

Networks and Operating Systems

Chapter 21: Networking Stacks and OS Architecture

(252-0052-00)

Gustavo Alonso & Timothy Roscoe

Frühjahrssemester 2010

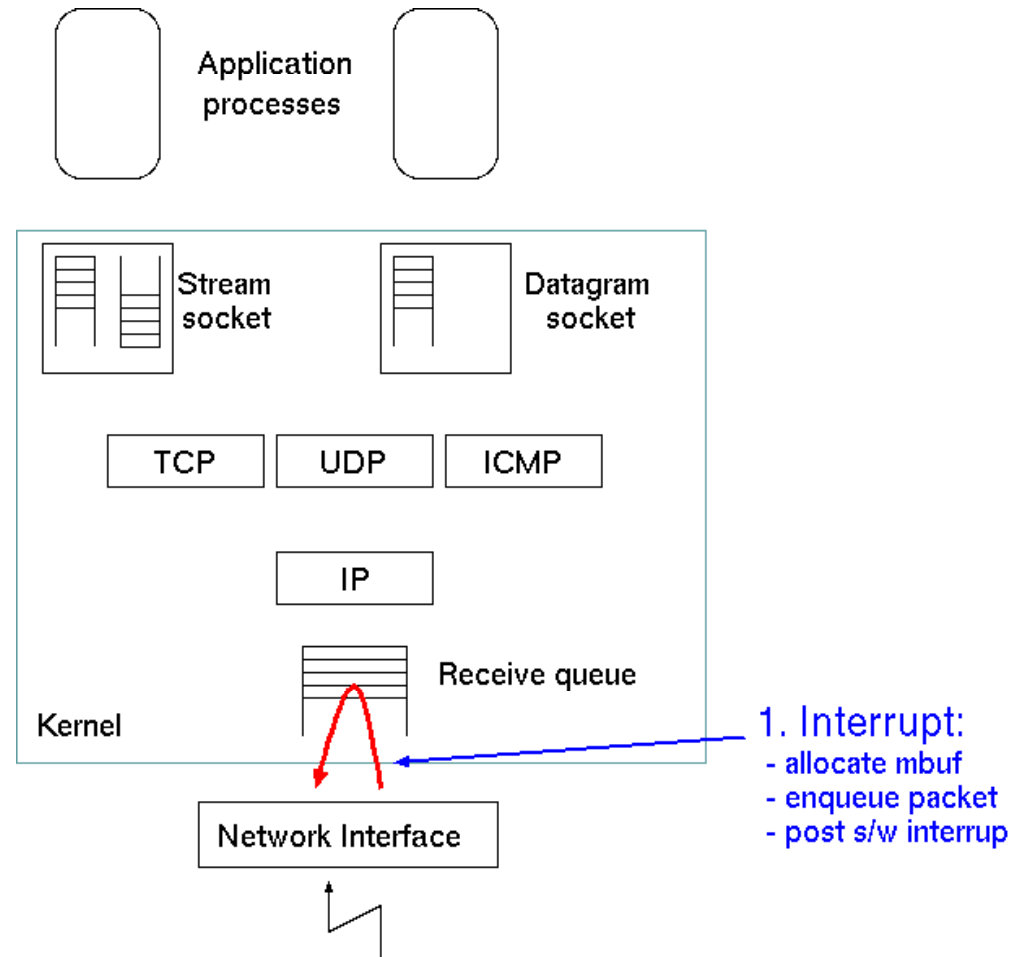
Network stack implementation

Networking stack

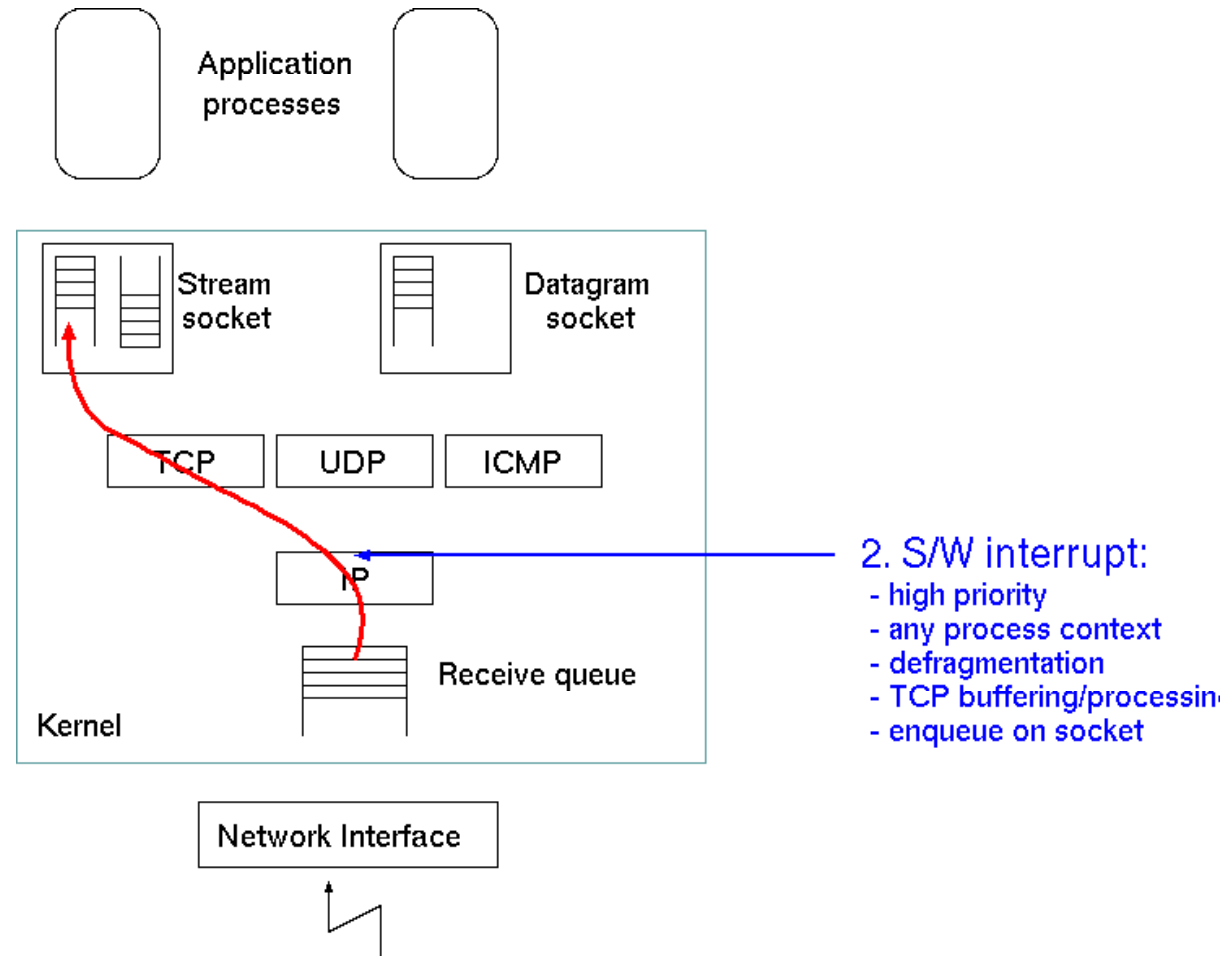


- Probably most important peripheral
 - GPU is increasingly not a peripheral
 - Disk interfaces look increasingly like a network
- But...
 - NO standard OS textbook talks about the network stack!
- Good references:
 - The 4.4BSD book (for Unix at least)
 - George Varghese: “Network Algorithmics” (up to a point)

