Architectural styles for software systems

The client-server style

Prof. Paolo Ciancarini
Software Architecture
CdL M Informatica
Università di Bologna

Agenda

- Client server style
- CS two tiers
- CS three tiers
- CS n-tiers

Client-Server (CS): Overview

- CS has been conceived in the context of distributed systems, aiming to solve a problem of resources shared among several computers
- The driving idea is to make unbalanced the partners' roles within a communication process
- One or many *servers* provide services to instances of subsystems, called *clients*
- Each client calls on the server, which performs some service and returns the result

Client-Server: Overview

- Functions performed by **client**:
  - Requires a service through the server interface(s)
  - Allows to users to communicate with the system through a certain input (Customized user interface)
  - Front-end processing of input data
- Functions performed by **server**:
  - Centralized data management
  - Services provider
  - Back-end processing of the data provided by client

Client server pattern: main

- A client-server system can be logically divided into three parts:
  - The **presentation tier**, in charge of managing the user interface (graphic events, input fields check, etc…)
  - The actual **application logic tier**
  - The **data management tier** for the management of persistent data and transaction managers

The business logic tier

- Some patterns help in structuring the business logic:
  - Transaction script: a procedure that takes the input from the presentation, processes it with validations and calculations, stores data in the database, and invokes any operations from other systems. It then replies with more data to the presentation, perhaps doing more calculation to help organize and format the reply.
  - Domain model: an object model of a domain incorporates both behavior and data
  - Table module: a single instance that handles the business logic for all rows in a database table or view (as in relational database)

Fowler, Patterns of enterprise application architecture

Client-Server pattern

- A client-server system architecture can be:
  - 2-tiered: presentation and application logic modeled as a single tier contained in the client
  - 3-tiered: one tier for each part, with dedicated servers for application logic and data management
  - n-tiered: the last two parts spread in a chain of servers

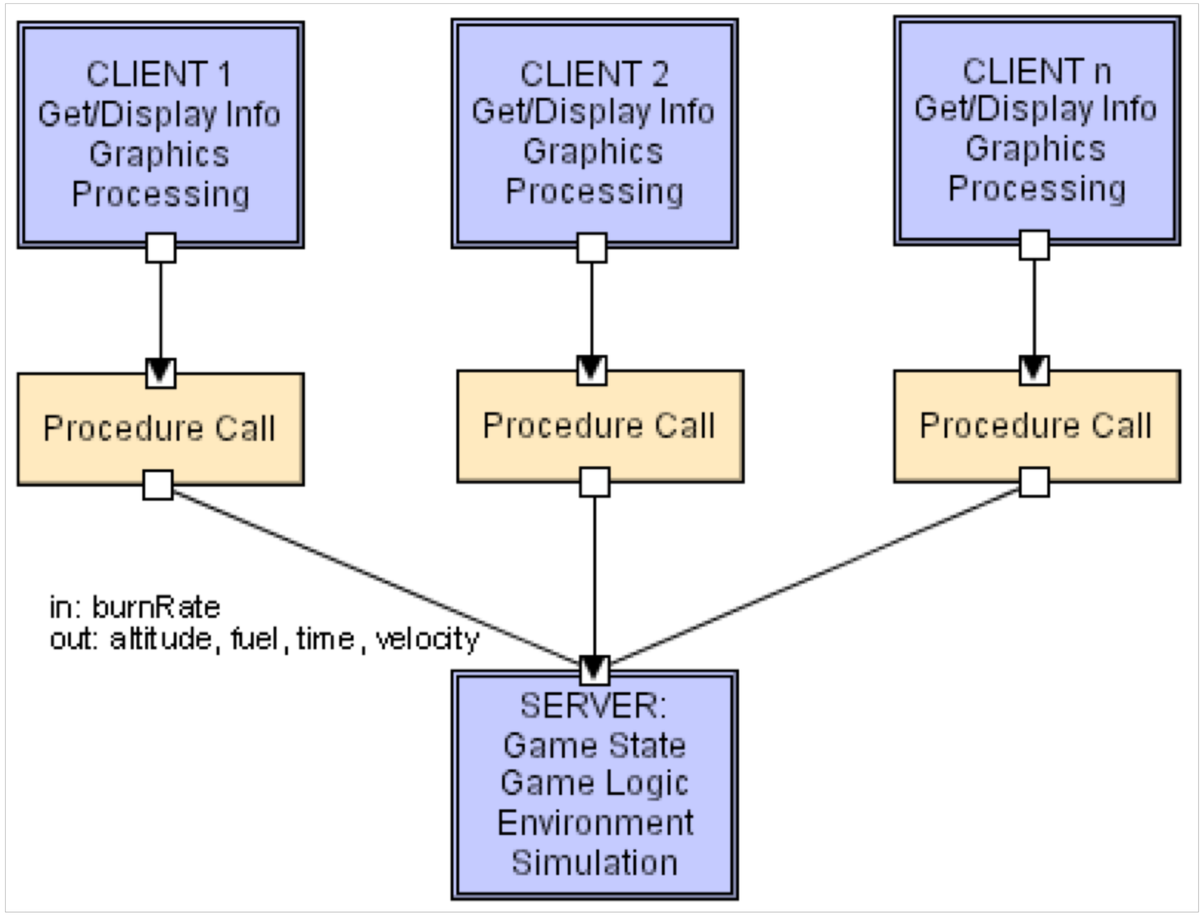
Client: CRC

Class Client	**Collaborators**
Responsibilities •Component: handles User interaction •Asks the Server for some service	• Server

Server: CRC

Class Server	**Collaborators**
Responsibilities • Component (offers some services) • Provides an API for receiving/answering messages	

Client-Server: Architecture



CS 2-tiers: Overview

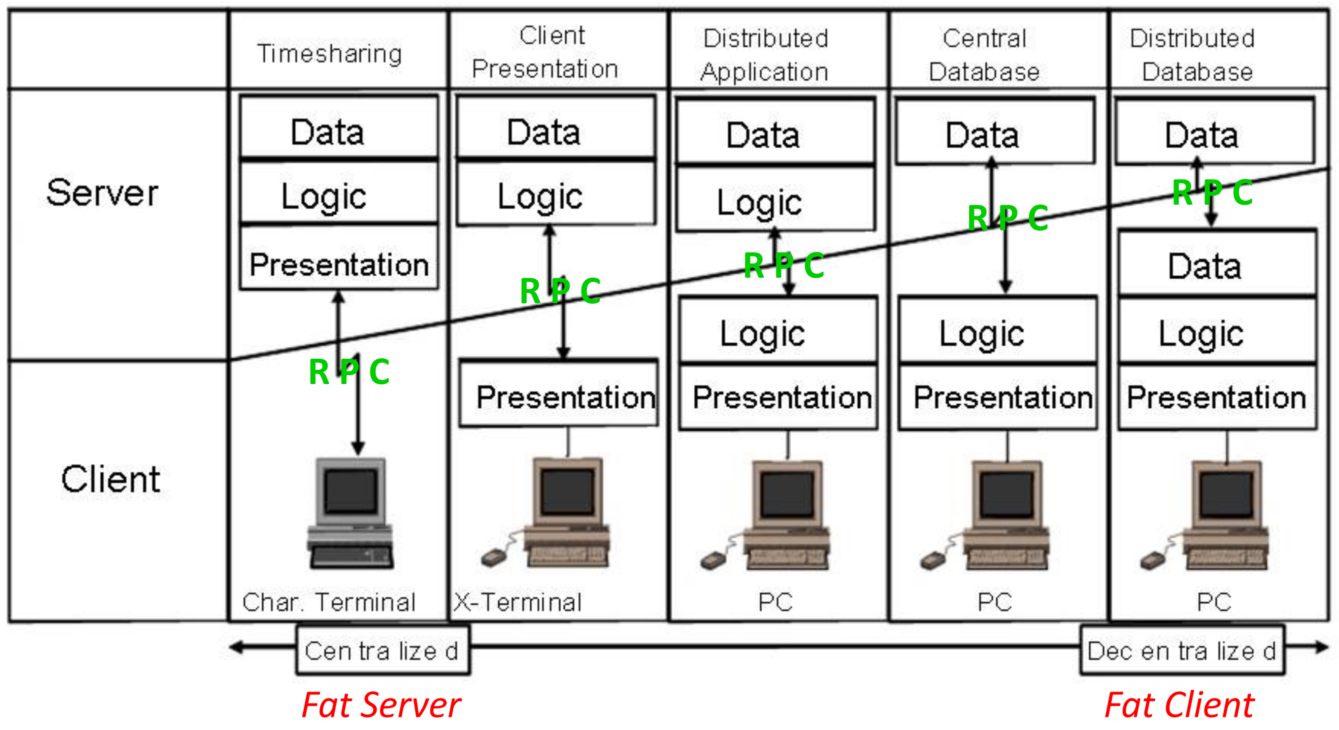
- This style is used to describe client-server systems where the client requests resources and the server responds directly to the request using its own resources
- The server does not call on another application in order to provide part of the service
- If the server is more powerful than its clients, it is possible to connect many clients at the same time

