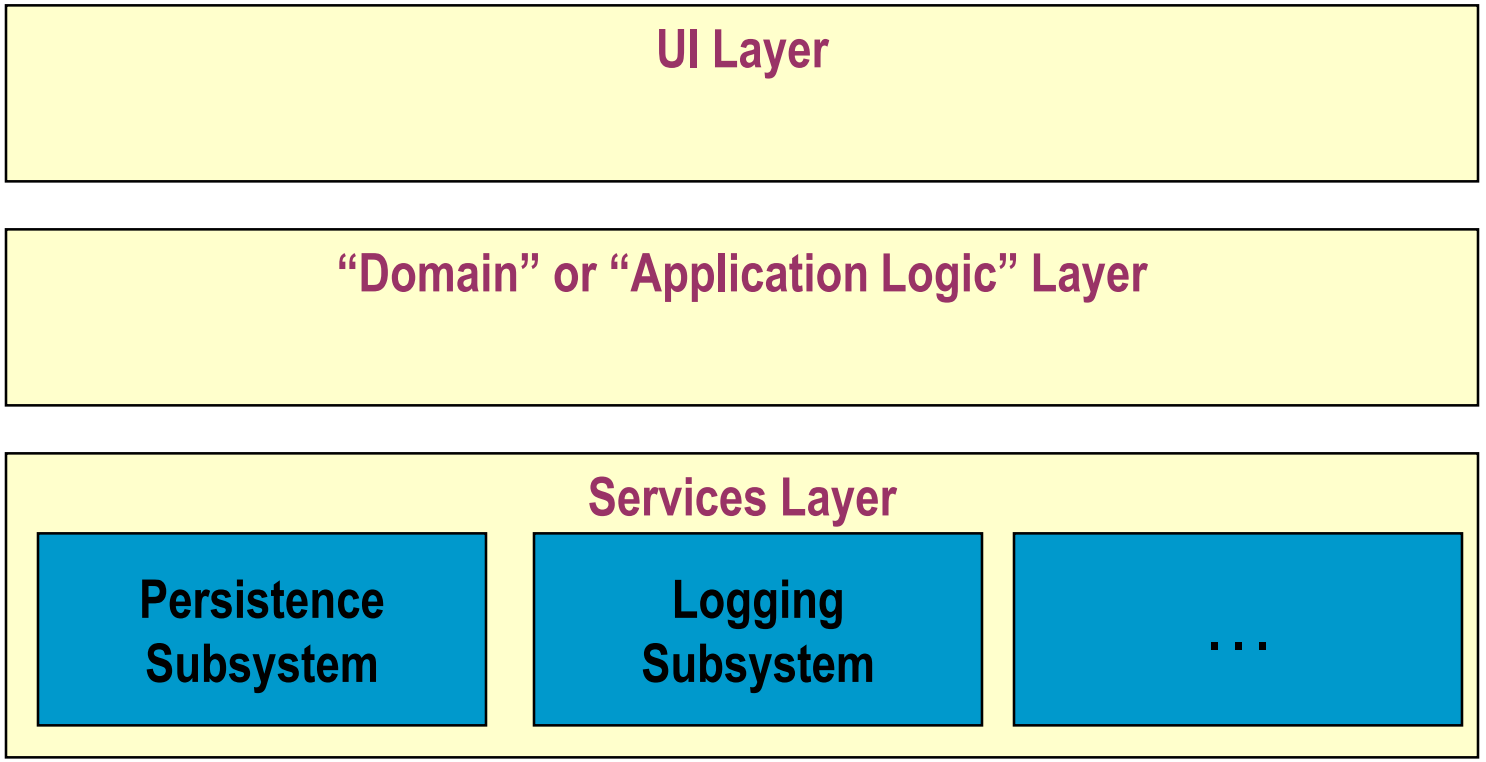
Advanced Software Design
Architecture & Patterns

Gary Stewart
gstewart@cs.uct.ac.za
slides taken from prof. edwin blake

LAYERED ARCHITECTURE

Advice: Multi-Tier Layered Architectures

- Separate presentation and application logic, and other areas of concern.



Layered Architecture

- Logical architecture organised into layers
  - Layer: very coarse-grained grouping of classes (or packages or subsystems) with cohesive responsibility for a major aspect of the system.
- Relationship between layers:
  - “higher” layers call services of “lower” layers
  - Strict Layered Architecture: layer uses only layer directly below
  - Relaxed: can use several lower layers
- Typical layers in an OO system:
  - User Interface
  - Application Logic and Domain Objects
  - Technical Services
    - Application-independent, reusable across systems

More Layers

- GUI windows
- reports
- speech interface
- HTML, XML, XSLT, JSP, Javascript, ...

UI
(**Presentation**, View)

- handles presentation layer requests
- workflow
- session state
- window/page transitions
- consolidation/transformation of disparate data for presentation

Application
(Workflow, Process, Mediation, Controller)

- handles application layer requests
- implementation of domain rules
- domain services (*POS*, *Inventory*) services may be used by just one application, but there is also the possibility of multi-application services

Domain
(Business, Appl Logic, Model)

- very general low-level business services used in many business domains
- E.g., *CurrencyConverter*

Business Infrastructure
(Low-level Business Services)

- (relatively) high-level technical services and frameworks
- *Persistence*, *Security*

Technical Services
(Technical Infrastructure, High-level Technical Services)

- low-level technical services, utilities, and frameworks
- *data structures, threads, maths, file, DB,*
- *and network I/O*

Foundation
(Core Services, Base Services, Low-level Technical Services/Infrastructure)

