Trigger-less DAQ
Part (2)

Umberto Marconi

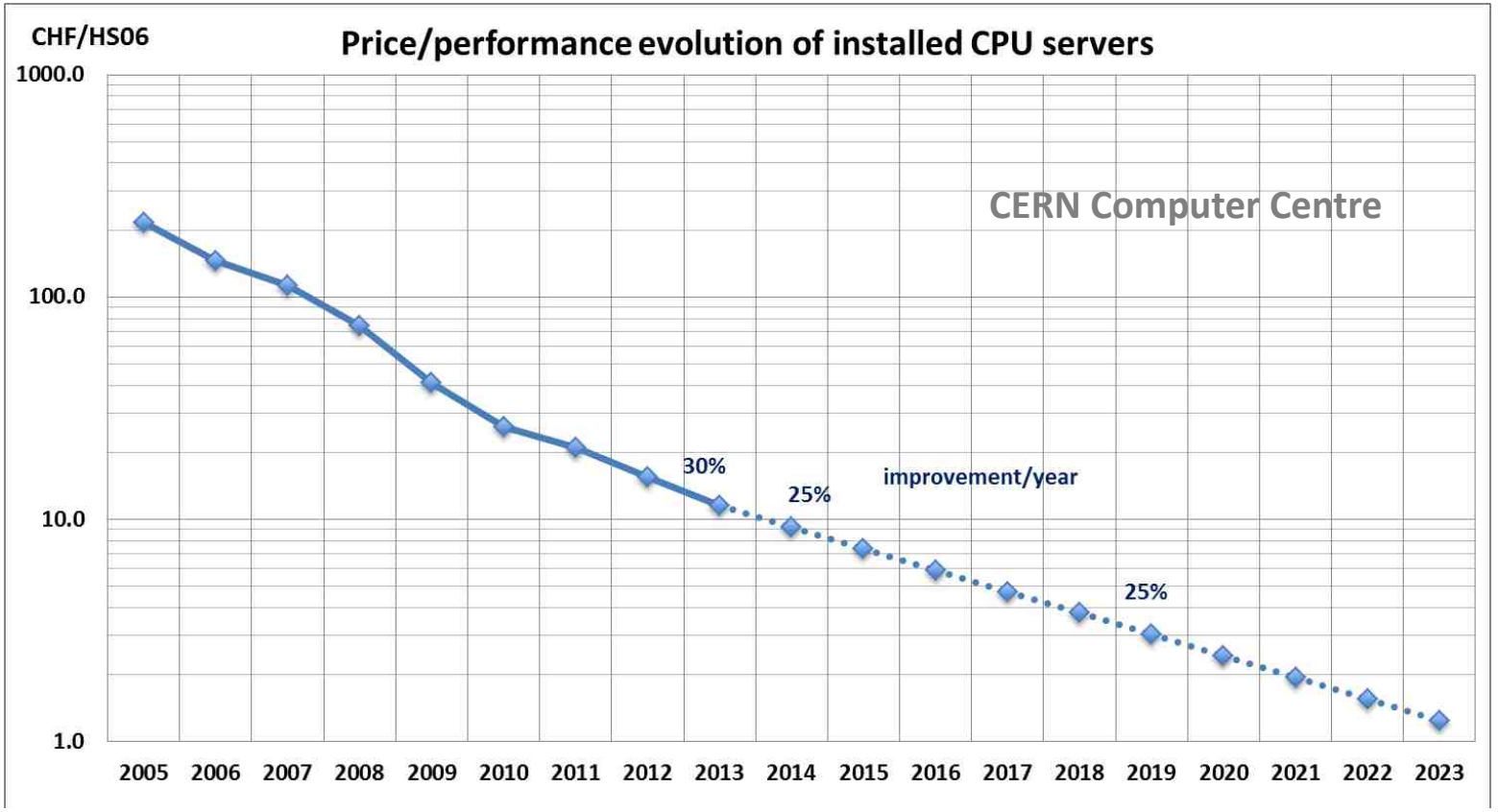
INFN Bologna

Catania 2014

INFN
Istituto Nazionale
di Fisica Nucleare

Computers

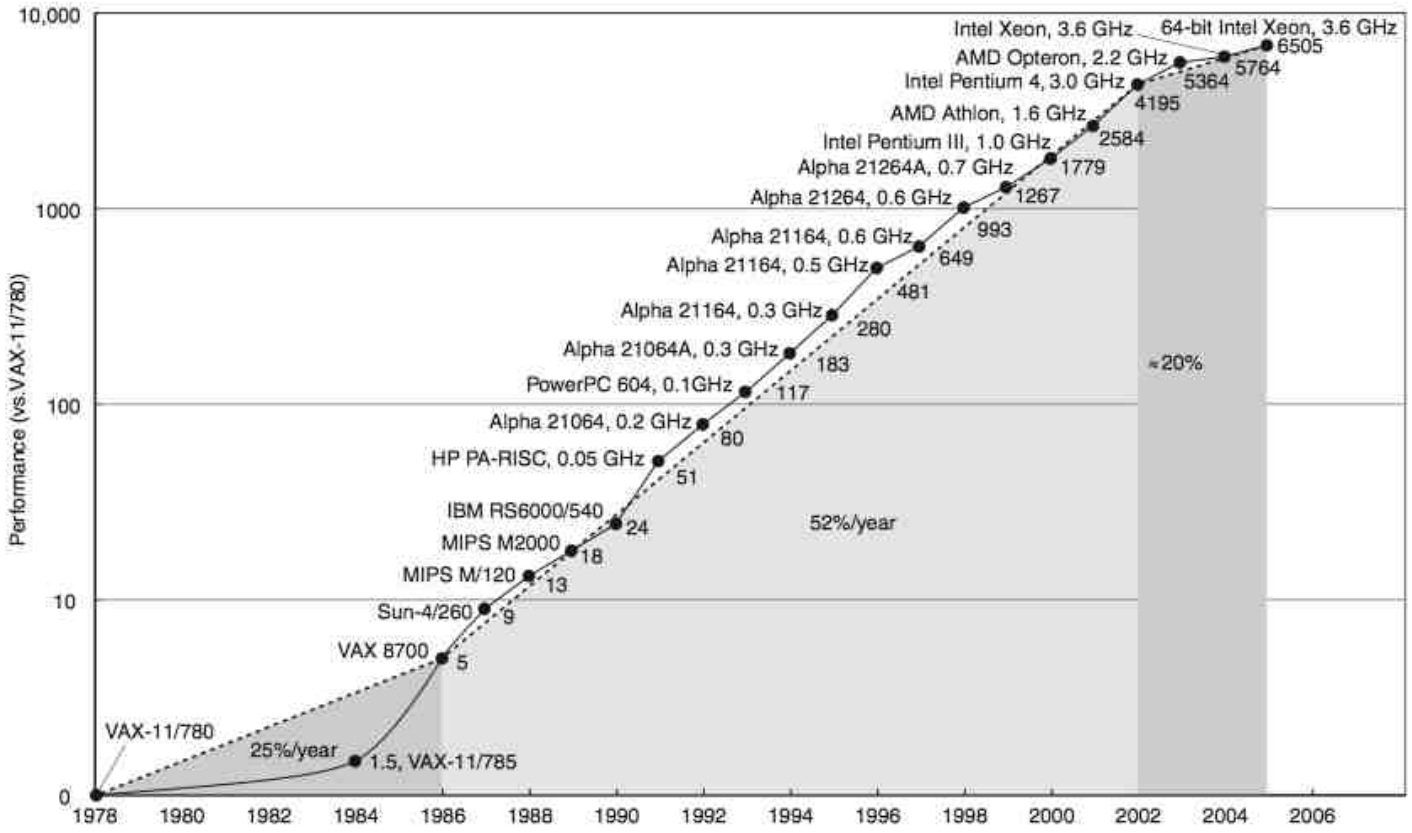
Server price/performance vs time



HS06 is a benchmark proposed by WLCG to measure the performance of an HEP worker node.

- Price/performance improvements are achieved via:
  - Technology and fabrication of smaller structure sizes;
  - Longer amortization periods of semiconductor fabrication plants for the same chip generation.

History of Processor Performance



Performance relative to the VAX 11/780 as measured by the SPECint benchmarks

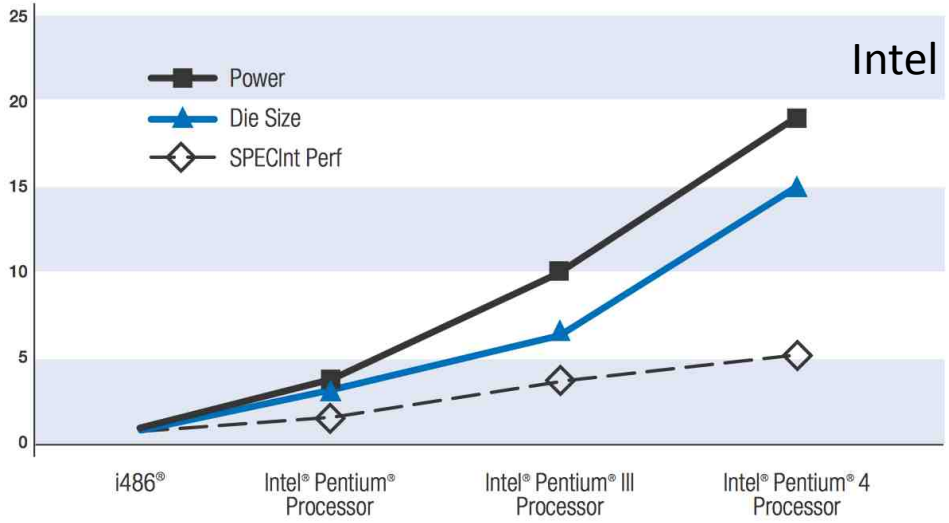
- Since 2002, the limits of power, availability of instruction-level parallelism and memory latency have slowed down **single processor** performance to about 20% per year.

Processor performance

Computing execution time = Instr. count × **CPI** × Clock Period

CPI: average number of clock cycles per instruction

- Efforts to improve system performance on single processor systems have focused on processing more instructions faster through **higher clock speeds**, **instruction-level parallelism** (ILP) and **caches** (in order to overcome the memory wall).
  - ILP refers to techniques to increase the number of instructions executed each clock cycle. For example, super-scalar processor have multiple execution units that can process several instructions each clock cycle.



Die size and transistors

Intel 386 to Pentium 4

Intel 386: 275K transistors (die size = 43mm^2)

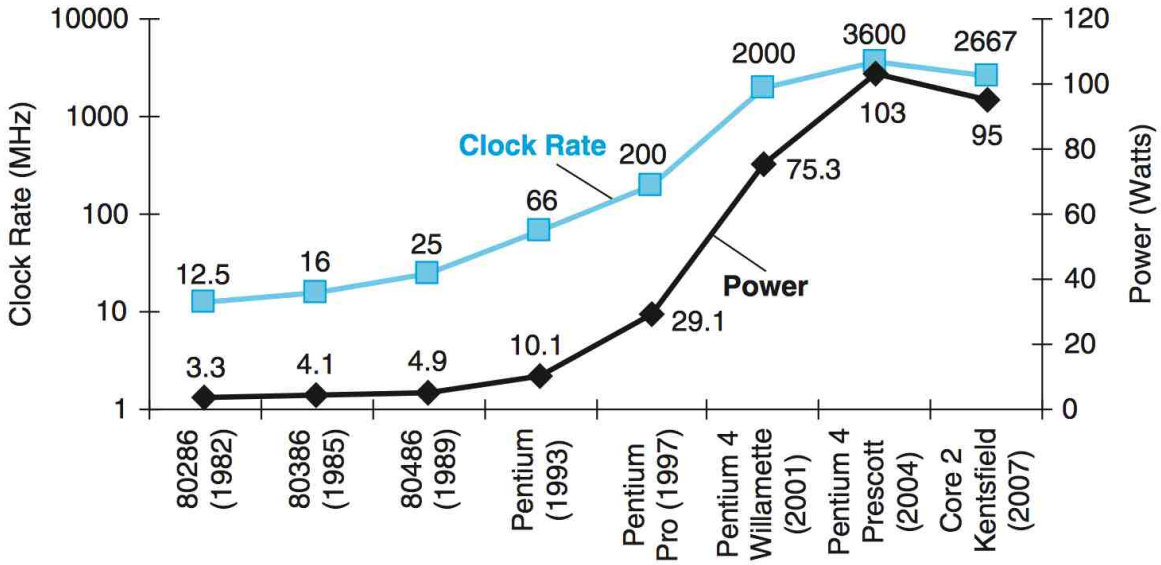
Pentium 4: 42M transistors (die size = 217mm^2)

Devices on chip:

- 5X from increased die size;
- 30X from technology scaling.

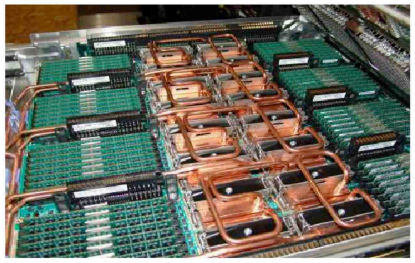
The power wall

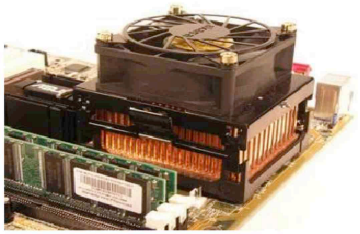
The Power Wall means faster computers get really hot.



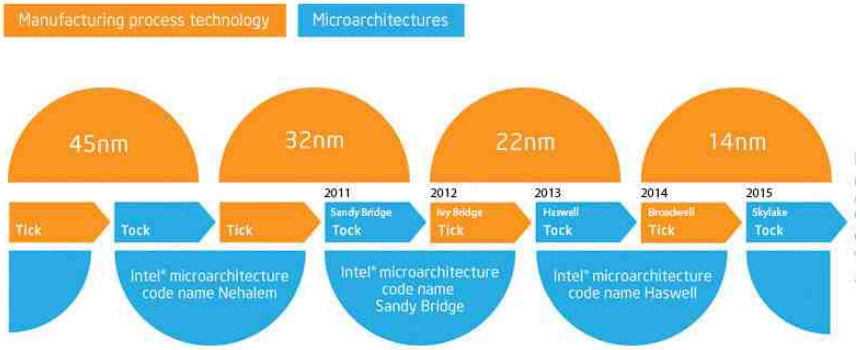
The power dissipation of the CPU chip reached ~100 W/cm^2 that is beyond the capacity of inexpensive cooling techniques.

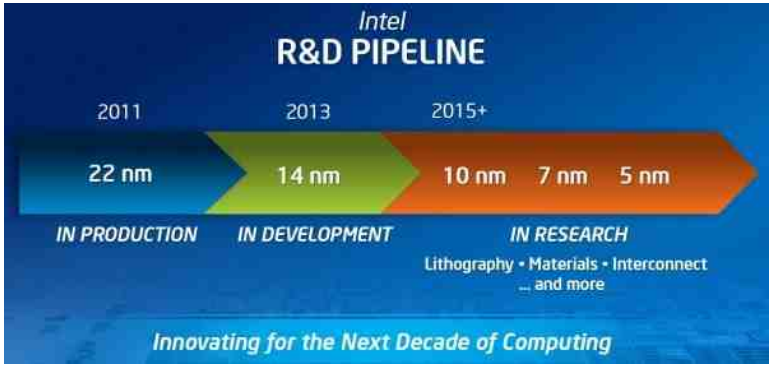
Copper heat sink





CPUs (towards many-cores)





Intel® Xeon® Processor E5-2600 v3 Product Family
22nm Haswell

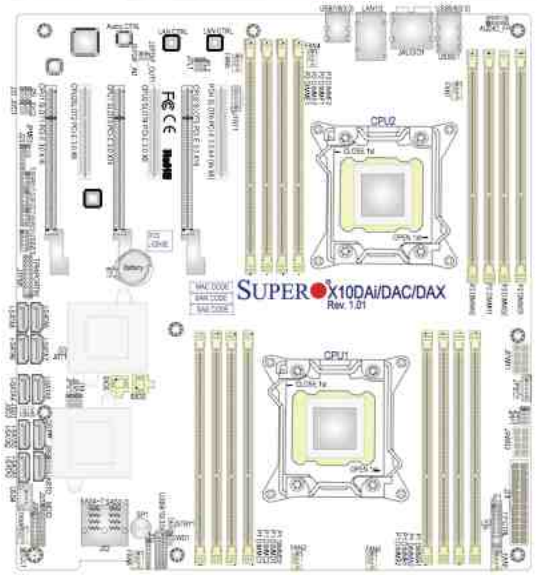
Nome prodotto	Stato	Data di lancio	Numero di core	TDP massimo	Prezzo consigliato per il cliente
Intel® Xeon® Processor E5-2683 v3 (35M Cache, 2.00 GHz)	Launched	Q3'14	14	120 W	TRAY: $1846.00
Intel® Xeon® Processor E5-2695 v3 (35M Cache, 2.30 GHz)	Launched	Q3'14	14	120 W	TRAY: $2424.00 BOX : $2428.00
Intel® Xeon® Processor E5-2697 v3 (35M Cache, 2.60 GHz)	Launched	Q3'14	14	145 W	TRAY: $2702.00 BOX : $2706.00
Intel® Xeon® Processor E5-2698 v3 (40M Cache, 2.30 GHz)	Announced	Q3'14	16	135 W	N/A
Intel® Xeon® Processor E5-2699 v3 (45M Cache, 2.30 GHz)	Announced	Q3'14	18	145 W	N/A
Intel® Xeon® Processor E5-2628L v3 (25M Cache, 2.00 GHz)	Launched	Q3'14	10	75 W	TRAY: $1364.00
Intel® Xeon® Processor E5-2650 v3 (25M Cache, 2.30 GHz)	Launched	Q3'14	10	105 W	TRAY: $1166.00 BOX : $1171.00
Intel® Xeon® Processor E5-2660 v3 (25M Cache, 2.60 GHz)	Launched	Q3'14	10	105 W	TRAY: $1445.00 BOX : $1449.00
Intel® Xeon® Processor E5-2670 v3 (30M Cache, 2.30 GHz)	Launched	Q3'14	12	120 W	TRAY: $1589.00 BOX : $1593.00
Intel® Xeon® Processor E5-2690 v3 (30M Cache, 2.60 GHz)	Launched	Q3'14	12	135 W	BOX : $2094.00 TRAY: $2090.00

