SOFTWARE ENGINEERING

CHAPTER-9
ARCHITECTURAL DESIGN

Software Engineering: A Practitioner's Approach, 7th edition

Originated by Roger S. Pressman

Software School, Fudan University
Spring Semester, 2016

Software Architecture

The software architecture of a program or computing system is the *structure or structures* of the system, which comprise the software ***components***, the ***externally visible properties*** of those components, and the ***relationships*** among them.

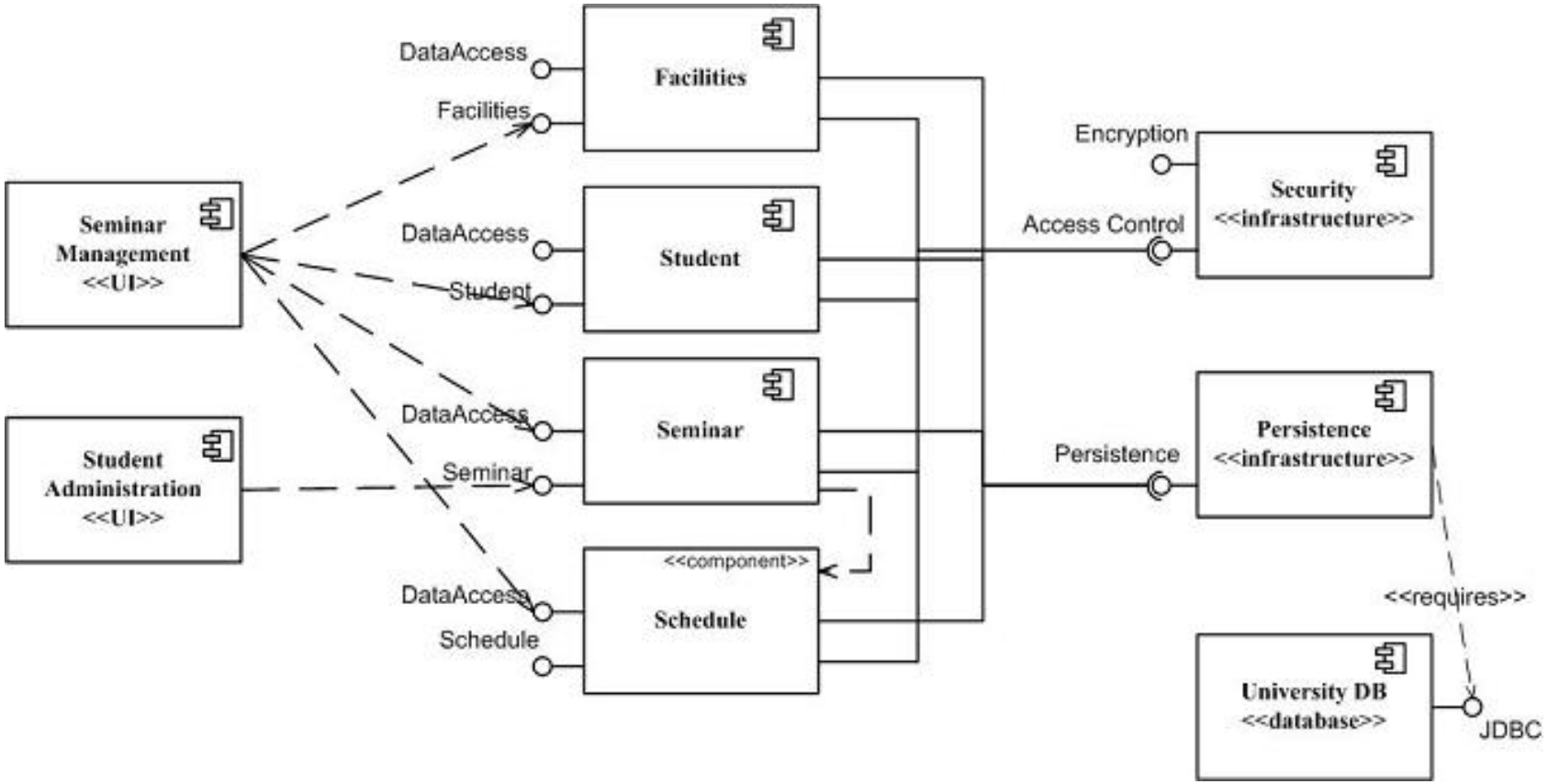
— *Bass. et al.*

Software Architecture (Cont.)

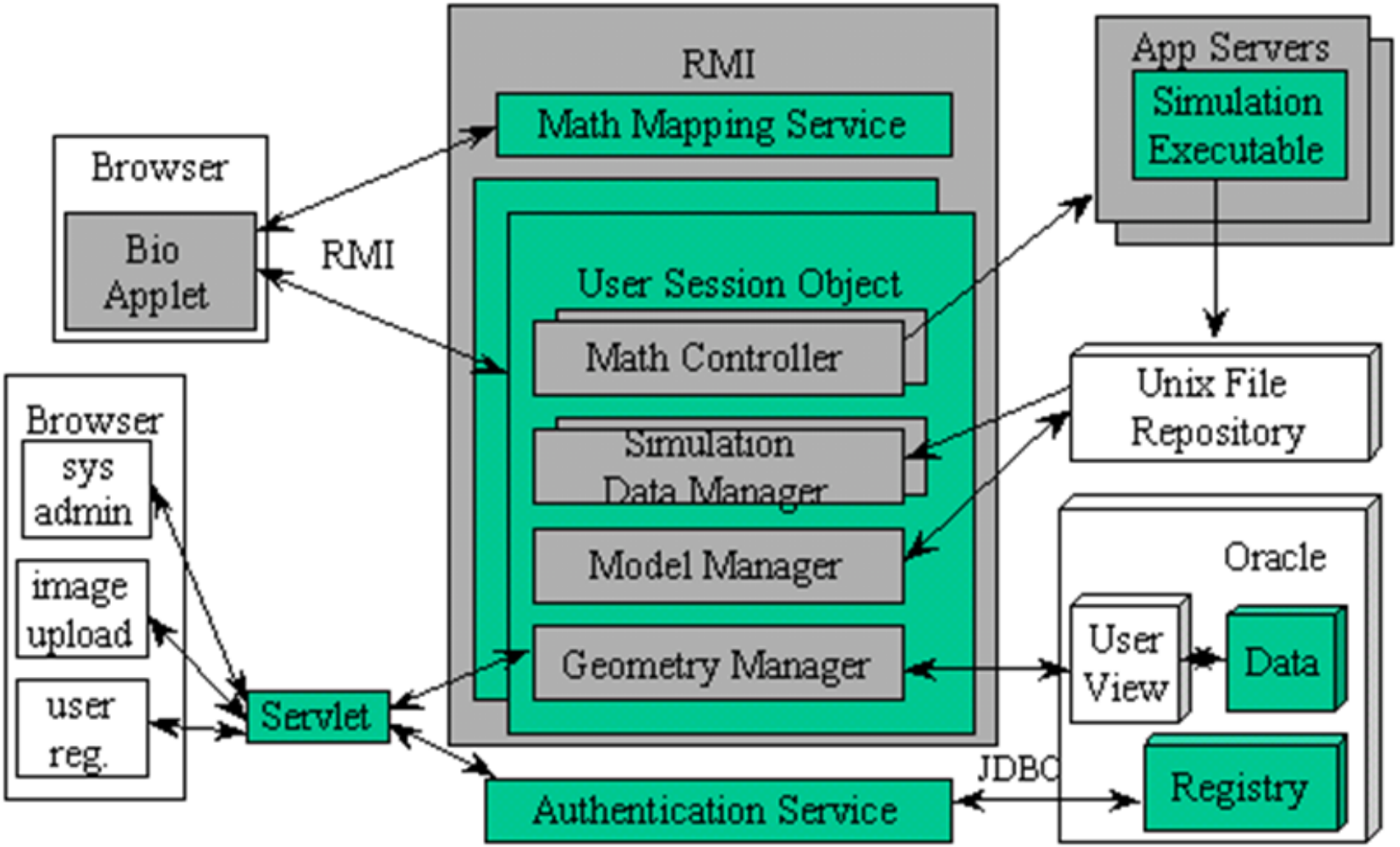
- Software Components
  - Structural elements representing high-level decompositions
  - Can be something as simple as program modules or classes
  - Can also be extended to include databases and "middleware", e.g. for networking
- External Properties of Components
  - Functional properties, e.g. interfaces, protocols
  - Nonfunctional properties, e.g. promised throughput
- Relationships Among Components
  - Static structural relations
  - Dynamic interactions

And, other convention, standard, constraints that may influence the integration

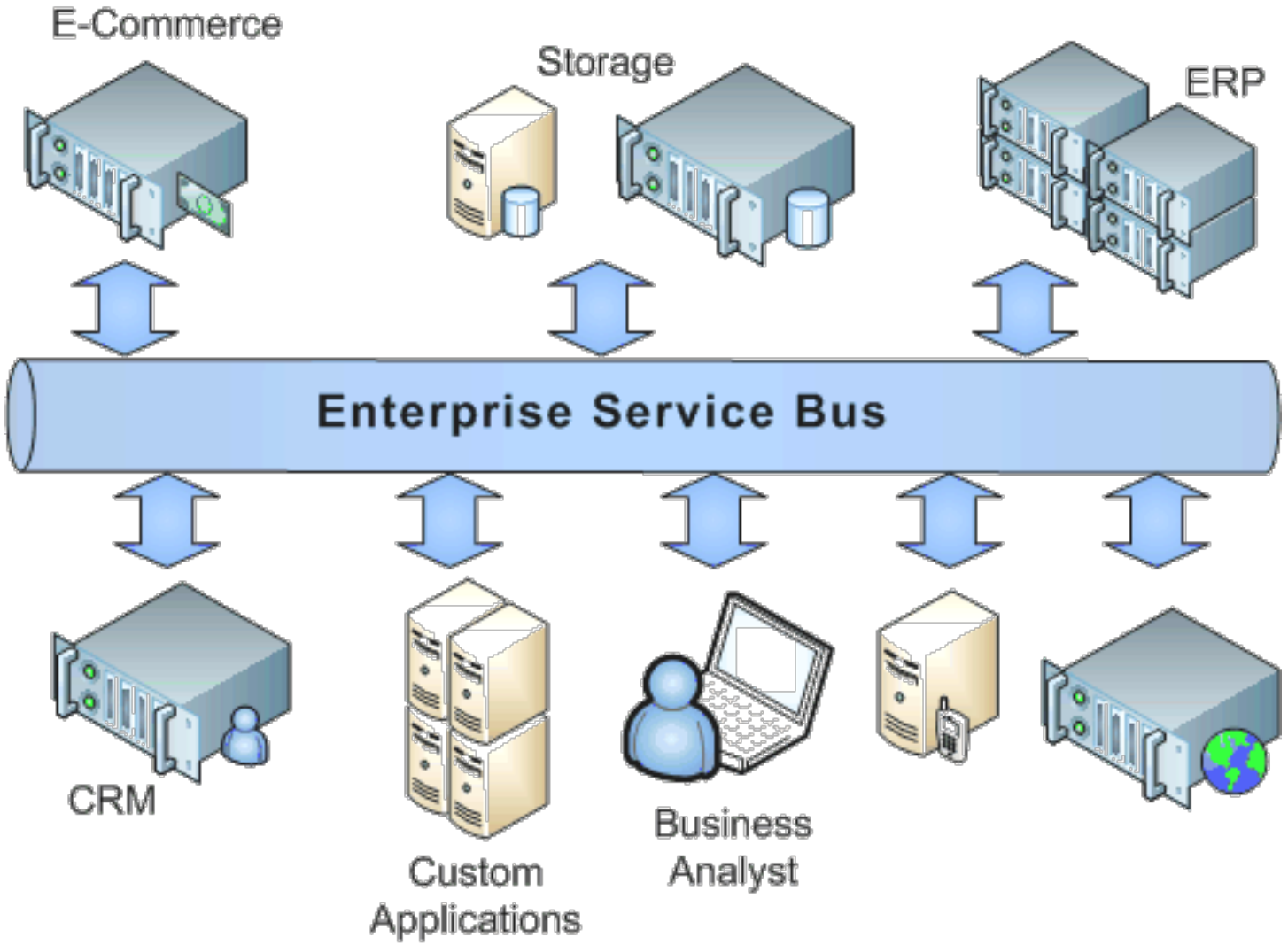
Architecture: Example



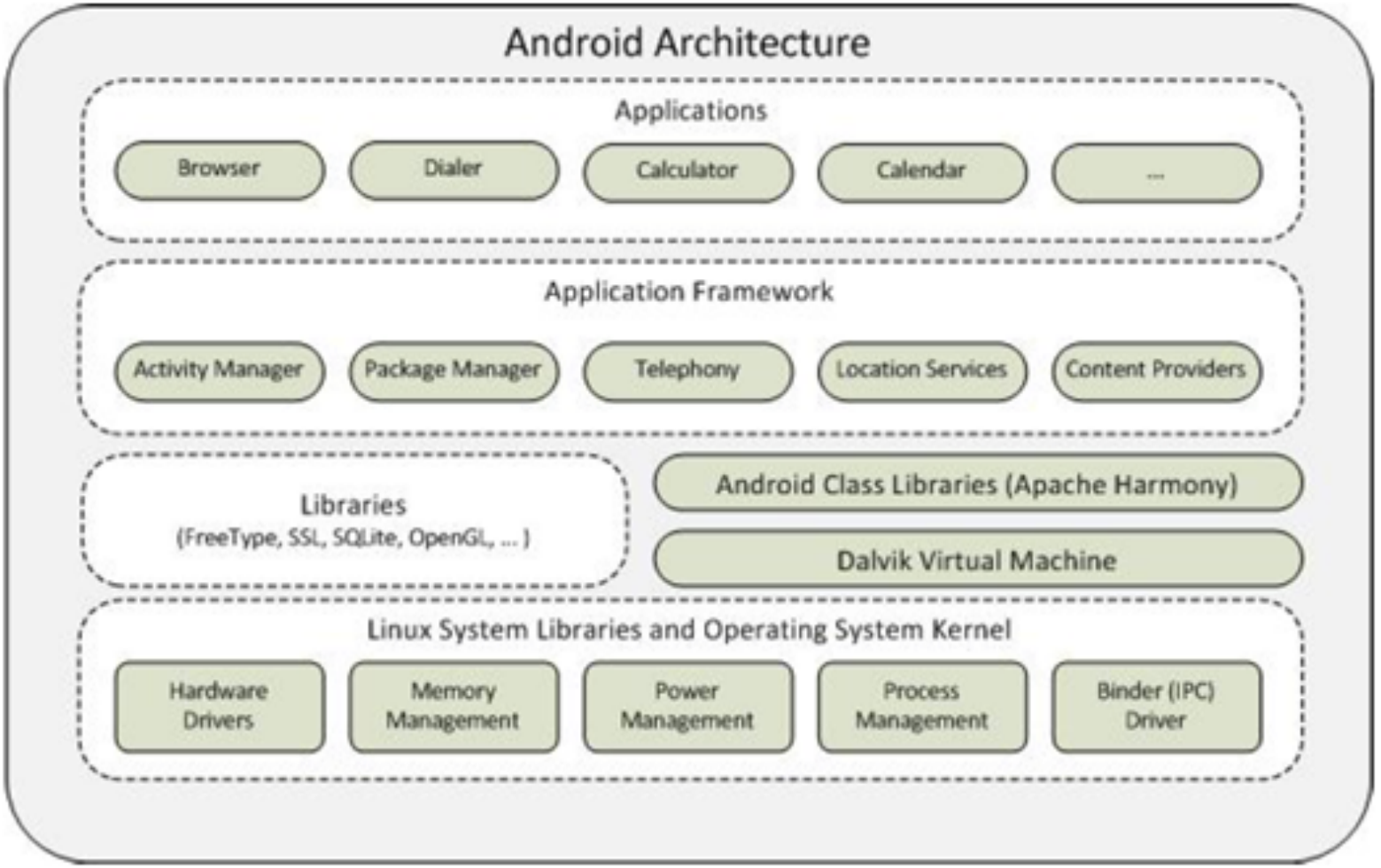
Architecture: Example



Architecture: Example



Architecture: Example



WHY ARCHITECTURE?

