Introduction to Distributed Systems

C.Vidrașcu
https://profs.info.uaic.ro/~vidrascu

Agenda

- Flynn's taxonomy
- Distributed systems types
- Design issues for DS
- History of DS

Flynn's taxonomy (1966) (1)

- The processor (CPU) has 2 functional parts:
  - the control unit : decides what operations need to be done
  - the processing unit : executes those operations
- A control unit may command one or several processing units

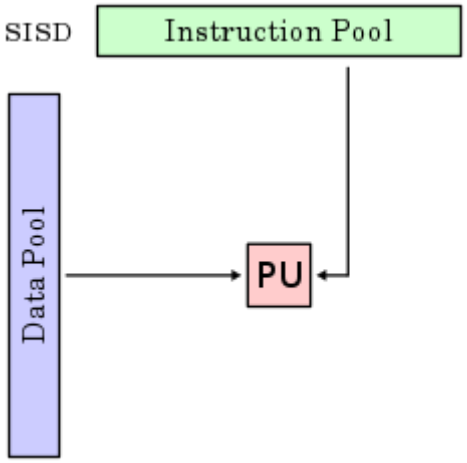
Flynn's taxonomy (2)

- SISD
  - Single Instruction, Single Data
- SIMD
  - Single Instruction, Multiple Data
- MISD
  - Multiple Instruction, Single Data
- MIMD
  - Multiple Instruction, Multiple Data

SISD (1)

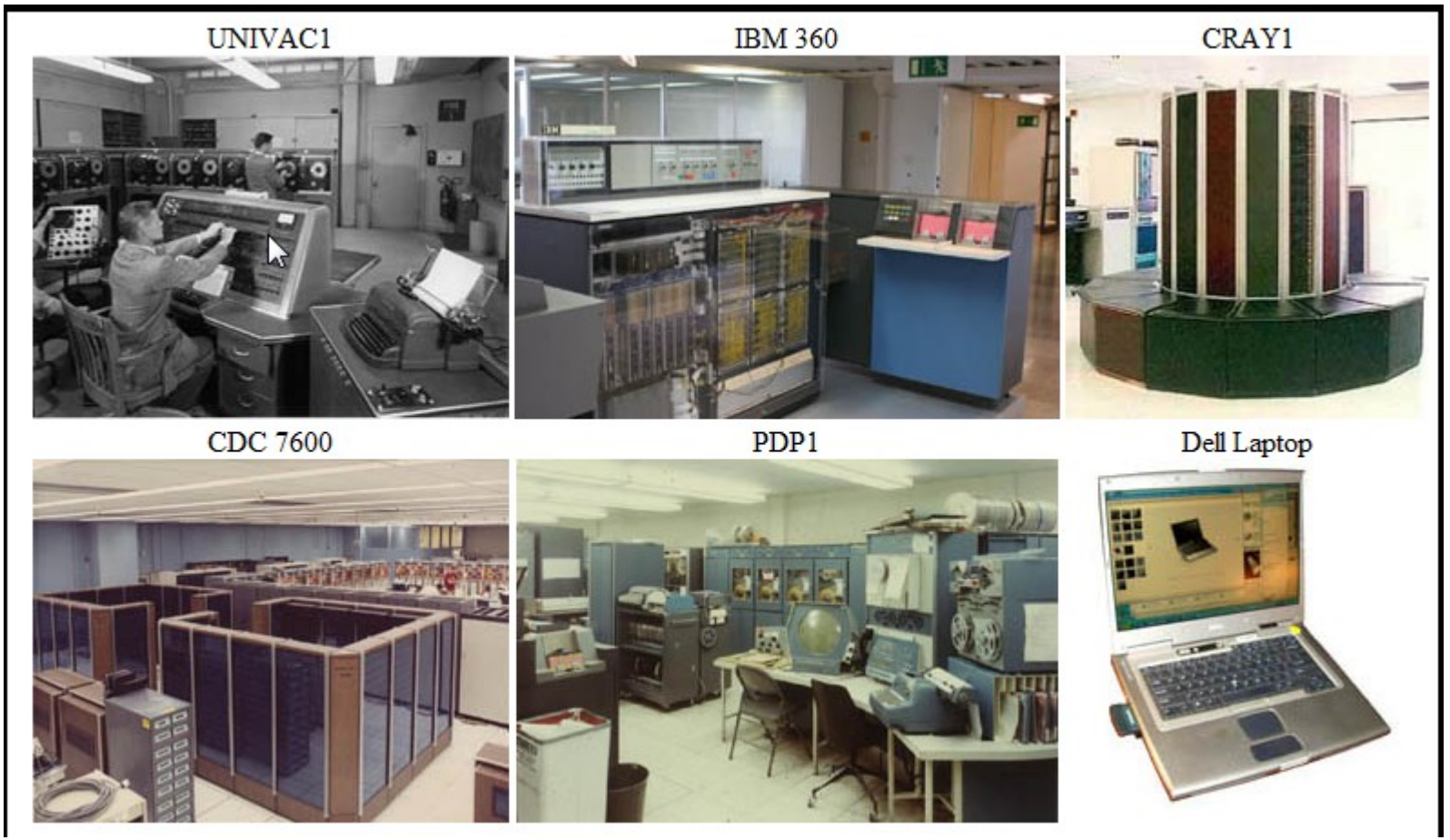
- only one control unit
- only one processing unit
- von Neumann architecture
- it works completely sequencially:

