Luca Cabibbo

# Architettura dei Sistemi Software

# Comunicazione asincrona

Abbiamo detto che parliamo di due paradigmi di programmazione distribuita:

- invocazione remota ("si chiede")
- comunicazione asincrona (basata sullo scambio di messaggi, sostiene molto meglio delle qualità)

Nel progetto uno degli aspetti è il passaggio dal primo al secondo

**dispensa asw450**

ottobre 2024

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

*The Police*



## - Riferimenti

- Luca Cabibbo. **Architettura del Software**: Strutture e Qualità. Edizioni Efesto, 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.

# - 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 elabora 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

Ha dei dati che non può o non vuole elaborare. Incapsula questi dati in un messaggio e li manda al consumatore. L'obiettivo è lo stesso ma non sto chiedendo. Vedremo che essendo una comunicazione asincrona non aspetto la risposta, ma in generale potrei anche non aspettarmela proprio

Nell'invocazione sceglie il server, qui verrà scelto il consumatore in qualche modo

# Sulla comunicazione sincrona

- Prima di parlare della comunicazione asincrona, discutiamo limiti e inconvenienti dell'invocazione remota
- 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 un risultato
  - fornisce una sincronizzazione tra client e server
    In alcuni casi serve, ad esempio nel caso di verifica di una password, ma in altre situazioni non è così vincolante e anzi rischia di rallentare il tutto
  - tuttavia, questa modalità di comunicazione non sempre è necessaria o la più opportuna



# Sulla comunicazione sincrona

Non sempre chi richiede un'elaborazione si attende o vuole un risultato

- Tuttavia, non sempre questa modalità di comunicazione è necessaria o è la più opportuna
  - in alcuni casi, un componente vuole che venga eseguita un'elaborazione – ma potrebbe non essere interessato a ottenere un risultato (o a ottenerlo in modo sincrono)
  - spesso è desiderabile una modalità di interazione meno accoppiata e più flessibile In alcuni casi si desidera un accoppiamento più debole
    - non si vuole l'accoppiamento con uno specifico server, servizio o operazione
  - spesso si vogliono evitare le conseguenze negative su alcune qualità causate dalla comunicazione sincrona
    - perché può propagare rallentamenti o guasti, o può ridurre la concorrenza
    - perché la gestione di questi aspetti può rendere più complicata la realizzazione dei client

# 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 modificabilità, 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*)

L'idea fondamentale è che la comunicazione sia basata sullo scambio di messaggi (infatti si chiama messaging). In particolare questi messaggi conterranno dati, documenti, risposte, richieste di invocazioni, eventi di dominio. La comunicazione è indiretta e asincrona

# 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

- 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 – qui siamo interessati alla comunicazione tra componenti software

# 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

In generale per fare qualcosa di interessante servirà scambiarsi molti messaggi. Partiamo da uno solo



# - 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

Anche il client ha dei dati e vuole eseguire un'operazione, ma sa quale è. Qui il produttore ha solo i dati

  - 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 i dati e li elabora

# La comunicazione è indiretta

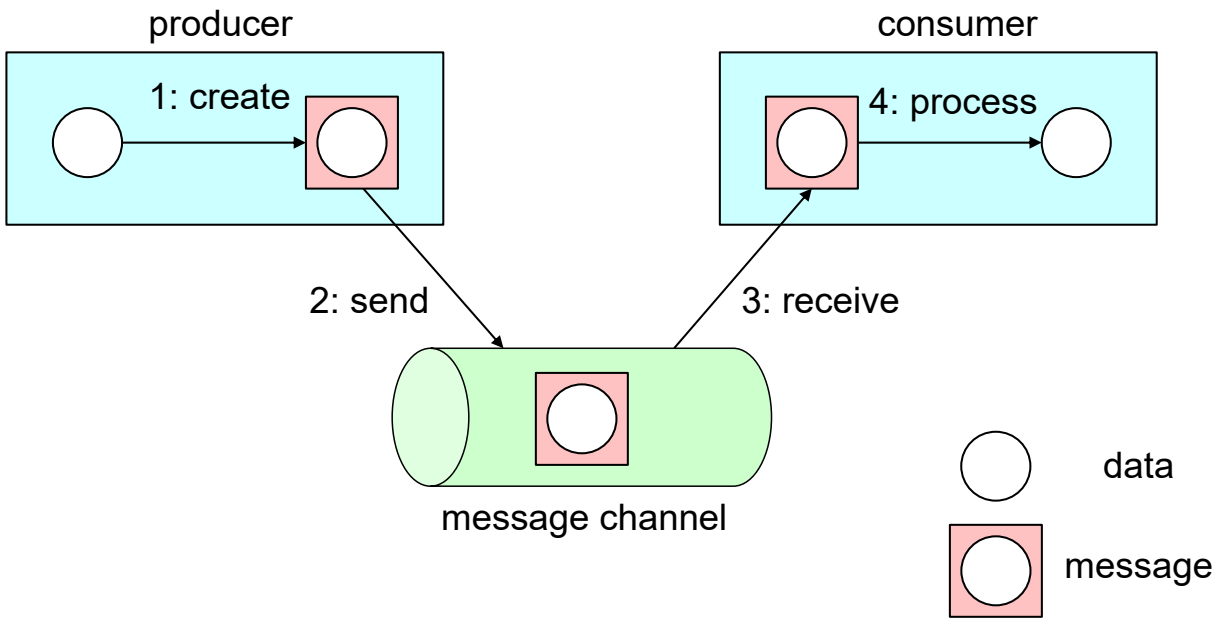
- La comunicazione è *indiretta* – il produttore non trasmette il messaggio direttamente a uno specifico consumatore
  - il produttore invia il messaggio a un *canale per messaggi* – specifico per quel tipo di messaggi
  - quindi, un consumatore riceve il messaggio prelevandolo da quel canale per messaggi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

