Midterm Exam

- Answer key and questions are available on the course webpage
- Please let me know if you have questions (come and see me during the office hours or we can find another time)
- Reminder: *Appeals: Should you wish to appeal a grade that you have received on a laboratory assignment, exam, or anything else, you must do so within one week of receiving the graded item.* (Syllabus)

Project 2 Grades

- Grades will be available on Blackboard this week
- Graders:
  - Alex: Presentation, Developments Artifacts, Evaluations (comments)
  - Kurt: Modularity, Stability, Documentation, References
- Please let us know if you have questions
- **You are <u>NOT</u> allowed to ask another team about the grade they gave you on the code base (ASK the instructor, he approved those evaluations)**
- <u>Reminder:</u> *Appeals: Should you wish to appeal a grade that you have received on a laboratory assignment, exam, or anything else, you must do so within one week of receiving the graded item.* (Syllabus)

Architectural Design

Prof. Alex Bardas

Design Model Elements (1/3)

- **Data elements**
  - Data model → structure of data: support component design
  - Data model → database architecture: application/business
- **Architectural elements**
  - Overview of the software-to-be
  - Information from application domain
  - Analysis classes (relationships, collaborations) and flow diagrams are transformed into design realizations
  - Patterns and “styles”

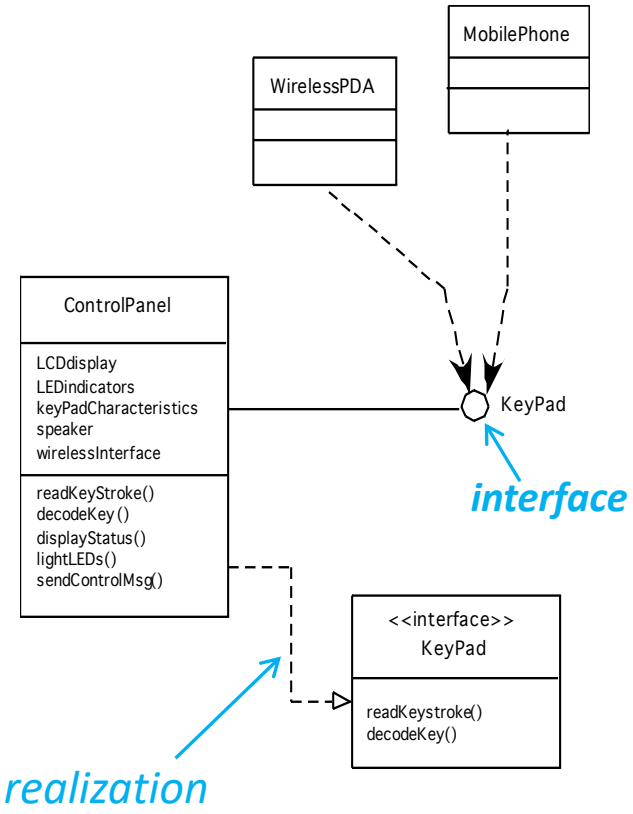
Design Model Elements (2/3)

- **Interface elements**
  - User Interface (UI)
  - External Interfaces to other systems, devices, networks or other producers or consumers of information
  - Internal Interfaces between design components

esthetic, ergonomic, and technical considerations

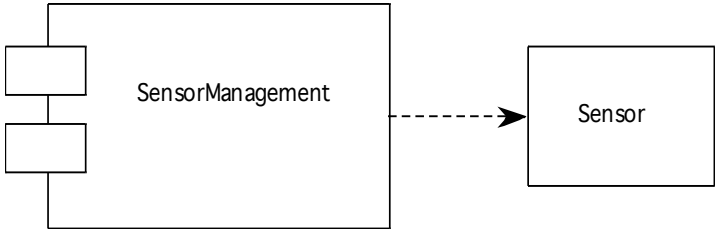
defined in requirements

aligned with component-level design

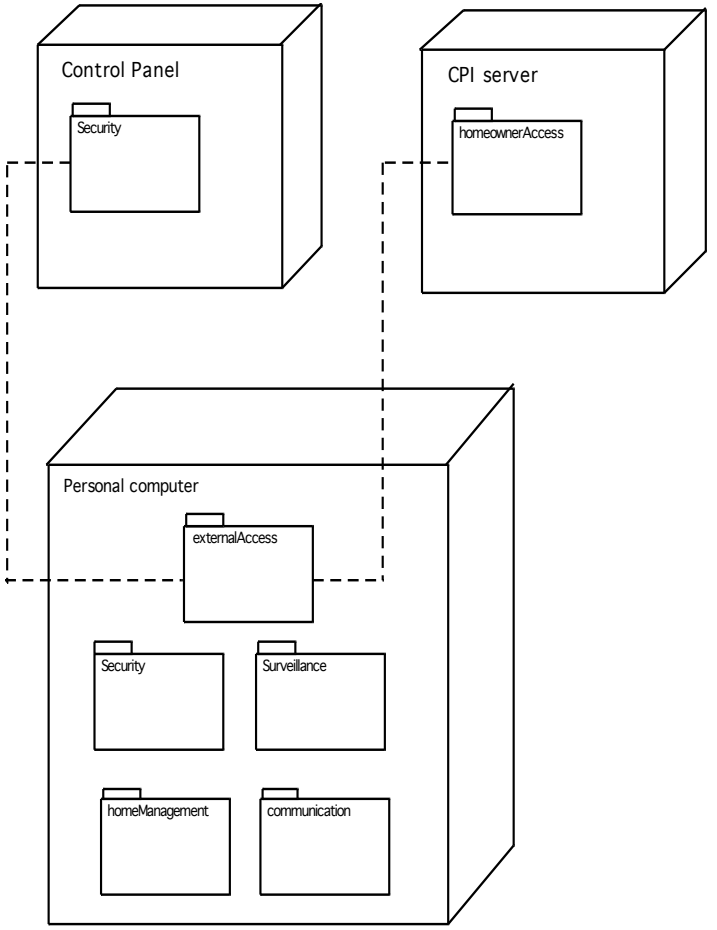


Design Model Elements (3/3)

- **Component elements**
  - Component internal details
  - Data objects
  - Interface for component operations



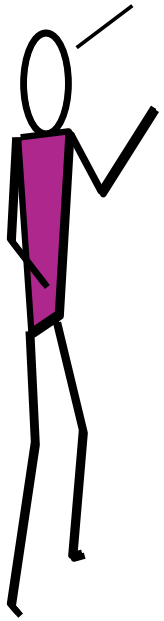
- **Deployment elements**
  - Physical allocation, hardware configuration, etc.

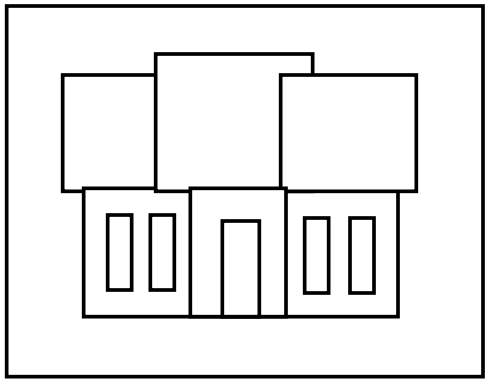


Architectural Design

Customer requirements

"four bedrooms, three baths, lots of glass ..."

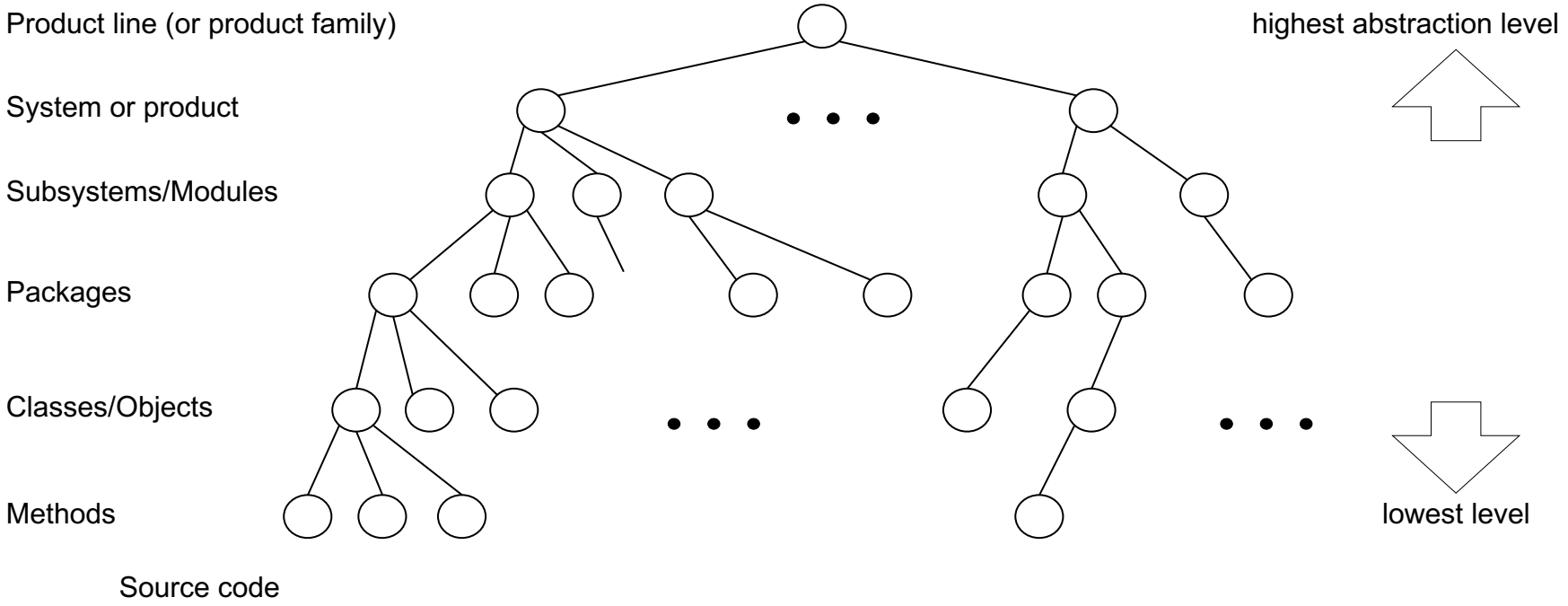




architectural design

Hierarchical Organization of Software

- Different views of abstraction

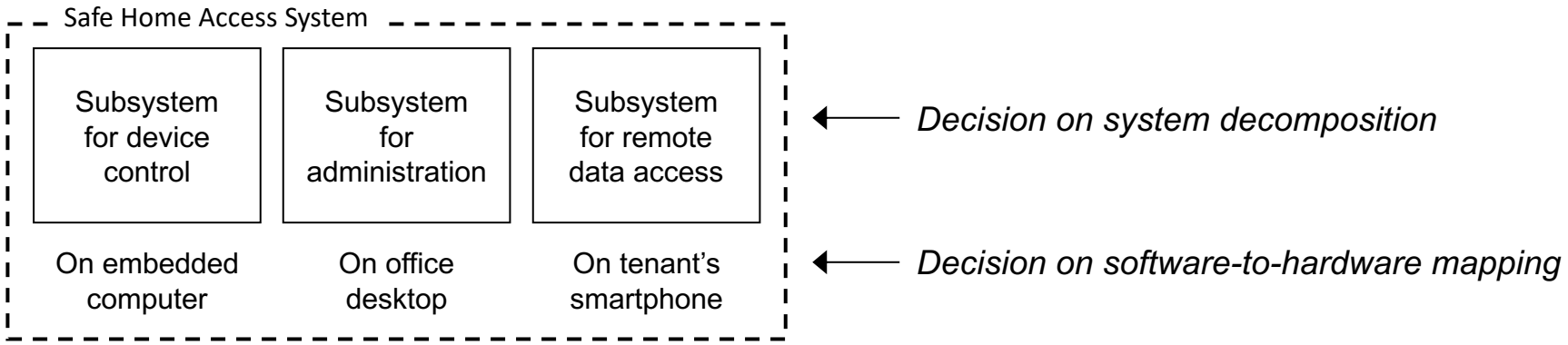


Architectural Design

- Architectural design
  - The design process for identifying the sub-systems that make up a system and the framework for sub-system control and communication
- The output of this design process is a description of the software architecture
- Architectural design should be an early stage of the system design process