TCP receive in Unix

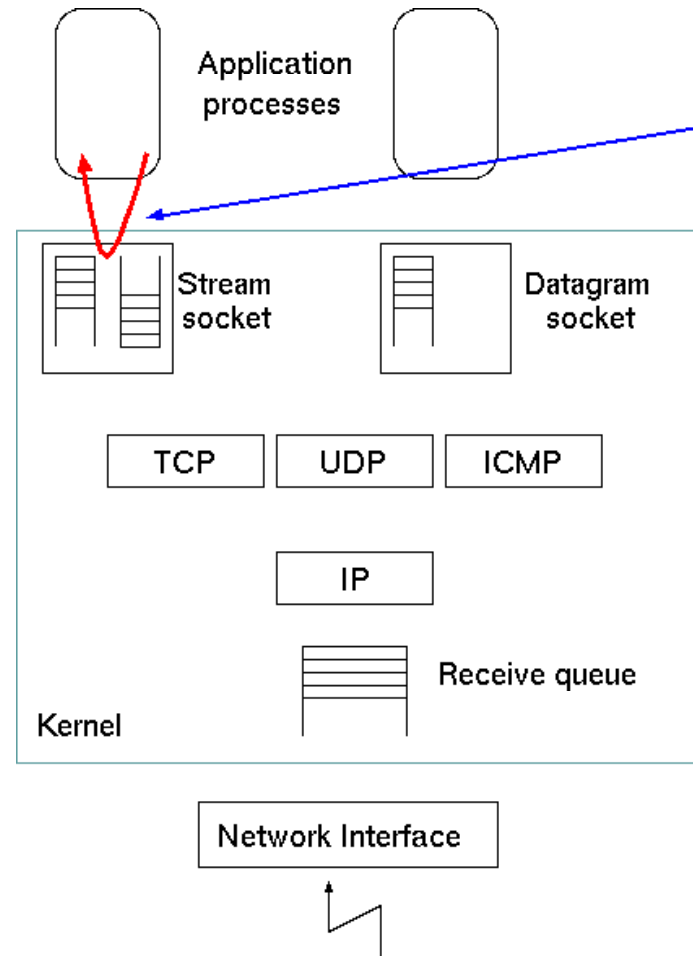


TCP receive in Unix



2. S/W interrupt:
- high priority
 - any process context
 - defragmentation
 - TCP buffering/processing
 - enqueue on socket