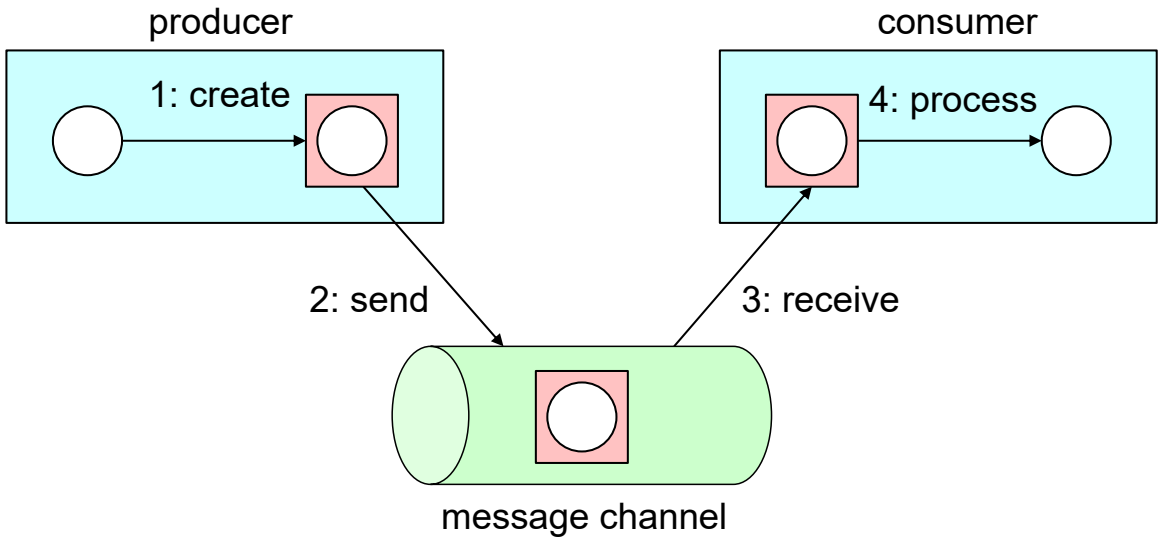
# Comunicazione asincrona



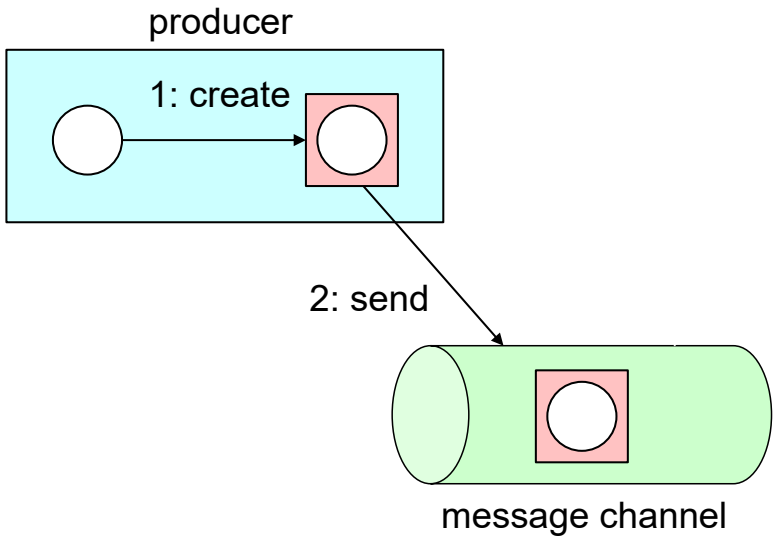



# La comunicazione è indiretta

In questa slide, nell'animazione, quando il produces è in esecuzione e invia il messaggio, il consumer non esiste proprio (non è in esecuzione). Una volta che ha inviato il messaggio, anche il producer termina. Solo dopo il consumer va in esecuzione e preleva il messaggio dal canale per messaggi.

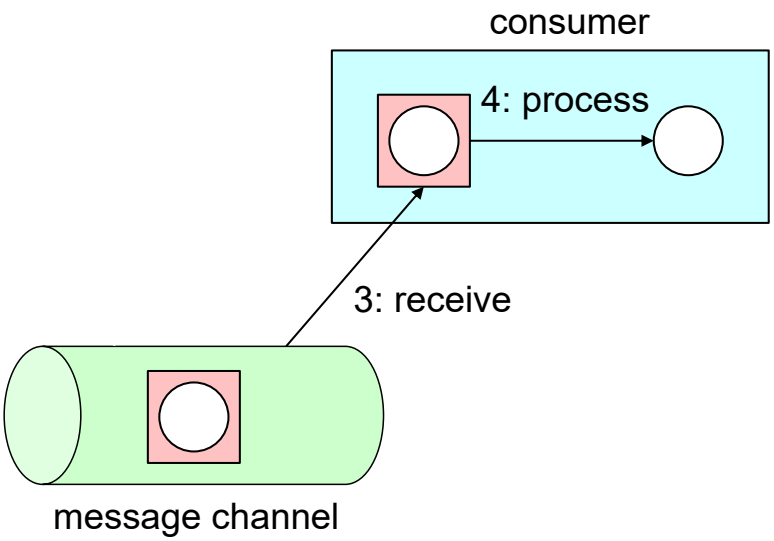


# La comunicazione è asincrona






# La comunicazione è asincrona



# Invocazione implicita

- La comunicazione asincrona è una forma di *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

Applicazione di abstract common services. Fai in modo che uno offra un servizio astratto che possa essere richiesto mediante una espressione in un linguaggio, nel caso minimale un nodo parametrico, nel caso reale il parametro è un'espressione di un linguaggio. Il nostro messaggio codifica dei dati sulla base di qualche formalismo comune tra produttore e consumatore. Quello che fa il consumatore dipenderà dalla codifica dei dati.