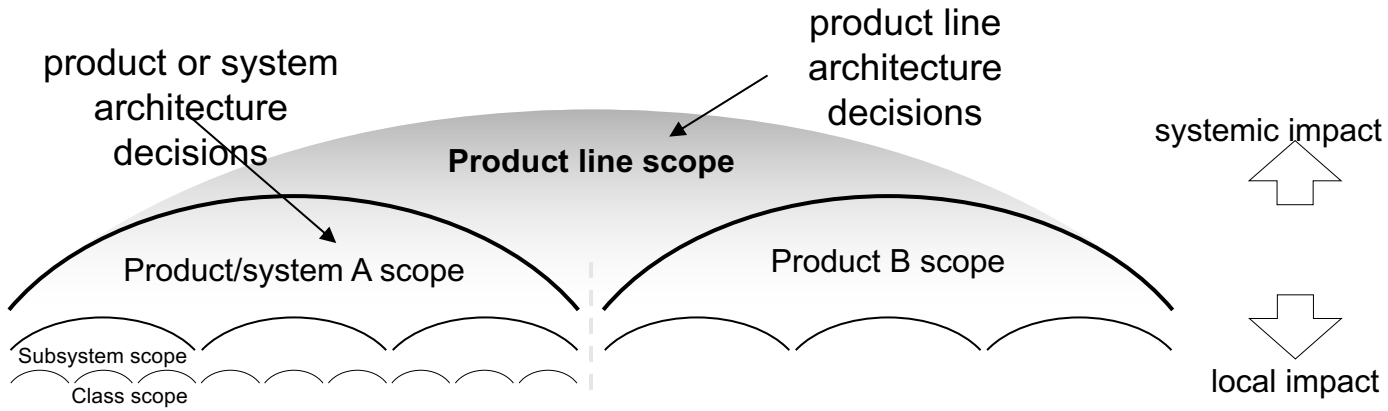
Software Architecture

- Software architecture is a set of **high-level decisions** that determine the key components of the system-to-be and their communications
  - Main (principal) decisions made throughout the development *and* evolution of a software system
  - Made early and affect large parts of the system ("design philosophy") — such decisions are hard to modify later



Architectural Decisions

- Given the current level of system scope, a decision is "**architectural**" if it can be made only by considering the present scope
  - It could not be made from a more narrowly-scoped, local perspective



- Architectural decisions should focus on high impact, high priority areas that are in strong alignment with the business strategy

Why Architecture?

- The architecture is a representation that enables a software engineer to:

  (1) Analyze the effectiveness of the design in meeting its stated requirements

  (2) Consider architectural alternatives at a stage when making design changes is still relatively easy

  (3) Reduce the risks associated with the construction of the software

Why is it Important?

- Software architecture representations facilitate the communication between all parties (stakeholders)
- Architecture highlights, early design decisions that will have a profound impact on all software engineering work that follows
- Architecture ”constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together”

How to Make Architectural Decisions?

- Start with the requirements that define how the system will interact with the environment
- Then, do a system decomposition
  - Tackle complexity by “divide and conquer”
  - See if some parts already exist & can be reused
  - Focus on creative parts
  - Support flexibility and future evolution by decoupling unrelated parts (“separation of concerns”)
- Decision making often involves compromise
  - Limitations and constraints in the context
  - Business priorities, available resources, core competences, competitors’ moves, technology trends, existing investments, etc.

Key Concerns

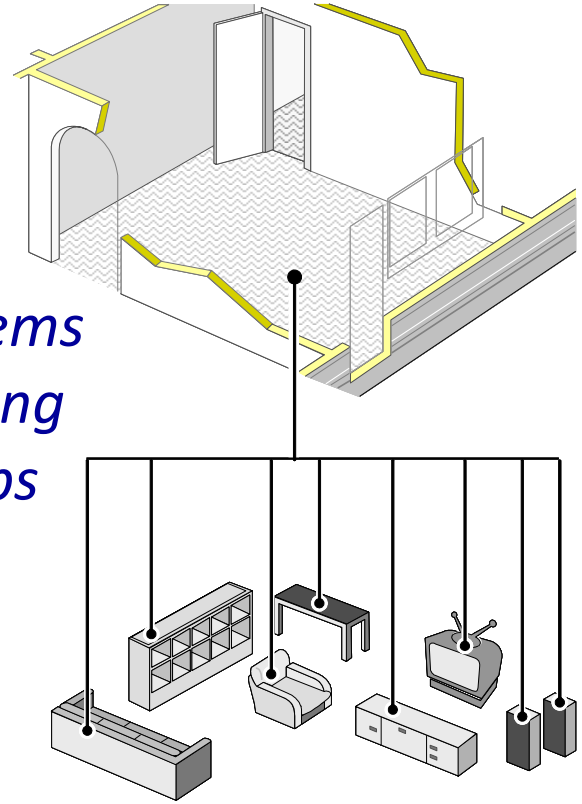
- Key concerns when making main (principal) decisions
  - System decomposition
    - How do we break up the system into the right pieces?
    - Do we have all the necessary pieces?
    - Do the pieces *fit* together?
  - Cross-cutting concerns
    - Broad-scoped qualities or properties of the system
    - Tradeoffs among the qualities
  - Conceptual integrity

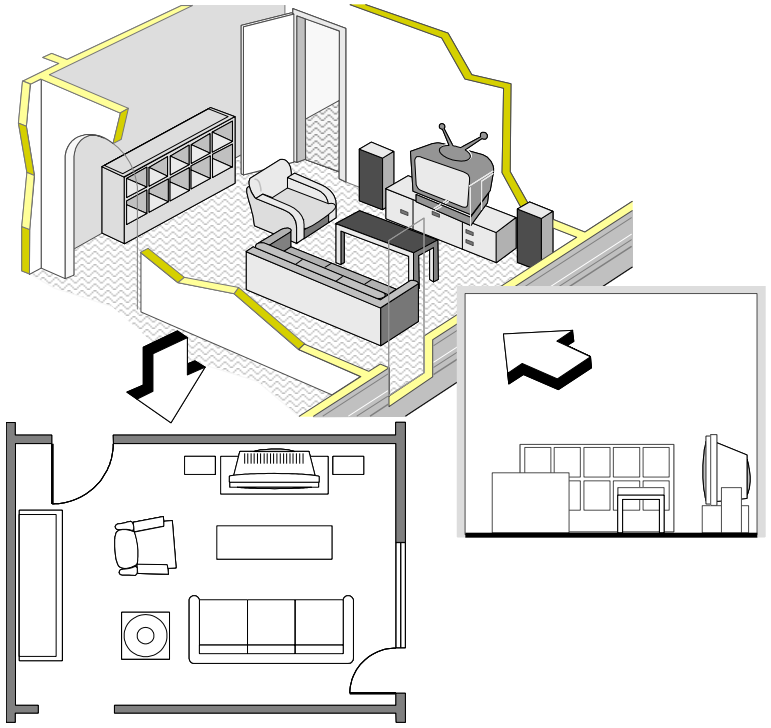
Decomposition

- Partition vs. Projection

simplifying representation – reducing dimensions

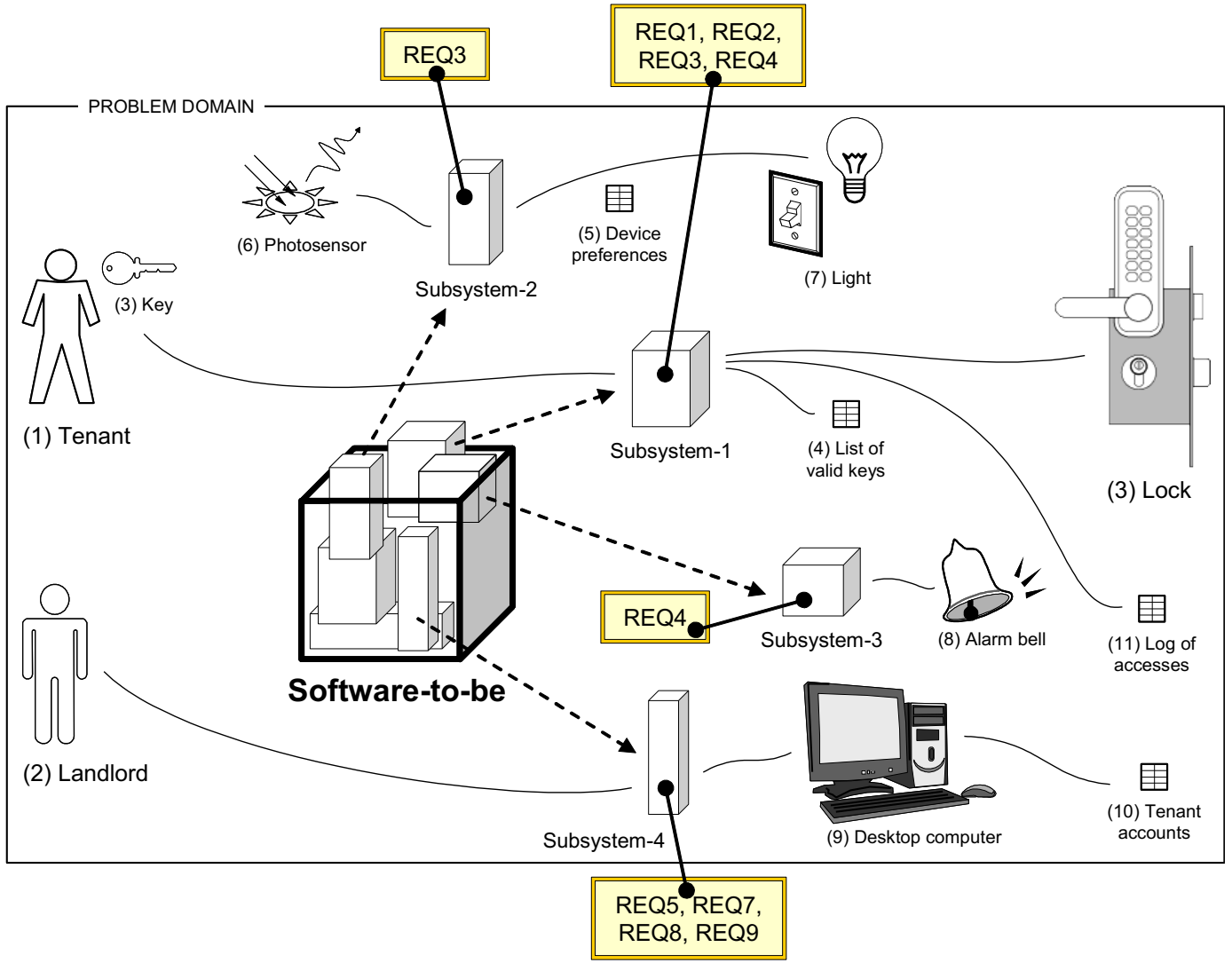
isolating items and removing relationships





An Example

REQ1: Keep door locked and auto-lock
REQ2: Lock when “LOCK” pressed
REQ3: Unlock when valid key provided
REQ4: Allow mistakes but prevent dictionary attacks
REQ5: Maintain a history log
REQ6: Adding/removing users at runtime
REQ7: Configuring the device activation preferences
REQ8: Inspecting the access history
REQ9: Filing inquiries



Architectural Design Process

- System structuring
  - The system is decomposed into several principal sub-systems and communications between these sub-systems are identified
- Control modelling
  - A model of the control relationships between the different parts of the system is established
- Modular decomposition
  - The identified sub-systems are decomposed into modules

Sub-systems and Modules

A sub-system is a system in its own right whose operation is independent of the services provided by other sub-systems.

