Structured Design (Yourdon)

Agenda

1. Architectural Design (a brief overview)
   1.1 Introduction
   1.2 Data design
   1.3 Architectural styles
2. Structured Design (Yourdon)
   Transform mapping technique
   Transaction mapping technique

Architectural Design

- It represents the structure of data and program components (modules) in the system.
- It considers the architectural style that the system will take, the structure and properties of the components (modules) and interrelationships that occur among all architectural components.
- Architectural design begins with data design and then proceeds to the derivation of one or more representations of the architectural structure of the system.

Architectural Design (cont.)

- Alternative architectural styles (or patterns) may be analyzed in an attempt to derive the structure that is best suited to customer requirements and quality attributes. Even though an architectural pattern conveys an image of a system, it is not an architecture as such. An architectural pattern is rather a concept that captures essential elements of a software architecture.
- Furthermore, patterns are often defined as something "strictly described and commonly available". For example, layered architecture is a call-and-return style, when it defines an overall style to interact. When it is strictly described and commonly available, it is a pattern.
- Once the alternative has been selected, the architecture is elaborated using an architectural (high level) design method (e.g. Structured Design – Yourdon)

Data Design

- The data design activity translates data object (entities) defined as part of analysis model (see lectures 3 and 4, etc) into data structures at the software component level, and when necessary, a database architecture at the application level.

- In some situations, a database must be designed and built specifically for a new system. In others, one or more existing databases are used.

Architectural Style

An architectural style defines "a family of systems in terms of a pattern of structural organization, a vocabulary of components and connectors, with constraints on how they can be combined"

- M. Shaw and D. Garlan

Different Architectural Style

Each style describes a system category that encompasses:

(1) a set of components (e.g., a database, computational modules) that perform a function required by a system,

(2) a set of connectors that enable "communication, coordination and cooperation" among components,

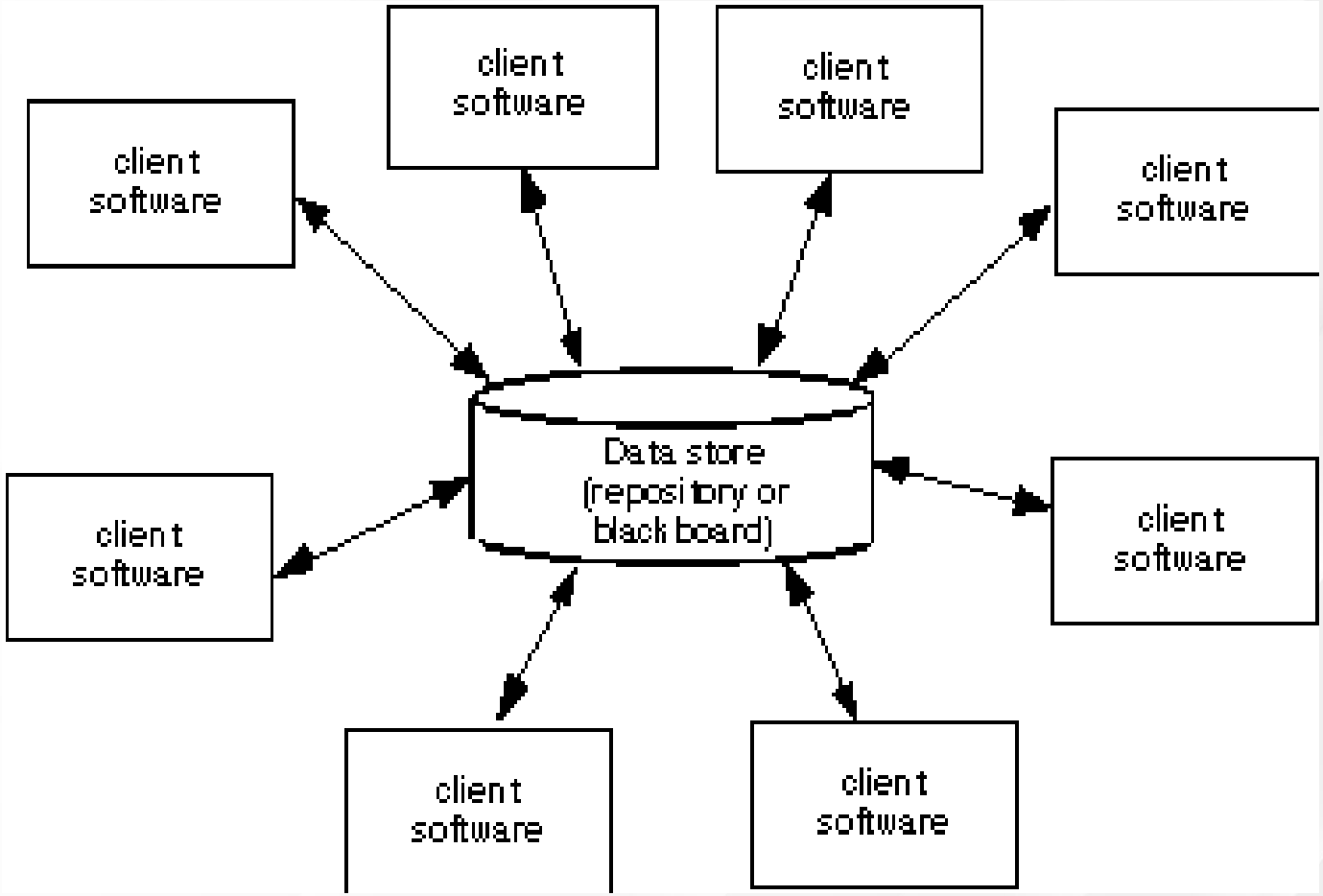
(3) constraints that define how components can be integrated to form the system, and

(4) semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

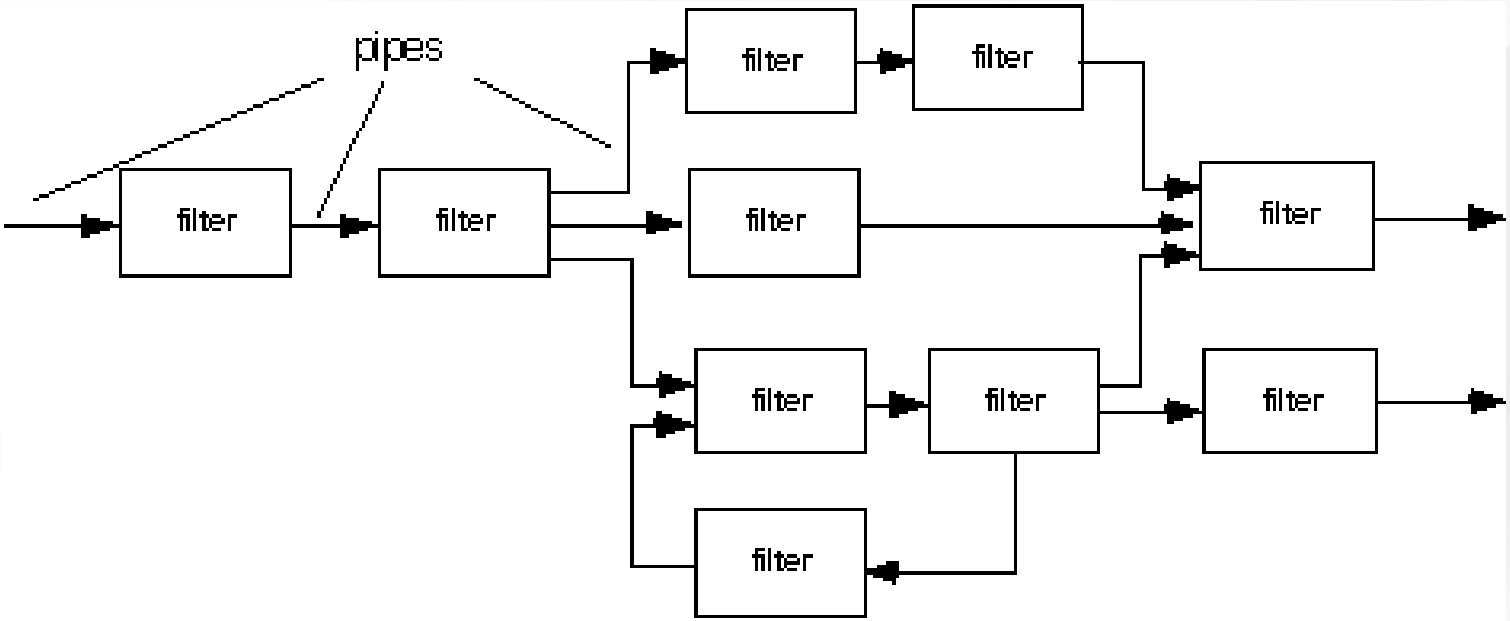
Different Architectural Style

- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Layered architectures
- Client Server architecture

