Sistemi Middleware

Advanced Computer Programming

Prof. Luigi De Simone

Sistemi middleware

- **Contenuti**
  - Sistemi distribuiti e EAI
  - Il concetto di middleware
    - Trasparenza
    - Modelli e tecnologie
  - Esempi di sistemi middleware
    - RPC, MOM, TS, DOM, RDA, CM
- **Riferimenti**
  - Introduzione a CORBA – S. Russo, C. Savy, D. Cotroneo, A. Sergio, McGraw-Hill, CAP 1.

Alcuni domini e applicazioni su rete

- **Finance and commerce**: Amazon, eBay e relative tecnologie di pagamento, per es., PayPal e on-line banking;
- **Information society**: World Wide Web, search engines – Google/Yahoo – digital libraries; user-generated contents (YouTube, Wikipedia); social networks;
- **Entertainment**: on-line gaming, streaming audio-video;
- **Transportation**: location tecnologies, traffic management, web-based map services (Google Maps);
- **Science**: grid computing; storage e analisi di grosse quantità di dati;
- **Utility e cloud-computing**: le risorse (hardware, software) sono fornite da *provider* e non acquistate dagli utenti;
- …

Sistemi distribuiti

Alcune Definizioni:

"Un sistema distribuito è un sistema i cui componenti, localizzati in computer connessi in rete, comunicano e coordinano le loro azioni solo attraverso **scambio di messaggi**" - *G. Coulouris et al.*

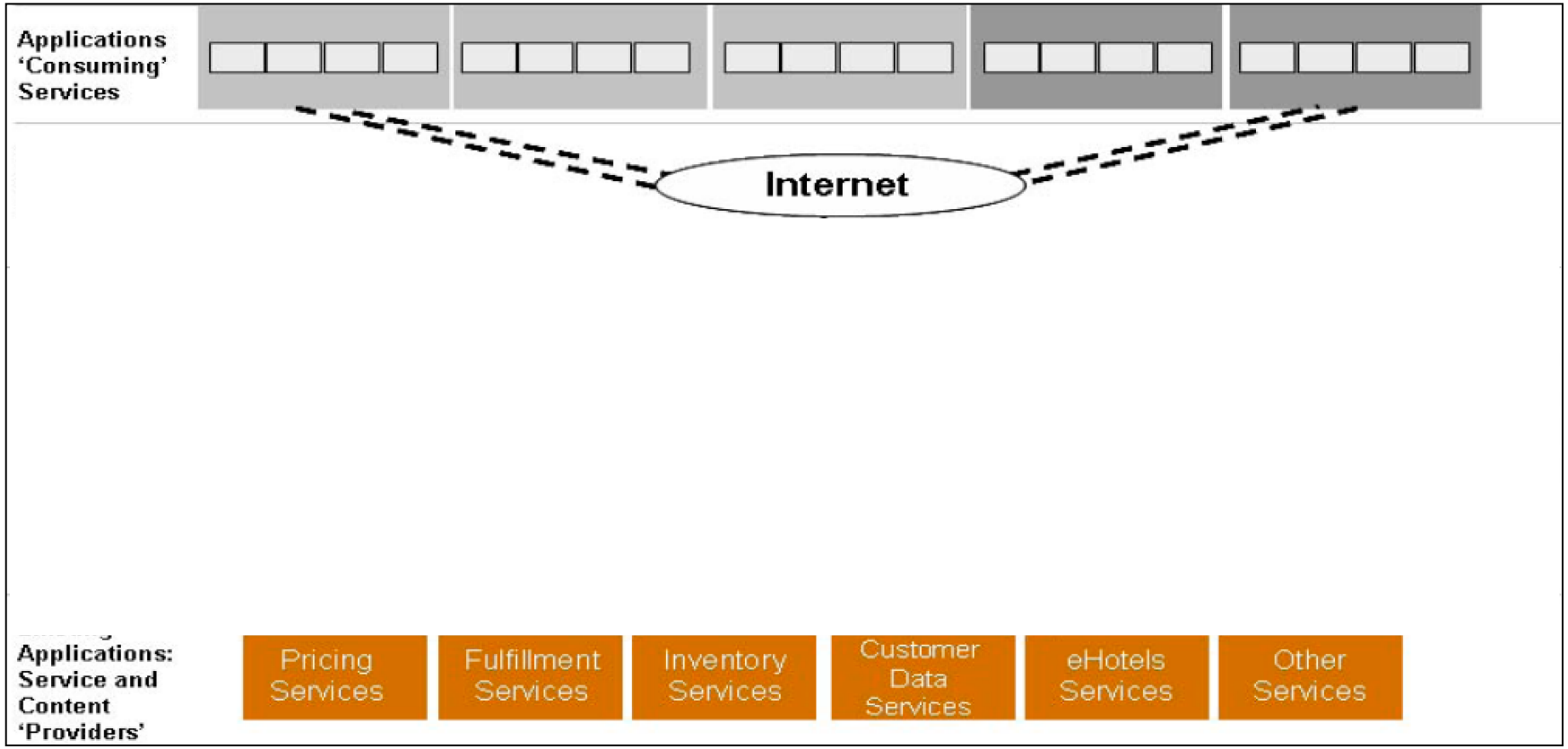
"Il mio programma gira su un sistema distribuito quando non funziona per colpa di una macchina di cui non ho mai sentito parlare" - *L. Lamport*

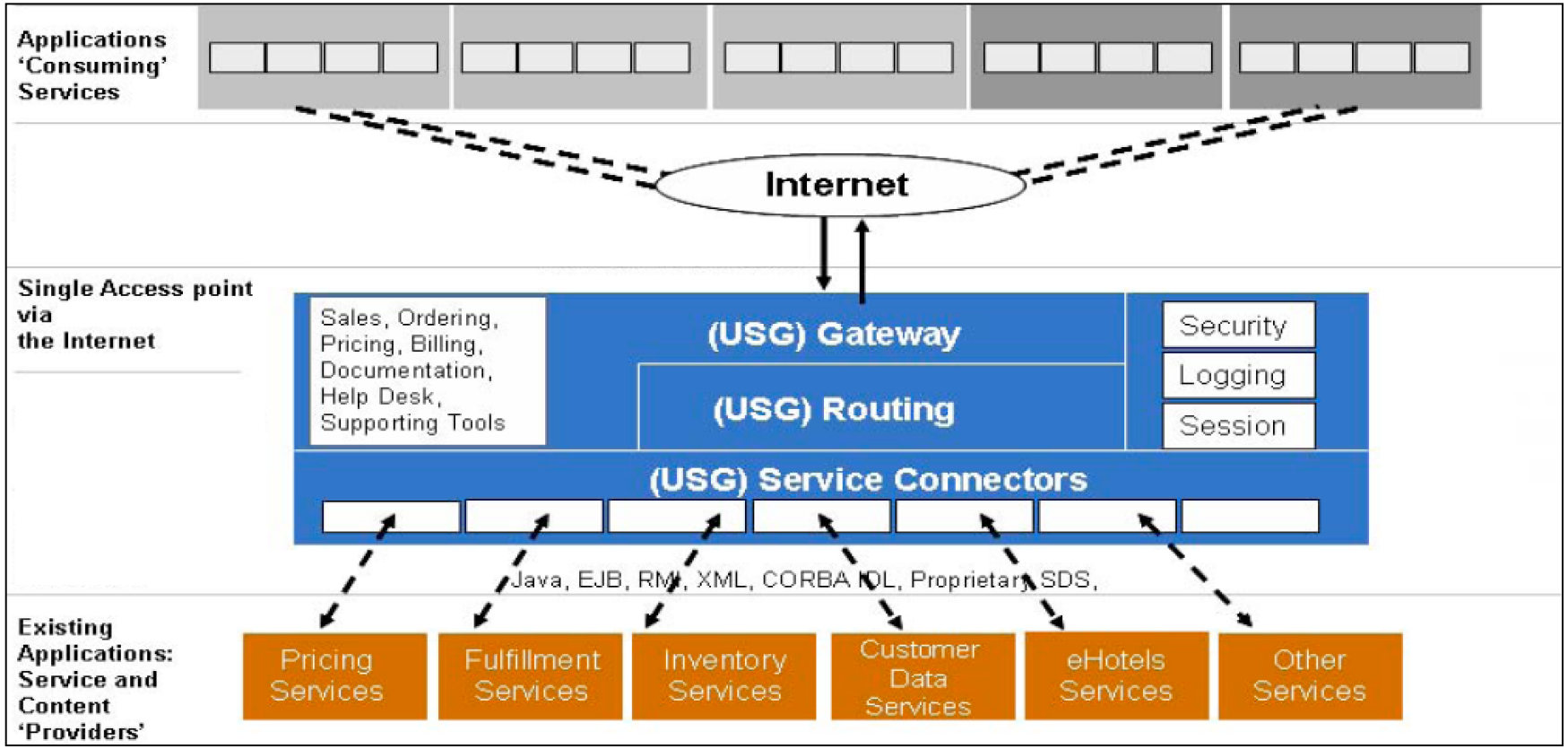
Implicazioni:

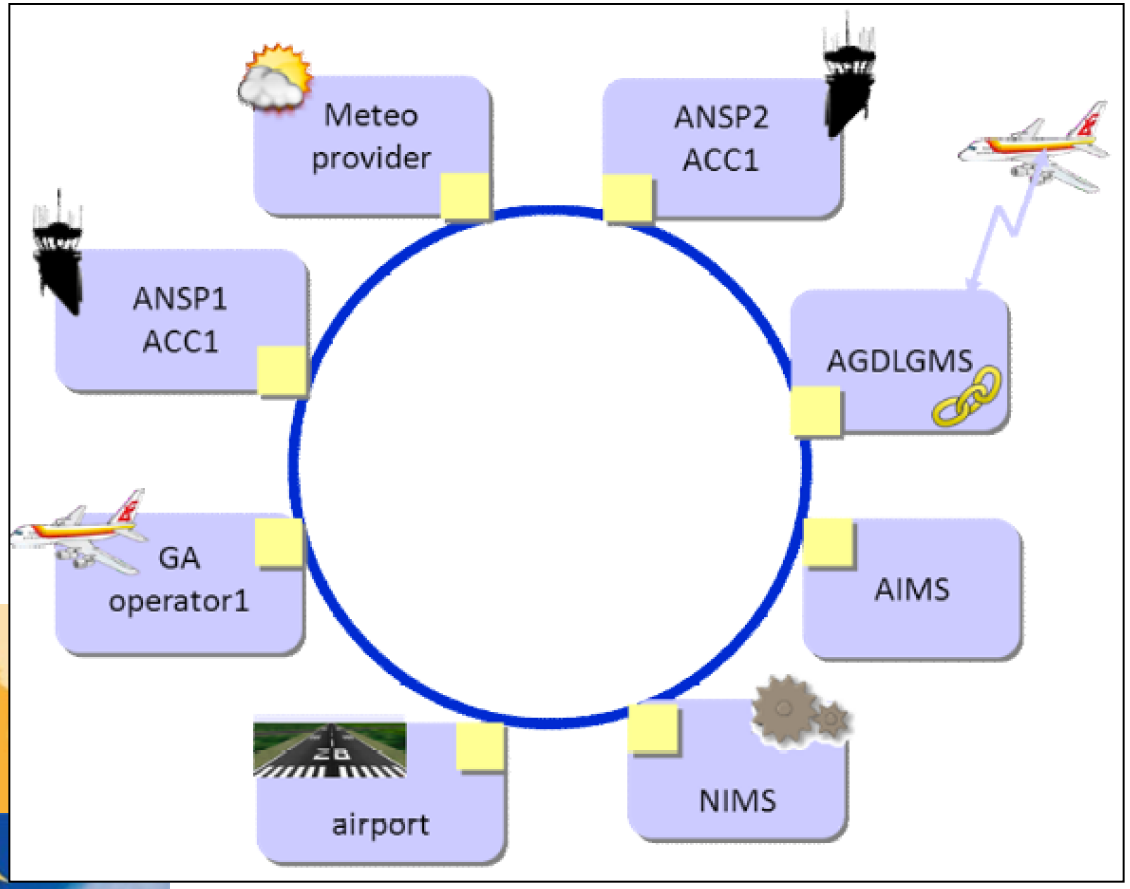
- **Elaborazione concorrente**;
- **Assenza di clock globale**;
- **Malfunzionamenti indipendenti**.

Heterogeneous distributed computing

- **I sistemi distribuiti sono in generale eterogenei**
- L'eterogeneità dei computer e delle tecnologie -hardware e software - è da considerarsi la norma, non l'eccezione, anche all'interno di una stessa grande azienda o organizzazione, pubblica o privata
- Il paradigma di elaborazione più generale prevede componenti interoperanti in ambiente distribuito ed eterogeneo (**heterogeneous distributed computing**)


Applications 'Consuming' Services
Internet
Applications: Service and Content 'Providers'
Pricing Services
Fulfillment Services
Inventory Services
Customer Data Services
eHotels Services
Other Services


Applications 'Consuming' Services
Internet
Single Access point via the Internet
Sales, Ordering, Pricing, Billing, Documentation, Help Desk, Supporting Tools
(USG) Gateway
(USG) Routing
Security
Logging
Session
(USG) Service Connectors
Java, EJB, RMI, XML, CORBA IDL, Proprietary SDS,
Existing Applications: Service and Content 'Providers'
Pricing Services
Fulfillment Services
Inventory Services
Customer Data Services
eHotels Services
Other Services


Meteo provider
ANSP2 ACC1
ANSP1 ACC1
AGDLGMS
GA operator1
AIMS
airport
NIMS


Amsterdam
Actual route
Rome

The Single European Sky

Eterogeneità – 1/2

- In ambiente distribuito vari fattori aumentano la complessità dello sviluppo di **software di qualità**:
  - le applicazioni sono distribuite su una rete di calcolatori, di caratteristiche differenti (*mainframes*, *servers*, *workstations*, *personal computers*) e dotati di hardware e di sistemi operativi diversi e spesso incompatibili
  - occorre adoperare linguaggi di programmazione diversi:
    - linguaggi di alto livello, come C# o Java, per lo sviluppo rapido della parte di presentazione di un'applicazione (la cosiddetta *presentation logic*) e per le funzionalità di un'applicazione (*business logic*)
    - linguaggi tradizionali come COBOL, C, C++ per funzionalità *legacy* o di basso livello (che richiedono maggior interazione con il S.O.)
    - linguaggi di scripting, come phyton, php, jsp, asp.net, per lo sviluppo di funzionalità server-side

Eterogeneità – 2/2

- I dati sono distribuiti su più nodi di elaborazione, memorizzati in archivi o con sistemi di gestione di basi di dati (DBMS) diversi, e in generale condivisi da applicazioni locali e remote
- Nello sviluppo di software su rete – anche in ambiente omogeneo – occorre fronteggiare problematiche quali
  - sicurezza,
  - guasti,
  - contesa e condivisione delle risorse,

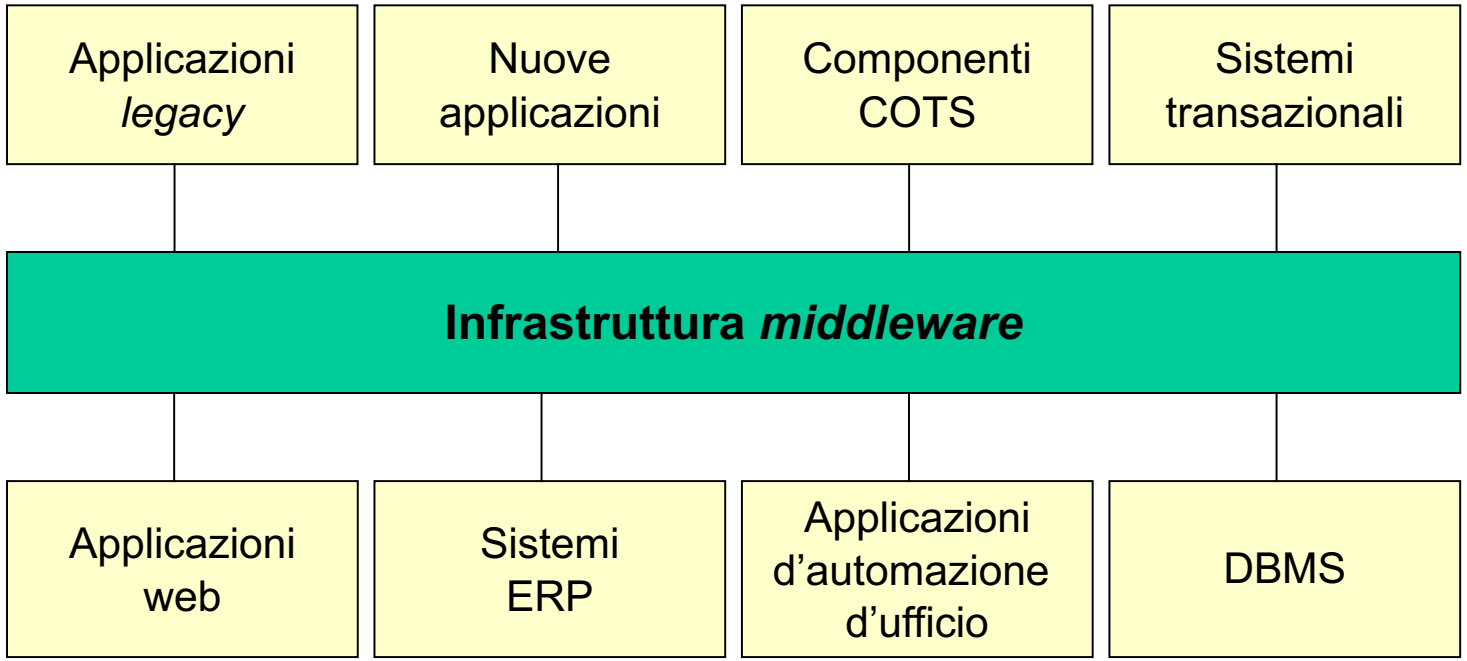
  che si presentano ben più complessi che per i sistemi centralizzati; l'eterogeneità aggiunge ulteriore complessità a questi stessi problemi

