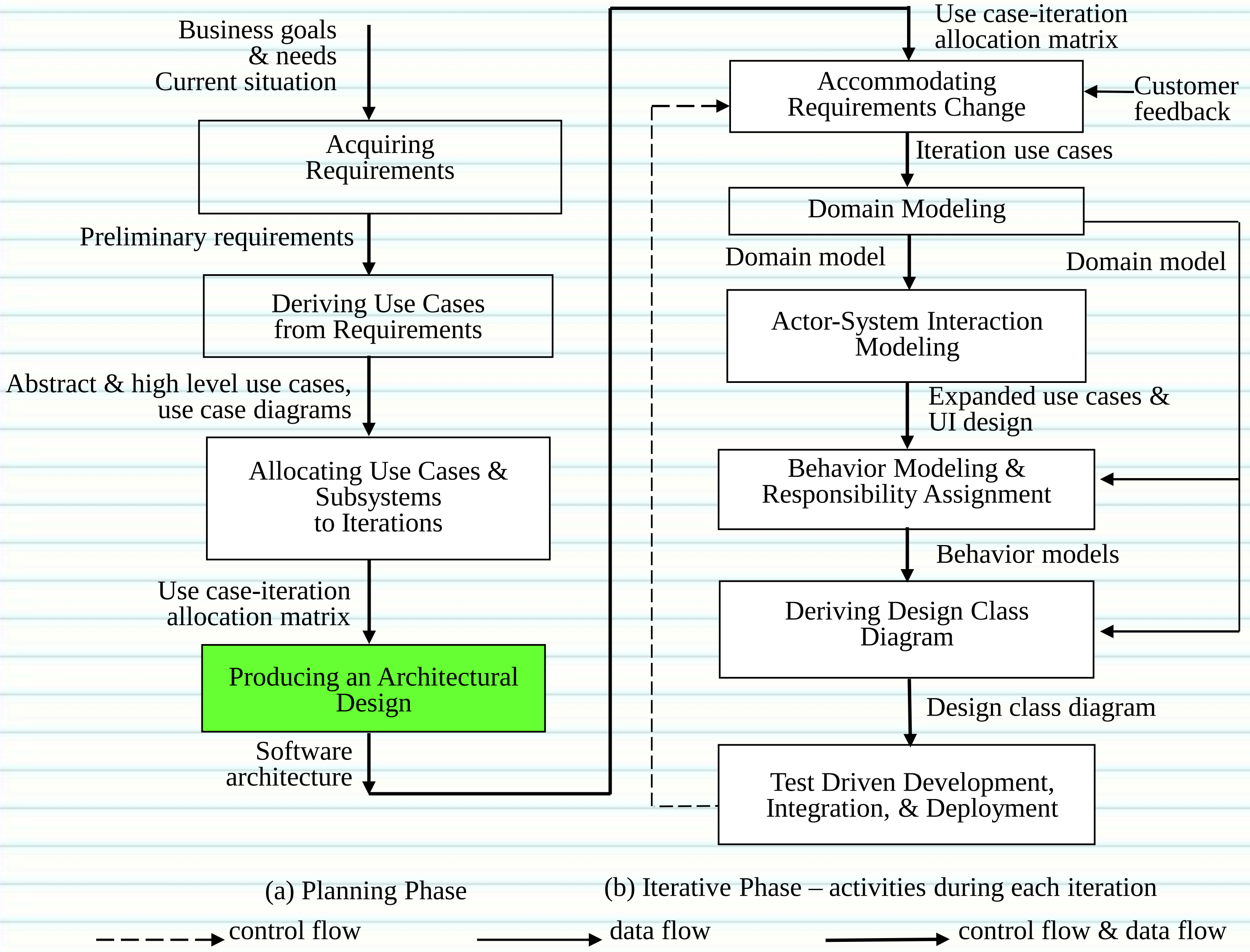
Chapter 6 – Architecture Design

Dr John H Robb, PMP, IEEE SEMC
UT Arlington
Computer Science and Engineering

Key Takeaway Points

- The software architecture of a system or subsystem refers to the style of design of the structure of the system including the interfacing and interaction among its subsystems and components.
- Different types of systems require different design methods and architectural styles.

Architectural Design in Methodology Context


Business goals & needs
Current situation
Acquiring Requirements
Preliminary requirements
Deriving Use Cases from Requirements
Abstract & high level use cases, use case diagrams
Allocating Use Cases & Subsystems to Iterations
Use case-iteration allocation matrix
Producing an Architectural Design
Software architecture
Use case-iteration allocation matrix
Accommodating Requirements Change
Customer feedback
Iteration use cases
Domain Modeling
Domain model
Domain model
Actor-System Interaction Modeling
Expanded use cases & UI design
Behavior Modeling & Responsibility Assignment
Behavior models
Deriving Design Class Diagram
Design class diagram
Test Driven Development, Integration, & Deployment
(a) Planning Phase
(b) Iterative Phase – activities during each iteration
control flow
data flow
control flow & data flow

