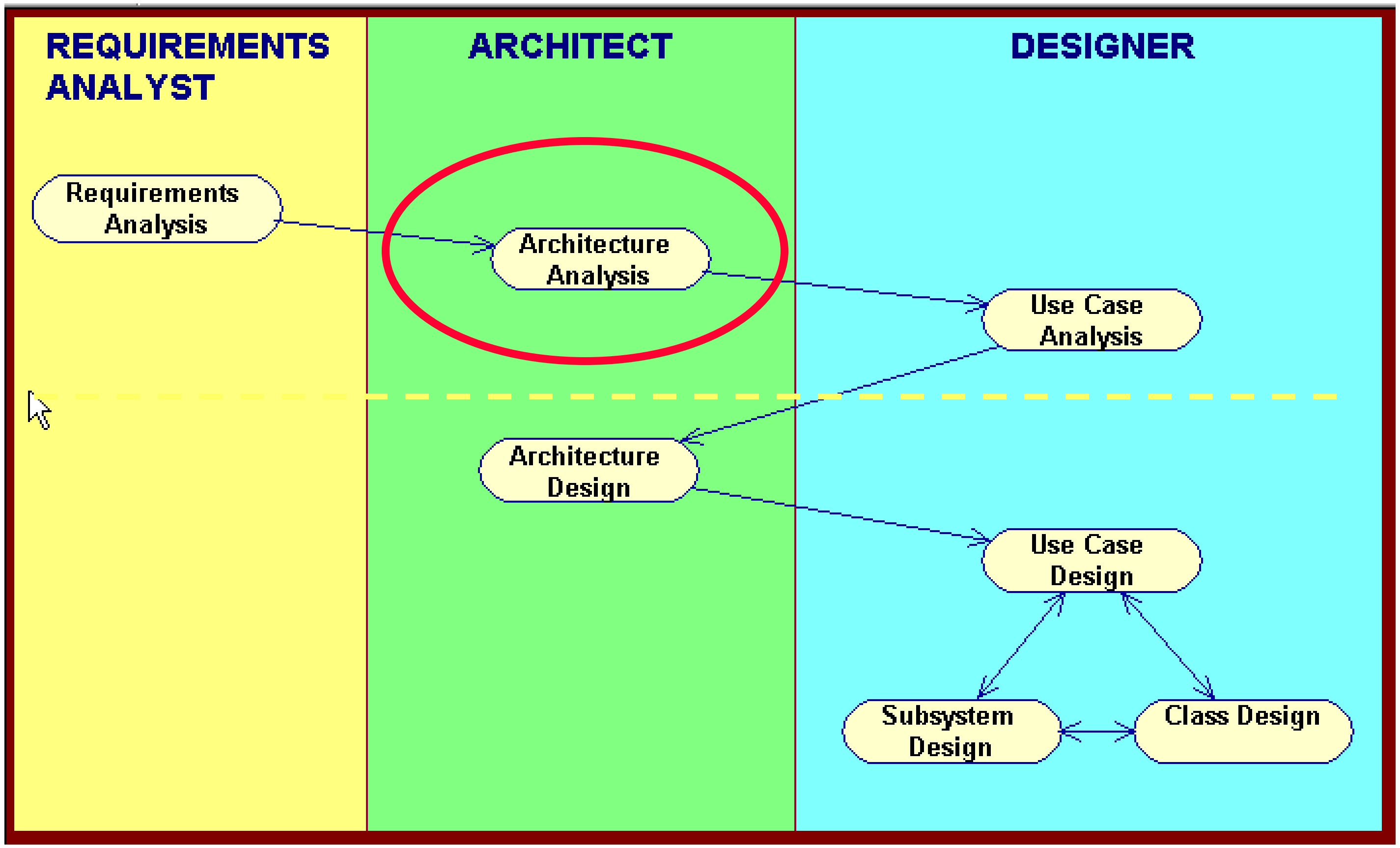
Architectural Analysis

The whole is contained in every part.

Basic RUP OOAD Activities



Architectural Analysis Topics

1. System Architecture
2. Analysis Mechanisms
3. Key Abstractions

====================================

THERE ARE THESE TWO YOUNG FISH SWIMMING ALONG AND THEY HAPPEN TO MEET AN OLDER FISH SWIMMING THE OTHER WAY, WHO NODS AT THEM AND SAYS 'MORNING, BOYS. HOW'S THE WATER?'