- Architecture is a representation (high-level abstraction) of a system that enables the software engineer to
  - analyze the effectiveness of the design in meeting its stated requirements
  - consider architectural alternatives at a stage when making design changes is still relatively easy
  - reduce the risks associated with the construction of the software

Why is Architecture Important?

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture "constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together" [BAS03].

Architectural Descriptions

- The IEEE Computer Society has proposed IEEE-Std-1471-2000, *Recommended Practice for Architectural Description of Software-Intensive System,* [IEE00]
  - to establish a conceptual framework and vocabulary for use during the design of software architecture,
  - to provide detailed guidelines for representing an architectural description, and
  - to encourage sound architectural design practices.
- The IEEE Standard defines an *architectural description* (AD) as a "a collection of products to document an architecture."
  - The description itself is represented using multiple views, where each *view* is "a representation of a whole system from the perspective of a related set of [stakeholder] concerns."

Data Design

- At the architectural level … → Database
  - Design of one or more **databases** to support the application architecture
  - Design of methods for "mining" the content of multiple databases
    - navigate through existing databases in an attempt to extract appropriate business-level information
    - Design of a data warehouse—a large, independent database that has access to the data that are stored in databases that serve the set of applications required by a business

Data Design (Cont.)

- At the component level … → Data structure design
  - refine data objects and develop a set of data abstractions
  - implement data object attributes as one or more data structures
  - review data structures to ensure that appropriate relationships have been established
  - simplify data structures as required

Architectural Styles

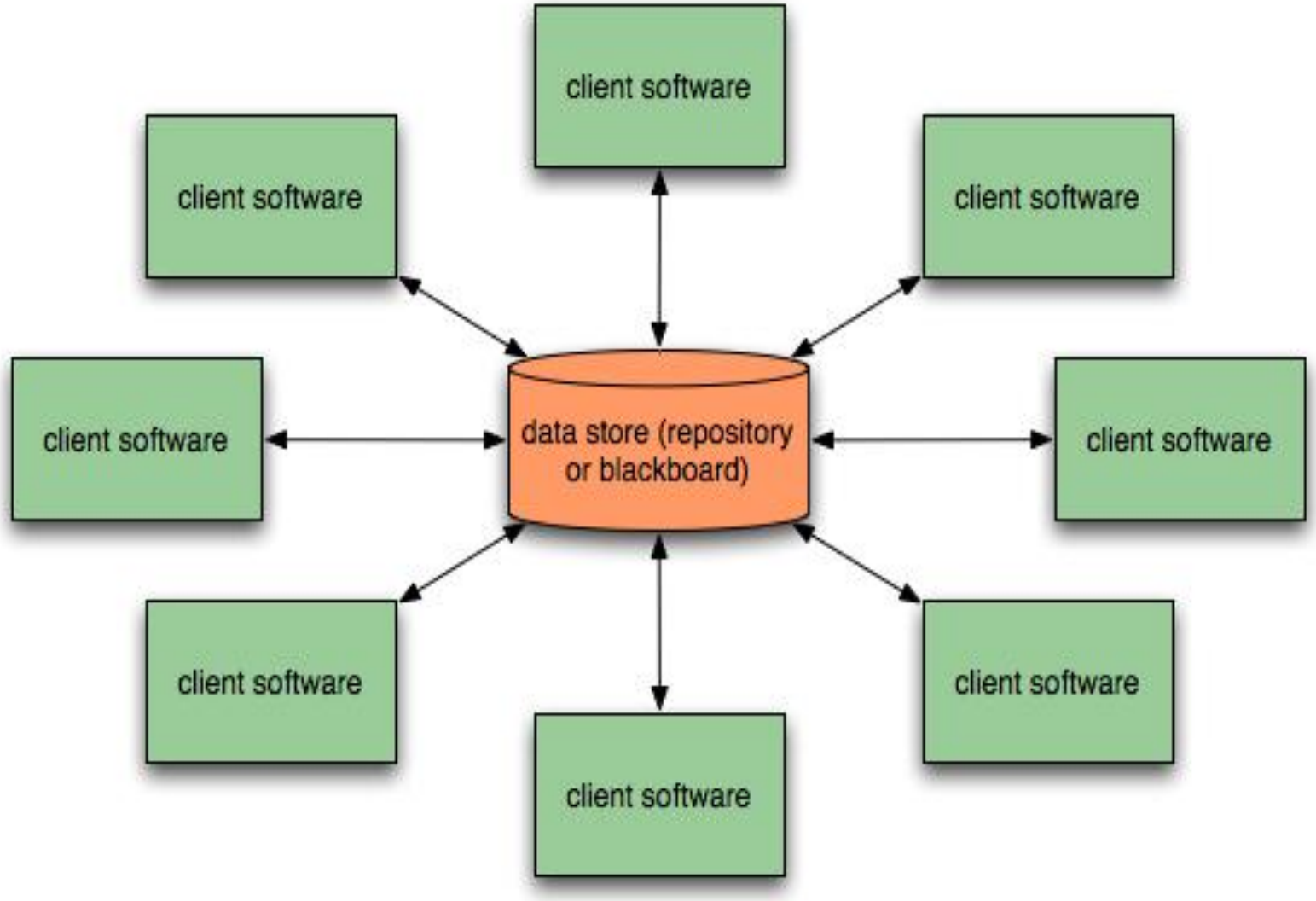
Architectural Styles

- Each style describes a system category that encompasses:
  - a set of components (e.g. a database, computational modules) that perform a function required by a system,
  - a set of connectors that enable "communication, coordination, and cooperation" among components,
  - constraints that define how components can be integrated to form the system, and
  - semantic models that enable a designer to understand the overall properties of a system.

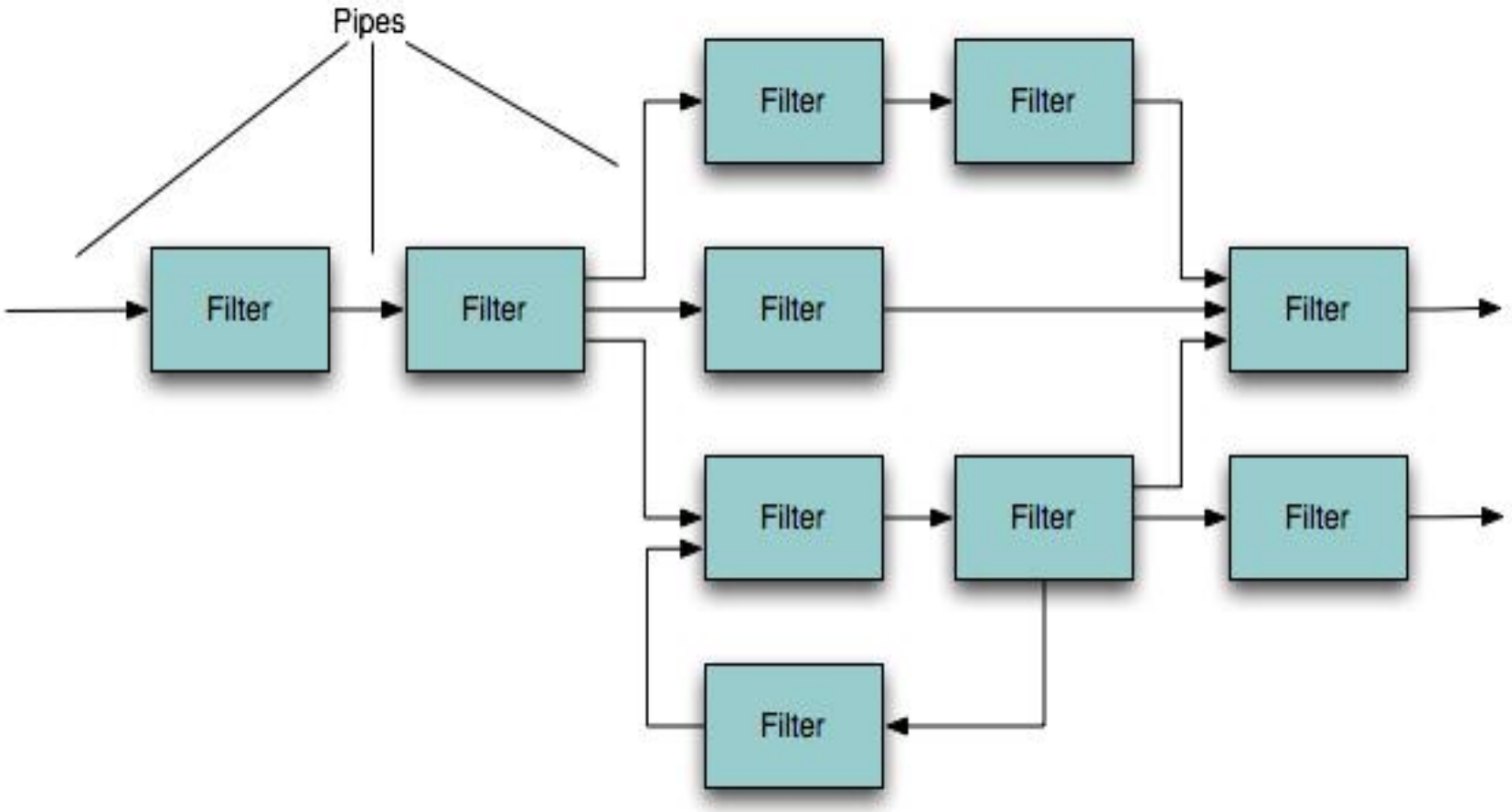
Specific Styles

- Data-centered architecture
- Data flow architecture
- Call and return architecture
- Object-oriented architecture
- Layered architecture

