Managment and Analysis of Physical Data – part I a.a. 2019-20

Introduction

G.Collazuol
Dipartimento Fisica e Astronomia UNIPD

- Overview
- Lectures Plan
- Exams

Gianmaria Collazuol

Dipartimento di Fisica ed Astronomia, UniPd
Via Marzolo 8 - Stanza 249
gianmaria.collazuol@unipd.it
tel. 049 827 7134
cell. 0xC802BC08

Main interests:

- EXPERIMENTAL PHYSICS Neutrino Phys, Cosmic Rays, High Energy Particles

- DEVELOPMENT of INNOVATIVE DETECTORs & ELECTRONICS (analog/digital)
  photo-detectors, silicon pixel and gas based detectors

- DIGITAL ELECTRONICS for Trigger and DAQ

Hardware & Machine Learning

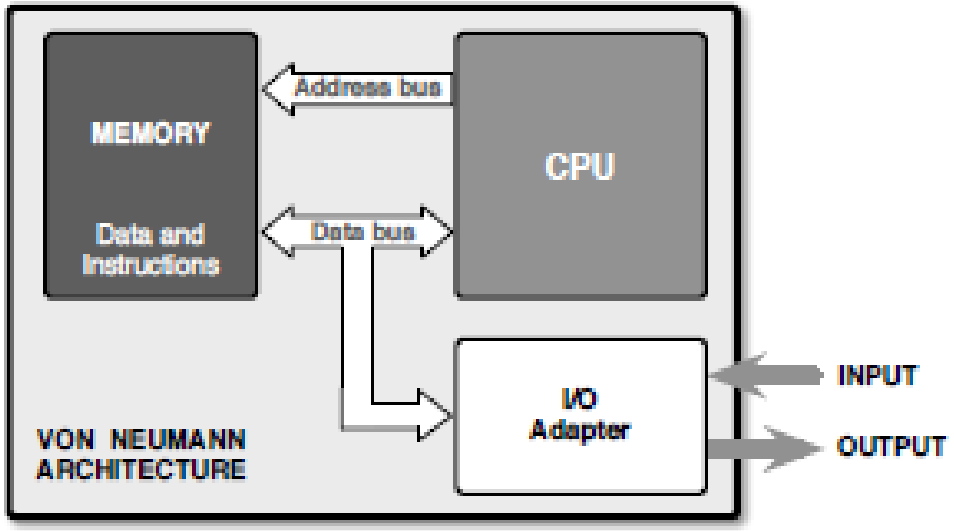
Three latest development boosted "Machine Learning"

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

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.

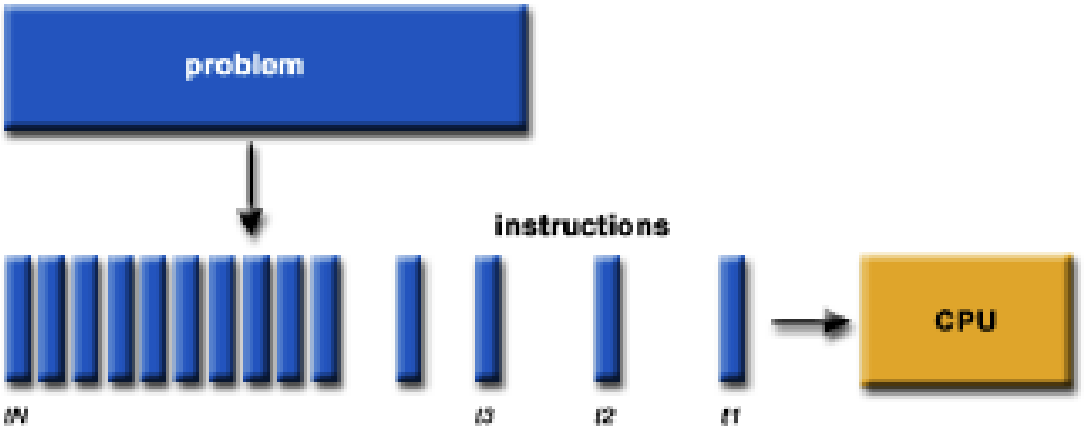
1. Processors
→ Digital architectures with memories

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

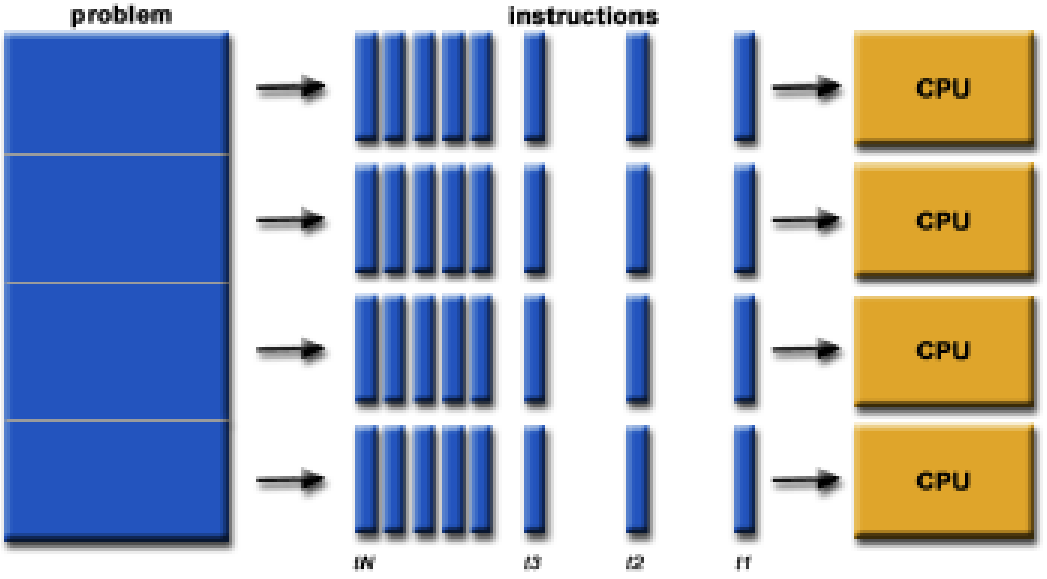
Official-sounding definition: The simultaneous use of multiple compute resources to solve a computational problem.

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

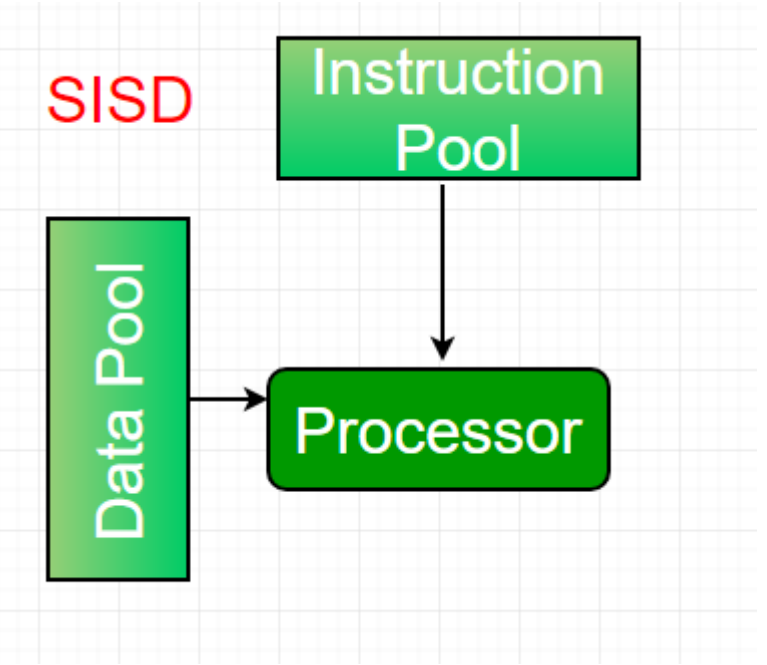


1. Processors → Flynn's taxonomy

Classification of computer architectures, proposed by Michael J. Flynn

- SISD – traditional serial architecture in computers.
- 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

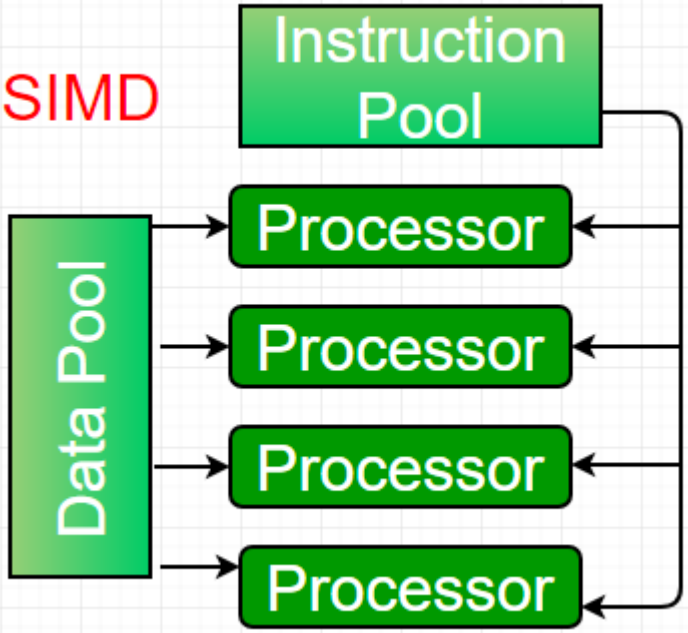


1. Processors → Flynn‘s taxonomy

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

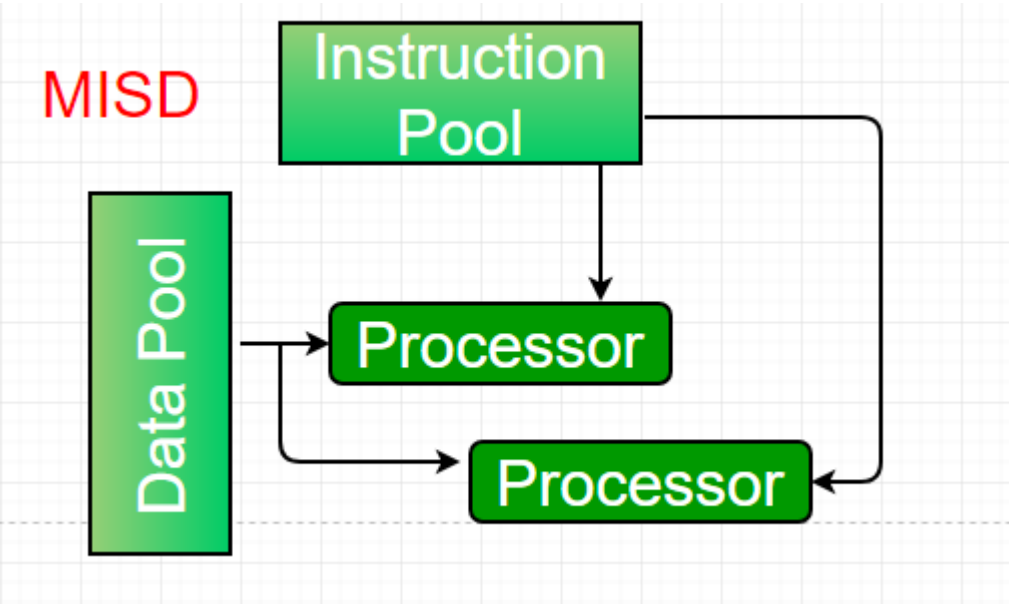


1. Processors → Flynn‘s taxonomy

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.

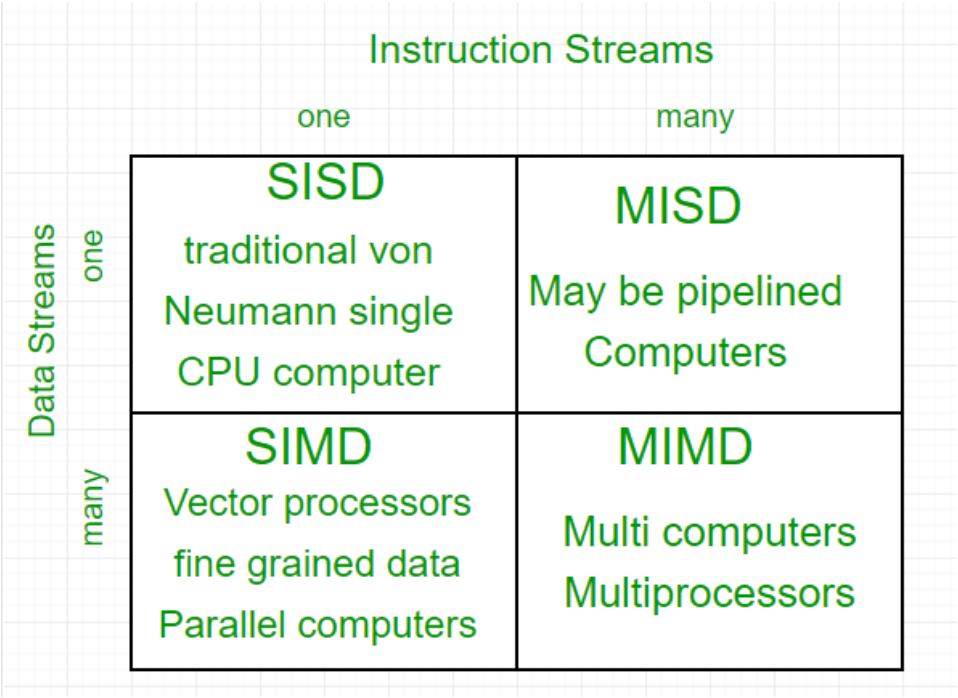
Data Streams \ Instruction Streams	one	many
one	**SISD** traditional von Neumann single CPU computer	**MISD** May be pipelined Computers
many	**SIMD** Vector processors fine grained data Parallel computers	**MIMD** Multi computers Multiprocessors

