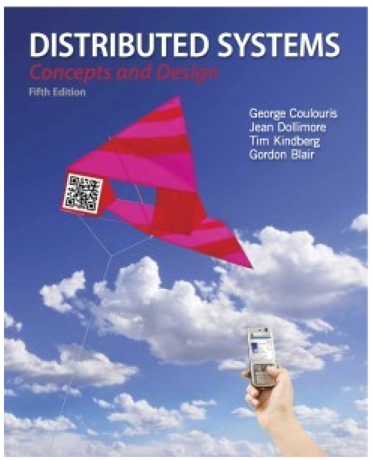
Week 1
An Introduction to Distributed Systems



Reference:

Chapter 1 & Chapter 2

Distributed Systems: Concepts and Design

Couolouris, Dollimore, Kindberg and Blair

Edition 5, © Addison Wesley 2011

Learning Objectives

- Describe distributed systems in terms of:
  - Definition of distributed systems
  - Motivation of constructing distributed systems
  - Characteristics of distributed systems
  - Examples of popular distributed systems
- Recognise challenges of constructing distributed systems in terms of:
  - Heterogeneity, Openness, Security, Scalability, Failure Handling, Concurrency and Transparency.

Learning Objectives

- Appraise architectural models of distributed systems in terms of:
  - Definition of architectural models
  - Client/Server model and its variations
  - Peer-to-Peer model
- Interpret fundamental models of distributed systems in terms of:
  - Interaction Models
  - Failure Models
  - Security Models

Definition of Distributed Systems

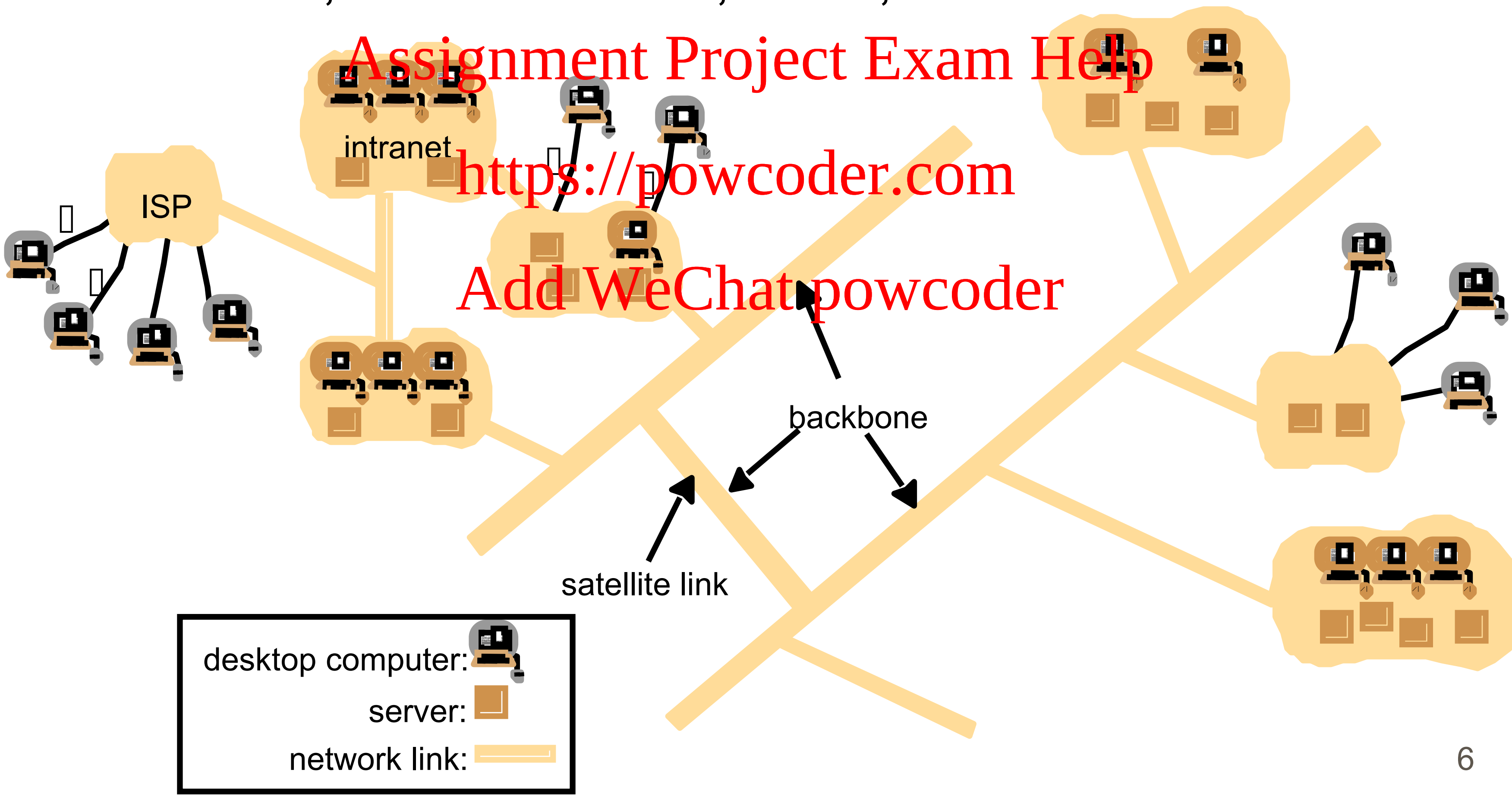
- A distributed system is defined as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.
- This definition leads to the following characteristics of distributed systems.
  - Concurrency
  - No global clock
  - Independent failure

