Dipartimento di Informatica
e Sistemistica Antonio Ruberti

Sapienza
Università di Roma

Architetture Software Orientate ai Servizi

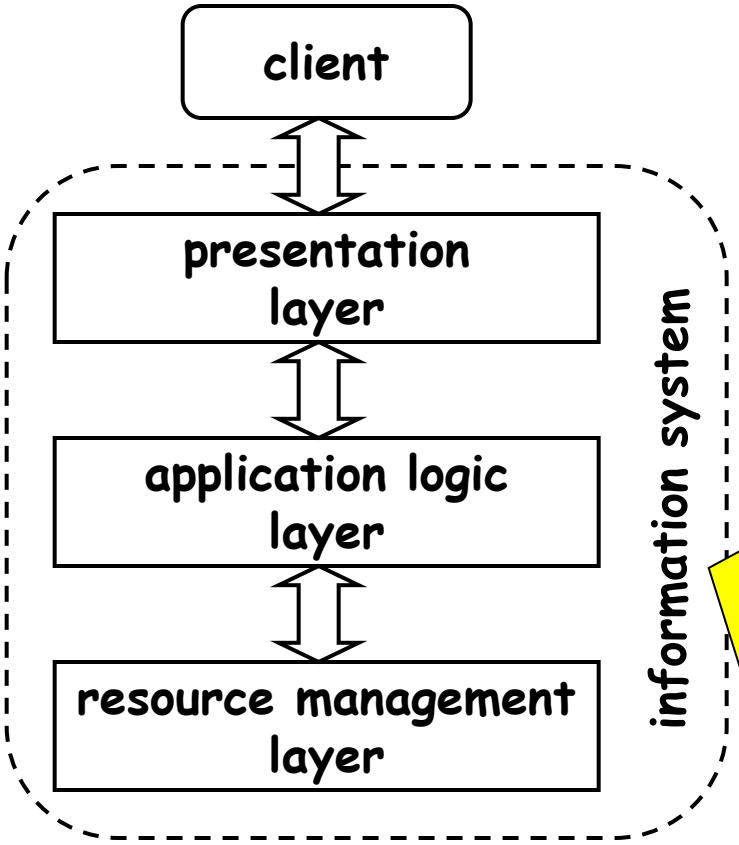
Concetti di Base di Sistemi Informativi Distribuiti & Middleware & Web Technologies

Le figure presentate in queste slide sono Copyright Springer Verlag Berlin Heidelberg 2004. Esse sono state fornite dagli autori al docente per soli scopi didattici.

The figures presented in these slides are Copyright Springer Verlag Berlin Heidelberg 2004. They have been provided directly by the authors for teaching purposes.

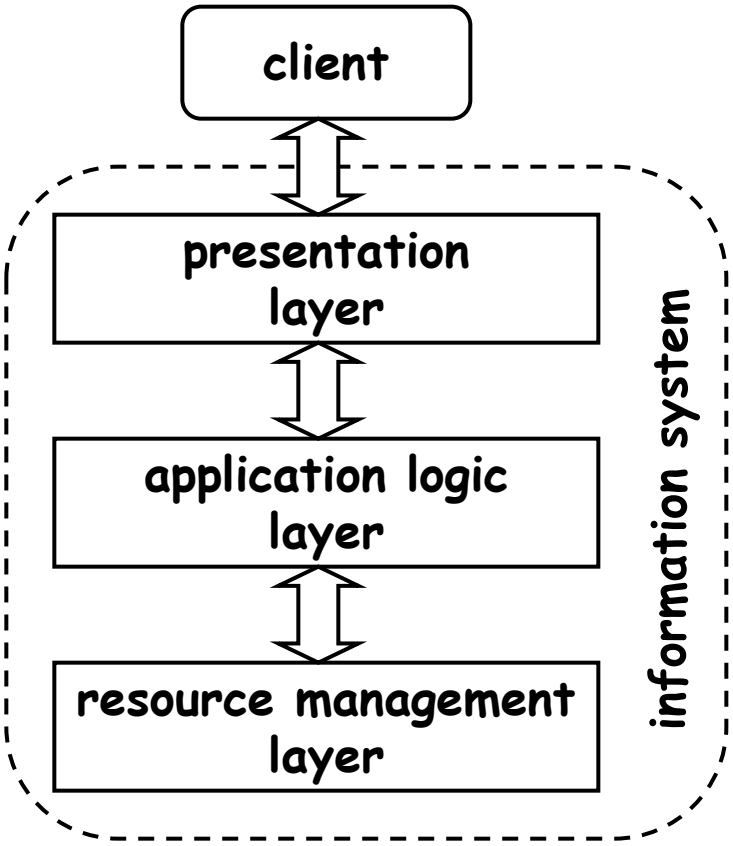
Layer di un Sistema Informativo

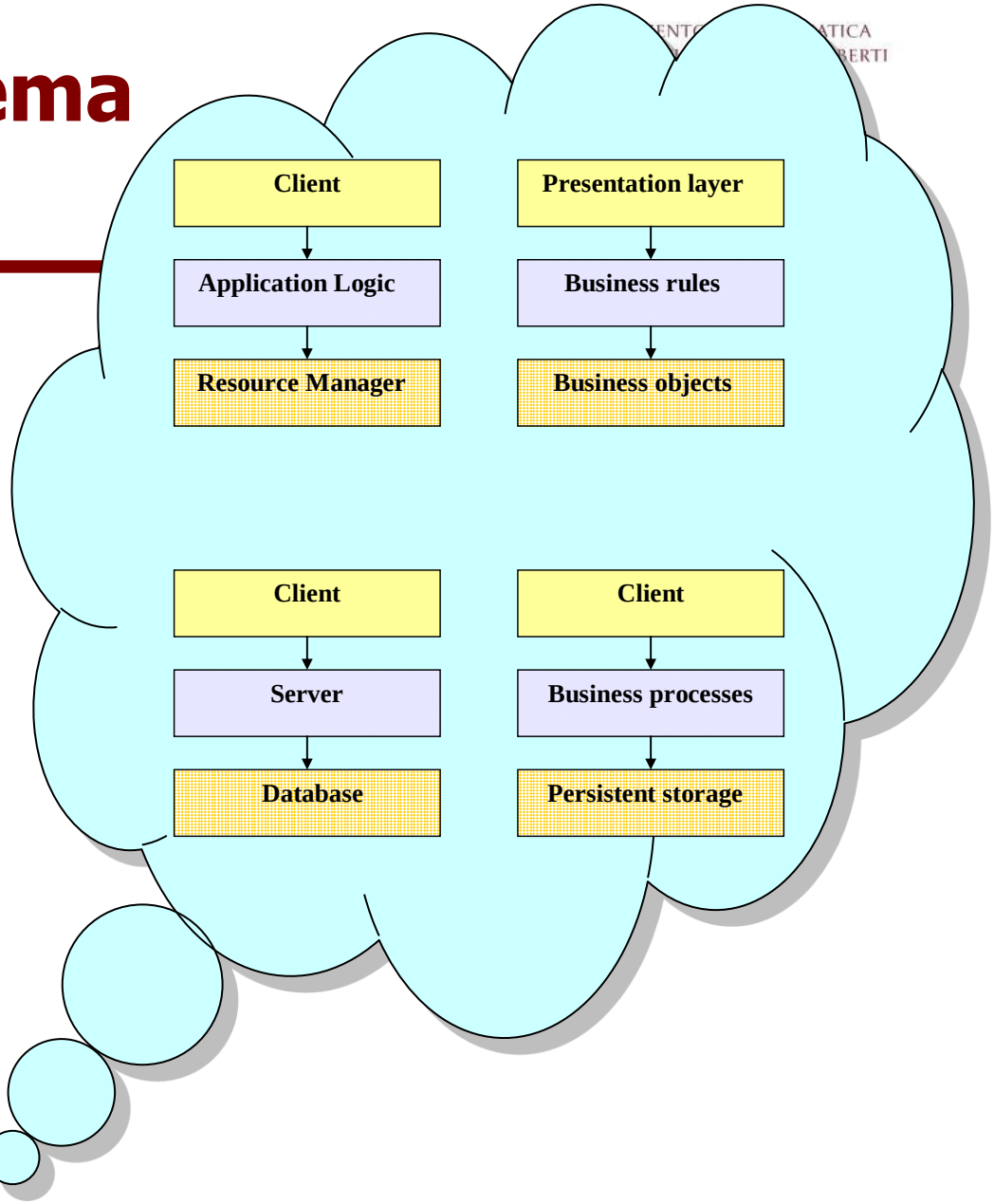
Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma



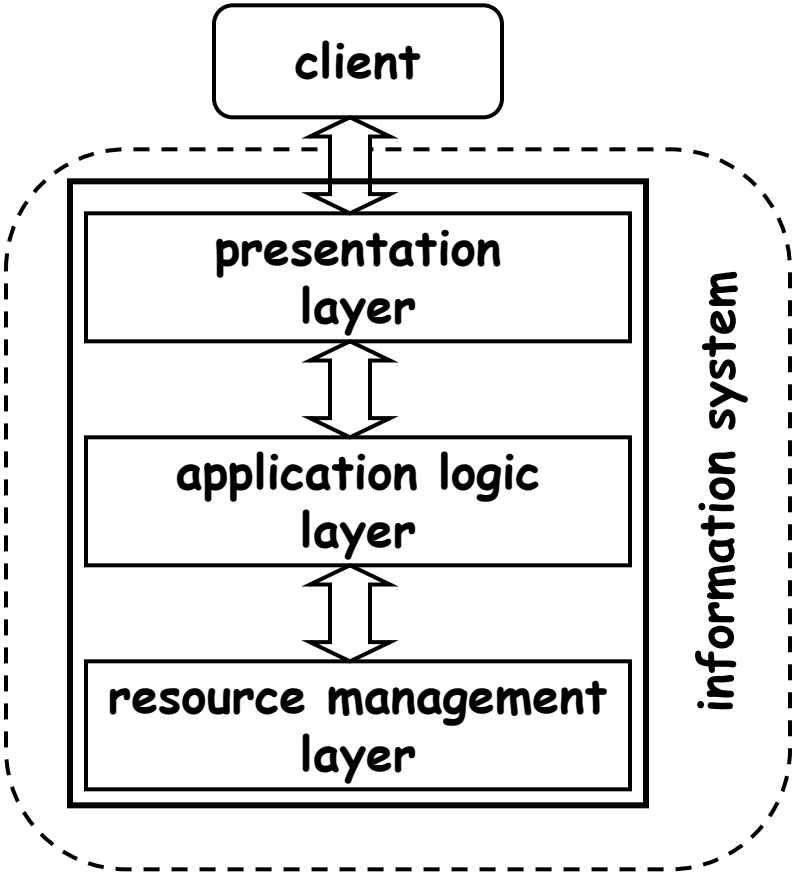
Client is any user or program that wants to perform an operation over the system. Clients interact with the system through a presentation layer
The application logic determines what the system actually does. It takes care of enforcing the business rules and establish the business processes. The application logic can take many forms: programs, constraints, business processes, etc.
The resource manager deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic. This is typically a database but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence

Layer di un Sistema Informativo





Architettura 1-tier

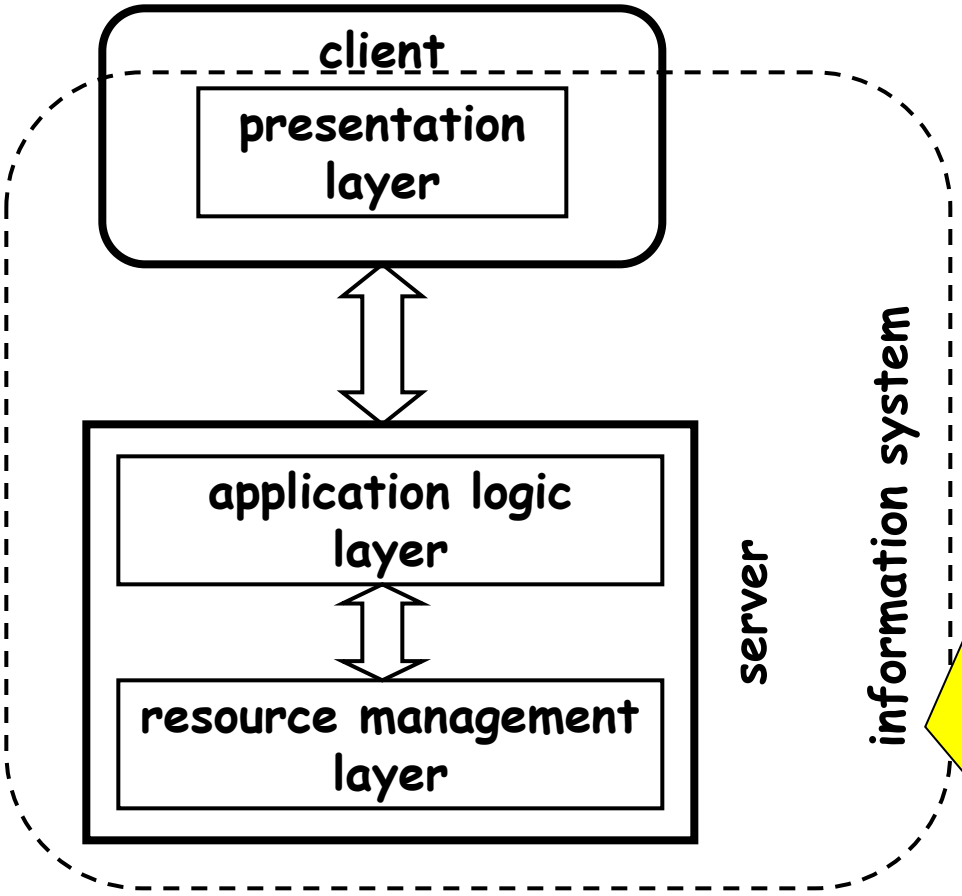


no forced context switches in the control flow (everything happens within the system)

