15-413

Lecture Notes on System Design

Bernd Bruegge

School of Computer Science

Carnegie Mellon University

Pittsburgh PA 15213

8 & 13 October 1998

Outline of the Lecture

- Odds and Ends:
  - **Requirements Analysis Tasks**
  - **RAD Submission Process**
  - **Project Calendar for October**
  - **Presentations**
- System Design

Collaborative Requirements Analysis

- The PAID system is a collection of applications and services
- Team level analysis of applications and services are provided by User Interface, Network, Authentication, Learning and Database.
- Each team writes a requirements analysis document from the perspective of their subsystem.
- The architecture team and the documentation team are responsible for the RAD integration and documentation.

RAD Schedule

- ✓ 9/15 Release of Template (project management)
  - ✓ **Each team works on the RAD for their subsystem**
- ✓ 9/30 Team RADs are due (teams)
  - ✓ **Project management reviews the team RADs**
- ✓ 10/5 Reviews are due (project management)
  - → **Teams incorporate review comments**
- ❖ 10/10 Second revision of team RADs due (teams)
- ❖ 10/15 Integrated version of RAD (architecture team, documentation team)

RAD Model Integration

- All models must be submitted in Together/J.
- Each model description must follow the model documentation standard
  - **UML Notation**
  - **Navigation text for each model**
- Task of the Architecture and Documentation team is to integrate the individual system modules and identify problems with consistency, completeness and ambiguities. If these problems are not solvable during the integration they *will be discussed and identified during the analysis review (Oct 22 & 27)*

Typical RAD Problems

- Consistency problems:
  - **Different signatures by method provider and method user**
- Completeness problems:
  - **Dangling associations (associations pointing to nowhere)**
  - **Double defined classes**
  - **Missing classes (imported by one module but not defined anywhere)**
- Ambiguity problems:
  - **Entities referred to in more than one of the models are not identical**
  - **Names are spelled differently in different models or within the same model**
  - **Classes in different subsystems have the same name but different meanings**

Presentation Schedule

- Presentations
  - **Analysis Review: Oct 22 & 27, 9:00-10:20am**
  - **System Design Review: Nov 11, 9:00-10:20am**
  - **Object Design & Implementation Review: Nov 24**
  - **Client Acceptance Test: Dec 10 (precise date to be announced)**
- Speakers:
  - **5-6 Speakers per Review**
  - **15-20 minute presentations**
  - **Every student will have to give one presentation**
  - **Scheduling and elicitation of speakers on the announce bboard**

Analysis Review Agenda

- Review will be run as a meeting
- Primary facilitator & Minute taker (2 Coaches)
- Project Management (1 Speaker, 15 minutes)
  - **Problem statement and global requirements**
  - **Project Management issues & Process model for PAID Project**
- Subsystem presentations (5 Speakers, 15 minutes each)
  - **Go through one or two use cases**
  - **Show object diagram for subsystem (class hierarchy if it exists)**
  - **Present illustrative prototype for user interface**
  - **Present state diagrams for objects with important dynamic behavior**
- Discussion (15 minutes)
- Identification of Action Items (5 minutes)

Project Calendar

October 1998

Thu	Fri	Sat	Sun	Mon	Tue	Wed
1	2	3	4	5	6	7 **Project Mgt RAD Review Due**
8 System Design I System Design II **SDD template**	**9**	10	11	12 **Second Rev Team RADs due**	13 Design Patterns, Gamma, Ch. 1	14
15 Design Patterns II	16	17	18	19 **RAD Integration Due**	20 Prototyping	21 **RAD Review Presentation Deadline**
22 Analysis Review	23	24	25	26	27 Analysis Review **SDD Due**	28
29 Database Management	30	31	1	2	3 **Unit Testing** **Test Manual Template Out**	4

Advice for presentations

- Present the global picture
- Project Management
  - **Is our communication structure working?**
    - **How is the communication with the client?**
    - **Communication with management?**
    - **Intergroup communication**
    - **How about the architecture meetings, project meetings?**
    - **How about the bulletin boards?**
  - **What can be improved?**
  - **Anything wrong with the phases (Planning, Analysis, Design, Implementation,....)?**
  - **Is iteration working?**
  - **Is our incremental process working?**

Presentation of PAID Subsystems

- Services (Sets of methods provided by PAID subsystems)
  - **Services provided by your subsystem**
  - **Services needed by other subsystems**
  - **Any mismatch between "needed from others" and "provided by you"?**
  - **Services needed from other subsystems but not yet provided by them**
- Analysis Models (Functional, Object and Dynamic Models needed to explain subsystem)
- Design and Implementation Issues
  - **Nonfunctional and constraints (pseudo requirements)**

Miscellaneous Presentation Advice

- Dry-run (send e-mail to bruegge@cs.cmu.edu if you want to schedule a meeting)
  - **Discussion of presentation techniques**
  - **Tools for presentations**
- There is no winner or looser

Design

- 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

Why is Design so Difficult?

- Analysis: Concentrates the application domain
- Design: Has to worry about the implementation domain
  - **Design knowledge is a moving target**
  - **The reasons for design decisions are changing very rapidly**
    - **Halftime knowledge in software engineering: About 3-5 years**
    - **What I teach today will be out of date in 3 years**
    - **Cost of hardware rapidly sinking**
- “Design window”:
  - **Time in which design decisions have to be made**

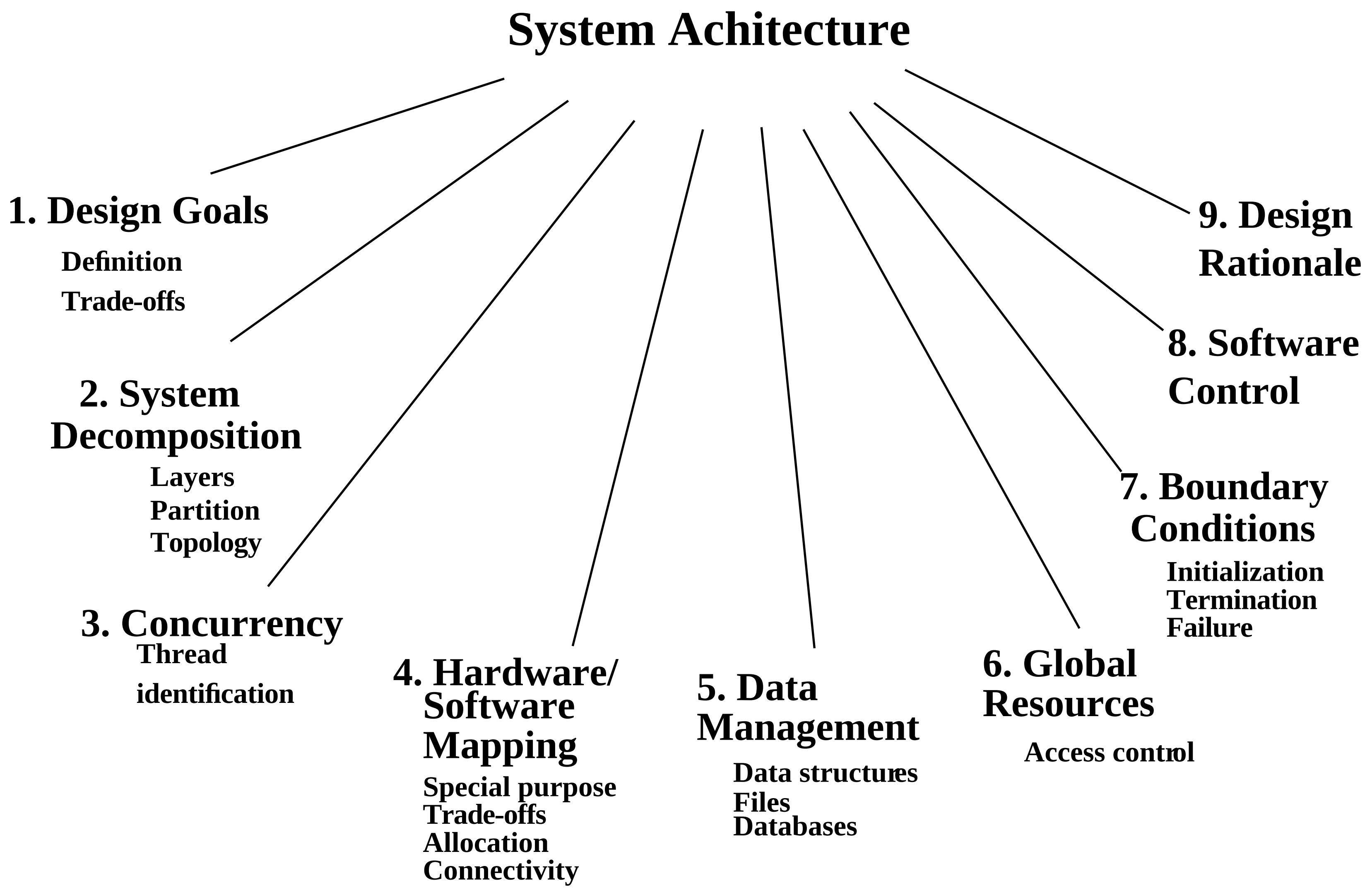
Outline of the next two classes

1. Design Goals
2. Subsystem decomposition
3. Identify concurrency
4. Allocate subsystems to processors
5. Specify management of data
6. Identify global resources and access methods
7. Define control
8. Define boundary conditions

System Architecture Patterns

System Design Template

System Design



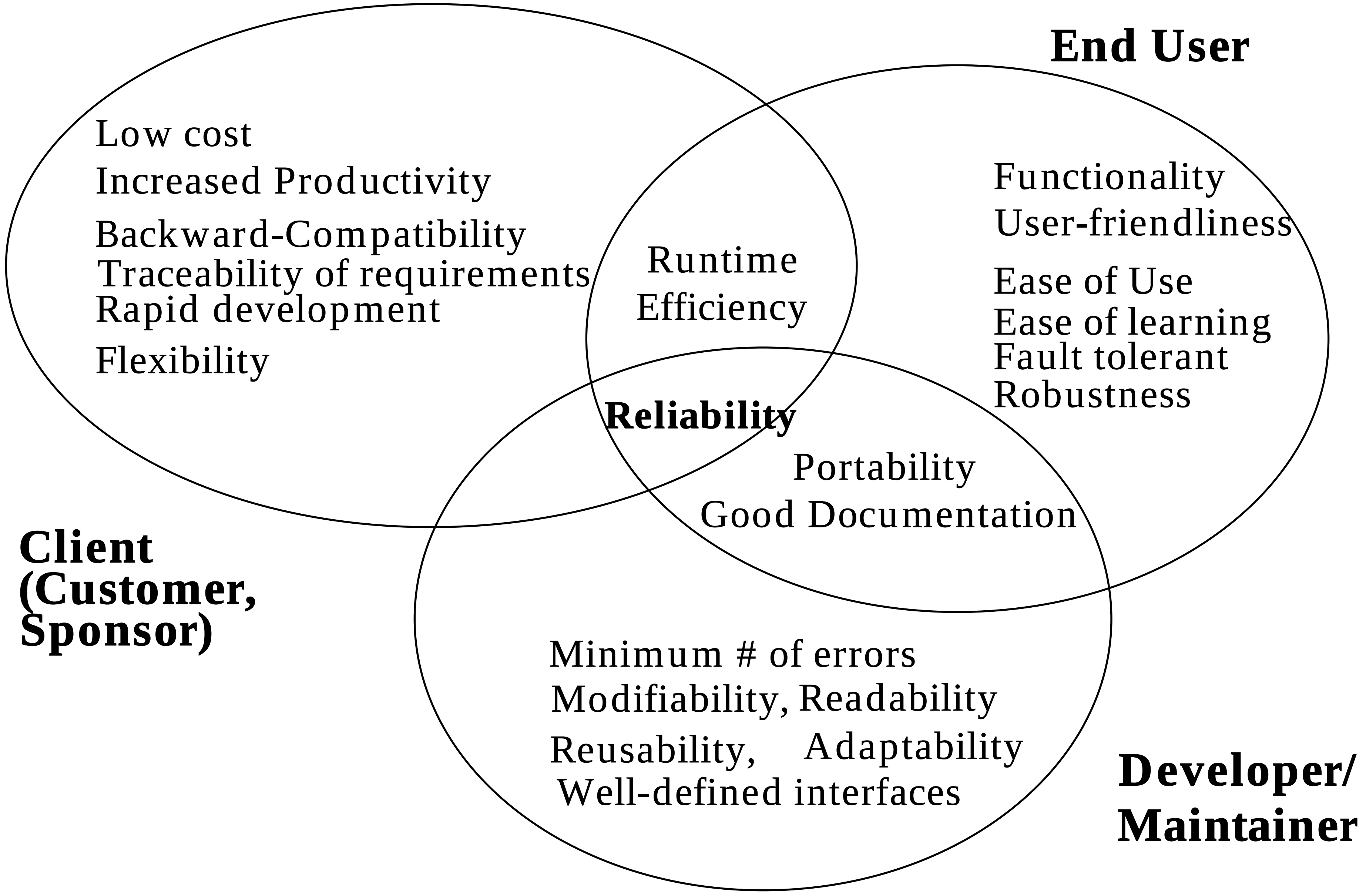
How to use the results from the Requirements Analysis for System Design

