The Microkernel Overhead

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

D3S

Martin Děcký

decky@d3s.mff.cuni.cz

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

HelenOS

Why am I Here?

- HelenOS developer since 2005
- Computer science researcher
  - Distributed and component systems
  - Formal verification of OS correctness
- Monolithic and microkernel OSes have both their pros and cons
  - The microkernel overhead is a particular source of many misconceptions

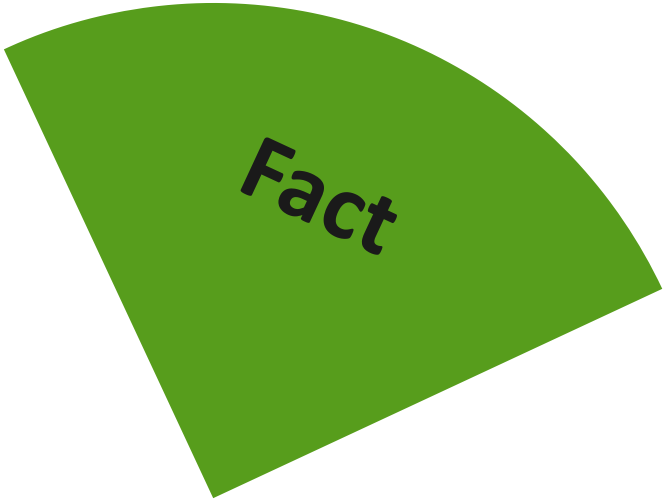
Disclaimer

For the sake of brevity, some of the following slides might be oversimplified.

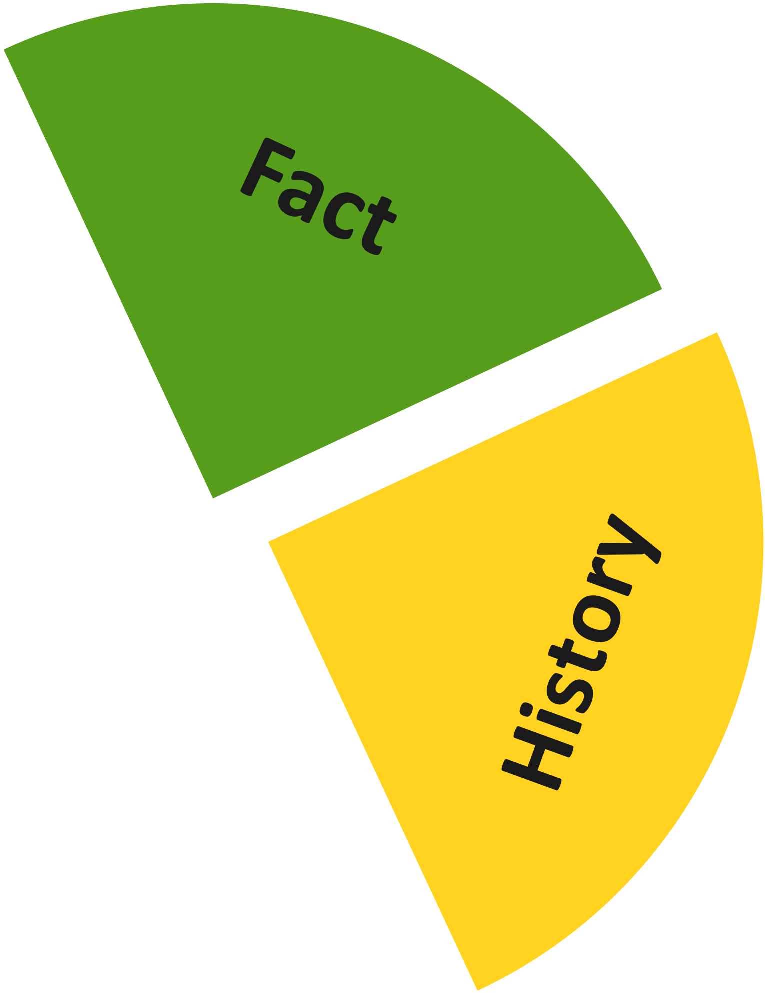
When making important decisions, always consult the original references, rely on your own observations and draw your own conclusions.

The Overhead?

