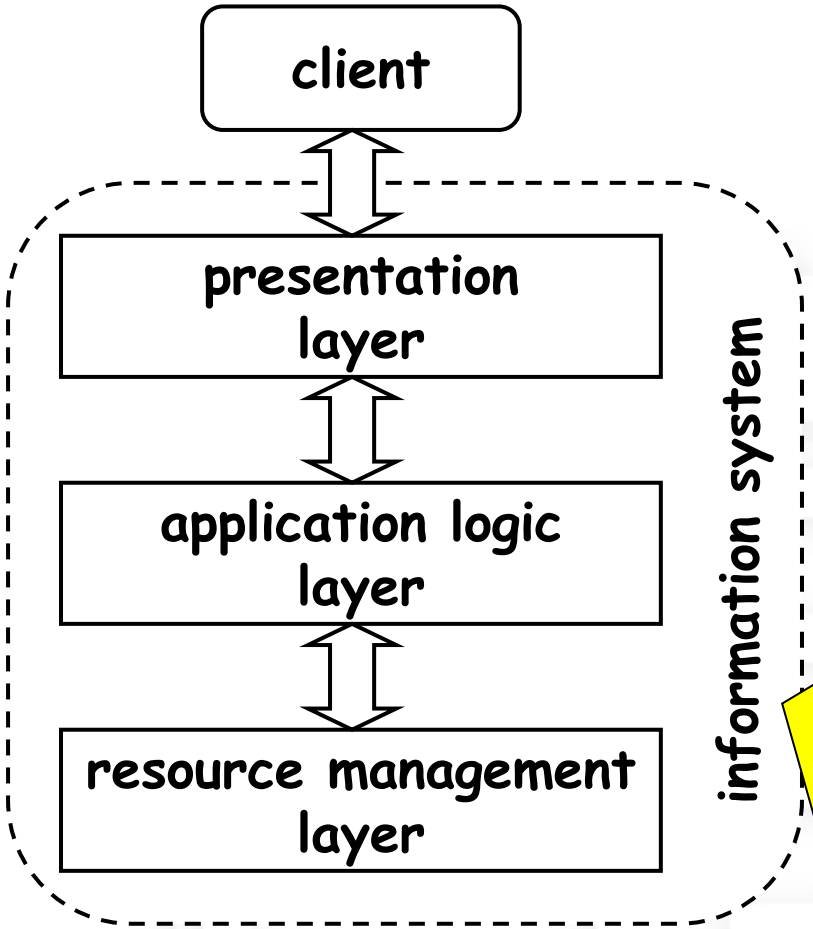
Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

Software Engineering

Distributed programming – Basic technologies
Remote Procedure Call (RPC) & Message Oriented Middleware (MOM)
RPC & MOM in Java: Java RMI & JMS

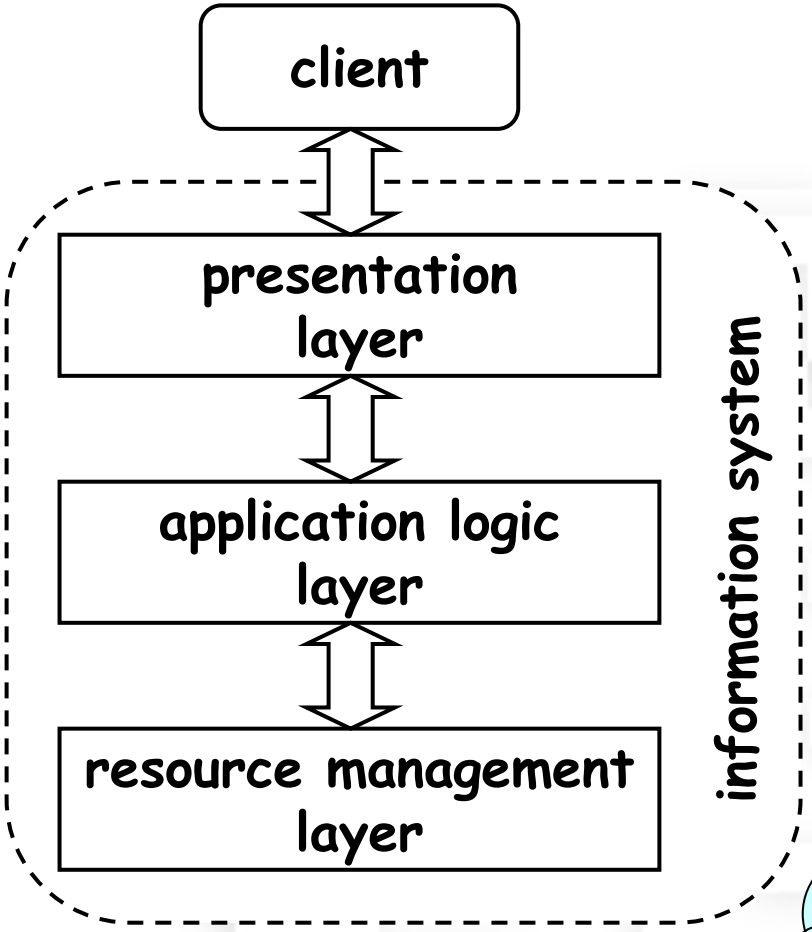
LAYERS VS. TIERS

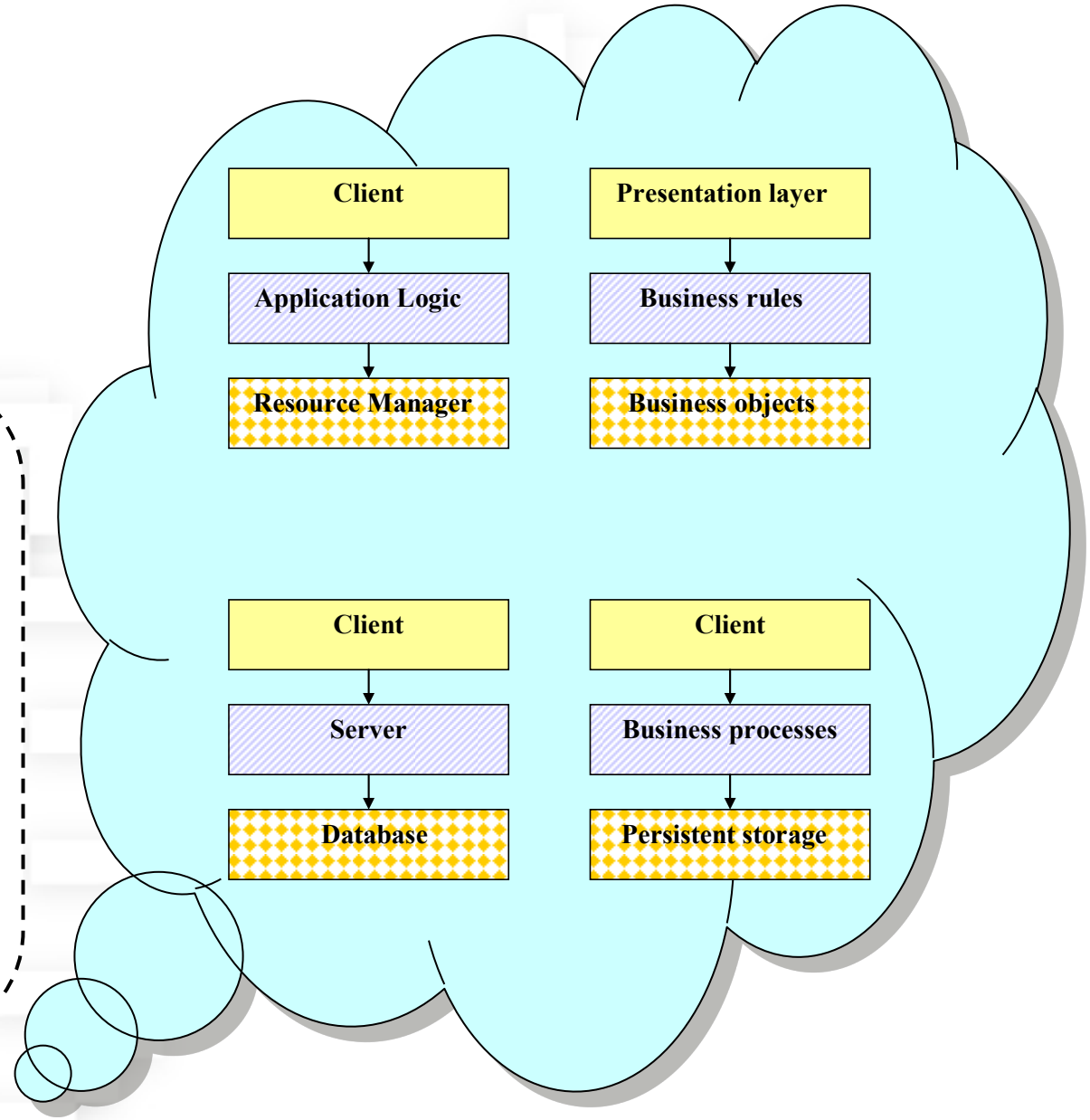


Client is any user or program that wants to perform an operation over the system. Clients interact with the system through a presentation layer

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.

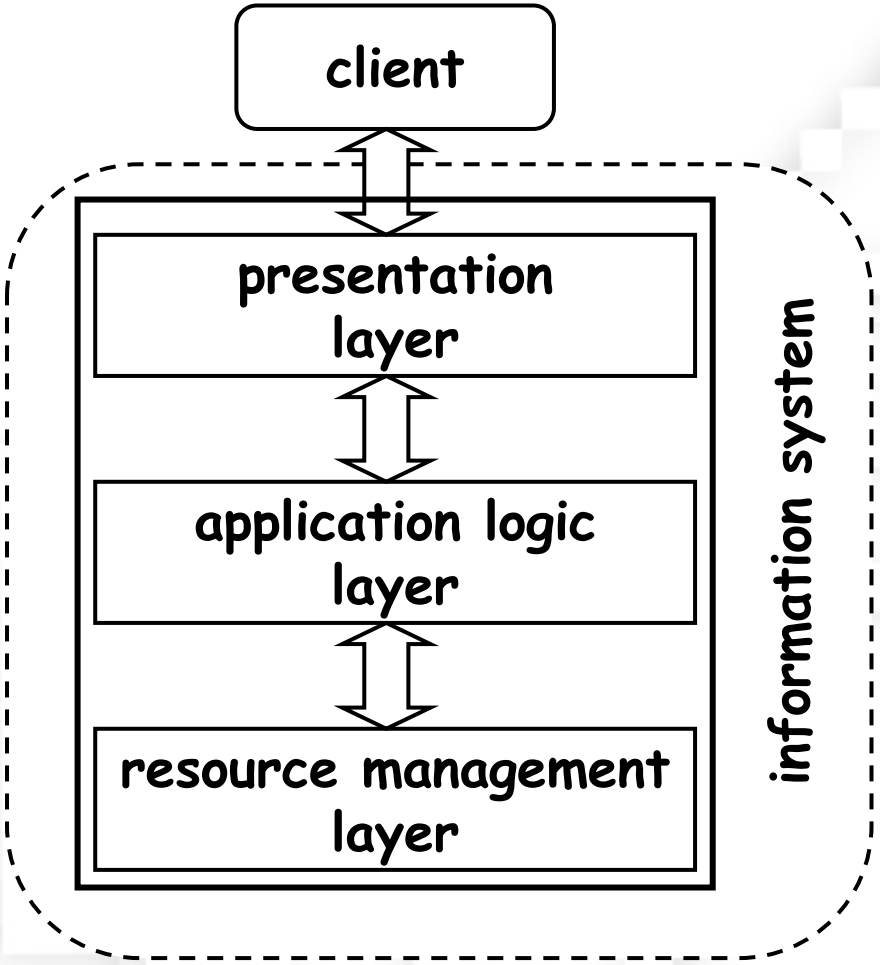
The resource manager deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic. This is typically a database but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence





