Sistemi Distribuiti

Laurea in Informatica

Flavio De Paoli

flavio.depaoli@unimib.it

●●● INSIDE&S Lab ●●●

http://inside.disco.unimib.it/

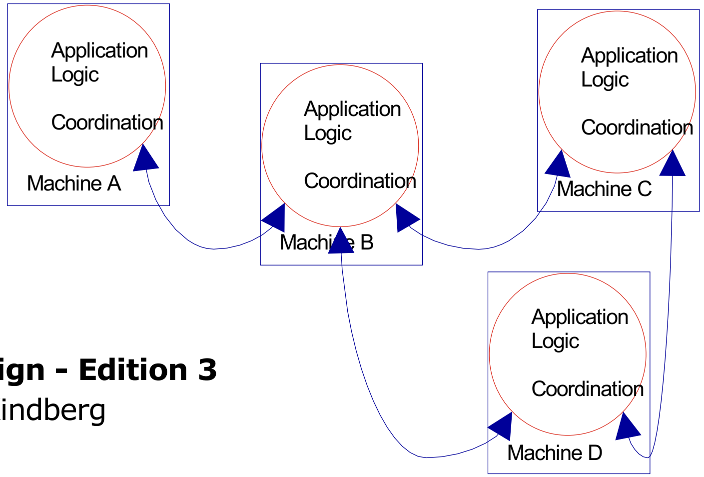
Contenuti

Parte 1 – Concetti e modelli base

- Definizione di Sistema Distribuito
- Architetture software
- Il modello client-server
- Proprietà e caratteristiche fondamentali

Sistema Distribuito Definizione 1

"We define a distributed system as one in which *hardware or software components* located at *networked computers communicate* and *coordinate* their actions only by *passing messages*."

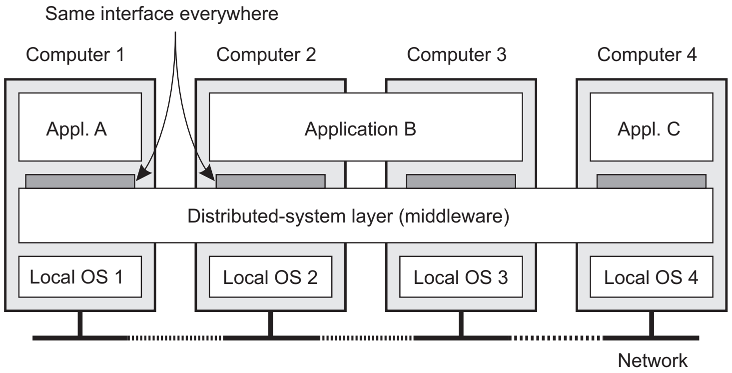


Distributed Systems: Concepts and Design - Edition 3
George Coulouris, Jean Dollimore and Tim Kindberg
Addison-Wesley, ©Pearson Education 2001
ISBN 0201-619-180

Sistema Distribuito Definizione 2

A distributed system is:

a collection of *autonomous computing elements* that appears to its users as a *single coherent system*.



Distributed Systems: Principles and Paradigms
Andrew S. Tanenbaum and Maarten van Steen
Pearson-Prentice Hall, 2002

Distributed Systems 3rd edition
Maarten van Steen and Andrew S. Tanenbaum
Published by Maarten van Steen (2017)
www.distributed-systems.net

Distributed systems

Definition

- A distributed system is a collection of *autonomous computing elements* that appears to its users as a *single coherent system*.

Characteristic features

- Autonomous computing elements, also referred to as nodes, be they *hardware devices* or *software processes*.

- Single coherent system: users or applications perceive a single system ⇒ *nodes need to collaborate*.

Collection of autonomous nodes

Independent behavior

- Each node is *autonomous* and will thus have its *own notion of time*: there is no global clock.
- Leads to fundamental *synchronization* and *coordination* problems.

Collection of nodes

- How to manage *group membership*?
- Groups can be *open* (any node can participate) or *closed* (only selected members can join the group)
- How to know that you are indeed communicating with an *authorized member*?