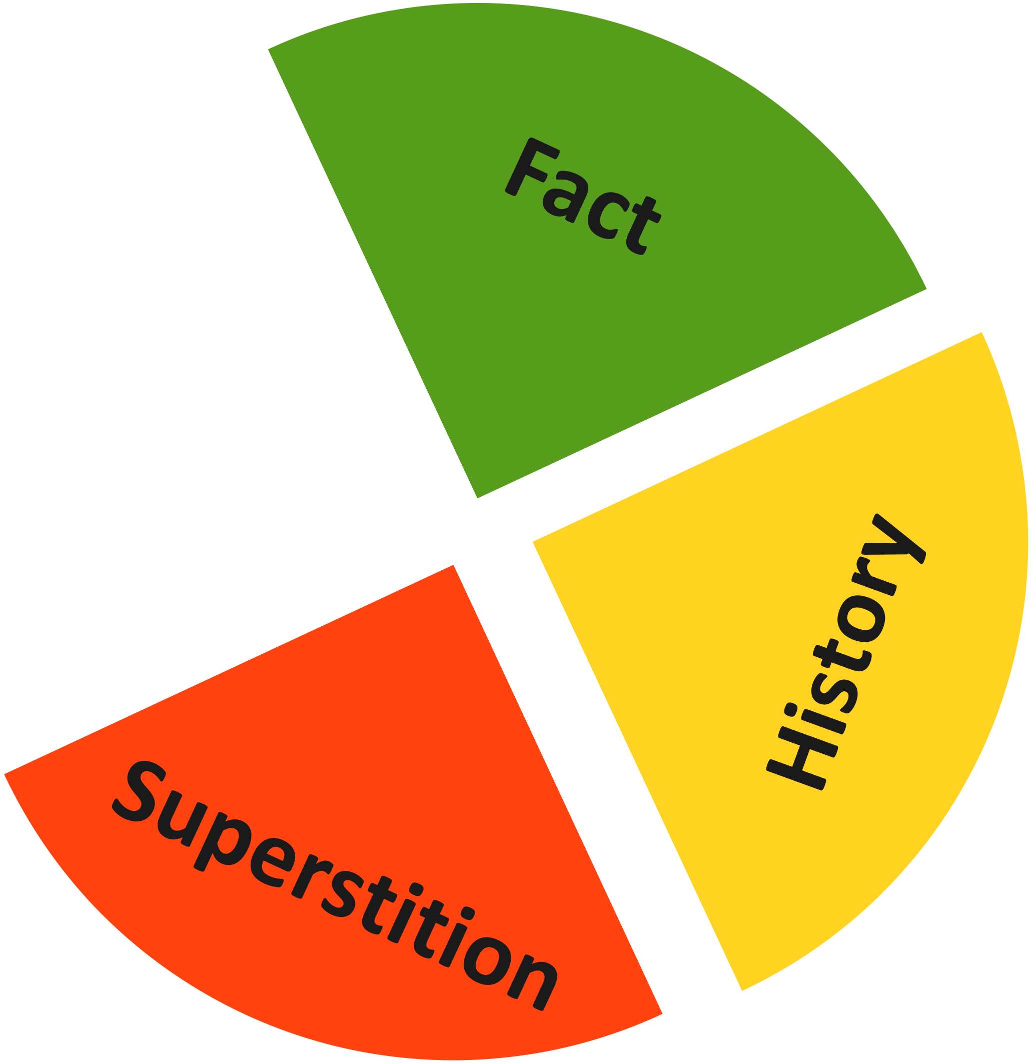
The Overhead?



The Overhead?



The Overhead?

