TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

OPERATING-SYSTEM CONSTRUCTION

Material based on slides by Olaf Spinczyk, Universität Osnabrück

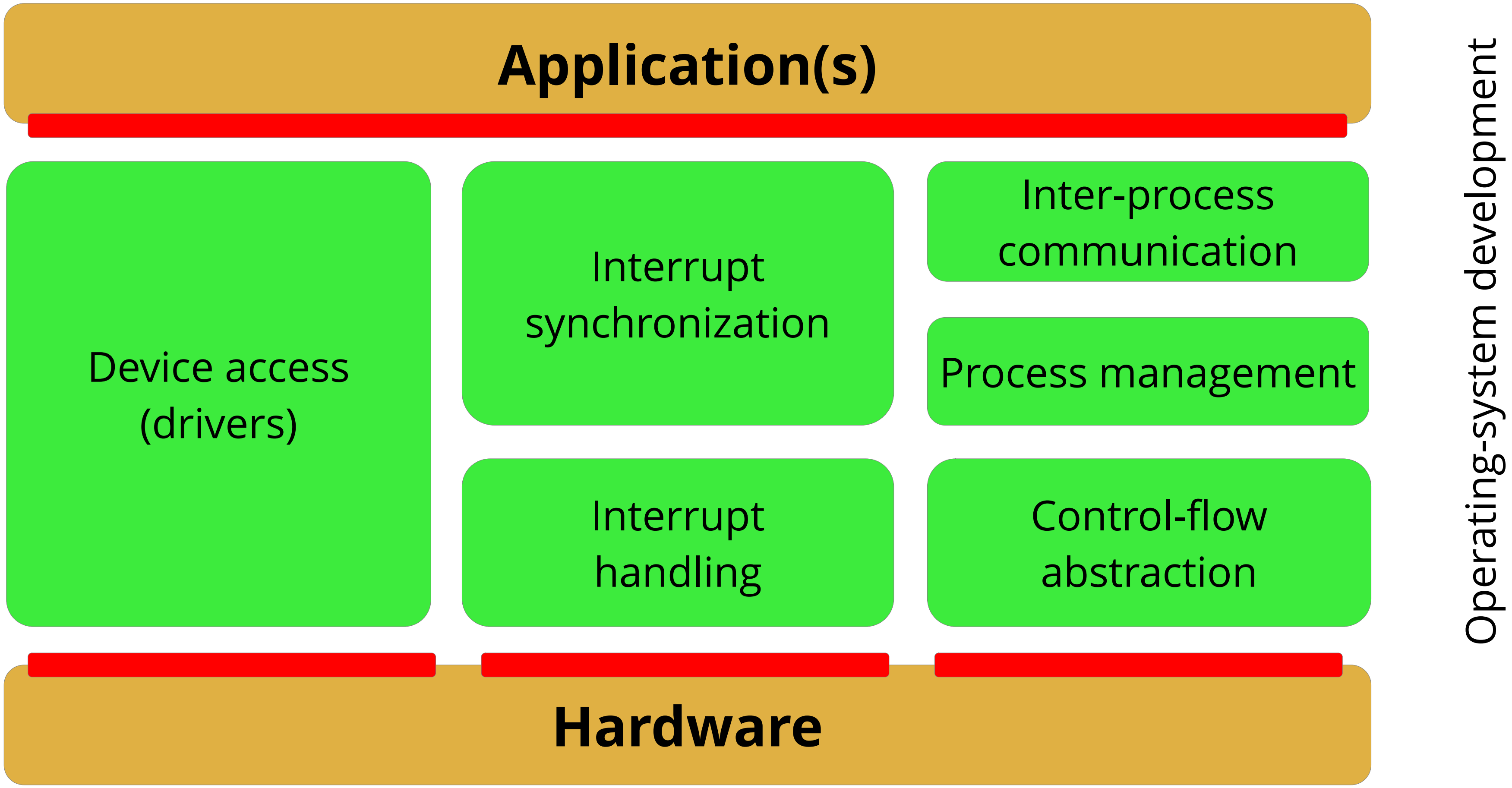
PC Bus Systems (and how to program them)

https://tud.de/inf/os/studium/vorlesungen/betriebssystembau

HORST SCHIRMEIER

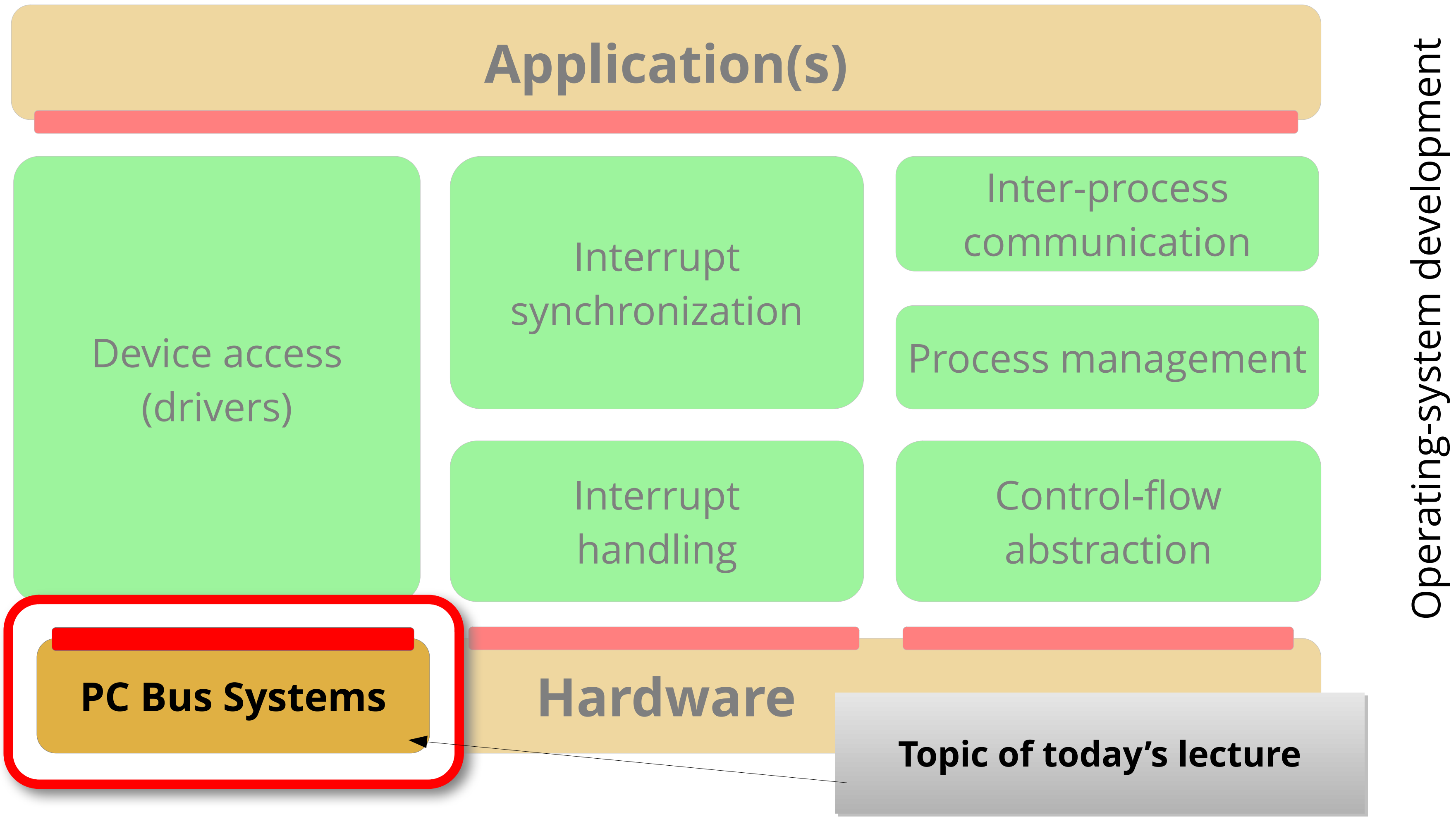
Overview: Lectures

Structure of the “OO-StuBS” operating system:



Overview: Lectures

Structure of the “OO-StuBS” operating system:



Topic of today’s lecture

Agenda

- History
  - PC Bus Systems
- PCI Bus
- PCI from an Operating-System Perspective
  - Initialization, PCI BIOS, ...
- PCI Extensions and Successors
  - AGP
  - PCI-X
  - PCI Express
  - HyperTransport
- Summary

Agenda

- **History**
  - PC Bus Systems
- PCI Bus
- PCI from an Operating-System Perspective
  - Initialization, PCI BIOS, ...
- PCI Extensions and Successors
  - AGP
  - PCI-X
  - PCI Express
  - HyperTransport
- Summary

History – PC Bus Systems

- Requirements on PC bus systems grew continuously:

Bus System	PC	ISA	VLB	MCA	EISA	…
CPUs	≥ 8088	≥ 286	≥ 386	≥ 386	≥ 386	
typ. clock freq.	4,7 MHz	8 MHz	25–50 MHz	10–25 MHz	8,33 MHz	
Multi-Master	no	no	yes (v.2)	yes	yes	
Bus Width	8 bits	16 bits	32/64 bits	32 bits	32 bits	
Address Space	1 MB	16 MB	4 GB	4 GB	4 GB	
Transfer Rate	1 MB/s	4–5 MB/s	40/64 MB/s (Burst)	40 MB/s (Burst)	33 MB/s (Burst)	

History – PC Bus Systems

- Requirements on PC bus systems grew continuously:

Bus System	… PCI	AGP	PCI-X	PCI Express	HyperTransport
CPUs	≥ 486	≥ 486	≥ P6	≥ P4 (Xeon)	≥ Hammer (AMD)
typ. clock freq.	33/66 MHz	66 MHz	up to 133 MHz	(variable)	(variable)
Multi-Master	yes	no (max. 1 dev.)	yes	point-to-point	yes, different topologies possible
Bus Width	32/64 bits	32 bits	32/64	up to 32 *lanes*	up to 32 *links*
Address Space	4 GB/16 EB	4 GB	4 GB/16 EB	4 GB/16 EB	4 GB/16 EB
Transfer Rate	132/528 MB/s (*Burst*)	*n* x 266 MB/s (1x, 2x, …8x)	1064 MB/s (Burst)	*n* x 250 MB/s (1x, 2x, …16x)	n x 100–400 MB/s (Burst, per *link*)

Agenda

- History
  - PC Bus Systems
- **PCI Bus**
- **PCI from an Operating-System Perspective**
  - Initialization, PCI BIOS, …
- PCI Extensions and Successors
  - AGP
  - PCI-X
  - PCI Express
  - HyperTransport
- Summary

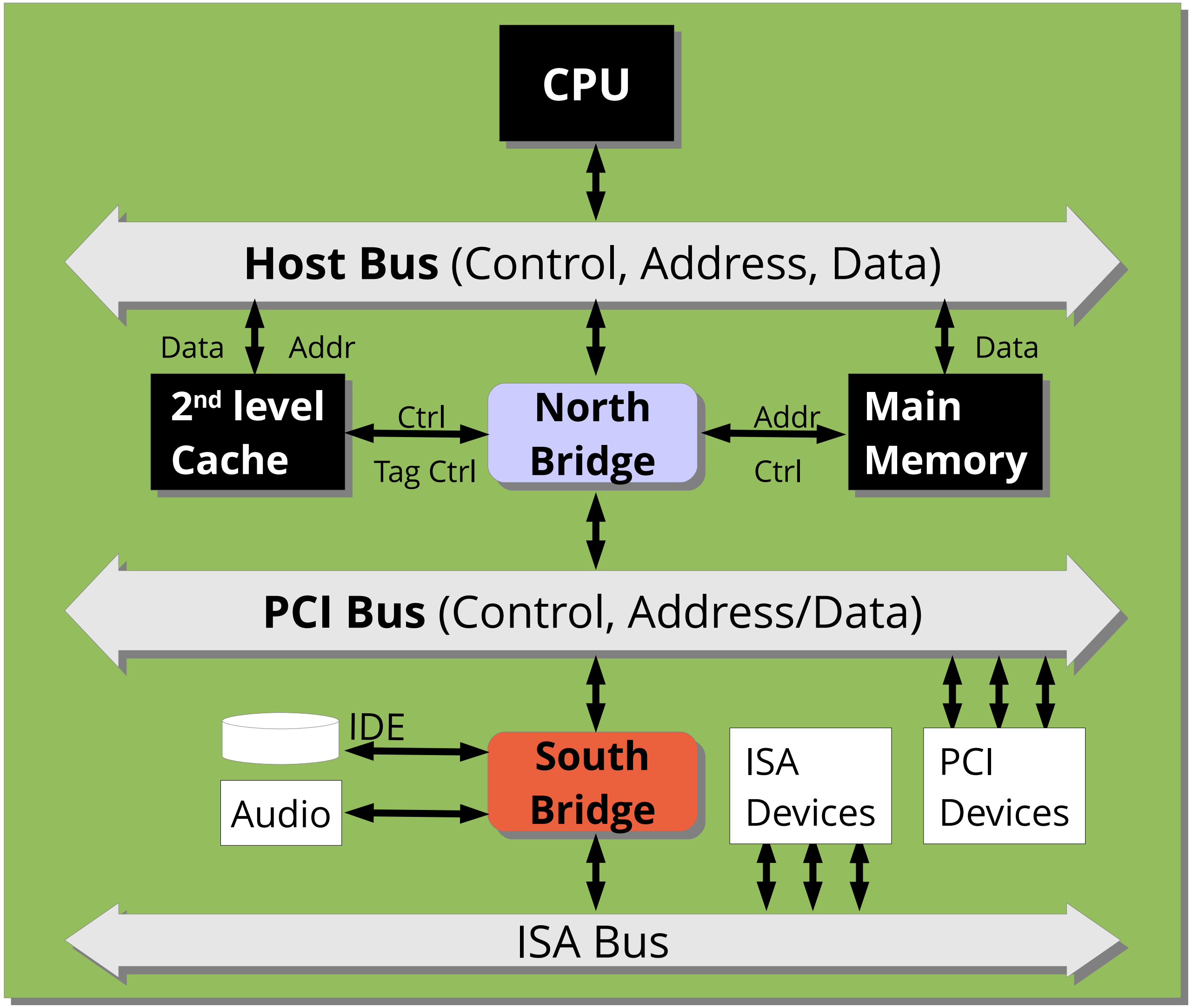
PCI-based PC Systems

- **Typical architecture** of the first PCI systems:

The ***North Bridge*** **decouples host bus** and **PCI bus**. Thus, PCI units and CPU can work in parallel.

The PCI connection between North and ***South Bridge*** was later replaced by something faster.

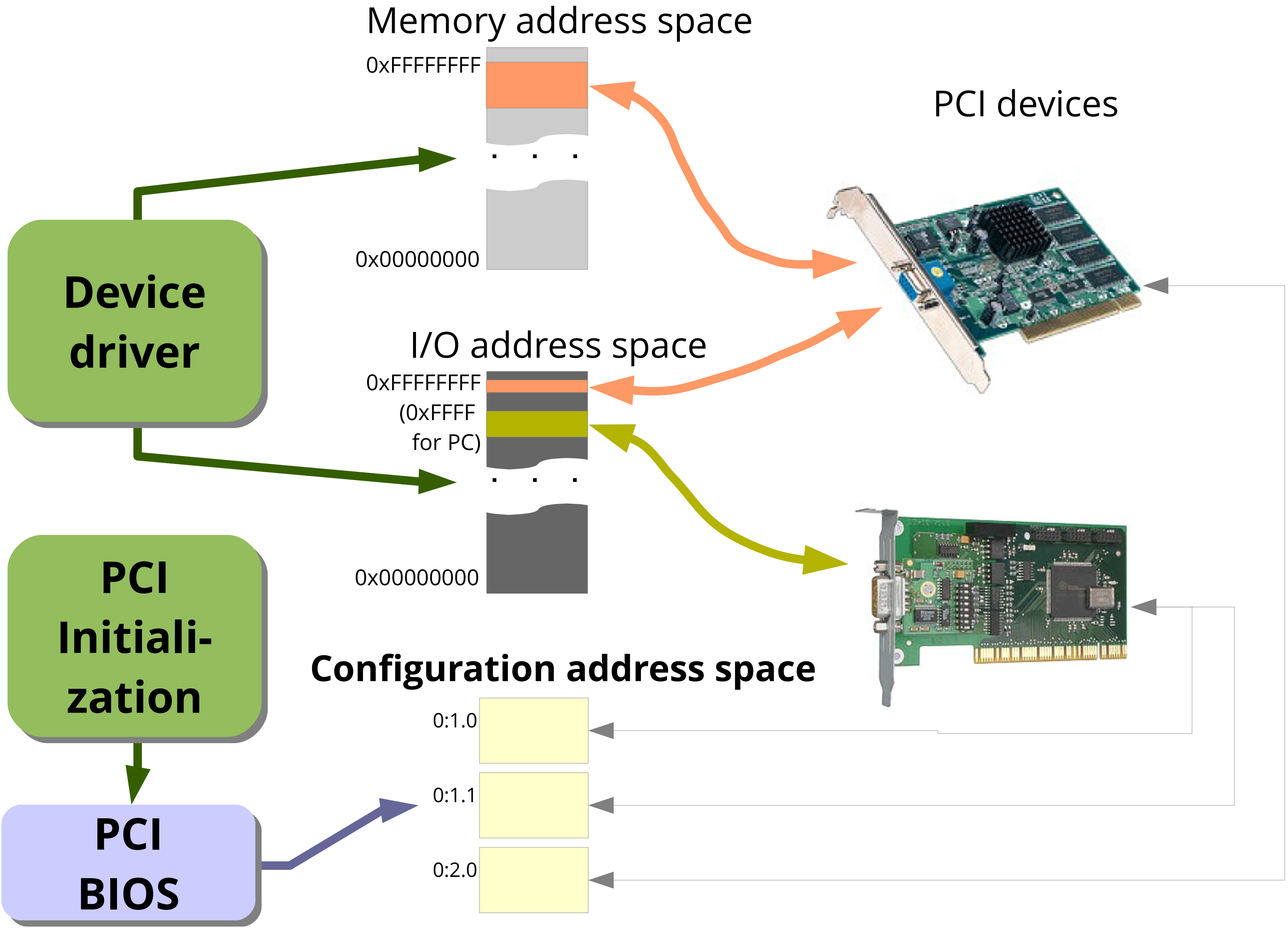
Bridges help integrating ISA and PCI transparently in one system.