Motivation of Constructing Distributed Systems

- To motivate the benefits of resource sharing
  - Hardware: disks, printers
  - Software: programs
  - Data: files, database records
- Patterns of resource sharing vary widely
  - Machines that access a search engine would never contact with one another directly.
  - In CSCW (Computer-Supported Cooperative Working), machines would cooperate directly.

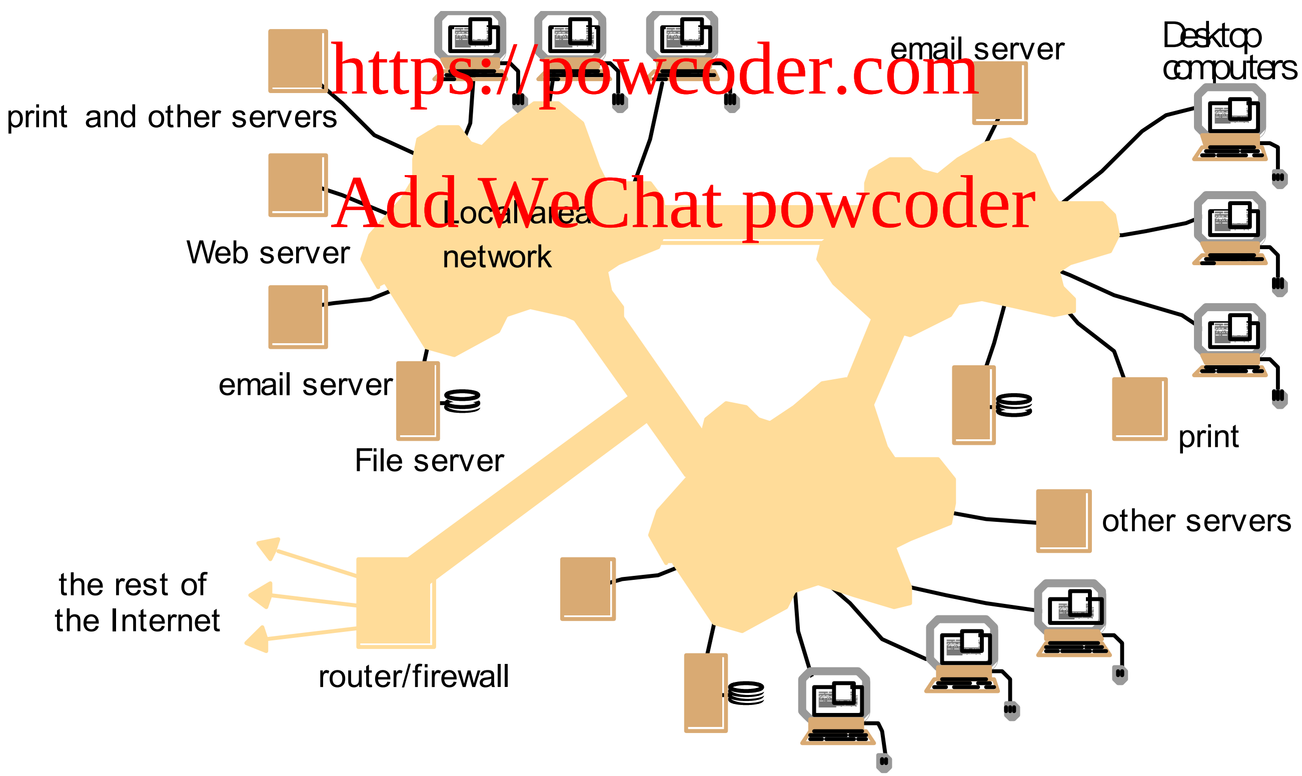
Examples of Distributed Systems

- The Internet is a very large distributed system that allows users throughout the world to make use of its services, such as WWW, email, and file transfer etc.



Examples of Distributed Systems

- An intranet is a part of the Internet that is separately administered and uses a firewall to enforce its own local security policies. Users in an intranet share data by means of file services.