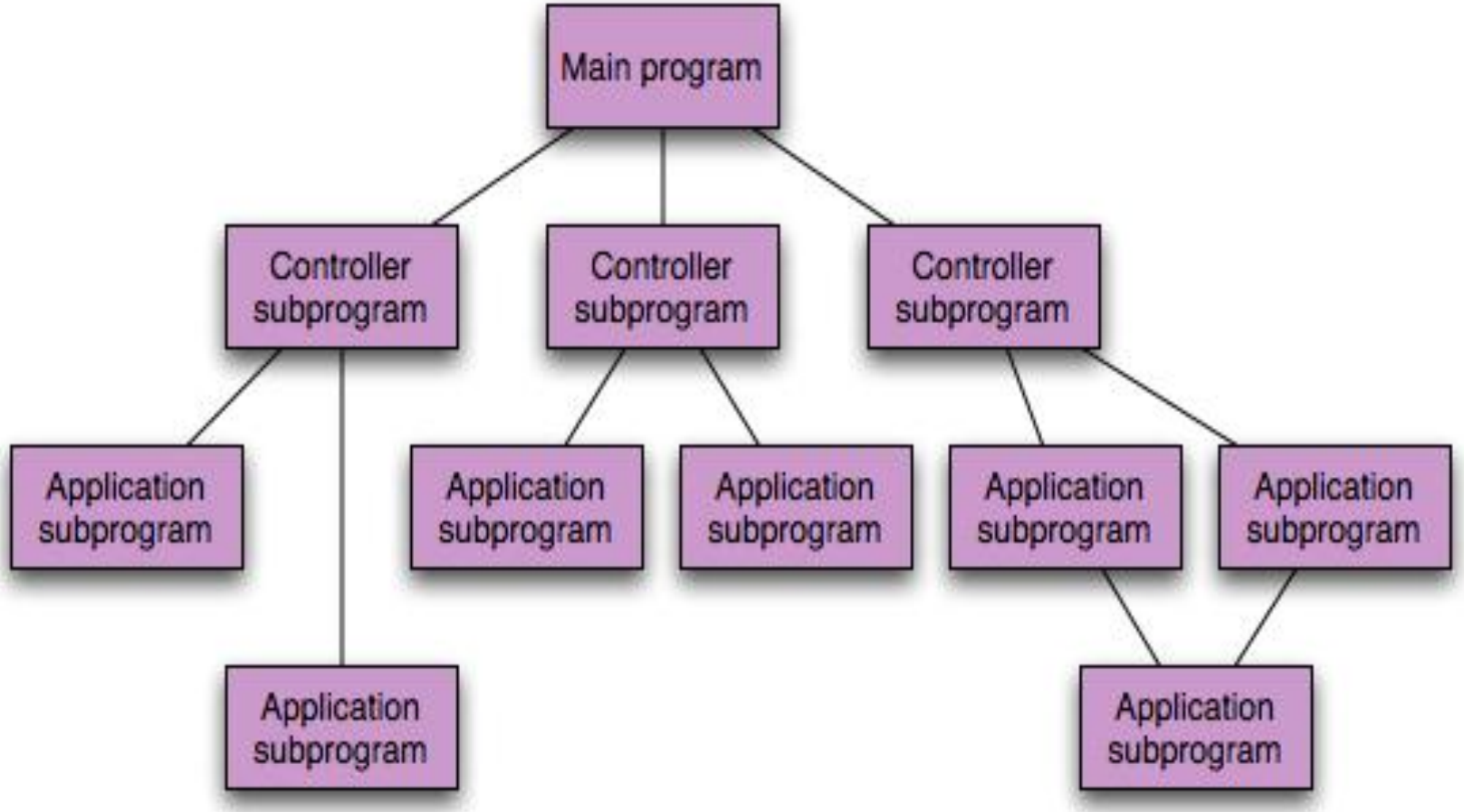
Data-Centered Architecture



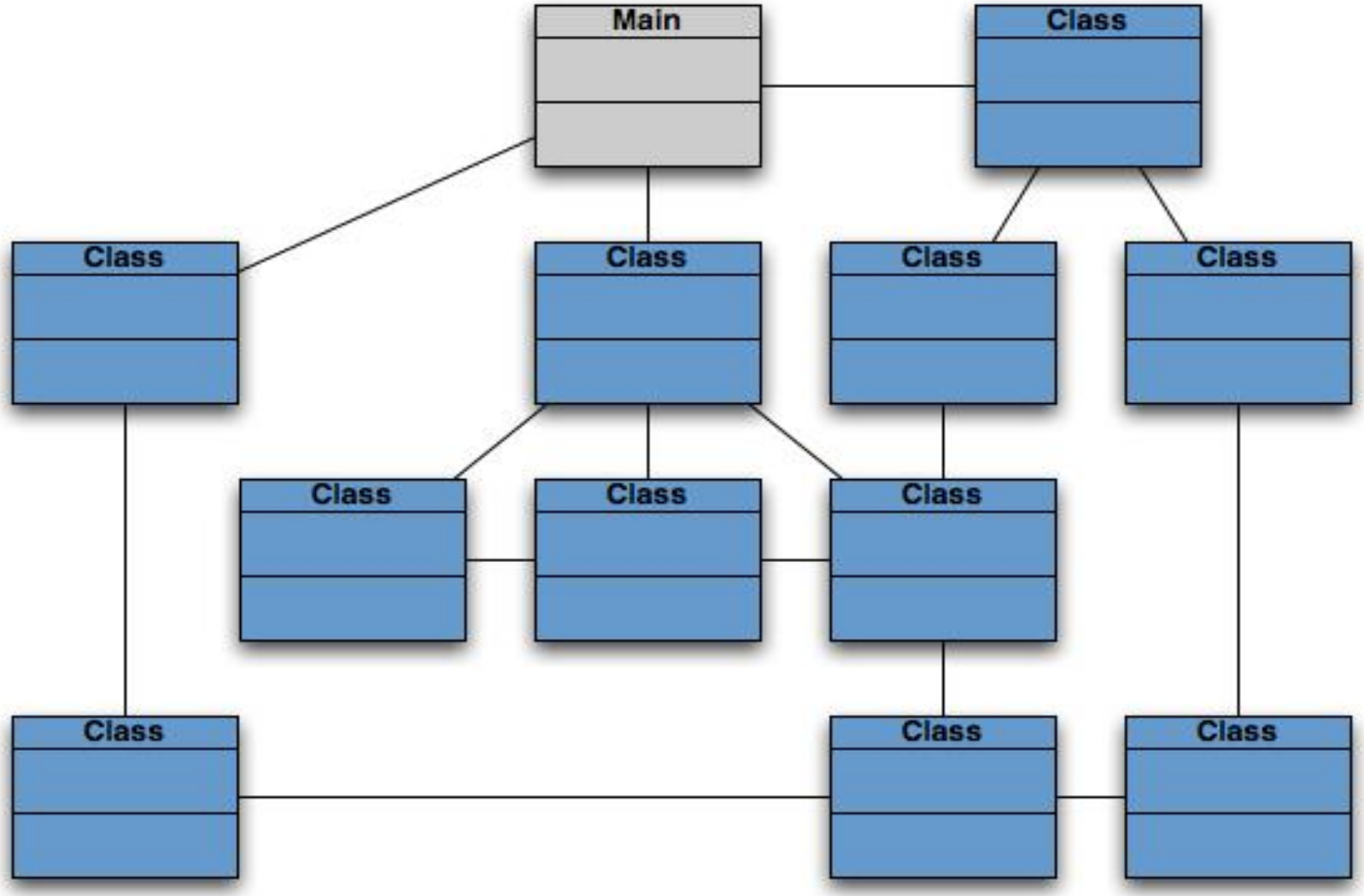
Data-Flow Architecture



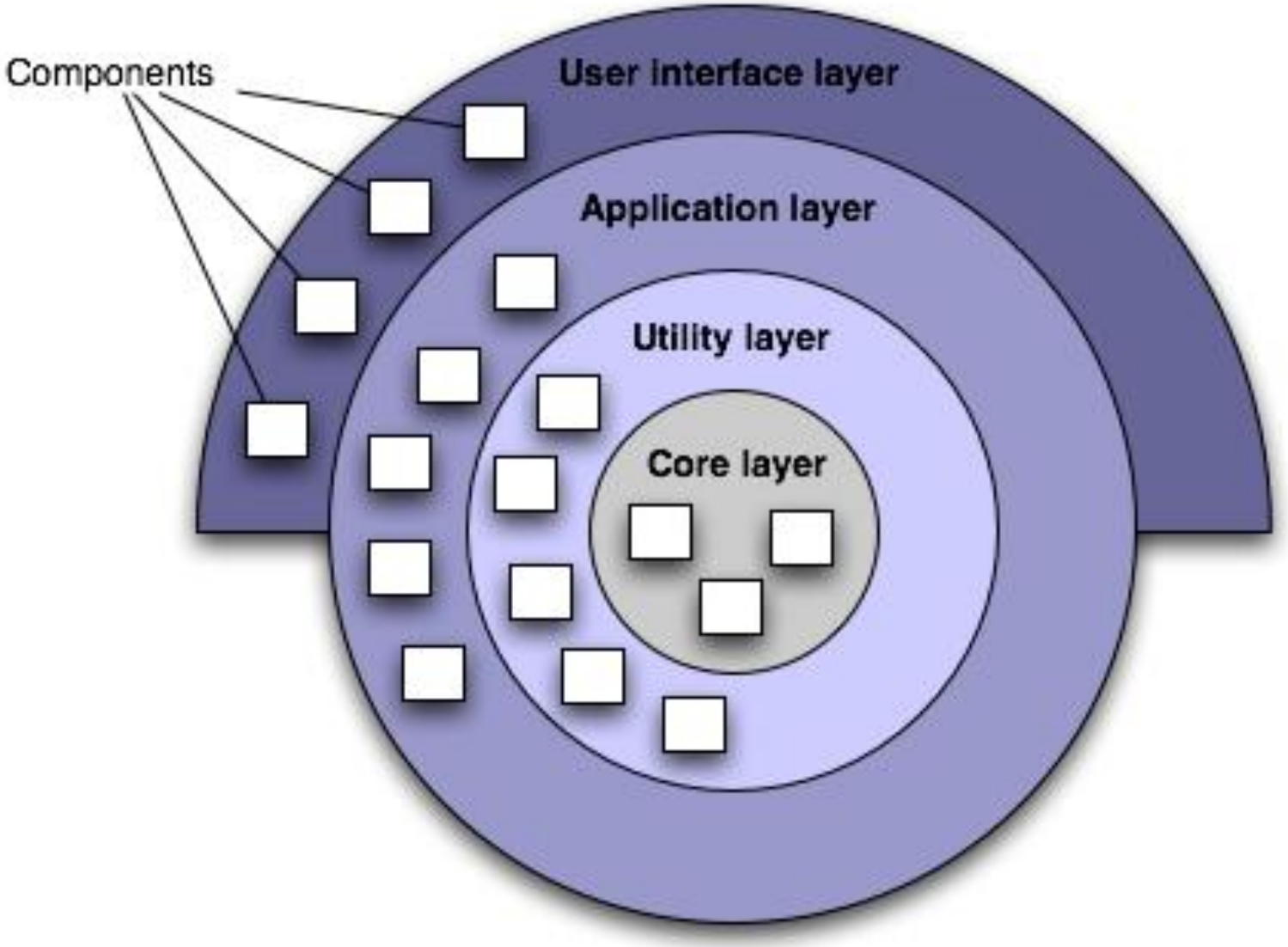
Call and Return Architecture



Object-Oriented Architecture



Layered Architecture



Architectural Patterns

- Concurrency – to simulate parallelism
  - operating system process management
  - task scheduler
- Persistence – Data persists if it survives past the execution of the process that created it.
  - database management system
  - application level persistence
- Distribution – in a distributed environment
  - broker – "middle-man" between the client component and a server component

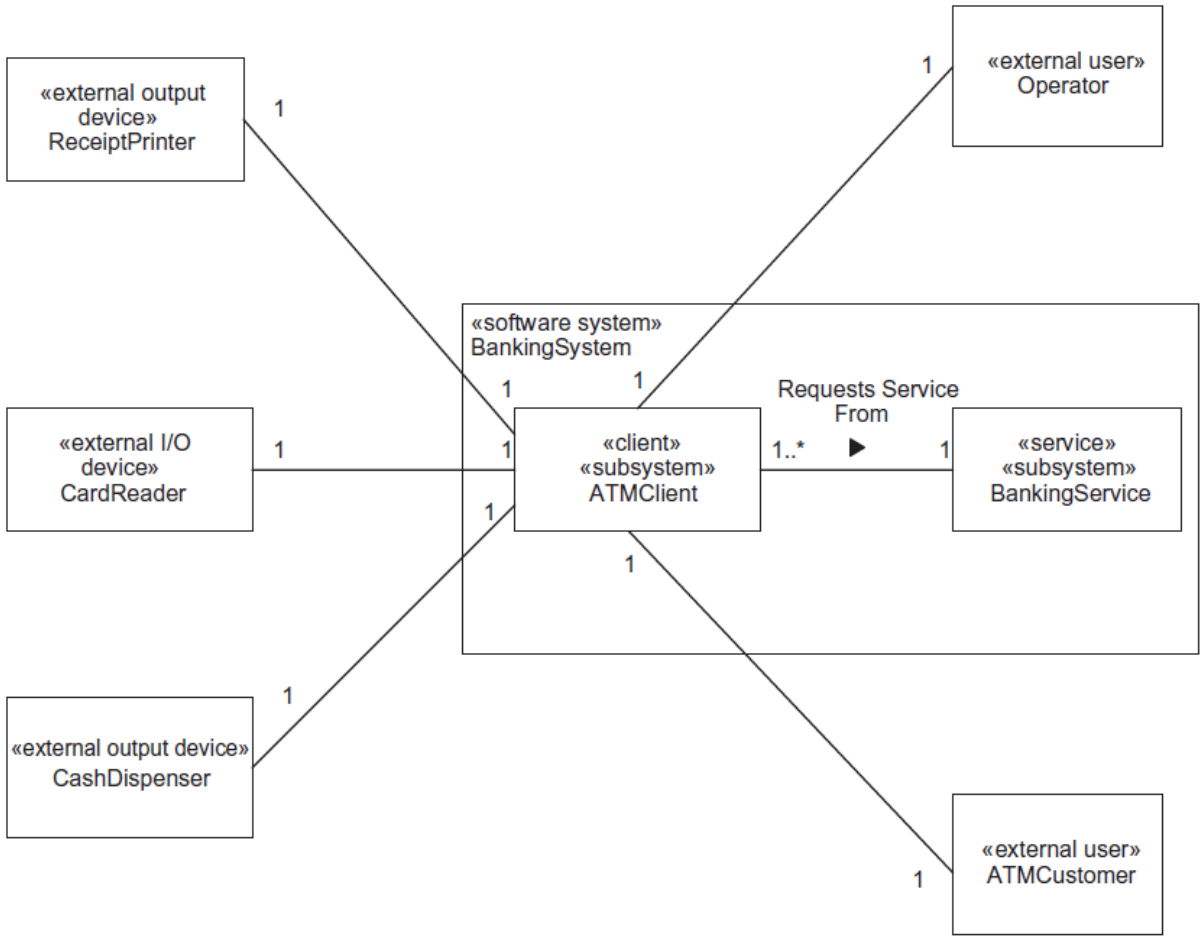
Multiple Views of Software Architecture

- Structural View
- Dynamic View
- Deployment View
- Component-based View

Structural View

- Structural view is a static view, which does not change with time
- At the highest level, subsystems are depicted on a class diagram
- Static structural relationship between subsystems are represented as
  - Composite or aggregate classes
  - Multiplicity of associations among them

Structural View (Example)

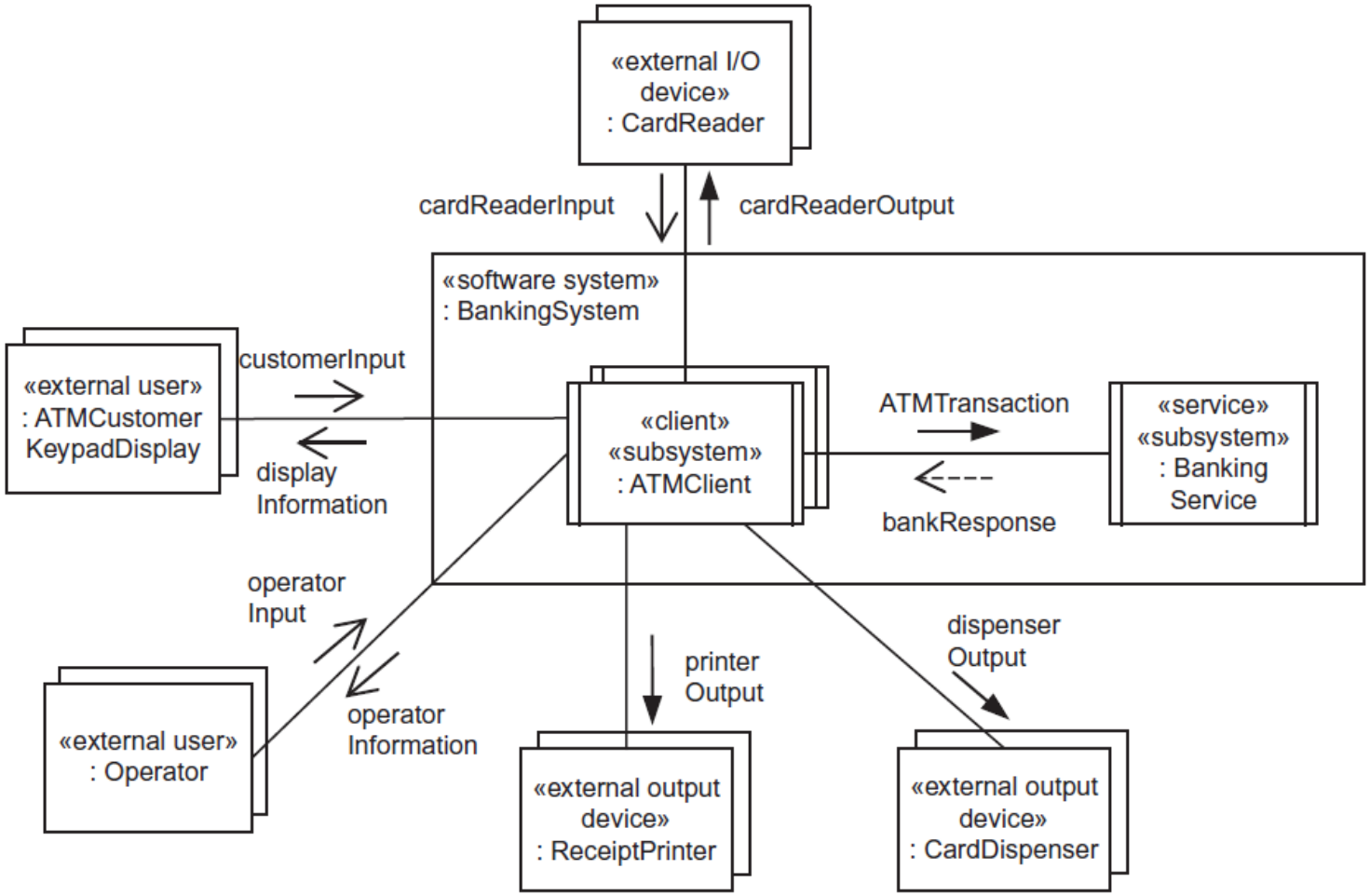


Structural view of client/server software architecture: high-level class diagram for Banking System

Dynamic View

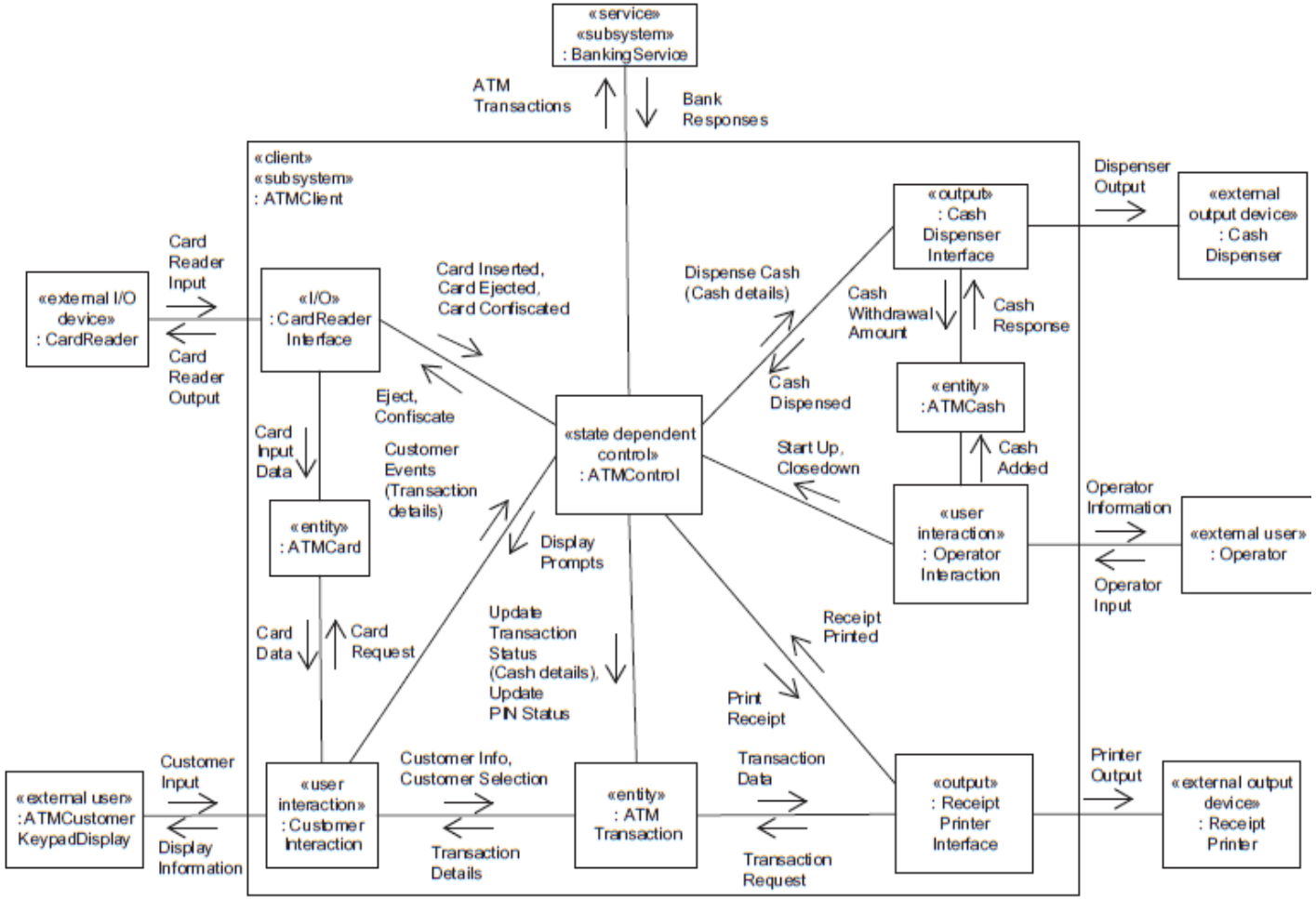
- Dynamic view is a behavioral view
- Depicted on a communication diagram
- A subsystem communication diagram shows the subsystems and the message communication between them
- Subsystems can be deployed to different nodes, so they are depicted as concurrent components: they execute in parallel and communicate with each other over a network