Enterprise Application Integration – 1/4

- Sistemi complessi funzionanti raramente vengono sviluppati integralmente *ex novo*; tipicamente essi evolvono a partire da sistemi esistenti già funzionanti
- L'**integrazione** di sistemi informativi sviluppati in momenti, con linguaggi e con tecniche diversi, ed operanti su piattaforme eterogenee, è un problema centrale delle tecnologie software
- Si richiede sempre più di:
  - utilizzare applicazioni di terze parti (sistemi COTS, *Commercial Off-The-Shelf*);
  - riutilizzare applicazioni esistenti (sistemi ereditati o *legacy*)

Enterprise Application Integration – 2/4

- L'integrazione di applicazioni, siano esse sistemi di nuovo sviluppo, sistemi ereditati o sistemi COTS, è da considerarsi lo scenario tipico di sviluppo software su larga scala



Enterprise Application Integration – 3/4

- L'integrazione di applicazioni (EAI) è un processo sovente più complesso dello sviluppo *ex novo*

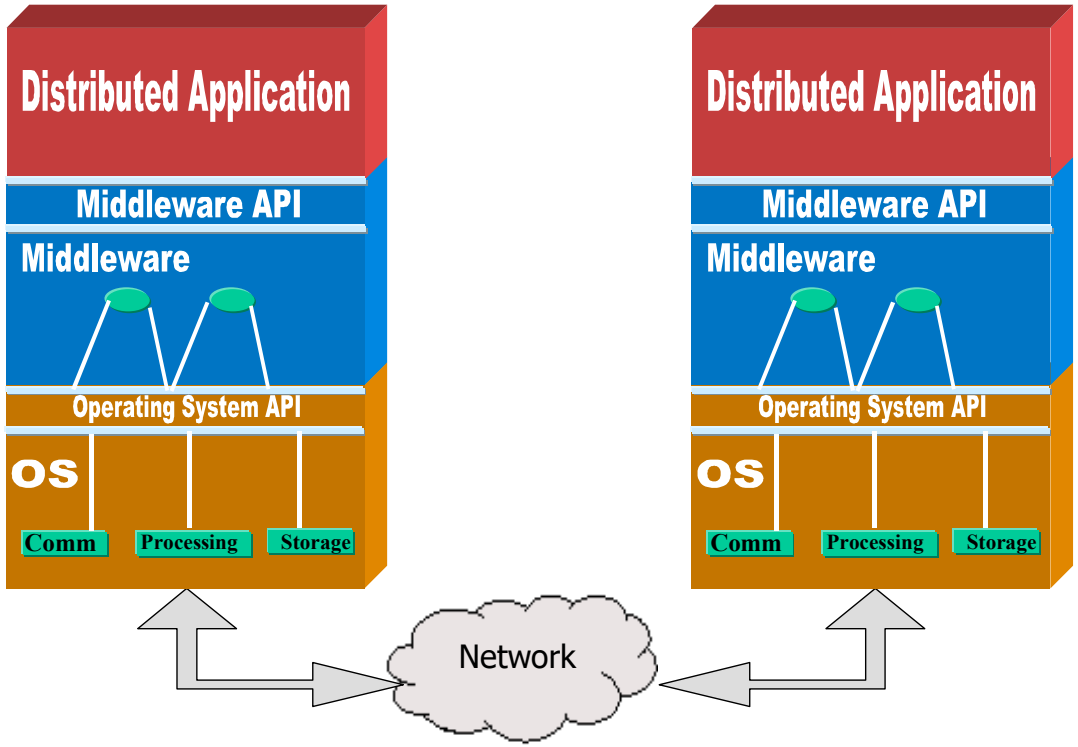
SVILUPPO DI APPLICAZIONI	INTEGRAZIONE DI APPLICAZIONI
Nuove applicazioni	Riuso di applicazioni esistenti, spesso non ben documentate, e uso di componenti COTS
Possibilità di scelta di tecnologie e prodotti	Vincoli su sistemi operativi, protocolli, linguaggi, ambienti di sviluppo ecc.; tecnologie dei sistemi esistenti spesso non interoperanti
Richiede competenze di analisi, progettazione, programmazione	Richiede competenze di architetture software, reingegnerizzazione, riuso, capacità di buona progettazione delle interfacce, adattamento di componenti, predisposizione all'evoluzione
Può essere svolto da giovani sviluppatori	Richiede familiarità con i sistemi esistenti; svolto da specialisti con anni di esperienza

Enterprise Application Integration – 4/4

- In questa prospettiva dello sviluppo software, la programmazione (intesa come codifica di moduli software in un linguaggio di programmazione) è, ancor meno che in passato, una fase critica del ciclo di vita del software, in termini sia di risorse sia di competenze tecnologiche richieste.
- Assume maggiore importanza la capacità di adattare componenti esistenti e non originariamente pensati per interoperare, mediante l'attenta progettazione delle interfacce
- Molte tecnologie *middleware* sono nate con il preciso obiettivo di fornire una risposta al problema dell'EAI

Il middleware – 1/2

- Con il termine *middleware* si intende uno strato software interposto tra il sistema operativo e le applicazioni, in grado di **fornire le astrazioni ed i servizi** utili per lo sviluppo di applicazioni distribuite



Il middleware – 2/2

- Lo strato *middleware* offre ai programmatori di applicazioni distribuite librerie di funzioni, o *middleware API* (*Application Programming Interface*), in grado di **mascherare l'eterogeneità** dei sistemi su rete
- Le piattaforme *middleware* vengono anche definite software di connettività tra applicazioni, o anche "*glue technologies*" (tecnologie collante), evidenziando la loro caratteristica di tecnologie di integrazione di applicazioni

Proprietà del livello middleware - 1/4

Il livello *middleware* può mascherare le eterogeneità mediante meccanismi di:

- **trasparenza del sistema operativo**
  - operando al di sopra del sistema operativo, i servizi forniti dalle API *middleware* possono essere definiti in maniera indipendente da esso, consentendo la portabilità delle applicazioni tra diverse piattaforme
- **trasparenza del linguaggio di programmazione**
  - la comunicazione tra componenti sviluppati il linguaggi diversi pone problemi connessi alle differenze tra i tipi di dato supportati ed alla diversità tra i meccanismi di scambio parametri nell'invocazione di sottoprogrammi;
  - il livello *middleware* può consentire l'interoperabilità, definendo un **sistema di tipi intermedio** e regole non ambigue di corrispondenza con i tipi dei linguaggi più diffusi

Proprietà del livello middleware - 2/4

- **trasparenza della locazione**
  - le risorse (dati e servizi) dovrebbero essere accessibili a livello logico, senza conoscerne la effettiva locazione fisica. E' **lo strato *middleware* che deve farsi carico della localizzazione** su rete del processo partner in una comunicazione, e del trasporto dei dati (*location transparency*)
- **trasparenza della migrazione**
  - dati e servizi possono venir rilocati durante il loro ciclo di vita. Se un componente migra, il *middleware* può consentire l'accesso a componenti mobili sulla rete, in maniera trasparente ai moduli clienti (*migration transparency)*

Proprietà del livello middleware - 3/4

- **trasparenza ai guasti**
  - una elaborazione distribuita può fallire anche solo in maniera parziale per guasti che si verificano nei singoli nodi o nel sottosistema di comunicazione
  - occorrono tecniche complesse per poter gestire uno stato globale consistente della computazione. Lo strato *middleware* può offrire al programmatore meccanismi ad alto livello per mascherare i guasti (*failure transparency*)
- **trasparenza della replicazione**
  - la replicazione di componenti è una delle tecniche più comuni per realizzare politiche di **tolleranza ai guasti**, ma può essere utilizzata anche per **migliorare le prestazioni**, attraverso un miglior bilanciamento del carico di elaborazione.
  - l'esistenza di più copie di un componente dovrebbe però essere trasparente ai suoi clienti (*replication transparency)*

