BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SS ZG653 (RL 9.1): Software Architecture

Introduction to Patterns

Instructor: Prof. SantonuSarkar

Christopher Alexander

Source:
http://en.wikipedia.org/wiki/Christopher_Alexander

- ***The Timeless Way of Building*** is a 1979 book that ties life and architecture together
- Much of SW Architecture derives from it
- ***A Pattern Language: Towns, Buildings, Construction*** is a 1977 book on architecture

What is a (Architecture) Pattern

- A set of components (or subsystems), their responsibilities, interactions, and the way they collaborate
  - Constraints or rules that decide the interaction
  - To solve a recurring architectural problem in a generic way
  - Synonymous to architecture style

– Buschmann, F. et al, Pattern Oriented Software Architecture – Volume1, Wiley, 1996

Properties of Patterns

- Addresses a recurring design problem that arises in specific design situations and presents a solution to it
- Document existing, well-proven design experience
- Identify and Specify abstractions at the high(est) level
- Provide a common vocabulary and understanding of design principles
- Helps to build complex systems
- Manage software complexity

A note on Design Principles

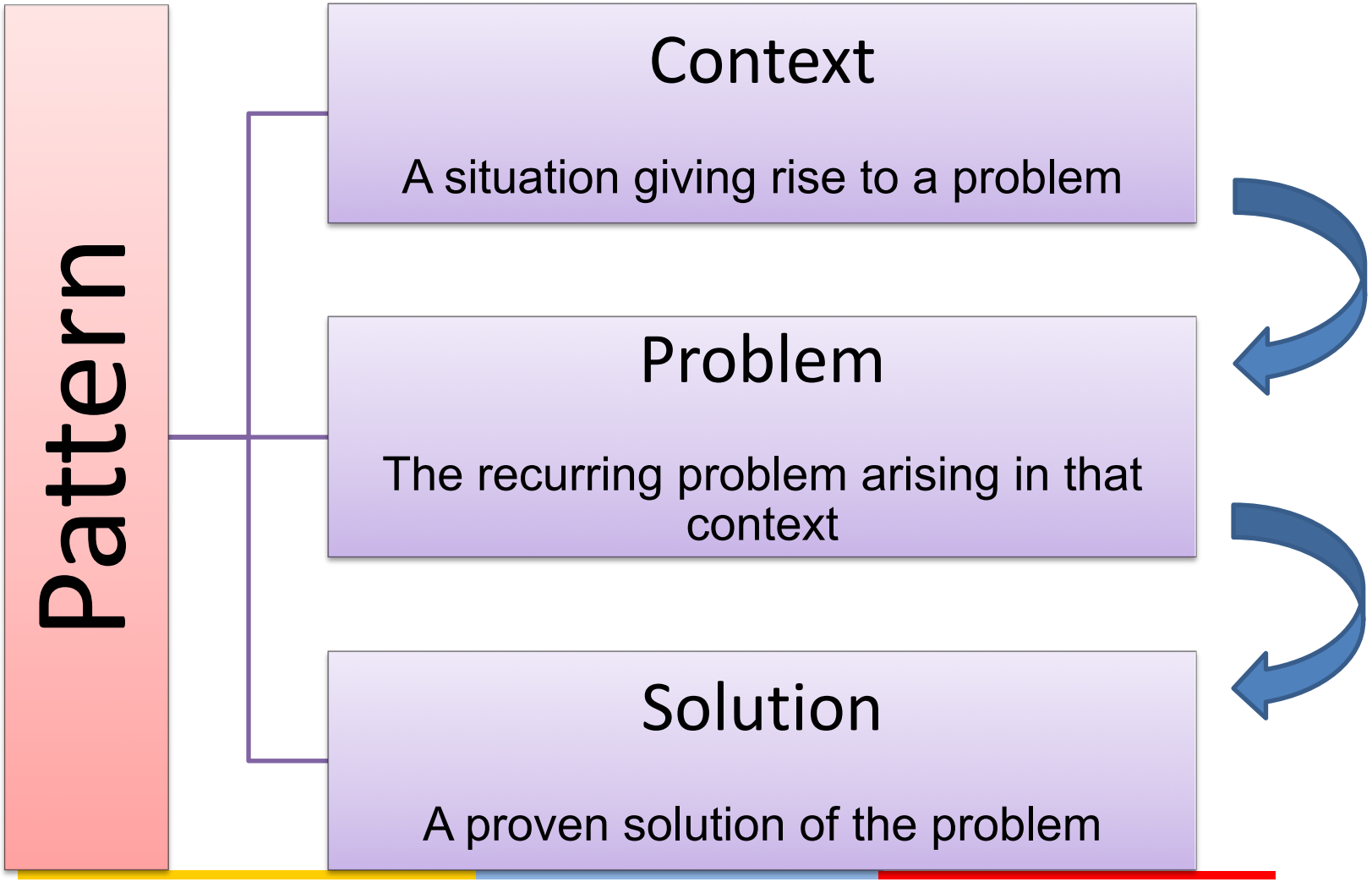
- A set of guidelines that helps to get a good design

- Robert Martin's book on Agile Software Development says
  - Avoid Rigidity (hard to change)
  - Avoid Fragility (whenever I change it breaks)
  - Avoid Immobility (can't be reused)

OO Design Principles

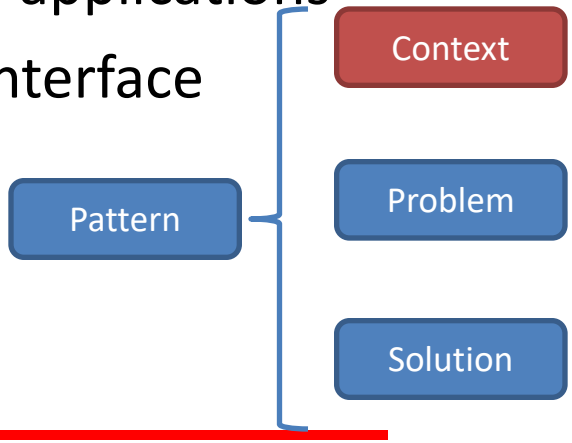
- Open close
  - Open to extension and close for modification
  - Template and strategy pattern
- Dependency inversion
  - Decouple two module dependencies (A→ B)
    - A holds the interface of B. Implementer of B implments the interface.
  - Adapter pattern
- Liskov's Substitution
  - Superclass can be replaced by subclass
- Interface Segregation
  - Don't pollute an interface. Define for a specific purpose
- Single responsibility
  - One class only one task