Dynamic View (Example)



Dynamic view of client/server software architecture: high-level communication diagram for Banking System

Dynamic View (Example)

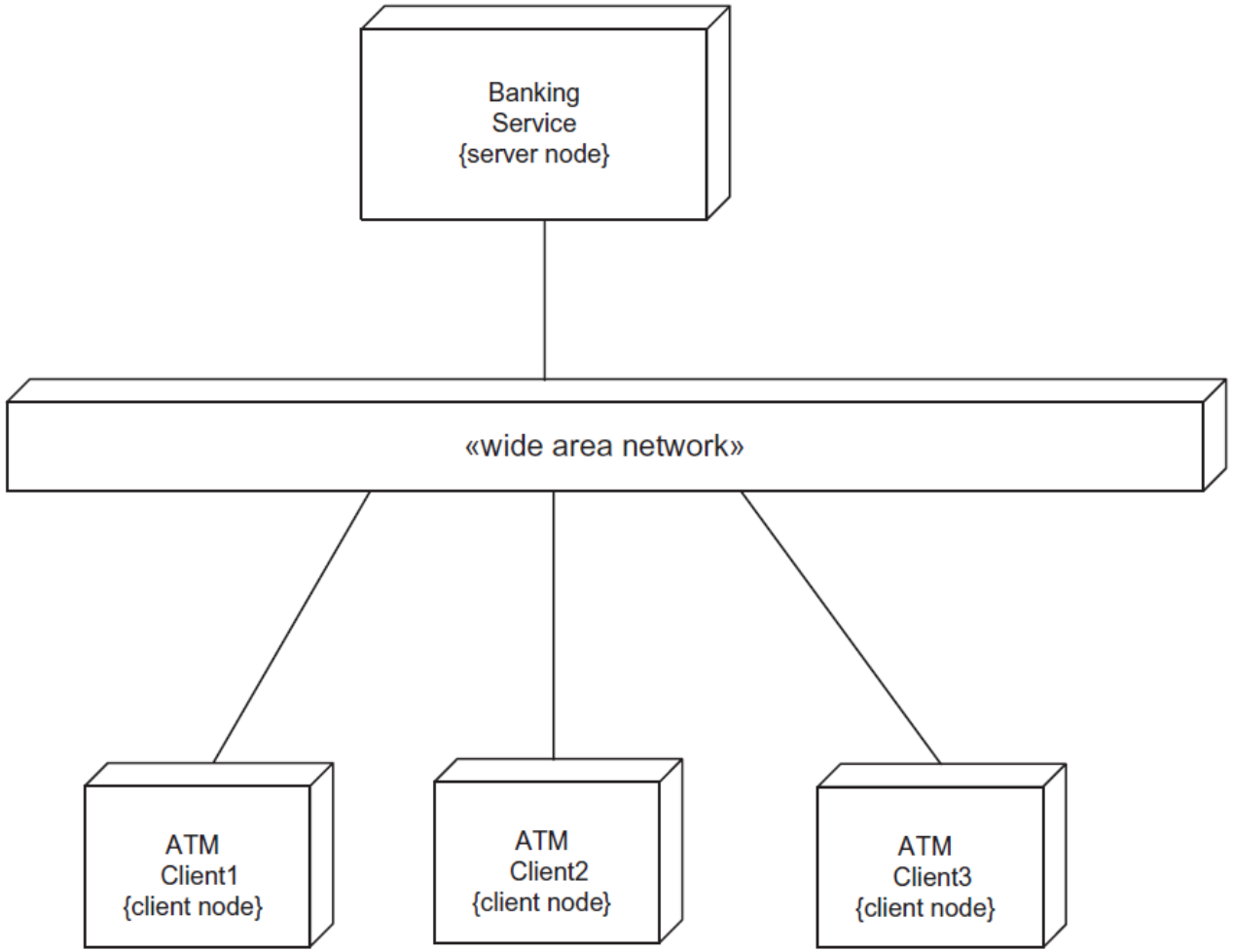


Integrated communication diagram for ATM Client subsystem

Deployment View

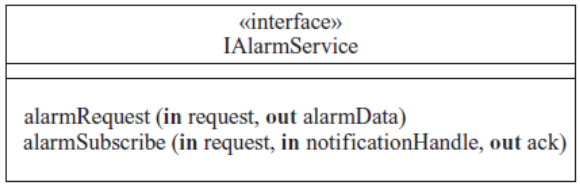
- Deployment view depicts the physical configuration of the software architecture, in particular how the subsystems of the architecture are allocated to physical nodes in a distributed configuration
- A deployment diagram
  - It can depict a specific deployment with a fixed number of nodes
  - Alternatively, it can depict the overall structure of the deployment – for example, identifying that a subsystem can have many instances, but not depicting the specific number of instances

Deployment View (Example)

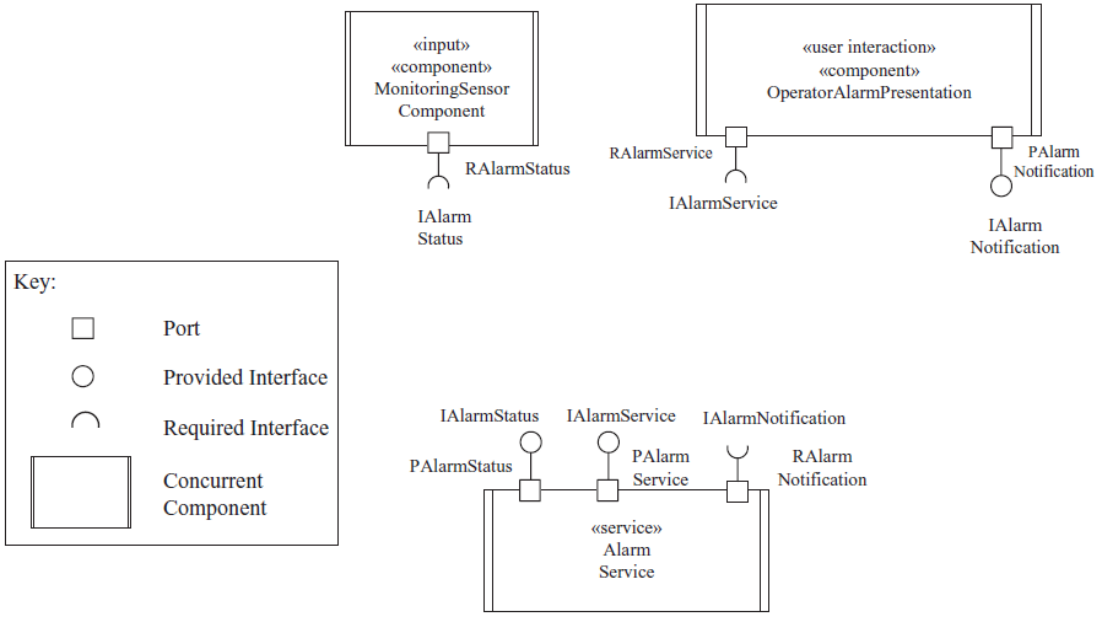


Deployment view of client/server software architecture: deployment diagram

COMPONENT-BASED VIEW

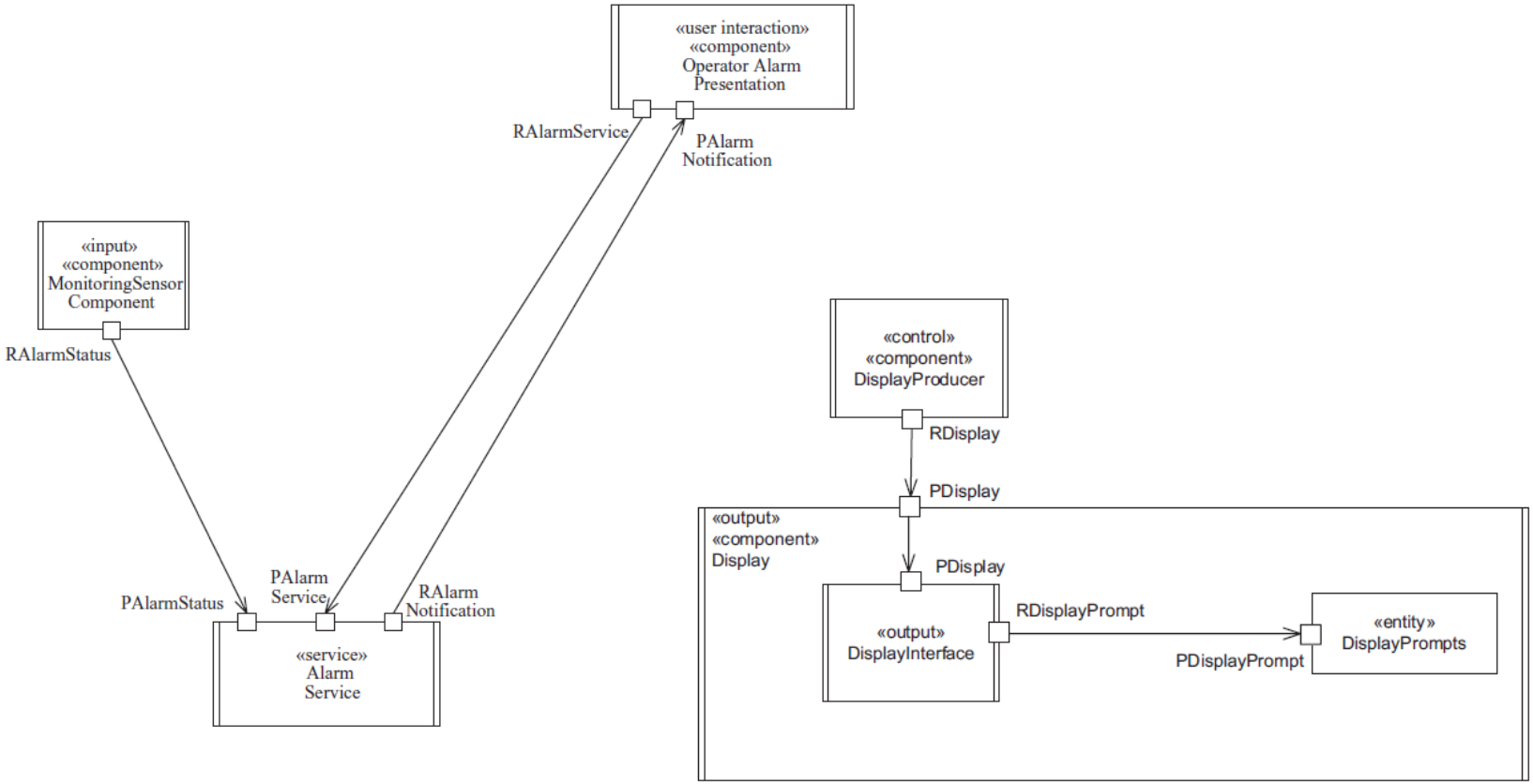


Example of component interfaces



Examples of component ports, with provided and required interfaces

COMPONENT-BASED VIEW



Analyzing Architectural Design

- 1. Collect scenarios.
- 2. Elicit requirements, constraints, and environment description.
- 3. Describe the architectural styles / patterns that have been chosen to address the scenarios and requirements:
  - module view
  - process view
  - data flow view
- 4. Evaluate quality attributes by considered each attribute in isolation.
- 5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style.
- 6. Critique candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

Assessing Alternative Architectural Designs

- Assessing Architecture Designs
  - Design often results in a number of architectural alternatives
  - Each should be assessed to determine the most appropriate for the problem to be solved
- ATAM (SEI): ***A***rchitecture ***T***rade-off ***A***nalysis ***M***ethod

ATAM Activities

- 1. Collect scenarios: a set of use cases is developed to represent the system from the user's point of view
- 2. Elicit requirements, constraints and environment description
- 3. Describe the architectural styles/patterns that have been choose to address the scenarios and requirements
- 4. Evaluate quality attributes by considering each attribute in isolation
- 5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style
- 6. Critique candidate architecture using the sensitivity analysis conducted in step 5

See 《ATAM: Method for Architecture Evaluation》

Architectural Complexity

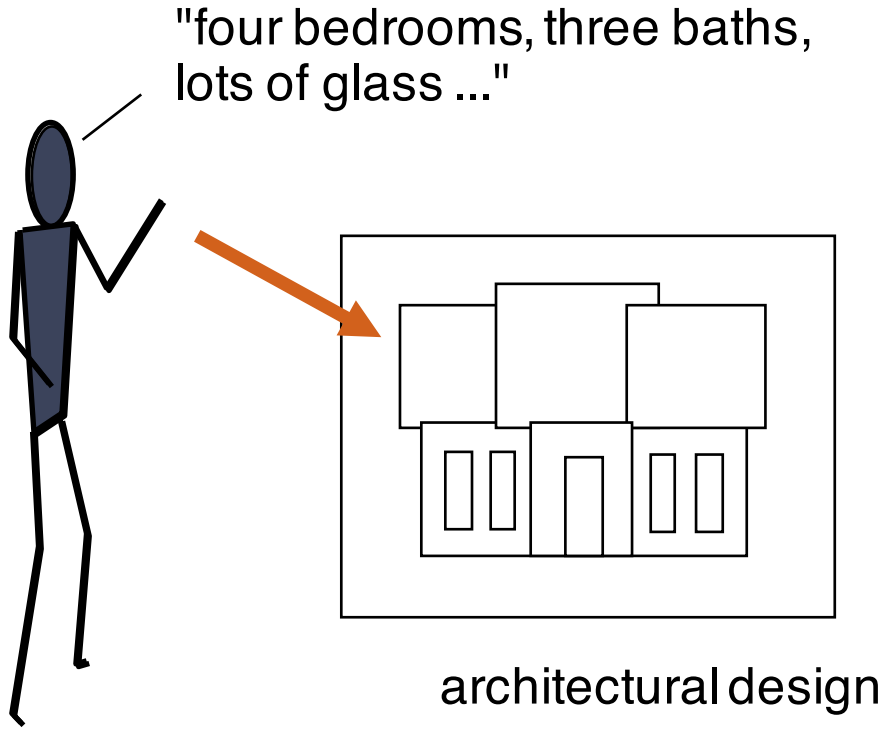
- the overall complexity of a proposed architecture is assessed by considering the dependencies between components within the architecture [Zha98]
  - *Sharing dependencies* represent dependence relationships among consumers who use the same resource or producers who produce for the same consumers.
  - *Flow dependencies* represent dependence relationships between producers and consumers of resources.
  - *Constrained dependencies* represent constraints on the relative flow of control among a set of activities.

