



Luca Cabibbo
Architettura
dei Sistemi
Software

Comunicazione asincrona

dispensa asw450
marzo 2021

*I'll send an S.O.S. to the world.
I hope that someone gets my
message in a bottle.
Yeah.*

The Police

1

Comunicazione asincrona

Luca Cabibbo ASW



- Riferimenti

- ❑ Luca Cabibbo. **Architettura del Software: Strutture e Qualità**. Edizioni Efestò, 2021.
 - Capitolo 25, **Comunicazione asincrona**
- ❑ Hohpe, G. and Woolf, B. **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. Addison-Wesley, 2004.
- ❑ [POSA4] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt. **Pattern-Oriented Software Architecture (vol. 4): A Pattern Language for Distributed Computing**. John Wiley & Sons, 2007
- ❑ Richardson, C. **Microservices Patterns: With examples in Java**. Manning, 2019.
- ❑ Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. **Distributed Systems: Concepts and Design**, fifth edition. Pearson, 2012.

2

Comunicazione asincrona

Luca Cabibbo ASW



- Obiettivi e argomenti

□ Obiettivi

- introdurre la comunicazione asincrona – basata sullo scambio di messaggi e la notifica di eventi, in modo indiretto e asincrono
- discutere la semantica della comunicazione asincrona

□ Argomenti

- introduzione alla comunicazione asincrona
- comunicazione asincrona
- semantica della comunicazione asincrona
- utilizzo di messaggi nelle applicazioni
- discussione



* Introduzione alla comunicazione asincrona

□ La comunicazione asincrona è un altro stile fondamentale di comunicazione nei sistemi distribuiti

- è un'astrazione di programmazione distribuita basata sullo scambio di messaggi, che consente a un componente (produttore) di inviare un messaggio, in modo indiretto e asincrono, a uno o più componenti (consumatori), affinché questi possano elaborare il messaggio
- una modalità di comunicazione offerta dai *message broker*
- in pratica, esistono diverse forme specifiche di comunicazione asincrona – qui ci concentriamo soprattutto sulle idee di base comuni, ma discutiamo anche alcune variazioni e differenze



Sulla comunicazione sincrona

- L'**invocazione remota** è una forma di comunicazione **sincrona**
 - un client invoca un'operazione offerta da un server per uno specifico servizio – direttamente o mediante un broker
 - durante l'esecuzione dell'operazione richiesta, il client rimane in attesa di una risposta
 - fornisce una sincronizzazione tra client e server

- Tuttavia, non sempre questa modalità di comunicazione è necessaria o opportuna
 - in alcuni casi, un componente vuole che venga eseguita un'elaborazione – ma potrebbe non essere interessato a ottenere una risposta (o a ottenerla in modo sincrono)
 - spesso è desiderabile una modalità di interazione meno accoppiata e più flessibile
 - spesso si vogliono evitare le conseguenze negative su alcune qualità causate dalla comunicazione sincrona



Verso la comunicazione asincrona

- La comunicazione asincrona è una modalità di interazione complementare a quella sincrona
 - un ulteriore paradigma fondamentale di programmazione distribuita
 - adatta quando la comunicazione sincrona non è necessaria o non è opportuna
 - per sostenere un accoppiamento più basso e per consentire interazioni più flessibili
 - per evitare le possibili conseguenze negative della comunicazione sincrona – e per sostenere qualità come prestazioni, scalabilità e disponibilità



* Comunicazione asincrona

- La **comunicazione asincrona** è un paradigma di comunicazione per componenti distribuiti, basato sullo *scambio di messaggi*
 - i componenti distribuiti interagiscono
 - inviandosi messaggi (oppure notifiche di eventi)
 - in modo indiretto
 - in modo asincrono
 - la comunicazione asincrona è anche chiamata *messaging* (“scambiarsi messaggi”) o *comunicazione basata su messaggi*
 - è supportata dai *message broker* e da altri servizi di middleware (*middleware orientato ai messaggi*)



Sulla “asincronia”

- Attenzione, non si confonda la *comunicazione asincrona* con l'*invocazione remota asincrona*
 - l'invocazione remota asincrona è, infatti, una forma di comunicazione “sincrona”
 - la comunicazione asincrona è invece una forma di comunicazione “veramente” asincrona – che consente anche la comunicazione tra componenti con esistenza indipendente



Un'analogia

- Ecco un'analogia che illustra la differenza tra comunicazione sincrona e comunicazione asincrona
 - una forma di comunicazione sincrona è quella che avviene tra due persone in una telefonata
 - una forma di comunicazione asincrona è quella che avviene tra due persone nello scambio di messaggi – di posta elettronica, su WhatsApp oppure su un social network
 - si tratta solo di un'analogia – siamo infatti interessati alla comunicazione tra componenti software, e non tra persone



Percorso di studio

- Presentiamo la comunicazione asincrona in due momenti
 - prima discutiamo il caso di base, relativo allo scambio di **un solo messaggio** tra **una sola coppia di componenti**
 - quindi discutiamo il caso di un sistema distribuito basato sullo scambio di **molti messaggi** tra **molti componenti**



- Comunicazione asincrona di base

- Nella comunicazione asincrona, ciascuna singola interazione consente a un componente di trasmettere dei dati a un altro componente
 - un primo componente (*produttore*) ha dei dati che vuole che vengano elaborati
 - allora, il produttore prepara un *messaggio* contenente questi dati, e poi lo invia a un altro componente
 - quindi, un secondo componente (*consumatore*) riceve questo messaggio – e poi ne estrae di dati e li elabora



La comunicazione è indiretta

- La comunicazione è *indiretta* – il produttore non trasmette il messaggio direttamente a uno specifico consumatore
 - il produttore invia il messaggio a un *canale per messaggi* – specifico per quel tipo di messaggi
 - quindi, un consumatore riceve il messaggio prelevandolo da quel canale per messaggi
 - il canale per messaggi costituisce un' *indirizzazione spaziale* nella comunicazione tra produttore e consumatore