Ci sono almeno tre motivi per cui si ha una riduzione dell'accoppiamento rispetto all'invocazione remota: c'è indirezione spaziale e temporale, e il fatto che il produttore non dipende dall'interfaccia del consumatore, dall'interfaccia procedurale. Dipende dall'interfaccia dei messaggi ch e il consumatore è in grado di accettare ma generalmente questo è considerato un accoppiamento minore.



# 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



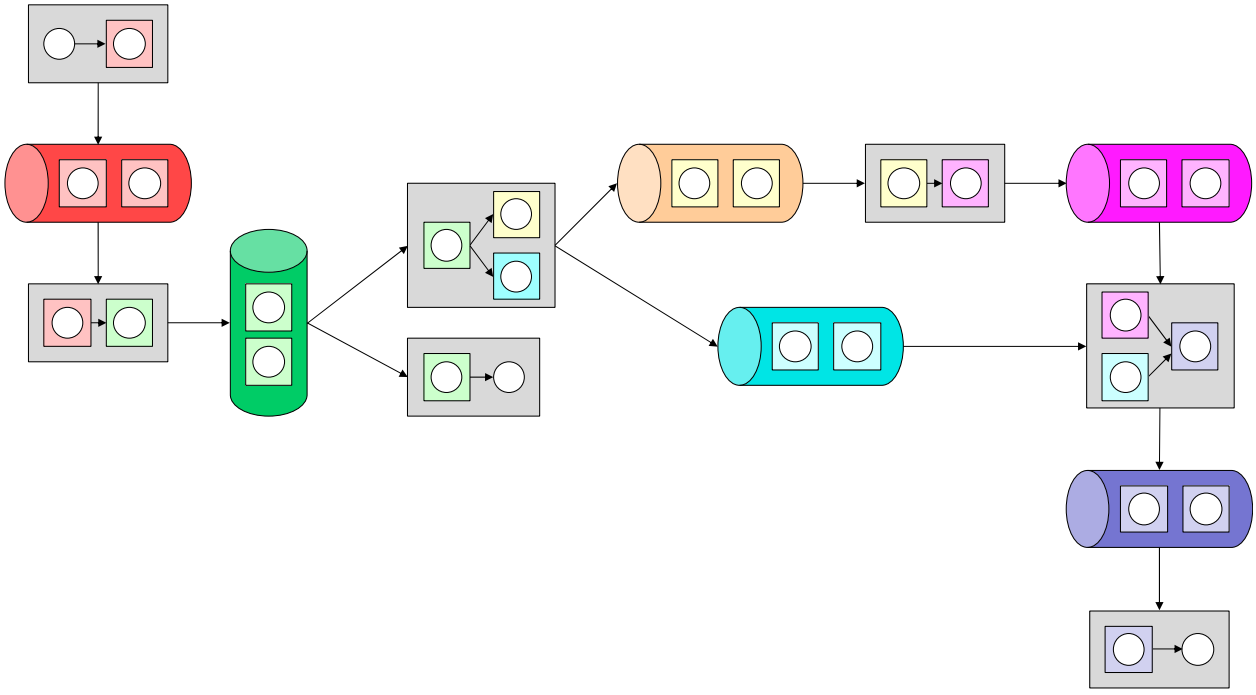
# - Sistemi basati sulla comunicazione asincrona

- 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

Il sistema nel suo complesso, diviso in componenti, riceve dall’esterno un messaggio (“trovato”) iniziale che scatena una serie di interazioni, quindi lo scambio di una serie di messaggi tra una serie di componenti.

Il sistema complessivo riceverà un FLUSSO di messaggi, ognuno dei quali genererà una serie di altri messaggi

# 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 Una pipe per ogni 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 Ogni canale avrà un nome
  - 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

Come nell'invocazione remota, la semantica dei messaggi riguarda alcuni aspetti fondamentali

- 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 (Esistono anche soluzioni non centralizzate)
    - 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 per il message broker viene di solito eseguito in un cluster



# - Tipologie di canali per messaggi

Alcuni message broker per esempio JMS (basato su Java) sono basati su questi due tipi 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" (0,1 o più)

# Canali e consumatori di messaggi

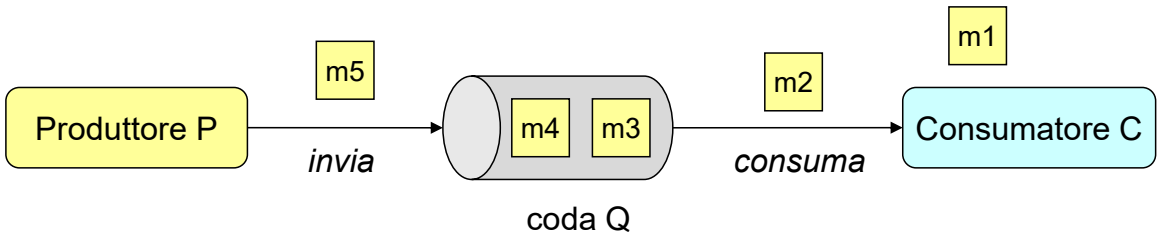
In generale in questi sistemi quando qualche componente consumer o producer si avvia, deve registrarsi ai canali di cui è interessato a ricevere i messaggi, soprattutto nel caso di un consumer. Più precisamente OGNI ISTANZA di un componente deve registrarsi ai canali interessanti per lui.