ADL

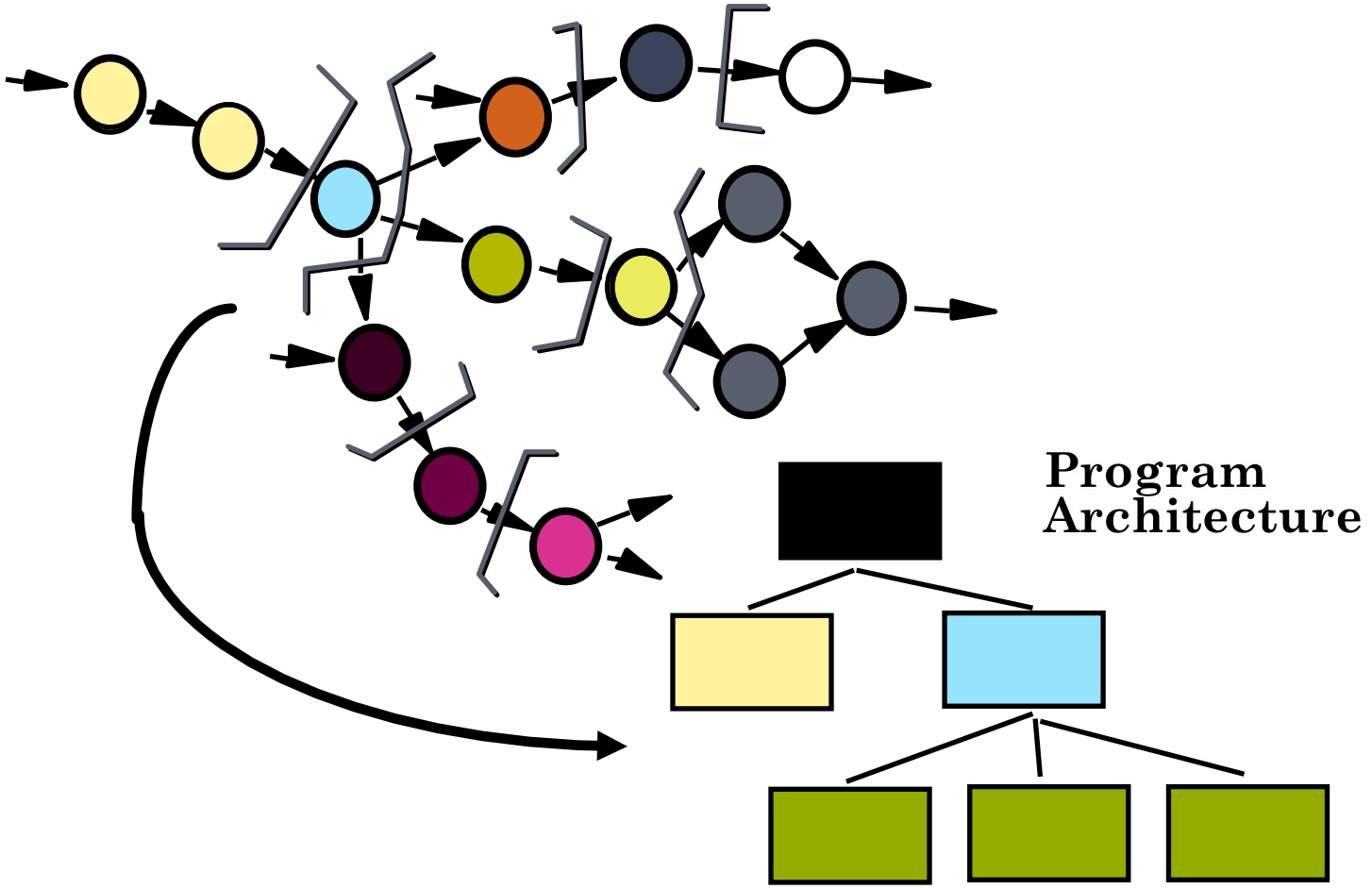
- *Architectural description language* (ADL) provides a semantics and syntax for describing a software architecture
- Provide the designer with the ability to:
  - decompose architectural components
  - compose individual components into larger architectural blocks and
  - represent interfaces (connection mechanisms) between components.

An Architectural Design Method

customer requirements

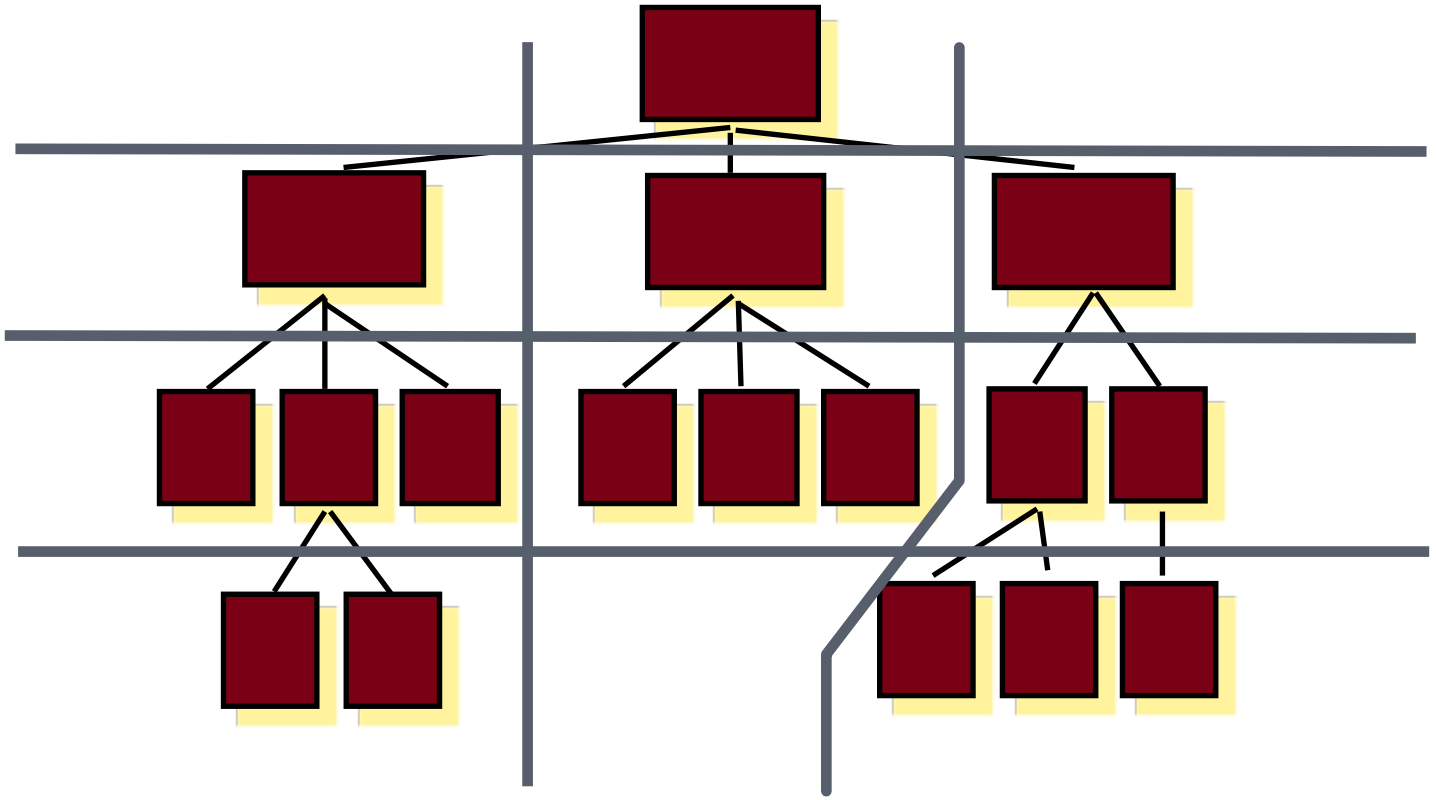


Deriving Program Architecture



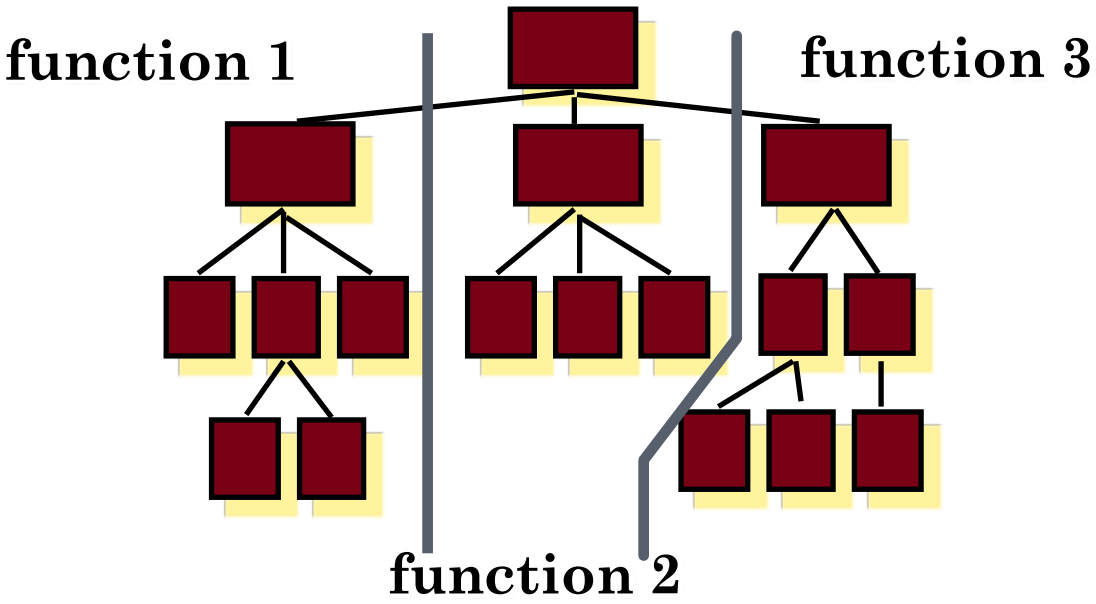
Partitioning the Architecture

"horizontal" and "vertical" partitioning are required



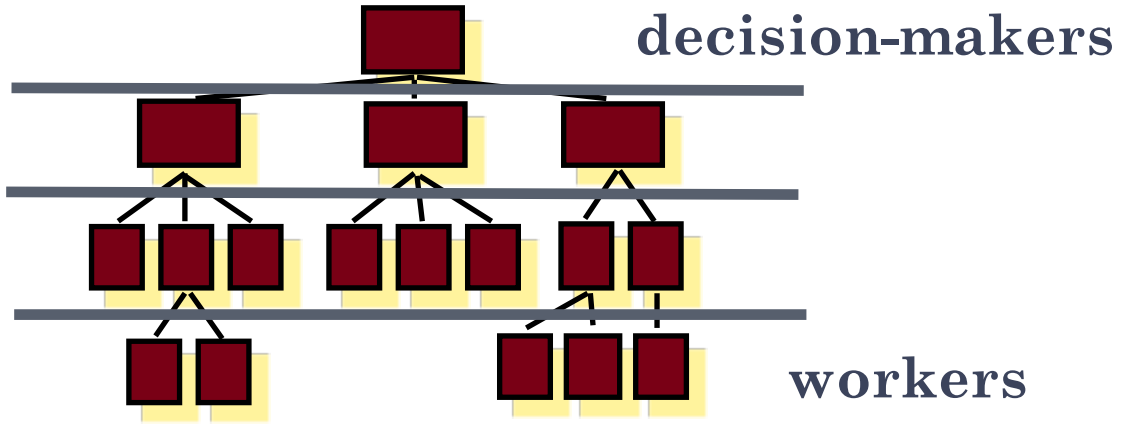
HORIZONTAL PARTITIONING

- define separate branches of the module hierarchy for each major function
- use control modules to coordinate communication between functions



Vertical Partitioning: Factoring

- design so that decision making and work are stratified
- decision making modules should reside at the top of the architecture

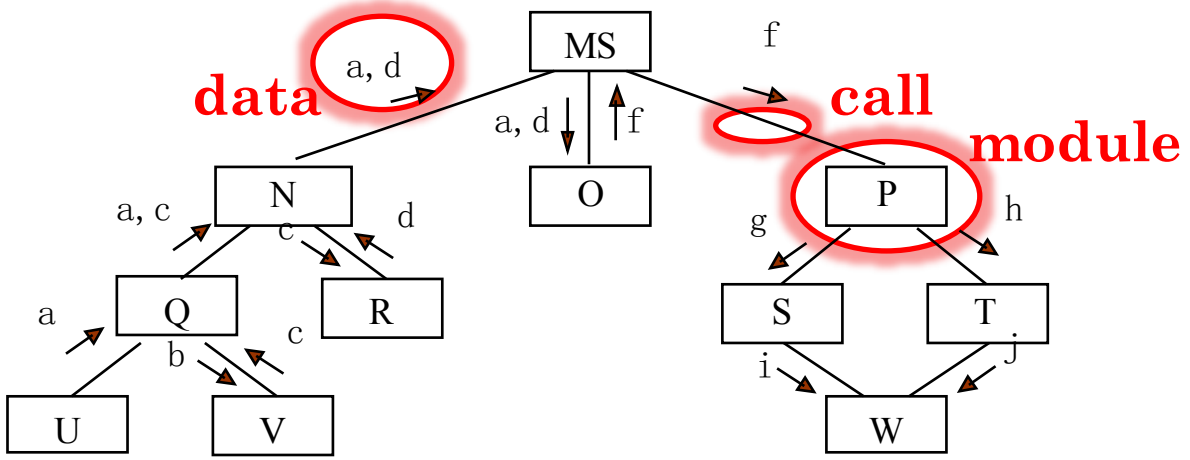


Why Partitioned Architecture?

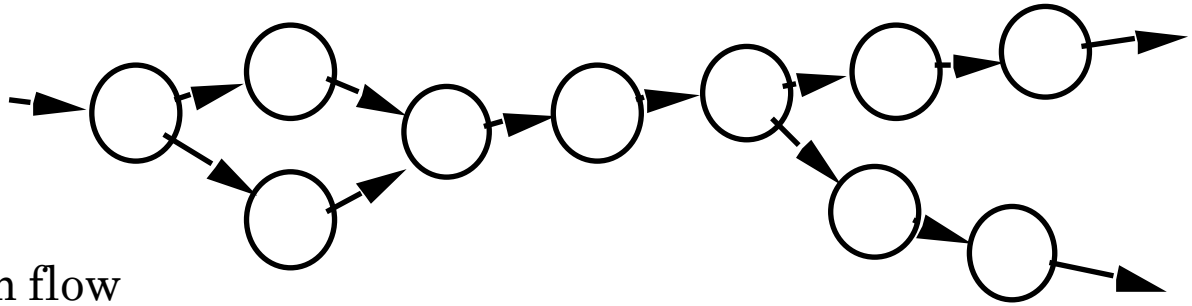
- results in software that is easier to test
- leads to software that is easier to maintain
- results in propagation of fewer side effects
- results in software that is easier to extend

