Managment and Analysis of Physical Data – part I a.a. 2020-21

Hardware for Real Time and On-Line Data Management and Processing

G.Collazuol
Dipartimento Fisica e Astronomia UNIPD

- Digital Systems and Data Path
- Hardware for Data Processing

The importance of Hardware in Data Management and Processing

Hardware is crucial in all steps of data journey from source to sink

The choice of hardware parts and configuraton is crucial for real-time applications requiring response in well defined time

Hardware deeply impacts on data type and quality and in data collection and processing time.

Consider the following steps:

- sensors → analog data → signal/noise shaping
- analod to digital conversion
- data buffering and preparation for transport (away from source)
- data processing / filtering / early selection
- data buffering and analysis
- digital to analog conversion
- response analog signal → actuators

The importance of Hardware in Data Management and Processing

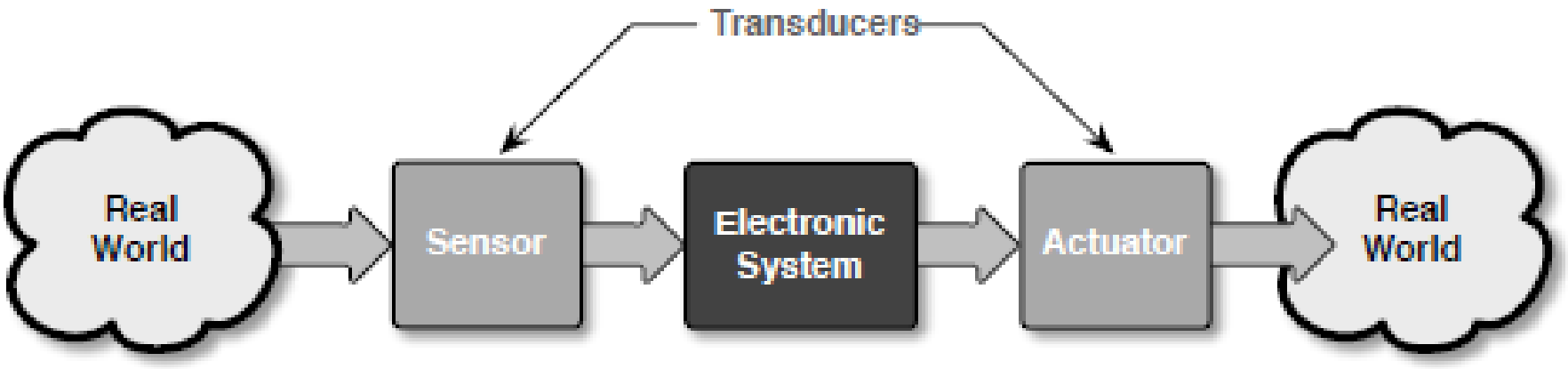
Examples of real-time systems where hardware choice is crucial

- Automotive
- Aerospace
- Military
- High Frenquency Trading
- Audio / Video / Broadcast
- Embedded Vision
- Healthcare / Medical
- Industrial IoT
- 5G / Wireless Communications
- Wired Communications
- Data Center Management Applications
- …
- Raw data selection and acquistion (Trigger and DAQ) for Physics Experiments HEP, Space, Nuclear, …

Hardware for Real Time / On-Line Data Management and Processing

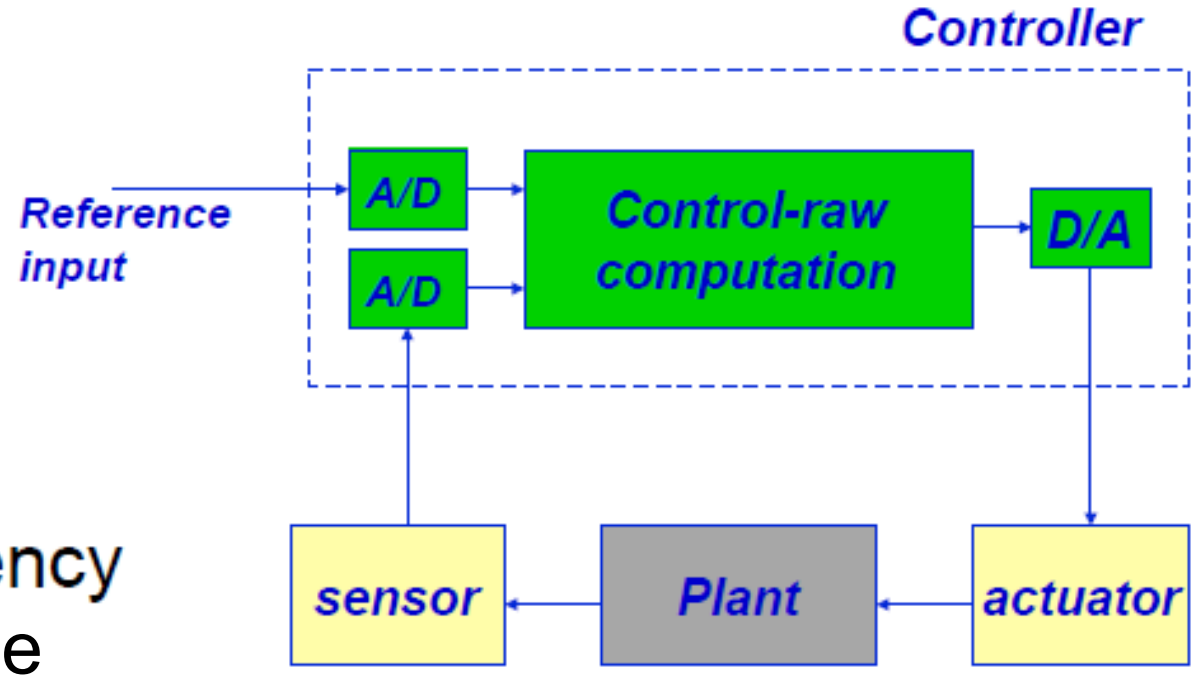
- Digital Systems and Data Path
- Hardware for Data Processing

Entire system involving signals of real world

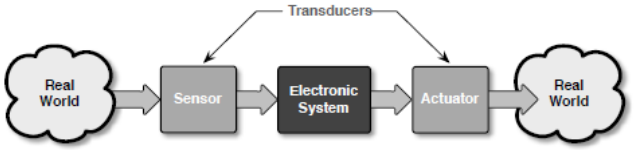


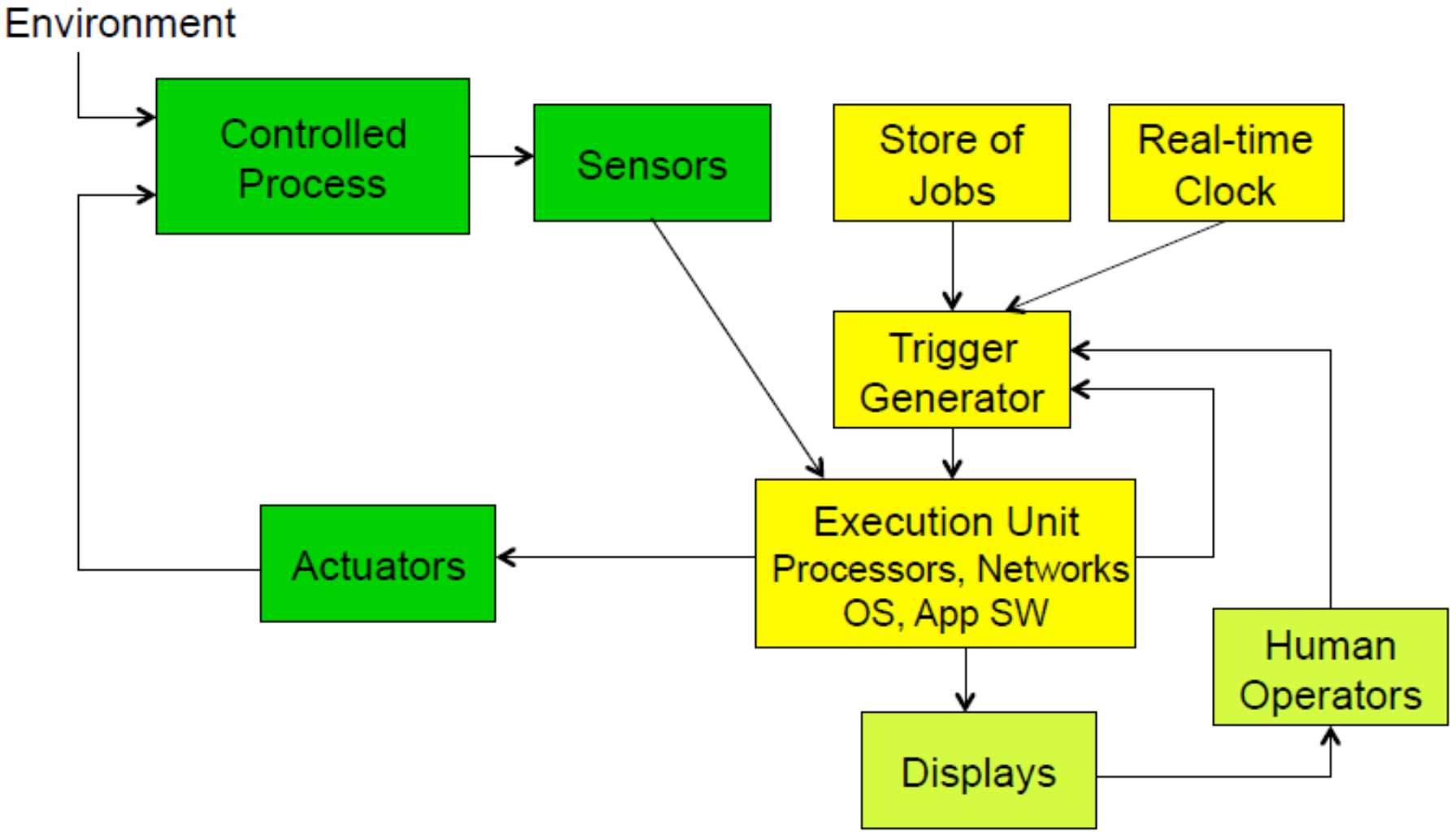
Real-time system

- needs timely computation
- deadlines, jitters, periodicity
- temporal dependency of system response

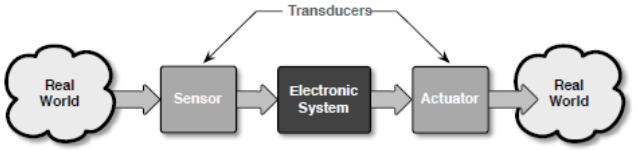


Real Time systems



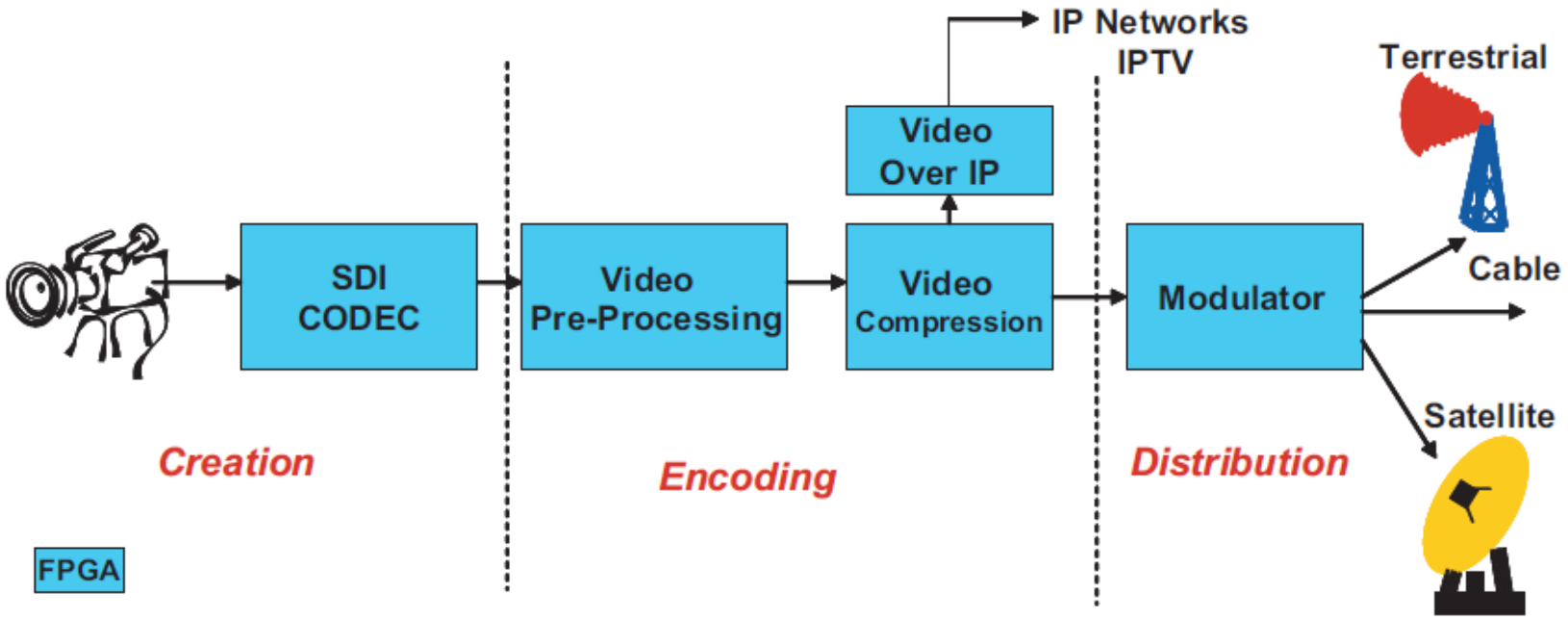


Audio/Video systems



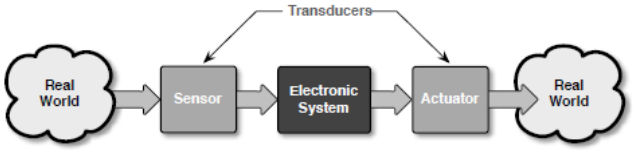
Video and Image Processing System Architectures

System architecture choices include standard cell ASICs, ASSPs, and programmable solutions such as digital signal processing (DSP) or media processors and FPGAs. Each of the approaches has advantages and disadvantages, with the ultimate choice depending on end equipment requirements and solution availability. Given the trends discussed above, the ideal architecture would have the following characteristics: high performance, flexibility, easy upgradability, low development cost, and a migration path to lower cost as the application matures and volume ramps.

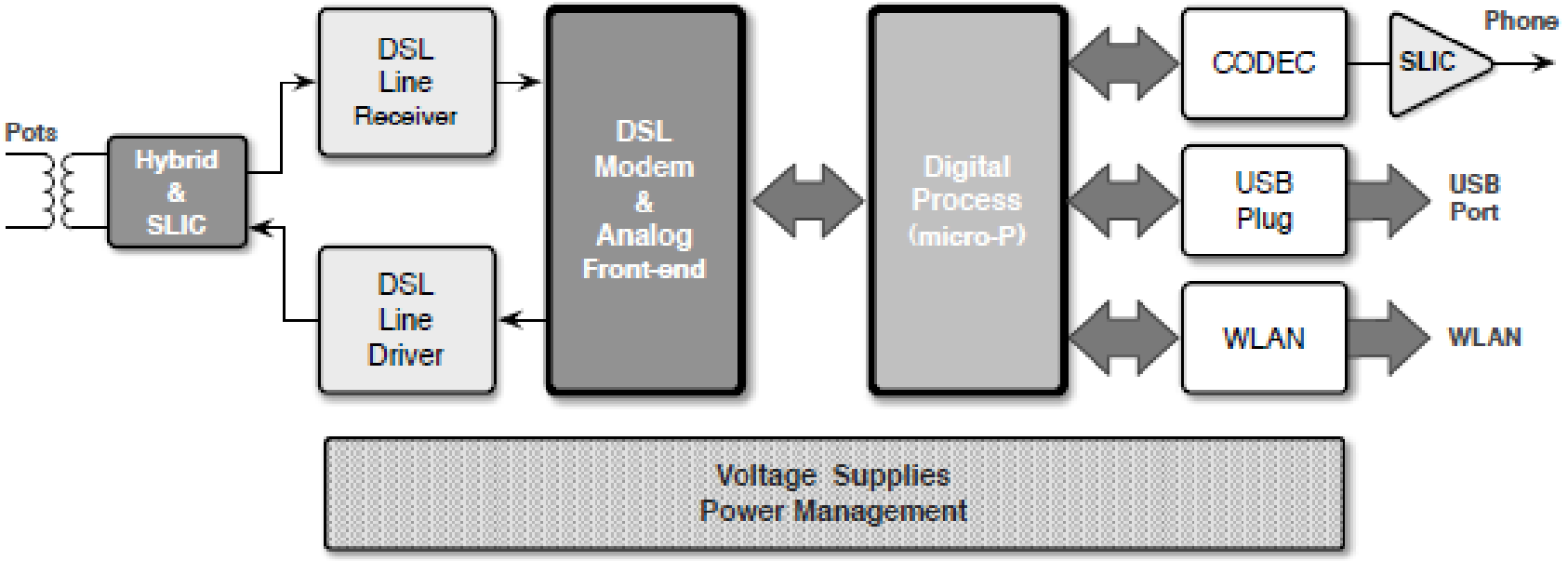


A broadcast system based on Field Programmable Gate Arrays (FPGAs)
→ constraints → … + timing and bandwidth (→ Latency and Throughput…)

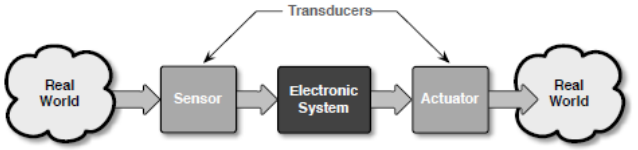
Wired Communication



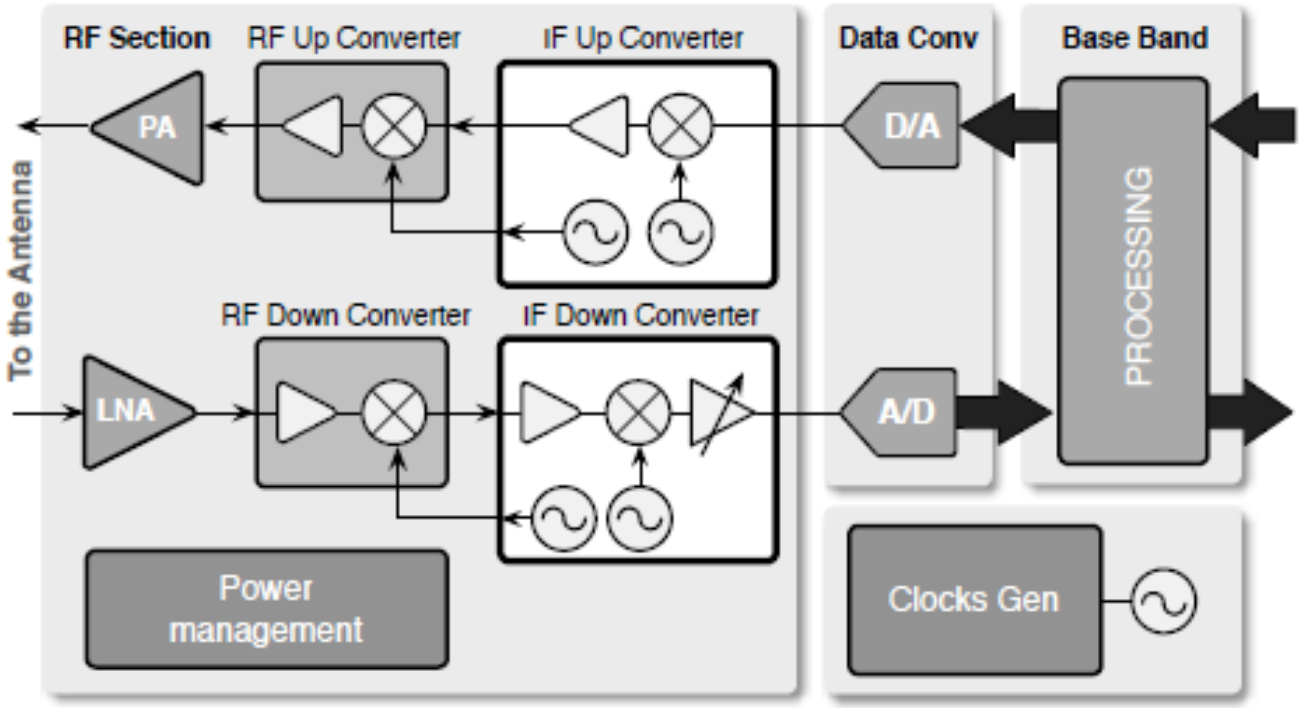
- Public Switched Telephone Network (*PSTN*).
- Digital Subscriber Line (*X-DSL*).



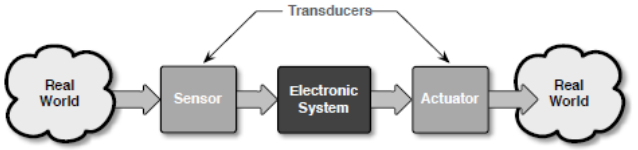
Wireless Communication



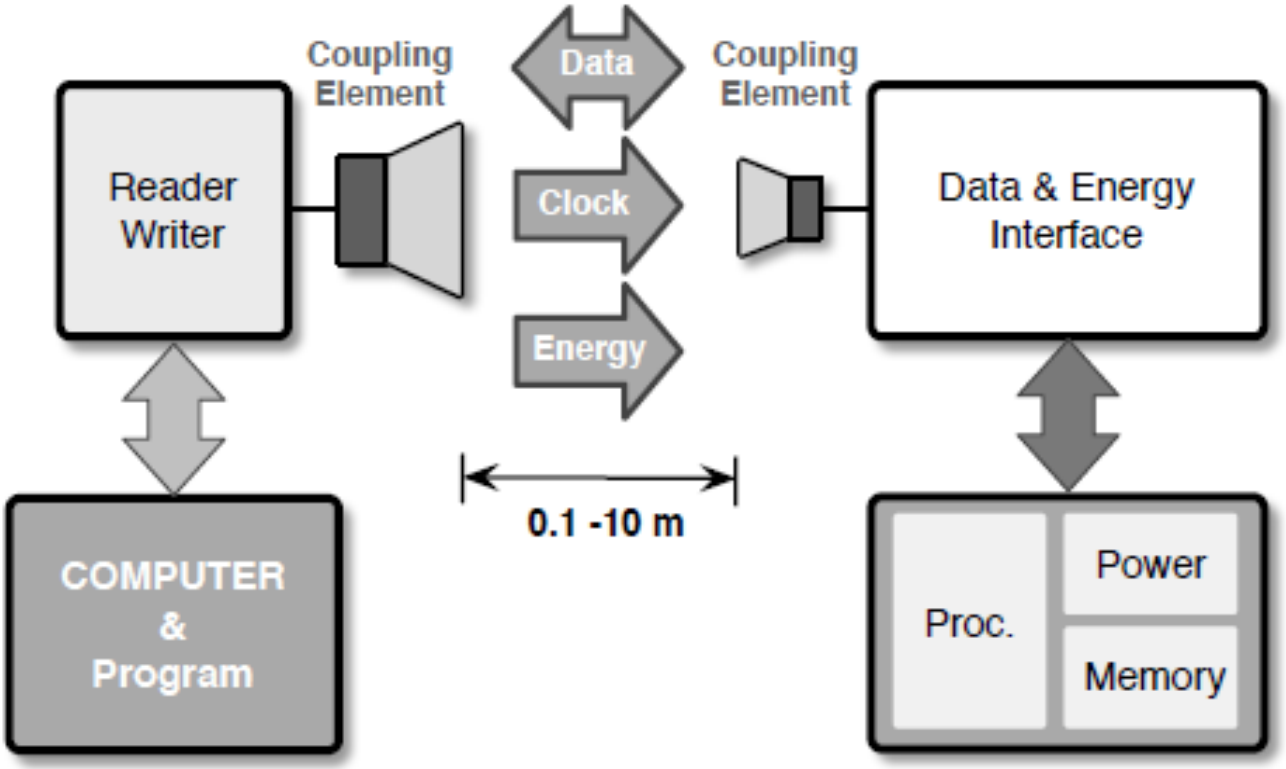
- *WiMAX* provides Broadband Wireless Access (*BWA*) up to 3–10 *mile* (5–15 *km*) for mobile stations.



Wireless Communication



- *RFID* stays for Radio Frequency IDentification.



Automotive systems

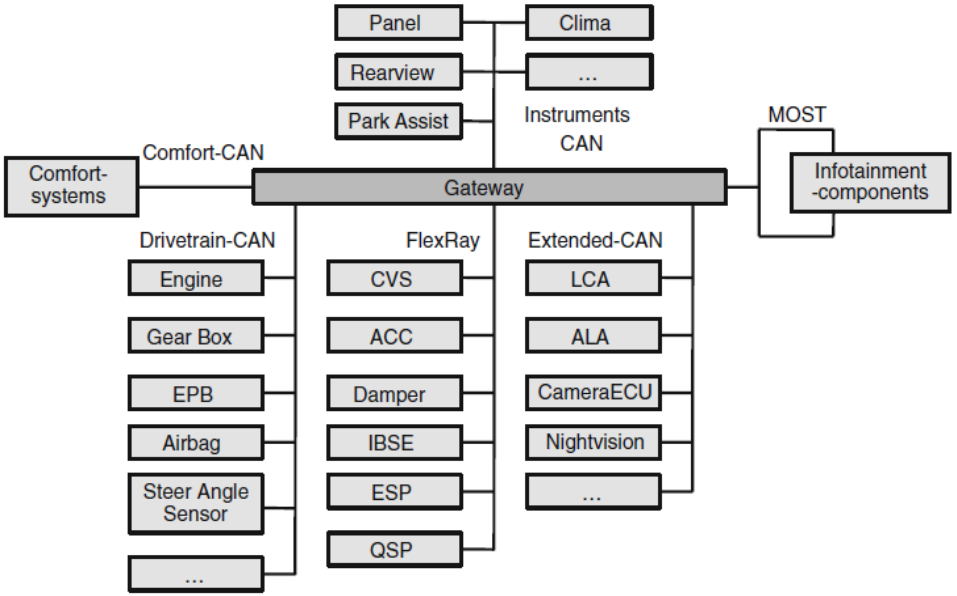
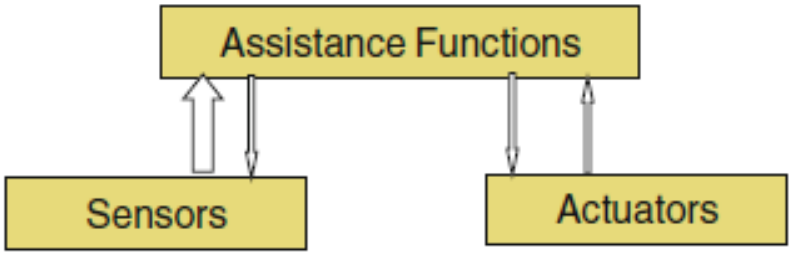
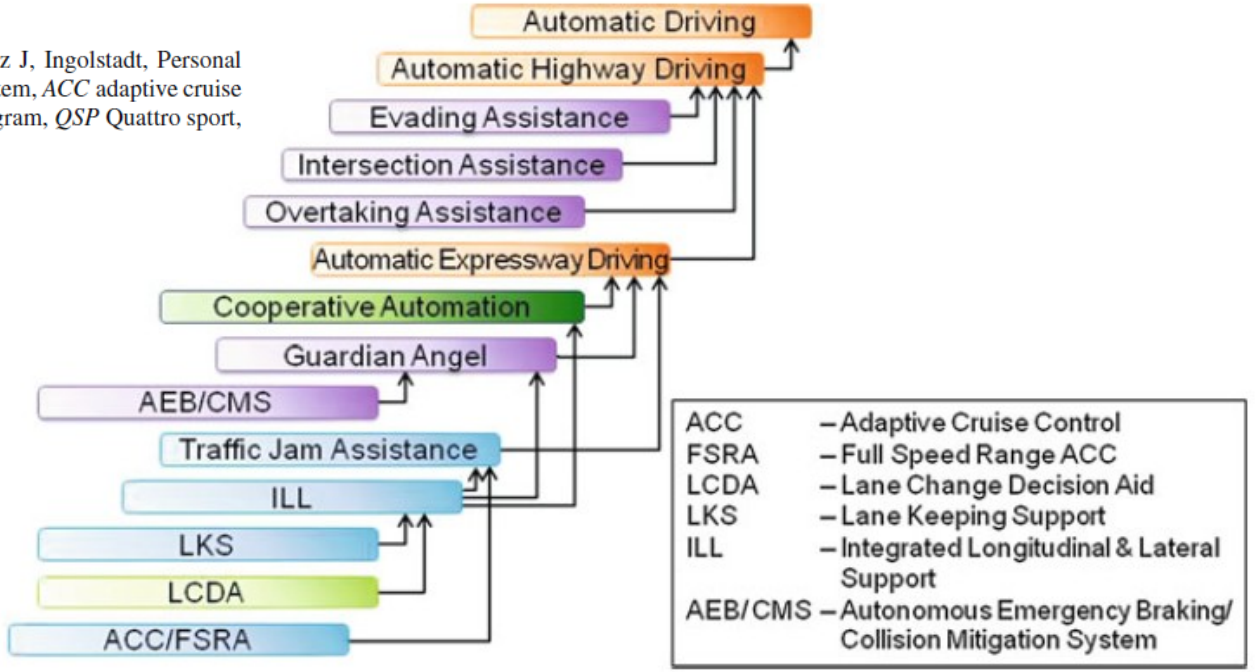


Fig. 2.8 Electronic hardware architecture of the current Audi A8 (Kötz J, Ingolstadt, Personal Communication). *EPB* electrical parking brake, *CVS* computer vision system, *ACC* adaptive cruise control, *IBSE* inertial based state estimation, *ESP* electronic stability program, *QSP* Quattro sport, *LCA* lane change assist, *ALA* adaptive light assist

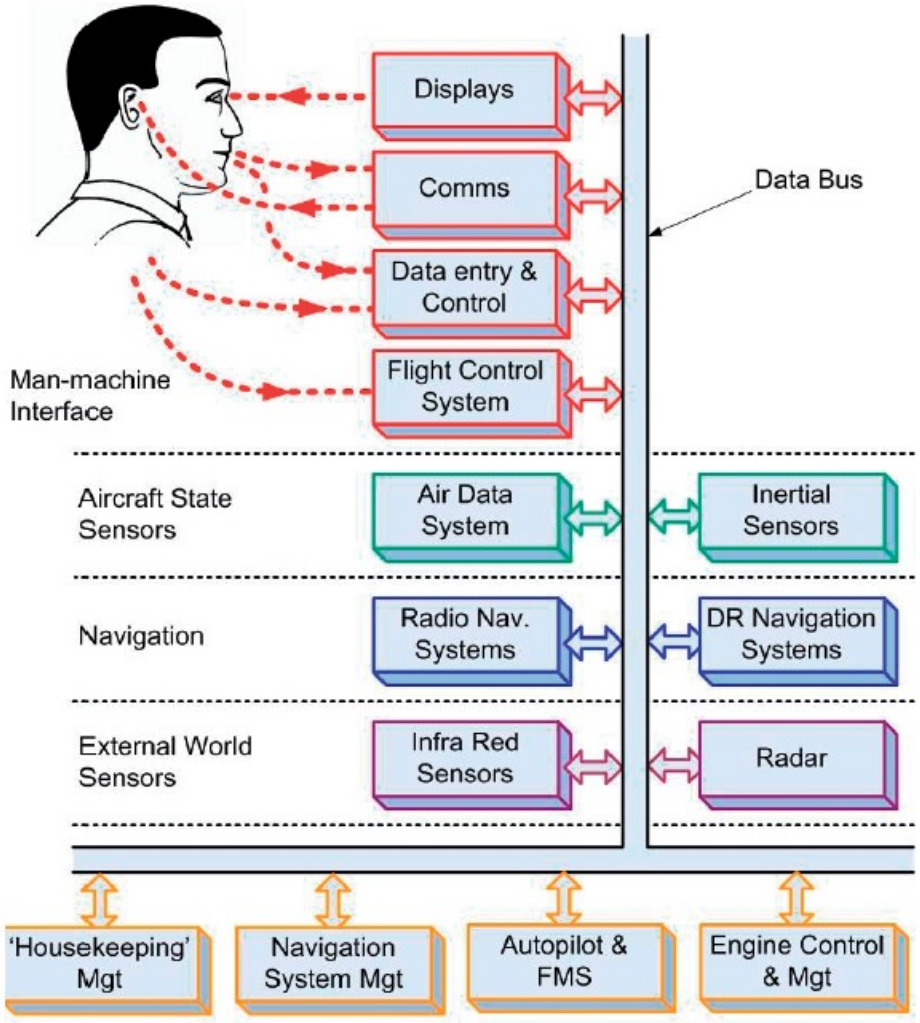


- Complex systems
- Timely responses
- Reliable components



Aerospace/Avionics systems

- Complex systems
- Timely responses
- Reliable components
- also against radiation



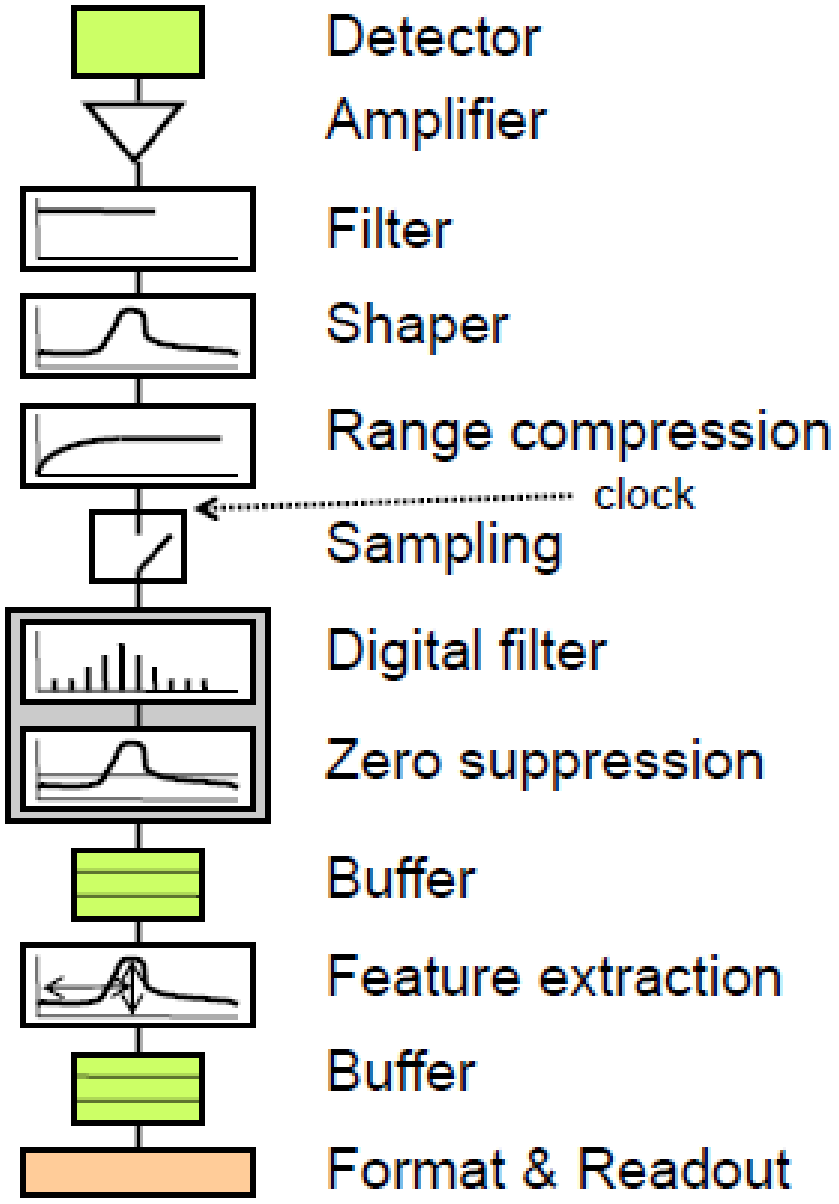
XILINX

WP402 (v1.0.1) March 7, 2012

Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors

By: Dagan White

Single event effects (SEEs) are of a growing concern in high-reliability system development, yet there is much disparity among users of ASICs and FPGAs with regard to understanding how susceptible their designs might be. The avionics and industrial system development guidance that currently exists is only broadly beginning to consider SEEs and their impact on system reliability. Unfortunately, standards such as DO-254, DO-178, ARP 4754, ARP 4761, and IEC 61508 provide little or no guidance on how to handle SEEs. This white paper highlights concerns regarding effects of SEEs on ASICs and FPGAs and points to analysis and mitigation techniques for handling SEEs.

