


COMP9322

Service Oriented Architectures

Introduction



Helen Paik

School of Computer Science and Engineering

University of New South Wales

Week 1

Who's Who in COMP9322

- ✦ *Lecturer-in-Charge*
- ✦ *Helen Paik (hpaik@cse)*
- ✦ *Office: K17 401A, Ext: 54095*
- ✦ *Consultation times: see Course Homepage*
- ✦ *Course Homepage:*
 - ✦ <http://www.cse.unsw.edu.au/~cs9322>

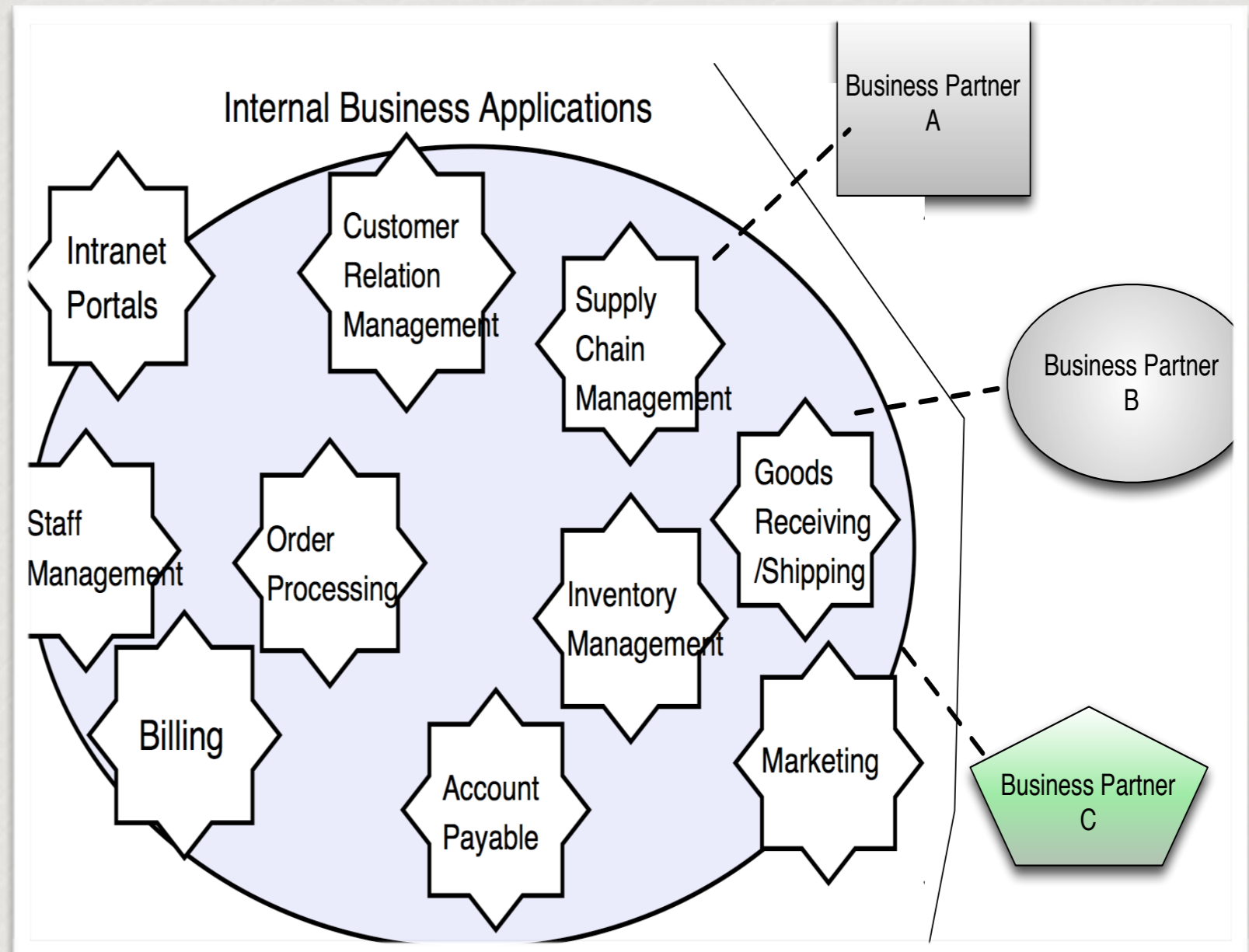
Course Aims

- ✦ *from building a web site (cs9321) to building web services (cs9322) ...*
- ✦ *context: “global/distributed/complex” business applications*
- ✦ *you should be able to:*
 - ✦ *understand the concept of services and business processes*
 - ✦ *articulate the motivation behind web service-based technologies*
 - ✦ *apply the knowledge in practical situations*

This course aims to provide students with a deep understanding of SOA, service-orientation paradigm, business processes and Web services as an implementation technology

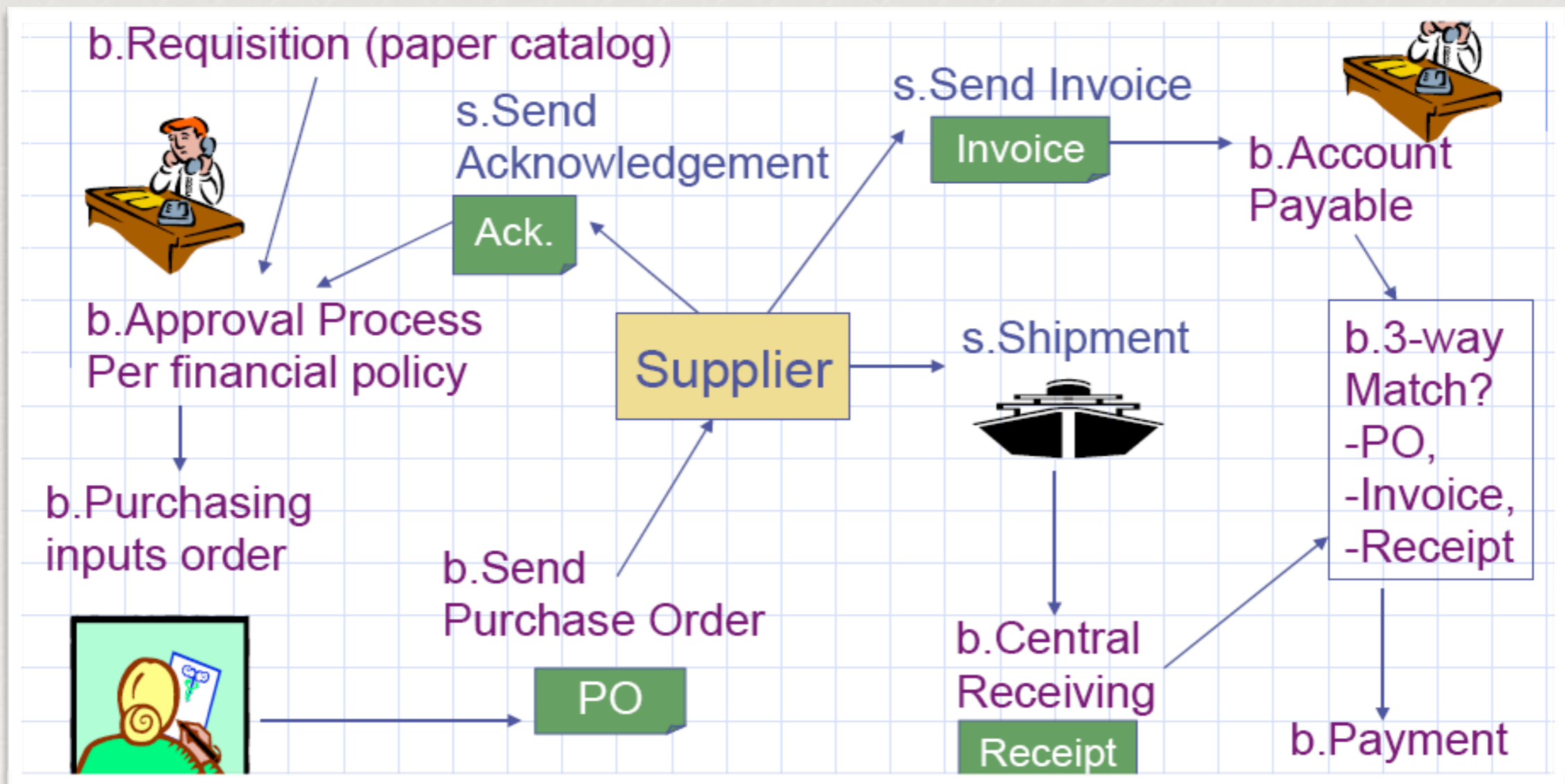
What are we learning? - course context

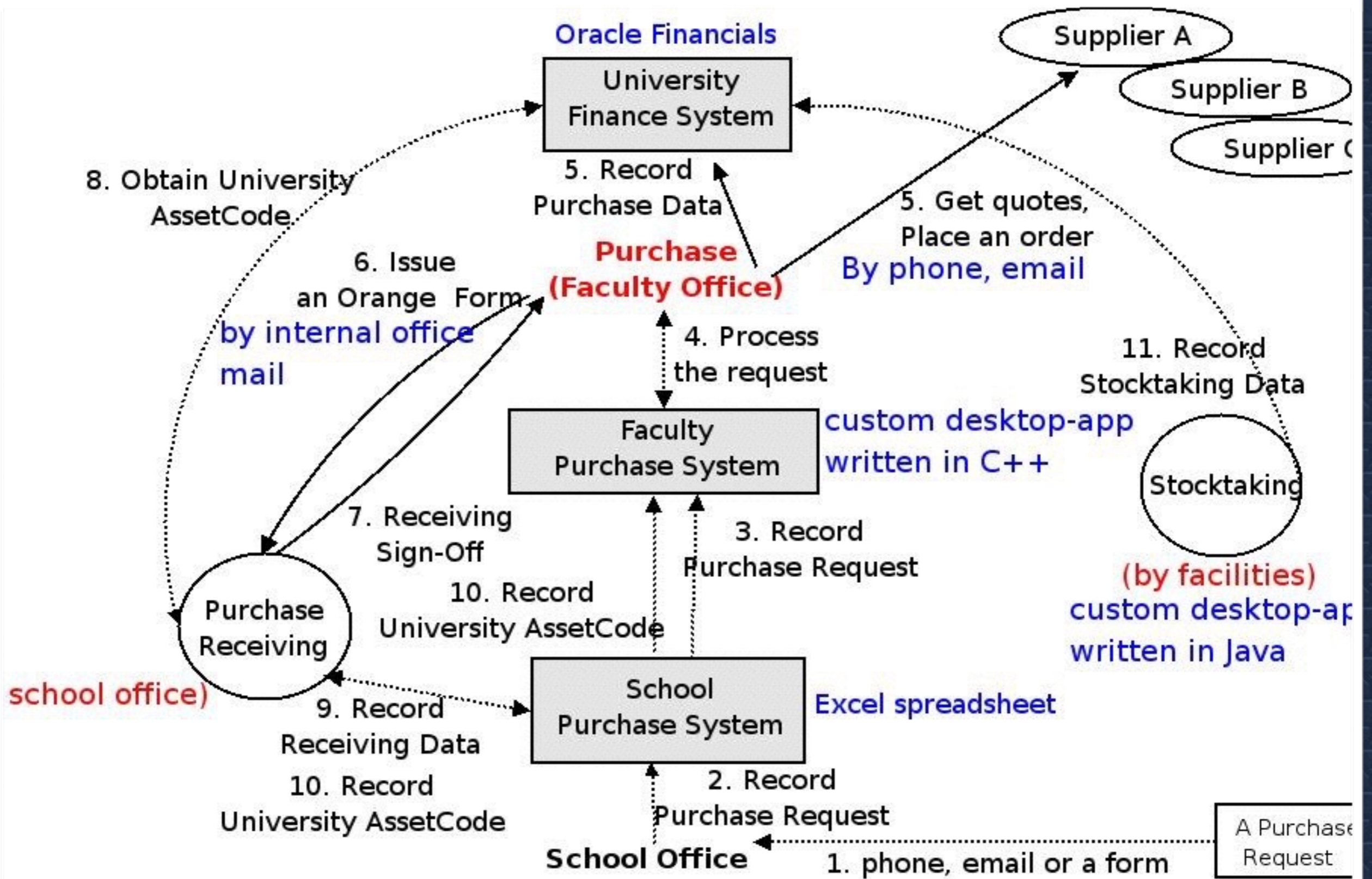
- ✦ *complex/distributed/global information systems in enterprises*
- ✦ *Pick any sizeable organisation: it has many departments performing different functionality - in silos, often supported by software systems*



A Typical Purchase Order Process

- ✦ *In reality: communication/coordination between the silos needed*

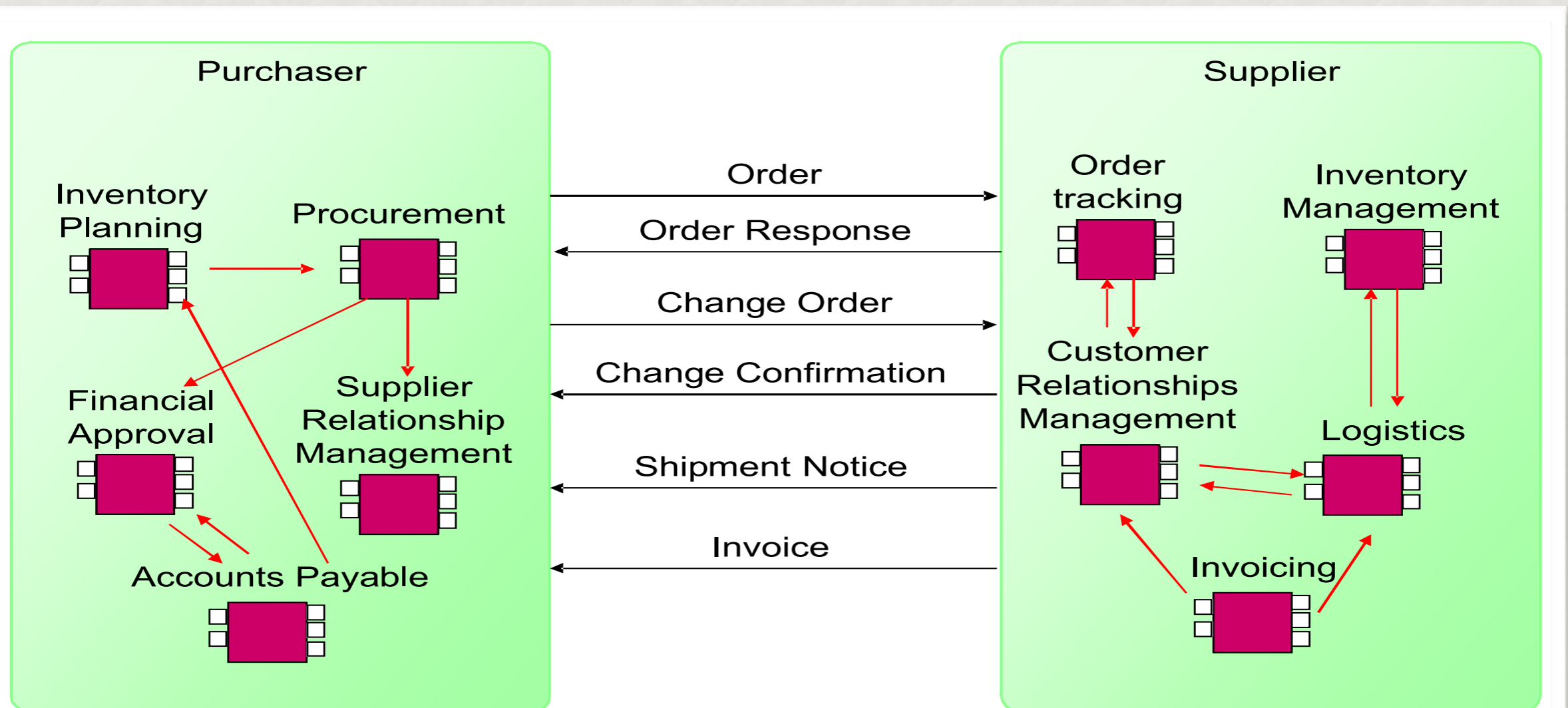




An example of (real) Purchase Order Process

The problem at a glance:

- How do we make this easy ...