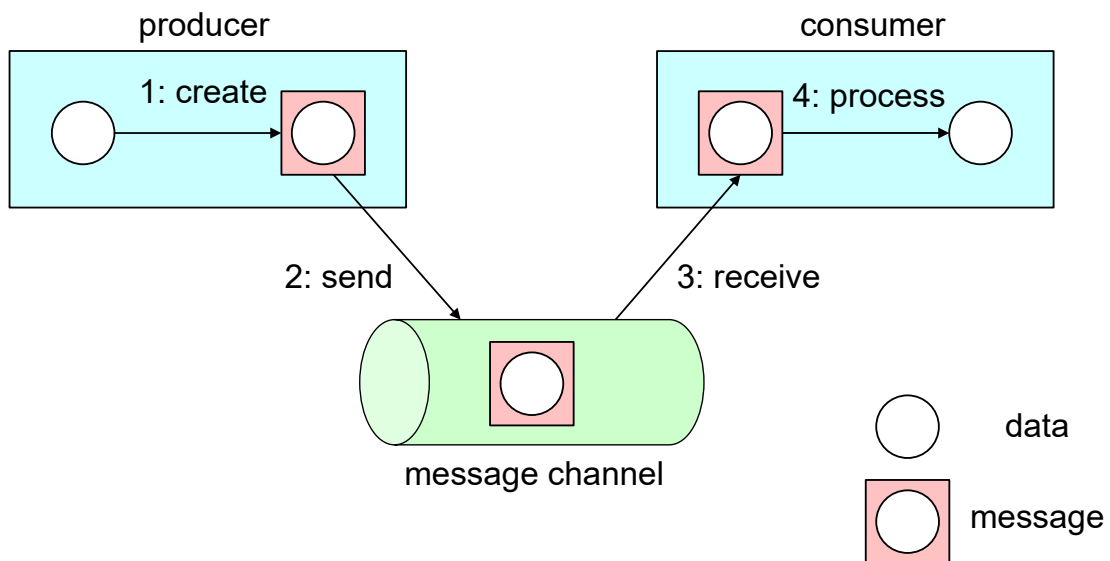


La comunicazione è asincrona

- La comunicazione è **asincrona** – l'invio e la ricezione del messaggio non sono sincronizzate
 - dopo che il produttore ha inviato il suo messaggio al canale per messaggi, prosegue nel suo lavoro (*send-and-forget*)
 - il consumatore del messaggio riceve e elabora il messaggio appena è disponibile a farlo
 - il canale per messaggi costituisce anche un'**indirizione temporale** nella comunicazione tra produttore e consumatore

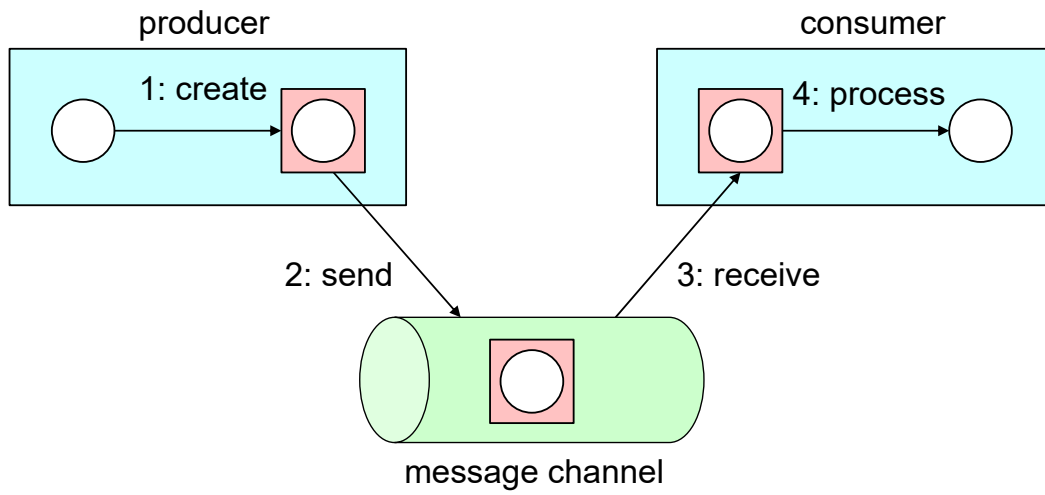


Comunicazione asincrona

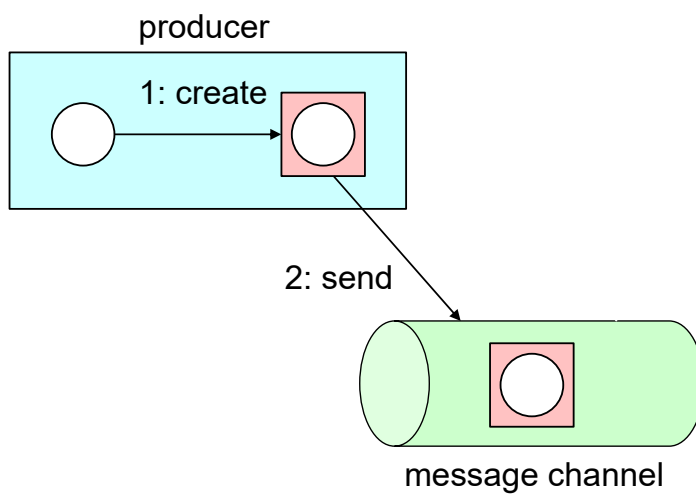




La comunicazione è indiretta

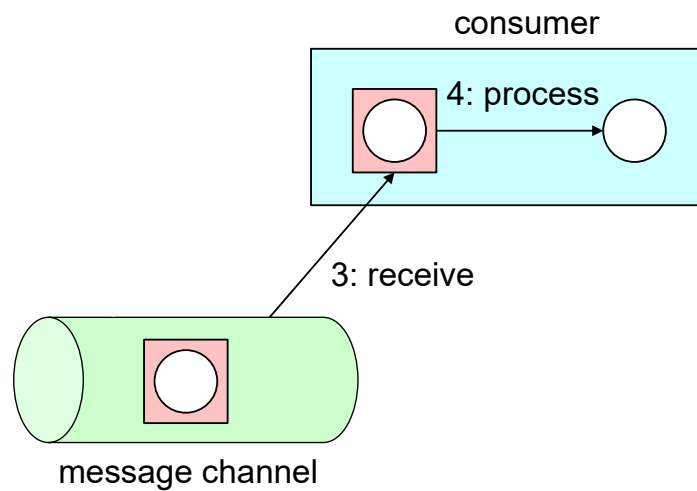


La comunicazione è asincrona





La comunicazione è asincrona



Invocazione implicita

- La comunicazione asincrona è chiamata anche *invocazione implicita*
 - quando un consumatore riceve un messaggio, lo elabora eseguendo un'operazione opportuna
 - questa operazione viene scelta **dal consumatore** del messaggio, sulla base del contenuto del messaggio
 - l'operazione eseguita non viene scelta sulla base di un'invocazione esplicita da parte del produttore del messaggio



