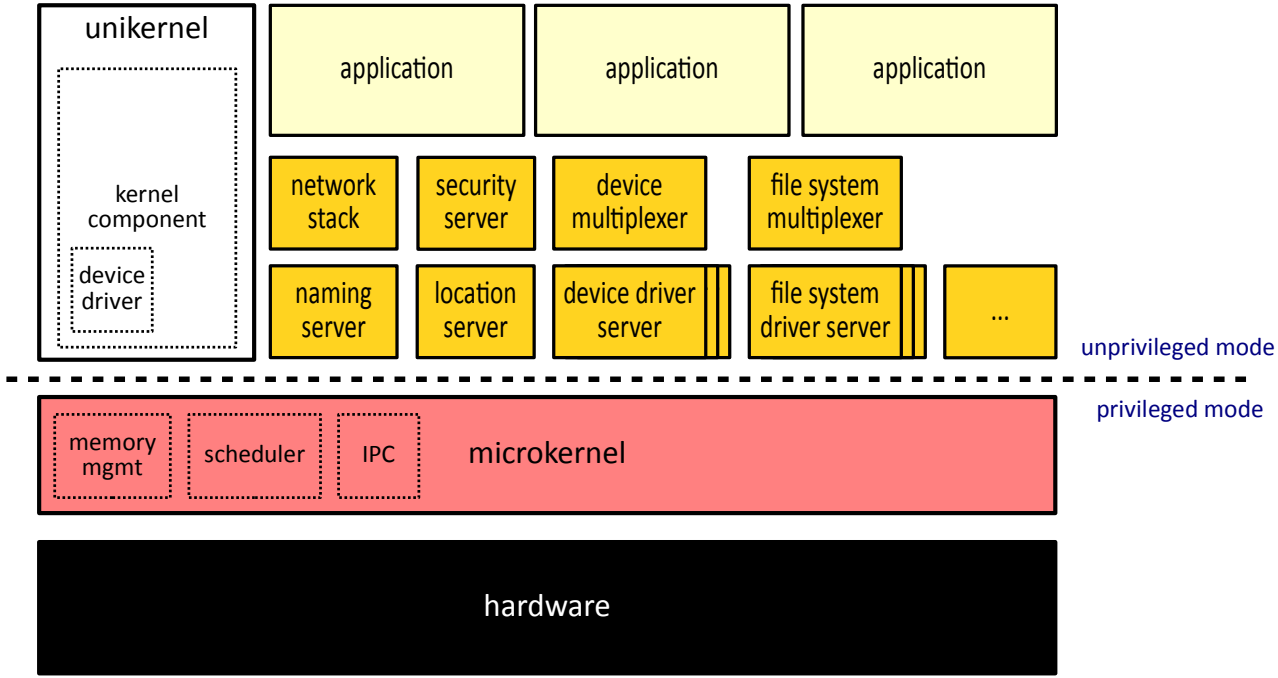
Architecture of User Space

- **Hypervisor with unikernels**



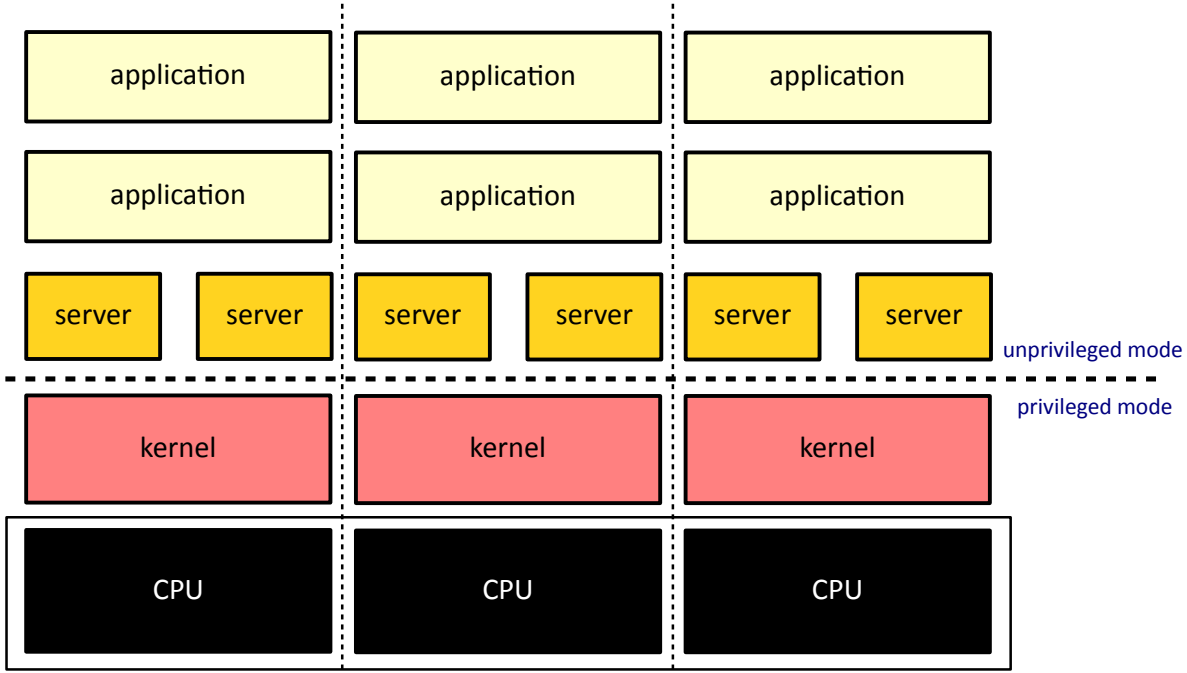
Architecture of User Space

- **Multiserver microkernel with unikernels for device drivers**



Architecture of User Space

- **Multikernel**



CAPABILITIES

Capabilities

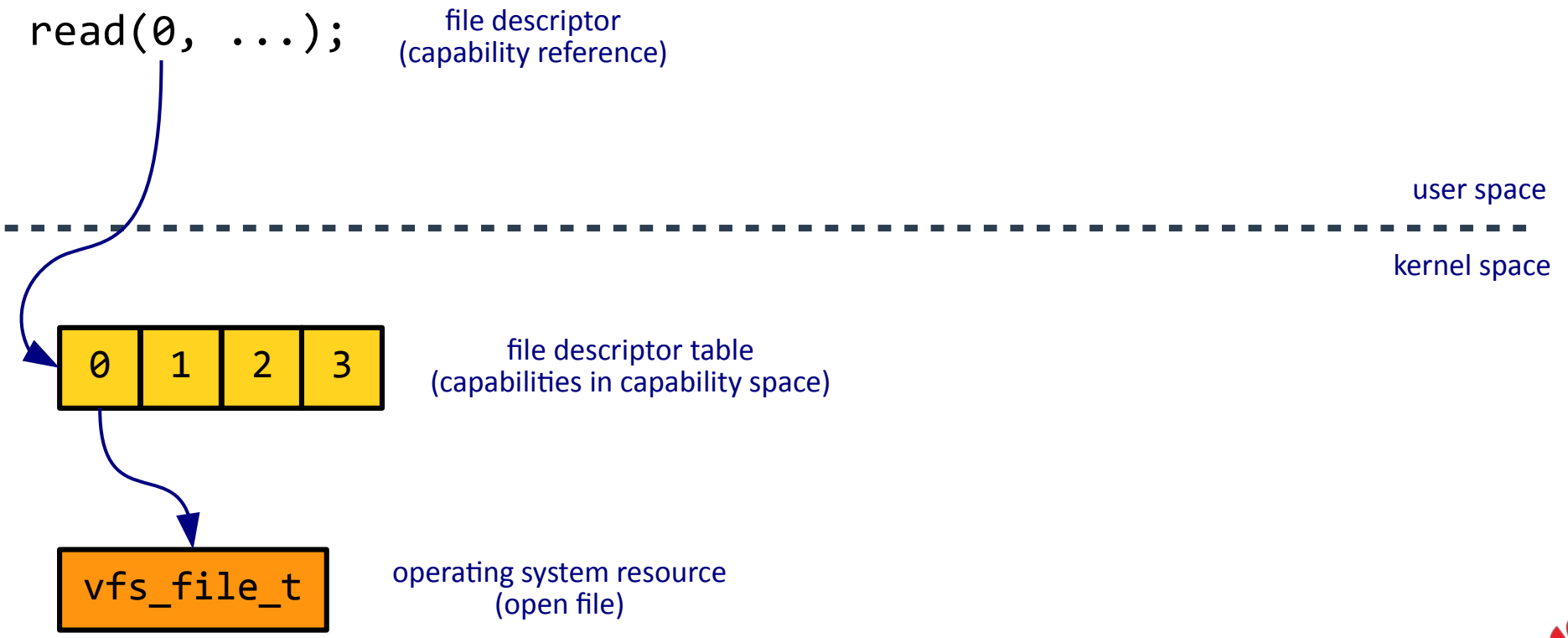
- **Motivation**
  - A **universal** and **pure** mechanism in the kernel to safely manage (all) operating system resources
    - Without implementing any specific management policy in the kernel (i.e. delegating the management policy completely to user space)
  - Potential secondary goal
    - Possibility to grant or delegate (parts of) the authority over resources from the original owner of a resource to other users
      - In a controllable fashion (i.e. including the possibility of revocation)

HUAWEI

Capabilities

- **Definition**
  - **Capability**
    - Object (instance of a given object type) identifying some specific (operating system) resource
      - Kernel object identifying a kernel-managed resource
      - Kernel object (proxy object) identifying a user space resource
      - User space object identifying a user space resource
  - **Capability reference**
    - Unforgeable identifier (handle) to a capability
      - Might be associated with permissions (e.g. permissible operations, methods) and ownership
  - **Capability space**
    - Each capability reference is local to a specific namespace (typically associated with a specific task, process) and does not have any meaning in other namespaces
      - Akin to (virtual) address space