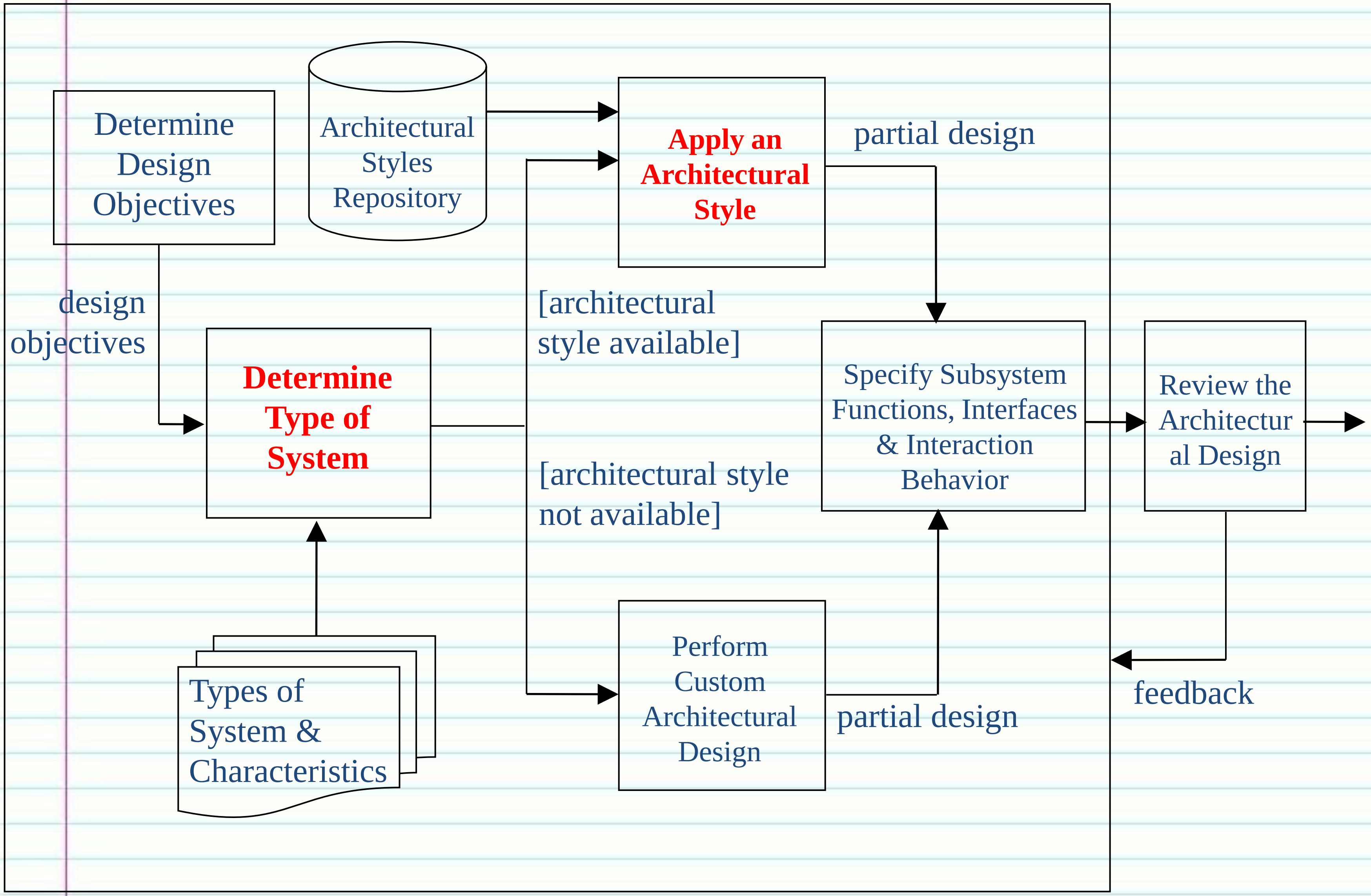
What Is Software Architectural Design

- The *software architecture* of a system or subsystem refers to the style of design of the structure of the system including the interfacing and interaction among its major components.
- Software architectural design is a decision-making process to determine the software architecture for the system under development.
- Or according to IEEE 1471-2000 Recommended Practice for Architectural Description of Software Intensive Systems
- Software architecture is the fundamental organization of a system, embodied in:
  – The components of the system
  – Relationships among the components
  – Relationships between the components and the environment
  – Principles governing the design and evolution of the system

Importance of Architectural Design

- The architecture of a software system has significant impact on system performance, efficiency, security, and maintainability.
- It is a primary artifact for conceptualization, constructing, managing, and evolving the system under development.
- Architectural design flaws may result in project failure.

Architectural Design Process


Determine Design Objectives
Architectural Styles Repository
Apply an Architectural Style
partial design
design objectives
[architectural style available]
Determine Type of System
Specify Subsystem Functions, Interfaces & Interaction Behavior
Review the Architectural Design
[architectural style not available]
Types of System & Characteristics
Perform Custom Architectural Design
partial design
feedback

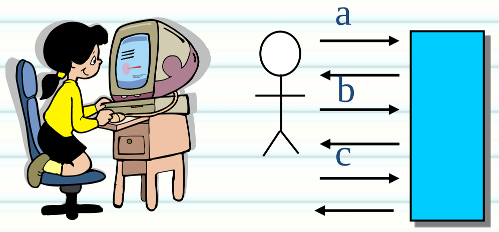
Guidelines for Architectural Design

1. Adapt an architectural style when possible.
2. Apply software design principles.
3. Apply design patterns.
4. Check against design objectives and design principles.
5. Iterate the steps if needed.

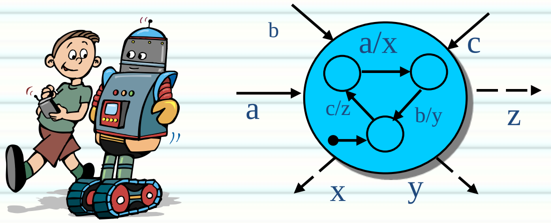
Architectural Design Considerations

- Ease of change and maintenance.
- Use of commercial off-the-shelf (COTS) parts.
- System performance – does the system require to process real-time data or a huge volume of transactions?
- Reliability.
- Security.
- Software fault tolerance.
- Recovery.

