Architectural views

Architectural views

- [Gacek, 1994]
  - "To accommodate the **different expectations** of the various **stakeholders**, a software architecture must incorporate different, **multiple views**"

Stakeholder	Concern
Customer	• Schedule and budget estimation • Feasibility and risk assessment • Requirements traceability • Progress tracking
User	• Consistency with requirements and usage scenarios • Future requirement growth accommodation • Performance, reliability, interoperability, etc.
Architect / System Eng.	• Requirements traceability • Support of trade-off analyses • Completeness, consistency of architecture
Developer	• Sufficient detail for design • **Reference for selecting / assembling components** • Maintain interoperability with existing systems
Maintainer	• Guidance on software modification • Guidance on architecture evolution • Maintain interoperability with existing systems

Architectural views

- Views of a Software Architecture [Gacek, 1994]
  - Static Topological
  - Behavioral / Operational
  - Dataflow
  - Computing Environment
  - Process Environment

Architectural views

- [Kruchten, 1995]
  - The 4+1 View Model
    - **Logical view**
    - **Process view**
    - **Physical view**
    - **Development view**
    - ***Scenarios***

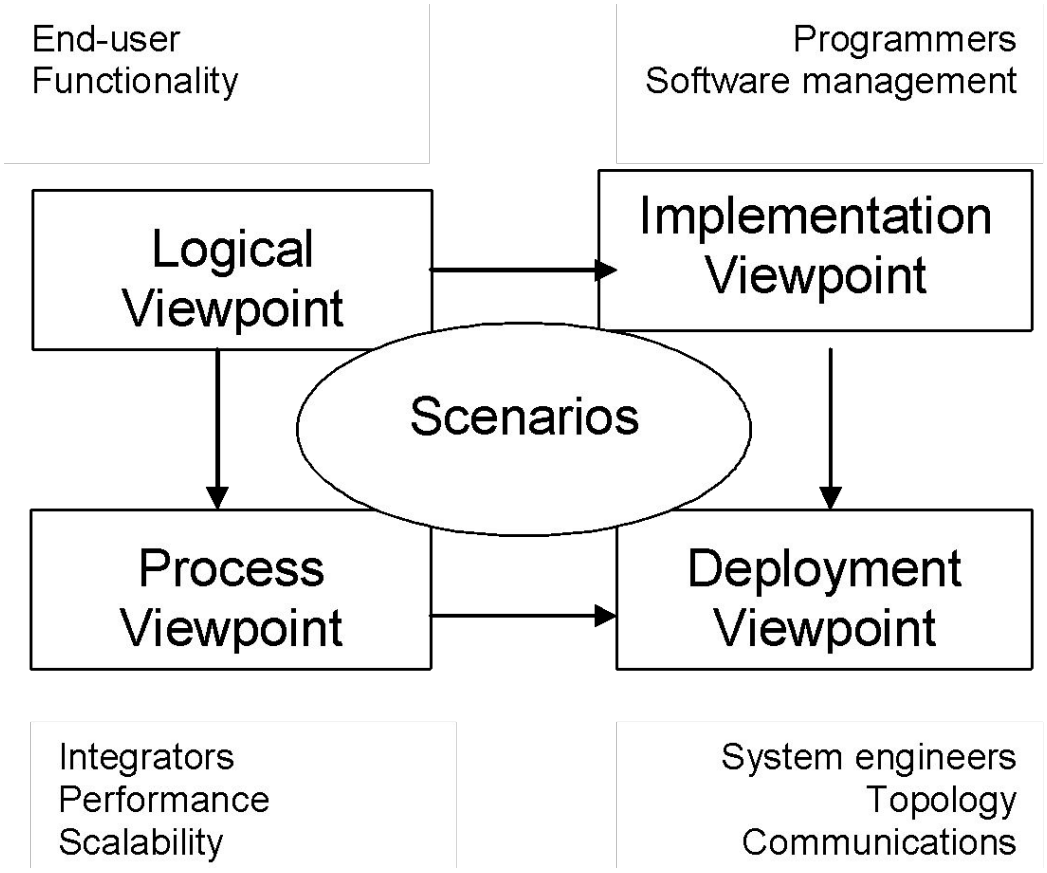
End users, customers, data specialists

System engineers

Project managers,
software-configuration staff members

  - ***Scenario-driven approach*** ***capture the system`s critical functionality***
  - ***Style Vs View…***

Kruchten's 4+1 view model



4 + 1: Logical Viewpoint

- The logical viewpoint supports the functional requirements, i.e., the services the system should provide to its end users.
- Typically, it shows the key abstractions (e.g., classes and interactions amongst them).

4 + 1: Process Viewpoint

- Addresses concurrent aspects at runtime (tasks, threads, processes and their interactions)
- It takes into account some nonfunctional requirements, such as performance, system availability, concurrency and distribution, system integrity, and fault-tolerance.

4 + 1: Deployment Viewpoint

- The deployment viewpoint defines how the various elements identified in the logical, process, and implementation viewpoints-networks, processes, tasks, and objects-must be mapped onto the various nodes.

- It takes into account the system's nonfunctional requirements such as system availability, reliability (fault-tolerance), performance (throughput), and scalability.

4 + 1: Implementation Viewpoint

- The imlementation viewpoint focuses on the organization of the actual software modules in the software-development environment.

- The software is packaged in small chunks-program libraries or subsystems-that can be developed by one or more developers.

4 + 1: Scenario Viewpoint

- The scenario viewpoint consists of a small subset of important scenarios (e.g., use cases) to show that the elements of the four viewpoints work together seamlessly.

- This viewpoint is redundant with the other ones (hence the "+1"), but it plays two critical roles:
  - it acts as a driver to help designers discover architectural elements during the architecture design;
  - it validates and illustrates the architecture design, both on paper and as the starting point for the tests of an architectural prototype.

Architectural views from Bass et al (view = representation of a structure)

- Module views
  - Module is unit of implementation
  - Decomposition, uses, layered, class
- Component and connector (C & C) views
  - These are runtime elements
  - Process (communication), concurrency, shared data (repository), client-server
- Allocation views
  - Relationship between software elements and environment
  - Work assignment, deployment, implementation

Module views

- Decomposition: units are related by “is a submodule of”, larger modules are composed of smaller ones
- Uses: relation is “uses” (calls, passes information to, etc). Important for modifiability
- Layered is special case of uses, layer *n* can only use modules from layers <*n*
- Class: generalization, relation “inherits from”

Component and connector views

- Process: units are processes, connected by communication or synchronization
- Concurrency: to determine opportunities for parallelism (connector = logical thread)
- Shared data: shows how data is produced and consumed
- Client-server: cooperating clients and servers

Constructing the Architecture

- Significant Use Cases
- Identify Systems & Subsystems
- Identify/Extract Classes (Structure, Relationship)
- Identify the potential for Concurrency & Distribution
- Manage the Data Stores
- Implement one or more architectural prototype
- Derive tests from the Use Cases
- Evaluate the architecture

Patterns and Architecture

- “A pattern is a common solution to a common problem in a given context.”
- “A framework is an architectural pattern that provides an extensible template for applications within a domain.”

