Software design: architectural design

- Packages, sub-systems and models
- Architectural styles

CE202 Software Engineering, Autumn term
Dr Michael Gardner, School of Computer Science and Electronic Engineering, University of Essex

Last week

- ⋯ we looked at the detailed design of the system
- This week we will look at **system design**, and specifically the high-level architecture of the system
  - structure of sub-systems
  - distribution of sub-systems on processors
  - communication between sub-systems

Subsystems

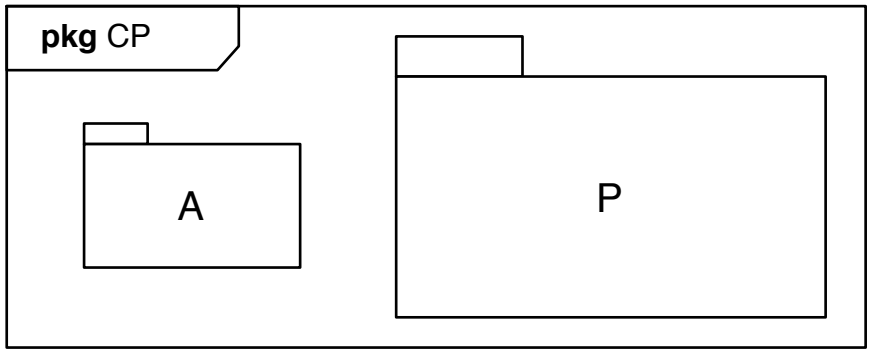
- A subsystem typically groups together elements of the system that share some common properties
- An object-oriented subsystem encapsulates a coherent set of responsibilities in order to ensure that it has integrity and can be maintained
- For example, the elements of one subsystem might all deal with the human–computer interface, the elements of another might all deal with data management and the elements of a third may all focus on a particular functional requirement.

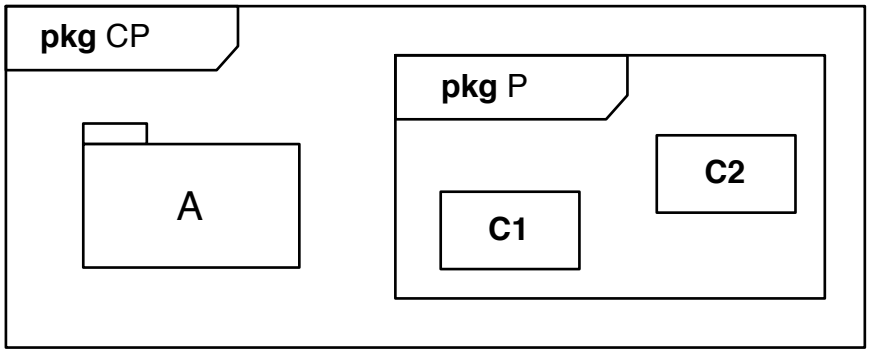
Subsystems

- The subdivision of an information system into subsystems has the following advantages
  - It produces smaller units of development
  - It helps to maximize reuse at the component level
  - It helps the developers to cope with complexity
  - It improves maintainability
  - It aids portability

Nested Diagrams

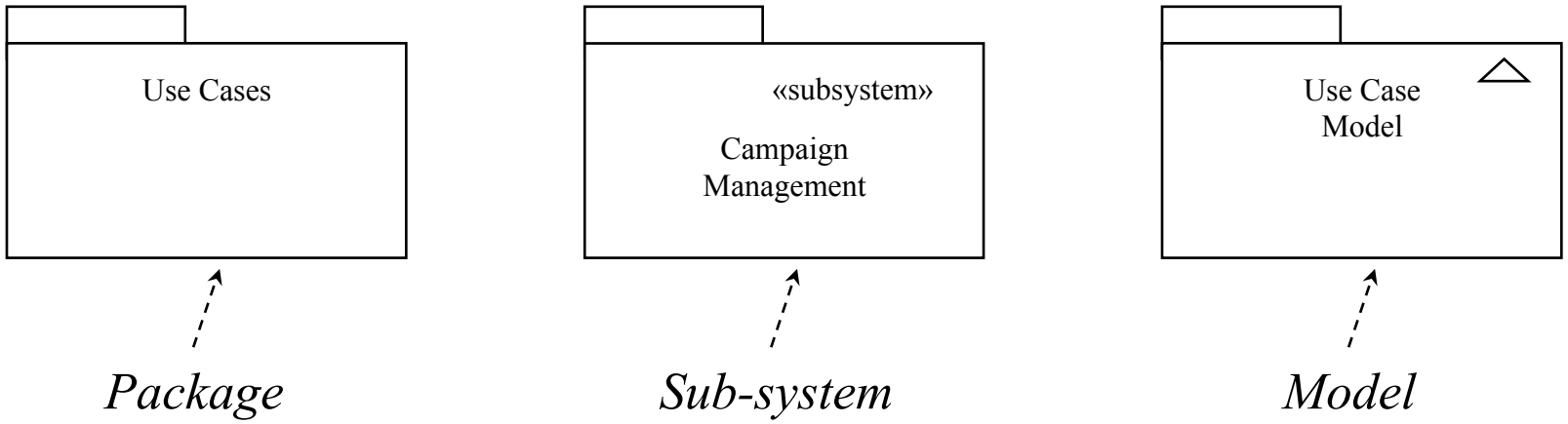
- UML 2 supports nested diagrams
  - e.g. an activity diagram inside a class
- The frame concept visually groups elements that belong to one diagram



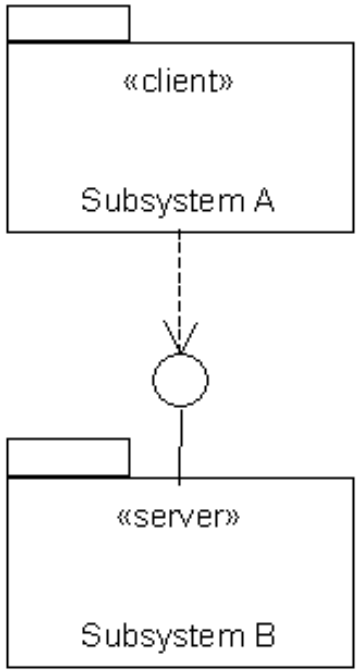


Packages, Sub-systems and Models

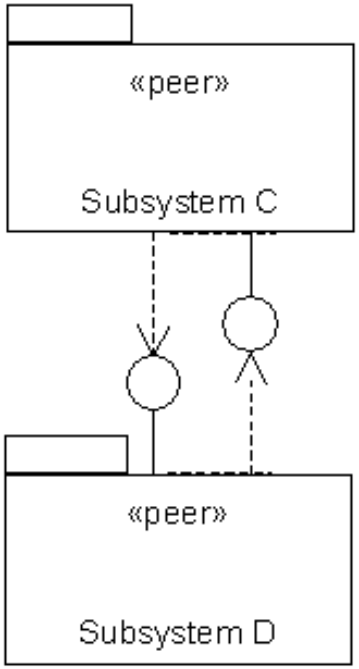
- UML has notation for showing subsystems and models, and also for packages, which are a mechanism for organising models (e.g. in CASE tools)



Styles of communication between subsystems



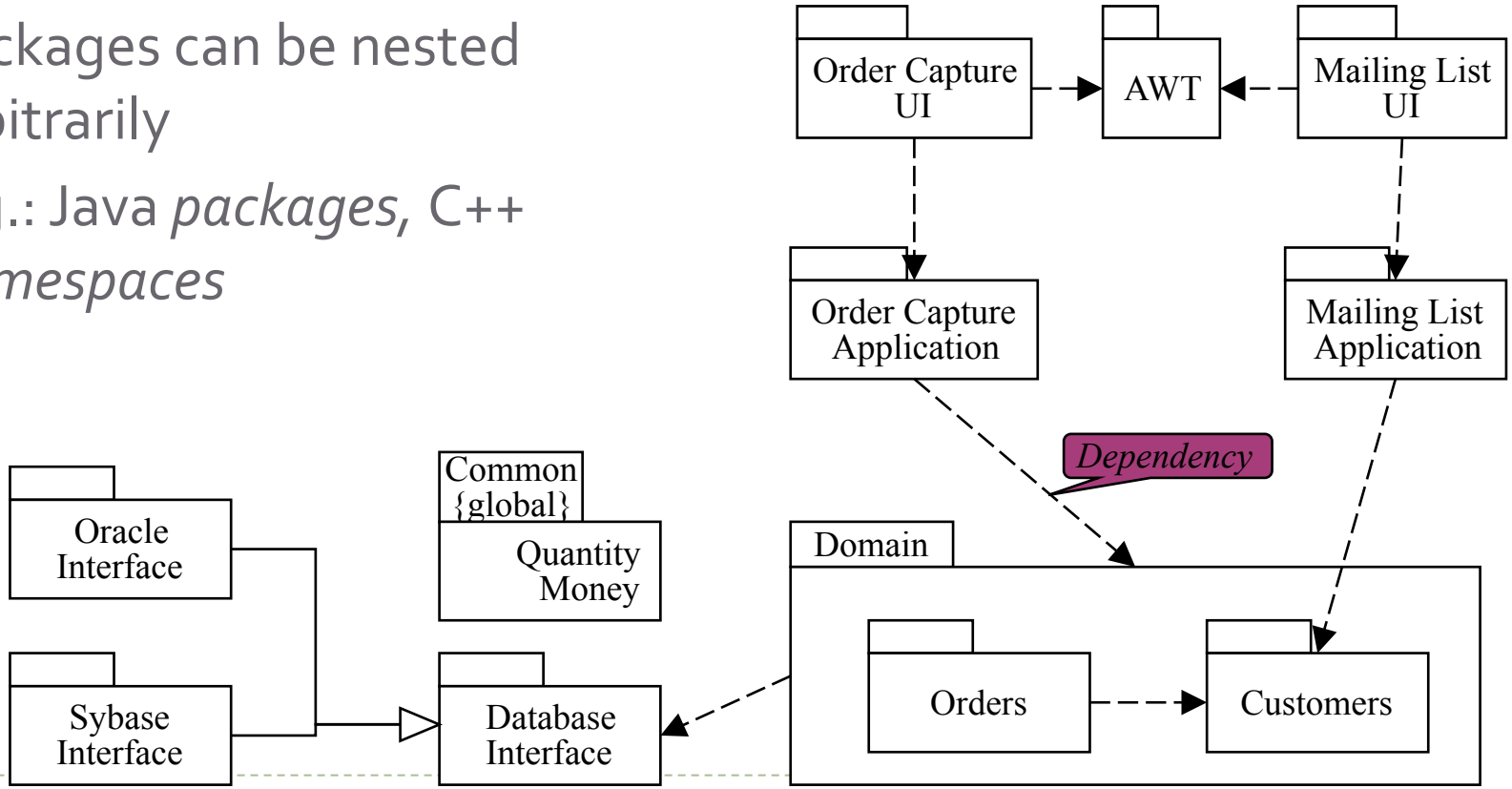
The server subsystem does not depend on the client subsystem and is not affected by changes to the client's interface.



