Architectural Styles

Truong Ho-Quang

https://en.wikipedia.org/wiki/De_Stijl

Schedule

We are HERE!

Week		Date	Time	Lecture	
3	L1	Wed, 20 Jan	10:15 – 12:00	Introduction & Organization	o
3	L2	Thu, 21 Jan	13:15 – 15:00	Architecting Process & Views	o
4		Tue, 26 Jan	10:15 – 12:00	**Skip**	
4	S1	Wed, 27 Jan	10:15 – 12:00	<< Supervision: Launch Assig ent 1>>	TAs
4	L3	Thu, 28 Jan	13:15 - 15:00	Roles/Responsibilities & Functional D composition	Truong Ho
5	L4	Mon, 1 Feb	13:15 – 15:00	Architectural Styles P1	Truong Ho
5	S2	Wed, 3 Jan	10:15 – 12:00	<< Supervision/Assignment>>	TAs
5	L5	Thu, 4 Jan	13:15 – 15:00	Architectural Styles P2	Truong Ho
6	L6	Mon, 8 Feb	13:15 – 15:00	Architectural Styles P3	Sam Jobara
6	S3	Wed, 10 Feb	10:15 – 12:00	<< Supervision/Assignment>>	TAs
6	L7	Thu, 11 Feb	13:15 – 15:00	Design Principles (Maintainability, Modifiability)	Truong Ho
7	L8	Mon, 15 Feb	13:15 – 15:00	Performance – Analysis & Tactics	Truong Ho
7	S4	Wed, 17 Feb	10:15 – 12:00	<< Supervision/Assignment>>	TAs
7	L9	Thu, 18 Feb	13:15 – 15:00	Tactics: Reliability, Availability, Fault Tolerance	TBD
8	L10	Mon, 22 Feb	13:15 – 15:00	Guest Lecture 1	TBD
8	S5	Wed, 24 Feb	10:15 – 12:00	<< Supervision/Assignment>>	TAs
8	L11	Thu, 25 Feb	13:15 – 15:00	Guest Lecture 2	TBD
9	L12	Mon, 1 Mar	13:15 – 15:00	Reverse Engineering & Correspondence	Truong Ho
9	S6	Wed, 3 Mar	10:15 – 12:00	<< Supervision/Assignment>>	TAs
9	L13	Thu, 4 Mar	13:15 – 15:00	To be determined (exam practice?)	Truong Ho
9		Fri, 5 Mar	Whole day	Group presentation of Assignment (TBD)	Teachers
11	Exam				

Teams and Supervisors

- Have you started?
- Any missing team members? – Inform me asap (truongh@chalmers.se)
- Contact with your supervisor to set up 2nd Supervision Session

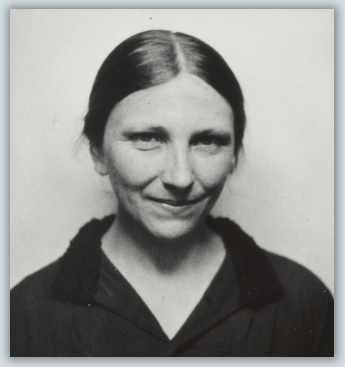
Supervisor	Groups
Mazen	Group 1, Team Cloud, Group 3, Group 4
Truong (*)	Group 5, Mind Optimizers, Group 7, Group 8

(*) Only a temporary supervisor, will be replaced by another supervisor soon.

Architectural Styles

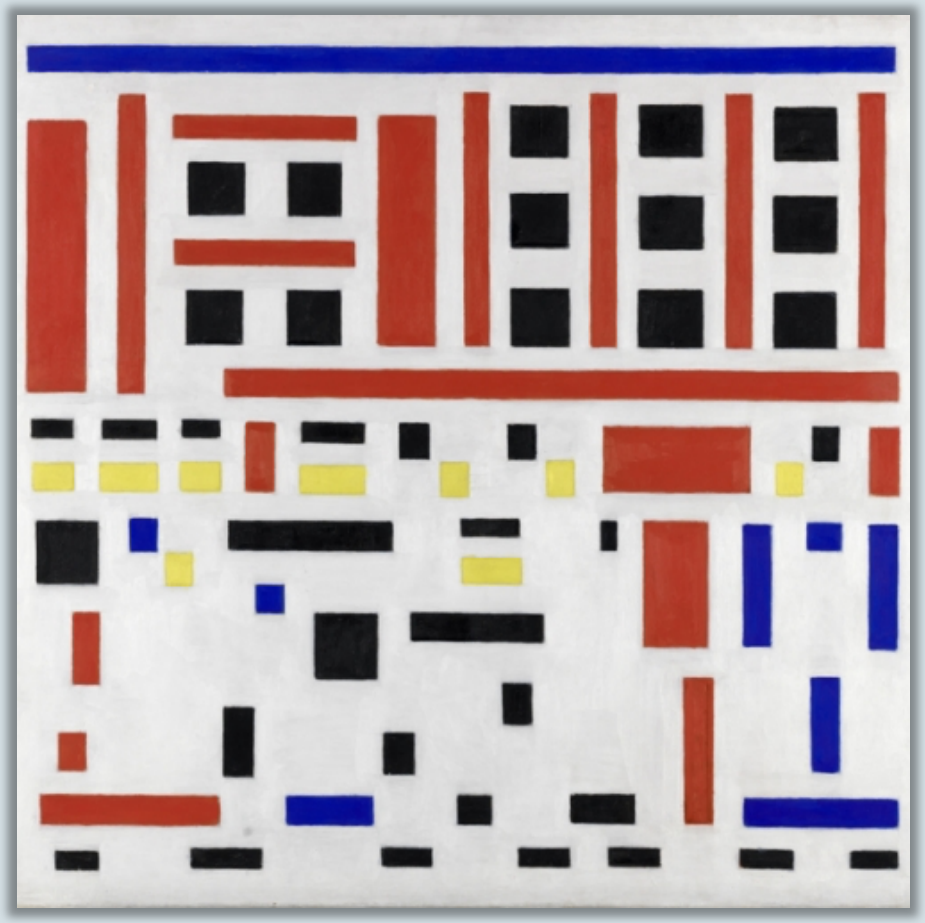
Truong Ho-Quang

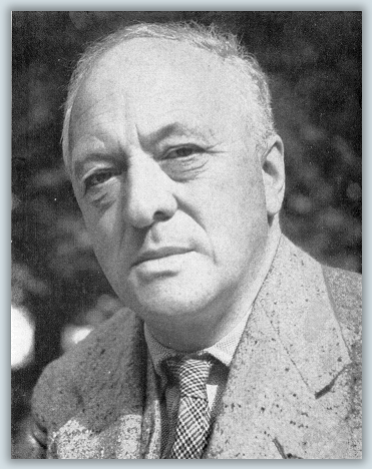
Piet Mondriaan



Truus Schröder-Schräder

Gerrit Rietveld





Bart van der Leck

(1876 – 1958)

[Composition no. 4, Uitgaan van de fabriek, 1917]

5

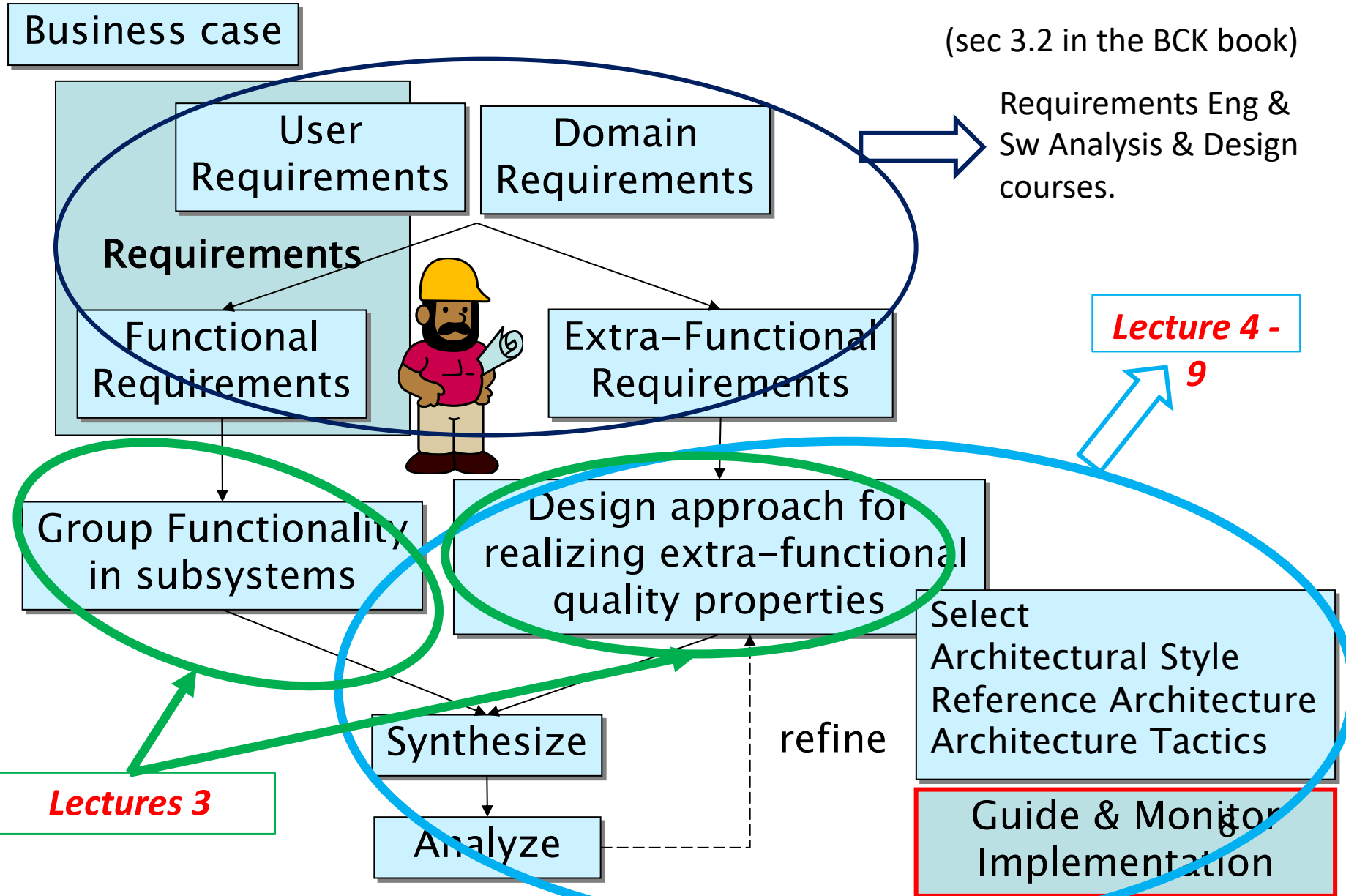
Which ‘Rules’ Define this style?

Recap

- *Forces* influence the design of the architecture
- *Architectural drivers*/*quality attributes* are the forces that dominate the architecture design decisions
- Notion of *roles, responsibilities, and collaboration*.

Where we are in the SW Arch Design Process?

(sec 3.2 in the BCK book)



Learning Objectives of this lecture

The task of the architect is to come up with a good metaphor for the system
Alexander Ran (Nokia)

- Conceptual Integrity
- Build vocabulary of architectural styles
  - a set of 'archetypes' that are often used
  - know their relative strengths and weaknesses
- Client-Server style
- Pipe and Filter style
- Know when to apply or *not* to apply a particular style

Conceptual Integrity

Anywhere you look in your system, you can tell that the design is part of the same overall design style, theme, mood …is about **Design and Style Consistency in all dimensions of the system**

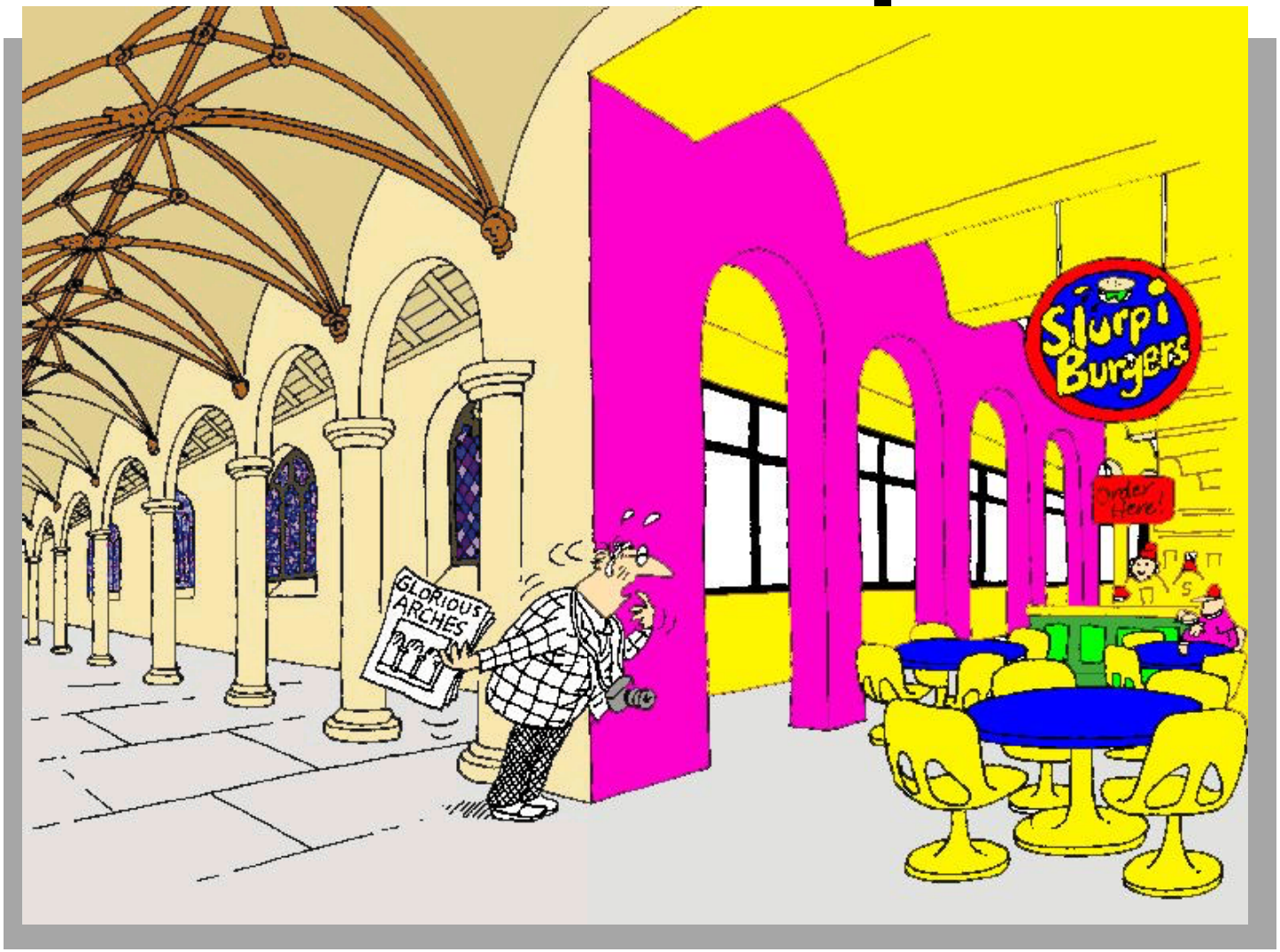
User interface, technologies, coding styles, naming conventions, directory structures, classes, components, interfaces, internal and external behavior, deployment…

Fed Brooks:
"It is better to have a system…reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas"

The Mythical Man-Month

Conceptual Integrity tries to limit the system complexity
Conceptual Integrity simplifies collaboration when creating software

Conceptual integrity counter example!