Pattern – Three-part Schema



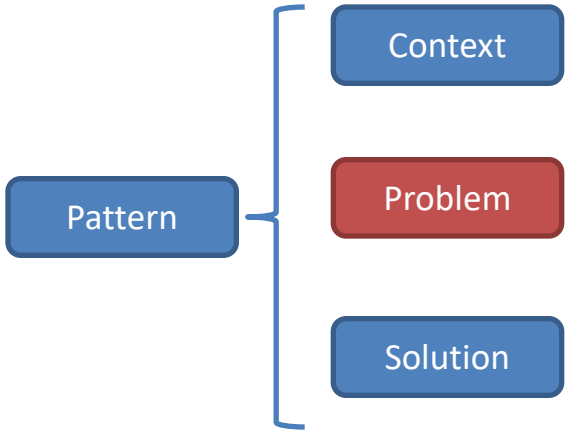
Context

- A scenario or situation where design problem arises
  - Describe situations in which the problem occurs
- Ideally the scenario should be generic, but it may not always be possible
  - Give a list of all known situations
- Example
  - Developing Messaging solution for mobile applications
  - Developing software for a Man Machine Interface



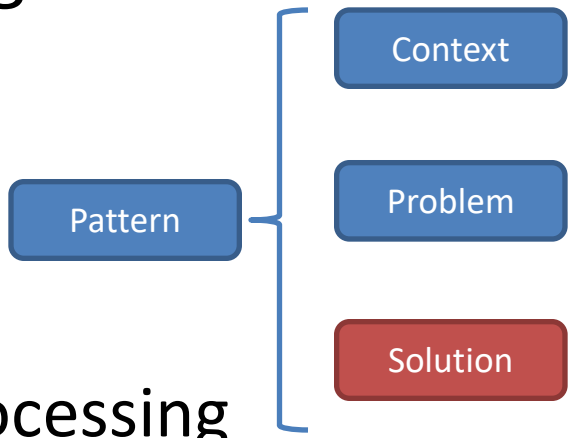
Problem

- Starts with a generic problem statement; captures the central theme
- Completed by *forces;* aspect of the problem that should be considered when solving it
  - It is a Requirement
  - It can be a Constraint
  - It can be a Desirable property
- Forces complement or contradict
- Example
  - Ease of modifying the User Interface (Personalization)



Solution

- Configuration to balance forces
  - Structure with components and relationships
  - Run-time behavior
- Structure: Addresses static part of the solution
- Run-time: Behavior while running – addresses the dynamic part
- Example
  - Building blocks for the application
  - Specific inputs events and their processing



Reference Model

- An ideal solution for a domain, comprising of only functional elements without any technology or operational platform
  - NGOSS framework in Telecom
  - Open financial services architecture based on the use of intelligent mobile devices," Electronic Commerce Research and Applications, 2008.

Architecture Style/Pattern

- A set of components (or subsystems), their responsibilities, interactions, and the way they collaborate
- address one or more quality concerns

Operational Platform and Infrastructure Pattern

- A topology of nodes where the functional blocks will be deployed at runtime
- A topology of hardware devices
  - Multi-tier pattern, IBM runtime patterns, availability patterns

Design Pattern

- software-centric solution to implement the application logic, data and the interaction. The solutions include (but not limited to) package structure, analysis patterns for data modeling
- GoF, Fowler's analysis pattern

Idioms

- Programming language level best practices

Abstract

Implementation

Pattern System

A pattern system for software architecture is a collection of patterns for software architecture, together with guidelines for their implementation, combination and practical use of software development

Pattern System

- Support the development of high-quality software systems; Functional and non-functional requirements
- It should comprise a sufficient base of patterns
- It should describe all its patterns uniformly
- It should expose the various relationships between patterns
- It should organize its constituent patterns
- It should support the construction of software systems
- It should support its own evolution

Pattern Classification

- It should be simple and easy to learn
- It should consist of only a few classification criteria
- Each classification criterion should reflect natural properties of patterns
- It should provide a ‘roadmap’
- The schema should be open to integration of new patterns

Problem Categories

Category	Description
Mud to Structure	Includes patterns that support suitable decomposition of an overall system task into cooperating subtasks
Distributed Systems	Includes patterns that provide infrastructures for systems that have components located in different processes or in several subsystems and components
Interactive Systems	Includes patterns that help to structure human-computer interaction
Adaptable Systems	Includes patterns that provide infrastructures for the extension and adaptation of application in response to evolving and changing functional requirements
Structural Decomposition	Includes patterns that support a suitable decomposition of subsystems and complex components into cooperating parts
Organization of Work	Includes patterns that define how components collaborate to provide a complex service

Problem Categories

Category	Description
Creation	Includes patterns that help with instantiating objects and recursive object structures
Service Variation	Comprises patterns that support changing the behavior of an object or component
Service Extension	Includes patterns that help to add new services to an object or object structure dynamically
Adaptation	Provides patterns that help with interface and data conversion
Access Control	Includes patterns that guard and control access to services or components
Management	Includes patterns for handling homogenous collections of objects, services and components in their entirety
Communication	Includes patterns that help organize communication between components
Resource handling	Includes patterns that help manage shared components and objects