Examples of architectural patterns

microservices, message bus, service requester/ consumer, MVC, MVVM, microkernel, n-tier, domain-driven design, and presentation-abstraction-control.

Architectural Styles

Software Architecture-definition

Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution

(from IEEE Standard on the Recommended Practice for Architectural Descriptions, 2000.)

Examples of Architecture Descriptions

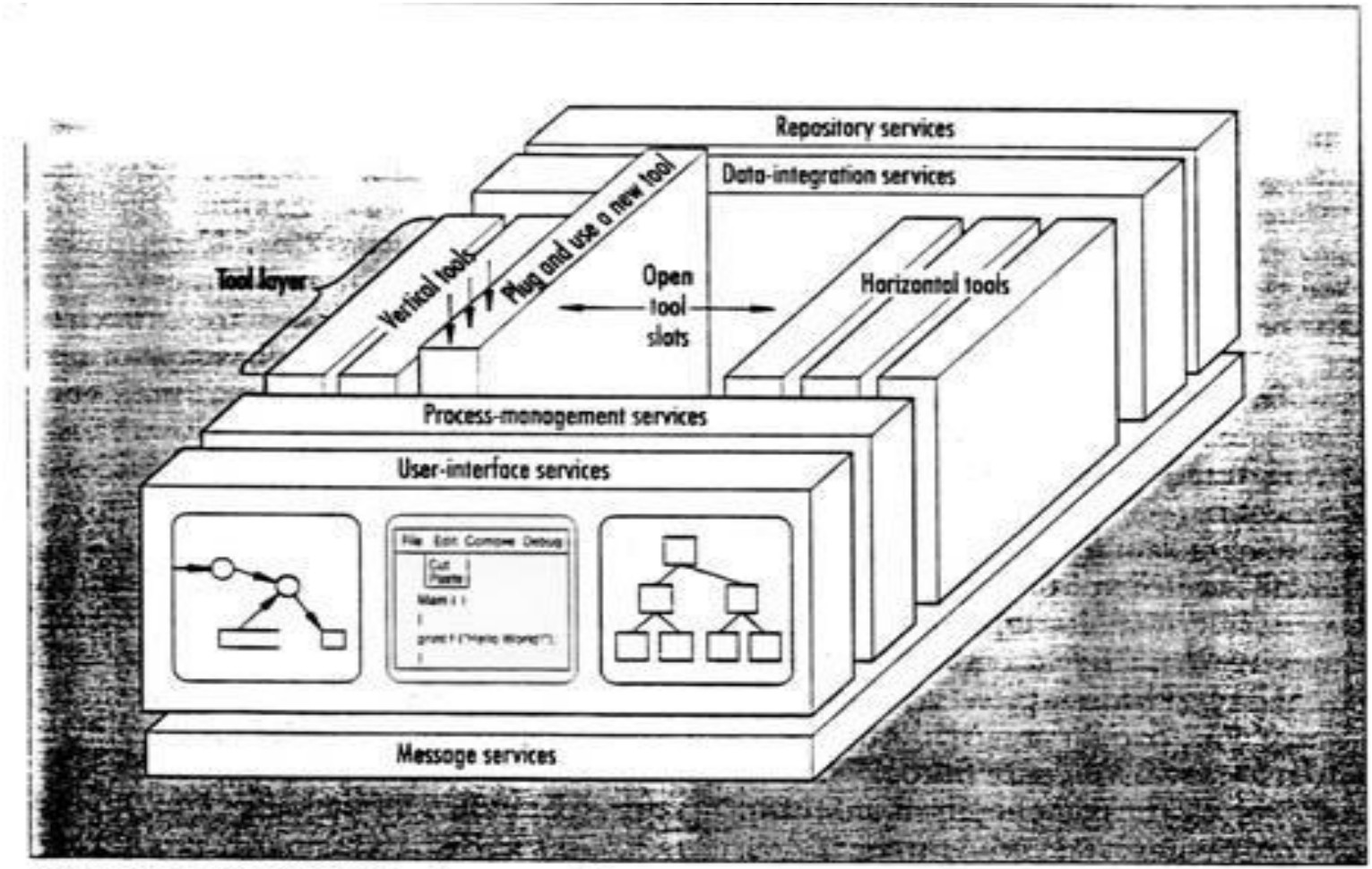


Figure 1. The NIST/ECMA reference model.

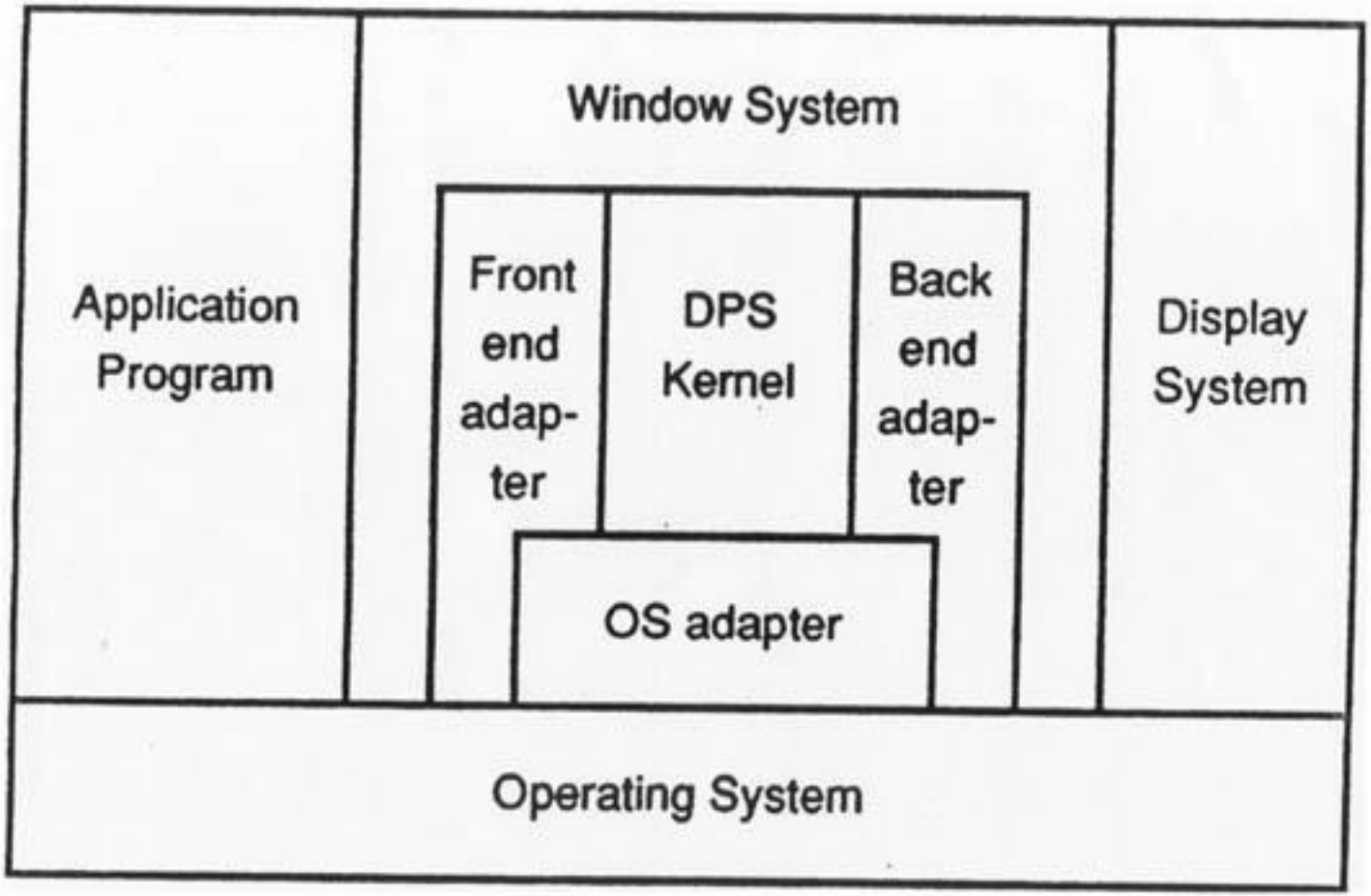


Figure 2. Display PostScript interpreter components.