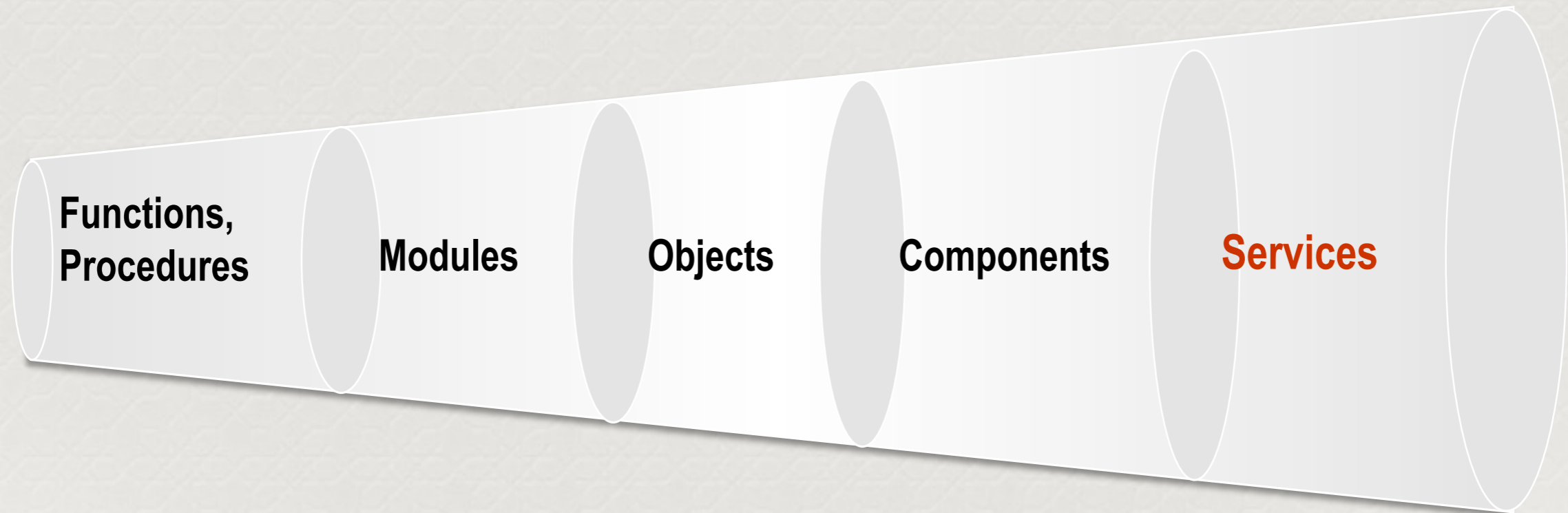


So, what are we learning, again ...?

- ✦ *What do we do when we have a complex problem? — we abstract:*
 - ✦ *... we need a model*
 - ✦ *... and methodology for model design*
 - ✦ *... and implementation/execution platform for realising the model*
- ✦ *Service-orientation or Service Oriented Architecture is such an approach.*
- ✦ *An analogy:*
 - ✦ *OO allows us to model/implement software components as objects,*
 - ✦ *SO allows us to model/implement software systems in terms of services*

The evolution of programming abstractions

*Evolution of Programming Abstractions: Dr Marcello La Rosa, QUT,
Introduction to Web Services*



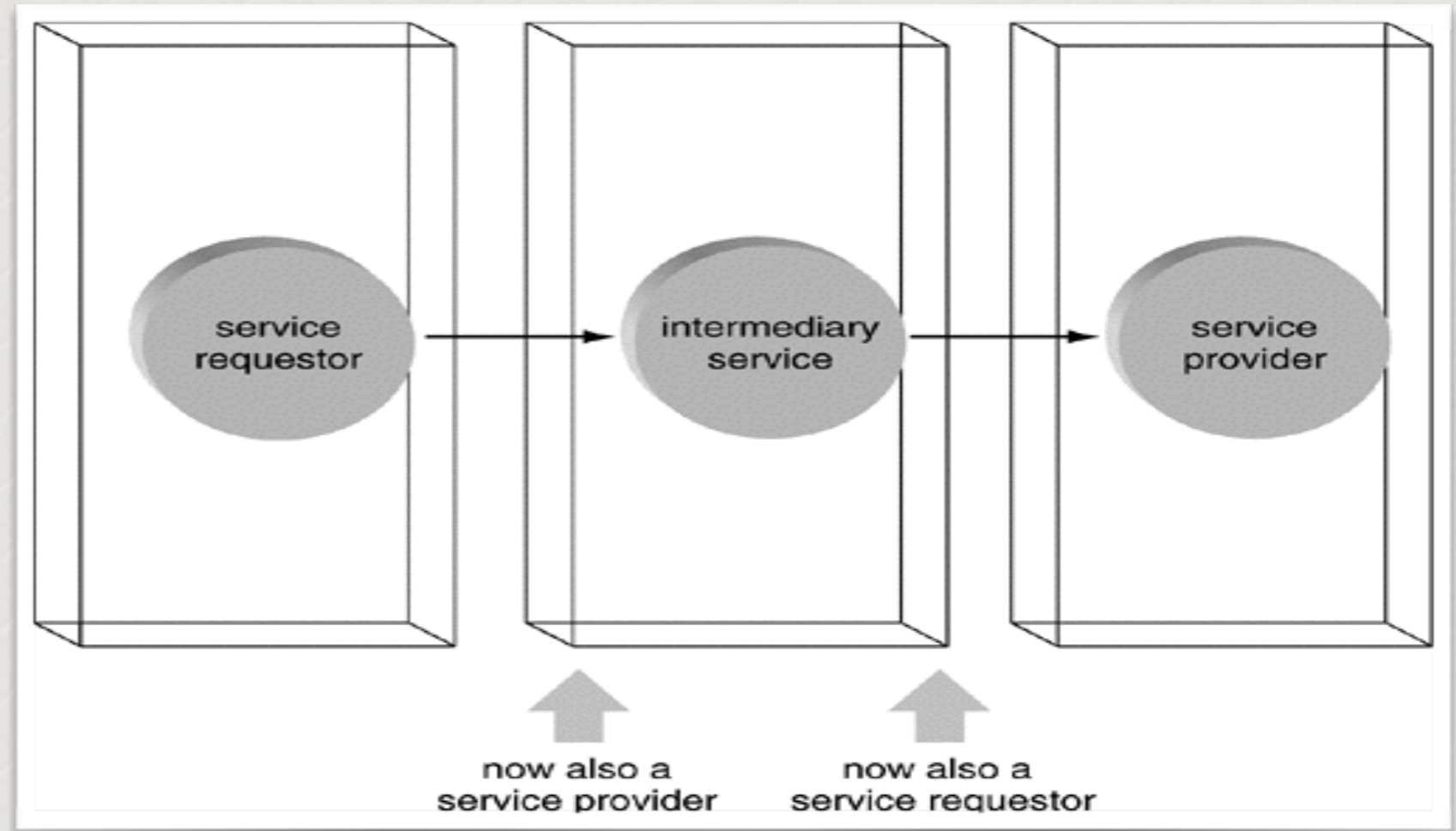
- ✦ *Lines of code vs. Services - consider software building exercise as 'building services', 'discovering services' and 'combining services'*

The evolution of programming abstractions

- ✦ *In SOA, we talk about software as a service ... That is, SOA is about building software systems composed of a collection of (software) services*
- ✦ *A software service:*
 - ✦ *A software asset that is deployed at an endpoint and is continuously maintained by a provider for user by one or multiple clients*
 - ✦ *Services have explicit contracts that establish their purpose and how they should be used*
 - ✦ *Software services are (supposed to be) reusable (“compose-able”)*

So, what are we learning again ...?

Simplified view of
Service
Orientation:



Service-orientation - a way of integrating your business as a set of linked services. If you can define the services, you can begin to link the services to realise more complicated 'services'

Learning Outcomes

- ✿ *At the end of this course, you will know (hopefully!)*
 - ✿ *How to describe the problem area and motivation behind SO paradigm.*
 - ✿ *How to design and implement services*
 - ✿ *How to design and implement business processes using services ***
- ✿ *Also: you should know different 'flavours' of services (WS-*, RESTful, Data Service)*

*** : Generally speaking, we identify a repeatable task within a business process as a service ... the tasks are services and the business process is a composition of services.*

Weekly Schedule ... Assessment

Weekly Schedule: <http://www.cse.unsw.edu.au/~cs9322>

- ★ *Assessment:*

- ★ *55% formal written exam: individual assessment.*
- ★ *35% on assignment work: group assessment (group of 1 or 2 only). Two assignments.*
- ★ *10% on three to 4-5 online quizzes (WebCMS-based quiz system, 'open' test)*
- ★ *Final Mark = quizzes + assignments + exam*

Labs and Assignments

★ *Labs:*

- ★ *A self-guided lab exercise is released (roughly) every two weeks from Week 2*
- ★ *You can do them in your own time.*
- ★ *You are encouraged to use lab consultation times, messageboard if help is needed.*
- ★ *Every week, after the lecture, there is a lab consultation at Piano (6-7.30pm)*

★ *Assignments:*

- ★ *Two assignments (35 marks total ...)*
- ★ *The assignments are group-based (group of 1 or 2 only).*
- ★ *Exact assessment methods/marking criteria to be announced in each assignment*

Supplementary Exam Policy

- ✦ *Supp Exam is only available to students who:*
 - ✦ *DID NOT attend the final exam*
 - ✦ *Have a good excuse for not attending*
 - ✦ *Have documentation for the excuse*
- ✦ *Submit special consideration within 72 hours*

Everybody gets exactly one chance to pass the final exam. For CSE supplementary assessment policy, follow the link in the course outline.

Plagiarism

- ✿ *UNSW (and CSE) considers plagiarism as a serious offence. A student who plagiarises will be dealt with by the school and possibly by the university.*
- ✿ *More information about the school policy on this can be found at:*
 - ✿ *www.cse.unsw.edu.au/~studentoffice/policies, follow the link “Originality of Assignment Submissions”*
- ✿ *UNSW’s learning centre also provides an online-resource containing information about plagiarism which you should be familiar with already*

Course Reference Books

The course content is based on a number of books. They are listed here as recommended textbooks.

- ✦ *(Alonso Book) Web Services by G Alonso, F Casati H Kuno and V Machiraju, Springer*
- ✦ *(Webber Book) Developing Enterprise Web Services: An Architect's Guide by S Chatterjee and J Webber, Prentice Hall*
- ✦ *(Blue Erl Book) Service-Oriented Architecture: Concepts, Technology, and Design by Thomas Erl, Prentice Hall*
- ✦ *(Mike Book) Web Services: Principles and Technology by Michael Papazoglou, Prentice Hall*

A Few Other Things ...

- ✿ *Use course homepage*
- ✿ *Read the course notice board*
- ✿ *Participate in the MessageBoard discussions*
- ✿ *Collaborative, helping-each-other-out environment*
- ✿ *Questions on Assignments/Labs -> **USE** Messageboard*
- ✿ *Use of laptops during lectures (?)*
- ✿ *Use of mobile phones during lectures (!)*

3-Tier Architecture



Homogenous

Language Dependent

Centralized Application Tiers

Code Centric Applications

Request/Reply Driven

HTML Pages

SOA



Heterogeneous

Language Independent

Massively Distributed Services

Flexible Composite Applications

Request/Reply, Pub/Sub, Events

AJAX Rich Internet Applications

Introduction to SOA

Backgrounds

Week 1 Objectives

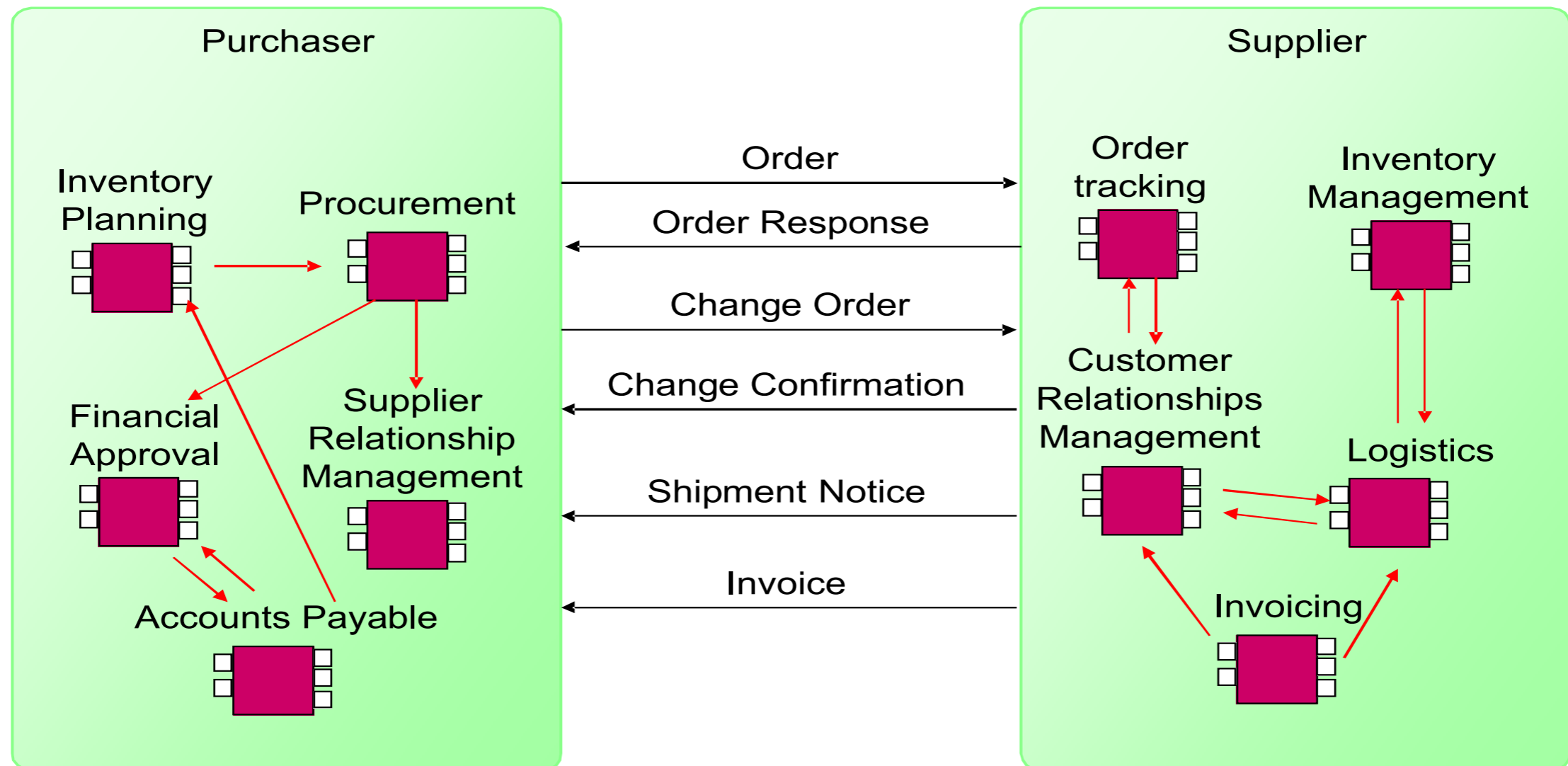
- ★ *Motivation*

- ★ *Web services are a form of distributed information system*
- ★ *Considering how distributed information system evolved help us understand the background of the technology*

