



Università degli Studi di Bologna
Scuola di Ingegneria

**Corso di
Reti di Calcolatori T**

Generalità, obiettivi e modelli di base

Antonio Corradi

Anno accademico 2023/2024

OGGETTO del CORSO

Reti di calcolatori e Sistemi distribuiti

Definizione delle architetture di interesse nel corso

Insieme di sistemi distinti in località diverse che usano la comunicazione e la cooperazione per ottenere e mostrare all'utente risultati coordinati

I sistemi distribuiti sono molto comuni e pervasivi e con forti motivazioni all'uso, anche se più complessi dei sistemi concentrati, per la possibilità di

- accesso a risorse remote da dovunque**
- condivisione risorse remote come se fossero locali**

Interesse rinnovato negli anni (con l'avvento e la diffusione dei sistemi globali e mobili)

DIMENSIONI dei SISTEMI d'INTERESSE

Utilizziamo sistemi di **piccola** (pochi nodi), **media** (decine),
grande dimensione (globali tipo Internet)

Molti problemi teorici (COMPLESSITÀ)

Concorrenza: moltissimi processi possono eseguire

Nessun tempo globale: non sincronizzazione degli orologi

Fallimenti indipendenti: molte cause di fallimento, crash di macchine e possibili problemi di rete

Vantaggi

- **TRASPARENZA della LOCAZIONE (nomi)**
- **DINAMICITÀ del SISTEMA (molta flessibilità)**
- **QUALITÀ dei SERVIZI (QoS – Quality of Service)**

SISTEMI DISTRIBUITI

MOTIVAZIONI tecniche ed economiche

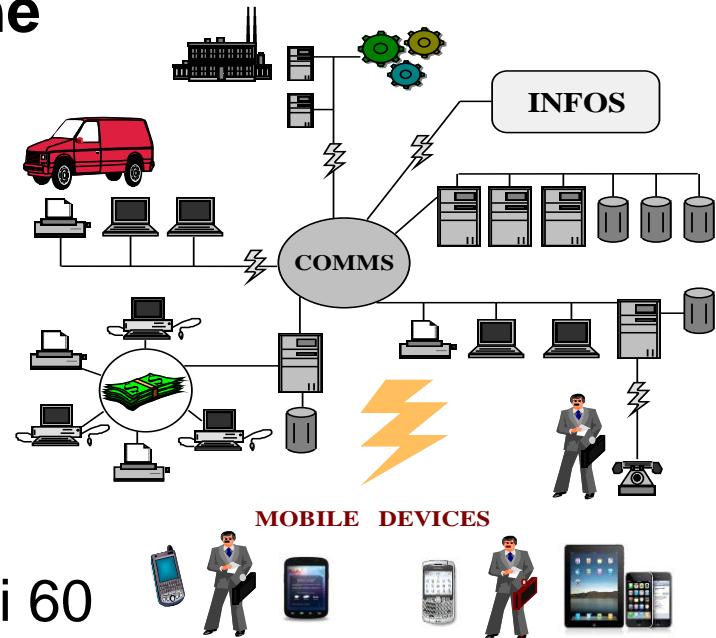
Network locali e globali:

- wide WAN,
- locali LAN, e anche
- reti wireless ...

Richieste distribuite per domande distribuite con accessi eterogenei

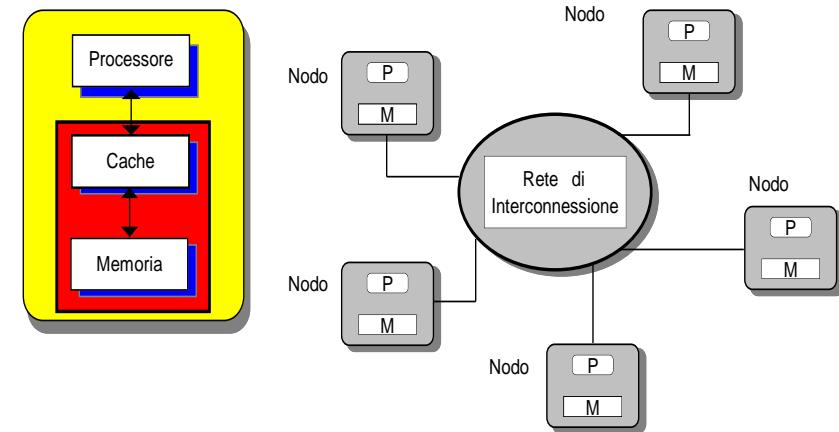
Primo settore: prenotazioni aeree - anni 60

- **accessibilità** e **condivisione** delle risorse
- **affidabilità (dependability)** per tollerare guasti
- **uniformità** in **crescita** e **scalabilità**
(indipendenza in prestazioni dal numero dei nodi del sistema)
- **apertura** del sistema o **openness** (capacità di evolvere e operare seguendo le ‘naturali’ evoluzioni delle specifiche)



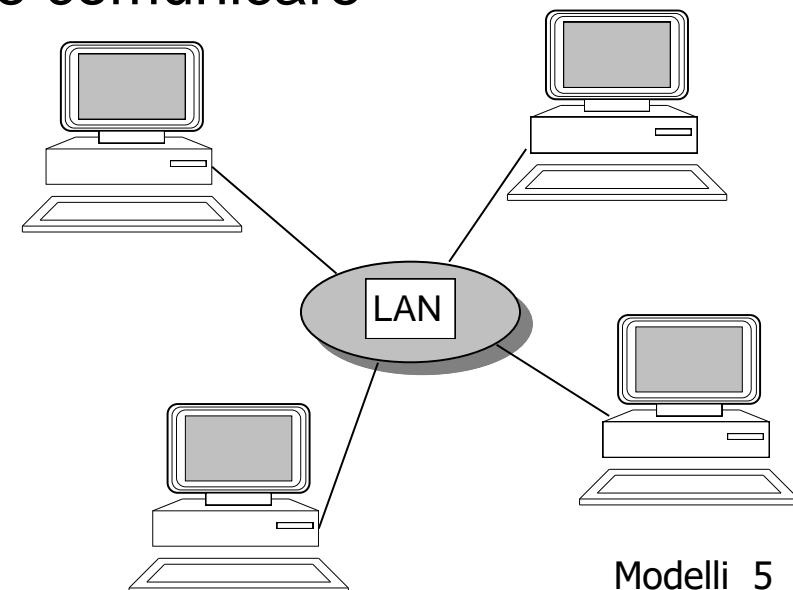
TIPICA ARCHITETTURA di INTERESSE

Architetture MIMD
(Multiple Instruction Multiple Data)
fatte di **nodi diversi**



Ogni nodo processore collegato
alla memoria privata organizzata a livelli e indipendente
e capace di operare autonomamente e comunicare

Reti di workstation
Calcolatori indipendenti connessi
da una o più reti locali



Il corso si occupa non di
architetture hardware
ma di **architetture software**

ARCHITETTURA SOFTWARE (?)

Per una applicazione distribuita

Analisi, sviluppo, sulla base di un algoritmo e sua codifica in linguaggi di programmazione fino alla esecuzione

MAPPING

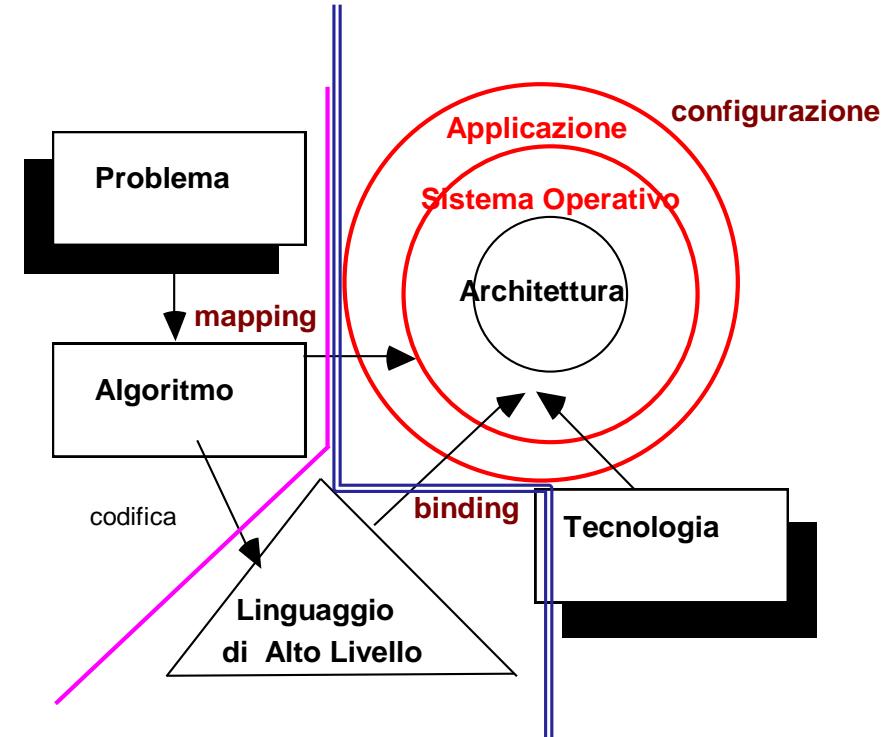
configurazione per l'architettura e
allocazione della parte logica

a risorse fisiche e località

BINDING di risorse

come ogni entità della applicazione
si lega alle risorse del sistema

Gestione STATICÀ vs. DINAMICA
del legame di **BINDING**



Nel corso massimo interesse per **esecuzione e supporto**

Modello di azione

Per una applicazione distribuita

Dobbiamo considerare che sono necessari

- **processi come attività**
- **azioni specifiche su dati locali**
- **protocolli per ottenere azioni coordinate su più nodi**

Rispetto a sistemi in ambito locale abbiamo aggiunto i **protocolli che consentono di ottenere risultati congiunti**

Protocollo

complesso di regole e procedure cui ci si deve attenere in determinate attività per la esecuzione corretta (vedi galateo)

Uso di Standard

di fatto (Cliente/Servitore) o di comitato (OSI, TCP/IP,)

Modelli di Comunicazione e Servizi

Per una applicazione distribuita

La **comunicazione tra processi** avviene attraverso protocolli ispirati a **modelli di due tipi**

- **Cliente Servitore**
- **Scambio di Messaggi**

I due modelli sono diversi in livello e capacità espressive, anche se consentono di esprimere qualunque azione si possa desiderare ed esprimere nel distribuito

Rispetto al concentrato abbiamo aggiunto i **protocolli**

- **Nel C/S, il modello di servizio prevede un fruitore e un fornitore dello stesso servizio**
- **nello Scambio di messaggi, abbiamo un mittente che manda informazioni ad uno o più riceventi senza prevedere risposta**

I due Modelli di COMUNICAZIONE

I modelli Client/Server e Scambio di Messaggi sono entrambi capaci di modellare ogni tipo di applicazione come capacità espressiva ma hanno caratteristiche diverse

Ognuno dei due si può implementare con l'altro

Client Server	Scambio di Messaggi
sincrono	asincrono
bloccante	non bloccante
comunicazione diretta	comunicazione indiretta
singolo ricevente	Riceventi molteplici

Noi useremo lo scambio di messaggi per fare il C/S (socket) e poi strumenti di C/S di vello applicativo

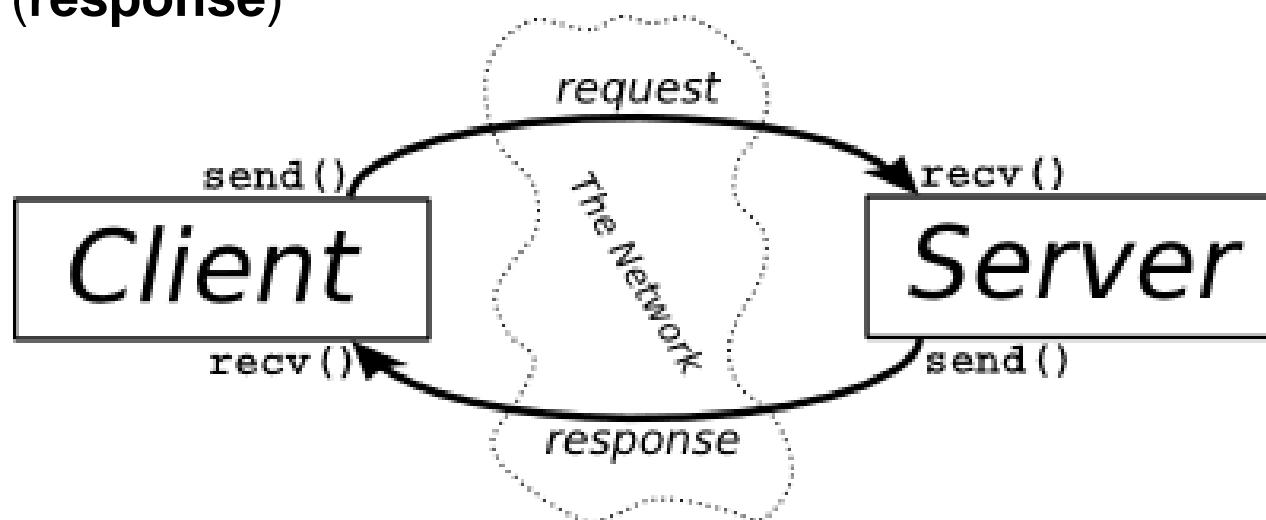
C / S usando SCAMBIO MESSAGGI

Una comunicazione Client/Server si può implementare con due scambi di messaggi e alcune regole di protocollo per i processi interagenti

Considerando due API locali `send()` e `receive()`

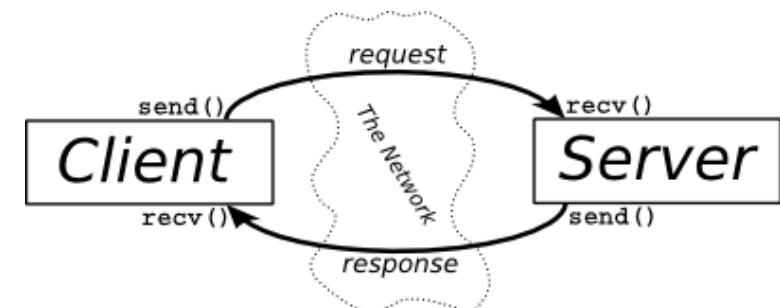
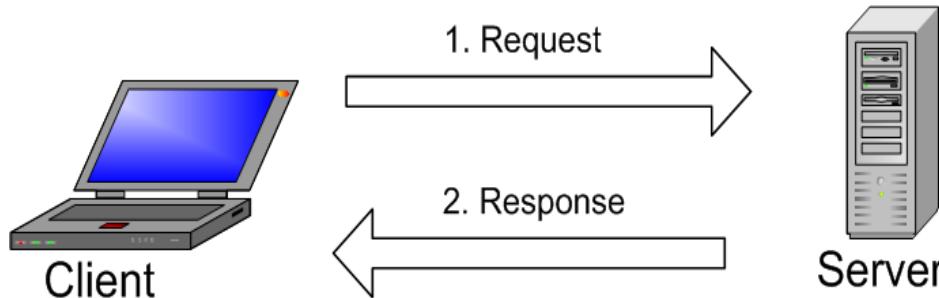
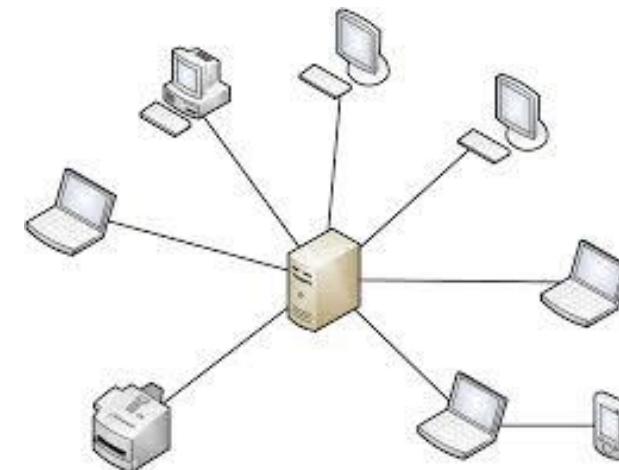
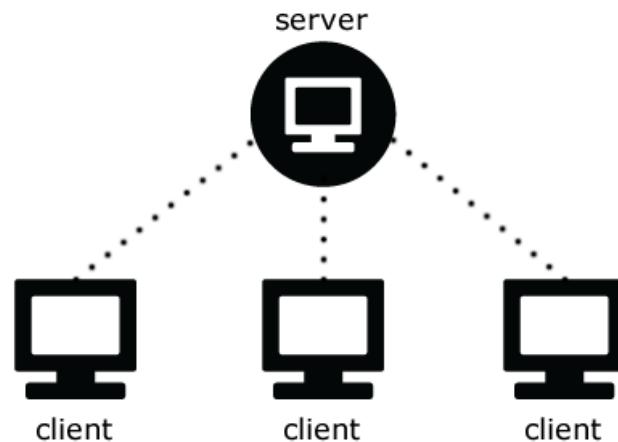
Il Cliente invia con una `send (request)` la richiesta al servitore e aspetta il risultato, che riceve con `recv (answer)`

Il Servitore considera una richiesta tra quelle in arrivo depositate con `recv (req)` e, dopo avere prodotto un risultato, lo manda al cliente con una `send (response)`



CLIENTE / SERVITORE !!!

Abbiamo molte idee del **modello Clienti / Server** a diversi livelli di dettaglio



Una diffusa applicazione distribuita

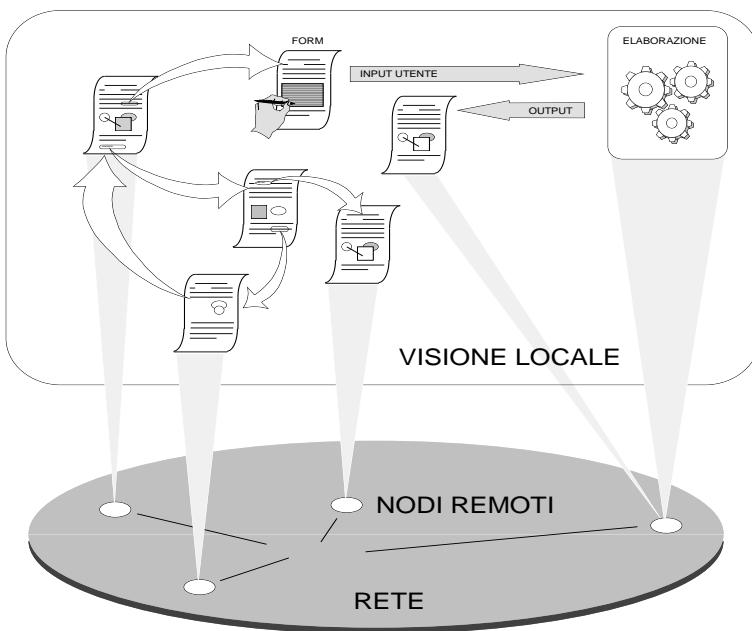
Il primo esempio è quello di un **accesso a pagine Web**

Il sistema non è tanto distribuito ma piuttosto semplicemente in rete

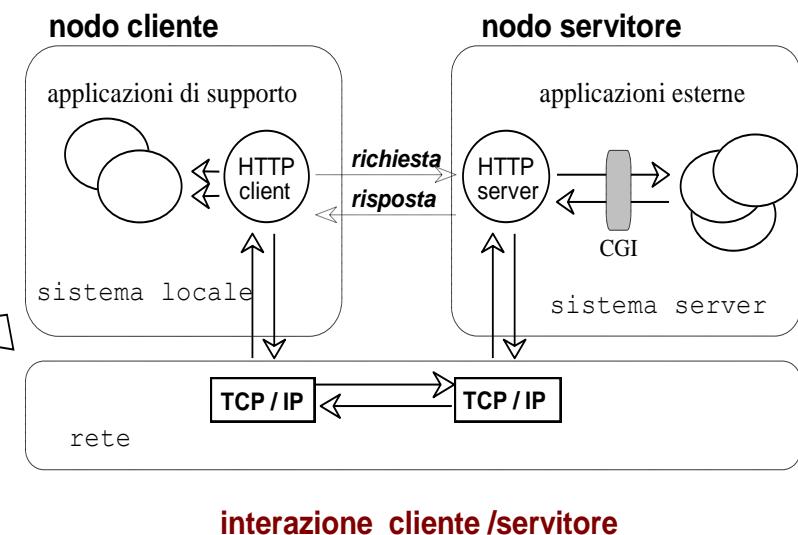
...

Un utente accede alle pagine Web che sono depositate e mantenute da vari server (in modo trasparente alla allocazione)

Visione **utente**
(architettura logica)



Visione **sistema**



ARCHITETTURA in GIOCO: WEB

I sistemi prevedono molti rapporti cliente/servitore

L'utente è cliente del browser

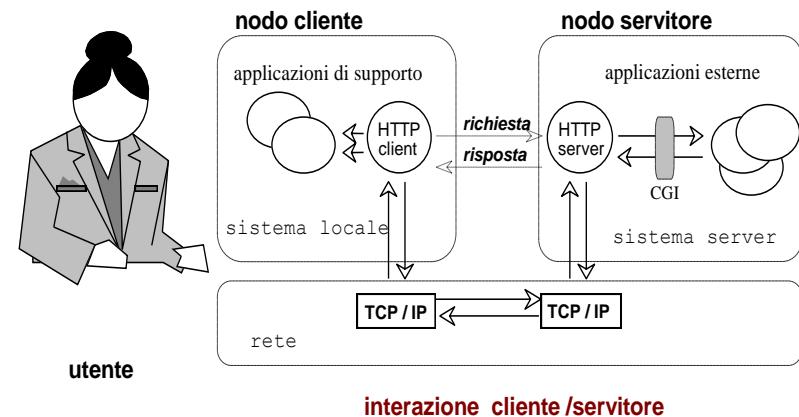
Il browser sulla macchina cliente è cliente del nodo server

Il cliente TCP è cliente della driver TCP del nodo server

Il cliente IP invia le cose al successivo (mittente /destinatario)

Ma anche si mettono in gioco molte altre relazioni C/S con consistenza e replicazione

- Nodi estremi che fanno cache
- Nodi intermedi o proxy che possono fare cache per molti nodi server
- Altri intermediari di organizzazione e verifiche di freschezza dei dati



interazione cliente /servitore

CLIENTE / SERVITORE

Il modello Cliente / Servitore è un modello di coordinamento tra due entità che cooperano per una comunicazione e un servizio

La implementazione è molto varia

Può comportare scelte e politiche molto diverse

I sistemi lo impiegano in molti modi, noi **definiamo un nostro default per il C / S**

Sempre Molti a 1 (N:1 o 1:N ossia 1 servitore, più clienti)

In termini di caratteristiche e proprietà presentate

- **Sincrono / Asincrono**
- **Bloccante / Non bloccante**
- **Asimmetrico / Simmetrico**
- **Dinamico / Statico**

CLIENTE / SERVITORE BASE

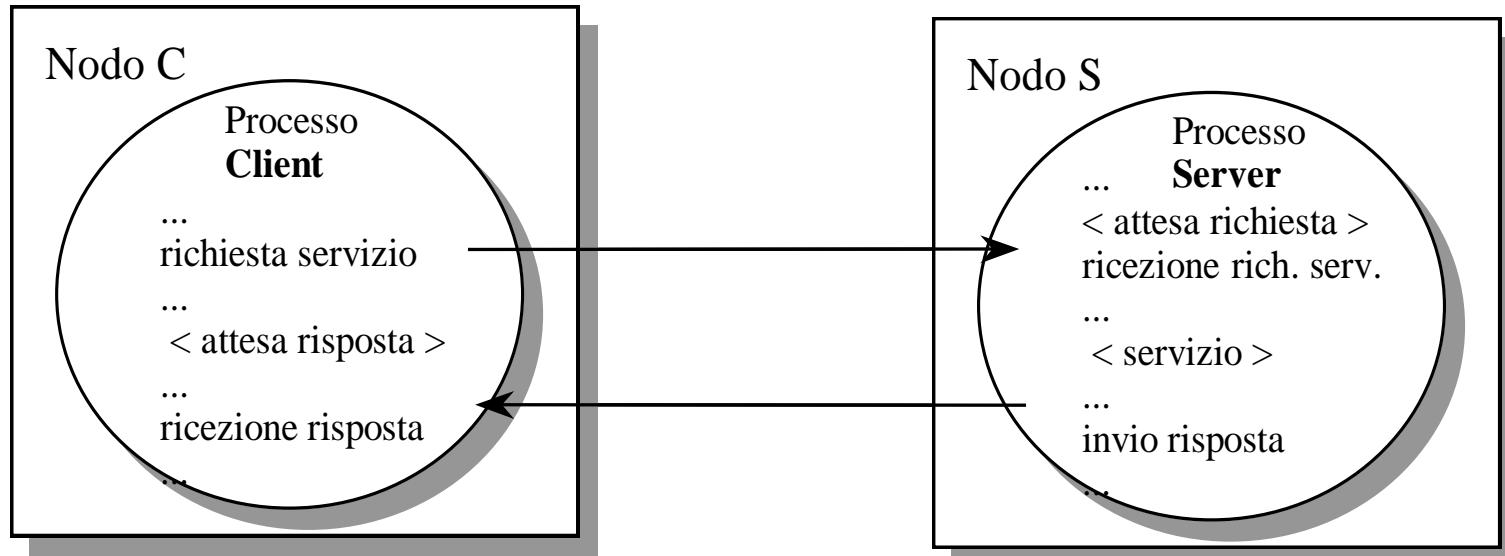
Un modello a **due entità**: il **CLIENTE** chiede, il **SERVER** risponde ai **molteplici clienti (clienti)**

il **cliente ha l'iniziativa** e si attiva per richiedere **un servizio e aspettare una risposta**