An Overview of the DISPLAY POSTSCRIPT™ System, Adobe Systems Incorporated, March 16, 1988, P. 10

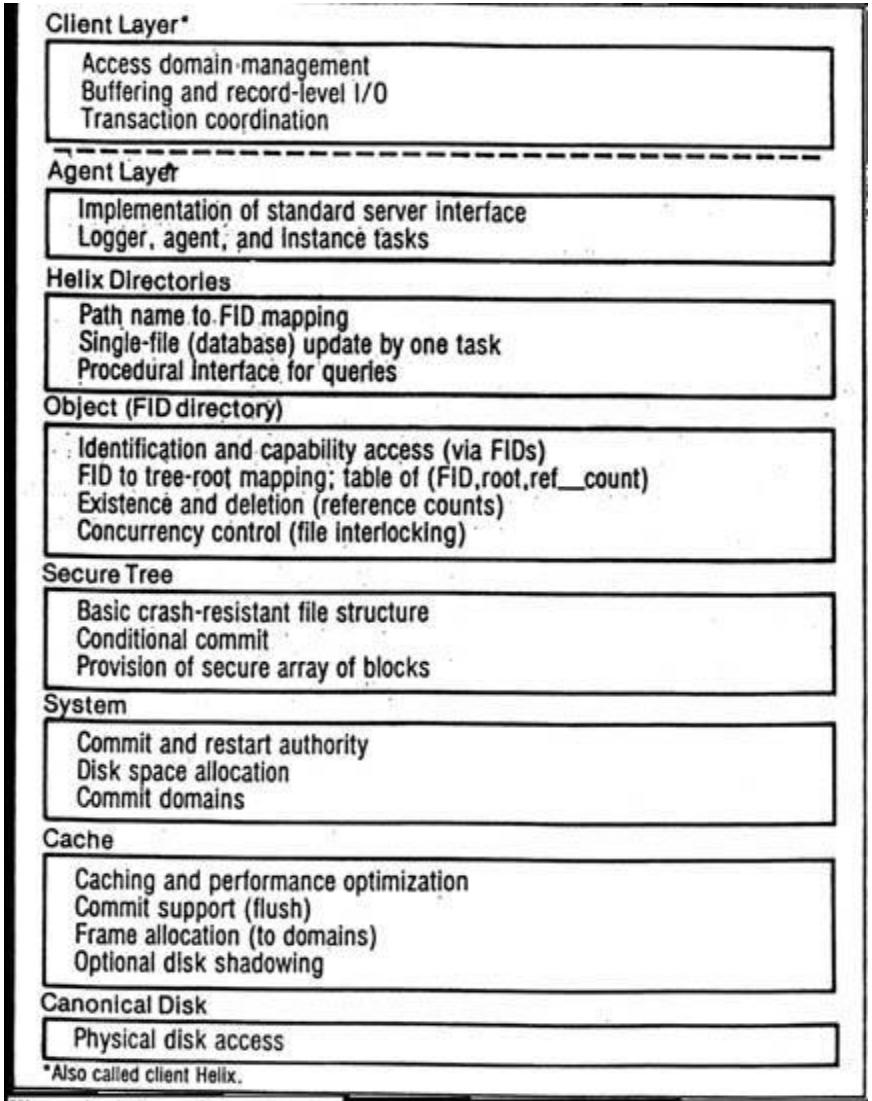


Figure 2. Abstraction layering.

IEEE Software, "Helix: The architecture of the XMS Distributed File System, Marek Fridrich and William Older, May 1985, Vol. 2, No. 3, P. 23

