Introduction

What is Enterprise?

- A business Organization
- In computer industry, term is often describe as the large organization that utilized computers
- E.g. Intranet example of an enterprise computing system

⇒Enterprise Application:

- A business Application
- Business application are complex, scalable ,distributed, component based and mission critical.

Trend(better tools, greater impact, powerful option)

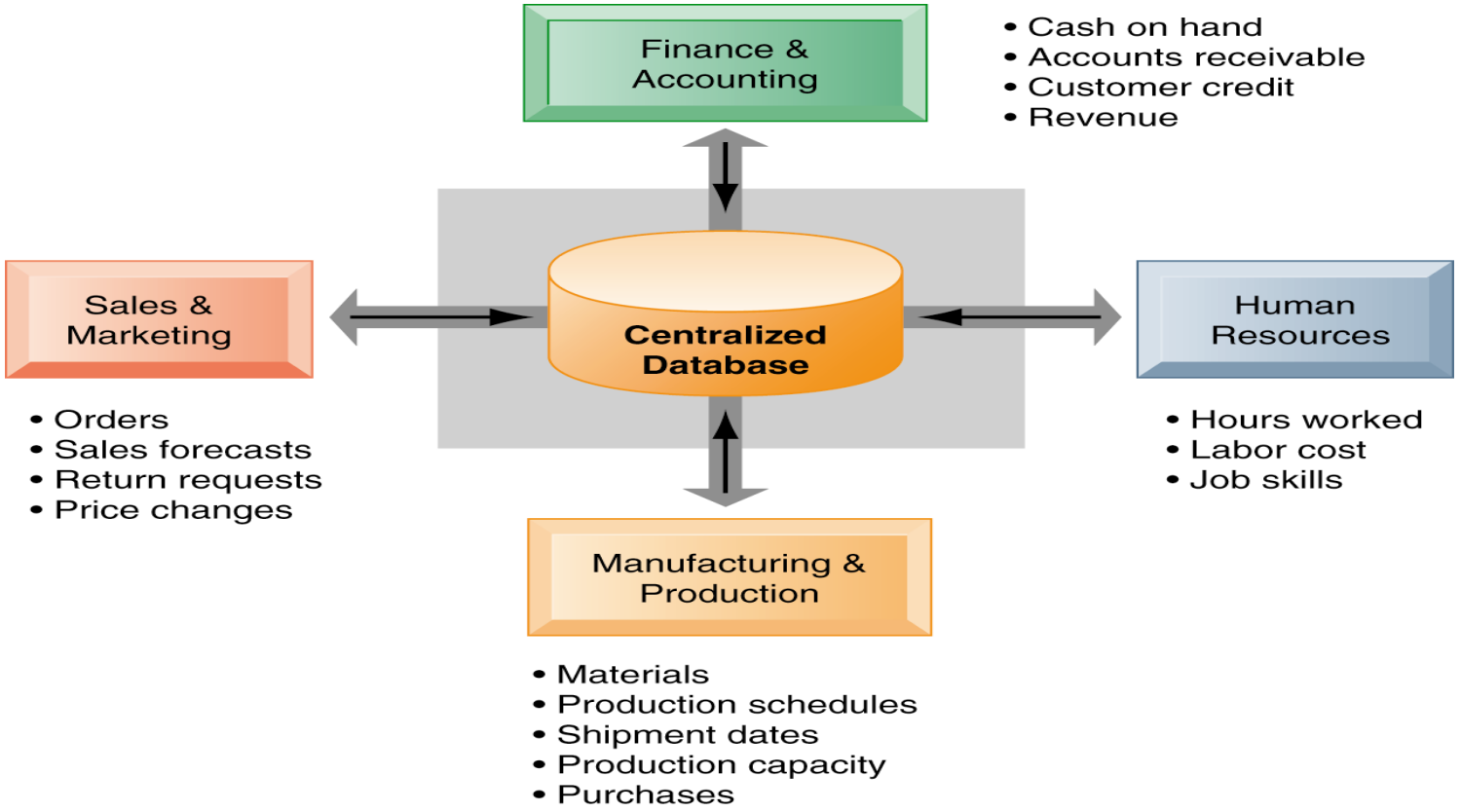
- Growth of User Experience (UX)
- Explosion of Development Tools
- Decline of Native Development
- Growing business focus on user experience
- Growing need for integration and web service
- Others…….?????

Enterprise Application Integration

Enterprise application integration (EAI) is the use of technologies and services across an enterprise to enable the integration of software applications and hardware systems. Many proprietary and open projects provide EAI solution support.EAI is related to middleware technologies. Other developing EAI technologies involve Web service integration, service-oriented architecture, content integration and business processes. Intercommunication between enterprise applications (EA), such as customer relations management (**CRM**), supply chain management (**SCM**) and business intelligence is not automated. Thus, EAs do not share common data or business rules. EAI links EA applications to simplify and automate business processes without applying excessive application or data structure changes. *However, EAI is challenged by different operating systems, database architectures and/or computer languages, as well as other situations where legacy systems are no longer supported by the original manufacturers.*

Keyword Web services and business integration

How enterprise System Works?



Enterprise systems feature a set of integrated software modules and a central database that enables data to be shared by many different business processes and functional areas throughout the enterprise.

What is EAI?