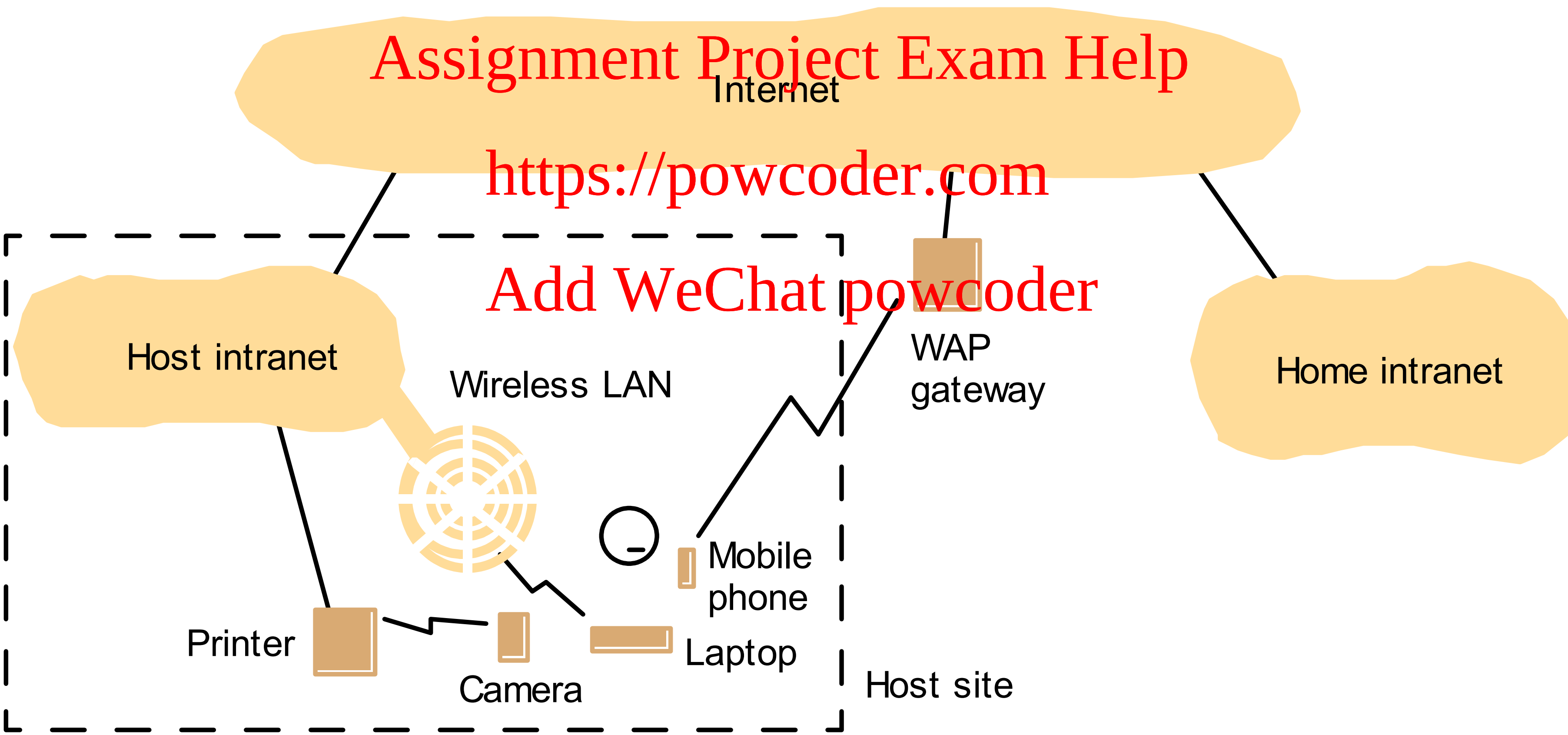


Examples of Distributed Systems

- Mobile computing
  - Laptops, PDAs and wearable computing devices
  - Mobility; Wireless
  - Host intranet, home intranet, and the Internet
  - Location-aware or context-aware computing
- Ubiquitous computing
  - Small computers embedded in appliances
  - Computing and communication behaviour will be transparently and intimately tied up with their physical function.

Examples of Distributed Systems

- Portable and handheld devices in a distributed system



Resource Sharing and Services on the WWW

- The WWW is an evolving system for publishing and accessing resources and services across the Internet.

- The WWW provides a hypertext structure among the documents, which contains links to other documents and resources.

- The WWW is based on three components:
  - Hypertext Markup Language (HTML)
  - Uniform Resource Locators (URLs)
  - Client/Server architecture based on Hypertext Transfer Protocol (HTTP)

Resource Sharing and Services on the WWW

- URL1: http://www.cqu.edu.au/
  - http://www.cqu.edu.au/index.html
  - A web page (homepage) on CQU web server
  - Access resource
- URL 2: http://www.google.com.au/search?hl=en&q=Distributed+Systems&btnG=Google+Search&meta=
  - Request a service: search
  - Keywords are provided as a query string to the server.

Challenges of Constructing Distributed Systems

- Heterogeneity
  - Services are running on heterogeneous environment.
  - Networks, computer hardware, operating systems, programming languages etc.
  - The differences must be dealt with if messages are to be exchanged.
  - Masking differences are addressed by using common standards and protocols, middleware or virtual machines.

Challenges of Constructing Distributed Systems

- Heterogeneity
  - TCP/IP protocol suite are for routing messages over the Internet and reliable delivery of messages.
  - CORBA (Common Object Request Broker Architecture) enables an object to invoke another object even if they are constructed by different languages and running on different machines.
  - The cross-platform benefits of Java Virtual Machine: ***Write once, run anywhere***.

