FLEXIBLE OPERATING SYSTEM ARCHITECTURE

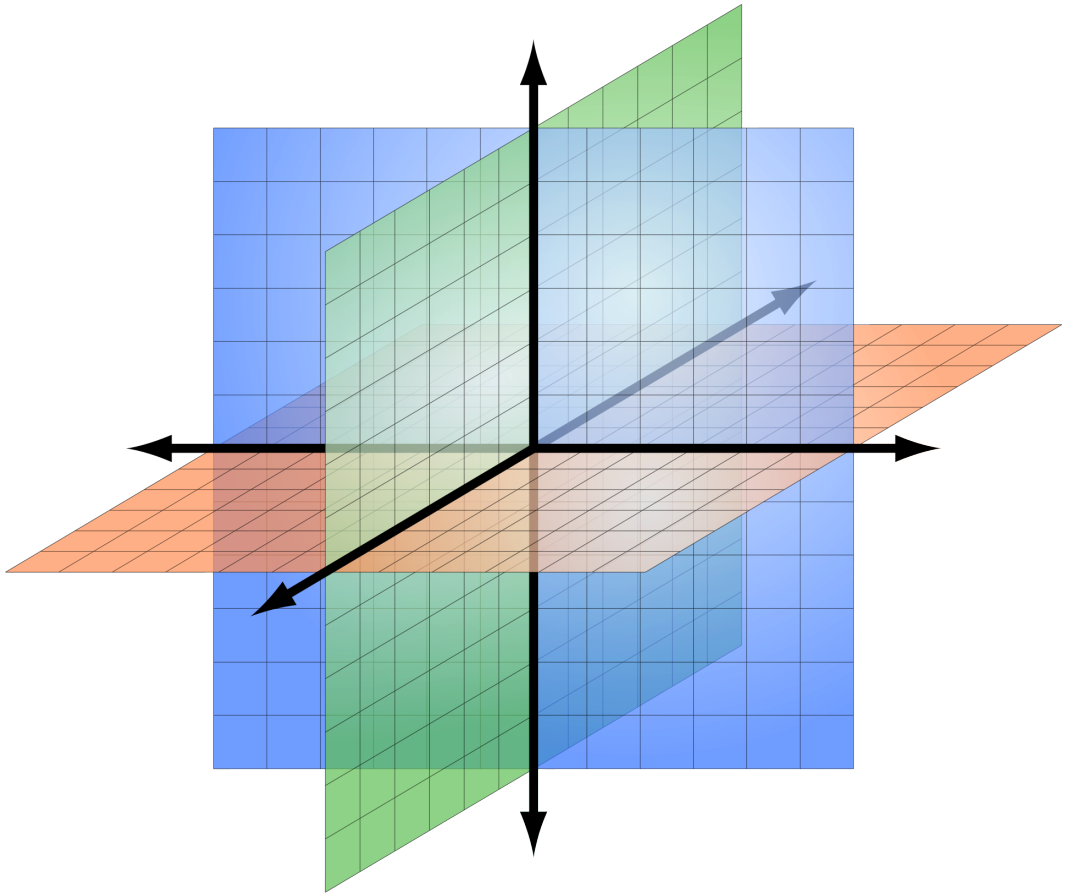
Martin Děcký
martin.decky@huawei.com

November 2019

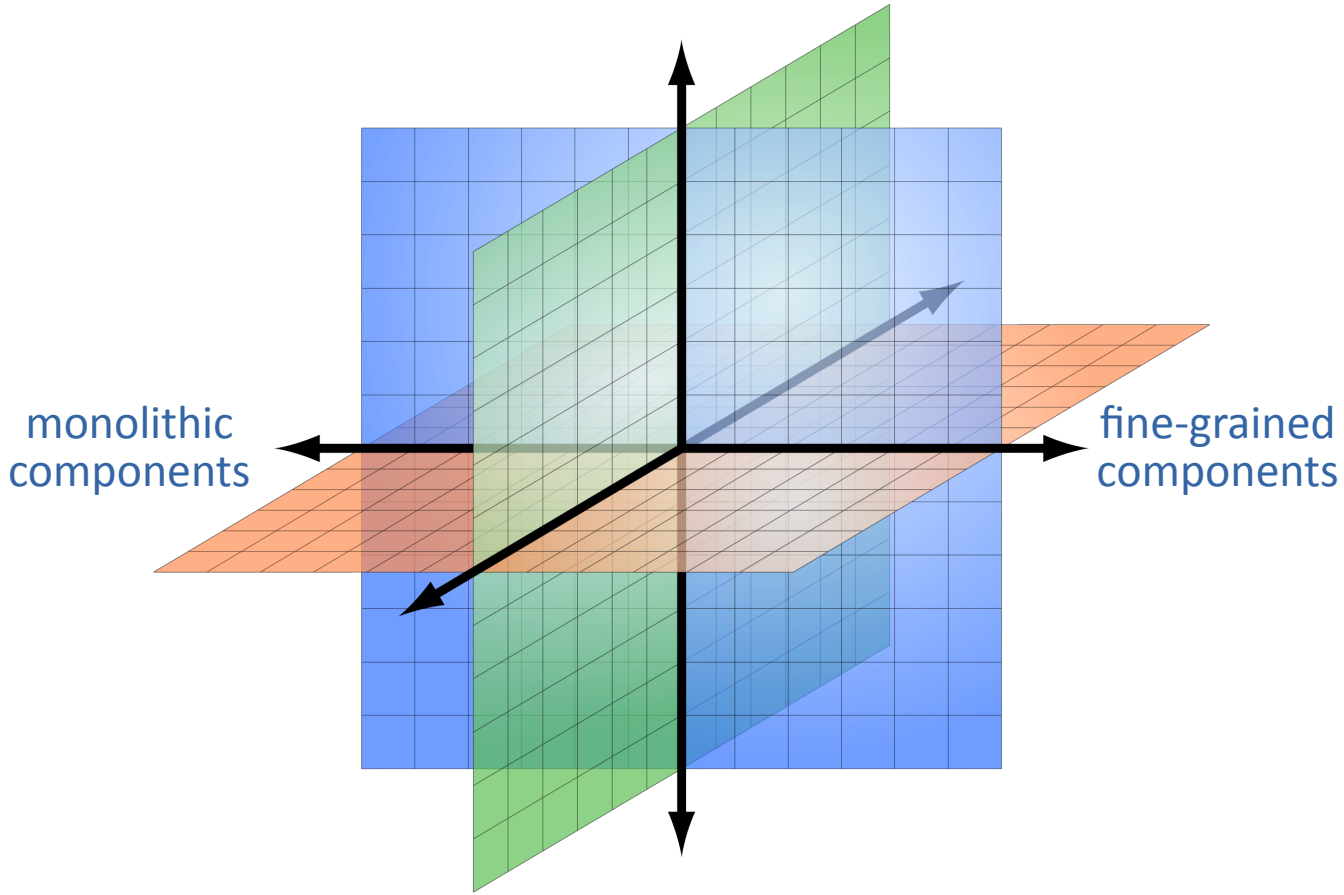
Huawei Dresden Research Center

- **Focusing on microkernel R&D**
  - Fundamental and applied research, design and prototype development
    - Topics ranging anywhere from formal verification to scalability
  - Collaboration with academia and other technology companies
  - Shaping the future product portfolio of Huawei
- **Grown from 0 to 18 researchers/developers during 2019**
  - Just moving to a shiny new office in the city center of Dresden
  - The plan is to restart hiring next year