Intel® Xeon® Processor E5-2600 Product Family
32nm Sandy Bridge

Nome prodotto	Stato	Data di lancio	Numero di core	TDP massimo	Prezzo consigliato per il cliente
Intel® Xeon® Processor E5-2648L (20M, 1.80 GHz, 8.0 GT/s Intel® QPI)	Launched	Q1'12	8	70 W	TRAY: $1346.00
Intel® Xeon® Processor E5-2658 (20M, 2.10 GHz, 8.0 GT/s Intel® QPI)	Launched	Q1'12	8	95 W	TRAY: $1479.00
Intel® Xeon® Processor E5-2687W (20M Cache, 3.10 GHz, 8.00 GT/s Intel® QPI)	Launched	Q1'12	8	150 W	BOX : $1890.00 TRAY: $1885.00
Intel® Xeon® Processor E5-2680 (20M Cache, 2.70 GHz, 8.00 GT/s Intel® QPI)	Launched	Q1'12	8	130 W	BOX : $1727.00 TRAY: $1723.00
Intel® Xeon® Processor E5-2660 (20M Cache, 2.20 GHz, 8.00 GT/s Intel® QPI)	Launched	Q1'12	8	95 W	BOX : $1333.00 TRAY: $1329.00
Intel® Xeon® Processor E5-2650L (20M Cache, 1.80 GHz, 8.00 GT/s Intel® QPI)	Launched	Q1'12	8	70 W	TRAY: $1107.00
Intel® Xeon® Processor E5-2630L (15M Cache, 2.00 GHz, 7.20 GT/s Intel® QPI)	Launched	Q1'12	6	60 W	TRAY: $662.00
Intel® Xeon® Processor E5-2643 (10M Cache, 3.30 GHz, 8.00 GT/s Intel® QPI)	Launched	Q1'12	4	130 W	TRAY: $885.00
Intel® Xeon® Processor E5-2609 (10M Cache, 2.40 GHz, 6.40 GT/s Intel® QPI)	Launched	Q1'12	4	80 W	BOX : $299.00 TRAY: $294.00
Intel® Xeon® Processor E5-2667 (15M Cache, 2.90 GHz, 8.00 GT/s Intel® QPI)	Launched	Q1'12	6	130 W	TRAY: $1552.00

Motherboards

Dual Socket



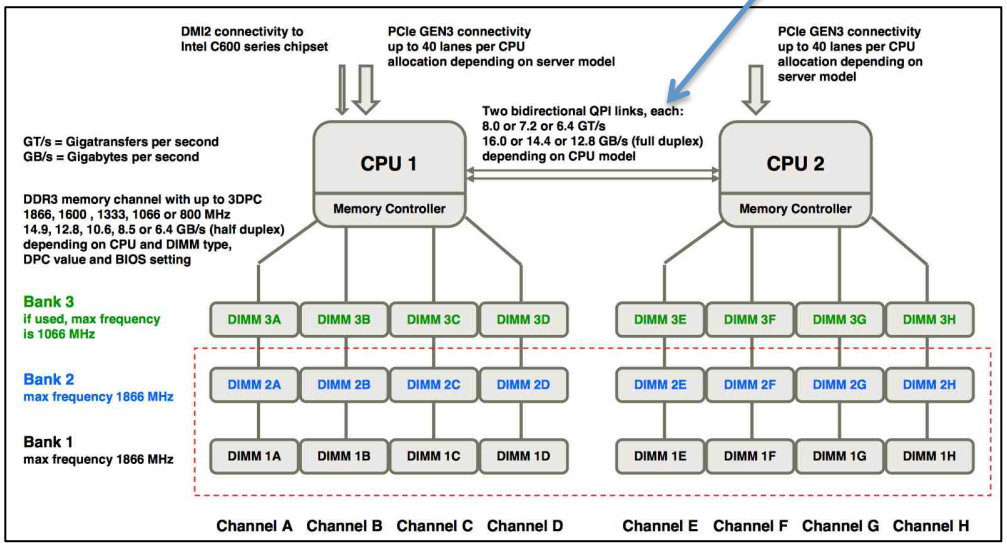
Key Features

1. Dual socket R3 (LGA 2011) supports Intel® Xeon® processor E5-2600 v3 family; QPI up to 9.6GT/s
2. Intel® C612 chipset
3. Up to 1TB reg. ECC DDR4 2133MHz; 16x DIMM sockets
4. 3x PCI-E 3.0 x16, 2x PCI-E 3.0 x8, and 1x PCI-E 2.0 x4 (in x8) slot
5. Intel® i210 Dual port GbE LAN
6. 10x SATA3 (6Gbps) ports w/ Intel C612 controller; RAID 0, 1, 5, 10
7. 6x USB 3.0 (4 rear, 2 via header) 5x USB 2.0 (2 rear, 2 via header, 1 Type A)
8. 7.1 HD Audio with optical S/PDIF
9. Thunderbolt Add-On Card support

Architecture

NUMA Non Uniform Memory Access

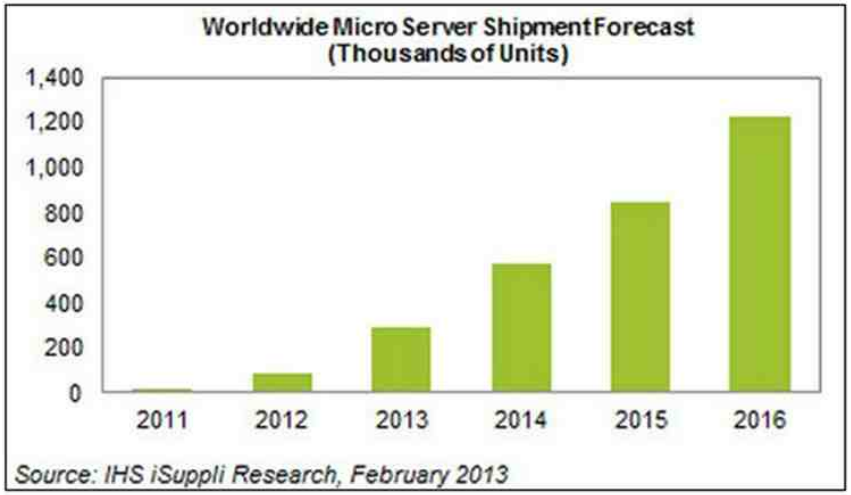
QPI



RAM

Standard name	Memory clock (MHz)	Cycle time[4] (ns)	I/O bus clock (MHz)	Data rate (MT/s)	V_{DDQ} (V)	Module name	Peak transfer rate (MB/s)	Timings (CL-tRCD-tRP)
DDR-200	100	10	100	200	2.5±0.2	PC-1600	1600	
DDR-266	133⅓	7.5	133⅓	266⅔		PC-2100	2133⅓	2.5-3-3
DDR-333	166⅔	6	166⅔	333⅓		PC-2700	2666⅔	
DDR-400A DDR-400B DDR-400C	200	5	200	400	2.6±0.1	PC-3200	3200	2.5-3-3 3-3-3 3-4-4

Servers



Microserver ships are expected to grow by a factor of 50 between 2011 and 2016

INFN Istituto Nazionale di Fisica Nucleare

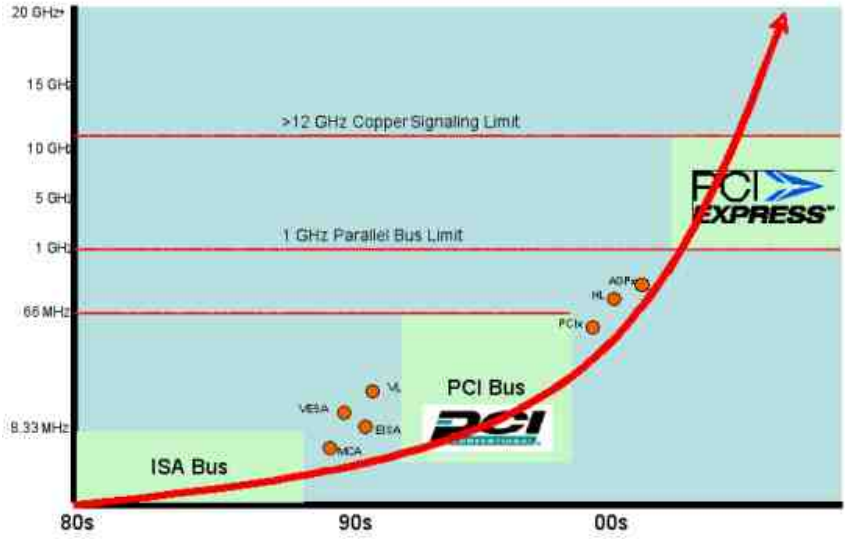
PCI Express

A local bus for chip-to-chip interconnects.

Disclaimer: difficult to cover all cases, this talk is a partial view.

PC buses evolution

- PCI 33 MHz was released in 1993, PCI-X (256 and 533 MHz) in 2002



Bus Type	Clock Frequency	Peak Bandwidth[*]
PCI 32-bit	33 MHz	133 MBytes/sec
PCI 32-bit	66 MHz	266 MBytes/sec
PCI-X 32-bit	66 MHz	266 MBytes/sec
PCI-X 32-bit	133 MHz	533 MBytes/sec
PCI-X 32-bit	266 MHz effective	1066 MBytes/sec
PCI-X 32-bit	533 MHz effective	2131 MByte/sec

- The date of release of PCIe Generation 1 is Q2, 2002

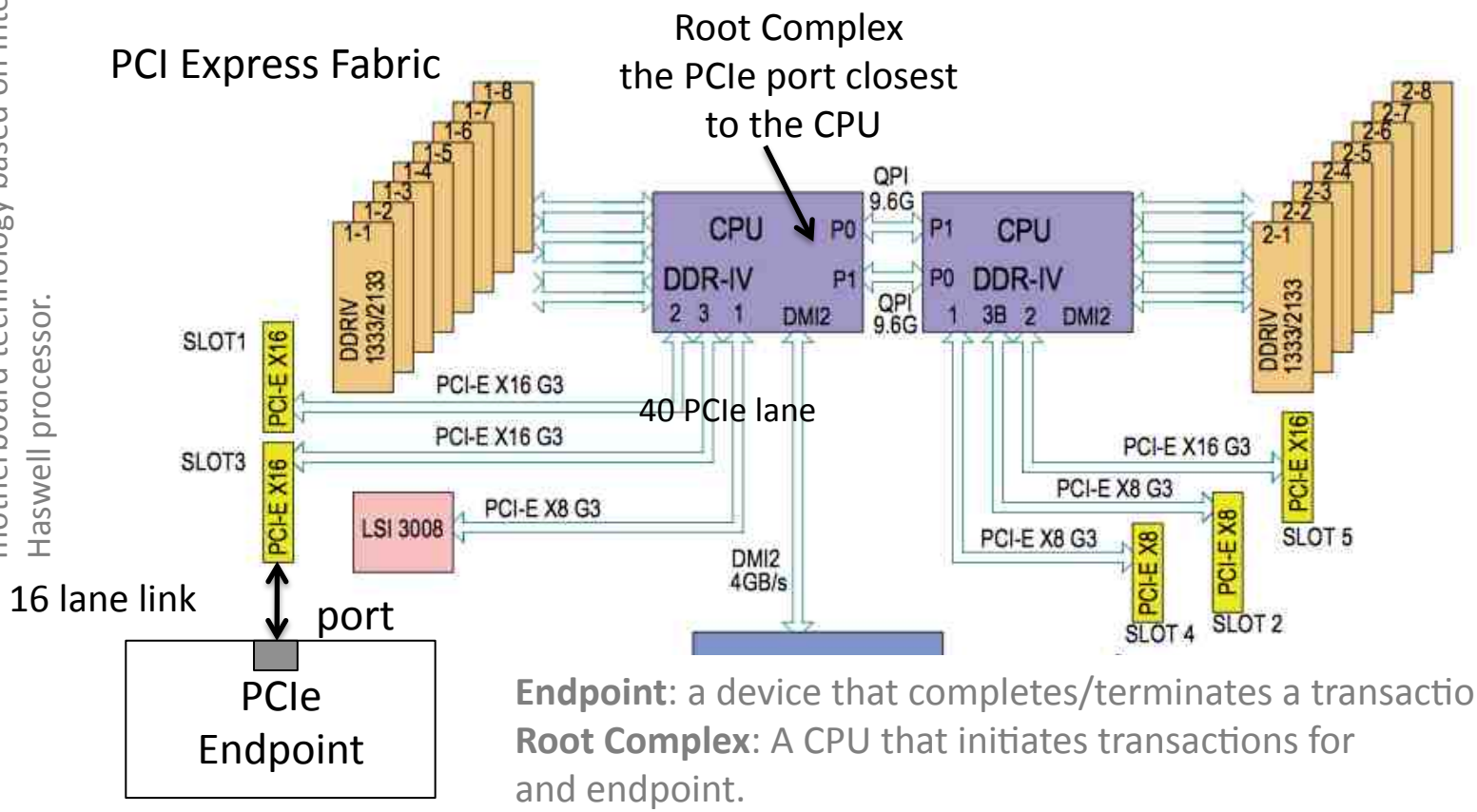
PCI Express link performance[21][22]

PCI Express version	Line code	Per lane			In a ×16 slot (16-lane)	
		Raw bit rate[a]	Bandwidth[a]		Raw bit rate[a]	Bandwidth[a]
1.0	8b/10b	2.5 GT/s	2 Gbit/s	250 MB/s	40 GT/s	4 GB/s
2.0	8b/10b	5 GT/s	4 Gbit/s	500 MB/s	80 GT/s	8 GB/s
3.0	128b/130b	8 GT/s	7.877 Gbit/s	984.6 MB/s	128 GT/s	15.754 GB/s
4.0	128b/130b	16 GT/s	15.754 Gbit/s	1969.2 MB/s	256 GT/s	31.508 GB/s

a. In each direction.

PCIe Terminology

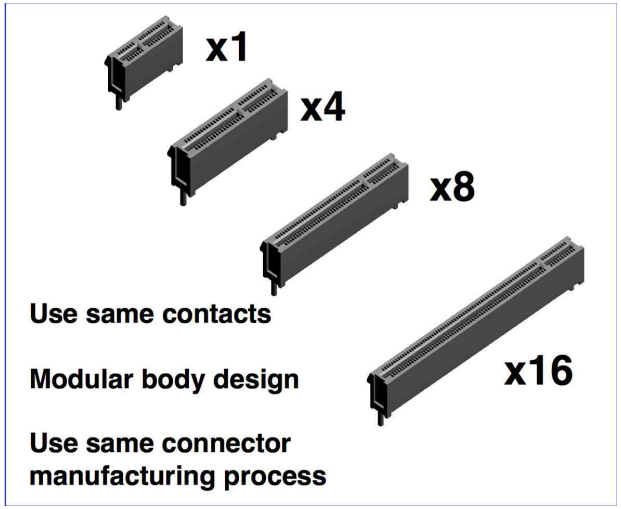
PCIe is a computer expansion bus for peripheral connection.

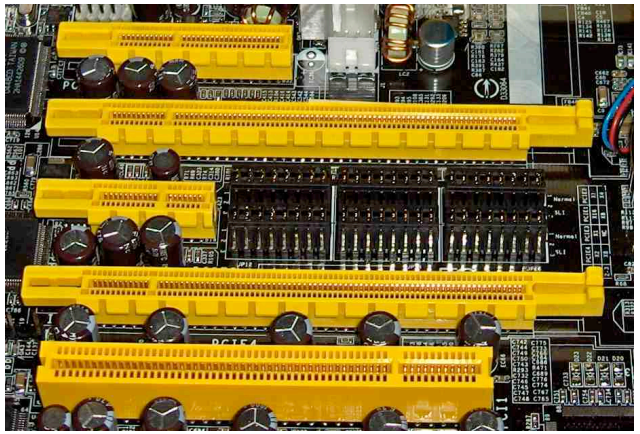


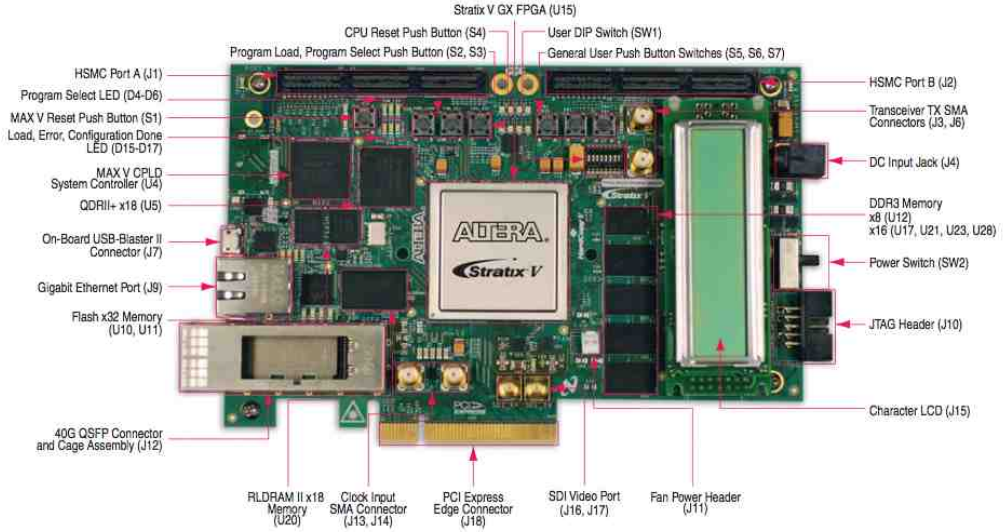
Computer motherboard

Endpoint: a device that completes/terminates a transaction.
Root Complex: A CPU that initiates transactions for and endpoint.
Port: A physical interface between two devices.
Lane: Two differential pares for Tx and Rx.

PCIe slots and cards





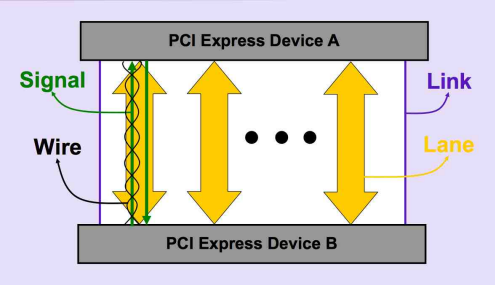


Smaller link-width cards can plug into larger link-width connectors

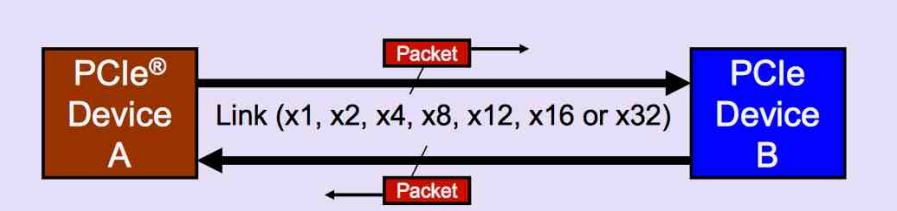
PCIe Terminology (2)

A PCIe link supports **full-duplex** communication between any two endpoints.