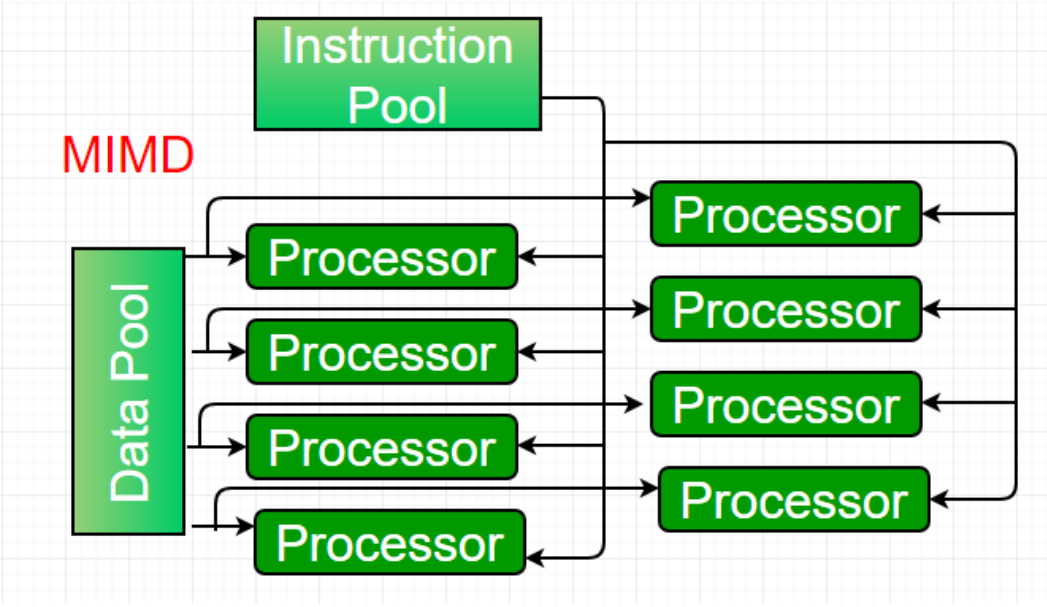


1. Processors → Flynn‘s taxonomy

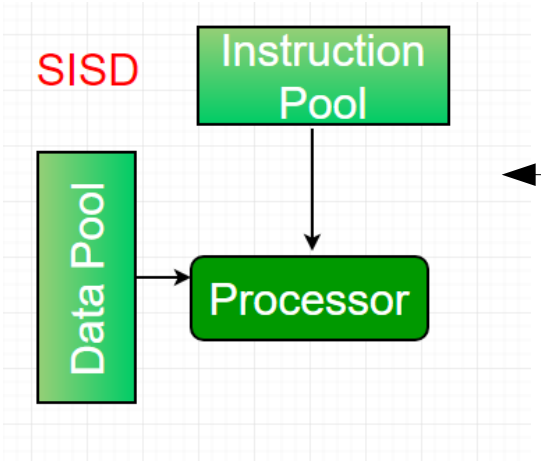
Classification of computer architectures, proposed by Michael J. Flynn

- 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.

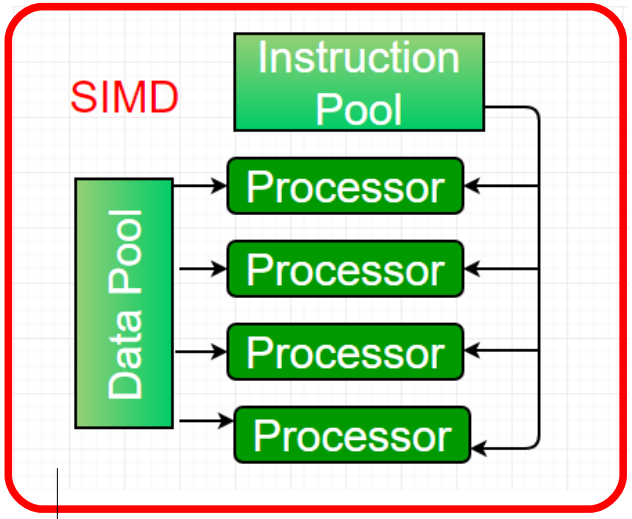


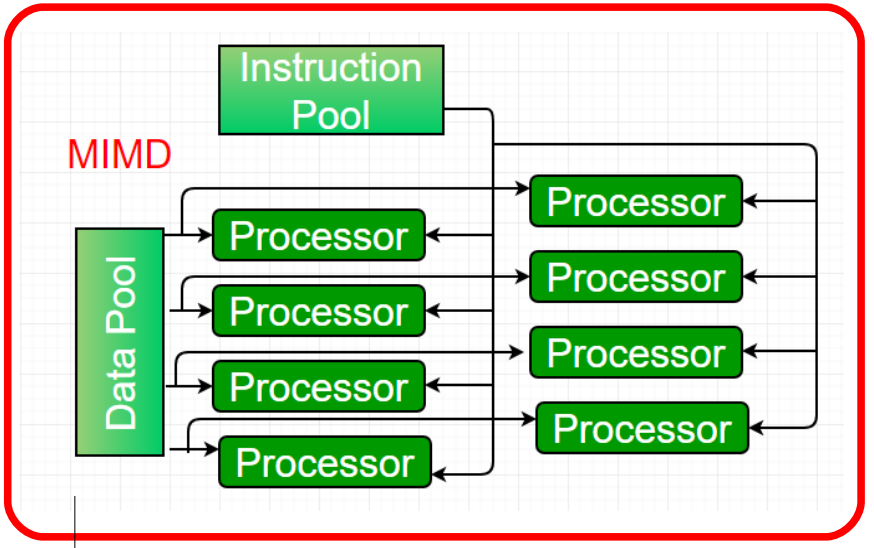


1. Processors → Flynn‘s taxonomy

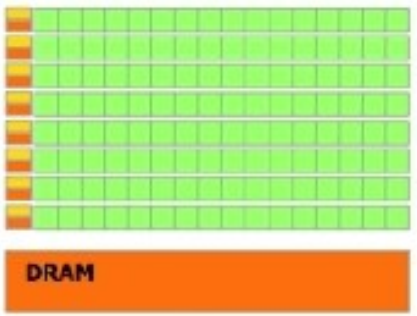


Single core
CPU





many-core
GPU



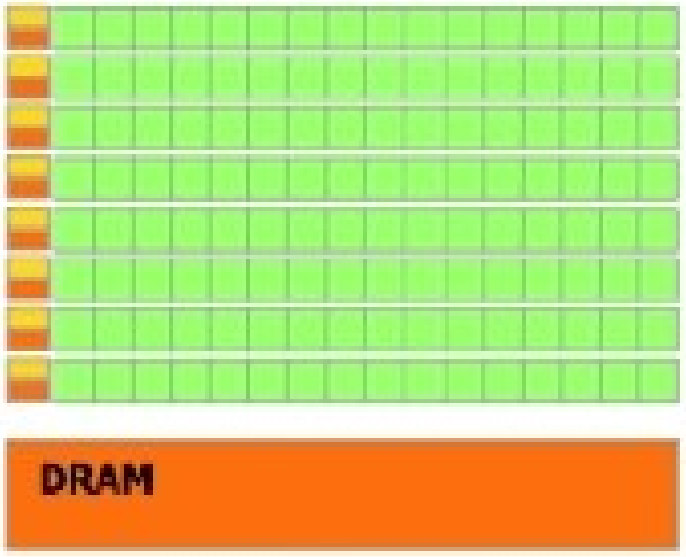
Modern
multi-core
CPU are partially MIMD

1. Processors → GPU structure

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

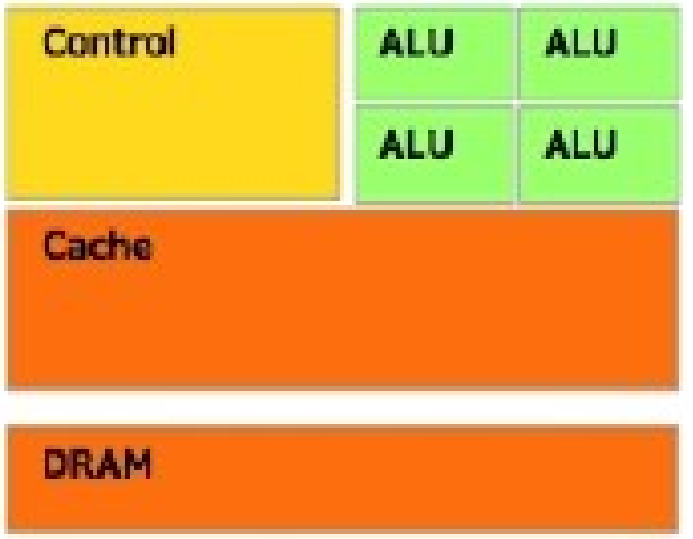
→ rendering real-time graphics: highly computational and memory intensive problem with enormous inherent parallelism

