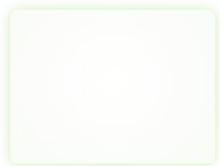
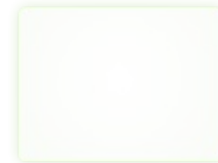


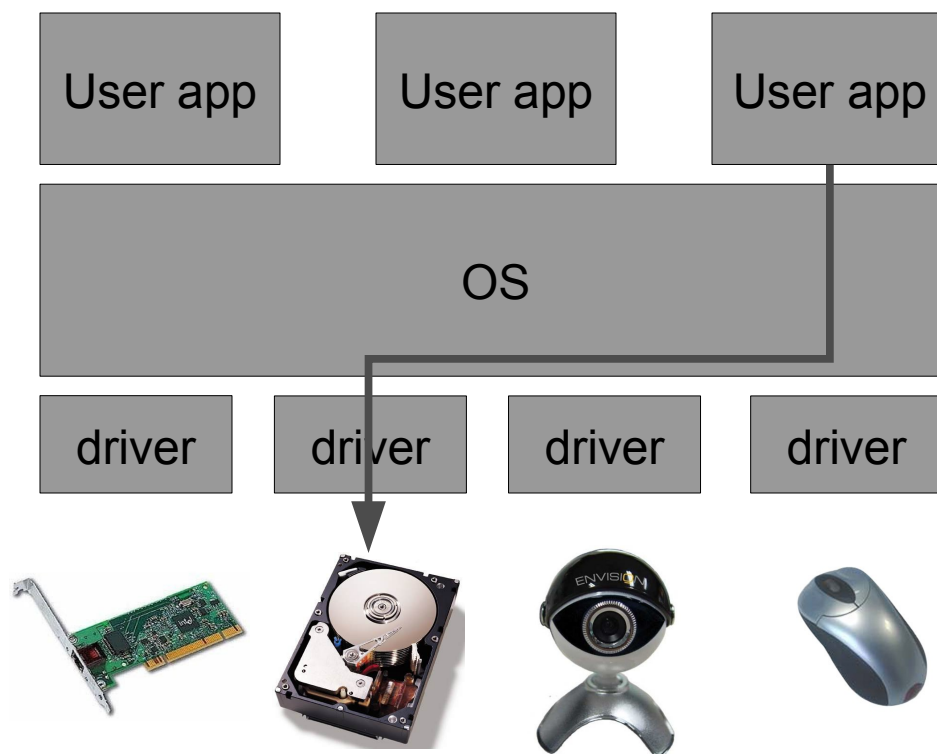
---

# Device Drivers

COMP9242  
2011/S2 Week 5



- Part 1: Introduction to device drivers
- Part 2: Overview of research on device driver reliability
- Part 3: Device drivers research at ERTOS

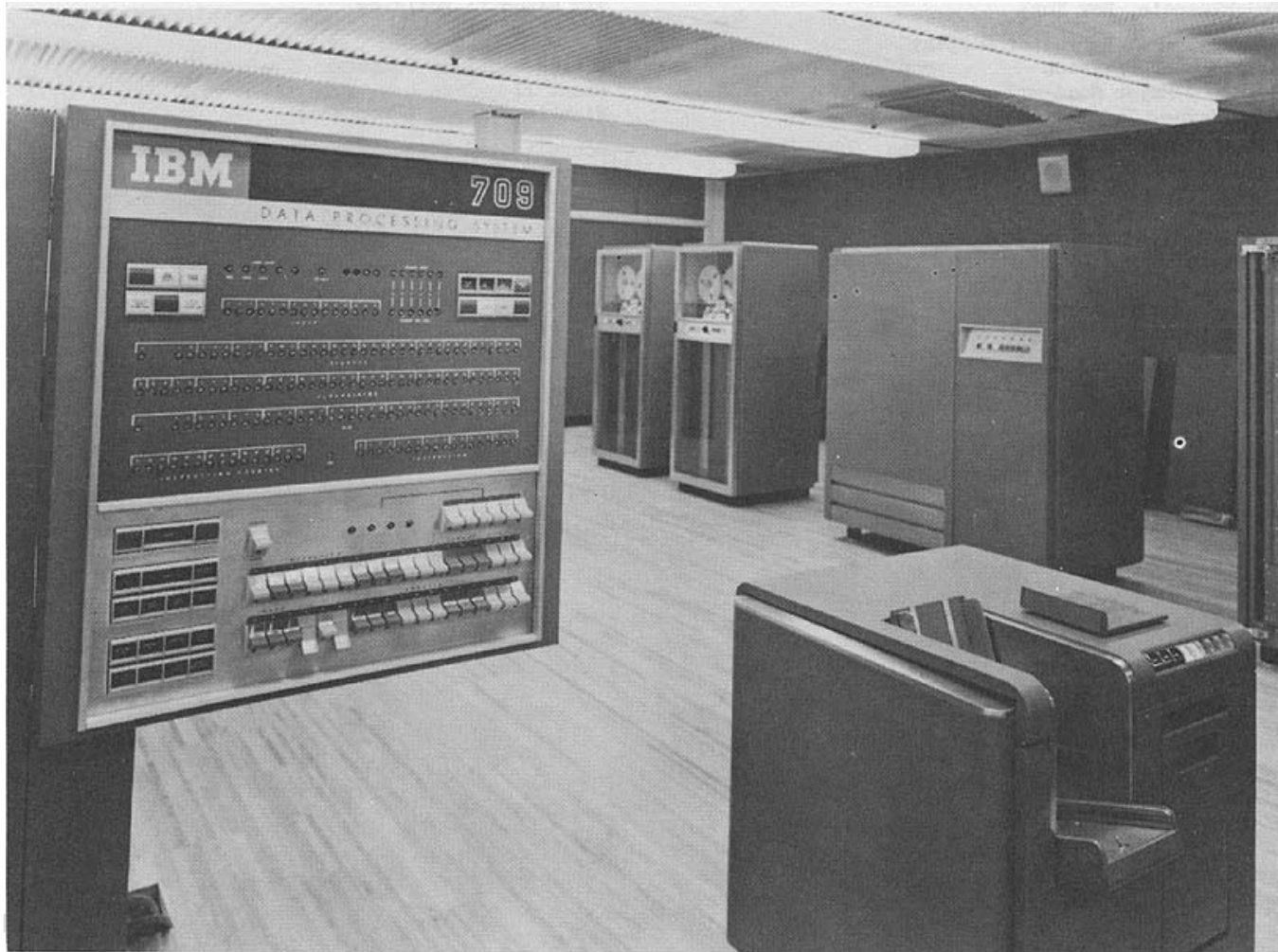


- 70% of OS code is in device drivers
  - 3,448,000 out of 4,997,000 loc in Linux 2.6.27
- A typical Linux laptop runs ~240,000 lines of kernel code, including ~72,000 loc in 36 different device drivers
- Drivers contain 3—7 times more bugs per loc than the rest of the kernel
- 70% of OS failures are caused by driver bugs

# Part 1: Introduction to device drivers

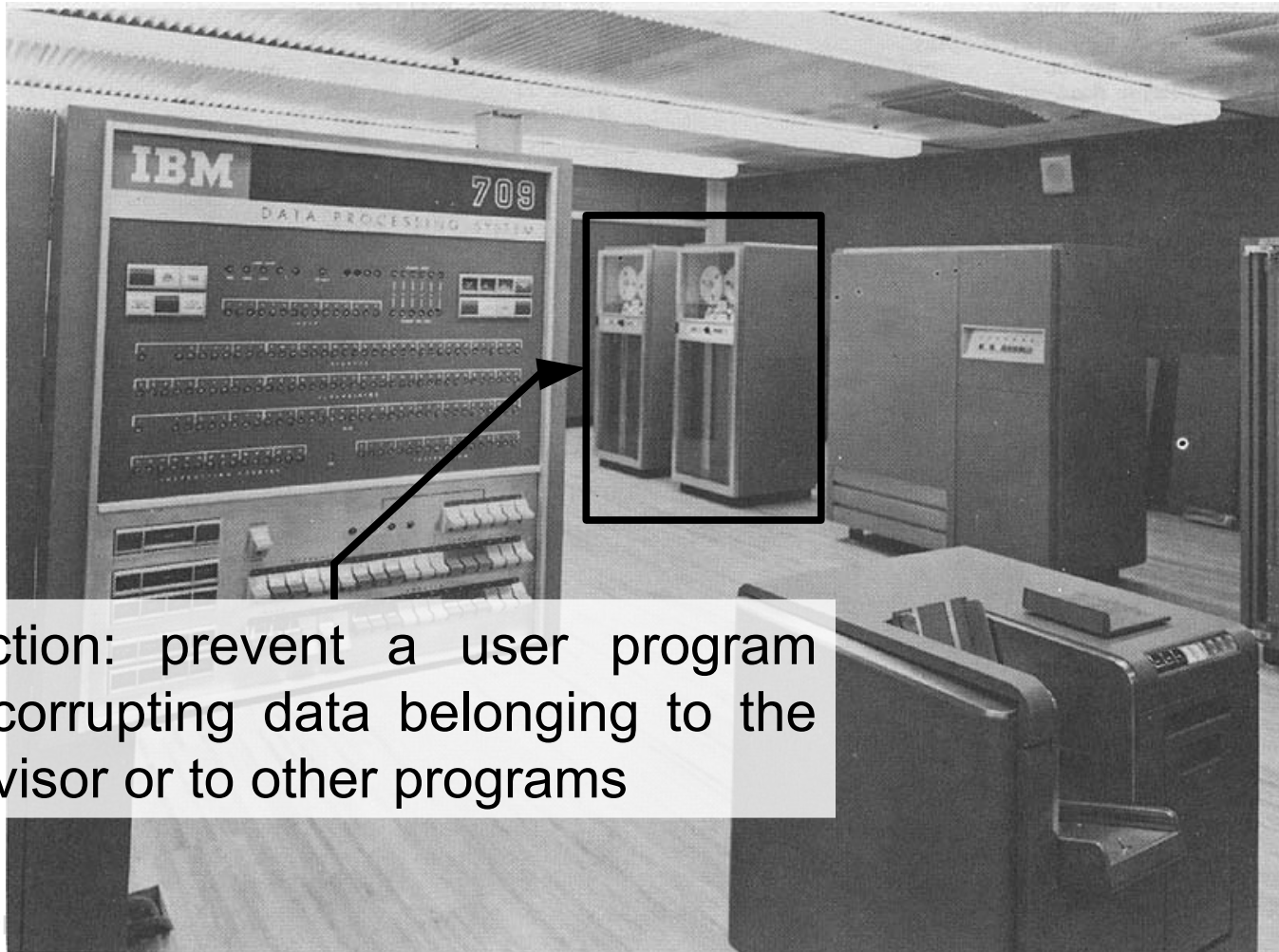
# OS archeology

The first (?) device drivers: I/O libraries for the IBM 709 batch processing system [1958]



# OS archeology

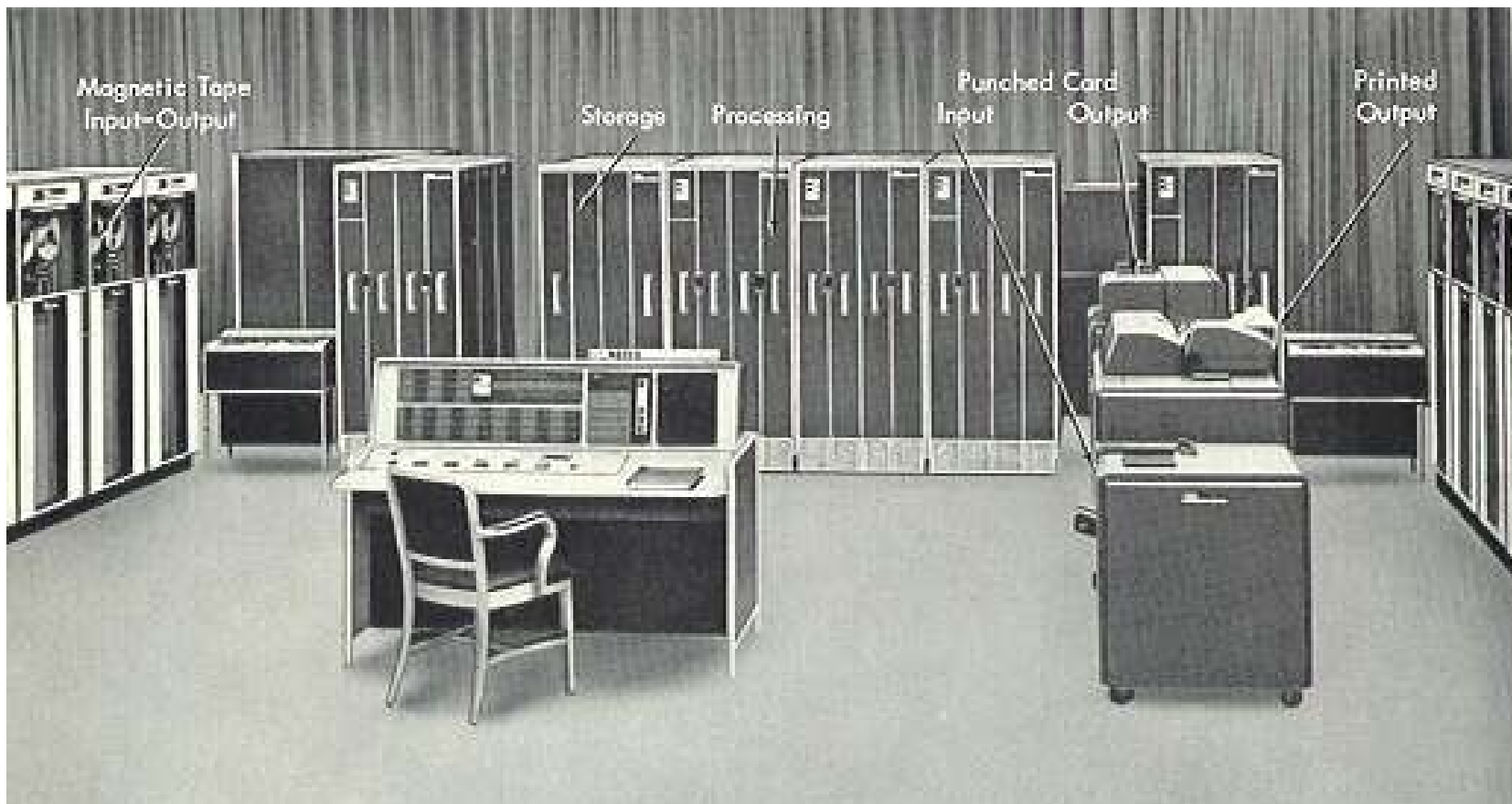
The first (?) device drivers: I/O libraries for the IBM 709 batch processing system [1958]



Protection: prevent a user program from corrupting data belonging to the supervisor or to other programs

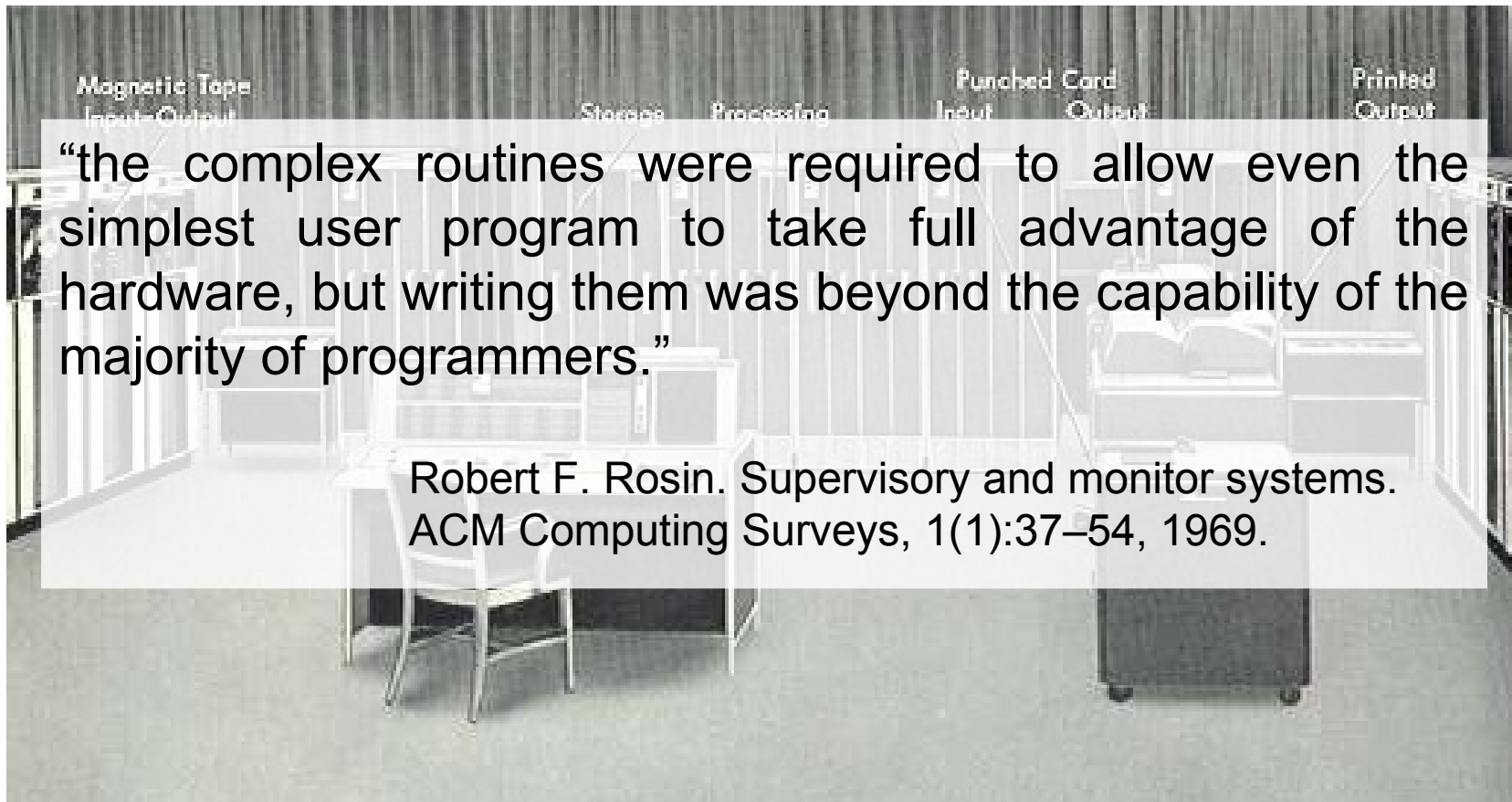
# OS archeology

IBM 7090 [1959] introduced I/O channels, which allowed I/O and computation to overlap