- an application coupling mechanism, to integrate individual applications into enterprise
- provides a unified interface to legacy systems
- defined as the uses of software and computer systems architectural principles to integrate a set of enterprise computer applications
- The real beauty of Enterprise Application Integration is making various separate applications work together to produce a unified set of functionality

Why EAI?

- System development over the last 20 years has tended to emphasize core functionality as opposed to integration
- Many systems are difficult to integrate with other similar systems
- Ultimately, it comes down to a cost issue. Building a system with integration in mind reduces the amount of money spent on further system development
- 30% of the IT budget is spent on linking/merging of databases and sources

Why EAI?

- It would be great if everyone used the same servers with the same operating system with the same clients, etc
- **Reality is very diverse**. We can expect a mix of mainframes, Windows, UNIX, Linux, VMS, as well as many other systems
- Getting them to work/share data together is the issue!
- The market is stocked with legacy systems designed to support single users without thought to integrating them into a larger whole.
- Many of these systems still provide value to the organization.
- Packaged applications often create their own set of issues and problems

Applying Technology

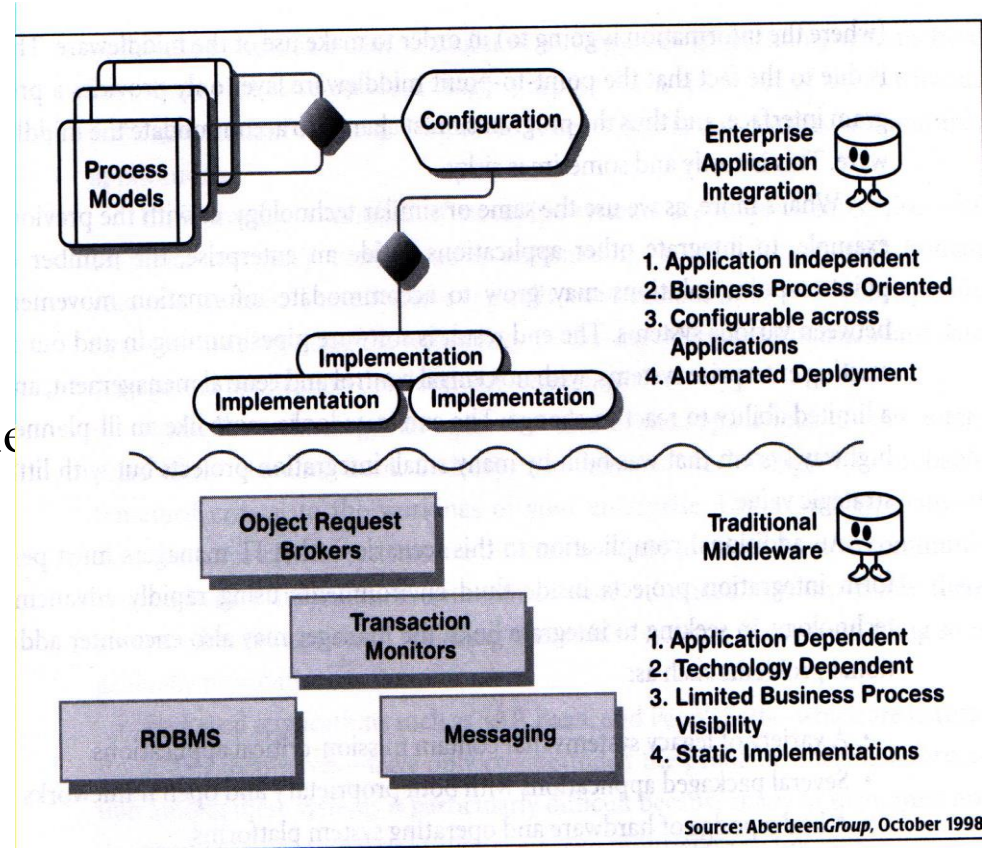
- The traditional solution to integration issues has been the introduction of 'middle-ware'.
- Middle-ware acts as a transport mechanism to perform integration typically on client/server based systems.
- Many flavors/standards of middle-ware exist:
  - Remote Procedure Calls (RPC)
  - Sockets
  - Common Object Reuse Broker Architecture (CORBA)
  - Distributed Computing Environment (DCE)
  - Java's Remote Method Invocation (RMI)

Applying Technology

- While middle-ware can be a solution, problems exist:
  - Changes are typically required to existing systems to incorporate middle-ware
  - No centralized management typically exists, so complex systems rapidly grow unmanageable
  - Technological advances tend to make middle-ware based systems look like an ill-planned highway system composed of small integration projects instead of a single over-reaching standard