Comunicazione asincrona e accoppiamento

- La comunicazione asincrona è caratterizzata da un **accoppiamento basso**
 - l'uso dei messaggi consente un disaccoppiamento rispetto all'interfaccia (operazioni) dei componenti
 - l'uso dei canali per messaggi consente un disaccoppiamento spaziale rispetto all'identità dei componenti
 - l'uso dei canali per messaggi consente anche un disaccoppiamento temporale tra i componenti



Su “produttore” e “consumatore”

- I termini “produttore” e “consumatore” hanno significato nell'ambito dello scambio di ciascun singolo messaggio
 - chi invia il messaggio è il **produttore** del messaggio
 - *producer* o *publisher*
 - chi riceve il messaggio è un **consumatore** del messaggio
 - *consumer* o *subscriber* o *listener*
 - in generale, però, ogni componente può sia inviare che ricevere messaggi

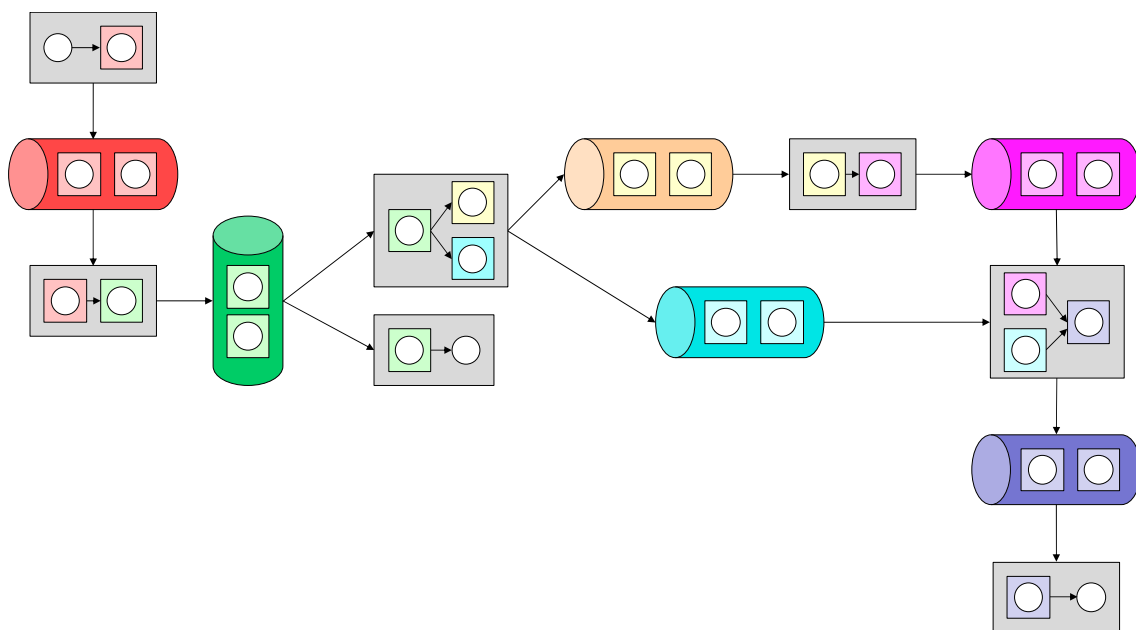


- Sistemi basati sulla comunicazione asincrona

- In un sistema distribuito basato sulla comunicazione asincrona, vengono in genere scambiati una molteplicità di messaggi, tra numerosi componenti e tramite molti canali per messaggi
 - l'invio iniziale di un messaggio "trovato" da parte di un componente può dare luogo a un'interazione che comprende lo scambio di molti messaggi tra più componenti
 - un sistema di questo tipo può essere organizzato sulla base dello stile architetturale Pipes and Filters



Sistemi e comunicazione asincrona





Sistemi e comunicazione asincrona

- Un sistema basato sullo scambio di messaggi può essere organizzato sulla base di Pipes and Filters
 - ogni componente (produttore e/o consumatore) può agire da filtro (“processor”) – può ricevere e consumare messaggi, elaborarli, e poi produrre e inviare altri messaggi
 - l’elaborazione di un messaggio iniziale “trovato” viene distribuita tra i diversi componenti del sistema ed effettuata in modo incrementale
 - i messaggi scambiati costituiscono dei flussi di dati all’interno del sistema
 - ciascun messaggio appartiene a una specifica tipologia di messaggi – ad es., **Ordini**, **Richieste di spedizione**, **Fatture e Pagamenti**



Sistemi e comunicazione asincrona

- Un sistema basato sullo scambio di messaggi può essere organizzato sulla base di Pipes and Filters
 - ciascun canale per messaggi è una pipe – che rappresenta una specifica tipologia di messaggi
 - ad es., un canale per gli **Ordini** e uno per i **Pagamenti**
 - a ciascuna tipologia di messaggi corrisponde un canale distinto – ogni canale è un “indirizzo logico” per lo scambio di messaggi di quella tipologia
 - ci possono molti componenti in grado di produrre e inviare messaggi di un certo tipo – e molti componenti in grado di ricevere e consumare messaggi di un certo tipo
 - “molti componenti” può voler dire più istanze di uno stesso tipo di componente, o anche istanze di tipi di componenti differenti



* Semantica della comunicazione asincrona

- La semantica della comunicazione asincrona riguarda diversi aspetti associati all'invio di un messaggio M da parte di un componente (produttore) – tra cui
 - quanti e quali componenti (consumatori) riceveranno il messaggio M?
 - è possibile che il messaggio M venga perso? che cosa succede se il messaggio M viene perso?
 - che cosa succede se si verifica un guasto nel componente a cui è stato assegnato il compito di elaborare il messaggio M?



- Implementazione

- La comunicazione asincrona basata sullo scambio di messaggi è supportata dai *message broker* o altri servizi di middleware
 - un'implementazione comune è un message broker realizzato come un server centralizzato per gestire i canali per messaggi e lo scambio dei messaggi
 - i canali vanno di solito configurati nel message broker prima di iniziare lo scambio dei messaggi – ma talvolta possono essere definiti in modo dinamico
 - poi, i componenti si scambiano messaggi, agendo come client nei confronti del message broker
 - il server viene di solito eseguito su un cluster



- Tipologie di canali per messaggi

- Due tipologie principali di canali per messaggi
 - **point-to-point channel** (canali punto-punto) – o *queue* (code)
 - un canale per la comunicazione “a-uno”
 - **publish-subscribe channel** (canali publish-subscribe) – *canali pub-sub* oppure *topic* (“argomenti”)
 - un canale per la comunicazione “a-molti”