A **module** is a system component that provides services to other components but would not normally be considered as a separate system

Real-world Sub-system Examples

- Typically organized as Java packages/C++ libraries/C# assemblies
- Database access layer
  - MySQL access, JDBC layer
- Security services
  - Encryption classes, signature classes (modules)
- External payment sub-system
- Email service sub-system
- Logging sub-system
- Financial transaction sub-system
- Marketing sub-system

Architectural Styles

- The architectural design is expressed as a block diagram
  - Nodes represent **components**
    - Procedures, modules, processes, tools, databases
  - Edges represent **connectors**
    - Procedure calls, event broadcasts, database queries, pipes
  - A set of **constraints** on how they can be integrated
  - Semantic **models** to help understand the overall properties of a system
- Architectural style:

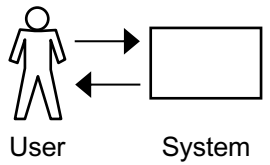
Defines a family of systems in terms of a pattern of structural organization

Architectural Models

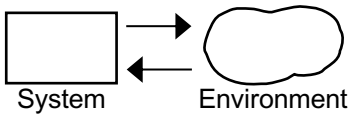
- Different architectural models may be produced during the design process
- Architectural models and their combination are selected to best fit the characteristics and constrains from requirements modeling
- Each model presents different perspectives on the architecture:
  - Static structural model: shows the major system components
  - Dynamic process model: shows the process structure of the system
  - Interface model: defines sub-system interfaces
  - Relationships model: e.g., data-flow model

Typical Problems

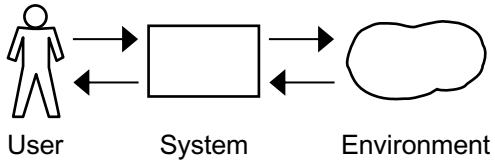
1. User works with computer system (environment irrelevant/ignored)



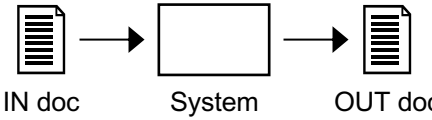
2. Computer system controls the environment (user not involved)



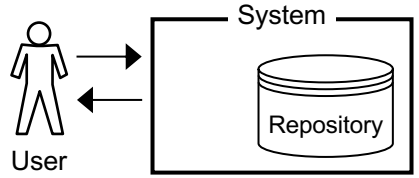
3. Computer system intermediates between the user and the environment



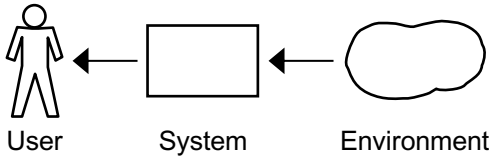
1.a) System transforms input document to output document



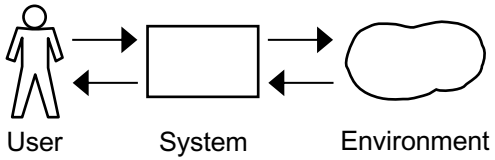
1.b) User edits information stored in a repository



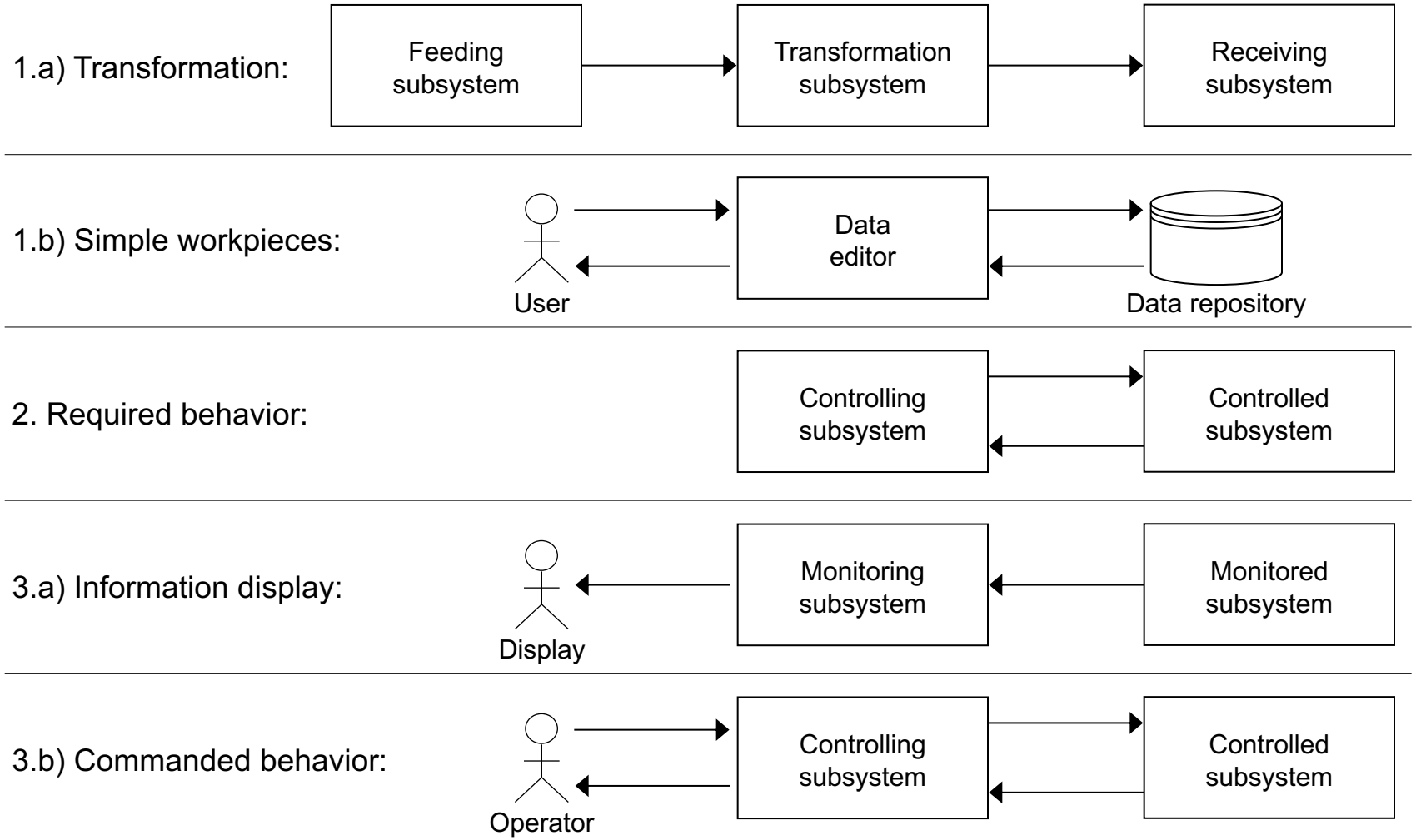
3.a) System observes the environment and displays information



3.b) System controls the environment as commanded by the user



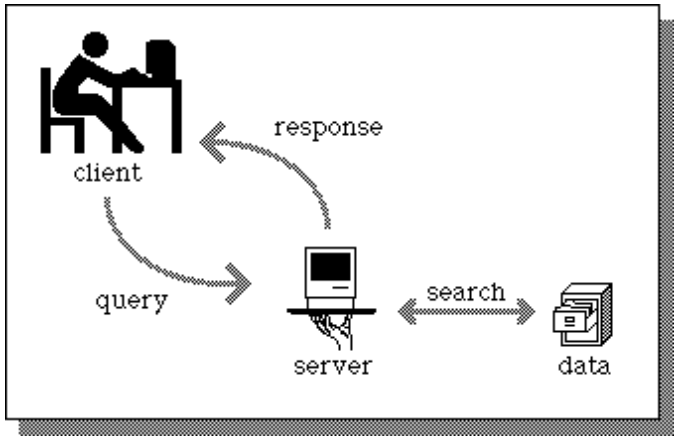
Problem Architecture



Architectural Models

- **The Repository Model**
  - Sub-systems must exchange data
- This may be done in two ways:
  - Shared data is held in a central database or repository and may be accessed by all sub-systems
    - When large amounts of data are to be shared, the repository model of sharing is most commonly used
  - Each sub-system maintains its own database and passes data explicitly to other sub-systems