Point-to-point connection



Packet based transaction protocol



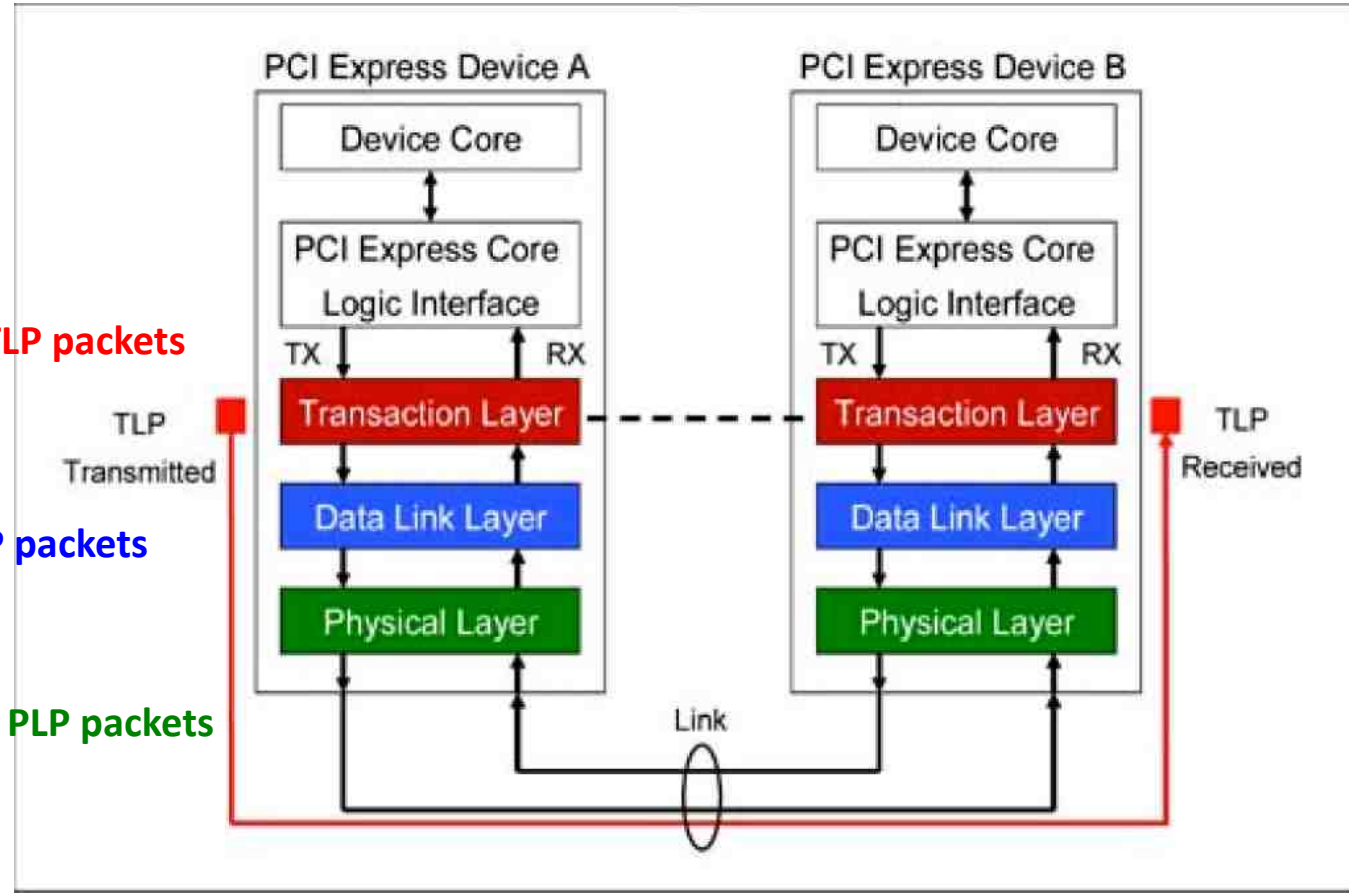
- PCIe communication is encapsulated in packets.
- PCI Express uses a **split-transaction protocol**: there are two transaction phases: the **request** and the **completion**.
  - The **requester**, sends out the request packet.
    The target of the request, is referred to as the **completer**.
  - For requests that require completions, the completer later sends back a completion packet (or packets) to the requester.

PCIe layered protocol

Transaction Layer → TLP packets

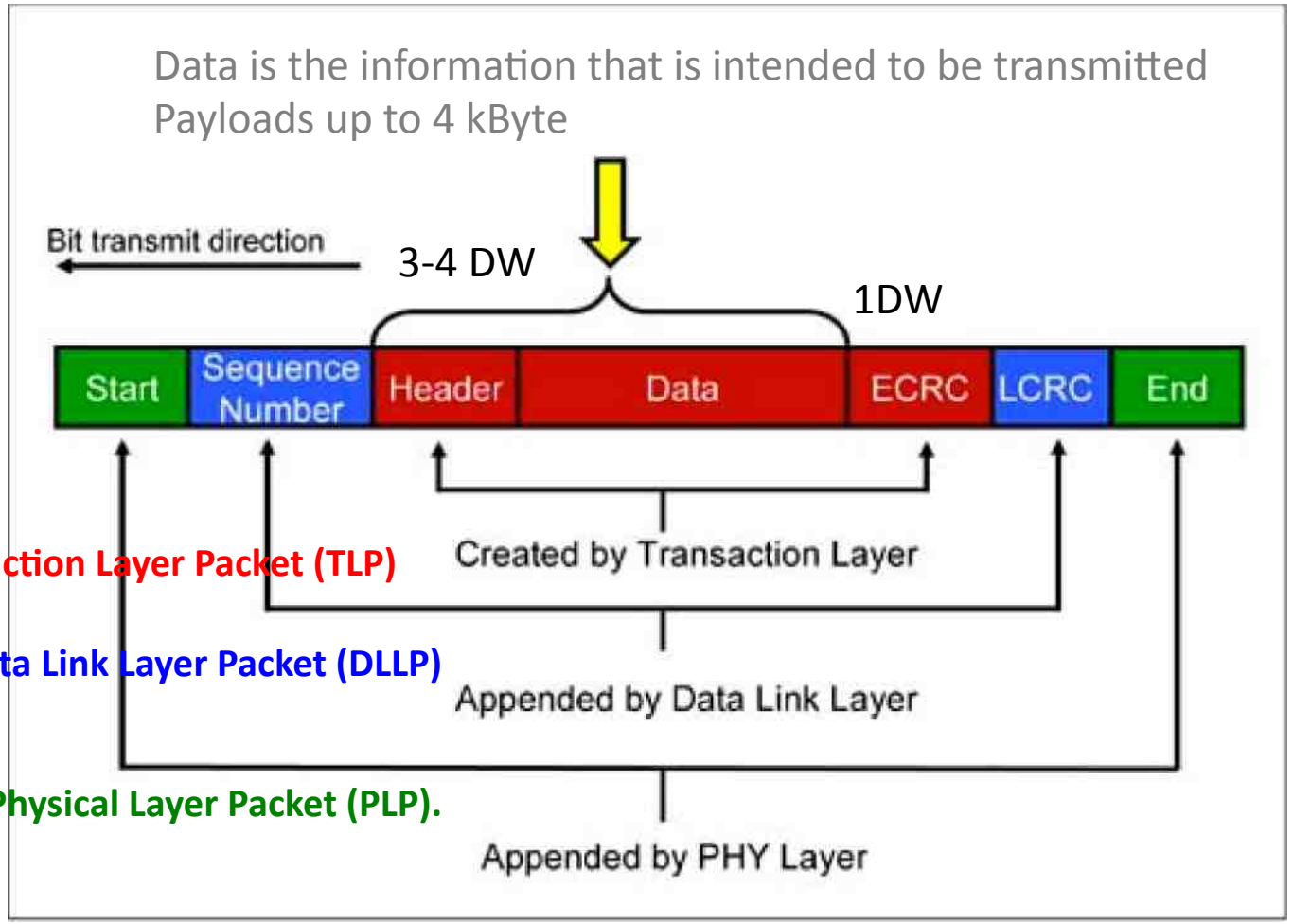
Data Link Layer → DLLP packets

Physical Layer Packet → PLP packets



- Each layer support specific responsibility and function.
- Each layer produces its own type of packet.
- Packets generated at a level only progress through the same layer at the receiving side.

TLP assembly



INFN Istituto Nazionale di Fisica Nucleare

PCI Express Transactions

- **Transactions** are defined as a series of one or more **TLP** packet transmissions, required to complete an information transfer between a requester and a completer.
- For **Non-posted** transactions, a requester transmits a TLP request packet to a completer. At a later time, **the completer returns a TLP completion packet** back to the requester. The purpose of the completion TLP is to confirm to the requester that the completer has received the request TLP.
- For **Posted** transactions, a requester transmits a TLP request packet to a completer. The completer however does **NOT return a completion** TLP back to the requester. **Posted transactions are optimized for best performance** in completing the transaction at the expense of the requester, not having knowledge of successful reception of the request by the completer.

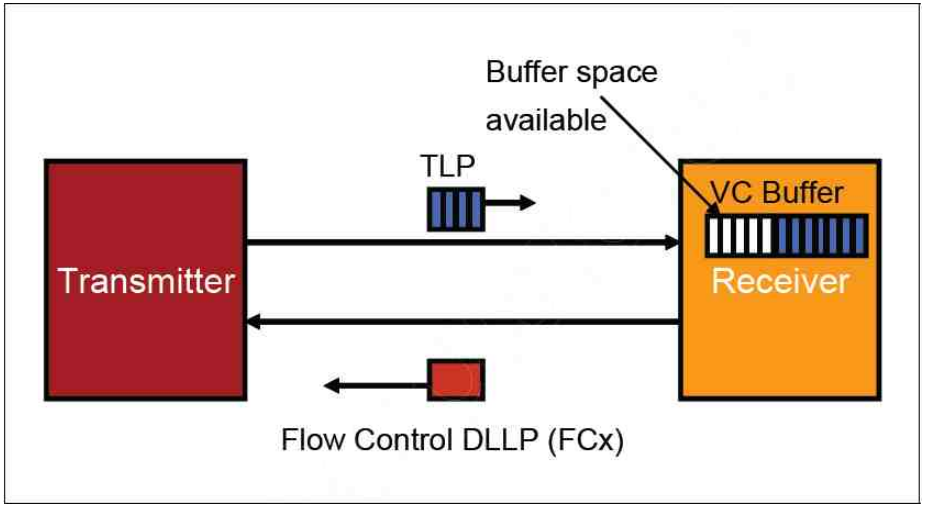
PCI Express Transactions (2)

TLP Packet Types	Abbreviated Name
Memory Read Request	MRd
Memory Read Request - Locked access	MRdLk
Memory Write Request	MWr
IO Read	IORd
IO Write	IOWr
Configuration Read (Type 0 and Type 1)	CfgRd0, CfgRd1
Configuration Write (Type 0 and Type 1)	CfgWr0, CfgWr1
Message Request without Data	Msg
Message Request with Data	MsgD
Completion without Data	Cpl
Completion with Data	CplD
Completion without Data - associated with Locked Memory Read Requests	CplLk
Completion with Data - associated with Locked Memory Read Requests	CplDLk

posted

posted

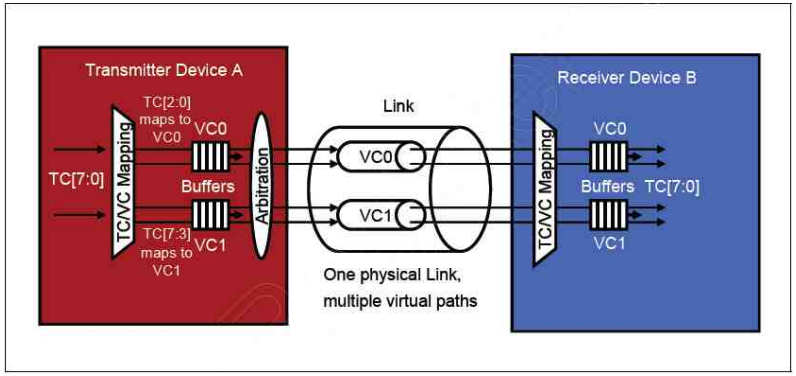
Flow Control



- The Transaction Layer ensures that it does not transmit a TLP over the Link to a remote receiver device unless the receiver device has virtual channel buffer space to accept TLPs (of a given traffic class).
- A receiver device transmits DLLPs called Flow Control Packets (FCx DLLPs) to the transmitter device on a periodic basis.
- The FCx DLLPs contain flow control credit information that updates the transmitter regarding how much buffer space is available in the receiver virtual channel buffer.
- The transmitter keeps track of this information and will only transmit TLPs out of its Transaction Layer if it knows that the remote receiver has buffer space to accept the transmitted TLP.

QoS

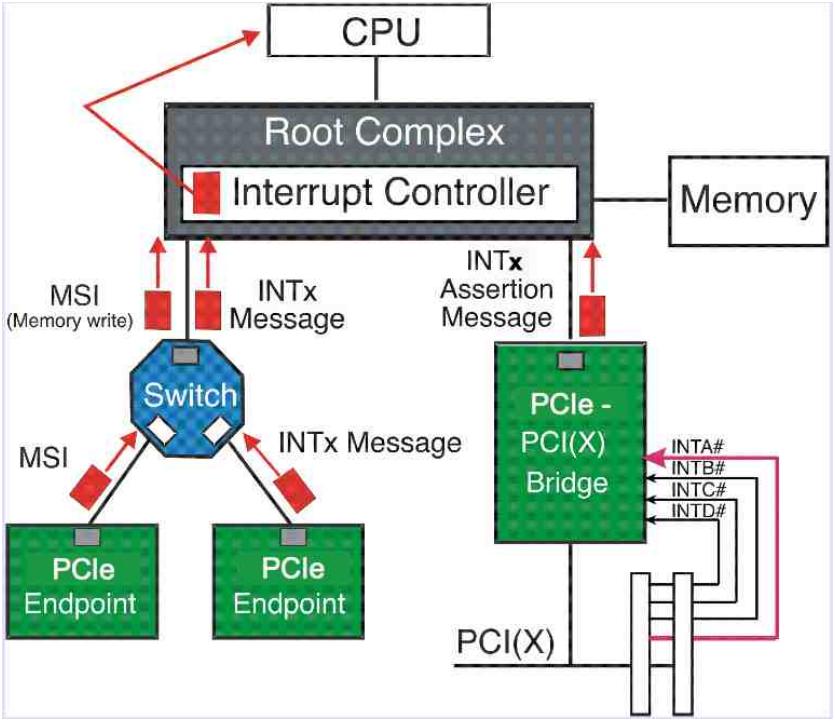
- QoS refers to the capability of routing packets from different applications through the fabric with differentiated priorities and deterministic latencies and bandwidth.
- Virtual Channels enable traffic classes allowing for traffic differentiation.
  8 VCs are supported in the specification.
- Every packet header contains 3 bit of the Traffic Class field, set by driver/endpoint.
- Transaction ordering rules guarantee that TLP traffic associated with a given traffic class is routed through the fabric in the correct order, to prevent potential deadlock or live-lock.



If Device A needs to transmit a TLP with TC label of 7 and another packet with TC label of 0, the two packets will be placed in VC1 and VC0 buffers, respectively. The arbitration logic arbitrates between the two VC buffers. Assume VC1 buffer is configured with higher priority than VC0 buffer. Thus, Device A will forward the TC7 TLPs in VC1 to the Link ahead of the TC0 TLPs in VC0.

Interrupts

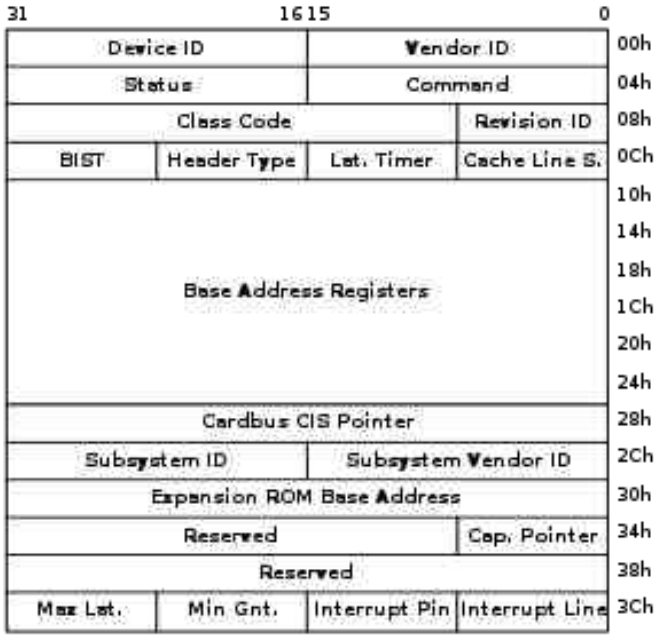
- PCIe generate an MSI to signal an interrupt: It merely consists of sending a TLP over the bus, which is simply a **posted Write Request**, with a special address, which the host has written into the peripheral's configuration space during initialization. The operating system (Linux) can then call the correct interrupt routine, without the need to guess who generated the interrupt. Neither is it really necessary to "clear" the interrupt, if the peripheral doesn't need the acknowledgment.



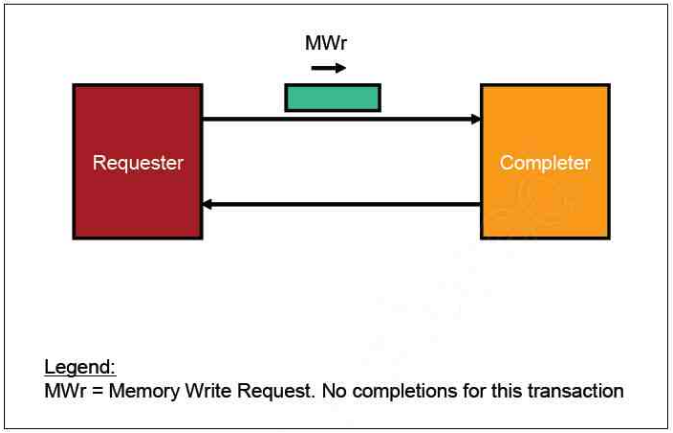
Enumeration

- Enumeration is the process of determining what devices are present in the system and making it available to the Operating System.
- PCIe performs **auto configuration** of the cards inserted in the fabric.
- It occurs only once at the power-up, after each link has been automatically negotiated by the Physical Layer.
  - Read of Vendor ID to validate if a device is present.
  - Reading of the header to determine if the device is and endpoint, switch or bridge.
  - Reading of the BAR to determine the memory extension.

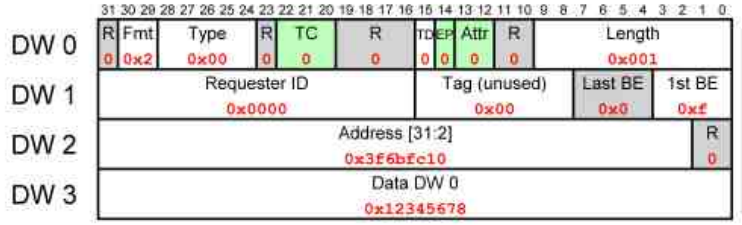
Standard registers Type 0 (endpoint) Configuration Space Header



Example: MWr posted write



- The write request packets that contain data are routed through the PCIe fabric using information in the **header** portion of the packets.
- The completer accepts the specified amount of data within the packets.
- If the write request is received by the completer in error, or it is unable to write the posted write data to the final destination due to an internal error, the requester is not informed via the hardware protocol. The completer could log an error and generate an error message notification to the root complex. Error handling software manages the error.