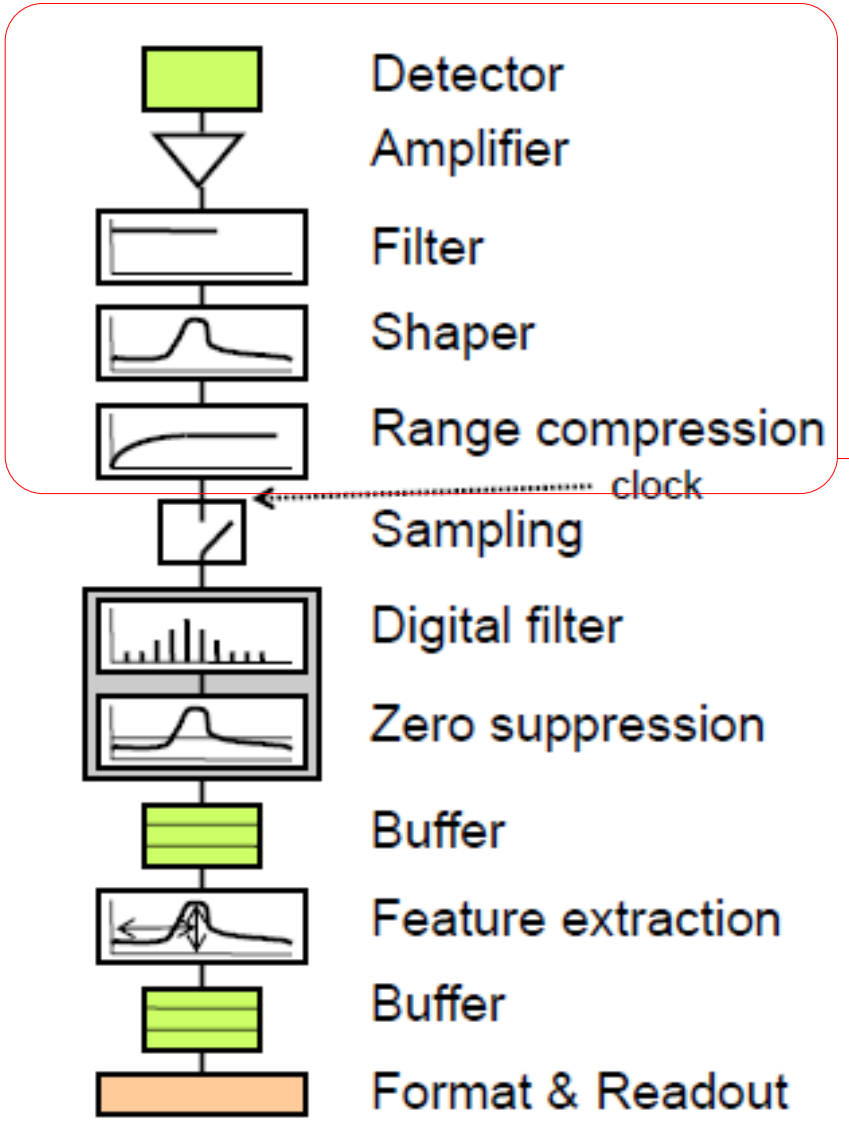


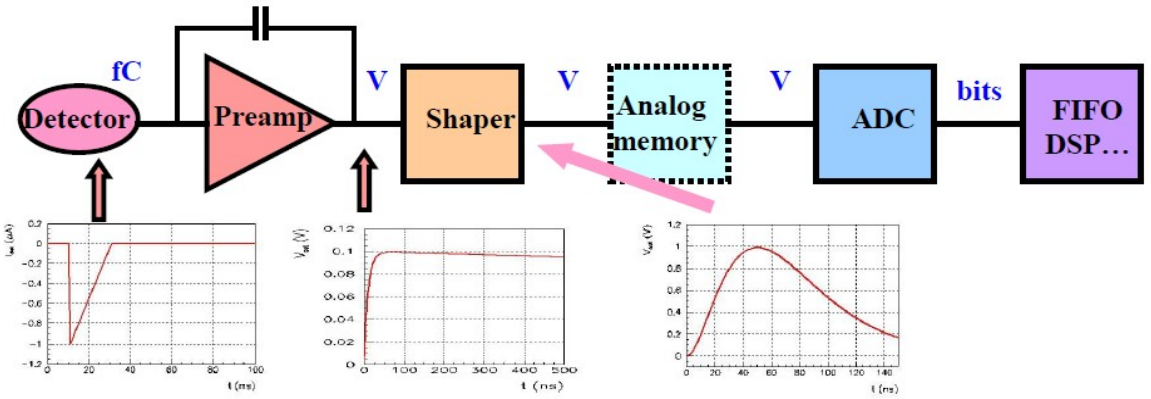
Specific system common to several experiments in Physics

- HEP
- Nuclear Phys
- AstroParticle,
- Nuclear Medicine

...

Trigger and Readout chain - FE

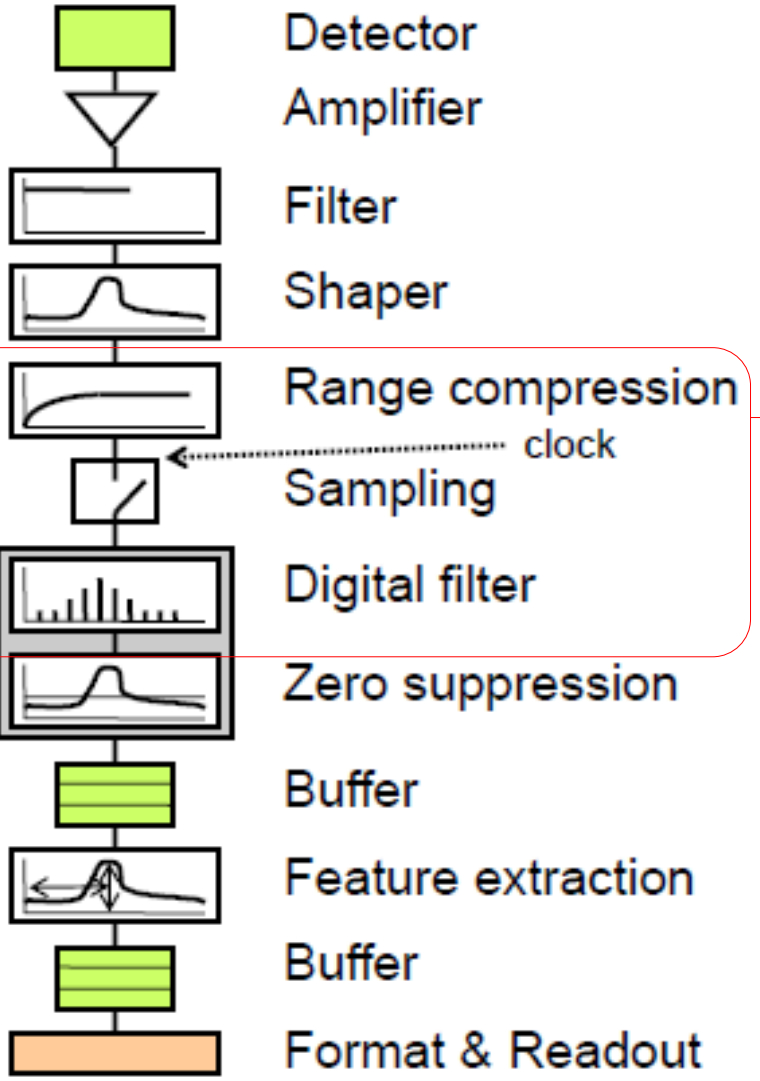


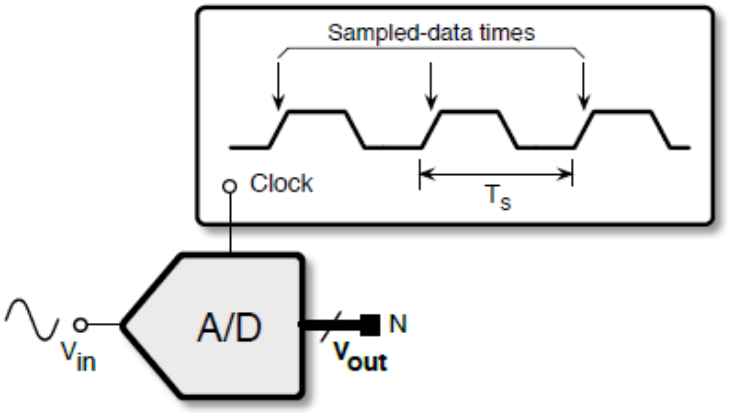


Front-End Chain

- Very small electrical signals O(fC)
  → need Amplification
- Filtering Signal, Rejecting noise
  → need shaping / filters
- Measure Amplitude and/or Timing
  → analog signal processing (linear/not linear)
  → then Analog to Digital (ADC, discrim., TDC)
- Many channels, up to $O(10^6)$
  → large scale electronics integration

Trigger and Readout chain - A/D

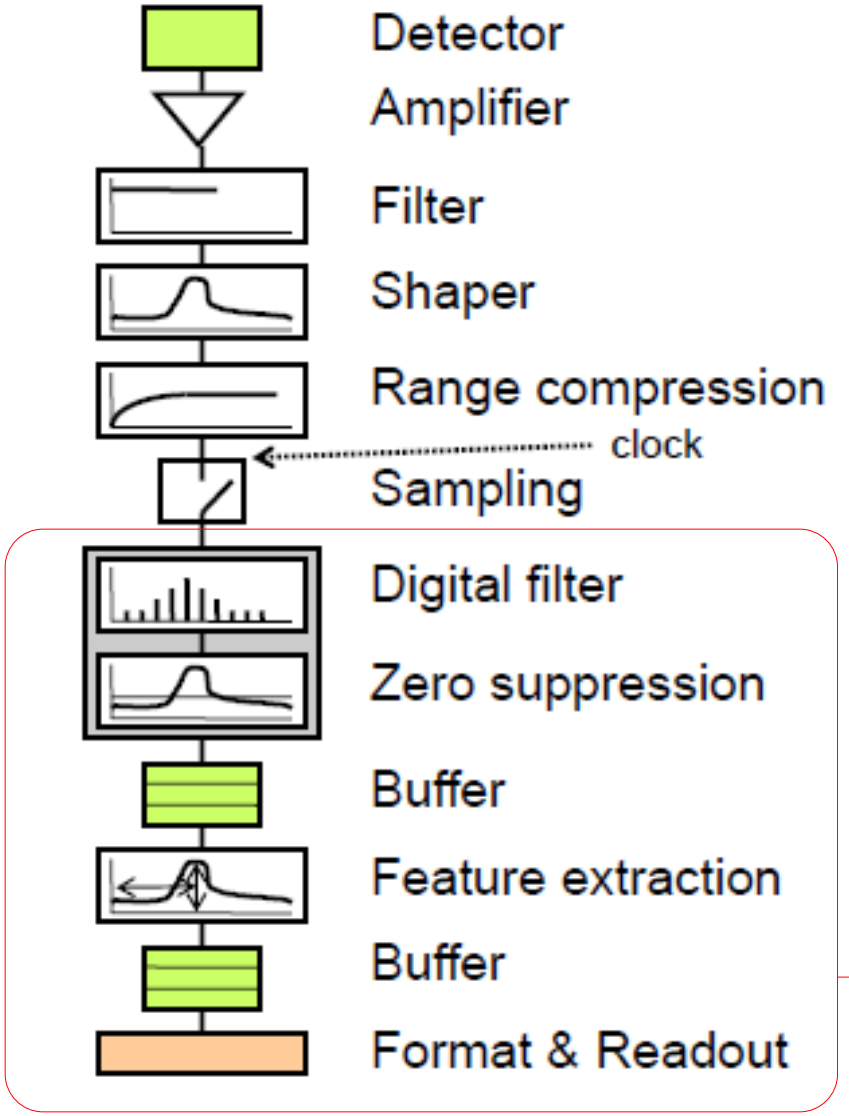




A/D conversion

- From Analog to Digital Domain
- Clock is needed
- Loss of information and Approximate Signal Reconstruction, Nyquist sampling
- Measure Amplitude and/or Timing
  → ADC / TDC, waveform sampling

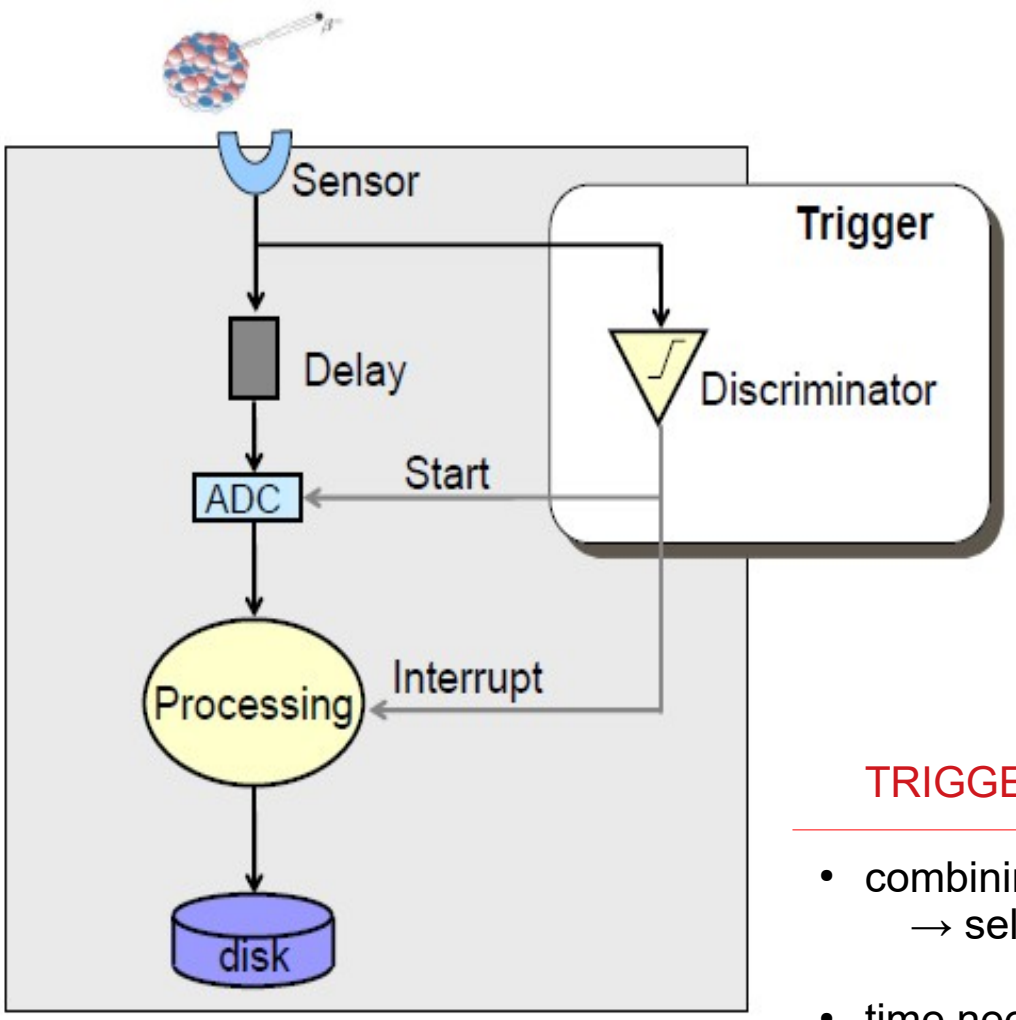
Trigger and Readout chain – DSP



- Filtering Signal, Rejecting noise
  → single channel S/N optimizatio
  → all channels: TRIGGER for selecting
    - samples of interesting events for Physics
    - samples for calibration and TRIGGER characterization
- Signal Compression and Formatting for transmission
- Buffering for absorbing fluctuations (Derandomizing)
- Decompression and Feature extraction

Digital Signal Processing

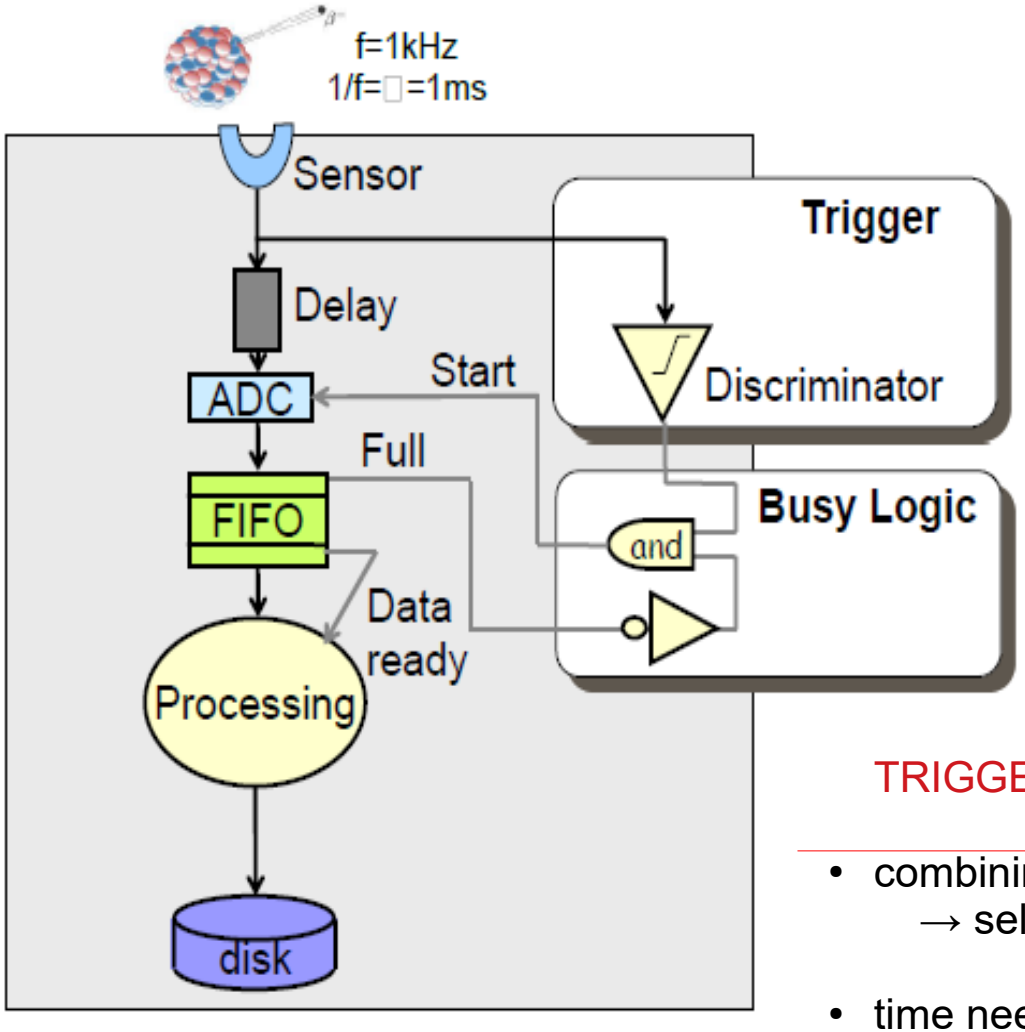
Trigger and Readout chain – Trigger



TRIGGER

- combining digitized information of many channels
  → selecting interesting events (eg against background)
- time needed for decision (latency)
  → information delayed, waiting for decision
- time needed for readout (dead-time)
  → trigger inhibited during dead-time (busy logic)
- bandwidth of DAQ transport channel (throughput)

Trigger and Readout chain – Trigger

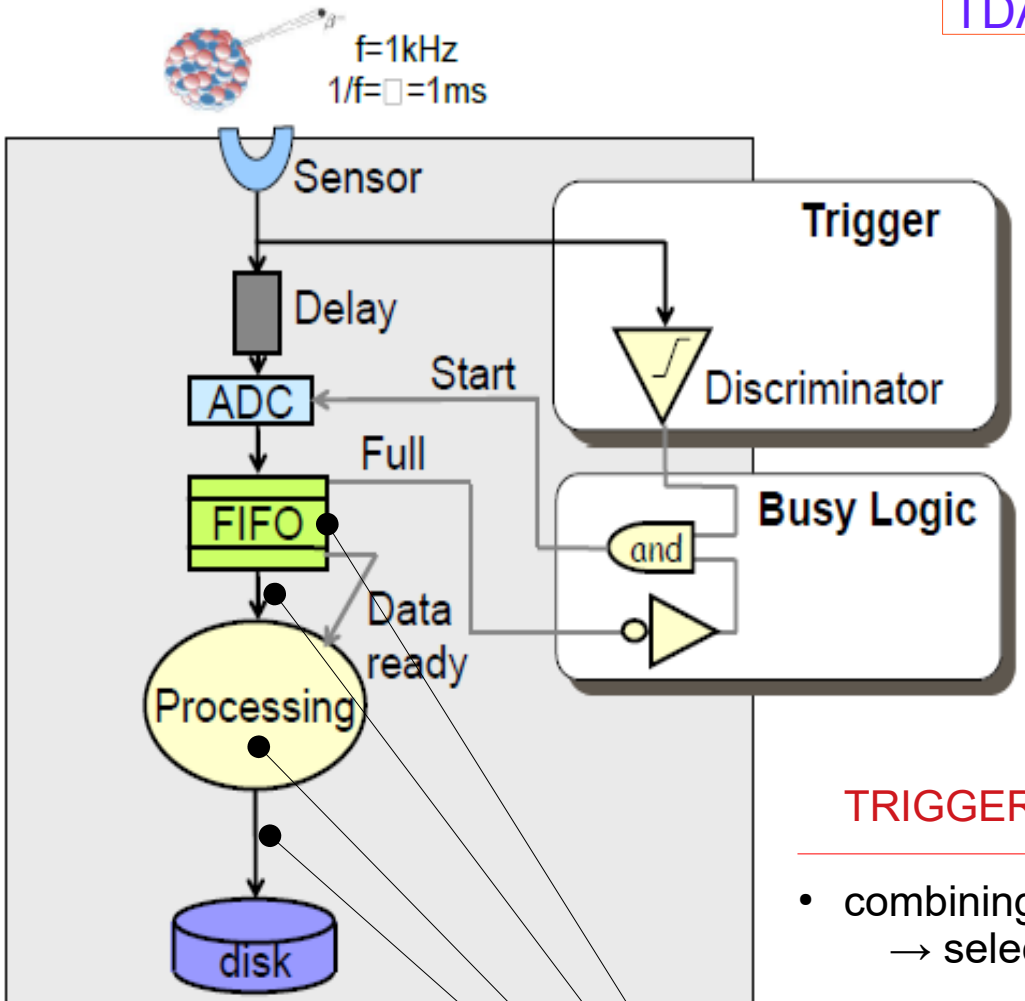


TRIGGER

- combining digitized information of many channels
  → selecting interesting events (eg against background)
- time needed for decision (latency)
  → information delayed, waiting for decision
- time needed for readout (dead-time)
  → trigger inhibited during dead-time (busy logic)

Trigger and Readout chain – Trigger-DAQ

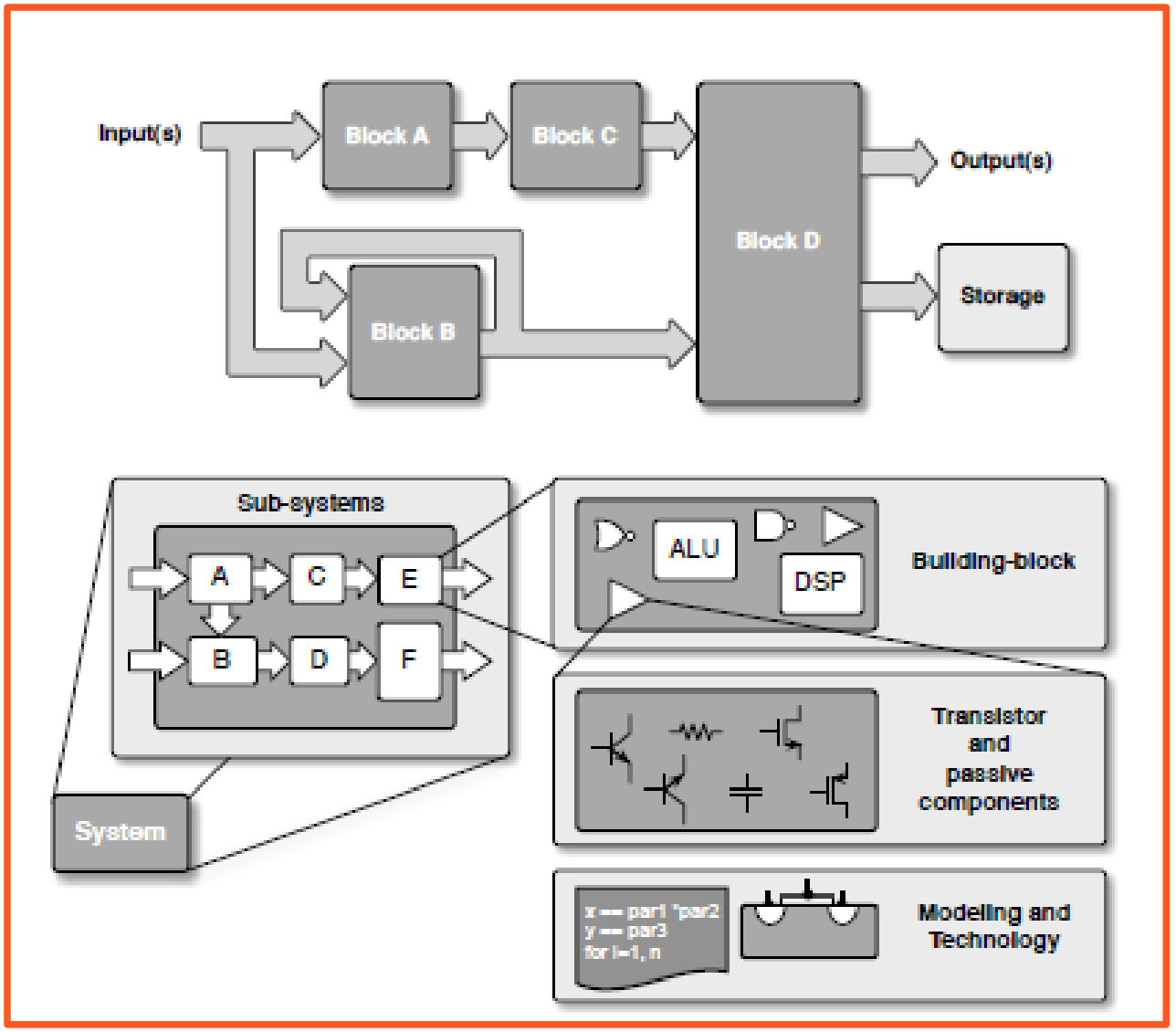
TDAQ - our example of reference for MAPD1 & 2
(see later CPU vs GPU's this lecture
see more details in lectures by prof. Meschi)



TRIGGER and DAQ Hardware → impact in data flow

- combining digitized information of many channels
  → selecting interesting events (eg against background)
- time needed for decision (latency)
  → information delayed, waiting for decision
- time needed for readout (dead-time)
  → trigger inhibited during dead-time (busy logic) (throughput)
- capacitance per unit time (bandwidth) of data transport channels and of real time data processing (→ throughput)

Analog & Digital systems

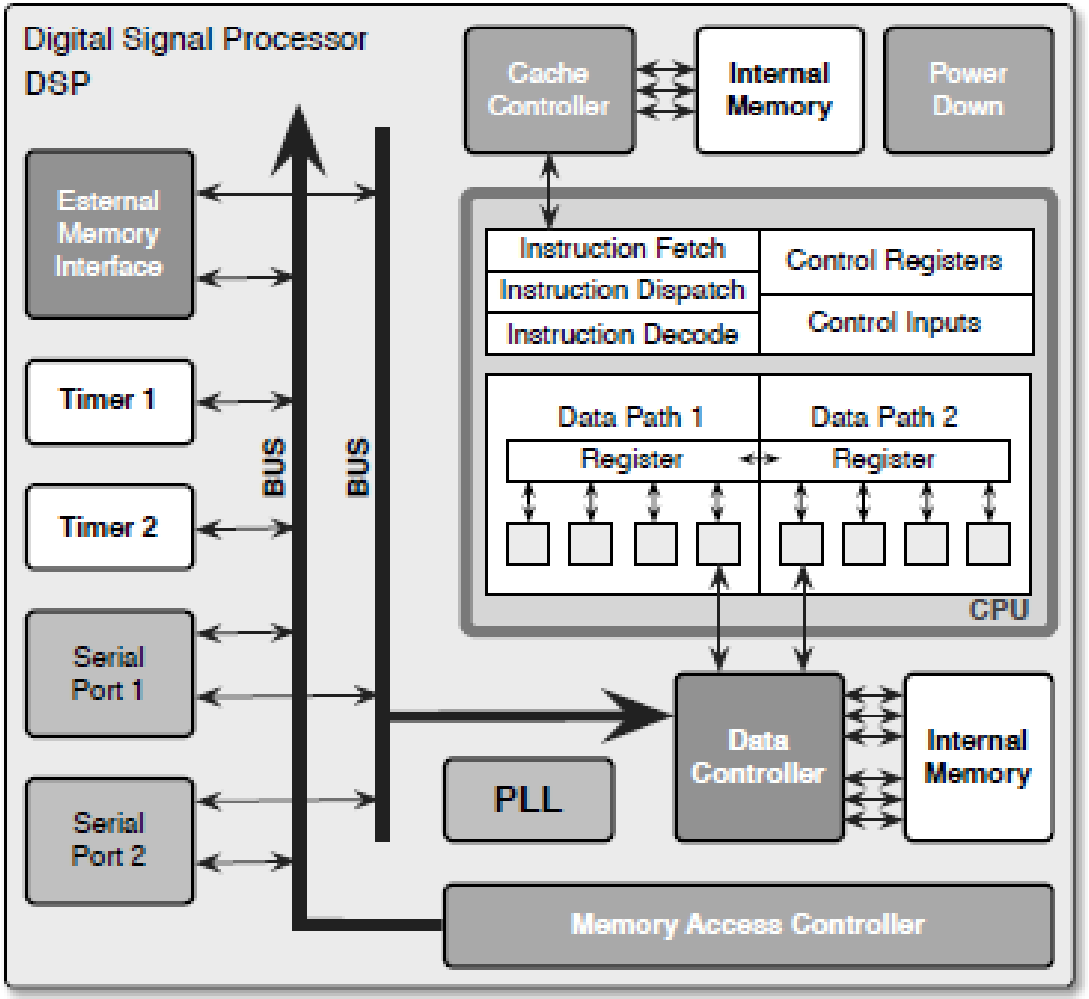


1) Analog building blocks
→ moving analog signals
→ buffering analog signals
→ processing (amplify, shape, …)
Analog signal vs noise