- Nonfunctional Requirements =>
  - **Chapter 1: Design Goals**
- Use Case Model =>
  - **Chapter 2: System decomposition (Based on functionality)**
- Object Model =>
  - **Chapter 4: Hardware/software mapping**
  - **Chapter 5: Data management**
- Dynamic Model =>
  - **Chapter 3: Identification of concurrency**
  - **Chapter 6: Global resource handling**
  - **Chapter 7: Software control**
  - **Chapter 8: Boundary conditions**

Design Goals
(Analysis Phase: Nonfunctional requirements)

- Reliability
- Modifiability
- Maintainability
- Understandability
- Adaptability
- Reusability
- Efficiency
- Portability
- Traceability of requirements
- Fault tolerance
- Backward-compatibility
- Cost-effectiveness
- Robustness
- High-performance
- Good documentation
- Well-defined interfaces
- User-friendliness
- Reuse of components
- Rapid development
- Minimum # of errors
- Readability
- Ease of learning
- Ease of remembering
- Ease of use
- Increased productivity
- Low-cost
- Flexibility

Relationship Between Design Goals



Typical Design Trade-offs

- Functionality vs. Usability
  - **Client vs. Developer**
- Cost vs. Robustness
  - **Client vs. User**
- Efficiency vs. Portability
  - **?**
- Rapid development vs. Functionality
  - **?**
- Cost vs. Reusability
  - **?**
- Backward Compatibility vs. Readability
  - **?**

Nonfunctional Requirements lead to Design Patterns

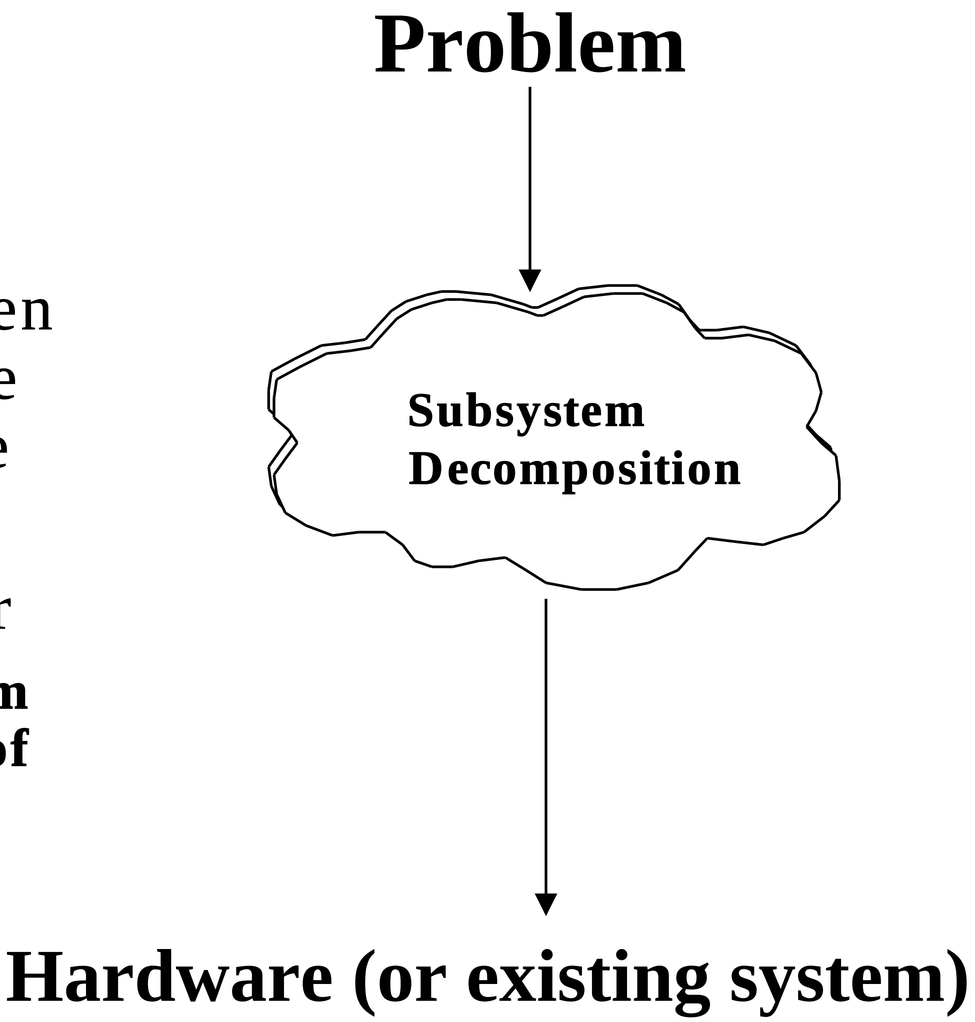
- **Read the problem statement again**
- **Use textual clues (similar to Abbot's technique in Analysis) to identify design patterns**
- *Text*: "manufacturer independent", "device independent", "must support a family of products"
  - **Abstract Factory Pattern**
- *Text*: "must interface with an existing object"
  - **Adapter Pattern**
- *Text*: "must deal with the interface to several systems, some of them to be developed in the future", " an early prototype must be demonstrated"
  - **Bridge Pattern**

Textual Clues in Nonfunctional Requirements

- *Text*: “complex structure”, “must have variable depth and width”
  - **Composite Pattern**
- *Text:* “must interface to an set of existing objects”
  - **Façade Pattern**
- *Text:* “must be location transparent”
  - **Proxy Pattern**
- *Text:* “must be extensible”, “must be scalable”
  - **Observer Pattern**
- *Text:* “must provide a policy independent from the mechanism”
  - **Strategy Pattern**

The Purpose of System Design

- Bridging the gap between the desired and available systems in a manageable way
- Use Divide and Conquer
  - **We model the new system to be developed as a set of subsystems**



2. System Decomposition

- Subsystem (UML: Package)
  - **Collection of classes, associations, operations, events and constraints that are interrelated**
  - **Seed for subsystems: UML Objects and Classes.**
- Service:
  - **Group of operations provided by the subsystem**
  - **Seed for services: Subsystem use cases**
- Service is specified by Subsystem interface:
  - **Specifies interaction and information flow from/to subsystem boundaries, but not inside the subsystem.**
  - **Should be well-defined and small.**
  - **Often called API: Application programmer's interface**
    - **Use "Service" during system design,**
    - **Use "API" during implementation**

Choosing Subsystems

- Criteria for subsystem selection: Most of the interaction should be within subsystems, rather than across subsystem boundaries.
  - **Client-Server: Does one subsystem always call the other for the service?**
  - **Peer-Peer: Which of the subsystems call each other for service?**
- Primary: What kind of service is provided by the subsystems (subsystem interface)?
- Secondary: Can the subsystems be hierarchically ordered?
- What kind of model is good for describing layers and partitions?

Partitions and Layers

- A large system is usually decomposed into subsystems using both, layers and partitions.
- **Partitions** vertically divide a system into several independent (or weakly-coupled) subsystems that provide services on the same level of abstraction.
- **A layer** is a subsystem that provides services to a higher level of abstraction

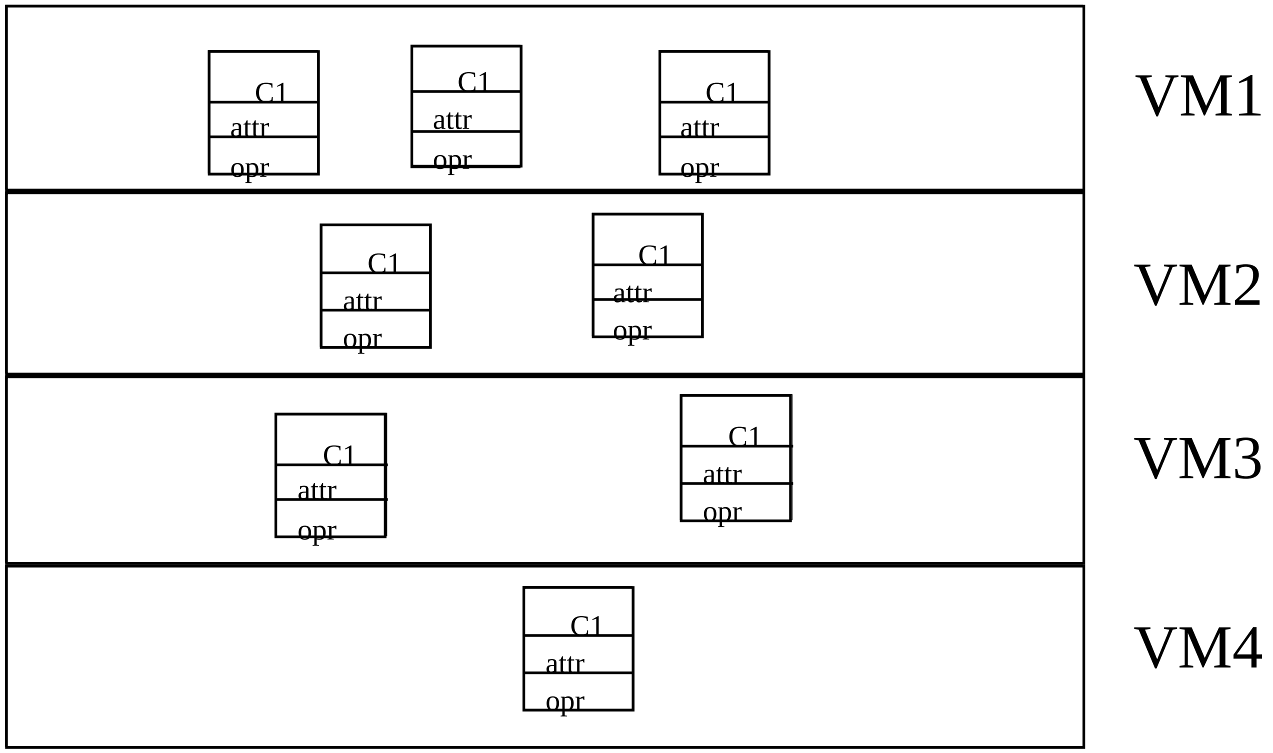
Relationships between subsystems

- Layer relationship
  - **Layer A "Calls" Layer B (runtime)**
  - **Layer A "Depends on" layer B ("make" dependency, compile time)**
- Partition relationship
  - **The subsystem have mutual but not deep knowledge about each other**
  - **Partition A "Calls" partition B and partition B "Calls" partition A**

Layering (Dijkstra, 1965)

- A system should be developed by an ordered set of virtual machines, each built in terms of the ones below it.

Problem



Existing System

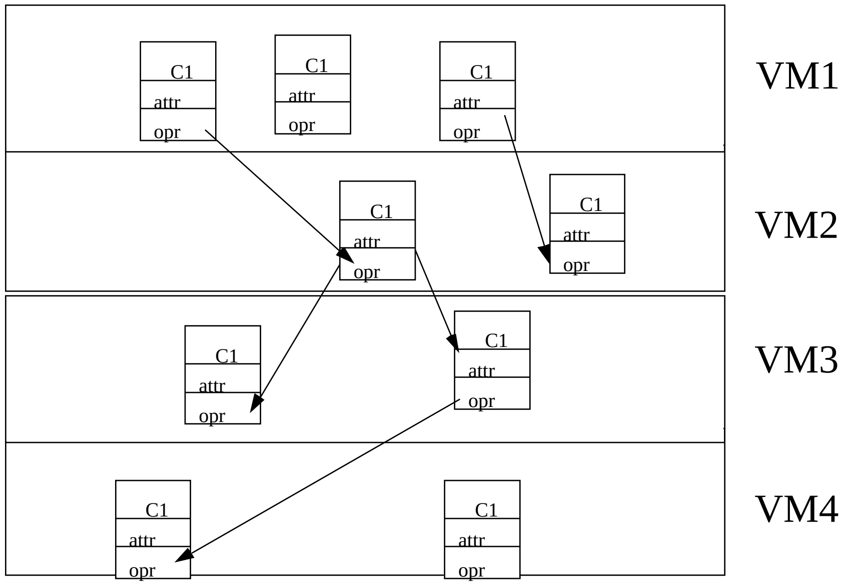
Virtual Machine

- A virtual machine is a subsystem connected to higher and lower level virtual machines by "provides services for" associations.

- A virtual machine is an abstraction that provides a set of attributes and operations.
- Virtual machines can implement two types of software architecture: closed and open architectures.