all is centralized, managing and controlling resources is easier

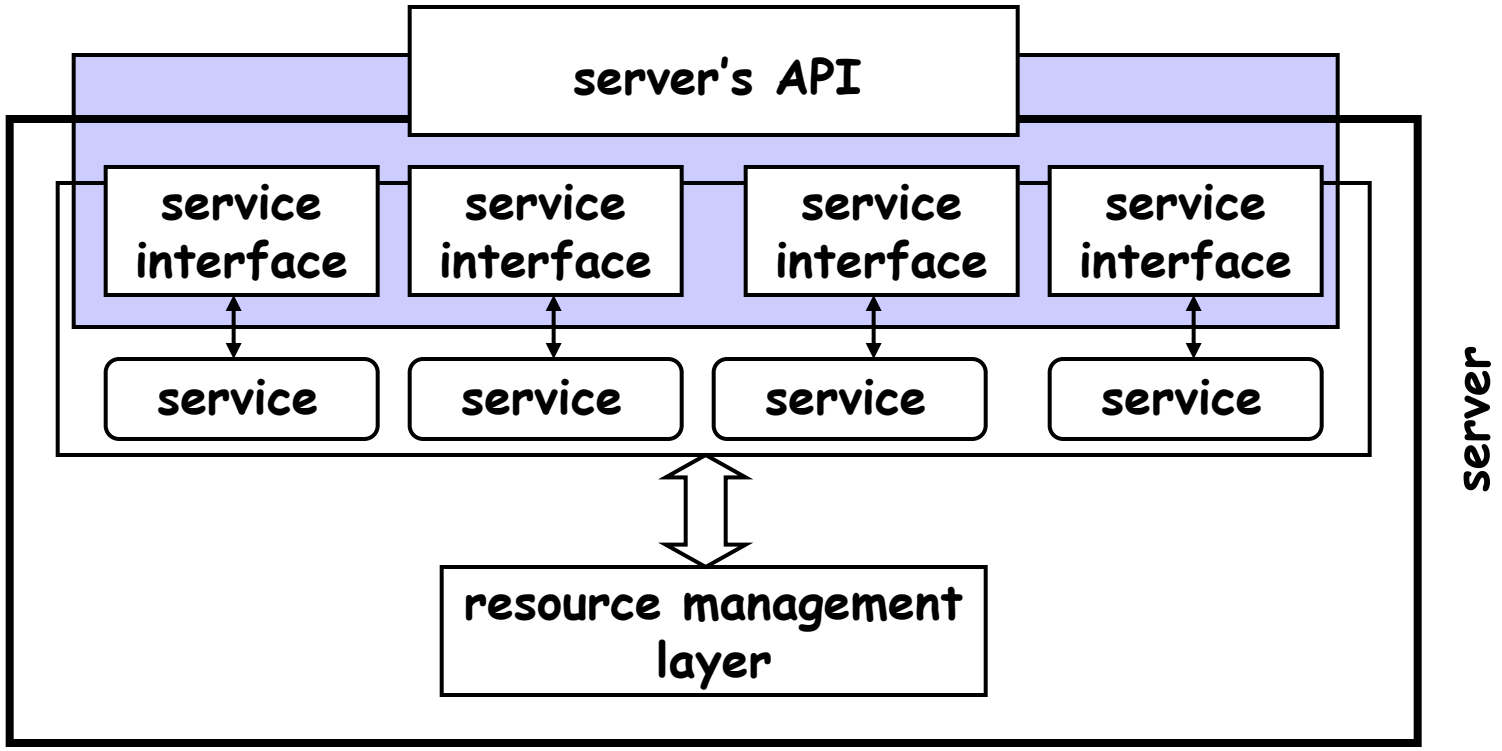
the design can be highly optimized by blurring the separation between layers.

Architettura 2-tier

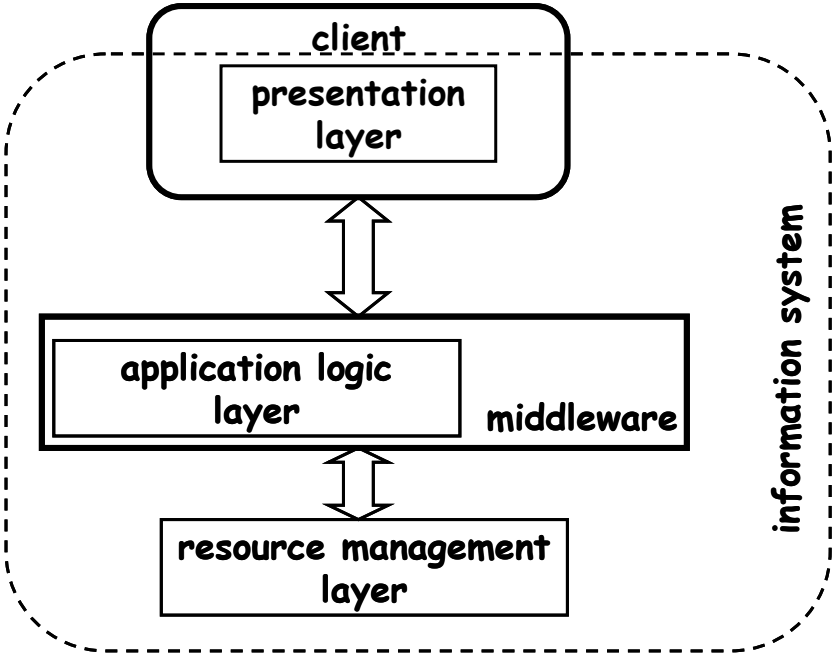


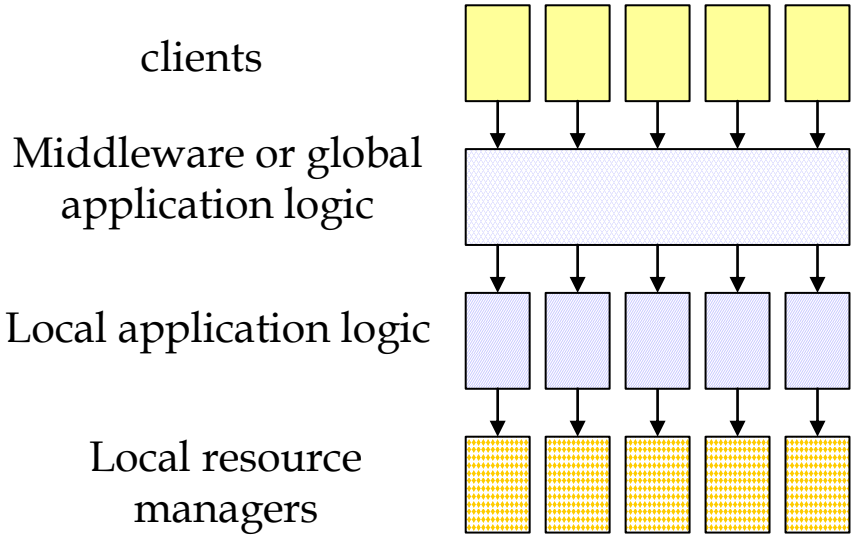
Clients are independent of each other: one could have several presentation layers depending on what each client wants to do.
One can take advantage of the computing power at the client machine to have more sophisticated presentation layers. This also saves computer resources at the server machine.
It introduces the concept of API (Application Program Interface). An interface to invoke the system from the outside. It also allows designers to think about federating the systems into a single system.
The resource manager only sees one client: the application logic. This greatly helps with performance since there are no client connections/sessions to maintain.

Servizi ed Interfacce



Architettura 3-tier

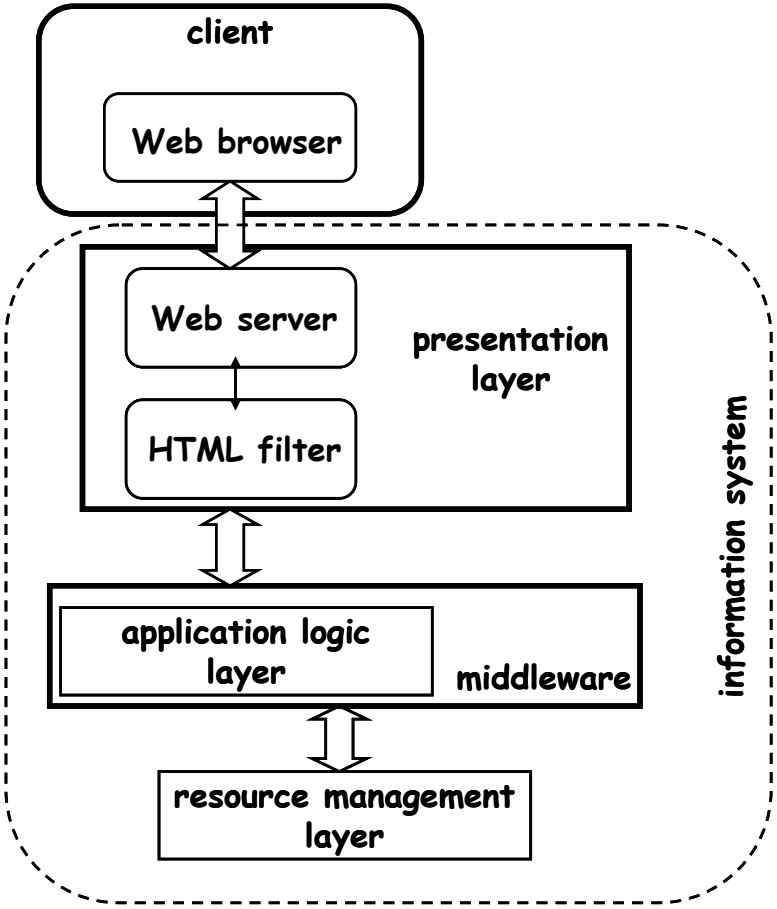


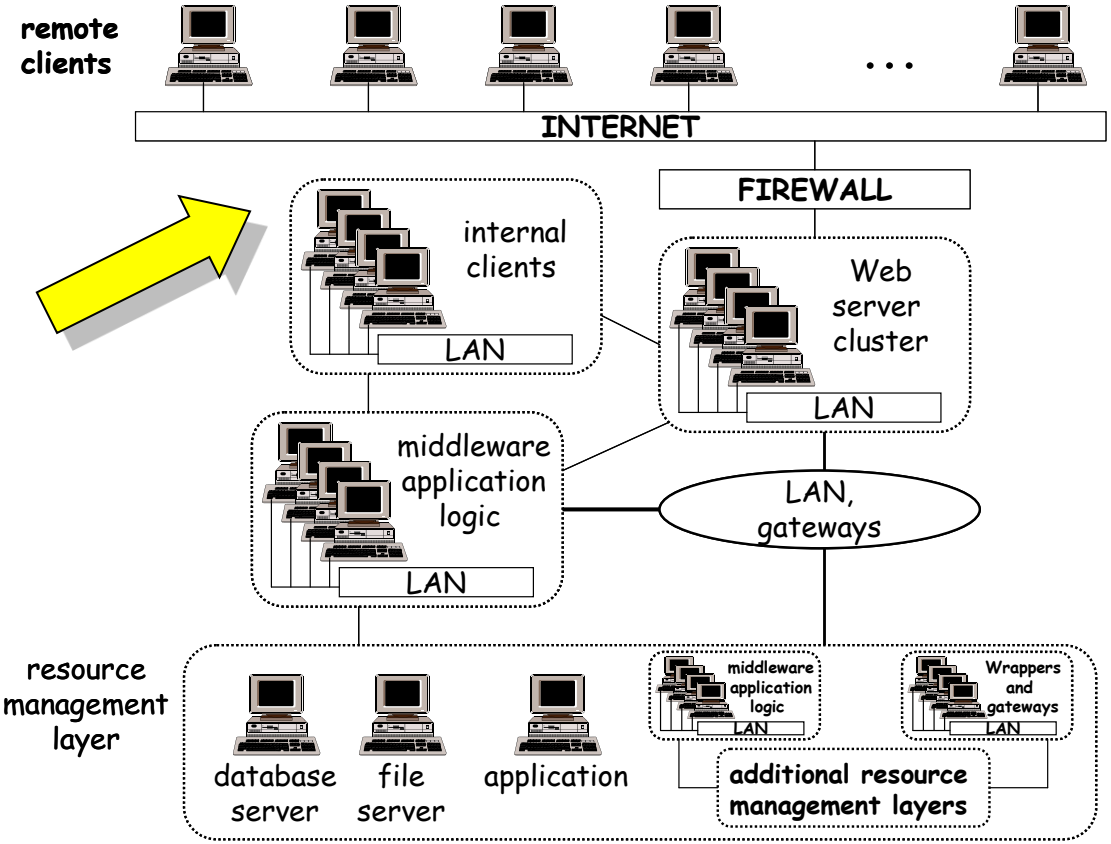


Middleware is just a level of indirection between clients and other layers of the system.
It introduces an additional layer of business logic encompassing all underlying systems.

