
Arsitektur Aplikasi Perangkat Enterprise #2

Oleh: Antonius Rachmat C, S.Kom, M.Cs

Enterprise Application Outline

- Evolusi enterprise application
 - Design of an enterprise application
 - Bottom up design
 - Top down design
 - Architecture of an enterprise application
 - One tier
 - Two tier (client/server)
 - Three tier (middleware)
 - N-tier architectures
 - Middleware: RPC, TP-Monitor, CORBA, MOM
 - Communication in an enterprise application
 - Blocking or synchronous interactions
 - Non-blocking or asynchronous interactions
-

Evolusi Enterprise Application

- Dahulu sistem bersifat “**Centralized Approach**”.
Yaitu sistem bersifat stand alone dan terpusat.
 - Single system for all processing needs
 - Physical limitations of scalability, single points of failure, dan limited accessibility from remote locationsBersifat **single-tier**: presentasi, logic business, code, dan data menjadi satu kesatuan, tidak dipisah-pisah.
 - Kekurangan **single-tier**:
 - Menyebabkan perubahan terhadap salah satu komponen diatas tidak mungkin dilakukan, karena akan mengubah semua bagian.
 - Tidak memungkinkan adanya **re-usable** component dan code.
-

Evolusi Enterprise Application

- Sekarang sistem bersifat “**Distributed Approach**”
 - Sistem bersifat tersebar dan multiproses.
 - Sistem ini bersifat **On Demand Software** dan **Software as Service**
 - Bersifat **multi-tier**:
 - presentasi, logic business, dan data terpisah-pisah menjadi lapisan-lapisan tersendiri.
-

Layering

- Layering salah satu teknik umum di mana para software designer menggunakan hal itu untuk **memecah** sebuah sistem yang rumit ke dalam bagian-bagian yang lebih sederhana.
 - Contoh pada networking: lapisan layer OSI dan TCP/IP.
 - Ketika sistem dibagi dalam layer-layer:
 - bagian sistem yang principal dalam software diatur dalam layer
 - setiap **upper layer bergantung** pada **lower layer**.
-

Layering

- **Higher layer** menggunakan service-service yang didefinisikan oleh **lower layer**
 - lower layer tidak perlu mengetahui the higher layer.
 - Setiap layer biasanya **menyembunyikan** lower layer-nya dari layer atasnya
 - Ex: layer 4 menggunakan services dari layer 3,
 - Layer 3 menggunakan services dari layer 2,
 - Layer 4 tidak tahu menahu tentang layer 2.
-

Kelebihan Layering

- User mengetahui aplikasi tersebut terdiri dari satu **single layer** saja **tanpa harus tahu** layer-layer yang lain.
 - Kita dapat memanfaatkan FTP service pada TCP tanpa harus tahu bagaimana cara kerja Ethernet Card secara fisik.
 - Kita **dapat mengganti** layer-layer dengan aplikasi lain yang mengimplementasikan **servis dasar** yang sama.
 - Dapat dibuat berbagai FTP software yang berjalan tanpa harus mengganti Ethernet, atau kabel-kabel.
 - Kita dapat **meminimalisasi ketergantungan** antar layer-layer.
 - Jika kita mengganti kabel jaringan, kita tidak perlu juga mengganti FTP service.
-

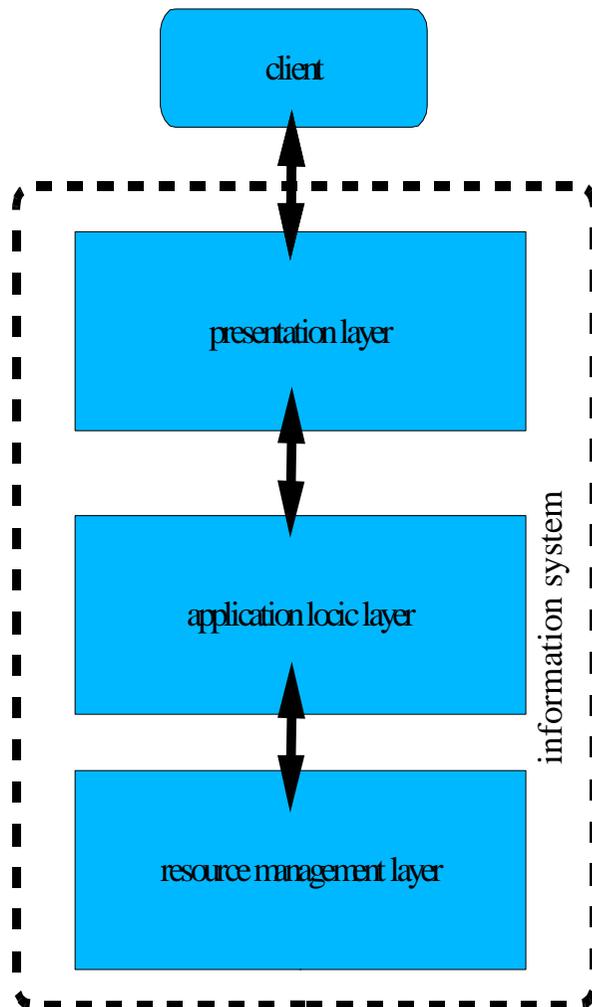
Kelebihan Layering

- Layer sangat mendukung **standarisasi**.
 - TCP / IP = standar
 - Sesudah layer terbentuk, kita dapat menggunakannya untuk **bermacam-macam servis** lainnya.
 - Contoh, TCP/IP digunakan oleh FTP, telnet, SSH, dan HTTP.
-

Kelemahan Layering

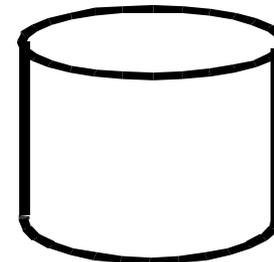
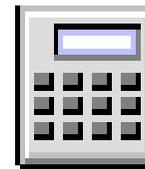
- Penggunaan layer menyebabkan dan menambah tingkat **kompleksitas** proses.
 - Setiap layer harus memiliki fungsinya masing-masing
 - Suatu proses harus melewati masing-masing layer tersebut terlebih dahulu baru dapat menghasilkan output.
 - Jadi masing-masing layer harus **memiliki kemampuan proses yang berlainan**.
 - Layer **mengkapsulasi** fungsi-fungsinya masing-masing sehingga kita tidak dapat mengetahui **detail** fungsi suatu layer.
 - Layer bekerja secara bersama-sama menjadi satu kesatuan sehingga seluruh layer harus bekerja secara **optimal**.
-

Layers of an IS Example



application

Interface client



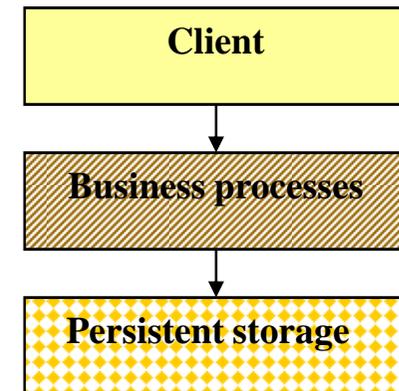
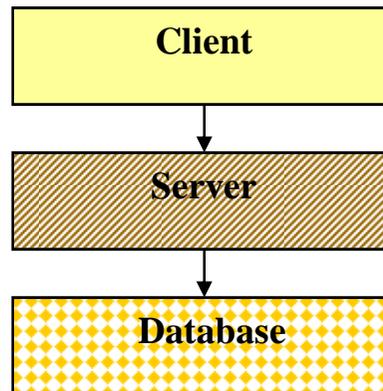
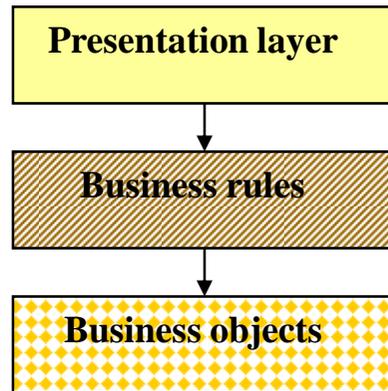
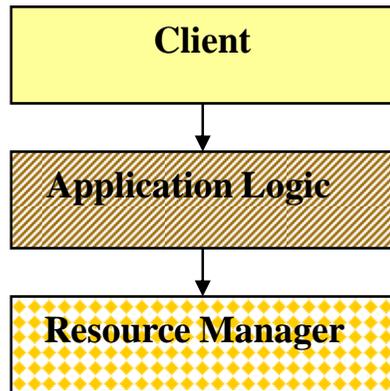
3 Principal Layers

- **Presentation logic:** mengatur bagaimana handle interaksi antara user dan software.
 - Bisa berupa simple command-line atau text-based menu system, tapi sekarang bisa berupa rich-client graphics UI atau HTML-based browser UI.
 - Presentation layer = menampilkan informasi ke user
 - Menginterpretasikan perintah dari user sebagai aksi terhadap business logic dan data source.
-

3 Principal Layers

- **Data source logic:** mengatur komunikasi dengan sistem lain dan manajemen data.
 - Bisa berupa transaction monitor dan database.
 - Ex: database / xml / text
 - **Domain logic / business logic.** mengatur tindakan aturan bisnis (aturan main) suatu aplikasi.
 - Ex: melakukan kalkulasi berdasarkan input dan data yang tersimpan,
 - validasi dari data yang datang dari layer presentasi,
 - menggambarkan secara tepat mana data source logic yang dibutuhkan, tergantung dari perintah yang diterima dari layer presentasi.
-

Layers and tiers



Layers and 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.
-

Designs of Distributed IS

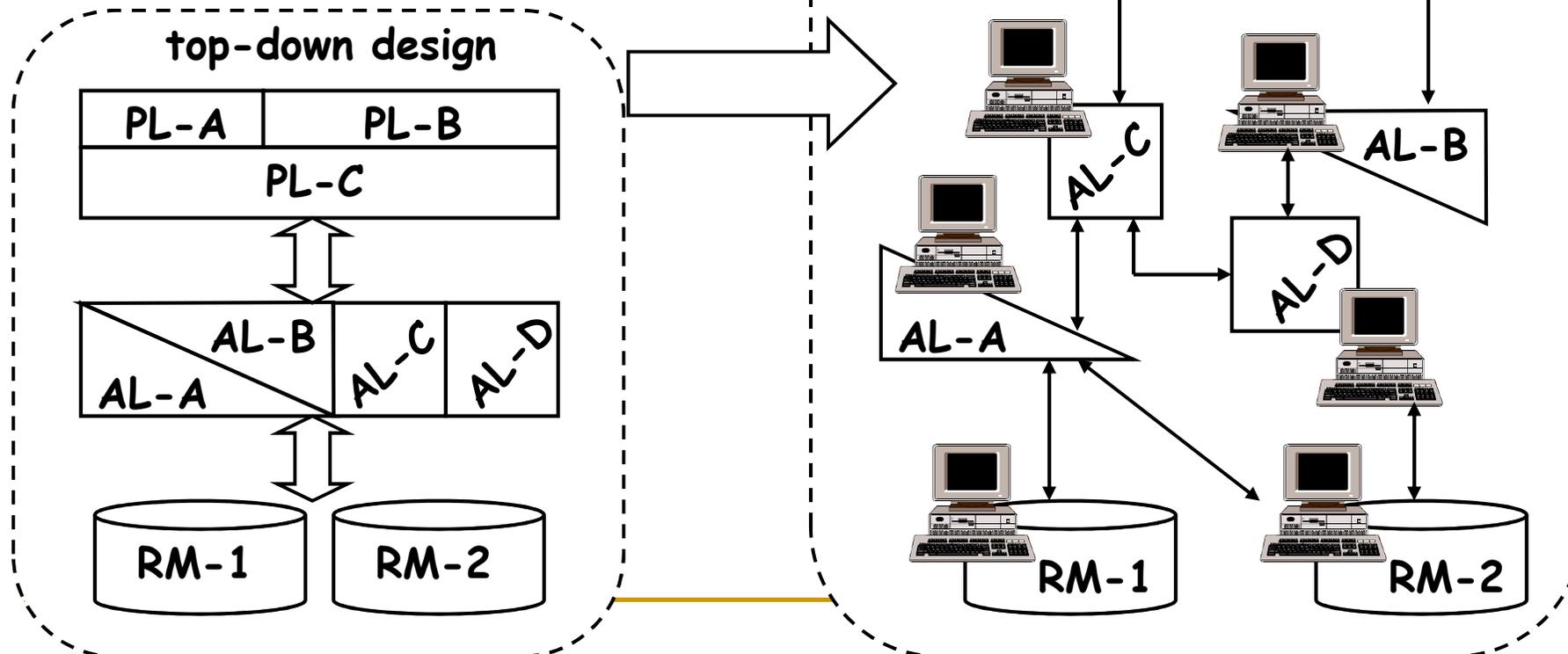
- ***top-down design***
 - ***bottom-up design***
-

top-down design

- starts with defining **functionality** desired by the client ('toplevel goals')
 - **implementation** of application logic
 - defining the **resources** needed by application logic
-

- The functionality of a system is **divided** among **several** modules.
- Modules **cannot act as a separate component**, their functionality **depends** on the functionality of other modules.
- Hardware is typically **homogeneous** and the system is designed to be distributed from the **beginning**.