Coherent system

Essence

- The collection of nodes operates the same, no matter where, when, and how interaction between a user and the system takes place.
- Examples
  - An end user cannot tell where a computation is taking place
  - Where data is exactly stored should be irrelevant to an application
  - If or not data has been replicated is completely hidden

Keyword is *distribution transparency*

The snag: partial failures

- It is inevitable that at any time only a part of the distributed system fails.
- Hiding partial failures and their recovery is often very difficult and in general impossible to hide.

Sintesi caratteristiche

- Caratteristiche fondamentali per tutti i sistemi distribuiti

- Gestione della memoria?
  - Non c’è memoria condivisa
  - Comunicazione via scambio messaggi
  - Non c'è stato globale: ogni componente (nodo, processo) conosce solo il proprio stato e può sondare lo stato degli altri)
- Gestione dell’esecuzione?
  - Ogni componente è autonomo => esecuzione concorrente
  - Il coordinamento delle attività è importante per definire il comportamento di un sistema/applicazione costituita da più componenti
- Gestione del tempo (temporizzazione)?
  - Non c’è un clock globale
  - Non c’è possibilità di controllo/scheduling globale
  - Solo coordinamento via scambio messaggi
- Tipi di fallimenti?
  - Fallimenti indipendenti dei singoli nodi (independent failures)
  - Non c'è fallimento globale

Sortware architecture

●●● INSIDE&S Lab ●●●
http://inside.disco.unimib.it/

Architetture software

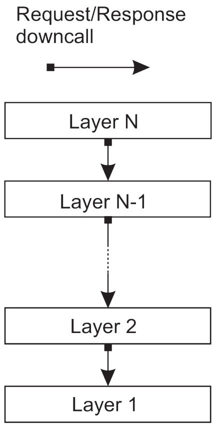
- Una architettura software definisce la struttura del sistema, le interfacce tra i componenti e i pattern di interazione (i protocolli)
- I sistemi distribuiti possono essere organizzati secondo diversi stili architetturali
    - Modello base: Architetture a strati (layered)
        - Sistemi operativi
        - Middleware
    - Architetture a livelli (tier)
        - Le applicazioni client server (2-tier, 3-tier)
    - Architetture basare sugli oggetti
        - Java-Remote Method Invocation (RMI)
    - Architetture centrate sui dati
        - Il Web come file system condiviso
    - Architetture basate su eventi
        - Applicazioni Web dinamiche basate su callback (AJAX)

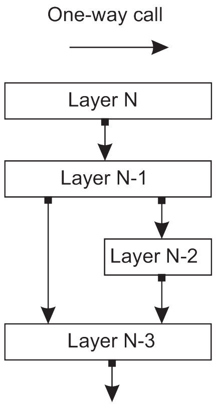
Architetture stratificate (layered)

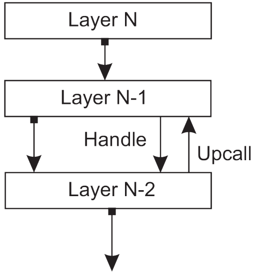
- Layered architecture definition:
  - *A layered architecture is a software architecture that organizes software in layers.*
  - *Each layer is built on top of another more general layer.*
  - *A layer can loosely be defined as a set of (sub)systems with the same degree of generality.*
  - *Upper layers are more application specific and lower are more general.*

- Esempi
  - Sistemi operativi
  - Middleware
  - Protocollo ISO/OSI per le reti di computer

Layered architecture: common cases







Pure layered organization

- Only downcalls to the next layer are allowed
- E.g., ISO/OSI network

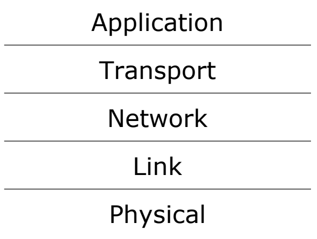
Mixed layered organization

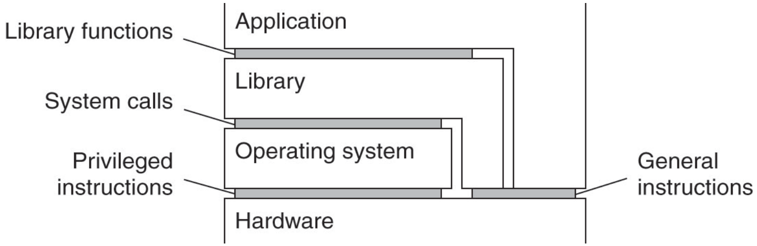
- Downcalls can reach more layers
- E.g., OS organization