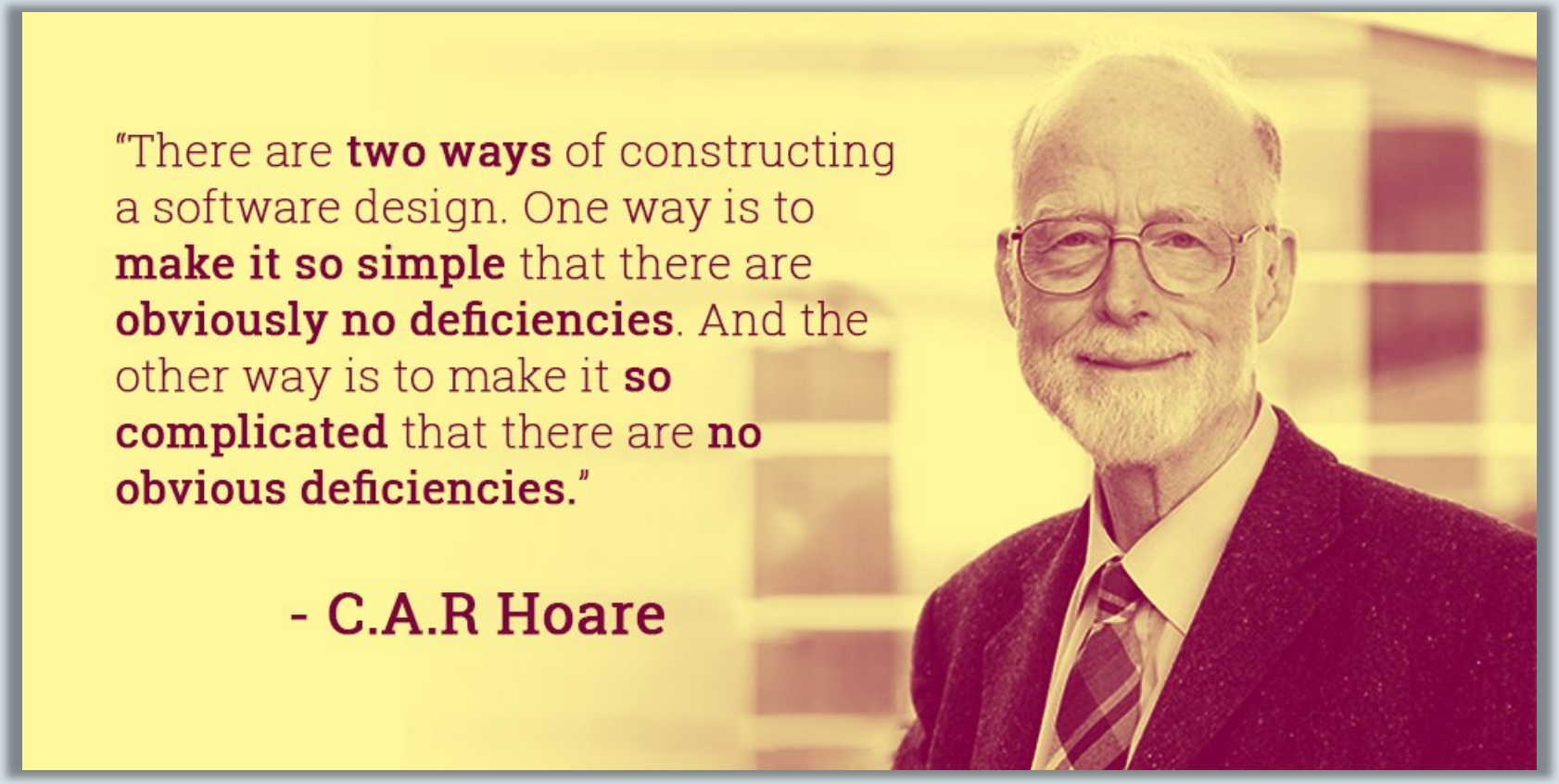


Conceptual Integrity in Software

- Uniformity – where possible
  - Same solution-approach/tactic/design-principle applied to similar problems
  - For example in
    - patterns of communication & control/data-flow
    - Naming of methods, parameters, variables
    - Structure of API's
    - Layering
    - Exception-handling
    - User interface (dialogs)

Harmony

Uniformity


"There are **two ways** of constructing a software design. One way is to **make it so simple** that there are **obviously no deficiencies**. And the other way is to make it **so complicated** that there are **no obvious deficiencies.**"
- C.A.R Hoare

CONTENTS

CONTENTS

What is an Architectural Style?

- Architectural styles are **collections of design conventions** for a set of design dimensions

  Some architectural styles emphasize different aspects such as:

  - Subdivision of **functionality**,
  - **topology**
  - **interaction/coordination** pattern between components

- An architecture can **use several** architectural styles
- Architectural styles are **not disjoint**
- Styles are **open-ended**; new styles may emerge

Architectural style

An *architectural style* is defined by:

a set of rules and constraints that prescribe

- **vocabulary/metaphor**: which types of components, interfaces & connectors must/may be used in a system.
  Possibly introducing domain-specific types
- **structure**: how components and connectors may be combined
- **behaviour**: how the system behaves
- **guidelines**: these support the application of the style (how to achieve certain system properties)

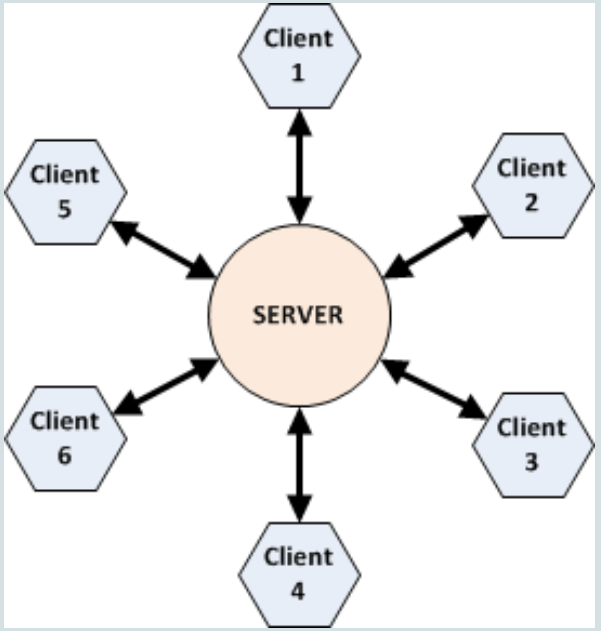
19

What is a Style?

Coordinated, aligned

CONTENTS

Client-Server Architectures



Nice source:

IT Architectures and Middleware:
Strategies for building Large Integrated Systems,
Chris Britton and Peter Bye, Addison Wesley, 2004

What is Client / Server?

Client: an application that makes requests (to the servers) and handles input/output with the system environment

Server: an application that services requests from clients

Client/Server System:
an application that is built from clients and servers

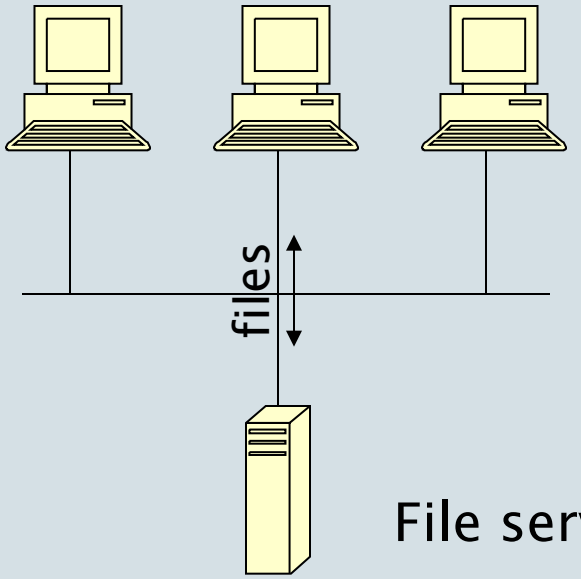
Typical application area:
distributed multi-user (business) information systems

In real-life: client *instructs/commands* the server.
Pitfall: keep an eye on (hidden) dependencies

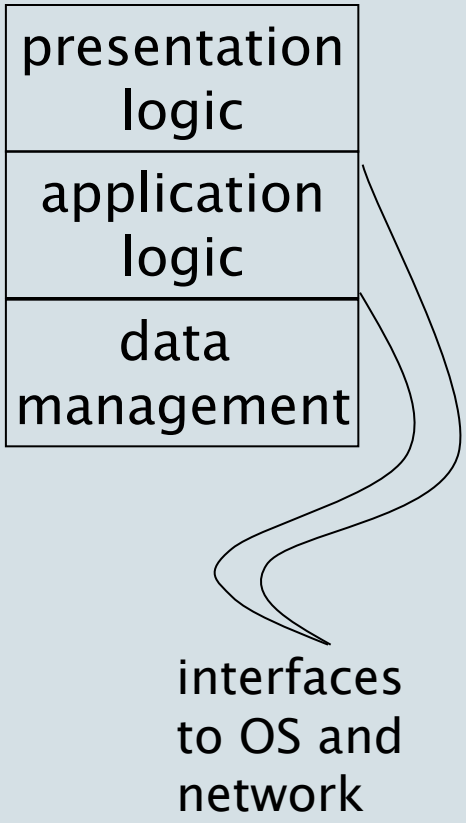
Why Client / Server?

±1980's

- multiple users want to share and exchange data
- first: attempt: shared file-server
- problem:
  scalability to ±10 due to contention for files and volume of data-transfer
- solution:
  perform processing on (file) server



3-tier Reference Model



presentation logic ([G]UI):
anything that involves system/user interaction
e.g. dialogs (management), forms, reports

application logic (data processing):
where the functionality of the application resides / where the actual computation of the system takes place

data management:
storing, retrieving and updating data

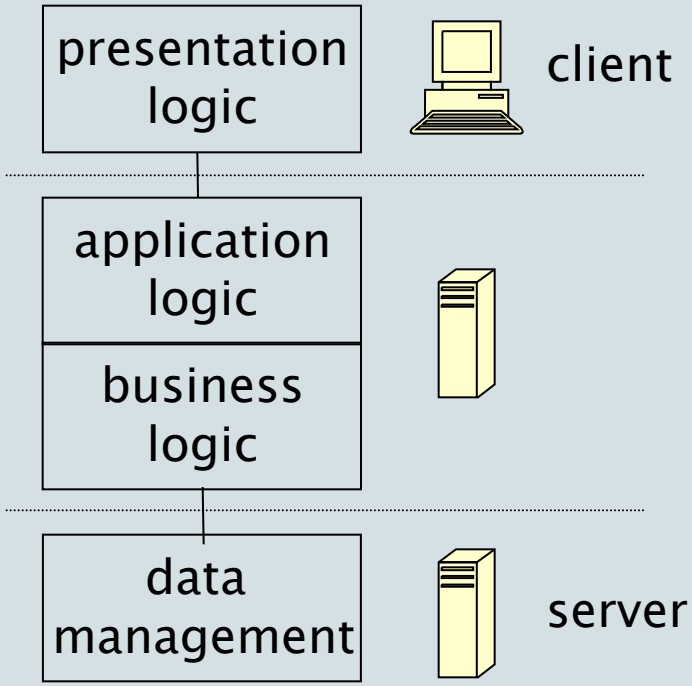
Client / Server Style

Concept: Separation of application in units of change

Components: presentation,
application logic,
business logic,
data management

Connector: 'uses' lower layer

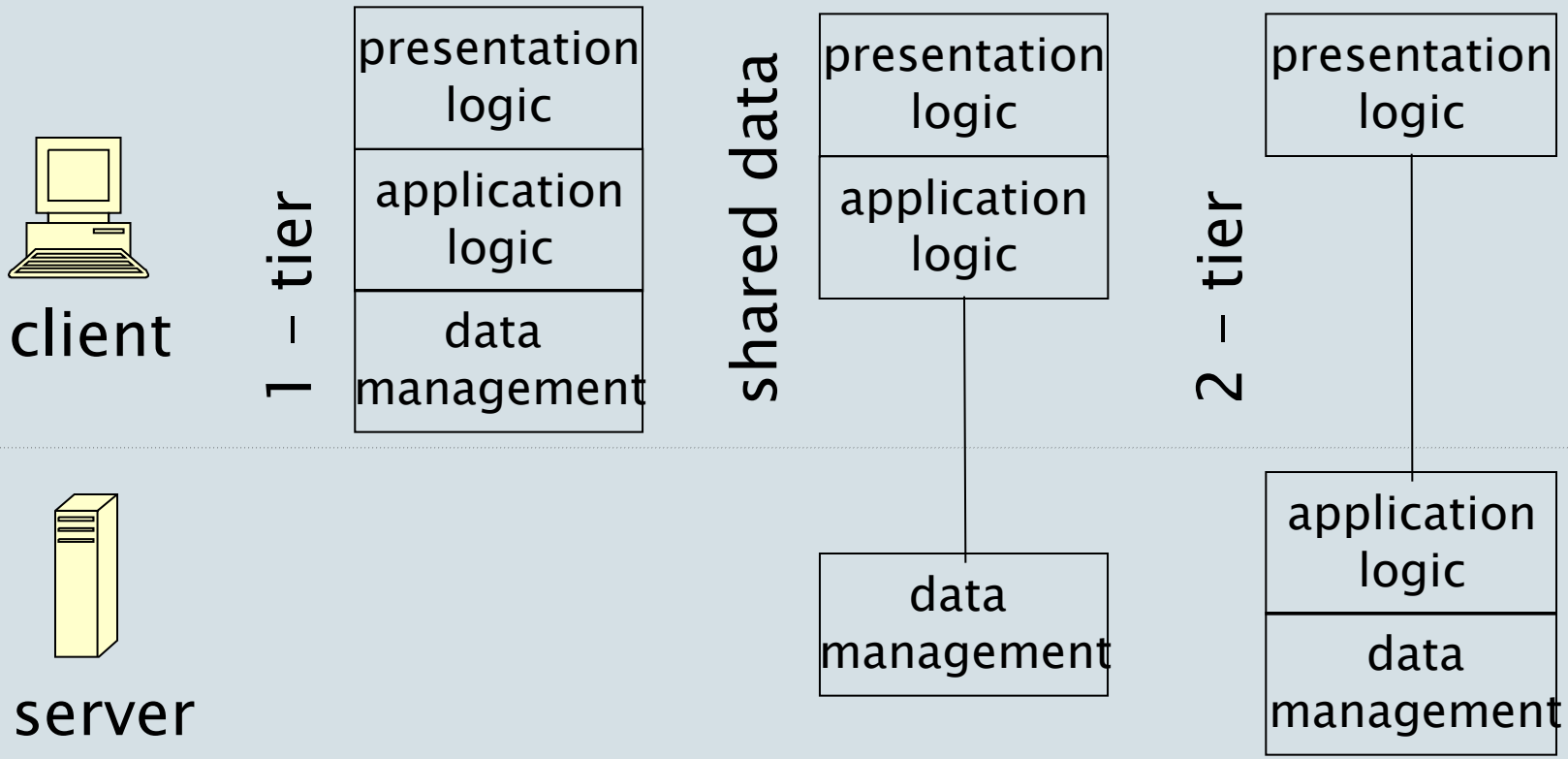
Interaction style: request/response



Deployment of C/S Model

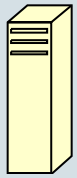
Rather than having the client do the processing …