Defining EAI

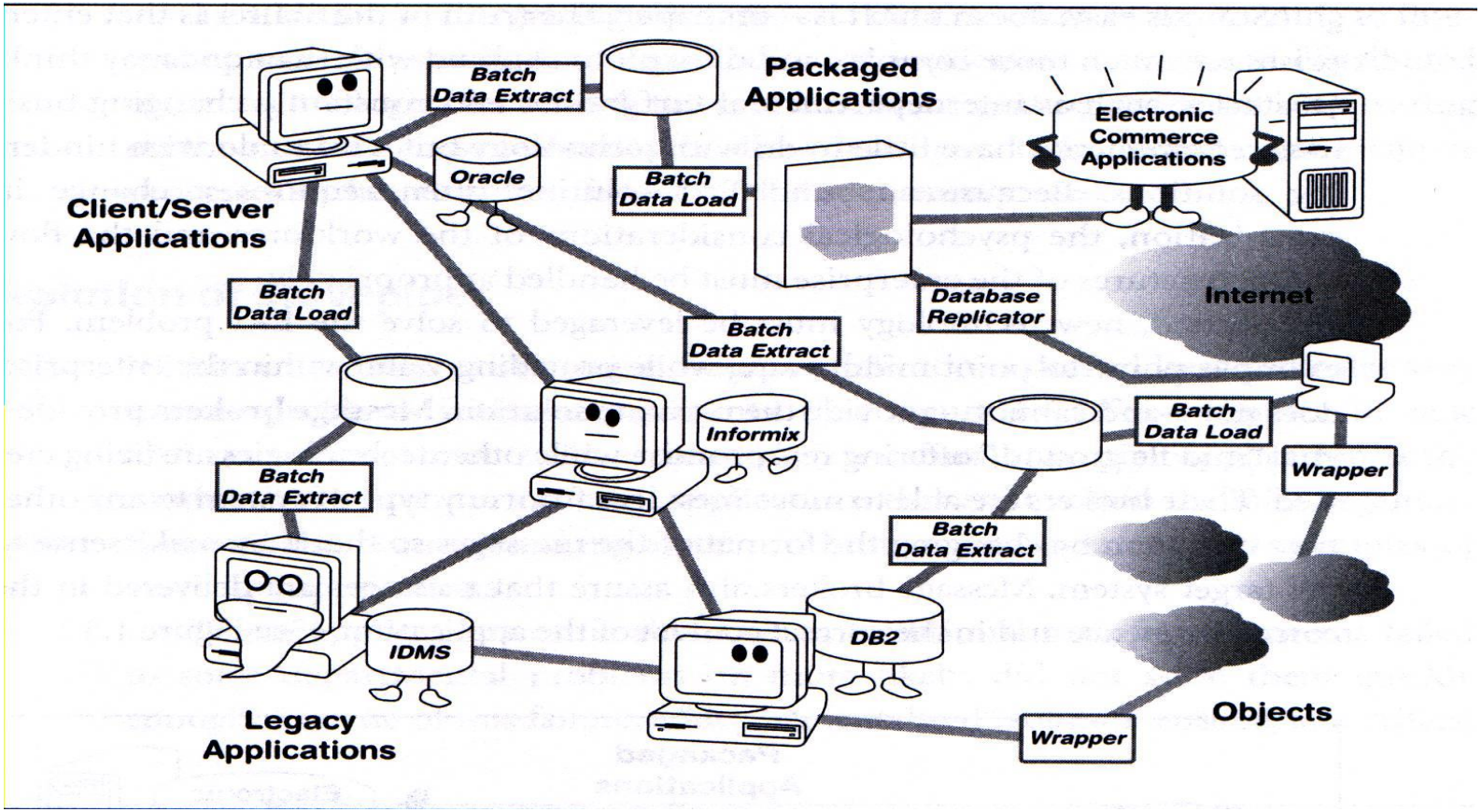
- Applying Technology
  - EAI focuses on the integration of both processes AND data while middle-ware is data oriented



Problem in EAI

- Most organizations lacked architectural foresight.
- As they upgraded from legacy systems, they moved to newer ‘open’ systems without consideration to how well these new systems would fit into their current structure and integrate with existing legacy systems.