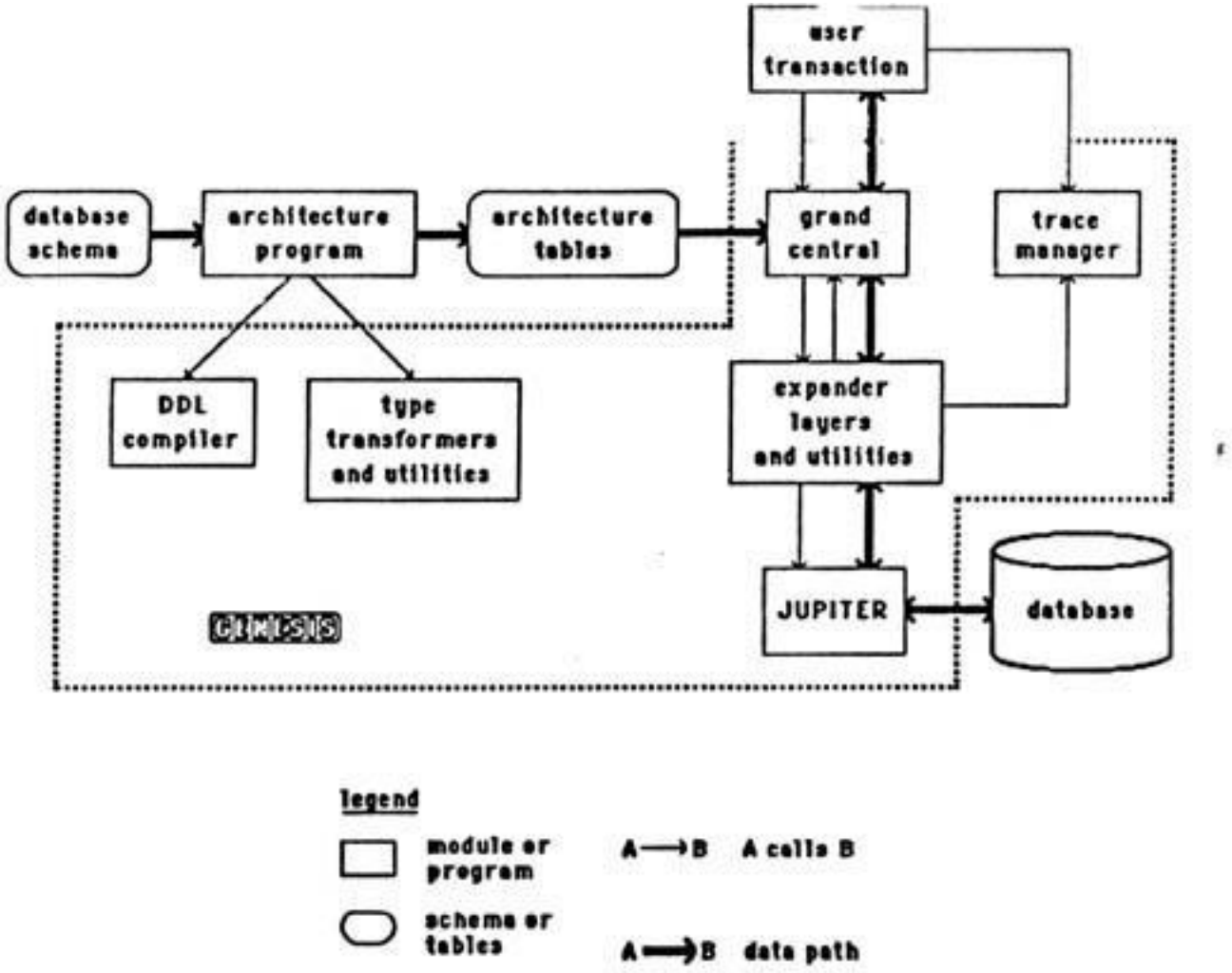


Figure 3.1 The Configuration of the GENESIS Prototype

Genesis: A Reconfiguration Database Management System, D. S. Batory, J.R. Barnett, J.F. Garza, K.P. Smith, K. Tsukuda, B.C. Twichell, T.E. Wise, Department of Computer Sciences, University of Texas at Austin,

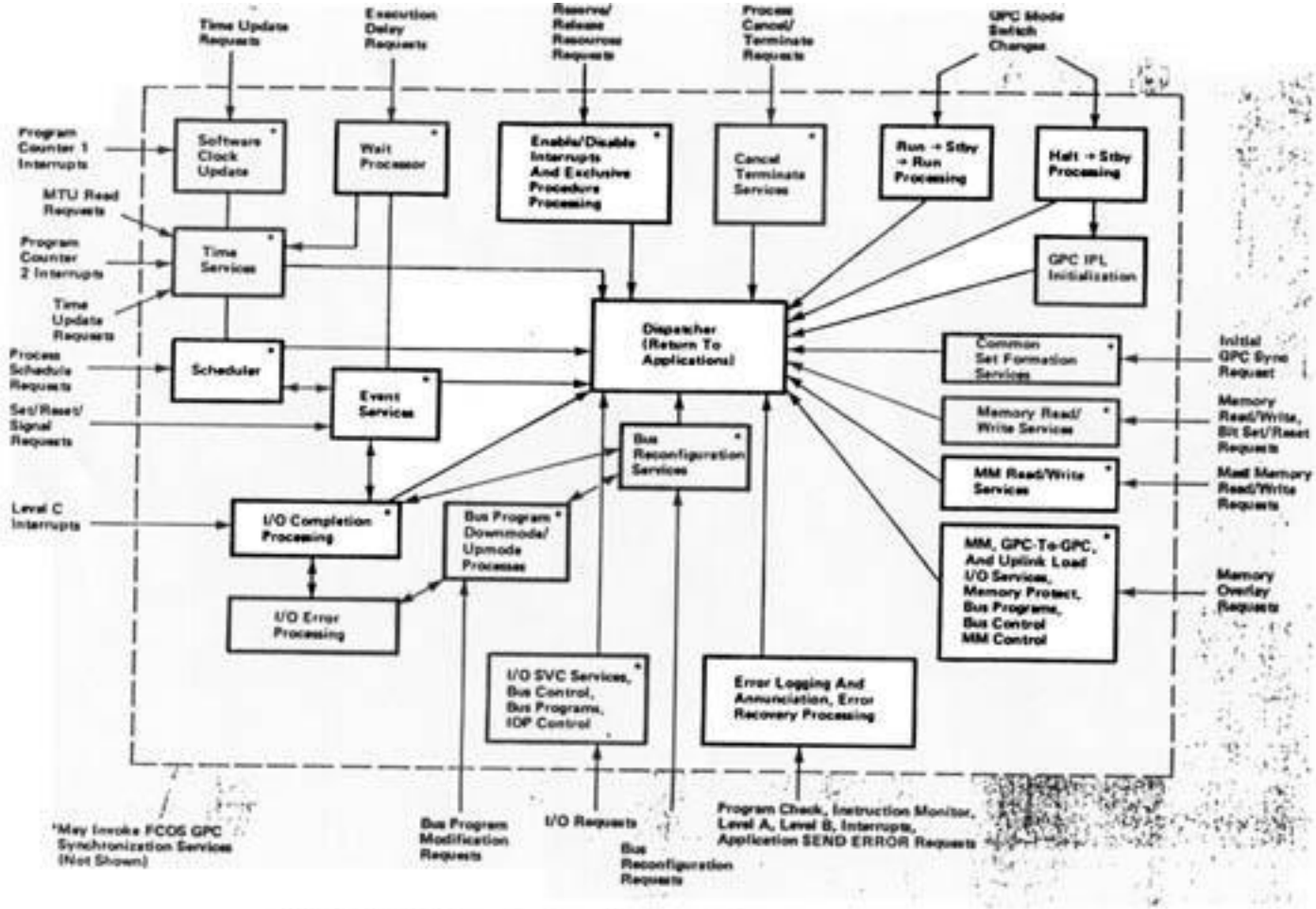
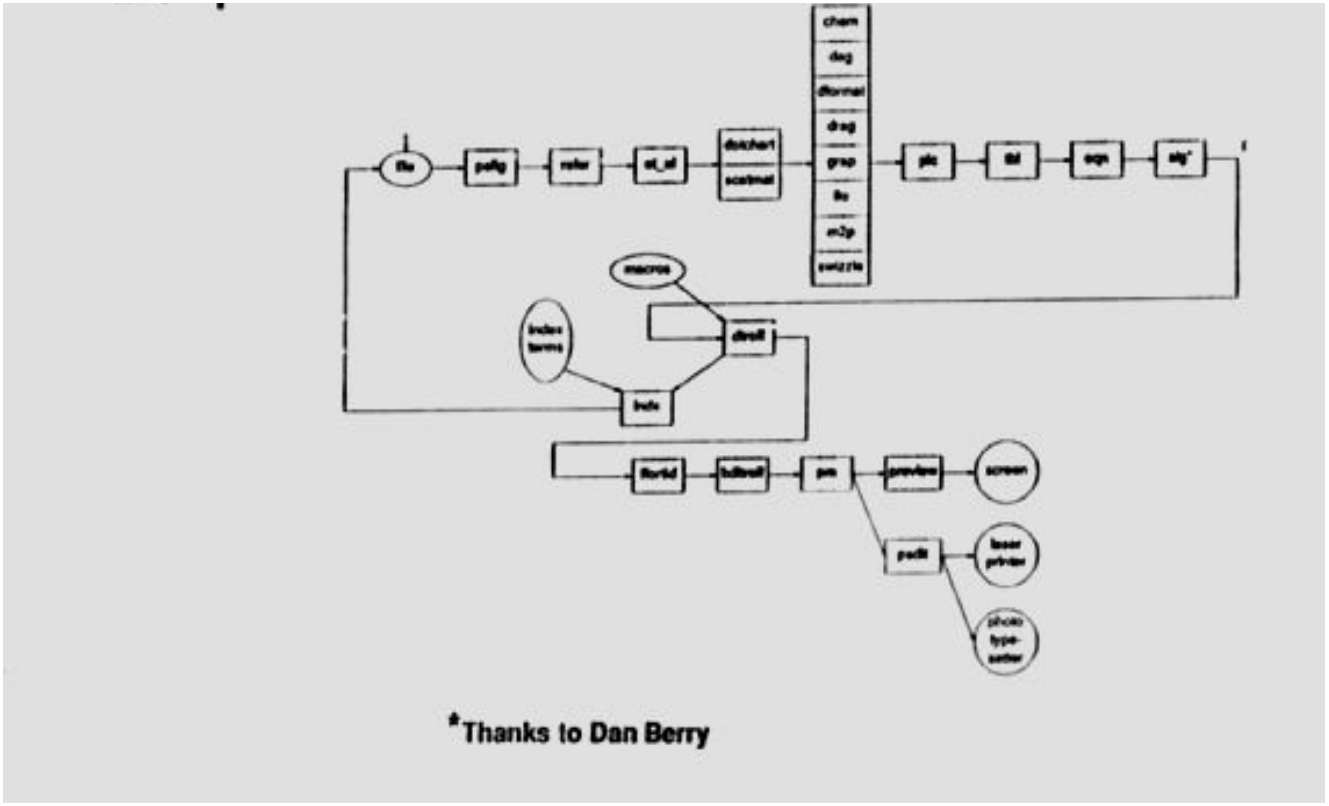