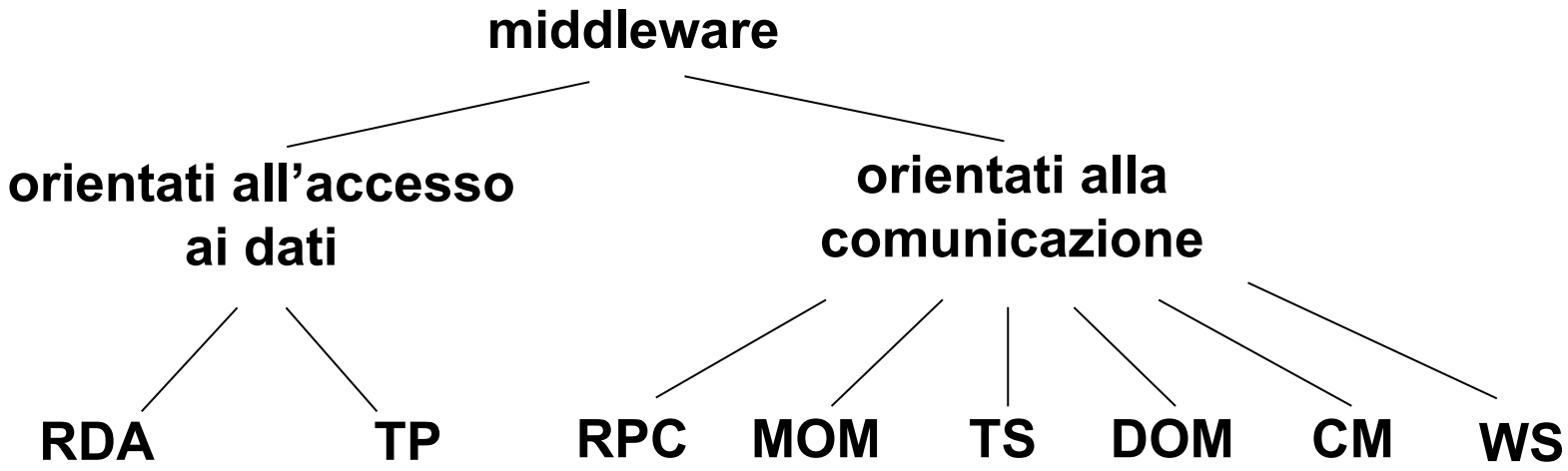
Proprietà del livello middleware - 4/4

- **trasparenza delle implementazioni commerciali**
  - alcune tecnologie *middleware* si presentano in realtà come specifiche di riferimento promosse da organizzazioni di produttori, spesso poi recepite dagli enti di standardizzazione;
  - le varie implementazioni commerciali sono quindi realizzate in maniera conforme allo standard, eventualmente differenziandosi tra loro per aspetti secondari o per servizi aggiuntivi fuori standard;
  - in questo modo è resa possibile l'interoperabilità tra applicazioni basate su realizzazioni commerciali diverse dello stesso tipo di *middleware*

Modelli di tecnologie middleware

- Nell'ultimo decennio sono state realizzate numerose tipologie di piattaforme *middleware*, sia in forma prototipale in ambito di ricerca, sia in forma commerciale
- Principali modelli di programmazione:
  - Chiamata di procedura remota (*Remote Procedure Call*, RPC)
  - Scambio messaggi (*Message-Oriented Middleware*, MOM)
  - Transazionale (*Transaction Processing*, TP)
  - Spazio delle tuple (*Tuple Space*, TS)
  - Accesso remoto ai dati (*Remore Data Access*, RDA)
  - Oggetti distribuiti (*Distributed Objects Middleware*, DOM)
  - Componenti (*Component Model*, CM)
  - Servizi web (*Web Services,* WS)

Una tassonomia dei sistemi middleware



Esempi delle varie tecnologie:

- RDA: ODBC, JDBC, Oracle DB Integrators
- TP: X/Open DTP
- RPC: SunRPC, OSF DCE RPC
- MOM: IBM MQSeries, AMQP, **JMS**, DDS
- TS: Linda, JavaSpaces, Jini
- DOM: **Java/RMI**, OMG CORBA, MS DCOM, .NET remoting
- CM: OMG CCM, EJB
- WS: JAX-WS, MS WCF

Il modello di chiamata di procedura remota

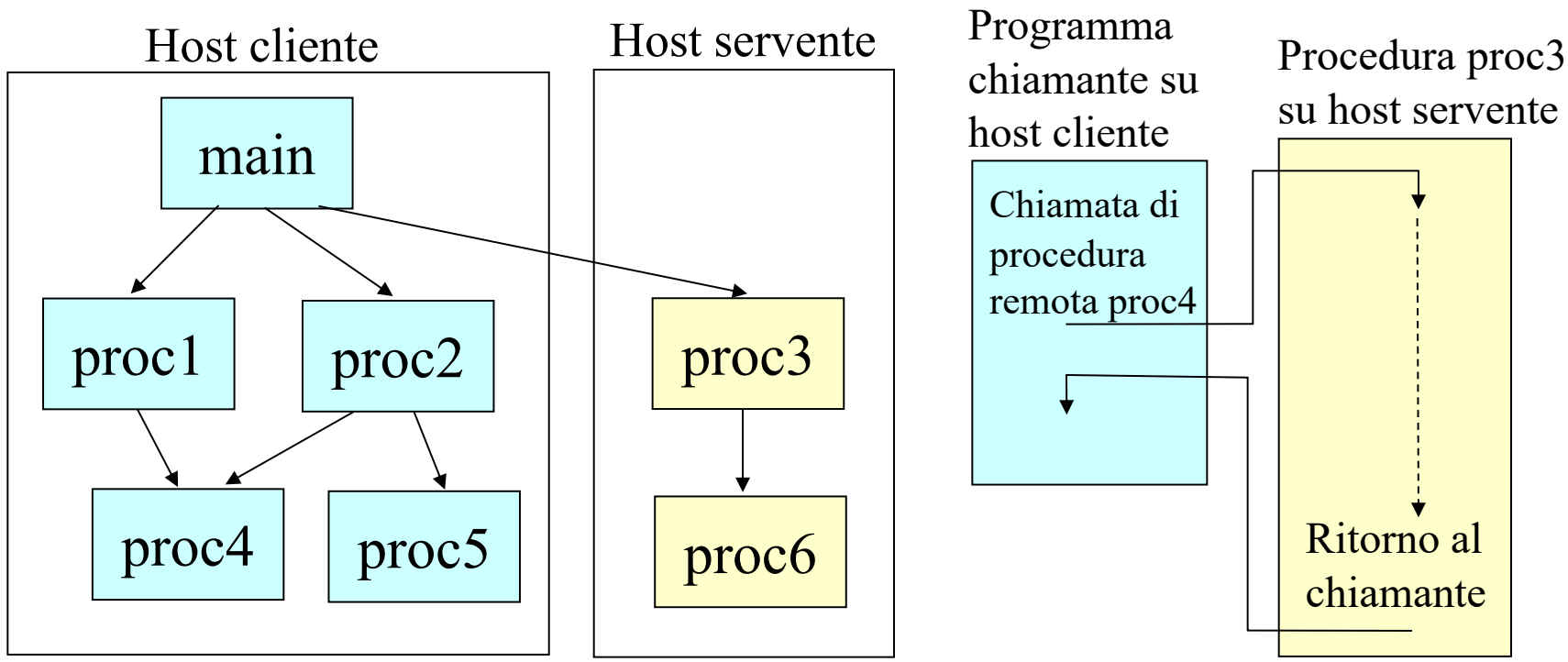
(Remote Procedure Call, RPC)

Il modello RPC

- Il modello di chiamata di procedura remota o RPC è un'estensione ai sistemi distribuiti del modello procedurale dei linguaggi di programmazione tradizionali

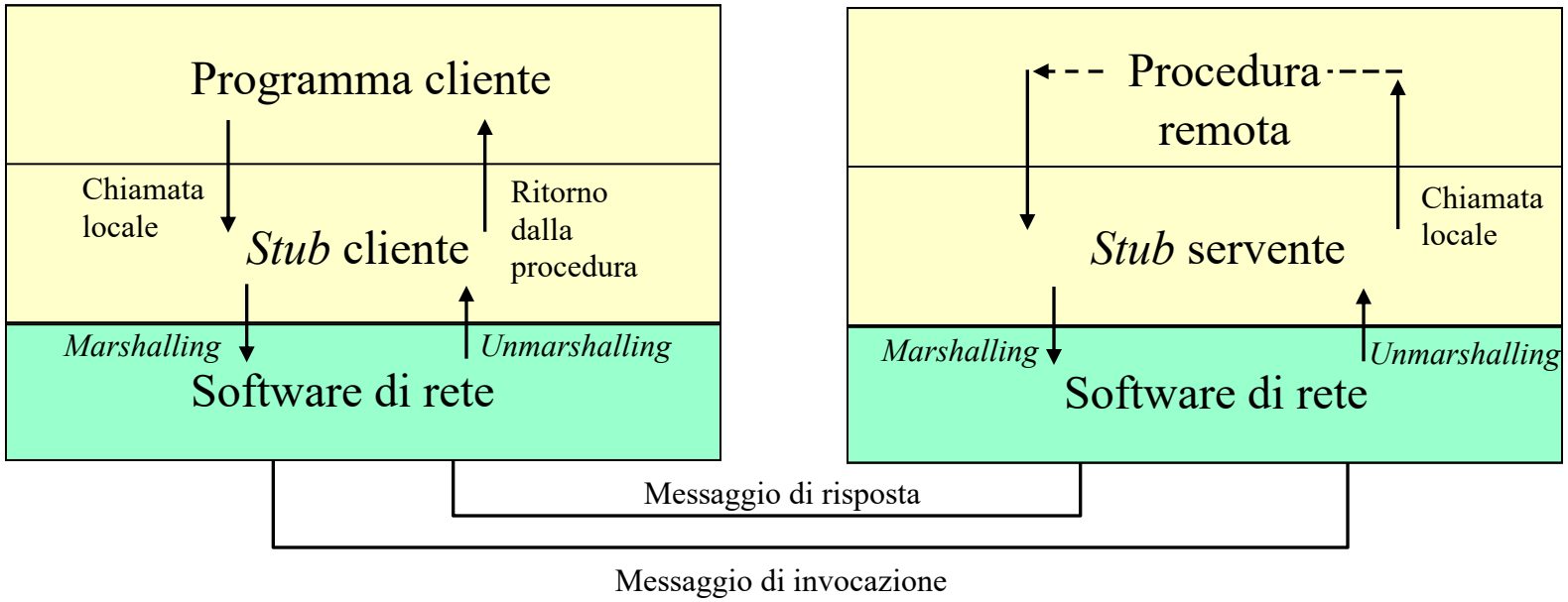
- Nei sistemi RPC la comunicazione tra processi è intrinsecamente di tipo **sincrono e bloccante**: il processo chiamante resta sospeso fino al completamento della procedura remota. La disponibilità di meccanismi asincroni (non bloccanti) nei sistemi RPC è da considerarsi un'eccezione.

