Architectures for Distributed Systems

Architectural styles are formulated in terms of

- **components** with well-defined interfaces
- the way that components are connected to each other through **connectors**
- the **data** exchanged between components

Connector: a mechanism that mediates communication, coordination, or cooperation among components.

Connector examples: facilities for (remote) procedure call, messaging, or streaming.

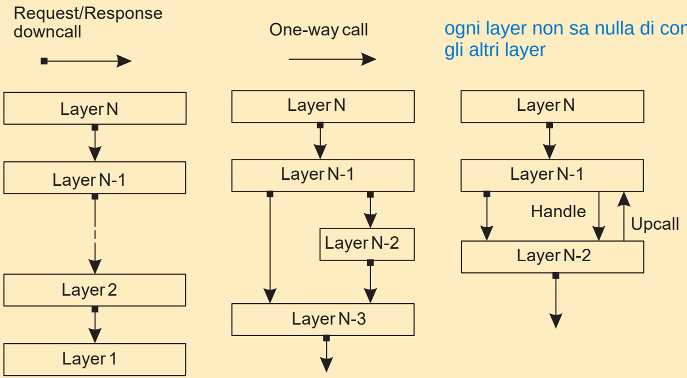
Architectural styles:

- Layered
- Object based
- Resource centered
- Publich-subscribe or event based

Layered architectures

Different layered organizations

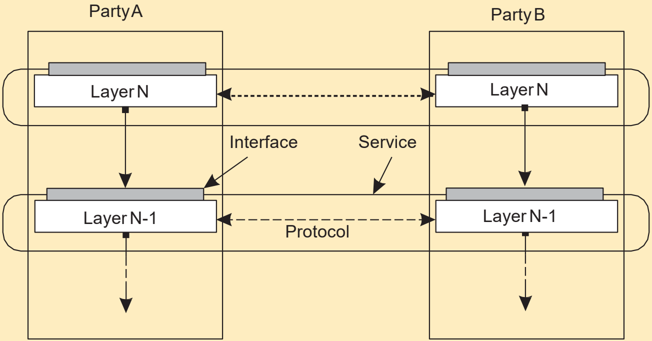
ogni layer non sa nulla di come operano gli altri layer



(a) (b) (c)

Example: communication protocols

Protocol, service, interface



Application Layering

Traditional three-layered view

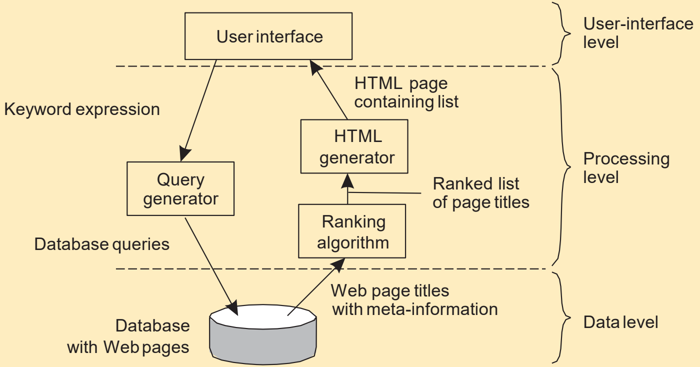
- Application-interface layer contains units for interfacing to users or external applications
- Processing layer contains the functions of an application, i.e., without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Observation

This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

Application Layering

Example: a simple search engine

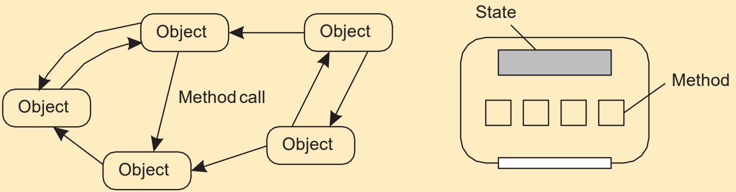


Object-based architectures

object distribuiti nel sistema, che interagiscono fra di loro tramite metodi

Essence

Components are objects, connected to each other through procedure calls. Objects may be placed on different machines; calls can thus execute across a network.