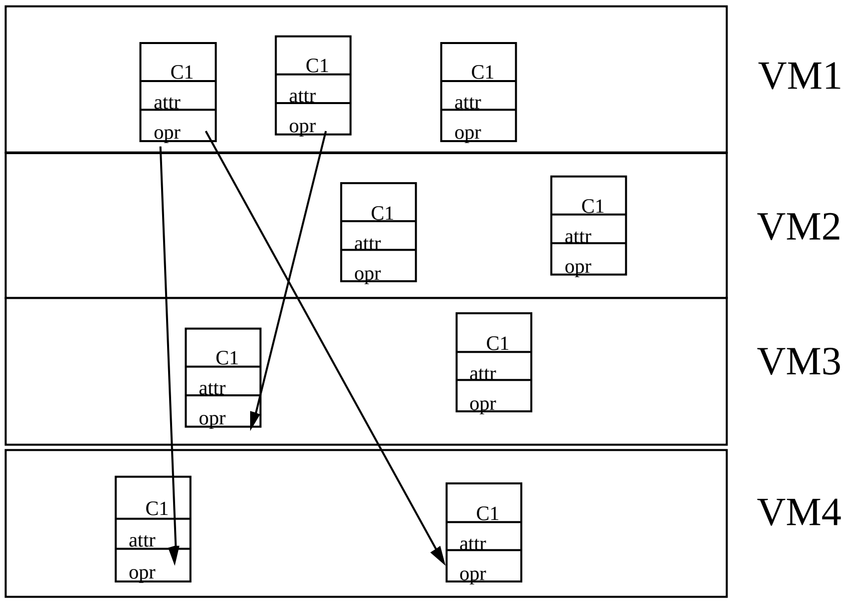
Opaque Architecture (Closed Layers)

- A virtual machine can only call operations from the layer below
- Design Goal: High Maintainability



Transparent Architecture (Open Layers)

- A virtual machine can call operations from any layers below
- Design Goal: Runtime Efficiency



Properties of Layered & Partitioned Systems

- Decomposition reduces complexity of system (divide and conquer)
- Closed architectures are more portable
- Open architectures are more efficient
- If a subsystem is a layer, it is usually called a virtual machine
- Layered systems often have chicken-and egg problem
    - **Example: Debugger opening a file while debugging a file system**

Other relationships between subsystems

- Client-Server relationship (Also called Client-Supplier)
  - **Client calls server, who performs service and returns result**
  - **Client knows the <u>interface</u> of the server ("its service")**
  - **Server does not need to know the interface of the client**
  - **Response in general immediately**
- Peer-to-Peer relationship
  - **Each of the subsystems may call on the others**
  - **Response not necessarily immediate**
  - **More complicated than client/server**
    - **Deadlocks are possible: "Communication cycles"**

Client/Server Architecture

- Often used in database systems:
  - **Frontend: User application (client)**
  - **Back end: Database access and manipulation (server)**
- Functions performed by client:
  - **Customized user interface**
  - **Front-end processing of data**
  - **Initiation of server remote procedure calls**
  - **Access to database server across the network**
- Functions performed by the database server:
  - **Centralized data management**
  - **Data integrity and database consistency**
  - **Database security**
  - **Concurrent operations (multiple user access)**
  - **Centralized processing (for example archiving)**

Design Goals for Client/Server Systems

- Portability:
  - **Server can be installed on a variety of machines and operating systems and functions in a variety of networking environments**
- Transparency:
  - **The server might itself be distributed (why?), but should provide a single "logical" service to the user**
- Performance:
  - **Client should be customized for interactive display-intensive tasks**
  - **Server should provide CPU-intensive operations**
- Expandability:
  - **Server has spare capacity to handle larger number of clients**
- Flexibility:
  - **Should be usable for a variety of user interfaces**
- Reliability
  - **System should survive individual node and/or communication link problems**

Properties of a Client/Server Architecture

- Consumer / Producer:
  - **Client is a consumer of services, server is provider of services**
- Shared resource
  - **Server often regulates access to a shared resource**
- Asymmetrical protocols
  - **Many to one association between clients and servers**
- Transparency of location
  - **Client does not need to know the location of the server**
- Mix and match
  - **Client and Servers can run on different platforms**

- Message-based
  - **Client and Servers are loosely coupled systems which interact through messages**
- Encapsulation of services
  - **The client tells the server the WHAT, the server knows HOW to do it.**
- Scalability:
  - **Horizontal scalability: Adding/Removing clients**
  - **Vertical scalability: Migrating to larger or faster servers**
- Integrity
  - **Server code and server data is centrally maintained:**
    - **Cheaper maintenance**
    - **Higher security**

Middleware

- **Definition:** The distributed software needed to support interactions between clients and servers.
  - **The Slash in Client/Server :-)**
  - **The glue that ties all clients to all servers**
- Middleware is becoming more and more a problem because of the existence of multiple servers, muliple vendors and multiple network.

Middleware design problem

- Library approach: Add new middleware software on each client whenever a new server appears
  - Requires linking of new library and recompilation of clients.
  - Clients become very large
- Server approach: Introduce a middleware service than handles all communications with the servers. (Gateway server)
  - Clients need to know only the protocol with the Gateway server
  - "Older" clients can continue to communicate with server without recompilation (as long as the gateway server is backward compatible)

Applications of Client/Server Architectures

- File Servers
  - **Client passes <u>request for file</u> over a network to the file server**
  - **Useful for *sharing files* over a network**
  - **Granularity supported by server:**
    - **File**
- Database Servers
  - **Client passes <u>query</u> to the server which finds the requested data and send them back to the client**
  - **Useful for *decision support systems* that require ad hoc queries and flexible reports**
  - **Granularity supported by server:**
    - **Table, table entry**

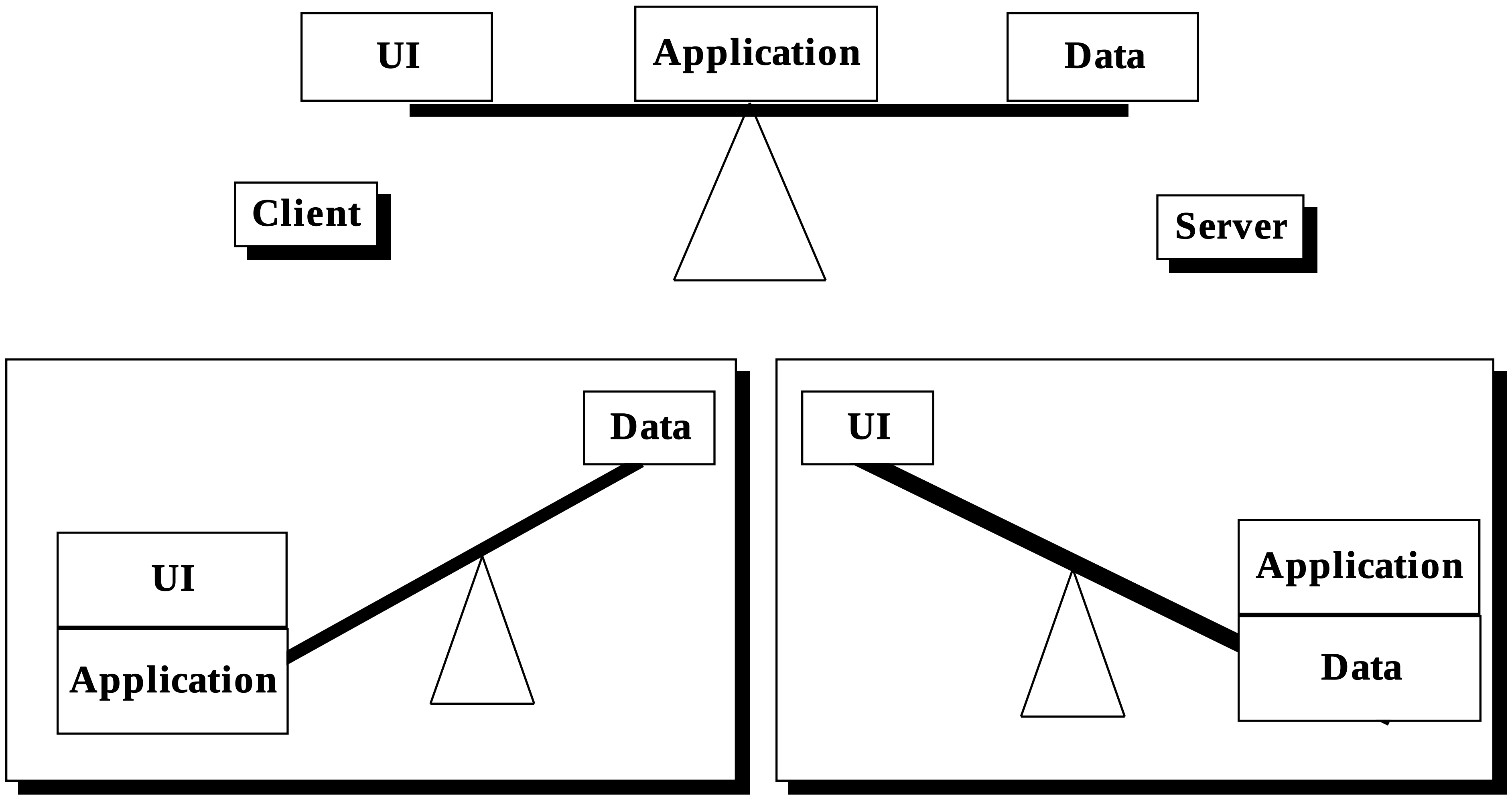
Applications of Client/Server Architectures

- Transaction Servers
  - **Clients sends a <u>transaction</u> (a series of logically connected queries) to the server which executes it locally, and sends the return result.**
  - **If one of the queries fails, the transaction fails**
  - **Useful for *online reservation systems* (airline, banking, ...)**
  - **Granularity supported by server: Table, table entry**
- Groupware Servers
  - **Client passes <u>information</u> (post, file, cursor movement) to server which stores them in a repository and notifies other clients.**
  - **Useful for *communication over space and/or time***
  - **Granularity supported by server:**
    - **Depends on groupware product (Lotus Notes: Files, Posts).**

Applications of Client/Server Architectures

- Object Servers
  - **Client invokes a <u>remote method</u> offered by a class residing on a server**
  - **Useful for *heterogenous environments* (multivendor, multinetwork, multiplatform)**
  - **Granularity supported by server:**
    - **Objects**

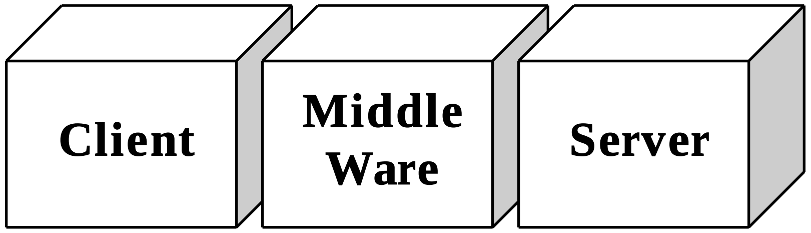
Fat Clients vs Fat Servers



Very Fat Client

Very Fat Server

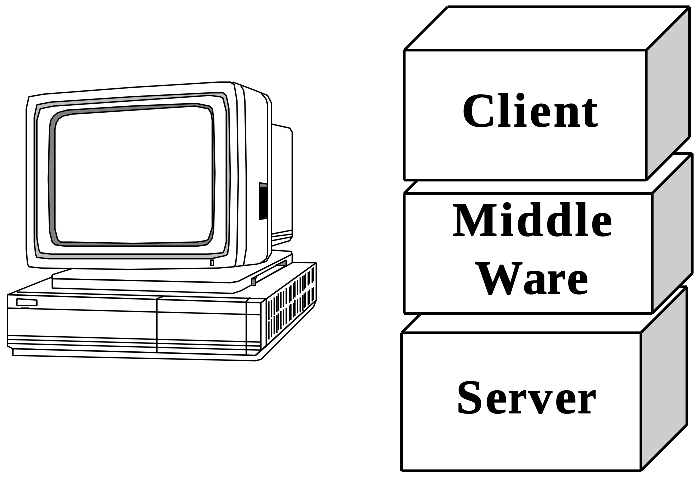
Four Types of Client/Server Configurations



- Single Machine Configuration
  - **"Client/Server for tiny shops and nomadic tribes"**
  - **Everything on a single machine**
- Single Server Configuration
  - **"Client/Server for small shops and departments"**
  - **Multiple clients/Single Server**
- Multi Server Configuration
  - **"Client/Server for global enterprises"**
  - **Multiple Clients/Multiple Servers**
- Peer-To-Peer Architecture
  - **Every machine is client and server**

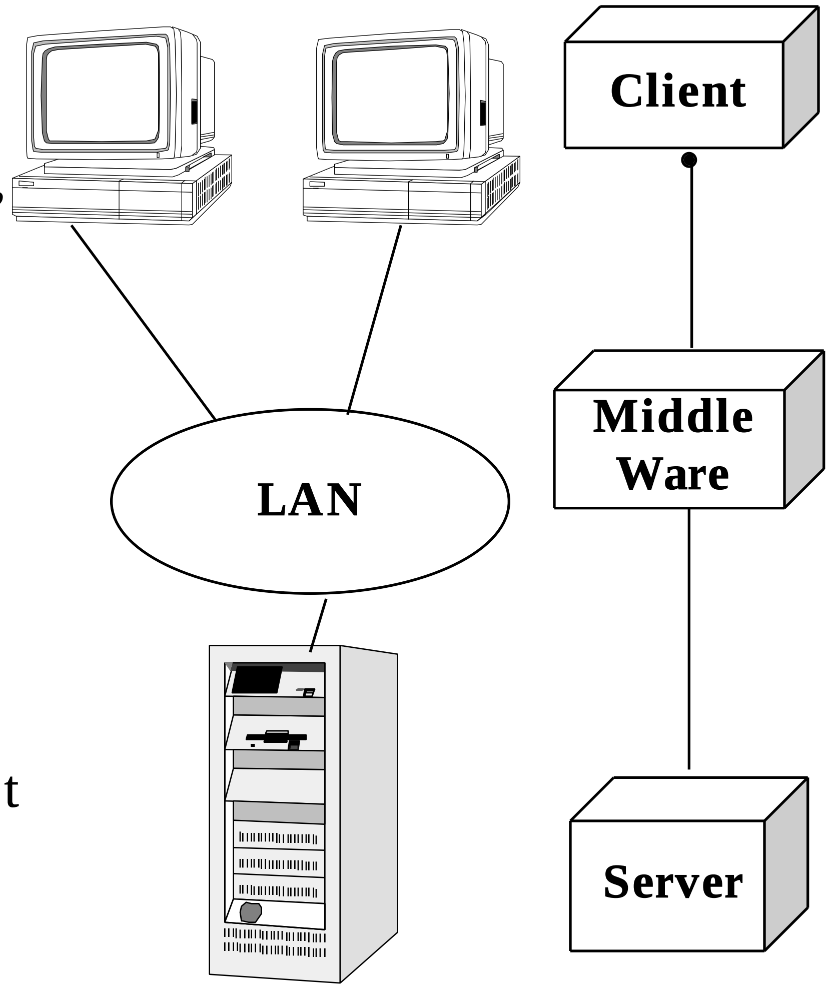
Single Machine Configuration

- "For tiny shops and nomads"
- Association: One to One
  - **One Client, One Server**
  - **Middleware: Could be as simple as Procedure Call**
- Hardware Mapping:
  - **Client and Server are running on the same machine.**
- Administration: Nonexistent problem
- Easy to use package
- Often done during system integration testing
- Problem with response times
- Needs operating system support