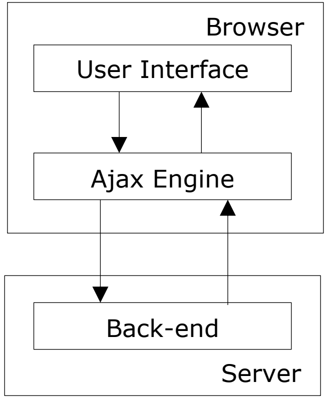
Mixed downcalls and upcalls

- Callbacks are also allowed
- E.g., Web app

Layered architecture: common cases







Pure layered organization

- Only downcalls to the next layer are allowed
- E.g., ISO/OSI network

Mixed layered organization

- Downcalls can reach more layers
- E.g., OS organization

Mixed downcalls and upcalls

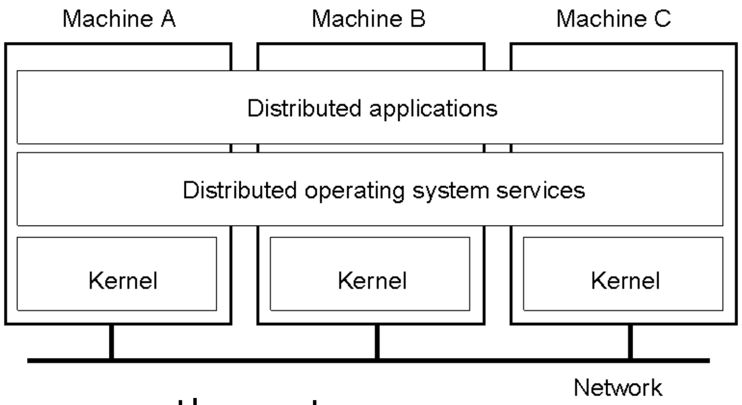
- Callbacks are also allowed
- E.g., Web app

Distributed Operating Systems

- An overview of
  - DOS (Distributed Operating Systems)
  - NOS (Network Operating Systems)
  - Middleware

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

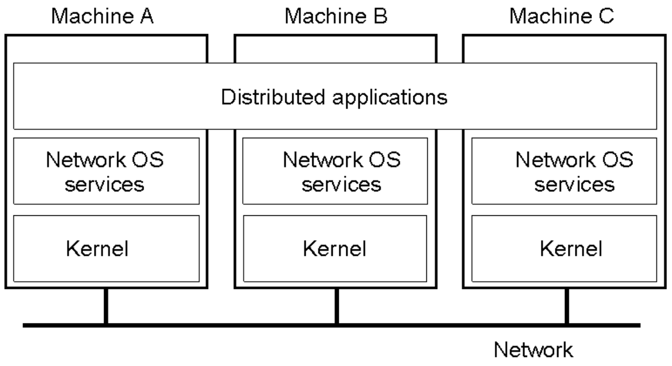
Distributed-Operating Systems



- Users not aware of multiplicity of machines
  - Access to remote resources like access to local resources
- Data Migration
  - Transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
- Computation Migration
  - Transfer the computation, rather than the data, across the system
- Process Migration – execute an entire process, or parts of it, at different sites
  - **Load balancing** – distribute processes across network to even the workload
  - **Computation speedup** – subprocesses can run concurrently on different sites
  - **Hardware preference** – process execution may require specialized processor
  - **Software preference** – required software may be available at only a particular site
  - **Data access** – run process remotely, rather than transfer all data locally