- ★ *Learning Outcomes (Week 1)*

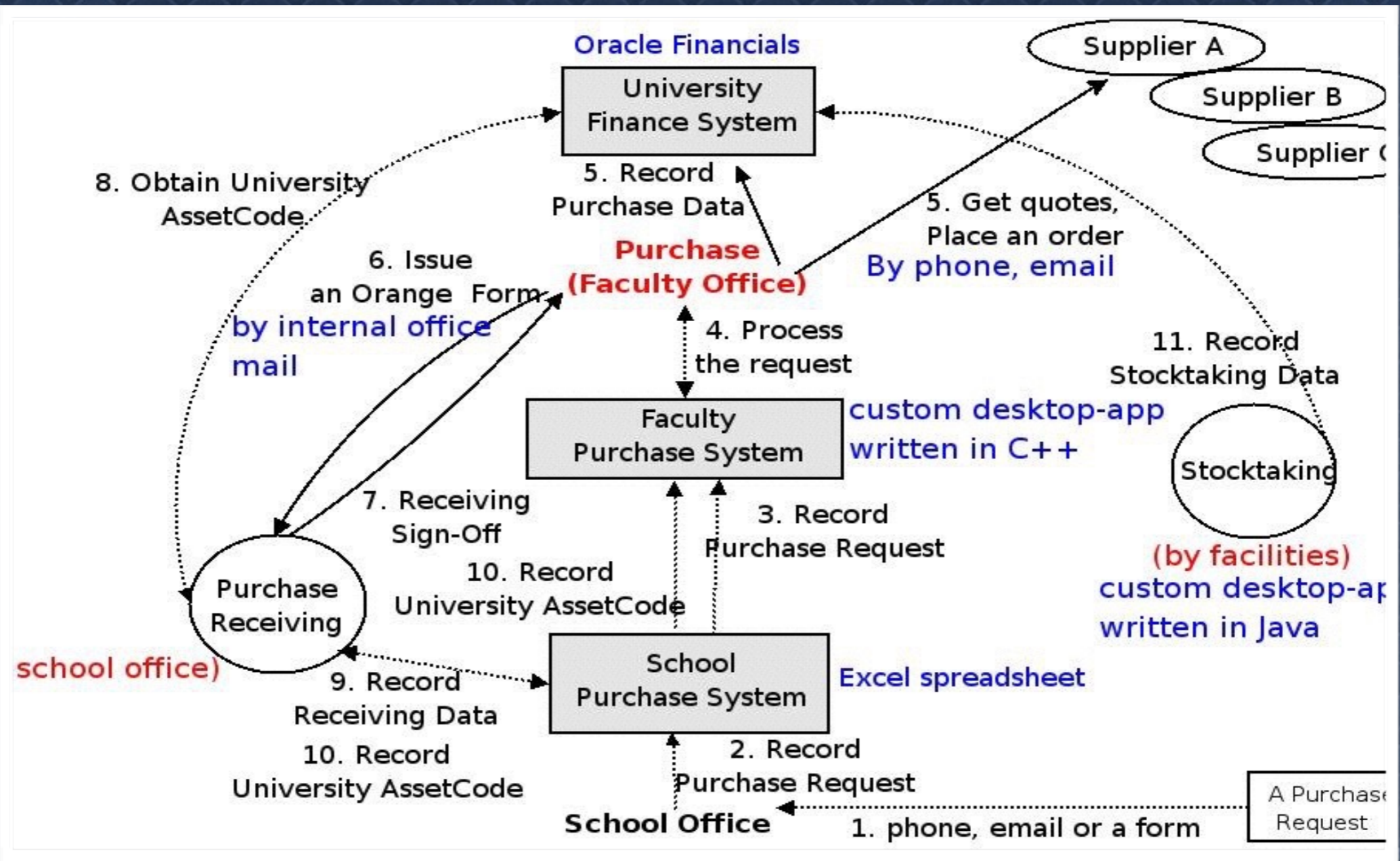
- ★ *Identify different conceptual layers in distributed information systems*
- ★ *Describe different architecture of distributed information systems*
- ★ *Explain the communication patterns in distributed information systems*
- ★ *Reference: (Alonso Book) Chapters 1-3*

The problem at a glance: how do we make this easy



Enterprise Application Integration (EAI)

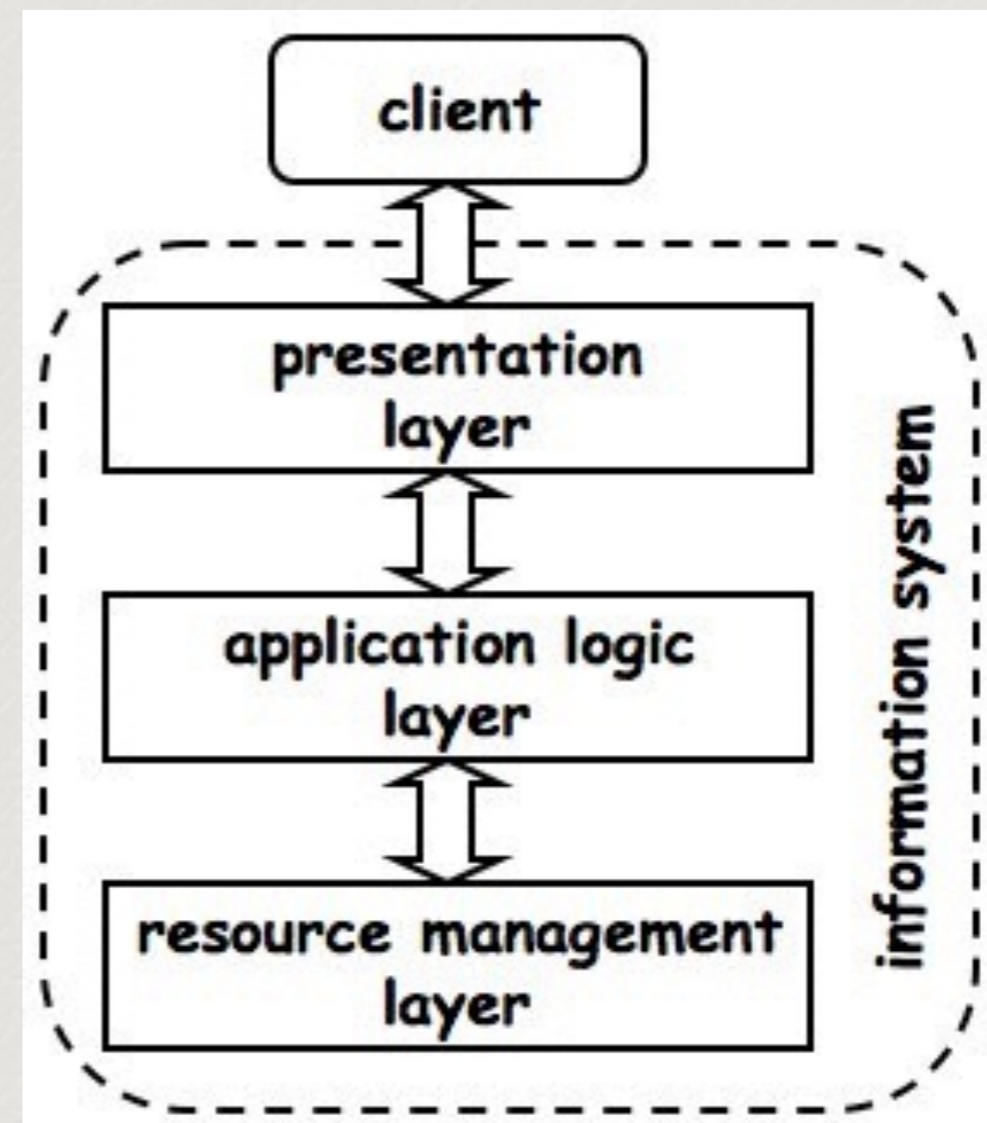
- ✦ *Motivations: Streamlining business operations, globalisation, competition, mergers and acquisition, new business models, technology development (e-commerce), etc.*
- ✦ *EAI definition by Hewlett Packard: A set of services and solutions for **brining together disparate application and business processes** as needed to meet the diverse information requirements of your customers, partners, suppliers and employees.*
- ✦ *Problems: systems to be integrated are not homogeneous.*
 - ✦ *they are individually developed (ad-hoc) systems overtime*
 - ✦ *some are “off-the-shelf” packages*
 - ✦ *different execution platforms, technologies and business rules*



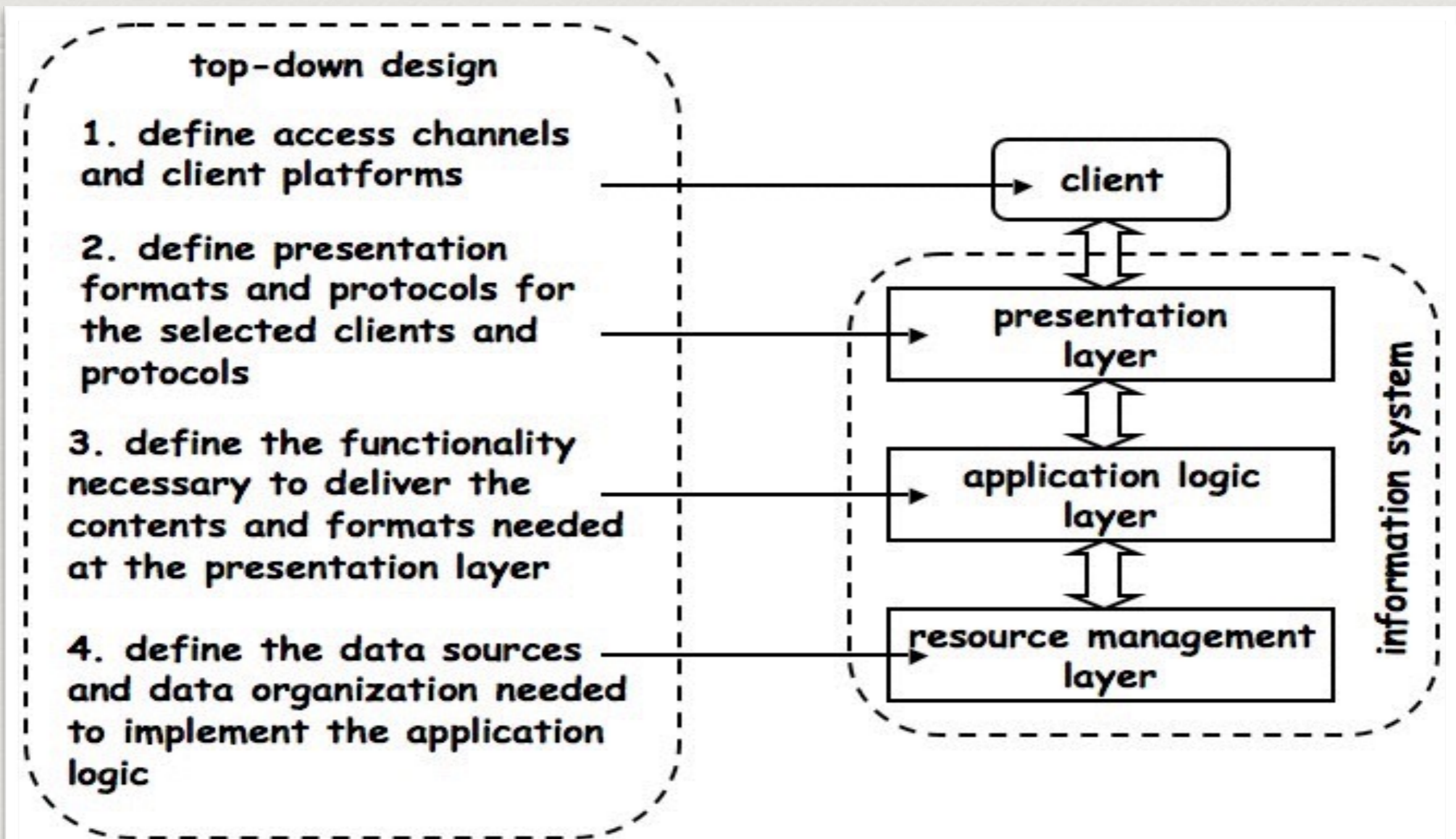
An example of (real) Purchase Order Process

Conceptual Design of Information Systems: Layers/Tiers

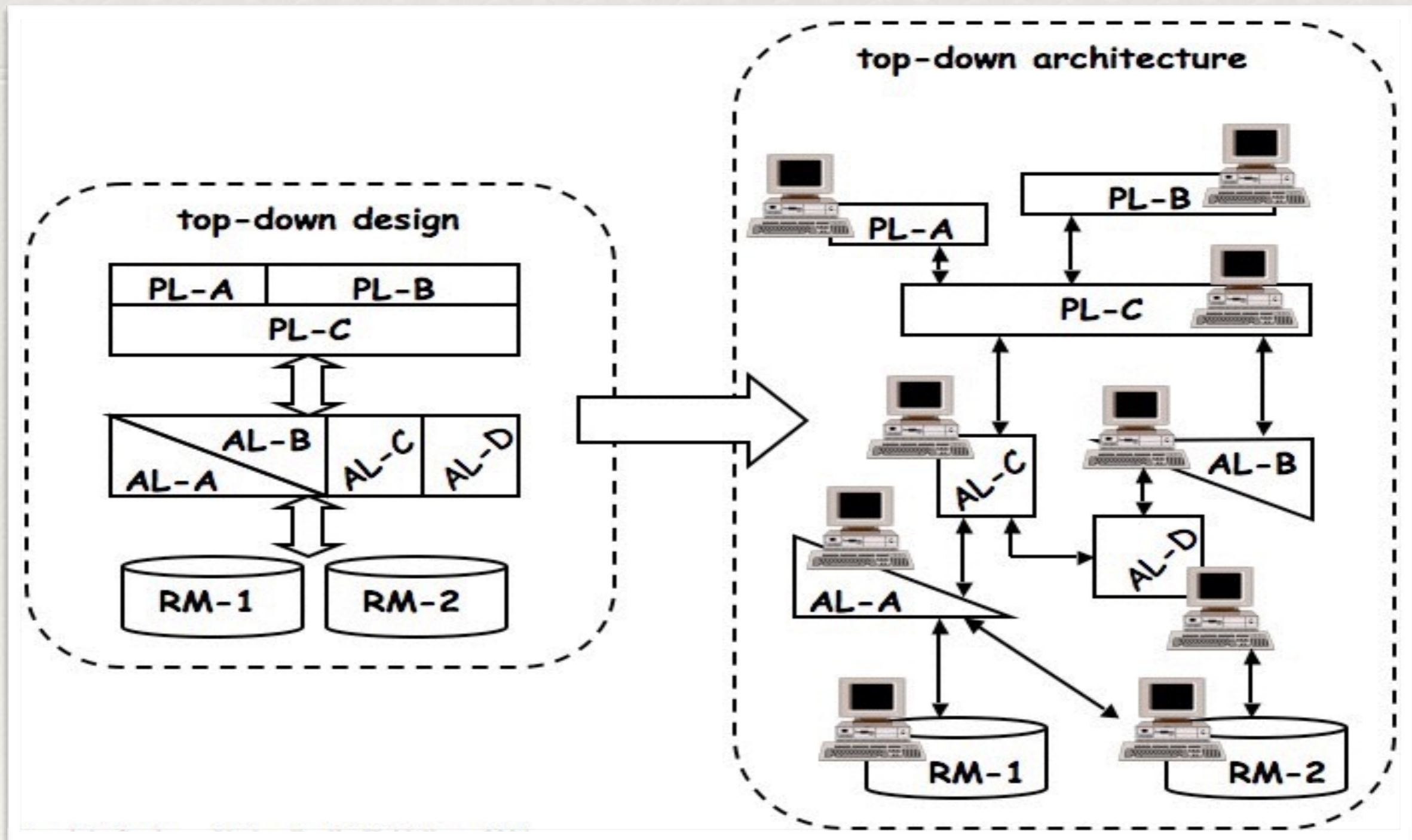
- ✦ **PL:** formatting, presenting information to clients (e.g., JSP)
- ✦ **AL:** determines what the system actually does, enforcing the business rules and processes (e.g., a program that implements customer registration)
- ✦ **RM:** storage, indexing, and retrieval of the data necessary to support the application logic layer (e.g, RDBMS)



Information System Design: Top-down Design



Information System Design: Top-down Design

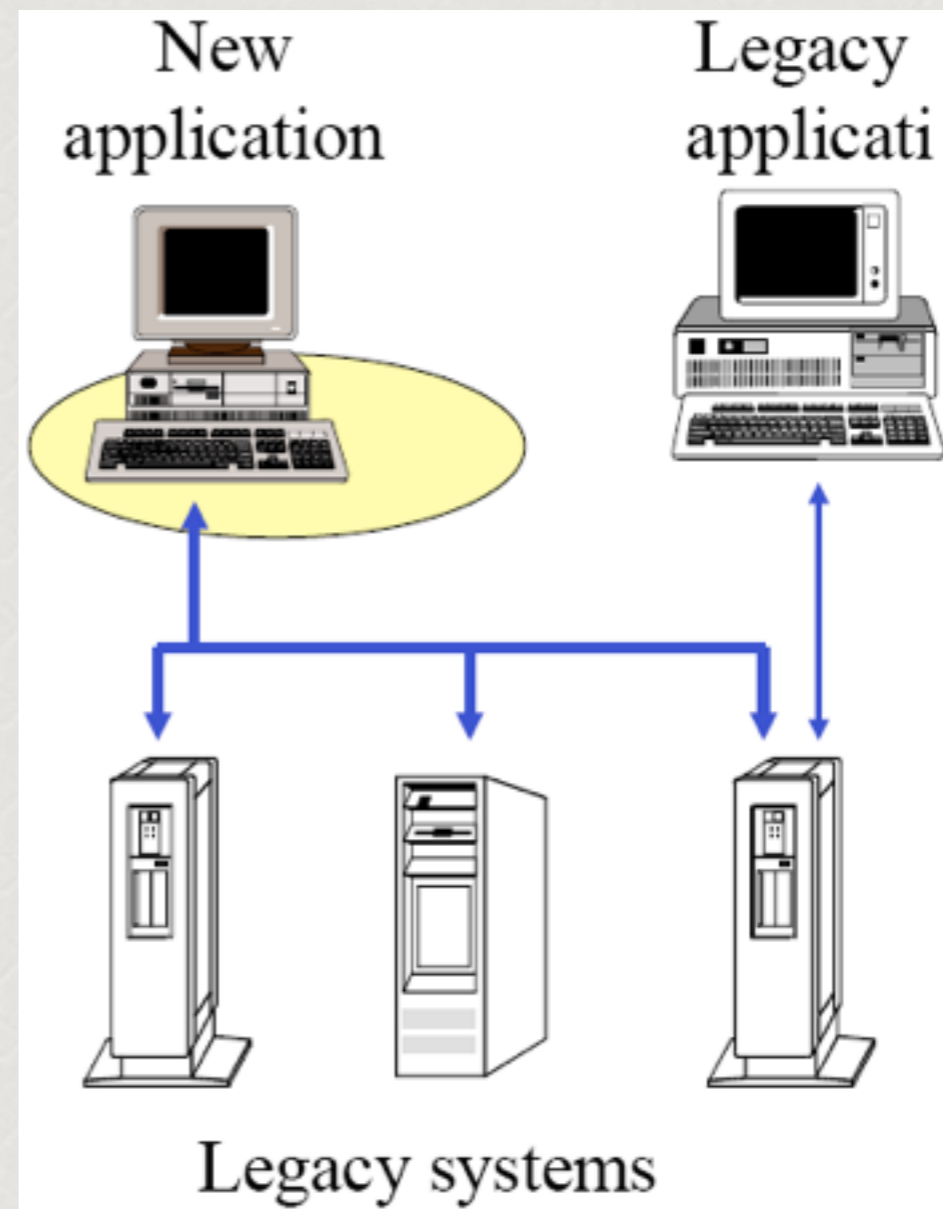


Characteristics of Top-down Design

- ✦ **goals:** Focus first on the high-level goals, then proceed to define everything required to achieve those goals
- ✦ **tightly coupled:** To simplify system development/maintenance, distributed nodes are usually created to run on homogeneous computing environments. Functionality of one component depends on the functionality of other components.
- ✦ **more control:** easy to address both functional and non-functional (e.g., performance) issues
- ✦ **from-scratch-development:** few information systems nowadays can be developed this way