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

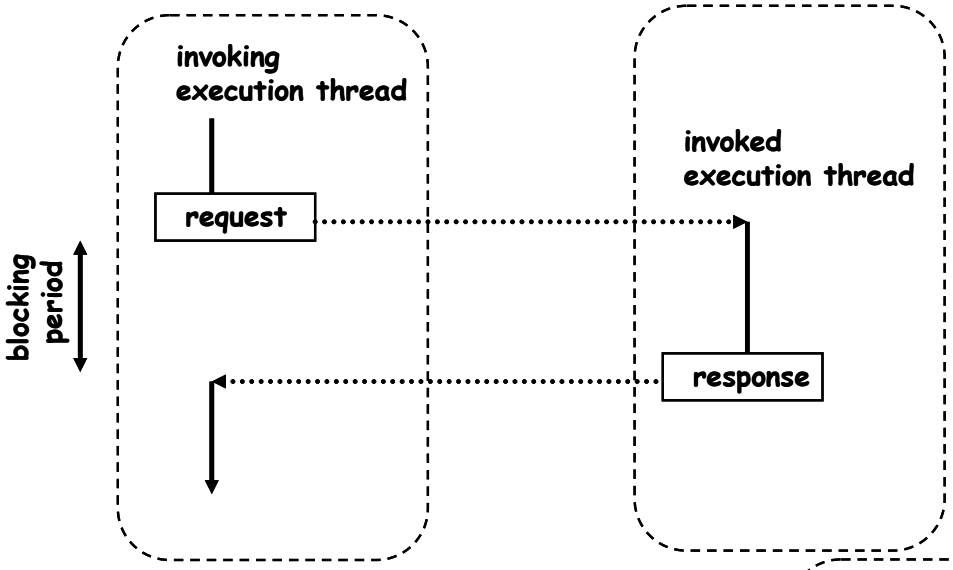
Architettura N-tier





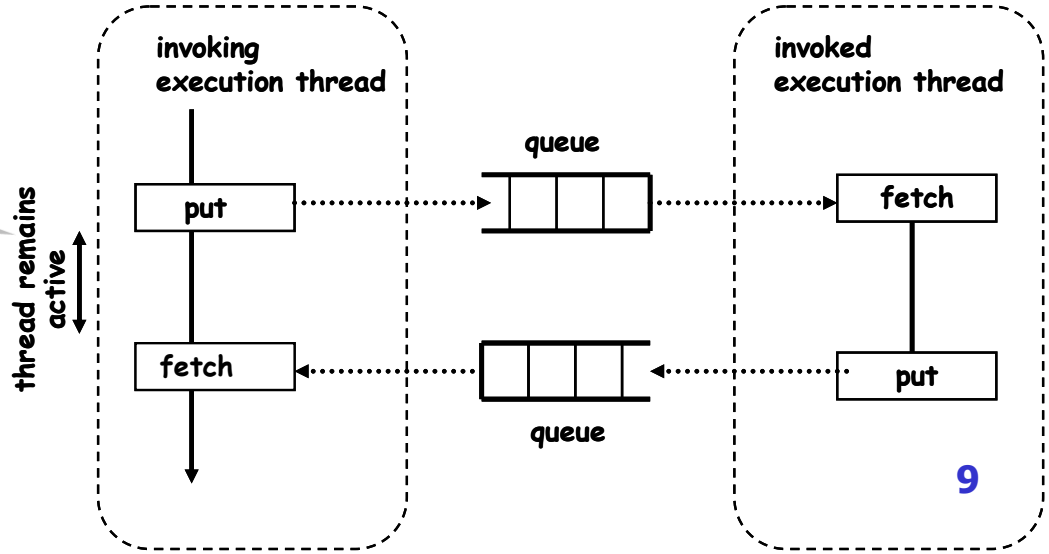
Paradigmi di Comunicazione

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma



Interazione bloccante (impropriamente detta "sincrona")

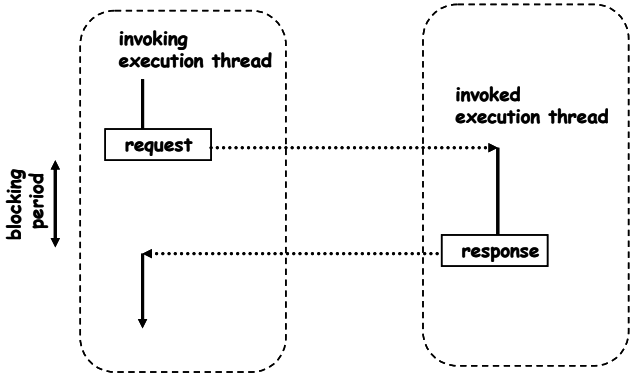
Interazione non-bloccante (impropriamente detta "asincrona")



Blocking or Synchronous Interaction

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA

- Traditionally, information systems use blocking calls (the client sends a request to a service and waits for a response of the service to come back before continuing doing its work)
- Synchronous interaction requires both parties to be “on-line”: the caller makes a request, the receiver gets the request, processes the request, sends a response, the caller receives the response.
- The caller must wait until the response comes back. The receiver does not need to exist at the time of the call (TP-Monitors, CORBA or DCOM create an instance of the service/server /object when called if it does not exist already) but the interaction requires both client and server to be “alive” at the same time



- Because it synchronizes client and server, this mode of operation has several disadvantages:
  - connection overhead
  - higher probability of failures
  - difficult to identify and react to failures
  - it is a one-to-one system; it is not really practical for nested calls and complex interactions (the problems becomes even more acute)

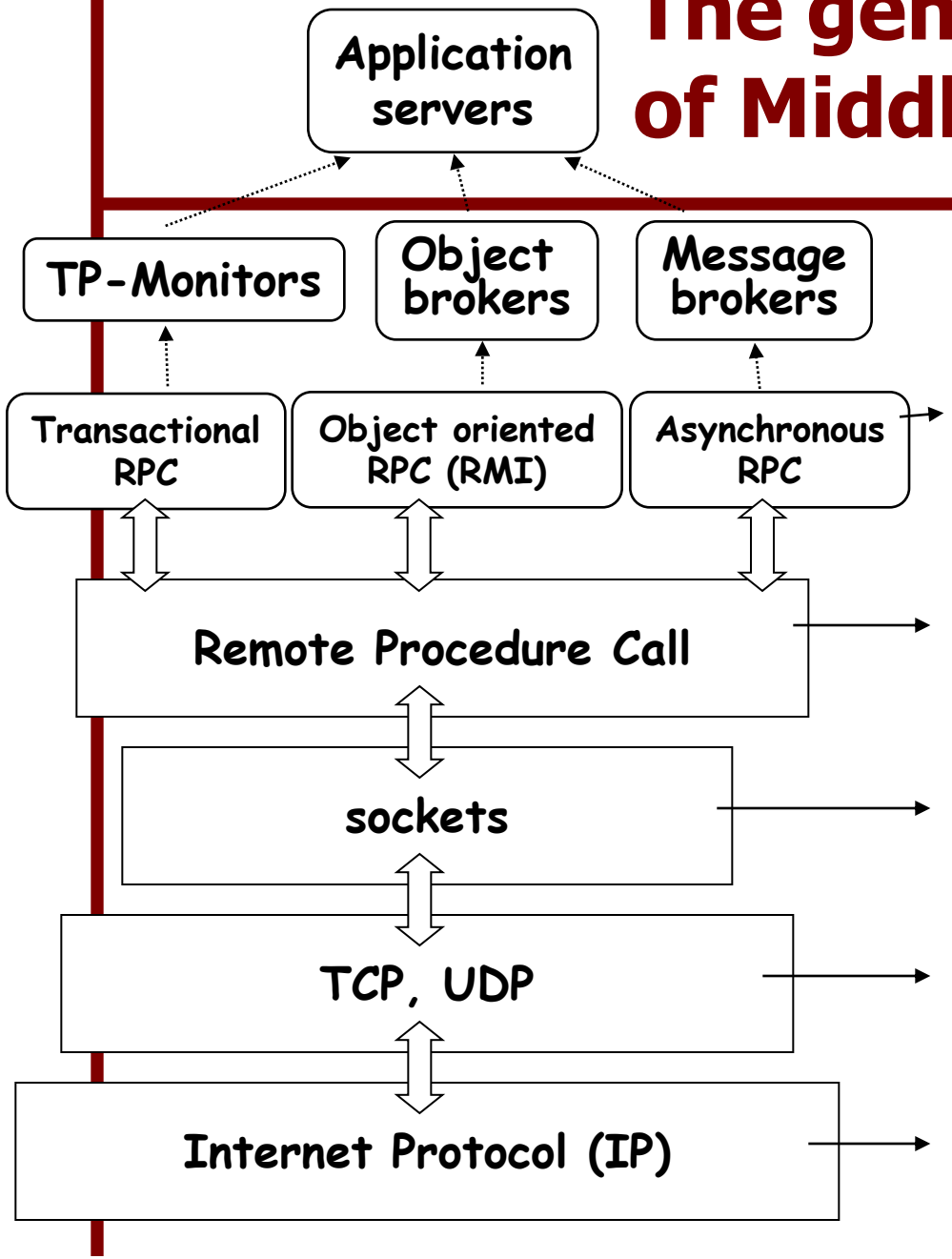
Middleware as a Programming Abstraction

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

- Programming languages and almost any form of software system evolve always towards higher levels of abstraction
  - hiding hardware and platform details
  - more powerful primitives and interfaces
  - leaving difficult task to intermediaries (compilers, optimizers, automatic load balancing, automatic data partitioning and allocation, etc.)
  - reducing the number of programming errors
  - reducing the development and maintenance cost of the applications developed by facilitating their portability
- Middleware is primarily a set of programming abstractions developed to facilitate the development of complex distributed systems
  - to understand a middleware platform one needs to understand its programming model
  - from the programming model the limitations, general performance, and applicability of a given type of middleware can be determined in a first approximation
  - the underlying programming model also determines how the platform will evolve and fare when new technologies evolve

The genealogy of Middleware

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma



Specialized forms of RPC, typically with additional functionality or properties but almost always running on RPC platforms

Remote Procedure Call:
hides communication details behind a procedure call and helps bridge heterogeneous platforms

sockets:
operating system level interface to the underlying communication protocols

TCP, UDP:
User Datagram Protocol (UDP) transports data packets without guarantees
Transmission Control Protocol (TCP) verifies correct delivery of data streams

Internet Protocol (IP):
moves a packet of data from one node to another

Middleware as Infrastructure

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

- As the programming abstractions reach higher and higher levels, the underlying infrastructure implementing the abstractions must grow accordingly
  - Additional functionality is almost always implemented through additional software layers
  - The additional software layers increase the size and complexity of the infrastructure necessary to use the new abstractions

- The infrastructure is also intended to support additional functionality that makes development, maintenance, and monitoring easier and less costly
  - RPC => transactional RPC => logging, recovery, advanced transaction models, language primitives for transactional demarcation, transactional file system, etc.
  - The infrastructure is also there to take care of all the non-functional properties typically ignored by data models, programming models, and programming languages: performance, availability, recovery, instrumentation, maintenance, resource management, etc.

Understanding Middleware

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA

To understand middleware, one needs to understand its dual role as programming abstraction and as infrastructure

PROGRAMMING ABSTRACTION

- Intended to hide low level details of hardware, networks, and distribution
- Trend is towards increasingly more powerful primitives that, without changing the basic concept of RPC, have additional properties or allow more flexibility in the use of the concept
- Evolution and appearance to the programmer is dictated by the trends in programming languages (RPC and C, CORBA and C++, RMI and Java, Web services and SOAP-XML)

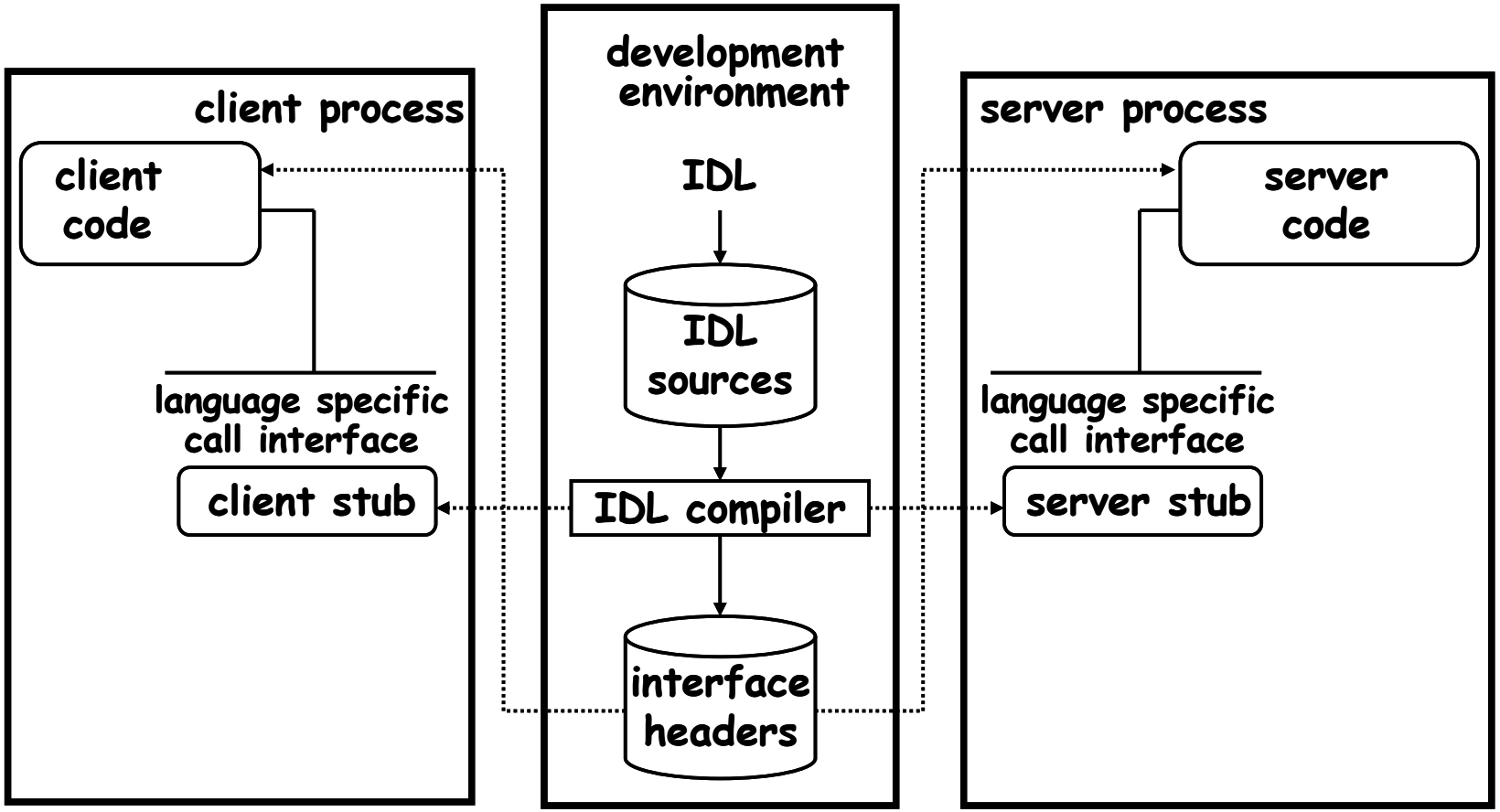
INFRASTRUCTURE

- Intended to provide a comprehensive platform for developing and running complex distributed systems
- Trend is towards service oriented architectures at a global scale and standardization of interfaces
- Another important trend is towards single vendor software stacks to minimize complexity and streamline interaction
- Evolution is towards integration of platforms and flexibility in the configuration (plus autonomic behavior)

Basic Middleware: RPC (1)

