COMP9321:
Data services engineering

Semester 1, 2018,

Week 1 Introduction to the course

By Helen Paik, CSE UNSW

Who's who in COMP9321

- Lecturer-in-Charge (LiC)
  - Helen Paik (h.paik@unsw.edu.au)
  - Office: K17 401A, Ext: 54095
- Course Administrator
  - Mohammad Ali Yaghub Zade Fard (CSE PhD student)
  - (m.yaghoubzadehfard@unsw.edu.au)
- Tutors
  - Mohammad Ali Yaghub Zade Fard (CSE PhD student)
  - Alireza Tabebordbar (CSE PhD student)
  - Daniel Ryu (PhD, CSE Adjunct Lecturer)
- Course Web site
  - http://www.cse.unsw.edu.au/~cs9321

COMP9321: Data services engineering

Previously known as Web applications engineering

What was taught and why needed revision

Many Web apps are now data-oriented or utilise data heavily.

-functionality requires combining or processing complex data from multiple sources

This course is not about big data nor cutting_edge data analytics

How to make the design and implementation of data-oriented application easy(a approach/techinique)

Data-oriented applications …

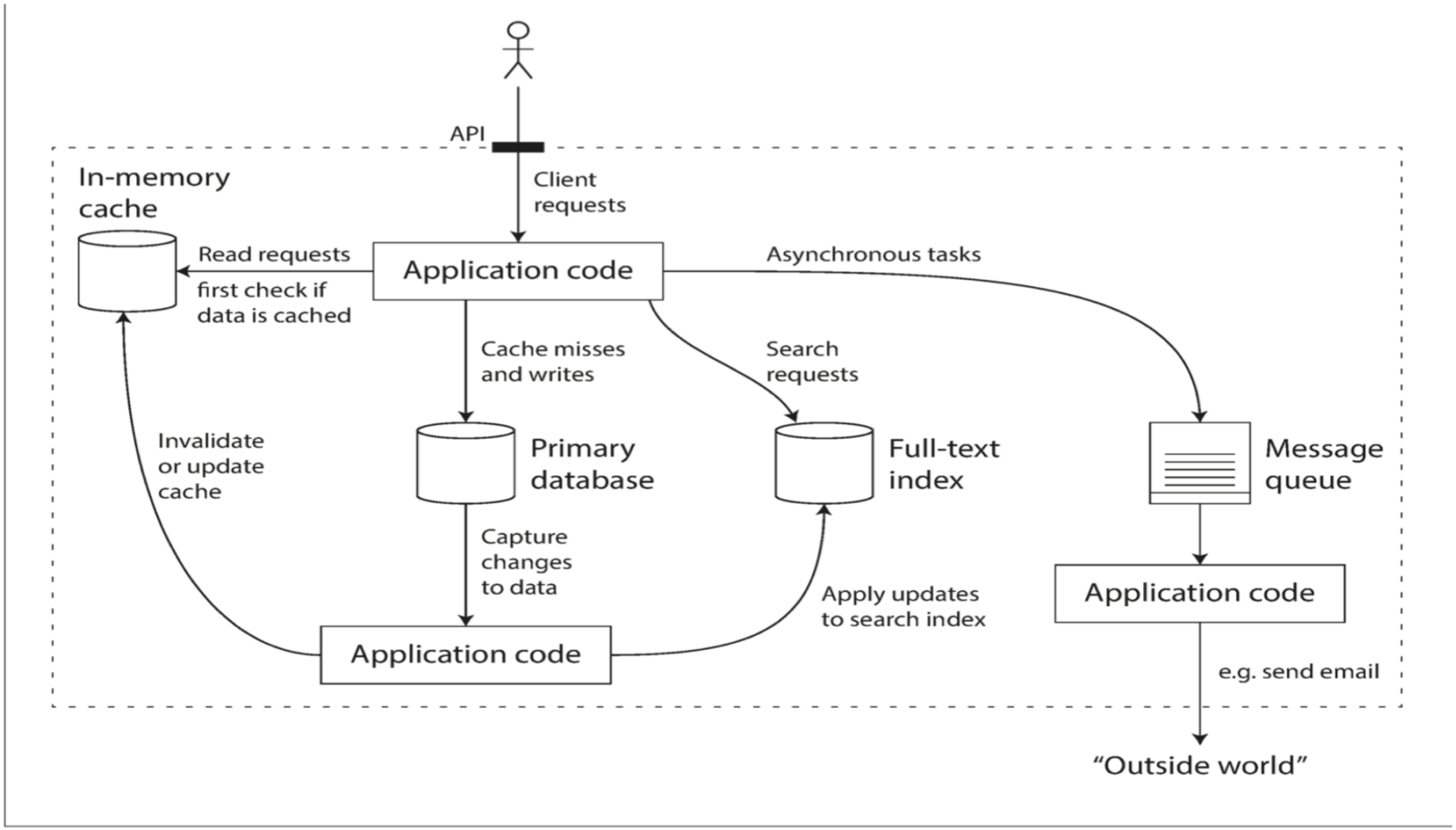


Figure 1-1. One possible architecture for a data system that combines several components.

Picture from Designing Data-Intensive Applications

Data-oriented applications …

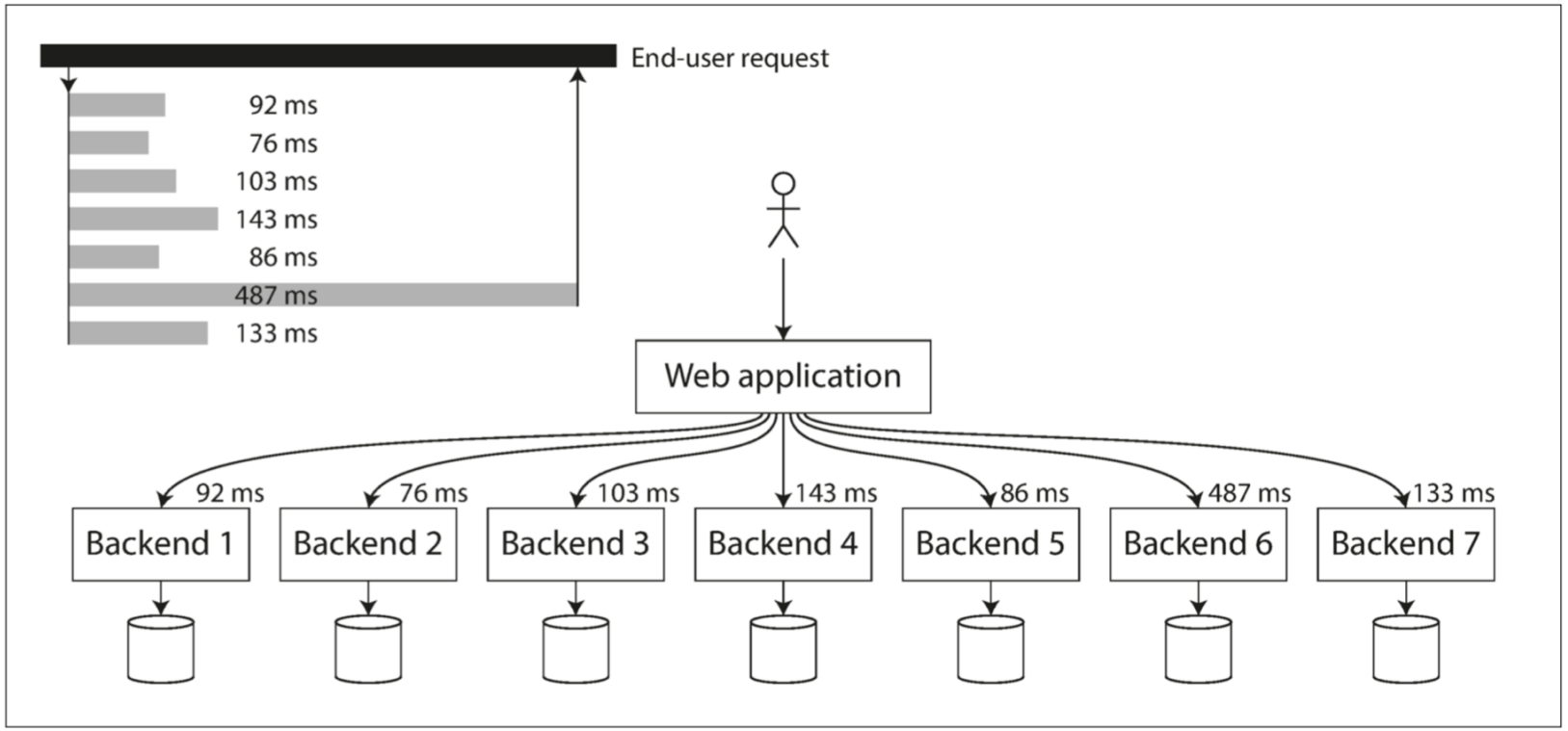


Figure 1-5. When several backend calls are needed to serve a request, it takes just a single slow backend request to slow down the entire end-user request.

Picture from Designing Data-Intensive Applications

So what is this course about?

Data Services Engineering

Data = is the problem we want to deal with, understanding the problems and possible ways to “get” data and “publish” data, how to discover or manage multiple data sources, etc.

Services = is the proposed solution/design approach to make our problem “manageable”

Engineering = (best practices, weighing options, we will think about these all throughout, at least try to) - obtain conceptual ideas as well as practical skills

In many ways, all these topics are related to the idea of “systems and integration of systems” and dealing with ‘heterogeneity’ of systems

异质性

Systems -> software applications/information systems

Heterogeneity at different levels: language, platform, schema (data, process)

Information Systems/Applications Integration

A set of services and solutions for bringing together disparate application and business processes as needed to meet the diverse information requirements of your customers, partners, suppliers and employees.

Motivations: Streamlining business operations, globalisation, competition, mergers and acquisition, new business models, technology development, etc. 兼并和收购

- e.g., merger of two companies (data + processes)

Problems: systems to be integrated are not homogeneous. 要整合的系统不是同质的。

- they are individually developed (ad-hoc) systems overtime
- some are “off-the-shelf” packages 现成的
- different execution platforms, technologies and business rules

Data Level Integration …

Data integration = combining data from different sources and providing users with a unified view over them

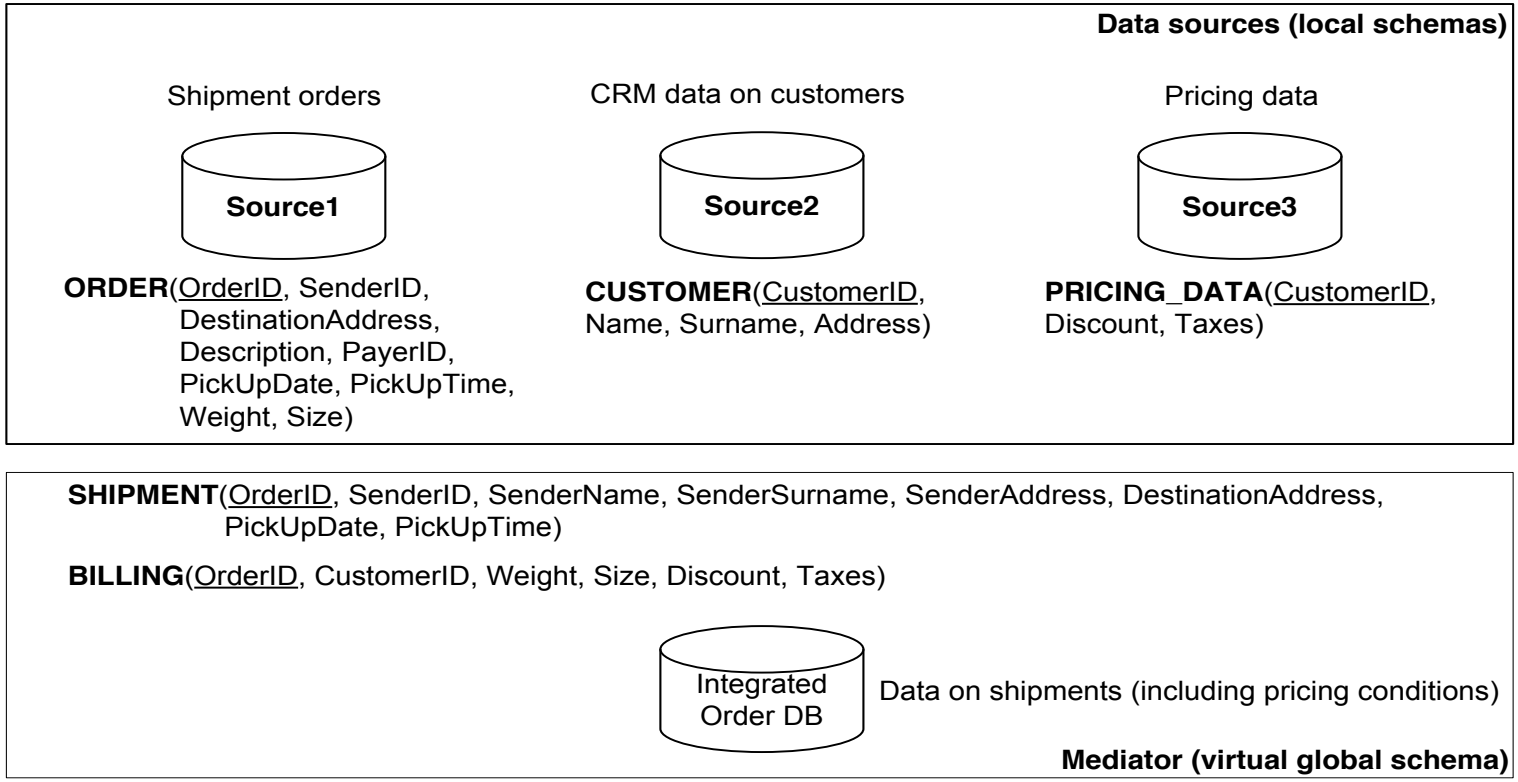


Fig. 2.2 Example of an integrated database storing shipment data extracted from different data sources. Each data source is characterized by a *local schema*. Data integration is performed according to a *vritual global schema* managed by the mediator.