AND THE TWO YOUNG FISH SWIM ON FOR A BIT, AND THEN EVENTUALLY ONE OF THEM LOOKS OVER AT THE OTHER AND SAYS 'WHAT THE HECK IS WATER?'

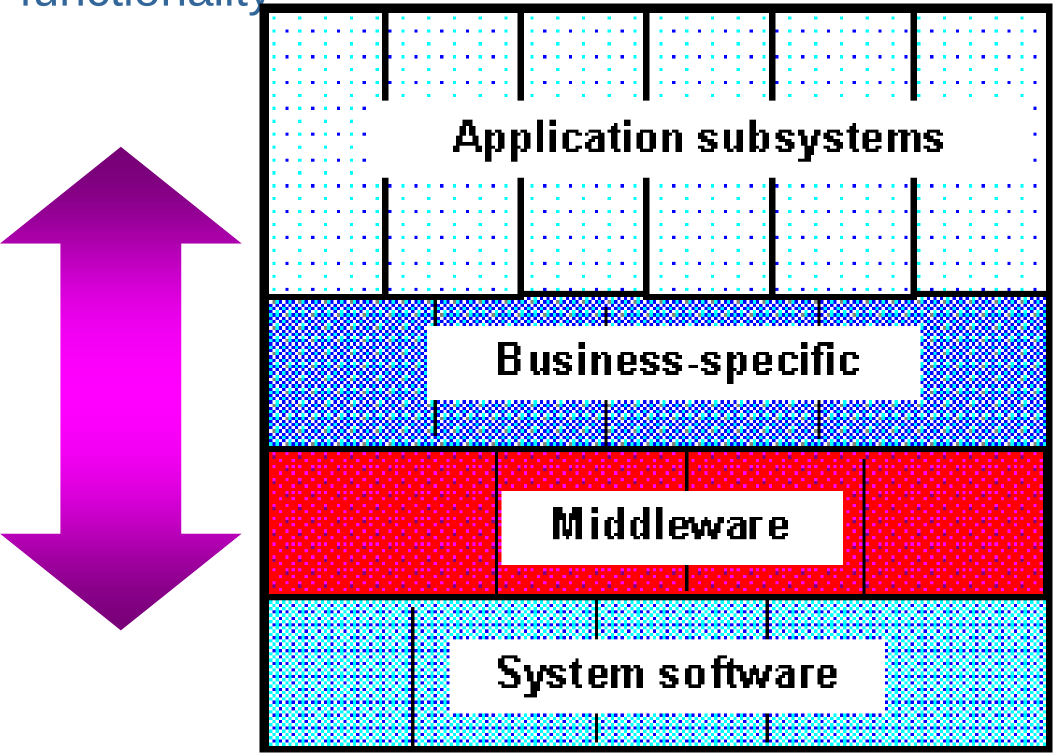
Design Patterns

- A design pattern is a customizable solution to a common design problem
  - Describes a common design problem
  - Describes the solution to the problem
  - Discusses the rationale, results and trade-offs of applying the pattern
- Design patterns provide the capability to
  - reuse successful designs
  - Capture master-class design expertise
- Architectural patterns are design patterns applied to software architecture. Examples: client-server application, enterprise architecture, web application, stand-alone desktop application.
- Architectural patterns are chosen during initial System Architecture.

Part 1 - Initial System Architecture

- Now is the time to decide if you are building a:
  - Stand-alone desktop application
  - Enterprise System
  - Web application
  - Client-server application
  - Distributed System (e.g. RESTful or SOAP solution)
  - Or some combination of the above.

Stand-alone System Layered Architecture



Specific functionality

General functionality

Distinct application subsystem that make up an application - contains the value adding software developed by the organization.