CS 2-tiers: Architecture (1)

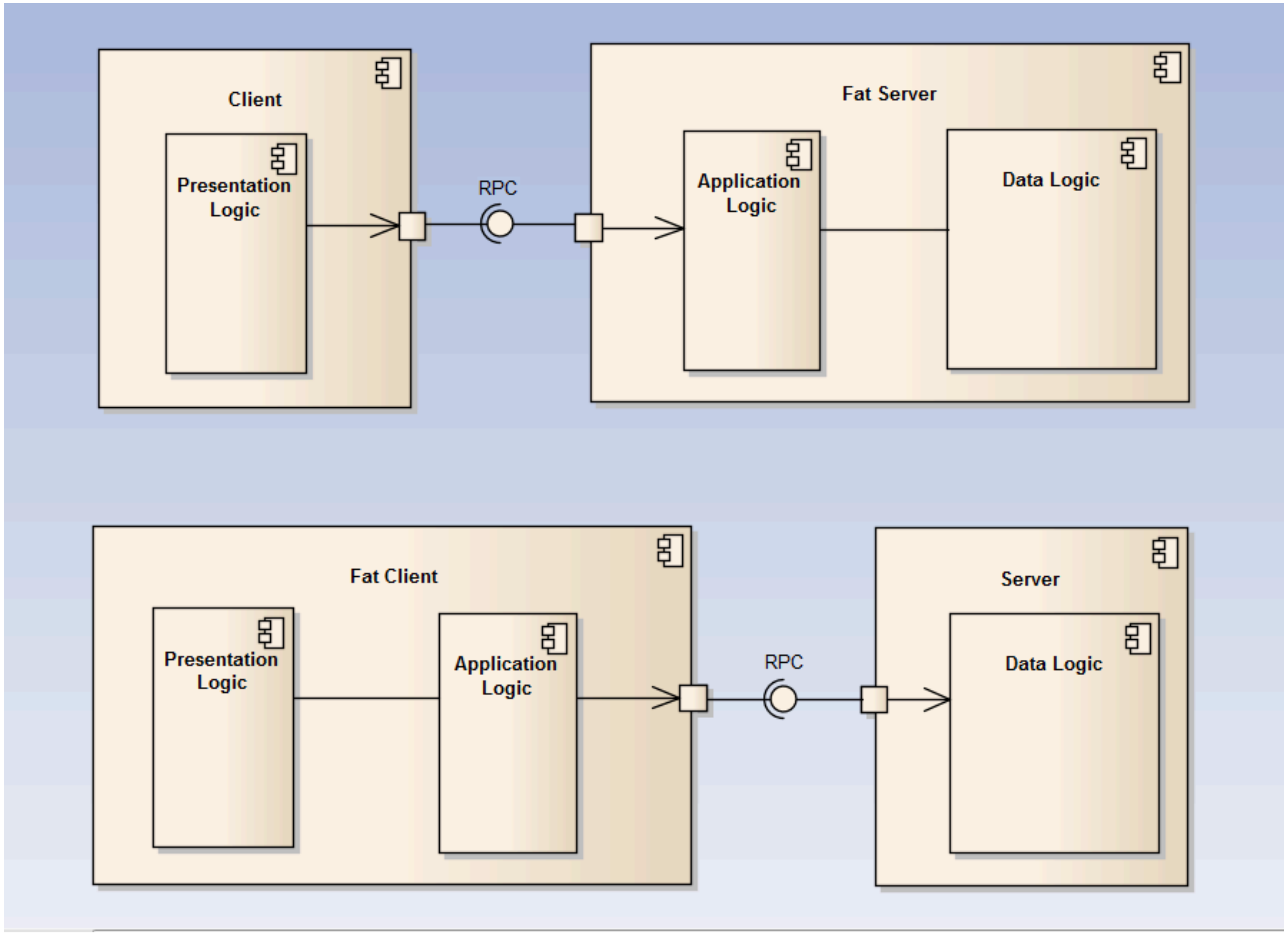
- Components:
  - **Client**
    - Active entity
    - Contains always the User System Interface subsystem (session, text input, dialog, and display management services)
    - If it contains the Application Logic subsystem (process management), than it is a *Fat Client*
  - **Server**
    - Reactive entity
    - Contains a Database Management subsystem (such as data and file services)
    - If it contains the Application Logic subsystem, than it's a *Fat Server*
- Connectors:
  - **Procedure Calls, RPC, or others**
    - They require a protocol

CS 2-tiers: Architecture (2)

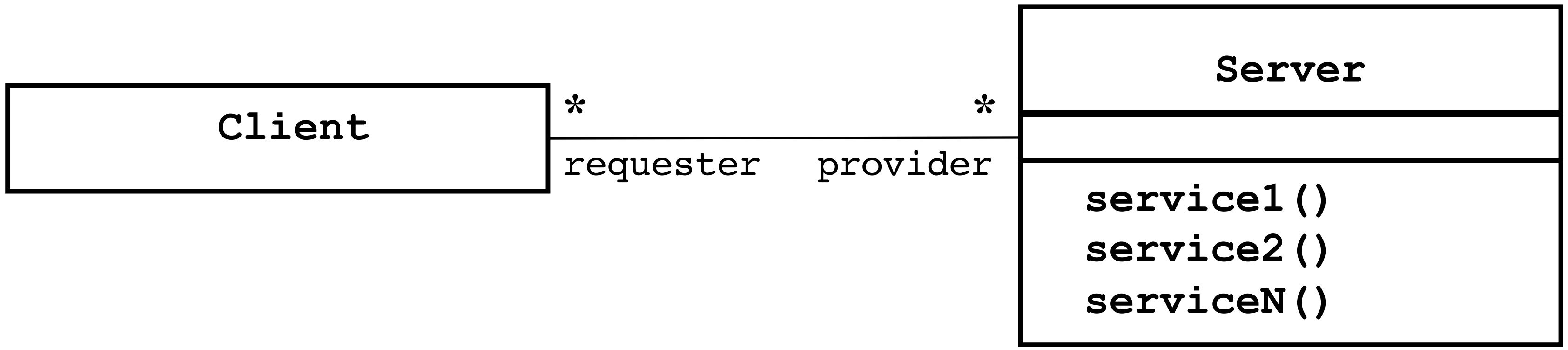
- The Application Logic may be present either at the *client side* within a user interface or within the database on the *server side* or spread on the both