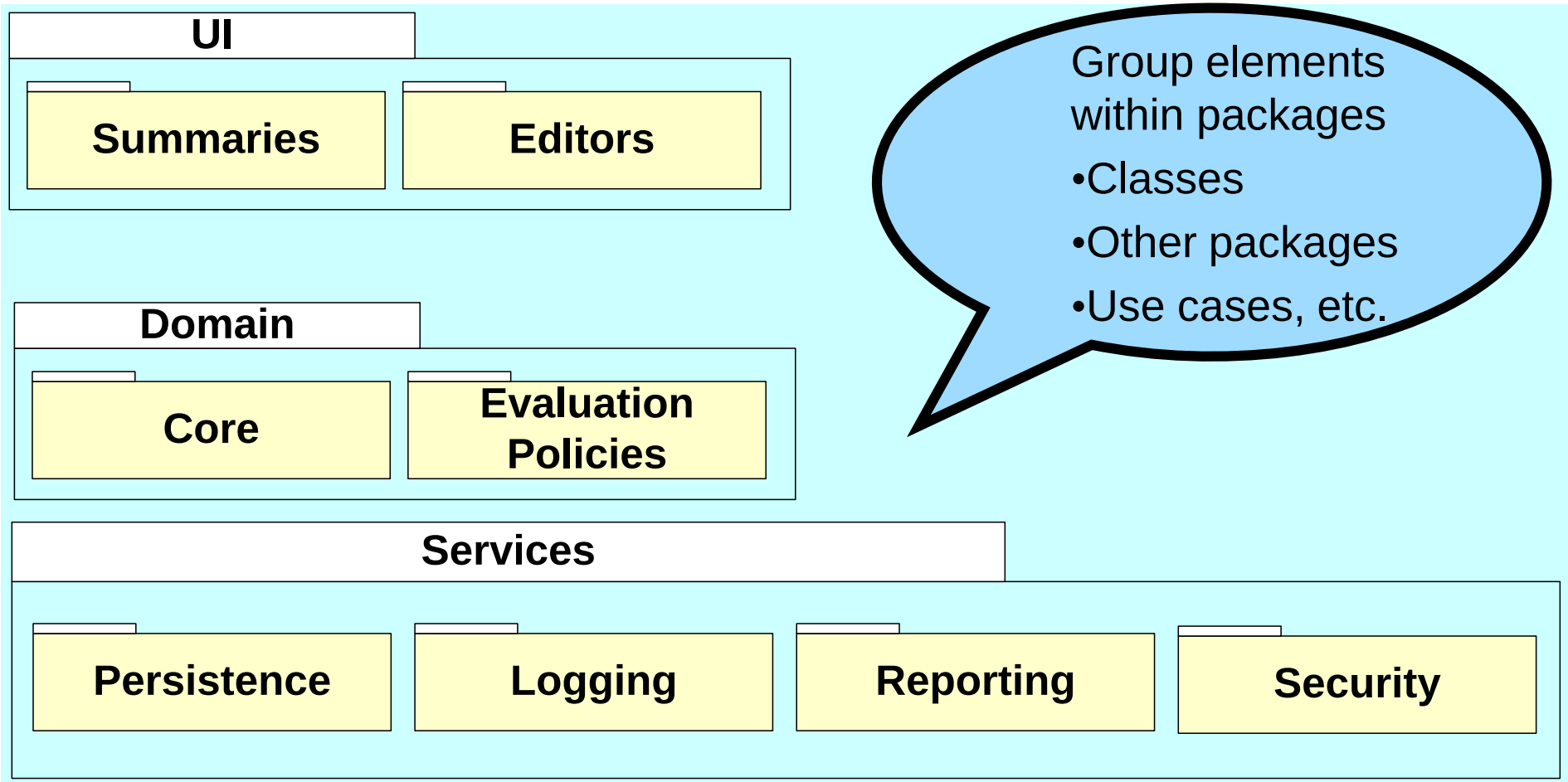
more application specific

dependency

width implies range of applicability

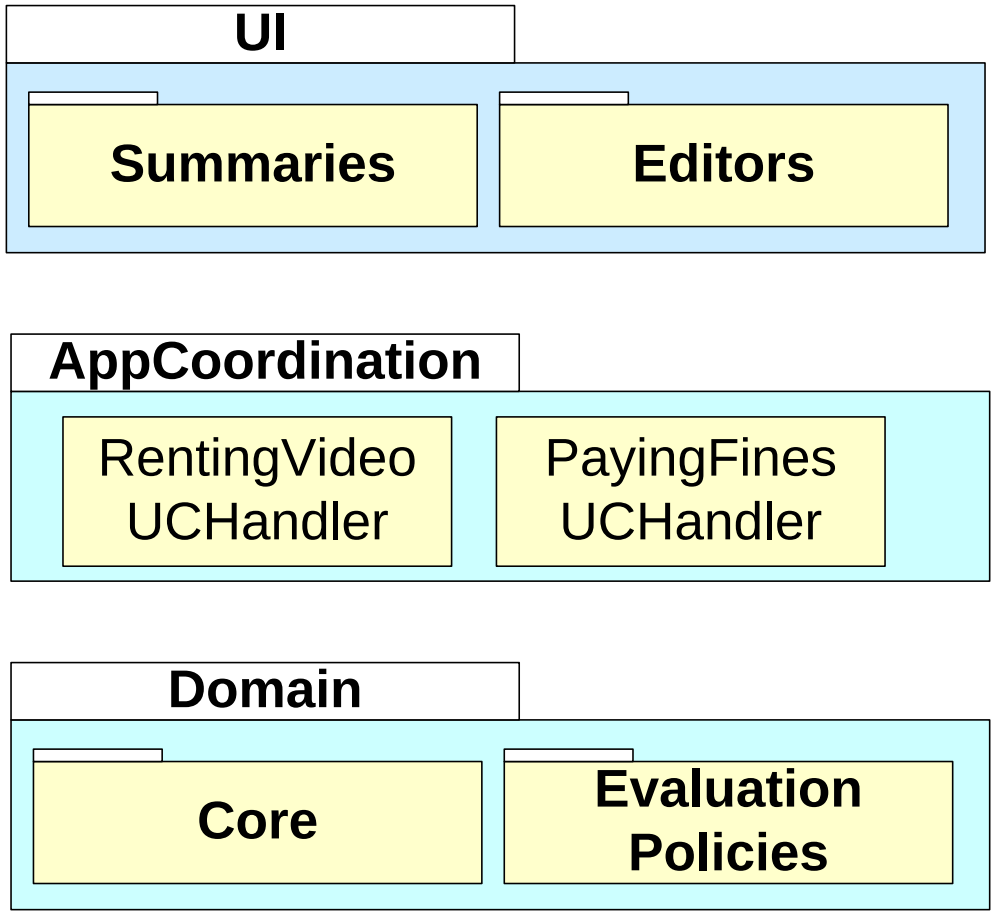
A Simple Logical Architecture

- In the UML, the logical partitioning is illustrated with package diagrams.



Advice: An Application Coordination Layer

- One strategy is to have “application coordination layer” whose objects represent use cases.
  - They may also hold session state.



Model-View Separation Principle

- Model ≡ domain layer
- View is the User Interface (UI) layer
- Model objects shouldn't have knowledge of View objects
- Don't assign domain responsibility to View objects
- Better separation of concerns with lower coupling
  - When domain layer independent of UI layer, domain layer can be used with another UI layer
- Facilitates multiple views of a given Model
  - Example: Application and web UIs for a given application