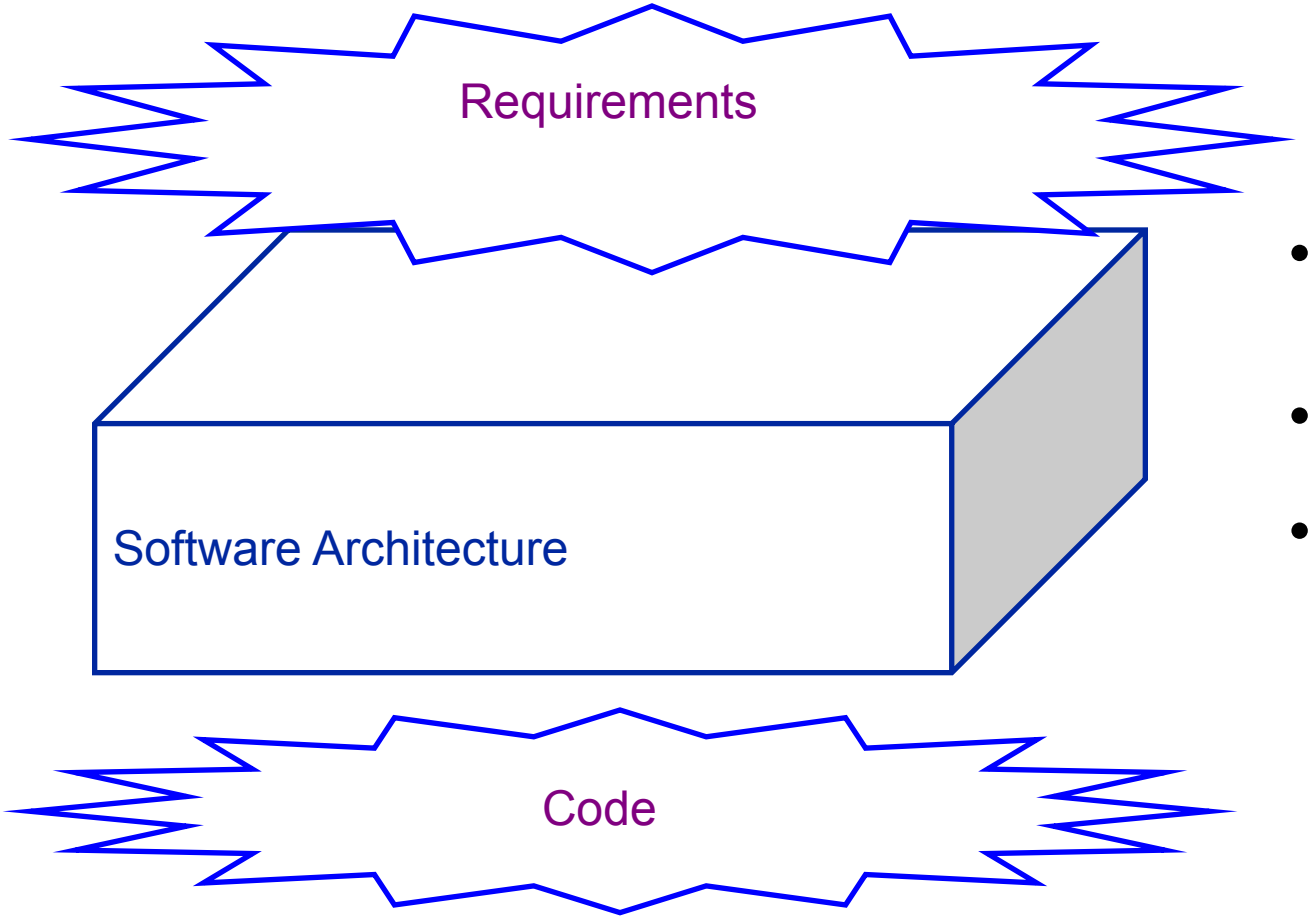
FIGURE 7. Flight Computer Operating System (The FCOS dispatcher coordinates and controls all work performed by the on-board computers.)

Communications of the ACM, "Architecture of the Space Shuttle Primary Avionics Software Systems," Gene D. Carlow, September 1984, Vol. 27, No. 9, P. 933



*Thanks to Dan Berry

The Role of Software Architecture



- Composition of large-scale components
- System-level abstractions
- Reuse of system-level design idioms

Elements of Architectural Descriptions: Styles

- An **architectural style** defines a ***family*** of architecture instances including
  - **Component/connector types**
    - the vocabulary of architectural building blocks
  - **Constraints** on how the building blocks can be used, including
    - **topological** rules
    - **interface** standards
    - **required** properties

Note: relationship between architectural styles and system instances is similar to that between types and instances

Common architectural styles

- One of the first listings by Shaw & Garlan (1996):
  - Pipes and filters
  - Batch sequential
  - Main program and subroutines
  - Object-oriented systems
  - Layers
  - Communicating processes
  - Event-based systems
  - Interpreters
  - Rule-based systems
  - Databases
  - Hypertext systems
  - Blackboards
- Further styles listed by Rozanski & Woods (2005)
  - Client-server
  - Tiered computing
  - Peer-to-peer
  - Publisher-subscriber
  - Asynchronous data replication
  - Distribution tree
  - Integration hub
  - Tuple space

Taxonomy of Architectural Styles

Data Flow

- Batch sequential
- Dataflow network (pipes&filters)
  - acyclic, fanout, pipeline, Unix
- Closed loop control

Call-and-return

- Main program/subroutines
- Information hiding
  - ADT, object, naive client/server