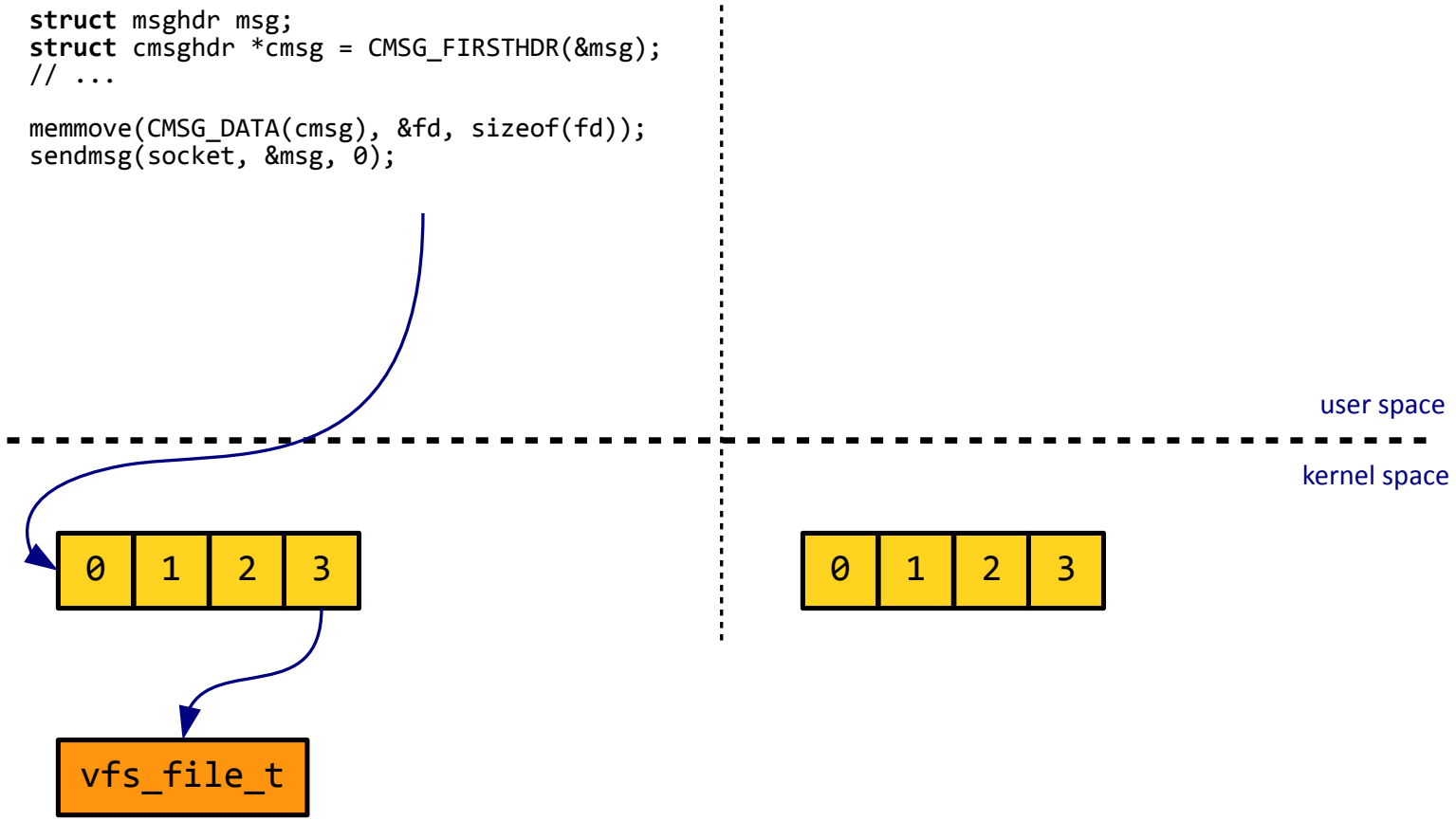
What Are Capabilities, Anyway?



Capability Granting

```
struct msghdr msg;
struct cmsghdr *cmsg = CMSG_FIRSTHDR(&msg);
// ...

memmove(CMSG_DATA(cmsg), &fd, sizeof(fd));
sendmsg(socket, &msg, 0);
```



Capability Granting

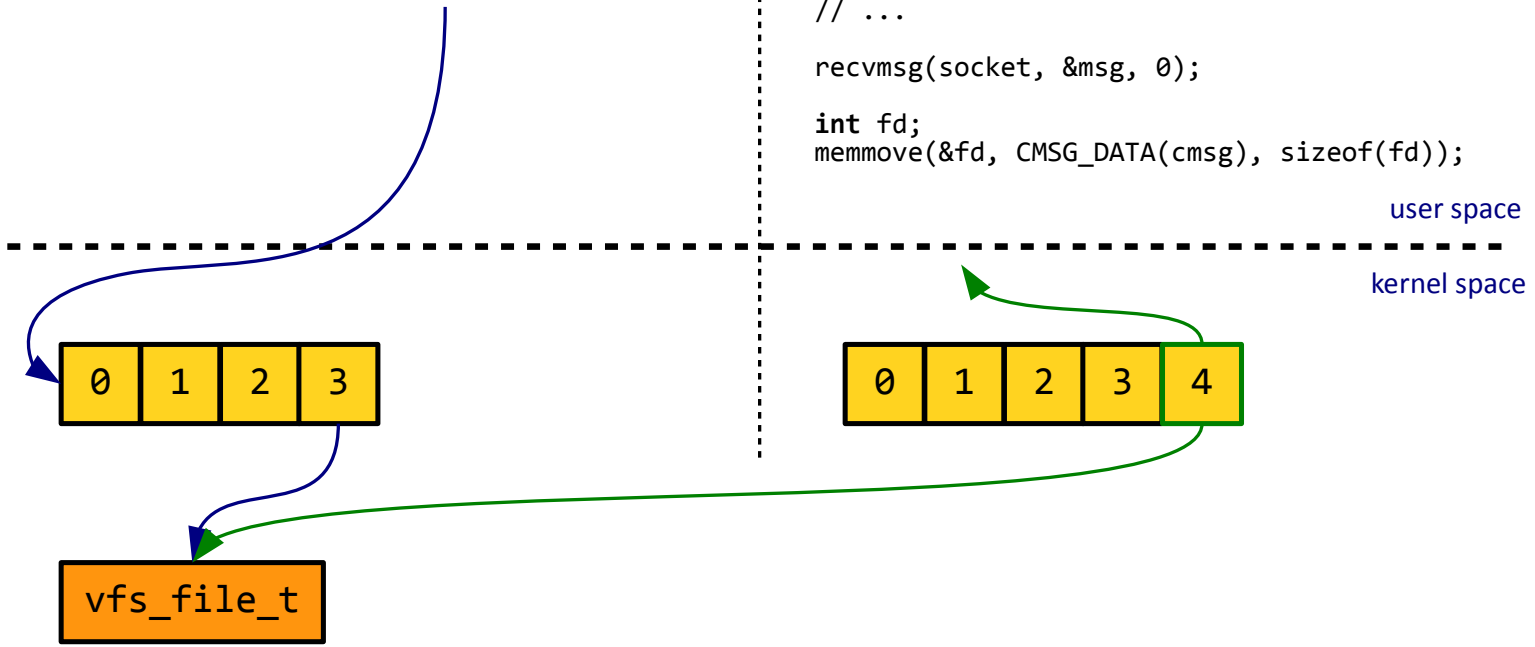
```
struct msghdr msg;
struct cmsghdr *cmsg = CMSG_FIRSTHDR(&msg);
// ...

memmove(CMSG_DATA(cmsg), &fd, sizeof(fd));
sendmsg(socket, &msg, 0);
```

```
struct msghdr msg;
struct cmsghdr *cmsg = CMSG_FIRSTHDR(&msg);
// ...

recvmsg(socket, &msg, 0);

int fd;
memmove(&fd, CMSG_DATA(cmsg), sizeof(fd));
```