Network-Operating Systems

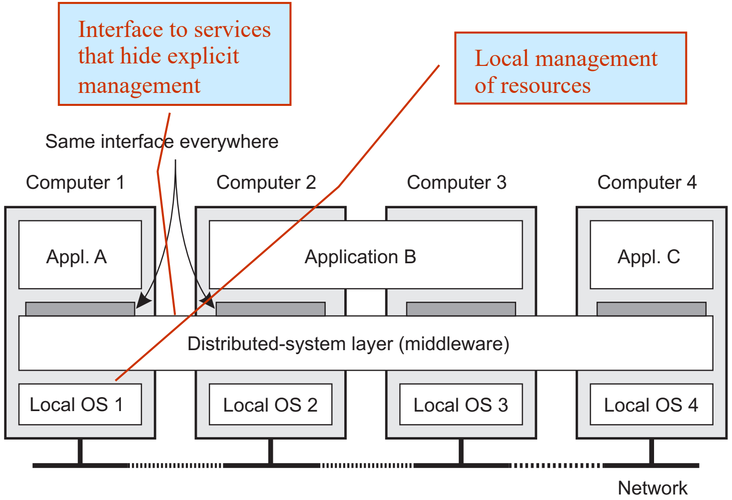
- Users are aware of multiplicity of machines.
- NOS provides explicit communication features
  - Direct communication between processes (socket)
  - Concurrent (i.e., independent) execution of processes that from a distributed application
  - Services, such as process migration, are handled by applications

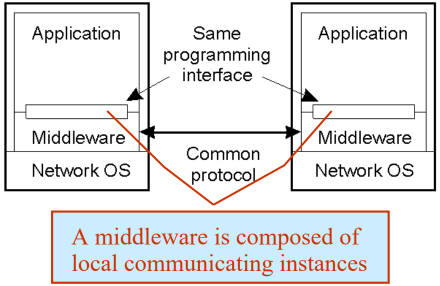


- Access to resources of various machines is done explicitly by:
  - Remote logging into the appropriate remote machine (telnet, ssh)
  - Remote Desktop (Microsoft Windows)
  - Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism

Middleware

- Distributed Operating Systems
  - Make services (e.g., data storage and process execution) *transparent* to applications
  - Rely on homogeneous machines (since they need to run the same software)
- Network Operating Systems
  - Services (e.g., data storage and process execution) *are explicitly managed* by applications
  - Rely on homogeneous machines (since they may run different software)
  - E.g., MacOSX, Windows10, Linux
- Middleware
  - Implements services (one or more) to *make them transparent* to applications
  - E.g., Java/RMI





Middleware services

- Services can address several issues, from general to domain specific
- Naming
  - Symbolic names are used to identify entities that are part of a DS
  - They can be used by registries to provide the real addresses (e.g., DNS, RMI registries), or implicitly by the middleware
- Access transparency
  - … defines and offers a communication model that hides details on message passing
- Persistence
  - ... defines and offers an automatic service for data storage (on file system or DB)
- Distributed transactions
  - … defines and offers a persistence models to automatically ensure consistency on read/write operations (usually on DBs)
- Security
  - … defines and offers models to protect access to data and services (with different levels of permissions) and computation integrity

Comparison between Systems

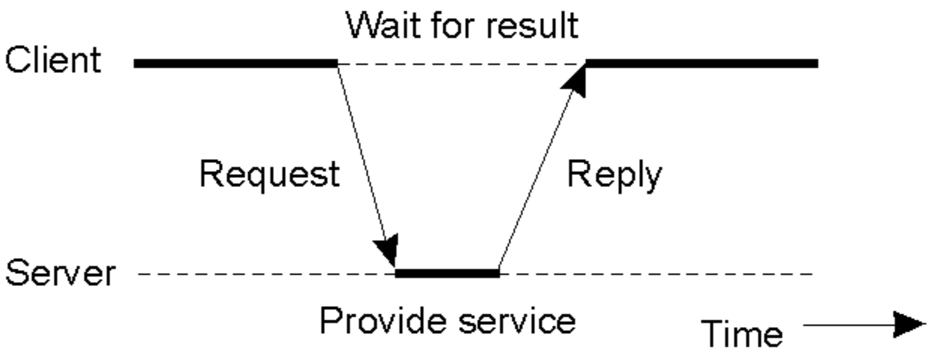
Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open