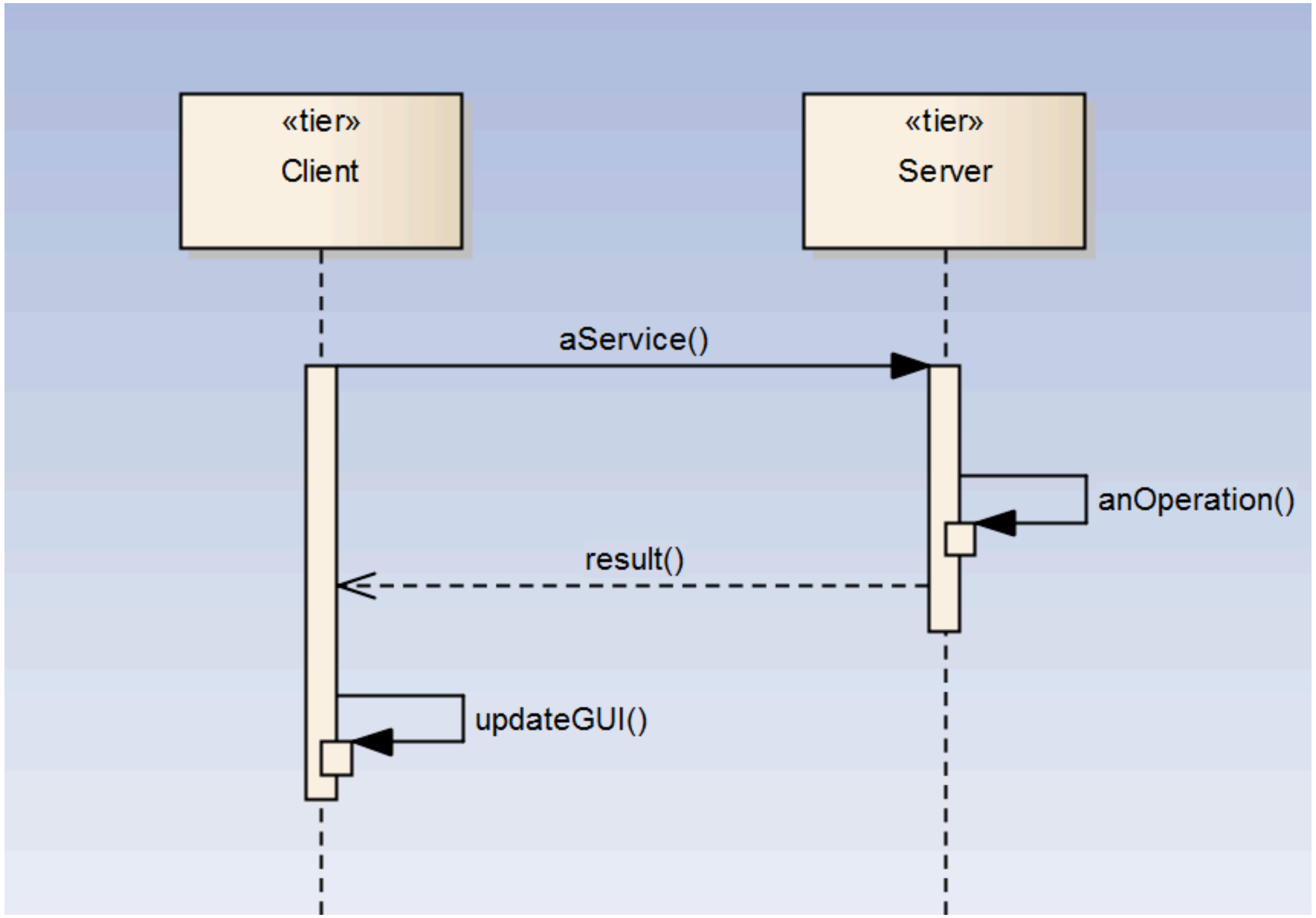
CS 2-tiers: Component Diagram



CS 2-tiers: Class Diagram



CS 2-tiers: Sequence Diagram



CS 2-tiers example (1): web browser and web server

HTTP (Hypertext Transfer Protocol):

- *ASCII-based request/reply protocol* that runs over TCP
- HTTPS: variant that first establishes symmetrically-encrypted channel via public-key handshake, suitable for sensitive info

- By convention, servers listen on TCP port 80 (HTTP) or 443 (HTTPS)
- Universal Resource Identifier (URI) format: scheme, host, port, resource, parameters, fragment
  - http://search.com:80/img/search/file?t=banana&client=firefox#p2

CS example: World Wide Web

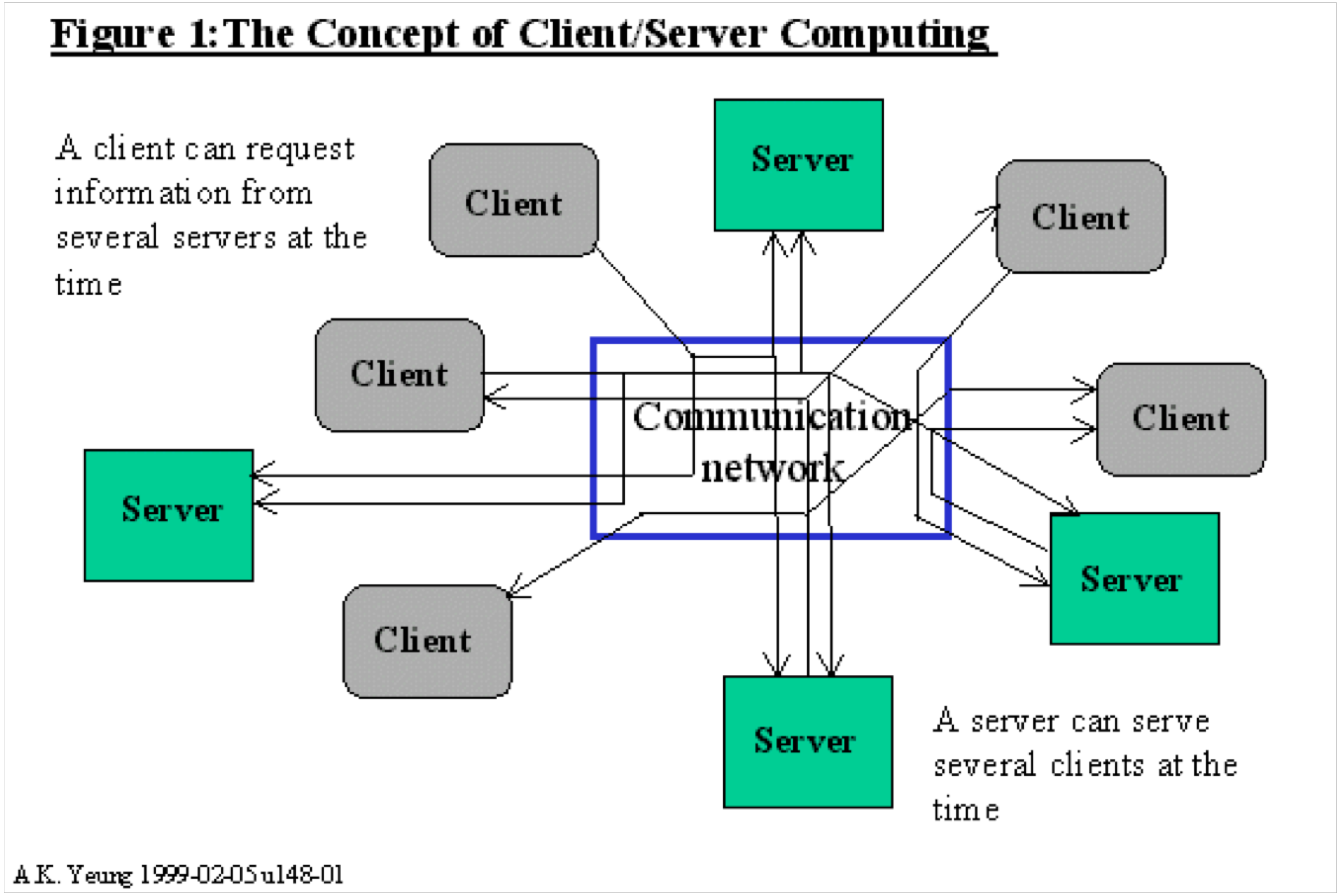


Figure 1: The Concept of Client/Server Computing
A client can request information from several servers at the time
Client
Server
Client
Client
Communication network
Client
Server
Server
Client
Server
A server can serve several clients at the time
A.K. Yeung 1999-02-05 u148-01

CS 2-tiers example (2): web browser and web server

- Client's request:

HTTP method & URI

```
GET /index.html HTTP/1.0
User-Agent: Mozilla/4.73 [en] (X11; U; Linux 2.0.35 i686)
Host: www.yahoo.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png,
   */*
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

- Server's reply:

Cookie data: up to 4KiB

MIME content type

```
HTTP/1.0 200 OK
Content-Length: 16018
Set-Cookie: B=2vsconq5p0h2n
Content-Type: text/html

