



Luca Cabibbo  
Architettura  
dei Sistemi  
Software

# Stili client-server e peer-to-peer

**dispensa asw420**  
ottobre 2023

*The best thing about the future  
is that it comes one day at a time.*  
Abraham Lincoln



## - Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
  - Capitolo 22, **Stili client-server e peer-to-peer**
- ❑ [SAP] Len Bass, Paul Clements, Rick Kazman. **Software Architecture in Practice**. Addison Wesley, third edition, 2013
- ❑ Cervantes, H. and Kazman, R. **Designing Software Architectures: A Practical Approach**. Addison Wesley, 2016.
- ❑ Microsoft. **Microsoft Application Architecture Guide: Patterns & Practices**, second edition. Microsoft Press, 2009.
- ❑ Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. **Distributed Systems: Concepts and Design**, fifth edition. Pearson, 2012.
- ❑ Alonso, G., Casati, F., Kuno, H., and Machiraju, V. **Web Services: Concepts, Architectures and Applications**. Springer, 2004.



# - Obiettivi e argomenti

## □ Obiettivi

- presentare due pattern architetture fondamentali per sistemi distribuiti – lo stile client-server e lo stile peer-to-peer

## □ Argomenti

- introduzione
- stile client-server
- stile peer-to-peer
- discussione



## \* Introduzione

- Gli stili client-server e peer-to-peer sono alla base di molti sistemi distribuiti
  - lo *stile client-server* (*architettura client-server*) consente a un insieme di client distribuiti di accedere ai servizi e alle risorse computazionali offerte da uno o più server
  - lo *stile peer-to-peer* (*architettura peer-to-peer*) consente a un insieme di peer distribuiti di condividere tra di loro dei servizi e delle risorse computazionali
  - questi stili possono essere applicati in numerose varianti
  - altri pattern architetture per sistemi distribuiti possono essere considerati delle evoluzioni di questi pattern fondamentali



# Servizi e risorse computazionali

- Questi due stili consentono l'accesso o la condivisione di “servizi” e “risorse computazionali”
  - servizio
    - un insieme di funzionalità, che client diversi possono usare per scopi differenti
  - risorse computazionali
    - risorse di memorizzazione – per la gestione e l'accesso a dati – ad es., un file system o una base di dati
    - risorse di elaborazione – la capacità di eseguire operazioni, servizi, applicazioni o computazioni
    - altre risorse – ad es., la condivisione di una stampante o di hardware specializzato
- questi due stili differiscono soprattutto nel posizionamento e nella gestione di questi servizi e di queste risorse



## \* Stile client-server

- Contesto
  - ci sono delle risorse computazionali o dei servizi che devono essere condivisi
  - un gran numero di utenti o “client” distribuiti vogliono accedere a queste risorse e servizi
  - è necessario controllare l'accesso a queste risorse e servizi, e la qualità del servizio
- Problema
  - bisogna gestire e consentire l'accesso a un insieme di risorse computazionali e servizi condivisi
  - si vogliono sostenere modificabilità e riuso di queste risorse e servizi
  - si vogliono sostenere scalabilità e disponibilità nell'accesso a queste risorse e servizi



## Stile client-server

### □ Soluzione (struttura)

- organizza il sistema come un insieme di server e di client
  - ciascun **server** è un componente software (un processo) in grado di fornire/erogare uno o più servizi ai suoi client
  - ciascun **client** è un componente software (un processo) interessato a fruire di uno o più servizi da uno o più server
- ciascun **servizio** rappresenta una funzionalità oppure una risorsa computazionale – ed è caratterizzato da un'interfaccia, basata su un connettore di tipo richiesta-risposta



## Stile client-server

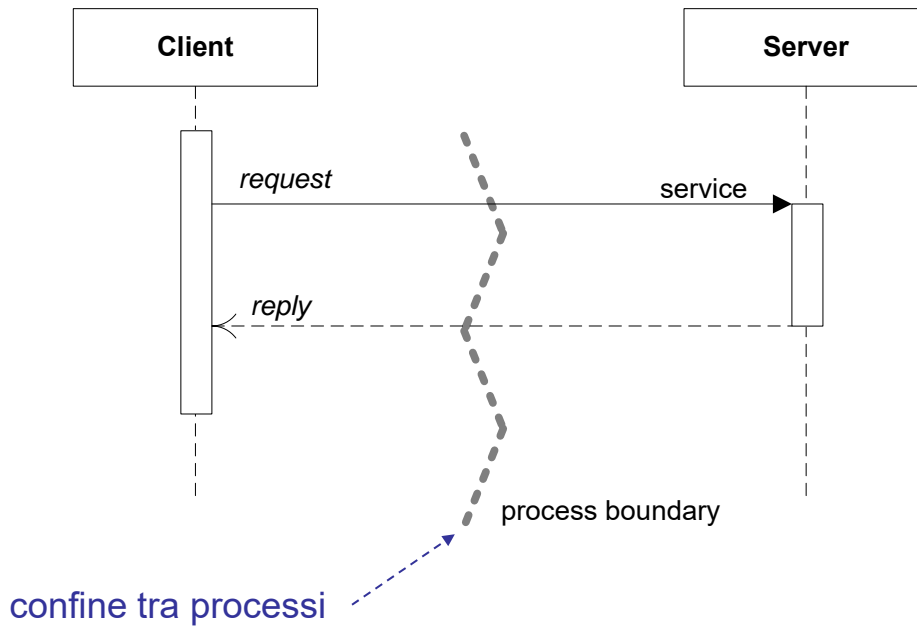
### □ Soluzione (dinamica)