→ 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
```

1. Processors → parallel algorithms

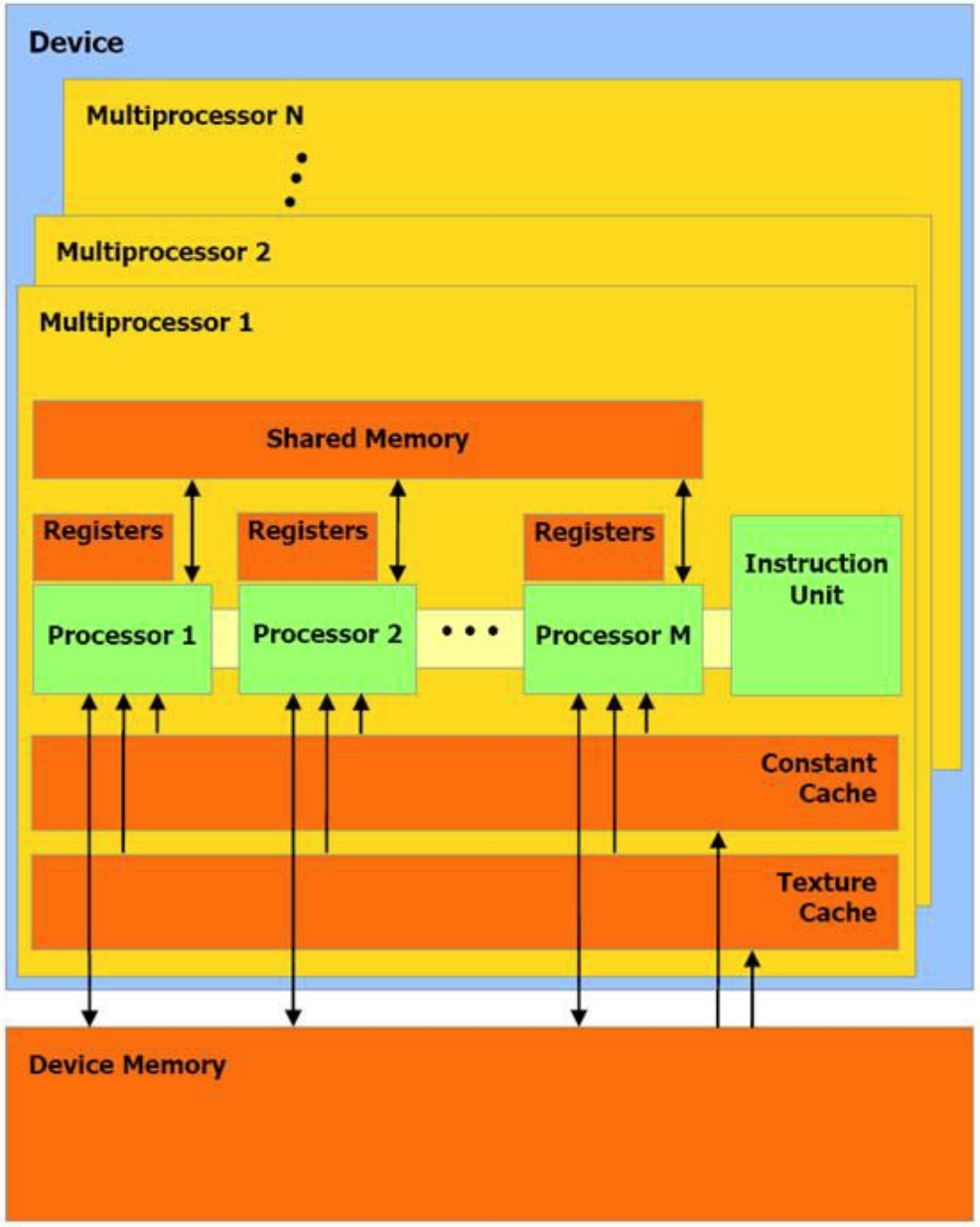
Numerical Computing
→ Fourier transform, Dense matrix operations, Sparse matrix operations, N-body…

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

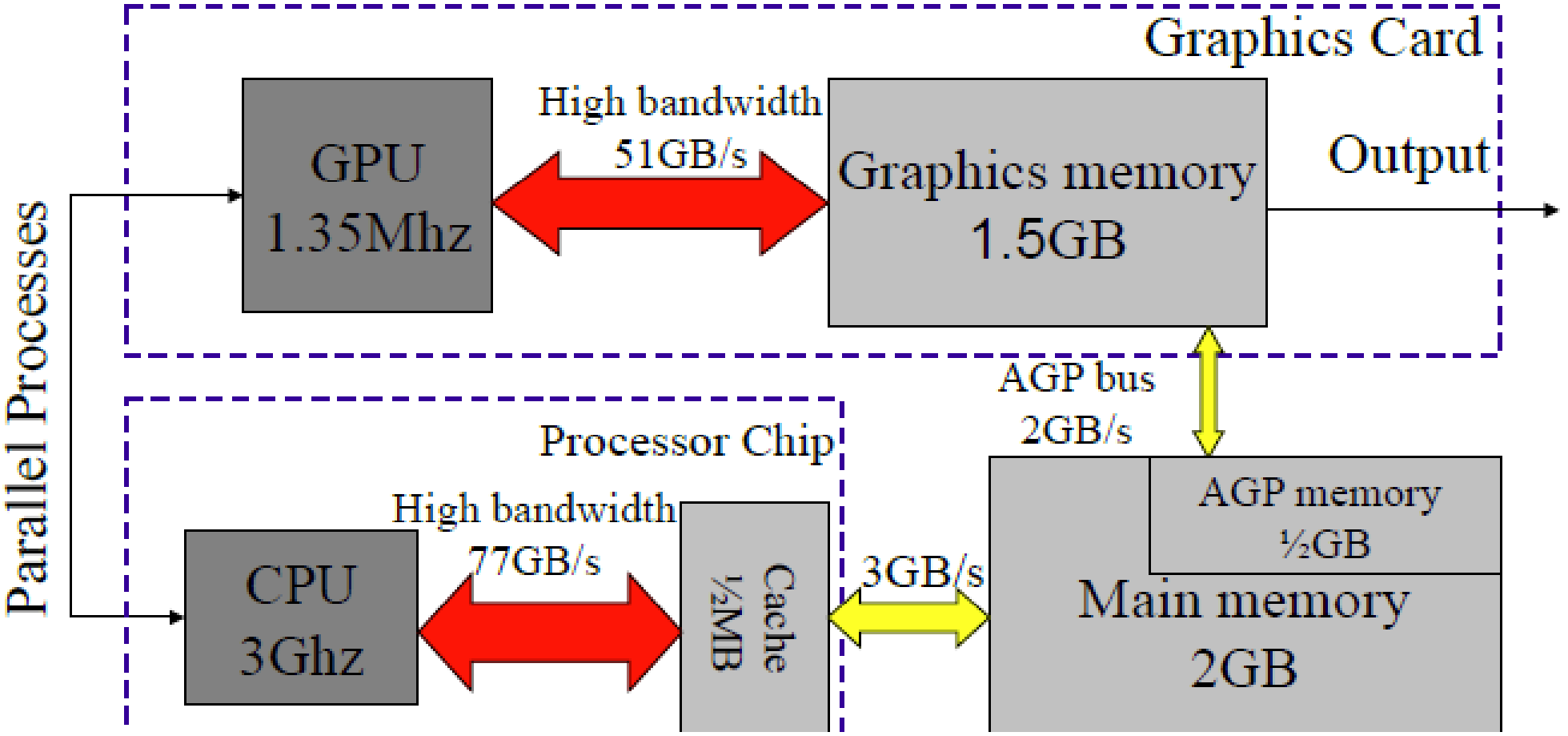
- interconnection to CPU via fast buses (PCI-express)

Memory… 6 types
On-chip / On-board

- Constant Memory
- Texture Memory
- Device Memory

1. Processors → GPU systems

High memory bandwidth

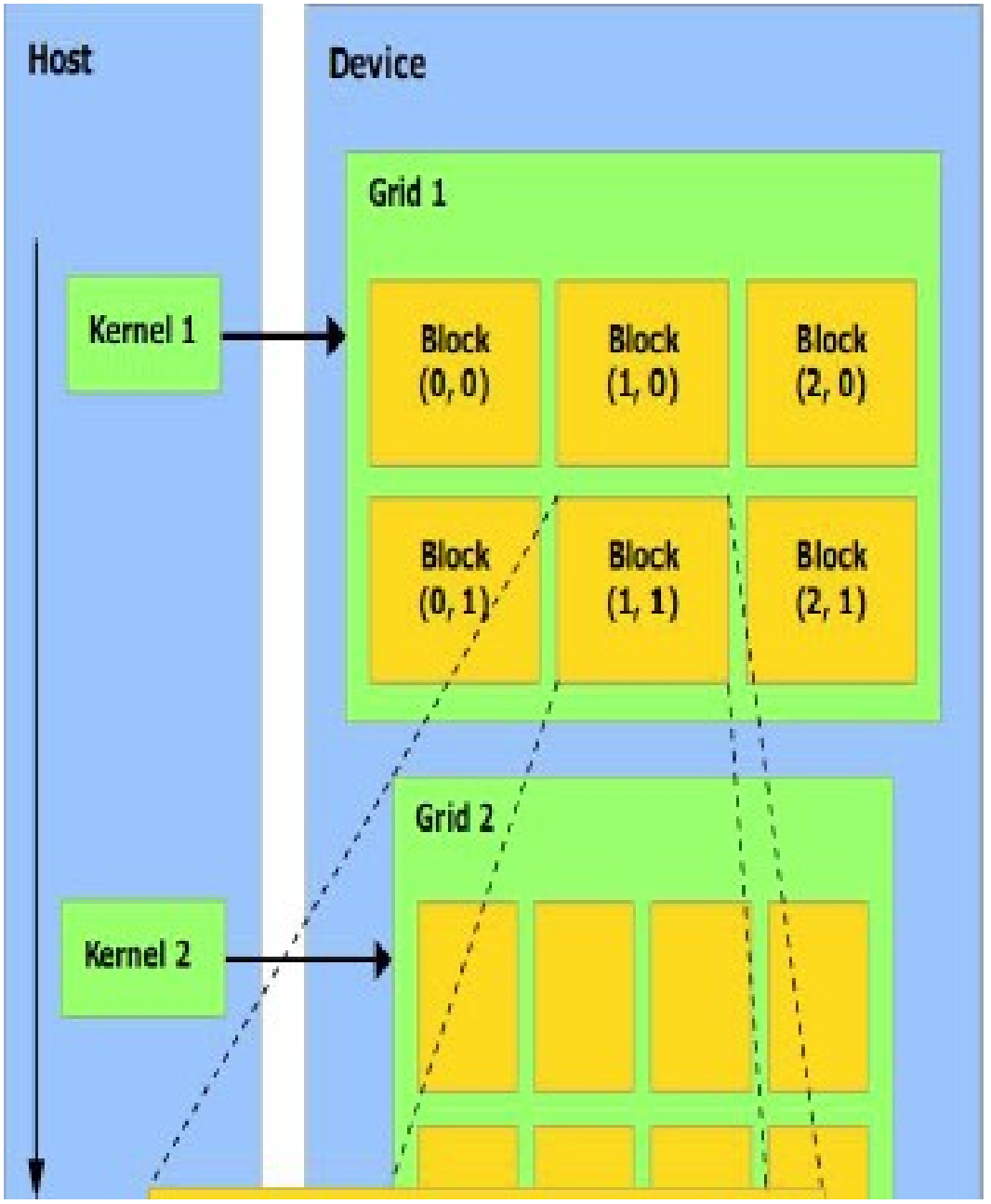


NVIDIA GPU Tesla C870 (2007)

1. Processors → GPU programming

Important Concepts:

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



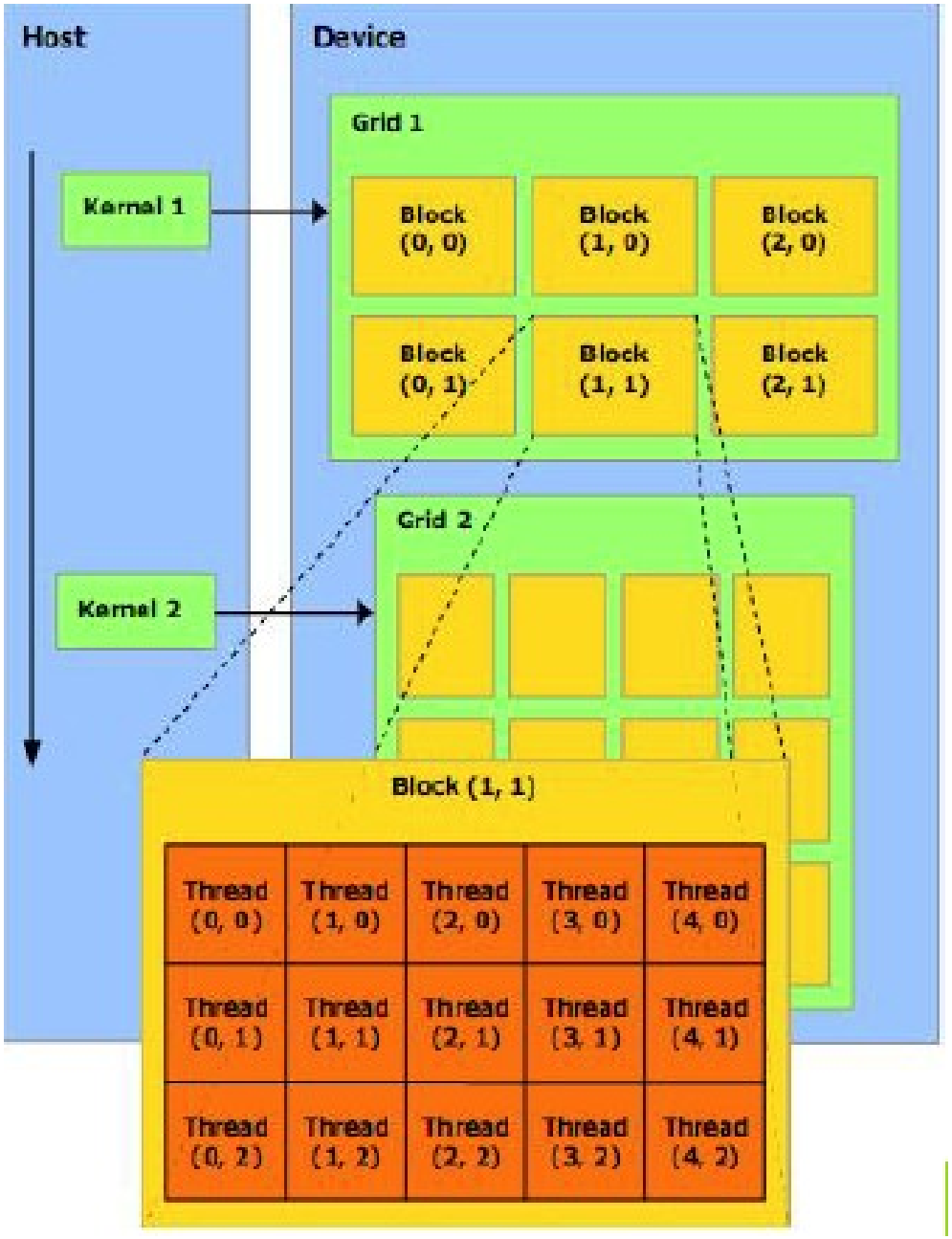
1. Processors → GPU programming

Important Concepts:

- Thread: basic execution unit
- Thread block:

  A batch of thread. Threads in a block cooperate together, efficiently share data. Thread/block have unique id
- Grid:

  A batch of thread block. that excuate same kernel. Threads in different block in the same grid cannot directly communicate with each other



1. Processors → GPU coding

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

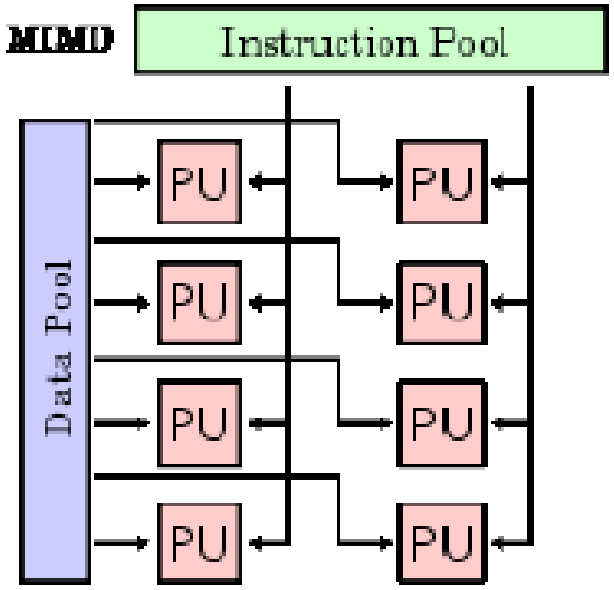
```
Void main()
{
  ...
  // allocation & transfer data to GPU
  //Excuate on N/256 blocks of 256 threads each
  addVector << N/256, 256 >> ( d_A, d_B, d_C);
  .....
}
```

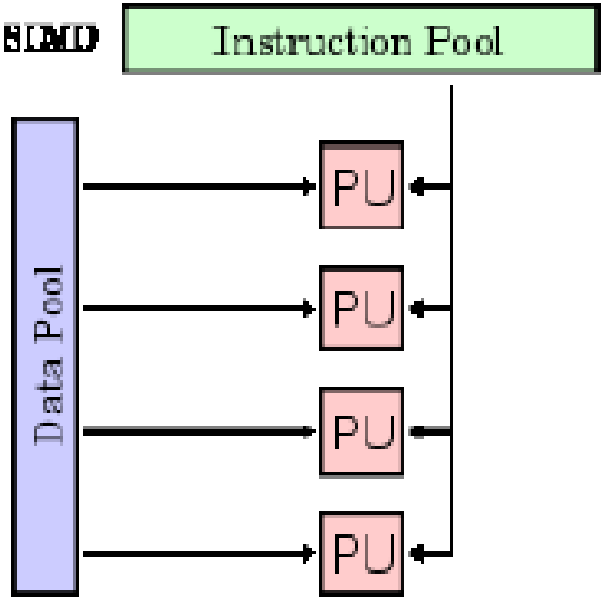
Host code

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

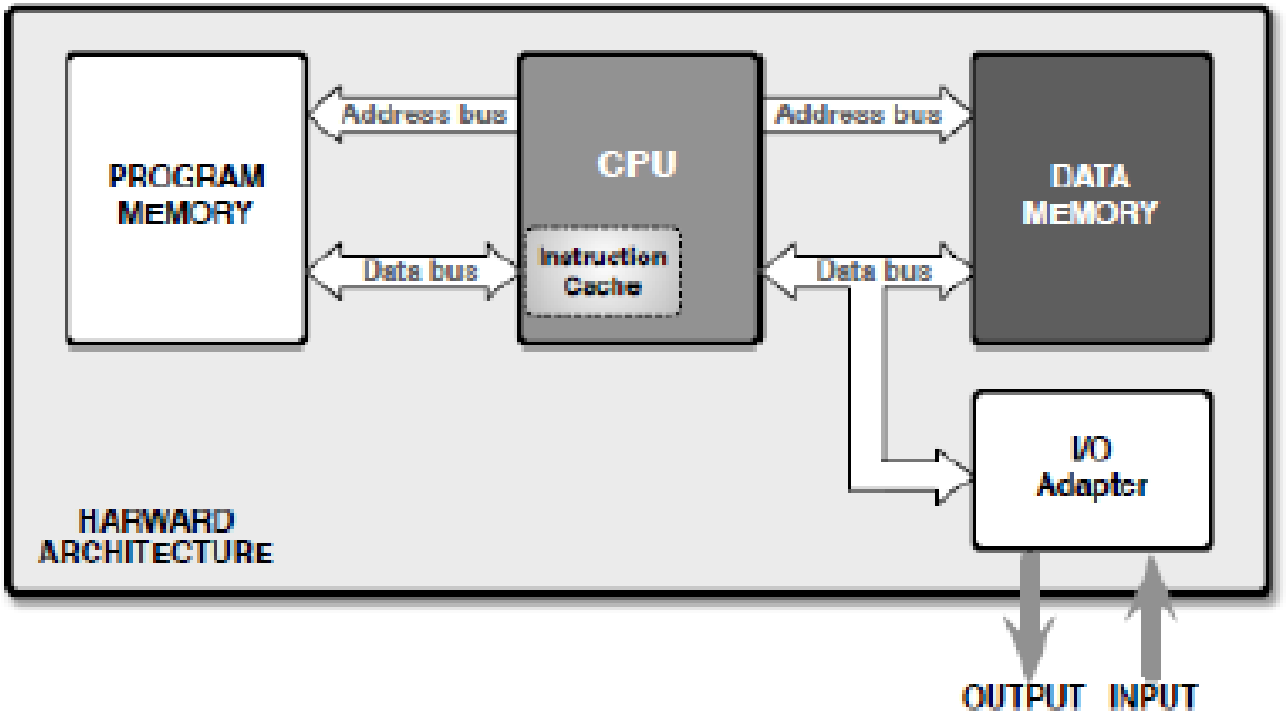
GPU many cores
- SIMD (Single Instruction / Multiple Data)
- all cores executing the same instruction at the same time, but working on different data
- only one instruction de-coder needed to control all cores