Distributed Information Systems

1-tier Architecture

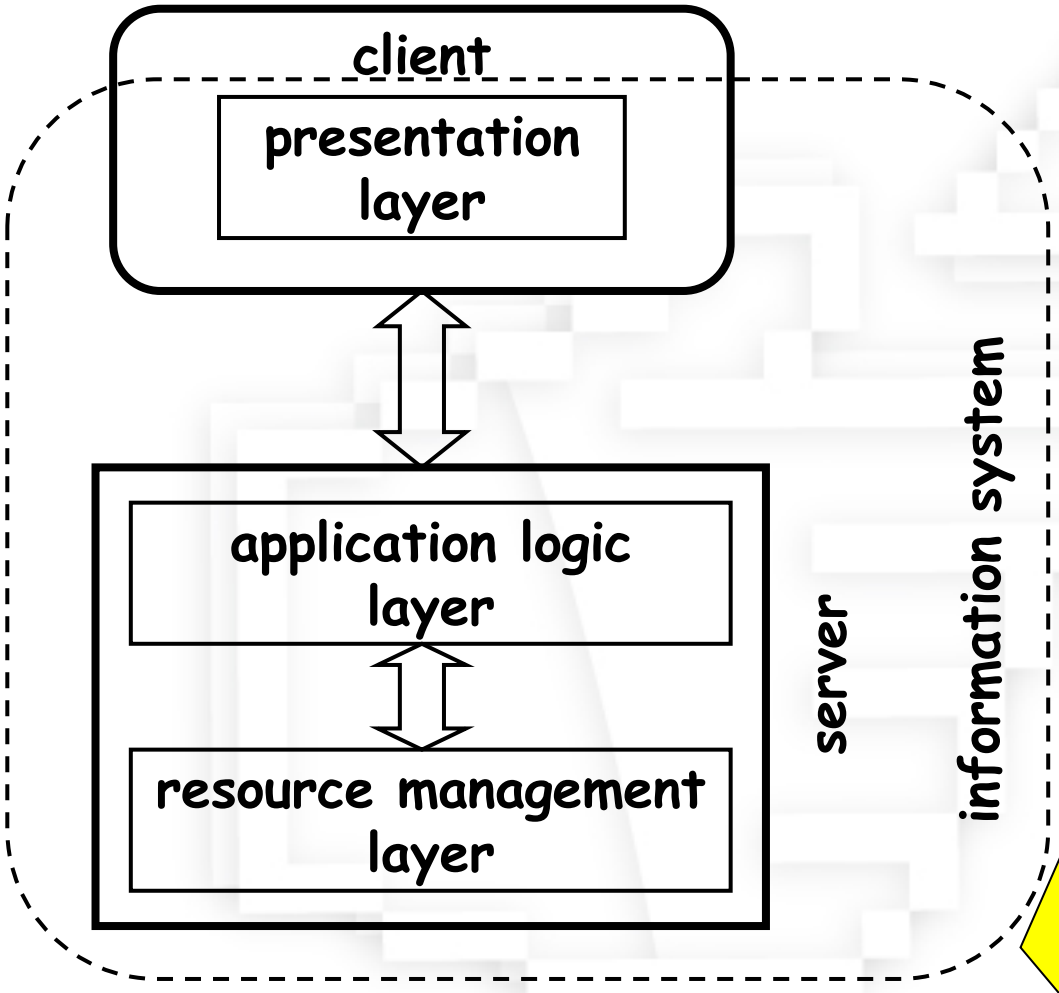


no forced context switches in the control flow (everything happens within the system)

all is centralized, managing and controlling resources is easier

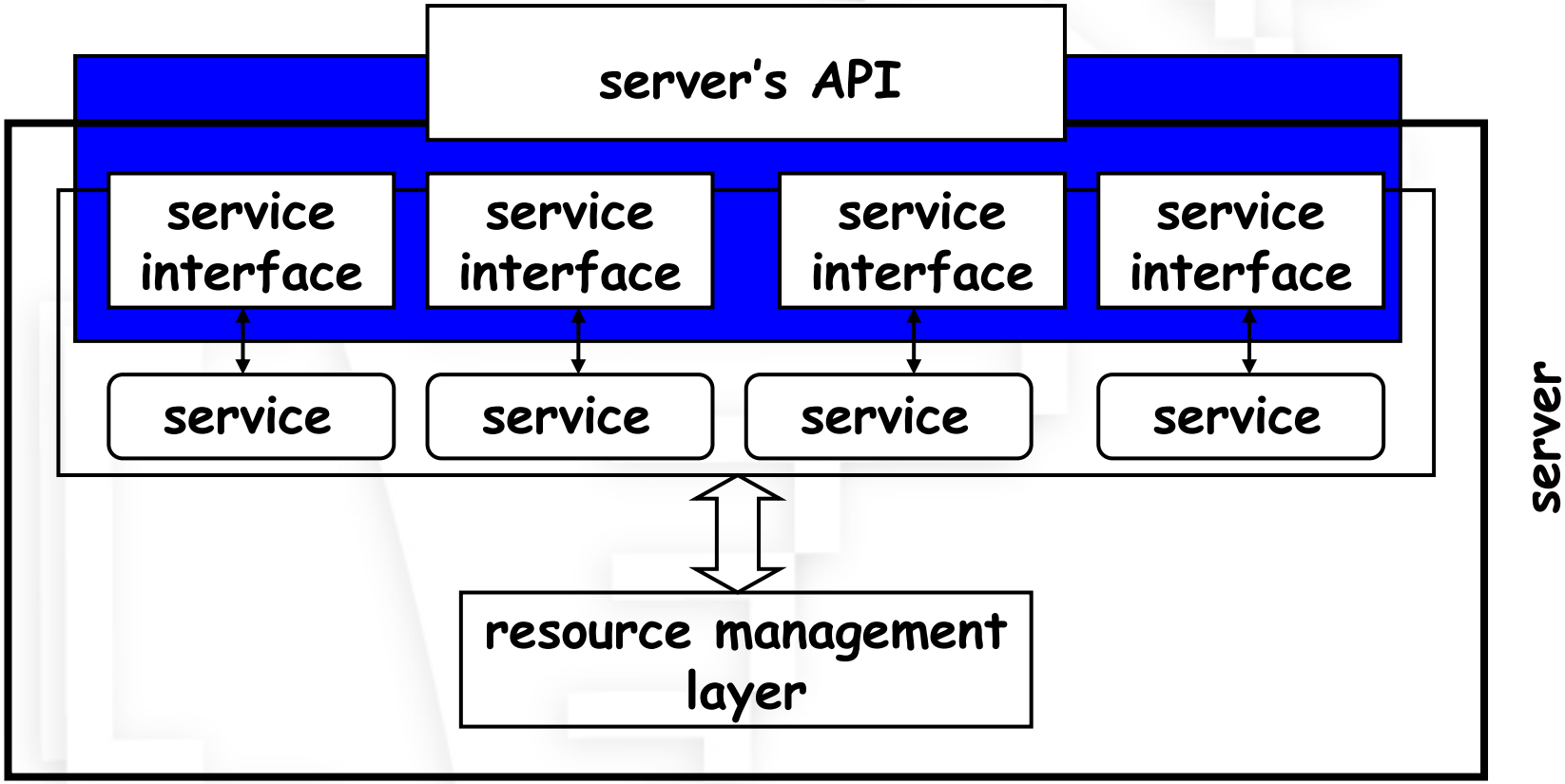
the design can be highly optimized by blurring the separation between layers.

2-tier Architecture

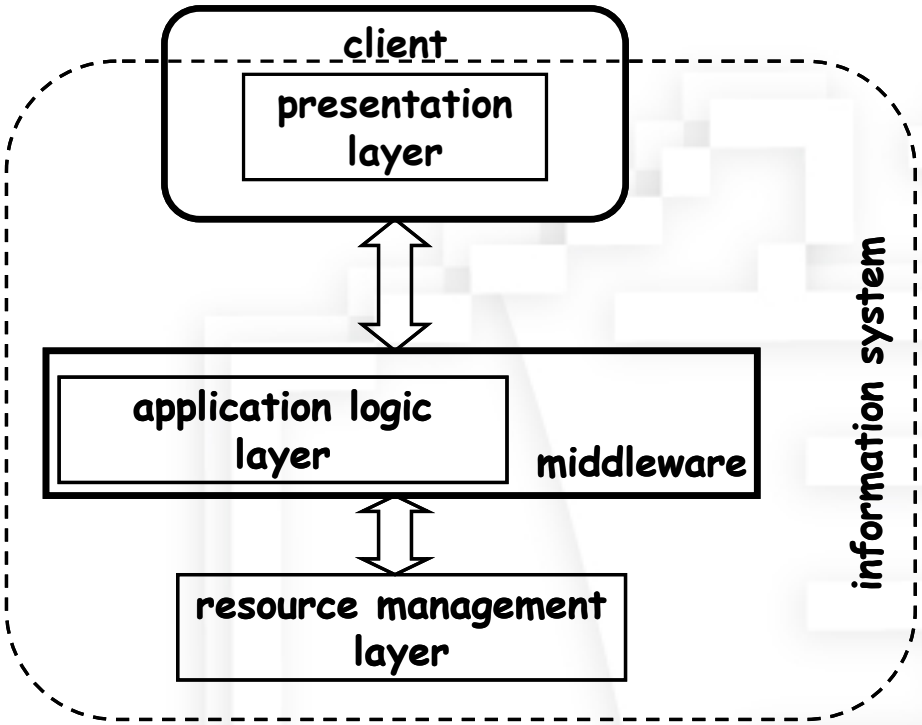


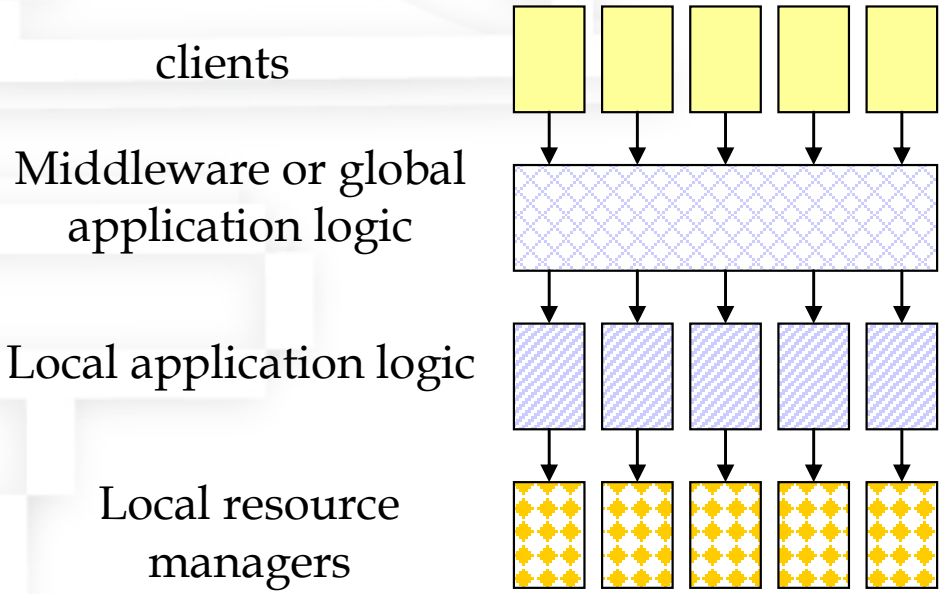
Clients are independent of each other: one could have several presentation layers depending on what each client wants to do.
One can take advantage of the computing power at the client machine to have more sophisticated presentation layers. This also saves computer resources at the server machine.
It 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.