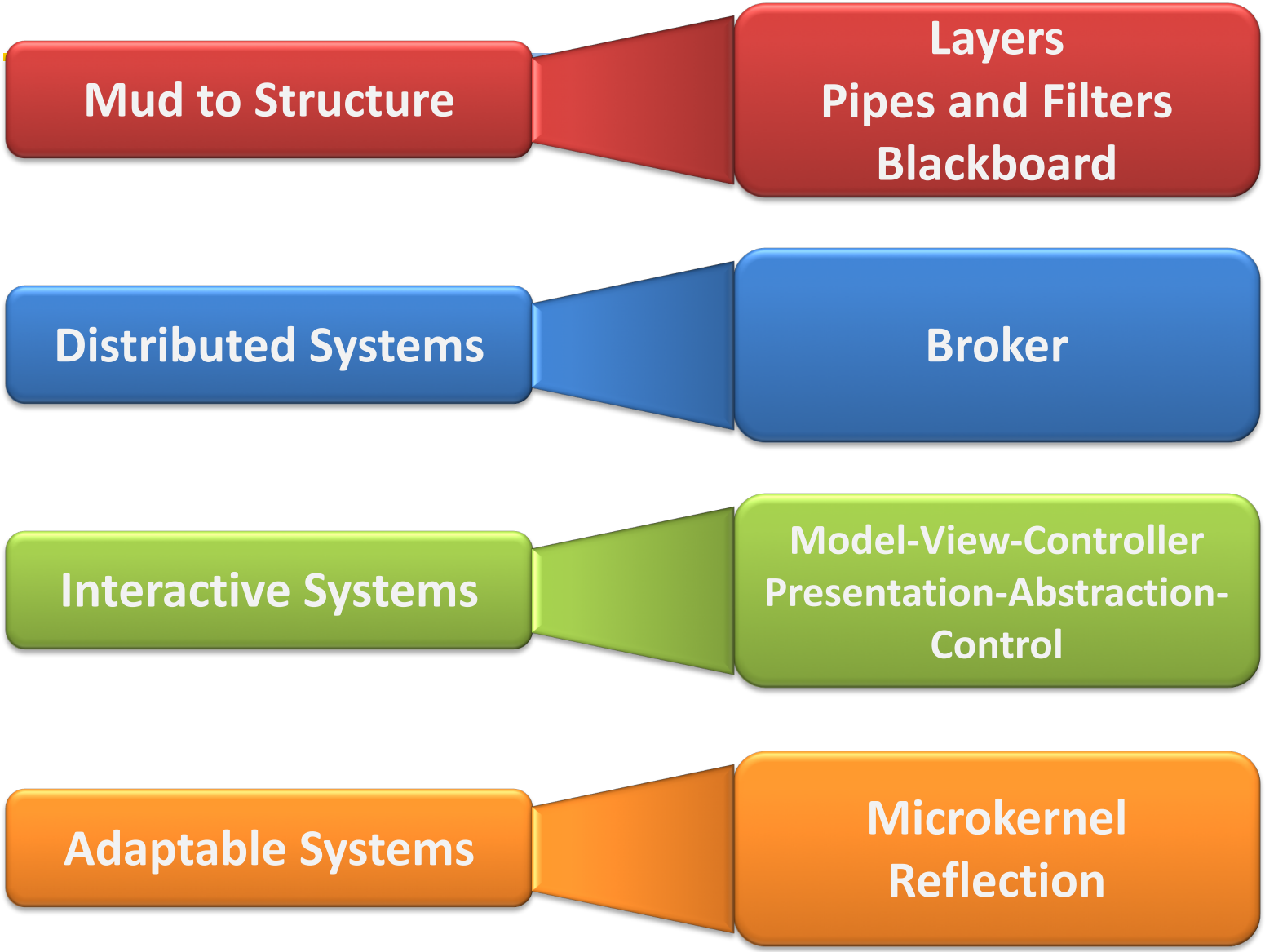
	Architectural Patterns	Design Patterns	Idioms
Mud to Structure	Layers, Pipes and Filters, Blackboard		
Distributed Systems	Broker, Pipes and Filters, Microkernel		
Interactive Systems	MVC, PAC		
Adaptable Systems	Microkernel, Reflection		
Creation		Abstract Factory, Prototype, Builder	Singleton, Factory Method
Structural Decomposition		Whole-Part, Composite	
Organisation of work		Master-Slave, Chain of Responsibility, Command, Mediator	
Access Control		Proxy, Façade, Iterator	
Service Variation		Bridge, Strategy, State	Template method
Service Extension		Decorator, Visitor	
Management		Command Processor, View Handler, Memento	
Adaptation		Adapter	
Communication		Publisher-subscriber, Forwarder-Receiver, Client-Dispatcher-Server	
Resource Handling		Flyweight	Counted Pointer

Mud to Structure

Mud to Structure

- Before we start a new system, we collect requirement from customer → transform those into specifications
  - Requirements → Architecture (Optimistic View)
- "Ball of mud" is the realization
- Cutting the ball along only one aspect (like along lines visible in the application domain may not be of help)
  - Need to consider functional and non-funcational attributes

Architectural Patterns



Thank You

SS ZG653 (RL 9.3) : Software Architecture

Layering Pattern

Instructor: Prof. SantonuSarkar

BITS Pilani

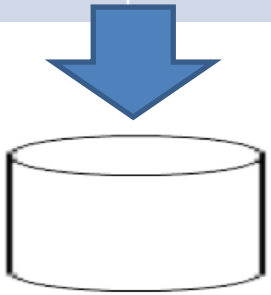
Pilani|Dubai|Goa|Hyderabad

Petstore example again…

Suppose that the store should provide the capability for a user to
Browse the catalog of products
Select a product and put it in shopping cart

Product is stored in a Table

Name	Category	Age	Price
Labrador	Dog	1	3500
Pug	Dog	1.5	1500
Goldfish1	Fish	0.5	50