Architectural Models



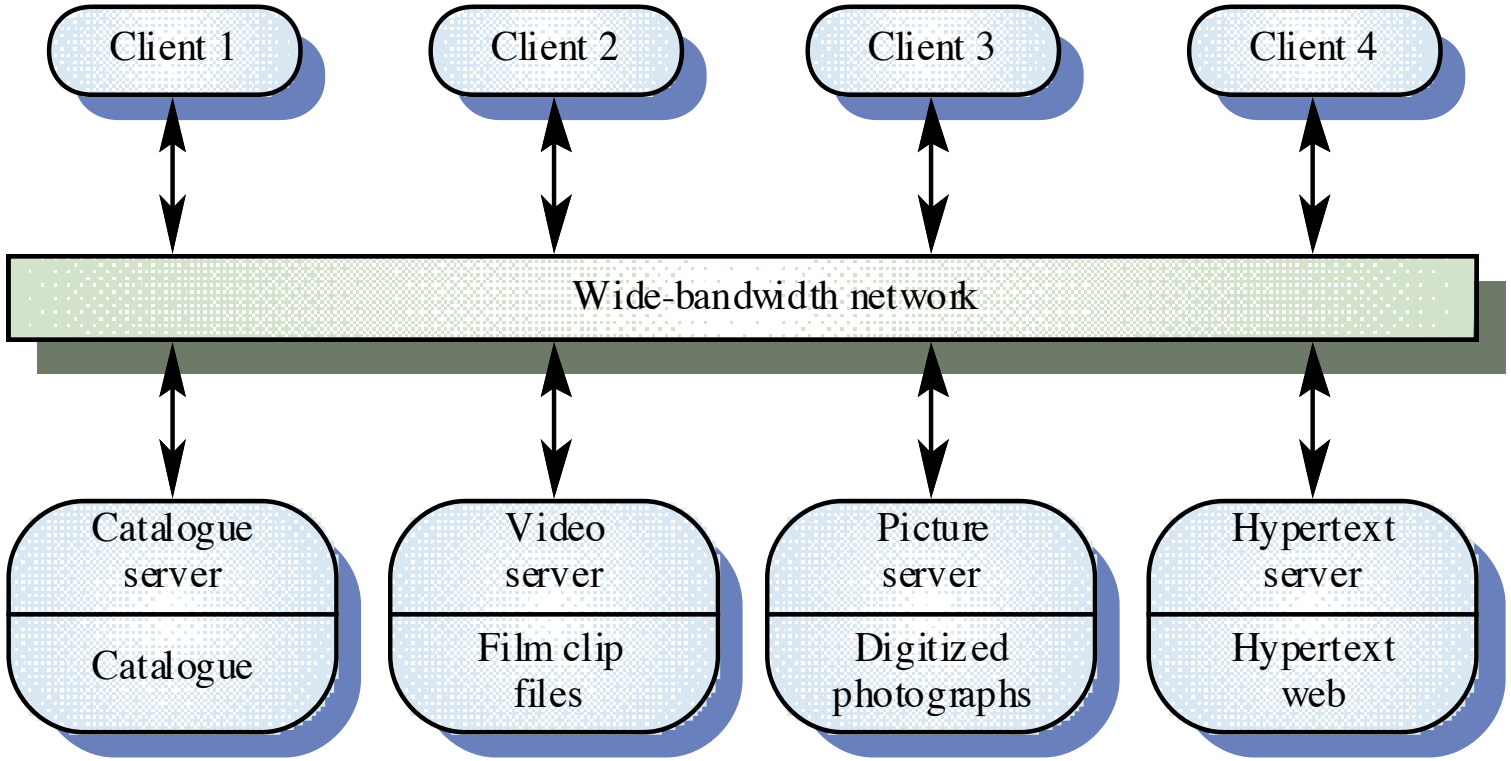
- **Client-Server Architecture**
  - A distributed system model
- Data and processing are distributed across a range of components:
  - Set of stand-alone servers: which provide specific services such as printing, data management, etc.
  - Set of clients: which call on these services
  - Network: which allows clients to access servers

Architectural Models

- **Client-Server Architecture**
  - Advantages
    - Distribution of data is straightforward
    - Makes effective use of networked systems. May require cheaper hardware
    - Easy to add new servers or upgrade existing servers
  - Disadvantages
    - No shared data model so sub-systems use different data organisation; data interchange may be inefficient
    - Redundant management in each server
    - No central register of names and services - it may be hard to find out what servers and services are available

Example

- Film and Picture Library

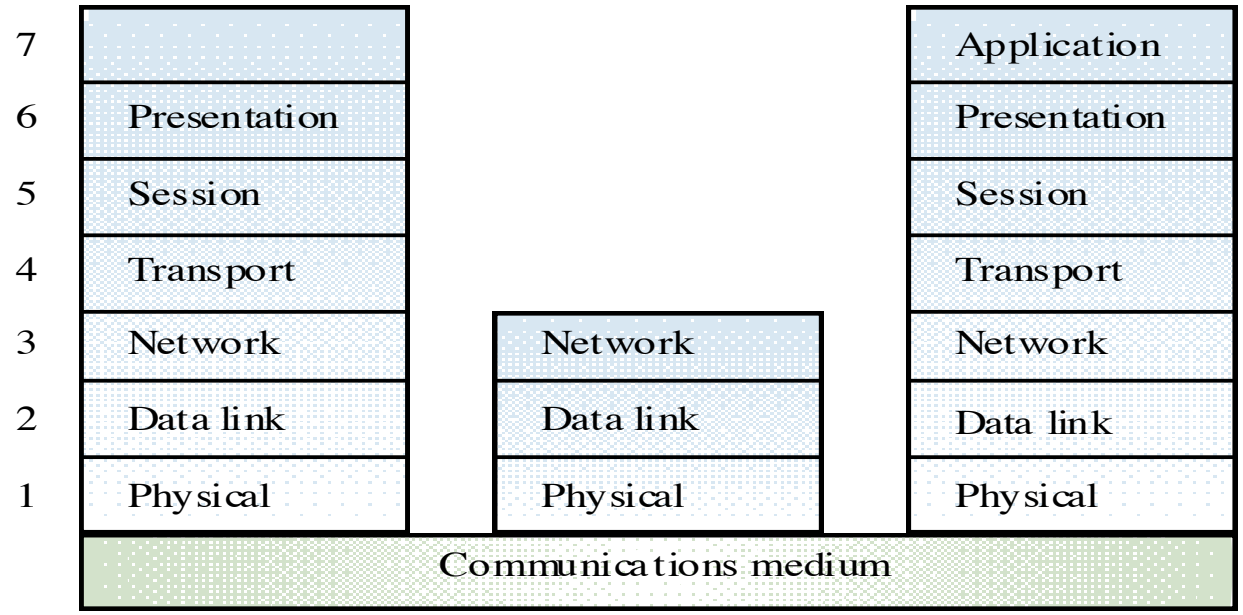


Architectural Models

- **Abstract Machine Model**
  - Used to model the interfacing of sub-systems
  - Organises the system into a set of layers (or abstract machines)
    - Each of which provide a set of services
  - Supports the incremental development of sub-systems in different layers
    - When a layer interface changes, only the adjacent layer is affected
- However, it is often difficult to structure systems in this way

Example

- ISO/OSI Network Model



Architectural Models

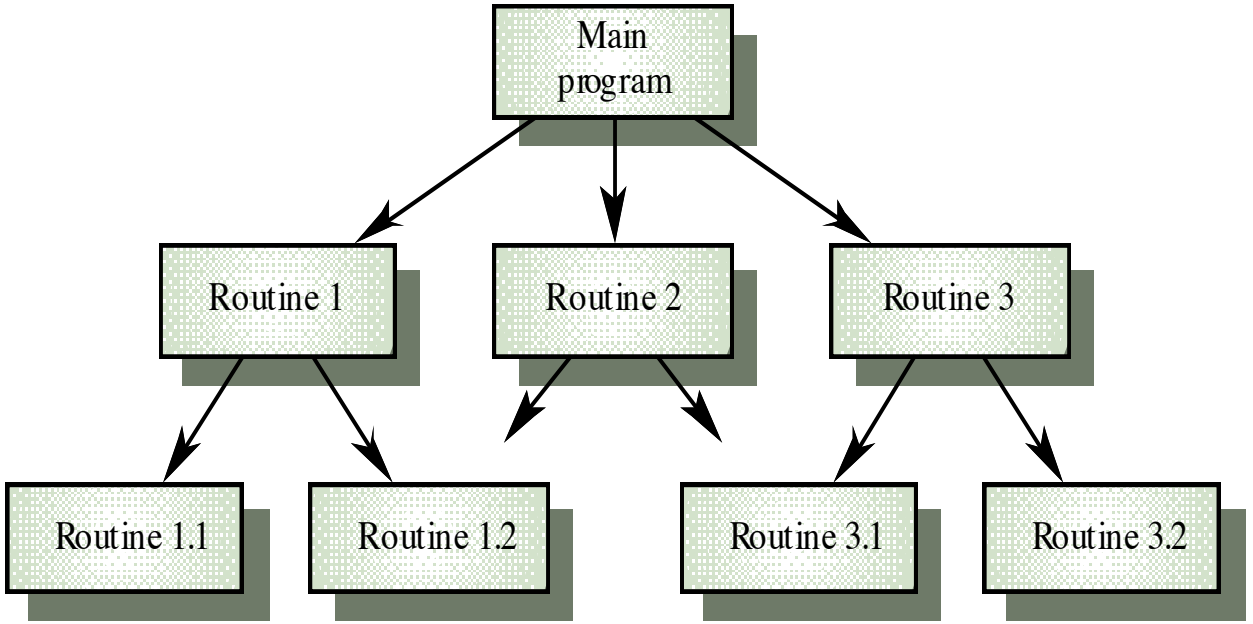
- **Control Models**
  - Concerned with the control flow between sub systems
- Centralised control
  - A control sub-system takes responsibility for managing the execution of other sub-systems
  - Sequential or parallel

Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Centralised control
  - **Call-return model**
    - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards
    - Such a model is embedded into familiar programming languages such as C, Java, etc.

Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Centralised control
  - **Call-return model**

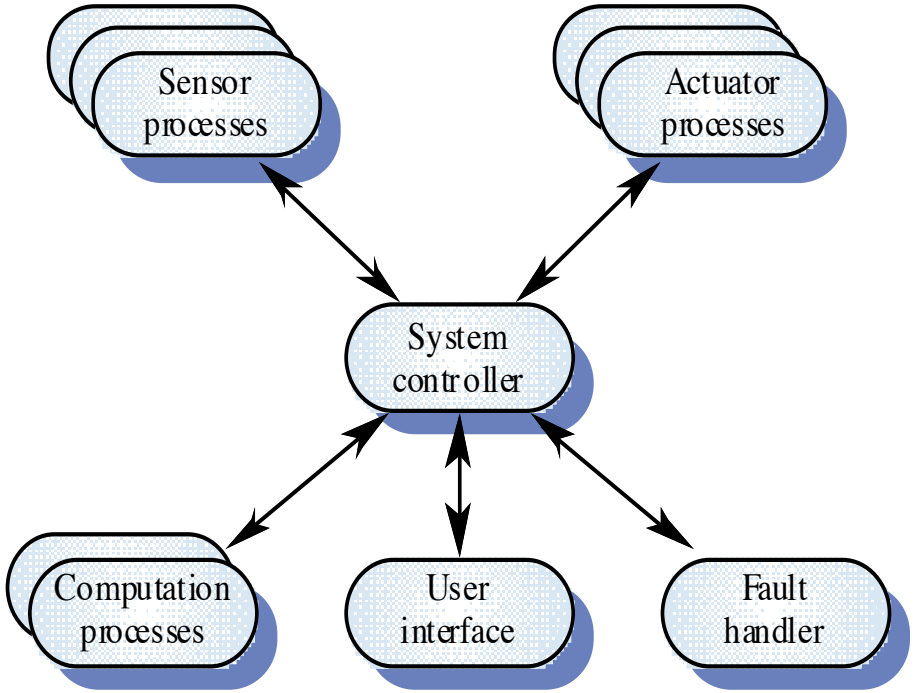


Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Centralised control
  - **Manager model**
    - One system component controls the stopping, starting and coordination of other system processes
    - Can be applied to concurrent or sequential systems

Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Centralised control
  - Manager model
    - e.g., real-time system