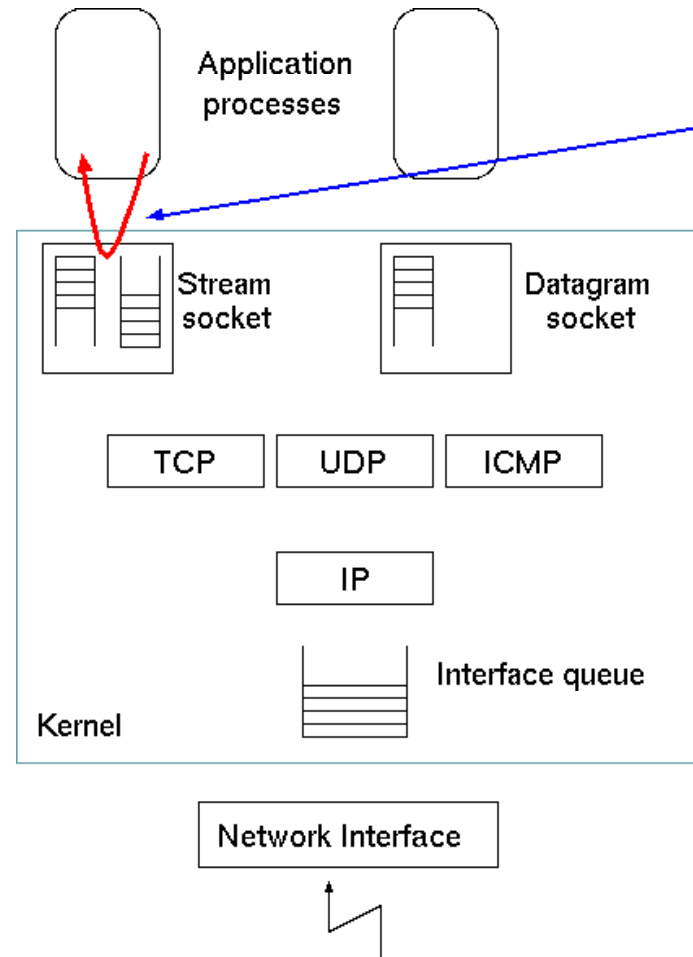
TCP receive in Unix



3. Application

- Copy from mbuf to user space
- application process context

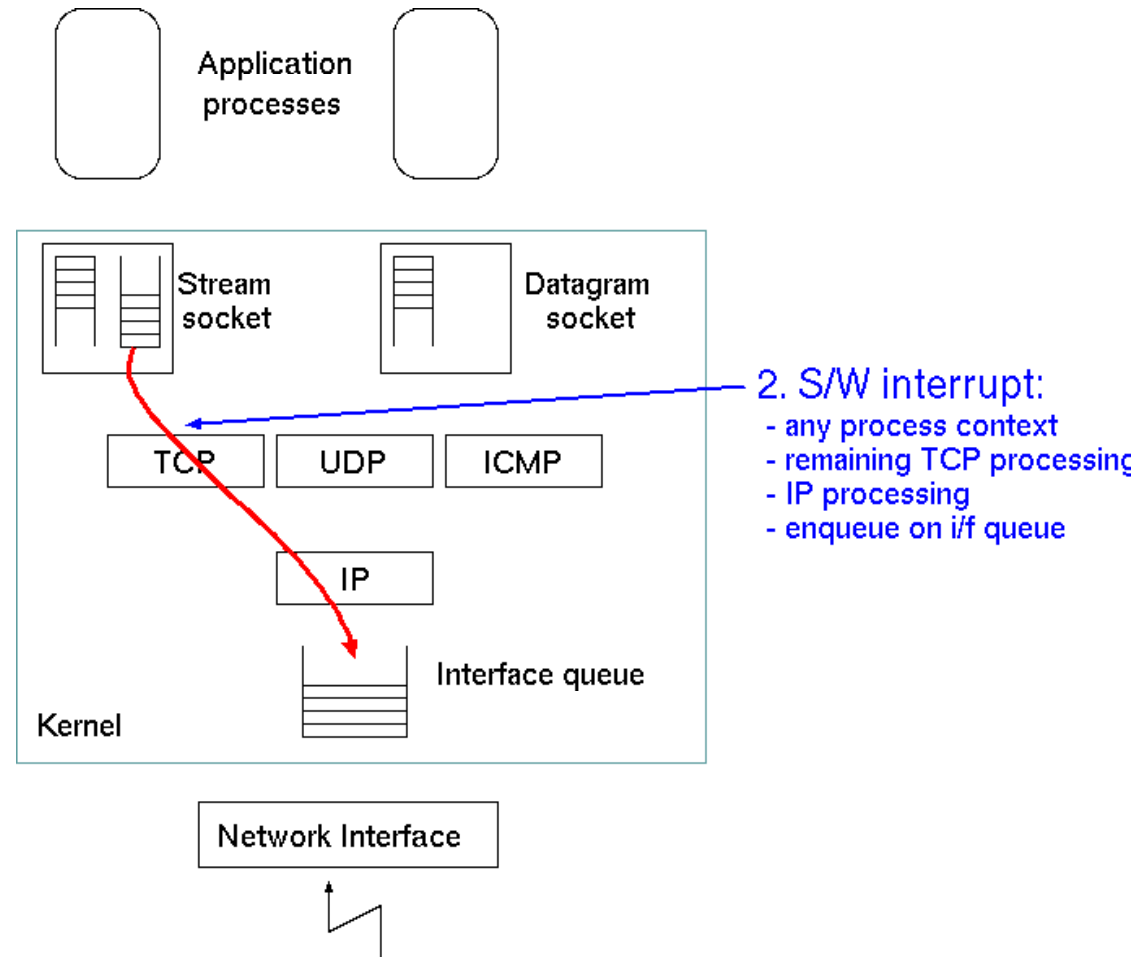
TCP send in Unix



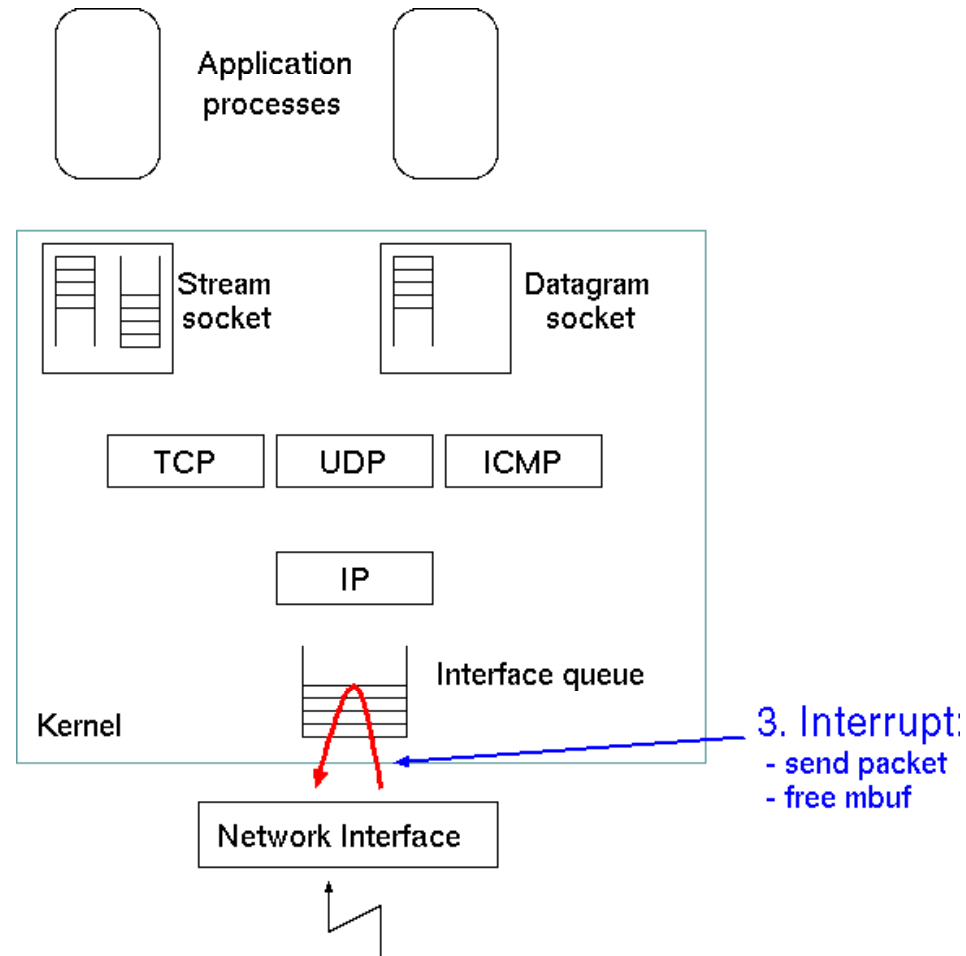
1. Application

- Copy from user space to mbu
- Call TCP code & process
- *Possibly* enqueue on socket queue

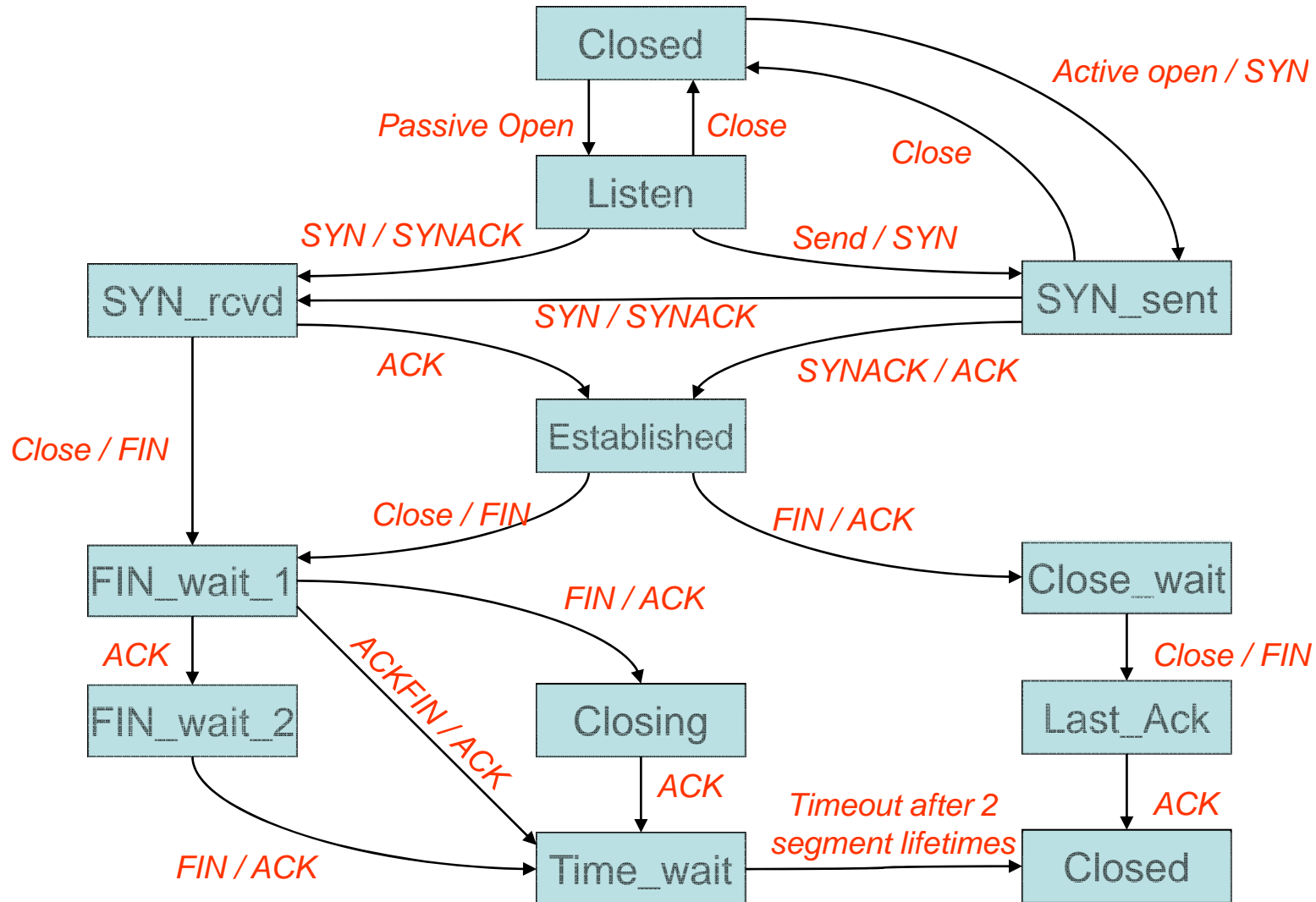
TCP send in Unix



TCP send in Unix



Recall TCP state machine



OS TCP state machine



- More complex! Also needs to handle:
 - Congestion control state (window, slow start, etc.)
 - Flow control window
 - Retransmission timeouts
 - Etc.
- State transitions triggered when:
 - User request: send, recv, open, close
 - Packet arrives
 - Timer expires
- Actions include:
 - Set or cancel a timer
 - Enqueue a packet on the transmit queue
 - Enqueue a packet on the socket receive queue
 - Create or destroy a *TCP control block*

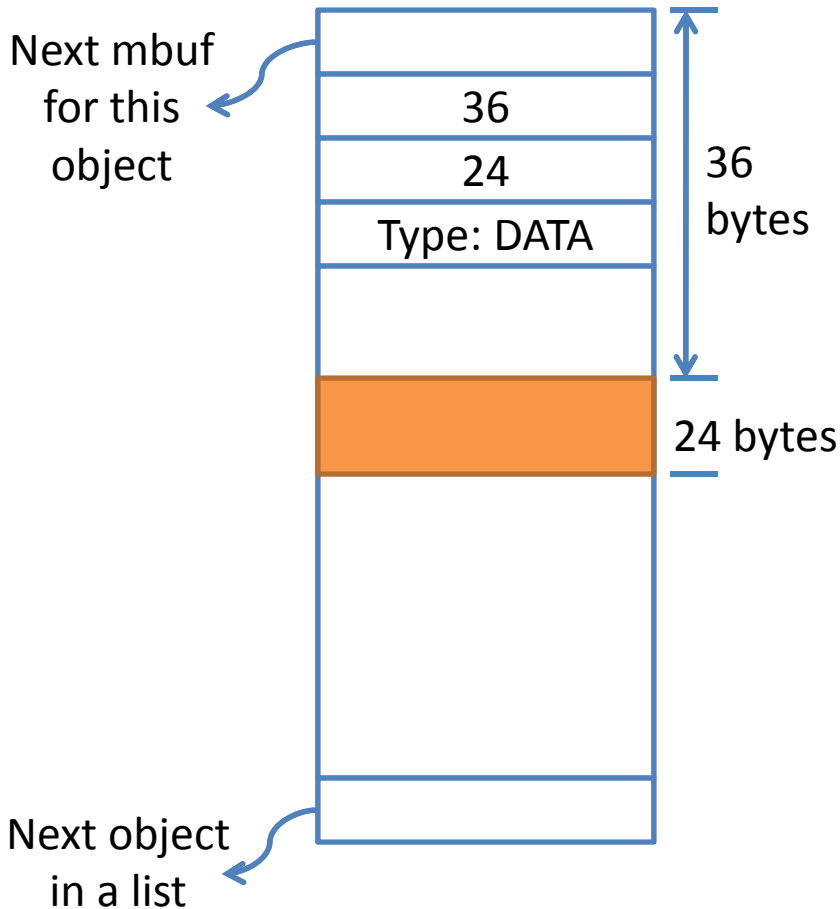
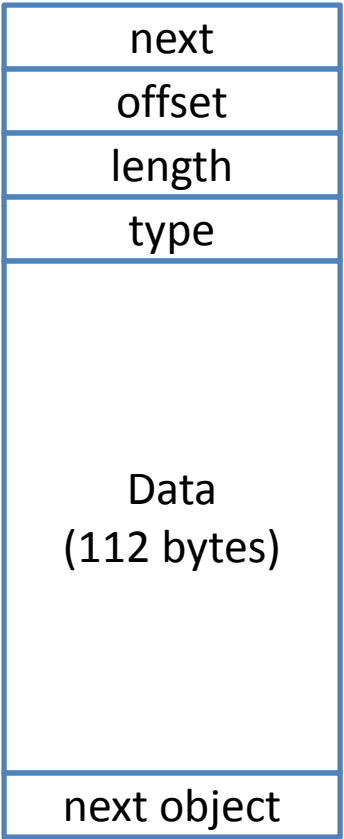
Memory mangement