# OS archeology

IBM 7090 [1959] introduced I/O channels, which allowed I/O and computation to overlap



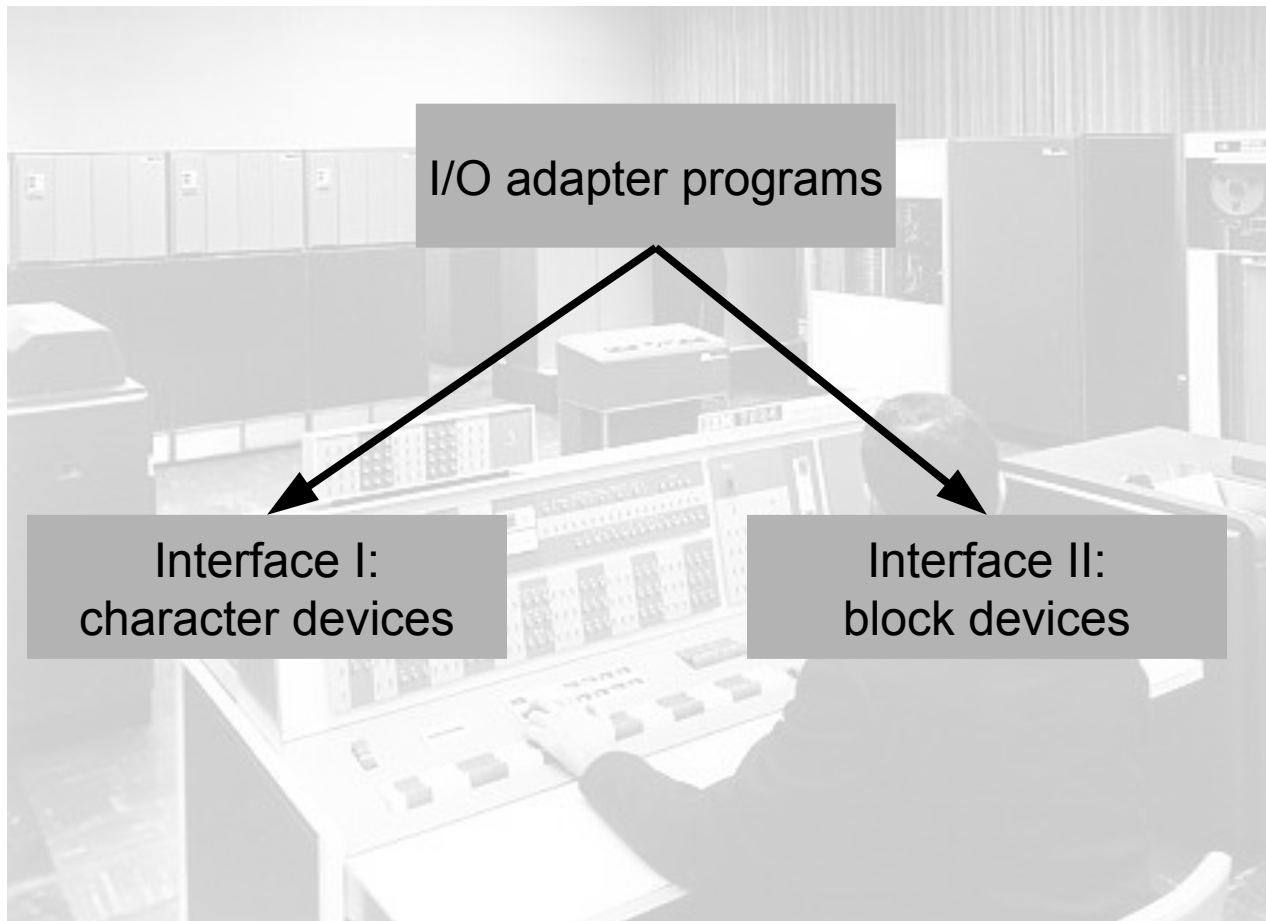
# OS archeology

IBM 7094 [1962] supported a wide range of peripherals: tapes, disks, teletypes, flexowriters, etc.



# OS archeology

IBM 7094 [1962] supported a wide range of peripherals: tapes, disks, teletypes, flexowriters, etc.



# OS archeology

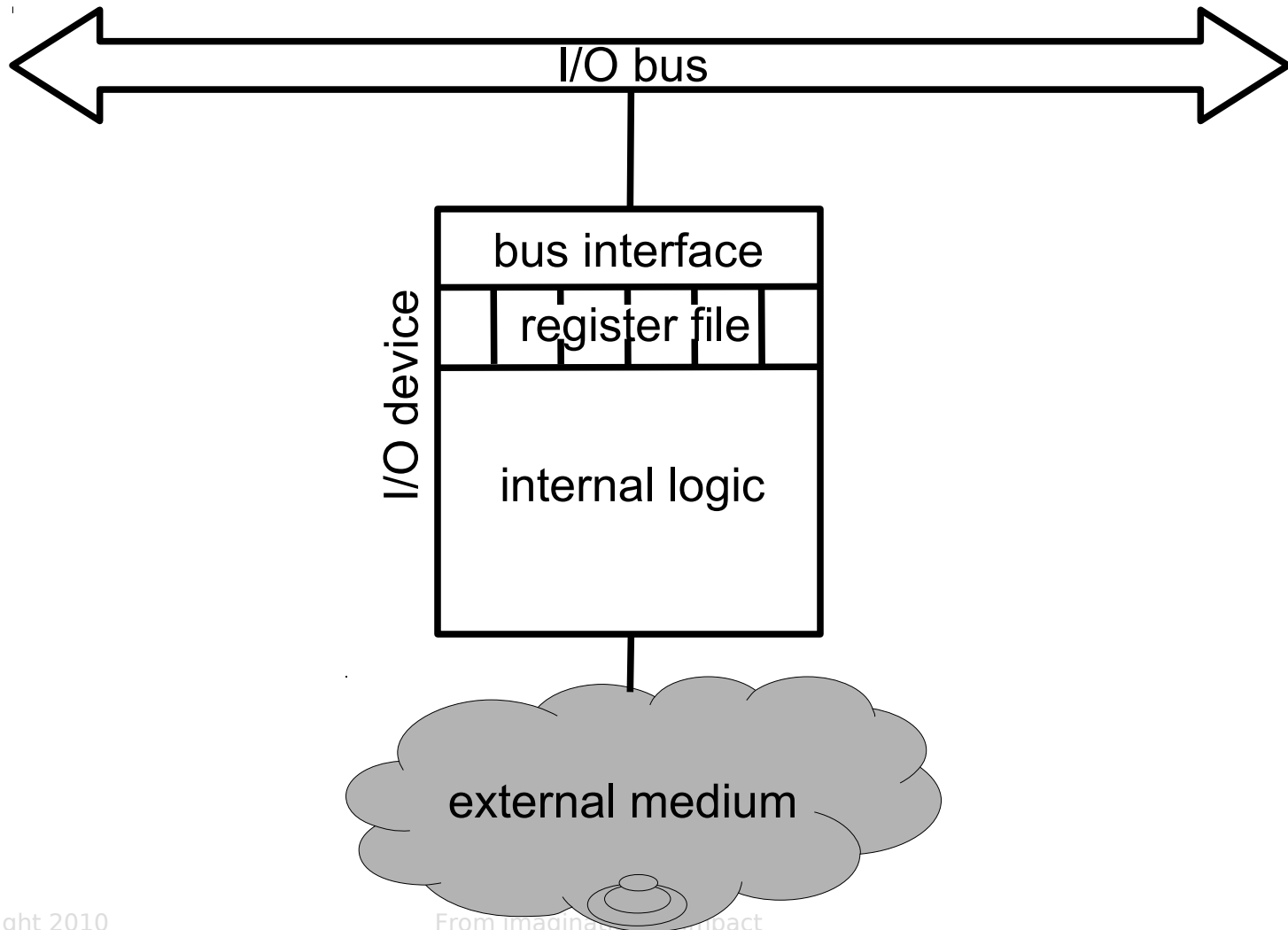
---

GE-635 [1963] introduced the master CPU mode. Only the hypervisor running in the master mode could execute I/O instructions

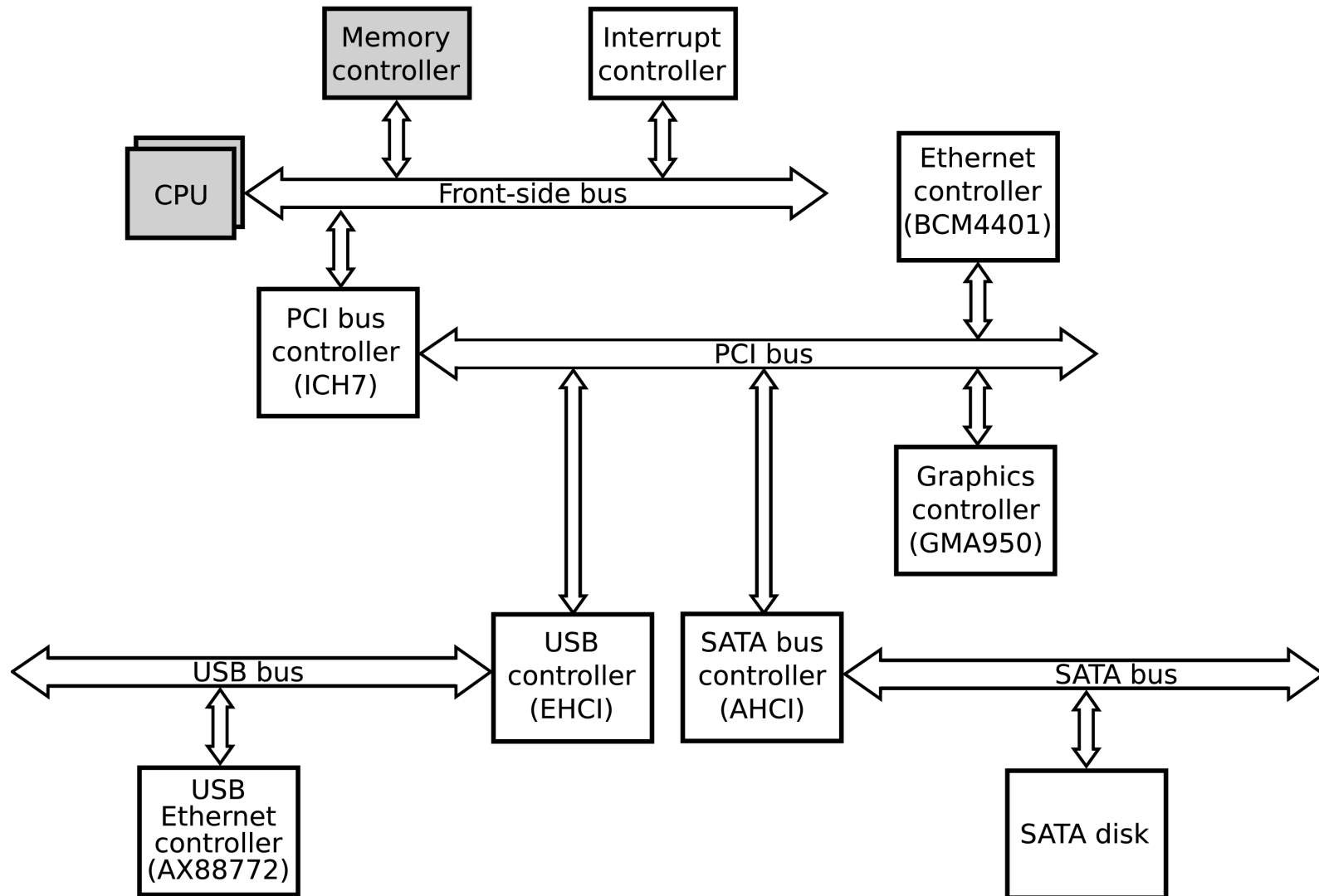


- Encapsulation
  - Hides low-level device protocol details from the client
- Unification
  - Makes similar devices look the same
- Protection (in cooperation with the OS)
  - Only authorised applications can use the device
- Multiplexing (in cooperation with the OS)
  - Multiple applications can use the device concurrently