Picture from Mashups: Concepts, Models and Architectures

Data Level Integration …

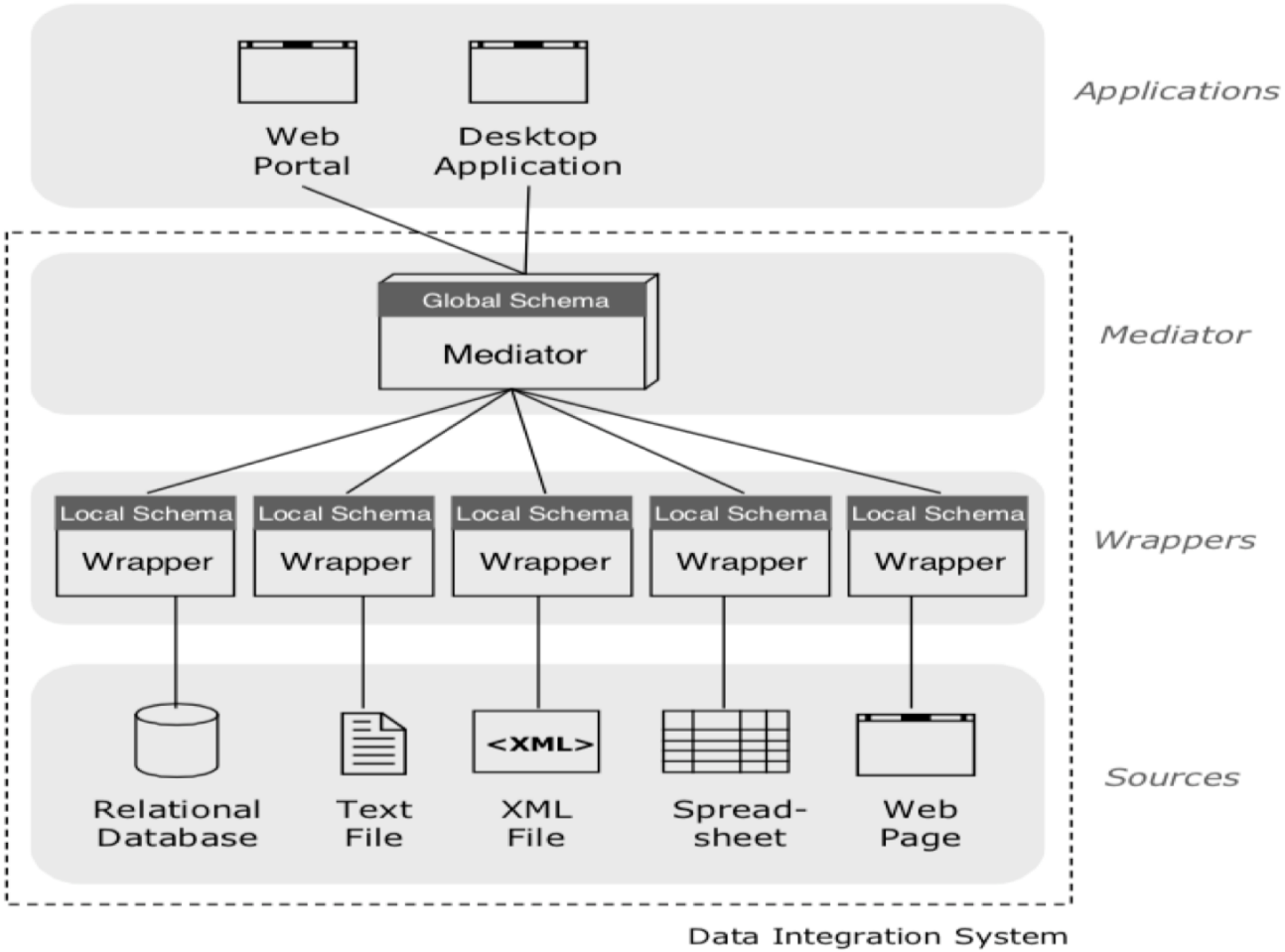
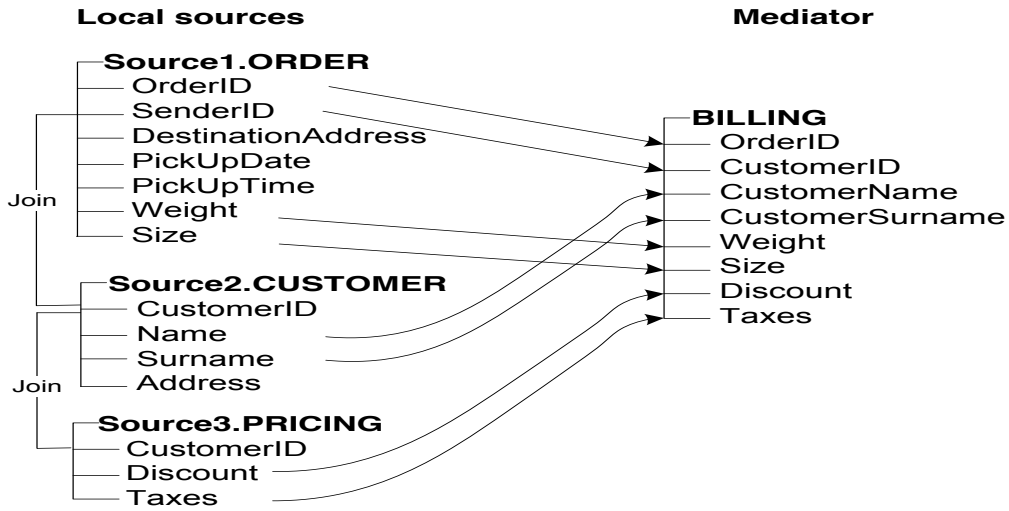


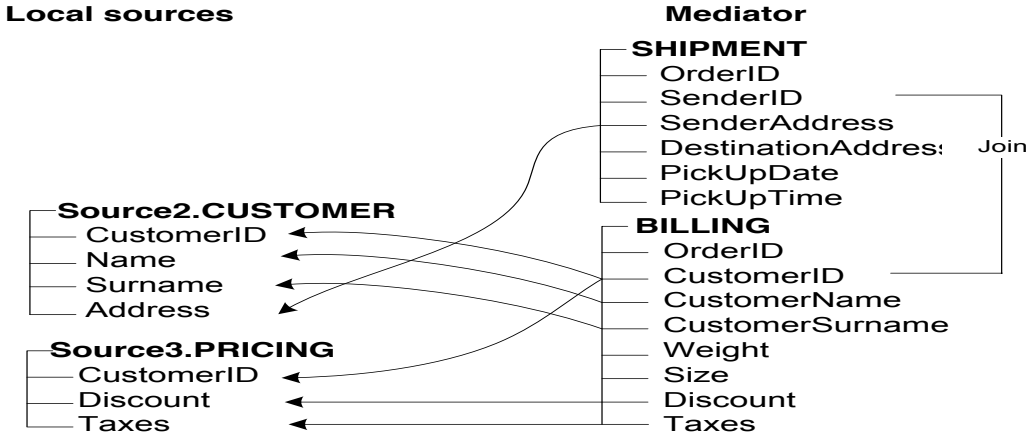
Figure 1: View-based Data Integration System (VDIS) Architecture

Picture from View-based Integration, Yannis Katsis, CSE, UC San Diego

Data Level Integration …



a) **GAV Mapping for the global relation BILLING**. The global relation is defined as a view on the local source relations.



b) **LAV Mapping for Source2 and Source3.** The local source relations are defined as views over the global relations.

Fig. 2.3 Example of GAV and LAV schema mappings for the integrated order DB.

Picture from Mashups: Concepts, Models and Architectures

Data Level Integration …

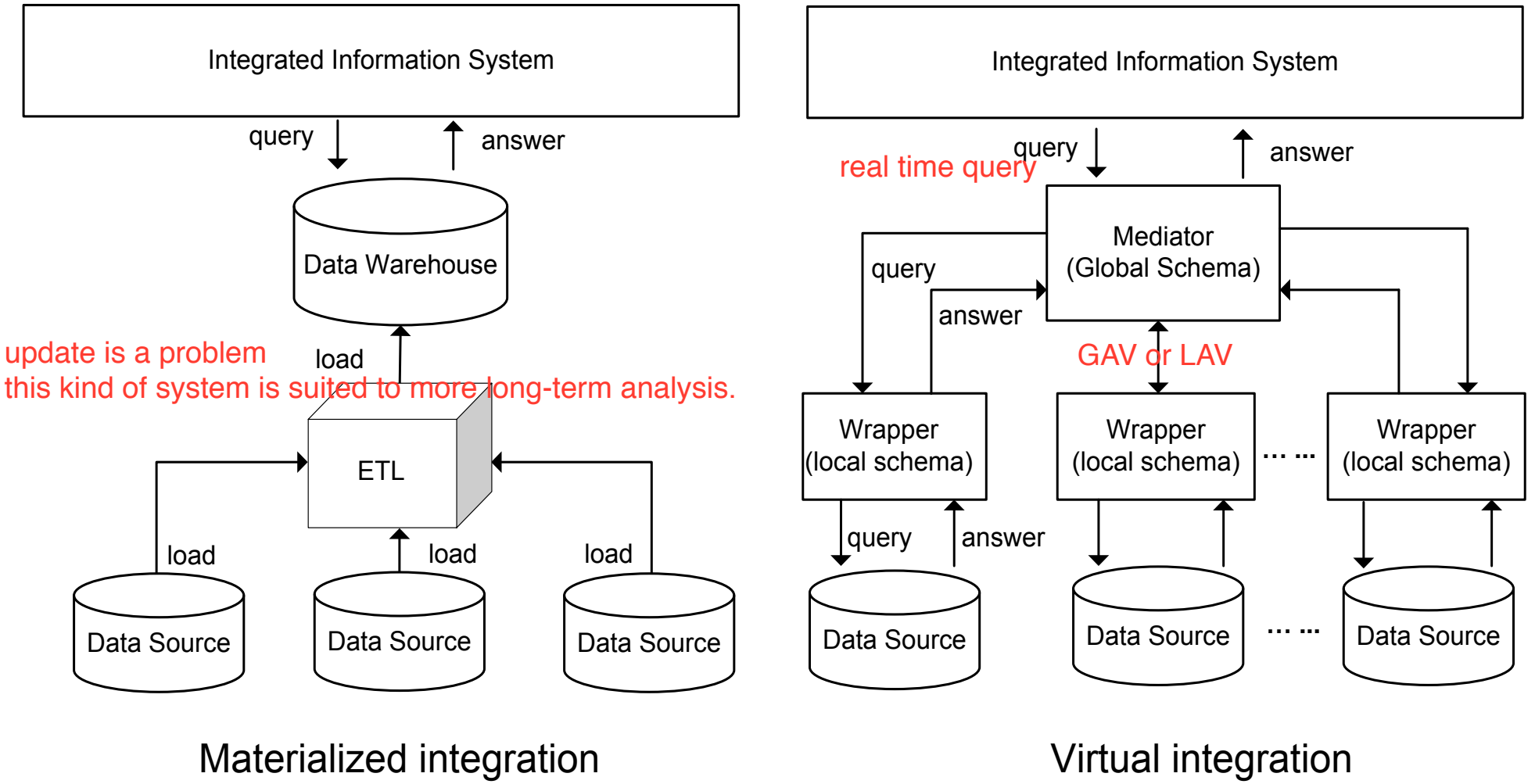
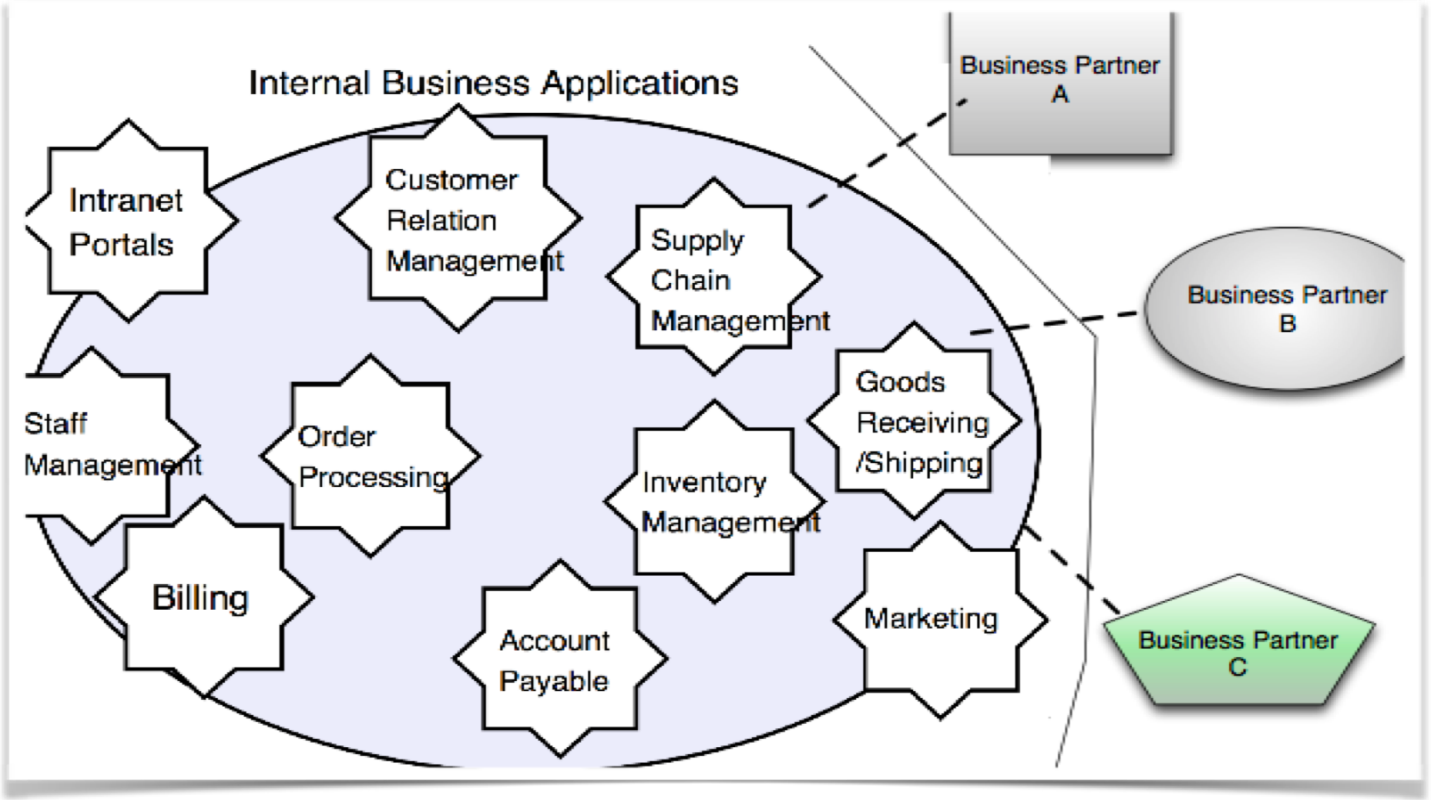


Fig. 2.1 Typical architectures for materialized and virtual data integration [3].