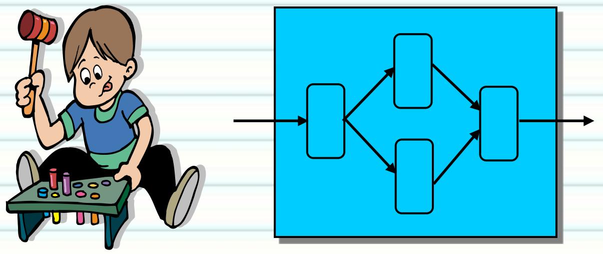
Four Common Types of Systems



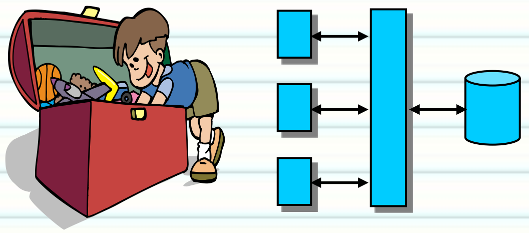
(a) Interactive subsystem



(b) Event-driven subsystem



(c) Transformational subsystem



(d) Database subsystem

Characteristics of Interactive Systems

- The interaction between system and actor consists of a relatively fixed sequence of actor requests and system responses.
- The system has to process and respond to each request.
- Often, the system interacts with only one actor during the process of a use case.
- The actor is often a human being although it can also be a device or another subsystem.
- The interaction begins and ends with the actor.
- The actor and the system exhibit a “client-server” relationship.
- System state reflects the progress of the business process represented by the use case.

Characteristics of Event-Driven Systems

- It receives events from, and controls external entities.
- It does not have a fixed sequence of incoming requests; requests arrive at the system randomly.
- It does not need to respond to every incoming event. Its response is state dependent—the same event may result in different responses depending on system state.
- It interacts with more than one external entity at the same time.
- External entities are often hardware devices or software components rather than human beings.
- Its state may not reflect the progress of a computation.
- It may need to meet timing constraints, temporal constraints, and timed temporal constraints.

Characteristics of Transformational Systems

- Transformational systems consist of a network of information-processing activities, transforming activity input to activity output.
- Activities may involve control flows that exhibit sequencing, conditional branching, parallel threads, synchronous and asynchronous behavior.
- During the transformation of the input into the output, there is little or no interaction between system and actor—it is a batch process.
- Transformational systems are usually stateless.
- Transformational systems may perform number crunching or computation intensive algorithms.
- The actors can be human beings, devices, or other systems.

Characteristics of Object-Persistence Systems

- It provides capabilities for storing and retrieving objects from a database or file system while hiding the storage media
- It provides object storage and retrieval capabilities to other subsystems.
- It hides the implementation from the rest of the system.
- It is responsible only for storing and retrieving objects, and does little or no business processing except performance considerations.
- It is capable of efficient storage, retrieval, and updating of a huge amount of structured and complex data.

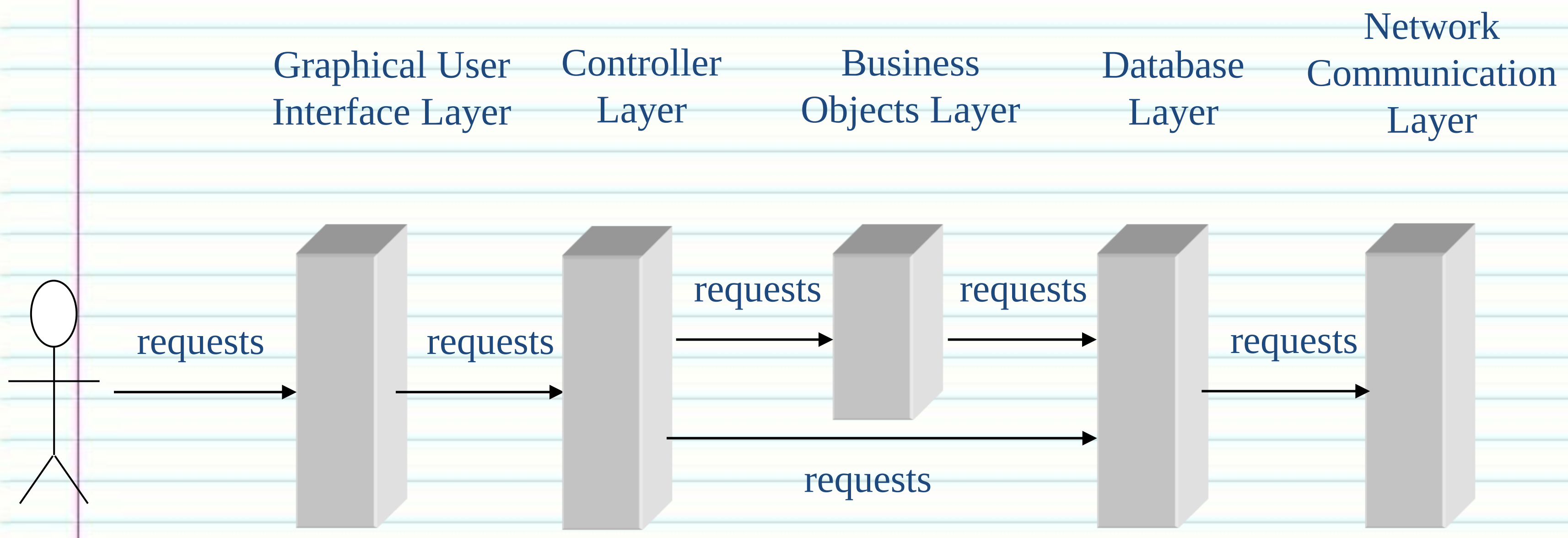
Recursive View of Systems and Subsystems

- Systems and subsystems are relative to each other - a system is a subsystem of a larger system, a subsystem is a system and may consist of other subsystems.
- A system may consist of different types of subsystems.
- The design method applied needs to match the type of subsystem under development.