Each peer subsystem depends on the other and each is affected by changes in the other's interface.

System design modelling technique: Package Diagrams

- Depicts the dependencies between the packages that make up a model.
- Package: A module containing any number of classes
  - Packages can be nested arbitrarily
  - E.g.: Java *packages*, C++ *namespaces*



Last week we looked at

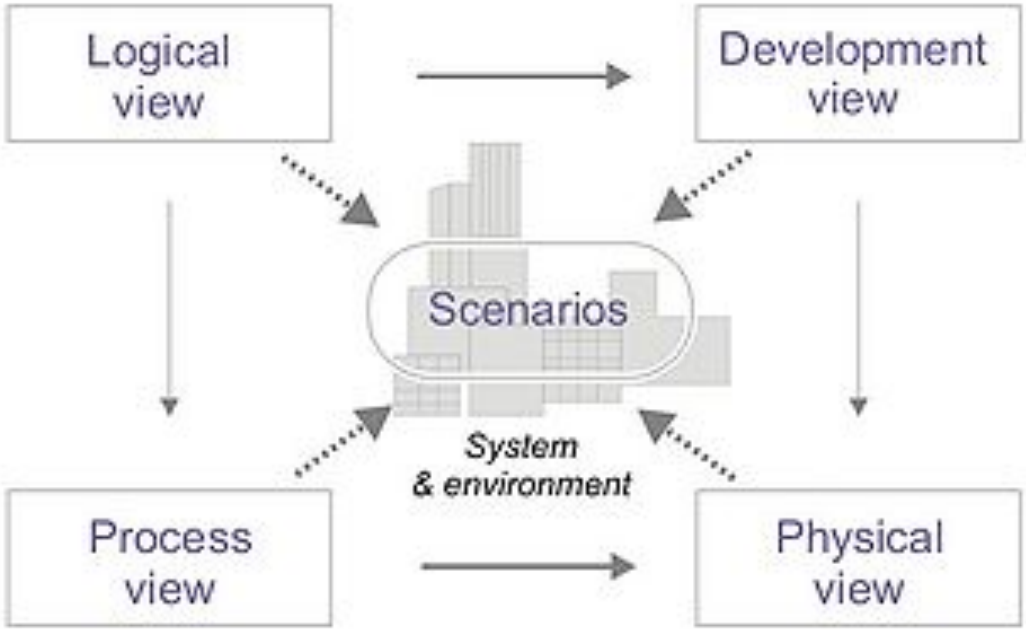
- Difference between **Logical** and **Physical** Design
- Difference between **System** Design and **Detailed** Design

- One way of bringing these together to describe the architecture of software-intensive systems is the **4+1 Architectural View (next slide)**

The 4+1 architectural view model

4+1 is a view model for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views" [Kruchten, 1995]

Selected use cases or scenarios are utilized to illustrate the architecture serving as the 'plus one' view



The 4+1 architectural view UML models

- **Logical view**: concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include **Class diagram, Communication diagram, Sequence diagram**.
- **Development view:** illustrates a system from a programmer's perspective and is concerned with software management (also known as the implementation view). It uses UML **Component and Package diagrams**.
- **Process view**: deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behaviour of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the **Activity diagram**.
- **Physical view:** depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components (also known as the deployment view). UML Diagrams used to represent physical view include the **Deployment diagram**.
- **Scenarios** : The description of an architecture is illustrated using a small set of use cases, or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as **use case view**.

System Architecture in UML
ADL

- Architecture Description Language
  - UML 2.0 added or changed features to support modeling architecture
  - Package diagrams (see previous slides)
  - Component diagrams (not covered)
  - Composite structure diagrams (not covered)
  - Deployment diagrams (not covered)

Architectural Styles

Components and connectors
Garlan & Shaw's catalogue

Software architecture: definition

- **[Garlan & Shaw 93]:**

  "… a level of design concerned with issues … beyond the algorithms and data structures of the computation; designing and specifying the overall system structure emerges as a new kind of problem."

  "Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives."

System Architecture
Key Definitions

- *System* is a set of components that accomplishes a specific function or set of functions.
- *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.
- *Architectural Description* is a set of products that document the architecture.

System Architecture
Key Definitions

- *Architectural View* is a representation of a particular system or part of a system from a particular perspective.
- *Architectural Viewpoint* is a template that describes how to create and use an architectural view. A viewpoint includes a name, stakeholders, concerns addressed by the viewpoint, and the modelling and analytic conventions.

(Garland & Anthony, 2003 & IEEE, 2000)

What is an Architectural Style?

"*An architectural style defines a family of systems in terms of a pattern of structural organization.*" [Shaw & Garlan 93]

Architectural style: describe –

- Types of **components**
- Types of **connectors**
- **Topology** (constraints on possible compositions)

Example: Shared Repository

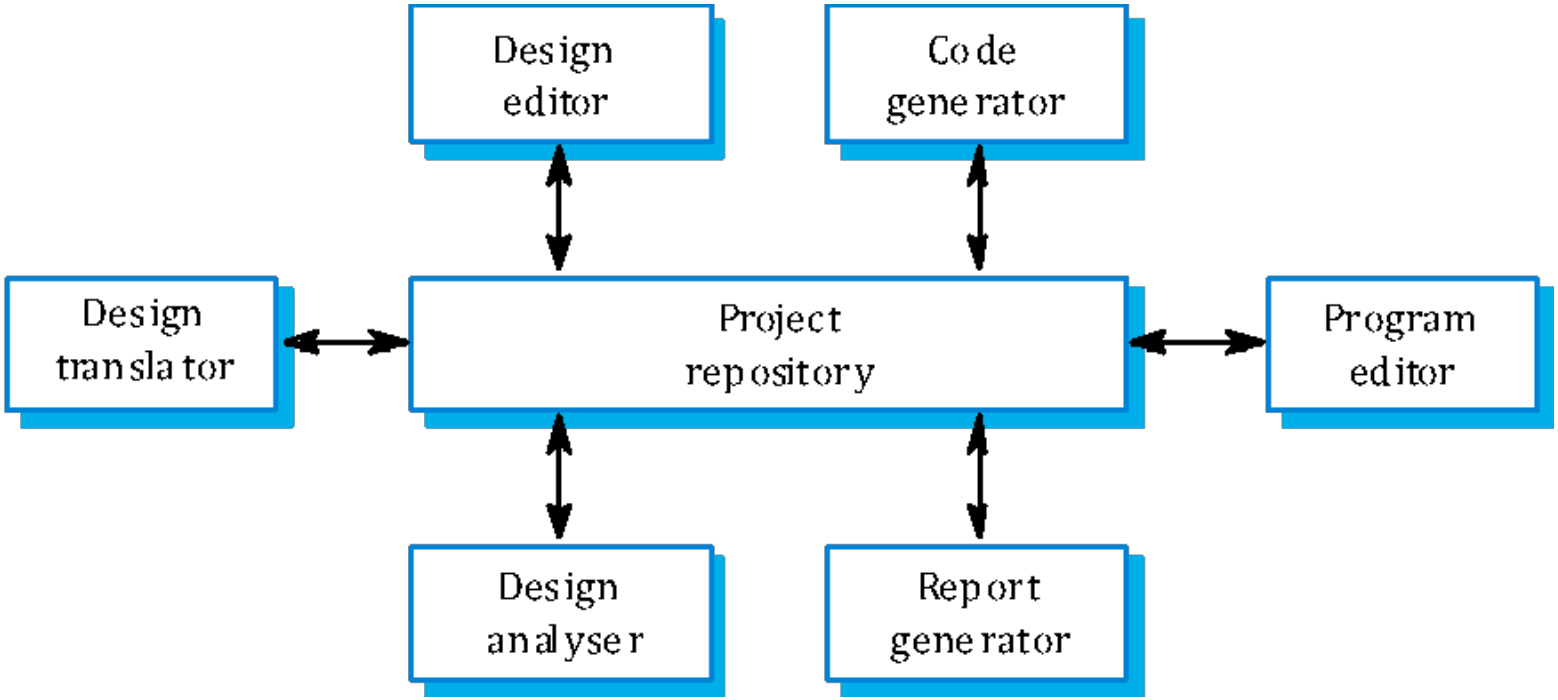
A central database is shared by different subsystems