# I/O device: a high-level view

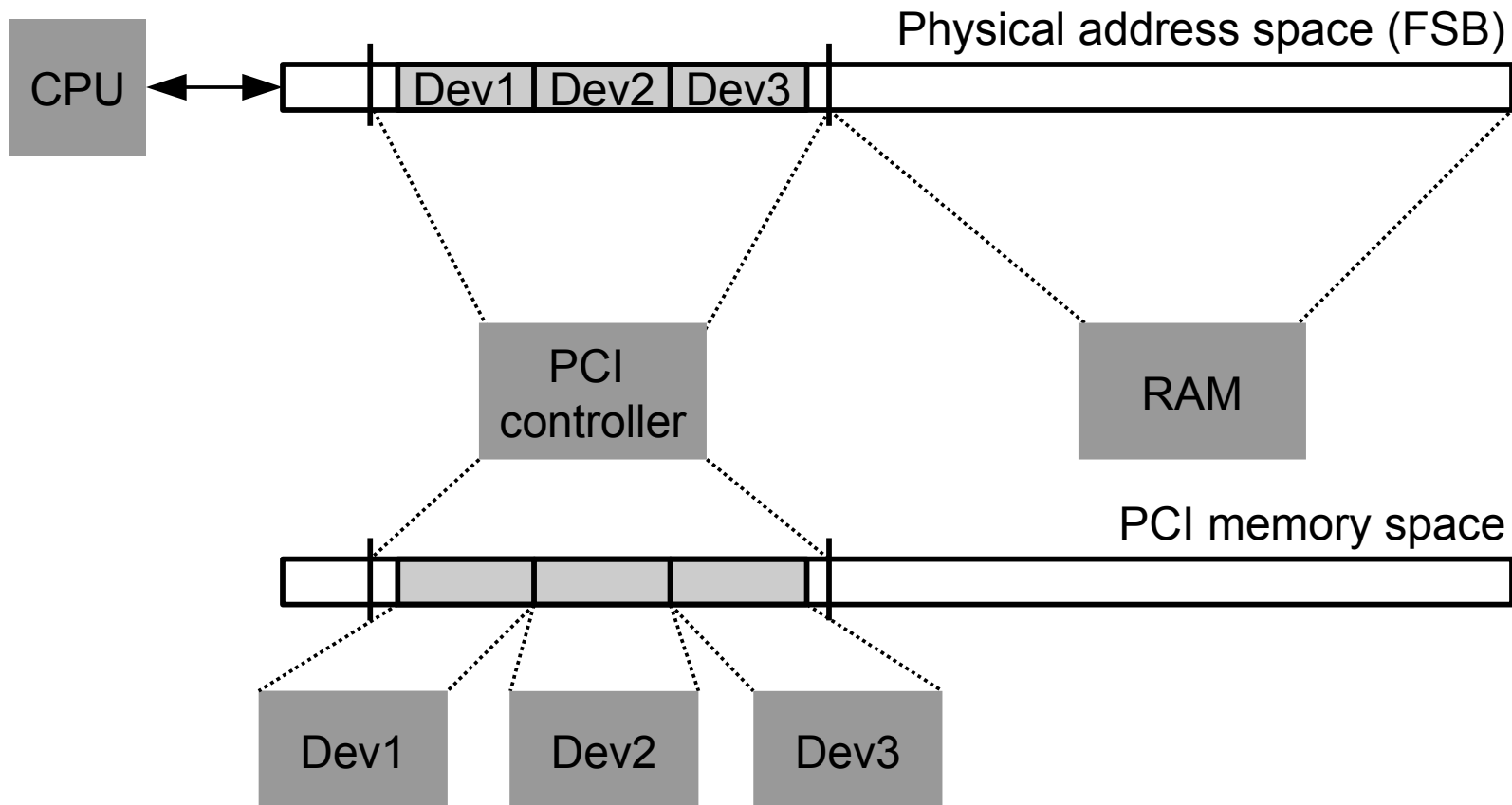


# I/O devices in a typical desktop system

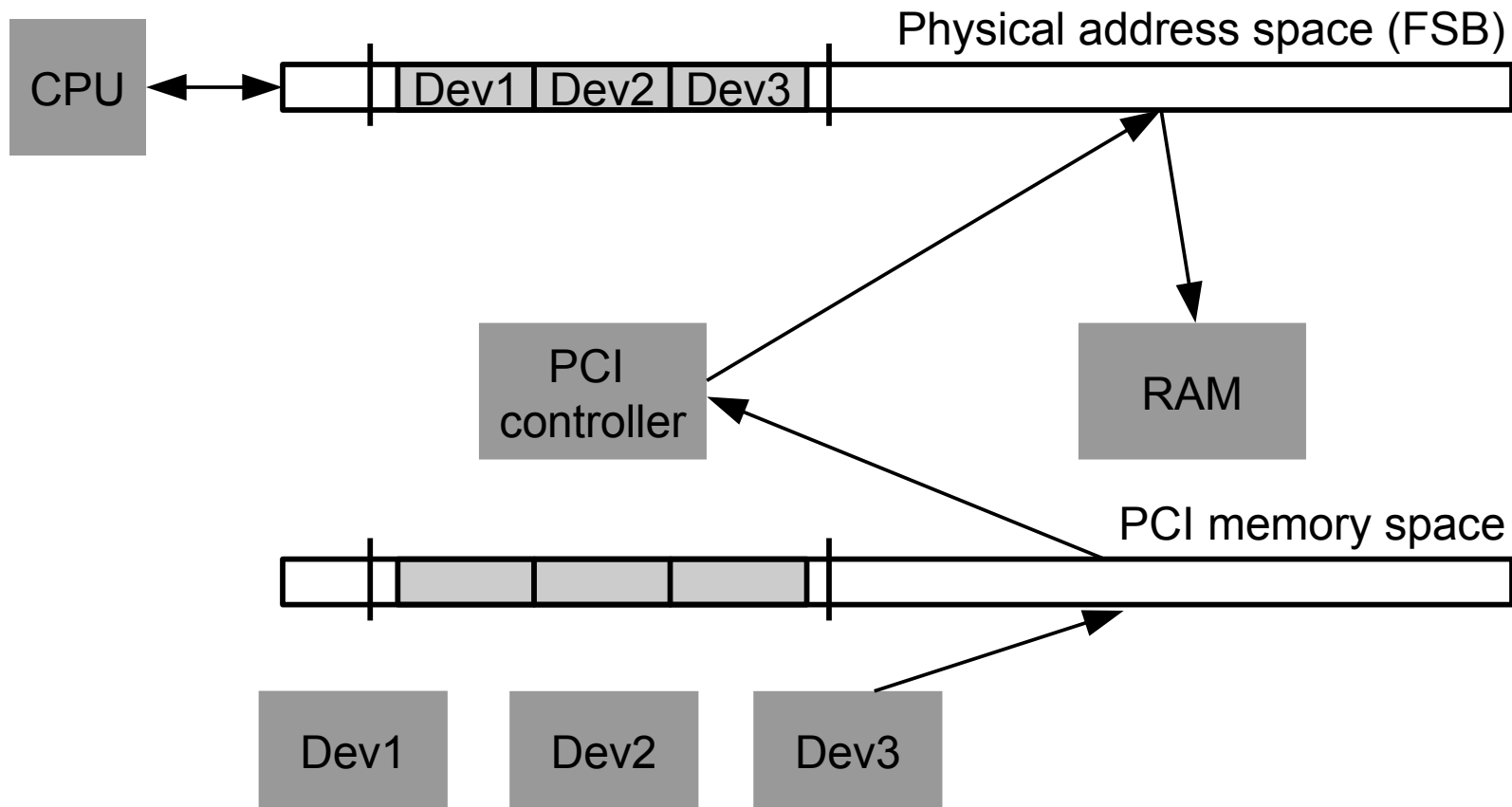


- PCI bus
  - Conventional PCI
    - Developed and standardised in early 90's
    - 32 or 64 bit shared parallel bus
    - Up to 66MHz (533MB/s)
  - PCI-X
    - Up to 133MHz (1066MB/s)
  - PCI Express
    - Consists of serial p2p links
    - Software-compatible with conventional PCI
    - Up to 16GB/s per device

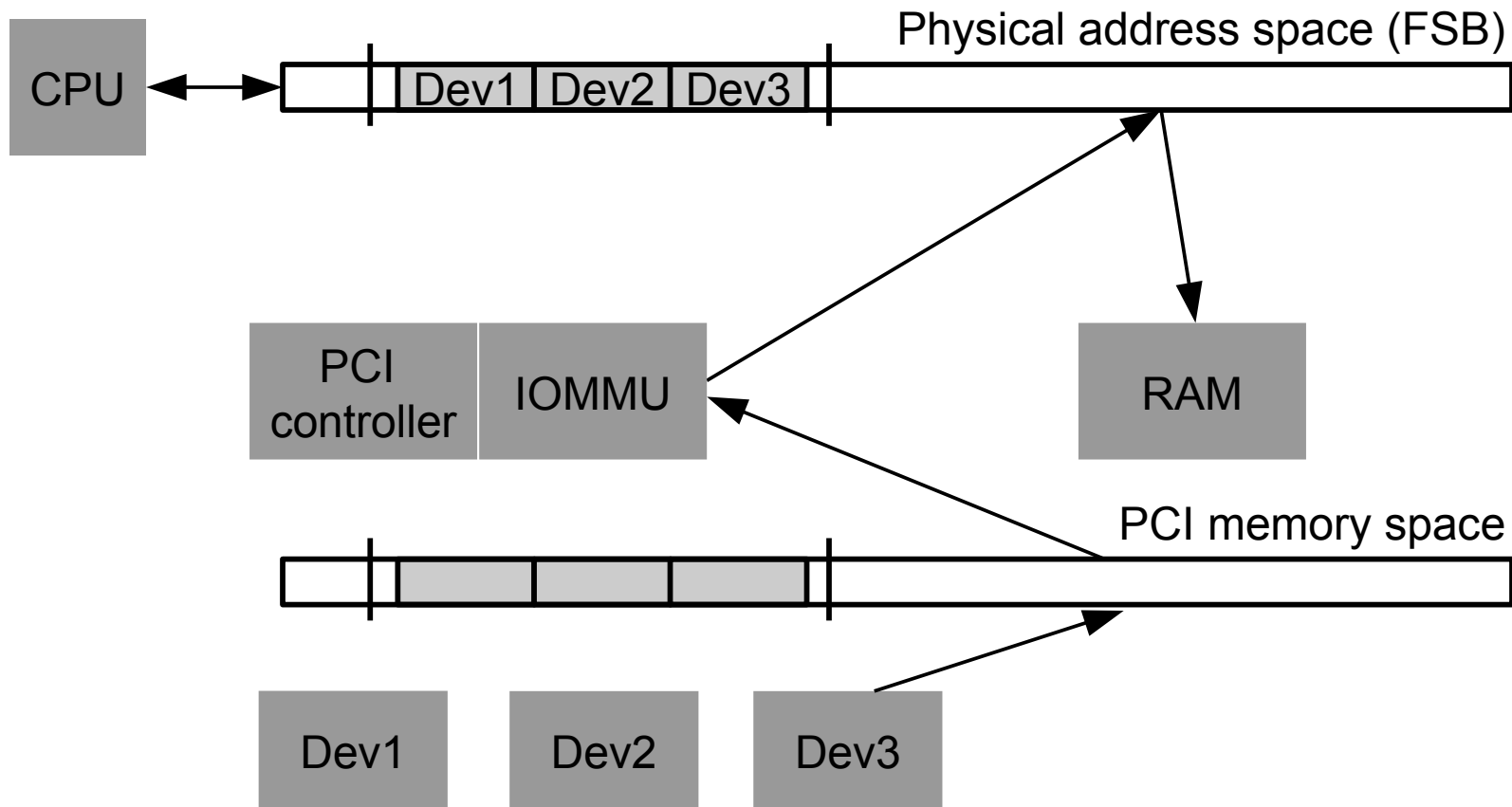
# PCI bus overview: memory space



# PCI bus overview: DMA



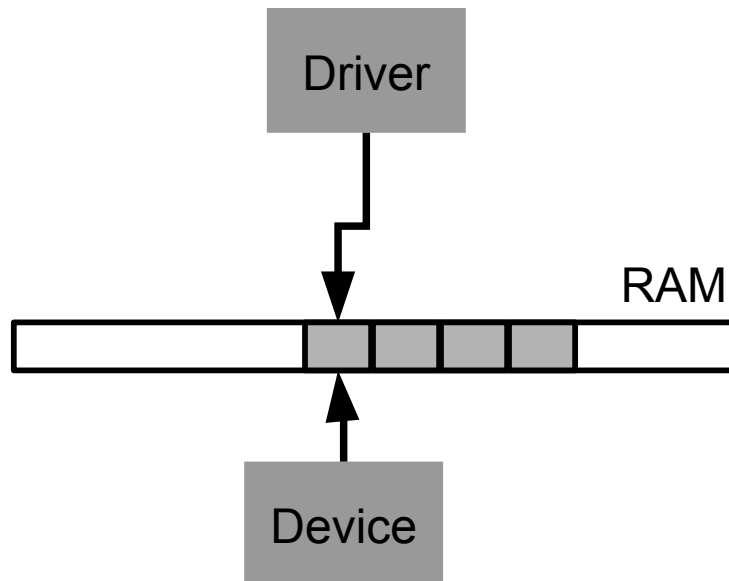
# PCI bus overview: DMA



# DMA descriptors

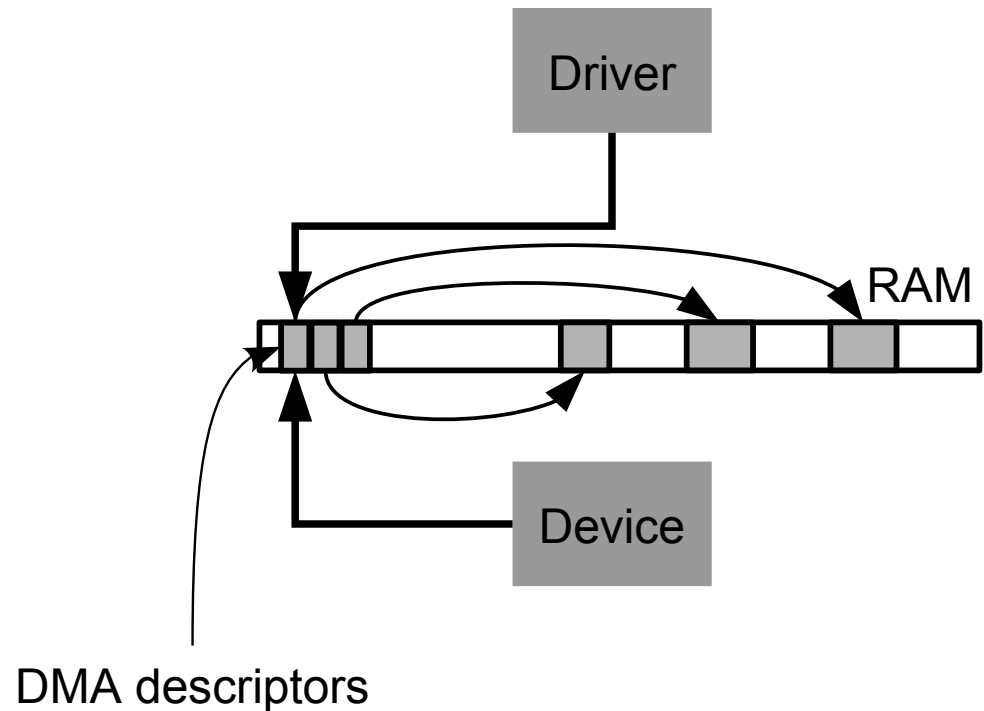
## Permanent DMA mappings

- Set up during driver initialisation
- Data must be copied to/from DMA buffers

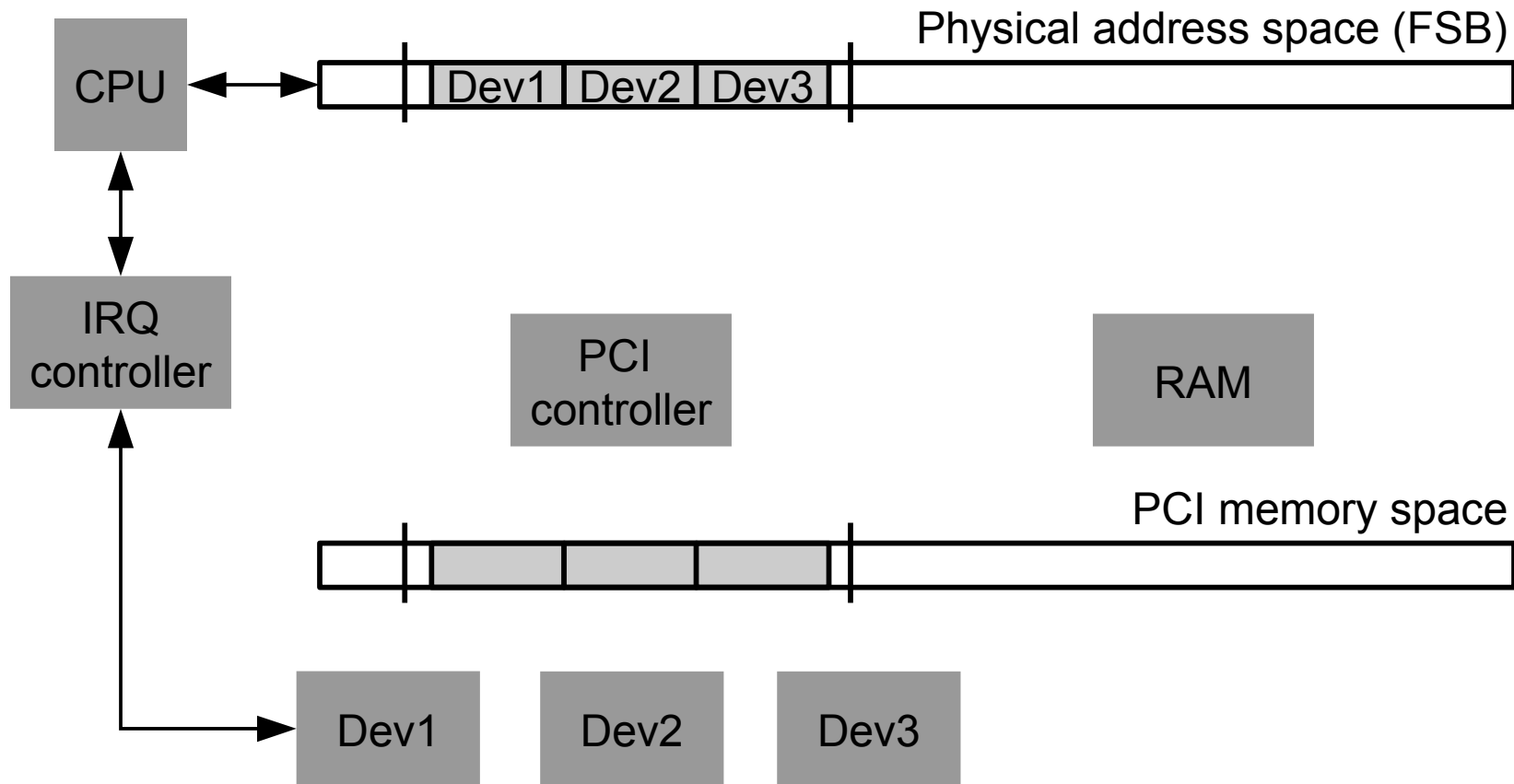


## Streaming mappings

- Created for each transfer
- Data is accessed in-place



# PCI bus overview: interrupts



# PCI bus overview: config space

- PCI configuration space
  - Used for device enumeration and configuration
  - Contains standardised device descriptors

31		16 15		0	
<b>Device ID</b>		<b>Vendor ID</b>		00h	
<b>Status</b>		<b>Command</b>		04h	
<b>Class Code</b>			<b>Revision ID</b>	08h	
<b>BIST</b>	<b>Header Type</b>	<b>Lat. Timer</b>	<b>Cache Line S.</b>	0Ch	
<b>Base Address Registers</b>				10h	
				14h	
				18h	
				1Ch	
				20h	
				24h	
<b>Cardbus CIS Pointer</b>				28h	
<b>Subsystem ID</b>		<b>Subsystem Vendor ID</b>		2Ch	
<b>Expansion ROM Base Address</b>				30h	
<b>Reserved</b>			<b>Cap. Pointer</b>	34h	
<b>Reserved</b>				38h	
<b>Max Lat.</b>	<b>Min Gnt.</b>	<b>Interrupt Pin</b>	<b>Interrupt Line</b>	3Ch	

# PCI bus overview: I/O space

---



- I/O space
  - obsolete

# Writing a driver for a PCI device

---



- Registration
  - Tell the OS which PCI device ID's the driver supports
- Instantiation
  - Done by the OS when it finds a driver with a matching ID
- Initialisation
  - Allocate PCI resources: memory regions, IRQ's
  - Enable bus mastering
  - Permanent DMA mappings: disable caching

# Writing a driver for a PCI device

---



- Interrupt handler
  - Return *ASAP* to re-enable interrupts; perform heavy-weight processing in a separate thread
- DMA
  - Streaming mappings: may require bounce buffers
- Power management
  - Prepare the device for a transition into a low-power state
  - Restore device configuration during wake-up