More Benefits of Layered Architecture

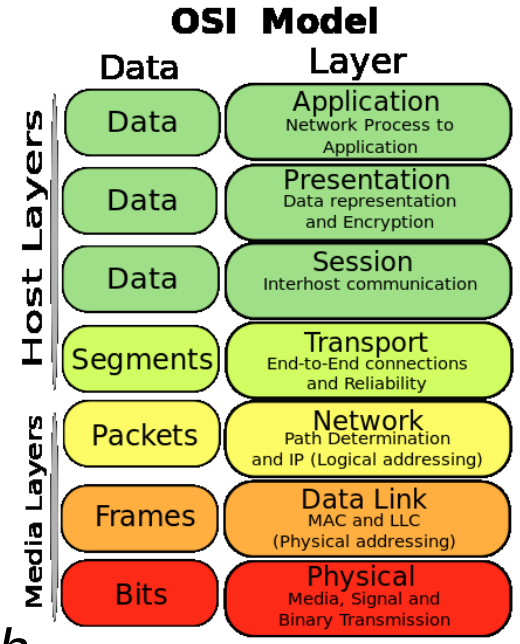
- System easier to comprehend: each layer has specific purpose
- Facilitates ‘high-cohesion’ within layers and ‘low-coupling’ between layers
- Prevents source code changes from rippling throughout system
  - Code changes should be confined within a specific layer
- Minimizing coupling makes it easier to swap out a technical service in a lower layer and replace it with another technology
  - Replacing RDBMS without major code re-write in application logic
- Lower layers contain more reusable functionality

Reference Architectures

- Idealized way of discussing and comparing domain-specific architectures.
- They do not represent actual real systems
- Reference architecture features make them easier to describe and understand.
  - A reference model provides a vocabulary for comparison.
  - It acts as a base against which systems can be evaluated.