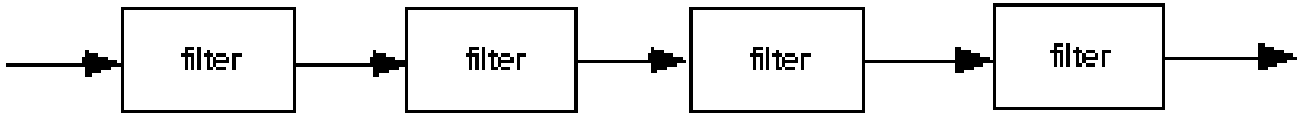
Data-Centered Architecture



Data Flow Architecture



(a) pipes and filters

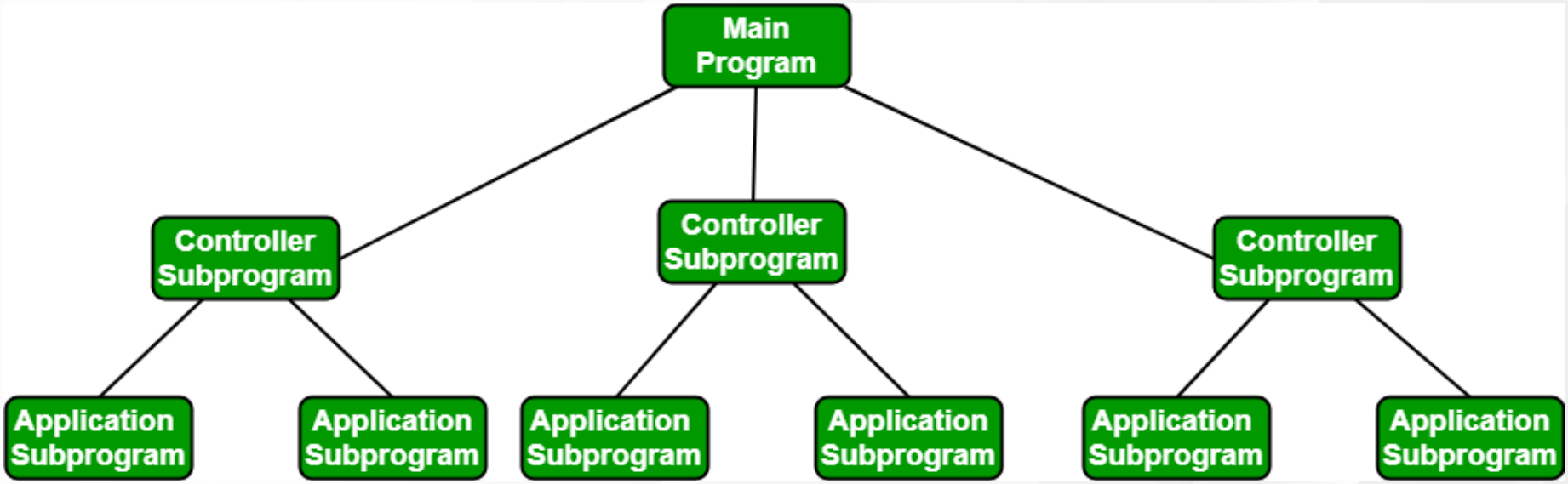


(b) batch sequential

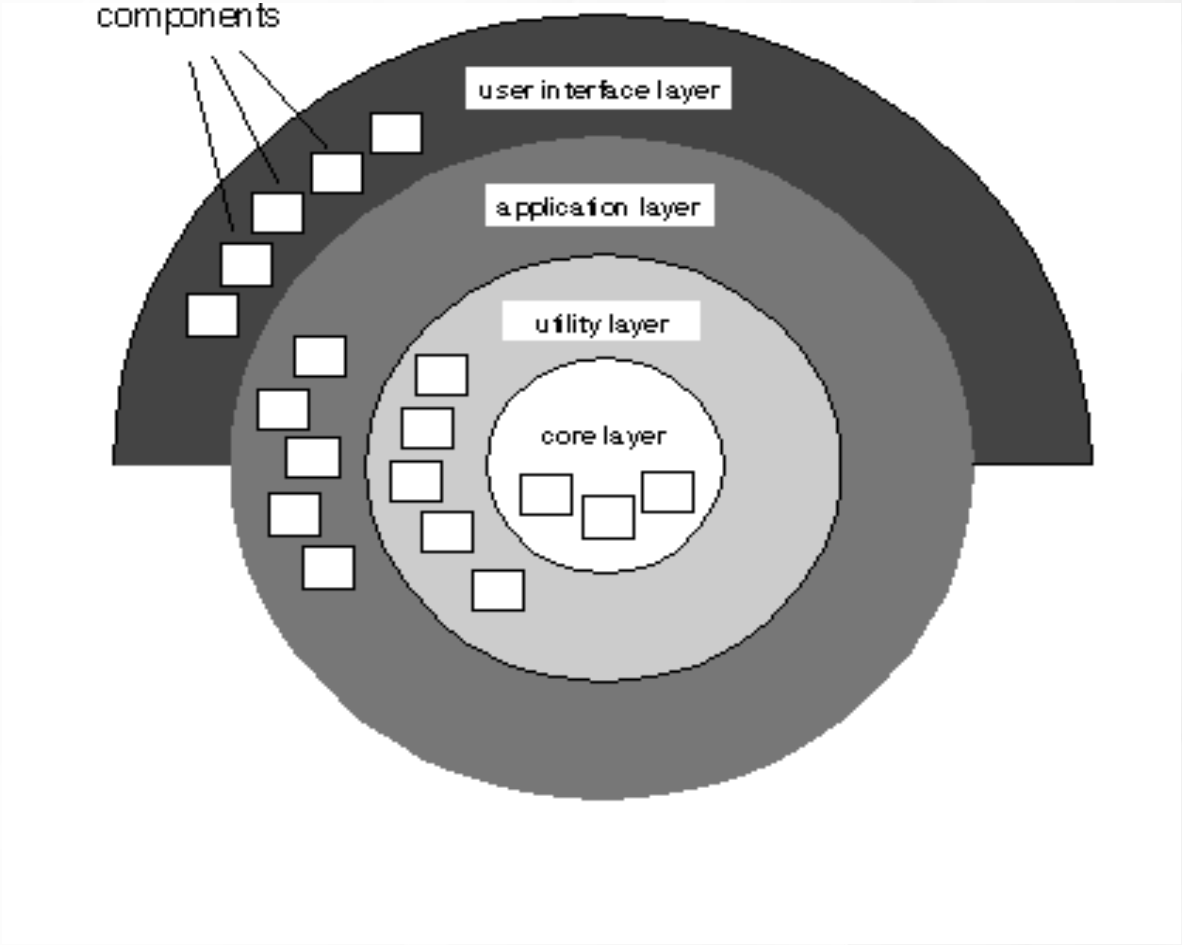
Pipe And Filter

- Break the system into independently executing components (filters)
- Connect the components together via FIFO queues (pipes)
- Each filter takes a single input and produces a single output