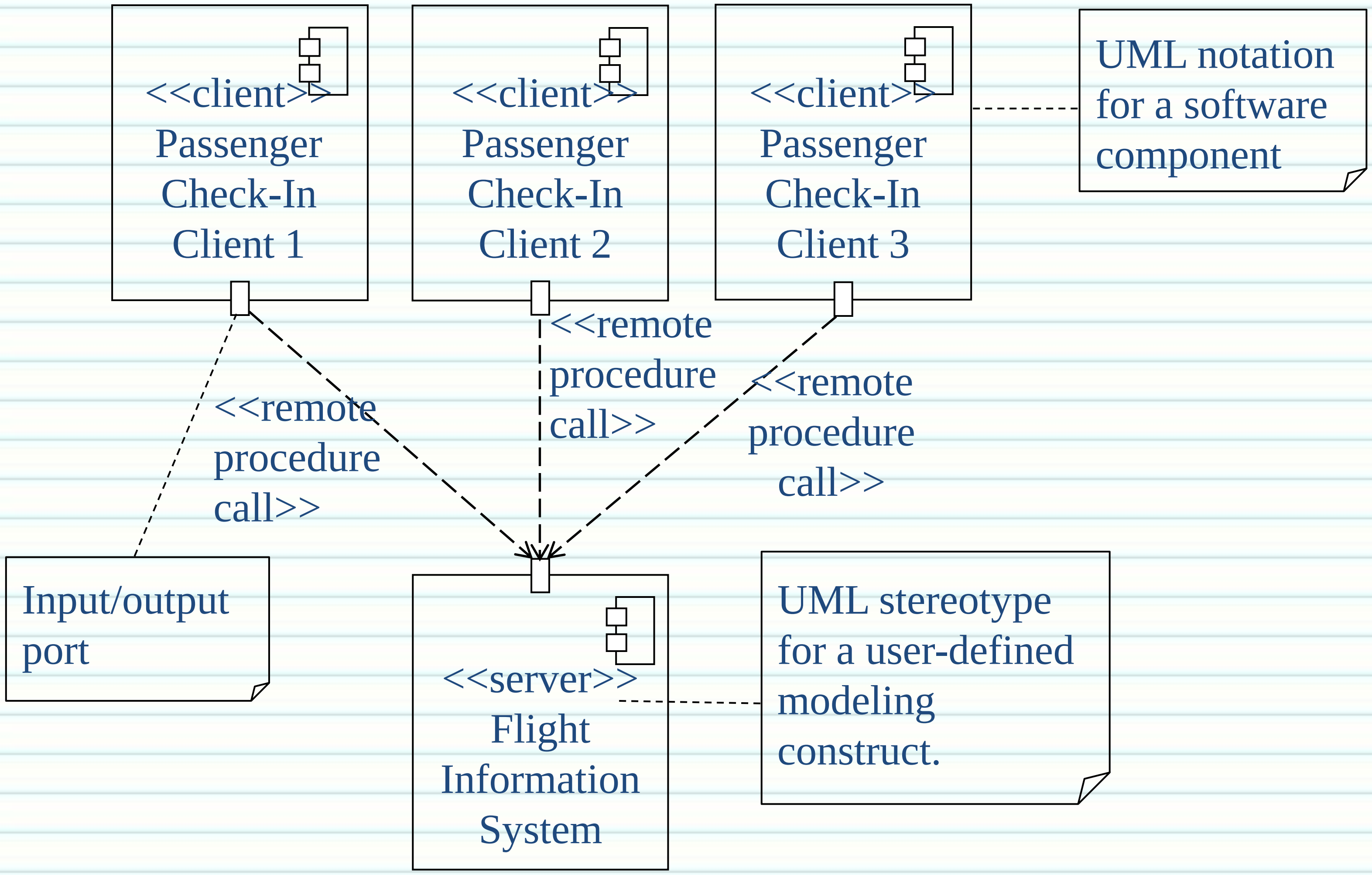
System Types and Architectural Styles

Type of System	Architectural Style
Interactive System	N-Tier
Event-Driven System	Event-Driven
Transformational System	Main Program and Subroutines
Object-Persistence Subsystem	Persistence Framework
Client-server	Client-server
Distributed, decentralized	Peer-to-peer
Heuristic problem-solving	Blackboard