Information System Design: Legacy Systems

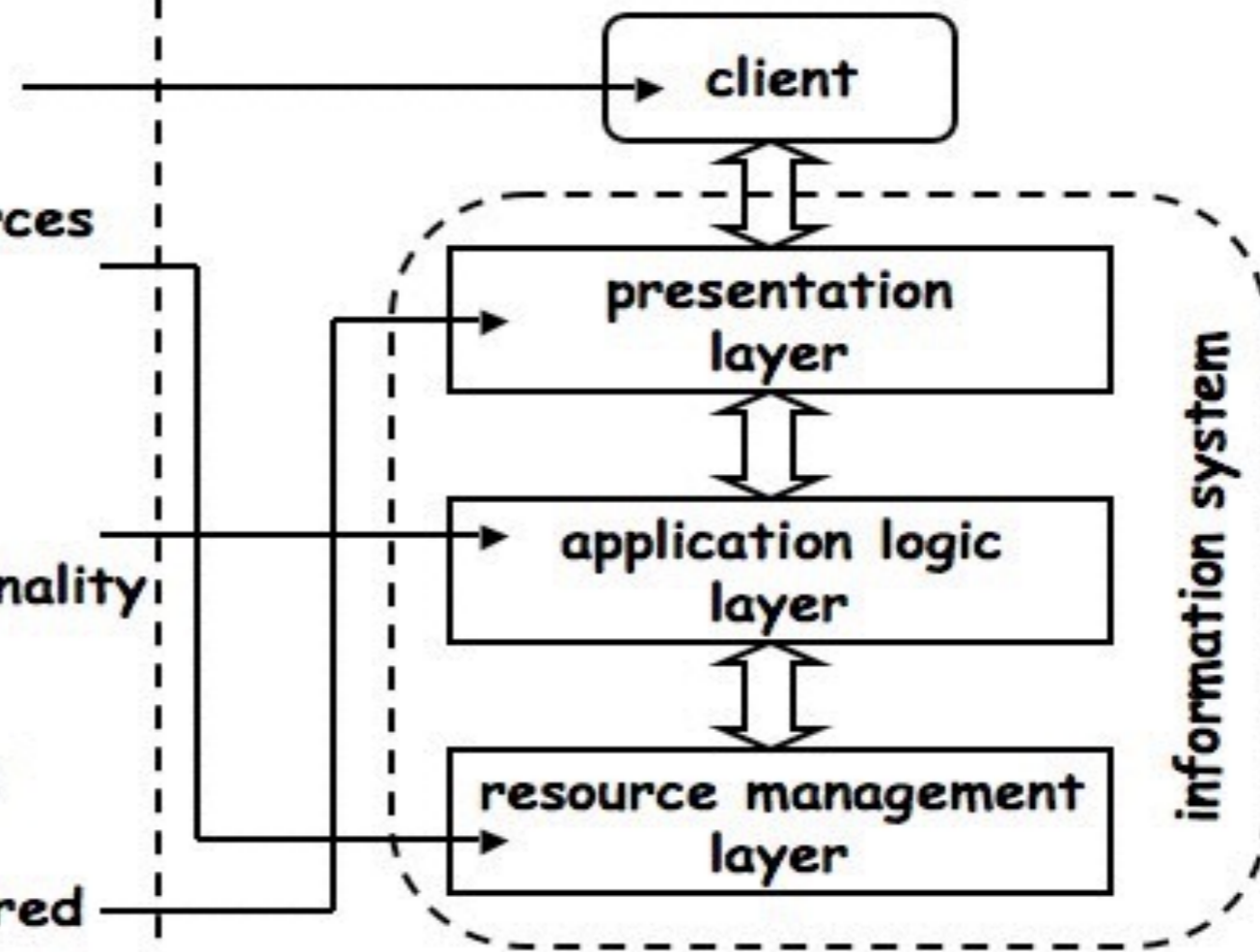
- ✦ *Legacy systems: a system that is to be used for purpose or context other than the one originally intended*
- ✦ *The functionality provided by legacy systems is predefined and cannot be modified*
- ✦ *The design is driven by characteristics of the lower layers*



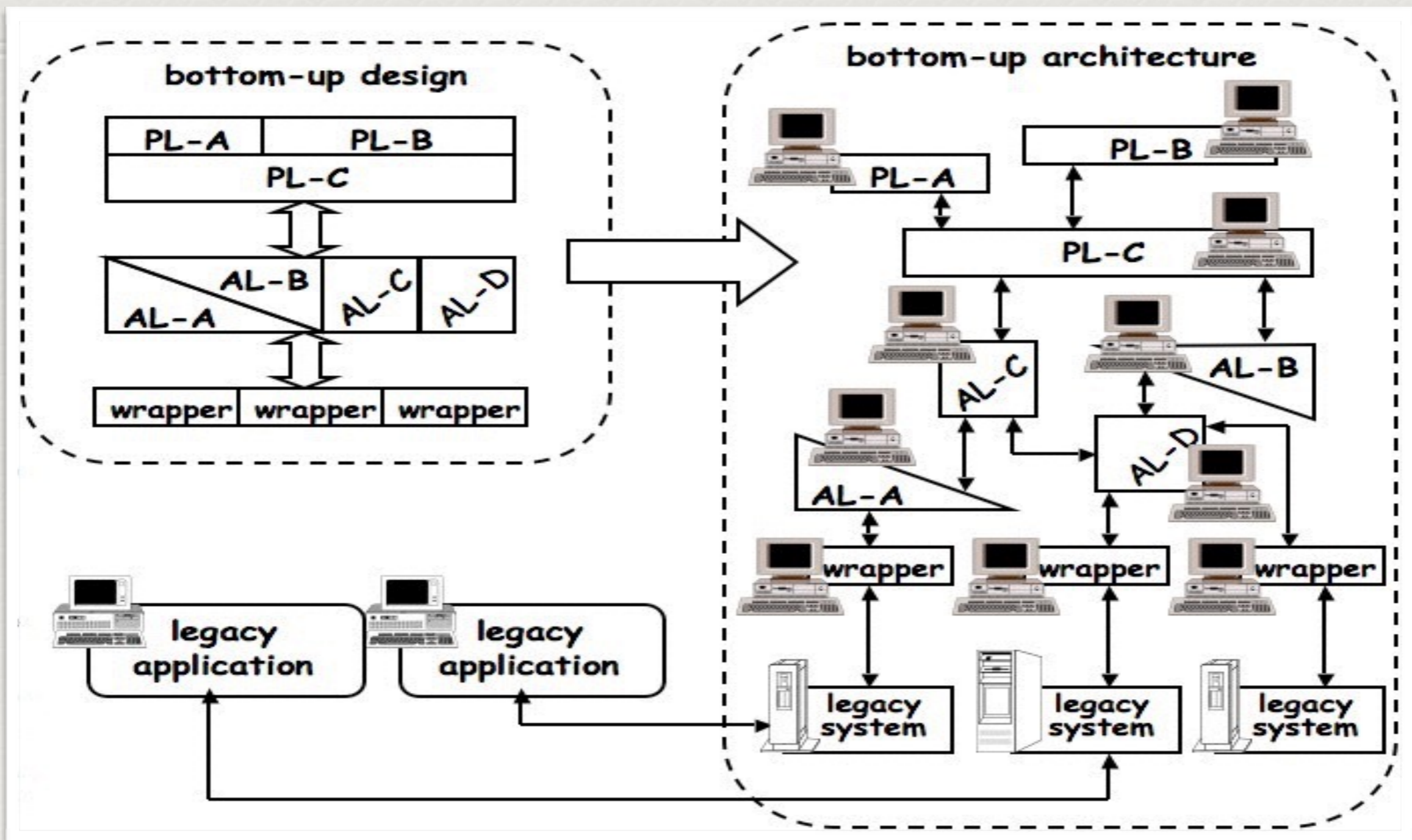
Information System Design: Bottom-Up Design

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



Information System Design: Bottom-Up Design



Characteristics of Bottom-Up Design

- ✦ **legacy systems:** *In a bottom up design, many of the basic components already exist. These are stand-alone systems which need to be integrated into a new system.*
- ✦ **loosely coupled:** *The components do not necessarily cease to work as stand-alone components. Often old applications continue running at the same time as new applications.*
- ✦ **wider usage:** *This approach is used widely because legacy systems exist and typically cannot be easily replaced.*

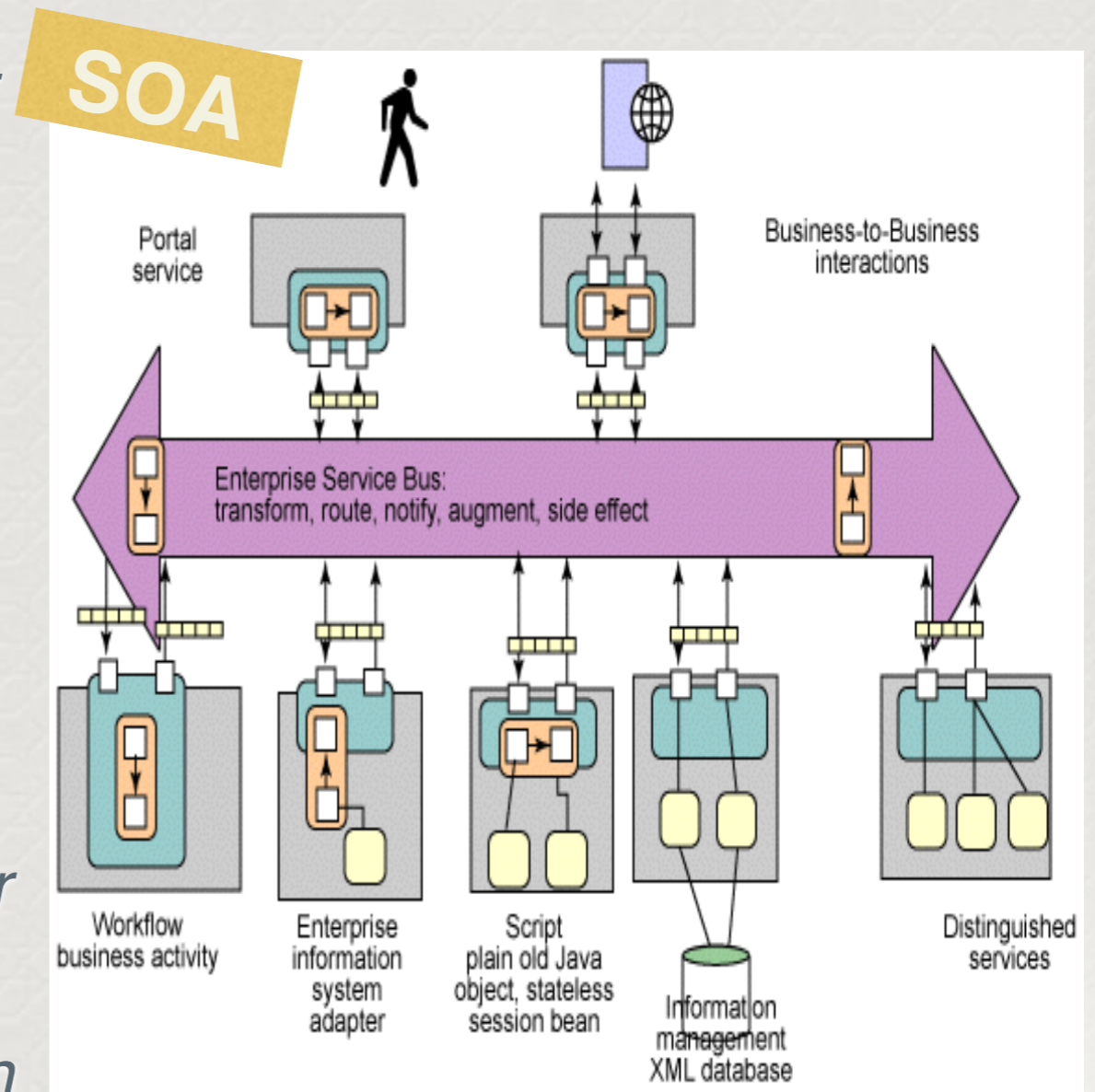
Information System Design

- Two approaches: Top-down, Bottom-up

- Note for SOA (web services):

- Nearly without exception, most distributed information systems these days are the result of a bottom-up design

- The advantage of SOA lies in their ability to make bottom-up design simpler to implement and maintain