Interfaces and “Services”



3-tier Architecture

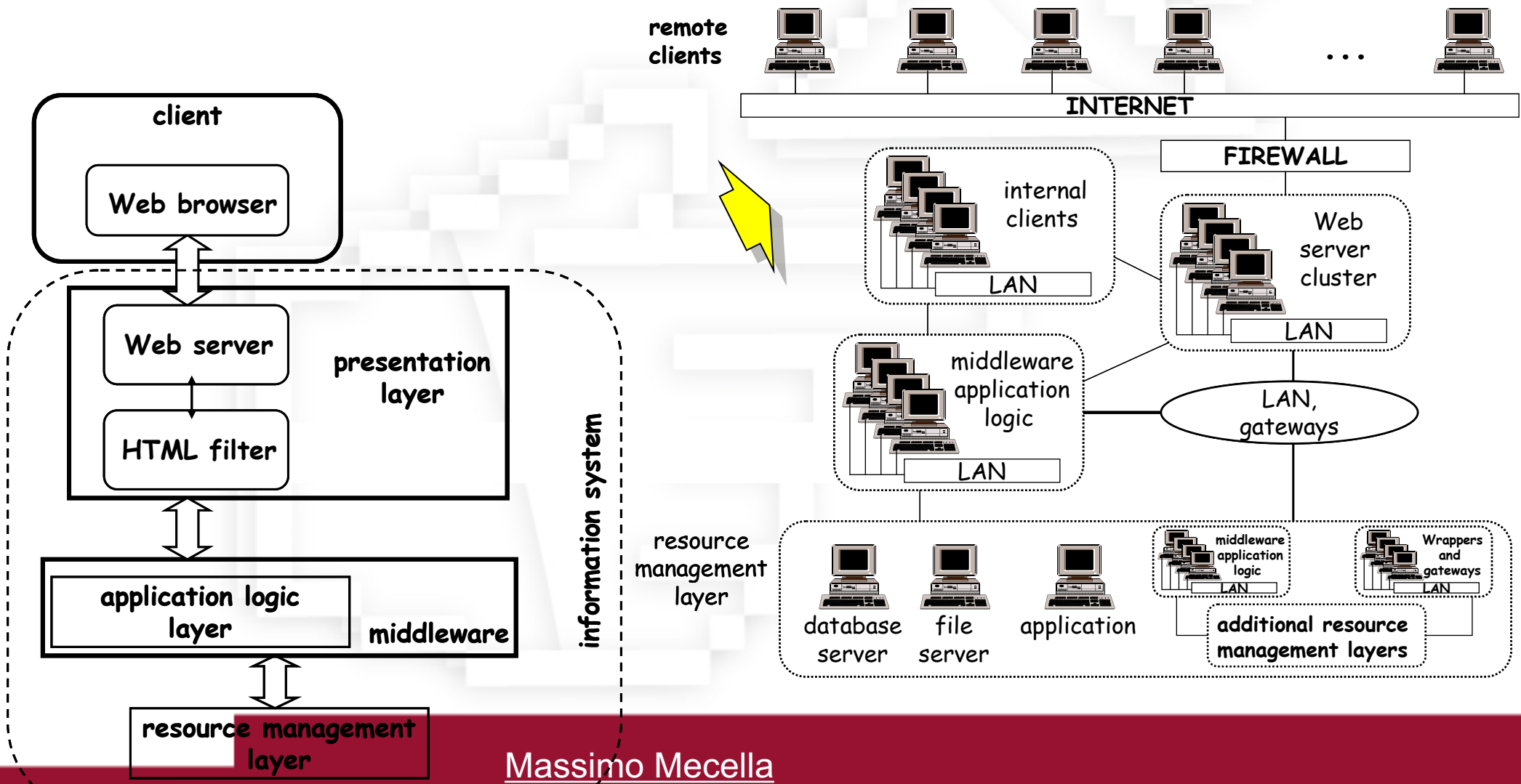




Distributed Information Systems

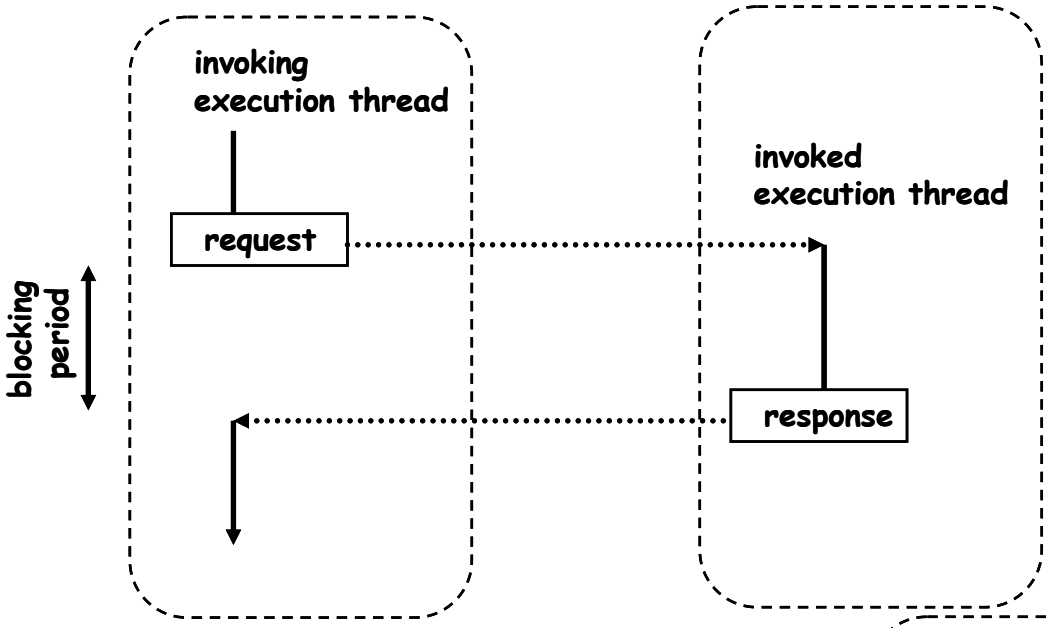
Middleware is just a level of indirection between clients and other layers of the system.
It introduces an additional layer of business logic encompassing all underlying systems.

N-tier Architecture



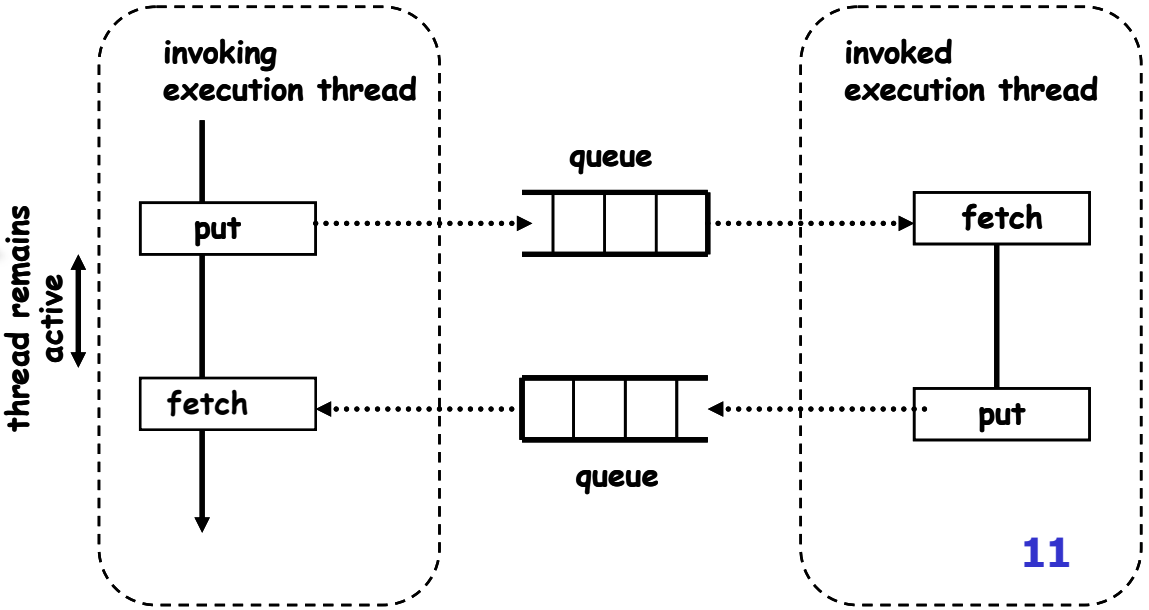
MIDDLEWARE

Communication primitives



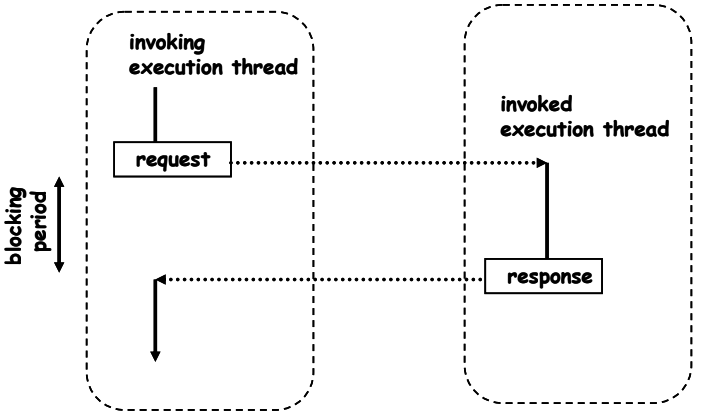
Blocking interaction (improperly «synchronous»)

Non-blocking interaction (improperly «asynchronous»)