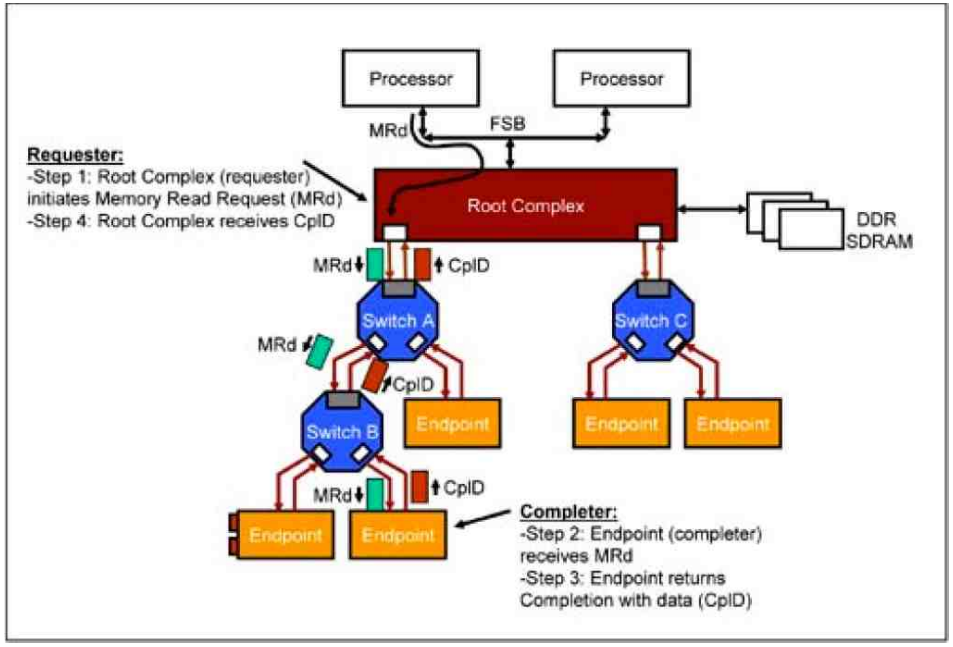


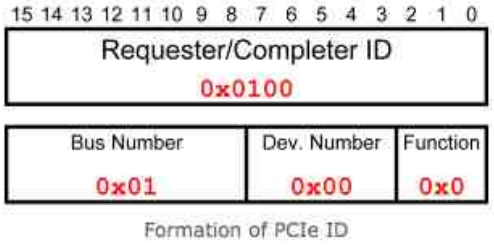
Example of Memory Write Request TLP

- The packet could then consist of four 32-bit words (4 DWs, Double Words) transmitted as 0x40000001, 0x0000000f, 0xfdaff040, 0x12345678
- The Fmt field, together with the Type field say this is a Memory Write Request.
- Length field has the value 0x001, indicating that this TLP has one DW (32-bit word) of data.
- The Requester ID field says that the sender of this packet is known by having ID zero: it's the Root Complex.
- The Address field is simply the address to which the first data DW is written. Well, bits 31-2 of this address. Note that the two LSBs of DW 2 in the TLP are zero, so DW 2 actually reads the write address itself.
  Multiply 0x3f6bfc10 by four, and you get 0xfdaff040.
- PCIe runs big Endian while Intel processors think little Endian. So if this was a regular PC computer, it was writing 0x78563412 in its software representation.

Example (2): non posted read

Non-Posted Memory Read Originated by CPU and Targeting an Endpoint





Formation of PCIe ID

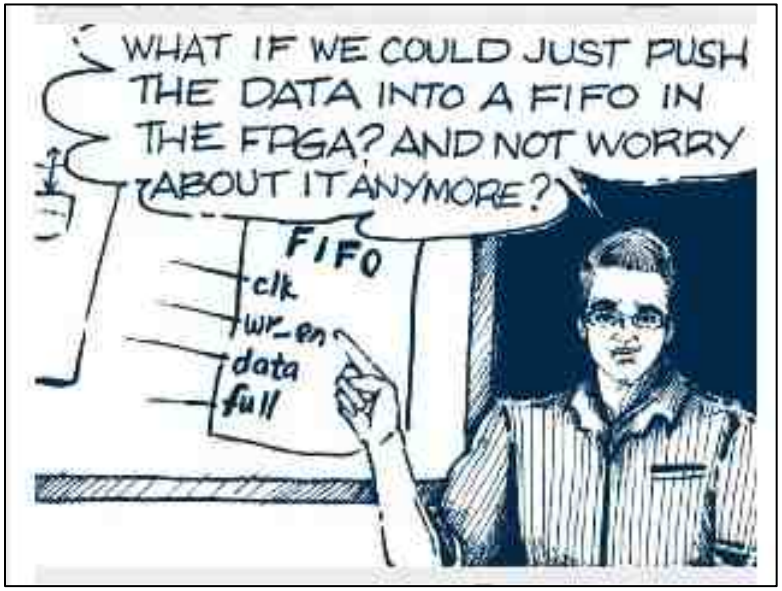
Since PCIe is essentially a packet network, with the possibility of switches on the way, these switches need to know where to send each TLP. There are three routing methods: by address, by ID and implicit.

- The **completer** decodes the contents of the header within the MRd packet, gathers the requested data and returns a completion packet with data (CplD).
- The header portion of the completion TLP contains the requester ID copied from the original request TLP. The requester ID is used to route the completion packet back to the root complex.
- The logical bridges within **Switch B** compares the bus number field of the requester ID in the CplD packet with the secondary and subordinate bus number configuration registers. The CplD packet is forwarded to the appropriate port (in this case the upstream port).
- The CplD packet moves to **Switch A** which forwards the packet to the root complex. The requester ID field of the completion TLP matches the root complex's ID.
- The **root complex** checks the completion status (hopefully "successful completion") and accepts the data. This data is returned to the CPU in response to its pending memory read transaction.

Performance

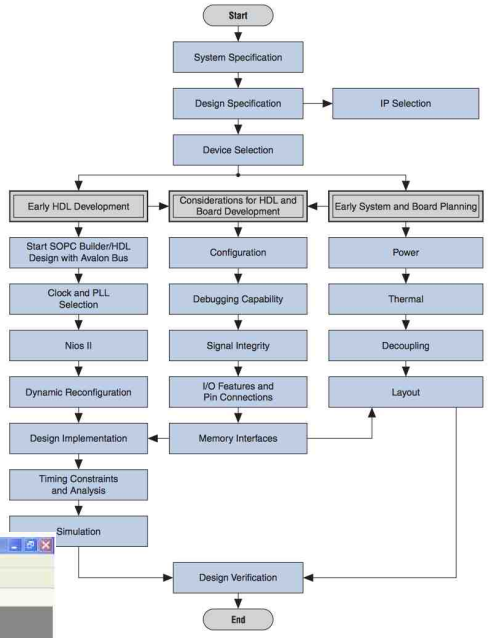
- **Theory is useful for looking at the maximum possible bandwidth**: Theory looks at the amount of overhead incurred when sending a packet and may assume that the TLPs are the only type of information being exchanged.
- **Link management** overhead includes ACK/NACK, power management and traffic control messages: not entirely predictable and system dependent.
- **System management overhead**: some operating systems produce more system management traffic than others.
- **MPS Maximum Payload Size** in most cases **the root complex is the limiting factor**. Upon link negotiation, all devices are configured to only transmit the MPS of the **least capable device** in the layer of the PCIe architecture hierarchy.
- Most **root complexes** only support MPS's of **128** or **256 bytes**.

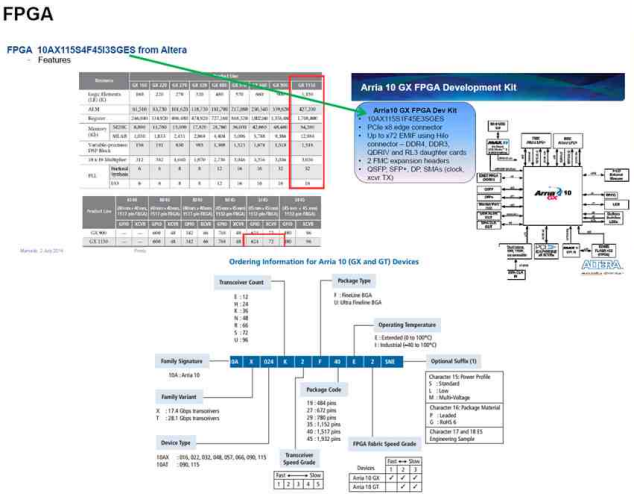
Implementation

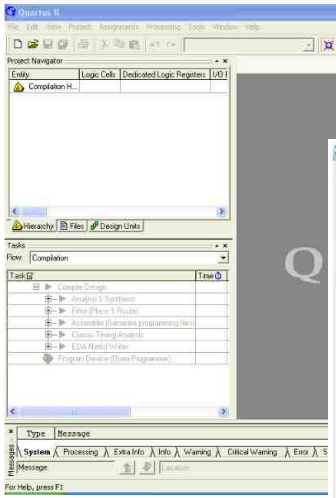


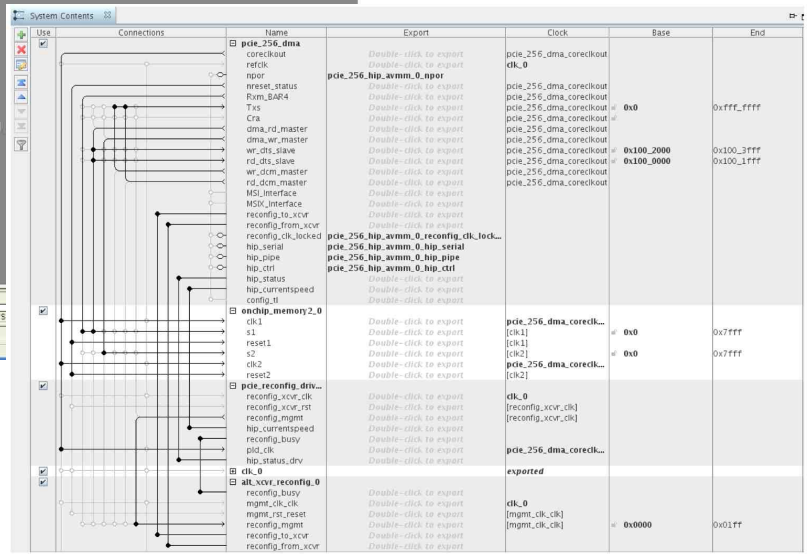
FPGA

- To test the PCIe readout beased approach
- Evaluation boards.
- Programming tools.
  - high level of abstraction
  - to develop and integrate HDL components
- IP core available: especially I/O interfaces
- Integration tools.

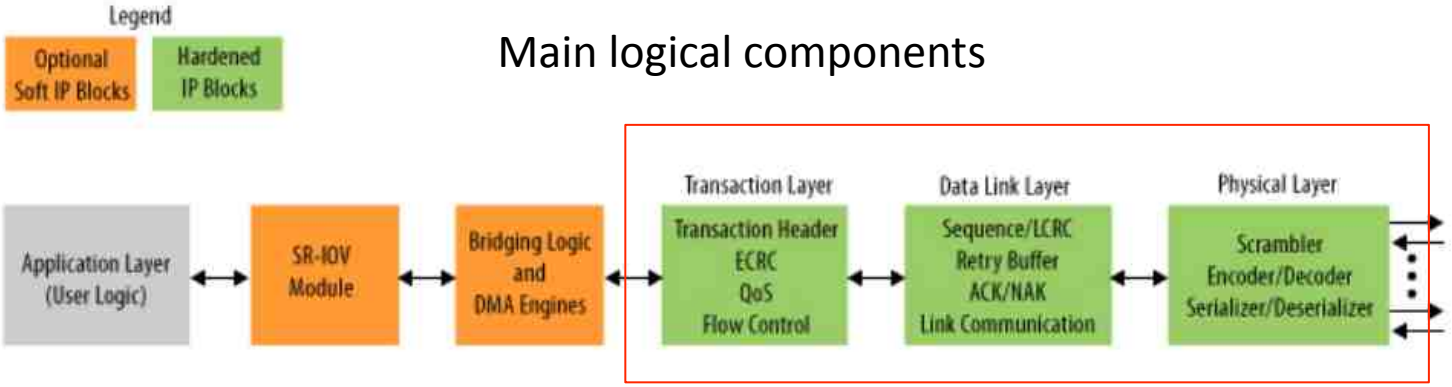




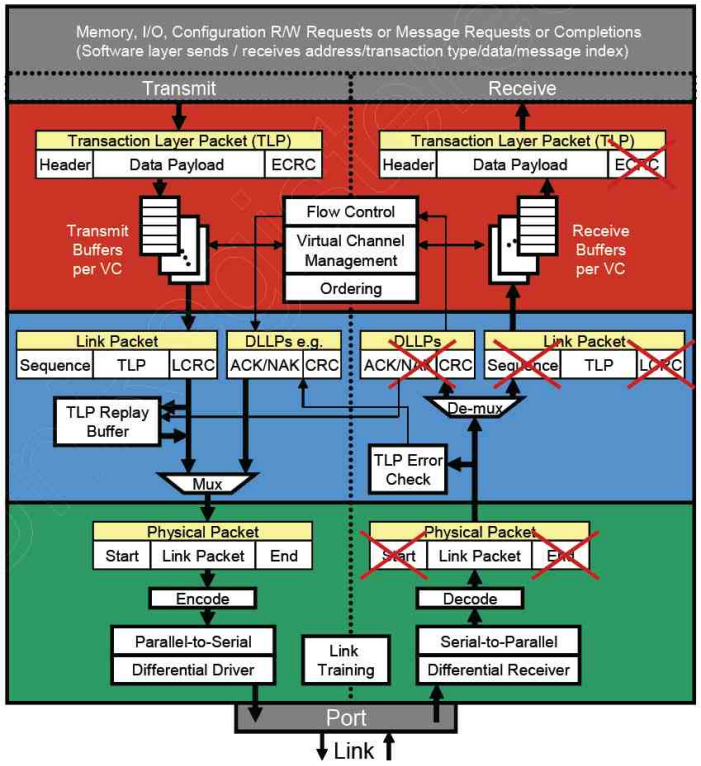




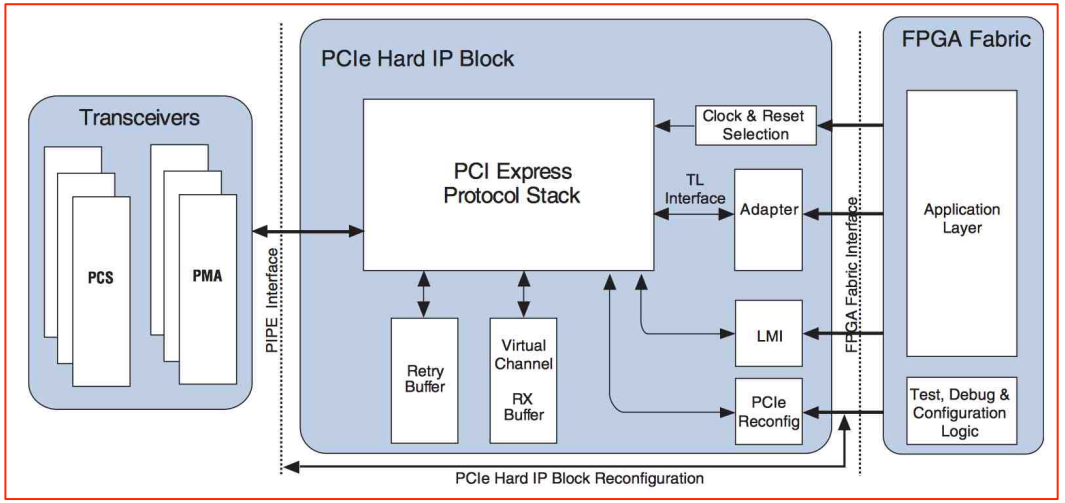
PCIe on FPGA: hard IP blocks



Hard IP block functionalities



Hard IP block



DMA-AvalonMM-PCIe

The Avalon® Memory Mapped (Avalon-MM) 256-bit Hard IP for PCI Express® IP Core.
An address-based read/write interface.