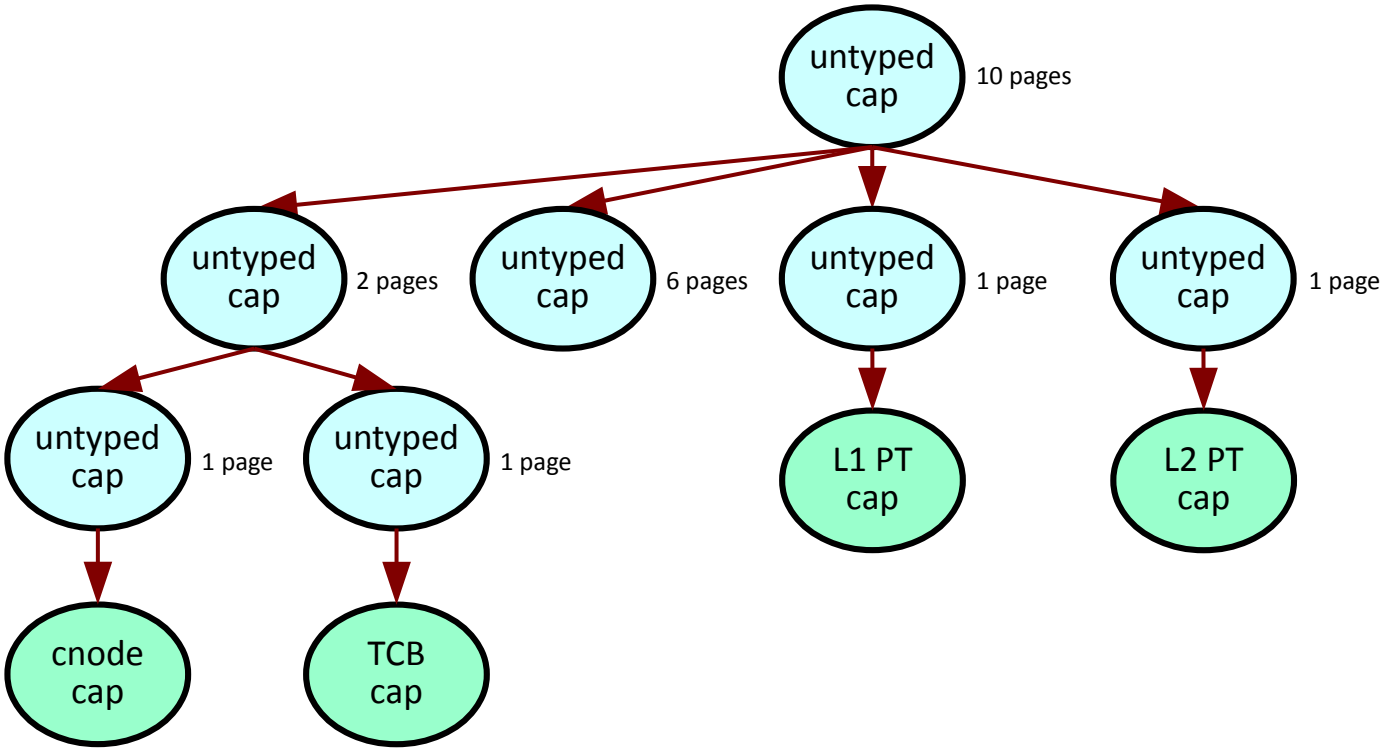
Chicken & Egg Problem

- **What if we want to represent all resources as capabilities?**
  - Even the resource (memory) needed to store the capabilities and capability references is a capability
    - We start with some basic capability (untyped capability) that represents (physical) memory
      - Encapsulated capability vs. naked capability
      - This capability can be retyped to a different capability or converted to multiple capabilities
        - Allocating kernel objects
        - Allocating capability nodes that bind capability references to capabilities
      - Bookkeeping objects (e.g. memory for page tables) might also be represented as capabilities