Structured Design

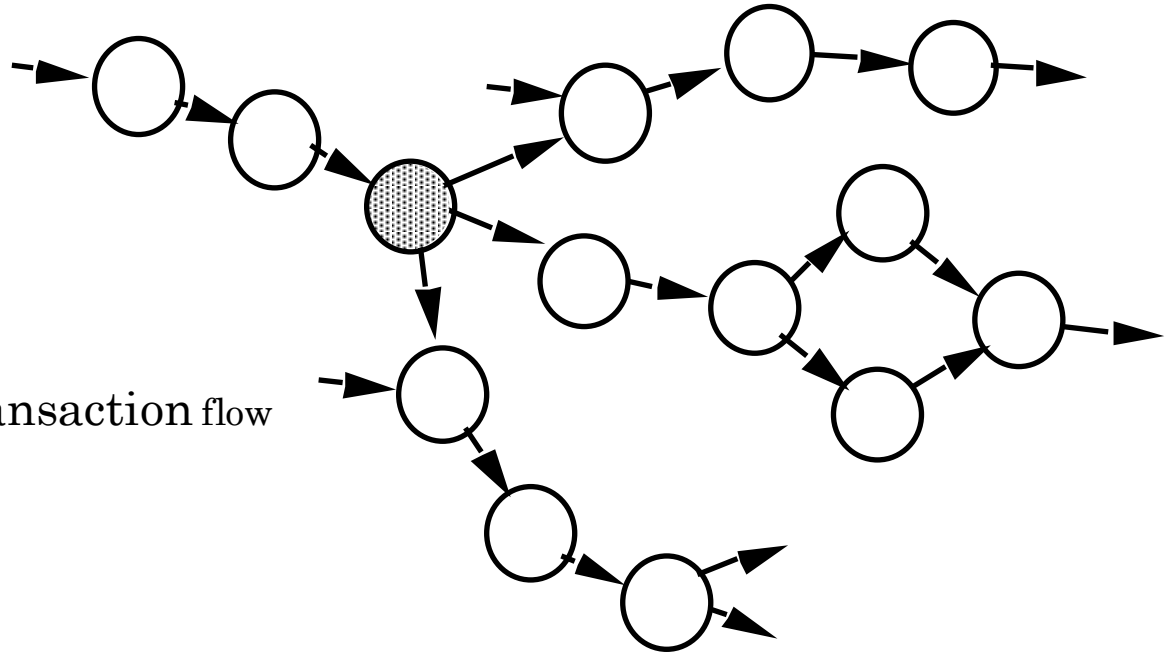
- Objective: to map the analysis model to *call and return* architecture
- Approach:
  - Data flow-oriented design method
  - DFD is mapped into a program architecture
- Notation: structure chart



Flow Characteristics

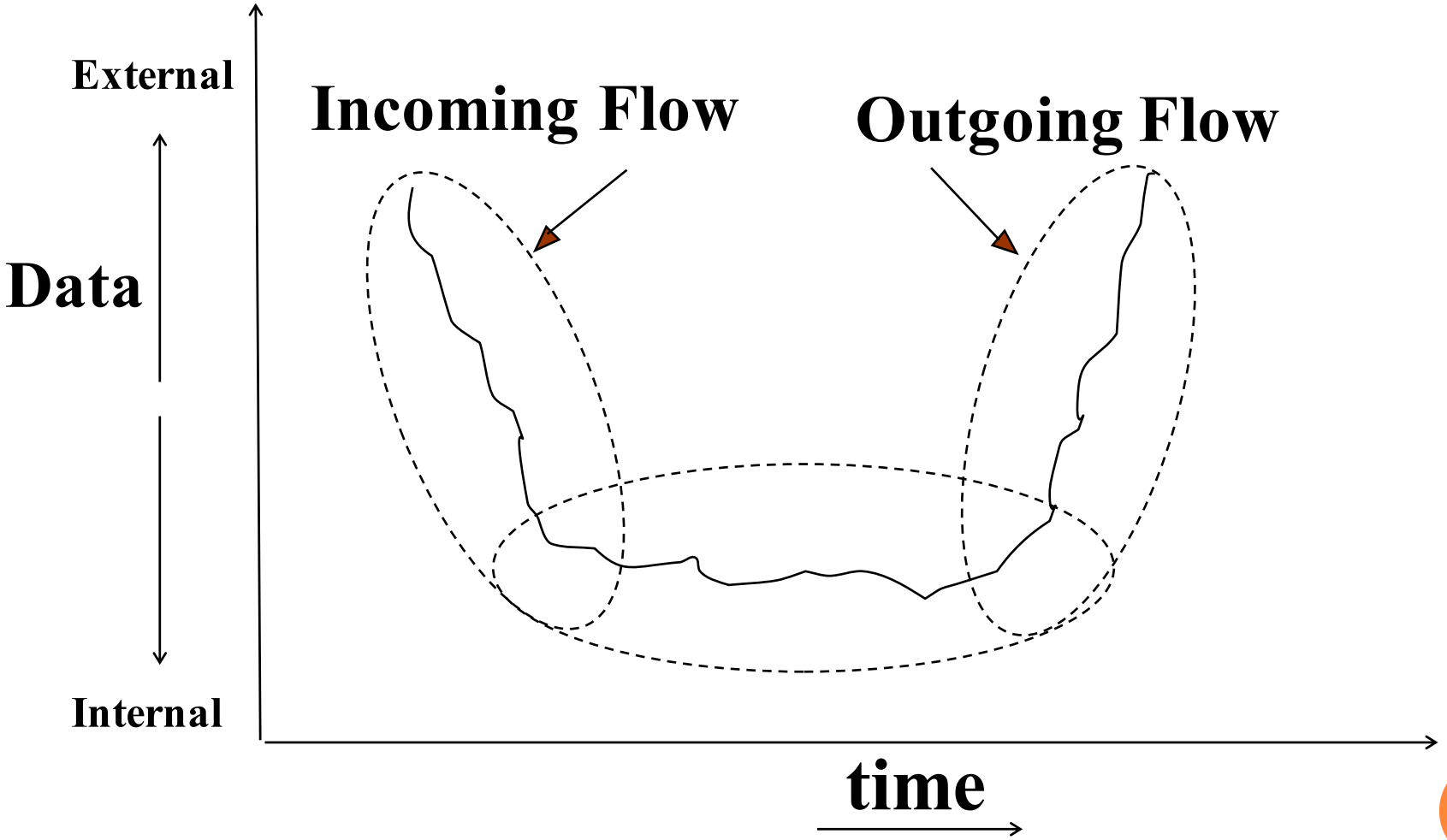


Transform flow

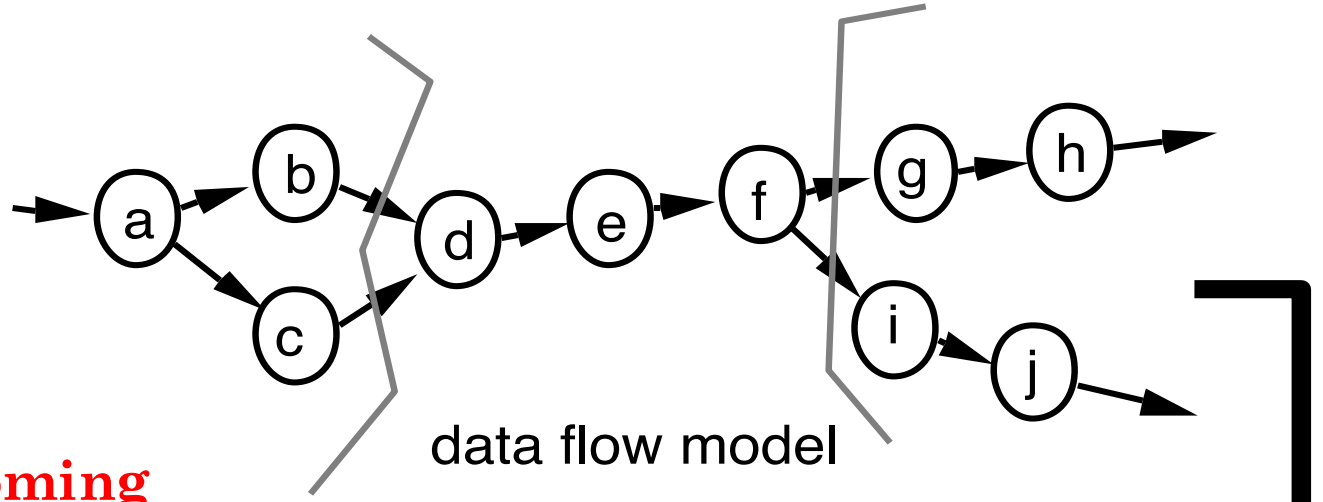


Transaction flow

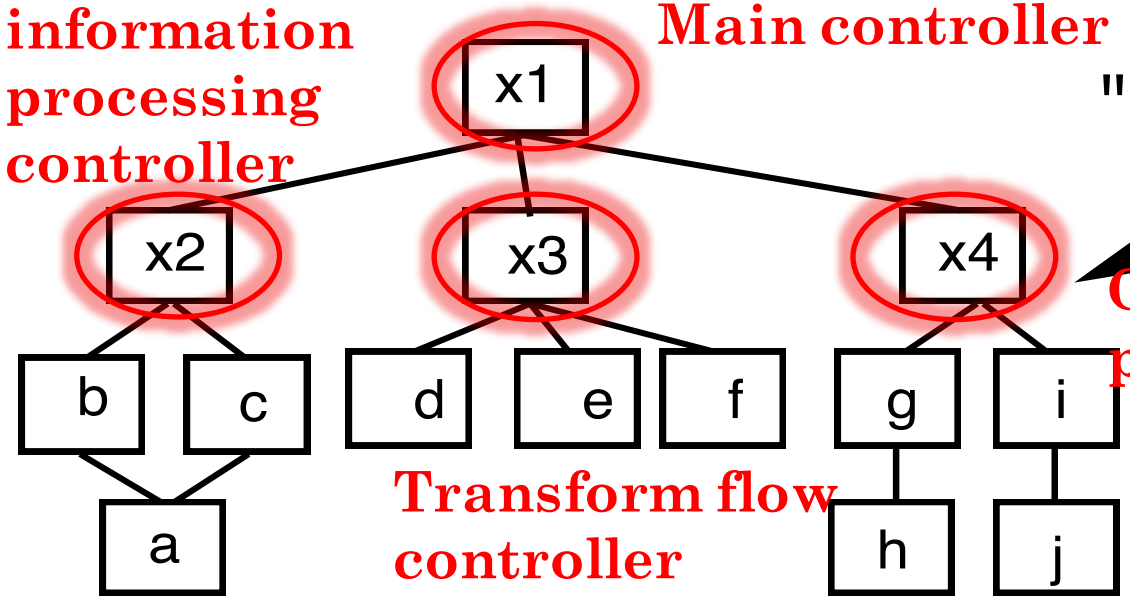
TRANSFORM FLOW



TRANSFORM MAPPING



"Transform" mapping

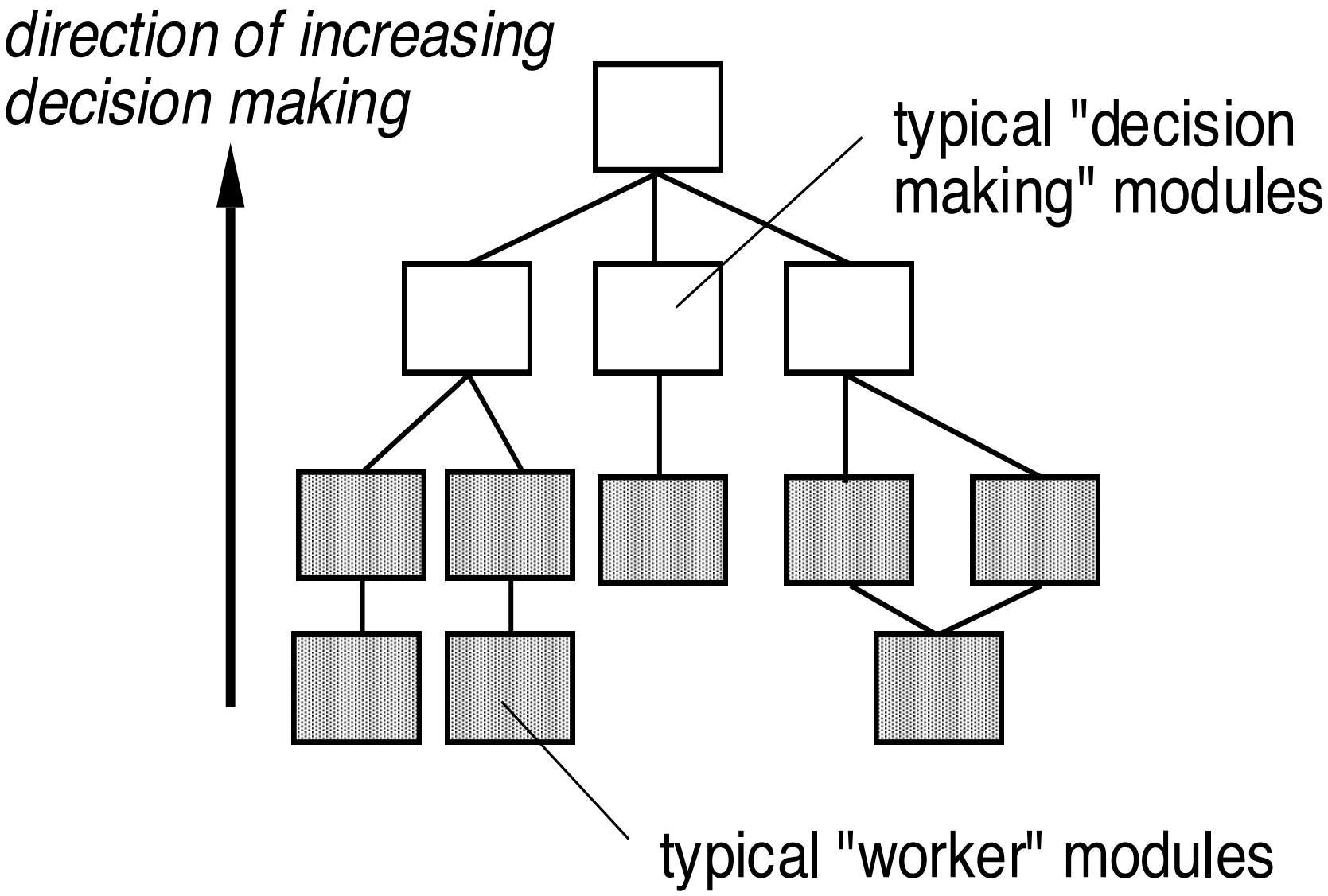


Outgoing information processing controller

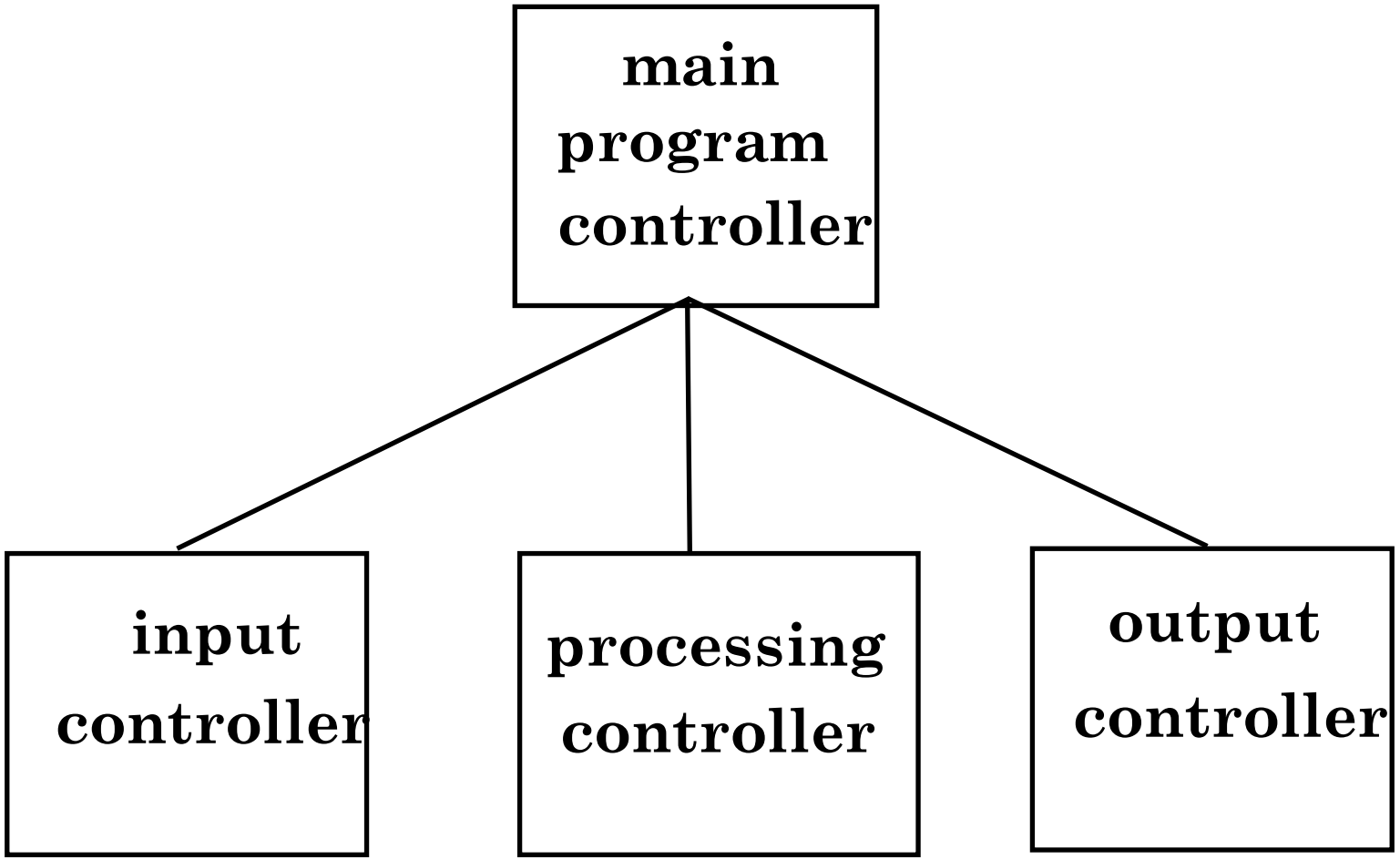
General Mapping Approach

- Isolate incoming and outgoing flow boundaries for ***transform flows***, Isolate the transform center
- Working from the boundary outward, map DFD ***transforms*** into corresponding modules
- Add *control modules* as required
- Refine the resultant program structure using effective modularity concepts

