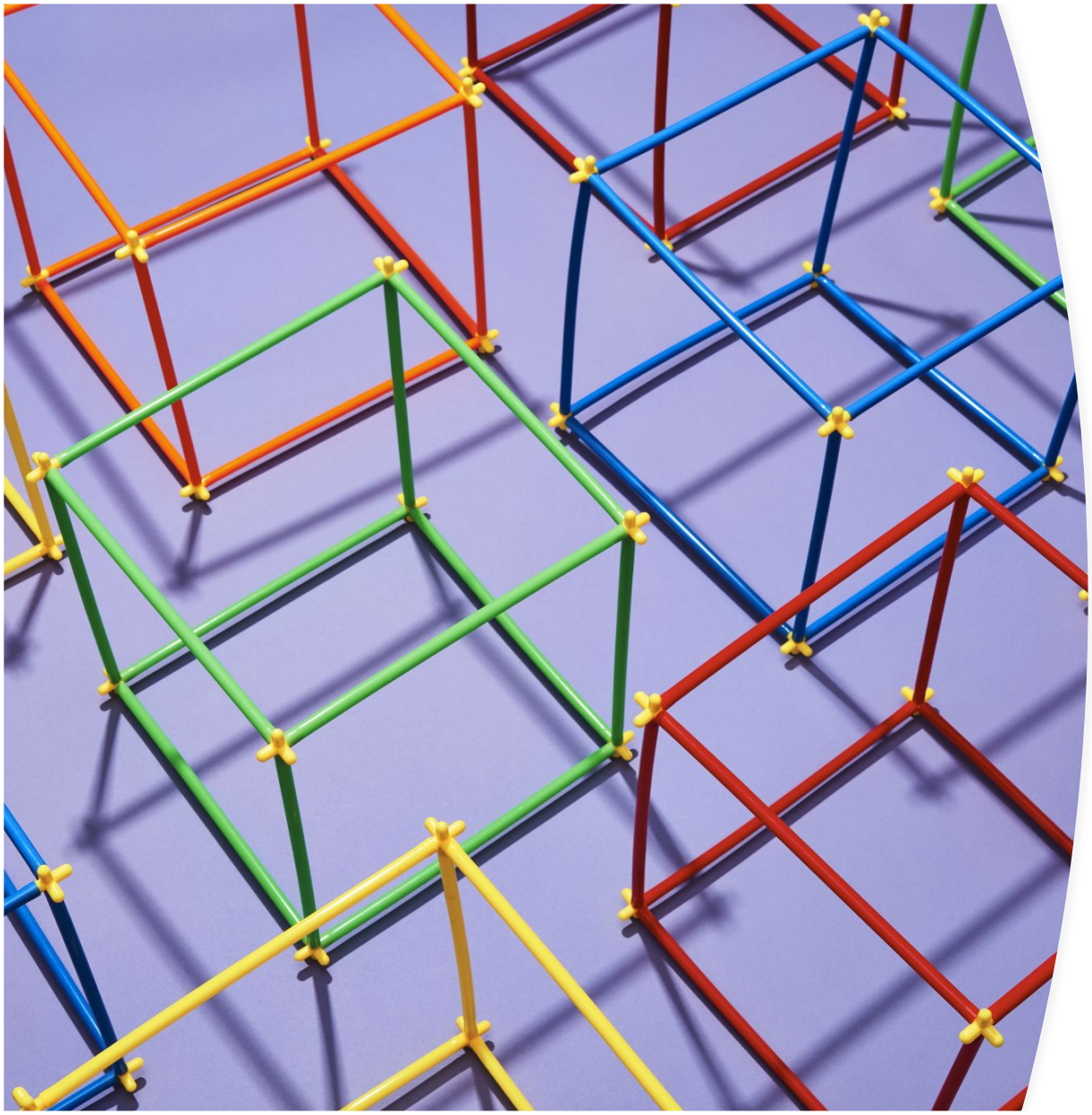
Embedded Operating System Architectures

Graziano Pravadelli



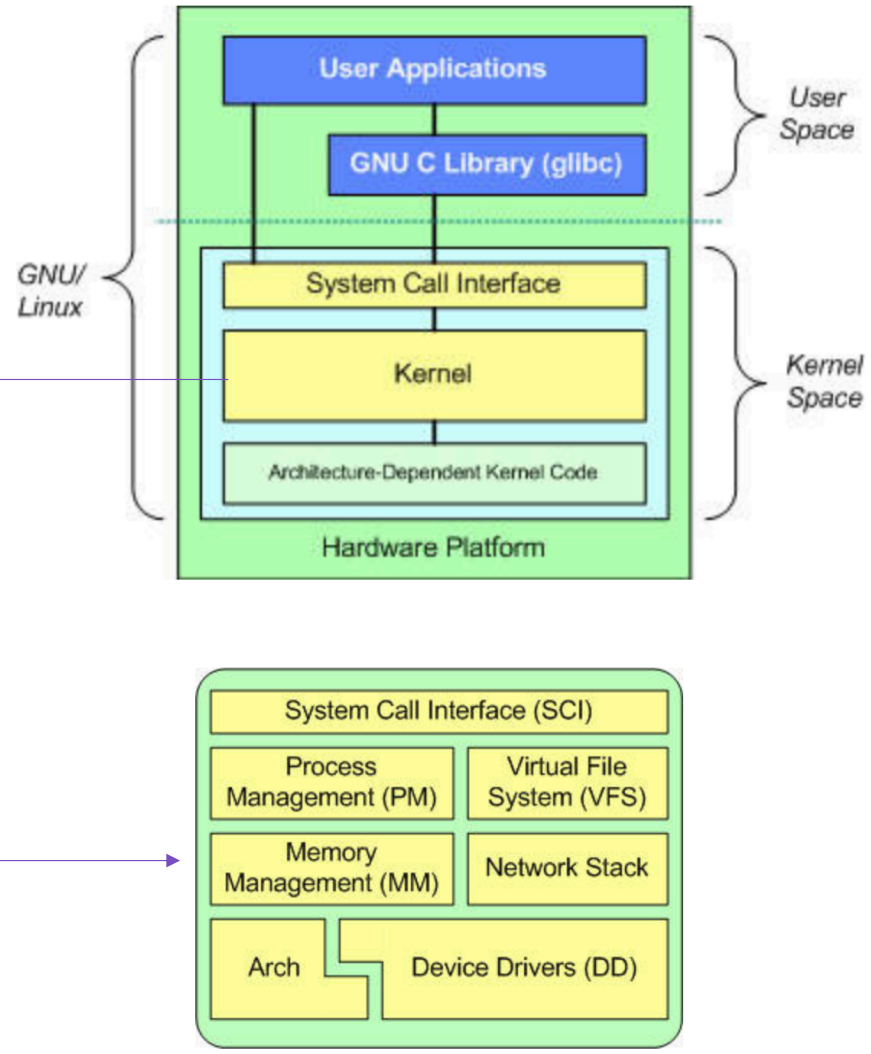
OS architecture

- Monolithic kernel
- Layered kernel
- Microkernel
- Exokernel

Monolithic kernel

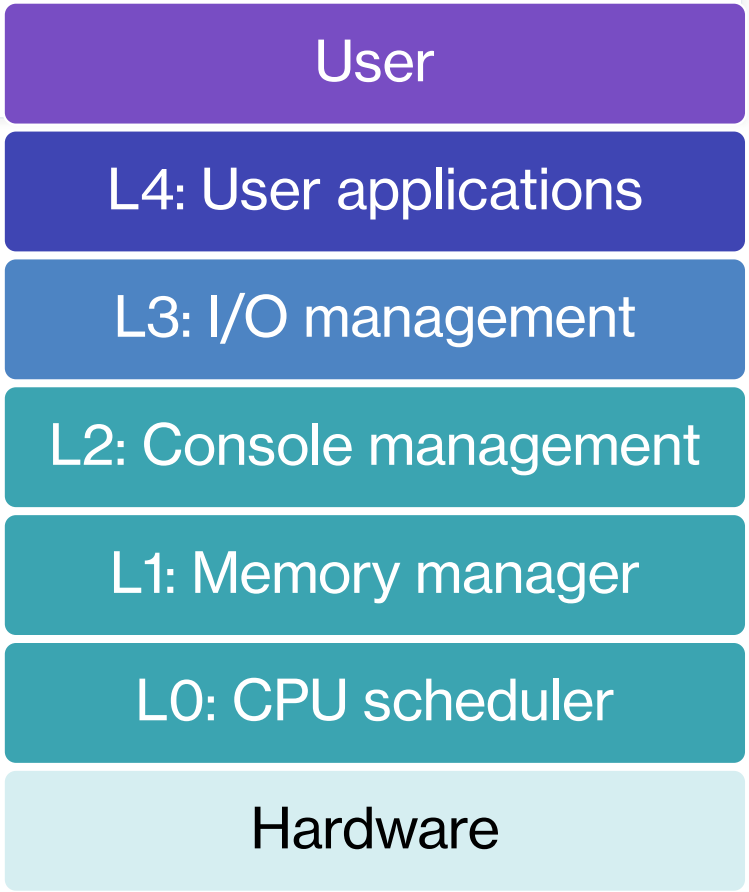
- A single binary file including all functionalities running in kernel mode:
  - CPU scheduling
  - Memory management
  - File system
- Pros
  - Very simple and fast
- Cons
  - Difficult to debug and maintain

Linux is a “monolithic” OS



Layered kernel

- Each layer implements a small set of functionalities
  - Well-defined interfaces among levels
  - Layer *i* uses services from layer *i-1* to implement functionalities for layer *i+1*
  - Implementation of layer *i* is hidden to the others
- Pros
  - Higher modularity
  - Changes/bugs affect only one level
- Cons
  - Difficulties in defining layers
  - Lower performance due to layer overhead