2) A/D & D/A conversion blocks
→ conversion of signal
from Analog to Digital

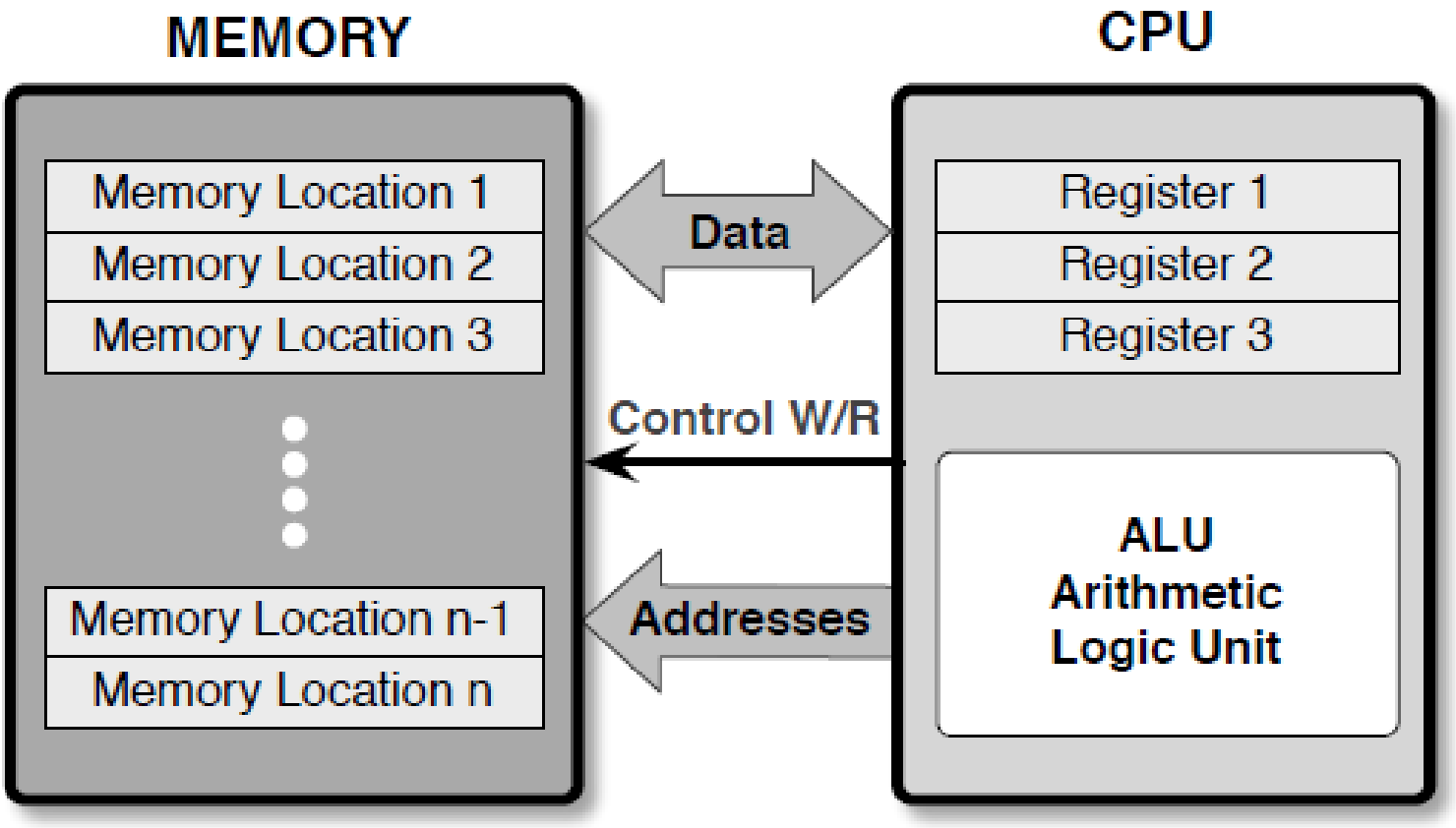
3) Digital blocks
→ moving digital data
→ buffering digital data
→ processing digital data
(signal vs noise)

Example Digital Systems
→ Digital Signal Processor (DSP)



Digital blocks

Example Digital Systems
→ Microprocessor



Digital blocks

Hardware for Real Time / On-Line Data Management and Processing

- Digital Systems and Data Path
- Hardware for Data Processing

Real Time MAPD → Hardware point of view

Three latest development boosted HW data processing

- Powerful distributed **processors**
- Cheap & fast on-board **memories** (“data buffers“) and cheap & high volume storage
- High bandwidth **interconnection** (“fast links“) to move data to/from the processors

Noticeably HW data processing boost on turn determined the recent “Machine Learning new era“

Real Time MAPD → Hardware point of view

Three latest development boosted HW data processing

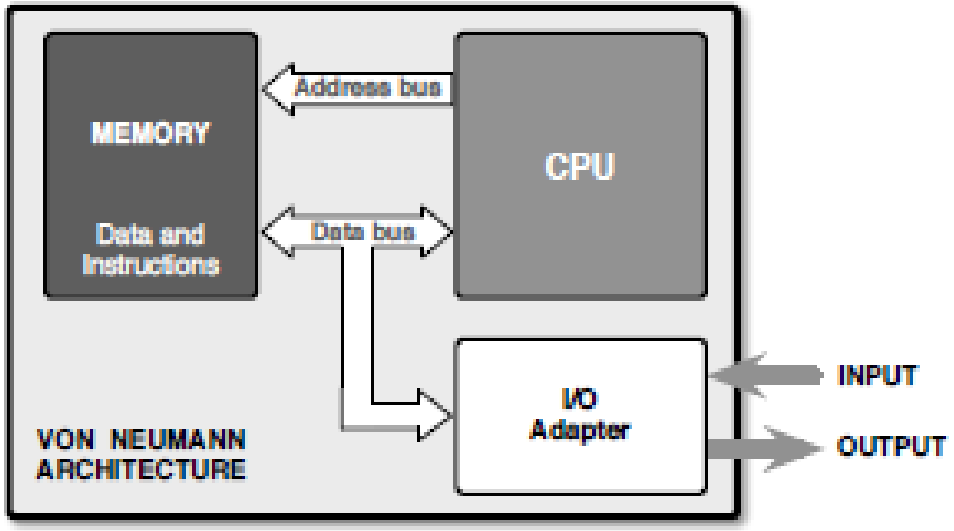
- Powerful distributed **processors**

→ use of ASICs HW chips (CPUs, GPUs)
and Programmable HW chips (FPGAs)
for processing data

1. Processors
→ Digital architectures with memories

Type of information stored in a memory: data or instructions.

Data are pure information. Instruction indicates the manner in which electronic circuits process data.



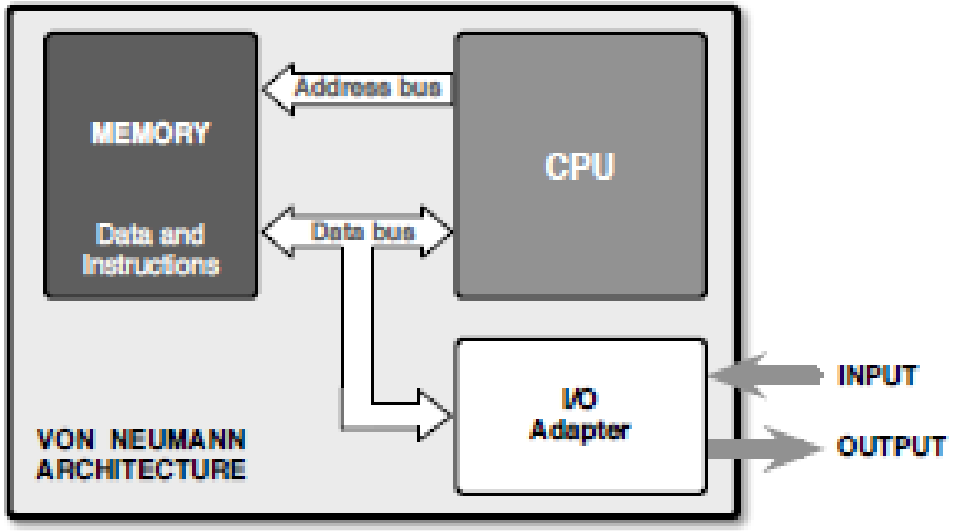
Von Neumann Architecture

To multiply two numbers it is necessary to fetch the two numbers and instructions for the multiplication procedure. Numbers enters in the Arithmetic Logic Unit (*ALU* that performs the multiplication. Then the result is possibly sent back to the memory for future use.

1. Processors
→ Digital architectures with memories

Type of information stored in a memory: data or instructions.

Data are pure information. Instruction indicates the manner in which electronic circuits process data.



Von Neumann Architecture

Note: modern CPU adopt hybrid von Neumann/Harvard Architecture: data and instructions share the same common memory but they are buffered into separate memory caches which the CPU can access indipendendly (see Harvard Architecture, later this lecture)

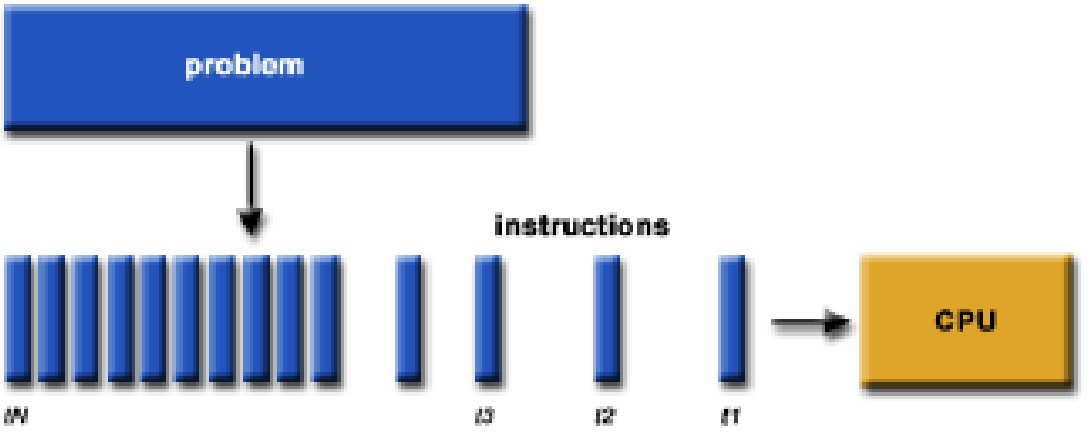
1. Processors
→ Digital architectures with memories

Von Neumann architecture: instructions are sent from memory to the CPU

Serial execution: Instructions are executed one after another on a single Central Processing Unit (CPU)

Problems:

- More expensive to produce
- More expensive to run
- Bus speed limitation



1. Processors
→ Parallel architectures

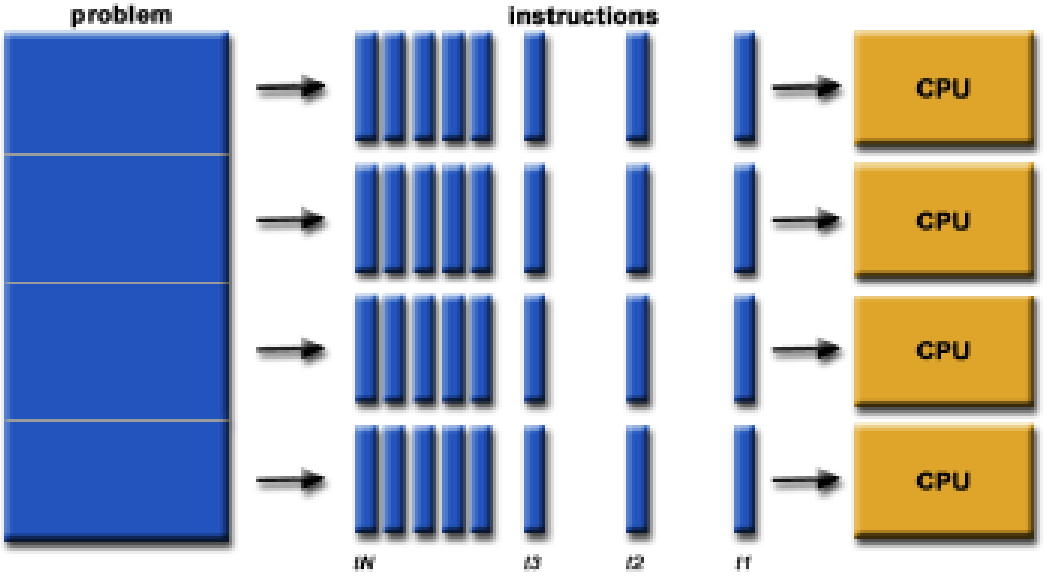
Official-sounding definition: The simultaneous use of multiple compute resources to solve a computational problem.

Benefits:

- Economical – requires less power and cheaper to produce
- Better performance – bus/bottleneck issue

Limitations:

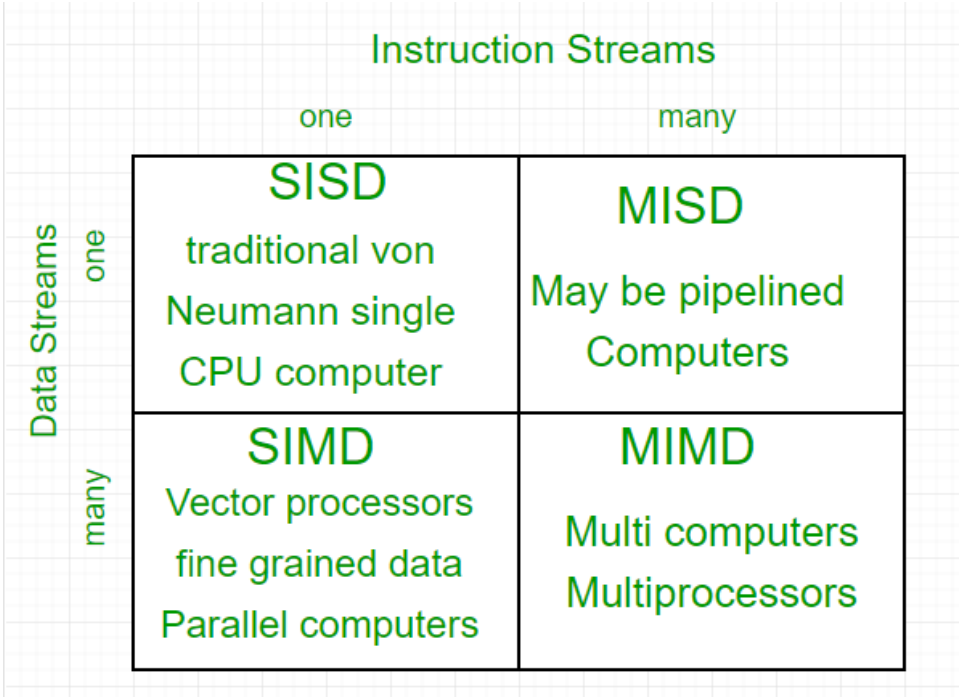
- New architecture – Von Neumann is all we know!
- New debugging difficulties – cache consistency issue

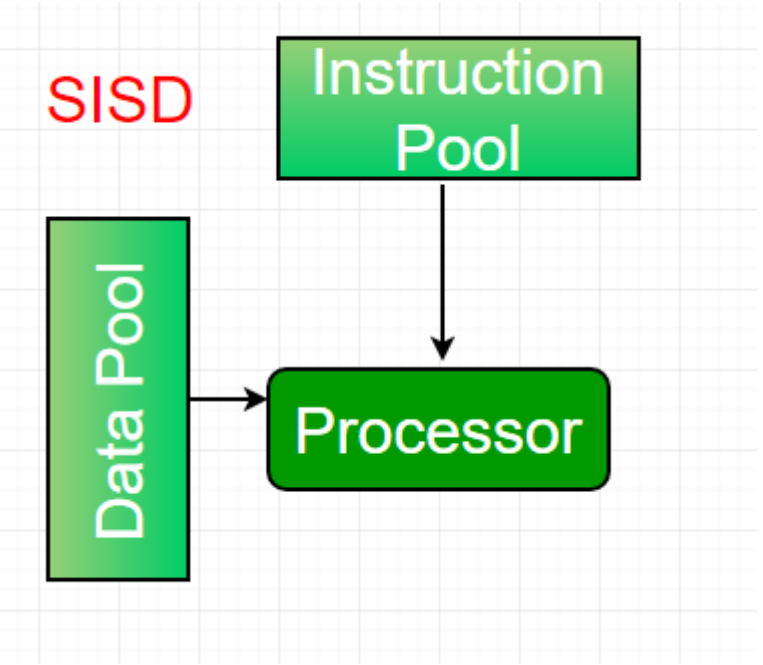


1. Processors → Flynn‘s taxonomy

Classification of computer architectures, proposed by Michael J. Flynn

- SISD – traditional serial architecture in computers. (single instuction / single datum)
- SIMD – parallel computer. One instruction is executed many times with different data (think of a for loop indexing through an array) (single instuction / multiple data)
- MISD - Each processing unit operates on the data independently via independent instruction streams. Not really used in parallel
- MIMD – Fully parallel and the most common form of parallel computing. (multiple instuctions / multiple data)



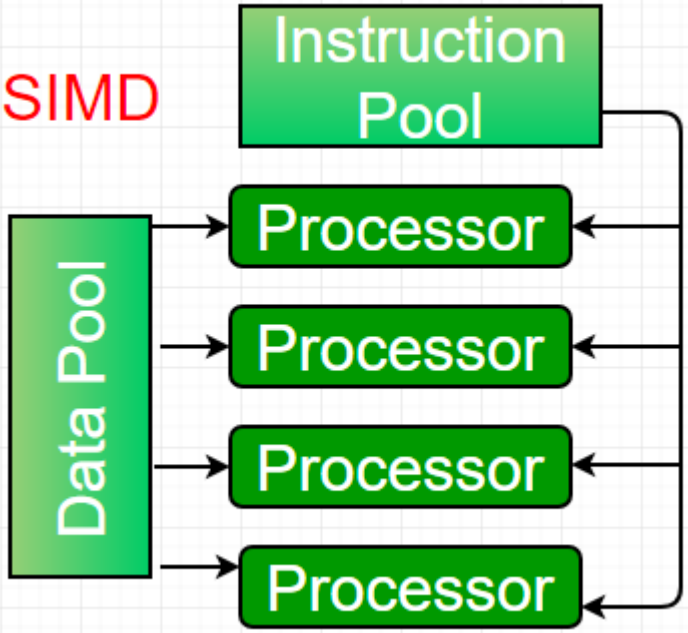


1. Processors → Flynn‘s taxonomy

Classification of computer architectures, proposed by Michael J. Flynn

- SISD – traditional serial architecture in computers.
- SIMD – parallel computer. One instruction is executed many times with different data (think of a for loop indexing through an array)
- MISD - Each processing unit operates on the data independently via independent instruction streams. Not really used in parallel
- MIMD – Fully parallel and the most common form of parallel computing.

Data Streams \ Instruction Streams	one	many
one	**SISD** traditional von Neumann single CPU computer	**MISD** May be pipelined Computers
many	**SIMD** Vector processors fine grained data Parallel computers	**MIMD** Multi computers Multiprocessors

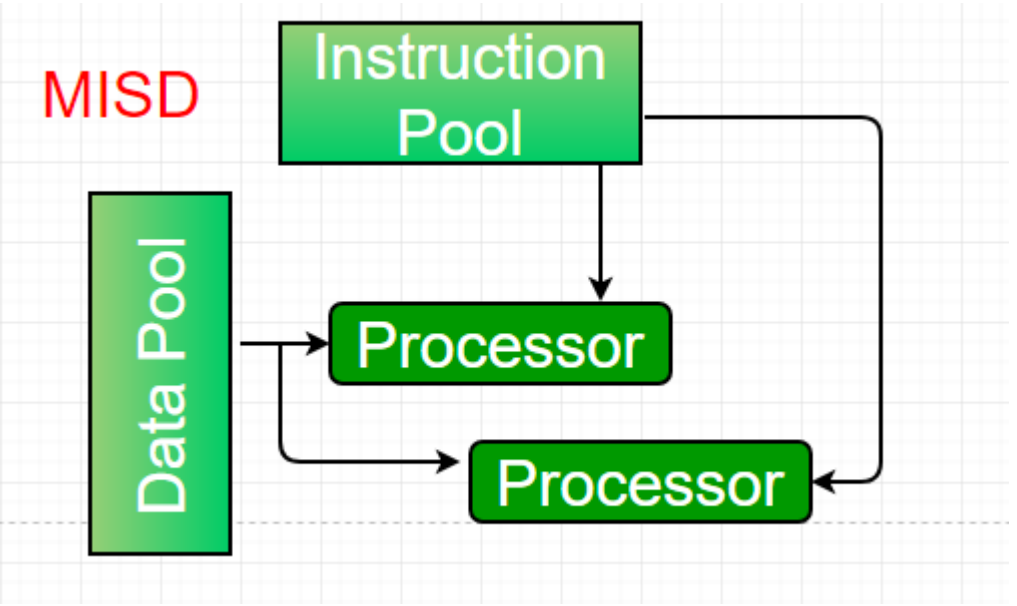


1. Processors → Flynn‘s taxonomy

Classification of computer architectures, proposed by Michael J. Flynn

- SISD – traditional serial architecture in computers.
- SIMD – parallel computer. One instruction is executed many times with different data (think of a for loop indexing through an array)
- MISD - Each processing unit operates on the data independently via independent instruction streams. Not really used in parallel
- MIMD – Fully parallel and the most common form of parallel computing.

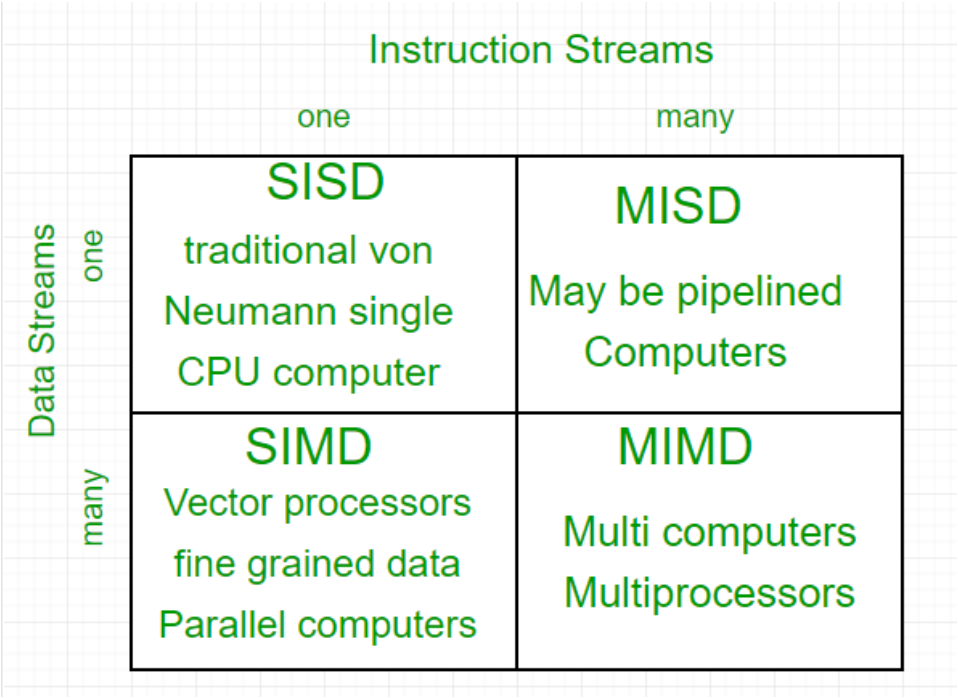
Data Streams \ Instruction Streams	one	many
one	**SISD** traditional von Neumann single CPU computer	**MISD** May be pipelined Computers
many	**SIMD** Vector processors fine grained data Parallel computers	**MIMD** Multi computers Multiprocessors

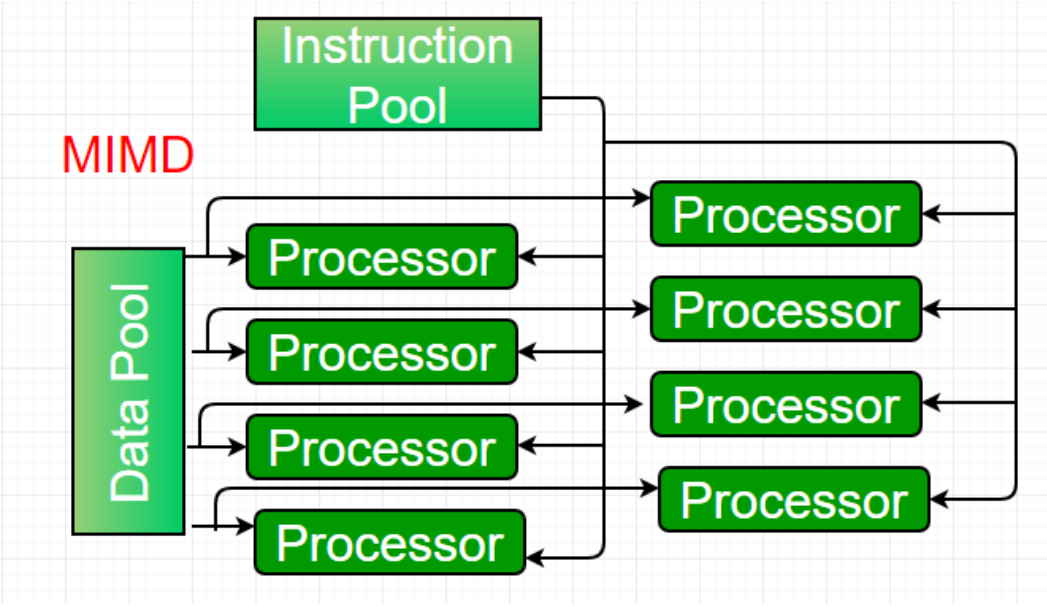


1. Processors → Flynn‘s taxonomy

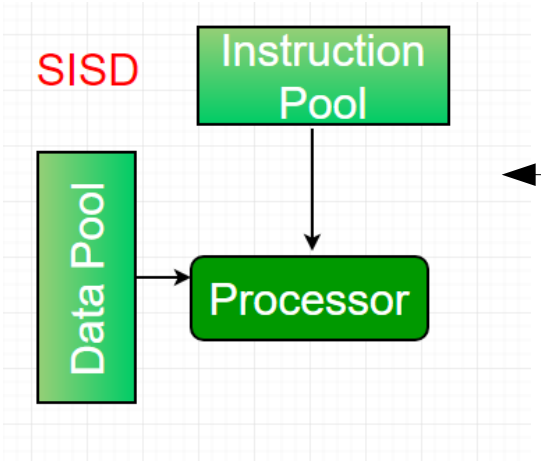
Classification of computer architectures, proposed by Michael J. Flynn

- SISD – traditional serial architecture in computers.
- SIMD – parallel computer. One instruction is executed many times with different data (think of a for loop indexing through an array)
- MISD - Each processing unit operates on the data independently via independent instruction streams. Not really used in parallel
- MIMD – Fully parallel and the most common form of parallel computing.