<html><head><title>Yahoo!</title><base href=http://www.yahoo.com/>
   <img width=230 height=33
     …etc…
```

CS 2-tiers: Pro & Cons (1)

- Benefits (towards a Monolithic Architecture):
  - Good security, because the users usually cannot see the database directly and can only access the data by starting the client
  - More scalable, because allows multiple users to access the database at the same time as long as they are accessing data in different parts of the database
  - Faster execution due to a shared workload

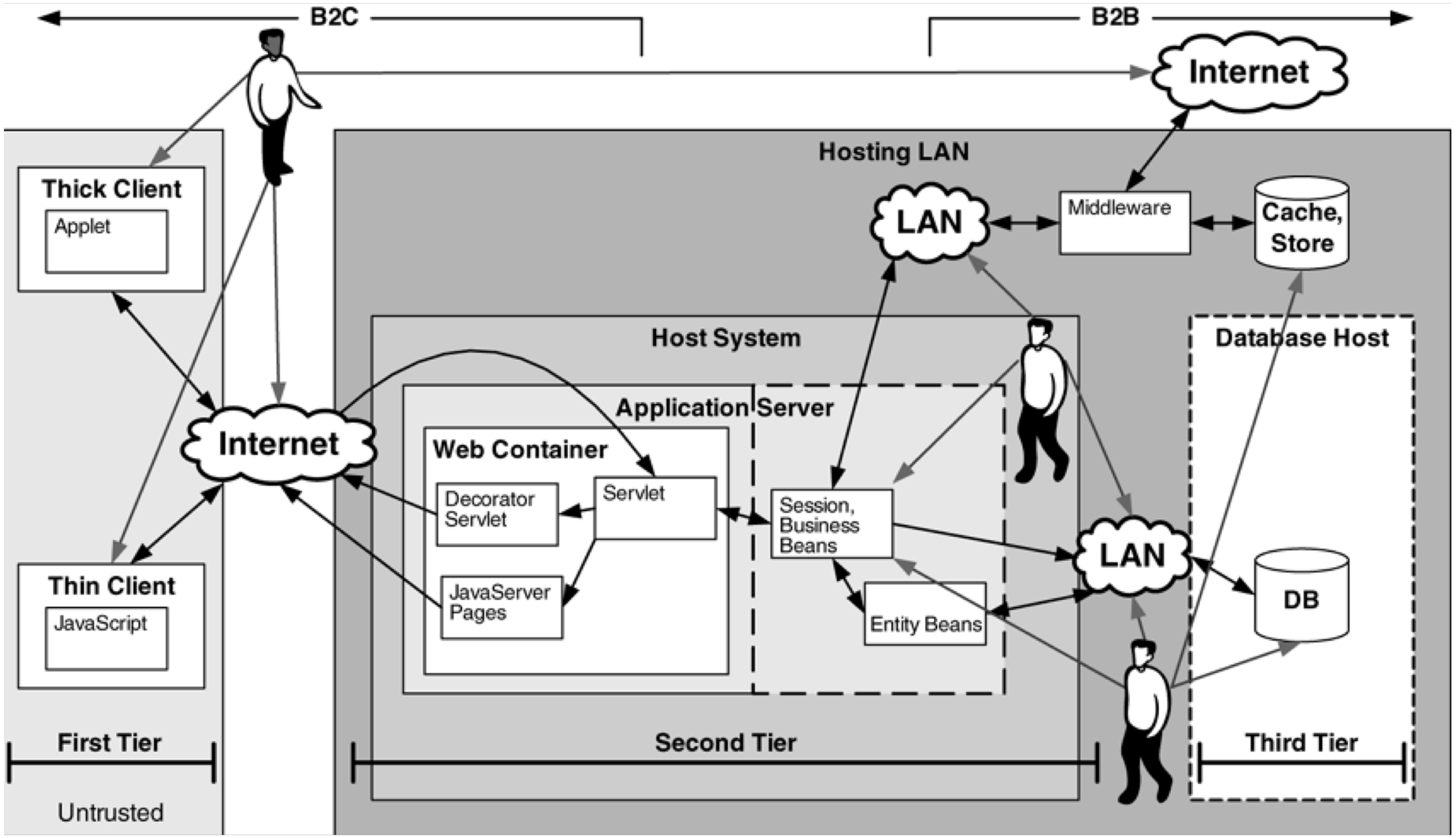
CS 2-tiers: Pro & Cons (2)

- Pitfalls:
  - Exhibits a heavy message traffic since the front-ends and the servers communicate intensively
  - The Application Logic is not managed by an *ad-hoc* component, but it’s “shared” by front-end and back-end
    - Client and server depend one from each other
    - It’s difficult to reuse the same interface to access different data
    - It’s difficult to interact with databases which have different front-ends
    - The business logic is encapsulated into the user interface, thus if the logic changes, the interface has to change as well

CS 3-tiers: Overview (1)

- In 3-tiers CS architecture, there is an intermediary tier, meaning the architecture is generally split up between:
  - A client, which requests the resources through a user interface (i.e. web browser) for presentation purposes