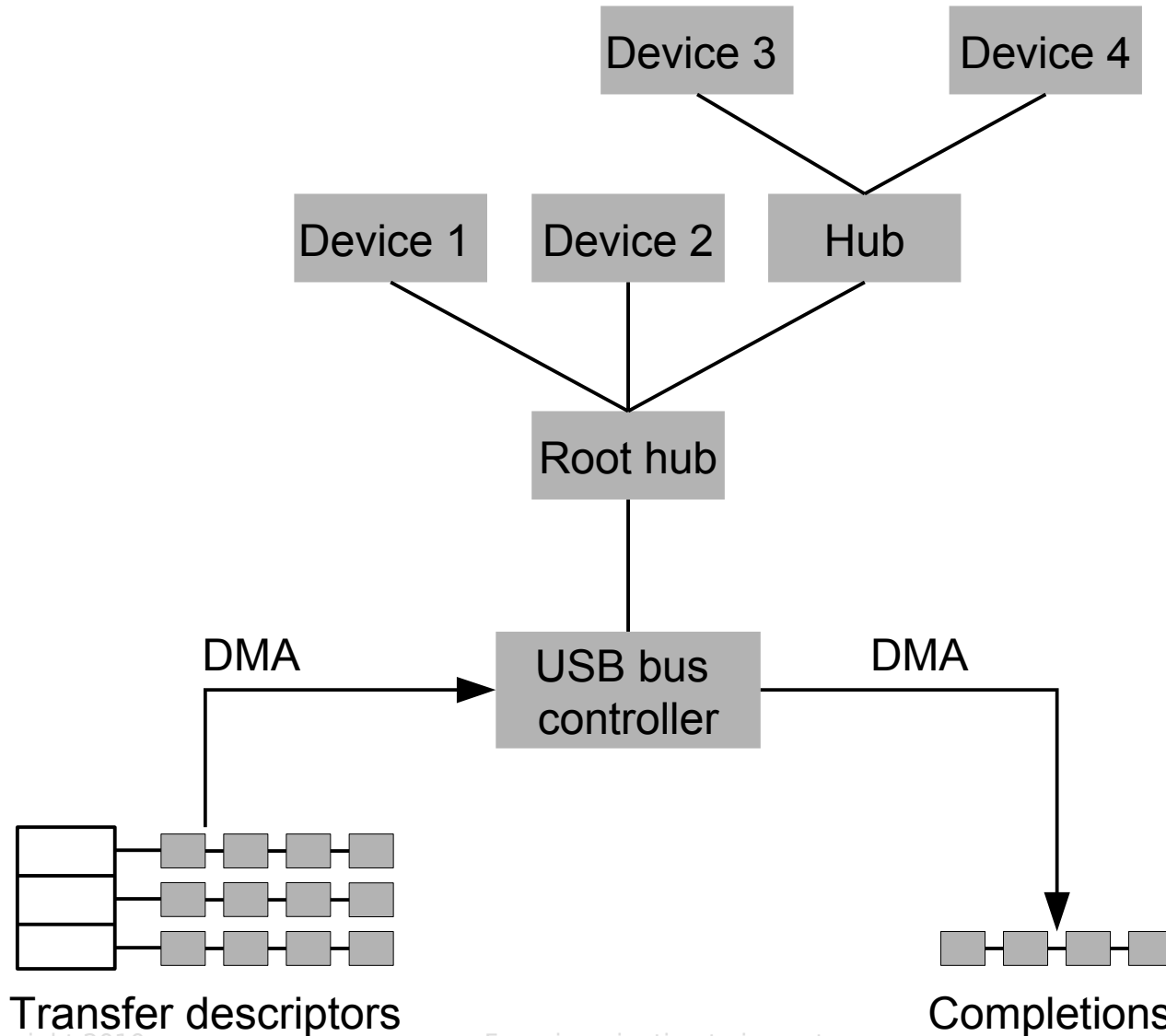
# USB bus overview

---

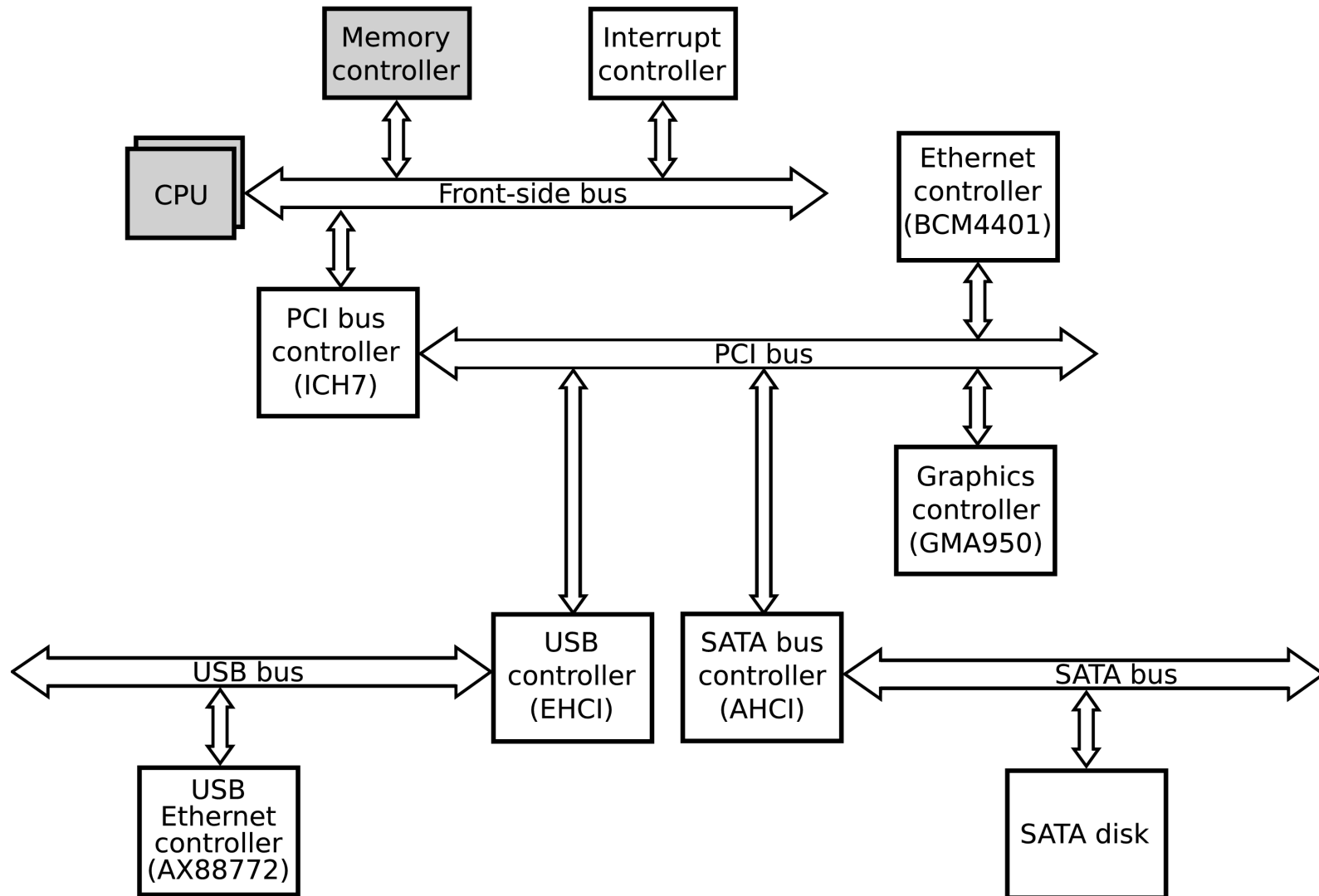


- USB bus
  - Host-centric
  - Distributed-system-style architecture (packet-based transfers)
  - Hot plug
  - Power management
    - Bus-powered and self-powered devices
  - USB 1.x
    - Up to 12Mb/s
  - USB 2.0
    - Up to 480Mb/s
  - USB 3.0
    - Up to 4.8Gb/s

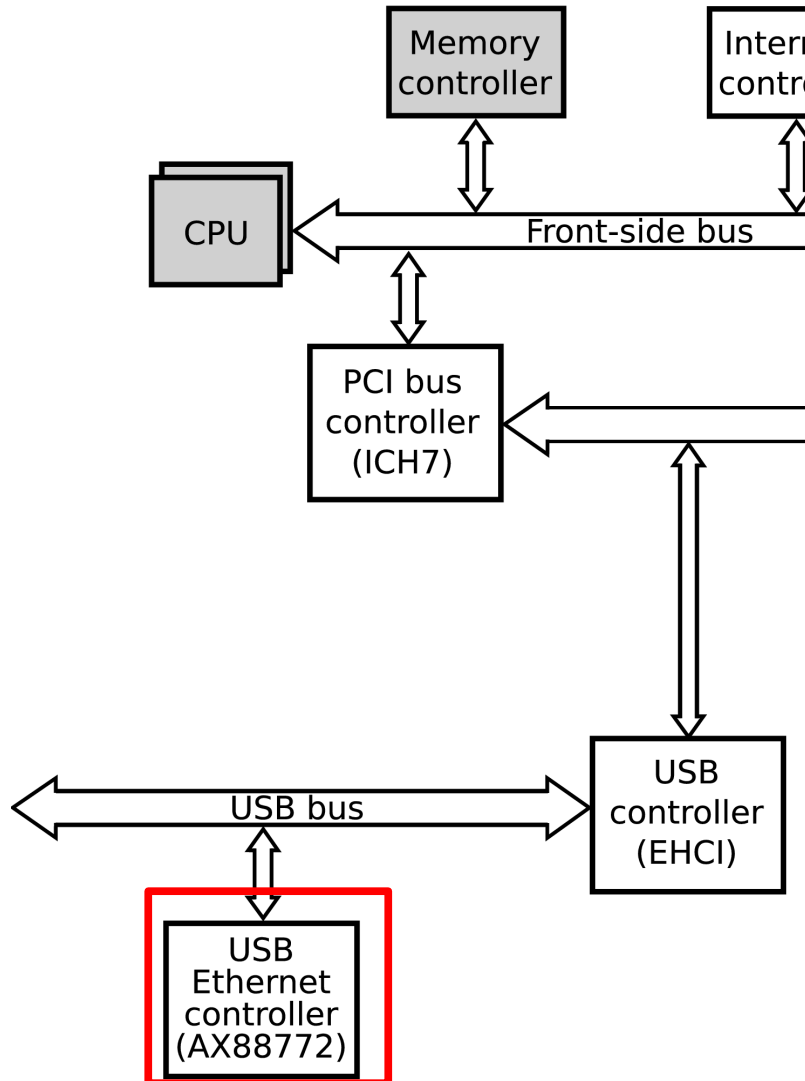
# USB bus overview



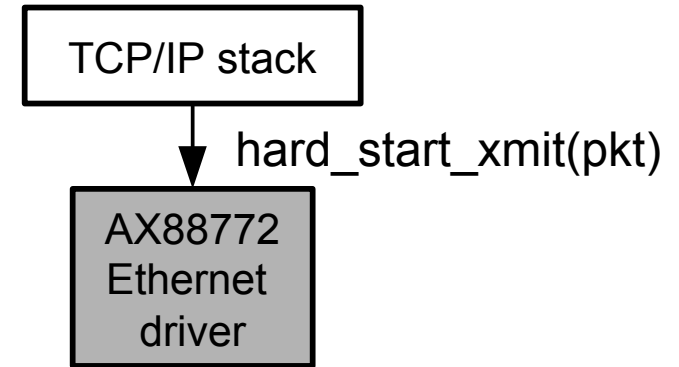
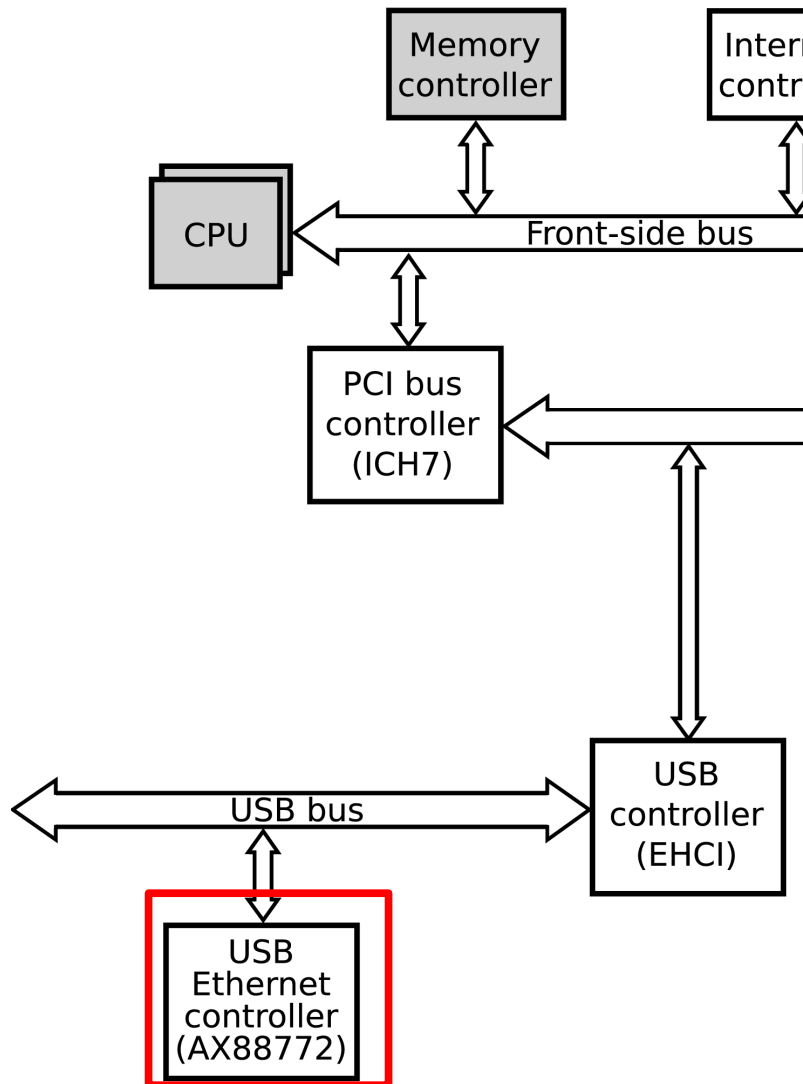
# I/O devices in a typical desktop system



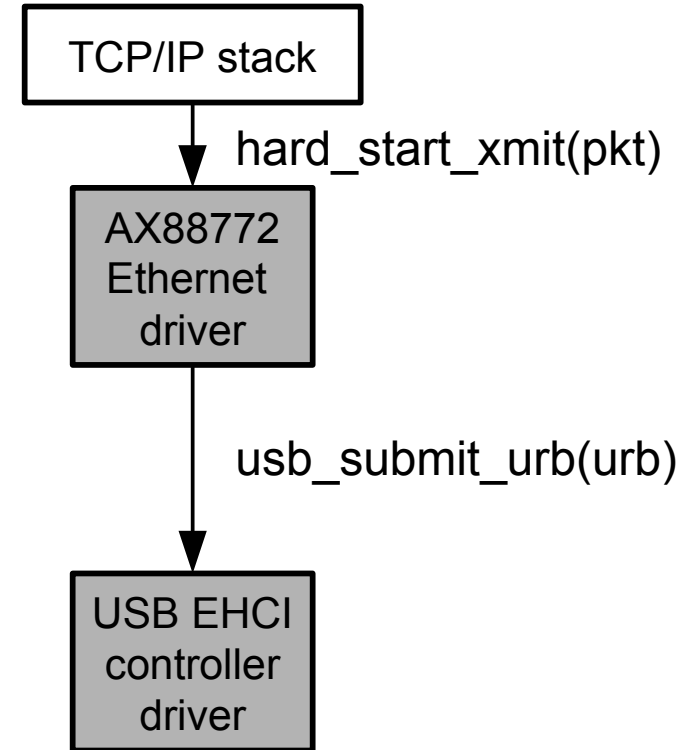
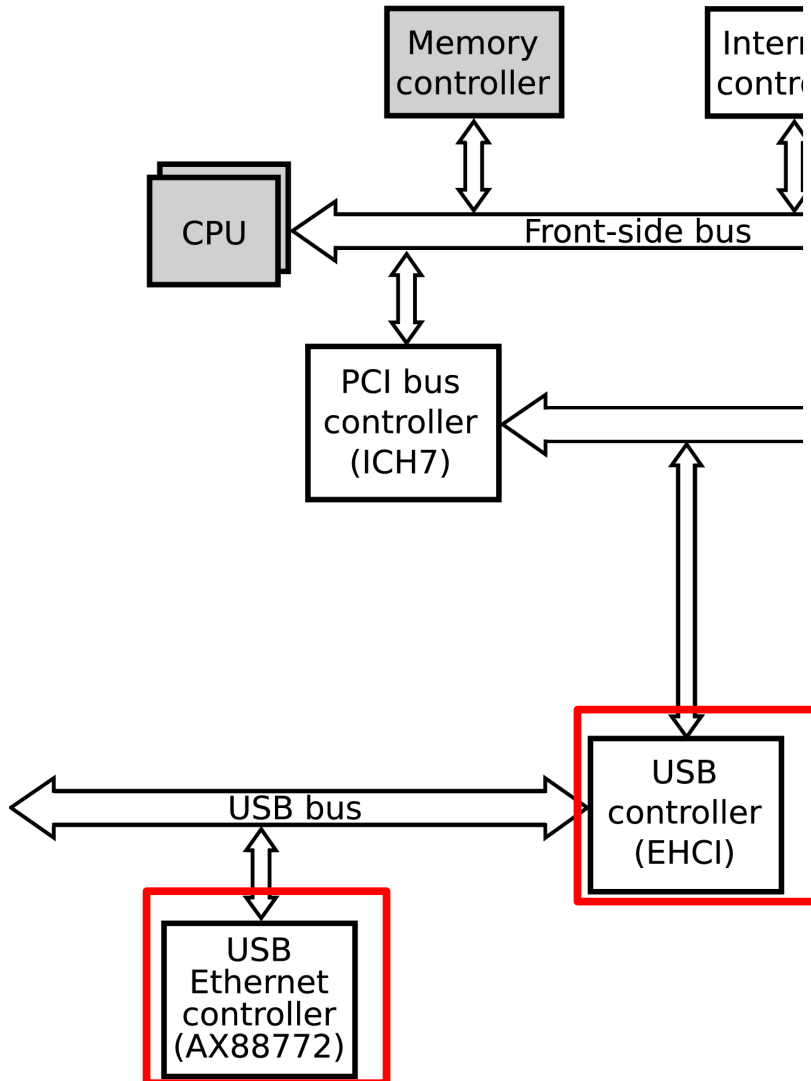
# Driver stacking