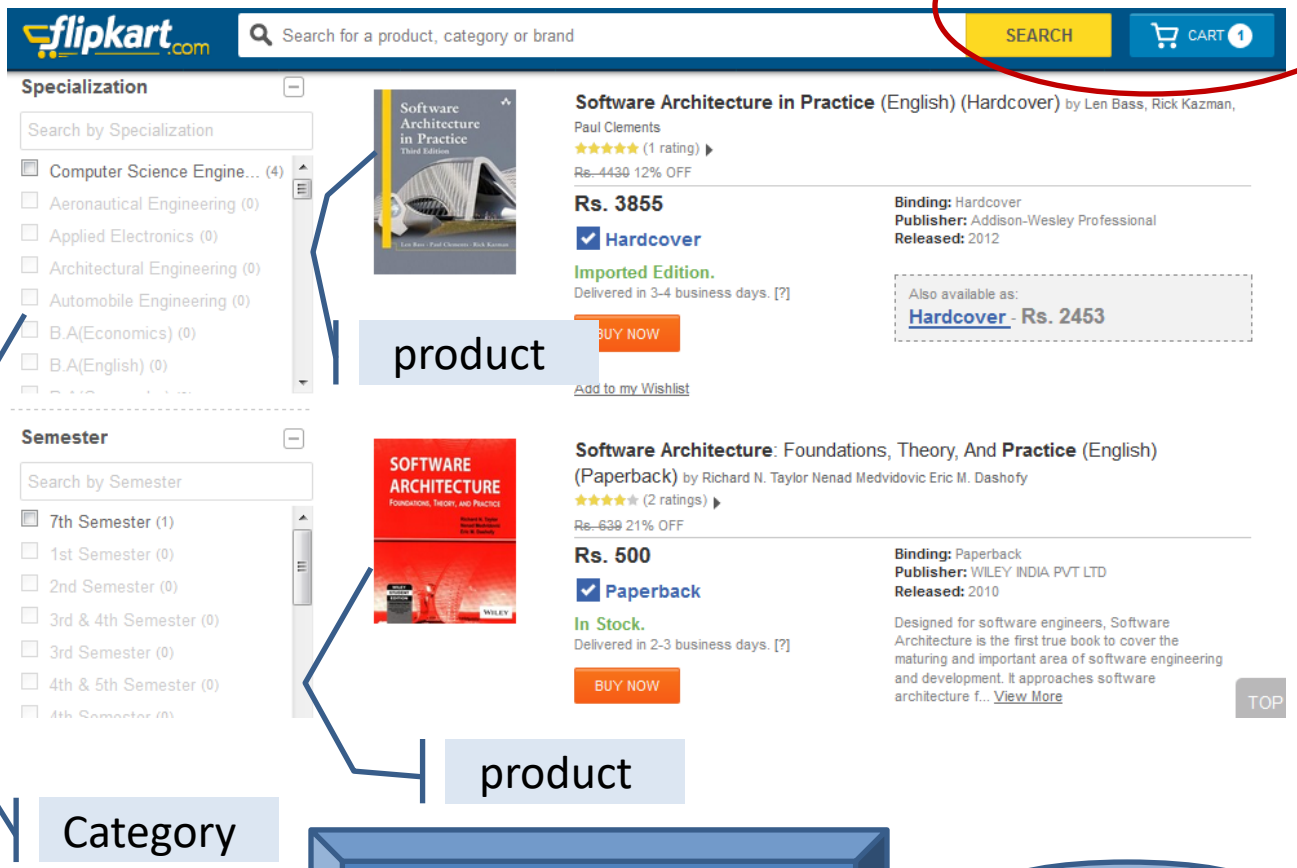


When you implement, it will look like Flipkart or Amazon…

When application runs, it displays products based on category

User Selects a productand puts it shopping cart

Many users accessing the application



What you need at a minimum?

- Three sets of classes
  - One set manages display of products, ease of selection, navigation
  - Another set manages the product management, pricing
  - Another set manages the database access
- UI Layer classes
- Business Layer classes
- Database Layer classes

Layers Architectural Pattern

Helps to structure application that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction

Layers

- Implementing protocols
- Conceptually different issues split into separate, interacting layers
- Functionality decomposed into layers; helps replace layer(s) with better or different implementation

Layers – 3 part schema

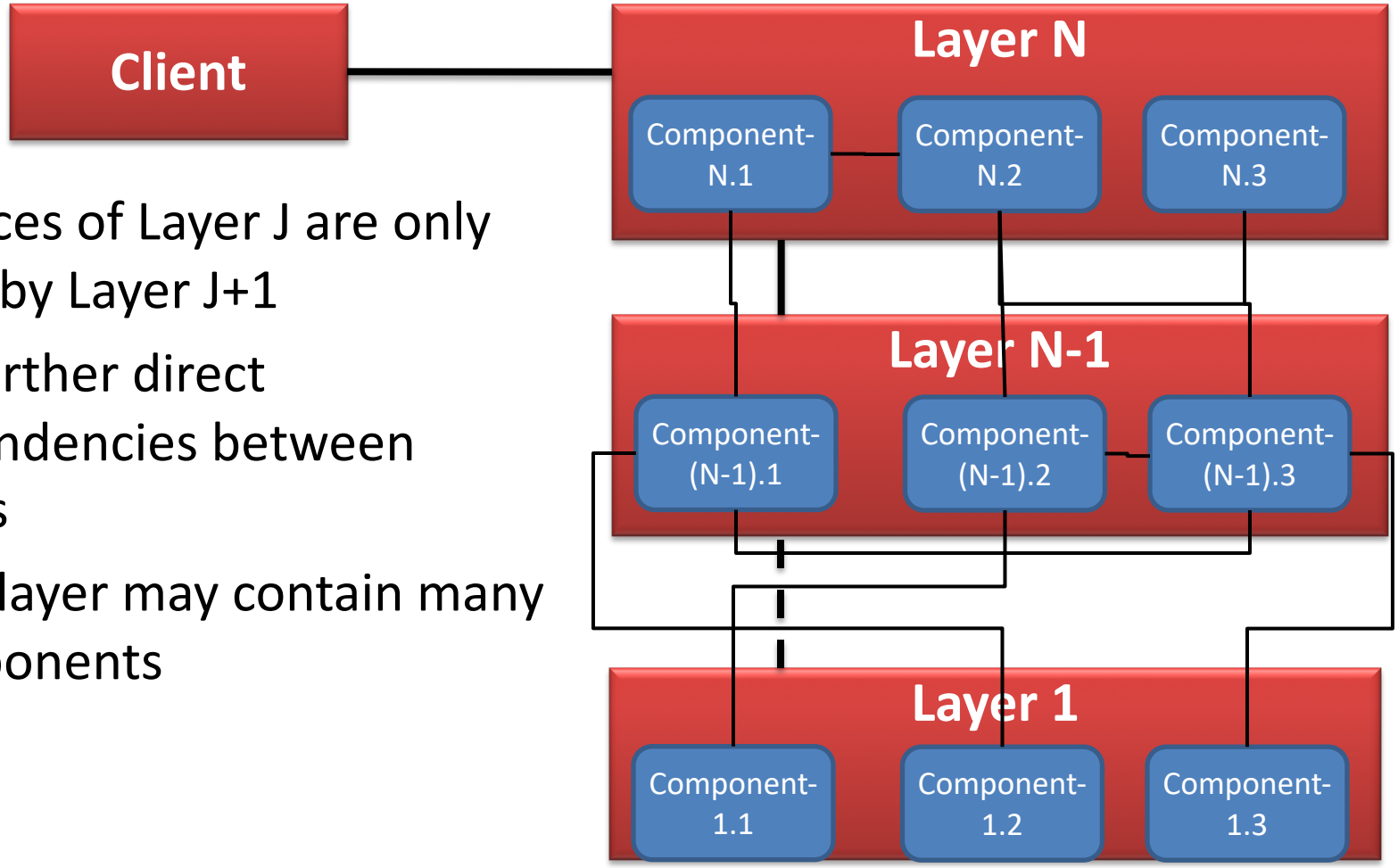
Context	**A large system that requires decomposition**
Problem	Mix of low- and high-level issues, where high-level operations rely on low-level ones A typical pattern of communication flow consists of requests moving from high level to low level, and answers to requests, incoming data and notification about events traveling in the opposite direction
Forces	•Code changes should not ripple through the system •Stable interfaces; standardization •Exchangeable parts •Grouping of responsibilities for better understandability and maintainability
Solution	Structure the system into appropriate number of layers