Challenges of Constructing Distributed Systems

- Openness
  - The degree to which new resource sharing services can be added and made available for use by a variety of client programs.
  - Open systems are characterised by the fact that their key interfaces are published.
  - Open systems are based on the provision of a uniform communication mechanism for access to shared resources.
  - Open systems can be constructed from heterogeneous hardware or software, but they conform to the published standards.

Challenges of Constructing Distributed Systems

- Security
  - Information resources in distribute systems have a high intrinsic value.
  - Security issues include:
    - A verity of attacks are from a variety of attackers
    - Goals are from stealing data to attacks like Denial of Service.
  - Security has three components:
    - Confidentiality
    - Integrity
    - Availability

Challenges of Constructing Distributed Systems

- Security
  - Security defences include:
    - Firewalls; Intrusion Detection Systems
    - Encryption; Digital Signature
    - Authentication
    - Access Control

Challenges of Constructing Distributed Systems

- Scalability
  - System remains operational and effective despite changes in numbers of resources and users.
  - Issues to design scalable distributed systems include:
    - Controlling the cost of physical resources
    - Controlling the performance loss
    - Preventing software resources running out
    - Avoiding performance bottlenecks

Challenges of Constructing Distributed Systems

- Failure Handling
  - When faults occur in hardware or software, program may produce incorrect results or stop.
  - Failures in distributed systems are partial.
  - Failure handling techniques include:
    - Detecting failures
    - Masking failures
    - Tolerating failures
    - Failure recovery
    - Redundancy

Challenges of Constructing Distributed Systems

- Concurrency
  - Services are shared by clients and allow multiple client requests to be processed concurrently.
  - Concurrency control is necessary to ensure data consistency.
    - Thread A: Get balance from account: Smith ($1,000)
    - Thread B: Get balance from account: Smith ($1,000)
    - Thread A: withdraw $100 from the account
    - Thread B: deposit $200 to the account
    - Thread A: update balance to account: Smith ($900)
    - Thread B: update balance to account: Smith ($1,200)
  - Operations must be synchronized in such a way that data consistency is ensured.

Challenges of Constructing Distributed Systems

- Transparency
  - The concealment from users and application programmers of the separation of components in a distributed system.
  - Access transparency
    - Local and remote resources can be accessed using identical operations.
  - Location transparency
    - Resources can be accessed without knowledge of their physical or network locations.
  - Concurrency transparency
    - Several processes can operate concurrently using shared resources without interference between them.

Challenges of Constructing Distributed Systems

- Transparency
  - Replication transparency
    - Multiple instances of resources can be used to increase reliability and performance without knowledge of the replicas by users or application programmers.
  - Failure transparency
    - Concealment of faults, allowing users and application programs complete their tasks despite the failure of hardware or software components.
  - Mobility transparency
    - Allows movement of resources and clients within a system without affecting the operations of users or programs.

Challenges of Constructing Distributed Systems

- Transparency
  - Performance transparency
    - Allows the system to be reconfigured to improve performance as loads vary.
  - Scaling transparency
    - Allows the system and applications to expand in scale without change to the system structure or the application algorithms.

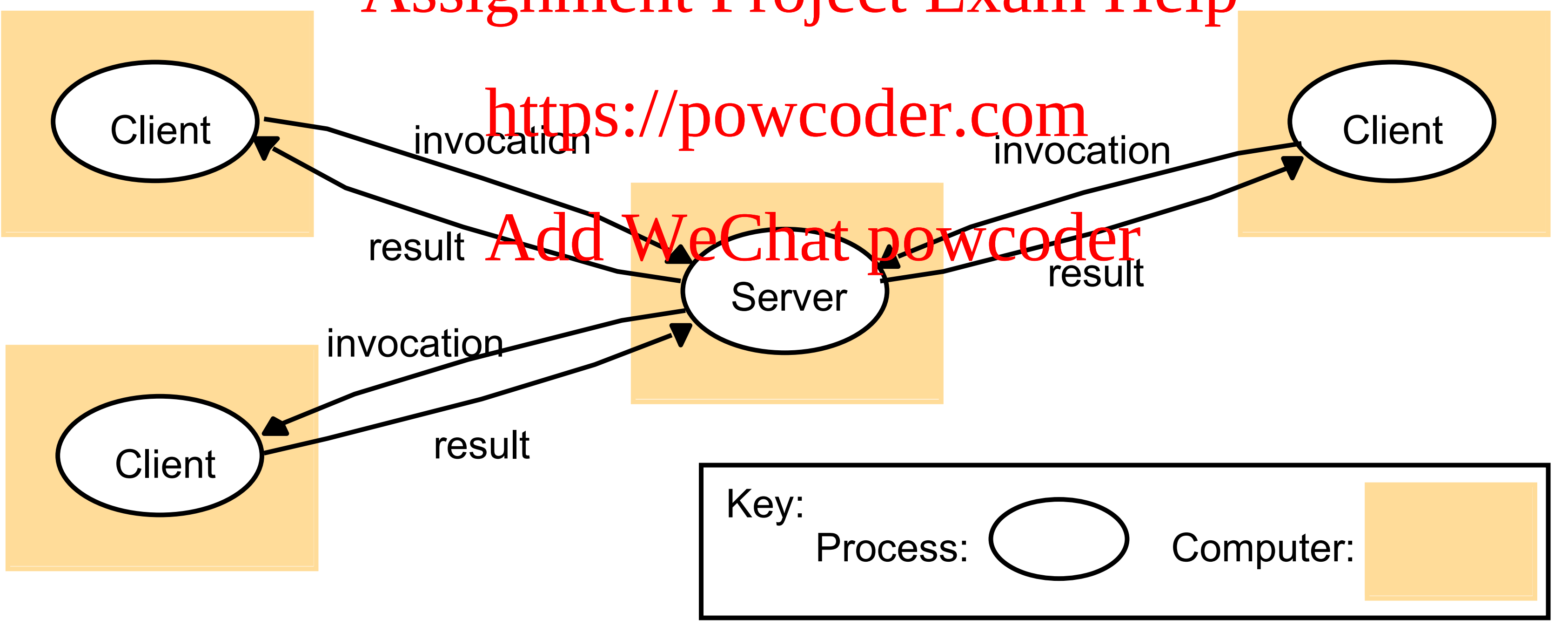
Architectural Models of Distribute Systems