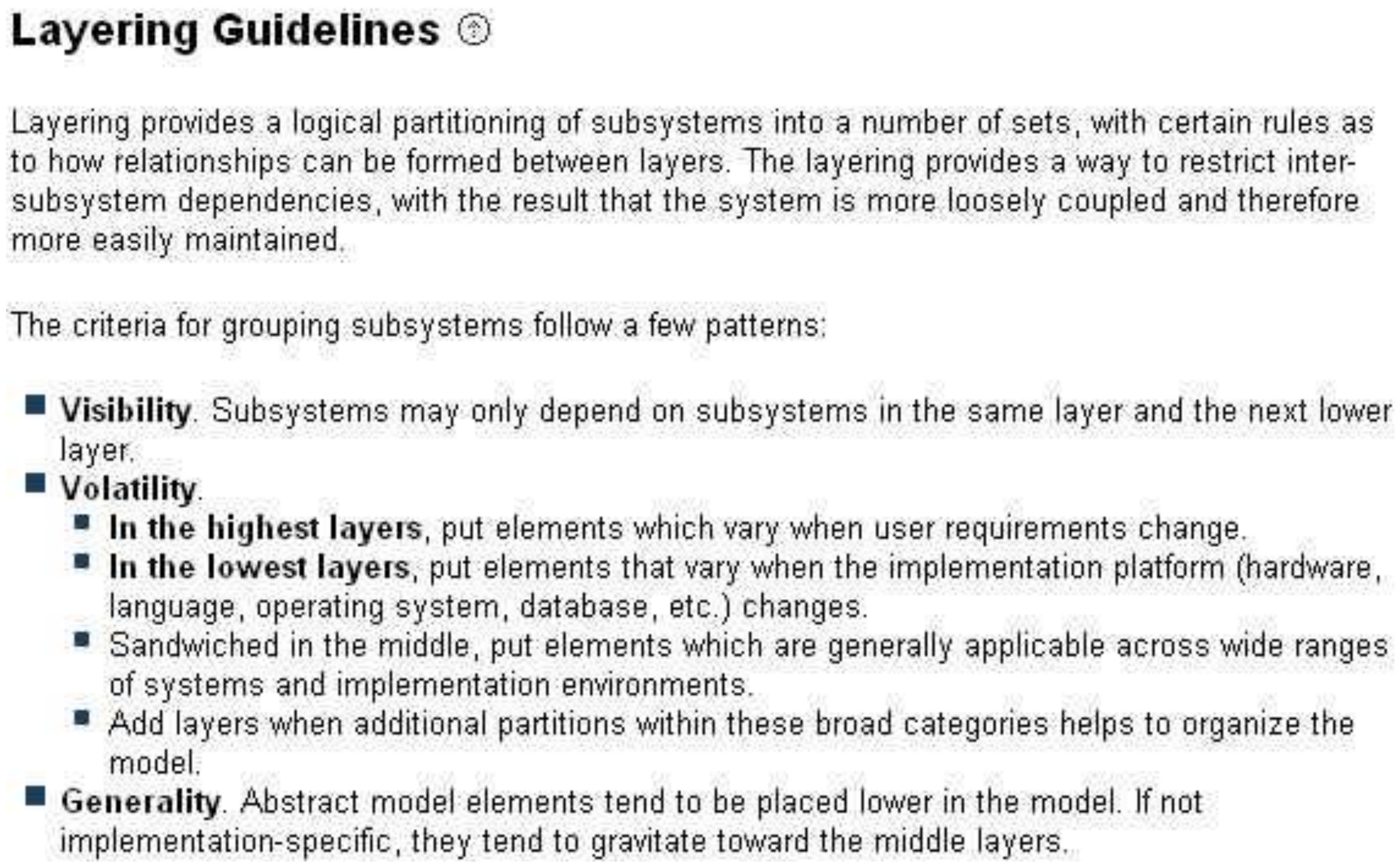
Business specific - contains a number of reusable sybsystems specific to the type of business.

Middleware - offers subsystems for utility classes and platform-independent services for distributed object computing in heterogeneous environments and so on.

System software - contains the software for the actual infrastructure such as operating systems, interfaces to specific hardware, device drivers and so on.

NOTE: During Architectural Analysis, the focus is on the upper layers – application and business layers. The middleware and system software layers will be detailed in Architectural Design.

Stand-alone Architectural Layers



Layering Guidelines

Layering provides a logical partitioning of subsystems into a number of sets, with certain rules as to how relationships can be formed between layers. The layering provides a way to restrict inter-subsystem dependencies, with the result that the system is more loosely coupled and therefore more easily maintained.

The criteria for grouping subsystems follow a few patterns:

- **Visibility**. Subsystems may only depend on subsystems in the same layer and the next lower layer.
- **Volatility**.
  - **In the highest layers**, put elements which vary when user requirements change.
  - **In the lowest layers**, put elements that vary when the implementation platform (hardware, language, operating system, database, etc.) changes.
  - Sandwiched in the middle, put elements which are generally applicable across wide ranges of systems and implementation environments.
  - Add layers when additional partitions within these broad categories helps to organize the model.
- **Generality**. Abstract model elements tend to be placed lower in the model. If not implementation-specific, they tend to gravitate toward the middle layers.