- functions like a vector unit





1. Processors
→ Digital architectures with memories

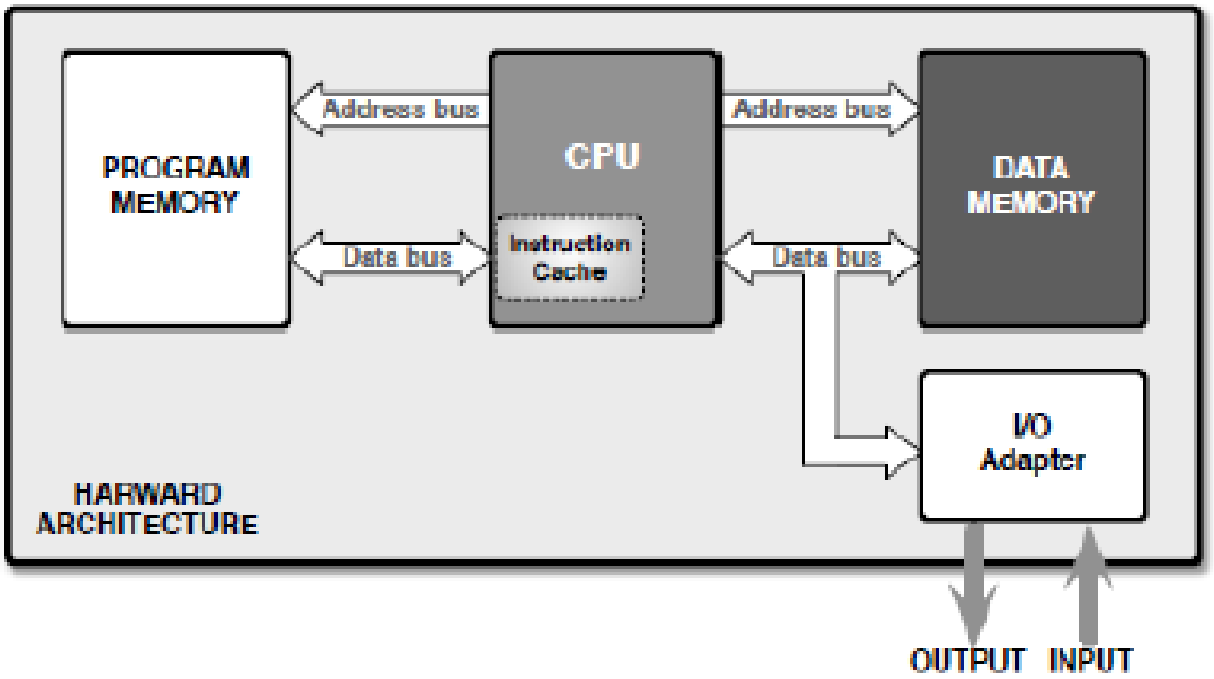
For heavy mathematical loads the Harvard architecture uses separate memories for data and instructions. The use of separate busses for data and instructions provides high speed.



Harvard Architecture

A Harvard architecture computer can thus be faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway
→ modern CPU's adopt hybrid von Neumann/Architecture: fast cache memories separated for data and for instructions…

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 Signal Proc. (DSP)

Digital signal processors (DSPs)
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 processing.

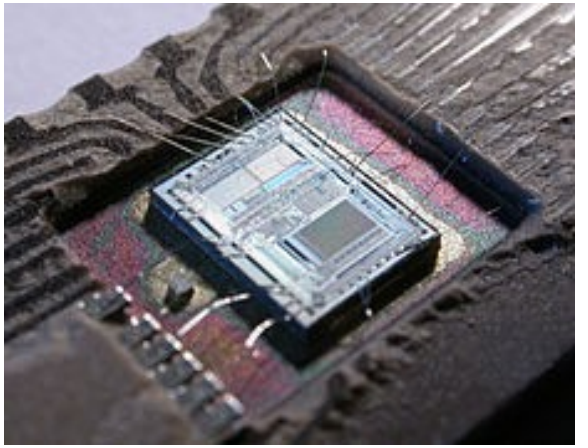
Digital signal processing algorithms typically require a large number of mathematical operations to be performed quickly and repeatedly on a series of data samples.
Signals are constantly converted from analog to digital, manipulated digitally, and then converted back to analog form. Many DSP applications 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



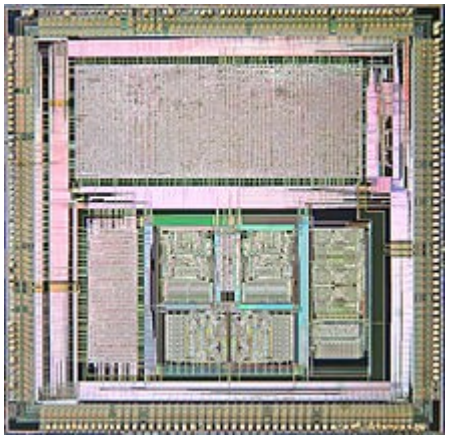
While some embedded systems are very sophisticated,
many have minimal requirements for memory and program length,
with no operating system, low software complexity and
can feature low power consumption

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.

1. Processors: ASICs vs FPGA

CPUs, GPUs, DSPs and Micor-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) is an 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.

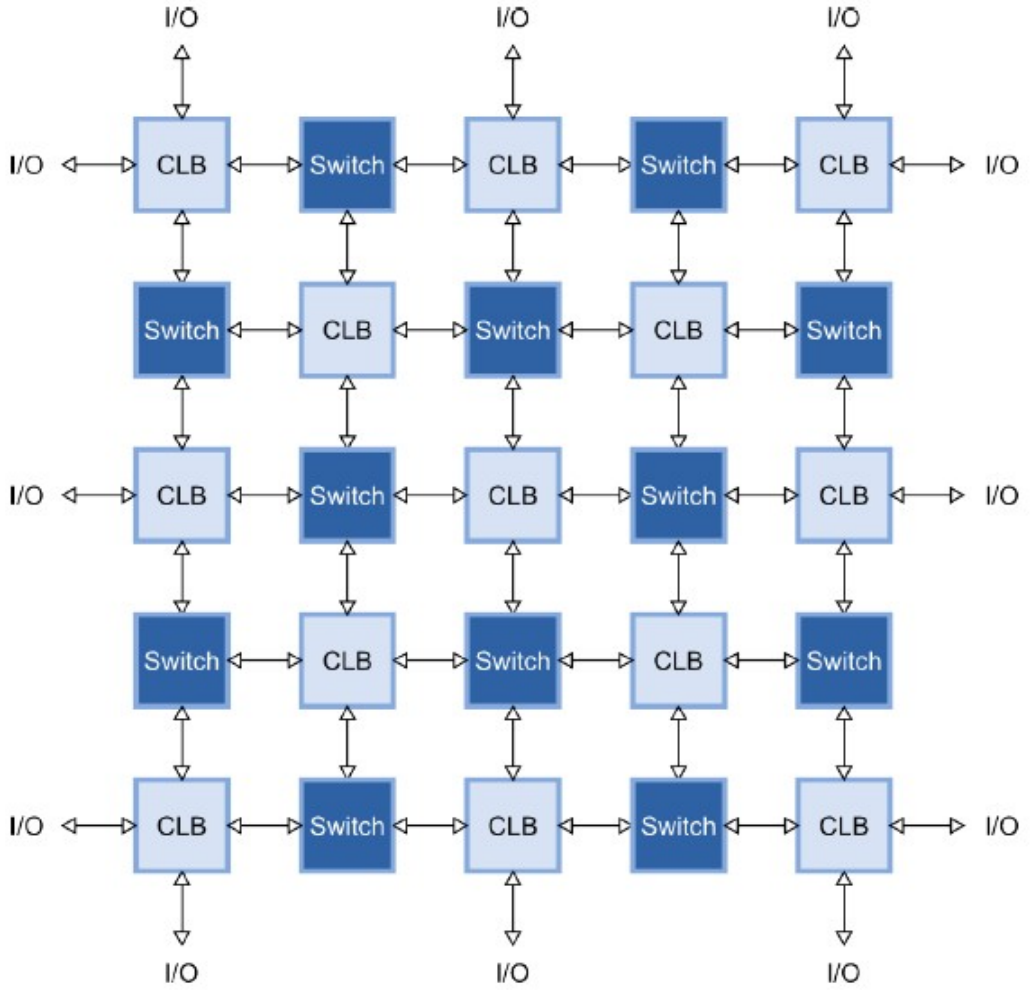
1. Processors: ASICs vs FPGA

Logic blocks can be configured
- to perform complex combinational functions, or
- merely simple logic gates like AND and XOR
- logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory

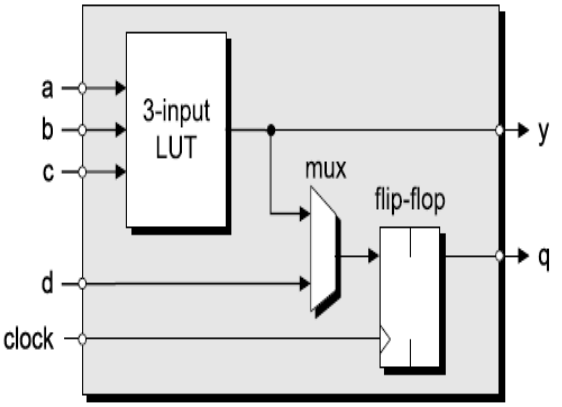
FPGAs can be reprogrammed to implement different logic functions
→ allowing flexible reconfigurable computing
(as performed in computer software)

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.

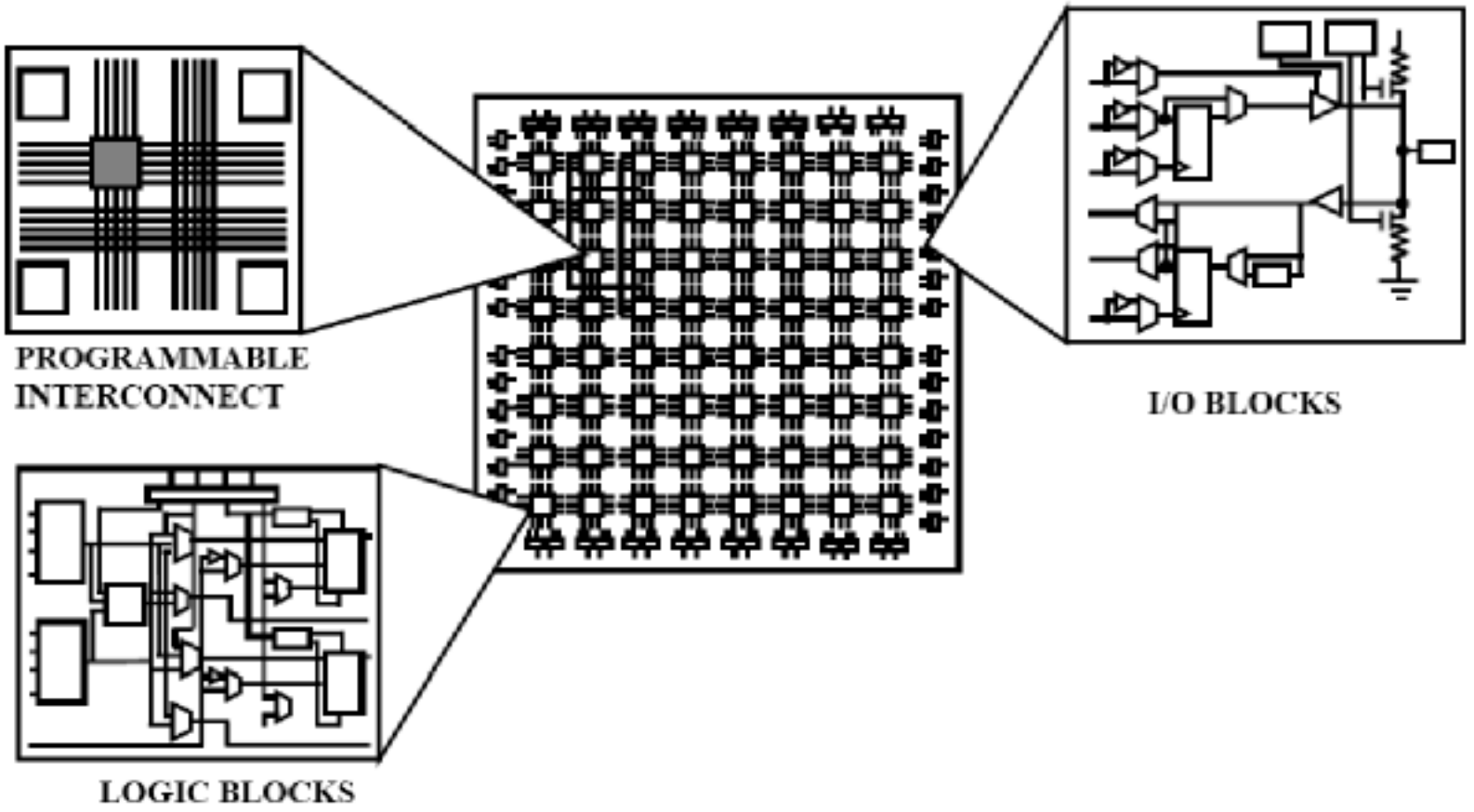


Example of CLB



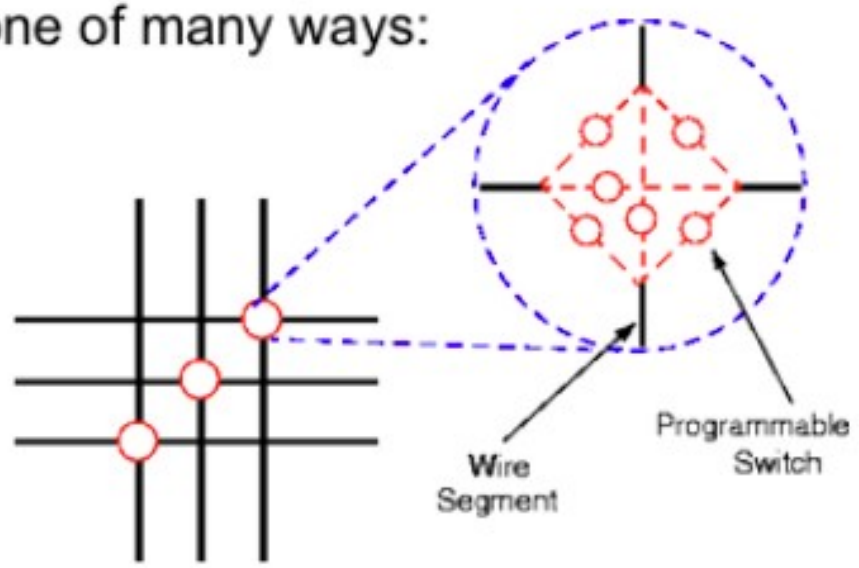
1) Look Up Table (LUT)
→ any combinatory function (3 input)
2) flip-flop (FF)
→ 1-bit memory
3) additional input
→ to combine large combinatory functions through multiplexer (MUX)

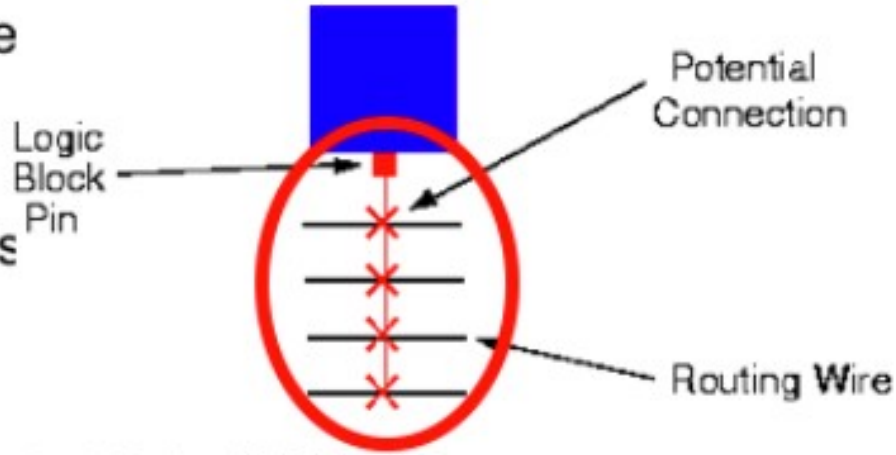
Field Programmable Gate Arrays - FPGA

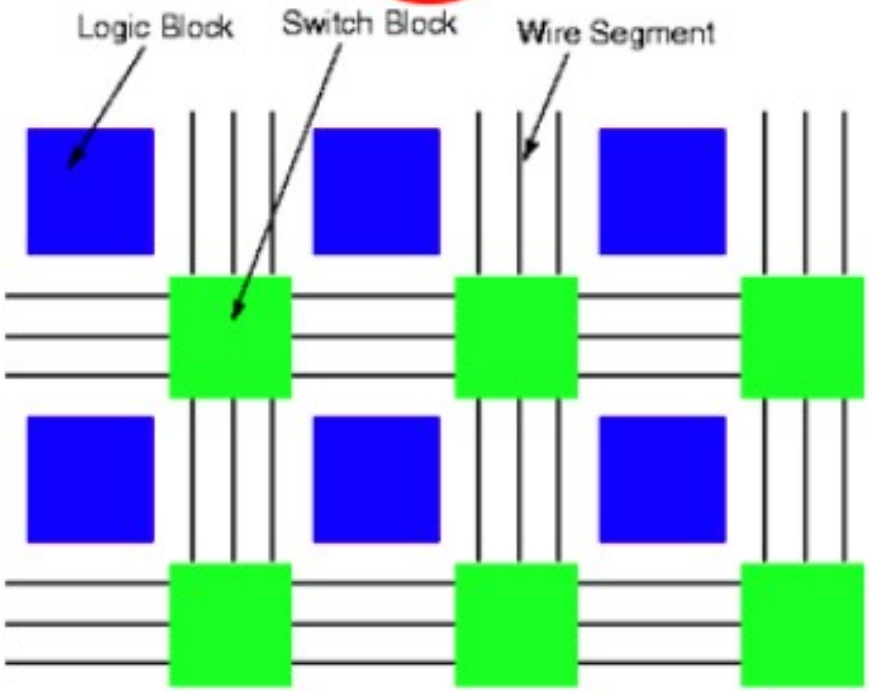


FPGA – Programmable Routing

- 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:



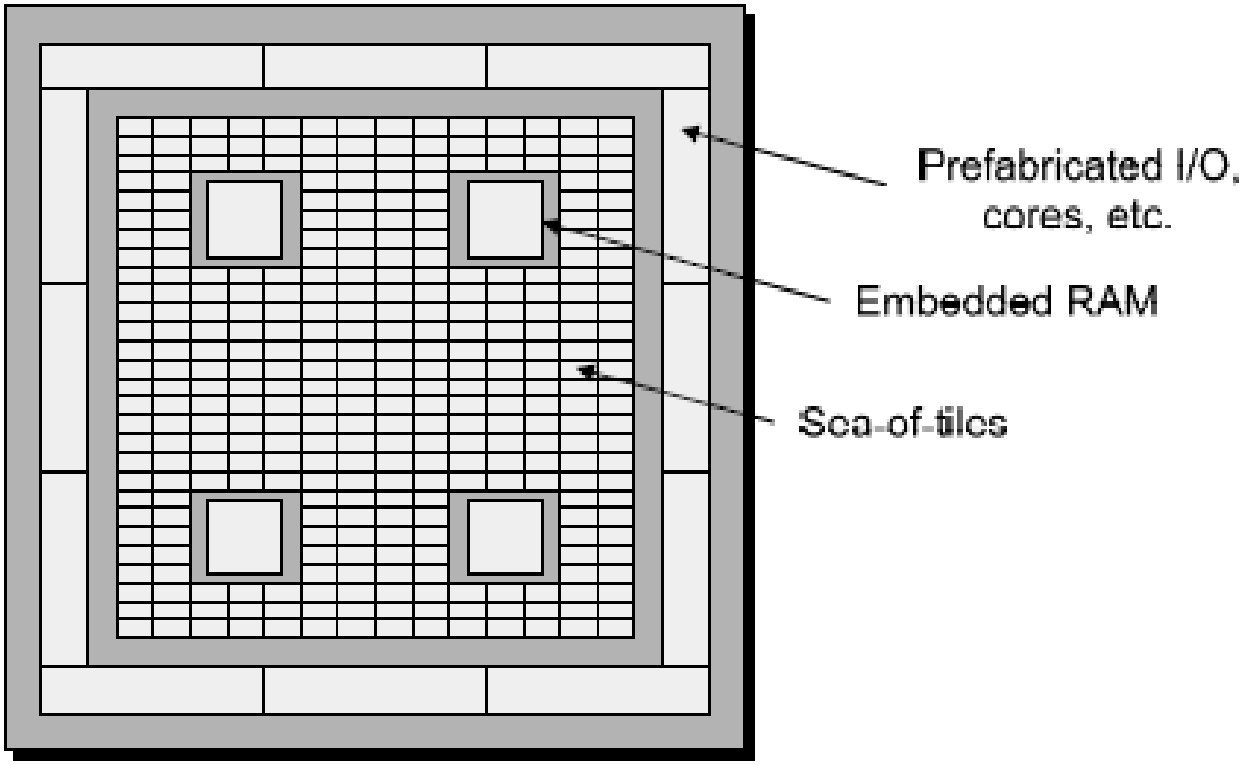




FPGA – Additional Blocks

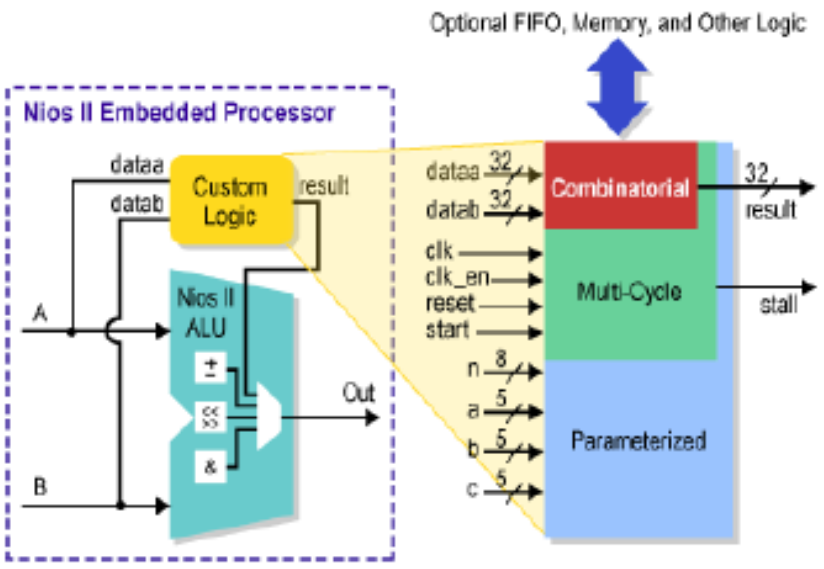
Le FPGA spesso includono moduli funzionali specifici come:

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

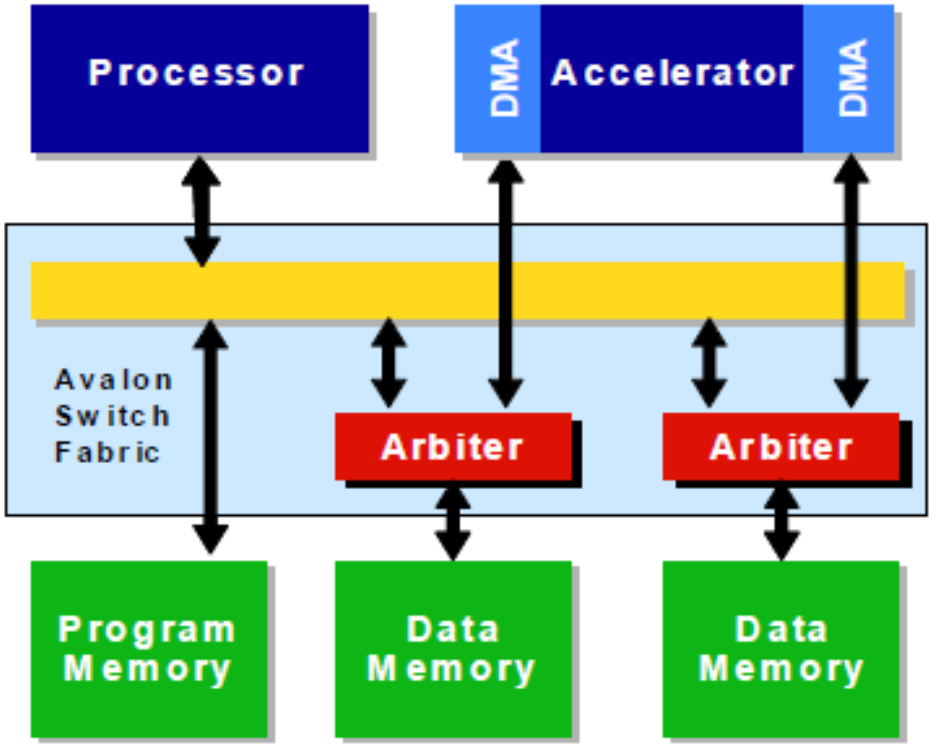


1. Processors: computing with FPGAs

Soft processor IP cores can be implemented within the FPGA logic. Nios II, MicroBlaze and Mico32 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.



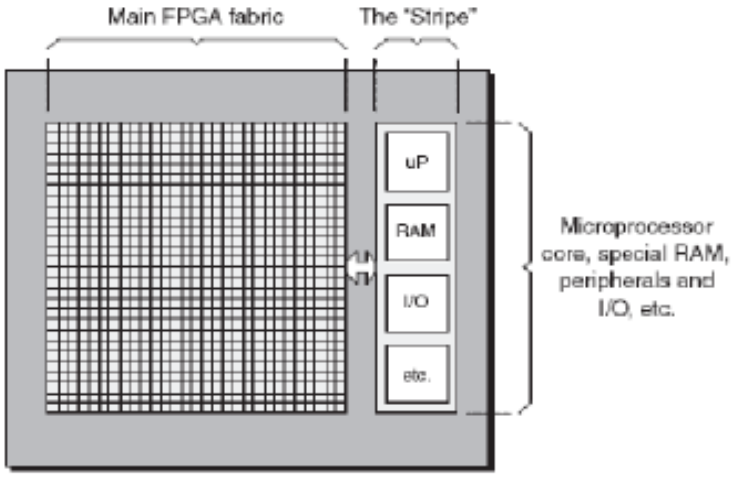
custom instruction implementation in the Nios II CPU (INTEL-ALTERA)

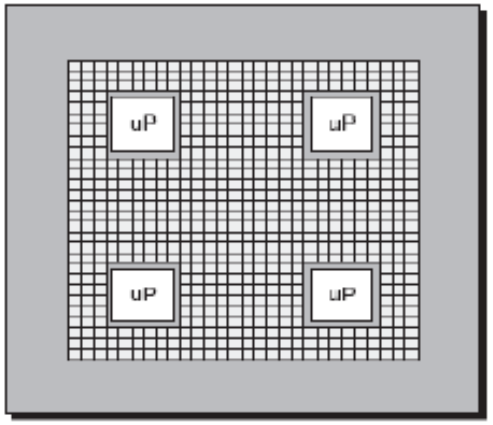


Hardware accelerator implementation with the Avalon switch fabric

1. Processors: computing with FPGAs

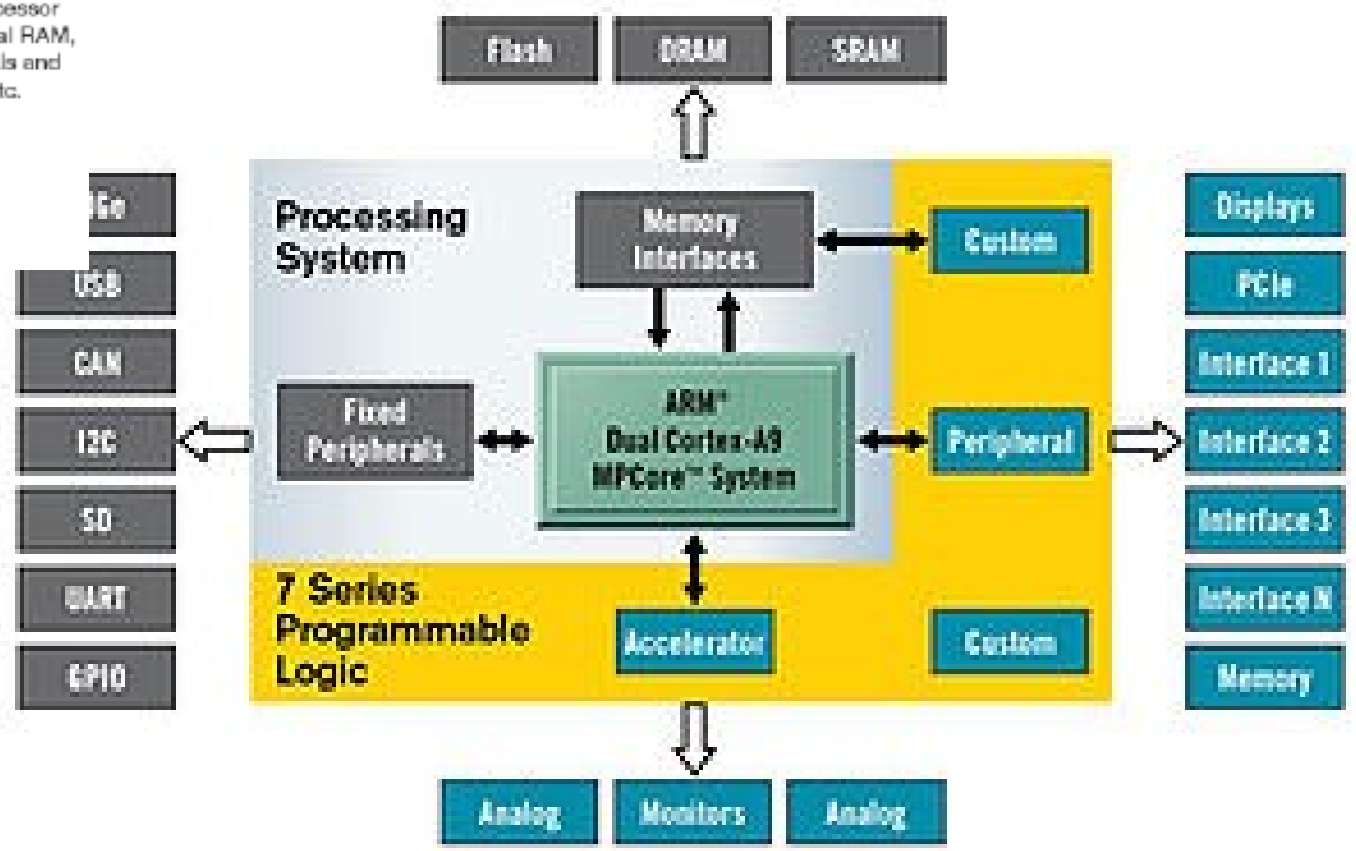
Hard processor IP cores can be implemented on the same FPGA chip





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

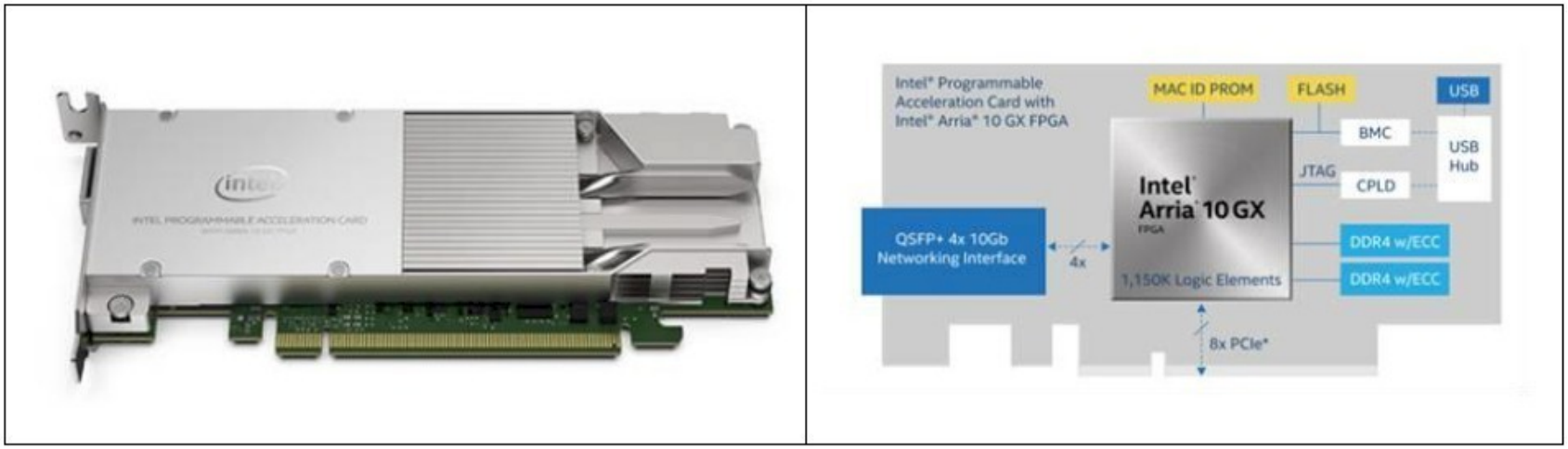
GPU and FPGA are deployed alonside CPUs as HW accelerators