Defining EAI

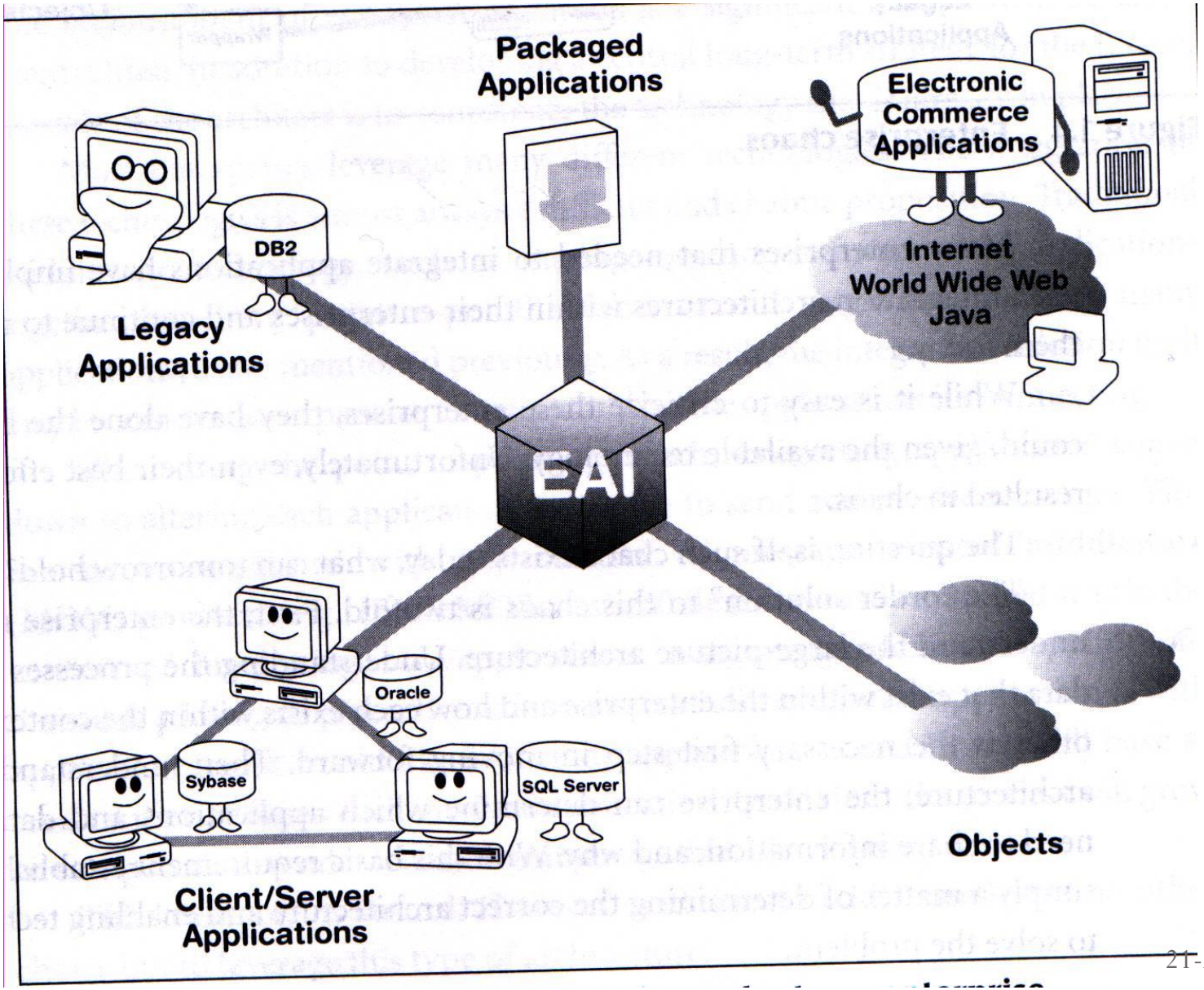


Enterprise Chaos!

How is EAI different?

- EAI focuses on the integration of both **business-level processes and data** whereas the traditional middleware approach is **data oriented**.
- EAI includes the notion of **reuse** as well as distribution of business processes and data.
- EAI allows users who understand very little about the details of the applications to integrate them

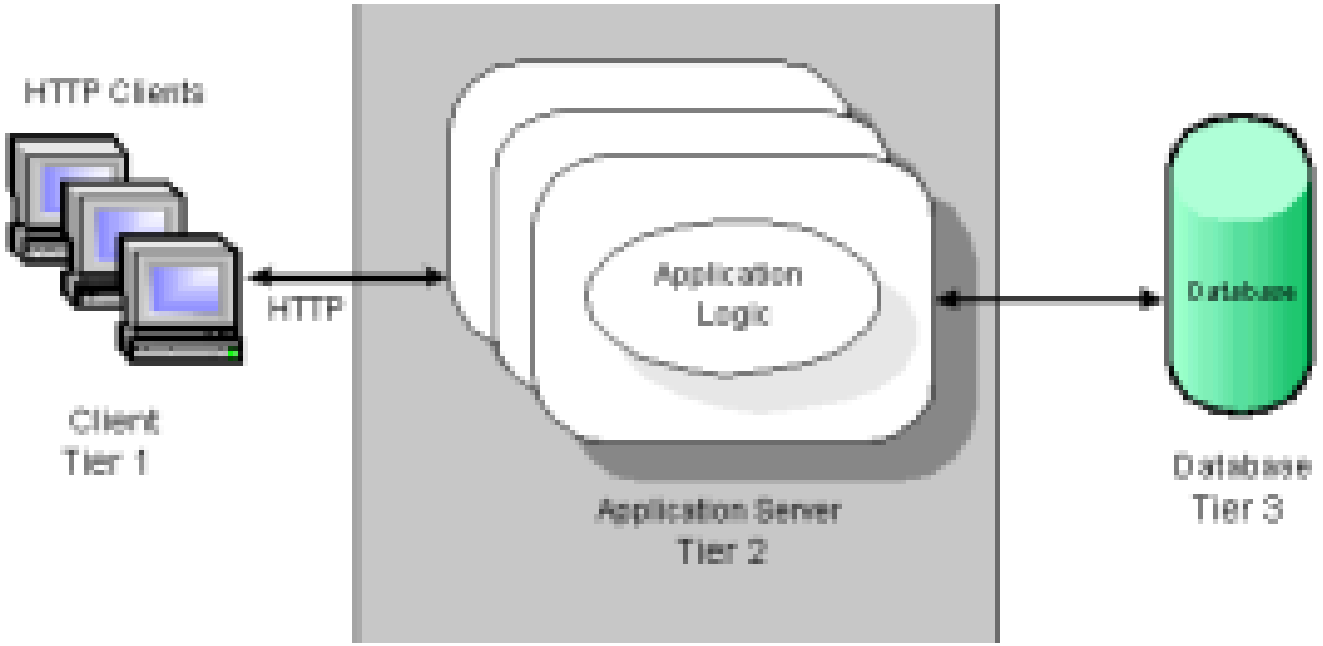
EAI Brings Order to the Enterprise



Purpose of EAI

- **Data (information) integration**: Ensuring that information in multiple systems is kept consistent. This is also known as EII (Enterprise Information Integration).
- **Process integration**: linking business processes across applications.
- **Vendor independence**: Extracting business policies or rules from applications and implementing them in the EAI system, so that even if one of the business applications is replaced with a different vendor's application, the business rules do not have to be re-implemented.
- **Common facade:** An EAI system could front-end a cluster of applications, providing a single consistent access interface to these applications and shielding users from having to learn to interact with different applications.

