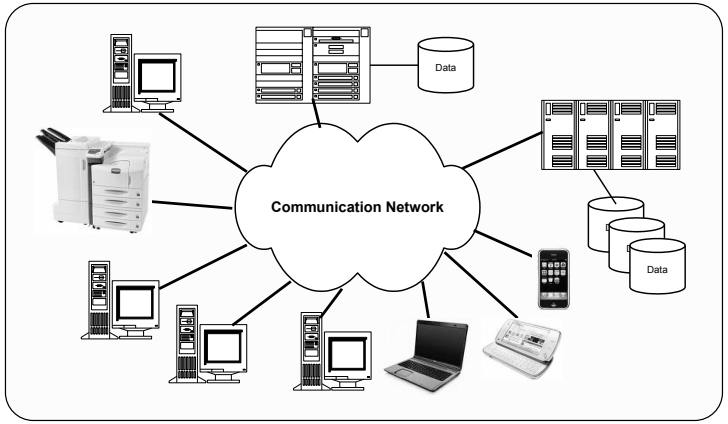
# *Distributed Systems*

Morris Sloman
Room 572

## Distributed Systems






## Why Distributed Systems?

Distributed computing is the most general means for the provision of computer processing. It offers users the advantages of interaction, cooperation and the sharing of facilities. Reduced incremental costs, improved availability, extensibility, and better response and performance are also potential system benefits.

***What is a distributed system?***

***How does it provide these advantages?***

***How can one construct or use such a system?***

***What are the design issues?***



## Course Structure

1. Overview of Distributed System Architecture
2. Distributed Components and their Interaction
3. RPC and Remote Invocation Implementation
4. Time Synchronisation
5. Distributed Systems Management
6. Ubiquitous Computing
7. Publish-subscribe Messaging
8. Peer to Peer Systems
9. Sensor Networks

# References

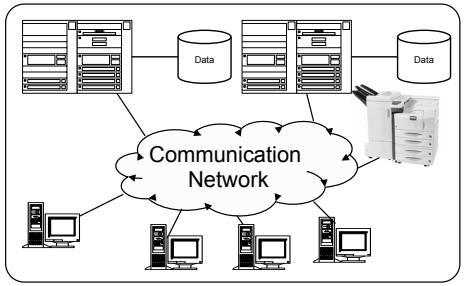
- Distributed Systems: Concepts and Design G.Coulouris, J.Dollimore, T.Kindberg, G. Blair Pearson 2011 (5th ed).
- Distributed Systems: Principles and Paradigms 2nd edition, A.S. Tanenbaum, M. Steen, Pearson, 2006. (A)
- Java in Distributed Systems, M. Boger, Wiley 2001
- Network and Distributed Systems Management. ed. Morris Sloman Addison Wesley 1994



# *Architecture*

- What are distributed systems? ***Definition***
- Characteristics and benefits
- Where are they used? ***Applications***
- Basic software structure –"layers"
- Client Server Architectures
- What are the main design issues?

# Definition




- A distributed system consists of a collection of autonomous computers interconnected by a computer network and equipped with distributed system software to form an integrated computing facility.

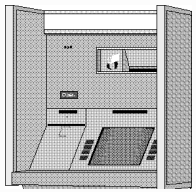
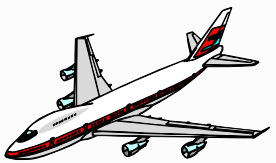
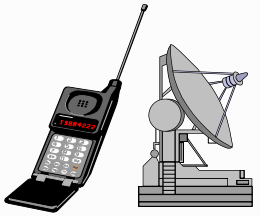
Processes and Databases interact in order to cooperate to achieve a common goal.

Processes co-ordinate their activities and exchange information by means of **messages** transferred over a communication network.



# Dependence on Distributed Computing

Distributed computer systems are critical for functioning of many organisations:

**Banks** **Transport** **Telecommunications**

# Characteristics/Advantages

- Resource sharing ➔ remote access to shared facilities
- Fault tolerance ➔ replication can remove single failure points
- Concurrency ➔ reduce response time by local processing
  ➔ improve throughput by parallelism
- Openness ➔ vendor independence via clearly defined interfaces and use of standards
- Scalability via multiple processors and multiple networks
- Incremental extensibility
- Modularity ➔ simpler design, installation & maintenance
- Flexibility ➔ incremental change of function & adaptation to new requirements
- Reflect application distribution
- But **no global time** ➔ difficult to support causality and consistency