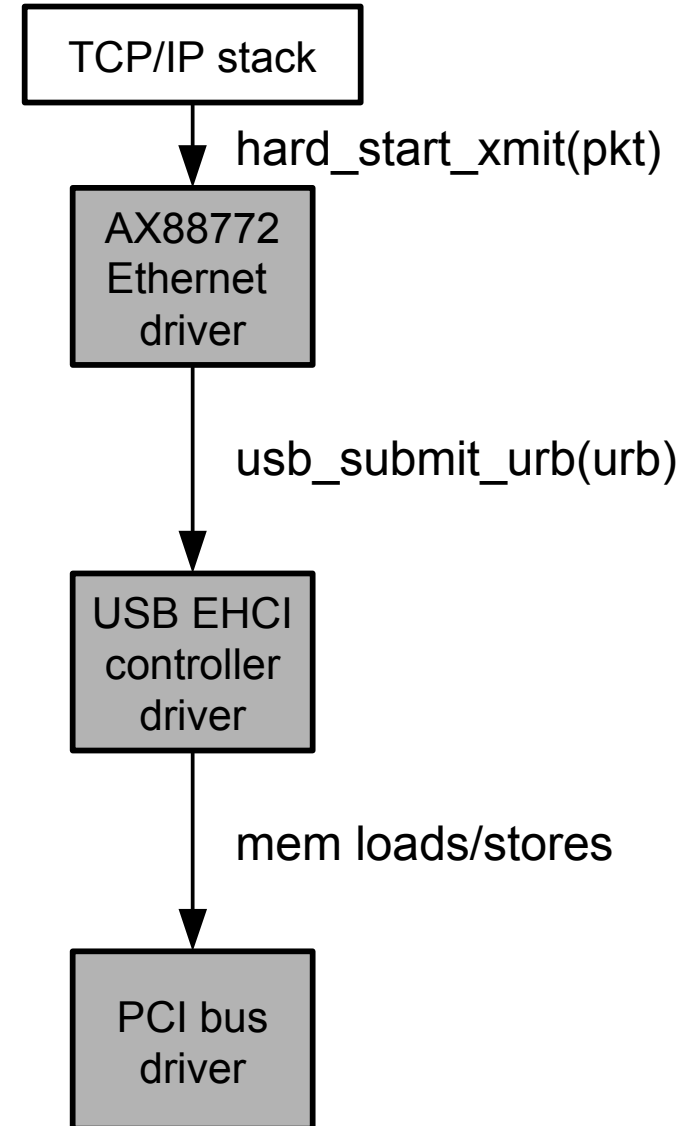
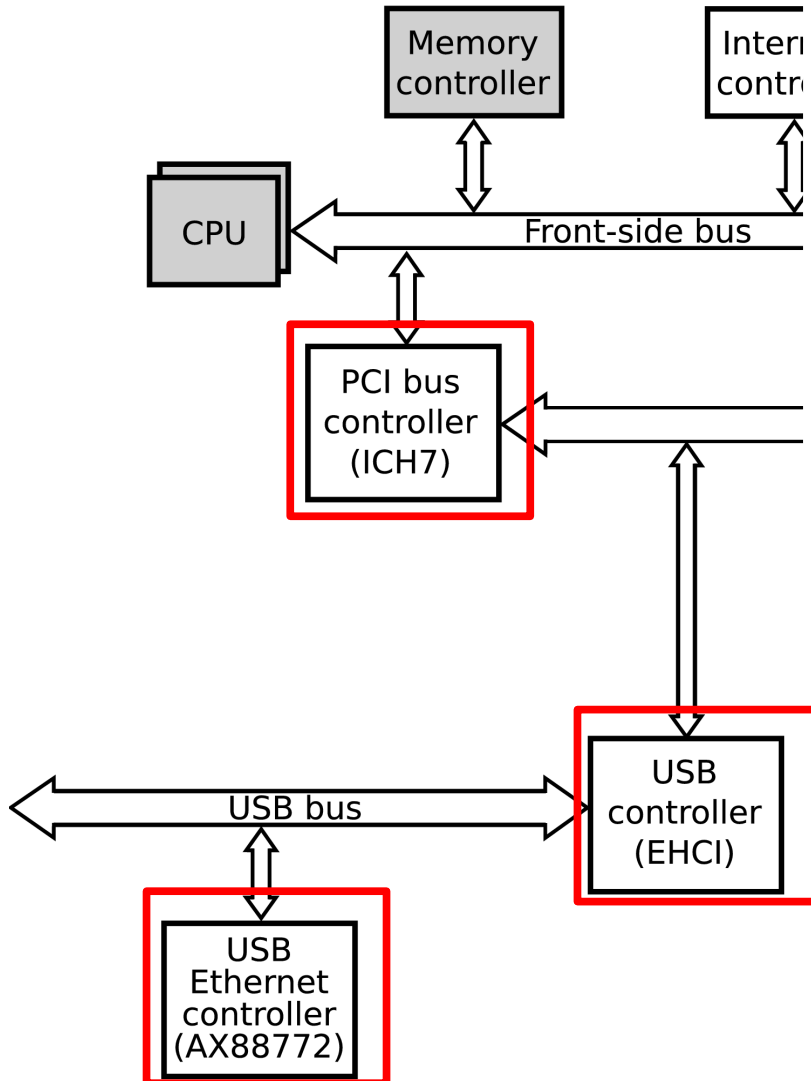
# Driver stacking



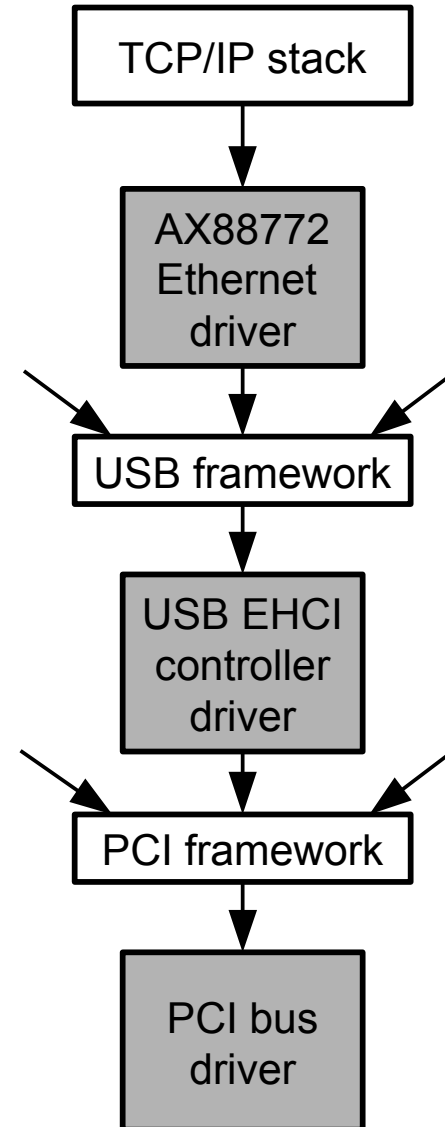
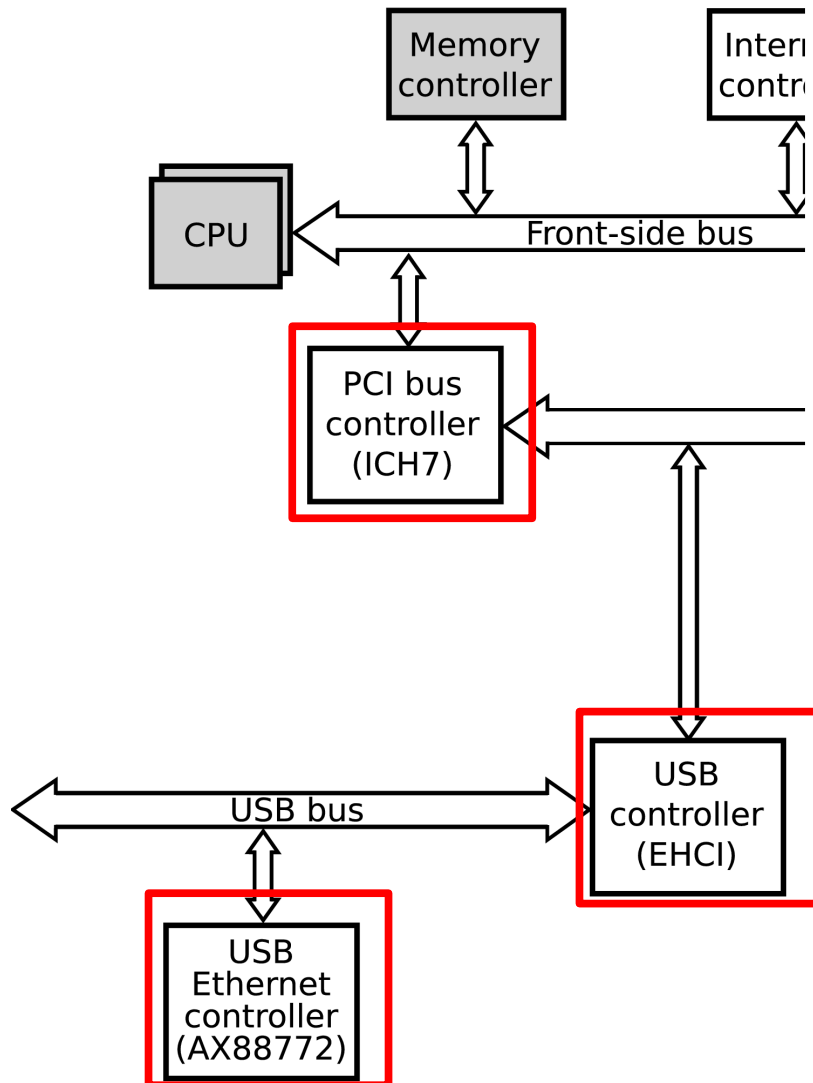
# Driver stacking



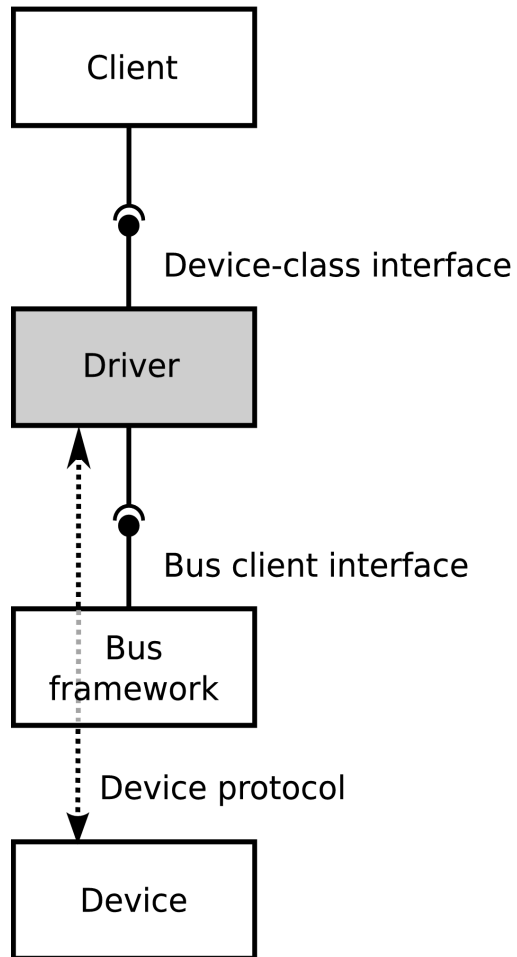
# Driver stacking



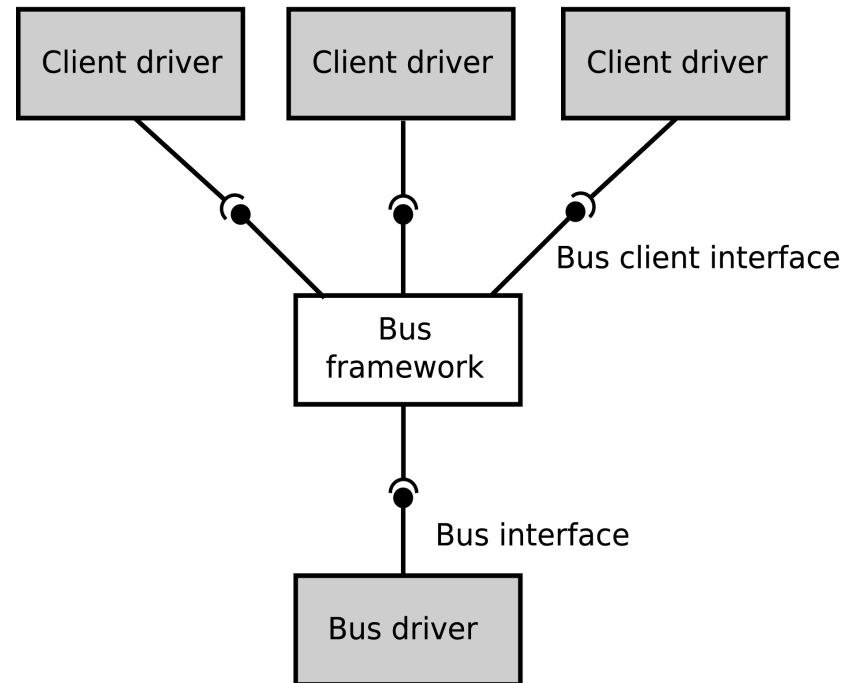
# Driver stacking



# Driver framework design patterns

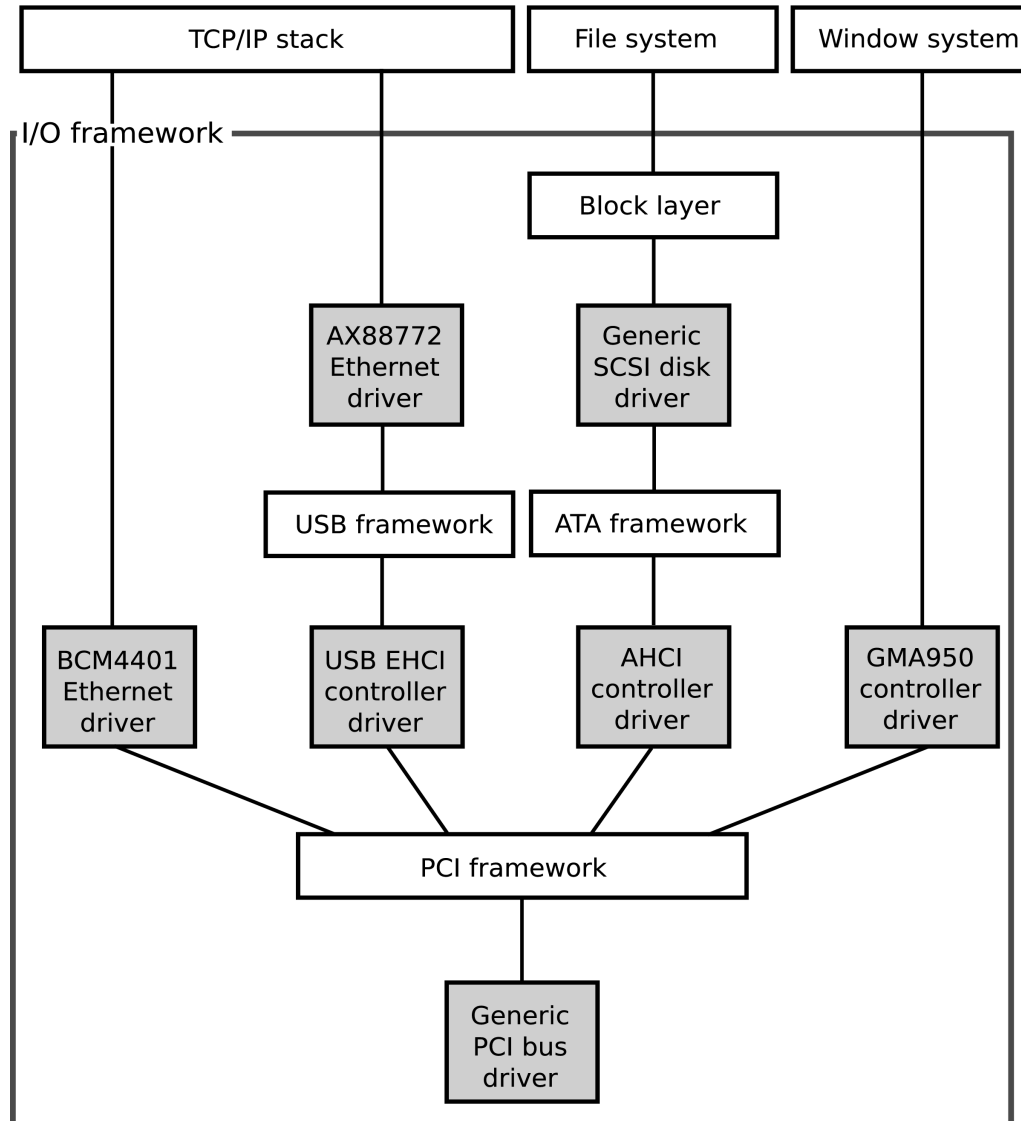


The driver pattern



The bus pattern

# Driver framework software architecture



Questions?