Multitier Architecture



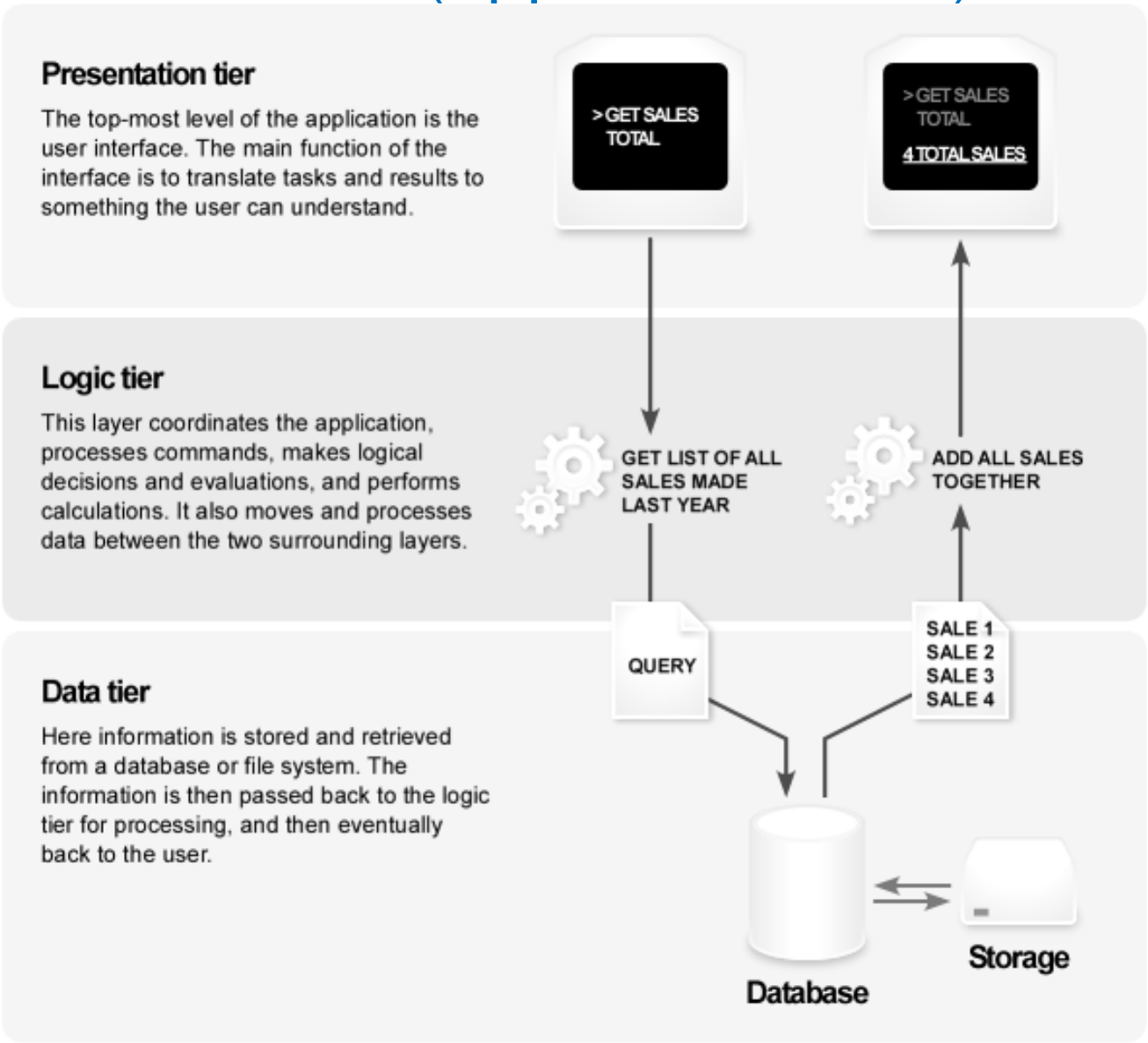
Multitier architecture :

In software engineering, **multi-tier architecture** (often referred to as **n-tier architecture**) is a client–server architecture in which presentation, application processing, and data management functions are logically separated. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of multi-tier architecture is the **three-tier architecture**.

N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application. **Three-tier** architectures typically comprise a *presentation* tier, a *business* or *data access [logic]* tier, and a *data* tier.

(*Wikipedia : Multitier Architecture*)

3-tier architecture (application view)



Advantages of the 3-tier architecture approach :

- the ability to **separate logical components** of an application ensures that applications are easy to manage and understand.