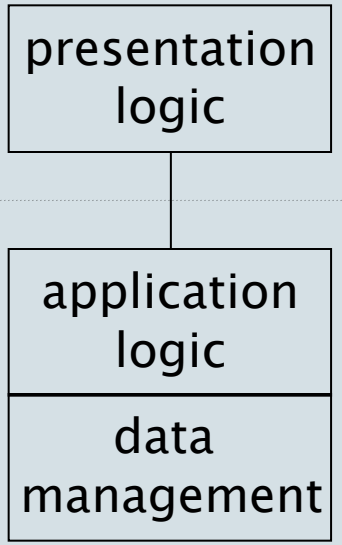
Move processing power to server such that the server sends a (condensed) response to request rather than a whole file



C/S Example: Thin Client

Thin Client C/S:
largest part of processing at the server-side





WWW Browser

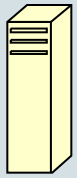
functional processing

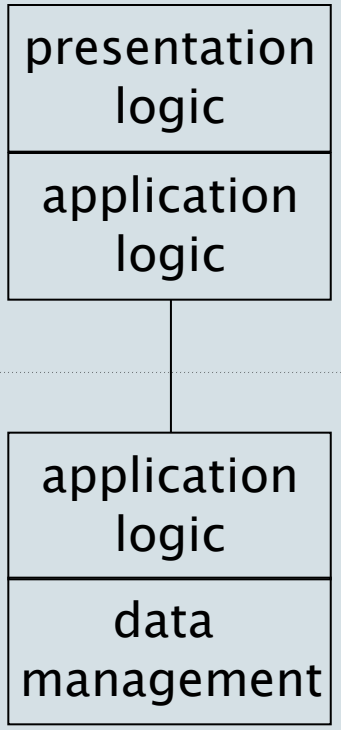
database storage and access

Network load: low
Config. Mngmnt: simple (only server)
Security: concentrated at server
Robustness: stateless clients => easy fault recovery

C/S Example: Thick Client

Thick Client:
significant processing
at the client-side





WWW Browser

application (specific)

application (generic)

database

Network load: high
Config. Mngmnt: complex (both client & server)
Security: complex (both client & server)
Robustness: clients have state => complex fault recovery

Client/Server in terms of Characteristics

Dependencies: client depends on the server

Topology: one or more clients may be connected to a server there are no connections between clients

Multiplicity: 1-to-1, directed

Synchronicity: synchronous or asynchronous

Mobility: easily supports client mobiliy

Binding: from compile to invocation time

Initiative: request (by client) / response (by server)

Periodicity: typically a-periodic (periodic possible)

Security: typically controlled at server, also possible at application/business layer

Example 3–tier System

Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier (application)

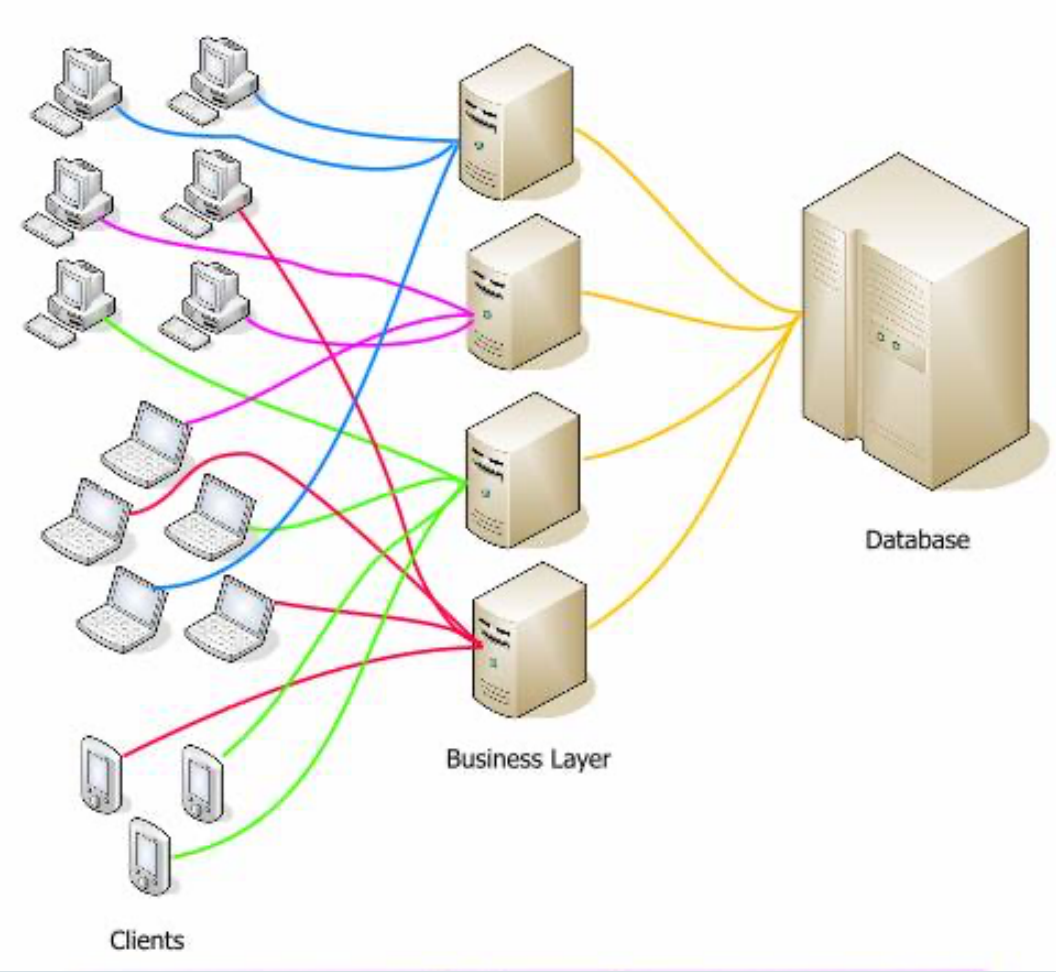
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

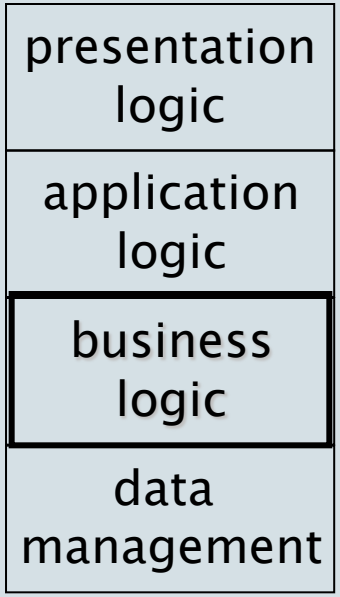
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Diagram from Wikipedia, 2007

Deployment: Many physical clients and servers



Business Logic Layer

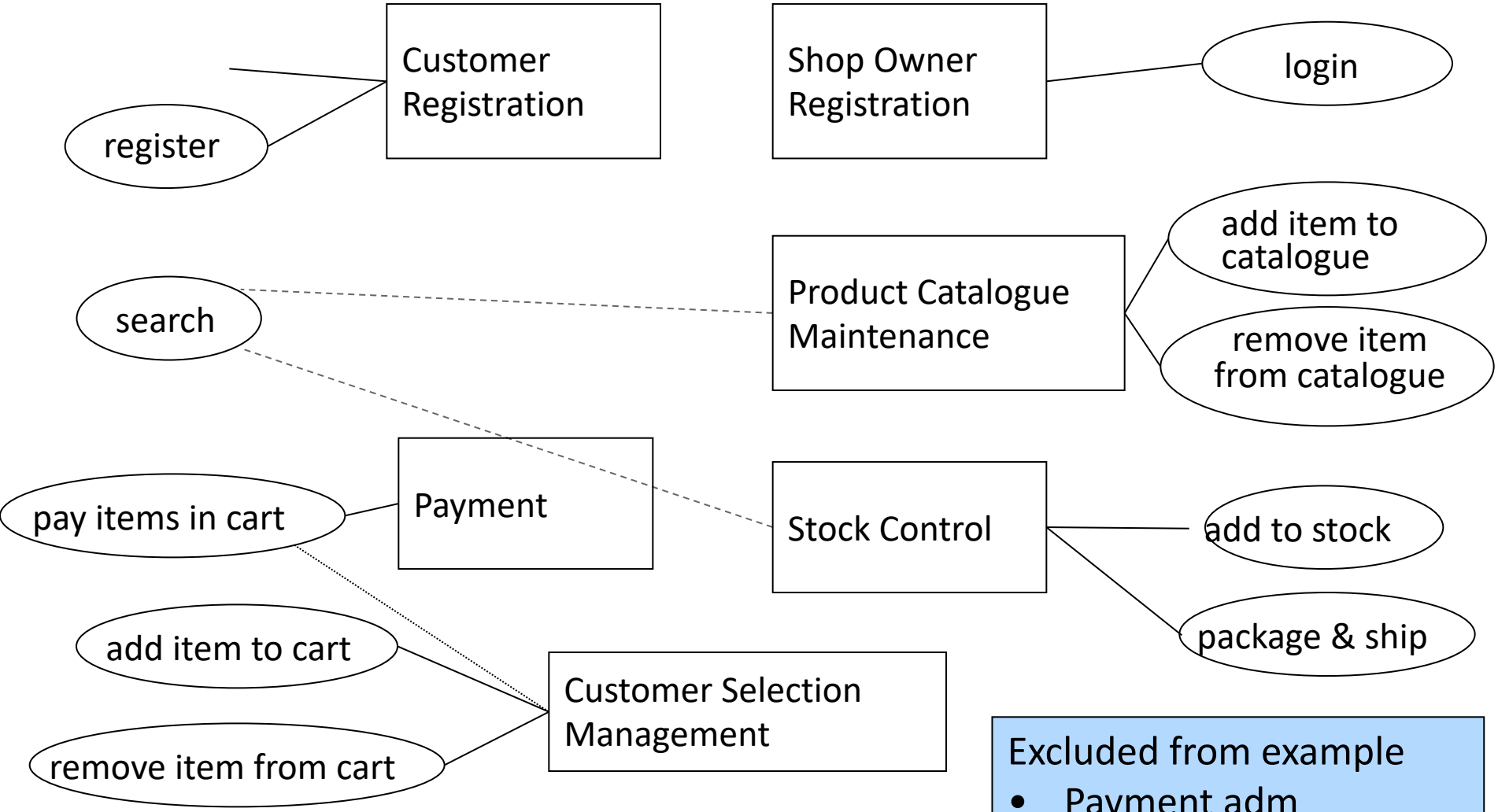


An even broader organizational scope of sharing and exchanging data requires
coordination across *multiple applications* and *databases*

The complexity of the middle tier ranges from
reactive with little intelligence
(e.g. resource management and interconnection services)
to active with much intelligence
(active enforcement of global constraints and coordination of activities across applications e.g. workflow)

Advantage of business logic-tier:
changes to business are localized
(compared to intertwining with application logic)