Microkernel

Essential services in the kernel

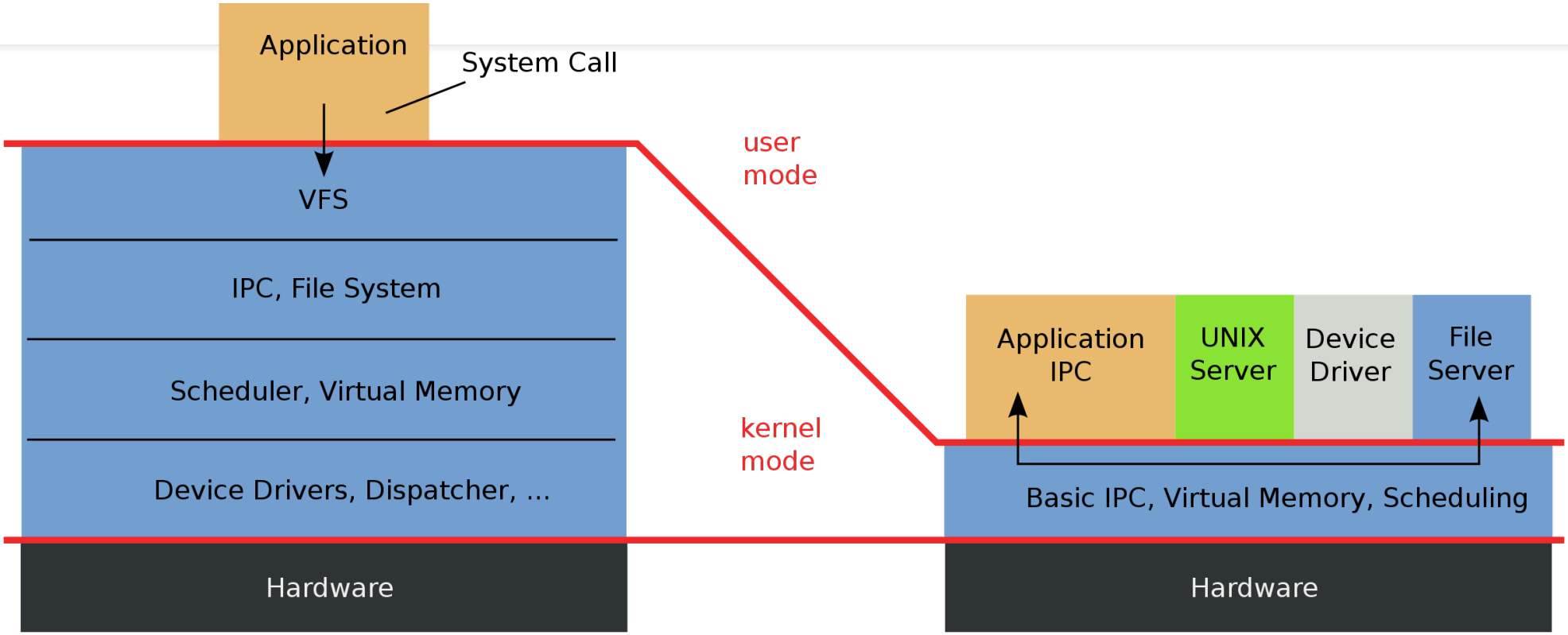
- Process management
- Memory management
- Communication

The rest is realized like user processes (clients)

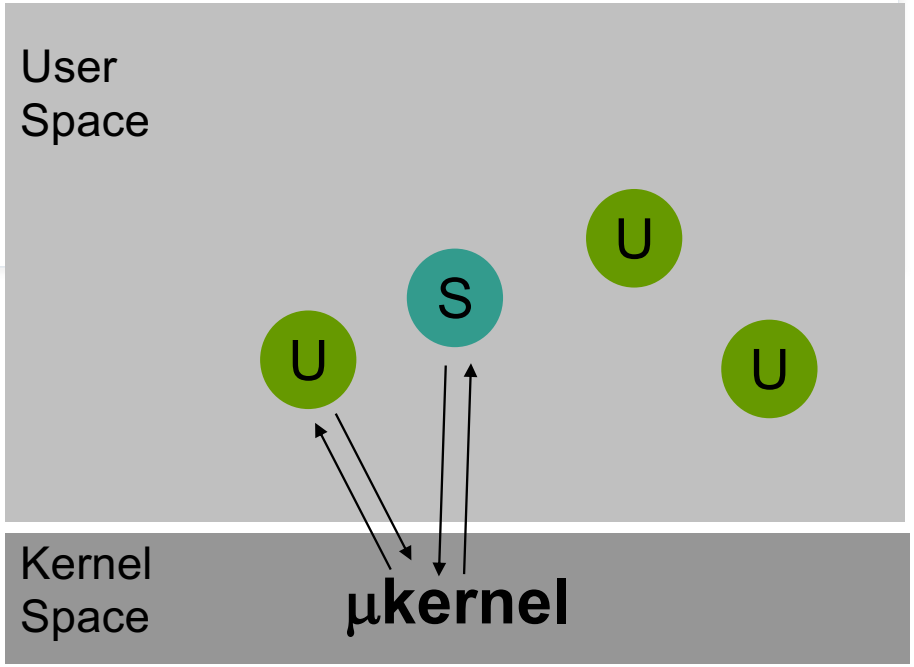
- Clients call servers for using services through message exchange