System Level Integration …

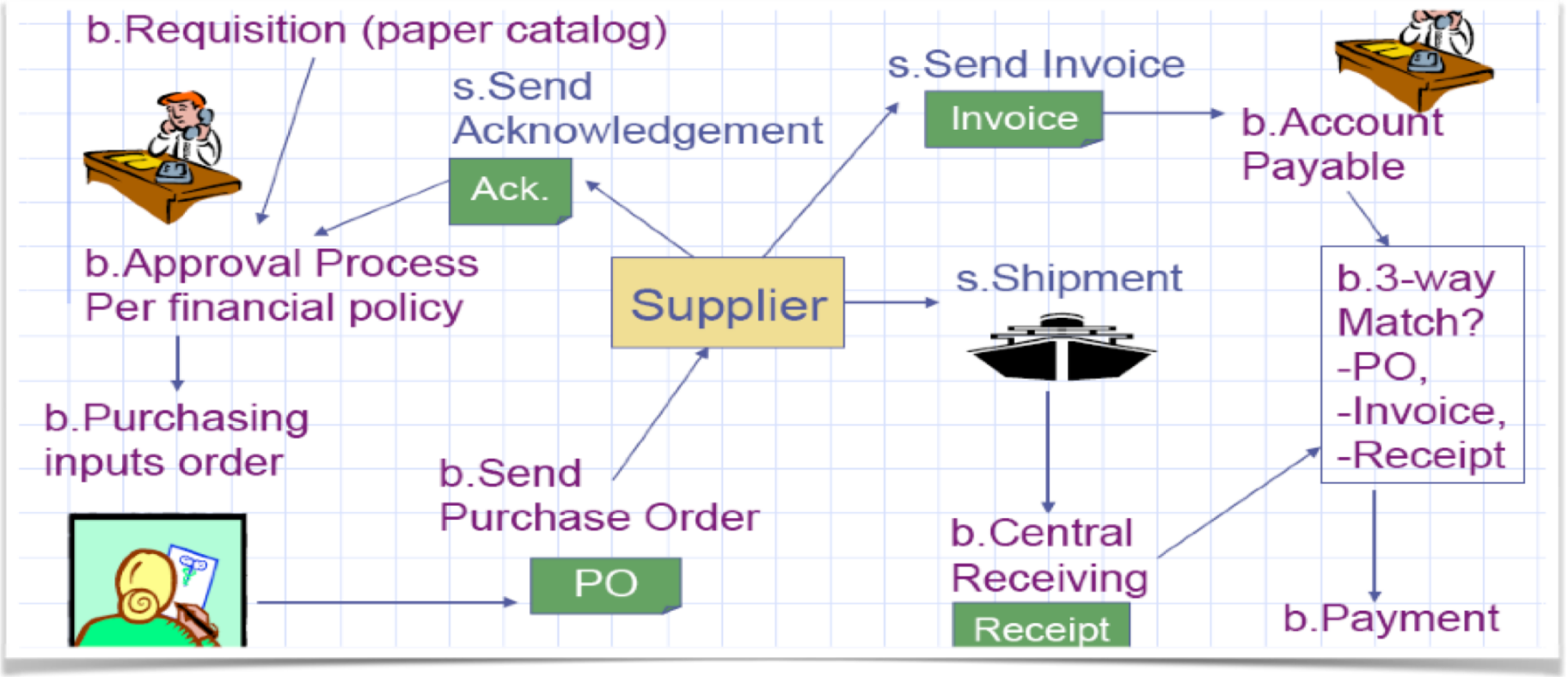
In enterprise environments, pick any sizeable organisation. You will see many departments performing different functionality

In silos, often supported by software systems

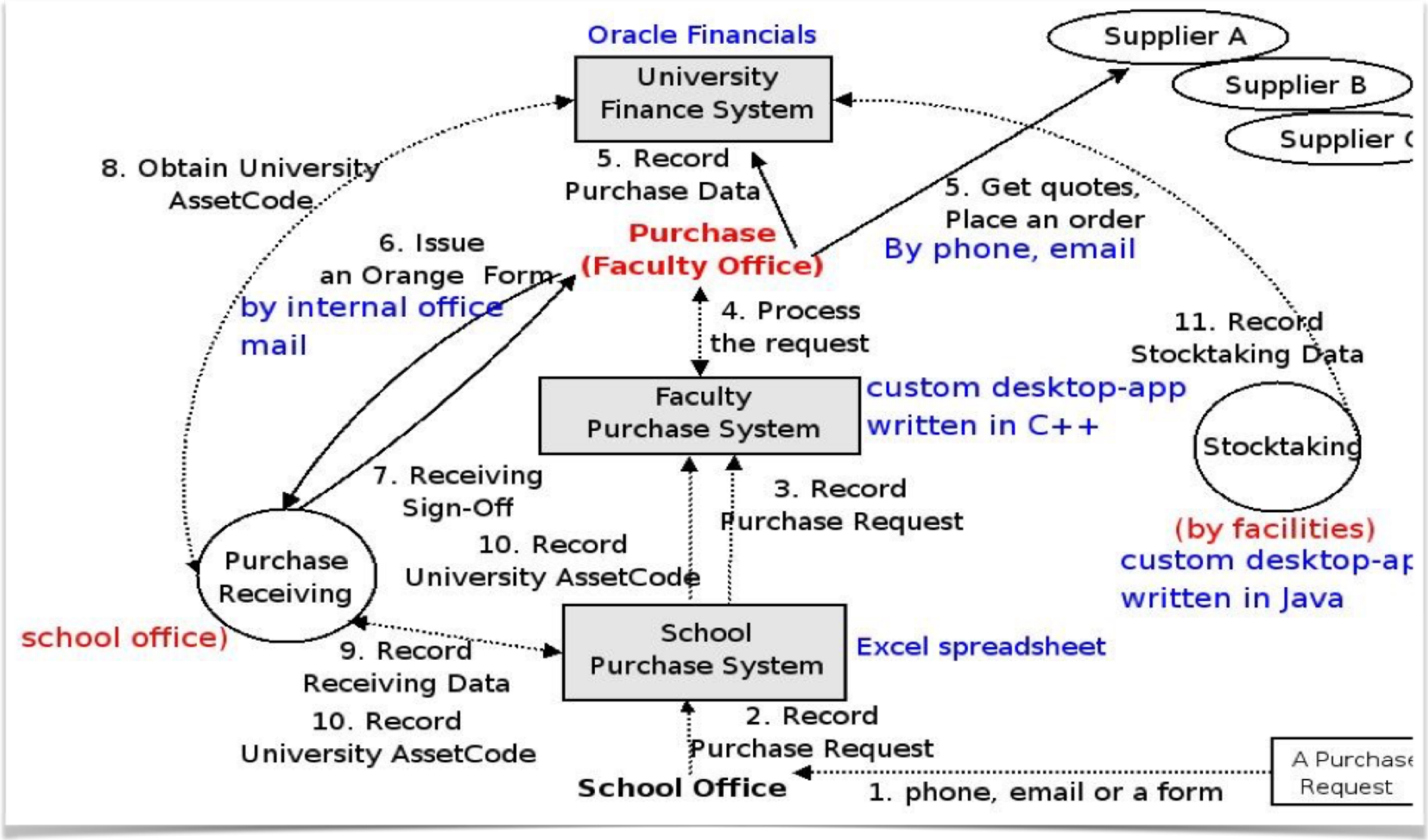


A Typical Purchase Order Process

In reality: communication/coordination between the silos needed

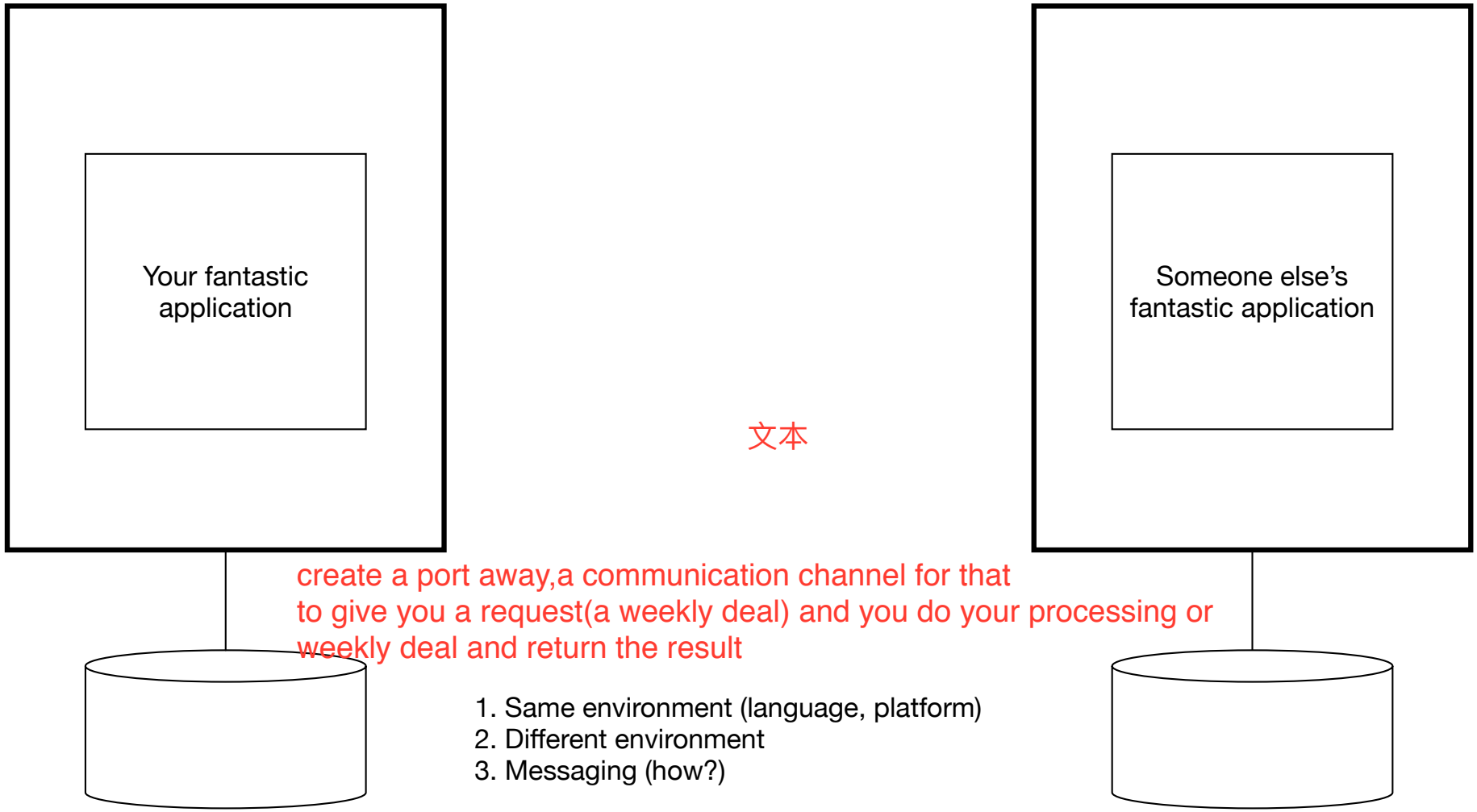


An example of (real) Purchase Order Process



Going outside of your system boundary …

obtain sth from some other systems



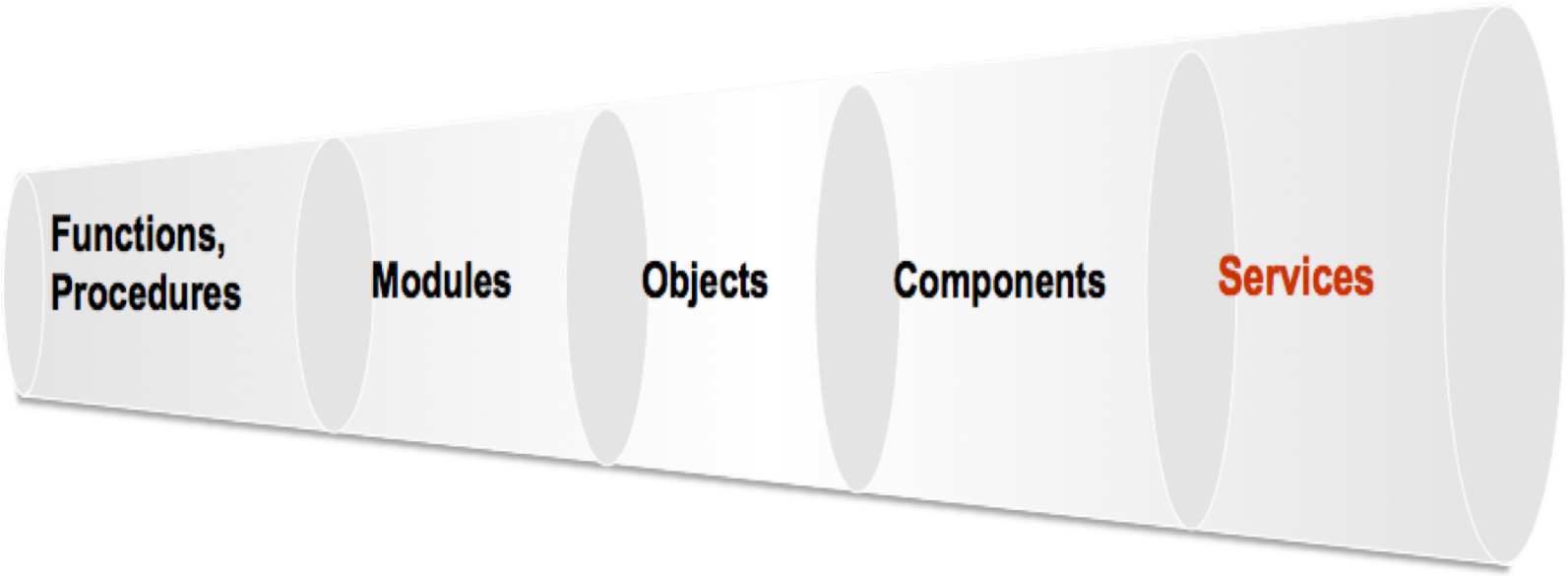
create a port away,a communication channel for that
to give you a request(a weekly deal) and you do your processing or weekly deal and return the result

1. Same environment (language, platform)
2. Different environment
3. Messaging (how?)

The evolution of programming abstractions

Services: “customer” and “service provider”

Lines of code vs. Services - consider software building exercise as ‘building services’, ‘discovering services’ and ‘combining services’



The evolution of programming abstractions

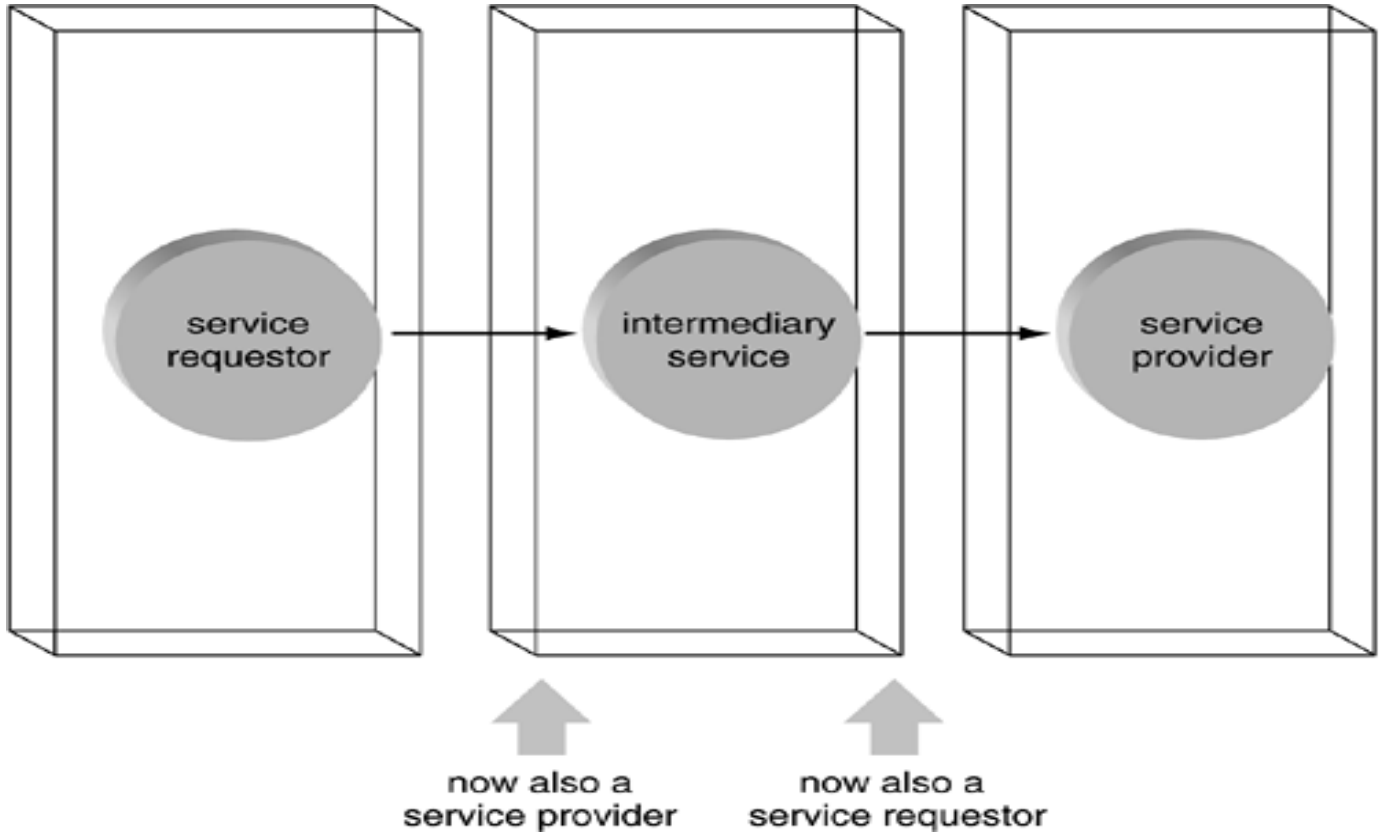
In SOA, we talk about software as a service ... That is, SOA is about building software systems composed of a collection of (software) services