Top down design



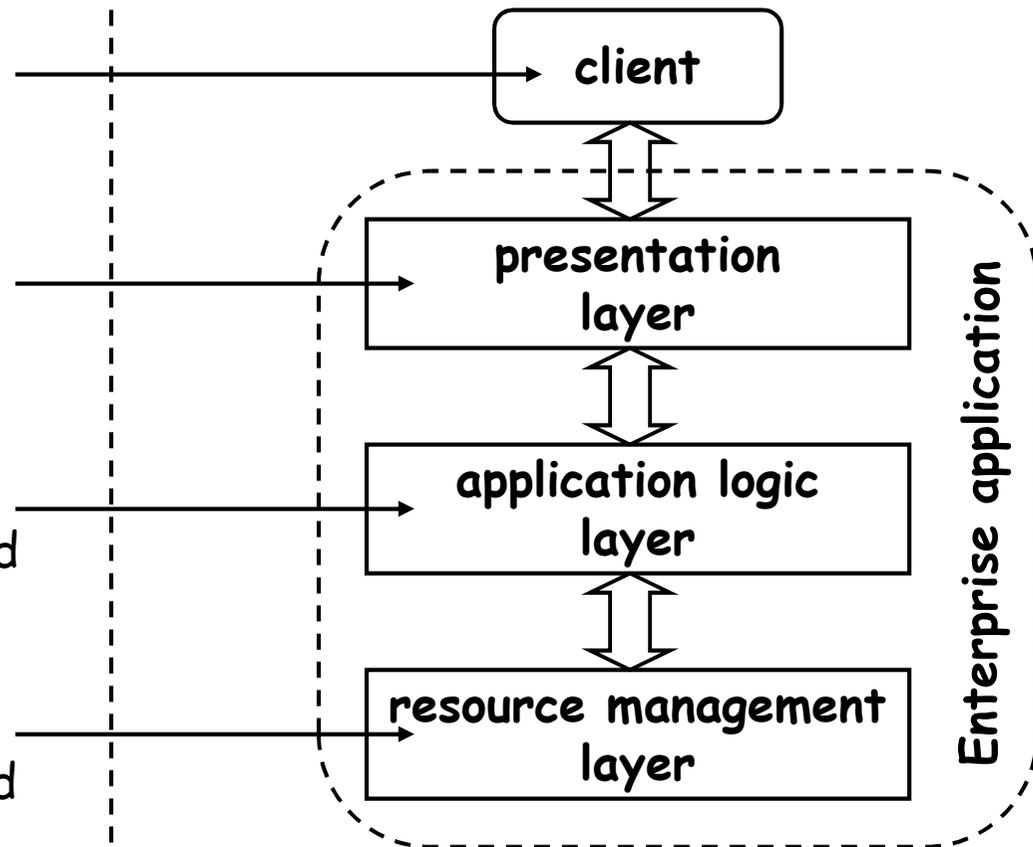
top-down design

- usually created to run in **homogenous** environments
 - results in **tightly coupled** components:
 - functionality of each component heavily **depends** on functionality of other components
 - *design is sometimes component based, but components are not standalone*
-

Top down design

top-down design

1. define **access channels** and client platforms
2. define **presentation formats and protocols** for the selected clients and protocols
3. define the **functionality** necessary to deliver the contents and formats needed at the presentation layer
4. define the **data sources** and data organization needed to implement the application logic



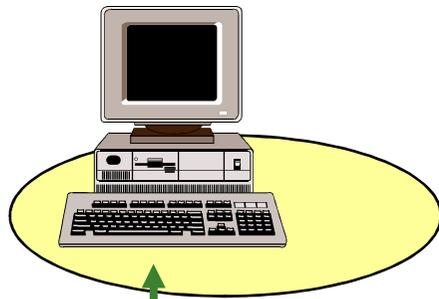
advantages & disadvantages of top down

- advantages:
 - design emphasises **final goals** of the system
- disadvantages
 - can only be designed from **scratch**
 - legacy systems **cannot be integrated**

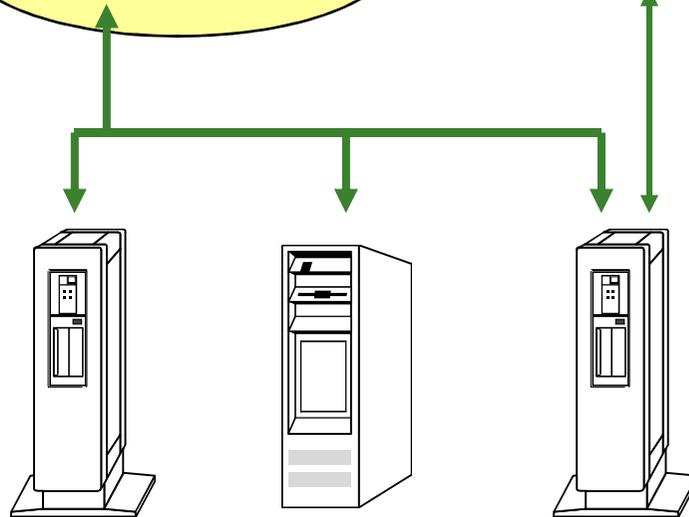
today **few ISs** are designed purely top-down

Bottom up design

New application



Legacy application



Legacy systems

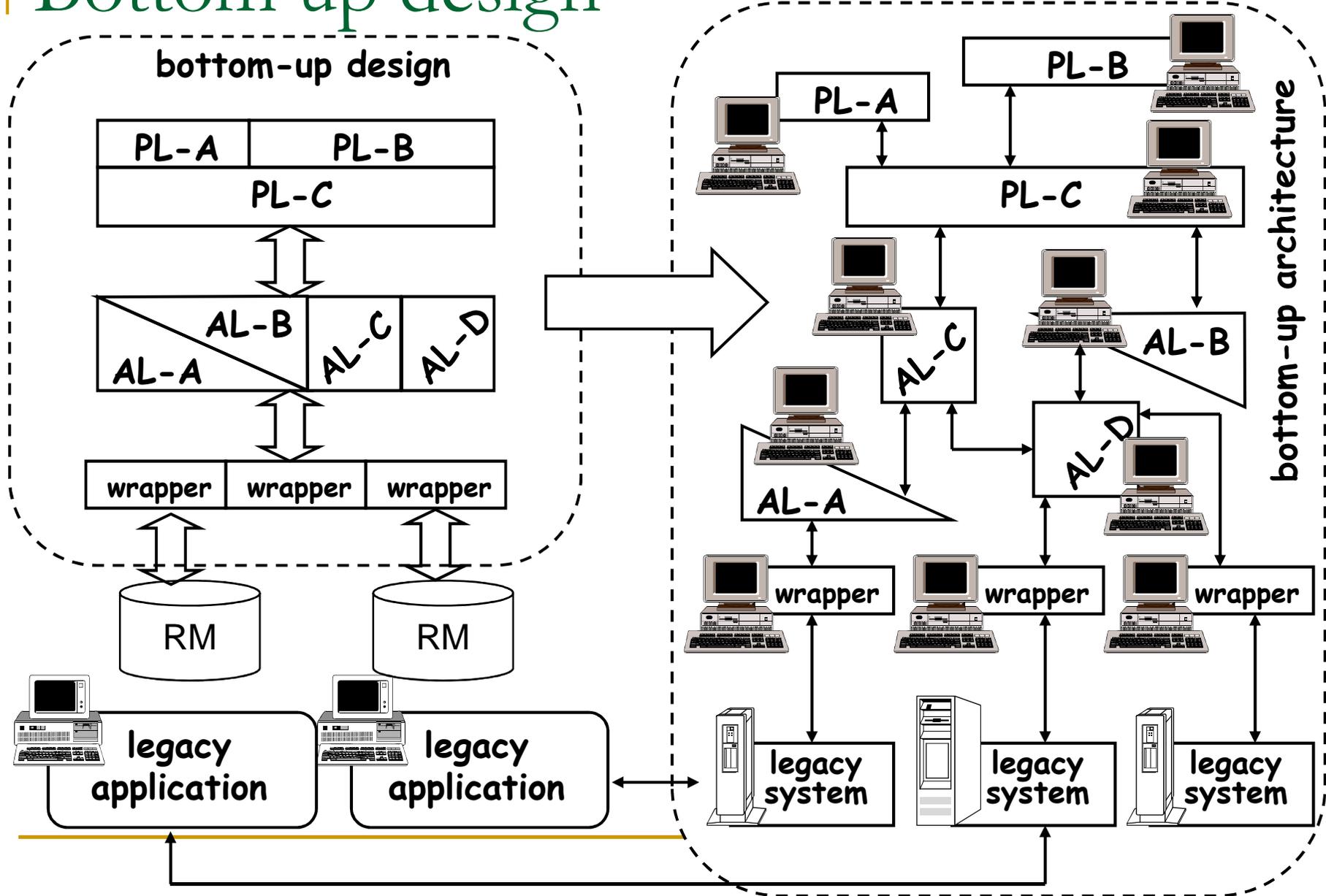
Bottom up design

- In a bottom up design, many of the basic components **already exist**.
 - These are stand alone systems which **need to be integrated** into new systems.
 - The components **do not** necessarily **ease** to work as stand alone components.
 - Often old applications **continue running** at the same time as new applications.
-

Bottom up design

- This approach has a **wide application** because the underlying systems already **exist** and **cannot be easily replaced**.
 - Much of the work and products in this area are **related to middleware**
 - **Middleware:** the intermediate layer used to *provide a common interface, bridge heterogeneity, and cope with distribution.*
 - **Web services** can make those designs more efficient, cost-effective and simpler to design
-