# Part 2: Overview of research on device driver reliability

- 70% of OS code is in device drivers
  - 3,448,000 out of 4,997,000 loc in Linux 2.6.27
- A typical Linux laptop runs ~240,000 lines of kernel code, including ~72,000 loc in 36 different device drivers
- Drivers contain 3—7 times more bugs per loc than the rest of the kernel
- 70% of OS failures are caused by driver bugs

# Understanding driver bugs

---



- Driver failures

# Understanding driver bugs

---

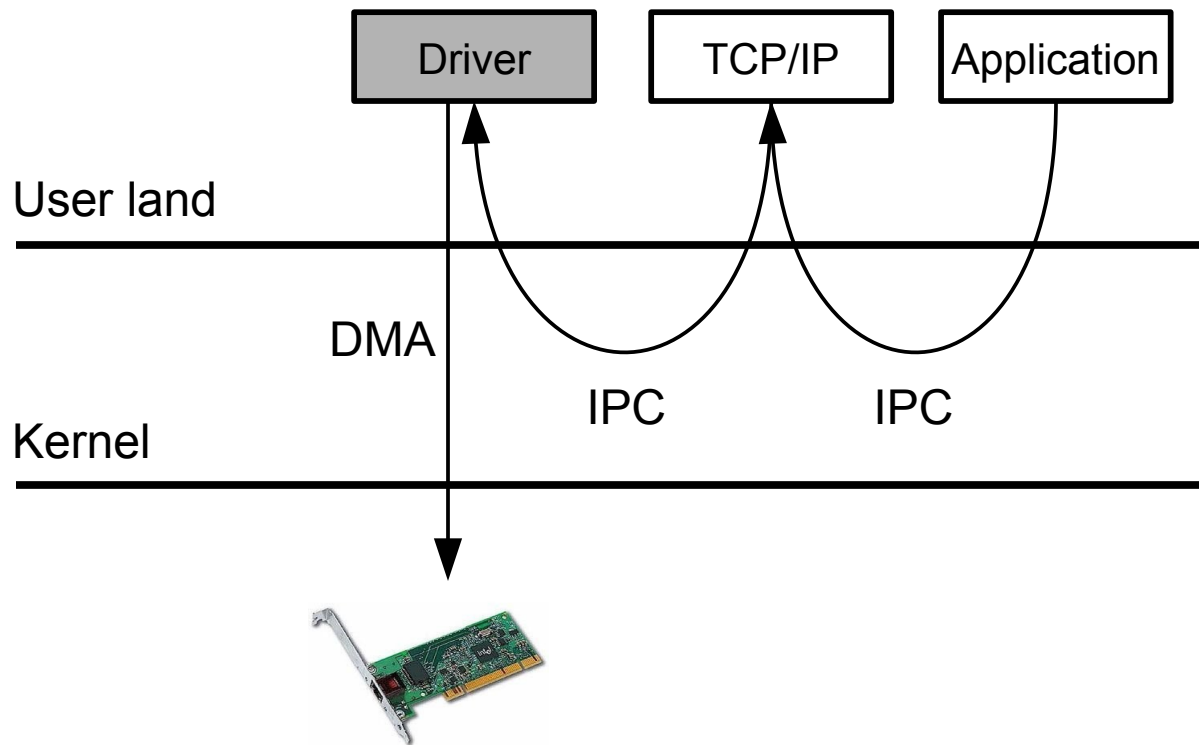


- Driver failures
  - Memory access violations
  - OS protocol violations
    - Ordering violations
    - Data format violations
    - Excessive use of resources
    - Temporal failure
  - Device protocol violations
    - Incorrect use of the device state machine
    - Runaway DMA
    - Interrupt storms
  - Concurrency bugs
    - Race conditions
    - Deadlocks

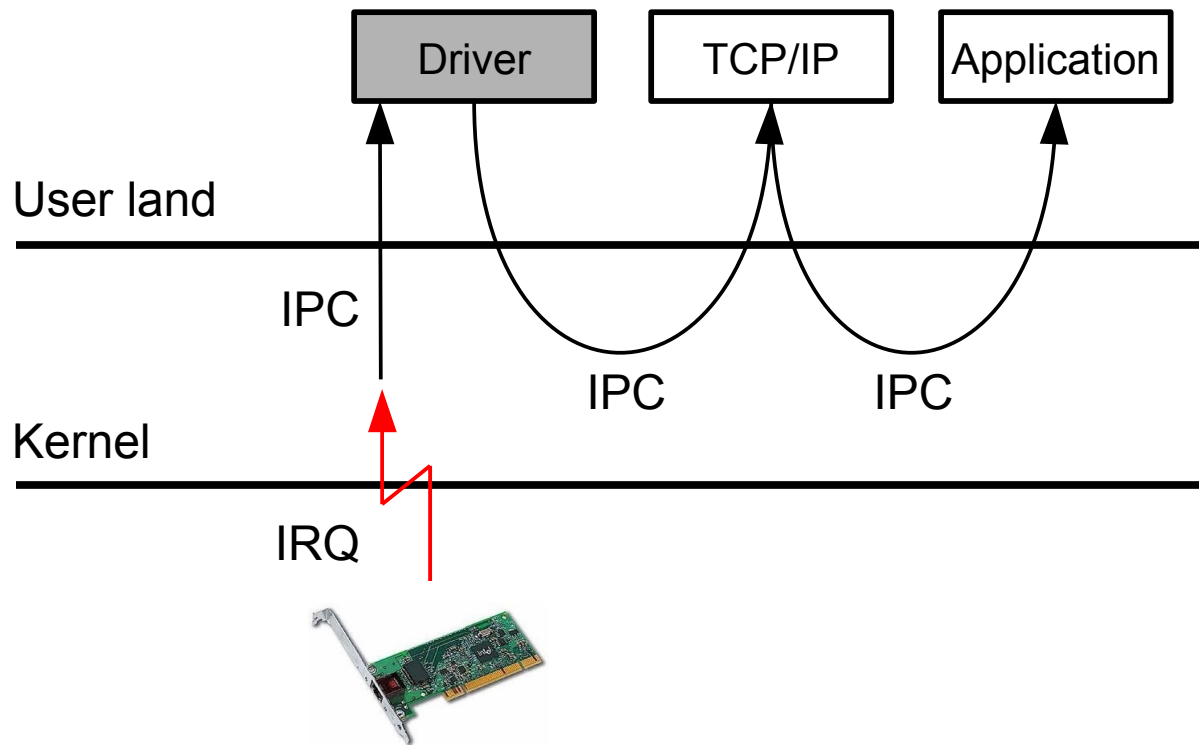
- User-level drivers

- Each driver is encapsulated inside a separate hardware protection domain
- Communication between the driver and its client is based on IPC
- Device memory is mapped into the virtual address space of the driver
- Interrupts are delivered to the driver via IPC's

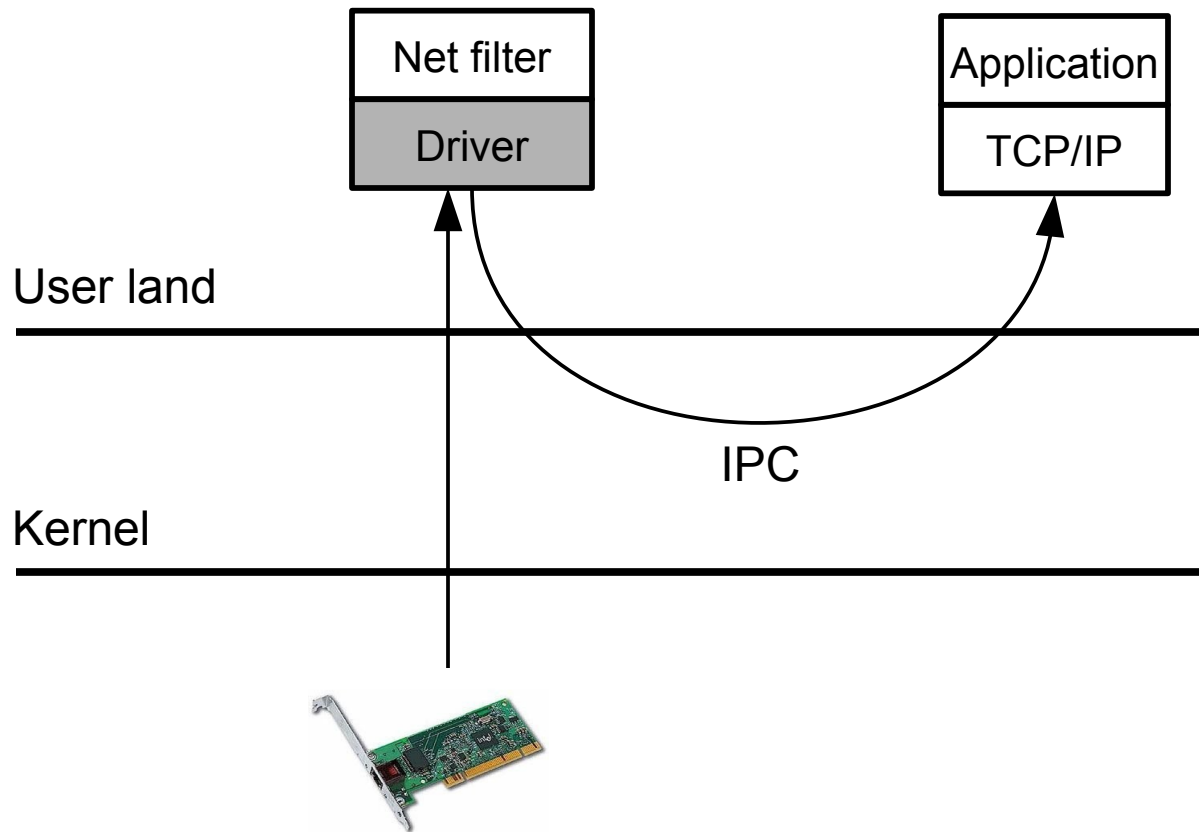
# User-level drivers in $\mu$ -kernel OSs



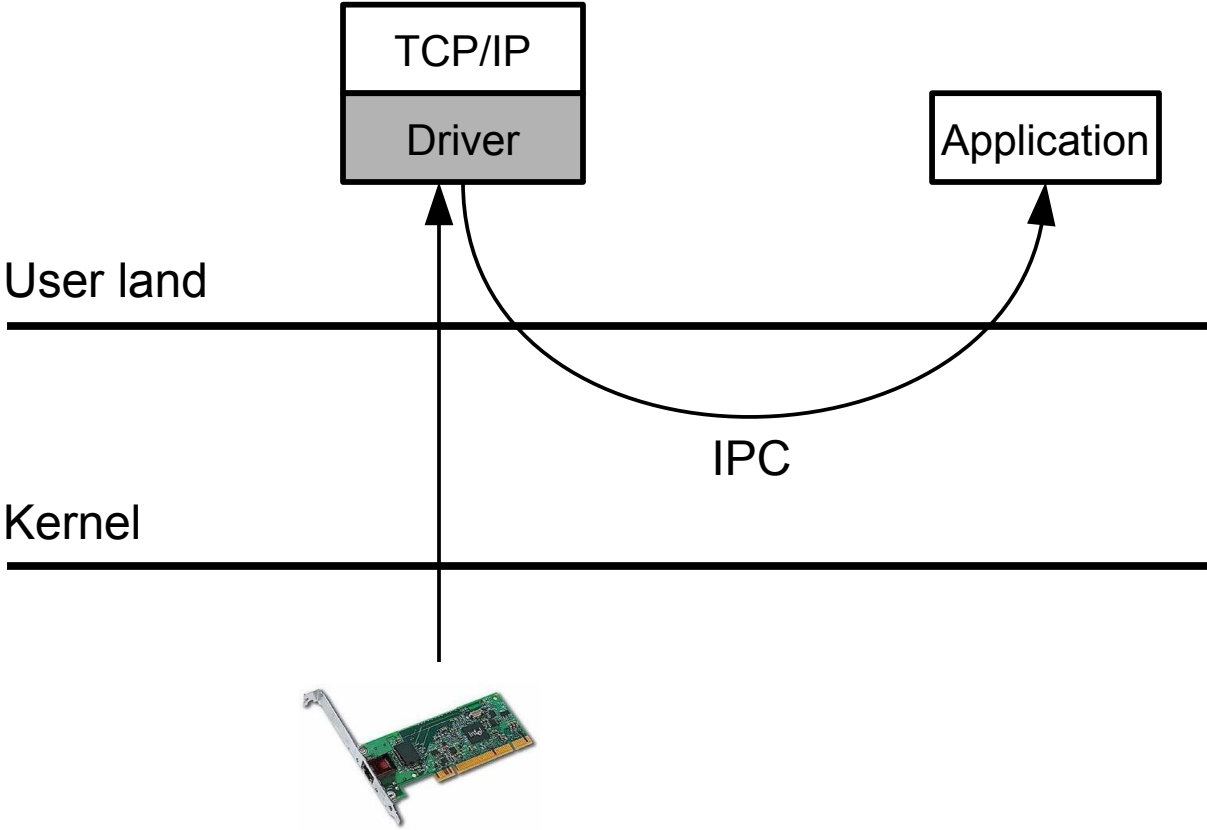
# User-level drivers in $\mu$ -kernel OSs



# User-level drivers in $\mu$ -kernel OSs



# User-level drivers in $\mu$ -kernel OSs



# Driver performance characteristics

---



- I/O throughput
  - Can the driver saturate the device?
- I/O latency
  - How does the driver affect the latency of a single I/O request?
- CPU utilisation
  - How much CPU overhead does the driver introduce?

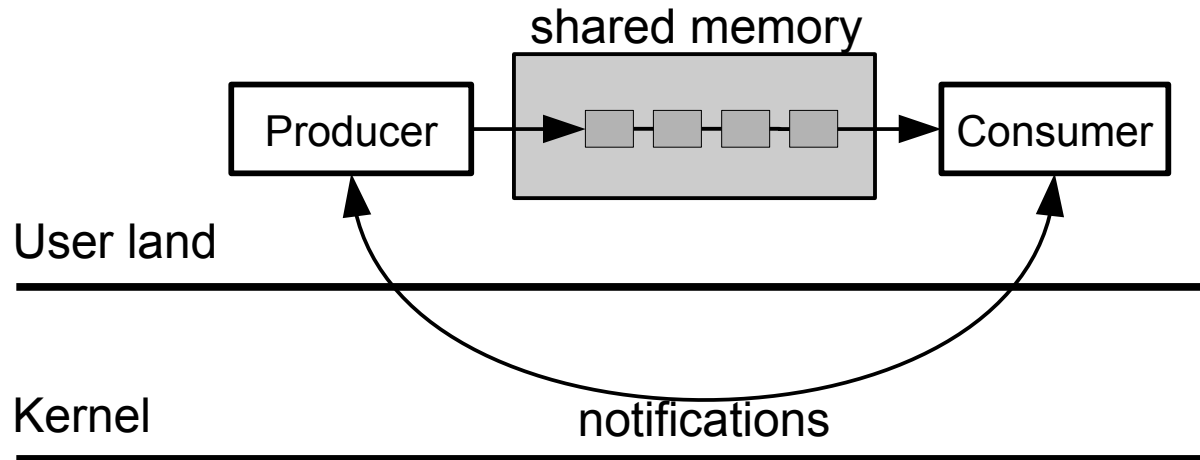
# Improving the performance of ULD

---



- Ways to improve user-level driver performance
  - Shared-memory communication
  - Request queueing
  - Interrupt coalescing