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") …
  - like lego blocks
  - "my" (the developer) service could be used in scenarios that I never anticipated

Simplified view of services (or API ?!)

you may send some requests in Json object and you may expect to receive some lists of Json objects



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

Week 1 Objectives

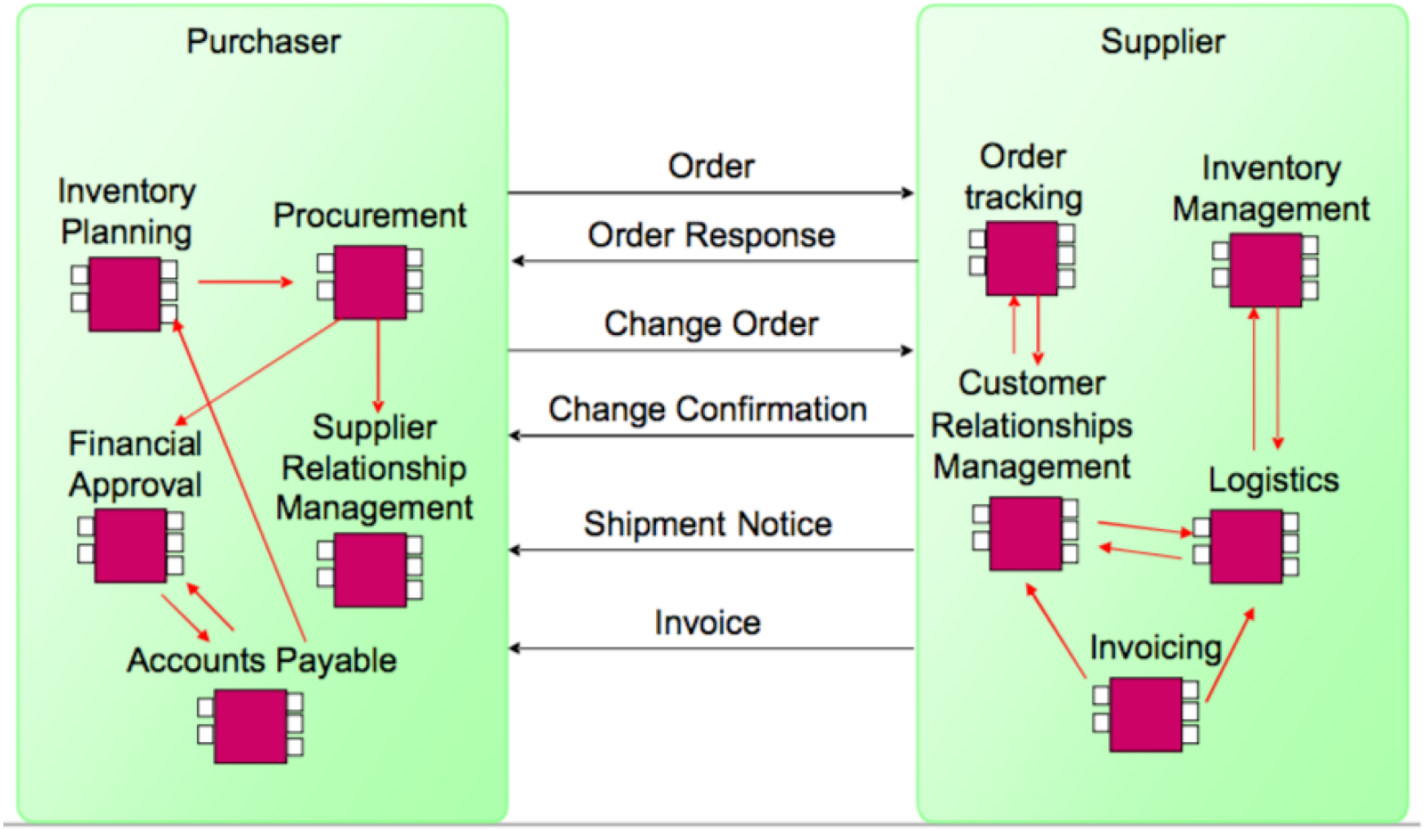
Motivation

- 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: Web Services by Alonso, Casati, Kuno and Machiraju, Chapters 1-3

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



Enterprise Application Integration (EAI)

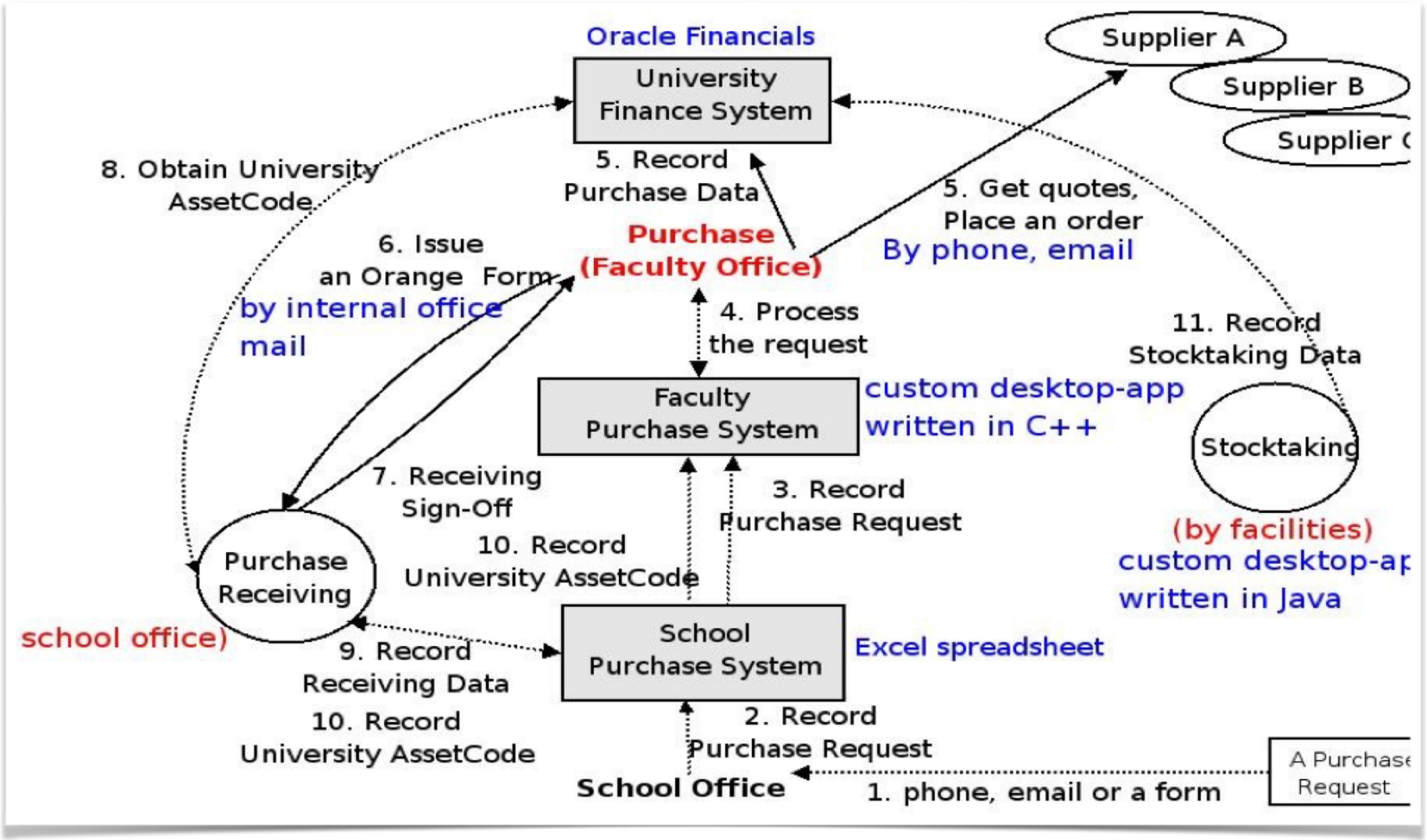
Motivations: Streamlining business operations, globalisation, competition, mergers and acquisition, new business models, technology development (e-commerce), etc.

EAI definition by Hewlett Packard: A set of services and solutions for bringing 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

three logical layers

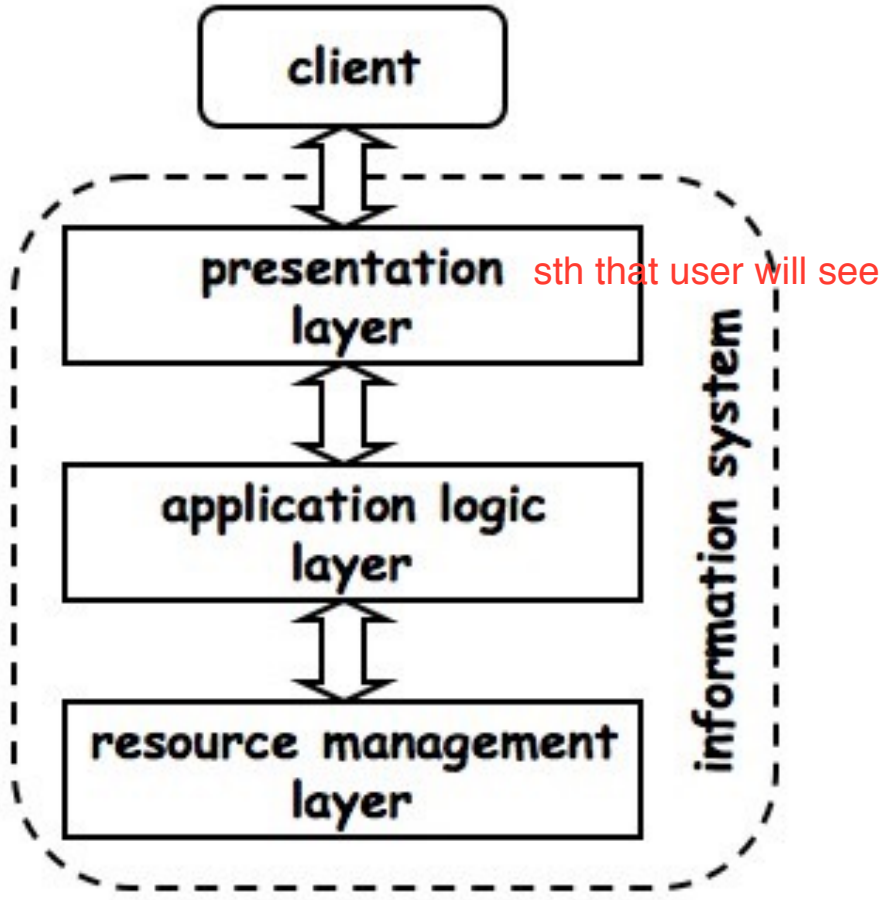
PL: formatting, presenting information to clients (e.g., JSP)

AL: determines what the system actually does, enforcing the business rules and processes (e.g., a program that implements customer registration)

What defines the system itself,What the system does is defined.

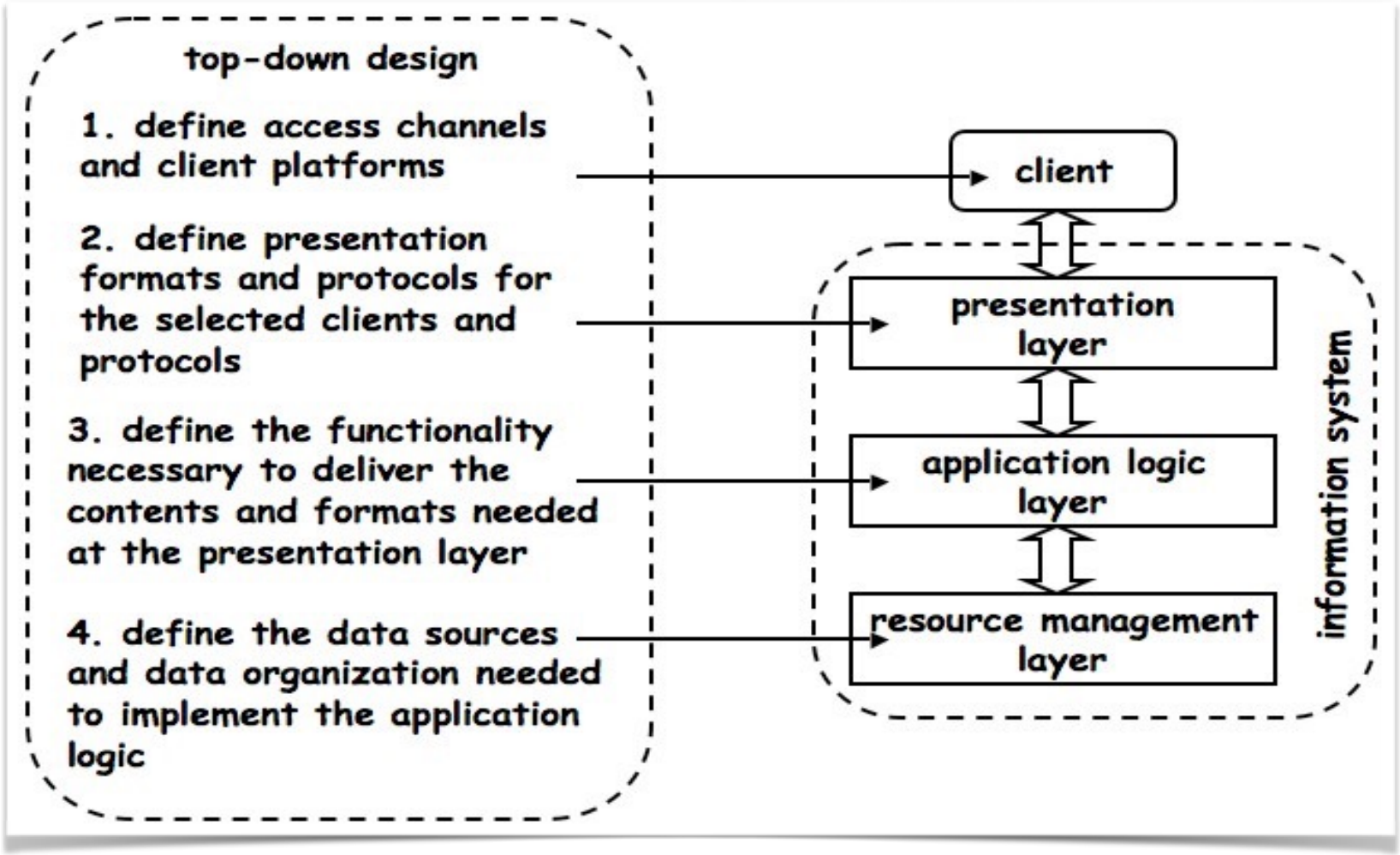
RM: storage, indexing, and retrieval of the data necessary to support the application logic layer (e.g, RDBMS)

is supported by the data



sth that user will see

Information System Design: Top-down Design



Characteristics of Top-down Design

goals: Focus first on the high-level objectives, then proceed to define everything required to achieve those goals

首先关注高层目标，然后着手定义实现这些目标所需的一切

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.