HUAWEI

Capability Derivation Tree

- **Permissible ways of retyping capabilities**

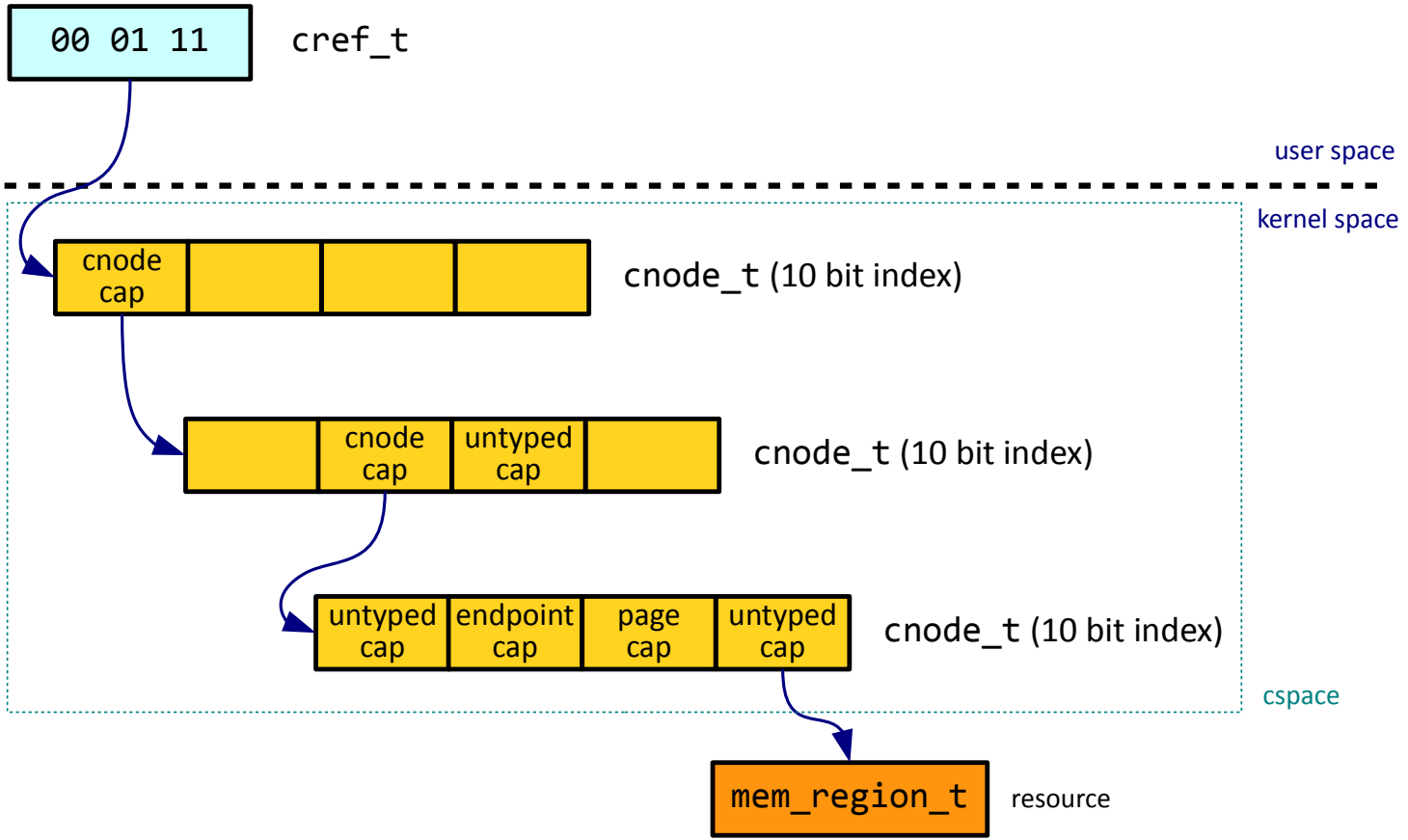


Representing Capability Space

- **Effective and efficient storage for capability nodes**
  - Criteria
    - Low memory overhead and fragmentation even for sparse capability spaces
    - Fast lookup of capability references (typically the most frequent operation)
    - Reasonably fast creation and removal of new capability references
    - Possibility to store metadata (e.g. permissions, ownership/delegations) and even actual kernel objects (up to a certain size) in-line
  - Typical candidates
    - Arrays
    - Hash tables
    - Radix trees

HUAWEI

Hierarchical Capability Space



Capability Operations

- **Actions that can be performed with capabilities**
  - The permissible set of operations might be defined/restricted by the capability reference itself
    - Each capability reference might permit different methods despite pointing to the same object
  - **Invoke**
    - Executing some “business logic” operation on the target object
  - **Clone**
    - Creating a duplicate capability reference
  - **Mint**
    - Creating a duplicate capability reference, but with restricted permissions

Capability Operations (2)

- **Derive**
  - Retyping the capability to a different capability type or converting it to multiple capabilities
    - Permissible retyping/conversions defined by the capability derivation tree
- **Delegate**
  - Passing the ownership of the capability reference to different capability space
- **Grant**
  - Creating a duplicate capability reference (possibly with restricted permissions) in a different capability space (while keeping ownership)
  - Might be done only once or recursively
- **Revoke**
  - Removing a granted capability reference from a different capability space

GET TO KNOW MICROKERNELS

Microkernels - The componen

Not secure | microkernel.info

µ–kernel.info

Microkernels are operating systems that outsource the traditional operating system functionality to ordinary user processes while providing them with mechanisms requisite for implementing it. Microkernel-based operating systems come in many different flavours, each having a distinctive set of goals, features and approaches. Some of the most often cited reasons for structuring the system as a microkernel is flexibility, security and fault tolerance. Many microkernels can take on the role of a hypervisor too. Microkernels and their user environments are most often implemented in the C or C++ programming languages with a little bit of assembly, but other implementation languages are possible too. In fact, each component of a microkernel-based system can be implemented in a different programming language.

Here is a list of active free, open source microkernel projects. If your project is missing or this page needs fixing, please create a pull request!