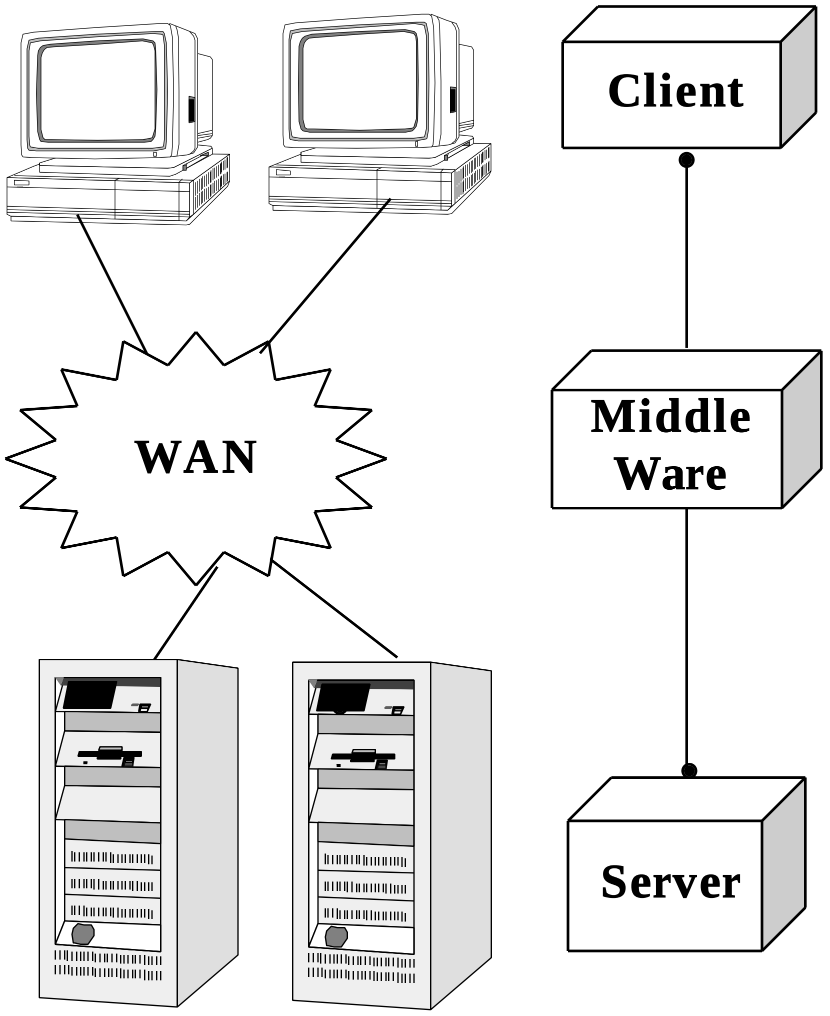
Single Server Configuration

- Client/Server for "small shops and departments"
- Assocation: Many to One:
  - **Clients: Multiple clients**
  - **Server: Single server (often a "local" server)**
  - **Middleware: Configuration file to look up server's name**
- Hardware Mapping:
  - **Clients and Servers on different machines**
- Administration: Easy (not an explicit role, everybody does it)
- Failures: Easy to identify (either client or server or wrong name in configuration file)
- 80% of today's client/server installations



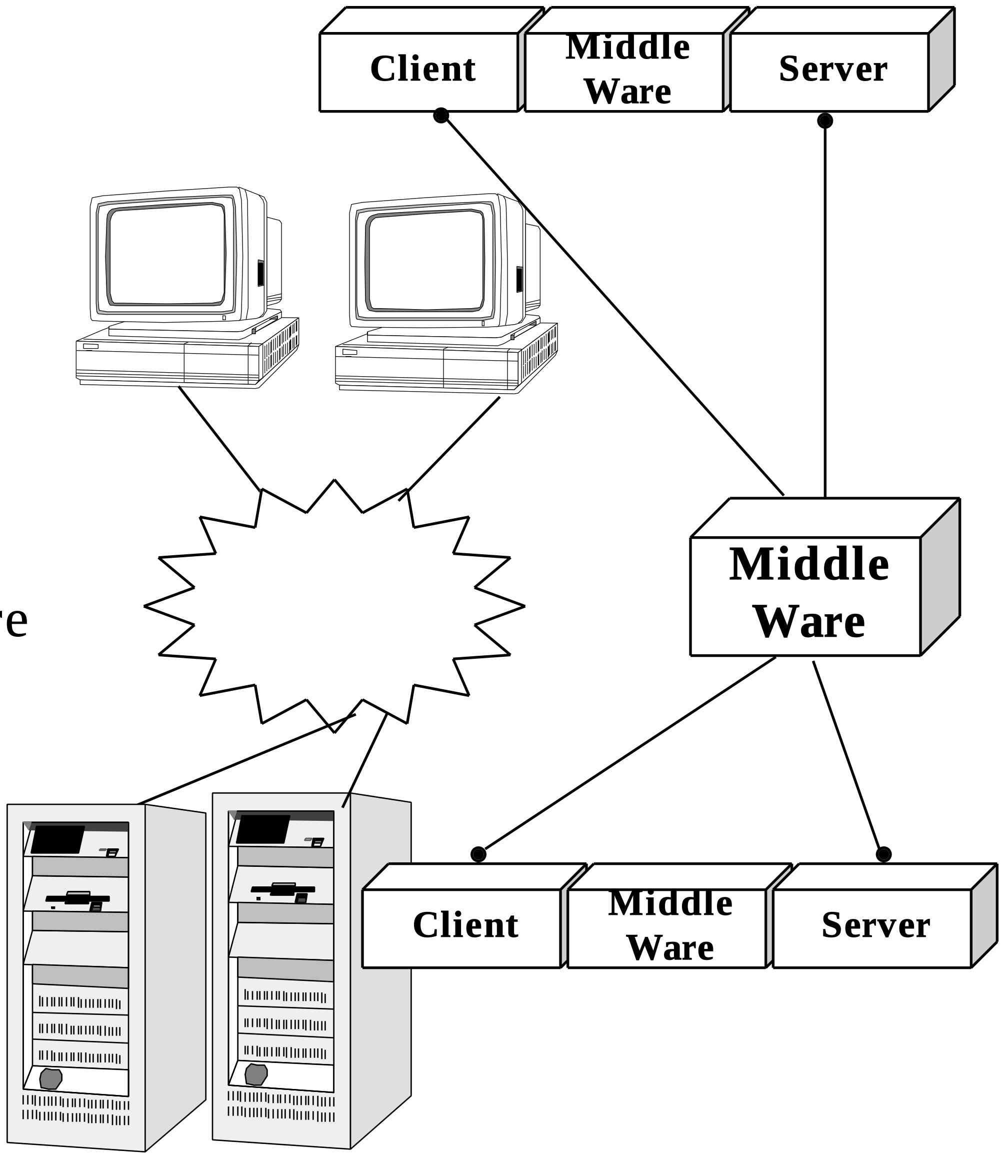
Multi Server Configuration

- For global enterprises
- Clients: Multiple clients
- Server: Mix of heterogenous servers. Partitioned because of
    - **Function they provide**
    - **Resource they control**
    - **Database they own**
    - **Fault tolerance**
    - **High performance**
- Middleware: CORBA, DCE, ODBC
- Administration: Special role
- Failures: Not always obvious



Peer-to-Peer Configuration

- Every machine is both a client and a server
- Middleware: RMI, CORBA, DCE
- Network: WAN
- Administration: Difficult and everybody has to do it
- Failures: Should be easy to identify with good middleware design

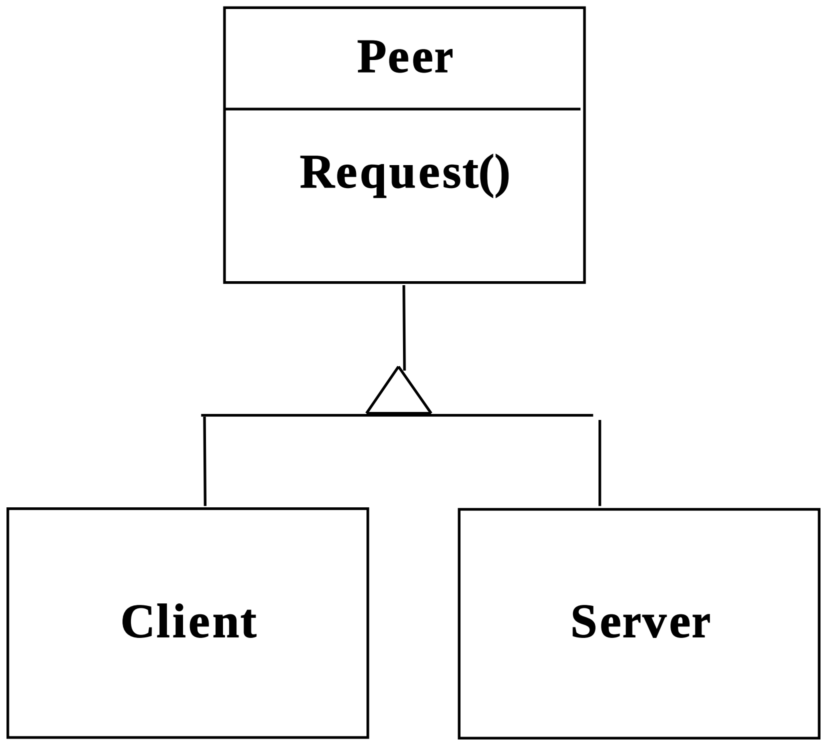


Peer-to-Peer Architectures….

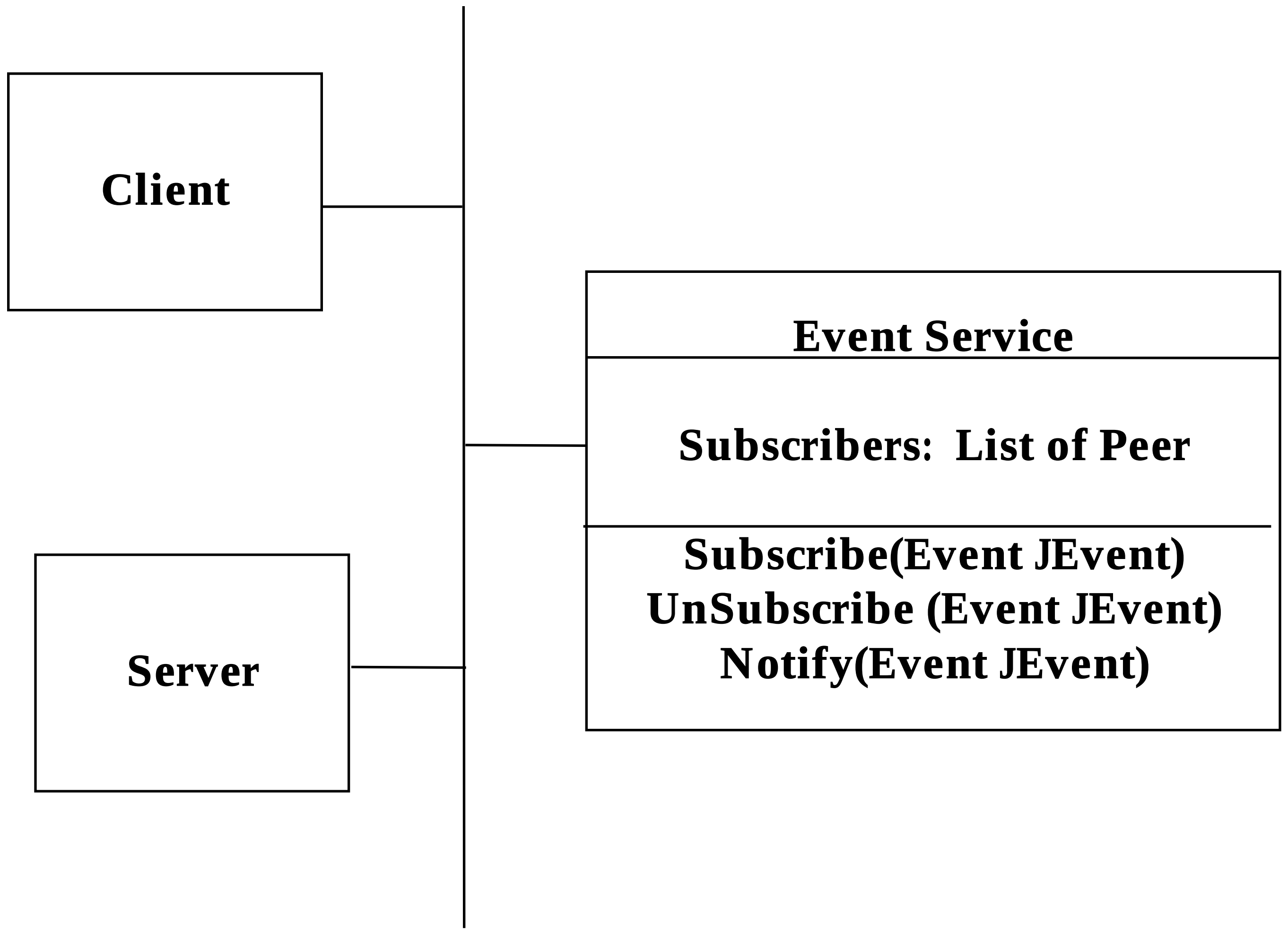
- Peer-to-peer communication is often needed
  - **Example: Database receives queries from application but also sends notifications to application when data have changed**
  - **“Server” may specifiy user interface for example, by sending a user interface to the client)**
  - **Back-end processing of data is needed (filtering, aggregation, forwarding)**
- Most important in peer-to-peer architectures: Protocol between subsystems
- PAID System Design issue:
  - **What are the "communication associations" between the subsystems of PAID? (Who calls whom?)**