Il modello Client-Server

●●● INSIDE&S Lab ●●●
http://inside.disco.unimib.it/

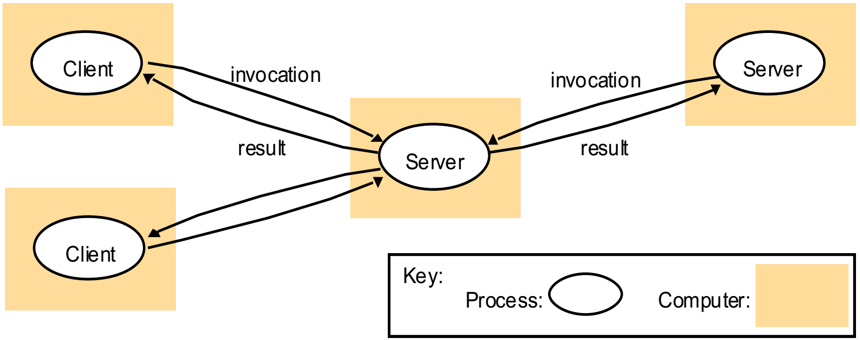
Il modello Client-Server

- Il modello di interazione tra un processo client e un processo server



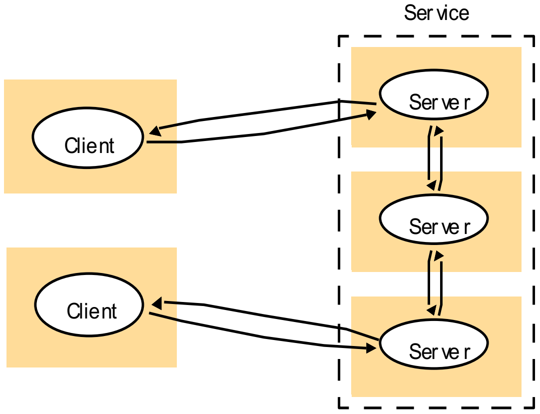
Client/Server

- L’architettura di base prevede che un client acceda ad un server con una richiesta e che il server risponda con un risultato.

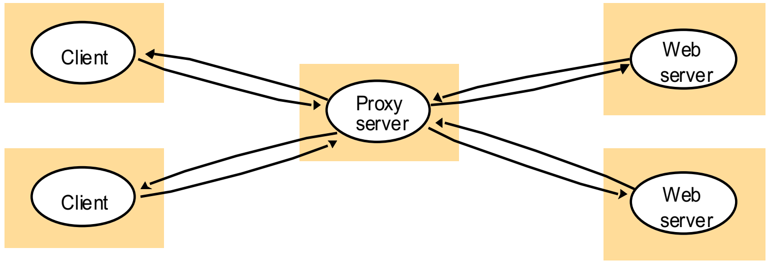


Configurazioni client/server

- Accesso a server multipli



- Accesso via proxy



Fundamental issues

Fundamental issues of any DS

Let's discuss an example: How to collect information on a given car on the company's official Web site? (e.g., Model S)

Think about each step you need to go through.

- Step 1: do you know the name of the manufacturer?
  - If not, What to do?
  - A: search the web to get the *manufacturer name* (=> Telsa)
- Step 2: is the manufacturer name enough?
  - No – Why? What to do?
  - A: search the Web with the *manufacturer name* to get its *address*. (e.g., phone numbers or street address)
- Step 3: is *any address* of the manufacturer enough?
  - No – Why?
  - Q: what kind of address do you need to know?
  - A: a network address (IP address) associated with the *right protocol* (HTTP for the Web) (=> https://www.tesla.com/)

Fundamental issues of any DS

- Step 4: is *any address* of the company website enough?
  - A: maybe not – Why? (e.g., https://www.tesla.com/ja_jp/)
  - Q: what is missing?
  - A: you need to *understand the content* (the language in this case)
  - Q: what is the right address?
  - A: the one that provides information that you can understand
    => https://www.tesla.com/it_it/

To sum up, you need to know:

- the manufacturer *name*
- the *address* associated with each way to contact the company
- the *protocol* to use to talk to the company
- the way to *understand* the exchanged information