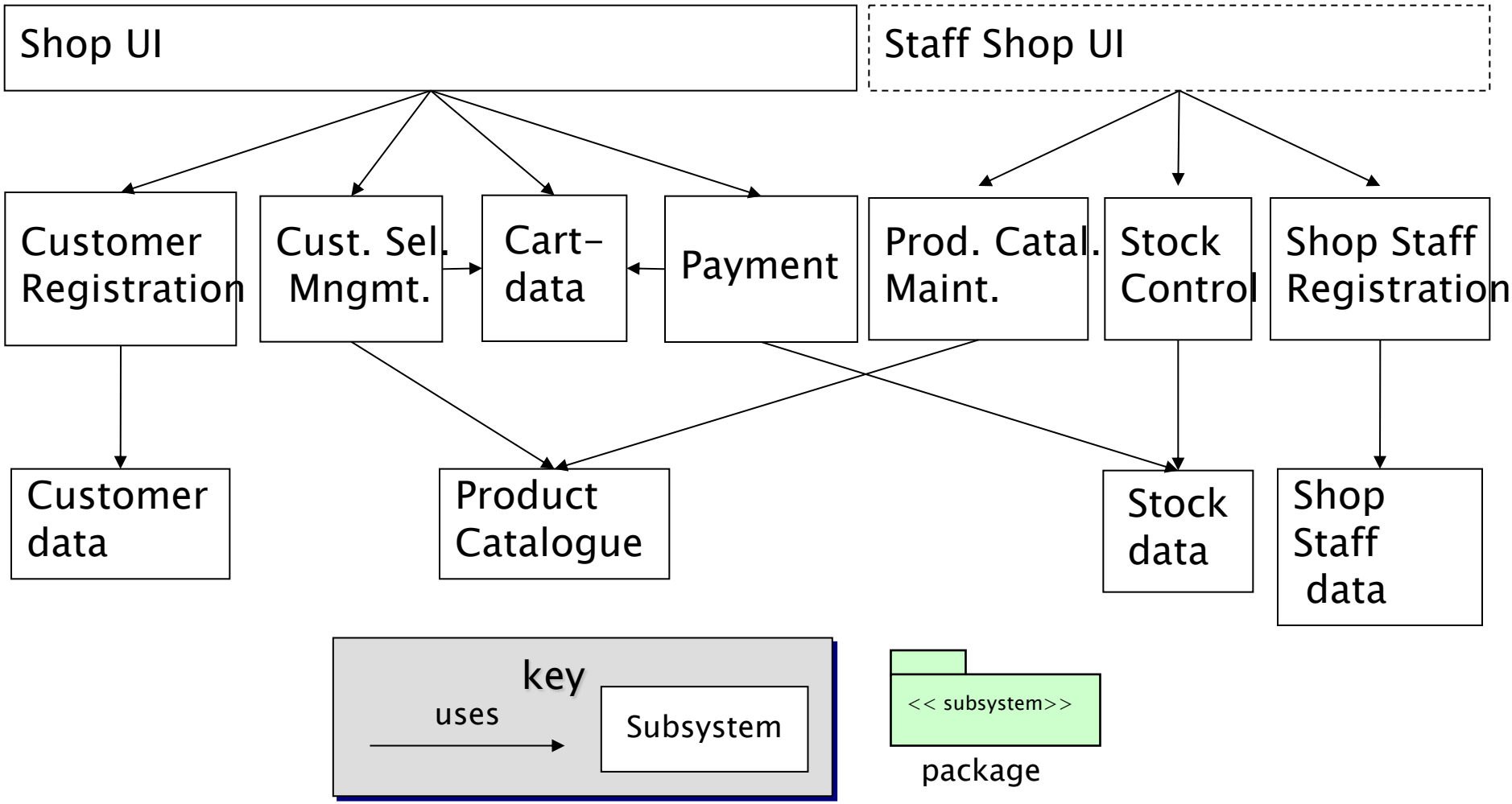
C/S Example: Web Shop



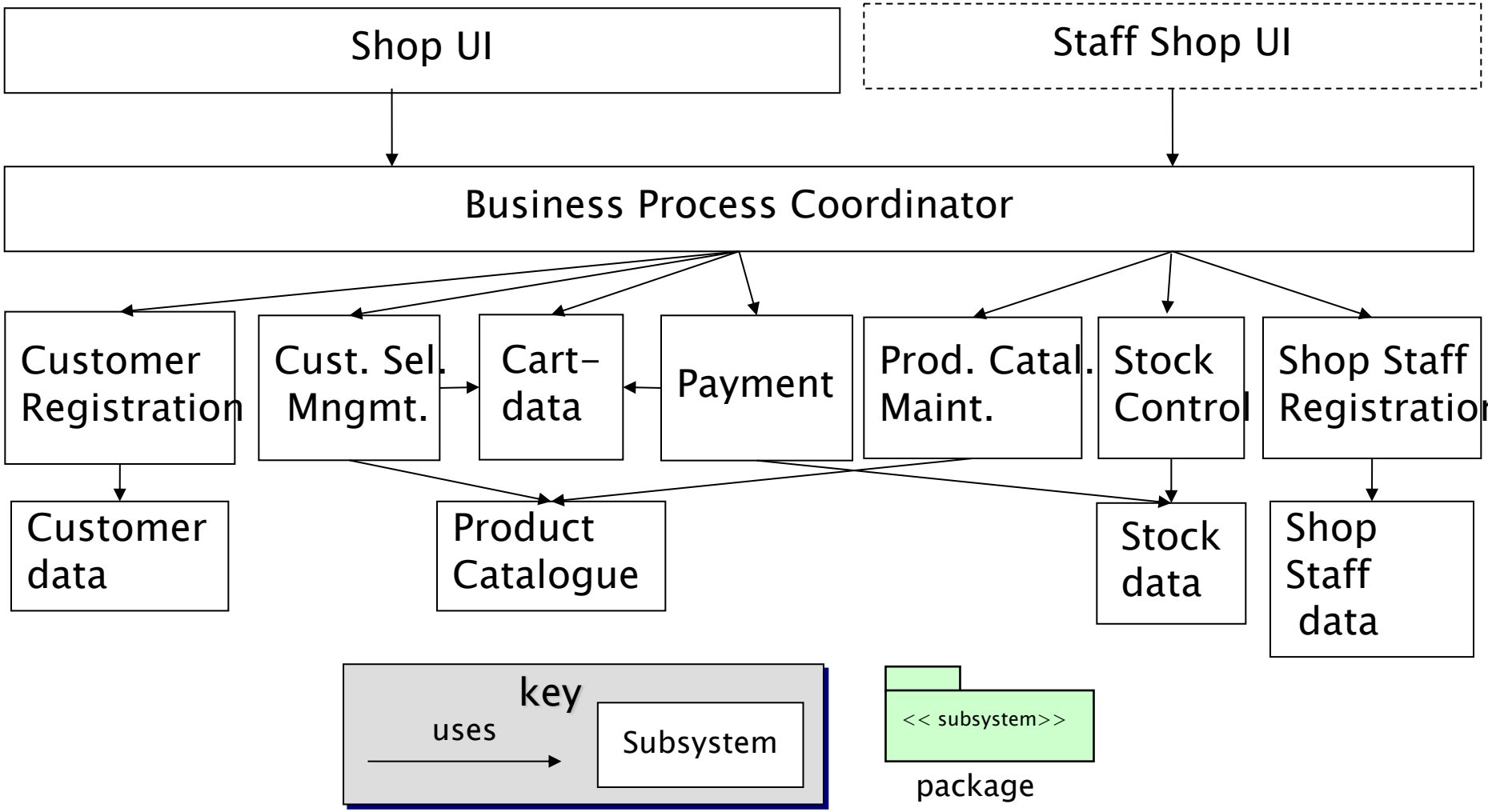
Excluded from example

- Payment adm
- Shop staff salary adm
- …

Identification of Dependencies



Introduce layer for business logic



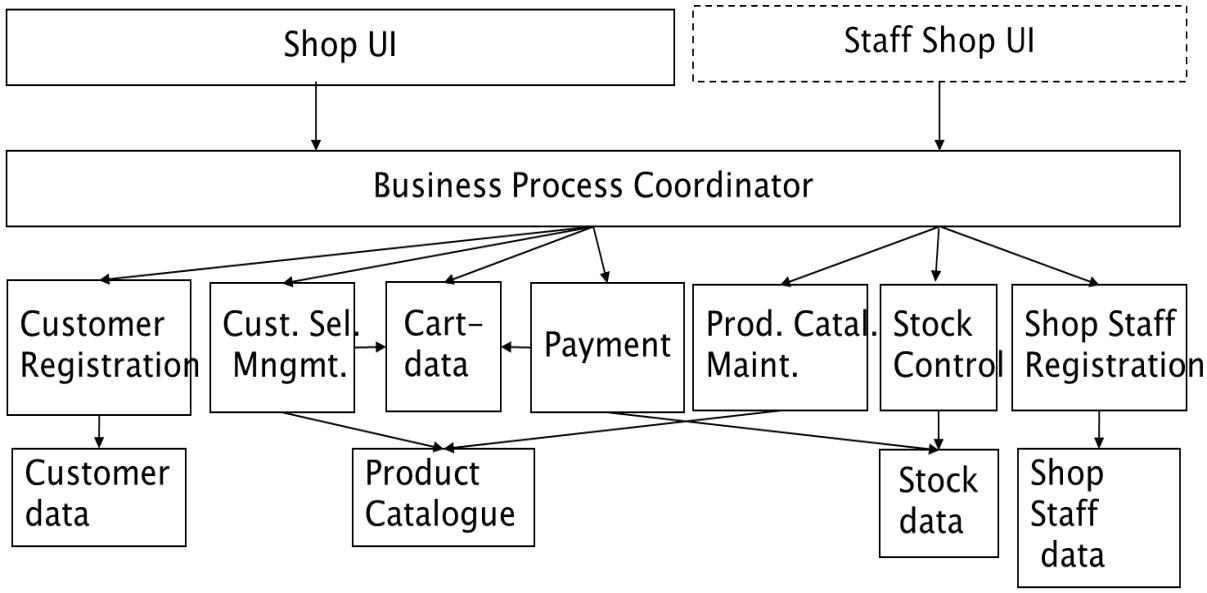
Stereotypes in layered CS-design

Interfacers