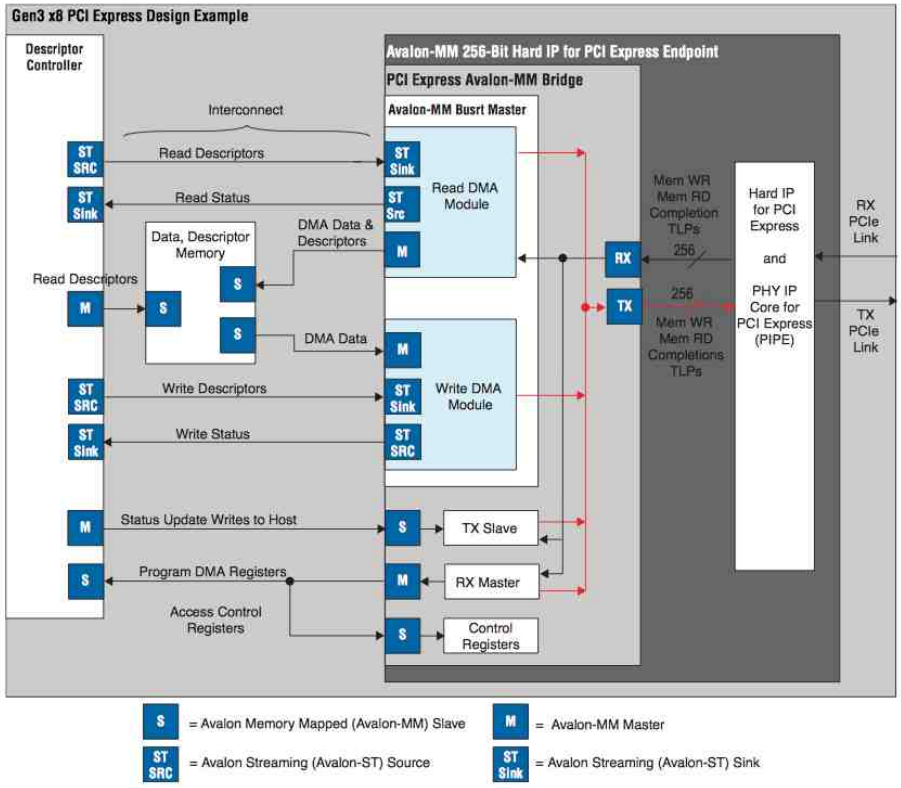
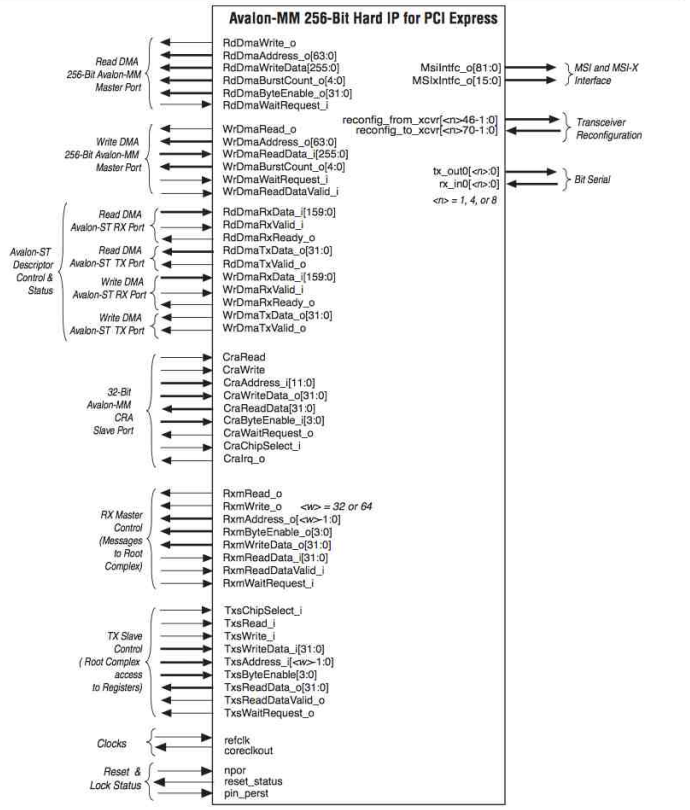


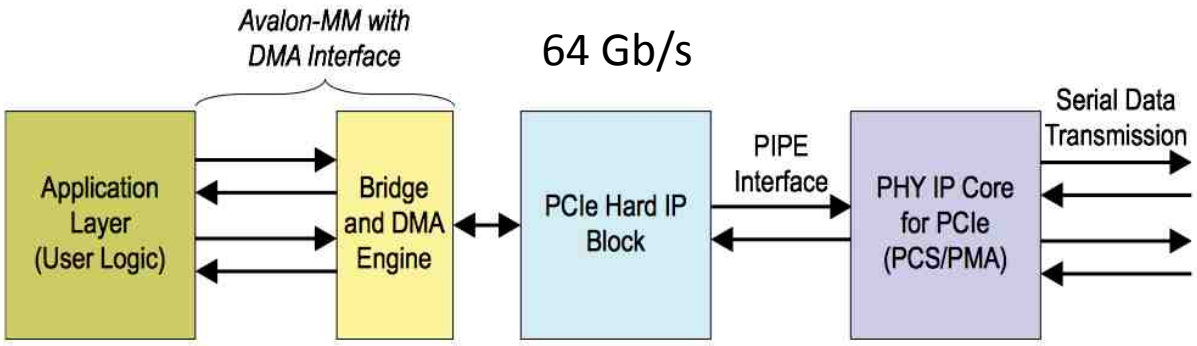
Figure 3–1. Signals in the Avalon-MM 256-Bit Hard IP for PCI Express IP Core



- The descriptor controller acts as the DMA controller.
- The IP core includes a highly efficient Read DMA and Write DMA modules capable of burst reads and writes.

http://www.altera.com/literature/ug/ug_pcie_avmm_dma.pdf

DMA-AvalonMM-PCIe



- It handles TLP encoding and decoding.
- It includes Read DMA and Write DMA engines.
- The Descriptor Controller manages the Read DMA and Write DMA modules.
- Host software programs the Descriptor Controller internal registers with the location of the descriptor table.
- The Descriptor Controller instructs the Read DMA module to copy the entire table to its internal FIFO. It then pushes the table entries to the Avalon-ST RX read and write ports of the Hard IP for PCI Express IP Core. The Descriptor Controller also sends DMA status upstream via an Avalon-MM TX slave port of the Hard IP for PCI Express IP Core.
- Data path 256 bits @ 250 MHz: 64 Gb/s

DMA descriptor controller

Each descriptor contains the source address, destination address, and size of the data to be moved. Source address specifies the location of the data to be moved from by the DMA

Device address space.

Descriptor Table Format

Table 4–1. Descriptor Table Format

Address Offset	Type	Description
0x00	Header	EPLAST - the Descriptor Controller writes the DMA Descriptor ID of the last descriptor in the Descriptor Table to indicate completion of the DMA operation.
0x20	Descriptor 0	Source address lower dword
0x24		Source address upper dword
0x28		Destination address lower dword
0x2C		Destination address upper dword
0x30		Control specifying DMA length in dwords
0x40	Descriptor 1	Source address lower dword
0x44		Source address upper dword
0x48		Destination address lower dword
0x4C		Destination address upper dword
0x50		Control field specifying DMA length
(<n+1>) × 0x20	Descriptor <n>	Source address lower dword
(<n+1>) × 0x20+4		Source address upper dword
(<n+1>) × 0x20+8		Destination address lower dword
(<n+1>) × 0x20+C		Destination address upper dword
(<n+1>) × 0x20+F		Control field specifying DMA length

Descriptor Controller DMA Write Registers

Table 4–2. DMA Write Registers

Address	Register	Access	Description
0x0000	Write DMA Control	R/W	Records the write control and status information and specifies the number of descriptors. Refer to Table 4–4 for details.
0x0004	Write DMA Status	R	Write DMA Status. Refer to Table 4–5 for details.
0x0008	RC Write Descriptor Base (Low)	R/W	Specifies the low-order 32-bits of the base address of the write descriptor table in the Root Complex memory.
0x000C	RC Write Descriptor Base (High)	R/W	Specifies the high-order 32-bits of the base address of the write descriptor table in the Root Complex memory.
0x0010	Last Write Descriptor Index	R/W	Specifies the last DMA Descriptor ID to be processed.
0x0014	EP Descriptor Table Base (Low)	RW	Specifies the lower 32 bits of the base address of the write descriptor table in Endpoint memory.
0x0018	EP Descriptor Table Base (High)	RW	Specifies the upper 32 bits of the base address of the write descriptor table in Endpoint memory.

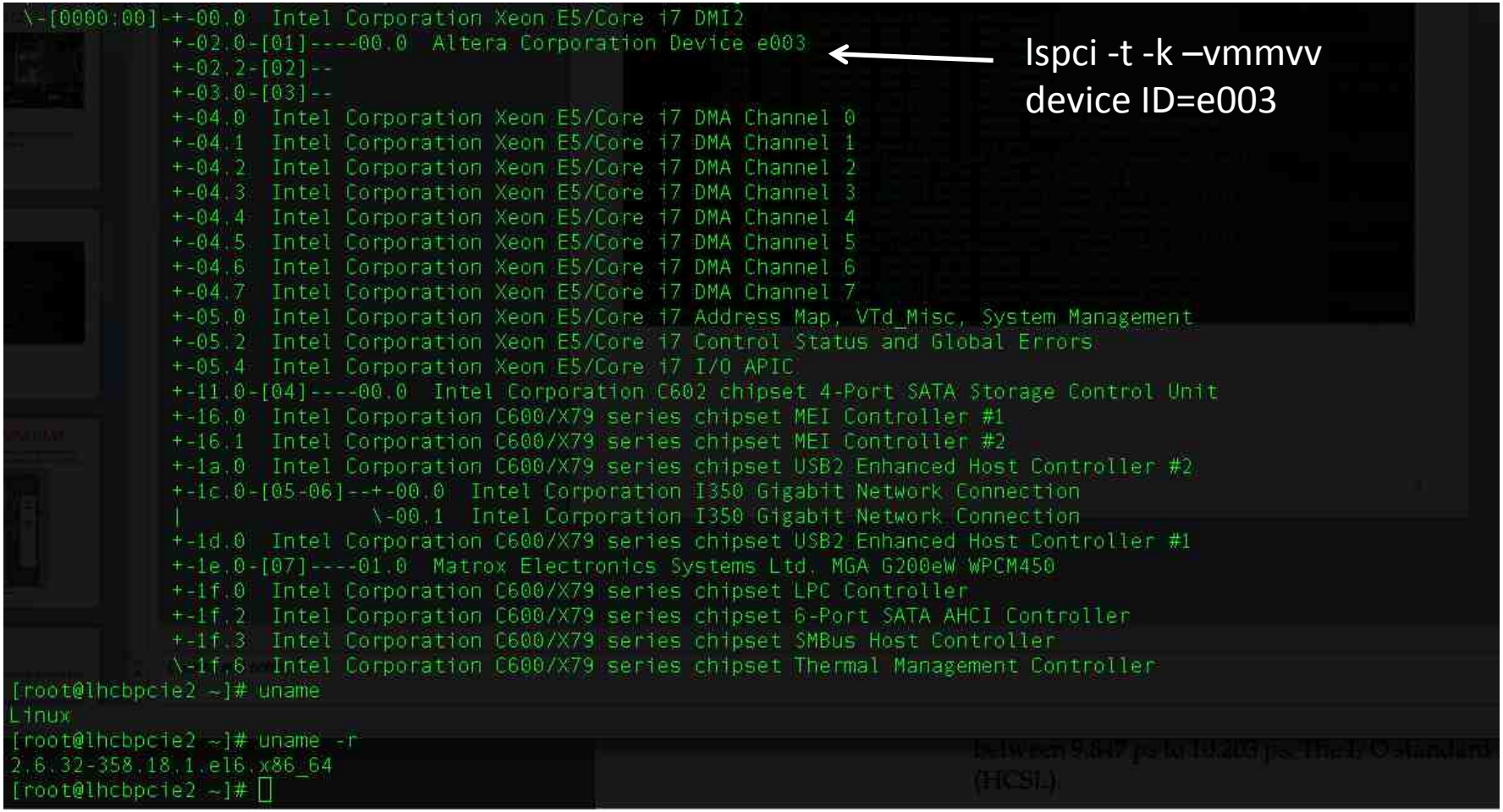
Table 4–5. DMA Status Register Bit Mappings

Bits	Name	Description
[31:10]	Reserved.	—
9	Busy	When asserted, a DMA is in progress.
8	Completed	When asserted, a single DMA descriptor has completed successfully.
[7:0]	Descriptor ID	Specifies DMA Descriptor ID of the descriptor currently being transferred.

The following table shows the DMA write registers. Host software writes these registers to set up a DMA transfer.

Startup

- After the server booting the device is known to the OS.



PCIe Device Driver

- From a software standpoint**, PCI and PCI Express devices are essentially the same**: the driver does not "know" the difference between two boards.
  - PCIe devices can take advantage of "advanced" features: hot-plugging, MSI, extended configuration space, etc.
- What is most relevant to the driver writer is PCIe's support for **auto-detection of interface boards**. The device driver must be able to access configuration information in the device in order to complete initialization.
- Each PCI peripheral is identified by a *bus* number, a *device* number, and a *function* number.
- **PCI and PCIe DMA Address Management**
  Consistent DMA mappings are synchronous and coherent. These mappings are usually mapped by the driver at initialization time and unmapped only when the device is no longer running. Consistent DMA mappings guarantee that the device and the CPU can access the data in parallel and can see each other's updates without explicit software flushing.

Device driver: DMA setting up

altera_dma.h

```
#ifndef _ALTERA_DMA_H
#define _ALTERA_DMA_H

#define ALTERA_DMA_DRIVER_NAME    "Altera DMA"
#define ALTERA_DMA_DEVFILE        "altera_dma"

#define ALTERA_DMA_BAR_NUM (6)
#define ALTERA_DMA_DESCRIPTOR_NUM 127

#define ALTERA_DMA_WR_DMA_CTRL          0x0000
#define ALTERA_DMA_WR_DESC_STATUS       0x0004
#define ALTERA_DMA_WR_RC_DESC_BASE_LOW  0x0008
#define ALTERA_DMA_WR_RC_DESC_BASE_HIGH 0x000C
#define ALTERA_DMA_WR_LAST_DESC_IDX     0x0010
#define ALTERA_DMA_WR_EP_DESC_BASE_LOW  0x0014
#define ALTERA_DMA_WR_EP_DESC_BASE_HIGH 0x0018
#define ALTERA_DMA_WR_DMA_PERF          0x001C

#define ALTERA_DMA_RD_DMA_CTRL          0x0100
#define ALTERA_DMA_RD_DESC_STATUS       0x0104
#define ALTERA_DMA_RD_RC_DESC_BASE_LOW  0x0108
#define ALTERA_DMA_RD_RC_DESC_BASE_HIGH 0x010C
#define ALTERA_DMA_RD_LAST_DESC_IDX     0x0110
#define ALTERA_DMA_RD_EP_DESC_BASE_LOW  0x0114
#define ALTERA_DMA_RD_EP_DESC_BASE_HIGH 0x0118
#define ALTERA_DMA_RD_DMA_PERF          0x011C

#define ALTERA_TOGGLE_DMA_BR            0x0200

#define ALTERA_DMA_NUM_DWORDS           0x1000

#define ONCHIP_MEM_BASE                 0x0000
#define ONCHIP_MEM_DESC_MEM_BASE        0x8000

#define DESC_CTRLLER_BASE               0x8000

#define CTL_STS_BITS                    0x0100

#define TIMEOUT_THRESH                  0xFFFF
```

- Host software (driver) writes the content of the DMA registers in order to set up the DMA transfer mechanism.

Table 4–2. DMA Write Registers

Address	Register	Access	Description
0x0000	Write DMA Control	R/W	Records the write control and status information and specifies the number of descriptors. Refer to Table 4–4 for details.
0x0004	Write DMA Status	R	Write DMA Status. Refer to Table 4–5 for details.
0x0008	RC Write Descriptor Base (Low)	R/W	Specifies the low-order 32-bits of the base address of the write descriptor table in the Root Complex memory.
0x000C	RC Write Descriptor Base (High)	R/W	Specifies the high-order 32-bits of the base address of the write descriptor table in the Root Complex memory.
0x0010	Last Write Descriptor Index	R/W	Specifies the last DMA Descriptor ID to be processed.
0x0014	EP Descriptor Table Base (Low)	RW	Specifies the lower 32 bits of the base address of the write descriptor table in Endpoint memory.
0x0018	EP Descriptor Table Base (High)	RW	Specifies the upper 32 bits of the base address of the write descriptor table in Endpoint memory.

altera_dma.c

```
[root@lhcbpcie2 dma-master]# grep ALTERA_DMA_WR_DMA_CTRL *.c
altera_dma.c:       writel (CTL_STS_BITS | bk_ptr->altera_dma_descriptor_num, bk_ptr->bar[0]+DESC_CTRLLER_BASE+ALTERA_DMA_WR_DMA_CTRL);
altera_dma.c:        writel (CTL_STS_BITS | bk_ptr->altera_dma_descriptor_num, bk_ptr->bar[0]+DESC_CTRLLER_BASE+ALTERA_DMA_WR_DMA_CTRL);
[root@lhcbpcie2 dma-master]#
```

Device init

```
File Edit Options Buffers Tools C Help
#include <linux/time.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/module.h>
#include <linux/device.h>
#include <linux/pci.h>
#include <linux/cdev.h>
#include <linux/dma-mapping.h>
#include <linux/interrupt.h>
#include <linux/delay.h>
#include <linux/sched.h>
#include "altera_dma_cmd.h"
#include "altera_dma.h"

static unsigned short vid = 0x1172;
static unsigned short did = 0xE003;
module_param(vid, ushort, S_IRUGO);
module_param(did, ushort, S_IRUGO);
bool stoppe = false;
u8 *rp_buffer_virt_addr = 0;

static int start_stop_b_DMA(struct altera_pcie_dma_bookkeep *bk_ptr)
{
  // bk_ptr->stop_dma = !bk_ptr->stop_dma;
  printk(KERN_DEBUG "Toggle Buffer Ring DMA mode");

  printk(KERN_DEBUG "toggling DMA ");
  if (stoppe)
    {
      printk(KERN_DEBUG "stop DMA RING...");
      writel (0x0000, bk_ptr->bar[0]+DESC_CTRLLER_BASE+ALTERA_TOGGLE_DMA_BR);
```

Table 3: Device Identification Register Settings

Parameter	Value
Vendor ID	0x00001172
Device ID	0x0000E003
Revision ID	0x00000001
Class Code	0x00000000
Subsystem Vendor ID	0x00000000
Subsystem Device ID	0x00002861

PCIe test tools (3)

- Motherboard/CPU

Device init

```
[root@lhcbpcie2 ~]# lsmod|less

Module                  Size  Used by
altera_dma             12394  0
autofs4                26481  3
coretemp                7381  0
cpufreq_ondemand       10544  32
acpi_cpufreq            7763  0
freq_table              4936  2 cpufreq_ondemand,acpi_cpufreq
mperf                   1557  1 acpi_cpufreq
ipv6                  318183  76
uinput                  7992  0
iTCO_wdt                7115  0
```

altera_dma module

pci_register_driver()

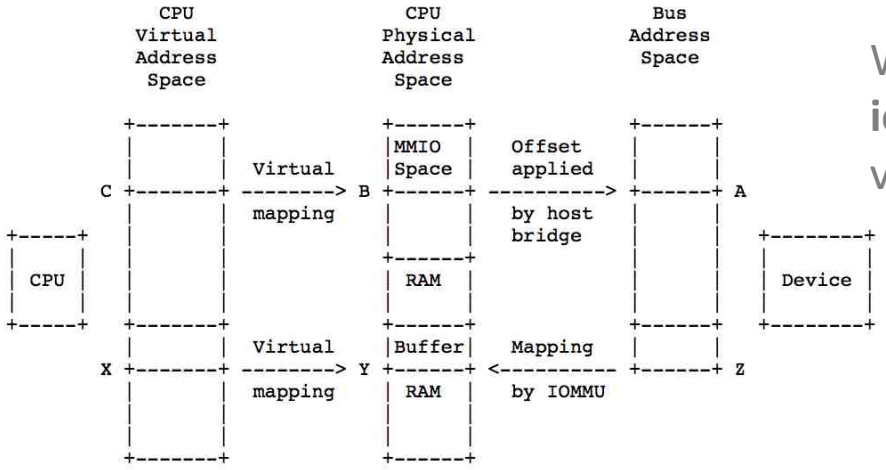
```
[root@lhcbpcie2 ~]# dmesg
Altera DMA: altera_dma_init(), Feb 20 2014 15:15:00
Altera DMA 0000:01:00.0: VID = 0x1172, Device = 0xe003, Class = 0x0, bus:slot.func = 01:00.00
  alloc irq_desc for 32 on node 0
  alloc kstat_irqs on node 0
alloc irq_2_iommu on node 0
Altera DMA 0000:01:00.0: PCI INT A -> GSI 32 (level, low) -> IRQ 32
Altera DMA 0000:01:00.0: pci_enable_device() successful on device [vid 0x1172, did 0xE003]
Altera DMA 0000:01:00.0: setting latency timer to 64
  alloc irq_desc for 117 on node 0
  alloc kstat_irqs on node 0
alloc irq_2_iommu on node 0
Altera DMA 0000:01:00.0: irq 117 for MSI/MSI-X
Altera DMA 0000:01:00.0: pci_enable_msi() successful
Altera DMA 0000:01:00.0: using a 64-bit irq mask
Altera DMA 0000:01:00.0: irq pin: 1
Altera DMA 0000:01:00.0: irq line: 11
Altera DMA 0000:01:00.0: irq: 117
Altera DMA 0000:01:00.0: request irq: 11
Altera DMA 0000:01:00.0: BAR[0] 0xce600000-0xce60ffff flags 0x0012120c, length 65536
Altera DMA 0000:01:00.0: BAR[1] 0x00000000-0x00000000 flags 0x00000000, length 0
Altera DMA 0000:01:00.0: BAR[2] 0x00000000-0x00000000 flags 0x00000000, length 0
Altera DMA 0000:01:00.0: BAR[3] 0x00000000-0x00000000 flags 0x00000000, length 0
Altera DMA 0000:01:00.0: BAR[4] 0x00000000-0x00000000 flags 0x00000000, length 0
Altera DMA 0000:01:00.0: BAR[5] 0x00000000-0x00000000 flags 0x00000000, length 0
Altera DMA 0000:01:00.0: BAR[0] mapped to 0xffffc90017620000, length 65536
[root@lhcbpcie2 ~]#
```

pci_enable_device()