- 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 point-to-point, a uno solo dei consumatori registrati a C
    - se C è un canale publish-subscribe, a tutti i consumatori registrati a C

OGNI messaggio a TUTTI

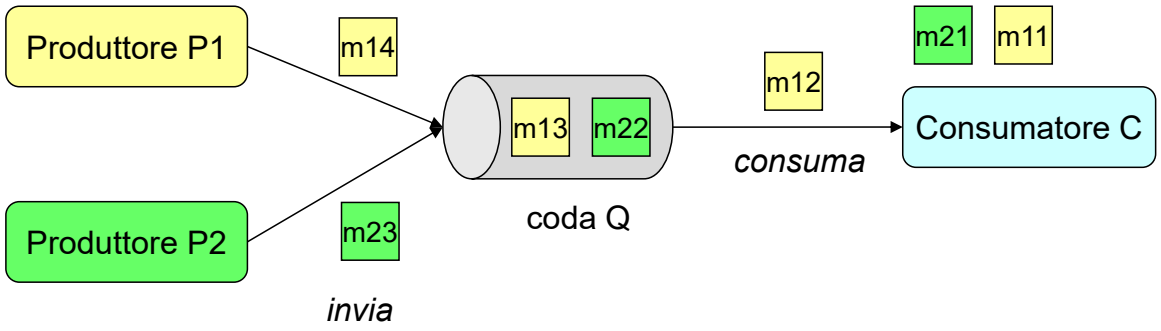


# Canale point-to-point



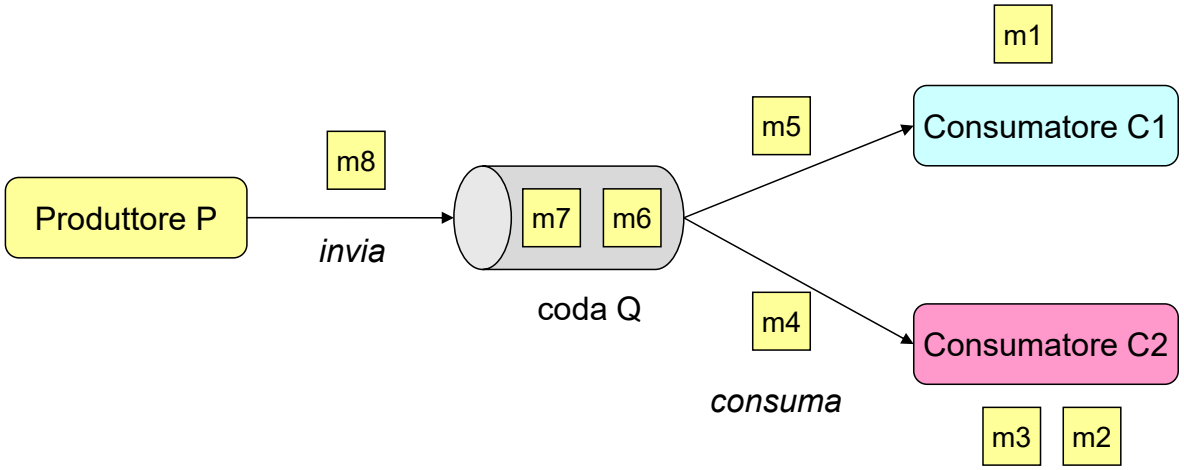
# Canale point-to-point: più produttori

Il tipo di messaggio è dato dalla forma, il colore serve a capire il produttore.
Quindi sono tutti messaggi dello stesso tipo, per questo vanno nello stesso canale