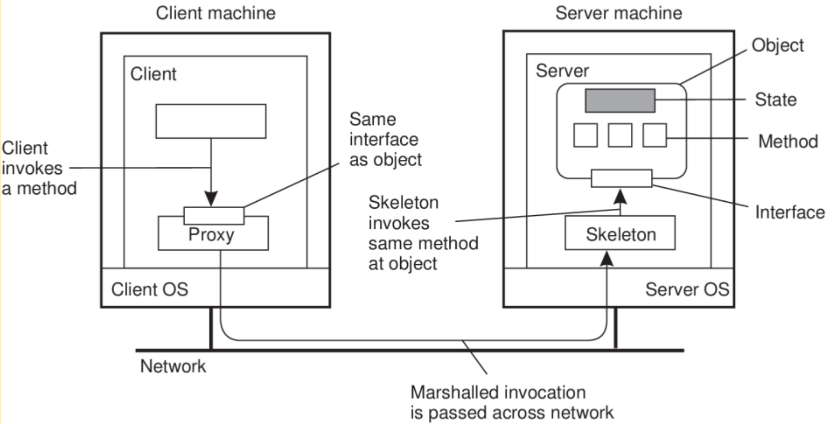


Encapsulation

Objects are said to encapsulate data and offer methods on that data without revealing the internal implementation.

Object based architectures

Remote object with client-side proxy



Evolution: Service Oriented architectures, service composition

Resource centered architectures

Essence

View a distributed system as a collection of resources, individually managed by components. Resources may be added, removed, retrieved, and modified by (remote) applications.

1. Resources are identified through a single naming scheme
2. All services offer the same interface
3. Messages sent to or from a service are fully self-described
4. After executing an operation at a service, that component forgets everything about the caller

Example: RESTful basic operations

utilizza il restful model quindi con le seguenti operazioni

Operation	Description
`PUT`	Create a new resource
`GET`	Retrieve the state of a resource in some representation
`DELETE`	Delete a resource
`POST`	Modify a resource by transferring a new state

Example: Amazon's Simple Storage Service

Essence

Objects (i.e., files) are placed into buckets (i.e., directories). Buckets cannot be placed into buckets. Operations on `ObjectName` in bucket `BucketName` require the following identifier:

http://BucketName.s3.amazonaws.com/ObjectName

Typical operations

All operations are carried out by sending HTTP requests:

- Create a bucket/object: `PUT`, along with the URI
- Listing objects: `GET` on a bucket name
- Reading an object: `GET` on a full URI

On interfaces

Issue

Many people like RESTful approaches because the interface to a service is so simple. The catch is that much needs to be done in the parameter space.

Amazon S3 SOAP interface

Bucket operations	Object operations
`ListAllMyBuckets`	`PutObjectInline`
`CreateBucket`	`PutObject`
`DeleteBucket`	`CopyObject`
`ListBucket`	`GetObject`
`GetBucketAccessControlPolicy`	`GetObjectExtended`
`SetBucketAccessControlPolicy`	`DeleteObject`
`GetBucketLoggingStatus`	`GetObjectAccessControlPolicy`
`SetBucketLoggingStatus`	`SetObjectAccessControlPolicy`

On interfaces

Simplifications

Assume an interface `bucket` offering an operation `create`, requiring an input string such as `mybucket`, for creating a bucket "mybucket."

SOAP

```
import bucket
bucket.create("mybucket")
```

RESTful

```
PUT "http://mybucket.s3.amazonsws.com/"
```

Publish – Subscribe architectures

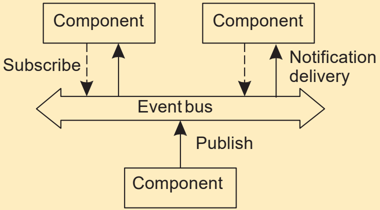
Coordination: temporal and referential coupling

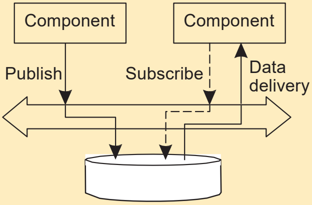
disponibile immediatamente dopo l'invio e quanto ho capito pure per poco tempo

	Temporally coupled	Temporally decoupled
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

Mailbox:mandi la richiesta ma non guardi la risposte la richiesta rimane sempre disponibile nel tempo
Event-based: mandi la richiesta e vieni notificato quando qualcuno risponde alla richiesta

Event-based and Shared data space





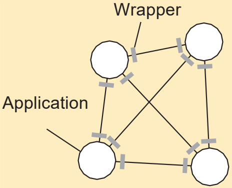
Using legacy to build middleware

Problem

The interfaces offered by a legacy component are most likely not suitable for all applications.