Blocking or Synchronous Interaction

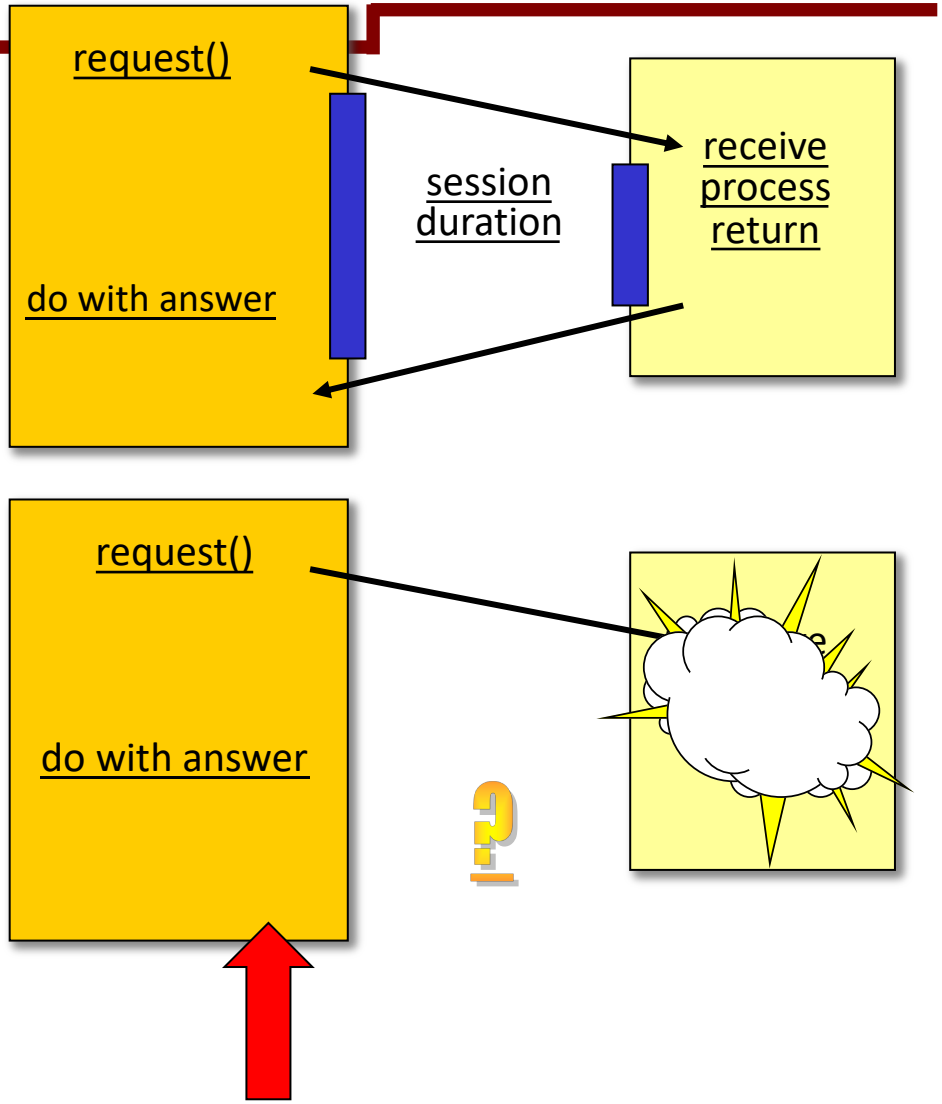
- Traditionally, distributed applications use blocking calls (the client sends a request to a service and waits for a response of the service to come back before continuing doing its work)
- Synchronous interaction requires both parties to be “on-line”: the caller makes a request, the receiver gets the 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 (e.g., some technologies 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)

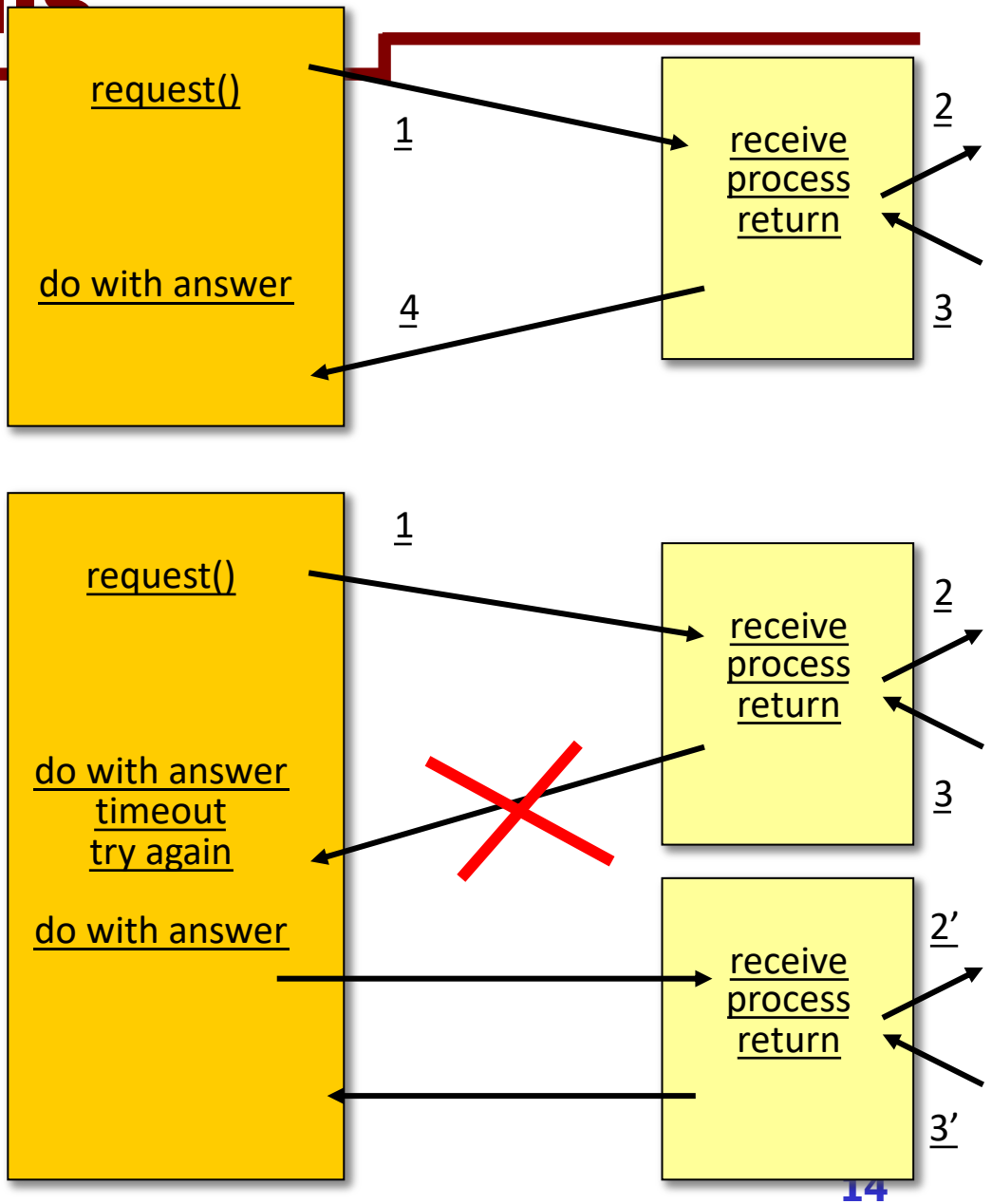
Overhead of Synchronism

- Synchronous invocations require to maintain a session between the caller and the receiver
- Maintaining sessions is expensive and consumes CPU resources. There is also a limit on how many sessions can be active at the same time (thus limiting the number of concurrent clients connected to a server)
- For this reason, client/server systems often resort to connection pooling to optimize resource utilization
  - have a pool of open connections
  - associate a thread with each connection
  - allocate connections as needed
- Synchronous interaction requires a context for each call and a context management system for all incoming calls. The context needs to be passed around with each call as it identifies the session, the client, and the nature of the interaction.



Failures in Synchronous Calls

- If the client or the server fail, the context is lost and resynchronization might be difficult.
  - If the failure occurred before 1, nothing has happened
  - If the failure occurs after 1 but before 2 (receiver crashes), then the request is lost
  - If the failure happens after 2 but before 3, side effects may cause inconsistencies
  - If the failure occurs after 3 but before 4, the response is lost but the action has been performed (do it again?)
- Who is responsible for finding out what happened?
- Finding out when the failure took place may not be easy. Worse still, if there is a chain of invocations (e.g., a client calls a server that calls another server) the failure can occur anywhere along the chain.



Two Solutions

ENHANCED SUPPORT

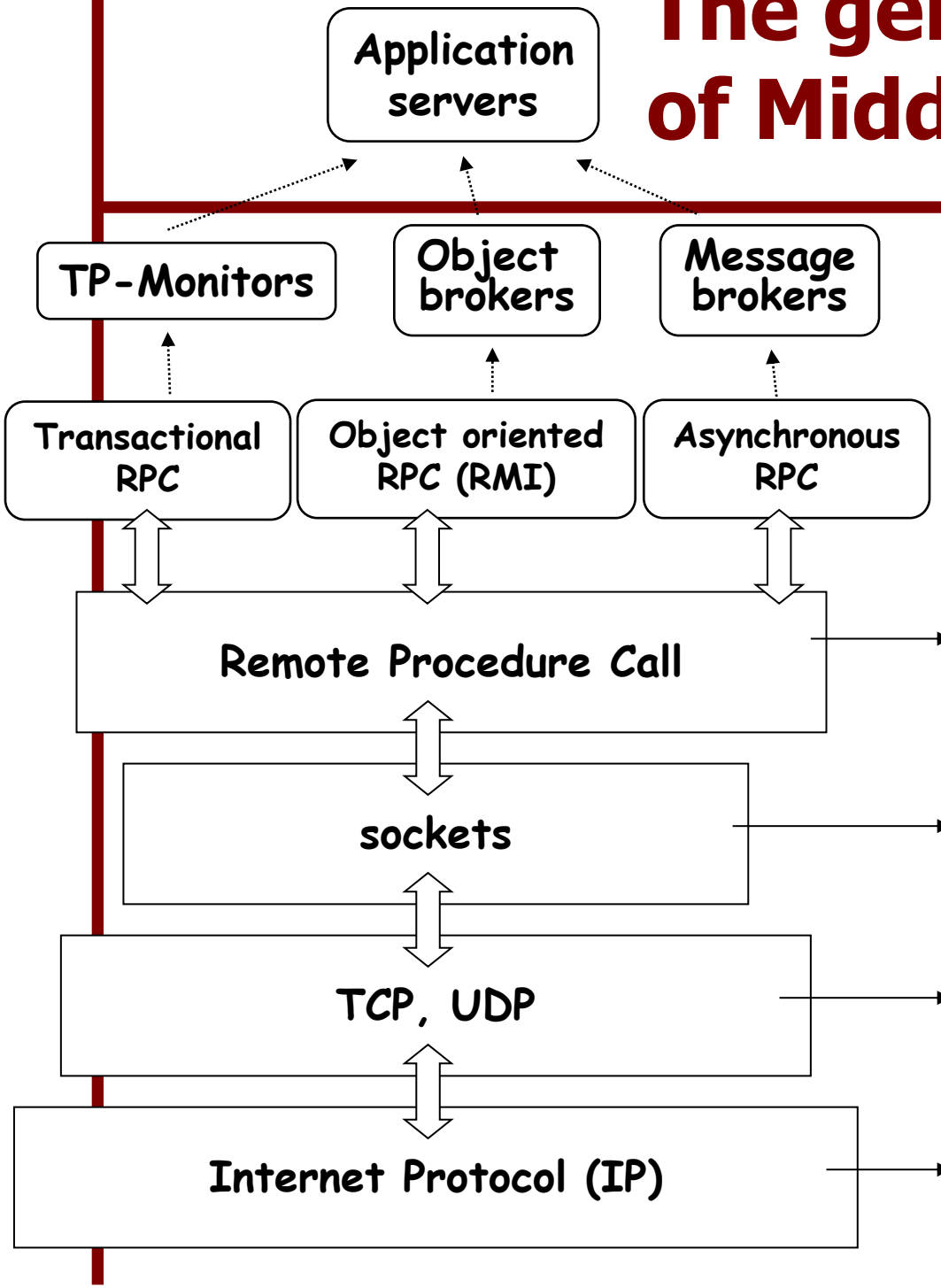
- Client/server systems and middleware platforms provide a number of mechanisms to deal with the problems created by synchronous interaction:
  - Transactional interaction: to enforce exactly once execution semantics and enable more complex interactions with some execution guarantees
  - Service replication and load balancing: to prevent the service from becoming unavailable when there is a failure (however, the recovery at the client side is still a problem of the client)