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

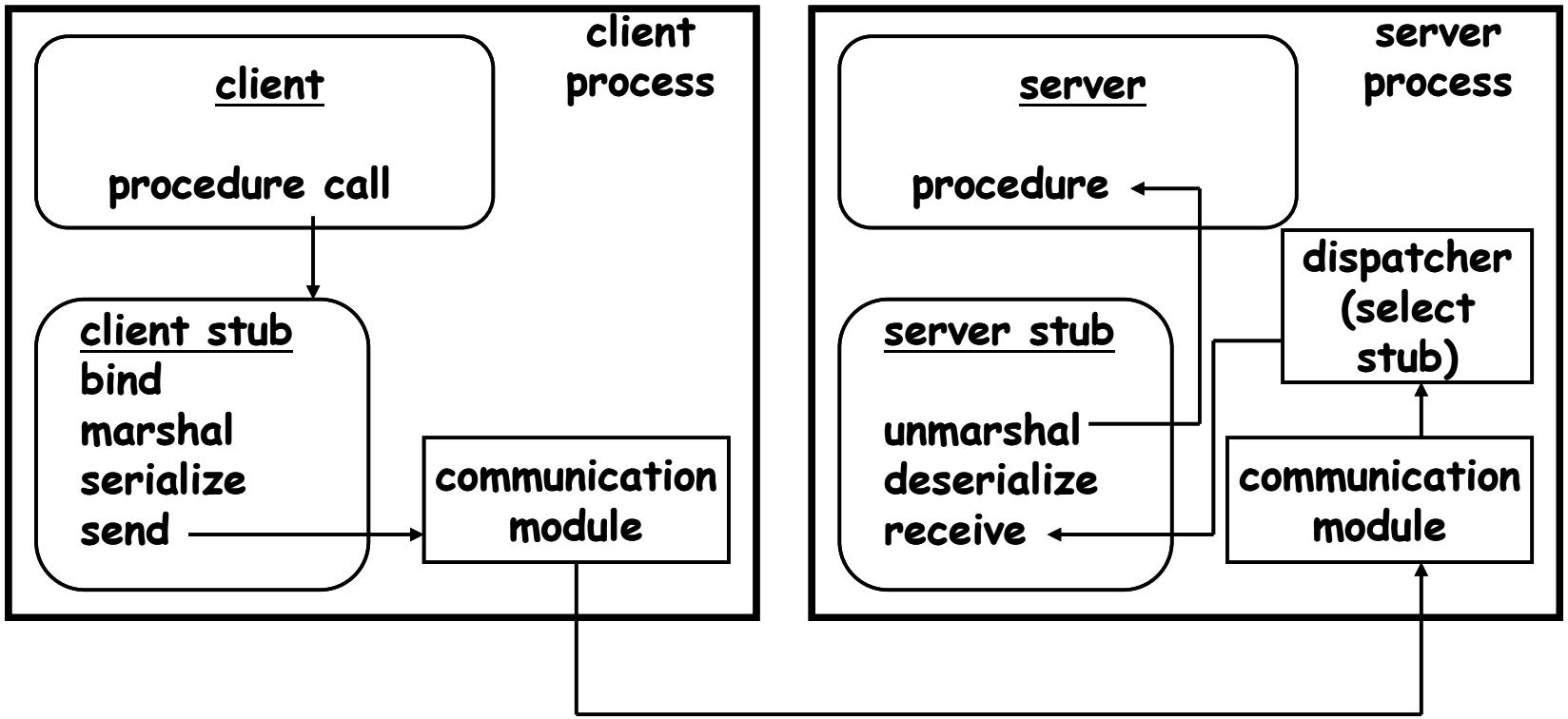
SAPIENZA
UNIVERSITÀ DI ROMA



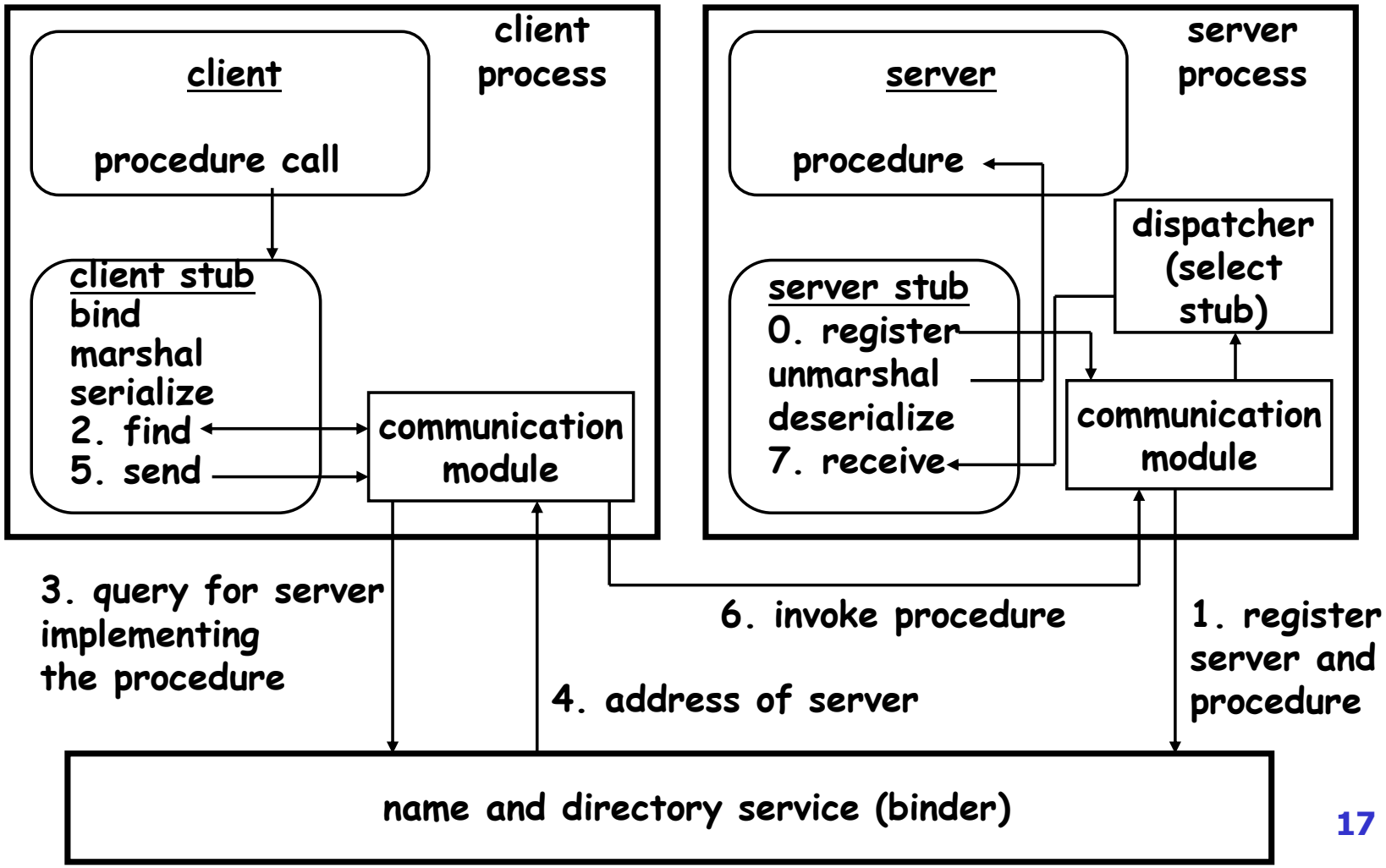
Basic Middleware: RPC (2)

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA



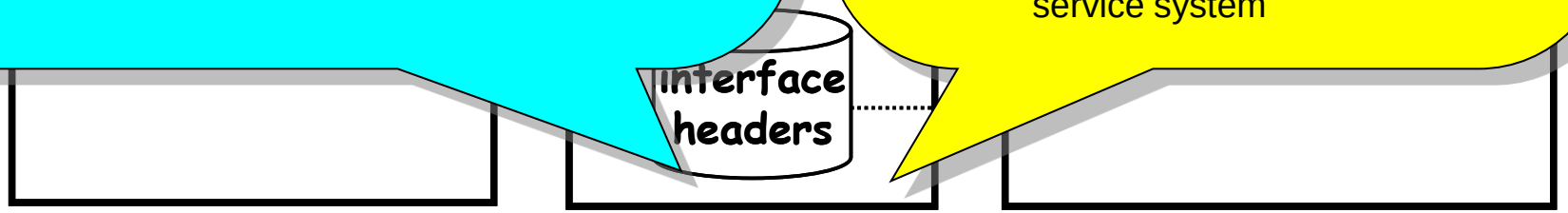
Basic Middleware (3)



Comments

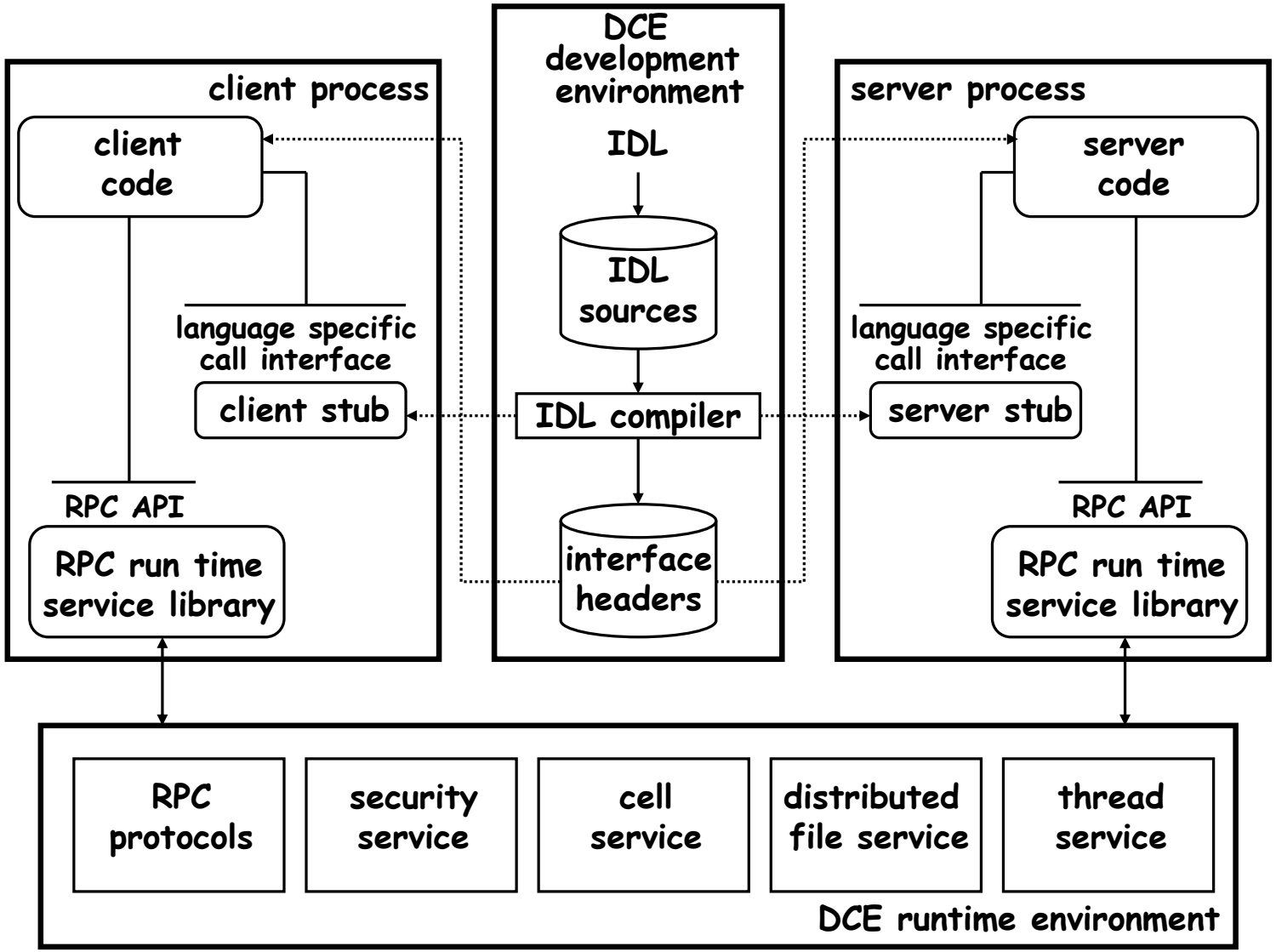
RPC is a point to point protocol in the sense that it supports the interaction between two entities (the client and the server)
When there are more entities interacting with each other (a client with two servers, a client with a server and the server with a database), RPC treats the calls as independent of each other. However, the calls are not independent
Recovering from partial system failures is very complex. Avoiding these problems using plain RPC systems is very cumbersome

- One cannot expect the programmer to implement a complete infrastructure for every distributed application. Instead, one can use an RPC system (our first example of low level middleware)
- What does an RPC system do?
  - Hides distribution behind procedure calls
  - Provides an interface definition language (IDL) to describe the services
  - Generates all the additional code necessary to make a procedure call remote and to deal with all the communication aspects
  - Provides a binder in case it has a distributed name and directory service system