OSI 7-Layer Model

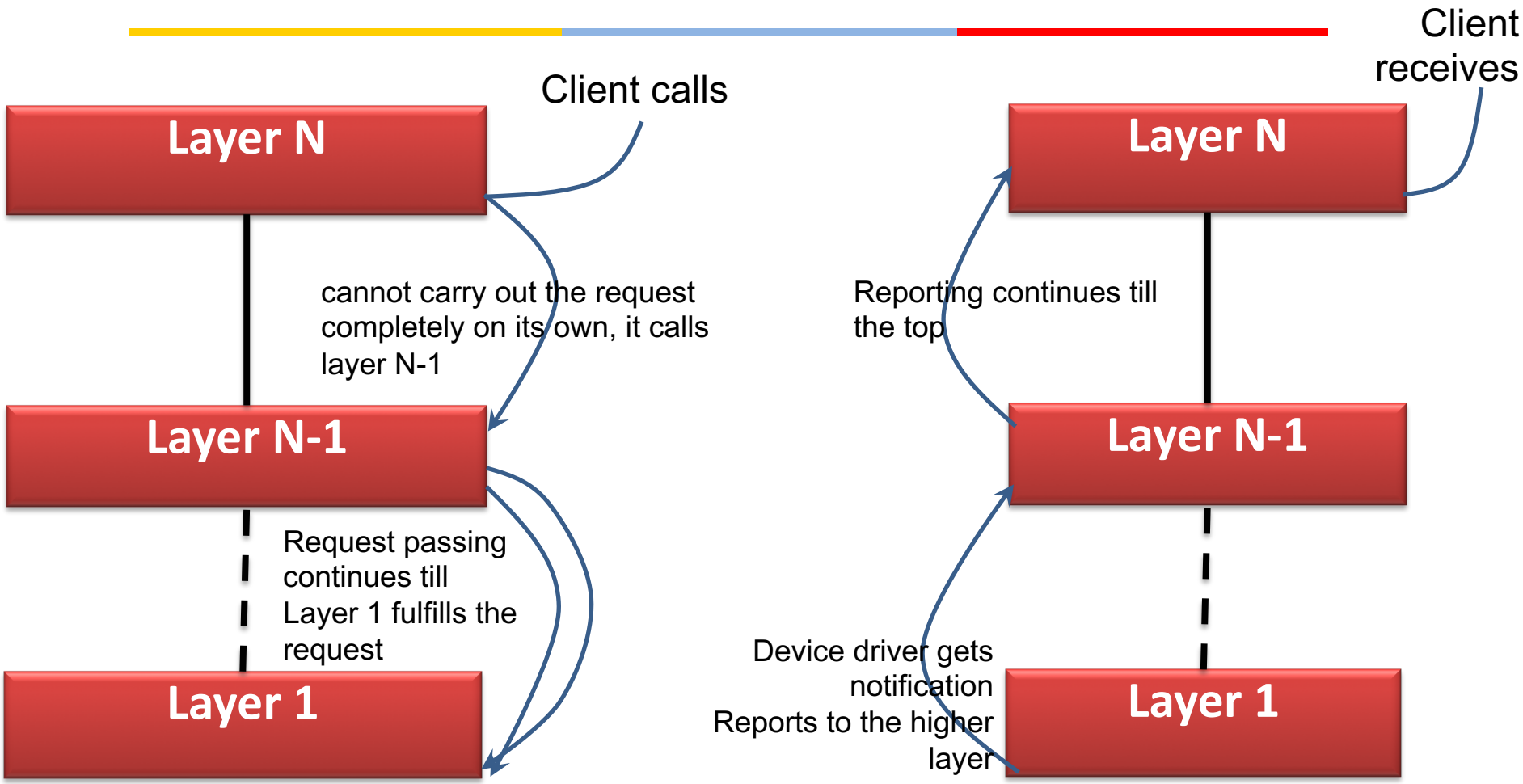
Application	Layer 7	Provides miscellaneous protocols for common activities
Presentation	Layer 6	Structures information and attaches semantics
Session	Layer 5	Provides dialog control and synchronization facilities
Transport	Layer 4	Breaks messages into packets and guarantees delivery
Network	Layer 3	Selects route from sender to receiver
Data Link	Layer 2	Detects and corrects errors in bit sequences
Physical	Layer 1	Transmits bits: velocity, bit-code, connection etc.