Monolithic Kernel based Operating System

Microkernel based Operating System



Operations



- External services implemented as server
- Interaction occurs between ***message exchange*** via the microkernel
  - The microkernel is essentially a “message manager” between external modules
- In practice, a client/server architecture on a single machine

Microkernel: pros and cons

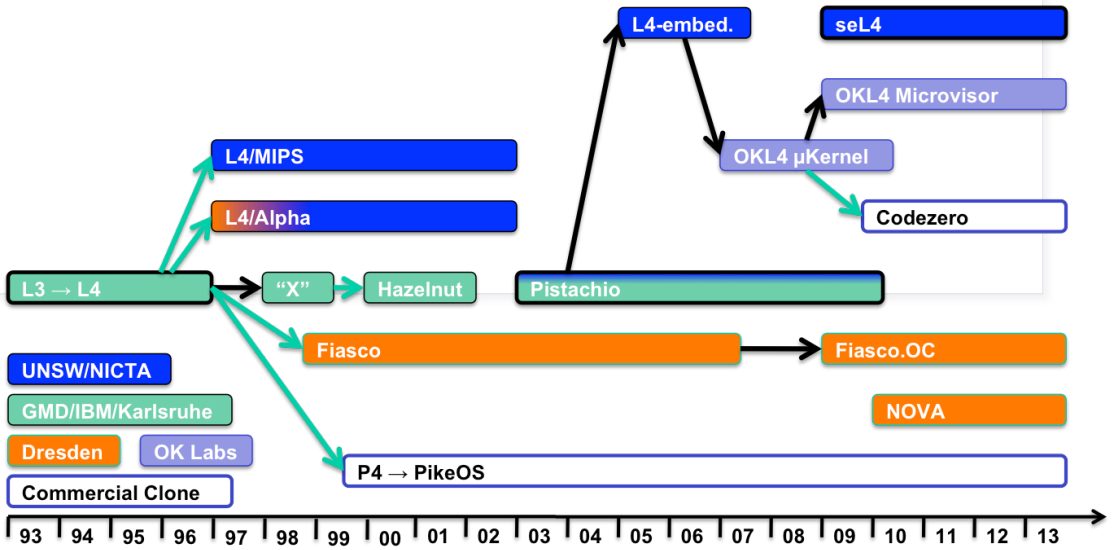
Advantages	Disadvantages

• More portable
  • Lower amount of changes required
• Easy to extend
  • New services without kernel changes
• More secure and reliable
  • Servers execute in user mode
• Essential services highly optimized
	• Low performance
	  • Overhead due to message exchange
	• Tendency to become monolithic
	  • E.g., Windows NT

Example: Mach

- Developed at CMU from 1985 to 1994
- At the basis of Apple's XNU kernel used in macOS, iOS, iPadOS, tvOS, and watchOS
- Mach 3.0 ~50x slower than native UNIX

Example: L4



- Defines a set of minimum primitives
  - Memory management
  - Threads and scheduling
  - IPC
- Complexity
  - 12KB code (Linux in 2020 ~27.8 millions of code lines)
  - 7 system calls
- Platforms
  - Ix86, Alpha, ARM, UltraSparc, ...

L4: Philosophy

“A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e., permitting competing implementations, would prevent the implementation of the system's required functionality”

J. Liedke

L4: Memory management

- Only address space management and memory protection
  - Mapping between virtual page and physical frame
  - Three primitives
    - **Grant**: a process provides pages to another process (address space)
    - **Map:** a process shares with another one its address space
    - **Flush:** a process requires address space previously given with grant or map
  - The address space is built recursively with these three primitives
    - Initially, a single space is held by a “base” process
- Memory management and paging are managed by out-of-kernel processes

L4: Thread

- Thread = execution unit inside an address space
- Thread has
  - Set of registers
  - Address space
  - Page fault manager
  - Scheduling parameters
  - ...
- The microkernel keeps the association between threads and address space

L4: IPC

- Microkernel provides support for the communication between address spaces
  - based on message passing
  - other IPC abstractions (and protection) built above the message scheme at user level
- Messages are also used for I/O and interrupt
  - I/O ports = part of the address space
  - Interrupt = HW messages
  - Kernel recognizes interrupts but do not manage them
    - It alerts the process that gets a grant of the corresponding space