Memory mapping

ioremap()

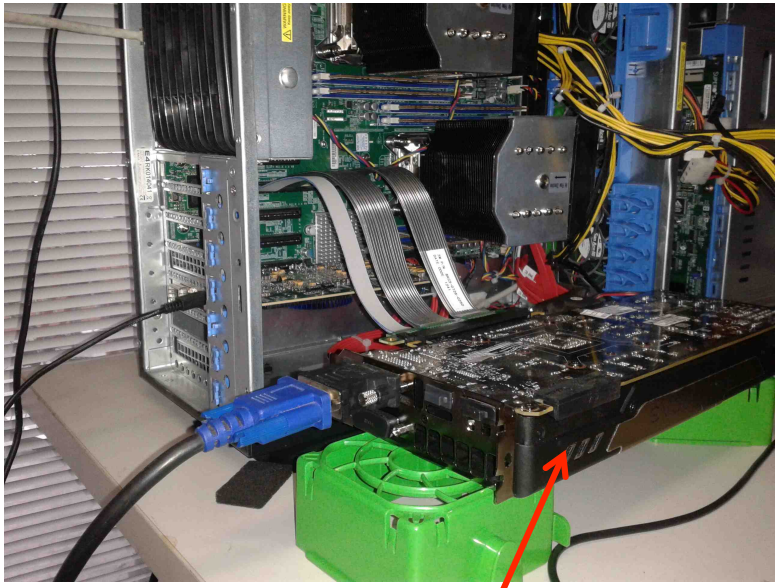
Device driver: Memory mapping



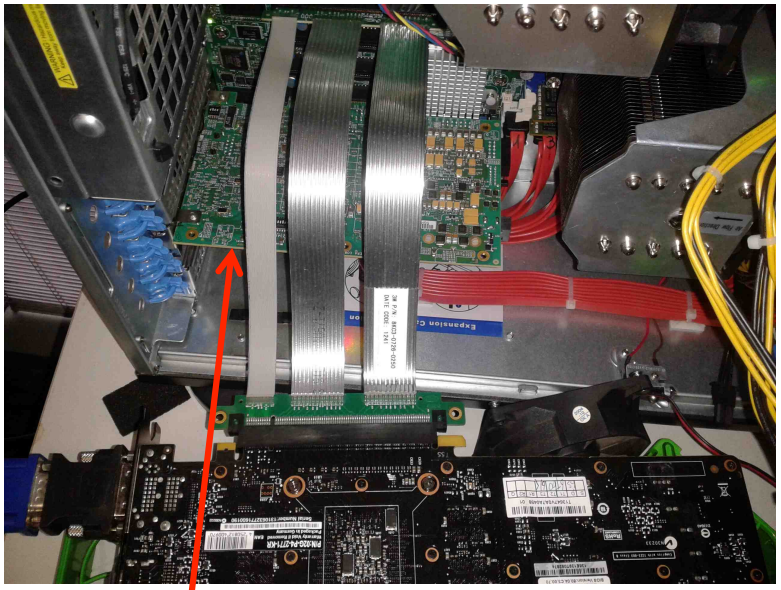
When a driver claims a device, it typically uses **ioremap()** to map physical address (B) at a virtual address (C).

```
static int __init map_bars(struct altera_pcie_dma_bookkeep *bk_ptr, struct pci_dev *dev)
{
  int i;
  for (i = 0; i < ALTERA_DMA_BAR_NUM; i++) {
    unsigned long bar_start = pci_resource_start(dev, i);
    //unsigned long bar_end = pci_resource_end(dev, i);
    //unsigned long bar_flags = pci_resource_flags(dev, i);
    bk_ptr->bar_length[i] = pci_resource_len(dev, i);
    if (!bk_ptr->bar_length[i]) {
        bk_ptr->bar[i] = NULL;
        continue;
    }
    bk_ptr->bar[i] = ioremap(bar_start, bk_ptr->bar_length[i]);
    if (!bk_ptr->bar[i]) {
      dev_err(&dev->dev, "could not map BAR[%d]", i);
      return -1;
    } else
    dev_info(&dev->dev, "BAR[%d] mapped to 0x%p, length %lu", i, bk_ptr->bar[i], (long unsigned int)bk_ptr->bar_length[i]);
  }
  return 0;
}
```

PCIe test tools



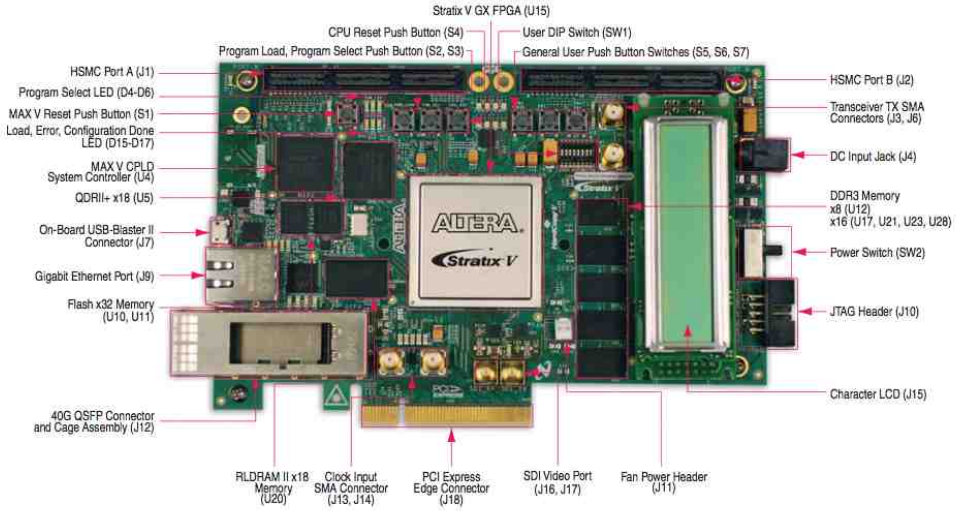
GPU used to test 16-lane PCIe-3 data transfer between the device and the host memory

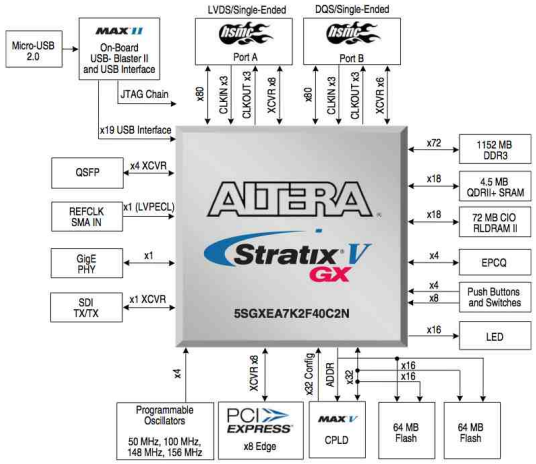


ALTERA development board 8-lane PCIe-3

PCIe test tools

- We have beeb using an ALTERA evaluation board, mounting a: Stratix V GX FPGA: 5SGXEA7K2F40C2N
- http://www.altera.com/literature/manual/rm_svgx_fpga_dev_board.pdf

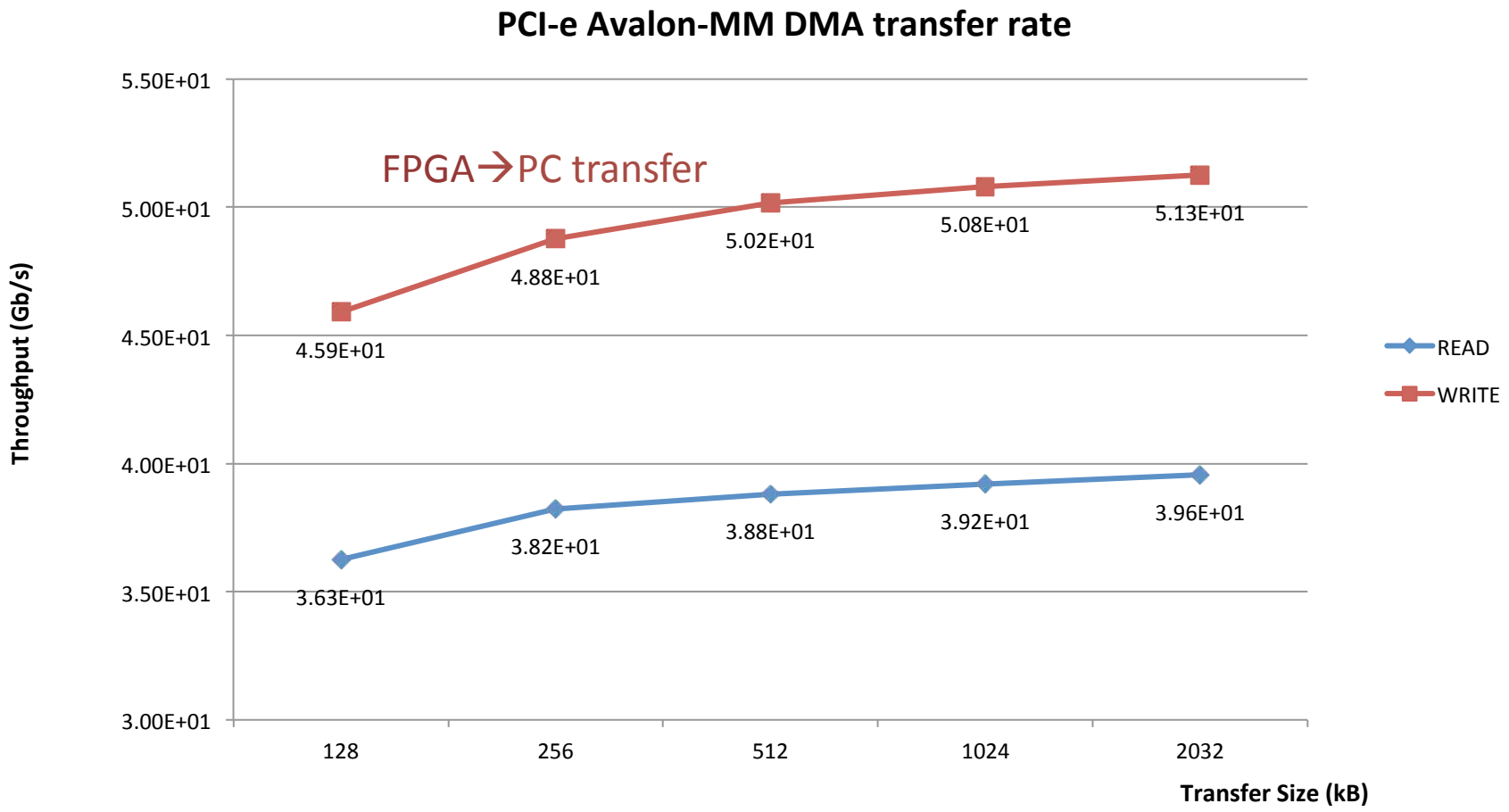




"The Stratix V GX FPGA development board is designed to fit entirely into a PC motherboard with a ×8 PCI Express slot that can accommodate a full height short form factor add-in card. This interface uses the Stratix V GX FPGA device's PCI Express hard IP block, saving logic resources for the user logic application".

Transfer rate

Number of DMA descriptors set according to:
transfer-size = n × data_word × 4.



Infiniband

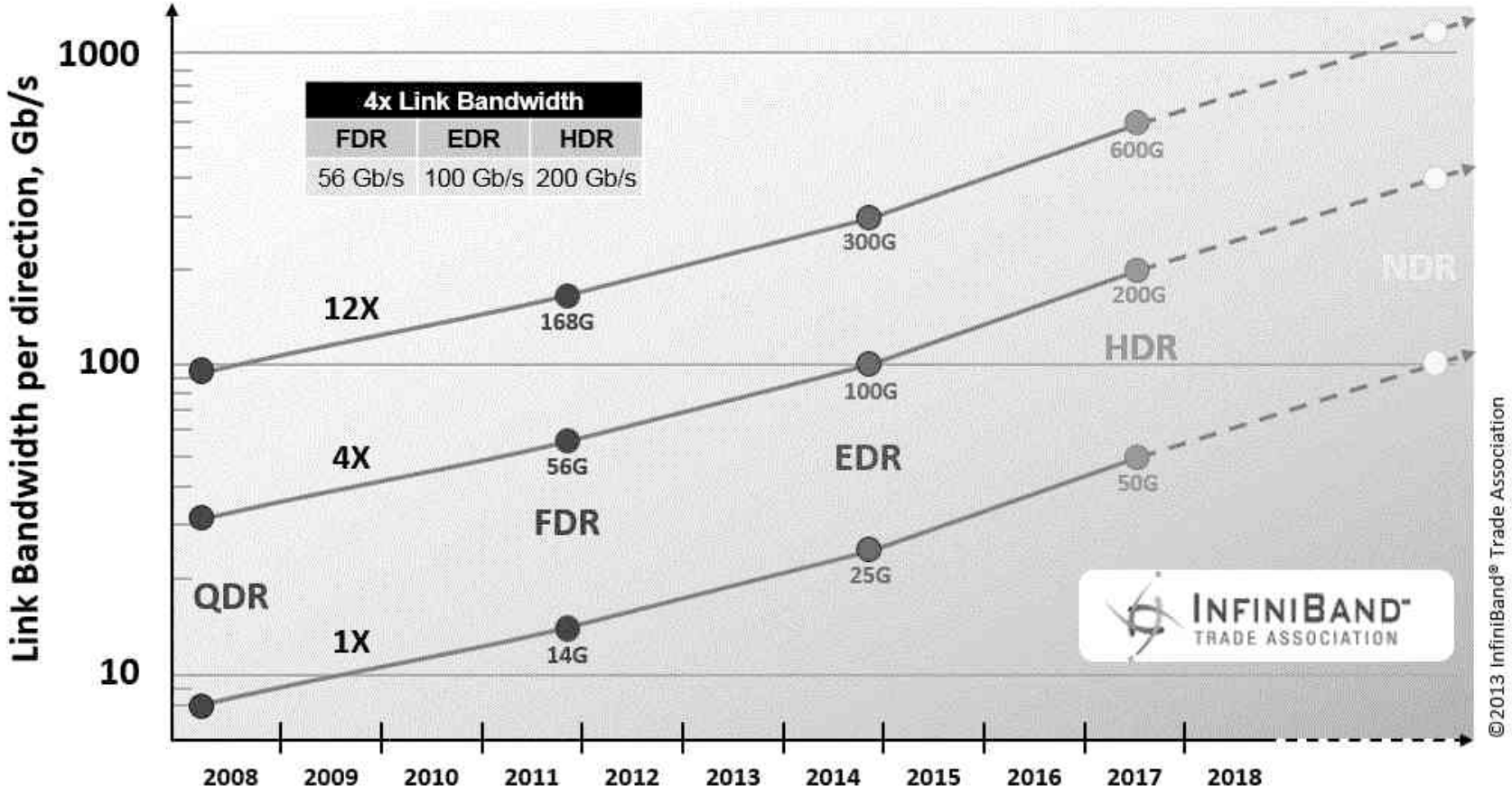
Introduction

- **InfiniBand (IB**) is a high performances network technology, developed by the IB Trade Association in 1999.
- InfiniBand (IB) is a high-speed, low latency, low CPU overhead, highly efficient and scalable server and storage interconnect technology:
  High bandwidth: ~ 100 Gb/s, **Low latency**: ~ 1 µs
- **The most important point is that every application has direct access to the virtual memory of devices in the fabric**:
  - Contrast this with the **traditional network environment** where the shared network resources are owned by the operating system and cannot be accessed by a user application.
- InfiniBand enables data transfer between servers and between server and storage without the involvement of the host CPU in the data path.
- One of the key capabilities of IB is its support for native **Remote Direct Memory Access** (RDMA).

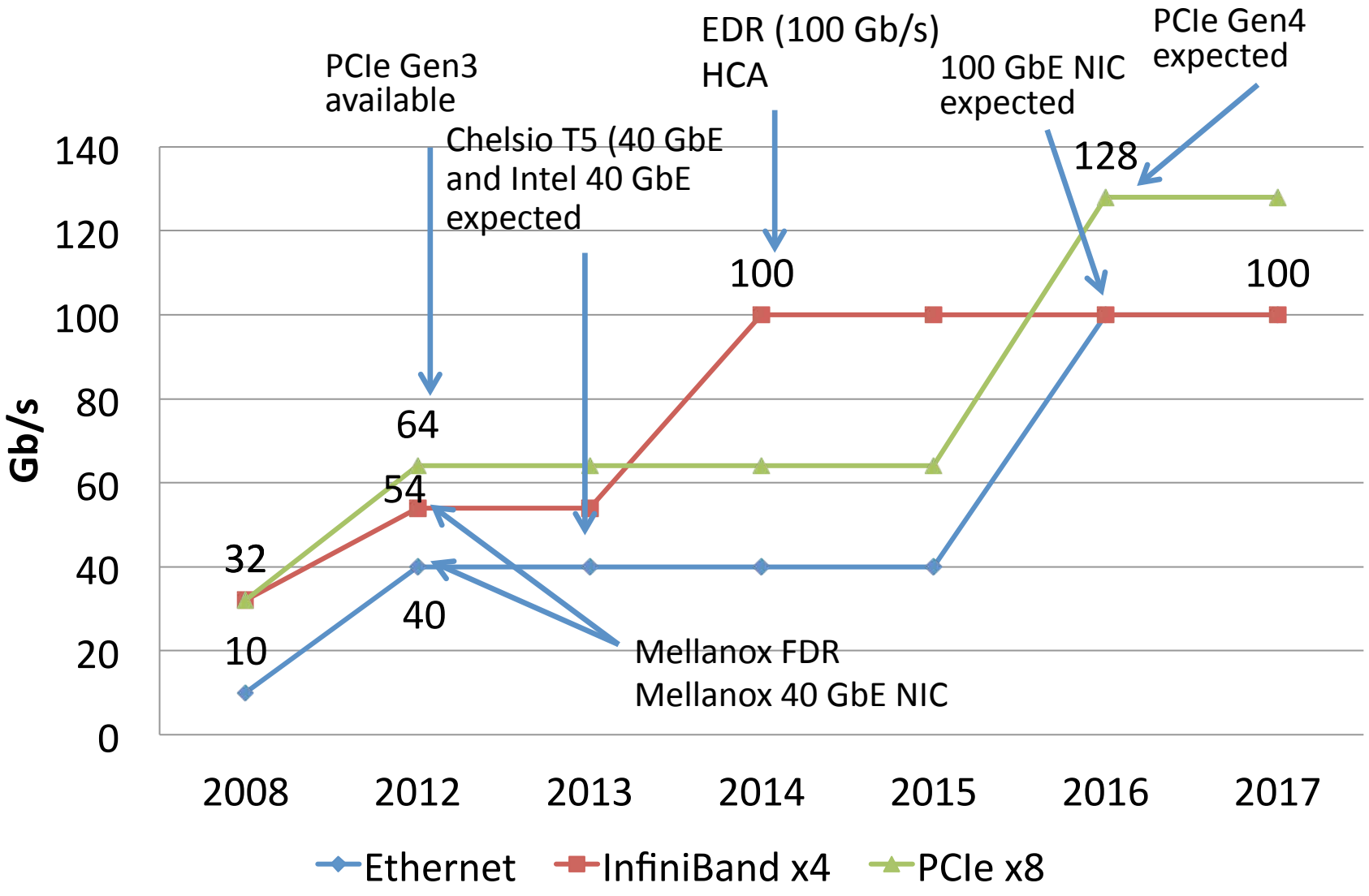
Traditional network environment

- In the traditional network environment the shared network resources are owned by the operating system and cannot be accessed by a user application.
- An application must rely on the involvement of the OS to move data from the application's virtual buffer space, through the network stack and out onto the wire.
- Similarly, at the other end, an application must rely on the operating system to retrieve the data on the wire on its behalf and place it in its virtual buffer space.