Il **servitore** deve ricevere la richiesta, **eseguire il servizio** e dare **una risposta al cliente**

Il **servitore** deve **servire molti clienti**

Uso di primitive di comunicazione tra entità remote (send – receive)



CLIENTE / SERVITORE BASE

Un modello a **due entità**: il **CLIENTE** chiede, il **SERVER** risponde ai **molteplici clienti (clienti)**

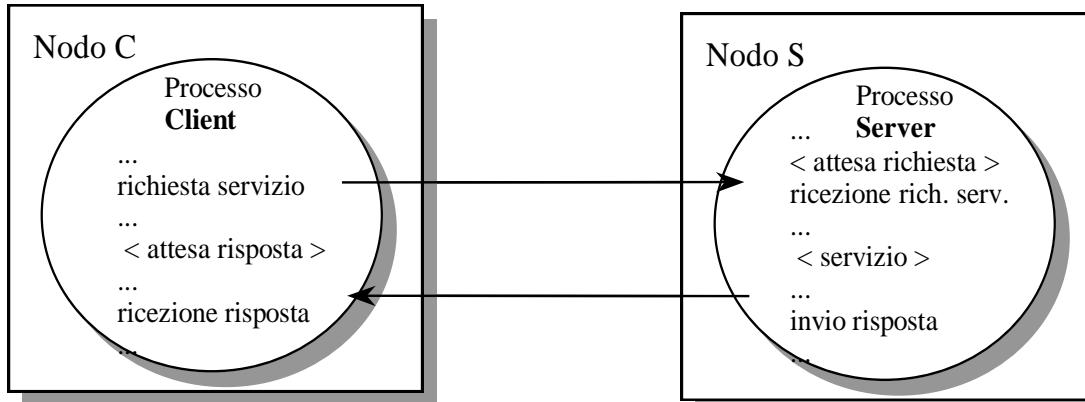
il **cliente** invoca il servizio e ne aspetta il completamento

MODELLO SINCRONO si prevede la risposta (**decisione semantica**)
BLOCCANTE si attende la risposta (**decisione locale**)

il **servitore** attende richieste e le riceve, le realizza e le attua, poi **risponde**
DIVERSE REALIZZAZIONI del server

il modello C/S risolve il problema del **rendez-vous** (per **sincronizzare i processi comunicanti**) con Server come processo sempre in attesa di richieste di servizio

Il supporto non è tenuto ad attivare il processo S alla ricezione di messaggi



**Se il server
non è attivo,
si segnala
errore**

MODELLO CLIENTE / SERVITORE

Il modello di comunicazione è **1 a molti (1 SERVER e N CLIENTI)**

Modello N:1, sincrono, bloccante, asimmetrico, dinamico

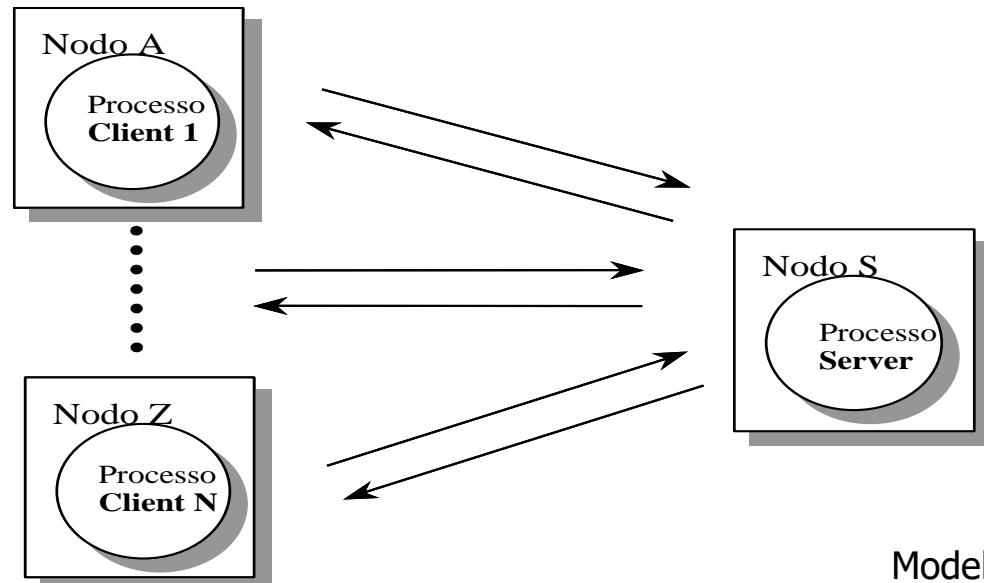
SINCRONO si prevede risposta dal servitore al cliente

BLOCCANTE il cliente aspetta la risposta dal servitore

ASIMMETRICO: il cliente **conosce** il servitore per inviare la invocazione, mentre il servitore **non conosce** a priori i clienti possibili

DINAMICO: il legame (binding) tra **cliente e servitore è dinamico, ossia può cambiare il servitore che risponde alle richieste tra diverse invocazioni**

Dinamicità e Asimmetria sono anche caratteristiche del nostro modello a default



PROGETTO CLIENTE / SERVITORE

Il progetto del **server** è più **complesso** rispetto al progetto del **cliente**

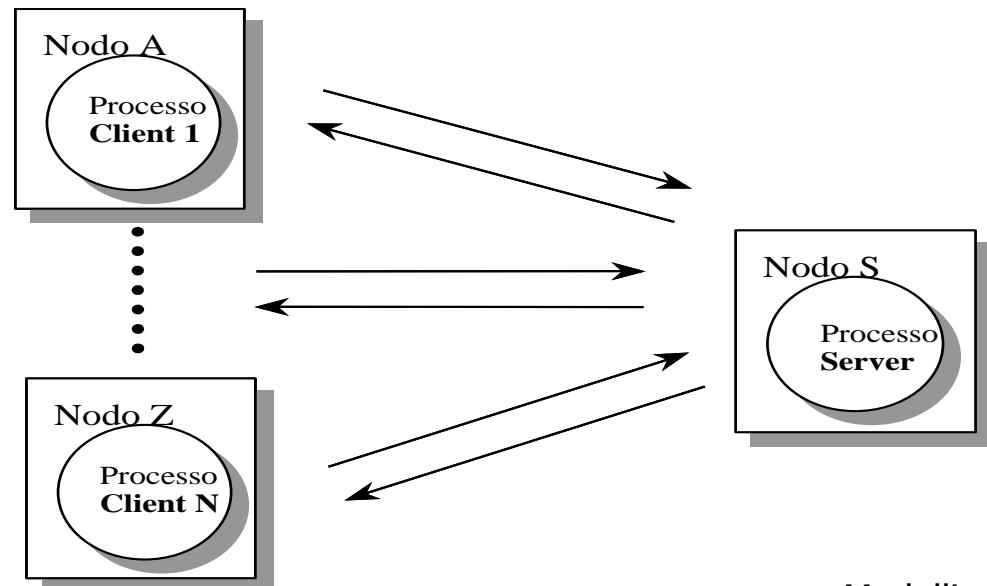
il servizio deve essere sempre pronto alle eventuali richieste

Server dura all'infinito ossia è eterno (demone)

il **Server**, oltre alla **comunicazione**, deve realizzare il servizio con azioni locali, come accedere alle **risorse del sistema** considerando **molteplici clienti** e anche problemi di:

- integrità dei dati
- accessi concorrenti
- **autenticazione utenti**
- **autorizzazione all'accesso**
- **privacy delle informazioni**

**Nel modello base
le trascuriamo**



CLIENTE / SERVITORE A DEFAULT

Il modello Cliente / Servitore a default consiste di

- **Un servitore**
- **Molti clienti**

Che devono cooperare per il progetto attraverso protocollo comune

Il servitore deve essere **presente per ogni cliente**

La **iniziativa per il rapporto** è del cliente

Proprietà

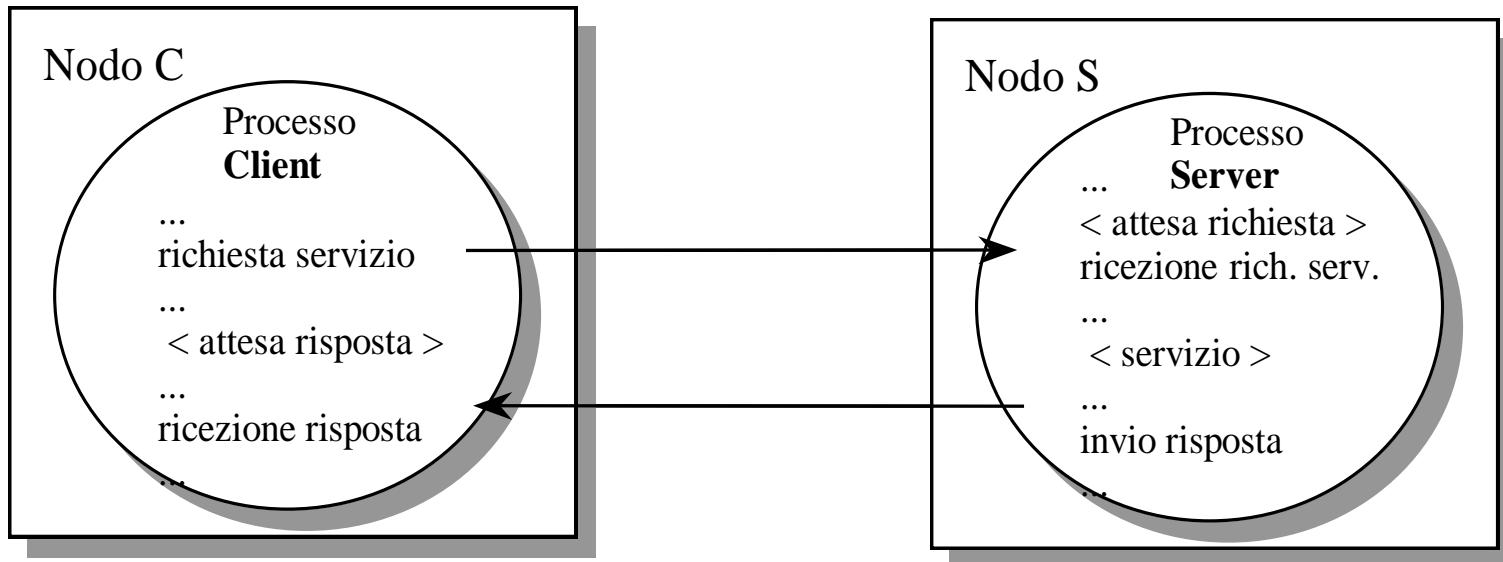
- **Rapporto Molti a 1** (il servitore serve molti clienti)
- **Sincrono** (sia servitore sia clienti prevedono risultato)
- **Bloccante** (il cliente aspetta il risultato)
- **Asimmetrico** (il servitore non conosce il cliente)
- **Dinamico** (il cliente non ha sempre lo stesso servitore)

CLIENTE / SERVITORE BASE

Il modello **CLIENTE / SERVITORE A DEFAULT** viene implementato attraverso comunicazioni tra nodi remoti (*primitive send / receive*)

- e se guasto
- e se servizio lento
- e se si attende troppo
- come quantificare il tempo?
- quanto mantenere la relazione?

Consideriamo la assunzione di lavorare su più nodi



IL CLIENTE (VARIAZIONI)

Focalizziamo il **Client** → tipicamente lavora in modo **sincrono** e **con interazione bloccante**: c'è sempre una risposta e il cliente la aspetta

Punto importante: se la risposta non arriva (entro un certo tempo), possiamo inferire che il server sia down? **NO**

Un server in crash è molto simile a uno congestionato, che ha troppe richieste pendenti da server e non è più capace di onorare le richieste in tempi 'ragionevoli'

Ma per quanto aspetta il Client? Per sempre → non accettabile 😞

Dopo la richiesta, o la risposta arriva o, dopo un intervallo predeterminato detto **timeout**, scatta una eccezione

Il timeout è molto importante nella fase di design:

È un modo di fare recovery dalla possibilità di un errore → Generando eccezione poiché Il server potrebbe essere crashed 😞

La eccezione permette di esprimere una eventuale azione di compensazione

ripetizione? con un altro server? chiudere tutto? riportare all'utente?

VARIAZIONI del CLIENTE

Ragioniamo per una generica **interazione C / S**

il Client deve fare la richiesta e aspettare

Se arriva risposta ok. In caso di problemi, **azione compensativa**

Se non arriva risposta? Non si aspetta per sempre:

Decisione locale di timeout che scatena la **eccezione locale** e poi

- Richiesta ad un altro server ...
- Ripetizione delle richiesta: dopo quanto tempo, quante volte?

La **iniziativa è del Client** che decide se ripetere la richiesta

In caso di insuccesso, si rinuncia ⇒ il server è guasto (?)

Il server potrebbe anche essere **lento e congestionato** nel servizio da richieste precedenti

- Il cliente **aspetta fino ad una risposta** (se ne ha bisogno)
- Il cliente chiede ripetutamente lo stesso servizio ma non attende per ogni richiesta, fino ad una risposta che si possa fornire in tempi accettabili (**polling di ripetizioni**)

IL CLIENTE ... (VARIAZIONI)

Per questo tipo di **interazione C / S oltre il default sincrono**

Modelli verso la asincronia (senza risposta) o sincroni a risposta differita (sincroni non bloccanti)

modello di interazione pull, normale cliente / servitore

Si semplifica il progetto server e il cliente decide quando ripetere la richiesta, quanto spesso e quante volte

Il cliente ha sempre la iniziativa

modello di interazione push, un modello opposto per la consegna del risultato

Il cliente fa la richiesta, **una volta sola**, si **sblocca** e può fare altro

Il server arriva a fare il servizio e ha la **responsabilità di consegna del risultato** al cliente

Il modello push fa diventare il server cliente di ogni suo cliente, scaricando il cliente (*senza cicli attivi di richieste*), ma caricando di ulteriori compiti il servitore stesso

IL SERVITORE ...

Il servitore deve gestire tutta la **interazione C / S** e molti clienti e quindi richiede un **progetto complesso**

- Il Server deve essere sempre presente (**demone**)
- Il Server deve mantenere una coda delle richieste da servire da cui prelevare le operazioni da eseguire
- Il Server o svolge una **operazione alla volta** o è capace di portare avanti **molte operazioni insieme**
 - Server sequenziale o Server concorrente (parallelo o meno)
- Il Server deve dare **continuità alle interazioni con i clienti**
 - Server senza stato o Server con stato della interazione
- Il Server può avere delle **interazioni più complesse e differenziate**
 - Server che manda notifiche ai clienti (?)
 - Server che riconosce i clienti autorizzati (?)
 - Server che si coordina con altri servitori per un servizio coordinato globale (?)

Bisogna considerare quali siano i costi ed il risultato

IL SERVITORE ALL'INTERNO

un Server sequenziale deve aspettare le richieste che sono in coda
e le serve una alla volta

Ciclo infinito

prende una richiesta, la serve, e dà risposta
passa alla richiesta successiva ...

Ciclo di lavoro **sequenziale** molto semplificato

un Server concorrente deve essere capace di servire più di una richiesta alla volta, e quindi estrae una richiesta in coda, ma prima di terminare il servizio e dare risposta, può (e deve) estrarre anche altre richieste in coda e servirle contemporaneamente

Il **server concorrente** prende una richiesta comincia a servirla e prima di dare risposta può lavorare anche su richieste precedenti o attivandone successive...

Ciclo di lavoro **concorrente o parallelo** più complesso

Un server concorrente può essere un **unico processo** con più attività o **parallelo** (molte attività e processi distinti, pesanti o leggeri)

SERVITORE SEQUENZIALE

Servitore iterativo o sequenziale

che serve una richiesta alla volta e **itera** il servizio

Dal punto di vista cliente abbiamo **due indicatori distinti**

- **tempo di elaborazione** (di servizio) di una richiesta T_{Servizio}

T_s tempo per servizio di una richiesta isolata

- **tempo di risposta** osservato dal cliente T_{Risposta}

T_R ritardo totale tra la spedizione della richiesta e l'arrivo
della risposta dal server

$$T_R = T_s + 2 T_c + T_q$$

T_c tempo di comunicazione medio

T_q tempo di accodamento medio

Con lunghe code di richieste, il tempo di risposta può diventare
anche molto maggiore del tempo di elaborazione della richiesta

SERVITORE ITERATIVO

Servitore iterativo

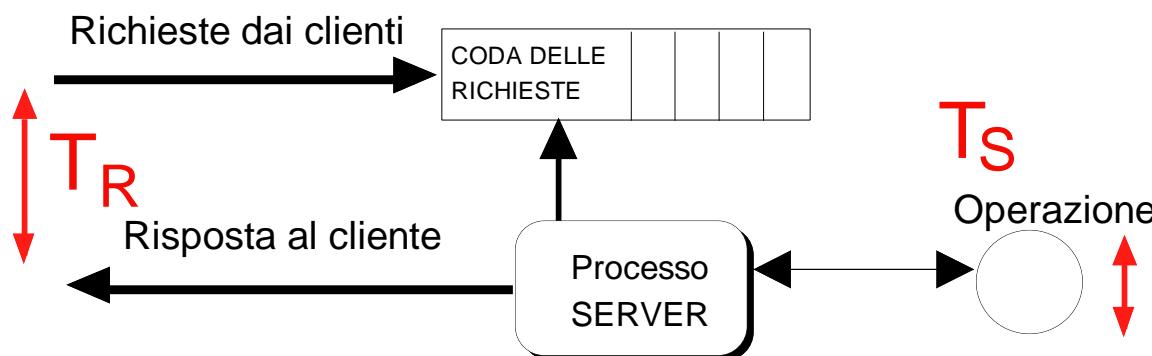
che serve una richiesta alla volta e **accoda le altre in attesa**
(coda delle richieste in attesa di servizio)

Tralasciando il tempo di comunicazione, se N è la lunghezza media della coda, l'attesa media è $N/2 * T_s$ e

$$T_R \text{ (medio)} = (N/2+1) * T_s$$

Soluzioni per limitare l'overhead

limitare la lunghezza della coda (sveltendo il servizio) e
rifiutare le richieste a coda piena (rifiutando servizio)



SERVITORE CONCORRENTE

Servitore che serve più clienti insieme

che serve molte richieste contemporaneamente e può ottimizzare l'uso della risorsa processore eliminando i tempi di idle

$$T_R = T_S + 2 T_C + T_Q + T_I + T_G$$

T_C comunicazione e T_Q accodamento (trascutibili)

T_I tempo di interleaving (può sottrarre tempo)

T_G tempo di generazione del processo (dipende dalla tecnologia)

La **concorrenza** può produrre significative riduzioni del T_R

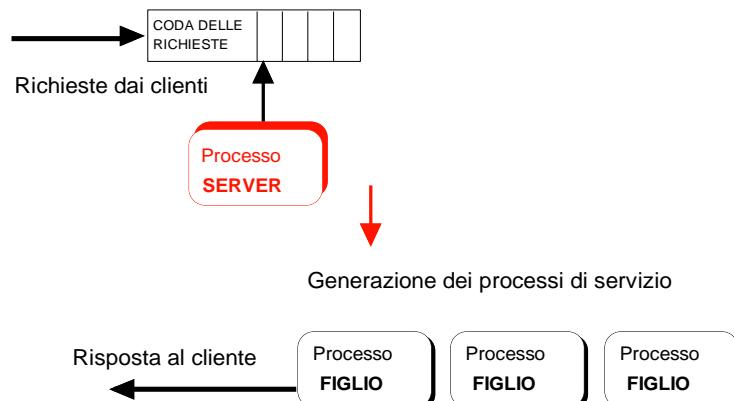
- se la risposta richiede un tempo di attesa significativo di I/O o di sospensione del servizio con possibilità di interleaving
- se le richieste richiedono tempi di elaborazione molto variabili
- se il server è eseguito in un multiprocessore, cioè con servizi in reale parallelismo

SERVITORE CONCORRENTE

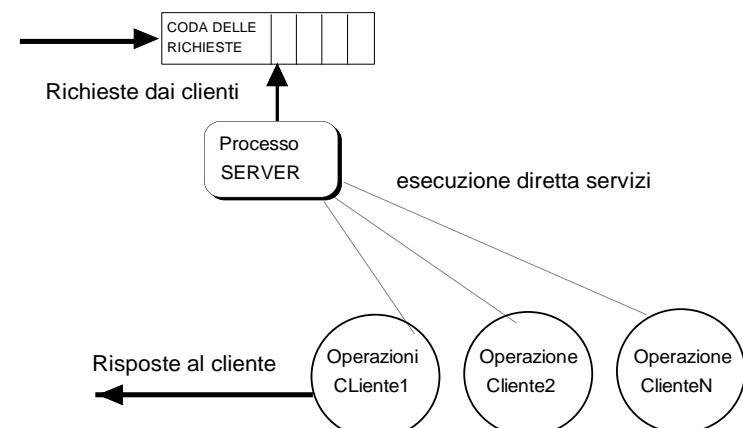
Il progetto di un servitore concorrente può seguire diversi schemi

Servitore concorrente multiprocesso: un processo server si occupa della coda delle richieste e genera processi figli, uno per ogni servizio (T_G)

Servitore concorrente monoprocesso: un unico processo server si divide tra il servizio della coda delle richieste e le operazioni vere e proprie (non c'è il costo T_G)



**servitore concorrente
multiprocesso**



**servitore concorrente
singolo processo**

IL SERVITORE ... (VARIAZIONI)

Nella interazione con un **Server** sequenziale o concorrente
un cliente può non lavorare in modo sincrono bloccante

Se si usano altri protocolli, come nel caso di **timeout**, il cliente dopo un certo tempo abbandona la richiesta e non aspetta risultato

Casi da tenere in conto:

- **Invio di risultato successivo:** dopo il timeout, il supporto assorbe i risultati e li butta via
- **Ripetizione delle richiesta:** il **server deve riconoscere le richieste ripetute dallo stesso cliente**

Il server deve riconoscere ripetizioni della richiesta ⇒ e fare un solo servizio e fornire la stessa risposta ottenuta da una sola esecuzione
Coordinamento facilitato dalla **coda di richieste** e da un **supporto dello stato** delle richieste

Il client deve identificare in modo unico le richieste

Il server dopo avere fatto la operazione e prodotto il risultato deve mantenerlo fino alla consegna richiesta dal cliente specifico

FORME AVANZATE di CLIENTE / SERVITORE

Sono possibili **molte forme** di **interazione C / S**
appena cominciamo ad esaminare sistemi reali

Modello di interazione pull e push

per arrivare ad una interazione flessibile e adatta ai casi che possono servire nei diversi possibili usi

Il **polling** è una sequenza di richieste cliente/servitore (iniziativa cliente)

Il **modello push** comporta un rapporto C/S dal cliente al servitore e un successivo rapporto C/S dal servitore al cliente (poi iniziativa servitore)

Modello di interazione push è

molto usato per ampliare la fascia di utenza. Un utente

- registra una richiesta di interesse per un **feed RSS** (RDF Site Summary – o Rich Site Summary) che si incarica della emissione di notizie ai registrati e
- poi riceve ogni nuova notizia su iniziativa del servitore



MODELLI a DELEGAZIONE

In caso di **interazione sincrona non bloccante**, si possono **delegare le funzionalità di ricezione del risultato ad una entità che opera al posto del responsabile** e lo libera di un compito

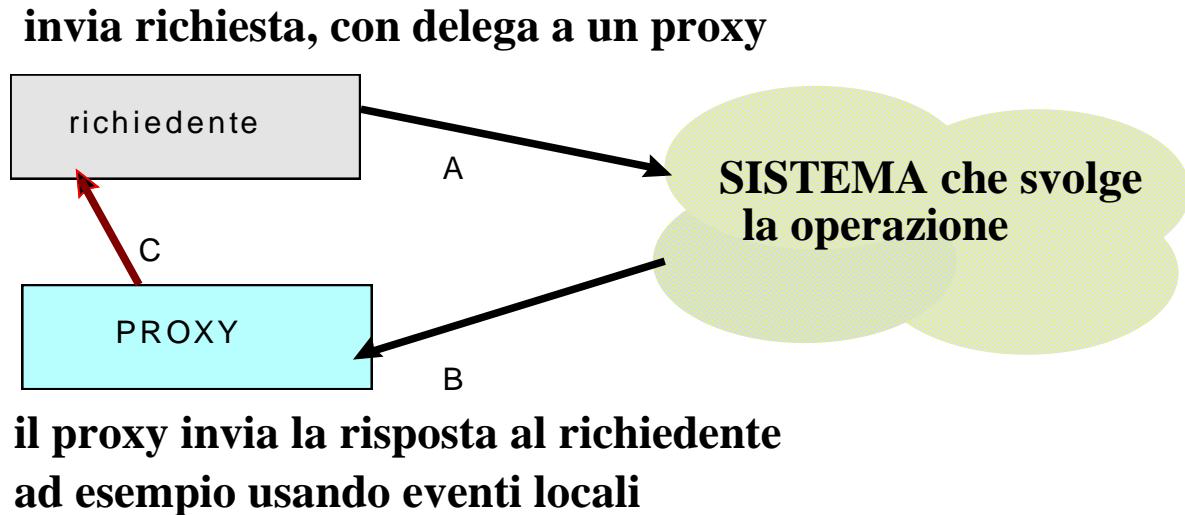
Entità PROXY, DELEGATE, AGENTI, ATTORI
che svolgono una funzione al posto di qualcun altro

Un cliente lascia una altra entità ad aspettare una risposta ad una operazione fatta ad un server lento

Il proxy lavora in modo push per fornire la risposta al cliente stesso

La risposta viene
consegnata o in modo
push o pull

**Si possono avere
molte modalità
diverse di interazione
tra proxy e cliente
iniziale**



Modello Cliente / Servitore

Il modello Client/Server è **intrinsecamente molto utile** perché è **molto adatto** alla **interazione** in un **ambiente distribuito** organizzato su **molti nodi** con **risorse e servizi diversi** da fornire

Le regole di comunicazione del C/S sono molto **chiare** e ad **alto livello**:

Il **C/S** propone un modello **asimmetrico** e **dinamico** dove il **Client** conosce il servitore specifico solo alla chiamata e tipicamente interagisce con il server in modo **sincrono** e **bloccante (a default)**

Il modello C/S implica un **accoppiamento molto stretto tra le entità interagenti** che devono essere **presenti insieme per interagire**

Non c'è modo di ottenere relazione se il Client e il Server non sono simultaneamente presenti nel sistema

Pensiamo a due parti di un'organizzazione che debbano scambiare informazioni di notte, vedi banche che assumono la non compresenza (come e quando?)