Interacting processes

- Communicating processes
- Event systems
  - implicit invocation, pure events

Data-oriented repository

- Transactional databases
  - True client/server
- Blackboard
- Modern compiler

Data-sharing

- Compound documents
- Hypertext
- Fortran COMMON

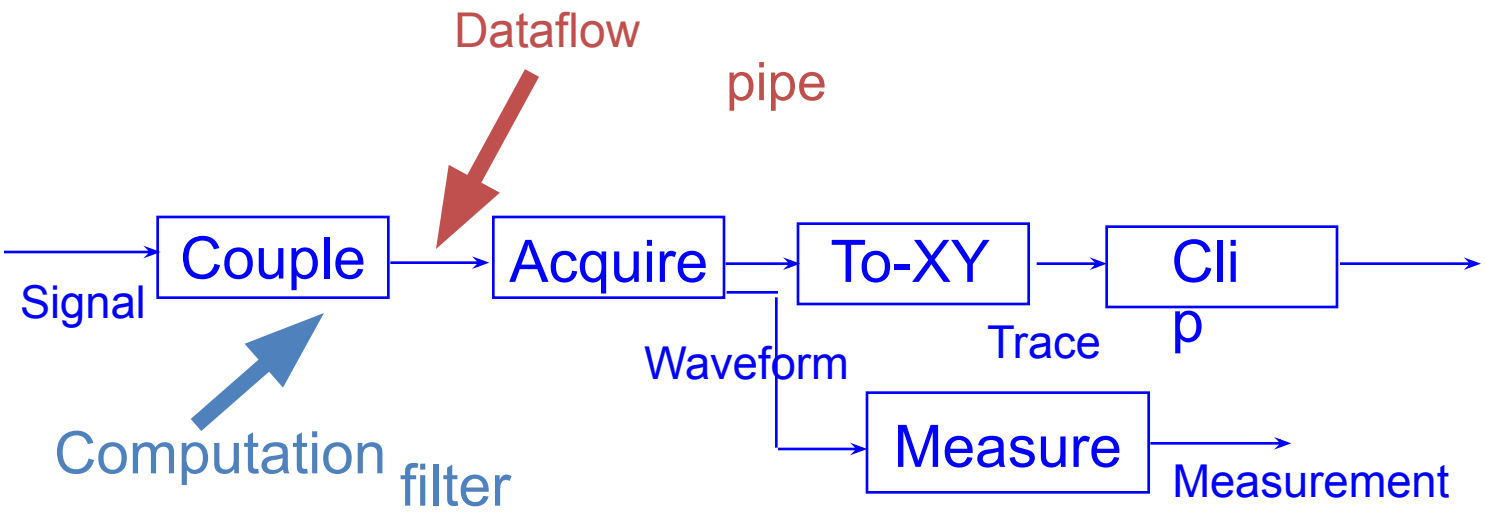
Hierarchical

- Layered
  - Interpreter

Styles: Questions to Address

- System Model
  - What is the overall organizational pattern?
- Structure
  - What are the basic components and connectors?
  - What topologies are allowed?
- Computation
  - What is the underlying computational model?
  - How is control and data transferred between components?
- Properties
  - Why is this style useful?
  - What kinds of properties are exposed?
- Analyses
  - What kinds of analysis does the style support?

Data Flow:
Pipes and Filters



Pipes and Filters

- Filter
  - Incrementally transform some amount of the data at inputs to data at outputs
    - Stream-to-stream transformations
  - Use little local context in processing stream
  - Preserve no state between instantiations
- Pipe
  - Move data from a filter output to a filter input
  - Pipes form data transmission graphs
- Computational Model
  - Run pipes and filters (non-deterministically) until no more computations are possible.
- Analysis
  - Functional composition; additive latencies; etc.

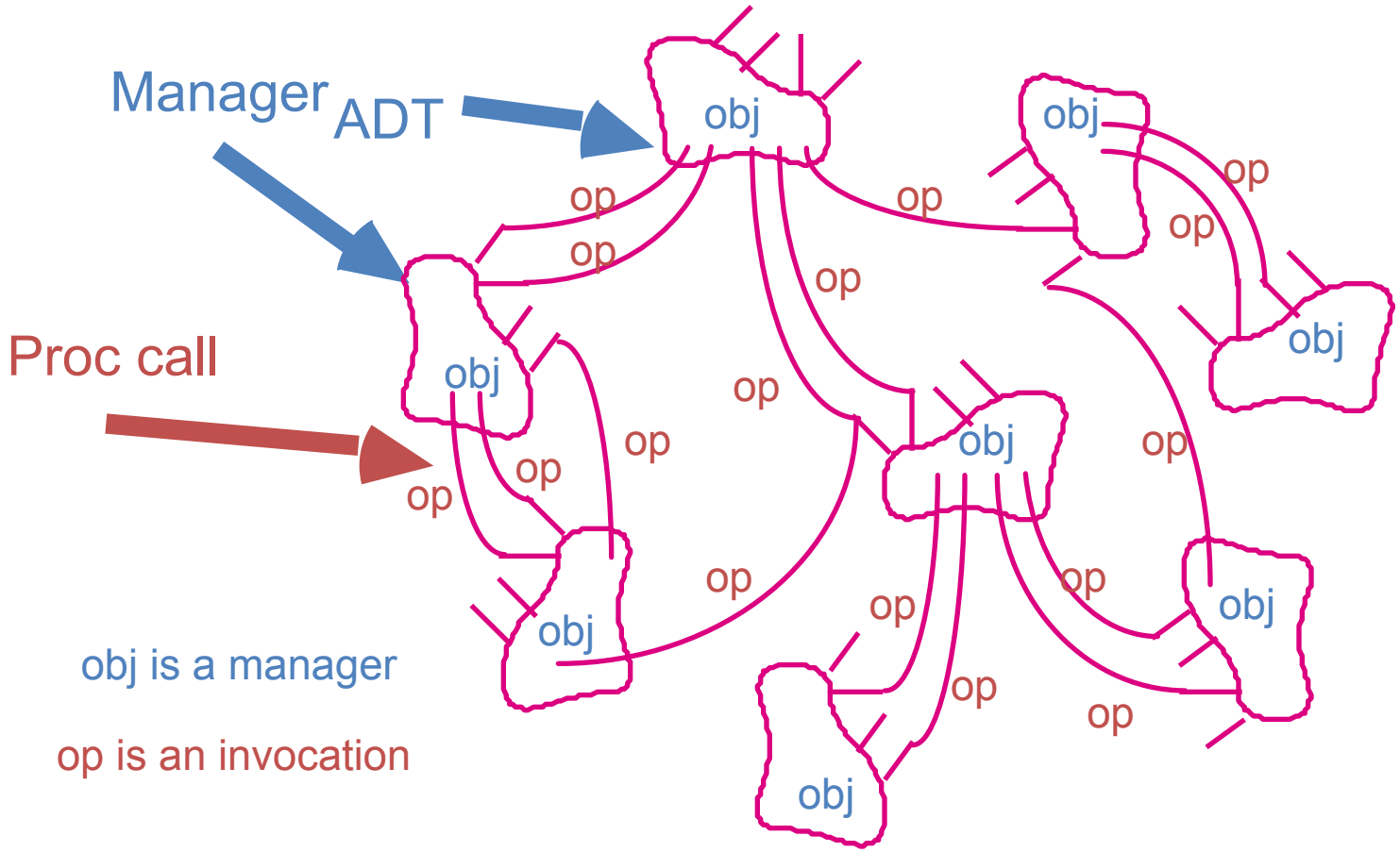
Properties

- Designer to understand the overall input/output behavior of a system.
- Reuse: any two filters can be hooked together.
- Maintain and enhance
- Permit specialized analysis, throughput and deadlock analysis.
- Concurrent execution.