# Financial Trading: Requirements

**Required Functionality**

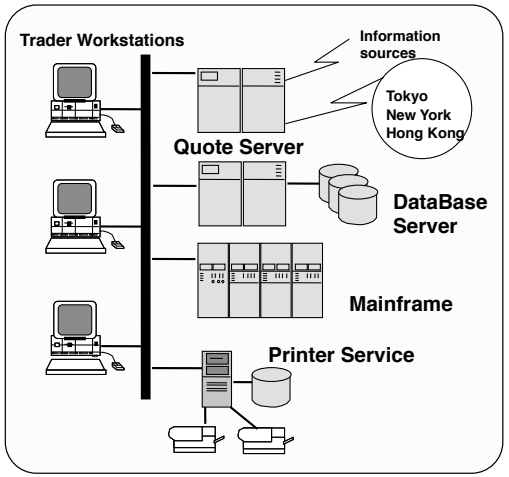
Selective viewing of market data
Fast display management
Fast processing capabilities
Networking for intercommunication
Link between accounting & financial dealing
Risk management & hedging strategies
Use market data directly in analysis packages
Automatic record and bookkeeping

**Required Properties**

integrity ➔ don't lose data
reliability ➔ don't go down
speed ➔ old news is not news
extensibility/scalability ➔ system matches the business



# Distributed Design



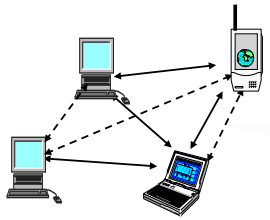

- **Integrated digital and video**
- **Integrated data and news**
- **Links to positions, clearing and accounting**
- **Paperless trading**
- **Powerful workstations...**
  - **Colour charts, graphics,**
  - **Realtime analysis**
  - **Expert systems**

**Architectural Approach**
**Data Broadcast and filtering – tagged messages**
**"Clients" register interest in particular kinds of data.**
**Receive relevant data when broadcast, and filter out other data.**



# Peer-to-peer (P2P) Resource Sharing

- Very large scale – potentially millions of users
- Share processing eg Seti@home, United Devices, Avaki, Akamai
- 'Share' music files eg Gnutella, Kazaa
- Collaboration e.g. Groove
- Main problem is locating resources without centralised directory

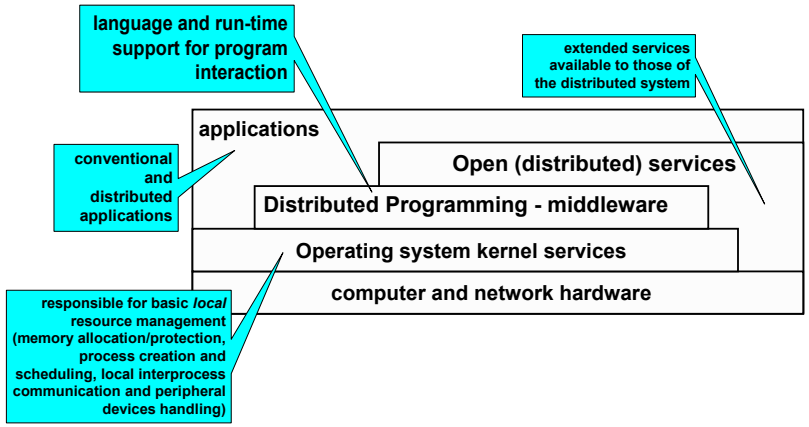


Where is the directory? Hard to find information

Publish and query to directory. ⟷

Get data from peers ⇠⇢

# Basic software structure






# Basic software structure

- **Open services**:
  - support the introduction of new services
  - provide access to distributed services, including the coordination required for remote resource use (sharing, protection, synchronisation, recovery....)
  - *e.g. Jini resource discovery,*
- **Distributed programming support**:
  - supports interaction (such as remote procedure call) for conventional languages and support for special purpose languages.
  - e.g. Java RMI, RPC



# Open Distributed Processing - RM

**Viewpoint** = abstract representation of a system
NOT phases in lifecycle model

**Enterprise Viewpoint**

- Overall goals, policies & organisational structure
- Roles & activities within organisation(s)
- Policies & constraints regarding inter-organisation interactions
- Community: configuration of objects established to meet an objective – specifies roles, relationships and policies

**Information Viewpoint**

- Modelling of information structures, information flows and knowledge representation
- Includes constraints on data
- No distinction between manual & automated information processing



# ODP Viewpoints

**Computational Viewpoint**

- Programming functions – IPC, object interfaces
- Application program structuring – independent of computer system on which it will run
- No distinction between processing & storage objects
- Includes configuration – object instantiation and bindings.