L'accoppiamento forte (strong coupling) potrebbe essere troppo vincolante 😞

Modello SCAMBIO DI MESSAGGI

Il modello **Client/Server** presenta un forte accoppiamento: si possono pensare altre strategie accoppiate in modo più lasco (light coupling) e modello asincrono

Lo **scambio di messaggi** ad esempio si presenta come **non accoppiato**

Quando inviamo una e-mail

- non c'è risposta **(asincrono)**
- non si attende in alcun modo,
ma si va avanti per poi recuperare la risposta
 (sincrono non bloccante)

Spesso lo **scambio di messaggi è asincrono e poco sincronizzante**

Molti strumenti e meccanismi supportano questo tipo di interazioni e sono usati per la **loro flessibilità**

ma sono anche molto di **più basso livello e difficili da usare**

Spesso usati a livello di supporto di sistema e non a livello di utente
in quanto troppo primitivi e poco disciplinati

C/S vs. Eventi e Scambio di Messaggi

Eventi, PUB/SUB, e Scambio di messaggi Sender/Receiver

Modelli a scarso (anche minimo) accoppiamento (loose coupling)
non impone la **compresenza delle entità interagenti**

Molto flessibile, primitivo, ed espressivo, ma non facile da usare

Molto low level (e adatto per ogni possibile uso del sistema)

si permettono **molti diversi usi eterogenei di sistema**, anche

Supporto ad ogni possibile tipo di comunicazione, anche
ogni forma di **MX e BX**

Client / Server

Modello a accoppiamento forte (strong coupling)

che implica **compresenza di entrambe le entità interagenti**

Meccanismo molto adatto per comunicazioni di alto livello e semplici

Molto high level (molto adatto per uso applicativo e utente)

ma **non così flessibili** per situazioni diverse e specifiche

Impossibilità di Multicast (MX) and Broadcast (BX)

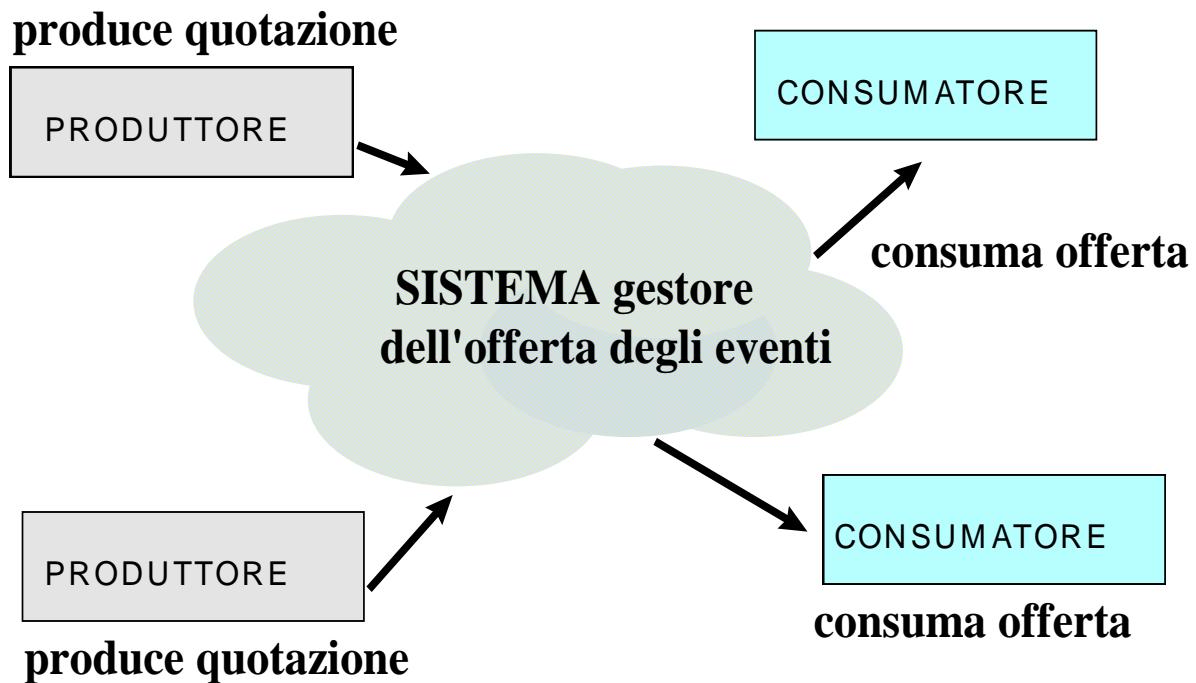
MODELLO ASINCRONI a EVENTI

Modello diverso dal C/S per molti aspetti (**MOLTI A MOLTI**)

Il modello ad eventi è **asincrono e fortemente disaccoppiato**: prevede di avere **molti** produttori, **molti** consumatori, e ad assumere l'esistenza di un **sistema di supporto**

si gestisce l'invio di messaggi disaccoppiando gli interessati

- I **produttori** segnalano
- I **consumatori** ricevono
dopo sottoscrizione
(modello pub/sub)
- Il **gestore degli eventi**
segna l'occorrenza
degli eventi e i
relativi messaggi
agli interessati
sottoscrittori
(push dei messaggi)



IMPLEMENTAZIONE: CONNESSIONE

Nella **interazione C / S**, si considerano due tipi principali riguardo all'insieme delle richieste

- **interazione connection-oriented (con connessione)**
si stabilisce un **canale di comunicazione virtuale** prima di iniziare lo scambio dei dati (es. connessione telefonica)
- **interazione connection-less (senza connessione)**
senza connessione virtuale, ma semplice scambio di messaggi isolati tra loro (es. il sistema postale)

La scelta tra le due forme dipende dal **tipo di applicazione** e anche da vincoli imposti dal **livello di comunicazione** sottostante

Per esempio, in **Internet** il **livello di trasporto** prevede i protocolli TCP e UDP basati su IP (tipicamente *connectionless* e *best effort*)

- **TCP con connessione**, reliable (affidabile) e preserva l'ordine di invio dei messaggi e a maggiore affidabilità
- **UDP senza connessione**, non reliable e non preserva ordine messaggi

Ma al di sotto c'è sempre IP

INTERCONNESSIONE FISICA

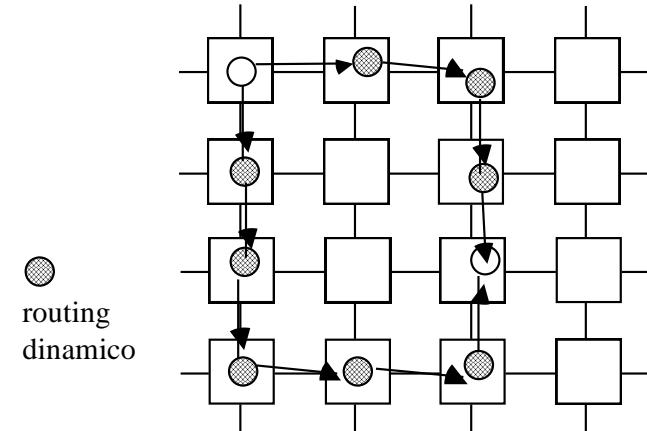
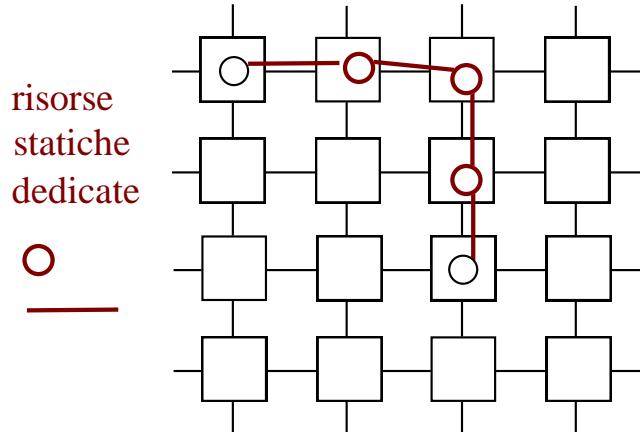
Nella **interazione C / S**, è anche importante come si trasferiscono i messaggi parte della sequenza di comunicazione

- **connessione (OSI)**

Tutti i messaggi seguono la **stessa strada (route)** per la coppia mittente destinatario **decise staticamente e impegnano risorse nei nodi interessati e nei nodi intermedi predeterminati**

- **senza connessione (IP)**

I messaggi possono seguire strada diverse **decise dinamicamente e non impegnano staticamente risorse intermedie**



Alcuni modelli a **connessione** (come **TCP basato su IP**), non impegnano risorse intermedie ma solo sul mittente / destinatario

IMPLEMENTAZIONE: VISIBILITÀ

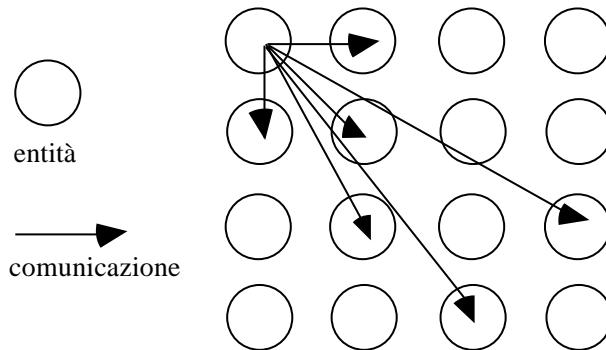
Nella **comunicazione**, è anche importante se si possa essere in visibilità di tutti i potenziali partecipanti (**scalabilità**)

concetto di località (limiti alla comunicazione) vs. globalità (nessun vincolo)

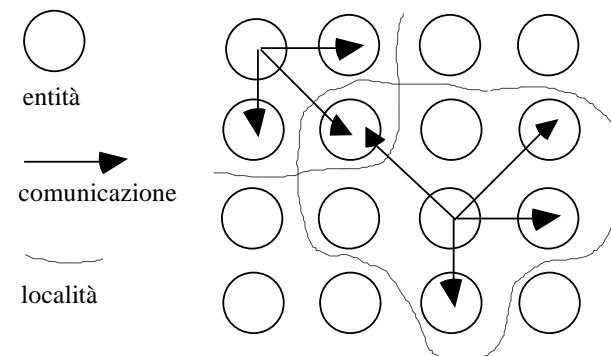
modelli globali non impongono restrizioni alle interazioni \Rightarrow operazioni non scalabili dipendenti dal diametro del sistema

modelli locali (o ristretti) prevedono limiti alla interazione ⇒ operazioni (forse) scalabili poco dipendenti dal diametro del sistema

Modelli globali



Modelli locali



Si va verso la località (con vincoli) per ottenere scalabilità

STATO nel CLIENTE / SERVITORE

Nella **interazione C / S**, un aspetto centrale è lo **stato della interazione** ossia che si tenga traccia della comunicazione e delle azioni precedenti (**o meglio ne tenga traccia il servitore**)

In questo caso di stato, la interazione è facilitata essendo **riconosciuta e gestita in termini di protocollo**

Possiamo quindi avere: **relazioni C/S con stato o meno (sul servitore)**

- | | |
|------------------|--|
| stateless | non si tiene traccia dello stato: ogni messaggio è completamente indipendente dagli altri e auto contenuto |
| stateful | si mantiene lo stato dell'interazione tra chi interagisce: un messaggio (e operazione conseguente) può dipendere da quelli precedenti |

In genere, il cliente richiedente tende a livello applicativo a tenere traccia dello stato, in caso di azioni ripetute

STATO nel CLIENTE / SERVITORE

Uso dello stato

Lo stato della interazione consente di avere un accordo ‘predefinito’ tra le entità interagenti e di avere una gestione condivisa che facilita la comunicazione

Quindi il cliente può avere un protocollo semplificato, assumendo che il servitore (lo stesso) lo facili mantendo lo stato

In caso di ripetizione di **10 richieste** negoziate e predeterminate e sempre richieste in ordine (lettura di un elemento di un array di 10 elementi), ogni richiesta non deve specificare l'indice da chiedere, ma solo la nuova esigenza del dato successivo

Lo stato delle interazione memorizza quante azioni sono occorse (vedi lettura sequenziale da un file)

CLIENTE / SERVITORE RIPETUTO

Lo stato dell'interazione è fortemente significativo nel caso di interazione ripetuta e di un insieme di comunicazioni tra gli stessi attori

Lo stato è quindi una sintesi memorizzata da una delle parti di come la comunicazione sta andando avanti

Lo stato dell'interazione memorizzato nel Server (che diventa **stateful**)

Un Server stateful garantisce efficienza (le dimensioni dei messaggi sono più contenute e otteniamo una migliore velocità di risposta del Server)

Un Server stateless è più leggero e più affidabile in presenza di malfunzionamenti (soprattutto causati dalla rete) ma lo **stato deve essere mantenuto da ogni cliente**

In caso di server stateless, **lo stato della interazione deve essere mantenuta dal singolo cliente**

STATO nel SERVITORE

In caso di **server stateful**, il Server deve potere identificare il Client e tenerne **traccia per interazioni future**

Ci sono notevoli differenze tra un file server **stateful** e stateless
stateful API più semplici

```
key=open(filename, intentions);  
rc=read(key, buffer, howmany);  
rc=write(key, buffer, howmany);
```

La ripetizione di operazione fa avanzare l'I/O pointer

stateless API più complete e complesse

```
rc=read(filename, from, buffer, howmany);  
rc=write(filename, from, buffer, howmany);
```

Ogni operazione è autocontenuta e specifica tutto, in questo caso il primo byte oggetto dell'operazione a partire dall'inizio del file (**from**)
(NOTA: il file system NFS di SUN è stateless)

Le operazioni devono essere **controllate dal cliente** che mantiene lo **stato di ogni file** a cui accedere e **la storia della interazione**

SERVITORE STATEFUL

Un **server stateful** deve sicuramente prevedere e gestire un **impegno di risorse ulteriore**

il Server deve identificare la **sessione del Client** e tenerne **traccia per interazioni future**: ad esempio può più facilmente riconoscere ed autorizzare utenti e operazioni

L'impegno di risorse può anche durare **molto tempo** e potrebbe **crescere pericolosamente con l'accumulo di molte (troppe) altre richieste**

```
key=open(filename, intentions);
```

L'impegno inizia alla open e registra e attiva la sessione

```
ok=close(key);
```

Fino ad un chiusura che potrebbe non arrivare

Il vantaggio è di avere un **minore costo delle operazioni in termini di banda impegnata, sicurezza, e garanzie di ripristino**, in caso di problemi dei clienti

Ma costi superiori al servitore

STATO INTERAZIONE del SERVITORE

I modelli **stateless** portano a un progetto del **cliente più complesso**, ma semplificano il **progetto del server** che non deve mantenere stato

I modelli di **interazione stateful** tendono a richiedere **al server di mantenere lo stato della interazione**

Si pensi allo stato mantenuto sul server per ogni file aperto

La scelta tra server **stateless** o **stateful** deriva dall'applicazione e dai protocolli di interazione

Un'interazione **stateless** è sensata e possibile (viable) SOLO se il protocollo è progettato per operazioni **idempotenti**

IDEMPOTENZA

In caso di progetti stateless, il protocollo è corretto e il progetto del servitore è molto semplificato se le operazioni sono idempotenti

il cliente può invocare ripetutamente le operazioni anche molte volte ed ottenere sempre lo stesso effetto con le sue eventuali ripetizioni della stessa richiesta

Le operazioni idempotenti tendono a produrre lo stesso risultato, anche se ripetute più volte

Ogni richiesta potrebbe non arrivare, o arrivare fuori ordine o arrivare ripetuta, ma essendo **idempotente** non si hanno problemi

Per esempio, un Server fornisce sempre la stessa risposta ad un messaggio M indipendentemente da altri messaggi (in particolare, lo stesso M) ricevuti dal Server stesso

(ovviamente, a meno che lo stato delle risorse corrispondenti non sia variato)

rc=read(filename, from, buffer, howmany);

STATO della INTERAZIONE

I **modelli con stato** hanno il **server** che deve mantenere traccia della interazione

Per quanto tempo e con che costi?

Ovviamente si deve ridurre il costo

Si distingue lo stato in base alla durata massima:

- **Stato permanente** mantenuto per sempre
- **Stato soft o a tempo** che rimane per un tempo massimo

Si pensi ad un server web che deve **mantenere le risorse** per reggere tutte le richieste dei clienti che hanno acceduto (sessioni in atto)

Si pensi ad un server che deve riconoscere tutti i clienti che sono **autorizzati** ad accedere (username e password) tramite tabelle di riconoscimento

STATO della INTERAZIONE

I modelli con **stato (sul server)** hanno un costo in **risorse richieste** ed una **complessità di progetto del server superiore** rispetto ai modelli **senza stato**

Il server deve farsi carico di mantenere tutto lo stato, specie in casi critici

In caso **stateless**, la **complessità del server è limitata e viene ripartita su ogni singolo cliente che deve tenere traccia della interazione** e deve specificare anche tutte le informazioni relative allo stato stesso in ogni invocazione

`rc=read(filename, from, buffer, howmany);`

Ogni operazione deve prevedere tutte le informazioni

PROGETTO del SERVITORE

Una proprietà che caratterizza il **server** è la possibile **concorrenza** delle azioni, cioè la *possibilità di portare avanti più operazioni*

Il **server** è tipicamente una sola attività e un solo **processo**, ma la concorrenza può migliorare le prestazioni

si pensi ad un **server Web** che deve rispondere a moltissime richieste contemporaneamente

Un **Server iterativo o sequenziale** processa le richieste di servizio una alla volta, e mette in **coda** di attesa le altre

possibile basso utilizzo delle risorse, in quanto non c'è sovrapposizione tra elaborazione ed I/O

Un **Server concorrente** può gestire **molte richieste di servizio** insieme (in modo concorrente anche se non in parallelo), cioè accettare una richiesta prima del termine di quella in corso di servizio
migliori prestazioni ottenute da sovrapposizione elaborazione ed I/O ma maggiore complessità progettuale

PROGETTO SERVITORE

Possiamo avere molti diversi schemi di cooperazione per servizio che possiamo classificare in base a **diverse proprietà**

SERVIZIO

**sequenziale/iterativo
concorrente**

**con stato interazione senza
stato sul server**

**con connessione
senza connessione**

S E R V E R	Tipo di comunicazione	
	connessione	senza connessione
sequenziale iterativo		
concorrente singolo processo		
concorrente multi processo		

La scelta del tipo di Server dipende dalle caratteristiche del servizio da fornire

Lo stato ha effetto sul protocollo e sulle entità

MODELLI di SERVIZIO

Servitore sequenziale o iterativo

si possono introdurre ritardi se la coda di richieste cresce

Servitore concorrente

capacità di servire richieste insieme sfruttando tempi morti

Servitore senza stato sul server

il servitore dimentica le richieste appena le ha eseguite

Servitore con stato interazione

il servitore deve tenere traccia della interazione con i clienti

Servitore senza connessione

ogni richiesta arriva in modo indipendente (fuori ordine)

Servitore con connessione

le richieste arrivano in ordine di emissione del cliente

PROPRIETÀ del SERVITORE

Nel progetto di una interazione, scegliamo in base a caratteristiche tecnologiche, **ad esempio il sistema operativo** di supporto, e anche del **protocollo** che vogliamo realizzare e dei **vincoli di costo**

In un ambiente Unix, la generazione di un **processo pesante** (fork) è facile e semplificata per la condivisione, si possono utilizzare **servitori multiprocesso per avere la concorrenza**
Un **server Web** per **Unix** tende a generare un processo per ogni richiesta di servizio e deve farlo in modo efficiente

In un ambiente Java, la generazione di un **thread** (processo leggero) è facile e semplificata per la condivisione, si possono utilizzare **servitori multiprocesso per avere la concorrenza**
Un **server Web** per **ambienti Java** tende a generare un thread per ogni richiesta di servizio e lo fa in modo efficiente visto il basso costo del supporto dei thread (pure a livello applicativo)

PROCESSI e OGGETTI

Processi e oggetti ⇒ vanno d'accordo?

sono entità spesso separabili e ortogonali (a volte)

Esistono entrambi durante la esecuzione di una applicazione?