Il modello RPC



Funzionamento del meccanismo RPC

- In fase di compilazione vengono automaticamente generate speciali funzioni di interfaccia dette *stub*, una dal lato cliente ed una dal lato servente, che vengono poi collegate con le due parti dell'applicazione.
- Gli *stub* implementano in maniera in gran parte trasparente al programmatore lo scambio parametri e la comunicazione attraverso la rete tra il programma chiamante (cliente) e la procedura remota chiamata (servente). A tal fine gli *stub* accedono direttamente ai servizi di trasporto dei protocolli di rete.



Il ruolo degli stub

- Lo *stub* cliente:
  - preleva i parametri di scambio dal chiamante
  - li impacchetta in un apposito messaggio, che affida al software di rete affinché sia trasmesso alla macchina dove risiede il servente
  - attende il messaggio di risposta che indica il completamento della procedura remota;
  - spacchetta il messaggio prelevando i valori dei parametri di uscita
  - restituisce i parametri al programma chiamante, che riprende controllo
- Lo *stub* servente:
  - attende (dal software di rete) un messaggio di invocazione della procedura
  - spacchetta il messaggio, prelevando i parametri di scambio
  - trasmette i parametri alla procedura chiamata, cui cede il controllo
  - riprende il controllo al termine della procedura, ed impacchetta i parametri di uscita in un messaggio di risposta al cliente
  - invia il messaggio di risposta e si mette nuovamente in attesa di una richiesta
- Il programma chiamante invoca una procedura a tutti gli effetti locale (*stub* cliente), ed analogamente la procedura remota viene invocata come una procedura locale da parte dello *stub* servente.

Marshalling dei parametri

- L'operazione di impacchettamento (spacchettamento) dei parametri della chiamata di procedura, svolta dagli stub, prende il nome di *marshalling* (*unmarshalling*)
- Il *marshalling* dei parametri consiste di:
  - una conversione di formato dei dati, per tener conto delle differenze di rappresentazione tra cliente e servente: in generale le due piattaforme sono eterogenee, ed è possibile che adottino rappresentazioni interne diverse. Per es.:
    - interi in complementi alla base o complementi diminuiti
    - a parità di rappresentazione, ordinamento dei byte *big endian* o *little endian*
  - una serializzazione dei dati, che vengono trasformati in sequenze di byte ed impacchettati, secondo un formato compreso da cliente e servente. In particolare occorre linearizzare i dati strutturati, quali *array* e *record*

Marshalling dei parametri

- In generale, esistono i seguenti metodi per consentire a due nodi (*sender* e *receiver*) di scambiare dei dati:
  - i dati sono convertiti dal sender in un **formato esterno** (**concordato** tra sender e receiver), e quindi trasmessi dal sender; il receiver converte i dati nel proprio formato locale:
    - in tal caso si parla di **"external data representation".**
  - i dati sono trasmessi nel formato del sender, e sono accompagnati da un'indicazione sul formato utilizzato.
- Alcuni approcci di external data representation / marshalling:
  - CORBA Common Data Representation (CDR);
  - Sun XDR (utilizzato in Sun RPC);
  - **Java object serialization**;
  - XML (Extensible Markup Language): rappresentazione dati in formato testuale;

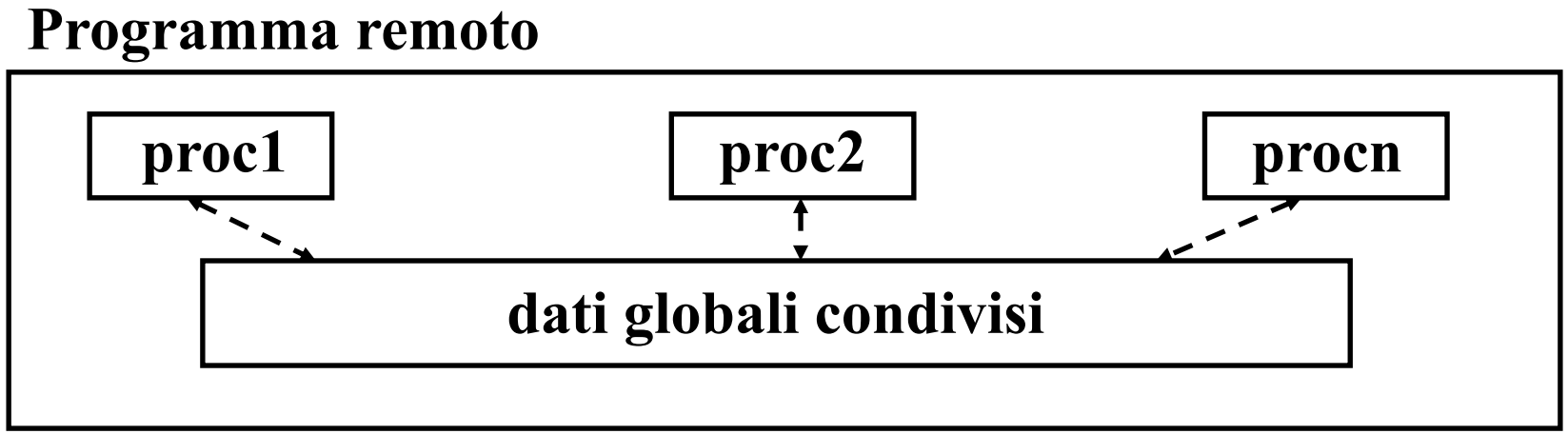
Semantica delle RPC

- Malfunzionamenti possibili
  - nella rete o nei singoli nodi

  che possono causare:
  - Perdita del messaggio di richiesta
  - Perdita del messaggio di risposta
  - Caduta del nodo servente dopo la ricezione della richiesta ma prima dell'invio della risposta
- Semantiche possibili per le RPC:
  - **Exactly once**: la procedura viene eseguita una ed una volta sola;
  - **At most once**: non è possibile garantire che la procedura sia stata eseguita ma, se è stata eseguita, lo è stata una volta sola;
  - **At least once**: la procedura viene eseguita almeno una volta;
  - **Zero or more**: non è possibile dire *se* e *quante volte* la procedura è stata eseguita.