Fundamental issues of any DS

Generally speaking, any distributed systems have to face four issues:

- Identify the counterpart
  - Who (process or resource) is my counterpart? = we need to assign names (naming)
- Accessing the counterpart
  - How can I reach a remote process or resource? = we need a reference (access point)
- Communicating 1
  - How can participants exchange messages? = we need to agree and share a format (protocol)
- Communicating 2
  - How can I understand the content of a message = we need to agree on data syntax and semantics (*still an open issue*)

Distribution transparency

Transparency = hide details to users, which may ignore what happens and (more important) cannot influence the provided service

- Naming
  - Symbolic names are used to identify resources that are part of a distributed system
- Access transparency
  - Hide differences in data representation and how a local or remote resource is accessed
- Location transparency
  - Hide where a resource is located in the net
- Relocation or mobility transparency
  - Hide that a resource may be moved to another location while in use
- Migration transparency
  - Hide that a resource may move to another location
- Replication transparency
  - Hide that a resource is replicated
- Concurrency transparency
  - Hide that a resource may be shared by several independent users (ensuring state consistency)
- Failure transparency
  - Hide the failure and recovery of a resource.
- Persistence transparency
  - Hide that a resource is volatile or stored permanently

Degree of transparency

Aiming at full distribution transparency may be too much:

- There are *communication latencies* that cannot be hidden
- *Completely hiding failures* of networks and nodes is (theoretically and practically) **impossible**
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server performed an operation before a crash
- Full transparency will *cost performance*, for example, it takes time
  - Keeping replicas exactly up-to-date with the master
  - Immediately flushing write operations to disk for fault tolerance
- Exposing distribution may be good
  - Making use of location-based services (finding your nearby printers)
  - When dealing with users in different time zones
  - When it makes it easier for a user to understand what’s going on (when e.g., a server does not respond for a long time, report it as failing)

Abstraction and Information Hiding

- Information hiding is a basic principle in Software Engineering
- Separation between
  "*what*" service a component or system provides and
  "*how*" that service has been implemented and deployed
  - The *what*
    - is defined by an *Interface Definition Languages* (*IDL*) to define the *Application Programming Interface* (*API*) of components or systems
    - It should be annotated semantically (i.e., meaning of the exchanged information and prescribed behavior)
  - The *how*
    - is implemented with a tool (e.g., framework, middleware) which is suitable for that specific problem or environment
    - It should be implemented with specific and effective algorithms and technology
- Interfaces should be
  - designed according to shared principles (e.g., communication mechanism)
  - complete (provide everything is needed)
  - neutral (independency from a specific implementation or deployment)