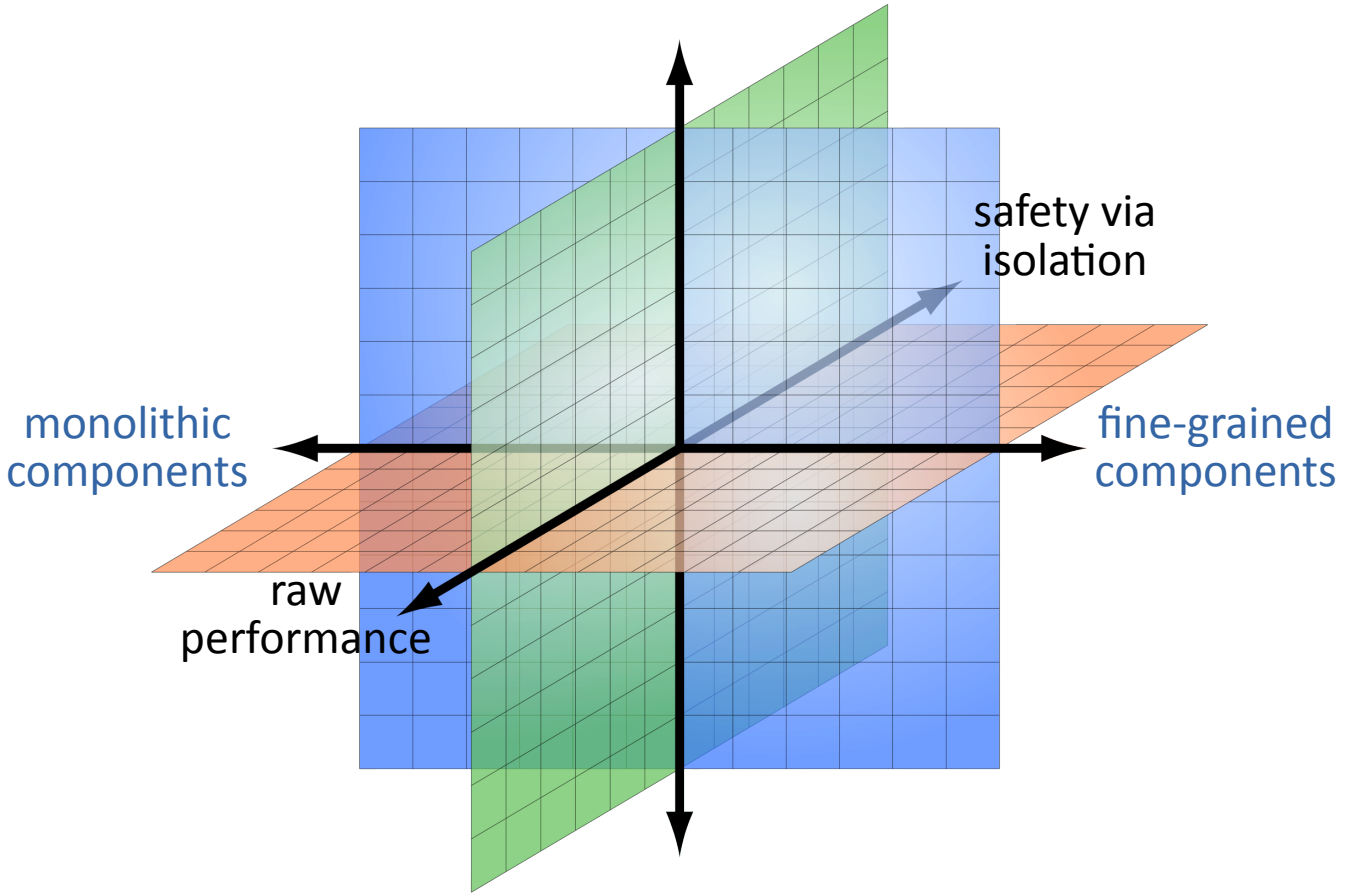
Motivation: OS Design Space



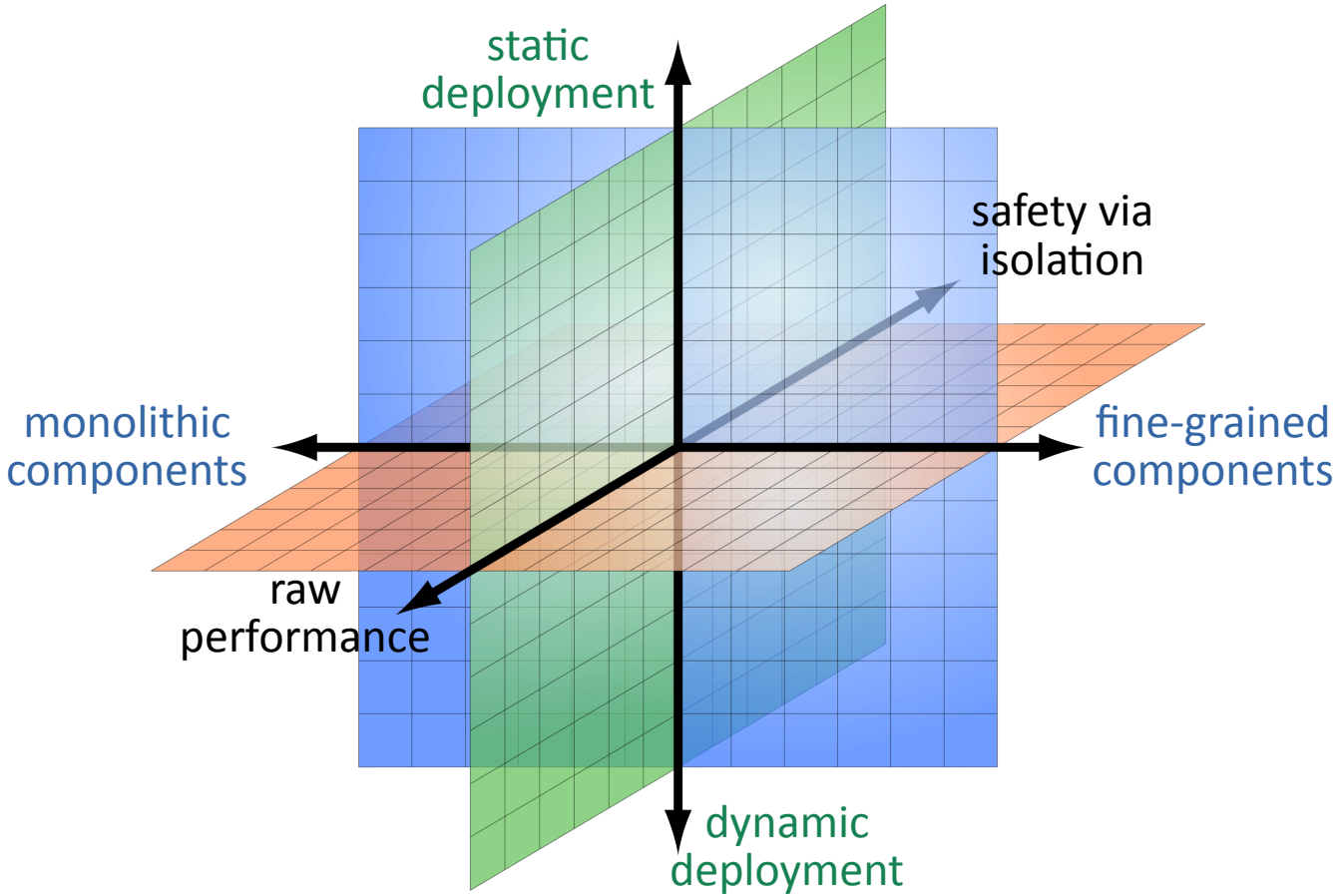
Motivation: OS Design Space



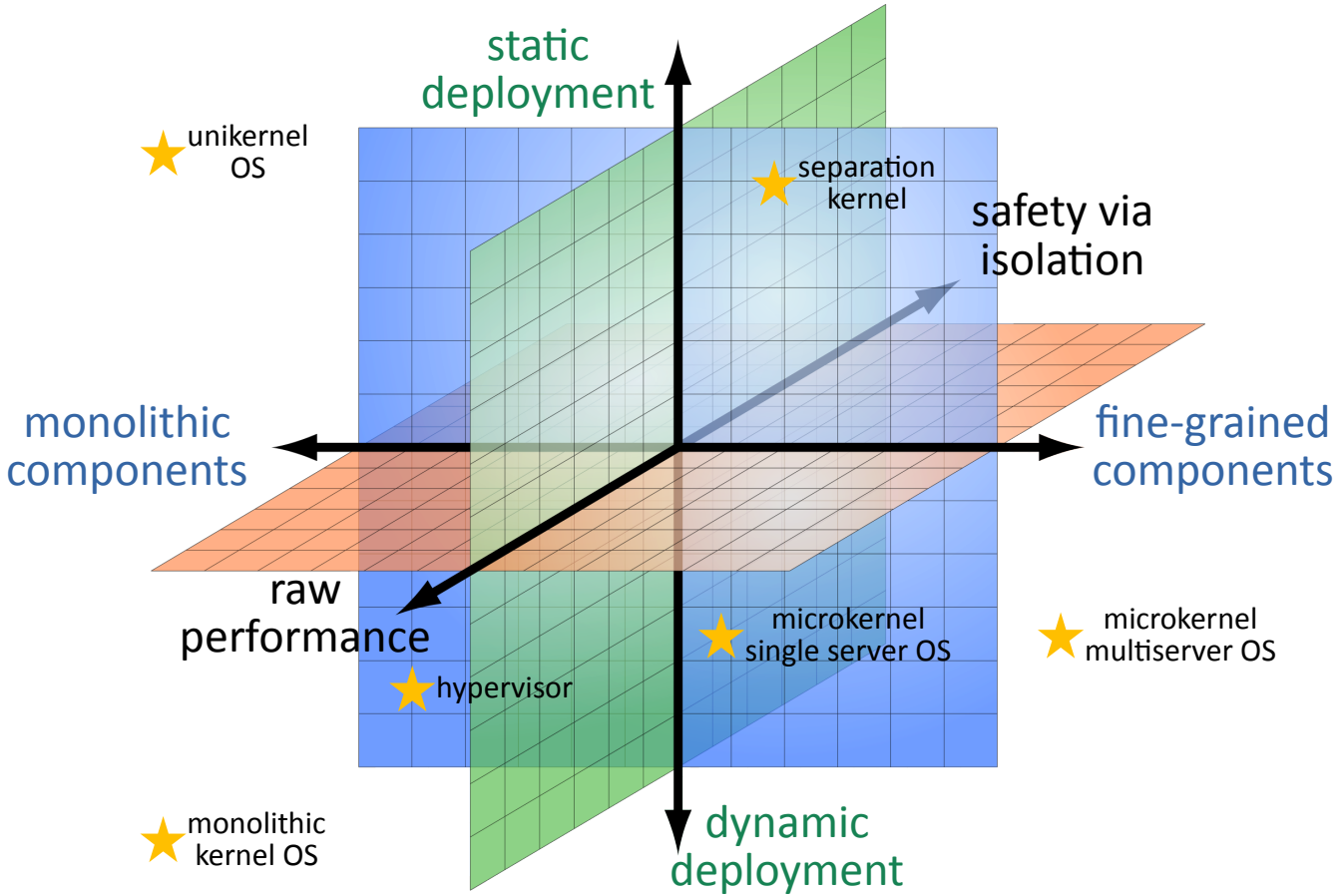
Motivation: OS Design Space



Motivation: OS Design Space



Motivation: OS Design Space



Problem Statement

- **There is no "one-size-fits-all" operating system design**
  - Endless debates whether monolithic kernels or microkernels are "better" are pointless
    - Better with respect to what?
      - Specific input (domain) requirements
      - Specific mix of desired trade-offs
- **Operating system design tends to be a "big upfront" decision**
  - Whether we go with an extreme design or a hybrid design, the design decisions affect the code base in a major way
    - Switching to a different design during implementation usually means a major rewrite of the code base

Problem Statement (2)