PCI – Basic Data

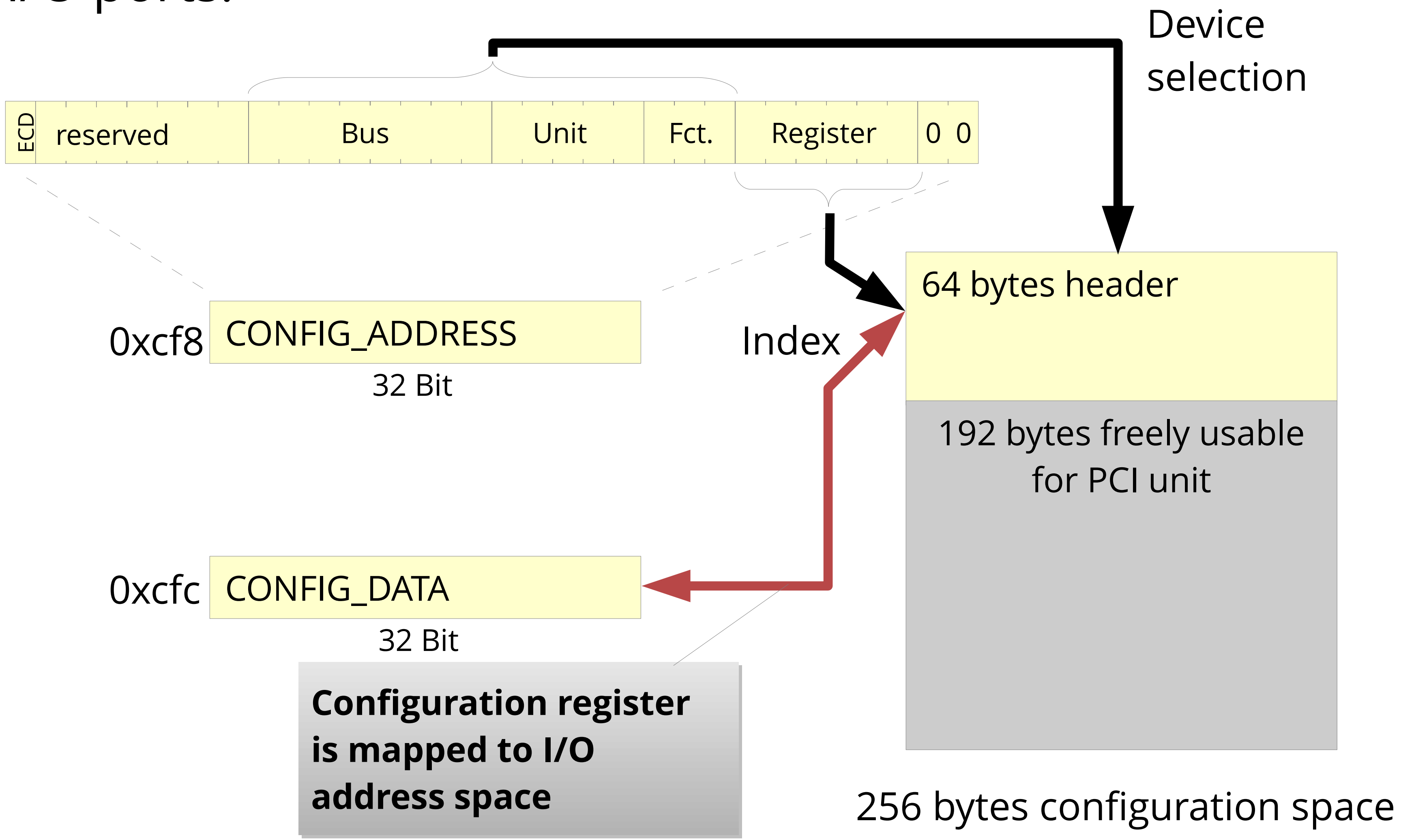
- Specification version 1.0 by Intel (1991)
  - Since 1993, the PCI-SIG issues specifications.
- CPU-type independent
  - PCI also exists in SPARC, Alpha, Arm and PowerPC systems!
- 32/64 bit, multiplexed address/data bus
- in burst mode max. 132 MB/s respectively 264 MB/s
- 4 interrupt lines (INTA-D)
- Scalable due to bridges and multi-function units
- *Multi-Master* capabilities (better than classic DMA)
- Mechanism to detect and auto-configure devices (resource allocation)

Interacting with PCI Devices



The PCI Configuration Space (1)

- On the PC, the configuration space is accessed indirectly via I/O ports:



Device selection

ECD | reserved | Bus | Unit | Fct. | Register | 0 0

0xcf8 CONFIG_ADDRESS

32 Bit

Index

0xcfc CONFIG_DATA

32 Bit

Configuration register is mapped to I/O address space

64 bytes header

192 bytes freely usable for PCI unit

256 bytes configuration space

The PCI Configuration Space (2)

- 64-byte header format:

	31	16	15	0
0x00	Device ID		Vendor ID	
0x04	Status		Command	
0x08	Class Code			Rev. ID
0x0c	BIST	Header	Latency	CLG
0x10	Base address registers			
0x14				
0x18				
0x1c				
0x20				
0x24				
0x28	reserved or *Cardbus CIS Pointer*			
0x2c	reserved or Subsystem IDs			
0x30	Expansion ROM base address			
0x34	reserved or *Capabilities Pointer*			
0x38	reserved			
0x3c	MaxLat	MinGNT	INT pin	INT line

Vendor ID 0xffff means ‘non-existent’.

Header bit 7=1 indicates a **multi-function device**.

BIST allows triggering a device’s built-in self test.

Vendor and **Device ID** uniquely identify a device. **Revision ID** and **Class Code** provide additional information.

The device can be activated and deactivated via the **Command** register.

Here we can define which address ranges the unit uses. At the same time, the device announces the (address) space it needs.

PCI Initialization

Before PCI devices can be accessed by their device drivers, the following steps have to be taken:

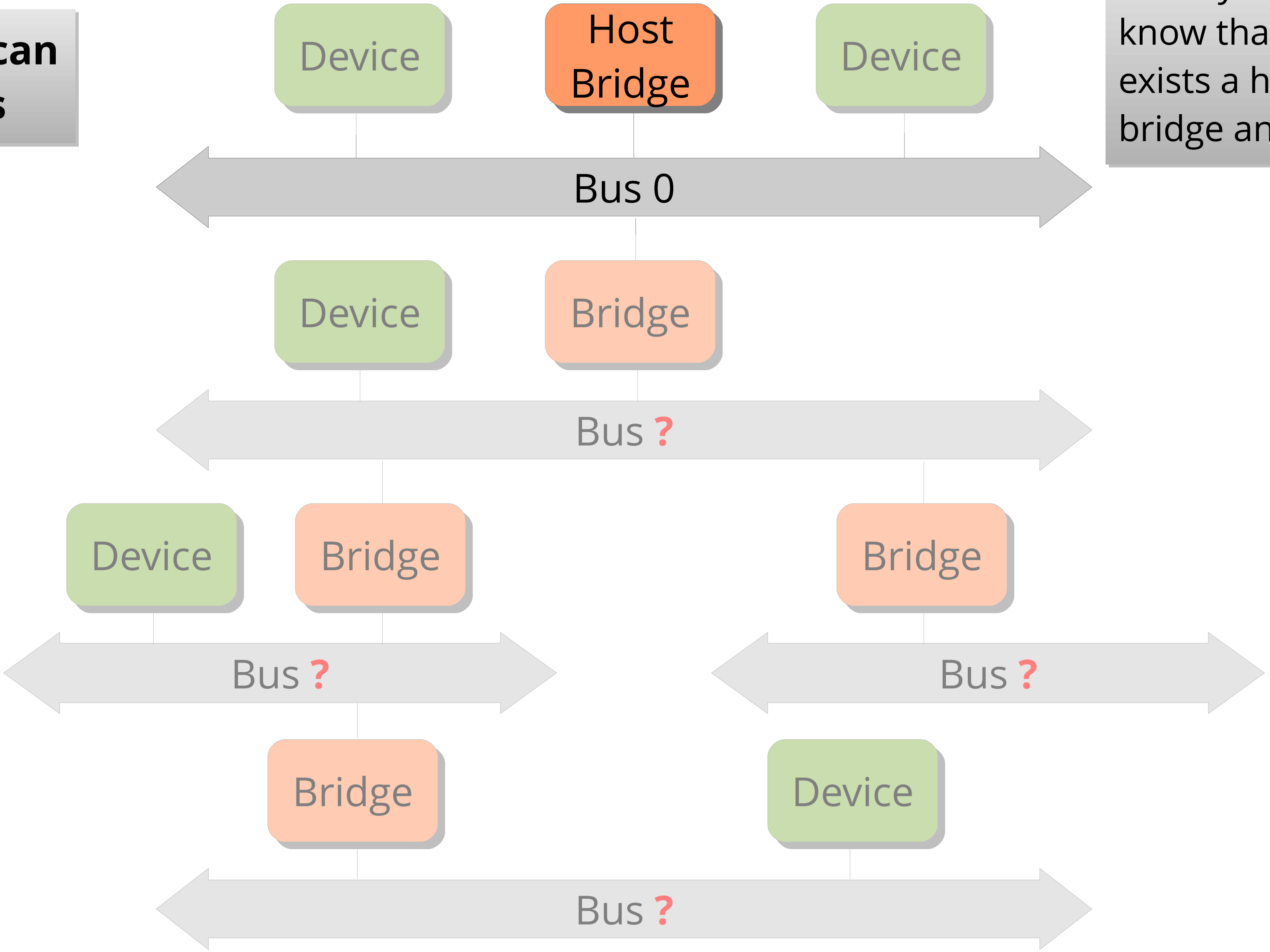
- Configure devices’ base address registers
- Configure **PCI Bridges**
  - Base address registers – depend on devices “below”!
  - Bus numbers (*Primary*, *Secondary*, *Subordinate*)
  - ***Subordinate*** is the number of the last downstream bus of this bridge
    (i.e., “below” a bridge are all buses numbered #secondary to #subordinate)