Considerare i modelli di esecuzione noti

UNIX

i **processi** sono attività esistenti durante la esecuzione (barriera di visibilità) e che possono accedere ad oggetti privati solamente

Java

gli **oggetti** sono presenti come risorse di memoria da riferire per il loro tempo di vita (come le classi)

i **processi (thread)** sono attività ed oggetti con vincoli specifici e capaci di eseguire sugli oggetti caricati in memoria

PROGETTO di C/S - CLIENTE

Ogni **processo che si attiva** (cliente e servitore) deve controllare, usare e consumare opportunamente i propri **argomenti di ingresso**, comunque sia stato progettato e programmato

Progetto dei Clienti ⇒ più semplice

Spesso sono sequenziali, potrebbero essere anche concorrenti o paralleli, a seconda delle architetture

- 1) I processi **clienti** sono sempre **filtri**, ossia dei **processi legati ad uno stream di input e che devono consumare tutto il contenuto di dati** che arriva dall'input stesso
- 2) In genere, anche il **server** impegnato in una relazione di servizio deve consumare tutta la **conversazione con ogni singolo cliente** e non lasciare le cose a metà
- 3) Rispetto del **protocollo negoziato** tra i due partecipanti

PROGETTO di C/S - SERVITORE

Progetto dei Servitori ⇒ più complesso per le operazione svolte dal server stesso a fronte delle richieste dei molteplici clienti (**molti a 1**)

Ma anche... ogni server deve essere **già presente** al momento delle richieste dei clienti ⇒ deve essere **sempre presente con un ciclo di vita infinito**

La garanzia attraverso server come processi **eterni**, **processi demoni** sulle macchine server e che sopravvivono alla durata delle singole applicazioni per vivere per tutta il ciclo di vita del sistema

I daemon sono tipici processi server che eseguono un **ciclo infinito di attesa** di richieste ed esecuzione delle stesse a volte **sequenziali**, ma spesso **concorrenti**

Java riconosce **thread daemon** rispetto a thread user e li esegue per tutta la durata di una sessione della virtual machine (JVM), terminandone l'esecuzione solo quando termina l'ultimo user thread

MODELLO di COMUNICAZIONE in C/S

Schema base C/S: asimmetrico, sincrono, bloccante, dinamico

I servizi forniti da un servitore su iniziativa del cliente che risponde a diversi clienti (**molti:1**)

I clienti conoscono il servitore e non sono noti a priori al servitore

MODELLO interazione sincrona (default)

ma anche **asincrona / non bloccante**

asincrona ⇒ nessun (interesse per il) risultato

non bloccante ⇒ **sincrona**, ma non interessa aspettare risultato

Molte variazioni sul MODELLO oltre il default

anche a time out, con push da parte del server

e anche considerando eventualmente più server possibili per portare a termine il servizio in modo da avere più possibilità

a default ⇒ interessa avere un'unica operazione e unico servizio

SERVIZIO vs PRODOTTO

PRODOTTO vs. SERVIZIO

Lamette e rasoio

GRATIS RASOIO
abbonamento al servizio lamette

Moka e caffè

GRATIS MACCHINETTA
Abbonamento cialde

La logica dei servizi rende necessario ragionare sui contratti di servizio e la qualità del servizio stesso

Servizio

Qualità del Servizio o QoS come aderenza ad un contratto implicito e esplicito

Che proprietà deve avere?

Social Network



PINTEREST

SOCIAL SITE
THAT IS ALL ABOUT
DISCOVERY

LARGEST
OPPORTUNITIES



USERS ARE:

17%
MALE
83%
FEMALE

20
MILLION
ACTIVE USERS



TWITTER

MICRO BLOGGING
SOCIAL SITE
THAT LIMITS EACH
POST TO **140**
CHARACTERS

LARGEST
PENETRATION
in the US

BUT SPREADING
SLOWLY AND STEADILY

5,700 TWEETS
HAPPEN
EVERY
SECOND

241
MILLION
ACTIVE USERS



FACEBOOK

SOCIAL SHARING
SITE THAT HAS
1+ BILLION
USERS WORLDWIDE

LARGEST
OPPORTUNITIES



COMMUNICATING WITH
CONSUMERS
IN A NON-OBTRUSIVE WAY

USERS SHARE
1 MILLION LINKS
EVERY 20 MINUTES



1+
BILLION
ACTIVE USERS



INSTAGRAM

SOCIAL SHARING
SITE ALL AROUND
PICTURES
AND NOW 15 SECOND
VIDEOS

MANY BRANDS
ARE PARTICIPATING
THROUGH THE USE OF

HASHTAGS
AND POSTING

PICTURES
CONSUMERS
CAN RELATE TO

MOST FOLLOWED
BRAND IS
NATIONAL
GEOGRAPHIC

200
MILLION
ACTIVE USERS



GOOGLE+

SOCIAL NETWORK
BUILT BY GOOGLE
THAT ALLOWS FOR
BRANDS
AND **USERS**
TO BUILD CIRCLES

NOT AS MANY
BRANDS ACTIVE,
BUT THE ONES THAT ARE
TEND TO BE A
GOOD FIT WITH A
GREAT FOLLOWING

25-35
YEAR
OLDS
ARE THE MOST
ACTIVE

540
MILLION
ACTIVE USERS



LINKEDIN

**BUSINESS
ORIENTED**
SOCIAL NETWORKING SITE

BRANDS THAT ARE
PARTICIPATING
ARE CORPORATE
BRANDS
GIVING POTENTIAL AND
CURRENT ASSOCIATES
A PLACE TO NETWORK
& CONNECT



POWERS
50% OF THE
WORLD'S HIRES

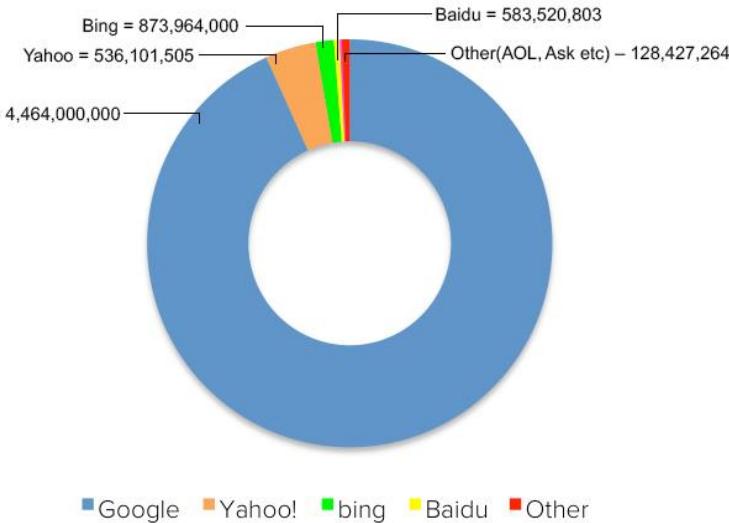
300
MILLION
USERS

Uso di Sistema

Uso di applicazioni motori di ricerca

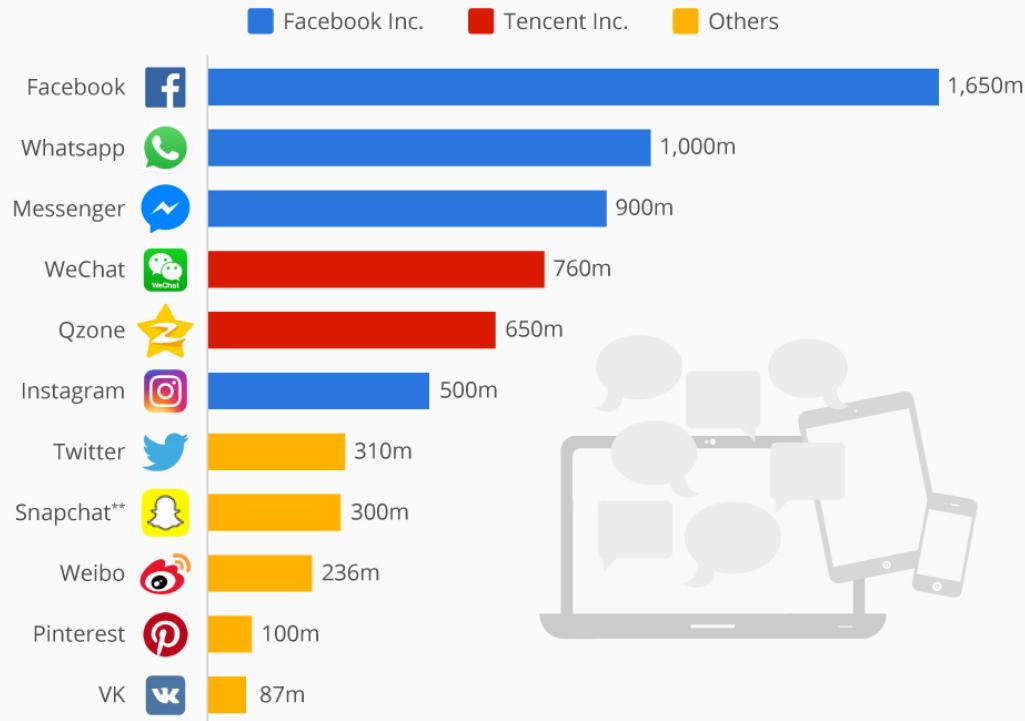
social media

Search Engine Statistics



Facebook Inc. Dominates the Social Media Landscape

Monthly active users of selected social networks and messaging services*



Today's Digital Marketing Ecosystem



NUOVI SCENARI

Uso di tool on line

JUL
2020

SOCIAL MEDIA USE AROUND THE WORLD

THE NUMBER OF PEOPLE WHO ACTIVELY USE SOCIAL NETWORKS AND MESSENGER SERVICES

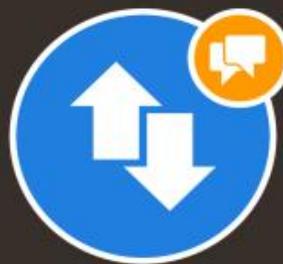
TOTAL NUMBER OF
ACTIVE SOCIAL
MEDIA USERS



SOCIAL MEDIA
PENETRATION (USERS
vs. TOTAL POPULATION *)



ANNUAL GROWTH IN
THE TOTAL NUMBER OF
SOCIAL MEDIA USERS



TOTAL NUMBER OF SOCIAL
MEDIA USERS ACCESSING
VIA MOBILE PHONES



PERCENTAGE OF TOTAL
SOCIAL MEDIA USERS
ACCESSING VIA MOBILE



3.96
BILLION

51 %

+10.5%
+376 MILLION

3.91
BILLION

99%

SOURCES: KEPiOS ANALYSIS; SOCIAL MEDIA COMPANY STATEMENTS AND EARNINGS ANNOUNCEMENTS; SOCIAL MEDIA PLATFORMS' SELF-SERVICE ADVERTISING TOOLS; CNNIC, MEDIASCOPE; CAFEBAZAAR (ALL LATEST DATA AVAILABLE IN JULY 2020). ***NOTES:** PENETRATION FIGURES ARE FOR TOTAL POPULATION, REGARDLESS OF AGE.

◆ **COMPARABILITY ADVISORY:** SOURCE AND BASE CHANGES.

NUOVI SCENARI POST COVID

Incremento di tool on line

JAN
2021

SOCIAL MEDIA USE AROUND THE WORLD

USE OF SOCIAL NETWORKS AND MESSENGER SERVICES, WITH DETAIL FOR MOBILE SOCIAL MEDIA USE

⚠ SOCIAL MEDIA USER NUMBERS MAY NOT REPRESENT UNIQUE INDIVIDUALS

TOTAL NUMBER OF
ACTIVE SOCIAL
MEDIA USERS*



4.20
BILLION

SOCIAL MEDIA USERS AS
A PERCENTAGE OF THE
GLOBAL POPULATION



53.6%

ANNUAL CHANGE IN
THE NUMBER OF GLOBAL
SOCIAL MEDIA USERS



+13.2%
+490 MILLION

TOTAL NUMBER OF SOCIAL
MEDIA USERS ACCESSING
VIA MOBILE PHONES



4.15
BILLION

PERCENTAGE OF TOTAL
SOCIAL MEDIA USERS
ACCESSING VIA MOBILE



98.8%

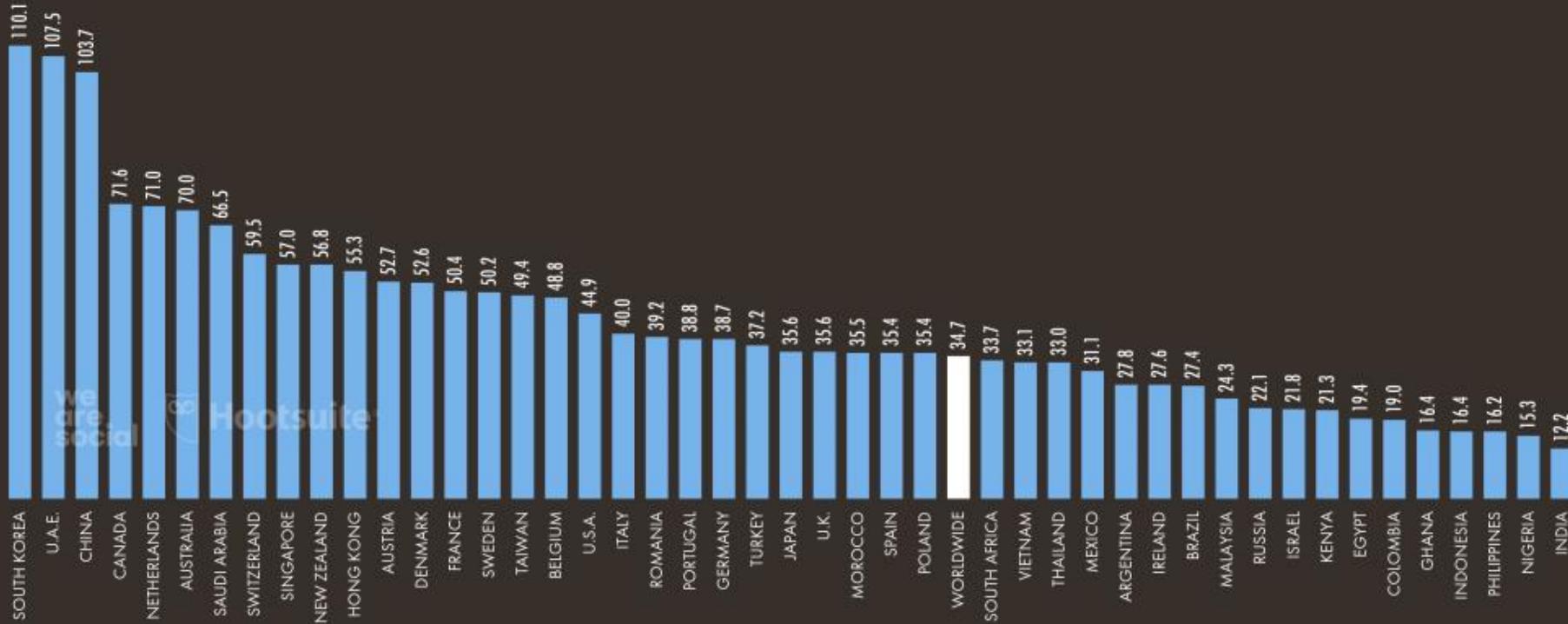
NUOVI SCENARI POST COVID

Incremento nell'uso di Internet

JUL
2020

MOBILE INTERNET CONNECTION SPEEDS

AVERAGE MOBILE INTERNET CONNECTION SPEED, IN MBPS



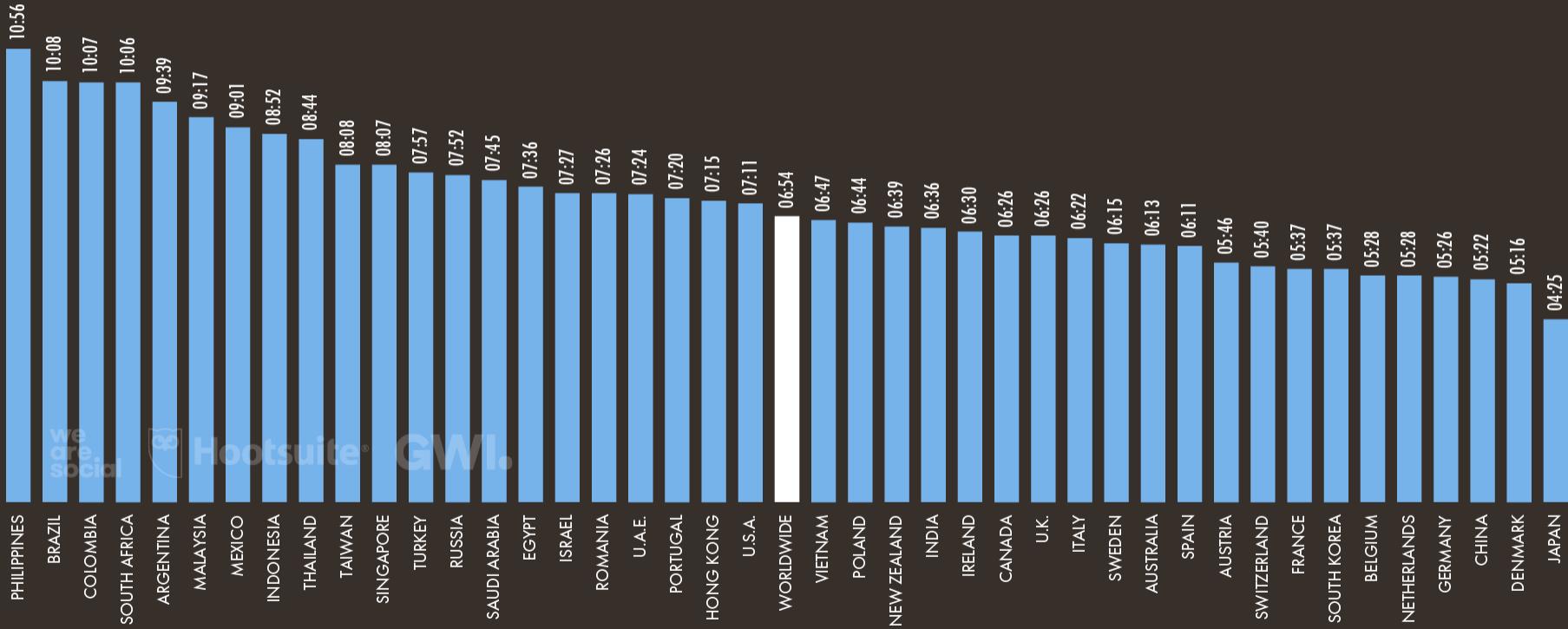
NUOVI SCENARI POST COVID

Incremento nell'uso di Internet

JAN
2021

DAILY TIME SPENT USING THE INTERNET

AVERAGE AMOUNT OF TIME (IN HOURS AND MINUTES) THAT INTERNET USERS AGED 16 TO 64 SPEND USING THE INTERNET EACH DAY ON ANY DEVICE



SOURCE: GWI (Q3 2020). FIGURES REPRESENT THE FINDINGS OF A BROAD GLOBAL SURVEY OF INTERNET USERS AGED 16 TO 64. SEE GLOBALWEBINDEX.COM FOR MORE DETAILS.

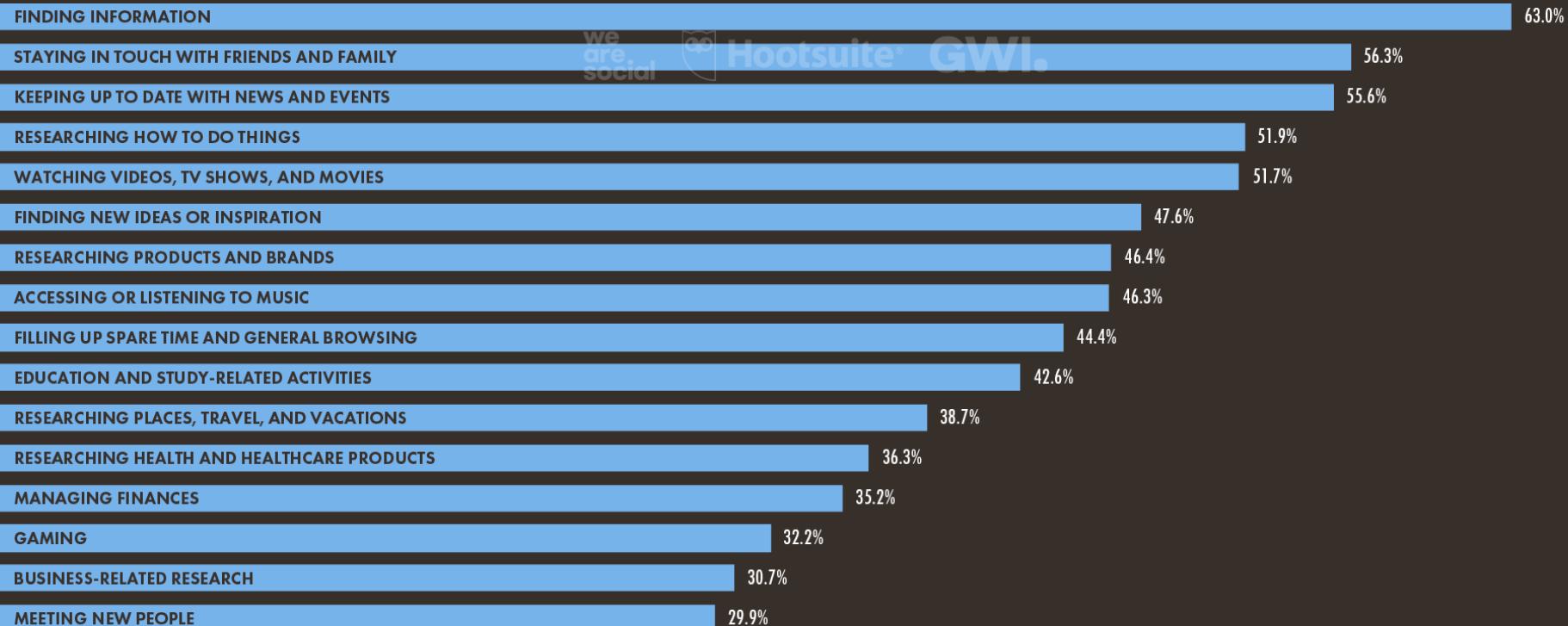
NUOVI SCENARI

Incremento nell'uso di Internet

JAN
2021

REASONS FOR USING THE INTERNET

PRIMARY REASONS WHY GLOBAL INTERNET USERS AGED 16 TO 64 USE THE INTERNET



SOURCE: GWI (Q3 2020). FIGURES REPRESENT THE FINDINGS OF A BROAD GLOBAL SURVEY OF INTERNET USERS AGED 16 TO 64. SEE GLOBALWEBINDEX.COM FOR MORE DETAILS.