Un middleware RPC: Sun RPC – 1/5

- L'implementazione RPC di Sun Microsystems prevede come entità in esecuzione su una macchina servente un programma, che contiene una o più procedure invocabili remotamente
- Le procedure all'interno di uno stesso programma operano nello stesso ambiente d'esecuzione, quindi possono condividere variabili
- E' possibile avere attive su un nodo più versioni dello stesso programma
- Una tripla di interi (programma, versione, procedura) è sufficiente al cliente per identificare la procedura da invocare



Un middleware RPC: Sun RPC – 2/5

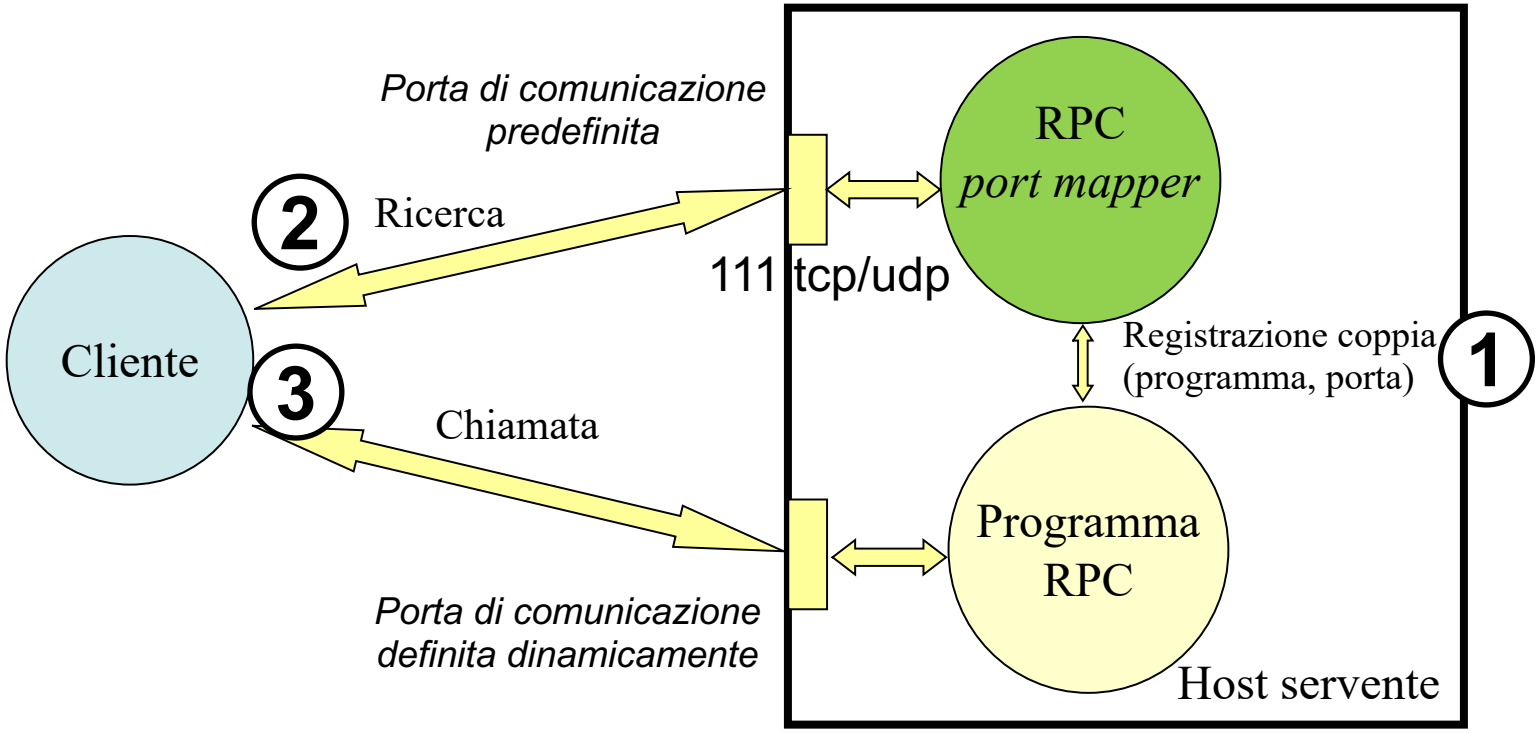
- E’ garantita la mutua esclusione tra le procedure all’interno di uno stesso programma remoto:
  - Al più una procedura può essere in esecuzione in un dato istante, anche in presenza di più clienti
- Per l’esternalizzazione dei dati viene adottato lo standard XDR (eXternal Data Representation)
- Sun RPC si basa su TCP o UDP
  - UDP: se il cliente riceve una risposta, la chiamata è stata eseguita almeno una volta **(semantica *at least once*)**
  - UDP: se il cliente non riceve risposta, la chiamata è stata eseguita zero o più volte **(semantica *zero or more*)**
  - TCP: se il cliente riceve una risposta, la chiamata è stata eseguita una volta **(semantica *exactly once*)**
  - TCP: se il cliente non riceve risposta, la chiamata è stata eseguita zero una volta **(semantica *at most once*)**
- Nel caso si usi UDP le procedure devono quindi essere **idempotenti**

Un middleware RPC: Sun RPC – 3/5

- In generale, nella programmazione con socket TCP/UDP, **cliente e servente devono concordare preventivamente il numero del porto** sul quale il servente è in ascolto, perché entrambi consultano una lista pubblica e prefissata di assegnazione dei porti
- Sun RPC adotta invece un meccanismo di ***binding* dinamico**
  - *Il servente ottiene **dinamicamente** un numero di porto disponibile*
  - *Il cliente non può dunque conoscere il porto finchè il server non viene attivato*
  - *Sul nodo server gira un processo **RPC port mapper** che gestisce una lista dei serventi attivi*
  - *All'attivazione di un programma RPC, il port mapper registra la coppia **(numero programma, numero porto)***
  - ***Il port mapper opera su una socket ad un porto prefissato (111), uguale per ogni nodo server***
  - *Il cliente deve contattare il port mapper del server prima di eseguire la RPC (presenta il numero programma e riceve il numero porto)*
- **Il *binding* dinamico consente la trasparenza della locazione**

Un middleware RPC: Sun RPC – 4/5

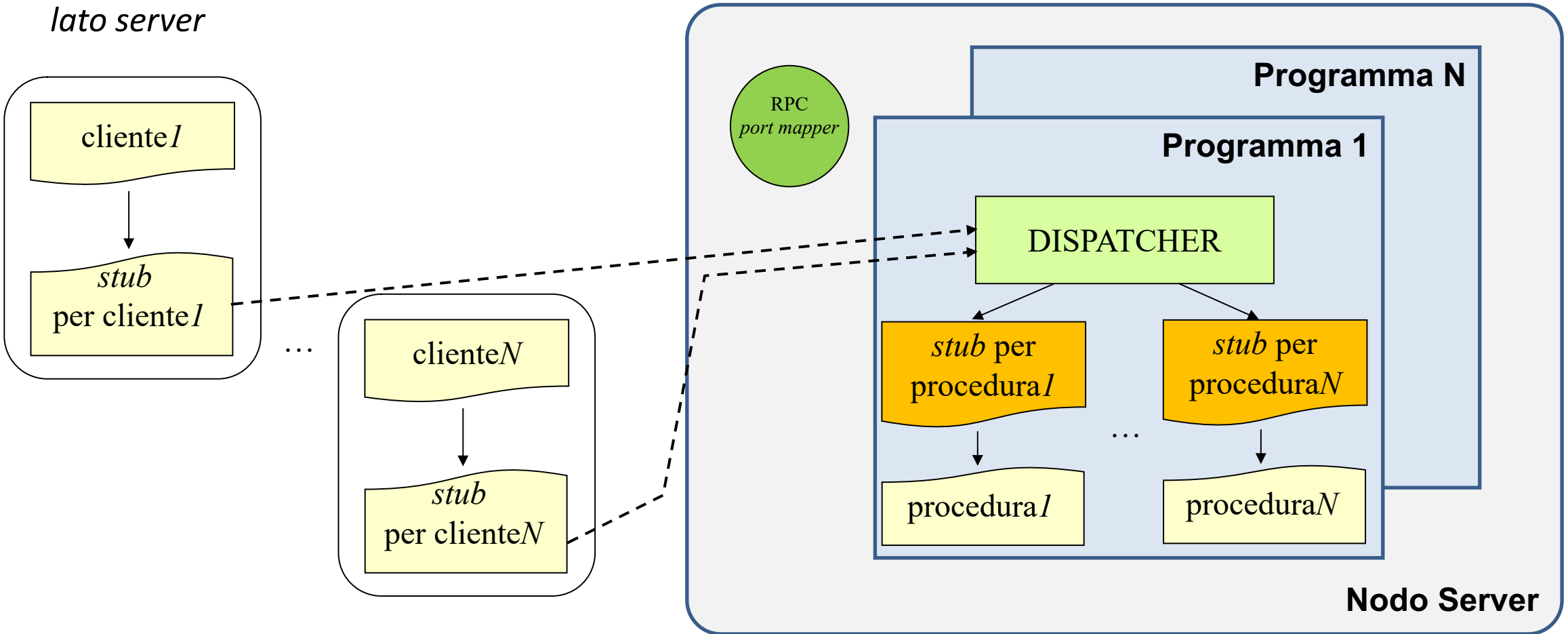
- Algoritmo del ***port mapper***
  - Crea una socket a un porto predefinito (111)
  - Accetta indefinitamente richieste di
    - Registrazione di un programma RPC (dai serventi)
    - Ricerca del porto di un programma RPC (dai clienti)



Un middleware RPC: Sun RPC – 5/5

- Una volta che il client ha trovato il porto da utilizzare per la *RPC call,* effettuerà la chiamata al proprio *stub locale* che automaticamente andrà a contattare lo stub opportuno tramite un componente chiamato ***dispatcher***

- Infatti, sul nodo servente il ***dispatcher*** si occuperà di inoltrare i messaggi dei clienti agli opportuni *stub lato server*