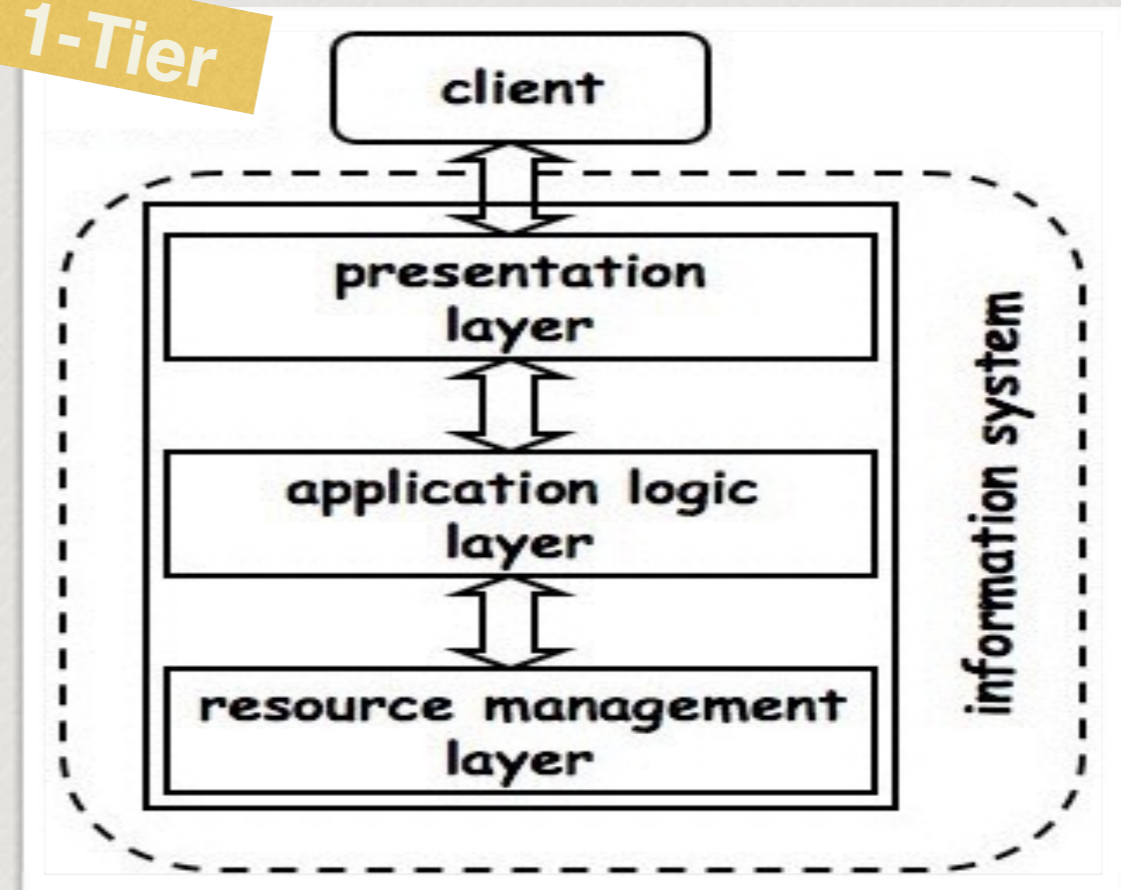


Architecture of an Information System

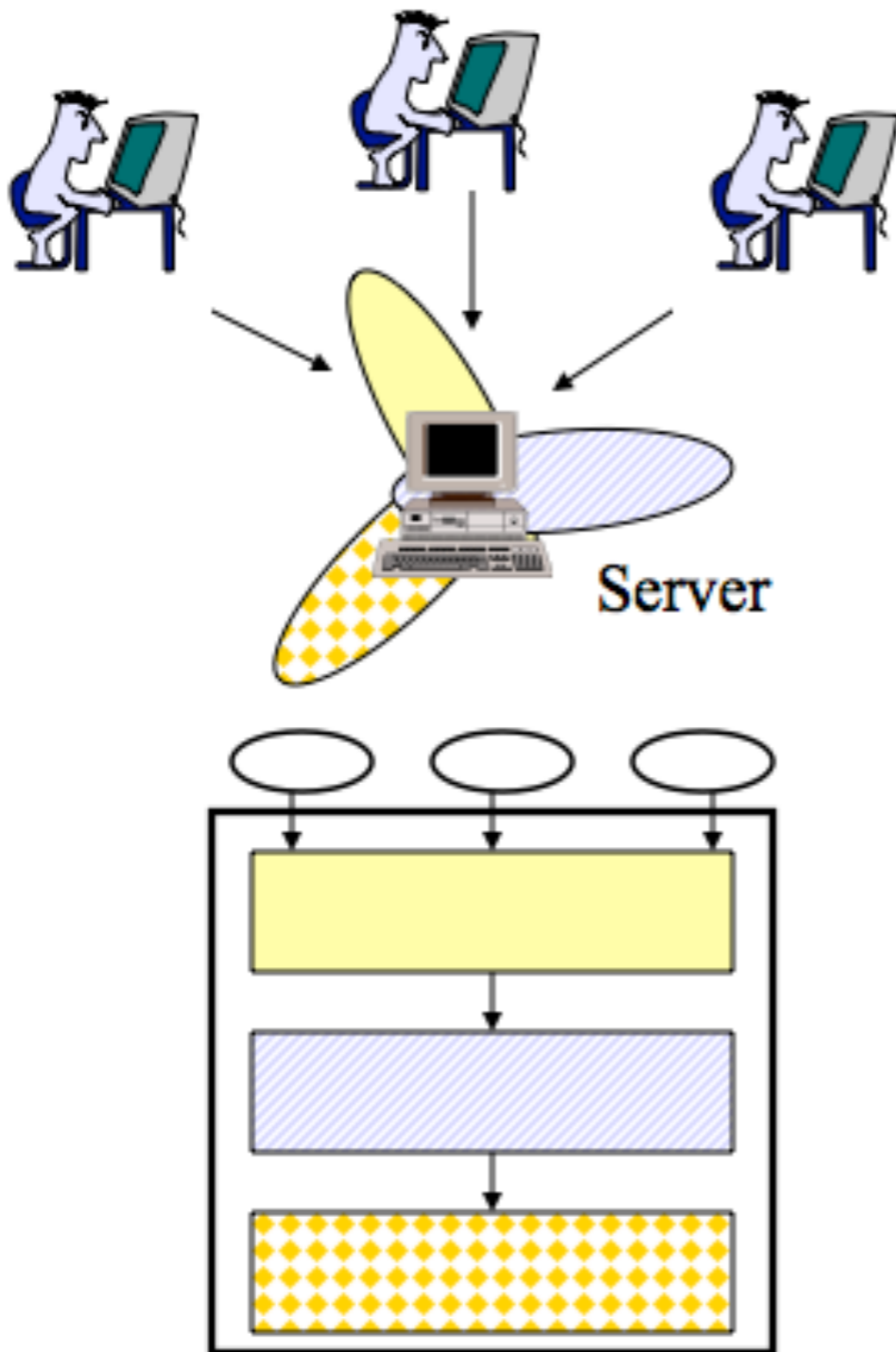
When the conceptual layers are implemented, they can be combined or distributed in different layers - forming 'Tiers'

- ✦ *users/programs access the system through "dumb" terminals, whose display is controlled by the information system (e.g., mainframes).*
- ✦ *(+) simple, highly optimised, centralised, no deployment or portability issue*
- ✦ *(-) no entry point except the client terminals (hard to be integrated into other systems)*

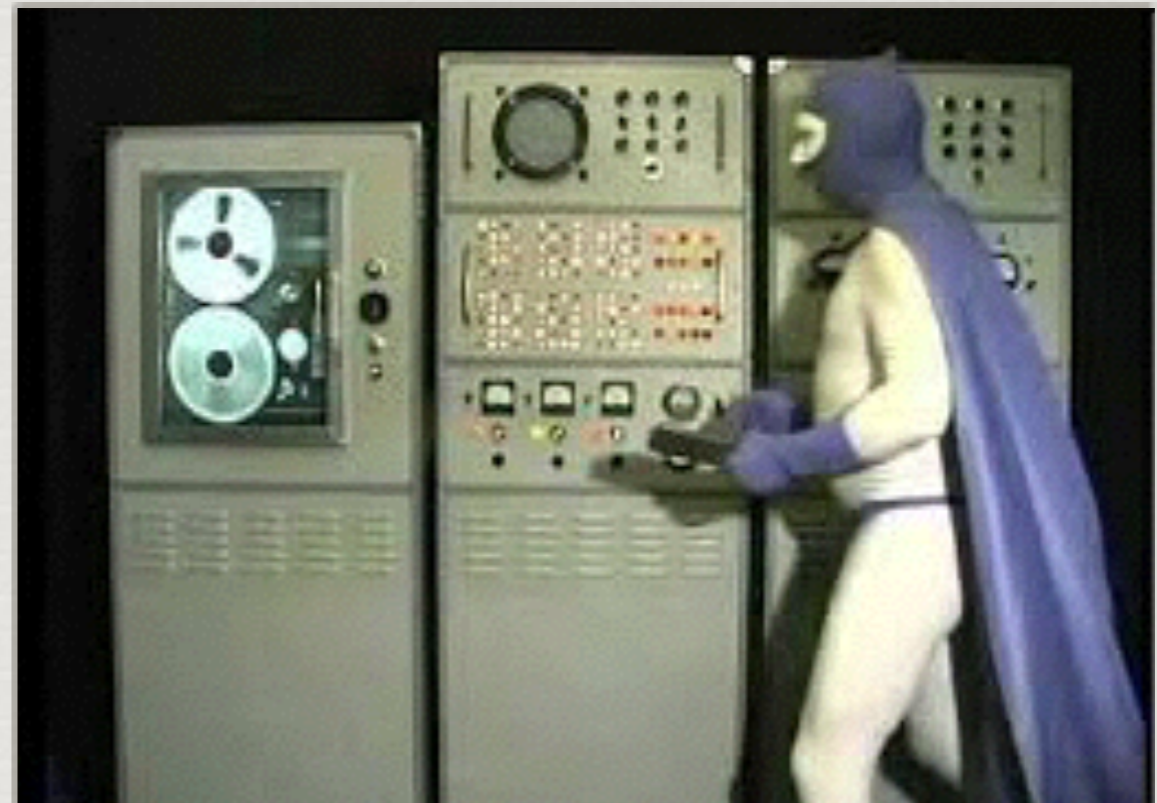
1-Tier



1-tier architecture



Users/programs access the system through terminals but what is displayed and how it appears is controlled by the server.

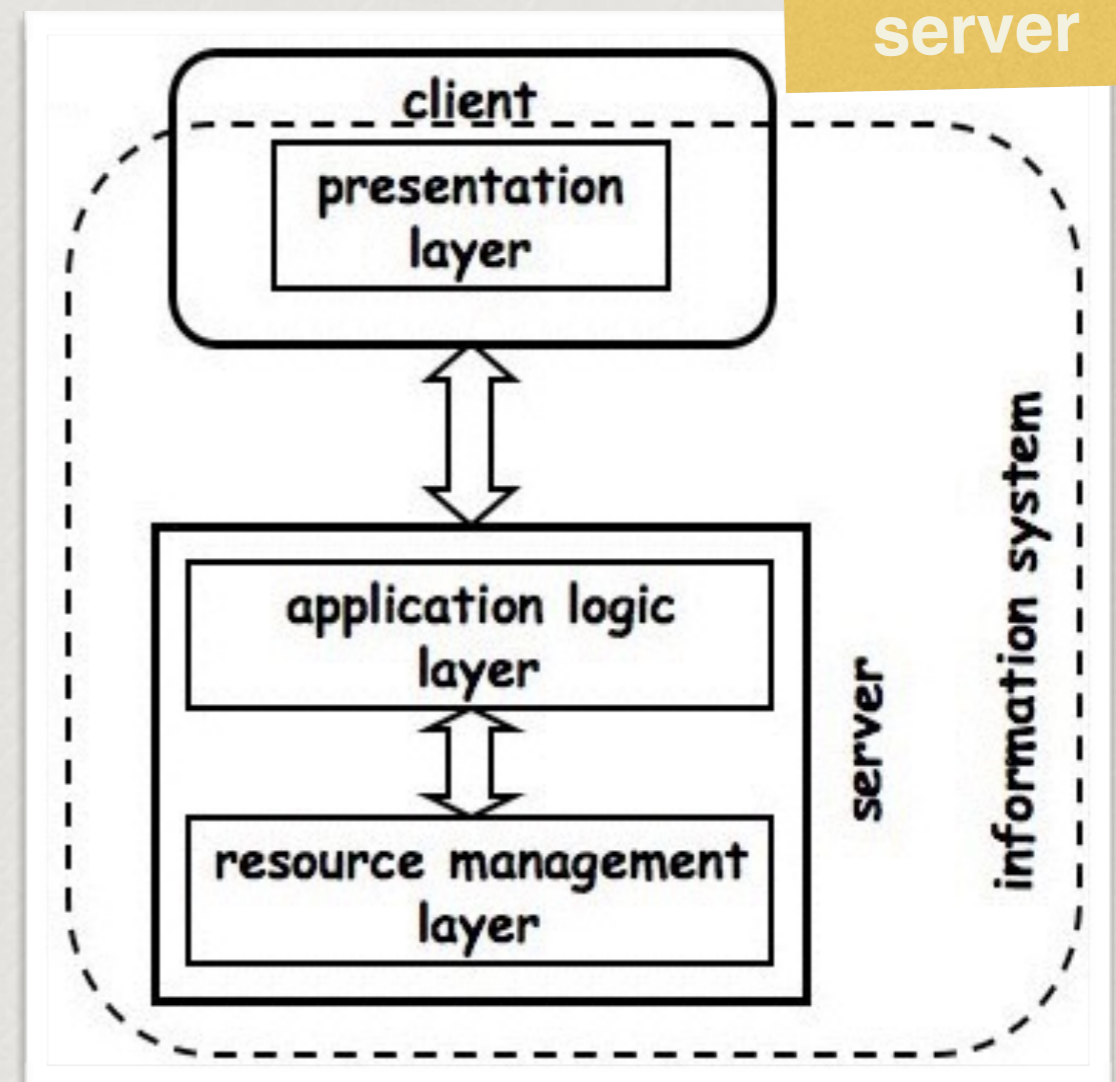


Architecture of an Information System

on the back of increasing client computing power (eg., PC), development of Local Area Network, etc.

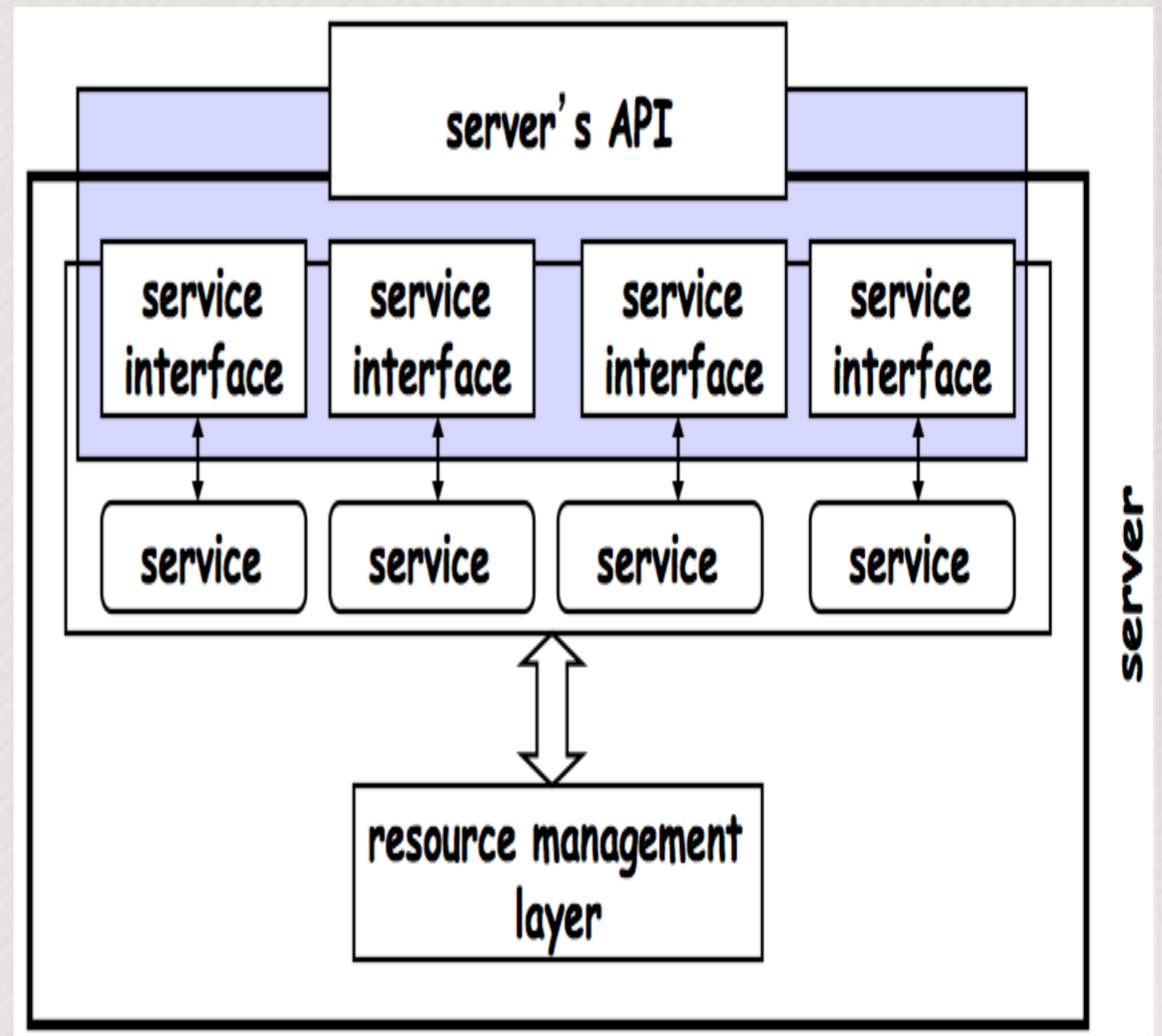
- ✦ *client can have more sophisticated presentation layers while also saving computer resources on the server.*
- ✦ *The resource manager still only sees one client: the application logic. This helps with performance since there are no client connections/sessions*
- ✦ *thin or fat clients, depending on the range of functionality client handles*