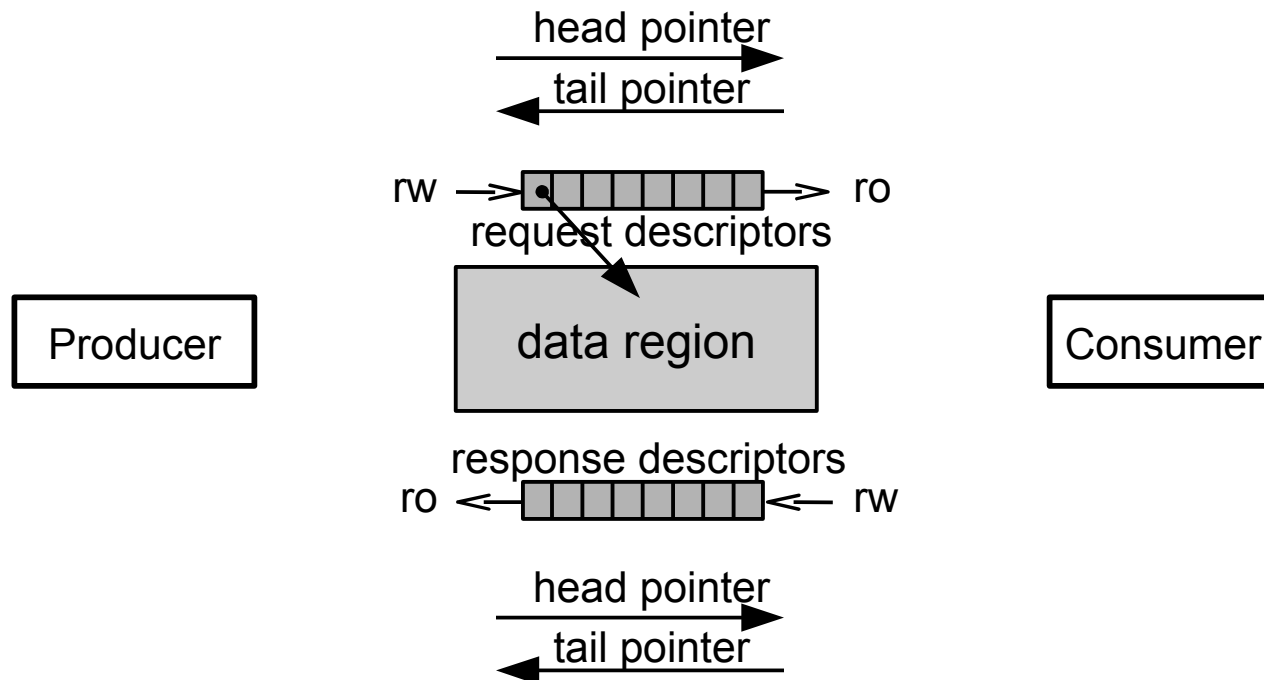
# Implementing efficient shared-memory communication



- Issues:
  - Resource accounting
  - Safety
  - Asynchronous notifications

# Rbufs

- Proposed in the Nemesis microkernel-based multimedia OS



# Early implementations of ULD

---



- Michigan Terminal System [1970's]
  - \_ OS for IBM System/360
  - \_ Apparently, the first to support user-level drivers
- Mach [1985-1994]
  - \_ Distributed multi-personality  $\mu$ -kernel-based multi-server OS
  - \_ High IPC overhead
  - \_ Eventually, moved drivers back into the kernel
- L3 [1987-1993]
  - \_ Persistent  $\mu$ -kernel-based OS
  - \_ High IPC overhead
  - \_ Improved IPC design: 20-fold performance improvement
  - \_ No data on driver performance available

# More recent implementations

---



- Sawmill [~2000]
  - Multiserver OS based on automatic refactoring of the Linux kernel
  - Hampered by software engineering problems
  - No data on driver performance available
- DROPS [1998]
  - L4 Fiasco-based real-time OS
  - ~100% CPU overhead due to user-level drivers
- Fluke [1996]
  - ~100% CPU overhead
- Mungi [1993—2006]
  - Single-address-space distributed L4-based OS
  - Low-overhead user-level I/O demonstrated for a disk driver

# Currently active systems

---

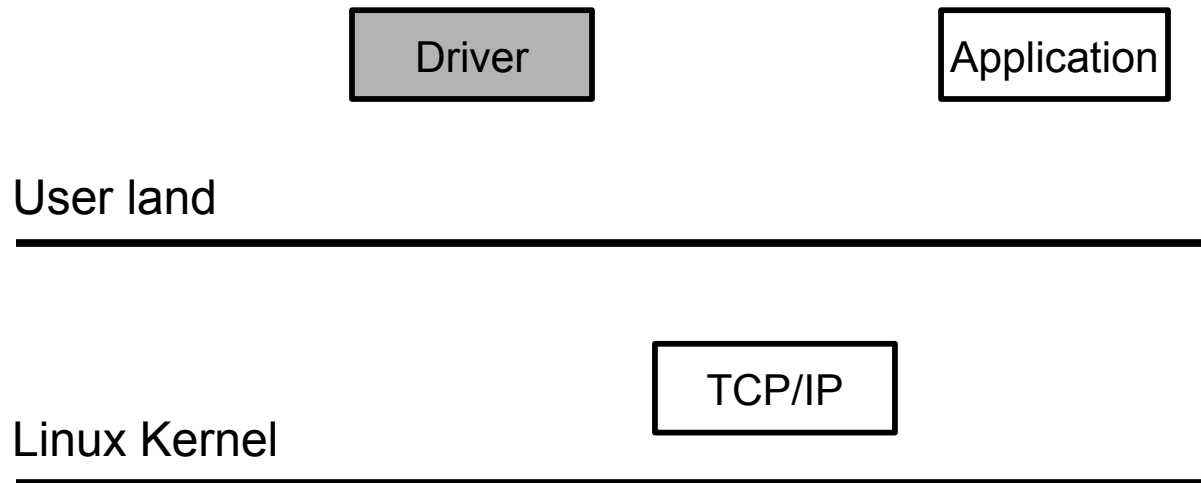


- Research
  - seL4
  - MINIX3
  - Nexus
- Commercial
  - OKL4
  - QNX
  - GreenHills INTEGRITY

# User-level drivers in a monolithic OS

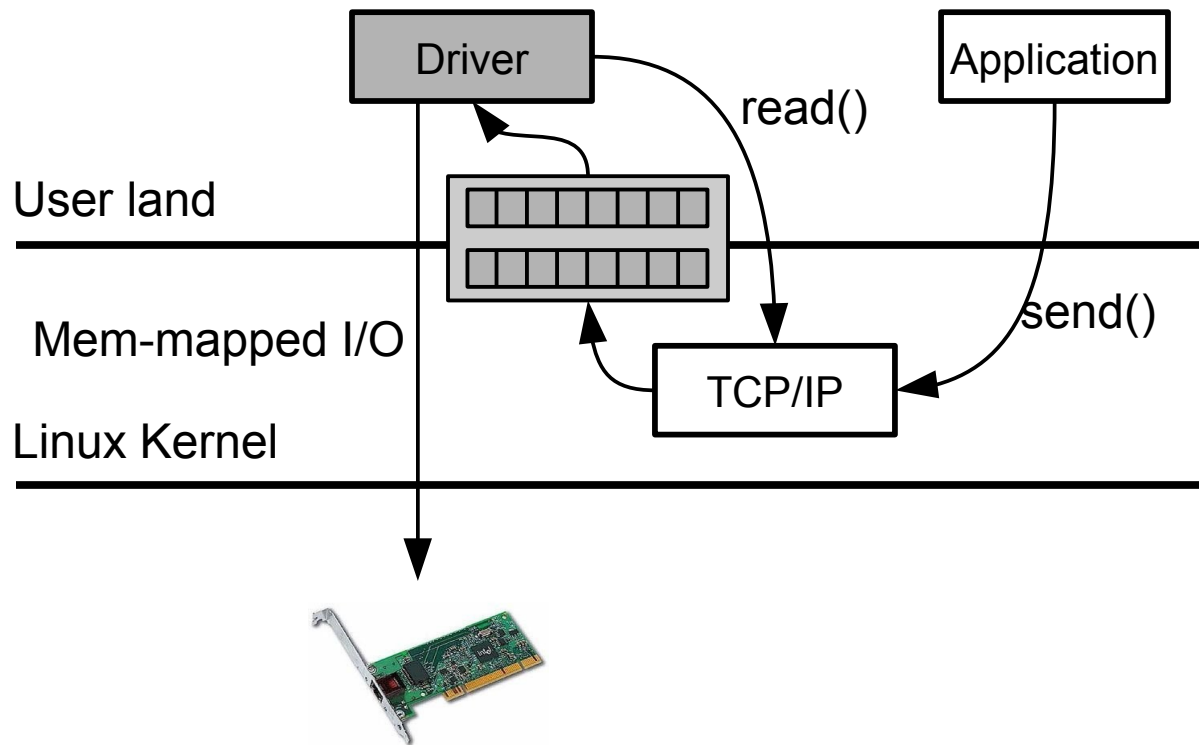


Ben Leslie et al. User-level device drivers: Achieved performance, 2005



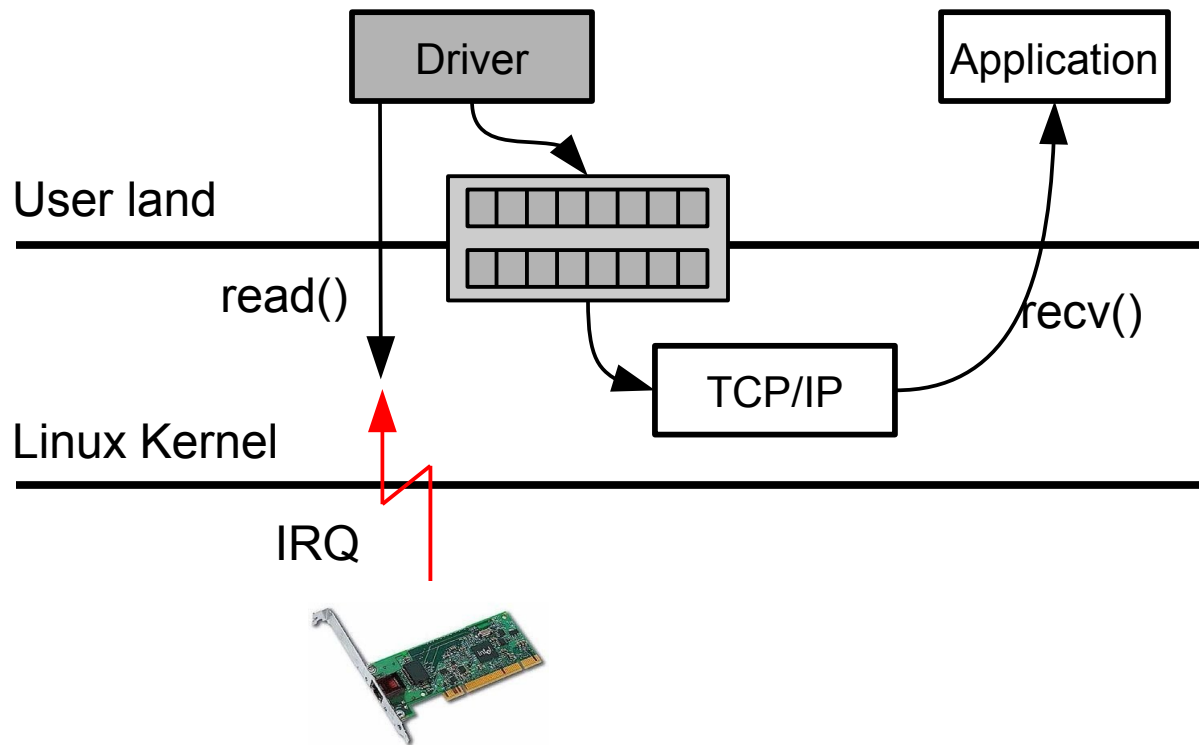
# User-level drivers in a monolithic OS

Ben Leslie et al. User-level device drivers: Achieved performance, 2005



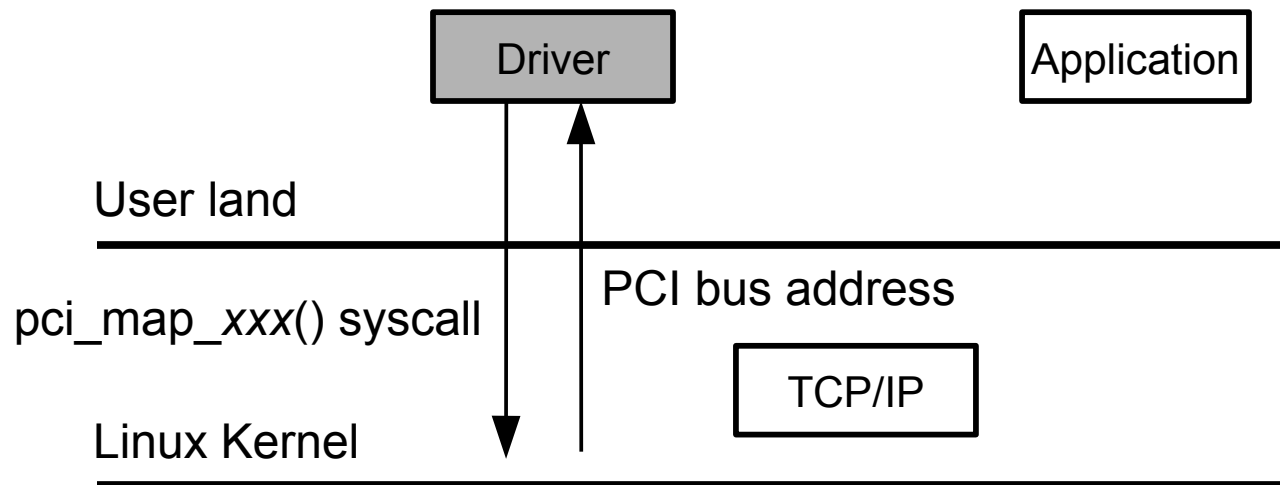
# User-level drivers in a monolithic OS

Ben Leslie et al. User-level device drivers: Achieved performance, 2005



# User-level drivers in a monolithic OS

Ben Leslie et al. User-level device drivers: Achieved performance, 2005



# User-level drivers in a monolithic OS

---

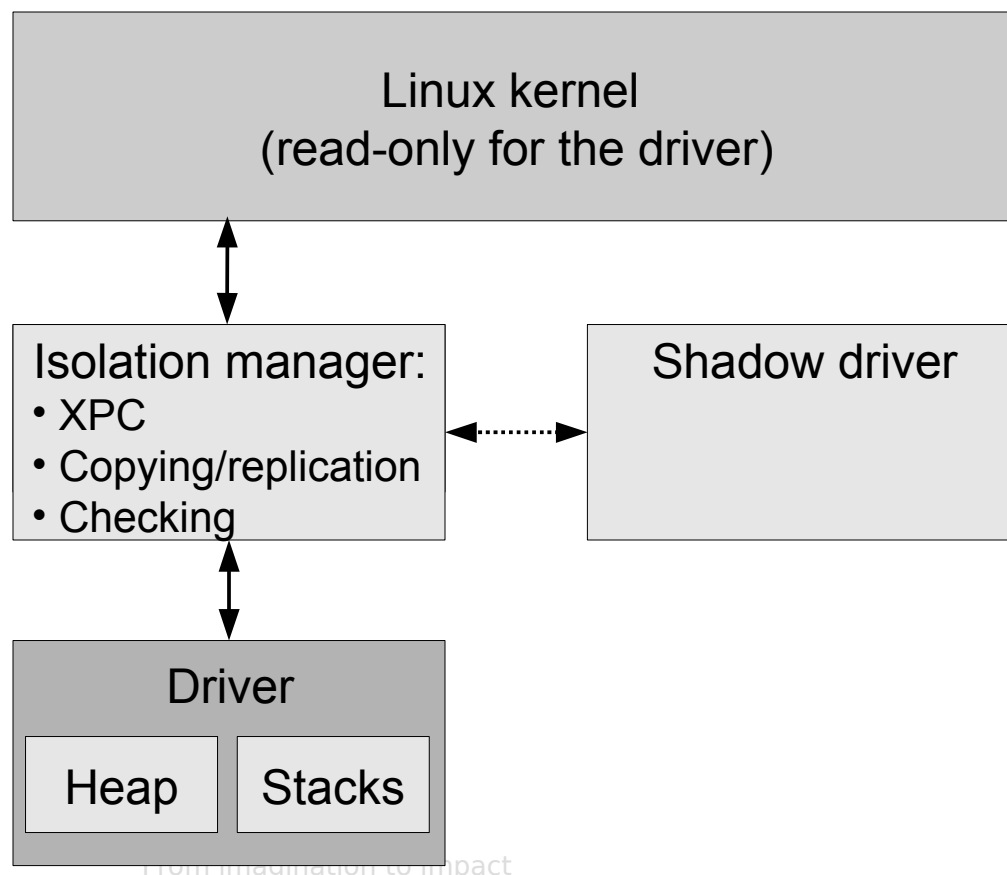


Ben Leslie et al. User-level device drivers: Achieved performance, 2005

- Performance
  - Up to 7% throughput degradation
  - Up to 17% CPU overhead
  - Aggressive use of interrupt rate limiting potentially affects latency (not measured).