为了简化系统开发/维护，分布式节点通常创建为在同构计算环境中运行。
一个组件的功能取决于其他组件的功能。

more control: relatively easy to address both functional and non-functional (e.g., performance) issues

from-scratch-development: few information systems nowadays can be developed this way

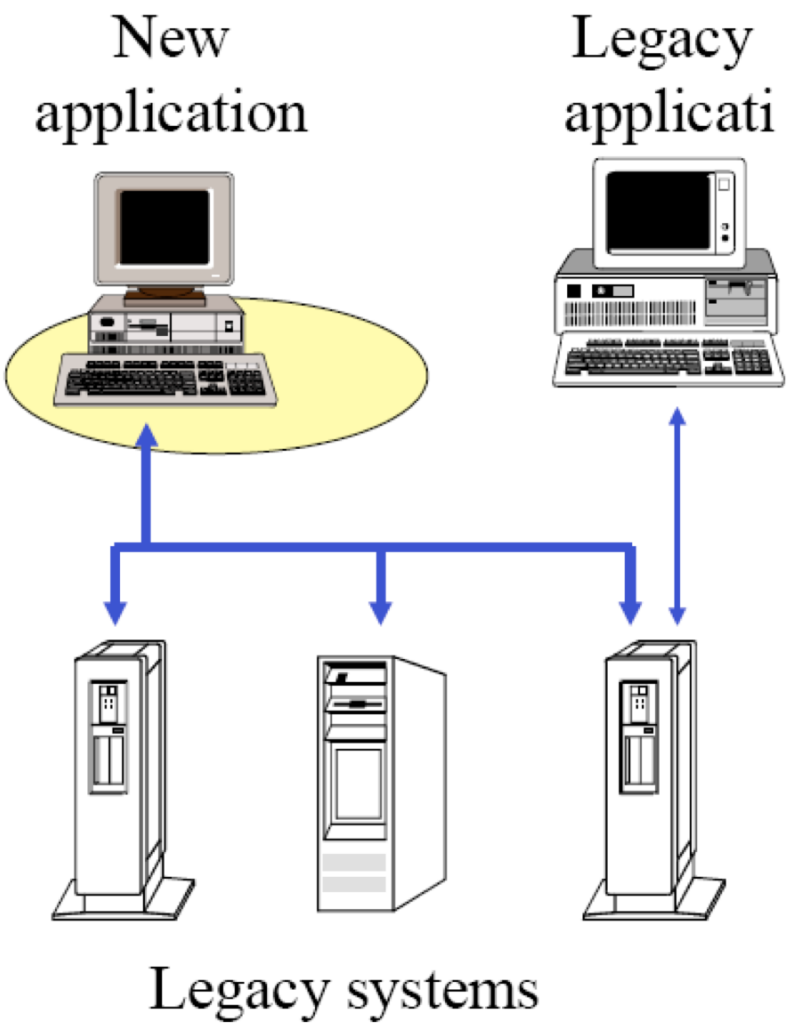
Information System Design: Legacy Systems

Any existing system that has to be now integrated in some other systems without replacing it because replacing it is too expensive, then those existing system becomes legacy system.

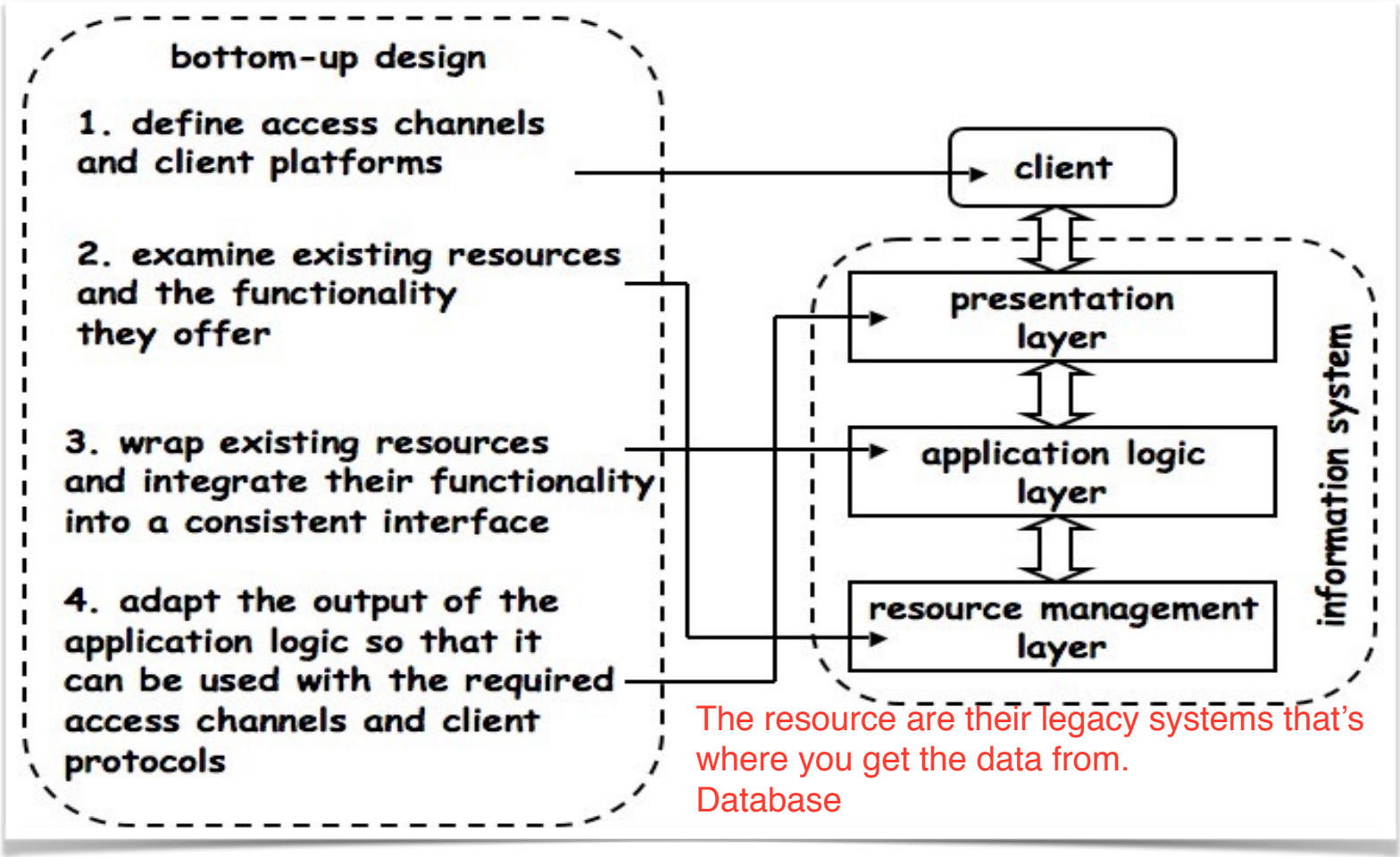
Legacy systems: a computer system or application program which continues to be used because of the cost of replacing or redesigning it

The functionality provided by legacy systems is predefined and cannot be modified

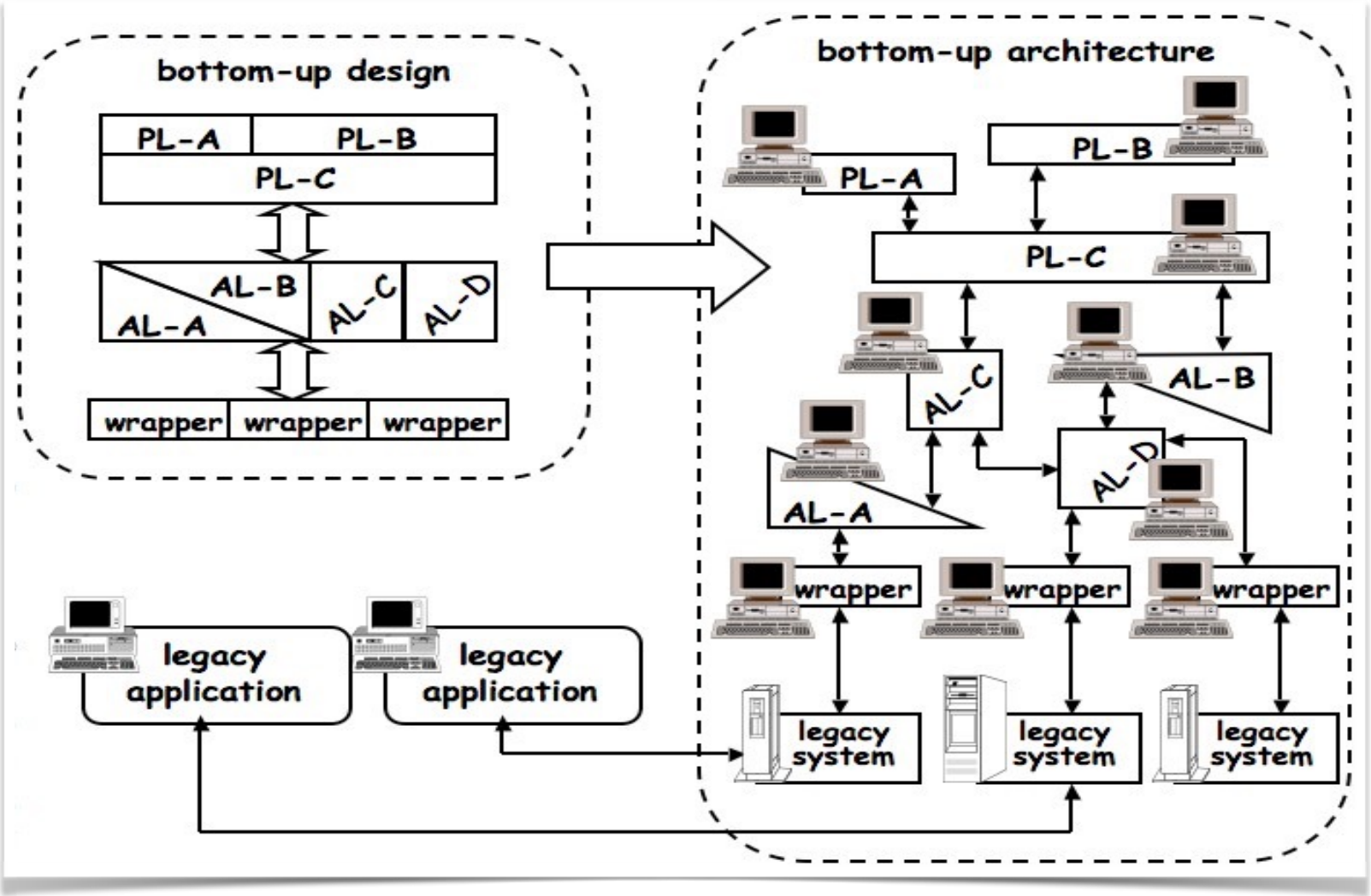
The design is driven by characteristics of the lower layers



Information System Design: Bottom-Up Design



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.

这些是需要整合到新系统中的独立系统。

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.

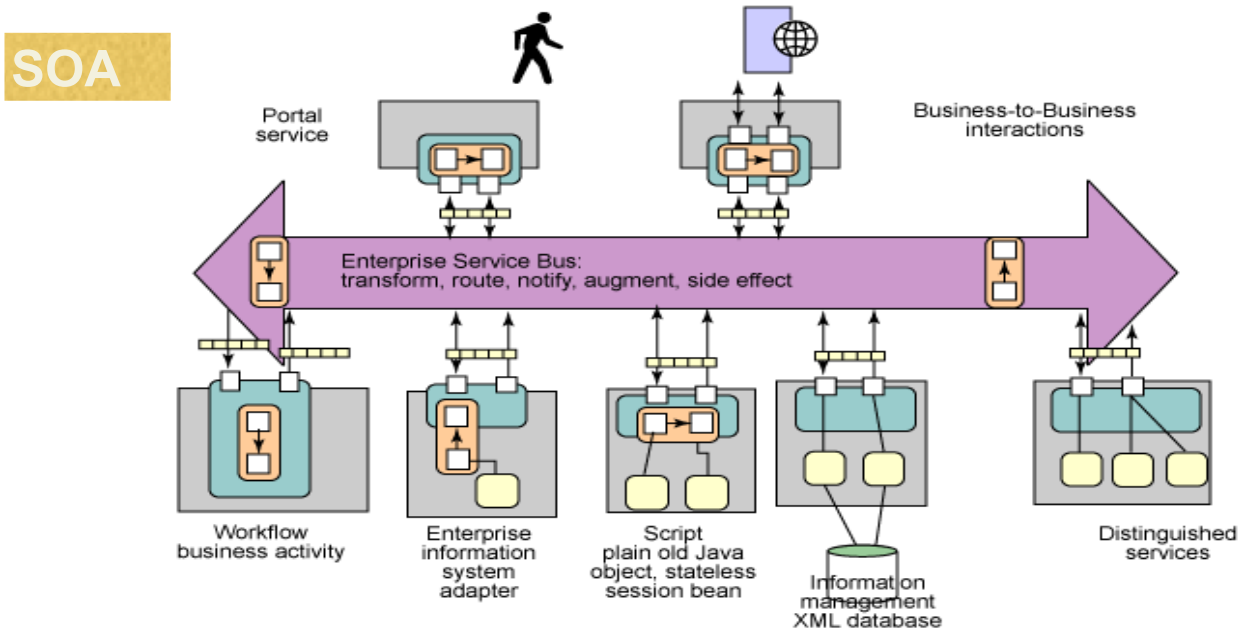
wider usage: This approach is used widely because legacy systems exist and typically cannot be easily replaced.

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'

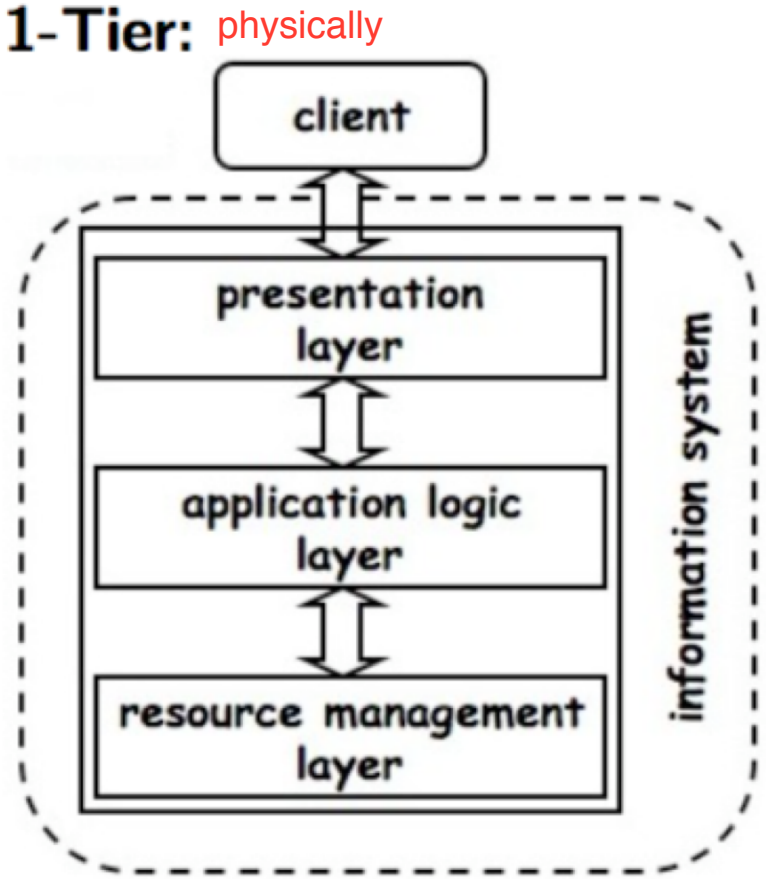
users/programs access the system through "dumb" terminals, whose display is controlled by the information system (e.g., mainframes).