  the execution of the next instruction only starts after finishing the execution of the previous one



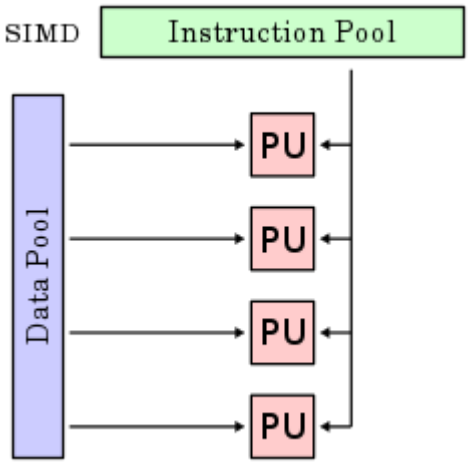
SISD (2)

- Examples:

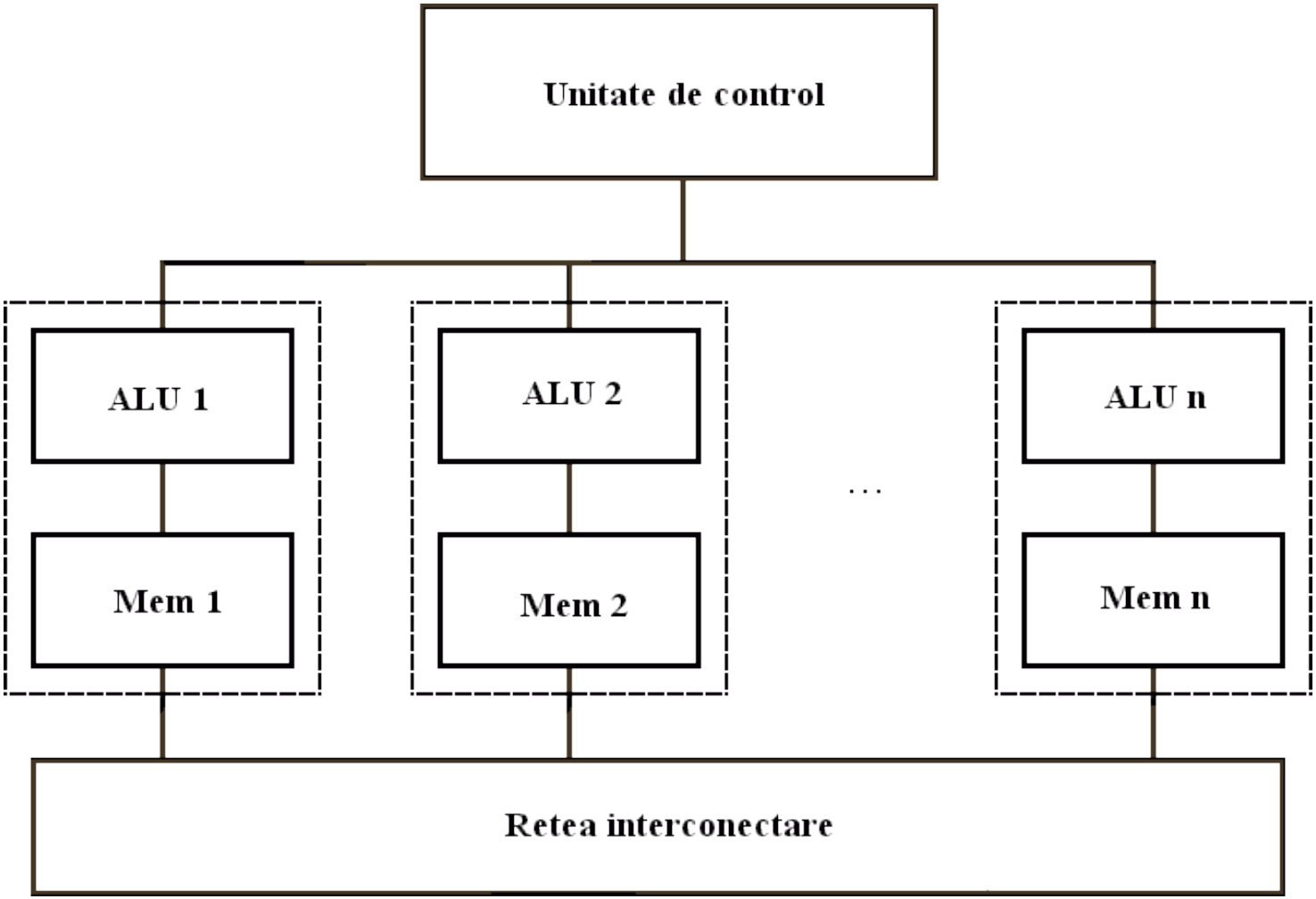


SIMD (1)

- one control unit and several processing units
- the same instruction is executed in parallel on different data
- implementations:
  - vector pipelines (e.g. IBM 9000)
  - processor arrays (e.g. ILLIAC IV)