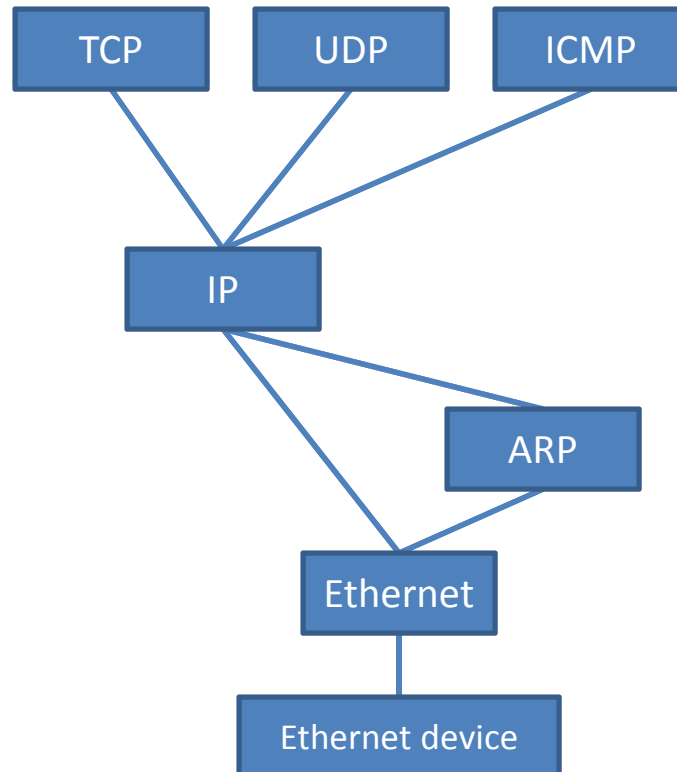
- Problem: how to ship packet data around
- Need a data structure that can:
 - Easily add, remove headers
 - Avoid copying lots of payload
 - Uniformly refer to half-defined packets
 - Fragment large datasets into smaller units
- Solution:
 - Data is held in a linked list of “buffer structures”

BSD Unix mbufs

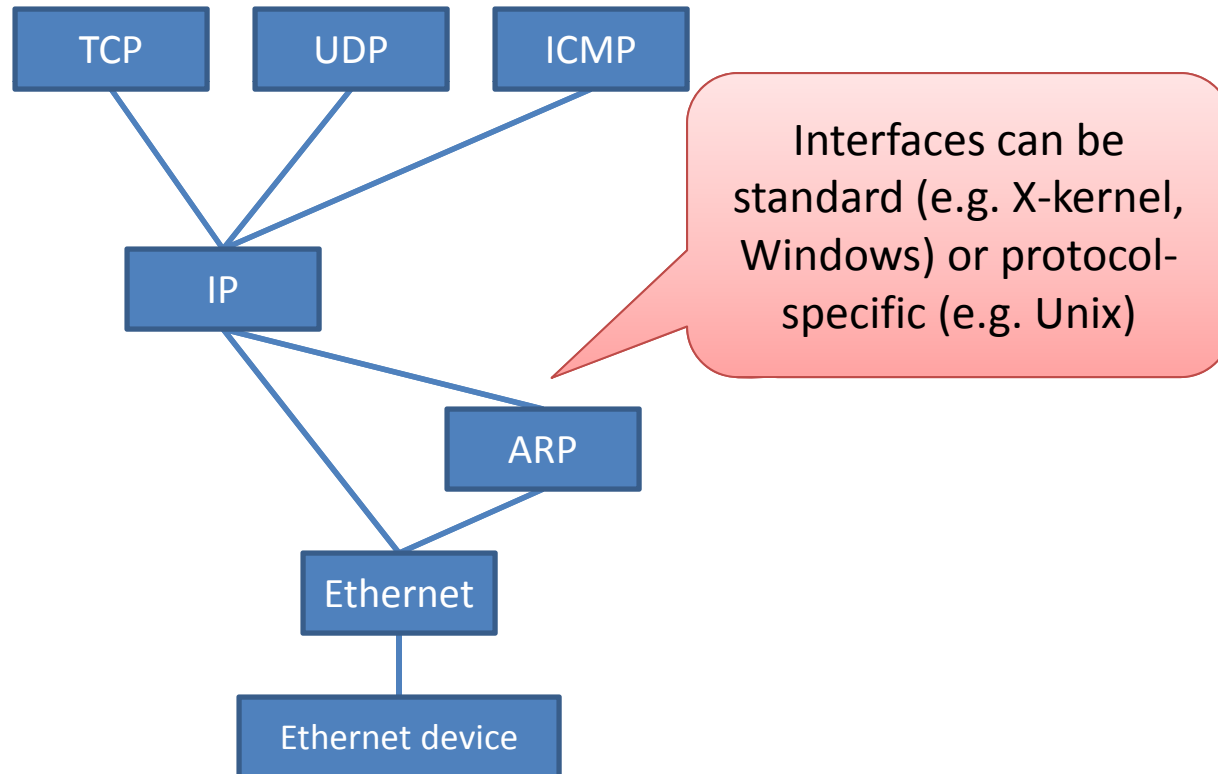
(Linux equivalent: sk_buffs)