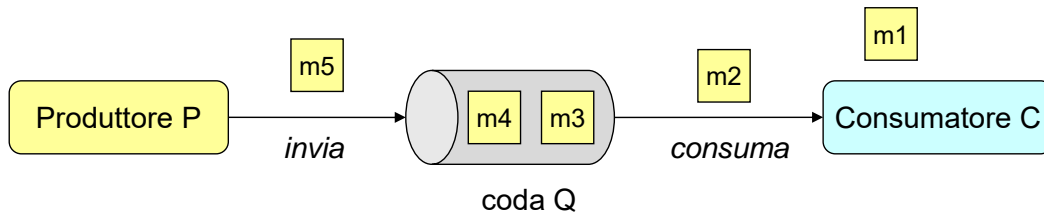


Canali e consumatori di messaggi

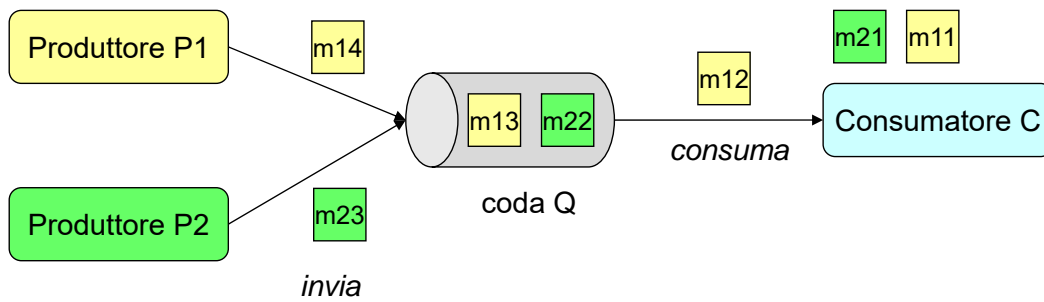
- I consumatori che sono interessati a ricevere i messaggi per un canale C devono registrarsi al canale C
 - per ciascun canale per messaggi ci possono essere più consumatori registrati a quel canale
 - quando viene inviato un messaggio M sul canale C, il message broker assegna il compito di consumare il messaggio M a uno o più tra i consumatori registrati a C
 - se C è un canale publish-subscribe, a tutti i consumatori registrati a C
 - se C è un canale point-to-point, a uno solo dei consumatori registrati a C