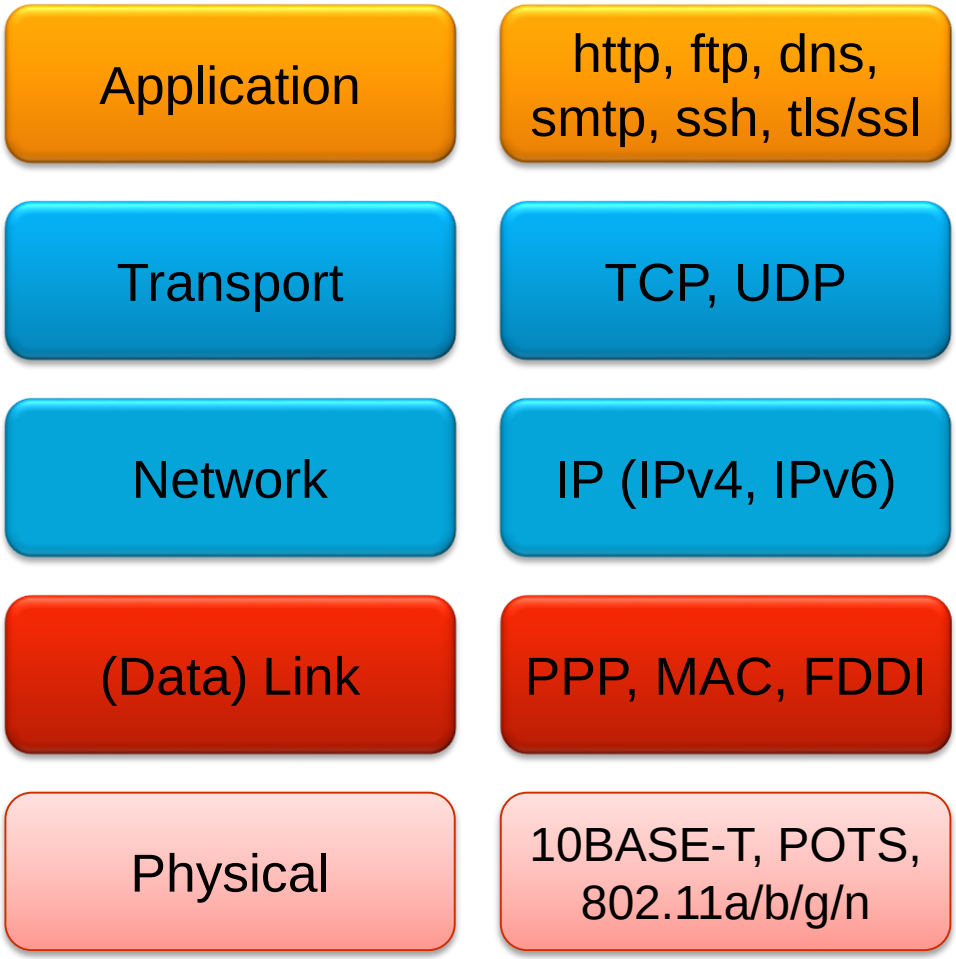
OSI Model for Distributed Systems

- OSI stack is seven-layer model for open systems interconnection.
  - Lower layers: physical interconnection,
  - Middle layers: data transfer
  - Upper layers: semantically meaningful application information.
- Allow conformant systems to communicate with each other.
- Each layer should depend on the layer below.
- *Performance problems with this layered approach*.
  - Vast physical differences between networks
- Well defined functional characteristics but not the non-functional.
- Developers implement their own higher-level facilities and skip layers in the model.
- See TCP/IP …



Internet Stack (TCP/IP Model)

- Internet stack is similar layered networking stack to OSI
- Pre-dates OSI Stack



Layering Considered Harmful!?

- Networking layering ⇒ functions of each layer are carried out completely before protocol data unit is passed to the next layer
  - Optimization of each layer has to be done separately
  - Hides information that lower layers may need to optimize performance
  - Layered model (TCP/IP & ISO OSI) causes conflict:
    - Layer N may duplicate lower level functionality (hop-hop error recovery vs end-to-end error recovery)
    - Layers may need the same information (time stamp)
    - Layer N may need layer N-2 information (lower layer packet sizes)
- Increased layering ⇒ increased complexity (via inter-layer dependencies)
  - "It is always possible to agglutinate multiple separate problems into a single complex interdependent solution. In most cases this is a bad idea."

Conclusion: horizontal separation may be more cost-effective and reliable than vertical

Basic Web 2.0 Reference Architecture Diagram

Resource tier: Capabilities or backend systems (data or processing) that support services consumed over the Internet:

Service tier: packages resources as a service and controls what goes in and out.

- Application servers deploying SOAP, EJB, PHP, Rails, ASP, …

Connectivity: means to reach service

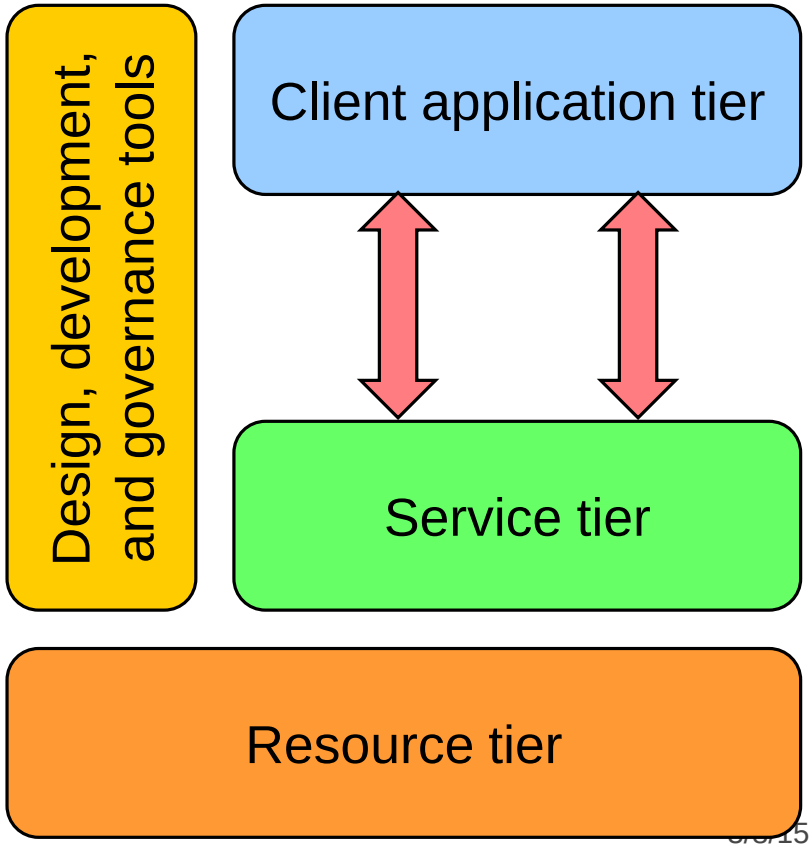
- Standards/protocols like XML over HTTP