Bandwidth

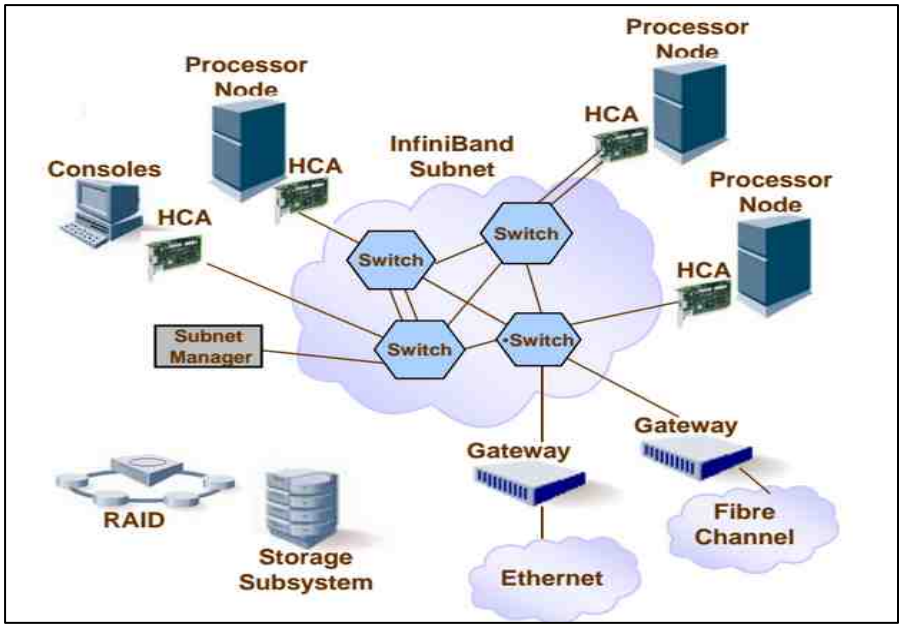

Link Bandwidth per direction, Gb/s
1000
100
10
4x Link Bandwidth
FDR
EDR
HDR
56 Gb/s
100 Gb/s
200 Gb/s
12X
4X
1X
QDR
FDR
EDR
HDR
NDR
168G
300G
600G
56G
100G
200G
14G
25G
50G
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
INFINIBAND
TRADE ASSOCIATION
©2013 InfiniBand® Trade Association

The evolution of Network Interconnects



IB Architecture Elements

- **Link**: serial connection, full duplex, can be aggregated (most of the systems use 4x)
- **Host Channel Adapter** (HCA): Device that terminate an IB link and executes transport level functions.
- **Switch**: A device that routes packets from one link to another of the same IB subnet
- **Routers**: A device that routes packets from one subnet to another.



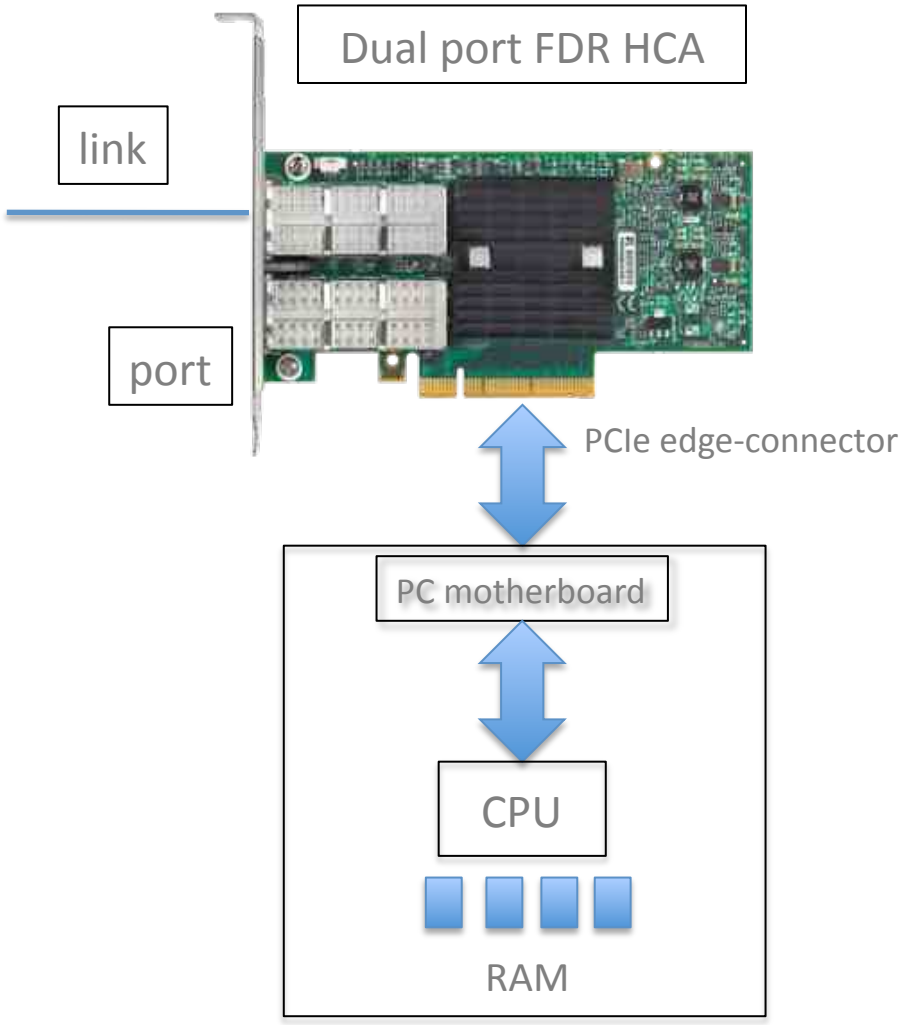
- **Subnet**: A set of **ports** and **links** with a common **SubnetID** and managed by a common **Subnet Manager.**
- **Subnet Manager**: The SM is the entity that discovers all of the devices in the subnet at the startup time, configures them and performs a periodic sweep of the subnet to detect any changes to the subnet's topology.
- During the **configuration** the SM assigns each of the **ports** a unique **Local Identifier Address** (**LID**) as well as a common SubnetID**.**

Components

- **Host Channel Adapter** (HCA): Device that terminate an IB link and executes transport level functions.
- **Switch**: A device that routes packets from one link to another of the same IB subnet
- **Routers**: A device that routes packets from one subnet to another.

Host Channel Adapter (HCA)

- **HCA** is the equivalent of the Ethernet NIC.
- HCA is connected to the motherboard of the host PC through **PCI Express**.
- Identified by the **Global Unique Identifier GUID**: is the equivalent of the Ethernet MAC address.
- The majority of the protocol layers is implemented in silicon, minimizing the burden placed on software and processor.
- **CPU offload** of the transport operations: The CPU of the host server is not involved in message passing. Network message transfers are handled by **hardware DMA transfer engines** within the channel adapter.



INFN Istituto Nazionale di Fisica Nucleare

Message passing

- The underlying concept behind IBA is the **message passing** between CAs.
- A **message** is a block of data that is passed as **packets** from the local memory of one CA to the local memory of another CA.
  - **Message length**: from zero to 2 GB.
  - **Packet size** (maximum packet payload size): 4 kB
- **Three types of message transfers foreseen**:
  - **Sending a message** from the local CA's memory to the destination CA's memory.
  - **Reading a a message** from the destination CA's and storing it in the local CA's memory.
  - Performing **an atomic RMW** (read/modify/write) in the destination CA's memory storing the returned data in the local CA's memory

INFN Istituto Nazionale di Fisica Nucleare

Message passing (2)

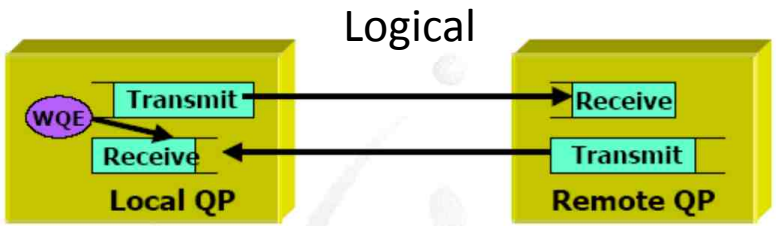
- **Writing a message to the remote CA's memory.**
- **Message Send operation**. The request does not tell the destination CA where to write data in its local memory. It is up to the destination CA to determine where to write the data in its local memory.
- **Remote DMA (RDMA) write operation**: the request specifies where data has to be written in the destination CA's local memory. The request packet contains:
  - the data payload field;
  - the memory start address;
  - the transfer length;
  - the write permission key.
- **Remote DMA (RDMA) read operation**: the CA issues a request to another CA to read a block of data from its local memory. The destination CA returns data in a series of RDMA read response packets. Upon receipt the requesting CA stores data in its own local memory.

Message passing (3)

- **Reading a message to the remote CA's memory.**
- **Remote DMA (RDMA) read operation**: the CA issues a request to another CA to read a block of data from its local memory. The destination CA returns data in a series of RDMA read response packets. Upon receipt the requesting CA stores data in its own local memory.
  - The memory start address;
  - The amount of data to be read;
  - The read permission key.
- **Atomic RMW in the remote CA's memory**:
  - **Atomic fetch and add operation**.
    - Memory address
    - The add value
    - Special key to access the location.
  - **Atomic compare and swap if equal operation**.
    - Memory address
    - The compare value
    - The swap value
    - Special key to access the location.

Message Transfer Mechanisms

Queue Pair: a bidirectional message transport engine



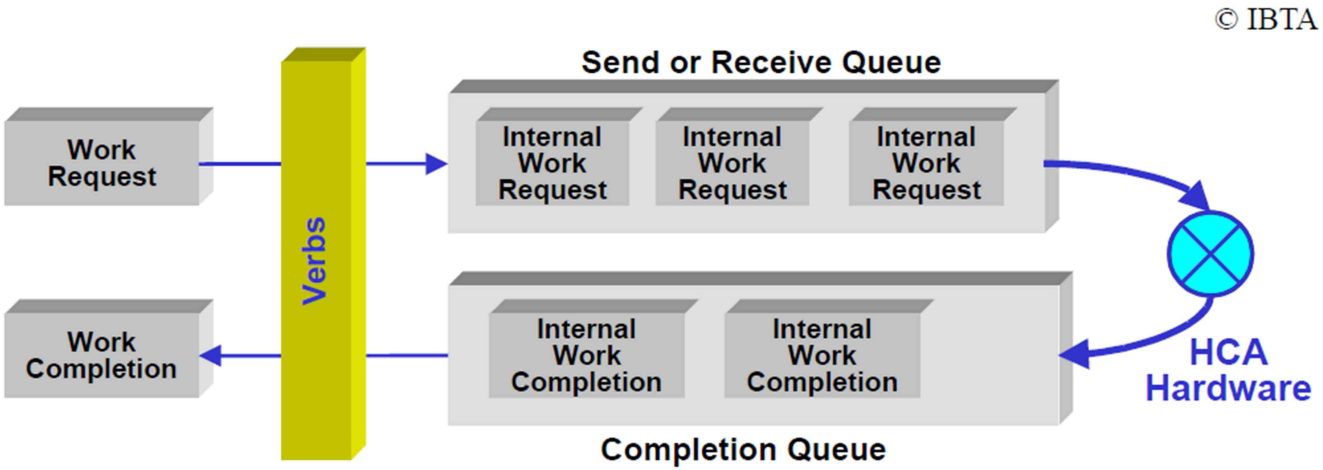
- Each CA implements a number of bi-directional message transport engines, each of which is referred as a **Queue Pair** (QP).
  - Multiple QP can be associated with each local CA port.
- A QP consists of two queues:
  Send Queue (SQ) and **Receive Queue (RQ)**.
- **SQ**: Message transfer are posted to this queue by software. The SQ logic transmits an outbound message transfer request (in a series of one or more request packets) to a remote RQ logic.
- **RQ**: software posts work request to this queue to handle inbound messages from remote SQ logic.

INFN Istituto Nazionale di Fisica Nucleare

Queue Pairs Types

- QP are the engines, therefore the **queue type** are referred as the **transport service types**.
- The type of QP must be **specified by software** when the QP is created.
- According to the IBA specifications there are **five types of QP**:
  - **RC**: Reliable Connected.
    - A RC connection is very similar to a TCP connection
  - **UC**: Unreliable Connected.
  - **RD**: Reliable Datagram.
  - **UD**: Unreliable Datagram.
  - **RAW**. Used to send and receive non-IBA packets that are encapsulated within an IBA packet.

Completion Queue



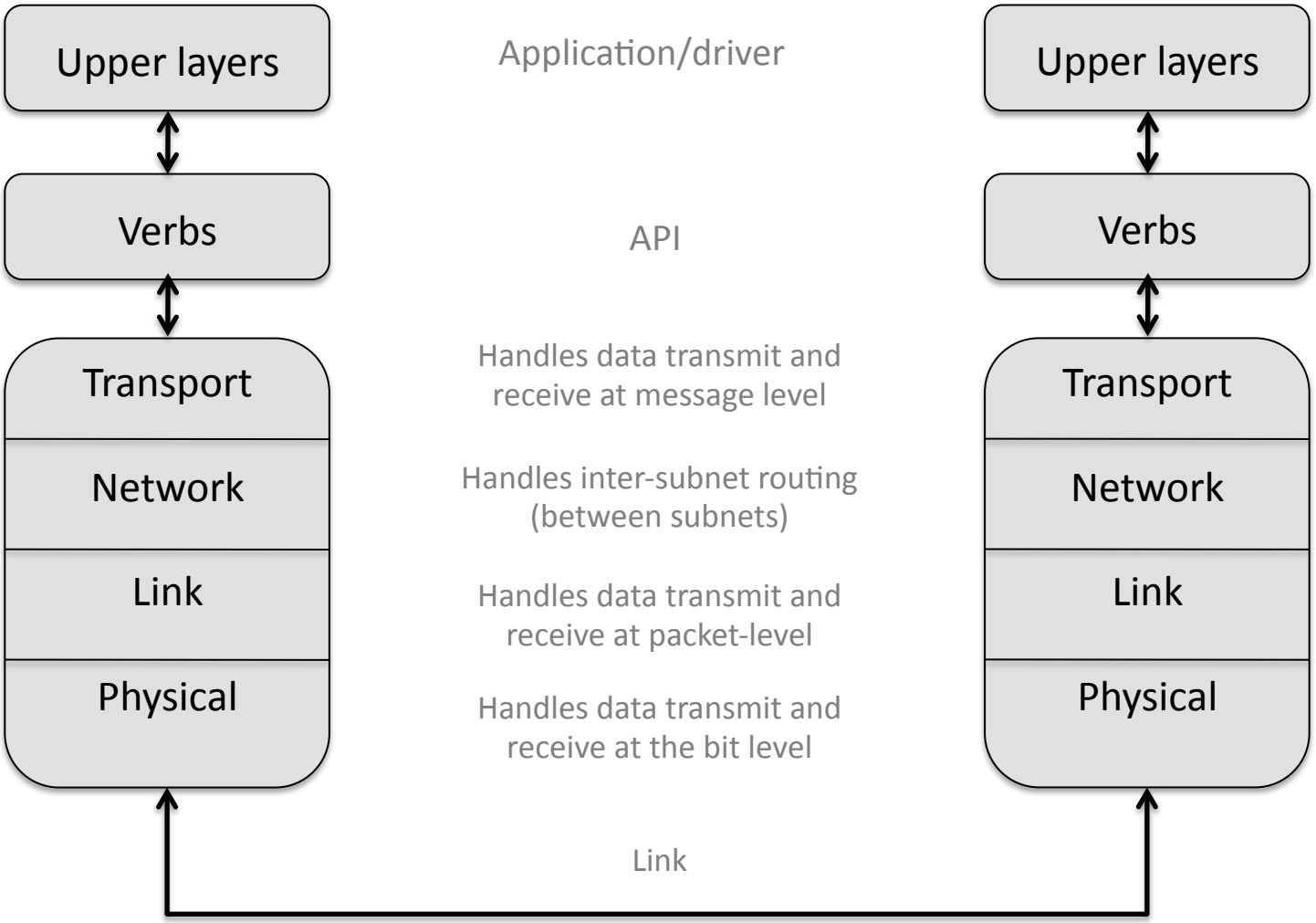
- A Completion Queue is a mechanism to notify the application about information of ended Work Requests (status, opcode, size, source).
- Every completion says that a specific WR was completed (both successfully completed WRs and unsuccessfully completed WRs).
- To send or receive messages, **Work Requests** (WRs) are placed onto a QP. When processing is completed, a **Work Completion** (WC) entry is optionally placed onto a **Completion Queue** (CQ) associated with the work queue.

INFN Istituto Nazionale di Fisica Nucleare

Memory registration

- Memory Registration is a mechanism that allows an application to describe a set of virtually contiguous memory locations or a set of physically contiguous memory locations to the network adapter as a virtually contiguous buffer using Virtual Addresses.
- The registration process **pins the memory pages** to prevent the pages from being swapped out and to keep physical <-> virtual mapping.
- During the registration, the OS checks the permissions of the registered block.
- The registration process writes the virtual to physical address table to the network adapter.
- When registering memory, permissions are set for the region. Permissions are local write, remote read, remote write, atomic, and bind.