➔ BIOS or OS must explore and initialize the PCI bus structure step by step

  - Bus numbers and address ranges must never be allocated twice

PCI Initialization in Linux

Step 1: Scan for buses

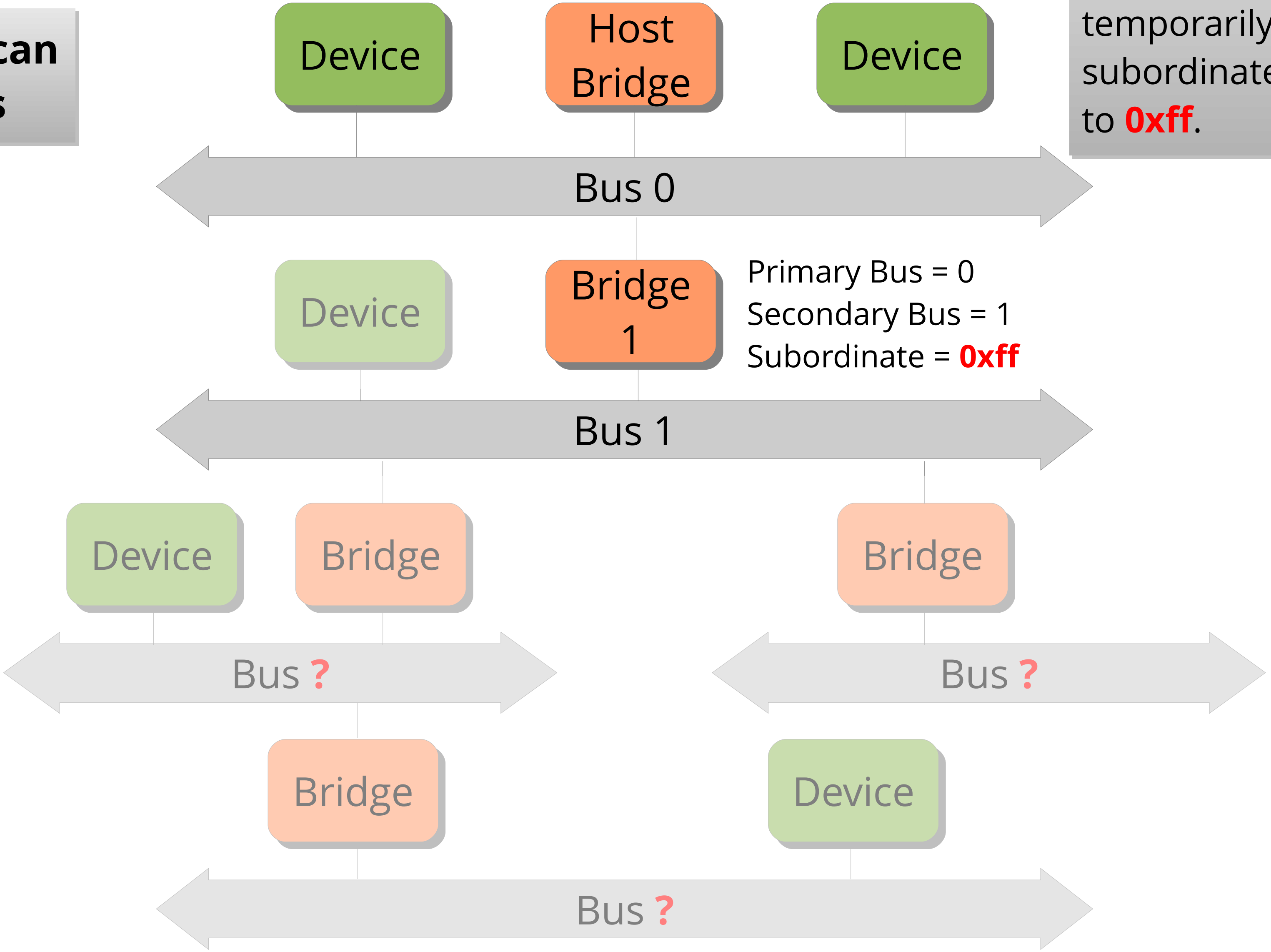


Initially we only know that there exists a host bridge and bus 0.

PCI Initialization in Linux

Step 1: Scan for buses

We find devices on **bus 0**. We initialize **bridge 1** and temporarily set its subordinate number to **0xff**.