A Real Example ('90): DCE

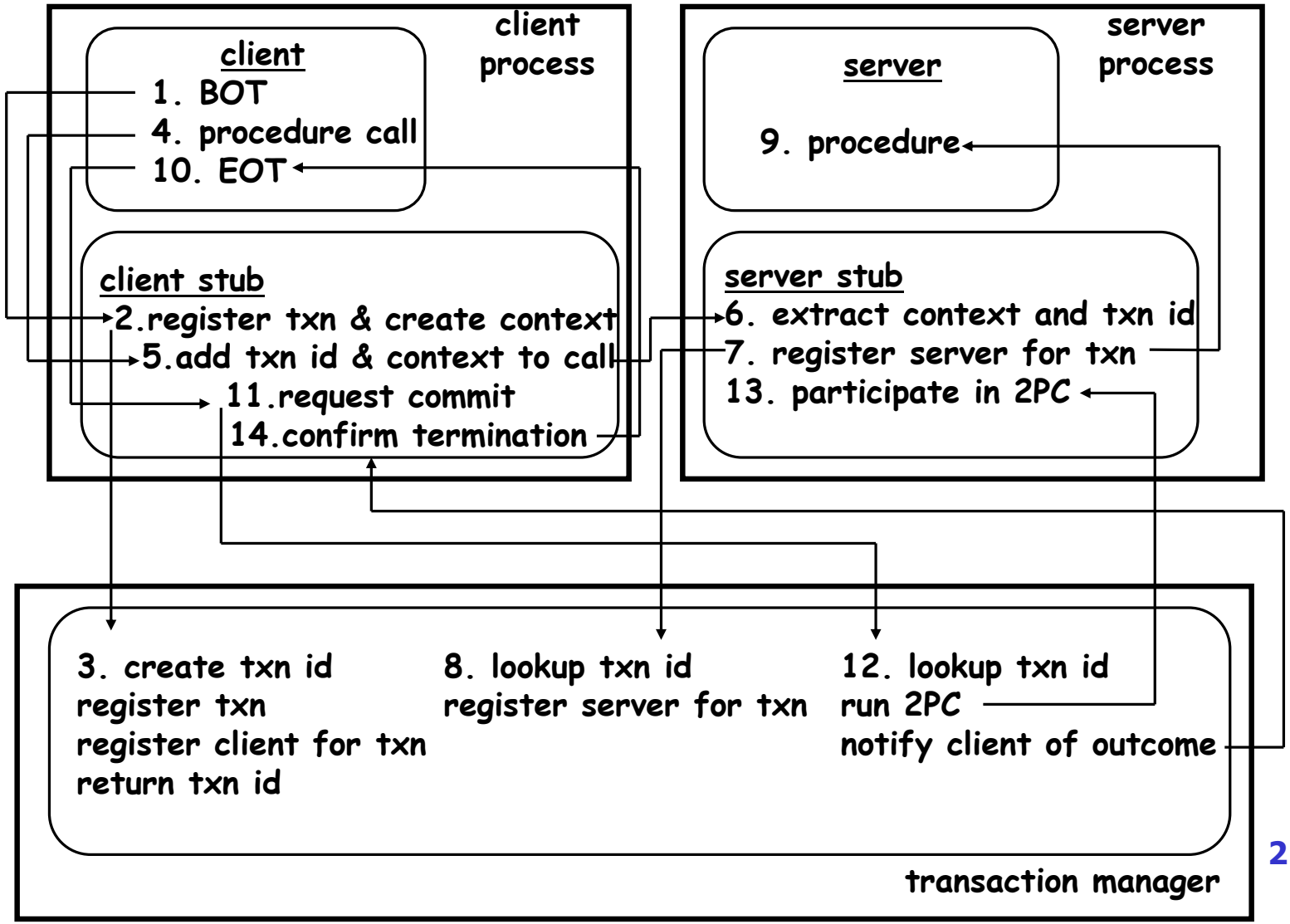
DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA



Transactional RPC

- The solution to this limitation is to make RPC calls transactional, that is, instead of providing plain RPC, the system should provide TRPC
- What is TRPC?
  - same concept as RPC plus …
  - additional language constructs and run time support (additional services) to bundle several RPC calls into an atomic unit
  - usually, it also includes an interface to databases for making end-to-end transactions using the XA standard (implementing 2 Phase Commit)
  - and anything else the vendor may find useful (transactional callbacks, high level locking, etc.)
- Simplifying things quite a bit, one can say that, historically, TP-Monitors are RPC based systems with transactional support. We have already seen an example of this: Encina

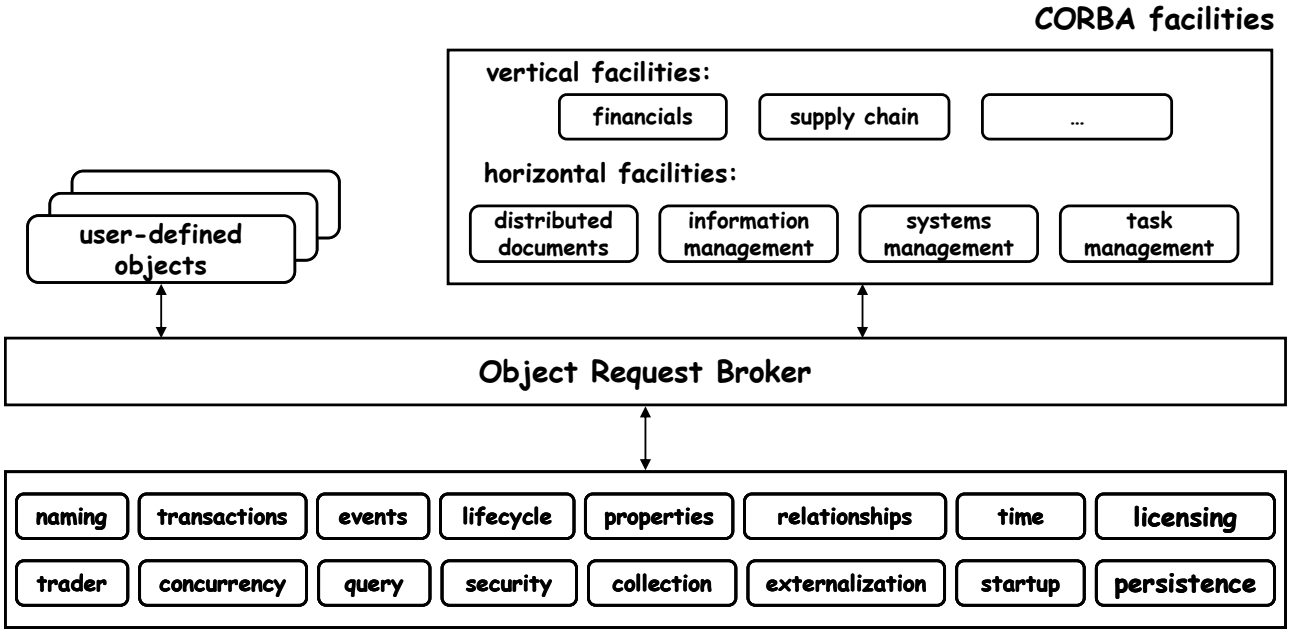
Transactional RPC



Object Broker: an example - CORBA

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA

- The Common Object Request Broker Architecture (CORBA) is part of the Object Management Architecture (OMA) standard, a reference architecture for component based systems
- The key parts of CORBA are:
  - Object Request Broker (ORB): in charge of the interaction between components
  - CORBA services: standard definitions of system services
  - A standardized IDL language for the publication of interfaces
  - Protocols for allowing ORBs to talk to each other
- CORBA was an attempt to modernize RPC by making it object oriented and providing a standard

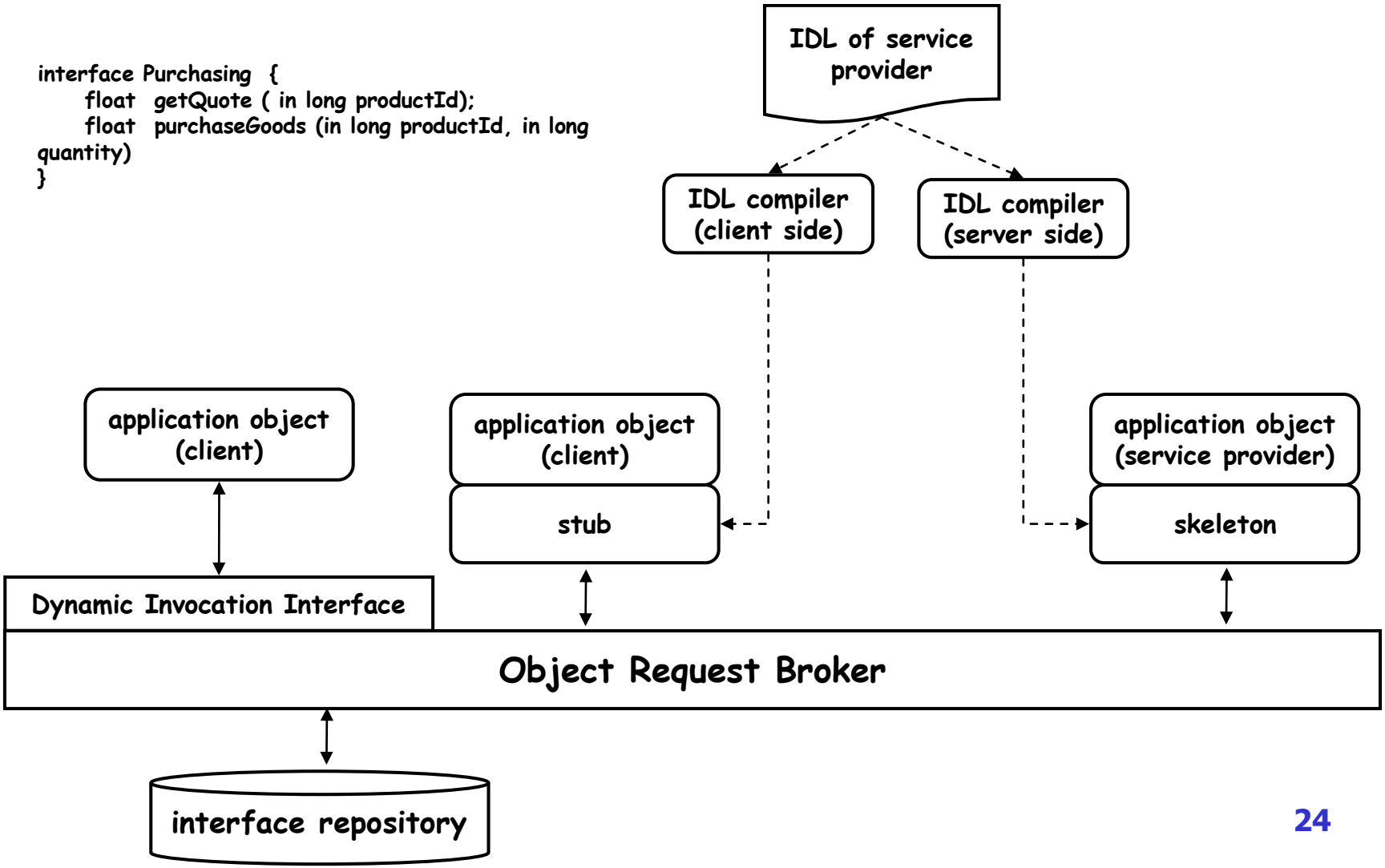


Something New ?

- CORBA follows the same model as RPC :
  - they are trying to solve the same problem
  - CORBA is often implemented on top of RPC
- Unlike RPC, however, CORBA proposes a complete architecture and identifies parts of the system to much more detail than RPC ever did (RPC is an inter-process communication mechanism, CORBA is a reference architecture that includes an inter-process communication mechanism)
- CORBA standardized component based architectures but many of the concepts behind were already in place long ago
- Development is similar to RPC:
  - define the services provided by the server using IDL (define the server object)
  - compile the definition using an IDL compiler. This produces the client stub (proxy, server proxy, proxy object) and the server stub (skeleton). The method signatures (services that can be invoked) are stored in an interface repository
  - Program the client and link it with its stub
  - Program the server and link it with its stub
- Unlike in RPC, the stubs make client and server independent of the operating system and programming language

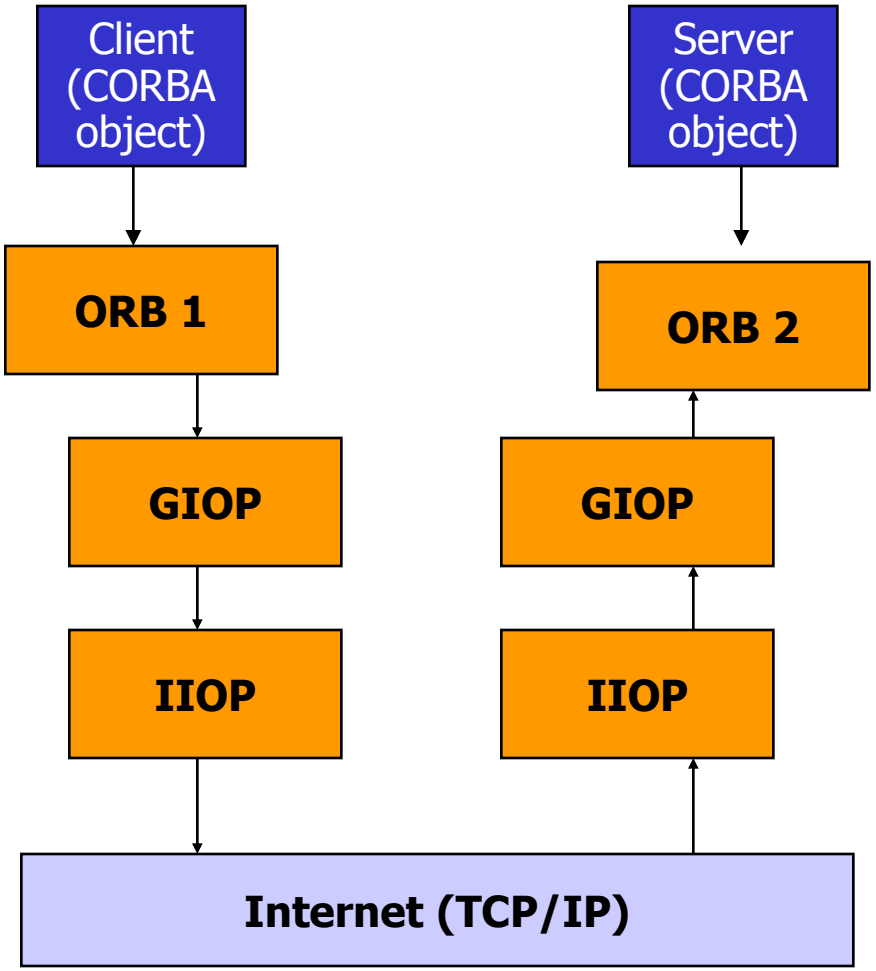
CORBA Development

```
interface Purchasing  {
     float  getQuote ( in long productId);
     float  purchaseGoods (in long productId, in long
quantity)
}
```



Objects Everywhere: IIOP and GIOP

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma



- In order for ORBs to be a truly universal component architecture, there has to be a way to allow ORBs to communicate with each other (one cannot have all components in the world under a single ORB)
- For this purpose, CORBA provides a General Inter-ORB Protocol (GIOP) that specifies how to forward calls from one ORB to another and get the requests back
- The Internet Inter-ORB Protocol specifies how GIOP messages are translated into TCP/IP
- There are additional protocols to allow ORBs to communicate with other systems
- The idea was sound but came too late and was soon superseded by Web services

The Best of Two Worlds: Object Monitors

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

- Middleware technology should be interpreted as different stages of evolution of an “ideal” system. Current systems do not compete with each other per se, they complement each other. The competition arises as the underlying infrastructures converge towards a single platform:

- OBJECT REQUEST BROKERS (ORBs): Reuse and distribution of components via a standard, object oriented interface and a number of services that add semantics to the interaction between components
- TRANSACTION PROCESSING MONITORS: An environment to develop components capable of interacting transactionally and the tools necessary to maintain transactional consistency

And Object Transaction Monitors ?