**2-Tier:
client/
server**



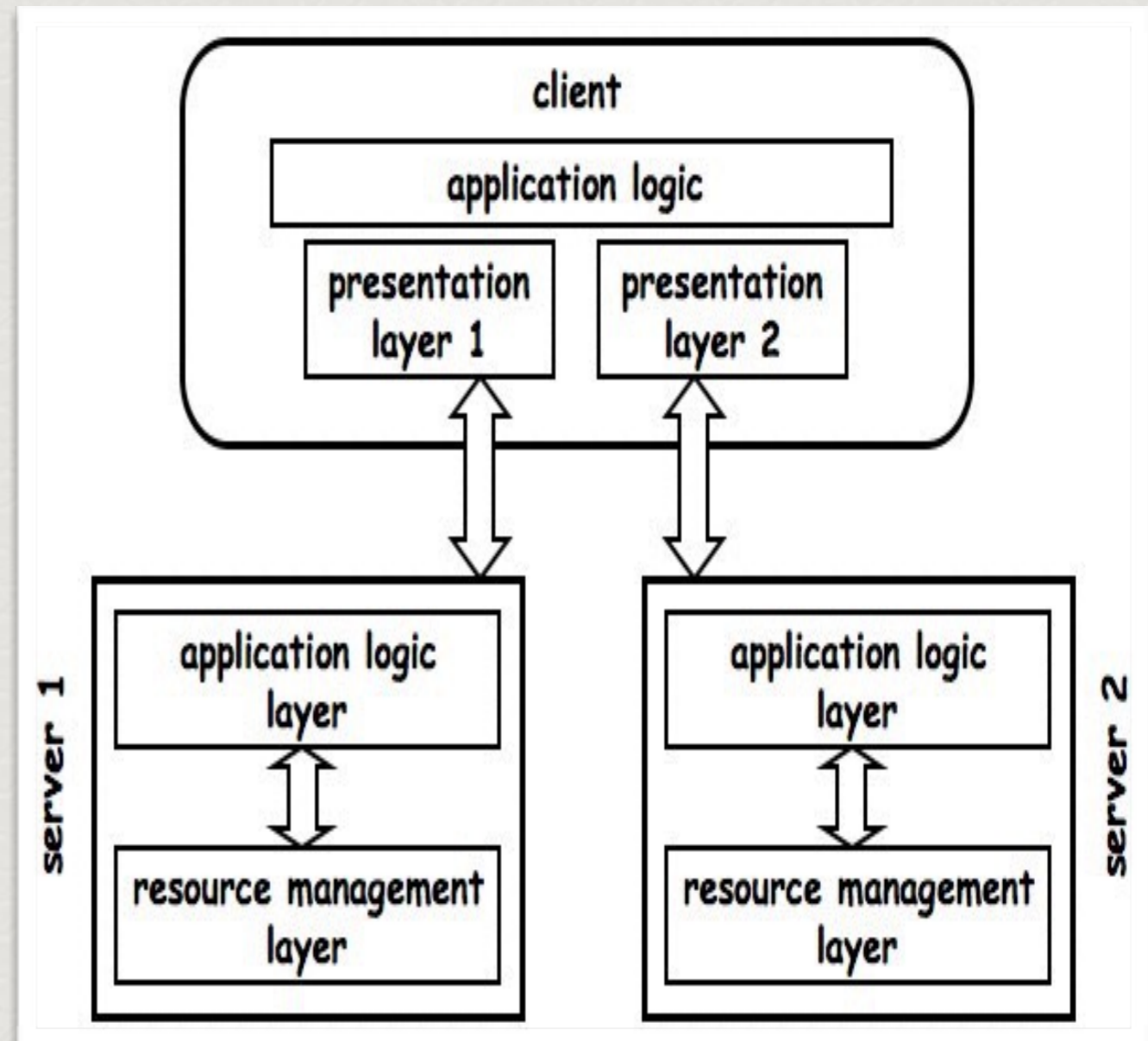
Key developments in distributed systems by 2-Tier

- ✦ *the notion of “service” (i.e., the client invokes a service implemented by the server)*
- ✦ *the notion of public service interface (how the client can invoke a given service)*
- ✦ *The concept of API -> can support diverse clients, change/evolve the server without affecting the clients.*



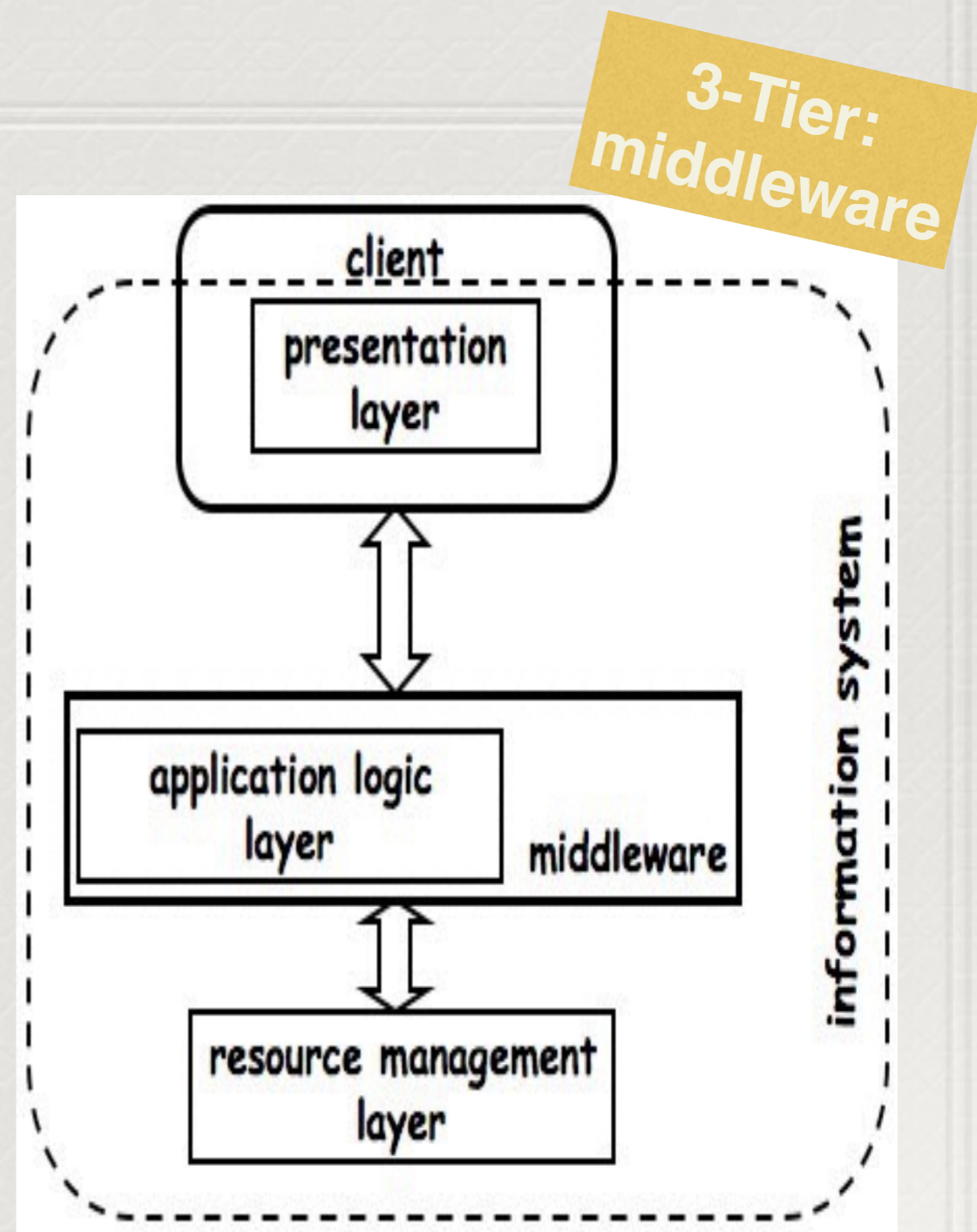
Problems with 2-Tier (in terms of integration)

- *For a client to be integrated to different servers, it needs to understand the API of each server*
- *Since the underlying servers do not know each other, the client must combine data from both servers, deal with exceptions and failures of the servers, coordinate the access to the servers and so on ...*
- *Client gets bigger and bigger and bigger ...*

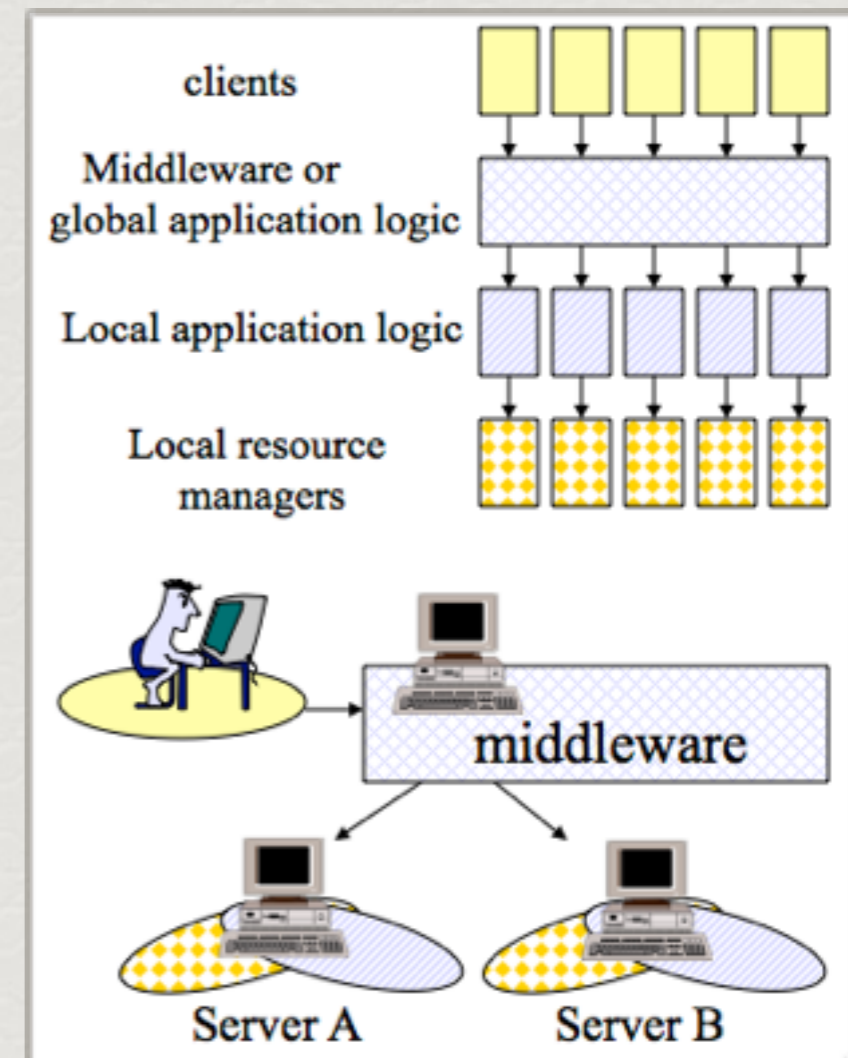
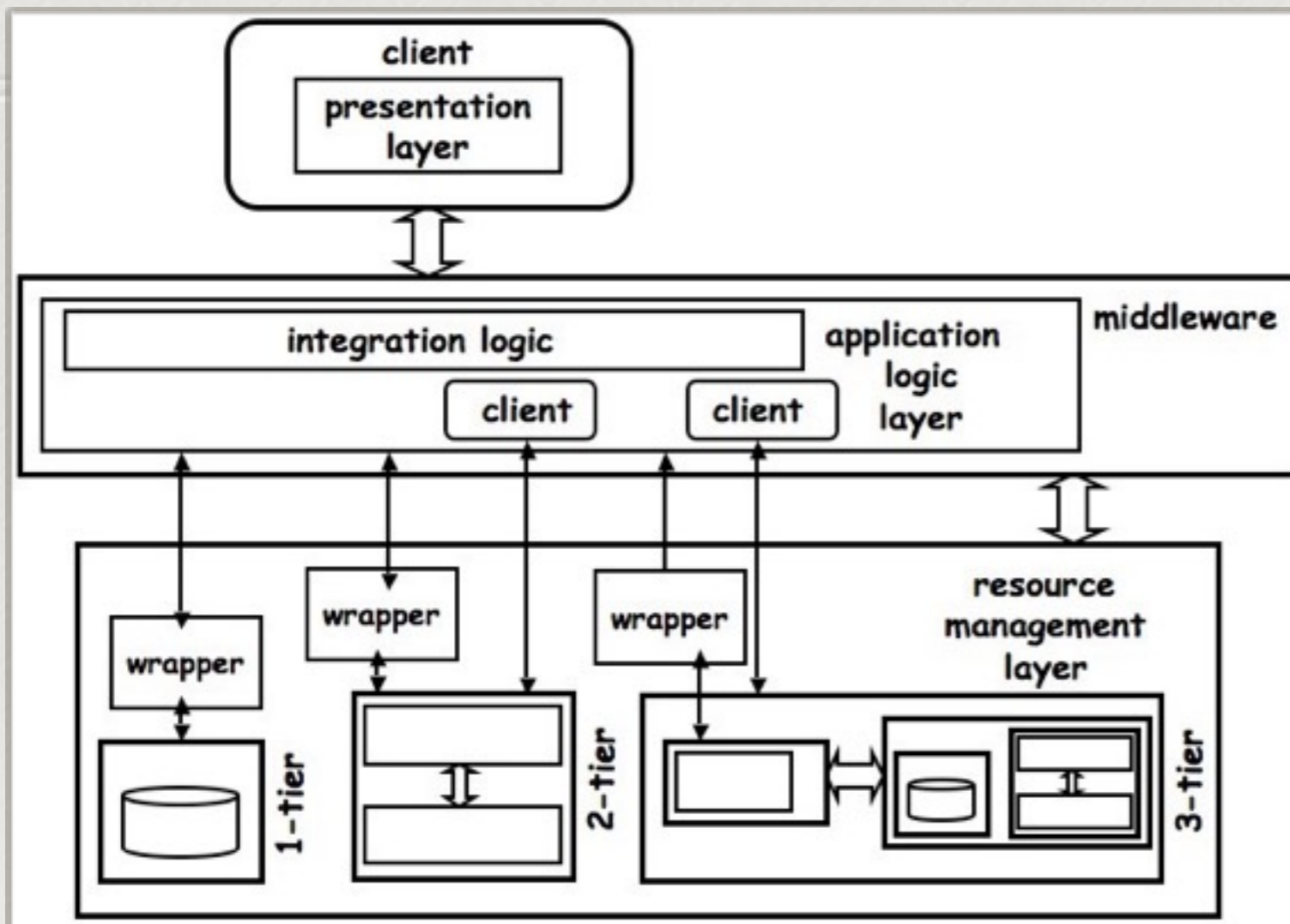


Architecture of an Information System

- *Added - application logic layer (aka middleware)*
- *introduces an additional layer of business logic encompassing all underlying systems*
- *By doing this, a middleware system:*
 - *simplifies the design of the clients by reducing the number of interfaces,*
 - *provides transparent access to the underlying systems,*
 - *takes care of locating resources, accessing them, and gathering results.*