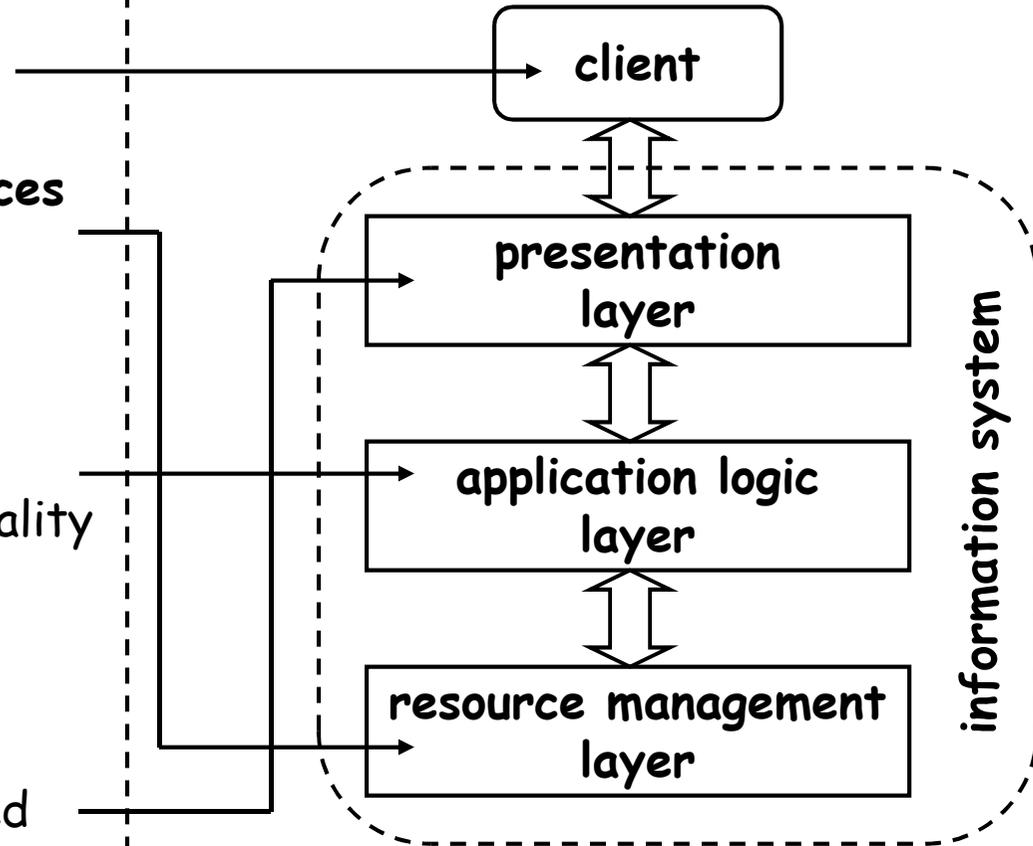
Bottom up design



Bottom up design

bottom-up design

1. define **access channels** and client platforms
2. examine **existing resources** and the **functionality** they offer
3. **wrap existing** resources and **integrate** their functionality into a consistent interface
4. **adapt** the output of the application logic so that it can be used with the required access channels and client protocols



Architecture of an Information

System - 4 types:

- **1 – tier**

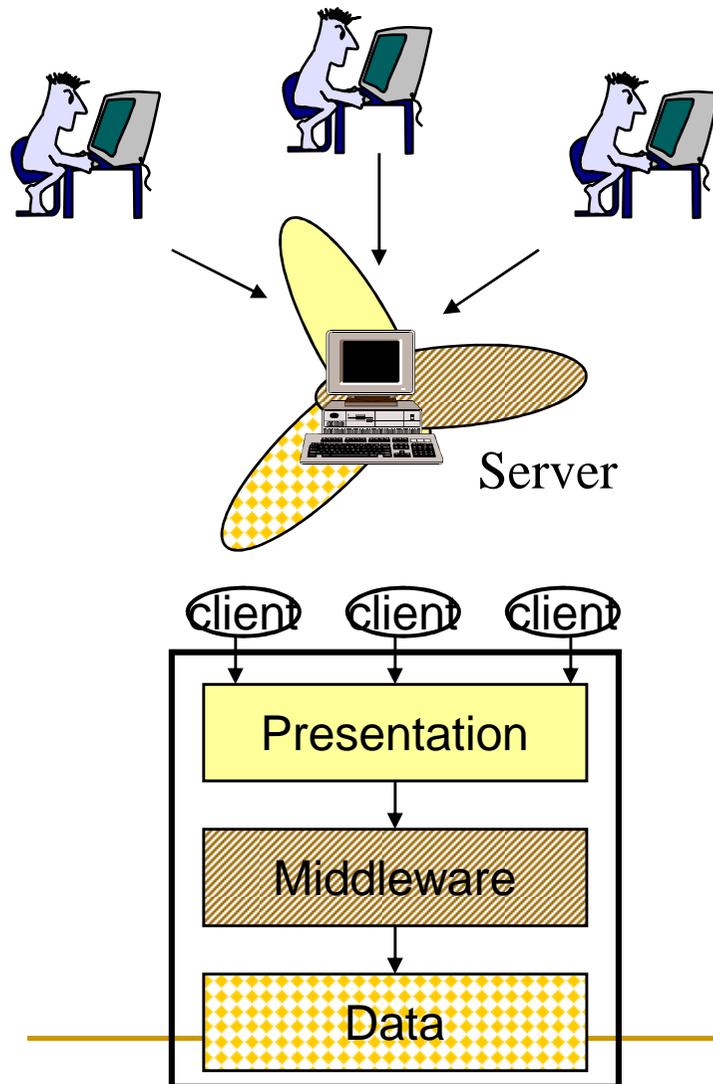
- **2 – tier**

- **3 – tier**

- ***n* – tier**

One tier: fully centralized

1-tier architecture



- The presentation layer, application logic and resource manager are built as a **monolithic** entity.
- Users/programs access the system through display **terminals** but what is displayed and how it appears is controlled by the server.
= “**dumb**” terminals
- This was the typical architecture of **mainframes**

1 – tier Architecture

advantages:

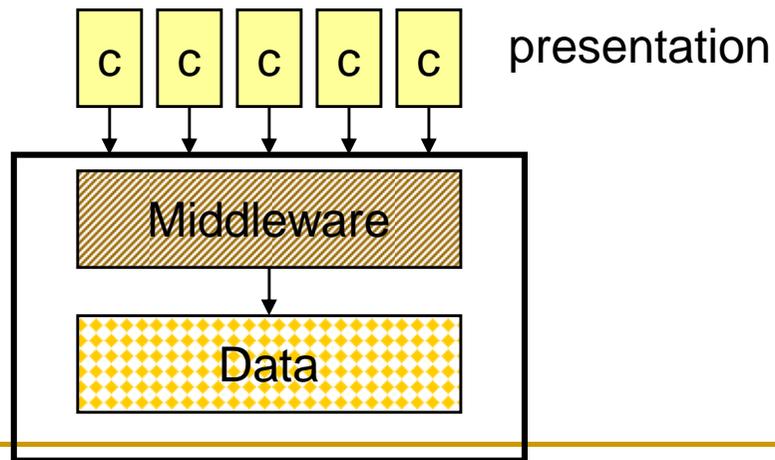
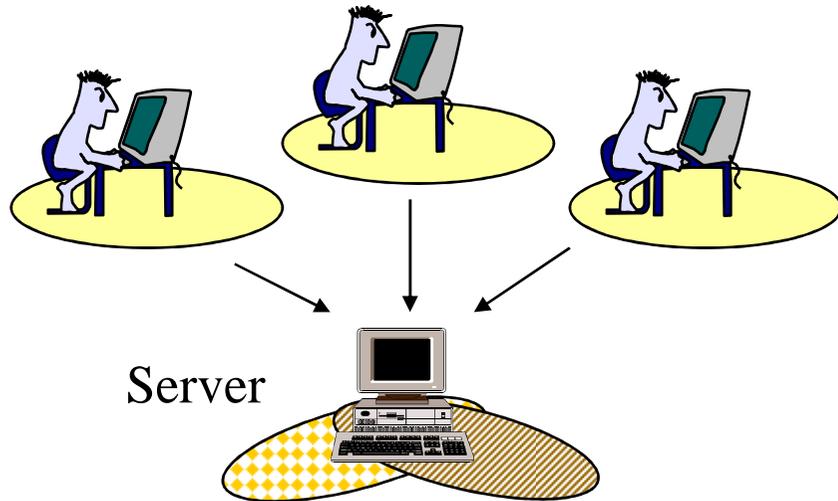
- easy to optimize performance
- no context switching
- no compatibility issues
- no client development, maintenance and deployment cost

disadvantages:

- monolithic pieces of code (high maintenance)
 - hard to modify
 - lack of qualified programmers for these systems
-

Two tier: client/server

2-tier architecture



Two Tier Architecture Advantages

- As computers became more powerful, it was possible to **move the presentation layer to the client**. This has several advantages:
 - Clients are **independent** of each other
 - One can take advantage of the **computing power at the client machine** to have more sophisticated presentation layers. (“sophisticated client”)
 - It introduces the concept of **API (Application Program Interface)**
 - The resource manager only sees **one client**: the application logic.
 - This greatly helps with performance since there are no client connections/sessions to maintain.
-

Disadvantages of Two Tier

- The server has to deal with **all possible** client connections.
 - There are **maximum number of clients**
 - Clients are “**tied**” to the system since there **is no standard presentation layer**.
 - If one wants to connect to two systems, then the *client needs two presentation layers*.
 - If the server **fails**, nobody can work.
 - All clients are all **competing** for the **same** resources.
-

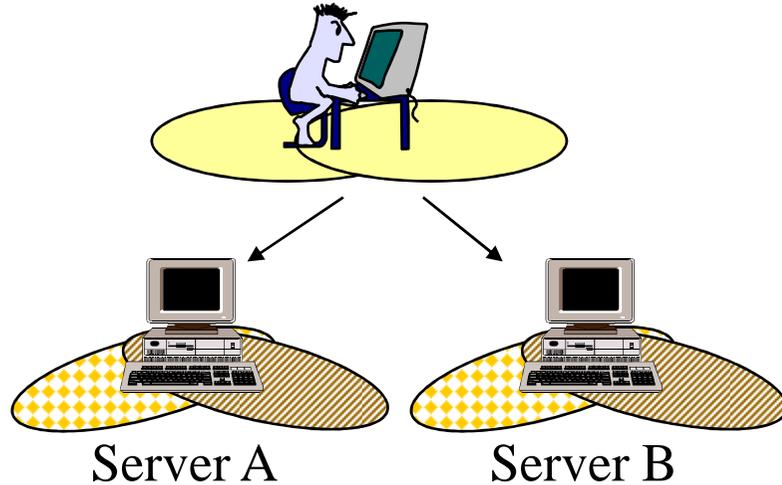
Karakteristik Client/Server

- **Service** : menyediakan layanan terpisah yang berbeda.
 - **Shared resource** : server dapat melayani beberapa client pada saat yang sama dan mengatur pengaksesan resource
 - **Asymmetrical Protocol** : antara client dan server merupakan hubungan one-to-many.
 - **Transparency Location** : proses server dapat ditempatkan pada mesin yang sama atau terpisah dengan proses client.
 - Client/server akan menyembunyikan lokasi server dari client.
-

Karakteristik Client/Server

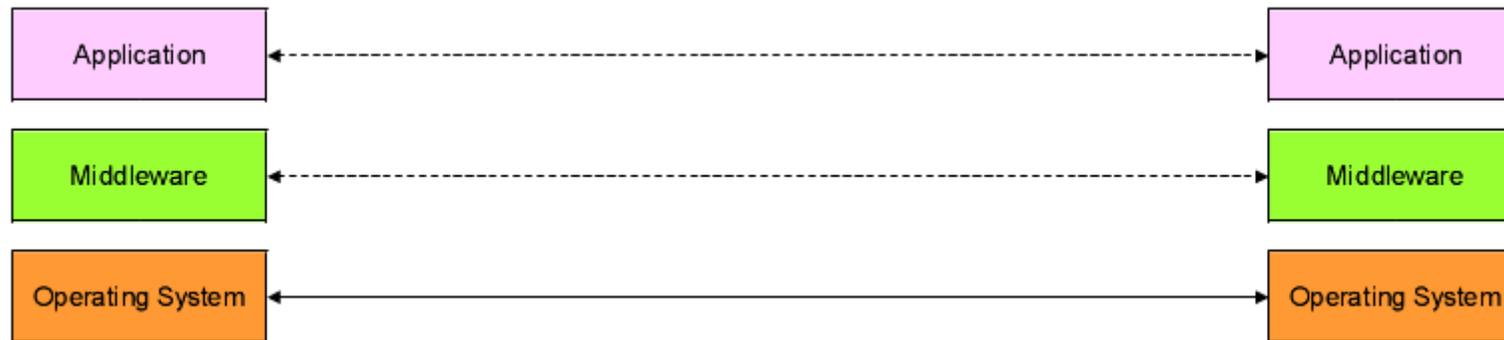
- **Mix-and-match** : tidak tergantung pada platform
 - **Message-based-exchange** : antara client dan server berkomunikasi dengan mekanisme pertukaran message.
 - **Encapsulation of service** : message dari client memberitahu server apa yang akan dikerjakan tanpa harus tahu detail service.
 - **Integrity** : kode dan data server diatur secara terpusat, sedangkan pada client tetap pada komputer tersendiri.
-

The main limitation of client/server



- the client **is the point of integration** (increasingly **fat clients**)
 - The responsibility of dealing with heterogeneous systems is **shifted to the client**.
 - The client **becomes responsible** for knowing where things are, how to get to them, and how to ensure consistency
- the underlying systems **don't know about each other**
 - Maybe there is **no common business logic**

Middleware (Layer perantara)



Software yang berfungsi sebagai **lapisan konversi** atau **penerjemah** diantara komponen aplikasi dengan tujuan untuk mengurangi kompleksitas pada aplikasi terdistribusi.