- i client interagiscono invocando i servizi dai server, tramite i relativi protocolli richiesta-risposta
- i server sono responsabili di erogare dei servizi specifici, su richiesta dei client
- le interazioni tra client e server avvengono sulla base del connettore di tipo richiesta-risposta per il servizio di interesse



# Stile client-server

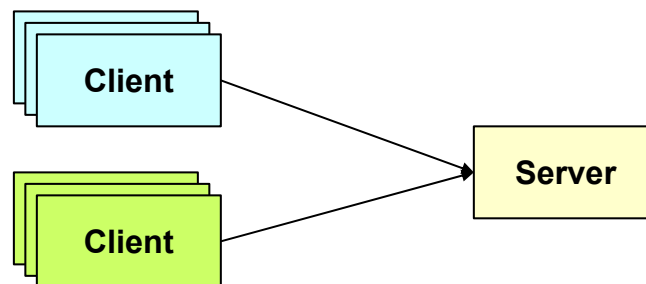
## ▣ Soluzione (dinamica)



# Stile client-server

## ▣ Soluzione (dinamica)

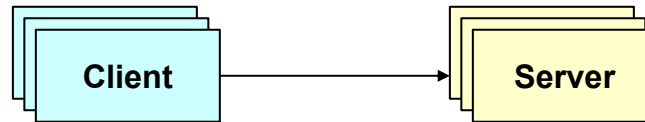
- un server può essere acceduto in modo concorrente da molti client (indipendenti tra di loro e di più tipologie)
- i server sono indipendenti dai loro client





## Stile client-server

- Esistono numerose varianti dell'architettura client- server
  - un servizio può essere erogato da un singolo server oppure può essere replicato ed erogato da molti server



- alcuni componenti possono agire sia da client che da server (di servizi differenti)



## Stile client-server

- **Discussione**
  - l'accesso alle risorse e ai servizi condivisi avviene in genere in uno o in pochi server, collocati e governati centralmente
    - sostiene la possibilità di controllare l'accesso alle risorse e la qualità dei servizi
  - le risorse e i servizi possono essere distribuiti e replicati su più server e più nodi (fisici o virtuali)
    - sostiene la disponibilità e la scalabilità
  - la centralizzazione delle risorse e dei servizi consente una loro modifica in un numero limitato di locazioni
    - sostiene la modificabilità di risorse e servizi



## Stile client-server

- Esempi di uso
  - molte applicazioni e servizi di Internet, file system distribuiti, basi di dati,...
  
- Conseguenze – in prima approssimazione
  - 😊 condivisione di risorse, centralizzazione di elaborazioni complesse e di dati sensibili
  - 😊 possibilità di sostenere prestazioni, scalabilità e disponibilità
  - 😞 overhead della comunicazione, sicurezza, ...
  - 😞 un server non replicato potrebbe essere un collo di bottiglia per prestazioni e scalabilità e un punto di fallimento singolo per la disponibilità



## - Pattern di deployment

- Nello stile architetturale client-server, i client e i server sono dei componenti software (e non hardware)
  - uno stile di solito adottato nella *vista funzionale* o nella *vista della concorrenza*
  
- L'applicazione dello stile client-server nella *vista di deployment* è descritto dai *pattern di deployment*
  - ciascuno di questi pattern rappresenta un modo diverso per allocare i processi di interesse su uno o più nodi



## Pattern di deployment e livelli (tier)

- I pattern di deployment sono basati sulla nozione di livello
  - un **livello** (*tier*) è un nodo o un gruppo di nodi su cui sono rilasciati alcuni processi di un sistema client-server
  - il sistema è organizzato come una sequenza di livelli
    - ciascun livello eroga servizi nei confronti del livello precedente e richiede servizi nei confronti del livello successivo
  - i livelli sono di solito organizzati in base al tipo di servizio (responsabilità) che forniscono