- Architectural models are used to show the possible ways in which components of a distributed system may interact.
  - The placement of components across a network of computers, providing useful patterns for the distribution of data and workload.
  - The inter-relationships between the components, outlining their functional roles and the patterns of communication between them.

- In distributed systems, the two main types of architectural models are:
  - Client/Server model
  - Peer-to-Peer model

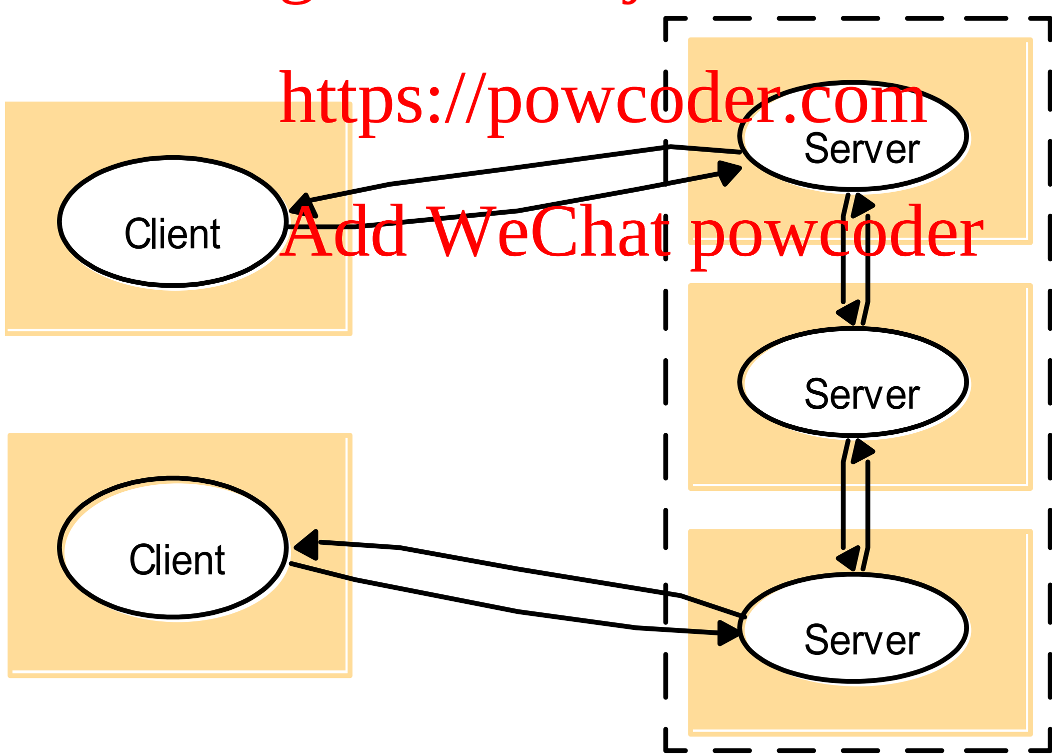
Client/Server Model – A Simple Case

- Clients invoke single servers which satisfy their requests individually.