Contoh Arsitektur yang menggunakan Middleware:
Client/Server

Middleware as Programming abstractions

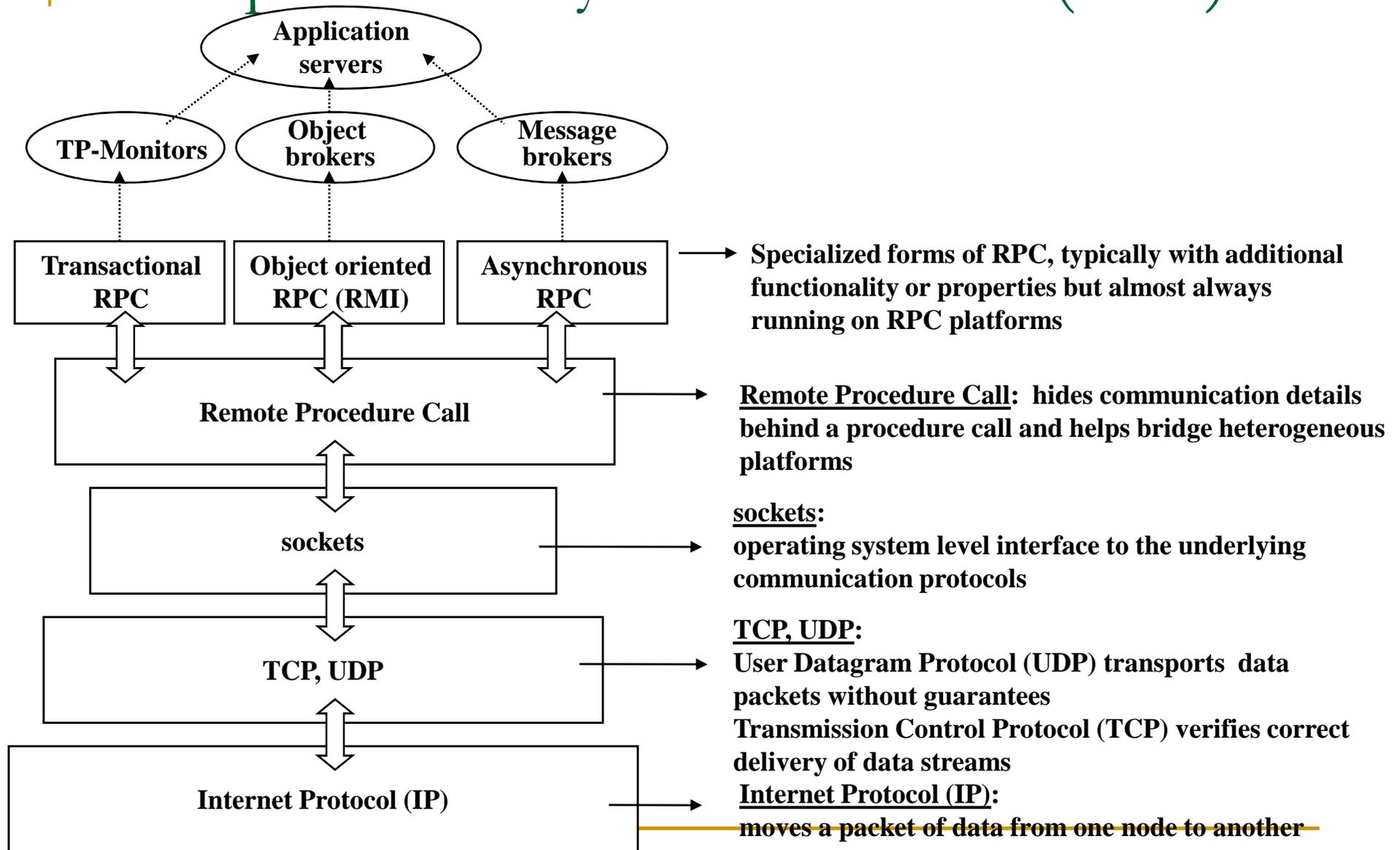
- **Abstraction** is a key concept in making software development easier for software developers
 - programming with abstractions can:
 - hide hardware/platform details
 - provide powerful building blocks
 - reduce programming errors
 - reduce development and maintenance costs
-

Middleware as Programming

abstractions

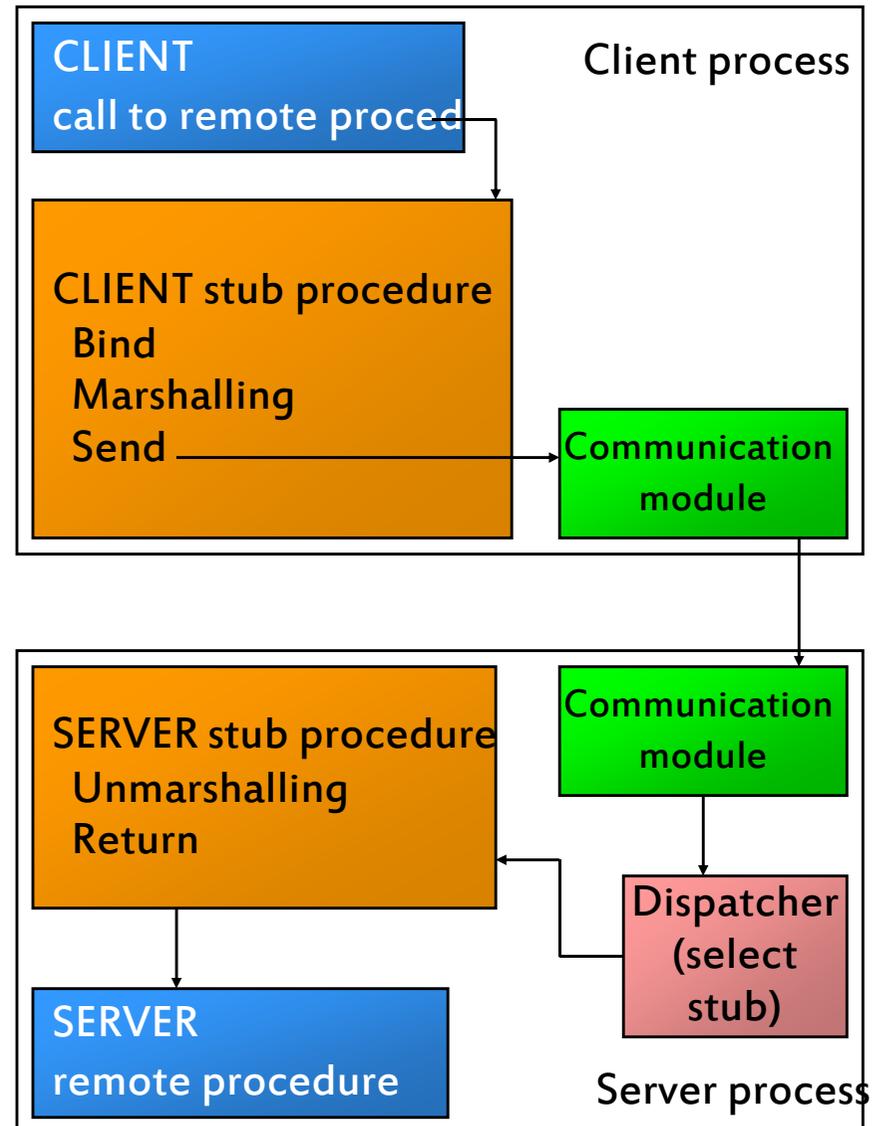
- Middleware can be seen **as a set of programming abstractions** that make it easier to develop complex distributed systems
 - Example of middleware:
 - remote communication mechanisms (Web services, CORBA, Java RMI, DCOM)
 - event notification and messaging services (Java Messaging Service etc.)
 - transaction services (TP Monitor)
 - naming services (Naming, LDAP)
 - Database connectivity (JDBC, ODBC)
-

Example hierarchy of middleware (RPC)

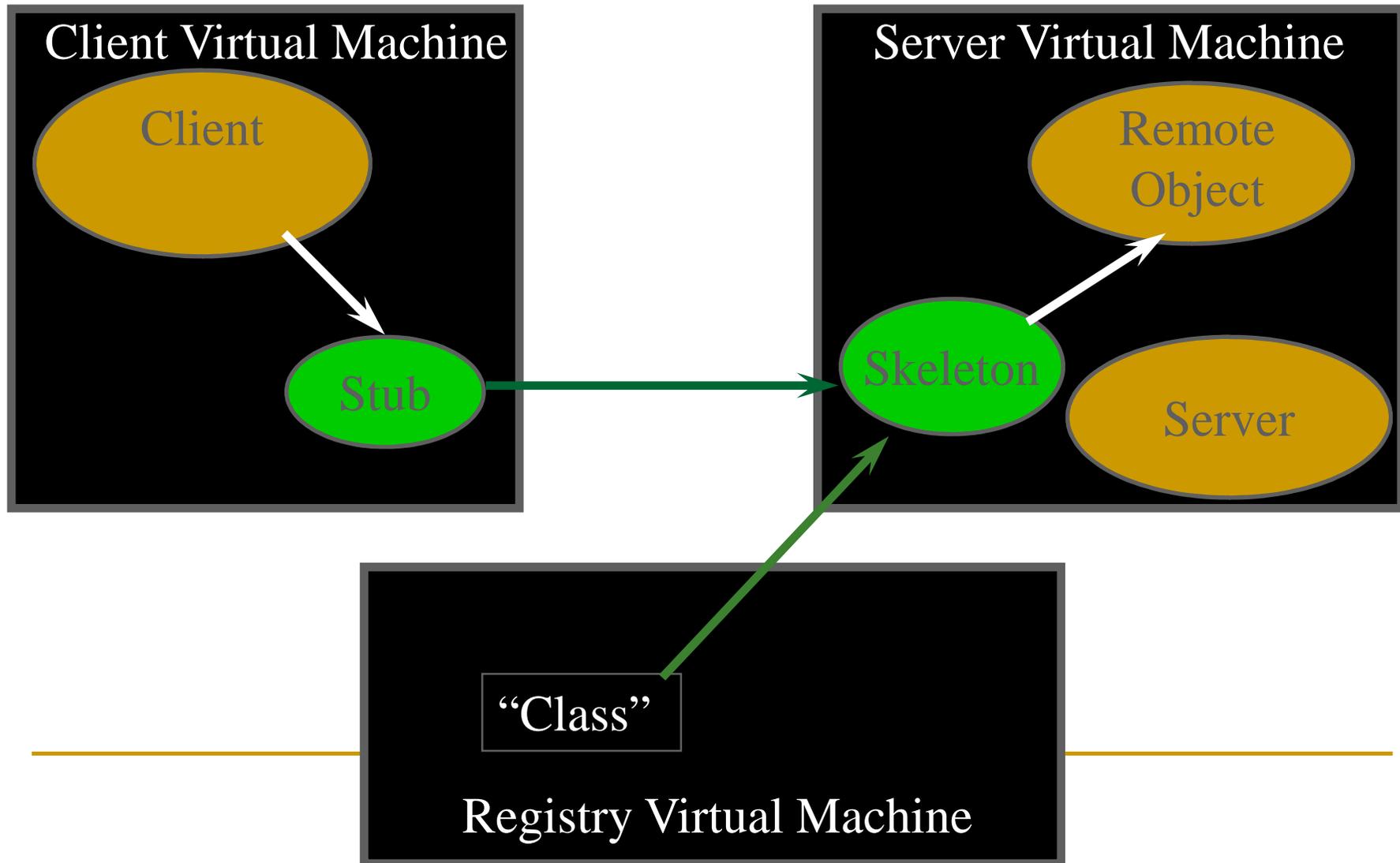


How RPC works?