Architectural Models

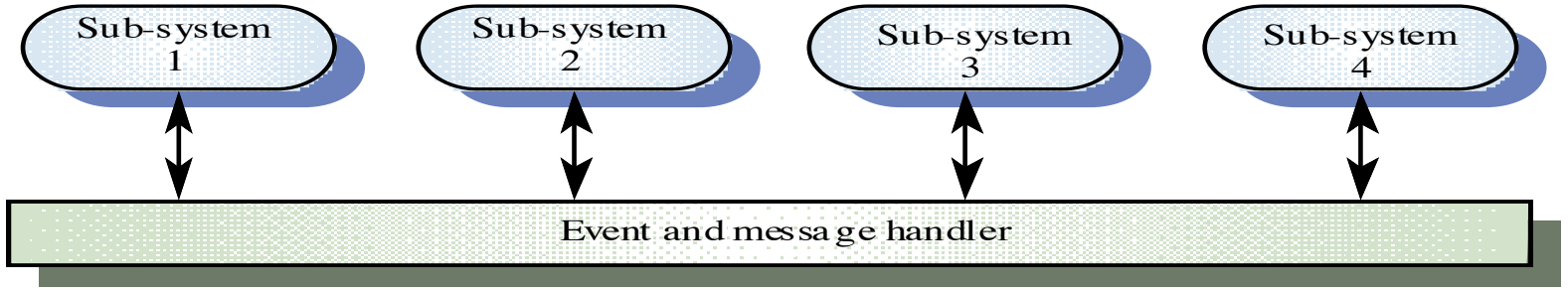
- **Control Models**
  - Concerned with the control flow between sub systems
- Event-based control
  - Driven by externally generated events
  - Each sub-system can respond to externally generated events from other sub-systems or the system's environment
  - Timing of the event is out of the control of the sub-systems

Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Event-based control
  - **Broadcast models**
    - Sub-systems register an interest in specific events
    - An event is broadcast to all sub-systems
    - When the events occur, control is transferred to the sub-system which can handle the event
    - Control policy is not embedded in the event
    - Effective in integrating sub-systems

Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Event-based control
  - **Broadcast models**

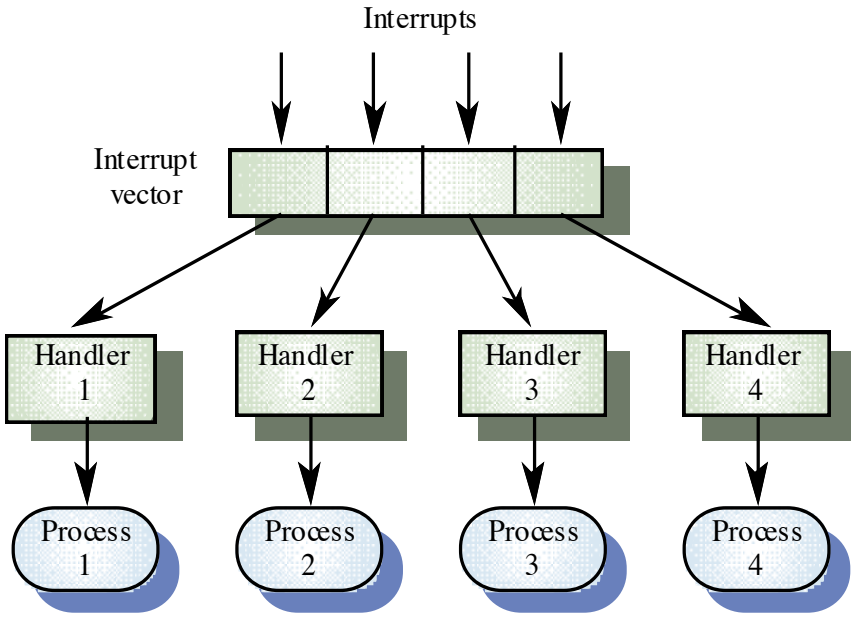


Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Event-based control
  - **Interrupt-Driven Systems**
    - A handler defined for each type of known interrupt
    - Each type is associated with a memory location and a hardware switch causes transfer to its handler
    - Used in real-time systems where a fast response to an event is essential

Architectural Models

- **Control Models**
  - Concerned with the control flow between sub systems
- Event-based control



Architectural Styles

- “Template for construction”
- 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

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

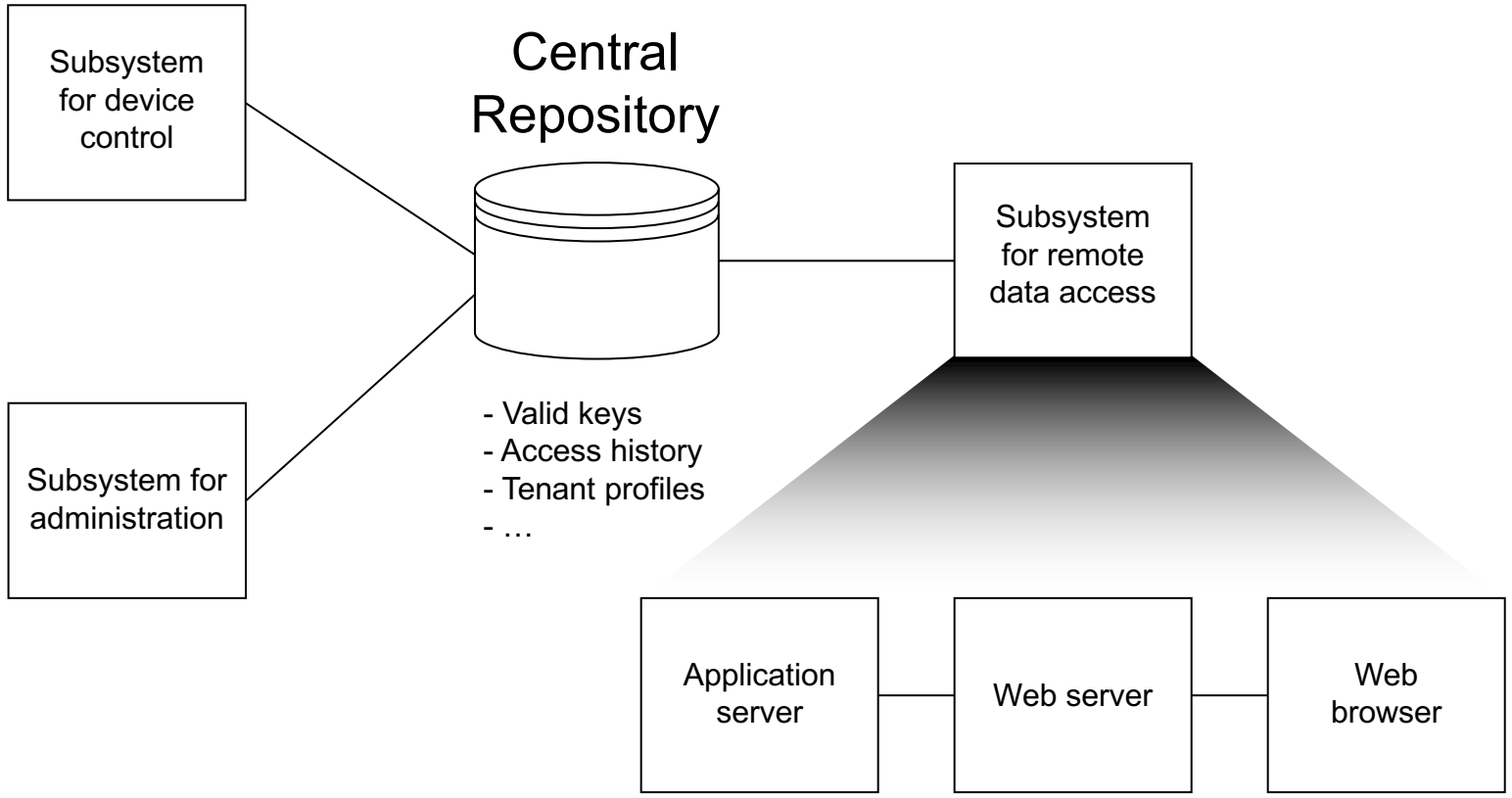
Architectural Genres

- Genre implies a specific category within the overall software domain

  e.g., AI, commercial, communications, content authoring, financial, games, etc.

  - Within each category, you encounter a number of *subcategories:* Within the genre of buildings, there are general styles: houses, condos, apartment buildings, office buildings, industrial building, warehouses, etc.
  - Within each general style, more specific *styles* might apply: Each style would have a structure that can be described using a set of predictable patterns

A Real System is a Combination of Styles



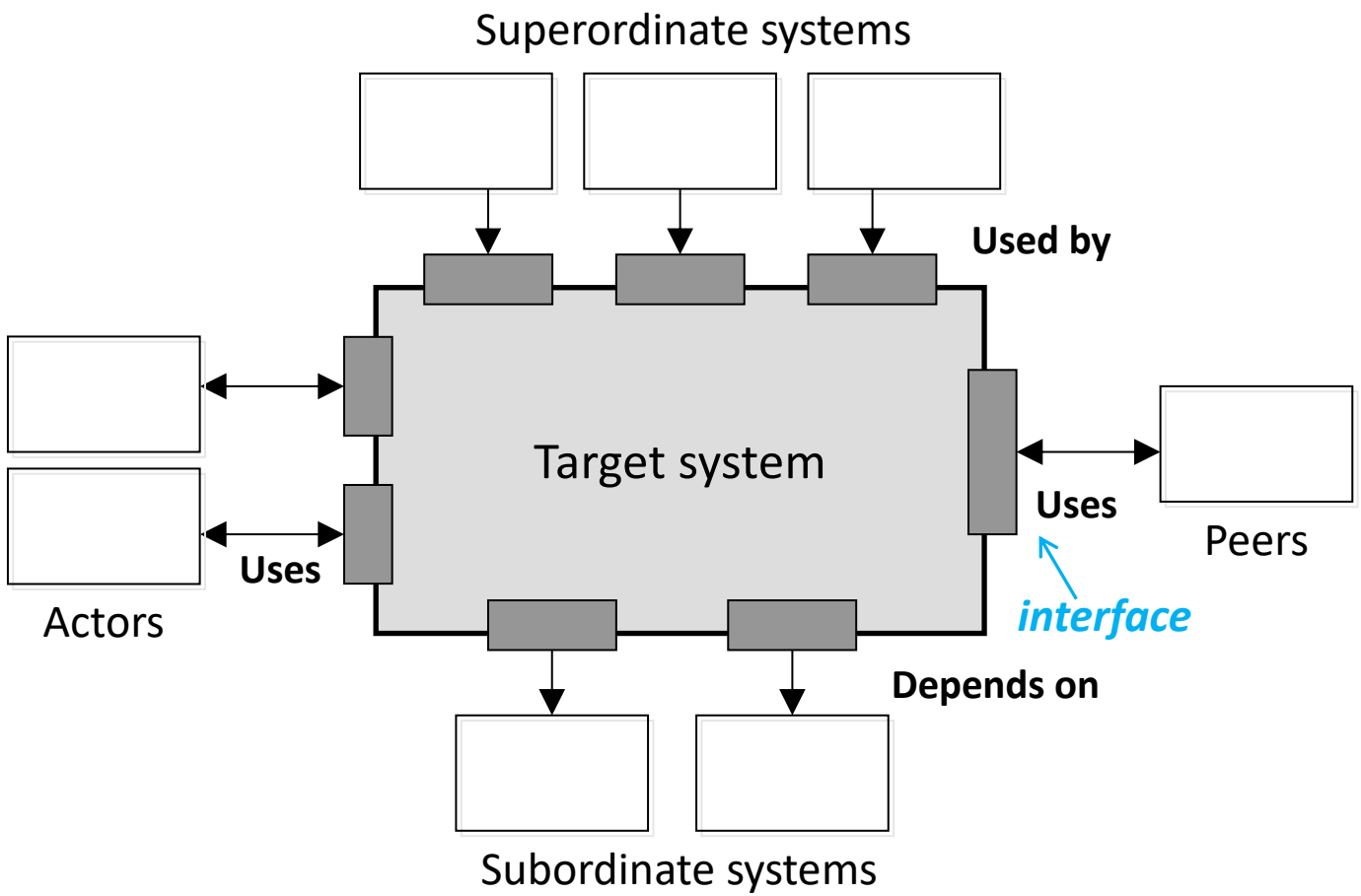
Architectural Design

- The software must be placed into *context*
  - The external entities
  - The interactions
- Identify a set of architectural *archetypes*
  - Similar to a class, it's an abstraction that represents one element of system behavior
- Define and refine software *components* to specify the structure of the software
  - Component implements an archetype