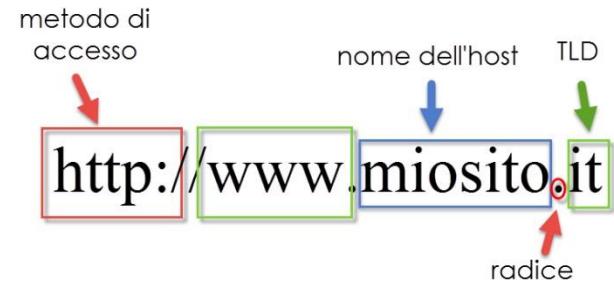
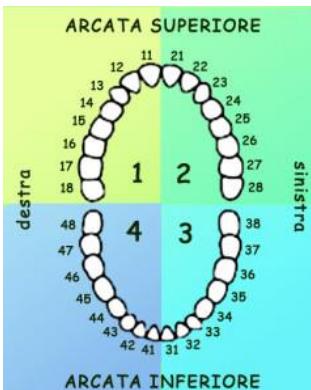
MODELLI e SISTEMI DI NOMI

**Per accedere a servizi è necessario poterli identificare e trovare
(e fare binding alle risorse disponibili)**

Questa funzione viene svolta dai sistemi di nomi, ossia servitori tipicamente capaci di fornire servizi di mantenimento e di gestione dei nomi



TAVOLA PERIODICA DEGLI ELEMENTI																			
http://www.periodictable.com/																			
GRUPPO	MASSA ATOMICA RELATIVA (1)																		
PERIODO	H	D	D ₂	He	Li	B	C	N	O	F	Ne	Na	Mg	Al	Si	P	S	Cl	
1	1.0079	2.0158	3.0104	4.0126	6.941	10.81	12.011	14.012	16.000	19.00	20.18	22.990	24.31	26.982	28.086	30.973	31.996		
2	3.0144	4.0126	5.0104	6.941	Li	Be	Cs	Fr	Ar	Ne	Ar	Ca	K	Rb	Fr	Ar	Ar	Ar	
3	7.0160	8.0126	9.0104	10.941	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	
4	11.990	12.990	13.990	14.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ga	Ge	As	Se
5	15.999	16.999	17.999	18.999	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Du	Ge	As	Se
6	19.998	20.998	21.998	22.998	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
7	23.997	24.997	25.997	26.997	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Du	Ge	As	Se
8	27.996	28.996	29.996	30.996	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
9	31.995	32.995	33.995	34.995	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
10	35.994	36.994	37.994	38.994	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
11	39.993	40.993	41.993	42.993	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
12	43.992	44.992	45.992	46.992	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
13	47.991	48.991	49.991	50.991	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
14	51.990	52.990	53.990	54.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
15	55.990	56.990	57.990	58.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
16	59.990	60.990	61.990	62.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
17	63.990	64.990	65.990	66.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
18	67.990	68.990	69.990	70.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
19	72.990	73.990	74.990	75.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
20	76.990	77.990	78.990	79.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
21	80.990	81.990	82.990	83.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
22	84.990	85.990	86.990	87.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
23	88.990	89.990	90.990	91.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
24	92.990	93.990	94.990	95.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
25	96.990	97.990	98.990	99.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
26	100.990	101.990	102.990	103.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
27	104.990	105.990	106.990	107.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
28	108.990	109.990	110.990	111.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
29	112.990	113.990	114.990	115.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
30	116.990	117.990	118.990	119.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
31	121.990	122.990	123.990	124.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
32	126.990	127.990	128.990	129.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
33	130.990	131.990	132.990	133.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
34	134.990	135.990	136.990	137.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
35	138.990	139.990	140.990	141.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
36	142.990	143.990	144.990	145.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
37	146.990	147.990	148.990	149.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
38	150.990	151.990	152.990	153.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
39	154.990	155.990	156.990	157.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
40	158.990	159.990	160.990	161.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
41	162.990	163.990	164.990	165.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
42	166.990	167.990	168.990	169.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
43	170.990	171.990	172.990	173.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
44	174.990	175.990	176.990	177.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
45	178.990	179.990	180.990	181.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
46	182.990	183.990	184.990	185.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
47	186.990	187.990	188.990	189.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
48	190.990	191.990	192.990	193.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
49	194.990	195.990	196.990	197.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
50	198.990	199.990	200.990	201.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
51	202.990	203.990	204.990	205.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
52	206.990	207.990	208.990	209.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
53	210.990	211.990	212.990	213.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
54	214.990	215.990	216.990	217.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
55	218.990	219.990	220.990	221.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
56	222.990	223.990	224.990	225.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
57	226.990	227.990	228.990	229.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
58	230.990	231.990	232.990	233.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
59	234.990	235.990	236.990	237.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
60	238.990	239.990	240.990	241.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
61	242.990	243.990	244.990	245.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
62	246.990	247.990	248.990	249.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
63	250.990	251.990	252.990	253.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
64	254.990	255.990	256.990	257.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
65	258.990	259.990	260.990	261.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
66	262.990	263.990	264.990	265.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
67	266.990	267.990	268.990	269.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
68	270.990	271.990	272.990	273.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
69	274.990	275.990	276.990	277.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
70	278.990	279.990	280.990	281.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
71	282.990	283.990	284.990	285.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
72	286.990	287.990	288.990	289.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
73	290.990	291.990	292.990	293.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
74	294.990	295.990	296.990	297.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
75	298.990	299.990	300.990	301.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
76	302.990	303.990	304.990	305.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
77	306.990	307.990	308.990	309.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
78	310.990	311.990	312.990	313.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
79	314.990	315.990	316.990	317.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
80	318.990	319.990	320.990	321.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge	As	Se	Br
81	322.990	323.990	324.990	325.990	Li	Be	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Zn	Ge</			

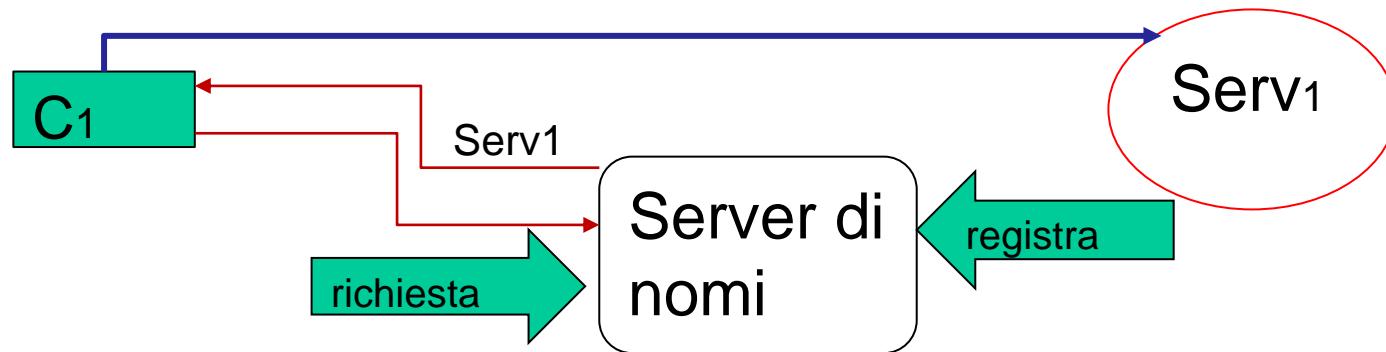


Servizi e Server di NOMI

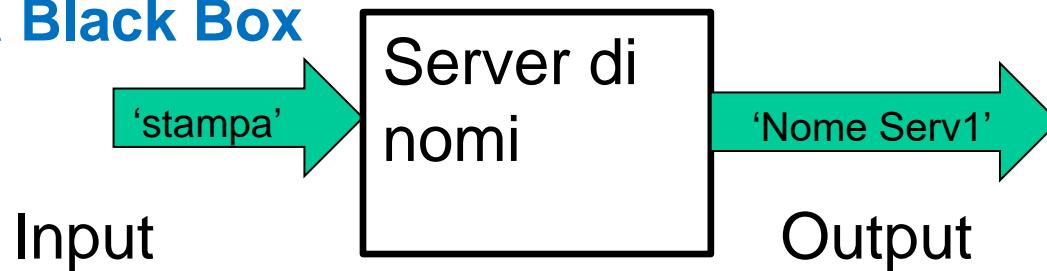
Nei sistemi distribuiti, i sistemi di nomi possono consentire di richiedere e ottenere il **binding** ad un servitore per poterlo riferire. In generale, ad un servizio di nomi clienti possono fare **richieste** per ottenere il nome del servitore di interesse e poterlo poi riferire.

Esempio: stampa

nome del servitore di stampa



Approccio a Black Box



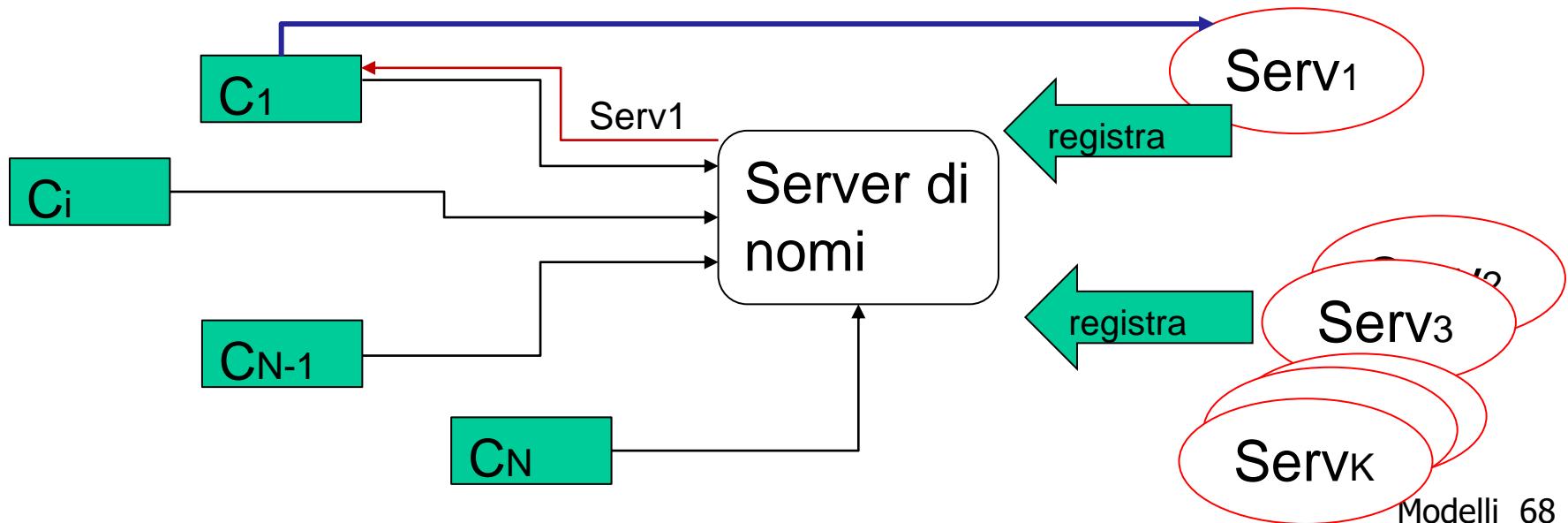
MODELLI DI NOMI

I sistemi di nomi sono il primo supporto architetturale

i nomi, ossia i riferimenti ad altre entità, sono distribuiti nel codice dei clienti, degli utilizzatori, delle librerie, ecc., e se ne deve garantire la consistenza

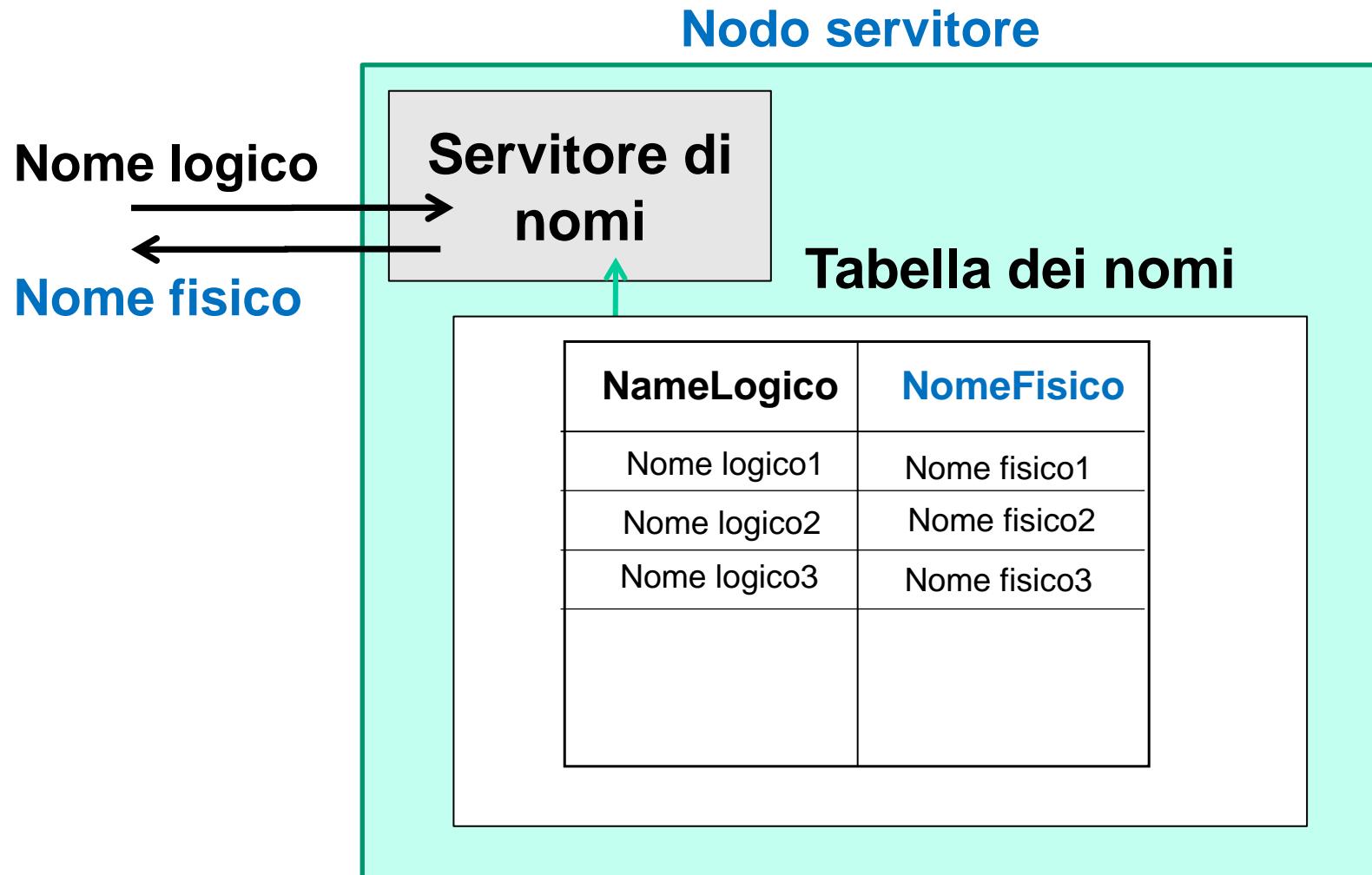
Un sistema di nomi gestisce ed accetta questa funzione di binding ossia di risoluzione del nome di una entità, specie se remota

Supporto implementato attraverso un servizio di risoluzione dei nomi o servizio di nomi a cui i server si registrano



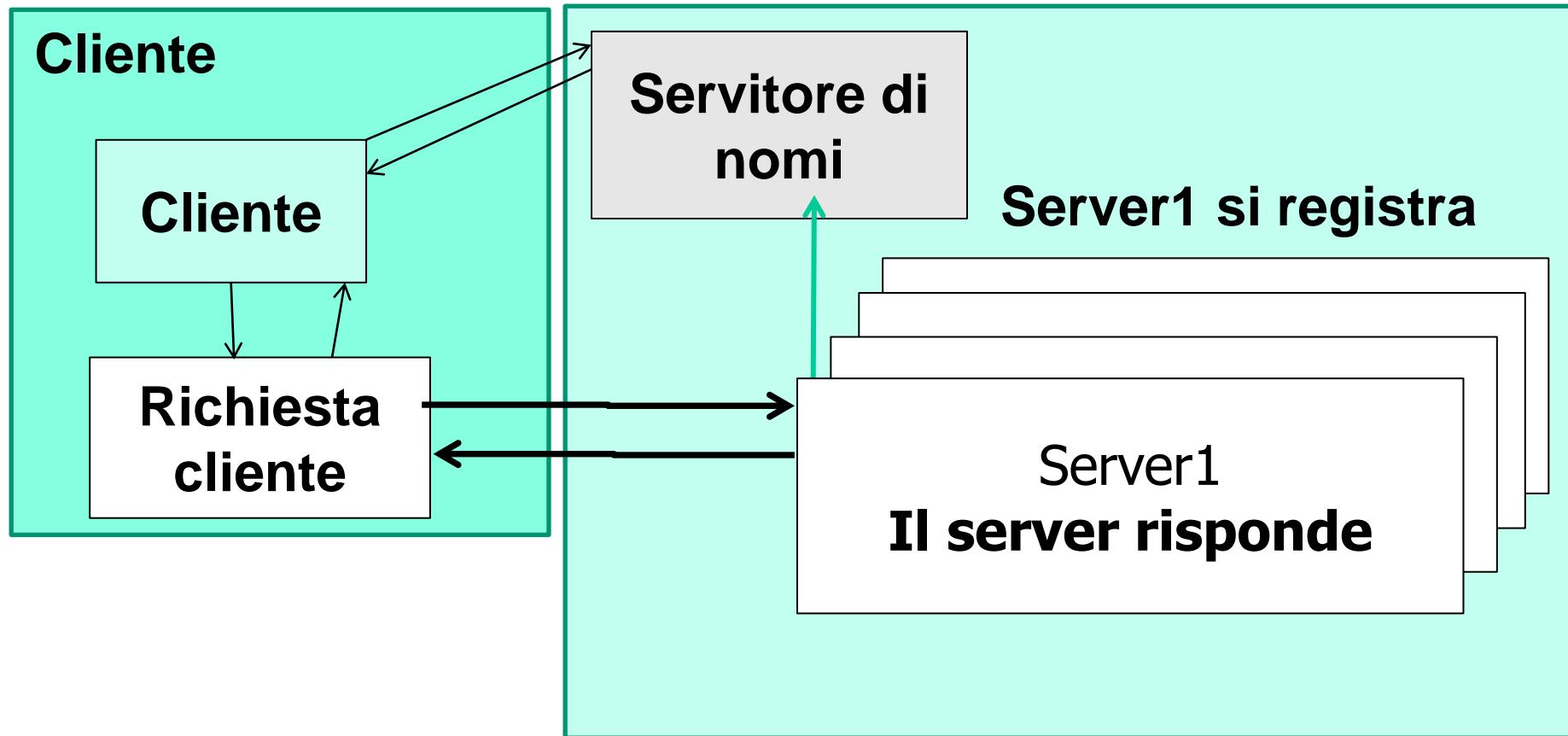
Server di nomi su unico nodo

Un server di nomi deve potere fare passare da un nome ad un altro, su richiesta avendo una tabella di corrispondenze



Server di nomi su unico nodo

Un unico server di nomi può registrare tutti i server di un nodo



Unico Nodo Servitore

MODELLO ad AGENTI MULTIPLI

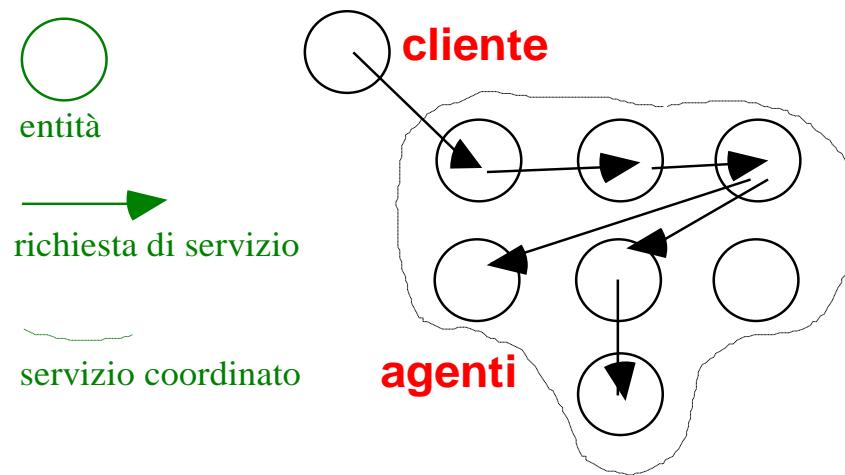
Schema in cui i servizi sono forniti dal coordinamento di più servitori detti **agenti** che forniscono un servizio globale unico (**modello ad almeno due tier - o livelli**)

Gli **agenti** forniscono il servizio coordinato e possono:

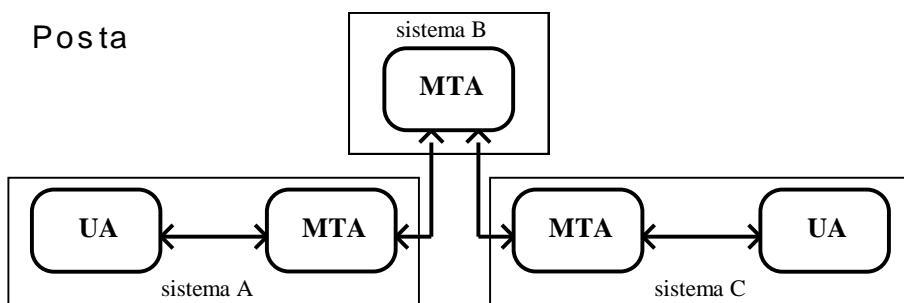
- **partizionare le capacità di servizio**
- **replicare le funzionalità di servizio**

in modo **trasparente al cliente**

cliente ed **agenti**



agenti di posta (mail)



ARCHITETTURE A PIÙ LIVELLI

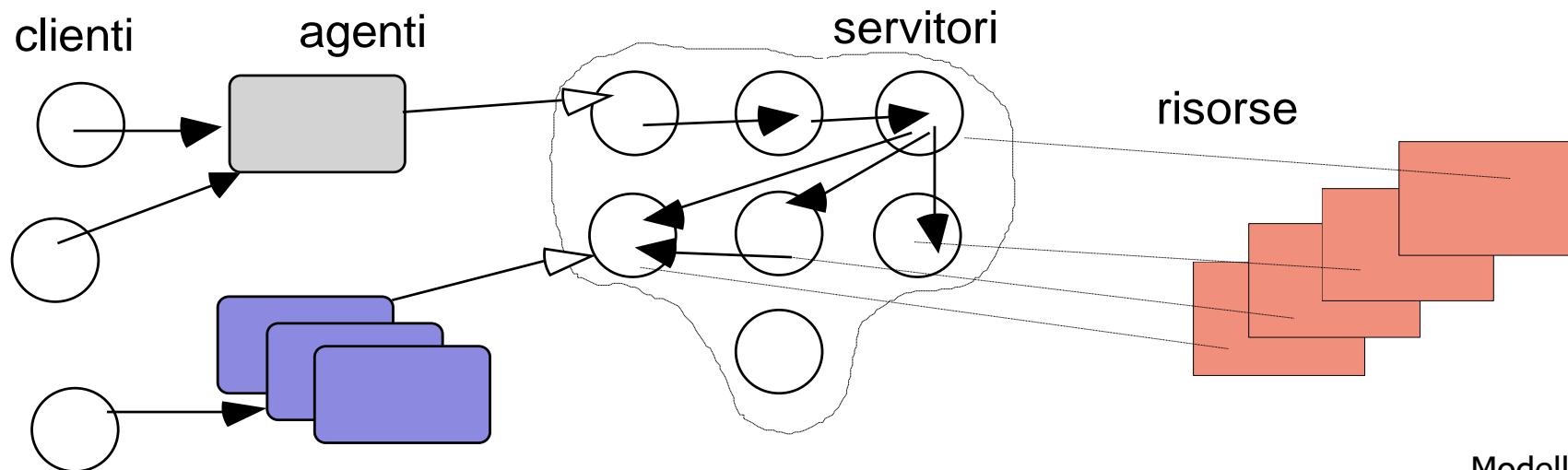
MODELLI a LIVELLI MULTIPLI per la divisione dei compiti

Clienti che interagiscono con Agenti

Agenti anche paralleli e capaci di coordinarsi

Server di operazione paralleli, replicati, e coordinati

Con necessità di **protocolli** di coordinamento, sincronizzazione, rilevazione e tolleranza ai guasti



Server di nomi in-the-large

Un unico server di nomi non può registrare tutti i server del mondo... per questione di memoria e di esecuzione

Sono necessari progetti più ampi, con servizi che usano modalità ad agenti multipli o simili: ossia molti servitori organizzati in una struttura ad-hoc del servizio

I servitori in-the-small possono essere semplici (RPC, RMI),
Quelli **globali in-the-large devono aggiungere fattibilità e qualità QoS** (DNS, X.500, servizi di mail, ...)