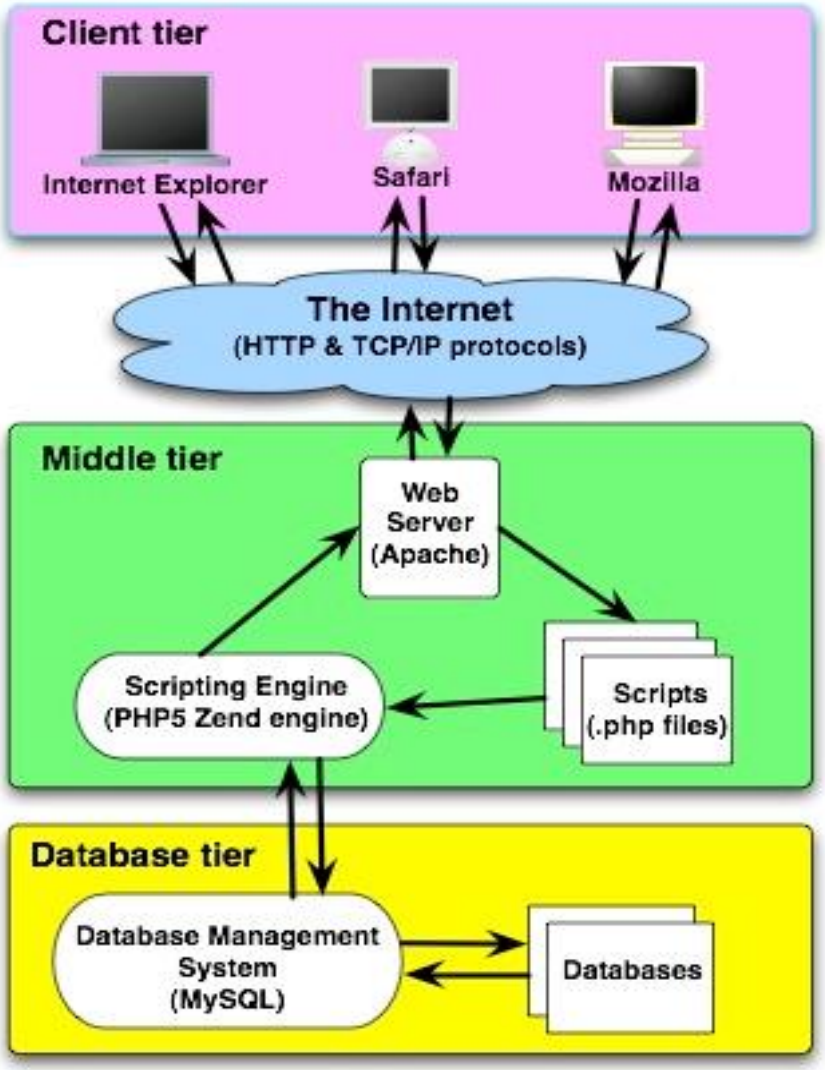
  i.e. experts can be employed that specialise in one of the layers e.g. user interface design

- because **communication can be controlled** between each logical tier of an application, changes in one tier, for example, the database access tier, do not have to affect the client component

  i.e. a change from one DBMS to another would only require a change to the component in the data access layer with little or no effect on the business/logic (middle) or UI layer.

- **specific tools and technologies suited to each layer** can be deployed (and may evolve at a different pace) .

Typical web-oriented 3-tier architecture



Web-oriented 3-tier architecture: tools & technologies

- **Presentation tier** – Browser / custom client, Client Side Scripting (JavaScript, ActionScript, VBScript etc.), Applets.

- **Logical Tier** – Web Server (Apache, IIS, Websphere etc.); Scripting Languages (PHP, Perl etc.), Programming Languages (Java, C, C# etc), Application Frameworks (Ruby on Rails etc.)

- **Data Tier** – Database Management System (DBMS) (Oracle, MySQL, SQL Server, DB2 etc.), XMLDB

N-Tier Architecture - Advantages

- Scalable:
  - this is due to its capability of multiple tier deployment and the tier decoupling it brought.
  - For example,
    - the data tier can be scaled up by database clustering without other tiers involving.
    - The web client side can be scaled up by load-balancer easily without affecting other tiers.
    - Windows server can be clustered easily for load balancing and failover.
    - In addition, business tier server can also be clustered to scale up the application, such as Weblogic cluster in J2EE.
- Better and finer security control to the whole system:
  - we can enforce the security differently for each tier if the security requirement is different for each tier.
  - For example,
    - business tier and data tier usually need higher security level than presentation tier does, then we can put these two high security tiers behind firewall for protection.
    - 1 or 2 tiers architecture cannot fully achieve this purpose because of a limited number of tiers.
    - Also, for N-Tier architecture, users cannot access business layer and data layer directly, all requests from users are routed by client presenter layer to business layer, then to data layer. Therefore, client presenter layer also serves as a proxy-like layer for business layer, and business layer serves as a proxy-like layer for data layer. These proxy-like layers provides further protection for their layers below.
- Better fault tolerance ability:
  - for example,
    - the databases in data layer can be clustered for failover or load balance purpose without affecting other layers.

- Independent tier upgrading and changing without affecting other tiers:
  - in object-oriented world, Interface-dependency implementation can decouples all layers very well so that each layer can change individually without affecting other layers too much.
  - Interface-dependency means a layer depends on another layer by interfaces only, not concrete classes.
  - Also, the dependency of a layer only on its directly-below layer also minimizes the side effect of a layer's change on the whole system.
  - For example,
    - if keep the interfaces unchanged, we can update or replace the implementation of any layer independently without affecting the whole system.
    - Due to the changing of business requirement and technology, changing the implementation of a layer to another totally different one does happen often.
- Friendly and efficient for development:
  - the decoupled layers
    - are logic software component groups mainly by functionality,
    - are very software development friendly and efficient.
  - Each layer
    - can be assigned individually to a team who specializes in the specific functional area; a specialized team can handle the relevant task better and more efficiently.