Peer-to-Peer Architecture and Notification

- Provision of push and pull notification within the same software architecture
- Pull Notification
  - **The client looks for certains information by requesting it explicitly from the server.**
- Push Notification
  - **Events are modeled as an inheritance hierarchy**
  - **The client subscribes to an event**
  - **When a certain event occurs, the server notifies all subscribers**



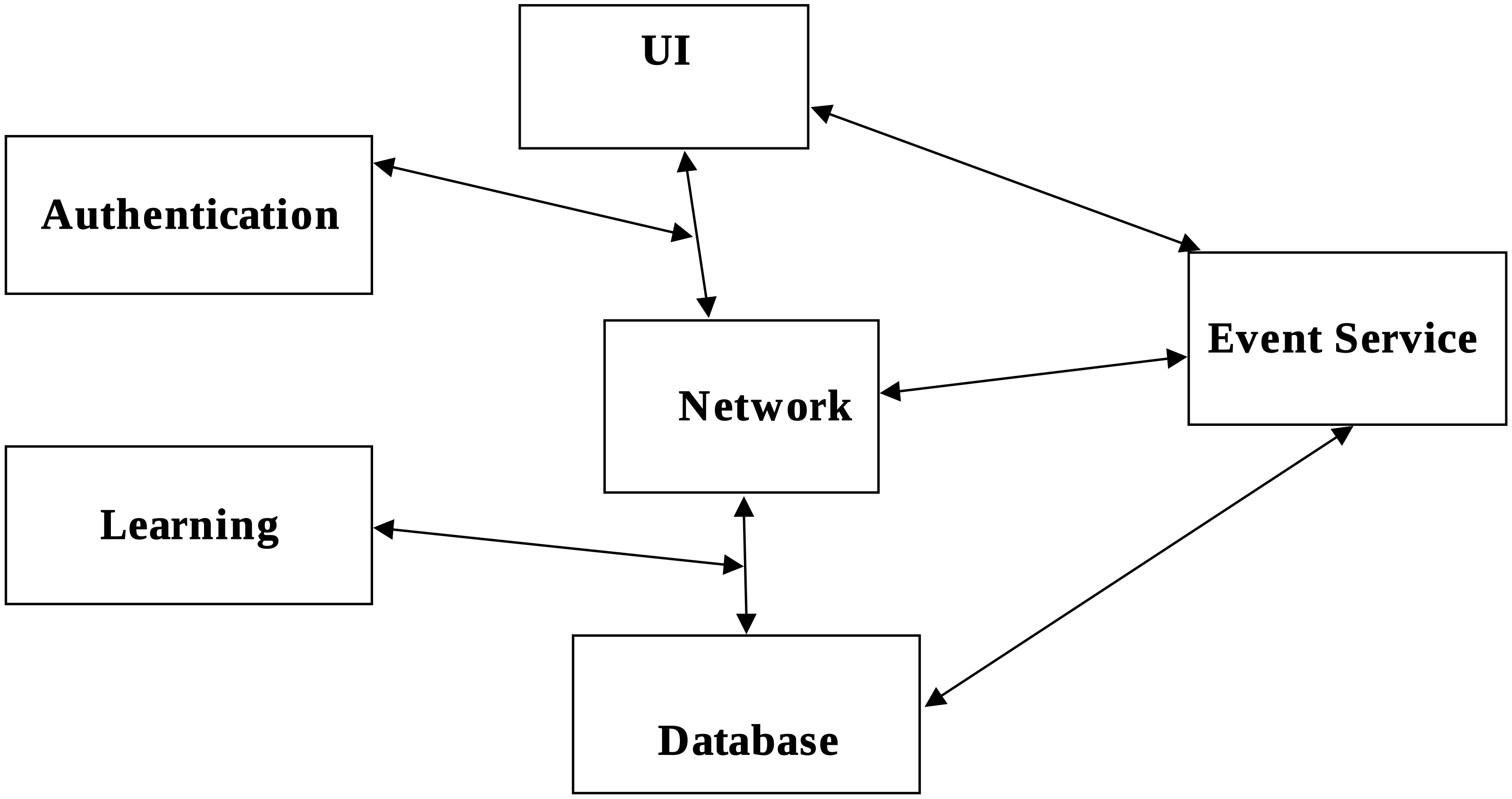
Push/Pull Notification with an Event Service



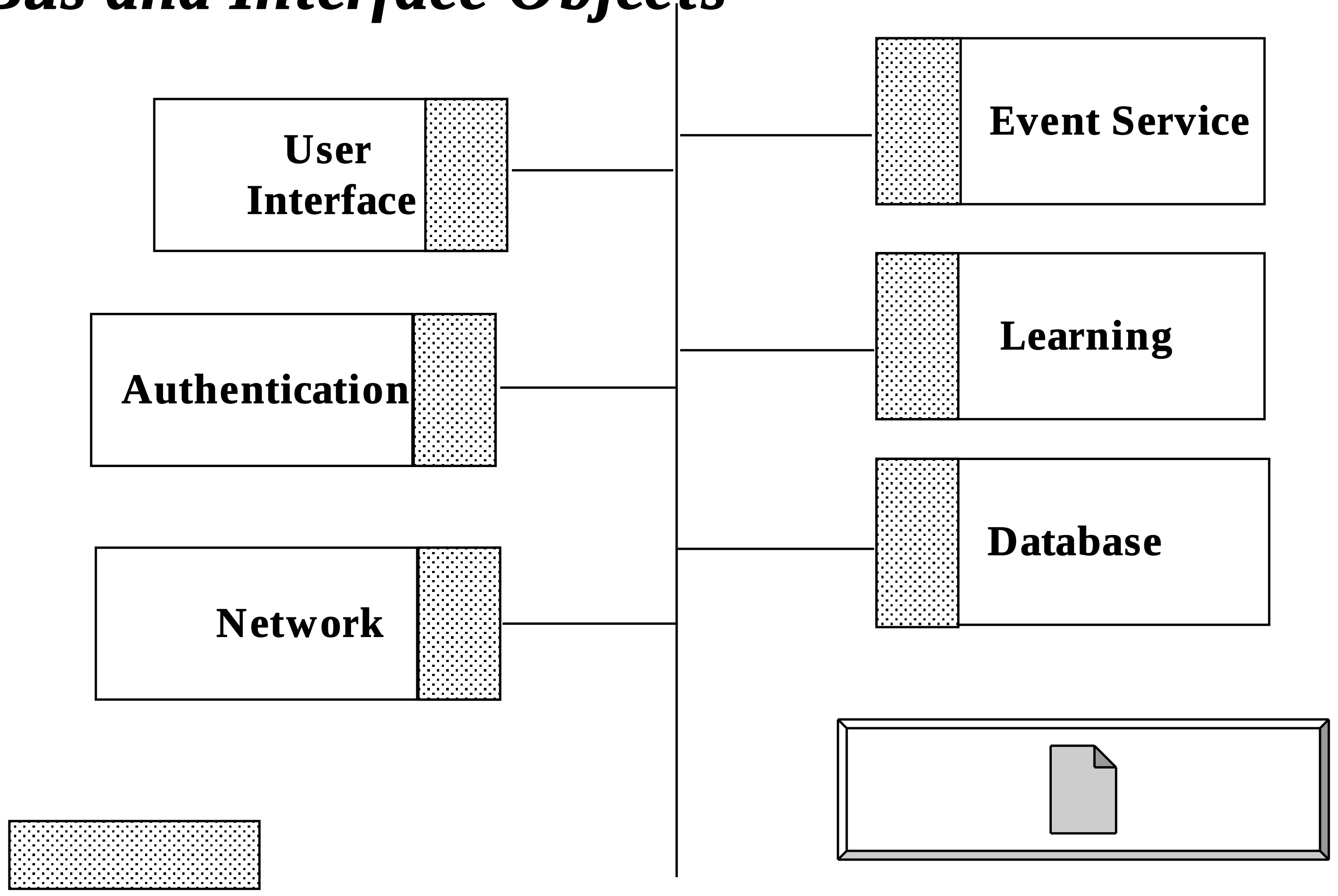
Choosing Subsystems

- Criteria for subsystem selection: Most of the interaction should be within subsystems, rather than across subsystem boundaries.
  - **Client-Server: Does one subsystem always call the other for the service?**
  - **Peer-Peer: Which of the subsystems call each other for service?**
- Primary Question:
  - **What kind of service is provided by the subsystems (subsystem interface)?**
- Secondary Question:
  - **Can the subsystems be hierarchically ordered?**
- What kind of model is good for describing layers and partitions?

Presentation of PAID as Peer-to-Peer Architecture



Alternative Presentation of PAID with Software Bus and Interface Objects

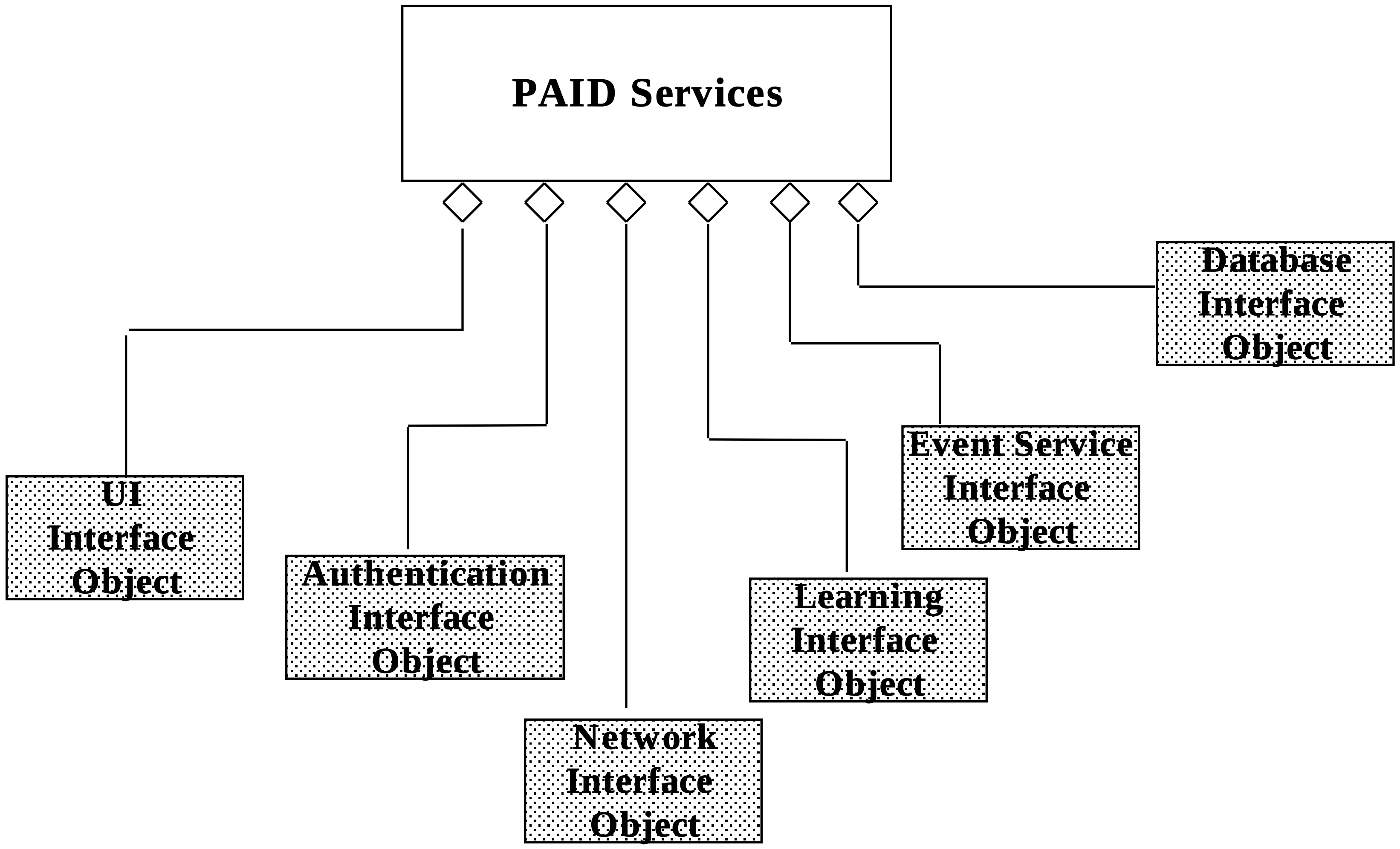


A SubsystemInterface Object publishes the service (= Set of public methods) provided by the subsystem