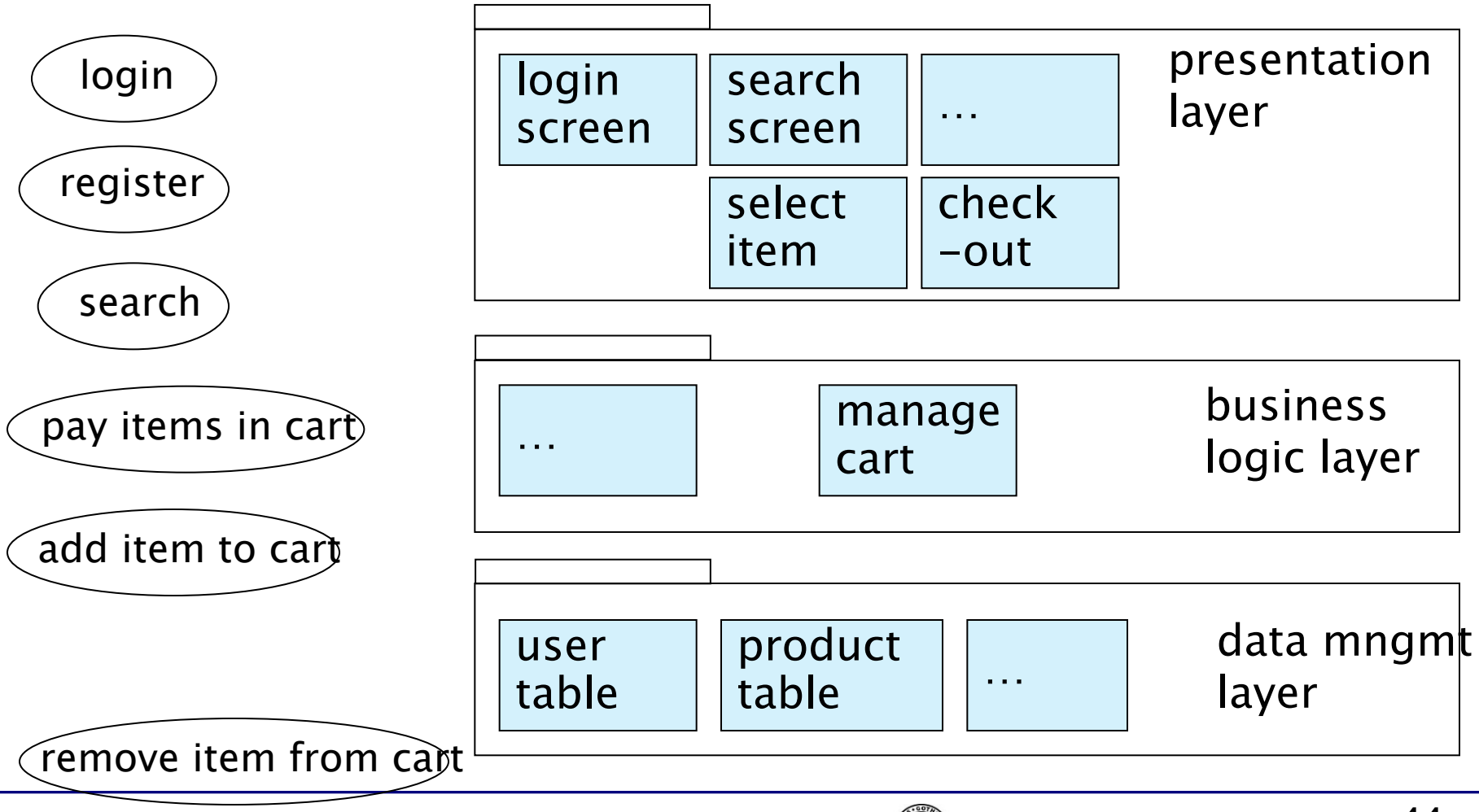
Coordinator

Service Providers

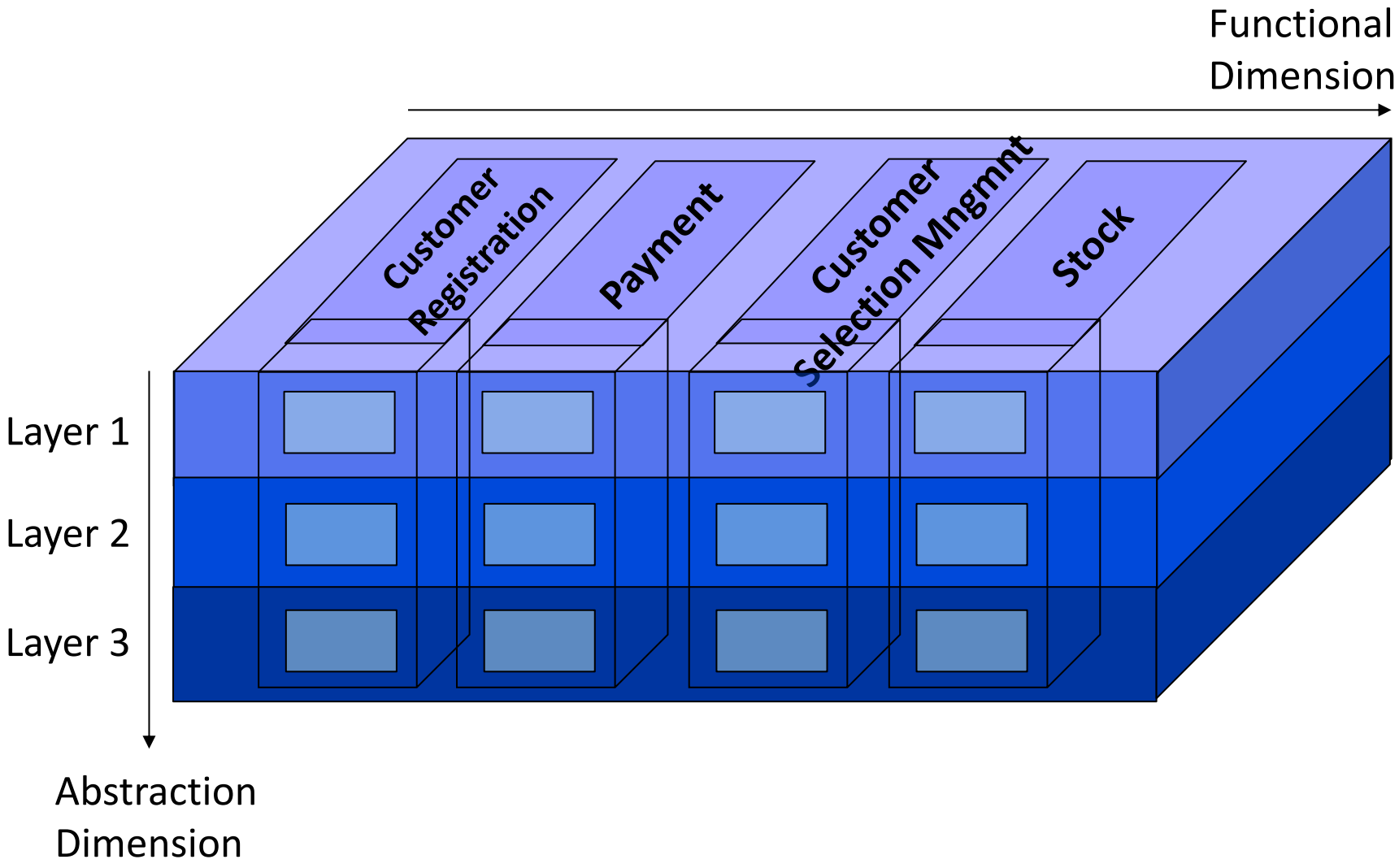
Information holders



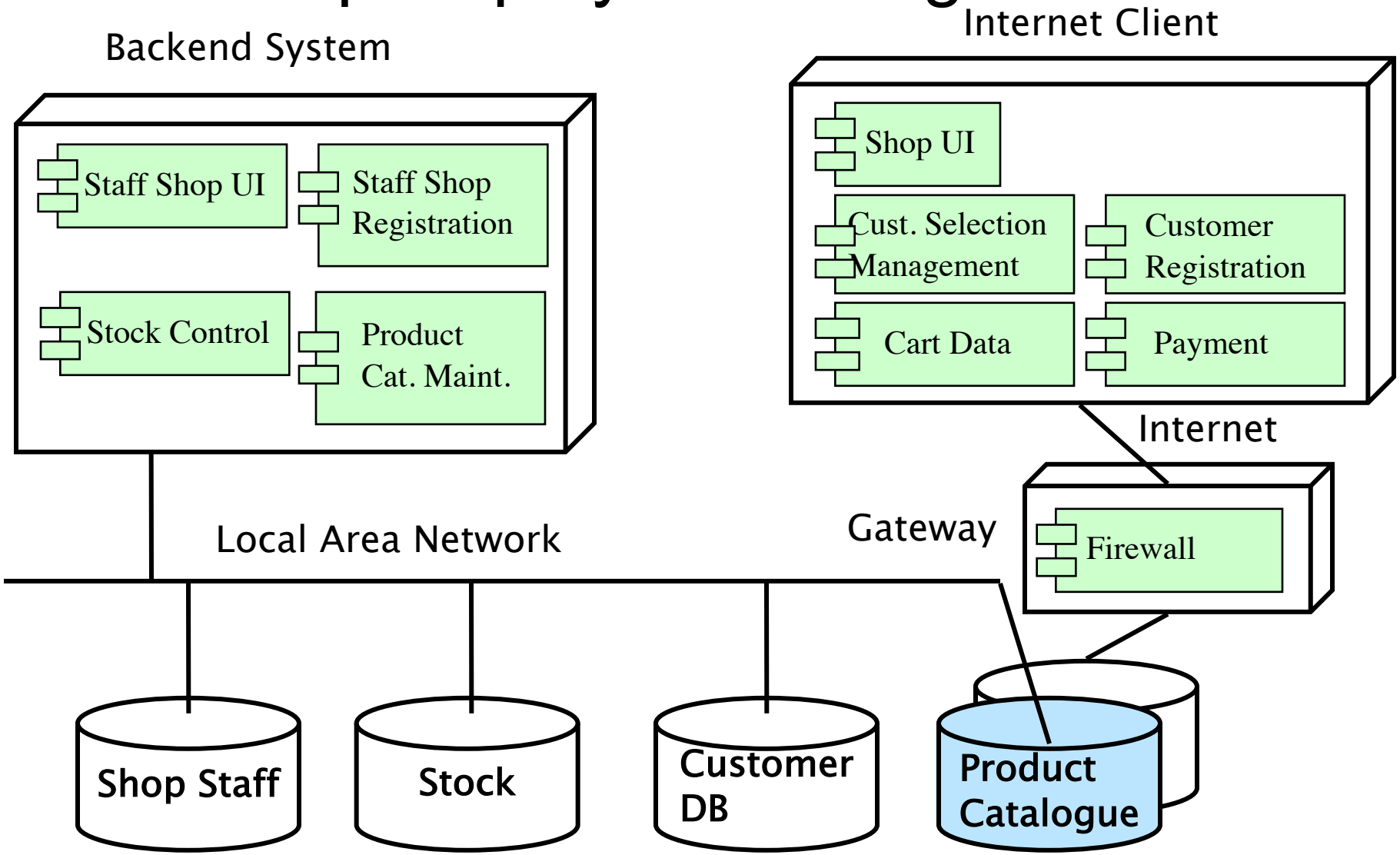
Identify support for Use Cases at different layers in the architecture



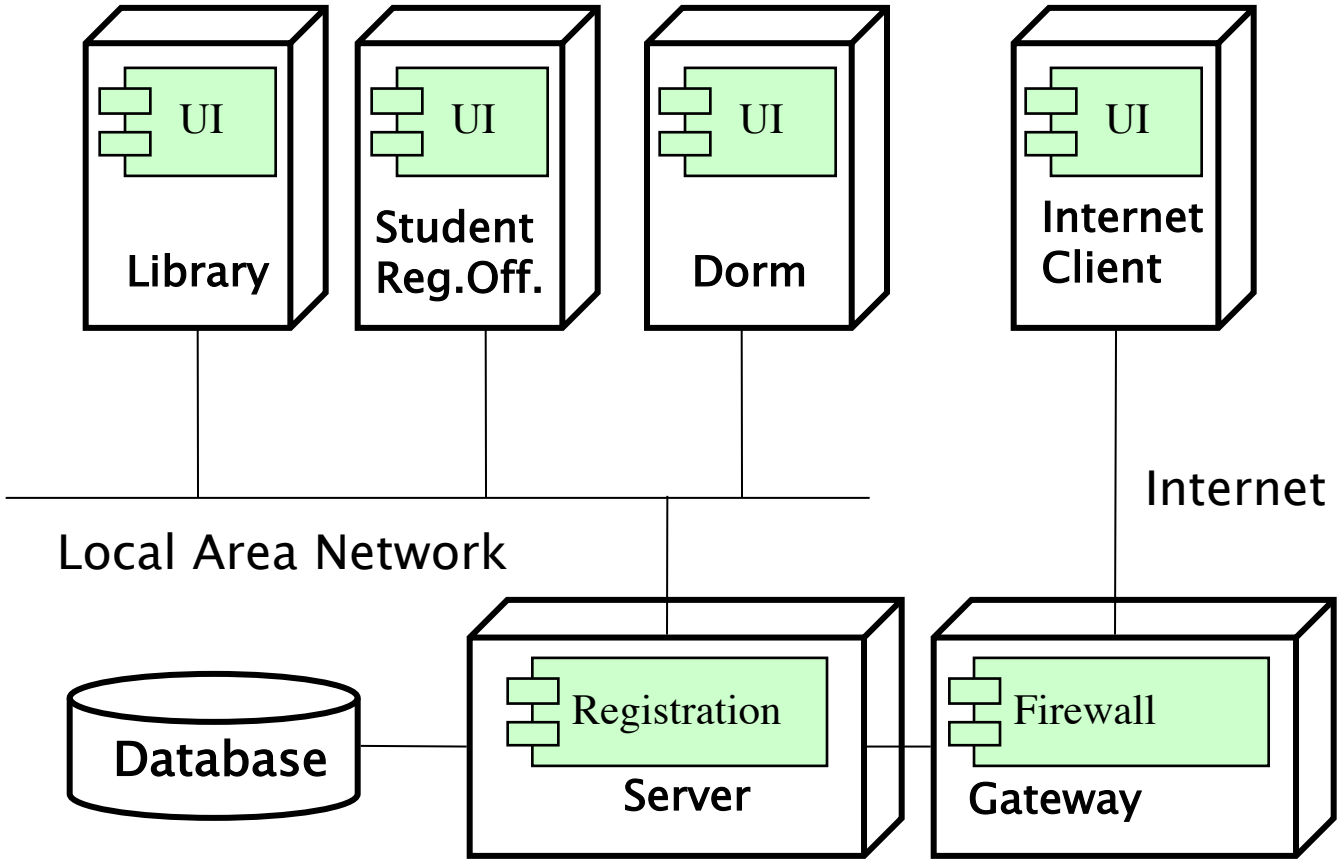
In which dimension(s) does C/S-style apply?