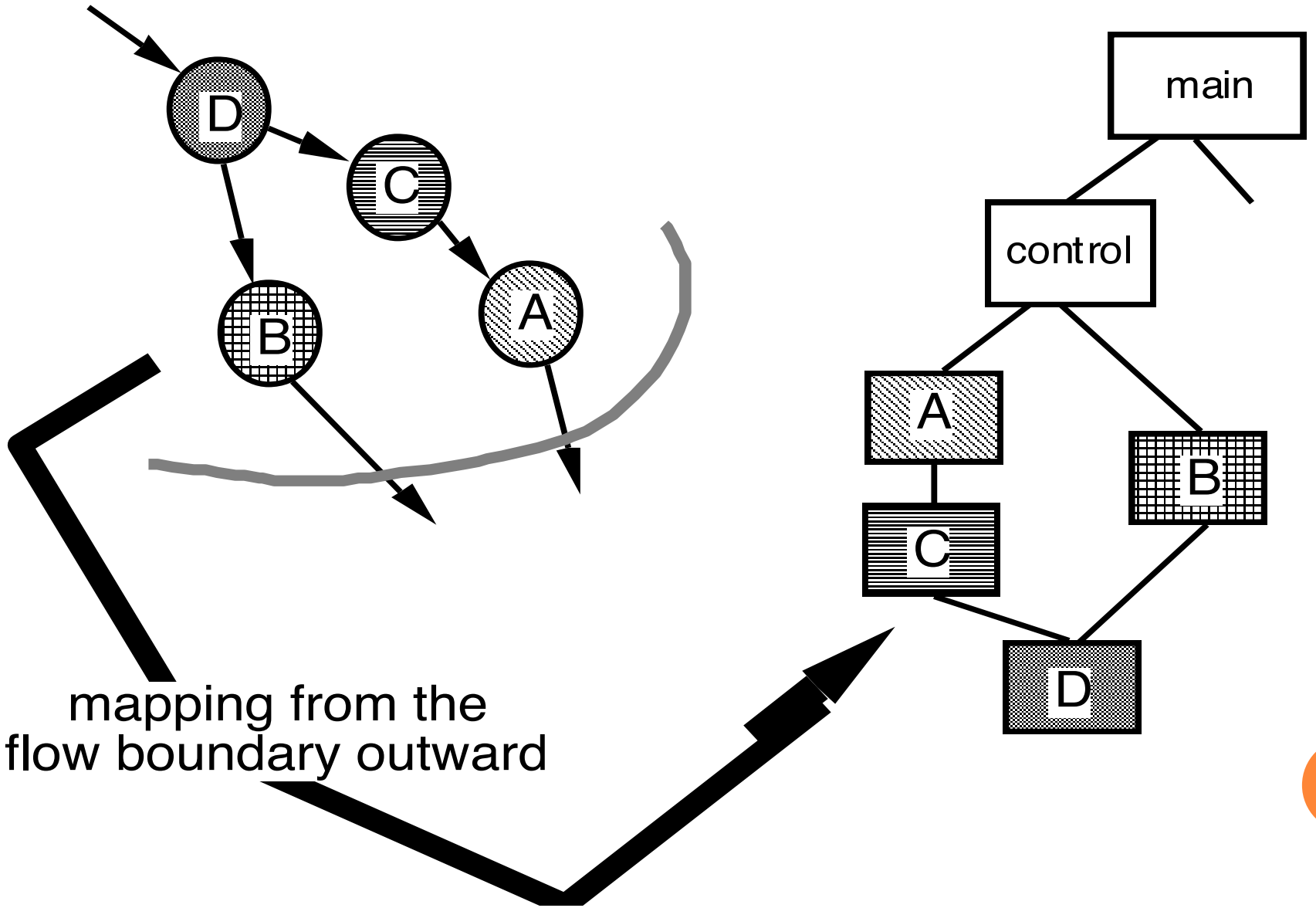
FACTORING


direction of increasing
decision making
typical "decision
making" modules
typical "worker" modules