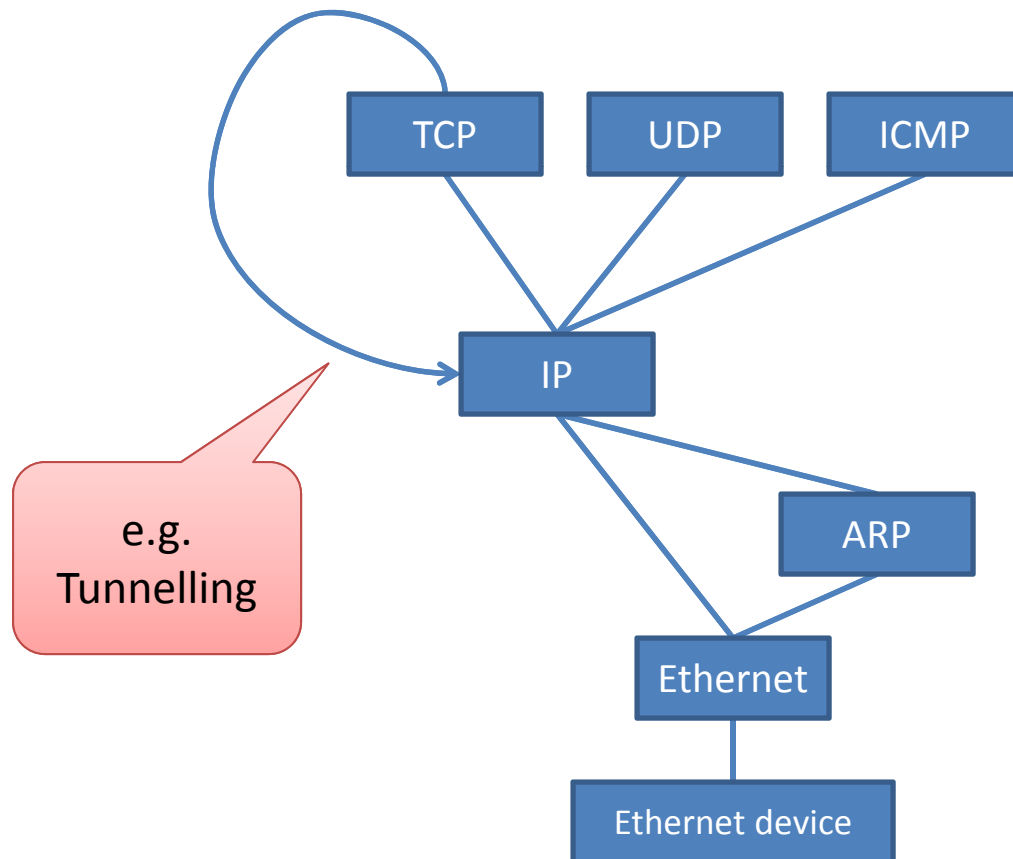
In-kernel protocol graph



In-kernel protocol graph



In-kernel protocol graph



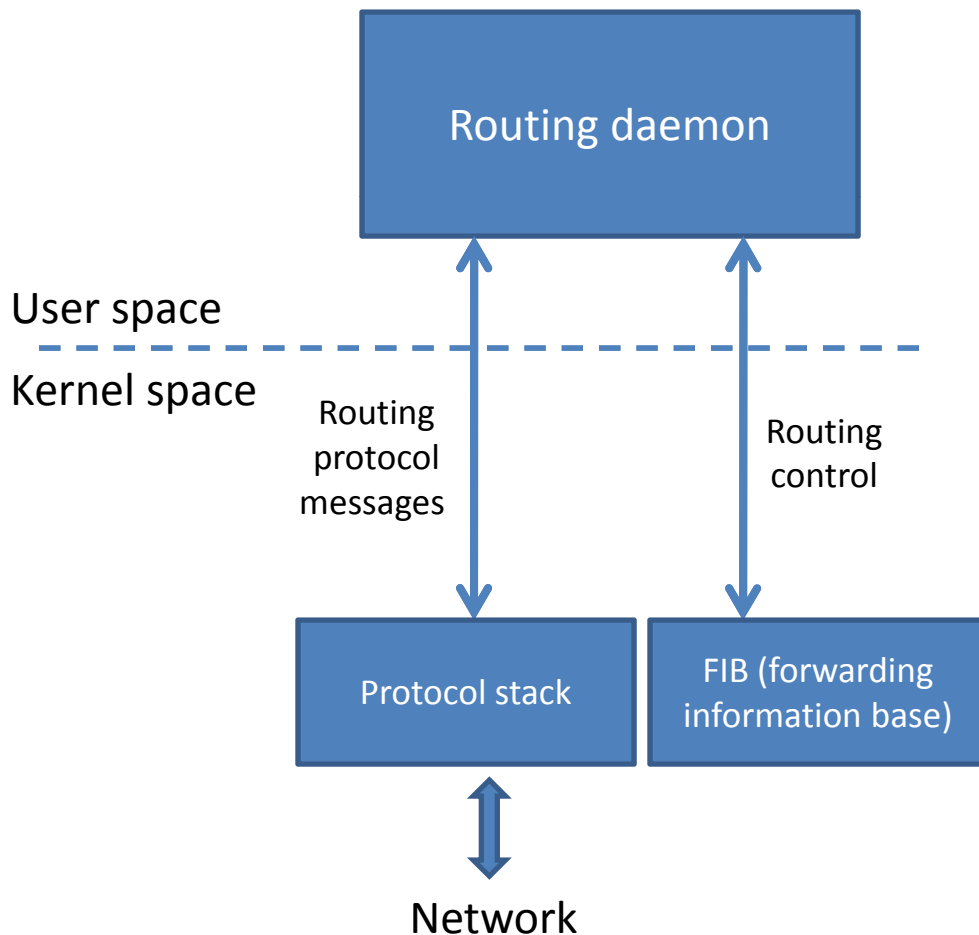
Protocol graphs



Graph nodes can be:

- Per-protocol (handle all flows)
 - Packets are “tagged” with demux tags
- Per-connection (instantiated dynamically)
 - Multiple *interfaces* as well as connections
 - Ethernet \leftrightarrow Ethernet \Rightarrow bridging
 - IP \leftrightarrow IP \Rightarrow IP routing

Routing



- OS protocol stacks include routing functionality
- Routing *protocols* typically in a user-space daemon
 - Non-critical
 - Easier to change
- *Forwarding* information typically in kernel
 - Needs to be fast
 - Integrated into protocol stack

Kernel Architecture

Operating system “architecture”



- Coarse-grained structure of the OS
 - How the complexity is factored
- Mapping onto:
 - Programming language features
 - Execution environment presented to applications
 - Address spaces
 - Hardware protection features (rings, levels, etc.)
 - Execution patterns (subroutines, threads,. coroutines)
 - Hardware execution (interrupts, traps, call gates)

Architectural models



- There are many, and they are *models*
 - Idealized, extreme view of how system is structured
 - Real systems always entail compromises
 - Hard to convey \Rightarrow it's good to build a few
- Think of these as tools for thinking about OSe
 - Each has its reasons
 - Solve particular problems at particular times
- In no particular order...

Monolithic systems



- Hardware provides time multiplexing
 - Interrupts
 - threads (sometimes)
- Language provides modularity & protection
 - Module calls
 - Inter-thread communication
- Earliest model for OS design
 - But remarkably persistent

Examples



More examples



Kernel-based systems



- Examples: Unix, VMS, Windows NT/XP/Vista
- Hardware enforces user vs. kernel mode
 - Machine in user space multiplexed into address spaces
- Kernel provides:
 - All shared services
 - All device abstraction
- Most common model today

Microkernels



- Examples: L4, Mach, Amoeba, Chorus
- Kernel provides:
 - Threads
 - Address spaces
 - IPC
 - **And nothing else!**
- All other functionality in **server processes**
 - Device drivers
 - File systems
 - Etc.
- Instead of syscalls, applications send IPC to servers

Microkernel history



- Early systems (1970s)
 - Michigan Terminal System (MTS)
 - CMU Hydra/C.mmp
- Separation of **policy** (user space) and **mechanism** (kernel)
 - Device drivers, pagers, etc. in user space
- Lacked the catchy “microkernel” buzzword

Microkernel history



- “1st Generation” microkernels
 - Amoeba, Mach, Chorus
- Manage complexity
 - Modularity through processes
- Motivations:
 - Ease of development / mgmt of complexity
 - Flexibility of policy
 - Security (small TCB) & Reliability (small kernel)
- Disastrous performance
 - Many more IPC calls than a monolithic kernel
 - 100 μ s IPC, independent of hardware