Web Shop Deployment Diagram

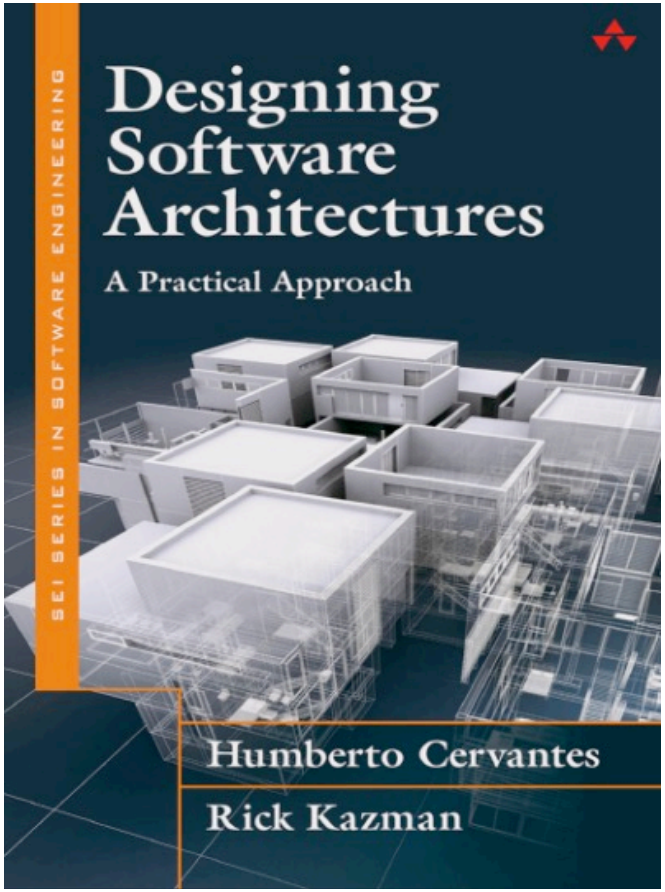


Deployment Example

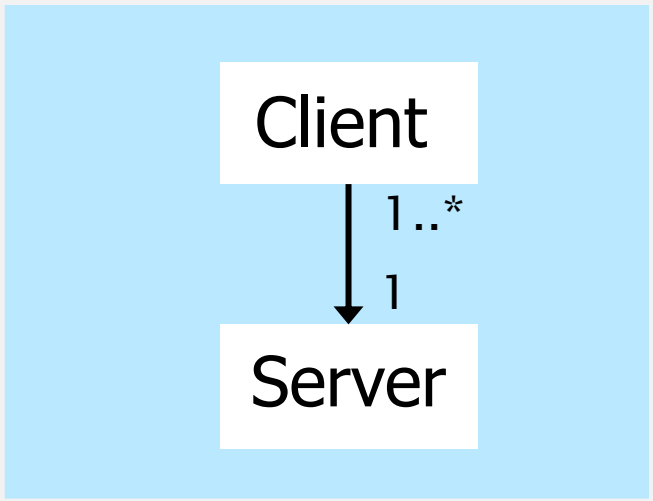


Deployment patterns for Client-Server

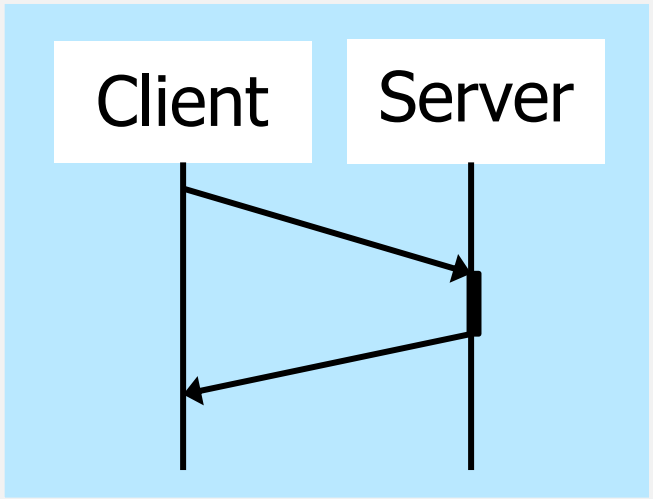
Used diagrams/slides from this book

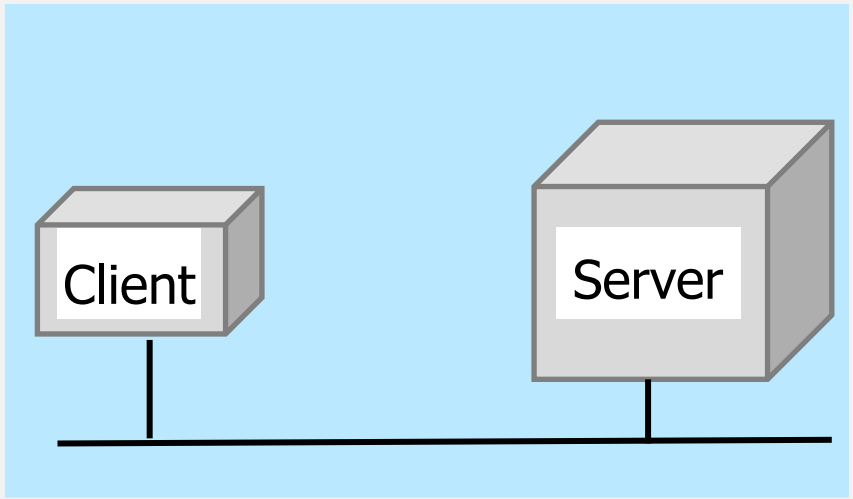


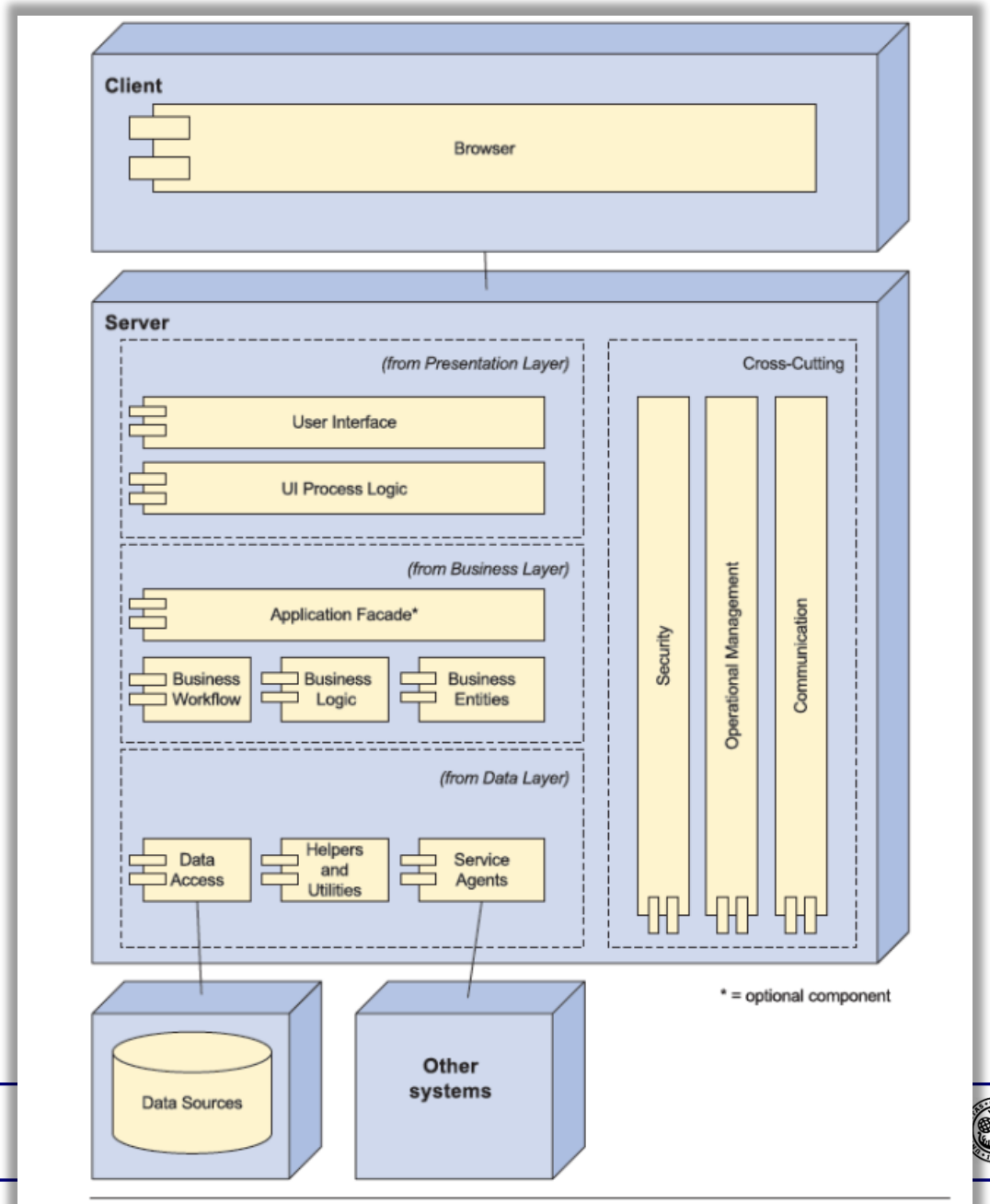
Client-Server (S+B+D)



There is more than one structure to a style!

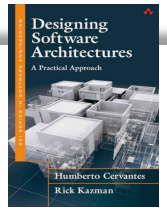





Client
Browser
Server
(from Presentation Layer)
User Interface
UI Process Logic
(from Business Layer)
Application Facade*
Business Workflow
Business Logic
Business Entities
(from Data Layer)
Data Access
Helpers and Utilities
Service Agents
Cross-Cutting
Security
Operational Management
Communication
* = optional component
Data Sources
Other systems


Designing Software Architectures
A Practical Approach
Humberto Cervantes
Rick Kazman

Deployment Patterns



- They provide models on how to physically structure the system to deploy it.

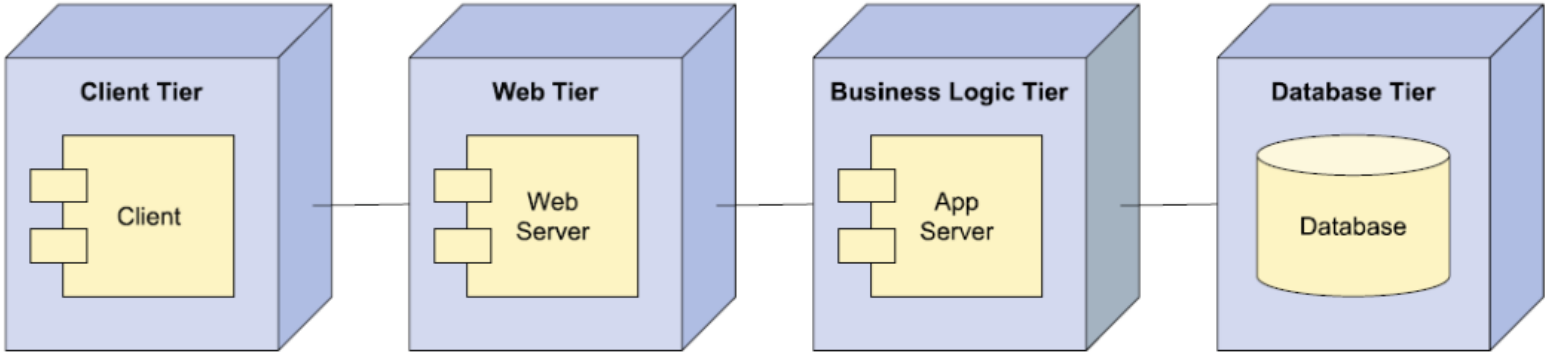


FIGURE 2.6 Four-tier deployment pattern from the *Microsoft Application Architecture Guide* (Key: UML)

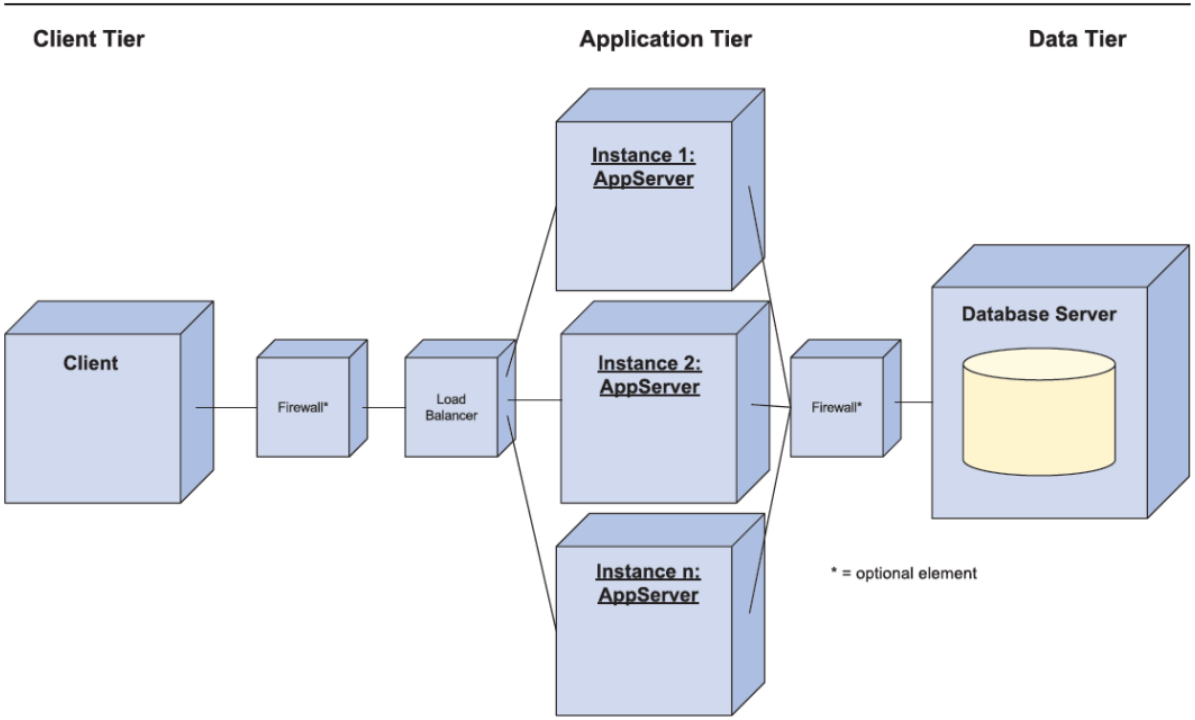


FIGURE 2.7 Load-Balanced Cluster deployment pattern for performance from the *Microsoft Application Architecture Guide* (Key: UML)

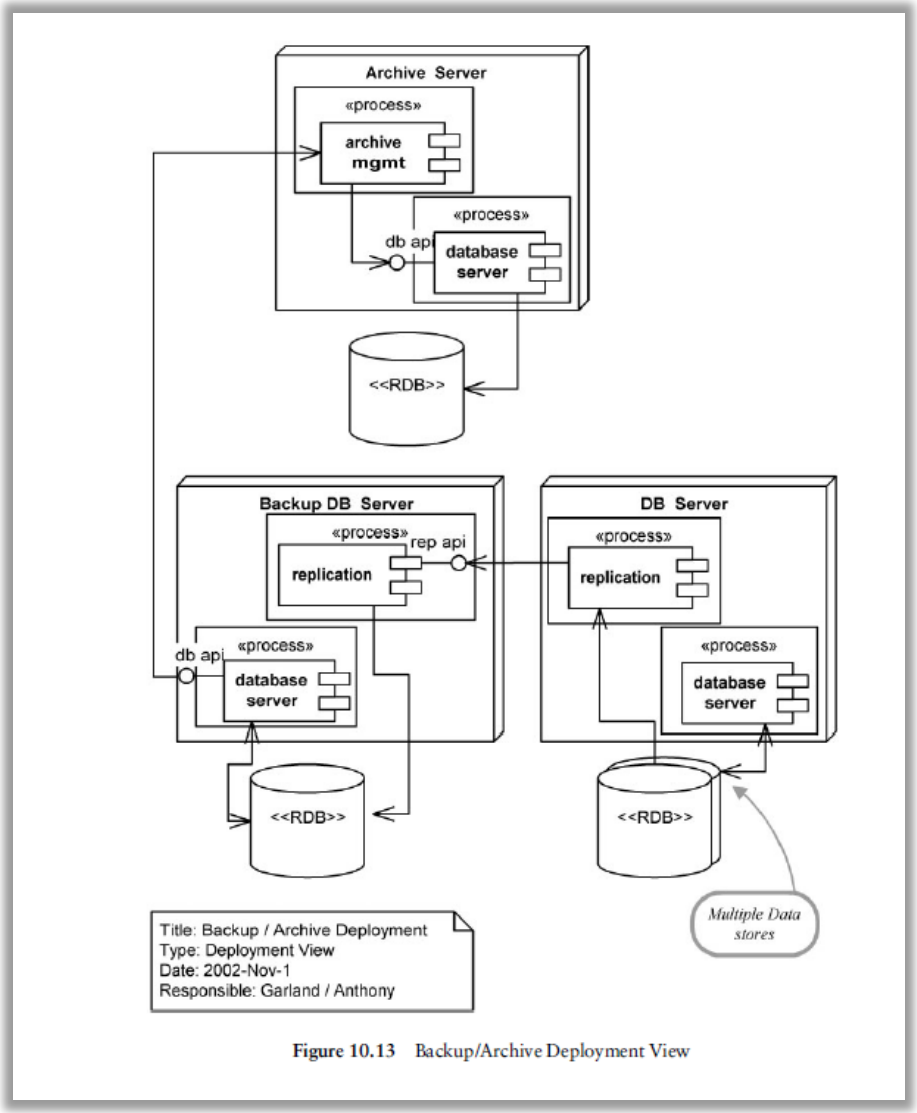
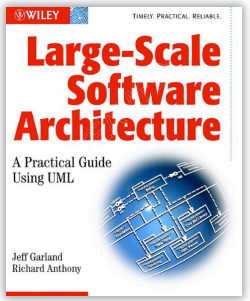