CPU
- Fast caches
- Branching adaptability
- High performance

→ CPUs are great for task parallelism

NVIDIA “TITAN RTX“ GPU

Intel Programmable Acceleration Card (PAC) – FPGA

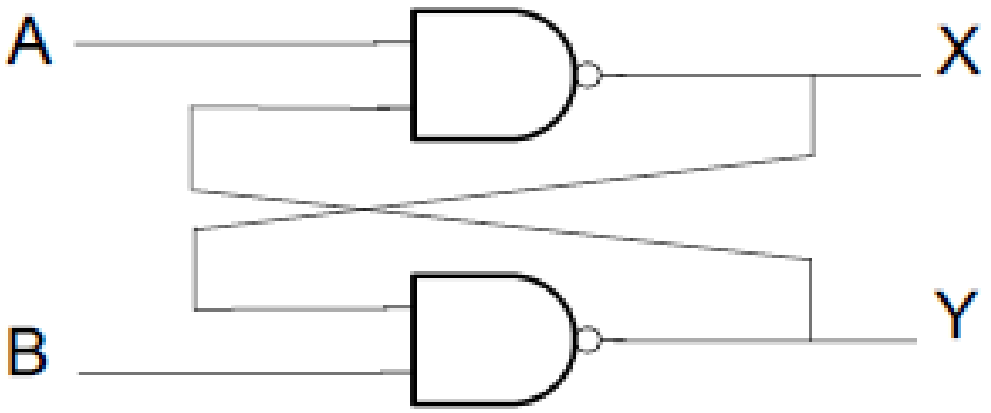


Hardware & Machine Learning

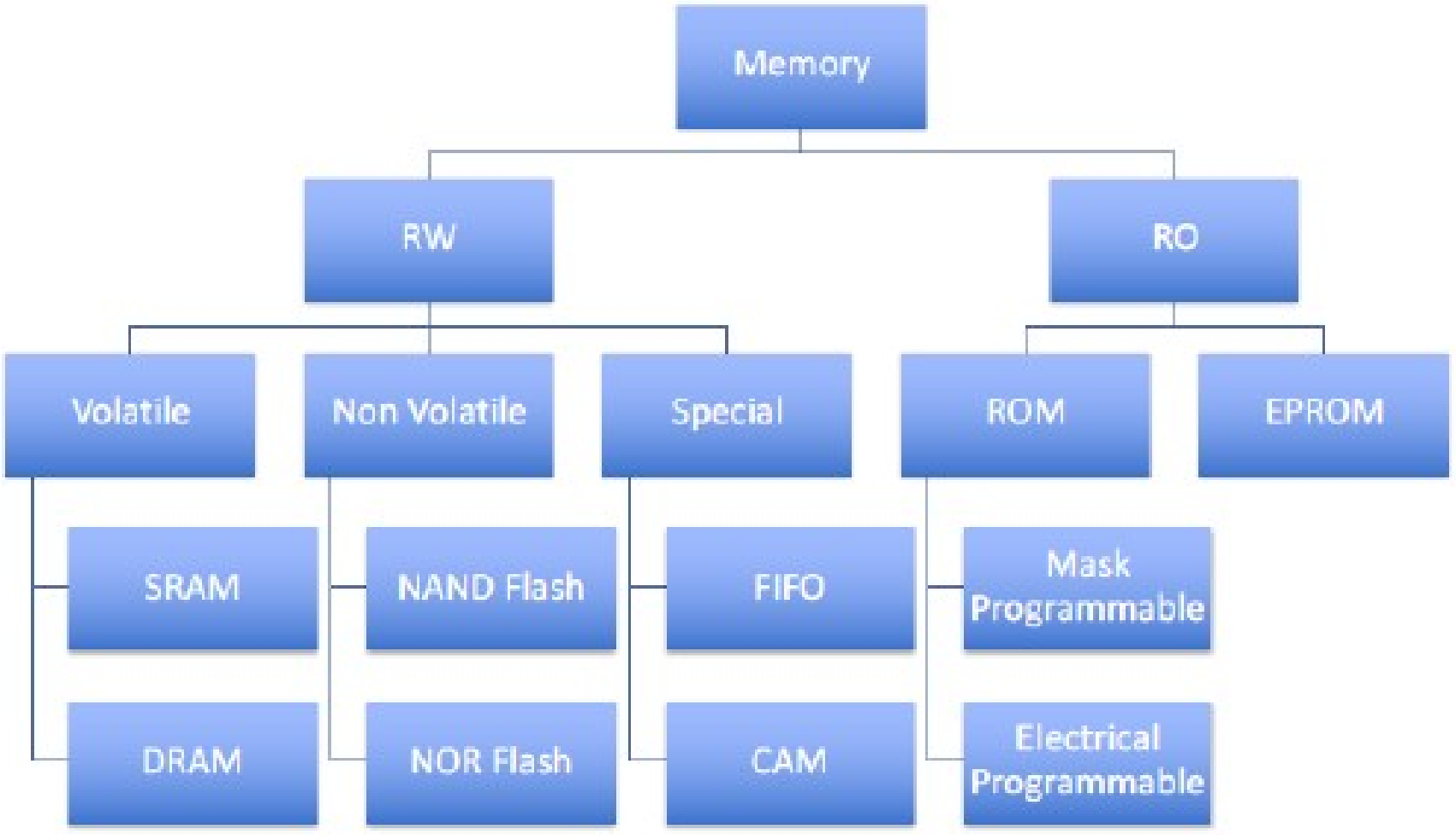
Three latest development boosted "Machine Learning"

- 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



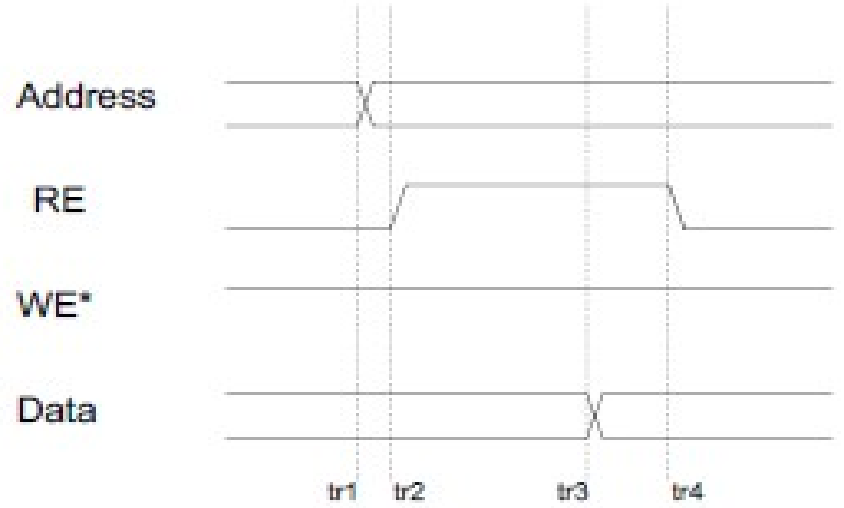
Different storage mechanisms, max capacities and speed...

2. Memory hierarchies in computers

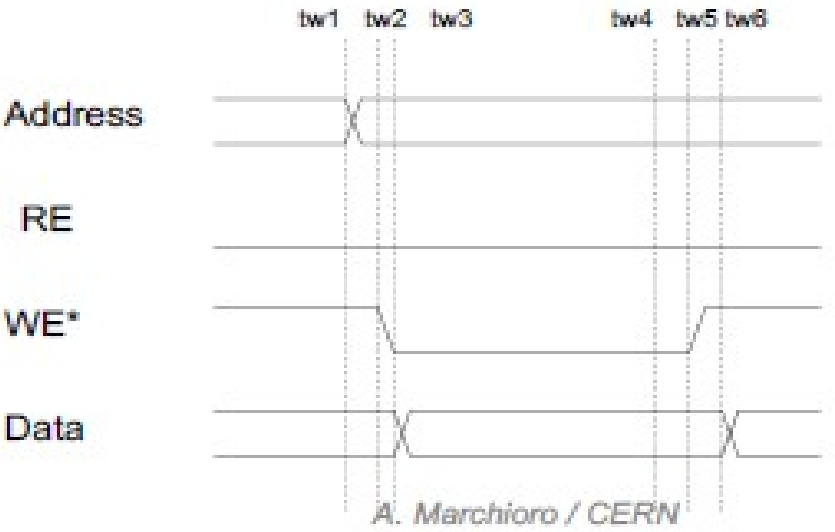
Level	Name	Speed	Size	Type
Level 1	Cache	1-2 Clock cycles (1-5 ns)	10KB-1MB	SRAM
Level 2	Main	10-50 Cycles (20-100 ns)	128 MB-100 GB	DRAM
Level 3	Disk	< 1ms	100GB-10TB	Magnetic- SSD
Level 4	Archive	< 1s	PB	Magnetic

2. Memory timing

Read Cycle

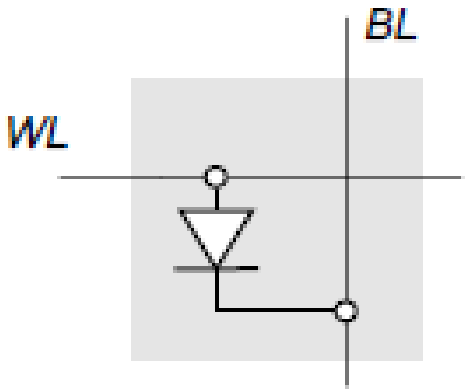


Write Cycle



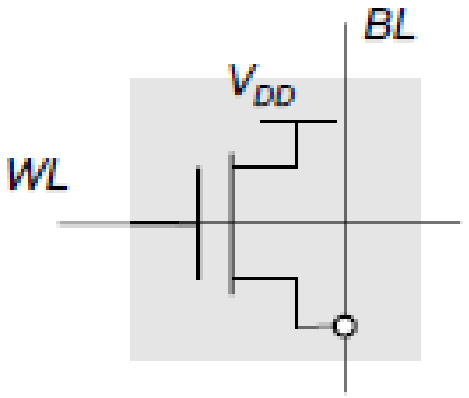
2. Memory – Read Only

1



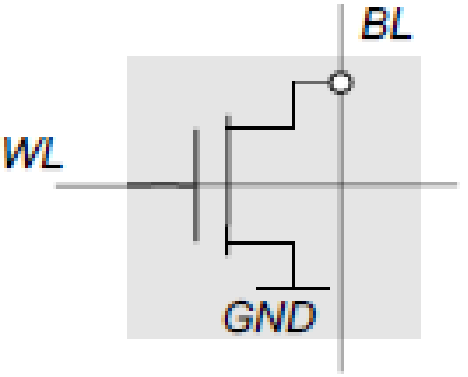
Diode ROM

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



PMOS ROM

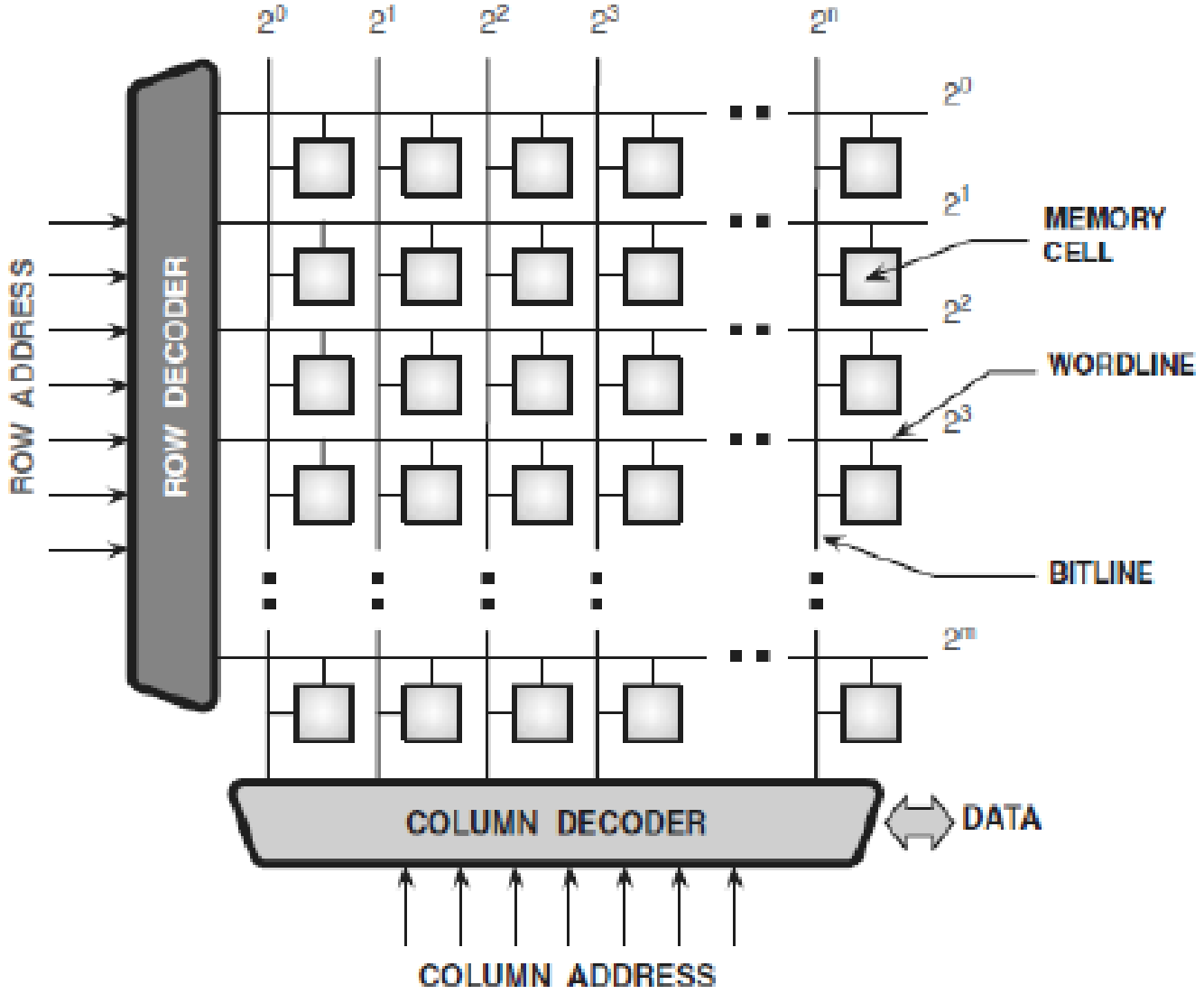
```
if (WL == 0) BL = 1
```



NMOS ROM

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

2. Memory arrays

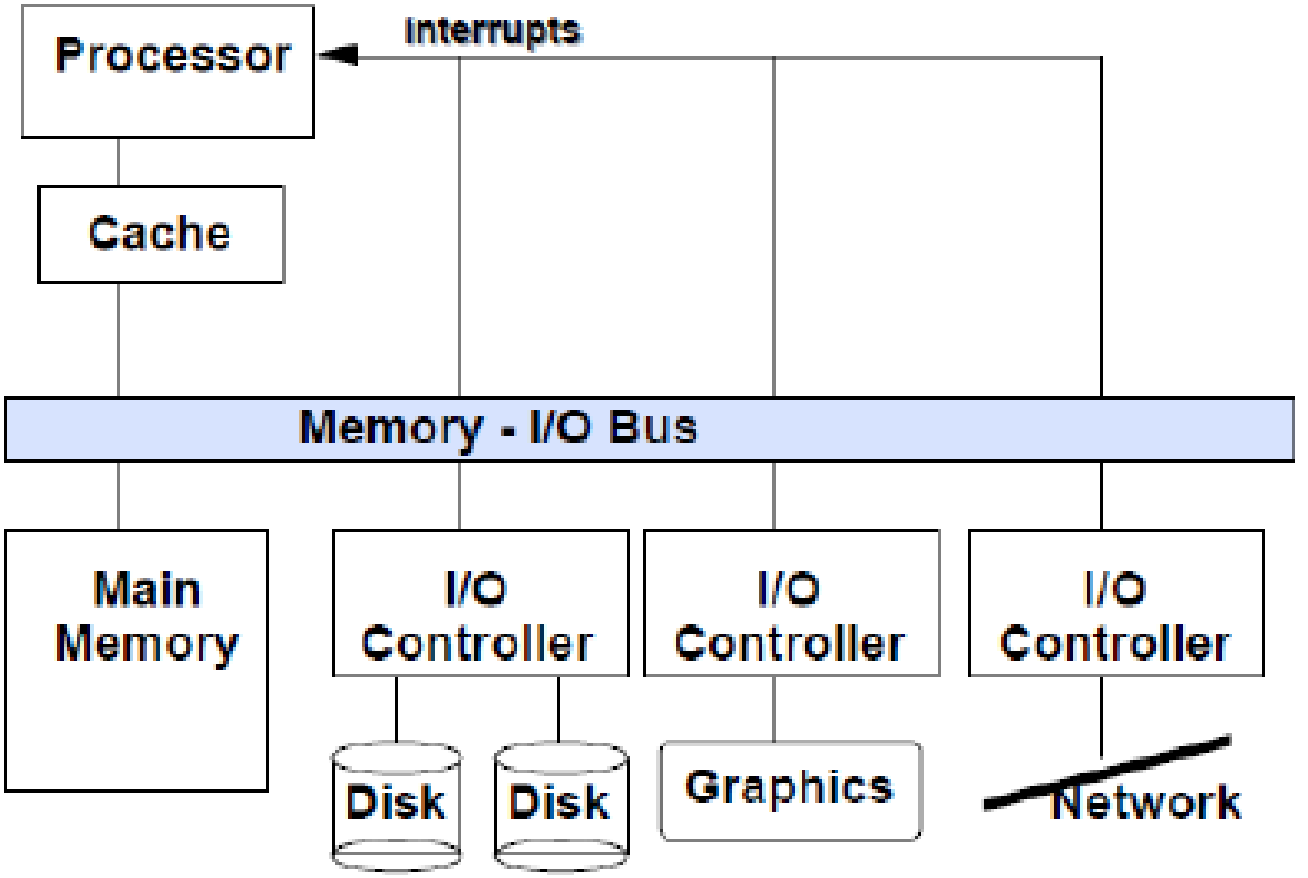


Hardware & Machine Learning

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

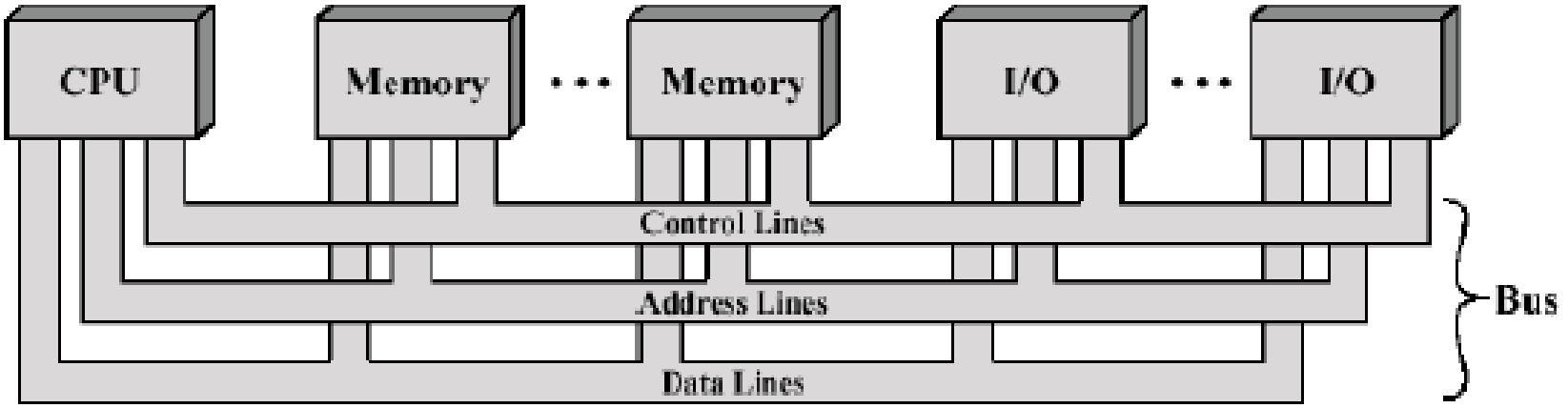


Time(workload) = Time(CPU) + Time(I/O) - Time(Overlap)

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

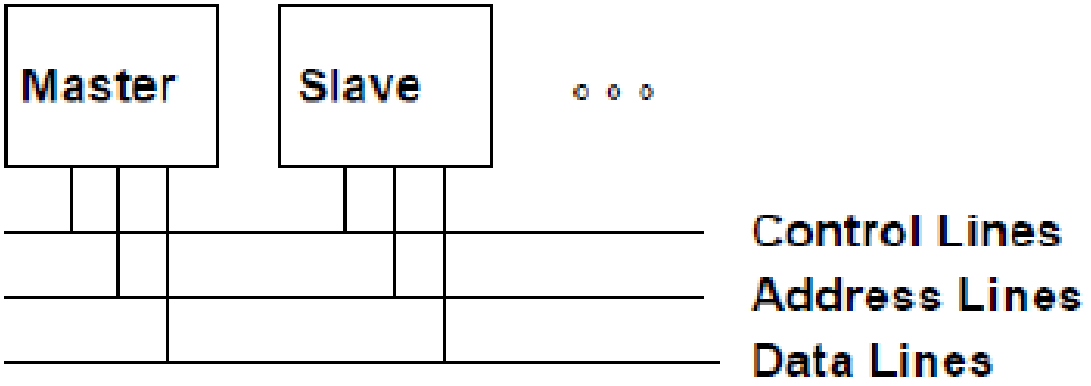
- Interconnect = glue that interfaces computer system components
- High speed hardware interfaces + logical protocols
- Networks, channels, backplanes

	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.