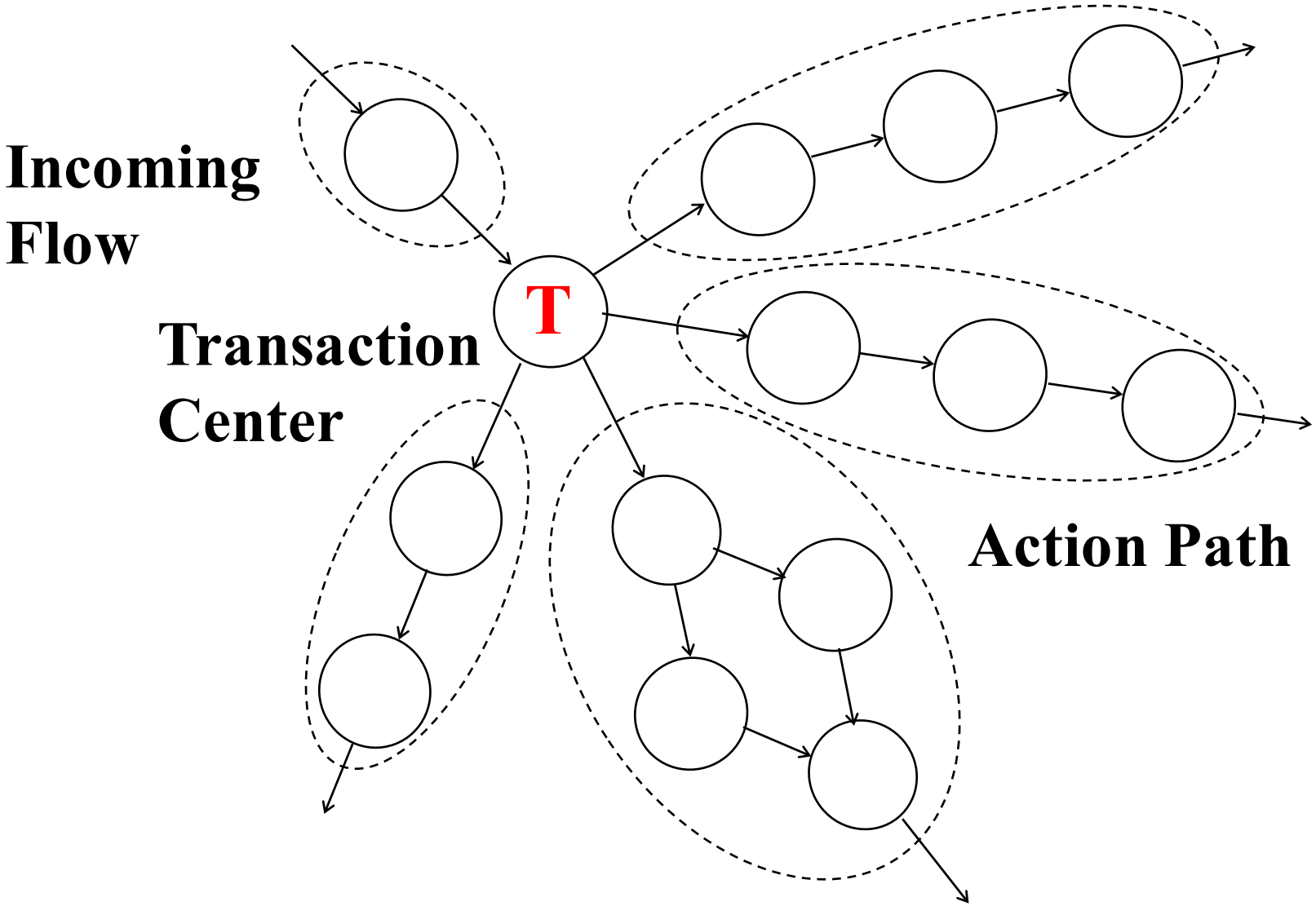
First Level Factoring



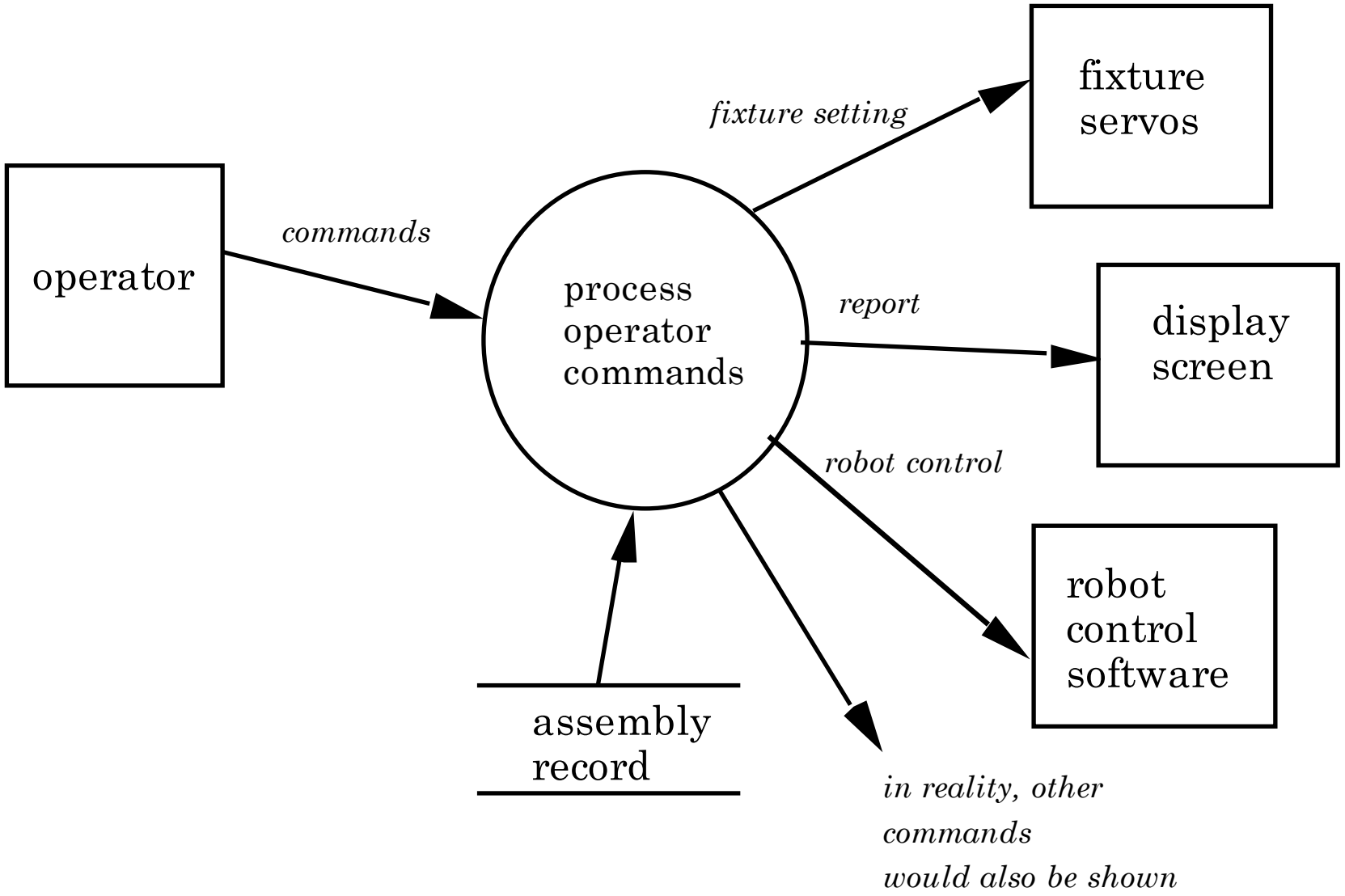
Second Level Factoring



TRANSACTION FLOW



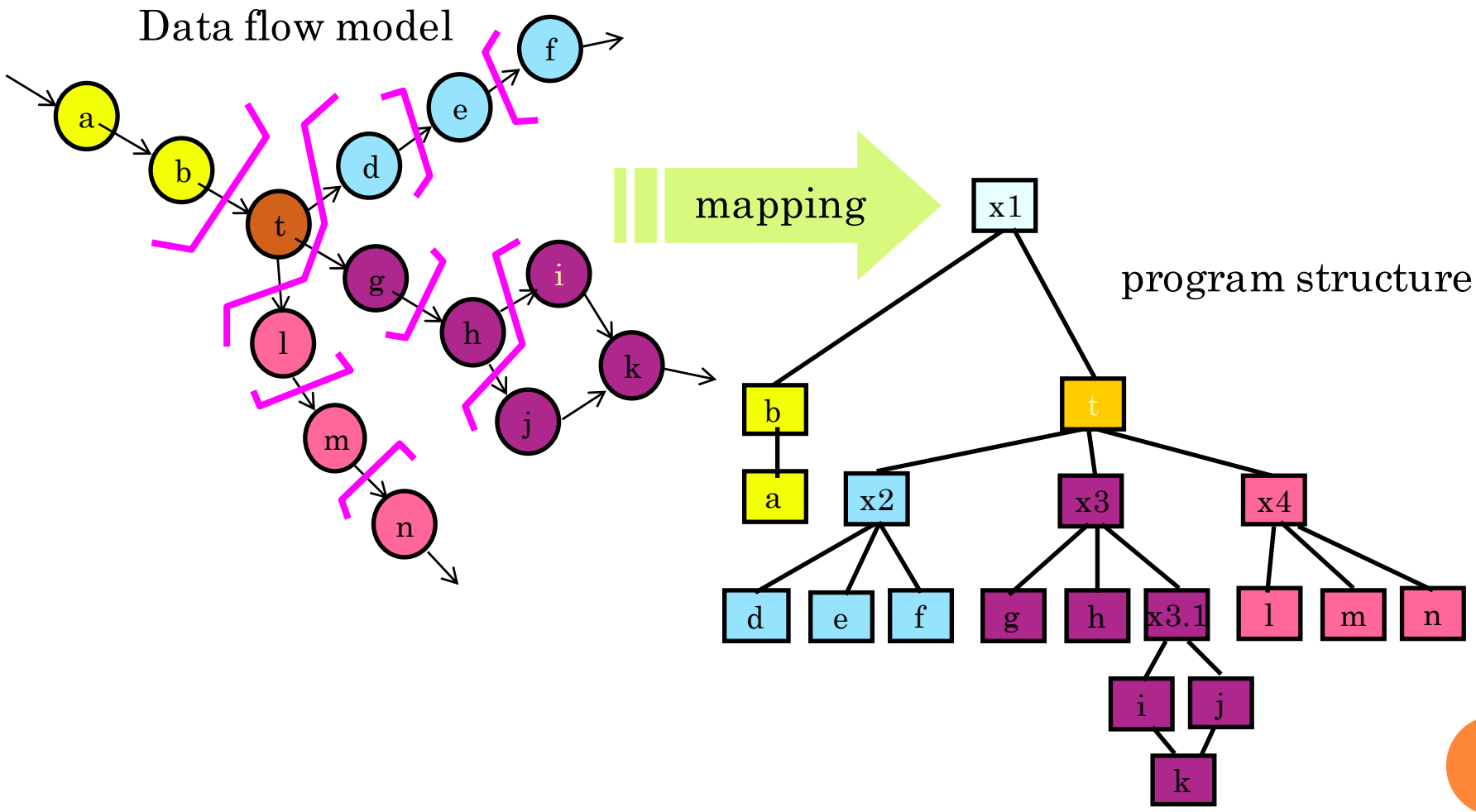
Transaction Flow Example



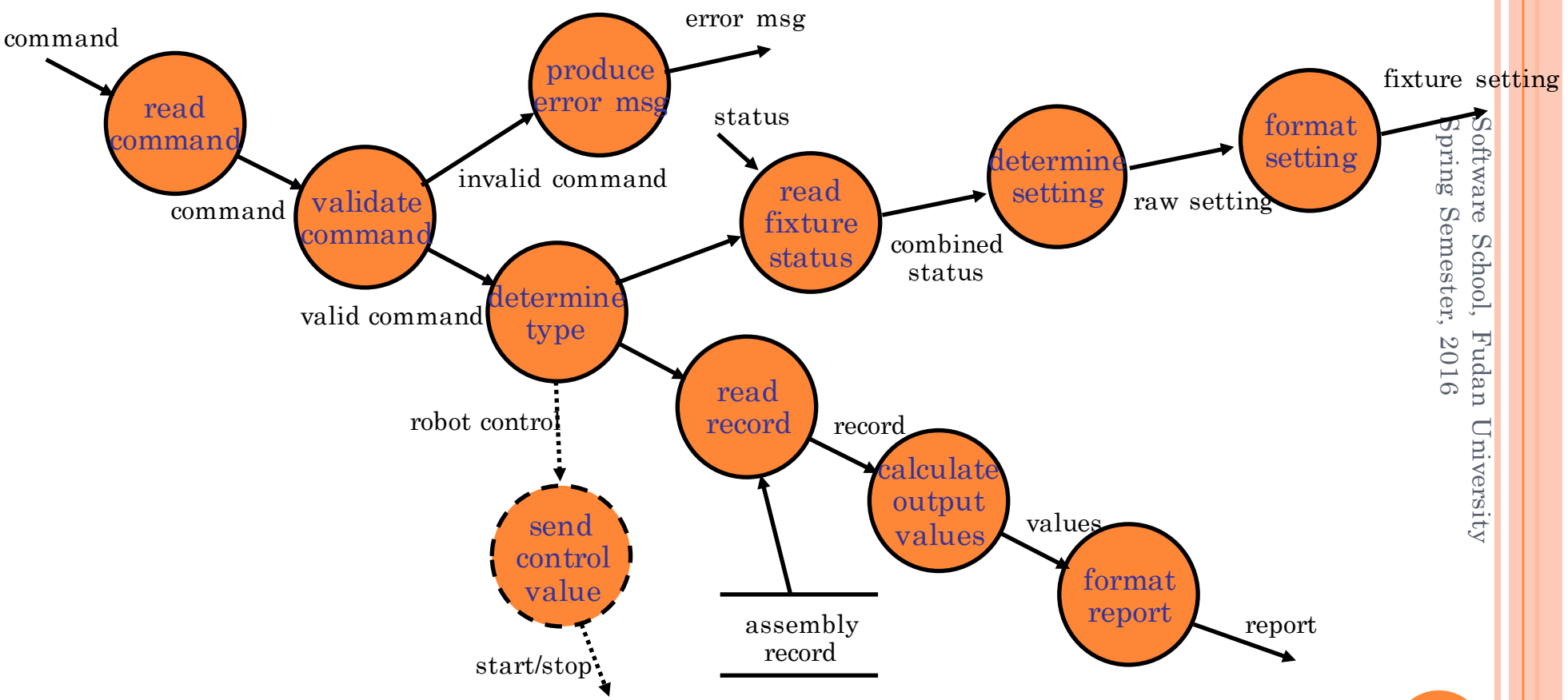
Transaction Mapping Principles

- Isolate the incoming flow path
- Define each of the action paths by looking for the "spokes of the wheel"
- Assess the flow on each action path
- Define the dispatch and control structure
- Map each action path flow individually

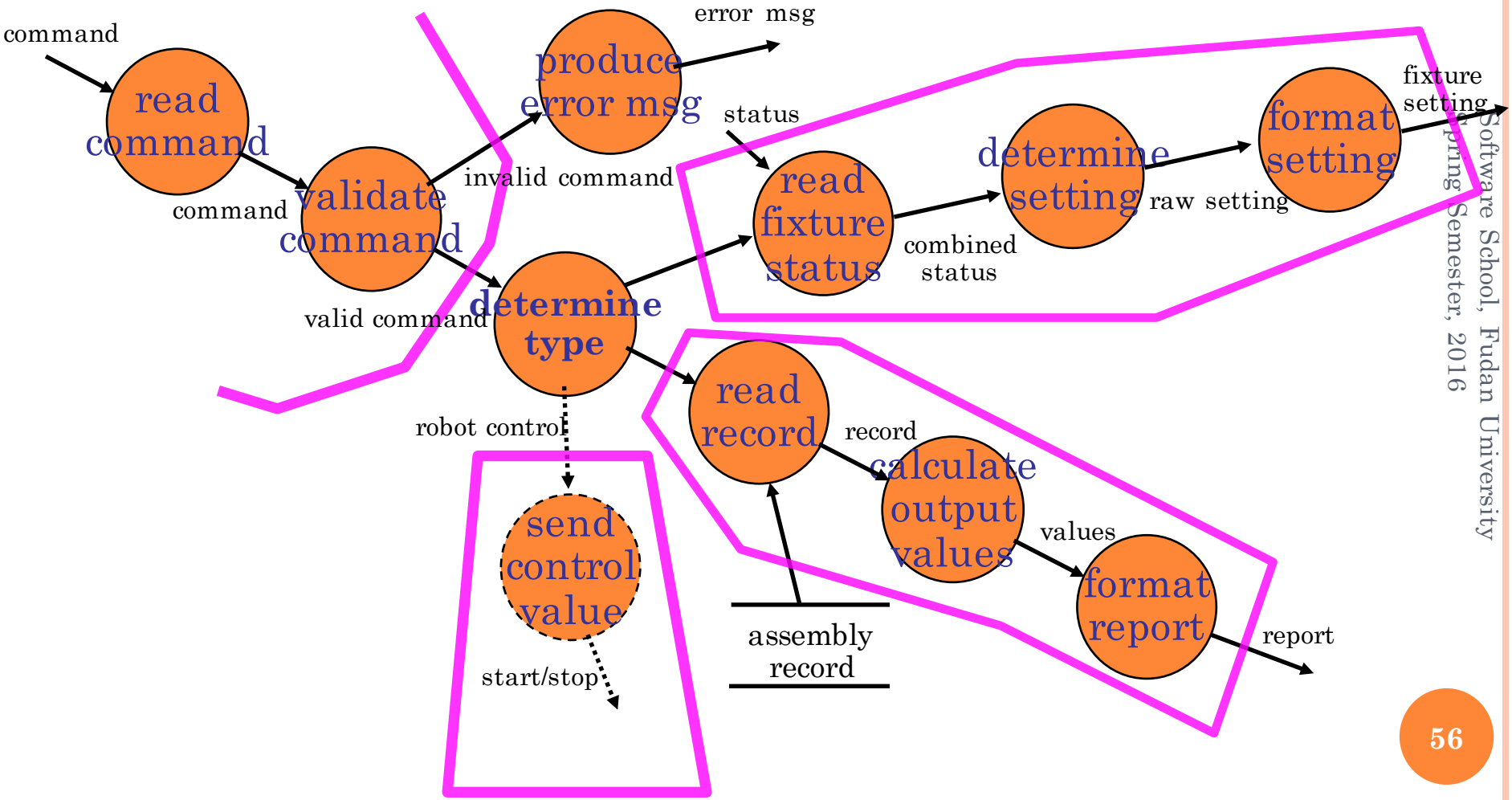
Transaction Mapping



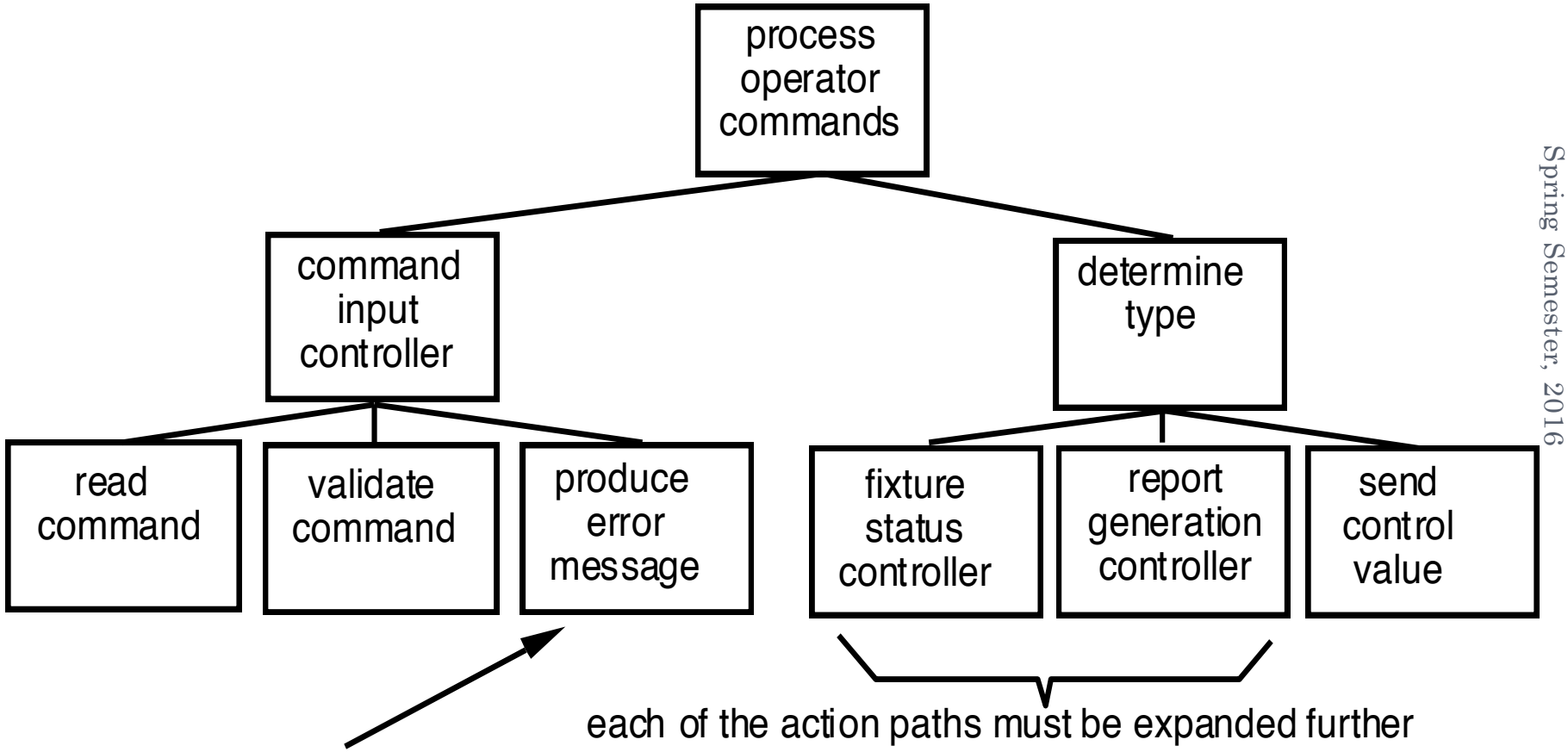
Data Flow Diagram (Transaction Flow)



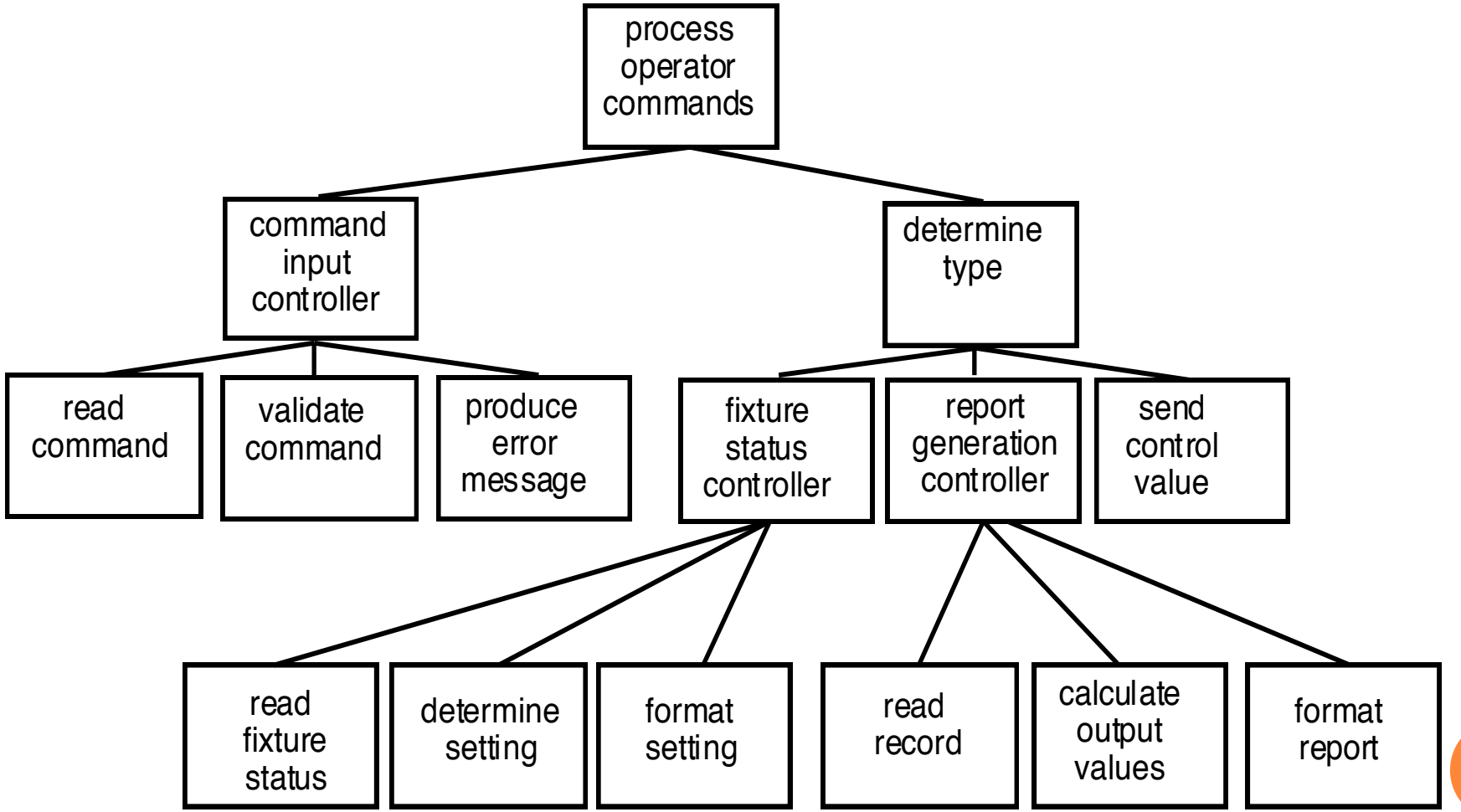
Isolate Flow Paths



Map the Flow Model



Refining the Structure Chart



End of Chapter 9