L4: Performance

- L4 primarily designed for performance
- Machine specific rather than platform independend
- Written in assembly
- L4 ICP 20x faster than Mach IPC

L4: References

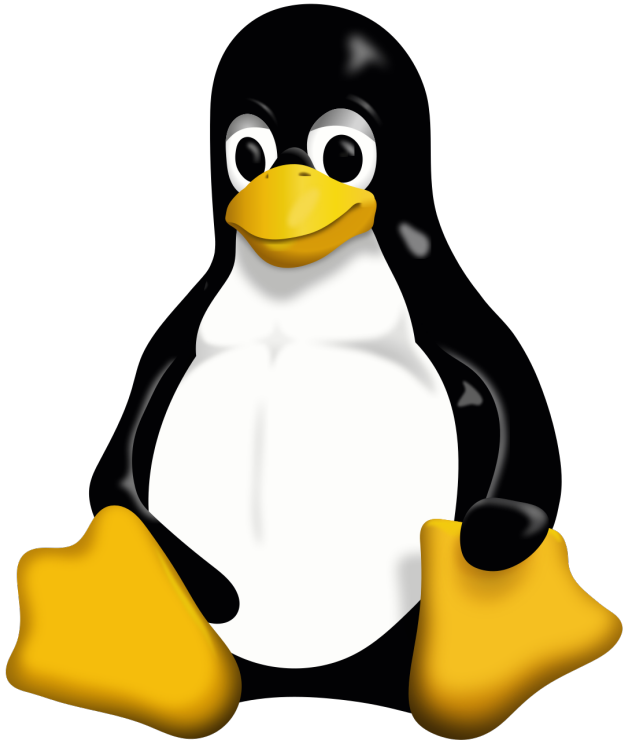
- J. Liedtke, “On µ-kernel construction”, Proceedings of ACM Symposium on Operating System Principles, 1995.
- A. Au, G. Heiser, “L4 user manual”, 1999.
- http://en.wikipedia.org/wiki/L4_microkernel_family

Alternatives to microkernel

- Loadable kernel modules
- Exokernel

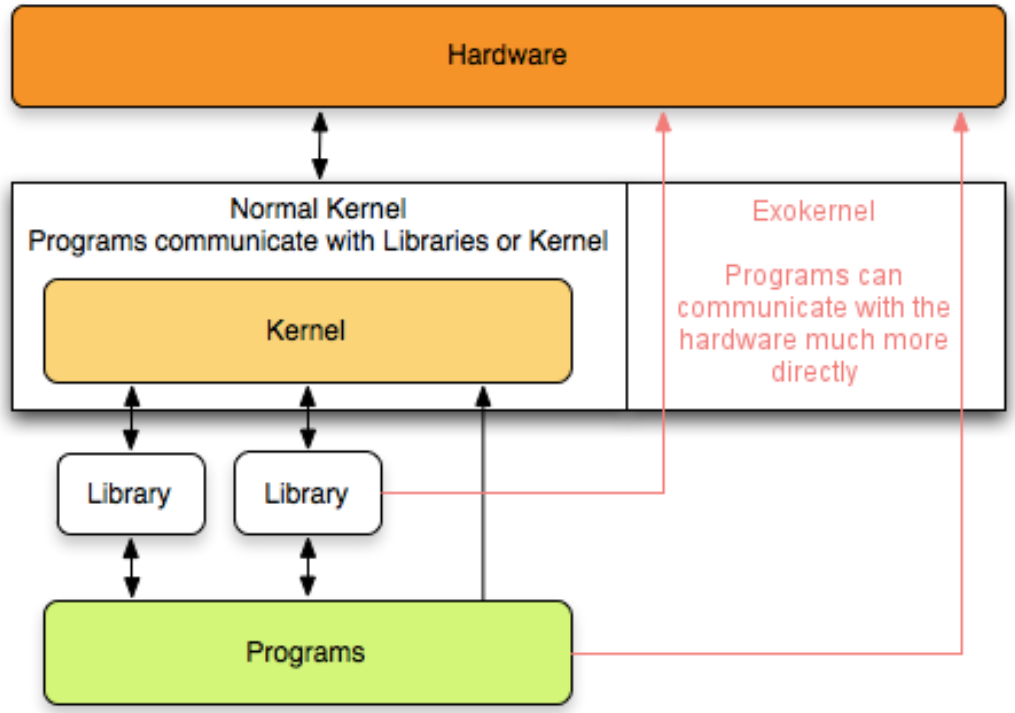
Loadable kernel modules

- Kernel includes basic components
- The rest is dynamically added, if necessary, at load or execution time
- Same address space like monolithic kernels
- Pros:
  - Module interfaces are well-defined like for layered kernels and microkernels
  - More efficient
    - No message exchange
  - More flexible
    - Modules can call each other