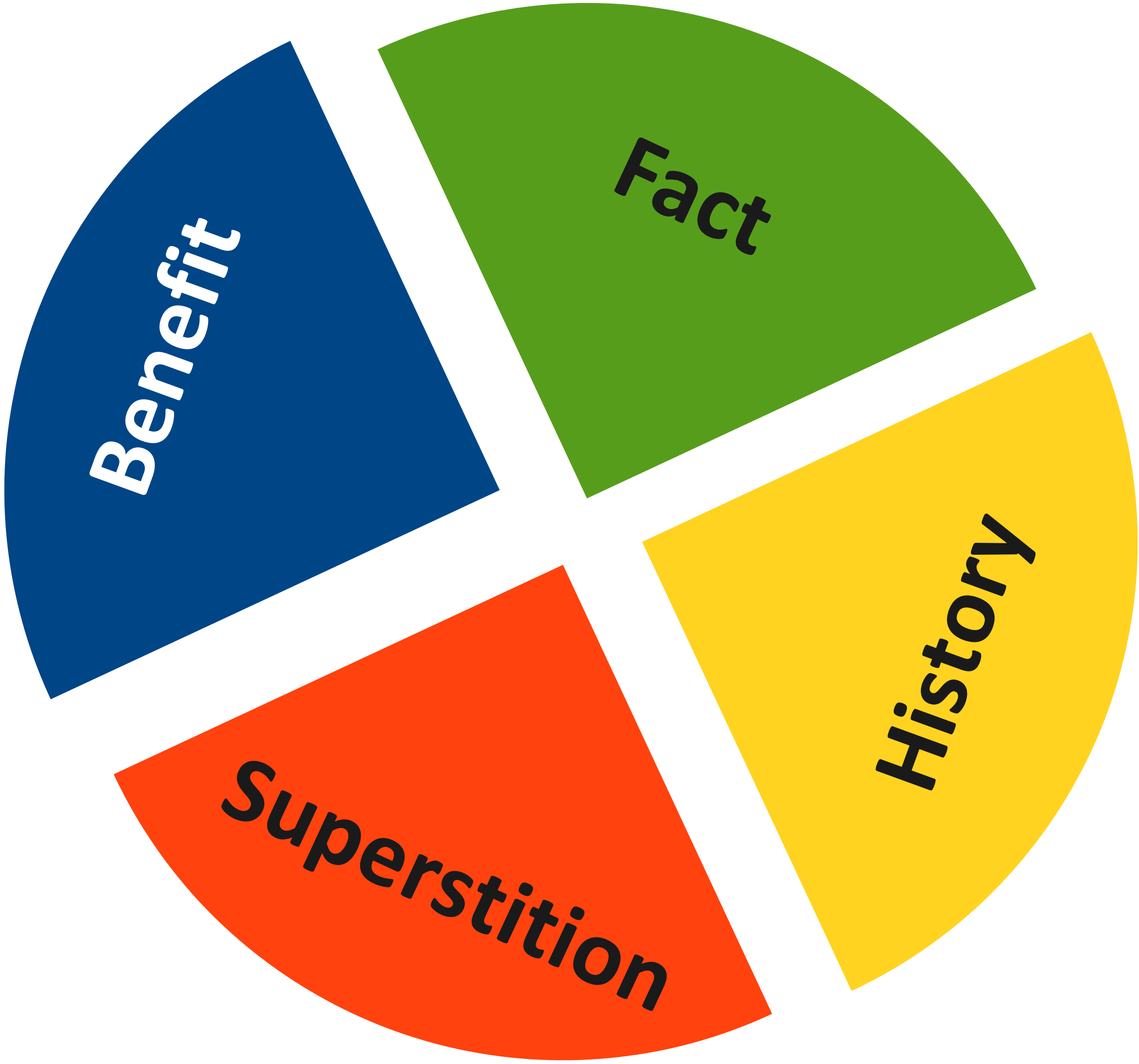


Fact

History

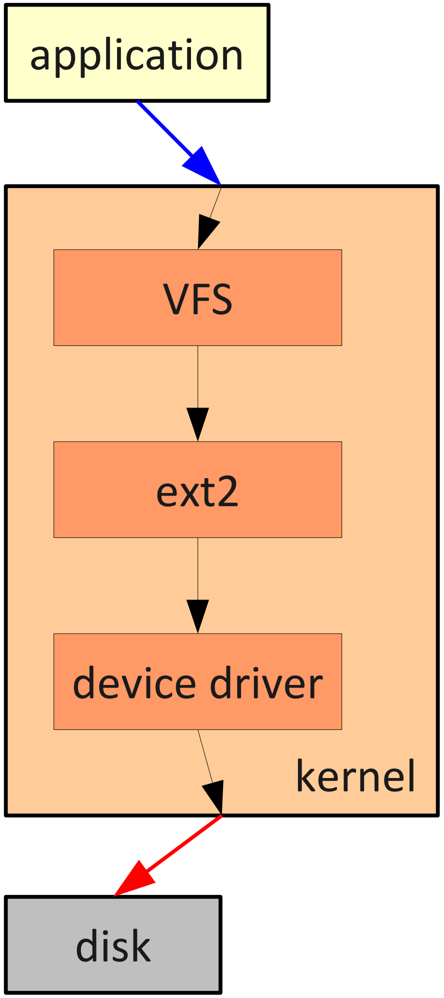
Superstition

The Overhead?