Applicability:

- Large, complex body of information
  - Inter-related records
- Sophisticated data processing and management
  - Selection queries, recovery, locking, security …
- Distributed systems with shared data

Shared Repository: Example

- CASE Toolset



Shared Repository: Discussion

- Pros:
  - Efficient use of resources
  - Consistency maintained
  - Uniform data format
  - Centralized control
    - Security is simpler
- Cons:
  - Evolution is difficult
  - Dependency on data traffic

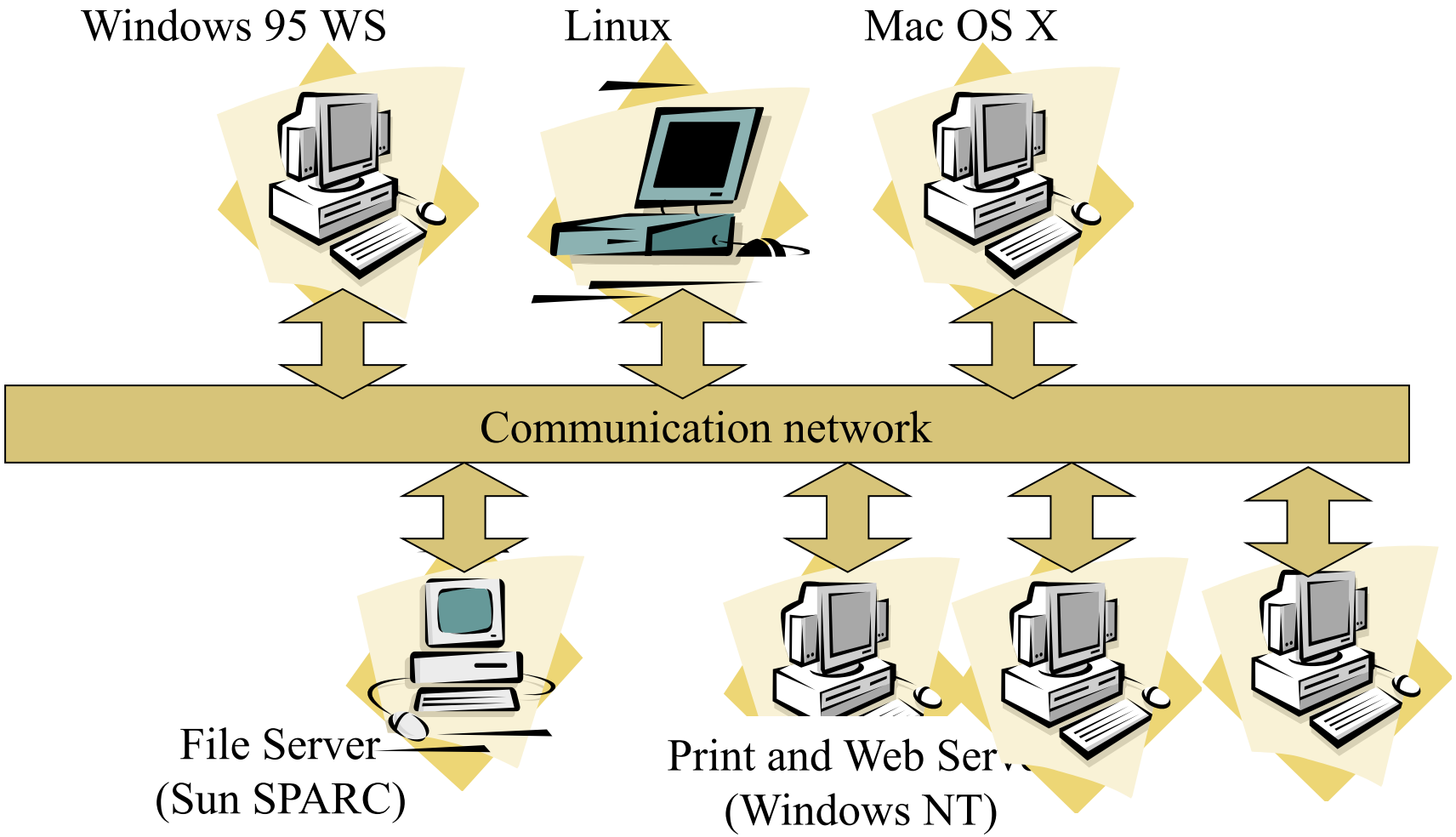
Client-Server

- A distributed system with --
  - Server(s): Stand-alone, powerful machines
  - Clients: Subsystem which employ the services simultaneously
  - Network: Communicating clients with servers
- Processing model:
  - Client initiates request
  - Request passes through network
  - Server responds
  - Response returned via network

Client–server communication

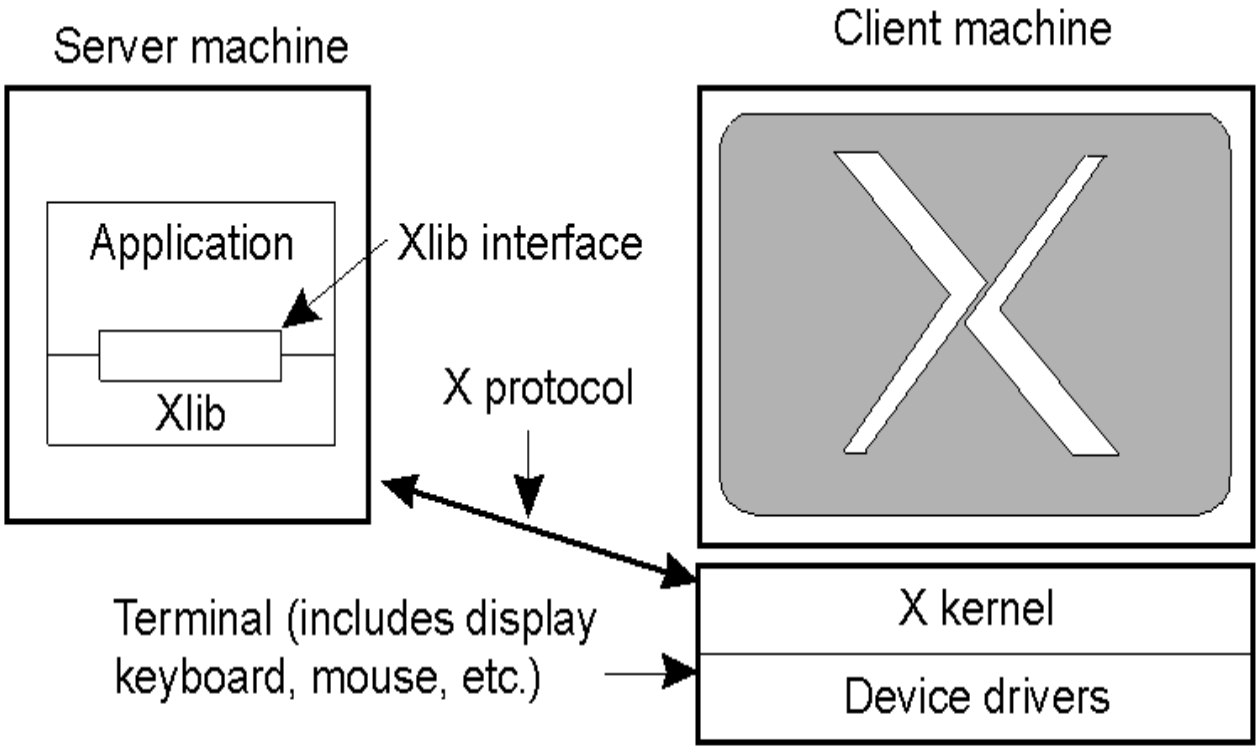
- Client–server communication requires the client to know the interface of the server subsystem, but the communication is only in one direction
- The client subsystem requests services from the server subsystem and not vice versa