Exokernel

- Extensible OS developed at MIT in 1994
- Main idea:
  - Extremization of microkernel idea
  - No abstractions implemented in kernel space
  - The only kernel function consists in HW resources multiplexing between processes



Exokernel: Philosophy

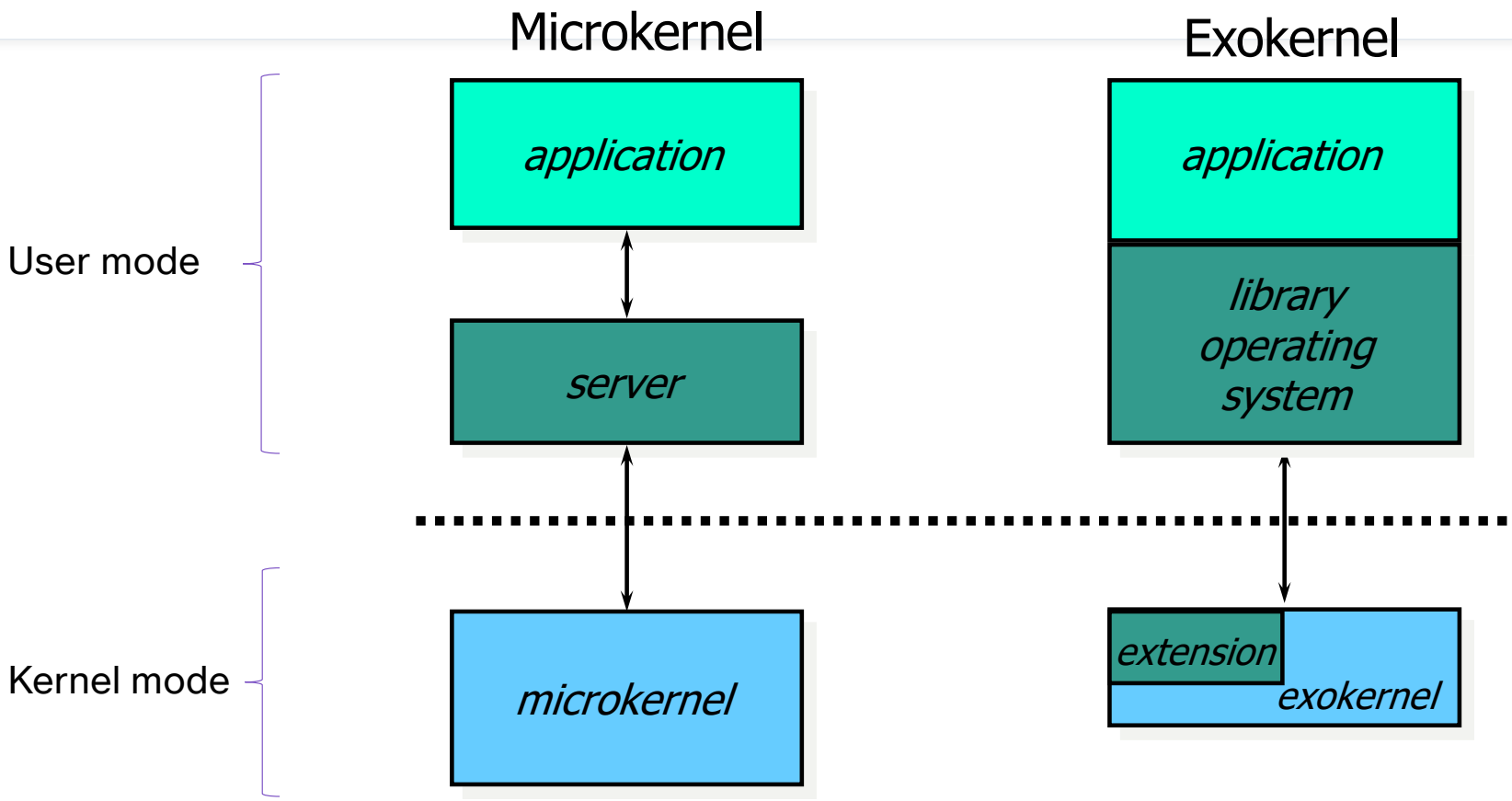
“Using innovations becomes a simple matter of linking in a new library rather than having to replace an entire system-wide operating system (and forcing all other users of the system to use it, and its bugs, in the process)”