N-Tier Architecture

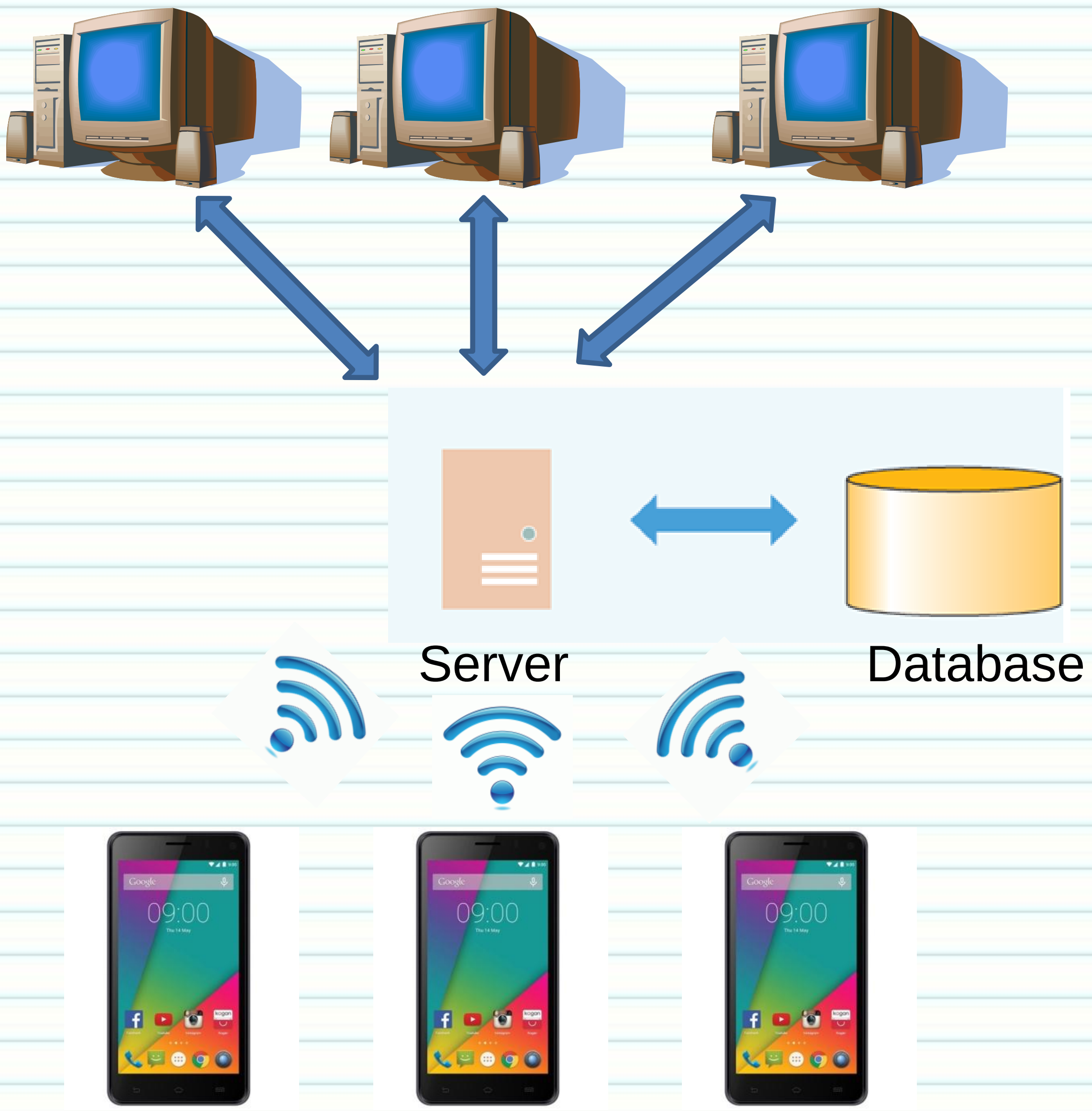


Client-Server Architecture



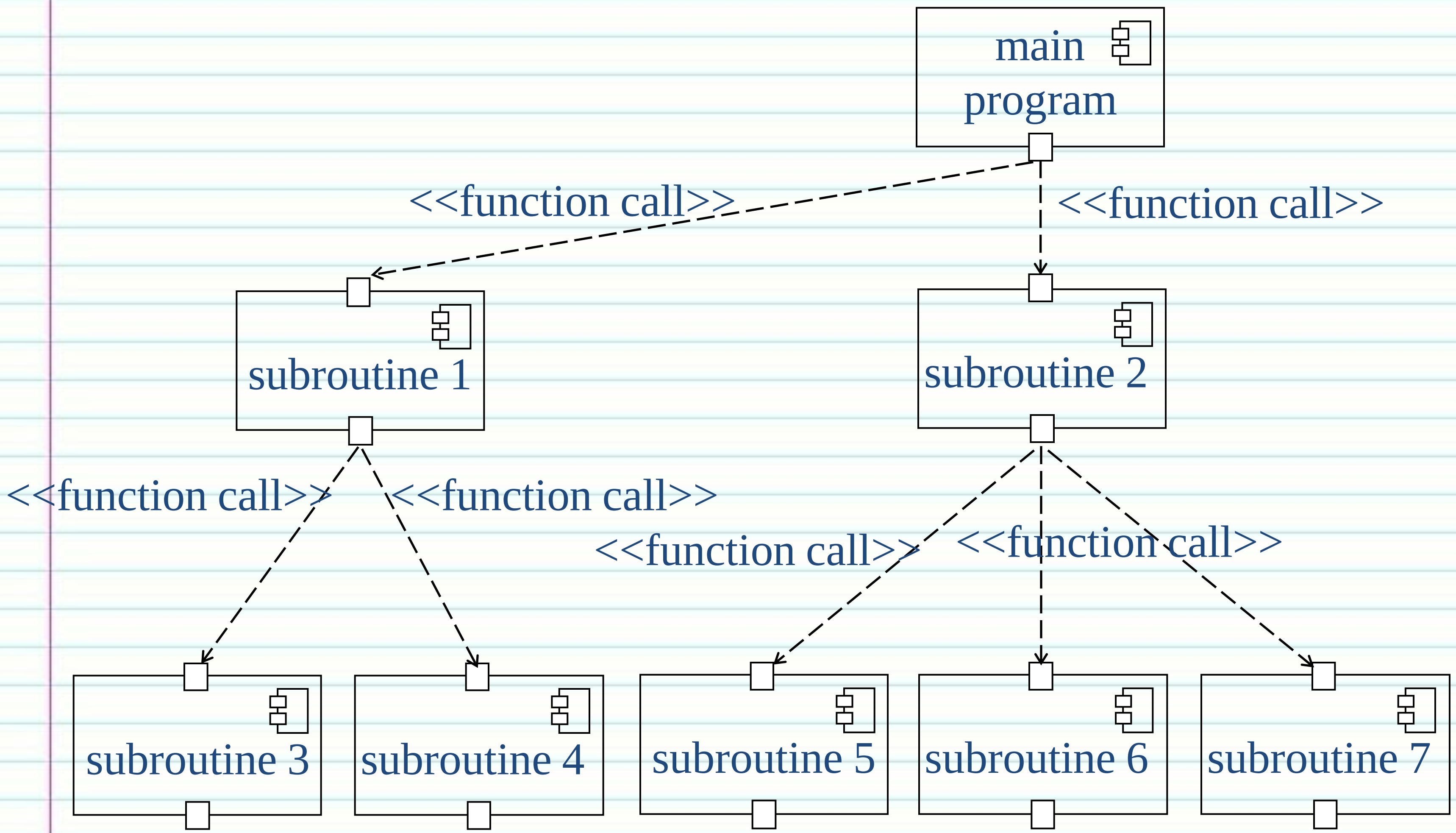
<<client>> Passenger Check-In Client 1
<<client>> Passenger Check-In Client 2
<<client>> Passenger Check-In Client 3
UML notation for a software component
<<remote procedure call>>
<<remote procedure call>>
<<remote procedure call>>
Input/output port
<<server>> Flight Information System
UML stereotype for a user-defined modeling construct.