Client tier: user view of services

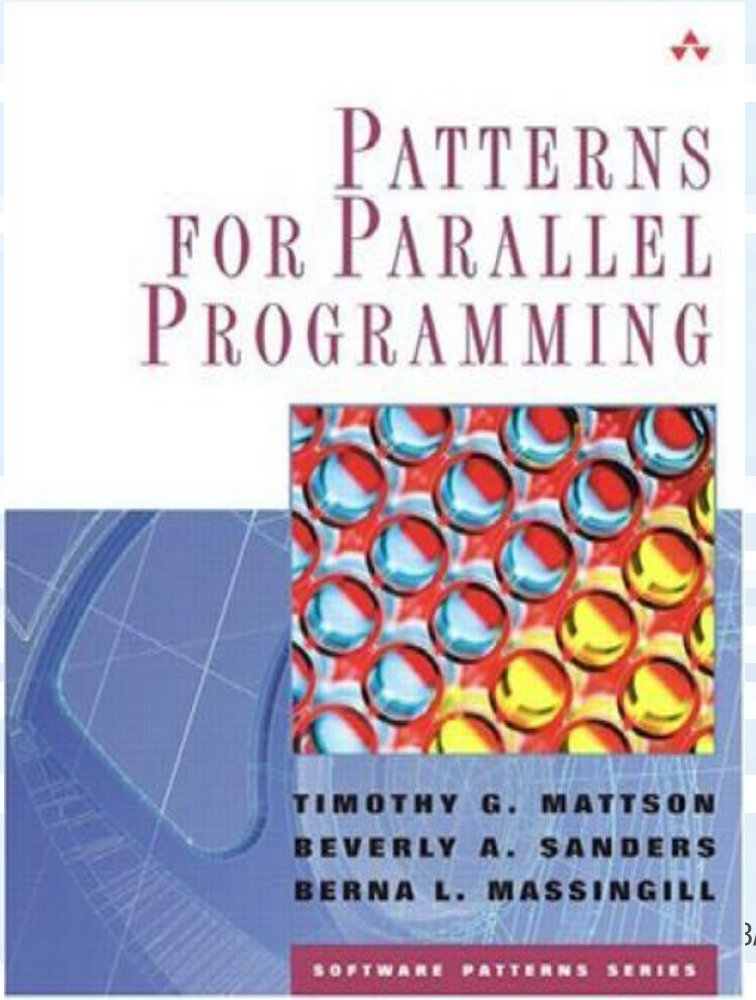
- Web browsers, Flash, Acrobat, iTunes, …

Design, development, and governance tools to build web apps

- Adobe Dreamweaver and Apple's tools (xCode, DashCode), other IDEs



PATTERNS AND CONCURRENCY



Parallel or Concurrent Programming

- The way to go (once again)
  - Multi-core
  - Server farms
- Up to now ***processing*** speedup has been due to ***processor*** speedup (Moore's law), but this is coming to an end
- Solution is parallel (or concurrent) processing:
  - A parallel program is the specification of a set of processes executing simultaneously, and communicating among themselves in order to achieve a common objective
- This is a very big, very tough, and increasingly important topic.
  - consider a few patterns that can potentially allow parallel execution.

Repository (or Blackboard)

- Suitable for applications in which the central issue is establishing, augmenting, and maintaining a complex central body of information.
- Typically the information must be manipulated in a wide variety of ways.
- An example of a repository or blackboard architecture may the management of work allocation process in a company.

Parallel Agent and Repository

- Data-centred concurrency: autonomous agents update state managed in a central repository.

Repository (Blackboard) of the resulting creation that is shared by all agents

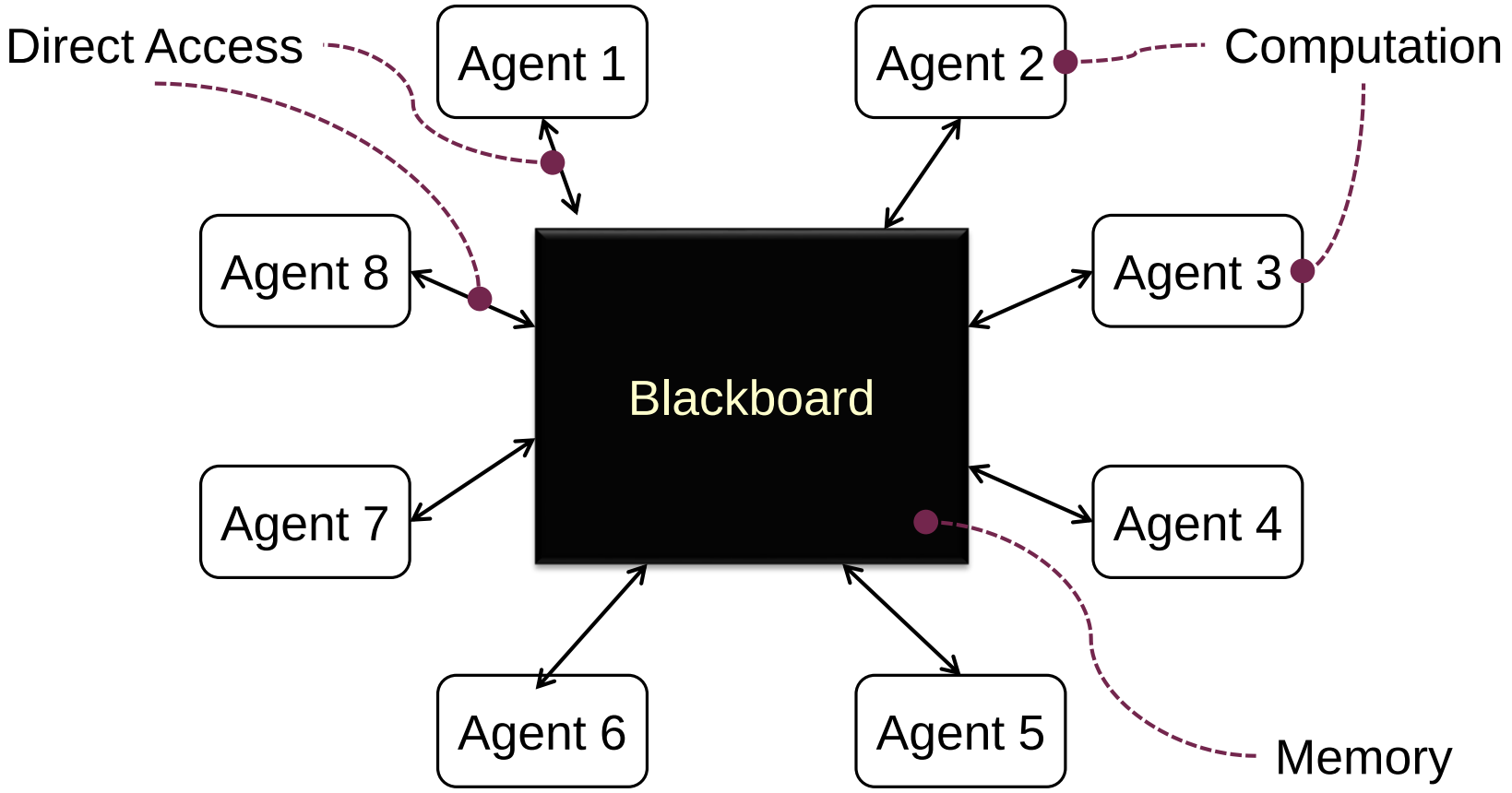
- Logical structure may be in a shared memory in a multi-core machine
- Can be distributed amongst agents as a block it "owns"