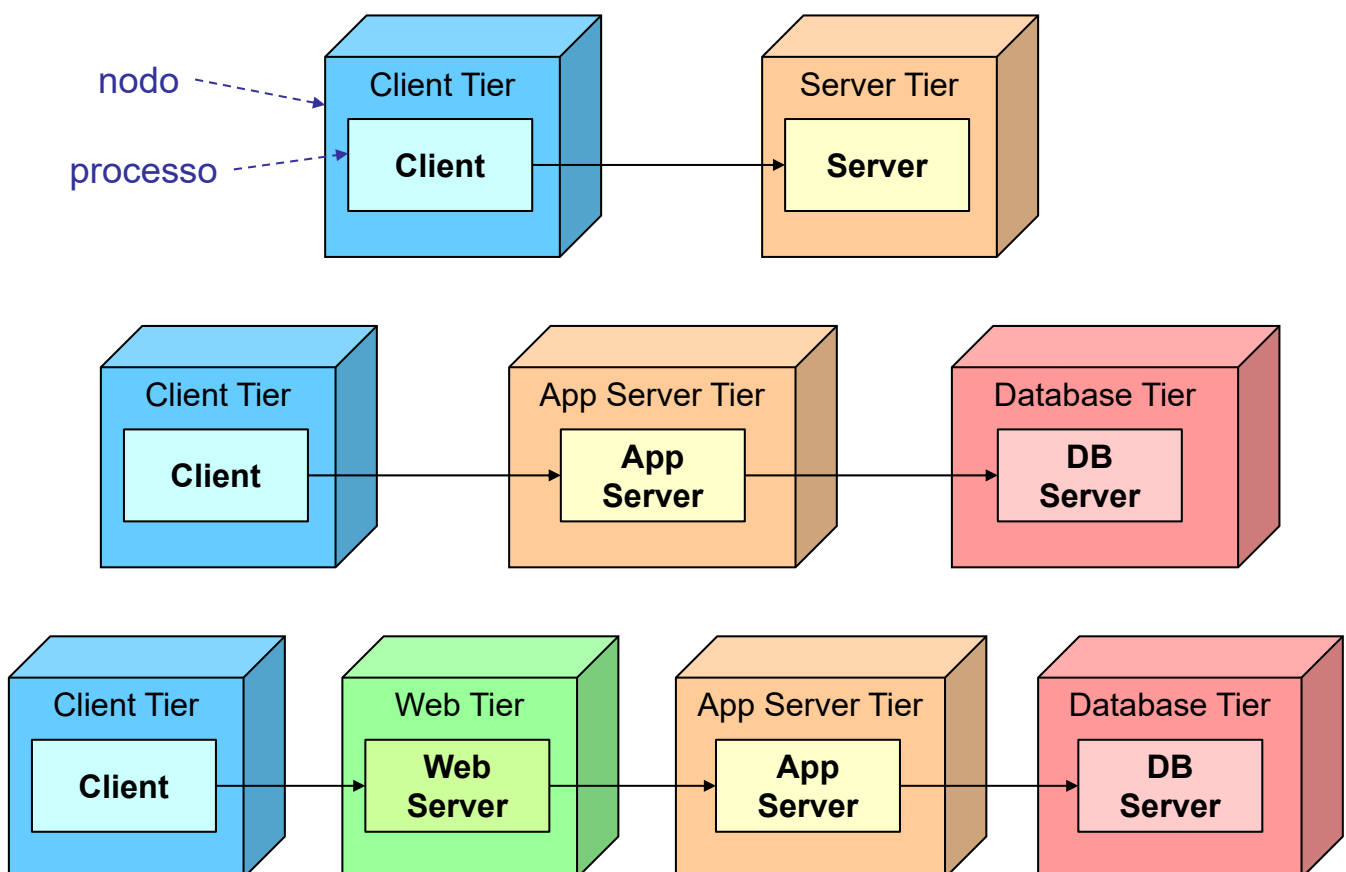
15

Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Pattern di deployment comuni



16

Stili client-server e peer-to-peer

Luca Cabibbo ASW





## Livelli e strati

- Le responsabilità funzionali di un sistema client-server, rilasciato su più livelli, sono spesso organizzate a strati
  - la vista **funzionale** è basata su un'architettura **a strati**
    - gli strati sono organizzati in base al livello di astrazione
  - la vista di **deployment** è basata su un'architettura **a livelli**
    - i livelli sono organizzati in base al tipo di servizio (responsabilità) che forniscono
    - il software in ciascun livello è spesso organizzato internamente a strati
- Discutiamo ora alcune corrispondenze comuni tra livelli e strati nell'architettura client-server
  - ipotesi: le responsabilità principali del sistema sono presentazione, logica di business e gestione dei dati

17

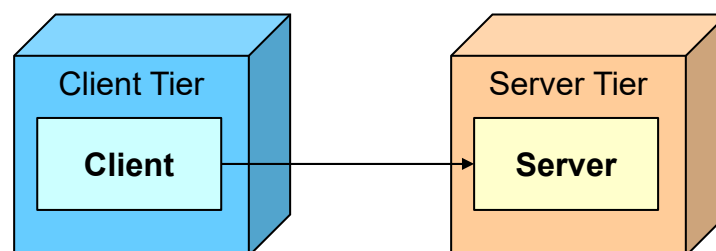
Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Pattern client-server a due livelli

- Pattern client-server a due livelli



- popolare negli anni '80
- due varianti principali
  - modello **thin-client**
  - modello **thick-client** (o **fat-client**)

18

Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Pattern client-server a due livelli

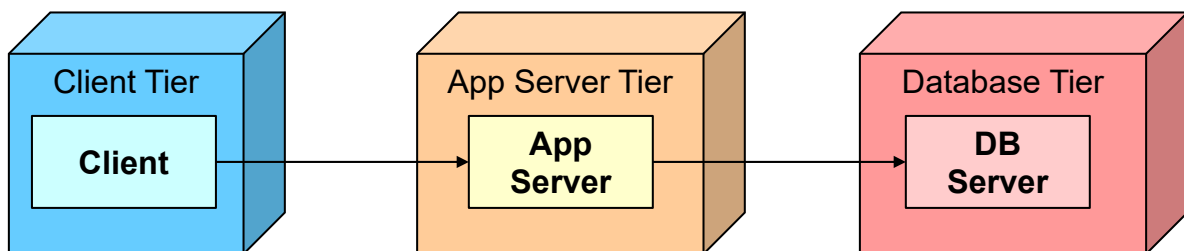
### Conseguenze

- 😊 negli anni '80, il modello thin-client è stata una soluzione per la migrazione dai sistemi legacy (basati su mainframe) a un'architettura distribuita
- 😊 il modello thick-client ha saputo utilizzare l'aumentata potenza di calcolo dei PC degli anni '80
- 😊 possibili più client – di tipo diverso
- 😊 semplice gestire l'aggiornamento dei server
- 😞 è in genere poco scalabile
- 😞 gestire l'aggiornamento dei client può essere problematico