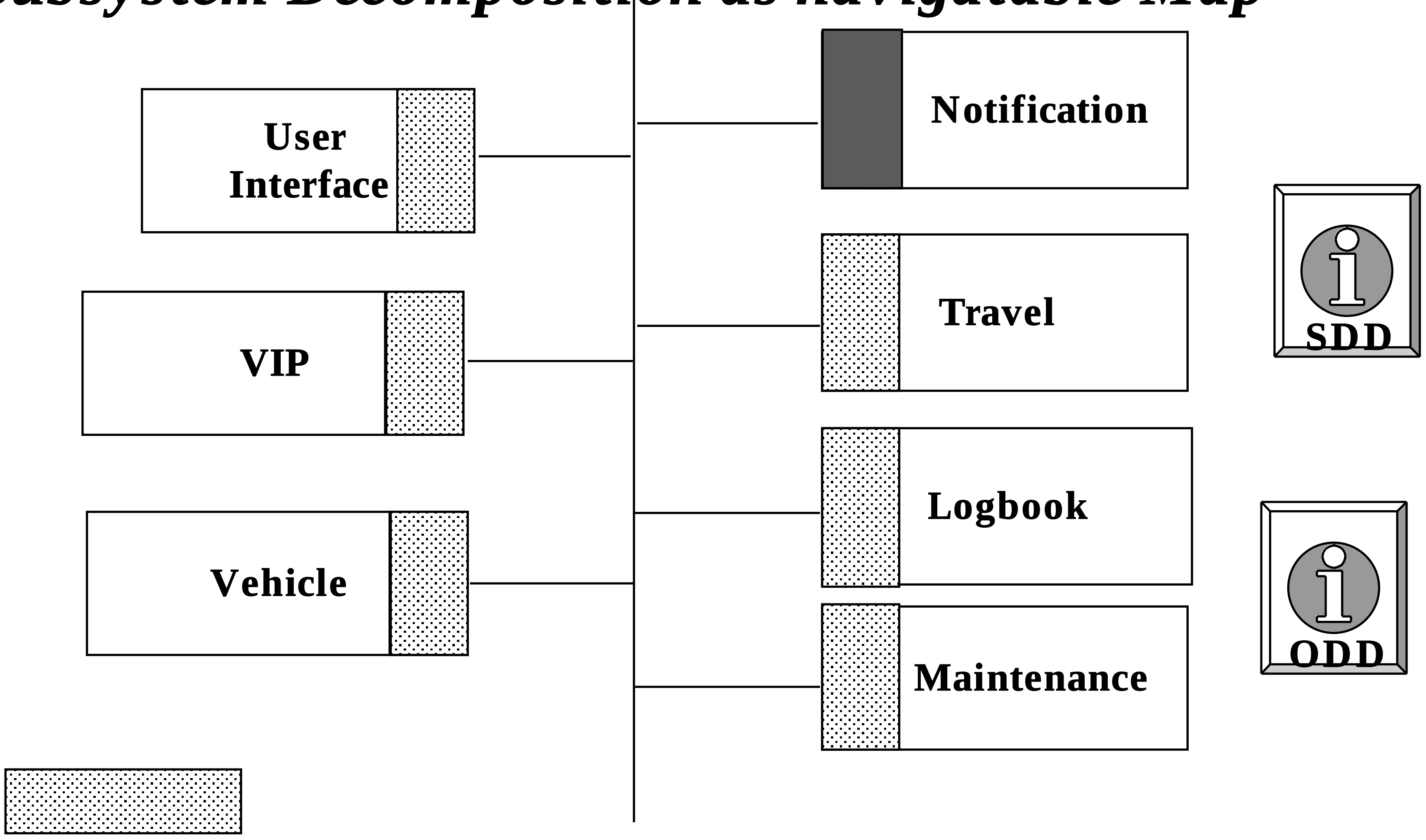
Definition: Subsystem Interface Object

- A Subsystem Interface Object publishes a service
  - **This is the set of public methods provided by the subsystem**
  - **Interface specification describes all the methods of the interface object**

PAID System as a set of services


PAID Services
Database Interface Object
UI Interface Object
Authentication Interface Object
Network Interface Object
Learning Interface Object
Event Service Interface Object

Example: JAMES Documentation Subsystem Decomposition as navigatable Map



A SubsystemInterface Object publishes the service (= Set of public methods) provided by the subsystem

3. Identify Concurrency

- Two objects are inherently concurrent if they can receive events at the same time without interacting
  - **Inherently concurrent objects should be assigned to different threads of control**
- Goal: Find as many concurrent objects as possible => High Performance
- Definition:
  - **A *thread of control* is a path through a set of state diagrams on which a single object is active at any time.**
- Each thread control should have its on processor (physically or logically)
- Objects with mutual exclusive activity should be folded into a single thread of control (Why?)

Concurrency Questions for PAID

- Which objects of the object model are independent?
  - **Describe the parallelism in PAID in terms of threads and processes/processors**
- Does the system provide access to multiple users?
  - **How many parallel users can the system have?**
- How many databases does PAID have?
  - **Can a single "query" consists of multiple queries?**

4. Allocate Subsystems: Hardware or Software?

- Major part of the design problem:
  - **How is the object model mapped on the existing hardware& software?**
- Much of the difficulty of designing a system comes from meeting externally-imposed hardware and software constraints.
  - **The designer does not have full freedom**

Hardware/Software Allocation Questions

- What are the estimated performance needs?
  - **(# of transactions/sec * transaction_processing_time)**
- What are the estimated hardware resource requirements?
  - **Number of processors depend on processing power required to maintain steady state load**
- Is the response time too slow? What can be done to improve it?
  - **If load is too demanding for a single processor, consider a speedup by distributing it across several processors.**
- What is the connectivity among physical units (Dealer, MLC, Headquarters)?
  - **Tree, star, matrix, ring**
- What is the appropriate communication protocol between the subsystems?
  - **Function of estimated bandwidth, latency and desired reliability)**

Hardware/Software Allocation Questions ctd

- Is certain functionality already available in hardware? (Smartcard)
- Do certain tasks require specific locations to control the hardware or to permit concurrent operation? Often true for embedded systems
  - **Example: ATM machine**
- General system performance questions:
  - **What is the desired response time?**
  - **What is the expected transaction rate? (requests/sec)?**
- I/O questions:
  - **Do you need an extra hardware to handle the data generation rate?**
  - **Does the response time or information flow rate exceed the available communication bandwidth between subsystems or a task and a piece of hardware?**
- Processor questions:
  - **Is the computation rate too demanding for a single processor?**
- Memory questions:
  - **Is there enough memory to buffer bursts of requests?**

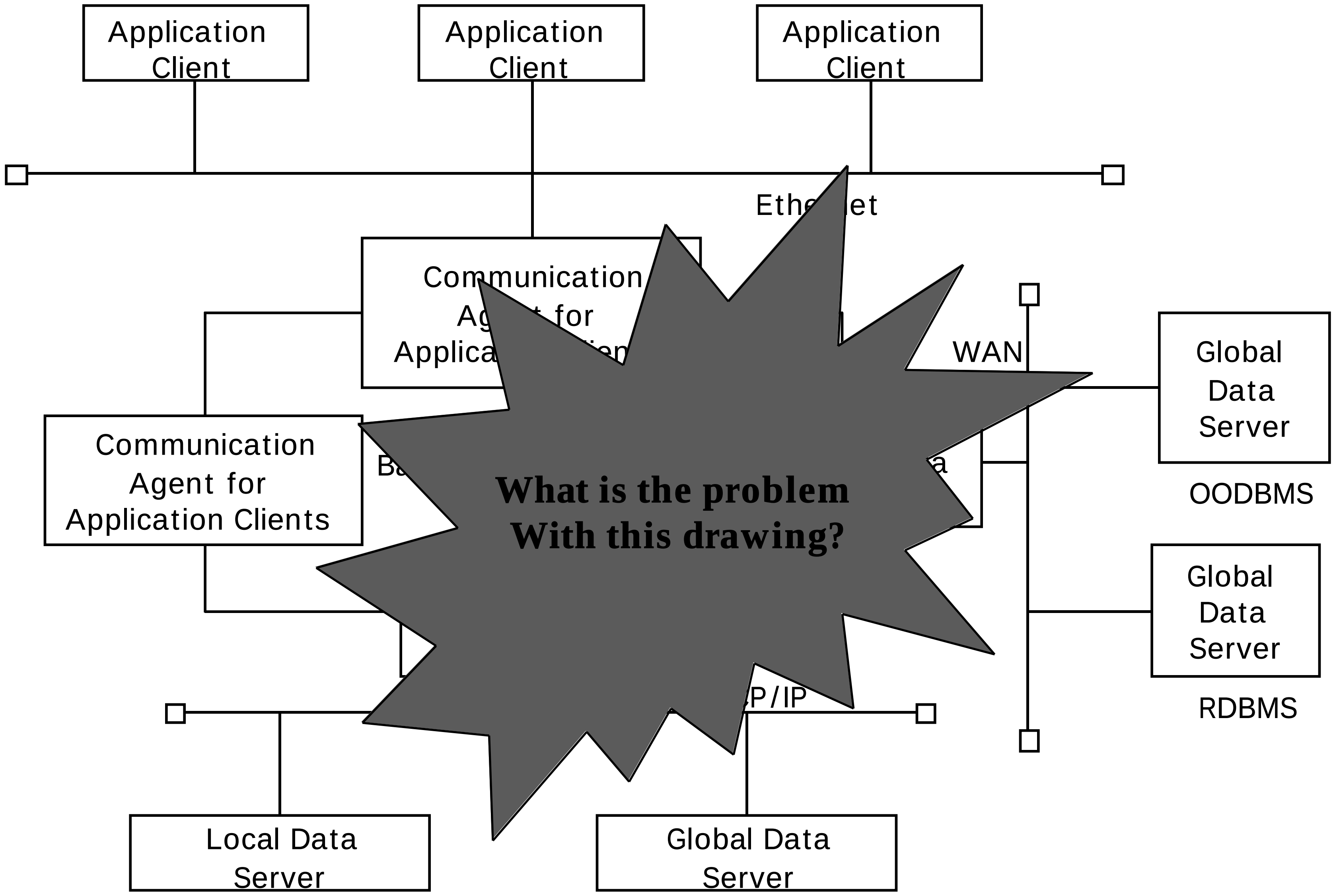
Connectivity (Associations)

- Describe the physical connectivity of the hardware (Physical layer in ISO's OSI Reference Model)
  - **Which associations in the object model are mapped to physical connections?**
  - **Which of the client-supplier relationships in the analysis/design model correspond to physical connections?**
- Describe the logical connectivity (software connections)
  - **Identify the logical connections that do not directly map into physical connections:**
    - **How are the logical connections implemented? By procedure calls? Remote procedure calls?**