Layers

- Services of Layer J are only used by Layer J+1
- No further direct dependencies between layers
- Each layer may contain many components



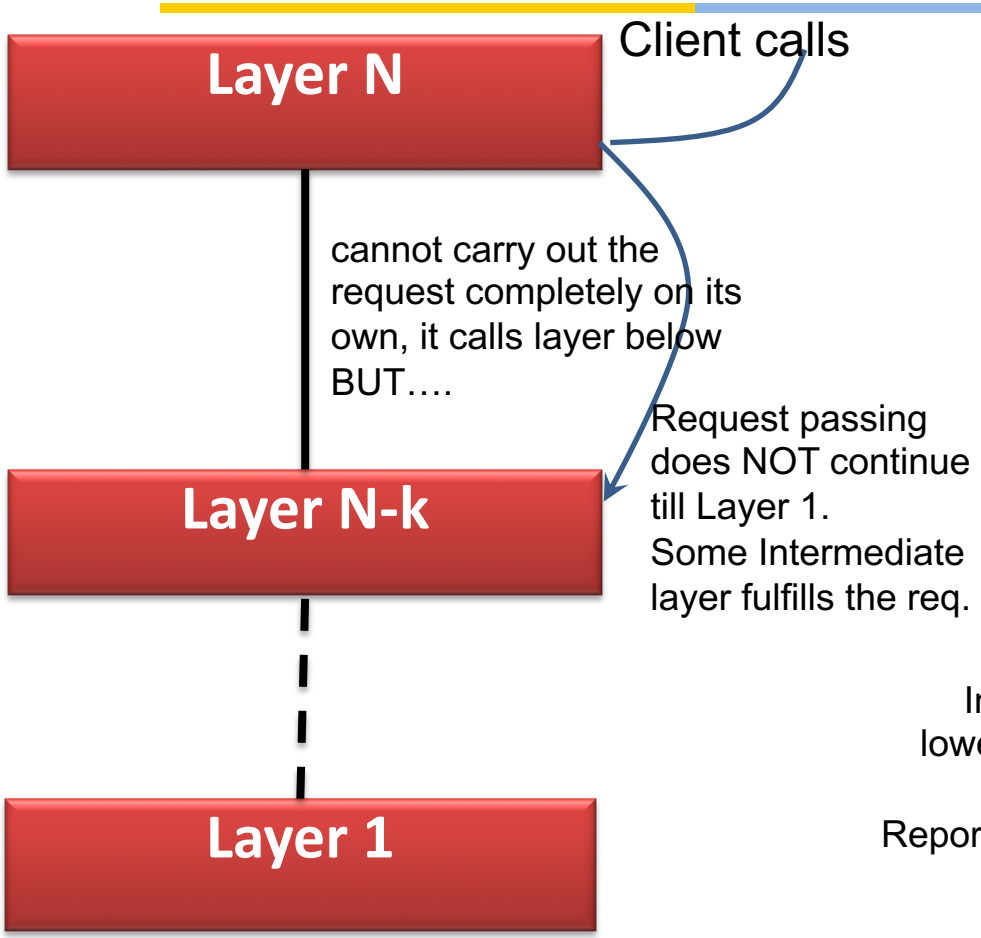
Dynamics