Call and Return Architecture

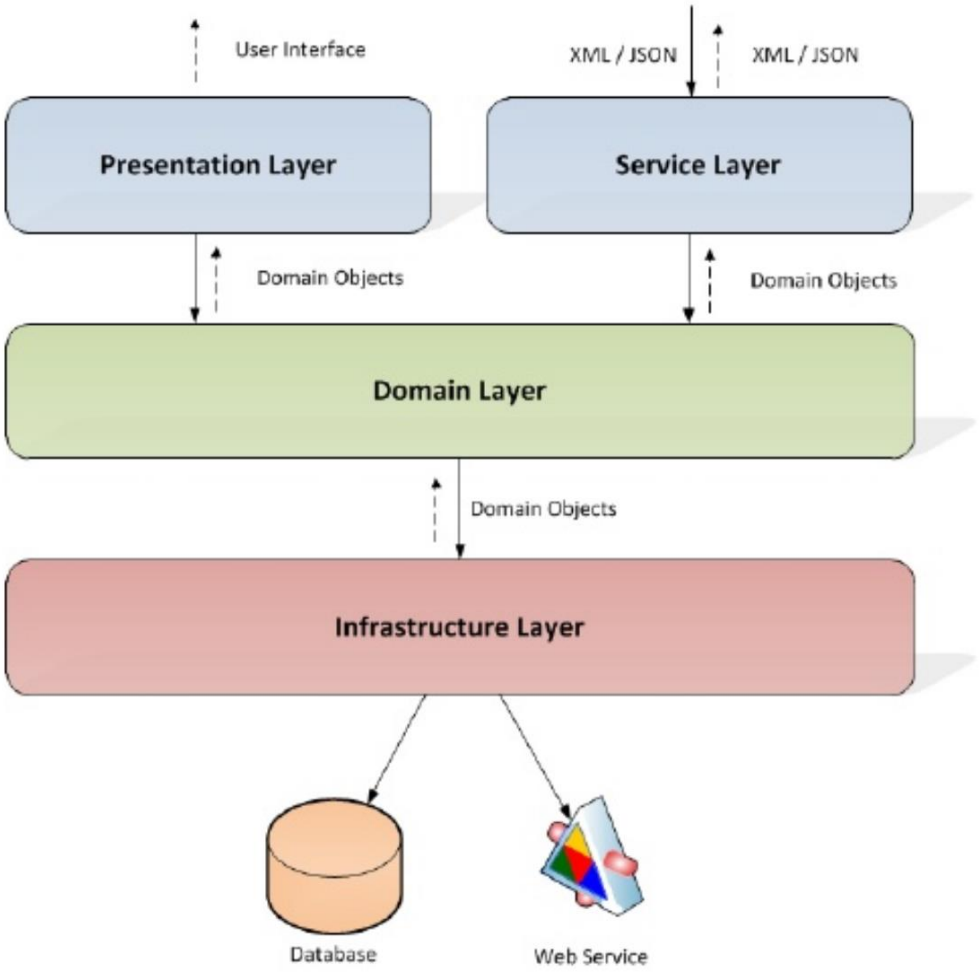


Layered Architecture



- Presentation
- Business
- database

3-Layer Architecture



Presentation Layer

- Also Know as Frontend Layer, User Interface (UI)
- Responsible for creating and displaying the user interface and handling user interaction.
- Data shown is fetched from the Domain Layer

Service Layer

- Also Know as Web Service Layer
- Responsible for exposing a web service API and returning the method result as XML or JSON.
- Data returned is retrieved from the Domain Layer

Domain Layer

- Also Know as Business Layer
- Responsible for all the business logic in the application
- Consists of a Domain Model and Domain Services

Infrastructure Layer

- Also Know as Data Access Layer, Repository Layer
- Responsible for querying a database, calling a web service, sending e-mail etc.

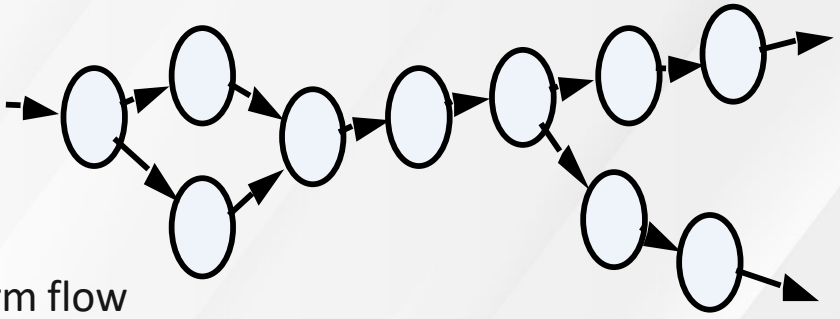
Why Partitioned Architecture?

- 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

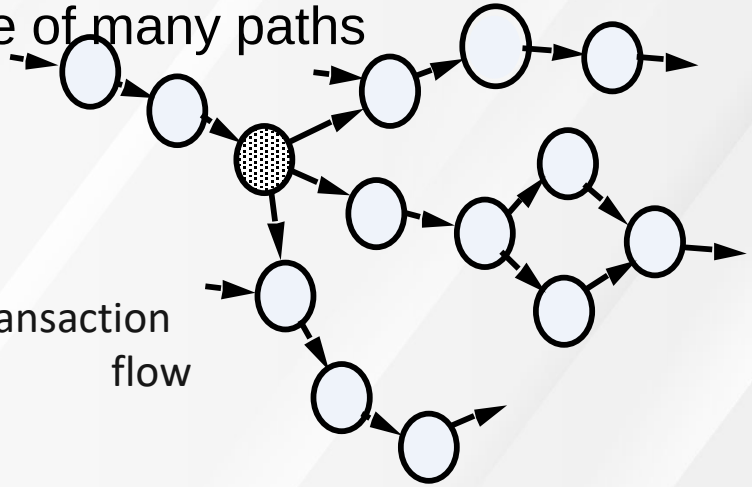
- Objective: to derive a program architecture that is partitioned
- approach:
  - the DFD is mapped into a program architecture
  - the Pspec and STD are used to indicate the content of each module
- Notation: structure chart

Flow Characteristics



Transform flow

- **transform flow** - overall data flow is sequential and flows along a small number of straight line paths
- **transaction flow** - a single data item triggers information flow along one of many paths

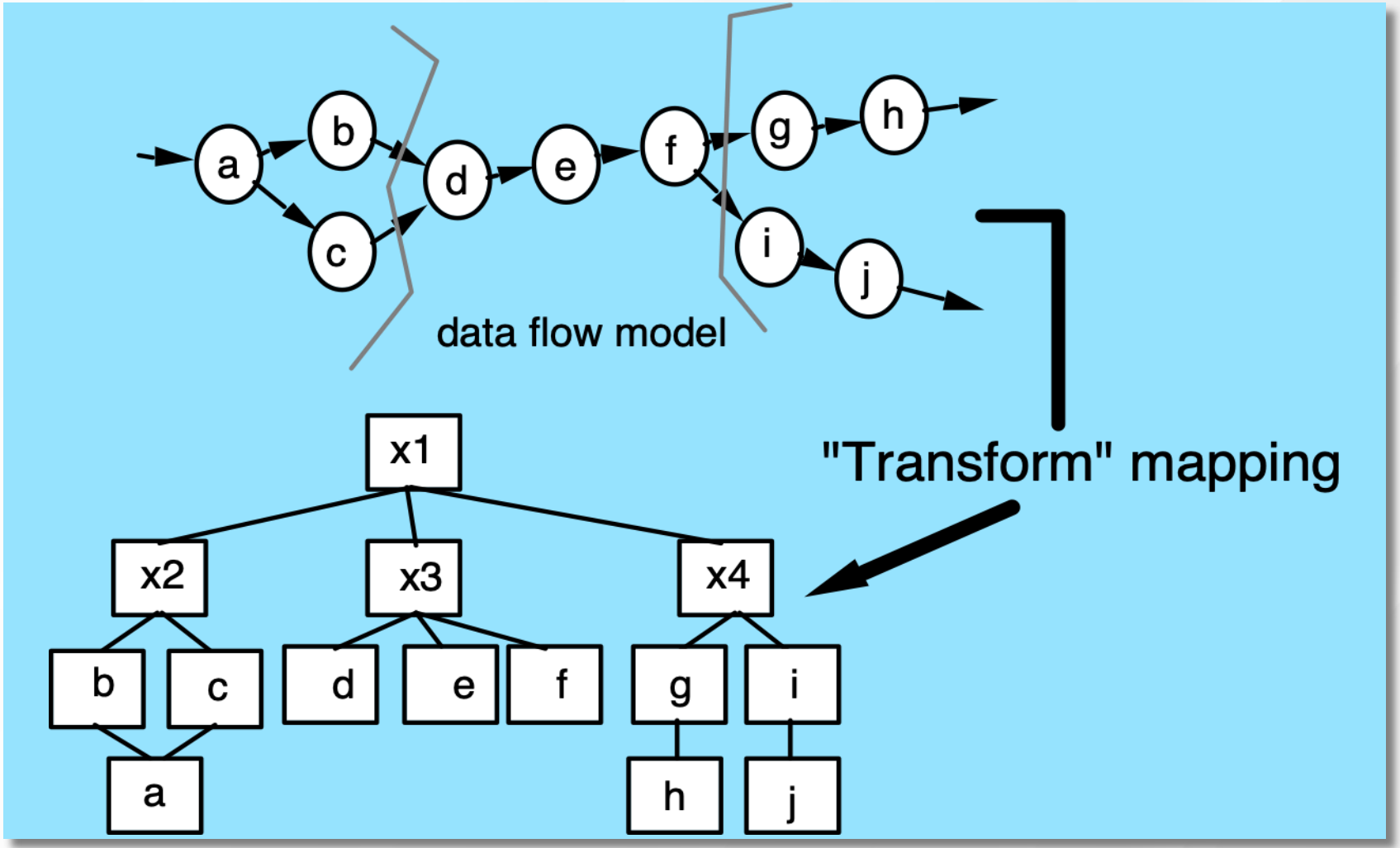


Transaction flow

General Mapping Approach

- isolate incoming and outgoing flow boundaries; for transaction flows, isolate the transaction 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

Transform Mapping



Steps on Transform Mapping

1. Review the fundamental system model
2. Review and refine DFD for the software
3. Determine whether the DFD has transform or transaction flow characteristics
4. Isolate the transform center by specifying incoming and outgoing flow boundaries
5. Perform first level factoring
6. Perform second level factoring
7. Refine the first iteration architecture using design heuristics for improved s/w quality

Steps on Transform Mapping

1. Review the fundamental system model

The fundamental system model or context diagram depicts the security function as a single transformation, Representing the external producers and consumers of data that flow into and out of the function.