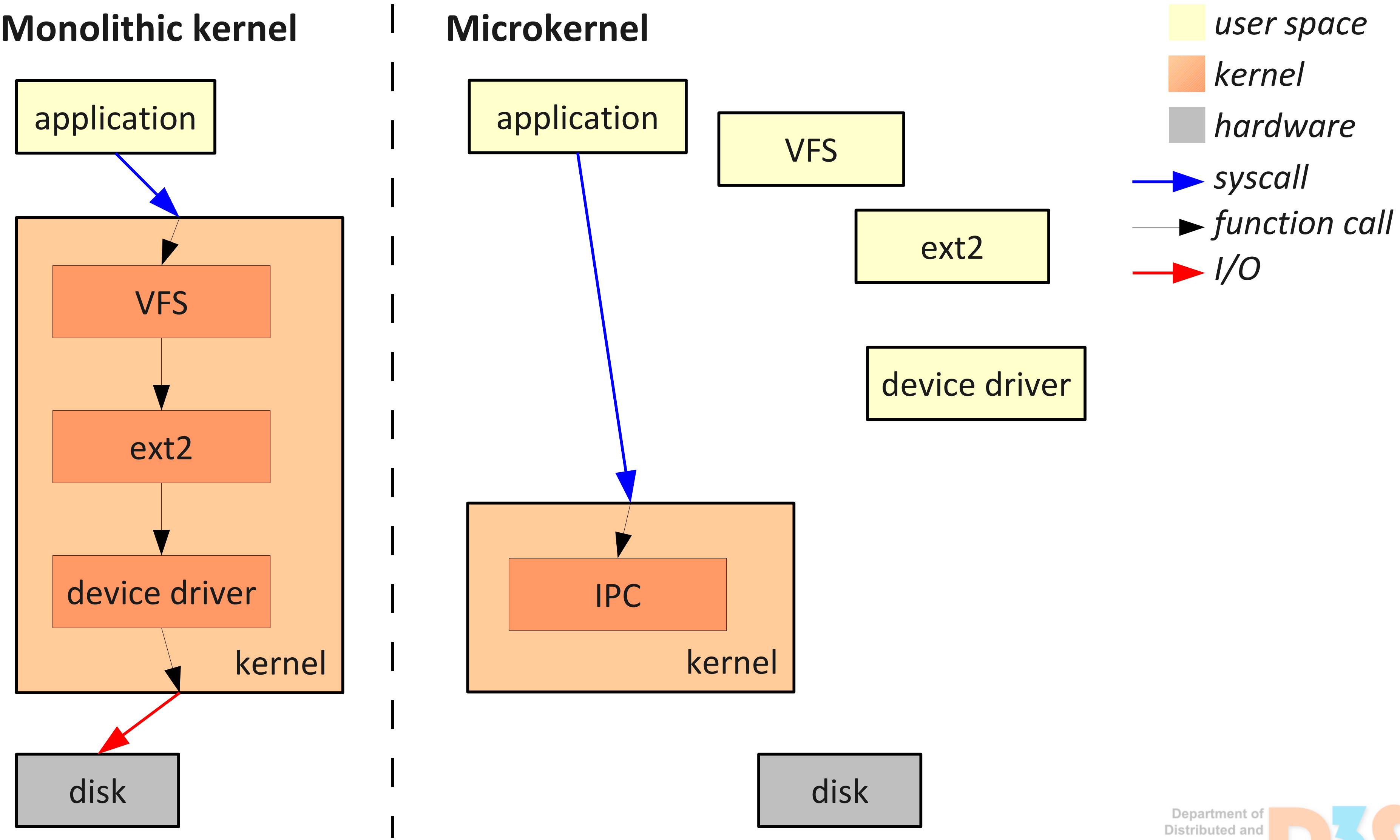
Microkernel Overhead as a Fact

Monolithic kernel



🟨 *user space*
🟧 *kernel*
⬜ *hardware*
→ *syscall*
→ *function call*
→ *I/O*

Microkernel Overhead as a Fact



Monolithic kernel

application

VFS

ext2

device driver

kernel

disk

Microkernel

application

VFS

ext2

device driver

IPC

kernel

disk