Client Server: Example 1



Client-Server: Example 2

- X-Windows

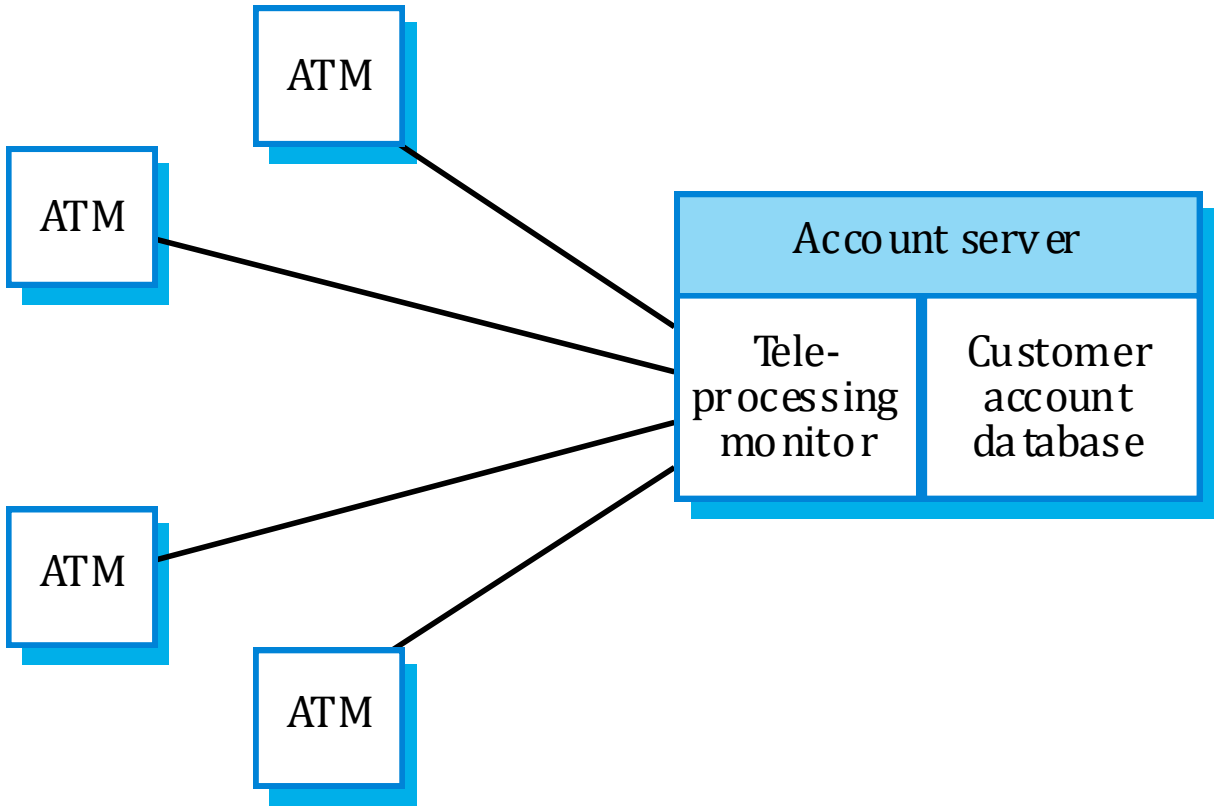


Client-Server: Example 3

- Email
  - Server: MS Exchange Server on exchange3.essex.ac.uk
    - The domain resolves to more than one machine
  - Clients: Outlook, Thunderbird, Outlook Express, Eudora, …
  - Protocols
    - Post Office Protocol Ver. 3 (POP3)
    - Internet Message Access Protocol (IMAP)
    - Exchange Server

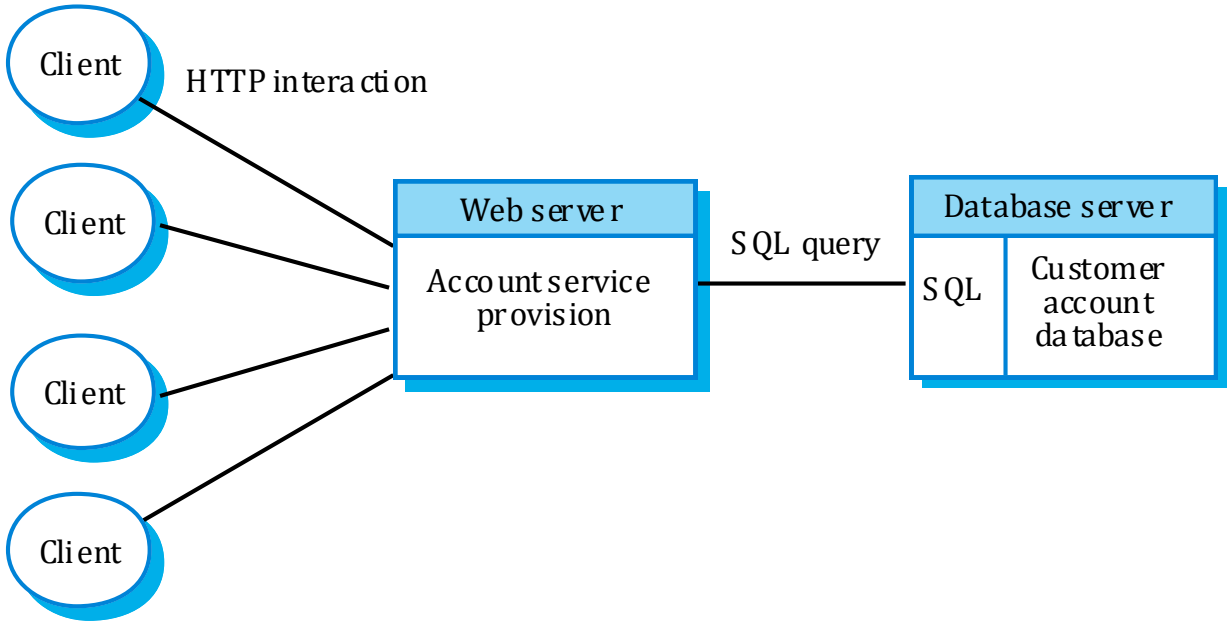
Client-Server: Example 4

- Automated teller machine



Client-Server: Example 5

- Three-tier client-server architecture
  - Customary in Web configurations such as Java Server Pages (JSP)



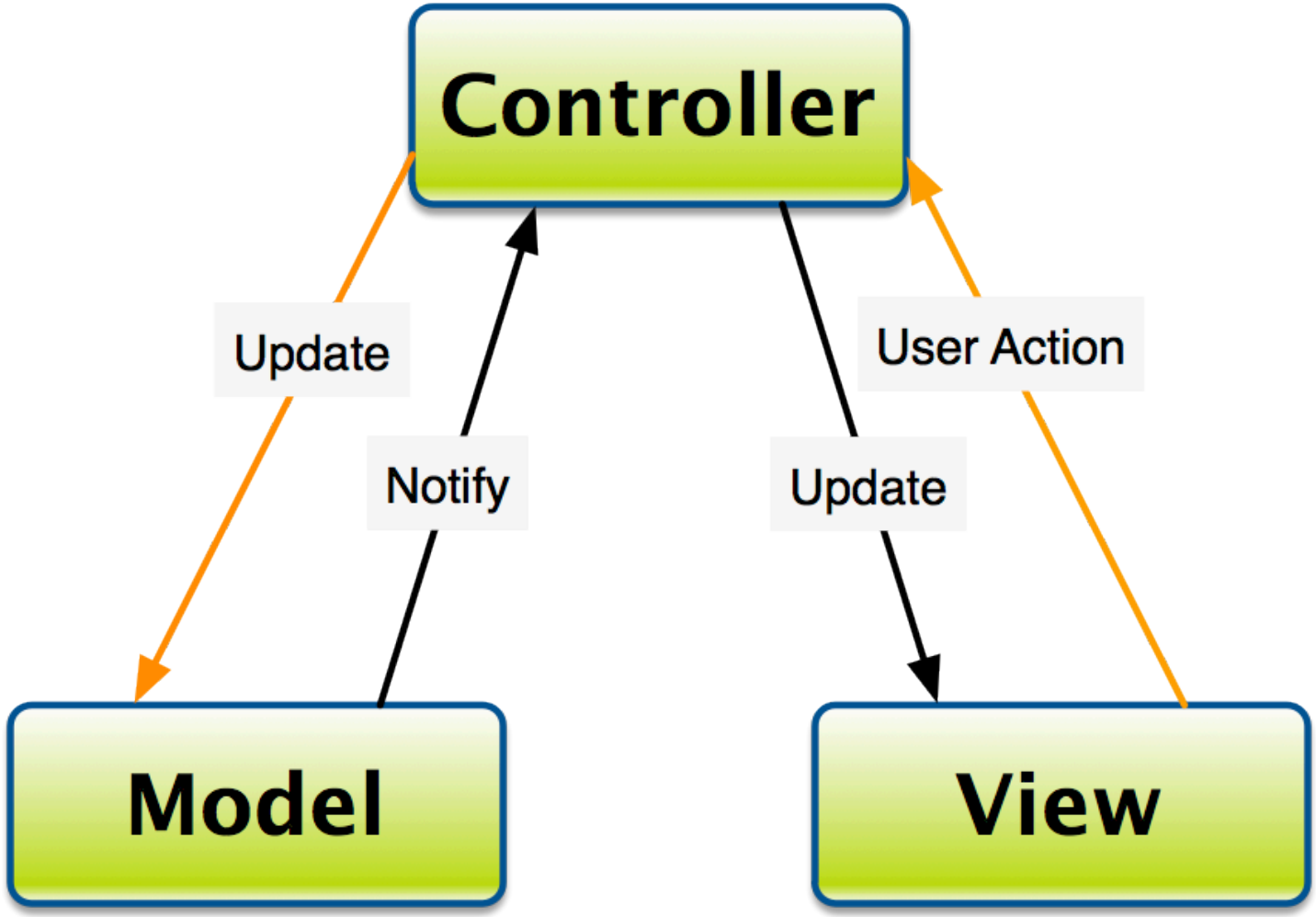
Client-Server: Properties

- Transparency
  - Possible distribution of server on different machines without clients' dependency
  - Clients know servers, not vice versa
- Heterogeneity
  - Allowing for different machines, operating systems
- Redundancy