ASYNCHRONOUS INTERACTION

- Using asynchronous interaction, the caller sends a message that gets stored somewhere until the receiver reads it and sends a response. The response is sent in a similar manner
- Asynchronous interaction can take place in two forms:
  - non-blocking invocation (a service invocation but the call returns immediately without waiting for a response, similar to batch jobs)
  - persistent queues (the call and the response are actually persistently stored until they are accessed by the client and the server)

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti
Sapienza
Università di Roma

The genealogy of Middleware



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

Understanding Middleware

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

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)

Middleware as a Programming Abstraction

- 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

Middleware as Infrastructure

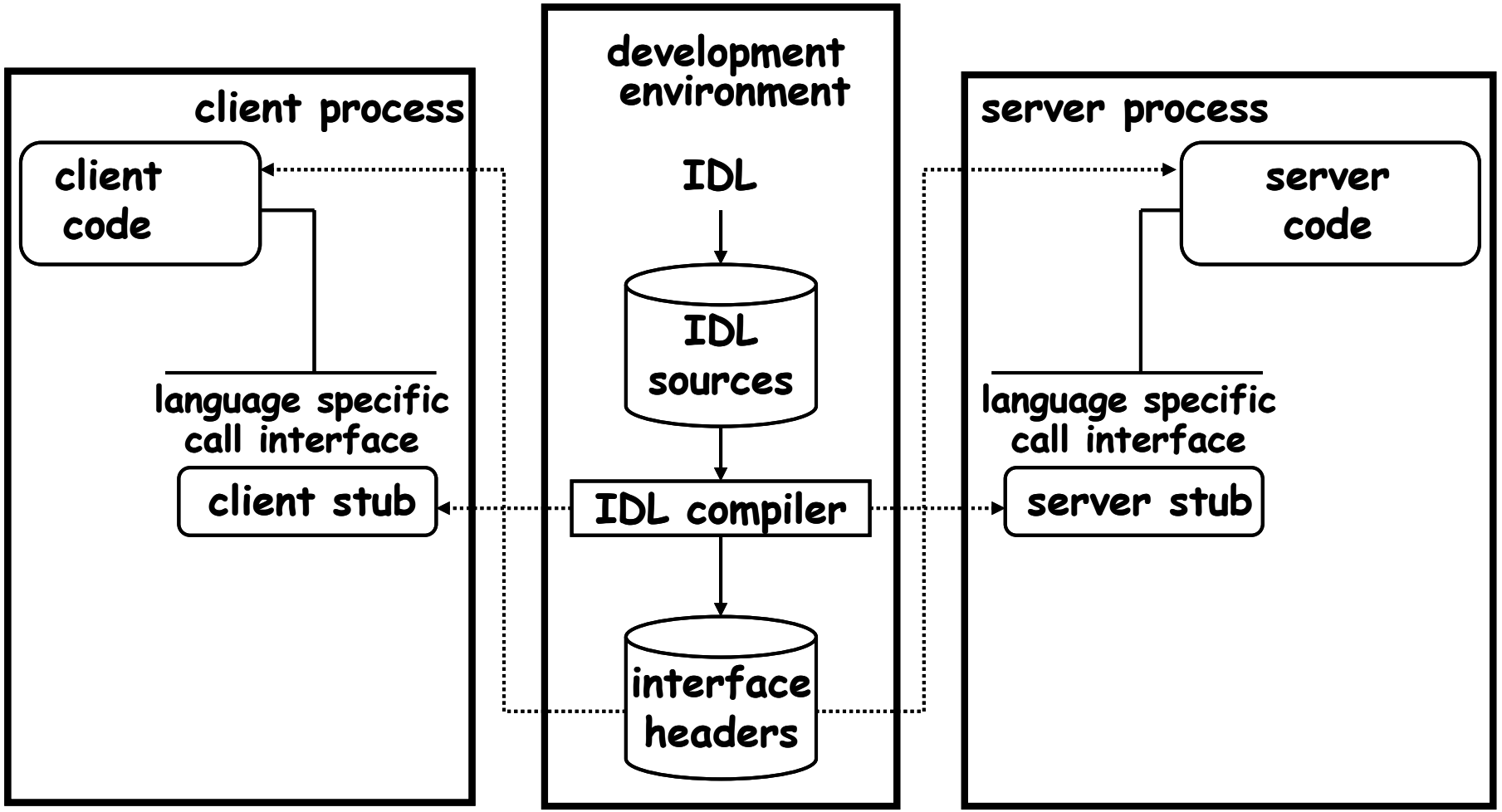
Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

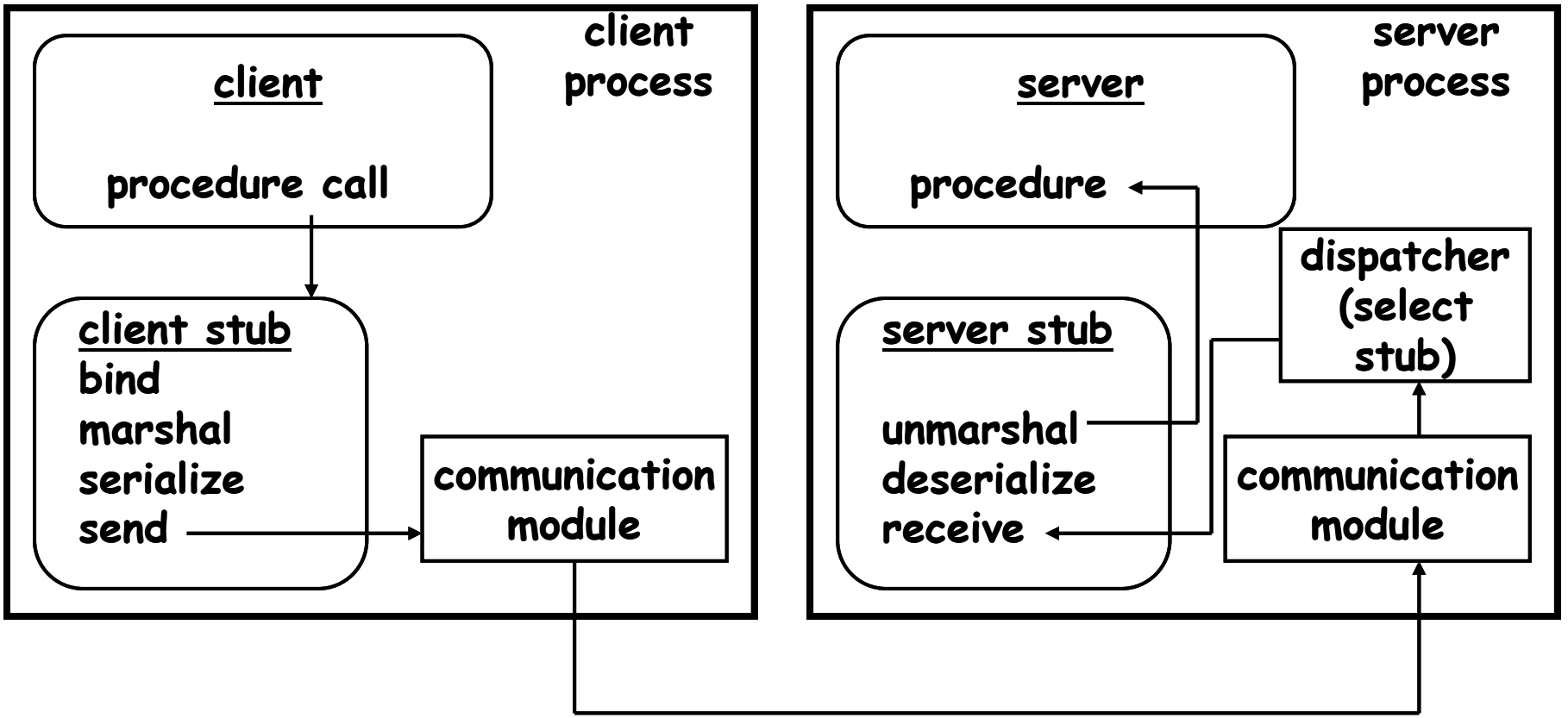
- As the programming abstractions reach higher and higher levels, the underlying infrastructure implementing the abstractions must grow accordingly
  - Additional functionality is almost always implemented through additional software layers
  - The additional software layers increase the size and complexity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