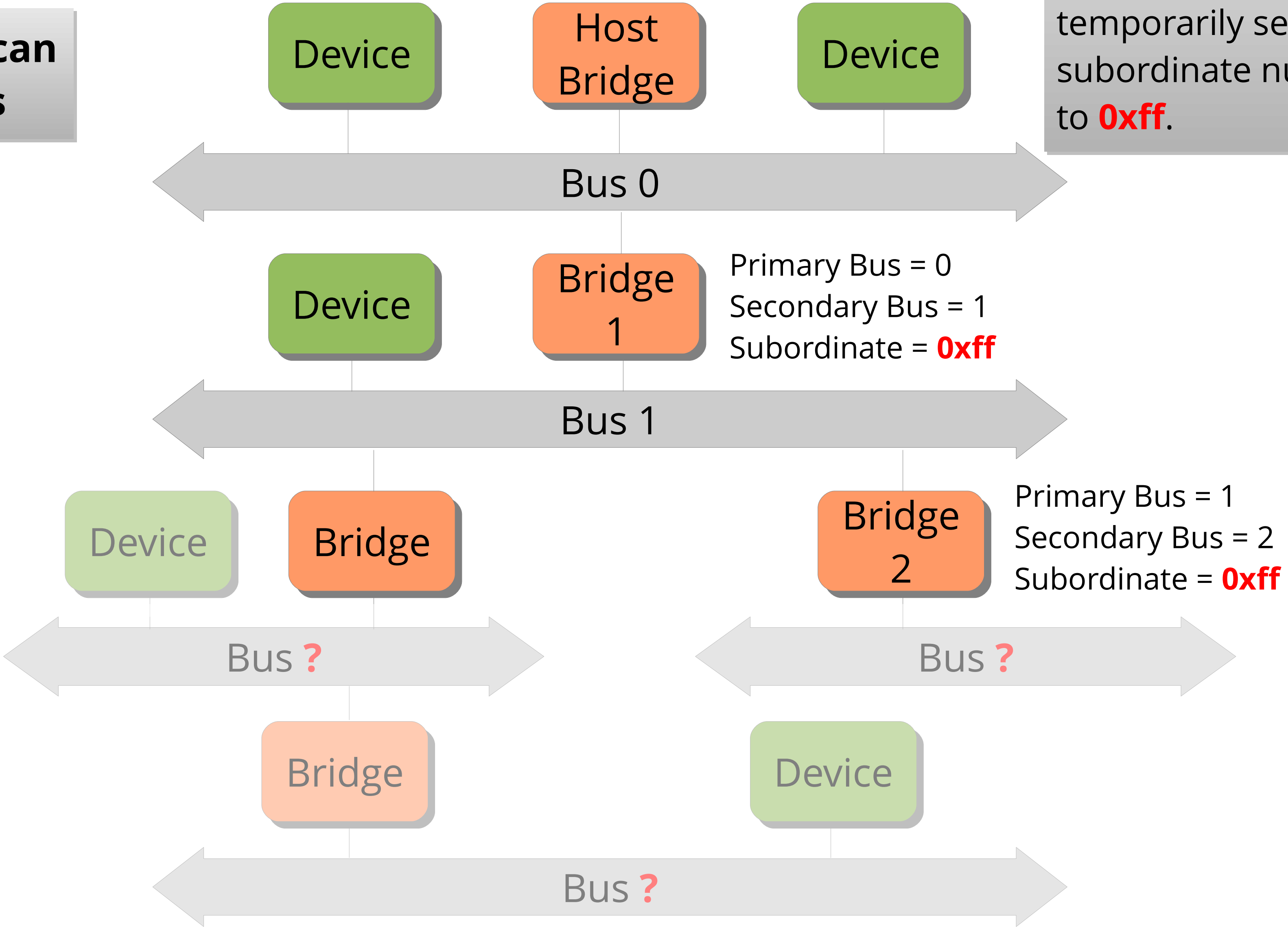


Primary Bus = 0
Secondary Bus = 1
Subordinate = **0xff**

PCI Initialization in Linux

Step 1: Scan for buses

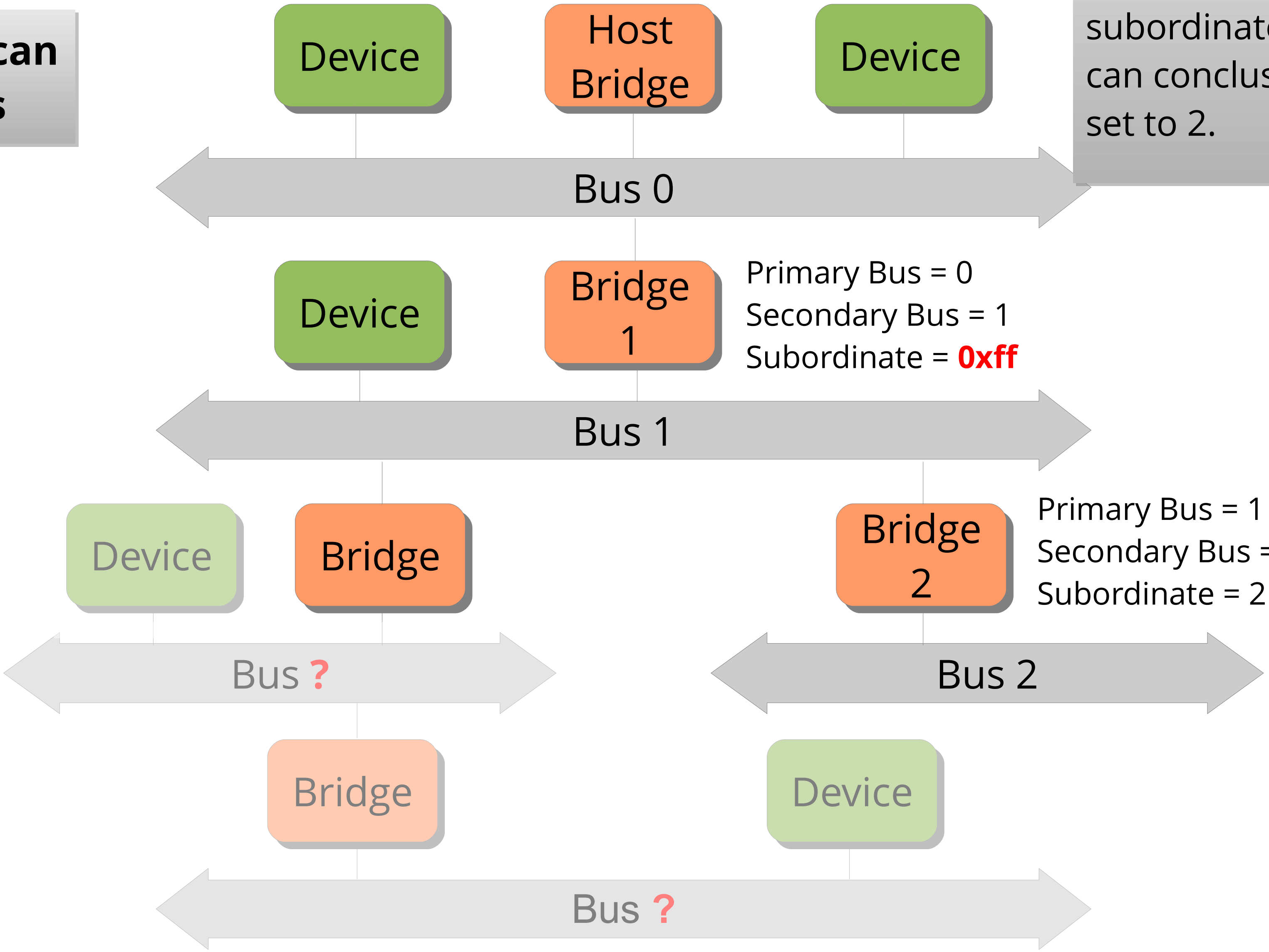
We find devices on **bus 1**, initialize **bridge 2**, and temporarily set its subordinate number to **0xff**.



PCI Initialization in Linux

We find devices on **bus 2**. As there are no more bridges, the subordinate number can conclusively be set to 2.

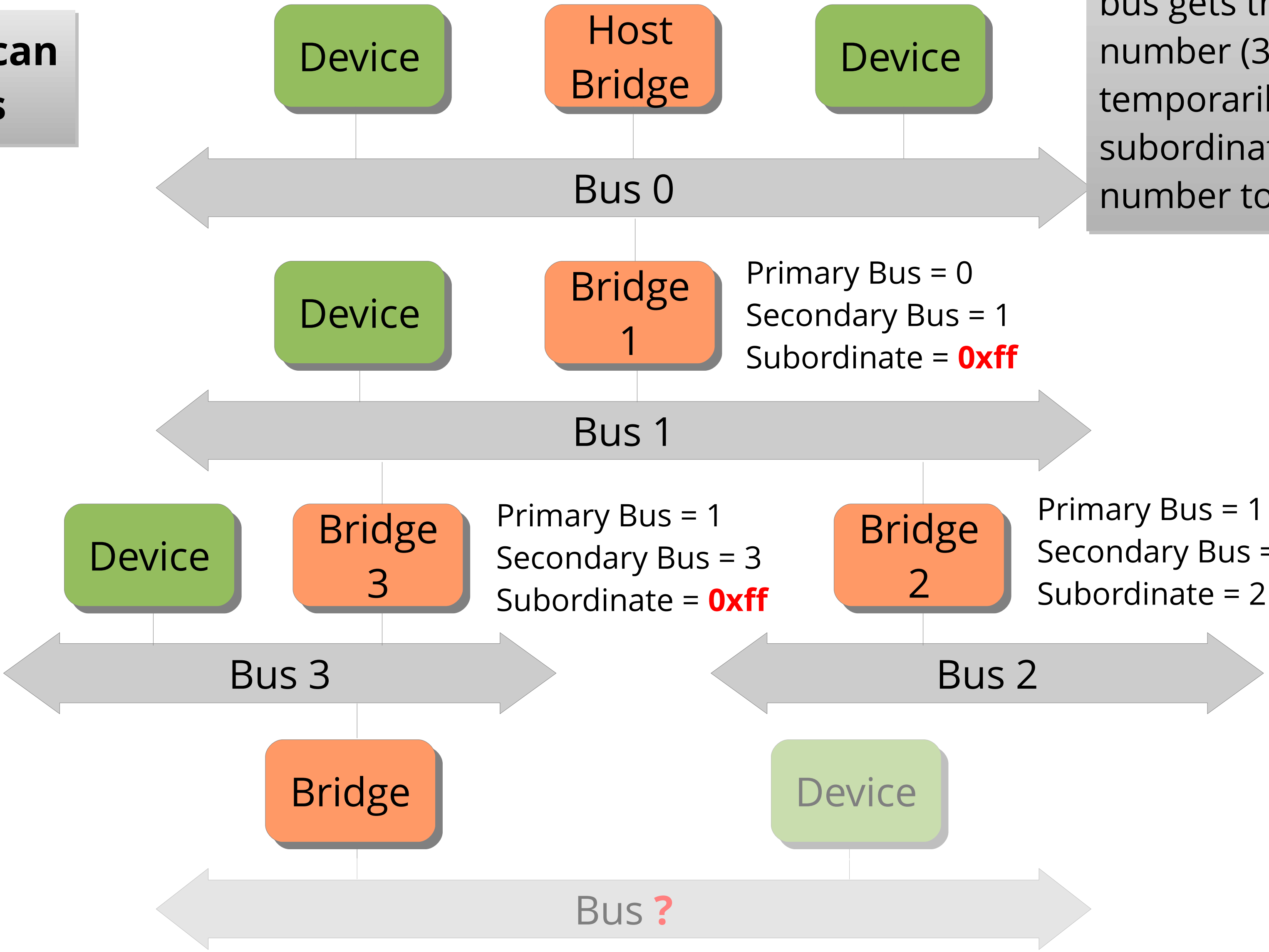
Step 1: Scan for buses



PCI Initialization in Linux

We continue on **bus 1** with **bridge 3**. The new bus gets the next number (3). We temporarily set the subordinate number to **0xff**.

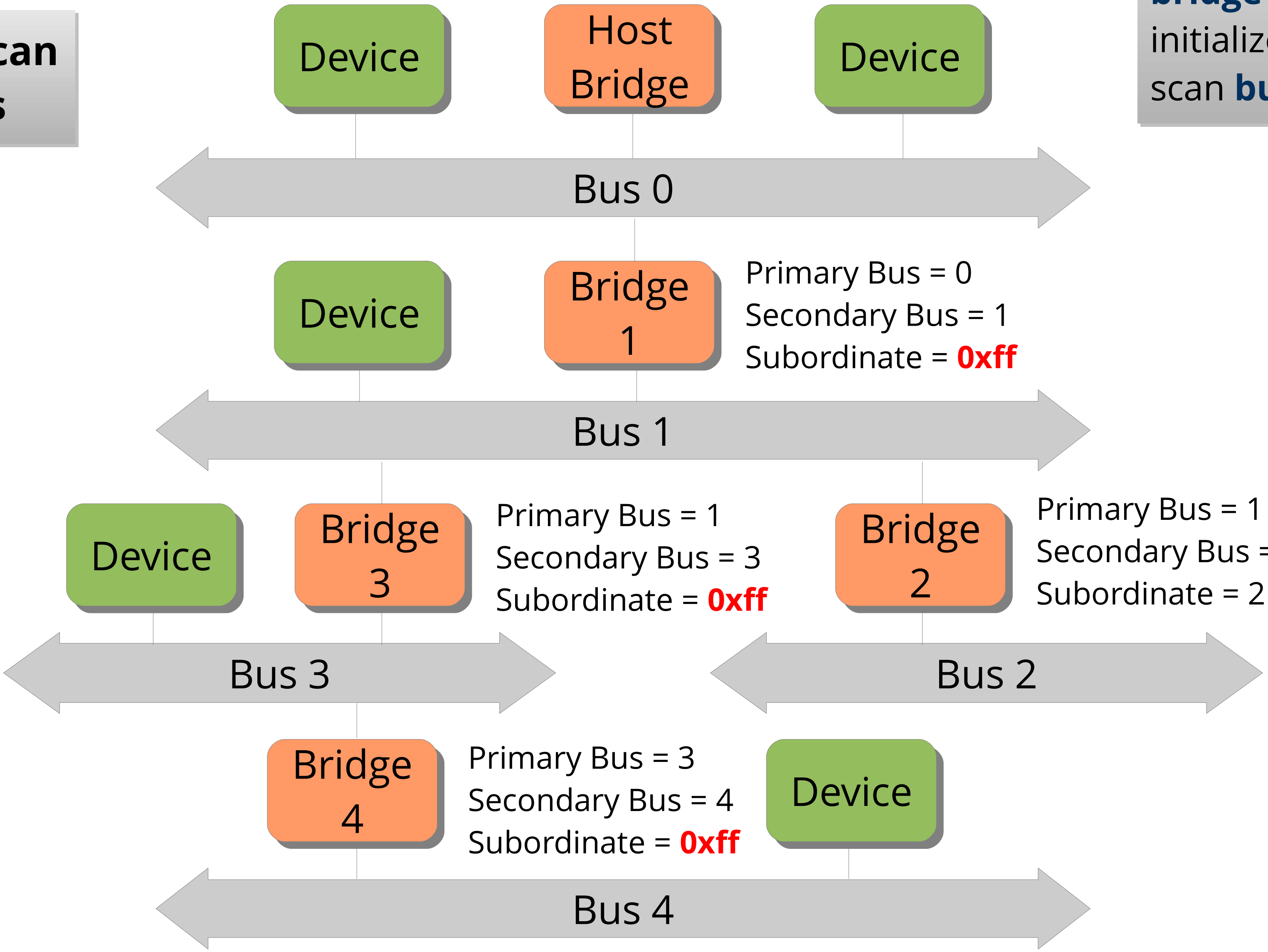
Step 1: Scan for buses



PCI Initialization in Linux

Step 1: Scan for buses

On **bus 3** there is another **bridge (4)**. We initialize it and scan **bus 4**.