Canale point-to-point

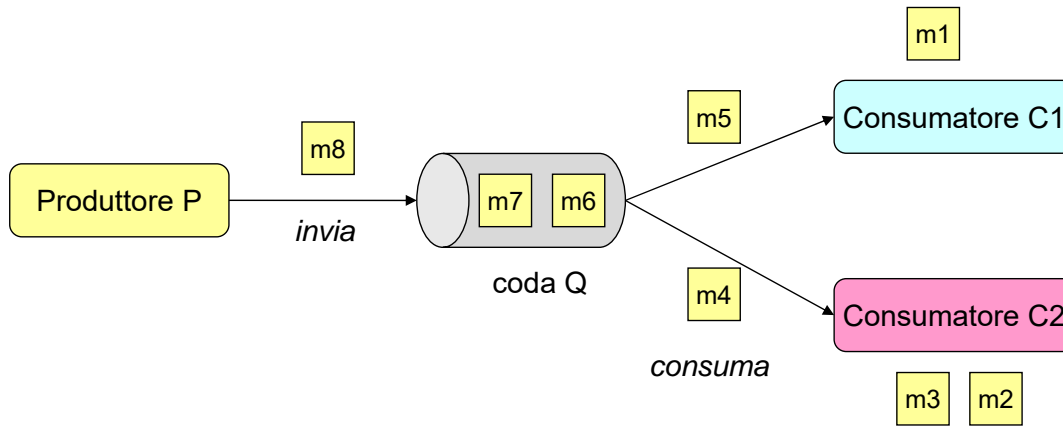


Canale point-to-point: più produttori

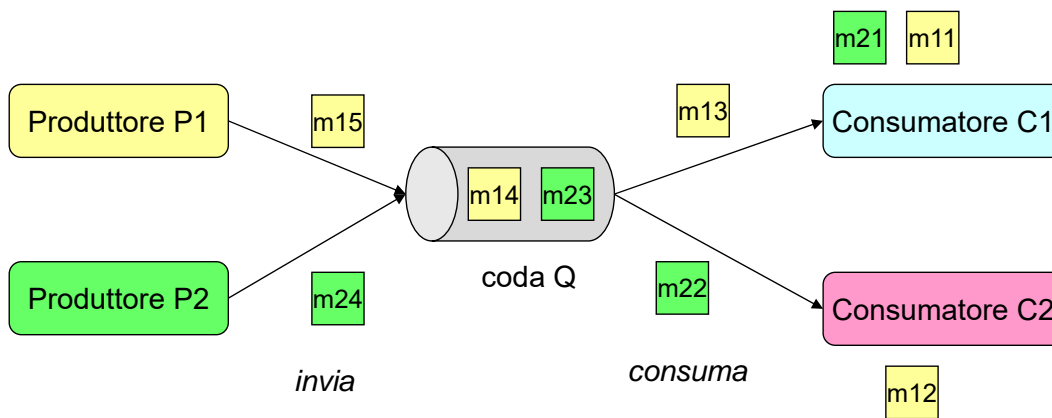




Canale point-to-point: più consumatori

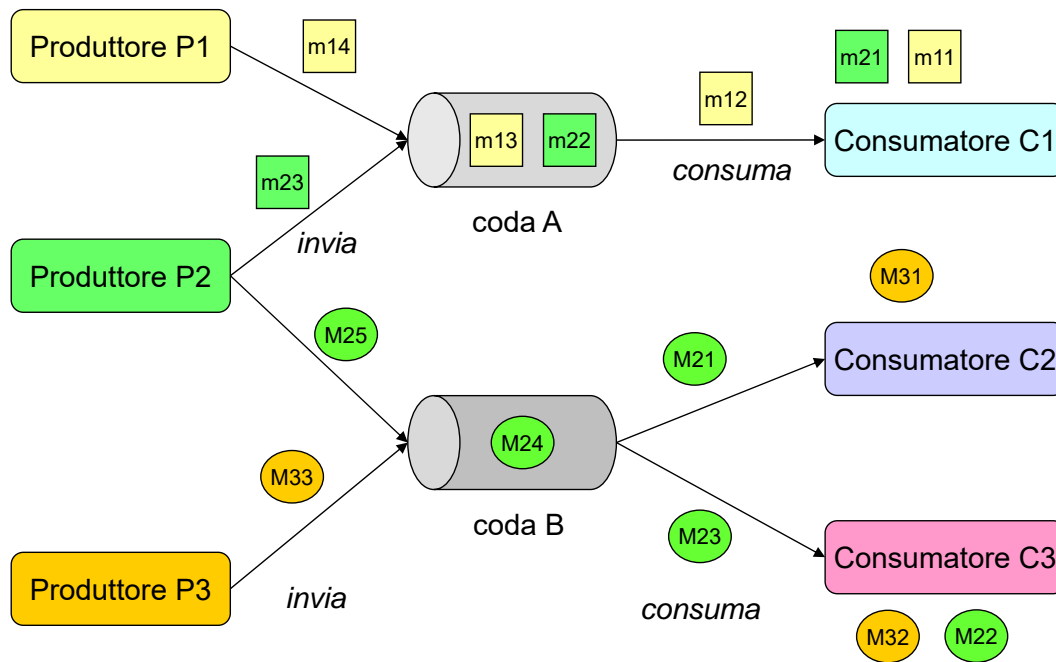


Canale point-to-point: più produttori e più consumatori

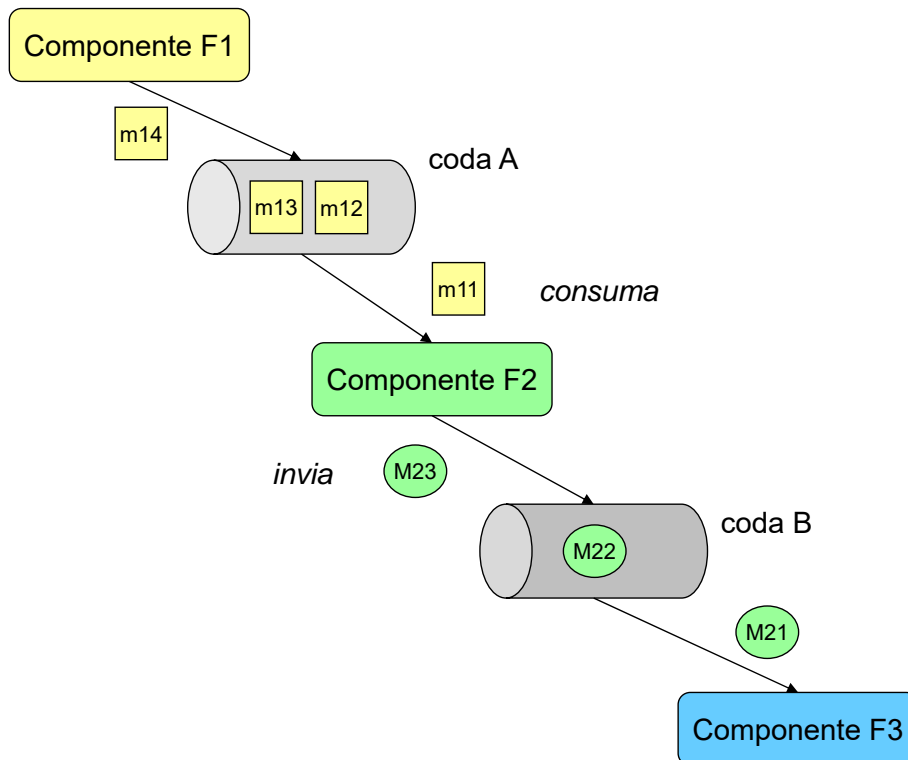




Canali point-to-point: più canali (più tipi di messaggi)