Alcuni principi di organizzazione necessari

Partizionamento - molti servitori, ciascuno **con una area limitata di gestione diversa**

Replicazione - molti servitori, ciascuno capace di lavorare **sulla stessa area**

Efficienza servizio - uso di **cache per località**

SISTEMI DI NOMI GLOBALI

Un sistema di nomi deve rendere possibile inserire **nuove entità durante la esecuzione oltre** quelle già esistenti e consentire di ritrovare rapidamente il nome delle entità

Dobbiamo organizzare il **sistema di nomi con una architettura che permetta fattibilità ed efficienza**, attraverso molteplici gestori **gestori partizionati**

ciascuno responsabile di una **sola parte** (partizione) dei riferimenti **località** (in generale i riferimenti più richiesti) –

Non un **solo gestore**, ma una **molteplicità** per rendere fattibile il servizio (**memoria e capacità**)

gestori replicati

ciascuno responsabile con altri di una parte (partizione) dei riferimenti in modo coordinato, per fare fronte a guasti di uno dei gestori dell'insieme –

Non un **solo gestore** (guasti o crash?), ma **più di uno** per superare guasti

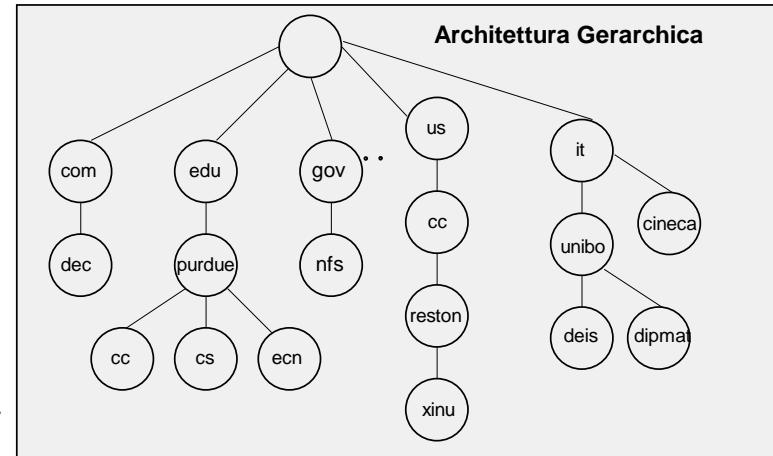
NOMI GLOBALE alla DNS

In caso di moltissimi servitori da registrare, tipicamente si devono prevedere molteplici gestori di nomi distinti e coordinarne la esecuzione (**agenti**) (vedi **DNS**)

I gestori devono essere organizzati tra loro in architettura ad esempio:

in un albero ossia in **livelli a partire dalla radice**

*gestore generale che coordina gestori di più basso livello
in molti livelli con **località informazioni***



Partizionamento dei gestori

Ogni gestore ha solo la responsabilità di una tabella locale di corrispondenza

Replicazione dei gestori

Molti gestori di nome per la stessa tabella

SINTASSI DEI NOMI

Necessità di conoscenza reciproca delle entità

il cliente che vuole richiedere un servizio deve poter riferire il servitore

Questo è reso possibile dal nome del servitore nel cliente come nome logico della applicazione servitore

I clienti possono usare molte sintassi di nomi diversi

- `indirizzoServizio` 123456
- `nomeGestore.nomeServitore` 123:123456
- `nomeNodo.nomeServizio` 123:stampa
- `nomeServitore` prcs#1234
- `nomeServizio` stampa

Nomi TRASPARENTI e NON TRASPARENTI alla allocazione

La trasparenza non lega il nome a dettagli di basso livello ☺

La non visibilità della allocazione o trasparenza nel nome dei servizi ai nodi consente di avere un più alto livello di astrazione

MODELLI DI BINDING

BINDING STATICO vs. DINAMICO

**Binding è il legame tra
la risorsa logica (nome) e
la risorsa fisica (target) e che
permette di risolvere il riferimento e trovare il target**

Come si qualificano i **nomi** e **quando** si risolvono i **riferimenti**?

Prima di eseguire

statico: i riferimenti risolti **prima** della esecuzione

Durante la esecuzione

dinamico: i riferimenti risolti **solo durante l'esecuzione e al momento del bisogno**

MODELLI DI BINDING

I nomi possono essere risolti o le risorse identificate e trovate prima della esecuzione

BINDING STATICO

In caso di sistemi concentrati, **binding tipicamente statico** ma esistono e sono diffuse per riutilizzo anche librerie dinamiche

si risolve il tutto staticamente e non si necessita di un servizio di nomi, vista la **invarianza dei nomi** e della **allocazione delle entità**
I nomi sono risolti **prima della esecuzione** e non è consentito **cambiare alcuna allocazione** (altrimenti insorgono problemi)

Però sempre di più il **binding statico è considerato rigido e inflessibile**

NOMI **DISTRIBUITI DINAMICI**

Nei sistemi distribuiti i **binding sono dinamici perché le risorse sono dinamiche e non prevedibili staticamente**, e i **sistemi di nomi** sono i server di nomi sempre presenti **durante l'esecuzione per consentire il binding dinamico**

In caso *dinamico*, un **servizio di nomi (name server)** è la **entità necessaria a favorire il binding dei clienti ai servitori** e mantiene una tabella per risolvere i nomi e per fornire il servizio durante la esecuzione coordinandosi con altri gestori

Un sistema di nomi fornisce il servizio di risoluzione su richiesta attraverso **tabelle di allocazione delle entità che vengono mantenute da un servitore**

Nomi non trasparenti

dipendenti dalla locazione

Nomi trasparenti

non dipendenti dalla locazione

Se le entità cambiano allocazione, non cambiano i nomi

nomeServizio:

123 : 123456

stampa

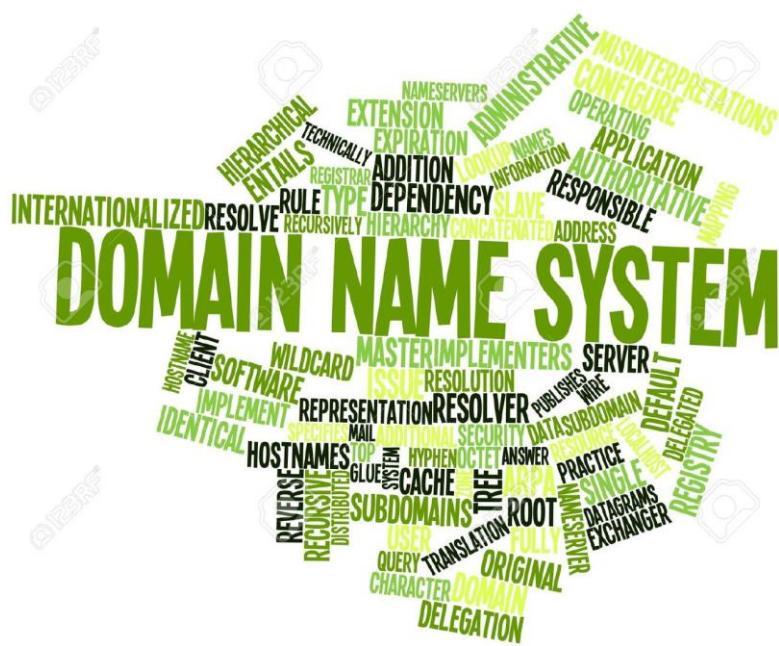
DNS: QUALE IP per NOME?

DNS come un servizio di **nomi globale in Internet** basato su un **insieme di gestori DNS coordinati** che si organizzano per rispondere a **query** che chiedono il **nome di IP** corrispondente ad un **nome di dominio**

Devono esistere sia i nomi di dominio sia i nomi di IP

Il servizio trasla dal secondo al primo

(dato un nome logico di dominio fornisce un indirizzo IP)



Principio di località del DNS

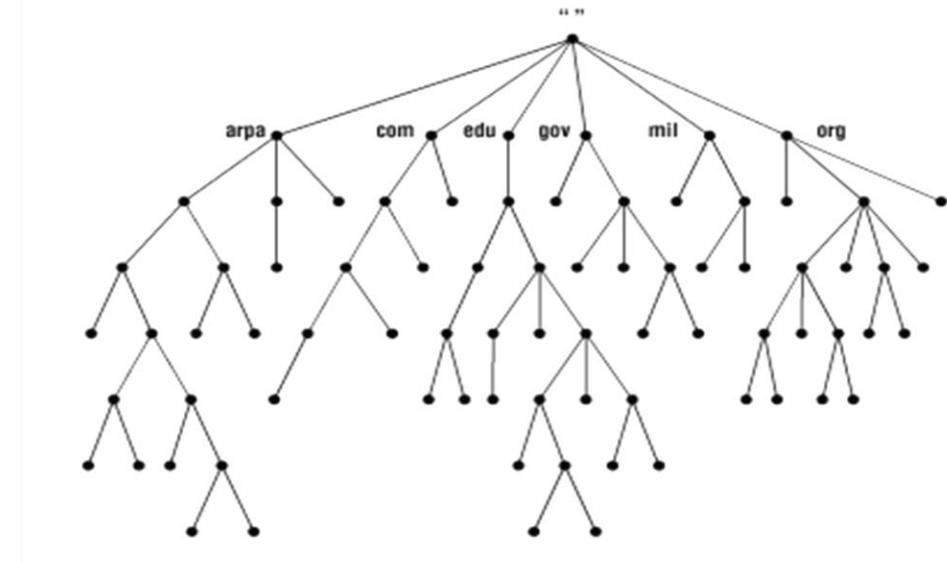
DNS come insieme di gestori **in una gerarchia (albero)** di gestori delle **tabelle locali** di **nomi logici** e relativi **indirizzi IP**

Ogni organizzazione ha *almeno un* proprio **gestore DNS** (anche più di uno) e coordina le richieste dei suoi utenti

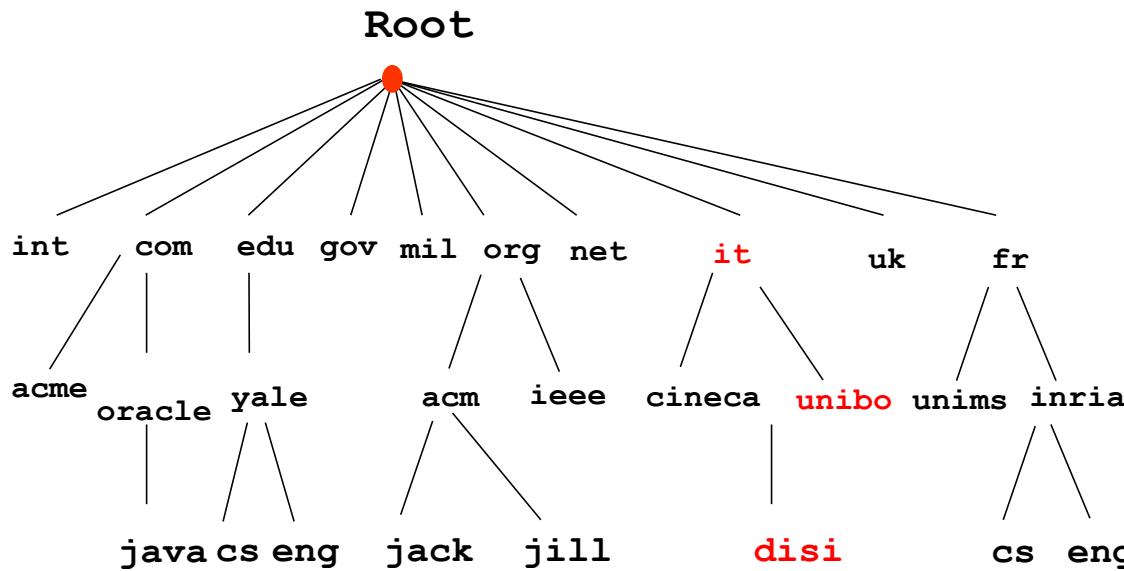
In generale vige il principio di **località per i servizi**

- le richieste **locali** per elementi della tabella sono **efficienti**
 - le richieste per elementi **globali** e non locali della tabella possono essere **meno efficienti**
-

Questo permette di avere un contratto di servizio differenziato in QoS e aspettative diverse



ALBERO di GESTORI DNS



Si sceglie un **albero** perché è una struttura **MOLTO ORDINATA** e **SEMPLICE**:

- Ogni nodo conosce solo un padre e molti figli
- Un percorso preciso congiunge due qualunque nodi dell'albero
- Il percorso implica un cammino coordinato tra servitori diversi

DOMAIN NAME SYSTEM (DNS)

DNS come insieme di gestori di **tabelle di nomi logici e di indirizzi IP**
obiettivo ⇒ **attuare corrispondenze** tra **nome logico host** e **indirizzo IP**

Primo passo iniziale – primi anni 80: uso di un file locale, /etc/hosts

Non sufficiente su scala globale (quante repliche? Scalabile?)

DNS **introduce nomi logici in una gerarchia di servitori (albero di DSN)**
intesa come gerarchia di domini logici e anche di centri di servizio

la corrispondenza tra nomi logici e indirizzi fisici avviene dinamicamente
tramite un servizio **di nomi unificato** che risponde alle richieste

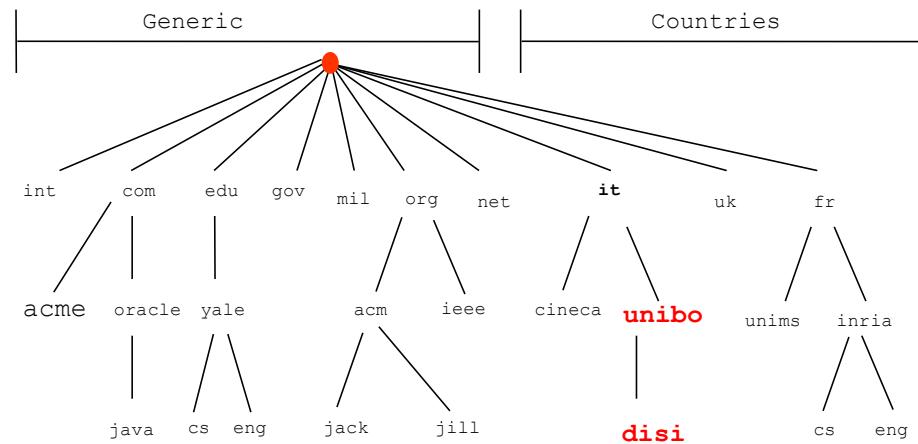
La traslazione è:

statica vs. dinamica

locale vs. globale

**in una gestione globale
non centralizzata ma favorendo
la località**

**Esempio di divisione dei compiti
e coordinamento**



NOMI di sistemi DNS

Ogni nome è diviso in parti che rappresentano **domini diversi (server)**
I domini di base (di primo livello) sono i seguenti:

Nome dominio	Significato
COM	Organizzazioni commerciali
EDU	Istituzioni per l'istruzione
GOV	Istituzioni governative
MIL	Gruppi militari
NET	Maggiori centri di supporto alla rete
ORG	Organizzazioni diverse dalle precedenti
ARPA	Dominio temporaneo dell'ARPANET (obsoleto)
INT	Organizzazioni internazionali (schema geografico)
<i>codice nazionale</i>	Ciascuna nazione (schema geografico)

Un nome logico: disi33.disi.unibo.**it**
riferisce un *dominio nazionale italiano*

Questo nome è a quattro livelli, come numero di domini innestati
Potremmo avere alias: disi33.cineca.**it**
con un nome è a tre livelli, come numero di domini

Il numero dei livelli è dinamico

LIVELLI di CONTESTO o DOMINIO DNS

Ogni nome permette dei mapping logici propri e corrispondenze diverse

`disi33.disi.unibo.it`

country

it = Italia,

organisation

unibo = Università di Bologna,

department

disi = Nome/Sigla Organizzazione locale,

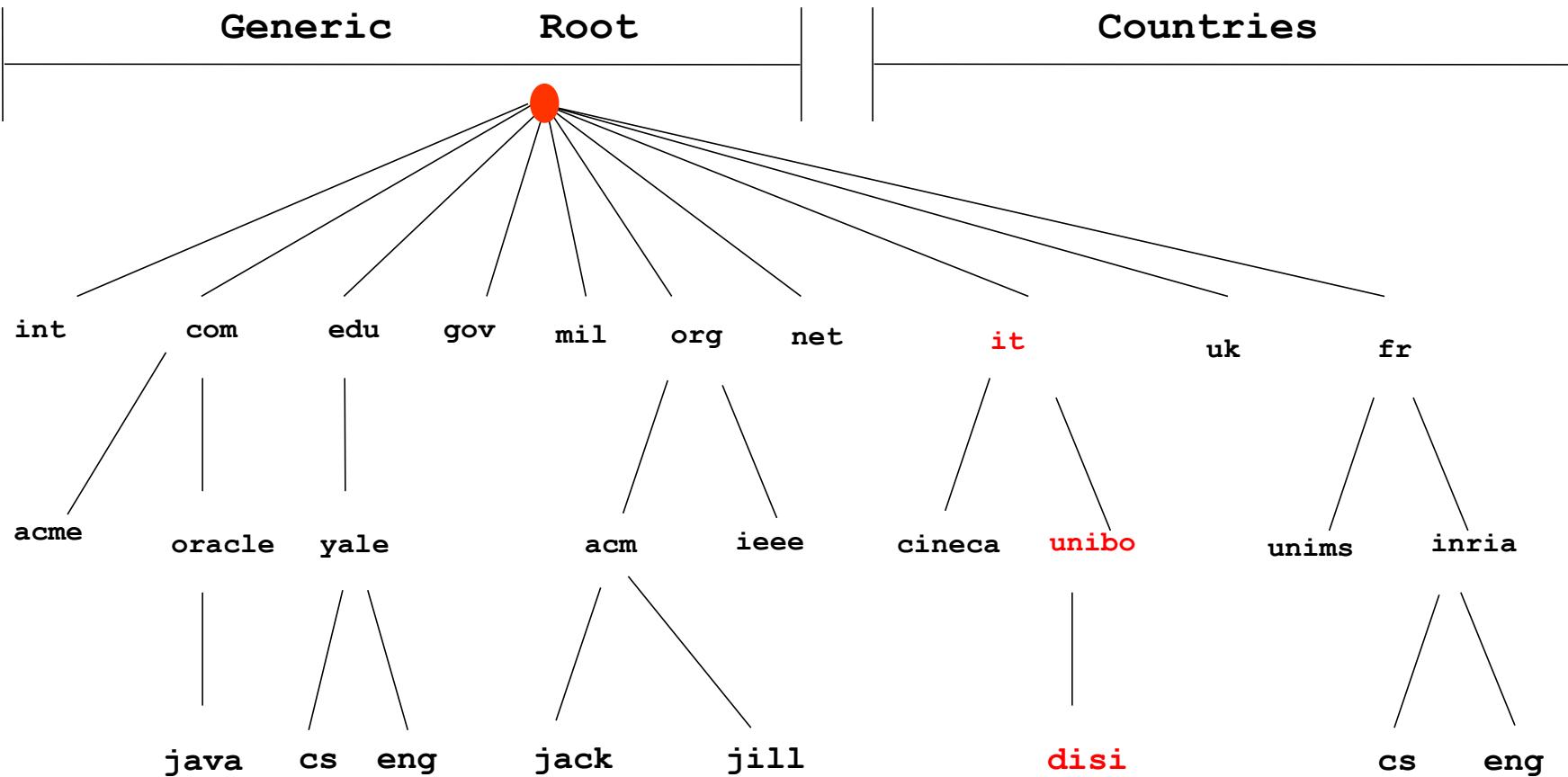
machine

disi33 = nome della macchina,

Livello	Descrizione	Nome dominio	Sigle
minimo	locale	disi33.disi.unibo.it	disi33 = Host
intermedio2	sottogruppo	disi.unibo.it	disi = Department
intermedio1	gruppo	unibo.it	unibo = Organisation
massimo	postazione	it	it = Italy

Livello	Descrizione	Nome dominio	Sigle
minimo	locale	disi33.cineca.it	disi33 = macchina
intermedio	gruppo	cineca.it	cineca = gruppo
massimo	organizzazione	it	it = Italia

ORGANIZZAZIONE DEL DNS

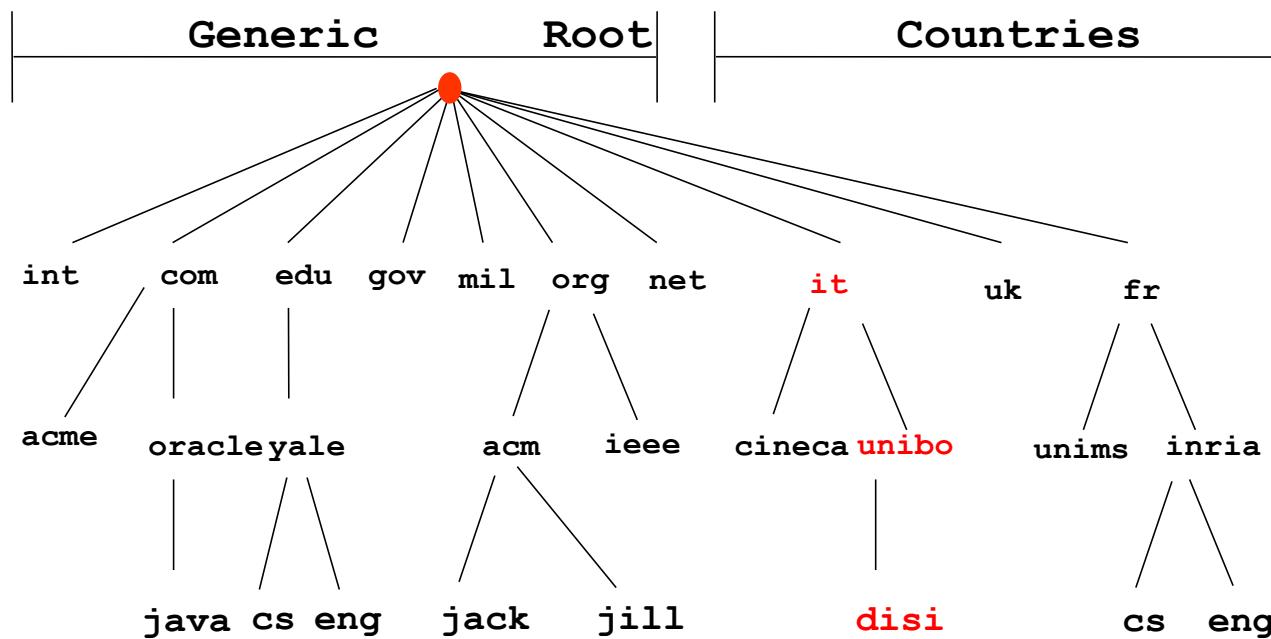


ORGANIZZAZIONE DNS in ALBERO UNICO

In un albero, ogni nodo conosce solo il **proprio padre e i figli**, non altri, rendendo la interconnessione e conoscenza di ogni server DNS molto limitata e predefinita

Un solo percorso unisce due nodi dell'albero

Anche dalla radice (nome assoluto)



Coordinamento di Gestori DNS

La implementazione di un sistema coordinato deve prevedere come i diversi agenti possano comunicare reciprocamente

I singoli gestori dei nomi sono responsabili di una sola tabella locale e quindi devono potere coordinarsi con gli altri server DNS per andare a consultare query su altre tabelle

Il sistema DNS, sulla base dei nomi di dominio, crea un albero che collega insieme tutti i DNS esistenti e che permette alle query che provengono da diversi domini e diversi utenti con la loro località di muoversi per cercare il gestore (e la tabella) che contiene il risultato

Abbiamo un albero globale che mette insieme tutti i servitori DNS

Il coordinamento (e la comunicazione conseguente) avviene solo tra gestori DNS

NOMI di DNS: `disi33.disi.unibo.it`

I singoli nomi sono **case insensitive** e al **max 63 char** per dominio
Il nome completo al **max 255 char**

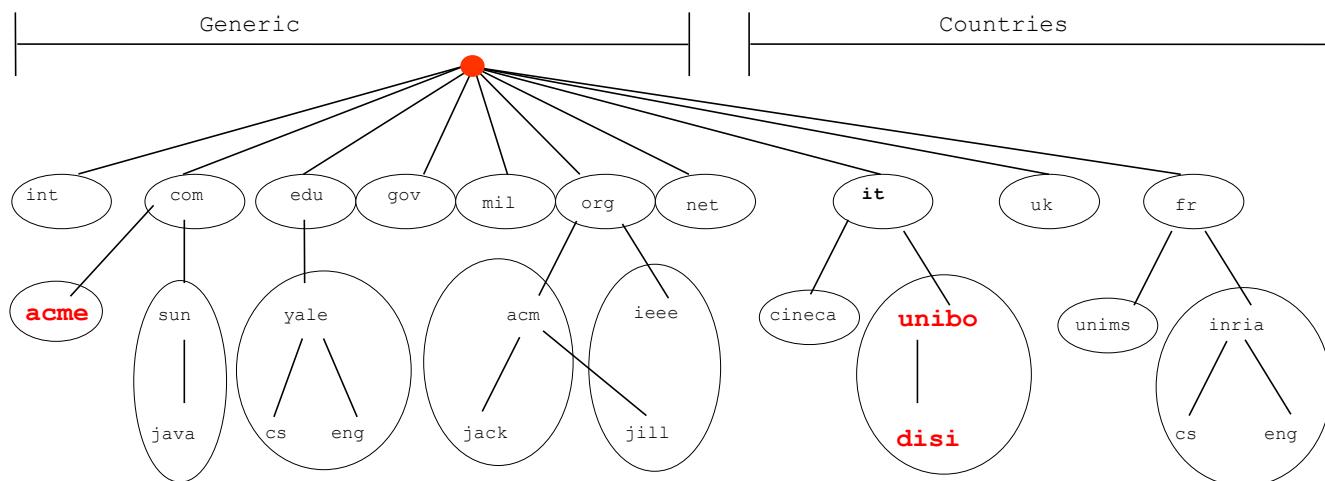
I domini sono del **tutto logici** e non sono collegati in nessun modo alle **reti fisiche** o alle **organizzazioni di rete** (**logico** vs. fisico)