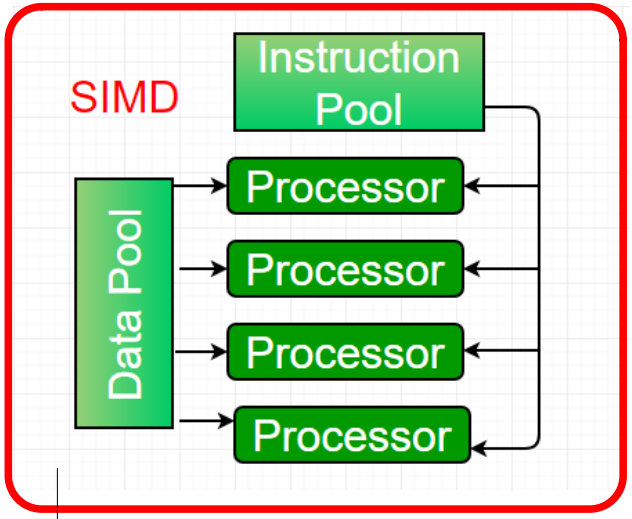


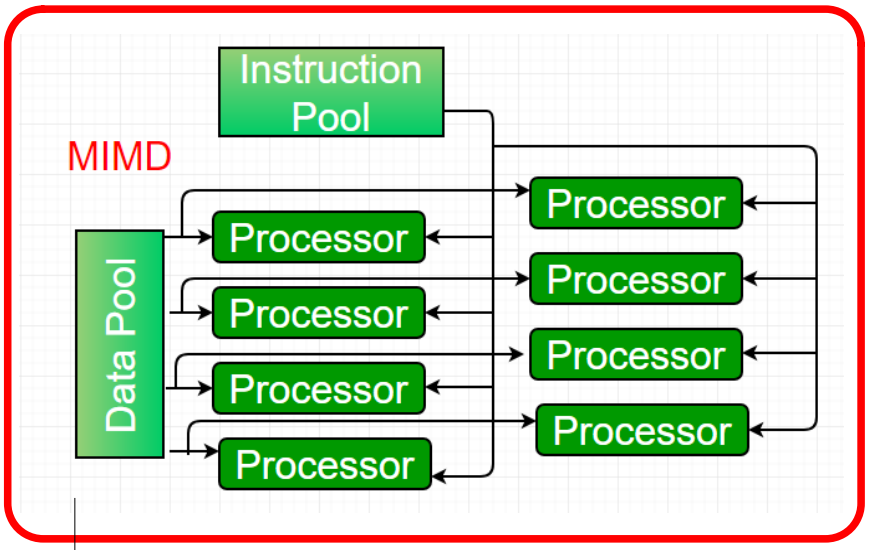


1. Processors → Flynn‘s taxonomy

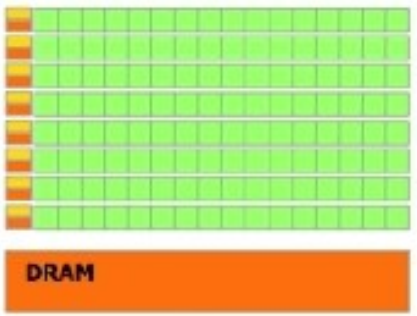


Single core
CPU





many-core
GPU

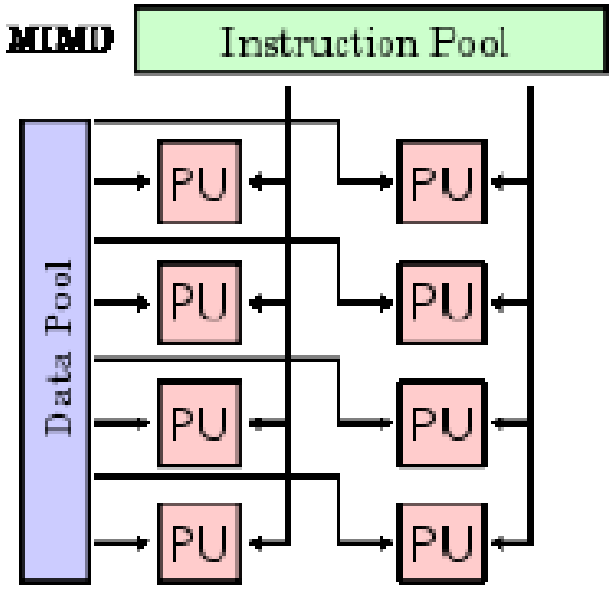


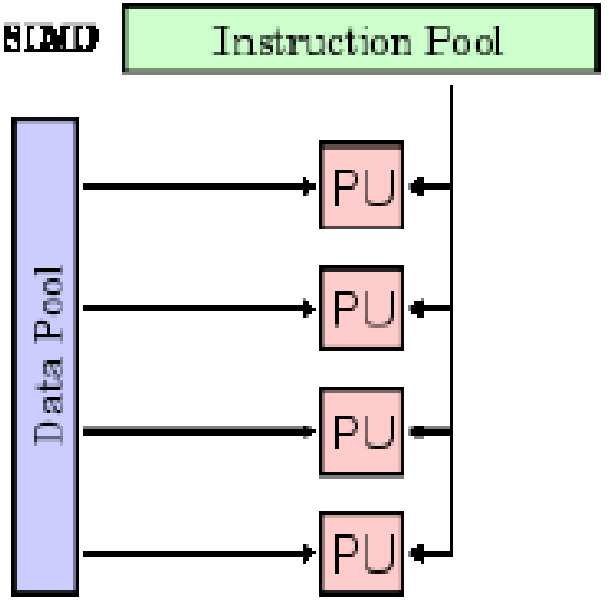
Modern
multi-core
CPU are partially MIMD

1. Processors → GPU vs CPU

CPU multiple cores
- MIMD (Multiple Instruction / Multiple Data)
- each core operates independently
- each can be working with a different code, performing different operations with entirely different data

GPU many cores
- SIMD (Single Instruction / Multiple Data)
- all cores executing the same instruction at the same time, but working on different data
- only one instruction de-coder needed to control all cores
- functions like a vector unit



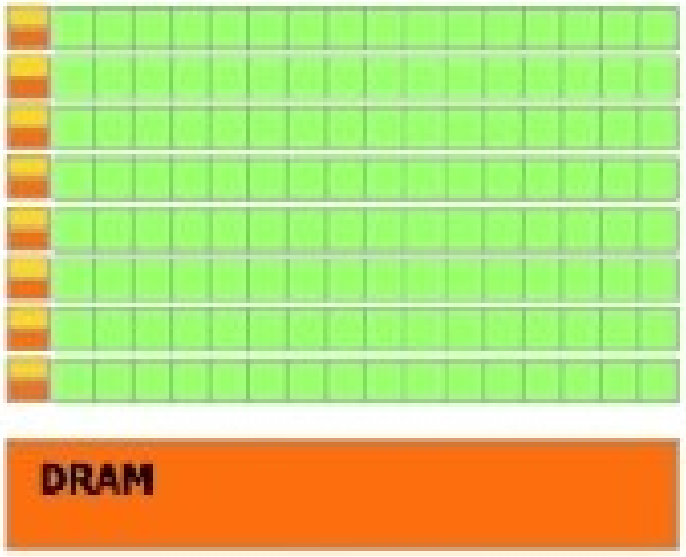


1. Processors → GPU structure

Graphics Processing Units (GPUs) have become faster and more efficient than CPUs in certain types of computations

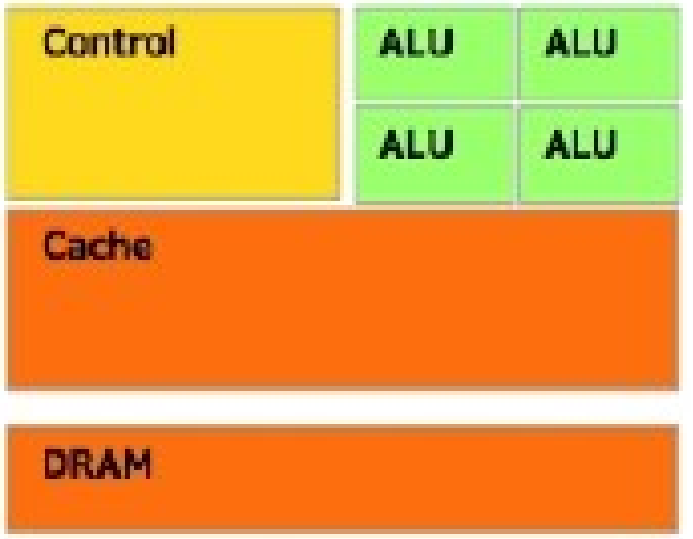
→ rendering real-time graphics: highly computational and memory intensive problem with enormous inherent parallelism

→ relative to a CPU, a much larger portion of a GPU's resources is devoted to data processing than to caching or control flow



GPU

CPU



1. Processors → GPU for matrix comput.

GPU's are able to do a lot more of parallel computations than a CPU can do
→ suited for **Vector and Matrix computation intensive tasks**

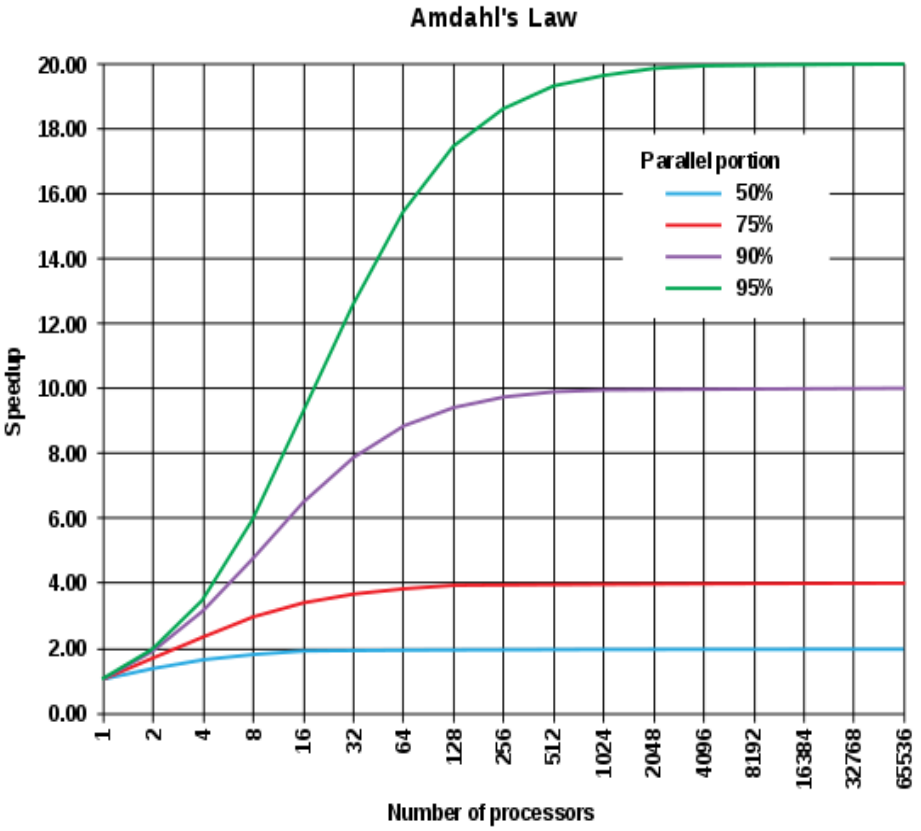
Using a GPU let's say you have 1000 maximum threads you can run
Example

`c[0]=a[0]+b[0]` // let's do it on thread 0
`c[1]=a[1]+b[1]` // let's do it on thread 1
...
`c[999]=a[990]+b[999]` // let's do it on thread 999
`c[1000]=a[1000]+b[1000]` // let's do it on thread 0
...

We are able to do it because value of c[0] doesn't depend upon any other values except a[0] and b[0] → each addition is independent of others
Task Parallelization → speed-up by factor of 1000 wrt fo serial loop (provided proper data)

`for(i=0;i<N;i++);c[i]=a[i]+b[i]`// serial loop… to be compiled

Parallel computing



- Several problems can be split in smaller problems to be solved concurrently

- In any case the maximum speed-up is not linear, but it depends on the serial part of the code

  → Amdahls's law

$$S_{latency} = \frac{1}{1 - p + \frac{p}{s}}$$

Importance of the hardware...

Where
- $S_{latency}$ = speedup in latency with respect to single thread execution
- p = time fraction of sigle thread that can be executed in parallel
- s = speedup factor due to use of improved resouces (eg number of threads)

1. Processors
→ highly parallelizable algorithms

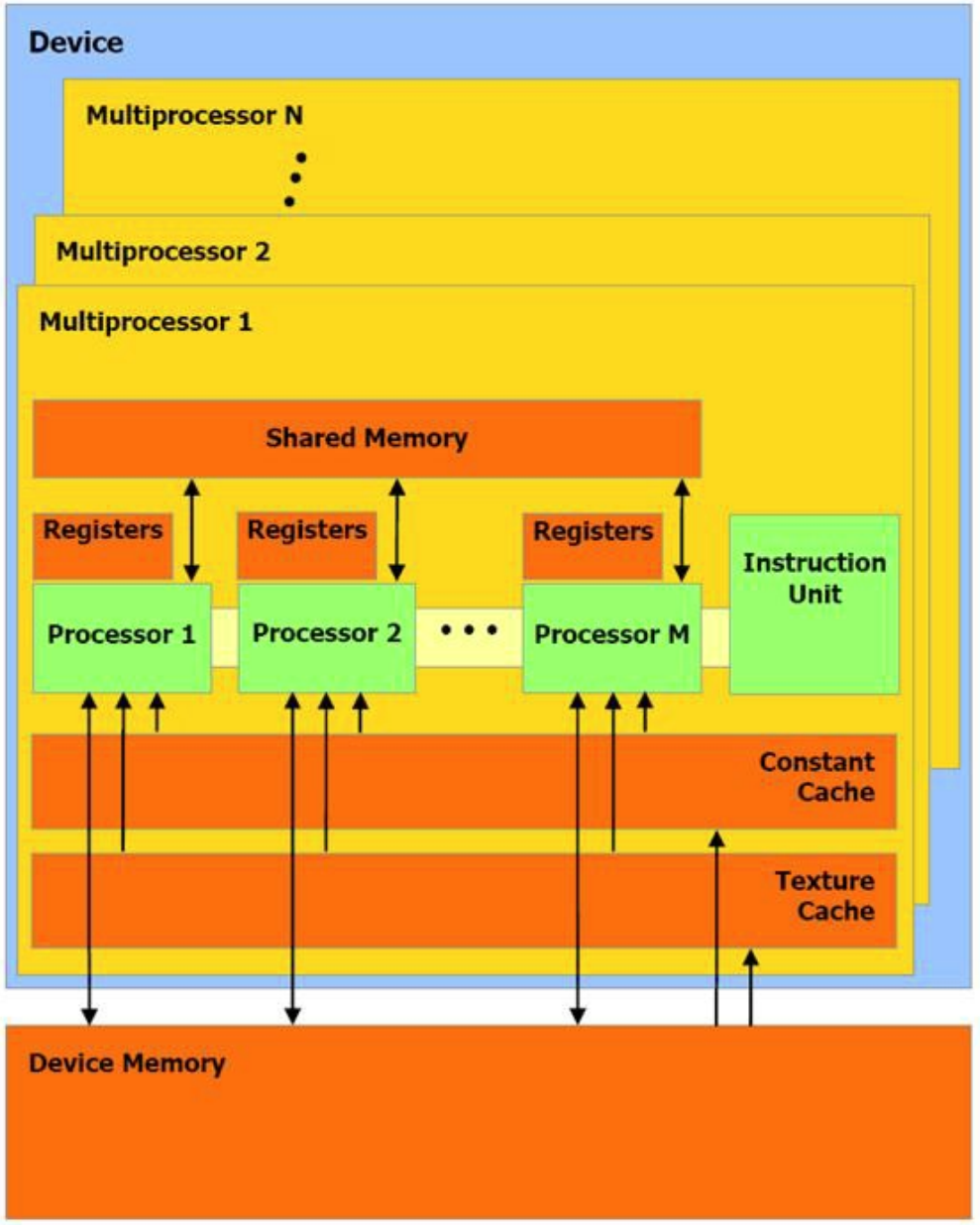
Numerical Computing
→ Fourier transform, Dense matrix operations, Sparse matrix operations, N-body…

Sequences and strings
→ Scan, Ranking , Sorting, Merging, Medians, Searching, String matching...

Trees and graphs
→ Trees. Connected components, Spanning trees, Shortest paths, Maximal independent set, Graph separators, ...

Computational Geometry
→ Convex hull, Closest pairs, Delaunay triangulation...

1. Processors → GPU systems



"Classical" GPUs

- devices on their own board

- feature various levels of local memory with very fast access

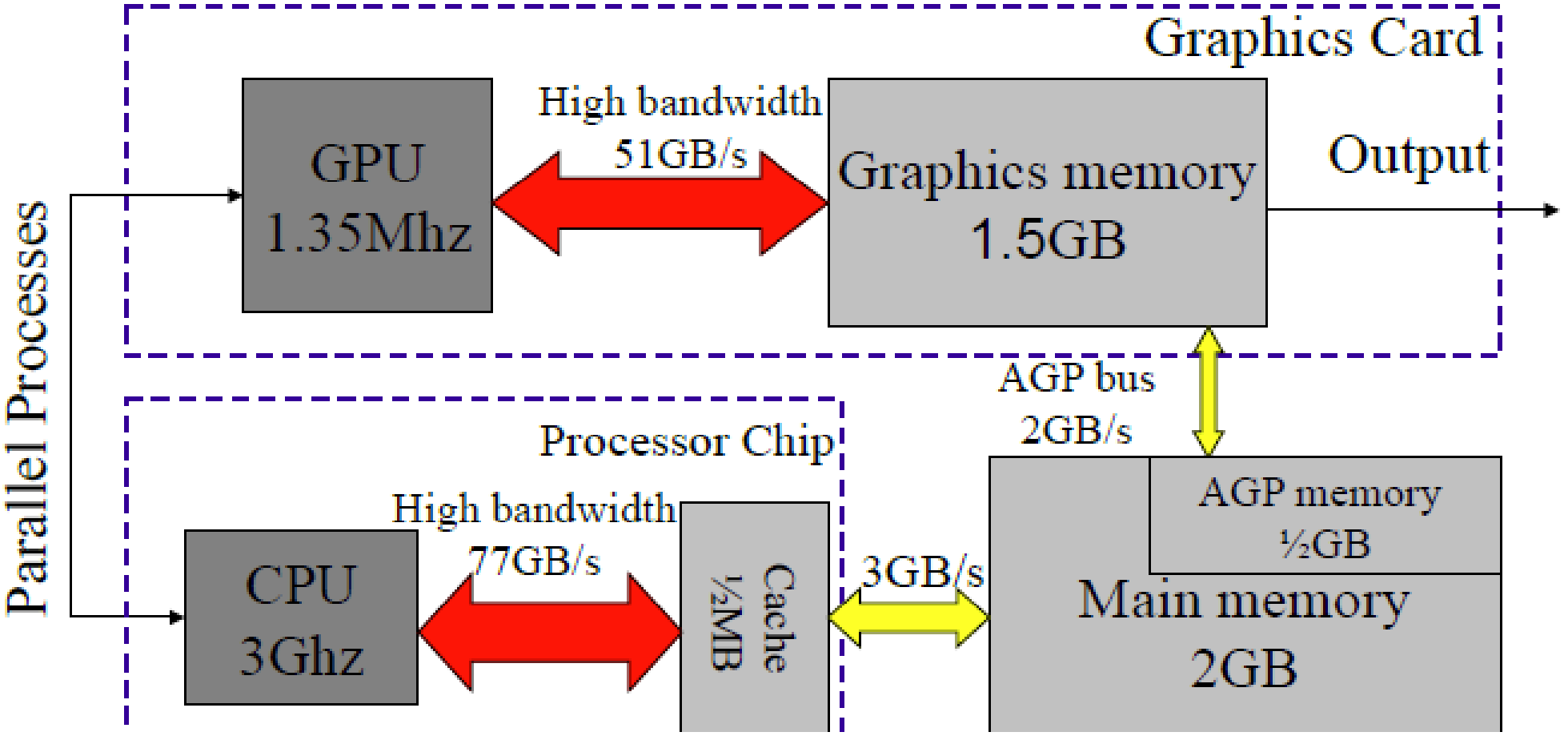
- interconnection to CPU via fast buses (PCI-express)

Memory… 6 types
On-chip / On-board

- Constant Memory
- Texture Memory
- Device Memory

1. Processors → GPU systems (early)

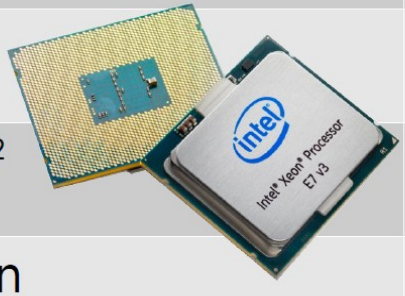
High memory bandwidth !



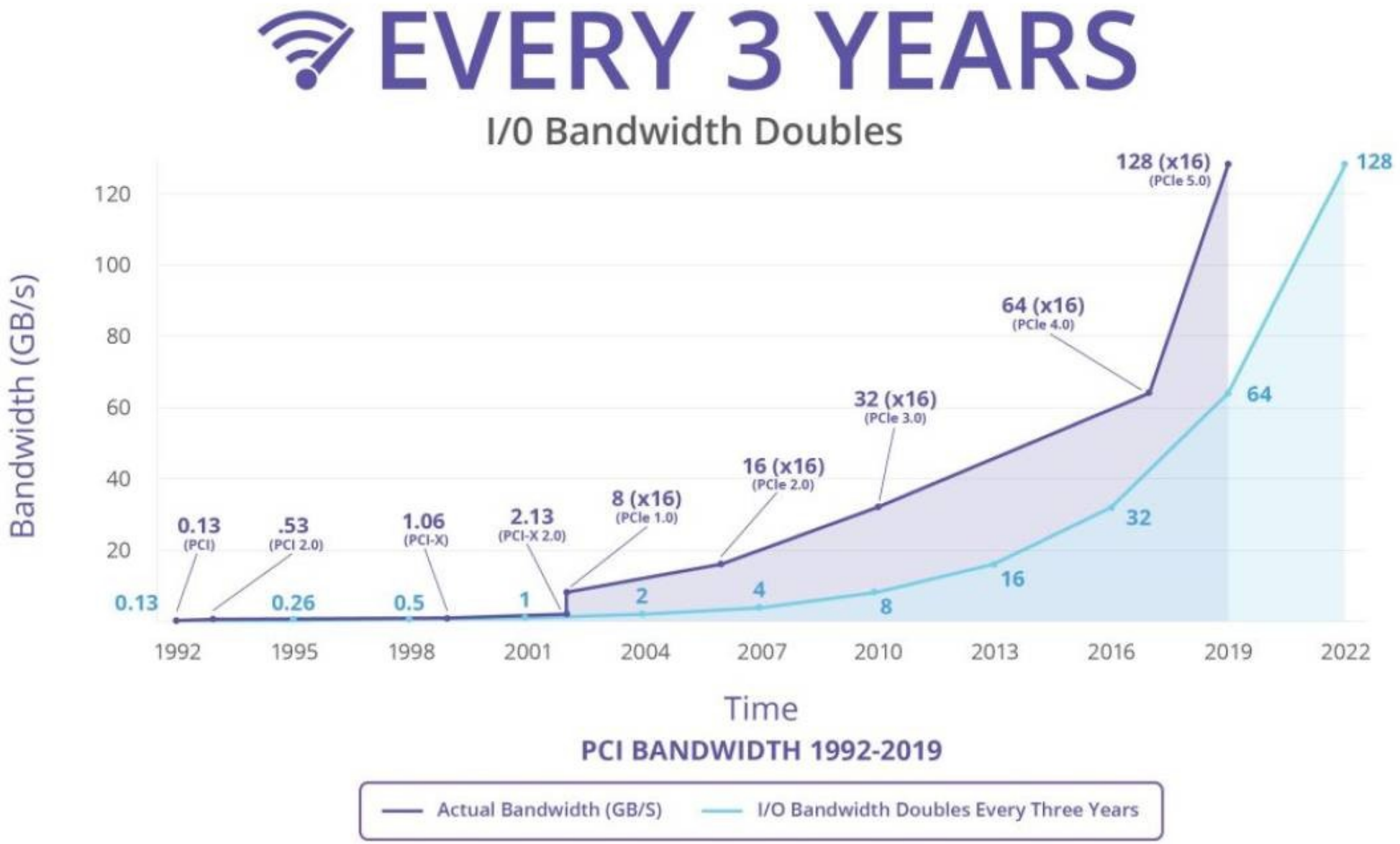
NVIDIA GPU Tesla C870 (2007) → → →

CPU vs GPU

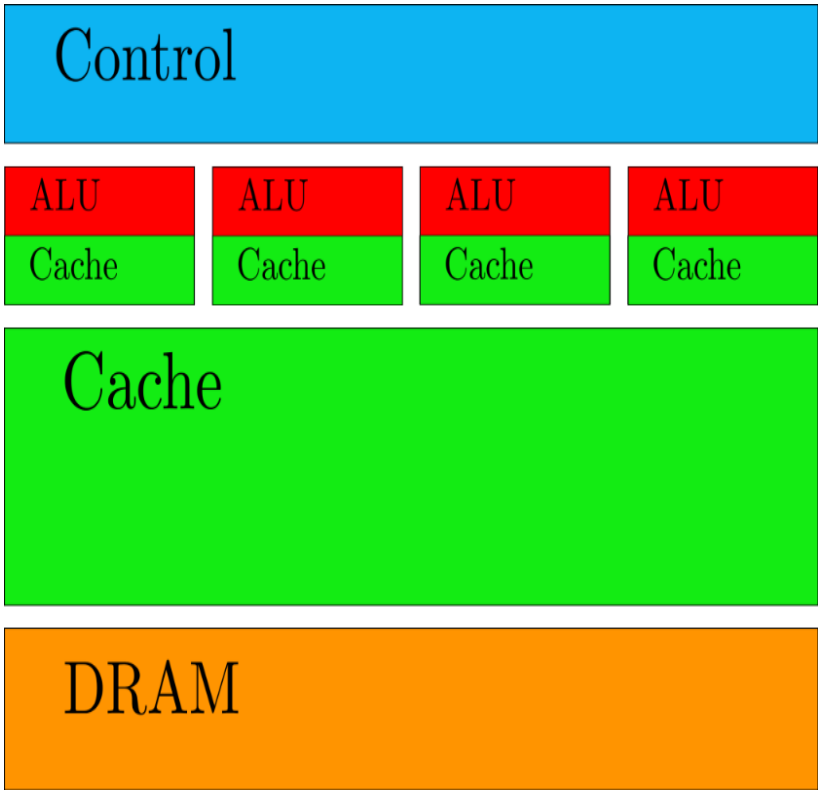
	Intel Core E7-8890 v3	**GeForce GTX 1080**
Core count	18 cores / 36 threads	20 SMs / 2560 cores
Frequency	2.5 GHz	1.6 GHz
Peak Compute Performance	1.8 TFLOPs	8873 GFLOPs
Memory bandwidth	Max. 102 GB/s	320 GB/s
Memory capacity	Max. 1.54 TB	8 GB
Technology	22 nm	16 nm
Die size	662 mm^2	314 mm^2
Transistor count	5.6 billion	7.2 billion
Model	Minimize latency	Hide latency through parallelism



CPU vs GPU


EVERY 3 YEARS
I/0 Bandwidth Doubles
Bandwidth (GB/s)
120
100
80
60
40
20
0.13
(PCI)
.53
(PCI 2.0)
1.06
(PCI-X)
2.13
(PCI-X 2.0)
8 (x16)
(PCIe 1.0)
16 (x16)
(PCIe 2.0)
32 (x16)
(PCIe 3.0)
64 (x16)
(PCIe 4.0)
128 (x16)
(PCIe 5.0)
0.13
0.26
0.5
1
2
4
8
16
32
64
128
1992
1995
1998
2001
2004
2007
2010
2013
2016
2019
2022
Time
PCI BANDWIDTH 1992-2019
Actual Bandwidth (GB/S)
I/O Bandwidth Doubles Every Three Years

CPU



- Multilevel and Large Caches
  - Convert long latency memory access
- Branch prediction
  - To reduce latency in branching
- Powerful ALU
- Memory management
- Large control part

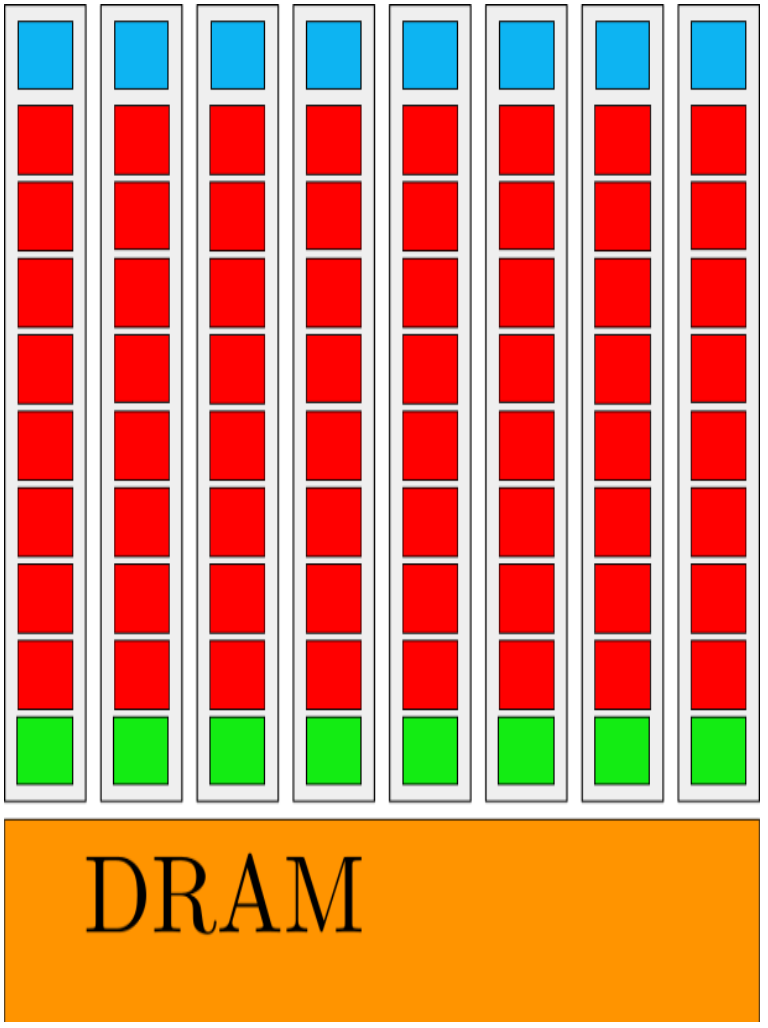
CPU: latency oriented design

GPU

- SIMT/D architecture

  (Single instruction Multiple Thread/Data)
- SMX (Streaming Multi Processors) to execute kernels
- Thread level parallelism
- Limited caching
- Limited control
- No branch prediction,

  but branch predication



GPU: throughput oriented design

Note: Instruction pipelining

Clock Cycles / status

0 /
Four instructions are waiting to be executed

1 /
The green instruction is fetched from memory

2 /
The green instruction is decoded
The purple instruction is fetched from memory

3 /
The green instruction is executed (actual operation is performed)
The purple instruction is decoded
The blue instruction is fetched

4 /
The green instruction's results are written back to the register file or memory
The purple instruction is executed
The blue instruction is decoded
The red instruction is fetched

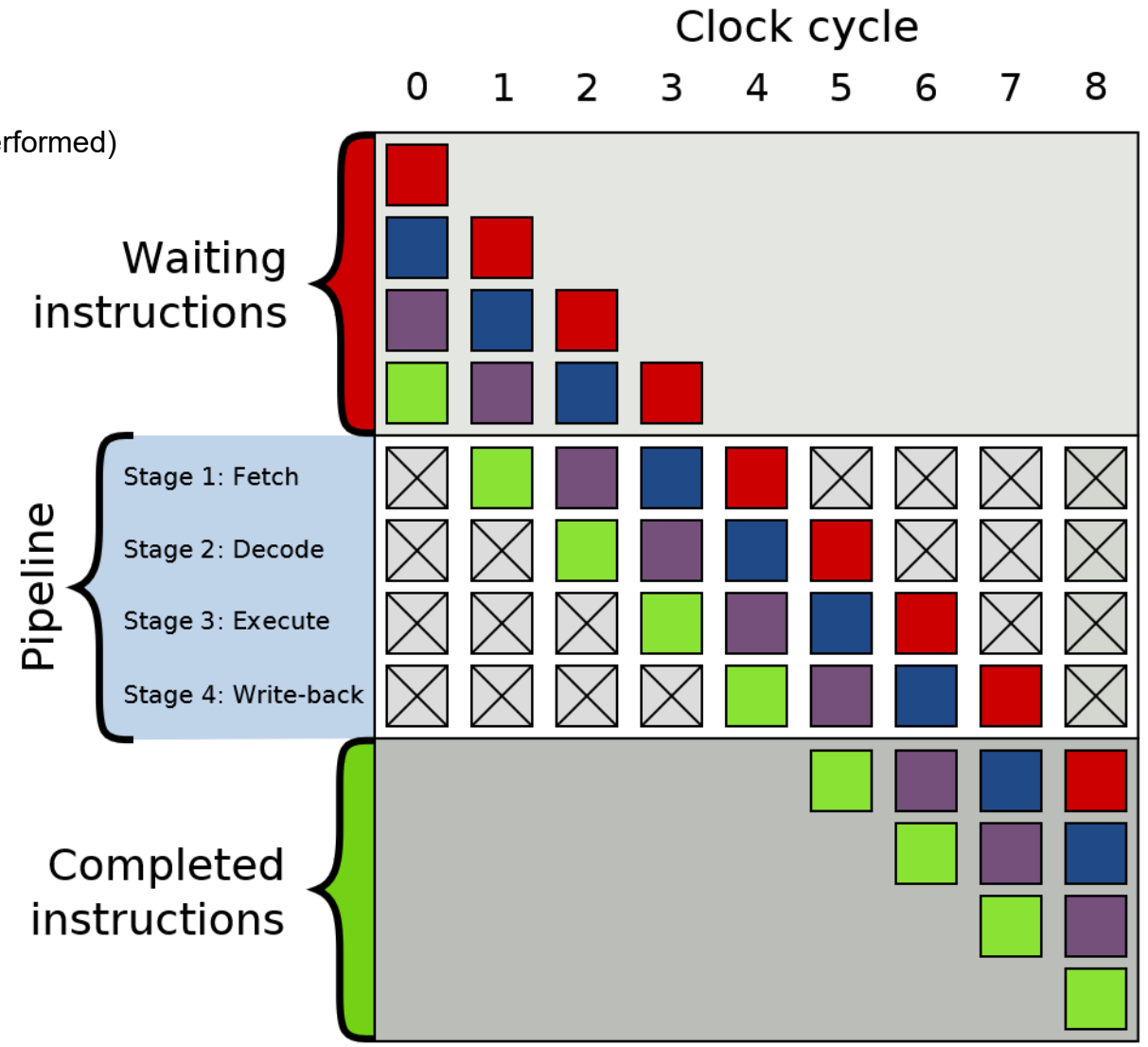
5 /
The execution of green instruction is completed
The purple instruction is written back
The blue instruction is executed
The red instruction is decoded

6 /
The execution of purple instruction is completed
The blue instruction is written back
The red instruction is executed

7 /
The execution of blue instruction is completed
The red instruction is written back

8 /
The execution of red instruction is completed

9 /
The execution of all four instructions is completed



Branching prediction / predication

Conditional code imply issues to execution in a computing system.
In particular when instruction pipelining is exploited as pipelining is slowed down by branches

Pseudocode Example:
if condition
    {dosomething}
else
    {dosomethingelse};

Solutions

1) Branching prediction:
dedicated digital bloks into a CPU attempt to avoid waste of time by trying to guess whether the conditional jump is most likely to be taken or not taken.
The branch that is guessed to be the most likely is then fetched and speculatively executed.
If it is later detected that the guess was wrong, then the speculatively executed or partially executed instructions are discarded and the pipeline starts over with the correct branch (incurring a delay)

Resulting Assembler code:
branch-if-condition to label1
dosomethingelse
branch-to label2
label1:
dosomething
label2:
...