Agents: intelligent agents that will act on blackboard

Controller: orchestrates agents access to the repository and creation of the aggregate result

- Can reside either in the agents or the blackboard

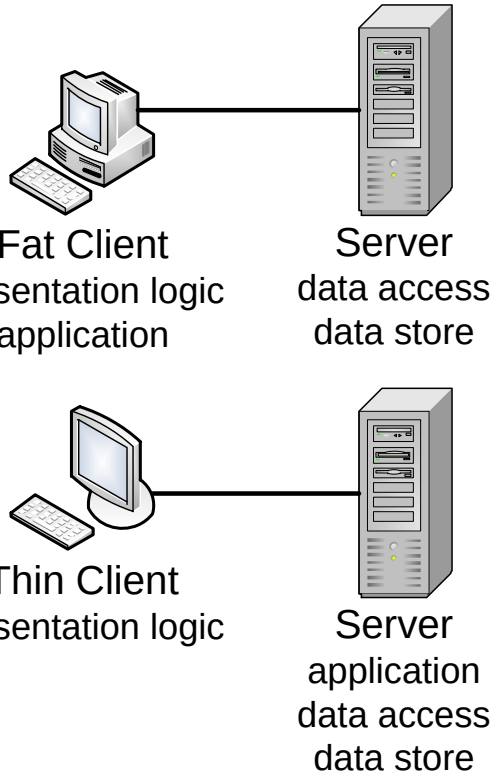
Repository — Blackboard



Client-Server Architecture

- Distributed system model which shows how data and processing is distributed across a range of components.
  - Can provide coarse grained parallelism.
  - Can be implemented on a single computer.
- 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.

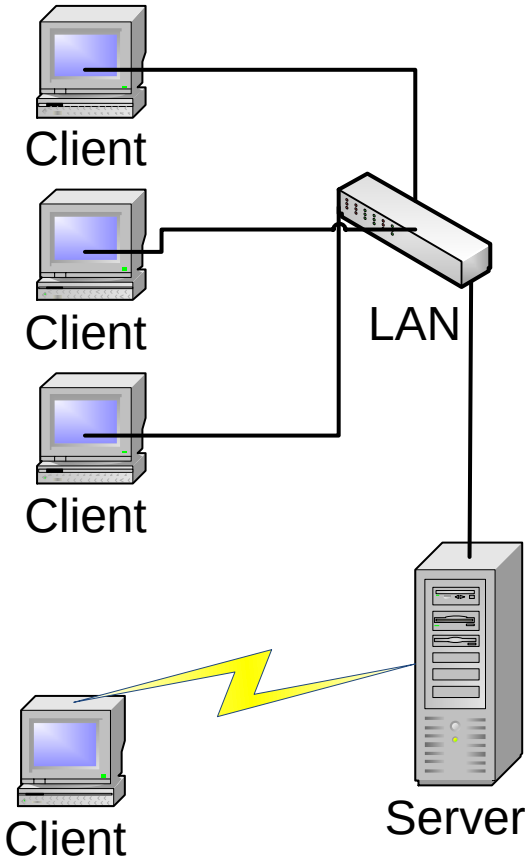
Client-Single Server



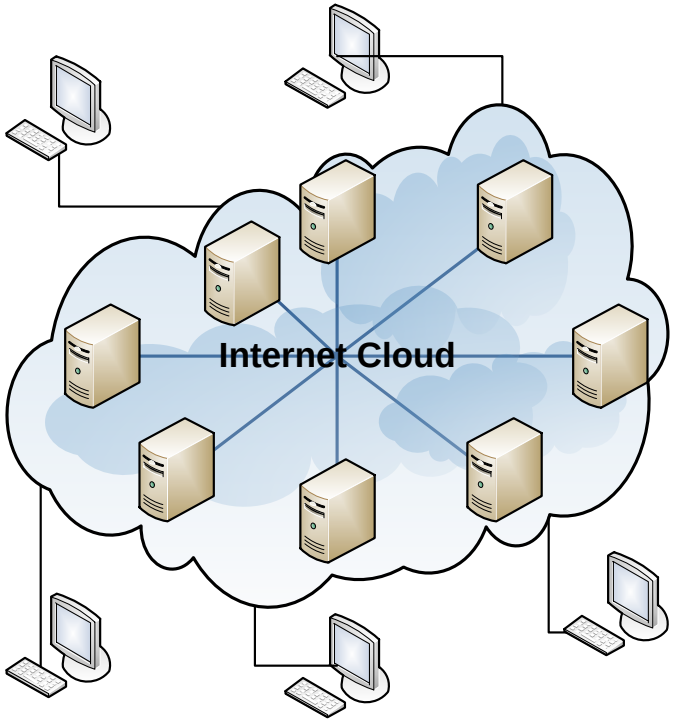
A client (such as email) requests a service from a server (email-host).