🟨 *user space*

🟧 *kernel*

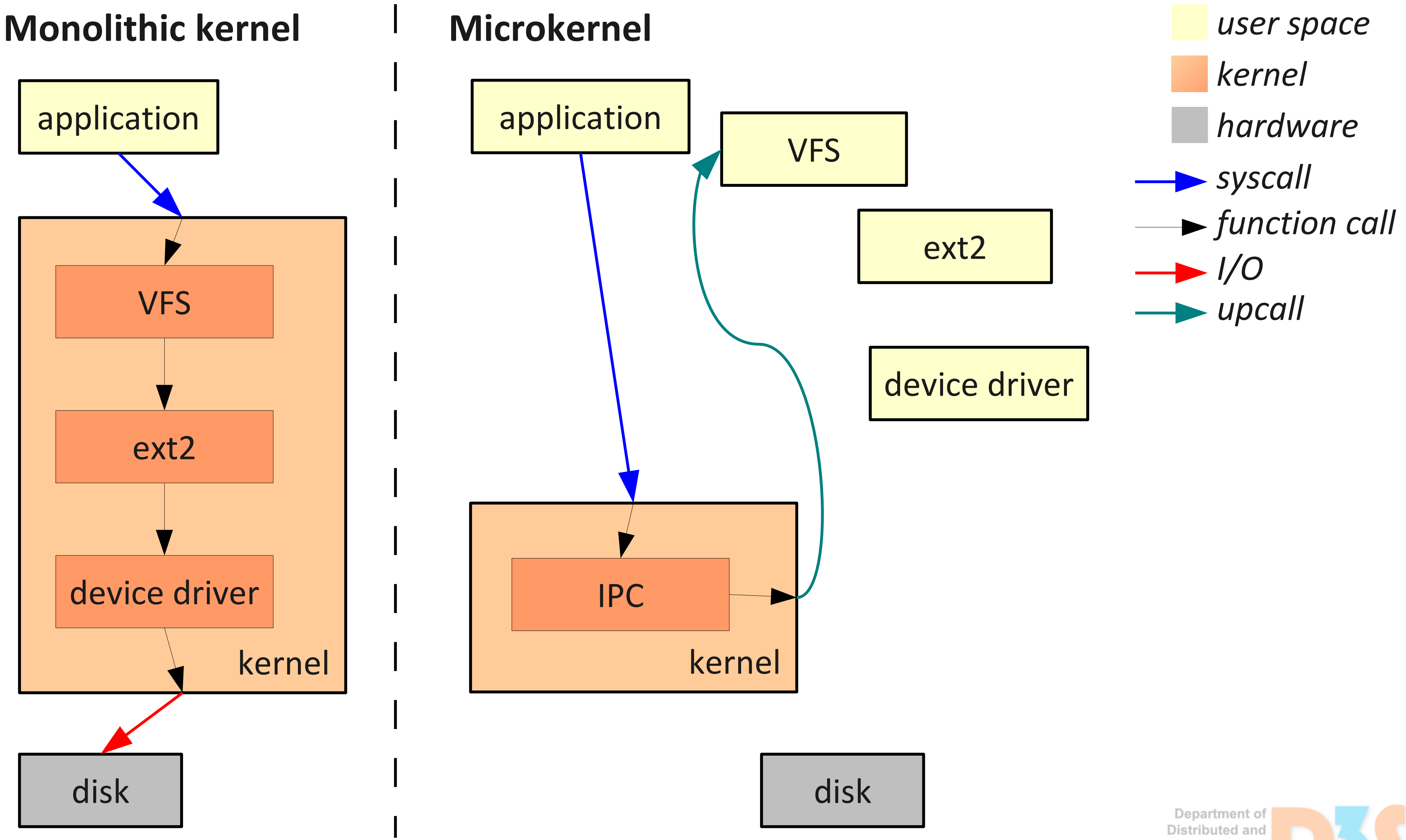
⬜ *hardware*

→ *syscall*

→ *function call*

→ *I/O*

Microkernel Overhead as a Fact



Monolithic kernel

application

VFS

ext2

device driver

kernel

disk

Microkernel

application

VFS

ext2

device driver

IPC

kernel

disk

user space

kernel

hardware

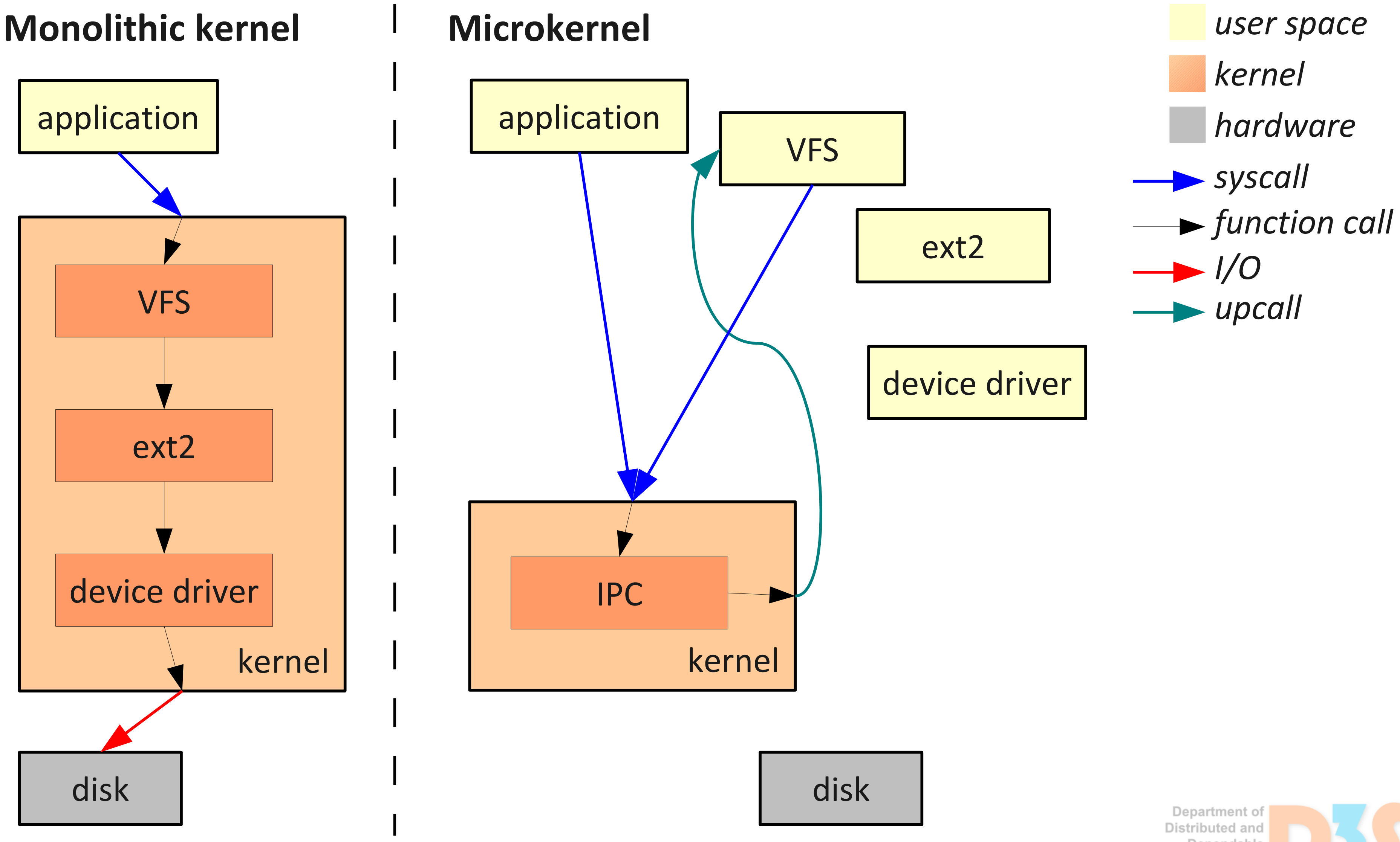