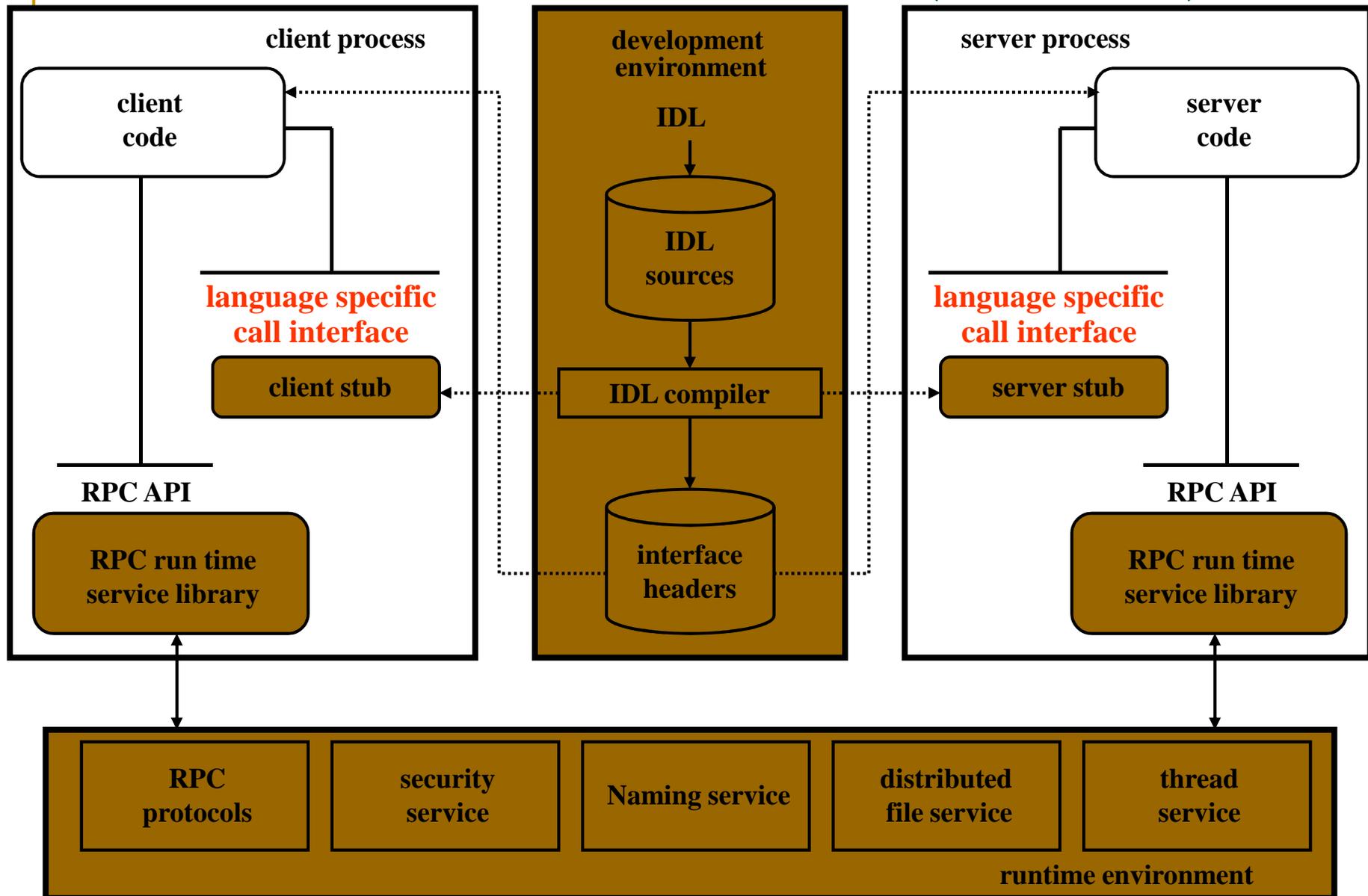
- What does an RPC system do?
 - **Hides distribution** behind procedure calls
 - Provides an **interface definition language (IDL)** to describe the services
 - **Generates** all the additional **code** necessary to make a procedure call remote and to deal with all the communication aspects
 - Provides a **binder** in case it has a **distributed name** and **directory service system**



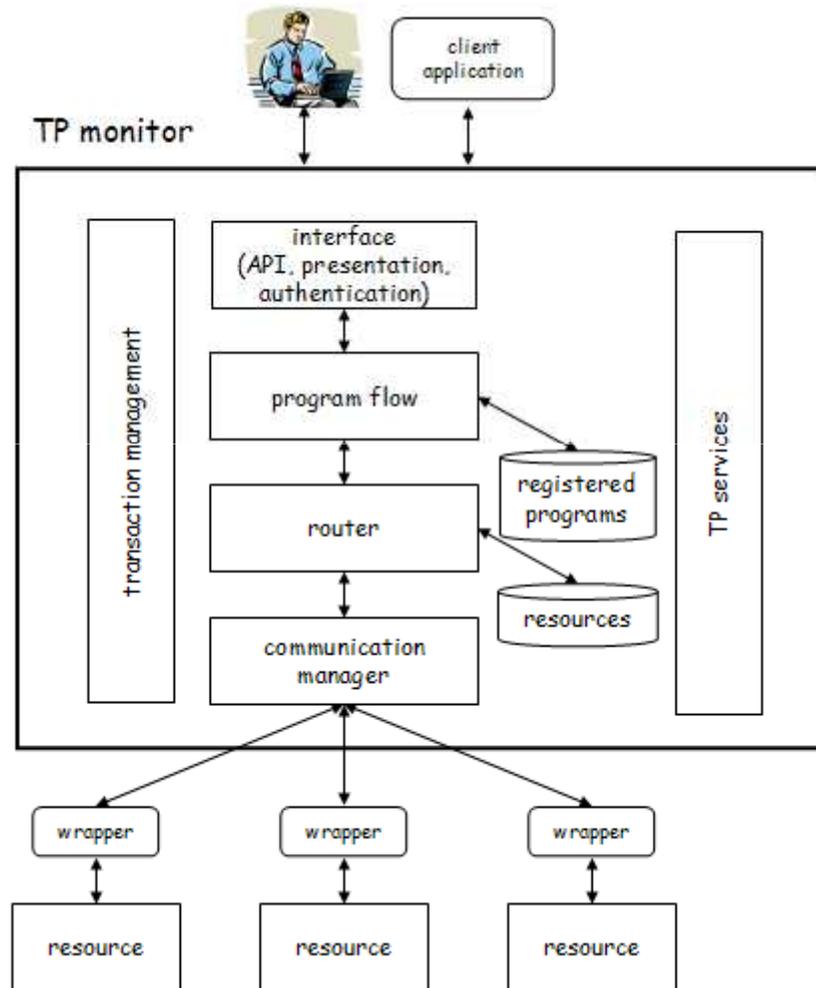
RMI System Architecture



Middleware as infrastructure (CORBA)