## Pattern client-server a tre livelli

### Pattern client-server a tre livelli



- popolare negli anni '90



## Pattern client-server a tre livelli

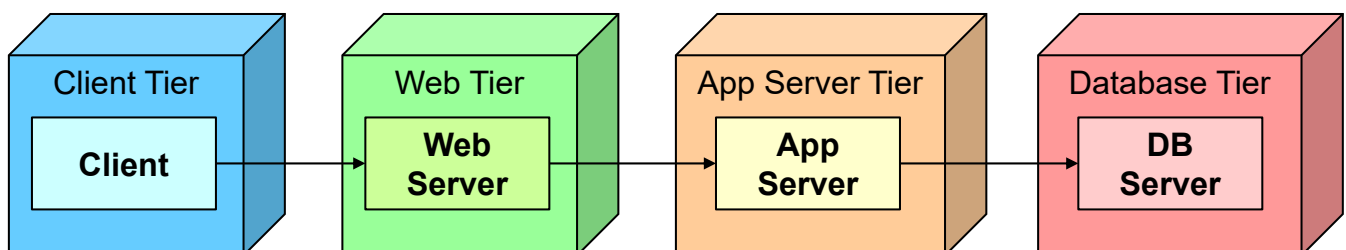
### Conseguenze

- 😊 consente una migliore distribuzione del carico di elaborazione e migliori prestazioni – questo può compensare il maggior overhead nella comunicazione
- 😊 supporto per disponibilità e scalabilità – il livello intermedio può essere un cluster di nodi server
- 😊 più semplice da gestire – rispetto al modello fat-client
- 😞 maggior complessità e maggior overhead nella comunicazione
- 😞 è più difficile decidere come allocare le responsabilità ai diversi livelli – questa decisione è complessa e costosa da cambiare
- 😞 può essere problematico gestire l'aggiornamento dei client



## Pattern client-server a quattro livelli

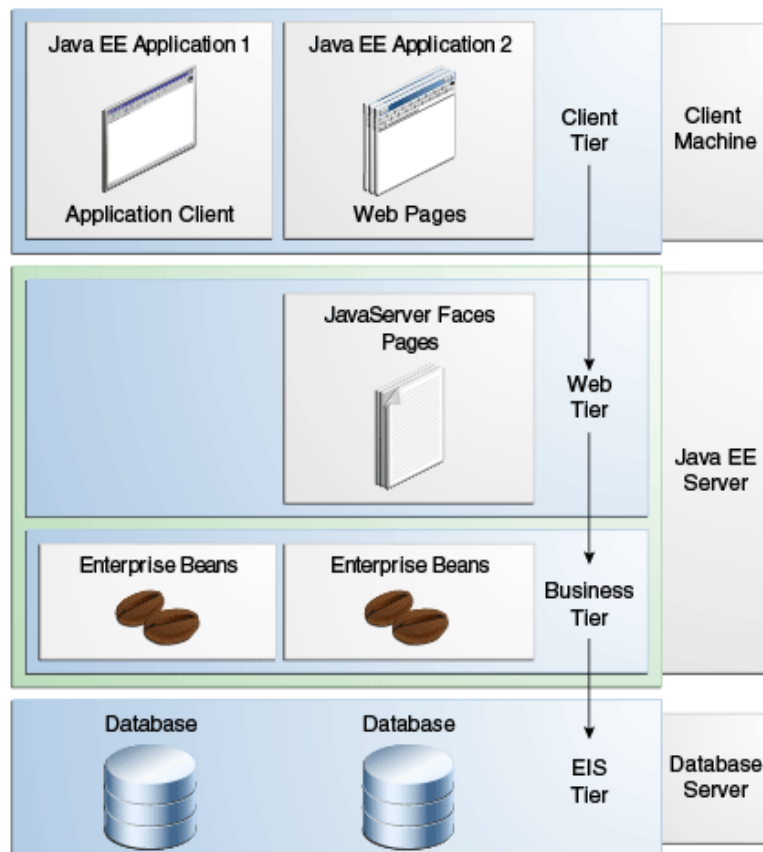
### Pattern client-server a quattro livelli



- un pattern popolare negli anni '00 (e ancora oggi), grazie alla disponibilità di opportune piattaforme e framework



## Esempio – piattaforma Java EE



23

Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Pattern client-server a quattro livelli

### Conseguenze

- 😊 architettura flessibile, che sostiene prestazioni, disponibilità e scalabilità
- 😊 sostiene la sicurezza di applicazioni che devono essere accedute pubblicamente
- 😊 semplice gestire l'aggiornamento di tutte le responsabilità
- 😞 la complessità è ancora maggiore

24

Stili client-server e peer-to-peer

Luca Cabibbo ASW



## - Opzioni per il client

- Alcune varianti dello stile client-server fanno riferimento soprattutto alle caratteristiche della presentazione e dei client
  - web application
  - rich client application
  - rich internet application
  - mobile application
- queste varianti differiscono anche nelle responsabilità lato server e nelle modalità di interazione tra client e server



## Web application



- *Web application*
  - il client è un browser web eseguito nel nodo client di un utente
  - il client comunica con il server web mediante HTTP
  - da considerare quando
    - l'applicazione deve essere accessibile su Internet
    - è sufficiente un'interfaccia utente semplice – non è richiesta un'interfaccia utente “ricca”
    - si vuole portabilità dell'interfaccia utente
    - non si vuole dover installare nei client nessun software specifico per l'applicazione
    - si vogliono usare solo un minimo di risorse lato client