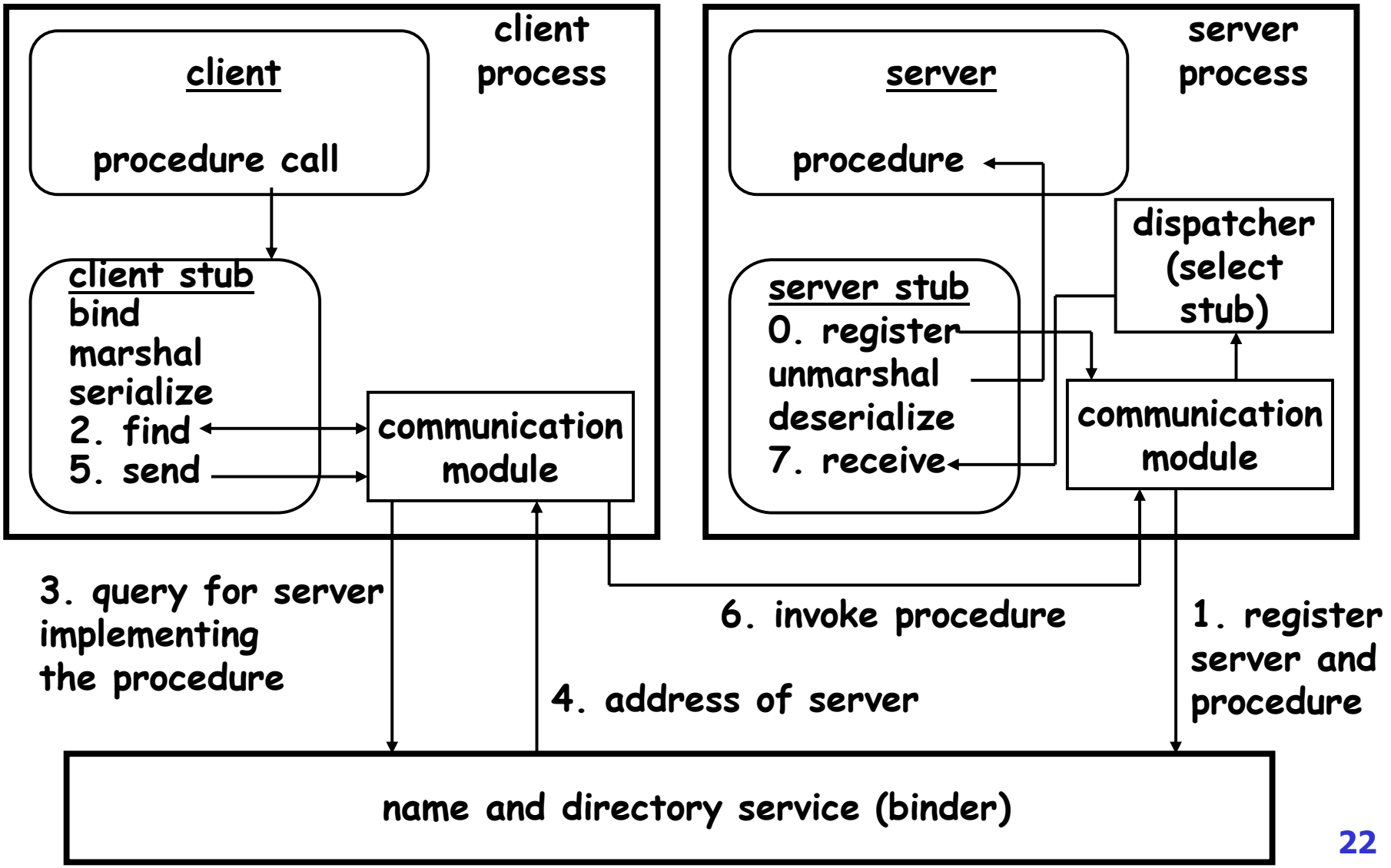
Basic Middleware: RPC (1)



Basic Middleware: RPC (2)



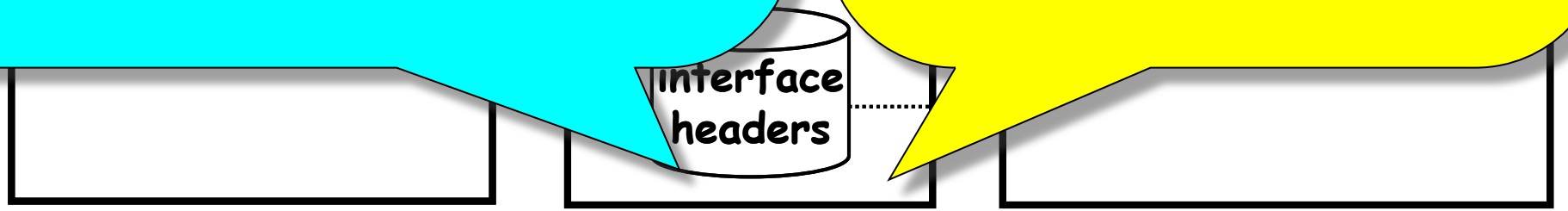
Basic Middleware (3)



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

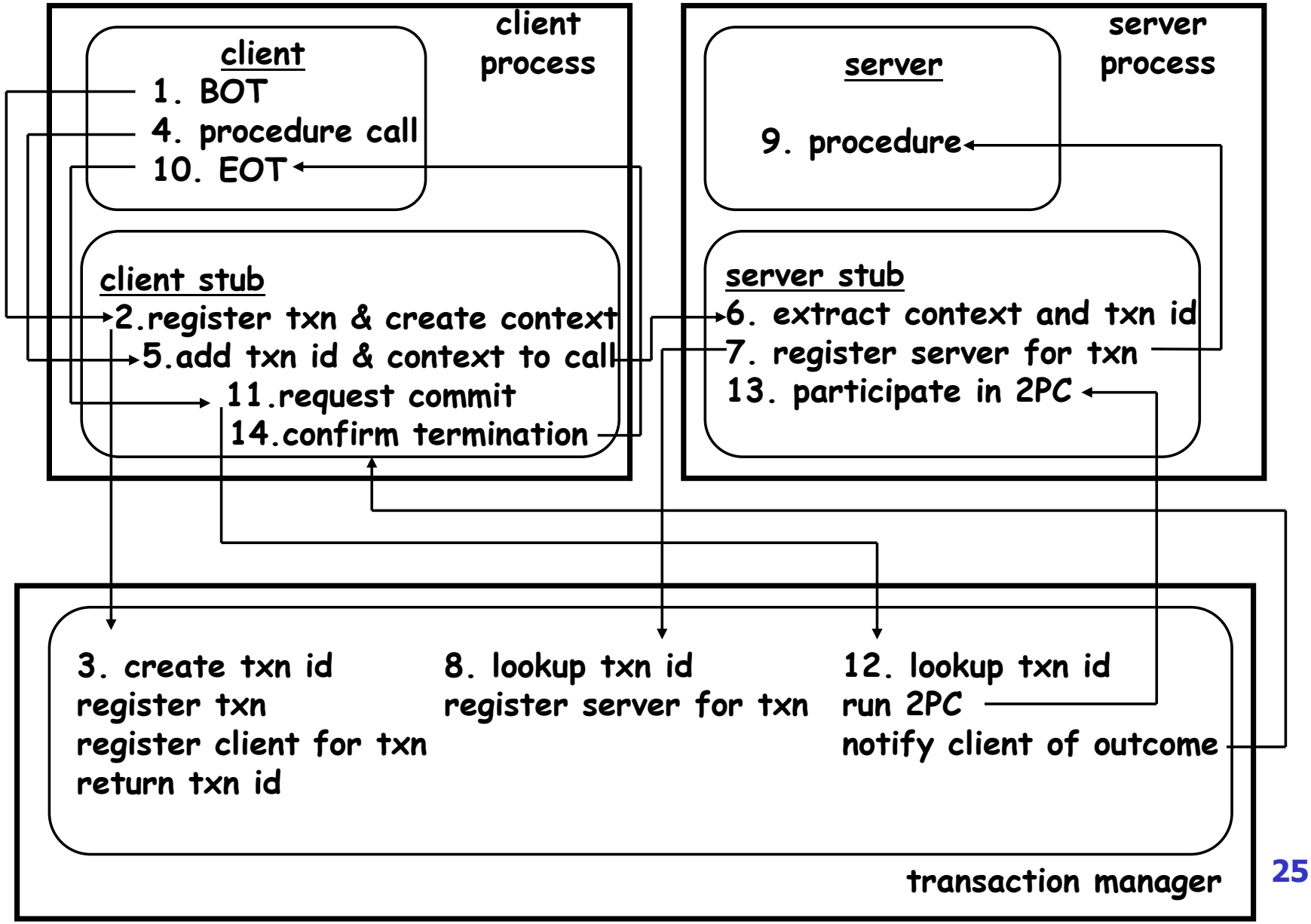
- One cannot expect the programmer to implement a complete infrastructure for every distributed application. Instead, one can use an RPC system (our first example of low level middleware)
- What does an RPC system do?
  - Hides distribution behind procedure calls
  - Provides an interface definition 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



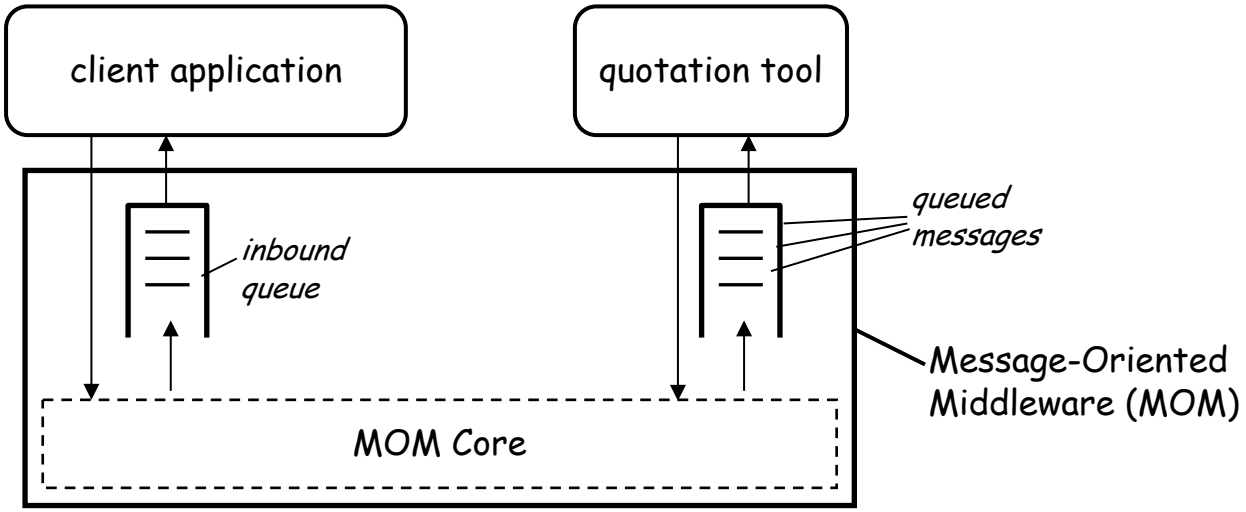
Transactional RPC

- 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.)