2) Braching predication

Branching prediction / predication

Conditional code imply issues to execution in a computing system.
In particular when instruction pipelining is exploited as pipelining is slowed down by branches

Pseudocode Example:
if condition
{dosomething}
else
{dosomethingelse};

Solutions

2) Braching predication:
all possible branch paths are coded inline, but some instructions execute while others do not.
I.e. each instruction is associated with a predicate and that the instruction will only be executed if the predicate is true

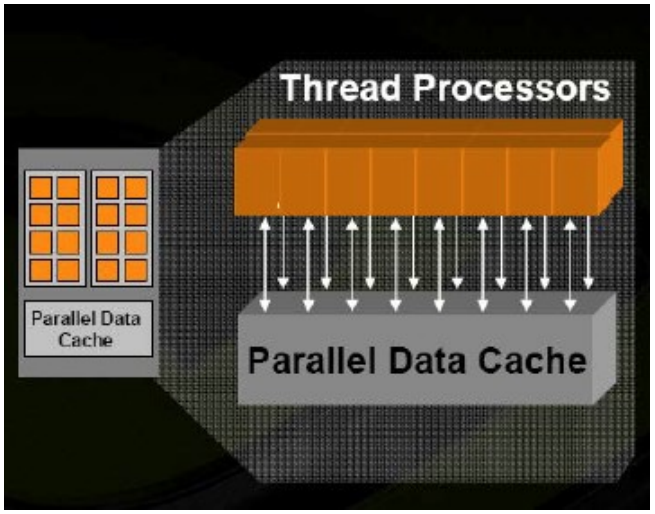
Resulting machine code:
(condition) dosomething
(not condition) dosomethingelse

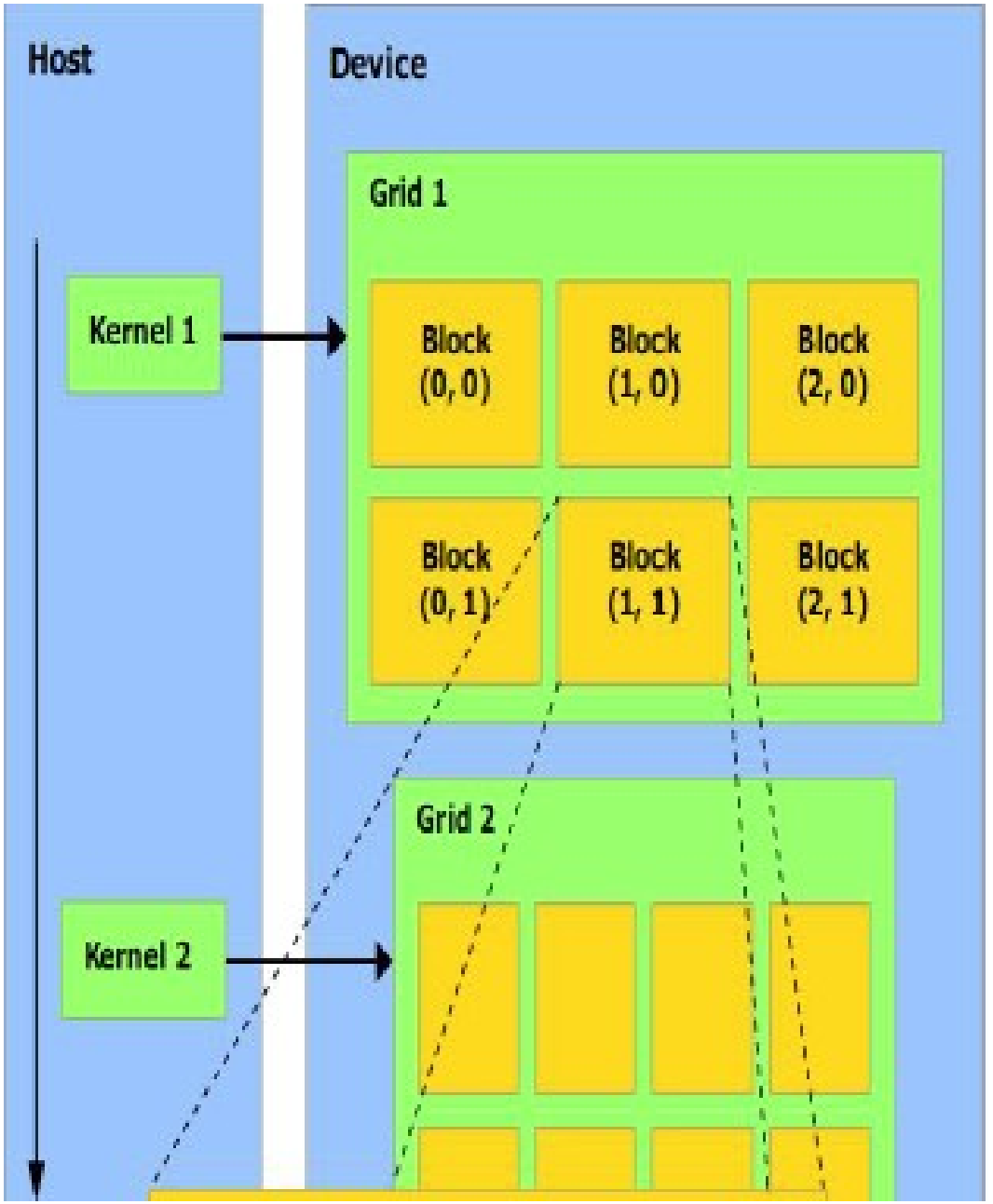
GPU (and in general single instruction multiple threads computing) a computing core executes the code of any branch (thread) including those with unmet condition, in which case the core does not commit the results computed in that execution path

1. Processors → GPU practical use

Important Concepts:

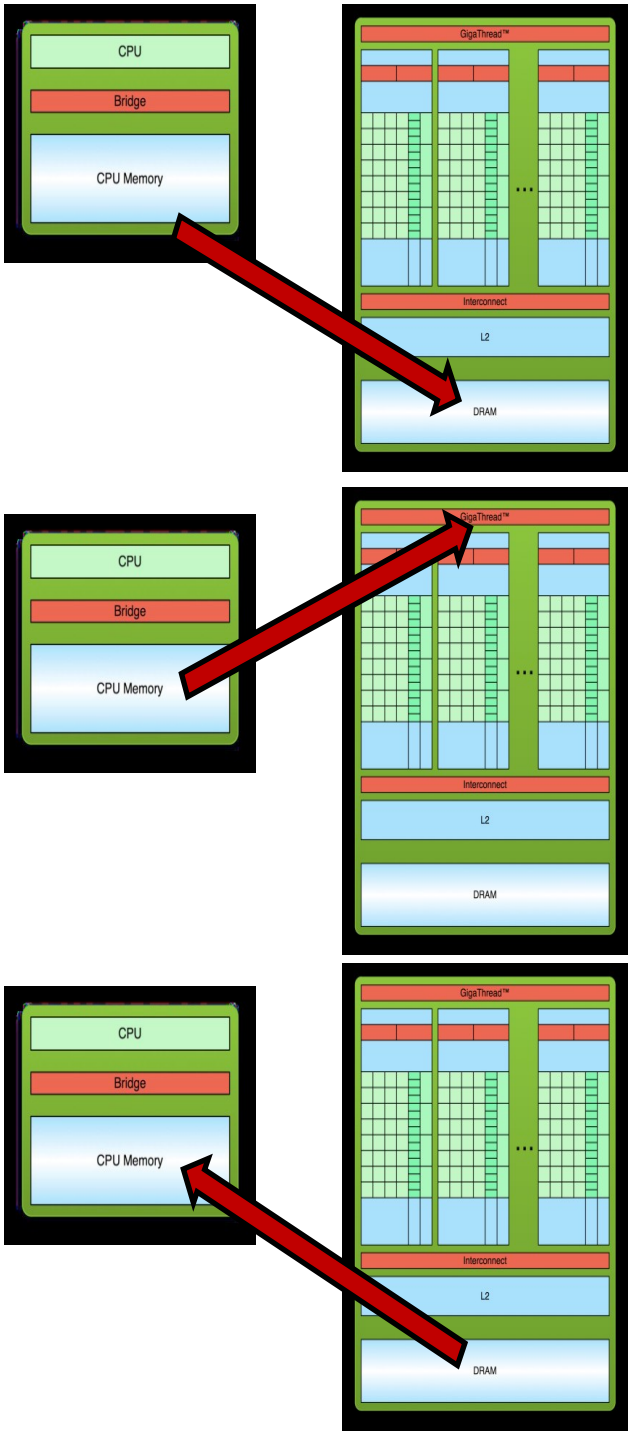
- Device: GPU, viewed as a co-processor.
- Host: CPU
- Kernel: data-parallel, computed-intensive positions of application running on the device.





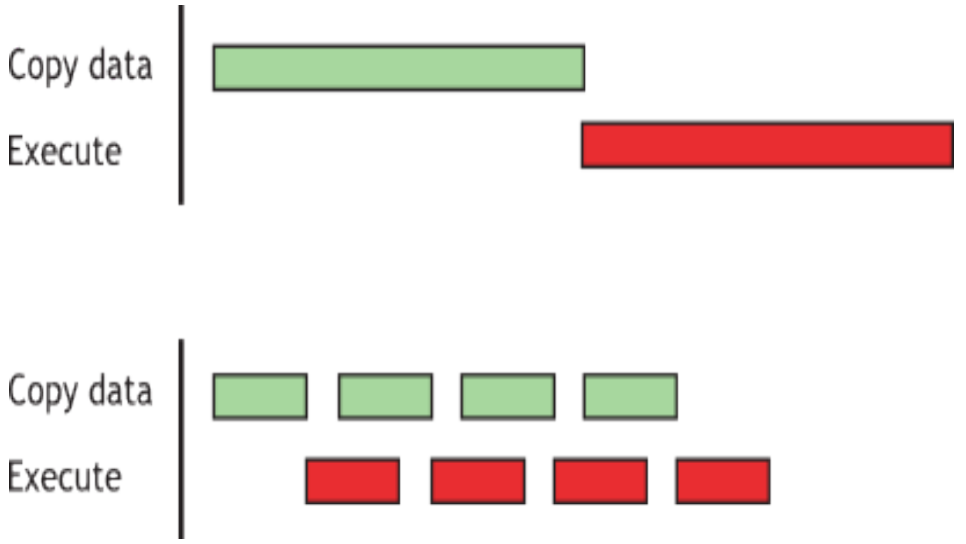
Processing model (with CUDA)

- What is **CUDA**?
  - It is a set of C/C++ extensions to enable the **GPGPU** computing on **NVIDIA GPUs**
  - Dedicated APIs allow to control almost all the functions of the graphics processor
- Three steps:
  - 1) copy data from **Host** to **Device**
  - 2) copy **Kernel** and execute
  - 3) copy back results



Streams

- The main purpose of all the GPU computing is to hide the latency
- In case of multiple data transfer from host to device the asynchronous data copy and kernel execution can be superimposed to avoid dead time




```
kernel<<< blocks, threads, bytes >>>();    // default stream
kernel<<< blocks, threads, bytes, 0 >>>(); // stream 0
```

Grids, blocks and threads

- The computing resources are logically (and physically) grouped in a flexible parallel model of computation:
  - 1D,2D and 3D grid
  - With 1D, 2D and 3D blocks
  - With 1D, 2D and 3D threads
  - Only threads can communicate and synchronize in a block
  - Threads in different blocks do not interact, threads in same block execute same instruction at the same time
- The “shape” of the system is decided at kernel launch time

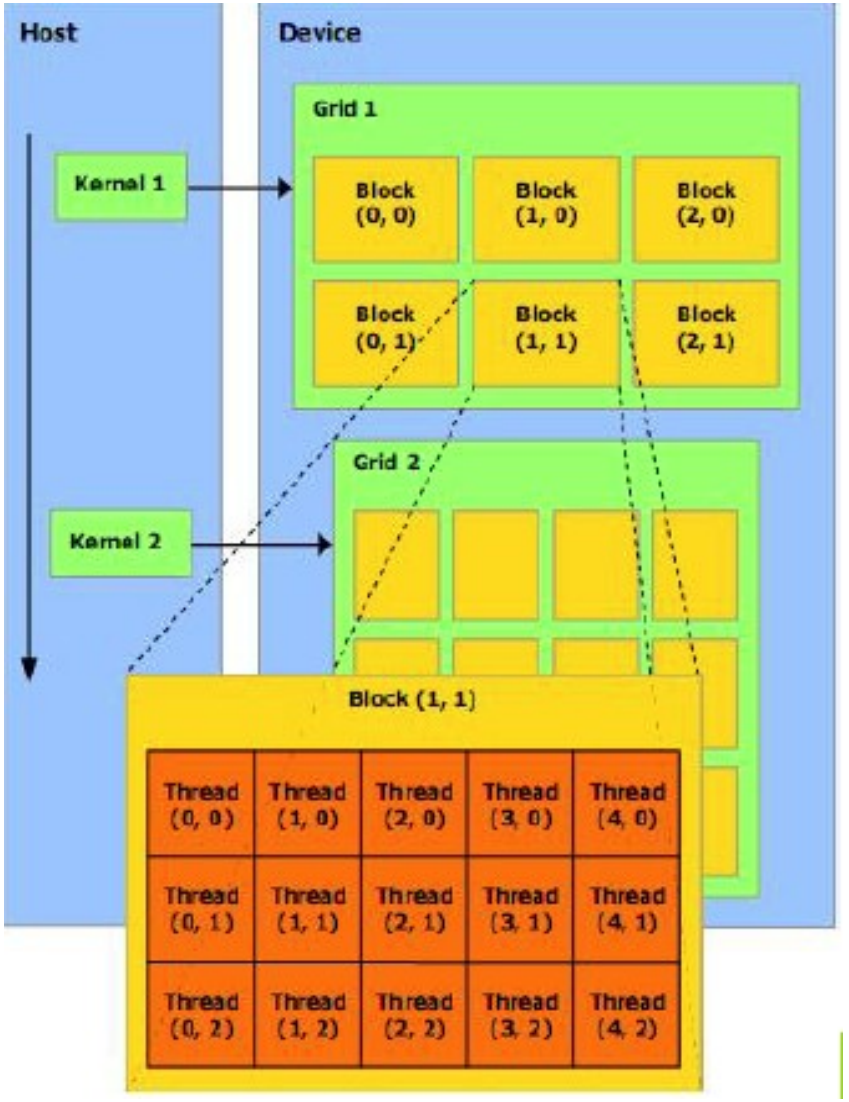
Grids, blocks and threads

- The computing resources are logically (and physically) grouped in a flexible parallel model of computation:
  - 1D,2D and 3D grid
  - With 1D, 2D and 3D blocks
  - With 1D, 2D and 3D threads

    threads → basic execution units

  - Only threads can cooperate

    by communicating, sharing data and synchronize in a block
  - Threads in different blocks do not interact
  - Threads in same block execute same instruction at the same time



The “shape” of the system is decided at kernel launch time

1. Processors → GPU coding (eg CUDA)

Simple matrix multiplication example:

cpu c program:

```
void addVector (fload *a, float *b,
                fload *c, int N)
{
    int i, index,
    for (i = 0; i<N, i++) {
        c[index] = a[index] + b[index];
    }
}

void main()
{
  .....
  addVector(a, b, c, N);
  .....
}
```

cuda program:

```
__global__ void addVector (fload *a, float *b,
                           float *c)
{
    int i = threadIdx.x + blockDim.x*blockIdx.x;
    c[i] = a[i] + b[i];
}
```

Device code

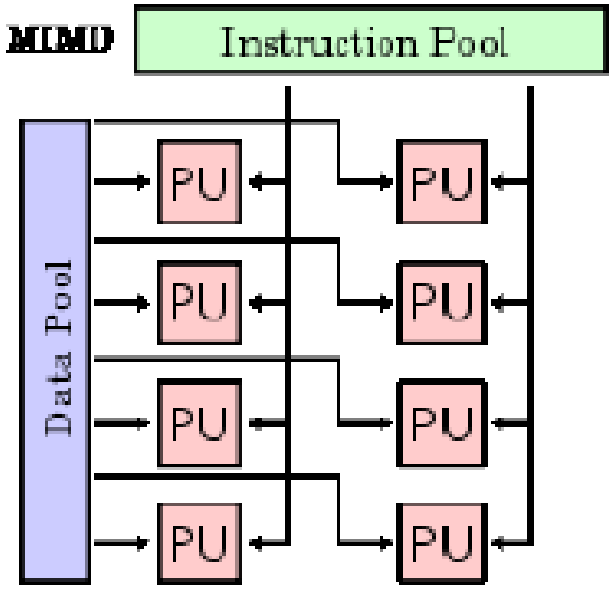
```
Void main()
{
  ...
  // allocation & transfer data to GPU
  //Excuate on N/256 blocks of 256 threads each
  addVector << N/256, 256 >> ( d_A, d_B, d_C);
  .....
}
```

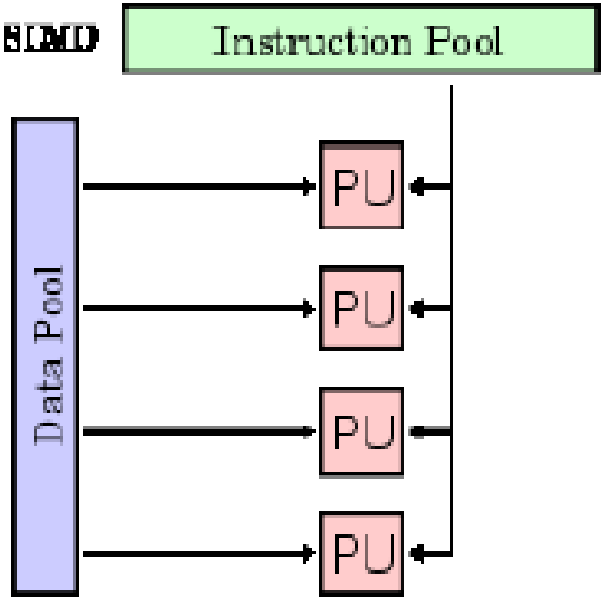
Host code

1. Processors → GPU and CPU

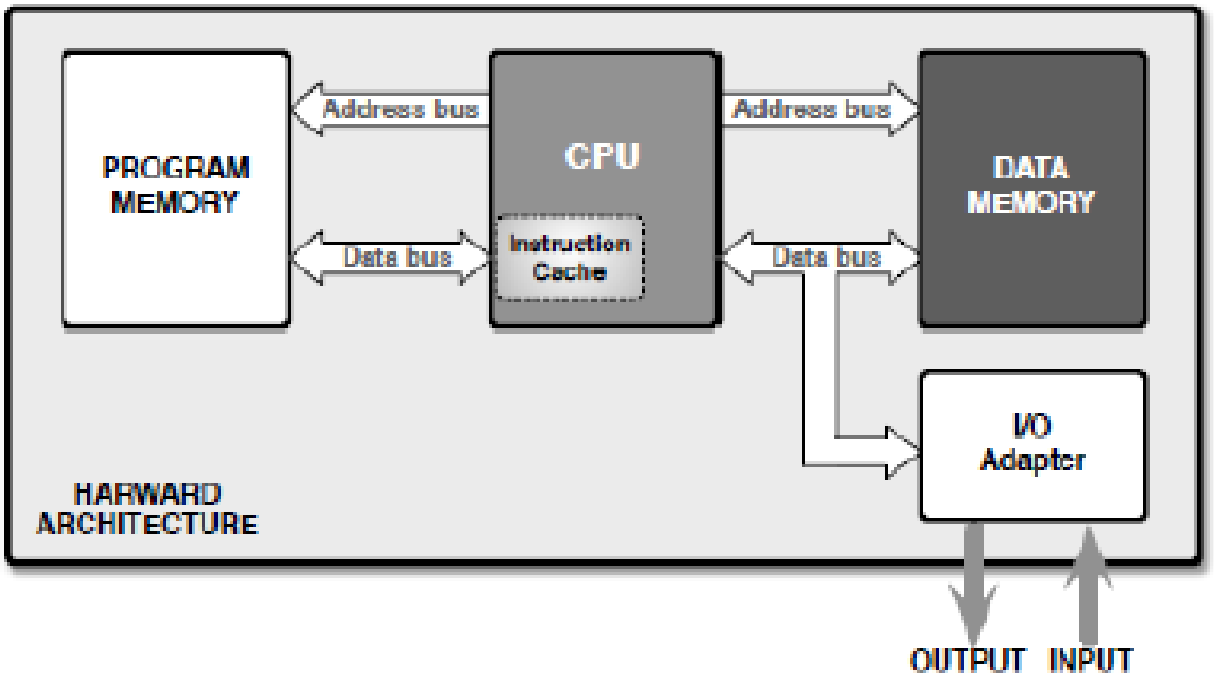
CPU multiple cores
- MIMD (Multiple Instruction / Multiple Data)
- each core operates independently
- each can be working with a different code, performing different operations with entirely different data

GPU many cores
- SIMD (Single Instruction / Multiple Data)
- all cores executing the same instruction at the same time, but working on different data
- only one instruction de-coder needed to control all cores
- functions like a vector unit





1. Processors
→ Digital architectures with memories



Harvard Architecture

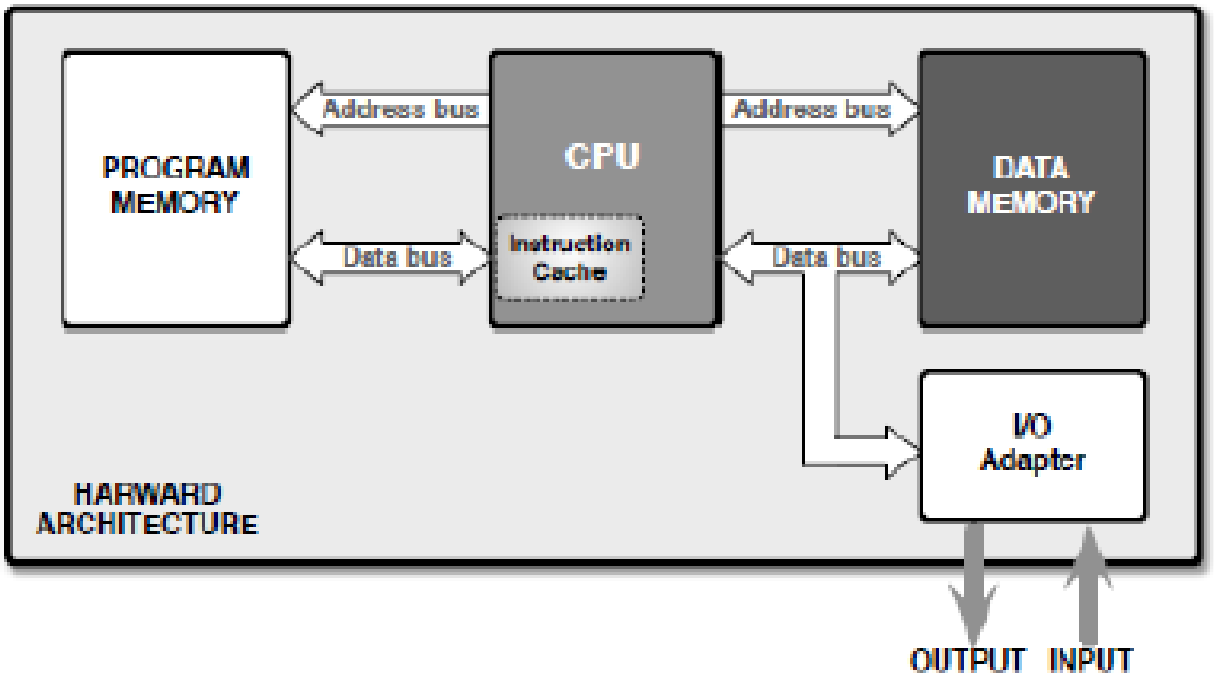
Harvard architecture machines are used mostly in applications where trade-offs (like the cost and power savings from omitting caches) outweigh the programming penalties from featuring distinct code and data address spaces

Benefits:

1) simultaneous instruction fetch and data R/W → faster performance

2) data and instructions memories are independent → memories can be of dirrerent types (R only, R/W) and feature different volumes, widths, …

1. Processors
→ Digital architectures with memories



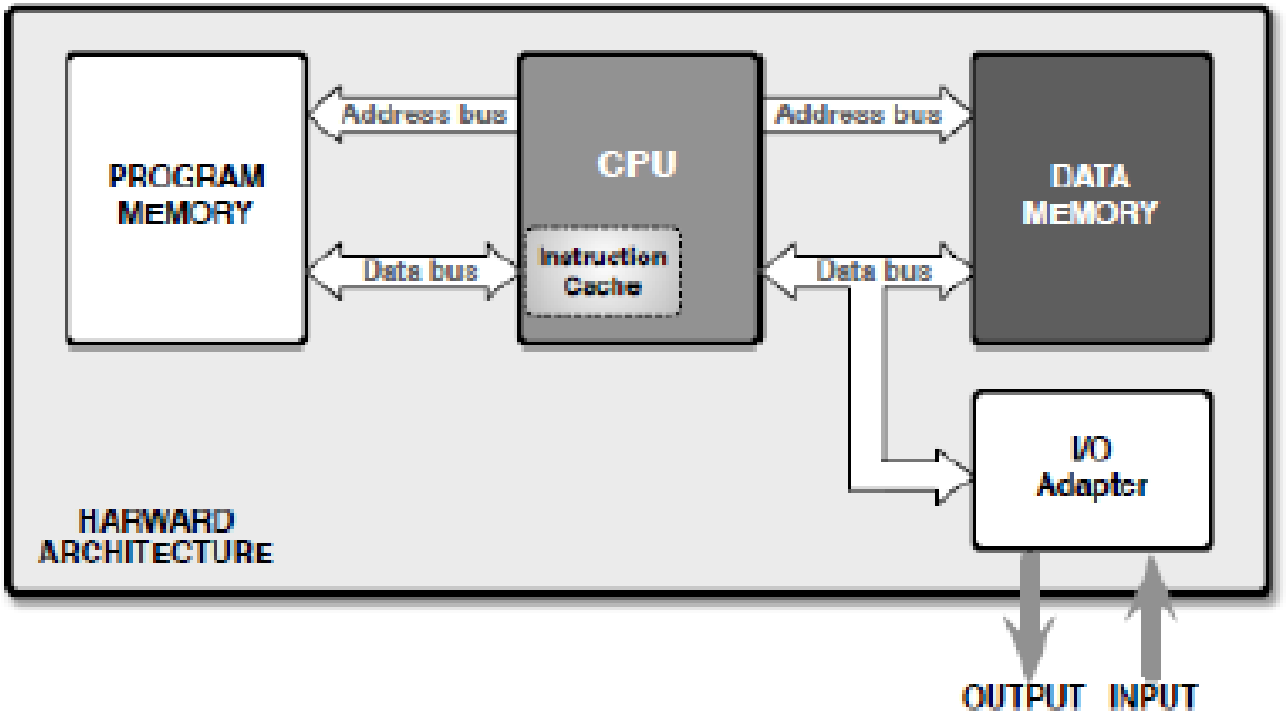
Harvard Architecture

Harvard architecture machines are used mostly in applications where trade-offs (like the cost and power savings from omitting caches) outweigh the programming penalties from featuring distinct code and data address spaces

Common examples are
- **Digital Signal Processors (DSPs)** and
- **Micro-controllers (μC)**

1. Processors
→ Digital architectures with memories

For heavy mathematical loads the Harvard architecture uses separate memories for data and instructions. The use of separate busses for data and instructions provides high speed.



Harvard Architecture

Note: modern CPUs and GPUs adopt hybrid von Neumann/Harvard architecture: where data and instructions share the same common memory but they are buffered into separate memory caches which the CPU can access indipendendly

1. Processors – Digital Signal Proc. (DSP)

Digital signal processors (DSPs) for Real Time applications generally execute small, highly optimized audio or video processing algorithms. They avoid caches because their behavior must be **extremely reproducible**
The difficulties of coping with multiple address spaces are of secondary concern to speed of execution.
Consequently, some DSPs feature multiple data memories in distinct address spaces to facilitate SIMD and VLIW (Very Long Instruction Word → instruction parallelism) processing

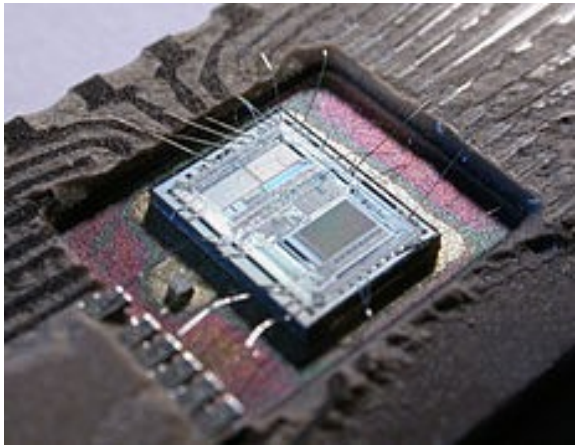
Digital signal processing algorithms typically require a large number of mathematical operations to be performed quickly and repeatedly on a series of data samples.
Signals are constantly converted from analog to digital, manipulated digitally, and then converted back to analog form. Many DSP applications have constraints on **latency**; that is, for the system to work, the DSP operation must be completed within some fixed time, and deferred (or batch) processing is not viable

1. Processors – Micro-Controllers

Microcontrollers are characterized by having
small amounts of program (flash memory) and data (SRAM) memory, and take advantage
of the Harvard architecture to speed processing by concurrent instruction and data access

The separate storage means the program and data memories may feature different bit widths,
for example using 16-bit-wide instructions and 8-bit-wide data.
They also mean that instruction prefetch can be performed in parallel with other activities.

A microcontroller can be considered a self-contained system
with a processor, memory and peripherals and can be used
as an embedded system. The majority of microcontrollers
in use today are embedded in other machinery,
such as automobiles, telephones, and peripherals for computers



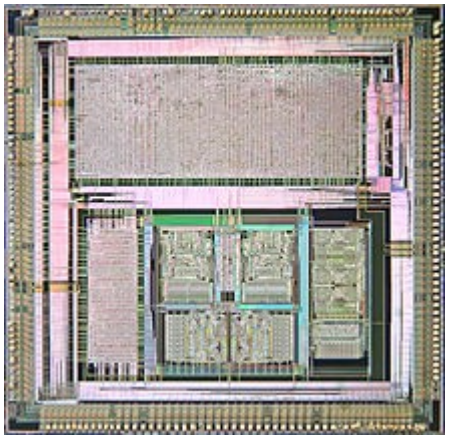
While some embedded systems are very sophisticated,
many have minimal requirements for memory and program length,
with no operating system, low software complexity and
can feature low power consumption

Typical input and output devices include switches, relays, solenoids, LED's, small or custom liquid-crystal displays, radio frequency devices, and sensors for data such as temperature, humidity, light level etc. Embedded systems usually have no keyboard, screen, disks, printers, or other recognizable I/O devices of a personal computer, and may lack human interaction devices of any kind → IoT boom

1. Processors: ASICs vs FPGA

CPUs, GPUs, DSPs and Micro-controllers
are Application-Specific Integrated Circuit (ASICs)

ASICs are in general Integrated Circuits customized for a particular use (rather than intended for general-purpose use)
ASICs might include entire microprocessors, memory blocks including ROM, RAM, EEPROM, flash memory and other large building blocks. Such ASICs are often termed SoCs (systems-on-chip). Designers of digital ASICs often use a hardware description language (HDL), such as Verilog or VHDL, to describe the functionality of ASICs



1. Processors: ASICs vs FPGA

Field-Programmable Gate Arrays (FPGAs) are IC designed to be configured by a customer or a designer after manufacturing

The FPGA configuration is generally specified using a hardware description language (HDL). Circuit diagrams were previously used to specify the configuration, but this is increasingly rare due to the advent of electronic design automation tools

FPGAs contain

- an array of programmable **logic blocks**, and
- hierarchy of **"reconfigurable interconnects"** that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations

1. Processors: ASICs vs FPGA

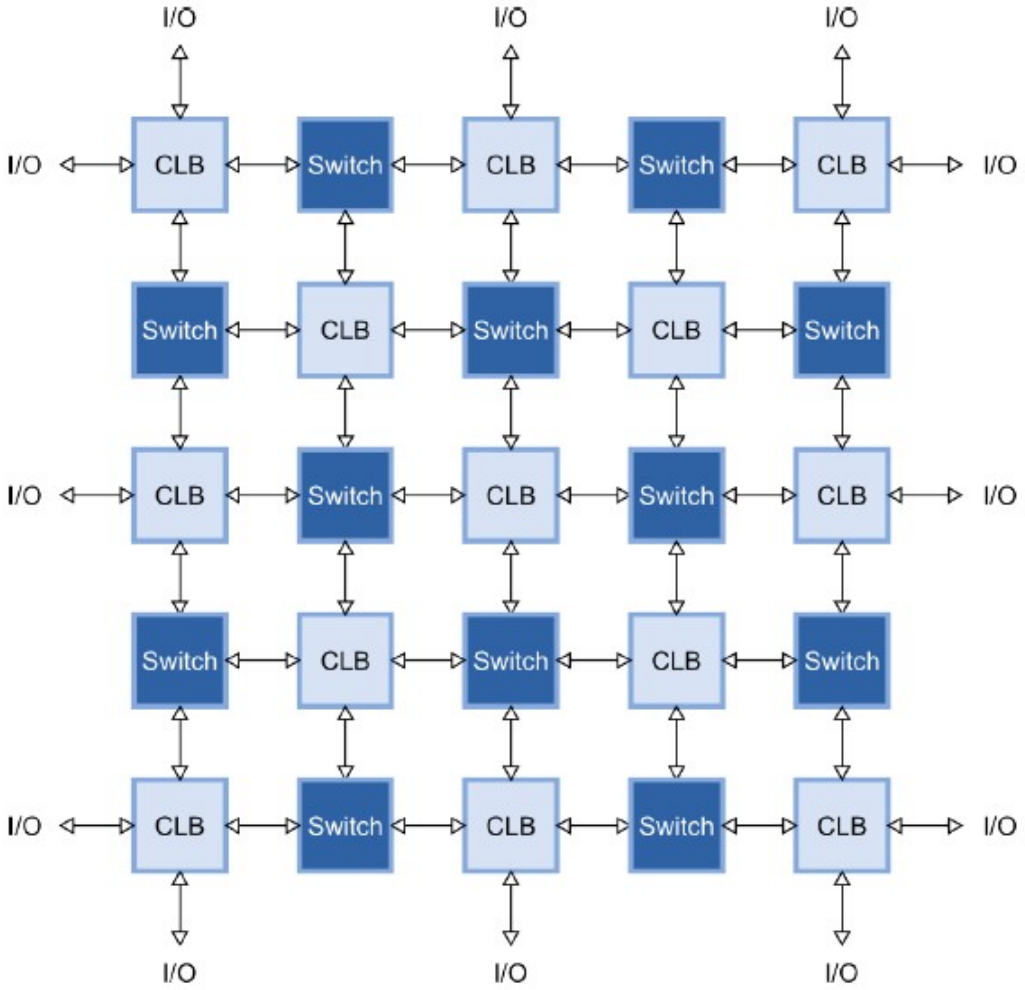
Logic blocks can be configured
- as simple logic gates like AND and XOR
- to perform complex combinational functions

Logic blocks also include memory elements,
which may be simple flip-flops or more complete memory blocks

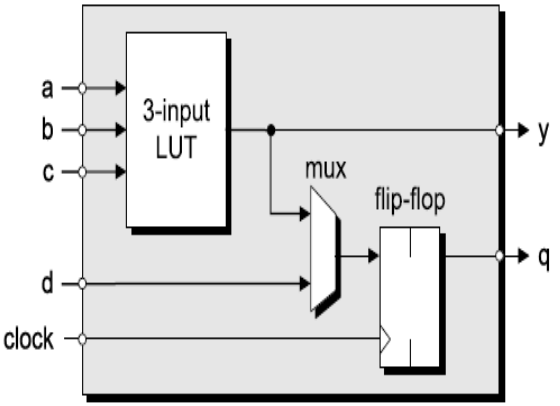
FPGAs can be reprogrammed to implement different logic functions
→ allowing flexible reconfigurable computing
(as performed in computer software)

Field Programmable Gate Arrays - FPGA

An FPGA takes a different approach. It has a bunch of simple, *configurable logic blocks* (CLBs) interspersed within a *switching matrix* that can rearrange the interconnections between the them. Each logic block is individually programmed to perform a logic function (such as AND, OR, XOR, etc.) and then the switches are programmed to connect the blocks so that the complete logic functions are implemented.