# Rich Client Application



## □ *Rich Client Application*

- il client (specifico per l'applicazione) viene installato ed eseguito sul nodo client dell'utente
- il client comunica con i servizi remoti dell'applicazione mediante un protocollo specifico
- l'interfaccia utente può fornire un'esperienza per l'utente ricca, altamente interattiva, e con elevate prestazioni
- da considerare quando
  - si vuole un'interfaccia utente "ricca" e altamente interattiva
  - è possibile installare nei nodi client del software specifico per l'applicazione
  - si vogliono sfruttare le risorse del nodo client
  - si vuole consentire l'uso dell'applicazione anche in modo disconnesso o con connettività intermittente



# Rich Internet Application



## □ *Rich Internet Application*

- il client (specifico per l'applicazione) può essere scaricato da un server web ed eseguito da un utente nel suo browser web
- sono applicazioni più complesse delle applicazioni web
- da considerare quando
  - si vuole un'interfaccia utente "ricca" e altamente interattiva
  - non si vuole dover installare nei client nessun software specifico per l'applicazione
  - si vogliono poter eseguire delle elaborazioni lato client
  - lato client è accettabile un accesso limitato alle risorse locali



## □ *Mobile application*

- un'applicazione (app) mobile viene eseguita su un dispositivo mobile
- le app mobili operano spesso come client di servizi remoti, mediante una connessione mobile
- le app mobili sono spesso realizzate come rich client application oppure come web application
- da considerare quando
  - si vuole eseguire l'applicazione su un dispositivo mobile
  - si vuole consentire l'uso dell'applicazione anche in caso di connettività poco affidabile o intermittente
  - è accettabile eseguire delle elaborazioni lato client, anche se con risorse limitate



## - Servizi, interfacce (API) e protocolli

- Nello stile client-server, ciascun servizio viene “pubblicato” mediante un'*interfaccia* – chiamata un'*API* (*application programming interface*)
  - ogni API è basata su un connettore, che definisce il protocollo e il formato delle richieste e delle risposte scambiate nelle interazioni tra il server e i suoi client
    - l'applicazione dello stile client-server richiede sempre di definire, per ciascun servizio, la sua interfaccia e il relativo protocollo
  - ci sono numerose possibilità su cui basare il protocollo



## - Interfacce fornite e interfacce richieste

- Nello stile client-server
  - i servizi offerti da un server costituiscono l'*interfaccia fornita* del server
  - i servizi richiesti da un client costituiscono l'*interfaccia richiesta* del client
  
- Inoltre
  - un server può avere più interfacce fornite
  - un client può avere più interfacce richieste
  - i server intermedi possono avere sia interfacce fornite che interfacce richieste



## - Stateless e stateful

- L'essere "stateful" oppure "stateless" è una caratterizzazione importante dei servizi e dei server che li erogano
  - riguarda la loro necessità o meno di gestire uno stato
  - una possibile definizione relativa ai **servizi**
    - un **servizio** è *stateful* se il suo comportamento dipende, in qualche modo, da quanto è successo nelle sue esecuzioni precedenti
    - un **servizio** è *stateless* se il suo comportamento non dipende da quanto è successo nelle sue esecuzioni precedenti
  - lo **stato** di cui si parla si riferisce alla necessità, per un sistema software nella sua interezza, di memorizzare queste informazioni di stato relative a un servizio offerto dal sistema





## Stateless e stateful

- Lo stato di cui si parla può riguardare
  - lo stato **persistente** generato dall'esecuzione delle operazioni del servizio
  - lo stato delle **conversazioni** (o **sessioni**) con i client che accedono al servizio
  - poiché molti servizi utili richiedono la gestione di uno stato persistente, viene spesso usata anche la seguente definizione alternativa
    - un servizio è **stateful** se l'esecuzione di un'operazione dipende dallo stato della conversazione/sessione con il client che ha richiesto l'operazione – altrimenti il servizio è **stateless**



## Stateless e stateful

- La proprietà di essere “stateful” o “stateless” si può riferire anche ai server usati per erogare i servizi
  - una possibile definizione relativa ai **server**
    - un **server** è **stateful** se scrive oppure legge localmente informazioni circa il proprio stato che vengono mantenute da un'esecuzione precedente a un'esecuzione successiva delle proprie operazioni
    - un **server** è **stateless** se non legge né scrive localmente informazioni circa il proprio stato
  - in questo caso, lo **stato** di cui si parla si riferisce alla necessità, per il server, di memorizzare localmente queste informazioni di stato



## Stateless e stateful

- L'essere "stateful" o "stateless" è una caratteristica importante
  - i server stateless possono essere replicati facilmente
    - per sostenere disponibilità e scalabilità
  - se un servizio è stateful, allora il sistema deve gestire (da qualche parte, di solito in uno o più dei suoi server) lo stato del servizio
    - questo ha impatto sul livello di accoppiamento e sulla scalabilità del sistema

35

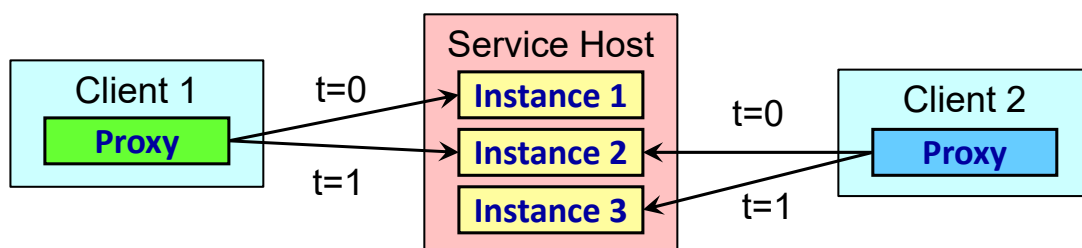
Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Servizi e server stateless