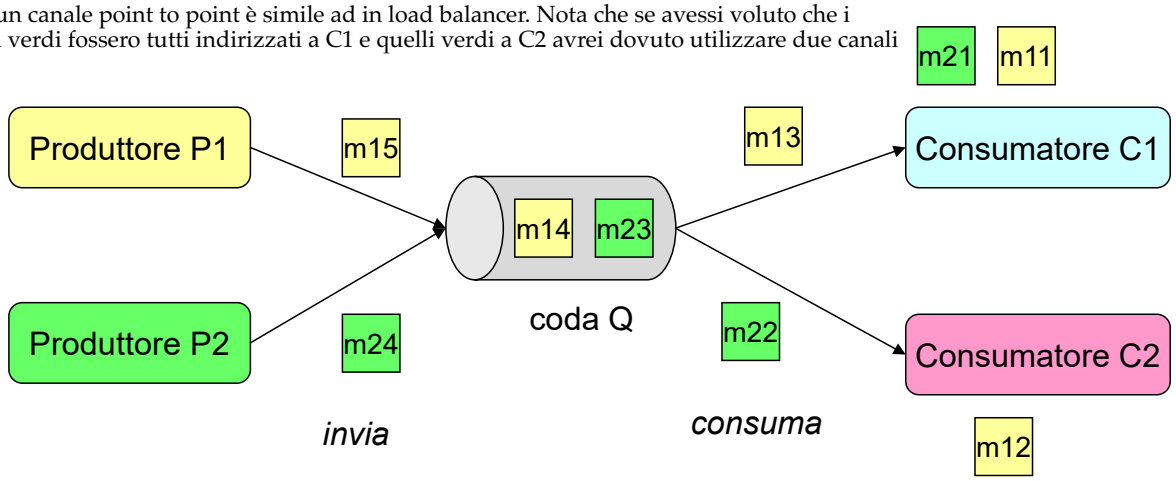



# Canale point-to-point: più consumatori



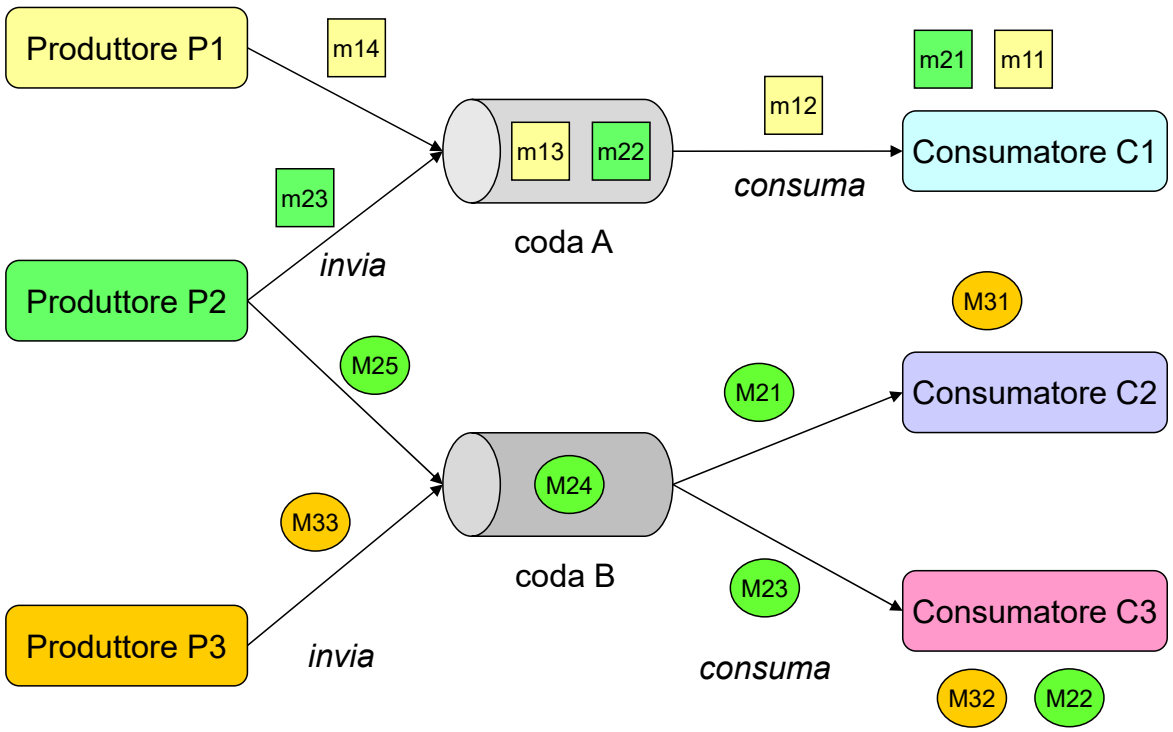
# Canale point-to-point: più produttori e più consumatori

L'uso di un canale point to point è simile ad in load balancer. Nota che se avessi voluto che i messaggi verdi fossero tutti indirizzati a C1 e quelli verdi a C2 avrei dovuto utilizzare due canali

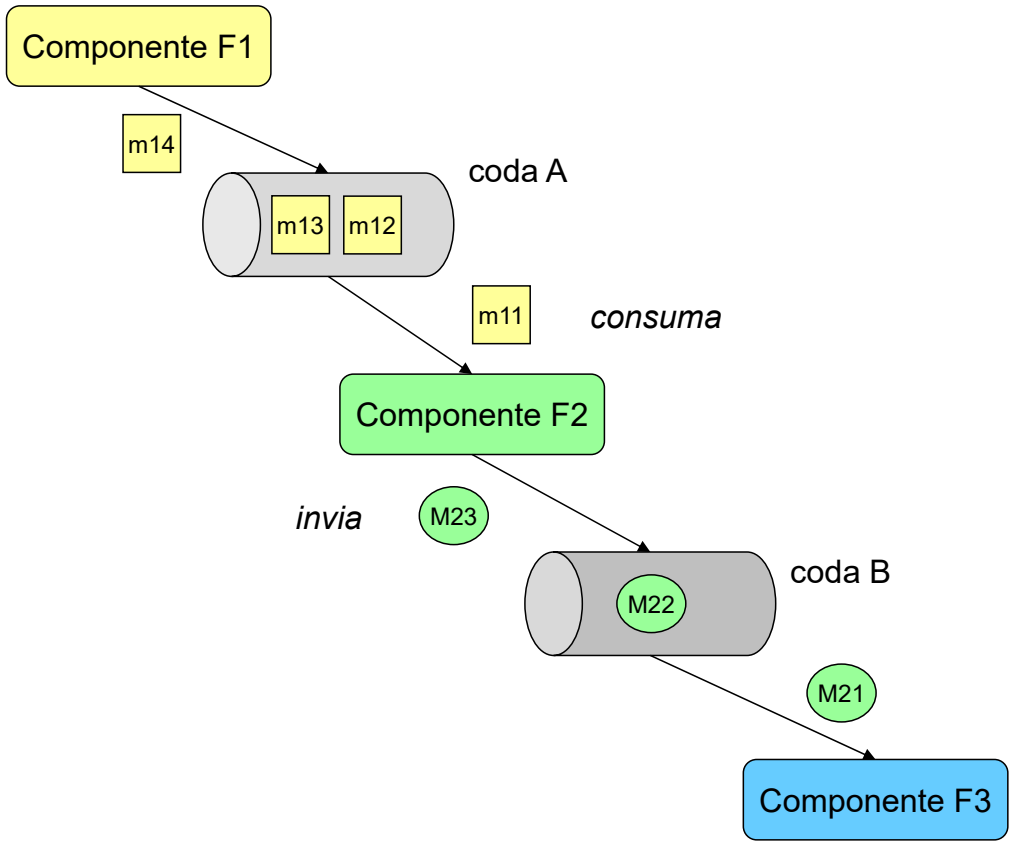





# Canali point-to-point: più canali (più tipi di messaggi)



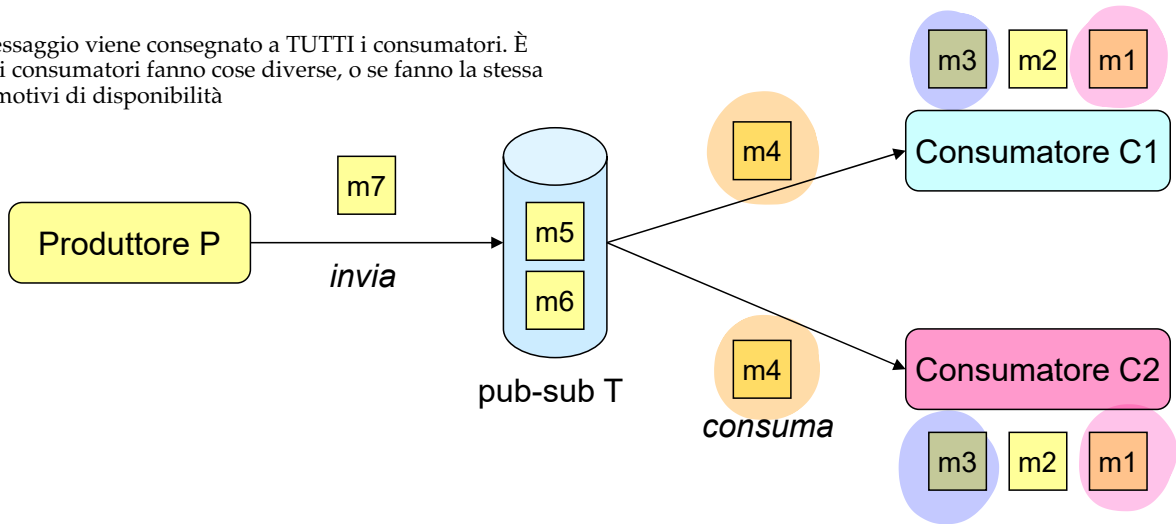
# Canale point-to-point: componenti consumatori/produttori