MUMSched Initial System Architecture

- Web Applications/Enterprise Architectures layering is based on a client-server architecture. The choice of making your solution a client-server solution or a stand-alone desktop solution is done during Architecture Analysis.
- At the analysis stage the layers should be the same for different implementations of client-server/enterprise architectures.
- During the architecture design stage (next week) we chose one particular implementation.

Part 2 - The Notion of Architectural Mechanisms

- A project's architectural mechanisms are standards for how common design problems are to be solved on this project
  - Promotes uniformity of solutions and increases possibility of reuse, ease of maintenance and minimizes entropy of multiple solution strategies
  - An architectural mechanism is really an architectural pattern for the project.
- Whenever possible, represent in diagrams as a placeholder
  - Short-hand representation for complex behavior

- Example: Plan to use a legacy database (since it is a known constraint) and represent the database as a placeholder in diagrams. Don't need to design the database or the infrastructure for connecting since company already knows how this should be done – the **mechanism** is known.

Sample Architecture Analysis Mechanisms

- Distribution of components on multiple tiers or via web services – e.g. SOAP or RESTful
- Transaction management
- Concurrency
- Persistence -- e.g. Object Relational Mapping - ORM
- Security -- e.g. Java Authentication and Authorization
- Error and Exception detection / handling / reporting
- Rules engine
- Wrapping of legacy systems
- Web Application Framework
- User Interface Framework

Example Analysis Mechanism - ORM

- Object-Relational Mapping allows us to gain all the advantages of Object Oriented Architecture and Design and then let our persistence manager map Entity classes to Relational Database tables, columns and rows.
- An ORM persistence manager default and elementary mapping provide a simple way to map Entities to our Database.
- With more complex mapping we can effectively map Entity relationships, dependencies, and inheritance.
- Further mapping strategies are available to optimize our design for improved performance of our database operations.

Object Relational Mapping

- During Architecture ***Design*** we will choose a particular ORM standard persistence manager and implementation (next week.)
- For Architectural Analysis we simply note that we will use some version of ORM.
- During the Architectural Analysis we do not worry about:
  - Tuning our ORM configuration and our queries to improve performance.
  - Fetching strategies, mapping class associations, relationships, etc.
- The result: our Use Case Analysis can be completed without worrying about any ORM implementation issues.

MUMSched Analysis Mechanisms

- Some example analysis mechanisms for our MUMSched project?

MUMSched Analysis Mechanisms

- Some example analysis mechanisms for our MUMSched project?
  - Use ORM for persistence
  - Web application for User Interface

Based on these decisions we just put place holders for these mechanisms during Use Case Analysis in our sequence diagrams and class diagrams.