syscall

function call

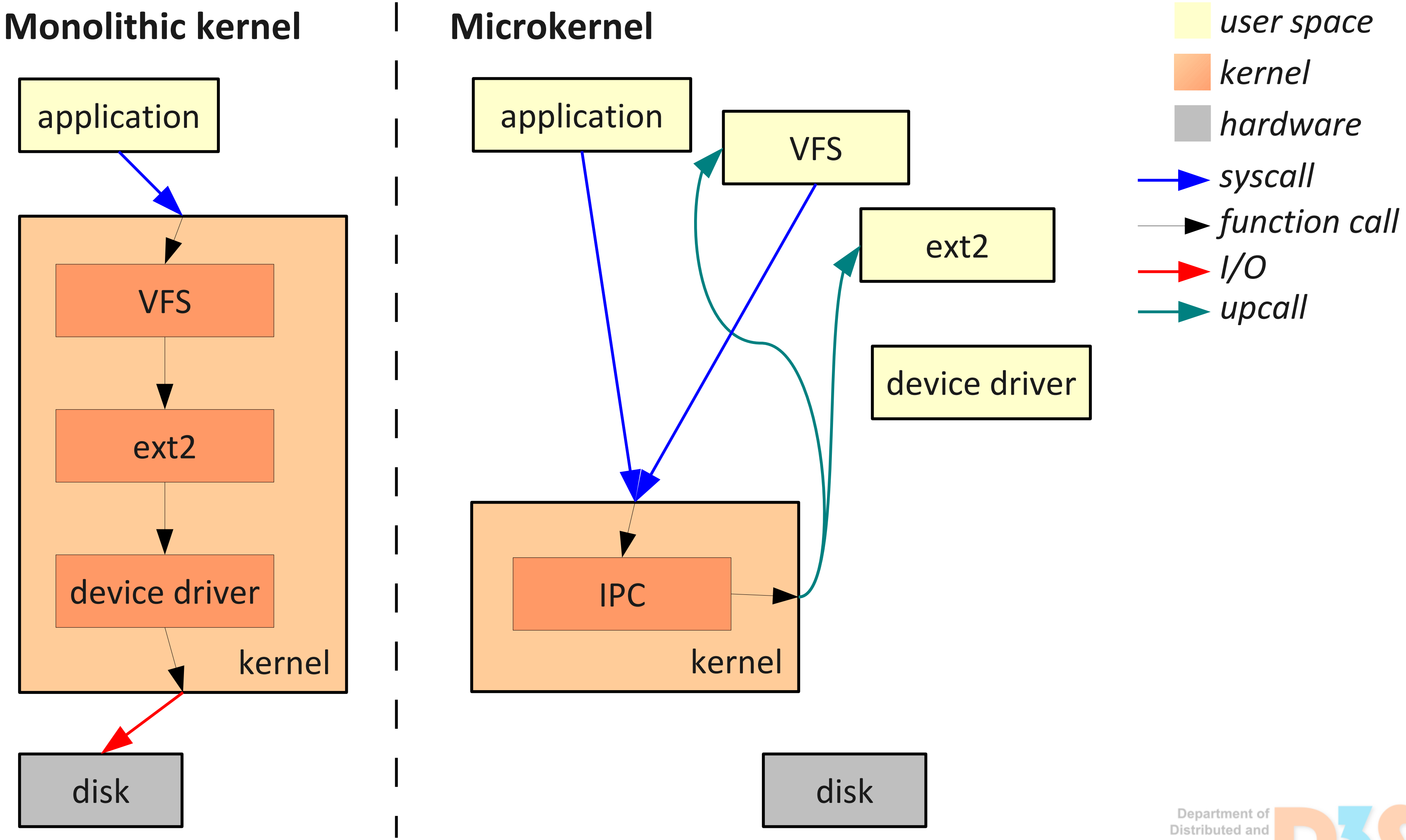
I/O

upcall

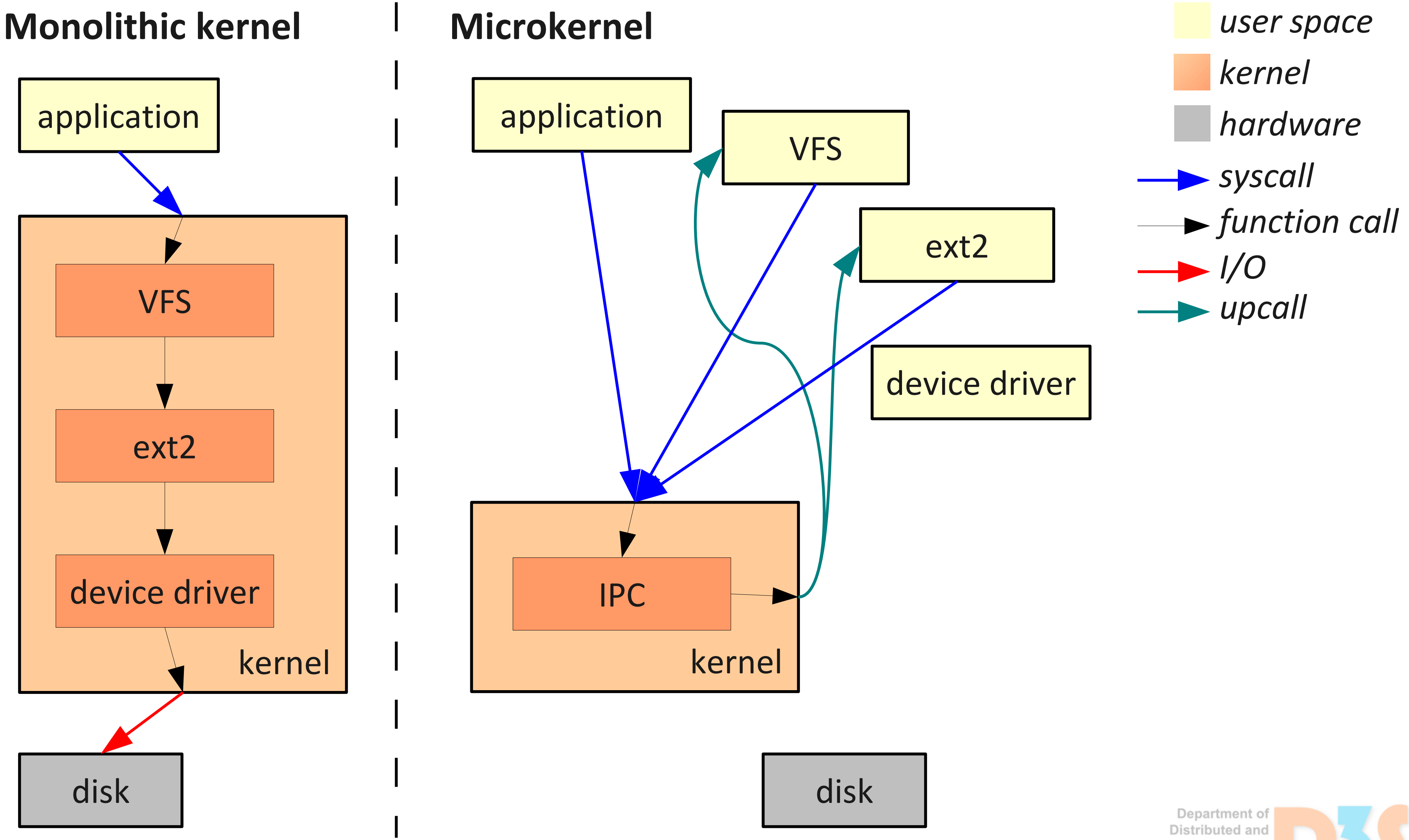
Microkernel Overhead as a Fact



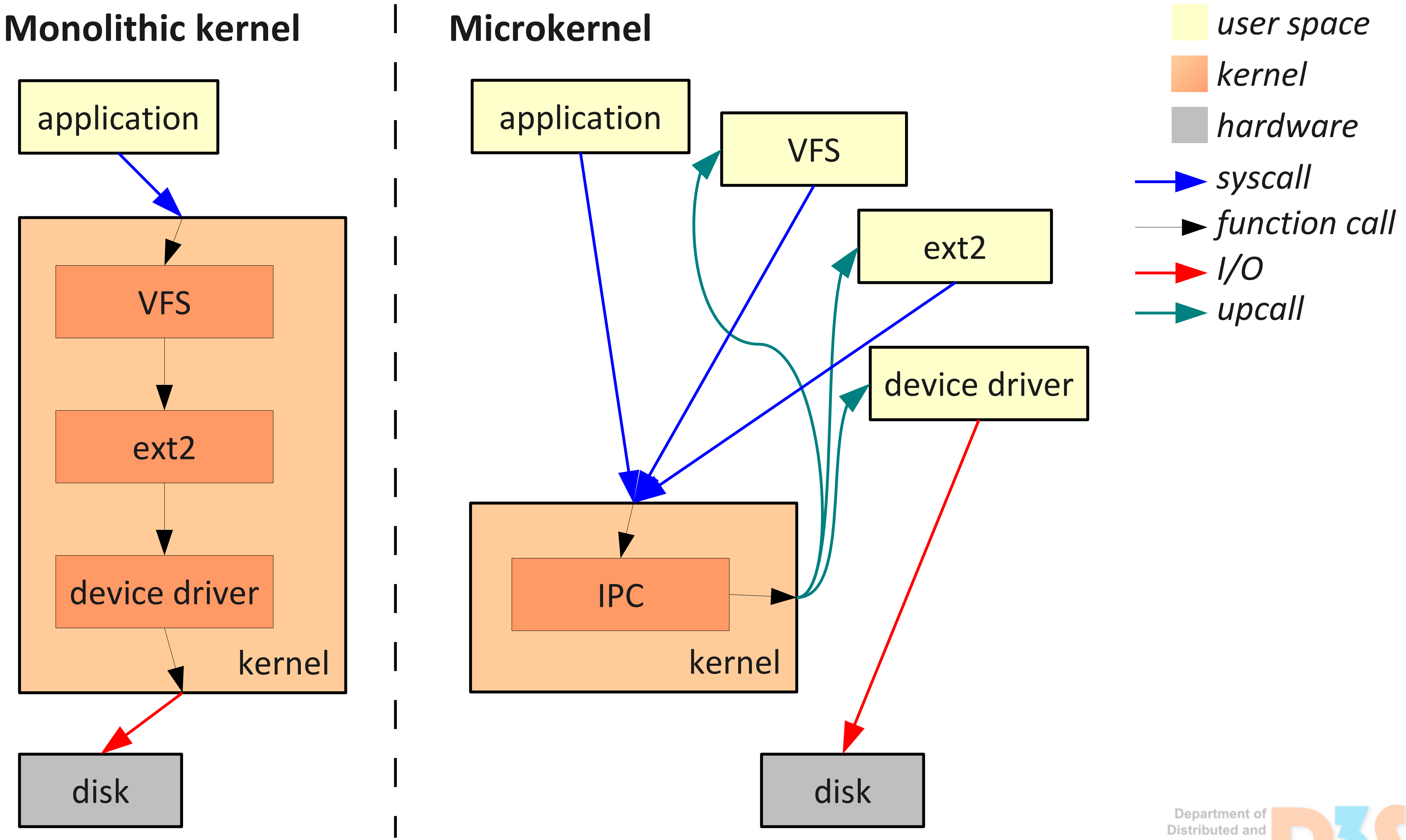
Microkernel Overhead as a Fact



Microkernel Overhead as a Fact



Microkernel Overhead as a Fact



Microkernel Overhead as a Fact (2)