Side note: Redundancy

- In engineering, redundancy is the duplication of critical components or functions of a system with the intention of increasing reliability of the system, usually in the case of a backup or fail-safe.
- There are four major forms of redundancy, these are:
  - Hardware redundancy, such as DMR and TMR (Dual and Triple Modular Redundant).
    - -- A machine which is Dual Modular Redundant has duplicated elements which work in parallel to provide one form of redundancy.
  - Information redundancy, such as error detection and correction methods
  - Time redundancy, including transient fault detection methods such as Alternate Logic
  - Software redundancy such as N-version programming

Architectural Style: Model-View-Controller



Model-View-Controller

- ***Model***—provides the central functionality of the application and is aware of each of its dependent view and controller components. Similar to UML Entity stereotype
- ***View***—corresponds to a particular style and format of presentation of information to the user. The view retrieves data from the model and updates its presentations when data has been changed in one of the other views. The view creates its associated controller. Similar to UML Boundary stereotype

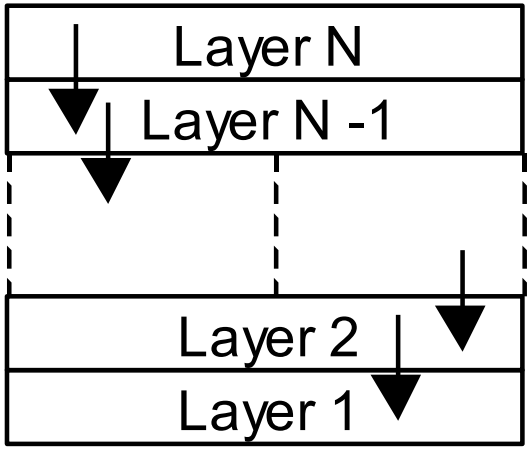
Model-View-Controller

- ***Controller***—accepts user input in the form of events that trigger the execution of operations within the model. These may cause changes to the information and in turn trigger updates in all the views ensuring that they are all up to date. Similar to UML Control stereotype
- Maps to UML stereotypes: Entity, Boundary, Control
- Maps to 3 tier client-server architecture

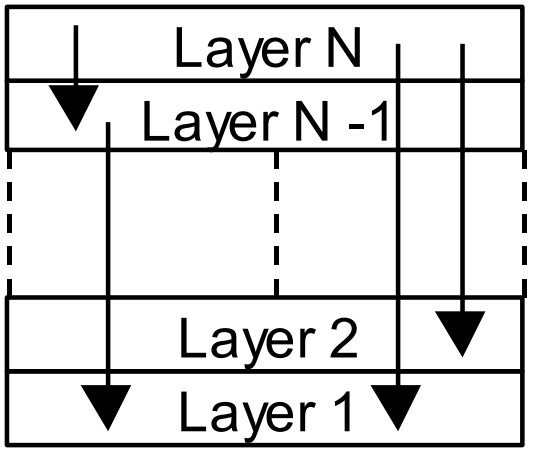
Layering and Partitioning

- Two general approaches to the division of a software system into subsystems
  - *Layering*—so called because the different subsystems usually represent different levels of abstraction
  - *Partitioning*, which usually means that each subsystem focuses on a different aspect of the functionality of the system as a whole
- Both approaches are often used together on one system

Schematic of a Layered Architecture



Closed architecture—messages may only be sent to the adjacent lower layer.



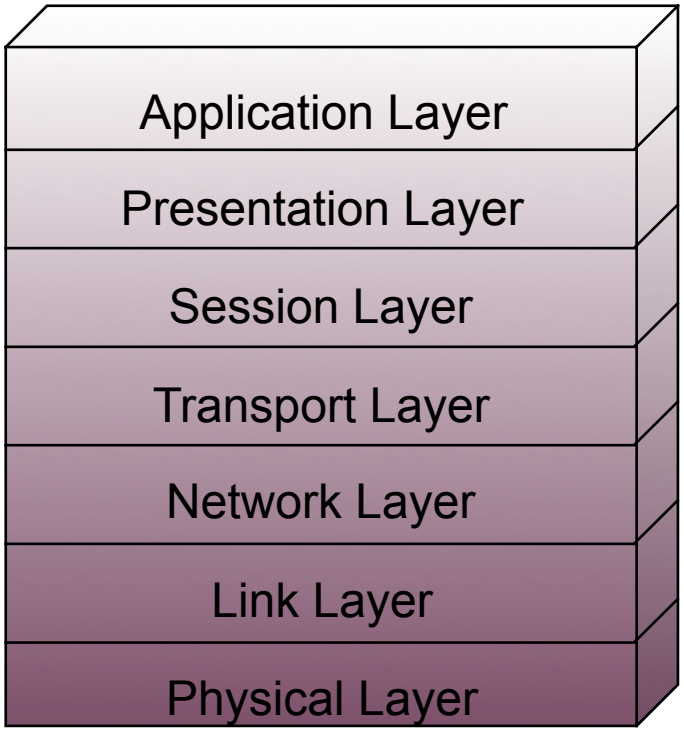
Open architecture—messages can be sent to any lower layer.

Layered Architecture

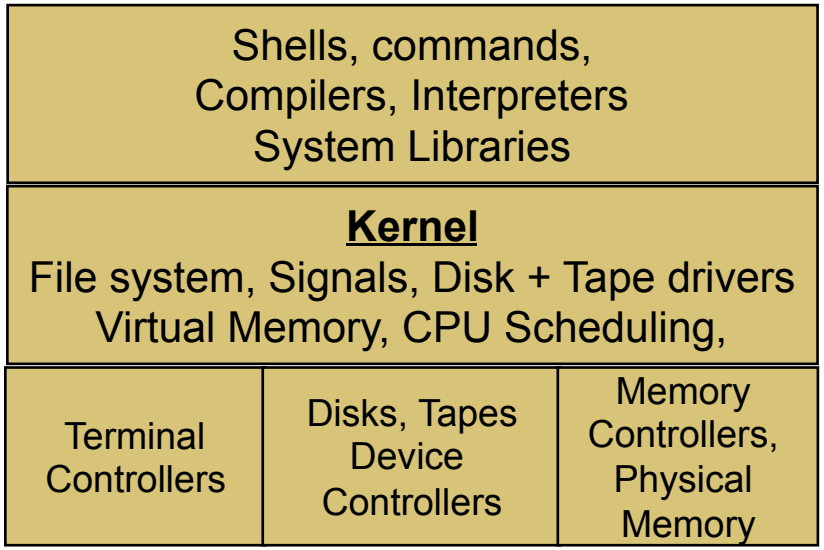
- A closed architecture **minimizes dependencies** between the layers and reduces the impact of a change to the interface of any one layer
- An open layered architecture produces more compact code, the services of all lower level layers can be accessed directly by any layer above them without the need for extra program code to pass messages through each intervening layer, but this breaks the encapsulation of the layers