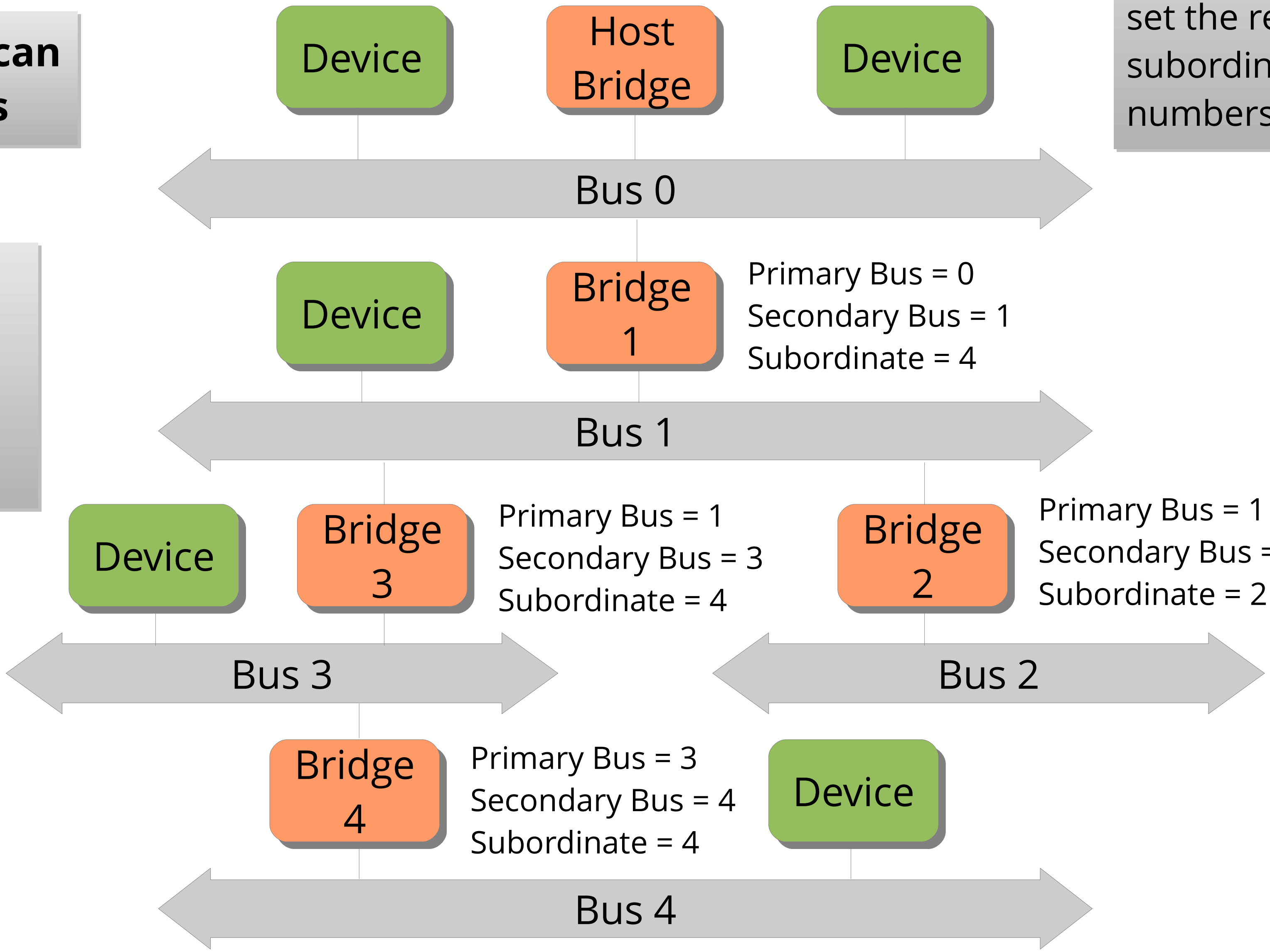


PCI Initialization in Linux

Step 1: Scan for buses

As no further buses exist, we can conclusively set the remaining subordinate numbers.

Finished! Now we can address the complete configuration space.



PCI Initialization in Linux

Step 2: Assign addresses

Algorithm:

- Align I/O and memory addresses of the bridge currently being configured to nearest 4 kiB respectively 1 MiB boundary
- For each device on the current bus (in ascending order of I/O address-space requests):
  - Reserve I/O and memory addresses
  - Update the global I/O and memory pointers
  - Initialize and activate the device
- Recursively apply the algorithm for all connected bridges
- Align resulting addresses (see above)
- Program and activate the bridge

The PCI BIOS – Overview

- Standardization by PCI-SIG (1993, draft by Intel 1991)
- Usually present on PCs, rarely on other computer types
- Configures PCI bridges and devices at boot time
  - minimally if a “Plug & Play” operating system is installed
  - otherwise completely
- After booting, the PCI BIOS allows …
  - searching for PCI devices by device class or type
  - accessing the configuration space
- Access via
  - BIOS interrupt 0x1a (Real Mode)
  - the “BIOS32 Service Directory” (Protected Mode)

The PCI BIOS – in Protected Mode

- The BIOS32 *Service Directory* (in principle) allows to access arbitrary BIOS components.
- It is located somewhere in the range 0xE0000–0xFFFFF:

Offset	**Size**	**Description**
0x00	4 Bytes	Signature "_32_"
0x04	4 Bytes	physical entry address (for call)
0x08	1 Byte	BIOS32 version (0)
0x09	1 Byte	Data-structure length / 16 (1)
0x0a	1 Byte	Checksum
0x0b	5 Bytes	reserved (0)

- With the BIOS32 service you can test whether a PCI BIOS is available.

The PCI BIOS – Functionality

- Specified PCI-BIOS functions:

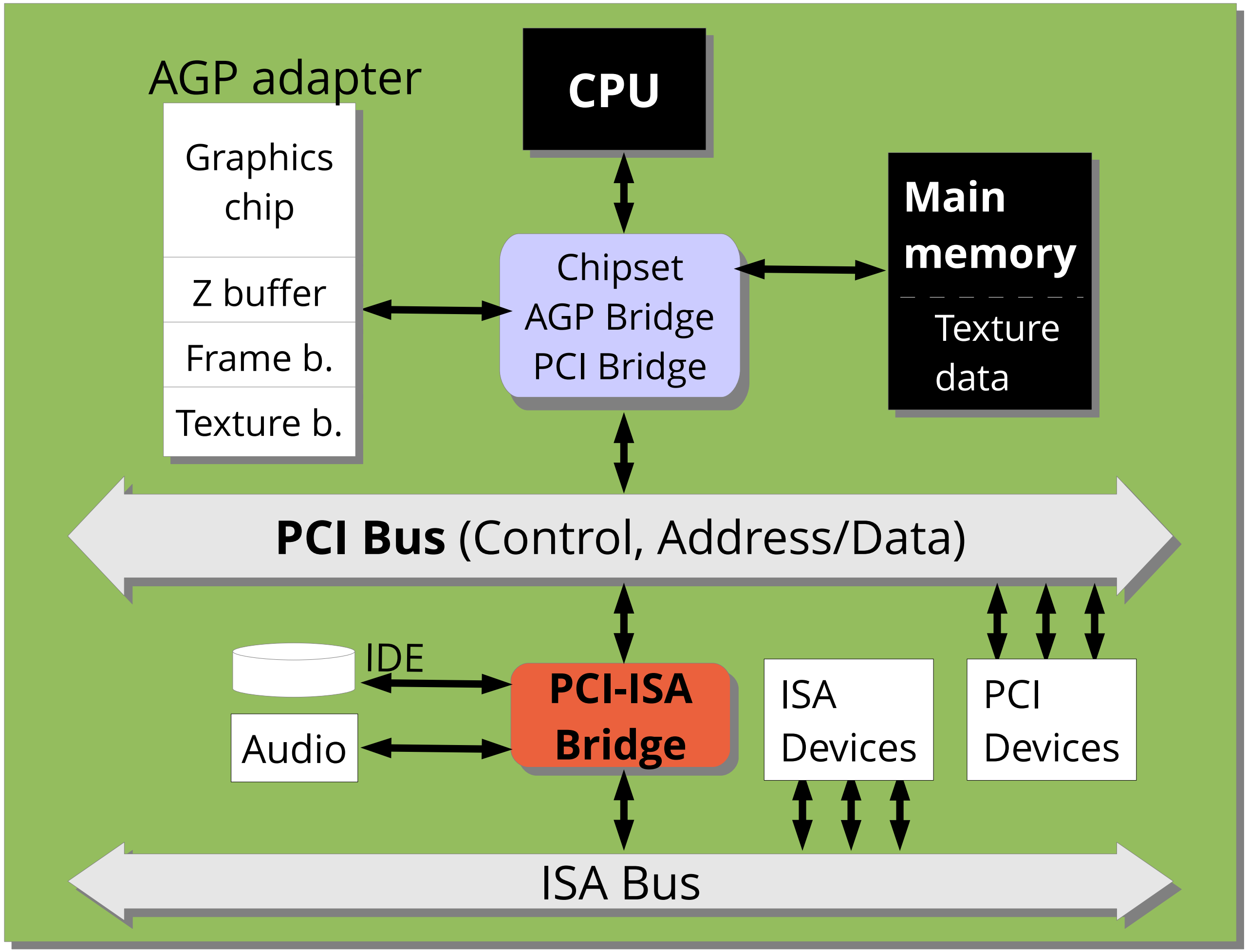
Function name	Arguments	Results
PCI BIOS Present	-	yes/no, last bus nr., init. mechanism
Find PCI Device	Device ID, Vendor ID, Index	Bus/Dev./Func. Nr.
Find PCI Class Code	Class Code, Index	Bus/Dev./Func. Nr.
Generate Special Cycle	Bus nr.	-
Get Interrupt Routing Opt.	Buffer memory	Routing possibilities
Set PCI Hardware Interrupt	Bus nr., Device nr., int. pin, int.-nr.	-
Read Configuration Byte/Word/DWord	Bus/Dev./Func./Reg. Nr.	read Byte/Word/DWord
Write Configuration Byte/Word/DWord	Bus/Dev./Func./Reg. Nr., Byte/Word/DWord to write	-