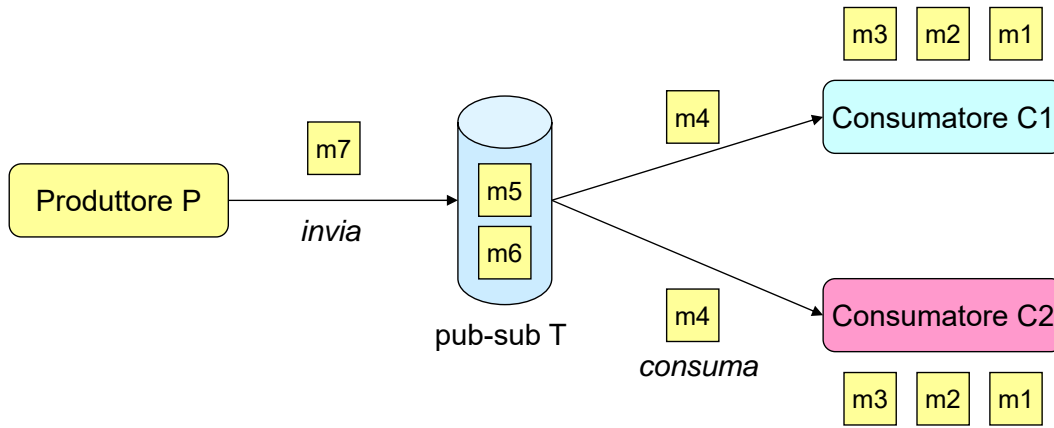


Canale point-to-point: componenti consumatori/produttori

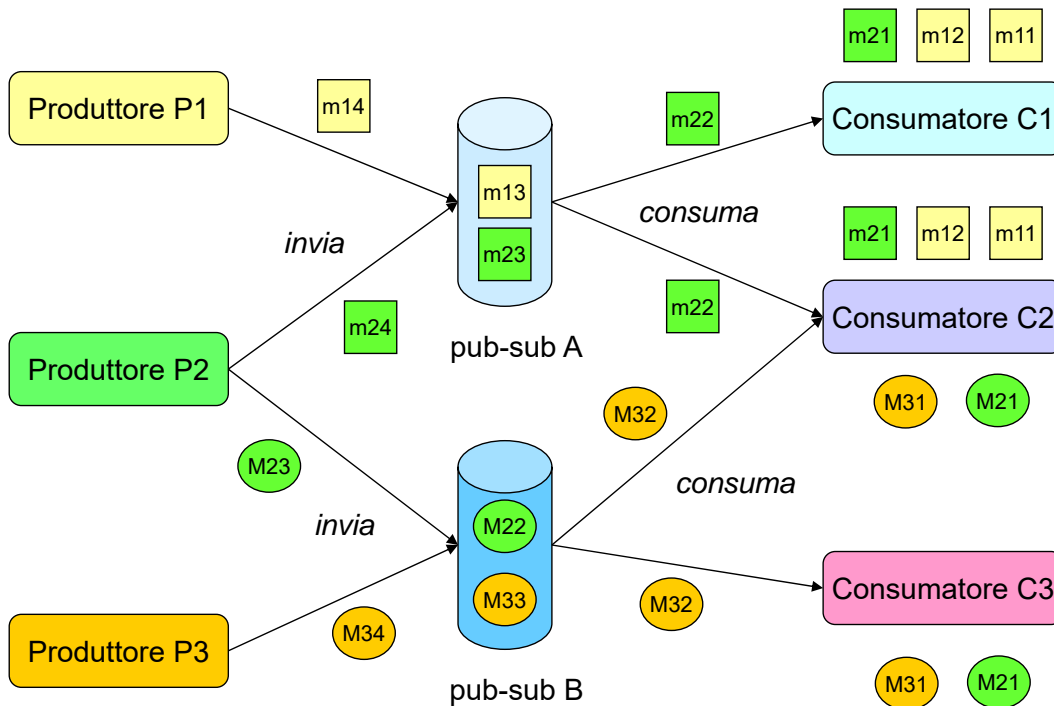




Canale publish-subscribe



Canali publish-subscribe





- Prestazioni e scalabilità

- La comunicazione asincrona può abilitare **prestazioni e scalabilità**
 - i messaggi vengono in genere trasmessi **appena possibile**, con una latenza bassa e un throughput elevato
 - le prestazioni e la scalabilità sono favorite anche dalla possibilità di utilizzare più componenti consumatori per uno stesso tipo di messaggi – per *Maintain multiple copies of computations*



- Un esempio

- Un esempio che descrive una possibile applicazione della comunicazione asincrona
 - ci sono componenti che hanno dei dati da elaborare
 - ad es., componenti che ricevono immagini da elaborare dagli utenti di un'applicazione
 - per ciascuno dei dati va eseguito un certo compito
 - ad es., un ridimensionamento dell'immagine
 - il compito si può eseguire indipendentemente su ciascun dato
 - ad es., separatamente per ciascuna immagine
 - c'è anche un componente in grado di svolgere questo compito
 - il compito non va necessariamente svolto in modo sincrono
 - ciascun compito può produrre dei risultati – ma non necessariamente una risposta a chi l'ha richiesto

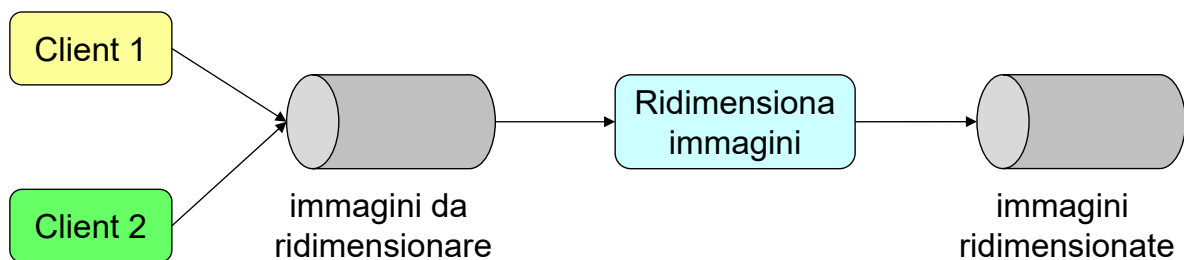


Un esempio

- In questo caso, il sistema potrebbe essere organizzato sulla base di componenti che comunicano in modo asincrono
 - i componenti che hanno dati da elaborare agiscono da produttori di messaggi
 - ad es., producono un messaggio per ciascuna immagine da elaborare
 - il componente che sa svolgere il compito agisce da consumatore di messaggi
 - ad es., riceve l'immagine, la ridimensiona e la salva da qualche parte – oppure la invia a un altro canale per messaggi



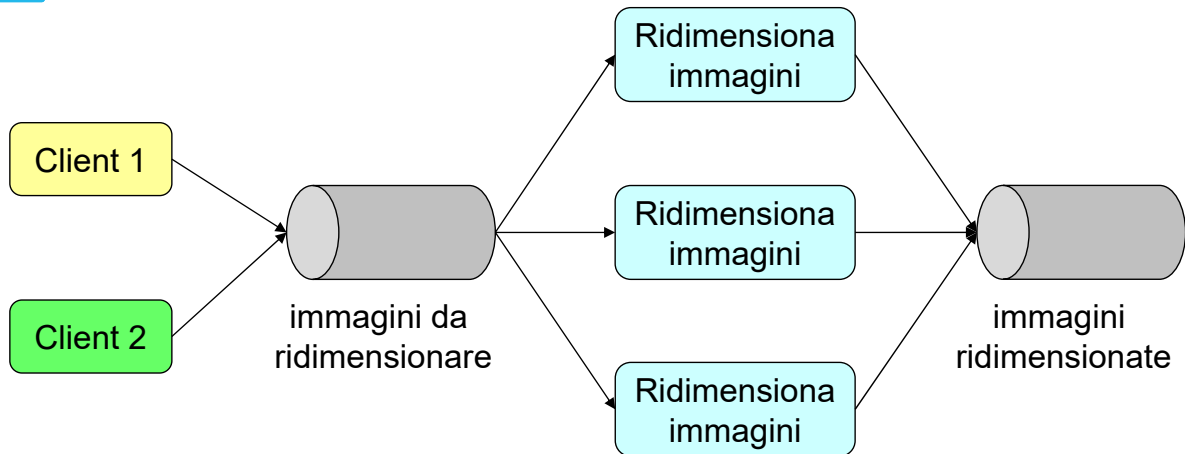
Un esempio



- l'uso di un canale point-to-point consente di gestire il compito da svolgere in modo asincrono



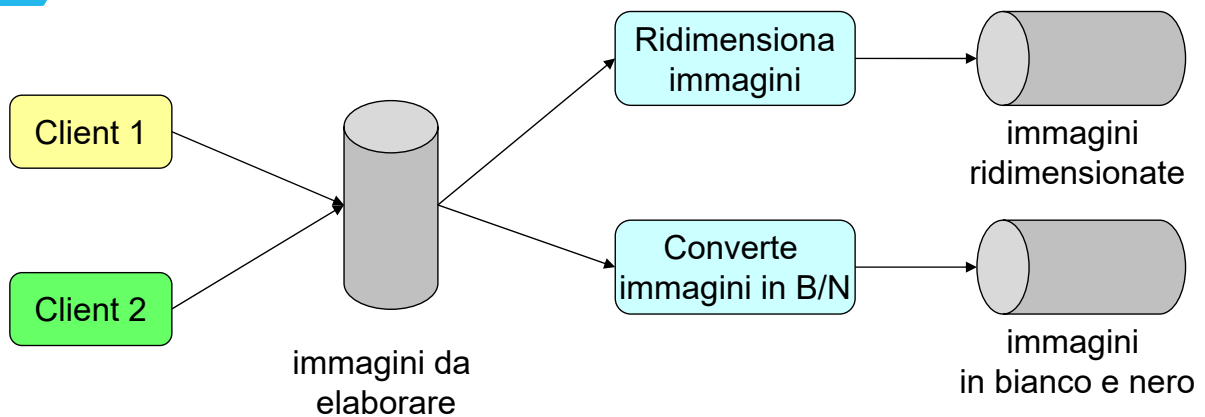
Un esempio



- un canale point-to-point consente anche di distribuire il carico relativo al compito da svolgere tra più consumatori