Steps on Transform Mapping

- 2. Review and refine DFD for the software

Information obtained from the requirements model is refined to produce greater detail.

Steps on Transform Mapping

- 3. Determine whether the DFD has transform or transaction flow characteristics

Evaluating the DFD, we see data entering the software along one incoming path and exiting along many outgoing paths. Therefore, an overall transform characteristic will be assumed for information flow.

Steps on Transform Mapping

- 4. Isolate the transform center by specifying incoming and outgoing flow boundaries

Incoming data flows along a path in which information is converted from external to internal form; outgoing flow converts internalized data to external form. Incoming and outgoing flow boundaries are open to interpretation. That is, different designers may select slightly different points in the flow as boundary locations.

Steps on Transform Mapping

- 5. Perform first level factoring

The program architecture derived using this mapping results in a top-down distribution of control. *Factoring leads to a* program structure in which top-level components perform decision making and Low-level components perform most input, computation, and output work. Middle-level Components perform some control and do moderate amounts of work.

Steps on Transform Mapping

- 6. Perform second level factoring

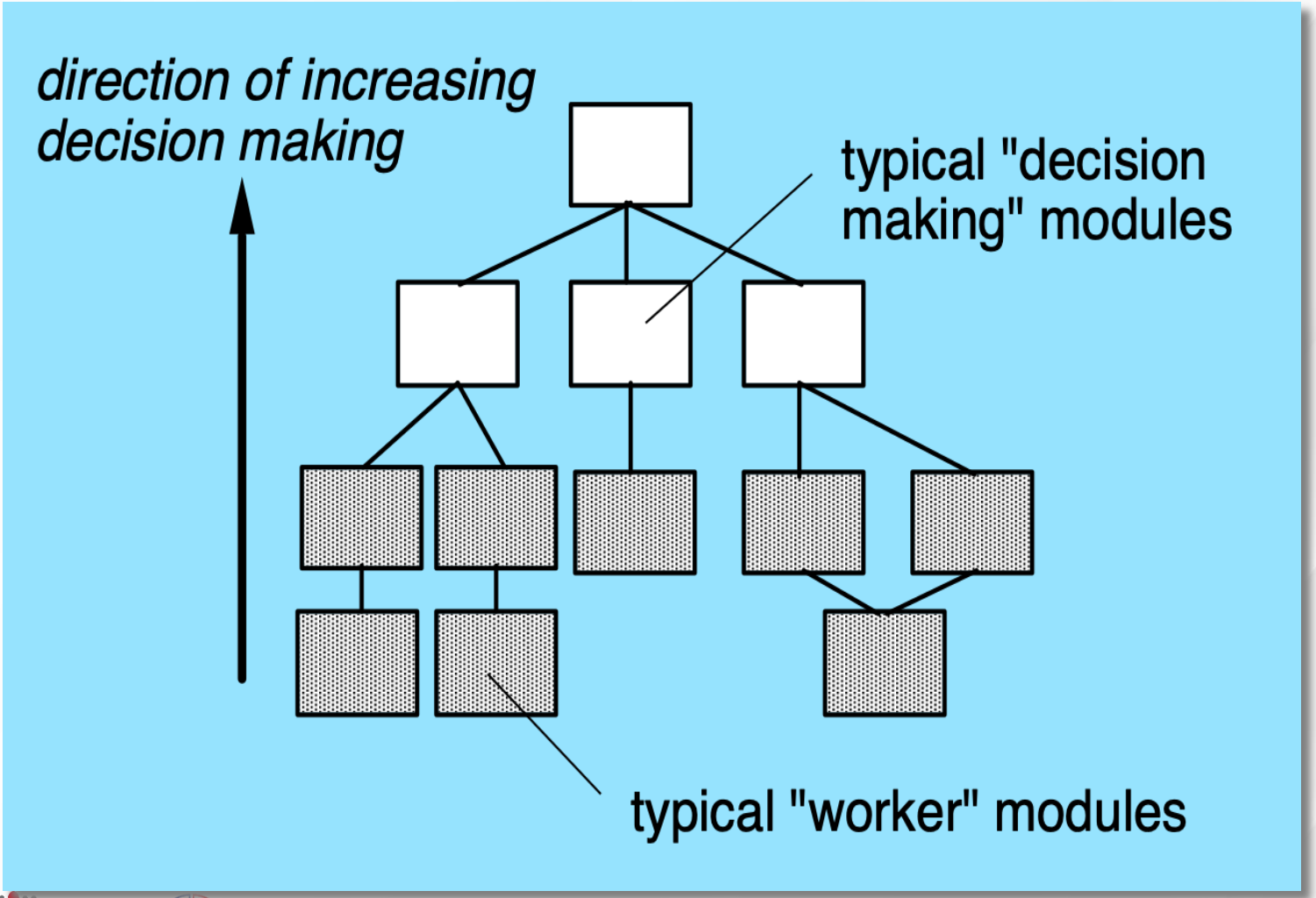
Second-level factoring is accomplished by mapping individual transforms (bubbles) of a DFD into appropriate modules within the architecture. Beginning at the transform center boundary and moving outward along incoming and then outgoing paths, transforms are mapped into subordinate levels of the software structure.

Steps on Transform Mapping

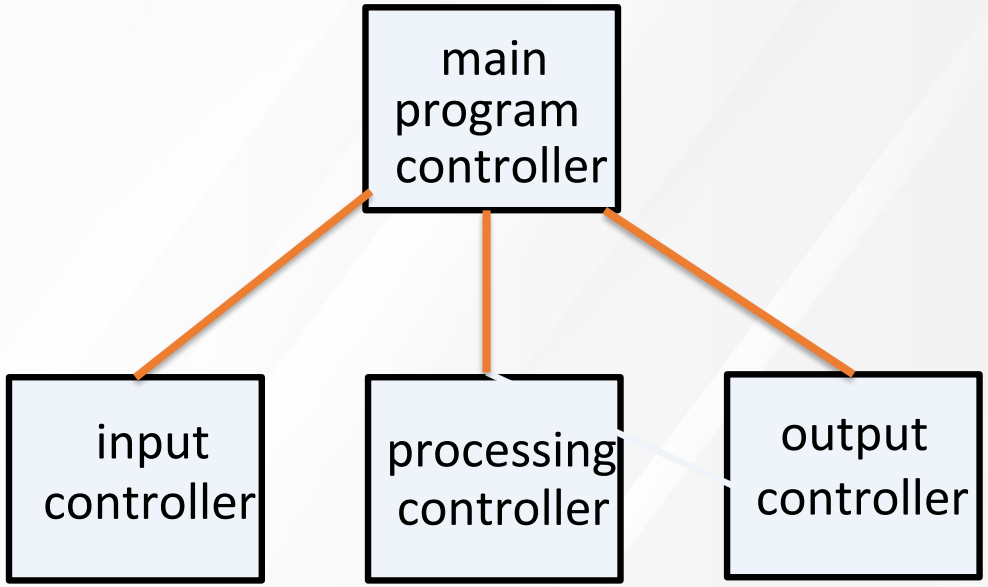
- 7. Refine the first iteration architecture using design heuristics for improved s/w quality

A first-iteration architecture can always be refined by applying concepts of functional independence. Components are exploded or imploded to produce sensible factoring, separation of concerns, good cohesion, minimal coupling, and most important, a structure that can be implemented without difficulty, tested without confusion, and maintained without grief.