D.R. Engler

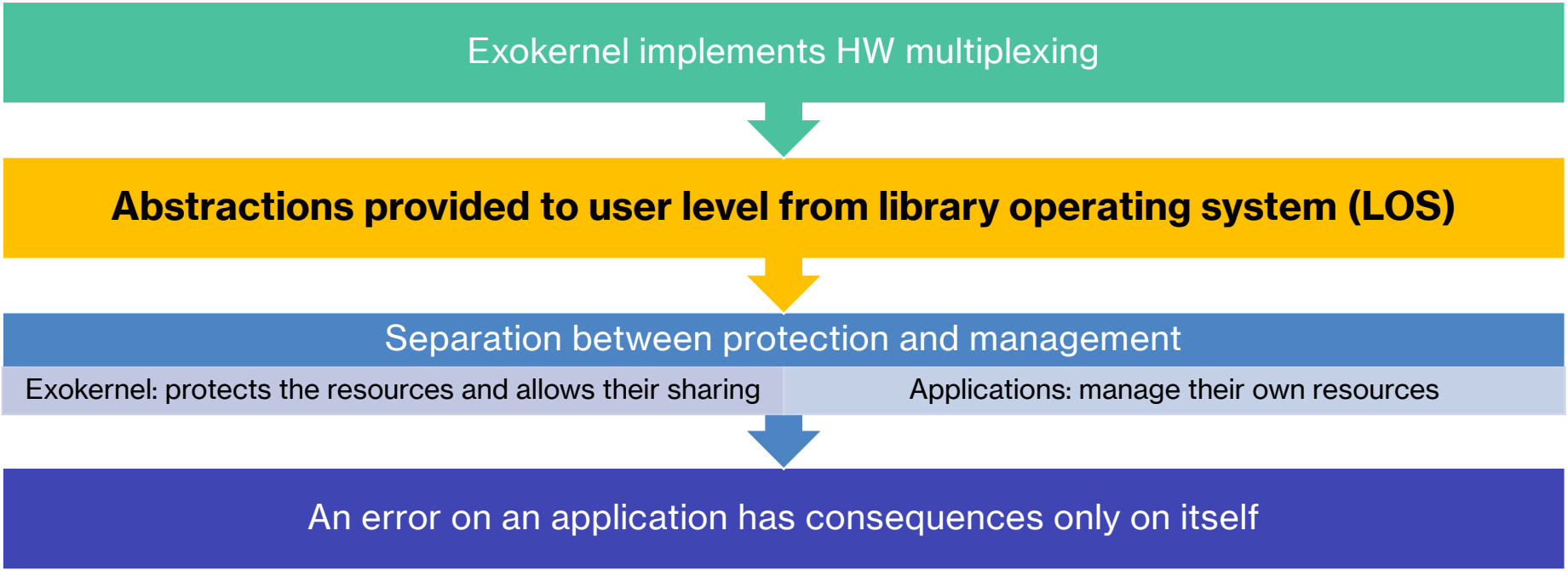
Exokernel vs. Standard OS

- OS basic functions
  - HW multiplexing
  - HW abstraction
    - Prevents the application of optimizations in specific cases
    - Discourages modifications to existing implementations
    - New abstractions can be only built above the existing ones
    - Expensive in terms of performance
- **Exokernel removes abstraction by adopting an application-level resources management**