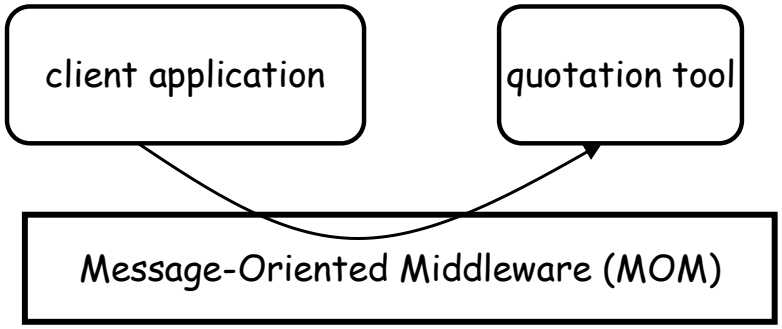
Object Monitor = ORB + TP-Monitor

Message Oriented Middleware

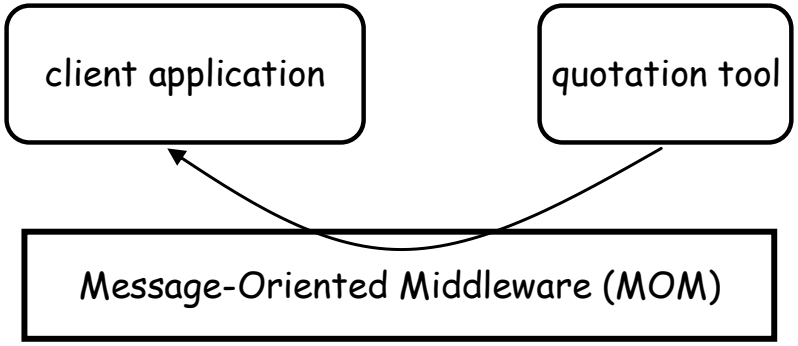
Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma

```
Message : quoteRequest {
 QuoteReferenceNumber: 325
 Customer: Acme,INC
 Item:#115 (Ball-point pen, blue)
 Quantity: 1200
 RequestedDeliveryDate: Mar 16,2003
 DeliveryAddress: Palo Alto, CA
}
```

```
Message: quote {
 QuoteReferenceNumber: 325
 ExpectedDeliveryDate: Mar 12, 2003
 Price:1200$
}
```



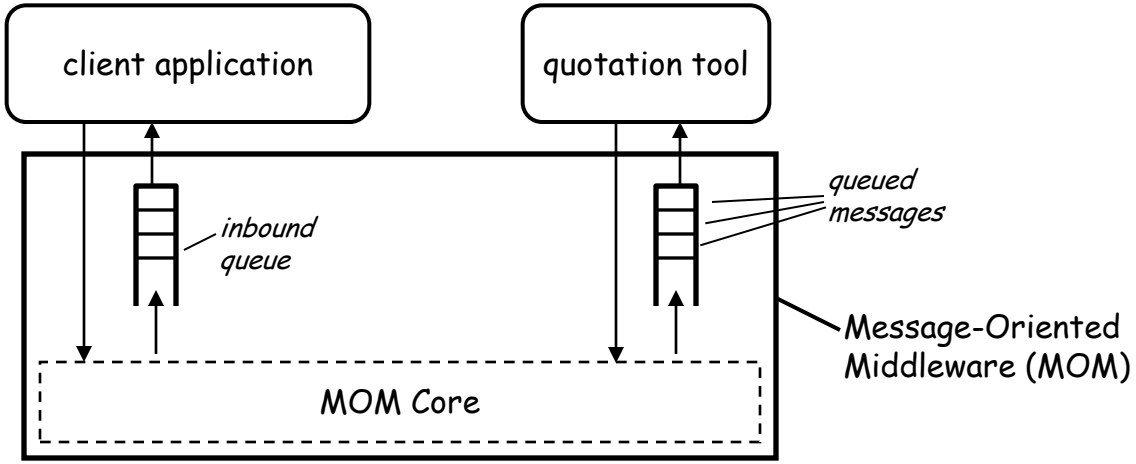
(a)



(b)

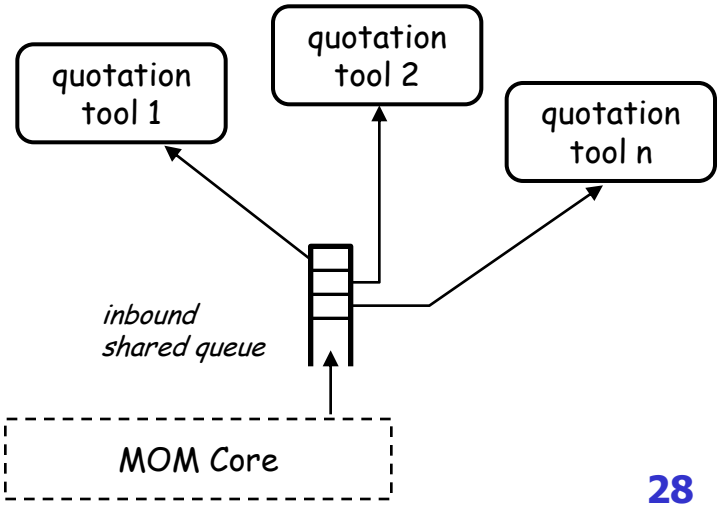
Message Oriented Middleware

Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma



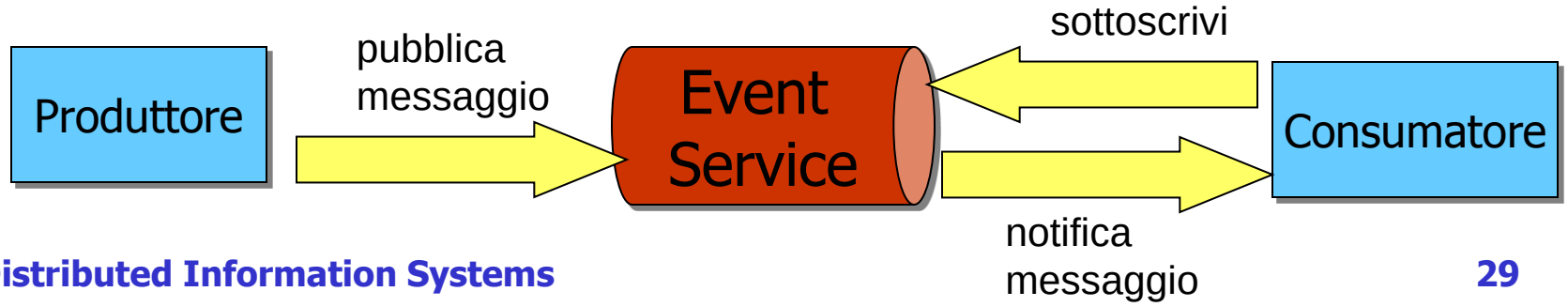
The queuing model

(Primitive) load balancing with shared queues



Publish/Subscribe

- Realizza un'interazione molti-a-molti e anonima
- I partecipanti si suddividono in due gruppi
  - publisher: produttori, inviano messaggi
  - subscriber: consumatori, esprimono interesse per determinate categorie di messaggi
- La comunicazione avviene attraverso un'entità centrale (event service) che riceve i messaggi dai publisher e li incrocia con gli interessi dei subscriber



Publish/Subscribe

- ***Topic-based***: le informazioni vengono suddivise in argomenti (*topic* o *subject*)
  - sottoscrizioni e pubblicazioni per argomento
  - canale logico ideale che connette un publisher a tutti i subscriber
- ***Content-based***: possibile utilizzare dei filtri per una selezione più accurata delle informazioni da ricevere (contenuto dei messaggi)
- La notifica può avvenire secondo due modalità
  - **push**: i sottoscrittori vengono invocati in callback, utilizzando un riferimento comunicato al momento della sottoscrizione
  - **pull**: i sottoscrittori eseguono un polling sull'event service quando hanno bisogno di messaggi

Esempio

```
Message QuoteREquest {
        QuoteRefereneceNumber: Integer
        Customer: String
        Item: String
        Quantity: String
        RequestedDeliveryDate: Timestamp
        DeliveryAddress: String
}
```

Con P&S topic-based un sottoscrittore può registrare il suo interesse a tutti i messaggi di tipo `QuoteRequest` (selezione della "classe")

Con P&S content-based un sottoscrittore può registrare il suo interesse a tutti i messaggi di tipo `QuoteRequest` tali che `RequestedDeliveryDate before June 30th, 2004 AND Quanity > 1000`

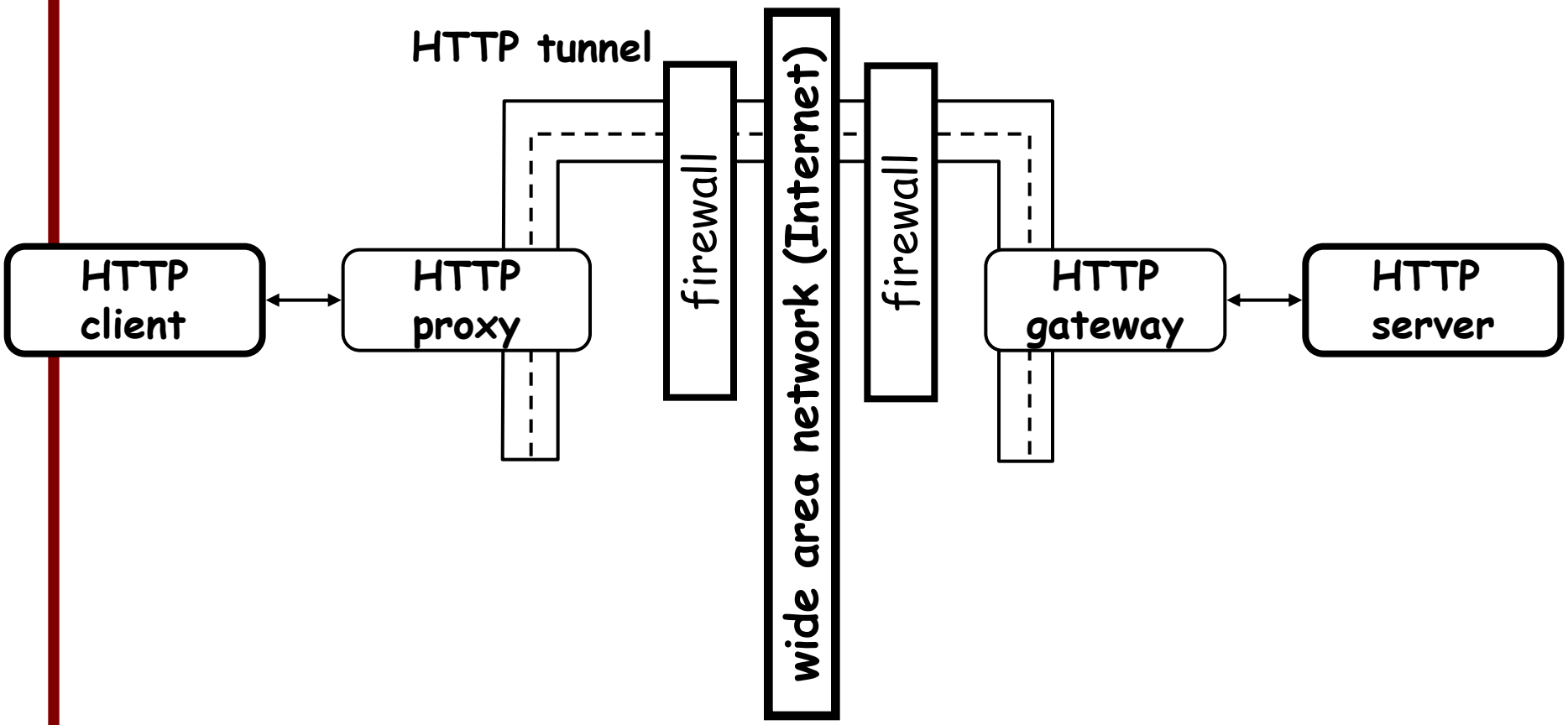
Si possono scrivere espressioni che permettono di filtrare tutti i messaggi di un tipo rispetto ai valori degli attributi (selezione dell' "istanza"). Diventa importante il potere espressivo del linguaggio di sottoscrizione (una sorta di query language)