Connectivity in Distributed Systems

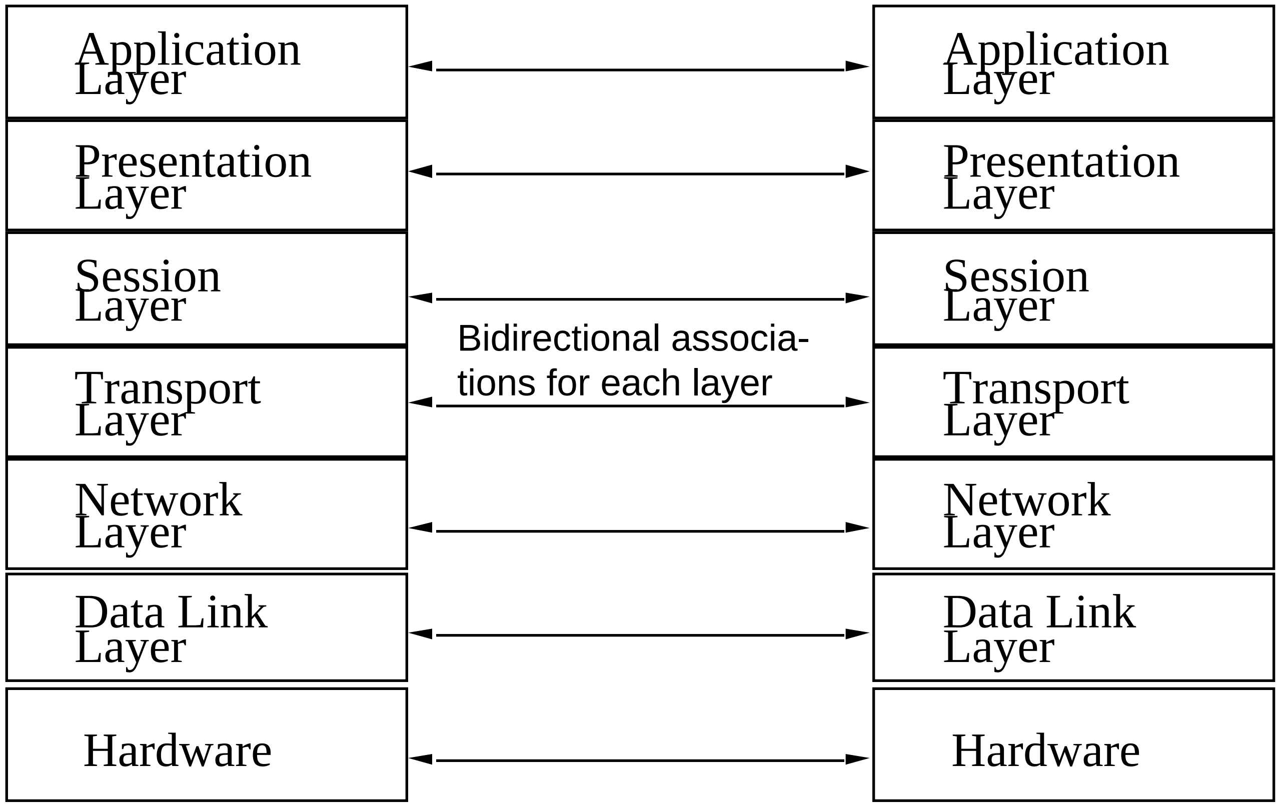
- If the architecture is distributed, we need to describe the network architecture (communication subsystem) as well:
  - **What are the transmission media?**
  - **What kind of connection channels and communication protocols are used?**
  - **Is the interaction asynchronous, synchronous or blocking?**
  - **What are the available bandwidth requirements**
    - **User Interface <-> Database Subsystem?**
    - **Event Service <-> All other PAID Subsystems?**
    - **Learning Subsystem <-> Database Subsystem?**

Example of a Physical Connectivity Drawing for a Distributed Database

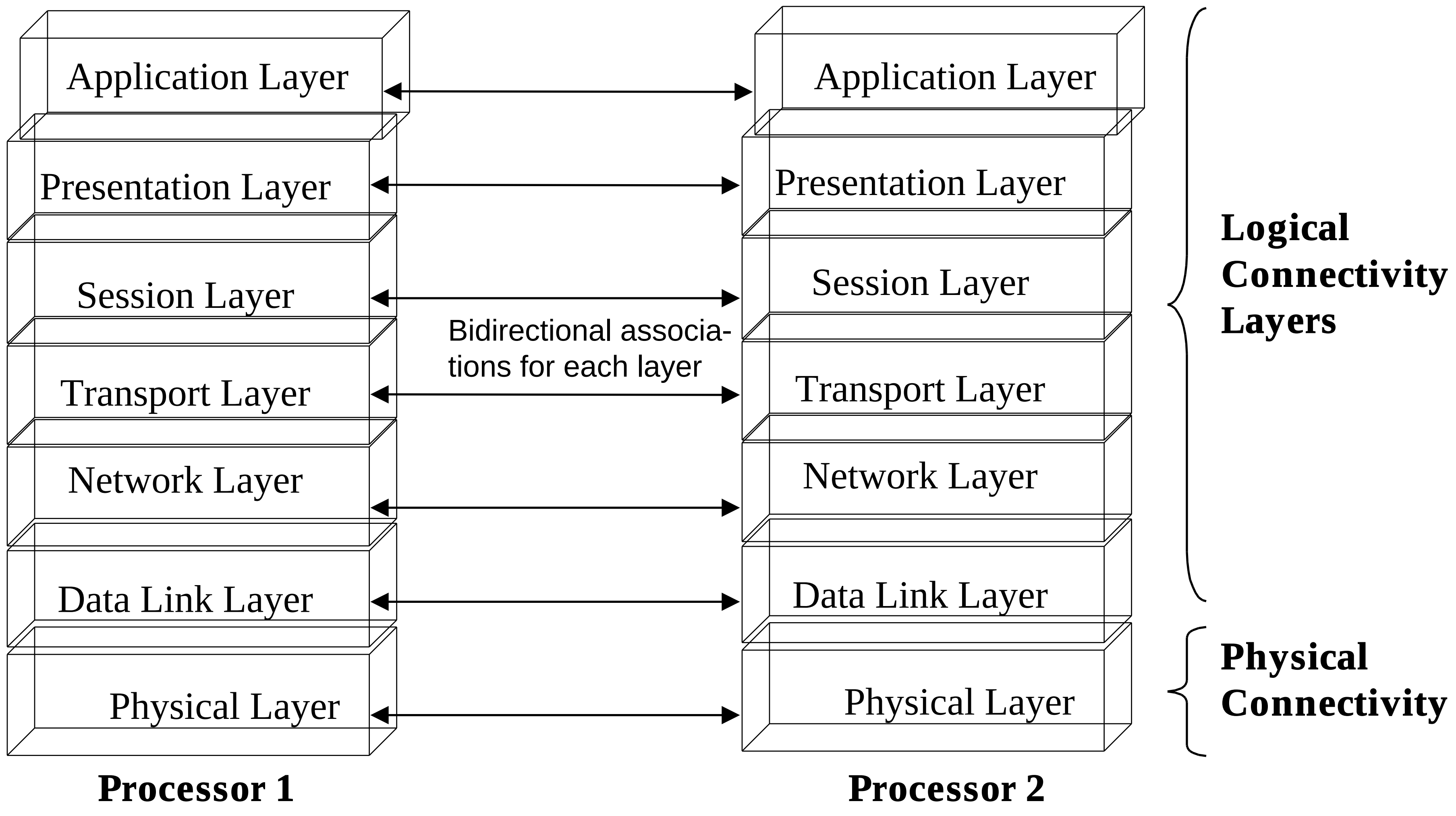


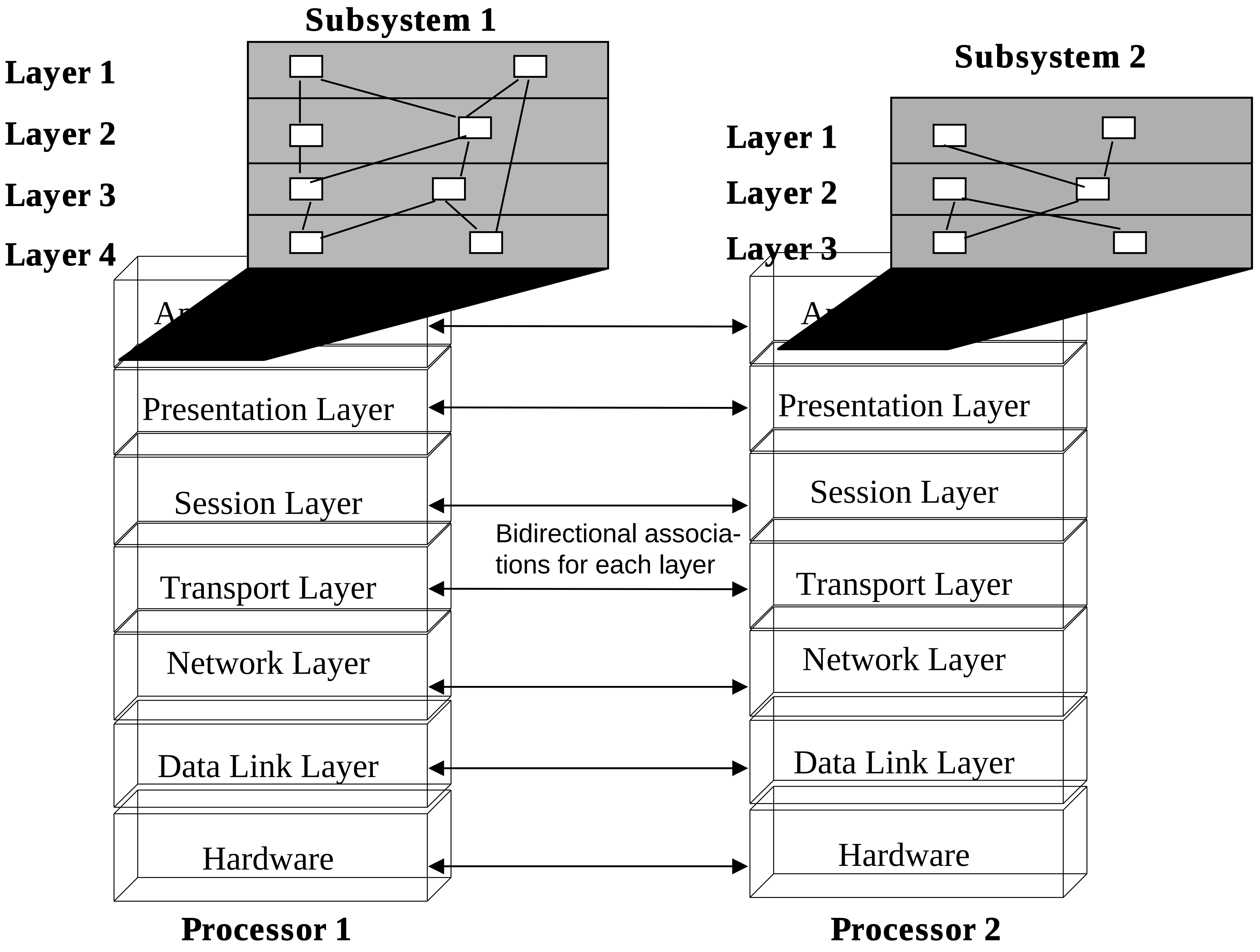
Physical vs Logical Connectivity

- ISO’s OSI Reference Model
  - **ISO = International Standard Organization**
  - **OSI = Open System Interconnection**
- Reference model defines 7 layers of network protocols and strict methods of communication between the layers.



Logical vs Physical Connectivity and the relationship to Subsystem Layering

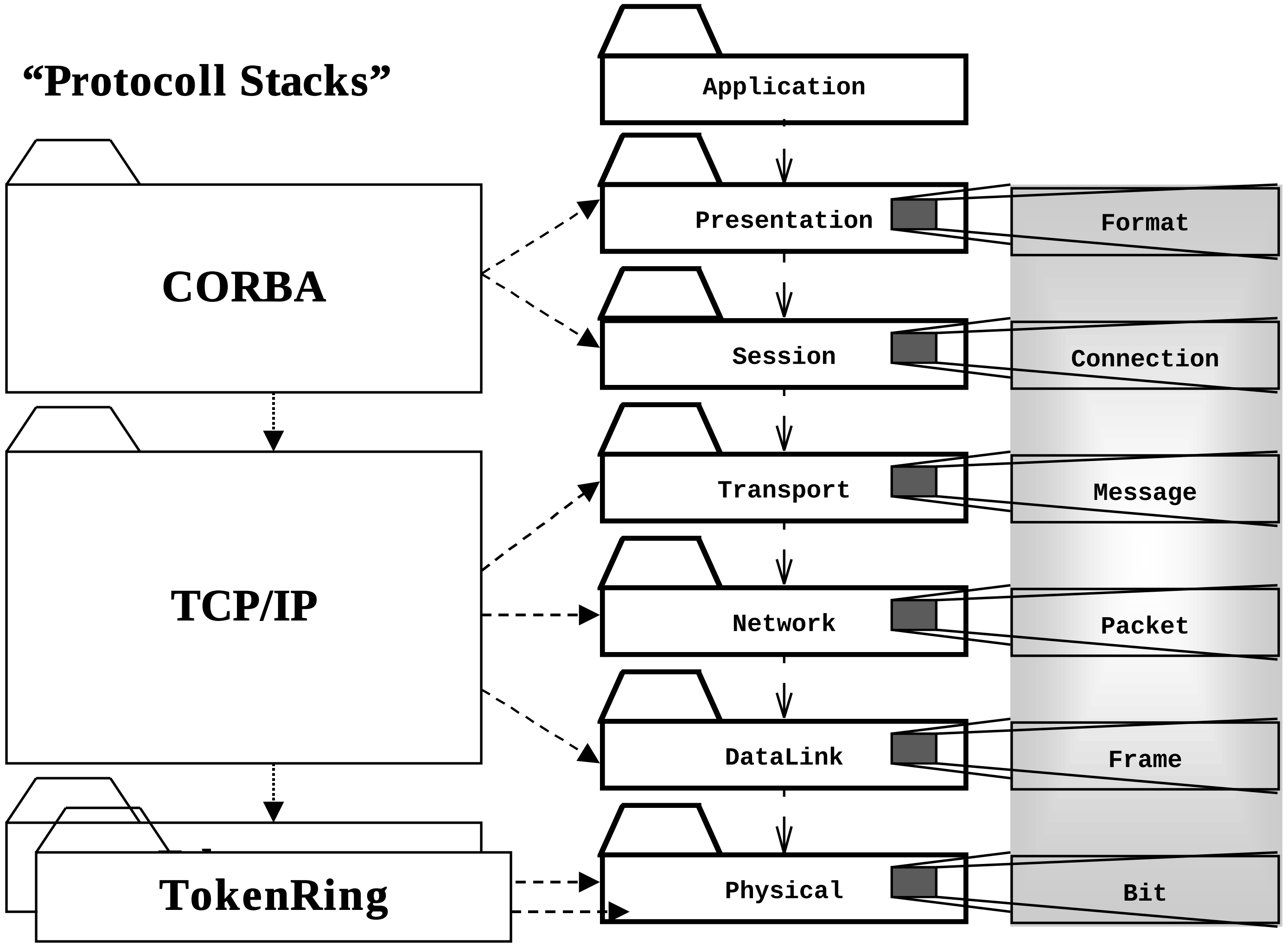



Subsystem 1
Layer 1
Layer 2
Layer 3
Layer 4
Subsystem 2
Layer 1
Layer 2
Layer 3
Ap
Ap
Presentation Layer
Presentation Layer
Session Layer
Session Layer
Bidirectional associa-
tions for each layer
Transport Layer
Transport Layer
Network Layer
Network Layer
Data Link Layer
Data Link Layer
Hardware
Hardware
Processor 1
Processor 2

Another View at the ISO Model

Defines an object-oriented open software architecture

Each layer is a UML package containing a set of objects



5. Management of Data

- Data Stores (Databases , files, ...) can provide clean separation points between subsystems with well-defined interfaces.
- A data store in general consists of
  - **Data structures**
    - **Volatile**
  - **Files**
    - **Cheap, simple, permanent storage**
    - **Low level, applications must have additional code to provide suitable level of abstraction**
  - **Data bases**
    - **Powerful, easy to port**
    - **Awkward interface with existing programming languages**
    - **Functionality often insufficient (mostly developed for commercial applications)**

File or Data base?