- Friendly for maintenance:
  - N-Tier architecture groups different things together mainly by functionality and then makes things clear, easily understandable and manageable.
- Friendly for new feature addition:
  - due to the logical grouped components and the decoupling brought by N-Tier architecture, new features can be added easily without affecting too much on the whole system.
- Better reusability:
  - due to the logically grouped components and the loose couplings among layers.
    - Loosely-coupled component groups are usually implemented in more general ways, so they can be reused by more other applications.

The Disadvantages of the N-Tier Deployment

- The performance of the whole application
  - may be slow if the hardware and network bandwidth aren't good enough because more networks, computers and processes are involved.
- More cost for hardware, network, maintenance and deployment
  - because more hardware and better network bandwidth are needed.

Tier and Layer

- Tier
  - the physical deployment computer.
    - Usually an individual running server is one tier.
    - Several servers may also be counted as multiple tier, such as server failover clustering.
- Layer
  - logic software component group mainly by functionality;
  - is used for software development purpose.
- Tier and Layer
  - Layer software implementation has many advantages and is a good way to achieve N-Tier architecture.
  - Layer and tier may or may not exactly match each other.
    - Each layer may run in an individual tier.
    - Multiple layers may also be able to run in one tier.
    - A layer may also be able to run in multiple tiers.
  - For example, in Diagram 2 below, the persistence layer in .NET can include two parts: persistence Lib and WCF data service, the persistence lib in the persistence layer always runs in the same process as business layer to adapt the business layer to the WCF data service. However, the WCF data service in persistence layer can run in a separate individual tier.
  - Here is another example: we may extract the data validation in business layer into a separate library (but still kept in business layer), which can be called by client presenter layer directly for a better client-side interactive performance. If this occurs, then data validation part of the business layer runs in the same process of the client presenter layer, the rest of business layer runs in a separate tier.

Model View Controller (MVC)

Smalltalk-80™

- In the MVC paradigm, the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of objects, each specialized for its task.
  - The **view** manages the graphical and/or textual output to the portion of the bitmapped display that is allocated to its application.
  - The **controller** interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate.
  - The **model** manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

Smalltalk-80™ *continued*

- The formal separation of these three tasks is an important notion that is particularly suited to Smalltalk-80 where the basic behavior can be embodied in abstract objects: *View*, *Controller*, *Model* and *Object*.

Sun says

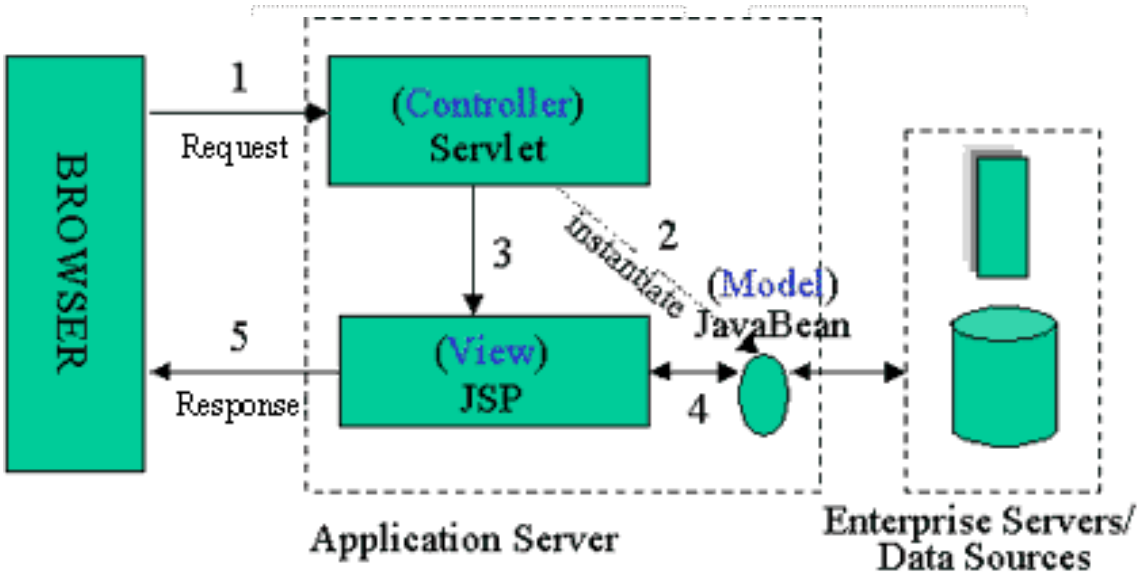
- Model-View-Controller ("MVC") is the recommended architectural design pattern for interactive applications
- MVC organizes an interactive application into three separate modules:
  - one for the application model with its data representation and business logic,
  - the second for views that provide data presentation and user input, and
  - the third for a controller to dispatch requests and control flow.

Sun continued

- Most Web-tier application frameworks use some variation of the MVC design pattern
- The MVC (architectual) design pattern provides a host of design benefits

Java Server Pages

- Model 2 Architecture to serve dynamic content
  - Model: Enterprise Beans with data in the DBMS
    - JavaBean: a class that encapsulates objects and can be displayed graphically
  - Controller: Servlets create beans, decide which JSP to return, do the bulk of the processing
  - View: The JSPs generated in the presentation layer (the browser)



OO-tips Says

- The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller.
- MVC was originally developed to map the traditional input, processing, output roles into the GUI realm:
  - Input --> Processing --> Output
  - Controller --> Model --> View

Wikipedia says

- **Model-View-Controller (MVC)** is a software architecture that separates an application's data model, user interface, and control logic into three distinct components so that modifications to one component can be made with minimal impact to the others.
- MVC is often thought of as a software design pattern. However, MVC encompasses more of the architecture of an application than is typical for a design pattern. Hence the term architectural pattern may be useful (Buschmann, et al 1996), or perhaps an aggregate design pattern.