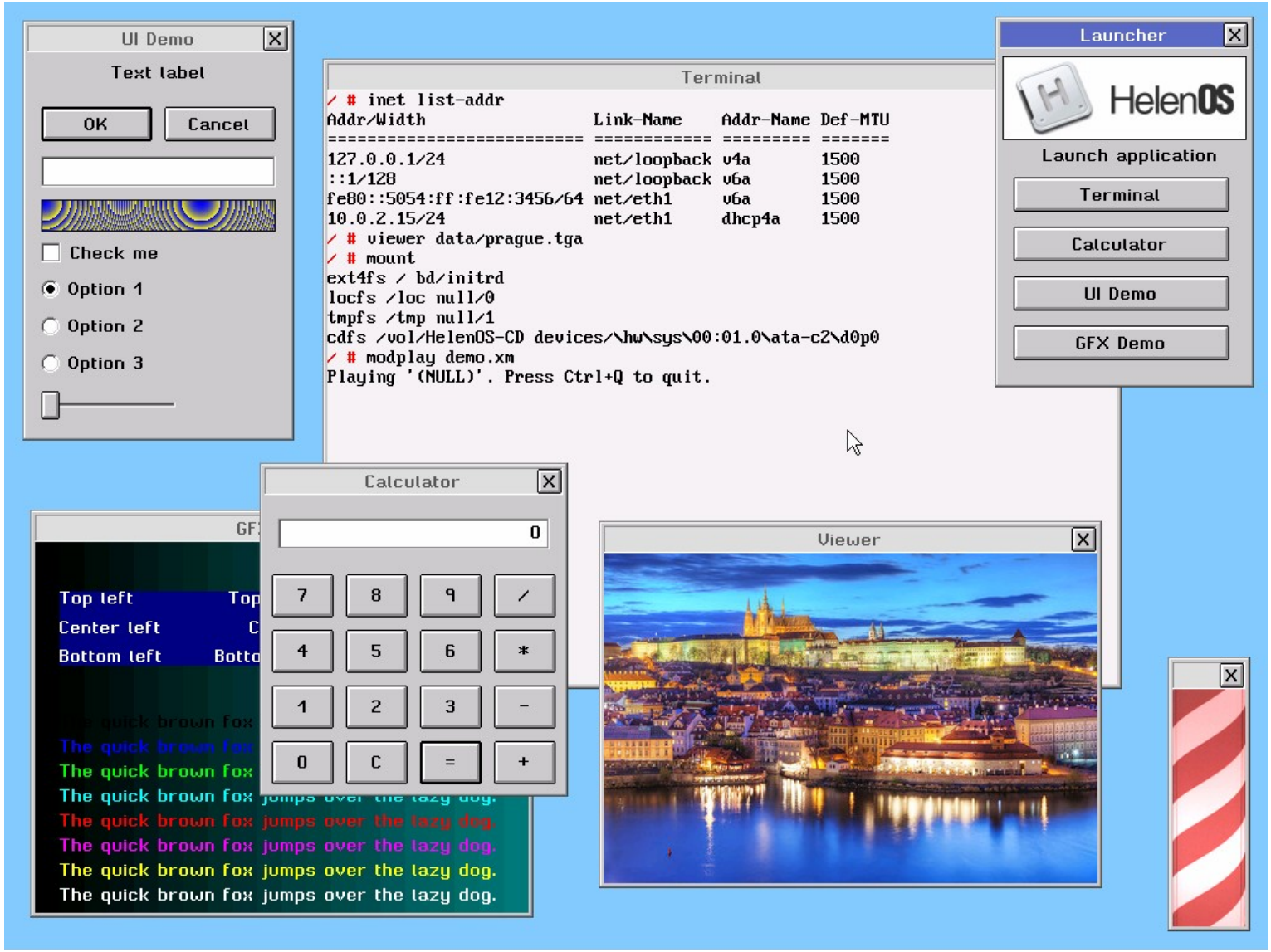
Escape

A UNIX-like microkernel operating system, that runs on x86, x86_64, ECO32 and MMIX. It is implemented from scratch and uses nearly no third-party components. To fit nicely into the UNIX philosophy, Escape uses a virtual file system to provide drivers and services. Both can present themselves as a file system or file to the user. (*github.com/Nils-TUD/Escape*)