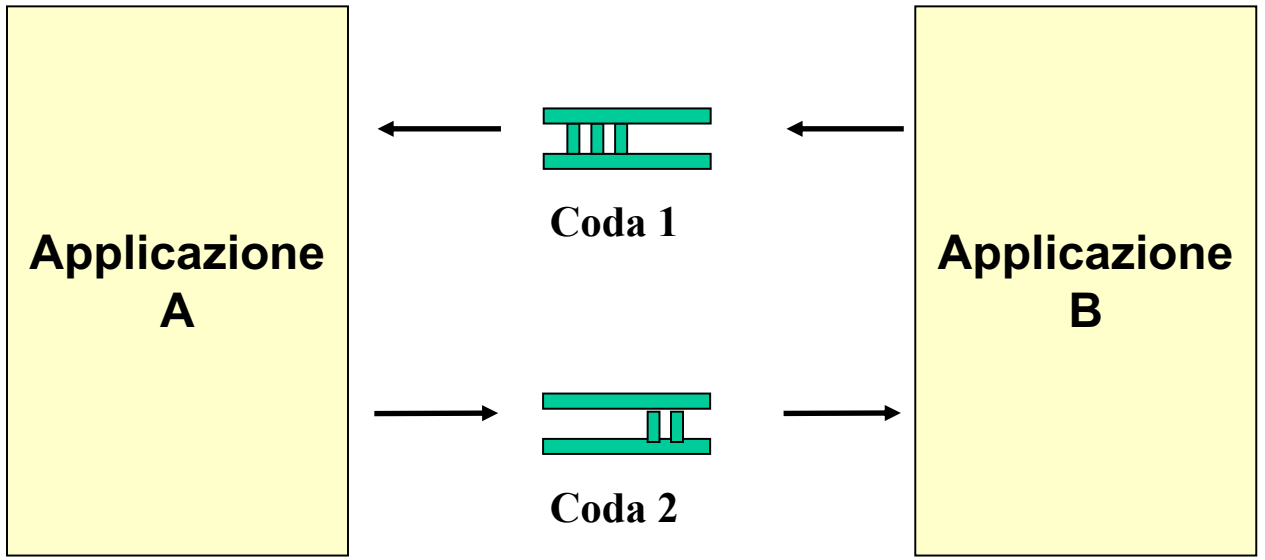


Il modello orientato ai messaggi

(*Message Oriented Middleware*, MOM)

Il modello a scambio messaggi – 1/3

- I sistemi *middleware* a **scambio messaggi** (*Message Oriented Middleware*, MOM) si basano sull'astrazione di una coda di messaggi tra processi interoperanti su rete, che rappresenta una generalizzazione del noto concetto di *mailbox*, tipico dei sistemi operativi



Il modello a scambio messaggi – 2/3

- Nel modello MOM la comunicazione è non è di tipo cliente-servente, ma tra pari (***peer-to-peer***), di tipo produttore-consumatore, e tipicamente **asincrona**, sebbene alcuni sistemi consentano anche uno scambio di messaggi di tipo sincrono

Il modello a scambio messaggi – 3/3

- I sistemi MOM possono adottare un **modello publish-subscribe**, in cui i processi
  - produttori **pubblicano messaggi differenziati per tipo**
  - consumatori **possono dichiararsi interessati ai messaggi in base al loro tipo**
- Le piattaforme MOM sono adatte per applicazioni **guidate dagli eventi** (***event-driven***)
  - quando si verifica un evento, un processo produttore affida al *middleware* la responsabilità di **prendere in consegna** un messaggio ed **inoltrarlo** o semplicemente **notificarne la disponibilità** ai processi interessati al suo consumo
  - Molti sistemi MOM realizzano anche meccanismi di **persistenza dei messaggi**

Il modello a oggetti distribuiti

(*Distributed Objects Middleware*, DOM)

Il modello a oggetti distribuiti – 1/4

- Il **modello a oggetti distribuiti** è un'estensione in ambiente distribuito delle tecniche di programmazione ad oggetti
- Schematizzando:

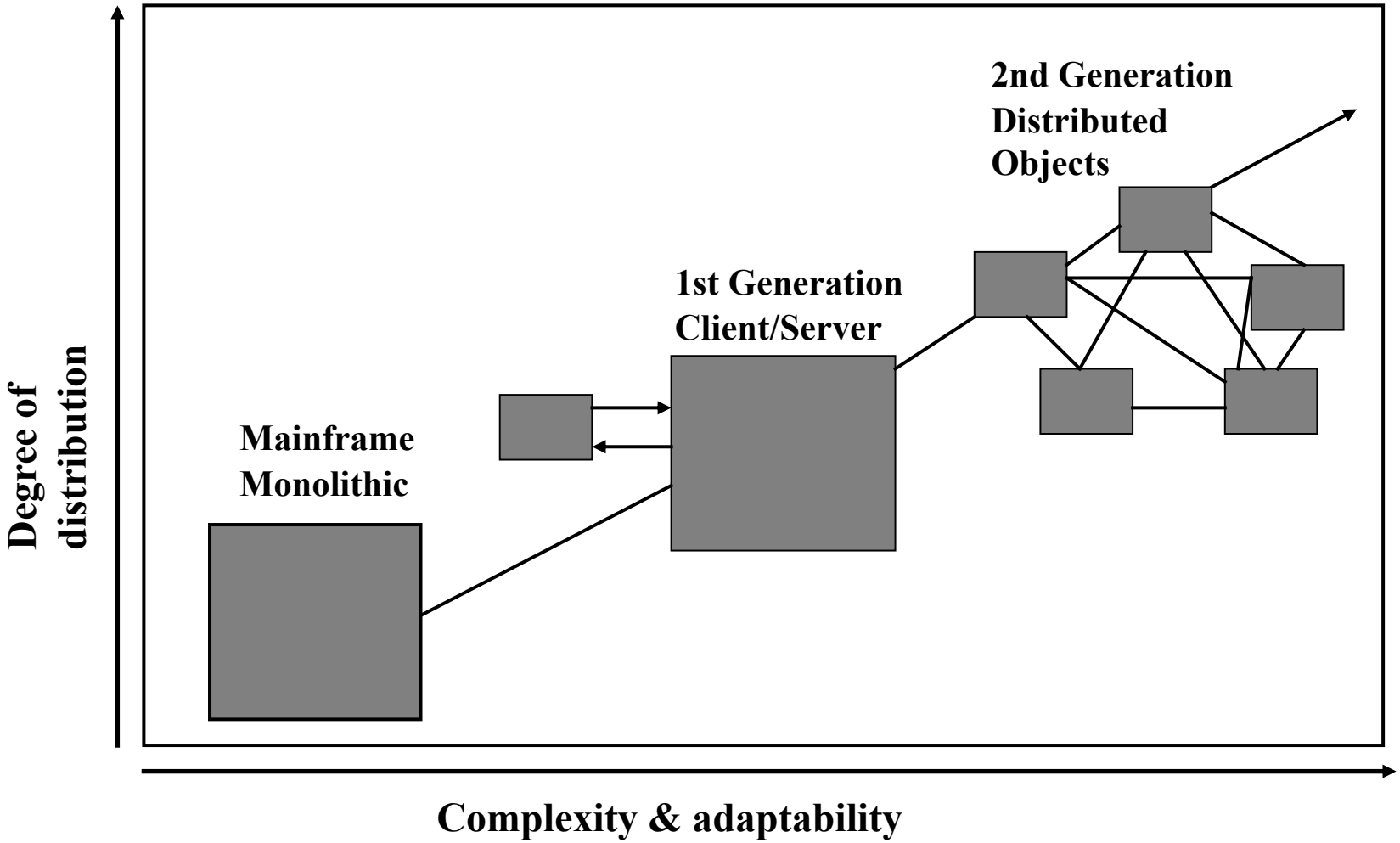
DO = OOP + C/S

Distributed objects = object-oriented programming + client-server

- L'utilizzo di una piattaforma *Distributed Object Middleware* (DOM) permette di **invocare un metodo su di un oggetto remoto come fosse locale**
- Esempi di tecnologie DOM:
  - Java RMI
  - OMG CORBA