Layered Architecture: examples

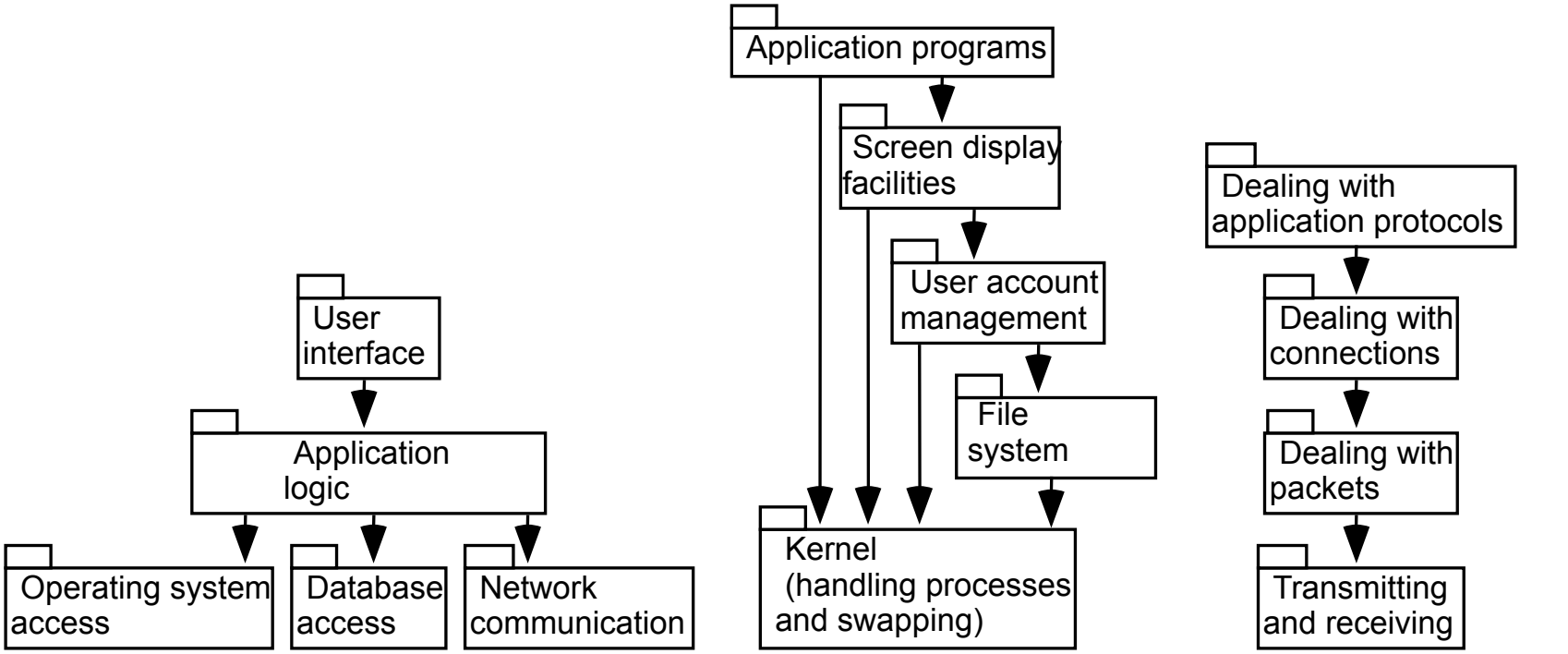
ISO Communication Protocols



UNIX Operating System



Layered Architecture: examples II

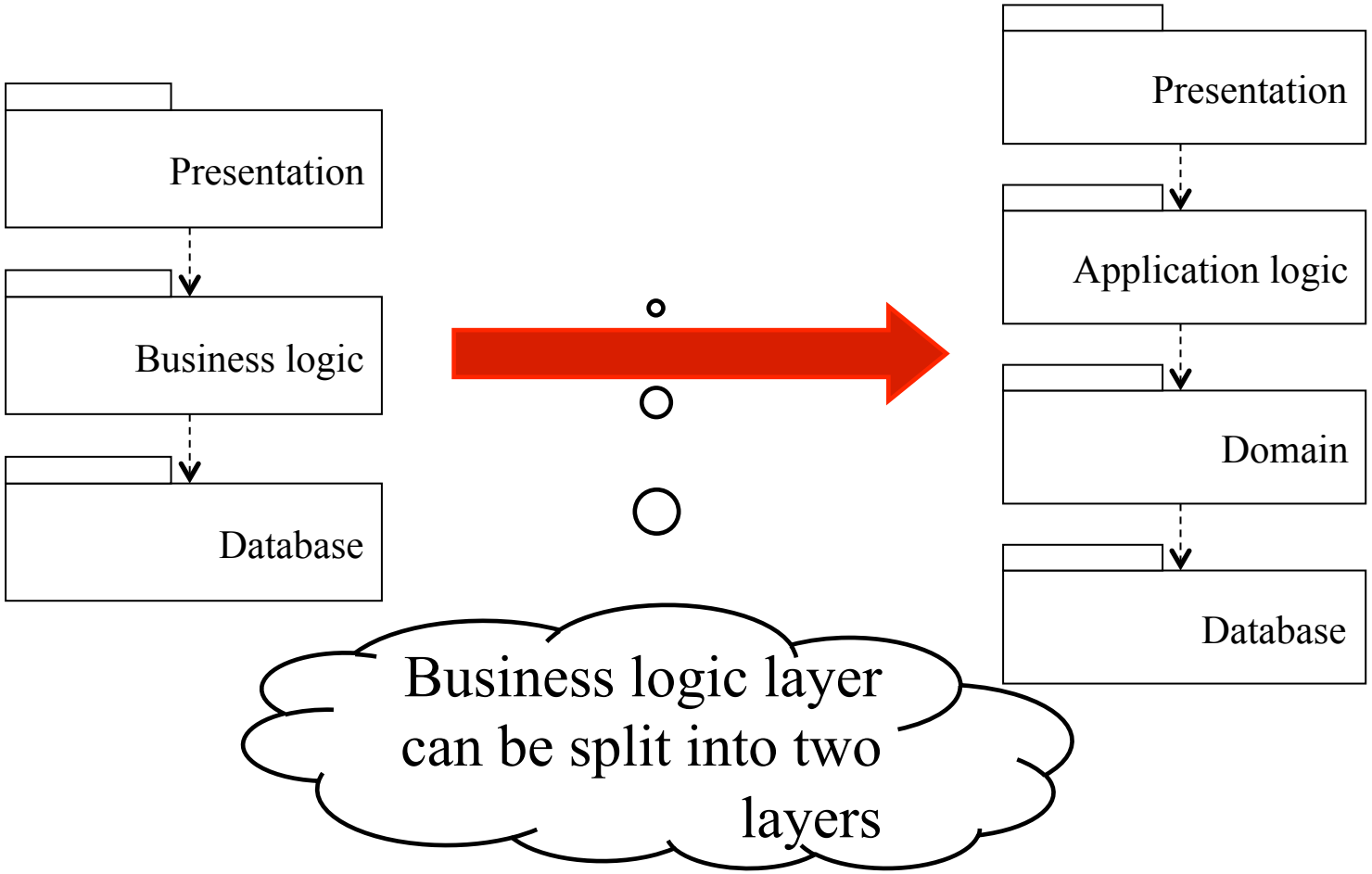


a) Typical layers in an application program

b) Typical layers in an operating system

c) Simplified view of layers in a communication system

Three & Four Layer Architectures.

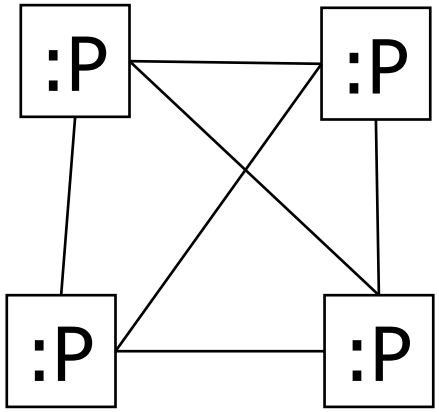


Peer-to-peer (P2P) communication

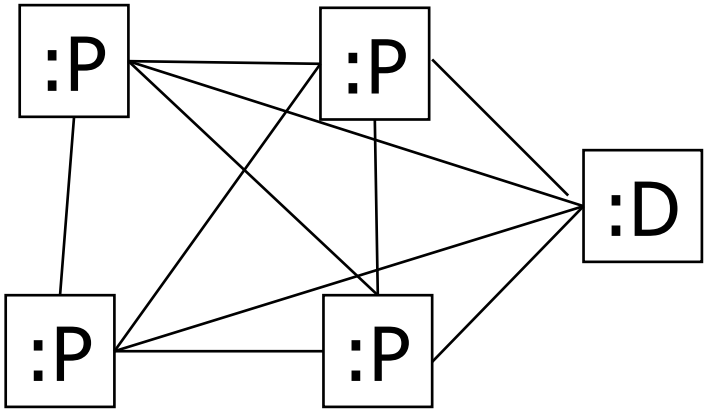
- Peer-to-peer communication requires each subsystem to know the interface of the other, thus coupling them more tightly
- The communication is two way since either peer subsystem may request services from the other

Peer-to-Peer (P2P)

- System composed of any number of peers
- Peer: client & server
  - Computations may be performed by any peer
- Peers popping in and out of existence