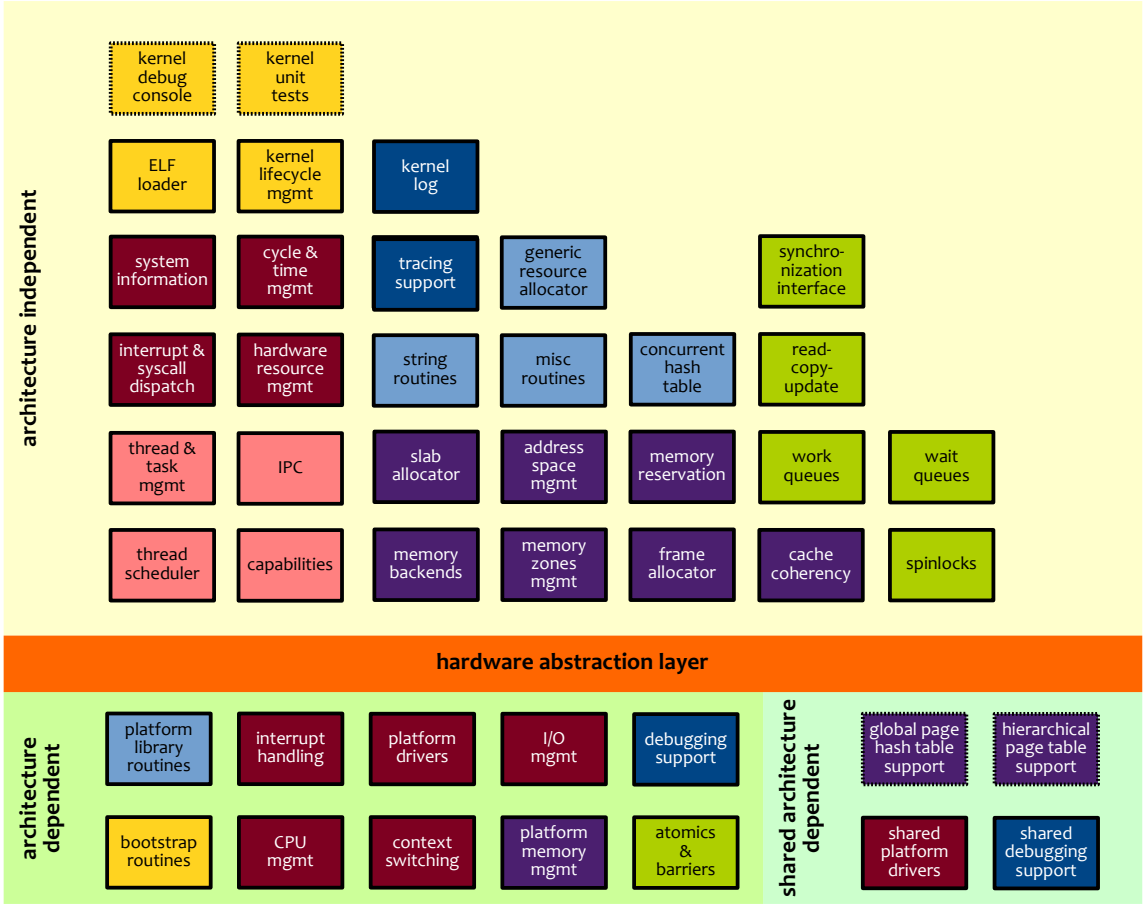
M³

M³

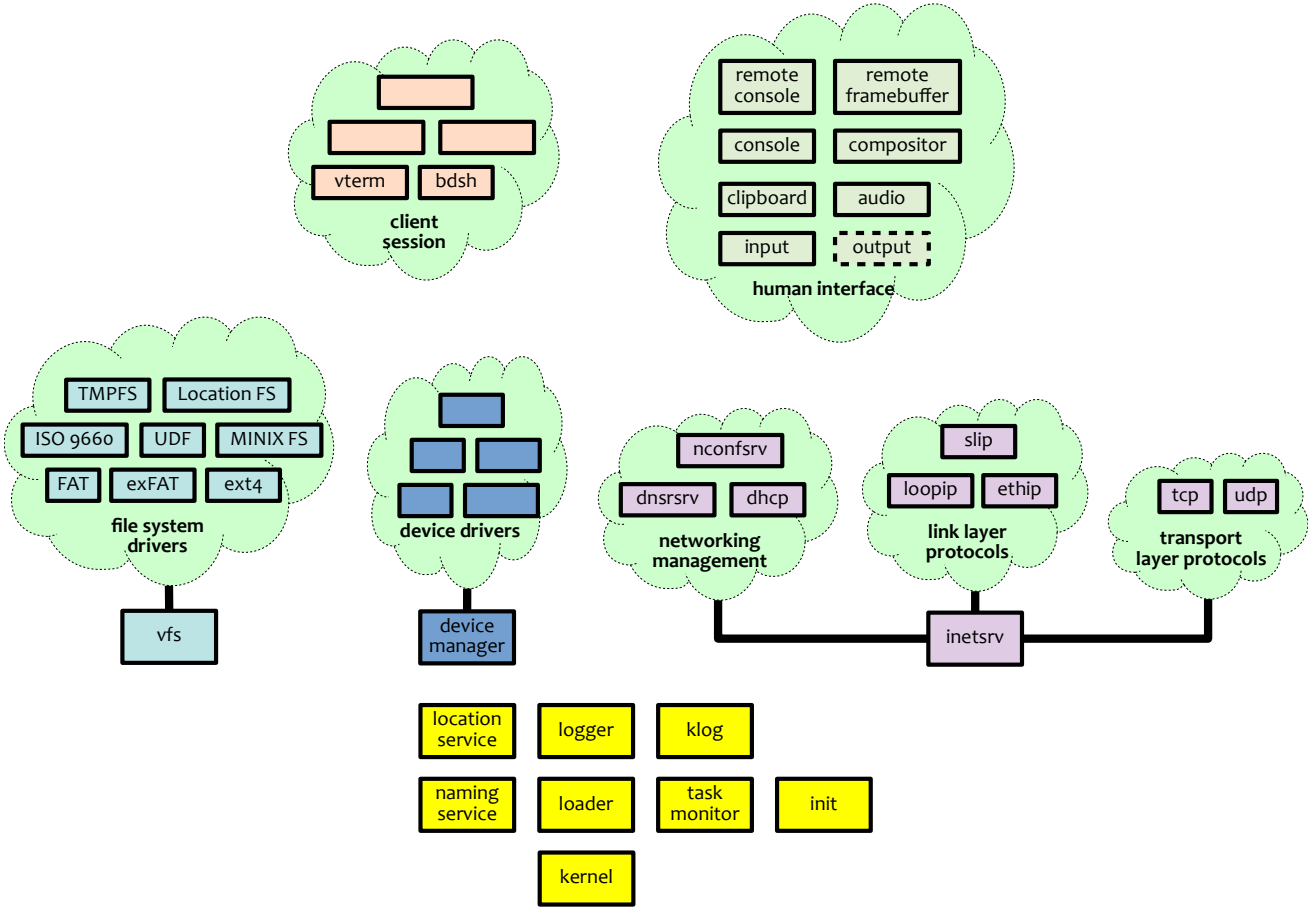
A microkernel-based system for heterogeneous manycores, that is developed as a hardware/OS co-design at the TU Dresden. It aims to support arbitrary cores (general purpose cores, DSPs, FPGAs, ASICs, ...) as first-class citizens. This is achieved by abstracting the heterogeneity of the cores via a new hardware component per core, called data transfer unit. (*github.com/TUD-OS/M3*)


UI Demo
Text label
OK
Cancel
Check me
Option 1
Option 2
Option 3
Terminal
/ # inet list-addr
Addr/Width Link-Name Addr-Name Def-MTU
127.0.0.1/24 net/loopback v4a 1500
::1/128 net/loopback v6a 1500
fe80::5054:ff:fe12:3456/64 net/eth1 v6a 1500
10.0.2.15/24 net/eth1 dhcp4a 1500
/ # viewer data/prague.tga
/ # mount
ext4fs / bd/initrd
locfs /loc null/0
tmpfs /tmp null/1
cdfs /vol/HelenOS-CD devices/\hw\sys\00:01.0\ata-c2\d0p0
/ # modplay demo.xm
Playing '(NULL)'. Press Ctrl+Q to quit.
Launcher
HelenOS
Launch application
Terminal
Calculator
UI Demo
GFX Demo
Calculator
0
7 8 9 /
4 5 6 *
1 2 3 -
0 C = +
Viewer
Top left
Center left
Bottom left
The quick brown fox jumps over the lazy dog.