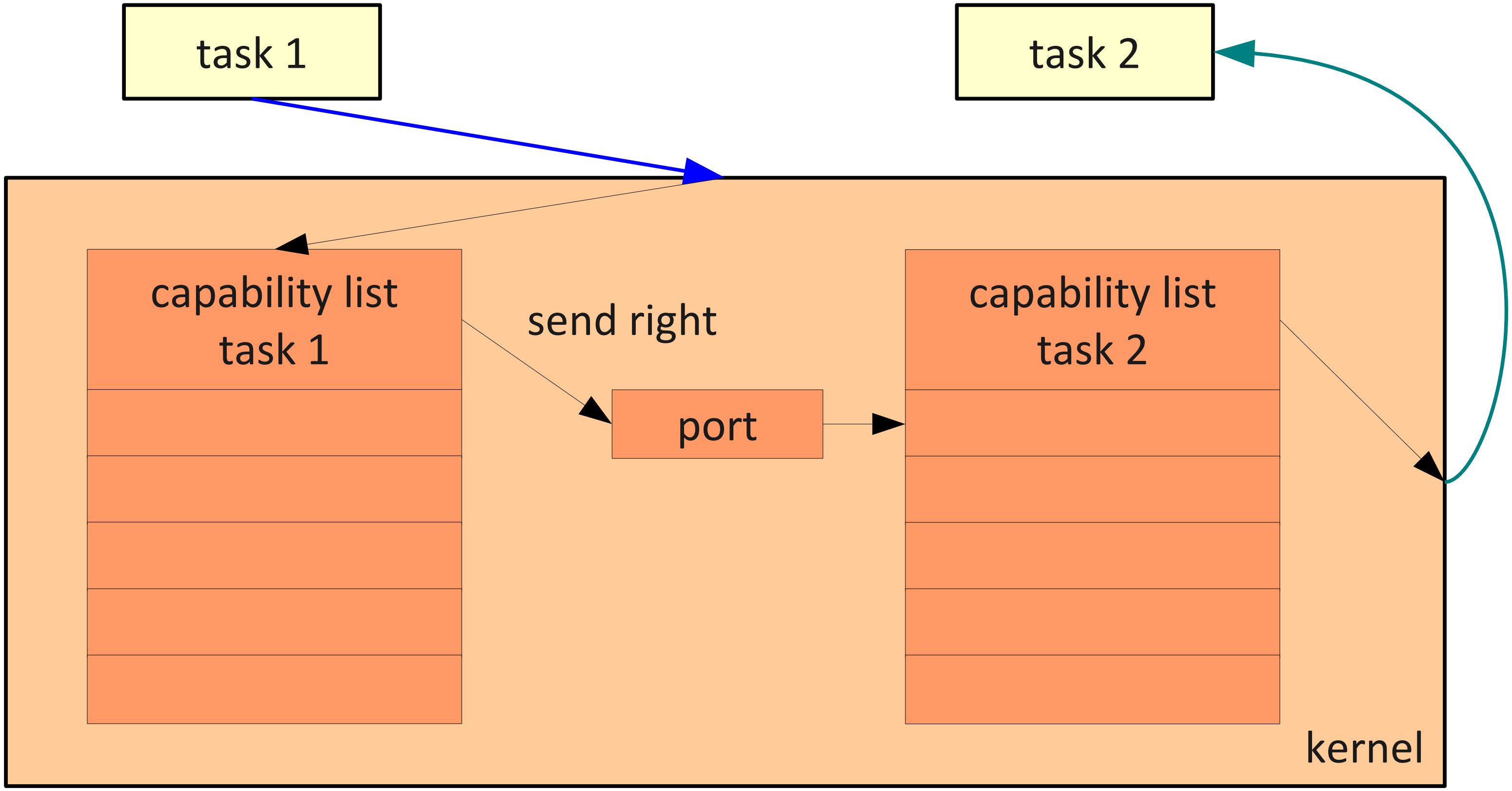
- Natural reasons for the microkernel overhead
  - More communication barriers
    - Function call → Upcall + Syscall
    - Synchronous execution → Context switch
  - Heavier operations
    - Jump → Mode switch
    - Argument passing → Argument queuing
    - → Scheduler execution
- What is the actual extent of the overhead?