Ogni dominio può essere usato in modo **relativo o assoluto**

Ogni **dominio relativo** fa riferimento al dominio che lo contiene `disi.unibo.it`, navigando sempre da **lì in su o in giù**

disi è interno a unibo, interno a it, che è interno alla root

Possibile gerarchia



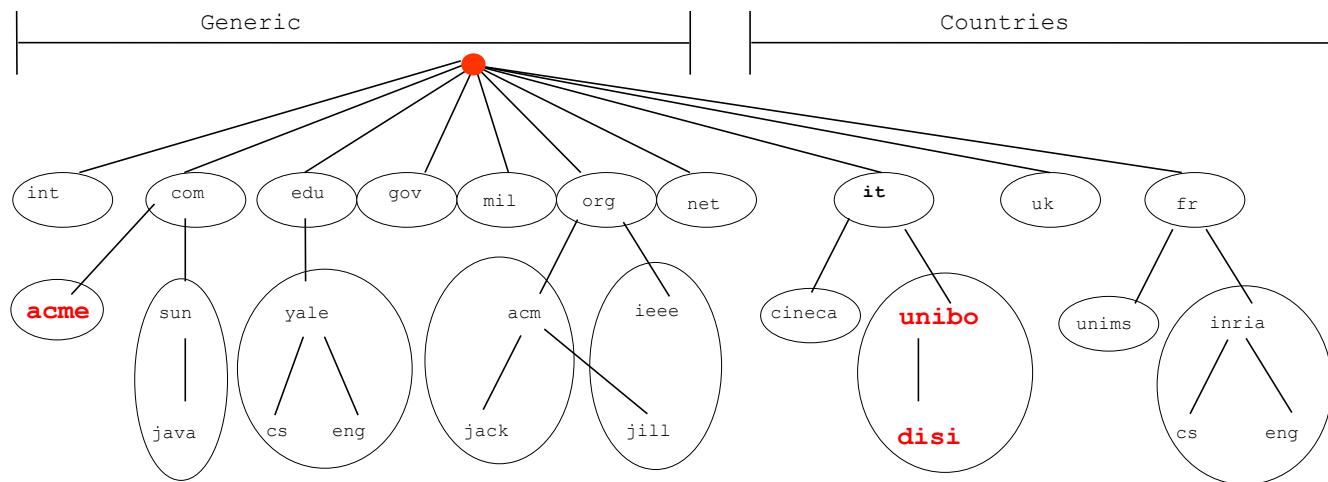
GERARCHIA di SERVITORI di DNS

Ogni dominio dei nomi corrisponde ad un **server di responsabilità**

I domini sono organizzati su **responsabilità primaria** di un **server** (detto di ZONA) suddiviso in zone per ragioni geografiche o di organizzazione

Ogni zona riconosce una autorità ossia un server che fornisce le corrette corrispondenze

Si pensi ad una azienda **acme.com** che si colloca nella **gerarchia** e che gestisce una **zona di responsabilità**

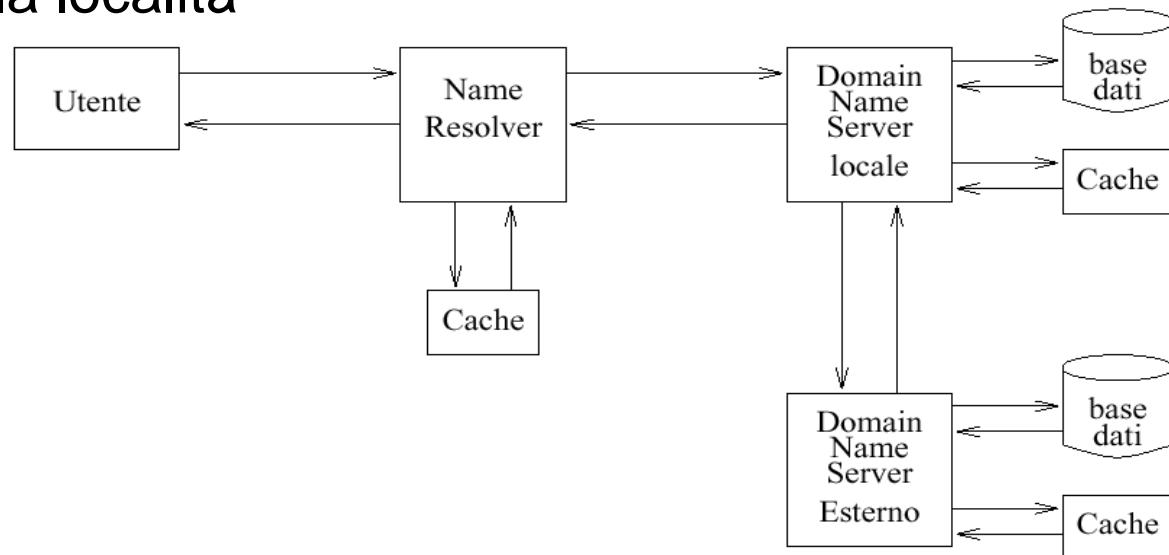


DELEGA di DNS

I diversi **servitori** che gestiscono i **domini suddivisi** in zone d'autorità possono a loro volta **delegare la gestione ad altri server e le richieste transitano sull'albero**

Un **server di dominio** può **delegare sottodomini a servitori sottostanti** (che se ne assumono responsabilità e autorità)

Le richieste sono smistate dal sistema DNS in modo opportuno
Ogni **richiesta di utente** viene fatta **al servizio di nomi in modo indiretto** tramite un **agente specifico (name resolver)** per la gestione dei nomi della località



Si noti il caching
(stato sulle entità)
di corrispondenze
per ogni livello
Per efficienza

ARCHITETTURA del DNS

I diversi server DNS sono organizzati per ottenere

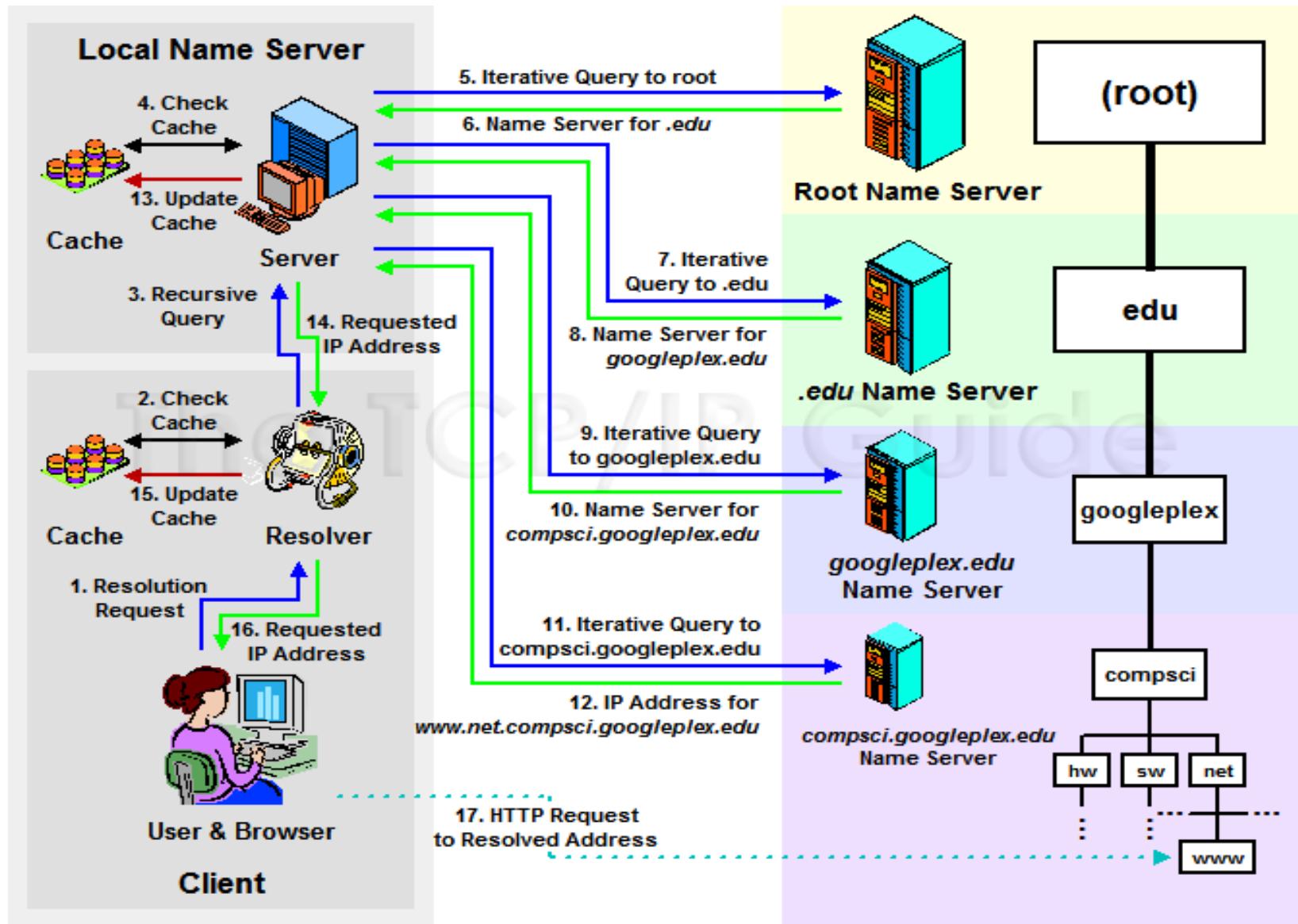
Diversi requisiti ⇒ affidabilità, efficienza, località

Ogni dominio ha **name resolver** e **domain name server** locali (*uno o più*) e usa estensivamente **cache** per conoscenze pregresse

Un cliente chiede un mappaggio al name resolver...

- Il **name resolver** fornisce la risposta o perché conosce già la corrispondenza (**cache**) o la trova attraverso una richiesta C/S ad un name server locale
- I **DNS name server** sono organizzati **come agenti** per ottenere informazioni reciprocamente (dalla più corretta autorità) e potere fornire a loro volta risposte consultando le rispettive tabelle DNS
- La **organizzazione ad albero** consente di muoversi tra i **DNS con le richieste necessarie** fino a raggiungere il dominio di autorità
- I diversi server DNS e i name resolver **sfruttano la replicazione** per fornire risposta anche in caso di problemi

ARCHITETTURA del DNS



REPLICAZIONE del DNS

Ogni dominio corrisponde a più **Domain Name Server**, di cui uno ha autorità sulla traslazione degli indirizzi. Ogni **Domain Name Server** non ha una visione completa, ma solo locale ([la tabella locale di corrispondenza](#))

Necessità di replicazione

- I diversi server DNS **sono anche replicati** per fornire servizio anche a fronte di guasti di server della gerarchia
- In genere, ogni zona ha un **primary master** responsabile per i dati della intera zona, in più una serie di **secondary master** copie del primary, con consistenza garantita dal protocollo DNS (non massima)
- I secondari **non eseguono e non rispondono alle query**
- Il primario di una zona può diventare il **backup** (master secondario) di un'altra zona
- allo start up il **secondario chiede i dati al primario** e può fare fronte al traffico in caso di guasto
- Ad intervalli prefissati, i **secondari chiedono le informazioni al primario** ([modello pull](#))

PROTOCOLLI di DNS

Efficienza su località

Per favorire le richieste a maggiore località, si mantengono i dati ottenuti in previsione di nuove richieste

Tutti i resolver e i DNS server mantengono informazioni di caching per ottimizzare i tempi di risposta al cliente

Protocolli di richiesta e risposta per il name server

I clienti e resolver fanno richieste usando di protocollo UDP
(comunicazione usando le porte 53)

Lo stesso tra i diversi DNS tra di loro

In caso di messaggi troppo lunghi si usa TCP, eventualmente dopo una risposta di eccezione: uso di TCP per l'aggiornamento dei diversi secondari che devono leggere delle tabelle di responsabilità altrui

RISOLUZIONE QUERY DNS - ATTIVO

Il **name resolver** conosce un **server di dominio** e attua le query locali sul suo **DNS server** (**il name resolver è replicato localmente**)

Il **resolver** invia una query al suo server DNS di dominio che risponde coordinandosi nel caso con altri server DNS

Il **protocollo** tra **server DNS** ha **due tipi di query**:

- **ricorsiva** che prevede una catena di server request/reply e
- **iterativa** che prevede un server DNS che attua una sequenza di richieste request/reply a server

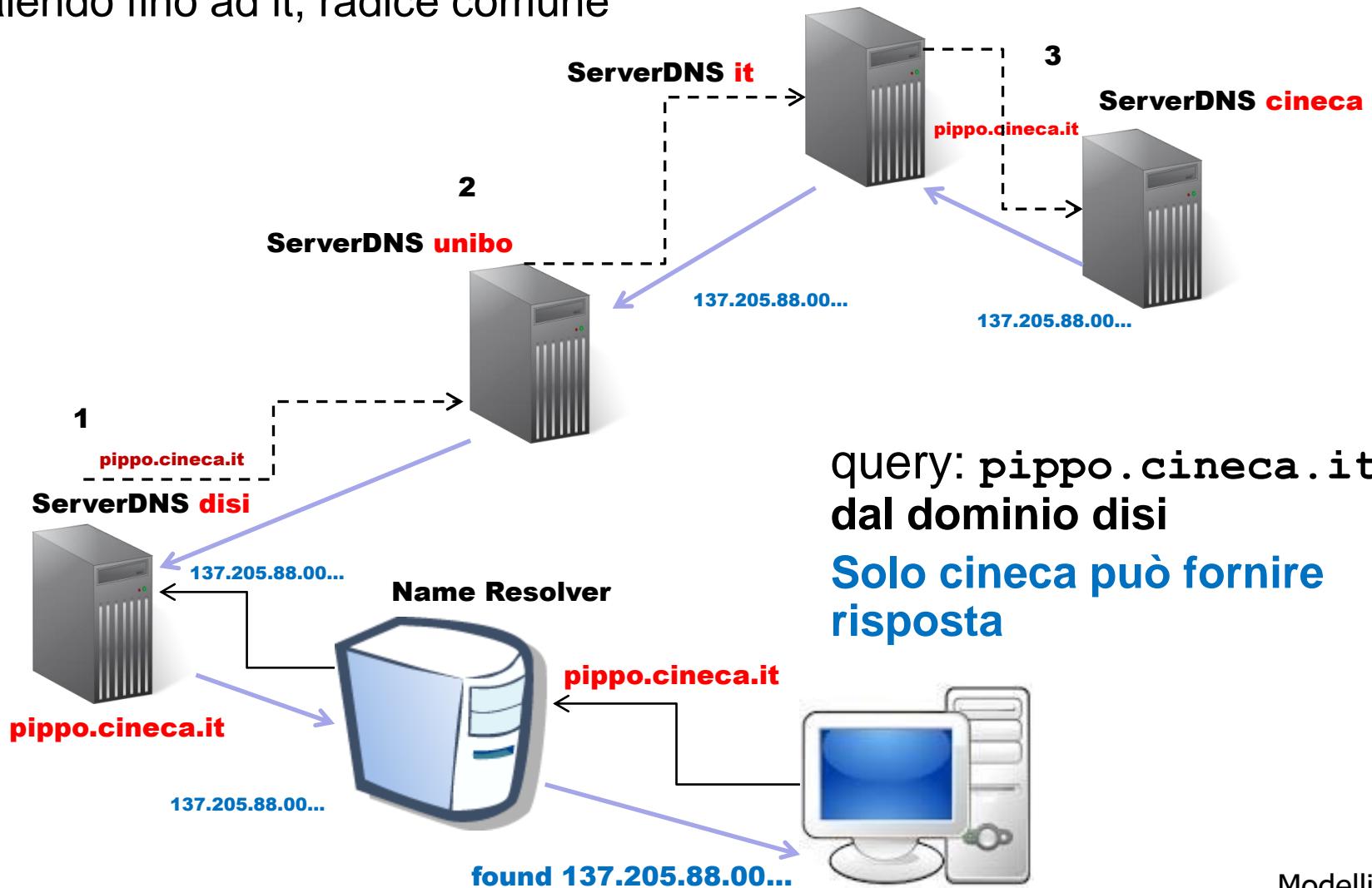
La **ricorsiva** comporta che al richiedente (DNS locale)
o si fornisca risposta (anche chiedendo ad altri)
o si segnali errore (dominio non esistente, etc.)

La **iterativa** comporta che al richiedente DNS locale si fornisca
o la risposta
o il migliore suggerimento come riferimento al migliore DNS server

QUERY DNS RICORSIVA nell'albero

query: **pippo.cineca.it** al server DNS **disi.unibo.it**

Risalendo fino ad it, radice comune

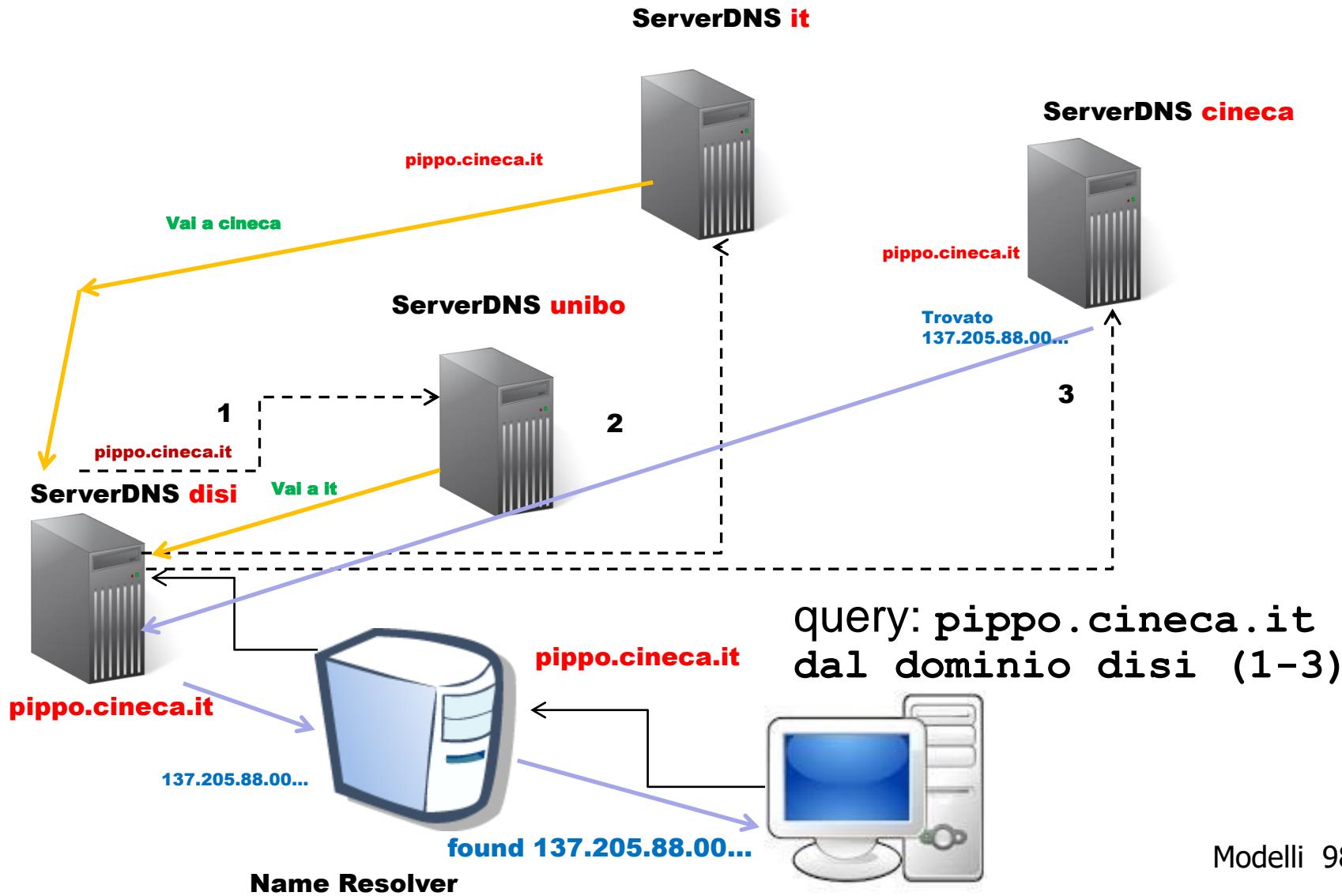


query: **pippo.cineca.it**
dal dominio **disi**

**Solo cineca può fornire
risposta**

QUERY DNS ITERATIVA nell'albero

query: pippo.cineca.it a server DNS disi.unibo.it



RISOLUZIONE NOMI tra DNS

Se il server possiede il **nome** (in tabella o cache), **risponde, altrimenti**

Se **query ricorsiva**, **cerca altre risposte** e rimane impegnato fino a risposta o time-out

Se **query iterativa**, il server che non possiede il nome, risponde con **un riferimento al gestore più vicino che possa ragionevolmente rispondere** (considerando l'albero dei servitori ed il nome fornito)

Ogni name server decide come rispondere

TIPICAMENTE, il name server locale fa **query ricorsiva**, senza conoscenza a priori della gerarchia ma solo del DNS locale

Il name server root e altri a livelli alti non consentono **query ricorsive**

Si usano **time-out** ed eventualmente il server ne consulta altri e, se le zone hanno secondari, ci si rivolge anche a quelli

TIPICAMENTE, si provano **diversi server** noti mandando a ciascuno almeno due o più ritrasmissioni con time-out crescenti (4 volte)

se non c'è risposta si assume che il server sia fallito
(timeout \Rightarrow server crash)

TABELLA NOMI DNS

Un server crea una tabella con un record sorgente per ogni risorsa e lo carica dinamicamente da un file di configurazione al caricamento, e lo aggiorna quando è il caso. Le query consultano la tabella con l'insieme dei **record, con molti attributi** come:

Nome dominio

Time to live (tempo di validità in secondi)

Classe (Internet IN)

Tipo (descrizione del tipo)

Valore (valore dell'attributo)

Tipo	Significato	Valore
SOA	Start of Authority	parametri della zona
A	IP host address	intero a 32 bit (dot notation)
MX	Mail exchange	server di domino di mail
NS	Name server	server per dominio corrente
CNAME	Canonical name	alias di nome in un dominio
PTR	Pointer	per corrispondenza inversa
HINFO	Host description	descrizione di host e SO
TXT	Text	testo qualunque

ESEMPIO FILE NOMI DNS

```
@ IN SOA promet1.deis.unibo.it. postmaster.deis.unibo.it.  
          (644 10800 1800 604800 86400)
```

```
; serial number, refresh, retry, expiration, TTL in sec
```

```
IN NS promet1.deis.unibo.it.
```

```
IN NS almadns.unibo.it.
```

```
MX 10 disi.unibo.it.
```

```
MX 40 mail.ing.unibo.it.
```

```
lab2 IN NS lab2fw.disi.unibo.it.
```

```
lab2fw IN A 137.204.57.136
```

```
IN HINFO HW:PC IBM SW:WINDOWS 95
```

```
IN WKS 137.204.57.136 TCP FTP TELNET SMTP
```

```
IN MX 40 lab2fw.disi.unibo.it.
```

```
deis18 IN TXT "Qualunque testo non significativo"
```

```
deis18 IN RP root.disi.unibo.it luca\ghedini.mail.ing.unibo.it
```

```
; record per responsabile
```

```
146 IN PTR disicorradi.disi.unibo.it.
```

```
; record per la corrispondenza inversa
```

QUERY NOMI DNS

A server DNS si fanno generalmente **query dirette**, cioè
si entra con un nome logico e ci si aspetta un (solo) valore IP



Uso di nslookup come tool o integrato via molti web site:

<http://www.intodns.com/>,

<http://network-tools.com/>, spesso con funzioni diverse di accesso

Sono possibili **query inverse**



QUERY DIRETTA DNS

> [unibo.it.](#)

Server: **promet1.disi.unibo.it** Address: 137.204.59.1, res_mkqry(0, unibo.it, 1, 1)

Got answer: HEADER:

opcode = QUERY, id = 15, rcode = NOERROR header flags: response, want recursion, recursion avail. questions =1, answers =1, authority rec. = 4, additional =4

QUESTIONS: unibo.it, type = A, class = IN

ANSWERS:

-> unibo.it internet address = 137.204.1.15 ttl = 58196 (16 hours 9 mins 56 secs)

AUTHORITY RECORDS:

-> unibo.it nameserver = dns2.cineca.it ttl = 155488 (1 day 19 hours 11 mins 28 s)

-> unibo.it nameserver = dns.nis.garr.it ttl = 155488 (1 day 19 hours 11 mins 28 s)