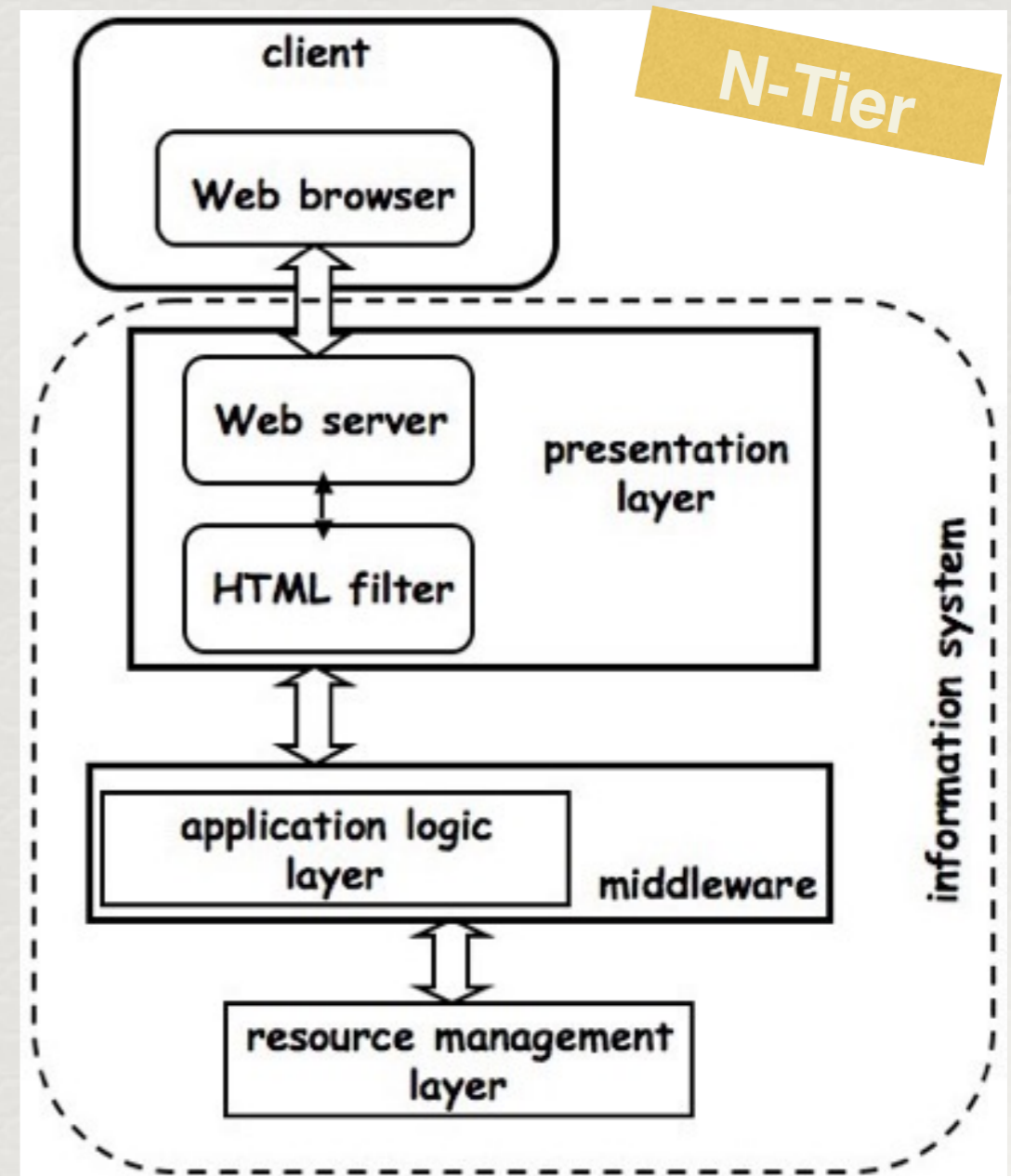
The Middleware in 3-Tier



It enables transparent access to the underlying systems, the integration of systems built using other architectures

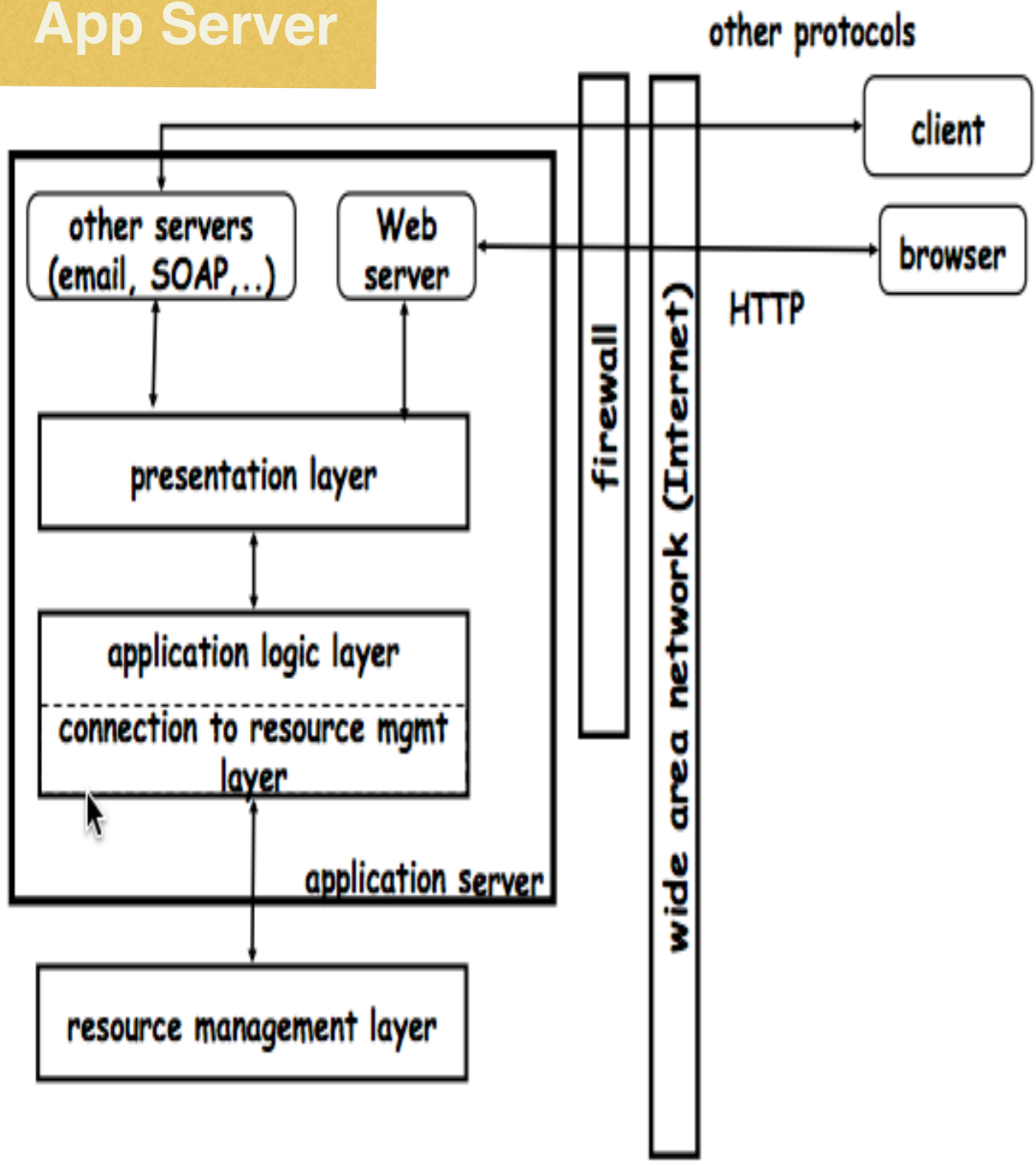
Architecture of an Information System

- ✦ *created by either connecting several 3-tier systems and/or by adding a Web layer*
- ✦ *normally, web layer is incorporated into a presentation layer that resides on the server side (part of the middleware infrastructure)*
- ✦ *The addition of the Web layer led to the notion of “**application servers**” which was used to refer to middleware platforms supporting Web access*

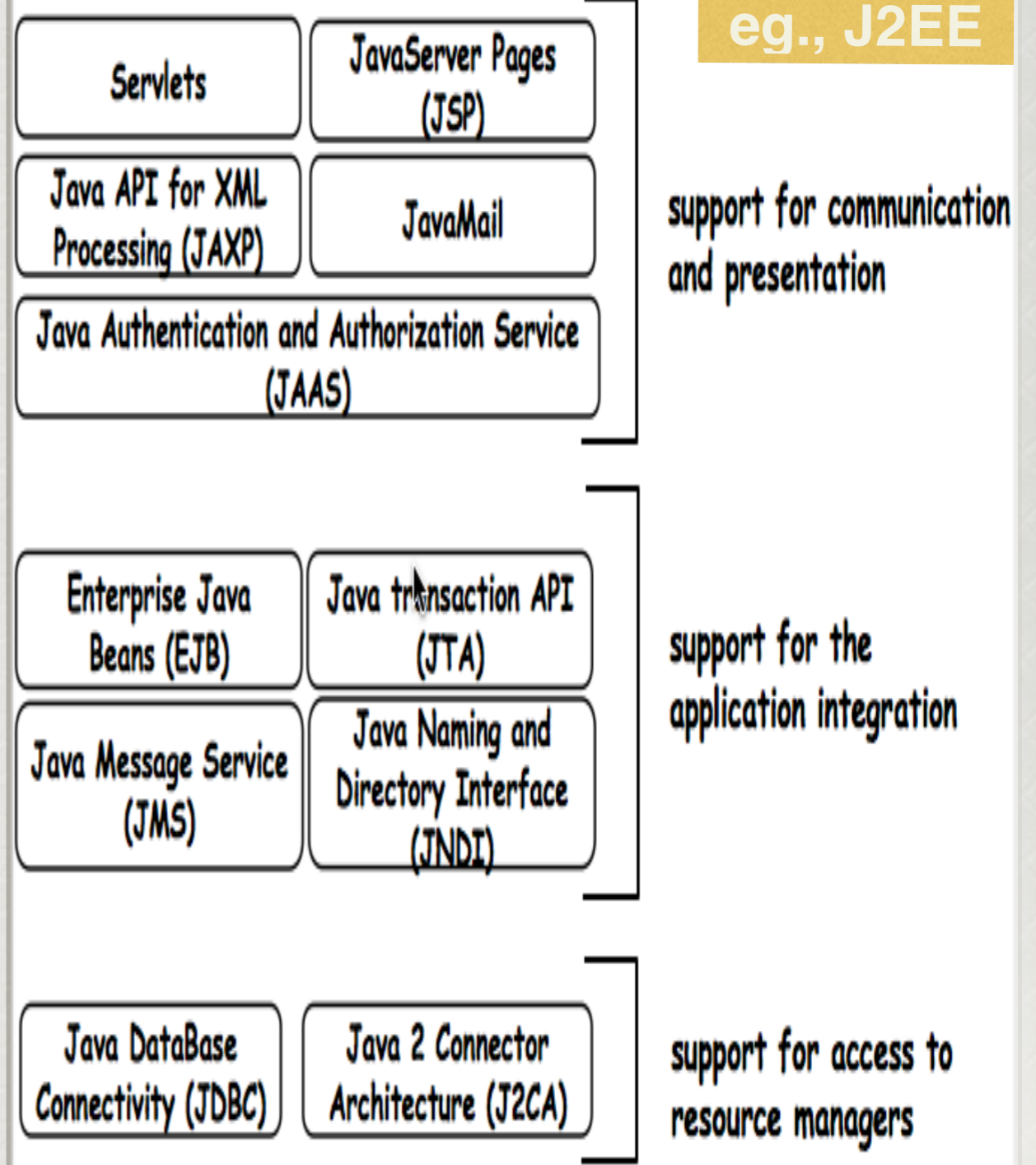


Application servers: a middleware platform that provides support for Web access.

App Server

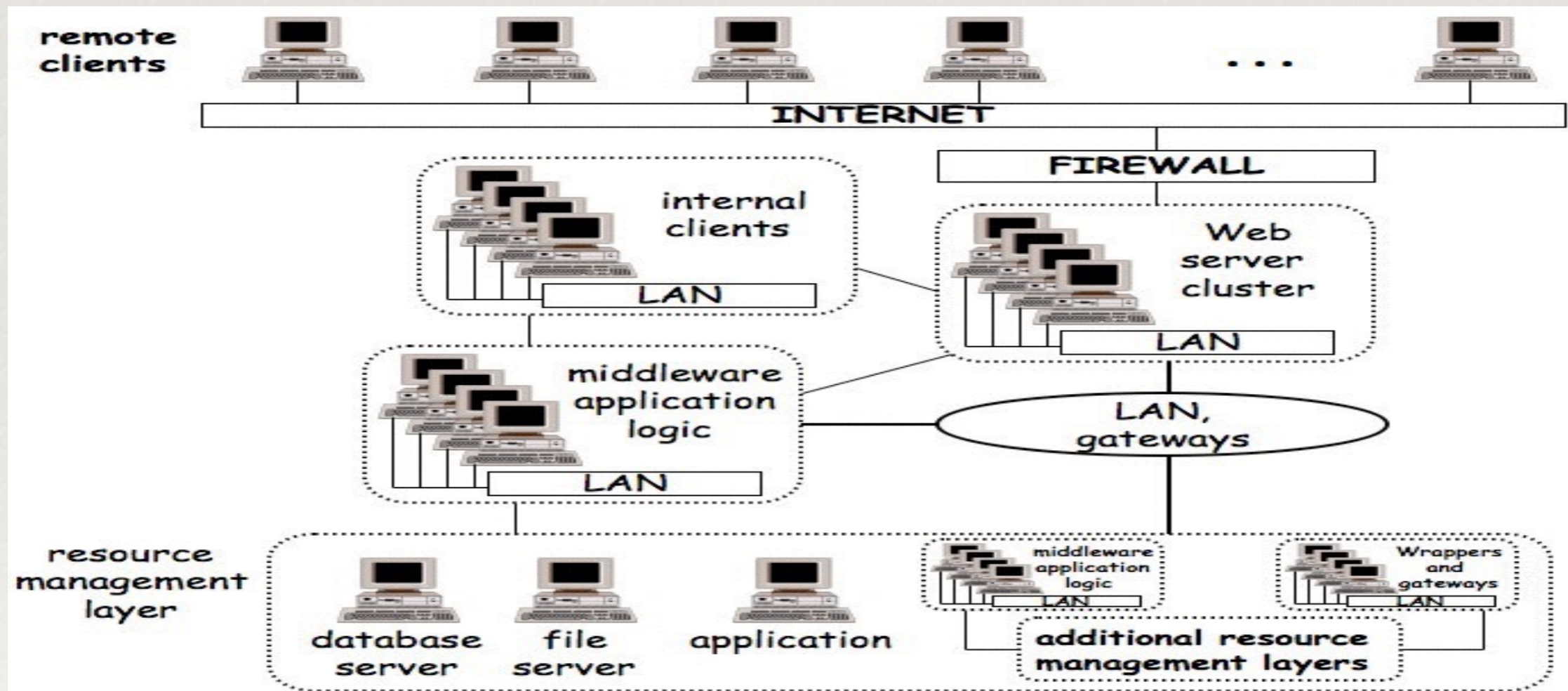


eg., J2EE

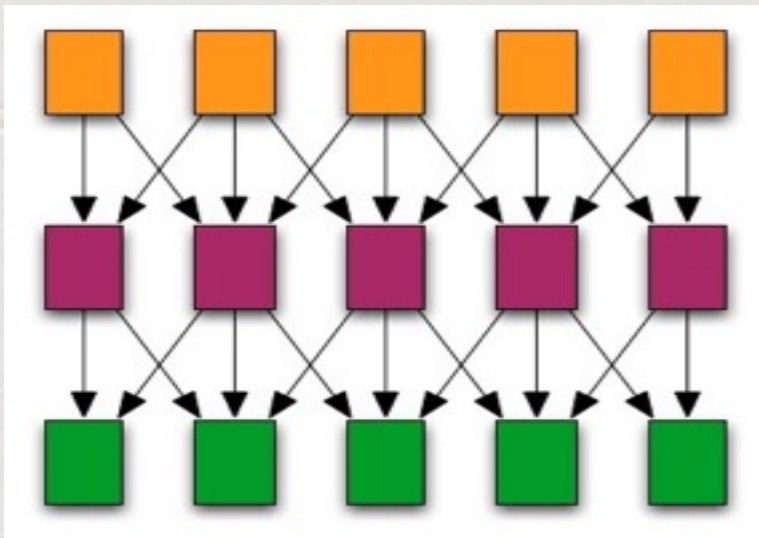


N-Tier Systems ...

Typically consist of a large collection of networks, gateways, individual computers, clusters of computers and links between systems ...