  - The application server, whose task it is to provide the requested resources, but by calling on another server
  - The data server, which provides the application server with the data it requires

Generic 3-tiers software architecture



CS 3-tiers: Overview (2)

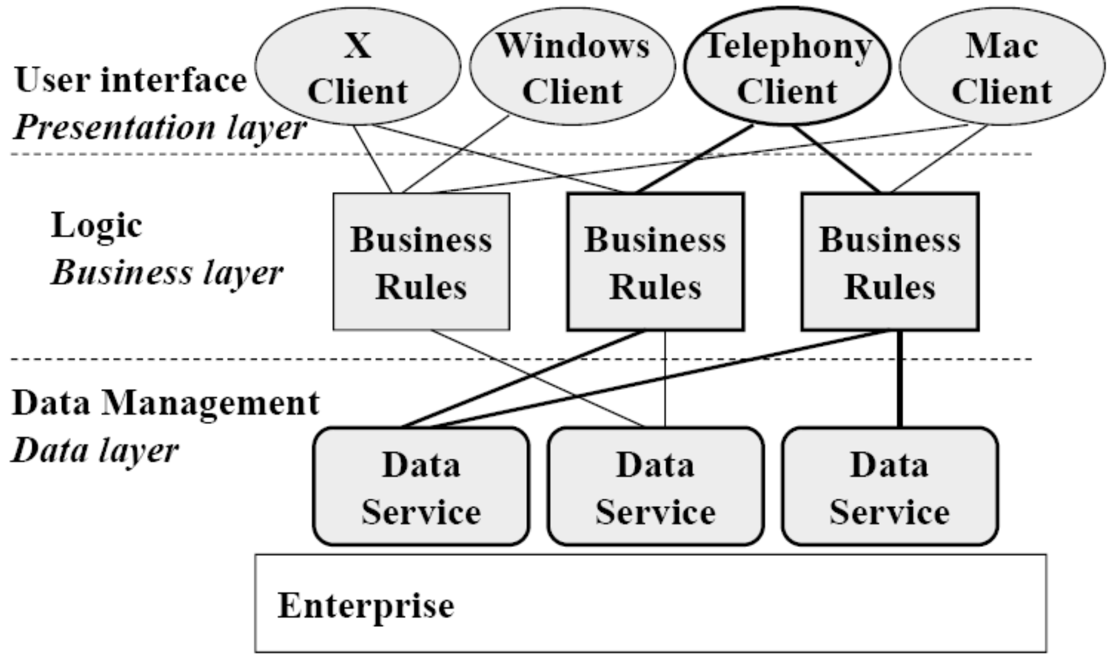
- Tiers work as they were not part of a single application:
  - Applications are conceived as collections of interacting components
  - Each component can take part to several applications at the same time
  - Each server component is specialised with a certain task:
    - web server, database server, etc…
- Tiers 1 and 3 do not communicate:
  - The user interface neither receives any data from data management, nor it can write data
  - Information passing (in both the directions) are filtered by the Application Logic

CS 3-tiers: Architecture (1)

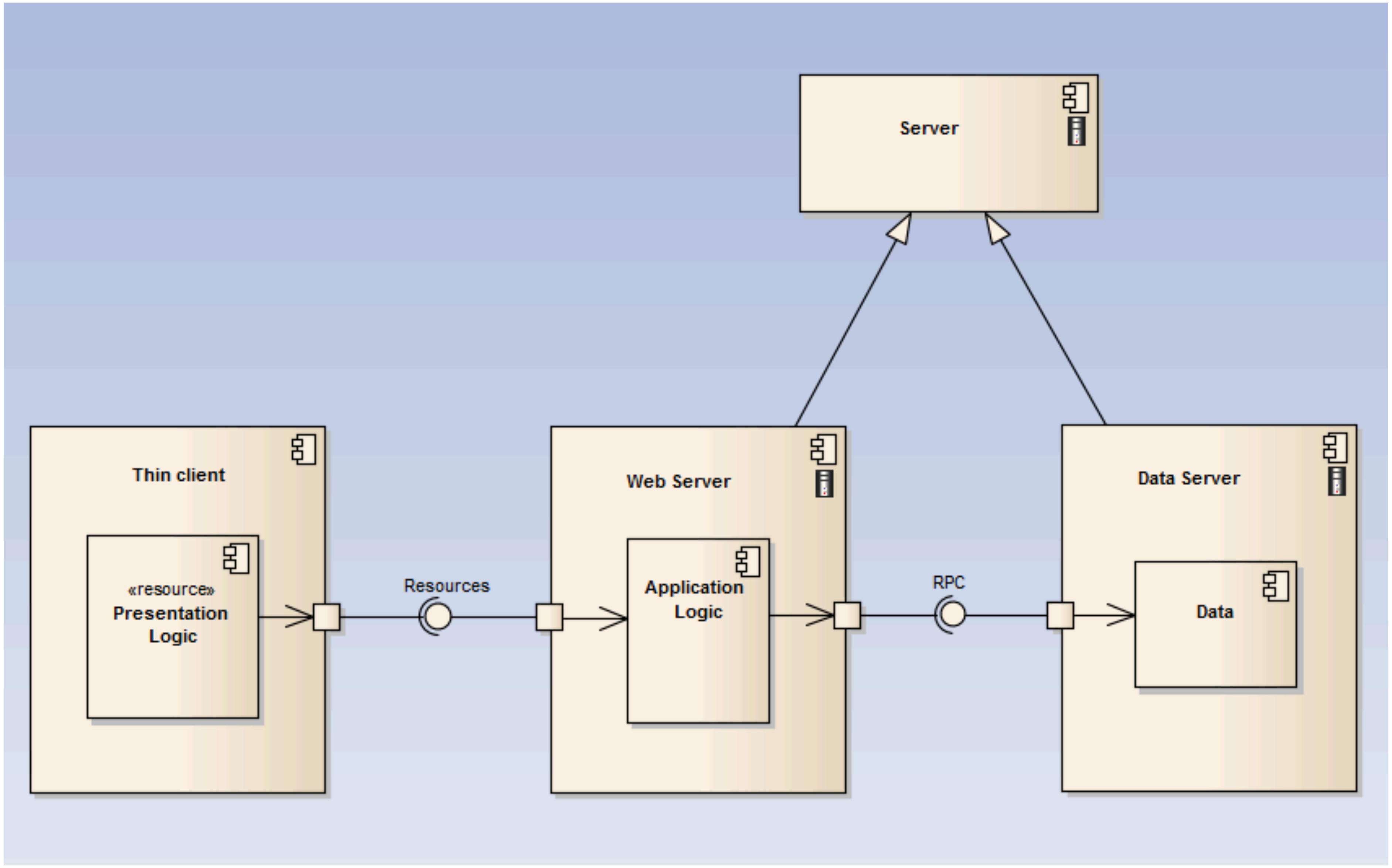
- Components:
  - **Client**
    - Active entity
    - “Thin” client containing only the Presentation Logic subsystem (session, text input, dialog, and display management services)
  - **Application Server**
    - Contains the Application Logic subsystem providing process management such as queuing, application execution, and database staging
  - **Data Server**
    - Passive entity
    - Contains the Database Management subsystem (such as data and file services)

CS 3-tiers: Architecture (2)