A wrapper or adapter offers an interface acceptable to a client application. Its functions are transformed into those available at the component.

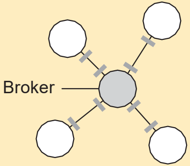
Two solutions: 1-on-1 or through a broker



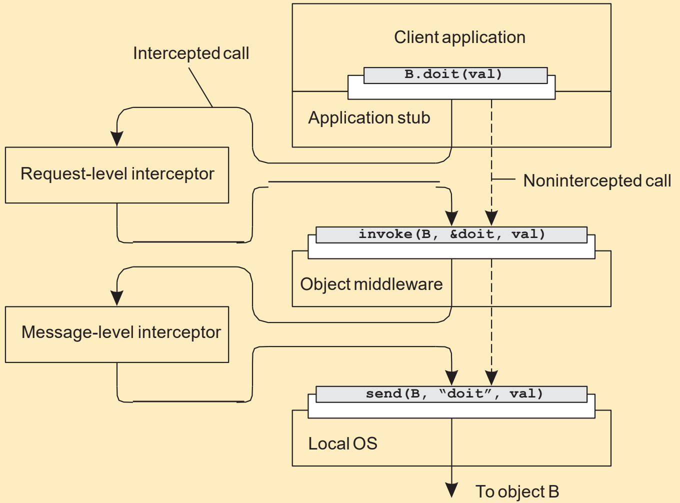
Complexity with N applications

1-on-1: requires $N \times (N-1) = \mathcal{O}(N^2)$ wrappers

broker: requires $2N = \mathcal{O}(N)$ wrappers



Intercept the usual flow of control



Evolution → modifiable middleware

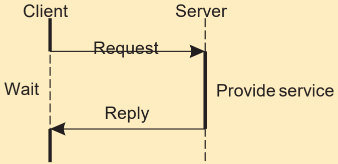
Implementig Architectural styles : Systems

Centralized systems

Basic Client-Server Model

Characteristics:

- There are processes offering services (servers)
- There are processes that use services (clients)
- Clients and servers can be on different machines
- Clients follow request/reply model with respect to using services

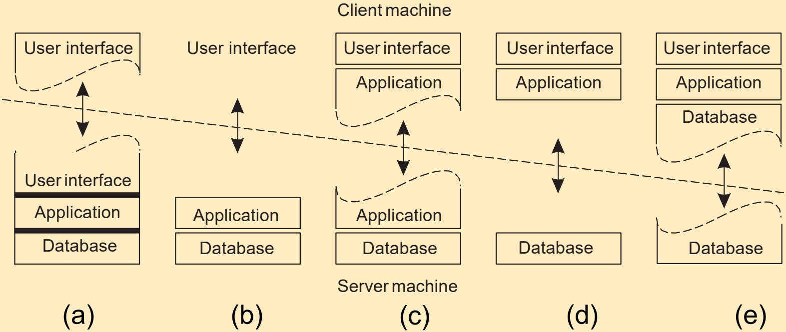


Multi-tiered centralized system architectures

Some traditional organizations

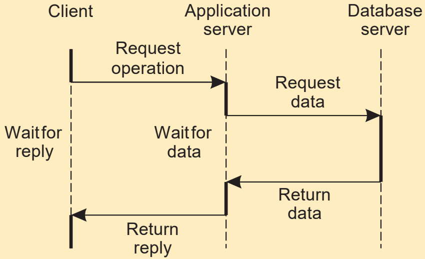
- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine

Traditional two-tiered configurations



Being client and server at the same time

Three-tiered architecture



Alternative organizations

Vertical distribution

Comes from dividing distributed applications into three logical layers, and running the components from each layer on a different server (machine).