- Un *servizio stateless* (rispetto alle sessioni) può essere implementato da un *server stateless*
  - adeguato quando la gestione di una richiesta è indipendente dalla storia delle richieste precedenti
  - ogni richiesta può essere gestita indipendentemente dalle altre richieste da parte dello stesso client
  - è possibile fare pooling delle istanze del server – le risorse di ogni server possono essere condivise tra i diversi client
    - impatto positivo sulla scalabilità



36

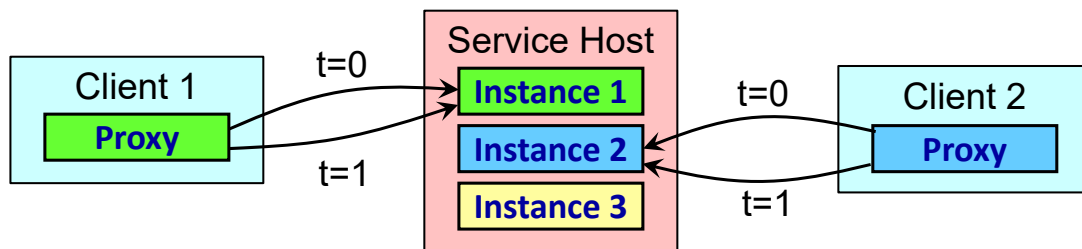
Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Servizi e server stateful

- Un *servizio stateful* (rispetto alle sessioni) può essere implementato da un *server stateful*
  - utile quando la gestione di una richiesta deve poter dipendere dalla storia delle richieste precedenti da parte di quel client
  - una possibilità è che ciascuna istanza di server gestisca uno o più client, per tutta la durata delle loro sessioni
    - impatto negativo sulla scalabilità



37

Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Implementazione di servizi stateful

- Come gestire il caso di un servizio stateful?
  - il sistema, da qualche parte, deve memorizzare lo stato di quel servizio
  - una soluzione comune consiste nell'utilizzare più server per implementare (complessivamente) il servizio
    - uno o più server stateful, usati esclusivamente per la memorizzazione di dati – ad es., un database server o una cache
    - uno o più server stateless, per la logica di business

38

Stili client-server e peer-to-peer

Luca Cabibbo ASW



## Implementazione di servizi stateful

- Alcune opzioni per implementare un servizio **stateful** (rispetto alle sessioni)
  - stato delle sessioni nell'application server (AS)
  - stato delle sessioni nei client
  - stato delle sessioni in una base di dati
  - stato delle sessioni in una cache (in memoria)
  - le diverse opzioni hanno caratteristiche e conseguenze diverse



## - Ulteriori considerazioni e varianti

- Nell'architettura client-server
  - l'invocazione dei servizi è di solito sincrona
    - il client effettua una richiesta e rimane in attesa di una risposta
    - tuttavia, alcuni servizi possono essere invocati in modo "asincrono"
  - l'interazione è asimmetrica
    - è iniziata dai client, che deve conoscere l'identità del server
    - tuttavia, è possibile che un server possa iniziare delle azioni nei confronti dei suoi client – sulla base di meccanismi di notifica o callback



## - Note terminologiche

- Alcuni modi comuni di utilizzare i termini “client” e “server”
  - nello stile architetturale client-server, sono usati per descrivere l'**organizzazione fondamentale di un sistema distribuito**
  - talvolta vengono usati per descrivere, localmente, la **relazione tra una coppia di elementi architetturali**
  - altre volte vengono usati per descrivere il ruolo rivestito da una coppia di elementi nell'ambito di **una singola interazione**
  - altre volte ancora vengono usati per indicare elementi hardware anziché elementi software runtime



## Note terminologiche

- Anche il termine “servizio” viene spesso usato con significati leggermente diversi
  - nello stile architetturale client-server
    - un servizio rappresenta un insieme di funzionalità
    - i servizi sono erogati da componenti server e fruiti da componenti client
  - nell'architettura a servizi
    - un servizio è un componente software o un'applicazione in grado di erogare un certo insieme di funzionalità



## \* Stile peer-to-peer

- Lo stile peer-to-peer (P2P) organizza un sistema distribuito in modo decentralizzato, come una rete di nodi (peer) che condividono le proprie risorse di calcolo
  - i sistemi client-server offrono una capacità computazionale limitata – perché forniscono l'accesso alle risorse gestite da uno o pochi server centralizzati
  - i sistemi P2P possono invece offrire una capacità computazionale molto alta (o illimitata) – perché possono fornire l'accesso alle risorse gestite da tutti i nodi di una rete – una rete aziendale o anche Internet



## Esempi di sistemi peer-to-peer

- Alcuni esempi di sistemi peer-to-peer
  - condivisione di contenuti (file sharing)
    - ad es., musica e distribuzioni di OS
  - comunicazione e collaborazione
    - ad es., chat
  - calcolo distribuito
    - ad es., seti@home, grid, cloud
  - sistemi per la gestione e la replicazione di dati distribuiti
    - ad es., basi di dati non relazionali e blockchain



# Caratteristiche dei sistemi peer-to-peer

- I sistemi P2P hanno anche la capacità di trattare l'instabilità come la norma
  - sono in grado di riorganizzarsi autonomamente – anche a fronte di peer che si connettono e sconnettono autonomamente al/dal sistema
  - sfruttano la replicazione delle risorse per sostenere disponibilità e scalabilità