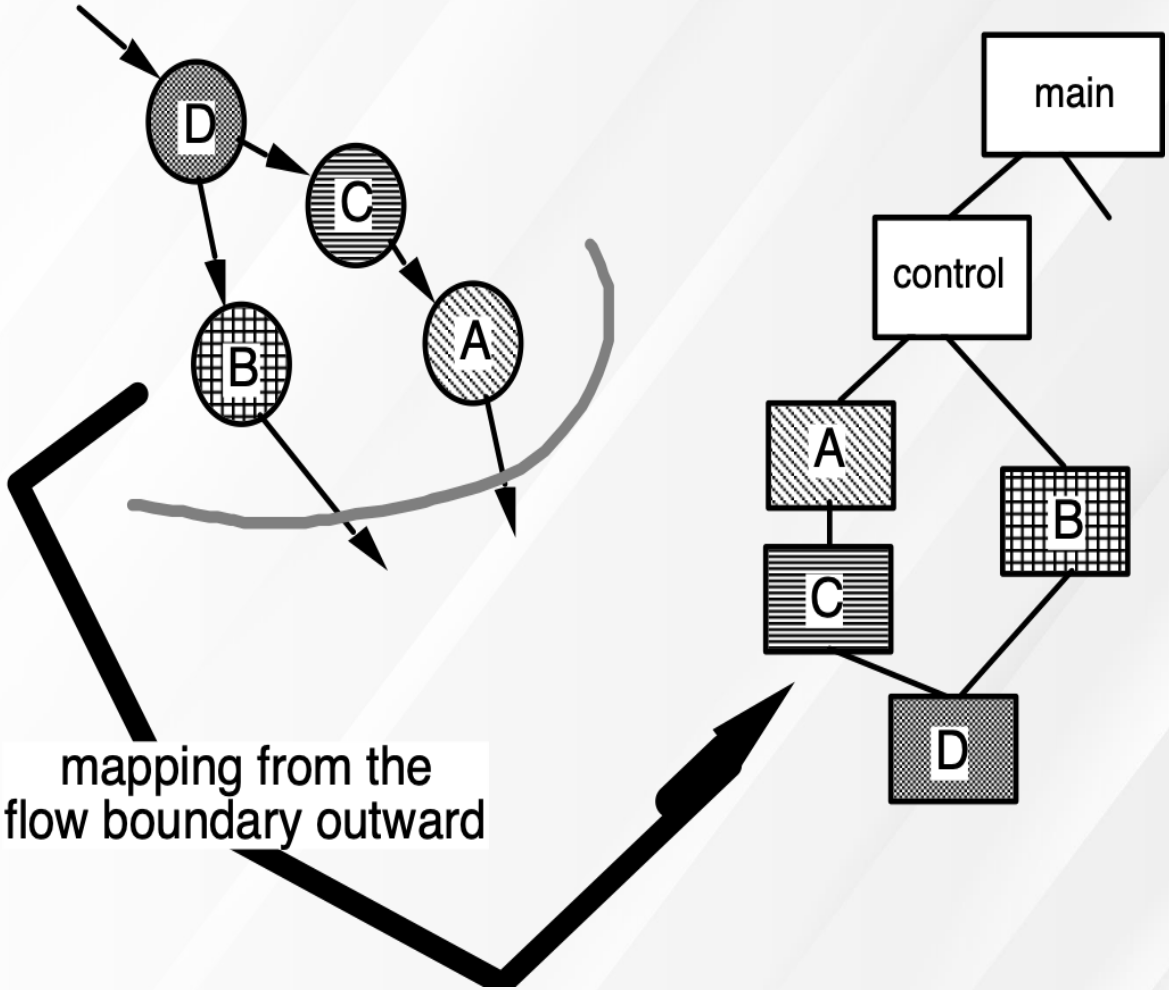
Factoring


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

First Level Factoring



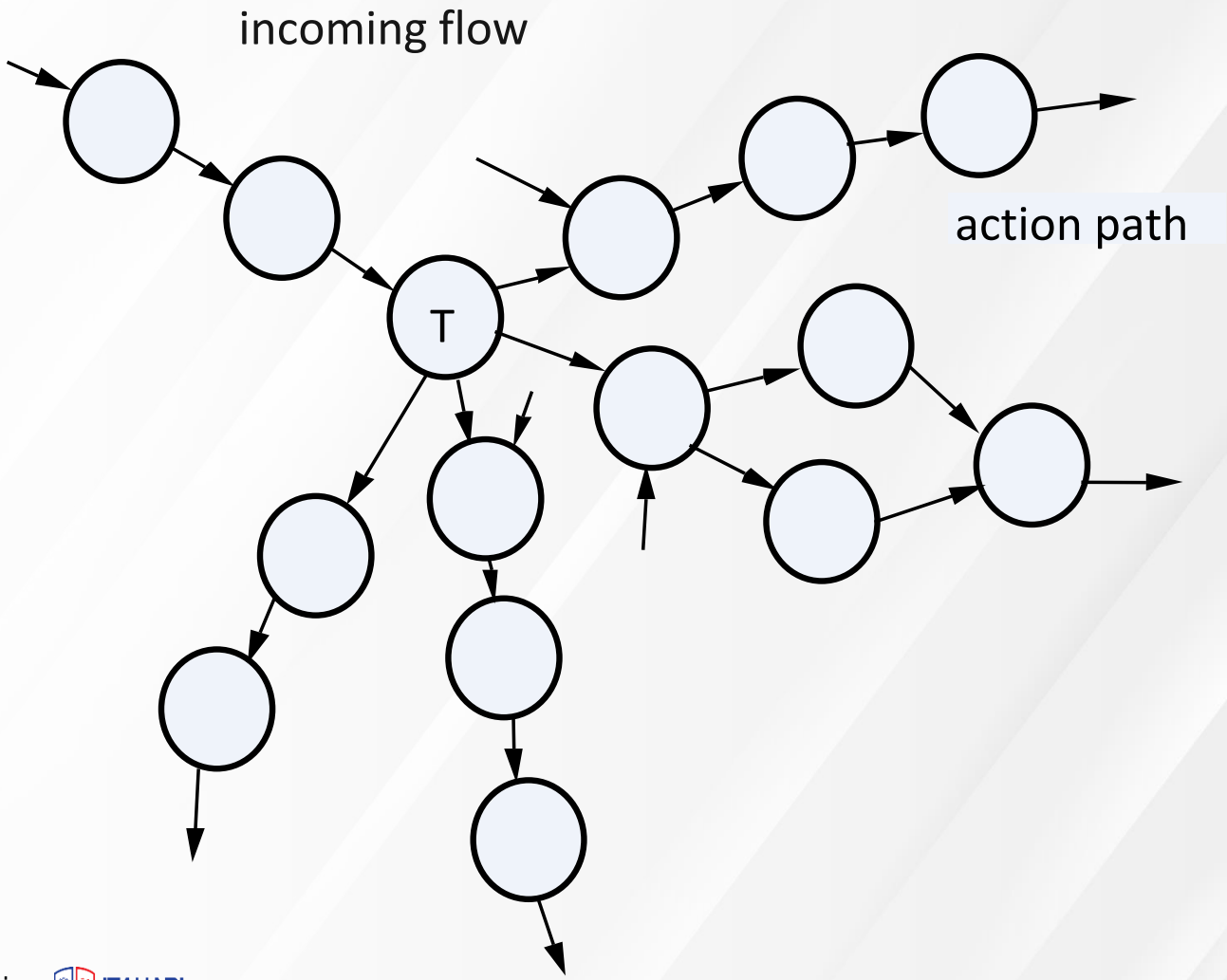
Second Level Factoring



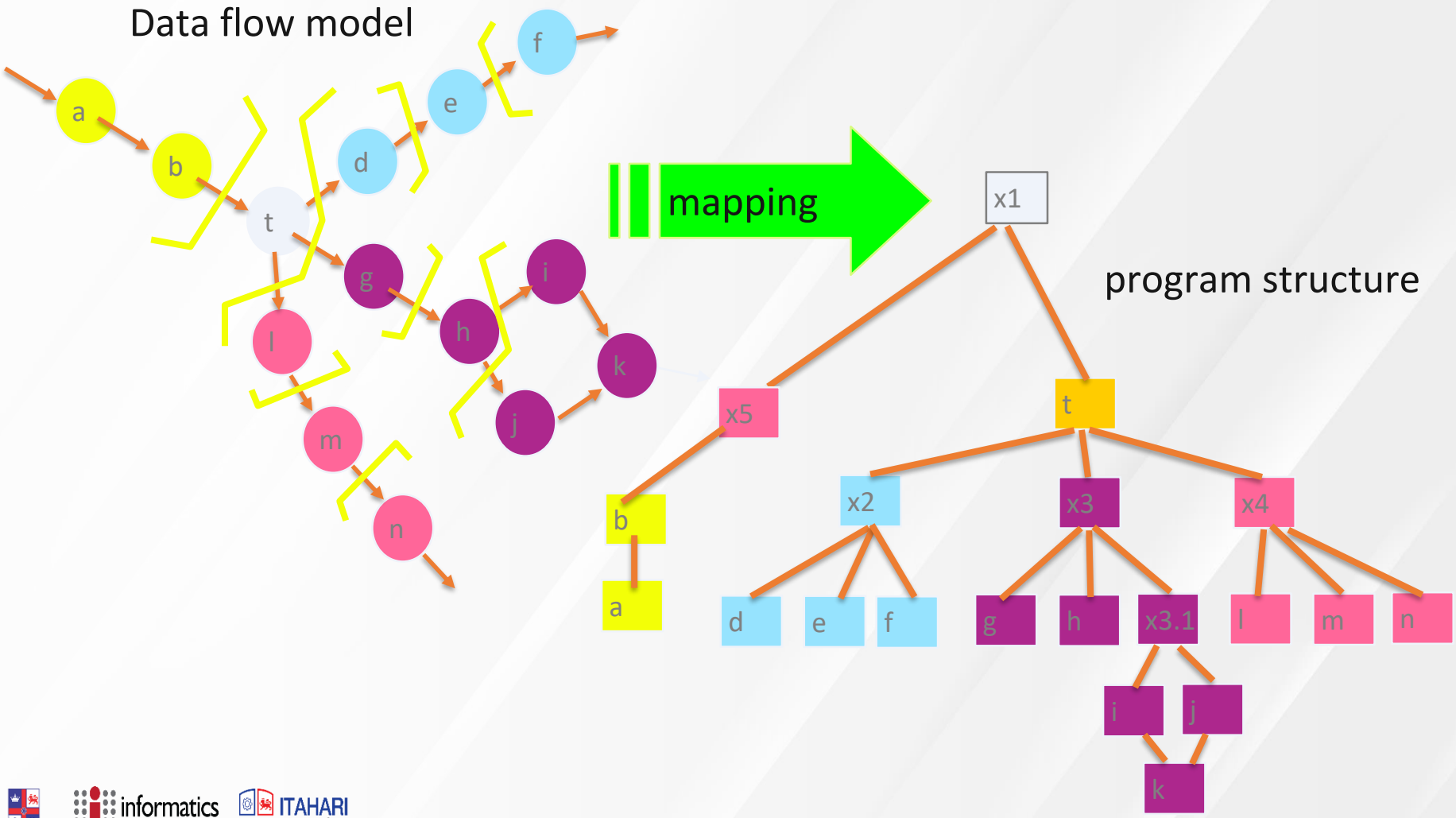
Transaction Mapping Principles

- 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

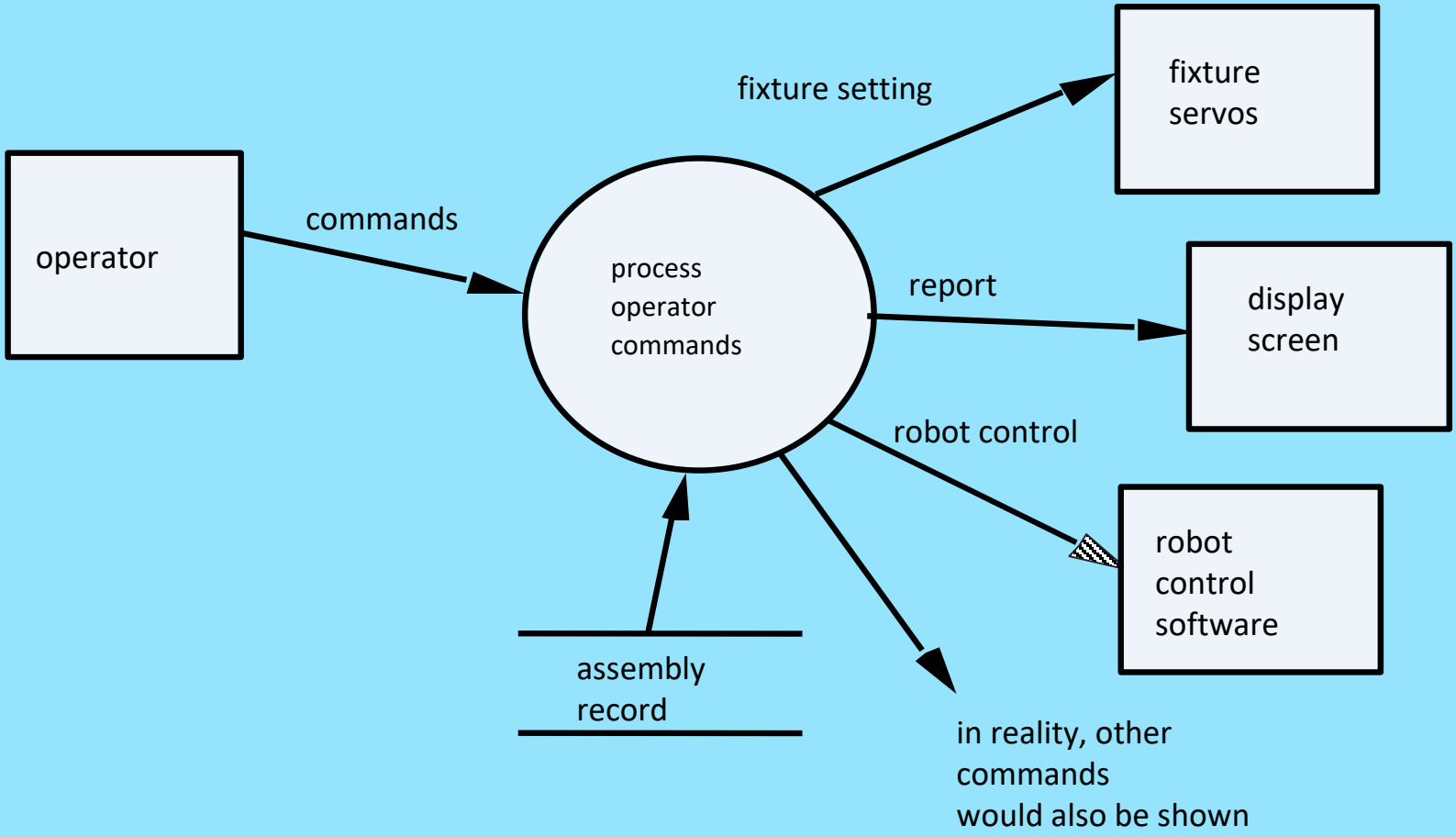
Transaction Flow



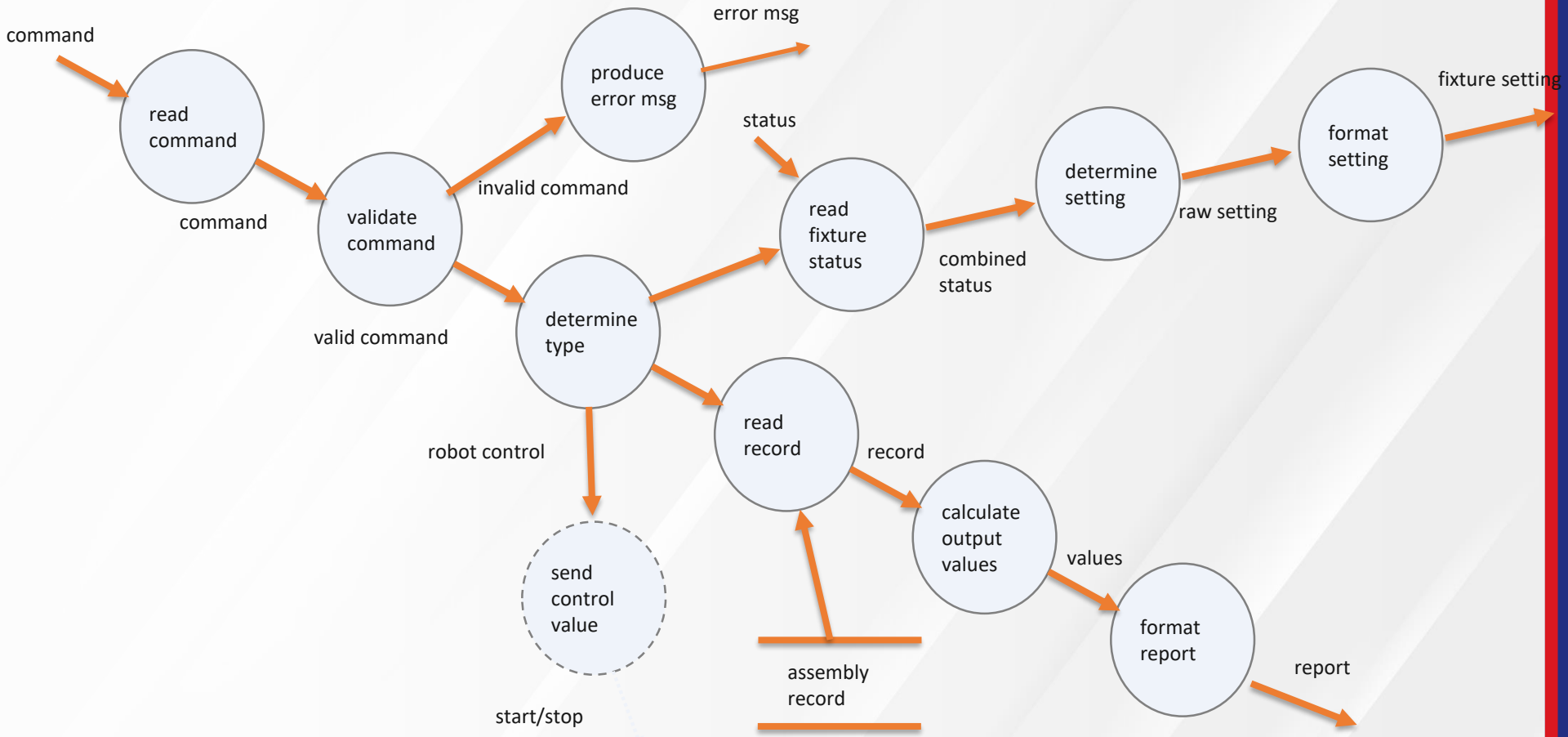
Transaction Mapping



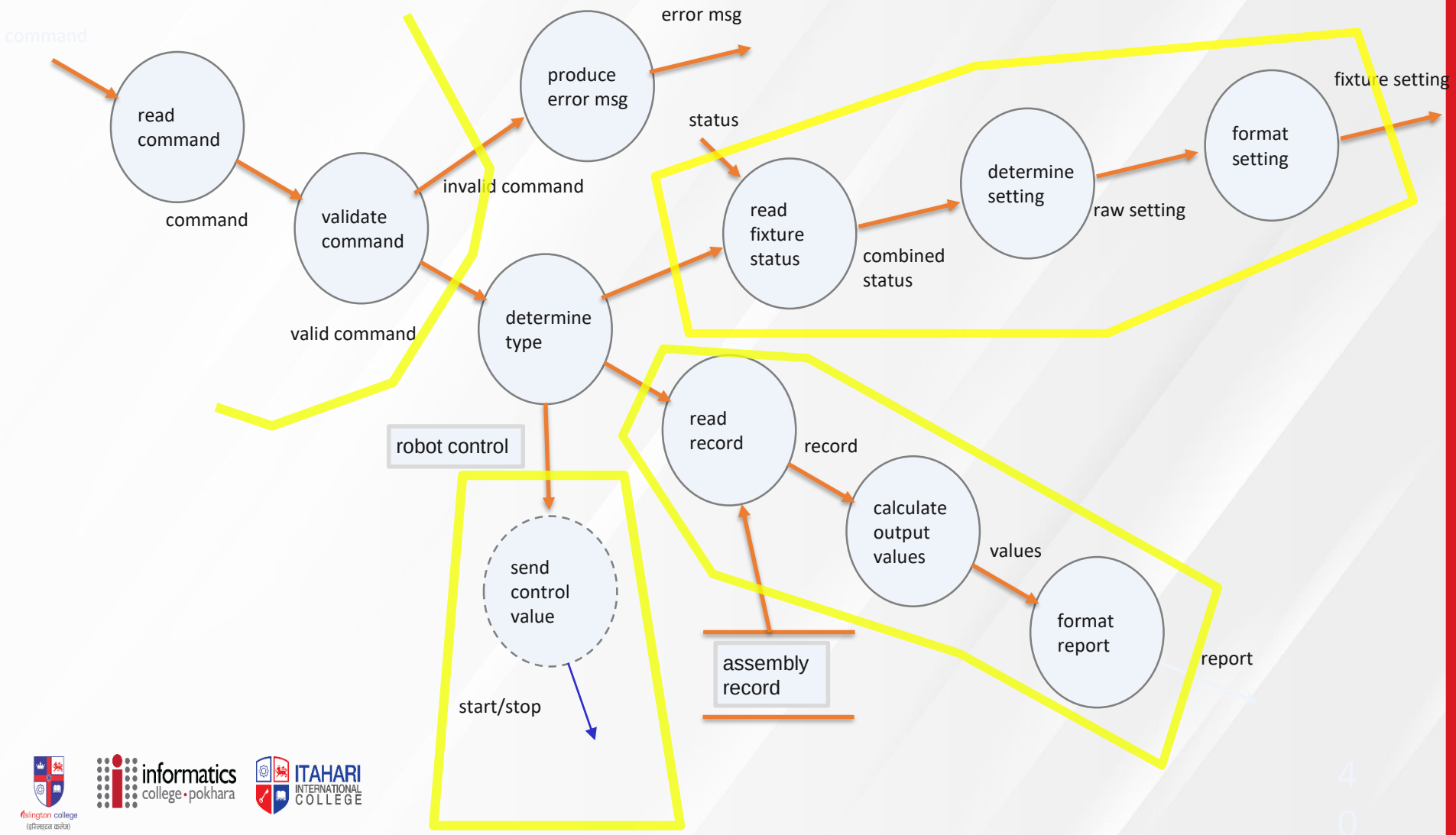
Transaction Example



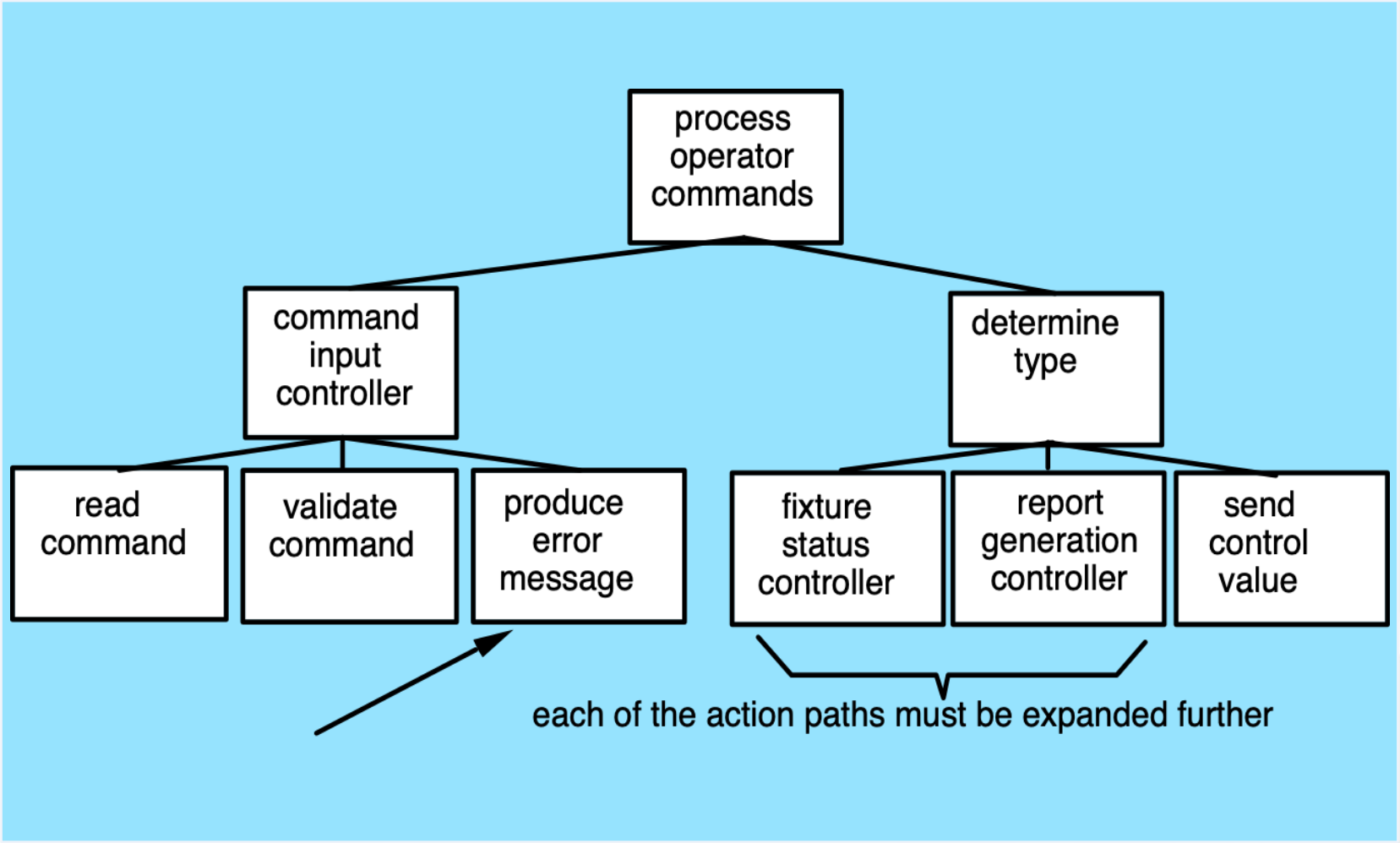