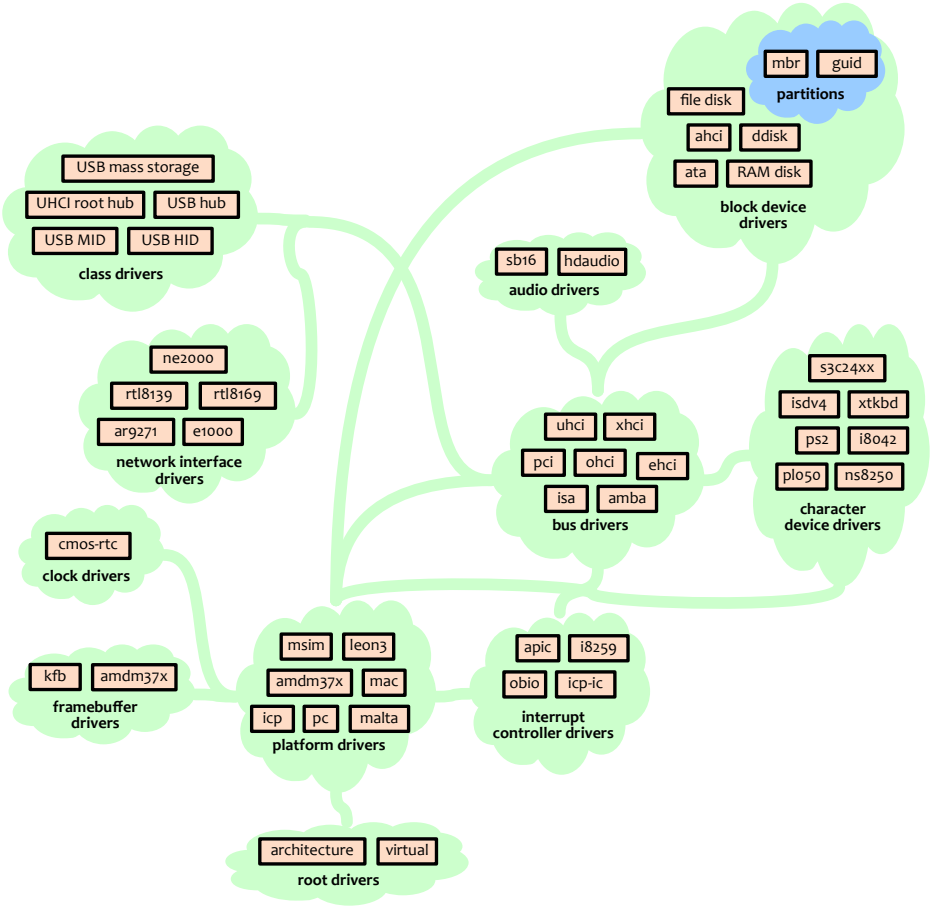
HelenOS Microkernel Functional Blocks



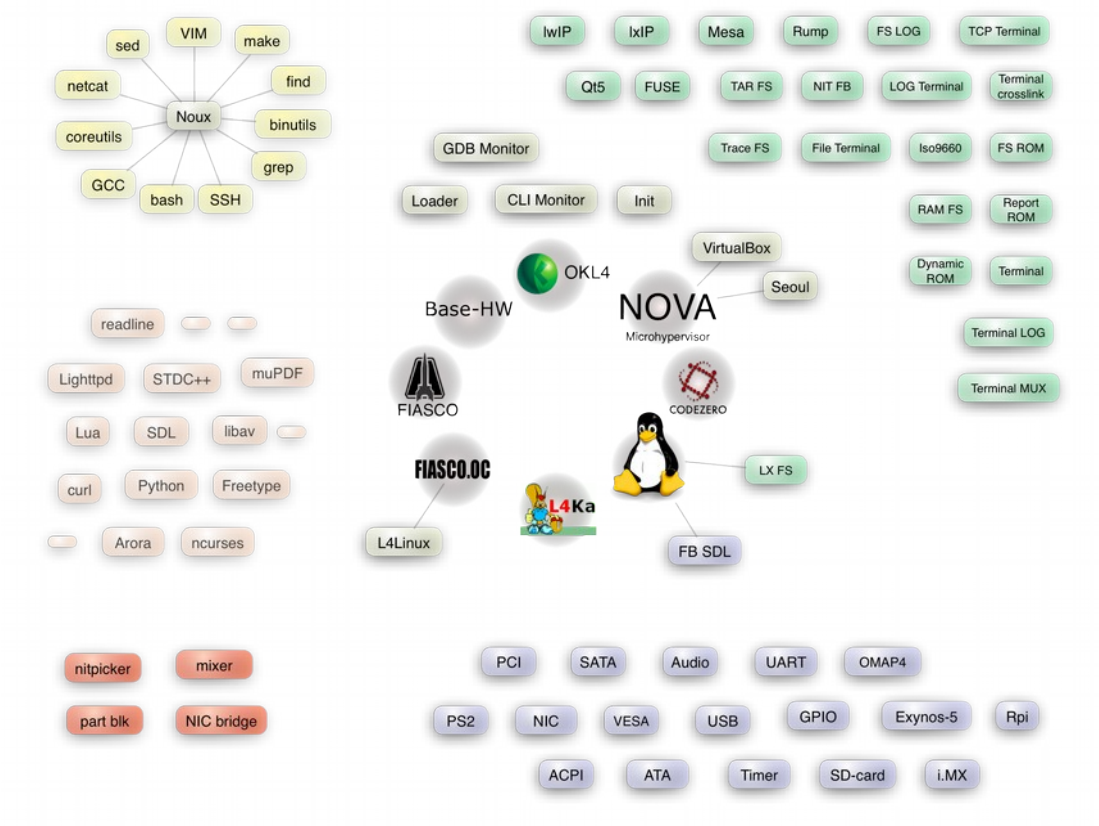
HelenOS User Space Architecture



HelenOS User Space Device Drivers



Genode OS Framework



[1] Feske N.: *Introducing kernel-agnostic Genode executables*, Genode Labs, FOSDEM 2017, https://fosdem.org/2017/schedule/event/microkernel_kernel_agnostic_genode_executables/

Further Reading

- **Du D., Hua Z., Xia Y., Zang B., Chen H.: *XPC: Architectural Support for Secure and Efficient Cross Process Call*, ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA), 2019**
  - https://ieeexplore.ieee.org/abstract/document/8980352
- **Matthias Lange: *The impact of Meltre and Specdown on microkernel systems (*)*, Microkernel Devroom, FOSDEM, 2019**
  - (*) Deliberate misspelling of Meltdown and Spectre
  - https://archive.fosdem.org/2019/schedule/event/meltre_specdown/

HUAWEI

Q&A


HUAWEI
THANK YOU!