Pure P2P



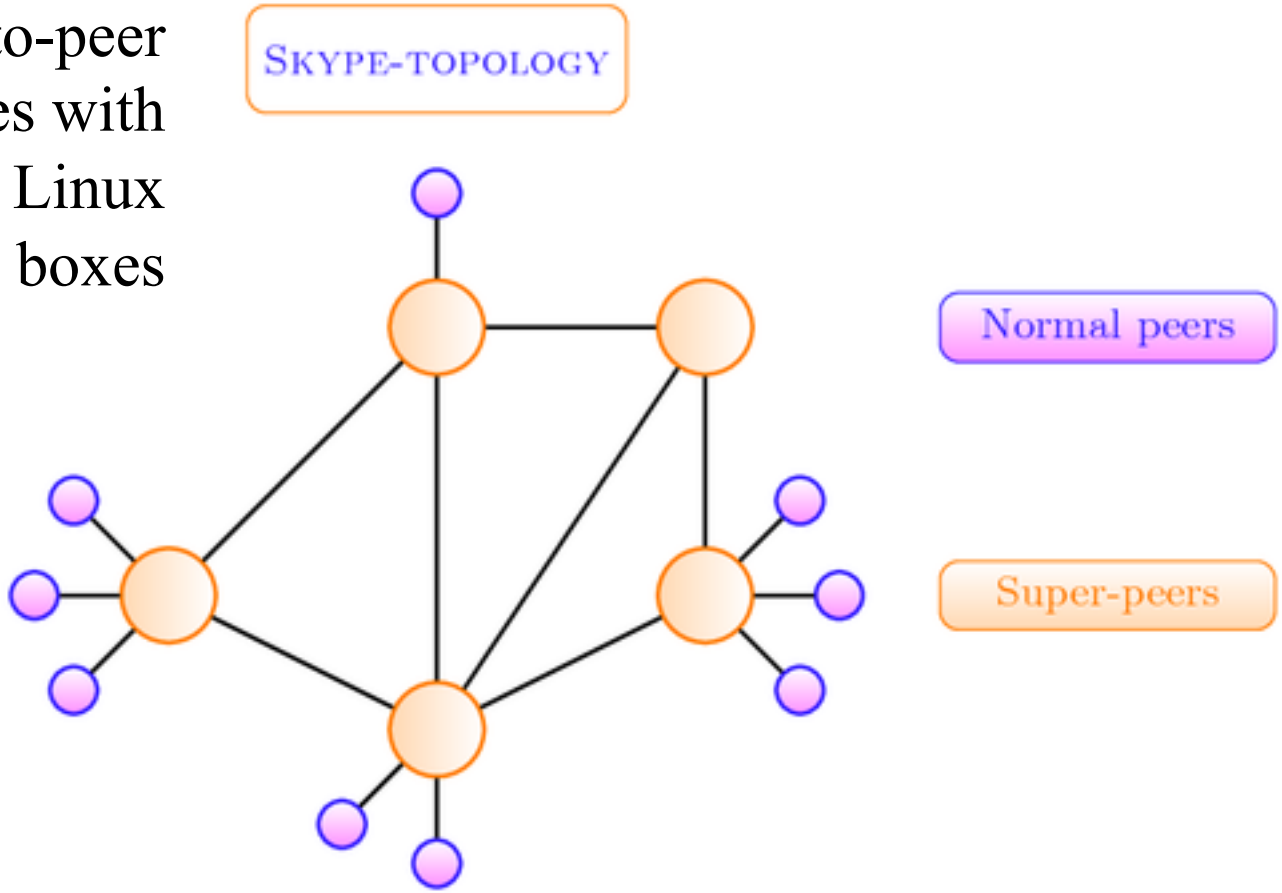
P2P with a discovery server

P2P: Examples I

- File sharing
  - Napster, Gnuttela, Kazaa, torrent applications…
- Messenger
  - Skype, ICQ, Yahoo Messenger, MSN Messenger …
  - NB. Skype is a hybrid peer to peer ie. has server
- JXTA
  - XML encoding of messages
  - Resources found via "advertisement"
    - Requires a "discovery" server

P2P example: Skype

Replacing peer-to-peer client machines with thousands of secure Linux boxes

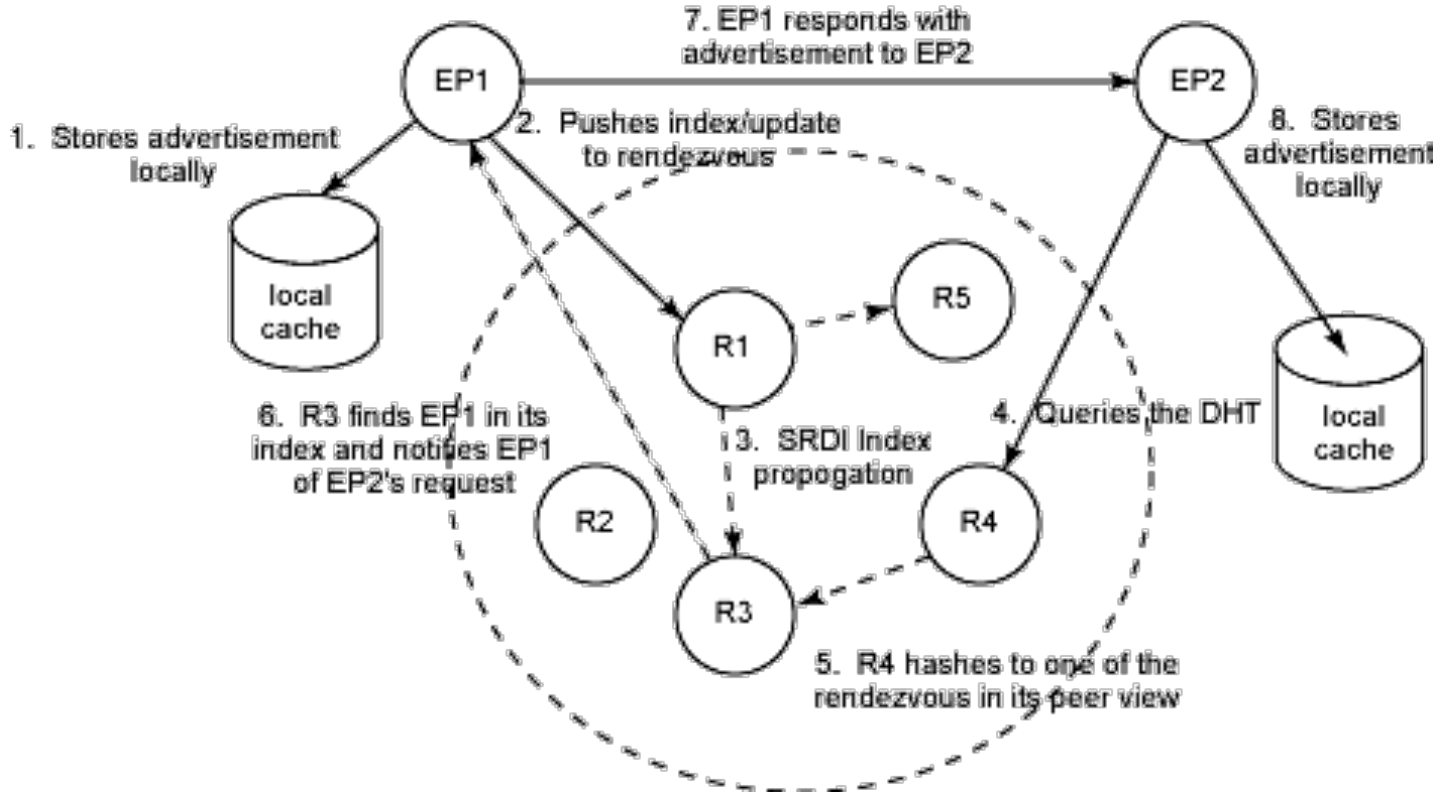


A note on Skype

- Since its introduction in 2003, the network has consisted of "supernodes" made up of regular users who had sufficient bandwidth, processing power, and other system requirements to qualify. These supernodes then transferred data with other supernodes in a peer-to-peer fashion. At any given time, there were typically a little more than 48,000 clients that operated this way.
- Skype is now being powered by more than 10,000 supernodes that are all hosted by the company.
- It's currently not possible for regular users to be promoted to supernode status. What's more, the boxes are running a version of Linux using grsecurity, a collection of patches and configurations designed to make servers more resistant to attacks.
- The banishment of user-supplied supernodes comes as the number of people simultaneously signed into Skype has mushroomed.
- According to Skype that number hit 41 million in April 2012, a 37-percent jump from the average number of concurrent users when Microsoft acquired Skype in 2011.

P2P: Advertisement and Discovery

- Advertisement & discovery in JXTA2



Steps in previous diagram

- 1. EP1 creates a pipe and stores its advertisement locally.
- 2. The index is updated locally and the changes are pushed to the connected rendezvous (R1).
- 3. Through a shared resource distributed (SRDI) index propagation, the rendezvous receiving the update (R1) replicates the new index information to the selected rendezvous (R3 and R5) in the super-peer network
- 4. Later, EP2 queries for EP1's pipe. This query is sent to its only connected rendezvous -- R4.
- 5. R4 hashes the query's attributes and redirects the request to another rendezvous in the super-peer network -- R3.
- 6. R3, having received EP1's index update through R1, immediately notifies EP1 of EP2's request.
- 7. EP1 sends a response directly to EP2 containing the requested pipe advertisement. At this point, the query is resolved.
- 8. EP2 may in turn decided to store the pipe advertisement, and the cycle begins again.

A note on JXTA

- The JXTA protocols defines a suite of six XML-based protocols that standardize the manner in which peers self-organize into peergroups, publish and discover peer resources, communicate, and monitor each other.
- The Endpoint Routing Protocol (ERP) is the protocol by which a peer can discover a route (sequence of hops) to send a message to another peer potentially traversing firewalls and NATs.
- The Rendezvous Protocol (RVP) is used for propagating a message within a peergroup.
- The Peer Resolver Protocol (PRP) is the protocol used to send a generic query to one or more peers, and receive a response (or multiple responses) to the query.
- The Peer Discovery Protocol (PDP) is used to publish and discover resource advertisements.
- The Peer Information Protocol (PIP) is the protocol by a which a peer may obtain status information about another peers.
- The Pipe Binding Protocol (PBP) is the protocol by which a peer can establish a virtual communication channel or pipe between one or more peers.
- The JXTA protocols permit the establishment a virtual network overlay on top of physical networks allowing peers to directly interact and organize independently of their network location and connectivity. The JXTA protocols have been designed to be easily implemented on unidirectional links and asymmetric transports.
- To create a Distributed Hash Table (DHT), each peer caches advertisements locally, and all locally stored advertisements are indexed.
- The index is pushed to the rendezvous node (JXTA 2 edge peer connects to only one rendezvous node at any time).
- The rendezvous super-peer network maintains the DHT containing the amalgamated indices. Queries are always sent to a rendezvous, as illustrated in the previous diagram

P2P: Discovery models

- Q. How peers are found?
- A. Discovery models
  - Some peers offer a directory service
    - "super peers"
    - rendezvous peers
    - Relay peers
  - Multicast – peers announce their presence on some communication channel
  - Neighbour model – via other peers