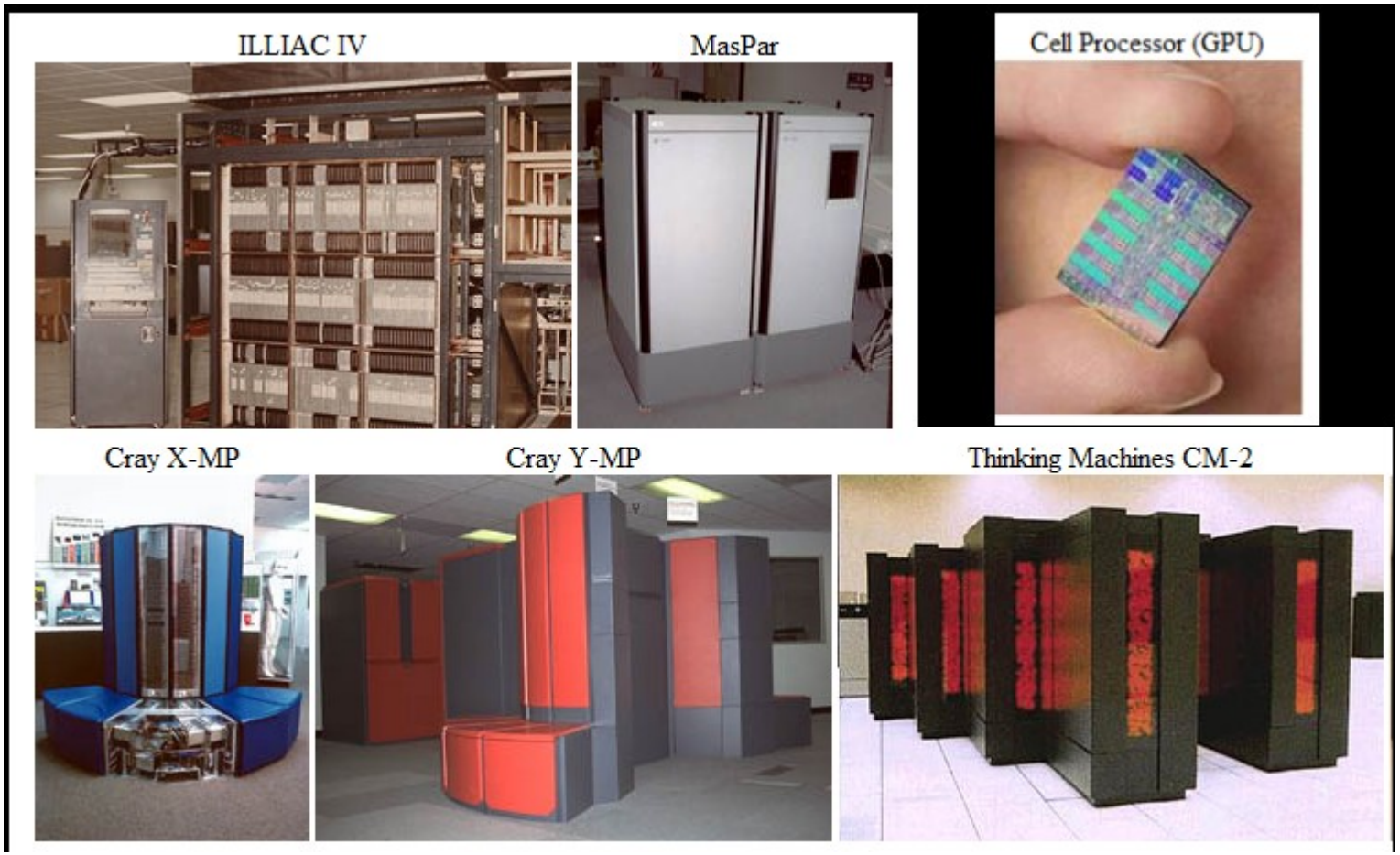


SIMD (2)


Unitate de control
ALU 1
ALU 2
...
ALU n
Mem 1
Mem 2
Mem n
Retea interconectare

SIMD (3)

- Examples:

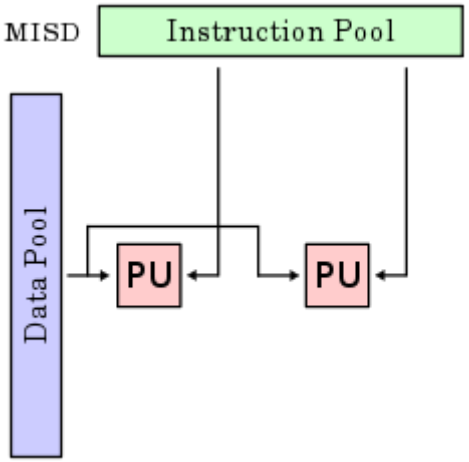


SIMD (4)

- not all operations can be done in parallel
- in practice, combinations SISD-SIMD
  - Intel: MMX, SSE
- the problem is caused by the conditional instructions
  - instruction streams diverge because of them
  - solution: disabling of processing units with incorrect data
  - drawback: loss of performance

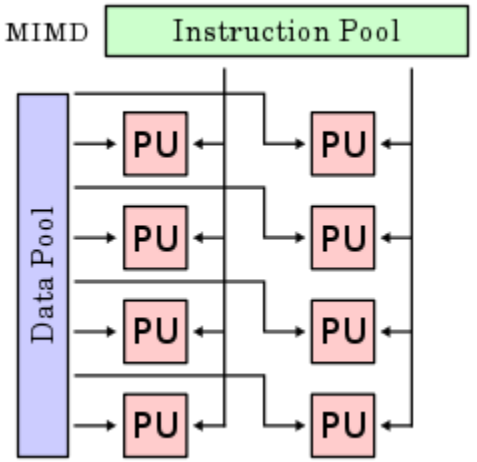
MISD

- several control units and only one processing unit
- different operations executed on the same data, in parallel
- implementation - ???
- pipeline systems
  - superscalar processor computers (e.g. CDC 6600)
- redundant systems