Representing a System in Context

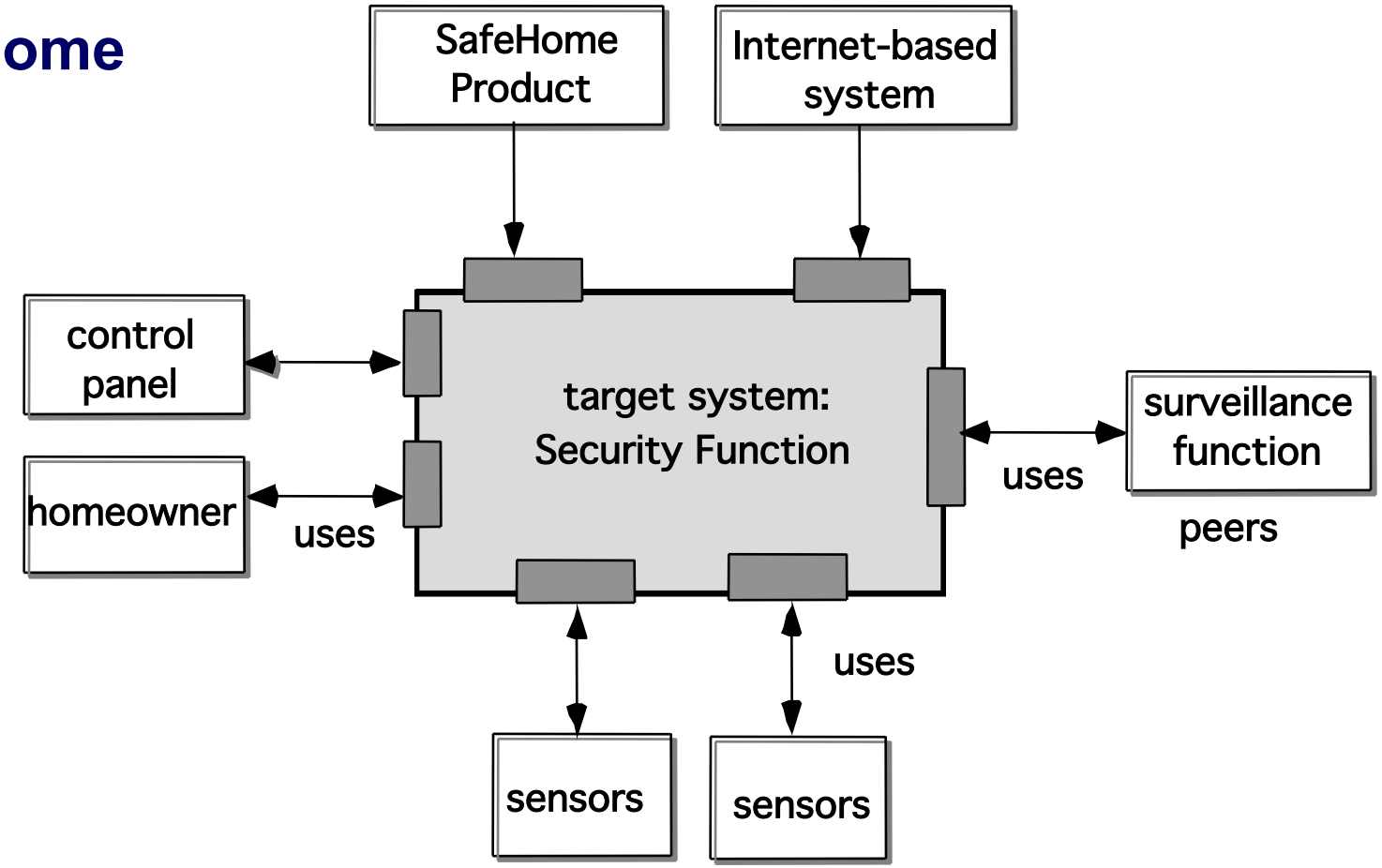
- **Architectural Context Diagram (ACD):** models how the target system interacts with entities ***external*** to its boundaries.
  1. *Superordinate systems* – those systems that use the target system as part of some higher level processing scheme
  2. *Subordinate systems* – those systems that are used by the target system and provide data or processing that are necessary to complete target system functionality
  3. *Peer-level systems* – those systems that interact on a peer-to-peer basis (i.e., information is either produced or consumed by the peers and the target system)
  4. *Actors* – those entities (people, devices) that interact with the target system by producing or consuming information that is necessary for requisite processing

Architectural Context Diagram



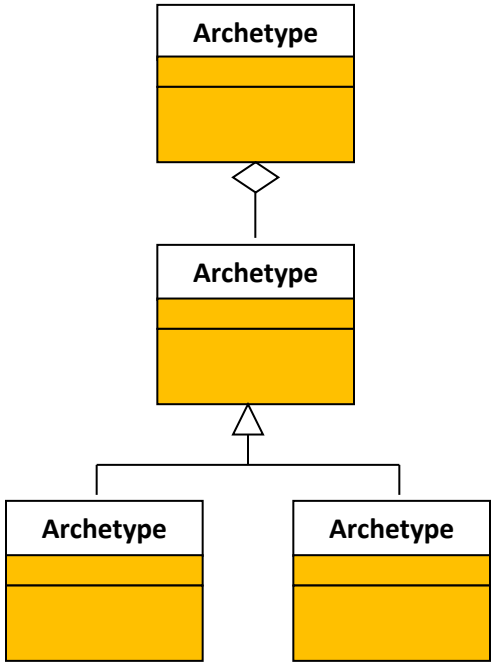
Example ACD

SafeHome

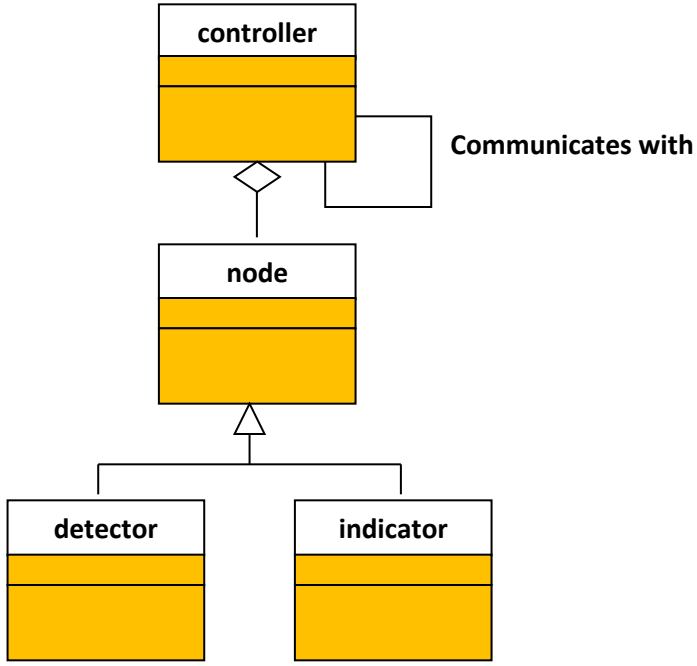


Defining Archetypes

- Archetype is a class or pattern that represents a ***core abstraction*** that is ***critical*** to the design of an architecture for the target system