The server generally runs on a large central device.

- Single point of failure in server
- Communications bottleneck

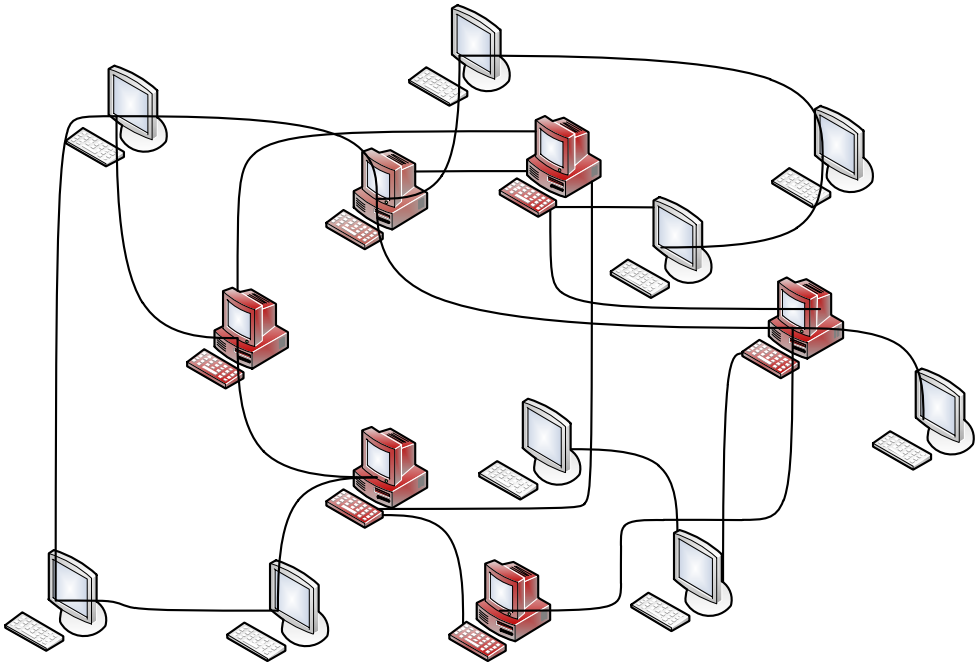


Client-Server and Peer-to-Peer Architecture



Internet Mediated Client-Server

- dominant architecture for internet
- add redundancy (server-farms or cloud)
- vulnerable to attack (denial of service)



Two-Tier Peer-2-Peer Architecture

- many devices perform server-functions
- any device that is acting as a client is able to find servers (with the assistance of other servers)