and support

- interoperability, portability, extendibility

Policies versus mechanisms

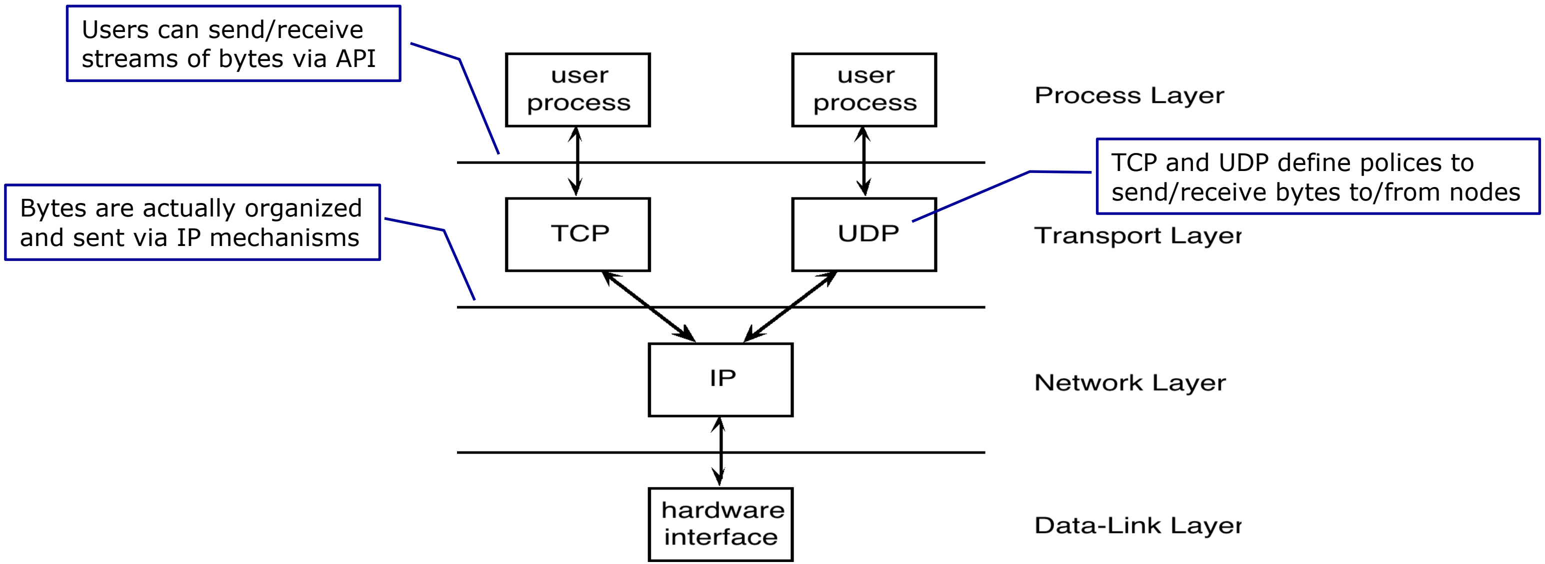
- A distributed systems should be composed of *independent* components
  - Logical independency: each component autonomously provides a service or perform a task
  - Composition: each component uses or collaborates with other components to perform or provide more complex task or services

- It can be facilitated by a clear separation between policies and mechanisms
  - mechanisms: capabilities provided by components
  - policies: how capabilities can be exploited to define a behavior
- In real applications, the separation is difficult to achieve

- Can you provide an example from operating systems?
(suggestion: think about process life cycle)
  - Context switch is a mechanism
  - Round Robin is a policy on top of context switch to define a behavior

Separating policy from mechanism

- An example: TCP and UDP over IP



Separating policy from mechanism

Observation on strict separation

- The stricter the separation between policy and mechanism, the more we need to make ensure proper mechanisms, potentially leading to many configuration parameters and complex management.

Finding a balance

- Hard coding policies often simplifies management and reduces complexity at the price of less flexibility.
- There is no obvious solution.

Distributed Systems basics

●●● INSIDE&S Lab ●●●
http://inside.disco.unimib.it/

Il concetto di protocollo

- Per poter capire le richieste e formulare le risposte i due processi devono concordare un protocollo

- *I protocolli definiscono il formato, l'ordine di invio e di ricezione dei messaggi tra i dispositivi, il tipo dei dati e le azioni da eseguire quando si riceve un messaggio*

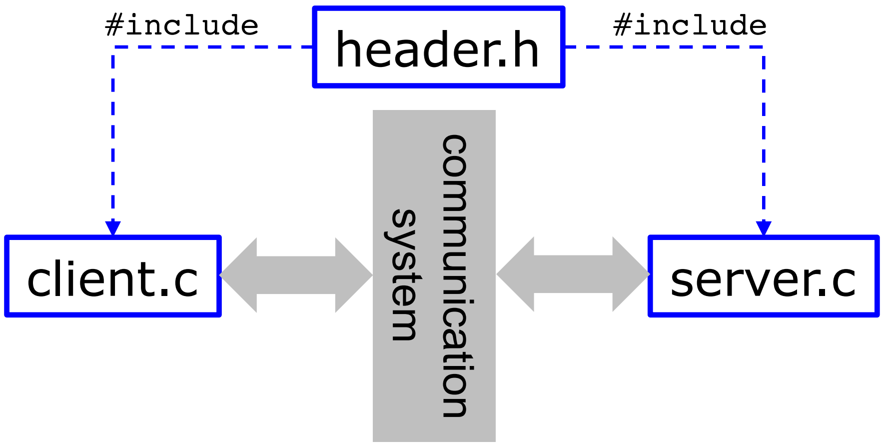
- Le applicazioni su TCP/IP
  - si scambiano *stream di byte* di lunghezza infinita (il meccanismo)
  - che possono essere segmentati in *messaggi* (la politica) definiti da un protocollo condiviso