Horizontal distribution

A client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set.

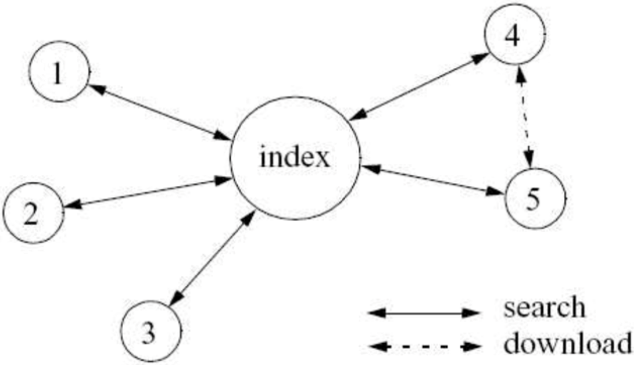
Peer-to-peer architectures

Processes are all equal: the functions that need to be carried out are represented by every process $\Rightarrow$ each process will act as a client and a server at the same time (i.e., acting as a servant).

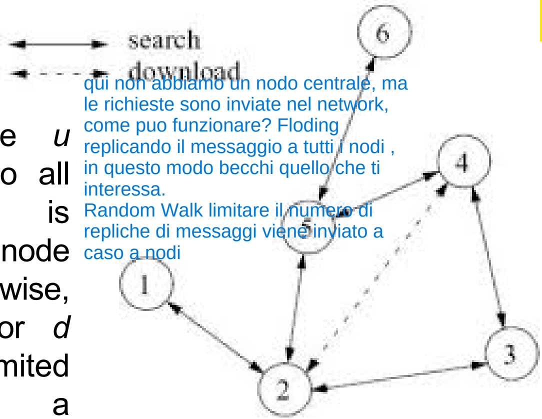
Centralized directory model

tipo piu semplice

come trovare il server che si occupa del servizio a cui sono interessato, si passa per il nodo centrale che si occupa della gestione



Request routing models



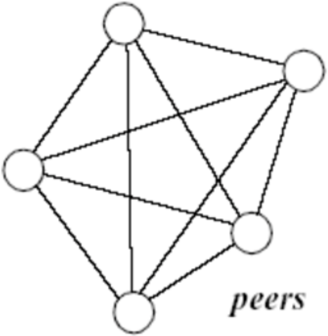
qui non abbiamo un nodo centrale, ma le richieste sono inviate nel network, come puo funzionare? Floding replicando il messaggio a tutti i nodi , in questo modo becchi quello che ti interessa.
Random Walk limitare il numero di repliche di messaggi viene inviato a caso a nodi

Flooding: issuing node *u* passes request for *d* to all neighbors. Request is ignored when receiving node had seen it before. Otherwise, *v* searches locally for *d* (recursively). May be limited by a Time-To-Live: a maximum number of hops

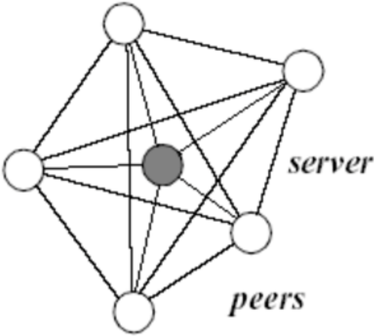
Random walk: issuing node *u* passes request for *d* to randomly chosen neighbor, *v* . If *v* does not have *d* , it forwards request to one of *its* randomly chosen neighbors, and so on.