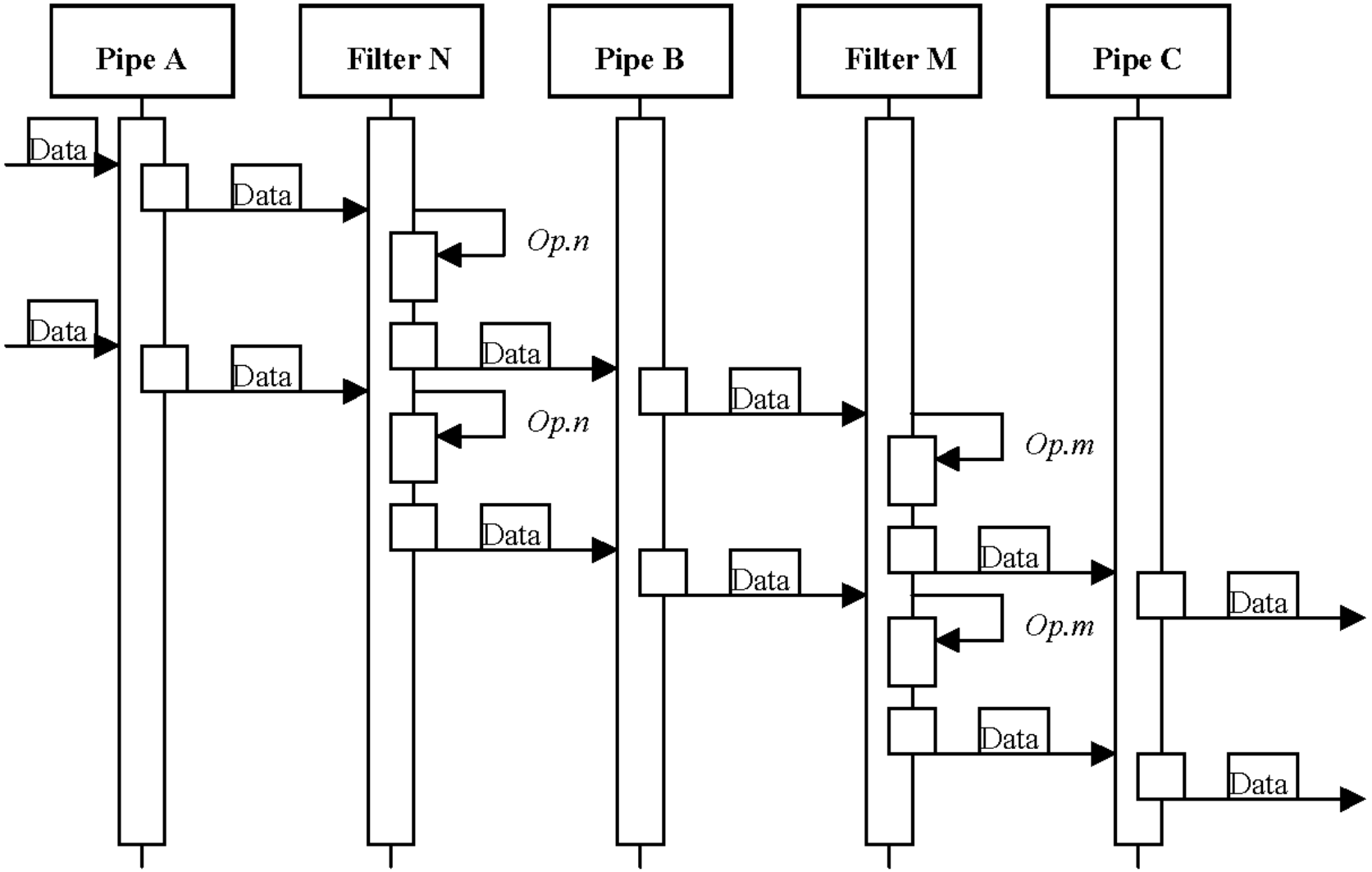
Pipeline or Pipe and Filter Pattern

- Suitable for applications that require a defined series of independent computations to be performed on ordered data.
  - Very useful if each computation can be done incrementally on the data
    - Then computations can proceed in parallel.
- The pattern attempts to decompose the problem into a set of computations, or filters, with operations, called pipes to stream data from one process to another.
  - the filters interact only via pipes.
  - “pure” filters have local processing and little state.

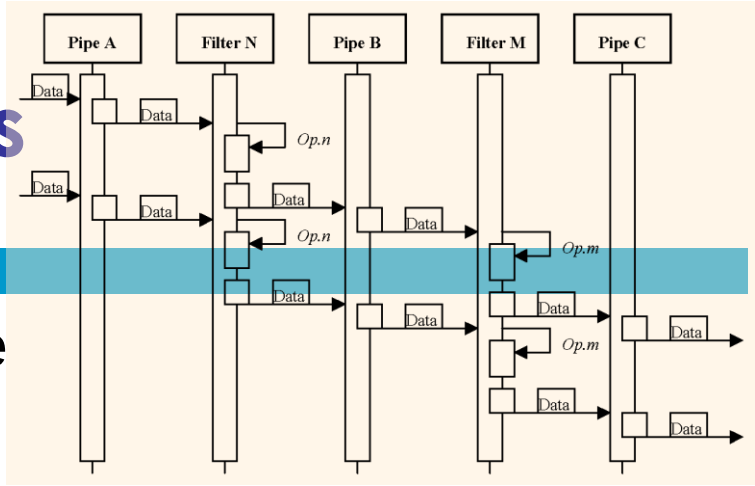
Pipes and Filters in Parallel

- The components of the pattern execute in parallel
- All filters and pipes are simultaneously active:
  - they accept the data,
  - but only filters operate on them,
  - and send them to the next step.
- Pipes synchronize the activity between filters

Parallel Pipe and Filter Diagram



Scenario of Pipes and Filters



- **Pipe A** receives data from a Data Source or another previous filter, synchronising and transferring it to the **Filter N**.
- **Filter N** receives the package of data, performs operation *Op.n* on it, and delivers the results to **Pipe B**.
  - At the same time, new data arrives to the **Pipe A**, which delivers it as soon as it can synchronise with **Filter N**.
  - **Pipe B** synchronises and transfers the data to **Filter M**.
- **Filter M** receives the data, performs *Op.m* on it, and delivers it to **Pipe C**, which sends it to the next filter or Data Sink.
  - Simultaneously, **Filter N** has received the new data, performed *Op.n* on it, and synchronised with **Pipe B** to deliver it.
- The previous steps are repeated over and over until no further data is received from the initial Data Source or previous filter.

CONCLUSION

Software Architecture

Introducing Patterns

Key Architecture Patterns

Layered Architecture

Patterns and Concurrency

Conclusion

Realizing Non-functional Requirements

Performance

- Localize critical operations and minimize communications.
- Use large rather than fine-grain components. Allow for concurrency.

Security

- Use a layered architecture with critical assets in the inner layers.

Safety

- Localize safety-critical features in a small number of sub-systems.

Availability

- Include redundant components and mechanisms for fault tolerance.
- Allow replacement of components without stopping the system.

Maintainability

- Use fine-grain, replaceable components.
- Avoid global/shared data structures.

Architecture Diagrams

Simple Block Diagrams

- Simple, informal block diagrams showing entities and relationships
- Most frequently used method.
- Good for communicating at a high level of abstraction.

Diagrams with Rich Semantics

- Do not show the types of relationships between entities
- Do not show visible properties of entities (interfaces)
- Costly to produce, time consuming

Purpose of Architecture: beforehand

- Software architecture can serve as a design plan for the negotiation of system requirements
- A means of structuring discussions with clients, developers, and managers.
- An essential tool for complexity management, hiding details, allowing the designers to focus on the key system abstractions.

Use of Architectural Models after Development

- A way of documenting an architecture that has been designed
  - To produce a complete system model showing the system components, their interfaces and their connections.
- For later maintenance and enhancements
- For other developers needing to integrate with your system