Transactional RPC

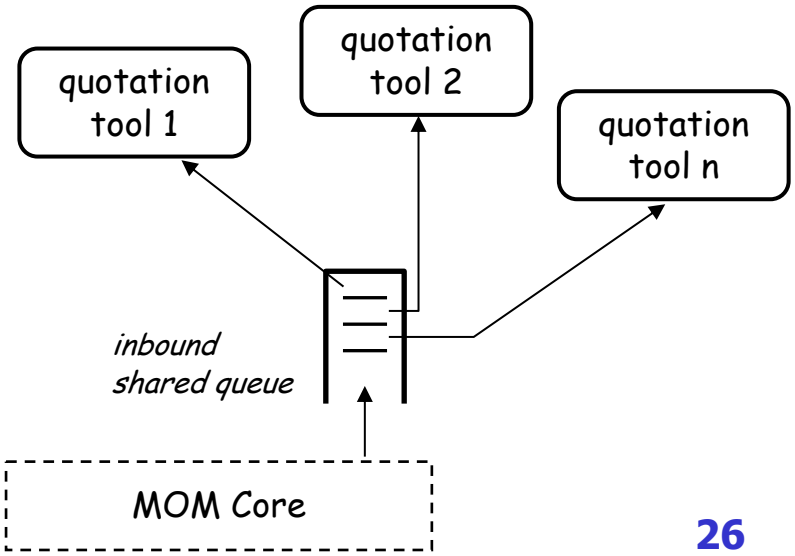


Message Oriented Middleware



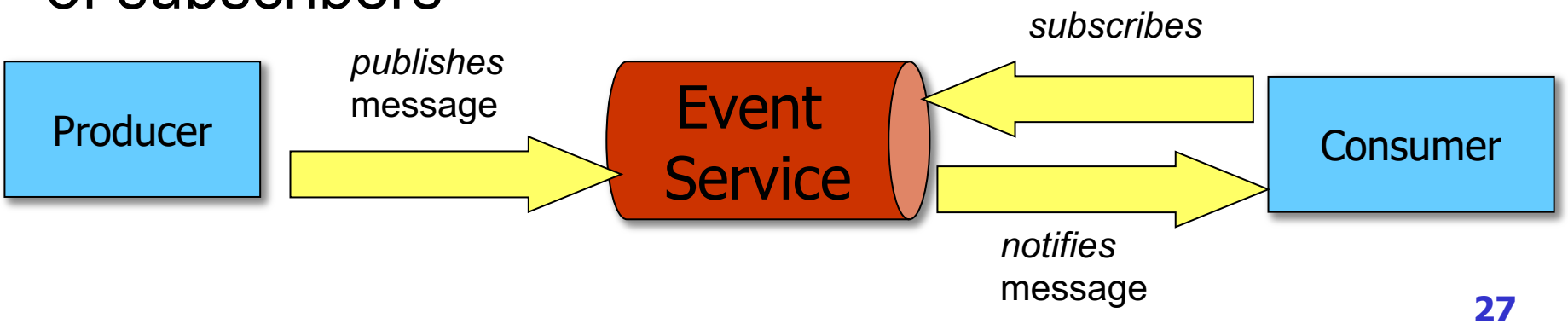
The queuing model

(Primitive) load balancing with shared queues



Publish/Subscribe

- A many-to-many anonymous interaction
- The participants are divided into two groups
  - publisher: producers, send messages
  - subscriber: consumers, express interest in certain categories of messages
- Communication takes place through a central entity (event service) that receives messages from publishers and cross-check them with the interests of subscribers



Publish/Subscribe

- **Topic-based:** information is divided into topics (topic or subject)
  - subscriptions and publications by topic
  - ideal logical channel that connects a publisher to all subscribers
- **Content-based:** filters can be used for a more accurate selection of information to be received (message content)
- The notification can be made in two ways
  - push: the subscribers are invoked in callback, using a reference communicated at the time of subscription
  - pull: the subscribers poll the event service when they need messages

Example

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

With P&S topic-based a subscriber registers its interest for messages of type `QuoteRequest` (~ interest for the "class")

With P&S content-based a subscriber registers its interest for messages of type `QuoteRequest` such that `RequestedDeliveryDate before June 30th, 2004 AND Quanity > 1000`

You can write expressions that allow you to filter all the messages of a type with respect to the attribute values (selection of the "instance"). The expressive power of the subscription language becomes important (a sort of query language)

Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

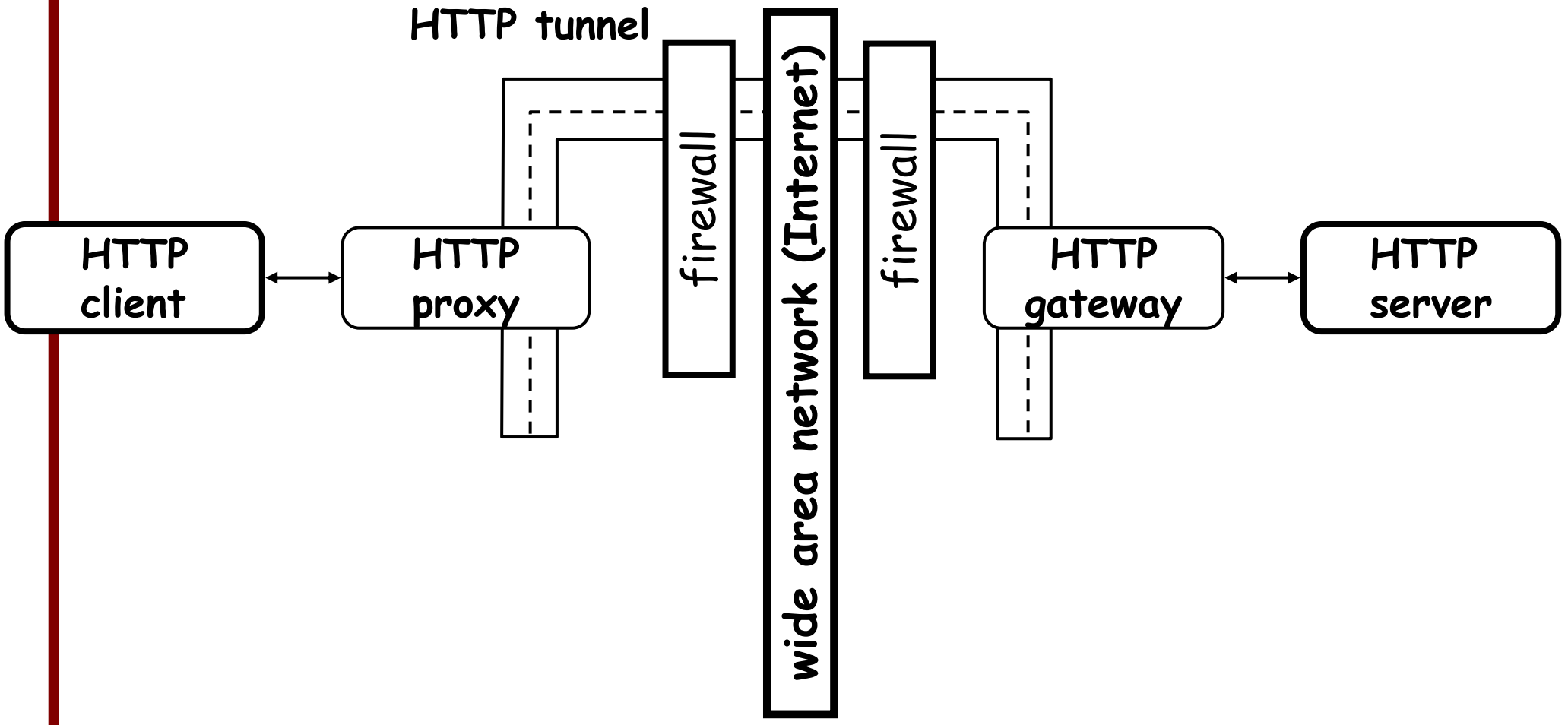
Sapienza
Università di Roma

MIDDLEWARE ON THE INTERNET

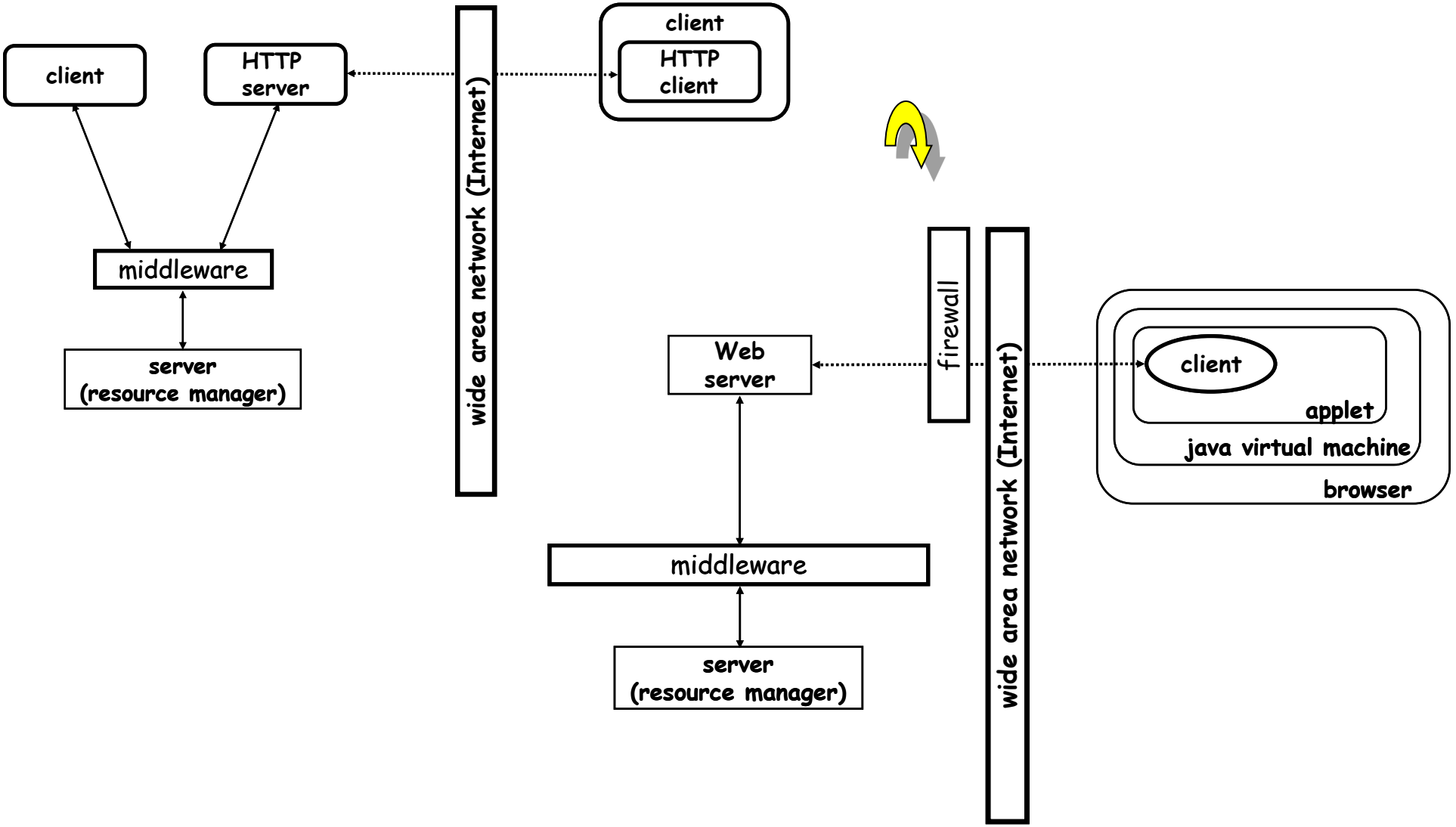
Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti

Sapienza
Università di Roma

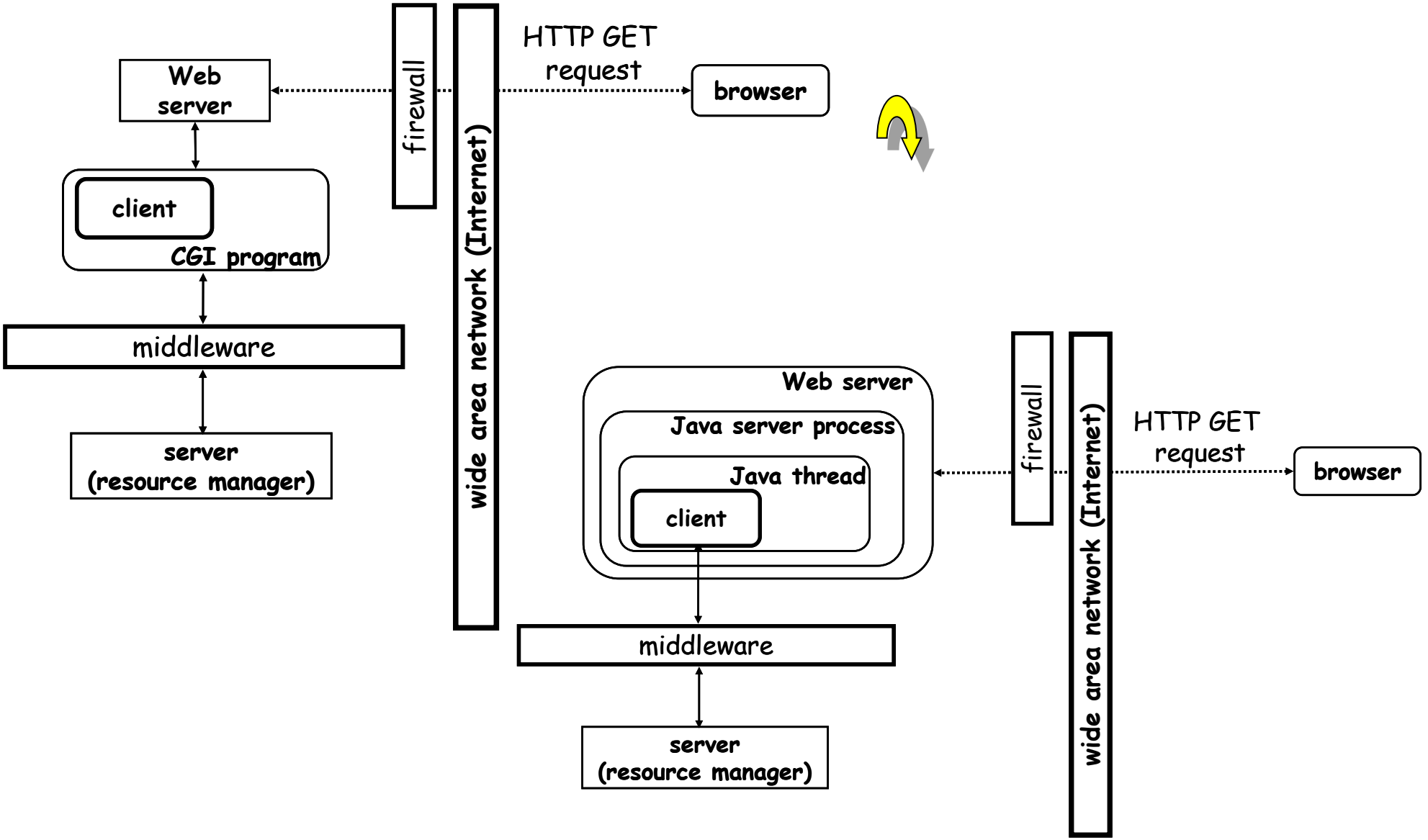
Proxy and Firewall



3-tier on the Web: Support for Remote Clients

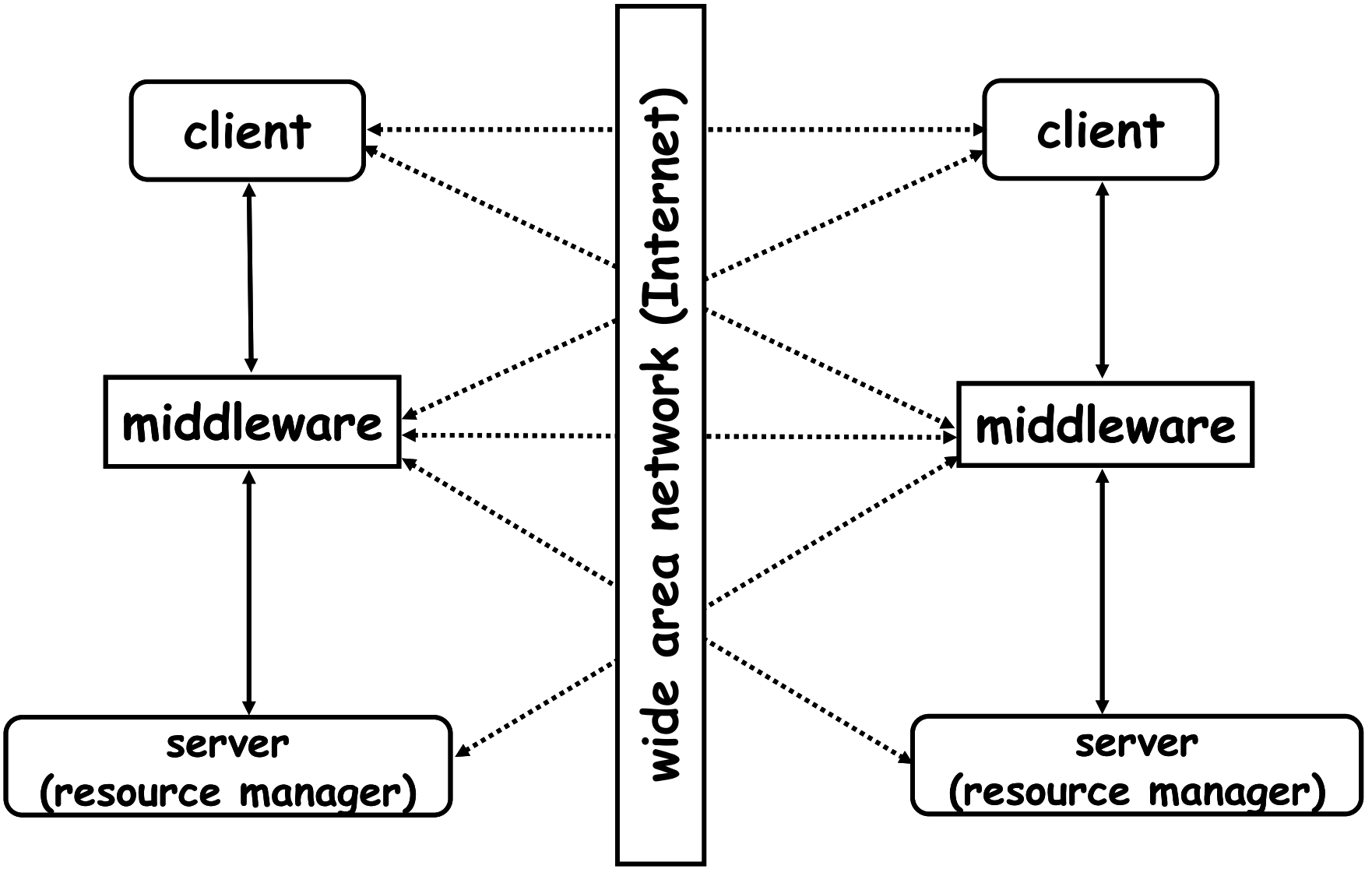


3-tier on the Web: Server Pages



Dipartimento di Ingegneria Informatica
Automatica e Gestionale Antonio Ruberti
Sapienza
Università di Roma

Integration Across the Internet

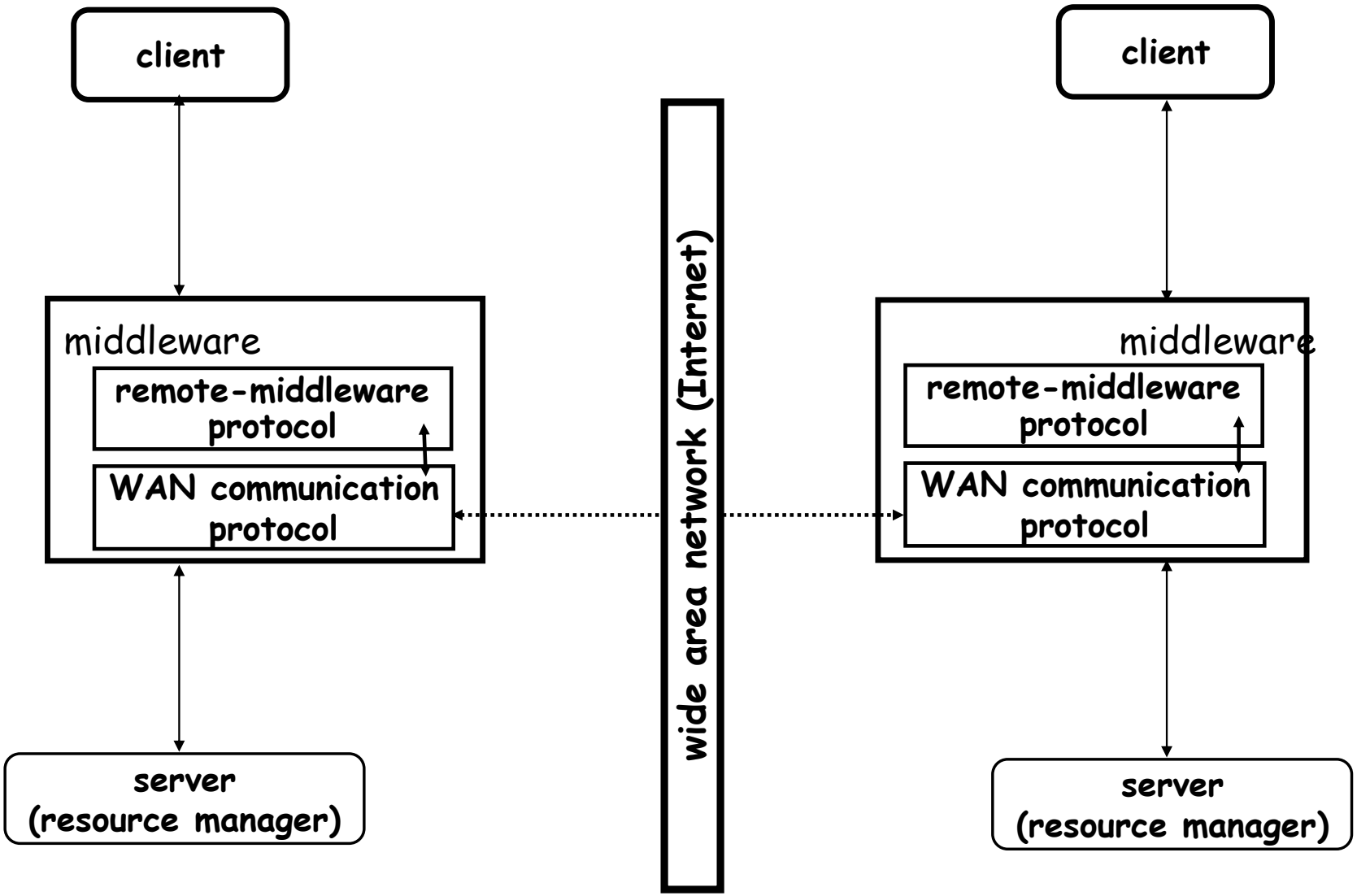


Integration M2M (middleware-to-middleware)

...

DIPARTIMENTO DI INGEGNERIA INFORMATICA AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA UNIVERSITÀ DI ROMA



... may require HTTP Tunnelling/Support