Disadvantages

- Not good for handling interactive application.
- Maintain correspondence between two separate but related streams
- Added work for each filter to parse and unparse its data.

Call-Return: Object-Oriented



Call-Return Systems

- Objects
  - Encapsulate representations
  - Provide interfaces to access services
- Connectors
  - Call-ret
  - urn; service invocation
- Computational Model
  - Services requested from known service provider; Requester blocks
- Analysis
  - Correctness of a component depends on correctness of services it invokes

Properties

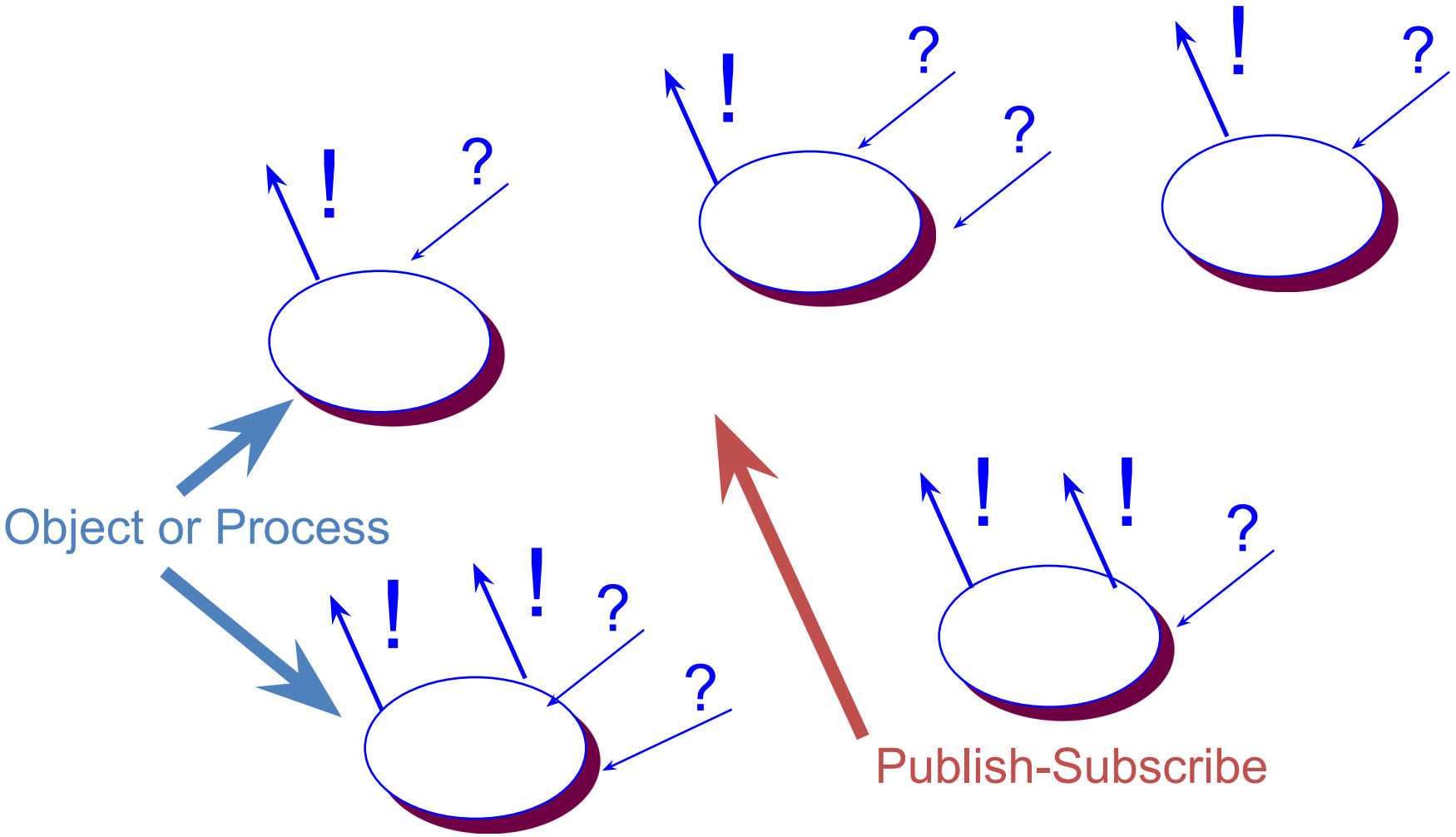
- An object hides its representation from its clients, it is possible to change the implementation without affecting those clients.
- Decompose problems into collections of interacting agents.

Disadvantages

- One object to interact with another it must know the identity of that other object.

- Side effect: if A uses object B and C also uses B, then C's effects on B look like unexpected side effects to A, and vice versa.