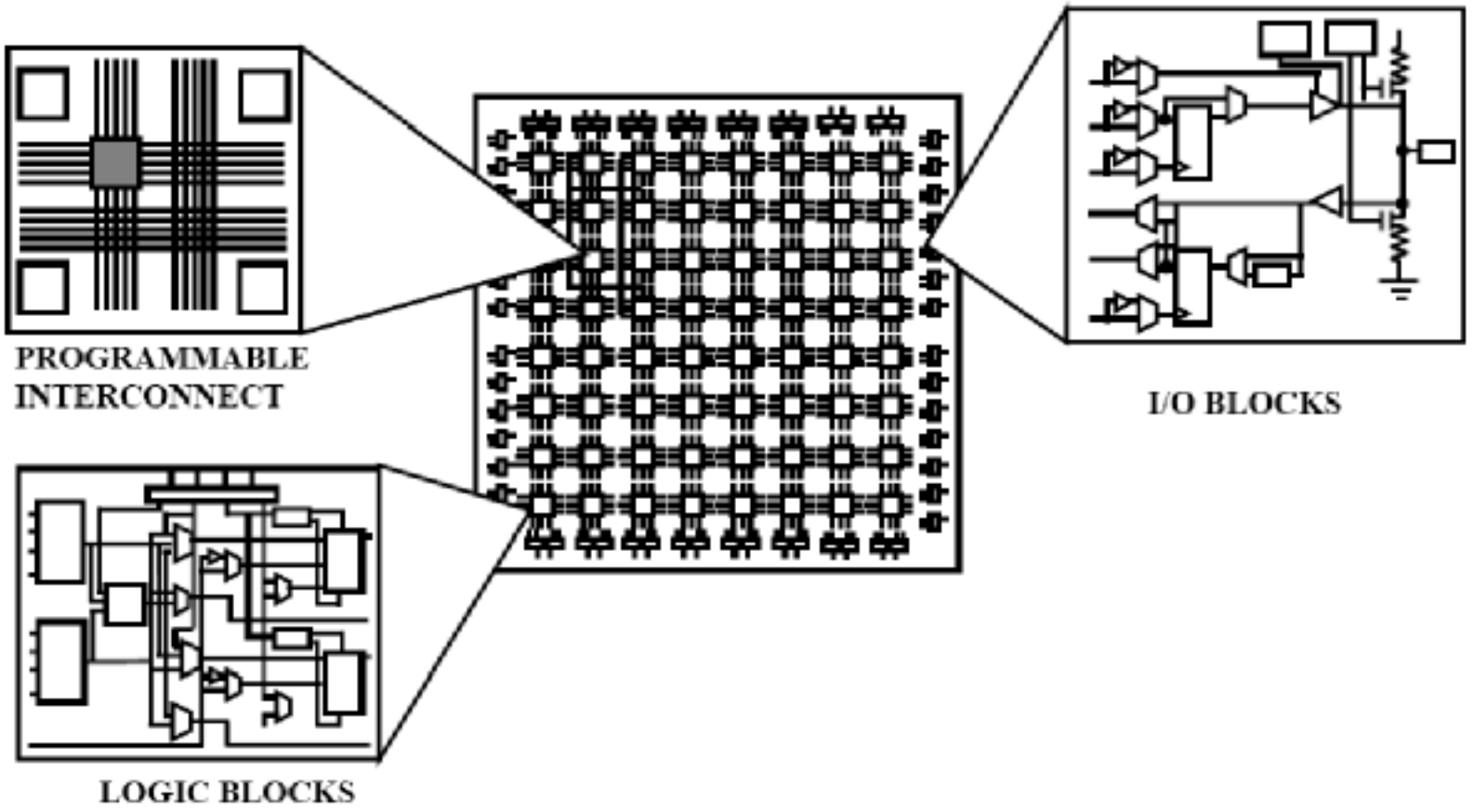


Example of CLB



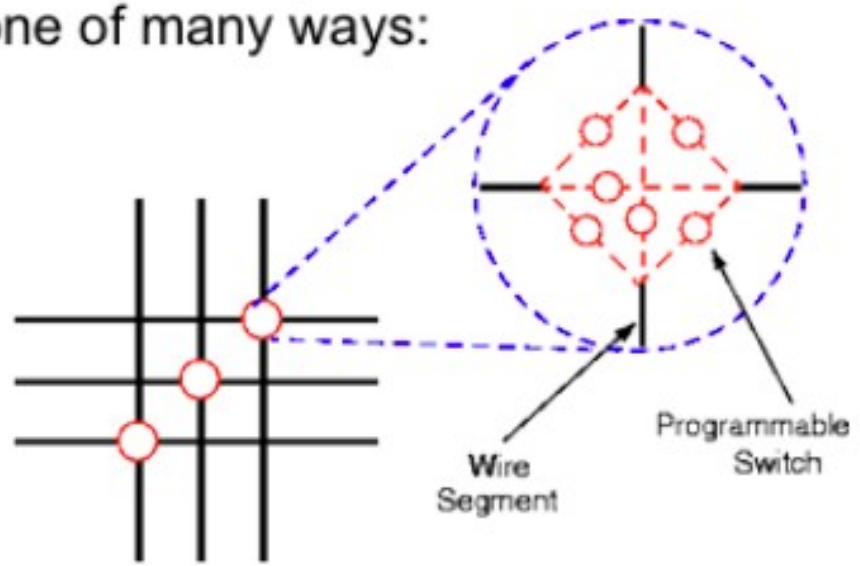
1) Look Up Table (LUT)
   → any combinatory function (3 inputs)
2) flip-flop (FF)
   → 1-bit memory
3) additional input
   → to combine large combinatory functions through multiplexer (MUX)
4) clock !!! for sync

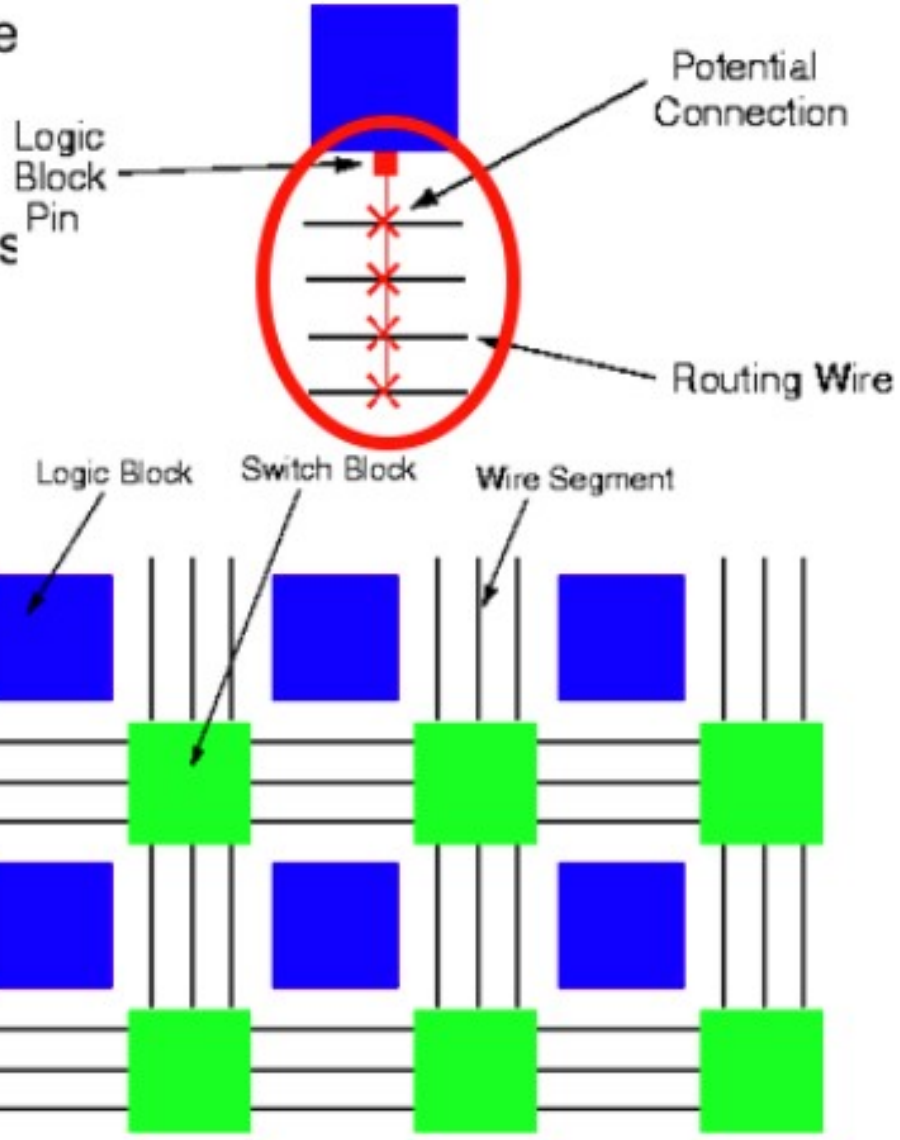
Field Programmable Gate Arrays - FPGA



FPGA – Programmable Routing

- Between rows and columns of logic blocks are wiring channels
- These are programmable – a logic block pin can be connected to one of many wiring tracks through a programmable switch
- Xilinx FPGAs have dedicated switch block circuits for routing (more flexible)
- Each wire segment can be connected in one of many ways:

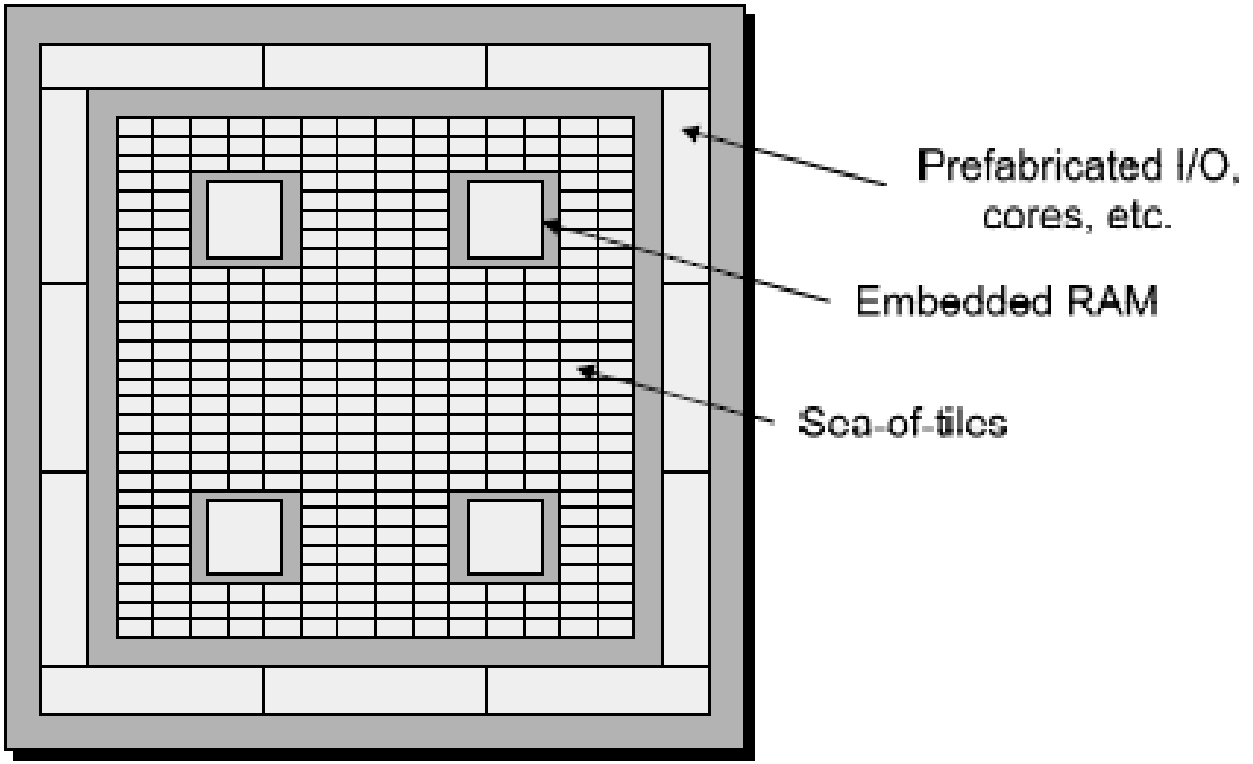




FPGA – Additional Blocks

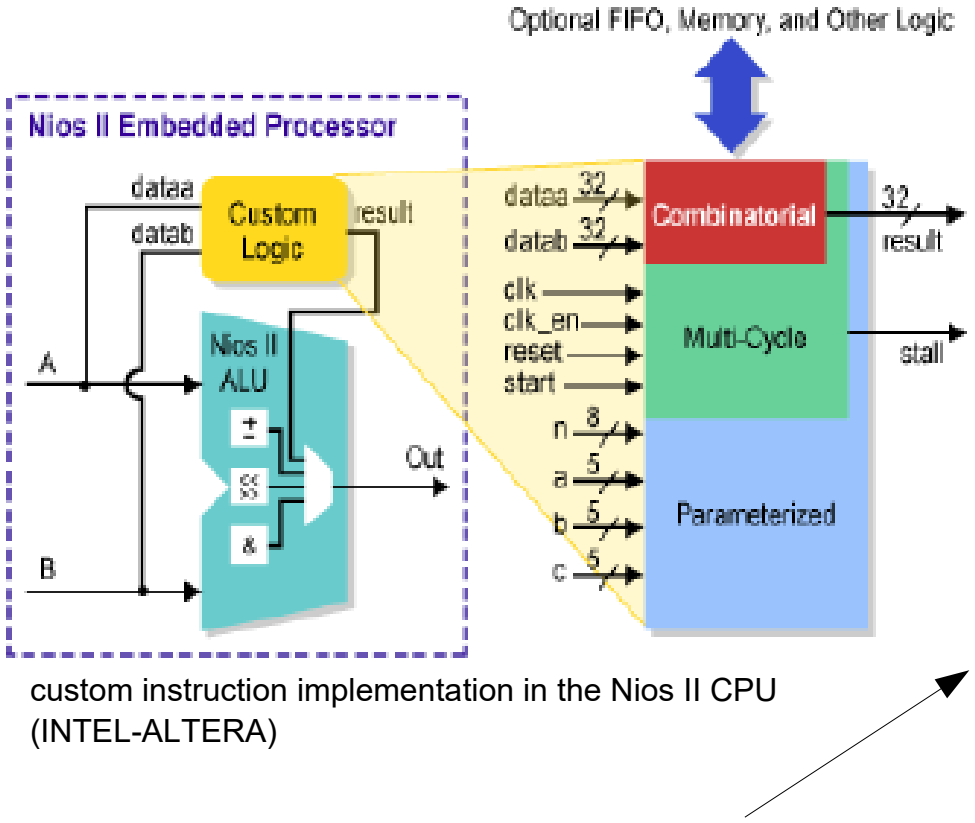
Le FPGA spesso includono moduli funzionali specifici come:

- sommatori
- moltiplicatori
- unità MAC
- memorie RAM
- core di microprocessori
- interfaccie di I/O veloci

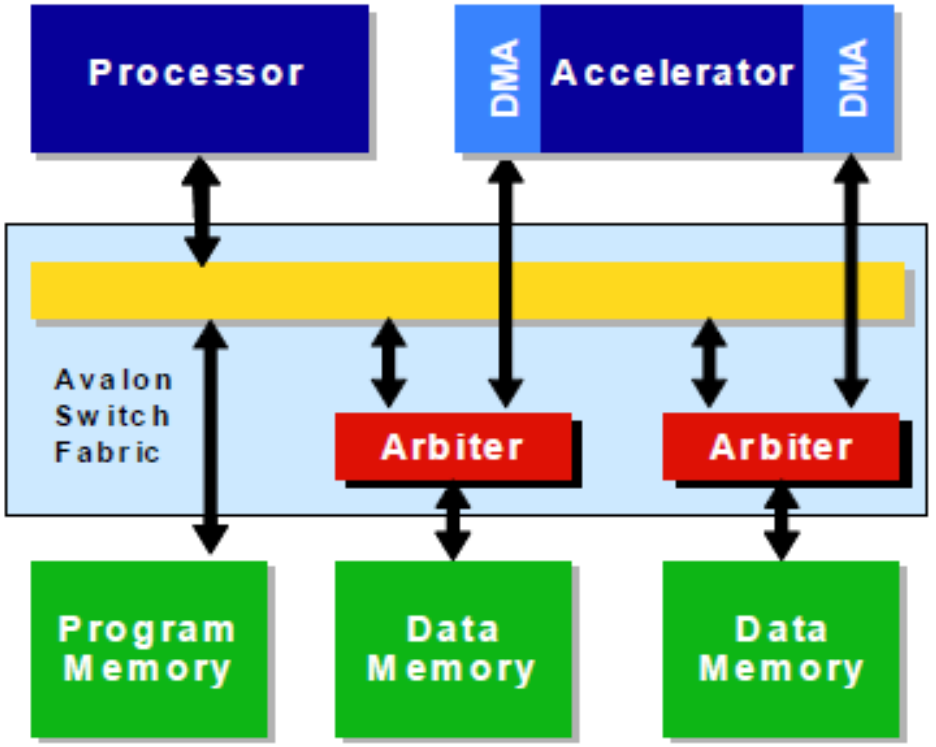


1. Processors: computing with FPGAs

Soft processor IP cores can be implemented within the FPGA logic. Nios II, MicroBlaze and Micro32 are examples of popular softcore processors. Many modern FPGAs are programmed at "run time", which has led to the idea of reconfigurable computing or reconfigurable systems → CPUs that reconfigure themselves to suit the task at hand



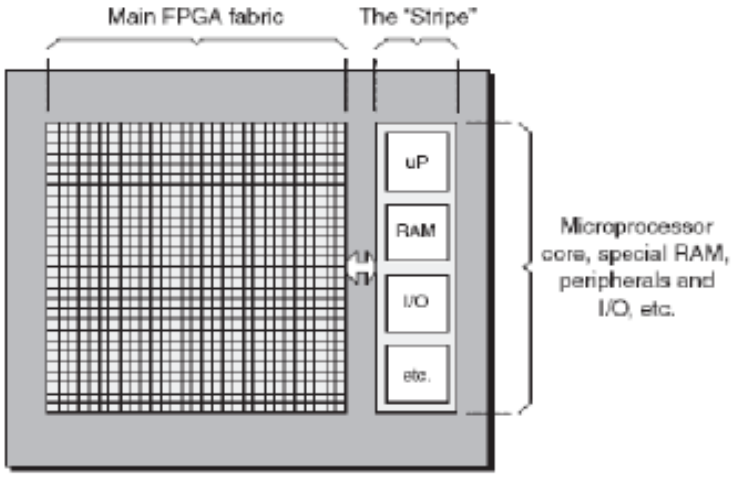
custom instruction implementation in the Nios II CPU (INTEL-ALTERA)

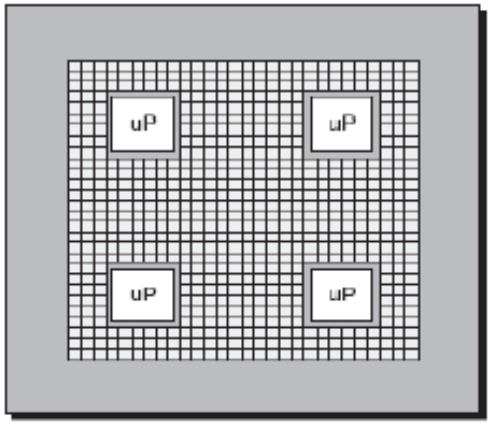


Hardware accelerator implementation with the Avalon switch fabric

1. Processors: computing with FPGAs

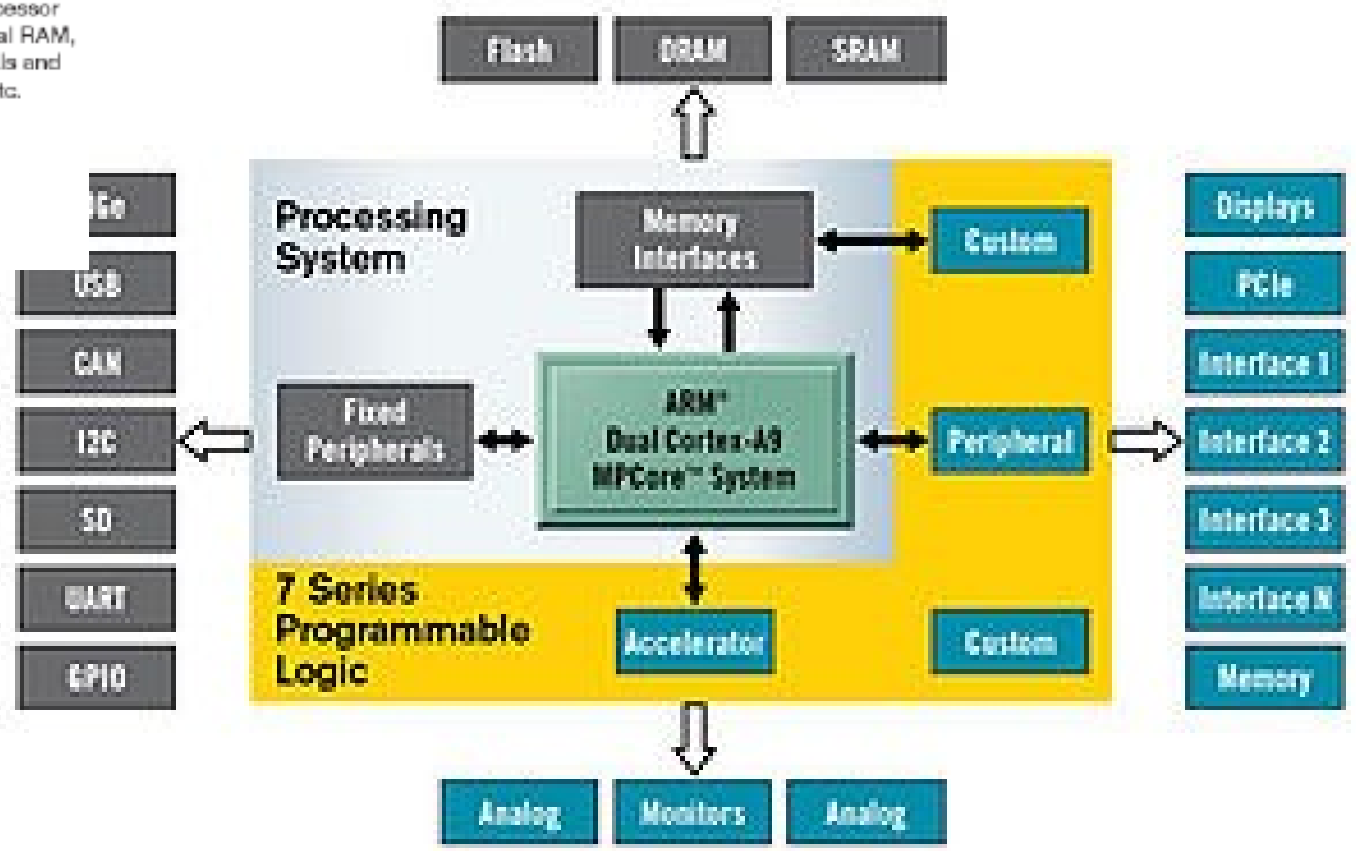
Hard processor IP cores can be implemented on the same FPGA chip





Four embedded cores

A Xilinx Zynq-7000 All Programmable System on a Chip.



1. Processors: hardware accelerators

GPU and FPGA are deployed alonside CPUs as HW accelerators

CPU
- Fast caches
- Branching adaptability
- High performance

→ CPUs are great for task parallelism

GPU
- Multiple ALUs
- Fast onboard memory
- High throughput on parallel tasks (program exec on each data fragment)

→ GPUs are great for data parallelism

FPGA

Hardware reprogrammable = extreme flexibility
- fine tuning Serial (Pipeline/Streaming) vs Parallel task balance
- Very Low latency for real-time applications
  - mission-critical applications (eg autonomous vehicles)
  - real-time stock-trading
  - online searches
  - TDAQ systems in High Energy Particle Physics, ...
- Fine grained (bit level) operations
- Power saving

→ FPGA are great for accelerating throughput for targeted functions in compute- and data-intensive workloads

1. Processors: hardware accelerators

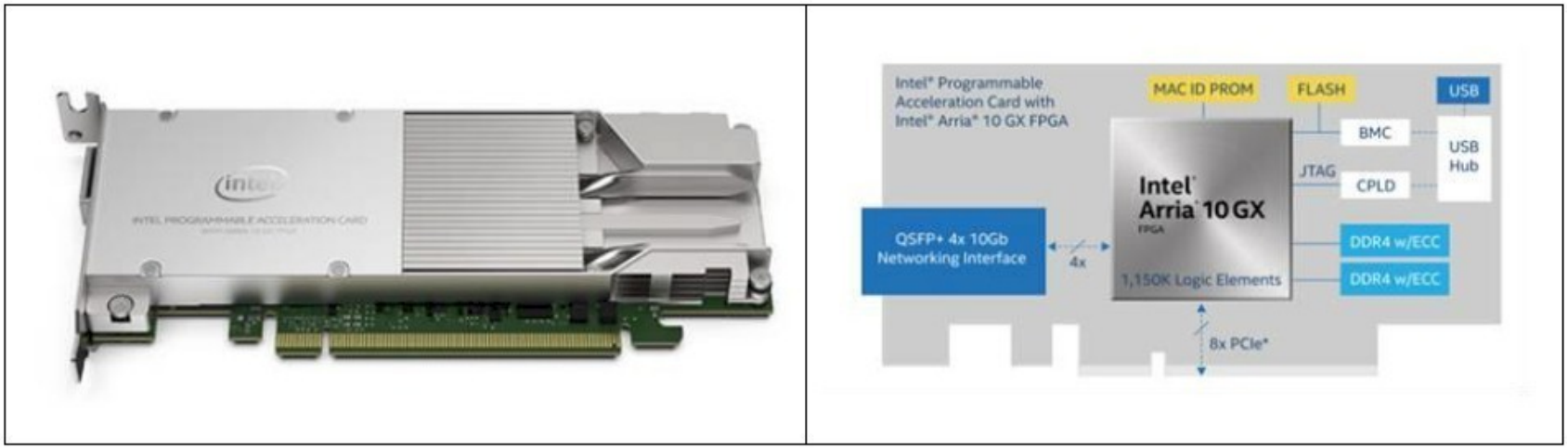
GPU and FPGA are deployed alonside CPUs as HW accelerators

CPU
- Fast caches
- Branching adaptability
- High performance

→ CPUs are great for task parallelism

NVIDIA "TITAN RTX" **GPU**

Intel Programmable Acceleration Card (PAC) – **FPGA**

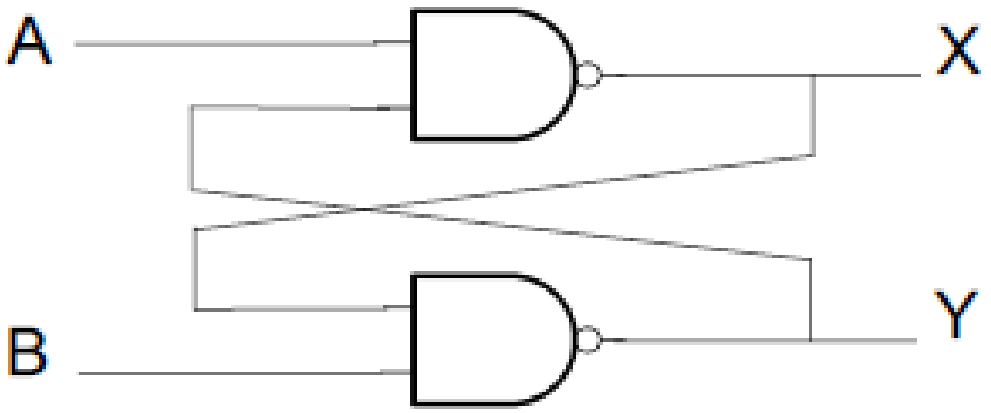


Hardware & Machine Learning

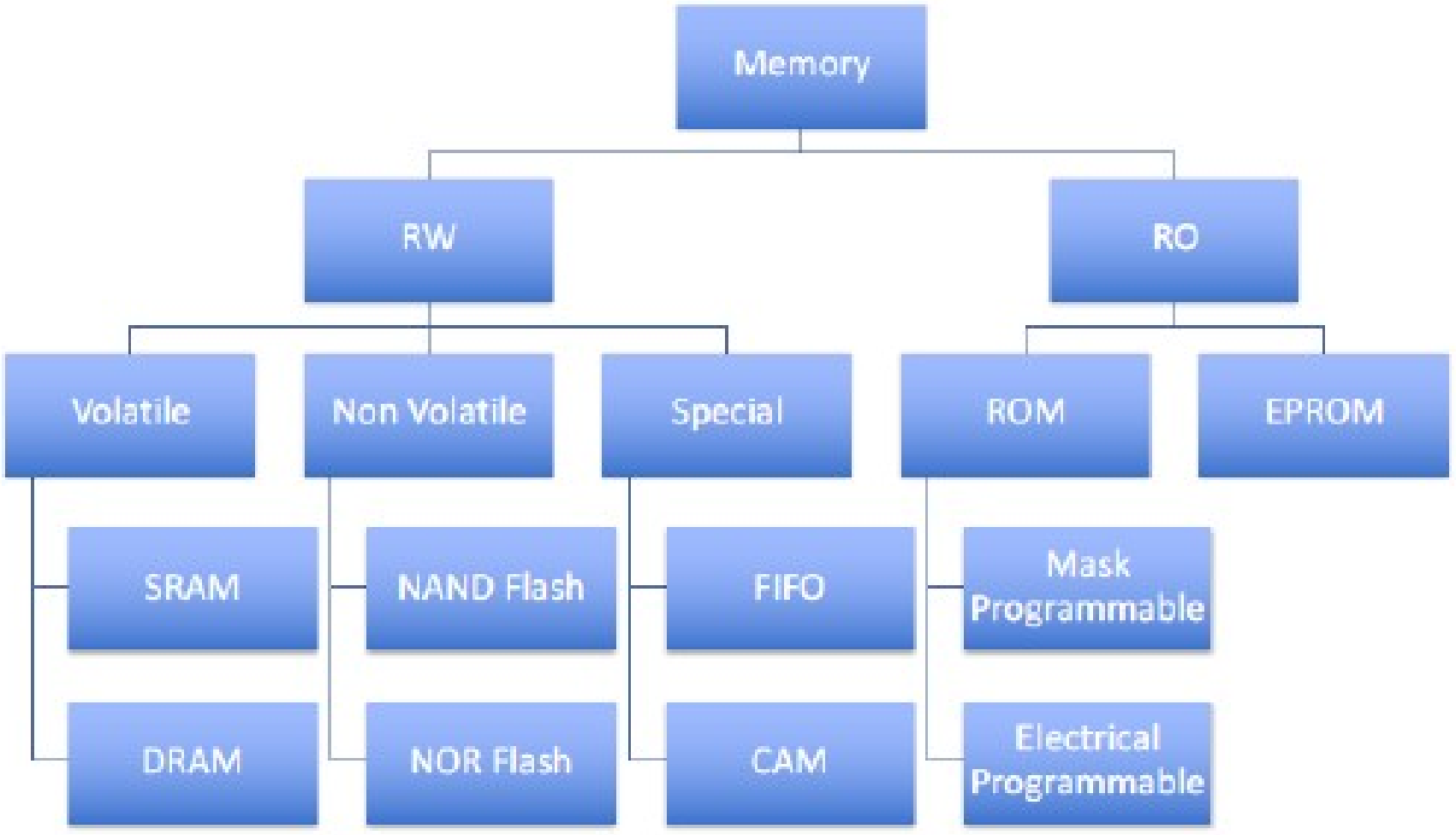
Three latest development boosted "Machine Learning"

- Powerful distributed processors
- Cheap & fast on-board memories
  and cheap & high volume storage
- High bandwidth interconnection
  to bring data to the processors

2. Memory – simple memories



2. Memory zoology



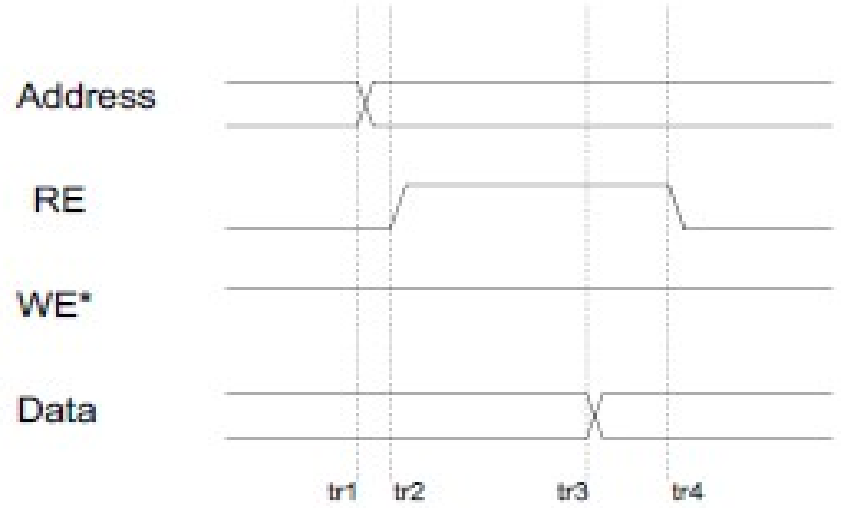
Different storage mechanisms, max capacities and speed...