-> unibo.it nameserver = dns.cineca.it ttl = 155488 (1 day 19 hours 11 mins 28 s)

-> unibo.it nameserver = almadns.unibo.it ttl = 155488 (1 day 19 hours 11 ms 28 s)

ADDITIONAL RECORDS:

-> dns2.cineca.it internet address = 130.186.1.1 ttl = 258410 (2 days 23 hours 46 mins 50 secs) -----

Non-authoritative answer:

Name: unibo.it Address: 137.204.1.15

QUERY DIRETTA DNS

> **cineca.it.**

Server: **promet1.disi.unibo.it** res_mkquery(0, cineca.it, 1, 1)

Got answer: HEADER:

opcode = QUERY, id = 28, rcode = NOERROR header flags: response, want recursion, recursion avail. questions =1, answers =0, authority records =1, addit. = 0

QUESTIONS: cineca.it, type = A, class = IN

AUTHORITY RECORDS:

-> cineca.it

ttl = 10784 (2 hours 59 mins 44 secs)

origin = dns.cineca.it

mail addr = hostmaster.cineca.it

serial = 1999052501

refresh = 86400 (1 day)

retry = 7200 (2 hours)

expire = 2592000 (30 days)

minimum ttl = 259200 (3 days)

Name: cineca.it Address: 130.186.1.1

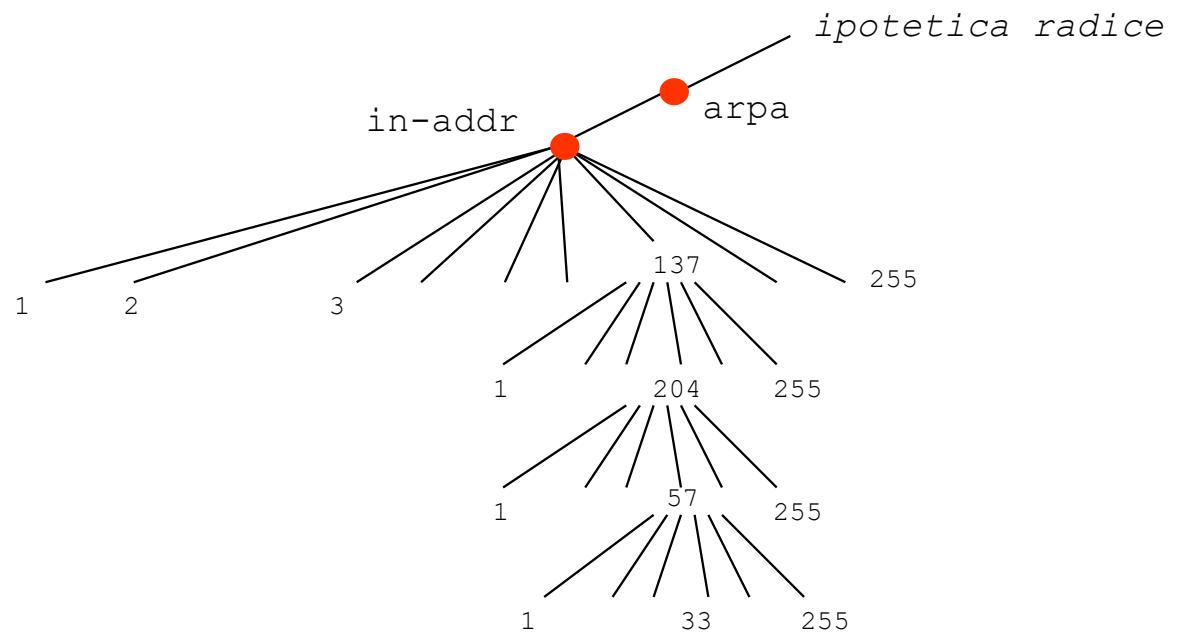
QUERY NOMI DNS

Sono possibili **query inverse**

si entra con un numero IP e ci si aspetta un nome logico

Spesso non sono visibili per questioni di privacy e sicurezza dei domini

Queste risoluzioni
richiedono
di mantenere
un **albero**
di corrispondenza
inverso



QUERY INVERSA DNS

Per una **query inversa**, si deve lavorare **sull'albero inverso di pointer** e con **l'indirizzo girato (in termini di byte)**

qui si vuole il corrispondente di **137.204.57.111...**

> set q=ptr

> 111.57.204.137.in-addr.arpa.

Server: promet1.deis.unibo.it Address: 137.204.59.1

111.57.204.137.in-addr.arpa

name = disi057111.ing.unibo.it

57.204.137.in-addr.arpa

nameserver = admii.arl.army.mil

57.204.137.in-addr.arpa

nameserver = almadns.unibo.it

57.204.137.in-addr.arpa

nameserver = promet.disi.unibo.it

57.204.137.in-addr.arpa

nameserver = promt1.disi.unibo.it

admii.arl.army.mil

internet address = 128.63.31.4

admii.arl.army.mil

internet address = 128.63.5.4

almadns.unibo.it

internet address = 137.204.1.15

...

Non tutti i server consentono queste query! Come mai? E come vengono prese le decisioni

QUERY INVERSA DNS

> 111.57.204.137.in-addr.arpa

Got answer:

HEADER:

opcode = QUERY, id = 17, rcode = NXDOMAIN
header flags: response, want recursion, recursion avail.
questions = 1, answers = 0, authority records = 1, additional = 0

QUESTIONS:

111.57.204.137.in-addr.arpa.disi.unibo.it, type = PTR, class = IN

AUTHORITY RECORDS:

-> disi.unibo.it
ttl = 10800 (3 hours)
primary name server = leporello.cs.unibo.it
responsible mail addr = dns.cs.unibo.it
serial = 1522308835
refresh = 3600 (1 hour)
retry = 300 (5 mins)
expire = 172800 (2 days)
default TTL = 43200 (12 hours)

QUERY INVERSA DNS

...

Got answer:

HEADER:

opcode = QUERY, id = 19, rcode = NOERROR
header flags: response, want recursion, recursion avail.
questions = 1, answers = 1, authority records = 0, additional = 0

QUESTIONS:

111.57.204.137.in-addr.arpa, type = PTR, class = IN

ANSWERS:

-> 111.57.204.137.in-addr.arpa
name = disi057111.ing.unibo.it
ttl = 86400 (1 day)

Risposta da un server non autorevole:

111.57.204.137.in-addr.arpa
name = **disi057111.ing.unibo.it**
ttl = 86400 (1 day)

ALTRI SISTEMI DI NOMI (OLTRE DNS)

Altri sistemi di nomi, oltre al DNS molto specifico, hanno organizzato i nomi attraverso **attributi e ricerca su questi**

OSI X.500 o Directory - Servizio standard di Direttorio e di Nomi con realizzazione partizionata, decentralizzata, disponibile 24/7

- **CCITT** definisce **X.500** come "una collezione di **sistemi aperti** che **cooperano** per mantenere un database logico di informazioni sugli **oggetti del mondo reale**. Gli utenti della Directory possono **leggere** o **modificare** l'informazione, o parte di essa, solo se hanno i **privilegi** necessari"

CCITT	ISO	TITLE
X.500	9594-1	Overview of Concepts, Models and Services
X.501	9594-2	Models
X.509	9594-8	Authentication Framework
X.511	9594-3	Abstract Service Definition
X.518	9594-4	Procedures for Distributed Operation
X.519	9594-5	Protocol Specifications
X.520	9594-6	Selected Attribute Types
X.521	9594-7	Selected Object Classes
X.525	9594-9	Replication

DIRECTORY X.500 OSI

X.500 è un insieme di nodi organizzati in un albero di interconnessione garantendo una certa QoS su operazioni

- Ogni nodo è costituito da attributi tipizzati che possono assumere valori: età = intero
- Ogni nodo prevede attributi in base alla gerarchia: aggiunge attributi rispetto al genitore
- Ogni gerarchia ha attributi in parte condivisi (padre) e poi differenziati da ogni nodo
- Le autorizzazioni alle operazioni sono molto granulari (anche su sottoalbero o nodo)

Non visibile e QoS interne

- I nodi sono replicati in modo trasparente e si garantisce sempre l'accesso alle informazioni
- Causa replicazione le operazioni raccomandate ed efficienti sono letture; le scritture devono essere limitate

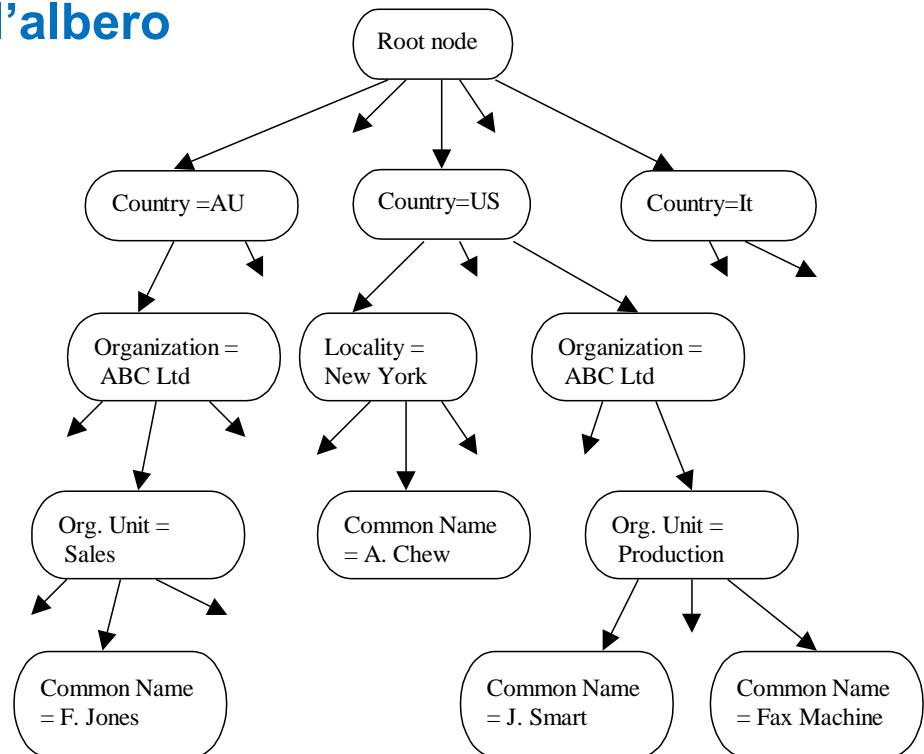
DIRECTORY X.500

X.500 è un insieme di standard di nomi, articolato e completo

La base è l'insieme delle informazioni che caratterizzano la struttura di **directory**, organizzate in un **albero logico** detto Directory Information Tree (**DIT**) a formare il Directory Information Base (**DIB**)

L'**albero logico** è costruito da nodi organizzati in base al valore di attributi del tutto liberi e definiti sull'albero

La novità sta nella **organizzazione basata sui contenuti** e le **ricerche** (operazioni di lettura) che si possono fare in **modo molto flessibile** per **singole entità** e anche per **attributo**, ritrovando gruppi di elementi (nodi) anche molto numerosi (interi sottoalberi e foreste)



DIRECTORY X.500 QUERY

**Su un directory X.500 si fanno query in lettura
che possono portare a identificare almeno un nodo
1 solo nodo**

- **Si definisce il nome del nodo attraverso un percorso assoluto o relativo**

Molti nodi (in gerarchia o meno)

- **La query è determinata dai valori degli attributi e dalle relazioni specificate tra queste** (le risposte potrebbero anche arrivare, richiedendo di attraversare l'intero albero)

I nodi ottenuti come risultato sono disponibili in modo completo in ogni valore degli attributi

**Come avviene la operazione di ricerca?
(sequenziale, parallela)**

DIRECTORY X.500 QUERY

Su un direttorio X.500 si fanno query in lettura
che possono portare ad identificare almeno un nodo

1 solo nodo

Nome specifico
di percorso



1 Nodo ritrovato
tutti gli attributi

Molti nodi (in gerarchia o meno)

espressione su
attributi



più Nodi trovati e
tutti gli attributi

La operazione di ricerca può essere molto articolata e quindi viene ad essere
molto parallelizzabile e in effetti parallelizzata

NOMI E QUERY IN DIRECTORY X.500

Ogni entry (o nodo) si può identificare attraverso il cammino

- **Distinguished Name** (DN) che identifica univocamente l'oggetto all'interno del DIT **(percorso assoluto)**
- **Relative Distinguished Name** (RDN) che definisce univocamente un oggetto all'interno di un contesto **(percorso relativo)**

DN o RDN può fungere da chiave per **identificare unicamente un solo nodo**

Alcuni insiemi di nodi si possono selezionare e ottenere con query **attraverso valori delle tuple**, insiemi di coppie **attributo = valore**

Ad esempio: country organization organization unit Common name
US ABCLtd Production J.Smart

Le ricerche portano a ottenere informazioni su **più nodi** ciascuno dei quali **contiene tutti i propri valori degli attributi**

NOMI E QUERY IN DIRECTORY X.500

Le ricerche (**lettura**) possono essere fatte in modo **globale** o **contestuale** per uno **specifico DN** ma **anche attraverso query su valori contenuti da attributi, ossia in base ad attributi ricercati su tutti i nodi dell'albero**

Una query porta ad avere come risultato **uno o più nodi**

Di conseguenza **tutti i valori degli attributi di quei nodi**

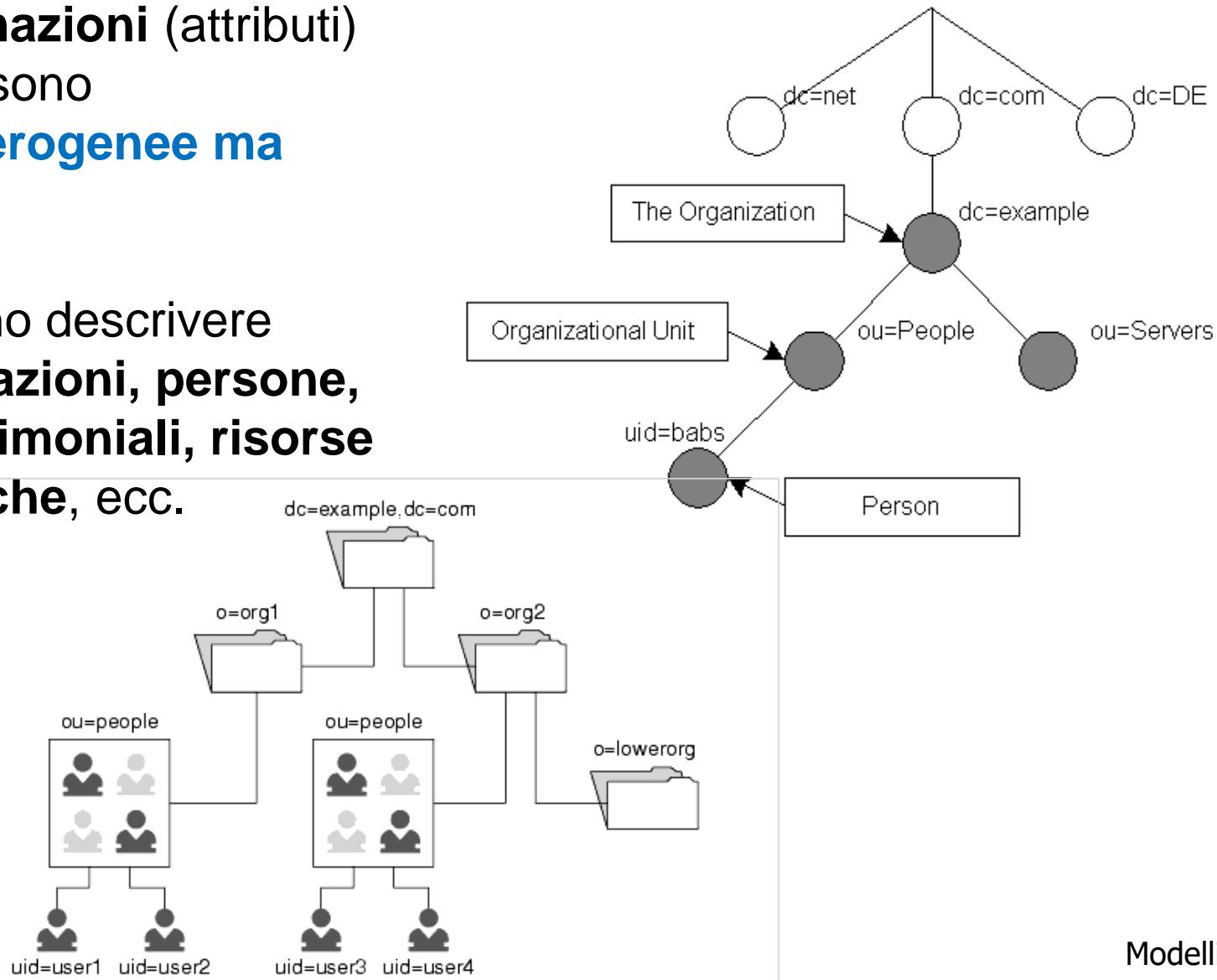
Se avessimo chiesto i nodi con organization ABCLtd avremmo una serie di nodi

	country	organization	organization unit	Common name
N1	US	ABCLtd	Production	J.Smart
N2	DK	ABCLtd	Management	J.Soren
N3	FR	ABCLtd	Industry	A.Serin
...				

INFORMAZIONI IN DIRECTORY X.500

Le **informazioni** (attributi) dei **nodi** sono **molti eterogenee ma ordinate**

Si possono descrivere **organizzazioni, persone, beni patrimoniali, risorse geografiche, ecc.**



RICERCHE CON FILTRI

I filtri sono **molti potenti come capacità espressiva**
ad esempio, sono permesse **condizioni logiche sugli attributi**

`CN=Corradi AND C=Italy`

anche con espressioni regolari `email=*\hotmail*`

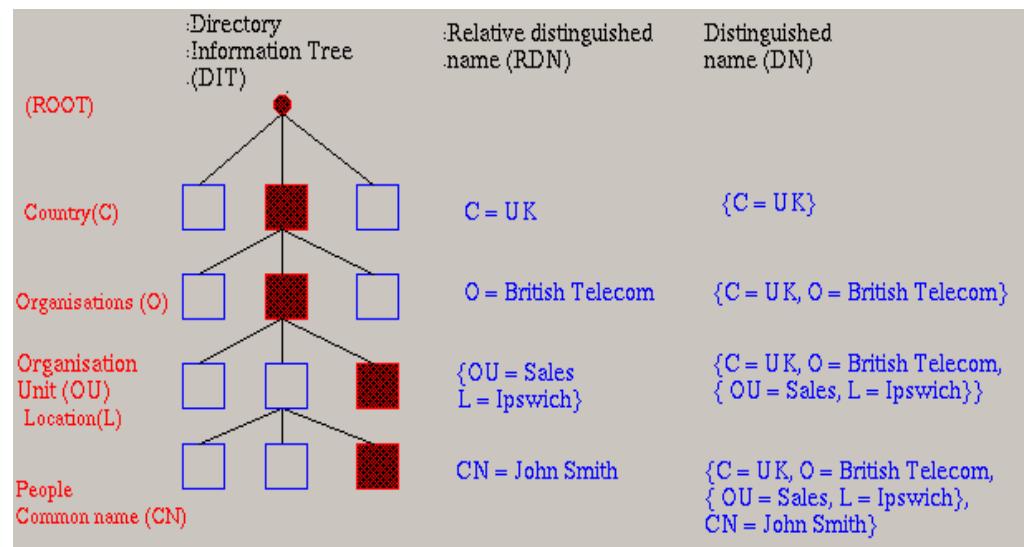
anche con condizioni aritmetiche (`age >18`) AND (`cookies <10`)

Le ricerche si possono applicare anche a scope limitati (contesti o sottoalberi)

**Le operazioni previste
sono molte**

La prima è il **bind** con il
directory poi **ricerche frequenti**
e **cambiamenti rari**

L'interfaccia con il direttorio
anche molto complessa tenendo
conto anche della durata delle
operazioni



SISTEMI DI NOMI GLOBALI

I **SISTEMI DI NOMI GLOBALI** sono molto **DISTRIBUITI**
dimensione di Internet (mondiali) **in-the-large**
Quindi non variabili o lentamente variabili

Si pensi alle operazioni su sistemi di nomi globali
e alla loro implementazione

Le operazioni previste sono sia di lettura, sia di scrittura
Le query di lettura sono molte e
le registrazioni / deregistrazioni poche

Quindi si ottimizzano le letture e si curano di meno le (rare)
operazioni di scrittura

NOMI di INTERNET e OSI

STANDARD di comunicazione ISO – OSI

OSI Open System Interconnection con obiettivo di comunicazione aperta tra reti e tecnologie diverse proprietarie

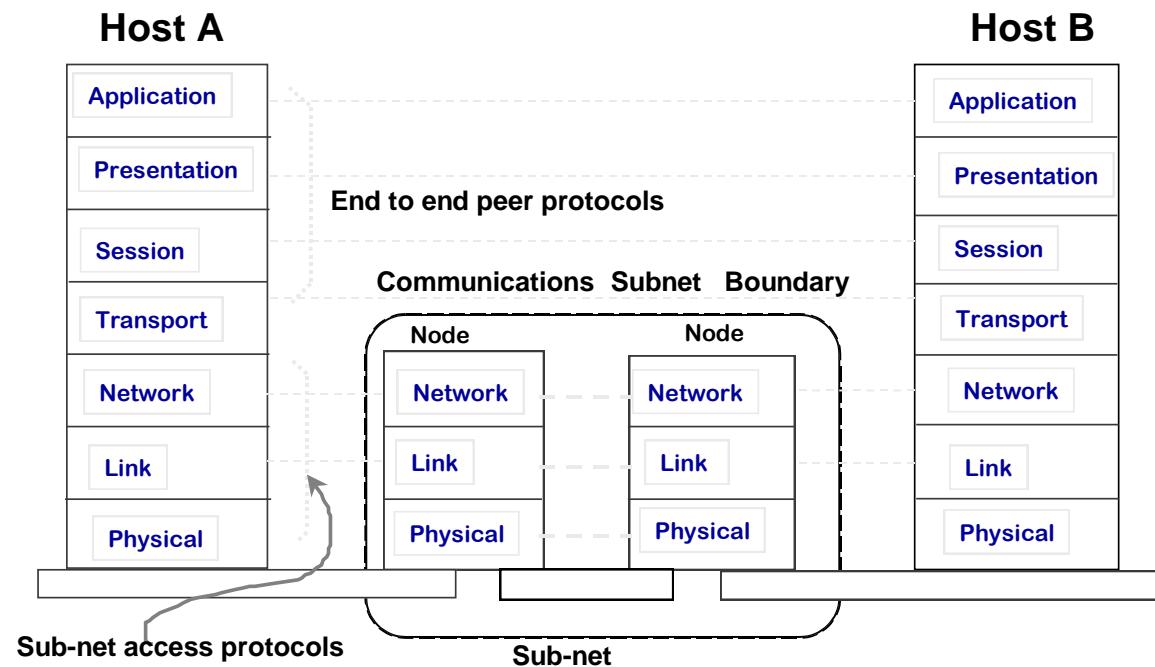
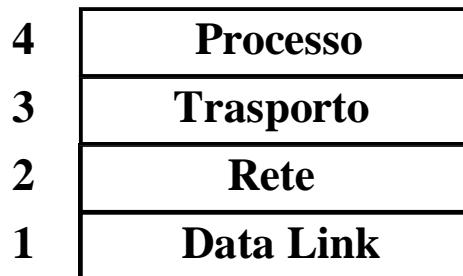
ESEMPI di sistemi APERTI: UNIX che non lega ad un produttore, con software free (open source) e con driver TCP/IP ossia Internet

Vantaggi degli standard aperti ⇒ INTEROPERABILITÀ

In TCP/IP

Livello 3: porte UDP e TCP

Livello 2: nomi IP



NOMI di INTERNET o TCP/IP

Livello Trasporto definisce le **porte** per i servizi

Livello IP, definisce i **nomi IP** per i diversi nodi nella comunicazione

NOMI IP: IP individua connessioni nella rete virtuale, definendo per ogni connessione un indirizzo Internet unico a 32 bit

IP-ADDRESS ⇒ suddiviso nella coppia (**NETID**, **HOSTID**)

STANDARD IETF prescrive nomi dati di autorità

Network Information Center (NIC) assegna il numero di rete, cioè l'informazione usata nei gateway per routing e poi la autorità locale definisce i nomi di host