Game of Boxes and Arrow



There is no problem in system design that cannot be solved by adding a level of indirection. There is no performance problem that cannot be solved by removing a level of indirection

- ✿ *Each box represents a part of the system. Each arrow represents a connection between two parts of the system.*
- ✿ *The more boxes, the more modular the system: more opportunities for distribution and parallelism. This allows encapsulation, component based design, reuse.*
- ✿ *The more boxes, the more arrows: more sessions (connections) need to be maintained, more coordination is necessary. The system becomes more complex to monitor and manage.*
- ✿ *The more boxes, the greater the number of context switches and intermediate steps to go through before one gets to the data. Performance suffers considerably.*

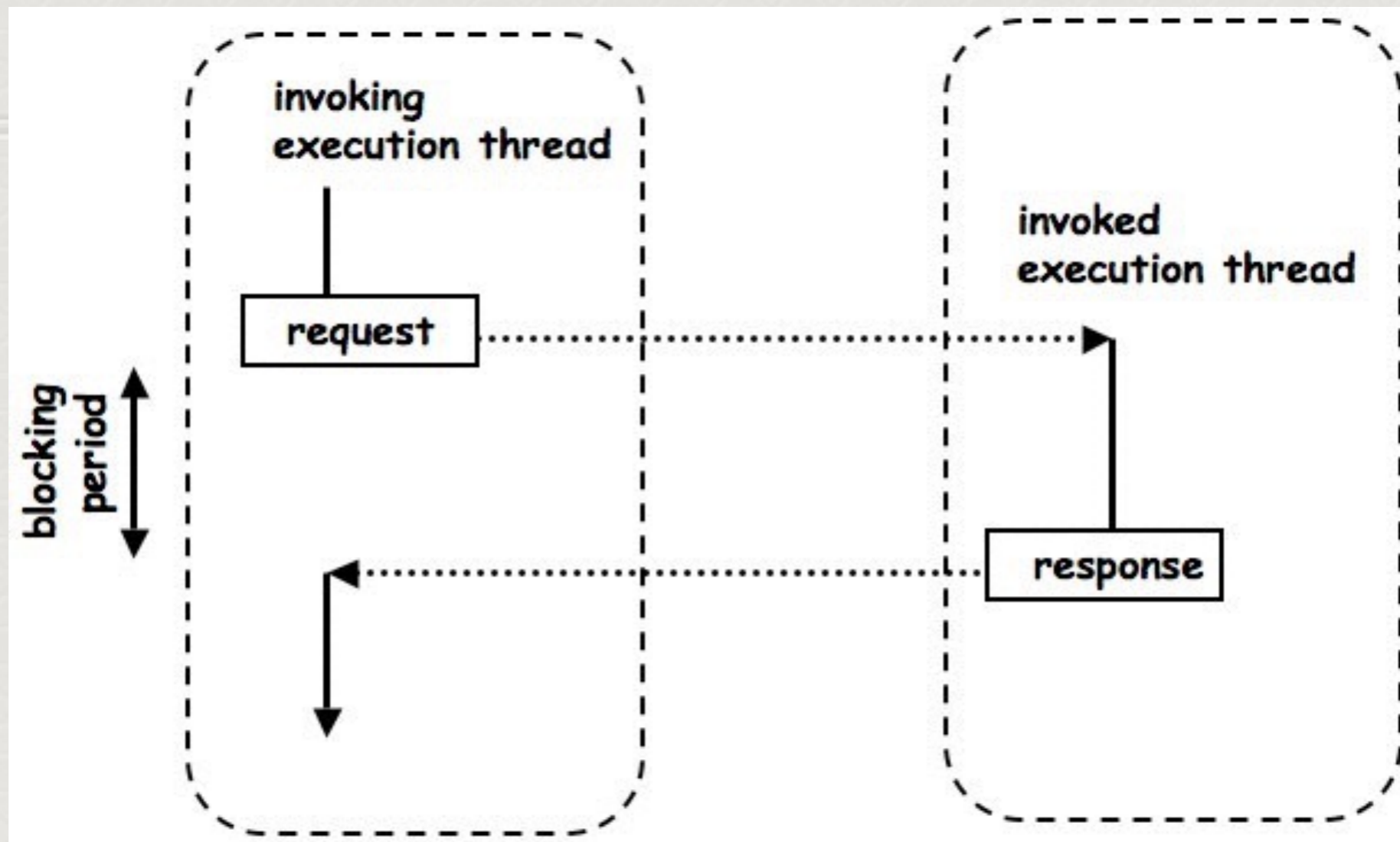
Architecture of an Information System

- ✦ *Four different architecture: 1, 2, 3 and N-Tier*
 - ✦ *2-Tier: introduced key developments in software design concepts (such as API, server-side services)*
 - ✦ *3-Tier: all about middleware (often refer to as “integration layer”)*
 - ✦ *N-Tier: could be 3-Tier plus Web-enablement layer or integration of many 3-Tier systems (modern application servers)*
- ✦ *Something to note for Web services:*
 - ✦ *More tiers, more boxes and more arrows -> increased complexity*
 - ✦ *Web services aim to reduce the complexity in 3-Tier/N-Tier architectures.*
 - ✦ *Web services add a new tier to middleware (integration layer) which is commonly understood by all parties (i.e., major standardisation effort)*

Communication in a Information System

- ✦ *When we separate layers and tiers in an information system, we assume that there is some form of communication between all these elements.*
- ✦ *There are two communication patterns widely used:*
 - ✦ *synchronous: blocking interaction*
 - ✦ *asynchronous: non blocking interaction*

Communication in an Information System



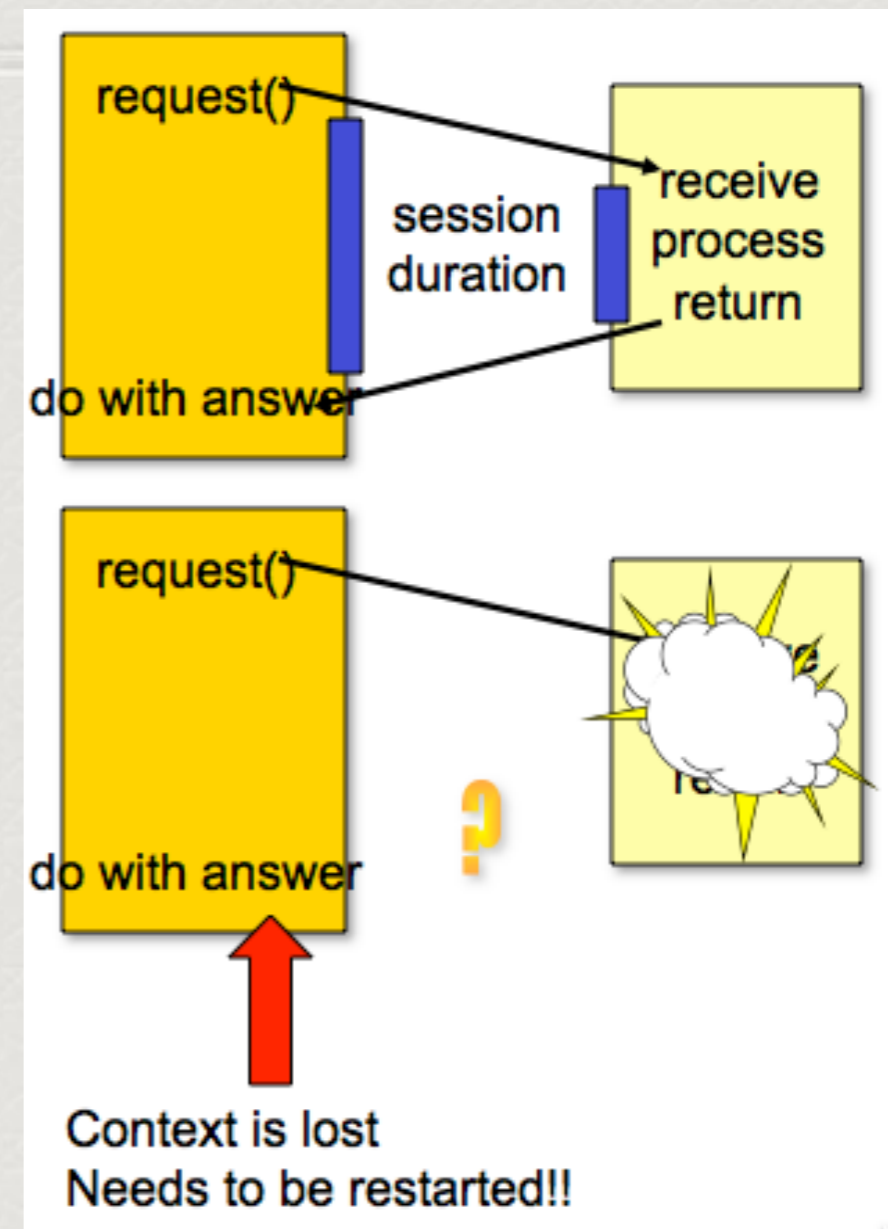
blocking calls (client waits while server processes a request)

Characteristics of Blocking Calls

- ✦ (+) *simple to understand and implement*
 - ✦ *logically easier to understand as the code follow natural organisation of procedures or method calls*
 - ✦ *easier to debug (e.g., strong correlation between the code that makes the call and the code that deals with the response)*
 - ✦ *common and widely used in traditional middleware*
 - ✦ *typical request-response type interactions*
- ✦ (-) *expensive and waste of resources*
 - ✦ *connection overhead (new connection for each request)*
 - ✦ *calling thread **must** wait*
 - ✦ *synchronous interaction requires both parties to be “on-line” -> higher probability of failures,*
 - ✦ *not suitable if the number of tiers increases*

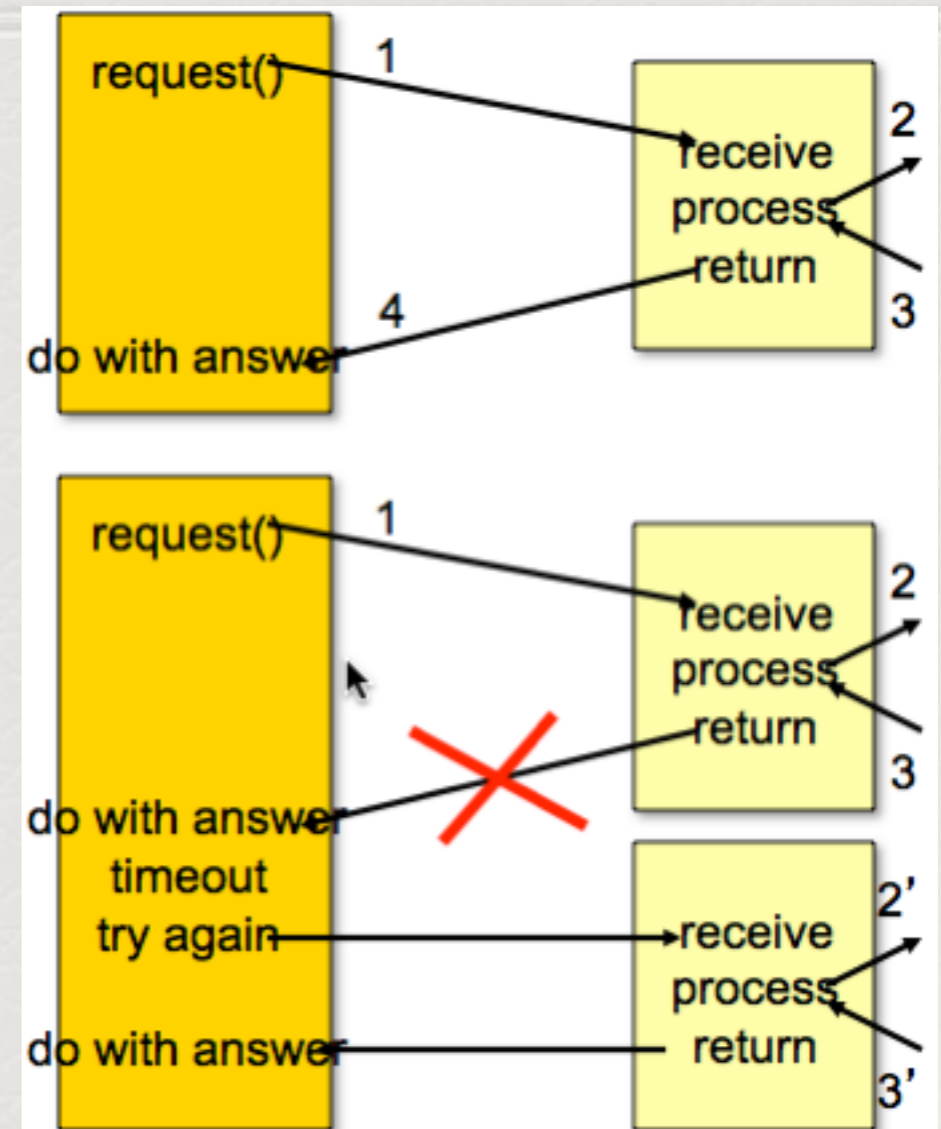
Overhead of Synchronisation ...

- ✦ *Synchronous invocations require to maintain a session between the caller and the receiver.*
- ✦ *Maintaining sessions is expensive and consumes CPU resources. For this reason, client/server systems often resort to connection pooling to optimise resource utilisation*
 - ✦ *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*



What to do when it fails ...?

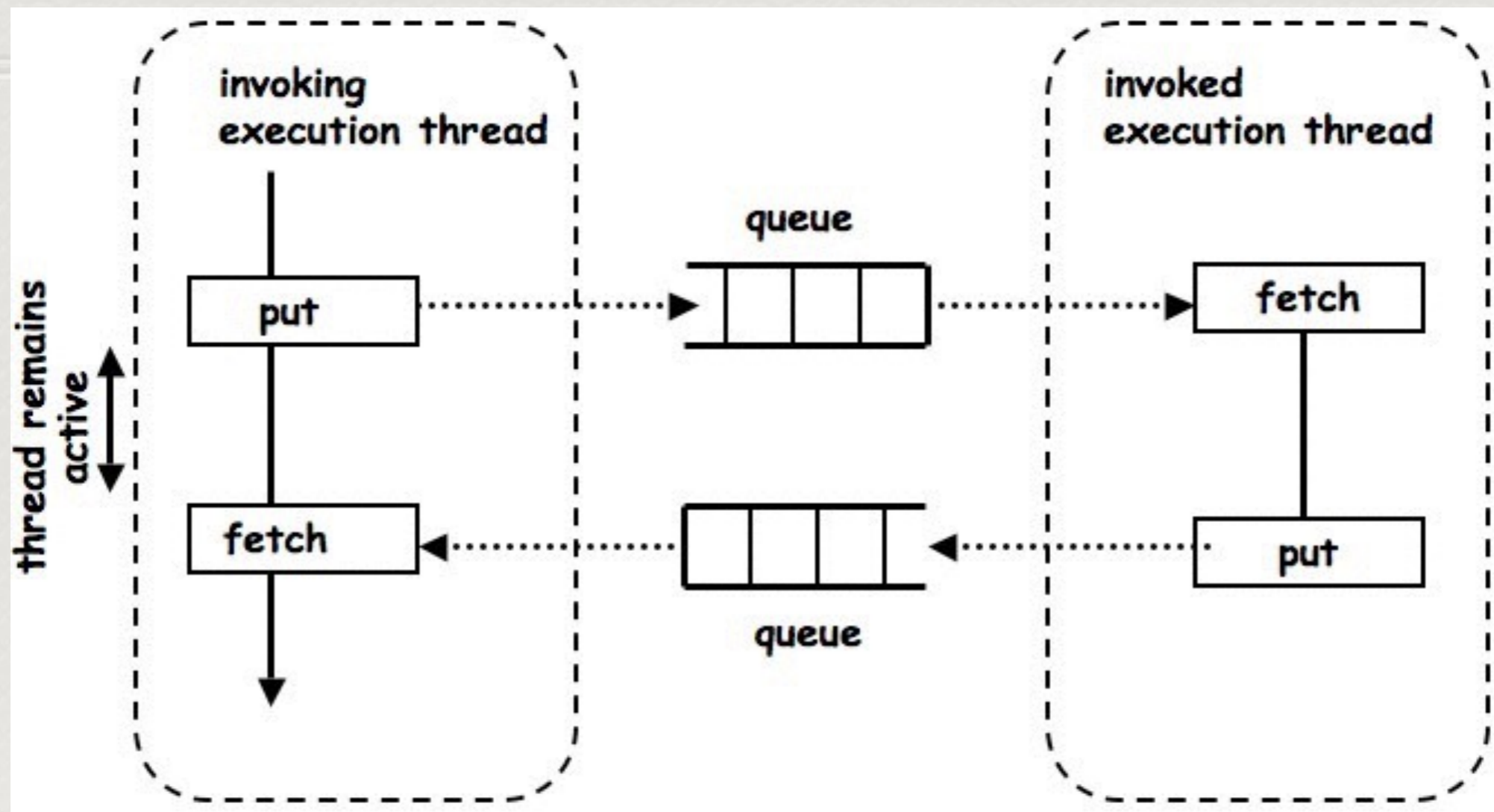
- ✦ *If the client or the server fail, the context is lost and resynchronisation might be difficult.*
- ✦ *failure occurred before 1, nothing has happened*
- ✦ *failure occurs after 1 but before 2 (receiver crashes), then the request is lost*
- ✦ *failure happens after 2 but before 3, side effects may cause inconsistencies*
- ✦ *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

- ✦ *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*

Communication in an Information System



non blocking calls (queues) allow the caller to continue working while the request is processed.

Characteristics of Non Blocking Calls

- ✿ *a call to the server returns immediately*
- ✿ *client can continue to run and occasionally check with server to see if a response is ready*
- ✿ *typically implemented via message queues*
 - ✿ *(-) adds complexity to client architecture*
 - ✿ *(+) more modular (less dependancy between communicating parties),*
- ✿ *more natural way to implement complex interactions between heterogeneous systems*
- ✿ *more suitable for non request-response type communications (e.g., multicast, publish/subscribe)*

- Can you name two approaches to information system design?
- Can you identify the layers in information system?
- When implemented, the conceptual layers are combined/distributed in many ways and form (.....)?
- Can you name a key concept born out of 2-Tier architecture?
- Where is middleware in the 3-Tier architecture?
- Can you give me an example of non blocking communication pattern?

Summary

From Next Week ...

*Let's have a look at the course schedule
again.*