Introduction: P2P overlay models

P2P= pear to pear system



Pure P2P



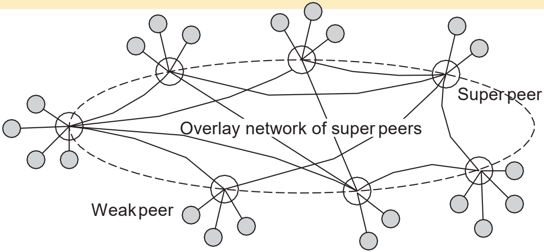
Hybrid P2P

Super-peer networks

Essence

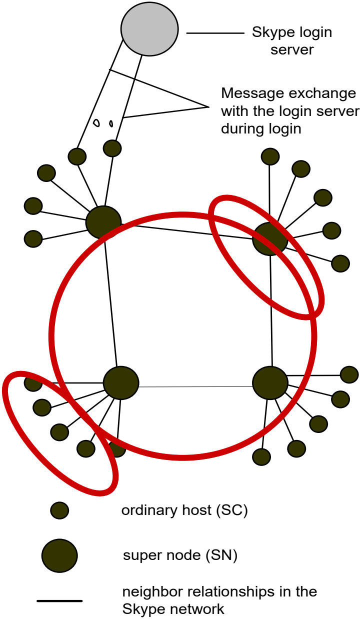
It is sometimes sensible to break the symmetry in pure peer-to-peer networks:

- When searching in unstructured P2P systems, having index servers improves performance
- Deciding where to store data can often be done more efficiently through brokers.



Example: the Skype network

The Skype Network



- Ordinary host (OH)
  - A Skype client
- Super nodes (SN)
  - A Skype client
  - Has public IP address, ‘sufficient’ bandwidth, CPU and memory
- Login server
  - Stores Skype id’s, passwords, and buddy lists
  - Used at login for authentication

Skype's principle operation: *A* wants to contact *B*

Both *A* and *B* are on the public Internet

A TCP connection is set up between *A* and *B* for control packets.
The actual call takes place using UDP packets between negotiated ports.

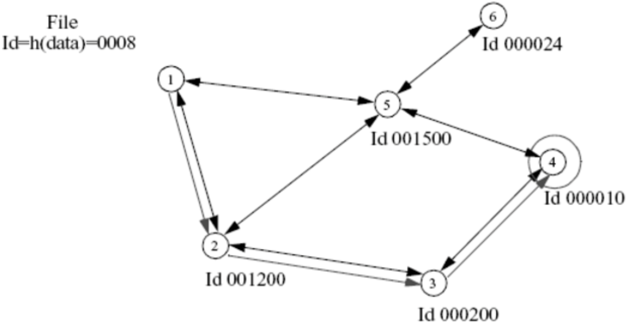
A operates behind a firewall, while *B* is on the public Internet

A sets up a TCP connection (for control packets) to a super peer *S*
S sets up a TCP connection (for relaying control packets) to *B*
The actual call takes place through UDP and directly between *A* and *B*

Both *A* and *B* operate behind a firewall

A connects to an online super peer *S* through TCP
S sets up TCP connection to *B*.
For the actual call, another super peer is contacted to act as a relay *R*: *A* sets up a connection to *R*, and so will *B*.
All voice traffic is forwarded over the two TCP connections, and through *R*.

Document routing model



Structured P2P

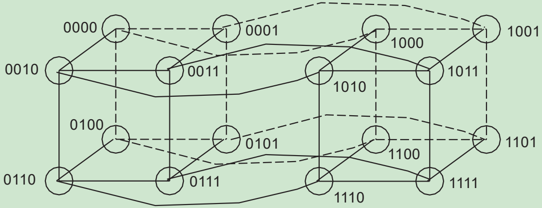
Essence

Make use of a semantic-free index: each data item is uniquely associated with a key, in turn used as an index. Common practice: use a hash function

$$key(data\ item) = hash(data\ item\text{'s}\ value).$$

P2P system now responsible for storing (*key*,*value*) pairs.

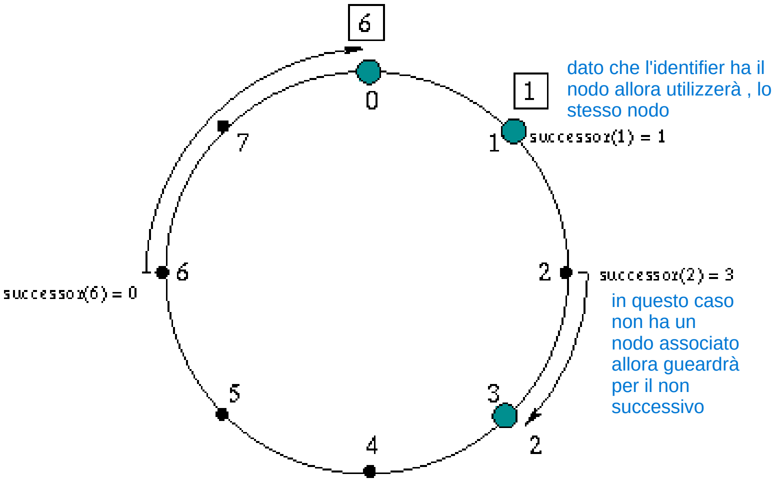
Simple example: hypercube



Looking up d with key $k \in \{0,1,2,\ldots,2^4-1\}$ means routing request to node with identifier k.