- A complete device-driver reliability solution for Linux:

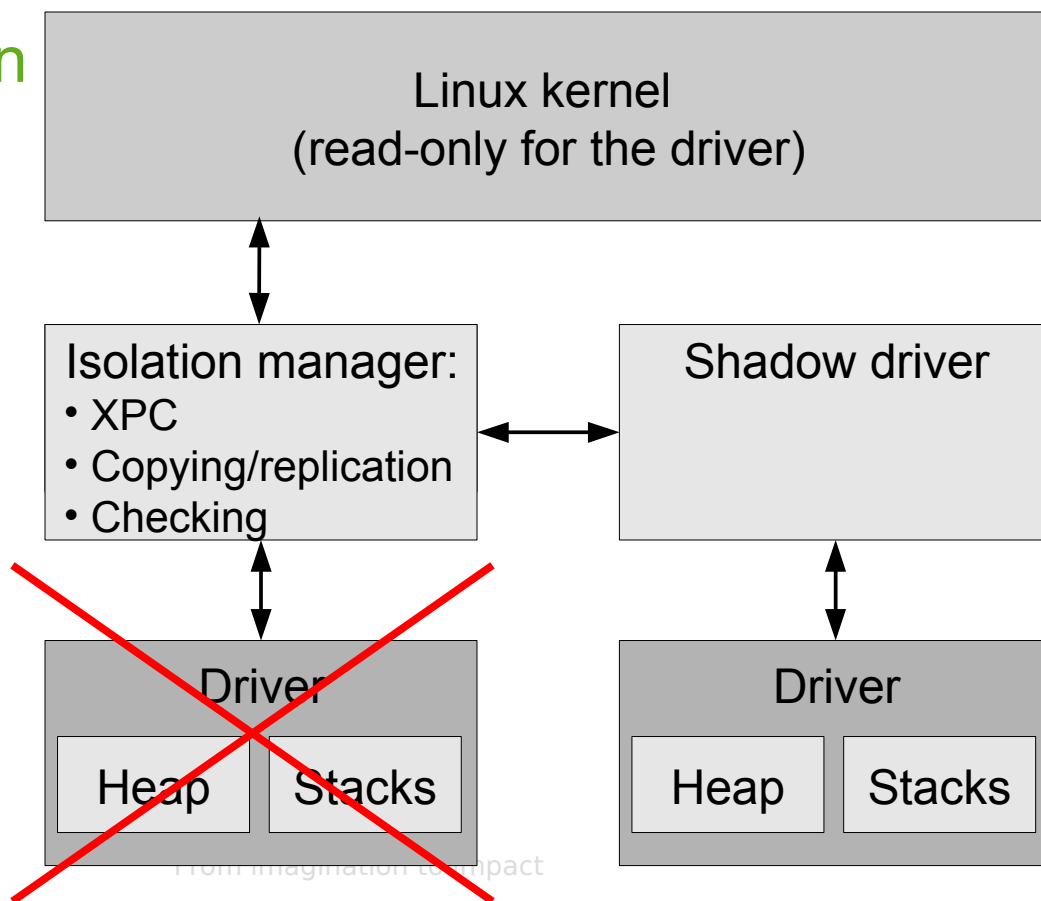
- Fault isolation
- Fault detection
- Recovery



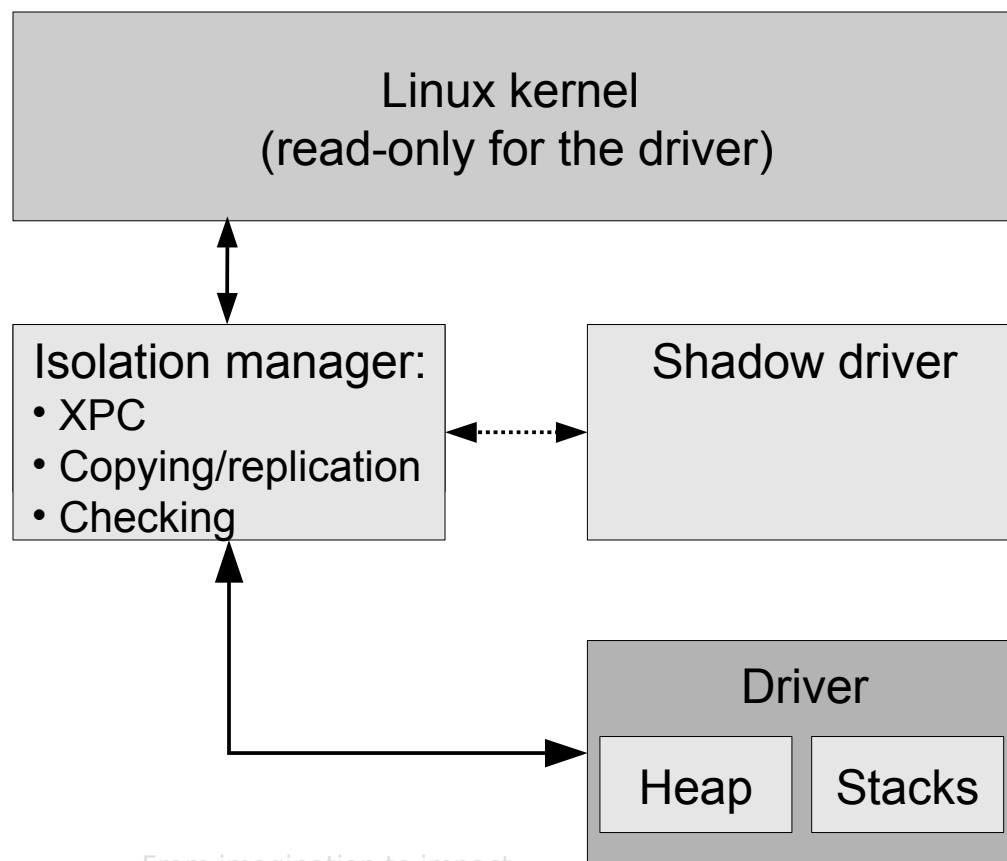
# Nooks

- A complete device-driver reliability solution for Linux:

- Fault isolation
- Fault detection
- Recovery



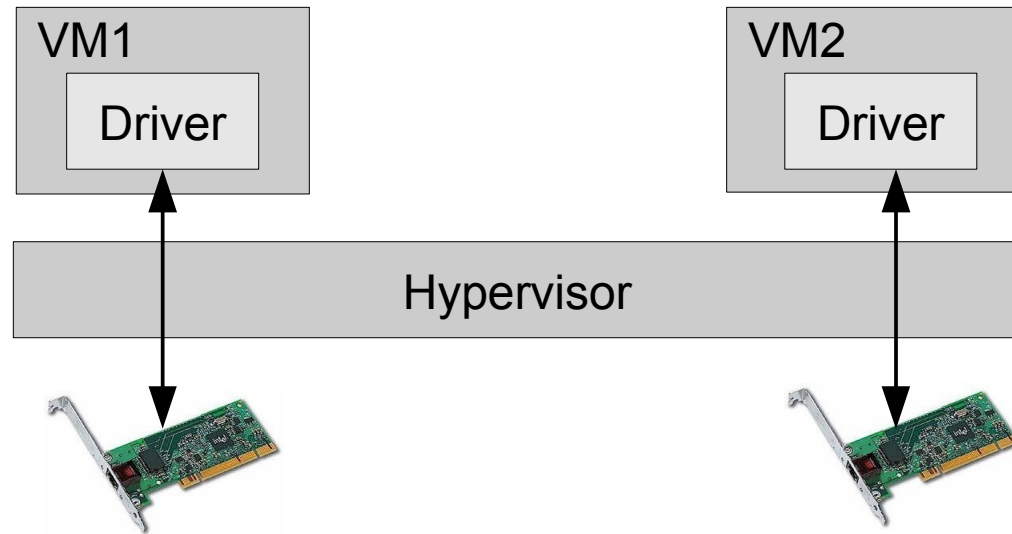
- A complete device-driver reliability solution for Linux:
  - Fault isolation
  - Fault detection
  - Recovery



- A complete device-driver reliability solution for Linux:
  - Fault isolation
  - Fault detection
  - Recovery
- Problems
  - The driver interface in Linux is not well defined. Nooks must simulate the behaviour of hundreds of kernel and driver entry points.
- Performance
  - 10% throughput degradation
  - 80% CPU overhead

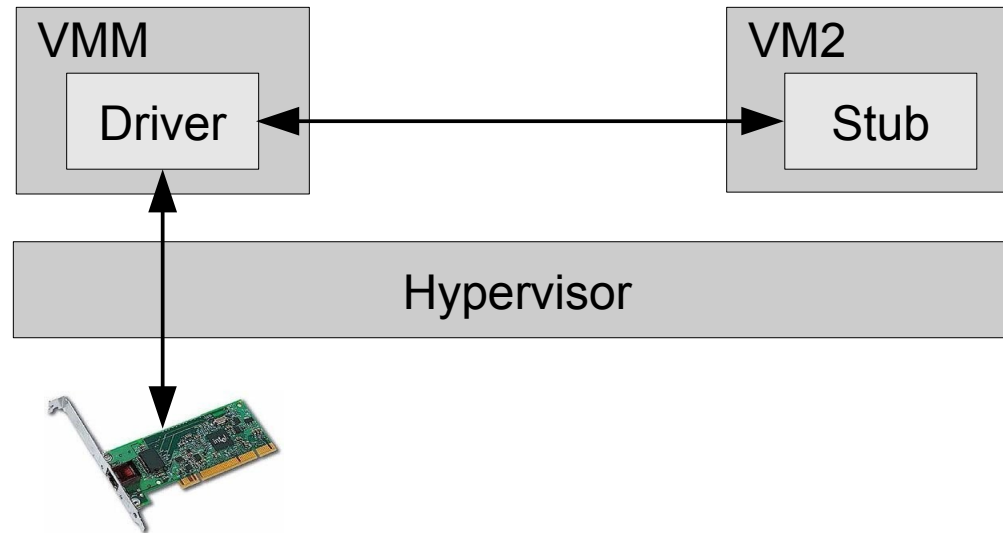
# Virtualisation and user-level drivers

- Direct I/O

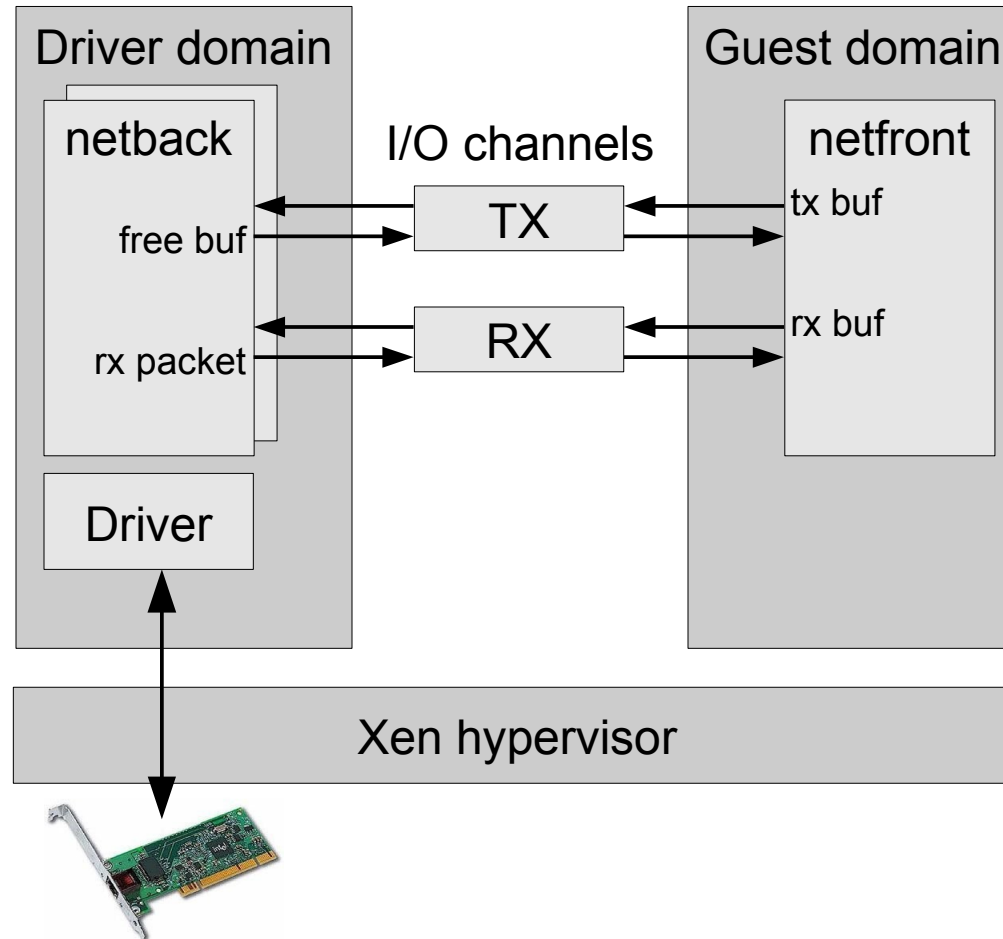


# Virtualisation and user-level drivers

- Paravirtualised I/O

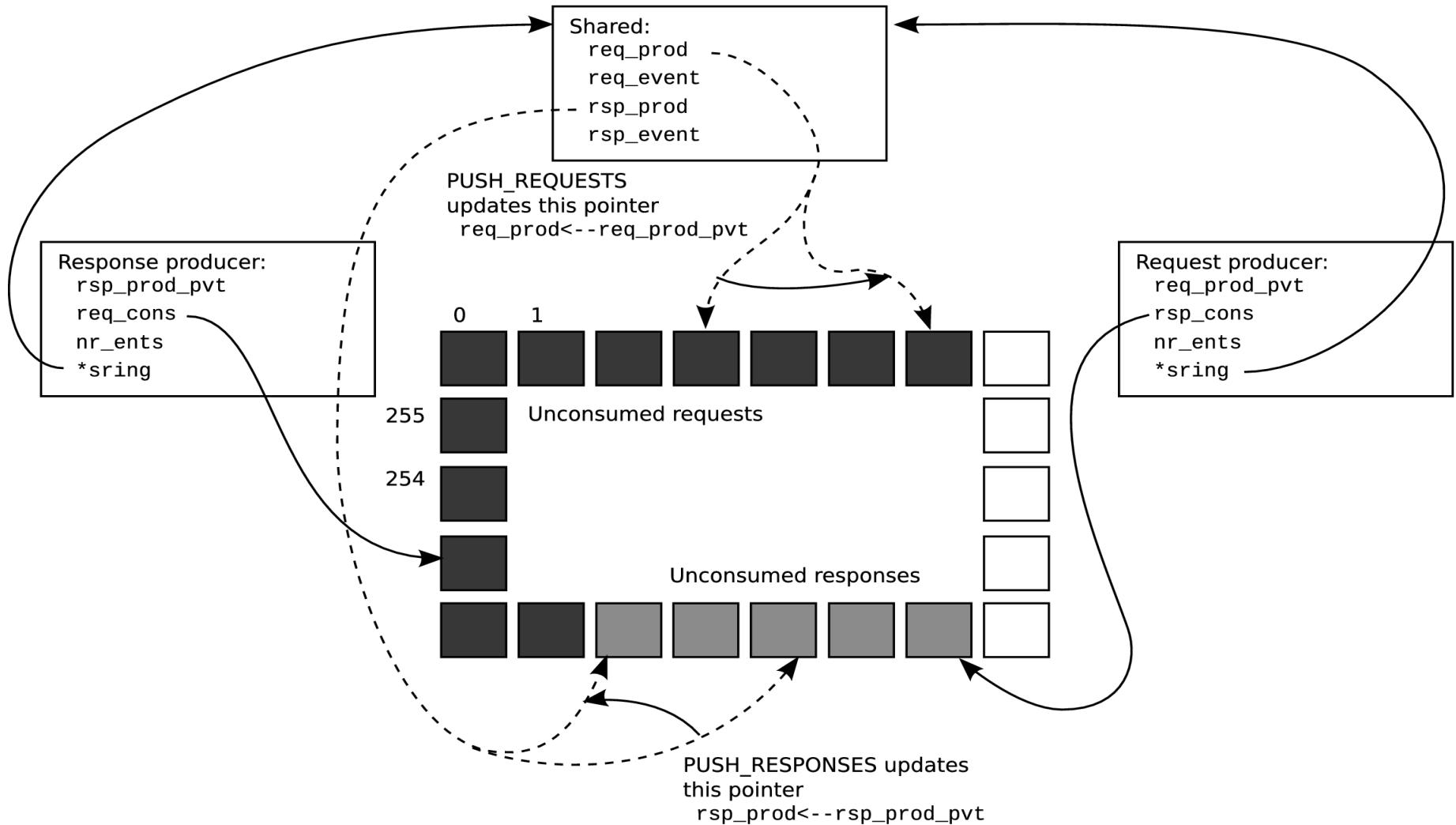


# Paravirtualised I/O in Xen



- Xen I/O channels are similar to rbufs, but use a single circular buffer for both requests and completions and rely on mapping rather than sharing

# Xen I/O channels



- Performance overhead of the original implementation: 300%
  - \_ Long critical path (increased instructions per packet)
  - \_ Higher TLB and cache miss rates (more cycles per instructions)
  - \_ Overhead of mapping
- Optimisations
  - \_ Avoid mapping on the send path (the driver does not need to “see” the packet content)
  - \_ Replace mapping with copying on the receive path
  - \_ Avoid unaligned copies
  - \_ Optimised implementation of page mapping
  - \_ CPU overhead down to 97% (worst-case receive path)

- Implementing drivers using safe languages
  - Java OSs: KaffeOS, JX
    - Every process runs in a separate protection domain with a private heap. Process boundaries are enforced by the language runtime. Communication is based on shared heaps.
  - House (Haskell OS)
    - Bare-metal Haskell runtime. The kernel and drivers are in Haskell.
    - User programs can be written in any language.
  - SafeDrive
    - Extends C with pointer type annotations enforced via static and runtime checking
    - `unsigned n;`  
`struct e1000 buffer * count(n) bufinfo;`

- Implementing drivers using safe languages
  - Singularity OS
    - The entire OS is implemented in Sing#
    - Every driver is encapsulated in a separate software-isolated process
    - Processes communicated via messages sent across channels
    - Sing# provides means to specify and statically enforce channel protocols

- Static analysis
  - SLAM, Blast, Coverity
  - Generic programming faults
    - Release acquired locks; do not acquire a lock twice
    - Do not dereference user pointers
    - Check potentially NULL-pointers returned from routine
  - Driver-specific properties
    - “if a driver calls another driver that is lower in the stack, then the dispatch routine returns the same status that was returned by the lower driver”
    - “drivers mark I/O request packets as pending while queuing them”
  - Limitations
    - Many properties are beyond reach of current tools or are theoretically undecidable (e.g., memory safety)

Questions?