Agenda

- History
  - PC Bus Systems
- PCI Bus
- PCI from an Operating-System Perspective
  - Initialization, PCI BIOS, …
- **PCI Extensions and Successors**
  - AGP
  - PCI-X
  - PCI Express
  - HyperTransport
- Summary

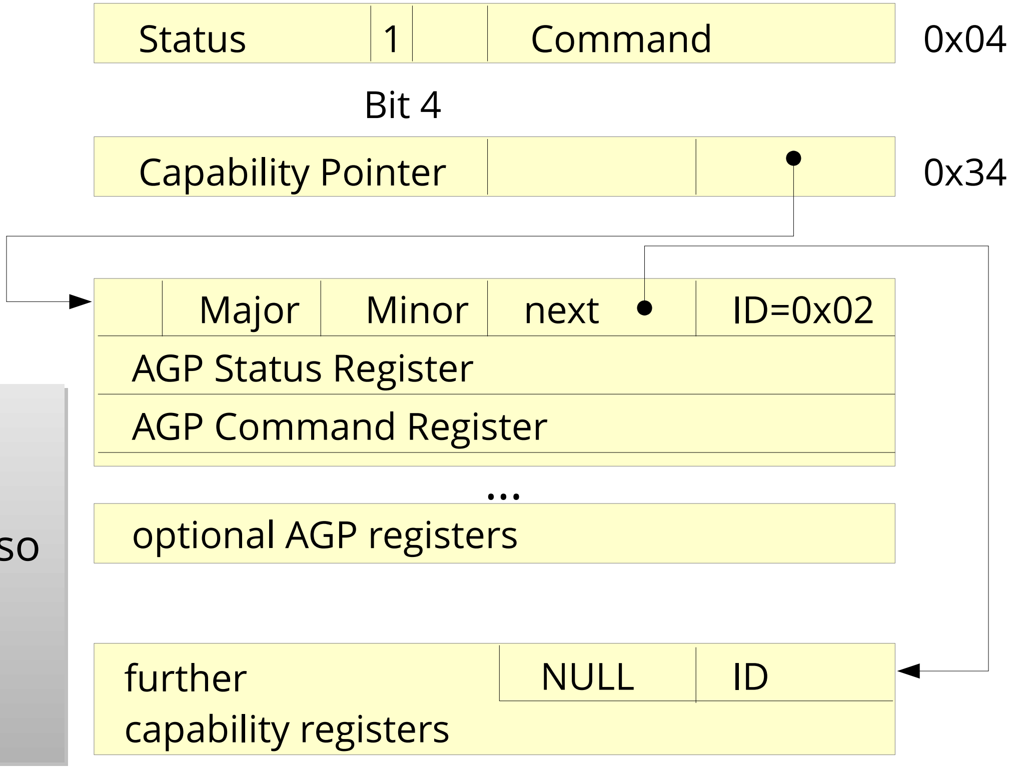
Accelerated Graphics Port – Hardware

- AGP, 1997: fast 1:1 connection of **one** (3D) graphics adapter
  - (theoretically) N x 266 MB/s transfer rate for AGP 1x, 2x, 4x, …



AGP – Initialization

- AGP adapter and bridge present themselves to the system like a PCI-to-PCI bridge and a normal PCI device
  - full software compatibility
- Special AGP registers can be accessed through the **capability list** in the configuration space:



Status | 1 | Command — 0x04

Bit 4

Capability Pointer — 0x34

Major | Minor | next | ID=0x02

AGP Status Register

AGP Command Register

...

optional AGP registers

further capability registers | NULL | ID

Via AGP status and command registers we can primarily determine the AGP version and timing parameters.

Aside from AGP extensions, the capability list is also e.g. used for PCI power management.

The optional AGP registers unfortunately have device-specific meaning.

PCI-X (eXtended)

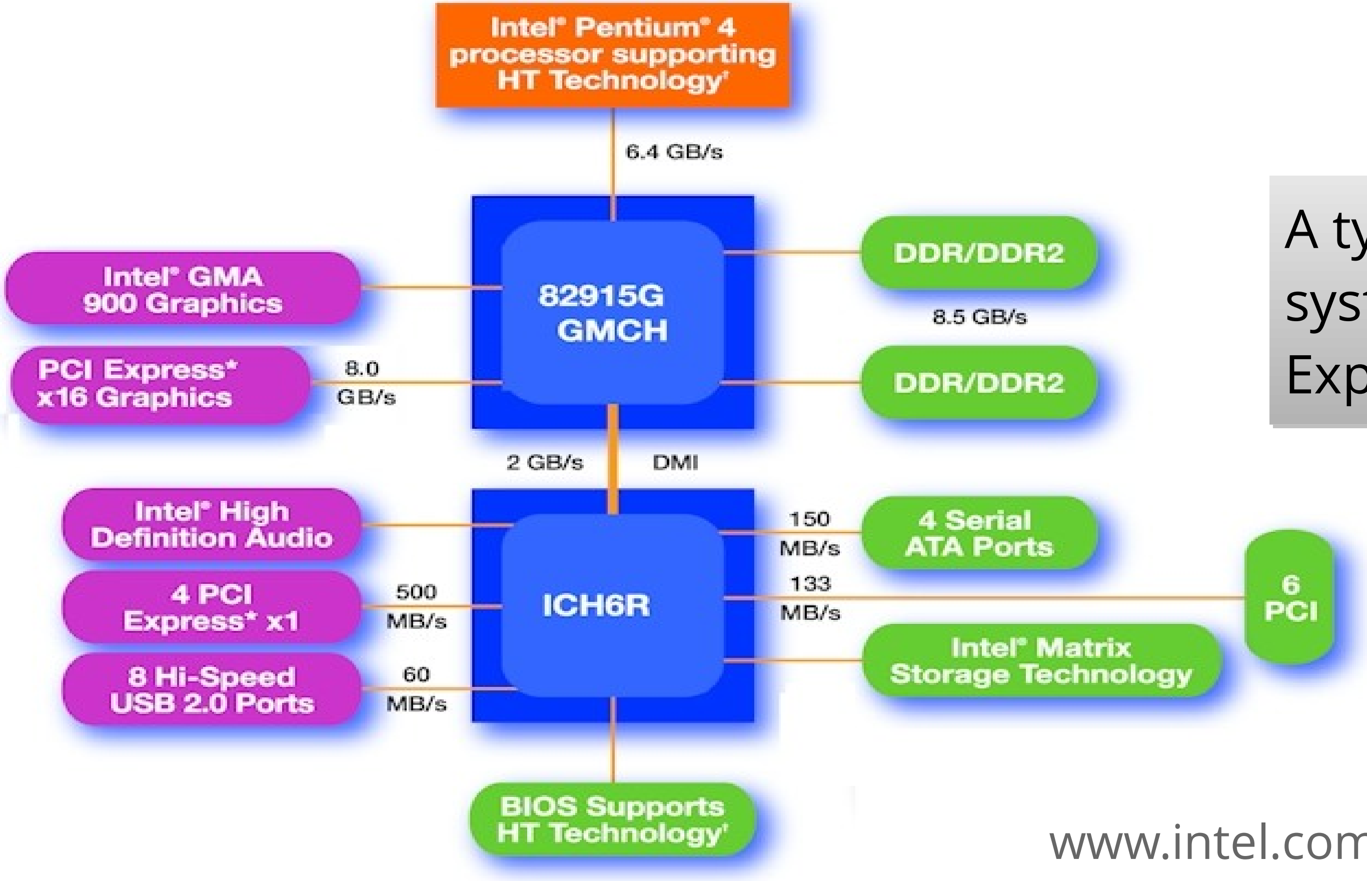
- PCI-bus extension (1999)
  - defined by the ***PCI Special Interest Group*** (SIG) in the PCI 3.0 standard
- Allows for higher bandwidth at full compatibility
  - The PCI-X bus uses the configuration of the **slowest** device

PCI adapter type		**PCI (conventional)**			**PCI-X**	
Bus frequency		33 MHz	33 MHz	66 MHz	66 MHz	133 MHz
Voltage		5 V	3.3 V/univ.	3.3 V/univ.	3.3 V/univ.	3.3 V/univ.
Mainboard						
PCI	33 MHz	33 MHz	33 MHz	33 MHz	33 MHz	33 MHz
PCI	66 MHz	-	33 MHz	66 MHz	33/66 MHz	33/66 MHz
PCI-X	66 MHz	-	33 MHz	33/66 MHz	66 MHz	66 MHz
PCI-X	100 MHz	-	33 MHz	33/66 MHz	66 MHz	100 MHz
PCI-X	133 MHz	-	33 MHz	33/66 MHz	66 MHz	133 MHz

- Besides the increased clock frequency: ***Split Transactions***
  - accessible again via the **capabilities list**