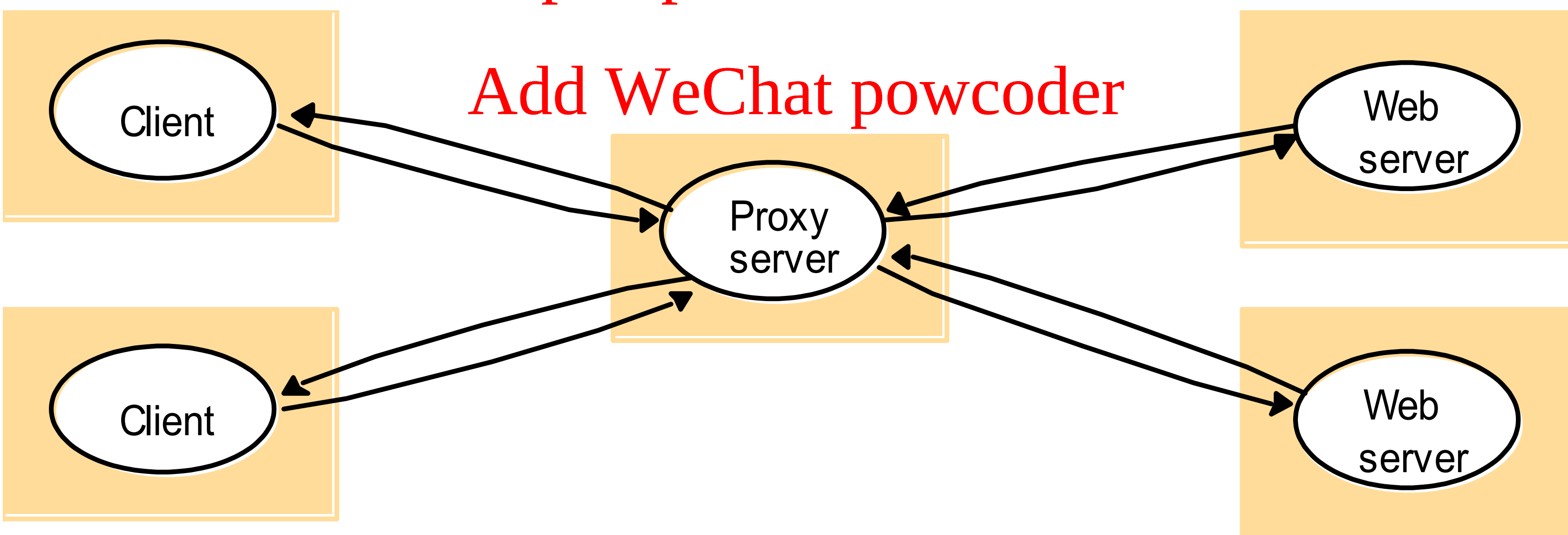
Client/Server Model – A Variation

- Servers on distinct hosts interact as appropriate to provide service.



Client/Server Model – Using Proxy and Caches

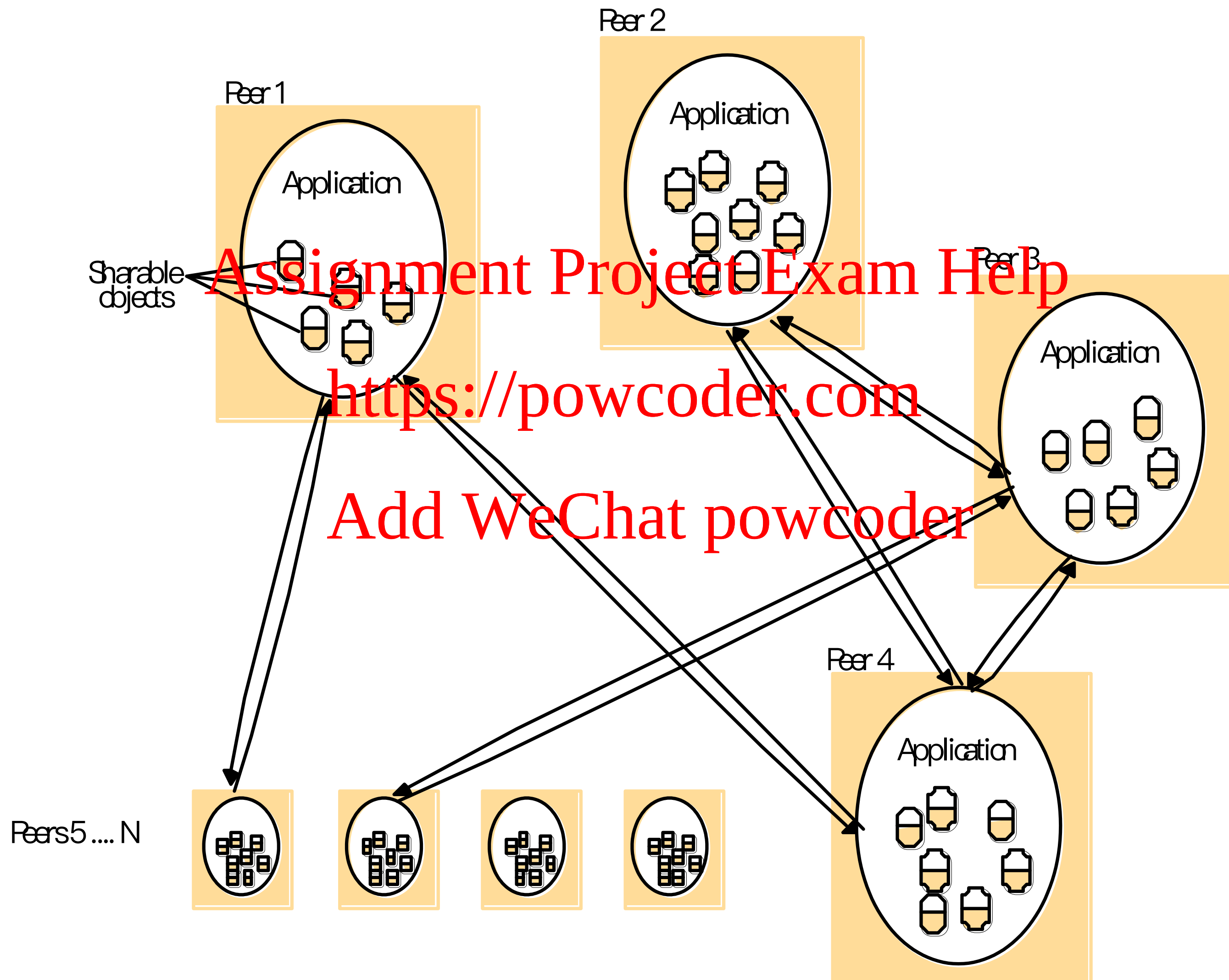
- A cache is used to avoid performance bottlenecks and reduce network traffic.
- Data resources are moved closer to the clients.
- A proxy server may host a cache.