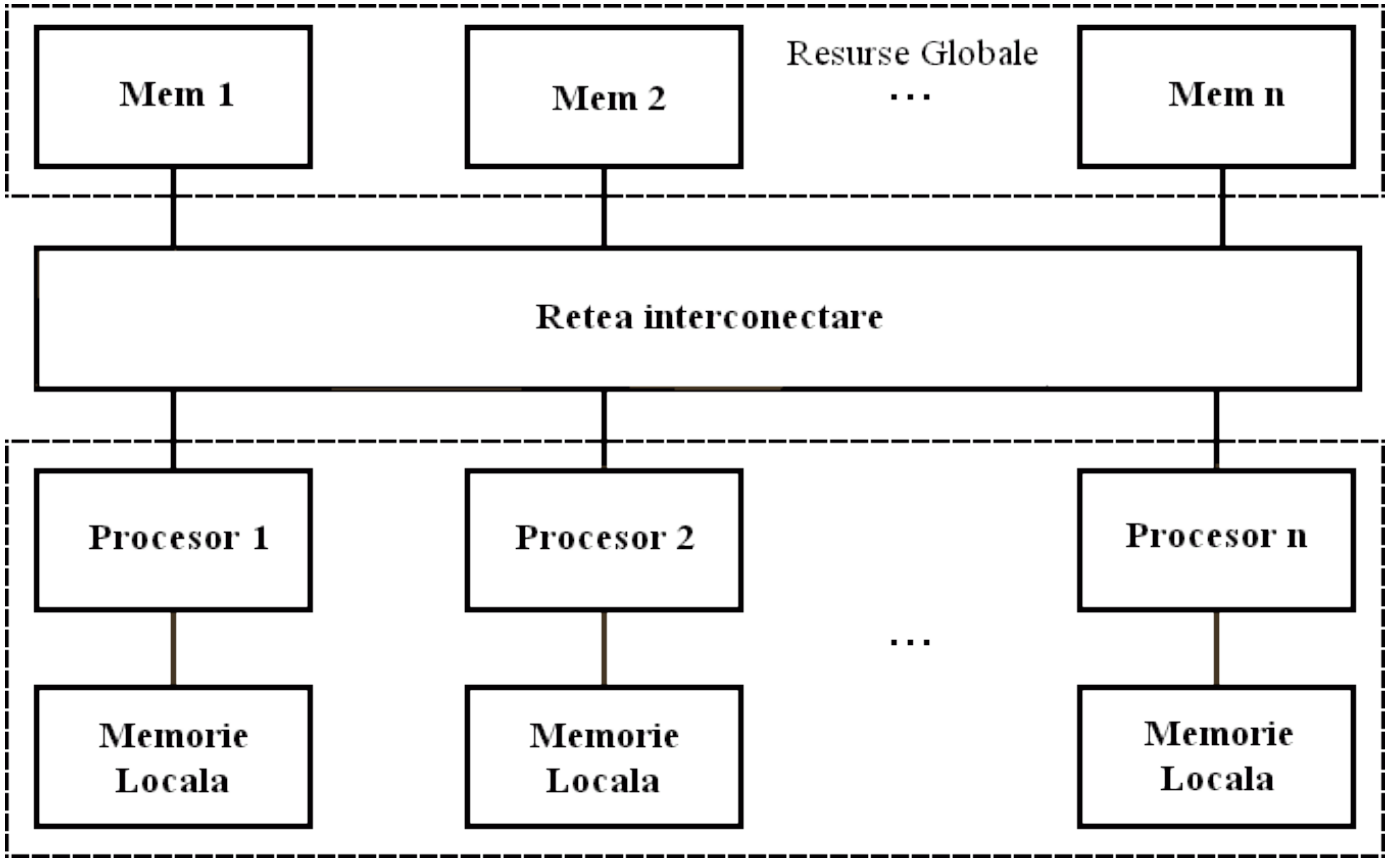


MIMD (1)

- several control units and several processing units
- more than one instruction streams, executed:
  - independently
  - cooperatively
- distributed systems are MIMD



MIMD (2)


Mem 1
Mem 2
Resurse Globale
...
Mem n
Retea interconectare
Procesor 1
Procesor 2
...
Procesor n
Memorie Locala
Memorie Locala
Memorie Locala

MIMD (3)

- Examples:

IBM POWER5

HP/Compaq Alphaserver

Intel IA32

AMD Opteron

Cray XT3

IBM BG/L

MIMD - classification

- computers can have shared memory or not
  - computers with shared memory - multiprocessors
    - all processors have the same address space for accessing the memory
    - generally speaking, are called parallel systems
  - computers without shared memory - multicomputers
    - each processor has its own address space
    - generally speaking, are called distributed systems
- based on the interconnection topology:
  - bus vs. switched

Interconnection

- *tightly coupled* systems
  - fast transfer of messages between the processing units
  - transfer data rate - high
- *loosely coupled* systems
  - slow transfer, low data rate
- this classification can be done both for software, as well as for hardware

Distributed systems types

- diferrent combinations are posible based on the hardware/software interconnection types
- *parallel systems*: systems which are tightly coupled în hardware
- *distributed systems* are usually loosely coupled in hardware
- variants of *distributed systems*:
  - network operating systems
  - "true" distributed systems

Parallel systems

- tightly coupled software, running on tightly coupled hardware
- hardware: a single multiprocessor computer (i.e. a computer with several physical CPUs or, nowadays, with several cores in CPU)
- software: e.g. any time-sharing operating system for SMPs (UNIX, Windows, etc.)