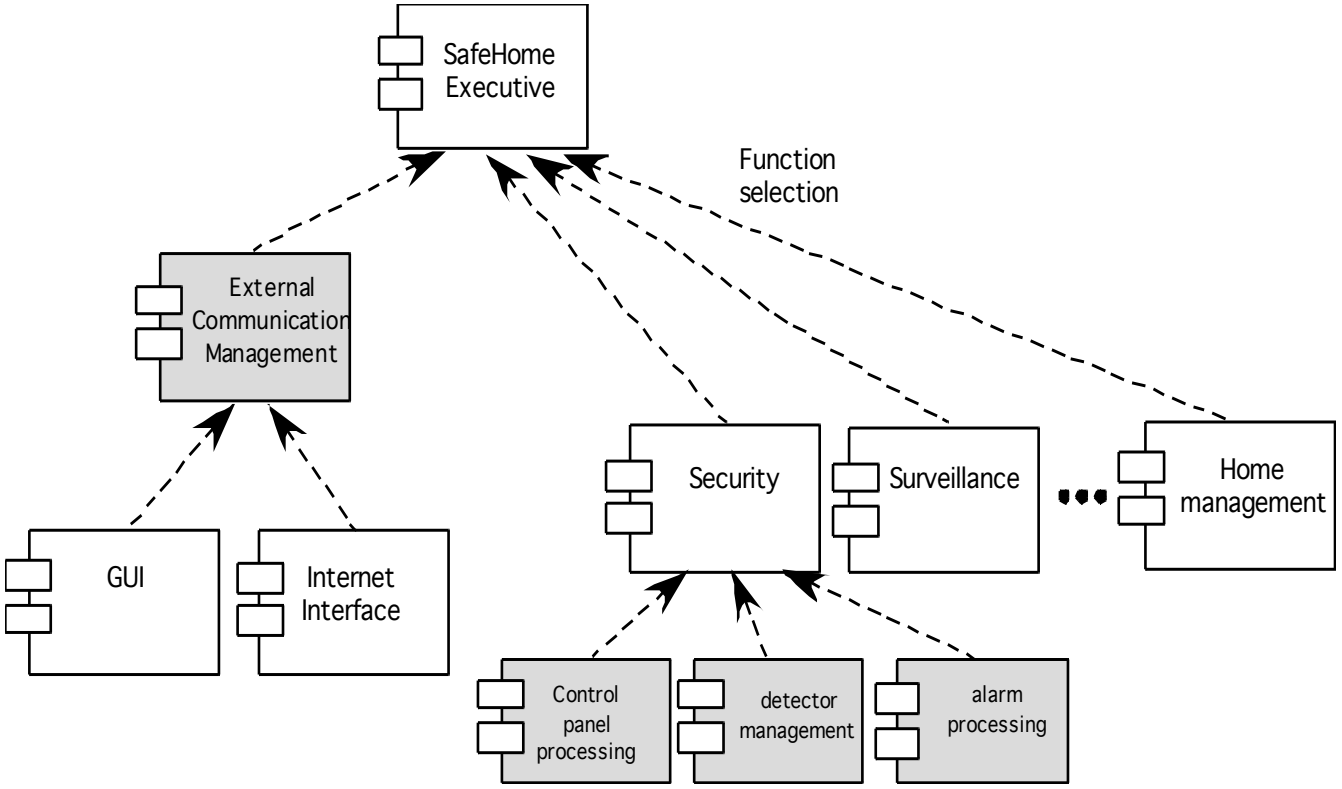
Target system



SafeHome Security Function

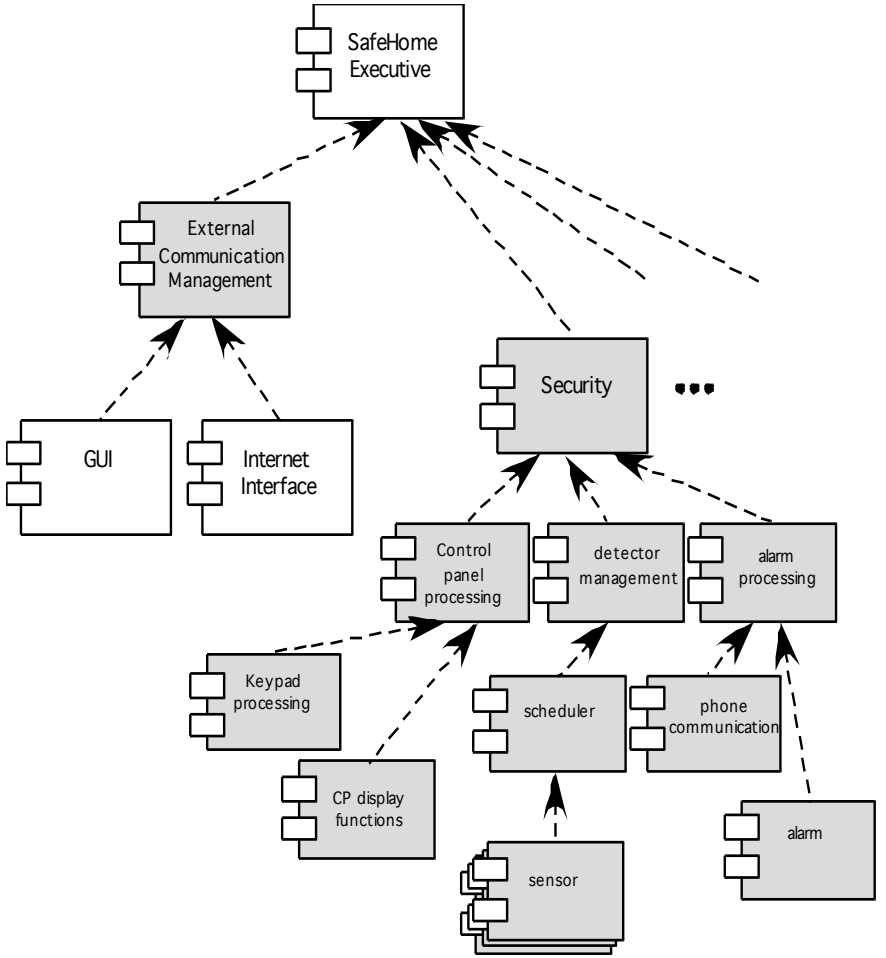
Refining the Architecture into Components

Top-level components in the SafeHome architecture



Instantiation of the Architecture

- Further refinement: to apply the abstract architecture within the problem domain
- Instantiation elaborates components with additional detail



Assessing Alternative Designs

- Architecture Trade-off Analysis Method (ATAM)
  - Developed by SEI
  - An iterative method to assess design tradeoffs

1. Collect scenarios (use cases).

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 (reliability, performance, security, etc.) by considering each attribute in isolation.

5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style.

6. Critically analyze candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

Assessing Alternative Designs

- Architectural Complexity
  - The overall complexity of a proposed architecture is assessed by considering the dependencies between components within the architecture
    - *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

Modular Decomposition

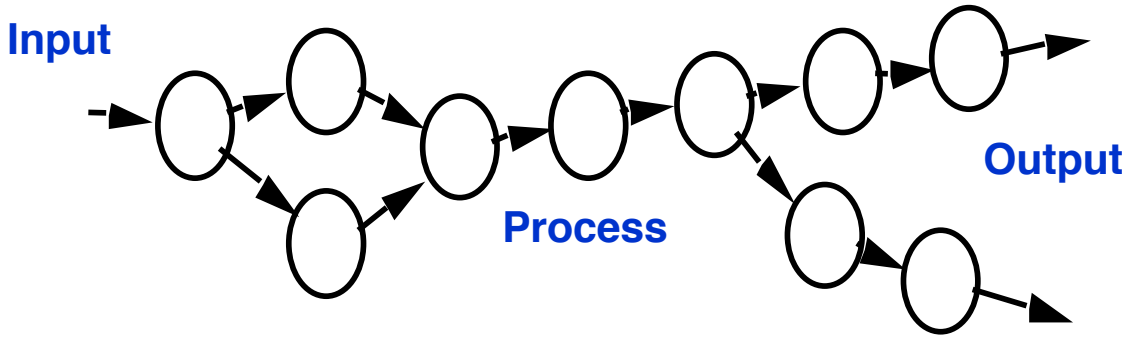
- Sub-systems are decomposed into modules:
  - Object model
    - The system is decomposed into interacting objects
  - Data-flow model
    - The system is decomposed into functional modules which transform inputs to outputs. Also known as the pipeline model
- If possible, decisions about concurrency should be delayed until modules are implemented

Object Models

- Structure the system into a set of loosely coupled objects with well-defined interfaces
- Object-oriented decomposition aims at identifying
  - Object classes
  - Their attributes and operations
- When implemented, objects are created from these classes and some control model used to coordinate object operations

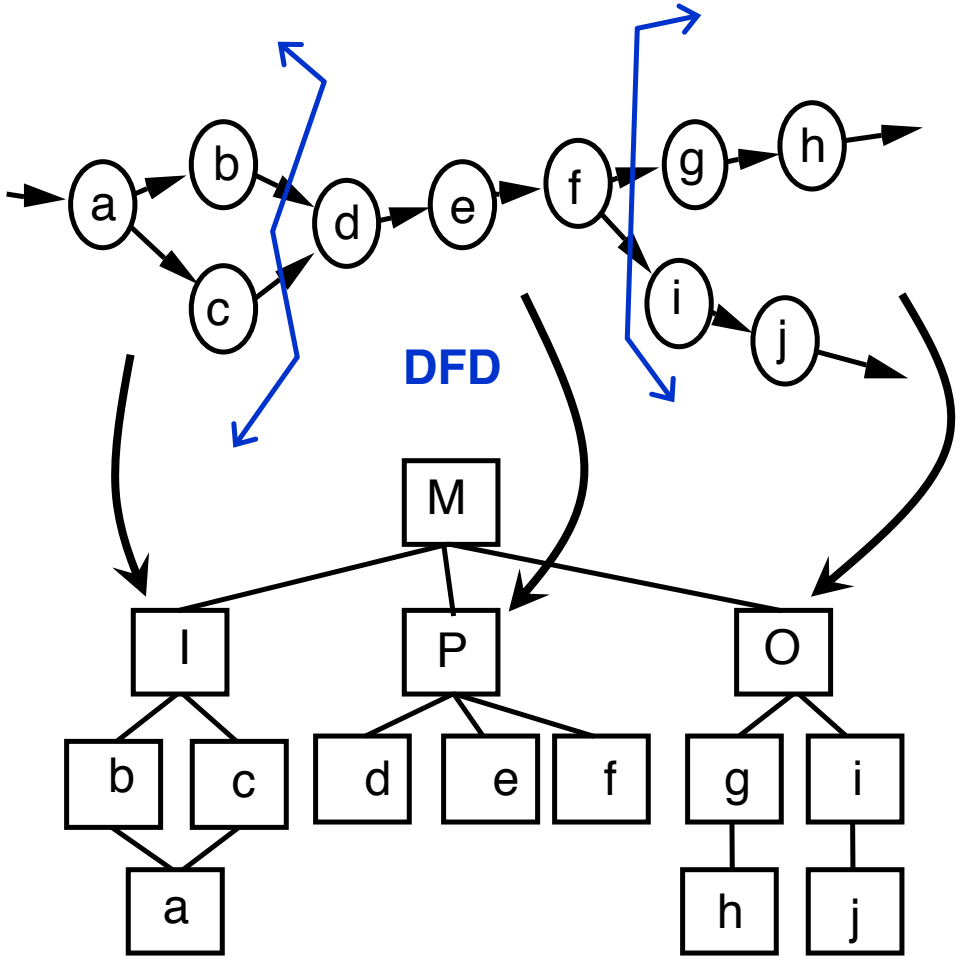
Data-Flow Models

- Functional transformations process their inputs to produce outputs
  - May be referred to as a **pipe and filter** model (as in UNIX shell)
  - When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems
- Not really suitable for interactive systems



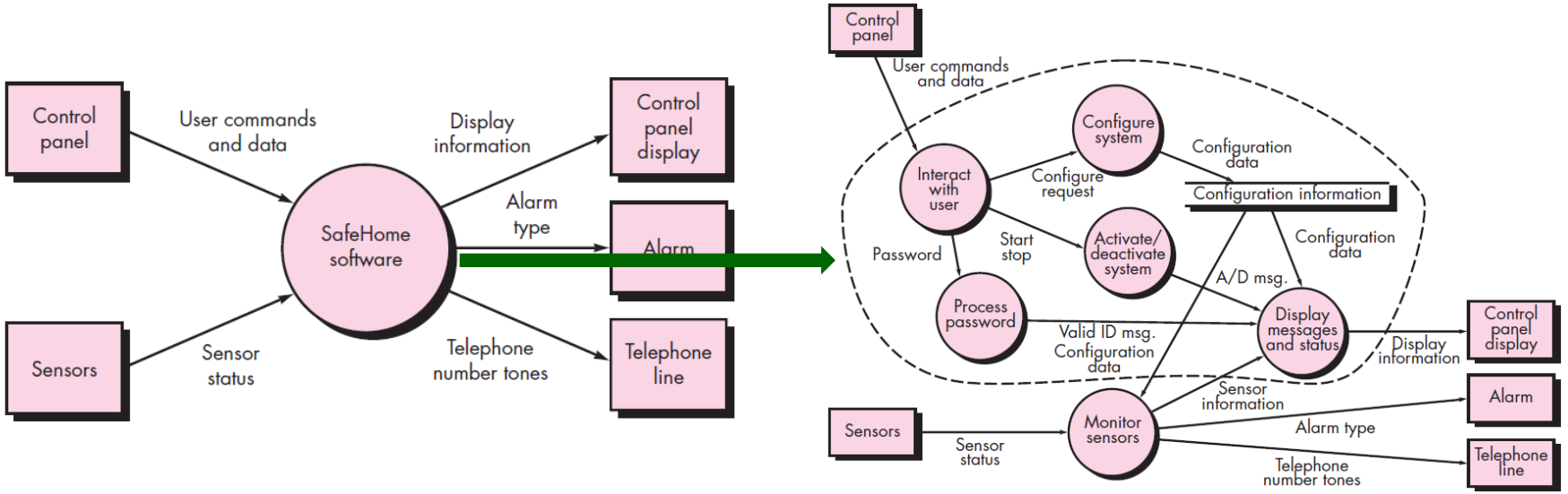
Transform Mapping

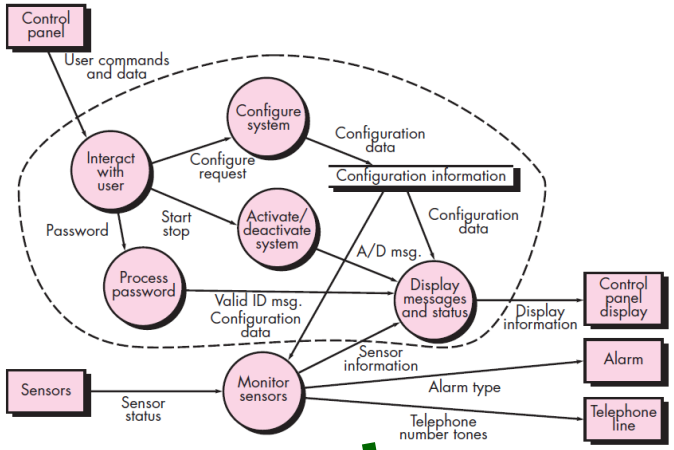
- Transforms a Data Flow Diagram (DFD) to an architectural style.
  - e.g., deriving the call-and-return architecture