Mobile Architecture

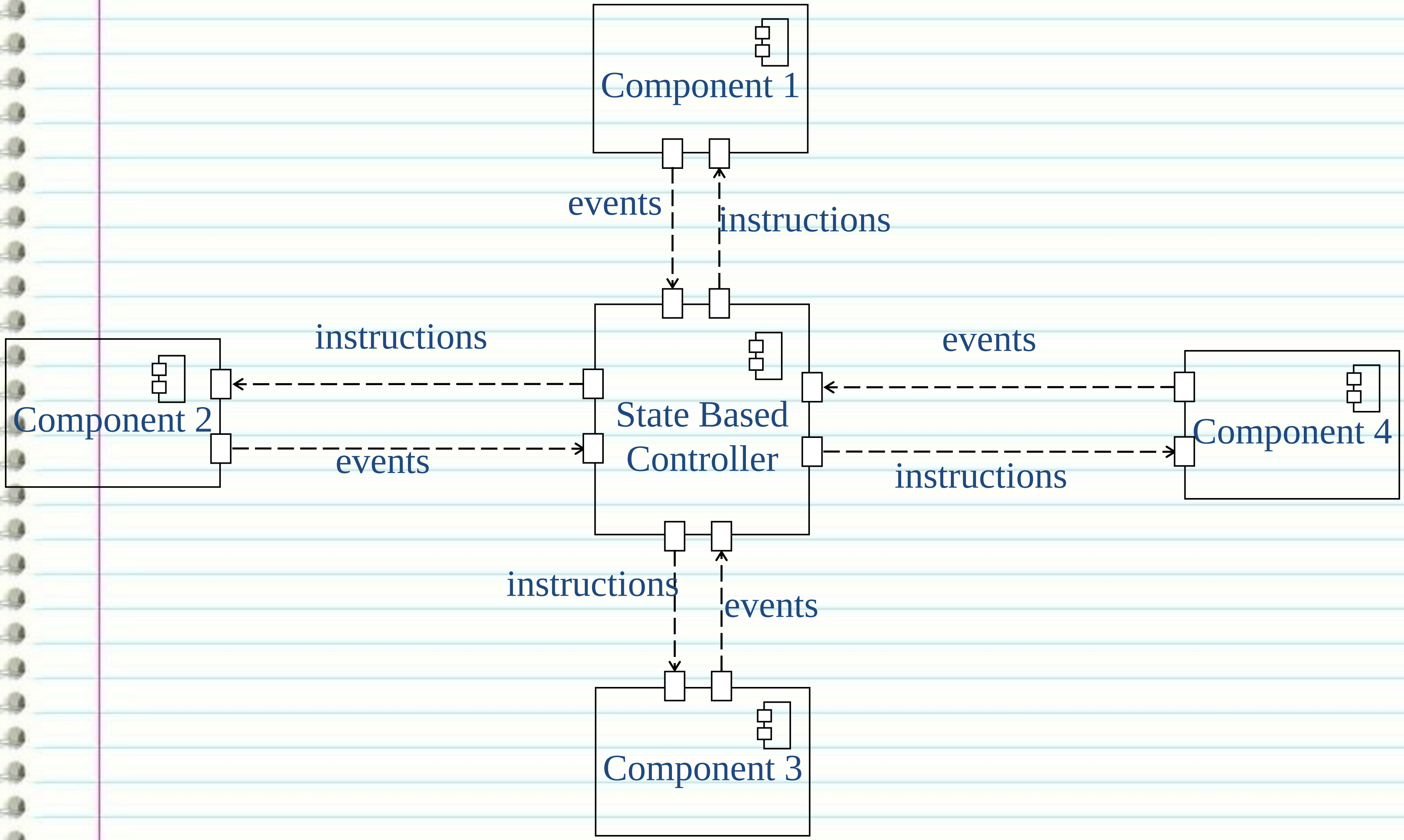


There are many possible solutions here

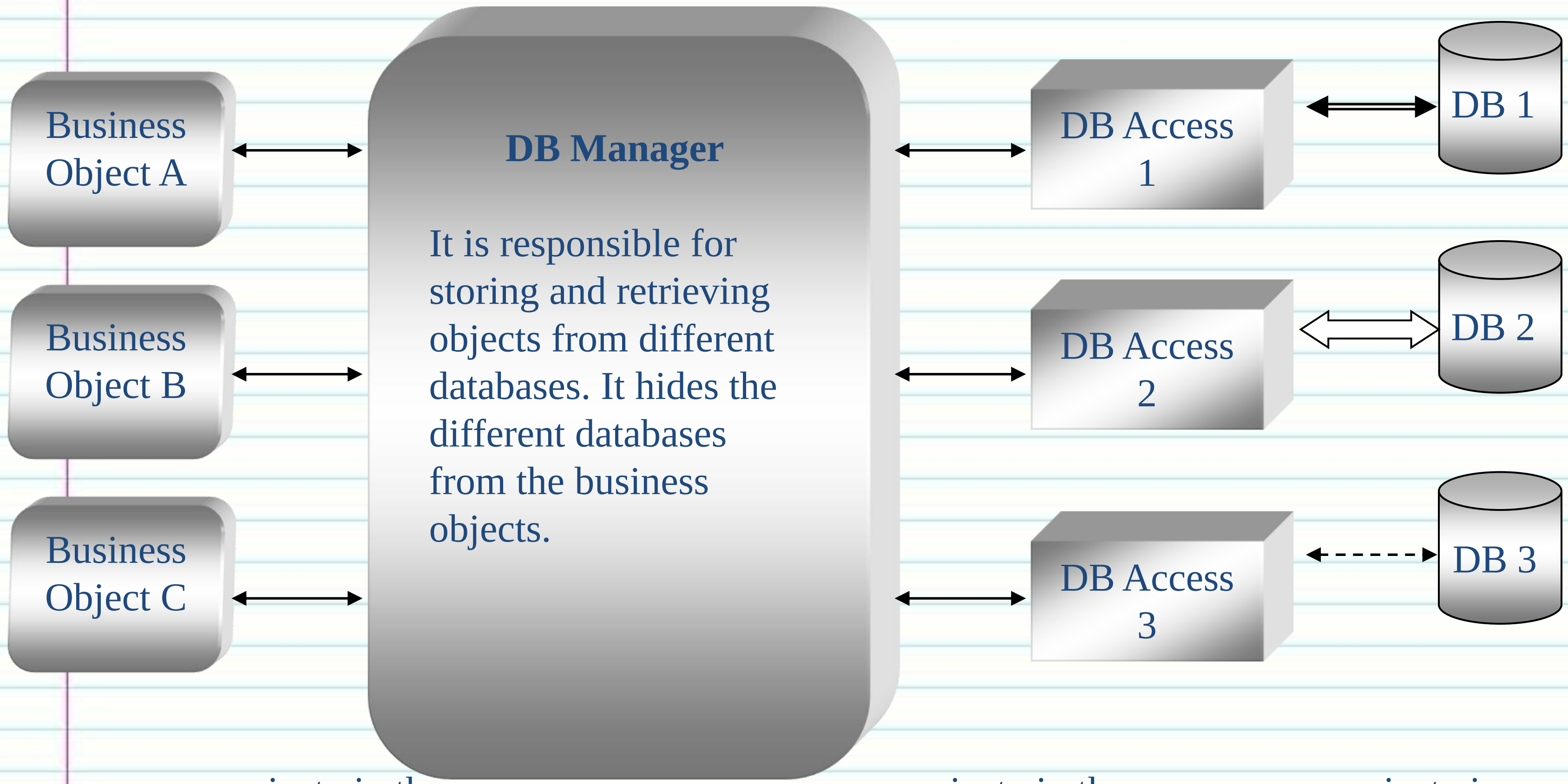
Main Program and Subroutine Architecture