Network operating systems (1)

- loosely coupled software, running on loosely coupled hardware
- independent computers (workstations), connected in a LAN network
- each workstation has its own operating system
  - the same operating system on all workstations

  or
  - not, i.e. different OSes, or different versions of it

Network operating systems (2)

- from one workstation tasks can be executed on other workstations:
  - running some commands/programs remotely
  - file transfer
- more complex communications mechanisms
  - ex.: file server
- there has to be an agreement between all workstations about the syntax and the meaning of the messages transferred between them

  i.e. protocols for communication

"True" distributed systems (1)

- tightly coupled software, running on loosely coupled hardware
- the users see the distributed system as a single classical computer (*single-system image*)
  - what kind of user? end-users, but also programmers
  - end-users point of view: the running applications see the system's resources in a unified manner, no matter which machines they run on
  - programmers p.o.v.: system calls hide the distributed details

"True" distributed systems (2)

Characteristics of a distributed OS:

- Unified mechanism for interprocess communication
- Uniform management of processes
  - not acceptable to have different scheduling policies on different individual machines
- Unified file system
- Conclusion: all machines are running the same OS kernel
- Plus: a higher-level collection of *system management components* that coordinate the machines' activities

Design issues

- Distributed systems have (also) requirements different from those of classical SISD computers
- When we design a distributed system, what issues we have to deal with?
  - transparency
  - reliability
  - performance
  - scalability

Transparency

- at the end-user level
  - when running a command, the system hides from the user on how many (and which) computers that command will run
- at the programmer level
  - the system calls interfaces hide the distributed structure of the system

Aspects of transparency (1)

- location transparency
  - hardware and software resources can be located anywhere in the sistem
  - they can be accessed in the same way from any machine
  - examples of resources: processors, files, peripheral devices, etc.

Aspects of transparency (2)

- migration transparency
  - extends the location transparency for moving resources
  - if a resource migrates from one machine to another, must be possible to access it using the same name as before the migration
  - this issue is important for migrating resources between servers, without downtime

Aspects of transparency (3)

- replication transparency
  - The OS can make copies (duplicates) of resources without the users knowing about it
  - example: a couple of file servers, which works together to provide a distributed file-system
    - each server stores a subset of the files
    - when a client makes a request to a server, if the request implies a file stored somewhere else, the server forwards the request and sends back to the client the file received, but can keep a local copy of it for future accesses
    - goal: for performance

Aspects of transparency (4)

- concurrency transparency
  - refers to the concurent accesses at some resources made by several users/programs
  - the users will not notice the existence of other users
  - the accesses requested in parallel are solved by the OS sequencially

Aspects of transparency (5)

- parallelism transparency
  - goal: the programmers write applications as for classical uniprocessor computers, but the compiling and/or runtime environment takes advantage of the parallel computing capabilities of the system
  - in parallel computing theory this is called: implicit parallelism
  - It is hard to achieve in the current state of IT

Reliability

- A distributed system can have many computing nodes
  - it is very likely that some components goes down (i.e. malfunction)
  - availability – the periods of time in which the system is non-functional to be as short as possible
  - fault-tolerance – the system must be capable to function even when some components are down

Performance

- the reason for developing parallel and distributed system was to have an increased performance (i.e. computing power)
- these kind of systems are not justified economically for running simple tasks, which can be solved by classical computers
- the limiting factor of performance in distributed systems is communication
  - hardware and software
  - *fine-grained parallelism* vs. *coarse-grained parallelism*

Scalability

- describes how the performance of a parallel or distributed system is increased when the number of CPUs/machines is increased
- preferable – liniar increase
- liniar increase is hard to get
  - some applications are inherently sequencial
  - communication between nodes – bottleneck
  - centralized components or data tables – bottleneck
  - centralized vs. distributed algorithms

History of distributed systems

- 1954: DYSEAC, a general-purpose synchronous computer, with distributed control
- 1970s-1980s: important research advances in distributed computing, but implementations had modest commercial success
- 1990s: *clusters* of commodity hardware (PCs)
  - improved performance and availability over that of a single PC
- 2000s: *grids* – "super virtual computers" & Virtual Org.
  - a collection of computer resources from multiple locations (organisations/enterprises, etc.) to reach a common goal of them
- After 2007: *clouds* – grids with users from outside the VO
  - service models: SaaS, PaaS, IaaS
  - deployment models: public, private, hybrid

- **Bibliografie obligatorie**

  capitolele despre *sisteme distribuite* din

  – Silberschatz : "*Operating System Concepts*"
  (cap.19 din [OSC10])

  sau

  – Tanenbaum : "*Modern Operating Systems*"
  (cap.8 din [MOS4])

- **Bibliografie recomandată**

  – Andrew S. Tanenbaum : "*Introduction to Distributed Systems*", Prentice Hall, 1995