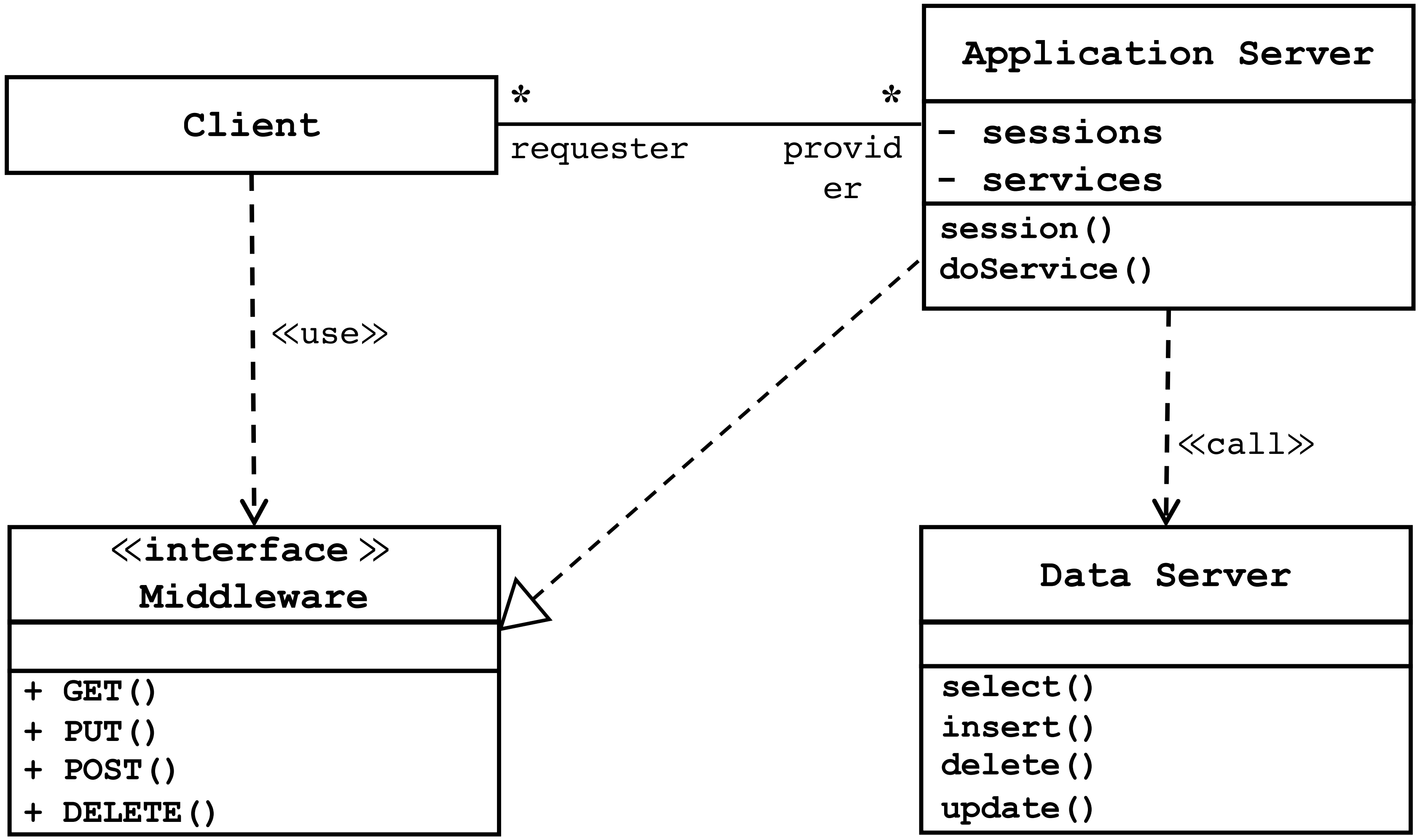
- Connectors:
  - **Remote Procedure Calls (RPC)**
    - Protocol: *The client calls for the business logic on the server, the business logic on the behalf of the client accesses the database*



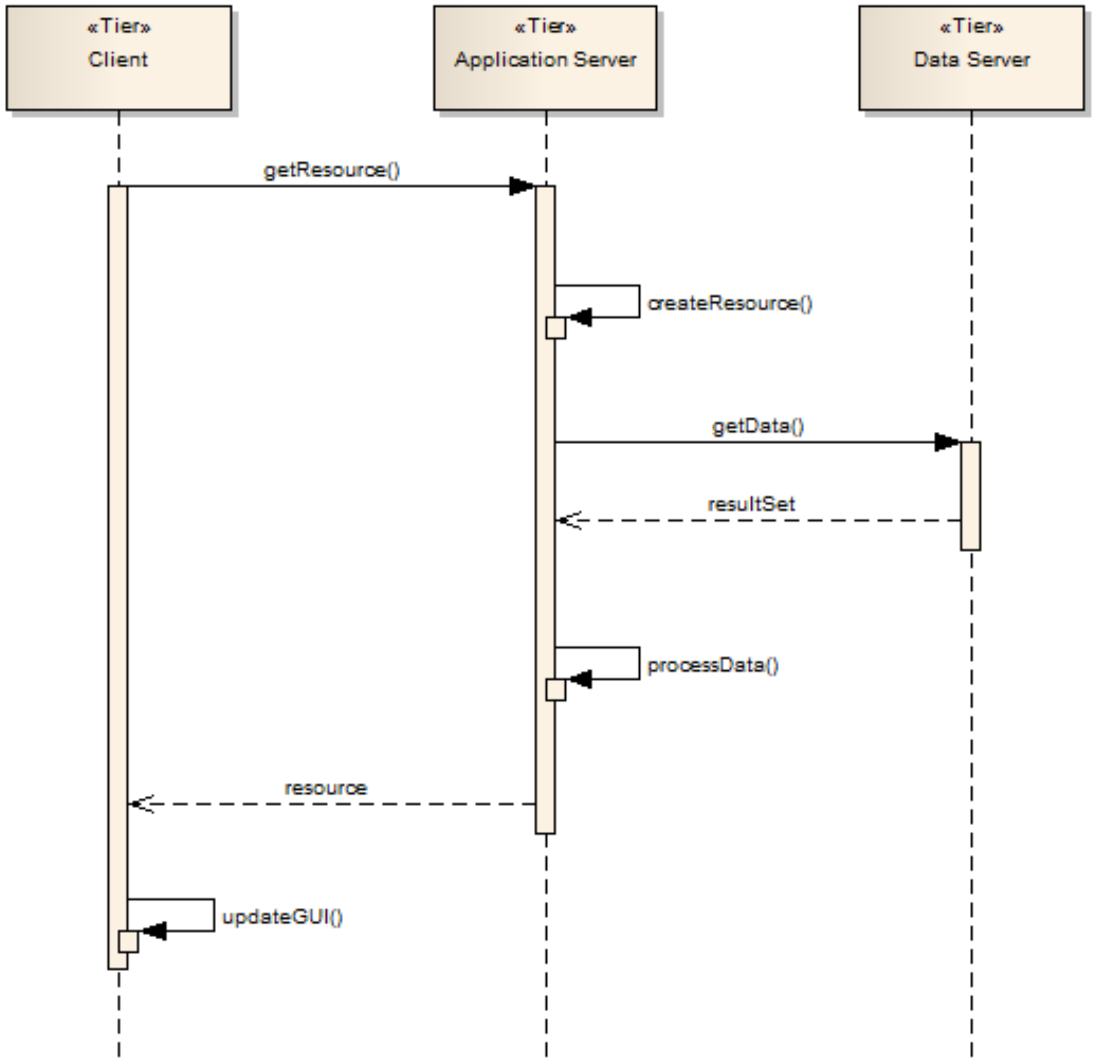
CS 3-tiers: Component Diagram



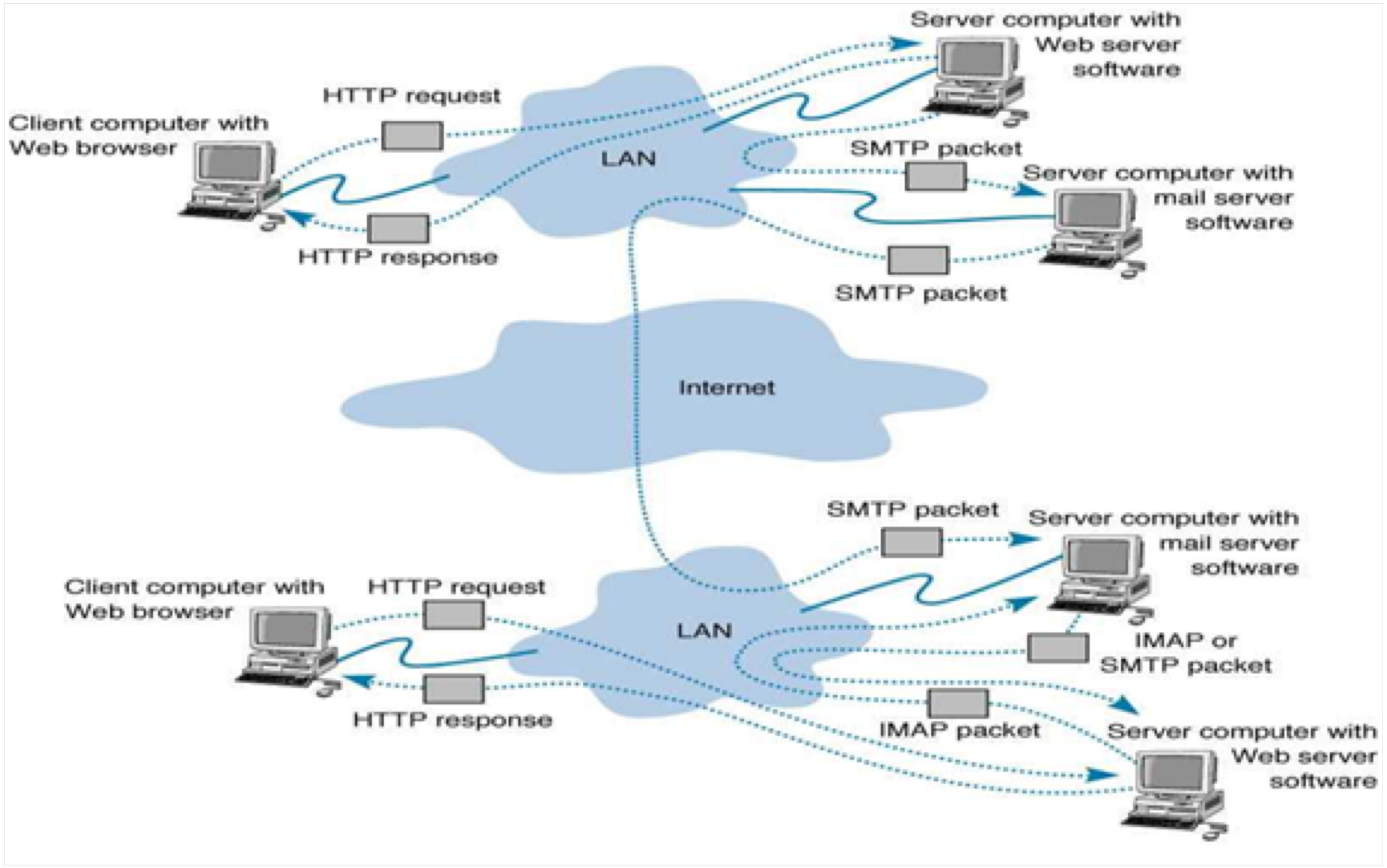
CS 3-tiers: Class Diagram



CS 3-tiers: Sequence Diagram

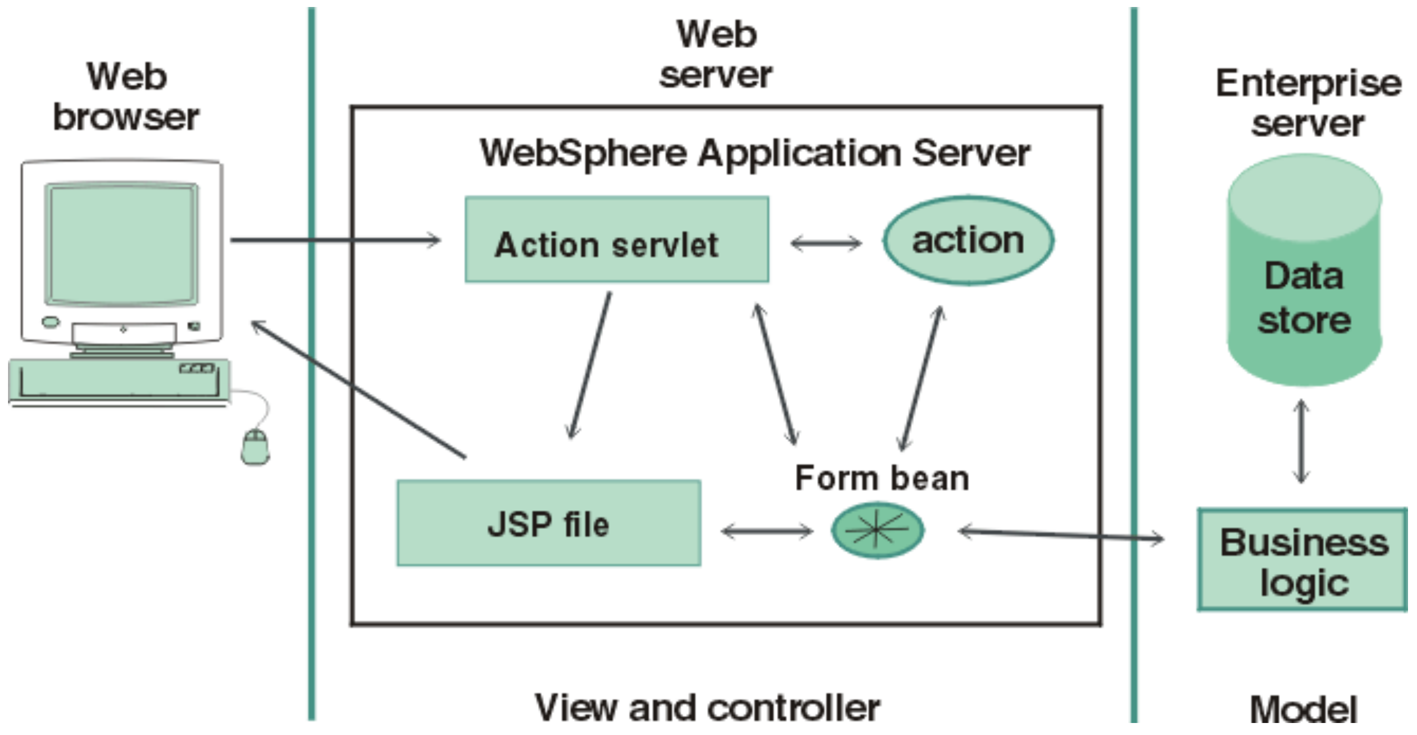


CS 3-tiers example: Web Mail



Server computer with Web server software
HTTP request
Client computer with Web browser
LAN
SMTP packet
Server computer with mail server software
HTTP response
SMTP packet
Internet
SMTP packet
Server computer with mail server software
Client computer with Web browser
HTTP request
LAN
IMAP or SMTP packet
HTTP response
IMAP packet
Server computer with Web server software

CS 3-tiers example: Struts framework



Copyright © IBM – Struts framework

- The browser sends a request to an Action servlet
- The Action servlet instantiates a Java bean that is connected to a database
- The Action servlet communicates with a JSP file
- The JSP file communicates with the Form bean
- The JSP file responds to the browser

CS 3-tiers: Pro & Cons (1)

- Benefits:
  - High flexibility and high modifiability:
    - Components can be used in several systems