Event-Driven Architecture



Object-Persistence Framework

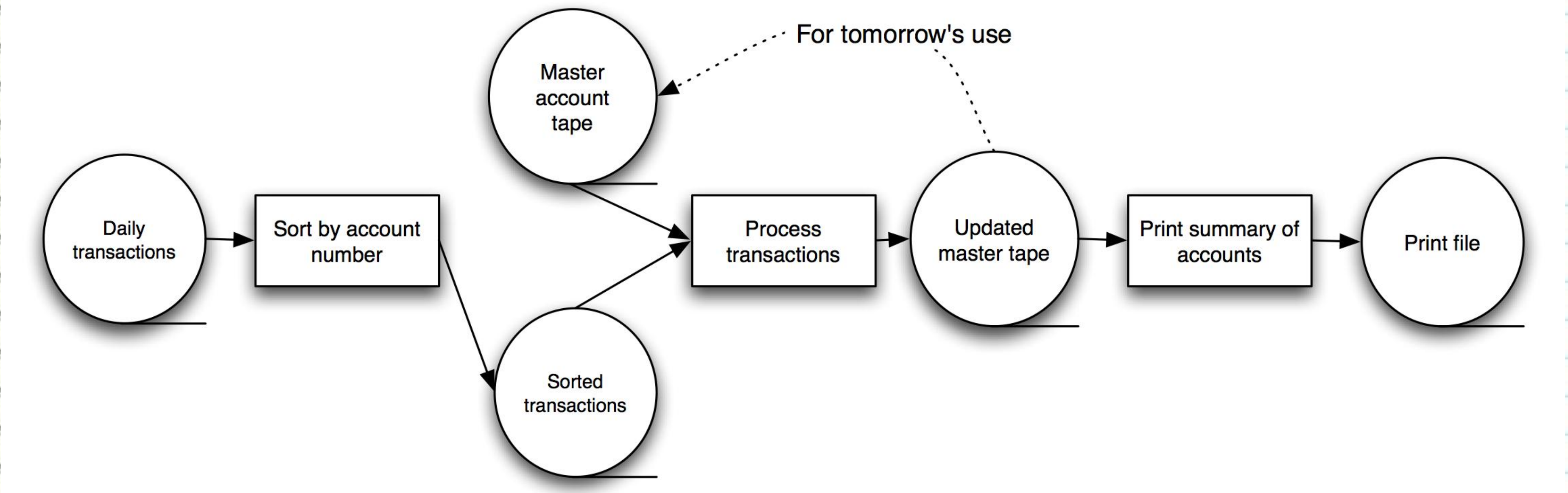

Business Object A
Business Object B
Business Object C
DB Manager
It is responsible for storing and retrieving objects from different databases. It hides the different databases from the business objects.
DB Access 1
DB Access 2
DB Access 3
DB 1
DB 2
DB 3

communicate in the object-oriented implementation language

communicate in the object-oriented implementation language

communicate in a DBMS specific language

Batch-Sequential: A Financial Application



Pipe and Filter Style

- Components are filters
  - Transform input data streams into output data streams
  - Possibly incremental production of output
- Connectors are pipes
  - Conduits for data streams
- Style invariants
  - Filters are independent (no shared state)
  - Filter has no knowledge of up- or down-stream filters
- Examples
  - UNIX shell signal processing
  - Distributed systems parallel programming
- Example: `ls invoices | grep -e August | sort`

Blackboard Style

- Two kinds of components
  - Central data structure — blackboard
  - Components operating on the blackboard
- System control is entirely driven by the blackboard state
- Examples
  - Typically used for AI systems
  - Integrated software environments (e.g., Interlisp)

Rule-Based Style

Inference engine parses user input and determines whether it is a fact/rule or a query. If it is a fact/rule, it adds this entry to the knowledge base. Otherwise, it queries the knowledge base for applicable rules and attempts to resolve the query.

- Components: User interface, inference engine, knowledge base
- Connectors: Components are tightly interconnected, with direct procedure calls and/or shared memory.
- Data Elements: Facts and queries
- Behavior of the application can be very easily modified through addition or deletion of rules from the knowledge base.
- Caution: When a large number of rules are involved understanding the interactions between multiple rules affected by the same facts can become *very* difficult.

Publish-Subscribe

Subscribers register/deregister to receive specific messages or specific content. Publishers broadcast messages to subscribers either synchronously or asynchronously.