- When should you choose a file?
  - **Voluminous data (bit maps)**
  - **Lots of raw data (core dump, event trace)**
  - **Data needs to be kept only for a short time**
  - **Low information density for archival files and history logs**
- When should you choose a data base?
  - **Data requires access at fine levels of details by multiple users**
  - **Data must be ported across multiple platforms (heterogeneous systems)**
  - **Data must be accessible by multiple application programs**
  - **Data management requires a lot of infrastructure**

Data Management Questions

- Should the data be distributed?
- Should the database be extensible?
- What is the expected request (query) rate? In the worst case?
- How often is the database accessed?
- What is the size of typical requests (queries), worst case requests?
- Do the data need to be archived?
- Does the system design try to hide the location of the databases (location transparency)?
- Is there a need for a single interface to access the data?
- What is the query format?
- Should the database be relational or object-oriented?

6. Global Resources

- Discusses access control
- Describes access rights for different classes of actors
- Describes how object guard against unauthorized access

Global Resource Questions

- Does the system need authentication?
- If yes, what is the authentication scheme?
  - **User name and password?**
  - **Tickets?**
- What is the user interface for authentication?
- Does the system have a network-wide name server?
- How is a service known to the rest of the system?
  - **At runtime? At compile time?**
  - **By Port?**
  - **By Name?**

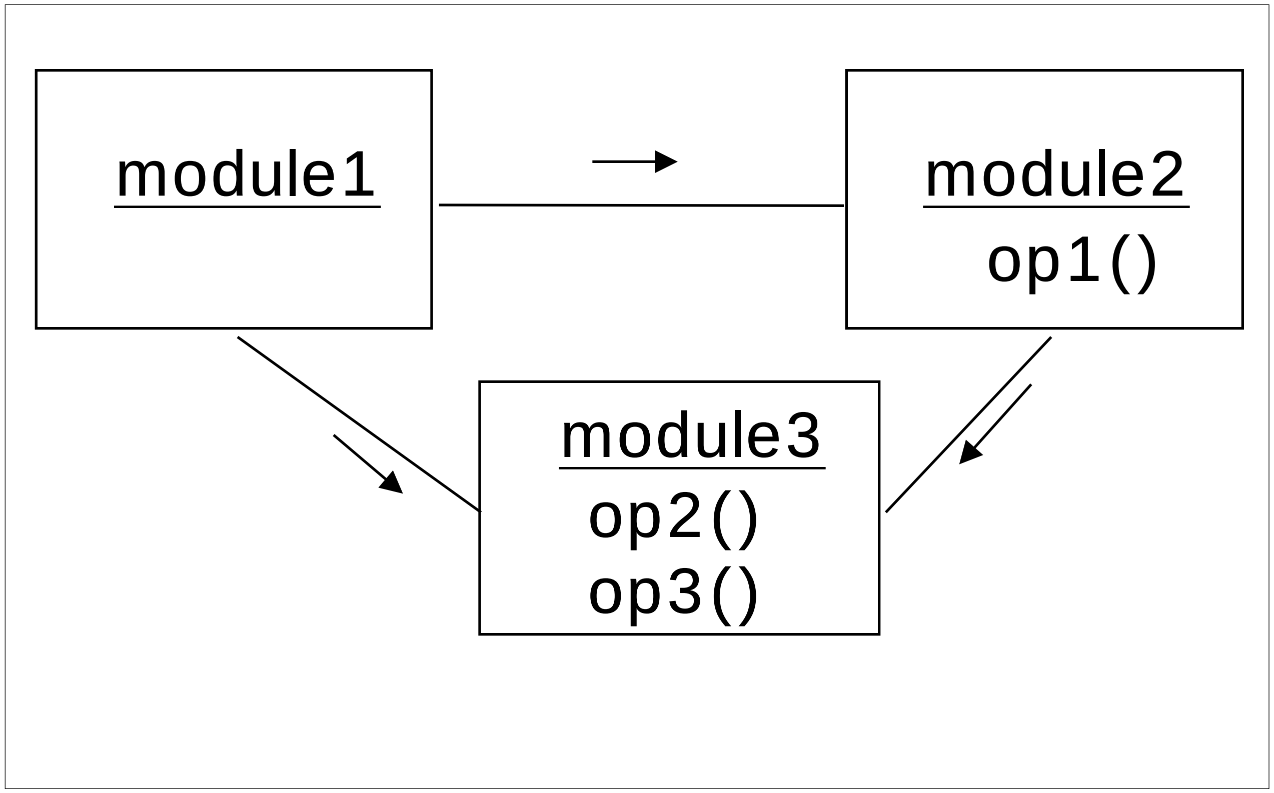
7. Decide on Software Control

- Choose explicit control (procedural languages)
  - **Centralized control**
    - **Procedure-driven control**
      - **Control resides within program code. Example: Main program calling procedures of subsystems.**
      - **Simple, easy to build**
    - **Event-driven control**
      - **Control resides within a dispatcher who calls subsystem functions via callbacks.**
      - **Flexible, good for user interfaces**

Software Control, ctd

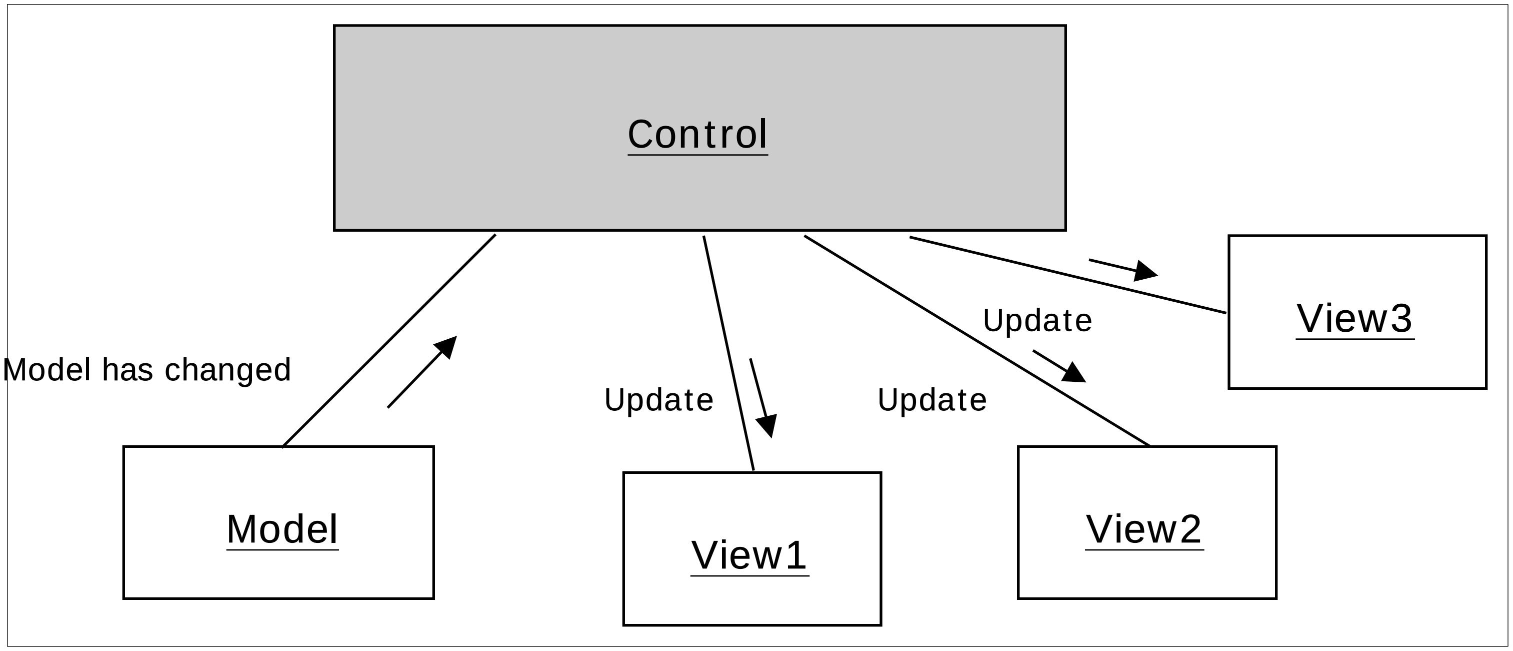
- **Decentralized control**
  - **Control resided in several independent objects (supported by some languages).**
  - **Possible speedup by parallelization, increased communication overhead.**
  - **Example: Message based system.**
- Or choose implicit control (non-procedural or declarative languages)
  - **Rule-based systems**
  - **Logic programming**

Procedure-Driven Control Example



Event-Based System Example: MVC

- Smalltalk-80 Model-View-Controller
- Client/Server Architecture



Centralized vs. Decentralized Designs

- Centralized Design
  - **One control object or subsystem ("spider") controls everything**
  - **Change in the control structure is very easy**
  - **Possible performance bottleneck**
- Decentralized Design
  - **Control is distributed**
  - **Spreads out responsibility**
  - **Fits nicely into object-oriented development**
- Should PAID use a centralized or decentralized design?

8. Boundary Conditions

- Most of the system design effort is concerned with steady-state behavior.
- However, the system design phase must also address the initiation and finalization of the system.
  - **Initialization**
    - **Describes how the system is brought from an unitialized state to steady-state ("startup scenario”).**
  - **Termination**
    - **Describes what resources are cleaned up and what tasks or other systems are notified upon termination ("termination scenarios").**
  - **Failure**
    - **Many possible causes: Bugs, errors, external problems (power supply). Good system design foresees fatal failures. Example: Catch Unix signals or Java Exceptions**

Boundary Condition Questions

- 9.1 Initialization
  - **How does the system start up?**
    - **What data need to be accessed at startup time?**
    - **What services have to registered?**
  - **What does the user interface do at start up time?**
    - **How does it present itself to the user?**
- 9.2 Termination
  - **Are single subsystems allowed to terminate?**
  - **Are other subsystems notified if a single subsystem terminates?**
  - **How are local updates communicated to the database?**
- 9.3 Failure
  - **How does the system behave when a node or communication link fails? Are there backup communication links?**
  - **How does the system recover from failure? Is this different from initialization?**

Design Rationale

- The system design describes a solution based on existing technology
  - **What issues were discussed?**
  - **What other solutions have been considered? Why were these not used?**
  - **What is the reasoning behind the decisions made?**
  - **What resolutions were made? Why?**
  - **What kind of technological enablers were discussed but not used? Did they influence your design?**
- Lecture on November 10

System Design Document: Assignments and Deadlines

- Individual Section Assignments:
  - **1. Goals and Trade-offs (Documentation Core Team)**
  - **2. System Decomposition (Architecture Core Team)**
  - **3. Concurrency Identification (Learning Team)**
  - **4. Hardware/Software Mapping (Network Team)**
  - **5. Data Management(Database Team)**
  - **6. Global Resource Handling (Authentication Team)**
  - **7. Software Control Implementation (Architecture Core Team)**
  - **8. Boundary Conditions (User Interface Team)**
  - **9. Design Rationale (All Teams)**
- Deadlines:
  - **Submission of section 1 to 8: Oct 27, 6pm.**
  - **Integration of section 1 to 8 (Documentation team): Nov 4, 3pm**
  - **Submission of section 9: Nov 17, 6pm**
  - **Integration, Revision of SDD and ODD (JavaDoc Documentation): Due Nov 23, 3pm**