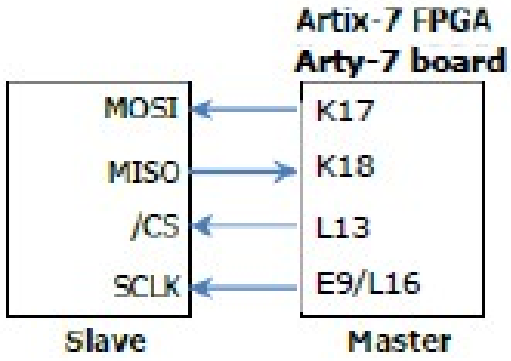
Asynchronous Bus Transfers: control lines (req., ack.) serve to orchestrate sequencing

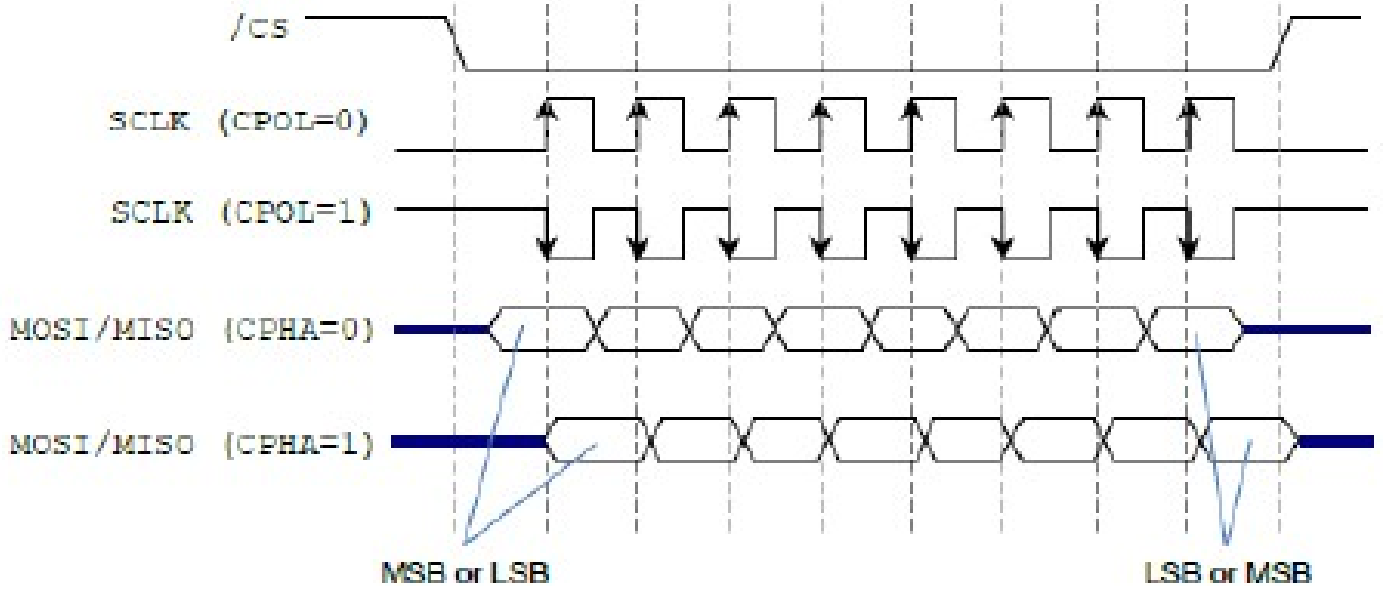
Synchronous Bus Transfers: sequence relative to common clock

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: example PCI-express

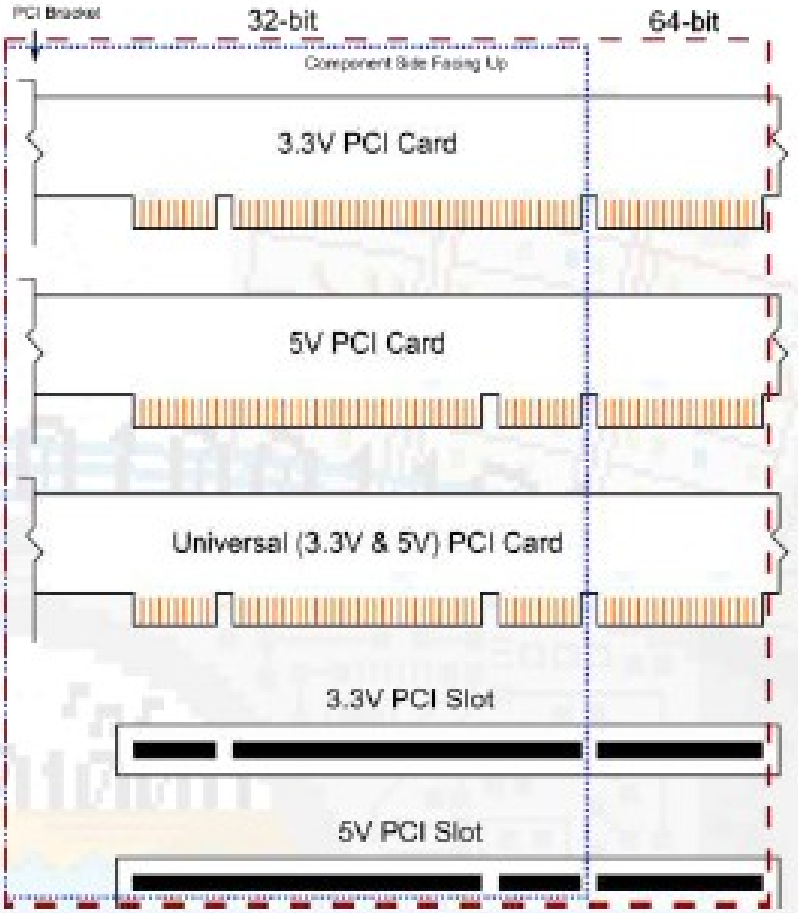
PCI (Peripheral Component Interconnect) Express is a popular standard for high-speed computer expansion overseen by PCI-SIG

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)

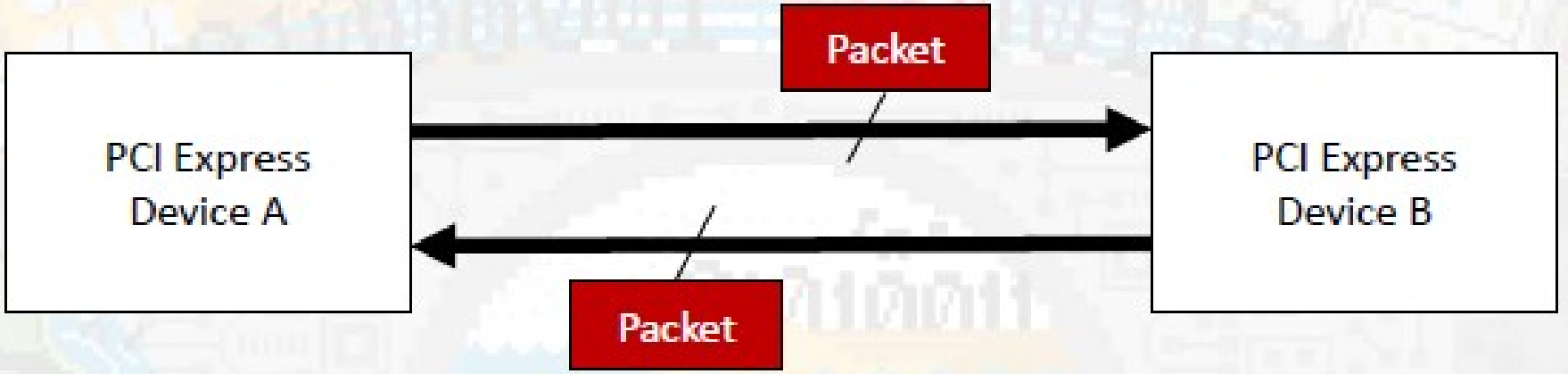
3. Interconnections: PCI-express

- 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)



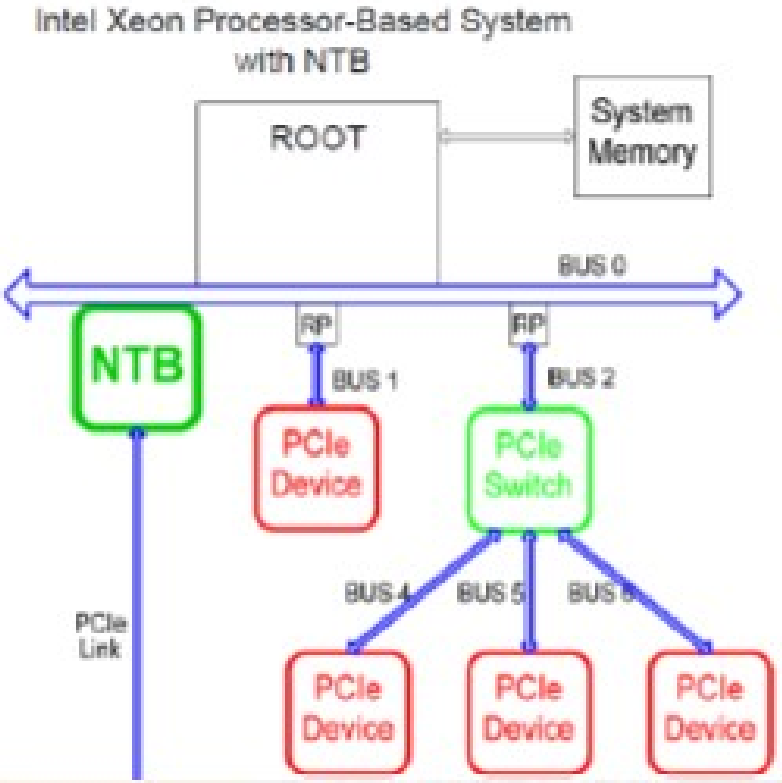
3. Interconnections: PCI-express

- 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



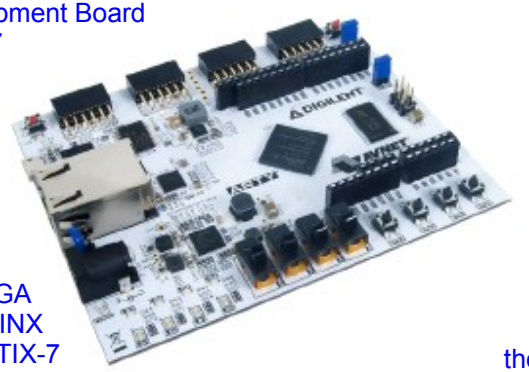
Lectures Plan

1. Introduction to Digital electronics and Digital systems
3 Lectures (Oct)
Lecturer: G.Collazuol

2. TDAQ systems in High Energy Physics
ie examples of Real-Time Data Management and Analysis
5 lectures (Oct/Nov)
Lecturer: E.Meschi (CERN)

3. FPGA laboratory
16 Lectures / Hands-on Laboratory sections (Nov/Dic/Jan)
Lecturers: A.Bergnoli (INFN), R.Isocrate (INFN), G.Collazuol

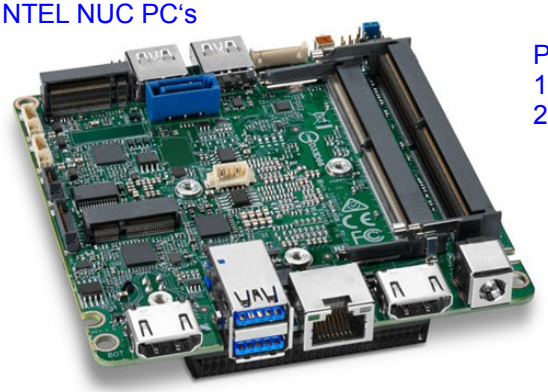
Development Board
ARTY-7



FPGA
XILINX
ARTIX-7

15 ARTY boards
and 15 NUC PCs
will be available for
the hands-on lab. classes

INTEL NUC PC's



PCs for
1) programming the FPGA
2) communication w/ FPGA v
ia fast Ethernet

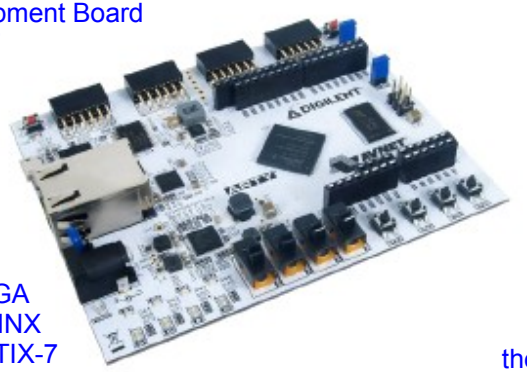
Lectures Plan

3. FPGA laboratory

16 Lectures / Hands-on Laboratory sections (Nov/Dic/Jan)
Lecturers: A.Bergnoli (INFN), R.Isocrate (INFN), G.Collazuol

Laboratory overview
A) Intoduction to FPGA, VHDL and firmware development framework
B) Simulations and test-benches