PCI Express

- … technically has little to do with the original PCI bus
- Bi-directional, serial, point-to-point connections
  - Bandwidth per **lane** per direction: 512 MB/s, 8GB/s at x16

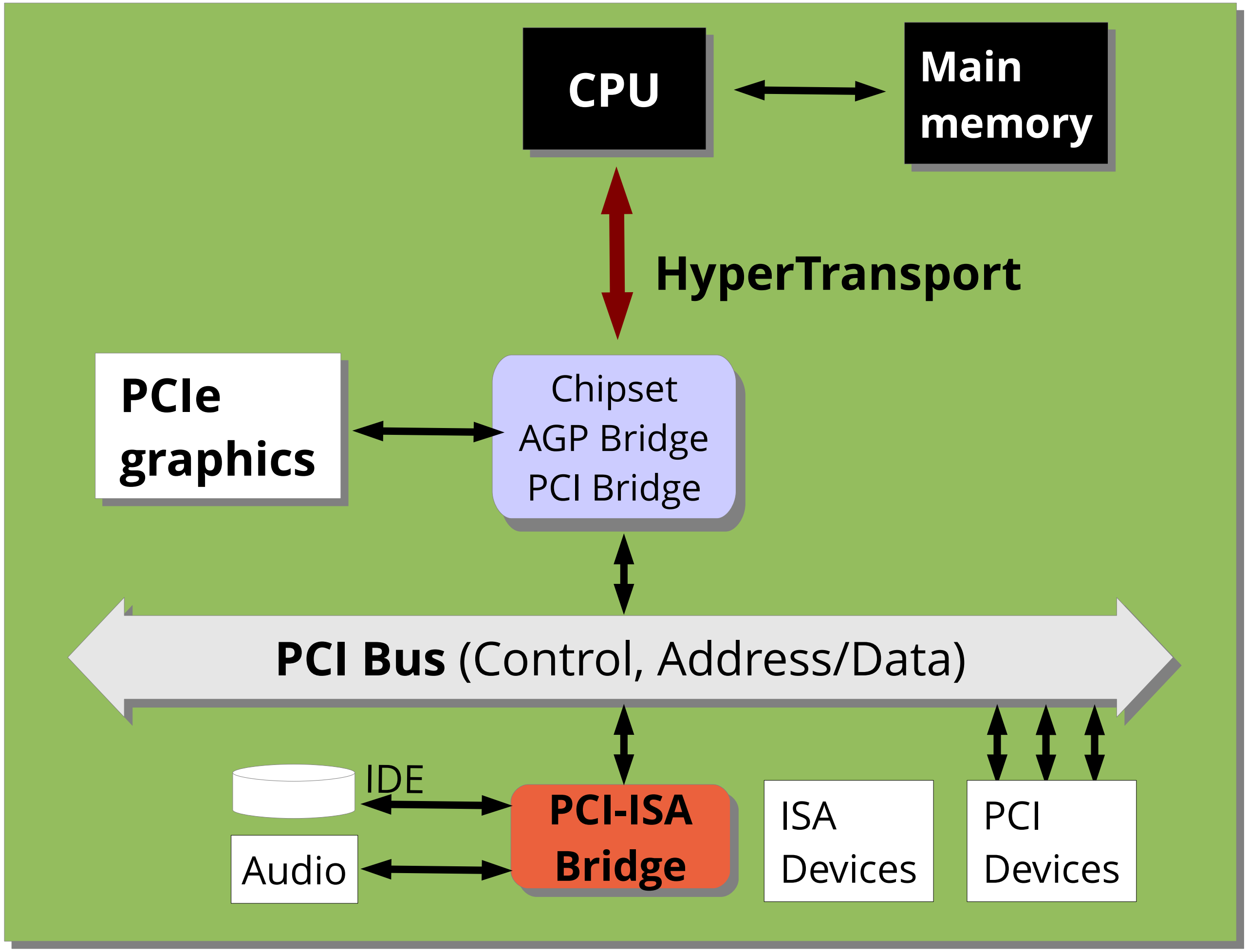


A typical PC system with PCI-Express devices

www.intel.com

HyperTransport

- (AMD) CPU integrates memory controller and L2 cache
- **Standardized communication** with North Bridge



HyperTransport

- Standardized in several versions 2001–2008
  - Consortium: AMD, Apple, Cisco, NVIDIA, Sun, …
- Bi-directional, point-to-point, links with 2–32 bits, clocked up to 3.2 GHz (DDR)
- Depending on version and configuration up to 12.8 GB/s (uni-dir.)
  - Successor: AMD **Infinity Fabric** (HyperTransport “superset”), up to 72 GB/s
- Device configuration like with PCI
- Further uses aside from FSB replacement
  - Inter-CPU communication
  - Chipset communication (Northbridge ⇔ Southbridge)
  - Communication with co-processors: HTX
- Intel pendants: ***QuickPath Interconnect*** (QPI, since 2008, 12.8 GB/s) and ***Ultra Path Interconnect*** (UPI, 2017, 20.8 GB/s)

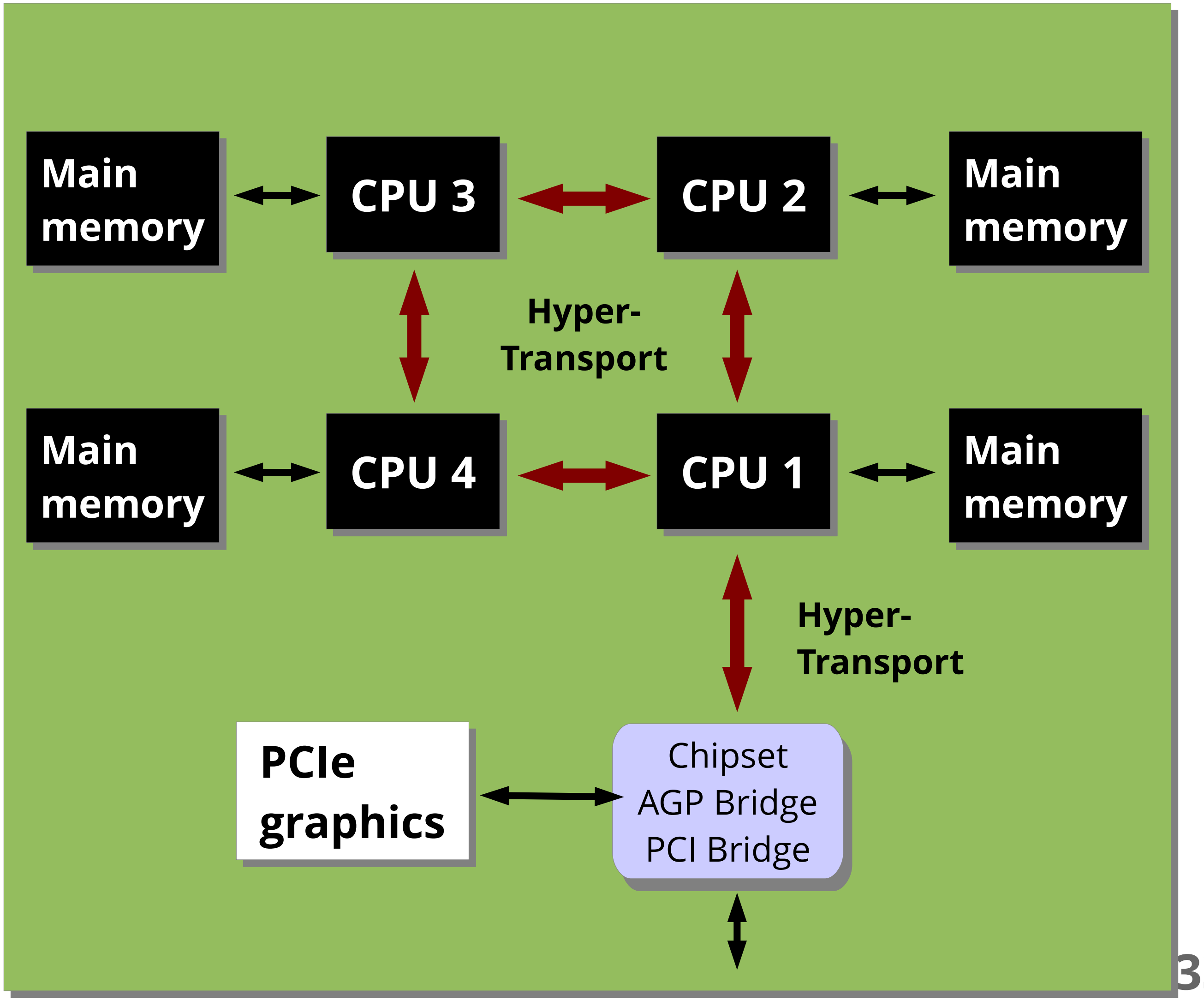
HyperTransport in MP Systems

- **NUMA** (Non-Uniform Memory Architecture)

CPUs (with multiple cores each) communicate via HyperTransport.

Global address space: Main memory attached to other CPUs can be addressed, but with higher latency.

The operating system must distribute tasks accordingly.



Agenda

- History
  - PC Bus Systems
- PCI Bus
- PCI from an Operating-System Perspective
  - Initialization, PCI BIOS, …
- PCI Extensions and Successors
  - AGP
  - PCI-X
  - PCI Express
  - HyperTransport
- **Summary**

Summary

- PCI has been dominating PC bus systems for decades
- Newer developments (PCI Express) hardly share similarities with the PCI bus from 1991
  - serial point-to-point connections and **switches**
- Aside from physical properties, PCI also defines a programming model
  - I/O and memory address spaces
  - Configuration and initialization via configuration space
  - Bus hierarchies
- Even the newest developments are compatible to PCI **on the programming-model level**