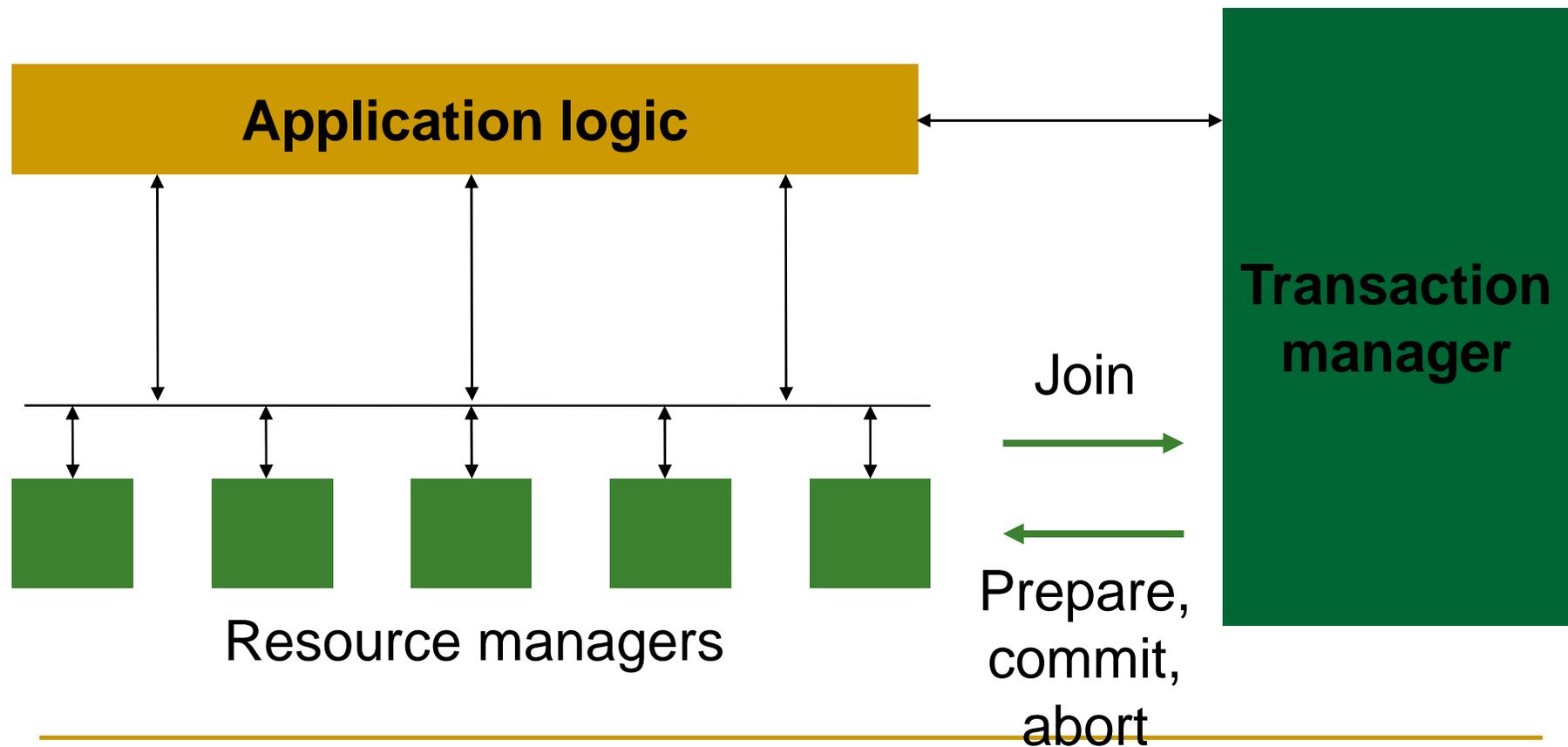


TP Monitor

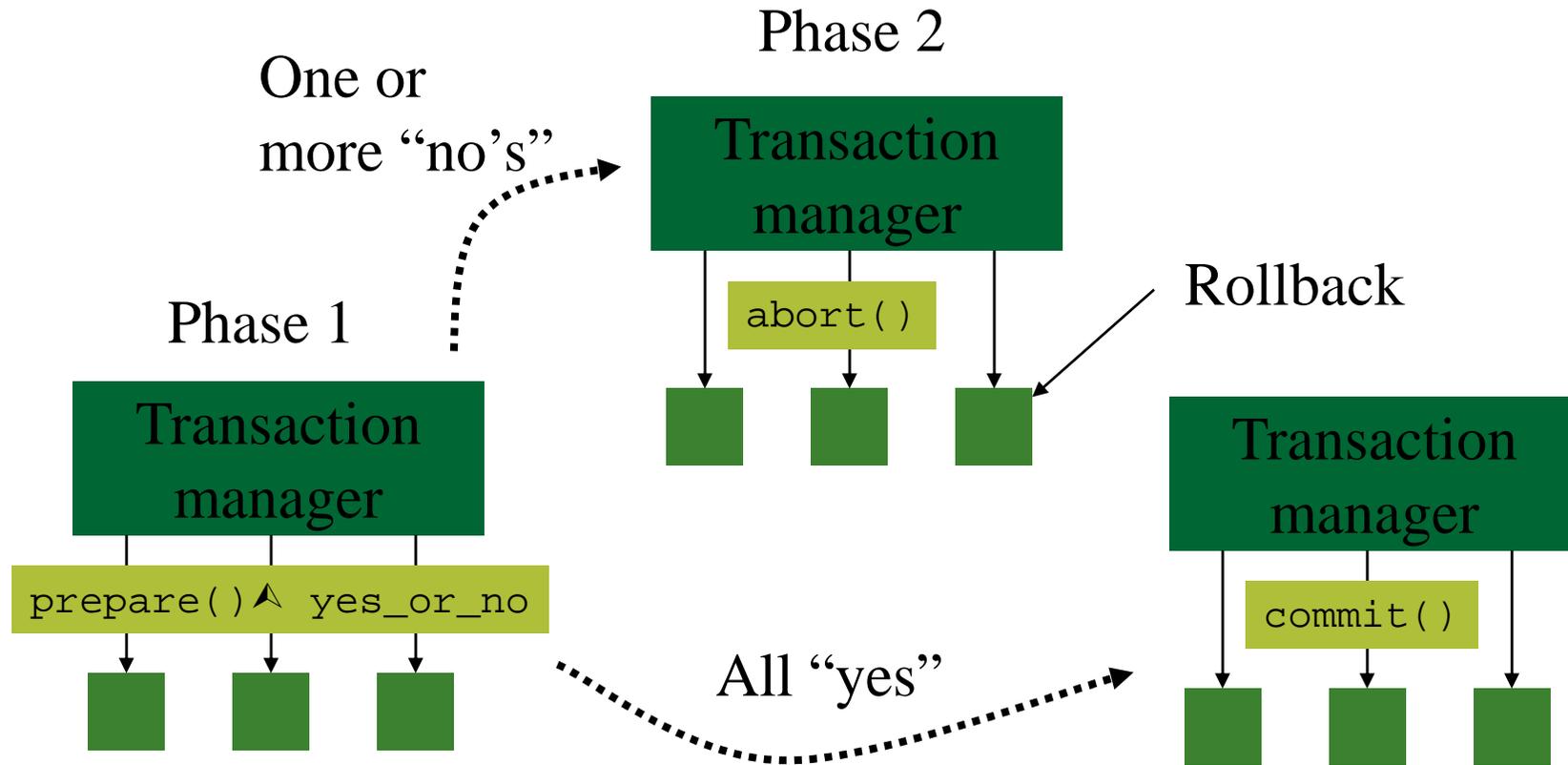


- TP Monitors are **middleware** systems that provide **transactional RPC**
- They, provide basic **RPC functionality** (IDLs, name servers, stub compilers, etc.)
- Used for banking **transactions**, purchasing plane tickets, etc
- A **TP-heavy monitor** provides
 - a full development environment
 - additional services (queues, priority scheduling, etc.)
 - support for authentication
 - its own solutions for replication, load balancing, storage management, etc.
- A **TP-lite system** is an extension to a database that
 - is implemented via threads, not processes
 - is based on stored procedures
 - does not provide a full development environment

Transaction Processing Architecture



Commit or abort



Message Oriented Middleware

- Mendukung **asynchronous** model message berbasis protokol TCP/IP
 - Menyediakan:
 - Kemampuan **message queue**
 - **Storage**: penyimpanan message
 - Ingat penyampaian pesan asynchronous
 - **Routing message**
 - Multicast / broadcast: pengiriman pesan lebih dari satu penerima
 - **Transformasi** pesan ke format standard secara otomatis (formatting message)
-

MOM (2)

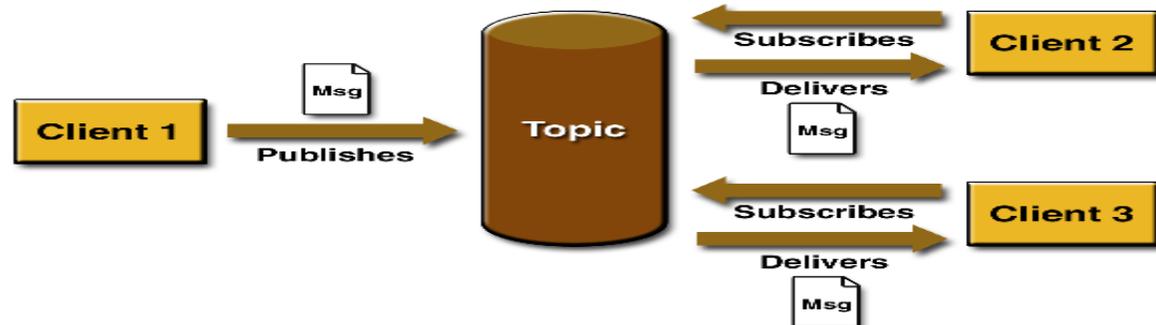
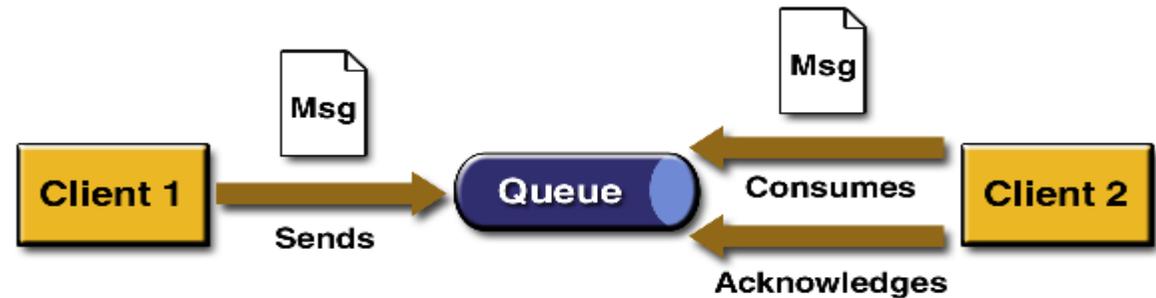
Two basic models

- **point-to-point**

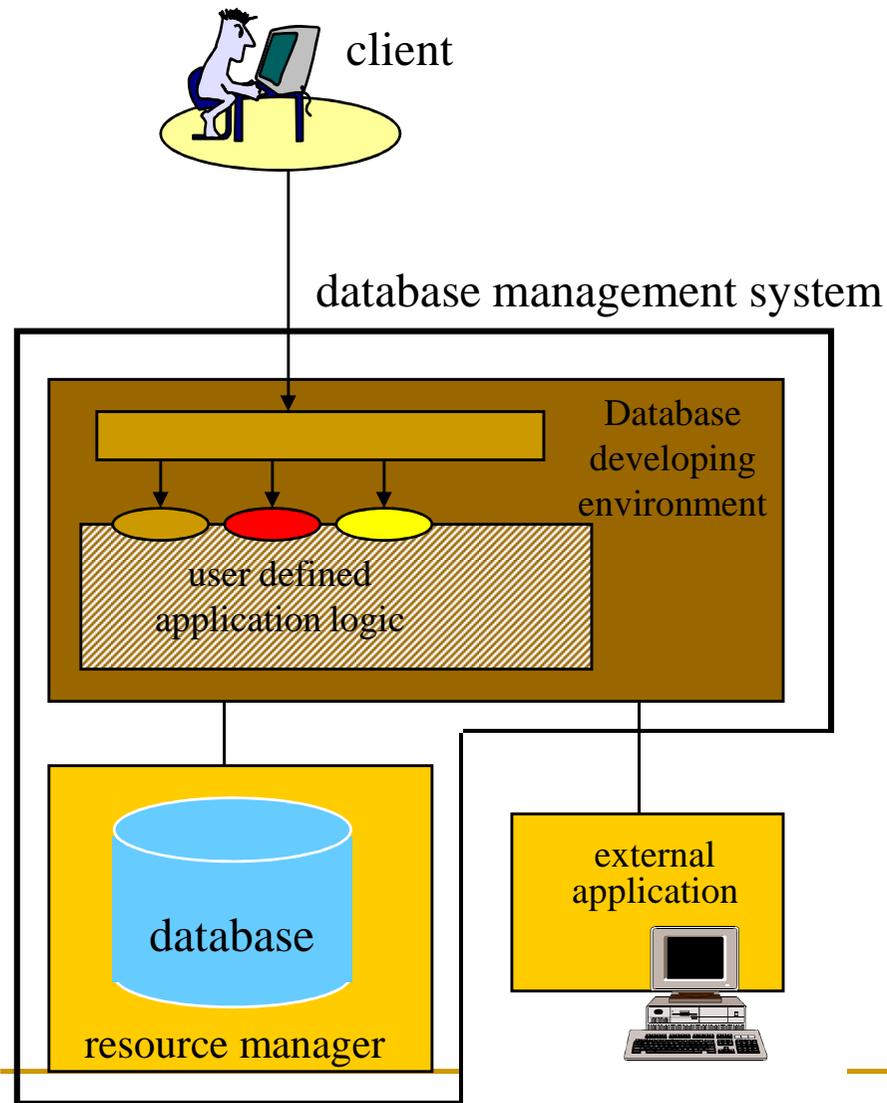
- **one component** posts a message to a server
- **one component** (and only one) will consume a posted message

- **publish/subscribe**

- allows a component to **publish** a message to a topic on a server
- components interested in a particular topic can **subscribe** to that topic (messages can be consumed by a number of components)
- when a component **publishes** a message, it subscribes to that topic and will **receive** the message