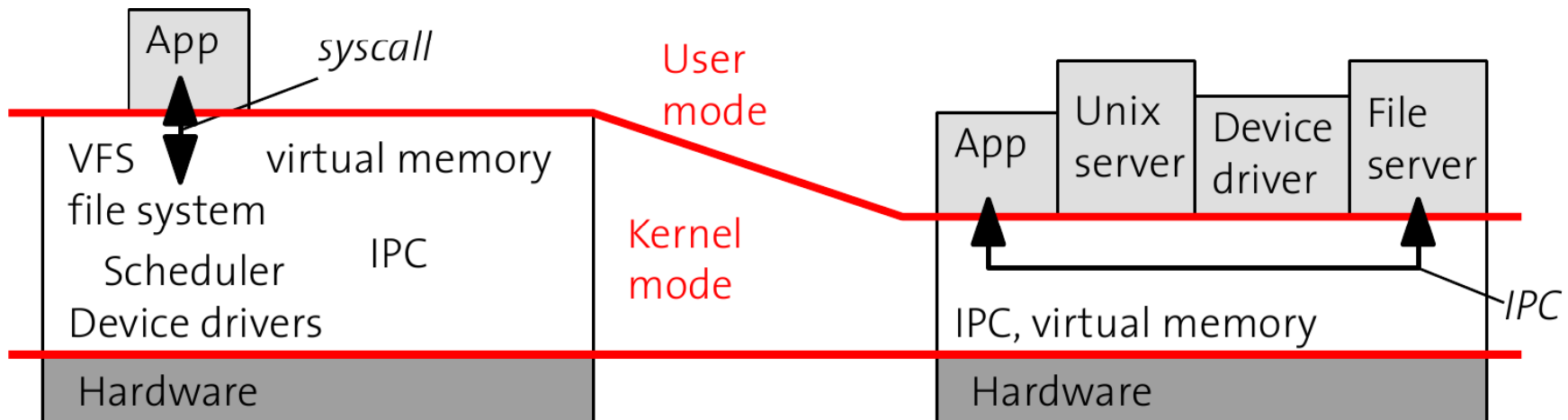
Microkernel history



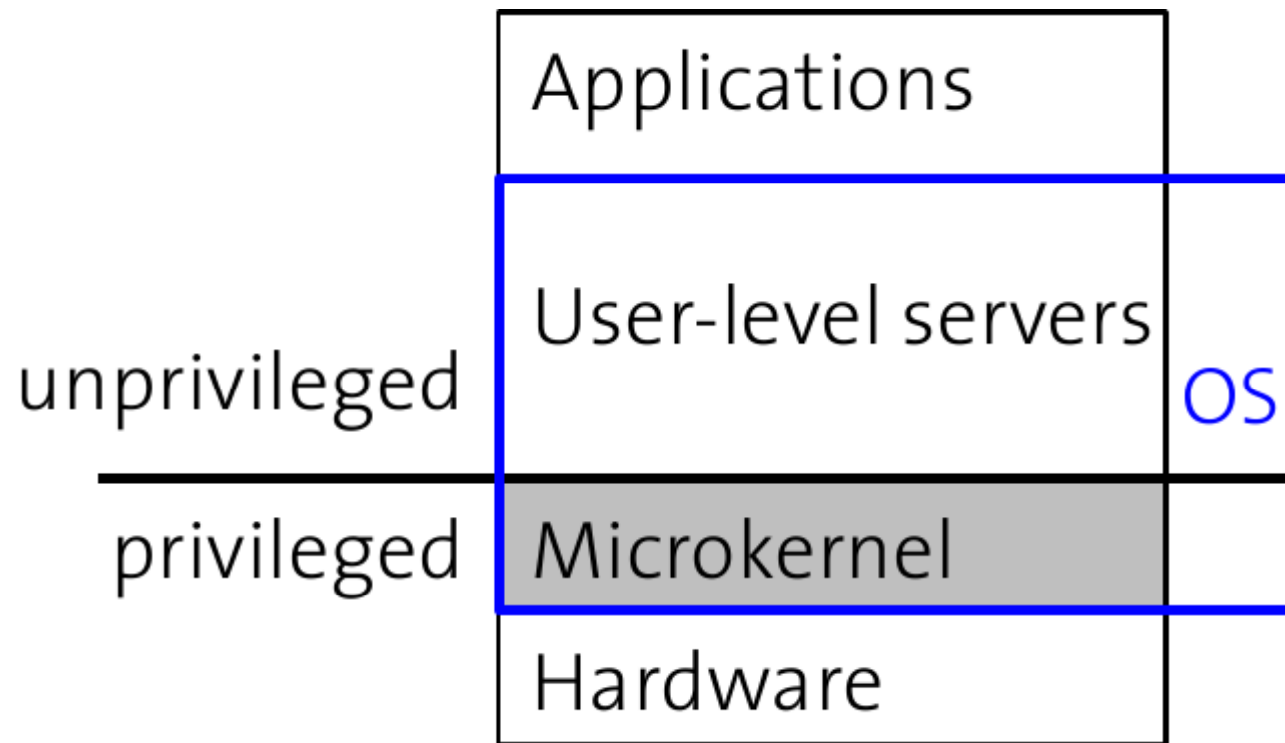
- Claim: μ kernels slow due to bad design
 - Liedtke, SOSP 95
 - Complex API, too many features
 - Poor design and implementation
 - Large cache footprint \Rightarrow memory bandwidth limited
- L4 kernel (1993-1995)
 - All in assembler, minimal functionality
 - Threads, address spaces, IPC, err, that's it.
 - Very small \Rightarrow fits in cache \Rightarrow CPU cycle limited
 - 20 x IPC performance of Mach