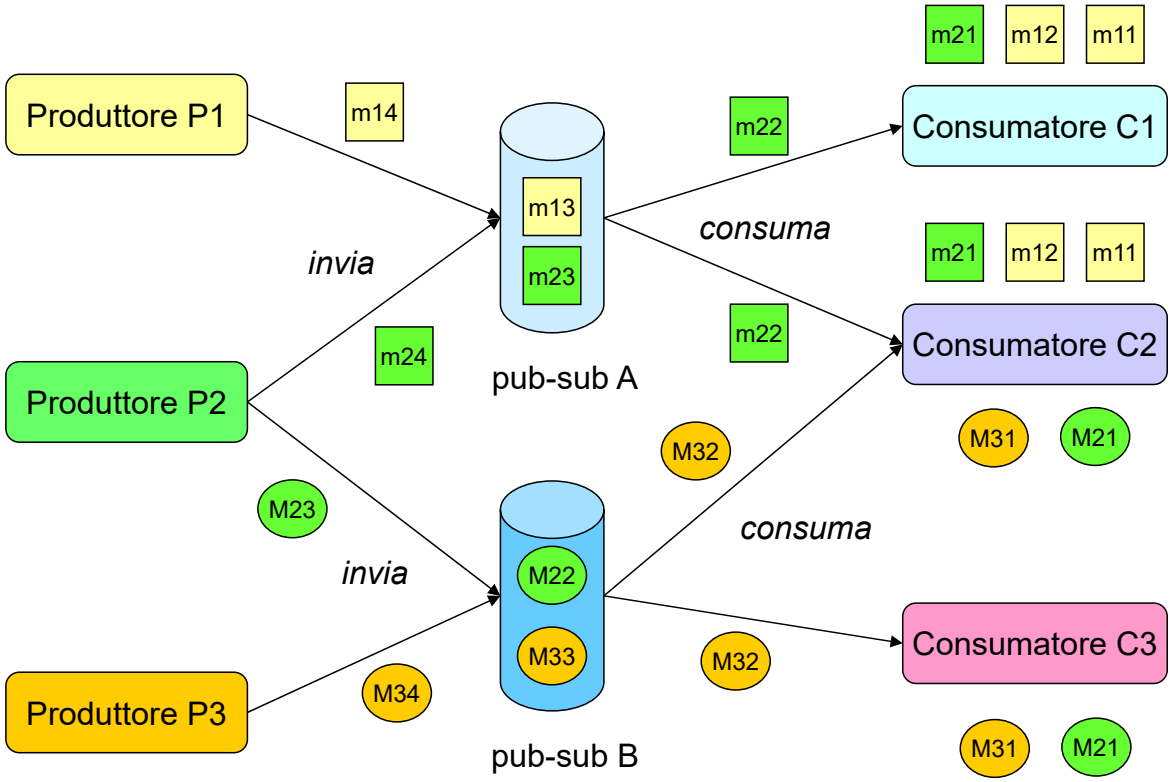



# Canale publish-subscribe

OGNI messaggio viene consegnato a TUTTI i consumatori. È utile o se i consumatori fanno cose diverse, o se fanno la stessa cosa per motivi di disponibilità



# Canali publish-subscribe






# - Prestazioni e scalabilità

In che senso prestazioni se abbiamo detto che un produttore non si aspetta nulla dopo l'invio del messaggio? Prestazioni nel senso che la consegna avviene "appena possibile", in condizioni di normale funzionamento le prestazioni sono alte

- 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*
  - la comunicazione asincrona favorisce la scalabilità (meglio dell'invocazione remota) anche perché
    - può evitare di diffondere rallentamenti e guasti a cascata – e può indurre un livello di utilizzazione maggiore delle risorse
    - il consumo complessivo di risorse è in genere minore

## - Un esempio

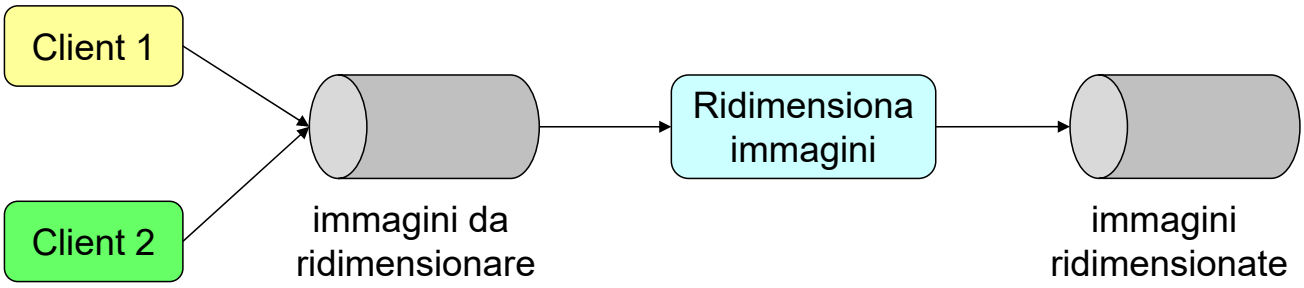
- Un esempio che descrive una possibile applicazione della comunicazione asincrona
  - ci sono componenti che hanno dei dati da elaborare
    - ad es., componenti che ricevono immagini da elaborare dagli utenti di un'applicazione
  - per ciascuno dei dati va eseguito un certo compito
    - ad es., un 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