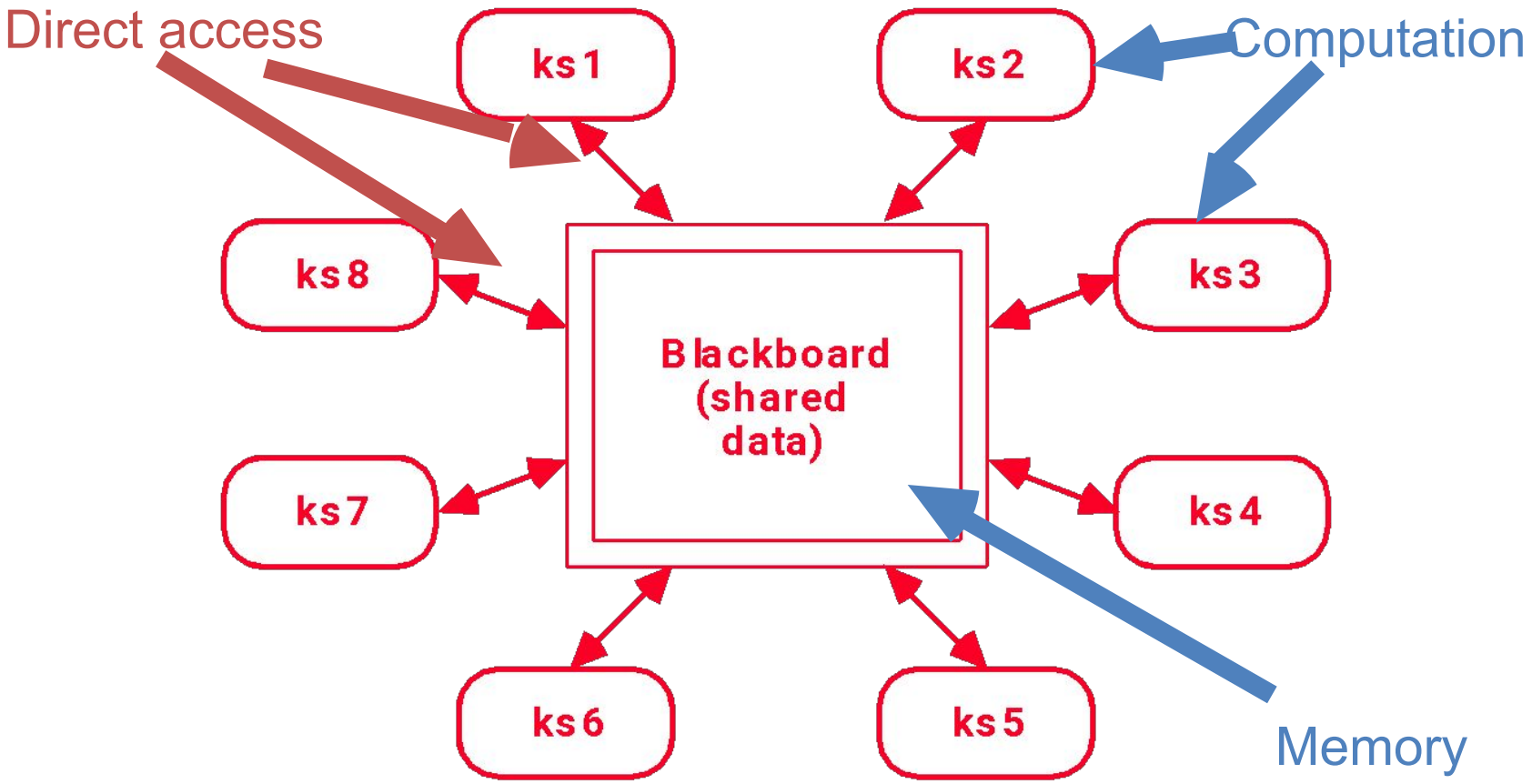
Loosely Coupled Components: Publish-Subscribe



Publish-Subscribe (Implicit Invocation)

- Components
  - Objects, processes
  - Have a set of methods
  - Announce (publish) events – via multicast
  - Subscribe to events by associating a procedure to call
- Connectors
  - Event space (bus)
  - Typically implemented as a dispatcher
- Computational model
  - When an event is announced invoke associated procedures (in any order)
  - Correctness of a component should not depend on correctness of components that subscribe to its announced events.

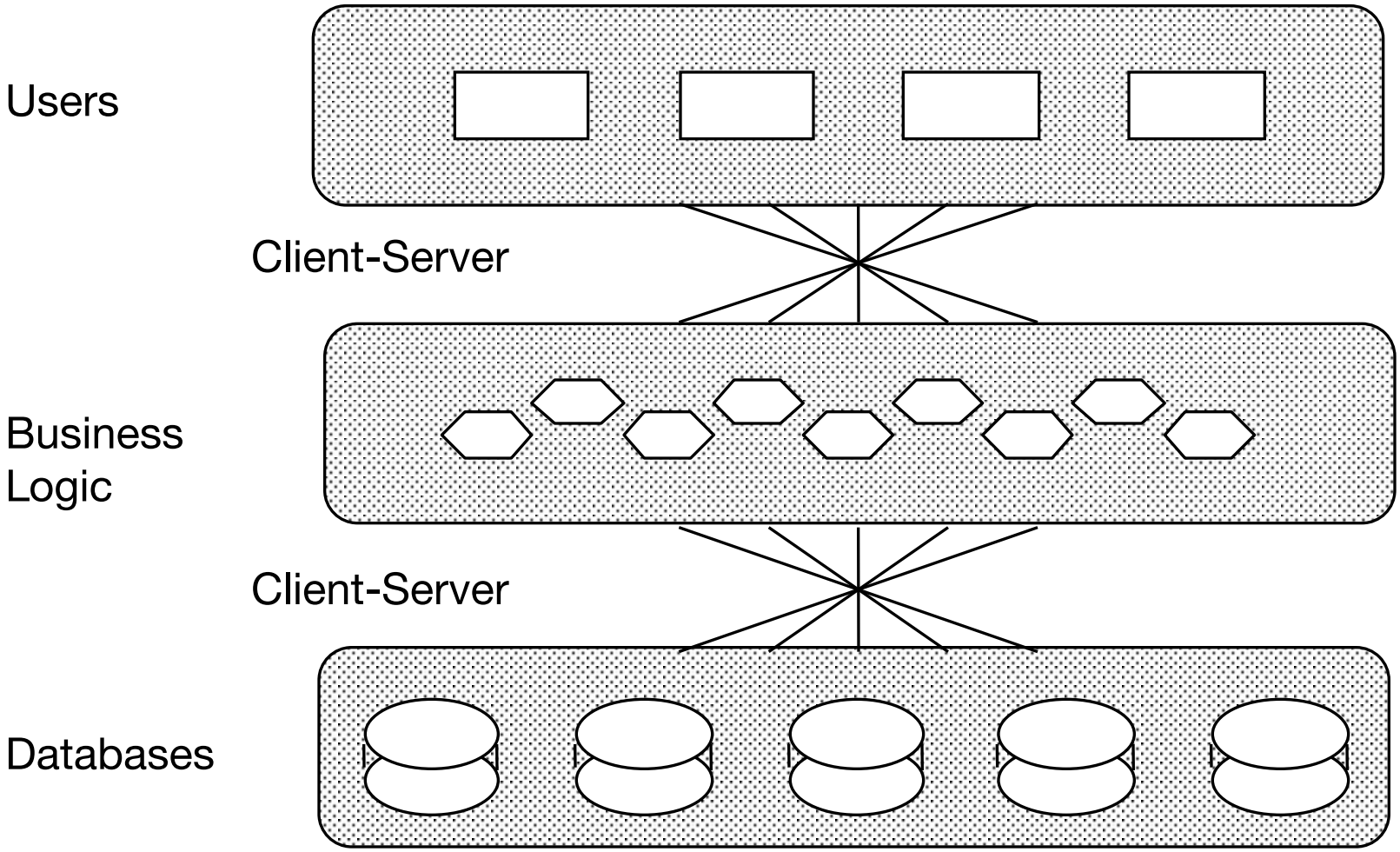
Data Oriented Repository: Blackboard



The Blackboard Model

- Knowledge Sources
  - World and domain knowledge partitioned into separate, independent computations
  - Respond to changes in blackboard
- Blackboard Data Structure
  - Entire state of problem solution
  - Hierarchical, non-homogeneous
  - Only means by which knowledge sources interact to yield solution
- Computational Model
  - Changes to data in blackboard by one knowledge source trigger the actions of other knowledge sources to create new blackboard data

Tiers:
3-Tiered Client Server



Many Others

- **Often styles are used in combination**
  - Example: each layer might be different style internally
  - Example: a component might have substructure defined in a different style than its surrounding
- **Many styles are closely tied to specific domains -- often by specializing a more generic style**
  - sometimes called component integration frameworks
  - N-tiered MIS Systems
  - OSI Protocol Stack
  - Instrumentation Systems
- **In many cases these are specialized to a particular product family,**
  - sometimes called a product line architecture