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

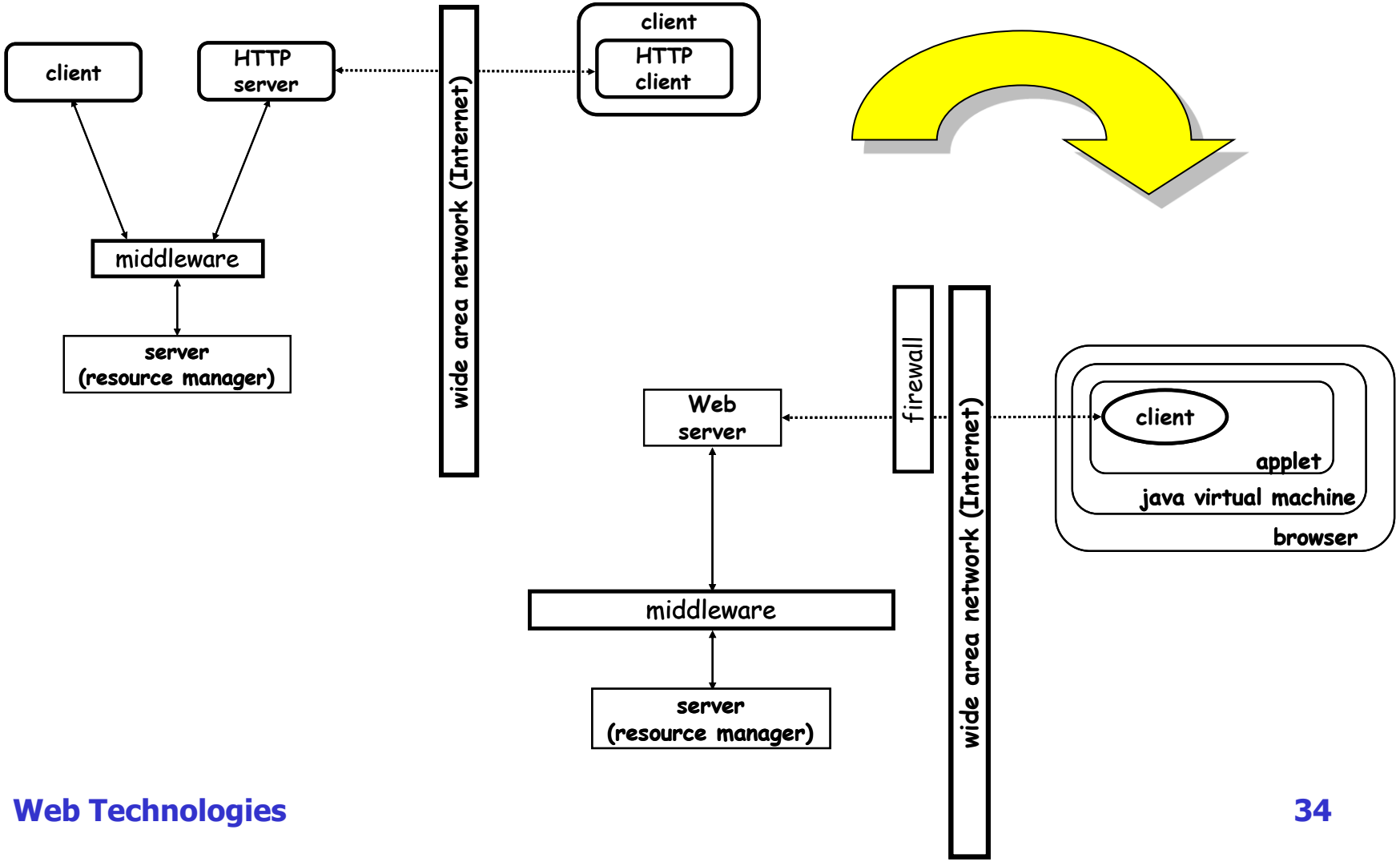
SAPIENZA
UNIVERSITÀ DI ROMA

Web Technologies

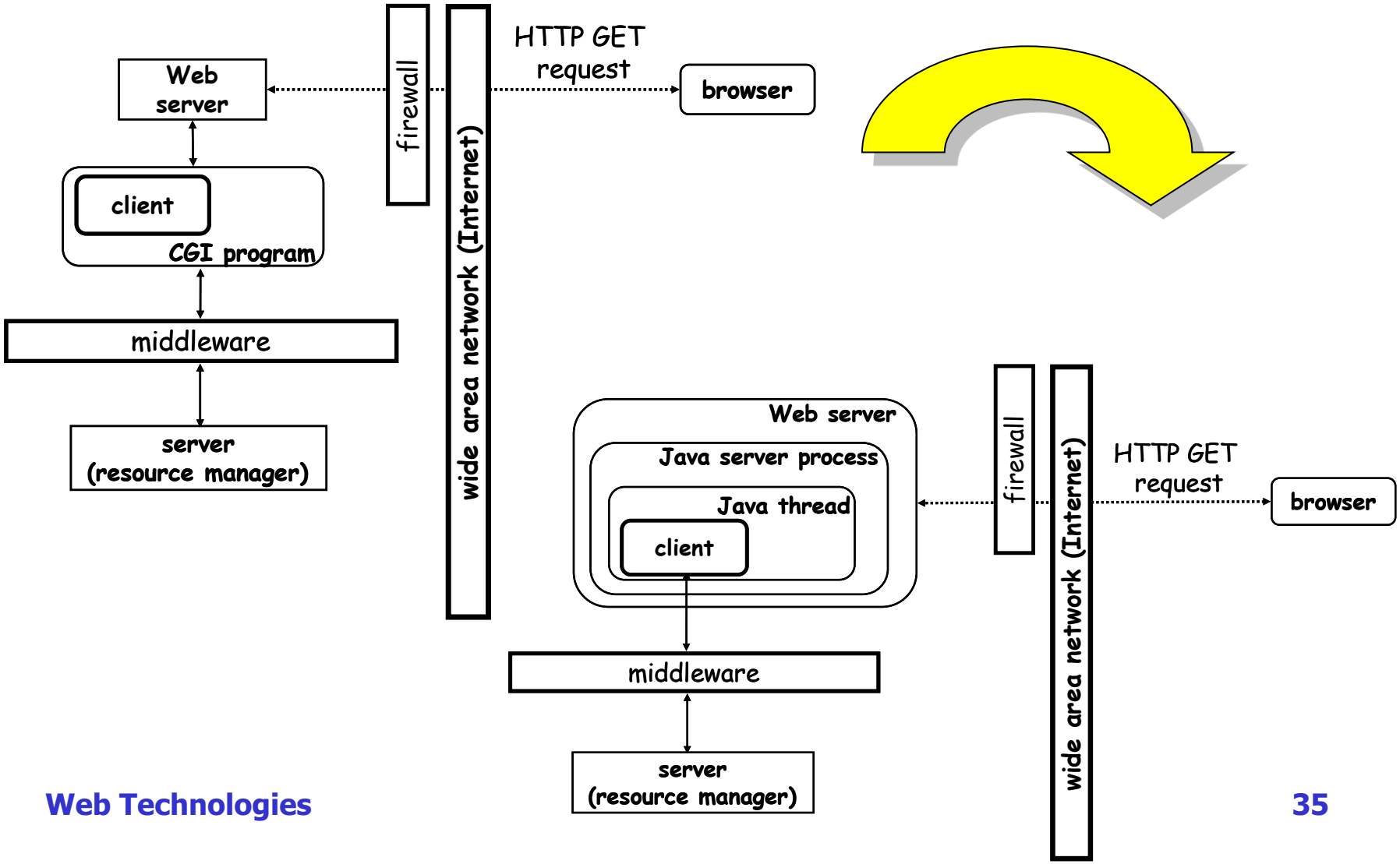
Proxy and Firewall



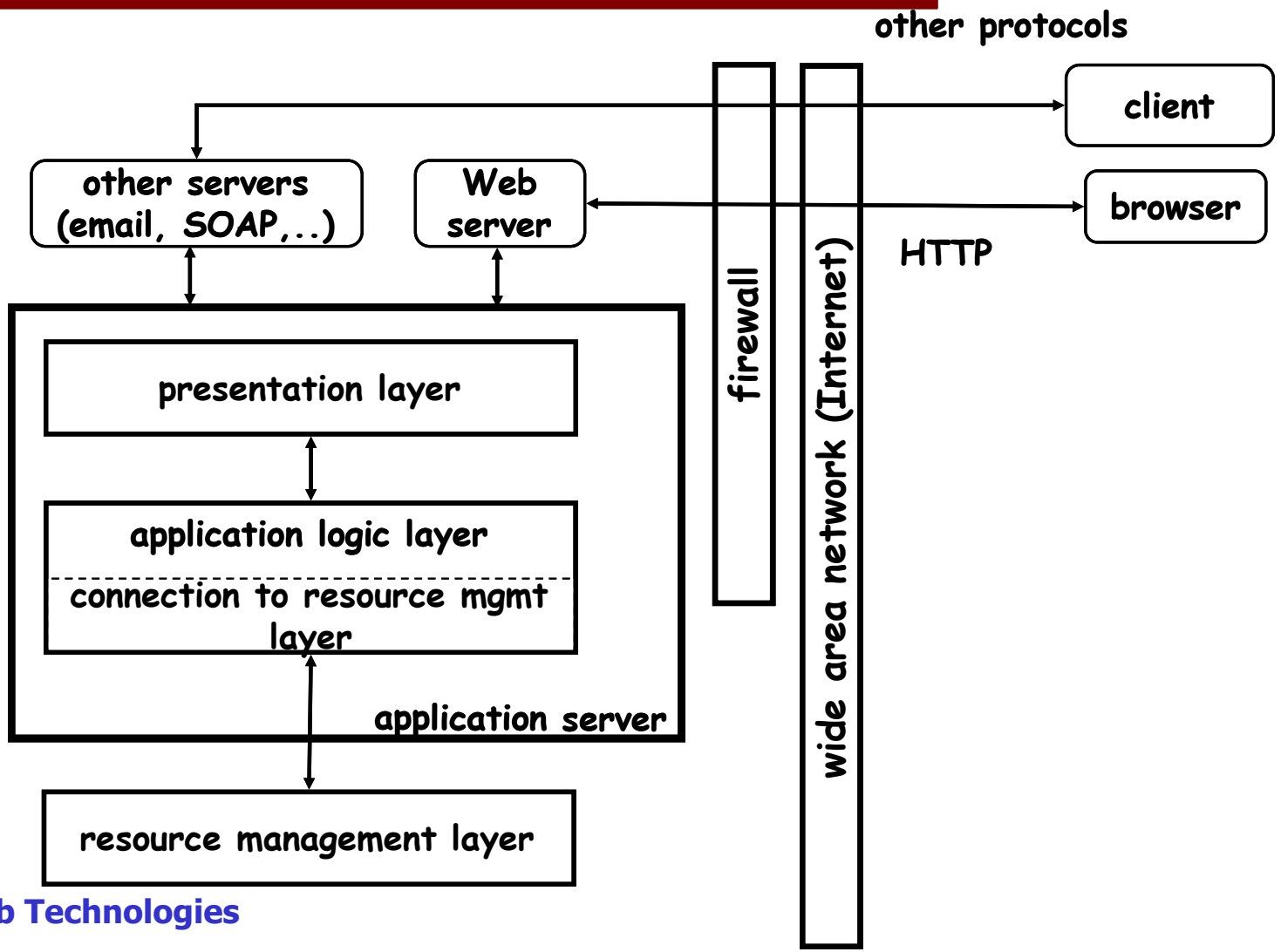
3-tier on the Web: Support for Remote Clients