Microkernel idea

- Small size \Rightarrow small cache footprint \Rightarrow fast



Microkernel \neq complete OS

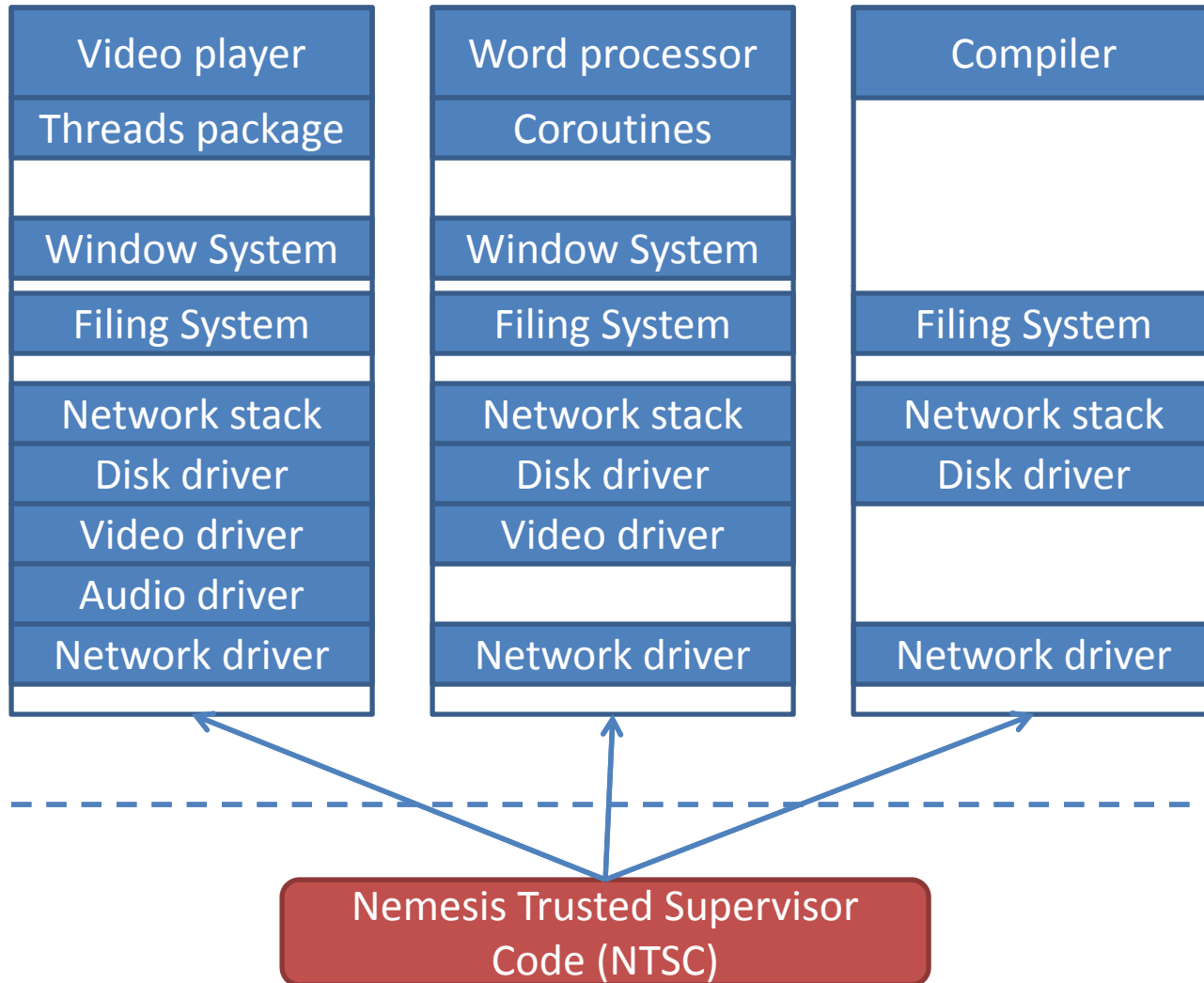


Exokernels

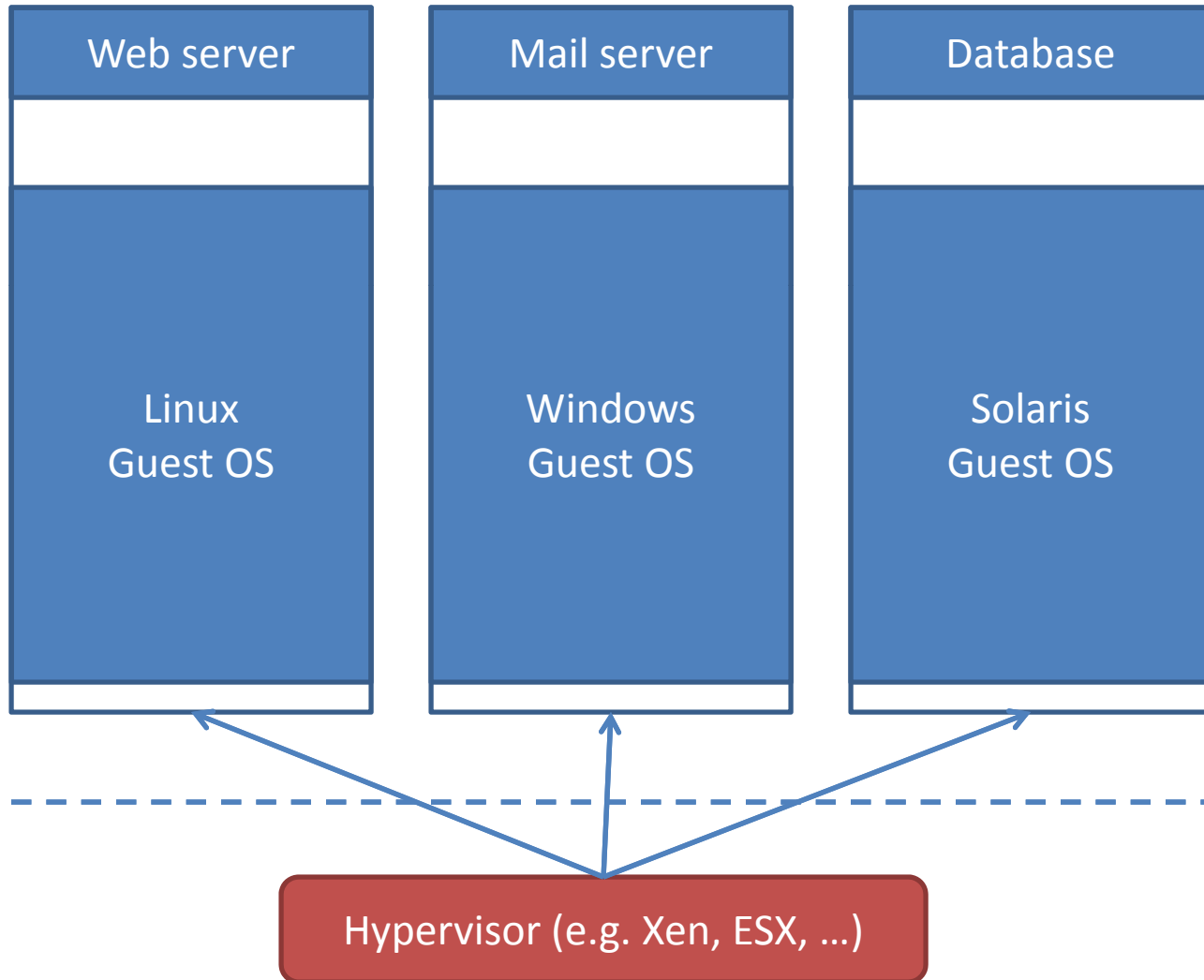


- Examples: Exokernel, Nemesis, [Xen, ESX]
- Kernel provides minimal multiplexing of h/w
- All other functionality in **userspace libraries**
 - Unlike microkernels, where this in **servers**
 - “LibraryOS” concept
- Enables:
 - Strong **isolation** between applications
 - High degree of application-specific **policies**

Example



Hypervisors



Multiprocessors

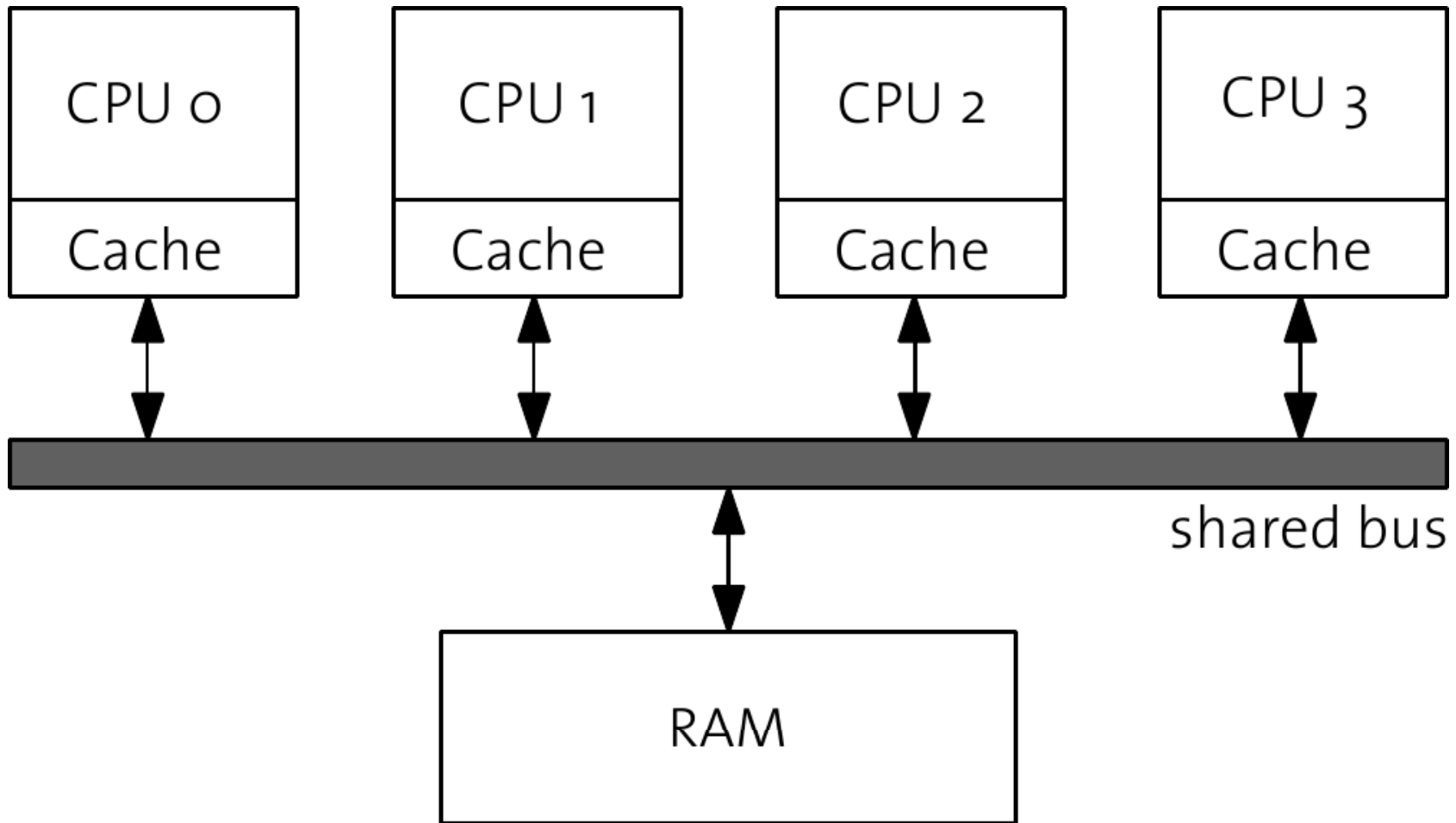
Multicore and Multiprocessing



- This course has said little about multiprocessors
 - Until recently, the ideas applied equally to uni- and multiprocessors
 - Just insert spinlocks as appropriate 😊
- Today, newly everything is a multiprocessor
 - Core counts increasing with Moore's Law
 - Perhaps new ideas are needed...