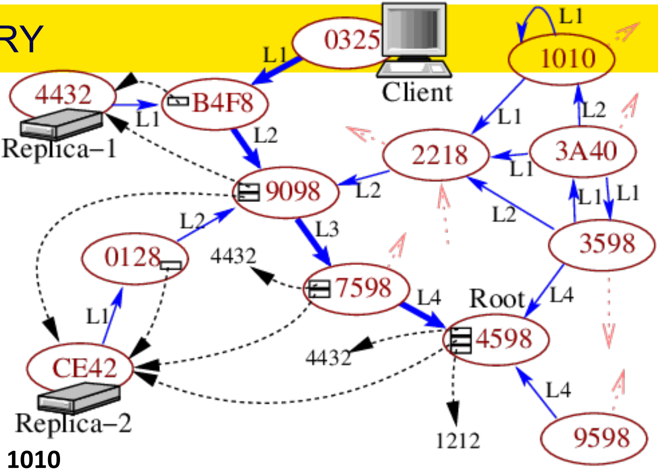
DHT: CHORD

solo 3 nodi ma 8 identifier



DHT: TAPESTRY

- **Nodes connected via links (solid arrows)**

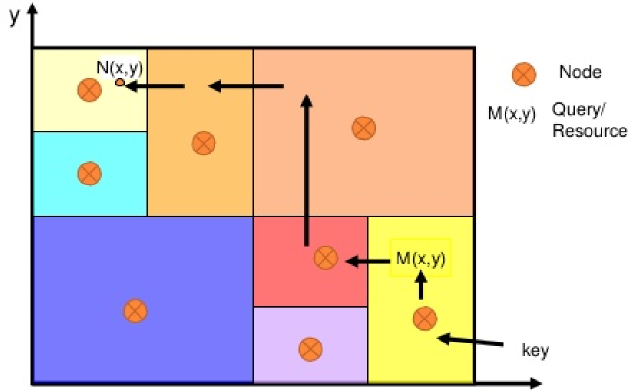


- **Nodes route one digit at a time:**
  e.g. search 4598 from 1010

  1010 → 2218 → 9098 → 7598 → 4598
- **Objects are associated with one particular " root " node (e.g. 4598).**
- **Servers publish replicas by sending messages toward root, leaving back-pointers (dotted arrows).**
- **Clients route directly to replicas by sending messages toward root until encountering pointer (e.g. 0325 → B4F8 → 4432)**

Content Addressable Network

- d-dimensional space with n zones where d=2 and n=8
- 2 zones are neighbor if d-1 dim overlap
- Algorithm: Choose the neighbor nearest to the destination



P2P architecture's comparison

P2P System	Algorithm Comparison Criteria		
	Parameters	**Hops to locate data**	**Reliability**
Napster	none	constant	Central server returns multiple download locations, client can retry
Gnutella	none	no guarantee	receive multiple replies from peers with available data, requester can retry
Chord	N - number of peers in network	$\log N$	replicate data on multiple consecutive peers, app retries on failure
CAN	N - number of peers in network d - number of dimensions	$d \cdot N^{1/d}$	multiple peers responsible for each data item, app retries on failure
Tapestry	N – number of peers in network b – base of the chosen identifier	$\log_b N$	replicate data across multiple peers, keep track of multiple paths to each peer
Pastry	N - number of peers in network b - base of the chosen identifier	$\log_b N$	replicate data across multiple peers, keep track of multiple paths to each peer

Hybrid systems: Edge-server architecture

Essence

Systems deployed on the Internet where servers are placed at the edge of the network: the boundary between enterprise networks and the actual Internet.