Figure 10.13 Backup/Archive Deployment View



Client / Server Summary

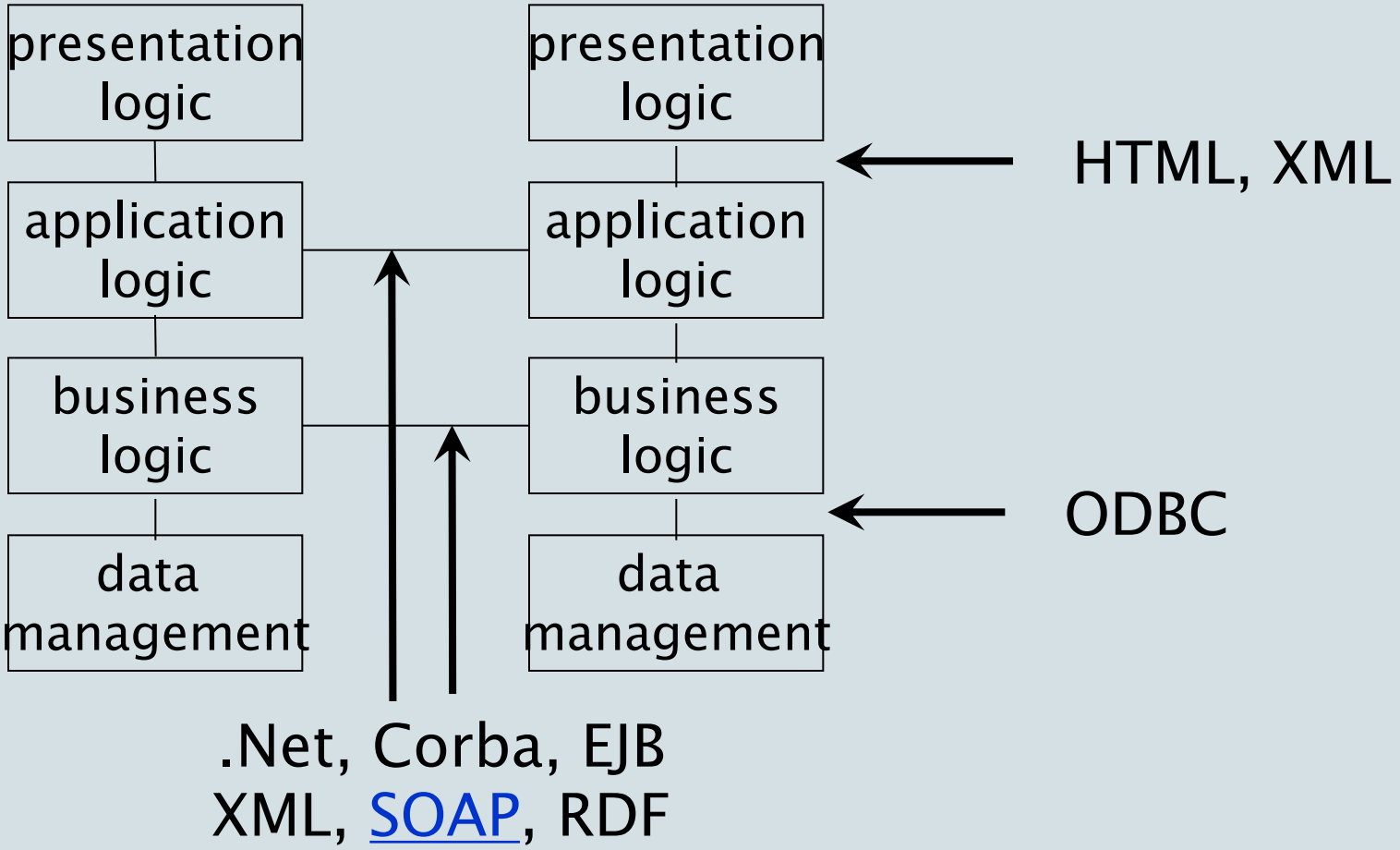
Task are mapped on platform where they are most efficiently handled

- presentation layer on client
- data management and storage on a server
- possible intermediate platforms for transaction multiplexing and global coordination

With the aim of obtaining

- scalability: changing number of clients
- interoperability: client may use data from multiple sources

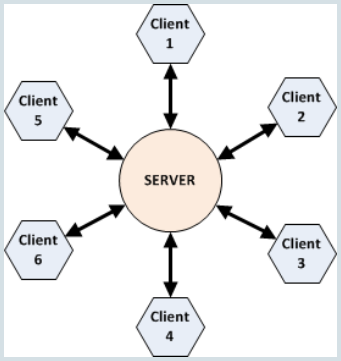
Interface Technologies



n-tier web applications

Layer	Functionality	Role	Technique
Client	User Interface	**Interfacer**	HTML, JavaScript
Presentation	Page Layout	**Interfacer**	JSP
Control	Command	**Coordinator**	Servlet
Business Logic	Business Delegate	**Controller**	POJO, Session EJB
Data Access	Domain Model	**Information Holder, Structurer**	JavaBean, Entity EJB
Resources	Database, Enterprise Services	**Service Provider**	RDBMS, Queues, Enterprise Service Bus

Client-Server



Advantages:

- centralized data access
  - higher security
- ease of maintenance
- scalability (in clients)
- interoperability (use of multiple sources)

Limitations:

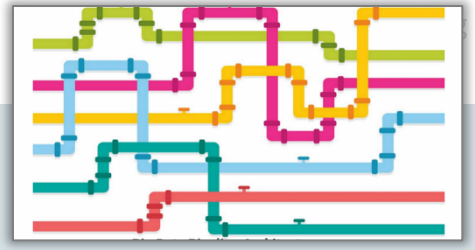
- single point of failure

WWW Sources for C/S

C/S-FAQ:
http://www.faqs.org/faqs/client-server-faq/

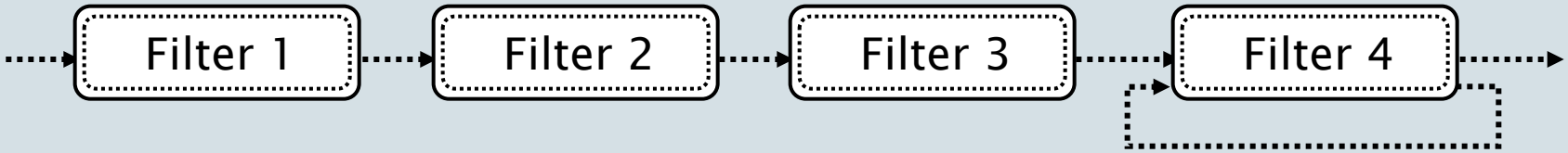
C/S info @ Software engineering institute Carnegie Mellon Univ.
http://www.sei.cmu.edu/str/descriptions/clientserver_body.html

CONTENTS



Pipe and Filter Style (1)

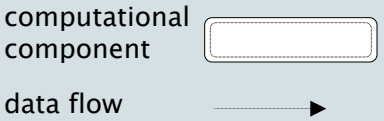
Concept: Series of filters / transformation
where each component is consumer and producer

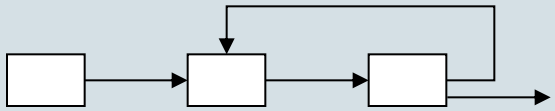


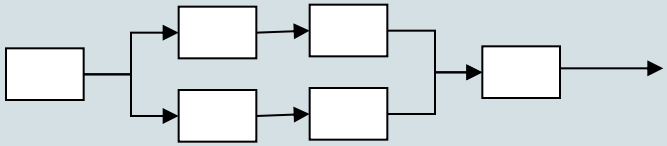
Components: filters / transformations
possibly also: sources and sinks

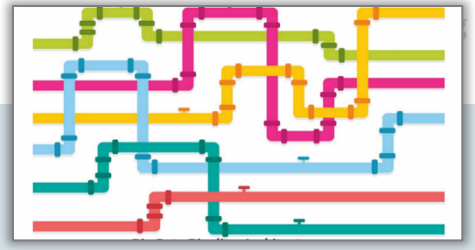
Connectors: pipes;
interaction style: streaming of data

Topology: linear; possible variations:
feedback-loops, splitting pipes





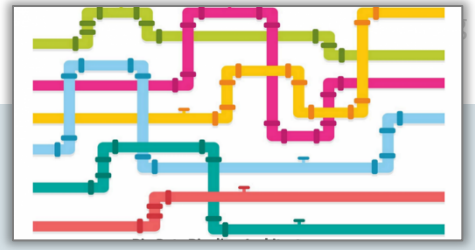




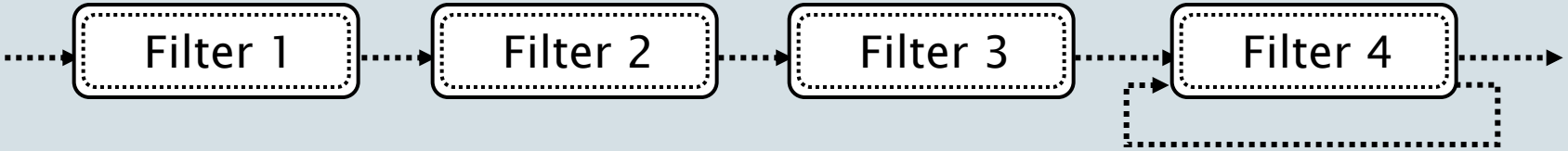
Special types of filters (?)

- **Pump (Producer/Source)**
  Produces data and puts it to an output port that is connected to the input end of a pipe.

- **Sink (Consumer)**
  Gets data from the input port that is connected to the output end of a pipe and consumes the data.



Pipe and Filter Style (2)



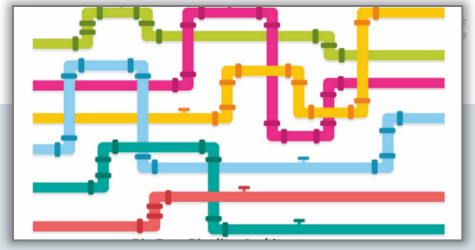
Constraints about the way filters and pipes can be joined:

- Unidirectional flow
- Control flow derived from data flow

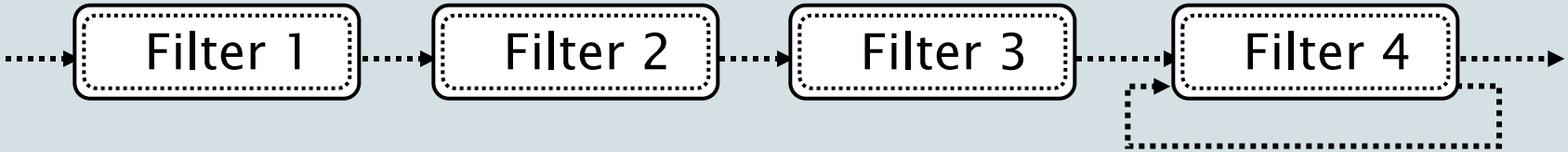
Behaviour Types:

a. **Batch sequential**
   Run to completion per transformation

b. **Continuous**
   Incremental transformation
   variants: push, pull, asynchronous



Pipe and Filter Style (3)



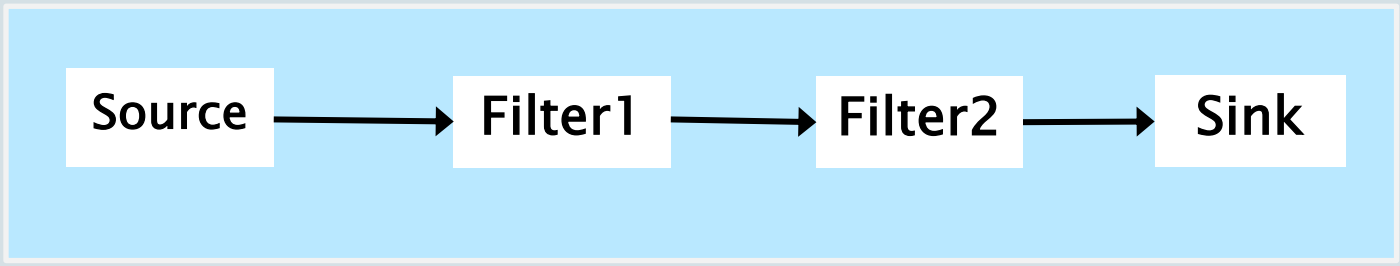
Semantic Constraints

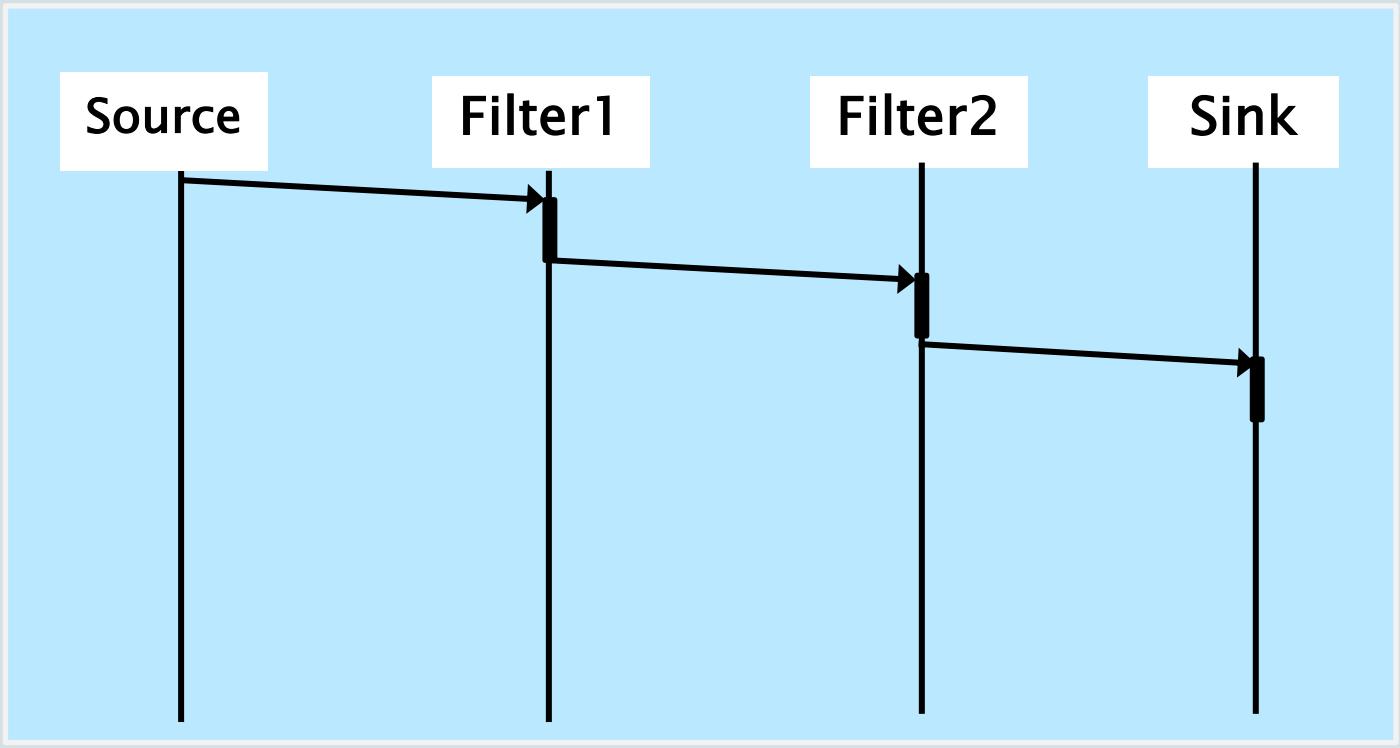
Filters are independent entities

- they do not share state
- they do not know their predecessor/successor

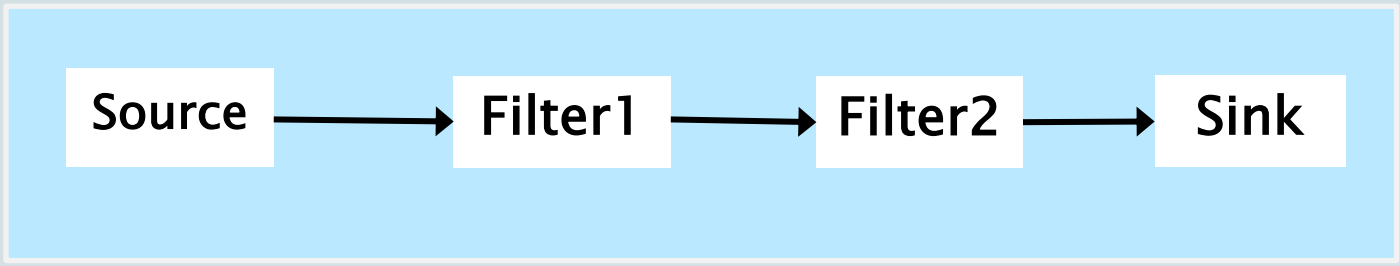
What are the dependencies between filters?
Compare this with Client Server?