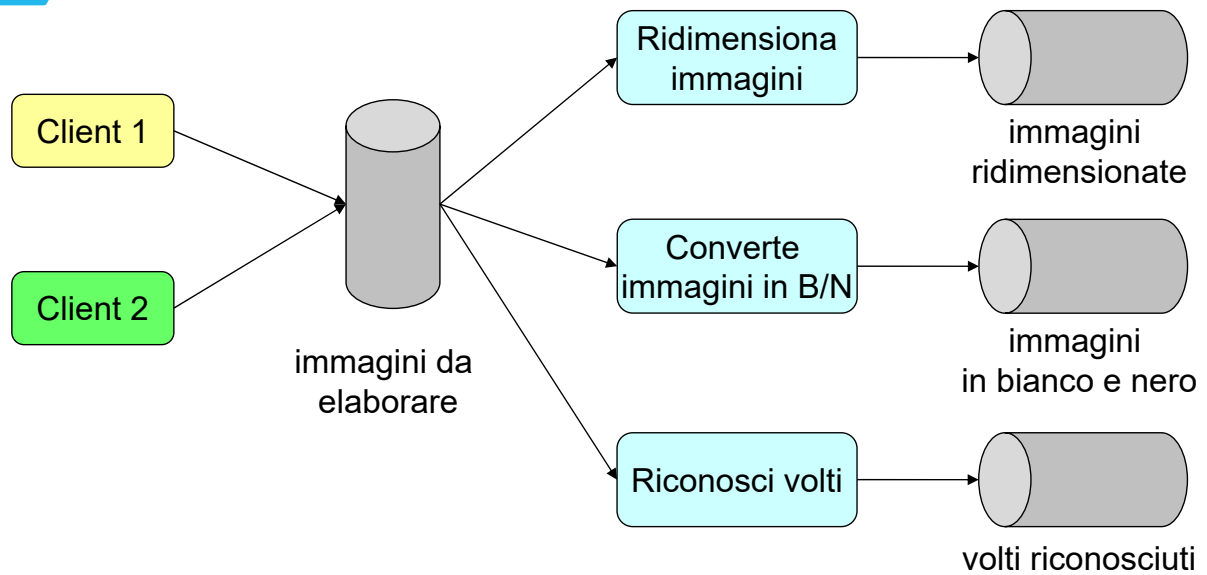
Un esempio



- un canale publish-subscribe consente di distribuire il carico relativo a più compiti da svolgere tra più consumatori, ciascuno dei quali è specializzato per un compito differente



Un esempio



- un canale publish-subscribe consente anche di aggiungere dinamicamente nuovi componenti per svolgere dei compiti aggiuntivi



Discussione

- Questo esempio consente di comprendere il supporto fornito dalla comunicazione asincrona a qualità come la modificabilità, le prestazioni e la scalabilità
 - ma la comunicazione asincrona sostiene anche altre qualità



- Altri tipi di canali per messaggi

- Alcuni message broker hanno caratteristiche diverse da quanto descritto finora
 - ad es., Kafka prevede un solo tipo “ibrido” di canale per messaggi
 - inoltre, introduce una nozione aggiuntiva di “gruppo di componenti”, tale che un messaggio inviato a un canale viene consegnato a un solo componente per ciascun gruppo
 - consente di consegnare messaggi in modalità point-to-point, publish-subscribe, o anche in modi più flessibili



- Aspetti temporali

- Altre varianti della semantica della comunicazione asincrona riguardano gli aspetti temporali e la modalità di memorizzazione dei messaggi nel message broker
 - un messaggio inviato a un canale point-to-point viene in genere conservato fino a quando non è stato consegnato a un consumatore – dopo di che, viene cancellato dal canale
 - tuttavia, è anche possibile assegnare a un messaggio una durata o scadenza



Aspetti temporali

- Altre varianti della semantica della comunicazione asincrona riguardano gli aspetti temporali e la modalità di memorizzazione dei messaggi nel message broker
 - un messaggio inviato a un canale publish-subscribe viene in genere consegnato a tutti i consumatori attualmente registrati presso quel canale – e poi viene cancellato dal canale
 - i consumatori per un canale publish-subscribe ricevono solo i messaggi inviati durante il periodo della loro registrazione
 - è possibile che un messaggio venga cancellato anche senza essere stato consegnato a nessun consumatore
 - sono anche possibili canali publish-subscribe “persistenti” – oppure registrazioni “durature”



Discussione

- La memorizzazione dei messaggi nel message broker richiede risorse
 - la gestione delle scadenze, dei canali persistenti e delle registrazioni durature richiede una quantità di risorse ancora maggiore
 - in generale, la modalità di memorizzazione dei messaggi va configurata in modo dipendente anche dal contesto applicativo



- Ordine dei messaggi

- In alcuni casi, un consumatore potrebbe ricevere i messaggi inviati da un produttore in un ordine differente da quello in cui sono stati inviati
 - ad es., questo può accadere se il message broker è implementato da un cluster di nodi e i diversi messaggi transitano su nodi differenti
 - se l'ordine in cui sono stati inviati i messaggi è importante
 - in alcuni casi, il message broker può fornire meccanismi per garantire la consegna in ordine di messaggi correlati
 - altrimenti, sono i consumatori che devono prevedere la possibilità di ricevere messaggi fuori ordine



- Perdita e duplicazione di messaggi

- Idealmente, il message broker dovrebbe consegnare ciascun messaggio con una semantica *exactly-once* – tuttavia, la consegna dei messaggi ha in genere una semantica *at-most-once* o *at-least-once*
 - quando il sistema opera normalmente, ciascun messaggio viene in effetti consegnato una sola volta
 - in situazioni anomale, i messaggi possono venire persi o essere consegnati più volte
 - in quest'ultimo caso, se opportuno
 - i consumatori dei messaggi devono effettuare elaborazioni idempotenti – o ignorare messaggi duplicati (ad es., sulla base di un id)
 - questo può essere problematico se ci sono più consumatori per uno stesso tipo di messaggi