# Stile peer-to-peer

- Contesto
  - ci sono diverse entità distribuite – ciascuna con le proprie risorse computazionali
  - queste entità devono poter cooperare e collaborare per fornire dei servizi a una comunità distribuita di utenti
  - ogni entità è considerata ugualmente importante nel poter avviare interazioni con le altre entità
- Problema
  - si vogliono organizzare un insieme di entità computazionali distribuite affinché possano condividere i loro servizi e risorse
  - queste entità sono tra di loro “equivalenti” o “pari”
  - si vogliono connettere queste entità sulla base di un protocollo comune
  - si vogliono sostenere scalabilità e disponibilità



## Stile peer-to-peer

### □ Soluzione

- organizza il sistema (o servizio) come un insieme di componenti *peer* che interagiscono tra di loro come “pari”
  - ogni peer è un componente software (un processo) – “peer” viene talvolta usato anche per indicare un nodo per un peer
  - i peer sono tutti ugualmente importanti – nessun peer dovrebbe essere critico per la salute del sistema o servizio
- la comunicazione avviene direttamente da peer a peer (*peer to peer*), sulla base di interazioni richiesta-risposta
  - ogni peer fornisce e richiede servizi simili, utilizzando uno stesso protocollo
  - ogni peer può interagire con ogni altro peer – le interazioni possono essere avviate da ciascun peer
- talvolta l’interazione consiste solo nell’inoltro di dati



## Stile peer-to-peer

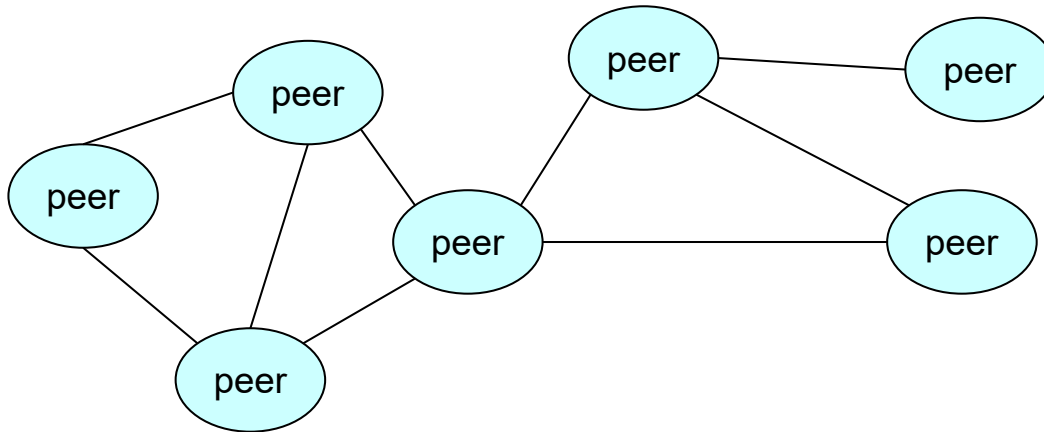
- Lo stile peer-to-peer riflette i meccanismi bidirezionali che possono sussistere tra due o più entità che interagiscono tra loro come pari
  - ciascun peer offre e consuma servizi simili
  - i servizi sono relativi alla gestione delle risorse che si vogliono condividere
  - i connettori peer-to-peer implicano delle interazioni bidirezionali





## Rete peer-to-peer

- Una **rete peer-to-peer** è l'insieme dei peer di un sistema P2P e dei collegamenti tra di essi
  - la rete può variare dinamicamente nel tempo



## - P2P – Esempi

- Ecco alcune applicazioni di esempio dello stile peer-to-peer
  - file sharing
    - ogni peer condivide la propria capacità di memorizzare file
    - un peer può
      - aggiungere localmente un file
      - cercare un file su altri peer
      - scaricare un file (o un frammento di file) da altri peer
      - cancellare localmente un file
    - ogni peer può eseguire o supportare operazioni richieste da altri peer



## P2P – Esempi

- Ecco alcune applicazioni di esempio dello stile peer-to-peer
  - esecuzione di task in un cluster
    - ogni peer del cluster condivide la propria capacità di eseguire dei task – ad es., servizi, job o container
    - un peer può
      - installare localmente un task
      - eseguire un task richiesto da un altro peer
      - chiedere l'esecuzione di un task ad altri peer
    - ogni peer può eseguire o supportare operazioni richieste da altri peer



## P2P – Esempi

- Ecco alcune applicazioni di esempio dello stile peer-to-peer
  - gestione di una base di dati distribuita – ad es., un insieme di coppie chiave-valore (K-V)
    - ogni peer condivide la propria capacità di memorizzazione
    - ciascun peer memorizza un sottoinsieme delle coppie K-V
    - ciascun peer può eseguire operazioni CRUD sulla base di dati
      - inserire una nuova coppia, cercare il valore associato a una chiave, aggiornare il valore associato a una chiave, cancellare una coppia
    - per eseguire un'operazione richiesta, un peer può interagire anche con altri peer



## - P2P in reti non controllate

- Lo stile peer-to-peer è stato spesso utilizzato in *reti P2P non controllate*
  - ogni peer è gestito da un utente – che è libero di collegarsi alla rete P2P, eseguire le operazioni di interesse, e poi abbandonare la rete P2P in qualunque momento
  - non è possibile fornire garanzie complessive sull'effettiva condivisione delle risorse