    - New functionalities can be added to the system by only modifying the components which are in charge of realizing them, or by plugging new components
  - More scalability and performance because we can add as many middle tiers as needed
  - “Thin” client, because only little communication is needed between the client and the middleware which implements the Application Logic

CS 3-tiers: Pro & Cons (2)

- Pitfalls:
  - The additional tier increases the complexity and cost of the system:
    - Ad hoc software libraries have to be used to allow the communication among components
    - Heavy network traffic
  - Legacy software:
    - Many companies make use of preexisting (often monolithic) software systems to manage data
    - Adapters have to be implemented to interoperate with the legacy software

CS n-tiers: Overview (1)

- In the 3-tier applications, the middle tier is generally not a monolithic program but is implemented as a collection of components that are initiated by several client-initiated business transaction
- Thus, one component can call other components to help it implement a request
  - Generally, a server can use services from other servers in order to provide its own service

CS n-tiers: Overview (2)

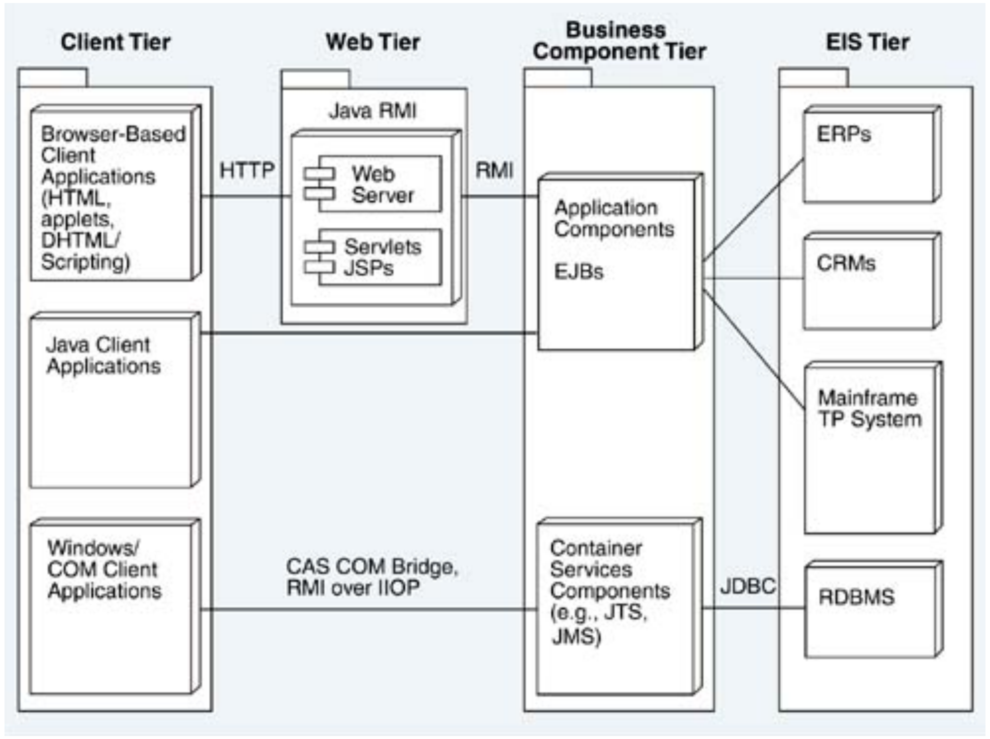
- Fundamental items:
  - User Interface (UI): a browser, a WAP mini browser, a graphical user interface (GUI)
  - Presentation logic, which defines what the UI has to show and how to manage users' requests