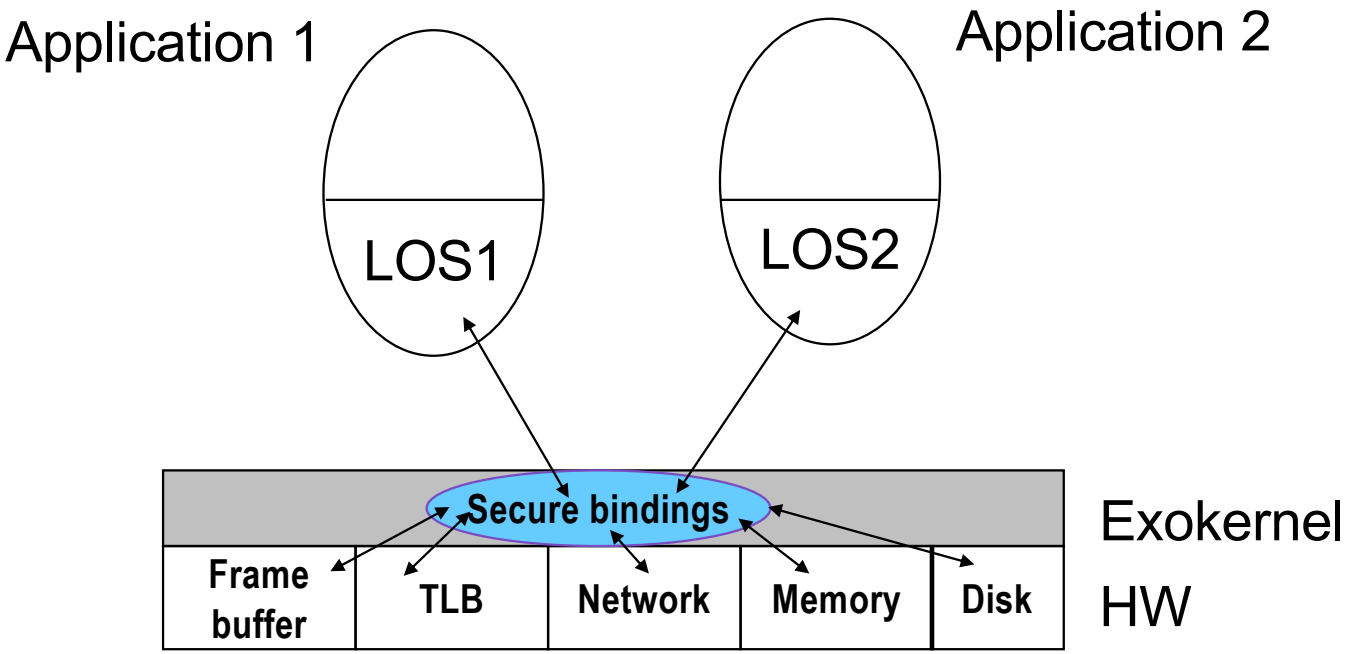
Exokernel vs. Microkernel



Exokernel: Design principles



Exokernel: Library Operating Systems



Exokernel: LOS basic principles

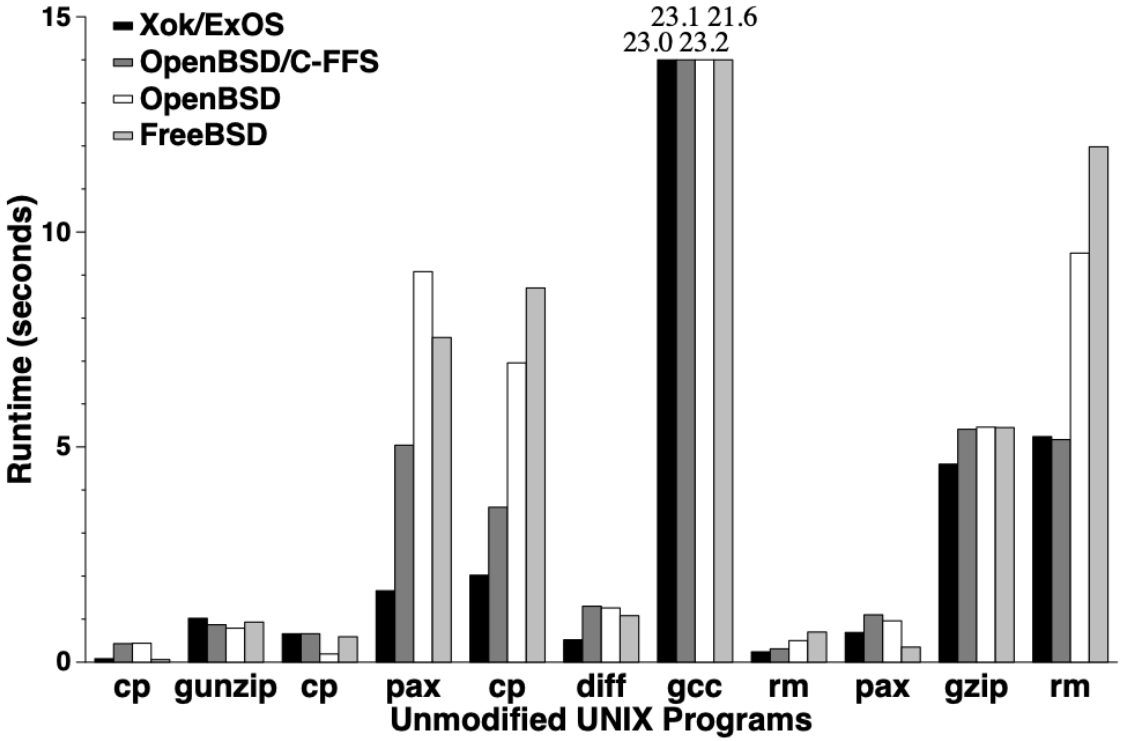
- Implement traditional abstractions
- Different LOS can coexist on the top of the same exokernel
- Safe allocation of resources:
  - *Secure binding:* LOS can bind resources to applications and manage related events (decoupling authorization from use of a resource)
  - *Visible resource revocation:* LOS actively participate to revocation of resources
  - *Abort protocols:* the exokernel break bindings of non-cooperative applications and informs the LOS

Exokernel: resource sharing

- Processor
  - It is a timeline where tasks can allocate intervals of time
  - A task can yield its time slice to another
  - Kernel notifies processor events (interrupts, exceptions) at the beginning/end of a time slice
- Memory
  - Kernel allocates physical pages and guarantee access only to tasks with “access” capability
  - Tasks can share pages by granting “access” capability

Exokernel: performance

- Very efficient, but:
  - how much expertise for writing code that benefit from exokernel?
- Not yet implemented in commercial OS



References

- Engler, “The Exokernel Operating System Architecture”, MIT, 1998
- Engler et al., “Exokernel: an OS for application-level resource management”, Symposium on Operating System Principles, 1995.
- Kaashoek et al., “Application performance and flexibility on exokernel systems”, Symposium on Operating System Principles, 1997.