2. Memory hierarchies in computers

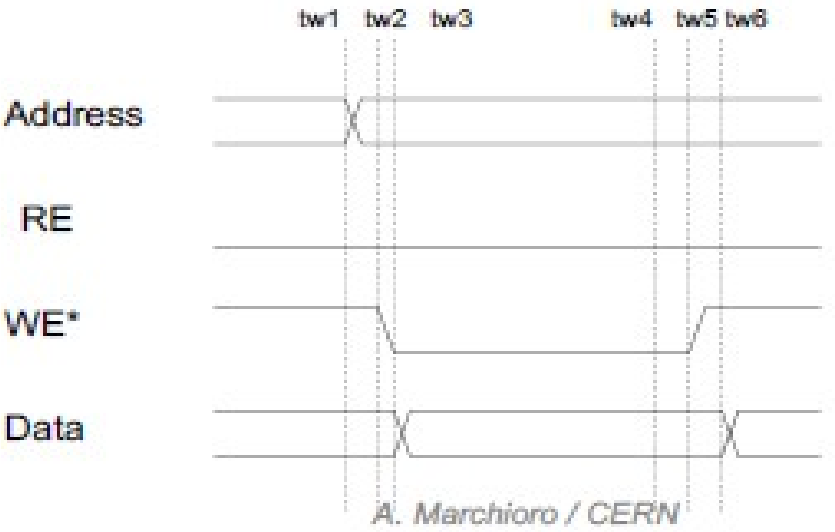
Level	Name	Speed	Size	Type
Level 1	Cache	1-2 Clock cycles (1-5 ns)	10KB-1MB	SRAM
Level 2	Main	10-50 Cycles (20-100 ns)	128 MB-100 GB	DRAM
Level 3	Disk	< 1ms	100GB-10TB	Magnetic- SSD
Level 4	Archive	< 1s	PB	Magnetic

2. Memory timing

Read Cycle

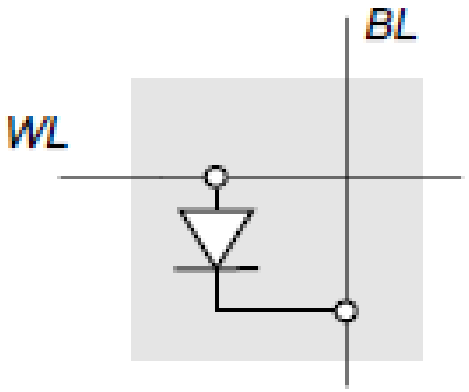


Write Cycle



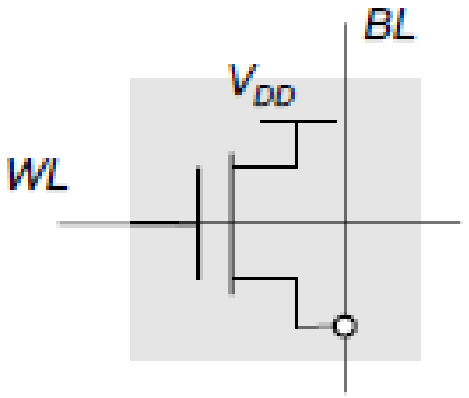
2. Memory – Read Only

1



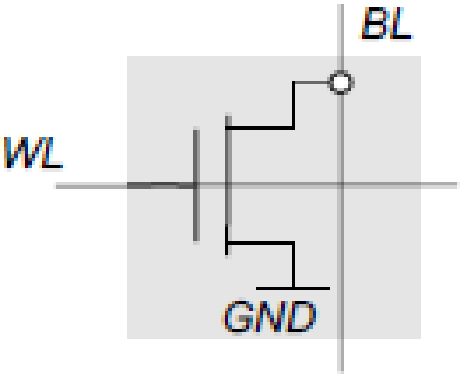
Diode ROM

```
if (WL == 1) BL = 1
```



PMOS ROM

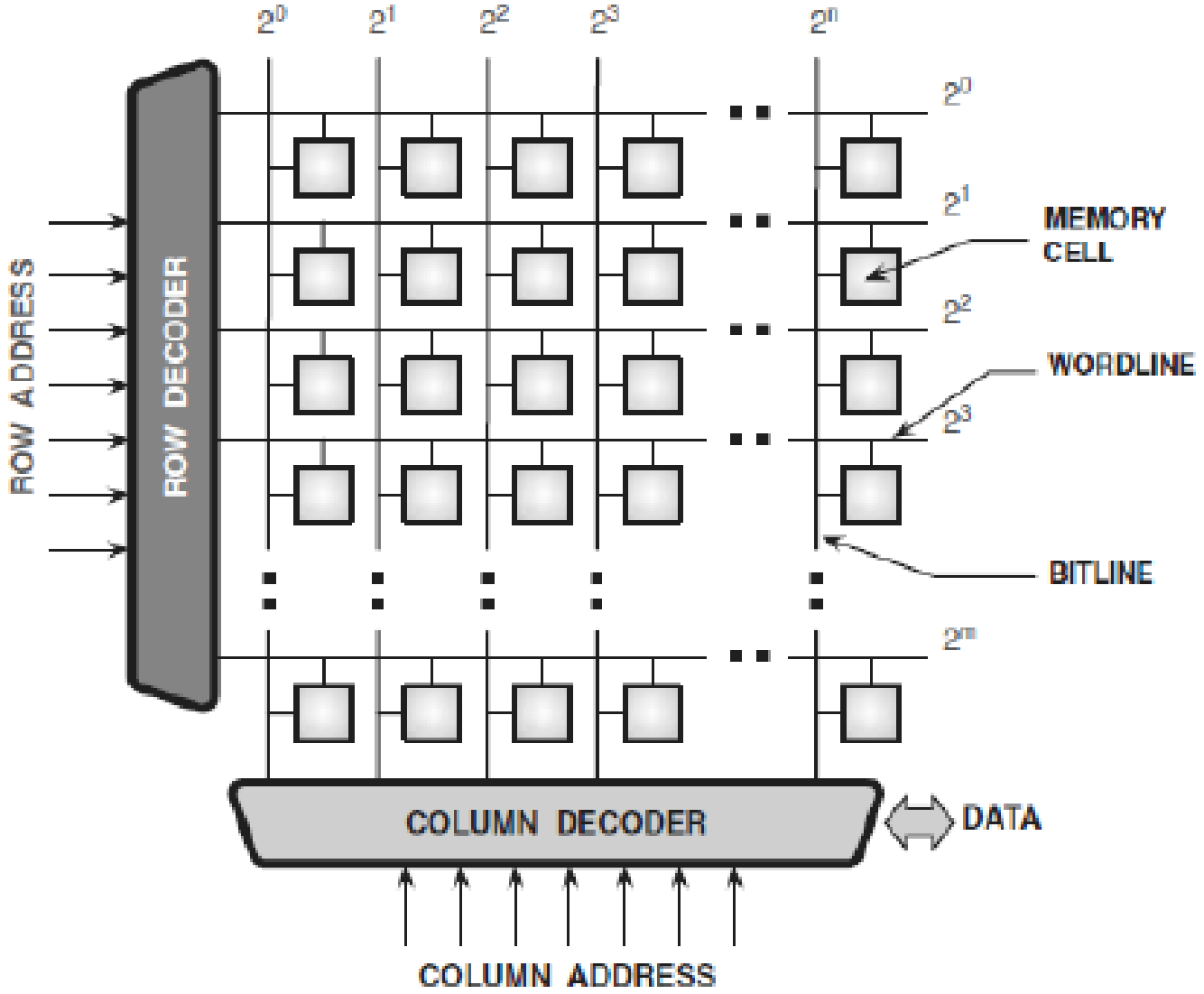
```
if (WL == 0) BL = 1
```



NMOS ROM

```
if (WL == 1) BL = 0
```

2. Memory arrays

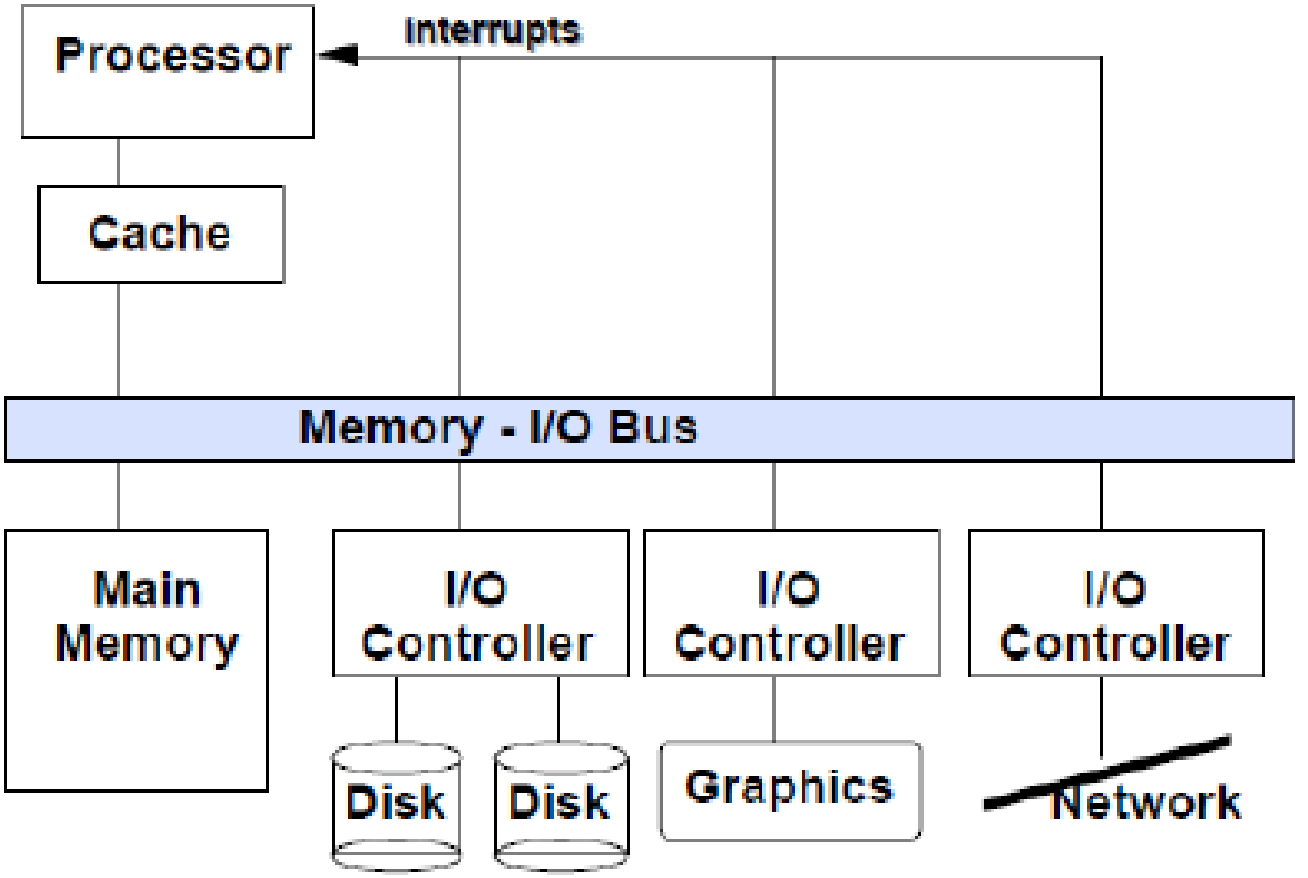


Hardware & Machine Learning

Three latest development boosted "Machine Learning"

- Powerful distributed processors
- Cheap & fast on-board memories
  and cheap & high volume storage
- High bandwidth interconnection
  to bring data to the processors

3. Interconnections: I/O buses

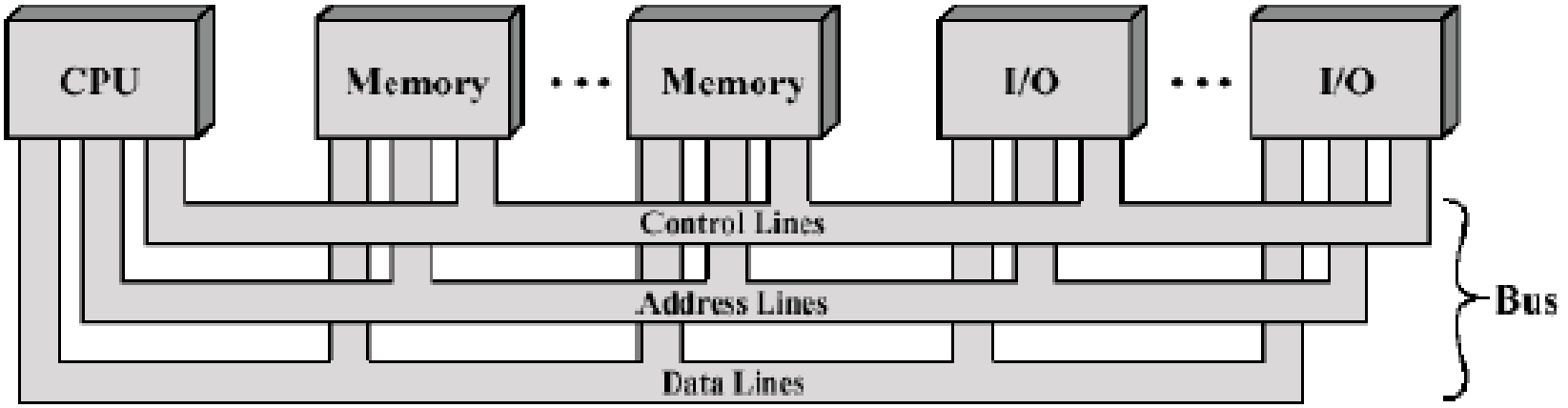


Time(workload) = Time(CPU) + Time(I/O) - Time(Overlap)

3. Interconnections: I/O buses

System interconnection structures may support these transfers:

- **Memory to processor**: the processor reads instructions and data from memory
- **Processor to memory**: the processor writes data to memory
- **I/O to processor**: the processor reads data from I/O device
- **Processor to I/O**: the processor writes data to I/O device
- **I/O to or from memory**: I/O module allowed to exchange data directly with memory without going through the processor - Direct Memory Access (DMA)



Reference: Computer Organization & Architecture (5th Ed), William Stallings

3. Interconnections: I/O buses

- **Address lines**
- **Data lines**
- **Control lines**:
  - **Memory write**: data on the bus written into the addressed location
  - **Memory read**: data from the addressed location placed on the bus
  - **I/O write**: data on the bus output to the addressed I/O port
  - **I/O read**: data from the addressed I/O port placed on the bus
  - **Bus REQ**: indicates a module needs to gain control of the bus
  - **Bus GRANT**: indicates that requesting module has been granted control of the bus
  - **Interrupt REQ**: indicates that an interrupt is pending
  - **Interrupt ACK**: Acknowledges that pending interrupt has been recognized
  - **Reset**: Initializes everything connected to the bus
  - **Clock**: on a synchronous bus, everything is synchronized to this signal

3. Interconnections: fast Links

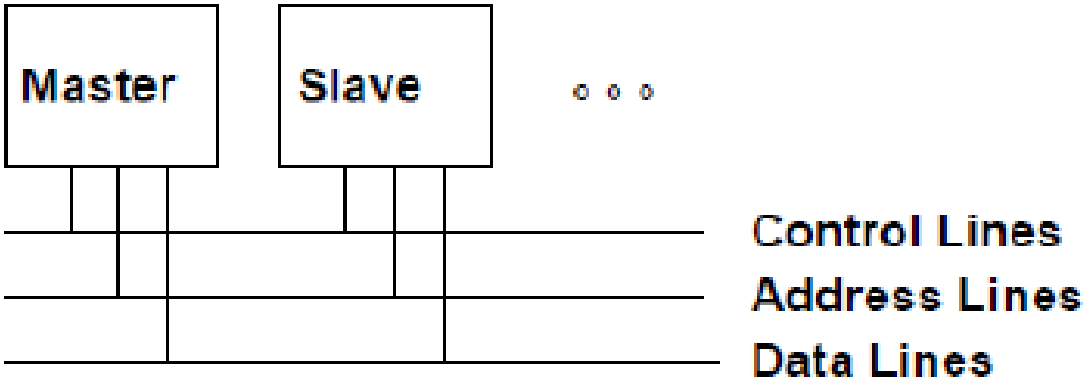
- Interconnect = glue that interfaces computer system components
- High speed hardware interfaces + logical protocols
- Networks, channels, backplanes

	Network	Channel	Backplane
Distance	>1000 m	10 - 100 m	1 m
Bandwidth	10 - 100 Mb/s	40 - 1000 Mb/s	320 - 1000+ Mb/s
Latency	high (>ms)	medium	low (<µs)
Reliability	low Extensive CRC	medium Byte Parity	high Byte Parity

message-based
narrow pathways
distributed arb

memory-mapped
wide pathways
centralized arb

3. Interconnections: protocols



Multibus: 20 address, 16 data, 5 control, 50ns Pause

Bus Master: has ability to control the bus, initiates transaction

Bus Slave: module activated by the transaction

Bus Communication Protocol: specification of sequence of events and timing requirements in transferring information.

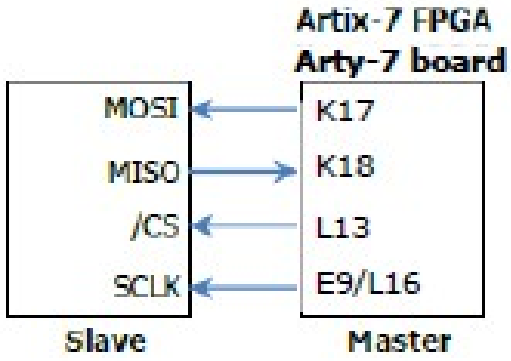
Asynchronous Bus Transfers: control lines (req., ack.) serve to orchestrate sequencing

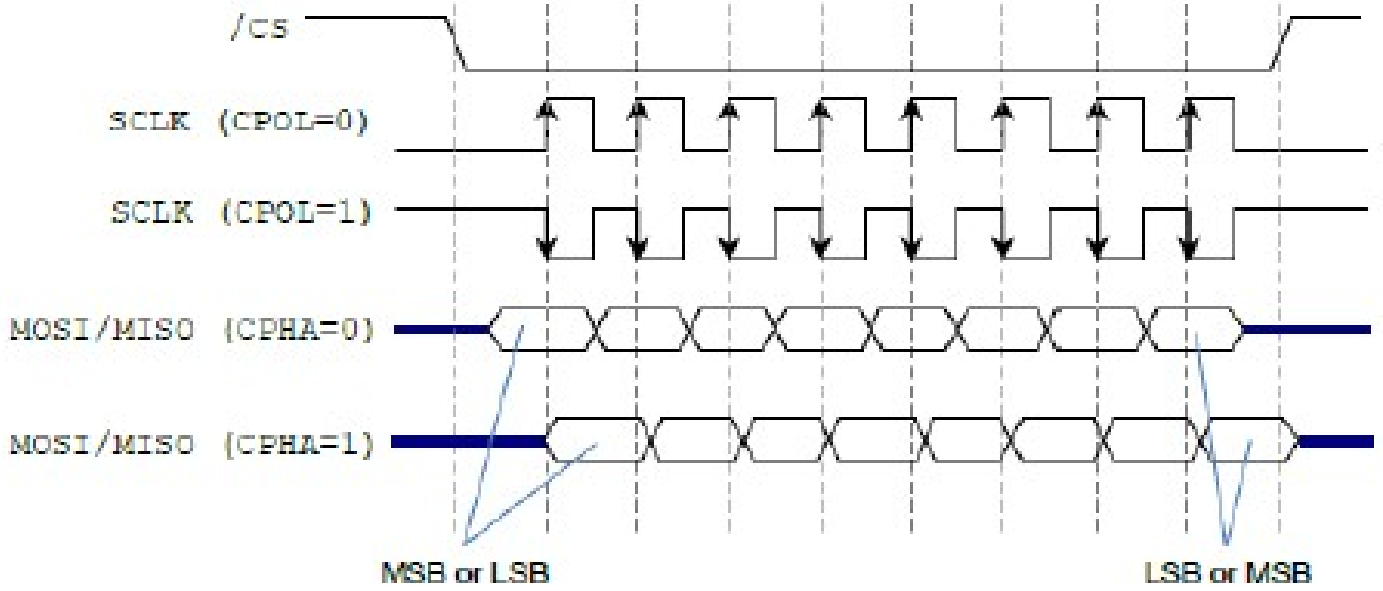
Synchronous Bus Transfers: sequence relative to common clock

3. Interconnections: example SPI

SPI INTERFACE

- Simple 4-wired synchronous (clock is transmitted) serial interface.
- SPI logic signals:
  - ✓ SCLK: Serial clock. Generated by Master.
  - ✓ MOSI: Master Output, Slave Input. Generated by Master.
  - ✓ MISO: Master Input, Slave Output. Generated by Slave.
  - ✓ /CS: Chip select (or Slave Select). Generated by Master.
- Messages are supported that are multiple of 8 bits.
- Clock polarity (CPOL) and Phase (CPHA):

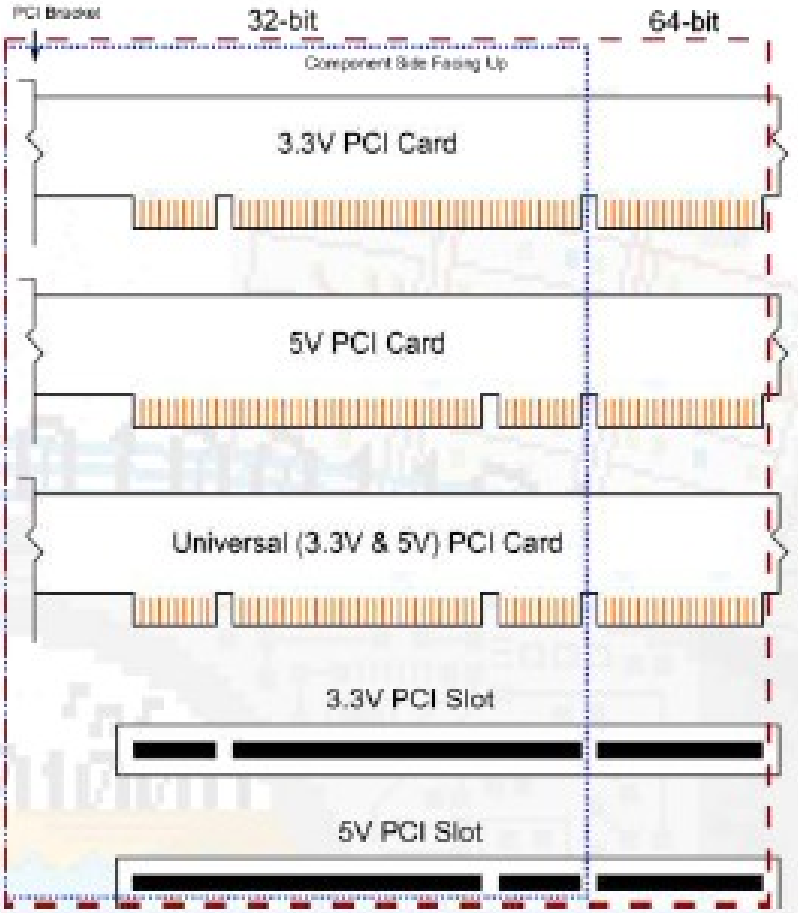




- It is commonly used for short distance communications within embedded systems. Microcontrollers and FPGA designs use SPI to communicate with internal/external peripherals. Large variety of SPI-capable peripherals available: sensors (e.g.: temperature, pressure), ADCs, DACs, touchscreens, memories, LCDs, SD cards.

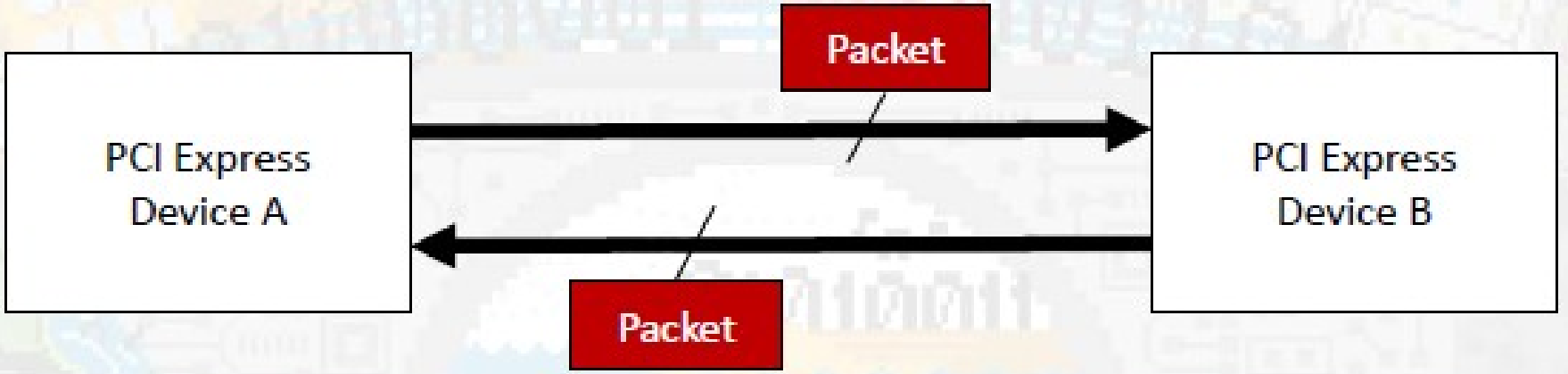
3. Interconnections: PCI

- 1992
- ***P**eripheral **C**omponent **I**nterconnect*
- Parallel Interface
- Bandwidth
  - 133 MB/s (~1.0 Gb/s) (32-bit@33 MHz)
  - 533 MB/s (~4.2 Gb/s) (64-bit@66 MHz)
- Plug-and-Play configuration (BARs)



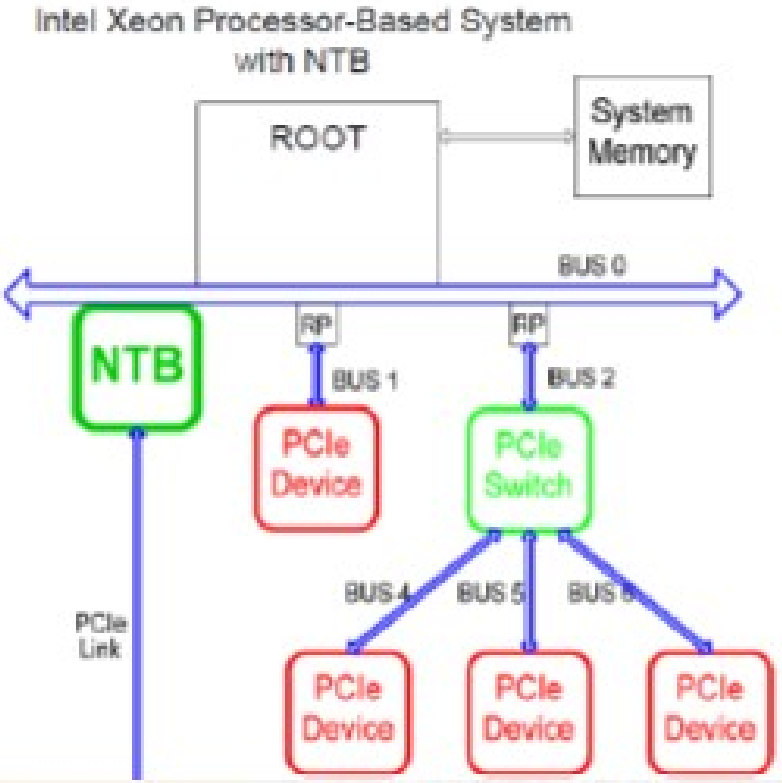
3. Interconnections: PCI-express

- Point-to-point connection
- “Serial” “bus” (fewer pins)
- Scalable link: **x1**, x2, **x4**, **x8**, x12, **x16**, x32
- Packet encapsulation



3. Interconnections: PCI-express

- Connects the processor and memory subsystems to the PCIe fabric via a *Root Port*
- Generates and processes transactions with *Endpoints* on behalf of the processor



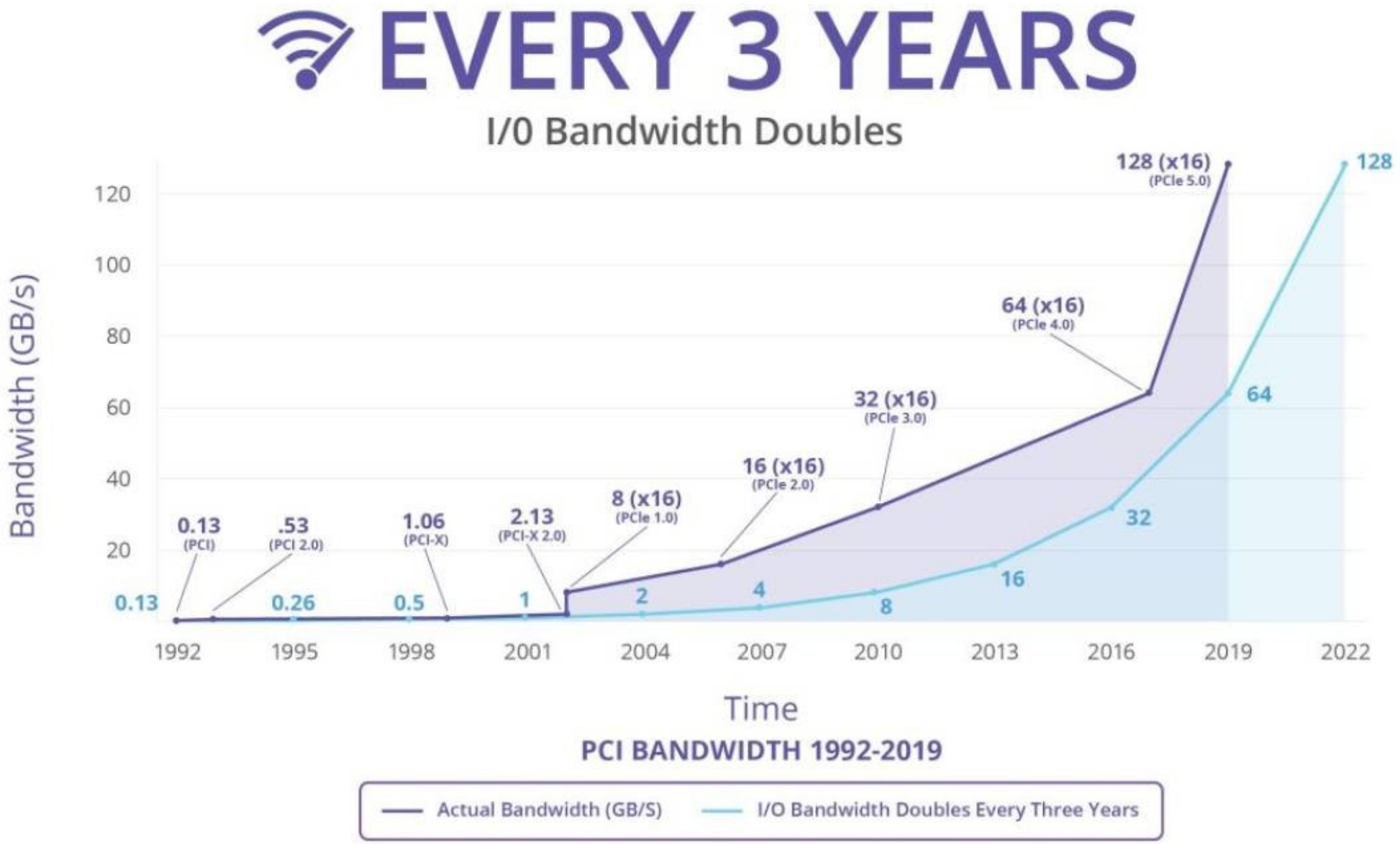
3. Interconnections: example PCI-express