## - P2P in reti controllate

- Lo stile peer-to-peer può essere usato anche in *reti P2P controllate* – come un insieme di nodi del data center di un'organizzazione
  - i peer sono controllati dall'organizzazione
  - i nodi si disconnettono dalla rete (solo) in caso di guasti
  - è possibile fornire garanzie sull'effettiva condivisione delle risorse
    - l'organizzazione può controllare l'aggiunta e la rimozione di nodi dalla rete – ragionando sul livello di replicazione delle risorse e sulla loro localizzazione



## - Scenari

### □ Avvio

- un peer si connette alla rete P2P
  - per scoprire e conoscere altri peer con cui poter interagire
  - per comunicare la propria presenza – ma anche il proprio insieme iniziale di contenuti o le proprie capacità
- la possibilità di aggiungere dinamicamente peer alla rete P2P favorisce la scalabilità del servizio



## Scenari

### □ Richiesta di servizi (accesso a risorse)

- un peer può poi avviare delle interazioni per ottenere dei servizi – chiedendo questi servizi ai peer che conosce
- un servizio comune è la *ricerca* di una risorsa – per trovare uno o più peer che possiedono la risorsa cercata
  - un peer che riceve la richiesta per una ricerca, la effettua localmente – ma la può anche propagare ad altri peer



## Scenari

- Richiesta di servizi (accesso a risorse)
  - un peer può poi avviare delle interazioni per ottenere dei servizi – chiedendo questi servizi ai peer che conosce
  - un altro servizio comune è il *consumo* di una risorsa
    - una richiesta di questo tipo può essere diretta a un solo peer, tra quelli che possiedono la risorsa
    - in alcuni casi un peer può consumare la risorsa da più peer, in modo concorrente
    - il peer che riceve una risorsa la può anche memorizzare localmente, per poi poterla condividere



## Scenari

- Richiesta di servizi (accesso a risorse)
  - un peer può poi avviare delle interazioni per ottenere dei servizi – chiedendo questi servizi ai peer che conosce
  - un altro possibile servizio è l'*aggiunta* di una risorsa
    - ci sono diverse possibilità
      - la risorsa viene aggiunta solo localmente dal peer che riceve la richiesta
      - oppure, il peer gira la richiesta a un altro peer
      - oppure, il peer aggiunge la risorsa localmente, e poi inoltra la richiesta ad altri peer, per fare replicare la risorsa



## Scenari

### □ Rimozione di peer

- è possibile che un peer (con le sue risorse) venga rimosso dalle rete P2P
- la rimozione di un peer non dovrebbe compromettere la disponibilità dei servizi offerti dalla rete peer-to-peer
  - questo avviene se i diversi peer hanno capacità sovrapponibili e se le risorse sono replicate su più peer



## Scenari

### □ Super-nodi

- alcune reti P2P hanno dei peer specializzati (super-nodi o super-peer) che forniscono servizi comuni o speciali agli altri peer



## Discussione

- Alcune conseguenze dello stile peer-to-peer
  - 😊 consente la condivisione di risorse tra un gran numero di nodi
  - 😊 può sostenere disponibilità, scalabilità, distribuzione del carico e prestazioni – soprattutto nelle reti controllate
  - 😞 a causa della forte decentralizzazione, è più complesso (o impossibile) gestire la sicurezza, la consistenza dei dati, gestire e controllare la disponibilità dei dati e dei servizi, effettuare backup e recovery
  - 😞 in molti casi è difficile fornire garanzie di qualità – soprattutto nelle reti non controllate



## Esempio: basi di dati distribuite



- Alcuni sistemi per la gestione di dati distribuiti (ad es., i database NoSQL o le object cache) in un cluster usano una combinazione dei seguenti meccanismi per sostenere scalabilità e disponibilità
  - replicazione dei dati su più nodi del cluster
    - replicazione master-slave (non è P2P), oppure
    - replicazione peer-to-peer – una soluzione spesso più efficace per la propagazione degli aggiornamenti
  - distribuzione (sharding) dei dati sui nodi del cluster
  - è anche possibile combinare replicazione e distribuzione dei dati



## - Note terminologiche

- Alcuni modi comuni di utilizzare i termini “peer” e “peer-to-peer”
  - nello stile architetturale peer-to-peer, sono usati per descrivere l'**organizzazione fondamentale di un sistema distribuito**
  - talvolta vengono usati per descrivere, localmente, la **relazione tra una coppia di elementi architeturali**
    - due elementi sono in una relazione peer-to-peer se ciascun elemento può iniziare interazioni nei confronti dell'altro
    - questo non sempre vuole dire che i due elementi implementano una stessa interfaccia – ad es., potrebbe voler dire solo che i due elementi comunicano con uno stesso protocollo (ad es., si scambiano messaggi) oppure anche solo che non sono in una relazione client-server



## \* Discussione

- Gli stili architeturali client-server e peer-to-peer sono alla base di molti sistemi distribuiti
  - le tecnologie sottostanti, e i relativi pattern di utilizzo, si sono evoluti nel corso degli anni per semplificare ulteriormente lo sviluppo dei sistemi distribuiti e per favorire le loro qualità e la loro interoperabilità





# Discussione

- Alcuni sistemi sono basati su una combinazione degli stili C-S e P2P – che vengono applicati separatamente in parti distinte del sistema
  - ad es., un sistema che implementa un servizio mediante un insieme di server che condividono tra loro le proprie risorse computazionali – e che offre questo servizio ai suoi client
  - è il caso di alcuni sistemi per la gestione di dati distribuiti