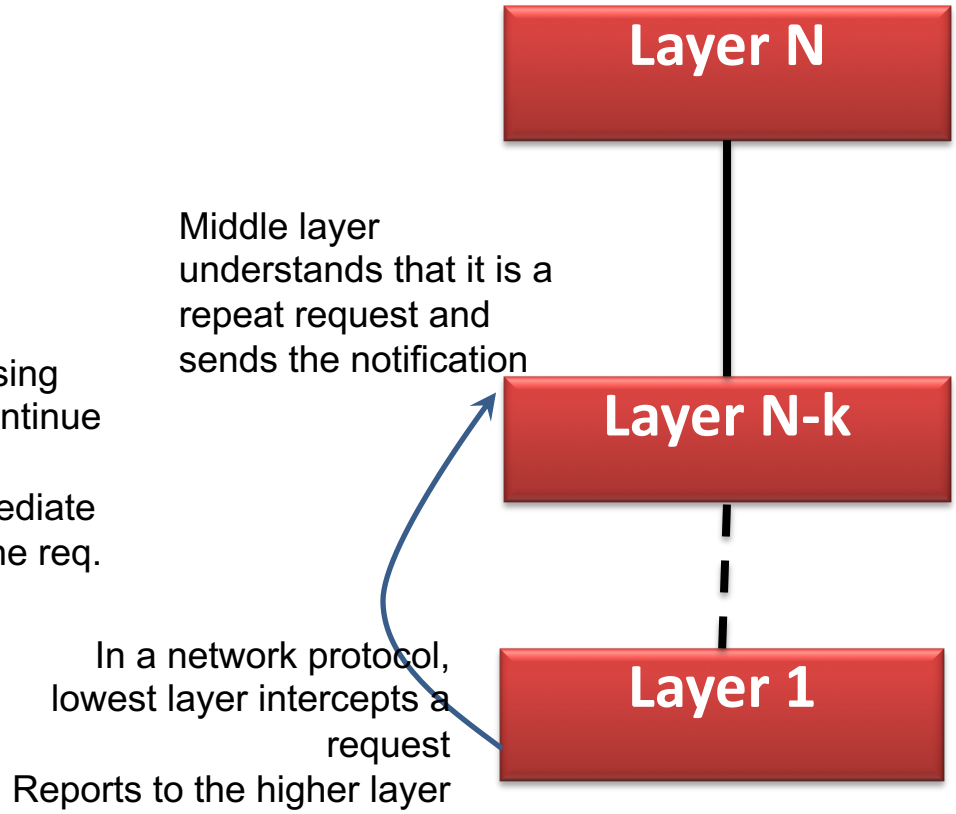
Scenario - I

Scenario - II

Dynamics



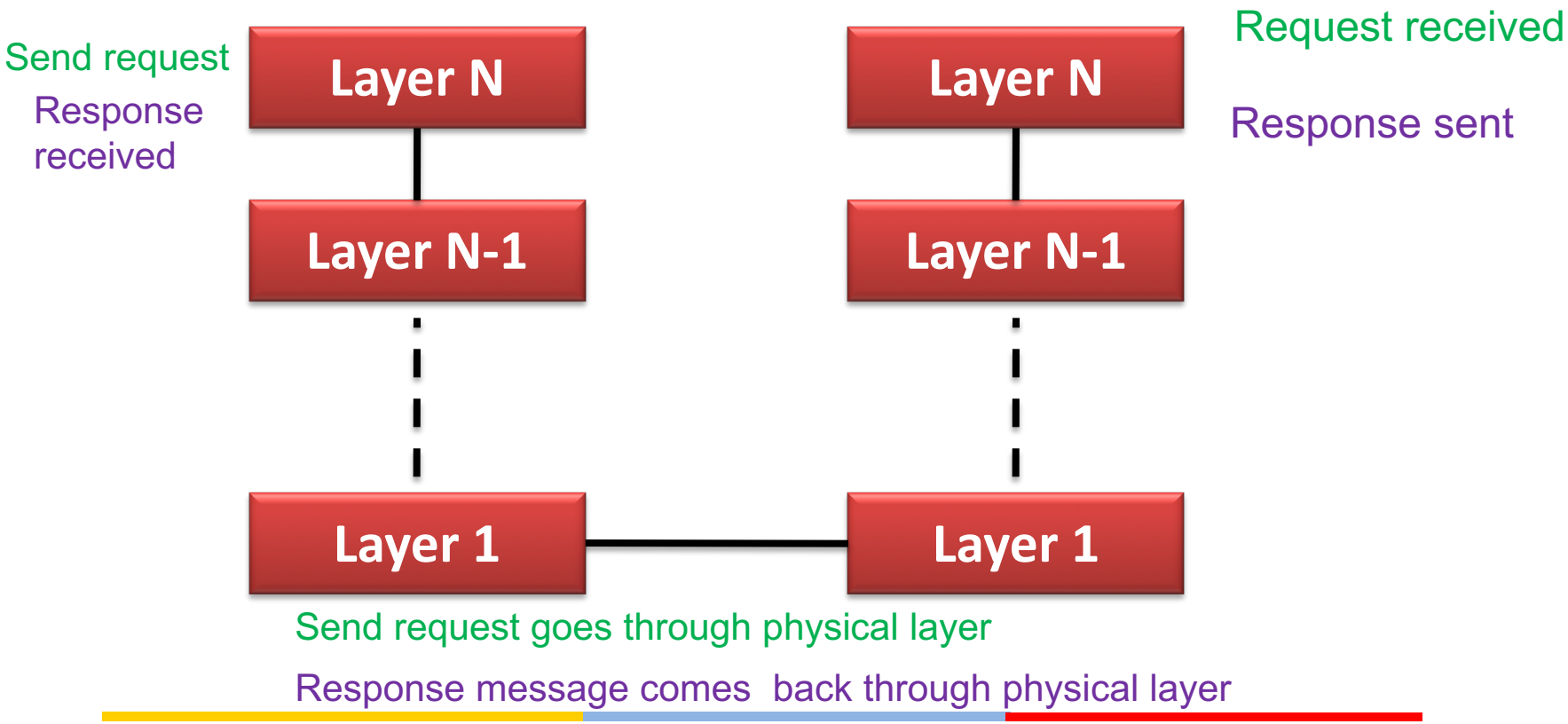
Scenario – III (layer of caching, where An intermediate layer returns the data)