- Components: Publishers, subscribers, proxies for managing distribution
- Connectors: Typically a network protocol is required. Content-based subscription requires sophisticated connectors.
- Data Elements: Subscriptions, notifications, published information
- Topology: Subscribers connect to publishers either directly or may receive notifications via a network protocol from intermediaries
- Qualities yielded Highly efficient one-way dissemination of information with very low-coupling of components

Peer-to-Peer Style

- State and behavior are distributed among peers which can act as either clients or servers.
- Peers: independent components, having their own state and control thread.
- Connectors: Network protocols, often custom.
- Data Elements: Network messages
- Topology: Network (may have redundant connections between peers); can vary arbitrarily and dynamically
- Supports decentralized computing with flow of control and resources distributed among peers. Highly robust in the face of failure of any given node. Scalable in terms of access to resources and computing power. But caution on the protocol!

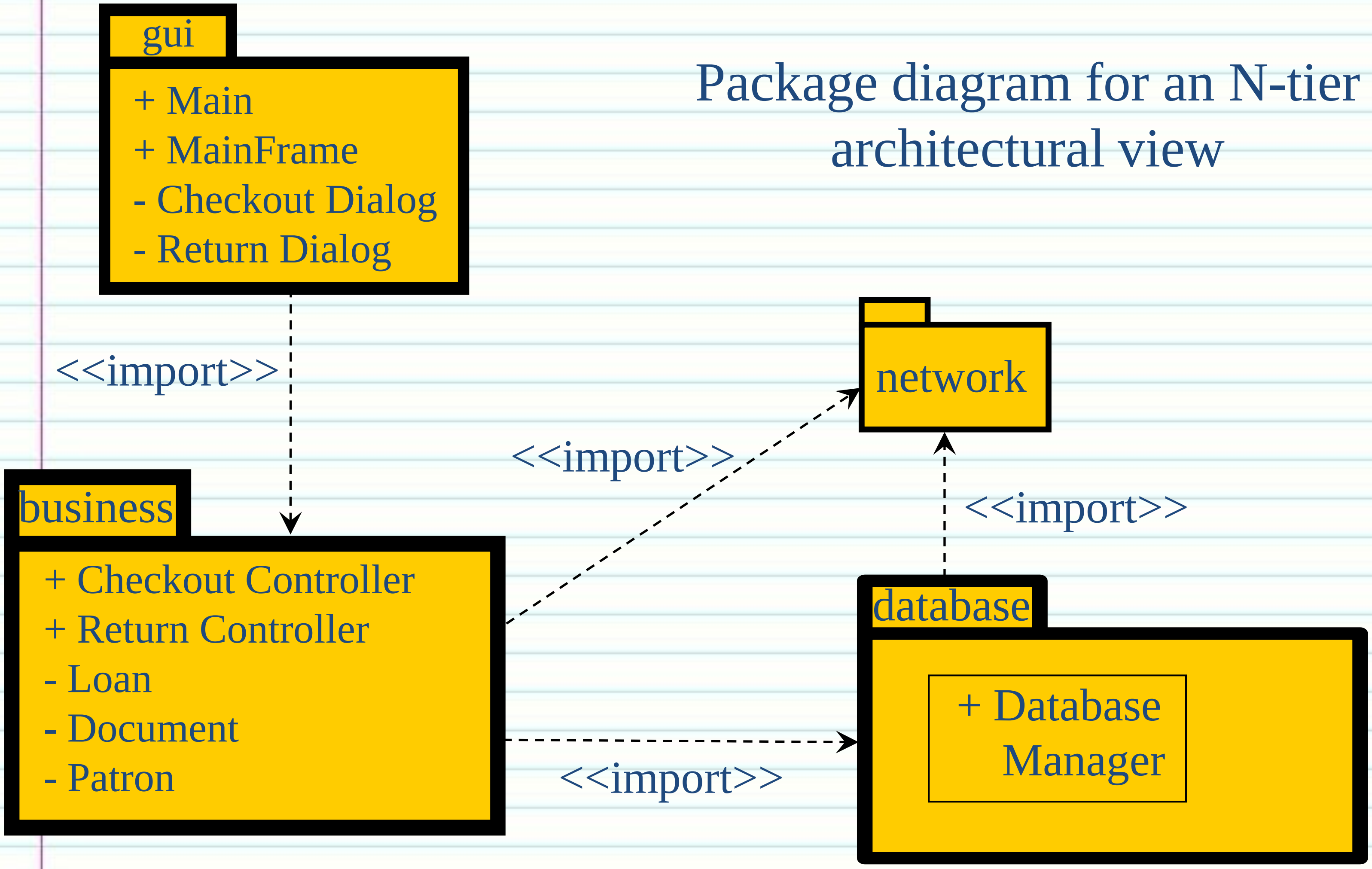
Perform Custom Architectural Design

- Not all application systems development projects can reuse an existing architectural style.
- Custom architectural design is required to meet the needs of an application system.
- Design patterns are useful for custom architectural design.

Review the Architectural Design

- Review to ensure that the requirements and architectural design objectives are met.
- Review to ensure that the architectural design satisfies software design principles.
- Review to ensure that the architectural design satisfies security requirements, constraints, and secure software design principles.

Architectural Style and Package Diagram



Package diagram for an N-tier architectural view

Legend: + public - private

Applying Software Design Principles

- Design for Change – design with a "built-in mechanism" to adapt to, or facilitate anticipated changes.
- Separation of Concerns – focusing on one aspect of the problem in isolation rather than tackling all aspects simultaneously.
- Information Hiding – shielding implementation detail of a module to reduce its change impact to other parts of the software system.
- High Cohesion – achieving a higher degree of relevance of the functions of a module to the module's core functionality.
- Low Coupling – reducing the run-time effect and change impact of a subsystem to other subsystems.
- Keep It Simple and Stupid – designing "stupid objects."

Guidelines for Architectural Design

1. Adapt an architectural style when possible.
2. Apply software design principles.
3. Apply design patterns.
4. Check against design objectives and design principles.
5. Iterate the steps if needed.

Applying Agile Principles

1. *Value working software over comprehensive documentation.*
2. *Apply the 20/80 rule - that is, good enough is enough.*