(+) simple, highly optimised, centralised, no deployment or portability issue

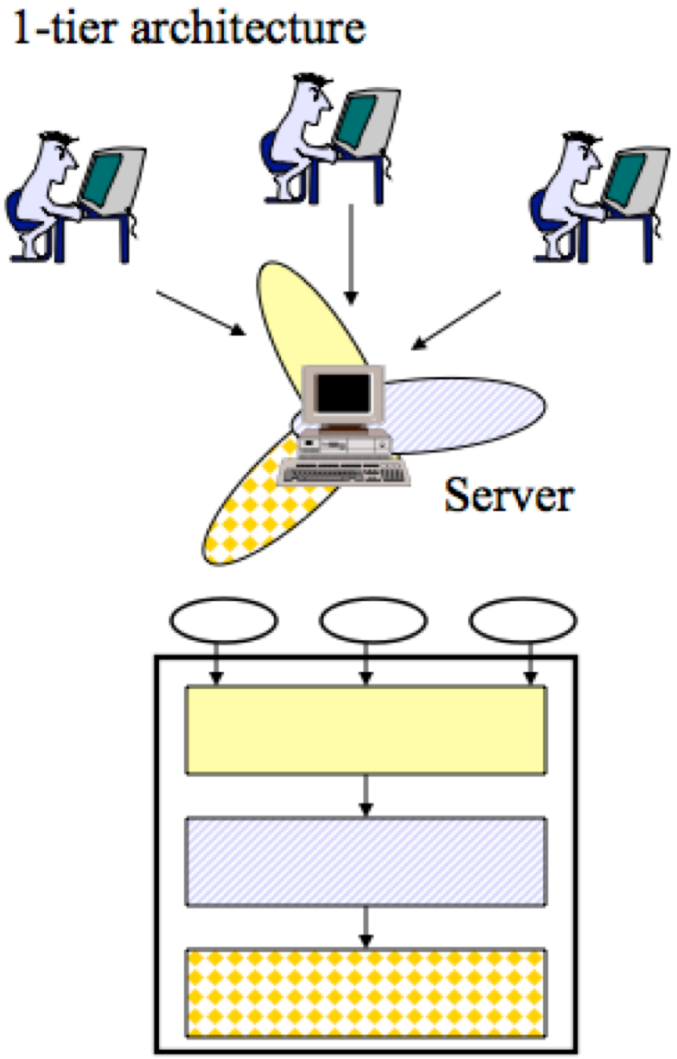
简单，高度优化，集中化，无部署或可移植性问题

(-) no entry point except the client terminals (hard to be integrated into other systems)

除客户终端外无其他入口（难以集成到其他系统中）



1-Tier



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.

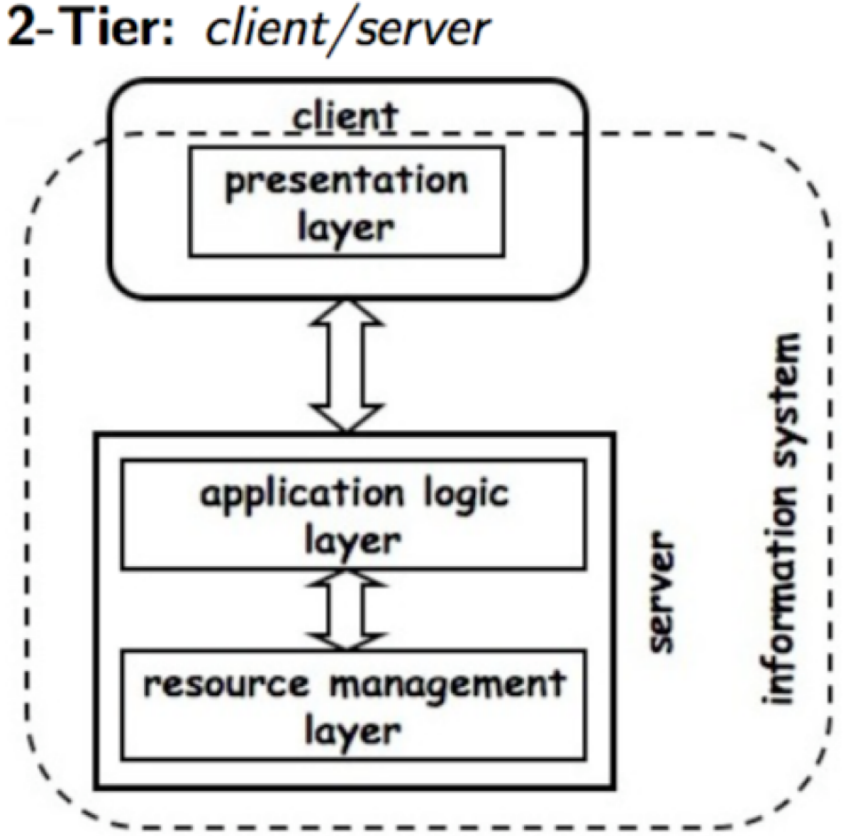
客户端可以有更复杂的表示层
同时还可以节省服务器上的计算机资源。

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

physically separated system



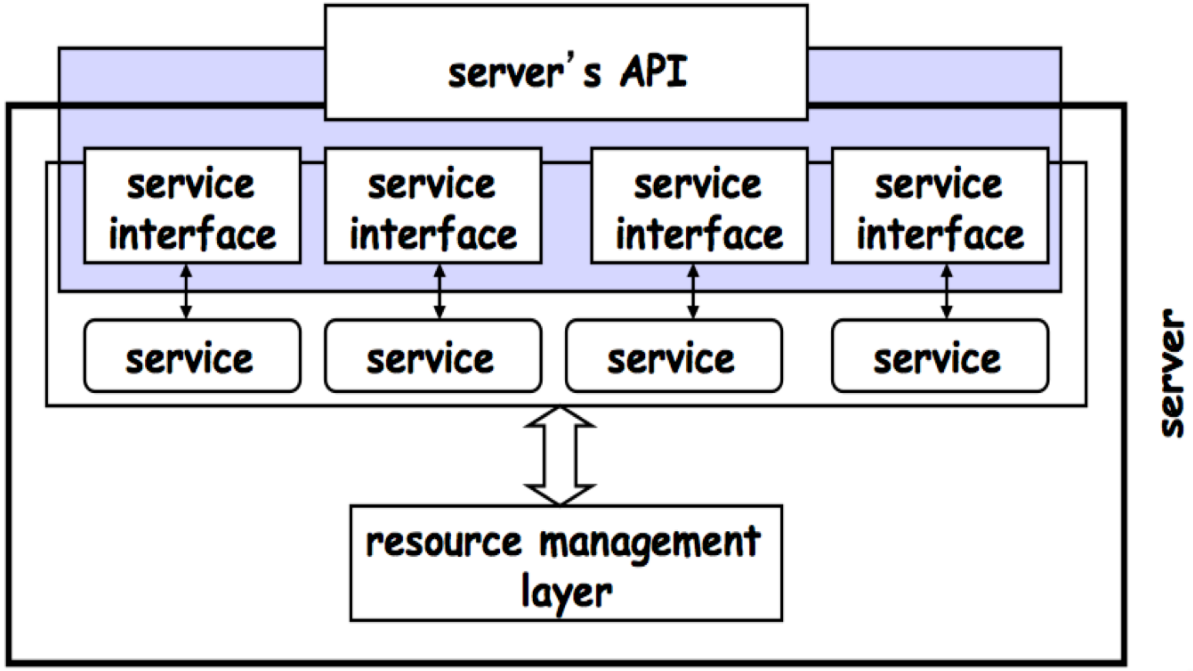
Key developments in distributed systems by 2-Tier

客户端调用服务器实现的服务

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) 客户端如何调用给定的服务

The concept of API -> can support diverse clients, change/evolve the server without affecting the clients.

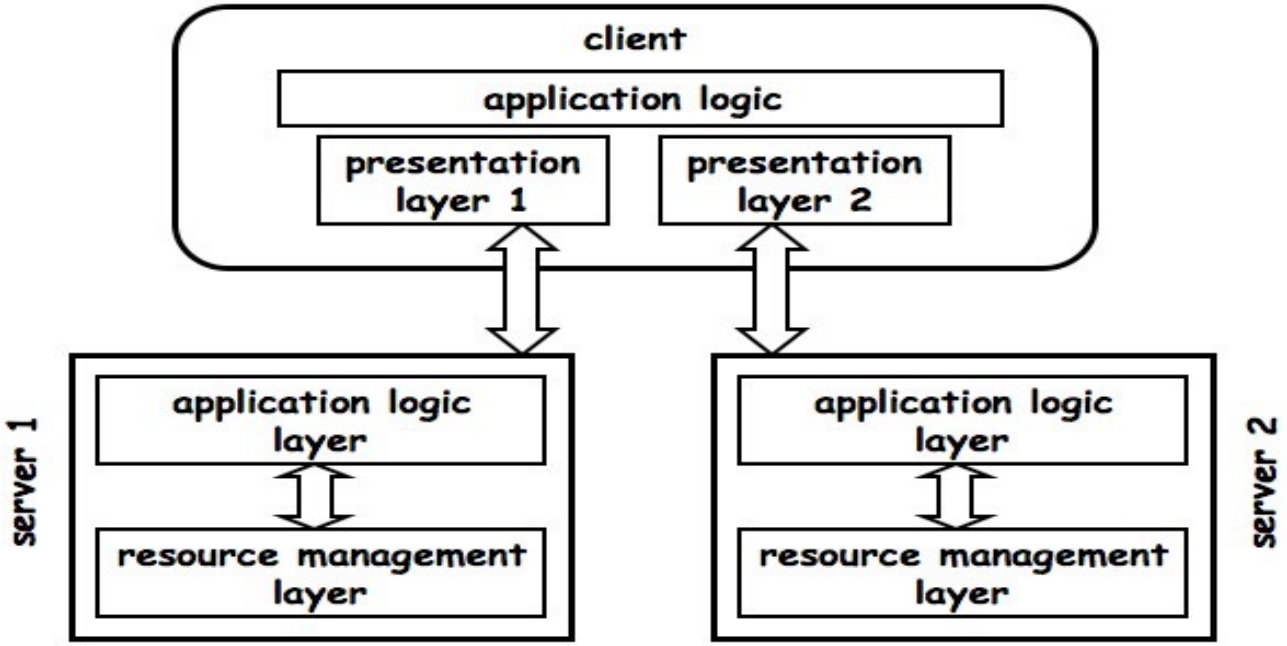


Problems with 2-Tier (in terms of integration)

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 ...
complicated



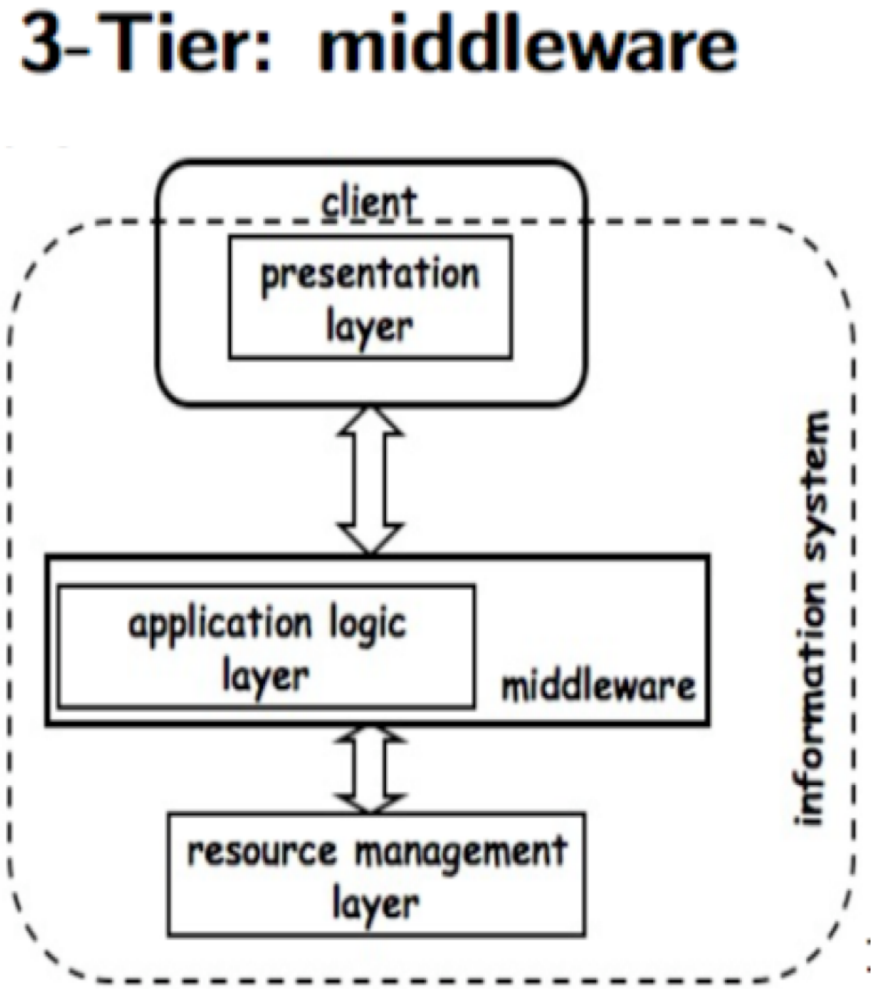
Architecture of an Information System

Added - application logic layer (aka middleware)

introduces an additional layer of business logic encompassing all underlying systems

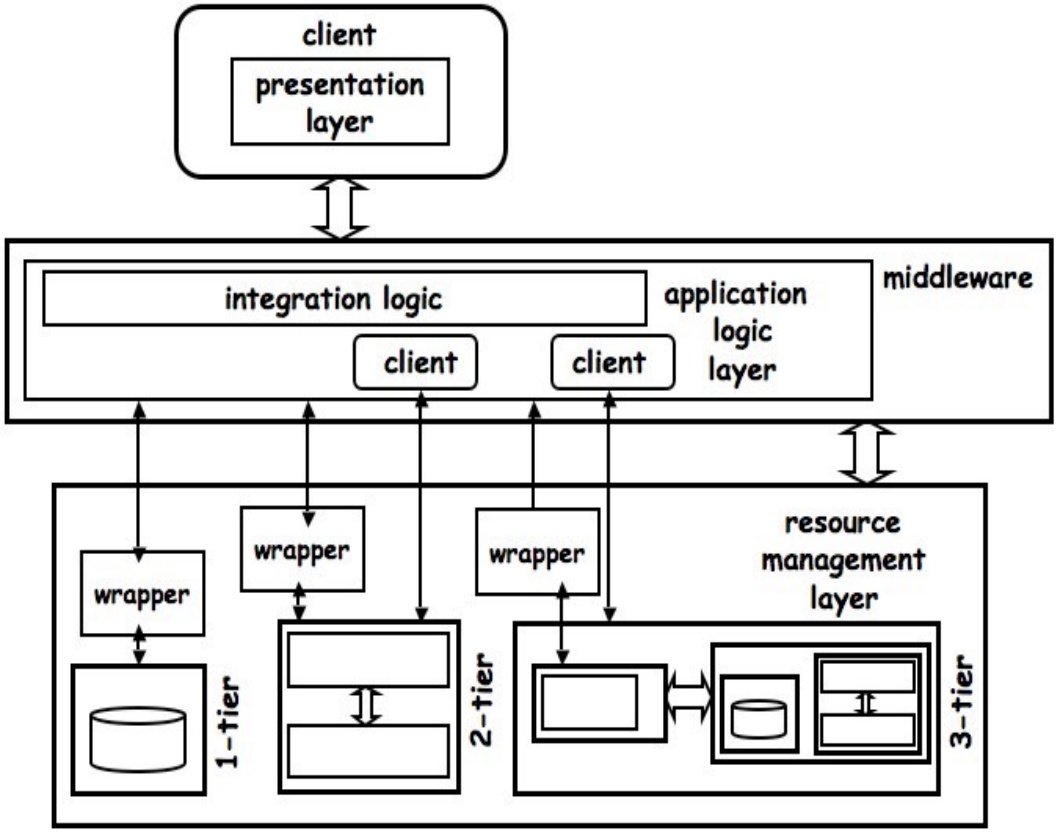
By doing this, a middleware system:

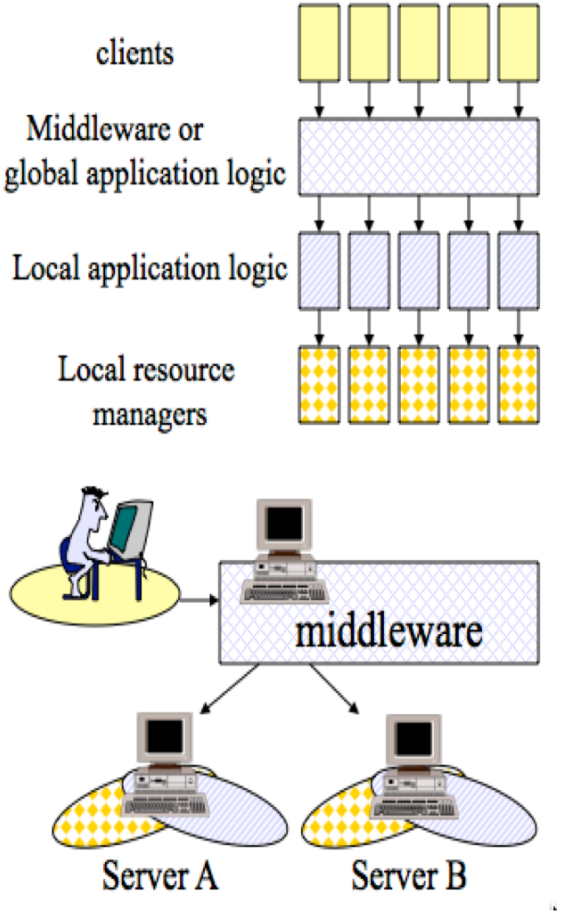
- simplifies the design of the clients by reducing the number of interfaces,
- provides transparent access to the underlying systems,
- takes care of locating resources, accessing them, and gathering results.



The Middleware in 3-Tier

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





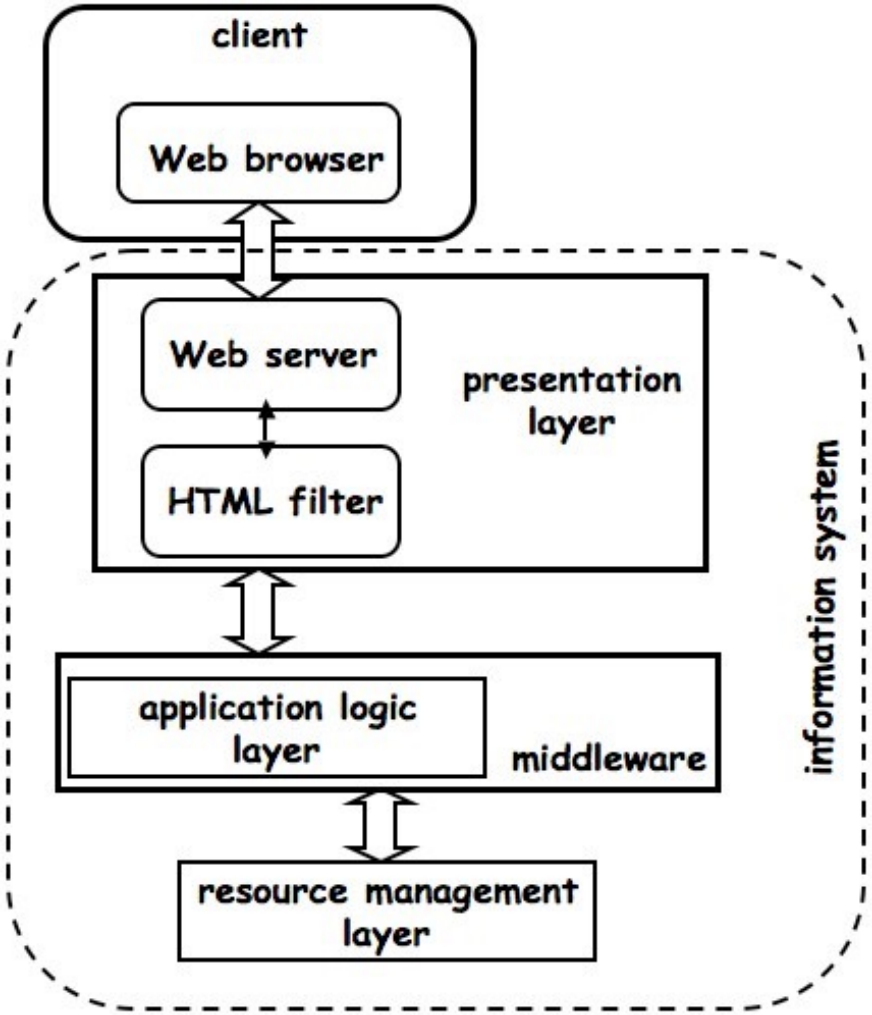
Architecture of an Information System

N-Tier

created by either connecting several 3-tier systems and/or by adding a Web layer

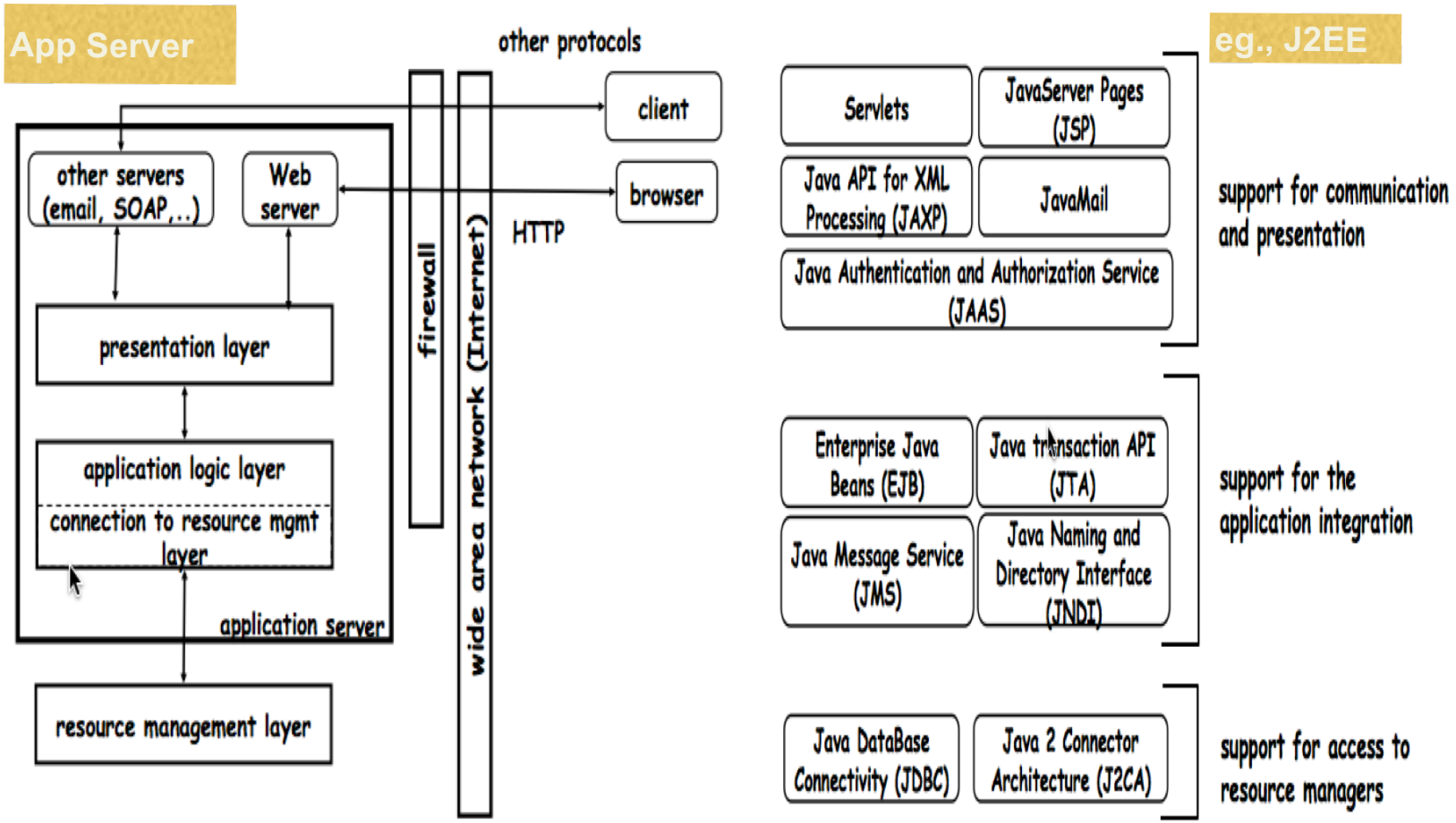
normally, web layer is incorporated into a presentation layer that resides on the server side (part of the middleware infrastructure)

The addition of the Web layer led to the notion of "application servers" which was used to refer to middleware platforms supporting Web access



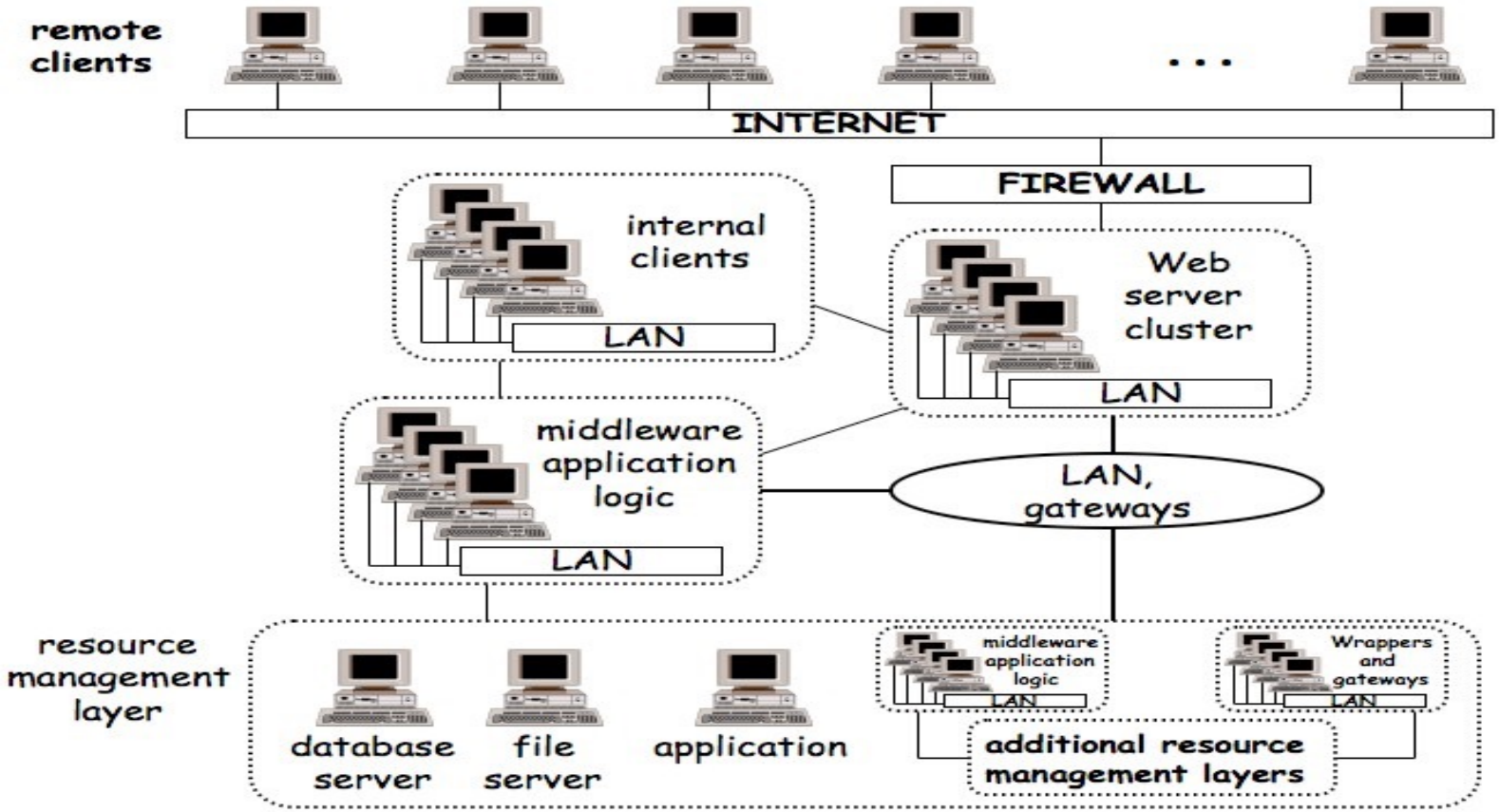
So-called “Application Servers” are …

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



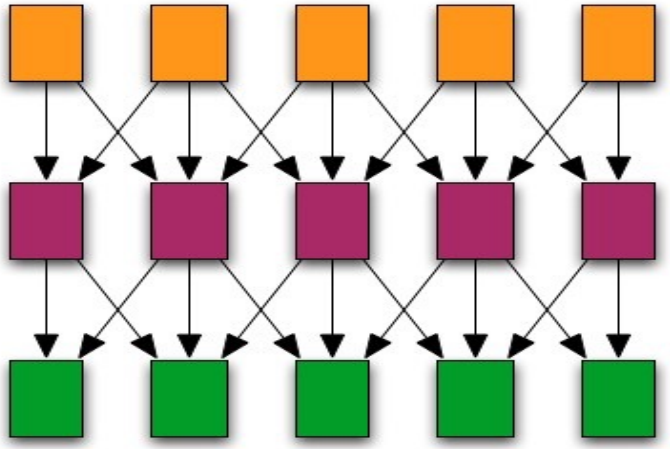
N-Tier Systems …

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

tier is that physical separation, so you have logical application idea

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.