- Esempi di protocollo applicativi
  - HTTP - HyperText Transfer Protocol
  - FTP - File Transfer Protocol
  - SMTP - Simple Mail Transfer Protocol

Elementi minimi per crare una applicazione

- Definizione del protocollo di comunicazione
- Condivisione del protocollo tra gli attori dell’applicazione

- Un esempio in C



Un esempio: file server remoto (1)

- Il file header.h definisce il protocollo che usano sia il client sia il server.

```
/* Definitions needed by clients and servers. */
#define TRUE 1
#define MAX_PATH 255            /* maximum length of file name */
#define BUF_SIZE 1024           /* how much data to transfer at once */
#define FILE_SERVER 243         /* file server's network address */

/* Definitions of the allowed operations*/
#define CREATE 1                /* create a new file */
#define READ 2                  /* read data from a file and return it */
#define WRITE 3                 /* write data to a file */
#define DELETE 4                /* delete an existing file */

/* Error codes. */
#define OK 0                    /* operation performed correctly */
#define E_BAD_OPCODE -1         /* unknown operation requested */
#define E_BAD_PARAM -2          /* error in a parameter */
#define E_IO -3                 /* disk error or other I/O error */

/* Definition of the message format. */
struct message {
  long source;                  /* sender's identity */
  long dest;                    /* receiver's identity */
  long opcode;                  /* requested operation */
  long count;                   /* number of bytes to transfer */
  long offset;                  /* position in file to start I/O */
  long result;                  /* result of the operation */
  char name[MAX_PATH];          /* name of file being operated on */
  char data[BUF_SIZE];          /* data to be read or written */
};
```

Un esempio: file server remoto (2)

- Struttura di un semplice server che realizza un rudimentale file server remoto.

```
#include <header.h>
void main(void) {
  struct message m1, m2;                            /* incoming and outgoing messages */
  int r;                                            /* result code */

  while(TRUE) {                                     /* server runs forever */
    receive(FILE_SERVER, &m1);                      /* block waiting for a message */
    switch(m1.opcode) {                             /* dispatch on type of request */
      case CREATE: r = do_create(&m1, &m2); break;
      case READ: r = do_read(&m1, &m2); break;
      case WRITE: r = do_write(&m1, &m2); break;
      case DELETE: r = do_delete(&m1, &m2); break;
      default: r = E_BAD_OPCODE;
    }

  m2.result = r;                                    /* return result to client */

  send(m1.source, &m2);                             /* send reply */
  }
}
```

Un esempio: file server remoto (3)

- Un client che usa il servizio per creare una copia di un file

```
#include <header.h>
int copy( char *src, char *dst){                    /* procedure to copy file using the server */
  struct message m1;                                /* message buffer */
  long position;                                    /* current file position */
  long client = 110;                                /* client's address */
  initialize();                                     /* prepare for execution */
  position= 0;

  do {
    m1.opcode = READ;                               /* operation is a read */
    m1.offset = position;                           /* current position in the file */
    m1.count = BUF_SIZE;                            /* how many bytes to read */
    strcpy(&m1.name, src);                          /* copy name of file to be read to message */
    send(FILE_SERVER, &m1);                         /* send the message to the file server */
    receive(client, &m1);                           /* block waiting for the reply */

    /* Write the data just received to the destination file */
    m1.opcode = WRITE;                              /* operation is a write */
    m1.offset =position;                            /* current position in the file */
    m1.count = m1.result;                           /* how many bytes to write */
    strcpy(&m1.name, dst);                          /* copy name of file to be written to buf */
    send(FILE_SERVER, &m1);                         /* send the message to the file server */
    receive(client, &m1);                           /* block waiting for the reply */
    position += m1.result;                          /* m1.result is number of bytes written */
  } while(m1.result > 0);                           /* iterate until done */

  return(m1.result >= 0 ? OK : m1 result);          /* return OK or error code */
}
```

END OF LESSON 1