IB Layers


Upper layers
Verbs
Transport
Network
Link
Physical
Application/driver
API
Handles data transmit and receive at message level
Handles inter-subnet routing (between subnets)
Handles data transmit and receive at packet-level
Handles data transmit and receive at the bit level
Link
Upper layers
Verbs
Transport
Network
Link
Physical

IB Layers (2)

- **Software** builds Work Request (WR) in system memory.
- **Software** posts WR to CA, QP, SQ.
- **Transport layer** handles transfers at message-level and segments messages into packets if necessary.
- **Network layer** adds GRH to each packet if necessary.
- **Link layer** handles transfer at packet level
  - insert SLID
  - select transmit buffer
  - check for flow control credits
  - performs the transmit buffer arbitration.
- **Physical layer** convert character to the bit coding, serializes ad transmits bit stream over the link

INFN Istituto Nazionale di Fisica Nucleare

Physical Layer

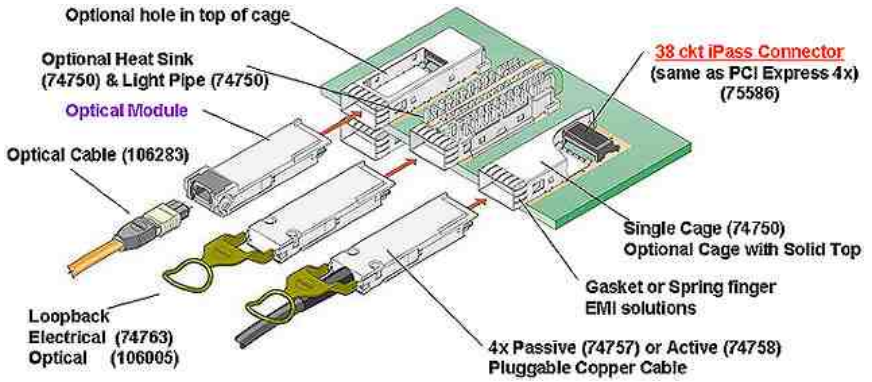
- The physical layer specifies how bits are placed on the wire to form symbols and defines the symbols used for framing (i.e., start of packet and end of packet), data symbols, and fill between packets (Idles). It specifies the signaling protocol as to what constitutes a validly formed packet.
- InfiniBand is a lossless fabric. Maximum Bit Error Rate (BER) allowed by the IB specification is 10^{-12}. The physical layer should guaranty affective signaling to meet this BER requirement.
- InfiniBand uses **serial stream** of bits to transfer data, with bit encoding.
- **Link width**: 1x, 4x, 12x, differential pairs per Tx and Rx.
- **Link speed**: Single (SDR), Double (DDR), Quad (QDR), etc.

Physical Layer (2)

- Industry standard components
- Copper cables and connectors
  - 20m SDR, 10m DDR, 7m QDR, etc.
- Optical cable and connectors
  - 300m SDR, 150m DDR, 100/300m QDR, etc.

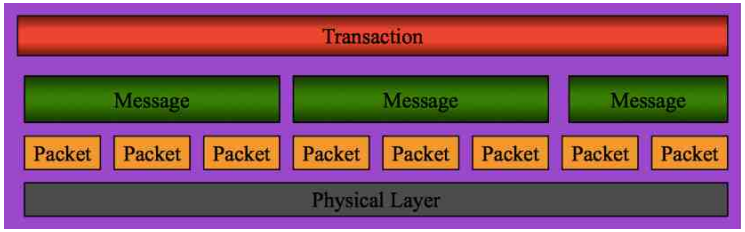
CX4 cables

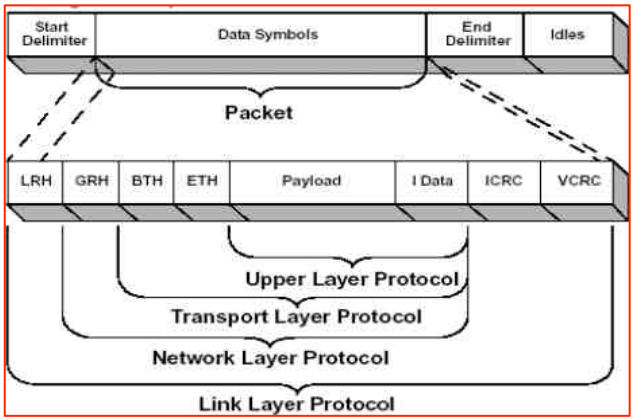
QSFP



Link Layer

- The link layer describes the packet format and protocols for packet operation, e.g. flow control and how packets are routed within a subnet between the source and destination

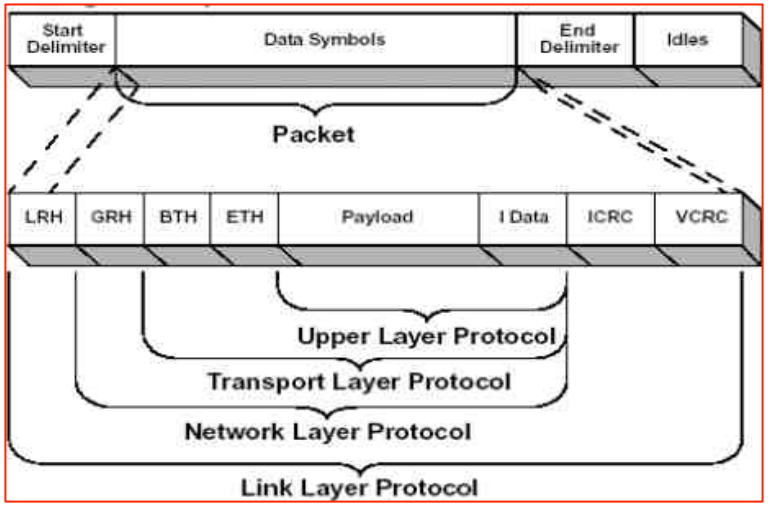




- Packets are routable end-to-end fabric unit of transfer.
- Link management packets: train and maintain link operation
- **Local Addressing:**
  - 16-bit field in the Local Routing Header (LRH) of all IB packets
  - Each CA port must be assigned a 16-bit unique **LID** address within the subnet at the startup time.
  - Assigned by the Subnet Manager

Addressing

- **Local Addressing**
  - Each CA port must be assigned a 16-bit unique **LID** address within the subnet at the startup time.
  - Assigned by the Subnet Manager
  - **BTH** base transport header: what type of message transfer request it is.
- **Global Addressing**
  - GRH Global address is used to route a packet from its source CA port in one subnet to the destination CA port in another subnet.
    IPv6 addresses are introduced.

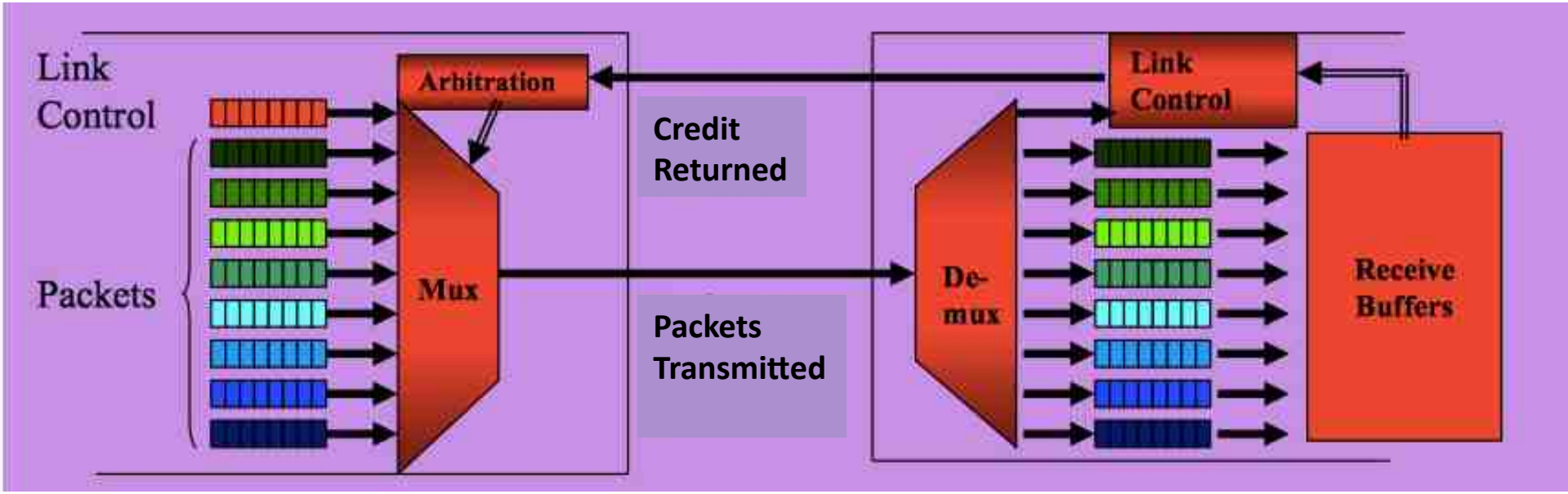


INFN Istituto Nazionale di Fisica Nucleare

Virtual Lanes (QoS)

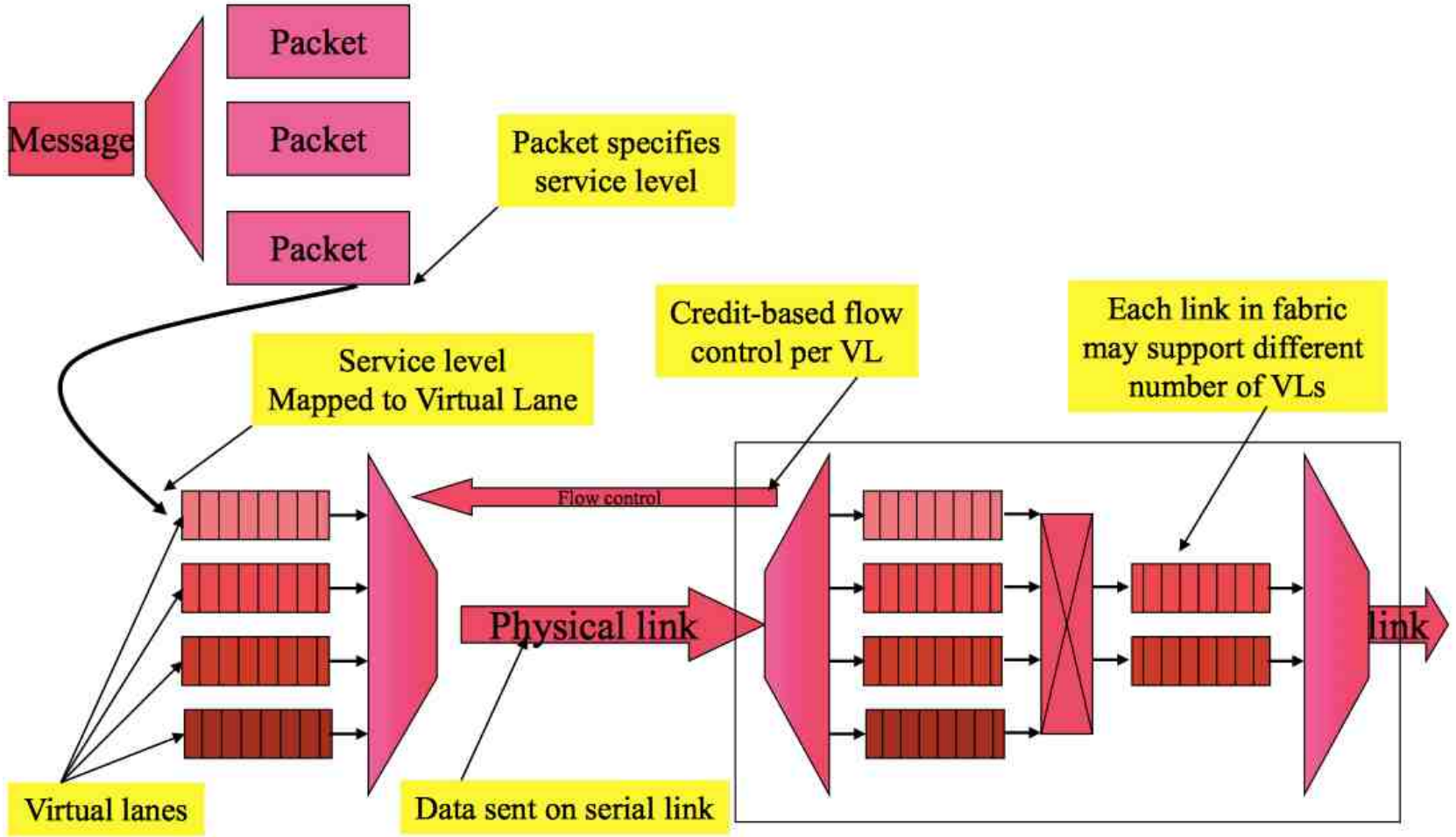
- **16 Service Levels (SLs)**
  - A field in the Local Routing Header (LRH) of an InfiniBand packet
  - Defines the requested QoS
- **Virtual Lanes (VLs)**
  - A mechanism for creating multiple channels within a single physical link.
  - Each VL is associated with a set of Tx/Rx buffers in a port
  - Each VL has separate flow-control
- A configurable Arbiter control the Tx priority of each VL
- Each SL is mapped to a VL
- IB Spec allows a total of 16 VLs (15 for Data & 1 for Management)
- Minimum of 1 Data and 1 Management required on all links
- Switch ports and HCAs may each support a different number of VLs
- VL 15 is a management VL and is not a subject for flow control

Link Layer: flow control



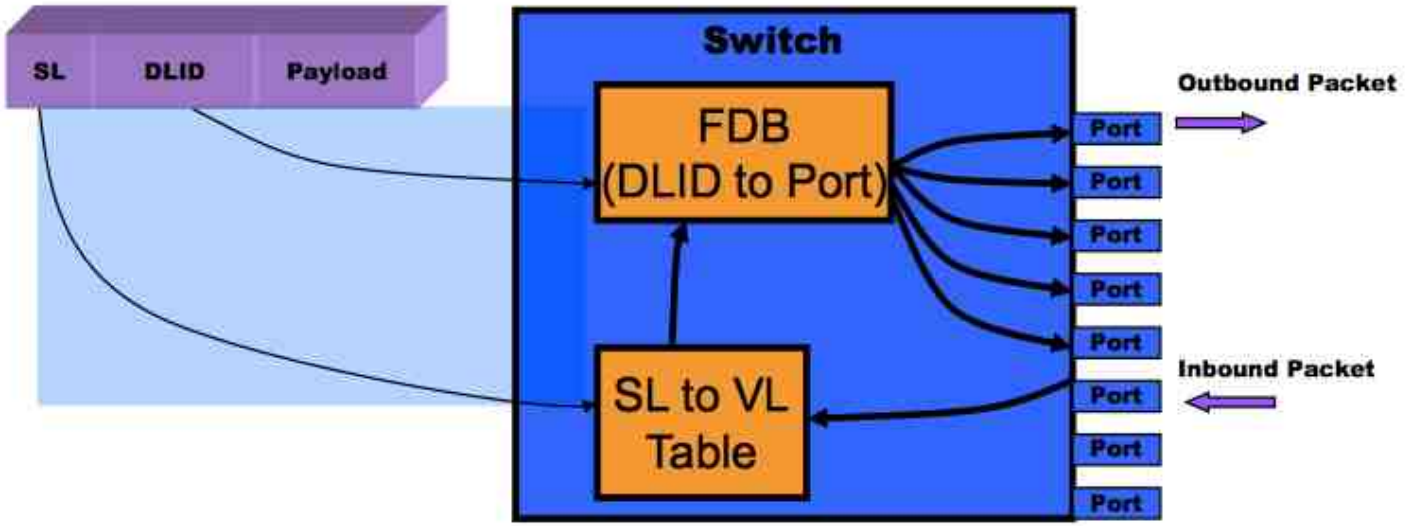
- Credit-based link-level flow control.
- Link flow control assures NO packet loss within the IB fabric even in the presence of congestion.
- Link Receivers grant packet receive buffer space credits per Virtual Lane: separate flow control per Virtual Lanes.

Example

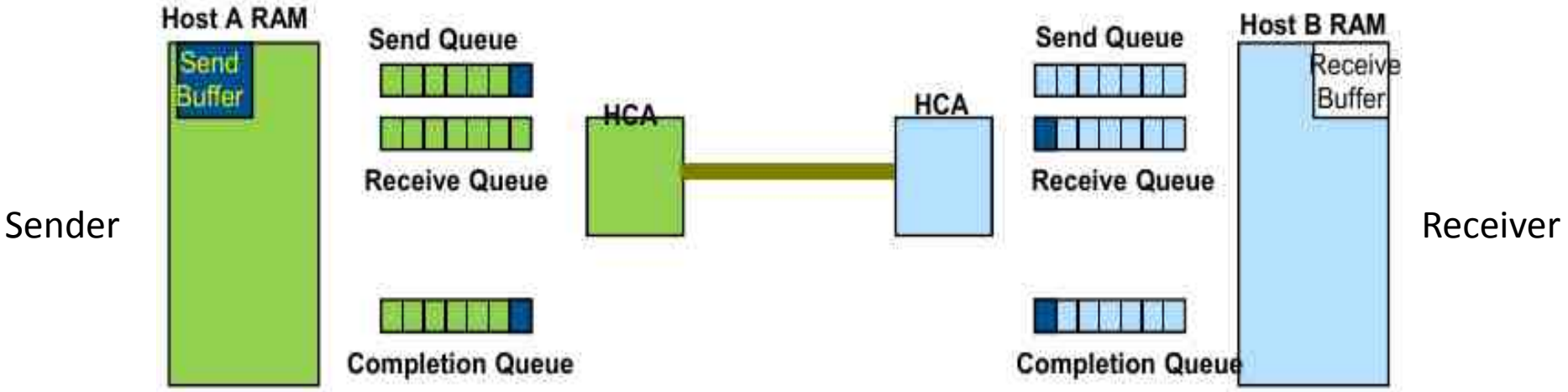

Packet
Message
Packet
Packet
Packet specifies service level
Service level Mapped to Virtual Lane
Credit-based flow control per VL
Each link in fabric may support different number of VLs
Flow control
Physical link
link
Virtual lanes
Data sent on serial link

Forwarding

- Switches use FDB (Forwarding Database)
- Based on DLID and SL a packet is sent to the correct output port.
- Multicast Destinations supported.



Send Operation Example



- The send operation allows you to send data to a remote QP's receive queue. The receiver must have previously posted a receive buffer to receive the data. The sender does not have any control over where the data will reside in the remote host.
- Host A (Sender) allocates a send buffer on User Space Virtual Memory and register it to the HCA
- Host B (Receiver) allocates a receive buffer on User Space Virtual Memory and register it to the HCA
- Host B communicates that is ready to receive messages on the completion queue.
- Host A HCA sends a message to the remote side.
- Host B receives packets (message) and generate an entry in the completion queue