Peer-to-Peer Model

- All processes play similar roles and result is achieved by cooperation.
- Pattern of communication will depend on the application requirements.
- The aim is to exploit the resources in a large number of peers for the fulfilment of a given task.
- ***Napster*** was a file sharing service that paved the way for decentralized P2P file-sharing programs.
- ***BitTorrent*** is a peer-to-peer file sharing communications protocol.

Peer-to-Peer Model



Fundamental Models of Distributed Systems

- By fundamental models, we wish to capture the following aspects of distributed systems:
  - Interaction
  - Failure
  - Security

Interaction Models

- Processes interact via message passing in complex ways, resulting in communication and coordination between processes.

- Two significant factors affecting interacting processes in a distributed system are:

  - Communication performance
  - Timing events

Interaction Models

- Communication Performance
  - Latency refers to the delay to a message's transmission.
  - Latency includes:
    - Transmission time
    - Delay in accessing the network
    - Operating system processing time
  - Bandwidth is the total amount of data that can be transmitted at a given time.
  - Jitter is the variation in time taken to deliver a series of messages.

Interaction Models

- Timing Events
  - Synchronous Systems

    - Time to execute each step of a process has known lower and upper bounds.

    - Each message transmitted over a channel is received within a known bounded time.

    - Each process has a local clock whose drift rate from real time has a known bound.
  - Defining these bounds may be difficult, but the benefits are many e.g. timeouts and failure models can be used effectively.

Interaction Models

- Timing Events
  - Asynchronous Systems place no bounds on:
    - Process execution speeds
    - Message transmission delays
    - Clock drift rates
  - Internet is the best example of such a system.
  - It is impossible to synchronise clocks in asynchronous systems.

Interaction Models

- Timing Events
  - In many cases, we are interested in knowing the ***order*** rather than the exact time of events.

    - Event A occurs before Event B
    - Event A occurs after Event B

    - Event A occurs concurrently with Event B

  - Example
    - X sends *m1* before Y receives *m1*
    - Y sends *m2* before X receives *m2*
    - Y receives *m1* before sending *m2*
    - Based on logical time, the emails will be displayed in a proper order, even if they arrived out of order.

Failure Models

- Failure model defines the ways in which failures may occur in order to provide an understanding of the effects of failures.
  - Processes may fail/crash.
  - Communication channels may fail.
- Failures types
  - Omission Failure
    - A component fails to perform actions that it is supposed to do.
    - Processes crash.
    - A communication channel does not transport message between buffers.

Failure Models

- Failures types
  - Arbitrary Failure
    - Used to describe the worst possible failure semantics.
    - A process sets wrong values in its data items or returns wrong values in response to an invocation.
    - Examples include corrupted data, duplicate responses.
    - Possible solutions include using sequence numbers, checksums.