P2P: Discussion

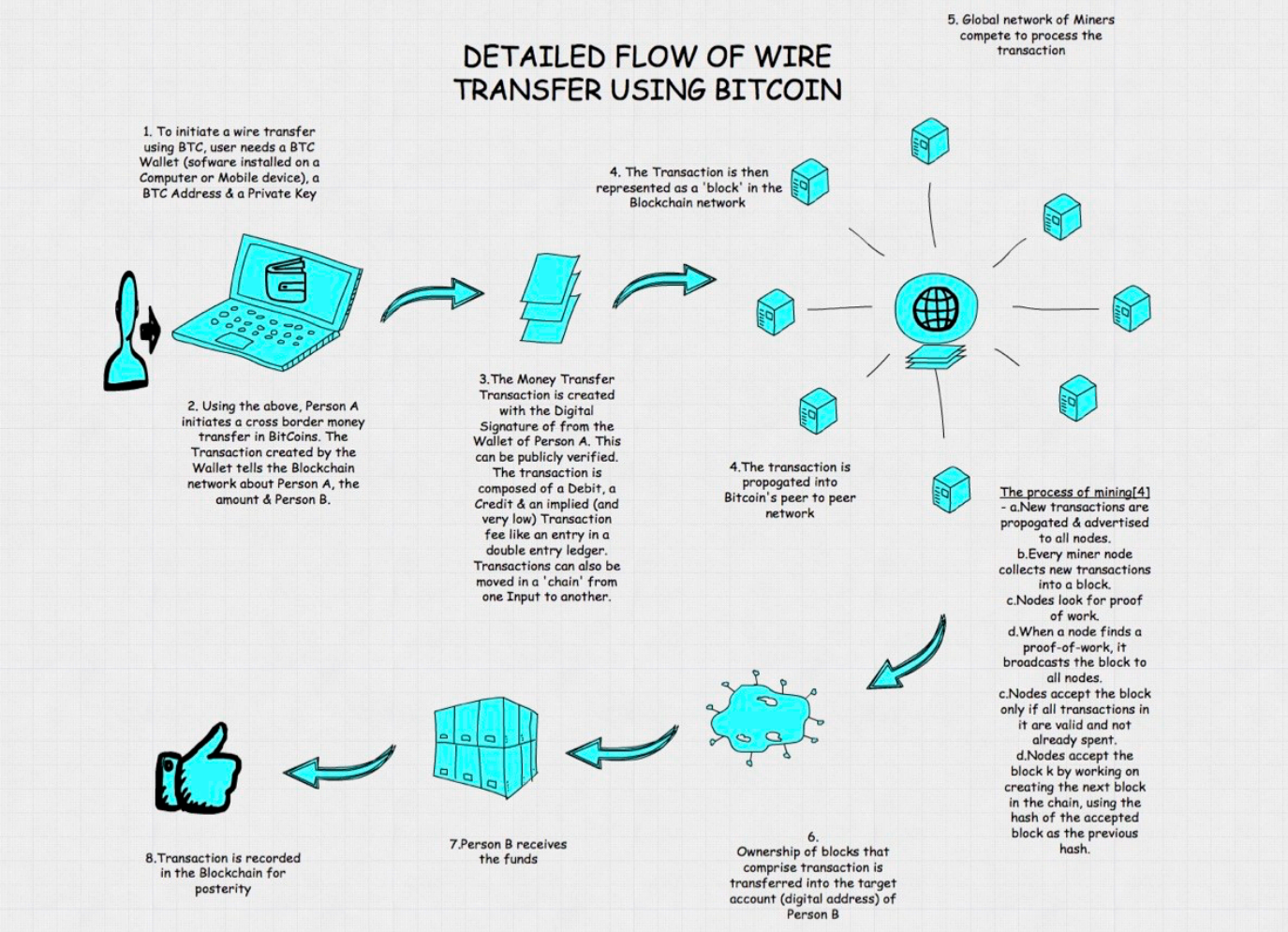
- Pros:
  - Very flexible
  - Evolution is with minimal effort
  - Efficient – resource utilization is close to optimal
  - Heterogeneous
- Cons:
  - Security: authentication is close to impossible (however, see next slide)
  - Duplication
  - Data corruption

Final P2P example: Blockchain

- Blockchain: *a distributed cryptographic ledger shared amongst all nodes participating in the network, over which every successfully performed transaction is recorded*

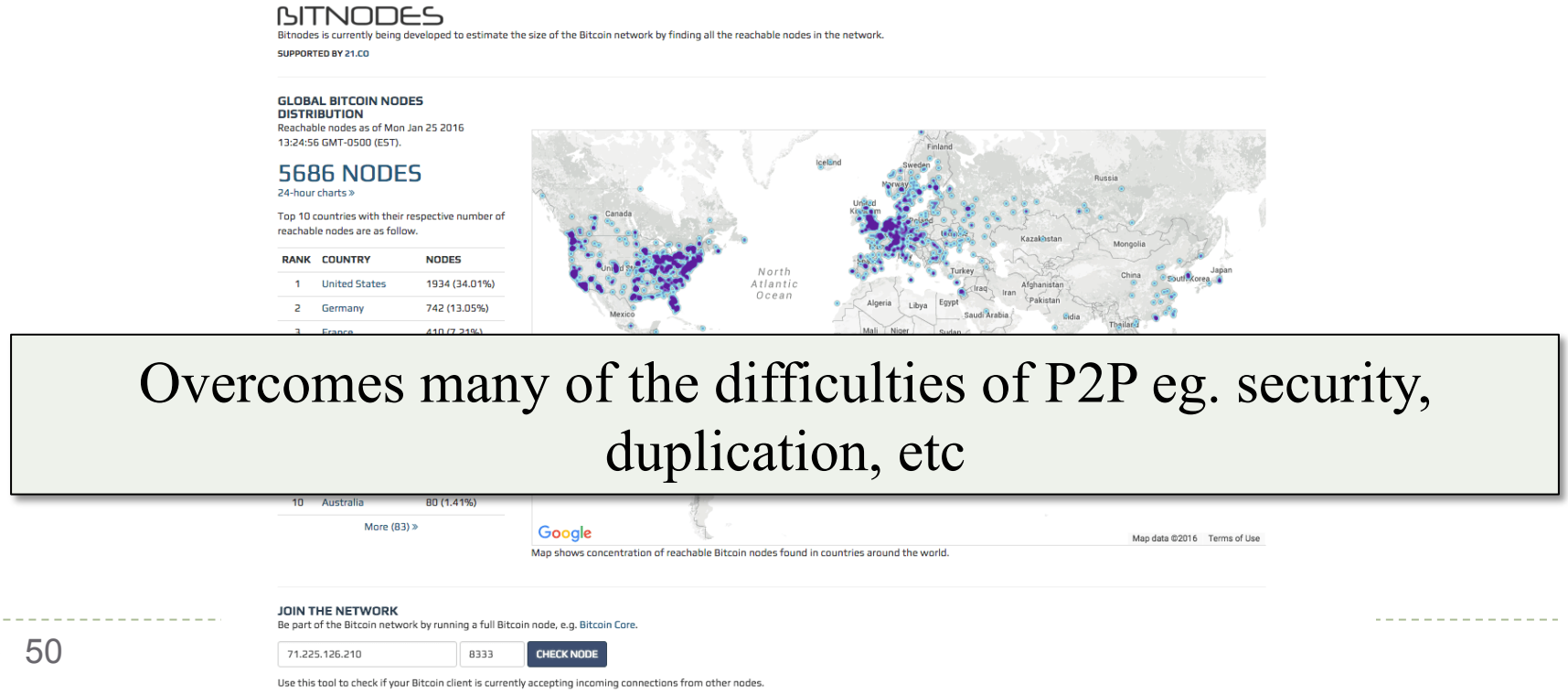
Example:
Bitcoin
A true P2P
distributed
economy

P2P with strong inbuilt crypotography



Blockchain P2P nodes

- The nodes in the BitCoin blockchain play the role of a **Central Bank** or a trusted third party. Every node maintains a **fully replicated copy** of a database that contains the payment history of every bitcoin ever created along with ownership information. As transactions happen using the currency, **a consensus mechanism** essentially dictates how nodes agree on blockchain updates.



Overcomes many of the difficulties of P2P eg. security, duplication, etc

Summary

- How to model sub-systems using nested diagrams in UML and Package diagrams
- Modeling the system architecture based on the 4+1 architectural view
- UML Architectural Description Language (ADL)
- The system architecture is defined during the system design process
- Some example architectural styles
  - Shared repository
  - Client/Server
  - Model/View/Controller
  - Layered architectures
  - P2p

Further reading

- Bennett, 8.3.4 Packages and dependencies
- Garlan & Shaw (1993). "**An Introduction to Software Architecture". In** "An Introduction to Software Architecture," *Advances in Software Engineering and Knowledge Engineering, Volume I,* edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993.
- Ian **Sommerville 2004** Software Engineering, 7th edition.
- Chapter 9: Architecting and designing software, Lethbridge/Laganière 2001
- Bennett, Chapter 13, System Design and Architecture
- JXTA - http://www.ibm.com/developerworks/java/library/j-jxta2/
- Kruchten, Philippe (1995, November). Architectural Blueprints — The "4+1" View Model of Software Architecture. IEEE Software 12 (6), pp. 42-50

Exercises

- Observe the differences between apparently-similar styles: What are the differences between –
  - Client-Server vs. Broker?
  - Rendezvous peer vs. broker?
  - Peer-to-Peer vs. Grid?