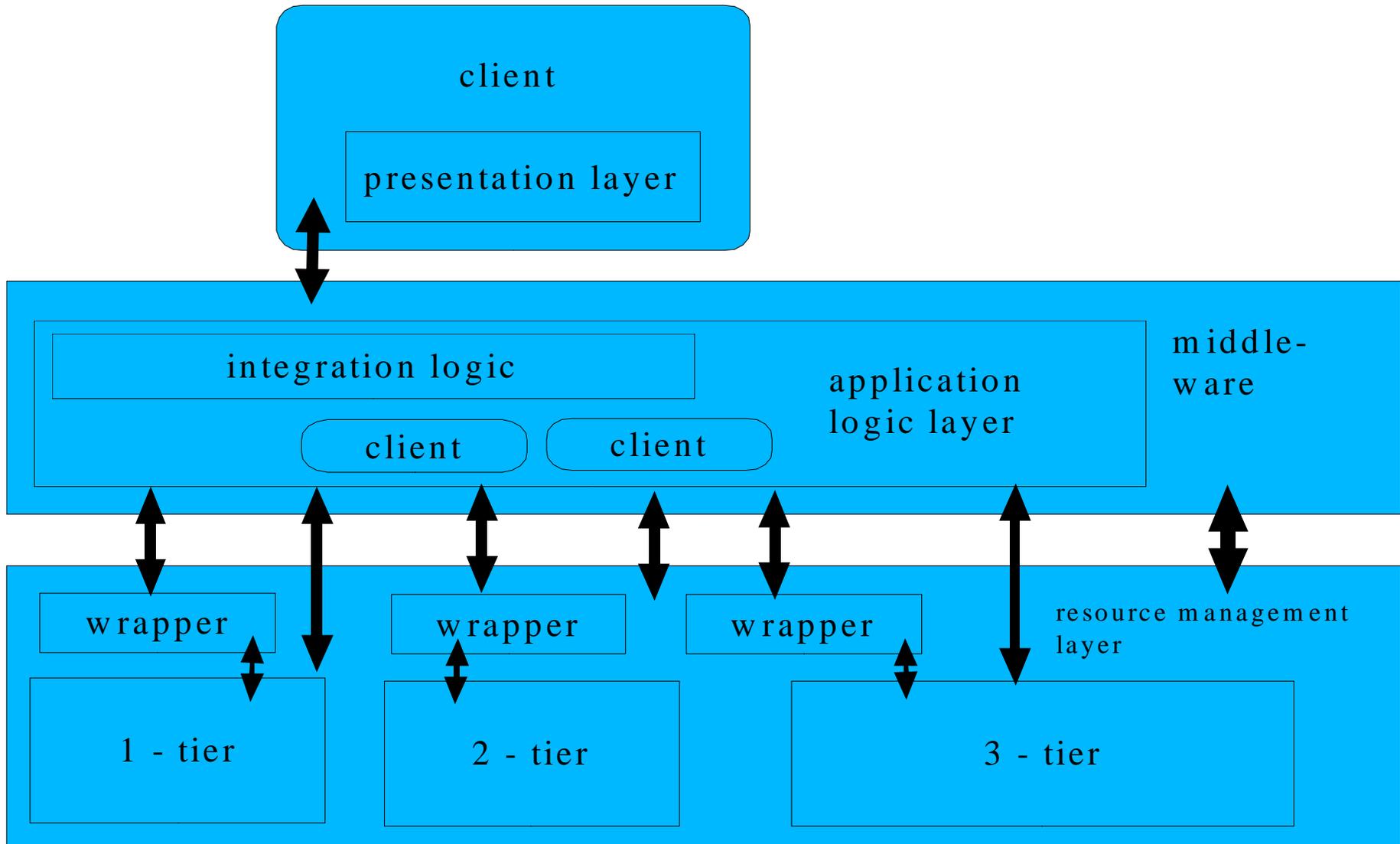


Databases and the 2 tier approach



- Databases are traditionally used to **manage data**.
- By doing this, **vendor** propose a 2 tier model with the **database** providing the tools necessary to implement **complex application logic**.
- These tools include: triggers, replication, stored procedures, queuing systems, standard access interfaces (**ODBC, JDBC**).

3 - tier



advantages & disadvantages

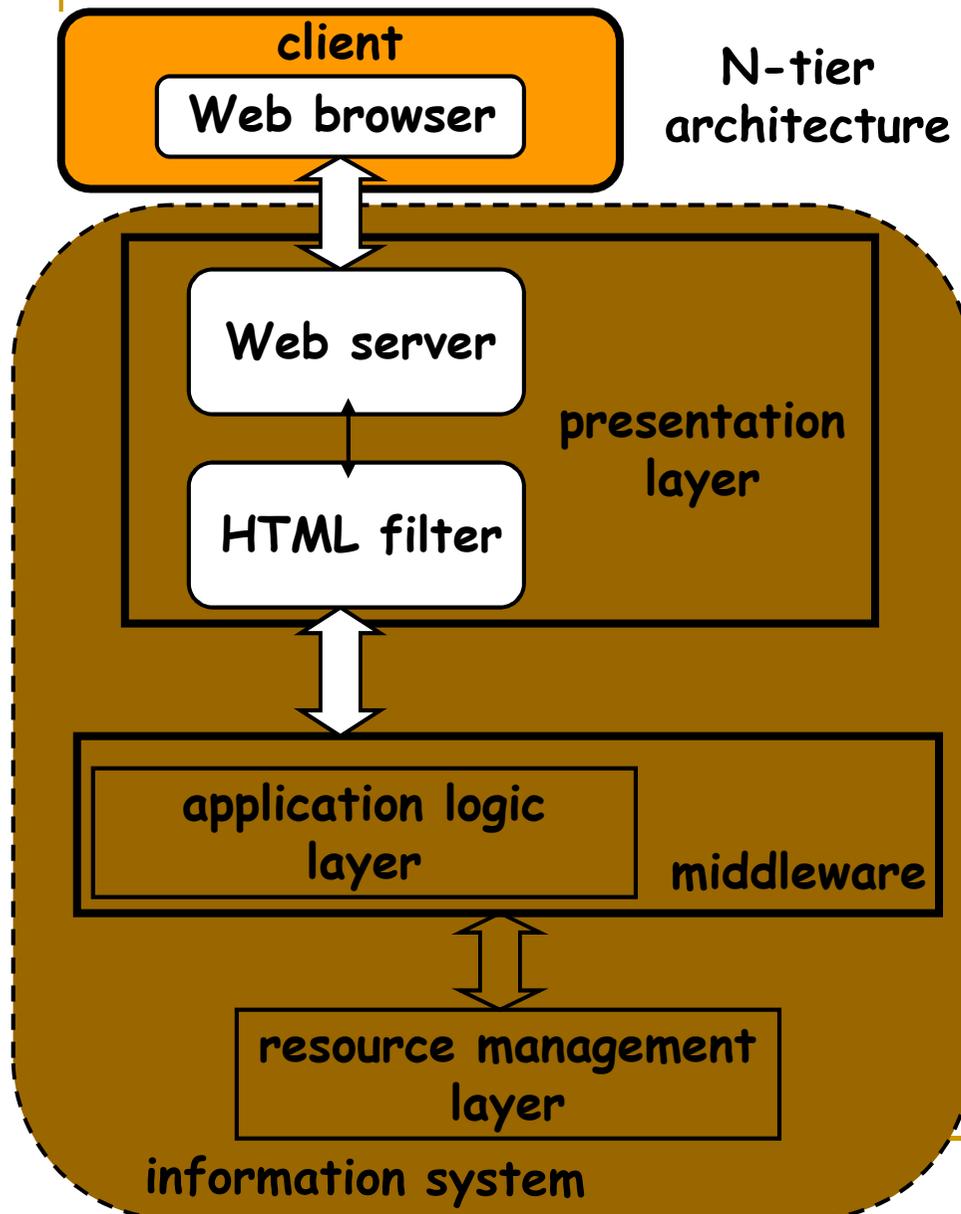
advantages

- **scalability** by running each layer on a different server
- scalability by distributing application logic (layer) across many nodes
- additional tier for integration logic

disadvantages

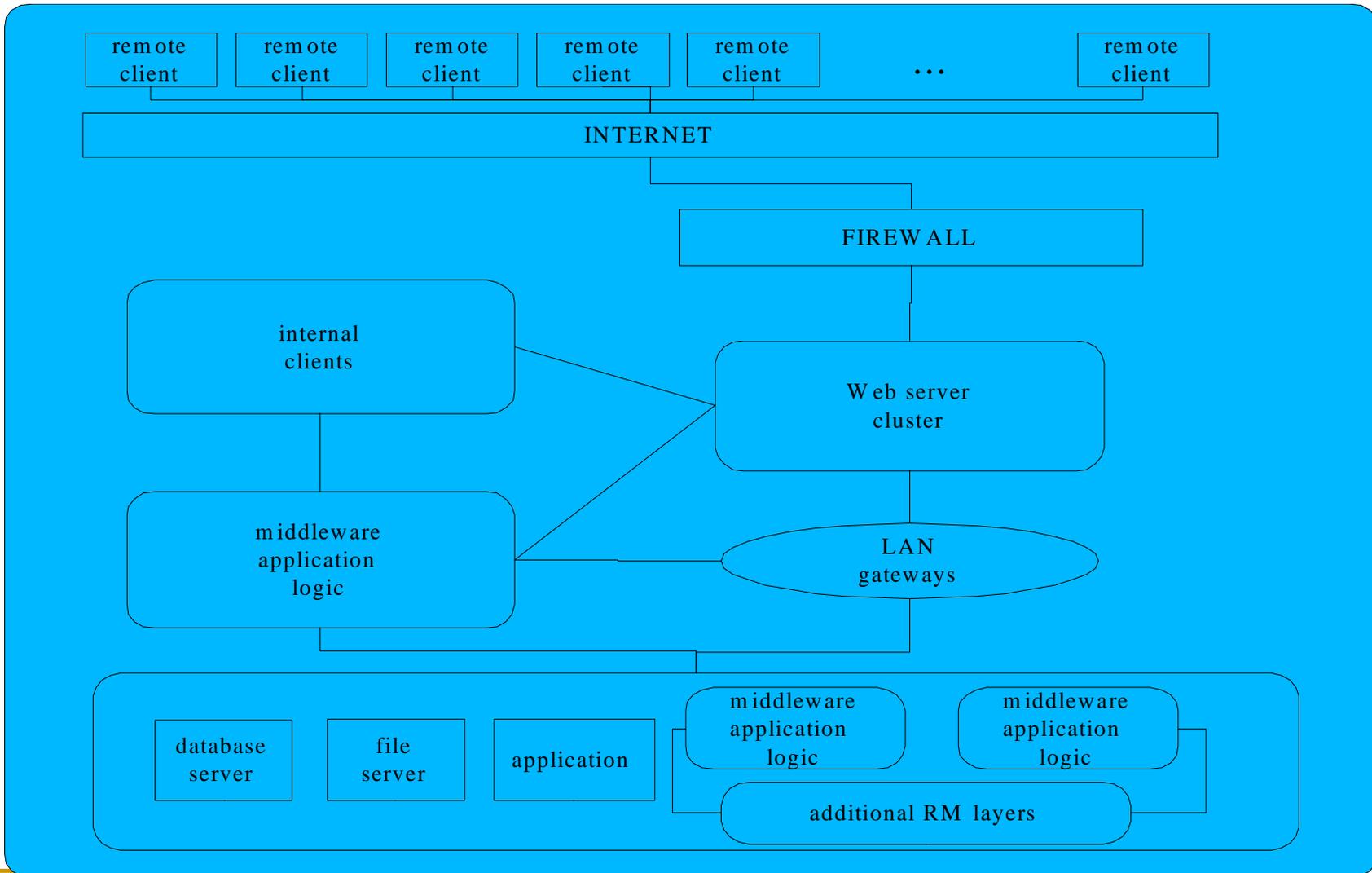
- **performance loss** if distributed over the internet
 - problem when **integrating** different 3 – tier systems
-

N-tier: connecting to the Web



- N-tier architectures result from **connecting several three tier systems** to each other and/or **by adding an additional layer** to allow clients to **access** the system through a Web server
- The Web layer was initially **external** to the system (a true additional layer)
- The addition of the Web layer led to the notion of “**application servers**”, which was used to refer to **middleware platforms** supporting access through the Web
- Ex: glass-fish, tomcat, Oracle App

n - tier



advantages & disadvantages

advantages

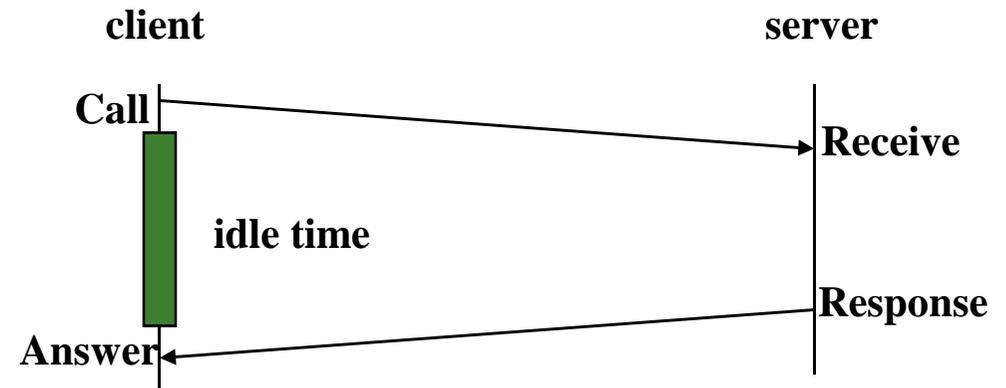
- better scalability
- higher fault tolerance
- higher throughput for less cost

disadvantages

- too much middleware involved
 - redundant functionality
 - difficulty and cost of developement
-