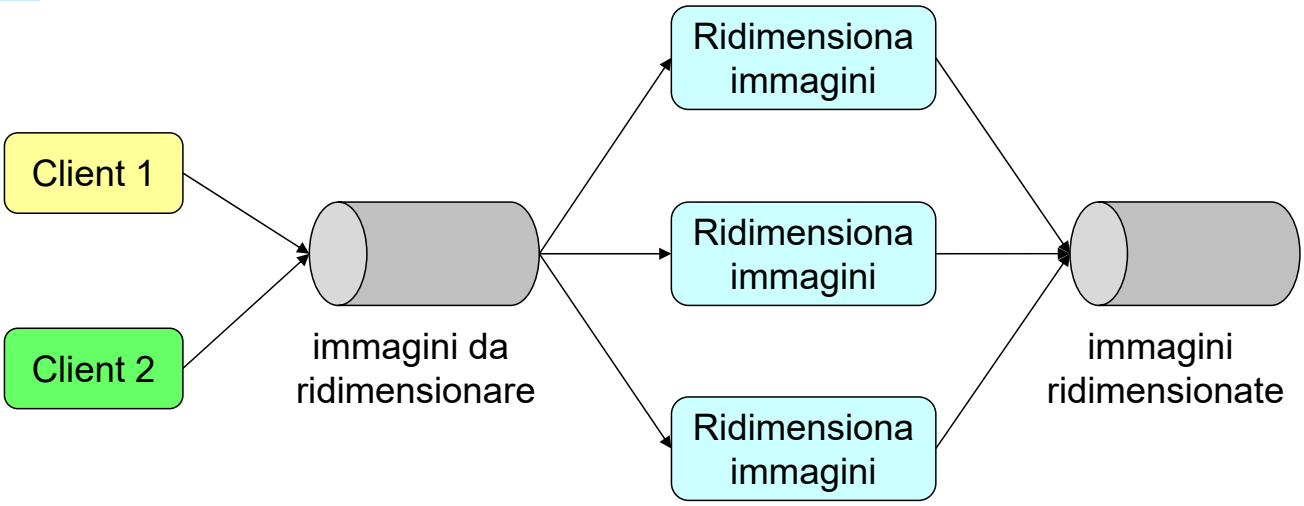
# Un esempio




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

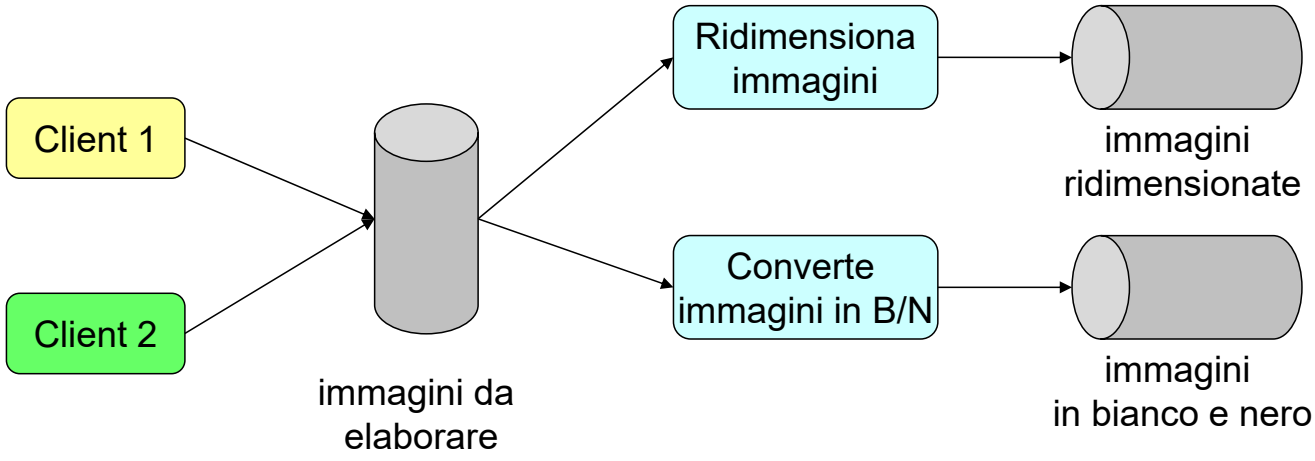


# Un esempio




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

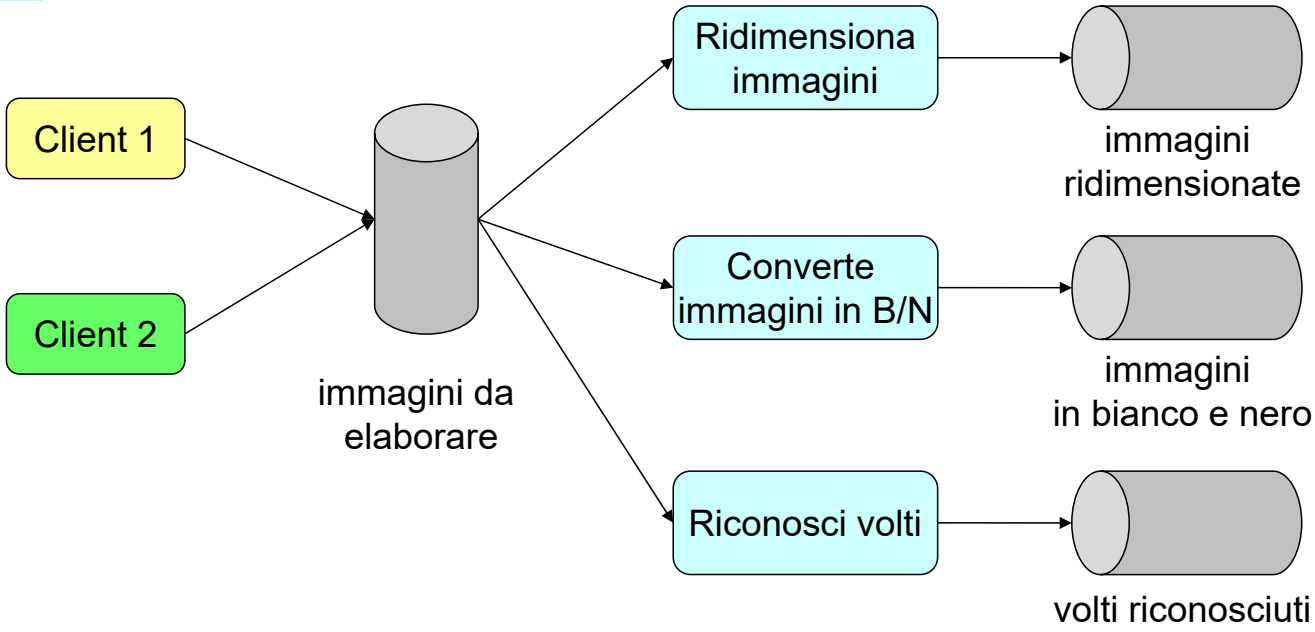
# Un esempio




- 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

    Es affidabilità



# - Altri tipi di canali per messaggi

- Alcuni message broker hanno caratteristiche diverse da quanto descritto finora

  Non prevede né canali point to point né canali pub-sub, ma un canale ibrido
  - 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 (JMS)
  - tuttavia, è anche possibile assegnare a un messaggio una durata o scadenza

Gli aspetti temporali devono essere configurati



# 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

    Quindi a quelli che sono registrati IN QUEL MOMENTO
    - 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"

    Per ricevere anche i messaggi precedenti alla propria registrazione

    Quando un componente si registra la prima volta, anche se opera in maniera parzialmente disconnessa riceverà tutti i messaggi da quel momento in poi

# 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

In generale questo ha impatto negativo sull'uso delle risorse e sulla scalabilità, ma queste tecniche possono essere usate anche esclusivamente su alcuni tipi di sintassi di messaggi che ci interessano particolarmente. In sintesi le cose vanno configurate e vanno configurate bene



# - 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 attraverso 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

Se li voglio ricevere in ordine sto penalizzando le prestazioni ma potrebbero esserci casi in cui questo è vitale, è un'altra cosa su cui devo ragionare. Tendenzialmente va previsto che i messaggi possano essere ricevuti non in ordine

# - Perdita e duplicazione di messaggi

I componenti agiscono come CLIENT nei confronti dei message broker quindi

- 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 – oppure 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) Livello minimo di affidabilità, il broker prova ad affidare il messaggio
    - i messaggi possono perdersi
  - consegna persistente Senza ack
    - i messaggi non possono perdersi, perché il message broker registra in memoria secondaria i messaggi che gli vengono inviati
  - acknowledgment Consegna persistente con ack
    - 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

Questo perché il consumatore a cui è stato consegnato il messaggio potrebbe essere andato in crash, quindi oltre alla consegna con ack si richiede anche che il consumer sia "attivo", confermando la elaborazione