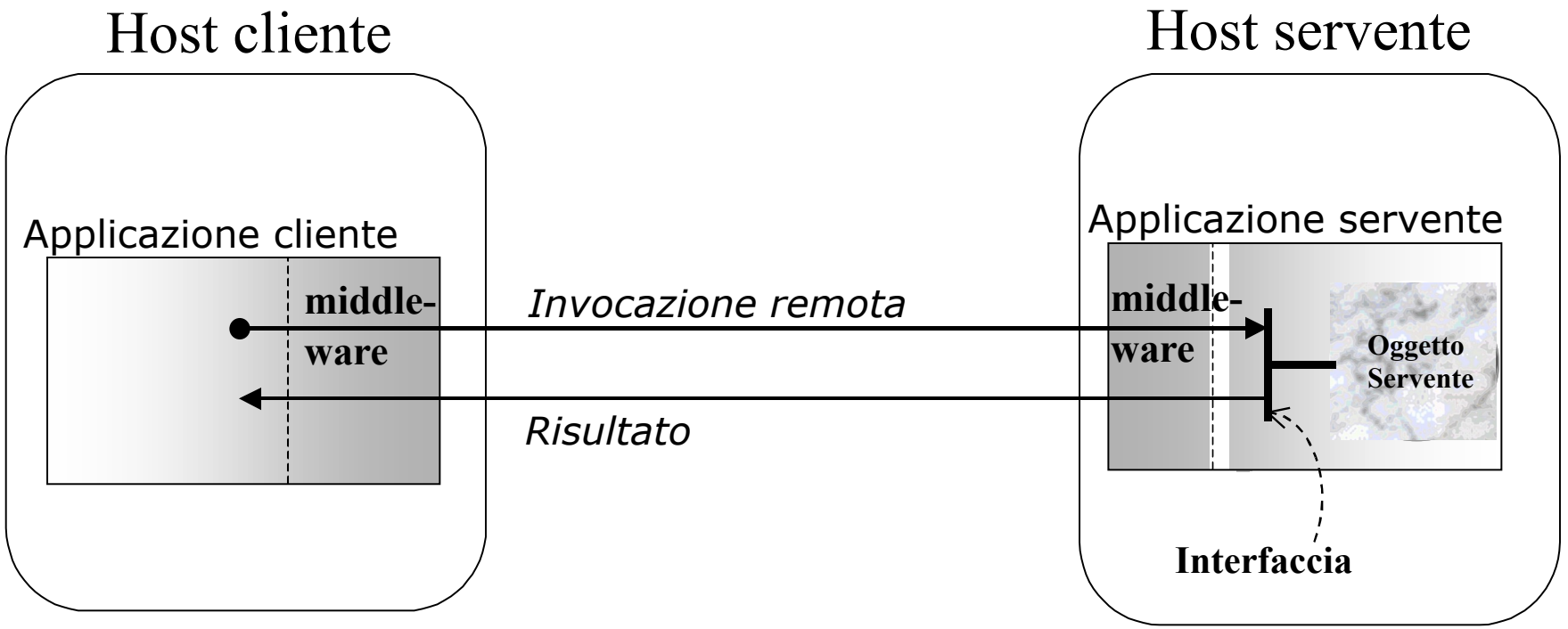
Il modello a oggetti distribuiti – 2/4

- Viene riguardato come la seconda generazione di sistemi software distribuiti



Il modello a oggetti distribuiti – 3/4

- I componenti di un'applicazione distribuita sono oggetti che risiedono su macchine diverse e comunicano mediante invocazione di metodi

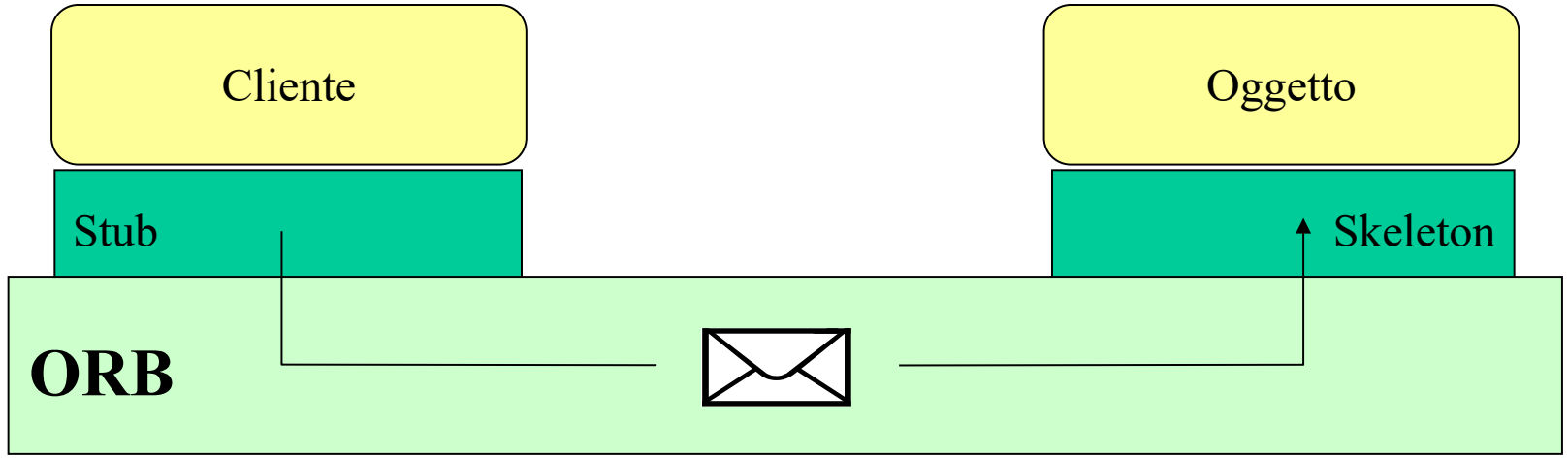


Il modello a oggetti distribuiti - 4/4

- **Il meccanismo di invocazione di un metodo su di un oggetto remoto è analogo a quello RPC.**
- Il programmatore applicativo viene sollevato dall'onere di gestione della complessità dovuta alla distribuzione dei componenti su piattaforme eterogenee, sviluppando software ad oggetti in maniera (il più possibile) simile al caso sequenziale
- Presenta strumenti di sviluppo simili al caso RPC:
  - **procedure stub**;
  - **Interface Definition Language** (**IDL**) e relativi compilatori;

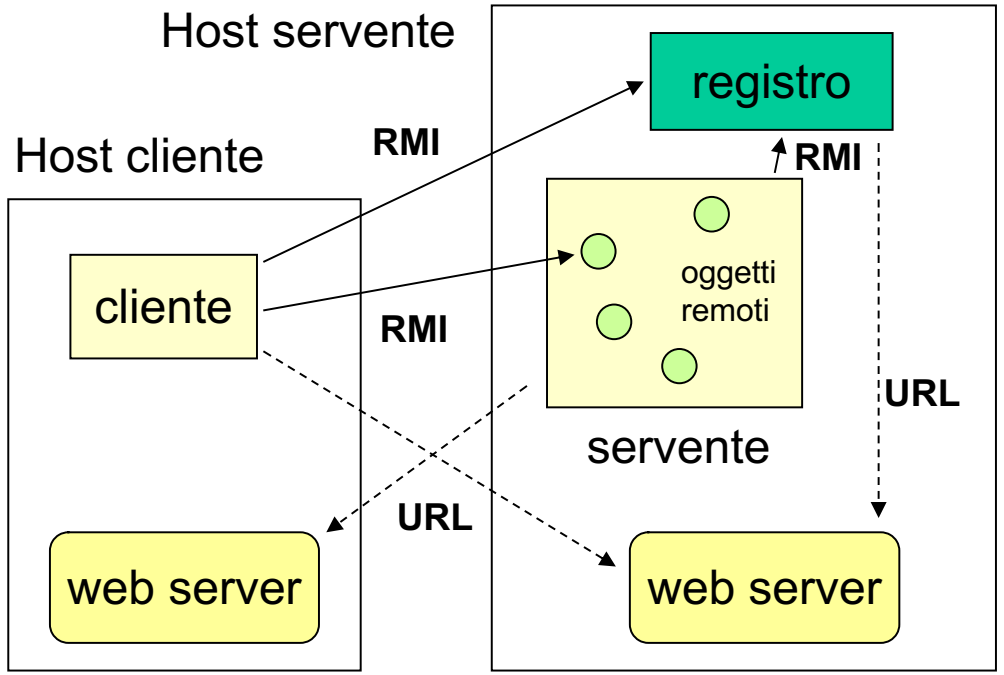
Object Request Broker (ORB)

- Le piattaforme Distributed Object Middleware si presentano spesso sotto forma di **Object Request Brokers (ORB)**
- Un **ORB** è una sorta di bus software per la comunicazione tra oggetti su rete (esempi di ORB OMG CORBA, Microsoft COM):
  - Al programmatore l'ORB si presenta come un insieme di interfacce applicative (API);
  - Supporta l'interazione client-oggetto (stub-ORB-skeleton).



Java RMI – 1/2

- Java RMI è un middleware a oggetti
  - **Specifico per la piattaforma Java**
  - Ambiente di elaborazione distribuita **omogeneo** (macchine virtuali Java)
- I processi serventi creano oggetti e ne registrano il nome in un **catalogo** (***RMI registry***)
- I clienti ottengono **riferimenti agli oggetti remoti** – eventualmente tramite ricerca nel catalogo – e ne invocano i metodi



Java RMI – 2/2

- RMI fornisce i meccanismi per
  - **scambiare** oggetti come parametri o valori di ritorno
  - **trasferire** su richiesta il codice di un oggetto
  - **serializzare** e trasferire lo stato di un oggetto

- Tali meccanismi consentono di realizzare sistemi con codice mobile