MVC Benefits

- Clarity of design
  - easier to implement and maintain
- Modularity
  - changes to one don't affect the others
  - can develop in parallel once you have the interfaces
- Multiple views
  - games, spreadsheets, powerpoint, Eclipse, UML reverse engineering, ….

Summary (MVC)

- The intent of MVC is to keep neatly separate objects into one of tree categories
  - Model
    - The data, the business logic, rules, strategies, and so on
  - View
    - Displays the model and usually has components that allows user to edit change the model
  - Controller
    - Allows data to flow between the view and the model
    - The controller mediates between the view and model

Model

- The Model's responsibilities
  - Provide access to the state of the system
  - Provide access to the system's functionality
  - Can notify the view(s) that its state has changed

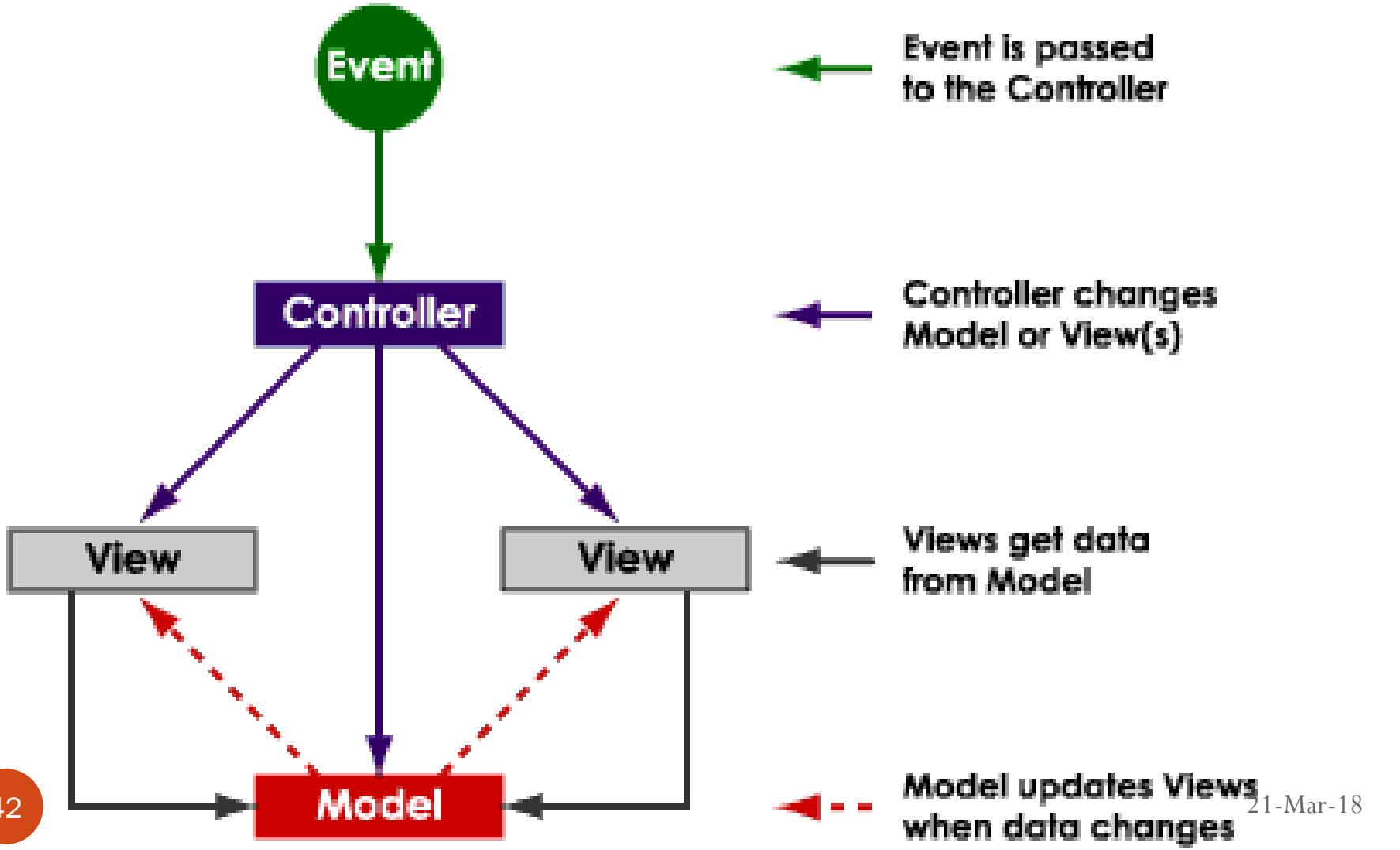
View

- The view's responsibilities
  - Display the state of the model to the user
- At some point, the model (a.k.a. the observable) must registers the views (a.k.a. observers) so the model can notify the observers that its state has changed

Controller

- The controller's responsibilities
  - Accept user input
    - Button clicks, key presses, mouse movements, slider bar changes
  - Send messages to the model, which may in turn notify it observers
  - Send appropriate messages to the view
- In Java, listeners are controllers

from http://www.enode.com/x/markup/tutorial/mvc.html)



Quiz?

Model? _____ View? _____ Controller? _____