La consegna transazionale è che la singola consegna fa parte di una transazione e la transazione contiene solo la consegna. Nella transazione un componente può includere nella transazione la ricezione, l'invio di messaggi e altre elaborazioni come l'accesso alla base di dati

Nota: l'invio e la ricezione di uno stesso messaggio NON possono mai fare parte della stessa transazione perché riguardano componenti diversi



# 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
    - dalla possibilità di eseguire il message broker in un cluster anziché in un singolo nodo



# - 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 software
  - 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

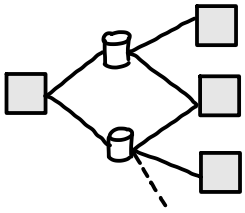
- Discutiamo ora alcuni aspetti relativi all'uso della comunicazione asincrona nello sviluppo delle applicazioni software
  - tre modi concreti di utilizzare i messaggi da scambiare in modo asincrono
    - eventi di dominio
    - comandi
    - documenti
  - due modalità di consumo dei messaggi da parte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



# - Eventi di dominio

- Gli eventi di dominio sono rappresentati dal pattern *Domain Events* [DDD]
  - un *evento di dominio* rappresenta qualcosa di interessante che è avvenuto nel dominio di un componente o servizio
    - ad es., "è stato creato un ristorante" oppure "è stato modificato il menu di un ristorante"
  - 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

Nei diversi componenti ogni volta che succede qualcosa che potrebbe interessare qualche altro componente questo viene notificato tramite evento di dominio



# - Comandi

- I comandi sono rappresentati dal pattern *Command* [GoF]
  - un *comando* rappresenta una richiesta di eseguire un'operazione a un componente o servizio, che include anche i parametri della richiesta

    È proprio una richiesta del tipo "esegui questa operazione"
  - 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*

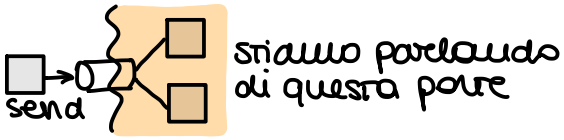


# - Documenti

- Un *documento* è un messaggio generico che contiene solo dei dati
  - ad es., un gruppo di dati da elaborare oppure una risposta a una richiesta
  - 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 Il consumatore dice al broker "quando ricevi un messaggio chiamami con questa operazione per recapitarmelo"
    - 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"




Nella modalità subscription il consumatore deve definire un'operazione associata alla ricezione del messaggio
Nella modalità polling non deve definire alcuna operazione associata alla ricezione del messaggio



# * 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