Level 2 Data Flow Diagram



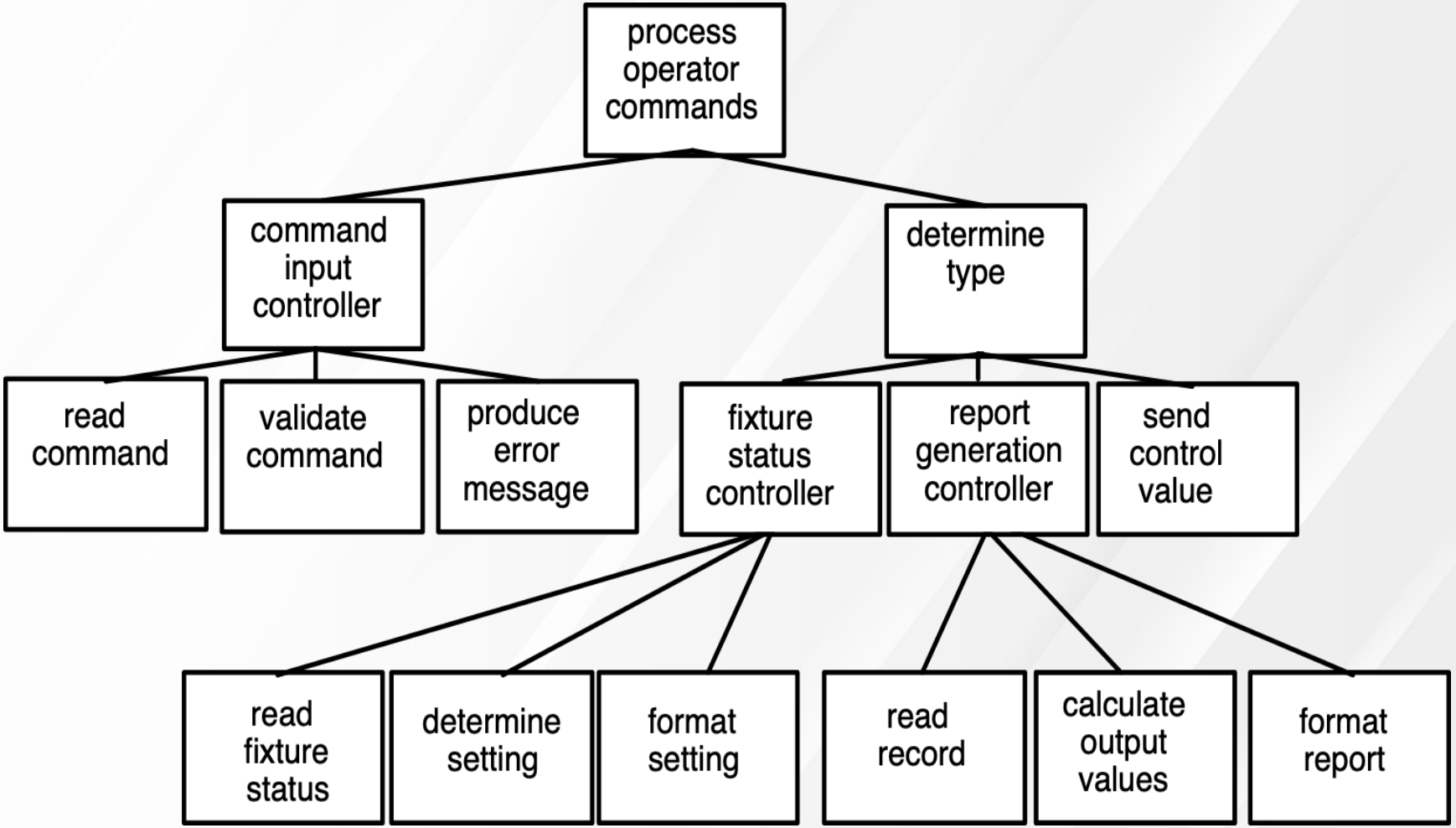
Isolate Flow Paths



Map the Flow Model



Refining the Structure Chart



Structure Chart

Structure charts are used to graphically model the hierarchy of processes within a system. Through the hierarchical format, the sequence of processes along with the movement of data and control parameters can be mapped for interpretation. The control structures of sequence, selection and repetition can all be represented within the chart for a modeled system.

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Structure Chart Use