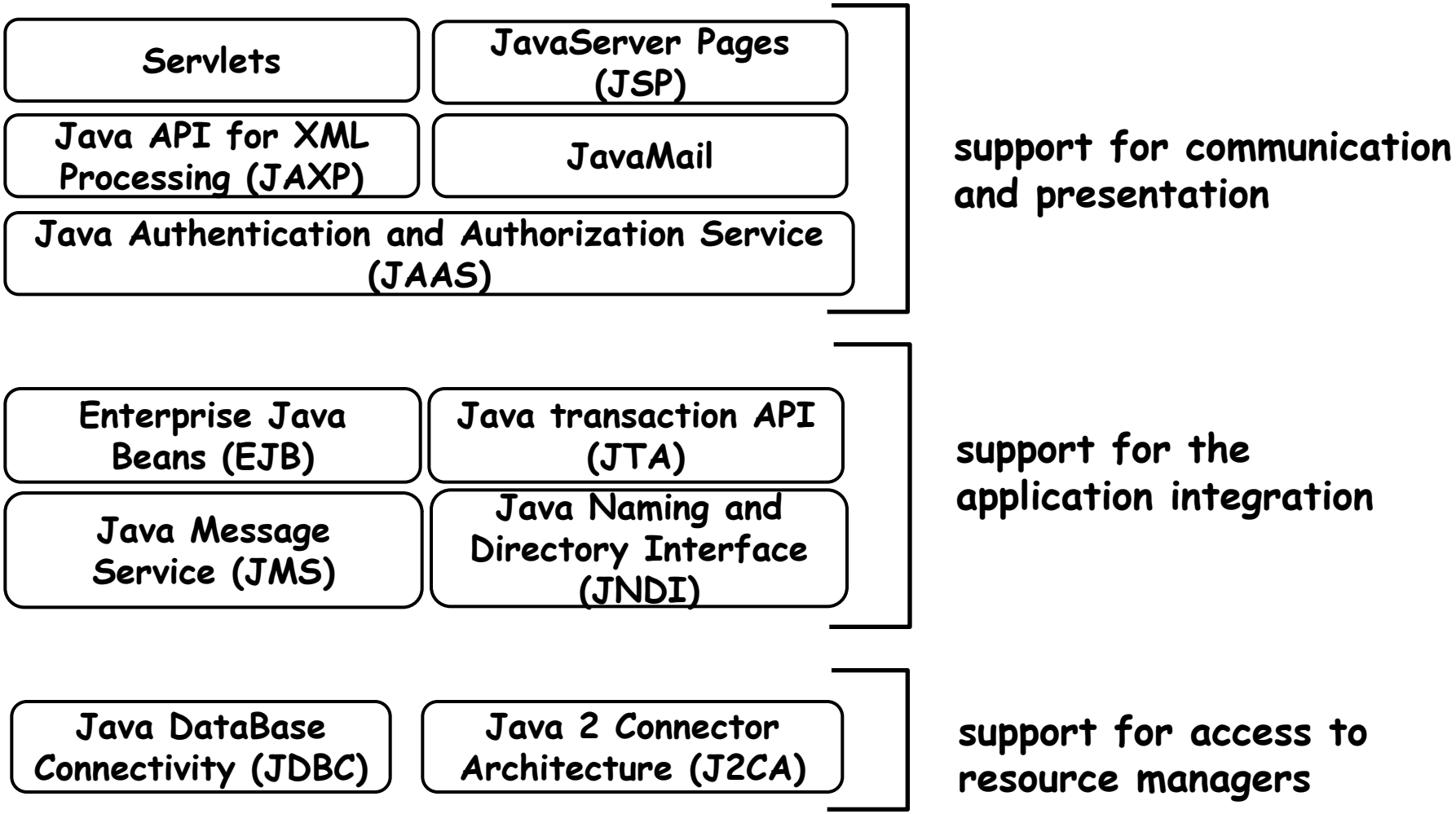
3-tier on the Web: Server Pages



Putting All Together: Application Servers

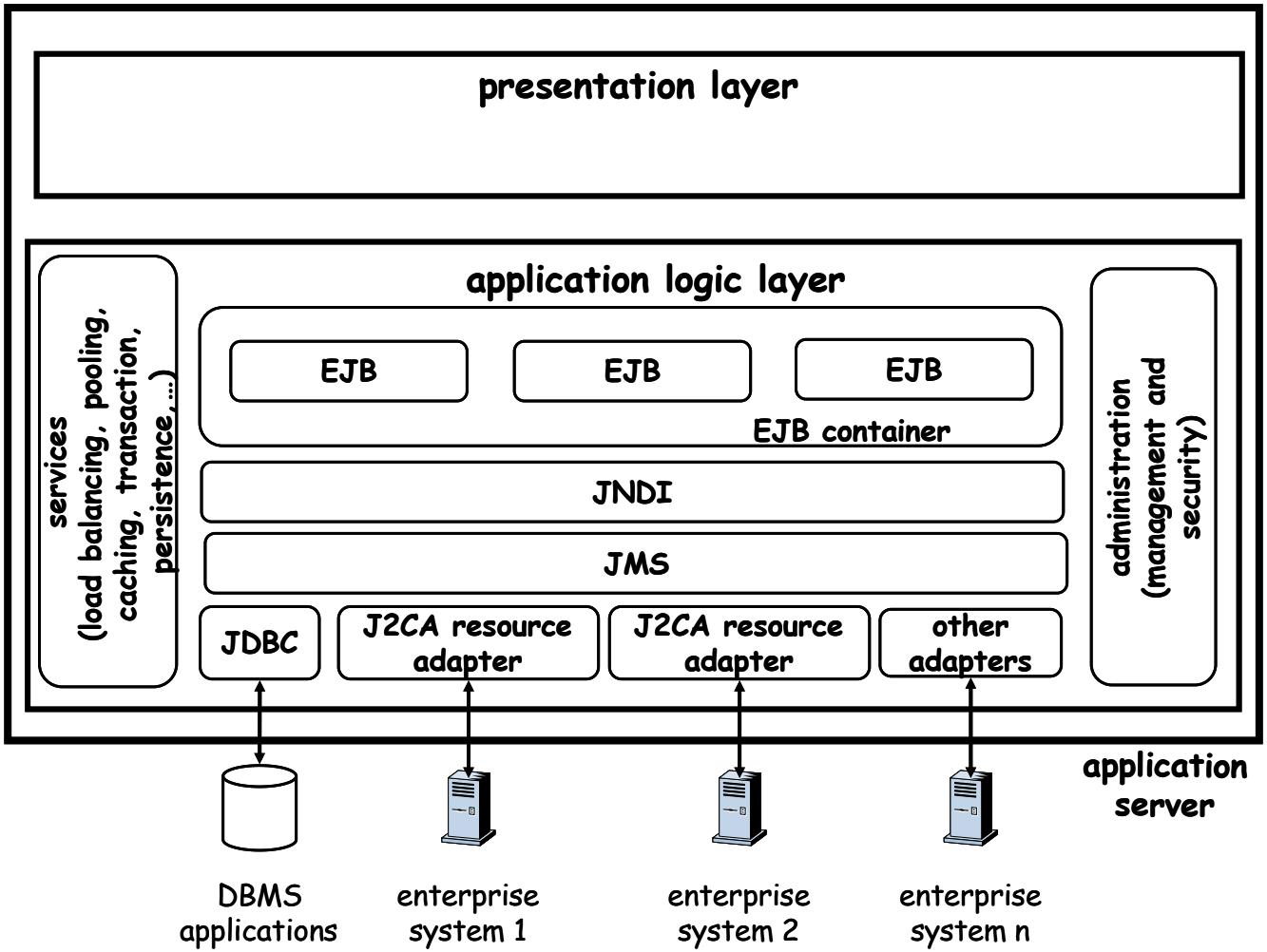


An Example: J2EE



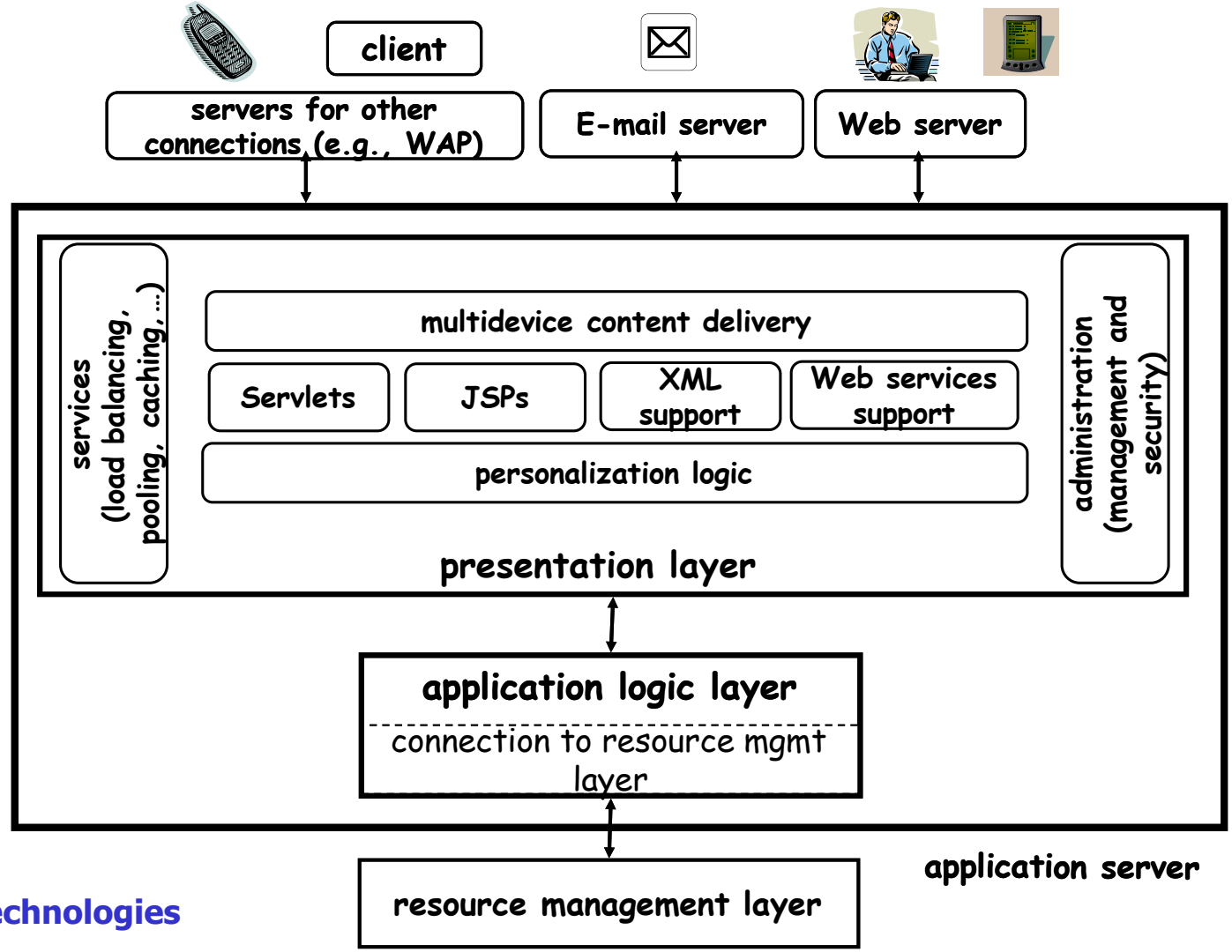
J2EE Support for the Application Logic

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA



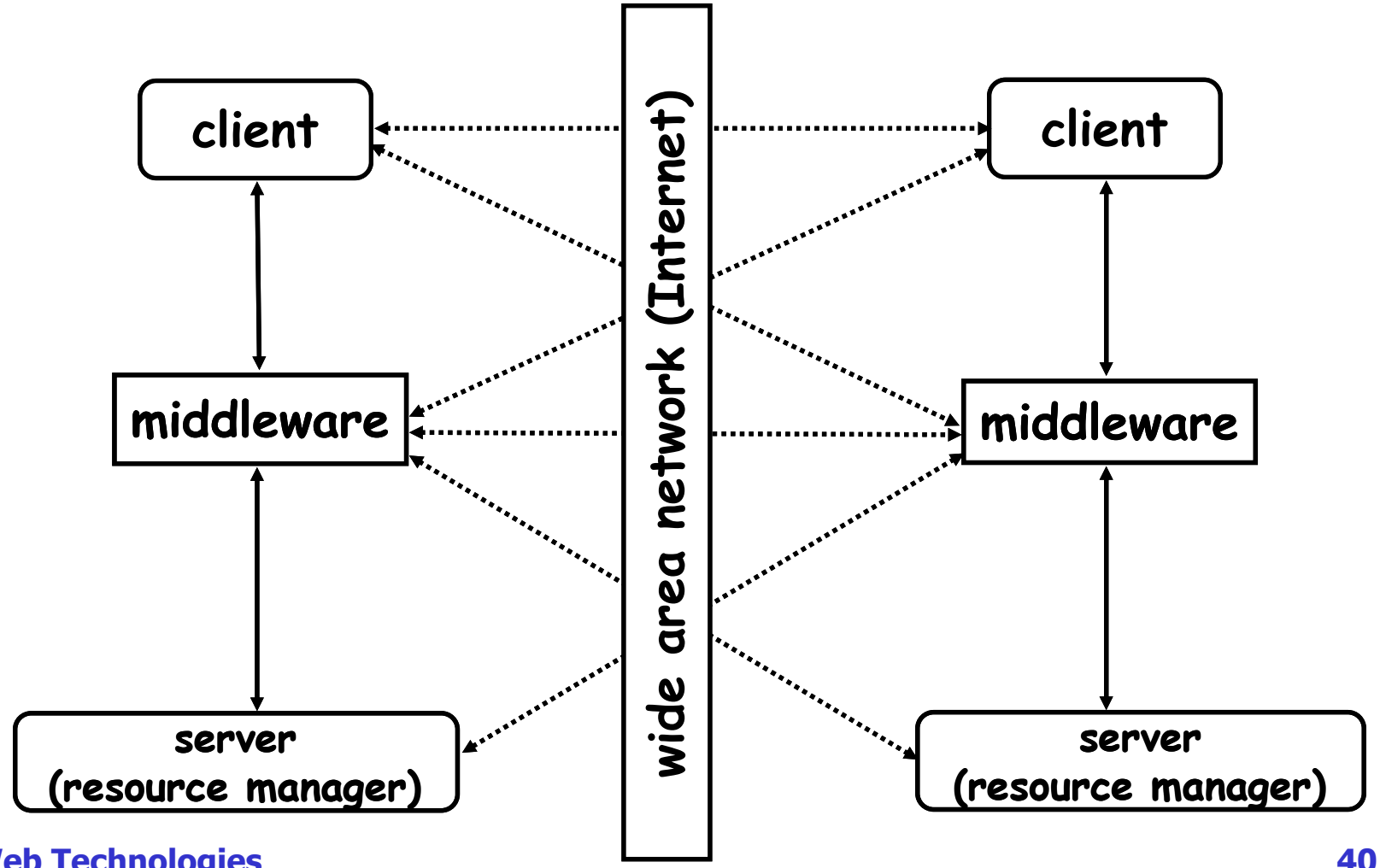
J2EE Support for the Presentation Layer

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI
SAPIENZA
UNIVERSITÀ DI ROMA



Integration Across the Internet

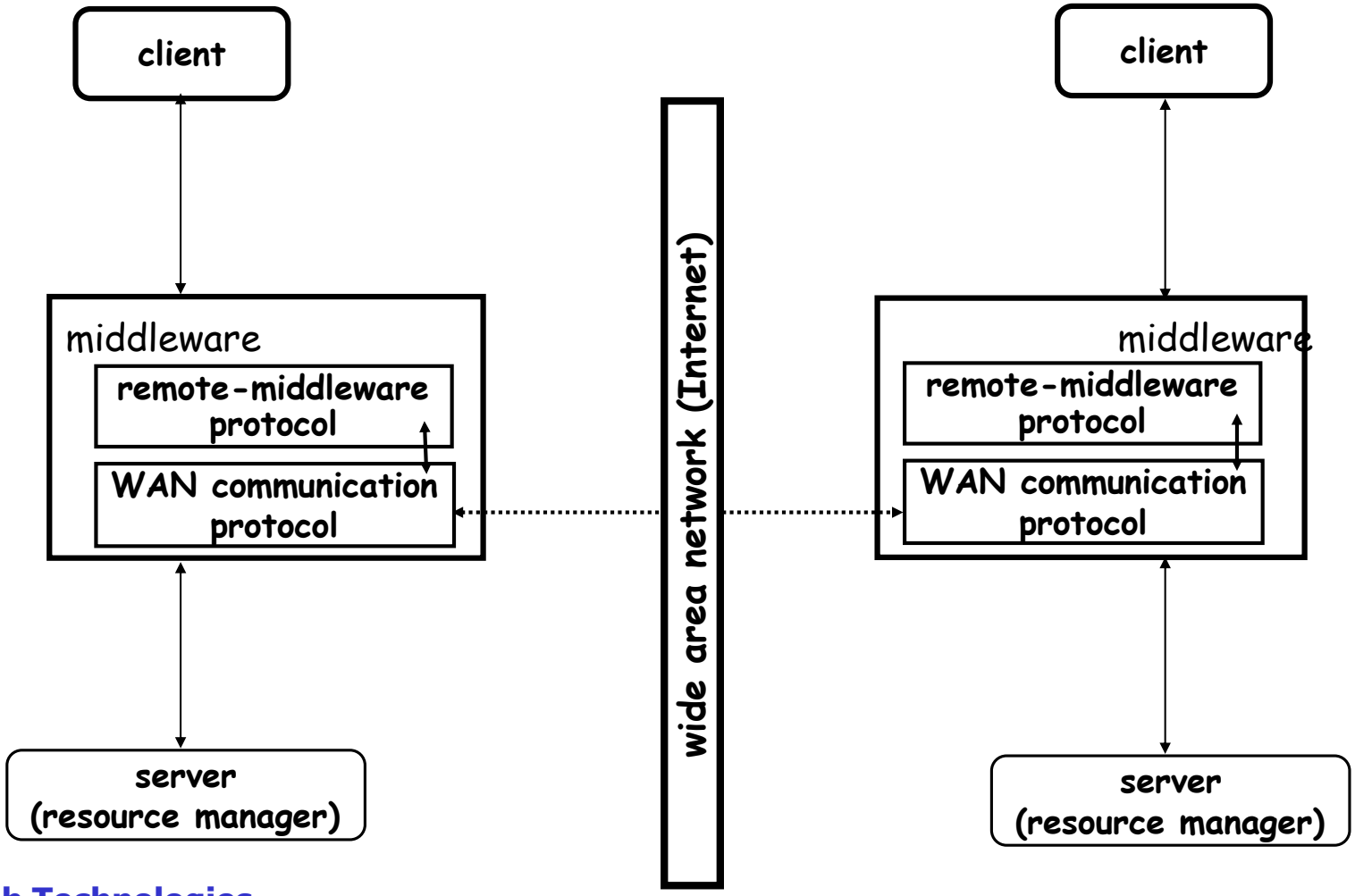
Dipartimento di Informatica
e Sistemistica Antonio Ruberti
Sapienza
Università di Roma



Integration M2M (middleware-to-middleware) ...

DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA



… May Requires HTTP Tunnelling/Support