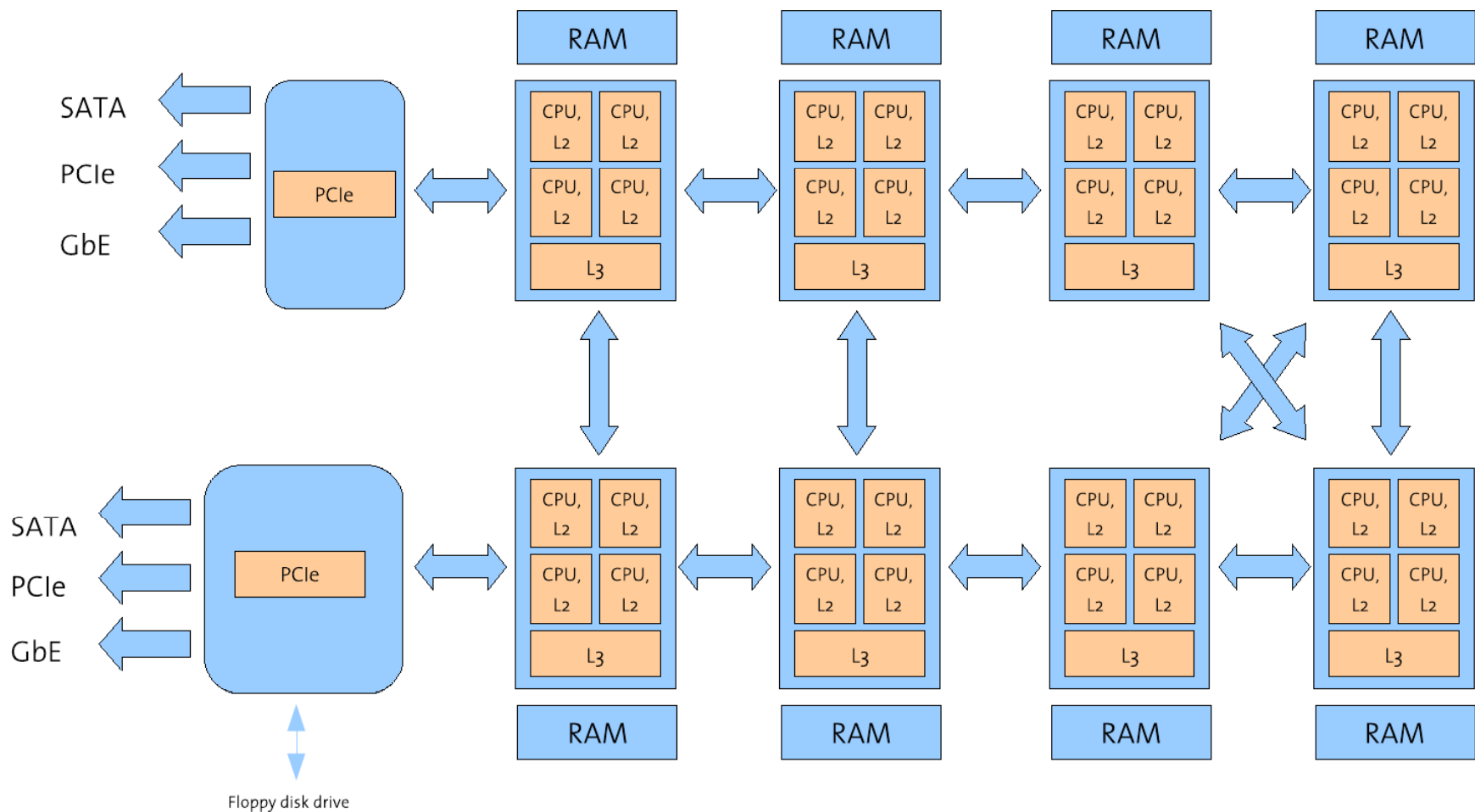
Old model: SMP

Symmetric Multiprocessor



NUMA

Non-uniform Memory Access

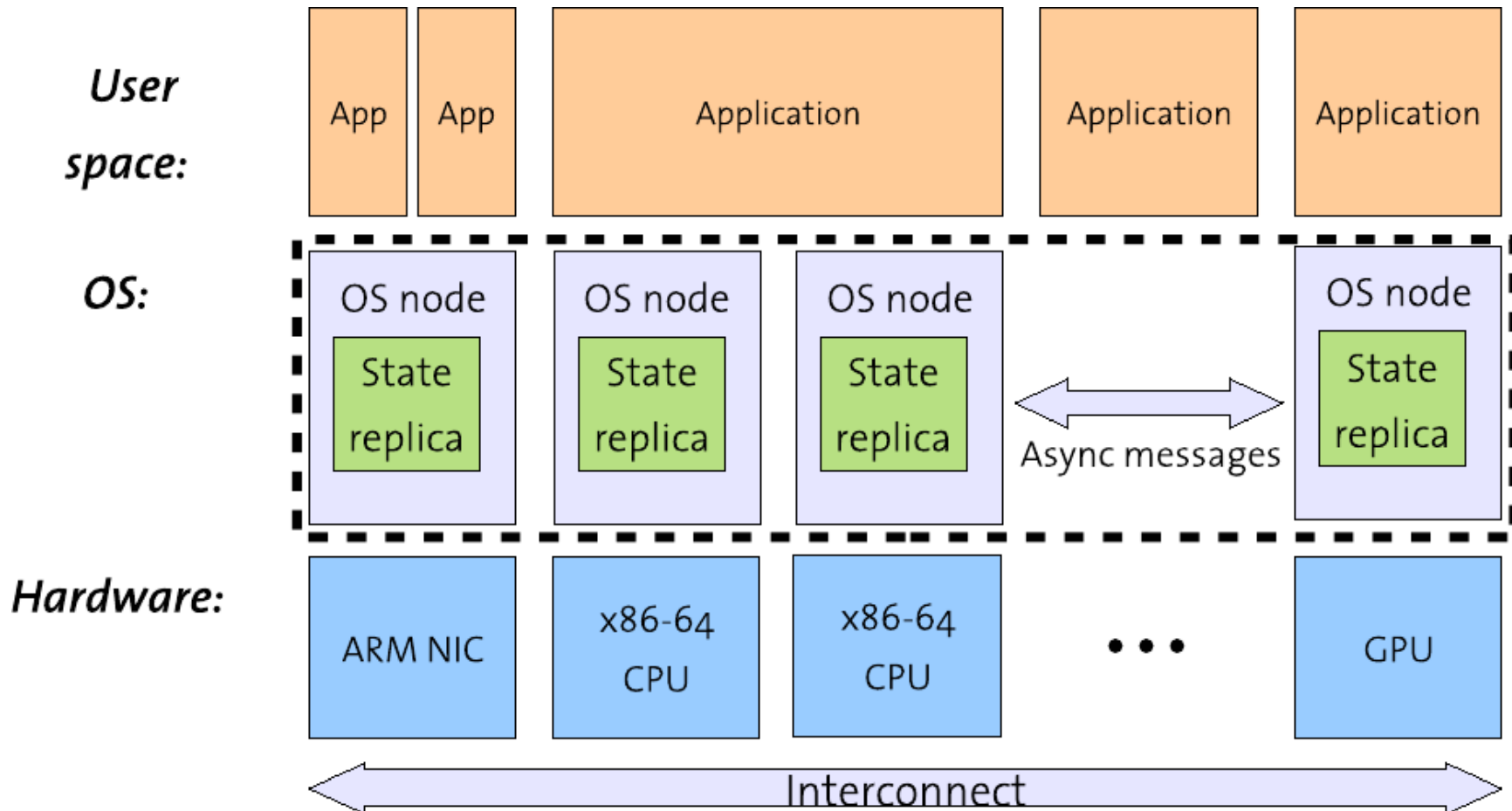


The problem: data sharing

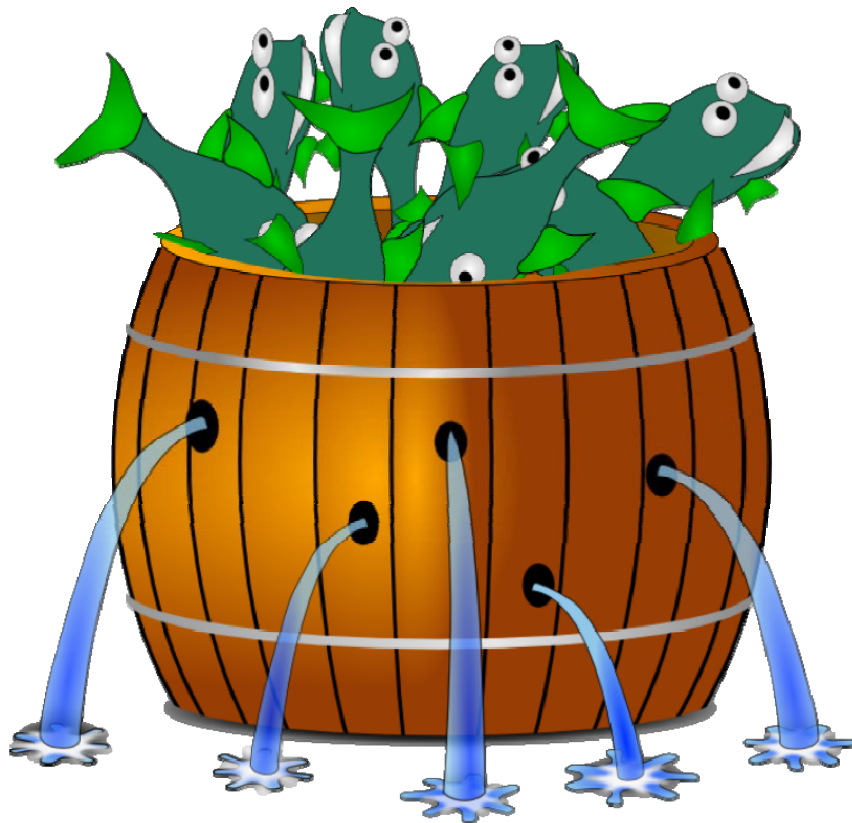


- Locks or other synchronization **can** scale
- Data sharing most likely **cannot**
 - Cache-lines must be moved between cores
 - Slow: stalls processor, congests interconnect
- Objective: manage and reduce ***communication***
 - Can't take shared memory for granted!
 - View inside of machine as a ***network*** itself
 - Build the OS as distributed system using messages

Multikernel



Barrelfish



- Our multikernel project
- Joint work with Microsoft Research
 - Open source
 - Hosted at ETHZ
- Runs on lots of fun hardware...
- Talk to me about projects if you're interested!

Thank you!
You've been a wonderful audience!
Good luck in the exam...