Describe functions and sub-functions of each part of system

(in more detail than a DFD)

Show relationships between common and unique modules of a computer program

Hierarchical, Modular structure

- Each layer in a program performs specific activities
- Each module performs a specific function

Structure Chart

Process:
Module / Subroutine

Process: Module / Subroutine
A series of instructions that are to be carried out by the program at a specific point.

Decision
Used to represent SELECTION and split the charts sequence into multiple paths.

Call Line
Indicates the path (SEQUENCE) between modules / subroutines.

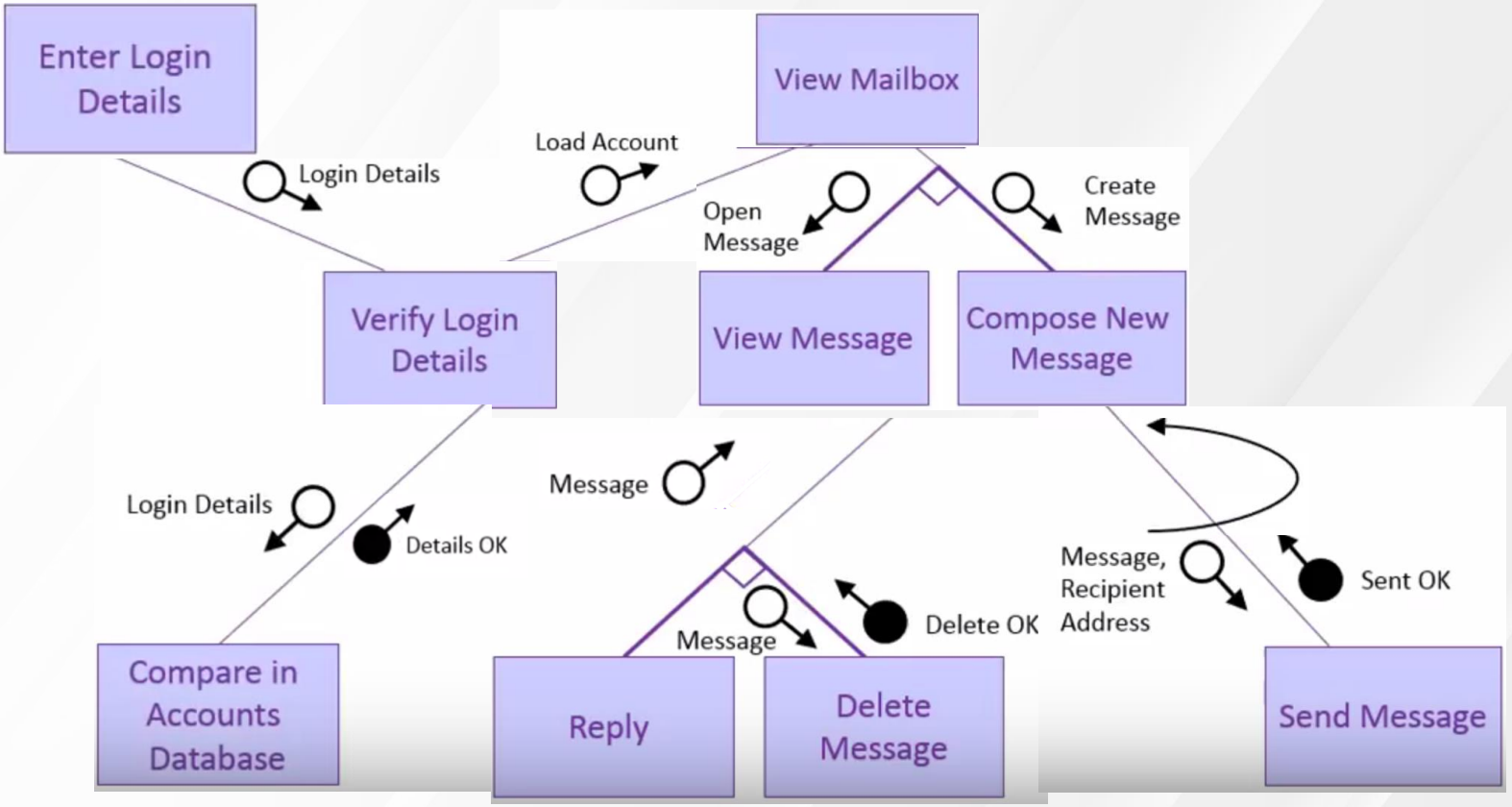
Repetition
Used to represent REPETITION and highlight that a process can occur multiple times.

Parameter
Indicates the flow of DATA between processes, which is labelled with the symbol

Control Parameter
Indicate that a criteria has been met, providing confirmation for the system to proceed. E.g. Flags.

Structure Chart

Structure Chart outlines the use of any Email Server



Structure Chart

Structure Chart outlines the use of any Salary Disbursement system