- **Possible reasons for the rigidity of the operating system design**
  - Need to deal with relatively tedious low-level abstractions and mechanisms in an efficient manner (context switching, memory management, isolation, communication)
  - Need to have several major mechanisms as “singletons” (no way of having multiple implementations coexist side-by-side)
  - Historically relatively basic toolchain support (C language, no explicit way of capturing the software architecture as formal artifacts)
  - Need for a stable API (not too many “moving parts” on the side of applications), compatibility with ancient APIs (e.g. POSIX)
  - Long way from a “toy OS” to a “real-world OS”

Flexible OS Architecture

- **Goal**
  - Providing an almost continuous spectrum of tunability between isolation/performance, component granularity and deployment modes [1]
    - By the same code base
- **Key aspects**
  - Fundamental theory
  - Generic framework, tools and formalisms
  - Individual mechanisms

Fundamental Theory

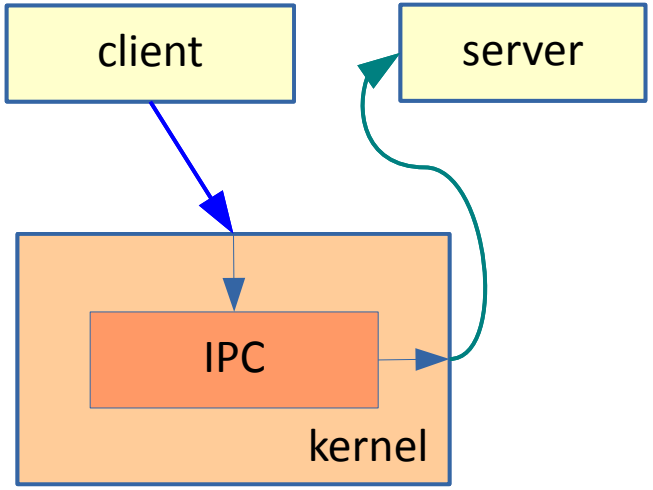
- **Baseline design**
  - Initial “rigid” design of the operating system code base (i.e. an actual working operating system implementation)
  - **Observation:** The more componentized (modular) the code base is the more suitable it is for achieving flexibility
    - Fine-grained microkernel multiserver baseline design is ideal
    - **The usual anxiety from performance overhead is completely subverted**
      - The whole point is to automatically merge fine-grained components into coarse-grained components if performance is essential
    - Fine-grained baseline design keeps all the benefits for the formal verification of correctness, etc.

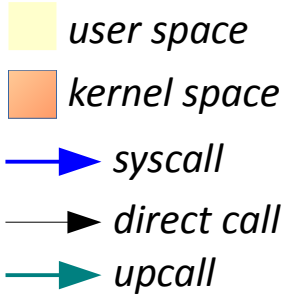
Fundamental Theory (2)

- **Deployment design**
  - Target run-time architecture of the operating system
    - Defined by the input (domain) requirements and the desired mix of trade-offs
    - Achieved from the baseline design using *mutators* (mechanisms for changing the design)
  - **Goal:** Keeping end-user applications (both clients and servers) as unaffected as possible by the mutations (on the level of API)