PCI (Peripheral Component Interconnect) Express is a popular standard for high-speed computer expansion overseen by PCI-SIG

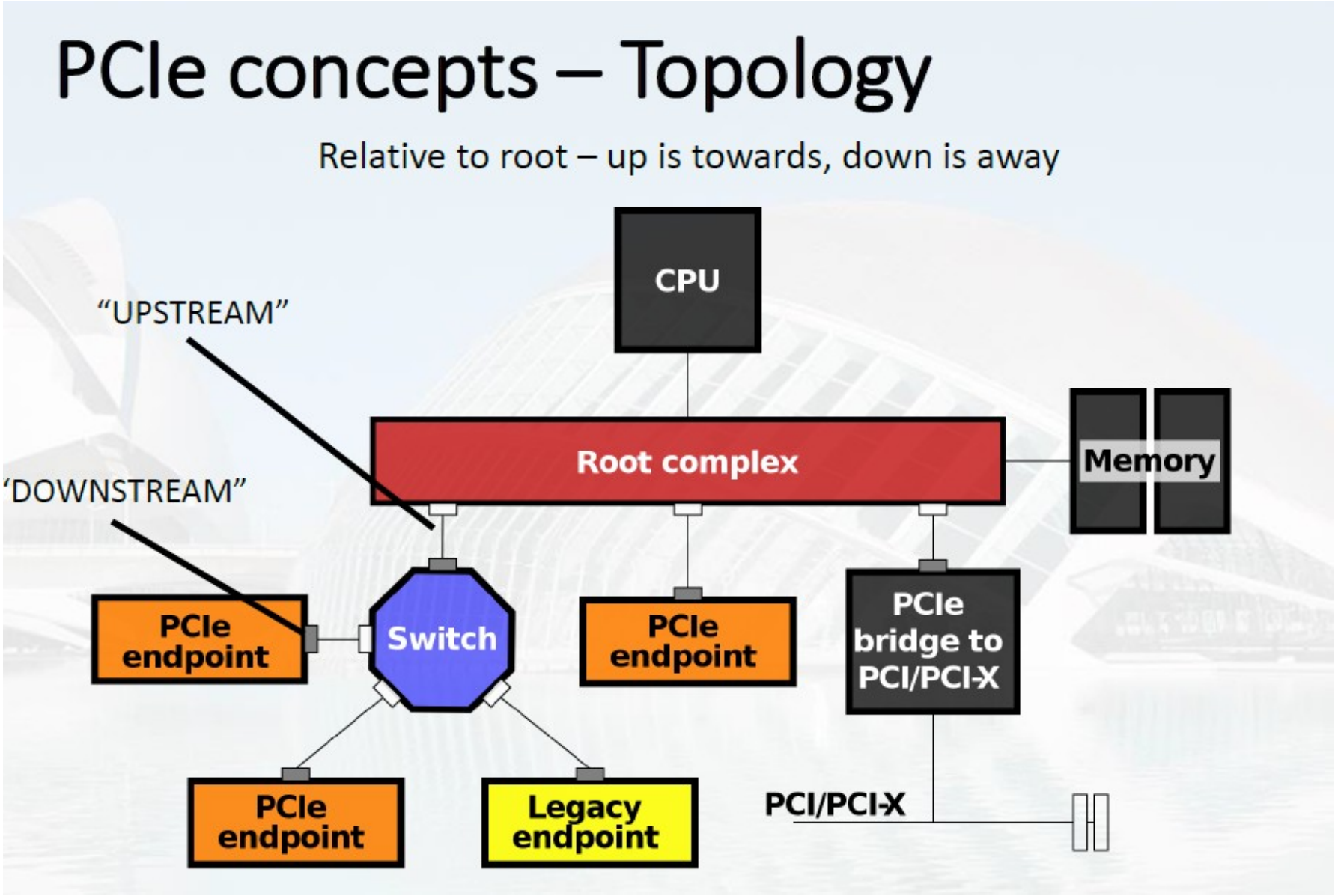
PCI-express in HEP

- Monitoring & Readout
  Several DAQ board at LHC and elsewhere (examples next..)
- Networking
  Ethernet (NIC), Infiniband (HCA), Omni-Path (HFI)
- Storage
  NVMe is the standard interface for high-end Solid-State Disks
- Computation
  Majority of GPGPUs, IBM CAPI (Coherent Accelerator Processor Interface)

PCI-express evolution



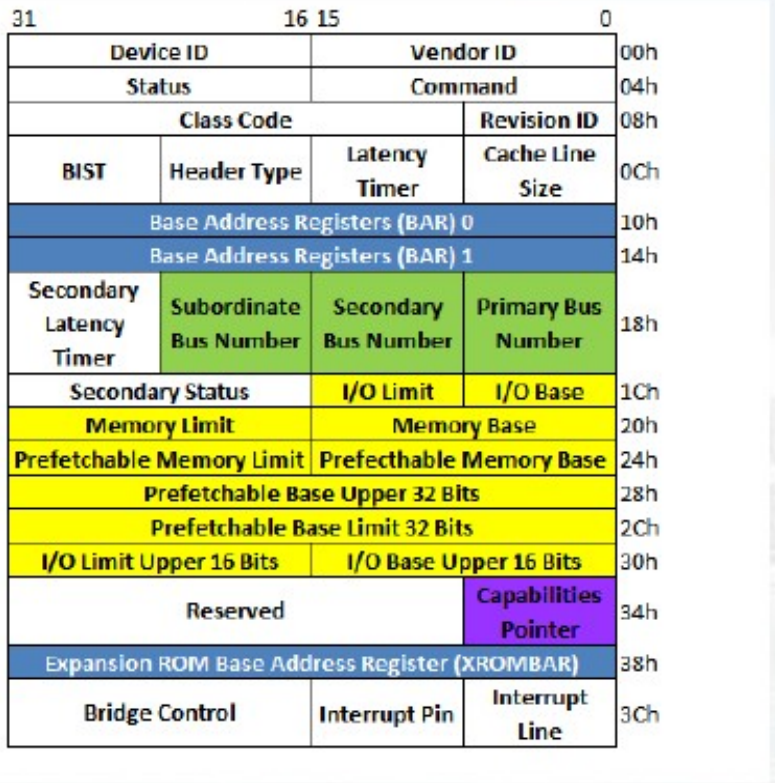
3. Interconnections: PCI-express



3. Interconnections: PCI-express

PCIe concepts – Address spaces

- Address spaces
  - Configuration (Bus/Device/Function)
  - Memory (64-bit)
  - I/O (32-bit)
- Configuration space
  - Base Address Registers (BARs) (32/64-bit)
  - Capabilities (linked list)



31		16 15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
Base Address Registers (BAR) 0				10h
Base Address Registers (BAR) 1				14h
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18h
Secondary Status		I/O Limit	I/O Base	1Ch
Memory Limit		Memory Base		20h
Prefetchable Memory Limit		Prefecthable Memory Base		24h
Prefetchable Base Upper 32 Bits				28h
Prefetchable Base Limit 32 Bits				2Ch
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30h
Reserved			Capabilities Pointer	34h
Expansion ROM Base Address Register (XROMBAR)				38h
Bridge Control		Interrupt Pin	Interrupt Line	3Ch

3. Interconnections: PCI-express

PCIe concepts – Memory & I/O

- Memory space maps cleanly to CPU semantics
  - 32-bits of address space initially
  - 64-bits introduced via Dual-Address Cycles (DAC)
    - Extra period of address time on PCI/PCI-X
    - 4DWORD header in PCI Express
  - Burstable (= Multiple DWORDs)
- I/O space maps cleanly to CPU semantics
  - 32-bits of address space
  - Non-burstable

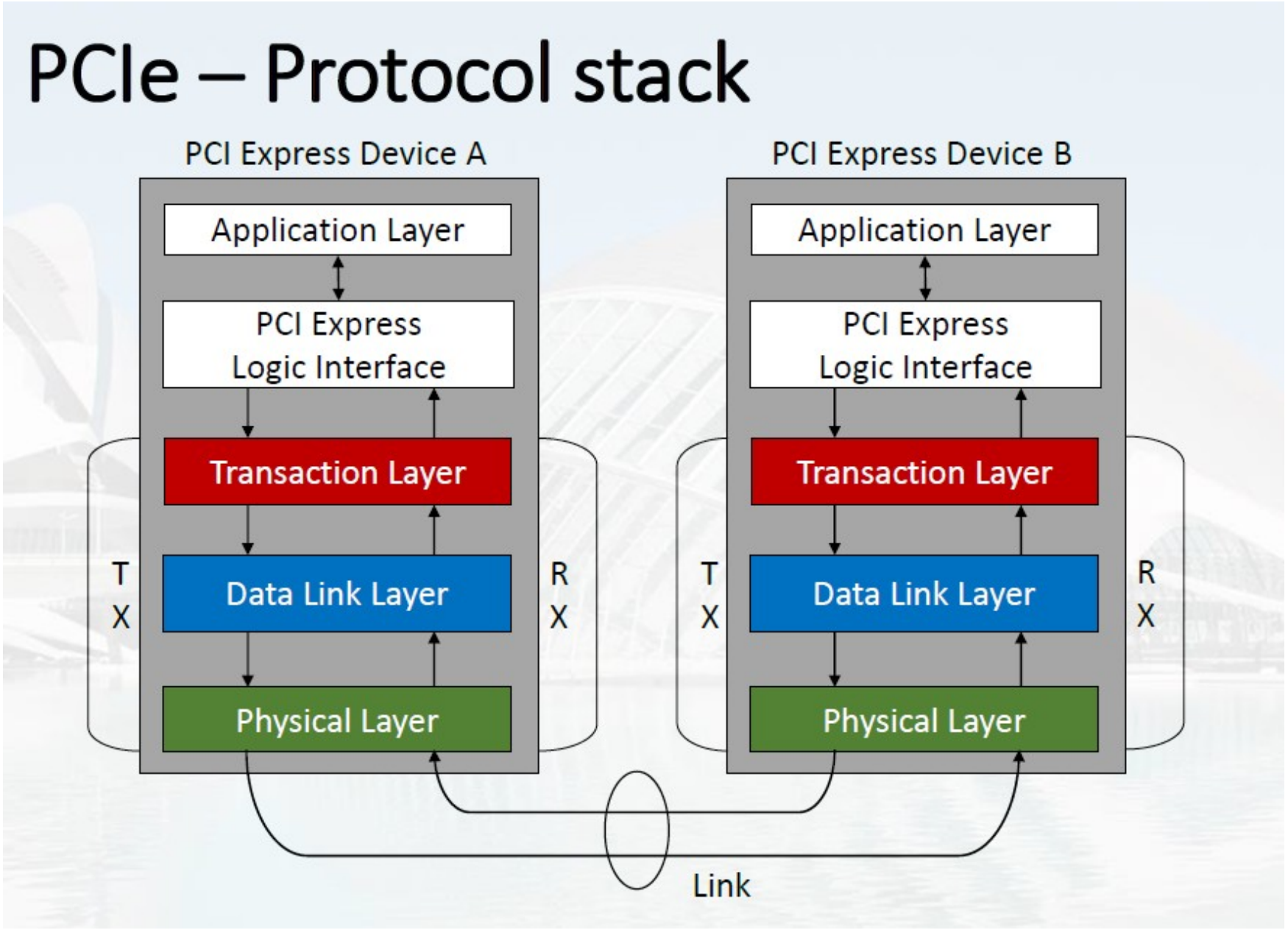
3. Interconnections: Addresses

PCIe concepts – Bus address

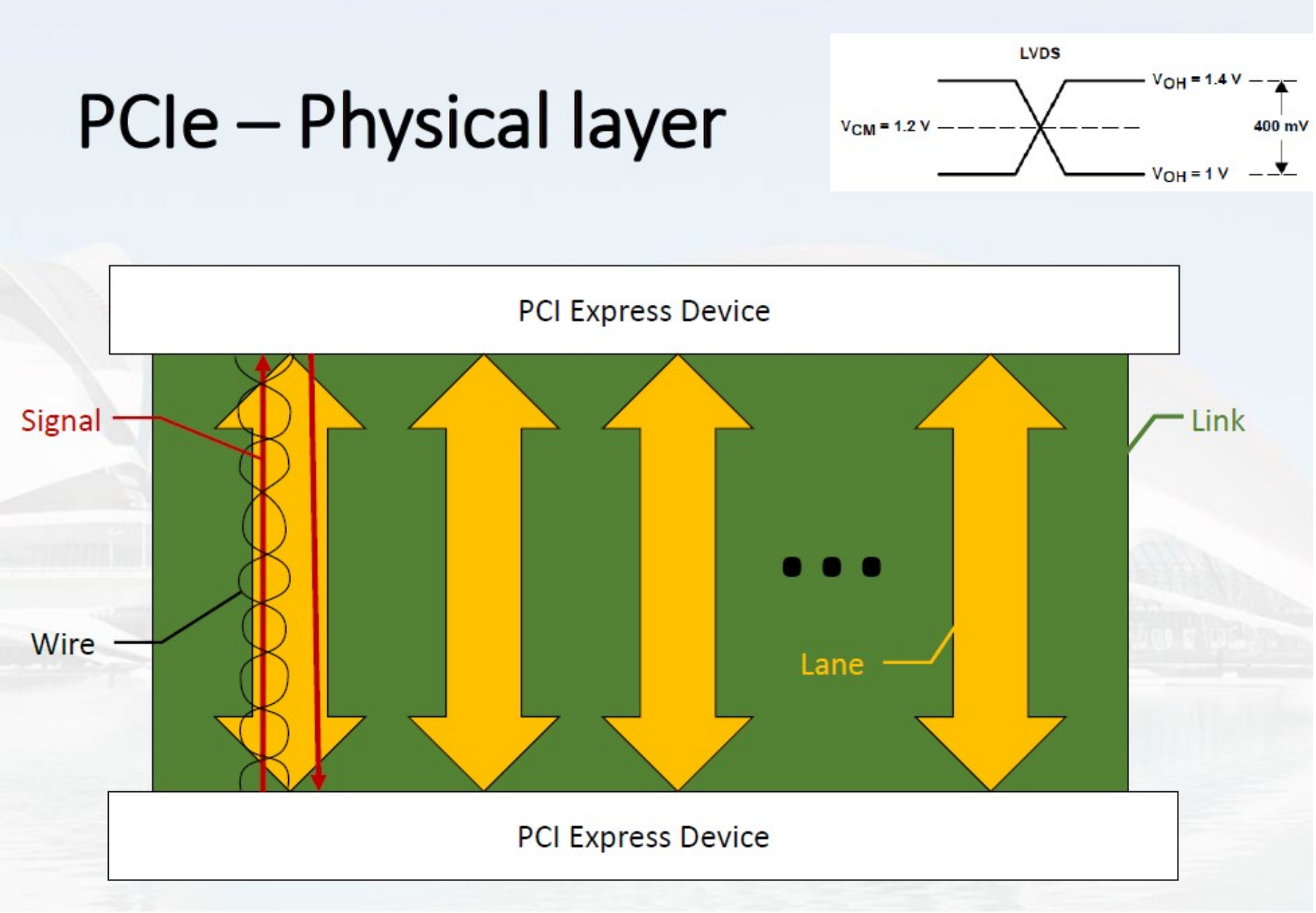
This is actually not specific to PCIe, but a generic reminder:

- Physical address: the address the CPU sends to the memory controller
- Virtual address: an indirect address created by the operating system, translated by the CPU to physical
- Bus address: an address understood by the devices connected to a specific bus
- On Linux, see: *pci_iomap*(), *remap_pfn_range*(), ...

3. Interconnections: PCI-express



3. Interconnections: PCI-express



3. Interconnections: PCI-express

PCIe – Data Link Layer

- ACK / NAK Packets
  - Error handling mechanism
- Flow Control Packets (FCPs)
  - Propagate credit allocation status
- Power Management Packets
- Vendor extensions
  - E.g.: CAPI, CCIX (memory coherency)

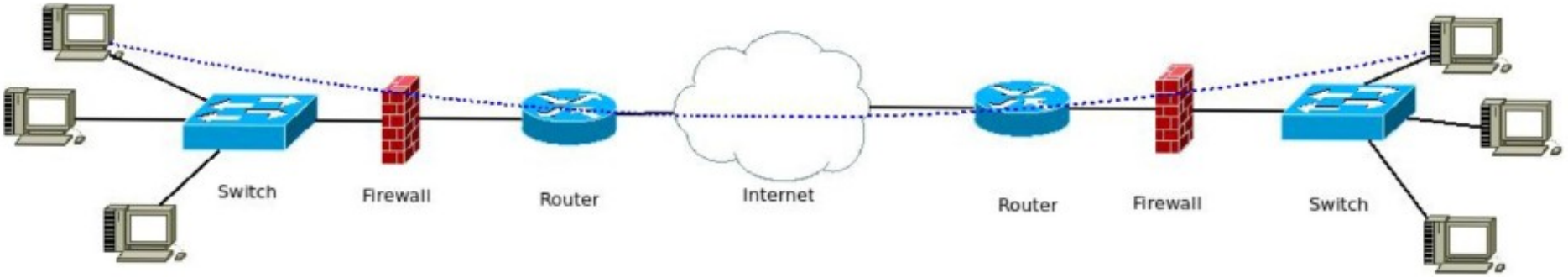
3. Interconnections: PCI-express

PCIe – Transaction layer

- Four possible transaction types
  - **Memory Read | Memory Write**
    - Transfer data from or to a memory mapped location
    - Address routing
  - **IO Read | IO Write**
    - Transfer data from or to an IO location (on a legacy endpoint)
    - Address routing
  - **Config Read | Config Write**
    - Discover device capabilities, status, parameters
    - ID routing (BDF)
  - **Messages**
    - Event signaling

3. Interconnections: Computer Networks

- Computer network consists of end devices, transmission mediums, transit equipment, protocols and applications.
- End devices can run applications that act as a source and generate the data and send it to the destination end devices for consumption.
- Transit devices forward data units between sources and destinations over various transmission mediums.
- Protocols are software constructs that enable the transmission of data units packaged in predefined formats such as frames and packets.



3. Interconnections: Computer Networks

- Networks can be classified by various characteristics, for example:
  - Physical and logical topology
  - Purpose of the network: enterprise networks support some internal activity while carrier networks print money.
  - Size:
    - LAN (Local Area Network)
    - MAN (Metro Area Network)
    - WAN (Wide Area Network)
    - Internet, the network of networks

3. Interconnections: Open System Interconn.

- OSI stands for Open Systems Interconnection, standardized by International organization for Standardization.
- OSI model is split into seven layers that provide the basic concepts for connecting between end devices.

OSI Layer	Example functions
Application	Application protocols such as NTP, SSH etc.
Presentation	Compression, encryption
Session	Authentication, checkpointing
Transport	TCP, UDP
Network	IP addresses, networks
Data-link	MAC and LLC sub-layers
Physical	Cables connectors and signals

3. Interconnections: TPC model

- IP protocol maintained by Internet Engineering Task Force (IETF)
  - *Rough consensus and running code.*
- Only four layers:
  - Network access layer – similar to physical and data link layer of the OSI model, e.g cables and Media Access Control (MAC) addresses reside here.
  - Internet layer – similarly to OSI, this is where IP addresses appear.
  - Transport layer – Similar to the transport layer of the OSI model. Concepts such as protocols (TCP/UDP etc.) belong to transport layer
  - Application layer – houses OSI session, presentation and application layer. Contains the application payload such as DNS, possible encryption and other formatting happens here.

3. Interconnections: TPC model

OSI Model	TCP/IP Model	Contents
Application	Application	DNS,DHCP and other applications
Presentation		Compression, encryption and various conversions
Session		Session establishment/teardown
Transport	Transport	TCP,UDP,ICMP,SCTP etc.
Network	Internet	IP
Data-link	Network access	MAC, Ethernet
Physical		Cables, NICs, optics etc.

3. Interconnections: Ethernet

- Contains technologies from the first two layers of the OSI model → physical and data-link layer
- Uses supposedly unique 6 byte MAC (Media Access Control) addresses to identify, locate and communicate over the network
- Builds broadcast domains where every end devices can talk to every other end device
- Data is encapsulated inside frames 

3. Interconnections: Ethernet

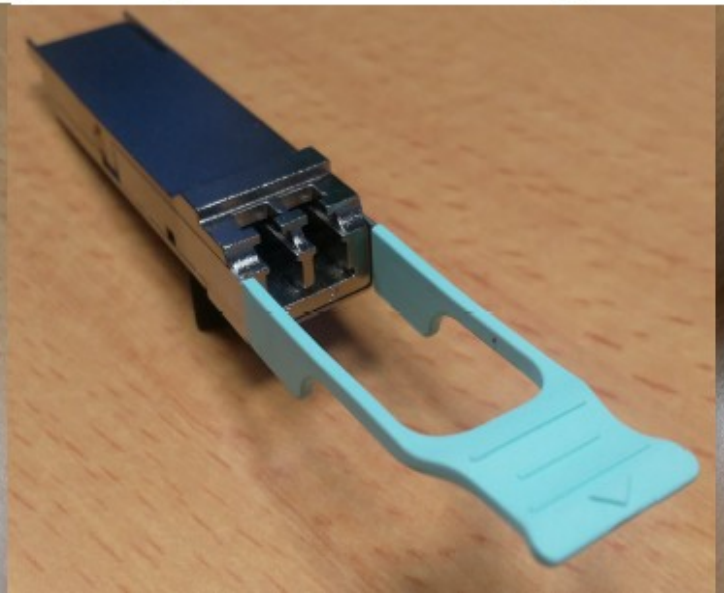
Ethernet frame structure

Preamble (7 B)	SFD (1 B)	Destination MAC (6 B)	Source MAC (6 B)	802.1Q (4 B)	Ethertype/ Length (2 B)	Payload (46-1500 B)	CRC/FCS (4 B)

- Preamble is a specific sequence that allows hardware to detect the new frame.
- Start frame delimiter tells that the destination address begins from the next byte.
- Destination and source MAC addresses follow the preamble.
- Non mandatory 802.1Q VLAN tag is used to identify or set the correct broadcast domain / VLAN for the frame: number between 0 and 4095.
- Ethertype signals the payload type that is encapsulated in the frame: 0x0800 for IP and 0x8100 for VLAN tagged frame.
- Payload is the actual application data. Larger than 1500 Byte payloads are also possible, but 1500 is the default.
- CRC checksums / frame check sequences allow the receiver to check that the frame arrived intact.

3. Interconnections: Ethernet implementation

- Ethernet comes in different shapes and speeds. Original experimental version had bandwidth of 2.94Mbps, whereas today we have 400Gbps available. More common variants being 1Gbps and 10Gbps.
- Commonly Ethernet can use either electric medium (copper cable) or optical (single mode or multimode). Optical mediums tend to support higher speeds over longer distances.

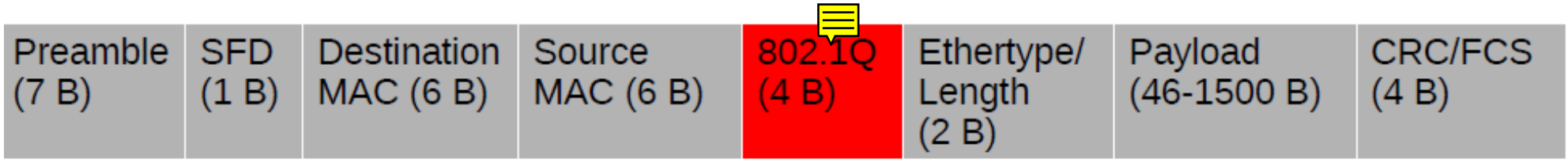




3. Interconnections: Ethernet switch

- Ethernet switch is a OSI layer two device that splits collision domains to allow full duplex operation.
- Switch monitors incoming traffic to learn MAC addresses to build MAC address table. This table is stored to a CAM (content addressable memory) and actual switching is (usually) done by an ASIC.
- MAC address table is used to transport traffic out via the correct port.
- Traffic towards unknown destinations is sent out of all but the original ingress port similarly to traffic with FF:FF:FF:FF:FF:FF destination address (Ethernet broadcast address). This behavior is essential concept in few of the more interesting malfunctions that can happen on an Ethernet network.

3. Interconnections: Ethernet VLAN

- Virtual LANs (VLAN) can be used to logically divide the switch into multiple broadcast domains.
- VLANs can share a medium and span multiple switches using VLAN tagging in a form of a 802.1Q header in the frame.



- VLANs have several benefits:
  - Reduced size of broadcast domains leads to less “line noise”.
  - Improved security as it is not trivial to directly communicate with a device in another VLAN.
  - It is possible to logically group end devices based on any characteristic.

3. Interconnections: IP packet structure

Version	Header size	ToS	Total length
Identifier	Flags	Fragment offset	
TTL	Protocol	Checksum	
Source IP			
Destination IP			
IP options			
Payload			

- Version, IPv4 or IPv6
- Header size
- ToS for ECN, DSCP etc. Qos
- Total length
- Identifier, fragmentation.
- Flags, DF
- Fragment offset, reassembly.
- TTL(Time to live)
- Protocol, TCP, UDP.
- Checksum

- Further headers (TCP, UDP etc.) are encapsulated via the payload.
- Think of this recursive encapsulation like sending letters:
- Brain (application) produces the actual payload.
- Person writes it down (presentation layer) in a suitable medium (transport).
- One adds a stamp and address (network layer) on an envelope (data-link).
- Letters moves via mail (physical layer).

3. Interconnections: Subnets

- IP address configuration consists of two essential elements: IP address and subnet mask.
- Subnet mask is used to determine the distribution of network and host address space. By moving bits between network and host portion we can manipulate the network size.
- Using 192.168.1.10/25 as an example:
- The first 25 bits represent the network, leaving the last 7 bits for host portion: 32-25=7
- 2^7=128 (0-127), 128-2 (network address 0 and broadcast 127) = 126 usable host addresses.
- Binary 11111111.11111111.11111111.10000000 turns into 255.255.255.128 in decimal giving the more human friendly subnet mask configuration parameter.

3. Interconnections: Routers

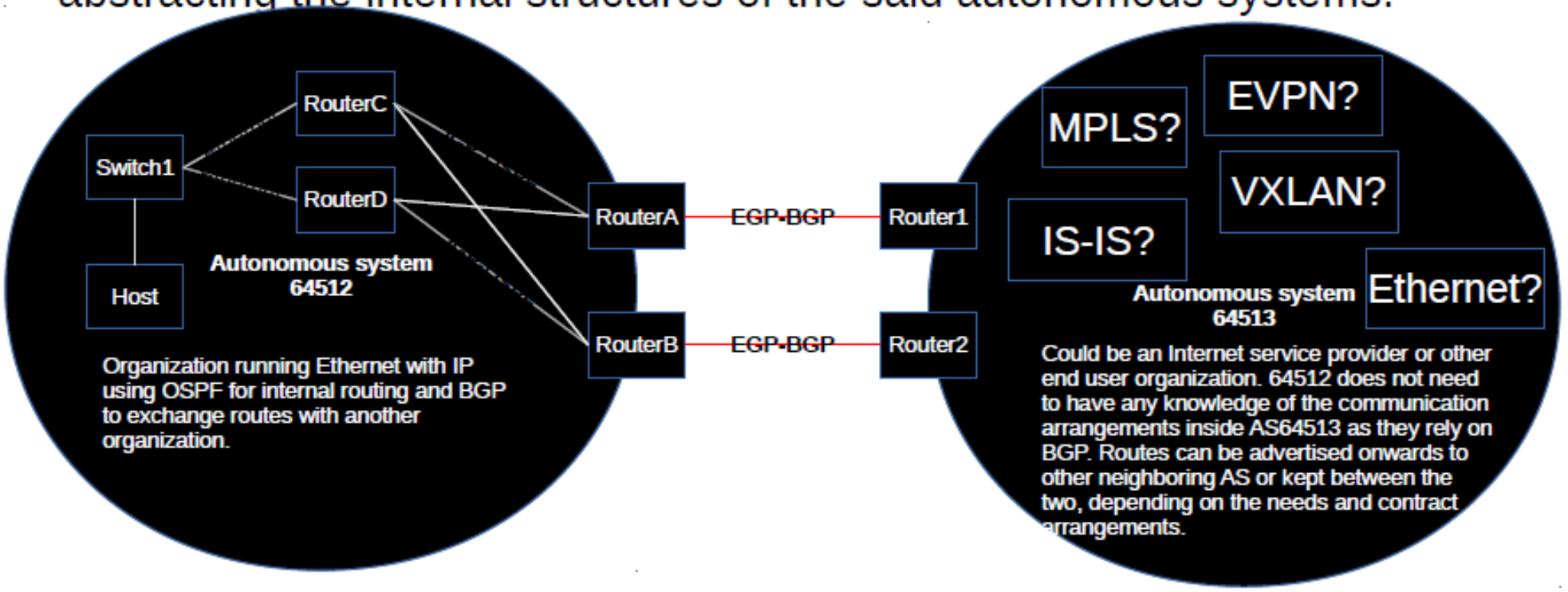
- DAQ networks are interconnected by routers.
- Routers utilize routing tables to build forwarding tables that in-turn are used by forwarding plane ASICs or network processors to forward transit traffic. So, forwarding table is (usually) a subset of the routing table.
- Buffering and storage can be on-chip CAM (fast, but expensive), external DRAM, HBM and the likes (bus speed might become an issue) or mixtures of the two.
- There are two main methods of populating routing tables:
  - Static routing maintained by hand
  - Routing protocols that exchange routing information between routers automatically.
  - Note that routing information is generally programmed using general purpose CPUs on the control plane.

3. Interconnections: IP routing steps

- Simplified steps taken by the network processor on a transit router:
  - Remove the link-layer header
  - Look-up for the destination address from the IP header
  - Possible policing/filtering based on IP and transport layer headers:
    - Protocol, port, flags, TTL, addresses etc.
    - Near line-rate on modern hardware, capabilities depend on hardware
  - Look-up for the destination address from the routing table
  - Look-up of destination link-layer address
  - Add correct link-layer header
  - Place the packet in the transit queue for sending
  - Decrement TTL by one and send – or drop, if congested – the packet to its way.

3. Interconnecting networks

- Ethernet provides LAN connectivity, IGPs provide internal routing and BGP takes care of advertising routes between autonomous systems while abstracting the internal structures of the said autonomous systems.



3. Interconnections: TPC, UDP, sockets

- TCP and UDP are constructs of OSI transport layer

- Protocols that provide end-to-end transmission of data, connections can rely on sockets for addressing, for example: 8.8.8.8:53 UDPsocket points to known Google public DNS resolver at IPv4 address 8.8.8.8, listening on port 53

- Applications open sockets to send data to a known destination socket where another application is hopefully listening

- Sockets come in flavors, for example: streaming socket (TCP), Datagram socket (UDP), Raw socket (ICMP)

- Think of sockets as the interface between application and the network, in terms of OSI layers socket could reside between transport and session layers

3. Interconn. - UDP User Datagram Protocol

- UDP has the following characteristics:
  - Unreliable but guarantees data integrity.
    - Means that it has no mechanism against packet loss, application has to take care of this.
    - UDP packets can arrive in different order than intended and application has to be prepared for this.
  - UDP is connectionless
    - Each packet is independent.
    - No concept of connection setup or tear-down.
    - Supports unicast, multicast, anycast and broadcast.
  - Example applications: (most) DNS queries, SNMP and VXLAN.

3. Interconn. - TCP Transmission Control Protocol

- TCP has the following characteristics:
  - Reliable
    - Data will reach the destination, or TCP will inform of this.
    - Data is delivered in order and integrity is guaranteed.
    - TCP utilizes sequence numbers to keep track of the data stream.
  - TCP is connection oriented
    - Connection is established using three way hand-shake before data is being transferred.
    - Implements sessions supporting both flow control and congestion management.
    - Still, some TCP-based services run in Internet over anycast.
  - Example applications: HTTP, FTP and SSH.

3. Interconn. - Using IP in LAN

- Application decides that it wants to send data to some IP destination, for this end node needs to figure out where to send the data:
  - Lookup on the local routing table to see if the destination address is local or if data has to be sent to a router for forwarding further.
- IP-addressing is a logical construct and relies on Ethernet layer for transport. In IPv4 broadcast based Address Resolution Protocol (ARP) is used to map IP-addresses to MAC-addresses. IPv6 relies on multicast based Neighbor Discovery Protocol (NDP) for similar purpose.
- Source and destination MAC is changed at every network layer transport hop, while IP information stays the same.

Additiona Material