Microkernel Overhead in History

- Benchmarks of Mach 3.0 (1997)
  - Single-server Mach vs. UNIX slowdown: 1.5×
    - 73 % of the slowdown due to IPC overhead
      - 80 % of the IPC overhead due to access rights and message validity checking
  - Communication barriers are unavoidable
    - It is the very cornerstone of microkernel design
    - Not accountable for more than a 3× higher overhead (not 3× higher slowdown)
  - The problem are the heavy operations

Microkernel Overhead in History (2)

- Mach asynchronous IPC



Microkernel Overhead in History (3)

- Mach asynchronous IPC
  - Complex access rights evaluation in the kernel
  - Complicated queuing in the kernel
    - Simple data structures (linked lists)
    - Excessive cache footprint
  - Sequential programming paradigm
    - IPC used mostly in synchronous manner

Microkernel Overhead in History (4)

- Benchmarks of L4 (1997)
  - Single-server L4 vs. UNIX slowdown: 1.03×
    - Single IPC call overhead comparable to single UNIX syscall overhead
      - 20× faster than on Mach
    - The slowdown caused only by the communication barriers
    - Even the overhead of a multiserver variant expected to be rather reasonable

Microkernel Overhead in History (5)

- L4 synchronous IPC
  - Explicit client/server rendez-vous
    - No need for full context switch
      - Data passed directly in registers and in shared memory
      - No rescheduling, no queuing
  - Highly optimized implementation
    - Small working set
      - Better spatial locality, cache friendly
    - Leaving the access right policies to the user space servers

Microkernel Overhead Superstition

- Mach 3.0 is a (too) well-known failed example
  - First-generation archetypal microkernel
  - Establishing the terminology, etc.
  - Part of university curricula all over the world
  - Never practically used as a microkernel with user space drivers
    - XNU: Single-server microkernel with **kernel drivers**
    - Hurd: Multiserver microkernel with **kernel drivers**

Microkernel Overhead Superstition (2)

- Opinion of the general public
  - Tanenbaum-Torvalds debate (1992)
    - Performance of microkernels mentioned several times
  - Hybrid design of Windows NT
    - Kernel space device drivers
    - Microkernel modularity, but a single address space
  - *“Microkernels are just research toys”*
  - *“Real-life performance demands monolithic kernels”*

Real-life Performance?

- Let's consider a theoretical 1.5× slowdown
  - A 50 % faster CPU required to compensate
  - In 1996

Intel Pentium @ 133 MHz	$300	2×
Intel Pentium @ 200 MHz	$599	

  - In 2012

Intel Xeon X5650 @ 2.67 GHz	$1004	1.66×
Intel Xeon X5690 @ 3.47 GHz	$1660	

Real-life Performance? (2)

- The microkernel trade-off
  - Run-time performance
    - We have to pay more to compensate for the overhead

vs.

  - Run-time reliability
    - We have to pay less because of improved reliability
      - Fault isolation, restarting of faulty services
      - Formal verification

Microkernel Overhead as a Benefit

- When can an overhead be of any benefit?
  - Technically: Never
  - Paradigm shift
    - Plain function calls optimized and unbeatable for sequential code performance
    - What if sequential code execution cannot utilize the hardware effectively?
      - Multicore (many-core) architectures

Microkernel Overhead as a Benefit (2)

- Parallel algorithms
  - Manual decomposition into work queues
    - Fork-join
  - Parallel programming abstractions
    - Actors, agents, dataflow concurrency, continuations
    - Future objects, promises
  - Implementation level
    - Similar to asynchronous IPC and multiserver design
    - Sequential overhead, but improved throughout

Microkernel Overhead as a Benefit (3)

- HelenOS asynchronous IPC
  - Elimination of unnecessary context switches
    - Cooperative scheduling in user space
  - Elimination of data copying
    - Shared memory between communicating tasks
  - Kernel queuing with reasonable performance
    - Smart concurrent data structures (hash tables, B+ trees)
    - No complex access rights, small code footprint
  - Asynchronous nature suitable for parallel processing
    - Non-blocking operations
    - Concepts of future objects and promises

Open Challenges

- Benefits of monolithic systems
  - Easier global prediction of resource usage patterns
    - I/O caches and buffers
      - Across various subsystems (block layer, filesystem layer, directory layer, etc.)
      - Read-ahead heuristics
    - Simpler reaction to resource pressure conditions
      - Mild vs. aggressive cache flushing, graceful degradation
  - Easier scheduler interaction
    - Priority boosting for interactive tasks, etc.

Open Challenges (2)

- Resources in microkernel multiserver systems
  - Implicitly shared resources (and their state) scattered among isolated servers
    - No central point for caching, future usage prediction (read-ahead), resource pressure evaluation (out-of-memory)
  - Possible solutions
    - Polymorphic caching server
    - Distributed shared resource management

Conclusion

- The microkernel overhead is a fact
  - Inherent property of the microkernel design
  - A price paid for the improved reliability & design
- The way the microkernel is implemented dramatically affects the extent of the overhead
  - From two-fold or worse slowdown to only several percents

Conclusion (2)

- Clever asynchronous IPC can provide adequate throughput on multicore systems
  - Non-blocking, non-sequential programming
  - Parallel processing of requests
- The resulting trade-off
  - Cost of hardware vs. Cost of reliability
- Still challenges to be solved
  - Transparent shared resources management

Q&A

References

- **Jochen Liedtke**: *Improving IPC by Kernel Design*
- **Jochen Liedtke**: *On µ-Kernel Construction*
- **Hermann Härtig**, **Michael Hohmuth**, **Jochen Liedtke**, **Sebastian Schönberg**, **Jean Wolter**: *The Performance of µ-Kernel-Based Systems*
- **Jan Stoess**: *Towards Effective User-Controlled Scheduling for Microkernel-Based Systems*
- **Sebastian Ottlik**: *Reducing Overhead in Microkernel Based Multiserver Operating Systems through Register Banks*