**Engineering Viewpoint**

- OS, communication system, database – implementation issues
- Provision of transparency mechanisms – fault tolerance, persistence etc.
- Processors & networks are visible

**Technology Viewpoint**

- Realised components from which distributed systems are built.
- Particular OS (Unix, Windows), protocols (FTP, TCP/IP), processors (Intel, sparc, ARM)

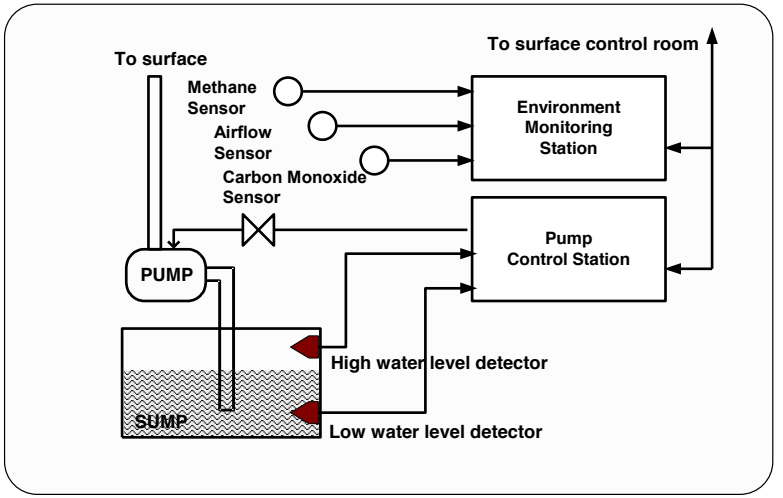
# Example: Pump for Mine Drainage

The pump is situated underground in a coal mine, and so for safety reasons it must not be started or continue running when the percentage of methane in the atmosphere exceeds a set safety limit. The pump controller obtains information on methane levels by communicating with a nearby environment monitoring station. As well as methane, this station also monitors carbon monoxide and airflow velocity. The environment monitoring station provides information to the surface and other plant controllers as well as to the pump controller.

Once *start* has been enabled by a command from the surface, the pump runs automatically controlled by the water level as sensed by the high and low level detectors. Detection of high level causes the pump to run until low level is reached. The surface may deactivate the pump with a *stop* command, and also *query* the status of the pump.

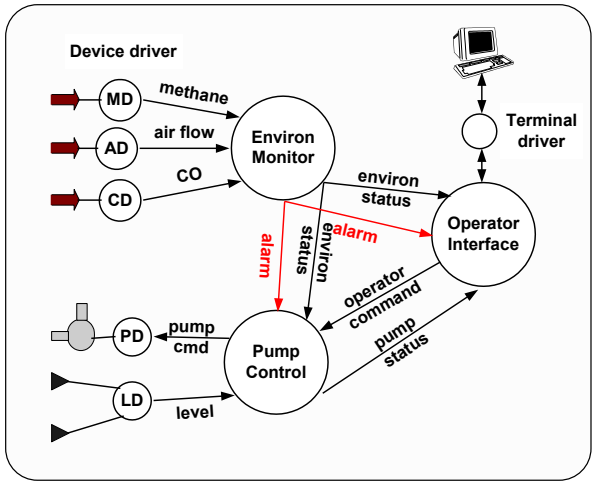


# Pump System Overview






# Pump Control Schematic




**Data Flow Diagrams**

Component Processes describe the functions of the system

Data flows = data type + direction



# Data Dictionary for Pump System

pump cmd = (on, off)
level = (high, low)
methane = real
airflow = real
alarm = signal

environ status = methane + airflow + CO

operator cmd = (start, stop, status)

pump status = (stopped, lowstop, methanestop, running)

# Distributed System Design Approach

Enterprise View
- **Specify requirements and identify interactions with the environment**
- **Identify main processing components – processes or threads of control**
- **Assume 1 process per device**

Information View
- **Identify data flows – direction and data types (dictionary)**
- **Ignore how interactions are initiated or types of interaction primitives**

Computation View
- **Decide on interaction primitives**
- **Decide on control flow i.e. whether data is pushed or pulled**
- **e.g. whether controllers are polled or event driven**
- **Specify component interfaces**
  **= interactions + signatures (parameter types)**
- **Specify component functions in terms of outline code and data structures**

Engineering View
- **Optimise and allocate to physical nodes**



# Design Issues

The following design issues will be addressed in this course:

- **Communication**: process interaction and synchronisation paradigms
- Distributed system **service provision**
- **Clock synchronisation** for global time and causality
- **Management** and specifying policy for distributed and ubiquitous systems
- **Publish-subscribe** systems
- Exemplar Distributed Applications – Sensor Networks and Peer to Peer



# Architecture Summary

- What are distributed systems – definition
- Why are they of interest – potential benefits
- Where are they used – applications

- **Architecture**
  - Basic software structure “layers”
  - Viewpoint decomposition
- The main design issues