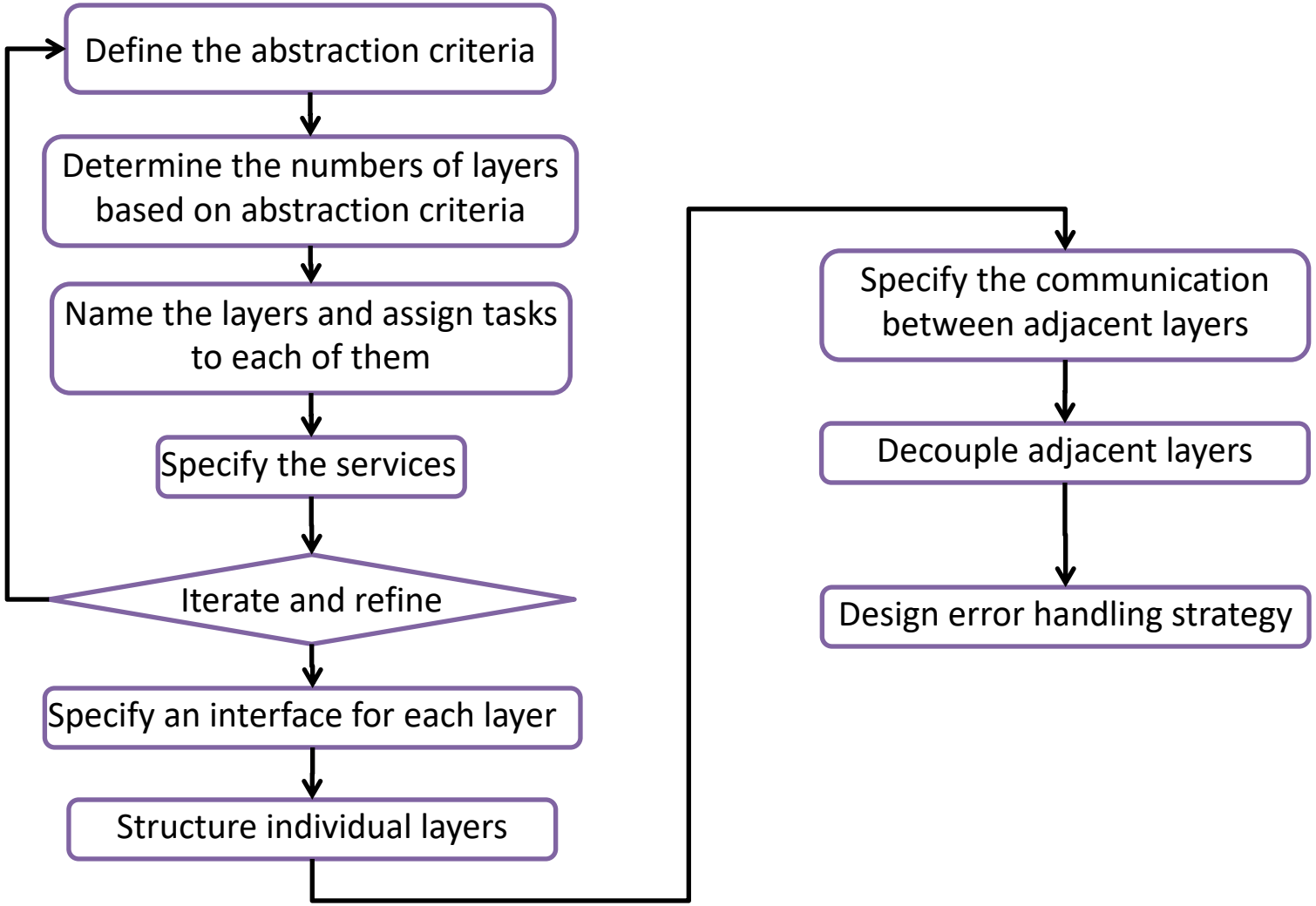
Scenario – IV (In network protocol design, intermediate layer responds based on a notification

Dynamics

- Scenario V
  - Involves two stacks communication with each other



Implementation Guideline



Define abstraction criteria

Level of abstractions define the layers. Heuristics can be

- Most generic components are in lowest layer whereas the domain-specific components are in top layer

- More stable components (which hardly undergoes change) are in lower layer. Use degree of stability to decide layers

- Distance from hardware
  - User-visible elements
  - Specific Application Modules
  - Common Service Levels
  - OS Interface Level
  - Hardware

Determine the no. of abstraction levels

- Typically each abstraction level is one layer
- Map the abstraction levels to layers
- Use mechanisms to keep number of layers to optimum number (say 3 layers for a typical self-service based application)
  - Too Few Layers → Can Result in Poor Structure
  - Too Many Layers → Impose Unnecessary Overhead

Complete Layer specification

A) Name the layer and assign tasks
   - Highest layers are system functionality perceived by the user
   - Lower layers are helpers
   - In bottom up approach – create generic tasks at the lowest level- sort of infrastructure
   - Requires experience to achieve this

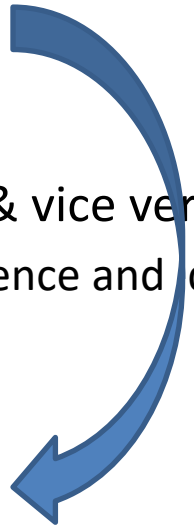
B) Specify the services
   - Strict separation of layers
   - No component should spread over two layers
   - Inverted pyramid of use

Construct Each Layer

- Specify layer interface
  - Use a black box approach
  - Layer N treats Layer N-1 as a black box
- Structure each layer
  - Identify components inside each layer
  - Bridge or strategy pattern can help
    - Supports multiple implementations of services provided by a layer
    - Supports Dynamic exchange of algorithms used by a user

Inter layer communication

- Identify communication mechanism
  - Push: upper layer invokes a service of lower one
  - Pull mechanism
- Layer decoupling
  - Lower layer not aware of higher layer & vice versa
    - Changes in Layer J can ignore the presence and identity of Layer J+1 [Suitable for Top-up communication]
  - What happens in Bottom up scenario?
    - Use of call backs
    - Upper layer registers with lower layer
    - Lower layer maintains mapping between event and callback functions
  - (reactor and commnd pattern)



Design an error handling strategy

- Define an efficient strategy
- Handling may be expensive – errors need to propagate

Benefits

Benefits

- Reuse of layers
- Support for standardization
- Dependencies are kept local
- Exchangeability

Liabilities

- Cascades of changing behavior
- Lower efficiency
- Unnecessary work
- Difficulty in establishing the correct granularity

Examples

Your application

Middleware- J2EE

- Can replace vendor (oracle, IBM,JBOSS)

JVM

- Can replace the vendor

OS

- Switch from Windows to Unix

Virtual Machines

Presentation

Application logic

- Application logic, business rules

Domain layer

- Conceptual model of domain elements

Database

- Tables, indexes

Information Systems

System services

Resource Mgmt

- Secuity monitor, process mgr, I/O mgr, virtual memory mgr

Kernel

- Interrupt, exception, thread scheduling & dispatching

Hardware abstraction

- Hides h/w differences between different processor families

Operating system- Windows NT

Can you find (at least 2) more popular uses and document them?

Layers

Pattern	Description
Context	A large system that requires decomposition
Problem	Mix of low- and high-level issues, where high-level operations rely on low-level ones A typical pattern of communication flow consists of requests moving from high level to low level, and answers to requests, incoming data and notification about events traveling in the opposite direction Forces •Code changes should not ripple through the system •Stable interfaces; standardization •Exchangeable parts •Grouping of responsibilities for better understandability and maintainability
Solution	Structure the system into appropriate number of layers
Variants	Relaxed Layered System Layering Through Inheritance
Benefits	Reuse of layers Support for standardization Dependencies are kept local Exchangeability
Liabilities	Cascades of changing behavior Lower efficiency Unnecessary work Difficulty in establishing the correct granularity

Thank You