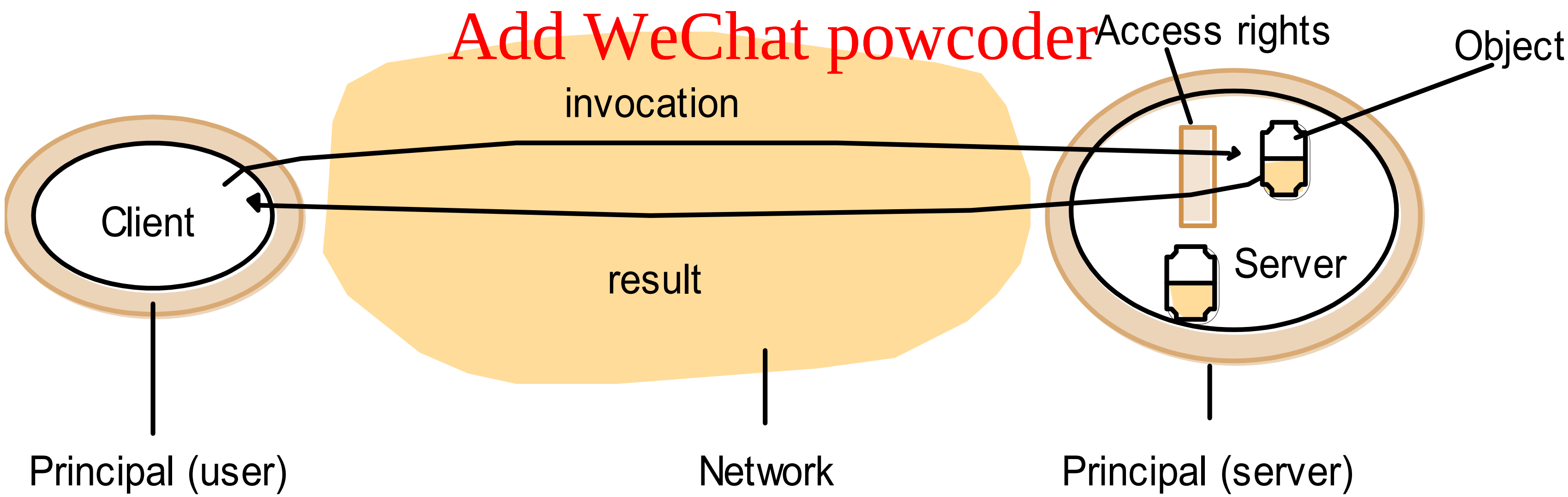
Failure Models

- Failures types
  - Timing Failure
    - Applicable to synchronous distributed systems
    - Relevant to multi-media systems
    - For example, a process's local clock exceeds the bounds on its rate of drift from real time.
- Masking Failures
  - It is possible to construct reliable services from components that exhibit failures.
  - Failure can be masked – requires low level retransmission/error checking.
  - Omission failure can be hidden by using a protocol that retransmits messages corrupted or lost.

Security Models

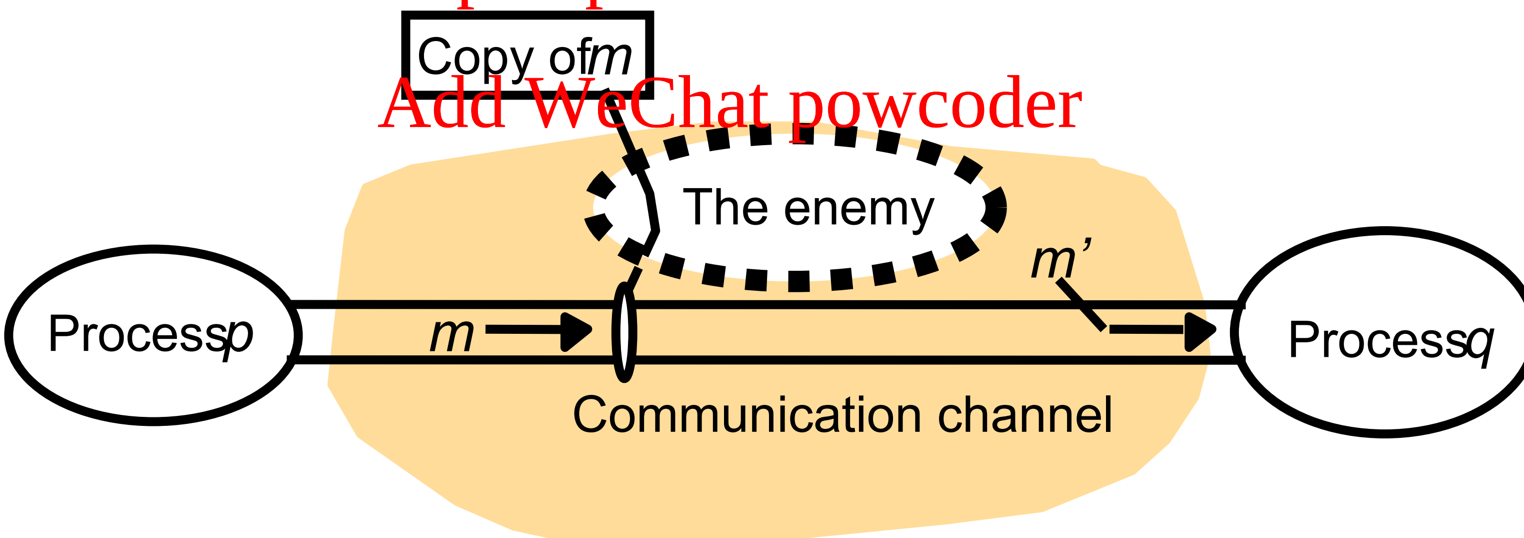
- The security of a distributed system can be achieved by securing the processes and the communication channels for their interactions.

- Access rights specifies who is allowed to perform operations on an object /resource.



Security Models

- Messages are exposed to attack because the network and the communication service that they use is *open.*



Security Models

- Secure channel has properties:
  - Each process knows the identity of the principal for whom the other process is executing.
  - Ensures the privacy and integrity of messages sent across it.
  - Each message includes timestamp to prevent replaying of messages.
- Defence mechanisms include:
  - Cryptography/encryption
  - Authentication
  - Digital Signature
  - Access control
  - Firewalls and Intrusion detection systems

Summary

- Resource sharing is the main motivating factor to construct distributed systems
- Distributed systems are everywhere, such as systems providing services over the Internet.
- The construction of distributed systems involve at least 7 challenges from heterogeneity to transparency.
- Client/Server and Peer-to-Peer models are popular architectural models of distributed systems.
- Interaction, Failure and Security comprise the fundamental models of distributed systems.