Basic Mutation

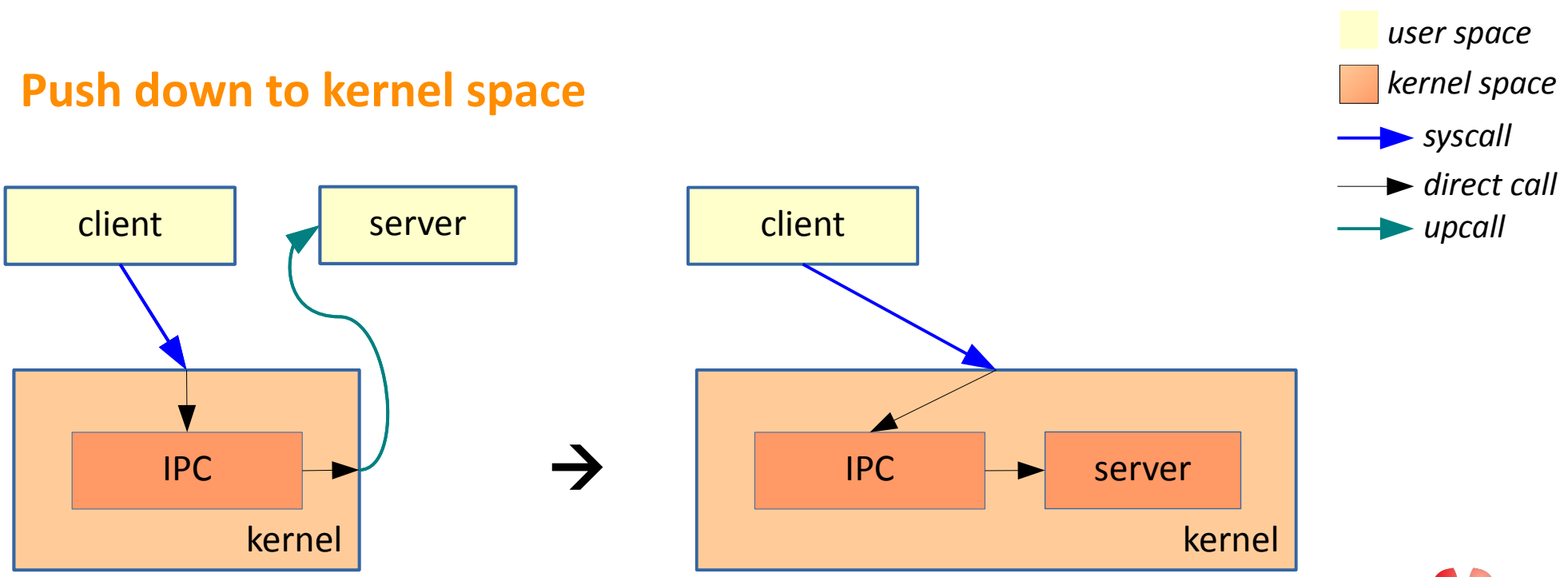
- **Push down to kernel space**





Basic Mutation

- Push down to kernel space



Concrete Mechanisms

- **Link-time approach**
  - Introducing mutators in the form of replacement code for the IPC mechanisms
    - Original client-side and server-side business logic stays unaffected
    - Additional isolation checks (if required) are introduced in the replacement code
    - When merging components, symbol renaming is used to mitigate namespace collisions
  - **Benefits:** Non-intrusive, support for binary-only (closed-source) components, covering a fair portion of the OS design space
  - **Drawbacks:** Some performance overhead cannot be fully eliminated
  - First prototype implemented on top of HelenOS

Other Aspects

- **Architecture Description Language**
  - Formal description the baseline design (component interfaces, bindings and dependencies)
- **Deployment Specification Language**
  - Formal description of the deployment design
    - Capturing input (domain) requirements and the desired mix of trade-offs
- **Performance engineering and modeling**
  - Semi-automatic derivation of the optimal mix of trade-offs

Conclusion

- There is no “one-size-fits-all” operating system design
- Operating system design tends to be a “big upfront” decision
- Flexible OS architecture aims at changing the fundamental paradigm
  - Baseline design vs. deployment design
  - Starting ideally with a fine-grained baseline design
  - Subverting the usual anxiety from performance overhead of microkernels
- Initial results very promising

Acknowledgements

- **Huawei Dresden Research Center**
  - Martin Beck
  - Sebastian Ertel
  - Ming Fu
  - Pramod Bhatotia
- **Huawei OS Kernel Lab**
  - Yu Li
  - Haibo Chen

References

[1] Děcký M.: *Application of Software Components in Operating System Design*, doctoral thesis, Charles University, 2015

[2] Polakovic J., Ozcan A. E., Stefani J.-B.: *Building Reconfigurable Component-based OS with THINK*, in the Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, 2006

Q&A


HUAWEI
THANK YOU!