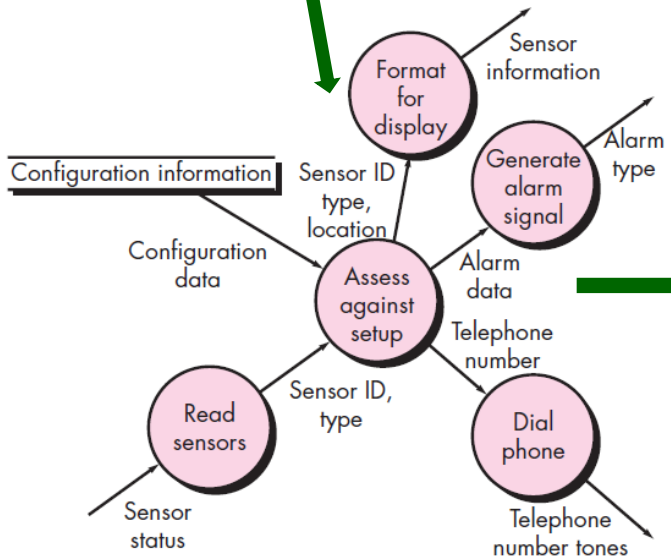
Example

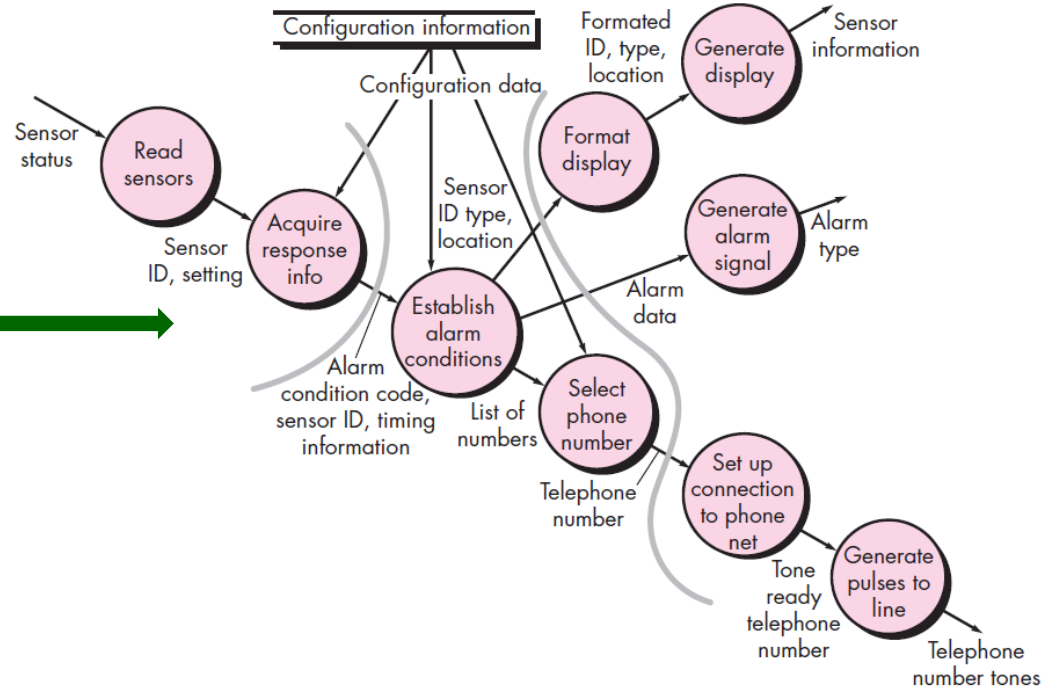
- **Step 1:** Review the fundamental system model
- **Step 2:** Review and refine DFDs





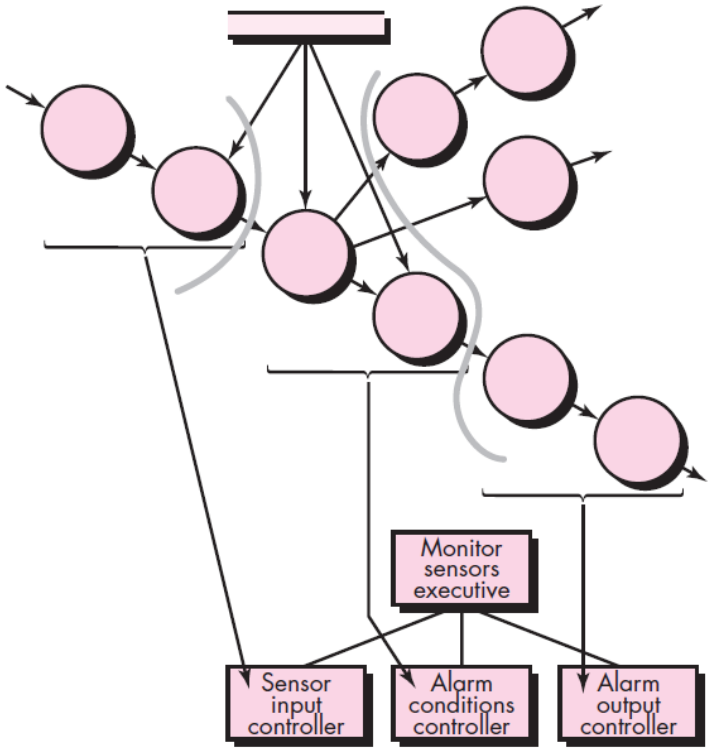
- Refine until each transform shows high cohesion – having a single distinct function
- Indicate flow boundaries





Example

- **Step 3**: Determine the flow characteristics (transform or transaction)
- **Step 4**: Isolate the transform center by specifying incoming and outgoing boundaries
  - Separates incoming and outgoing data flows
  - Select reasonable boundaries

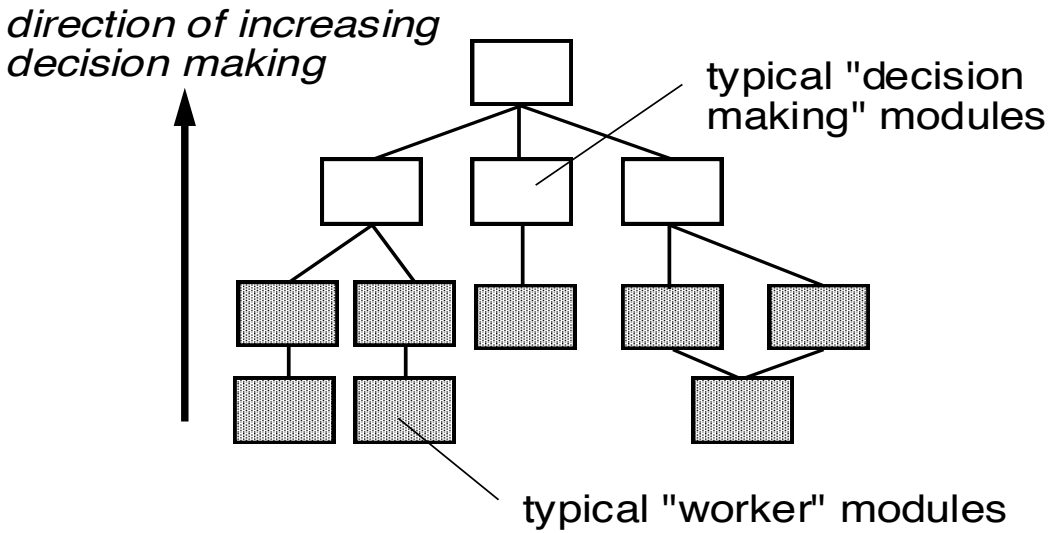


first-level factoring

Example

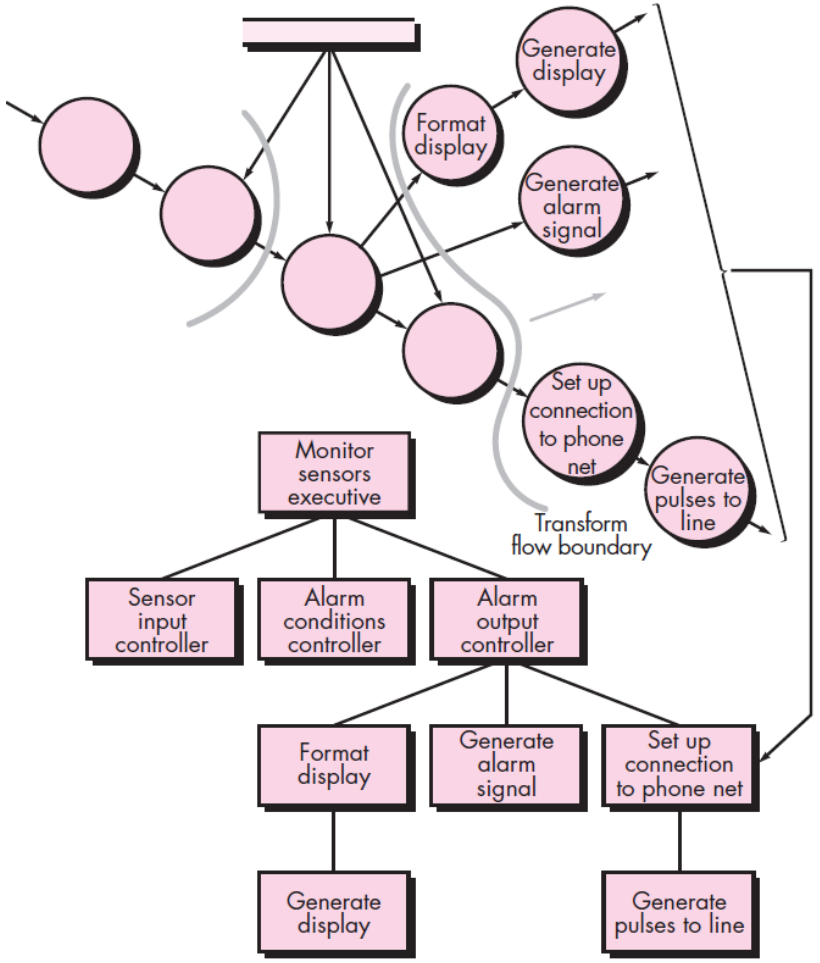
- **Step 5:** Perform first-level factoring
  - Factoring: A program structure of “decision makers” and “workers”

Top-down distribution of control



Example

- **Step 6**: Perform second-level factoring
  - Map transforms into modules
    - Begin at the transform center
    - Work from the boundary outward
    - Not necessarily one-to-one
  - "First-iteration" design
    - Process Specification (PSPEC) and a state transition diagram can be used
    - Indicate the content of each module
- **Step 7**: Refine the first-iteration design



References

- Prof. Fengjun Li's EECS 448 Fall 2015 slides

- This slide set has been extracted and updated from the slides designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill 2014) by Roger Pressman