  - Business logic, which manages the application business rules
  - Infrastructure services:
    - They provides further functionalities to the application components (messaging, transactions support)
  - Data tier:
    - Application Data level

CS n-tiers: Architecture (1)

- Typical components:
  - **Client**
    - Same as 3-tiers Architecture
  - **Web Server**
    - Session management
    - Content creation, format and delivery
  - **Application Server**
    - Data access objects
    - Transactions
    - Business logic
    - Resources adapters
  - **Data Server**
    - Same as 3-tiers Architecture

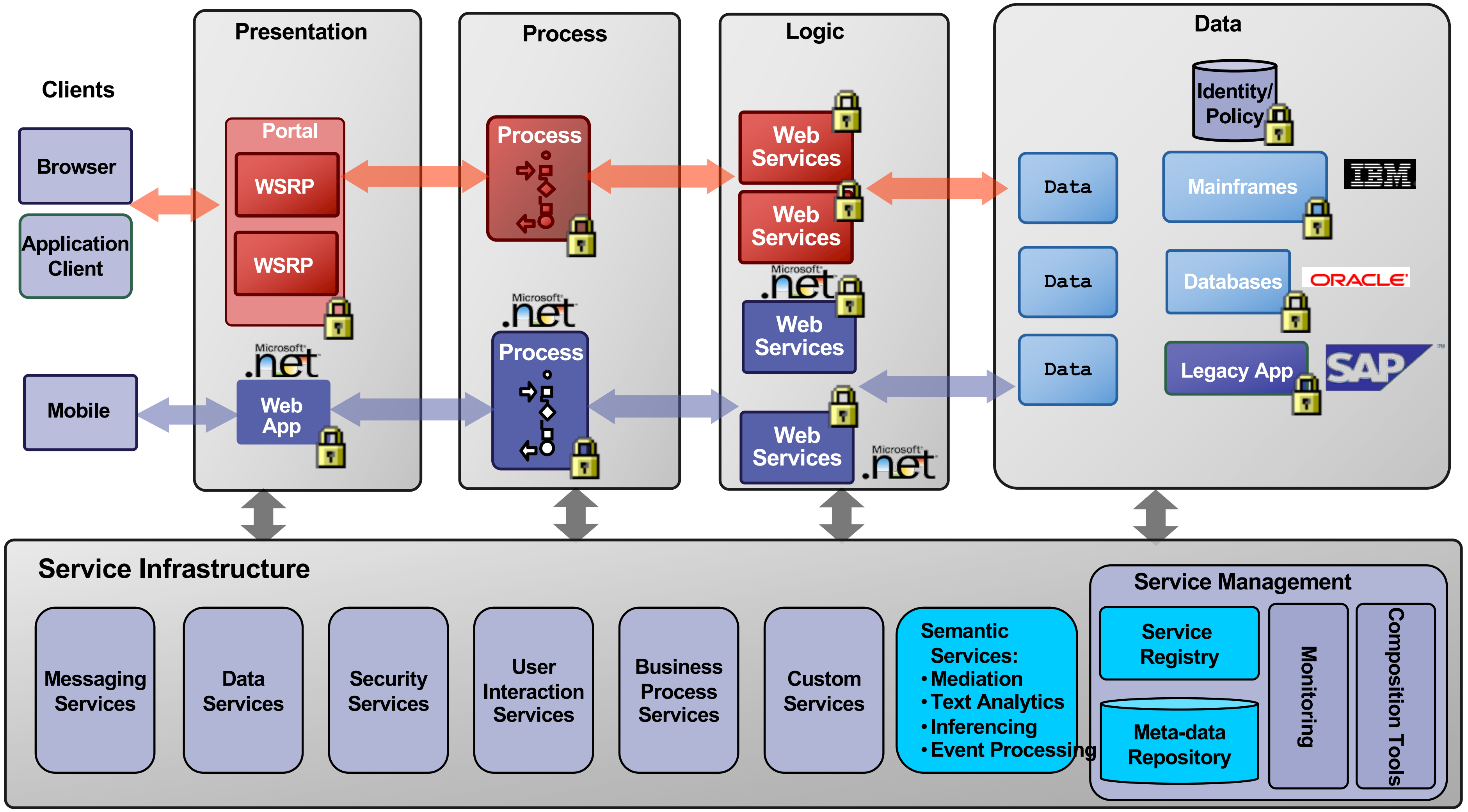
CS n-tiers: Architecture (2)

- Connectors:
  - **Remote Procedure Calls (RPC)**
    - Many protocols

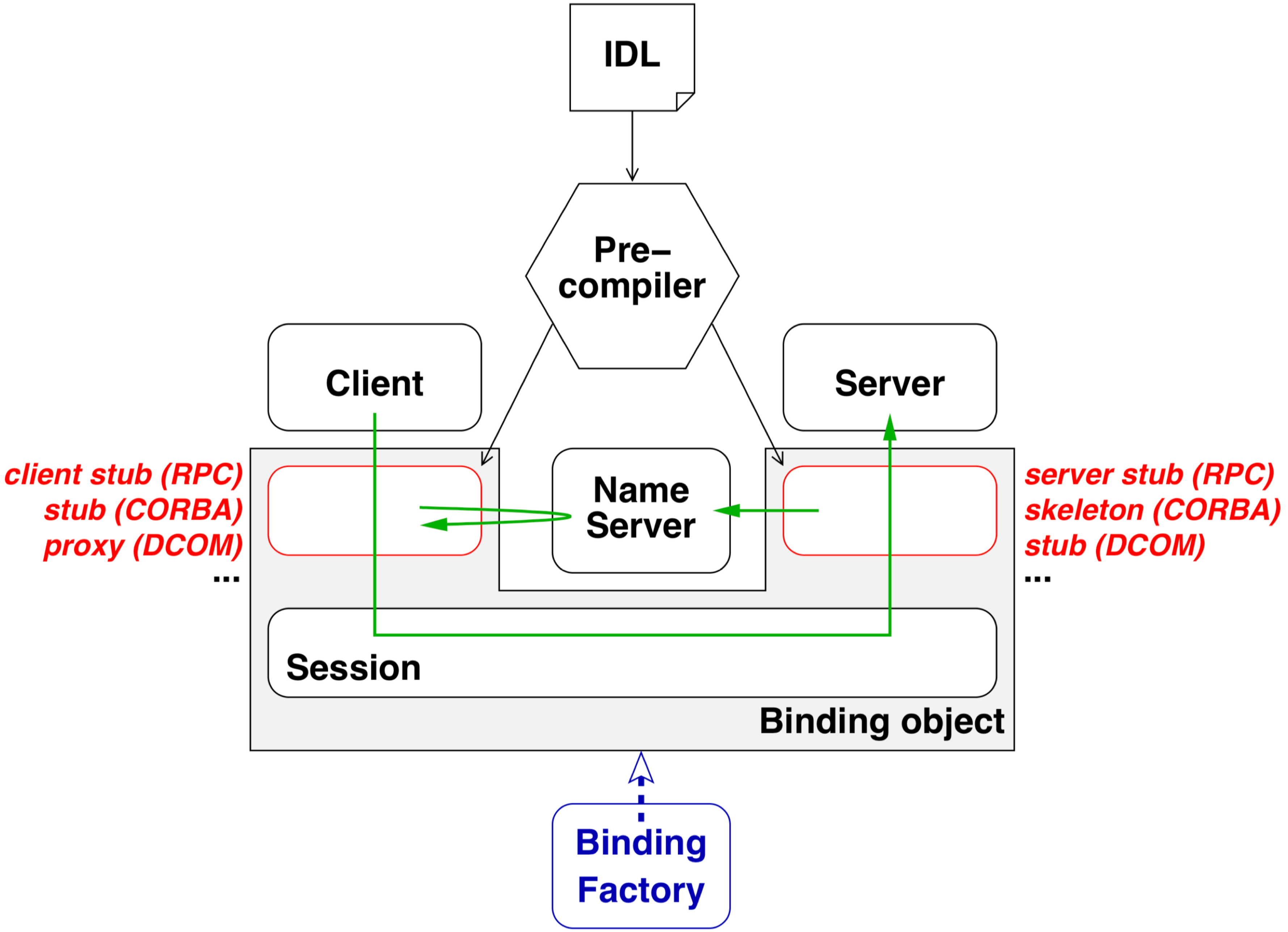


Client Tier
Web Tier
Business Component Tier
EIS Tier
Browser-Based Client Applications (HTML, applets, DHTML/ Scripting)
HTTP
Java RMI
Web Server
Servlets JSPs
RMI
Application Components
EJBs
ERPs
CRMs
Mainframe TP System
Java Client Applications
Windows/ COM Client Applications
CAS COM Bridge, RMI over IIOP
Container Services Components (e.g., JTS, JMS)
JDBC
RDBMS

Example: 5 tiers



Client server middleware



References

- **Fowler et al.,** *Patterns of Enterprise Application Architecture*, AW 2002
- **Clemens et al.,** *Documenting Software Architectures*, Addison Wesley, 2010

Questions?