C) Combinatoria and Sequential Logic circuits
D) Flash Memory
E) Virtual I/O and Integrated Logic Analyzer
F) Arithmetic Operations
→ example of Digital Filter (FIR)
→ example of Neutral Network implementation (character recognition)
E) Finite State Machines → Interconnect protocol example (SPI) → Access to Flash Memory
G) IPBus example → access to PC via ethernet
H) Leros a tiny micro-controller

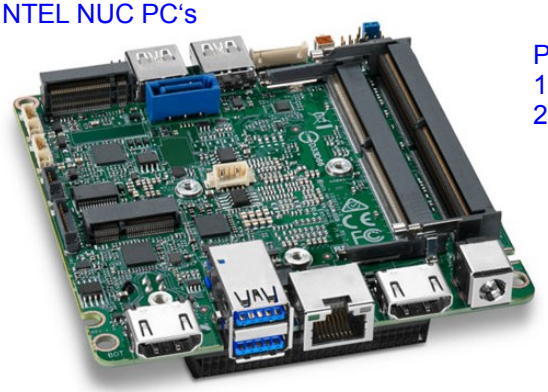
Development Board
ARTY-7



FPGA
XILINX
ARTIX-7

15 ARTY boards
and 15 NUC PCs
will be available for
the hands-on lab. classes

INTEL NUC PC's



PCs for
1) programming the FPGA
2) communication w/ FPGA v
ia fast Ethernet

Lectures Plan

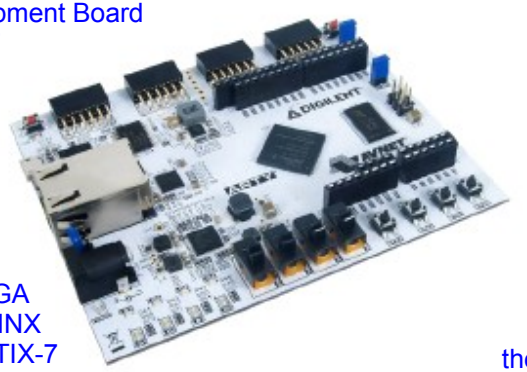
3. FPGA laboratory

16 Lectures / Hands-on Laboratory sections (Nov/Dic/Jan)

Lecturers: A.Bergnoli (INFN), R.Isocrate (INFN), G.Collazuol

- up to 15 groups (ideally 2 per group)
- off-line practice will be most important
- home-work will be evaluated

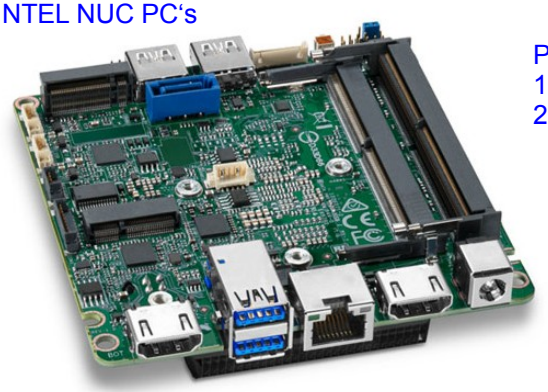
Development Board
ARTY-7



FPGA
XILINX
ARTIX-7

15 ARTY boards
and 15 NUC PCs
will be available for
the hands-on lab. classes

INTEL NUC PC's



PCs for
1) programming the FPGA
2) communication w/ FPGA v
ia fast Ethernet

Whish list - objectives

1. understading basic Digital blocks and Digital systems
2. Real-Time systems by example
3. learning FPGA programming (VHDL) hands-on

Final Examination

Proper balance between:

1. Oral exam
2. Homework exercises

Bibliography

Part 1

1. ISOTDAQ school Lectures
2. Maloberti - “Understanding Microelectronics – A Top-Bottom Approach“

Part 2 and **Laboratory**

3. Zwolinski - Digital System Design with VHDL

Material and Contacts

Contacts → Whatsapp group

Material exchange → git repository

Additiona Material