Blocking or synchronous interaction

- 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



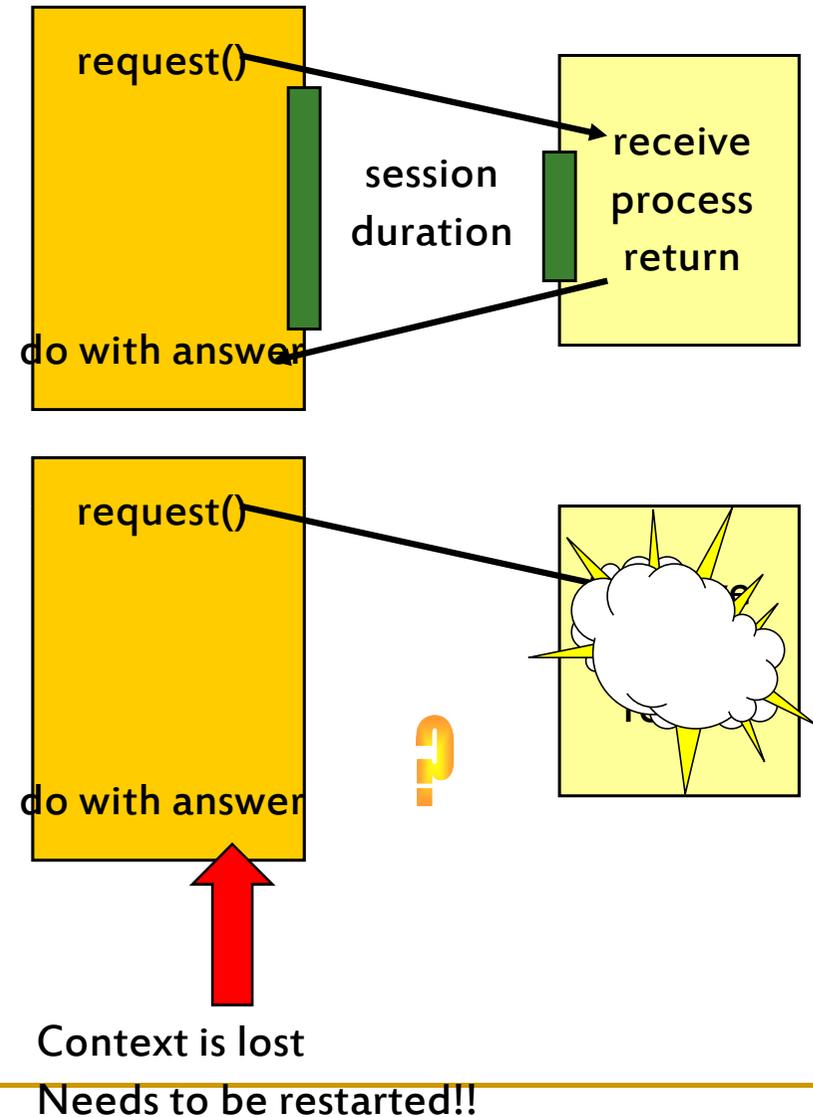
- Synchronous interaction requires both parties to be “**on-line**”:
 - the caller makes a request, the receiver gets the request, processes the request, sends a response, the caller receives the response.

Disadvantages of synchronous:

- 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)
-

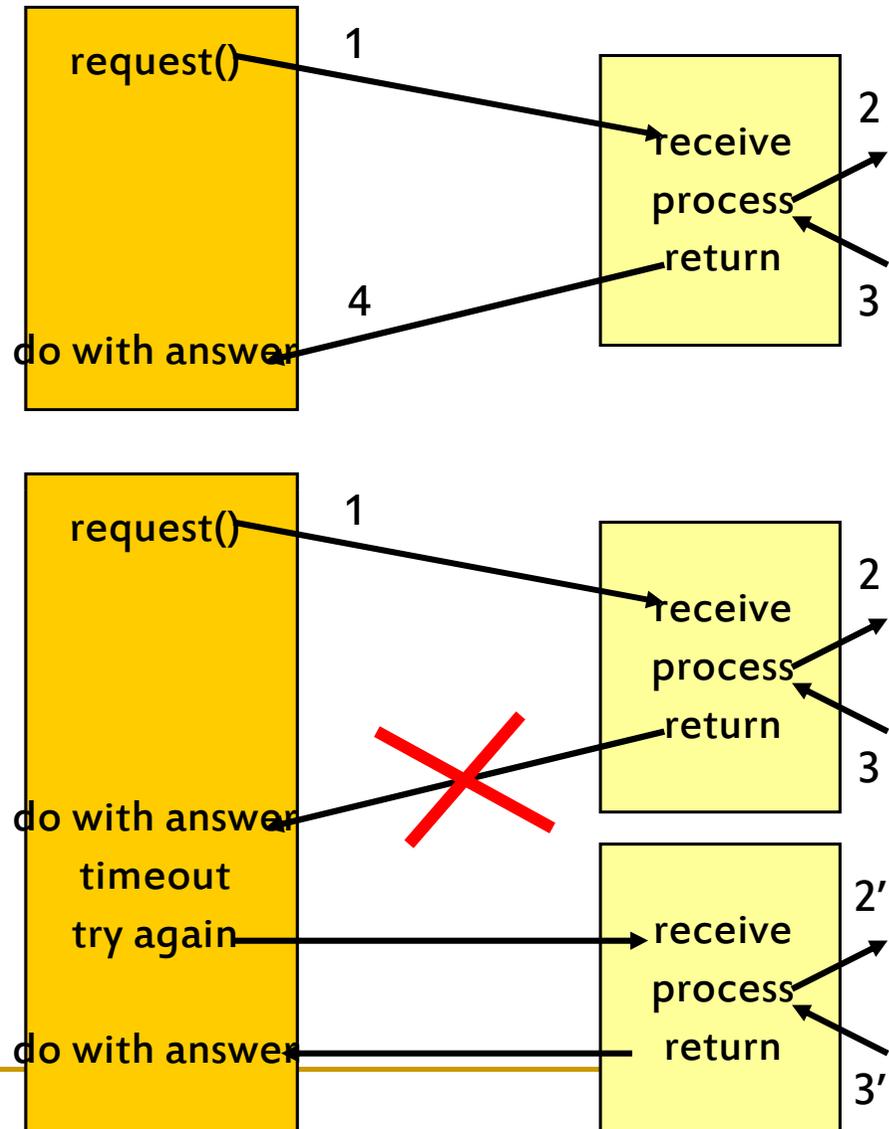
Overhead of synchronism

- 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** session
- 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.



Failures in synchronous calls

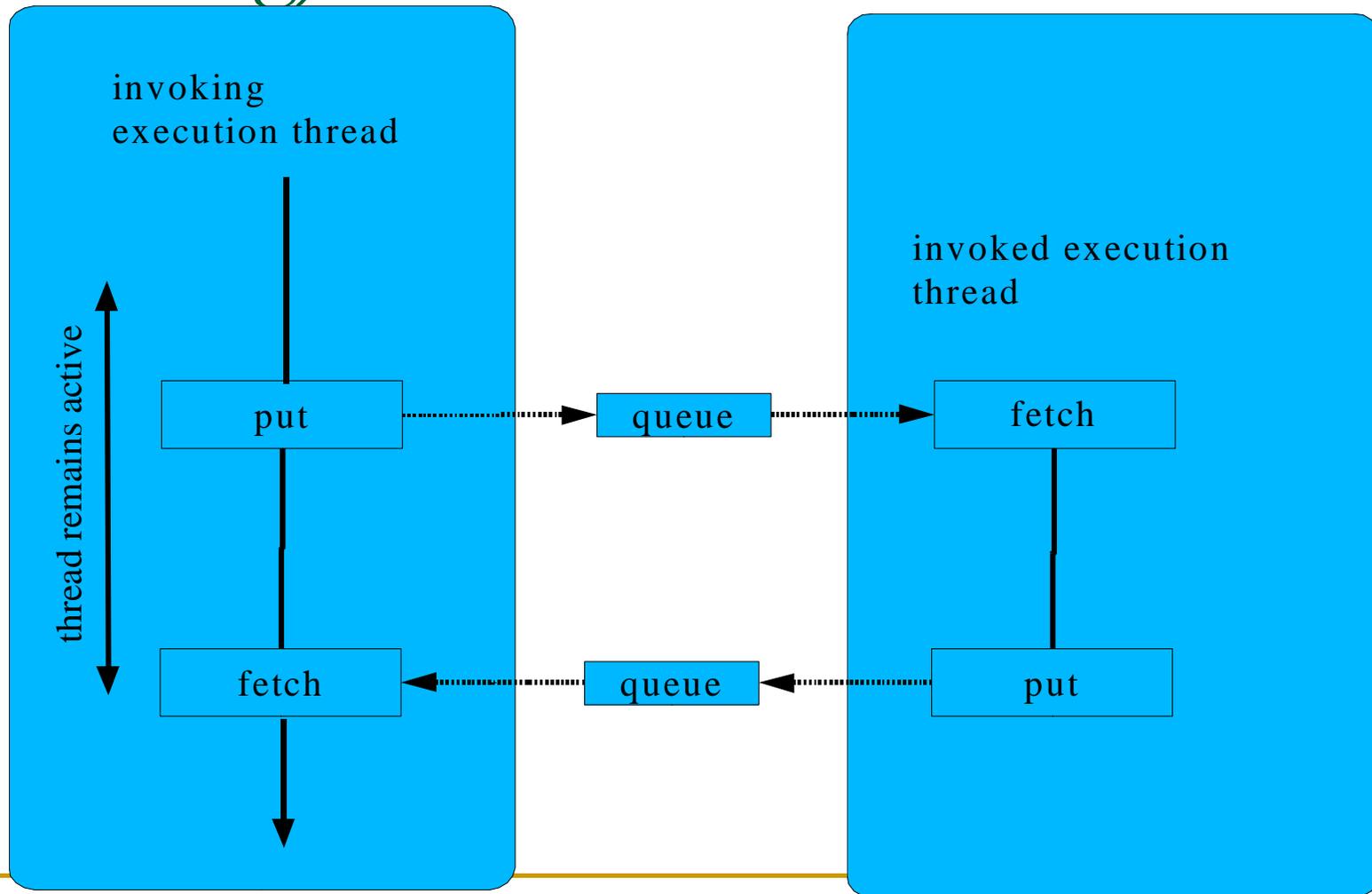
- If the client or the server fail, the context is **lost** and **resynchronization might be difficult**.
 - If the failure occurred before 1, **nothing has happened**
 - If the failure occurs after 1 but before 2 (receiver crashes), then the request is **lost**
 - If the failure happens after 2 but 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?



ASYNCHRONOUS INTERACTION

- **Provides Transactional interaction:** to enforce exactly once execution semantics and enable more complex interactions with some execution guarantees
 - **Provides 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)
 - Using **asynchronous** interaction, the caller sends a message that gets stored somewhere until the receiver reads it and sends a response.
 - 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)
-

asynchronous interactions (non blocking)



See u next week

- Developing Enterprise Application Techniques