当我们将信息系统中的层和层分开时，我们假设所有这些元素之间存在某种形式的交流。

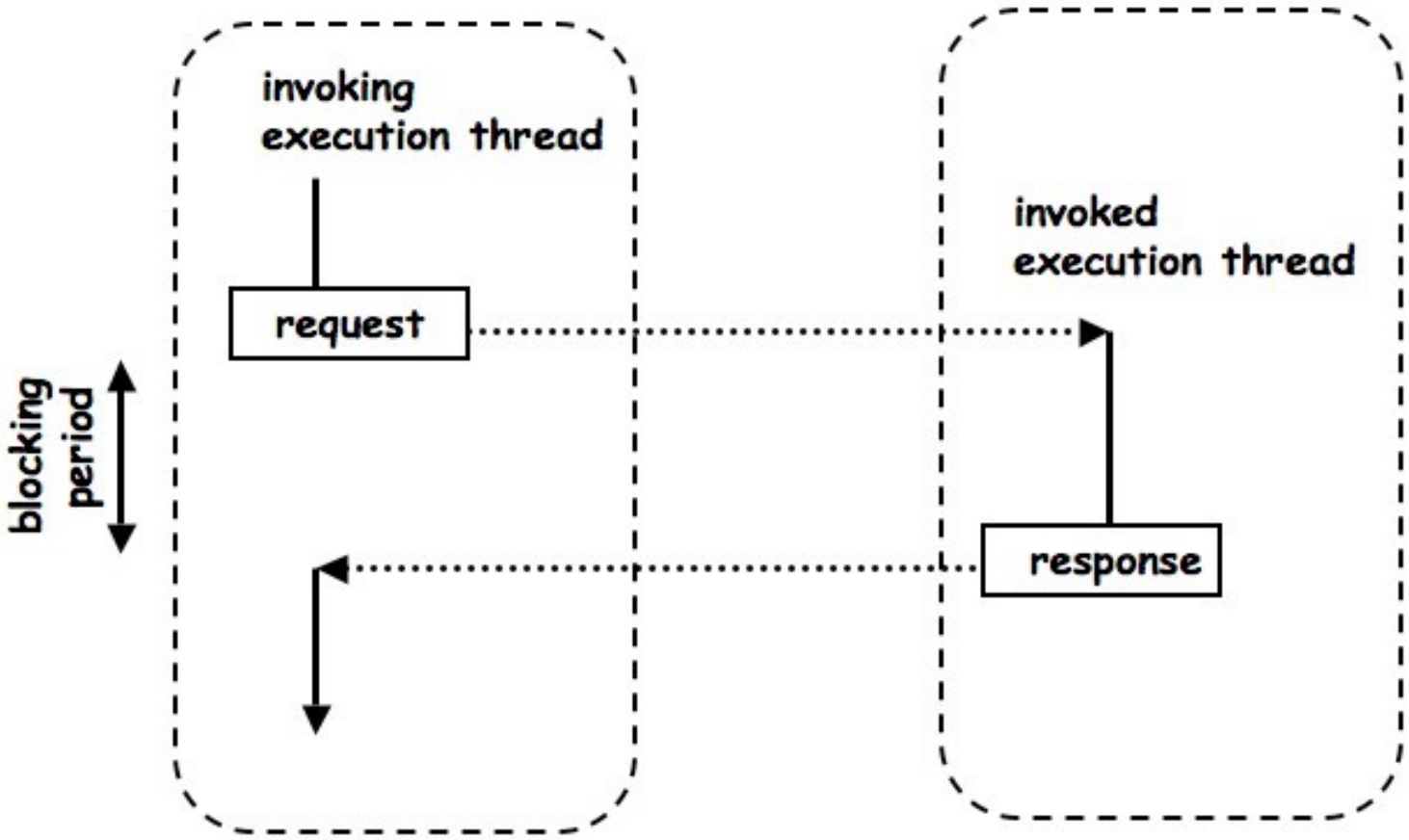
There are two communication patterns widely used: synchronous and asynchronous.

- synchronous: blocking interaction 阻止交互
- asynchronous: non blocking interaction

同步和异步

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

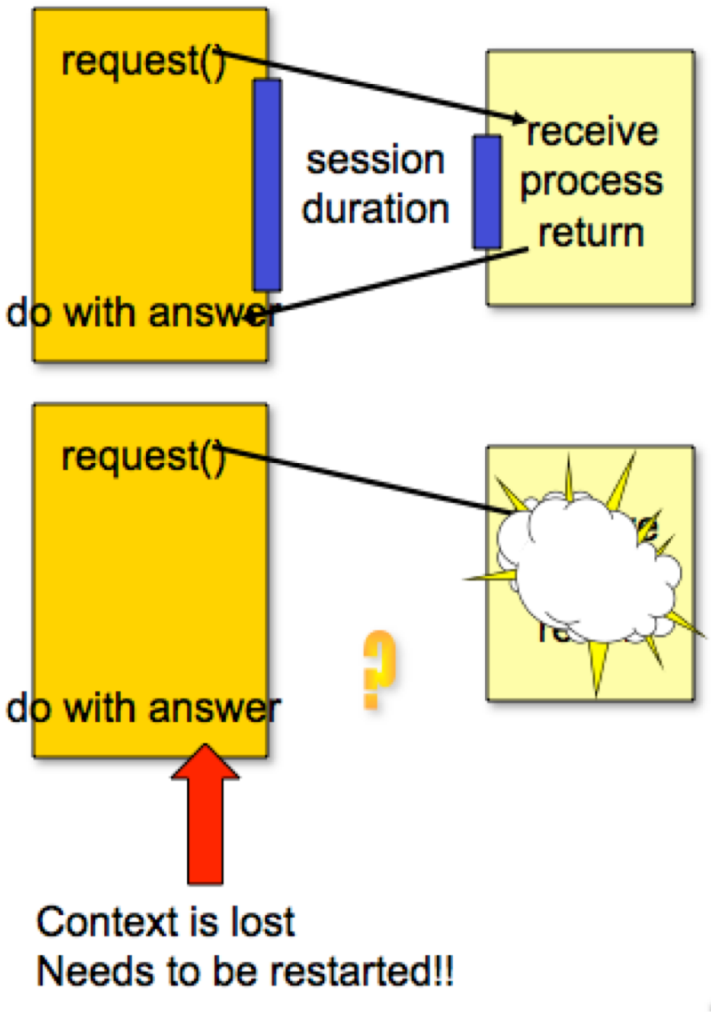
Overhead of Synchronisation …

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



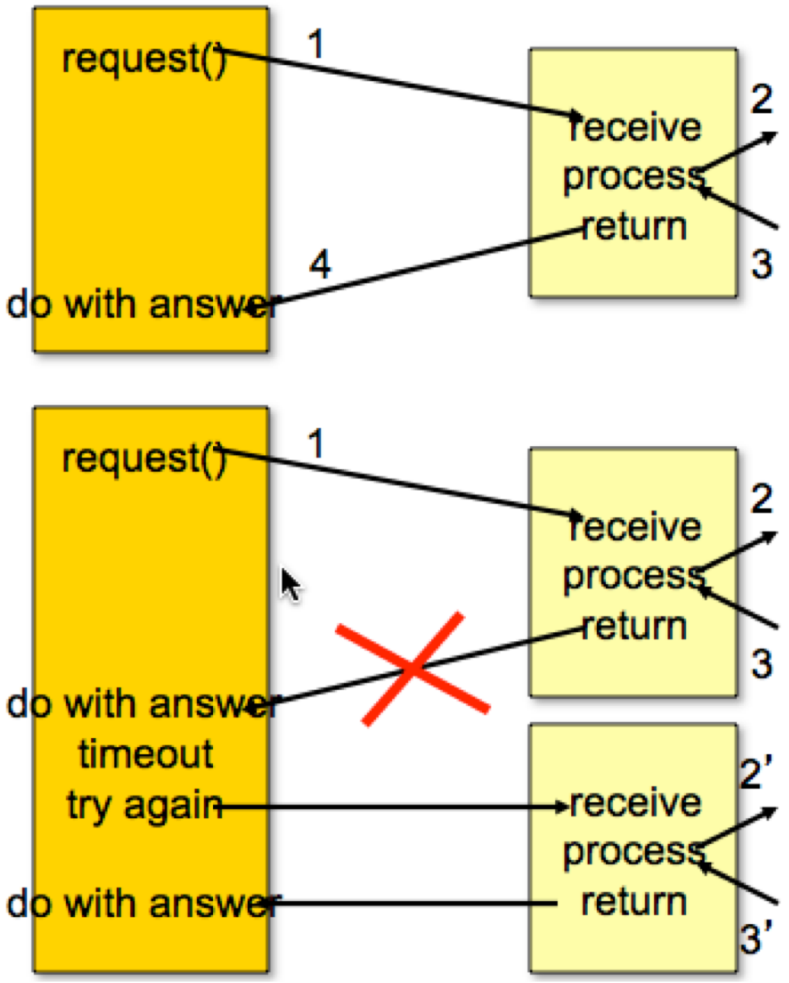
What to do when it fails …?

If the client or the server fail, the context is lost – recovery is difficult

- failure occurred before 1, nothing has happened
- failure occurs after 1 but before 2 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, 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:

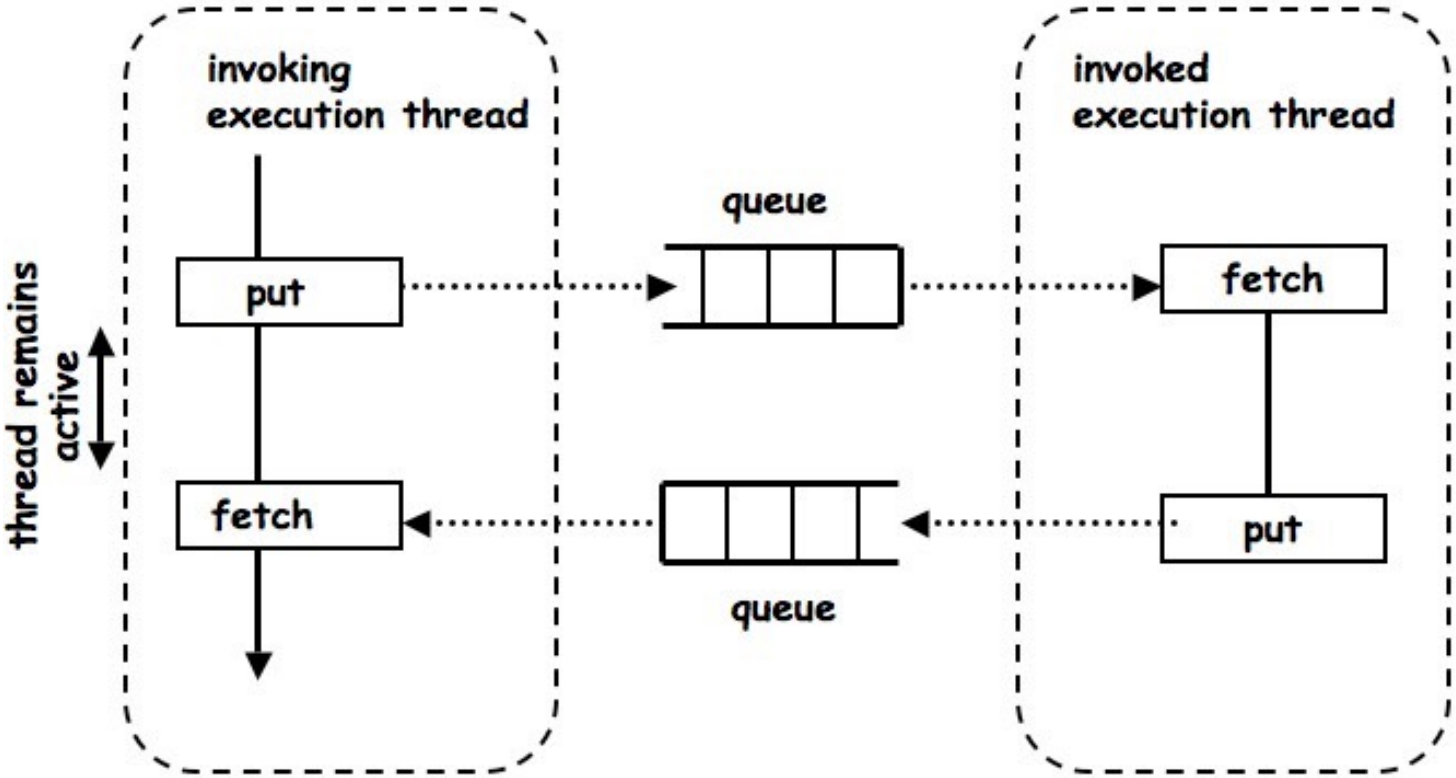
- Transactional interaction: to enforce exactly once execution semantics and enable more complex interactions with some execution guarantees

- 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)

Summary

- Can you name two approaches to information system design? Top-down , Bottom up(legacy system)
- Can you identify the layers in information system?
  presentation layer, application layer,resource, management layer
- When implemented, the conceptual layers are combined/distributed in many ways and form (……)? Tiers
- Can you name a key concept born out of 2-Tier architecture? (API/services)
- Problems in integrating 1-Tier or 2-Tier systems?
- Can you give me an example of non blocking communication pattern?

Let’s go back to course aims

Backdrop:

Software engineering has advanced rapidly in recent years. The knowledge-, service-, and cloud-based economy in parallel with the continuous improvement in connectivity, storage and data processing capabilities allow access to a data deluge from sensors, social-media, news, user-generated, government and private data sources.

Accordingly, in a modern data-oriented landscape, data-driven applications may need to deal with a collection of data sets - from unstructured, relational to NoSQL - that holds a vast amount of information gathered from various private/open data sources.

Therefore, well-engineered service-oriented functionalities are critical for ingesting, organizing and querying the growing volume of data in modern web-based applications.

Let’s go back to course aims

This course aims to introduce the student to core concepts and practical skills for engineering the data in service-oriented data-driven applications. Specifically, the course aims to answer these questions:

- *How to access and ingest data from various external sources?* API
- *How to process and store the data for applications?*
- *How to curate (e.g. Extract, Transform, Correct, Aggregate, and Merge/Split) and publish the data?*
- *How to apply available analytics to the data?*
- *How to visualize the data to communicate effectively*

Fundamentally, we will look at these questions through the lens of 'service-oriented' software design and implementation principles. At each topic, we will learn some core concepts, and how to implement the concepts in software through services.

Let's go back to course aims

Assumed Knowledge

Before commencing this course, we will assume that students have:

- completed one programming course (expected to be in Python)
- basic data modelling and relational database knowledge

These are assumed to have been acquired in the following courses: For Postgrad - COMP9021 and COMP9311. For Undergrad - COMP1531 and COMP2041.

NOTE: This course is not meant to be an advanced course …

Let’s go back to course aims

Student Learning Outcomes

- Describe the main requirements to design and implement APIs (REST APIs).
- Describe the main architecture of a modern Web-based, data-oriented applications
- Understand the various technologies involved in accessing, storing, publishing data
- Understand how to apply existing analytics and visualisation techniques to data
- Design and develop non-trivial data services solutions.
- Understand the basic issues with scalability, security/privacy and using online data processing platforms

this course does not aim to introduce you to the latest packages or products available in the market. Rather, it strives to teach students the basic concepts and the fundamental principles in their implementation technologies so that they are able to follow and absorb technological developments in this space.

Weekly Schedule … Assessment

Weekly Schedule: http://www.cse.unsw.edu.au/~cs9321

Assessment:

- 50% formal written exam: individual assessment.
- 40% on assignment work
  - Ass1 on building a simple REST API (individual) 10%
  - Ass2 on building a simple Data service API (individual) 10%
  - Ass3 on building a data mashup application (group) 20%
- 10% on 7-9 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 week.
- You can do them in your own time, but you are encouraged to try and complete as much as you can during the class.
- Use the forum. Share what you have learned/found

Lab consultations (from Week 2 to Week 10)

- Thursday 3-5pm (Brass, ME305) – by Alireza and Seung
- Friday 3-4pm (Tabla, K17 G07) – by Seung

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 (via myUNSW with supporting docs)

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

Student Conduct

Reminder …

Course Reference Books and Resources

Reminder …

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 and Use tutor's time

Use of laptops during lectures (?)

Use of mobile phones during lectures (!)

From Week 2 – Week 5

Web applications 101

Web service (API) 101 and a bit more

Web client technologies

Data visualisation techniques (over the browser)