- Affidabilità

- Un message broker può offrire diverse opzioni di consegna dei messaggi – a cui corrispondono livelli differenti di affidabilità
 - consegna non persistente (di tipo best effort)
 - i messaggi possono perdersi
 - consegna persistente
 - i messaggi non possono perdersi, perché il message broker registra in memoria secondaria i messaggi che gli vengono inviati
 - acknowledgment
 - un messaggio è considerato consumato con successo quando il consumatore a cui è stato assegnato ne conferma la ricezione
 - altrimenti, il message broker assegna il messaggio a un altro consumatore



Affidabilità

- Un message broker può offrire diverse opzioni di consegna dei messaggi – a cui corrispondono livelli differenti di affidabilità
 - consegna transazionale
 - un messaggio è considerato consumato con successo quando il consumatore a cui è stato assegnato ne conferma l'avvenuta elaborazione
 - altrimenti, il messaggio viene assegnato a un altro consumatore
 - transazioni
 - un componente può specificare una transazione per eseguire in modo atomico la ricezione e/o l'invio di uno o più messaggi – se la transazione fallisce, i messaggi vengono assegnati ad altri consumatori
 - queste transazioni possono talvolta includere anche compiti più ampi – ad es., l'accesso a una base di dati



Discussione

- Le diverse opzioni per l'affidabilità richiedono risorse sul message broker via via maggiori
 - l'aumento dell'affidabilità va di solito a scapito delle prestazioni e della scalabilità
 - in generale, il livello di affidabilità va configurato in modo dipendente anche dal contesto applicativo



Comunicazione asincrona e affidabilità

- Dunque, la comunicazione asincrona è una modalità di comunicazione che può sostenere anche l'**affidabilità**
 - la consegna dei messaggi può essere configurata con riferimento a diversi livelli di affidabilità
 - l'affidabilità (tolleranza ai guasti) è sostenuta anche dalla possibilità di utilizzare repliche ridondanti dei componenti consumatori



- Discussione

- La comunicazione asincrona prevede numerose varianti e opzioni – ciascuna delle quali può avere un impatto differente sui diversi attributi di qualità di un sistema
 - in generale, la modalità di utilizzo e di configurazione della comunicazione asincrona va stabilita in modo dipendente anche dal contesto applicativo – cercando un buon compromesso tra le opzioni disponibili



* Utilizzo di messaggi nelle applicazioni

- Discutiamo ora alcuni aspetti relativi all'uso della comunicazione asincrona nello sviluppo delle applicazioni software
 - tre modi concreti di utilizzare i messaggi da scambiare in modo asincrono
 - eventi di dominio
 - comandi
 - documenti
 - due modalità di consumo dei messaggi da parte di un consumatore



- Eventi di dominio

- Un *evento di dominio* [DDD] rappresenta qualcosa di interessante che è avvenuto nel dominio di un componente o servizio
 - un componente può inviare messaggi relativi ai propri eventi di dominio – in un canale publish-subscribe specifico per gli eventi di quel componente
 - i componenti interessati a questi eventi possono abbonarsi a questo canale



- Comandi

- Un *comando* [GoF] rappresenta una richiesta di eseguire un'operazione a un componente o servizio, che include anche i parametri della richiesta
 - un componente può ricevere messaggi che rappresentano comandi di richiesta per le proprie operazioni – in un canale point-to-point specifico per i comandi di quel componente
 - i componenti interessati possono inviare messaggi per comandi a questo canale
 - l'elaborazione dei comandi viene effettuata dal componente mediante un *command handler*



- Documenti

- Un *documento* è un messaggio generico che contiene solo dei dati
 - ad es., una risposta a una richiesta oppure un gruppo di dati da elaborare
 - per ogni tipologia di documenti viene utilizzato un canale specifico
 - il componente che riceve un documento decide come interpretare ed elaborare il messaggio ricevuto (“invocazione implicita”)



- Modalità di consumo dei messaggi

- La ricezione di un messaggio da parte di un consumatore può avvenire secondo due modalità principali
 - polling
 - su richiesta del consumatore
 - subscription
 - basata su una registrazione dei consumatori al canale
 - è il message broker che assegna i messaggi ai consumatori
 - molti message broker (e le API per il loro accesso) privilegiano la modalità “subscription”



* Discussione

- La comunicazione asincrona è un'astrazione di programmazione distribuita basata sullo scambio di messaggi
 - consente a un componente (produttore o publisher) di inviare un messaggio, in modo indiretto e asincrono, a uno o più componenti (consumatori o subscriber), affinché questi possano elaborare il messaggio
 - è complementare all'invocazione remota
 - sostiene qualità come modificabilità, prestazioni, scalabilità e affidabilità
 - questo stile di comunicazione è sostenuto dal pattern architetturale *Messaging* [POSA]



Quando usare la comunicazione asincrona?

- Quando preferire la comunicazione asincrona all'invocazione remota?
 - se i componenti di un'applicazione possono non essere tutti attivi contemporaneamente
 - se i componenti possono essere progettati in modo tale da inviare informazioni ad altri componenti – continuando a lavorare anche senza ricevere una risposta immediata
 - se i componenti devono/possono essere progettati in modo da ignorare le interfacce degli altri componenti, basando la comunicazione solo sul formato dei messaggi scambiati
 - come caso estremo, se i componenti che devono comunicare sono stati sviluppati indipendentemente l'uno dall'altro (integrazione di applicazioni)



Benefici della comunicazione asincrona

- Benefici della comunicazione asincrona
 - una forma di comunicazione remota
 - comunicazione indiretta
 - comunicazione asincrona
 - temporizzazione variabile
 - eliminazione di colli di bottiglia
 - abilita operazioni disconnesse
 - affidabilità della comunicazione
 - può favorire l'integrazione tra linguaggi e piattaforme
 - consente una migliore gestione dei thread



Sfide della comunicazione asincrona

- Sfide legate all'uso della comunicazione asincrona
 - maggior complessità di questo paradigma di programmazione – ad esempio
 - non c'è più un singolo thread di esecuzione
 - se l'esecuzione di un'operazione prevede dei risultati, anche questi arrivano di solito in modo asincrono
 - l'esecuzione dei thread di un componente che operano in modo asincrono può avvenire in un ordine qualunque
 - introduce un overhead che può avere un impatto negativo sulle prestazioni
 - problemi indotti dall'ordine di ricezione dei messaggi
 - è talvolta necessario gestire anche scenari sincroni
 - protocolli e implementazioni proprietarie – il “vendor lock-in” può limitare la portabilità