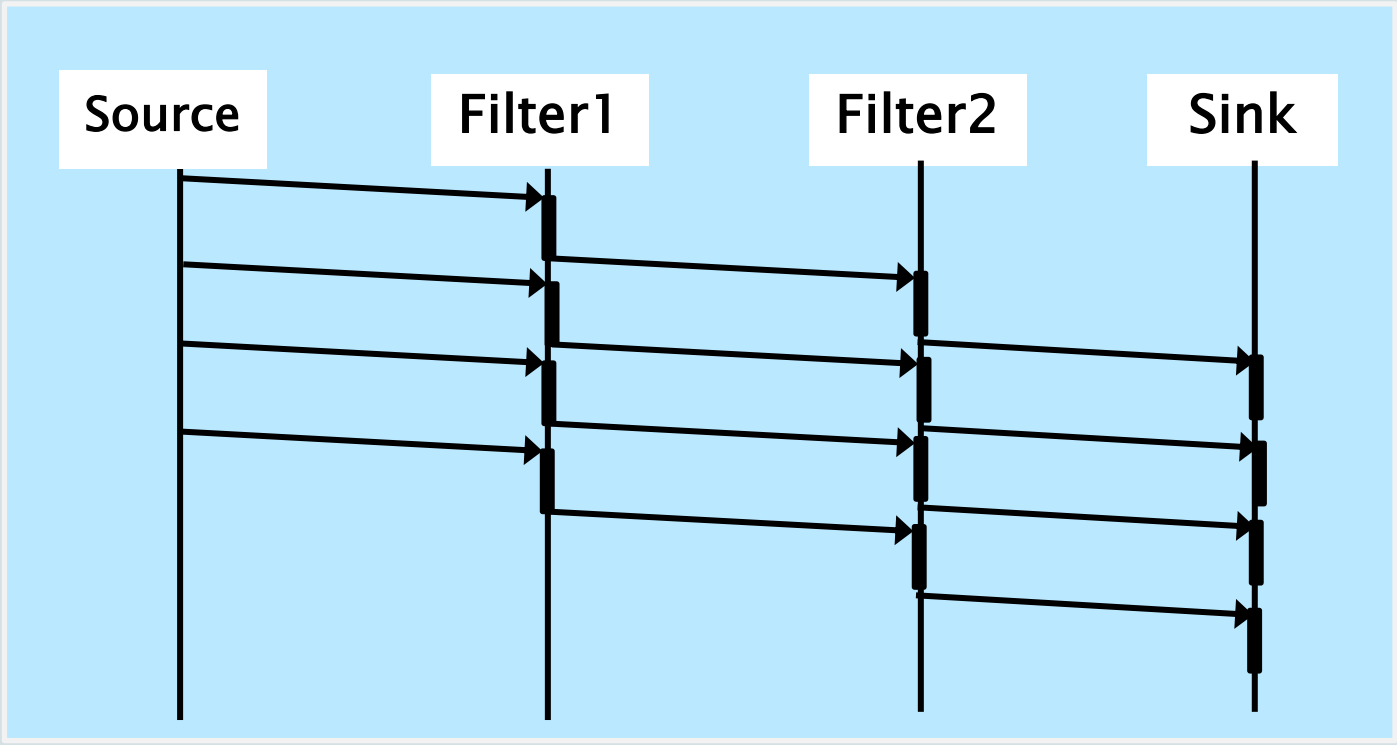
Pipe and Filter (Struct+Behaviour)



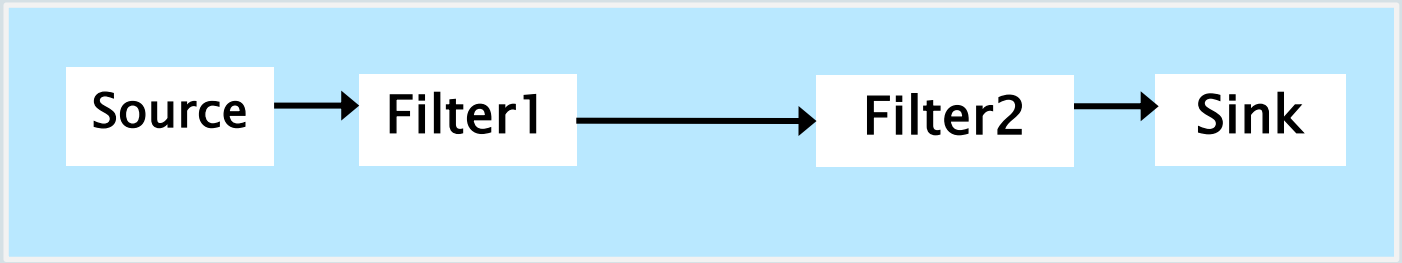


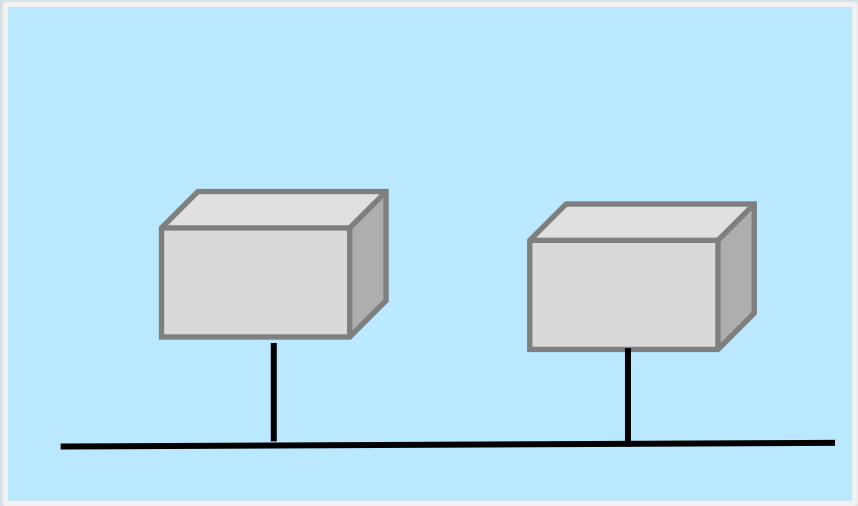
Pipe and Filter (Struct+Behaviour)





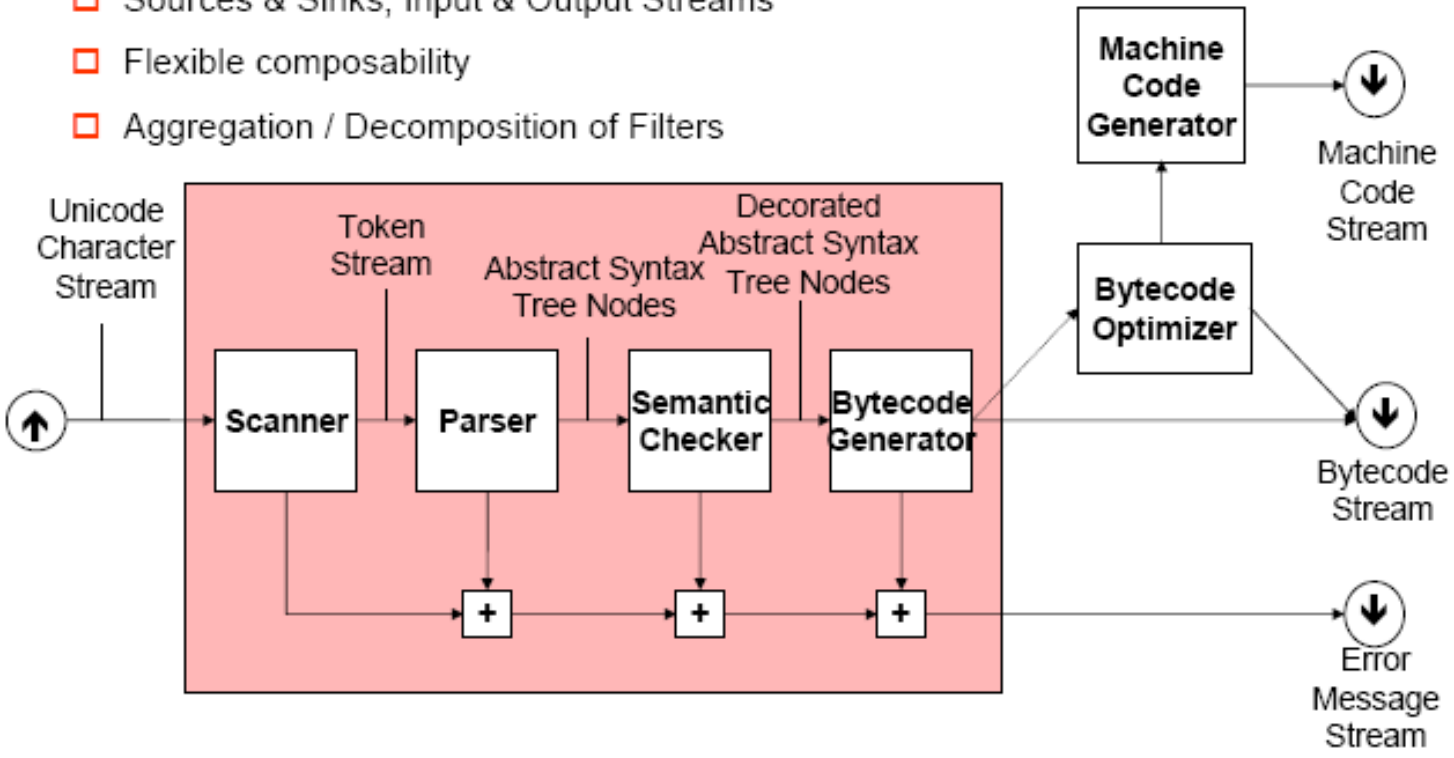
Pipe and Filter (Deployment)





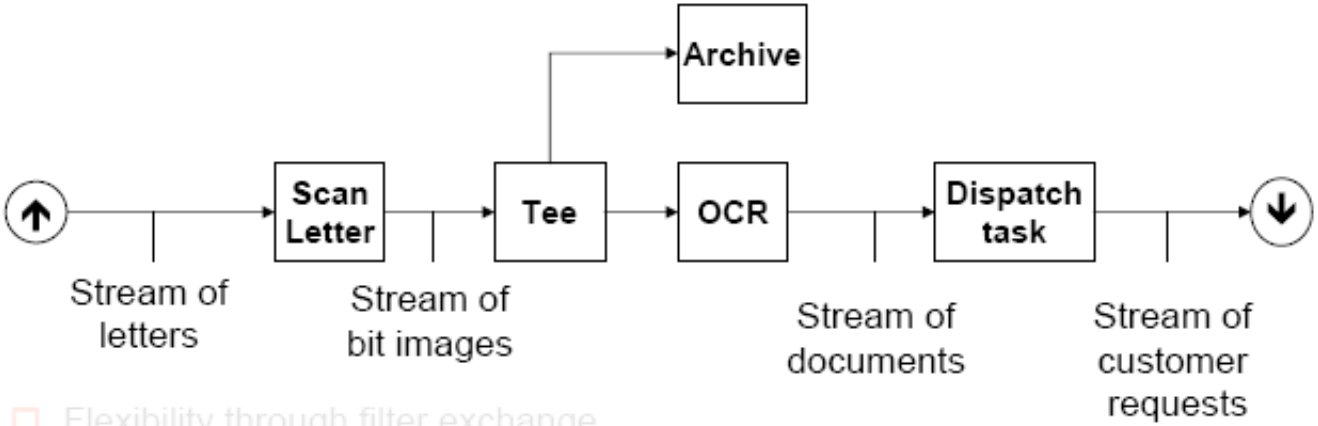
Example: P&F Compiler Architecture (1)

- Sources & Sinks, Input & Output Streams
- Flexible composability
- Aggregation / Decomposition of Filters



Example P&F Architecture

- No intermediate data structures necessary (but possible)
  (Pipeline processing subsumes batch processing)



- Flexibility through filter exchange
- Flexibility by recombination
- Reuse of filter components
- Rapid prototyping
- Parallel processing in a multiprocessor environment

Pipe and Filter Style (4a)

Advantages:

- Simplicity:
  - no complex component interactions
  - easy to analyze (deadlock, throughput, …)
- Easy to maintain and to reuse
- Filters are easy to compose (also hierarchically?)
- Can be easily made parallel or distributed

Pipe and Filter Style (4b)

Disadvantages:

- Interactive applications are difficult to create
- Filter ordering can be difficult
- Performance:
  - Enforcement of lowest common data representation, ASCII stream, may lead to (un)parse overhead
  - If output can only be produced after all input is received, an infinite input buffer is required (e.g. sort filter)
- If bounded buffers are used, deadlocks may occur

P&F Example: Linux commands

- ls | grep ‘architecture’ | sort
  - First ‘list files in directory’, then keep only files with ‘architecture’ in name, then sort this list

- ls | sort ls | grep ‘architecture’

This rearrangement works because components have the same input and output: the ‘lowest common denominator’ is a stream of lines of characters.

Pipe and Filter Style (5) Quality Factors

Extendibility: extends easily with new filters

Flexibility: – functionality of filters can be easily redefined,
– control can be re-routed
(both at design-time, run-time is difficult)

Robustness: 'weakest link' is limitation

Security: –

Performance: allows straightforward parallelisation

Pipe and Filter Style (6) Application Context

Rules of thumb for choosing pipe-and-filter (o.a. from Shaw/Buschman):

- if a system can be described by a **regular interaction pattern** of a collection of processing units at the same level of abstraction; e.g. a series of incremental stages (horizontal composition of functionality);
- if the computation involves the **transformation of streams of data** (processes with limited fan-in/fan-out)

Hint: use a looped-pipe-and-filter if the system does continuous controlling of a physical system

Typical application domain: signal processing

Summary

Conceptual Integrity
- uniformity, harmony, consistency in design

Architectural Styles
- Every Architect should have a standard set of architectural styles in his/her repertoire
  - Client/Server
  - Pipe and Filter

The choice for a style can make a big difference in the *quality properties* of a system