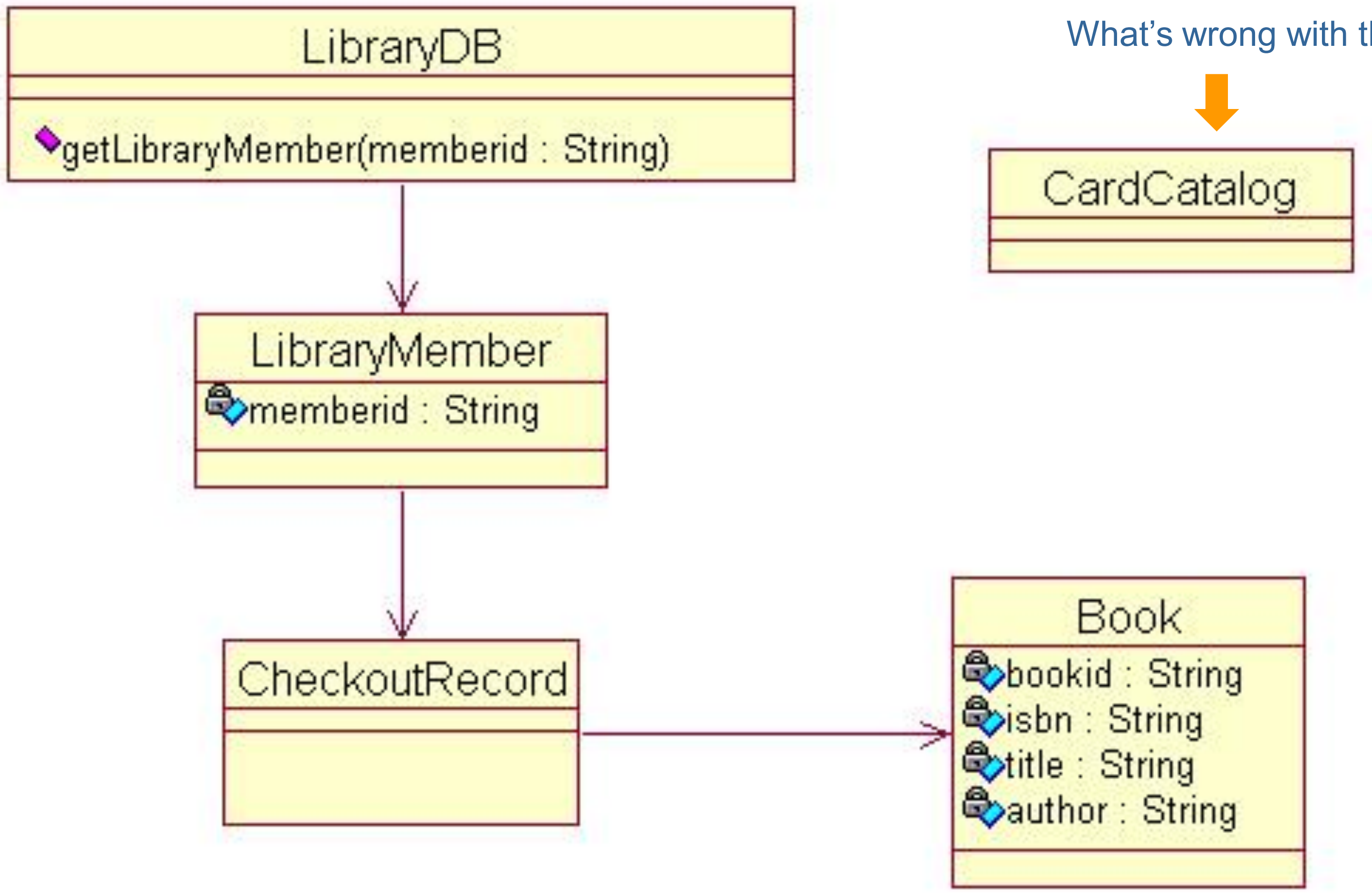
Part 3 -- Key Abstractions Guidelines

- Key abstractions are concepts identified during requirements analysis. Examples: Customer, Product, LegacyDB, Course, Section, Faculty, etc.
- Goal is "analysis classes" common across the use cases; these will be refined and fleshed out as the project evolves
- Architects will come up with these based on
  - Statement of the Problem
  - List of Requirements
  - Past experience with similar systems
  - Understanding of business domain

Key Abstractions Guidelines (cont)

- Include obvious entities acted upon by the use cases, e.g.,
  - Actors for which system has internal data (e.g., customer, student, etc; not librarian, not product manager, etc)
  - External systems which require system interaction (e.g. databases, messaging service)
  - Entities that are important parts of the business operation
    - Example: CheckoutRecord in the library example
- Include any obvious attributes and operations. Key abstraction diagrams are class diagrams.

Example: Key Abstractions



MUMSched Key Abstractions

- Each group list some key abstractions for our MUMSched project.

Review of Architectural Analysis

- Describe the rationale for defining analysis mechanisms.
- Give examples of a few MUMSched analysis mechanisms.
- How are key abstractions identified during architectural analysis? Why are they identified?

Architecture Analysis Summary

Three activities performed by system architect in preparation for use case analysis.

1. Identify an initial system architecture--preferably one that separates the project into nonchanging and changing subsets, and the changing elements should be grouped such that those grouped together tend to change together.
2. Architectural mechanisms are identified which are known solution patterns to complex problems. If we chose an enterprise system architecture we settle on an ORM and Web Application approach but not the specific frameworks.
3. Key abstractions are identified which will provide a common set of key domain elements across use case realizations.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

1. Architectural analysis produces an overall context for the rest of analysis.
2. During architectural analysis, the architect identifies inner characteristics of the system – upper layers and key abstractions – as well as the interface to external systems and strategies, in the form of analysis mechanisms.

3. **Transcendental consciousness** is the unifying basis of the our inner and outer world and all the details of life with our sense of our Self.
4. **Impulses within the transcendental field:** The unified value of pure consciousness allows to understand how the system level decisions unify all of the individual use cases of our project.